Department of Computer Science and Engineering (Data Science)

NAME: Alistair Saldanha **BATCH: K1**

SAPID: 60009200024

Experiment 3

(Greedy Algorithm)

Aim: Implementation of Prims's & Kruskal's method.

Prim's algorithm:

Theory:

Prim's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which

- form a tree that includes every vertex
- has the minimum sum of weights among all the trees that can be formed from the graph?

Algorithm:

Step 1:

- Randomly choose any vertex.
- The vertex connecting to the edge having least weight is usually selected.

Step 2:

- Find all the edges that connect the tree to new vertices.
- Find the least weight edge among those edges and include it in the existing tree.
- If including that edge creates a cycle, then reject that edge and look for the next least weight edge.

Step 3:

- Keep repeating step-02 until all the vertices are included and Minimum Spanning Tree (MST) is obtained.

Example:

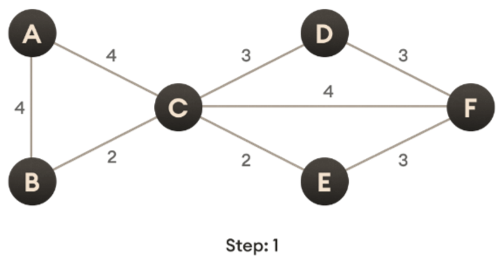


Figure 1. Start with a weighted graph

Figure 2.Choose a vertex

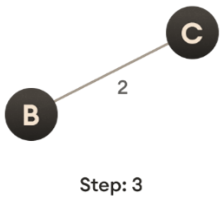


Figure 3.Choose the shortest edge from this vertex and add it

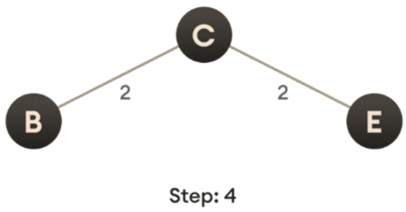


Figure 4.Choose the nearest vertex not yet in the solution

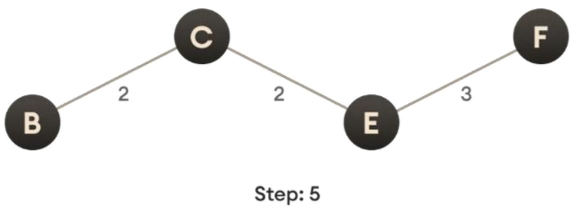


Figure 5.Choose the nearest edge not yet in the solution, if there are multiple choices, choose one at random

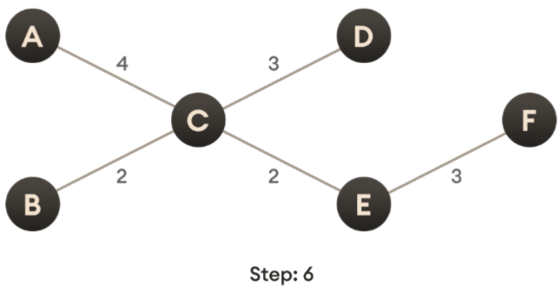


Figure 6.Repeat until you have a spanning tree

Complexity:

The time complexity of Prim's algorithm is O(E log V).

Kruskal's algorithm:

Theory:

Kruskal's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which

- form a tree that includes every vertex
- has the minimum sum of weights among all the trees that can be formed from the graph?

Algorithm:

Step 1:

- Sort all the edges from low weight to high weight.

Step 2:

- Take the edge with the lowest weight and use it to connect the vertices of graph.
- If adding an edge creates a cycle, then reject that edge and go for the next least weight edge.

Step 3:

- Keep adding edges until all the vertices are connected and a Minimum Spanning Tree (MST) is obtained.

Example:

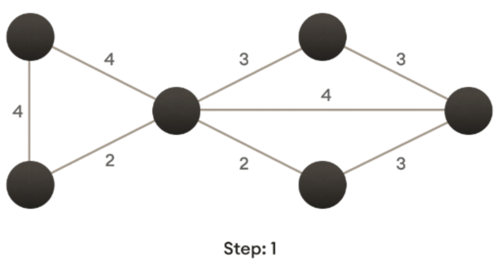


Figure 7. Start with a weighted graph

Department of Computer Science and Engineering (Data Science)

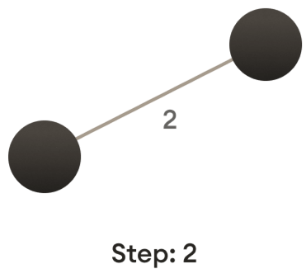


Figure 8. Choose the edge with the least weight, if there are more than 1, choose anyone

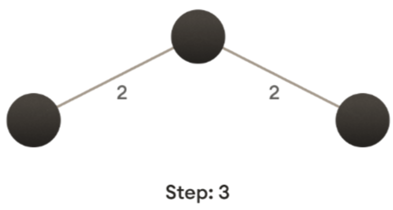


Figure 9. Choose the next shortest edge and add it

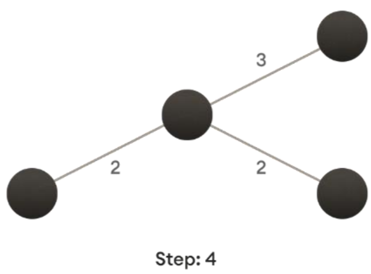


Figure 10. Choose the next shortest edge that doesn't create a cycle and add it

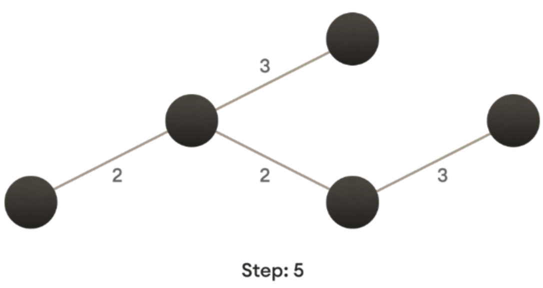


Figure 11. Choose the next shortest edge that doesn't create a cycle and add it

Department of Computer Science and Engineering (Data Science)

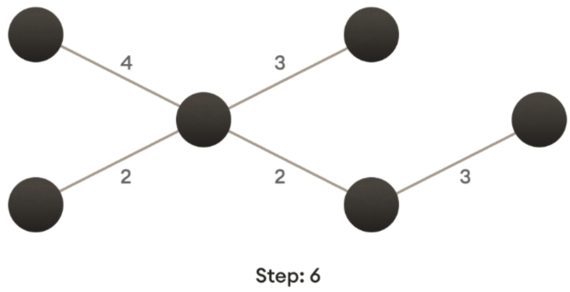


Figure 12.Repeat until you have a spanning tree

Complexity:

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

Kruskal

Code:

```
#include<stdio.h>
#define MAX 15
// Structure storing information of each edge
typedef struct edge {
	int u, v, weight;
} edge;
// Structure for storing all the edges of the graph.
typedef struct Edge_list {
	edge data[MAX];
	int n;
} Edge_list;
int G[MAX][MAX], n;
Edge_list E1; // to store the data of graph
Edge_list span_tree; // to store the data of Minimum Spanning Tree
// To check whether the given vertex forms a cycle
int find(int included[],int vertexno)
```

```
{
        return(included[vertexno]);
}
void union1(int included[],int c1,int c2)
{
        int i;
        for(i=0; i<n; i++)
        {
                if(included[i]==c2)
                        included[i]=c1;
        }
}
// Sort Function to sort the weights in ascending order
void sort() {
        int i,j;
        edge temp;
        for(i=1; i<E1.n; i++)
        {
                for(j=0; j<E1.n-1; j++)
                {
                        if(E1.data[j].weight>E1.data[j+1].weight)
                        {
                                temp=E1.data[j];
                                E1.data[j]=E1.data[j+1];
                                E1.data[j+1]=temp;
                        }
                }
        }
```

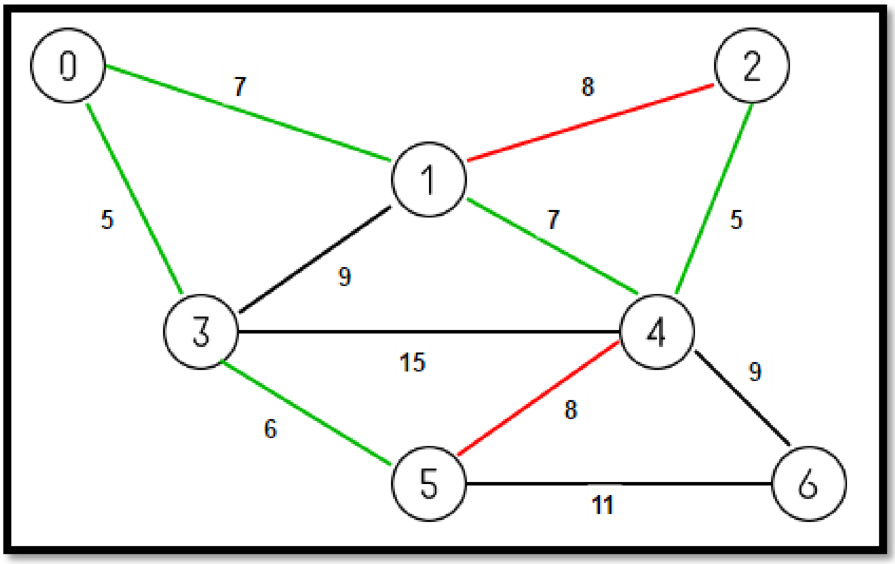
```
}
void print() {
	int i,cost=0;
	printf("\n--------------Spanning Tree---------------\n");
	for(i=0; i<span_tree.n; i++) {
		printf("\n%d-->%d
Cost=%d",span_tree.data[i].u,span_tree.data[i].v,span_tree.data[i].weight);
		cost += span_tree.data[i].weight;
	}
	printf("\n\nCost of the Minimum spanning tree = %d",cost);
}

// Kruskal's Algorithm
void kruskal() {
	int i,j,cno1,cno2,included[MAX];
	E1.n=0;
	for(i=0; i<n; i++)
	{
		for(j=0; j<i; j++)
		{
		// Weight is not 0 then include in edgelist E1 else discard
			if(G[i][j]!=0)
			{
				E1.data[E1.n].u=i;
				E1.data[E1.n].v=j;
				E1.data[E1.n].weight=G[i][j];
				E1.n++;
			}
		}
```

Department of Computer Science and Engineering (Data Science)

```
	}
	sort();
	for(i=0; i<n; i++)
		included[i]=i;
//	Initialise the spanning list to 0
	span_tree.n=0;
	for(i=0; i<E1.n; i++)
	{
		cno1=find(included,E1.data[i].u);
		cno2=find(included,E1.data[i].v);
//	If find() returns the same value that means they have the same parent (vertex when added forms a cycle)
		if(cno1!=cno2)
		{
			span_tree.data[span_tree.n]=E1.data[i];
			span_tree.n++;
			union1(included,cno1,cno2);
		}
	}
}
void main() {
	int i,j;
	printf("\nEnter number of vertices:");
	scanf("%d",&n);
	printf("\nEnter the adjacency matrix:\n");
	for(i=0; i<n; i++)
	{
		for(j=0; j<n; j++)
			scanf("%d",&G[i][j]);
```

```
    }
    kruskal();
    print();
}
```

Example:



Output:

```
Enter number of vertices:7

Enter the adjacency matrix:
0 7 0 5 0 0 0
7 0 8 9 7 0 0
0 8 0 0 5 0 0
5 9 0 0 15 6 0
0 7 5 0 0 8 9
0 0 0 6 8 0 11
0 0 0 0 9 11 0

--------------Spanning Tree---------------
u       v       Weights
3       0         5
4       2         5
5       3         6
1       0         7
4       1         7
6       4         9

Cost of the Minimum spanning tree = 39
--------------------------------
```

Prims

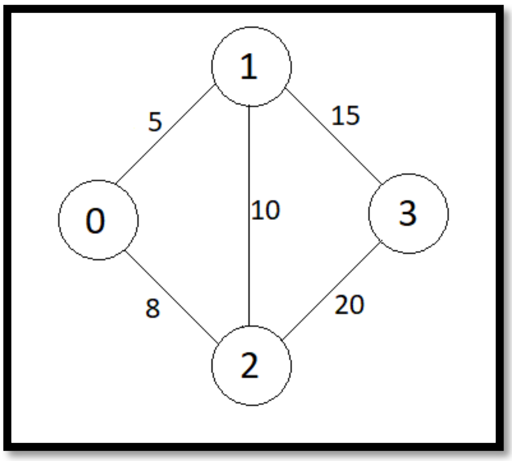
Code:

```
#include<stdio.h>
int cost[10][10], visited[10]={0}, i, j, n, no_of_edges=1, min, a, b, min_cost=0;
//input graph
void read_graph()
{
	printf("Enter 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("%d",&cost[i][j]);
         // if cost = 0 then initialize it by maximum value
         if(cost[i][j]==0)
           cost[i][j]=1000;
      }
   }
}
int prims()
{
	// logic for finding minimum cost of spanning tree
   visited[1]=1; // visited first node
   printf("----------Spanning Tree----------");
   while(no_of_edges < n)
```

```
 {
   min = 1000;
   // Looping and finding the minimum cost
   for(i=1;i<=n;i++)
   {
     for(j=1;j<=n;j++)
     {
//          if cost is minimum and vertex is not visited
       if(cost[i][j]<min)
       {
         if(visited[i] != 0)
         {
           min = cost[i][j];
           a = i;
           b = j;
         }
       }
     }
   }
   // if destination vertex is not visited, include the edge in min. spanning tree
   if(visited[b]==0)
   {
     printf("\n%d-->%d  cost = %d",a,b,min);
     min_cost += min;
     no_of_edges++;
   }
   visited[b]=1;
   // initialize with maximum value.
```

```
        cost[a][b] = cost[b][a] = 1000;
    }
    return min_cost;
}
int main()
{
        read_graph();
        min_cost = prims();
    printf("\n\nCost of the Minimum spanning tree = %d",min_cost);
    return 0;
}
```

Example:



Output:

```
Enter number of nodes 4
Enter the adjacency matrix:
0 5 8 0
5 0 10 15
8 10 0 20
0 15 20 0

1-->2  cost = 5
1-->3  cost = 8
2-->4  cost = 15
Cost of the Minimum spanning tree = 28
--------------------------------
```