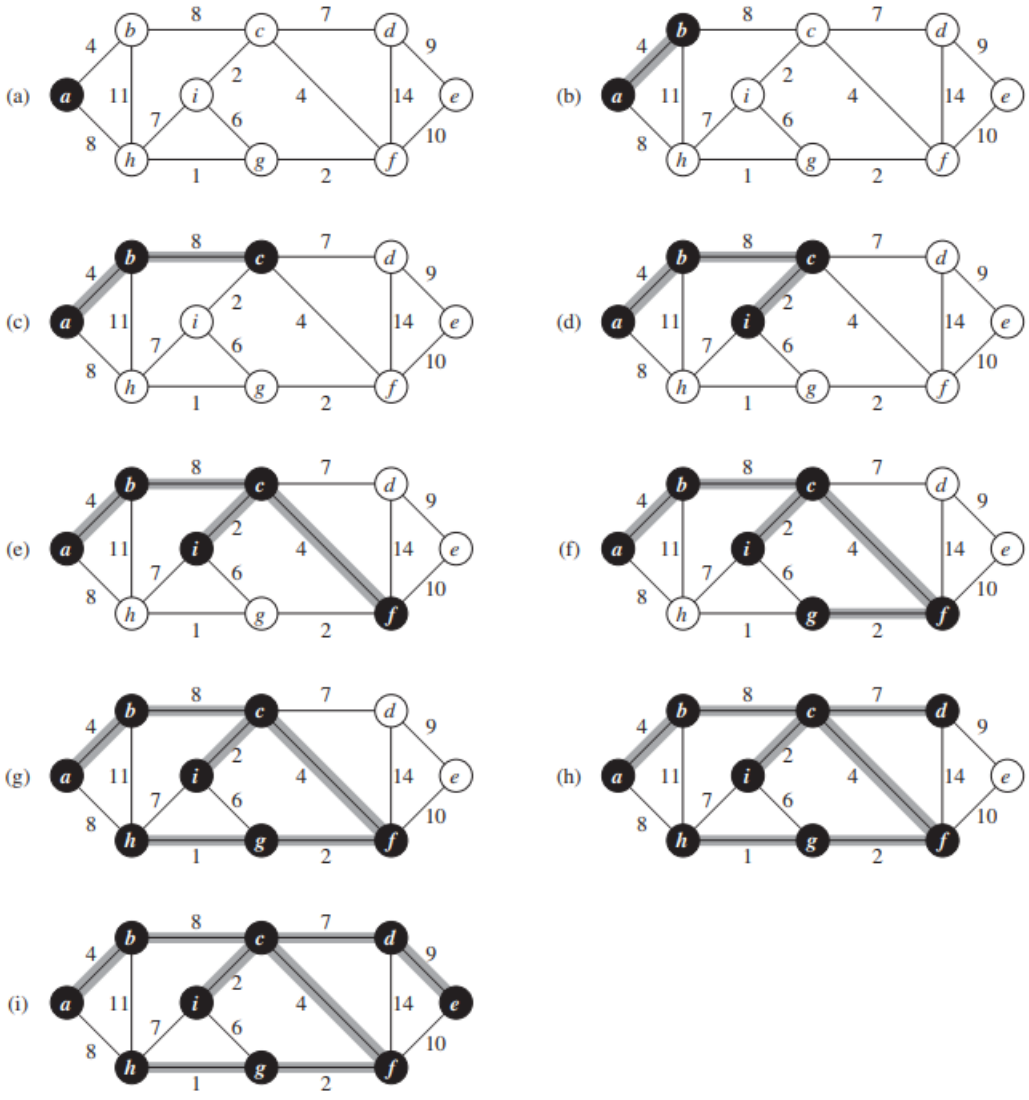
Name	**Advait Ravi Sapkal**
UID No.	**2021700055**
Experiment No.	**5**

AIM:	Prim's Algorithm to find the minimum spanning tree.
Program 1	
PROBLEM STATEMENT:	Given a graph G<V, E>, find a minimum spanning tree from the graph using Prim's algorithm.
ALGORITHM:	**Prim's Algorithm** **Invariant:** In every iteration, the minimum is extracted and put in set A forming the MST. The following loop invariant is maintained: 1. $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$. 2. The vertices already placed into the minimum spanning tree are those in $V - Q$. 3. For all vertices $v \in Q$, if $v.\pi \neq$ NIL, then $v.key < \infty$ and $v.key$ is the weight of a light edge $(v, v.\pi)$ connecting v to some vertex already placed into the minimum spanning tree. **Algorithm:** MST-PRIM(G, w, r) 1 **for** each $u \in G.V$ 2 $u.key = \infty$ 3 $u.\pi =$ NIL 4 $r.key = 0$ 5 $Q = G.V$ 6 **while** $Q \neq \emptyset$ 7 $u =$ EXTRACT-MIN(Q) 8 **for** each $v \in G.Adj[u]$ 9 **if** $v \in Q$ and $w(u, v) < v.key$ 10 $v.\pi = u$ 11 $v.key = w(u, v)$

Trace:



Code:

```c
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>

#define V 9


int minKey(int key[], bool mstSet[])
{
    int min = INT_MAX, min_index;
```

```
    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

int printMST(int parent[], int graph[V][V])
{
    printf("E   \tW(E)\n");
    for (int i = 1; i < V; i++)
        printf("%c - %c \t%d \n", parent[i]+65, i+65,
               graph[i][parent[i]]);
}

void prim_mst(int graph[V][V])
{
    int parent[V];

    int key[V];

    bool mstSet[V];


    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    key[0] = 0;

    parent[0] = -1;

    for (int count = 0; count < V - 1; count++) {


        int u = minKey(key, mstSet);

        mstSet[u] = true;

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

            if (graph[u][v] && mstSet[v] == false
                && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
```

```
    }
    printMST(parent, graph);
}

int main()
{
    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0},
                        { 4, 0, 8, 0, 0, 0, 0, 11, 0},
                        { 0, 8, 0, 7, 0, 4, 0, 0, 2},
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0},
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0},
                        { 0, 0, 4, 14, 10, 2, 0, 0, 0},
                        { 0, 0, 0, 0, 0, 2, 0, 1, 6},
                        { 8, 11, 0, 0, 0, 0, 1, 0, 7},
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0}};
    prim_mst(graph);
    return 0;
}
```

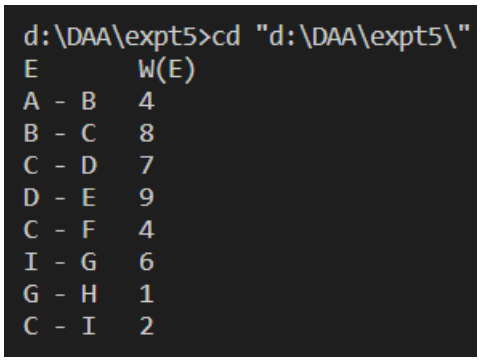
Analysis:

If a **linear method** O(V) of extracting min is used, the time complexity comes out as O(V2) as while loop executes V times, extracting min every time. Then we are decreasing the keys in a number proportional to E, so that complexity is E*C where C is a constant as array access is constant. Hence overall time complexity is O(V2+E)

If a **binary min heap** is used as a priority queue to extract min, the extract min operation in the loop takes O(VlogV) time. As the decrease key operation takes O(logV), The time for that in the loop is O(ElogV) Hence overall time complexity is O(VlogV + ElogV). As edges will be greater than or equal to vertices, we can say it to be O(ElogV).

If a **fibonacci heap** is used, the extract min takes O(lgV) amortized time, giving the time complexity O(VlogV) for extract min in the loop. It takes O(1) amortized time to decrease key, making the complexity for that in the loop as O(E*C). The overall complexity comes out to be O(VlogV+E), asymptotically O(VlogV) which is a bit better than previous case.

If **adjacency matrix graph** is used, the space complexity is of course O(V2)
If **adjacency list graph** is used, the space complexity is O(V+E)

RESULT:	**Output:**
	 
	`d:\DAA\expt5>cd "d:\DAA\expt5\"`
	`E  W(E)`
	`A - B  4`
	`B - C  8`
	`C - D  7`
	`D - E  9`
	`C - F  4`
	`I - G  6`
	`G - H  1`
	`C - I  2`
	
	Conclusions:
	
	• The algorithm starts with an arbitrary vertex and adds the minimum weight edge to a growing tree at each step until all vertices are included.
	• The MST found by Prim's algorithm is always unique for a given graph, regardless of the starting vertex.
	• Prim's algorithm can be used to solve various problems, such as finding the minimum cost of connecting a set of islands or the shortest path in a network of cities with varying distances.
	• A key advantage of Prim's algorithm is that it does not require the input graph to be connected or have distinct edge weights, and it can handle negative edge weights if modified accordingly.
	• The downside of Prim's algorithm is that it may not be the most efficient algorithm for finding the MST in some cases, such as when the graph is dense or when parallelization is required.
	• Overall, Prim's algorithm is a reliable and widely used algorithm for finding the MST, and it has various applications in real-world scenarios.