Team notebook

Mottakin Chowdhury

November 6, 2018

Contents

1 DP

1.1 Convex Hull Line Container

```
bool Q;

struct Line {
	mutable ll k, m, p; // slope, y-intercept, last optimal x
	bool operator<(const Line& o) const {
		return Q ? p < o.p : k < o.k;
	}
};

struct LineContainer : multiset<Line> {
	const ll inf = LLONG_MAX;
	ll div(ll a, ll b) { // floored division
		if (b < 0) a *= -1, b *= -1;
		if (a >= 0) return a / b;
		return -((-a + b - 1) / b);
	}

	// updates x->p, determines if y is unneeded
	bool isect(iterator x, iterator y) {
		if (y == end()) { x->p = inf; return 0; }
		if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
		else x->p = div(y->m - x->m, x->k - y->k);
		return x->p >= y->p;
	}

	void add(ll k, ll m) {
		auto z = insert({k, m, 0}), y = z++, x = y;
		while (isect(y, z)) z = erase(z);
		if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
		while ((y = x) != begin() && (--x)->p >= y->p) isect(x,
		    erase(y));
	}

	ll query(ll x) { // gives max value
		assert(!empty());
		Q = 1; auto l = *lower_bound({0, 0, x}); Q = 0;
		return l.k * x + l.m;
	}
};

// paths - vector of LineContainers
// a, b - LineContainers
// We want to take the pair-wise sum of the two line LineContainers
// and only keep the relevant ones. The sum is Minkowski Sum.

void convexsum(auto &a, auto &b)
{
	auto it1 = a.begin(), it2 = b.begin();
	while (it1 != a.end() && it2 != b.end())
	{
```

```
                universe.add((it1->k) + (it2->k), (it1->m) + (it2->m));
                if ((it1->p) < (it2->p)) it1++;
                else it2++;
        }
}

// We are merging all the LineContainers in paths.

void mergeall(int l, int r, auto &paths)
{
        if (l == r) return;
        int mid = (l + r) / 2;

        mergeall(l, mid, paths);
        mergeall(mid + 1, r, paths);

        convexsum(paths[l], paths[mid + 1]);

        for (auto it : paths[mid + 1]) paths[l].add(it.k, it.m);
}
```

1.2 Convex Hull Trick

```
struct cht
{
        vector<pii> hull;
        vector<int> id;

        int cur=0;

        cht()
        {
                hull.clear();
                id.clear();
        }

        // Might need double here

        bool useless(const pii left, const pii middle, const pii right)
        {
                return
                1LL*(middle.second-left.second)*(middle.first-right.first)
                >=1LL*(right.second-middle.second)*(left.first-middle.first);
        }

        // Inserting line a*x+b with index idx
        // Before inserting one by one, all the lines are sorted by slope

        void insert(int idx, int a, int b)
        {
                if(hull.empty())
                {
                        hull.pb(MP(a, b));
                        id.pb(idx);
                }
                else
                {
                        if(hull.back().first==a)
                        {
                                if(hull.back().second>=b)
                                {
                                        return;
                                }
                                else
                                {
                                        hull.pop_back();
                                        id.pop_back();
                                }
                        }
                        while(hull.size()>=2 &&
                            useless(hull[hull.size()-2], hull.back(), MP(a,
                            b)))
                        {
                                hull.pop_back();
                                id.pop_back();
                        }
                        hull.pb(MP(a,b));
                        id.pb(idx);
                }
        }

        // returns maximum value and the index of the line
        // Pointer approach: the queries are sorted non-decreasing
        // Otherwise, we will need binary search

        pair<ll,int> query(int x)
        {
                ll ret=-INF;
```

```
                int idx=-1;
                for(int i=cur ; i < hull.size() ; i++)
                {
                        ll tmp=1LL*hull[i].first*x + hull[i].second;

                        if(tmp>ret)
                        {
                                ret=tmp;
                                cur=i;
                                idx=id[i];
                        }
                        else
                        {
                                break;
                        }
                }
                return {ret,idx};
        }
};


// Slope decreasing, query minimum - Query point increasing.
// Slope increasing, query maximum - Query point increasing.
// Slope decreasing, query maximum - Query point decreasing.
// Slope increasing, query minimum - Query point decreasing.
```

1.3 Digit DP Sample 2

```
// For each case, output the case number and the number of integers in
    the range [A, B] which are
// divisible by K and the sum of its digits
// is also divisible by K.
int k, cases = 1;
ll dp[11][2][83][83];
int visited[11][2][83][83], flag;
string toString(int x)
{
        string temp = "";
        if (x == 0) return "0";
        while (x > 0)
        {
                int r = x % 10;
                temp = char(r + '0') + temp;
                x /= 10;
        }
        return temp;
}
ll calc(int idx, bool low, int modVal, int sumMod, string s)
{
        if (idx == s.size()) return (!modVal && !sumMod);
        if (visited[idx][low][modVal][sumMod] == flag)
                return dp[idx][low][modVal][sumMod];
        visited[idx][low][modVal][sumMod] = flag;
        int digit = low ? 9 : (s[idx] - '0');
        ll ret = 0;
        for (int i = 0; i <= digit; i++)
        {
                ret += calc(idx + 1, low || i < s[idx] - '0', (modVal * 10
                    + i) % k, (sumMod + i) % k, s);
        }
        return dp[idx][low][modVal][sumMod] = ret;
}
int main()
{
        int test;
        int a, b;
        cin >> test;
        while (test--)
        {
                cin >> a >> b >> k;
                if (k > 90)
                {
                        cout << "Case " << cases++ << ": 0" << endl;
                        continue;
                }
                string A = toString(a - 1);
                string B = toString(b);
                flag++;
                ll x = calc(0, 0, 0, 0, A);
                flag++;
                ll y = calc(0, 0, 0, 0, B);
                cout << "Case " << cases++ << ": " << y - x << endl;
        }
        return 0;
}
```

1.4 Digit DP Sample

```
// Calculate how many numbers in the range from A to B that have digit d
    in only the even positions and
// no digit occurs in the even position and the number is divisible by m.


string A, B; int m, d;
ll dp[2002][2002][2][2];

ll calc(int idx, int Mod, bool s, bool b)
{
        if(idx==B.size()) return Mod==0;

        if(dp[idx][Mod][s][b]!=-1)
                return dp[idx][Mod][s][b];

        ll ret=0;

        int low=s ? 0 : A[idx]-'0';
        int high=b ? 9 : B[idx]-'0';

        for(int i=low; i<=high; i++)
        {
                if(idx%2 && i!=d) continue;
                if(idx%2==0 && i==d) continue;

                ret=(ret+calc(idx+1, (Mod*10+i)%m, s || i>low, b ||
                    i<high))%mod;

                // if(ret>=mod) ret-=mod;
        }

        return dp[idx][Mod][s][b]=ret;
}

int main()
{
        // ios_base::sync_with_stdio(0);
        // cin.tie(NULL); cout.tie(NULL);
        // freopen("in.txt","r",stdin);

        cin>>m>>d>>A>>B;

        ms(dp,-1);

        prnt(calc(0,0,0,0));


        return 0;
}
```

1.5 Divide and Conquer DP

```
// http://codeforces.com/blog/entry/8219
// Divide and conquer optimization:
// Original Recurrence
//   dp[i][j] = min(dp[i-1][k] + C[k][j]) for k < j
// Sufficient condition:
//   A[i][j] <= A[i][j+1]
//   where A[i][j] = smallest k that gives optimal answer
// How to use:
//   // compute i-th row of dp from L to R. optL <= A[i][L] <= A[i][R] <=
    optR
//   compute(i, L, R, optL, optR)
//       1. special case L == R
//       2. let M = (L + R) / 2. Calculate dp[i][M] and opt[i][M] using
    O(optR - optL + 1)
//       3. compute(i, L, M-1, optL, opt[i][M])
//       4. compute(i, M+1, R, opt[i][M], optR)

// Example: http://codeforces.com/contest/321/problem/E
#include "../template.h"

const int MN = 4011;
const int inf = 1000111000;
int n, k;
ll cost[MN][MN], dp[811][MN];

inline ll getCost(int i, int j) {
   return cost[j][j] - cost[j][i-1] - cost[i-1][j] + cost[i-1][i-1];
}

void compute(int i, int L, int R, int optL, int optR) {
   if (L > R) return ;

   int mid = (L + R) >> 1, savek = optL;
   dp[i][mid] = inf;
```

```
    FOR(k,optL,min(mid-1, optR)+1) {
        ll cur = dp[i-1][k] + getCost(k+1, mid);
        if (cur < dp[i][mid]) {
            dp[i][mid] = cur;
            savek = k;
        }
    }
    compute(i, L, mid-1, optL, savek);
    compute(i, mid+1, R, savek, optR);
}

void solve() {
    cin >> n >> k;
    FOR(i,1,n+1) FOR(j,1,n+1) {
        cin >> cost[i][j];
        cost[i][j] = cost[i-1][j] + cost[i][j-1] - cost[i-1][j-1] +
            cost[i][j];
    }
    dp[0][0] = 0;
    FOR(i,1,n+1) dp[1][i] = inf;

    FOR(i,2,k+1) {
        compute(i, 1, n, 1, n);
    }
    cout << dp[k][n] / 2 << endl;
}
```

1.6 Dynamic Convex Hull Trick

```
// source:
    https://github.com/niklasb/contest-algos/blob/master/convex_hull/dynamic.cpp
// Used in problem CS Squared Ends
// Problem: A is an array of n integers. The cost of subarray A[l...r] is
    (A[l]-A[r])^2. Partition
// the array into K subarrays having a minimum total cost
// In case of initializing 'ans', check if 1e18 is enough. Might need
    LLONG_MAX

const ll is_query = -(1LL<<62);
struct Line {
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> { // will maintain upper hull
    for maximum
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;

        // **** May need long double typecasting here
        return (long double)(x->b - y->b)*(z->m - y->m) >= (long
            double)(y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};

int n, k;
ll a[10004];

int main()
{
    cin>>n>>k;
    FOR(i,1,n+1) cin>>a[i];
    vector<ll> dp(n+1,1e18);
    dp[0]=0;
    FOR(i,0,k)
```

```
    {
        HullDynamic hd;
        vector<ll> curr(n+1,1e18);

        FOR(j,1,n+1)
        {
            ll m=2*a[j];
            ll c=-a[j]*a[j]-dp[j-1];
            hd.insert_line(m,c);
            ll now=-hd.eval(a[j])+a[j]*a[j];
            curr[j]=now;
        }
        dp=curr;
    }
    prnt(dp[n]);

    return 0;
}
```

1.7 Edit Distance Recursive

```
int dp[34][34];
string a, b;

int editDistance(int i, int j)
{
        if (dp[i][j]!=-1)
                return dp[i][j];
        if (i==0)
                return dp[i][j]=j;
        if (j==0)
                return dp[i][j]=i;

        int cost;
        if (a[i-1]==b[j-1])
                cost=0;
        else
                cost=1;
        return
            dp[i][j]=min(editDistance(i-1,j)+1,min(editDistance(i,j-1)+1,
                                    editDistance(i-1,j-1)+cost));
}
```

```
int main()
{
        ms(dp,-1);
        cin>>a>>b;
        prnt(editDistance(a.size(),b.size()));
    return 0;
}
```

1.8 IOI Aliens by koosaga

```
#include <bits/stdc++.h>
using namespace std;
typedef long long lint;
typedef pair<lint, lint> pi;
const int MAXN = 100005;

vector<pi> v;
pi dp[MAXN];

struct point{
        lint first;
        lint second;
        int cnt;
};

struct cht{
        vector<point> v;
        void clear(){ v.clear(); }
        long double cross(point a, point b){
                return ((long double)(b.second - a.second) / (b.first -
                    a.first));
        }
        void add_line(int x, lint y, int z){
                while(v.size() >= 2 && cross(v[v.size()-2], v.back()) >
                    cross(v.back(), (point){x, y, z})){
                        v.pop_back();
                }
                v.push_back({x, y, z});
        }
        pi query(int x){
                int s = 0, e = v.size()-1;
                auto f = [&](int p){
                return v[p].first * x + v[p].second;
```

```
        };
        while(s != e){
            int m = (s+e)/2;
            if(f(m) <= f(m+1)) e = m;
            else s = m+1;
        }
        return pi(v[s].first * x + v[s].second, v[s].cnt);
    }
}cht;

pi trial(lint l){
    cht.clear();
    for(int i=1; i<=v.size(); i++){
        cht.add_line(2 * 2 * v[i-1].first, dp[i-1].first +
            2ll * v[i-1].first * v[i-1].first, dp[i-1].second);
        dp[i] = cht.query(-v[i-1].second);
        dp[i].first += 2ll * v[i-1].second * v[i-1].second + l; //
            l is penalty
        dp[i].second++;
        if(i != v.size()){
            lint c = max(0ll, v[i-1].second - v[i].first);
            dp[i].first -= 2 * c * c;
        }
    }
    return dp[v.size()];
}

long long take_photos(int n, int m, int k, std::vector<int> r,
    std::vector<int> c) {
    vector<pi> w;
    for(int i=0; i<n; i++){
        if(r[i] > c[i]) swap(r[i], c[i]);
        w.push_back({r[i]-1, c[i]});
    }
    sort(w.begin(), w.end(), [&](const pi &a, const pi &b){
        return pi(a.first, -a.second) < pi(b.first, -b.second);
    });
    for(auto &i : w){
        if(v.empty() || v.back().second < i.second){
            v.push_back(i);
        }
    }
    lint s = 0, e = 2e12;
    while(s != e){
        lint m = (s+e)/2;
        // See how many groups are made with penalty 2*m+1
        if(trial(2 * m + 1).second <= k) e = m;
        else s = m+1;
    }
    return trial(s * 2).first / 2 - s * k;
}
```

1.9 In-out DP

```
// The problem was to find the distance of the farthest node
// from each node. So we try to find such distance considering
// each node as a root.
const int N=10004;
int n, f[N], g[N], ans[N];
vpii graph[N]; vi prefix[N], suffix[N];

void clear()
{
    FOR(i,1,n+1) graph[i].clear(), prefix[i].clear(),
        suffix[i].clear();
    ms(f,0); ms(g,0); ms(ans,0);
}
void goforgun(int u, int p=-1, int d=0)
{
    if(p==-1) ans[u]=f[u];

    FOR(j,0,graph[u].size())
    {
        int v=graph[u][j].first;
        int w=graph[u][j].second;

        if(v==p) continue;
        // considering that jth child is deleted
        g[u]=max(prefix[u][j],suffix[u][j]);
        // if we are not in root, we also consider the case
        // where parent of u becomes child of u when u is the root
        // d is the cost between the edge (p--u)
        if(p!=-1) g[u]=max(g[p]+d,g[u]);
        // updating answer for v, here we consider the case when v
            is root
        ans[v]=max(f[v],g[u]+w);
        goforgun(v,u,w);
    }
```

```
}
// Precalculate prefix-max and suffix-max values
// max(prefix[u][j],suffix[u][j]) contains the maximum
// value of f[u] if jth child was deleted
void goforfun(int u, int p=-1)
{
	FOR(j,0,graph[u].size())
	{
		int v=graph[u][j].first;
		int w=graph[u][j].second;

		if(v==p) continue;
		goforfun(v,u);

		f[u]=max(f[u],f[v]+w);
	}
	int pref=0, suff=0;
	FOR(j,0,graph[u].size())
	{
		int v=graph[u][j].first;
		int w=graph[u][j].second;
		// important, we want to keep same size but avoid parent
		if(v==p)
		{
			prefix[u].pb(0);
			continue;
		}
		prefix[u].pb(pref);
		pref=max(pref,f[v]+w);
	}
	FORr(j,graph[u].size()-1,0)
	{
		int v=graph[u][j].first;
		int w=graph[u][j].second;
		if(v==p)
		{
			suffix[u].pb(0);
			continue;
		}
		suffix[u].pb(suff);
		suff=max(suff,f[v]+w);
	}
	// Reversing is important
	REVERSE(suffix[u]);
}
```

```
int main()
{
	while(scanf("%d", &n)!=EOF)
	{
		int u, w;
		FOR(i,2,n+1)
		{
			scanf("%d%d", &u, &w);

			graph[u].pb(MP(i,w));
			graph[i].pb(MP(u,w));
		}
		goforfun(1);
		goforgun(1);
		FOR(i,1,n+1) prnt(ans[i]);
		clear();
	}
	return 0;
}
```

1.10 Knuth Optimization

```
/*This trick works only for optimization DP over substrings for which
    optimal
middle point depends monotonously on the end points. Let mid[L,R] be the
    first
middle point for (L,R) substring which gives optimal result. It can be
    proven
that mid[L,R-1] <= mid[L,R] <= mid[L+1,R] - this means monotonicity of
    mid by
L and R. Applying this optimization reduces time complexity from O(k^3)
    to O(k^2)
 because with fixed s (substring length) we have m_right(L) = mid[L+1][R]
     = m_left(L+1).
 That's why nested L and M loops require not more than 2k iterations
     overall.*/

for (int s = 0; s <= k; s++) //s - length(size) of substring
	for (int L = 0; L + s <= k; L++) { //L - left point
		int R = L + s; //R - right point
		if (s < 2) {
			res[L][R] = 0; //DP base - nothing to break
```

```
                mid[L][R] = l; //mid is equal to left border
                continue;
            }
            int mleft = mid[L][R - 1]; //Knuth's trick: getting bounds
                on M
            int mright = mid[L + 1][R];
            res[L][R] = 1000000000000000000LL;
            for (int M = mleft; M <= mright; M++) { //iterating for M
                in the bounds only
                ll tres = res[L][M] + res[M][R] + (x[R] - x[L]);
                if (res[L][R] > tres) { //relax current solution
                    res[L][R] = tres;
                    mid[L][R] = M;
                }
            }
        }
ll answer = res[0][k];
```

1.11 LCS

```
string a, b;
int dp[100][100];
string l;
void printLcs(int i, int j)
{
        if (a[i] == '\0' || b[j] == '\0')
        {
                cout << l << endl;
                return;
        }
        if (a[i] == b[j])
        {
                l += a[i];
                printLcs(i + 1, j + 1);
        }
        else
        {
                if (dp[i + 1][j] > dp[i][j + 1])
                        printLcs(i + 1, j);
                else
                        printLcs(i, j + 1);
        }
}
```

```
void printAll(int i, int j)
{
        if (a[i] == '\0' || b[j] == '\0')
        {
                prnt(l);
                return;
        }
        if (a[i] == b[j])
        {
                l += a[i];
                printAll(i + 1, j + 1);
                l.erase(l.end() - 1);
        }
        else
        {
                if (dp[i + 1][j] > dp[i][j + 1])
                        printAll(i + 1, j);
                else if (dp[i + 1][j] < dp[i][j + 1])
                        printAll(i, j + 1);
                else
                {
                        printAll(i + 1, j);
                        printAll(i, j + 1);
                }
        }
}
int lcslen (int i, int j)
{
        if (a[i] == '\0' || b[j] == '\0')
                return 0;
        if (dp[i][j] != -1)
                return dp[i][j];
        int ans = 0;
        if (a[i] == b[j])
        {
                ans = 1 + lcslen(i + 1, j + 1);
        }
        else
        {
                int x = lcslen(i, j + 1);
                int y = lcslen(i + 1, j);
                ans = max(x, y);
        }
        return dp[i][j] = ans;
}
```

```
int main()
{
	cin >> a >> b;
	ms(dp, -1);
	cout << lcslen(0, 0) << endl;
	printLcs(0, 0);
	l.clear();
	printAll(0, 0);
	return 0;
}
```

1.12 LIS nlogk

```
vector<int> d;
int ans, n;

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        int x;
        scanf("%d", &x);
        vector<int>::iterator it = lower_bound(d.begin(), d.end(), x);
        if (it == d.end()) d.push_back(x);
        else *it = x;
    }
    printf("LIS = %d", d.size());
    return 0;
}
```

1.13 Matrix Expo Class

```
struct Matrix
{
	ll mat[MAX][MAX];

	Matrix(){}

	// This initialization is important.
	// Input matrix should be initialized separately
	void init(int sz)
	{
		ms(mat,0);
		for(int i=0; i<sz; i++) mat[i][i]=1;
	}
} aux;

void matMult(Matrix &m, Matrix &m1, Matrix &m2, int sz)
{
	ms(m.mat,0);

	// This only works for square matrix

	FOR(i,0,sz)
	{
		FOR(j,0,sz)
		{
			FOR(k,0,sz)
			{
				m.mat[i][k]=(m.mat[i][k]+m1.mat
					[i][j]*m2.mat[j][k])%mod;
			}
		}
	}
}

Matrix expo(Matrix &M, int n, int sz)
{
	Matrix ret;
	ret.init(sz);

	if(n==0) return ret;
	if(n==1) return M;

	Matrix P=M;

	while(n!=0)
	{
		if(n&1)
		{
			aux=ret;
			matMult(ret,aux,P,sz);
		}

		n>>=1;
```

```
        aux=P; matMult(P,aux,aux,sz);
    }

    return ret;
}
```

1.14 Palindrome in a String

```
bool isPalindrome[100][100];
// Find the palindromes of a string in O(n^2)

int main()
{
    ios_base::sync_with_stdio(0);
    // freopen("in.txt","r",stdin);

    string s;

    cin>>s;

    int len=s.size();

    for(int i=0; i<len; i++)
        isPalindrome[i][i]=true;

    for(int k=1; k<len; k++)
    {
        for(int i=0; i+k<len; i++)
        {
            int j=i+k;

            isPalindrome[i][j]=(s[i]==s[j]) &&
            (isPalindrome[i+1][j-1] || i+1>=j-1);
        }
    }

    return 0;
}
```

2 Data Structures

2.1 2D BIT

```
// Call with size of the grid
// Example: fenwick_tree_2d<int> Tree(n+1,m+1) for n x m grid indexed
    from 1

template <class T>
struct fenwick_tree_2d
{
    vector<vector<T>> x;
    fenwick_tree_2d(int n, int m) : x(n, vector<T>(m)) { }
    void add(int k1, int k2, int a) { // x[k] += a
        for (; k1 < x.size(); k1 |= k1 + 1)
            for (int k = k2; k < x[k1].size(); k |= k + 1)
                x[k1][k] += a;
    }
    T sum(int k1, int k2) { // return x[0] + ... + x[k]
        T s = 0;
        for (; k1 >= 0; k1 = (k1 & (k1 + 1)) - 1)
            for (int k = k2; k >= 0; k = (k & (k + 1)) - 1) s
                += x[k1][k];
        return s;
    }
};
```

2.2 2D Segment Tree

```
// Given a grid a[][], we ask sum of a subrectangle and als
// update some value on a cell.
void build_y(int vx, int lx, int rx, int vy, int ly, int ry) {
   if (ly == ry) {
      if (lx == rx)
         t[vx][vy] = a[lx][ly];
      else
         t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
   } else {
      int my = (ly + ry) / 2;
      build_y(vx, lx, rx, vy*2, ly, my);
      build_y(vx, lx, rx, vy*2+1, my+1, ry);
      t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
```

```
    }
}
void build_x(int vx, int lx, int rx) {
    if (lx != rx) {
        int mx = (lx + rx) / 2;
        build_x(vx*2, lx, mx);
        build_x(vx*2+1, mx+1, rx);
    }
    build_y(vx, lx, rx, 1, 0, m-1);
}
int sum_y(int vx, int vy, int tly, int try_, int ly, int ry) {
    if (ly > ry)
        return 0;
    if (ly == tly && try_ == ry)
        return t[vx][vy];
    int tmy = (tly + try_) / 2;
    return sum_y(vx, vy*2, tly, tmy, ly, min(ry, tmy))
         + sum_y(vx, vy*2+1, tmy+1, try_, max(ly, tmy+1), ry);
}
int sum_x(int vx, int tlx, int trx, int lx, int rx, int ly, int ry) {
    if (lx > rx)
        return 0;
    if (lx == tlx && trx == rx)
        return sum_y(vx, 1, 0, m-1, ly, ry);
    int tmx = (tlx + trx) / 2;
    return sum_x(vx*2, tlx, tmx, lx, min(rx, tmx), ly, ry)
         + sum_x(vx*2+1, tmx+1, trx, max(lx, tmx+1), rx, ly, ry);
}
void update_y(int vx, int lx, int rx, int vy, int ly, int ry, int x, int
     y, int new_val) {
    if (ly == ry) {
        if (lx == rx)
            t[vx][vy] = new_val;
        else
            t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
    } else {
        int my = (ly + ry) / 2;
        if (y <= my)
            update_y(vx, lx, rx, vy*2, ly, my, x, y, new_val);
        else
            update_y(vx, lx, rx, vy*2+1, my+1, ry, x, y, new_val);
        t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
    }
}
void update_x(int vx, int lx, int rx, int x, int y, int new_val) {
    if (lx != rx) {
        int mx = (lx + rx) / 2;
        if (x <= mx)
            update_x(vx*2, lx, mx, x, y, new_val);
        else
            update_x(vx*2+1, mx+1, rx, x, y, new_val);
    }
    update_y(vx, lx, rx, 1, 0, m-1, x, y, new_val);
}
```

2.3 A DSU Problem

```
/* Problem: You are given a graph with edge-weights. Each node has some
    color.
Now you are also given some queries of the form (starting_node, weight).
    The
query means you can start from the starting_node and you can visit only
    those
edges which have wight <= weight. You need to print the color which will
    occur
the maximum time in your journey. If two color occurs the same, output
    the lower
indexed one.
Solution: idea is to use DSU small to large merging with binary-lifting.
*/
const int LOG = 17;
int n, m, a[MAX], parent[MAX], up[MAX][LOG], weight[MAX][LOG];
// best (count,color_number) pair on a component after adding an edge
pii best[MAX];
// (weight,color) pair which means you can get max-occurrence of 'color'
    with 'weight'
vpii ans[MAX];
// (color,cnt) pair for each component, stores all of them
set<pii> color[MAX];
vector<array<int,3>> edges;

void clear()
{
        // Initialize weights and 2^j parent of each node.
        // Initially, each node is 2^j-th parent of itself
        FOR(i,1,n+1)
        {
                FOR(j,0,LOG)
```

```
                weight[i][j] = 1e9, up[i][j]=i;
        }
        edges.clear();
        FOR(i,1,n+1) color[i].clear(), ans[i].clear();
}

int findParent(int r)
{
        if(parent[r]==r) return r;
        return parent[r]=findParent(parent[r]);
}

void merge(int u, int v)
{
        // merge u into v
        for(auto it: color[u])
        {
                auto curr = color[v].lower_bound({it.first,-1});
                int cnt = it.second;

                if(curr!=end(color[v]) && curr->first==it.first)
                {
                        cnt+=curr->second;
                        color[v].erase(curr);
                }

                color[v].insert({it.first,cnt});
                // best (cnt,color) pair for component v, -it.first for
                    ensuring
                // that we get the smallest index while max-ing
                best[v] = max(best[v],{cnt,-it.first});
        }
}

void solve()
{
        FOR(i,1,n+1) parent[i] = i;
        for(auto it: edges)
        {
                auto [w,u,v] = it;

                u = findParent(u);
                v = findParent(v);

                if(color[u].size()>color[v].size())
                        swap(u,v);
                if(u!=v)
                {
                        merge(u,v);
                        parent[u] = v;
                        // after merging (u,v), we store best answer
                        // for the component v in ans[v]
                        ans[v].pb({w,-best[v].second});
                        up[u][0] = v;
                        weight[u][0] = w;
                }
                // note that if u==v, that edge and its weight won't
                    matter as
                // we have already added the smaller edges and the nodes
                    are already
                // connected
        }
        for(int i=1; i<LOG; i++)
        {
                for(int j=1; j<=n; j++)
                {
                        // 2^ith component of j in dsu
                        up[j][i] = up[up[j][i-1]][i-1];
                        // weight that we need to consider
                        weight[j][i] = weight[up[j][i-1]][i-1];
                }
        }
}

int main()
{
    int test, cases = 1;

    scanf("%d", &test);
    while(test--)
    {
        scanf("%d%d", &n, &m);
        clear();
        FOR(i,1,n+1)
        {
                scanf("%d", &a[i]);
                // initializing each node as a single component
                color[i].insert({a[i],1});
                best[i] = {1,-a[i]};
                ans[i].pb({0,a[i]});
```

```
        }

        int u, v, w;

        FOR(i,1,m+1)
        {
                scanf("%d%d%d", &u, &v, &w);
                edges.pb({w,u,v});
        }
        sort(begin(edges),end(edges));
        solve();

        int last = 0, q;
        scanf("%d", &q);
        printf("Case #%d:\n", cases++);
        while(q--)
        {
                scanf("%d%d", &u, &w);
                // the problem used this xor-ing to make the solution
                    online
                u^=last, w^=last;
                // u will be the component we can visit
                for(int i=LOG-1; i>=0; i--)
                {
                        if(weight[u][i]<=w)
                                u=up[u][i];
                }
                int idx = lower_bound(ALL(ans[u]),pii{w,inf}) -
                    begin(ans[u]) - 1;
                last = ans[u][idx].second;
                printf("%d\n", last);
        }
        clear();
    }
    return 0;
}
```

2.4 BIT Range Update Range Query

```
class BITrangeOperations
{
public:
        ll Tree[MAX+7][2];

        void update(int idx, ll x, bool t)
        {
                while(idx<=MAX)
                {
                        Tree[idx][t]+=x;
                        idx+=(idx&-idx);
                }
        }

        ll query(int idx, bool t)
        {
                ll sum=0;
                while(idx>0)
                {
                        sum+=Tree[idx][t];
                        idx-=(idx&-idx);
                }
                return sum;
        }

        // Returns sum from [0,x]
        ll sum(int x)
        {
                return (query(x,0)*x)-query(x,1);
        }

        void updateRange(int l, int r, ll val)
        {
                update(l,val,0);
                update(r+1,-val,0);
                update(l,val*(l-1),1);
                update(r+1,-val*r,1);
        }

        ll rangeSum(int l, int r)
        {
                return sum(r)-sum(l-1);
        }
};
```

2.5 Best Partial Sum in a Range

```
struct Node
{
	ll bestSum, bestPrefix, bestSuffix, segSum;
	Node()
	{
		bestSum=bestPrefix=bestSuffix=segSum=-INF;
	}
	void merge(Node &l, Node &r)
	{
		segSum=l.segSum+r.segSum;
		bestPrefix=max(l.bestPrefix,r.bestPrefix+l.segSum);
		bestSuffix=max(r.bestSuffix,r.segSum+l.bestSuffix);
		bestSum=max(max(l.bestSum,r.bestSum),l.bestSuffix+r.bestPrefix);
	}
}tree[150005];

void init(int node, int start, int end)
{
	if(start==end)
	{
		tree[node].bestSum=tree[node].segSum=a[start];
		tree[node].bestSuffix=tree[node].bestPrefix=a[start];
		return;
	}
	int left=node<<1;
	int right=left+1;
	int mid=(start+end)>>1;
	init(left,start,mid);
	init(right,mid+1,end);
	tree[node].merge(tree[left],tree[right]);
}
void update(int node, int start, int end, int i, int val)
{
	if(i<start || i>end)
		return;
	if(start>=i && end<=i)
	{
		tree[node].bestSum=tree[node].segSum=val;
		tree[node].bestSuffix=tree[node].bestPrefix=val;
		a[start]=val;
		return;
	}
	int left=node<<1;
	int right=left+1;
	int mid=(start+end)>>1;
	update(left,start,mid,i,val);
	update(right,mid+1,end,i,val);
	tree[node].merge(tree[left],tree[right]);
}
Node query(int node, int start, int end, int i, int j)
{
	if(i>end || j<start)
		return Node();
	if(start>=i && end<=j)
	{
		return tree[node];
	}
	int left=node<<1;
	int right=left+1;
	int mid=(start+end)>>1;
	Node l=query(left,start,mid,i,j);
	Node r=query(right,mid+1,end,i,j);
	Node n;
	n.merge(l,r);
	return n;
}
```

2.6 Binary Indexed Tree

```
ll Tree[MAX];
// This is equivalent to calculating lower_bound on prefix sums array
// LOGN = log(N)
int bit_search(int v)
{
	int sum = 0;
	int pos = 0;

	for(int i=LOGN; i>=0; i--)
	{
		if(pos + (1 << i) < N and sum + Tree[pos + (1 << i)] < v)
		{
			sum += Tree[pos + (1 << i)];
			pos += (1 << i);
		}
	}
	// +1 because 'pos' will have position of largest value less than
	    'v'
```

```
        return pos + 1;
}

void update(int idx, ll x)
{
        // Let, n is the number of elements and our queries are
        // of the form query(n)-query(l-1), i.e range queries
        // Then, we should never put N or MAX in place of n here.
        while(idx<=n)
        {
                Tree[idx]+=x;
                idx+=(idx&-idx);
        }
}

ll query(int idx)
{
        ll sum=0;
        while(idx>0)
        {
                sum+=Tree[idx];
                idx-=(idx&-idx);
        }
        return sum;
}

int main()
{
        // For point update range query:
        // Point update: update(x,val);
        // Range query (a,b): query(b)-query(a-1);

        // For range update point query:
        // Range update (a,b): update(a,v); update(b+1,-v);
        // Point query: query(x);

        // Let's just consider only one update: Add v to [a, b] while the
            rest elements of the array is 0.
        // Now, consider sum(0, x) for all possible x, again three
            situation can arise:
        // 1. 0    x < a : which results in 0
        // 2. a    x     b : we get v * (x - (a-1))
        // 3. b < x < n : we get v * (b - (a-1))
        // This suggests that, if we can find v*x for any index x, then we
            can get the sum(0, x) by subtracting T from it, where:
        // 1. 0    x < : Sum should be 0, thus, T = 0
        // 2. a    x     : Sum should be v*x-v*(a-1), thus, T = v*(a-1)
        // 3. b < x < n : Sum should be 0, thus, T = -v*b + v*(a-1)
        // As, we can see, knowing T solves our problem, we can use
            another BIT to store this additive amount from which we can
            get:
        // 0 for x < a, v*(a-1) for x in [a..b], -v*b+v(a-1) for x > b.

        // Now we have two BITs.
        // To add v in range [a, b]: Update(a, v), Update(b+1, -v) in the
            first BIT and Update(a, v*(a-1)) and Update(b+1, -v*b) on the
            second BIT.
        // To get sum in range [0, x]: you simply do Query_BIT1(x)*x -
            Query_BIT2(x);
        // Now you know how to find range sum for [a, b]. Just find sum(b)
            - sum(a-1) using the formula stated above.
    return 0;
}
```

2.7 Centroid Decomposition Sample

```
/* You are given a tree consisting of n vertices. A number is written on
    each vertex;
the number on vertex i is equal to a[i].
Let, g(x,y) is the gcd of the numbers written on the vertices belonging
    to the path from
x to y, inclusive. For i in 1 to 200000, count number of pairs (x,y)
    (1<=x<=y) such
that g(x,y) equals to i.
Note that 1<=x<=y does not really matter.
*/
vi graph[MAX];
int n, a[MAX], sub[MAX], total, cnt[MAX], cent, upto[MAX];
ll ans[MAX];
bool done[MAX];
set<int> take[MAX];

void dfs(int u, int p)
{
        sub[u] = 1;
        total++;

        for (auto v : graph[u])
```

```
        {
                if (v == p || done[v]) continue;
                dfs(v, u);
                sub[u] += sub[v];
        }
}

int getCentroid(int u, int p)
{
        // cout<<u<<" "<<sub[u]<<endl;
        for (auto v : graph[u])
        {
                if (!done[v] && v != p && sub[v] > total / 2)
                        return getCentroid(v, u);
        }

        return u;
}

void go(int u, int p, int val)
{
        ans[val]++;
        take[cent].insert(val);
        cnt[val]++;

        for (auto v : graph[u])
        {
                if (!done[v] && v != p)
                {
                        go(v, u, upto[v]);
                }
        }
}

void calc(int u, int p, int val)
{
        for (auto it : take[cent])
        {
                int g = gcd(val, it);
                ans[g] += cnt[it];
        }

        for (auto v : graph[u])
        {
                if (!done[v] && v != p)
                {
                        calc(v, u, upto[v]);
                }
        }
}

void clean(int u, int p, int val)
{
        cnt[val] = 0;

        for (auto v : graph[u])
        {
                if (!done[v] && v != p)
                {
                        clean(v, u, upto[v]);
                }
        }
}

void calcgcd(int u, int p, int val)
{
        upto[u] = val;

        for (auto v : graph[u])
        {
                if (!done[v] && v != p)
                {
                        calcgcd(v, u, gcd(val, a[v]));
                }
        }
}

void solve(int u)
{
        total = 0;
        dfs(u, -1);

        cent = getCentroid(u, -1);
        calcgcd(cent, -1, a[cent]);

        // debug("cent",cent);
        done[cent] = true;

        for (auto v : graph[cent])
        {
```

```
                if (done[v]) continue;
                // cout<<"from centroid "<<cent<<" going to node:
                    "<<v<<endl;
                calc(v, cent, upto[v]);
                go(v, cent, upto[v]);
        }

        for (auto v : graph[cent])
        {
                if (!done[v])
                        clean(v, cent, upto[v]);
        }

        for (auto v : graph[cent])
        {
                if (!done[v])
                        solve(v);
        }
}

int main()
{
        // ios_base::sync_with_stdio(0);
        // cin.tie(NULL); cout.tie(NULL);
        // freopen("in.txt","r",stdin);

        int test, cases = 1;

        scanf("%d", &n);
        FOR(i, 1, n + 1)
        {
                scanf("%d", &a[i]);
                ans[a[i]]++;
        }

        int u, v;

        FOR(i, 1, n)
        {
                scanf("%d%d", &u, &v);
                graph[u].pb(v);
                graph[v].pb(u);
        }

        solve(1);

        FOR(i, 1, MAX) if (ans[i]) printf("%d %lld\n", i, ans[i]);

        return 0;
}
```

2.8 Centroid Decomposition

```
int n, m, a, b, Table[MAX][20];
set<int> Graph[MAX];
int Level[MAX], nodeCnt, Subgraph[MAX], Parent[MAX], Ans[MAX];
void findLevel(int u)
{
        itrALL(Graph[u], it)
        {
                int v = *it;
                if (v != Table[u][0])
                {
                        Table[v][0] = u;
                        Level[v] = Level[u] + 1;
                        findLevel(v);
                }
        }
}
void Process()
{
        Level[0] = 0;
        ms(Table, -1);
        Table[0][0] = 0;
        findLevel(0);
// debug;
        for (int j = 1; 1 << j < n; j++)
        {
                for (int i = 0; i < n; i++)
                {
                        if (Table[i][j - 1] != -1)
                                Table[i][j] = Table[Table[i][j - 1]][j - 1];
                }
        }
// debug;
}
int findLCA(int p, int q)
{
        if (Level[p] < Level[q]) swap(p, q);
```

```
	int x = 1;
	while (true)
	{
		if ((1 << (x + 1)) > Level[p]) break;
		x++;
	}
	FORr(i, x, 0)
	{
		if (Level[p] - (1 << i) >= Level[q])
			p = Table[p][i];
	}
	if (p == q) return p;
	FORr(i, x, 0)
	{
		if (Table[p][i] != -1 && Table[p][i] != Table[q][i])
		{
			p = Table[p][i];
			q = Table[q][i];
		}
	}
	return Table[p][0];
}
int Dist(int a, int b)
{
	return Level[a] + Level[b] - 2 * Level[findLCA(a, b)];
}
void findSubgraph(int u, int parent)
{
	Subgraph[u] = 1;
	nodeCnt++;
	itrALL(Graph[u], it)
	{
		int v = *it;
		if (v == parent) continue;
		findSubgraph(v, u);
		Subgraph[u] += Subgraph[v];
	}
}
int findCentroid(int u, int p)
{
	itrALL(Graph[u], it)
	{
		int v = *it;
		if (v == p) continue;
		if (Subgraph[v] > nodeCnt / 2) return findCentroid(v, u);
	}
	return u;
}
void Decompose(int u, int p)
{
	nodeCnt = 0;
	findSubgraph(u, u);
	int Cent = findCentroid(u, u);
	if (p == -1) p = Cent;
	Parent[Cent] = p;
	itrALL(Graph[Cent], it)
	{
		int v = *it;
		Graph[v].erase(Cent);
		Decompose(v, Cent);
	}
	Graph[Cent].clear();
}
void update(int u)
{
	int x = u;
	while (true)
	{
		Ans[x] = min(Ans[x], Dist(x, u));
		if (x == Parent[x]) break;
		x = Parent[x];
	}
}
int query(int u)
{
	int x = u;
	int ret = INF;
	while (true)
	{
		ret = min(ret, Dist(u, x) + Ans[x]);
		if (x == Parent[x]) break;
		x = Parent[x];
	}
	return ret;
}
int main()
{
// ios_base::sync_with_stdio(0);
// cin.tie(NULL); cout.tie(NULL);
// freopen("in.txt","r",stdin);
```

```
// All the nodes are initially blue
// Then by updating, one node is colored red
// Upon query, return the closest red node of the given node
	scanf("%d%d", &n, &m);
	FOR(i, 0, n - 1)
	{
		scanf("%d%d", &a, &b);
		a--, b--;
		Graph[a].insert(b);
		Graph[b].insert(a);
	}
	Process();
// debug;
	Decompose(0, -1);
	FOR(i, 0, n) Ans[i] = INF;
	update(0);
	while (m--)
	{
		int t, x;
		scanf("%d%d", &t, &x);
		x--;
		if (t == 1) update(x);
		else printf("%d\n", query(x));
	}
	return 0;
}
```

2.9 Counting Inversions with BIT

```
ll tree[200005];
int n, a[200005], b[200005];

void update(int idx, ll x)
{
	while(idx<=n)
	{
		tree[idx]+=x;
		idx+=(idx&-idx);
	}
}

int query(int idx)
{
	ll sum=0;
	while(idx>0)
	{
		sum+=tree[idx];
		idx-=(idx&-idx);
	}
	return sum;
}



int main()
{
	// ios_base::sync_with_stdio(0);
	// cin.tie(NULL); cout.tie(NULL); // No 'endl'
	// freopen("in.txt","r",stdin);
	int test;
	// cin>>test;
	scanf("%d", &test);
	while(test--)
	{
		ms(tree,0);
		scanf("%d", &n);
		FOR(i,1,n+1)
		{
			scanf("%d", &a[i]);
			b[i]=a[i];
		}

		sort(b+1,b+n+1);

		// Compressing the array
		FOR(i,1,n+1)
		{
			int rank=int(lower_bound(b+1,b+1+n,a[i])-b-1);
			a[i]=rank+1;
		}
		// FOR(i,1,n+1) cout<<a[i]<<" "; cout<<endl;
		ll ans=0;
		FORr(i,n,1)
		{
			ans+=query(a[i]-1);
			update(a[i],1);
		}
		// prnt(ans);
```

```
		printf("%lld\n",ans);
	}


	return 0;
}
```

2.10 DSU on Tree Sample

```
/*
You are given a rooted tree with root in vertex 1. Each vertex is
    coloured in some colour.
Let's call colour c dominating in the subtree of vertex v if there are no
    other colours
that appear in the subtree of vertex v more times than colour c.
So it's possible that two or more colours will be dominating in
the subtree of some vertex. The subtree of vertex v is the vertex v
and all other vertices that contains vertex v in each path to the root.
For each vertex v find the sum of all dominating colours in the subtree
    of vertex v.
*/

int u, v, n, color[MAX], parent[MAX];
vi graph[MAX];
map<int, int> inNode[MAX];
int mxCnt[MAX]; ll sum[MAX], out[MAX];

void merge(int u, int v)
{
	// ***Important: swapping parents
	if (inNode[parent[u]].size() < inNode[parent[v]].size())
		swap(parent[u], parent[v]);

	for (auto it : inNode[parent[v]])
	{
		int f = it.first, s = it.second;

		inNode[parent[u]][f] += s;

		if (inNode[parent[u]][f] > mxCnt[parent[u]])
		{
			mxCnt[parent[u]] = inNode[parent[u]][f];
			sum[parent[u]] = f;
		}
		else if (inNode[parent[u]][f] == mxCnt[parent[u]])
		{
			sum[parent[u]] += f;
		}
	}

	inNode[parent[v]].clear();
}

void dfs(int u, int p)
{
	for (auto v : graph[u])
	{
		if (p == v) continue;
		dfs(v, u);
		merge(u, v);
	}

	out[u] = sum[parent[u]];
}

int main()
{
	ios_base::sync_with_stdio(0);
	// cin.tie(NULL); cout.tie(NULL);
	// freopen("in.txt","r",stdin);

	int test, cases = 1;

	n = getnum();

	FOR(i, 1, n + 1)
	{
		color[i] = getnum();
		parent[i] = i;
		inNode[i][color[i]] = 1;
		sum[i] = color[i];
		mxCnt[i] = 1;
	}

	FOR(i, 1, n)
	{
		u = getnum();
		v = getnum();
```

```
		graph[u].pb(v);
		graph[v].pb(u);
	}

	dfs(1, 0);

	FOR(i, 1, n + 1) printf("%lld ", out[i]); puts("");

	return 0;
}
```

2.11 Dynamic Segment Tree with Lazy Prop

```
// Solves SPOJ HORRIBLE. Range addition and range sum query.
struct node {
	int from, to;
	long long value, lazy;
	node *left, *right;
	node() {
		from = 1;
		to = 1e5;
		value = 0;
		lazy = 0;
		left = NULL;
		right = NULL;
	}
	void extend() {
		if (left == NULL) {
			left = new node();
			right = new node();
			left->from = from;
			left->to = (from + to) >> 1;
			right->from = ((from + to) >> 1) + 1;
			right->to = to;
		}
	}
};

node *root;
int tests, n, queries;
void update_tree(node *curr, int left, int right, long long value) {
	if (curr->lazy) {
		curr->value += (curr->to - curr->from + 1) * curr->lazy;
		if (curr->from != curr->to) {
			curr->extend();
			curr->left->lazy += curr->lazy;
			curr->right->lazy += curr->lazy;
		}
		curr->lazy = 0;
	}
	if ((curr->from) > (curr->to) || (curr->from) > right ||
	    (curr->to) < left) return;
	if (curr->from >= left && curr->to <= right) {
		curr->value += (curr->to - curr->from + 1) * value;
		if (curr->from != curr->to) {
			curr->extend();
			curr->left->lazy += value;
			curr->right->lazy += value;
		}
		return;
	}
	curr->extend();
	update_tree(curr->left, left, right, value);
	update_tree(curr->right, left, right, value);
	curr->value = curr->left->value + curr->right->value;
}
long long query_tree(node *curr, int left, int right) {
	if ((curr->from) > (curr->to) || (curr->from) > right ||
	    (curr->to) < left) return 0;
	if (curr->lazy) {
		curr->value += (curr->to - curr->from + 1) * curr->lazy;
		curr->extend();
		curr->left->lazy += curr->lazy;
		curr->right->lazy += curr->lazy;
		curr->lazy = 0;
	}
	if (curr->from >= left && curr->to <= right) return curr->value;
	long long q1, q2;
	curr->extend();
	q1 = query_tree(curr->left, left, right);
	q2 = query_tree(curr->right, left, right);
	return q1 + q2;
}

int main() {
	int type, p, q;
	long long v;
	int i;
```

```
	scanf("%d", &tests);
	while (tests--) {
		root = new node();
		scanf("%d %d", &n, &queries);
		for (i = 1; i <= queries; i++) {
			scanf("%d", &type);
			if (type == 0) {
				scanf("%d %d %lld", &p, &q, &v);
				if (p > q) swap(p, q);
				update_tree(root, p, q, v);
			}
			else {
				scanf("%d %d", &p, &q);
				if (p > q) swap(p, q);
				printf("%lld\n", query_tree(root, p, q));
			}
		}
	}
	return 0;
}
```

2.12 Dynamic Segment Tree

```
/*******************************************************************************
    Implicit segment tree with addition on the interval
    and getting the value of some element.
    Works on the intervals like [1..10^9].
    O(logN) on query, O(NlogN) of memory.
    Author: Bekzhan Kassenov.
    Based on problem 3327 from informatics.mccme.ru
    http://informatics.mccme.ru/moodle/mod/statements/view.php?chapterid=3327
*******************************************************************************/
#include <iostream>
#include <cstdio>
#include <cstdlib>

using namespace std;

typedef long long ll;
struct Node {
    ll sum;
    Node *l, *r;
    Node() : sum(0), l(NULL), r(NULL) { }
};
void add(Node *v, int l, int r, int q_l, int q_r, ll val) {
    if (l > r || q_r < l || q_l > r)
        return;
    if (q_l <= l && r <= q_r) {
        v -> sum += val;
        return;
    }
    int mid = (l + r) >> 1;
    if (v -> l == NULL)
        v -> l = new Node();
    if (v -> r == NULL)
        v -> r = new Node();
    add(v -> l, l, mid, q_l, q_r, val);
    add(v -> r, mid + 1, r, q_l, q_r, val);
}
ll get(Node *v, int l, int r, int pos) {
    if (!v || l > r || pos < l || pos > r)
        return 0;
    if (l == r)
        return v -> sum;
    int mid = (l + r) >> 1;
    return v -> sum + get(v -> l, l, mid, pos) + get(v -> r, mid + 1, r,
        pos);
}
int n, m, t, x, y, val;
char c;
int main() {
    Node *root = new Node();

    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &x);
        add(root, 0, n - 1, i, i, x);
    }
    scanf("%d", &m);
    for (int i = 0; i < m; i++) {
        scanf("\n%c", &c);
        if (c == 'a') {
            scanf("%d%d%d", &x, &y, &val);
            add(root, 0, n - 1, --x, --y, val);
        } else {
            scanf("%d", &x);
            printf("%I64d ", get(root, 0, n - 1, --x));
```

```
        }
    }
    return 0;
}
```

2.13 Fenwick Tree 3D

```
#define MAX 205
/// 3D Fenwick tree, Range updates and point queries
struct Fenwick3D{
    int n, m, r, tree[MAX][MAX][MAX];

    Fenwick3D(){
    }

    Fenwick3D(int a, int b, int c){
        clr(tree);
        n = a, m = b, r = c;
    }

    /// Add v to the cube from lower-right [i,j,k] to upper-left [1,1,1]
    void update(int i, int j, int k, int v){
        if ((i < 0) || (j < 0) || (i > n) || (j > m) || (k < 0) || (k >
            r)) return;

        while (i){
            int x = j;
            while (x){
                int y = k;
                while (y){
                    tree[i][x][y] += v;
                    y ^= (y & (-y));
                }
                x ^= (x & (-x));
            }
            i ^= (i & (-i));
        }
    }

    /// Add v to the cube from upper-left [x1,y1,z1] to lower-right
        [x2,y2,z2]
    void update(int x1, int y1, int z1, int x2, int y2, int z2){
        update(x2, y2, z2, 1), update(x1 - 1, y1 - 1, z2, 1);
        update(x1 - 1, y2, z1 - 1, 1), update(x2, y1 - 1, z1 - 1, 1);
        update(x1 - 1, y2, z2, -1), update(x2, y1 - 1, z2, -1);
        update(x2, y2, z1 - 1, -1), update(x1 - 1, y1 - 1, z1 - 1, -1);
    }

    /// Query for the value at index [i][j][k]
    int query(int i, int j, int k){
        int res = 0;
        while (i <= n){
            int x = j;
            while (x <= m){
                int y = k;
                while (y <= r){
                    res += tree[i][x][y];
                    y += (y & (-y));
                }
                x += (x & (-x));
            }
            i += (i & (-i));
        }
        return res;
    }
};
```

2.14 GP Hash Table

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

// For integer
gp_hash_table<int, int> table;

// Custom hash function approach is better
const int RANDOM =
    chrono::high_resolution_clock::now().time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<int, int, chash> table;

const ll TIME =
    chrono::high_resolution_clock::now().time_since_epoch().count();
const ll SEED = (ll)(new ll);
```

```
const ll RANDOM = TIME ^ SEED;
const ll MOD = (int)1e9+7;
const ll MUL = (int)1e6+3;
 struct chash{
    ll operator()(ll x) const { return std::hash<ll>{}((x ^ RANDOM) % MOD
        * MUL); }
};
gp_hash_table<ll, int, chash> table;

unsigned hash_f(unsigned x) {
    x = ((x >> 16) ^ x) * 0x45d9f3b;
    x = ((x >> 16) ^ x) * 0x45d9f3b;
    x = (x >> 16) ^ x;
    return x;
}
struct chash {
    int operator()(ll x) const { return hash_f(x); }
};
gp_hash_table<ll, int, chash> table[N][N];
// so table[i][j][k] is storing an integer for corresponding k as hash
unsigned hash_combine(unsigned a, unsigned b) { return a * 31 + b; }

// For pairs
// The better the hash function, the less collisions
// Note that hash function should not be costly
struct chash {
    int operator()(pii x) const { return x.first* 31 + x.second; }
};
gp_hash_table<pii, int, chash> table;

// Another recommended hash function by neal on CF
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
            chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
gp_hash_table<ll,int,custom_hash> safe_gp_hash_table;
unordered_map<ll,int,custom_hash> safe_umap;

typedef gp_hash_table<int, int, hash<int>,
equal_to<int>, direct_mod_range_hashing<int>, linear_probe_fn<>,
hash_standard_resize_policy<hash_prime_size_policy,
hash_load_check_resize_trigger<true>, true>>
gp;
gp Tree;
// Now Tree can probably be used for fenwick, indices can be long long
// S is an offset to handle negative value
// If values can be >= -1e9, S=1e9+1
// maxfen is the MAXN in fenwick, this case it was 2e9+2;
// Note that it was okay to declare gp in integer as the values were
// still in the range of int.
void add(long long p, int v) {
    for (p += S; p < maxfen; p += p & -p)
        Tree[p] += v;
}
int sum(int p) {
    int ans = 0;
    for (p += S; p; p ^= p & -p)
        ans += Tree[p];
    return ans;
}
```

2.15 HLD Sample Problem

```
// Query 1: From u to v, print the index of the minimum k numbers,
// and these numbers are removed
// Query 2: Add a value to the subtree of node v
int n, m, q;
int parent[MAX], depth[MAX], subsize[MAX], st[MAX], at[MAX];
int nxt[MAX], chain[MAX], pos, cnt;
ll in[MAX], aux[MAX];
int chainsz[MAX], head[MAX];
vi graph[MAX];
vector<ll> girls[MAX];

class SegmentTree
{
public:
```

```
pair<ll,int> Tree[4*MAX];
ll Lazy[4*MAX];
void build(int node, int l, int r)
{
	Lazy[node]=0;
	if(l==r)
	{
		Tree[node]={aux[l],at[l]};
		return;
	}
	int mid=(l+r)/2;
	build(lc,l,mid);
	build(rc,mid+1,r);
	Tree[node]=min(Tree[lc],Tree[rc]);
}
void pushdown(int node)
{
	if(Lazy[node])
	{
		Lazy[lc]+=Lazy[node];
		Lazy[rc]+=Lazy[node];
		Tree[lc].first+=Lazy[node];
		Tree[rc].first+=Lazy[node];
		Lazy[node]=0;
	}
}
void update(int node, int l, int r, int x, int y, ll val)
{
	if(x>r || y<l) return;
	if(x<=l && r<=y)
	{
		Tree[node].first+=val;
		Lazy[node]+=val;
		return;
	}

	if(l!=r) pushdown(node);

	int mid=(l+r)/2;
	update(lc,l,mid,x,y,val);
	update(rc,mid+1,r,x,y,val);
	Tree[node]=min(Tree[lc],Tree[rc]);
}
pair<ll,int> query(int node, int l, int r, int x, int y)
{
	if(x>r || y<l) return {INF,inf};
	if(x<=l && r<=y) return Tree[node];
	if(l!=r) pushdown(node);

	int mid=(l+r)/2;
	return min(query(lc,l,mid,x,y),query(rc,mid+1,r,x,y));
	}
} segtree;

class HLD
{
public:
	void init(int n)
	{
		for(int i=0; i<=n; i++) nxt[i]=-1, chainsz[i]=0;
		cnt=pos=1;
	}
	void dfs(int u, int p=-1, int d=0)
	{
		parent[u]=p;
		subsize[u]=1;
		depth[u]=d;

		for(auto v: graph[u])
		{
			if(v==p) continue;
			dfs(v,u,d+1);
			subsize[u]+=subsize[v];

			if(nxt[u]==-1 || subsize[v]>subsize[nxt[u]])
				nxt[u]=v;
		}
	}
	void decompose(int u, int p=-1)
	{
		chain[u]=cnt-1;
		at[pos]=u;
		st[u]=pos++; // Flatening nodes in order of heavy edges!
		aux[st[u]]=in[u];
		if(!chainsz[cnt-1]) head[cnt-1]=u;
		chainsz[cnt-1]++;

		if(nxt[u]!=-1) decompose(nxt[u],u);
		for(auto v: graph[u])
		{
```

```
                if(v==p || v==nxt[u]) continue;
                ++cnt;
                decompose(v,u);
        }
    }
    pair<ll,int> query(int u, int v)
    {
        pair<ll,int> ret={INF,inf};

        while(chain[u]!=chain[v])
        {
            if(depth[head[chain[u]]]<depth[head[chain[v]]])
                swap(u,v);
            int start=head[chain[u]];
            ret=min(ret,segtree.query(1,1,n,st[start],st[u]));
            u=parent[start];
        }

        if(depth[u]>depth[v]) swap(u,v);
        ret=min(ret,segtree.query(1,1,n,st[u],st[v]));
        return ret;
    }
} hld;

void handle(int u, int v, int k)
{
    vi ans;
    while(k--)
    {
        auto out=hld.query(u,v);
        if(out.first>=INF) break;
        // The node which has current minimum weight of a girl
        int idx=out.second;
        ll last=girls[idx].back();
        ans.pb(last);
        girls[idx].pop_back();
        segtree.update(1,1,n,st[idx],st[idx],abs(girls[idx].back()-last));
    }
    printf("%d ", (int)ans.size());
    FOR(i,0,ans.size()) printf("%d ", ans[i]);
    puts("");
}

int main()
{
    int test, cases = 1;
    scanf("%d%d%d", &n, &m, &q);
    int u, v;
    FOR(i,0,n-1)
    {
      scanf("%d%d", &u, &v);
      graph[u].pb(v);
      graph[v].pb(u);
    }
    FOR(i,1,m+1)
    {
      scanf("%d", &pos);
      // Girl i is in node i, same node can have multiple girls
      // Initial weight of each girl equals her index
      girls[pos].pb(i);
    }

    FOR(i,1,n+1) girls[i].pb(INF);
    FOR(i,1,n+1)
    {
            REVERSE(girls[i]);
            in[i]=girls[i].back();
    }
    hld.init(n);
    hld.dfs(1);
    hld.decompose(1);
    segtree.build(1,1,n);
    while(q--)
    {
      int t, k;
      scanf("%d", &t);
      if(t==1)
      {
            scanf("%d%d%d", &u, &v, &k);
            handle(u,v,k);
      }
      else
      {
            scanf("%d%d", &v, &k);
            segtree.update(1,1,n,st[v],st[v]+subsize[v]-1,k);
      }
    }
    return 0;
}
```

2.16 HashMap

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

struct hashmap{
    int t, sz, hmod;
    vector <int> id;
    vector <long long> key, val;

    inline int nextPrime(int n){
        for (int i = n; ;i++){
            for (int j = 2; ;j++){
                if ((j * j) > i) return i;
                if ((i % j) == 0) break;
            }
        }
        return -1;
    }

    void clear(){t++;}

    inline int pos(unsigned long long x){
        int i = x % hmod;
        while (id[i] == t && key[i] != x) i++;
        return i;
    }

    inline void insert(long long x, long long v){
        int i = pos(x);
        if (id[i] != t) sz++;
        key[i] = x, val[i] = v, id[i] = t;
    }

    inline void erase(long long x){
        int i = pos(x);
        if (id[i] == t) key[i] = 0, val[i] = 0, id[i] = 0, sz--;
    }

    inline long long find(long long x){
        int i = pos(x);
        return (id[i] != t) ? -1 : val[i];
    }

    inline bool contains(long long x){
        int i = pos(x);
        return (id[i] == t);
    }

    inline void add(long long x, long long v){
        int i = pos(x);
        (id[i] == t) ? (val[i] += v) : (key[i] = x, val[i] = v, id[i] = t,
            sz++);
    }

    inline int size(){
        return sz;
    }

    hashmap(){}
    hashmap(int m){
        srand(time(0));
        m = (m << 1) - ran(1, m);
        hmod = nextPrime(max(100, m));

        sz = 0, t = 1;
        id.resize(hmod + 0x1FF, 0);
        key.resize(hmod + 0x1FF, 0), val.resize(hmod + 0x1FF, 0);
    }
};

int main(){
}
```

2.17 Heavy Light Decomposition

```
int parent[MAX], depth[MAX], subsize[MAX];
int nxt[MAX], chain[MAX], st[MAX], pos, cnt;
int chainsz[MAX], head[MAX];
vi graph[MAX];

class HLD
```

```
{
public:
	void init(int n)
	{
		for(int i=0; i<=n; i++) nxt[i]=-1, chainsz[i]=0;
		cnt=pos=1;
	}
	void dfs(int u, int p=-1, int d=0)
	{
		parent[u]=p;
		subsize[u]=1;
		depth[u]=d;

		for(auto v: graph[u])
		{
			if(v==p) continue;
			dfs(v,u,d+1);
			subsize[u]+=subsize[v];

			if(nxt[u]==-1 || subsize[v]>subsize[nxt[u]])
				nxt[u]=v;
		}
	}
	void decompose(int u, int p=-1)
	{
		chain[u]=cnt-1;
		// May need to update in segment tree on pos with some
		    val[u]
		st[u]=pos++;
		// Take the node value to corresponding position
		// val[st[u]]=(ll)c[u]*a+b;
		if(!chainsz[cnt-1]) head[cnt-1]=u;
		chainsz[cnt-1]++;

		if(nxt[u]!=-1) decompose(nxt[u],u);

		for(auto v: graph[u])
		{
			if(v==p || v==nxt[u]) continue;
			++cnt;
			decompose(v,u);
		}
	}
	void update(int u, int v, ll add)
	{
		while(chain[u]!=chain[v])
		{
			if(depth[head[chain[u]]]<depth[head[chain[v]]])
				swap(u,v);
			int start=head[chain[u]];
			segtree.update(1,1,n,st[start],st[u],add);
			u=parent[start];
		}
		if(depth[u]>depth[v]) swap(u,v);
		segtree.update(1,1,n,st[u],st[v],add);
	}

	int query(int u, int v)
	{
		int ret=0;
		while(chain[u]!=chain[v])
		{
			if(depth[head[chain[u]]]<depth[head[chain[v]]])
				swap(u,v);
			int start=head[chain[u]];
			// query on respective ds
			ret+=bit.query(st[start],st[u]);
			u=parent[start];
		}

		if(depth[u]>depth[v]) swap(u,v);
		ret+=bit.query(st[u],st[v]);

		return ret;
	}
} hld;
```

2.18 How Many Values Less than a Given Value

```
// How many values in a range are less than or equal to the given value?
// The key idea is to sort the values under a node in the segment tree
    and use binary search to find
// the required count
// Complexity is O(nlog^2n) for building
// The actual problem needed the number of such values and the cumulative
    sum of them
// Tree[node].All has all the values and Tree[node].Pref has the prefix
    sums
```

```
// Remember: upper_bound gives the number of values less than or equal to
    given value in a sorted range
struct info
{
	vector<ll> All, Pref;
} Tree[MAX * 4];
ll T[MAX], Prefix[MAX];
void build(int node, int l, int r)
{
	if (l == r)
	{
		Tree[node].All.pb(T[l]);
		Tree[node].Pref.pb(T[l]);
		return;
	}
	int mid = (l + r) / 2;
	build(lc, l, mid);
	build(rc, mid + 1, r);
	for (auto it : Tree[lc].All)
		Tree[node].All.pb(it);
	for (auto it : Tree[rc].All)
		Tree[node].All.pb(it);
	SORT(Tree[node].All);
	ll now = 0;
	for (auto it : Tree[node].All)
	{
		Tree[node].Pref.pb(now + it);
		now += it;
	}
}
pair<ll, ll> query(int node, int l, int r, int x, int y, int val)
{
	if (x > r || y < l) return MP(0LL, 0LL);
	if (x <= l && r <= y)
	{
		int idx = upper_bound(Tree[node].All.begin(),
		    Tree[node].All.end(), val) - Tree[node].All.begin();
		if (idx > 0) return MP(Tree[node].Pref[idx - 1], idx);
		return MP(0LL, 0LL);
	}
	int mid = (l + r) / 2;
	pair<ll, ll> ret, left, right;
	left = query(lc, l, mid, x, y, val);
	right = query(rc, mid + 1, r, x, y, val);
	ret.first += left.first; ret.second += left.second;
	ret.first += right.first; ret.second += right.second;
	return ret;
}
```

2.19 Li Chao Tree Lines

```
typedef int ftype;
typedef complex<ftype> point;
#define x real
#define y imag

ftype dot(point a, point b) {
    return (conj(a) * b).x();
}

ftype f(point a, ftype x) {
    return dot(a, {x, 1});
}

const int maxn = 2e5;

point line[4 * maxn];

void add_line(point nw, int v = 1, int l = 0, int r = maxn) {
    int m = (l + r) / 2;
    bool lef = f(nw, l) < f(line[v], l);
    bool mid = f(nw, m) < f(line[v], m);
    if(mid) {
        swap(line[v], nw);
    }
    if(r - l == 1) {
        return;
    } else if(lef != mid) {
        add_line(nw, 2 * v, l, m);
    } else {
        add_line(nw, 2 * v + 1, m, r);
    }
}

int get(int x, int v = 1, int l = 0, int r = maxn) {
    int m = (l + r) / 2;
    if(r - l == 1) {
        return f(line[v], x);
```

```
    } else if(x < m) {
        return min(f(line[v], x), get(x, 2 * v, l, m));
    } else {
        return min(f(line[v], x), get(x, 2 * v + 1, m, r));
    }
}
```

2.20 Li Chao Tree Parabolic Sample

```
/* Problem:
Given n functions yi(x) = a0 + a1x + a2x^2 + a3x^3 and q queries. For
    each query, you are
given an integer t and you are required to find out yi (i  i  n) that
    minimizes the value of yi(t).

Li Chao Tree works for functions that intersect only in one point.
    Constraints of the problem were
such that there would always be at most on intersecting point between to
    functions with x>=350. So
we bruteforced for x<350 and built Li Chao Tree for x>=350
*/

const int N=1e5; // Max query points
const int offset=350; // Bruteforce for this limit
struct fun
{
     ll a, b, c, d;
     fun(){a=0, b=0, c=0, d=INF;}
     fun(ll a, ll b, ll c, ll d) :
          a(a), b(b), c(c), d(d) { }
     ll eval(ll x)
     {
          return a*x*x*x+b*x*x+c*x+d;
     }
} Tree[4*N+5];

ll aux[offset+5];

void init()
{
     ms(aux,63);
     FOR(i,1,4*N) Tree[i]=fun();
}
```

```
int compare(ll x, ll y)
{
     if(x<y) return -1;
     return x>y;
}

void update(int node, int l, int r, int x, int y, fun fx)
{
     if(x>r || y<l) return;
     if(x<=l && r<=y)
     {
          // cout<<"x-y: "<<x<<" "<<y<<endl;
          // cout<<"l-r: "<<l<<" "<<r<<endl;

          // fx - new function, Tree[node] - old function
          int mid=(l+r)/2;

          int fl=compare(fx.eval(l),Tree[node].eval(l));
          int fr=compare(fx.eval(r),Tree[node].eval(r));
          int fm1=compare(fx.eval(mid),Tree[node].eval(mid));
          int fm2=compare(fx.eval(mid+1),Tree[node].eval(mid+1));

          // New function is worse for l to r, no point of adding it.
          if(fl>=0 && fr>=0) return;

          // New function is better for l to r, add it
          if(fl<=0 && fr<=0)
          {
               Tree[node]=fx;
               return;
          }

          // New function is better for l to mid, add it. Old function
              can still be
          // better for right segment.
          if(fl<=0 && fm1<=0)
          {
               // Sending the old function to right segment
               update(rc,mid+1,r,x,y,Tree[node]);
               Tree[node]=fx;
               return;
          }
```

```
        // New function is worse for l to mid, but this can be better
            for right segment.
        if(fl>=0 && fm1>=0)
        {
            update(rc,mid+1,r,x,y,fx);
            return;
        }

        // New function worse for mid+1 to r, but can be better for
            left segment
        if(fm2>=0 && fr>=0)
        {
            update(lc,l,mid,x,y,fx);
            return;
        }

        // New function better for mid+1 to r, add it, old function
            can still be better for left.
        if(fm2<=0 && fr<=0)
        {
            update(lc,l,mid,x,y,Tree[node]);
            Tree[node]=fx;
            return;
        }
    }
    else if(l<r)
    {
        int mid=(l+r)/2;

        update(lc,l,mid,x,y,fx);
        update(rc,mid+1,r,x,y,fx);
    }
}

ll query(int node, int l, int r, int x)
{
    if(l==r) return Tree[node].eval(x);

    int mid=(l+r)/2;

    ll ret=Tree[node].eval(x);

    if(x<=mid) ret=min(ret,query(lc,l,mid,x));
    else ret=min(ret,query(rc,mid+1,r,x));

    return ret;
}

void calc(fun &fx)
{
    for(int i=0; i<offset; i++)
    {
        aux[i]=min(aux[i],fx.eval(i));
        // prnt(fx.eval(i));
    }
}

int main()
{
    // ios_base::sync_with_stdio(0);
    // cin.tie(NULL); cout.tie(NULL);
    // freopen("in.txt","r",stdin);

    int test, cases = 1;

    scanf("%d", &test);

    int n;

    while(test--)
    {
        scanf("%d", &n);

        init();
        int a, b, c, d;

        FOR(i,0,n)
        {
            scanf("%d%d%d%d", &d, &c, &b, &a);
            fun fx=fun(a,b,c,d);
            calc(fx);
            update(1,1,N,offset,N,fx);
        }

        int q, x;
        scanf("%d", &q);

        while(q--)
        {
            scanf("%d", &x);
```

```
                if(x<offset) printf("%lld\n", aux[x]);
                else
                {
                      ll out=query(1,1,N,x);
                      printf("%lld\n", out);
                }
           }
      }

      return 0;
}
```

2.21 Mo Algorithm Example

```
struct info
{
       int l, r, id;
       info(){}
       info(int l, int r, int id) : l(l), r(r), id(id){}
};

int n, t, a[2*MAX];
info Q[2*MAX];
int Block, cnt[1000004];
ll ans=0;
ll Ans[2*MAX];

// always constant
// Farther improvement: When dividing the elements into blocks, we may
    sort the first block in the
// ascending order of right borders, the second  in descending, the third
     in ascending order again, and so on.

// inline bool comp(const info &a, const info &b)
// {
//     if(a.l/blockSZ!=b.l/blockSZ)
//         return a.l<b.l;
//     if((a.l/blockSZ)&1)
//         return a.r<b.r;
//     return a.r>b.r;
// }

inline bool comp(const info &a, const info &b)
{
       if(a.l/Block==b.l/Block) return a.r<b.r;
       return a.l<b.l;
}

inline void Add(int idx)
{
       ans+=(2*cnt[a[idx]]+1)*a[idx];
       cnt[a[idx]]++;

       /* Actual meaning of the above code

       ans-=cnt[a[idx]]*cnt[a[idx]]*a[idx];
       cnt[a[idx]]++;
       ans+=cnt[a[idx]]*cnt[a[idx]]*a[idx];

       */
}

inline void Remove(int idx)
{
       ans-=(2*cnt[a[idx]]-1)*a[idx];
       cnt[a[idx]]--;

       /* Actual meaning of the above code

       ans-=cnt[a[idx]]*cnt[a[idx]]*a[idx];
       cnt[a[idx]]--;
       ans+=cnt[a[idx]]*cnt[a[idx]]*a[idx];

       */
}

int main()
{
       // ios_base::sync_with_stdio(0);
       // cin.tie(NULL); cout.tie(NULL);
       // freopen("in.txt","r",stdin);

       // Problem: For each query, find the value cnt[a[i]]*cnt[a[i]]*a[i]

       scanf("%d%d", &n, &t);

       Block=sqrt(n);
```

```
	FOR(i,1,n+1) a[i]=getnum();

	FOR(i,0,t)
	{
		Q[i].l=getnum();
		Q[i].r=getnum();
		Q[i].id=i;
	}

	sort(Q,Q+t,comp);

	int Left=0, Right=-1;

	FOR(i,0,t)
	{
		while(Left<Q[i].l)
		{
			Remove(Left);
			Left++;
		}
		while(Left>Q[i].l)
		{
			Left--;
			Add(Left);
		}
		while(Right<Q[i].r)
		{
			Right++;
			Add(Right);
		}
		while(Right>Q[i].r)
		{
			Remove(Right);
			Right--;
		}

		Ans[Q[i].id]=ans;
	}

	FOR(i,0,t) printf("%lld\n", Ans[i]);

	return 0;
}
```

2.22 Mo on Tree Path

```
int aux[MAX], b[MAX], n, m, weight[MAX], u, v;
vi graph[MAX];
int parent[MAX][17], st[MAX], en[MAX], tag = 0, dist[MAX], blocSZ;
int go[100005], lca[100005], cnt[MAX], t[MAX];
bool seen[MAX];
struct info
{
	int u, v, id;
	bool fl;
	info() {}
	info(int u, int v, int id, bool fl) : u(u), v(v), id(id), fl(fl) {
	    }
};
vector<info> Q;
// "Unordered"
void compress(int n, int *in, int *out)
{
	unordered_map <int, int> mp;
	for (int i = 1; i <= n; i++) out[i] = mp.emplace(in[i],
	    mp.size()).first->second;
}
void dfs(int u, int p, int d)
{
	parent[u][0] = p;
	st[u] = ++tag;
	dist[u] = d;
	for (auto v : graph[u])
	{
		if (v != p) dfs(v, u, d + 1);
	}
	en[u] = ++tag;
	aux[st[u]] = u;
	aux[en[u]] = u;
}
void sparse()
{
	for (int j = 1; 1 << j < n; j++)
	{
		for (int i = 1; i <= n; i++)
		{
			if (parent[i][j - 1] != -1)
				parent[i][j] = parent[parent[i][j - 1]][j -
				    1];
```

```
                }
        }
}
int query(int p, int q)
{
        if (dist[p] < dist[q]) swap(p, q);
        int x = 1;
        while (true)
        {
                if ((1 << (x + 1)) > dist[p]) break;
                x++;
        }
        FORr(i, x, 0) if (dist[p] - (1 << i) >= dist[q]) p = parent[p][i];
        if (p == q) return p;
        FORr(i, x, 0)
        {
                if (parent[p][i] != -1 && parent[p][i] != parent[q][i])
                {
                        p = parent[p][i];
                        q = parent[q][i];
                }
        }
        return parent[p][0];
}
int ans = 0;
void doit(int idx)
{
        if (!seen[aux[idx]])
        {
                cnt[b[idx]]++;
                if (cnt[b[idx]] == 1) ans++;
        }
        else
        {
                cnt[b[idx]]--;
                if (cnt[b[idx]] == 0) ans--;
        }
        seen[aux[idx]] ^= 1;
}
int main()
{
        // Each node has some weight associated with it
        // u v : ask for how many different integers that represent the
            weight of
        // nodes there are on the path from u to v.
        ms(parent, -1);
        scanf("%d%d", &n, &m);
        blocSZ = sqrt(n);
        FOR(i, 1, n + 1)
        {
                scanf("%d", &weight[i]);
        }
        FOR(i, 1, n)
        {
                scanf("%d%d", &u, &v);
                graph[u].pb(v);
                graph[v].pb(u);
        }
        dfs(1, 0, 0);
        sparse();
        compress(n, weight, t);
        (1, 1) << endl;
        FOR(i, 1, 2 * n + 1) b[i] = t[aux[i]];
        FOR(i, 0, m)
        {
                scanf("%d%d", &u, &v);
                lca[i] = query(u, v);
                if (st[u] > st[v]) swap(u, v);
                if (lca[i] == u) Q.pb(info(st[u], st[v], i, 0));
                else Q.pb(info(en[u], st[v], i, 1));
        }
        sort(Q.begin(), Q.end(), [](const info & a, const info & b)->bool
        {
                if (a.u / blocSZ == b.u / blocSZ) return a.v < b.v;
                return a.u < b.u;
        });
        int L = 1, R = 0;
        FOR(i, 0, Q.size())
        {
                int l = Q[i].u, r = Q[i].v, anc = lca[Q[i].id];

                while (R < r) { R++; doit(R); }
                while (R > r) { doit(R); R--; }
                while (L > l) { L--; doit(L); }
                while (L < l) { doit(L); L++; }

                if (Q[i].fl)
                {
                        if (!cnt[b[st[anc]]])
                                go[Q[i].id] = ans + 1;
```

```
                else go[Q[i].id] = ans;
            }
            else go[Q[i].id] = ans;
        }
        FOR(i, 0, m) printf("%d\n", go[i]);
        return 0;
}
```

2.23 Order Statistics Tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>

using namespace __gnu_pbds;
using namespace __gnu_cxx;

// Order Statistic Tree
/* Special functions:

            find_by_order(k) --> returns iterator to the kth largest
                element counting from 0
            order_of_key(val) --> returns the number of items in a set
                that are strictly smaller than our item
*/

typedef tree< int, null_type, less<int>,
rb_tree_tag, tree_order_statistics_node_update> ordered_set;
```

2.24 Ordered Multiset

```
#include <bits/stdtr1c++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

/*** Needs C++11 or C++14 ***/

/// Treap supporting duplicating values in set
/// Maximum value of treap * ADD must fit in long long

struct Treap{ /// hash = 96814
    int len;
    const int ADD = 1000010;
    const int MAXVAL = 1000000010;
    tr1::unordered_map <long long, int> mp; /// Change to int if only int
        in treap
    tree<long long, null_type, less<long long>, rb_tree_tag,
        tree_order_statistics_node_update> T;

    Treap(){
        len = 0;
        T.clear(), mp.clear();
    }

    inline void clear(){
        len = 0;
        T.clear(), mp.clear();
    }

    inline void insert(long long x){
        len++, x += MAXVAL;
        int c = mp[x]++;
        T.insert((x * ADD) + c);
    }

    inline void erase(long long x){
        x += MAXVAL;
        int c = mp[x];
        if (c){
            c--, mp[x]--, len--;
            T.erase((x * ADD) + c);
        }
    }

    /// 1-based index, returns the K'th element in the treap, -1 if none
        exists
    inline long long kth(int k){
        if (k < 1 || k > len) return -1;
        auto it = T.find_by_order(--k);
        return ((*it) / ADD) - MAXVAL;
    }

    /// Count of value < x in treap
    inline int count(long long x){
```

```
        x += MAXVAL;
        int c = mp[--x];
        return (T.order_of_key((x * ADD) + c));
    }

    /// Number of elements in treap
    inline int size(){
        return len;
    }
};

int main(){
}
```

2.25 Persistent Segment Tree 1

```
// Calculate how many distinct values are there in a given range
// Persistent Segment Tree implementation
// Actually used in Codeforces - The Bakery

int n, k, a[MAX], last[MAX], nxt[MAX];
int idx=1;
int Tree[64*MAX], L[64*MAX], R[64*MAX], root[2*MAX], rt[MAX];
int pos[MAX];

void build(int node, int l, int r)
{
        if(l==r)
        {
                Tree[node]=0;
                return;
        }

        L[node]=++idx;
        R[node]=++idx;

        // cout<<node<<" "<<L[node]<<" "<<R[node]<<endl;

        int mid=(l+r)/2;

        build(L[node],l,mid);
        build(R[node],mid+1,r);
        Tree[node]=0;
}

int update(int node, int l, int r, int pos, int val)
{
        int x;
        x=++idx;

        if(l==r)
        {
                Tree[x]=val;
                return x;
        }

        L[x]=L[node]; R[x]=R[node];

        int mid=(l+r)/2;

        if(pos<=mid) L[x]=update(L[x],l,mid,pos,val);
        else R[x]=update(R[x],mid+1,r,pos,val);

        Tree[x]=Tree[L[x]]+Tree[R[x]];

        return x;
}

int query(int node, int l, int r, int x, int y)
{
        if(x>r || y<l) return 0;
        if(x<=l && r<=y) return Tree[node];

        int mid=(l+r)/2;

        int q1=query(L[node],l,mid,x,y);
        int q2=query(R[node],mid+1,r,x,y);

        return q1+q2;
}

int getCost(int l, int mid)
{
        return query(root[rt[mid]],1,n,l,mid);
}

int main()
```

```
{
    int test, cases=1;

    scanf("%d%d", &n, &k);

    build(1,1,n);

    root[0]=1;
    int t=1;

    FOR(i,1,n+1)
    {
        scanf("%d", &a[i]);

        int k=pos[a[i]];

        if(!k)
        {
                root[t]=update(root[t-1],1,n,i,1);
                t++;
        }
        else
        {
                root[t]=update(root[t-1],1,n,k,0);
                t++;
                root[t]=update(root[t-1],1,n,i,1);
                t++;
        }

        rt[i]=t-1;
        pos[a[i]]=i;
    }

    return 0;
}
```

2.26 Persistent Segment Tree 2

```
const int MAXN = (1 << 20);

struct node
{
        int sum;
        node *l, *r;
        node() { l = nullptr; r = nullptr; sum = 0; }
        node(int x) { sum = x; l = nullptr; r = nullptr;  }
};

typedef node* pnode;

pnode merge(pnode l, pnode r)
{
        pnode ret = new node(0);
        ret->sum = l->sum + r->sum;
        ret->l = l;
        ret->r = r;
        return ret;
}

pnode init(int l, int r)
{
        if(l == r) { return (new node(0));}

        int mid = (l + r) >> 1;
        return merge(init(l, mid), init(mid + 1, r));
}

pnode update(int pos, int val, int l, int r, pnode nd)
{
        if(pos < l || pos > r) return nd;
        if(l == r) { return (new node(val)); }

        int mid = (l + r) >> 1;
        return merge(update(pos, val, l, mid, nd->l), update(pos, val, mid
            + 1, r, nd->r));
}

int query(int qL, int qR, int l, int r, pnode nd)
{
        if(qL <= l && r <= qR) return nd->sum;
        if(qL > r || qR < l) return 0;

        int mid = (l + r) >> 1;
    return query(qL, qR, l, mid, nd->l) + query(qL, qR, mid + 1, r,
        nd->r);
}

int get_kth(int k, int l, int r, pnode nd)
```

```
{
        if(l == r) return l;

        int mid = (l + r) >> 1;
        if(nd->l->sum < k) return get_kth(k - nd->l->sum, mid + 1, r,
            nd->r);
        else return get_kth(k, l, mid, nd->l);
}
```

2.27 Persistent Trie

```
#include <bits/stdc++.h>

using namespace std;

// Problem: find maximum value (x^a[j]) in the range (l,r) where l<=j<=r

const int N = 1e5 + 100;
const int K = 15;

struct node_t;
typedef node_t * pnode;

struct node_t {
  int time;
  pnode to[2];
  node_t() : time(0) {
    to[0] = to[1] = 0;
  }
  bool go(int l) const {
    if (!this) return false;
    return time >= l;
  }
  pnode clone() {
    pnode cur = new node_t();
    if (this) {
      cur->time = time;
      cur->to[0] = to[0];
      cur->to[1] = to[1];
    }
    return cur;
  }
};

pnode last;
pnode version[N];

void insert(int a, int time) {
  pnode v = version[time] = last = last->clone();
  for (int i = K - 1; i >= 0; --i) {
    int bit = (a >> i) & 1;
    pnode &child = v->to[bit];
    child = child->clone();
    v = child;
    v->time = time;
  }
}

int query(pnode v, int x, int l) {
  int ans = 0;
  for (int i = K - 1; i >= 0; --i) {
    int bit = (x >> i) & 1;
    if (v->to[bit]->go(l)) { // checking if this bit was inserted before
        the range
      ans |= 1 << i;
      v = v->to[bit];
    } else {
      v = v->to[bit ^ 1];
    }
  }
  return ans;
}

void solve() {
  int n, q;
  scanf("%d %d", &n, &q);
  last = 0;
  for (int i = 0; i < n; ++i) {
    int a;
    scanf("%d", &a);
    insert(a, i);
  }
  while (q--) {
    int x, l, r;
    scanf("%d %d %d", &x, &l, &r);
    --l, --r;
    printf("%d\n", query(version[r], ~x, l));
    // Trie version[r] contains the trie for [0...r] elements
```

```
  }
}
```

2.28 RMQ Sparse Table

```
const int MAXN = (1 << 20);
const int MAXLOG = 20;

struct sparse_table
{
	int dp[MAXN][MAXLOG];
	int prec_lg2[MAXN], n;

	sparse_table() { memset(dp, 0, sizeof(dp)); memset(prec_lg2, 0,
	    sizeof(prec_lg2)); n = 0; }

	void init(vector<int> &a)
	{
		n = a.size();
		for(int i = 2; i < 2 * n; i++) prec_lg2[i] = prec_lg2[i >>
		    1] + 1;
		for(int i = 0; i < n; i++) dp[i][0] = a[i];
		for(int j = 1; (1 << j) <= n; j++)
			for(int i = 0; i < n; i++)
				dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j
				    - 1))][j - 1]);
	}

	int query(int l, int r)
	{
		int k = prec_lg2[r - l + 1];
		return min(dp[l][k], dp[r - (1 << k) + 1][k]);
	}
};
```

2.29 Range Sum Query by Lazy Propagation

```
int a[MAX + 7], tree[4 * MAX + 7], lazy[4 * MAX + 7];
void build(int node, int l, int r)
{
	if (l == r)
	{
		tree[node] = a[l];
		return;
	}
	if (l >= r) return;
	int mid = (l + r) / 2;
	build(node * 2, l, mid);
	build(node * 2 + 1, mid + 1, r);
	tree[node] = tree[node * 2] + tree[node * 2 + 1];
}
void upd(int node, int l, int r, int v)
{
	lazy[node] += v;
	tree[node] += (r - l + 1) * x;
}
void pushDown(int node, int l, int r) //passing update information to the
    children
{
	int mid = (l + r) / 2;
	upd(node * 2, l, mid, lazy[node]);
	upd(node * 2 + 1, mid + 1, r, lazy[node]);
	lazy[node] = 0;
}
void update(int node, int l, int r, int x, int y, int v)
{
	if (x > r || y < l) return;
	if (x >= l && r <= y)
	{
		upd(node, l, r, v);
		return;
	}
	pushDown(node, l, r);
	int mid = (l + r) / 2;
	update(node * 2, l, mid, x, y, v);
	update(node * 2 + 1, mid + 1, r, x, y, v);
	tree[node] = tree[node * 2] + tree[node * 2 + 1];
}
```

2.30 Rope

```
#include <ext/rope>
#include <bits/stdtr1c++.h>
```

```
#define MAX 50010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;
using namespace __gnu_cxx;

rope <char> R[MAX];
int d = 0, ye = 0, vnow = 0;
char str[105], out[10000010];

int main(){
    int n, i, j, k, v, p, c, x, flag;

    while (scanf("%d", &n) != EOF){
        d = 0, vnow = 0;
        while (n--){
            scanf("%d", &flag);

            if (flag == 1){
                scanf("%d %s", &p, str);
                p -= d, vnow++;
                R[vnow] = R[vnow - 1];
                R[vnow].insert(p, str); /// Insert string str after
                    position p
            }

            if (flag == 2){
                scanf("%d %d", &p, &c);
                p -= d, c -= d, vnow++;
                R[vnow] = R[vnow - 1];
                R[vnow].erase(p - 1, c); /// Remove c characters starting
                    at position p
            }

            if (flag == 3){
                scanf("%d %d %d", &v, &p, &c); /// Print c characters
                    starting at position p in version v

                v -= d, p -= d, c -= d;
                rope <char> sub = R[v].substr(p - 1, c); /// Get the
                    substring of c characters from position p in version v
                for (auto it: sub){
                    out[ye++] = it;
                    if (it == 'c') d++;
                }
                out[ye++] = 10;
            }
        }
    }

    fwrite(out, 1, ye, stdout);
    return 0;
}
```

2.31 Segment Tree with Lazy Prop

```
// Maximum in a range with lazy propagation.
class SegmentTree
{
public:
        ll Tree[4*MAX], Lazy[4*MAX];
        void pushdown(int node)
        {
                if(Lazy[node])
                {
                        Lazy[lc]+=Lazy[node];
                        Lazy[rc]+=Lazy[node];
                        Tree[lc]+=Lazy[node];
                        Tree[rc]+=Lazy[node];
                        Lazy[node]=0;
                }
        }

        void build(int node, int l, int r)
        {
                Lazy[node]=0;
                if(l==r)
                {
                        Tree[node]=in[l]; // input values
                        return;
                }
                int mid=(l+r)/2;
                build(lc,l,mid);
                build(rc,mid+1,r);
                Tree[node]=max(Tree[lc],Tree[rc]);
                Lazy[node]=0;
```

```
	}
	// Range update
	void update(int node, int l, int r, int x, int y, ll val)
	{
		// puts("range update");
		if(x>r || y<l) return;
		if(x<=l && r<=y)
		{
			Tree[node]+=val;
			Lazy[node]+=val;
			return;
		}

		if(l!=r) pushdown(node);

		int mid=(l+r)/2;
		update(lc,l,mid,x,y,val);
		update(rc,mid+1,r,x,y,val);
		Tree[node]=max(Tree[lc],Tree[rc]);
	}
	// Range query
	ll query(int node, int l, int r, int x, int y)
	{
		if(x>r || y<l) return -INF;
		if(x<=l && r<=y) return Tree[node];
		if(l!=r) pushdown(node);

		int mid=(l+r)/2;
		return max(query(lc,l,mid,x,y),query(rc,mid+1,r,x,y));
	}
} segtree;
```

2.32 Splay Tree

```
/**
    Splay Tree :
    Node:
        void addIt(int ad) : adding an integer in a range
        void revIt() : reversing flag
        void upd() : push_up( gather from child)
        void pushdown() : pass values to the child( like lazy propagation)
    Splay:
        Node* newNode(int v,Node* f) :Returns Pointer of a node whose
            parent is f,and value v
        Node* build(int l,int r,Node* f) : building [L,R] which parent is f
        void rotate(Node* t,int d) : Rotation of Splay Tree
        void splay(Node* t,Node* f) : Splaying , t resides just below the f
        void select(int k,Node *f) : Select k th element in the tree
            ,splay it to the just below f
        Node*&get(int l, int r) : Getting The node for segment [L,R]
        void reverse(int l,int r) : Reverse a segment
        void del(int p) : deletes entry a[p]
        void split(int l,int r,Node*&s1) : Split the array and s1 stores
            the [L,R] segment
        void cut(int l,int r) : Cut the segment [L,R] and insert in at the
            end
        void insert(int p,int v): Insert after p,( 0 means before the
            array) an element whose value is v
        void insertRange(int pos,Node *s): Insert after pos, an segment
            denoted by s
        int query(int l,int r): Output desired result for [L,R]
        void addRange(int l,int r,int v): Add v to all the element in
            segment [L,R]
        void output(int l,int r) : Output the segment [L,R]
**/

/*
The following code answers the following queries
1 L R Output Maximum value in range [L,R]
2 L R Reverse the array [L,R]
3 L R v add v in range [L,R]
4 pos removes entry from pos
5 pos v - insert an element after position v

We assumes the initial array stored in ar[]={1,2,3,4... n}
*/

typedef int T;

const int N = 2e5+50; // >= Node + Query
T ar[N];          // Initial Array
struct Node{
    Node *ch[2],*pre; // child and parent
    T val; // Value stored in each node
    int size; //size of the subtree rooted at this node
    T mx; // additional info stored to solve problems, here maximum value
    T sum;
```

```
    T add;//lazy updates
    bool rev;// reverse flag
    Node(){size=0;val=mx=-1e9;add=0;}
    void addIt(T ad){
        add+=ad;
        mx+=ad;
        sum += size*ad;
        val+=ad;
    }
    void revIt(){
        rev^=1;
    }
    void upd(){
        size=ch[0]->size+ch[1]->size+1;
        mx=max(val,max(ch[0]->mx,ch[1]->mx));
        sum= ch[0]->sum + ch[1]->sum + val;
    }
    void pushdown();
}Tnull,*null=&Tnull;
void Node::pushdown(){
    if (add!=0){
        for (int i=0;i<2;++i)
            if (ch[i]!=null) ch[i]->addIt(add);
        add = 0;
    }
    if (rev){
        swap(ch[0],ch[1]);
        for (int i=0;i<2;i++)
            if (ch[i]!=null) ch[i]->revIt();
        rev = 0;
    }
}
struct Splay{
    Node nodePool[N],*cur; // Static Memory and cur pointer
    Node* root; // root of the splay tree
    Splay(){
        cur=nodePool;
        root=null;
    }

    void clear(){
        cur=nodePool;
        root=null;
    }
    Node* newNode(T v,Node* f){
        cur->ch[0]=cur->ch[1]=null;
        cur->size=1;
        cur->val=v;
        cur->mx=v;cur->sum = 0;
        cur->add=0;
        cur->rev=0;
        cur->pre=f;
        return cur++;
    }

    Node* build(int l,int r,Node* f){
        if(l>r) return null;
        int m=(l+r)>>1;
        Node* t=newNode(ar[m],f);
        t->ch[0]=build(l,m-1,t);
        t->ch[1]=build(m+1,r,t);
        t->upd();
        return t;
    }

    void rotate(Node* x,int c){
        Node* y=x->pre;
        y->pushdown();
        x->pushdown();

        y->ch[!c]=x->ch[c];
        if (x->ch[c]!=null) x->ch[c]->pre=y;
        x->pre=y->pre;
        if (y->pre!=null)
        {
            if (y->pre->ch[0]==y) y->pre->ch[0]=x;
            else y->pre->ch[1]=x;
        }
        x->ch[c]=y;
        y->pre=x;
        y->upd();
        if (y==root) root=x;
    }

    void splay(Node* x,Node* f){
        x->pushdown();
        while (x->pre!=f){
            if (x->pre->pre==f){
                if (x->pre->ch[0]==x) rotate(x,1);
                else rotate(x,0);
```

```
        }else{
            Node *y=x->pre,*z=y->pre;
            if (z->ch[0]==y){
                if (y->ch[0]==x) rotate(y,1),rotate(x,1);
                else rotate(x,0),rotate(x,1);
            }else{
                if (y->ch[1]==x) rotate(y,0),rotate(x,0);
                else rotate(x,1),rotate(x,0);
            }
        }
    }
    x->upd();
}
void select(int k,Node* f){
    int tmp;
    Node* x=root;
    x->pushdown();
    k++;
    for(;;){
        x->pushdown();
        tmp=x->ch[0]->size;
        if (k==tmp+1) break;
        if (k<=tmp) x=x->ch[0];
        else{
            k-=tmp+1;
            x=x->ch[1];
        }
    }
    splay(x,f);
}

Node*&get(int l, int r){
    select(l-1,null);
    select(r+1,root);
    return root->ch[1]->ch[0];
}

void reverse(int l,int r){
    Node* o=get(l,r);
    o->rev^=1;
    splay(o,null);
}
void del(int p)
{
        select(p-1,null);
        select(p+1,root);
        root->ch[1]->ch[0] = null;
        splay(root->ch[1],null);
}
void split(int l,int r,Node*&s1)
{
    Node* tmp=get(l,r);
    root->ch[1]->ch[0]=null;
    root->ch[1]->upd();
    root->upd();
    s1=tmp;
}
void cut(int l,int r)
{
    Node* tmp;
    split(l,r,tmp);
    select(root->size-2,null);
    root->ch[1]->ch[0]=tmp;
    tmp->pre=root->ch[1];
    root->ch[1]->upd();
    root->upd();
}

void init(int n){
    clear();
    root=newNode(0,null);
    root->ch[1]=newNode(n+1,root);
    root->ch[1]->ch[0]=build(1,n,root->ch[1]);
    splay(root->ch[1]->ch[0],null);
}


void insertPos(int pos,T v)
{
        select(pos,null);
        select(pos+1,root);
        root->ch[1]->ch[0] = newNode(v,root->ch[1]);
        splay(root->ch[1]->ch[0],null);
}
void insertRange(int pos,Node *s)
{
    select(pos,null);
    select(pos+1,root);
    root->ch[1]->ch[0] = s;
    s->pre = root->ch[1];
```

```
        root->ch[1]->upd();
        root->upd();
    }
    T query(int l,int r)
    {
            Node *o = get(l,r);
            return o->mx;

    }
    void addRange(int l,int r,T v)
    {
            Node *o = get(l,r);
            o->add += v;
            o->val += v;
            o->sum += o->size * v;
            splay(o,null);

    }
    void output(int l,int r){
        for (int i=l;i<=r;i++){
            select(i,null);
            cout<<root->val<<endl;
        };
    }
}St;



int main()
{
    int n,m,a,b,c;

    scanf("%d%d", &n, &m);

    for(int i= 1;i <= n;i ++ ) ar[i] = i;
    St.init(n);

    FOR(i,1,m+1)
    {
        scanf("%d%d", &a, &b);
        St.cut(a,b);
    }

    St.output(1,n);

    return 0;
}
```

2.33 Venice Technique

```
/*
We want a data structure capable of doing three main update-operations
    and some
sort of query. The three modify operations are: add: Add an element to
    the set.
remove: Remove an element from the set. updateAll: This one normally
    changes in
this case subtract X from ALL the elements. For this technique it is
    completely
required that the update is done to ALL the values in the set equally.
And also for this problem in particular we may need one query:
getMin: Give me the smallest number in the set.
*/
// Interface of the Data Structure
struct VeniceSet {
        void add(int);
        void remove(int);
        void updateAll(int);
        int getMin(); // custom for this problem
        int size();
};

/*
Imagine you have an empty land and the government can make queries of the
    following
type: * Make a building with A floors. * Remove a building with B floors.
    * Remove
C floors from all the buildings. (A lot of buildings can be vanished) *
    Which is the
smallest standing building. (Obviously buildings which are already
    banished don't count)
The operations 1,2 and 4 seems very easy with a set, but the 3 is very
    cost effective
probably O(N) so you might need a lot of workers. But what if instead of
    removing C
floors we just fill the streets with enough water (as in venice) to cover
    up the
```

```
first C floors of all the buildings :O. Well that seems like cheating but
    at least
those floor are now vanished :). So in order to do that we apart from the
    SET
we can maintain a global variable which is the water level. so in fact if
    we
have an element and want to know the number of floors it has we can just
    do
height - water_level and in fact after water level is for example 80, if
    we
want to make a building of 3 floors we must make it of 83 floors so that
    it
can touch the land.
*/
struct VeniceSet {
	multiset<int> S;
	int water_level = 0;
	void add(int v) {
		S.insert(v + water_level);
	}
	void remove(int v) {
		S.erase(S.find(v + water_level));
	}
	void updateAll(int v) {
		water_level += v;
	}
	int getMin() {
		return *S.begin() - water_level;
	}
	int size() {
		return S.size();
	}
};
VeniceSet mySet;
for (int i = 0; i < N; ++i) {
	mySet.add(V[i]);
	mySet.updateAll(T[i]); // decrease all by T[i]
	int total = T[i] * mySet.size(); // we subtracted T[i] from all
	    elements

	// in fact some elements were already less than T[i]. So we
	    probbaly are counting
	// more than what we really subtracted. So we look for all those
	    elements
	while (mySet.getMin() < 0) {
		// get the negative number which we really did not
		    subtracted T[i]
		int toLow = mySet.getMin();

		// remove from total the amount we over counted
		total -= abs(toLow);

		// remove it from the set since I will never be able to
		    substract from it again
		mySet.remove(toLow);
	}
	cout << total << endl;
}
cout << endl;
```

3 Game

3.1 Green Hacenbush

```
// Green Hackenbush
vi graph[505];
int go(int u, int p)
{
	int ret = 0;
	for (auto &v : graph[u])
	{
		if (v == p) continue;
		ret ^= (go(v, u) + 1);
	}
	return ret;
}
int u, v, n;
int main()
{
// ios_base::sync_with_stdio(0);
// cin.tie(NULL); cout.tie(NULL);
// freopen("in.txt","r",stdin);
	int test, cases = 1;
	cin >> test;
	while (test--)
	{
		cin >> n;
```

```
		FOR(i, 0, n - 1)
		{
			cin >> u >> v;
			graph[u].pb(v);
			graph[v].pb(u);
		}
		if (go(1, 0)) puts("Alice");
		else puts("Bob");
		FOR(i, 1, n + 1) graph[i].clear();
	}
	return 0;
}
```

3.2 Green Hackenbush 2

```
//
// Green Hackenbush
//
// Description:
//   Consider a two player game on a graph with a specified vertex (root).
//   In each turn, a player eliminates one edge.
//   Then, if a subgraph that is disconnected from the root, it is
    removed.
//   If a player cannot select an edge (i.e., the graph is singleton),
//   he will lose.
//
//   Compute the Grundy number of the given graph.
//
// Algorithm:
//   We use two principles:
//     1. Colon Principle: Grundy number of a tree is the xor of
//        Grundy number of child subtrees.
//        (Proof: easy).
//
//     2. Fusion Principle: Consider a pair of adjacent vertices u, v
//        that has another path (i.e., they are in a cycle). Then,
//        we can contract u and v without changing Grundy number.
//        (Proof: difficult)
//
//   We first decompose graph into two-edge connected components.
//   Then, by contracting each components by using Fusion Principle,
//   we obtain a tree (and many self loops) that has the same Grundy
//   number to the original graph. By using Colon Principle, we can
//   compute the Grundy number.
//
// Complexity:
//   O(m + n).
//
// Verified:
//   SPOJ 1477: Play with a Tree
//   IPSC 2003 G: Got Root?
//
//
#include <iostream>
#include <vector>
#include <cstdio>
#include <algorithm>
#include <functional>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define TEST(s) if (!(s)) { cout << __LINE__ << " " << #s << endl;
    exit(-1); }

struct hackenbush {
	int n;
	vector<vector<int>> adj;

	hackenbush(int n) : n(n), adj(n) { }
	void add_edge(int u, int v) {
		adj[u].push_back(v);
		if (u != v) adj[v].push_back(u);
	}

	// r is the only root connecting to the ground
	int grundy(int r) {
		vector<int> num(n), low(n);
		int t = 0;
		function<int(int, int)> dfs = [&](int p, int u) {
			num[u] = low[u] = ++t;
			int ans = 0;
			for (int v : adj[u]) {
				if (v == p) { p += 2 * n; continue; }
				if (num[v] == 0) {
					int res = dfs(u, v);
```

```
                                low[u] = min(low[u], low[v]);
                                if (low[v] > num[u]) ans ^= (1 + res)
                                    ^ 1; // bridge
                                else                ans ^= res;
                                                // non bridge
                        } else low[u] = min(low[u], num[v]);
                }
                if (p > n) p -= 2 * n;
                for (int v : adj[u])
                        if (v != p && num[u] <= num[v]) ans ^= 1;
                return ans;
        };
        return dfs(-1, r);
    }
};

int main() {
    int cases; scanf("%d", &cases);
    for (int icase = 0; icase < cases; ++icase) {
        int n; scanf("%d", &n);
        vector<int> ground(n);
        int r;
        for (int i = 0; i < n; ++i) {
            scanf("%d", &ground[i]);
            if (ground[i] == 1) r = i;
        }
        int ans = 0;
        hackenbush g(n);
        for (int i = 0; i < n - 1; ++i) {
            int u, v;
            scanf("%d %d", &u, &v);
            --u; --v;
            if (ground[u]) u = r;
            if (ground[v]) v = r;
            if (u == v) ans ^= 1;
            else g.add_edge(u, v);
        }
        int res = ans ^ g.grundy(r);
        printf("%d\n", res != 0);
    }
}
```

4 Geometry

4.1 Convex Hull

```
struct PT
{
        int x, y;
        PT(){}
        PT(int x, int y) : x(x), y(y) {}
        bool operator < (const PT &P) const
        {
                return x<P.x || (x==P.x && y<P.y);
        }
};



ll cross(const PT p, const PT q, const PT r)
{
        return (ll)(q.x-p.x)*(ll)(r.y-p.y)-(ll)(q.y-p.y)*(ll)(r.x-p.x);
}

vector<PT> Points, Hull;

void findConvexHull()
{
        int n=Points.size(), k=0;

        SORT(Points);

        // Build lower hull

        FOR(i,0,n)
        {
                while(Hull.size()>=2 &&
                    cross(Hull[Hull.size()-2],Hull.back(),Points[i])<=0)
                {
                        Hull.pop_back();
                        k--;
                }
                Hull.pb(Points[i]);
                k++;
        }
```

```
	// Build upper hull

	for(int i=n-2, t=k+1; i>=0; i--)
	{
		while(Hull.size()>=t &&
		    cross(Hull[Hull.size()-2],Hull.back(),Points[i])<=0)
		{
			Hull.pop_back();
			k--;
		}
		Hull.pb(Points[i]);
		k++;
	}

	Hull.resize(k);
}
```

4.2 Counting Closest Pair of Points

```
int n;
struct Points
{
	double x, y;
	Points() {}
	Points(double x, double y) : x(x), y(y) { }
	bool operator<(const Points &a) const
	{
		return x < a.x;
	}
};
bool comp1(const Points &a, const Points &b)
{
	return a.x < b.x;
}
bool comp2(const Points &a, const Points &b)
{
	return a.y < b.y;
}
void printPoint(Points a)
{
	cout << a.x << " " << a.y << endl;
}
Points P[10005];
typedef set<Points, bool(*)(const Points&, const Points&)> setType;
typedef setType::iterator setIT;
setType s(&comp2);
double euclideanDistance(const Points &a, const Points &b)
{
// prnt((double)(a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
	return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
}
map<double, map<double, int> > CNT;
int main()
{
// ios_base::sync_with_stdio(0);
// cin.tie(NULL); cout.tie(NULL);
// freopen("in.txt","r",stdin);
	while ((cin >> n) && n)
	{
		FOR(i, 0, n) cin >> P[i].x >> P[i].y;
		sort(P, P + n, comp1);
		FOR(i, 0, n)
		{
// printPoint(P[i]);
			s.insert(P[i]);
			CNT[P[i].x][P[i].y]++;
		}
// To check repeated points :/
// for(auto it: s) printPoint(it);
		double ans = 10000;
		int idx = 0;
		FOR(j, 0, n)
		{
// cout<<"Point now: "; printPoint(P[j]);
			if (CNT[P[j].x][P[j].y] > 1) ans = 0;
			Points it = P[j];
			while (it.x - P[idx].x > ans)
			{
				s.erase(P[idx]);
				idx++;
			}
			Points low = Points(it.x, it.y - ans);
			Points high = Points(it.x, it.y + ans);
			setIT lowest = s.lower_bound(low);
			if (lowest != s.end())
			{
				setIT highest = s.upper_bound(high);
```

```
                        for (setIT now = lowest; now != highest;
                            now++)
                        {
                                double cur = sqrt(euclideanDistance
                                                (*now, it));
// prnt(cur);
                                if (cur == 0) continue;
// cout<<"Here:"<<endl;
// printPoint(*now); printPoint(it); prnt
                                (cur);
                                if (cur < ans)
                                {
                                        ans = cur;
                                }
                        }
                }
                s.insert(it);
        }
// cout<<"Set now:"<<endl;
// for(auto I: s) printPoint(I);
        if (ans < 10000) cout << setprecision(4) << fixed << ans
            << endl;
        else prnt("INFINITY");
        s.clear();
        CNT.clear();
    }
    return 0;
}
```

4.3 Maximum Points to Enclose in a Circle of Given Radius with Angular Sweep

```
typedef pair<double,bool> pdb;

#define START 0
#define END 1

struct PT
{
    double x, y;
    PT() {}
    PT(double x, double y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y) {}
    PT operator + (const PT &p) const { return PT(x+p.x, y+p.y); }
    PT operator - (const PT &p) const { return PT(x-p.x, y-p.y); }
    PT operator * (double c)  const { return PT(x*c, y*c ); }
    PT operator / (double c)  const { return PT(x/c, y/c ); }
};

PT p[505];
double dist[505][505];
int n, m;

void calcDist()
{
    FOR(i,0,n)
    {
        FOR(j,i+1,n)
            dist[i][j]=dist[j][i]=sqrt((p[i].x-p[j].x)*(p[i].x-p[j].x)
                +(p[i].y-p[j].y)*(p[i].y-p[j].y));
    }
}

// Returns maximum number of points enclosed by a circle of radius
    'radius'
// where the circle is pivoted on point 'point'
// 'point' is on the circumfurence of the circle

int intelInside(int point, double radius)
{
    vector<pdb> ranges;

    FOR(j,0,n)
    {
        if(j==point || dist[j][point]>2*radius) continue;

        double a1=atan2(p[point].y-p[j].y,p[point].x-p[j].x);
        double a2=acos(dist[point][j]/(2*radius));

        ranges.pb({a1-a2,START});
        ranges.pb({a1+a2,END});
    }

    sort(ALL(ranges));

    int cnt=1, ret=cnt;

    for(auto it: ranges)
```

```
	{
		if(it.second) cnt--;
		else cnt++;
		ret=max(ret,cnt);
	}

	return ret;
}

// returns maximum amount of points enclosed by the circle of radius r
// Complexity: O(n^2*log(n))

int go(double r)
{
	int cnt=0;

	FOR(i,0,n)
	{
		cnt=max(cnt,intelInside(i,r));
	}

	return cnt;
}
```

4.4 Point in Polygon Binary Search

```
int sideOf(const PT &s, const PT &e, const PT &p)
{
	ll a = cross(e-s,p-s);
	return (a > 0) - (a < 0);
}

bool onSegment(const PT &s, const PT &e, const PT &p)
{
	PT ds = p-s, de = p-e;
	return cross(ds,de) == 0 && dot(ds,de) <= 0;
}

/*
Main routine
Description: Determine whether a point t lies inside a given polygon
    (counter-clockwise order).
The polygon must be such that every point on the circumference is visible
    from the first point in the vector.
It returns 0 for points outside, 1 for points on the circumference, and 2
    for points inside.
*/

int insideHull2(const vector<PT> &H, int L, int R, const PT &p) {
	int len = R - L;
	if (len == 2) {
		int sa = sideOf(H[0], H[L], p);
		int sb = sideOf(H[L], H[L+1], p);
		int sc = sideOf(H[L+1], H[0], p);
		if (sa < 0 || sb < 0 || sc < 0) return 0;
		if (sb==0 || (sa==0 && L == 1) || (sc == 0 && R ==
		    (int)H.size()))
			return 1;
		return 2;
	}
	int mid = L + len / 2;
	if (sideOf(H[0], H[mid], p) >= 0)
		return insideHull2(H, mid, R, p);
	return insideHull2(H, L, mid+1, p);
}

int insideHull(const vector<PT> &hull, const PT &p) {
	if ((int)hull.size() < 3) return onSegment(hull[0], hull.back(),
	    p);
	else return insideHull2(hull, 1, (int)hull.size(), p);
}
```

4.5 Rectangle Union

```
struct info
{
	int x, ymin, ymax, type;
	info(){}
	info(int x, int ymin, int ymax, int type) :
		x(x), ymin(ymin), ymax(ymax), type(type) { }

	bool operator < (const info &p) const
	{
		return x<p.x;
	}
```

```
};

vector<info> in;
int n, x, y, p, q, m;
vi take;
int Lazy[4*MAX], Tree[4*MAX];

void update(int node, int l, int r, int ymin, int ymax, int val)
{
	if(take[l]>ymax || take[r]<ymin) return;

	if(ymin<=take[l] && take[r]<=ymax)
	{
		Lazy[node]+=val;

		if(Lazy[node]) Tree[node]=take[r]-take[l];
		else Tree[node]=Tree[lc]+Tree[rc];

		return;
	}

	if(l+1>=r) return;

	int mid=(l+r)/2;

	update(lc,l,mid,ymin,ymax,val);
	update(rc,mid,r,ymin,ymax,val);

	if(Lazy[node]) Tree[node]=take[r]-take[l];
	else Tree[node]=Tree[lc]+Tree[rc];
}

ll solve()
{
	take.clear(); ms(Tree,0); ms(Lazy,0);
	take.pb(-1);

	FOR(i,0,in.size())
	{
		take.pb(in[i].ymin);
		take.pb(in[i].ymax);
	}

	SORT(take);
	take.erase(unique(ALL(take)),take.end());
	m=take.size()-1;

	// VecPrnt(take);

	update(1,1,m,in[0].ymin,in[0].ymax,in[0].type);

	int prv=in[0].x; ll ret=0;

	FOR(i,1,in.size())
	{
		ret+=(ll)(in[i].x-prv)*Tree[1];
		prv=in[i].x;
		update(1,1,m,in[i].ymin,in[i].ymax,in[i].type);
	}

	return ret;
}

int main()
{
    // ios_base::sync_with_stdio(0);
    // cin.tie(NULL); cout.tie(NULL);
    // freopen("in.txt","r",stdin);

    int test, cases=1;

    scanf("%d", &test);

    while(test--)
    {
        scanf("%d", &n);

        in.clear();

        FOR(i,0,n)
        {
            scanf("%d%d%d%d", &x, &y, &p, &q);

            in.pb(info(x,y,q,1));
            in.pb(info(p,y,q,-1));
        }

        SORT(in);

        ll ans=solve();
```

```
        printf("Case %d: %lld\n", cases++, ans);
    }

    return 0;
}
```

5 Graph

5.1 0-1 BFS

```
// Useful when the graph only has weights 0 or 1.
// Complexity becomes O(V+E)

for all v in vertices:
        dist[v] = inf
dist[source] = 0;
deque d
d.push_front(source)
while d.empty() == false:
        vertex = get front element and pop_front
        // Go to all edges
        for all edges e of form (vertex , u):
                // consider relaxing with 0 or 1 weight edges
                if travelling e relaxes distance to u:
                        relax dist[u]
                        if e.weight = 1:
                                d.push_back(u)
                        else:
                                d.push_front(u)
```

5.2 2-SAT 2

```
// for x or y add !x -> y, !y -> x
// x and y = (!x or y) and (x or !y) and (!x or !
/*
    inline void add_implication(int a, int b){
        if (a < 0) a = n - a;
        if (b < 0) b = n - b;
        adj[a].push_back(b);
        rev[b].push_back(a);
    }

    inline void add_or(int a, int b){
        add_implication(-a, b);
        add_implication(-b, a);
    }

    inline void add_xor(int a, int b){
        add_or(a, b);
        add_or(-a, -b);
    }

    inline void add_and(int a, int b){
        add_or(a, b);
        add_or(a, -b);
        add_or(-a, b);
    }

    /// force variable x to be true (if x is negative, force !x to be
        true)
    inline void force_true(int x){
        if (x < 0) x = n - x;
        add_implication(neg(x), x);
    }

    /// force variable x to be false (if x is negative, force !x to be
        false)
    inline void force_false(int x){
        if (x < 0) x = n - x;
        add_implication(x, neg(x));
    }
*/

struct tSAT{
    int n, id[MAX][2];
    vi G[MAX];
    int ord, dis[MAX], low[MAX], sid[MAX], scc;
    stack <int> s;

    tSAT(int n): n(n){
        int now=0;
        f(i,1,n+1){
            f(j,0,2){
```

```
            id[i][j]=++now;
        }
    }
}

tSAT() {}

void add_edge(int u,int tu,int v,int tv){
    G[id[u][tu]].pb(id[v][tv]);
}

bool feasible(){
    scc=0; ord=0;
    mem(dis,0); mem(sid,0);
    f(i,1,2*n+1){
        if(!dis[i]) tarjan(i);
    }
    f(i,1,n+1){
        if(sid[id[i][0]]==sid[id[i][1]]) return false;
    }
    return true;
}

vi solution(){
    vi ans;
    f(i,1,n+1){
        if(sid[id[i][0]]>sid[id[i][1]]) ans.pb(i);
    }
    return ans;
}

void tarjan(int u){
    s.push(u);
    dis[u]=low[u]=++ord;
    f(i,0,G[u].size()){
        int v=G[u][i];
        if (!dis[v]){
            tarjan(v);
            low[u]=min(low[v],low[u]);
        }
        else if (!sid[v]){
            low[u]=min(dis[v],low[u]);
        }
    }
    if (low[u]==dis[u]){
            ++scc;
            while(1){
                int t=s.top();
                s.pop();
                sid[t]=scc;
                if (t==u) break;
            }
        }
    }
}
```

5.3 2-SAT

```
const int N=20004;
int n, m, root=-1, leader[N], truths[N];
vi graph[N], rev[N], order;
bool visited[N];
// clear() if necessary
void dfs_reverse(int u)
{
    visited[u]=true;
    FOR(j,0,rev[u].size())
    {
        int v=rev[u][j];
        if(!visited[v])
            dfs_reverse(v);
    }
    order.pb(u);
}
void dfs(int u)
{
    visited[u]=true;
    leader[u]=root;

    FOR(j,0,graph[u].size())
    {
        int v=graph[u][j];

        if(!visited[v])
            dfs(v);
    }
}
void solve()
```

```
{
    for(int i=2*m; i>=1; i--)
    {
        if(!visited[i])
        {
            dfs_reverse(i);
        }
    }
    // important
    REVERSE(order);
    ms(visited,false);

    FOR(i,0,order.size())
    {
        if(!visited[order[i]])
        {
            root=order[i];
            dfs(order[i]);
        }
    }
}

bool sameSCC(int u, int v)
{
    return leader[u]==leader[v];
}
bool assign()
{
    FOR(i,0,order.size())
    {
        int u=order[i];

        if(u>m)
        {
            if(sameSCC(u,u-m)) return false;
            if(truths[leader[u]]==-1)
            {
                truths[leader[u]]=true;
                truths[leader[u-m]]=false;
            }
        }
        else
        {
            if(sameSCC(u,m+u)) return false;
            if(truths[leader[u]]==-1)
            {
                truths[leader[u]]=true;
                truths[leader[u+m]]=false;
            }
        }
    }
    return true;
}
int main()
{
    int test, cases = 1;
    scanf("%d", &test);
    while(test--)
    {
        scanf("%d%d", &n, &m);

        int u, v;
        ms(truths,-1);

        FOR(i,1,n+1)
        {
            scanf("%d%d", &u, &v);
            // For each clause (u or v), we add to edges - (~u to v), (~v
                to u)
            if(u > 0)
            {
                if(v > 0)
                {
                    graph[m+u].push_back(v); graph[m+v].push_back(u);
                    rev[v].push_back(m+u); rev[u].push_back(m+v);
                } else {
                    graph[m+u].push_back(m-v); graph[-v].push_back(u);
                    rev[m-v].push_back(m+u); rev[u].push_back(-v);
                }
            } else {
                if(v > 0) {
                    graph[-u].push_back(v); graph[m+v].push_back(m-u);
                    rev[v].push_back(-u); rev[m-u].push_back(m+v);
                } else {
                    graph[-u].push_back(m-v); graph[-v].push_back(m-u);
                    rev[m-v].push_back(-u); rev[m-u].push_back(-v);
                }
            }
        }
```

```
        solve();
        bool okay=assign();

        if(okay)
        {
            printf("Case %d: Yes\n", cases++);

            vi allow;

            FOR(i,1,m+1)
            {
                if(truths[leader[i]])
                {
                    allow.pb(i);
                }
            }

            printf("%d", (int)allow.size());
            FOR(i,0,allow.size()) cout<<" "<<allow[i];
            cout<<endl;
        }
        else printf("Case %d: No\n", cases++);
    }
    return 0;
}
```

5.4 Articulation Points and Bridges

```
vi graph[100];
int dfs_num[100], dfs_low[100], parent[100], cnt;
int dfsroot, rootchild;
int art_v[100];

void articulate(int u)
{
        dfs_low[u]=dfs_num[u]=cnt++;
        for (ul j=0; j<graph[u].size(); j++)
        {
                int v=graph[u][j];
                if (dfs_num[v]==-1)
                {
                        parent[v]=u;
                        if (u==dfsroot)
                                rootchild++;
                        articulate(v);
                        if (dfs_low[v]>=dfs_num[u])
                                art_v[u]=true;
                        if (dfs_low[v]>dfs_num[u])
                                cout<<"Edge "<<u<<" & "<<v<<" is a
                                    bridge."<<endl;
                        dfs_low[u]=min(dfs_low[u],dfs_low[v]);
                }
                else if (v!=parent[u])
                        dfs_low[u]=min(dfs_low[u],dfs_num[v]);
        }
}

int main()
{
        int n, m, u, v;
        cin>>n>>m;
        for (int i=0; i<m; i++)
        {
                cin>>u>>v;
                graph[u].pb(v);
                graph[v].pb(u);
        }
        cnt=0;
        ms(dfs_num,-1);
        for (int i=0; i<n; i++)
        {
                if (dfs_num[i]==-1)
                {
                        dfsroot=i;
                        rootchild=0;
                        articulate(i);
                        art_v[dfsroot]=(rootchild>1);
                }
        }
        prnt("Articulation points:");
        for (int i=0; i<n; i++)
        {
                if (art_v[i])
                        cout<<"Vertex: "<<i<<endl;
        }
        return 0;
}
```

5.5 BCC

```
struct MagicComponents {

	struct edge {
		ll u, v, id;
	};

	ll num, n, edges;

	vector<ll> dfs_num, low, vis;
	vector<ll> cuts; // art-vertices
	vector<edge> bridges; // bridge-edges
	vector<vector<edge>> adj; // graph
	vector<vector<edge>> bccs; // all the bccs where bcc[i] has all
	    the edges inside it
	deque<edge> e_stack;

	// Nodes are numberd from 0

	MagicComponents(const ll& _n) : n(_n) {
		adj.assign(n, vector<edge>());
		edges = 0;
	}

	void add_edge(const ll& u, const ll& v) {
		adj[u].push_back({u,v,edges});
		adj[v].push_back({v,u,edges++});
	}

	void run(void) {
		vis.assign(n, 0);
		dfs_num.assign(n, 0);
		low.assign(n, 0);
		bridges.clear();
		cuts.clear();
		bccs.clear();
		e_stack = deque<edge>();
		num = 0;

		for (ll i = 0; i < n; ++i) {
			if (vis[i]) continue;
			dfs(i, -1);
		}
	}

	void dfs(const ll& node, const ll& par) {
		dfs_num[node] = low[node] = num++;
		vis[node] = 1;
		ll n_child = 0;
		for (edge& e : adj[node]) {
			if (e.v == par) continue;
			if (vis[e.v] == 0) {
				++n_child;
				e_stack.push_back(e);
				dfs(e.v, node);

				low[node] = min(low[node], low[e.v]);
				if (low[e.v] >= dfs_num[node]) {
					if (dfs_num[node] > 0 || n_child > 1)
						cuts.push_back(node);
					if (low[e.v] > dfs_num[node]) {
						bridges.push_back(e);

						pop(node);
					} else pop(node);
				}
			} else if (vis[e.v] == 1) {
				low[node] = min(low[node], dfs_num[e.v]);
				e_stack.push_back(e);
			}
		}
		vis[node] = 2;
	}

	void pop(const ll& u) {
		vector<edge> list;
		for (;;) {
			edge e = e_stack.back();
			e_stack.pop_back();
			list.push_back(e);
			if (e.u == u) break;
		}
		bccs.push_back(list);
	}

	//# Make sure to call run before calling this function.
	// Function returns a new graph such that all two connected
	// components are compressed into one node and all bridges
	// in the previous graph are the only edges connecting the
```

```
// components in the new tree.
// map is an integer array that will store the mapping
// for each node in the old graph into the new graph. //$
MagicComponents component_tree(vector<ll>& map) {
	vector<char> vis(edges);
	for (const edge& e : bridges)
		vis[e.id] = true;

	ll num_comp = 0;
	map.assign(map.size(), -1);
	for (ll i = 0; i < n; ++i) {
		if (map[i] == -1) {
			deque<ll> q;
			q.push_back(i);
			map[i] = num_comp;
			while (!q.empty()) {
				ll node = q.front();
				q.pop_front();
				for (const edge& e : adj[node]) {
					if (!vis[e.id] && map[e.v] ==
					    -1) {
						vis[e.id] = true;
						map[e.v] = num_comp;
						q.push_back(e.v);
					}
				}
			}
		}
		++num_comp;
	}

	MagicComponents g(num_comp);
	vis.assign(vis.size(), false);
	for (ll i = 0; i < n; ++i) {
		for (const edge& e : adj[i]) {
			if (!vis[e.id] && map[e.v] < map[e.u]) {
				vis[e.id] = true;
				// This is an edge in the bridge tree
				// we can add this edge to a new
				    graph[] and this will
				// be our new tree. We can now do
				    operations on this tree
				g.add_edge(map[e.v], map[e.u]);
			}
		}
	}
	return g;
}

//# Make sure to call run before calling this function.
// Function returns a new graph such that all biconnected
// components are compressed into one node. Cut nodes will
// be in multiple components, so these nodes will also have
// their own component by themselves. Edges in the graph
// represent components to articulation points
// map is an integer array that will store the mapping
// for each node in the old graph into the new graph.
// Cut points to their special component, and every other node
// to their specific component. //$
MagicComponents bcc_tree(vector<ll>& map) {
	vector<ll> cut(n, -1);
	ll size = bccs.size();
	for (const auto& i : cuts)
		map[i] = cut[i] = size++;

	MagicComponents g(size);
	vector<ll> used(n);
	for (ll i = 0; i < bccs.size(); ++i) {
		for (const edge& e : bccs[i]) {
			vector<ll> tmp = {e.u,e.v};
			for (const ll& node : tmp) {
				if (used[node] != i+1) {
					used[node] = i+1;
					if (cut[node] != -1)
						g.add_edge(i,
						    cut[node]);
					else map[node] = i;
				}
			}
		}
	}
	return g;
}
};
```

5.6 Bellman Ford

```
// Is there a negative cycle in the graph?
```

```
bool bellman(int src)
{
    // Nodes are indexed from 1
        for (int i = 1; i <= n; i++)
                dist[i] = INF;
        dist[src] = 0;
    for(int i = 2; i <= n; i++)
    {
        for (int j = 0; j < edges.size(); j++)
        {
            int u = edges[j].first;
            int v = edges[j].second;
            ll weight = adj[u][v];
            if (dist[u]!=INF && dist[u] + weight < dist[v])
                dist[v] = dist[u] + weight;
        }
    }
        for (int i = 0; i < edges.size(); i++)
    {
                int u = edges[i].first;
                int v = edges[i].second;
                ll weight = adj[u][v];
        // True if neg-cylce exists
                if (dist[u]!=INF && dist[u] + weight < dist[v])
                        return true;
    }
        return false;
}
```

5.7 Cycle in a Directed Graph

```
// Finds a cycle starting from a node u
const int N = 1005;
bool visited[N], instack[N];
stack<int> st;
vi graph[N]; int n;
vpii cycle; // contains the edges of the cycle
bool findCycle(int u)
{
     if(!visited[u])
     {
          visited[u]=true;
          instack[u]=true;
          st.push(u);
          for(auto v: graph[u])
          {
               if(!visited[v] && findCycle(v)) return true;
               else if(instack[v])
               {
                    cycle.pb({u,v});
                    st.pop();
                    int t=u;
                    while(v!=t)
                    {
                         cycle.pb({st.top(),t});
                         t=st.top();
                         st.pop();
                    }
                    return true;
               }
          }
     }
     instack[u]=false;
     st.pop();
     return false;
}
void find()
{
     FOR(i,1,n+1)
     {
          ms(visited,false);
          ms(instack,false);
          if(findCycle(i))
          {
               // A cycle found starting from i
          }
     }
}
```

5.8 Dijkstra!

```
struct road
{
       int u, w;
       road (int a, int b)
       {
```

```
                u=a; w=b;
        }
        bool operator < (const road & p) const
        {
                return w>p.w;
        }
};

int d[100], parent[100], start, end;
mvii g, cost;

void dijkstra (int n)
{
        ms(d,INF);
        ms(parent,-1);
        priority_queue <road> Q;
        Q.push(road(start,0));
        d[start]=0;
        while (!Q.empty())
        {
                road t=Q.top();
                Q.pop();
                int u=t.u;
                for (ul i=0; i<g[u].size(); i++)
                {
                        int v=g[u][i];
                        if (d[u]+cost[u][i]<d[v])
                        {
                                d[v]=d[u]+cost[u][i];
                                parent[v]=u;
                                Q.push(road(v,d[v]));
                        }
                }
        }
        return;
}

int main()
{
        int n, m, road_out, road_cost, cases=1;
        while (scanf("%d", &n) && n)
        {
                for (int i=1; i<=n; i++)
                {
                        cin>>m;
                        for (int j=1; j<=m; j++)
                        {
                                cin>>road_out>>road_cost;
                                g[i].pb(road_out);
                                cost[i].pb(road_cost);
                        }
                }
                scanf("%d%d", &start, &end);
                dijkstra(n);
                //cout<<d[end]<<endl;
                g.clear(); cost.clear();
                int current=end;
                vi path;
                while (current!=start)
                {
                        path.pb(parent[current]);
                        current=parent[current];
                }
                printf("Case %d: Path = ", cases++);
                for (int j=(int)path.size()-1; j>-1; j--)
                        cout<<path[j]<<" ";
                printf("%d; %d second delay\n", end, d[end]);
        }
        return 0;
}
```

5.9 Dominator Tree

```
// Problem: LightOJ Sabotaging Contest
// n - number of cities, m - number of edges, (u,v,t) - edge and cost
// Each of the q lines gives a query of k cities n[1],n[2],...,n[k];
// We have to find the number of nodes where if any one of them is
    removed, the
// shortest path to 0 from n[1]...n[k] will be increased. We also have to
    print
// the number of nodes which will be affected by such removal.

/* Solution
        Run Dijkstra, build shortest path dag, take topsort order and
            reverse it,
        according to the reversed order add one edge at a time to build
            dominator tree
```

```
	Finally, run dfs to find the level of each node and subtree size.
	    Answer is the
	(level of the lca of the nodes n[1]...n[k] + 1) and subtree size
	    of this ancestor
*/

vi graph[MAX], cost[MAX], dag[MAX], parent[MAX], Tree[MAX];
int u, v, t, n, m;
int dist[MAX];
vector<int> all;
int L[MAX], table[MAX][18], sub[MAX];
bool visited[MAX];

void clear()
{
	FOR(i,0,n)
	{
		graph[i].clear();
		cost[i].clear();
		dag[i].clear();
		parent[i].clear();
		Tree[i].clear();
		sub[i]=0;
	}
	all.clear();
	ms(table,-1);
	ms(visited,false);
}

void dfs(int u)
{
	sub[u]++;

	FOR(j,0,Tree[u].size())
	{
		int v=Tree[u][j];
		dfs(v);
		sub[u]+=sub[v];
	}
}

int query(int p, int q)
{
	if(L[p]<L[q]) swap(p,q);

	int x=1;

	while(true)
	{
		if((1<<(x+1))>L[p])
			break;
		x++;
	}

	FORr(i,x,0)
	{
		if(L[p]-(1<<i) >= L[q])
			p=table[p][i];
	}

	if(p==q) return p;

	FORr(i,x,0)
	{
		if(table[p][i]!=-1 && table[p][i]!=table[q][i])
		{
			p=table[p][i];
			q=table[q][i];
		}
	}

	return table[p][0];
}

void build(int curr)
{
	for(int j=1; (1<<j) < n; j++)
	{
		if(table[curr][j-1]!=-1)
			table[curr][j]=table[table[curr][j-1]][j-1];
	}
}

void dijkstra()
{
	priority_queue<pii,vpii,greater<pii> > PQ;
	PQ.push(pii(0,0));
	FOR(i,0,n) dist[i]=INF;
	dist[0]=0;
```

```
        while(!PQ.empty())
        {
                pii t=PQ.top();
                PQ.pop();

                int u=t.second;

                FOR(j,0,graph[u].size())
                {
                        int v=graph[u][j];

                        if(dist[u]+cost[u][j]<dist[v])
                        {
                                dist[v]=dist[u]+cost[u][j];
                                PQ.push(pii(dist[v],v));
                        }
                }
        }
}

void buildDag()
{
        FOR(i,0,n)
        {
                FOR(j,0,graph[i].size())
                {
                        int v=graph[i][j];

                        if(dist[i]!=INF && dist[v]!=INF &&
                            dist[v]==dist[i]+cost[i][j])
                        {
                                dag[i].pb(v);
                                parent[v].pb(i);
                        }
                }
        }
}

void topsort(int u)
{
        visited[u]=true;

        FOR(j,0,dag[u].size())
        {
                if(!visited[dag[u][j]]) topsort(dag[u][j]);
        }

        all.pb(u);
}

void buildTree()
{
        L[0]=0;
        REVERSE(all);

        FOR(i,0,all.size())
        {
                int now=all[i];

                if(parent[now].size())
                {
                        int anc=parent[now][0];

                        FOR(j,1,parent[now].size())
                        {
                                anc=query(anc,parent[now][j]);
                        }

                        L[now]=L[anc]+1;
                        table[now][0]=anc;
                        Tree[anc].pb(now);

                        build(now);
                }
        }
}

int main()
{
    int test, cases=1;

    scanf("%d", &test);

    while(test--)
    {
        scanf("%d%d", &n, &m);

        FOR(i,0,m)
        {
                scanf("%d%d%d", &u, &v, &t);
```

```
            graph[u].pb(v);
            graph[v].pb(u);
            cost[u].pb(t);
            cost[v].pb(t);
        }

        dijkstra();
        buildDag();
        topsort(0);
        buildTree();
        dfs(0);

        int q; scanf("%d", &q);

        printf("Case %d:\n", cases++);

        while(q--)
        {
            int x, u;

            scanf("%d", &x);

            int anc=-1;

            FOR(i,0,x)
            {
                scanf("%d", &u);

                if(dist[u]==INF) continue;

                if(anc==-1) anc=u;
                else anc=query(anc,u);
            }

            if(anc==-1) printf("0\n");
            else printf("%d %d\n", L[anc]+1, sub[anc]);
        }

        clear();
    }
    return 0;
}
```

5.10 Edge Coloring

```
/* Problem: Given a bipartite graph, find out minimum number of colors
to color all the edges such that no two adjacent edges have a same color
assigned. Number of minimum colors equals the max degree of a vertex in
the bipartite graph. We also need to assign colors to each edge.
*/

/* Comment by 300iq:
Minimum answer is max degree.
If max degree         1, we can just color all edges in one color.
Else, let's split edges of the graph into two sets, such that max degree
will be          (max degree + 1) / 2 in each of these two sets. You can
    do it
with euler circuit, add some dummy vertex to the left and right part, and
connect with them vertices with odd degree (and maybe you need to connect
them if they have odd degree). And then color the edges into two colors by
order of euler circuit. Then separate all inital edges into two groups by
    the
color. Then let's solve recursively for these two sets, and then just
    merge the answers.
*/

struct edge
{
    int a, b;
};

const int N = 1e6 + 7;
vector <int> g[N];
int col[N];
vector <edge> glob_edges;
bool vis[N];
bool us[N];
int pp;

void dfs(int v)
{
    us[v] = true;
    while (g[v].size() > 0)
    {
        int ind = g[v].back();
        g[v].pop_back();
        if (vis[ind])
        {
```

```
            continue;
        }
        vis[ind] = true;
        col[ind] = (pp ^= 1);
        dfs(glob_edges[ind].a == v ? glob_edges[ind].b :
            glob_edges[ind].a);
    }
}

vector <int> solve(vector <edge> e)
{
    if (e.empty())
    {
        return {};
    }
    vector <int> l, r;
    for (auto c : e)
    {
        l.push_back(c.a);
        r.push_back(c.b);
    }
    sort(l.begin(), l.end());
    sort(r.begin(), r.end());
    l.resize(unique(l.begin(), l.end()) - l.begin());
    r.resize(unique(r.begin(), r.end()) - r.begin());
    glob_edges.clear();
    int x = (int) l.size() + 1, y = (int) r.size() + 1;
    for (int i = 0; i < x + y; i++) g[i].clear();
    int ind = 0;
    for (auto &c : e)
    {
        c.a = lower_bound(l.begin(), l.end(), c.a) - l.begin();
        c.b = lower_bound(r.begin(), r.end(), c.b) - r.begin();
        auto ret = c;
        ret.b += x;
        glob_edges.push_back(ret);
        g[ret.a].push_back(ind);
        g[ret.b].push_back(ind);
        ind++;
    }
    vector <int> ids;
    int mx = 0;
    for (int i = 0; i < x + y; i++)
    {
        us[i] = 0;
        mx = max(mx, (int) g[i].size());
        if (g[i].size() % 2)
        {
            ids.push_back(i);
        }
    }
    bool bad = false;
    for (int i = 0; i < x + y; i++)
    {
        if (g[i].size() > 1)
        {
            bad = true;
        }
    }
    if (!bad)
    {
        vector <int> res(ind);
        return res;
    }
    else
    {
        vector <int> deg(x + y);
        for (int v : ids)
        {
            if (v < x)
            {
                glob_edges.push_back({v, x + y - 1});
                g[v].push_back(ind);
                g[x + y - 1].push_back(ind);
                ind++;
            }
            else
            {
                glob_edges.push_back({v, x - 1});
                g[v].push_back(ind);
                g[x - 1].push_back(ind);
                ind++;
            }
        }
        if (g[x - 1].size() % 2)
        {
            glob_edges.push_back({x - 1, x + y - 1});
            g[x - 1].push_back(ind);
            g[x + y - 1].push_back(ind);
            ind++;
```

```
}
for (int i = 0; i < ind; i++)
{
    col[i] = -1;
}
for (int i = 0; i < ind; i++)
{
    vis[i] = 0;
}
for (int i = 0; i < x + y; i++)
{
    if (!g[i].empty())
    {
        dfs(i);
    }
}
vector <edge> to_l, to_r;
vector <int> cols;
for (int i = 0; i < (int) e.size(); i++)
{
    cols.push_back(col[i]);
    if (col[i] == 0)
    {
        to_l.push_back(e[i]);
    }
    else
    {
        to_r.push_back(e[i]);
    }
}
auto x = solve(to_l);
auto y = solve(to_r);
int mx = *max_element(x.begin(), x.end()) + 1;
int p_x = 0, p_y = 0;
vector <int> ans;
for (int i = 0; i < (int) e.size(); i++)
{
    if (cols[i] == 0)
    {
        ans.push_back(x[p_x++]);
    }
    else
    {
        ans.push_back(mx + y[p_y++]);
    }
        }
        return ans;
    }
}
int main()
{
    vector <edge> e;
    int l, r, m;
    cin >> l >> r >> m;
    for (int i = 0; i < m; i++)
    {
        int a, b;
        cin >> a >> b;
        a--, b--;
        e.push_back({a, b});
    }
    auto ret = solve(e);
    cout << *max_element(ret.begin(), ret.end()) + 1 << '\n';
    for (int c : ret)
    {
        cout << c + 1 << '\n';
    }
}
```

5.11 Edmonds Matching

```
/*
 * Algorithm: Edmonds Blossom Maximum Matching in Generel Graph
 * Order : O( N^4 )
 * Note : vertx must be indexing based
 */

#include<stdio.h>
#include<string.h>
using namespace std;
#define MAX_V 103
#define MAX_E MAX_V*MAX_V

long nV,nE,Match[MAX_V];
long Last[MAX_V], Next[MAX_E], To[MAX_E];
long eI;
long q[MAX_V], Pre[MAX_V], Base[MAX_V];
bool Hash[MAX_V], Blossom[MAX_V], Path[MAX_V];
```

```
void Insert(long u, long v) {
    To[eI] = v, Next[eI] = Last[u], Last[u] = eI++;
    To[eI] = u, Next[eI] = Last[v], Last[v] = eI++;
}

long Find_Base(long u, long v) {
    memset( Path,0,sizeof(Path));
    for (;;) {
        Path[u] = 1;
        if (Match[u] == -1) break;
        u = Base[Pre[Match[u]]];
    }
    while (Path[v] == 0) v = Base[Pre[Match[v]]];
    return v;
}

void Change_Blossom(long b, long u) {
    while (Base[u] != b) {
        long v = Match[u];
        Blossom[Base[u]] = Blossom[Base[v]] = 1;
        u = Pre[v];
        if (Base[u] != b) Pre[u] = v;
    }
}

long Contract(long u, long v) {
    memset( Blossom,0,sizeof(Blossom));
    long b = Find_Base(Base[u], Base[v]);
    Change_Blossom(b, u);
    Change_Blossom(b, v);
    if (Base[u] != b) Pre[u] = v;
    if (Base[v] != b) Pre[v] = u;
    return b;
}

void Augment(long u) {
    while (u != -1) {
        long v = Pre[u];
        long k = Match[v];
        Match[u] = v;
        Match[v] = u;
        u = k;
    }
}
```

```
long Bfs( long p ){
    memset( Pre,-1,sizeof(Pre));
    memset( Hash,0,sizeof(Hash));
    long i;
    for( i=1;i<=nV;i++ ) Base[i] = i;
    q[1] = p, Hash[p] = 1;
    for (long head=1, rear=1; head<=rear; head++) {
        long u = q[head];
        for (long e=Last[u]; e!=-1; e=Next[e]) {
            long v = To[e];
            if (Base[u]!=Base[v] and v!=Match[u]) {
                if (v==p or (Match[v]!=-1 and Pre[Match[v]]!=-1)) {
                    long b = Contract(u, v);
                    for( i=1;i<=nV;i++ ) if (Blossom[Base[i]]==1) {
                        Base[i] = b;
                        if (!Hash[i]) {
                            Hash[i] = 1;
                            q[++rear] = i;
                        }
                    }
                } else if (Pre[v]==-1) {
                    Pre[v] = u;
                    if (Match[v]==-1) {
                        Augment(v);
                        return 1;
                    }
                    else {
                        q[++rear] = Match[v];
                        Hash[Match[v]] = 1;
                    }
                }
            }
        }
    }
    return 0;
}

long Edmonds_Blossom( void ){
    long i,Ans = 0;
    memset( Match,-1,sizeof(Match));
    for( i=1;i<=nV;i++ ) if (Match[i] == -1) Ans += Bfs(i);
    return Ans;
}
```

```
int main( void ){
    eI = 0;
    memset( Last,-1,sizeof(Last));

}
```

5.12 Faster Weighted Matching

```
#include <bits/stdtr1c++.h>

#define MAX 1002
#define MAXIMIZE +1
#define MINIMIZE -1

using ll = long long;

#define inf (~0U >> 1)
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;
/* call:
    wm::hungarian(number_of_nodes_on_left,on_right,matrix_of_weights,flag)
   match[i] contains matched right node with i-th left node
*/
namespace wm{ /// hash = 581023
    bool visited[MAX];
    ll U[MAX], V[MAX], P[MAX], way[MAX], minv[MAX], match[MAX],
        ar[MAX][MAX];

    /// n = number of row and m = number of columns in 1 based, flag =
        MAXIMIZE or MINIMIZE
    /// match[i] contains the column to which row i is matched
    ll hungarian(ll n, ll m, ll mat[MAX][MAX], ll flag){
        clr(U), clr(V), clr(P), clr(ar), clr(way);

        for (ll i = 1; i <= n; i++){
            for (ll j = 1; j <= m; j++){
                ar[i][j] = mat[i][j];
                if (flag == MAXIMIZE) ar[i][j] = -ar[i][j];
            }
        }
        if (n > m) m = n;

        ll i, j, a, b, c, d, r, w;
        for (i = 1; i <= n; i++){
            P[0] = i, b = 0;
            for (j = 0; j <= m; j++) minv[j] = inf, visited[j] = false;

            do{
                visited[b] = true;
                a = P[b], d = 0, w = inf;

                for (j = 1; j <= m; j++){
                    if (!visited[j]){
                        r = ar[a][j] - U[a] - V[j];
                        if (r < minv[j]) minv[j] = r, way[j] = b;
                        if (minv[j] < w) w = minv[j], d = j;
                    }
                }

                for (j = 0; j <= m; j++){
                    if (visited[j]) U[P[j]] += w, V[j] -= w;
                    else minv[j] -= w;
                }
                b = d;
            } while (P[b] != 0);

            do{
                d = way[b];
                P[b] = P[d], b = d;
            } while (b != 0);
        }
        for (j = 1; j <= m; j++) match[P[j]] = j;

        return (flag == MINIMIZE) ? -V[0] : V[0];
    }
}
```

5.13 Global Minimum Cut

```
/*Given an undirected graph G = (V, E), we define a cut of G to be a
    partition
```

```
of V into two non-empty sets A and B. Earlier, when we looked at network
flows, we worked with the closely related definition of an s-t cut:
    there, given
a directed graph G = (V, E) with distinguished source and sink nodes s
    and t,
an s-t cut was defined to be a partition of V into sets A and B such that
    s  A
and t    B. Our definition now is slightly different, since the
    underlying graph
is now undirected and there is no source or sink.
This problem can be solved by max-flow. First we remove undirected edges
    and replace
them by two opposite directed edge. Now we fix a node s. Then we consider
    each of
the n nodes as t and run max-flow. The minimum of those values is the
    answer.
This is O(n^3).
*/

struct Stoer_Wagner{
  vector <vl> weights;
  Stoer_Wagner(ll N){
    weights.resize(N,vl(N,0));
  }
  void AddEdge(ll from, ll to, ll cap){
    weights[from][to]+=cap;
    weights[to][from]+=cap;
  }
  pair<ll, vl> GetMinCut() {
    ll N = weights.size();
    vl used(N), cut, best_cut;
    ll best_weight = -1;

    for (ll phase = N-1; phase >= 0; phase--) {
      vl w = weights[0];
      vl added = used;
      ll prev, last = 0;
      for (ll i = 0; i < phase; i++) {
        prev = last;
        last = -1;
        for (ll j = 1; j < N; j++)
    if (!added[j] && (last == -1 || w[j] > w[last])) last = j;
        if (i == phase-1) {
          for (ll j = 0; j < N; j++) weights[prev][j] += weights[last][j];
          for (ll j = 0; j < N; j++) weights[j][prev] = weights[prev][j];
          used[last] = true;
          cut.push_back(last);
          if (best_weight == -1 || w[last] < best_weight) {
            best_cut = cut;
            best_weight = w[last];
          }
        } else {
          for (ll j = 0; j < N; j++)
            w[j] += weights[last][j];
          added[last] = true;
        }
      }
    }
    return make_pair(best_weight, best_cut);
  }
};

int main() {
  ll T;
  sl(T);
  f(t,1,T+1){
    ll N,M;
    sll(N,M);
    Stoer_Wagner SW(N);
    f(i,0,M){
      ll a,b,c;
      slll(a,b,c);
      SW.AddEdge(a-1,b-1,c);
    }
    pf("Case #%lld: ",t); pfl(SW.GetMinCut().x);
  }
}
```

5.14 Hopcroft Karp

```
vector< int > graph[MAX];
int n, m, match[MAX], dist[MAX];
int NIL=0;

bool bfs()
{
    int i, u, v, len;
    queue< int > Q;
```

```
    for(i=1; i<=n; i++)
    {
        if(match[i]==NIL)
        {
            dist[i] = 0;
            Q.push(i);
        }
        else dist[i] = INF;
    }
    dist[NIL] = INF;
    while(!Q.empty())
    {
        u = Q.front(); Q.pop();
        if(u!=NIL)
        {
            len = graph[u].size();
            for(i=0; i<len; i++)
            {
                v = graph[u][i];
                if(dist[match[v]]==INF)
                {
                    dist[match[v]] = dist[u] + 1;
                    Q.push(match[v]);
                }
            }
        }
    }
    return (dist[NIL]!=INF);
}

bool dfs(int u)
{
    int i, v, len;
    if(u!=NIL)
    {
        len = graph[u].size();
        for(i=0; i<len; i++)
        {
            v = graph[u][i];
            if(dist[match[v]]==dist[u]+1)
            {
                if(dfs(match[v]))
                {
                    match[v] = u;
                    match[u] = v;
                    return true;
                }
            }
        }
        dist[u] = INF;
        return false;
    }
    return true;
}

int hopcroft_karp()
{
    int matching = 0, i;
    // match[] is assumed NIL for all vertex in graph
    // All nodes on left and right should be distinct
    while(bfs())
        for(i=1; i<=n; i++)
            if(match[i]==NIL && dfs(i))
                matching++;
    return matching;
}

void clear()
{
    FOR(j,0,MAX) graph[j].clear();
    ms(match,NIL);
}

int main()
{
        // ios_base::sync_with_stdio(0);
        // cin.tie(NULL); cout.tie(NULL);
        // freopen("in.txt","r",stdin);

    // SPOJ - Fast Maximum Matching

        int p, x, y;

        scanf("%d%d%d", &n, &m, &p);

        FOR(i,0,p)
        {
                scanf("%d%d", &x, &y);
                graph[x].pb(n+y);
                graph[n+y].pb(x);
```

```
    }

    printf("%d\n", hopcroft_karp());

    return 0;
}
```

5.15 Hungarian Weighted Matching

```
// hungarian weighted matching algo
// finds the max cost of max matching, to find mincost, add edges as
    negatives
// Nodes are indexed from 1 on both sides
template<typename T>
struct KuhnMunkras { // n for left, m for right
  int n, m, match[maxM];
  T g[maxN][maxM], lx[maxN], ly[maxM], slack[maxM];
  bool vx[maxN], vy[maxM];

  void init(int n_, int m_) {
    ms(g,0); n = n_, m = m_;
  }

  void add(int u, int v, T w) {
    g[u][v] = w;
  }

  bool find(int x) {
    vx[x] = true;
    for (int y = 1; y <= m; ++y) {
      if (!vy[y]) {
        T delta = lx[x] + ly[y] - g[x][y];
        if (delta==0) {
          vy[y] = true;
          if (match[y] == 0 || find(match[y])) {
            match[y] = x;
            return true;
          }
        } else slack[y] = min(slack[y], delta);
      }
    }
    return false;
  }

  T matching() { // maximum weight matching
    fill(lx + 1, lx + 1 + n, numeric_limits<T>::lowest());
    ms(ly,0);
    ms(match,0);
    for (int i = 1; i <= n; ++i) {
      for (int j = 1; j <= m; ++j) lx[i] = max(lx[i], g[i][j]);
    }
    for (int k = 1; k <= n; ++k) {
      fill(slack + 1, slack + 1 + m, numeric_limits<T>::max());
      while (true) {
        ms(vx,0);
        ms(vy,0);
        if (find(k)) break;
        else {
          T delta = numeric_limits<T>::max();
          for (int i = 1; i <= m; ++i) {
            if (!vy[i]) delta = min(delta, slack[i]);
          }
          for (int i = 1; i <= n; ++i) {
            if (vx[i]) lx[i] -= delta;
          }
          for (int i = 1; i <= m; ++i) {
            if (vy[i]) ly[i] += delta;
            if (!vy[i]) slack[i] -= delta;
          }
        }
      }
    }
    T result = 0;
    for (int i = 1; i <= n; ++i) result += lx[i];
    for (int i = 1; i <= m; ++i) result += ly[i];
    return result;
  }
};
```

5.16 Johnson's Algorithm

```
/// Johnson's algorithm for all pair shortest paths in sparse graphs
/// Complexity: O(N * M) + O(N * M * log(N))

const long long INF = (1LL << 60) - 666;
```

```cpp
struct edge{
    int u, v;
    long long w;
    edge(){}
    edge(int u, int v, long long w) : u(u), v(v), w(w){}

    void print(){
        cout << "edge " << u << " " << v << " " << w << endl;
    }
};

bool bellman_ford(int n, int src, vector <struct edge> E, vector <long
     long>& dis){
    dis[src] = 0;
    for (int i = 0; i <= n; i++){
        int flag = 0;
        for (auto e: E){
            if ((dis[e.u] + e.w) < dis[e.v]){
                flag = 1;
                dis[e.v] = dis[e.u] + e.w;
            }
        }
        if (flag == 0) return true;
    }
    return false;
}

vector <long long> dijkstra(int n, int src, vector <struct edge> E,
     vector <long long> potential){
    set<pair<long long, int> > S;
    vector <long long> dis(n + 1, INF);
    vector <long long> temp(n + 1, INF);
    vector <pair<int, long long> > adj[n + 1];

    dis[src] = temp[src] = 0;
    S.insert(make_pair(temp[src], src));
    for (auto e: E){
        adj[e.u].push_back(make_pair(e.v, e.w));
    }

    while (!S.empty()){
        pair<long long, int> cur = *(S.begin());
        S.erase(cur);

        int u = cur.second;

        for (int i = 0; i < adj[u].size(); i++){
            int v = adj[u][i].first;
            long long w = adj[u][i].second;

            if ((temp[u] + w) < temp[v]){
                S.erase(make_pair(temp[v], v));
                temp[v] = temp[u] + w;
                dis[v] = dis[u] + w;
                S.insert(make_pair(temp[v], v));
            }
        }
    }
    return dis;
}

void johnson(int n, long long ar[MAX][MAX], vector <struct edge> E){
    vector <long long> potential(n + 1, INF);
    for (int i = 1; i <= n; i++) E.push_back(edge(0, i, 0));

    assert(bellman_ford(n, 0, E, potential));
    for (int i = 1; i <= n; i++) E.pop_back();

    for (int i = 1; i <= n; i++){
        vector <long long> dis = dijkstra(n, i, E, potential);
        for (int j = 1; j <= n; j++){
            ar[i][j] = dis[j];
        }
    }
}

long long ar[MAX][MAX];

int main(){
    vector <struct edge> E;
    E.push_back(edge(1, 2, 2));
    E.push_back(edge(2, 3, -15));
    E.push_back(edge(1, 3, -10));

    int n = 3;
    johnson(n, ar, E);
    for (int i = 1; i <= n; i++){
        for (int j = 1; j <= n; j++){
            printf("%d %d = %lld\n", i, j, ar[i][j]);
        }
    }
```

```
    return 0;
}
```

5.17 Kruskal

```
struct edge
{
	int u, v, w;
	bool operator < (const edge & p) const
	{
		return w < p.w;
	}
};
edge get;
int parent[100];
vector <edge> e;
int find(int r)
{
	if (parent[r] == r)
		return r;
	return parent[r] = find(parent[r]);
}
int mst(int n)
{
	sort(e.begin(), e.end());
	for (int i = 1; i <= n; i++)
		parent[i] = i;
	int cnt = 0, s = 0;
	for (int i = 0; i < (int)e.size(); i++)
	{
		int u = find(e[i].u);
		int v = find(e[i].v);
		if (u != v)
		{
			parent[u] = v;
			cnt++;
			s += e[i].w;
			if (cnt == n - 1)
				break;
		}
	}
}
```

5.18 LCA 2

```
int n, lef[MAX], rig[MAX], dist[MAX], table[2 * MAX][18];
vi graph[MAX], stk;

void dfs(int u, int p, int d)
{
	dist[u] = d;
	lef[u] = rig[u] = stk.size();
	stk.pb(u);
	for (auto v : graph[u])
	{
		if (v == p) continue;
		dfs(v, u, d + 1);
		rig[u] = stk.size();
		stk.pb(u);
	}
}

int lca(int u, int v)
{
	int l = min(lef[u], lef[v]);
	int r = max(rig[u], rig[v]);
	int g = __builtin_clz(r - l + 1) ^ 31;
	return dist[table[l][g]] < dist[table[r - (1 << g) + 1][g]] ?
	      table[l][g] : table[r - (1 << g) + 1][g];
}

void build()
{
	dfs(1, -1, 0);

	for (int i = 0; i < stk.size(); i++) table[i][0] = stk[i];
	for (int j = 1; (1 << j) <= stk.size(); j++)
	{
		for (int i = 0; i + (1 << j) <= stk.size(); i++)
		{
			table[i][j] = (dist[table[i][j - 1]] < dist[table[i
			    + (1 << (j - 1))][j - 1]] ?
			        table[i][j - 1] : table[i + (1 << (j -
			            1))][j - 1]);
		}
	}
}
```

5.19 LCA

```
vi graph[100];
int P[100], L[100], table[100][20];

void dfs(int from, int to, int depth)
{
	P[to]=from;
	L[to]=depth;
	FOR(i,0,(int)graph[to].size())
	{
		int v=graph[to][i];
		if(v==from)
			continue;
		dfs(to,v,depth+1);
	}
}

int query(int n, int p, int q)
{
	if(L[p]<L[q]) swap(p,q);

	int x=1;

	while(true)
	{
		if((1<<(x+1))>L[p])
			break;
		x++;
	}

	FORr(i,x,0)
	{
		if(L[p]-(1<<i) >= L[q])
			p=table[p][i];
	}

	if(p==q) return p;

	FORr(i,x,0)
	{
		if(table[p][i]!=-1 && table[p][i]!=table[q][i])
		{
			p=table[p][i];
			q=table[q][i];
		}
	}

	return P[p];
}


void build(int n)
{
	ms(table,-1);

	FOR(i,0,n)
		table[i][0]=P[i];

	for(int j=1; 1<<j < n; j++)
	{
		for(int i=0; i<n; i++)
		{
			if(table[i][j-1]!=-1)
				table[i][j]=table[table[i][j-1]][j-1];
		}
	}
}
```

5.20 Manhattan MST

```
int n;
vi graph[MAX], cost[MAX];

struct point {
	int x, y, index;
	bool operator<(const point &p) const { return x == p.x ? y < p.y :
	    x < p.x; }
} p[MAX];

struct node {
	int value, p;
} T[MAX];

struct UnionFind {
	int p[MAX];
	void init(int n) { for (int i = 1; i <= n; i++) p[i] = i; }
	int find(int u) { return p[u] == u ? u : p[u] = find(p[u]); }
```

```
        void Union(int u, int v) { p[find(u)] = find(v); }
} dsu;

struct edge {
        int u, v, c;
        bool operator < (const edge &p) const {
                return c < p.c;
        }
};
vector<edge> edges;

int query(int x) {
        int r = inf, p = -1;
        for (; x <= n; x += (x & -x)) if (T[x].value < r) r = T[x].value,
            p = T[x].p;
        return p;
}

void modify(int x, int w, int p) {
        for (; x > 0; x -= (x & -x)) if (T[x].value > w) T[x].value = w,
            T[x].p = p;
}

int dist(point &a, point &b) {
        return abs(a.x - b.x) + abs(a.y - b.y);
}

void add(int u, int v, int c) {
        edges.pb({u, v, c});
}

void kruskal() {
        dsu.init(n);
        SORT(edges);
        for (edge e : edges) {
                int u = e.u, v = e.v, c = e.c;
                // cout<<u<<" "<<v<<" "<<c<<endl;
                if (dsu.find(u) != dsu.find(v)) {
                        graph[u].push_back(v);
                        graph[v].push_back(u);
                        cost[u].push_back(c);
                        cost[v].push_back(c);
                        dsu.Union(u, v);
                }
        }
}

int manhattan() {
        for (int i = 1; i <= n; ++i) p[i].index = i;
        for (int dir = 1; dir <= 4; ++dir) {
                if (dir == 2 || dir == 4) {
                        for (int i = 1; i <= n; ++i) swap(p[i].x, p[i].y);
                } else if (dir == 3) {
                        for (int i = 1; i <= n; ++i) p[i].x = -p[i].x;
                }
                sort(p + 1, p + 1 + n);
                vector<int> v; static int a[MAX];
                for (int i = 1; i <= n; ++i) a[i] = p[i].y - p[i].x,
                    v.push_back(a[i]);
                sort(v.begin(), v.end());
                v.erase(unique(v.begin(), v.end()), v.end());
                for (int i = 1; i <= n; ++i) a[i] = lower_bound(v.begin(),
                    v.end(), a[i]) - v.begin() + 1;
                for (int i = 1; i <= n; ++i) T[i].value = inf, T[i].p = -1;
                for (int i = n; i >= 1; --i) {
                        int pos = query(a[i]);
                        if (pos != -1) add(p[i].index, p[pos].index,
                            dist(p[i], p[pos]));
                        modify(a[i], p[i].x + p[i].y, i);
                }
        }
}

int main()
{
        int test, cases = 1;

        scanf("%d", &n);

        // points
        FOR(i,1,n+1)
        {
                scanf("%d%d", &p[i].x, &p[i].y);
        }

        manhattan();
        kruskal();

        // graph = manhattan mst adjacency list
        // cost = corresponding cost of edges
```

```
	return 0;
}
```

5.21 Max Flow Dinic 2

```
//
// Dinic's maximum flow
//
// Description:
//   Given a directed network G = (V, E) with edge capacity c: E->R.
//   The algorithm finds a maximum flow.
//
// Algorithm:
//   Dinic's blocking flow algorithm.
//
// Complexity:
//   O(n^2 m), but very fast in practice.
//   In particular, for a unit capacity graph,
//   it runs in O(m min{m^{1/2}, n^{2/3}}).
//
// Verified:
//   SPOJ FASTFLOW
//
// Reference:
//   E. A. Dinic (1970):
//   Algorithm for solution of a problem of maximum flow in networks with
    power estimation.
//   Soviet Mathematics Doklady, vol. 11, pp. 1277-1280.
//
//   B. H. Korte and J. Vygen (2008):
//   Combinatorial Optimization: Theory and Algorithms.
//   Springer Berlin Heidelberg.
//

#include <iostream>
#include <vector>
#include <cstdio>
#include <queue>
#include <algorithm>
#include <functional>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())

const long long INF = (1ll << 50);
struct graph {
	typedef long long flow_type;
	struct edge {
		int src, dst;
		flow_type capacity, flow;
		size_t rev;
	};
	int n;
	vector<vector<edge>> adj;
	graph(int n) : n(n), adj(n) { }
	void add_edge(int src, int dst, flow_type capacity) {
		adj[src].push_back({src, dst, capacity, 0,
		    adj[dst].size()});
		adj[dst].push_back({dst, src, 0, 0, adj[src].size() - 1});
	}
	flow_type max_flow(int s, int t) {
		vector<int> level(n), iter(n);
		function<int(void)> levelize = [&]() { // foward levelize
			level.assign(n, -1); level[s] = 0;
			queue<int> Q; Q.push(s);
			while (!Q.empty()) {
				int u = Q.front(); Q.pop();
				if (u == t) break;
				for (auto &e : adj[u]) {
					if (e.capacity > e.flow &&
					    level[e.dst] < 0) {
						Q.push(e.dst);
						level[e.dst] = level[u] + 1;
					}
				}
			}
			return level[t];
		};
		function<flow_type(int, flow_type)> augment = [&](int u,
		    flow_type cur) {
			if (u == t) return cur;
			for (int &i = iter[u]; i < adj[u].size(); ++i) {
				edge &e = adj[u][i], &r = adj[e.dst][e.rev];
```

```
                        if (e.capacity > e.flow && level[u] <
                            level[e.dst]) {
                                flow_type f = augment(e.dst, min(cur,
                                    e.capacity - e.flow));
                                if (f > 0) {
                                        e.flow += f;
                                        r.flow -= f;
                                        return f;
                                }
                        }
                }
                return flow_type(0);
        };
        for (int u = 0; u < n; ++u) // initialize
                for (auto &e : adj[u]) e.flow = 0;

        flow_type flow = 0;
        while (levelize() >= 0) {
                fill(all(iter), 0);
                for (flow_type f; (f = augment(s, INF)) > 0; )
                        flow += f;
        }
        return flow;
    }
};

int main() {
    for (int n, m; scanf("%d %d", &n, &m) == 2; ) {
        graph g(n);
        for (int i = 0; i < m; ++i) {
            int u, v, w;
            scanf("%d %d %d", &u, &v, &w);
            //g.add_edge(u, v, w);
            g.add_edge(u - 1, v - 1, w);
        }
        printf("%lld\n", g.max_flow(0, n - 1));
    }
}
```

5.22 Max Flow Dinic

```
/*
Add s', t', s, t to the graph. For edges with lower bound L and upper
bound R, replace the edge with capacity R-L.
Let, sum[u] = (sum of lowerbounds of ingoing edges to u) - (sum of
    lowerbounds of
                            outgoing edges from u),
here u can be all nodes of the graph, including s and t. For all such u,
    if sum[u]>0
add edge (s',u,sum[u]), add sum[u] to a value 'total', otherwise add edge
    (u,t',-sum[u]).
Lastly add (t,s,INF). Then run max-flow from s' to t'.
A feasible flow won't exist if flow from s' to t' < total, otherwise
if we run a maxflow from s to t (not s' to t'), we get the max-flow
    satisfying the bounds.
***
To find the minimal flow satisfying the bounds, we do a binary search on
    the
capacity of the edge (t,s,INF). Each time during binary search, we check
    if a feasible
flow exists or not with current capacity of (t,s,INF) edge.
*/

struct Edge
{
        int to, rev, f, cap;
};


class Dinic
{
public:

        int dist[MAX], q[MAX], work[MAX], src, dest;
        vector<Edge> graph[MAX];
        // MAX equals to node_number

        void init(int sz)
        {
                FOR(i,0,sz+1) graph[i].clear();
        }

        void clearFlow(int sz)
        {
                FOR(i,0,sz+1)
                {
                        FOR(j,0,graph[i].size())
                                graph[i][j].f=0;
```

```
        }
}

void addEdge(int s, int t, int cap)
{
        Edge a={t,(int)graph[t].size(),0,cap};
        Edge b={s,(int)graph[s].size(),0,0};

        // If our graph has bidirectional edges
        // Capacity for the Edge b will equal to cap
        // For directed, it is 0

        graph[s].emplace_back(a);
        graph[t].emplace_back(b);
}

bool bfs()
{
        ms(dist,-1);
        dist[src]=0;
        int qt=0;
        q[qt++]=src;

        for(int qh=0; qh<qt; qh++)
        {
                int u=q[qh];

                for(auto &e: graph[u])
                {
                        int v=e.to;

                        if(dist[v]<0 && e.f<e.cap)
                        {
                                dist[v]=dist[u]+1;
                                q[qt++]=v;
                        }
                }
        }

        return dist[dest]>=0;
}

int dfs(int u, int f)
{
        if(u==dest) return f;

                for(int &i=work[u]; i<(int)graph[u].size(); i++)
                {
                        Edge &e=graph[u][i];

                        if(e.cap<=e.f) continue;

                        int v=e.to;

                        if(dist[v]==dist[u]+1)
                        {
                                int df=dfs(v,min(f,e.cap-e.f));

                                if(df>0)
                                {
                                        e.f+=df;
                                        graph[v][e.rev].f-=df;

                                        return df;
                                }
                        }
                }

                return 0;
        }

        int maxFlow(int _src, int _dest)
        {
                src=_src;
                dest=_dest;

                int result=0;

                while(bfs())
                {
                        // debug;
                        fill(work,work+MAX,0);
                        while(int delta=dfs(src,INF))
                                result+=delta;
                }

                return result;
        }
};
```

5.23 Max Flow Edmond Karp

```
//
// Maximum Flow (Edmonds-Karp)
//
// Description:
//   Given a directed network G = (V, E) with edge capacity c: E->R.
//   The algorithm finds a maximum flow.
//
// Algorithm:
//   Edmonds-Karp shortest augmenting path algorithm.
//
// Complexity:
//   O(n m^2)
//
// Verified:
//   AOJ GRL_6_A: Maximum Flow
//
// Reference:
//   B. H. Korte and J. Vygen (2008):
//   Combinatorial Optimization: Theory and Algorithms.
//   Springer Berlin Heidelberg.
//

#include <iostream>
#include <vector>
#include <queue>
#include <cstdio>
#include <algorithm>
#include <functional>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())

const int INF = 1 << 30;
struct graph {
	int n;
	struct edge {
		int src, dst;
		int capacity, residue;
		size_t rev;
	};
	edge &rev(edge e) { return adj[e.dst][e.rev]; };

	vector<vector<edge>> adj;
	graph(int n) : n(n), adj(n) { }
	void add_edge(int src, int dst, int capacity) {
		adj[src].push_back({src, dst, capacity, 0,
		    adj[dst].size()});
		adj[dst].push_back({dst, src, 0, 0, adj[src].size() - 1});
	}
	int max_flow(int s, int t) {
		for (int u = 0; u < n; ++u)
			for (auto &e : adj[u]) e.residue = e.capacity;
		int total = 0;
		while (1) {
			vector<int> prev(n, -1); prev[s] = -2;
			queue<int> que; que.push(s);
			while (!que.empty() && prev[t] == -1) {
				int u = que.front(); que.pop();
				for (edge &e : adj[u]) {
					if (prev[e.dst] == -1 && e.residue >
					    0) {
						prev[e.dst] = e.rev;
						que.push(e.dst);
					}
				}
			}
			if (prev[t] == -1) break;
			int inc = INF;
			for (int u = t; u != s; u = adj[u][prev[u]].dst)
				inc = min(inc, rev(adj[u][prev[u]]).residue);
			for (int u = t; u != s; u = adj[u][prev[u]].dst) {
				adj[u][prev[u]].residue += inc;
				rev(adj[u][prev[u]]).residue -= inc;
			}
			total += inc;
		} // { u : visited[u] == true } is s-side
		return total;
	}
};

int main() {
	for (int n, m; scanf("%d %d", &n, &m) == 2; ) {
		graph g(n);
		for (int i = 0; i < m; ++i) {
			int u, v, w;
```

```
                scanf("%d %d %d", &u, &v, &w);
                g.add_edge(u, v, w);
            }
            printf("%d\n", g.max_flow(0, n - 1));
        }
}
```

5.24 Max Flow Ford Fulkerson

```
//
// Ford-Fulkerson's maximum flow
//
// Description:
//   Given a directed network G = (V, E) with edge capacity c: E->R.
//   The algorithm finds a maximum flow.
//
// Algorithm:
//   Ford-Fulkerson's augmenting path algorithm
//
// Complexity:
//   O(m F), where F is the maximum flow value.
//
// Verified:
//   AOJ GRL_6_A: Maximum Flow
//
// Reference:
//   B. H. Korte and J. Vygen (2008):
//   Combinatorial Optimization: Theory and Algorithms.
//   Springer Berlin Heidelberg.
//

#include <iostream>
#include <vector>
#include <cstdio>
#include <algorithm>
#include <functional>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())

const int INF = 1 << 30;
struct graph {
        typedef long long flow_type;
        struct edge {
                int src, dst;
                flow_type capacity, flow;
                size_t rev;
        };
        int n;
        vector<vector<edge>> adj;
        graph(int n) : n(n), adj(n) { }
        void add_edge(int src, int dst, flow_type capacity) {
                adj[src].push_back({src, dst, capacity, 0,
                    adj[dst].size()});
                adj[dst].push_back({dst, src, 0, 0, adj[src].size() - 1});
        }
        int max_flow(int s, int t) {
                vector<bool> visited(n);
                function<flow_type(int, flow_type)> augment = [&](int u,
                    flow_type cur) {
                        if (u == t) return cur;
                        visited[u] = true;
                        for (auto &e : adj[u]) {
                                if (!visited[e.dst] && e.capacity > e.flow) {
                                        flow_type f = augment(e.dst,
                                            min(e.capacity - e.flow, cur));
                                        if (f > 0) {
                                                e.flow += f;
                                                adj[e.dst][e.rev].flow -= f;
                                                return f;
                                        }
                                }
                        }
                        return flow_type(0);
                };
                for (int u = 0; u < n; ++u)
                        for (auto &e : adj[u]) e.flow = 0;

                flow_type flow = 0;
                while (1) {
                        fill(all(visited), false);
                        flow_type f = augment(s, INF);
                        if (f == 0) break;
                        flow += f;
                }
```

```
		return flow;
	}
};

int main() {
	for (int n, m; scanf("%d %d", &n, &m) == 2; ) {
		graph g(n);
		for (int i = 0; i < m; ++i) {
			int u, v, w;
			scanf("%d %d %d", &u, &v, &w);
			g.add_edge(u, v, w);
		}
		printf("%d\n", g.max_flow(0, n - 1));
	}
}
```

5.25 Max Flow Goldberg Tarjan

```
//
// Maximum Flow (Goldberg-Tarjan, aka. Push-Relabel, Preflow-Push)
//
// Description:
//   Given a directed network G = (V, E) with edge capacity c: E->R.
//   The algorithm finds a maximum flow.
//
// Algorithm:
//   Goldberg-Tarjan's push-relabel algorithm with gap-heuristics.
//
// Complexity:
//   O(n^3)
//
// Verified:
//   SPOJ FASTFLOW
//
// Reference:
//   B. H. Korte and Jens Vygen (2008):
//   Combinatorial Optimization: Theory and Algorithms.
//   Springer Berlin Heidelberg.
//

#include <iostream>
#include <vector>
#include <cstdio>
#include <queue>
#include <algorithm>
#include <functional>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())

const long long INF = (1ll << 50);
struct graph {
	typedef long long flow_type;
	struct edge {
		int src, dst;
		flow_type capacity, flow;
		size_t rev;
	};
	int n;
	vector<vector<edge>> adj;
	graph(int n) : n(n), adj(n) { }

	void add_edge(int src, int dst, int capacity) {
		adj[src].push_back({src, dst, capacity, 0,
		    adj[dst].size()});
		adj[dst].push_back({dst, src, 0, 0, adj[src].size() - 1});
	}

	flow_type max_flow(int s, int t) {
		vector<flow_type> excess(n);
		vector<int> dist(n), active(n), count(2 * n);
		queue<int> Q;
		auto enqueue = [&](int v) {
			if (!active[v] && excess[v] > 0) { active[v] =
			    true; Q.push(v); }
		};
		auto push = [&](edge & e) {
			flow_type f = min(excess[e.src], e.capacity -
			    e.flow);
			if (dist[e.src] <= dist[e.dst] || f == 0) return;
			e.flow += f;
			adj[e.dst][e.rev].flow -= f;
			excess[e.dst] += f;
			excess[e.src] -= f;
			enqueue(e.dst);
```

```
        };

        dist[s] = n; active[s] = active[t] = true;
        count[0] = n - 1; count[n] = 1;
        for (int u = 0; u < n; ++u)
                for (auto &e : adj[u]) e.flow = 0;
        for (auto &e : adj[s]) {
                excess[s] += e.capacity;
                push(e);
        }
        while (!Q.empty()) {
                int u = Q.front(); Q.pop();
                active[u] = false;

                for (auto &e : adj[u]) push(e);
                if (excess[u] > 0) {
                        if (count[dist[u]] == 1) {
                                int k = dist[u]; // Gap Heuristics
                                for (int v = 0; v < n; v++) {
                                        if (dist[v] < k) continue;
                                        count[dist[v]]--;
                                        dist[v] = max(dist[v], n + 1);
                                        count[dist[v]]++;
                                        enqueue(v);
                                }
                        } else {
                                count[dist[u]]--; // Relabel
                                dist[u] = 2 * n;
                                for (auto &e : adj[u])
                                        if (e.capacity > e.flow)
                                                dist[u] = min(dist[u],
                                                    dist[e.dst] + 1);
                                count[dist[u]]++;
                                enqueue(u);
                        }
                }
        }

        flow_type flow = 0;
        for (auto e : adj[s]) flow += e.flow;
        return flow;
    }
};

int main() {
        for (int n, m; scanf("%d %d", &n, &m) == 2; ) {
                graph g(n);
                for (int i = 0; i < m; ++i) {
                        int u, v, w;
                        scanf("%d %d %d", &u, &v, &w);
                        g.add_edge(u, v, w);
                }
                printf("%d\n", g.max_flow(0, n - 1));
        }
}
```

5.26 Maximum Bipartite Matching and Min Vertex Cover

```
int n, m, p; // n = # of nodes on left, m = # of nodes on right
vi bp[N]; // bipartite graph
int matched[N], revmatch[N];
bool seen[N], visited[2][N];

bool trymatch(int u)
{
        FOR(j,0,bp[u].size())
        {
                int v=bp[u][j];
                if(seen[v]) continue;

                seen[v]=true;

                // v is on right, u on left
                if(matched[v]<0 || trymatch(matched[v]))
                {
                        matched[v]=u;
                        revmatch[u]=v;
                        return true;
                }
        }

        return false;
}

// 0 based
int maxbpm(int sz)
{
        ms(matched,-1);
```

```
	ms(revmatch,-1); // for min-vertex-cover

	int ret=0;

	FOR(i,0,sz)
	{
		ms(seen,false);
		if(trymatch(i)) ret++;
	}

	return ret;
}

void dfsLast(int u, bool side)
{
	if(visited[side][u]) return;
	visited[side][u]=true;

	if(!side)
	{
		for(int i=0; i<n; i++)
		{
			if(graph[u][i] && matched[u]!=i)
				dfsLast(i,1-side);
		}
	}
	else dfsLast(matched[u],1-side);
}

void findMinVertexCover()
{
	FOR(i,0,n)
	{
		if(revmatch[i]==-1)
		{
			dfsLast(i,0);
		}
	}
	// Assuming both sides have n nodes
	vi mvc, mis; // min vertex cover, max independent set
	FOR(i,0,n)
	{
		if(!visited[0][i] || visited[1][i]) mvc.pb(i);
		if(!(!visited[0][i] || visited[1][i])) mis.pb(i);
	}
}
// The following probably optimizes for large graphs
bool trymatch(int u)
{
	// tag is used so that we don't clear seen each time
	if(seen[u]==tag) return false;
	seen[u]=tag;
	FOR(j,0,bp[u].size())
	{
		int v=bp[u][j];
		// first we only consider any matched[v]==-1 case
		if(matched[v]<0)
		{
			matched[v]=u;
			return true;
		}
	}
	FOR(j,0,bp[u].size())
	{
		int v=bp[u][j];
		// Now we go deeper and call trymatch
		if(trymatch(matched[v]))
		{
			matched[v]=u;
			return true;
		}
	}
	return false;
}
```

5.27 Maximum Matching in General Graphs (Randomized Algorithm)

```
#include <time.h>
#define MAX 1010
bool adj[MAX][MAX];
int n, ar[MAX][MAX];
const int MOD = 1073750017;
int expo(long long x, int n){
    long long res = 1;

    while (n){
        if (n & 1) res = (res * x) % MOD;
```

```
        x = (x * x) % MOD;
        n >>= 1;
    }

    return (res % MOD);
}
int rank(int n){ /// hash = 646599
    long long inv;
    int i, j, k, u, v, x, r = 0, T[MAX];

    for (j = 0; j < n; j++){
        for (k = r; k < n && !ar[k][j]; k++){}
        if (k == n) continue;

        inv = expo(ar[k][j], MOD - 2);
        for (i = 0; i < n; i++){
            x = ar[k][i];
            ar[k][i] = ar[r][i];
            ar[r][i] = (inv * x) % MOD;
        }

        for (u = r + 1; u < n; u++){
            if (ar[u][j]){
                for (v = j + 1; v < n; v++){
                    if (ar[r][v]){
                        ar[u][v] = ar[u][v] - (((long long)ar[r][v] *
                            ar[u][j]) % MOD);
                        if (ar[u][v] < 0) ar[u][v] += MOD;
                    }
                }
            }
        }
        r++;
    }

    return r;
}
int tutte(int n){
    int i, j;
    srand(time(0));

    clr(ar);
    for (i = 0; i < n; i++){
        for (j = i + 1; j < n; j++){
            if (adj[i][j]){
                unsigned int x = (rand() << 15) ^ rand();
                x = (x % (MOD - 1)) + 1;
                ar[i][j] = x, ar[j][i] = MOD - x;
            }
        }
    }

    return (rank(n) >> 1);
}
int main(){
    int T = 0, t, m, i, j, a, b;

    scanf("%d", &t);
    while (t--){
        clr(adj);
        scanf("%d %d", &n, &m);
        while (m--){
            scanf("%d %d", &a, &b);
            a--, b--;
            adj[a][b] = adj[b][a] = true;
        }

        printf("Case %d: %d\n", ++T, tutte(n));
    }
    return 0;
}
```

5.28 Min Cost Arborescence

```
// Min Cost Arboroscense class in C++
// Directed MST
// dir_mst returns the cost O(EV)?

struct Edge {
    int u, v;
    ll dist;
    int kbps;
};

struct MinCostArborescence{
    int n, m;
    Edge allEdges[MAX];
    int done[62], prev[62], id[62];
```

```
ll in[62];

void init(int n)
{
    this->n = n;
    m = 0;
}

void add_Edge(int u, int v, ll dist)
{
    allEdges[m++] = {u,v,dist,0};
}

void add_Edge(Edge e)
{
    allEdges[m++] = e;
}

ll dir_mst(int root) {
    ll ans = 0;
    while (true) {
        for (int i = 0; i < n; i++) in[i] = INF;
        for (int i = 0; i < m; i++) {
            int u = allEdges[i].u;
            int v = allEdges[i].v;
            if (allEdges[i].dist < in[v] && u != v) {
                in[v] = allEdges[i].dist;
                prev[v] = u;
            }
        }

        for (int i = 0; i < n; i++) {
            if (i == root) continue;
            if (in[i] == INF) return -1;
        }

        int cnt = 0;
        memset(id, -1, sizeof(id));
        memset(done, -1, sizeof(done));
        in[root] = 0;

        for (int i = 0; i < n; i++)
        {
            ans += in[i];
            int v = i;
            while (done[v] != i && id[v] == -1 && v != root) {
                done[v] = i;
                v = prev[v];
            }
            if (v != root && id[v] == -1) {
                for (int u = prev[v]; u != v; u = prev[u])
                    id[u] = cnt;
                id[v] = cnt++;
            }
        }
        if (cnt == 0) break;
        for (int i = 0; i < n; i++)
            if (id[i] == -1) id[i] = cnt++;
        for (int i = 0; i < m; i++) {
            int v = allEdges[i].v;
            allEdges[i].u = id[allEdges[i].u];
            allEdges[i].v = id[allEdges[i].v];
            if (allEdges[i].u != allEdges[i].v)
                allEdges[i].dist -= in[v];
        }
        n = cnt;
        root = id[root];
    }
    return ans;
  }
} Arboroscense;
```

5.29 Min Cost Max Flow 1

```
//
// Minimum Cost Maximum Flow (Tomizawa, Edmonds-Karp's successive
    shortest path)
//
// Description:
//  Given a directed graph G = (V,E) with nonnegative capacity c and
    cost w.
//  The algorithm find a maximum s-t flow of G with minimum cost.
//
// Algorithm:
//  Tomizawa (1971), and Edmonds and Karp (1972)'s
//  successive shortest path algorithm,
//  which is also known as the primal-dual method.
```

```
//
// Complexity:
//   O(F m log n), where F is the amount of maximum flow.


// Caution: Probably does not support Negative Costs
// Negative cost is supported in an implementation named:
    mincostmaxflow2.cpp


#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
#define TEST(s) if (!(s)) { cout << __LINE__ << " " << #s << endl;
    exit(-1); }

const long long INF = 1e9;
struct graph {
  typedef int flow_type;
  typedef int cost_type;
  struct edge {
    int src, dst;
    flow_type capacity, flow;
    cost_type cost;
    size_t rev;
  };
  vector<edge> edges;
  void add_edge(int src, int dst, flow_type cap, cost_type cost) {
    adj[src].push_back({src, dst, cap, 0, cost, adj[dst].size()});
    adj[dst].push_back({dst, src, 0, 0, -cost, adj[src].size()-1});
  }
  int n;
  vector<vector<edge>> adj;
  graph(int n) : n(n), adj(n) { }

  pair<flow_type, cost_type> min_cost_max_flow(int s, int t) {
    flow_type flow = 0;
    cost_type cost = 0;

    for (int u = 0; u < n; ++u) // initialize
      for (auto &e: adj[u]) e.flow = 0;

    vector<cost_type> p(n, 0);

    auto rcost = [&](edge e) { return e.cost + p[e.src] - p[e.dst]; };

    for (int iter = 0; ; ++iter) {
      vector<int> prev(n, -1); prev[s] = 0;
      vector<cost_type> dist(n, INF); dist[s] = 0;
      if (iter == 0) { // use Bellman-Ford to remove negative cost edges
        vector<int> count(n); count[s] = 1;
        queue<int> que;
        for (que.push(s); !que.empty(); ) {
          int u = que.front(); que.pop();
          count[u] = -count[u];
          for (auto &e: adj[u]) {
            if (e.capacity > e.flow && dist[e.dst] > dist[e.src] +
                rcost(e)) {
              dist[e.dst] = dist[e.src] + rcost(e);
              prev[e.dst] = e.rev;
              if (count[e.dst] <= 0) {
                count[e.dst] = -count[e.dst] + 1;
                que.push(e.dst);
              }
            }
          }
        }
      } else { // use Dijkstra
        typedef pair<cost_type, int> node;
        priority_queue<node, vector<node>, greater<node>> que;
        que.push({0, s});
        while (!que.empty()) {
          node a = que.top(); que.pop();
          if (a.snd == t) break;
          if (dist[a.snd] > a.fst) continue;
          for (auto e: adj[a.snd]) {
            if (e.capacity > e.flow && dist[e.dst] > a.fst + rcost(e)) {
              dist[e.dst] = dist[e.src] + rcost(e);
              prev[e.dst] = e.rev;
              que.push({dist[e.dst], e.dst});
            }
          }
        }
      }
      if (prev[t] == -1) break;

      for (int u = 0; u < n; ++u)
        if (dist[u] < dist[t]) p[u] += dist[u] - dist[t];

      function<flow_type(int,flow_type)> augment = [&](int u, flow_type
          cur) {
```

```
      if (u == s) return cur;
      edge &r = adj[u][prev[u]], &e = adj[r.dst][r.rev];
      flow_type f = augment(e.src, min(e.capacity - e.flow, cur));
      e.flow += f; r.flow -= f;
      return f;
    };
    flow_type f = augment(t, INF);
    flow += f;
    cost += f * (p[t] - p[s]);
  }
  return {flow, cost};
 }
};
```

5.30 Min Cost Max Flow 2

```
// By zscoder
// From problem: CF Anti Palindromize - 884F
// Thank you ZS.
// Works as max-cost-max-flow if the costs are considered negative
// Slower due to SPFA in some cases?

struct Edge{
    int u, v;
    long long cap, cost;

    Edge(int _u, int _v, long long _cap, long long _cost){
        u = _u; v = _v; cap = _cap; cost = _cost;
    }
};

struct MinCostFlow{
    int n, s, t;
    long long flow, cost;
    vector<vector<int> > graph;
    vector<Edge> e;
    // if cost is double, dist should be double
    vector<long long> dist;
    vector<int> parent;

    MinCostFlow(int _n){
        // 0-based indexing
        n = _n;
        graph.assign(n, vector<int> ());
    }

    void addEdge(int u, int v, long long cap, long long cost, bool
         directed = true){
        graph[u].push_back(e.size());
        e.push_back(Edge(u, v, cap, cost));

        graph[v].push_back(e.size());
        e.push_back(Edge(v, u, 0, -cost));

        if(!directed)
            addEdge(v, u, cap, cost, true);
    }

    pair<long long, long long> getMinCostFlow(int _s, int _t){
        s = _s; t = _t;
        flow = 0, cost = 0;

        while(SPFA()){
            flow += sendFlow(t, 1LL<<62);
        }

        return make_pair(flow, cost);
    }

    // not sure about negative cycle
    bool SPFA(){
        parent.assign(n, -1);
        dist.assign(n, 1LL<<62);     dist[s] = 0;
        vector<int> queuetime(n, 0); queuetime[s] = 1;
        vector<bool> inqueue(n, 0);  inqueue[s] = true;
        queue<int> q;                q.push(s);
        bool negativecycle = false;


        while(!q.empty() && !negativecycle){
            int u = q.front(); q.pop(); inqueue[u] = false;

            for(int i = 0; i < graph[u].size(); i++){
                int eIdx = graph[u][i];
                int v = e[eIdx].v; ll w = e[eIdx].cost, cap = e[eIdx].cap;

                if(dist[u] + w < dist[v] && cap > 0){
                    dist[v] = dist[u] + w;
```

```
                parent[v] = eIdx;

                if(!inqueue[v]){
                    q.push(v);
                    queuetime[v]++;
                    inqueue[v] = true;

                    if(queuetime[v] == n+2){
                        negativecycle = true;
                        break;
                    }
                }
            }
        }
    }

    return dist[t] != (1LL<<62);
}

long long sendFlow(int v, long long curFlow){
    if(parent[v] == -1)
        return curFlow;
    int eIdx = parent[v];
    int u = e[eIdx].u; ll w = e[eIdx].cost;

    long long f = sendFlow(u, min(curFlow, e[eIdx].cap));

    cost += f*w;
    e[eIdx].cap -= f;
    e[eIdx^1].cap += f;

    return f;
}
};
```

5.31 Min Cost Max Flow 3

```
// This gave AC for CF 813D Two Melodies but the other one was TLE
// By sgtlaugh
// flow[i] contains the amount of flow in i-th edge
namespace mcmf{
    const int MAX = 1000010;
    const int INF = 1 << 25;

int cap[MAX], flow[MAX], cost[MAX], dis[MAX];
int n, m, s, t, Q[10000010], adj[MAX], link[MAX], last[MAX],
    from[MAX], visited[MAX];

void init(int nodes, int source, int sink){
    m = 0, n = nodes, s = source, t = sink;
    for (int i = 0; i <= n; i++) last[i] = -1;
}

void addEdge(int u, int v, int c, int w){
    adj[m] = v, cap[m] = c, flow[m] = 0, cost[m] = +w, link[m] =
        last[u], last[u] = m++;
    adj[m] = u, cap[m] = 0, flow[m] = 0, cost[m] = -w, link[m] =
        last[v], last[v] = m++;
}

bool spfa(){
    int i, j, x, f = 0, l = 0;
    for (i = 0; i <= n; i++) visited[i] = 0, dis[i] = INF;

    dis[s] = 0, Q[l++] = s;
    while (f < l){
        i = Q[f++];
        for (j = last[i]; j != -1; j = link[j]){
            if (flow[j] < cap[j]){
                x = adj[j];
                if (dis[x] > dis[i] + cost[j]){
                    dis[x] = dis[i] + cost[j], from[x] = j;
                    if (!visited[x]){
                        visited[x] = 1;
                        if (f && rand() & 7) Q[--f] = x;
                        else Q[l++] = x;
                    }
                }
            }
        }
        visited[i] = 0;
    }
    return (dis[t] != INF);
}
// we can return all the flow values for each edge from this function
// vi solve()
pair <int, int> solve(){
    int i, j;
```

```
        int mincost = 0, maxflow = 0;

        while (spfa()){
            int aug = INF;
            for (i = t, j = from[i]; i != s; i = adj[j ^ 1], j = from[i]){
                aug = min(aug, cap[j] - flow[j]);
            }
            for (i = t, j = from[i]; i != s; i = adj[j ^ 1], j = from[i]){
                flow[j] += aug, flow[j ^ 1] -= aug;
            }
            maxflow += aug, mincost += aug * dis[t];
        }
        // edges are indexed from 0 to m
        // vi ret(flow,flow+m)
        // to find flow of a specific edge, we just noticed that flow[2*i]
            contains
        // the flow amount in i-th edge
        return make_pair(mincost, maxflow);
    }
}
```

5.32 Min Cost Max Flow with Bellman Ford

```
const int InfCost = 1e9;

struct MinimumCostMaximumFlow {
        typedef int Index; typedef int Flow; typedef int Cost;
        static const Flow InfCapacity = 1e9;
        struct Edge {
                Index to; Index rev;
                Flow capacity; Cost cost;
        };
        vector<vector<Edge> > g;
        void init(Index n) { g.assign(n, vector<Edge>()); }
        void add(Index i, Index j, Flow capacity = InfCapacity, Cost cost
            = Cost()) {
                Edge e, f; e.to = j, f.to = i; e.capacity = capacity,
                f.capacity = 0; e.cost = cost, f.cost = -cost;
                g[i].push_back(e); g[j].push_back(f);
                g[i].back().rev = (Index)g[j].size() - 1; g[j].back().rev
                    = (Index)g[i].size() - 1;
        }
        void addB(Index i, Index j, Flow capacity = InfCapacity, Cost cost
            = Cost()) {
                add(i, j, capacity, cost);
                add(j, i, capacity, cost);
        }
        pair<Cost, Flow> minimumCostMaximumFlow(Index s, Index t, Flow f =
            InfCapacity,
                bool bellmanFord = false) {
                int n = g.size();
                vector<Cost> dist(n); vector<Index> prev(n); vector<Index>
                    prevEdge(n);
                pair<Cost, Flow> total = make_pair(0, 0);
                vector<Cost> potential(n);
                while (f > 0) {
                        fill(dist.begin(), dist.end(), InfCost);
                        if (bellmanFord || total.second == 0) {
                                dist[s] = 0;
                                for (int k = 0; k < n; k++) {
                                        bool update = false;
                                        for (int i = 0; i < n; i++)
                                                if (dist[i] != InfCost)
                                                        for (Index ei = 0; ei
                                                            <
                                                            (Index)g[i].size();
                                                            ei ++) {
                                                                const Edge &e =
                                                                    g[i][ei];
                                                                if (e.capacity
                                                                    <= 0)
                                                                    continue;
                                                                Index j = e.to;
                                                                    Cost d =
                                                                    dist[i] +
                                                                    e.cost;
                                                                if (dist[j] > d
                                                                    ) {
                                                                        dist[j]
                                                                            = d;
                                                                        prev[j]
                                                                            = i;
                                                                        prevEdge[j]
                                                                            = ei;
                                                                        update =
                                                                            true;
                                                                }
```

```
                        }
                if (!update) break;
        }
} else {
        vector<bool> vis(n);
        priority_queue<pair<Cost, Index> > q;
        q.push(make_pair(-0, s)); dist[s] = 0;
        while (!q.empty()) {
                Index i = q.top().second; q.pop();
                if (vis[i]) continue;
                vis[i] = true;
                for (Index ei = 0; ei <
                    (Index)g[i].size(); ei ++) {
                        const Edge &e = g[i][ei];
                        if (e.capacity <= 0) continue;
                        Index j = e.to; Cost d =
                            dist[i] + e.cost +
                            potential[i] -
                            potential[j];
                        if (d < dist[i]) d = dist[i];
                        if (dist[j] > d) {
                                dist[j] = d; prev[j] =
                                    i; prevEdge[j] =
                                    ei;
                                q.push(make_pair(-d,
                                    j));
                        }
                }
        }
}
if (dist[t] == InfCost) break;
if (!bellmanFord) for (Index i = 0; i < n; i ++)
    potential[i] += dist[i];
Flow d = f; Cost distt = 0;
for (Index v = t; v != s; ) {
        Index u = prev[v]; const Edge &e =
            g[u][prevEdge[v]];
        d = min(d, e.capacity); distt += e.cost; v =
            u;
}
f -= d; total.first += d * distt; total.second += d;
for (Index v = t; v != s; v = prev[v]) {
        Edge &e = g[prev[v]][prevEdge[v]];
        e.capacity -= d; g[e.to][e.rev].capacity +=
            d;
                        }
                }
                return total;
        }
} network;
```

5.33 Minimum Path Cover in DAG

```
#include <bits/stdtr1c++.h>

#define MAX 505
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;
/// Minimum path cover/Maximum independent set in DAG
namespace dag{
    /// For transitive closure and minimum path cover with not
        necessarily disjoint vertex
    bool ar[MAX][MAX];

    vector <int> adj[MAX];
    bool visited[MAX], first_set[MAX], second_set[MAX];
    int n, L[MAX], R[MAX], D[MAX], Q[MAX], dis[MAX], parent[MAX];

    inline void init(int nodes){ /// Number of vertices in DAG
        n = nodes;
        for (int i = 0; i < MAX; i++) adj[i].clear();
    }

    inline void add_edge(int u, int v){ /// 0 based index, directed edge
        of DAG
        adj[u].push_back(v);
    }

    bool dfs(int i){
        int len = adj[i].size();
        for (int j = 0; j < len; j++){
            int x = adj[i][j];
            if (L[x] == -1 || (parent[L[x]] == i)){
                if (L[x] == -1 || dfs(L[x])){
```

```
                L[x] = i, R[i] = x;
                return true;
            }
        }
    }
    return false;
}

bool bfs(){
    clr(visited);
    int i, j, x, d, f = 0, l = 0;

    for (i = 0; i < n; i++){
        if (R[i] == -1){
            visited[i] = true;
            Q[l++] = i, dis[i] = 0;
        }
    }

    while (f < l){
        i = Q[f++];
        int len = adj[i].size();
        for (j = 0; j < len; j++){
            x = adj[i][j], d = L[x];
            if (d == -1) return true;

            else if (!visited[d]){
                Q[l++] = d;
                parent[d] = i, visited[d] = true, dis[d] = dis[i] + 1;
            }
        }
    }
    return false;
}

void get_path(int i){
    first_set[i] = true;
    int j, x, len = adj[i].size();

    for (j = 0; j < len; j++){
        x = adj[i][j];
        if (!second_set[x] && L[x] != -1){
            second_set[x] = true;
            get_path(L[x]);
        }
    }
}

void transitive_closure(){ /// Transitive closure in O(n * m)
    clr(ar);
    int i, j, k, l;
    for (i = 0; i < n; i++){
        l = adj[i].size();
        for (j = 0; j < l; j++){
            ar[i][adj[i][j]] = true;
        }
        adj[i].clear();
    }

    for (k = 0; k < n; k++){
        for (i = 0; i < n; i++){
            if (ar[i][k]){
                for (j = 0; j < n; j++){
                    if (ar[k][j]) ar[i][j] = true;
                }
            }
        }
    }

    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            if (i != j && ar[i][j]){
                adj[i].push_back(j);
            }
        }
    }
}
/// Minimum vertex disjoint path cover in DAG. Handle isolated
    vertices appropriately
int minimum_disjoint_path_cover() {
    int i, res = 0;
    memset(L, -1, sizeof(L));
    memset(R, -1, sizeof(R));

    while (bfs()){
        for (i = 0; i < n; i++){
            if (R[i] == -1 && dfs(i)) res++;
        }
    }
```

```
    return n - res;
}

int minimum_path_cover(){ /// Minimum path cover in DAG. Handle
    isolated vertices appropriately
    transitive_closure();
    return minimum_disjoint_path_cover();
}
/// Minimum vertex cover of DAG, equal to maximum bipartite matching
vector <int> minimum_vertex_cover(){
    int i, res = 0;
    memset(L, -1, sizeof(L));
    memset(R, -1, sizeof(R));

    while (bfs()){
        for (i = 0; i < n; i++){
            if (R[i] == -1 && dfs(i)) res++;
        }
    }

    vector <int> v;
    clr(first_set), clr(second_set);
    for (i = 0; i < n; i++){
        if (R[i] == -1) get_path(i);
    }

    for (i = 0; i < n; i++){
        if (!first_set[i] || second_set[i]) v.push_back(i);
    }

    return v;
}
/// Maximum independent set of DAG, all vertices not in minimum
    vertex cover
vector <int> maximum_independent_set() {
    vector <int> v = minimum_vertex_cover();
    clr(visited);
    int i, len = v.size();
    for (i = 0; i < len; i++) visited[v[i]] = true;

    vector <int> res;
    for (i = 0; i < n; i++){
        if (!visited[i]) res.push_back(i);
    }
    return res;
}
```

5.34 Prim MST

```
vector <ll> graph[10003], cost[10003];
bool visited[10003];
ll d[10003];
int n, m;

int minKey()
{
        ll mini=INF;
        int minidx;
        for (int i=1; i<=n; i++)
        {
                if (!visited[i] && d[i]<mini)
                        mini=d[i], minidx=i;
        }
        return minidx;
}

ll Prim()
{
        FOR(i,0,10003)
        {
                d[i]=INF;
                visited[i]=false;
        }
        d[1]=0;
        for (int i=1; i<=n-1; i++)
        {
                int u=minKey();
                visited[u]=true;
                FOR(j,0,graph[u].size())
                {
                        int v=graph[u][j];
                        if(!visited[v] && cost[u][j]<d[v])
                                d[v]=cost[u][j];
                }

        }
        ll ret=0;
```

```
        FOR(j,1,n+1)
        {
                // cout<<d[j]<<endl;
                if(d[j]!=INF)
                        ret+=d[j];
        }
        return ret;
}

int main()
{
        int a, b, c;
        scanf("%d%d", &n, &m);
        FOR(i,0,m)
        {
                scanf("%d%d%d", &a, &b, &c);
                graph[a].pb(b);
                graph[b].pb(a);
                cost[a].pb(c);
                cost[b].pb(c);
        }
        cout<<Prim()<<endl;

    return 0;
}
```

5.35 Push Relabel 2

```
/*
    Implementation of highest-label push-relabel maximum flow
    with gap relabeling heuristic.

    Running time:
        O(|V|^2|E|^{1/2})

    Usage:
        - add edges by AddEdge()
        - GetMaxFlow(s, t) returns the maximum flow from s to t

    Input:
        - graph, constructed using AddEdge()
        - (s, t), (source, sink)

    Output:
        - maximum flow value

    Todo:
        - implement Phase II (flow network from preflow network)
        - implement GetMinCut()
*/

// To obtain the actual flow values, look at all edges with capacity > 0
// Zero capacity edges are residual edges

template <class T> struct Edge {
    int from, to, index;
    T cap, flow;

    Edge(int from, int to, T cap, T flow, int index): from(from), to(to),
        cap(cap), flow(flow), index(index) {}
};

template <class T> struct PushRelabel {
    int n;
    vector <vector <Edge <T>>> adj;
    vector <T> excess;
    vector <int> dist, count;
    vector <bool> active;
    vector <vector <int>> B;
    int b;
    queue <int> Q;

    PushRelabel (int n): n(n), adj(n) {}

    void AddEdge (int from, int to, int cap) {
        adj[from].push_back(Edge <T>(from, to, cap, 0, adj[to].size()));
        if (from == to) {
            adj[from].back().index++;
        }
        adj[to].push_back(Edge <T>(to, from, 0, 0, adj[from].size() - 1));

    }

    void Enqueue (int v) {
        if (!active[v] && excess[v] > 0 && dist[v] < n) {
            active[v] = true;
            B[dist[v]].push_back(v);
            b = max(b, dist[v]);
```

```
    }
}

void Push (Edge <T> &e) {
    T amt = min(excess[e.from], e.cap - e.flow);
    if (dist[e.from] == dist[e.to] + 1 && amt > T(0)) {
        e.flow += amt;
        adj[e.to][e.index].flow -= amt;
        excess[e.to] += amt;
        excess[e.from] -= amt;
        Enqueue(e.to);
    }
}

void Gap (int k) {
    for (int v = 0; v < n; v++) if (dist[v] >= k) {
        count[dist[v]]--;
        dist[v] = max(dist[v], n);
        count[dist[v]]++;
        Enqueue(v);
    }
}

void Relabel (int v) {
    count[dist[v]]--;
    dist[v] = n;
    for (auto e: adj[v]) if (e.cap - e.flow > 0) {
        dist[v] = min(dist[v], dist[e.to] + 1);
    }
    count[dist[v]]++;
    Enqueue(v);
}

void Discharge(int v) {
    for (auto &e: adj[v]) {
        if (excess[v] > 0) {
            Push(e);
        } else {
            break;
        }
    }

    if (excess[v] > 0) {
        if (count[dist[v]] == 1) {
            Gap(dist[v]);
        } else {
            Relabel(v);
        }
    }
}

T GetMaxFlow (int s, int t) {
    dist = vector <int>(n, 0), excess = vector<T>(n, 0), count =
        vector <int>(n + 1, 0), active = vector <bool>(n, false), B =
        vector <vector <int>>(n), b = 0;

    for (auto &e: adj[s]) {
        excess[s] += e.cap;
    }

    count[0] = n;
    Enqueue(s);
    active[t] = true;

    while (b >= 0) {
        if (!B[b].empty()) {
            int v = B[b].back();
            B[b].pop_back();
            active[v] = false;
            Discharge(v);
        } else {
            b--;
        }
    }
    return excess[t];
}

T GetMinCut (int s, int t, vector <int> &cut);
};
```

5.36 Push Relabel

```
#define sz(x) (int)(x).size()

struct Edge {
    int v;
    ll flow, C;
    int rev;
```

```
};

template <int SZ> struct PushRelabel {
    vector<Edge> adj[SZ];
    ll excess[SZ];
    int dist[SZ], count[SZ+1], b = 0;
    bool active[SZ];
    vi B[SZ];

    void addEdge(int u, int v, ll C) {
        Edge a{v, 0, C, sz(adj[v])};
        Edge b{u, 0, 0, sz(adj[u])};
        adj[u].pb(a), adj[v].pb(b);
    }

    void enqueue (int v) {
        if (!active[v] && excess[v] > 0 && dist[v] < SZ) {
            active[v] = 1;
            B[dist[v]].pb(v);
            b = max(b, dist[v]);
        }
    }

    void push (int v, Edge &e) {
        ll amt = min(excess[v], e.C-e.flow);
        if (dist[v] == dist[e.v]+1 && amt > 0) {
            e.flow += amt, adj[e.v][e.rev].flow -= amt;
            excess[e.v] += amt, excess[v] -= amt;
            enqueue(e.v);
        }
    }

    void gap (int k) {
        FOR(v,1,SZ+1) if (dist[v] >= k) {
            count[dist[v]] --;
            dist[v] = SZ;
            count[dist[v]] ++;
            enqueue(v);
        }
    }

    void relabel (int v) {
        count[dist[v]] --; dist[v] = SZ;
        for (auto e: adj[v]) if (e.C > e.flow) dist[v] = min(dist[v],
            dist[e.v] + 1);
        count[dist[v]] ++;
        enqueue(v);
    }

    void discharge(int v) {
        for (auto &e: adj[v]) {
            if (excess[v] > 0) push(v,e);
            else break;
        }
        if (excess[v] > 0) {
            if (count[dist[v]] == 1) gap(dist[v]);
            else relabel(v);
        }
    }

    ll maxFlow (int s, int t) {
        for (auto &e: adj[s]) excess[s] += e.C;

        count[0] = SZ;
        enqueue(s); active[t] = 1;

        while (b >= 0) {
            if (sz(B[b])) {
                int v = B[b].back(); B[b].pop_back();
                active[v] = 0; discharge(v);
            } else b--;
        }
        return excess[t];
    }
};

PushRelabel<50000> network;
```

5.37 SCC Kosaraju

```
// Kosaraju's strongly connected component
//
// Description:
//   For a graph G = (V, E), u and v are strongly connected if
//   there are paths u -> v and v -> u. This defines an equivalent
//   relation, and its equivalent class is called a strongly
//   connected component.
```

```
//
// Algorithm:
//   Kosaraju's algorithm performs DFS on G and rev(G).
//   First DFS finds topological ordering of SCCs, and
//   the second DFS extracts components.
//
// Complexity:
//   O(n + m)
//
// Verified:
//   SPOJ 6818
//
// References:
//   A. V. Aho, J. E. Hopcroft, and J. D. Ullman (1983):
//   Data Structures and Algorithms,
//   Addison-Wesley.
//
#include <iostream>
#include <vector>
#include <cstdio>
#include <cstdlib>
#include <map>
#include <set>
#include <cmath>
#include <cstring>
#include <functional>
#include <algorithm>
#include <unordered_map>
#include <unordered_set>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())


struct graph {
  int n;
  vector<vector<int>> adj, rdj;
  graph(int n) : n(n), adj(n), rdj(n) { }
  void add_edge(int src, int dst) {
    adj[src].push_back(dst);
    rdj[dst].push_back(src);
  }
  vector<vector<int>> strongly_connected_components() { // kosaraju
    vector<int> ord, visited(n);
    vector<vector<int>> scc;
    function<void(int, vector<vector<int>>&, vector<int>&)> dfs
    = [&](int u, vector<vector<int>> &adj, vector<int> &out) {
      visited[u] = true;
      for (int v : adj[u])
        if (!visited[v]) dfs(v, adj, out);
      out.push_back(u);
    };
    for (int u = 0; u < n; ++u)
      if (!visited[u]) dfs(u, adj, ord);
    fill(all(visited), false);
    for (int i = n - 1; i >= 0; --i)
      if (!visited[ord[i]])
        scc.push_back({}), dfs(ord[i], rdj, scc.back());
    return scc;
  }
};

int main() {
  int n, m;
  scanf("%d %d", &n, &m);
  graph g(n);
  for (int k = 0; k < m; ++k) {
    int i, j;
    scanf("%d %d", &i, &j);
    g.add_edge(i - 1, j - 1);
  }

  vector<vector<int>> scc = g.strongly_connected_components();
  vector<int> outdeg(scc.size());
  vector<int> id(n);
  for (int i = 0; i < scc.size(); ++i)
    for (int u : scc[i]) id[u] = i;
  for (int u = 0; u < n; ++u)
    for (int v : g.adj[u])
      if (id[u] != id[v]) ++outdeg[id[u]];

  if (count(all(outdeg), 0) != 1) {
    printf("0\n");
  } else {
    int i = find(all(outdeg), 0) - outdeg.begin();
    sort(all(scc[i]));
```

```
    printf("%d\n%d", scc[i].size(), scc[i][0] + 1);
    for (int j = 1; j < scc[i].size(); ++j)
      printf(" %d", scc[i][j] + 1);
    printf("\n");
  }
}
```

5.38 SCC Tarjan

```
stack<int> st;
vector<vector<int> > scc;
int low[MAX], disc[MAX], comp[MAX];
int dfs_time;
bool in_stack[MAX];

vi graph[MAX];
int n; // node count indexed from 1

void dfs(int u)
{
	low[u] = dfs_time;
	disc[u] = dfs_time;
	dfs_time++;

	in_stack[u] = true;
	st.push(u);

	int sz = graph[u].size(), v;
	for(int i = 0; i < sz; i++)
	{
		v = graph[u][i];

		if(disc[v] == -1)
		{
			dfs(v);
			low[u] = min(low[u], low[v]);
		}
		else if(in_stack[v] == true)
			low[u] = min(low[u], disc[v]);
	}

	if(low[u] == disc[u])
	{
		scc.push_back(vector<int>());
		while(st.top() != u)
		{
			scc[scc.size() - 1].push_back(st.top());
			in_stack[st.top()] = false;
			st.pop();
		}

		scc[scc.size() - 1].push_back(u);
		in_stack[u] = false;
		st.pop();
	}
}

int tarjan()
{
	memset(comp, -1, sizeof(comp));
	memset(disc, -1, sizeof(disc));
	memset(low, -1, sizeof(low));
	memset(in_stack, 0, sizeof(in_stack));
	dfs_time = 0;

	while(!st.empty())
		st.pop();

	for(int i = 1; i <= n; i++)
		if(disc[i] == -1)
			dfs(i);

	int sz = scc.size();
	for(int i = 0; i < sz; i++)
		for(int j = 0; j < (int)scc[i].size(); j++)
			comp[scc[i][j]] = i;

	return sz;
}
```

5.39 SPFA

```
int dist[MAX], inq[MAX];
void spfa(int source)
{
	FOR(i,1,n+1) inq[i]=false, dist[i]=inf; // or INF
```

```
	dist[source]=0;
	queue<int> Q;
	Q.push(source);
	inq[source]=true;

	while(!Q.empty())
	{
		int u=Q.front();
		Q.pop();
		FOR(j,0,graph[u].size())
		{
			int v=graph[u][j];

			if(dist[u]+cost[u][j]<dist[v])
			{
				dist[v]=dist[u]+cost[u][j];
				if(!inq[v])
				{
					Q.push(v);
					inq[v]=true;
				}
			}
		}
		inq[u]=false;
	}
}
```

5.40 Tree Construction with Specific Vertices

```
/* This code builds an auxiliary tree from the given vertices to do
further operations. Example problem: CF 613D */

void dfs(int u, int p=0, int d=0)
{
	tin[u]=++t;
	parent[u][0]=p;
	level[u]=d;
	for(auto v: graph[u])
	{
		if(v==p) continue;
		dfs(v,u,d+1);
	}
	tout[u]=t;
}
void cleanup(vi &vtx)
{
	for(auto it: vtx)
	{
		tree[it].clear();
	}
}
bool isancestor(int u, int v) // Check if u is an ancestor of v
{
	return (tin[u]<=tin[v]) && (tout[v]<=tout[u]);
}
// building the auxiliary tree. Nodes are in vtx
void sortbyEntry(vi &vtx)
{
	// Sort by entry time
	sort(begin(vtx), end(vtx), [](int x, int y){
		return tin[x]<tin[y];
	});
}
void release(vi &vtx)
{
	// removing duplicated nodes
	SORT(vtx);
	vtx.erase(unique(begin(vtx),end(vtx)),end(vtx));
}
void buildTree(vi &vtx)
{
	stack<int> st;
	st.push(vtx[0]);
	FOR(i,1,vtx.size())
	{
		while(!isancestor(st.top(),vtx[i]))
			st.pop();
		tree[st.top()].pb(vtx[i]);
		st.push(vtx[i]);
	}
}
int work(vi &vtx)
{
	sortbyEntry(vtx);
	int sz=vtx.size();
	// Finding all the ancestors, there are few of them
	FOR(i,0,sz-1)
	{
```

```
            int anc=query(vtx[i],vtx[i+1]);
            vtx.pb(anc);
        }
        release(vtx);
        sortbyEntry(vtx);
        buildTree(vtx);
        // Do necessary operation on the built auxiliary tree
        cleanup(vtx);
        // return result
}
```

5.41 kth Shortest Path Length

```
int n, m, x, y, k, a, b, c;
vi Graph[103], Cost[103];
vector<priority_queue<int> > d(103);
priority_queue < pii > Q;

void goDijkstra()
{

        // Here, elements are sorted in decreasing order of the first
            elements
        // of the pairs and then the second elements if equal first
            element.

        // d[i] is the priority_queue of the node i where the best k path
            length
        // will be stored in decreasing order. So, d[i].top() has the
            longest of the
        // first k shortest path.

        d[x].push(0);
        Q.push(MP(x,0));
        // Q contains the nodes in the increasing order of their cost
        // Since the priority_queue sorts the pairs in decreasing order of
            their
        // first element and then second element, to sort it in increasing
            order
        // we will negate the cost and push it.

        while(!Q.empty())
        {
                pii t=Q.top(); Q.pop();
                int u=t.first, costU=-t.second;
                // Since the actual cost was negated.

                FOR(j,0,Graph[u].size())
                {
                        int v=Graph[u][j];

                        // prnt(v); prnt(d[v].size());

                        // Have we already got k shortest paths? Or is the
                            longest path can be made better?
                        if(d[v].size()<k || d[v].top()>costU+Cost[u][j])
                        {
                                int temp=costU+Cost[u][j];
                                d[v].push(temp);
                                Q.push(MP(v,-temp));
                        }
                        if(d[v].size()>k) d[v].pop();
                // If we have more than k shortest path for the current node, we
                    can pop
                // the worst ones.
                        }
        }

        if(d[y].size()<k) prnt(-1);
        // We have not found k shortest path for our destination.
        else prnt(d[y].top());
}

int main()
{
        // ios_base::sync_with_stdio(0);
        // cin.tie(NULL); cout.tie(NULL);
        // freopen("in.txt","r",stdin);

        while(scanf("%d%d", &n, &m) && n+m)
        {
                scanf("%d%d%d", &x, &y, &k);

                FOR(i,0,m)
                {
                        scanf("%d%d%d", &a, &b, &c);

                        Graph[a].pb(b);
```

```
                Cost[a].pb(c);
        }

        goDijkstra();

        FOR(i,0,103) Graph[i].clear(), Cost[i].clear();
        FOR(i,0,103)
        {
                while(!d[i].empty()) d[i].pop();
        }

        while(!Q.empty()) Q.pop();
    }


    return 0;
}
```

6 Math

6.1 CRT Diophantine

```
// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b. Here, z is unique modulo M = lcm(x,y).
// Return (z,M). On failure, M = -1.
PII chinese_remainder_theorem(int x, int a, int y, int b) {
    int s, t;
    int d = extended_euclid(x, y, s, t);
    if (a % d != b % d) return make_pair(0, -1);
    return make_pair(mod(s * b * x + t * a * y, x * y) / d, x * y / d);
}

// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i. Note that the solution is
// unique modulo M = lcm_i (x[i]). Return (z,M). On
// failure, M = -1. Note that we do not require the a[i]'s
// to be relatively prime.

PII chinese_remainder_theorem(const VI &x, const VI &a) {
    PII ret = make_pair(a[0], x[0]);
    for (int i = 1; i < x.size(); i++) {
        ret = chinese_remainder_theorem(ret.second, ret.first,
            x[i], a[i]);
        if (ret.second == -1) break;
    }
    return ret;
}

// computes x and y such that ax + by = c; on failure, x = y =-1
void linear_diophantine(int a, int b, int c, int &x, int &y) {
    int d = gcd(a, b);
    if (c % d) {
        x = y = -1;
    } else {
        x = c / d * mod_inverse(a / d, b / d);
        y = (c - a * x) / b;
    }
}
```

6.2 Euler Phi

```
int phi[MAX];
void phi()
{
    for (int i = 1; i < MAX; i++) phi[i] = i;
    for (int i = 2; i < MAX; i++)
    {
        if (phi[i] == i)
        {
            for (int j = i; j < MAX; j += i)
            {
                phi[j] /= i;
                phi[j] *= (i - 1);
            }
        }
    }
}
```

6.3 FFT 1

```
const int MAXN = (1 << 21); // May not need to be changed
```

```
struct complex_base
{
    double x, y;
    complex_base(double _x = 0, double _y = 0) { x = _x; y = _y; }
    friend complex_base operator-(const complex_base &a, const
        complex_base &b) { return complex_base(a.x - b.x, a.y - b.y); }
    friend complex_base operator+(const complex_base &a, const
        complex_base &b) { return complex_base(a.x + b.x, a.y + b.y); }
    friend complex_base operator*(const complex_base &a, const
        complex_base &b) { return complex_base(a.x * b.x - a.y * b.y, a.y
        * b.x + b.y * a.x); }
    friend void operator/=(complex_base &a, const double &P) { a.x /= P;
        a.y /= P; }
};

int bit_rev[MAXN];

void fft(complex_base *a, int lg)
{
    int n = (1 << lg);
    for (int i = 1; i < n; i++)
    {
        bit_rev[i] = (bit_rev[i >> 1] >> 1) | ((i & 1) << (lg - 1));
        if (bit_rev[i] < i) swap(a[i], a[bit_rev[i]]);
    }

    for (int len = 2; len <= n; len <<= 1)
    {
        double ang = 2 * PI / len;
        complex_base w(1, 0), wn(cos(ang), sin(ang));
        for (int j = 0; j < (len >> 1); j++, w = w * wn)
            for (int i = 0; i < n; i += len)
            {
                complex_base u = a[i + j], v = w * a[i + j + (len >> 1)];
                a[i + j] = u + v;
                a[i + j + (len >> 1)] = u - v;
            }
    }
}

void inv_fft(complex_base *a, int lg)
{
    int n = (1 << lg);
    for (int i = 1; i < n; i++)
    {
        bit_rev[i] = (bit_rev[i >> 1] >> 1) | ((i & 1) << (lg - 1));
        if (bit_rev[i] < i) swap(a[i], a[bit_rev[i]]);
    }

    for (int len = 2; len <= n; len <<= 1)
    {
        double ang = -2 * PI / len;
        complex_base w(1, 0), wn(cos(ang), sin(ang));

        for (int j = 0; j < (len >> 1); j++, w = w * wn)
            for (int i = 0; i < n; i += len)
            {
                complex_base u = a[i + j], v = w * a[i + j + (len >> 1)];
                a[i + j] = u + v;
                a[i + j + (len >> 1)] = u - v;
            }
    }

    for (int i = 0; i < n; i++)
        a[i] /= n;
}

complex_base A[MAXN], B[MAXN];

vector<ll> mult(vector<ll> a, vector<ll> b)
{
    if (a.size() * b.size() <= 256)
    {
        vector<ll> ans(a.size() + b.size(), 0);
        for (int i = 0; i < (int)a.size(); i++)
            for (int j = 0; j < (int)b.size(); j++)
                ans[i + j] += a[i] * b[j];

        return ans;
    }

    int lg = 0; while ((1 << lg) < (a.size() + b.size())) ++lg;
    for (int i = 0; i < (1 << lg); i++) A[i] = B[i] = complex_base(0, 0);
    for (int i = 0; i < (int)a.size(); i++) A[i] = complex_base(a[i], 0);
    for (int i = 0; i < (int)b.size(); i++) B[i] = complex_base(b[i], 0);

    fft(A, lg); fft(B, lg);
    for (int i = 0; i < (1 << lg); i++)
        A[i] = A[i] * B[i];
    inv_fft(A, lg);
```

```
    vector<ll> ans(a.size() + b.size(), 0);
    for (int i = 0; i < (int)ans.size(); i++)
        ans[i] = (int)(A[i].x + 0.5);

    return ans;
}
```

6.4 FFT 2

```
// Tested:
// - FBHC 2016 R3 - Problem E
// - https://open.kattis.com/problems/polymul2 (need long double)
// Note:
// - a[2] will have size <= 2*n
// - When rounding, careful with negative numbers:
int my_round(double x) {
       if (x < 0) return -my_round(-x);
       return (int) (x + 1e-3);
}
const int N = 1 << 18;
typedef complex<long double> cplex; // may need long double
int rev[N];
cplex wlen_pw[N], fa[N], fb[N];

void fft(cplex a[], int n, bool invert) {
       for (int i = 0; i < n; ++i) if (i < rev[i]) swap (a[i], a[rev[i]]);

       for (int len = 2; len <= n; len <<= 1) {
              double alpha = 2 * PI / len * (invert ? -1 : +1);
              int len2 = len >> 1;

              wlen_pw[0] = cplex(1, 0);
              cplex wlen(cos(alpha), sin(alpha));
              for (int i = 1; i < len2; ++i) wlen_pw[i] = wlen_pw[i - 1]
                  * wlen;

              for (int i = 0; i < n; i += len) {
                     cplex t, *pu = a + i, *pv = a + i + len2,
                              *pu_end = a + i + len2, *pw = wlen_pw;
                     for (; pu != pu_end; ++pu, ++pv, ++pw) {
                            t = *pv * *pw;
                            *pv = *pu - t;
                            *pu += t;
                     }
              }
       }

       if (invert) FOR(i, 0, n) a[i] /= n;
}
void calcRev(int n, int logn) {
       FOR(i, 0, n) {
              rev[i] = 0;
              FOR(j, 0, logn) if (i & (1 << j)) rev[i] |= 1 << (logn - 1
                  - j);
       }
}
void mulpoly(int a[], int b[], ll c[], int na, int nb, int &n) {
       int l = max(na, nb), logn = 0;
       for (n = 1; n < l; n <<= 1) ++logn;
       n <<= 1; ++logn;
       calcRev(n, logn);

       FOR(i, 0, n) fa[i] = fb[i] = cplex(0);
       FOR(i, 0, na) fa[i] = cplex(a[i]);
       FOR(i, 0, nb) fb[i] = cplex(b[i]);

       fft(fa, n, false);
       fft(fb, n, false);

       FOR(i, 0, n) fa[i] *= fb[i];
       fft(fa, n, true);
       // if everything is double/long double, we don't add 0.5
       FOR(i, 0, n) c[i] = (ll)(fa[i].real() + 0.5);
}
// call
    mulpoly(first_poly,second_poly,output,size_first,size_second,size_output)
```

6.5 FFT Extended

```
#include <bits/stdtr1c++.h>

#define MAXN 1048576 /// 2 * MAX at least
using namespace std;

/// Change long double to double if not required
```

```
namespace fft {
int len, last = -1, step = 0, rev[MAXN];
long long C[MAXN], D[MAXN], P[MAXN], Q[MAXN];
struct complx {
    long double real, img;
    inline complx() {
        real = img = 0.0;
    }
    inline complx conjugate() {
        return complx(real, -img);
    }
    inline complx(long double x) {
        real = x, img = 0.0;
    }
    inline complx(long double x, long double y) {
        real = x, img = y;
    }
    inline complx operator + (complx other) {
        return complx(real + other.real, img + other.img);
    }
    inline complx operator - (complx other) {
        return complx(real - other.real, img - other.img);
    }
    inline complx operator * (complx other) {
        return complx((real * other.real) - (img * other.img), (real *
            other.img) + (img * other.real));
    }
} u[MAXN], v[MAXN], f[MAXN], g[MAXN], dp[MAXN], inv[MAXN];

void build(int& a, long long* A, int& b, long long* B) {
    while (a > 1 && A[a - 1] == 0) a--;
    while (b > 1 && B[b - 1] == 0) b--;

    len = 1 << (32 - __builtin_clz(a + b) - (__builtin_popcount(a + b) ==
        1));
    for (int i = a; i < len; i++) A[i] = 0;
    for (int i = b; i < len; i++) B[i] = 0;

    if (!step++) {
        dp[1] = inv[1] = complx(1);
        for (int i = 1; (1 << i) < MAXN; i++) {
            double theta = (2.0 * acos(0.0)) / (1 << i);
            complx mul = complx(cos(theta), sin(theta));
            complx inv_mul = complx(cos(-theta), sin(-theta));
            int lim = 1 << i;
            for (int j = lim >> 1; j < lim; j++) {
                dp[2 * j] = dp[j], inv[2 * j] = inv[j];
                inv[2 * j + 1] = inv[j] * inv_mul;
                dp[2 * j + 1] = dp[j] * mul;
            }
        }
    }

    if (last != len) {
        last = len;
        int bit = (32 - __builtin_clz(len) - (__builtin_popcount(len) ==
            1));
        for (int i = 0; i < len; i++) rev[i] = (rev[i >> 1] >> 1) + ((i &
            1) << (bit - 1));
    }
}
/// Fast Fourier Transformation, iterative divide and conquer
void transform(complx *in, complx *out, complx* ar) {
    for (int i = 0; i < len; i++) out[i] = in[rev[i]];
    for (int k = 1; k < len; k <<= 1) {
        for (int i = 0; i < len; i += (k << 1)) {
            for (int j = 0; j < k; j++) {
                complx z = out[i + j + k] * ar[j + k];
                out[i + j + k] = out[i + j] - z;
                out[i + j] = out[i + j] + z;
            }
        }
    }
}
/// Fast Fourier Transformation, iterative divide and conquer unrolled
    and optimized
void transform_unrolled(complx *in, complx *out, complx* ar) {
    for (int i = 0; i < len; i++) out[i] = in[rev[i]];
    for (int k = 1; k < len; k <<= 1) {
        for (int i = 0; i < len; i += (k << 1)) {
            complx z, *a = out + i, *b = out + i + k, *c = ar + k;
            if (k == 1) {
                z = (*b) * (*c);
                *b = *a - z, *a = *a + z;
            }

            for (int j = 0; j < k && k > 1; j += 2, a++, b++, c++) {
                z = (*b) * (*c);
```

```
                *b = *a - z, *a = *a + z;
                a++, b++, c++;
                z = (*b) * (*c);
                *b = *a - z, *a = *a + z;
            }
        }
    }
}
bool equals(int a, long long* A, int b, long long* B) {
    if (a != b) return false;
    for (a = 0; a < b && A[a] == B[a]; a++) {}
    return (a == b);
}
/// Square of a polynomial
int square(int a, long long* A) {
    build(a, A, a, A);
    for (int i = 0; i < len; i++) u[i] = complx(A[i], 0);
    transform_unrolled(u, f, dp);
    for (int i = 0; i < len; i++) u[i] = f[i] * f[i];
    transform_unrolled(u, f, inv);
    for (int i = 0; i < len; i++) A[i] = (f[i].real / (long double)len) +
        0.5;
    return a + a - 1;
}
/// Multiplies two polynomials A and B and return the coefficients of
    their product in A
/// Function returns degree of the polynomial A * B
int multiply(int a, long long* A, int b, long long* B) {
    if (equals(a, A, b, B)) return square(a, A); /// Optimization

    build(a, A, b, B);
    for (int i = 0; i < len; i++) u[i] = complx(A[i], B[i]);
    transform_unrolled(u, f, dp);
    for (int i = 0; i < len; i++) {
        int j = (len - 1) & (len - i);
        u[i] = (f[j] * f[j] - f[i].conjugate() * f[i].conjugate()) *
            complx(0, -0.25 / len);
    }
    transform_unrolled(u, f, dp);
    for (int i = 0; i < len; i++) A[i] = f[i].real + 0.5;
    return a + b - 1;
}
/// Modular multiplication
int mod_multiply(int a, long long* A, int b, long long* B, int mod) {
    build(a, A, b, B);
    int flag = equals(a, A, b, B);
    for (int i = 0; i < len; i++) A[i] %= mod, B[i] %= mod;
    for (int i = 0; i < len; i++) u[i] = complx(A[i] & 32767, A[i] >> 15);
    for (int i = 0; i < len; i++) v[i] = complx(B[i] & 32767, B[i] >> 15);

    transform_unrolled(u, f, dp);
    for (int i = 0; i < len; i++) g[i] = f[i];
    if (!flag) transform_unrolled(v, g, dp);

    for (int i = 0; i < len; i++) {
        int j = (len - 1) & (len - i);
        complx c1 = f[j].conjugate(), c2 = g[j].conjugate();

        complx a1 = (f[i] + c1) * complx(0.5, 0);
        complx a2 = (f[i] - c1) * complx(0, -0.5);
        complx b1 = (g[i] + c2) * complx(0.5 / len, 0);
        complx b2 = (g[i] - c2) * complx(0, -0.5 / len);
        v[j] = a1 * b2 + a2 * b1;
        u[j] = a1 * b1 + a2 * b2 * complx(0, 1);
    }
    transform_unrolled(u, f, dp);
    transform_unrolled(v, g, dp);

    long long x, y, z;
    for (int i = 0; i < len; i++) {
        x = f[i].real + 0.5, y = g[i].real + 0.5, z = f[i].img + 0.5;
        A[i] = (x + ((y % mod) << 15) + ((z % mod) << 30)) % mod;
    }
    return a + b - 1;
}
/// Multiplies two polynomials where intermediate and final values fits
    in long long
int long_multiply(int a, long long* A, int b, long long* B) {
    int mod1 = 1.5e9;
    int mod2 = mod1 + 1;
    for (int i = 0; i < a; i++) C[i] = A[i];
    for (int i = 0; i < b; i++) D[i] = B[i];

    mod_multiply(a, A, b, B, mod1);
    mod_multiply(a, C, b, D, mod2);
    for (int i = 0; i < len; i++) {
        A[i] = A[i] + (C[i] - A[i] + (long long)mod2) * (long long)mod1 %
            mod2 * mod1;
    }
    return a + b - 1;
```

```
}
int build_convolution(int n, long long* A, long long* B) {
    int i, m, d = 0;
    for (i = 0; i < n; i++) Q[i] = Q[i + n] = B[i];
    for (i = 0; i < n; i++) P[i] = A[i], P[i + n] = 0;
    n *= 2, m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) ==
        1));
    for (i = n; i < m; i++) P[i] = Q[i] = 0;
    return n;
}
/***
    Computes the circular convolution of A and B, denoted A * B, in C
    A and B must be of equal size, if not normalize before calling
        function
    Example to demonstrate convolution for n = 5:

    c0 = a0b0 + a1b4 + a2b3 + a3b2 + a4b1
    c1 = a0b1 + a1b0 + a2b4 + a3b3 + a4b2
    ...
    c4 = a0b4 + a1b3 + a2b2 + a3b1 + a4b0
    Note: If linear convolution is required, pad with zeros
        appropriately, as in multiplication
***/
/// Returns the convolution of A and B in A
void convolution(int n, long long* A, long long* B) {
    int len = build_convolution(n, A, B);
    multiply(len, P, len, Q);
    for (int i = 0; i < n; i++) A[i] = P[i + n];
}
/// Modular convolution
void mod_convolution(int n, long long* A, long long* B, int mod) {
    int len = build_convolution(n, A, B);
    mod_multiply(len, P, len, Q, mod);
    for (int i = 0; i < n; i++) A[i] = P[i + n];
}
/// Convolution in long long
void long_convolution(int n, long long* A, long long* B) {
    int len = build_convolution(n, A, B);
    long_multiply(len, P, len, Q);
    for (int i = 0; i < n; i++) A[i] = P[i + n];
}
/// Hamming distance vector of B with every substring of length |pattern|
    in str
/// str and pattern consists of only '1' and '0'
/// str = "01111000010011111111111001000110100010001111010111"
/// pattern = "10011010011011101011010000"
/// Sum of values in hamming distance vector = 321
vector <int> hamming_distance(const char* str, const char* pattern) {
    int n = strlen(str), m = strlen(pattern);
    for (int i = 0; i < n; i++) P[i] = Q[i] = 0;
    for (int i = 0; i < n; i++) P[i] = str[i] == '1' ? 1 : -1;
    for (int i = 0, j = m - 1; j >= 0; i++, j--) Q[i] = pattern[j] == '1'
        ? 1 : -1;

    vector <int> res;
    fft::multiply(n, P, m, Q);
    for (int i = 0; (i + m) <= n; i++) {
        res.push_back(m - ((P[i + m - 1] + m) >> 1));
    }
    return res;
}
}
```

6.6 FFT Modulo

```
// Caution: Got TLE in divide and conquer + FFT problem

template<class T, class T2> inline void chkmax(T &x, const T2 &y) { if (x
    < y) x = y; }
template<class T, class T2> inline void chkmin(T &x, const T2 &y) { if (x
    > y) x = y; }
const int MAXN = (1 << 19);
int mod=1009;

inline void addmod(int& x, int y, int mod) { (x += y) >= mod && (x -=
    mod); }
inline int mulmod(int x, int y, int mod) { return x * 1ll * y % mod; }

struct complex_base
{
        long double x, y;
        complex_base(long double _x = 0, long double _y = 0) { x = _x; y =
            _y; }
        friend complex_base operator-(const complex_base &a, const
            complex_base &b) { return complex_base(a.x - b.x, a.y - b.y);
            }
        friend complex_base operator+(const complex_base &a, const
            complex_base &b) { return complex_base(a.x + b.x, a.y + b.y);
```

```
        }
    friend complex_base operator*(const complex_base &a, const
        complex_base &b) { return complex_base(a.x * b.x - a.y * b.y,
        a.y * b.x + b.y * a.x); }
    friend void operator/=(complex_base &a, const long double &P) {
        a.x /= P; a.y /= P; }
};

int bit_rev[MAXN];

void fft(complex_base *a, int lg)
{
    int n = (1 << lg);
    for (int i = 1; i < n; i++)
    {
        bit_rev[i] = (bit_rev[i >> 1] >> 1) | ((i & 1) << (lg -
            1));
        if (bit_rev[i] < i) swap(a[i], a[bit_rev[i]]);
    }

    for (int len = 2; len <= n; len <<= 1)
    {
        long double ang = 2 * PI / len;
        complex_base w(1, 0), wn(cos(ang), sin(ang));
        for (int j = 0; j < (len >> 1); j++, w = w * wn)
            for (int i = 0; i < n; i += len)
            {
                complex_base u = a[i + j], v = w * a[i + j +
                    (len >> 1)];
                a[i + j] = u + v;
                a[i + j + (len >> 1)] = u - v;
            }
    }
}

void inv_fft(complex_base *a, int lg)
{
    int n = (1 << lg);
    for (int i = 1; i < n; i++)
    {
        bit_rev[i] = (bit_rev[i >> 1] >> 1) | ((i & 1) << (lg -
            1));
        if (bit_rev[i] < i) swap(a[i], a[bit_rev[i]]);
    }

    for (int len = 2; len <= n; len <<= 1)
    {
        long double ang = -2 * PI / len;
        complex_base w(1, 0), wn(cos(ang), sin(ang));

        for (int j = 0; j < (len >> 1); j++, w = w * wn)
            for (int i = 0; i < n; i += len)
            {
                complex_base u = a[i + j], v = w * a[i + j +
                    (len >> 1)];
                a[i + j] = u + v;
                a[i + j + (len >> 1)] = u - v;
            }
    }

    for (int i = 0; i < n; i++)
        a[i] /= n;
}

complex_base A[MAXN], B[MAXN];

vector<int> mult(const vector<int> &a, const vector<int> &b)
{
    if (a.size() * b.size() <= 128)
    {
        vector<int> ans(a.size() + b.size(), 0);
        for (int i = 0; i < (int)a.size(); i++)
            for (int j = 0; j < (int)b.size(); j++)
                ans[i + j] = (ans[i + j] + a[i] * 1ll *
                    b[j]) % mod;

        return ans;
    }

    int lg = 0; while ((1 << lg) < (a.size() + b.size())) ++lg;
    for (int i = 0; i < (1 << lg); i++) A[i] = B[i] = complex_base(0,
        0);
    for (int i = 0; i < (int)a.size(); i++) A[i] = complex_base(a[i],
        0);
    for (int i = 0; i < (int)b.size(); i++) B[i] = complex_base(b[i],
        0);

    fft(A, lg); fft(B, lg);
    for (int i = 0; i < (1 << lg); i++)
        A[i] = A[i] * B[i];
```

```
	inv_fft(A, lg);

	vector<int> ans(a.size() + b.size(), 0);
	for (int i = 0; i < (int)ans.size(); i++)
		ans[i] = (int64_t)(A[i].x + 0.5) % mod;

	return ans;
}

vector<int> mult_mod(const vector<int> &a, const vector<int> &b)
{
	/// Thanks pavel.savchenkov

	// a = a0 + sqrt(MOD) * a1
	// a = a0 + base * a1
	int base = (int)sqrtl(mod);

	vector<int> a0(a.size()), a1(a.size());
	for (int i = 0; i < (int)a.size(); i++)
	{
		a0[i] = a[i] % base;
		a1[i] = a[i] / base;
	}

	vector<int> b0(b.size()), b1(b.size());
	for (int i = 0; i < (int)b.size(); i++)
	{
		b0[i] = b[i] % base;
		b1[i] = b[i] / base;
	}

	vector<int> a01 = a0;
	for (int i = 0; i < (int)a.size(); i++)
		addmod(a01[i], a1[i], mod);

	vector<int> b01 = b0;
	for (int i = 0; i < (int)b.size(); i++)
		addmod(b01[i], b1[i], mod);

	vector<int> C = mult(a01, b01); // 1

	vector<int> a0b0 = mult(a0, b0); // 2
	vector<int> a1b1 = mult(a1, b1); // 3

	vector<int> mid = C;
	for (int i = 0; i < (int)mid.size(); i++)
	{
		addmod(mid[i], -a0b0[i] + mod, mod);
		addmod(mid[i], -a1b1[i] + mod, mod);
	}

	vector<int> res = a0b0;
	for (int i = 0; i < (int)res.size(); i++)
		addmod(res[i], mulmod(base, mid[i], mod), mod);

	base = mulmod(base, base, mod);
	for (int i = 0; i < (int)res.size(); i++)
		addmod(res[i], mulmod(base, a1b1[i], mod), mod);

	return res;
}
```

6.7 FFT by XraY

```
typedef long double ld;
#define mp make_pair
#define eprintf(...) fprintf(stderr, __VA_ARGS__)
#define sz(x) ((int)(x).size())

const ld pi = acos((ld) - 1);
//BEGIN ALGO
namespace FFT {
struct com {
	ld x, y;

	com(ld _x = 0, ld _y = 0) : x(_x), y(_y) {}

	inline com operator + (const com &c) const {
		return com(x + c.x, y + c.y);
	}
	inline com operator - (const com &c) const {
		return com(x - c.x, y - c.y);
	}
	inline com operator * (const com &c) const {
		return com(x * c.x - y * c.y, x * c.y + y * c.x);
	}
	inline com conj() const {
		return com(x, -y);
```

```
	}
};

const static int maxk = 21, maxn = (1 << maxk) + 1;
com ws[maxn];
int dp[maxn];
com rs[maxn];
int n, k;
int lastk = -1;

void fft(com *a, bool torev = 0) {
	if (lastk != k) {
		lastk = k;
		dp[0] = 0;

		for (int i = 1, g = -1; i < n; ++i) {
			if (!(i & (i - 1))) {
				++g;
			}
			dp[i] = dp[i ^ (1 << g)] ^ (1 << (k - 1 - g));
		}

		ws[1] = com(1, 0);
		for (int two = 0; two < k - 1; ++two) {
			ld alf = pi / n * (1 << (k - 1 - two));
			com cur = com(cos(alf), sin(alf));

			int p2 = (1 << two), p3 = p2 * 2;
			for (int j = p2; j < p3; ++j) {
				ws[j * 2 + 1] = (ws[j * 2] = ws[j]) * cur;
			}
		}
	}
	for (int i = 0; i < n; ++i) {
		if (i < dp[i]) {
			swap(a[i], a[dp[i]]);
		}
	}
	if (torev) {
		for (int i = 0; i < n; ++i) {
			a[i].y = -a[i].y;
		}
	}
	for (int len = 1; len < n; len <<= 1) {
		for (int i = 0; i < n; i += len) {
			int wit = len;
			for (int it = 0, j = i + len; it < len; ++it, ++i,
			    ++j) {
				com tmp = a[j] * ws[wit++];
				a[j] = a[i] - tmp;
				a[i] = a[i] + tmp;
			}
		}
	}
}

com a[maxn];
int mult(int na, int *_a, int nb, int *_b, long long *ans) {
	if (!na || !nb) {
		return 0;
	}
	for (k = 0, n = 1; n < na + nb - 1; n <<= 1, ++k) ;
	assert(n < maxn);
	for (int i = 0; i < n; ++i) {
		a[i] = com(i < na ? _a[i] : 0, i < nb ? _b[i] : 0);
	}
	fft(a);
	a[n] = a[0];
	for (int i = 0; i <= n - i; ++i) {
		a[i] = (a[i] * a[i] - (a[n - i] * a[n - i]).conj()) *
		    com(0, (ld) - 1 / n / 4);
		a[n - i] = a[i].conj();
	}
	fft(a, 1);
	int res = 0;
	for (int i = 0; i < n; ++i) {
		long long val = (long long) round(a[i].x);
		assert(abs(val - a[i].x) < 1e-1);
		if (val) {
			assert(i < na + nb - 1);
			while (res < i) {
				ans[res++] = 0;
			}
			ans[res++] = val;
		}
	}
	return res;
}
};
```

```
int main()
{
	// ios_base::sync_with_stdio(0);
	// cin.tie(NULL); cout.tie(NULL);
	// freopen("in.txt","r",stdin);

	int test, cases = 1;



	return 0;
}
```

6.8 Fast Integer Cube and Square Root

```
unsigned int fast_sqrt(unsigned int n){
    unsigned int c, g;

    c = g = 0x8000;
    for (; ;){
        if ((g * g) > n) g ^= c;
        c >>= 1;
        if (!c) return g;
        g |= c;
    }
}

int fast_cbrt(int n){
    int x, r = 30, res = 0;

    for (; r >= 0; r -= 3){
        res <<= 1;
        x = (3 * res * (res + 1)) + 1;
        if ((n >> r) >= x){
            res++;
            n -= (x << r);
        }
    }

    return res;
}

unsigned long long fast_sqrt(unsigned long long n){
    unsigned long long c, g;

    c = g = 0x80000000;
    for (; ;){
        if ((g * g) > n) g ^= c;
        c >>= 1;
        if (!c) return g;
        g |= c;
    }
}

unsigned long long fast_cbrt(unsigned long long n){
    int r = 63;
    unsigned long long x, res = 0;

    for (; r >= 0; r -= 3){
        res <<= 1;
        x = (res * (res + 1) * 3) + 1;
        if ((n >> r) >= x){
            res++;
            n -= (x << r);
        }
    }

    return res;
}

int main(){

}
```

6.9 Fast Walsh-Hadamard Transform

```
const int N = 1<<16;

template <typename T>
struct FWT {
	void fwt(T io[], int n) {
		for (int d = 1; d < n; d <<= 1) {
			for (int i = 0, m = d<<1; i < n; i += m) {
				for (int j = 0; j < d; j++) { /// Don't
				    forget modulo if required
					T x = io[i+j], y = io[i+j+d];
```

```
                        io[i+j] = (x+y), io[i+j+d] = (x-y);
                            // xor
                        // io[i+j] = x+y; // and
                        // io[i+j+d] = x+y; // or
                    }
                }
            }
        }
        void ufwt(T io[], int n) {
            for (int d = 1; d < n; d <<= 1) {
                for (int i = 0, m = d<<1; i < n; i += m) {
                    for (int j = 0; j < d; j++) { /// Don't
                        forget modulo if required
                        T x = io[i+j], y = io[i+j+d];
                         /// Modular inverse if required here
                        io[i+j] = (x+y)>>1, io[i+j+d] =
                            (x-y)>>1; // xor
                        // io[i+j] = x-y; // and
                        // io[i+j+d] = y-x; // or
                    }
                }
            }
        }
        // a, b are two polynomials and n is size which is power of two
        void convolution(T a[], T b[], int n) {
            fwt(a, n);
            fwt(b, n);
            for (int i = 0; i < n; i++)
                a[i] = a[i]*b[i];
            ufwt(a, n);
        }
        // for a*a
        void self_convolution(T a[], int n) {
            fwt(a, n);
            for (int i = 0; i < n; i++)
                a[i] = a[i]*a[i];
            ufwt(a, n);
        }
};
FWT<ll> fwt;
```

6.10 Faulhaber's Formula (Custom Algorithm)

```
#include <bits/stdtr1c++.h>

#define MAX 1010
#define MOD 1000000007
using namespace std;
namespace fool{
    #define MAXN 10000

    tr1::unordered_map <unsigned long long, int> mp;
    int inv, P[MAX], binomial[MAX][MAX], dp[MAXN][MAX];

    long long expo(long long x, long long n){
        x %= MOD;
        long long res = 1;

        while (n){
            if (n & 1) res = (res * x) % MOD;
            x = (x * x) % MOD;
            n >>= 1;
        }

        return (res % MOD);
    }

    void init(){
        int i, j;
        mp.clear();
        inv = expo(2, MOD - 2);

        P[0] = 1;
        for (i = 1; i < MAX; i++){
            P[i] = (P[i - 1] << 1);
            if (P[i] >= MOD) P[i] -= MOD;
        }

        for (i = 0; i < MAX; i++){
            for (j = 0; j <= i; j++){
                if (i == j || !j) binomial[i][j] = 1;
                else{
                    binomial[i][j] = (binomial[i - 1][j] + binomial[i -
                        1][j - 1]);
                    if (binomial[i][j] >= MOD) binomial[i][j] -= MOD;
                }
            }
        }
```

```
    for (i = 1; i < MAXN; i++){
        long long x = 1;
        for (j = 0; j < MAX; j++){
            dp[i][j] = dp[i - 1][j] + x;
            if (dp[i][j] >= MOD) dp[i][j] -= MOD;
            x = (x * i) % MOD;
        }
    }
}

/// Returns (1^k + 2^k + 3^k + .... n^k) % MOD
long long F(unsigned long long n, int k){
    if (n < MAXN) return dp[n][k];

    if (n == 1) return 1;
    if (n == 2) return (P[k] + 1) % MOD;
    if (!k) return (n % MOD);
    if (k == 1){
        n %= MOD;
        return (((n * (n + 1)) % MOD) * inv) % MOD;
    }

    unsigned long long h = (n << 10LL) | k; /// Change hash function
        according to limits of n and k
    long long res = mp[h];
    if (res) return res;

    if (n & 1) res = F(n - 1, k) + expo(n, k);
    else{
        long long m, z;
        m = n >> 1;
        res = (F(m, k) * P[k]) % MOD;

        m--, res++;
        for (int i = 0; i <= k; i++){
            z = (F(m, i) * binomial[k][i]) % MOD;
            z = (z * P[i]) % MOD;
            res += z;
        }
    }

    res %= MOD;
    return (mp[h] = res);
}
```

```
    long long faulhaber(unsigned long long n, int k){
        ///fool::init();
        return F(n, k);
    }
}

int main(){
    fool::init();
    int t, i, j;
    long long n, k, res;

    cin >> t;
    while (t--){
        cin >> n >> k;
        res = fool::faulhaber(n, k);
        cout << res << endl;
    }
    return 0;
}
```

6.11 Faulhaber's Formula

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 2510
#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int S[MAX][MAX], inv[MAX];

int expo(long long x, int n){
    x %= MOD;
    long long res = 1;

    while (n){
        if (n & 1) res = (res * x) % MOD;
        x = (x * x) % MOD;
        n >>= 1;
    }
```

```
    return (res % MOD);
}

void Generate(){
    int i, j;
    for (i = 0; i < MAX; i++) inv[i] = expo(i, MOD - 2);

    S[0][0] = 1;
    for (i = 1; i < MAX; i++){
        S[i][0] = 0;
        for (j = 1; j <= i; j++){
            S[i][j] = ( ((long long)S[i - 1][j] * j) + S[i - 1][j - 1]) %
                MOD;
        }
    }
}

int faulhaber(long long n, int k){
    n %= MOD;
    if (!k) return n;

    int j;
    long long res = 0, p = 1;
    for (j = 0; j <= k; j++){
        p = (p * (n + 1 - j)) % MOD;
        res = (res + (((S[k][j] * p) % MOD) * inv[j + 1])) % MOD;
    }

    return (res % MOD);
}

int main(){
    Generate();
    printf("%d\n", faulhaber(1001212, 1000));
    return 0;
}
```

6.12 Gauss Elimination Equations Mod Number Solutions

```
ll pow(ll base, ll p, ll MOD)
{
        if(p == 0) return 1;
        if(p % 2 == 0) { ll d = pow(base, p / 2, MOD); return (d * d) %
            MOD; }
        return (pow(base, p - 1, MOD) * base) % MOD;
}

ll inv(ll x, ll MOD) { return pow(x, MOD - 2, MOD); }

// If MOD equals 2, it becomes XOR operation and we can use vector of
    bitsets to build equation
// Complexity becomes 1/32

ll gauss(vector<vector<ll> > &a, ll MOD)
{
        int n = a.size(), m = a[0].size() - 1;

        for(int i = 0; i < n; i++)
                for(int j = 0; j <= m; j++)
                        a[i][j] = (a[i][j] % MOD + MOD) % MOD;

        vector<int> where(m, -1);
        for(int col = 0, row = 0; col < m && row < n; col++)
        {
                int sel = row;
                for(int i = row; i < n; i++)
                        if(a[i][col] > a[sel][col])
                                sel = i;

                        if(a[sel][col] == 0) { where[col] = -1; continue;
        }

        for(int i = col; i <= m; i++)
                        swap(a[sel][i], a[row][i]);
                where[col] = row;

                ll c_inv = inv(a[row][col], MOD);
                for(int i = 0; i < n; i++)
                        if(i != row)
                        {
                                if(a[i][col] == 0) continue;
                                ll c = (a[i][col] * c_inv) % MOD;
                                for(int j = 0; j <= m; j++)
                                    a[i][j] = (a[i][j] - c * a[row][j] % MOD
                                        + MOD) % MOD;
                        }
```

```
            row++;
    }
    vector<ll> ans(m, 0);
    ll result = 1;
    // for counting rank, take the count of where[i]==-1
    for(int i = 0; i < m; i++)
        if(where[i] != -1) ans[i] = (a[where[i]][m] * inv(a[where[i]][i],
            MOD)) % MOD;
               else result = (result * MOD) % mod;
        // This is validity check probably.May not be needed
    for(int i = 0; i < n; i++)
        {
               ll sum = a[i][m] % MOD;
               for(int j = 0; j < m; j++)
                      sum = (sum + MOD - (ans[j] * a[i][j]) % MOD) % MOD;

               if(sum != 0) return 0;
        }

        return result;
}
```

6.13 Gauss Jordan Elimination

```
// Gauss-Jordan elimination with full pivoting.
//
// Uses:
//   (1) solving systems of linear equations (AX=B)
//   (2) inverting matrices (AX=I)
//   (3) computing determinants of square matrices
//
// Running time: O(n^3)
//
// INPUT:   a[][] = an nxn matrix
//          b[][] = an nxm matrix
//
// OUTPUT:  X      = an nxm matrix (stored in b[][])
//          A^{-1} = an nxn matrix (stored in a[][])
//          returns determinant of a[][]

// Example used: LightOJ Snakes and Ladders

#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

const double EPS = 1e-10;

typedef vector<int> VI;
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;

T GaussJordan(VVT &a, VVT &b) {
  const int n = a.size();
  const int m = b[0].size();
  VI irow(n), icol(n), ipiv(n);
  T det = 1;

  for (int i = 0; i < n; i++) {
    int pj = -1, pk = -1;
    for (int j = 0; j < n; j++) if (!ipiv[j])
      for (int k = 0; k < n; k++) if (!ipiv[k])
        if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk = k;
            }
    if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl;
        exit(0); }
    ipiv[pk]++;
    swap(a[pj], a[pk]);
    swap(b[pj], b[pk]);
    if (pj != pk) det *= -1;
    irow[i] = pj;
    icol[i] = pk;

    T c = 1.0 / a[pk][pk];
    det *= a[pk][pk];
    a[pk][pk] = 1.0;
    for (int p = 0; p < n; p++) a[pk][p] *= c;
    for (int p = 0; p < m; p++) b[pk][p] *= c;
    for (int p = 0; p < n; p++) if (p != pk) {
      c = a[p][pk];
      a[p][pk] = 0;
      for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
      for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
    }
  }
```

```
  for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
    for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
  }

  return det;
}

int main() {
  const int n = 4;
  const int m = 2;
  double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
  double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
  VVT a(n), b(n);
  for (int i = 0; i < n; i++) {
    a[i] = VT(A[i], A[i] + n);
    b[i] = VT(B[i], B[i] + m);
  }

  double det = GaussJordan(a, b);

  // expected: 60
  cout << "Determinant: " << det << endl;

  // expected: -0.233333 0.166667 0.133333 0.0666667
  //           0.166667 0.166667 0.333333 -0.333333
  //           0.233333 0.833333 -0.133333 -0.0666667
  //           0.05 -0.75 -0.1 0.2
  cout << "Inverse: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
      cout << a[i][j] << ' ';
    cout << endl;
  }

  // expected: 1.63333 1.3
  //           -0.166667 0.5
  //           2.36667 1.7
  //           -1.85 -1.35
  cout << "Solution: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++)
      cout << b[i][j] << ' ';
    cout << endl;
  }
}
```

6.14 Gauss Xor

```
const int MAXN = (1 << 20);
const int MAXLOG = 64;

struct basis
{
    int64_t base[MAXLOG];

    void clear()
    {
            for(int i = MAXLOG - 1; i >= 0; i--)
                base[i] = 0;
    }

    void add(int64_t val)
    {
        for(int i = MAXLOG - 1; i >= 0; i--)
                if((val >> i) & 1)
                {
                    if(!base[i]) { base[i] = val; return; }
                    else val ^= base[i];
                }
        }

    inline int size()
    {
        int sz = 0;
        for(int i = 0; i < MAXLOG; i++)
                sz += (bool)(base[i]);
            return sz;
    }

    int64_t max_xor()
    {
            int64_t res = 0;
            for(int i = MAXLOG - 1; i >= 0; i--)
                if(!((res >> i) & 1) && base[i])
                    res ^= base[i];

            return res;
```

```
        }

    bool can_create(int64_t val)
    {
                for(int i = MAXLOG - 1; i >= 0; i--)
                        if(((val >> i) & 1) && base[i])
                                val ^= base[i];

                return (val == 0);
    }
};
```

6.15 Gaussian 1

```
void gauss(vector< vector<double> > &A) {

    int n = A.size();

        for(int i = 0; i < n; i++){
        int r = i;
        for(int j = i+1; j < n; j++)
            if(fabs(A[j][i]) > fabs(A[r][i]))
                r = j;
        if(fabs(A[r][i]) < EPS) continue;
        if(r != i)
            for(int j = 0; j <= n; j++)
                swap(A[r][j], A[i][j]);
        for(int k = 0; k < n; k++){
            if(k != i){
                for(int j = n; j >= i; j--)
                    A[k][j] -= A[k][i]/A[i][i]*A[i][j];
            }
        }
    }

    // solve: A[x][n]/A[x][x] for each x
}
```

6.16 Gaussian 2

```
const double eps = 1e-9;

// *****may return empty vector

vector<double> gauss(vector<vector<double>> &a)
{
        int n = a.size(), m = a[0].size() - 1;

        vector<int> where(m, -1);
        for(int col = 0, row = 0; col < m && row < n; col++)
    {
        int sel = row;
        for(int i = row; i < n; i++)
                if(abs(a[i][col]) > abs(a[sel][col]))
                        sel = i;

                if(abs(a[sel][col]) < eps) { where[col] = -1; continue; }

        for(int i = col; i <= m; i++)
                        swap(a[sel][i], a[row][i]);
                where[col] = row;

                for(int i = 0; i < n; i++)
                        if(i != row)
                        {
                                if(abs(a[i][col]) < eps) continue;
                double c = a[i][col] / a[row][col];
                for(int j = 0; j <= m; j++)
                    a[i][j] -= c * a[row][j];
                        }

                row++;
    }

    vector<double> ans(m, 0);
    for(int i = 0; i < m; i++)
        if(where[i] != -1)
                        ans[i] = a[where[i]][m] / a[where[i]][i];

        // Validity check?
        // May need to remove the following code

    for(int i = 0; i < n; i++)
        {
                double sum = a[i][m];
                for(int j = 0; j < m; j++)
```

```
                sum -= ans[j] * a[i][j];

            if(abs(sum) > eps) return vector<double>();
        }

        return ans;
}
```

6.17 Karatsuba

```
#define MAX 131072 /// Must be a power of 2
#define MOD 1000000007

unsigned long long temp[128];
int ptr = 0, buffer[MAX * 6];

/// n is a power of 2
void karatsuba(int n, int *a, int *b, int *res){ /// hash = 829512
    int i, j, h;
        if (n < 17){ /// Reduce recursive calls by setting a threshold
        for (i = 0; i < (n + n); i++) temp[i] = 0;
        for (i = 0; i < n; i++){
            if (a[i]){
                for (j = 0; j < n; j++){
                    temp[i + j] += ((long long)a[i] * b[j]);
                }
            }
        }
            for (i = 0; i < (n + n); i++) res[i] = temp[i] % MOD;
            return;
        }

        h = n >> 1;
        karatsuba(h, a, b, res);
        karatsuba(h, a + h, b + h, res + n);
        int *x = buffer + ptr, *y = buffer + ptr + h, *z = buffer + ptr +
            h + h;

        ptr += (h + h + n);
        for (i = 0; i < h; i++){
        x[i] = a[i] + a[i + h], y[i] = b[i] + b[i + h];
        if (x[i] >= MOD) x[i] -= MOD;
        if (y[i] >= MOD) y[i] -= MOD;
        }

        karatsuba(h, x, y, z);
        for (i = 0; i < n; i++) z[i] -= (res[i] + res[i + n]);
        for (i = 0; i < n; i++){
        res[i + h] = (res[i + h] + z[i]) % MOD;
        if (res[i + h] < 0) res[i + h] += MOD;
        }
        ptr -= (h + h + n);
}

/// multiplies two polynomial a(degree n) and b(degree m) and returns the
    result modulo MOD in a
/// returns the degree of the multiplied polynomial
/// note that a and b are changed in the process

int mul(int n, int *a, int m, int *b){ /// hash = 903808
    int i, r, c = (n < m ? n : m), d = (n > m ? n : m), *res = buffer +
        ptr;
    r = 1 << (32 - __builtin_clz(d) - (__builtin_popcount(d) == 1));
    for (i = d; i < r; i++) a[i] = b[i] = 0;
    for (i = c; i < d && n < m; i++) a[i] = 0;
    for (i = c; i < d && m < n; i++) b[i] = 0;

    ptr += (r << 1), karatsuba(r, a, b, res), ptr -= (r << 1);
    for (i = 0; i < (r << 1); i++) a[i] = res[i];
    return (n + m - 1);
}

int a[MAX * 2], b[MAX * 2];

int main(){
    int i, j, k, n = MAX - 10;
    for (i = 0; i < n; i++) a[i] = ran(1, 1000000000);
    for (i = 0; i < n; i++) b[i] = ran(1, 991929183);
    clock_t start = clock();
    mul(n, a, n, b);
    dbg(a[n / 2]);
    for (i = 0; i < (n << 1); i++){
        if (a[i] < 0) puts("YO");
    }
    printf("%0.5f\n", (clock() - start) / (1.0 * CLOCKS_PER_SEC));
    return 0;
}
```

6.18 Linear Diophantine

```
int extended_euclid(int a, int b, int &x, int &y) {
    int xx = y = 0;
    int yy = x = 1;
    while (b) {
        int q = a / b;
        int t = b; b = a%b; a = t;
        t = xx; xx = x - q*xx; x = t;
        t = yy; yy = y - q*yy; y = t;
    }
    return a;
}
// Linear Diophantine Equation Solution: Given, a*x+b*y=c. Find valid x
    and y if possible.
bool linear_diophantine (int a, int b, int c, int & x0, int & y0, int &
    g) {
    g = extended_euclid (abs(a), abs(b), x0, y0);
    if (c % g != 0)
        return false;
    x0 *= c / g;
    y0 *= c / g;
    if (a < 0)  x0 *= -1;
    if (b < 0)  y0 *= -1;
    return true;
}
// for each integer k, // x1 = x + k * b/g // y1 = y - k * a/g
// is a solution to the equation where g = gcd(a,b).
void shift_solution (int & x, int & y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}
// Now How many solution where x in range[x1,x2] and y in range[y1,y2] ?
int find_all_solutions(int a,int b,int c,int &minx,int &maxx,int
    &miny,int &maxy)
{
    int x,y,g;
    if(linear_diophantine(a,b,c,x,y,g) == 0) return 0;
    a/=g, b/=g;
    int sign_a = a>0 ? +1 : -1;
    int sign_b = b>0 ? +1 : -1;
    shift_solution (x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution (x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;
    shift_solution (x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution (x, y, a, b, -sign_b);
    int rx1 = x;
    shift_solution (x, y, a, b, - (miny - y) / a);
    if (y < miny)
        shift_solution (x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;
    shift_solution (x, y, a, b, - (maxy - y) / a);
    if (y > maxy)
        shift_solution (x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap (lx2, rx2);
    int lx = max (lx1, lx2);
    int rx = min (rx1, rx2);

    return (rx - lx) / abs(b) + 1;
}
```

6.19 Matrix Expo

```
struct Matrix
{
        ll mat[MAX][MAX];
        Matrix(){}
        // This initialization is important.
        // Input matrix should be initialized separately
        void init(int sz)
        {
                ms(mat,0);
                for(int i=0; i<sz; i++) mat[i][i]=1;
        }
} aux;

void matMult(Matrix &m, Matrix &m1, Matrix &m2, int sz)
{
        ms(m.mat,0);
```

```
	// This only works for square matrix
	FOR(i,0,sz)
	{
		FOR(j,0,sz)
		{
			FOR(k,0,sz)
			{
				m.mat[i][k]=(m.mat[i][k]+m1.mat[i][j]*m2.mat[j][k])%mod;
			}
		}
	}
	/* We can also do this if MOD*MOD fits long long
	long long MOD2 = MOD * MOD;
	for(int i = 0; i < n; i++)
		for(int j = 0; j < n; j++) {
			long long tmp = 0;
				for(int k = 0; k < n; k++) {
				// Since A and B are taken modulo MOD, the
				    product A[i][k] * B[k][j] is
				// not more than MOD * MOD.
				tmp += A[i][k] * 1ll * B[k][j];
				while(tmp >= MOD2) // Taking modulo MOD2 is
				    easy, because we can do it by subtraction
				tmp -= MOD2;
			}
		result[i][j] = tmp % MOD; // One % operation per resulting
		    element
		}
	*/
}

Matrix expo(Matrix &M, int n, int sz)
{
	Matrix ret;
	ret.init(sz);

	if(n==0) return ret;
	if(n==1) return M;

	Matrix P=M;

	while(n!=0)
	{
		if(n&1)
		{
			aux=ret;
			matMult(ret,aux,P,sz);
		}

		n>>=1;

		aux=P; matMult(P,aux,aux,sz);
	}

	return ret;
}
```

6.20 Number Theoretic Transform

```
const ll mod=786433;

vi getdivs(int p)
{
	int q=p-1;
	vi div;
	for(int j=2; j*j<=q; j++)
	{
		if(q%j==0)
		{
			div.pb(j);
			while(q%j==0) q/=j;
		}
	}
	if(q!=1) div.pb(q);
	return div;
}

bool check(int e, int p, vi divs)
{
	for(auto d: divs)
	{
		if(bigmod((ll)e,(ll)(p-1)/d,(ll)p)==1)
			return false;
	}
	return true;
}

int getRoot(int p)
```

```
{
	int e=2;
	vi divs=getdivs(p);
	while(!check(e,p,divs)) e++;
	return e;
}

/* getRoot(mod) returns a value which is used as prr in the following code
 and G in the next one */
// Code 1
ll ipow(ll a, ll b, ll m = mod)
{
	ll ret = 1;
	while (b)
	{
		if (b & 1) ret = ret * a % m;
		a = a * a % m;
		b >>= 1;
	}
	return ret;
}
namespace fft{
	typedef ll base;
	void fft(vector<base> &a, bool inv){
		int n = a.size(), j = 0;
		vector<base> roots(n/2);
		for(int i=1; i<n; i++){
			int bit = (n >> 1);
			while(j >= bit){
				j -= bit;
				bit >>= 1;
			}
			j += bit;
			if(i < j) swap(a[i], a[j]);
		}
		int prr = 10; // Got from calling getRoot(mod);
		int ang = ipow(prr, (mod - 1) / n);
		if(inv) ang = ipow(ang, mod - 2);
		for(int i=0; i<n/2; i++){
			roots[i] = (i ? (1ll * roots[i-1] * ang % mod) : 1);
		}
		for(int i=2; i<=n; i<<=1){
			int step = n / i;
			for(int j=0; j<n; j+=i){
				for(int k=0; k<i/2; k++){
					base u = a[j+k], v = a[j+k+i/2] *
					    roots[step * k] % mod;
					a[j+k] = (u+v+mod)% mod;
					a[j+k+i/2] = (u-v+mod)%mod;
				}
			}
		}
		if(inv) for(int i=0; i<n; i++) a[i] *= ipow(n, mod-2),
		    a[i] %= mod;
	}
	vector<ll> multiply(vector<ll> &v, vector<ll> &w){
		vector<base> fv(v.begin(), v.end()), fw(w.begin(),
		    w.end());
		int n = 2; while(n < v.size() + w.size()) n <<= 1;
		fv.resize(n); fw.resize(n);
		fft(fv, 0); fft(fw, 0);
		for(int i=0; i<n; i++) fv[i] *= fw[i];
		fft(fv, 1);
		vector<ll> ret(n);
		for(int i=0; i<n; i++) ret[i] = fv[i];
		return ret;
	}
}

// Code 2
struct NTT
{
	vi A, B, w[2], rev;
	ll P, M, G;
	NTT(ll mod) {P=mod; G=10;}
	void init(ll n)
	{
		for(M=2; M<n; M<<=1);
		M<<=1;
		A.resize(M); B.resize(M);
		w[0].resize(M); w[1].resize(M); rev.resize(M);

		for(ll i=0; i<M; i++)
		{
			ll x=i, &y=rev[i];
			y=0;
			for(ll k=1; k<M; k<<=1, x>>=1)
				(y<<=1)|=(x&1);
		}
```

```
        ll x=bigmod(G,(P-1)/M,mod);
        ll y=bigmod(x,P-2,mod);

        w[0][0]=w[1][0]=1LL;

        for(ll i=1; i<M; i++)
        {
                w[0][i]=(w[0][i-1]*x)%P;
                w[1][i]=(w[1][i-1]*y)%P;
        }
}
void ntransform(vector<ll> &a, ll f)
{
        for(ll i=0; i<M; i++)
        {
                if(i<rev[i]) swap(a[i], a[rev[i]]);
        }
        for(ll i=1; i<M; i<<=1)
        {
                for(ll j=0, t=M/(i<<1); j<M; j+=(i<<1))
                {
                        for(ll k=0, l=0; k<i; k++, l+=t)
                        {
                                ll x=a[j+k+i]*1LL*w[f][l]%P;
                                ll y=a[j+k];
                                a[j+k+i]=y-x<0?y-x+P:y-x;
                                a[j+k]=y+x>=P?y+x-P:y+x;
                        }
                }
        }
        if(f)
        {
                ll x=bigmod(M,P-2,mod);
                for(ll i=0; i<M; i++) a[i]=a[i]*1LL*x%P;
        }
}
void multiply(vector<ll> &X, vector<ll> &Y, vector<ll> &res)
{
        init(max(X.size(),Y.size()));
        for(ll i=0; i<M; i++) A[i]=B[i]=0;
        for(ll i=0; i<X.size(); i++) A[i]=X[i];
        for(ll i=0; i<Y.size(); i++) B[i]=Y[i];
        ntransform(A,0);
        ntransform(B,0);
        res.clear();
                res.resize(M);
                for(ll i=0; i<M; i++) res[i]=A[i]*1LL*B[i]%P;
                ntransform(res,1);
        }
};
```

6.21 Segmented Sieve

```
#define MAX 1000010

#define BASE_SQR 216
#define BASE_LEN 10010
#define BASE_MAX 46656
#define chkbit(ar, i) (((ar[(i) >> 6]) & (1 << (((i) >> 1) & 31))))
#define setbit(ar, i) (((ar[(i) >> 6]) |= (1 << (((i) >> 1) & 31))))

int p, primes[BASE_LEN];
unsigned int base[(BASE_MAX >> 6) + 5], isprime[(MAX >> 6) + 5];

void Sieve(){
    clr(base);
    int i, j, k;

    for (i = 3; i < BASE_SQR; i++, i++){
        if (!chkbit(base, i)){
            k = i << 1;
            for (j = (i * i); j < BASE_MAX; j += k){
                setbit(base, j);
            }
        }
    }

    p = 0;
    for (i = 3; i < BASE_MAX; i++, i++){
        if (!chkbit(base, i)){
            primes[p++] = i;
        }
    }
}

int SegmentedSieve(long long a, long long b){
    long long j, k, x;
    int i, d, counter = 0;
```

```
    if (a <= 2 && 2 <= b) counter = 1; /// 2 is counted separately if in
        range
    if (!(a & 1)) a++;
    if (!(b & 1)) b--;
    if (a > b) return counter;

    clr(isprime);
    for (i = 0; i < p; i++){
        x = primes[i];
        if ((x * x) > b) break;

        k = x << 1;
        j = x * ((a + x - 1) / x);
        if (!(j & 1)) j += x;
        else if (j == x) j += k;

        while (j <= b){
            setbit(isprime, j - a);
            j += k;
        }
    }

    /// Other primes in the range except 2 are added here
    d = (b - a + 1);
    for (i = 0; i < d; i++, i++){
        if (!chkbit(isprime, i) && (a + i) != 1) counter++;
    }

    return counter;
}

int main(){
    Sieve();
    int T = 0, t, i, j, a, b;

    scanf("%d", &t);
    while (t--){
        scanf("%d %d", &a, &b);
        printf("Case %d: %d\n", ++T, SegmentedSieve(a, b));
    }
    return 0;
}
```

6.22 Sieve (Bitmask)

```
#define LEN 78777
#define MAX 1000010
#define chkbit(ar, i) (((ar[(i) >> 6]) & (1 << (((i) >> 1) & 31))))
#define setbit(ar, i) (((ar[(i) >> 6]) |= (1 << (((i) >> 1) & 31))))
#define isprime(x) (( (x) && ((x)&1) && (!chkbit(ar, (x)))) || ((x) == 2))

int p, prime[LEN];
unsigned int ar[(MAX >> 6) + 5] = {0};

void Sieve(){
    int i, j, k;
    setbit(ar, 0), setbit(ar, 1);

    for (i = 3; (i * i) < MAX; i++, i++){
        if (!chkbit(ar, i)){
            k = i << 1;
            for (j = (i * i); j < MAX; j += k) setbit(ar, j);
        }
    }

    p = 0;
    prime[p++] = 2;
    for (i = 3; i < MAX; i++, i++){
        if (isprime(i)) prime[p++] = i;
    }
}

int main(){
    Sieve();
    printf("%d\n", p);
    int i;
    for (i = 0; i < 60; i++){
        if (isprime(i)) printf("%d\n", i);
    }
}
```

6.23 Sieve

```
vi primes;
bool status[MAX+7];
```

```
// Finds all the primes upto MAX

void sieve()
{
	for(int i=4; i<=MAX; i+=2)
		status[i]=true;

	for(int i=3; i*i<=MAX; i++)
	{
		if(!status[i])
		{
			for(int j=i*i; j<=MAX; j+=i+i)
				status[j]=true;
		}
	}

	primes.pb(2);

	FOR(i,3,MAX)
	{
		if(!status[i])
			primes.pb(i);
	}
}
```

6.24 Simplex

```
/*
 * Algorithm : Simplex ( Linear Programming )
 * Author : Simon Lo
 * Note: Simplex algorithm on augmented matrix a of dimension (m+1)x(n+1)
 * returns 1 if feasible, 0 if not feasible, -1 if unbounded
 * returns solution in b[] in original var order, max(f) in ret
 * form: maximize sum_j(a_mj*x_j)-a_mn s.t. sum_j(a_ij*x_j)<=a_in
 * in standard form.
 * To convert into standard form:
 * 1. if exists equality constraint, then replace by both >= and <=
 * 2. if variable x doesn't have nonnegativity constraint, then replace by
 * difference of 2 variables like x1-x2, where x1>=0, x2>=0
 * 3. for a>=b constraints, convert to -a<=-b
 * note: watch out for -0.0 in the solution, algorithm may cycle
 * EPS = 1e-7 may give wrong answer, 1e-10 is better
 */

#define MAX 107
#define INF 1000000007
#define EPS (1e-12)

void Pivot( long m,long n,double A[MAX+7][MAX+7],long *B,long *N,long
    r,long c )
{
    long i,j;
    swap( N[c],B[r] );
    A[r][c] = 1/A[r][c];
    for( j=0;j<=n;j++ ) if( j!=c ) A[r][j] *= A[r][c];
    for( i=0;i<=m;i++ ){
        if( i!=r ){
            for( j=0;j<=n;j++ ) if( j!=c ) A[i][j] -= A[i][c]*A[r][j];
            A[i][c] = -A[i][c]*A[r][c];
        }
    }
}

long Feasible( long m,long n,double A[MAX+7][MAX+7],long *B,long *N )
{
    long r,c,i;
    double p,v;
    while( 1 ){
        for( p=INF,i=0;i<m;i++ ) if( A[i][n]<p ) p = A[r=i][n];
        if( p > -EPS ) return 1;
        for( p=0,i=0;i<n;i++ ) if( A[r][i]<p ) p = A[r][c=i];
        if( p > -EPS ) return 0;
        p = A[r][n]/A[r][c];
        for( i=r+1;i<m;i++ ){
            if( A[i][c] > EPS ){
                v = A[i][n]/A[i][c];
                if( v<p ) r=i,p=v;
            }
        }
        Pivot( m,n,A,B,N,r,c );
    }
}

long Simplex( long m,long n,double A[MAX+7][MAX+7],double *b,double &Ret )
{
    long B[MAX+7],N[MAX+7],r,c,i;
    double p,v;
    for( i=0;i<n;i++ ) N[i] = i;
```

```
    for( i=0;i<m;i++ ) B[i] = n+i;
    if( !Feasible( m,n,A,B,N ) ) return 0;
    while( 1 ){
        for( p=0,i=0;i<n;i++ ) if( A[m][i] > p ) p = A[m][c=i];
        if( p<EPS ){
            for( i=0;i<n;i++ ) if( N[i]<n ) b[N[i]] = 0;
            for( i=0;i<m;i++ ) if( B[i]<n ) b[B[i]] = A[i][n];
            Ret = -A[m][n];
            return 1;
        }
        for( p=INF,i=0;i<m;i++ ){
            if( A[i][c] > EPS ){
                v = A[i][n]/A[i][c];
                if( v<p ) p = v,r = i;
            }
        }
        if( p==INF ) return -1;
        Pivot( m,n,A,B,N,r,c );
    }
}

// Caution: long double can give TLE
typedef long double ld;
typedef vector<ld> vd;
typedef vector<vd> vvd;

const ld EPS = 1e-10;

struct LPSolver {
  int m, n;
  vi B, N;
  vvd D;

  LPSolver(const vvd &A, const vd &b, const vd &c) :
    m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, vd(n + 2)) {
    for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] =
        A[i][j];
    for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n + 1]
        = b[i]; }
    for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
    N[n] = -1; D[m + 1][n] = 1;
  }

  void Pivot(int r, int s) {
    ld inv = 1.0 / D[r][s];
    for (int i = 0; i < m + 2; i++) if (i != r)
        for (int j = 0; j < n + 2; j++) if (j != s)
            D[i][j] -= D[r][j] * D[i][s] * inv;
    for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
    for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }

  bool Simplex(int phase) {
    int x = phase == 1 ? m + 1 : m;
    while (true) {
      int s = -1;
      for (int j = 0; j <= n; j++) {
        if (phase == 2 && N[j] == -1) continue;
        if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] <
            N[s]) s = j;
      }
      if (D[x][s] > -EPS) return true;
      int r = -1;
      for (int i = 0; i < m; i++) {
        if (D[i][s] < EPS) continue;
        if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
            (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i]
                < B[r]) r = i;
      }
      if (r == -1) return false;
      Pivot(r, s);
    }
  }

  ld Solve(vd &x) {
    int r = 0;
    for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
    if (D[r][n + 1] < -EPS) {
      Pivot(r, n);
      if (!Simplex(1) || D[m + 1][n + 1] < -EPS) return
          -numeric_limits<ld>::infinity();
      for (int i = 0; i < m; i++) if (B[i] == -1) {
          int s = -1;
          for (int j = 0; j <= n; j++)
            if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[j]
                < N[s]) s = j;
          Pivot(i, s);
        }
```

```
    }
    if (!Simplex(2)) return numeric_limits<ld>::infinity();
    x = vd(n);
    for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
    return D[m][n + 1];
  }
};

/* Equations are of the matrix form Ax<=b, and we want to maximize
the function c. We are given coeffs of A, b and c. In case of minimizing,
we negate the coeffs of c and maximize it. Then the negative of returned
'value' is the answer.
All the constraints should be in <= form. So we may need to negate the
coeffs.
*/

int main() {

  const int m = 4;
  const int n = 3;
  ld _A[m][n] = {
    { 6, -1, 0 },
    { -1, -5, 0 },
    { 1, 5, 1 },
    { -1, -5, -1 }
  };
  ld _b[m] = { 10, -4, 5, -5 };
  ld _c[n] = { 1, -1, 0 };

  vvd A(m);
  vd b(_b, _b + m);
  vd c(_c, _c + n);
  for (int i = 0; i < m; i++) A[i] = vd(_A[i], _A[i] + n);

  LPSolver solver(A, b, c);
  vd x;
  ld value = solver.Solve(x);

  cerr << "VALUE: " << value << endl; // VALUE: 1.29032
  cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
  for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
  cerr << endl;
  return 0;
}
```

6.25 Sum of Kth Power

```
LL mod;
LL S[105][105];
// Find 1^k+2^k+...+n^k % mod
void solve() {
        LL n, k;
        scanf("%lld %lld %lld", &n, &k, &mod);
        /*
        x^k = sum (i=1 to k) Stirling2(k, i) * i! * ncr(x, i)
        sum (x = 0 to n) x^k
                = sum (i = 0 to k) Stirling2(k, i) * i! * sum (x = 0 to n)
                    ncr(x, i)
                = sum (i = 0 to k) Stirling2(k, i) * i! * ncr(n + 1, i + 1)
                = sum (i = 0 to k) Stirling2(k, i) * i! * (n + 1)! / (i +
                    1)! / (n - i)!
                = sum (i = 0 to k) Stirling2(k, i) * (n - i + 1) * (n - i
                    + 2) * ... (n + 1) / (i + 1)
        */
        S[0][0] = 1 % mod;
        for (int i = 1; i <= k; i++) {
                for (int j = 1; j <= i; j++) {
                        if (i == j) S[i][j] = 1 % mod;
                        else S[i][j] = (j * S[i - 1][j] + S[i - 1][j - 1])
                            % mod;
                }
        }

        LL ans = 0;
        for (int i = 0; i <= k; i++) {
                LL fact = 1, z = i + 1;
                for (LL j = n - i + 1; j <= n + 1; j++) {
                        LL mul = j;
                        if (mul % z == 0) {
                                mul /= z;
                                z /= z;
                        }
                        fact = (fact * mul) % mod;
                }
                ans = (ans + S[k][i] * fact) % mod;
        }
        printf("%lld\n", ans);
}
```

7 Miscellaneous

7.1 Bit Hacks

```
unsigned int reverse_bits(unsigned int v){
    v = ((v >> 1) & 0x55555555) | ((v & 0x55555555) << 1);
    v = ((v >> 2) & 0x33333333) | ((v & 0x33333333) << 2);
    v = ((v >> 4) & 0x0F0F0F0F) | ((v & 0x0F0F0F0F) << 4);
    v = ((v >> 8) & 0x00FF00FF) | ((v & 0x00FF00FF) << 8);
    return ((v >> 16) | (v << 16));
}

/// Returns i if x = 2^i and 0 otherwise
int bitscan(unsigned int x){
    __asm__ volatile("bsf %0, %0" : "=r" (x) : "0" (x));
    return x;
}

/// Returns next number with same number of 1 bits
unsigned int next_combination(unsigned int x){
    unsigned int y = x & -x;
    x += y;
    unsigned int z = x & -x;
    z -= y;
    z = z >> bitscan(z & -z);
    return x | (z >> 1);
}

int main(){
}
```

7.2 Divide and Conquer on Queries

```
/* You are given an array a[] of size n, an integer m and bunch of
    queries (l,r).
For each query, you have to answer the number of subsequences of the
    subarray (a[l]...a[r])
whose sum is divisible by m.
*/
const int N = 2 * MAX + 7;
int n, m, a[N];
int input[N][3]; // inputs queries, answer sored in input[i][2]
int dp_left[N][20], dp_right[N][20];
int ans[N];

void solve(int L, int R, vi all)
{
        if(L>R || all.empty()) return;
        // initialize only this range
        FOR(i,L,R+1) FOR(j,0,m) dp_left[i][j] = 0, dp_right[i][j] = 0;

        int mid = (L+R)/2;

        dp_left[mid][0] = 1;
        dp_right[mid-1][0] = 1;
        // calculate the number of subsequences starting from mid-1 to L
            in dp_left
        for(int i=mid-1; i>=L; i--)
        {
                for(int j=0; j<m; j++)
                {
                        int taken = (a[i]+j)%m;

                        dp_left[i][taken] = (dp_left[i][taken] +
                            dp_left[i+1][j]) % mod;
                        dp_left[i][j] = (dp_left[i][j] + dp_left[i+1][j]) %
                            mod;
                }
        }
        // calculate the number of subsequences starting from mid to R in
            dp_right
        for(int i=mid; i<=R; i++)
        {
                for(int j=0; j<m; j++)
                {
                        int taken = (a[i]+j)%m;

                        dp_right[i][taken] = (dp_right[i][taken] +
                            dp_right[i-1][j]) % mod;
                        dp_right[i][j] = (dp_right[i][j] +
                            dp_right[i-1][j]) % mod;
                }
        }

        vi ls, rs;
        for(auto idx: all)
        {
```

```
        int l = input[idx][0], r = input[idx][1];

        if(l>mid) rs.pb(idx);
        else if(r<mid) ls.pb(idx);
        else
        {
                if(l==r && l==mid) // query is just on mid,
                    specially handled
                {
                        ans[idx] = ((a[mid] % m == 0) ? 2: 1);
                }
                else if(l==mid) // starts from mid
                {
                        ans[idx] = dp_right[r][0];
                }
                else if(r==mid) // ends in mid
                {
                        int rem = a[mid] % m;

                        ans[idx] = dp_left[l][0];
                        ans[idx] = (ans[idx] +
                            dp_left[l][(m-rem)%m]) % mod;
                }
                else
                {
                        // merge both sides and calculate answer for
                            current query
                        for(int j=0; j<m; j++)
                        {
                                ans[idx] = (ans[idx] + (dp_left[l][j]
                                    * dp_right[r][(m-j)%m]) % mod) %
                                    mod;
                        }
                }
        }
    }
    // find answer for other queries by divide and conquer
    solve(L,mid,ls);
    solve(mid+1,R,rs);
}
```

7.3 Gilbert Curve for Mo

```
inline int64_t gilbertOrder(int x, int y, int pow, int rotate) {
        if (pow == 0) {
                return 0;
        }
        int hpow = 1 << (pow-1);
        int seg = (x < hpow) ? (
                (y < hpow) ? 0 : 3
        ) : (
                (y < hpow) ? 1 : 2
        );
        seg = (seg + rotate) & 3;
        const int rotateDelta[4] = {3, 0, 0, 1};
        int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
        int nrot = (rotate + rotateDelta[seg]) & 3;
        int64_t subSquareSize = int64_t(1) << (2*pow - 2);
        int64_t ans = seg * subSquareSize;
        int64_t add = gilbertOrder(nx, ny, pow-1, nrot);
        ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add - 1);
        return ans;
}

struct Query {
        int l, r, idx; // queries
        int64_t ord; // Gilbert order of a query
        // call query[i].calcOrder() to calculate the Gilbert orders
        inline void calcOrder() {
                ord = gilbertOrder(l, r, 21, 0);
        }
};
// sort the queries based on the Gilbert order
inline bool operator<(const Query &a, const Query &b) {
        return a.ord < b.ord;
}
```

7.4 HakmemItem175

```
/// Only for non-negative integers
/// Returns the immediate next number with same count of one bits, -1 on
    failure
long long hakmemItem175(long long n){
    if (n == 0) return -1;
    long long x = (n & -n);
    long long left = (x + n);
```

```
    long long right = ((n ^ left) / x) >> 2;
    long long res = (left | right);
    return res;
}

/// Returns the immediate previous number with same count of one bits, -1
    on failure
long long lol(long long n){
    if (n == 0 || n == 1) return -1;
    long long res = ~hakmemItem175(~n);
    return (res == 0) ? -1 : res;
}
```

7.5 Header

```
// g++ -O2 -static -std=c++11 source.cpp
#pragma comment(linker, "/stack:200000000")
#pragma GCC optimize("unroll-loops")

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef vector <int> vi;
typedef vector <string> vs;
typedef pair <int, int> pii;
typedef vector<pii > vpii;

#define MP make_pair
#define SORT(a) sort (a.begin(), a.end())
#define REVERSE(a) reverse (a.begin(), a.end())
#define ALL(a) a.begin(), a.end()
#define PI acos(-1)
#define ms(x,y) memset (x, y, sizeof (x))
#define inf 1e9
#define INF 1e16
#define pb push_back
#define MAX 100005
#define debug(a,b) cout<<a<<": "<<b<<endl
#define Debug cout<<"Reached here"<<endl
#define prnt(a) cout<<a<<"\n"
#define mod 1000000007LL
#define FOR(i,a,b) for (int i=(a); i<(b); i++)
#define FORr(i,a,b) for (int i=(a); i>=(b); i--)
#define itrALL(c,itr) for(__typeof((c).begin())
    itr=(c).begin();itr!=(c).end();itr++)
#define lc ((node)<<1)
#define rc ((node)<<1|1)
#define VecPrnt(v) FOR(J,0,v.size()) cout<<v[J]<<" "; cout<<endl
#define endl "\n"
#define PrintPair(x) cout<<x.first<<" "<<x.second<<endl
#define EPS 1e-9
#define ArrPrint(a,st,en) for(int J=st; J<=en; J++) cout<<a[J]<<" ";
    cout<<endl;

/* Direction Array */

// int fx[]={1,-1,0,0};
// int fy[]={0,0,1,-1};
// int fx[]={0,0,1,-1,-1,1,-1,1};
// int fy[]={-1,1,0,0,1,1,-1,-1};

/***************** END OF HEADER *****************/

int main()
{
    // ios_base::sync_with_stdio(0);
    // cin.tie(NULL); cout.tie(NULL);
    // freopen("in.txt","r",stdin);

    int test, cases = 1;

    return 0;
}
```

7.6 Integral Determinant

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 1010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
```

```
const long long MOD = 4517409488245517117LL;
const long double OP = (long double)1 / 4517409488245517117LL;

long long mul(long long a, long long b){
   long double res = a;
   res *= b;
   long long c = (long long)(res * OP);
   a *= b;
   a -= c * MOD;
   if (a >= MOD) a -= MOD;
   if (a < 0) a += MOD;
   return a;
}

long long expo(long long x, long long n){
    long long res = 1;

    while (n){
        if (n & 1) res = mul(res, x);
        x = mul(x, x);
        n >>= 1;
    }

    return res;
}

int gauss(int n, long long ar[MAX][MAX]){
    long long x, y;
    int i, j, k, l, p, counter = 0;

    for (i = 0; i < n; i++){
        for (p = i, j = i + 1; j < n && !ar[p][i]; j++){
            p = j;
        }
        if (!ar[p][i]) return -1;

        for (j = i; j < n; j++){
            x = ar[p][j], ar[p][j] = ar[i][j], ar[i][j] = x;
        }

        if (p != i) counter++;
        for (j = i + 1; j < n; j++){
            x = expo(ar[i][i], MOD - 2);
            x = mul(x, MOD - ar[j][i]);

            for (k = i; k < n; k++){
                ar[j][k] = ar[j][k] + mul(x, ar[i][k]);
                if (ar[j][k] >= MOD) ar[j][k] -= MOD;
            }
        }
    }
    return counter;
}

/// Finds the determinant of a square matrix
/// Returns 0 if the matrix is singular or degenerate (hence no
    determinant exists)
/// Absolute value of final answer should be < MOD / 2

long long determinant(int n, long long ar[MAX][MAX]){
    int i, j, free;
    long long res = 1;

    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            if (ar[i][j] < 0) ar[i][j] += MOD;
        }
    }

    free = gauss(n, ar);
    if (free == -1) return 0; /// Determinant is 0 so matrix is not
        invertible, singular or degenerate matrix

    for (i = 0; i < n; i++) res = mul(res, ar[i][i]);
    if (free & 1) res = MOD - res;
    if ((MOD - res) < res) res -= MOD; /// Determinant can be negative so
        if determinant is more close to MOD than 0, make it negative

    return res;
}

int n;
long long ar[MAX][MAX];

int main(){
    int t, i, j, k, l;

    while (scanf("%d", &n) != EOF){
        if (n == 0) break;
```

```
        for (i = 0; i < n; i++){
            for (j = 0; j < n; j++){
                scanf("%lld", &ar[i][j]);
            }
        }

        printf("%lld\n", determinant(n, ar));
    }
    return 0;
}
```

7.7 Inverse Modulo 1 to N (Linear)

```
int fact[MAX], inv[MAX];
int expo(int a, int b){
    int res = 1;

    while (b){
        if (b & 1) res = (long long)res * a % MOD;
        a = (long long)a * a % MOD;
        b >>= 1;
    }
    return res;
}
void Generate(){
    int i, x;
    for (fact[0] = 1, i = 1; i < MAX; i++) fact[i] = ((long long)i *
        fact[i - 1]) % MOD;

    /// inv[i] = Inverse modulo of fact[i]
    inv[MAX - 1] = expo(fact[MAX - 1], MOD - 2);
    for (i = MAX - 2; i >= 0; i--) inv[i] = ((long long)inv[i + 1] * (i +
        1)) % MOD;

    /// Inverse modulo of numbers 1 to MAX in linear time below
    inv[1] = 1;
        for (i = 2; i < MAX; i++){
        inv[i] = MOD - ((MOD / i) * (long long)inv[MOD % i]) % MOD;
        if (inv[i] < 0) inv[i] += MOD;
        }
}

int main(){
    Generate();
    printf("%d\n", inv[35]);
    printf("%d\n", expo(fact[35], MOD - 2));
    return 0;
}
```

7.8 Josephus Problem

```
/// Josephus problem, n people numbered from 1 to n stand in a circle.
/// Counting starts from 1 and every k'th people dies
/// Returns the position of the m'th killed people
/// For example if n = 10 and k = 3, then the people killed are 3, 6, 9,
    2, 7, 1, 8, 5, 10, 4 respectively

/// O(n)
int josephus(int n, int k, int m){
    int i;
    for (m = n - m, i = m + 1; i <= n; i++){
        m += k;
        if (m >= i) m %= i;
    }
    return m + 1;
}

/// O(k log(n))
long long josephus2(long long n, long long k, long long m){ /// hash =
    583016
    m = n - m;
    if (k <= 1) return n - m;

    long long i = m;
    while (i < n){
        long long r = (i - m + k - 2) / (k - 1);
        if ((i + r) > n) r = n - i;
        else if (!r) r = 1;
        i += r;
        m = (m + (r * k)) % i;
    }
    return m + 1;
}

int main(){
    int n, k, m;
```

```
    printf("%d\n", josephus(10, 1, 2));
    printf("%d\n", josephus(10, 1, 10));
}
```

7.9 MSB Position in O(1)

```
int msb(unsigned x)
{
        union {
                double a; int b[2];
        };
        a = x;
        return (b[1] >> 20) - 1023;
}
```

7.10 Nearest Smaller Values on Left-Right

```
// Linear time all nearest smaller values, standard stack-based algorithm.
// ansv_left stores indices of nearest smaller values to the left in res.
    -1 means no smaller value was found.
// ansv_right likewise looks to the right. v.size() means no smaller
    value was found.
void ansv_left(vector<int>& v, vector<int>& res) {
        stack<pair<int, int> > stk; stk.push(make_pair(INT_MIN, v.size()));
        for (int i = v.size()-1; i >= 0; i--) {
                while (stk.top().first > v[i]) {
                        res[stk.top().second] = i; stk.pop();
                }
                stk.push(make_pair(v[i], i));
        }
        while (stk.top().second < v.size()) {
                res[stk.top().second] = -1; stk.pop();
        }
}

void ansv_right(vector<int>& v, vector<int>& res) {
        stack<pair<int, int> > stk; stk.push(make_pair(INT_MIN, -1));
        for (int i = 0; i < v.size(); i++) {
                while (stk.top().first > v[i]) {
                        res[stk.top().second] = i; stk.pop();
                }
                stk.push(make_pair(v[i], i));
        }
        while (stk.top().second > -1) {
                res[stk.top().second] = v.size(); stk.pop();
        }
}
```

7.11 Next Small

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 250010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int ar[MAX], L[MAX], R[MAX], stack[MAX], time[MAX];

void next_small(int n, int* ar, int* L){
    int i, j, k, l, x, top = 0;

    for (i = 0; i < n; i++){
        x = ar[i];
        if (top && stack[top] >= x){
            while (top && stack[top] >= x) k = time[top--];
            L[i] = (i - k + 2);
            stack[++top] = x;
            time[top] = k;
        }
        else{
            L[i] = 1;
            stack[++top] = x;
            time[top] = i + 1;
        }
    }
}
/*** L[i] contains maximum length of the range from i to the left such
     that the minimum of this range
     is not less than ar[i].
     Similarly, R[i] contains maximum length of the range from i to the
         right such that the minimum
     of this range is not less than ar[i]
```

```
    For example, ar[] = 5 3 4 3 1 2 6
                L[]  = 1 2 1 4 5 1 1
                R[]  = 1 3 1 1 3 2 1
***/

void fill(int n, int* ar, int* L, int* R){
    int i, j, k;
    for (i = 0; i < n; i++) L[i] = ar[n - i - 1];

    next_small(n, L, R);
    next_small(n, ar, L);

    i = 0, j = n - 1;
    while (i < j){
        k = R[i], R[i] = R[j], R[j] = k;
        i++, j--;
    }
}

int main(){
    int n, i, j, k;

    scanf("%d", &n);
    for (i = 0; i < n; i++) scanf("%d", &ar[i]);

    fill(n, ar, L, R);
    for (i = 0; i < n; i++) printf("%d ", ar[i]);
    puts("");
    for (i = 0; i < n; i++) printf("%d ", R[i]);
    puts("");
    for (i = 0; i < n; i++) printf("%d ", L[i]);
    puts("");
    return 0;
}
```

7.12 Random Number Generation

```
#include <bits/stdc++.h>
#include <random>
#include <chrono>

using namespace std;

// Seeding non-deterministically
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

random_device rd;
mt19937 mt(rd());
uniform_real_distribution<double> r1(1.0, 10.0);
uniform_int_distribution<int> r2(1,INT_MAX);
normal_distribution<double> r3(1.0,10.0);
exponential_distribution<> r4(5);

int main()
{
        cout<<rng()<<endl;
        cout<<r1(mt)<<endl;
        cout<<r2(mt)<<endl;
        cout<<r4(mt)<<endl;
        return 0;
}
```

7.13 Russian Peasant Multiplication

```
// calculate (a*b)%m
// Particularly useful when a, b, m all are large like 1e18
ll RussianPeasantMultiplication(ll a, ll b, ll m)
{
        ll ret=0;

        while(b)
        {
                if(b&1)
                {
                        ret+=a;
                        if(ret>=m) ret-=m;
                }

                a=(a<<1);

                if(a>=m) a-=m;

                b>>=1;
        }
```

```
    return ret;
}
```

7.14 Stable Marriage Problem

```
// Gale-Shapley algorithm for the stable marriage problem.
// madj[i][j] is the jth highest ranked woman for man i.
// fpref[i][j] is the rank woman i assigns to man j.
// Returns a pair of vectors (mpart, fpart), where mpart[i] gives
// the partner of man i, and fpart is analogous
pair<vector<int>, vector<int> > stable_marriage(vector<vector<int> >&
    madj, vector<vector<int> >& fpref) {
    int n = madj.size();
    vector<int> mpart(n, -1), fpart(n, -1);
    vector<int> midx(n);
    queue<int> mfree;
    for (int i = 0; i < n; i++) {
        mfree.push(i);
    }
    while (!mfree.empty()) {
        int m = mfree.front(); mfree.pop();
        int f = madj[m][midx[m]++];
        if (fpart[f] == -1) {
            mpart[m] = f; fpart[f] = m;
        } else if (fpref[f][m] < fpref[f][fpart[f]]) {
            mpart[fpart[f]] = -1; mfree.push(fpart[f]);
            mpart[m] = f; fpart[f] = m;
        } else {
            mfree.push(m);
        }
    }
    return make_pair(mpart, fpart);
}
```

7.15 Thomas Algorithm

```
/// Equation of the form: (x_prev * l) + (x_cur * p) + (x_next * r) = rhs
struct equation{
    long double l, p, r, rhs;

    equation(){}
    equation(long double l, long double p, long double r, long double rhs
        = 0.0):
        l(l), p(p), r(r), rhs(rhs){}
};
/// Thomas algorithm to solve tri-digonal system of equations in O(n)
vector <long double> thomas_algorithm(int n, vector <struct equation> ar){
    ar[0].r = ar[0].r / ar[0].p;
    ar[0].rhs = ar[0].rhs / ar[0].p;
    for (int i = 1; i < n; i++){
        long double v = 1.0 / (ar[i].p - ar[i].l * ar[i - 1].r);
        ar[i].r = ar[i].r * v;
        ar[i].rhs = (ar[i].rhs - ar[i].l * ar[i - 1].rhs) * v;
    }
    for (int i = n - 2; i >= 0; i--) ar[i].rhs = ar[i].rhs - ar[i].r *
        ar[i + 1].rhs;
    vector <long double> res;
    for (int i = 0; i < n; i++) res.push_back(ar[i].rhs);
    return res;
}
```

7.16 U128

```
#include <bits/stdtr1c++.h>

using namespace std;

typedef unsigned long long int U64;

struct U128{
    U64 lo, hi;
    static const U64 bmax = -1;
    static const size_t sz = 128;
    static const size_t hsz = 64;

    inline U128() : lo(0), hi(0) {}
    inline U128(unsigned long long v) : lo(v), hi(0) {}

    inline U128 operator-() const {
        return ~U128(*this) + 1;
    }

    inline U128 operator~() const {
            U128 t(*this);
```

```
            t.lo = ~t.lo;
            t.hi = ~t.hi;
            return t;
    }

inline U128 &operator +=(const U128 &b) {
    if (lo > bmax - b.lo) ++hi;
    lo += b.lo;
    hi += b.hi;
    return *this;
}

inline U128 &operator -= (const U128 &b){
    return *this += -b;
}

inline U128 &operator *= (const U128 &b) {
    if (*this == 0 || b == 1) return *this;
    if (b == 0){
        lo = hi = 0;
        return *this;
    }

    U128 a(*this);
    U128 t = b;
    lo = hi = 0;

    for (size_t i = 0; i < sz; i++) {
        if((t & 1) != 0) *this += (a << i);
        t >>= 1;
    }
    return *this;
}

inline U128 &operator /= (const U128 &b) {
            U128 rem;
            divide(*this, b, *this, rem);
            return *this;
}

inline U128 &operator %= (const U128 &b) {
            U128 quo;
            divide(*this, b, quo, *this);
            return *this;
}
```

```
inline static void divide(const U128 &num, const U128 &den, U128
     &quo, U128 &rem) {
    if(den == 0) {
        int a = 0;
        quo = U128(a / a);
    }
    U128 n = num, d = den, x = 1, ans = 0;

    while((n >= d) && (((d >> (sz - 1)) & 1) == 0)) {
        x <<= 1;
        d <<= 1;
    }

    while(x != 0) {
        if(n >= d) {
            n -= d;
            ans |= x;
        }
        x >>= 1, d >>= 1;
    }
    quo = ans, rem = n;
}

inline U128 &operator&=(const U128 &b) {
    hi &= b.hi;
    lo &= b.lo;
    return *this;
}

inline U128 &operator|=(const U128 &b) {
    hi |= b.hi;
    lo |= b.lo;
    return *this;
}

inline U128 &operator<<=(const U128& rhs) {
            size_t n = rhs.to_int();
    if (n >= sz) {
        lo = hi = 0;
        return *this;
    }

    if(n >= hsz) {
        n -= hsz;
```

```
        hi = lo;
        lo = 0;
    }

    if(n != 0) {
        hi <<= n;
        const U64 mask(~(U64(-1) >> n));
        hi |= (lo & mask) >> (hsz - n);
        lo <<= n;
    }
    return *this;
    }

inline U128 &operator>>=(const U128& rhs) {
          size_t n = rhs.to_int();
    if (n >= sz) {
        lo = hi = 0;
        return *this;
    }

    if(n >= hsz) {
        n -= hsz;
        lo = hi;
        hi = 0;
    }

    if(n != 0) {
        lo >>= n;
        const U64 mask(~(U64(-1) << n));
        lo |= (hi & mask) << (hsz - n);
        hi >>= n;
    }

    return *this;
    }

inline int to_int() const { return static_cast<int> (lo); }
inline U64 to_U64() const { return lo; }

inline bool operator == (const U128 &b) const {
    return hi == b.hi && lo == b.lo;
}

inline bool operator != (const U128 &b) const {
    return !(*this == b);
}

inline bool operator < (const U128 &b) const {
    return (hi == b.hi) ? lo < b.lo : hi < b.hi;
}

inline bool operator >= (const U128 &b) const {
    return ! (*this < b);
}

inline U128 operator & (const U128 &b) const {
    U128 a(*this); return a &= b;
}

inline U128 operator << (const U128 &b) const {
    U128 a(*this); return a <<= b;
}

inline U128 operator >> (const U128 &b) const {
    U128 a(*this); return a >>= b;
}

inline U128 operator * (const U128 &b) const {
    U128 a(*this); return a *= b;
}

inline U128 operator + (const U128 &b) const {
    U128 a(*this); return a += b;
}

inline U128 operator - (const U128 &b) const {
    U128 a(*this); return a -= b;
}

inline U128 operator % (const U128 &b) const {
    U128 a(*this); return a %= b;
}

inline void print(){
    U128 x = *this;
    char str[128];
    int i, j, len = 0;

    do{
        str[len++] = (x % 10).lo + 48;
```

```
            x /= 10;
        } while (x != 0);

        reverse(str, str + len);
        str[len] = 0;
        puts(str);
    }
};

inline U128 gcd(U128 a, U128 b){
    if (b == 0) return a;
    return gcd(b, a % b);
}

inline U128 expo(U128 b, U128 e){
    U128 res = 1;
    while (e != 0){
        if ((e & 1) != 0) res *= b;
        e >>= 1, b *= b;
    }
    return res;
}

inline U128 expo(U128 x, U128 n, U128 m){
    U128 res = U128(1);
    while (n != 0){
        if ((n & 1) != 0){
            res *= n;
            res %= m;
        }
        x *= x;
        x %= m;
        n >>= 1;
    }
    return res % m;
}

struct Rational{
    U128 p, q;

    inline Rational(){
        p = 0, q = 1;
    }

    inline Rational(U128 P, U128 Q) : p(P), q(Q){
        simplify();
    }

    inline void simplify() {
        U128 g = gcd(p, q);
        p /= g;
        q /= g;
    }

    inline Rational operator+ (const Rational &f) const {
        return Rational(p * f.q + q * f.p, q * f.q);
    }
    inline Rational operator- (const Rational &f) const {
        return Rational(p * f.q - q * f.p, q * f.q);
    }
    inline Rational operator* (const Rational &f) const {
        return Rational(p * f.p, q * f.q);
    }
    inline Rational operator/ (const Rational &f) const {
        return Rational(p * f.q, q * f.p);
    }
};

int main(){
    U128 X = U128(9178291938173ULL);
    U128 Y = U128(123456789123456ULL);
    U128 M = U128(10000000000000000000ULL);

    for (int i = 0; i < 10000; i++){
        U128 R = expo(X, Y, M);
    }
    return 0;
}
```

7.17 Useful Templates

```
template <class T> inline T bigmod(T p, T e, T M)
{
    ll ret = 1;
    for (; e > 0; e >>= 1)
    {
        if (e & 1) ret = (ret * p) % M;
        p = (p * p) % M;
```

```
    } return (T)ret;
}

template <class T> inline T gcd(T a, T b) {if (b == 0)return a; return
    gcd(b, a % b);}
template <class T> inline T modinverse(T a, T M) {return bigmod(a, M - 2,
    M);}
template <class T> inline T lcm(T a, T b) {a = abs(a); b = abs(b); return
    (a / gcd(a, b)) * b;}
template <class T, class X> inline bool getbit(T a, X i) { T t = 1;
    return ((a & (t << i)) > 0);}
template <class T, class X> inline T setbit(T a, X i) { T t = 1; return
    (a | (t << i)); }
template <class T, class X> inline T resetbit(T a, X i) { T t = 1; return
    (a & (~(t << i)));}

inline ll getnum()
{
    char c = getchar();
    ll num, sign = 1;
    for (; c < '0' || c > '9'; c = getchar())if (c == '-')sign = -1;
    for (num = 0; c >= '0' && c <= '9';)
    {
        c -= '0';
        num = num * 10 + c;
        c = getchar();
    }
    return num * sign;
}

inline ll power(ll a, ll b)
{
    ll multiply = 1;
    FOR(i, 0, b)
    {
        multiply *= a;
    }
    return multiply;
}
```

7.18 int128

```
__int128 input(){
    string s;
    cin >> s;
    ll fst = (s[0] == '-') ? 1 : 0;
    __int128 v = 0;
    f(i,fst,s.size()) v = v * 10 + s[i] - '0';
    if(fst) v = -v;
    return v;
}

ostream& operator << (ostream& os,const __int128& v) {
    string ret, sgn;
    __int128 n = v;
    if(v < 0) sgn = "-", n = -v;
    while(n) ret.pb(n % 10 + '0'), n /= 10;
    reverse(all(ret));
    ret = sgn + ret;
    os << ret;
    return os;
}

int main(){
    __int128 n = input();
    cout << n << endl;
}
```

8 Notes

9 String

9.1 A KMP Application

```
/* You are given a text t and a pattern p. For each index of t, find
how many proper prefixes of p ends in this position. Similarly, find how
    many proper
suffixes start from this position.
While calculating the failure function, we can find for each position of
    the pattern p
how many of its own prefixes end in that position. After calculating that
    in dp[i],
we can just feel table[i] for text t.
*/
```

```
int pi[N], dp[N];
void prefixFun(string &p)
{
	int now;
	pi[0]=now=-1;

	dp[0]=1; // 0th character is a prefix ending in itself, base case

	for(int i=1; i<p.size(); i++)
	{
		while(now!=-1 && p[now+1]!=p[i])
			now=pi[now];

		if(p[now+1]==p[i]) pi[i]=++now;
		else pi[i]=now=-1;

		if(pi[i]!=-1) // calculate the # of prefixes end in this
		    position of p
			dp[i]=dp[pi[i]]+1;
		else dp[i]=1;
	}
}

int kmpMatch(string &p, string &t, int *table)
{
	int now=-1;
	FOR(i,0,t.size())
	{
		while(now!=-1 && p[now+1]!=t[i])
			now=pi[now];
		if(p[now+1]==t[i])
		{
			++now;
			table[i]=dp[now]; // table for text t
		}
		else now=-1;
		if(now+1==p.size())
		{
			now=pi[now];
		}
	}
}
```

9.2 Aho Corasick 2

```
int n;
string s, p[MAX];
map<char, int> node[MAX];
int root, nnode, link[MAX];
vi ending[MAX], exist[MAX];
// exist[i] has all the ending occurrences of the input strings
void insertword(int idx)
{
	int len = p[idx].size();
	int now = root;
	FOR(i, 0, len)
	{
		if (!node[now][p[idx][i]])
		{
			node[now][p[idx][i]] = ++nnode;
			node[nnode].clear();
		}
		now = node[now][p[idx][i]];
	}
// which strings end in node number 'now'?
	ending[now].pb(idx);
}
void populate(int curr)
{
// Because 'suffix-links'. It links a node with the longest proper suffix
	for (auto it : ending[link[curr]])
		ending[curr].pb(it);
}
void populate(vi &curr, int idx)
{
// So word number it ends in idx-th character of the text
	for (auto it : curr)
	{
		exist[it].pb(idx);
	}
}
void push_links()
{
	queue<int>q;
	link[0] = -1;
	q.push(0);
	while (!q.empty())
	{
```

```
            int u = q.front();
            q.pop();
            itrALL(node[u], it)
            {
                  char ch = it->first;
                  int v = it->second;
                  int j = link[u];
            // use map.find()
                  while (j != -1 && !node[j][ch])
                        j = link[j];
                  if (j != -1)link[v] = node[j][ch];
                  else link[v] = 0;
                  q.push(v);
                  populate(v);
            }
      }
}

void traverse()
{
      int len = s.size();
      int now = root;
      FOR(i, 0, len)
      {
      // use map.find()
            while (now != -1 && !node[now][s[i]])
                  now = link[now];
            if (now != -1) now = node[now][s[i]];
            else now = 0;
            populate(ending[now], i);
      }
}
```

9.3 Aho Corasick Occurrence Relation

```
// Suppose we have n<=1000 strings. Total summation of the length of
    these strings
// can be 1e7. Now we are given queries. In each query, we are given
    indices of
// two strings and asked if one of them occurs in another as a substring.
// We need to find this relation efficiently. We will use Aho-Corasick.
// The solution is to build a graph where vertices denote indices of
    strings and an edge
// from u to v denotes that string[u] occurs in string[v].

#define ALPHABET_SIZE 26
#define MAX_NODE 1e6
int n; // number of strings
string in[N], p;

int node[MAX_NODE][ALPHABET_SIZE];
int root, nnode, link[MAX_NODE], termlink[MAX_NODE], terminal[MAX_NODE];
bool graph[N][N];

// termlink[u] = a link from node u to a node which is a terminal node
// terminal node is a node where an ending of an input string occurs
// terminal[node] = the index of the string which ends in node

/* Solution:
// For every node of the Aho-Corasick structure find and remember the
nearest terminal node (termlink[u]) in the suffix-link path; Once again
    traverse
all strings through Aho-Corasick. Every time new symbol is added, add an
    arc from the node
corresponding to the current string (in the graph we build, not
    Aho-Corasick) to
the node of the graph corresponding to the nearest terminal in the
    suffix-link path;
The previous step will build all essential arcs plus
some other arcs, but they do not affect the next step in any way;
Find the transitive closure of the graph.
*/

void init()
{
    root=0;
    nnode=0;
    ms(terminal,-1);
    ms(termlink,-1);
}

void insertword(int idx)
{
       p=in[idx];
    int len=p.size();
    int now=root;
```

```
    FOR(i,0,len)
    {
        int x=p[i]-'a';

        if(!node[now][x])
        {
            node[now][x]=++nnode;
        }

        now=node[now][x];
    }

    terminal[now]=idx; // string with index idx ends in now
}

void push_links()
{
    queue<int>q;
    link[0]=-1;
    q.push(0);

    while(!q.empty())
    {
        int u=q.front();
        q.pop();

        for(int i=0; i<ALPHABET_SIZE; i++)
        {
               if(!node[u][i]) continue;

            int v=node[u][i];
            int j=link[u];
            // use map.find()
            while(j!=-1 && !node[j][i])
                j=link[j];

            if(j!=-1) link[v]=node[j][i];
            else link[v]=0;

                      // Finding nearest terminal nodes
            if(terminal[link[v]]!=-1)
                termlink[v]=link[v];
            else termlink[v]=termlink[link[v]];

            q.push(v);
        }
    }
}

void buildgraph()
{
       FOR(i,0,n)
       {
              int curr=root;

              FOR(j,0,in[i].size())
              {
                     char ch=in[i][j];
                     curr=node[curr][(int)ch-'a'];

                     int st=curr;
                     if(terminal[st]==-1) st=termlink[st];

                     for(int k=st; k>=0; k=termlink[k])
                     {
                            if(terminal[k]==i) continue;
                            if(graph[i][terminal[k]]) break;
                            graph[i][terminal[k]]=true;

                            // cout<<"edge: "<<i<<" "<<terminal[k]<<endl;
                     }
              }
       }
}

// Finally, find transitive closure of the graph. If O(n^3) is possible,
    we can use
// Floyd-Warshall. Otherwise, run dfs from each node and add an edge from
    current starting
// node to each reachable node. An edge in this transitive closure
    denotes the occurrence relation.
```

9.4 Aho Corasick

```
int n; // n is the number of dictionary word
string s,p; // dictionary words are inputted in p, s is the traversed text
```

```
#define MAX_NODE 250004

map<char,int> node[MAX_NODE]; // use 2d array maybe?
int root, nnode, link[MAX_NODE], endof[504], travis[MAX_NODE];
pii level[MAX_NODE];

void init()
{
    root=0;
    nnode=0;
    travis[root]=0; // number of time a node is traversed by s
    level[root]=MP(0,root); // level, node
    node[root].clear();
}

void insertword(int idx)
{
    int len=p.size();
    int now=root;

    FOR(i,0,len)
    {
        // use map.find()
        if(!node[now][p[i]])
        {
            node[now][p[i]]=++nnode;
            node[nnode].clear();

            travis[nnode]=0;
            level[nnode]=MP(level[now].first+1,nnode);
        }

        now=node[now][p[i]];
    }

    endof[idx]=now; // end of dictionary word idx
}

void push_links()
{
    queue<int>q;
    link[0]=-1;
    q.push(0);

    while(!q.empty())
    {
        int u=q.front();
        q.pop();

        itrALL(node[u],it)
        {
            char ch=it->first;
            int v=it->second;
            int j=link[u];

            // use map.find()
            while(j!=-1 && !node[j][ch])
                j=link[j];

            if(j!=-1) link[v]=node[j][ch];
            else link[v]=0;

            q.push(v);
        }
    }
}

void traverse()
{
    int len=s.size();
    int now=root;

    travis[root]++;

    FOR(i,0,len)
    {
        // use map.find()
        while(now!=-1 && !node[now][s[i]])
            now=link[now];

        if(now!=-1) now=node[now][s[i]];
        else now=0;

        travis[now]++;
    }

    sort(level,level+nnode+1,greater<pii>());

    FOR(i,0,nnode+1)
    {
```

```
        now=level[i].second;
        travis[link[now]]+=travis[now];
    }
}

void driver()
{
    init();
    FOR(i,0,n)
    {
        // input p
        insertword(i);
    }
    // input s
    push_links();
    traverse();
    // number of occurence of word i in s is travis[endof[i]]
}
```

9.5 Double Hash

```
const int p1 = 7919;
const int mod1 = 1000004249;
const int p2 = 2203;
const int mod2 = 1000000289;

ll pwr1[MAX+7], pwr2[MAX+7];

void precalc()
{
    ll pw1 = 1, pw2 = 1;

    FOR(i,0,MAX)
    {
        pwr1[i] = pw1;
        pwr2[i] = pw2;

        pw1 = (pw1 * p1) % mod1;
        pw2 = (pw2 * p2) % mod2;
    }

    pwr1[MAX] = pw1;
    pwr2[MAX] = pw2;
}

struct Hash {
    ll h1[MAX], h2[MAX];
    int n; // length of s

    Hash(char *s, int n): n(n) {
        ll th1 = 0, th2 = 0;
        FOR(i, 0, n) {
            th1 = (th1 + s[i] * pwr1[i]) % mod1;
            th2 = (th2 + s[i] * pwr2[i]) % mod2;
            h1[i] = th1;
            h2[i] = th2;
        }
    }
    Hash() {}
    pair<ll, ll> getHash(ll i, ll j) {

        if(i>j) return {0,0};

        ll ret1, ret2;
        if (!i) {
            ret1 = h1[j];
            ret2 = h2[j];
        }
        else {
            // Note: may need to do modinverse
            // in that case, precalc inv1[] and inv2[]
            ret1 = (h1[j] - h1[i - 1]) % mod1;
            if (ret1 < 0) ret1 += mod1;
            ret2 = (h2[j] - h2[i - 1]) % mod2;
            if (ret2 < 0) ret2 += mod2;
        }
        return MP(ret1, ret2);
    }
};
```

9.6 Dynamic Aho Corasick Sample

```
/* Problem: We have three types of queries: add a string to our
    dictionary,
delete an existing string from our dictionary, and for the given string s
    find
```

```
the number of occurrences of the strings from the dictionary. If some
    string p
from dictionary has several occurrences in s, we should count all of them.

Solution: If we have N strings in the dictionary, maintain log(N) Aho
    Corasick
automata. The i-th automata contains the first 2^k strings not included
    in the
previous automata. For example, if we have N = 19, we need 3 automata:
    {s[1]...s[16]},
{s[17]...s[18]}, and {s[19]}. To answer the query, we can traverse the
    logN automata
using the given query string.

To handle addition, first construct an automata using the single string,
    and then
while there are two automatons with the same number of strings, we merge
    them by
constructing a new automaton using brute force.
Complexity becomes
    O(total_length_of_all_string*log(number_of_insert_operations)).

To handle deletion, we just insert with a value -1 to store in endpoints
    of each
added string.
*/
const int N = 3e5+7; // maximum number of nodes
const int ALPHA = 26; // alphabate size
int node[20][N][ALPHA]; // stores nodes for id-th automaton
struct ahoCorasick
{
    int root, nnode;
    int link[N], cnt[N], id;
    bool dead;

    void init(int idx)
    {
        dead = false; id = idx; root = 0; nnode = 0;
        ms(node[id][root],0);
        FOR(i,0,nnode+1) cnt[i] = 0;
    }
    void clear()
    {
        dead = true;
    }
```

```
// insert a word in automaton number 'id'.
void insertword(string &s, int val)
{
    int len = s.size();
    int now = root;

    FOR(i,0,len)
    {
        int nxt = s[i]-'a';

        if(!node[id][now][nxt])
        {
            node[id][now][nxt] = ++nnode;
            ms(node[id][nnode],0);
            cnt[nnode]=0;
        }
        now=node[id][now][nxt];
    }
    cnt[now]+=val;
    // an occurrence of a string happened in 'now'
    // if val=-1, it means an occurrence is removed
}
void insertdict(vector<string> &dict, vector<int> &vals)
{
    // dict is the dictionary for current automaton
    // and vals can be 1 or -1 denoting addition or deletion of
        corresponding string
    FOR(i,0,dict.size()) insertword(dict[i],vals[i]);
}
void pushLinks()
{
    queue<int> Q; link[root]=-1;
    Q.push(root);
    while(!Q.empty())
    {
        int u = Q.front(); Q.pop();

        for(int i=0; i<ALPHA; i++)
        {
            if(!node[id][u][i]) continue;
            int v = node[id][u][i];
            int l = link[u];

            while(l!=-1 && !node[id][l][i]) l = link[l];
            if(l!=-1) link[v] = node[id][l][i];
```

```
                else link[v] = 0;
                cnt[v]+=cnt[link[v]];
                Q.push(v);
            }
        }
    }
    // Returns how many occurrences of dictionary are there in query
        string p
    int query(string &p)
    {
        int u = root, ret = 0;
        for(char ch: p)
        {
            int nxt = ch-'a';
            while(u!=-1 && !node[id][u][nxt]) u = link[u];
            if(u==-1) u = 0;
            else u = node[id][u][nxt];
            ret+=cnt[u];
        }
        return ret;
    }
} aho[20];

vector<string> dict[20];
vector<int> vals[20];
// handles addition and deletion dynamically
void add(string &s, int val)
{
        dict[0].pb(s);
        vals[0].pb(val);
        for(int i=0; i<20; i++)
        {
                if(dict[i].size()>(1<<i)) // merging two automata
                {
                        for(auto it: dict[i]) dict[i+1].pb(it);
                        for(auto it: vals[i]) vals[i+1].pb(it);
                        dict[i].clear();
                        vals[i].clear();
                        aho[i].clear(); // i-th automata is not relevant
                            anymore
                }
                else
                {
                        aho[i].init(i);
                        aho[i].insertdict(dict[i],vals[i]);
                        aho[i].pushLinks();
                        break;
                }
        }
}

int n, t;
char in[N];
void Test()
{
        ahoCorasick aho;
        aho.init(0);
        string s = "o"; aho.insertword(s,1);
        s = "m"; aho.insertword(s,1);
        s = "h"; aho.insertword(s,1);
        aho.pushLinks();
        s = "moohh";
        prnt(aho.query(s));
}
int main()
{
    // Test();
    FOR(i,0,20) aho[i].clear();
    scanf("%d", &n);
    while(n--)
    {
        scanf("%d%s", &t, in);
                string x = in;
        if(t==1) add(x,1);
        else if(t==2) add(x,-1)
        else
        {
                ll ans = 0;
                FOR(i,0,20) if(!aho[i].dead) ans+=aho[i].query(x);
                printf("%lld\n", ans);
                fflush(stdout); // needed because the problem forced
                    online solution
        }
    }
    return 0;
}
```

9.7 Dynamic Aho Corasick

```cpp
#include <bits/stdtr1c++.h>

#define LOG 19
#define LET 26
#define MAX 300010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

struct aho_corasick{
    int id, edge[256];
    vector <long long> counter;
    vector <string> dictionary;
    vector <map<char, int> > trie;
    vector <int> leaf, fail, dp[LET];

    inline int node(){
        leaf.push_back(0);
        counter.push_back(0);
               trie.push_back(map<char, int>());
               return id++;
    }

    inline int size(){
        return dictionary.size();
    }

    void clear(){
        trie.clear();
        dictionary.clear();
        fail.clear(), leaf.clear(), counter.clear();
        for (int i = 0; i < LET; i++) dp[i].clear();

        id = 0, node();
        for (int i = 'a'; i <= 'z'; i++) edge[i] = i - 'a'; /// change
            here if different character set
    }

    aho_corasick(){
        clear();
    }

    inline void insert(const char* str){
        int j, x, cur = 0;
        for (j = 0; str[j] != 0; j++){
            x = edge[str[j]];
            if (!trie[cur].count(x)){
                int next_node = node();
                trie[cur][x] = next_node;
            }
            cur = trie[cur][x];
        }

        leaf[cur]++;
        dictionary.push_back(str);
    }

    inline void build(){ /// remember to call build
        vector <pair<int, pair<int, int> > > Q;
        fail.resize(id, 0);
        Q.push_back({0, {0, 0}});

        for (int i = 0; i < LET; i++) dp[i].resize(id, 0);
        for (int i = 0; i < id; i++){
            for (int j = 0; j < LET; j++){
                dp[j][i] = i;
            }
        }

        for(int i = 0; i < Q.size(); i++){
            int u = Q[i].first;
            int p = Q[i].second.first;
            char c = Q[i].second.second;
            for(auto& it: trie[u]) Q.push_back({it.second, {u, it.first}});

            if (u){
                int f = fail[p];
                while (f && !trie[f].count(c)) f = fail[f];
                if(!trie[f].count(c) || trie[f][c] == u) fail[u] = 0;
                else fail[u] = trie[f][c];
                counter[u] = leaf[u] + counter[fail[u]];

                for (int j = 0; j < LET; j++){
                    if (u && !trie[u].count(j)) dp[j][u] = dp[j][fail[u]];
                }
            }
        }
```

```
    }

    inline int next(int cur, char ch){
        int x = edge[ch];
        cur = dp[x][cur];
        if (trie[cur].count(x)) cur = trie[cur][x];
        return cur;
    }

    long long count(const char* str){ /// total number of occurrences of
         all words from dictionary in str
        int cur = 0;
        long long res = 0;

        for (int j = 0; str[j] && id > 1; j++){ /// id > 1 because build
             will not be called if empty dictionary in dynamic aho corasick
            cur = next(cur, str[j]);
            res += counter[cur];
        }
        return res;
    }
};

struct dynamic_aho{ /// dynamic aho corasick in N log N
    aho_corasick ar[LOG];

    dynamic_aho(){
        for (int i = 0; i < LOG; i++) ar[i].clear();
    }

    inline void insert(const char* str){
        int i, k = 0;
        for (k = 0; k < LOG && ar[k].size(); k++){}

        ar[k].insert(str);
        for (i = 0; i < k; i++){
            for (auto s: ar[i].dictionary){
                ar[k].insert(s.c_str());
            }
            ar[i].clear();
        }
        ar[k].build();
    }

    long long count(const char* str){
        long long res = 0;
        for (int i = 0; i < LOG; i++) res += ar[i].count(str);
        return res;
    }
};

char str[MAX];

int main(){
    dynamic_aho ar[2];
    int t, i, j, k, l, flag;

    scanf("%d", &t);
    while (t--){
        scanf("%d %s", &flag, str);
        if (flag == 3){
            printf("%lld\n", ar[0].count(str) - ar[1].count(str));
            fflush(stdout);
        }
        else ar[flag - 1].insert(str);
    }
    return 0;
}
```

9.8 KMP 2

```
char text[MAX], patt[MAX];
int pi[MAX], n, m;

void Process()
{
        int now=-1;
        pi[0]=-1;

        for(int i=1; i<m; i++)
        {
                while(now!=-1 && patt[now+1]!=patt[i])
                        now=pi[now];
                if(patt[now+1]==patt[i]) pi[i]=++now;
                else pi[i]=now=-1;
        }
}
```

```
void Search()
{
	int now=-1;

	for(int i=0; i<n; i++)
	{
		while(now!=-1 && patt[now+1]!=text[i])
			now=pi[now];
		if(patt[now+1]==text[i]) ++now;
		else now=-1;
		if(now==m-1)
		{
			cout<<"match at "<<i-now<<endl;
			now=pi[now]; // match again
		}
	}
}

int main()
{
	// ios_base::sync_with_stdio(0);
	// cin.tie(NULL); cout.tie(NULL);
	// freopen("in.txt","r",stdin);

	cin>>text>>patt;

	n=strlen(text); m=strlen(patt);

	Process();
	Search();

	// FOR(i,0,m) cout<<pi[i]<<" "; cout<<endl;

	return 0;
}
```

9.9 KMP 3

```
string p, t;
int pi[MAX], cnt[MAX];

void prefixFun()
{
	int now;
	pi[0]=now=-1;

	for(int i=1; i<p.size(); i++)
	{
		while(now!=-1 && p[now+1]!=p[i])
			now=pi[now];

		if(p[now+1]==p[i]) pi[i]=++now;
		else pi[i]=now=-1;
	}
}

int kmpMatch()
{
	int now=-1;
	FOR(i,0,t.size())
	{
		cout<<"now: "<<i<<" "<<now<<endl;
		while(now!=-1 && p[now+1]!=t[i])
			now=pi[now];
		if(p[now+1]==t[i])
		{
			++now;
			cnt[now]++;
		}
		else now=-1;
		if(now+1==p.size())
		{
			// match found
			// cout<<"match and setting "<<now<<" to
			    "<<pi[now]<<endl;
			now=pi[now]; // match again
		}
	}
}

int main()
{
	// ios_base::sync_with_stdio(0);
	// cin.tie(NULL); cout.tie(NULL);
	// freopen("in.txt","r",stdin);

	cin>>t>>p;
```

```
        prefixFun();
        FOR(i,0,p.size()) cout<<pi[i]<<" "; cout<<endl;
        prnt(kmpMatch());
        FOR(i,0,p.size()) cout<<cnt[i]<<" "; cout<<endl;
        FORr(i,p.size()-1,0)
        {
                if(pi[i]==-1) continue;
                cnt[pi[i]]+=cnt[i];
        }
        FOR(i,0,p.size()) cout<<cnt[i]<<" "; cout<<endl;

        return 0;
}
```

9.10 Manacher-s Algorithm

```
#include <bits/stdtr1c++.h>
using namespace std;
/*** Manacher's algorithm to generate longest palindromic substrings for
     all centers ***/
/// When i is even, pal[i] = largest palindromic substring centered from
     str[i / 2]
/// When i is odd, pal[i] = largest palindromic substring centered
     between str[i / 2] and str[i / 2] + 1

vector <int> manacher(char *str){ /// hash = 784265
    int i, j, k, l = strlen(str), n = l << 1;
    vector <int> pal(n);

    for (i = 0, j = 0, k = 0; i < n; j = max(0, j - k), i += k){
        while (j <= i && (i + j + 1) < n && str[(i - j) >> 1] == str[(i +
             j + 1) >> 1]) j++;
        for (k = 1, pal[i] = j; k <= i && k <= pal[i] && (pal[i] - k) !=
             pal[i - k]; k++){
            pal[i + k] = min(pal[i - k], pal[i] - k);
        }
    }

    pal.pop_back();
    return pal;
}

int main(){
    char str[100];
    while (scanf("%s", str)){
        auto v = manacher(str);
        for (auto it: v) printf("%d ", it);
        puts("");
    }
    return 0;
}
```

9.11 Minimum Lexicographic Rotation

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

/// Lexicographically Minimum String Rotation
int minlex(char* str){ /// Returns the 0-based index
    int i, j, k, n, len, x, y;
    len = n = strlen(str), n <<= 1, i = 0, j = 1, k = 0;

    while((i + k) < n && (j + k) < n) {
        x = i + k >= len ? str[i + k - len] : str[i + k];
        y = j + k >= len ? str[j + k - len] : str[j + k];
        if(x == y) k++;
        else if (x < y){
            j += ++k, k = 0;
            if (i >= j) j = i + 1;
        }
        else{
            i += ++k, k = 0;
            if (j >= i) i = j + 1;
        }
    }

    return (i < j) ? i : j;
}

int t;
char str[50010];
```

```
int main(){
    gets(str);
    while (gets(str)){
        printf("%d\n", minlex(str));
    }
    return 0;
}
```

9.12 Palindrome Factorization

```
#include <bits/stdtr1c++.h>

#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Minimum palindromic factorization for all prefixes in O(N log N)

namespace pal{
    int pl[MAX][2], gpl[MAX][2];

    inline void set(int *ar, int x, int y, int z) {
        ar[0] = x, ar[1] = y, ar[2] = z;
    }

    inline void set(int ar[][2], int i, int v) {
        if (v > 0) ar[i][v & 1] = v;
    }

    inline void copy(int *A, int *B) {
        A[0] = B[0], A[1] = B[1], A[2] = B[2];
    }

    inline void update(int ar[][2], int i, int v) {
        if (v > 0 && (ar[i][v & 1] == -1 || ar[i][v & 1] > v)) ar[i][v &
            1] = v;
    }

    /// Returns a vector v such that,
    /// v[i] is the minimum k so that the prefix string str[0:i] can be
        partitioned into k disjoint palindromes
    inline vector <int> factorize(const char* str){
        int g[32][3], gp[32][3], gpp[32][3];
        int i, j, k, l, d, u, r, x, pg = 0, pgp = 0, pgpp = 0, n =
            strlen(str);

        clr(g), clr(gp), clr(gpp);
        for (int i = 0; i < n; i++) gpl[i][0] = MAX, gpl[i][1] = MAX + 1;

        for (j = 0; j < n; j++){
            for (u = 0, pgp = 0; u < pg; u++){
                i = g[u][0];
                if ((i - 1) >= 0 && str[i - 1] == str[j]){
                    g[u][0]--;
                    copy(gp[pgp++], g[u]);
                }
            }

            pgpp = 0, r = -(j + 2);
            for (u = 0; u < pgp; u++){
                i = gp[u][0], d = gp[u][1], k = gp[u][2];
                if ((i - r) != d){
                    set(gpp[pgpp++], i, i - r, 1);
                    if (k > 1) set(gpp[pgpp++], i + d, d, k - 1);
                }
                else set(gpp[pgpp++], i, d, k);
                r = i + (k - 1) * d;
            }

            if (j - 1 >= 0 && str[j - 1] == str[j]){
                set(gpp[pgpp++], j - 1, j - 1 - r, 1);
                r = j - 1;
            }
            set(gpp[pgpp++], j, j - r, 1);

            int *ptr = gpp[0];
            for (u = 1, pg = 0; u < pgpp; u++){
                int *x = gpp[u];
                if (x[1] == ptr[1]) ptr[2] += x[2];
                else {
                    copy(g[pg++], ptr);
                    ptr = x;
                }
            }
```

```
            copy(g[pg++], ptr);

            pl[j + 1][(j & 1) ^ 1] = j + 1;
            pl[j + 1][j & 1] = MAX + (j & 1);
            for (u = 0; u < pg; u++){
                i = g[u][0], d = g[u][1], k = g[u][2];
                r = i + (k - 1) * d;

                update(pl, j + 1, pl[r][0] + 1);
                update(pl, j + 1, pl[r][1] + 1);
                if (k > 1) {
                    update(pl, j + 1, gpl[i + 1 - d][0]);
                    update(pl, j + 1, gpl[i + 1 - d][1]);
                }

                if (i + 1 >= d) {
                    if (k > 1) {
                        update(gpl, i + 1 - d, pl[r][0] + 1);
                        update(gpl, i + 1 - d, pl[r][1] + 1);
                    }
                    else{
                        set(gpl, i + 1 - d, pl[r][0] + 1);
                        set(gpl, i + 1 - d, pl[r][1] + 1);
                    }
                }
            }
        }

        vector <int> res(n, 0);
        for (i = 0; i < n; i++) res[i] = min(pl[i + 1][0], pl[i + 1][1]);
        return res;
    }
}

int main(){

}
```

9.13 Palindromic Tree

```
class PalindromicTree
{
public:
int s[MAX], Link[MAX], Len[MAX], Edge[MAX][26];
int node, lastPal, n;
ll cnt[MAX];

void init()
{
        s[n++]=-1;
        Link[0]=1; Len[0]=0;
        Link[1]=1; Len[1]=-1;
        node=2;
}

int getLink(int v)
{
        while(s[n-Len[v]-2]!=s[n-1]) v=Link[v];
        return v;
}

void addLetter(int c)
{
        // cout<<char(c+'a')<<" "<<n<<endl;

        s[n++]=c;
        lastPal=getLink(lastPal);

        if(!Edge[lastPal][c])
        {
                Len[node]=Len[lastPal]+2;
                Link[node]=Edge[getLink(Link[lastPal])][c];
                cnt[node]++;
                Edge[lastPal][c]=node++;
        }
        else
        {
                cnt[Edge[lastPal][c]]++;
        }

        lastPal=Edge[lastPal][c];
}

void clear()
{
        FOR(i,0,node+1)
        {
                cnt[i]=0;
```

```
                ms(Edge[i],0);
            }
            n=0;
            lastPal=0;
        }
} PTA;
```

9.14 String Split by Delimiter

```
template<typename Out>
void split(const std::string &s, char delim, Out result) {
    std::stringstream ss(s);
    std::string item;
    while (std::getline(ss, item, delim)) {
        *(result++) = item;
    }
}

std::vector<std::string> split(const std::string &s, char delim) {
    std::vector<std::string> elems;
    split(s, delim, std::back_inserter(elems));
    return elems;
}

// Continuous input
string line;
while( getline(cin,line) )
{
        stringstream ss( line ); // initialize kortesi
        int num; vector< int > v;
        while( ss >> num ) v.push_back( num ); // :P
        sort( v.begin(), v.end() );
        // print routine
}
```

9.15 Suffix Array 2

```
// You are given two strings A and B, consisting only of lowercase
    letters from the English alphabet.
// Count the number of distinct strings S, which are substrings of A, but
    not substrings of B
```

```
LL substr_count(int n,char *s)
{
        VI cnt(128);
        for(int i=0;i<n;i++)
                cnt[s[i]]++;
        for(int i=1;i<128;i++)
                cnt[i]+=cnt[i-1];
        VI p(n);
        for(int i=0;i<n;i++)
                p[--cnt[s[i]]]=i;
        VVI c(1,VI(n));
        int w=0;
        for(int i=0;i<n;i++)
        {
                if(i==0 || s[p[i]]!=s[p[i-1]]) w++;
                c[0][p[i]] = w-1;
        }

        for(int k=0,h=1;h<n;k++,h*=2)
        {
                VI pn(n);
                for(int i=0;i<n;i++) {
                        pn[i] = p[i] - h;
                        if(pn[i]<0) pn[i] += n;
                }
                VI cnt(w,0);
                for(int i=0;i<n;i++)
                        cnt[c[k][pn[i]]]++;
                for(int i=1;i<w;i++)
                        cnt[i]+=cnt[i-1];
                for(int i=n;i--;)
                        p[--cnt[c[k][pn[i]]]]=pn[i];
                w=0;
                c.push_back(VI(n));
                for(int i=0;i<n;i++)
                {
                        if(i==0 || c[k][p[i]] != c[k][p[i-1]]) {
                                w++;
                        } else {
                                int i1 = p[i]  + h; if(i1>=n) i1-=n;
                                int i2 = p[i-1] + h; if(i2>=n) i2-=n;
                                if(c[k][i1]!=c[k][i2]) w++;
                        }
                        c[k+1][p[i]] = w-1;
```

```
                }
        }

        LL ans = LL(n)*(n-1)/2;
        for(int k=1;k<n;k++)
        {
                int i=p[k];
                int j=p[k-1];
                int cur = 0;
                for (int h=c.size(); h--;)
                        if (c[h][i] == c[h][j]) {
                                cur += 1<<h;
                                i += 1<<h;
                                j += 1<<h;
                        }
                ans-=cur;
        }
        return ans;
}

char s[200005];
int n, m;

void input()
{
        scanf("%s", s);

        n=strlen(s)+1;
        s[n-1]='a'-1;

        scanf("%s", s+n);
        m=strlen(s+n)+1;

        s[n+m-1]='a'-2;
        s[n+m]=0;
}

void solve()
{
        ll p=substr_count(n,s);
        ll r=substr_count(m,s+n);
        ll q=substr_count(n+m,s)-(ll)n*m;

        // cout<<p<<" "<<r<<" "<<q<<endl;

        ll t=q-p-r;

        t=abs(t);

        prnt(p-t);
}

int main()
{
    // ios_base::sync_with_stdio(0);
    // cin.tie(NULL); cout.tie(NULL);
    // freopen("in.txt","r",stdin);

    int test, cases=1;

    input();
    solve();

    return 0;
}
```

9.16 Suffix Array

```
// sa[i] -> ith smallest suffix of the string (indexed from 1)
// height[i] -> Longest common substring between Suffix(sa[i]) and
    Suffix(sa[i-1]), indexed
// from i=2.
// rak[i] -> The position of i th index of the main string in suffix
    array.
// rak[6]=1 means 6th suffix is in 1st position in sa

const int N = 2e6+5;
int wa[N],wb[N],wv[N],wc[N];
int r[N],sa[N],rak[N], height[N], lg[N];

int cmp(int *r,int a,int b,int l)
{
     return r[a] == r[b] && r[a+l] == r[b+l];
}

void da(int *r,int *sa,int n,int m)
{
     int i,j,p,*x=wa,*y=wb,*t;
```

```
	for( i=0;i<m;i++) wc[i]=0;
	for( i=0;i<n;i++) wc[x[i]=r[i]] ++;
	for( i=1;i<m;i++) wc[i] += wc[i-1];
	for( i= n-1;i>=0;i--)sa[--wc[x[i]]] = i;
	for( j= 1,p=1;p<n;j*=2,m=p){
		for(p=0,i=n-j;i<n;i++)y[p++] = i;
		for(i=0;i<n;i++)if(sa[i] >= j) y[p++] = sa[i] - j;
		for(i=0;i<n;i++)wv[i] = x[y[i]];
		for(i=0;i<m;i++) wc[i] = 0;
		for(i=0;i<n;i++) wc[wv[i]] ++;
		for(i=1;i<m;i++) wc[i] += wc[i-1];
		for(i=n-1;i>=0;i--) sa[--wc[wv[i]]] = y[i];
		for(t=x,x=y,y=t,p=1,x[sa[0]] = 0,i=1;i<n;i++) x[sa[i]]=
		    cmp(y,sa[i-1],sa[i],j) ? p-1:p++;

	}
}

void calheight(int *r,int *sa,int n)
{

	int i,j,k=0;
	for(i=1;i<=n;i++) rak[sa[i]] = i;
	for(i=0;i<n;height[rak[i++]] = k ) {

		for(k?k--:0, j=sa[rak[i]-1] ; r[i+k] == r[j+k] ; k ++) ;
	}

}

int dp[N][22];

void initRMQ(int n)
{
	for(int i= 1;i<=n;i++) dp[i][0] = height[i];
	for(int j= 1; (1<<j) <= n; j ++ ){
		for(int i = 1; i + (1<<j) - 1 <= n ; i ++ ) {
			dp[i][j] = min(dp[i][j-1] , dp[i + (1<<(j-1))][j-1]);
		}
	}

}

int askRMQ(int L,int R)
{
	int k = lg[R-L+1];
	// int k=0;
	// while((1<<(k+1)) <= R-L+1) k++;
	return min(dp[L][k], dp[R - (1<<k) + 1][k]);
}
// Precalculate powers of two to answer askRMQ in O(1)
int preclg2()
{
	for(int i=2; i<N; i++)
	{
		lg[i]=lg[i-1];
		if((i&(i-1))==0) lg[i]++;
	}
}

int main()
{
	string s; cin>>s;
	int n=s.size(), cnt=0;

	FOR(i,0,s.size())
	{
		r[i]=s[i]-'a'+1;
		// prnt(r[i]);
		cnt=max(cnt,r[i]);
	}

	r[n]=0; // This is very important, if there are testacases!
	da(r,sa,n+1,cnt+1); // cnt+1 is must, cnt=max of r[i]
	calheight(r,sa,n);

	for(int i=1; i<=n; i++)
		printf("sa[%d] = %d\n", i, sa[i]);

	for(int i=2; i<=n; i++)
		printf("height[%d] = %d\n", i, height[i]);

	for(int i=1; i<=n; i++)
		printf("rank[%d] = %d\n", sa[i], rak[sa[i]]);

	// Must call initRMQ(len)
	// To find lcp between any two suffix i and j, call askRMQ(L+1,R)
	// where L=min(rak[sa[i]],rak[sa[j]]), R=max(rak[sa[i]],rak[sa[j]]).
```

```
    /* A Reminder: Sometimes when we concatenate strings, we do that by
        adding
    separators. We might need to add same separator or different
        separators.
    And it might also need to add a separator at the end of the total
        strings.
    */

      return 0;
}
```

9.17 Suffix Automata 2

```
struct state {
    ll len, link;
    map<char,ll, less<char> >next; // use less for kth lexiograpical
        string evaluation
};

state st[MAX*2];
ll sz, last;

void sa_init() {
    sz = last = 0;
    st[0].len = 0;
    st[0].link = -1;
    st[0].next.clear();
    ++sz;
}

ll cnt[MAX*2];

ll distSubtringCount;

void sa_extend (char c) {
    ll cur = sz++;
    st[cur].next.clear();
    cnt[cur] = 1;
    st[cur].len = st[last].len + 1;
    ll p;
    for (p=last; p!=-1 && !st[p].next.count(c); p=st[p].link){
        st[p].next[c] = cur;
    }
    if (p == -1){
        st[cur].link = 0;
    }
    else {
        ll q = st[p].next[c];
        if (st[p].len + 1 == st[q].len){
            st[cur].link = q;
        }
        else {
            ll clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            for (; p!=-1 && st[p].next[c]==q; p=st[p].link){
                st[p].next[c] = clone;
            }
            distSubtringCount-=st[q].len-st[st[q].link].len;
            st[q].link = st[cur].link = clone;
            distSubtringCount+=st[q].len-st[st[q].link].len;
            distSubtringCount+=st[clone].len-st[st[clone].link].len;
        }
    }
    last = cur;
    distSubtringCount+=st[cur].len-st[st[cur].link].len;
}

void calc_cnt(){
    vpl sorter;
    f(i,0,sz) sorter.pb(mp(st[i].len, i));
    sort(all(sorter));
    fd(i,sz-1,-1){
        ll k = sorter[i].second;
        cnt[st[k].link] += cnt[k];
    }
}


ll get_cnt(string s){
    ll now = 0;
    f(i,0,s.size()){
        if(!st[now].next.count(s[i])) return 0;
        now = st[now].next[s[i]];
    }
    return cnt[now];
}
```

```
ll first_occur(string s){
    ll now = 0;
    f(i,0,s.size()){
        if(!st[now].next.count(s[i])) return -1;
        now = st[now].next[s[i]];
    }
    return st[now].len - s.size();
}

string lcs (string s, string t) {
        sa_init();
        for (int i=0; i<(int)s.length(); ++i)
                sa_extend (s[i]);

        int v = 0, l = 0,
                best = 0, bestpos = 0;
        for (int i=0; i<(int)t.length(); ++i) {
                while (v && ! st[v].next.count(t[i])) {
                        v = st[v].link;
                        l = st[v].len;
                }
                if (st[v].next.count(t[i])) {
                        v = st[v].next[t[i]];
                        ++l;
                }
                if (l > best)
                        best = l, bestpos = i;
        }
        return t.substr (bestpos-best+1, best);
}
// Kth Lexiographicaly smallest string
ll dp[MAX];  // dp[i] = number of different substring starting from i

ll F(ll u){
    if(dp[u] != -1) return dp[u];
    dp[u] = 1;
    for(map <char, ll> :: iterator it = st[u].next.begin(); it !=
        st[u].next.end(); it++){
        dp[u] += F(it->second);
    }
    return dp[u];
}

string klex(ll u, ll k){
    if(!k) return "";
    for(map <char, ll> :: iterator it = st[u].next.begin(); it !=
        st[u].next.end(); it++){
        ll num = F(it->second);
        if(num < k) k -= num;
        else{
            string ret;
            ret.pb(it->first);
            ret = ret + klex(it->second, k-1);
            return ret;
        }
    }
}

ll min_cyclic_shift(string s){
    sa_init();
    f(i,0,s.size()) sa_extend(s[i]);
    f(i,0,s.size()) sa_extend(s[i]);
    ll now = 0;
    f(i,0,s.size()) now = st[now].next.begin()->second;
    return st[now].len - s.size();
}

char s[MAX];

int main(){
    string s;
    cin >> s;
    cout << min_cyclic_shift(s) << endl;
}
```

9.18 Suffix Automata

```
// Counts number of distinct substrings

struct suffix_automaton
{
    map<char, int> to[MAX];
    int len[MAX], link[MAX];
    int last, psz = 0;

    void add_letter(char c)
    {
```

```
int p = last, cl, q;
if(to[p].count(c))
{
    q = to[p][c];
    if(len[q] == len[p] + 1)
    {
        last = q;
        return;
    }

    cl = psz++;
    len[cl] = len[p] + 1;
    to[cl] = to[q];
    link[cl] = link[q];
    link[q] = cl;
    last = cl;

    for(; to[p][c] == q; p = link[p])
        to[p][c] = cl;

    return;
}

last = psz++;
len[last] = len[p] + 1;

for(; to[p][c] == 0; p = link[p])
    to[p][c] = last;

if(to[p][c] == last)
{
    link[last] = p;
    return;
}

q = to[p][c];
if(len[q] == len[p] + 1)
{
    link[last] = q;
    return;
}

cl = psz++;
len[cl] = len[p] + 1;
to[cl] = to[q];
        link[cl] = link[q];
        link[q] = cl;
        link[last] = cl;

        for(; to[p][c] == q; p = link[p])
            to[p][c] = cl;
    }

    void clear()
    {
        for(int i = 0; i < psz; i++)
            len[i] = 0, link[i] = 0, to[i].clear();
        psz = 1;
        last = 0;
    }

    void init(string s)
    {
        clear();
        for(int i = 0; i < s.size(); i++)
            add_letter(s[i]);
    }

    suffix_automaton() {psz = 0; clear();}
};

string s;
suffix_automaton SA;
ll cnt[MAX];
vi endpos[MAX];

int main()
{
    // ios_base::sync_with_stdio(0);
    // cin.tie(NULL); cout.tie(NULL);
    // freopen("in.txt","r",stdin);

    int test, cases=1;

    cin>>s;

    SA.clear();
    FOR(i,0,s.size())
        SA.add_letter(s[i]), cnt[SA.last]++;
```

```
    FOR(i,0,SA.psz)
    {
        endpos[SA.len[i]].pb(i);
    }

    ll ans=0;

    FORr(i,SA.psz-1,1)
    {
        for(auto it: endpos[i])
        {
                cnt[SA.link[it]]+=cnt[it];
                ans+=(SA.len[it]-SA.len[SA.link[it]]); // distinct
                    occurrences
            // cnt[it] has occurrence of substring ending at node it
        }
    }

    // cnt[x] has occurrences of state x
    // To calculate occurrence of an input string, we visit the automata
        using the letters
    // of the input string and find the last_state where it finishes
    // The cnt[last_state] should be the occurrence of this string

    prnt(ans);

    return 0;
}
```

9.19 Trie 1

```
struct Node
{
        int cntL, cntR, lIdx, rIdx;
        Node()
        {
                cntL = cntR = 0;
                lIdx = rIdx = -1;
        }
};
Node Tree[MAX];
int globalIdx = 0;
class Trie
{
public:
        void insert(int val, int idx, int depth)
        {
                for (int i = depth - 1; i >= 0; i--)
                {
                        bool bit = val & (1 << i);
                        // cout<<"bit now: "<<bit<<endl;
                        if (bit)
                        {
                                Tree[idx].cntR++;
                                if (Tree[idx].rIdx == -1)
                                {
                                        Tree[idx].rIdx = ++globalIdx;
                                        idx = globalIdx;
                                }
                                else idx = Tree[idx].rIdx;
                        }
                        else
                        {
                                Tree[idx].cntL++;
                                if (Tree[idx].lIdx == -1)
                                {
                                        Tree[idx].lIdx = ++globalIdx;
                                        idx = globalIdx;
                                }
                                else idx = Tree[idx].lIdx;
                        }
                }
        }
        int query(int val, int compVal, int idx, int depth)
        {
                int ans = 0;
                for (int i = depth - 1; i >= 0; i--)
                {
                        bool valBit = val & (1 << i);
                        bool compBit = compVal & (1 << i);
                        if (compBit)
                        {
                                if (valBit)
                                {
                                        ans += Tree[idx].cntR;
                                        idx = Tree[idx].lIdx;
                                }
                                else
```

```
                        {
                                ans += Tree[idx].cntL;
                                idx = Tree[idx].rIdx;
                        }
                }
                else
                {
                        if (valBit)
                        {
                                idx = Tree[idx].rIdx;
                        }
                        else
                        {
                                idx = Tree[idx].lIdx;
                        }
                }
                if (idx == -1) break;
        }
        return ans;
    }
    void clear()
    {
        for (int i = 0; i <= globalIdx; i++)
        {
            Tree[i].cntL = 0;
            Tree[i].cntR = 0;
            Tree[i].rIdx = -1;
            Tree[i].lIdx = -1;
        }
        globalIdx = 0;
    }
};
int main()
{
    // ios_base::sync_with_stdio(0);
    // cin.tie(NULL); cout.tie(NULL);
    // freopen("in.txt","r",stdin);
    // Given an array of positive integers you have to print the
        number of
    // subarrays whose XOR is less than K.
    int test, n, k, x;
    Trie T;
    scanf("%d", &test);
    while (test--)
    {
        scanf("%d%d", &n, &k);
        T.insert(0, 0, 20);
        int pre = 0;
        ll ans = 0;
        FOR(i, 0, n)
        {
            scanf("%d", &x);
            pre ^= x;
            // prnt(pre);
            ans += T.query(pre, k, 0, 20);
            T.insert(pre, 0, 20);
        }
        printf("%lld\n", ans);
        T.clear();
    }
    return 0;
}
```

9.20 Trie 2

```
const int MaxN = 100005;
int sz;

int nxt[MaxN][55];
int en[MaxN];

bool isSmall(char ch)
{
    return ch>='a' && ch<='z';
}

int getId(char ch)
{
    if(isSmall(ch)) return ch-'a';
    else return ch-'A'+26;
}

void insert (char *s, int l)
{
    int v = 0;

    for (int i = 0; i < l; ++i) {
```

```
                int c=getId(s[i]);

                if (nxt[v][c]==-1)
                {
                        ms(nxt[sz],-1);
                        nxt[v][c]=sz++;
                        en[sz]=0;
                        // created[sz] = true;
                }

                v = nxt[v][c];
        }
        ++en[v];
}

int search (char *tmp, int l) {

        int v = 0;

        for (int i = 0; i < l; ++i) {

                int c=getId(tmp[i]);

                if (nxt[v][c]==-1)
                        return 0;

                v = nxt[v][c];
        }
        return en[v];
}

void init()
{
        sz=1;
        en[0]=0;
        ms(nxt[0],-1);
}
```

9.21 Z Algorithm

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 100010
#define min(a,b) ((a)<(b) ? (a):(b))
#define max(a,b) ((a)>(b) ? (a):(b))
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

char str[MAX];
int n, Z[MAX];

void ZFunction(){ /// Z[i] = lcp of the suffix starting from i with str
    int i, j, k, l, r, p;
    Z[0] = n, l = 0, r = 0;
    for (i = 1; i < n; i++){
        if (i > r){
            k = 0;
            while ((i + k) < n && str[i + k] == str[k]) k++;
            Z[i] = k;
            if (Z[i]) l = i, r = i + Z[i] - 1;
        }
        else{
            p = i - l;
            if (Z[p] < (r - i + 1)) Z[i] = Z[p];
            else{
                k = r + 1;
                while (k < n && str[k - i] == str[k]) k++;
                l = i, r = k - 1;
                Z[i] = (r - l + 1);
            }
        }
    }
}

/// Z[i] = lcp of the suffix starting from i with str
void ZFunction(char* str){ /// hash = 998923
    int i, l, r, x;

    l = 0, r = 0;
    for (i = 1; str[i]; i++){
        Z[i] = max(0, min(Z[i - l], r - i));
        while (str[i + Z[i]] && str[Z[i]] == str[i + Z[i]]) Z[i]++;
        if ((i + Z[i]) > r) l = i, r = i + Z[i];
    }
    Z[0] = i;
}
```

```
int main(){
    scanf("%s", str);
    n = strlen(str);
    ZFunction();
    return 0;
}
```