Experiment: 4

Write a program to implement Kruksal’s algorithm for a graph.

Student Name: Nitish Rai

UID: 23MCA20326

Branch: Computers Application

Section/Group: 4(A)

Semester: 2nd

Date of Performance: 23/02/2024

Subject Name: Design and Analysis of Algorithms

Subject Code: 23CAH-511

A. The task is to:

Implement Kruskal's algorithm to find the minimum spanning tree of a graph represented by an adjacency matrix. This involves sorting edges by weight, using Union-Find to avoid cycles, and selecting edges that connect different sets until a spanning tree is formed.

B. Steps of Experiment:

- Understand the problem and divide it into parts.
- Install JDK: If Java isn't installed on your system, download and install it.
- Open IDE(Integrated development environment) like VS code.
- Start by creating a new Java file with a .java extension (e.g., program.java).
- Write your code in a structural manner taking care of indentation to maintain readability.
- Execute your code

C. Algorithm:

- Sort all the edges from low weights to high.
- Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle then reject this edge.
- Keep adding edges until we reach all vertices.

D. Pseudocode:

```
KRUSKAL(G):
A = ∅
For each vertex v ∈ G.V:
MAKE-SET(v)
For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):
if FIND-SET(u) ≠ FIND-SET(v):
A = A ∪ {(u, v)}
UNION(u, v)
return A
```

E. Code:

```
// Kruskal's algorithm in Java

import java.util.*;

class Graph {
  class Edge implements Comparable<Edge> {
    int src, dest, weight;

    public int compareTo(Edge compareEdge) {
      return this.weight - compareEdge.weight;
    }
  };
```

```
// Union
class subset {
  int parent, rank;
};

int vertices, edges;
Edge edge[];

// Graph creation
Graph(int v, int e) {
  vertices = v;
  edges = e;
  edge = new Edge[edges];
  for (int i = 0; i < e; ++i)
    edge[i] = new Edge();
}

int find(subset subsets[], int i) {
  if (subsets[i].parent != i)
    subsets[i].parent = find(subsets, subsets[i].parent);
  return subsets[i].parent;
}

void Union(subset subsets[], int x, int y) {
  int xroot = find(subsets, x);
  int yroot = find(subsets, y);

  if (subsets[xroot].rank < subsets[yroot].rank)
    subsets[xroot].parent = yroot;
  else if (subsets[xroot].rank > subsets[yroot].rank)
    subsets[yroot].parent = xroot;
  else {
    subsets[yroot].parent = xroot;
    subsets[xroot].rank++;
  }
}
```

```
// Applying Krushkal Algorithm
void KruskalAlgo() {
  Edge result[] = new Edge[vertices];
  int e = 0;
  int i = 0;
  for (i = 0; i < vertices; ++i)
    result[i] = new Edge();

  // Sorting the edges
  Arrays.sort(edge);
  subset subsets[] = new subset[vertices];
  for (i = 0; i < vertices; ++i)
    subsets[i] = new subset();

  for (int v = 0; v < vertices; ++v) {
    subsets[v].parent = v;
    subsets[v].rank = 0;
  }
  i = 0;
  while (e < vertices - 1) {
    Edge next_edge = new Edge();
    next_edge = edge[i++];
    int x = find(subsets, next_edge.src);
    int y = find(subsets, next_edge.dest);
    if (x != y) {
      result[e++] = next_edge;
      Union(subsets, x, y);
    }
  }
  for (i = 0; i < e; ++i)
    System.out.println(result[i].src + " - " + result[i].dest + ": " + result[i].weight);
}

public static void main(String[] args) {
  int vertices = 6; // Number of vertices
  int edges = 8; // Number of edges
  Graph G = new Graph(vertices, edges);

  G.edge[0].src = 0;
  G.edge[0].dest = 1;
  G.edge[0].weight = 4;
```

```
    G.edge[1].src = 0;
    G.edge[1].dest = 2;
    G.edge[1].weight = 4;

    G.edge[2].src = 1;
    G.edge[2].dest = 2;
    G.edge[2].weight = 2;

    G.edge[3].src = 2;
    G.edge[3].dest = 3;
    G.edge[3].weight = 3;

    G.edge[4].src = 2;
    G.edge[4].dest = 5;
    G.edge[4].weight = 2;

    G.edge[5].src = 2;
    G.edge[5].dest = 4;
    G.edge[5].weight = 4;

    G.edge[6].src = 3;
    G.edge[6].dest = 4;
    G.edge[6].weight = 3;

    G.edge[7].src = 5;
    G.edge[7].dest = 4;
    G.edge[7].weight = 3;
    G.KruskalAlgo();
  }
}
```

F. Output:

```
PS C:\Users\Nitish\Desktop\JAVA\javaComeBack> cd "c:\Users\Nitish\Desktop\JAVA\javaComeBack\" ; if ($?) { javac KruskalAlgo.java } ; if ($?) { java KruskalAlgo }
1 - 2: 2
2 - 5: 2
2 - 3: 3
3 - 4: 3
0 - 1: 4
PS C:\Users\Nitish\Desktop\JAVA\javaComeBack>
```

G. Time Complexity:

O(E log E)

Learning outcomes:

- Understand the concepts of functions, classes and BFS.
- Understand how to use DFS.
- Understand the concept of the creation of a graph.
- Understanding how to deal with errors..
- Understand what the Kruskal algorithm is and how to implement it.
- Understand the adjacency matrix.