Hall's Theorem

md.mottakin.chowdhury

July 2018

1 Theorem

Suppose G is a bipartite graph with bipartition (A, B). There is a matching that covers A if and only if for every subset $X \subseteq A$, $N(X) \geq |X|$ where $N(X)$ is the number of neighbors of X.

2 Problem

You are given a string s consisting of characters from a to f, and also some conditions. Each conditions is of the form (p, t) where p is an index, t is a string, possibly of length 1. Each condition means that on position p, you have to place one character from t, which can only consist of letters a to f. All the indices are distinct.

If there is no condition for a particular position, then any letter from a to f can be placed there.

Reorganize string s such that each condition is met and the resulting string is lexicographically smallest.

3 Solution

The problem can be solved using Hall's Theorem.

We consider a bipartite graph where each partition B consists of the allowed characters and partition A is the indices (1 to n). An edge from A to B means that we can place this character. Of course if there is no edge for a particular position of A, we have to add an edge to every character of partition B.

There are 64 possible subsets for those the 6 allowed characters. For each subset X, we calculate $N(X)$. Here note that a neighbour will be counted in $N(X)$ if it is connected to any one or more of the element of the subset.

Now, we try to build the reorganized string one character at a time. We try to place the smallest allowed character on our current position. For this we check if the suffix starting from the next position can be constructed. This checking can be done using Hall's Theorem.

How? After placing the current character, we decrement its occurrence count by 1 as we are considering it placed. Now for each of the subsets of the allowed letters, we count total number of occurrence we are still left with for current subset. If the $N(X)$ for subset X in the remaining positions is less than the calculated number of occurrence, we are sure that the suffix starting from this position cannot be built.

4 Code

```
string s;
int occ[6], p, q, n, N[MAX][64];
bool graph[MAX][6];

void calcSubset()
{
        FOR(i,0,n)
        {
                FOR(j,0,64)
                {
                        bool flag=false;
                        FOR(k,0,6)
                        {
                                if(j&(1<<k))
                                {
                                        if(graph[i][k])
                                        {
                                                flag=true;
                                                break;
                                        }
                                }
                        }

                        N[i][j]+=flag;
                        if(i) N[i][j]+=N[i-1][j];
                }
        }
}

bool canBuildSuffix(int pos)
{
        FOR(i,0,64)
        {
                int val=0;
                FOR(j,0,6)
                {
                        if(i&(1<<j))
                                val+=occ[j];
                }
                if(val>N[n-1][i]-N[pos-1][i]) return false;
        }
        return true;
}

void solve()
{
        FOR(i,0,n)
        {
                int t=0;
                FOR(j,0,6) if(graph[i][j]) t++;
                if(t==0) FOR(j,0,6) graph[i][j]=true;
        }

        calcSubset();

        string ans="";
        bool canbuild=false;

        FOR(i,0,n)
        {
                FOR(j,0,6)
                {
                        if(!graph[i][j] || occ[j]==0) continue;
                        occ[j]--;
                        if(canBuildSuffix(i+1))
                        {
                                canbuild=true;
                                ans+=char(j+'a');
                                break;
                        }

                        occ[j]++;
                }
                if(!canbuild) break;
        }

        if(!canbuild) prnt("Impossible");
        else prnt(ans);
}

int main()
{
      cin>>s;
      FOR(i,0,s.size())
      {
                  occ[s[i]-'a']++;
      }
      n=s.size();
      cin>>q;
      while(q--)
      {
                  cin>>p>>s;
                  FOR(i,0,s.size())
                          graph[p-1][s[i]-'a']=true;
      }
      solve();

      return 0;
}
```