**Task 8 Minimum Spanning Tree using Kruskal’s Algorithm**

**Greedy Technique – Kruskal’s algorithm**

Given an undirected weighted connected graph, find the Really Special SubTree in it. The Really Special SubTree is defined as a subgraph consisting of all the nodes in the graph and:

* There is only one exclusive path from a node to every other node.
* The subgraph is of minimum overall weight (sum of all edges) among all such subgraphs.
* No cycles are formed

To create the Really Special SubTree, always pick the edge with smallest weight. Determine if including it will create a cycle. If so, ignore the edge. If there are edges of equal weight available:

* Choose the edge that minimizes the sum  where  and  are vertices and  is the edge weight.
* If there is still a collision, choose any of them.

Print the overall weight of the tree formed using the rules.

For example, given the following edges:

u v wt

1 2 2

2 3 3

3 1 5

First choose 1→2  at weight 2 . Next choose  2→3at weight 3 . All nodes are connected without cycles for a total weight of .3+2 = 5.

**Input Format**

The first line has two space-separated integers g\_nodes and g\_edges , the number of nodes and edges in the graph.The next  lines each consist of three space-separated integers g\_from ,g\_to  and g\_weight , where g\_from  and g\_to   denote the two nodes between which the undirected edge exists and g\_weight , denotes the weight of that edge.

4 6

1 2 5

1 3 3

4 1 6

2 4 7

3 2 4

3 4 5

**OUTPUT:**

12

**Test Case 1:** Assume an input graph with more than one edge having a same weight, in such case which edge should be selected as the next visiting vertex by applying Kruskal’s algorithm.

**Test Case 2:** Construct a Maximum Spanning tree for the input graph by applying Kruskal’s algorithm.

**Aim:**

Create a C program to find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm

**Algorithm**:

Step 1 : Create a graph where all edges are connected.

Step 2 : Sort all the edges from low weight to high

Step 3: Take the edge with the lowest weight and add it to the spanning tree

Step 4 : If adding the edge created the cycle then reject that edge

Step 4 : Keep adding the edges until we reach all vertices

**Program:**

  #include <stdio.h>

    #include <stdlib.h>

     int i, j, k, a, b, u, v, n, ne = 1;

    int min, mincost = 0, cost[9][9], parent[9];

    int find(int);

    int uni(int, int);

    void main() {

      printf("\n\tImplementation of Kruskal's Algorithm\n");

      printf("\nEnter the no. of vertices:");

      scanf("%d", & n);

      printf("\nEnter the cost adjacency matrix:\n");

      for (i = 1; i <= n; i++) {

        for (j = 1; j <= n; j++) {

          scanf("%d", & cost[i][j]);

          if (cost[i][j] == 0)

            cost[i][j] = 999;

        }

      }

      printf("The edges of Minimum Cost Spanning Tree are\n");

      while (ne < n) {

        for (i = 1, min = 999; i <= n; i++) {

          for (j = 1; j <= n; j++) {

            if (cost[i][j] < min) {

              min = cost[i][j];

              a = u = i;

              b = v = j;

            }

          }

        }

        u = find(u);

        v = find(v);

        if (uni(u, v)) {

          printf("%d edge (%d,%d) =%d\n", ne++, a, b, min);

          mincost += min;

        }

        cost[a][b] = cost[b][a] = 999;

      }

      printf("\n\tMinimum cost = %d\n", mincost);

          }

    int find(int i) {

      while (parent[i])

        i = parent[i];

      return i;

    }

    int uni(int i, int j) {

      if (i != j) {

        parent[j] = i;

        return 1;

      }

      return 0;

}

**OUTPUT:**

Enter the no. of vertices:3

Enter the cost adjacency matrix:

9

8

7

6

5

4

3

2

3

The edges of Minimum Cost Spanning Tree are

1 edge (3,2) =2

2 edge (3,1) =3

Minimum cost = 5

**TestCase 1:** Assume an input graph with more than one edge having a same weight, in such case which edge should be selected as the next visiting vertex by applying Kruskal’s algorithm.

**Algorithm**:

Step 1 : Create a graph where all edges are connected.

Step 2 : Sort all the edges from low weight to high

Step 3: Take the edge with the lowest weight and add it to the spanning tree

Step 4 : If adding the edge created the cycle then reject that edge

Step 4 : Keep adding the edges until we reach all vertices

**Program:**

#include <stdio.h>

#define MAX 30

typedef struct edge {

int u, v, w;

} edge;

typedef struct edge\_list {

edge data[MAX];

int n;

} edge\_list;

edge\_list elist;

int Graph[MAX][MAX], n;

edge\_list spanlist;

void kruskalAlgo();

int find(int belongs[], int vertexno);

void applyUnion(int belongs[], int c1, int c2);

void sort();

void print();

// Applying Krushkal Algo

void kruskalAlgo() {

int belongs[MAX], i, j, cno1, cno2;

elist.n = 0;

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

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

if (Graph[i][j] != 0) {

elist.data[elist.n].u = i;

elist.data[elist.n].v = j;

elist.data[elist.n].w = Graph[i][j];

elist.n++;

}

}

sort();

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

belongs[i] = i;

spanlist.n = 0;

for (i = 0; i < elist.n; i++) {

cno1 = find(belongs, elist.data[i].u);

cno2 = find(belongs, elist.data[i].v);

if (cno1 != cno2) {

spanlist.data[spanlist.n] = elist.data[i];

spanlist.n = spanlist.n + 1;

applyUnion(belongs, cno1, cno2);

}

}

}

int find(int belongs[], int vertexno) {

return (belongs[vertexno]);

}

void applyUnion(int belongs[], int c1, int c2) {

int i;

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

if (belongs[i] == c2)

belongs[i] = c1;

}

// Sorting algo

void sort() {

int i, j;

edge temp;

for (i = 1; i < elist.n; i++)

for (j = 0; j < elist.n - 1; j++)

if (elist.data[j].w > elist.data[j + 1].w) {

temp = elist.data[j];

elist.data[j] = elist.data[j + 1];

elist.data[j + 1] = temp;

}

}

// Printing the result

void print() {

int i, cost = 0;

for (i = 0; i < spanlist.n; i++) {

printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);

cost = cost + spanlist.data[i].w;

}

printf("\nSpanning tree cost: %d", cost);

}

int main() {

int i, j, total\_cost;

n = 6;

Graph[0][0] = 0;

Graph[0][1] = 4;

Graph[0][2] = 4;

Graph[0][3] = 0;

Graph[0][4] = 0;

Graph[0][5] = 0;

Graph[0][6] = 0;

Graph[1][0] = 4;

Graph[1][1] = 0;

Graph[1][2] = 2;

Graph[1][3] = 0;

Graph[1][4] = 0;

Graph[1][5] = 0;

Graph[1][6] = 0;

Graph[2][0] = 4;

Graph[2][1] = 2;

Graph[2][2] = 0;

Graph[2][3] = 3;

Graph[2][4] = 4;

Graph[2][5] = 0;

Graph[2][6] = 0;

Graph[3][0] = 0;

Graph[3][1] = 0;

Graph[3][2] = 3;

Graph[3][3] = 0;

Graph[3][4] = 3;

Graph[3][5] = 0;

Graph[3][6] = 0;

Graph[4][0] = 0;

Graph[4][1] = 0;

Graph[4][2] = 4;

Graph[4][3] = 3;

Graph[4][4] = 0;

Graph[4][5] = 0;

Graph[4][6] = 0;

Graph[5][0] = 0;

Graph[5][1] = 0;

Graph[5][2] = 2;

Graph[5][3] = 0;

Graph[5][4] = 3;

Graph[5][5] = 0;

Graph[5][6] = 0;

kruskalAlgo();

print();

}

**Output:**

**Solution 1:**

2 - 1 : 2

5 - 2 : 2

3 - 2 : 3

4 - 3 : 3

1 - 0 : 4

Spanning tree cost: 14

**Test Case 2:** Construct a Maximum Spanning tree for the input graph by applying Kruskal’s algorithm.

Algorithm:

**Algorithm**:

Step 1 : Create a graph where all edges are connected.

Step 2 : Sort all the edges from high weight to low

Step 3: Take the edge with the highestt weight and add it to the spanning tree

Step 4 : If adding the edge created the cycle then reject that edge

Step 4 : Keep adding the edges until we reach all vertices

Program:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_EDGES 25000

#define MAX\_NODES 1000

typedef struct {

int u, v, w;

} Edge;

int n, m;

Edge edges[MAX\_EDGES];

int parent[MAX\_NODES];

int cmp(const void \*a, const void \*b) {

Edge \*x = (Edge \*)a, \*y = (Edge \*)b;

return y->w - x->w;

}

int find(int x) {

if (parent[x] == x)

return x;

return parent[x] = find(parent[x]);

}

void kruskal() {

int i, u, v, cost = 0;

qsort(edges, m, sizeof(Edge), cmp);

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

parent[i] = i;

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

u = find(edges[i].u), v = find(edges[i].v);

if (u != v) {

parent[u] = v;

cost += edges[i].w;

}

}

printf("The maximum spanning tree has a cost of %d.\n", cost);

}

int main() {

int i;

printf("Enter the number of nodes and edges: ");

scanf("%d %d", &n, &m);

printf("Enter the edges (u v w):\n");

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

scanf("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].w);

kruskal();

return 0;

}

OUTPUT:

Enter the number of nodes and edges: 3 3

Enter the edges (u v w):

1 2 5

2 3 8

1 3 7

The maximum spanning tree has a cost of 15.