PART A

(PART A : TO BE REFFERED BY STUDENTS)

Experiment No.05

A.1 Aim:

Implementation of Greedy method algorithm - Prim's algorithm

A.2 Prerequisite:

C programming

A.3 Outcome:

After successful completion of this experiment students will be,

Apply concepts of Trees and Graphs to a given problem.

Apply the concept of Greedy and Dynamic Programming approach to solve problems.

A.4 Theory:

Greedy Algorithm Approach:

A Greedy algorithm is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So the problems where choosing locally optimal also leads to a global solution are best fit for Greedy.

Prims Algorithm:

Prim's Algorithm is used to find the minimum spanning tree from a graph. Prim's algorithm is a greedy algorithm that is used to form a minimum spanning tree for a connected weighted undirected graph

Working Principle:

- Prim's Algorithm works from one vertex and grows the rest of the tree by adding one vertex at a time, by adding the associated edges
- Prim's algorithm starts with the single node and explore all the adjacent nodes with all the connecting edges at every step
- The edges with the minimal weights causing no cycles in the graph got selected

Algorithm:

Step 1: Create a set A that keeps track of vertices already included in MST

Step 2: Assign a key value to all vertices in the graph

Assign key value 0 for the start vertex and ∞ for other vertices

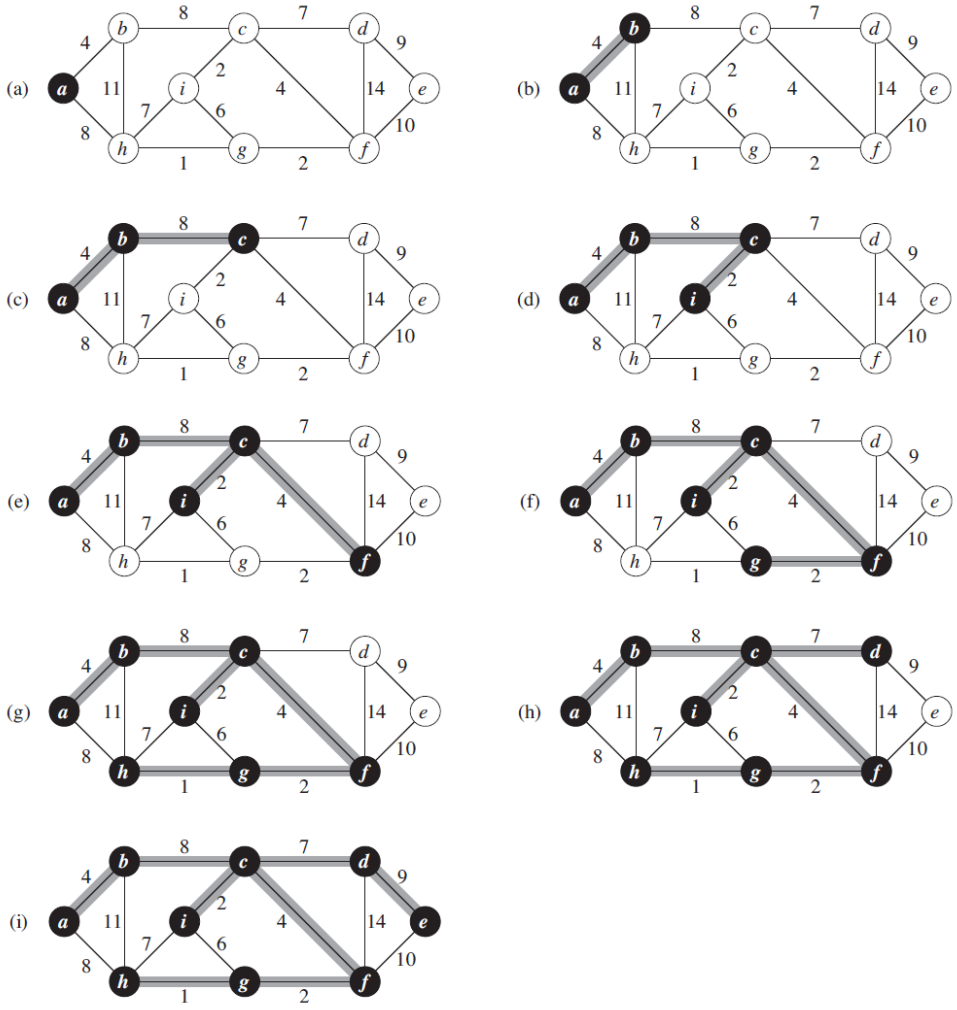
Step 3: While A doesn't include all vertices in the graph

Pick a vertex v which is not there in A and has minimum key value

Include v to A

Update key value of all adjacent vertices of v

Example:



PART B

(PART B : TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

Roll. No.71	Name: Omkar Gujja
Class: 2	Batch: MTRX
Date of Experiment: 13/03/2021	Date of Submission: 20/03/2021
Grade:	

B.1 Software Code written by student:

```
#include<stdio.h>

#include<stdlib.h>


#define MAX 10

#define TEMP 0

#define PERM 1

#define infinity 9999

#define NIL -1


struct edge

{

	int u;

	int v;

};


int n;

int adj[MAX][MAX];
```

```
int predecessor[MAX];
int status[MAX];
int length[MAX];

void create_graph();
void maketree(int r, struct edge tree[MAX]);
int min_temp();

int main()
{
    int wt_tree = 0;
    int i, root;
    struct edge tree[MAX];

    create_graph();

    printf("\nEnter root vertex : ");
    scanf("%d",&root);

    maketree(root, tree);
```

```
    printf("\nEdges to be included in spanning tree are : \n");

    for(i=1; i<=n-1; i++)
    {
        printf("%d-> ",tree[i].u);
        printf("%d\n",tree[i].v);
        wt_tree += adj[tree[i].u][tree[i].v];
    }
    printf("\nWeight of spanning tree is : %d\n", wt_tree);

    return 0;

}/*End of main()*/

void maketree(int r, struct edge tree[MAX])
{
    int current,i;
    int count = 0;/*number of vertices in the tree*/
```

```
/*Initialize all vertices*/
for(i=0; i<n; i++)
{
      predecessor[i] = NIL;
      length[i] = infinity;
      status[i] = TEMP;
}

/*Make length of root vertex 0*/
length[r] = 0;

while(1)
{
      /*Search for temporary vertex with minimum length
      and  make it current vertex*/
      current = min_temp();

      if(current == NIL)
      {
            if(count == n-1) /*No temporary vertex left*/
```

```
                return;
            else /*Temporary vertices left with length infinity*/
            {
                printf("\nGraph is not connected, No spanning tree
possible\n");
                exit(1);
            }
        }

        /*Make the current vertex permanent*/
    status[current] = PERM;

        /*Insert the edge ( predecessor[current], current) into the
tree,
        except when the current vertex is root*/
        if(current != r)
        {
      count++;
          tree[count].u = predecessor[current];
          tree[count].v = current;
        }
```

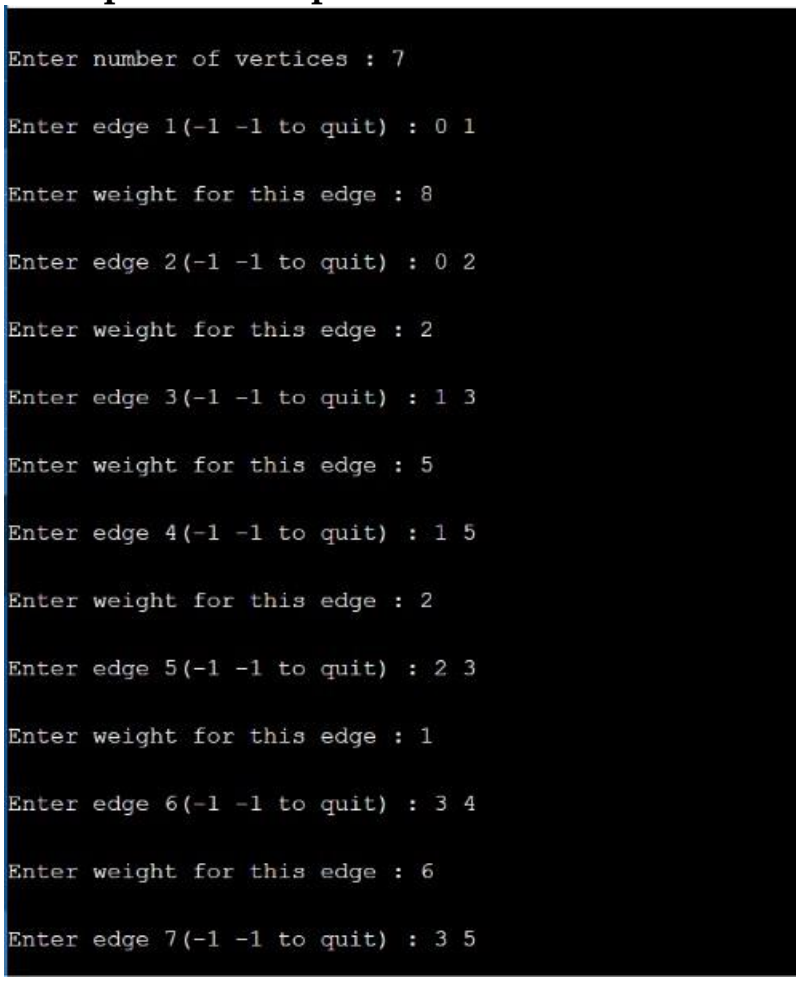
```
            for(i=0; i<n; i++)
                  if(adj[current][i] > 0 && status[i] == TEMP)
                        if(adj[current][i] < length[i])
                        {
                              predecessor[i] = current;
                              length[i] = adj[current][i];
                        }
      }

}/*End of make_tree( )*/

/*Returns the temporary vertex with minimum value of length
  Returns NIL if no temporary vertex left or
  all temporary vertices left have pathLength infinity*/
int min_temp()
{
      int i;
      int min = infinity;
      int k = -1;
```

```
    for(i=0; i<n; i++)
    {
        if(status[i] == TEMP && length[i] < min)
        {
            min = length[i];
            k = i;
        }
    }

    return k;
}/*End of min_temp()*/

void create_graph()
{
    int i,max_edges,origin,destin,wt;

    printf("\nEnter number of vertices : ");
    scanf("%d",&n);
    max_edges = n*(n-1)/2;
```

```
for(i=1; i<=max_edges; i++)
{
    printf("\nEnter edge %d(-1 -1 to quit) : ",i);
    scanf("%d %d",&origin,&destin);
    if((origin == -1) && (destin == -1))
        break;
    printf("\nEnter weight for this edge : ");
    scanf("%d",&wt);
    if( origin >= n || destin >= n || origin < 0 || destin < 0)
    {
        printf("\nInvalid edge!\n");
        i--;
    }
    else
    {
        adj[origin][destin] = wt;
        adj[destin][origin] = wt;
    }
}
```

}

B.2 Input and Output:



```
Enter weight for this edge : 9

Enter edge 8(-1 -1 to quit) : 4 5

Enter weight for this edge : 10

Enter edge 9(-1 -1 to quit) : 4 6

Enter weight for this edge : 3

Enter edge 10(-1 -1 to quit) : 5 6

Enter weight for this edge : 4

Enter edge 11(-1 -1 to quit) : -1 -1

Enter root vertex : 0

Edges to be included in spanning tree are :
0-> 2
2-> 3
3-> 1
1-> 5
5-> 6
6-> 4

Weight of spanning tree is : 17
```

B.3 Observations and learning:

We have observed how Prim's Algorithm works

B.4 Conclusion:

We have understood the Prims's Algorithm for minimum spanning.

B.5 Question of Curiosity

Q.1 What is minimum spanning tree ?
Soln.

The cost of the spanning tree is the sum of the weights of all the edges in the tree. There can be many spanning trees. Minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees.

Q.2 What is the difference between Prims and Kruskal's algorithm

PRIMS ALGORITHM VERSUS KRUSHAL ALGORITHM

PRIMS ALGORITHM	KRUSHAL ALGORITHM
A greedy algorithm that finds a minimum spanning tree for a weighted undirected graph.	A minimum spanning tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest
Generates the minimum spanning tree starting from the root vertex	Generates the minimum spanning tree starting from the least weighted edge
Selects the root vertex	Selects the shortest edge
Selects the shortest edge connected to the root vertex	Selects the next shortest edge