Noah Buchanan
Problem Set 8
Algorithms

November 24, 2020

UASpanningTree:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Queue;

public class UASpanningTree {

  /******************************
   * Name: Noah Buchanan
   * Username: ua505
   * Problem Set: PS8
   * Due Date: Nov 24, 2020
   ******************************/

  public int n;
  public int j;
  public int k;
  public final Vertex nil = new Vertex();
  public ArrayList<Vertex> vertices = new ArrayList<>();
  public ArrayList<Edge> edges = new ArrayList<>();
  public HashMap<Vertex, ArrayList<Vertex>> adj = new HashMap<>();

  public UASpanningTree(String filename) throws IOException {

    BufferedReader br = new BufferedReader(new FileReader(filename));
```

```
String line = "";

line = br.readLine();

String[] x = line.split(" ");

n = Integer.parseInt(x[0]);
j = Integer.parseInt(x[1]);
k = Integer.parseInt(x[2]);

for (int i = 0; i < n; i++) {
  vertices.add(new Vertex());
  adj.put(vertices.get(i), new ArrayList<Vertex>());
}

int count = 0;

while ((line = br.readLine()) != null) {

  x = line.split(" ");

  if (x[0].equalsIgnoreCase("P")) {

    edges.add(new Edge(true, v
    vertices.get(Integer.parseInt(x[1]) - 1),
        vertices.get(Integer.parseInt(x[2]) - 1)));
    vertices.get(Integer.parseInt(x[1]) -
     1).edges.put(vertices.get(Integer.parseInt(x[2]) - 1),
        edges.get(count));
    adj.get(vertices.get(Integer.parseInt(x[1]) -

     1)).add(vertices.get(Integer.parseInt(x[2]) - 1));
  } else {

    edges.add(new Edge(false, vertices.get
    (Integer.parseInt(x[1]) - 1),
        vertices.get(Integer.parseInt(x[2]) - 1)));
    vertices.get(Integer.parseInt(x[1]) -
     1).edges.put(vertices.get(Integer.parseInt(x[2]) - 1),
        edges.get(count));
    adj.get(vertices.get(Integer.parseInt(x[1]) -
     1)).add(vertices.get(Integer.parseInt(x[2]) - 1));
  }

  count++;
  }

}

public static void main(String[] args) throws IOException {

  boolean SpanningTree = false;
  UASpanningTree G = new UASpanningTree(args[0]);
  int combination = factorial(G.n);
  ArrayList<Edge> ommits = new ArrayList<>();
  int count = 0;
  while (SpanningTree == false) {
    Vertex u = G.vertices.get(G.n - 1);

    Edge e = G.primms(G, G.vertices.get(0));
    for (int i = 0; i < G.vertices.size(); i++) {
      if (i == G.n-1) {
        SpanningTree = true;
      }
      if(u != null) {
        if(u.pi != null) {
          u = u.pi;
        }
      } else {
        break;
      }
    }

    for (int i = 0; i < G.edges.size(); i++) {
      if (G.edges.get(i) == e) {
        G.edges.get(i).weight = -1000;
        G.edges.remove(i);
      }
    }
    if(count >= combination) {
      break;
    }
    System.out.println(SpanningTree);

  }

}

public class Vertex {

  HashMap<Vertex, Edge> edges = new HashMap<>();
  Vertex pi;
  int d;
}

public class Edge {

  public Edge(boolean color, Vertex From, Vertex To) {
    this.purple = color;
    this.To = To;
    this.From = From;
  }

  public Edge() {

  }

  boolean purple;
  Vertex To;
  Vertex From;
  int weight = 1;

}

public Edge primms(UASpanningTree G, Vertex r) {
  Edge e = new Edge();
  int purpleCount = 0;
  for (Vertex u : G.vertices) {
    u.d = Integer.MAX_VALUE;
    u.pi = null;
  }
  r.d = 0;
  r.pi = nil;
  Queue<Vertex> Q = new LinkedList<Vertex>();
  for (Vertex u : G.vertices) {
    Q.add(u);
  }
  while (!Q.isEmpty()) {

    Vertex u = Q.remove();
    for (Vertex v : G.adj.get(u)) {

      if (Q.contains(v) && u.edges.get(v).weight <= v.d && purpleCount < k) {

        v.pi = u;
        v.d = u.edges.get(v).weight;
```

```
            if (u.edges.get(v).purple) {
              purpleCount++;
              e = u.edges.get(v);
            }
          }

        }
      }
      return e;

    }

    public static int factorial(int n) {
          int fact = 1;
          int i = 1;
          while(i <= n) {
             fact *= i;
             i++;
          }
          return fact;
       }

  }
```