

**Name : Mayuresh Bhagwan Nehe**

**PRN No: - 20190802076**

### **Lab 7**

**Aim-** In this lab, we would be solving problems based on graphs.

### **Explanation:**

**1.Kruskal's algorithm :** Kruskal's algorithm creates a minimum spanning tree from a weighted undirected graph by adding edges in ascending order of weights till all the vertices are contained in it. Kruskal's algorithm gets greedy as it chooses edges in increasing order of weights. The algorithm makes sure that the addition of new edges to the spanning tree does not create a cycle within it.

### **Algorithm:**

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are  $(V-1)$  edges in the spanning tree.

### **Pseudocode:**

KRUSKAL(G):

$A = \emptyset$

For each vertex  $v \in G.V$ :

    MAKE-SET( $v$ )

For each edge  $(u, v) \in G.E$  ordered by increasing order by weight( $u, v$ ):

    if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ ):

$A = A \cup \{(u, v)\}$

UNION( $u, v$ )

return  $A$

**2.Prim's Algorithm:** 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:**

1) Create a set `min_span_tree_set` that keeps track of vertices already included in MST.

2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE.

3) Assign key value as 0 for the first vertex so that it is picked first.

4) While `min_span_tree_set` doesn't include all vertices

a) Pick a vertex  $u$  which is not there in `min_span_tree_set` and has minimum key value.

b) Include  $u$  to `min_span_tree_set`.

c) Update key value of all adjacent vertices of  $u$ . To update the key values, iterate through all adjacent vertices. For every adjacent vertex  $v$ , if weight of edge  $u-v$  is less than the previous key value of  $v$ , update the key value as weight of  $u-v$ . The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

## **Pseudocode:**

$T = \emptyset;$

$U = \{ 1 \};$

while ( $U \neq V$ )

    let  $(u, v)$  be the lowest cost edge such that  $u \in U$  and  $v \in V - U;$

$T = T \cup \{(u, v)\}$

$U = U \cup \{v\}$

## **3.Dijkstra's Algorithm:**

Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph

It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph.

### **Algorithm:**

1) Create a set Set (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.

2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.

3) While Set doesn't include all vertices

    a) Pick a vertex  $u$  which is not there in Set and has minimum distance value.

    b) Include  $u$  to Set.

c) Update distance value of all adjacent vertices of  $u$ . To update the distance values, iterate through all adjacent vertices. For every adjacent vertex  $v$ , if sum of distance value of  $u$  (from source) and weight of edge  $u-v$ , is less than the distance value of  $v$ , then update the distance value of  $v$ .

### **Pseudocode:**

```
funcDijkstra(G, S)
  for each vertex V in G
    distance[V] <- infinite
    previous[V] <- NULL
  If V != S, add V to Priority Queue Q
  distance[S] <- 0
  while Q IS NOT EMPTY
    U <- Extract MIN from Q
    for each unvisited neighbour V of U
      tempDistance <- distance[U] + edge_weight(U, V)
      if tempDistance < distance[V]
        distance[V] <- tempDistance
        previous[V] <- U
  return distance[], previous[]
```

### **OUTPUT :**

### **Dijkstra's Algorithm :**

The image shows a PyCharm IDE window with the file `20190802076_DAA_Dijkstra's_algorithm.py` open. The code defines a graph with 9 vertices and 14 edges, and implements Dijkstra's algorithm to find the shortest path from vertex 'a' to vertex 't'.

```

graph LR
    0((0)) --- 1((1))
    0 --- 2((2))
    0 --- 3((3))
    0 --- 4((4))
    1 --- 2
    1 --- 3
    2 --- 3
    2 --- 4
    3 --- 4
    3 --- 5((5))
    4 --- 5
    4 --- 6((6))
    5 --- 6
    5 --- 7((7))
    6 --- 7
    6 --- 8((8))
    7 --- 8
    7 --- t((t))
    8 --- t

```

The output of the algorithm is displayed in the Run console:

```

Dijkstra's Algorithm Implementation:
Vertex Distance from 'a' :
0 0
1 4
2 12
3 19
4 21
5 11
6 9
7 8
8 14

```

Process finished with exit code 0

## Kruskal's Algorithm :

The image shows a PyCharm IDE window with the file `20190802076_DAA_Kruskal's_algorithm.py` open. The code defines a graph with 4 vertices and 5 edges, and implements Kruskal's algorithm to find the Minimum Spanning Tree.

```

graph LR
    0((0)) --- 1((1))
    0 --- 2((2))
    0 --- 3((3))
    1 --- 2
    2 --- 3

```

The output of the algorithm is displayed in the Run console:

```

Kruskal's Algorithm Implementation:
Edges in the Minimum Spanning Tree:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Spanning Tree 19

```

Process finished with exit code 0

## Prim's Algorithm :

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help MN - C:\Users\mayur\AppData\Roaming\JetBrains\PyCharmCE2020.2\scratches\20190802076_DAA_Prim's_algorithm.py
Scratches 20190802076_DAA_Prim's_algorithm.py 20190802076_DAA_Prim's_algorithm.py
Project MN C:\Users\mayur\PycharmProjects\MN
20190802076_DAA_L2.py
20190802076_DAA_LAB4_Q1.py
20190802076_DAA_LAB4_Q2.py
20190802076_DAA_LAB4_Q3.py
20190802076_LAB3_DAA.py
20190802076_LAB5_Q1.py
20190802076_LAB5_Q2.py
20190802076_LAB5_Q3.py
20190802076_LAB5_Q4.py
20190802076_LAB6.py
main.py
20190802076_DAA_Prim's_algorithm.py
66
67
68 t = Graph(5)
69 t.graph = [[0, 2, 0, 6, 0],
70           [2, 0, 3, 8, 5],
71           [0, 3, 0, 0, 7],
72           [6, 8, 0, 0, 9],
73           [0, 5, 7, 9, 0]]
74
75 t.prim_algorithm()
Run: 20190802076_DAA_Prim's_algorithm
"E:\python 8\python.exe" C:/Users/mayur/AppData/Roaming/JetBrains/PyCharmCE2020.2/scratches/20190802076_DAA_Prim's_algorithm.py
Prim's Algorithm Implementation:
Edge Weight of tree
0 - 1 2
1 - 2 3
0 - 3 6
1 - 4 5
Process finished with exit code 0
Activate Windows
Go to Settings to activate Windows.
69:28 CRLF UTF-8 4 spaces Python 3.8
03:46 PM
21-11-2020
```

**Conclusion-** Therefore, we can implement,

- 1) Kruskal's Algorithm
- 2) Prim's Algorithm and
- 3) Dijkstra's Algorithms