**WORKSHEET 2.4**

**1. Aim:**

Write a program to construct minimum spanning tree using-

(a) Kruskal’s Algorithm (b) Prim’s Algorithm

**2. Problem Description:**

To implement Kruskal’s and prim’s algorithm in minimum spanning tree

**Kruskal’s Algorithm -**

**3. Algorithm:**

for each vertex *V* in *G* do

define a *Cloud(v)* of 🡨 {*v*}

let *Q* be a priority queue.

Insert all edges into *Q* using their weights as the key

*T* 🡨 ∅

while *T* has fewer than *n*-1 edges do

edge *e* = *T.removeMin()*

Let *u*, *v* be the endpoints of *e*

if *Cloud(v)* ≠ *Cloud(u)* then

Add edge *e* to *T*

Merge *Cloud(v)* and *Cloud(u)*

return *T*

**4. Computational Complexity:-**

The time complexity Of Kruskal's Algorithm is: O(E log E).

**5. Pseudo Code :-**

**MST- KRUSKAL (G, w)**

Step 1. A ← ∅

Step 2. for each vertex v ∈ V [G]

Step 3. do MAKE - SET (v)

Step 4. sort the edges of E into non decreasing order by weight w

Step 5. for each edge (u, v) ∈ E, taken in non decreasing order by weight

Step 6. do if FIND-SET (μ) ≠ if FIND-SET (v)

Step 7. then A ← A ∪ {(u, v)}

Step 8. UNION (u, v)

Step 9. return A

**6. Source Code:**

#include <iostream>

#include <algorithm>

using namespace std;

const int MAX = 1e4 + 5;

int id[MAX], nodes, edges;

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

void init()

{

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)

{

x = p[i].second.first;

y = p[i].second.second;

cost = p[i].first;

if(root(x) != root(y))

{

minimumCost += cost;

union1(x, y);

}

}

return minimumCost;

}

int main()

{

int x, y;

long long weight, cost, minimumCost;

init();

cout <<"Enter Nodes and edges"<<endl;

cin >> nodes >> edges;

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

{

cout<<"Enter the value of X, Y and edges"<<endl;

cin >> x >> y >> weight;

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

}

sort(p, p + edges);

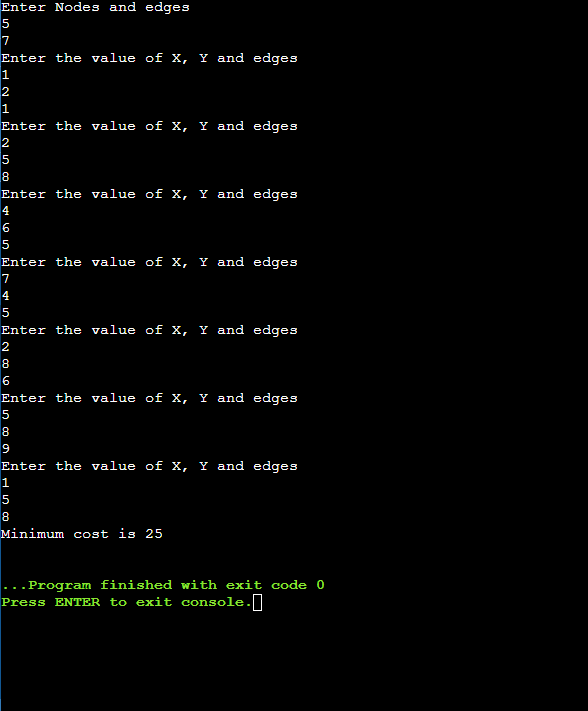
minimumCost = kruskal(p);

cout <<"Minimum cost is "<< minimumCost << endl;

return 0;

}

1. **Screenshot of Output:**



**Prim’s Algorithm -**

**8. Algorithm:**

*Q* ←new heap-based priority queue

*s* ←a vertex of *G*

for all *v* ∈ *G.vertices*()

if *v* = *s*

*setDistance*(*v,* 0)

else

*setDistance*(*v,* ∞)

*setParent*(*v,* ∅)

*l* ← *Q.insert*(*getDistance*(*v*)*,* *v*)

*setLocator*(*v,l*)

while ¬*Q.isEmpty*()

*u* ← *Q.removeMin*()

for all *e* ∈ *G.incidentEdges*(*u*)

*z* ← *G.opposite*(*u,e*)

*r* ← *weight*(*e*)

if *r* < *getDistance*(*z*)

*setDistance*(*z,r*)

*setParent*(*z,e*)  
 *Q.replaceKey*(*getLocator*(*z*)*,r*)

**9. Computational Complexity:-**

Time Complexity : O(V2), If the input [graph is represented using an adjacency list](https://www.geeksforgeeks.org/archives/27134), then the time complexity of Prim’s algorithm can be reduced to O(E \* logV) with the help of a binary heap.

In this implementation, we are always considering the spanning tree to start from the root of the graph

Auxiliary Space Complexity : O(V)

**10. Pseudo Code :-**

T = ∅;

U = { 1 };

while (U ≠ V)

let (u, v) be the lowest cost edge such that u ∈ U and v ∈ V - U;

T = T ∪ {(u, v)}

U = U ∪ {v}

**11. Source Code:**

#include <stdio.h>

#include <limits.h>

#define vertices 5

int minimum\_key(int k[], int mst[])

{

int minimum = INT\_MAX, min,i;

for (i = 0; i < vertices; i++)

if (mst[i] == 0 && k[i] < minimum )

minimum = k[i], min = i;

return min;

}

void prim(int g[vertices][vertices])

{ int s=0;

int parent[vertices];

int k[vertices];

int mst[vertices];

int i, count,edge,v;

for (i = 0; i < vertices; i++)

{

k[i] = INT\_MAX;

mst[i] = 0;

}

k[0] = 0;

parent[0] = -1;

for (count = 0; count < vertices-1; count++)

{

edge = minimum\_key(k, mst);

mst[edge] = 1;

for (v = 0; v < vertices; v++)

{

if (g[edge][v] && mst[v] == 0 && g[edge][v] < k[v])

{

parent[v] = edge, k[v] = g[edge][v];

}

}

}

printf("\n Edge \t Weight\n");

for (i = 1; i < vertices; i++) {

printf(" %d <-> %d %d \n", parent[i], i, g[i][parent[i]]);

s=s+g[i][parent[i]];

}

printf( "Overall Minimum Weight :");

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

}

int main()

{

int g[vertices][vertices] = {{0, 0, 3, 0, 0},

{0, 0, 10, 4, 0},

{3, 10, 0, 2, 6},

{0, 4, 2, 0, 1},

{0, 0, 6, 1, 0},

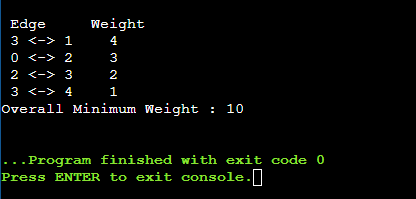
};

prim(g);

return 0;

}

1. **Screenshot of Output:**



1. **Learning & Outcomes:**

* Learned about the minimum spanning tree, how it works.
* Examined the Kruskal’s Algorithm
* Also learnt Prim’s Algorithm
* Additionally, also learned how to build an MST for a given graph using Kruskal's Algoritm and Prim’s Algorithm.
* Learnt the complexities of both the algorithm.