Noah Buchanan
Problem Set 9
Algorithms

December 15, 2020

My approach simply takes Dijkstra's algorithm and instead of running it on just one node, I run it on all vertices and save the result of the paths from running it each vertex making a sort of pseudo all pairs shortest path that follows the constraints set forth. I then use these different paths and find the smallest weight path from my weights file for the lookup and use the pred file to determine the path used to get there for that weight. The runtime comes out to $O(ev \log v)$ for the precompute so in the case of my implementation k would be the number of edges, and n would be the vertices, satisfying the $O(kn \log n)$ constraint. The runtime of each lookup is $O(n)$ as I have to look through all possible paths to make sure I am finding the smallest one in the file.

UAMapGraphCompute:

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

public class UAMapGraphCompute {

  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 StringBuilder weights = new StringBuilder();
  public StringBuilder pred = new StringBuilder();
```

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

  UAMapGraphCompute compute = new UAMapGraphCompute(args[0],args[1],args[2]);
  compute.ModifiedAllPairsDijkstra(compute, compute.vertices.get(0));
  BufferedWriter bw = new BufferedWriter(new FileWriter(args[1]));
  bw.write(compute.weights.toString());
  bw.close();
  bw = new BufferedWriter(new FileWriter(args[2]));
  bw.write(compute.pred.toString());
  bw.close();
}
```

```
public UAMapGraphCompute(String input, String output1, String output2) throws

  BufferedReader br = new BufferedReader(new FileReader(input));

  String line = "";

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

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

    //setting up the Vertices
    Vertex a = new Vertex(Integer.parseInt(split[0]),nil,0);
    Vertex b = new Vertex(Integer.parseInt(split[1]),nil,0);
    if(a.index == vertices.size()) {
      a = vertices.get(vertices.size()-1);
      vertices.add(b);
    } else {
      vertices.add(a);
      vertices.add(b);
    }

    //setting up the edges to and from a to b and b to a
```

```
    a.edges.put(b, new Edge(a,b,Integer.parseInt(split[2])));
    b.edges.put(a, new Edge(b,a,Integer.parseInt(split[2])));
    //putting the vertices in their adjacency list for use later in the algorit
    if(adj.get(a) == null && adj.get(b) == null) {

      adj.put(a, new ArrayList<Vertex>());
      adj.put(b, new ArrayList<Vertex>());
      adj.get(a).add(b);
      adj.get(b).add(a);
    } else if(adj.get(a) == null || adj.get(b) == null){

      if(adj.get(a)==null) {

        adj.put(a, new ArrayList<Vertex>());
        adj.get(a).add(b);
        adj.get(b).add(a);
      } else {

        adj.put(b, new ArrayList<Vertex>());
        adj.get(a).add(b);
        adj.get(b).add(a);
      }

    }
    //putting edges from a and b into edges list
    edges.add(a.edges.get(b));
    edges.add(b.edges.get(a));

  }

}


public void ModifiedAllPairsDijkstra(UAMapGraphCompute G, Vertex s) {

  for(int j = 0; j < G.vertices.size(); j++) {

    Dijkstra(G, G.vertices.get(j));

    for(int i = 0; i < vertices.size(); i++) {
      if(i != j) {
        weights.append(vertices.get(j).index + " " + vertices.get(i).index + "

      }
      pred.append(vertices.get(i).index + " " + vertices.get(i).pi.index + "\n
    }
```

```
    weights.append("————————————————————————————\n");
  }

}


public void Dijkstra(UAMapGraphCompute G, Vertex s) {
  Queue<Vertex> Q = new LinkedList<Vertex>();
  Q.add(s);
  for(Vertex v: G.vertices) {
    v.d = Integer.MAX_VALUE-100000;//I was getting some weird errors from this
    v.pi = G.nil;
    if(v != s) {
      Q.add(v);
    }
  }
  s.d = 0;
  while(!Q.isEmpty()) {

    Vertex u = Q.remove();

    for(Vertex v: G.adj.get(u)) {

      RELAX(u,v);
    }
  }
}




public void RELAX(Vertex u, Vertex v) {
  if(v.d > u.d + u.edges.get(v).weight) {
    v.d = u.d + u.edges.get(v).weight;
    v.pi = u;
  }
}






public class Edge {

  public Edge(Vertex From, Vertex To, int weight) {
```

```
      this.To = To;
      this.From = From;
      this.weight = weight;
    }

    public Edge() {

    }

    Vertex To;
    Vertex From;
    int weight = 1;

  }




  public class Vertex {

    public Vertex(int index, Vertex pi, int d) {
      this.index = index;
      this.pi = pi;
      this.d = d;
    }

    public Vertex() {

    }

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

}
```

UAMapGraphLookup:

```
import java.io.BufferedReader;
import java.io.FileReader;
```

```
import java.io.IOException;
import java.util.ArrayList;



public class UAMapGraphLookup {

  public static final Vertex nil = new Vertex();

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

    int src = 0;
    int dest = 0;
    BufferedReader br = new BufferedReader(new FileReader(args[2]));

    String line = "";

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


      String[] split = line.split(" ");
      src = Integer.parseInt(split[0]);
      dest = Integer.parseInt(split[1]);

      lookup(src,dest, args[0], args[1]);


    }
  }

  public static void lookup(int src, int dest, String weights, String pred) thro

    BufferedReader br1 = new BufferedReader(new FileReader(weights));
    BufferedReader br2 = new BufferedReader(new FileReader(pred));

    String line = "";
    int weight = 0;
    int count = 0;

    while((line = br1.readLine()) != null) {

      if(!line.equals("----------------------------------------")) {
        String[] split = line.split(" ");
        if(Integer.parseInt(split[0]) == src && Integer.parseInt(split[1]) == de
          weight = Integer.parseInt(split[2]);
          count++;
```

```
            break;
          }
        }
        count++;
      }

      if(weight == 0) {
        System.out.println("PATH UNDEFINED");
      } else {
        ArrayList<Vertex> vertices = new ArrayList<>();
        for(int i = 1; i < 4; i++) {
          vertices.add(new Vertex(i));
        }


        int index = (count - count % 3) + 1;
        count = 1;
        while((line = br2.readLine()) != null) {

          if(count >= index && count < index + 3) {

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

            if(Integer.parseInt(split[1]) != 0) {
              vertices.get(Integer.parseInt(split[1])-1).next = vertices.get(Integ
            }
          }
          count++;
        }

        System.out.print("SOURCE: " + src + "\tDEST: " + dest + "\t\tWEIGHT: " + w
        Vertex v = vertices.get(src-1);
        count = 0;
        while(v.next != null && count < Math.abs(src-dest)) {//a janky fix but it
          System.out.print(v.index + " => ");
          v = v.next;
          count++;
        }
        System.out.println(v.index);
      }
    }

    public static class Vertex {

      public Vertex(int index) {
        this.index = index;
```

```
    }

    public Vertex() {

    }


    Vertex pi;
    Vertex next;
    int index;
  }

}
```