



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Experiment No. 6
Prim's Algorithm
Date of Performance:
Date of Submission:



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Title: Prim's Algorithm.

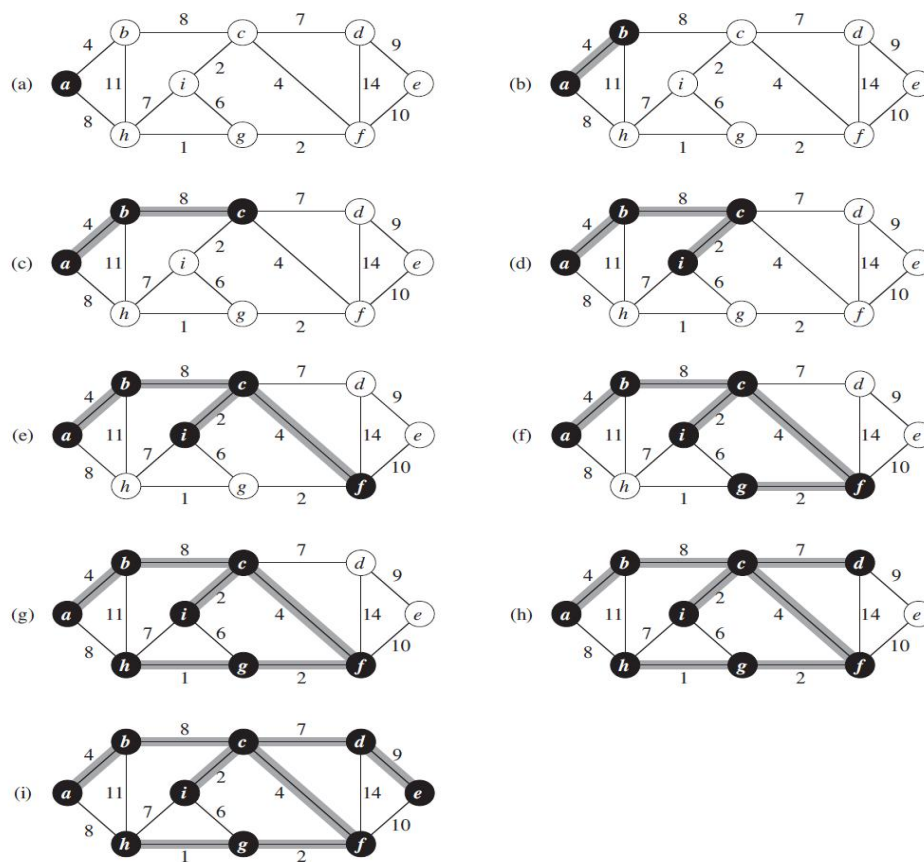
Aim: To study and implement Prim's Minimum Cost Spanning Tree Algorithm.

Objective: To introduce Greedy based algorithms

Theory:

Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a 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 the edges in the tree is minimized. The algorithm operates by building this tree one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.

Example:





Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Algorithm and Complexity:

```
1  Algorithm Prim( $E, cost, n, t$ )
2  //  $E$  is the set of edges in  $G$ .  $cost[1 : n, 1 : n]$  is the cost
3  // adjacency matrix of an  $n$  vertex graph such that  $cost[i, j]$  is
4  // either a positive real number or  $\infty$  if no edge  $(i, j)$  exists.
5  // A minimum spanning tree is computed and stored as a set of
6  // edges in the array  $t[1 : n - 1, 1 : 2]$ .  $(t[i, 1], t[i, 2])$  is an edge in
7  // the minimum-cost spanning tree. The final cost is returned.
8  {
9      Let  $(k, l)$  be an edge of minimum cost in  $E$ ;
10      $mincost := cost[k, l]$ ;
11      $t[1, 1] := k$ ;  $t[1, 2] := l$ ;
12     for  $i := 1$  to  $n$  do // Initialize near.
13         if  $(cost[i, l] < cost[i, k])$  then  $near[i] := l$ ;
14         else  $near[i] := k$ ;
15      $near[k] := near[l] := 0$ ;
16     for  $i := 2$  to  $n - 1$  do
17     { // Find  $n - 2$  additional edges for  $t$ .
18         Let  $j$  be an index such that  $near[j] \neq 0$  and
19          $cost[j, near[j]]$  is minimum;
20          $t[i, 1] := j$ ;  $t[i, 2] := near[j]$ ;
21          $mincost := mincost + cost[j, near[j]]$ ;
22          $near[j] := 0$ ;
23         for  $k := 1$  to  $n$  do // Update  $near[ ]$ .
24             if  $((near[k] \neq 0) \text{ and } (cost[k, near[k]] > cost[k, j]))$ 
25                 then  $near[k] := j$ ;
26     }
27     return  $mincost$ ;
28 }
```

Time Complexity is $O(n^2)$, Where, n = number of vertices **Theory:**

Implementation:

```
#include<stdio.h>
```

```
#include<conio.h>
```

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

```
int visited[10]={0},min,mincost=0,cost[10][10];
```

```
void main()
```



```
{  
  
    clrscr();  
  
    printf("Enter the number of nodes:");  
  
    scanf("%d",&n);  
  
    printf("Enter the adjacency matrix:\n");  
  
    for(i=1;i<=n;i++)  
    for(j=1;j<=n;j++)  
    {  
        scanf("\t%d",&cost[i][j]);  
        if(cost[i][j]==0)  
            cost[i][j]=999;  
    }  
  
    visited[1]=1;  
  
    while(ne < n)  
    {  
        for(i=1,min=999;i<=n;i++)  
        for(j=1;j<=n;j++)  
        {  
            if(cost[i][j]< min)  
            if(visited[i]!=0)  
            {  
                min=cost[i][j];  
                a=u=i;  
                b=v=j;  
            }  
        }  
    }  
}
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

```
        if(visited[u]==0 || visited[v]==0)
        {
            printf("\nEdge %d:(%d %d) cost:%d",ne++,a,b,min);
            mincost+=min;
            visited[b]=1;
        }
        cost[a][b]=cost[b][a]=999;
    }

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

    getch();
}

#include<stdio.h>

#include<conio.h>

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

int visited[10]={0},min,mincost=0,cost[10][10];

void main()
{
    clrscr();

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

    scanf("%d",&n);

    printf("Enter the adjacency matrix:\n");

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



```
{  
  
    scanf("\t%d",&cost[i][j]);  
  
    if(cost[i][j]==0)  
        cost[i][j]=999;  
  
}  
  
visited[1]=1;  
  
while(ne < n)  
{  
    for(i=1,min=999;i<=n;i++)  
        for(j=1;j<=n;j++)  
            if(cost[i][j]< min)  
                if(visited[i]!=0)  
                {  
                    min=cost[i][j];  
                    a=u=i;  
                    b=v=j;  
                }  
            if(visited[u]==0 || visited[v]==0)  
            {  
                printf("\nEdge %d:(%d %d) cost:%d",ne++,a,b,min);  
                mincost+=min;  
                visited[b]=1;  
            }  
        cost[a][b]=cost[b][a]=999;
```



```
}  
  
printf("\n Minimun cost=%d",mincost);  
  
getch();  
  
}
```

Output:

```
Enter the number of nodes:3  
Enter the adjacency matrix:  
1  
2  
3  
4  
5  
6  
7  
8  
9  
  
Edge 1:(1 2) cost:2  
Edge 2:(1 3) cost:3  
Minimun cost=5_
```

Conclusion: Implementing Prim's algorithm has proven to be effective in generating minimum spanning trees, efficiently connecting all nodes in a graph while minimizing total edge weight. This experiment underscores the algorithm's practical applicability in optimizing network connectivity, demonstrating its importance in various real-world scenarios.