PROJECT #3

Comp 482

Hrag Ayvazian

12/14/17

Hrag Ayvazian

```
/*****************isStronglyConnected**************************/
   public boolean isStronglyConnected() {
      int[] p = new int[nVertices];
      for(int i = 0; i < nVertices; i++) {
         for(int j = 0; j < nVertices; j++) {
            p[j] = -1;
         }
         dfs(i,p);
         for(int j = 0; j < nVertices; j++) {
            if(p[j] == -1) {
               return false;
            }
         }
      }
      return true;
   }

   private void dfs(int s, int[] p) {
      for(int i = 0; i < adjList[s].size(); i++) {
         if(p[adjList[s].get(i).vertex2] == -1) {
            p[adjList[s].get(i).vertex2] = s;
            dfs(adjList[s].get(i).vertex2, p);
         }
      }
   }
```

For my isStronglyConnected I got a runtime analysis of O(|V|^2+|V||E|). The way I got this runtime was by starting off at the isStronglyConnected method. Then looking at the next step in the isStronglyConnected method we see a for loop that loops the number of vertices which gives us a big O of |V|. After that we see another for loop that is nested inside the loop before giving us a time complexity of big O(|V|^2). Once that nested for loop is done I choose to used Depth First Search(DFS) and so I branch to the DFS method. In the DFS method I had a for loop that checks the number of edges on each vertices so the time complexity of this will be big O(|V||E|). However, when we merge it back with the isStronglyConnected method we will get V O(|V||E|) and so when you distribute the V you'll get O(|V|^2+|V||E|). After the DFS is done we have a nother for loop that loops around the number of vertices that will give us a time complexity of O(|V|^2). Then we are done with the isStronglyConnected but we have to evaluate all of the time complexities we have and since we have O(|V|^2) and O(|V|^2+|V||E|) and since O(|V|^2) is in O(|V|^2+|V||E|) we have an overall time complexity of O(|V|^2+|V||E|).

Hrag Ayvazian

Graph.java

```
/*
* Name: Hrag Ayvazian
* Date: 12/7/2017
* COMP 482
* Project #3
* */
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

public class Graph {
    //-----------------------------------------------------
    private ArrayList<EdgeNode>[] adjList;
    private int nVertices;
    private int nEdges;
    private String fileName;

    public static void main (String[] args)
    {
        //A
        System.out.println("Instructor Testcase A");
        System.out.println("\nDijkstra Shortest Paths");
        Graph g1 = new Graph("inputA.txt");
        g1.printGraph();
        int start = 1;

        SPPacket spp = g1.dijkstraShortestPaths(start);
        System.out.println("\nPrint shortest paths from start
vertex  = " + start);
        g1.printShortestPaths( spp );


        if( g1.isStronglyConnected())
            System.out.println( "\nGraph is strongly
connected");
        else
            System.out.println( "\nGraph is not strongly
connected");

        //B
        System.out.println("Instructor Testcase B");
        System.out.println("\nBellman Ford Shortest Paths");
        g1 = new Graph("inputB.txt");
        g1.printGraph();
```

```
        start = 0;

        spp = g1.bellmanFordShortestPaths(start);
        if( spp != null)
        {
            System.out.println("\nPrint shortest paths from
start vertex  = " + start);
            g1.printShortestPaths( spp );
        }
        else
            System.out.println("Graph has a negative cycle");

        //C
        System.out.println("Instructor Testcase C");
        System.out.println("\nBFS Shortest paths Shortest
Paths");
        g1 = new Graph("inputC.txt");
        g1.printGraph();
        start = 5;

        spp = g1.bfsShortestPaths(start);
        System.out.println("\nPrint shortest paths from start
vertex  = " + start);
        g1.printShortestPaths( spp );

        //D
        System.out.println("Instructor Testcase D");
        System.out.println("\nBellman Ford Shortest Paths");
        g1 = new Graph("inputD.txt");
        g1.printGraph();
        start = 0;

        spp = g1.bellmanFordShortestPaths(start);
        if( spp != null)
        {
            System.out.println("\nPrint shortest paths from
start vertex  = " + start);
            g1.printShortestPaths( spp );
        }
        else
            System.out.println("\nGraph has a negative cycle");

        if( g1.isStronglyConnected())
            System.out.println( "\nGraph is strongly
connected");
        else
```

```
            System.out.println( "\nGraph is not strongly
connected");

    }  //end main

    /******************Constructor**********************/
    public Graph(String inputFileName) {
        fileName = inputFileName;
        try {
            Scanner input = new Scanner(new File(fileName));
            nVertices = input.nextInt();
            adjList = new ArrayList[nVertices];
            nEdges = 0;
            for(int i = 0; i < nVertices; i++) {
                adjList[i] = new ArrayList<EdgeNode>();
            }
            while(input.hasNextInt()) {
                int v1 = input.nextInt();
                adjList[v1].add(new
EdgeNode(v1,input.nextInt(),input.nextInt()));
                nEdges++;
            }
        }
        catch(FileNotFoundException e) {}
    }

    /******************Print graph method***************/

    public void printGraph() {
        System.out.println("Graph: nVertices = " + nVertices + "
nEdges = " + nEdges);
        System.out.println("Adjacency Lists");
        for(int i = 0; i < nVertices; i++) {
            System.out.print("v = " + i + " ");
            System.out.println(adjList[i]);
        }
    }

    /******************BFS Shortest paths******************/
    public SPPacket bfsShortestPaths(int start) {
        int[] parent = new int[nVertices];
        int[] distance = new int[nVertices];
        boolean[] visited = new boolean[nVertices];
        int l = 0;
        Queue<Integer> visit = new PriorityQueue<>();
        for(int i = 0; i < nVertices; i++) {
```

```
            parent[i] = -1;
            distance[i] = Integer.MAX_VALUE;
            visited[i] = false;
        }
        visited[start] = true;
        distance[start] = l;
        visit.add(start);
        while(!visit.isEmpty()) {
            int current = visit.remove();
            int numAdj = adjList[current].size();
            for(int i = 0; i < numAdj; i++) {
                int dest = adjList[current].get(i).vertex2;
                if(!visited[dest]) {
                    visit.add(dest);
                    visited[dest] = true;
                    distance[dest] = distance[current] + 1;
                    parent[dest] = current;
                }
            }
        }
        return new SPPacket(start, distance, parent);
    }

    /**********Dijkstra's Shortest Path
Algorithm*****/
    public SPPacket dijkstraShortestPaths(int start) {
        int[] parent = new int[nVertices];
        int[] distance = new int[nVertices];
        boolean[] visited = new boolean[nVertices];
        Queue<Integer> visit = new PriorityQueue<>();
        for(int i = 0; i < nVertices; i++) {
            parent[i] = -1;
            distance[i] = Integer.MAX_VALUE;
            visited[i] = false;
        }
        visited[start] = true;
        distance[start] = 0;
        visit.add(start);
        while(!visit.isEmpty()) {
            int current = visit.remove();
            int numAdj = adjList[current].size();
            for(int i = 0; i < numAdj; i++) {
                int dest = adjList[current].get(i).vertex2;
                if(!visited[dest]) {
                    visit.add(dest);
                    visited[dest] = true;
```

```
                }
                if(distance[current] +
adjList[current].get(i).weight < distance[dest]) {
                    distance[dest] = distance[current] +
adjList[current].get(i).weight;
                    parent[dest] = current;
                }
            }
        }
        return new SPPacket(start, distance, parent);
    }

    /**********************Bellman Ford Shortest
Paths***************/
    public SPPacket bellmanFordShortestPaths(int start){
        int[] p = new int[nVertices];
        int[] d = new int[nVertices];
        EdgeNode[] edges = new EdgeNode[nEdges];
        int k = 0;
        for(int i = 0; i < nVertices; i++) {
            p[i] = -1;
            d[i] = Integer.MAX_VALUE;
            for(int j = 0; j < adjList[i].size(); j++) {
                edges[k] = adjList[i].get(j);
                k++;
            }
        }
        d[start] = 0;
        for(int i = 0; i < nVertices - 1; i++) {
            for(int j = 0; j < nEdges; j++) {
                if(d[edges[j].vertex1] + edges[j].weight <
d[edges[j].vertex2]) {
                    d[edges[j].vertex2] = d[edges[j].vertex1] +
edges[j].weight;
                    p[edges[j].vertex2] = edges[j].vertex1;
                }
            }
        }
        for(int j = 0; j < nEdges; j++) {
            if((long)d[edges[j].vertex1] + edges[j].weight <
(long)d[edges[j].vertex2]) {
                return null;
            }
        }
        return new SPPacket(start, d, p);
    }
```

```
    /************************Prints shortest
paths*************************/
    public void printShortestPaths(SPPacket spp) {
        System.out.println(spp);
    }


/******************isStronglyConnected*****************************
/
    public boolean isStronglyConnected() {
        int[] p = new int[nVertices];
        for(int i = 0; i < nVertices; i++) {
            for(int j = 0; j < nVertices; j++) {
                p[j] = -1;
            }
            dfs(i,p);
            for(int j = 0; j < nVertices; j++) {
                if(p[j] == -1) {
                    return false;
                }
            }
        }
        return true;
    }

    private void dfs(int s, int[] p) {
        for(int i = 0; i < adjList[s].size(); i++) {
            if(p[adjList[s].get(i).vertex2] == -1) {
                p[adjList[s].get(i).vertex2] = s;
                dfs(adjList[s].get(i).vertex2, p);
            }
        }
    }
}//end Graph class

/*****************************************/
class EdgeNode {
    int vertex1;
    int vertex2;
    int weight;
    public EdgeNode(int v1, int v2, int w) {
        vertex1 = v1;
        vertex2 = v2;
        weight = w;
    }
```

```
    public String toString() {
        return "(" + vertex1 + "," + vertex2 + "," + weight +
")";
    }

}

/************************************************/
class SPPacket {
    int[] d;//distance array
    int[] parent;//parent path array
    int source;//source vertex
    public SPPacket(int start, int[] dist, int[] pp) {
        source = start;
        d = dist;
        parent = pp;
    }
    public int[] getDistance() {
        return d;
    }

    public int[] getParent() {
        return parent;
    }

    public int getSource() {
        return source;
    }
    public String toString() {
        String str = "";
        str += ("Shortest Paths from vertex " + source + " to
vertex\n");
        for(int i = 0; i < parent.length; i++) {
            str += (i + ": [");
            String s = "";
            int p = i;
            if(p != source) {
                while(parent[p] != -1) {
                    s = parent[p] + "," + s;
                    p = parent[p];
                }
            }
            str += s;
            str += i;
            str += ("] Path Weight = " + d[i] + "\n");
        }
```

```
        return str;
    }
}
```

Hrag Ayvazian

Test Case

Instructor Testcase A

Dijkstra Shortest Paths
Graph: nVertices = 5 nEdges = 9
Adjacency Lists
v = 0 [(0,1,1), (0,2,1)]
v = 1 [(1,0,2), (1,3,5)]
v = 2 [(2,0,7), (2,4,3)]
v = 3 [(3,2,6), (3,4,4)]
v = 4 [(4,1,3)]

Print shortest paths from start vertex = 1
Shortest Paths from vertex 1 to vertex
0: [1,0] Path Weight = 2
1: [1] Path Weight = 0
2: [1,0,2] Path Weight = 3
3: [1,3] Path Weight = 5
4: [1,0,2,4] Path Weight = 6

Graph is strongly connected
Instructor Testcase B

Bellman Ford Shortest Paths
Graph: nVertices = 5 nEdges = 7
Adjacency Lists
v = 0 [(0,1,2), (0,4,4)]
v = 1 [(1,2,-1)]
v = 2 [(2,0,3), (2,3,1)]
v = 3 [(3,1,2)]
v = 4 [(4,3,-3)]

Print shortest paths from start vertex = 0
Shortest Paths from vertex 0 to vertex
0: [0] Path Weight = 0
1: [0,1] Path Weight = 2
2: [0,1,2] Path Weight = 1
3: [0,4,3] Path Weight = 1
4: [0,4] Path Weight = 4

Instructor Testcase C

Hrag Ayvazian

BFS Shortest paths Shortest Paths
Graph: nVertices = 6 nEdges = 10
Adjacency Lists
v = 0 [(0,1,1), (0,5,6)]
v = 1 [(1,2,1)]
v = 2 [(2,1,2), (2,4,1)]
v = 3 [(3,0,1), (3,4,1)]
v = 4 [(4,0,1), (4,5,4)]
v = 5 [(5,2,1)]

Print shortest paths from start vertex = 5
Shortest Paths from vertex 5 to vertex
0: [5,2,4,0] Path Weight = 3
1: [5,2,1] Path Weight = 2
2: [5,2] Path Weight = 1
3: [3] Path Weight = 2147483647
4: [5,2,4] Path Weight = 2
5: [5] Path Weight = 0

Instructor Testcase D

Bellman Ford Shortest Paths
Graph: nVertices = 6 nEdges = 7
Adjacency Lists
v = 0 [(0,5,-3)]
v = 1 [(1,2,2)]
v = 2 [(2,4,1), (2,0,1)]
v = 3 [(3,5,2)]
v = 4 [(4,0,-3)]
v = 5 [(5,1,1)]

Graph has a negative cycle

Graph is not strongly connected