**A Practical Activity Report For**

**Data Structures and Algorithms (UCS406)**

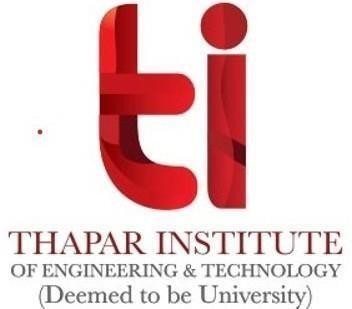
Submitted By: **Vivek Arora**

**101715178**

**(ENC 8)**

Submitted To:

**Dr. Sanjay Sharma**



**ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT THAPAR INSTITUTE OF ENGINEERING &TECHNOLOGY, (DEEMED TO BEUNIVERSITY),**

**PATIALA, PUNJAB**

**ASSIGNMENT 11**

**QUESTION 1:**

Write a program for the implementation of Breadth First Search(BFS) for a given

Graph.

#include<iostream>

#include<list>

using namespace std;

class grph

{

int V;

list&lt;int&gt; \*adj;

public:

grph(int V);

void adEg(int v, int w);

void BFS(int s);

};

grph::grph(int V)

{

this-&gt;V = V;

adj = new list&lt;int&gt;[V];

}

void grph::adEg(int v, int w)

{

adj[v].push\_back(w);

}

void grph::BFS(int s)

{

bool \*vsitd = new bool[V];

for(int i = 0; i &lt; V; i++)

vsitd[i] = false;

list&lt;int&gt; queue;

vsitd[s] = true;

queue.push\_back(s);

list&lt;int&gt;::iterator i;

while(!queue.empty())

{

s = queue.front();

cout &lt;&lt; s &lt;&lt; &quot; &quot;;

queue.pop\_front();

for (i = adj[s].begin(); i != adj[s].end(); ++i)

{

if (!vsitd[\*i])

{

vsitd[\*i] = true;

queue.push\_back(\*i);

}

}

}

}

int main()

{

grph g(4);

g.adEg(0, 3);

g.adEg(0, 2);

g.adEg(1, 2);

g.adEg(2, 0);

g.adEg(3, 1);

g.adEg(3, 3);

cout &lt;&lt; &quot;Following is Breadth First Traversal &quot;

&lt;&lt; &quot;(starting from vertex 2) \n&quot;;

g.BFS(2);

return 0;

}

**QUESTION 2:**

**Write a program for the implementation of Depth First Search(DFS) for a given graph**

#include<iostream>

#include<list>

using namespace std;

class grph

{

int V;

list&lt;int&gt; \*adj;

void DFSUtil(int v, bool vstd[]);

public:

grph(int V); // Constructor

void adEg(int v, int w);

void DFS(int v);

};

grph::grph(int V)

{

this-&gt;V = V;

adj = new list&lt;int&gt;[V];

}

void grph::adEg(int v, int w)

{

adj[v].push\_back(w);

}

void grph::DFSUtil(int v, bool vstd[])

{

vstd[v] = true;

cout &lt;&lt; v &lt;&lt; &quot; &quot;;

list&lt;int&gt;::iterator i;

for (i = adj[v].begin(); i != adj[v].end(); ++i)

if (!vstd[\*i])

DFSUtil(\*i, vstd);

}

void grph::DFS(int v)

{

bool \*vstd = new bool[V];

for (int i = 0; i &lt; V; i++)

vstd[i] = false;

DFSUtil(v, vstd);

}

int main()

{

grph g(4);

g.adEg(0, 1);

g.adEg(1, 2);

g.adEg(2, 2);

g.adEg(2, 0);

g.adEg(2, 3);

g.adEg(3, 3);

cout &lt;&lt; &quot;Following is Depth First Traversal&quot;

&quot; (starting from vertex 2) \n&quot;;

g.DFS(2);

return 0;

}

**QUESTION 3:** **Write a program for Dijkstra’s Shortest path algorithm for a given graph.**

#include<iostream>

#include<climits>

using namespace std;

int findMinVertex(int\* distance, bool\* visited, int n){

    int minVertex = -1;

    for(int i = 0; i < n; i++){

    if(!visited[i] && (minVertex == -1 ||  distance[i] < distance[minVertex])){

    minVertex = i;

    }

    }

    return minVertex;

}

void dijkstra(int\*\* edges, int n){

    int\* distance = new int[n];

    bool\* visited = new bool[n];

    for(int i = 0; i < n; i++){

    distance[i] = INT\_MAX;

    visited[i] = false;

    }

    distance[0] = 0;

    for(int i = 0; i < n - 1; i++){

    int minVertex = findMinVertex(distance, visited, n);

    visited[minVertex] = true;

    for(int j = 0; j < n; j++){

    if(edges[minVertex][j] != 0 && !visited[j]){

    int dist = distance[minVertex] + edges[minVertex][j];

    if(dist < distance[j]){

    distance[j] = dist;

    }

    }

    }

    }

    for(int i = 0; i < n; i++){

    cout << i << " " << distance[i] << endl;

    }

    delete [] visited;

    delete [] distance;

}

int main() {

    int n;

    int e;

    cin >> n >> e;

    int\*\* edges = new int\*[n];

    for (int i = 0; i < n; i++) {

    edges[i] = new int[n];

    for (int j = 0; j < n; j++) {

    edges[i][j] = 0;

    }

    }

    for (int i = 0; i < e; i++) {

    int f, s, weight;

    cin >> f >> s >> weight;

    edges[f][s] = weight;

    edges[s][f] = weight;

    }

    cout << endl;

    dijkstra(edges, n);

    for (int i = 0; i < n; i++) {

    delete [] edges[i];

    }

    delete [] edges;

}

**QUESTION 4:**

**Write a program to for Prim’s and Kruskal algorithms for finding the minimum spanning tree.**

**Prim’s Algorithm :-**

#include<iostream>

#include<climits>

using namespace std;

int findMinVertex(int\* weights, bool\* visited, int n){

    int minVertex = -1;

    for(int i = 0; i < n; i++){

    if(!visited[i] && (minVertex == - 1 || weights[i] < weights[minVertex])){

    minVertex = i;

    }

    }

    return minVertex;

}

void prims(int\*\* edges, int n){

    int\* parent = new int[n];

    int\* weights = new int[n];

    bool\* visited = new bool[n];

    for(int i = 0; i < n; i++){

    visited[i] = false;

    weights[i] = INT\_MAX;

    }

    parent[0] = -1;

    weights[0] = 0;

    for(int i = 0; i < n - 1; i++){

    // Find Min Vertex

    int minVertex = findMinVertex(weights, visited, n);

    visited[minVertex] = true;

    // Explore un visted neighbours

    for(int j = 0; j < n; j++){

    if(edges[minVertex][j] != 0 && !visited[j]){

    if(edges[minVertex][j] < weights[j]){

    weights[j] = edges[minVertex][j];

    parent[j] = minVertex;

    }

    }

    }

    }

    for(int i = 1; i < n; i++){

    if(parent[i] < i){

    cout << parent[i] < " << i << " " << weights[i] << endl;

    }else{

    cout << i << " " << parent[i] << " " << weights[i] << endl;

    }

    }

}

int main() {

    int n;

    int e;

    cin >> n >> e;

    int\*\* edges = new int\*[n];

    for (int i = 0; i < n; i++) {

    edges[i] = new int[n];

    for (int j = 0; j < n; j++) {

    edges[i][j] = 0;

    }

    }

    for (int i = 0; i < e; i++) {

    int f, s, weight;

    cin >> f >> s >> weight;

    edges[f][s] = weight;

    edges[s][f] = weight;

    }

    cout << endl;

    prims(edges, n);

    for (int i = 0; i < n; i++) {

    delete [] edges[i];

    }

    delete [] edges;

}

**Kruksal’s Algorithm**

#include <iostream>

#include <vector>

#include <utility>

#include <algorithm>

using namespace std;

const int MAX = 1e4 + 5;

int id[MAX], nodes, edges;

pair <long long, pair<int, int> > p[MAX];

void initialize()

{

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

     id[i] = i;

}

int root(int x)

{

while(id[x] != x)

{

     id[x] = id[id[x]];

     x = id[x];

}

return x;

}

void union1(int x, int y)

{

int p = root(x);

int q = root(y);

id[p] = id[q];

}

long long kruskal(pair<long long, pair<int, int> > p[])

{

int x, y;

long long cost, minimumCost = 0;

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

{

     // Selecting edges one by one in increasing order from the beginning

     x = p[i].second.first;

     y = p[i].second.second;

     cost = p[i].first;

     // Check if the selected edge is creating a cycle or not

     if(root(x) != root(y))

     {

         minimumCost += cost;

         union1(x, y);

     }

}

return minimumCost;

}

int main()

{

int x, y;

long long weight, cost, minimumCost;

initialize();

cin >> nodes >> edges;

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

{

     cin >> x >> y >> weight;

     p[i] = make\_pair(weight, make\_pair(x, y));

}

// Sort the edges in the ascending order

sort(p, p + edges);

minimumCost = kruskal(p);

cout << minimumCost << endl;

return 0;

}

**QUESTION 5:** **Write a program using Greedy approach for fractional knapsack problem.**

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

typedef struct {

   int v;

   int w;

   float d;

} Item;

void input(Item items[],int sizeOfItems) {

   cout << "Enter total "<< sizeOfItems <<" item's values and weight" <<

   endl;

   for(int I = 0; I < sizeOfItems; i++) {

   cout << "Enter "<< i+1 << " V ";

   cin >> items[i].v;

   cout << "Enter "<< i+1 << " W";

   cin >> items[i].w;

   }

}

void display(Item items[], int sizeOfItems) {

   int i;

   cout << "values: ";

   for(i = 0; i < sizeOfItems; i++) {

   cout << items[i].v << "\t";

   }

   cout << endl << "weight: ";

   for (I = 0; I < sizeOfItems; i++) {

   cout << items[i].w << "\t";

   }

   cout << endl;

}

bool compare(Item i1, Item i2) {

   return (i1.d > i2.d);

}

float knapsack(Item items[], int sizeOfItems, int W) {

   int i, j, pos;

   Item mx, temp;

   float totalValue = 0, totalWeight = 0;

   for (i = 0; i < sizeOfItems; i++) {

   items[i].d = items[i].v / items[i].w;

   }

   sort(items, items+sizeOfItems, compare);

   for(i=0; i<sizeOfItems; i++) {

   if(totalWeight + items[i].w<= W) {

      totalValue += items[i].v ;

      totalWeight += items[i].w;

   } else {

      int wt = W-totalWeight;

      totalValue += (wt \* items[i].d);

      totalWeight += wt;

      break;

   }

   }

   cout << "total weight in bag " << totalWeight<<endl;

   return totalValue;

}

int main() {

   int W;

   Item items[4];

   input(items, 4);

   cout << "Entered data \n";

   display(items,4);

   cout<< "Enter Knapsack weight \n";

   cin >> W;

   float mxVal = knapsack(items, 4, W);

   cout << "Max value for "<< W <<" weight is "<< mxVal;

}

**QUESTION 6:** **Write a program using dynamic programming strategy for finding the Fibonacci value of 5 th term.**

#include <iostream>

using namespace std;

int fibo\_dp(int n) {

   int \*ans = new int[n+1];

   ans[0] = 0;

   ans[1] = 1;

   for(int i = 2; i <= n; i++) {

   ans[i] = ans[i-1] + ans[i-2];

   }

   return ans[n];

}

int main() {

   cout << fibo\_dp(5) << endl;

}