**Experiment No. 2: Greedy method strategy**

**Date:**

**Aim**: Write a C program to implement the following program using Greedy method

strategy

1. Prim’s algorithm.
2. Kruskal’s algorithm.
3. Single source shortest path algorithm.

**THEORY:**

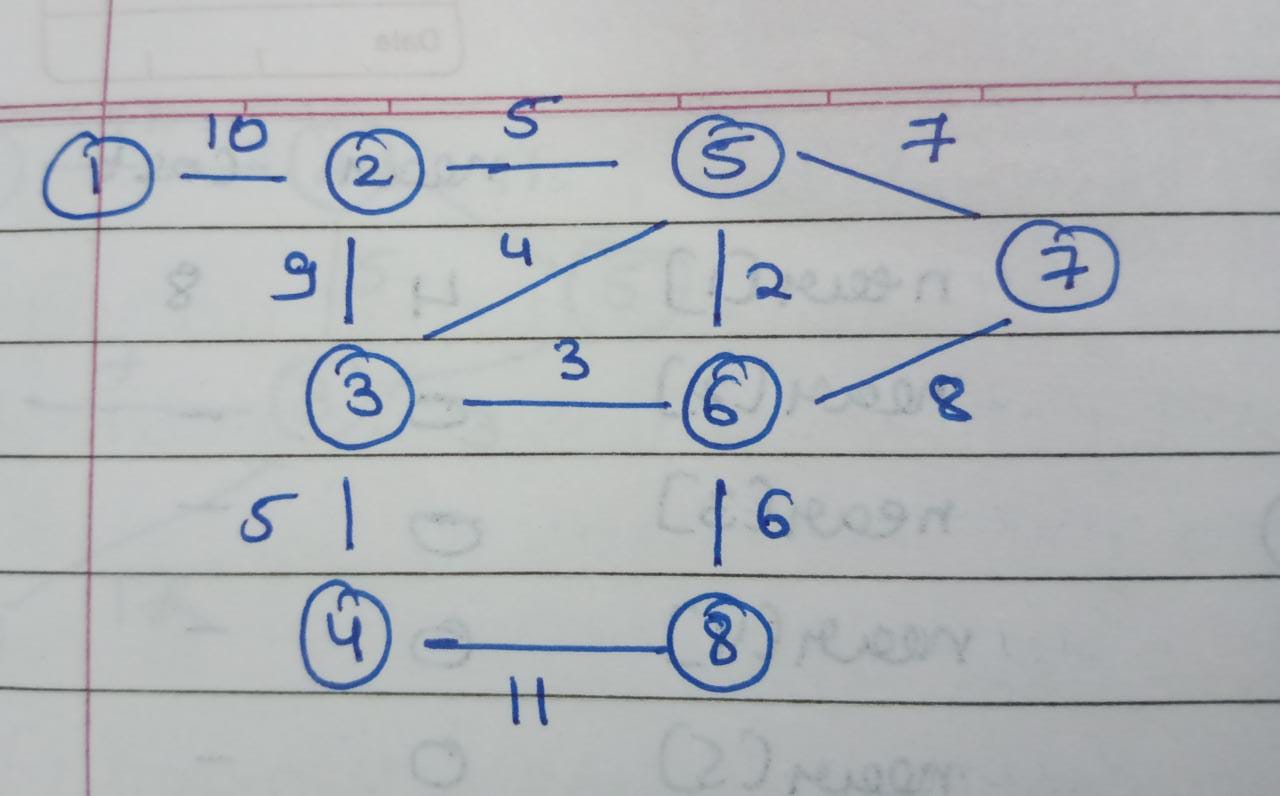
* The greedy algorithm is a problem-solving approach that chooses the best available option at each step of the solution process.
* It's based on the principle of making locally optimal choices with the hope of finding a global optimum solution.
* The algorithm starts with an empty solution set and iteratively builds it up by selecting the best available option at each step.
* The selection of the best option is based on a certain criterion, which can be the highest or lowest value, the maximum or minimum weight, or any other relevant metric.
* The algorithm does not revisit the choices made in previous steps, which may result in a suboptimal solution.
* The greedy algorithm is useful for solving optimization problems, such as the Knapsack problem, minimum spanning tree, and shortest path problem.
* It's often fast and efficient, but may not always guarantee the optimal solution, particularly in complex problems.
* The correctness of the greedy algorithm depends on the problem structure and the choice of the criterion for selecting the best option.
* Sometimes, a greedy approach can be combined with other algorithms to improve the quality of the solution

**a) Prim’s algorithm**

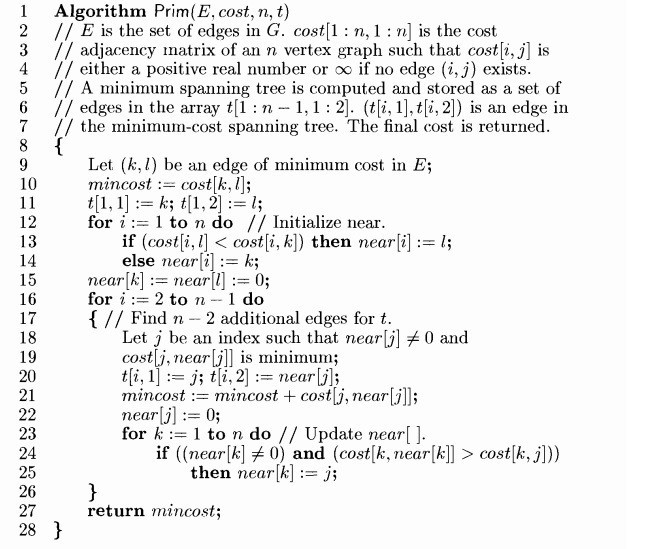
**Date:**

**Problem Statement:**

Write a c program to implement a minimum cost spanning tree using prims algorithm on following graph:



# **Algorithm**



**Performance analysis:**

1. Time complexity:

Using and adjacency matrix to represent the graph: O(V^2) where V is the number of vertices.

1. Space complexity:

Using an adjacency matrix: O(V^2)

# **Code:**

#include <stdio.h>

#include <limits.h>

#define v 8

int g[v][v]={

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

{10,0,9,0,5,0,0,0},

{0,9,0,5,4,3,0,0},

{0,0,5,0,0,0,0,11},

{0,5,4,0,0,2,7,0},

{0,0,3,0,2,0,8,6},

{0,0,0,0,7,8,0,0},

{0,0,0,11,0,6,0,0},

};

int t[v][2];

int near[v];

void setlimit(){

for(int i=0;i<v;i++){

for(int j=0;j<v;j++){

if(g[i][j]==0){

g[i][j]=INT\_MAX;

}

}

}

}

void printcost(){

for(int i=0;i<v;i++){

for(int j=0;j<v;j++){

printf("%d ",g[i][j]);

}

printf("\n");

}

}

void minedge(int \*k,int \*l){

setlimit();

int min=g[0][0];

int i,j;

for(i=0;i<=v/2;i++){

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

if(g[i][j]<min){

\*k=i;\*l=j;

min=g[i][j];

}

}

}

}

void printnear(){

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

printf("%d ",near[i]+1);

printf("\n");

}

void prims(){

int k,l;

int mincost=0;

minedge(&k,&l);

t[0][0]=k;

t[0][1]=l;

for(int i=0;i<v;i++){

if(g[i][k]<g[i][l])

near[i]=k;

else

near[i]=l;

}

near[k]=-1;near[l]=-1;

mincost+=g[k][l];

for(int i=1;i<v-1;i++){

printnear();

int minid=0;

int min=INT\_MAX;

for(int j=0;j<v;j++){

if(near[j]>-1 && min>g[j][near[j]]){

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

min=g[j][near[j]];

t[i][0]=j;

t[i][1]=near[j];

minid=j;

}

}

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

mincost+=min;

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

near[minid]=-1;

for(int p=0;p<v;p++){

if(near[p]>-1 && g[p][near[p]]>g[p][minid])

near[p]=minid;

}

}

printf("mincost= %d",mincost);

}

int main(){

prims();

}

**Output:**

PS C:\Users\P JEEVESH NAIDU> cd "c:\P Jeevesh Naidu\college\second year\lV sem\madf codes\" ; if ($?) { gcc prims.c -o prims } ; if ($?) { .\prims }

6 5 6 6 0 0 5 6

1

2

2

mincost= 5

6 5 0 3 0 0 5 6

1

1

mincost= 10

2 0 0 3 0 0 5 6

0

3

3

mincost= 15

2 0 0 0 0 0 5 6

0

6

7

7

mincost= 21

2 0 0 0 0 0 5 0

0

6

6

mincost= 28

2 0 0 0 0 0 0 0

0

0

mincost= 38

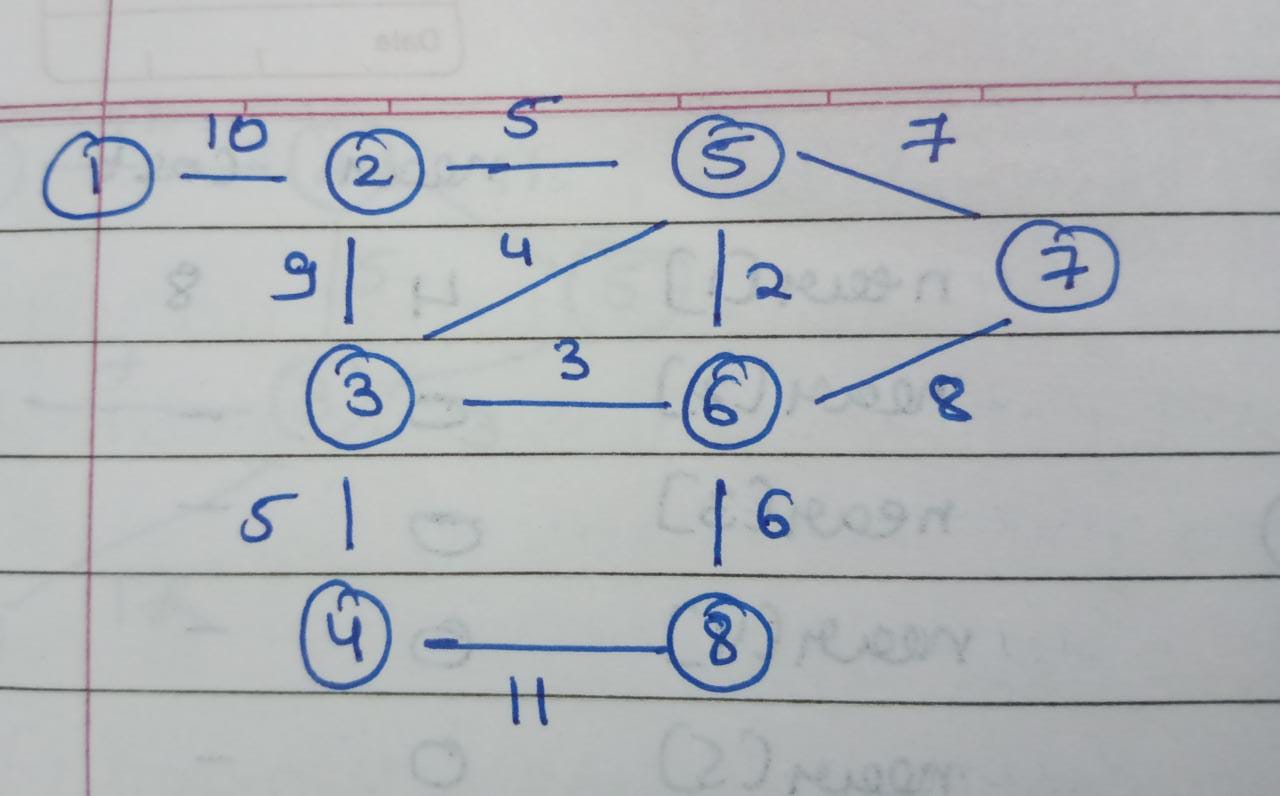
mincost= 38

**b) Kruskals algorithm**

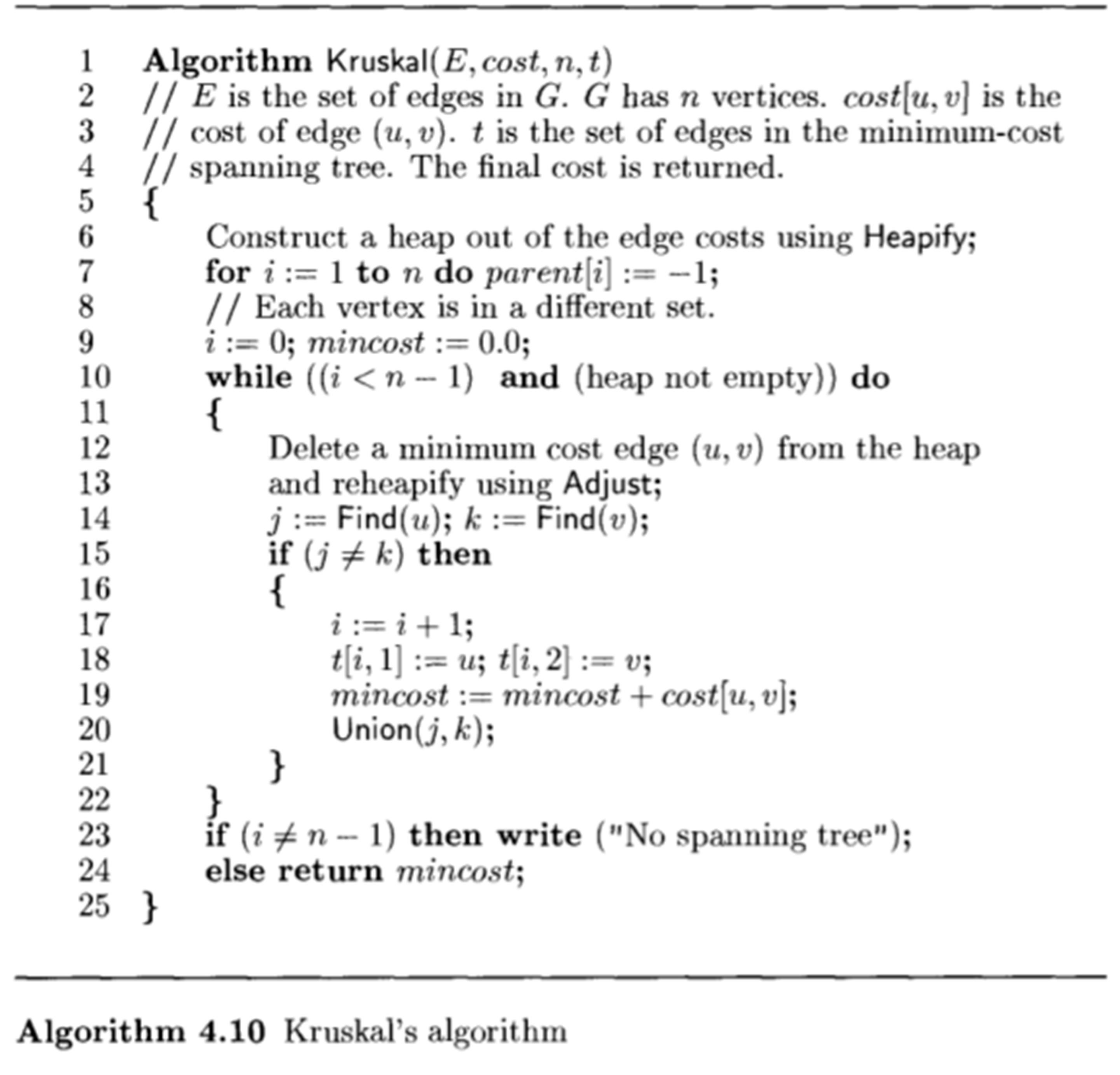
**Date:**

**Problem Statement:**

Write a c program to implement a minimum cost spanning tree using kruskals algorithm on following graph:



# **Algorithm**



Performance analysis:

1. Time complexity:

Sorting the edges takes O(E log E) time, where E is the number of edges in the graph.

Performing the union-find operations takes O(E α(V)) time, where α is the inverse Ackermann function, which grows very slowly and is considered constant for practical purposes.

Thus, the total time complexity of Kruskal's algorithm is O(E log E).

1. Space complexity:

Kruskal's algorithm requires space to store the edges and the disjoint sets used in the union-find operations. The space complexity is O(E + V).

# **Code:**

#include <stdio.h>

int adj[100][100], cost[100][100];

struct EDGE

{

int origin, destin, edgecost;

};

void init\_adj(int n)

{

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

for (int j = 0; j < n; j++)

adj[i][j] = 0;

}

void init\_cost(int n)

{

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

for (int j = 0; j <= n; j++)

cost[i][j] = \_\_INT32\_MAX\_\_;

}

int accept\_graph(FILE \*fp, int n)

// int accept\_graph(int n)

{

int max\_edges = n \* (n - 1) / 2;

int origin, destin, edgecost;

int e = 0;

printf("Enter the edges of the graph and their respective cost.\n");

for (int i = 0; i < max\_edges; i++)

{

// printf("Enter the edge,( 0 0 randval) to quit :");

// scanf("%d %d %d", &origin, &destin, &edgecost);

fscanf(fp, "%d %d %d", &origin, &destin, &edgecost);

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

break;

if (origin > n || destin > n || origin <= 0 || destin <= 0)

{

printf("Invalid edge.\n");

i--;

}

else

{

adj[origin - 1][destin - 1] = 1;

cost[origin - 1][destin - 1] = edgecost;

adj[destin - 1][origin - 1] = 1;

cost[destin - 1][origin - 1] = edgecost;

e++;

}

}

return e;

}

void display\_arr(int a[], int n)

{

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

printf("%d ", a[i] + 1);

printf("\n");

}

void swap(struct EDGE \*a, struct EDGE \*b)

{

struct EDGE temp = \*a;

\*a = \*b;

\*b = temp;

}

void heapify(struct EDGE adj[], int e, int i)

{

int smallest = i;

int l = 2 \* i + 1, r = 2 \* i + 2;

if (l < e && adj[l].edgecost <= adj[smallest].edgecost)

smallest = l;

if (r < e && adj[r].edgecost <= adj[smallest].edgecost)

smallest = r;

if (smallest != i)

{

swap(&adj[i], &adj[smallest]);

heapify(adj, e, smallest);

}

}

int Find(int parent[], int u)

{

while (parent[u] != -1)

u = parent[u];

return u;

}

void Union(int parent[], int u, int v)

{

parent[u] = v;

}

int kruskal(int n, int e, int t[][2])

{

struct EDGE heap[e];

int in = 0;

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

{

for (int j = 0; j < n; j++)

{

if (j > i && cost[i][j] != \_\_INT32\_MAX\_\_)

{

heap[in].origin = i;

heap[in].destin = j;

heap[in++].edgecost = cost[i][j];

}

}

}

for (int i = e / 2 - 1; i >= 0; i--)

heapify(heap, e, i);

int parent[n];

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

parent[i] = -1;

int i = 1, mincost = 0;

int u, v, j, k;

while (i < n)

{

swap(&heap[0], &heap[e - 1]);

struct EDGE minedge = heap[e - 1];

e--;

heapify(heap, e, 0);

u = minedge.origin;

v = minedge.destin;

j = Find(parent, u);

k = Find(parent, v);

if (j != k)

{

t[i][0] = u, t[i][1] = v, mincost += cost[u][v];

Union(parent, j, k);

printf("(%d, %d) ", u + 1, v + 1);

for (int l = 0; l < n; l++)

{

int val = 1;

if (parent[l] == -1)

val = 0;

printf("%d ", parent[l] + val);

}

printf("Cost: %d", mincost);

printf("\n");

i++;

}

}

if (i != n)

return -1;

else

return mincost;

}

int main()

{

int e, n;

FILE \*fp;

fp = fopen("graph1.txt", "r+");

printf("Enter the number of vertices of the graph.\n");

scanf("%d", &n);

int t[n][2];

init\_adj(n);

init\_cost(n);

e = accept\_graph(fp, n);

fclose(fp);

// e = accept\_graph(n);

int mincost = kruskal(n, e, t);

if (mincost != -1)

printf("Mincost: %d\n", mincost);

else

printf("No MST.\n");

return 0;

}

**Output:**

PS C:\Users\P JEEVESH NAIDU> cd "c:\P Jeevesh Naidu\college\second year\lV sem\madf codes\" ; if ($?) { gcc kruskalsmatrix.c -o kruskalsmatrix } ; if ($?) { .\kruskalsmatrix }

Enter the number of vertices of the graph.

8

Enter the edges of the graph and their respective cost.

(5, 6) -1 -1 -1 -1 6 -1 -1 -1 Cost: 2

(3, 6) -1 -1 6 -1 6 -1 -1 -1 Cost: 5

(2, 5) -1 6 6 -1 6 -1 -1 -1 Cost: 10

(3, 4) -1 6 6 -1 6 4 -1 -1 Cost: 15

(6, 8) -1 6 6 8 6 4 -1 -1 Cost: 21

(5, 7) -1 6 6 8 6 4 -1 7 Cost: 28

(1, 2) 7 6 6 8 6 4 -1 7 Cost: 38

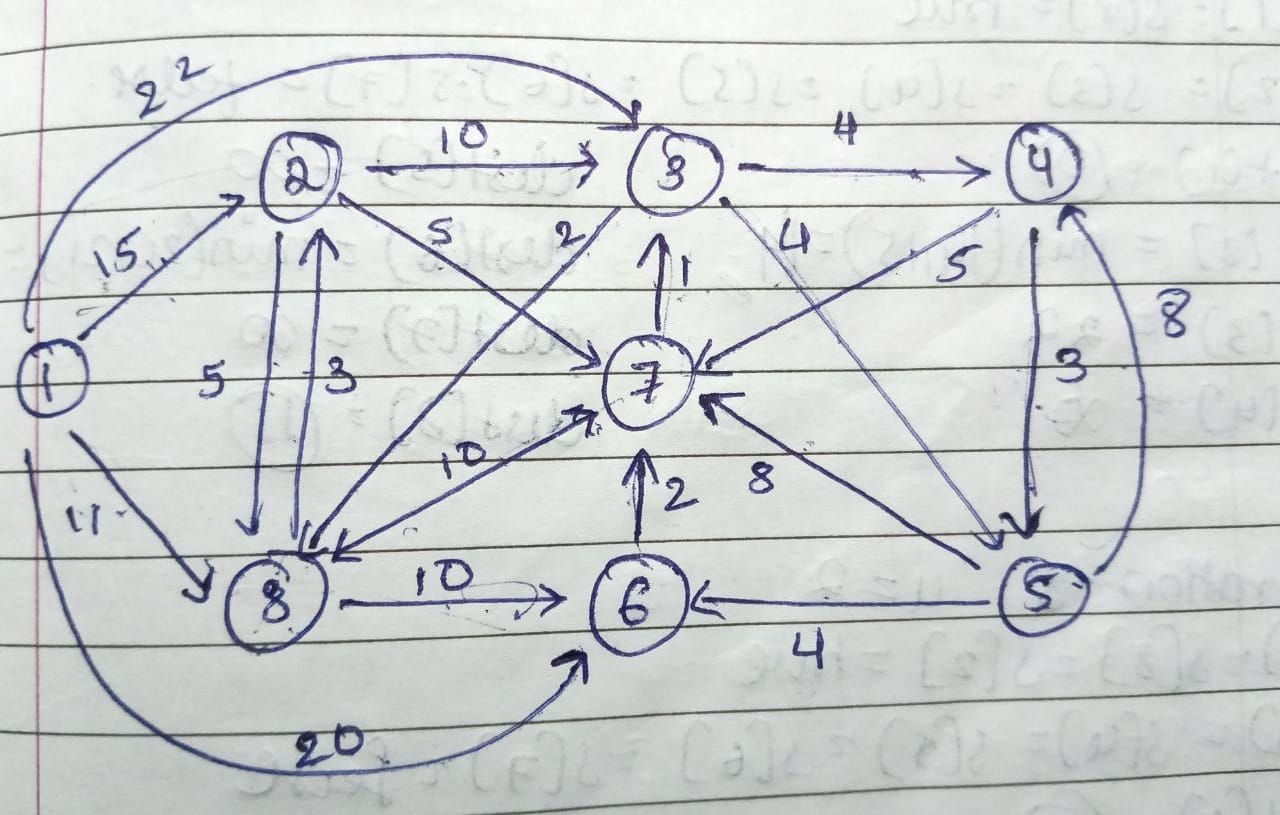
Mincost: 38

**c) Single source shortest path algorithm**

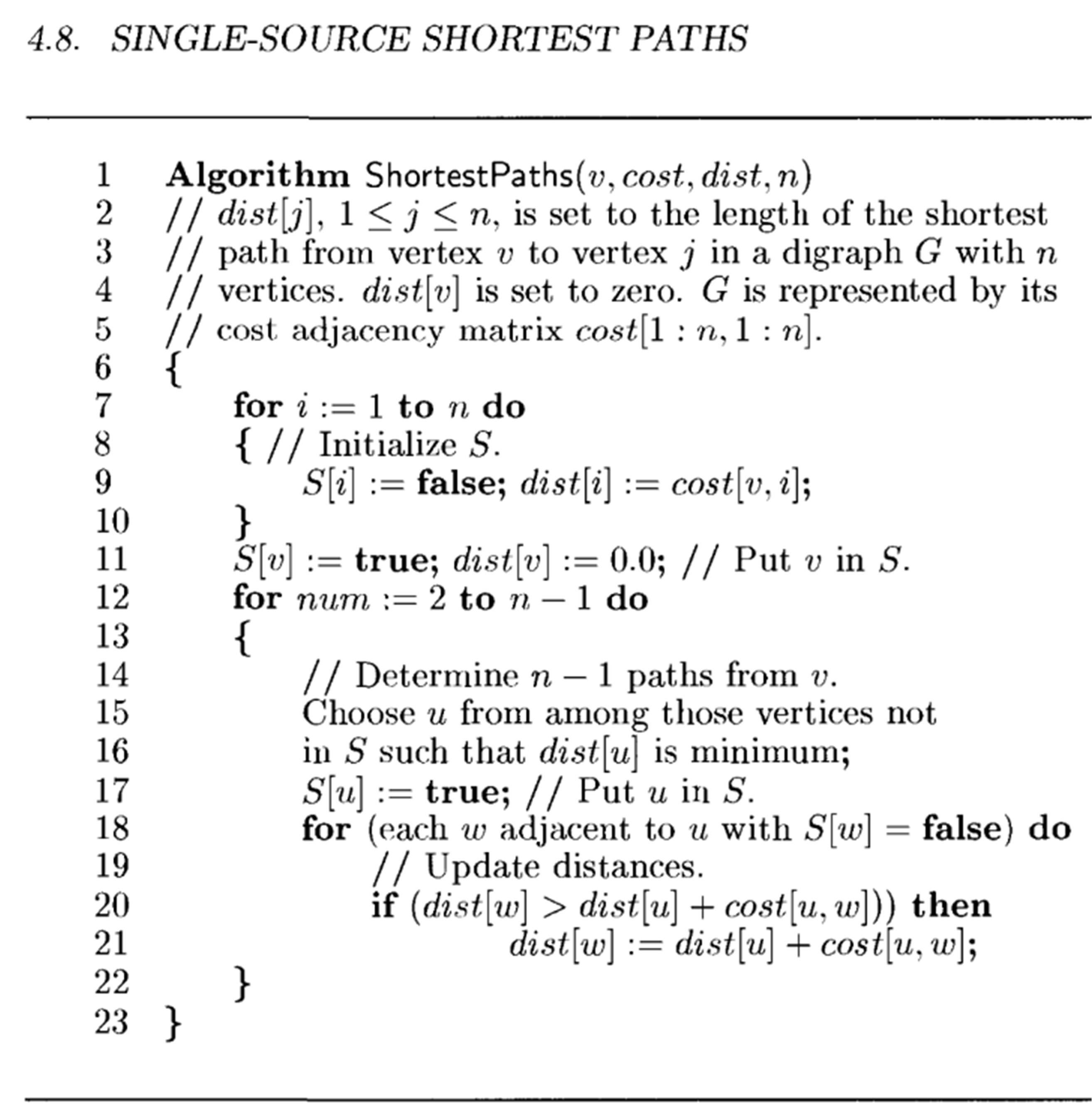
**Date:**

**Problem Statement:**

Write a c program to find shortest path on following graph:



**Algorithm**



Performance analysis:

1. Time complexity:

Using a binary heap to implement the priority queue: O(E log V) time complexity, where E is the number of edges and V is the number of vertices in the graph.

Using a Fibonacci heap to implement the priority queue: O(E + V log V) time complexity.

1. Space complexity:

Space complexity: O(V + E), where V is the number of vertices and E is the number of edges in the graph.

# **Code:**

#include <stdio.h>

#include <limits.h>

#define N 10

typedef enum

{

false,

true

};

void fillinf(int cost[][N], int n)

{

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

for (int j = 1; j <= n; j++)

cost[i][j] = INT\_MAX;

}

void fillzeroes(int e[][N], int n)

{

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

for (int j = 1; j <= n; j++)

e[i][j] = 0;

}

void getmat(int cost[][N], int n, int e)

{

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

{

int s, d, w;

scanf("%d %d %d", &s, &d, &w);

if (s >= 1 && s <= n && d >= 1 && d <= n)

cost[s][d] = w;

}

}

void showmat(int a[][N], int n)

{

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

{

for (int j = 1; j <= n; j++)

a[i][j] == INT\_MAX ? printf("x ") : printf("%d ", a[i][j]);

printf("\n");

}

printf("\n");

}

void shortestpath(int cost[][N], int dist[N], int n, int v)

{

int s[N];

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

s[i] = false, dist[i] = cost[v][i];

s[v] = true, dist[v] = 0;

printf("Iteration 1\n");

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

printf("dist[%d] = %d\n", i, dist[i]);

printf("\n");

for (int i = 2; i <= n - 1; i++)

{

int u, min = INT\_MAX;

for (int k = 1; k <= n; k++)

if (s[k] == false && dist[k] < min)

min = dist[k], u = k;

printf("u: %d\n", u);

s[u] = true;

for (int j = 1; j <= n; j++)

{

if (cost[u][j] != INT\_MAX && s[j] == false)

if (dist[j] > dist[u] + cost[u][j])

dist[j] = dist[u] + cost[u][j];

}

printf("Iteration %d\n", i);

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

printf("dist[%d] = %d\n", i, dist[i]);

printf("\n");

}

}

int main()

{

int cost[N][N], dist[N], n, v, e;

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

scanf("%d", &n);

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

scanf("%d", &e);

fillinf(cost, n);

getmat(cost, n, e);

scanf("%d", &v);

shortestpath(cost, dist, n, v);

printf("\n");

printf("Starting vertex: %d\n", v);

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

printf("dist[%d] = %d\n", i, dist[i]);

return 0;

}

Output:

Enter the number of nodes: 8

Enter the number of edges: 20

1 2 15

1 3 22

1 8 11

1 6 20

2 8 5

2 3 10

2 7 5

3 8 2

3 5 4

3 4 4

4 5 3

4 7 5

5 4 8

5 6 4

5 7 8

6 7 2

7 3 1

7 8 10

8 6 10

8 2 3

1

1

Iteration 1

dist[1] = 0

dist[2] = 15

dist[3] = 22

dist[4] = 2147483647

dist[5] = 2147483647

dist[6] = 20

dist[7] = 2147483647

dist[8] = 11

u: 8

Iteration 2

dist[1] = 0

dist[2] = 14

dist[3] = 22

dist[4] = 2147483647

dist[5] = 2147483647

dist[6] = 20

dist[7] = 2147483647

dist[8] = 11

u: 2

Iteration 3

dist[1] = 0

dist[2] = 14

dist[3] = 22

dist[4] = 2147483647

dist[5] = 2147483647

dist[6] = 20

dist[7] = 19

dist[8] = 11

u: 7

Iteration 4

dist[1] = 0

dist[2] = 14

dist[3] = 20

dist[4] = 2147483647

dist[5] = 2147483647

dist[6] = 20

dist[7] = 19

dist[8] = 11

u: 3

Iteration 5

dist[1] = 0

dist[2] = 14

dist[3] = 20

dist[4] = 24

dist[5] = 24

dist[6] = 20

dist[7] = 19

dist[8] = 11

u: 6

Iteration 6

dist[1] = 0

dist[2] = 14

dist[3] = 20

dist[4] = 24

dist[5] = 24

dist[6] = 20

dist[7] = 19

dist[8] = 11

u: 4

Iteration 7

dist[1] = 0

dist[2] = 14

dist[3] = 20

dist[4] = 24

dist[5] = 24

dist[6] = 20

dist[7] = 19

dist[8] = 11

Starting vertex: 1

dist[1] = 0

dist[2] = 14

dist[3] = 20

dist[4] = 24

dist[5] = 24

dist[6] = 20

dist[7] = 19

dist[8] = 11

**CONCLUSION:**

Greedy method strategy was studied. The programs for (a) prims algorithm, (b) Kruskals algorithm, (c) Single source shortest path algorithm were studied and implemented successfully.