Bangalore Institute of Technology

# Department of Computer Science and Engineering DESIGN AND ANALYSIS OF ALGORITHMS LAB (BCSL404)

1. **Design and implement C Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.**

**Aim :** To apply Kruskal's Algorithm for computing the minimum spanning tree is directly based on the generic MST algorithm. It builds the MST in forest.

**Definition:** Kruskal’s algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This mean it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a minimum spanning forest .It is an example of a greedy algorithm**.**

Efficiency:With an efficient sorting algorithm, the time efficiency of Kruskal’s algorithm will be in O(E log E)

**Algorithm**

Start with an empty set A, and select at every stage the shortest edge that has not been chosen or rejected, regardless of where this edge is situated in graph.

* + Initially, each vertex is in its own tree in forest.
  + Then, algorithm consider each edge in turn, order by increasing weight.
  + If an edge (*u*, *v*) connects two different trees, then (*u*, *v*) is added to the set of edges of the MST, and two trees connected by an edge (*u*, *v*) are merged into a single tree.
  + On the other hand, if an edge (*u*, *v*) connects two vertices in the same tree, then edge (*u*, *v*) is discarded.

Kruskals algorithm can be implemented using **disjoint set** data structure or **priority queue** data structure.

|  |  |
| --- | --- |
| I Kruskal's algorithm implemented with disjoint-sets data structure. |  |
| **MST\_KRUSKAL (G, w)**   1. A ← {} // A will ultimately contains the edges of the MST 2. for each vertex v in V[G] 3. do Make\_Set (*v*) 4. Sort edge of E by nondecreasing weights w 5. for each edge (*u*, *v*) in E 6. do if FIND\_SET (*u*) ≠ FIND\_SET (*v*) 7. then A = AU{(*u*, *v*)} 8. UNION (*u*, *v*) 9. Return A   #include<stdio.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\t Implementation 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)

{

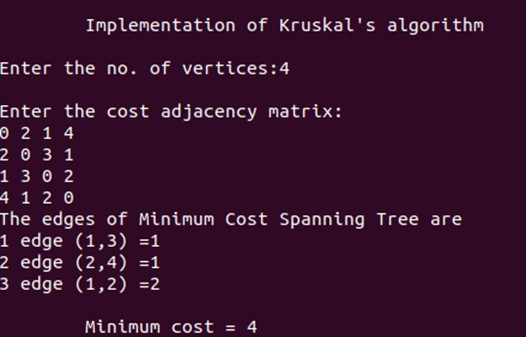
if(i!=j)

{

parent[j]=i; return 1;

}

return 0;}



# Input/Output:

Enter the n value:5

Enter the cost adjacency matrix:

0 10 15 9 999

10 0 999 17 15

15 999 0 20 999

9 17 20 0 18

999 15 999 18 0

The edges of minimum cost spanning tree are:

1 edge (1 ,4)

2 edge (1 ,2)

3 edge (1 ,3)

4 edge (2,5)

Minimum cost =49

**Program 2** Design and implement C Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

**Aim:**To find minimum spanning tree of a given graph using prim’s algorithm

**Definition:** Prim’s is an algorithm that finds a minimum spanning tree for a connected weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all edges in the tree is minimized. Prim’s algorithm is an example of a greedy algorithm.

**Algorithm**

MST\_PRIM (G, *w*, *v*)

// Prim’s algorithm for constructing a minimum spanning tree

// Input : A weighted connected graph G=(V,E)

// Output : The set of edges composing a minimum spanning tree of G

1. Q ← V[G]
2. for each *u* in Q do
3. key [*u*] ← ∞
4. key [*r*] ← 0
5. π[*r*] ← NIl
6. while queue is not empty do
7. *u* ← EXTRACT\_MIN (Q)
8. for each *v* in Adj[*u*] do
9. if v is in Q and *w*(*u*, *v*) < key [*v*]
10. then π[*v*] ← *w*(*u*, *v*)
11. key [*v*] ← *w*(*u*, *v*)

Program:

#include<stdio.h>

int visited[10]={0}, cost[10][10], min, mincost=0; int i,j,ne=1, a, b, u, v;

int main()

{

int num;

printf("\n\t\t\tPrim's Algorithm"); printf("\n\nEnter the number of nodes= "); scanf("%d", &num);

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

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

{

for(j=1; j<=num; j++)

{

scanf("%d", &cost[i][j]); if(cost[i][j]==0)

cost[i][j]=999;

}

}

visited[1]=1; while(ne < num)

{

for(i=1,min=999;i<=num;i++) for(j=1;j<=num;j++)

if(cost[i][j]< min) if(visited[i]!=0)

{

min=cost[i][j]; a=u=i;

b=v=j;

}

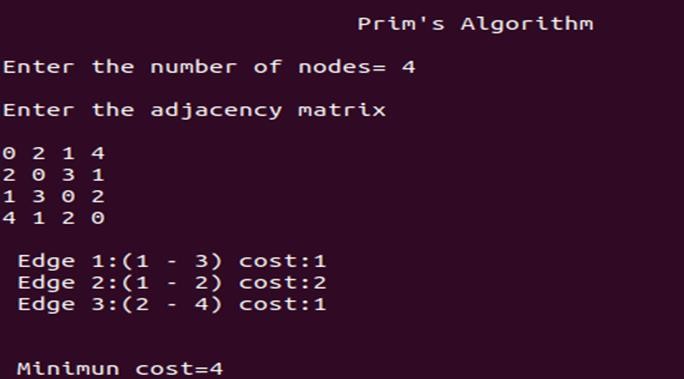
printf("\n Edge %d:(%d - %d) cost:%d",ne++,a,b,min); mincost=mincost+min;

visited[b]=1; cost[a][b]=cost[b][a]=999;

}

printf("\n\n\n Minimun cost=%d",mincost); return 0; }

OUTPUT:



1)

**2)Input/output:** Enter n value:3 Enter the graph data: 0 10 1

10 0 6

1 6 0

Enter the souce node:1 1 -> 3 cost=1

3 -> 2 cost=7 minimum Cost=7