Bangladesh University of Business & Technology (BUBT)

Department of Computer Science and Engineering

Lab Assignment: Spring-2021

Course Code: CSE 242 | Course Title: Algorithms Lab

Intake: 44th Program: B.Sc. in CSE

Submitted by

Name: Md. Al Emam

Id: 19202103242

Problem 1:

Write a program to build a max heap using the given data.

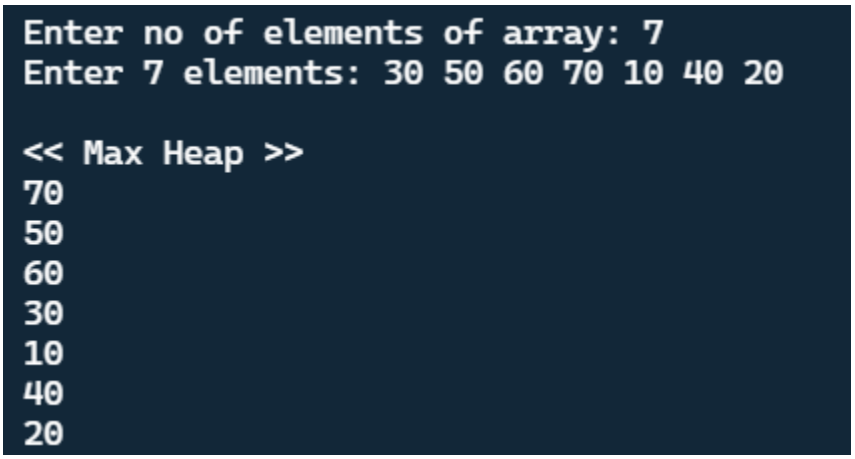
Sample Input: 30 50 60 70 10 40 20

Solution:

```
#include <bits/stdc++.h>
using namespace std;
void max_heap(int a[], int i, int n)
{
   int j, t;
   t = a[i];
   j = 2 * i;
   while (j <= n)
   {
      if (j < n && a[j + 1] > a[j])
         j = j + 1;
      if (t > a[j])
         break;
      else if (t <= a[j])
      {
         a[j / 2] = a[j];
         j = 2 * j;
      }
   }
   a[j / 2] = t;
   return;
}
void build_maxheap(int  a[], int n)
```

```
{
   int i;
   for (i = n / 2; i >= 1; i--)
   {
      max_heap(a, i, n);
   }
}
int main()
{
   int n, i;
   cout << "Enter no of elements of array: ";
   cin >> n;
   int a[30];
   cout << "Enter " << n << " elements: ";
   for (i = 1; i <= n; i++)
   {
      cin >> a[i];
   }
   build_maxheap(a, n);
   cout << "\n<< Max Heap >>\n";
   for (i = 1; i <= n; i++)
   {
      cout << a[i] << endl;
   }
}
```

Output:



Problem 2:

Write down a program to find a single source shortest paths using Dijkstra's Shortest Path Algorithm.

Solution:

```
#include<bits/stdc++.h>
using namespace std;
#define MAX 100
#define INF 9999

void dijk(int adj[MAX][MAX],int n,int start)
{
   int cost[MAX][MAX],dist[MAX],visited[MAX],pred[MAX];
   int i,j,counts,mindist,nextnode;

   for(i=0;i<n;i++)
   {
```

```
   for(j=0;j<n;j++)
   {
      if(adj[i][j]==0)
      {
         cost[i][j]=INF;
      }
      else
      {
         cost[i][j]=adj[i][j];
      }
   }
}

for(i=0;i<n;i++)
{
   dist[i]=cost[start][i];
   pred[i]=start;
   visited[i]=0;
}

dist[start]=0;
visited[start]=1;
counts=1;
while(counts<n-1)
{
   mindist=INF;
   for(i=0;i<n;i++)
   {
      if(dist[i] < mindist  && !visited[i])
```

```
      {
         mindist=dist[i];
         nextnode=i;
      }
   }
   visited[nextnode]=1;
   for(i=0;i<n;i++)
   {
      if(!visited[i])
      {
         if(mindist+cost[nextnode][i]  < dist[i])
         {
            dist[i]=mindist+cost[nextnode][i];
            pred[i]=nextnode;
         }
      }
   }
   counts++;
}

for(i=0;i<n;i++)
{
   if(i!=start)
   {
      cout << "The distance of node " <<i<< " is " << dist[i] << endl;
      cout << "The path is: " << i;

      j=i;
      do
```

```
        {
            j=pred[j];
            cout << "<-" << j;
        }while(j!=start);
        cout << endl;
      }
   }
}
int main()
{
   int adj[MAX][MAX],start;
   int i,j,n,e;
   cout << "Enter no of node & edge: ";
   cin>>n>>e;
   cout << "Enter the graph:\n";
   for(i=0;i<e;i++)
   {
      int n1,n2,w;
      cin>>n1>>n2>>w;
      adj[n1][n2]=w;
      adj[n2][n1]=w;

   }
   /*cout << endl;
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         cout << adj[i][j] << "   ";
```

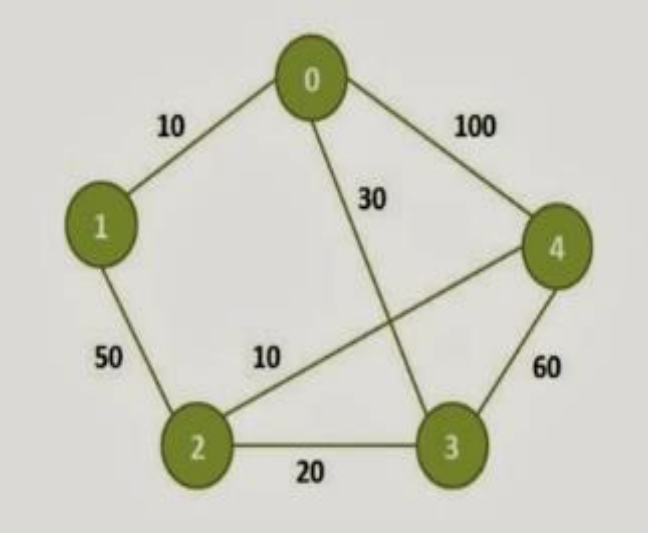
```
    }
    cout << endl;
  }*/

  cout << "Enter the starting node: ";
  cin>>start;

  dijk(adj,n,start);

  return 0;
}
```

Graph:



Output:

```
Enter no of node & edge: 5 7
Enter the graph:
0 1 10
0 3 30
0 4 100
1 2 50
2 3 20
2 4 10
3 4 60
Enter the starting node: 0
The distance of node 1 is 10
The path is: 1<-0
The distance of node 2 is 50
The path is: 2<-3<-0
The distance of node 3 is 30
The path is: 3<-0
The distance of node 4 is 60
The path is: 4<-2<-3<-0
```

Problem 3:

Write down a program to find a single source shortest paths using Bellman Ford Shortest Path Algorithm.

Solution:

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

```
using namespace std;
#define INF 9999

vector<vector<pair<int,int>>> adj(1000);
vector<int>dis;
vector<int>par;
int source;

int pathfind(int dest)
{
   if(dest!=source && par[dest]==-1)
   {
      cout << "Path not found\n";
      return 0;
   }
   if(dest==source)
   {
      cout << "Path: " << source;
      return 0;
   }
   pathfind(par[dest]);
   cout << " " << dest;
}
int main()
{
   ifstream input("bellmanford_input.txt");
   if(!input.is_open())
   {
      cout << "File not open!\n";
```

```
}

int totalnode,node1,node2,weight;
input>>totalnode;
while(input>>node1>>node2>>weight)
{
   adj[node1].push_back(make_pair(node2,weight));
}

/*for(int i=0;i<totalnode;i++)
{
   cout << i << ": ";
   for(int j=0;j<adj[i].size();j++)
   {
      cout << adj[i][j].first << "("<<adj[i][j].second<<")->";
   }
   cout << endl;
}*/

cout << "Enter source: ";
cin>> source;

dis.assign(totalnode,INF);
par.assign(totalnode,-1);
dis[source]=0;

for(int i=0; i<totalnode-1; i++)
{
   for(int u=0; u<totalnode; u++)
```

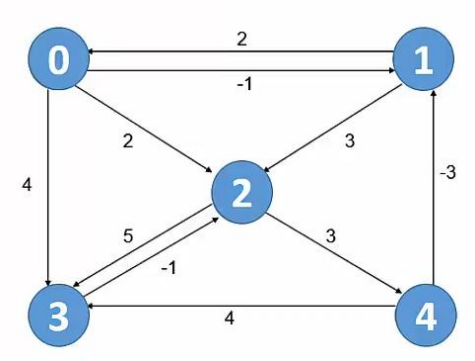
```
    {
        for(int j=0; j<adj[u].size(); j++)
        {
            pair<int,int> v=adj[u][j];
            if(dis[u]==INF)
                continue;
            if(dis[u]+v.second<dis[v.first])
            {
                dis[v.first]=dis[u]+v.second;
                par[v.first]=u;
            }
        }
    }
}

bool hascycle = false;
for(int u=0; u<totalnode; u++)
{
    for(int j=0; j<adj[u].size(); j++)
    {
        pair<int,int> v=adj[u][j];
        if(dis[u]==INF)
            continue;
        if(dis[u]+v.second<dis[v.first])
        {
            hascycle = true;
            break;
        }
    }
```

```
        if(hascycle)
        {
            break;
        }
    }

    if(hascycle)
    {
        cout << "The graph has negative cycle\n";
    }
    else
    {
        int dest;
        cout << "Enter destination: ";
        cin>>dest;
        cout << "Distance: " << dis[dest] << endl;
        pathfind(dest);
    }
    return 0;
}
```

Graph 1:



Input File 1:

bellmanford_input.txt

File Edit Format Vie

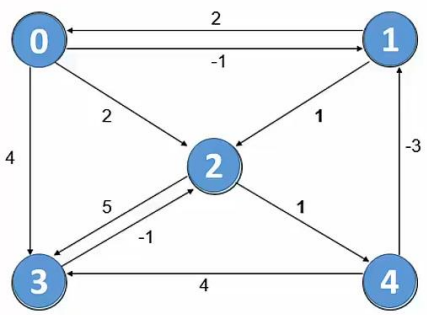
```
5
0 1 -1
0 2 2
0 3 4
1 0 2
1 2 3
2 3 5
2 4 3
3 2 -1
4 3 4
4 1 -3
```

Output 1:

```
Enter source: 3
Enter destination: 0
Distance: 1
Path: 3 2 4 1 0
Process returned 0 (0x0)
```

```
Enter source: 0
Enter destination: 4
Distance: 5
Path: 0 2 4
Process returned 0 (0x0)
```

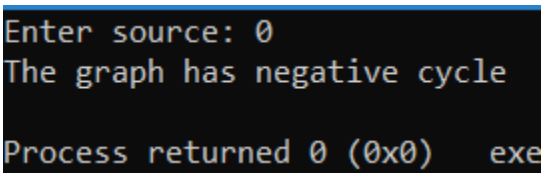
Graph 2:



Input File 2:

```
5
0 1 -1
0 2 2
0 3 4
1 0 2
1 2 1
2 3 5
2 4 1
3 2 -1
4 3 4
4 1 -3
```

Output 2:



Problem 4:

Write down a program to find all pair shortest paths using Floyd Warshall Algorithm.

Solution:

```
#include<bits/stdc++.h>
using namespace std;
#define INF 9999

int adj[100][100];
int parent[100][100];

int pathfind(int  source,int des)
{
   if(source==des)
   {
      cout << "Path: " << source;
      return 0;
   }
   pathfind(source,parent[source][des]);
   cout << "->" << des;
}
void floyd(int totalnode)
```

```
{
    for(int k=1; k<=totalnode; k++)
    {
        for(int i=1; i<=totalnode; i++)
        {
            for(int j=1; j<=totalnode; j++)
            {
                if(adj[i][k]+adj[k][j] < adj[i][j])
                {
                    adj[i][j]=adj[i][k]+adj[k][j];
                    parent[i][j]=parent[k][j];
                }
            }
        }
    }
}
int main()
{
    ifstream input("floyd_input.txt");
    if(!input.is_open())
    {
        cout << "File not open!\n";
    }

    int totalnode,node1,node2,weight;
    input>>totalnode;

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

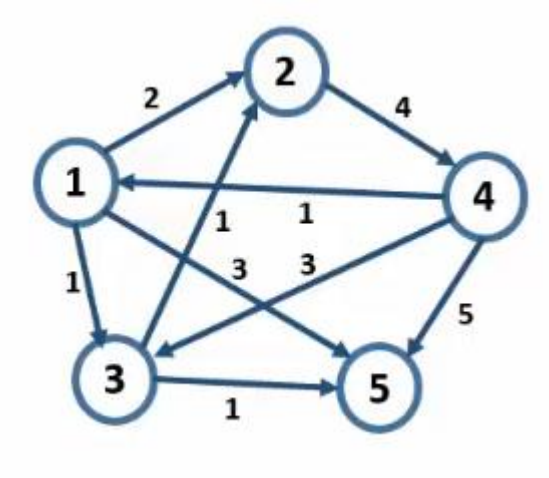
```
    for(int j=1; j<=totalnode; j++)
    {
        if(i==j)
        {
            adj[i][j]=0;
        }
        else
        {
            adj[i][j]=INF;
        }
        parent[i][j]=i;
    }
}
while(input>>node1>>node2>>weight)
{
    adj[node1][node2]=weight;
}

floyd(totalnode);

for(int i=1;i<=totalnode;i++)
{
    for(int j=1;j<=totalnode;j++)
    {
        cout << "Cost from " << i << " to " << j << ": " << adj[i][j] << endl;
        pathfind(i,j);
        cout << endl << endl;
    }
}
```

```
    return 0;
}
```

Graph:



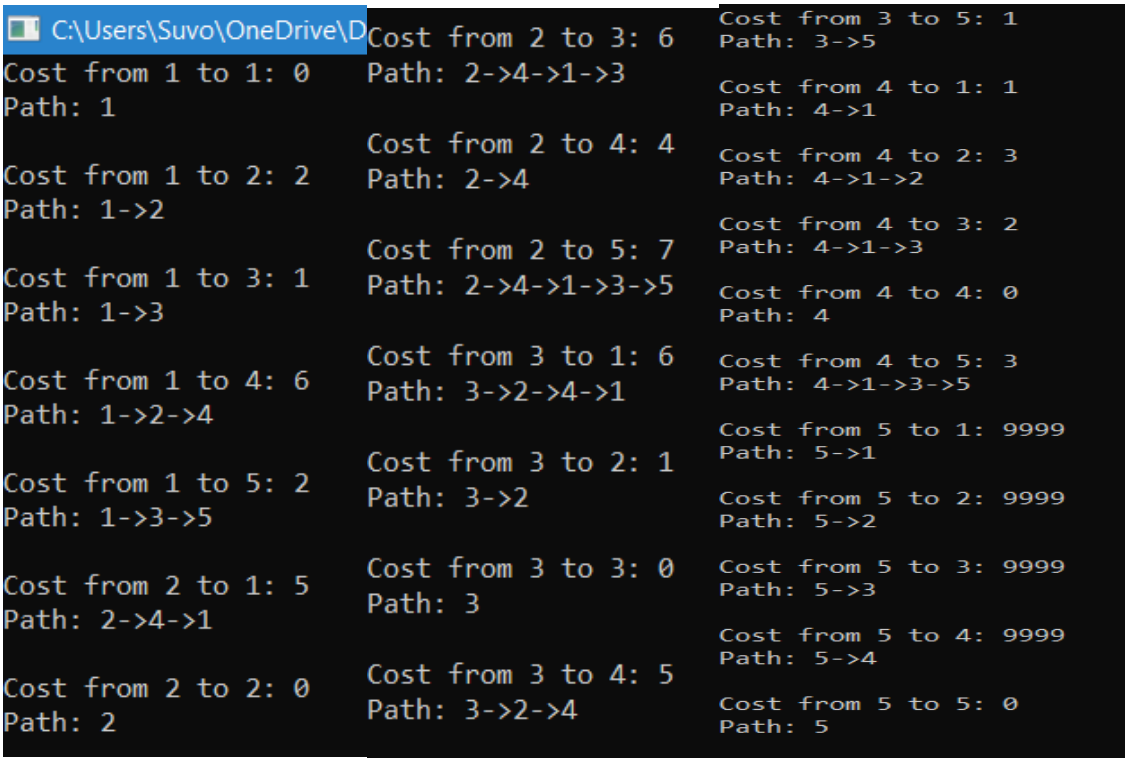
Input File:

floyd_input.txt -

File Edit Format

```
5
1 2 2
1 3 1
1 5 3
2 4 4
3 2 1
3 5 1
4 1 1
4 3 3
4 5 5
```

Output:


C:\Users\Suvo\OneDrive\D
Cost from 1 to 1: 0
Path: 1
Cost from 1 to 2: 2
Path: 1->2
Cost from 1 to 3: 1
Path: 1->3
Cost from 1 to 4: 6
Path: 1->2->4
Cost from 1 to 5: 2
Path: 1->3->5
Cost from 2 to 1: 5
Path: 2->4->1
Cost from 2 to 2: 0
Path: 2
Cost from 2 to 3: 6
Path: 2->4->1->3
Cost from 2 to 4: 4
Path: 2->4
Cost from 2 to 5: 7
Path: 2->4->1->3->5
Cost from 3 to 1: 6
Path: 3->2->4->1
Cost from 3 to 2: 1
Path: 3->2
Cost from 3 to 3: 0
Path: 3
Cost from 3 to 4: 5
Path: 3->2->4
Cost from 3 to 5: 1
Path: 3->5
Cost from 4 to 1: 1
Path: 4->1
Cost from 4 to 2: 3
Path: 4->1->2
Cost from 4 to 3: 2
Path: 4->1->3
Cost from 4 to 4: 0
Path: 4
Cost from 4 to 5: 3
Path: 4->1->3->5
Cost from 5 to 1: 9999
Path: 5->1
Cost from 5 to 2: 9999
Path: 5->2
Cost from 5 to 3: 9999
Path: 5->3
Cost from 5 to 4: 9999
Path: 5->4
Cost from 5 to 5: 0
Path: 5