

## Practical 10

Q) Implementing Heap with different operations performed.

1. Write a program to implement max heap

→Code:

```
#include <iostream>
#include <conio.h>

using namespace std;

void max_heapify(int * a, int i, int n)
{
    int j, temp;

    temp = a[i];

    j = 2 * i;

    while (j <= n)
    {
        if (j < n && a[j + 1] > a[j])

            j = j + 1;

        if (temp > a[j])

            break;

        else if (temp <= a[j])

        {

            a[j / 2] = a[j];
            j = 2 * j;

        }

    }

    a[j / 2] = temp;

    return;
```

```

}

void build_maxheap(int * a, int n)
{
    int i;

    for (i = n / 2; i >= 1; i--)
    {
        max_heapify(a, i, n);
    }
}

int main()
{
    int n, i, x;

    cout << "Enter the number of elements of array : \n";

    cin >> n;

    int a[20];

    for (i = 1; i <= n; i++)
    {
        cout << "Enter the element " << (i) << endl;
        cin >> a[i];
    }

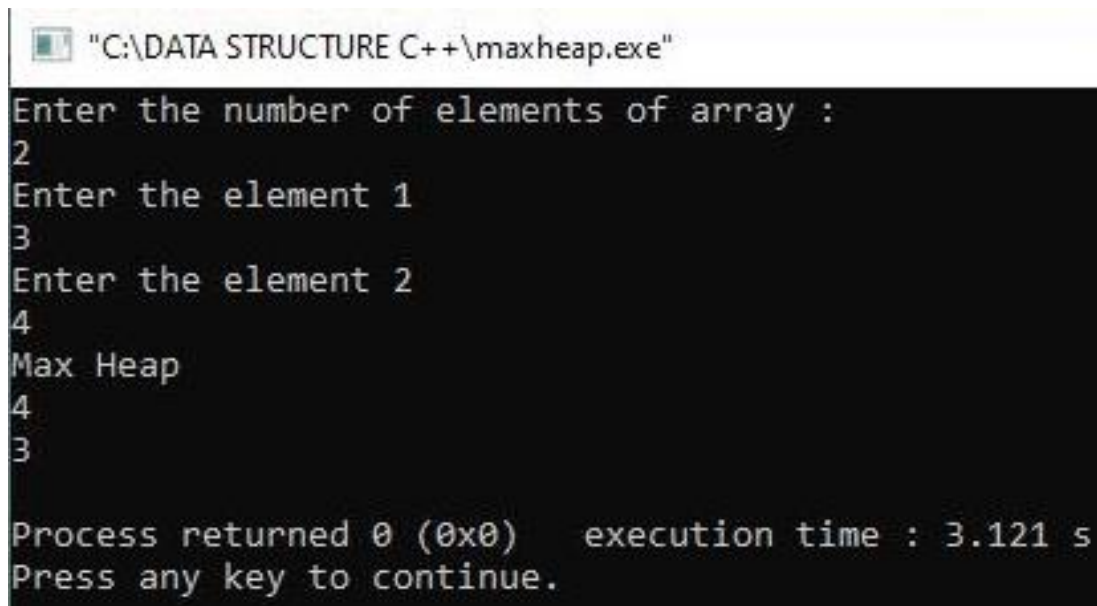
    build_maxheap(a, n);

    cout << "Max Heap\n";

    for (i = 1; i <= n; i++)
    {
        cout << a[i] << endl;
    }
}

```

Output:



```
"C:\DATA STRUCTURE C++\maxheap.exe"
Enter the number of elements of array :
2
Enter the element 1
3
Enter the element 2
4
Max Heap
4
3

Process returned 0 (0x0)   execution time : 3.121 s
Press any key to continue.
```

## 2 Write a program to implement min heap

→Code:

```
#include <iostream>
#include <conio.h>

using namespace std;

void min_heapify(int * a, int i, int n)
{
    int j, temp;

    temp = a[i];

    j = 2 * i;

    while (j <= n)
    {
        if (j < n && a[j + 1] < a[j])

            j = j + 1;

        if (temp < a[j])

            break;

        else if (temp >= a[j])

        {

            a[j / 2] = a[j];

            j = 2 * j;

        }

    }

    a[j / 2] = temp;

    return;
}
```

```

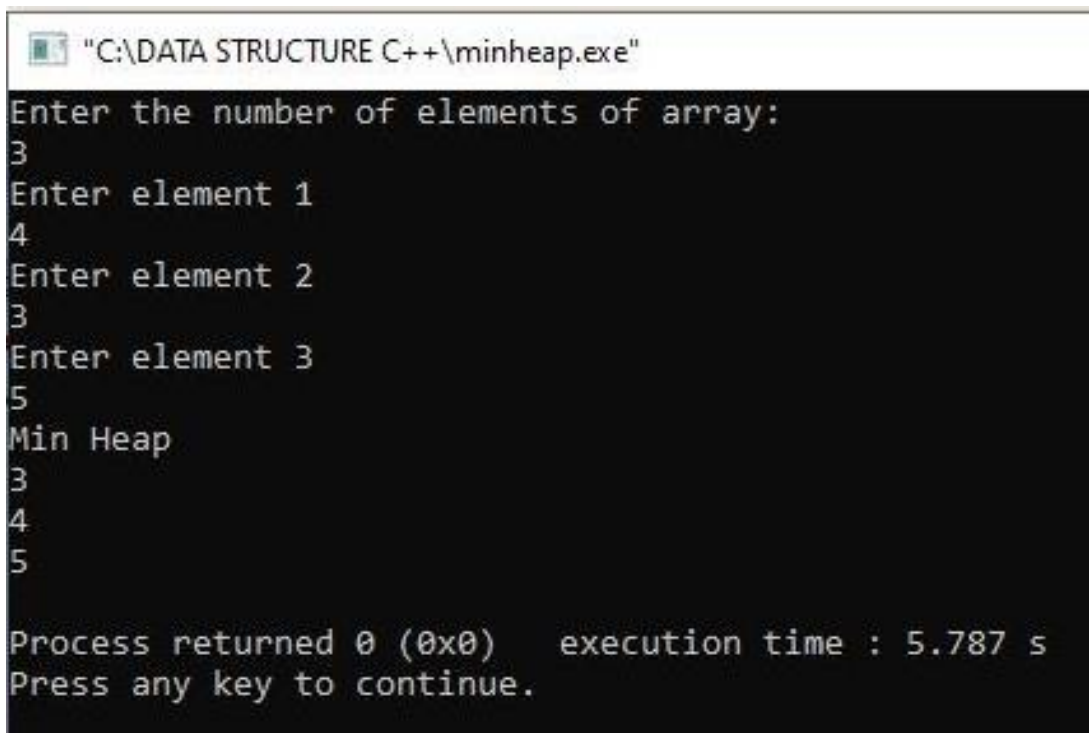
void build_minheap(int * a, int n)
{
    int i;
    for (i = n / 2; i >= 1; i--)
    {
        min_heapify(a, i, n);
    }
}

int main()
{
    int n, i, x;
    cout << "Enter the number of elements of array:\n";
    cin >> n;
    int a[20];
    for (i = 1; i <= n; i++)
    {
        cout << "Enter element " << (i) << endl;
        cin >> a[i];
    }
    build_minheap(a, n);
    cout << "Min Heap\n";
    for (i = 1; i <= n; i++)
    {
        cout << a[i] << endl;
    }
}

```

}

Output:



```
"C:\DATA STRUCTURE C++\minheap.exe"
Enter the number of elements of array:
3
Enter element 1
4
Enter element 2
3
Enter element 3
5
Min Heap
3
4
5

Process returned 0 (0x0)   execution time : 5.787 s
Press any key to continue.
```

## Practical 11

Q) Write a program to create a graph storage structure (eg. Adjacency matrix)

→Code:

```
#include<iostream>
using namespace std;

class adjMatrix
{
    int ** adj;

    bool * visited;

    int n, i, j;

public:
    adjMatrix(int n)
    {
        this -> n = n;

        visited = new bool[n];

        adj = new int * [n];

        for (i = 1; i <= n; i++)
        {
            adj[i] = new int[n];

            for (j = 1; j <= n; j++)
            {
                adj[i][j] = 0;
            }
        }
    }

    int add_edge(int origin, int dest)
```

```

{
    if (origin > n || dest > n || origin < 0 || dest < 0)
    {
        cout << "Wrong nodes";
    } else
    {
        adj[origin][dest] = 1;
    }
}

int display()
{
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            cout << adj[i][j] << "\t";
        }
        cout << "\n";
    }
}

};

int main()
{
    int nodes, Max_edges, i, origin, dest;

    cout << "Enter maximum node: ";

```

```
cin >> nodes;

adjMatrix am(nodes);

Max_edges = nodes * (nodes - 1);

cout << "Enter -1 -1 to exit";

for (i = 0; i < Max_edges; i++)
{
    cout << "\nEnter edges: ";

    cin >> origin >> dest;

    if ((origin == -1) && (dest == -1))

        break;

    else

        am.add_edge(origin, dest);
}

am.display();

return 0;
}
```

Output:

```
"C:\DATA STRUCTURE C++\adjancy.exe"
Enter maximum node: 3
Enter -1 -1 to exit
Enter edges: 1 2

Enter edges: 2 3

Enter edges: 3 3

Enter edges: 1 3

Enter edges: -1 -1
0      1      1
0      0      1
0      0      1

Process returned 0 (0x0)   execution time : 38.151 s
Press any key to continue.
```

## Practical 12

Perform various hashing techniques with Linear Probe as collision resolution scheme.

→ Code:

```
#include<iostream>
#include<conio.h>
#include<stdio.h>
#include<iomanip>

using namespace std;

const int SIZE = 10;

static int coll;

class hash1
{
    long key;
    long index;
    long arr[10];

    public:

        void directHash();
        void subHash();
        void modDivision();
        void linProbe();
        void digitExHash();
        void foldShiftHash();
        void foldBoundHash();
        void display();

};

void hash1::modDivision()
{
    for (int i = 0; i < 10; i++)

        arr[i] = -1;

    for (int i = 1; i <= 7; i++)
    {
```

```

int x;

cout << "\nEnter a number";

cin >> x;

index = x % 10;

while (arr[index] != -1)

    index = (index + 1) % 10;

arr[index] = x;

}

}

void hash1::display()

{

    cout << "\nHASH TABLE\n";

    for (int i = 0; i < 10; i++)

        cout << setw(8) << i;

    cout << "\n";

    for (int i = 0; i < 10; i++)

        cout << setw(8) << arr[i];

}

void hash1::directHash()

{

    for (int i = 0; i < 10; i++)

        arr[i] = -1;

    for (int i = 1; i <= 10; i++)

    {

        int x;

```

```

    cout << "Enter numbers from 1 to 10\n";

    cin >> x;

    int index = x;

    arr[index] = x;

}

}

void hash1::subHash()

{

    for (int i = 0; i < 10; i++)

        arr[i] = -1;

    for (int i = 1; i <= 7; i++)

    {

        int x;

        cout << "Enter numbers from 1001 to 1010\n";
        cin >> x;

        int index = x - 1000;

        arr[index] = x;

    }

}

void hash1::digitExHash()

{

    for (int i = 0; i < 10; i++)

        arr[i] = -1;

    for (int i = 1; i <= 10; i++)

    {

```

```

int x;

cout << "Enter a number of 6 digits\n";

cin >> x;

int index = 0;

long r, inc = 100000, incr = 1000;

for (int i = 1; i <= 6; i++)
{
    if (i == 1 || i == 3 || i == 5)
    {
        incr = incr / 10;

        r = (x / inc) % 10;

        index = index + (r * incr);
    }

    inc = inc / 10;
}

index = index % 10;

while (arr[index] != -1)

    index = (index + 1) % 10;

arr[index] = x;
}
}

void hash1::foldShiftHash()
{
    for (int i = 0; i < 10; i++)

        arr[i] = -1;

```

```

for (int i = 1; i <= 10; i++)
{
    int x;

    cout << "Enter a number of 4 digits\n";

    cin >> x;

    index = 0;

    long no, no1, no2, no3;

    no1 = x / 100;

    no3 = no1 * 100;

    no2 = x % no3;

    index = no1 + no2;

    index = index % 10;

    if (index == -1)
    {
        arr[index] = x;
    }

    while (arr[index] != -1)

        index = (index + 1) % 10;

    arr[index] = x;
}

}

void hash1::foldBoundHash()
{
    for (int i = 0; i < 10; i++)

        arr[i] = -1;

```

```

for (int i = 1; i <= 10; i++)
{
    int x;

    cout << "Enter a number of 4 digits\n";

    cin >> x;

    index = 0;

    long no, no1, no2, no3;

    no1 = x / 100;

    no3 = no1 * 100;

    no2 = x % no3;

    int tmp = 0;

    while (no1 > 0)
    {
        int rem = no1 % 10;

        tmp = (tmp * 10) + rem;

        no1 = no1 / 10;
    }

    int tmp1 = 0;

    while (no2 > 0)
    {
        int rem1 = no2 % 10;

        tmp1 = (tmp1 * 10) + rem1;

        no2 = no2 / 10;
    }

    index = tmp + tmp1;

    index = index % 10;

```

```

    if (index == -1)
    {
        arr[index] = x;
    }

    while (arr[index] != -1)

        index = (index + 1) % 10;

        arr[index] = x;
    }
}

int main()
{
    hash1 h;

    int op;

    cout << "Enter 1 for Direct Hashing\nEnter 2 for Subtraction Hashing\nEnter 3 for
Modulo Division Hashing" << endl;

    cout << "Enter 4 for Digit Extraction Hashing\nEnter 5 for Shift Fold Hashing\nEnter
6 for Shift Boundary Hashing" << endl;

    cout << "\nEnter 7 to exit\n" << endl;

    cin >> op;

    for (int i = 0; i < SIZE; i++)
    {
        switch (op)
        {
            case 1:

                h.directHash();

                h.display();

                break;

```

```
case 2:
    h.subHash();
    h.display();
    break;
case 3:
    h.modDivision();
    h.display();
    break;
case 4:
    h.digitExHash();
    h.display();
    break;
case 5:
    h.foldShiftHash();
    h.display();
    break;
case 6:
    h.foldBoundHash();
    h.display();
    break;
}
}
return 0;
}
```

Output:

## Practical 13

Q) Write a program to create a minimum spanning tree using any method Kruskal's Algorithm or Prim's Algorithm.

→Code:

```
#include<iostream>

#include<stdlib.h>

#define max 30

using namespace std;

struct edge
{
    int weight;

    int u;

    int v;

    struct edge * link;
};

struct edge * frnt = NULL;

struct edge * tmp;

int i, j, wt;

int father[max];

struct edge tree[max];

int wt_tree;

int cnt = 0;

void make_tree();

void insert_tree(int i, int j, int wt);
void insert_pque(int i, int j, int wt);
struct edge * del_pque();
```

```

void create_graph()
{
    int i, n, max_edges, origin, destin;

    cout << "Enter the no. of nodes : ";

    cin >> n;

    max_edges = n * (n - 1) / 2;

    for (i = 1; i < max_edges; i++)
    {
        cout << "Enter edges (0 0 to quit) weight : ";

        cin >> origin;

        cin >> destin;

        if ((origin == 0) && (destin == 0))

            break;

        cout << "Enter weight for this edge : ";
        cin >> wt;

        if (origin > n || destin > n || origin <= 0 || destin <= 0)
        {
            cout << "Invalid edge \n";

            i--;
        } else

            insert_pque(origin, destin, wt);
    }

    if (i < n - 1)
    {
        cout << "Spanning tree is not possible \n";
        exit(1);
    }
}

```

```

    }
}

void insert_pque(int i, int j, int wt)
{
    struct edge * tmp, * q;
    tmp = (struct edge * ) malloc(sizeof(struct edge));
    tmp -> u = i;
    tmp -> v = j;
    tmp -> weight = wt;
    if (frnt == NULL || tmp -> weight < frnt -> weight)
    {
        tmp -> link = frnt;
        frnt = tmp;
    } else
    {
        q = frnt;
        while (q -> link != NULL && q -> link -> weight <= tmp -> weight)
            q = q -> link;
        tmp -> link = q -> link;
        q -> link = tmp;
        if (q -> link == NULL)
            tmp -> link = NULL;
    }
}

void make_tree()

```

```

{
    edge * tmp;

    int node1, node2, root_n1, root_n2, wt_root = 0, n, cnt = 0;
    while (cnt < n - 1)
    {
        tmp = del_pque();

        node1 = tmp -> u;
        node2 = tmp -> v;

        cout << "N1 =" << node1;
        cout << "N2 =" << node2;

        while (node1 > 0)
        {
            root_n1 = node1;
            node1 = father[node1];
        }

        while (node2 > 0)
        {
            root_n2 = node2;
            node2 = father[node2];
        }

        cout << "root N1= " << root_n1;
        cout << "root N2= " << root_n2;

        if (root_n1 != root_n2)
        {
            insert_tree(tmp -> u, tmp -> v, tmp -> weight);
            wt_tree = wt_tree + tmp -> weight;
        }
    }
}

```

```

        father[root_n2] = root_n1;
    }
}
}

void insert_tree(int i, int j, int wt)
{
    cout << "This edge inserted in the spanning tree \n";
    cnt++;
    tree[cnt].u = i;
    tree[cnt].v = j;
    tree[cnt].weight = wt;
}

struct edge * del_pque()
{
    struct edge * tmp;
    tmp = frnt;
    cout << "Edge processed \n" << tmp -> u;
    cout << "Edge processed \n" << tmp -> v;
    cout << "Edge processed \n" << tmp -> weight;
    frnt = frnt -> link;
    return tmp;
}

int main()
{
    int i, j, wt_tree, cnt = 0;

```

```
struct edge tree[max];

create_graph();

make_tree();

cout << "Edges to be included in spanning tree \n";

for (i = 1; i <= cnt; i++)

{

    cout << tree[i].u;

    cout << tree[i].v;

}

cout << "Weight of this spanning tree is :" << wt_tree;
return 0;

}
```

Output:

```
"C:\DATA STRUCTURE C++\hashing.exe"
3
Enter numbers from 1 to 10
8
Enter numbers from 1 to 10
2
Enter numbers from 1 to 10
1

HASH TABLE
  0      1      2      3      4      5      6      7      8      9
-1      1      2      3      4      5      6      7      8      -1Enter numbers from 1 to 10
```

## Practical 14

Implementation of Graph traversal. (DFS and BFS)

1. Write a program to implement graph traversal.

→Code:

```
#include<iostream>
#include<stdio.h>
#define max 20
using namespace std;
int adj[max][max];
bool visited[max];
int n;
int frnt;
void create_graph()
{
    int i, max_edges, origin, destin;
    cout << "Enter no. of nodes: ";
    cin >> n;
    max_edges = n * (n - 1);
    for (i = 1; i <= max_edges; i++)
    {
        cout << "Enter edge (0 0 to quit) : " << i << "\n";
        cin >> origin >> destin;
        if (origin == 0 || destin == 0)
            break;
        if (origin > n || destin > n || origin <= 0 || destin <= 0)
        {
            cout << "Invalid edge \n";
            i--;
        } else
```

```

    {
        adj[origin][destin] = 1;
    }
}

void display()
{
    int i, j;
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            cout << adj[i][j] << "\t";
        }
        cout << "\n";
    }
}

void dfs(int v)
{
    int i, stack[max], top = -1, pop_v, j, t;
    int c;

    top++;
    stack[top] = v;
    while (top >= 0)
    {
        pop_v = stack[top];
        top--;
        if (visited[pop_v] == false)
        {

```

```

    cout << pop_v;
    visited[pop_v] = true;
} else
    continue;
for (i = n; i >= 1; i--)
{
    if (adj[pop_v][i] == 1 && visited[i] == false)
    {
        top++;
        stack[top] = i;
    }
}
}
}

void bfs(int v)
{
    int i, frnt, rear;
    int que[20];
    frnt = rear = -1;
    cout << v;
    visited[v] = true;
    rear++;
    frnt++;
    que[rear] = v;
    while (frnt <= rear)
    {
        v = que[frnt];
        frnt++;
        for (i = 1; i <= n; i++)

```

```

{
    if (adj[v][i] == 1 && visited[i] == false)
    {
        cout << i << "\t";
        visited[i] = true;
        rear++;
        que[rear] = i;
    }
}
}

void adj_nodes(int v)
{
    int i;
    for (i = 1; i <= n; i++)
    {
        int i;
        for (i = 1; i <= n; i++)
        {
            if (adj[v][i] == 1)
            {
                cout << i;
                cout << "\n";
            }
        }
    }
}

int main()
{
    int i, v, ch;

```

```

create_graph();
while (1)
{
    cout << "\n";
    cout << "1. Adjacency Matrix \n";
    cout << "2. Depth first search using stack\n";
    cout << "3. Breadth first search\n";
    cout << "4. Exit \n";
    cout << "Enter your choice\n";
    cin >> ch;
    switch (ch)
    {
        case 1:
            cout << "Adjacency Matrix \n";
            display();
            break;
        case 2:
            cout << "Enter starting node for Depth First Search: \n";
            cin >> v;
            for (i = 1; i <= n; i++)
                visited[i] = false;
            dfs(v);
            break;
        case 3:
            cout << "Enter starting node for Breadth First Search: \n";
            cin >> v;
            for (i = 1; i <= n; i++)
                visited[i] = false;
            bfs(v);

```

```
    break;
case 4:
    break;
default:
    cout << "Wrong Choice";
    break;
}
}
return 0;
}
```

Output:

```
"C:\DATA STRUCTURE C++\BST_DSF.exe"
2. Depth first search using stack
3. Breadth first search
4. Exit
Enter your choice
2
Enter starting node for Depth First Search:
3
3
1. Adjacency Matrix
2. Depth first search using stack
3. Breadth first search
4. Exit
Enter your choice
3
Enter starting node for Breadth First Search:
5
5
1. Adjacency Matrix
2. Depth first search using stack
3. Breadth first search
4. Exit
Enter your choice
3
Enter starting node for Breadth First Search:
4
4
1. Adjacency Matrix
2. Depth first search using stack
3. Breadth first search
4. Exit
Enter your choice
```

"C:\DATA STRUCTURE C++\BST\_DSF.exe"

```
2. Depth first search using stack
3. Breadth first search
4. Exit
Enter your choice
2
Enter starting node for Depth First Search:
3
3
1. Adjacency Matrix
2. Depth first search using stack
3. Breadth first search
4. Exit
Enter your choice
3
Enter starting node for Breadth First Search:
5
5
1. Adjacency Matrix
2. Depth first search using stack
3. Breadth first search
4. Exit
Enter your choice
3
Enter starting node for Breadth First Search:
4
4
1. Adjacency Matrix
2. Depth first search using stack
3. Breadth first search
4. Exit
Enter your choice
```