Andreas Landgrebe
May 2, 2015
Computer Science 250: Analysis of Algorithms
Lab 9: Graphs and Graph Algorithms

Part One: An Airline Information System

```
/*
Andreas Landgrebe
Lab 9 Computer Science 250


*/
import java.util.Vector;
import java.util.*;
import java.io.*;

public class lab9airlines {
   public double weight;

   //DijkstraSP DSP = new DijkstraSP();
   //KruskaMST MST = new KruskaMST();
   //DirectedEdge DE = new DirectedEdge();
   //EdgeWeightedGraph EWG = new EdgeWeightedGraph();
   //Edge e = new Edge();

   public DijkstraSP DSP;
   public DijkstraAllPairsSP DAPSP;
   public KruskalMST MST;
   public DirectedEdge DE;
   public EdgeWeightedDigraph EWD;
   public EdgeWeightedGraph EWG;
   public Edge E;

   public DepthFirstSearch DFS;
   public BreadthFirstPaths BFS;

   public Vector<String> city;

   public String response;
   public String choice;
   public String cityOne;
   public String cityTwo;
   public String cityThree;

   public int c = 1;
   public int d;
   public int n;
   public int v;
   public int w;

   In in = new In();
   Graph G = new Graph(in);
```

```java
//Graph G = new Graph();

public void readTheTextFile(File file, Scanner scan) throws
    FileNotFoundException {
   scan = new Scanner(file);

   if(scan.hasNextInt()) {
      n = scan.nextInt();
   } else {
      System.out.println("Error");
      System.exit(0);
   } //if-else

   city = new Vector<String>();
   EWD = new EdgeWeightedDigraph(n+1);
   EWG = new EdgeWeightedGraph(n+1);

   for(int i = 0; i < n + 1; i++) {
      cityOne = scan.nextLine();
      city.add(i, cityOne);
   } //for

   scan.useDelimiter(" ");


   while(scan.hasNext()) {
      try {
         v = scan.nextInt();
         cityTwo = findCity(v, city);

         w = scan.nextInt();
         cityThree = findCity(w, city);

         d = scan.nextInt();
         weight = Double.parseDouble(scan.nextLine());

         DE = new DirectedEdge (v, cityTwo, w, cityThree, d, weight);

         EWD.addEdge(DE);

         E = new Edge(v, cityTwo, w, cityThree, d, weight);

         EWG.addEdge(E);

         EWD.addEdge(DE);

         DE = new DirectedEdge(w, cityThree, v, cityTwo, d, weight);

         EWD.addEdge(DE);
```

```
          E = new Edge(c, cityTwo, w, cityThree, d, weight);

          EWG.addEdge(E);

       } //try
       catch(InputMismatchException IME){
          IME.printStackTrace();

          System.out.println("Price: " + v + " " + w + " $ " + d);
          System.out.println(scan.nextFloat());

          break;
       } //catch
    }//while

    scan.close();

 }//readTheTextFile method


 public String findCity(int n, Vector city){
    return(String) city.elementAt(n);
 } //findCity method

 public void promptUser(Scanner scan) {
    scan = new Scanner(System.in);

    while(c != 0){

       System.out.println("Print All: 0 \n Span Tree: 1 \n Short: 2 \n
           Dollar: 3 \n Add: 4 \n Remove: 5 \n Quit : 6 \n");

       choice = scan.nextLine();
       switch(choice) {

          case"0": printAll();
          break;

          case"1": spanTree();
          break;

          case"2": shortest(scan);
          break;

          case"3": lowestPrice(scan);
          break;

          case"4": add(scan);
          break;
```

```
        case"5": remove(scan);
        break;

        case"6": c = 0;
        break;

      } //switch-case
   } //while
   scan.close();
} //promptUser method

public void printAll(){

   System.out.println(" * Departure * Destination * Distance * Cost
       *");
   System.out.println(EWD);
   System.out.println("The size of the disatnce vector" +
       city.size());

}//printAll method



public void spanTree(){

   MST = new KruskalMST(EWG);
   for(Edge E : MST.edges()) {
      StdOut.println(DE);
   }//for
   StdOut.printf("%.5f\n", MST.weight());
}//spanTree method


public void shortest(Scanner scan){

   System.out.println("Fewest Miles: 0 \n Lowest Price: 1 \n
       FewestFlights: 2");

   choice = scan.nextLine();
   switch(choice){
      case "0": fewestMiles(scan);
      break;

      case "1": lowestPrice(scan);
      break;

      case "2": fewestFlights(scan);
      break;
   } //switch-case
```

```
}//shortest method

public void fewestMiles(Scanner scan){
   System.out.println("Please Wait ... Fewest Miles Being Calculated,
       \n To stop, please type in STOP");

   choice = scan.nextLine();

   if(choice.equals("STOP")){
      return;
   } //if

   System.out.println("Starting City Name");
   cityTwo = scan.nextLine();

   if(city.indexOf(cityTwo) == -1){
      System.out.println("Sorry, no flight recorded");
      return;
} else {//if
   v = city.indexOf(cityTwo);
}//else

System.out.println("\n Enter Name of Arrival City");
cityThree = scan.nextLine();

if(city.indexOf(cityThree) == -1){
   System.out.println("Sorry, no flight recorded");
   return;
} else{ //if
   w = city.indexOf(cityThree);
}//else

DSP = new DijkstraSP(EWD, v);

if(DSP.hasPathTo(w)) {
   StdOut.printf("%s to %s (%.2f) ", cityTwo, cityThree,
       DSP.distTo(w));

   for(DirectedEdge ED : DSP.pathTo(w)) {
      StdOut.println(ED + " ");
   }//for

   StdOut.println();
} else {//if
   StdOut.printf("%s to %s (_) no path\n", cityTwo, cityThree);
}//else

}//fewestMiles method

public void lowestPrice(Scanner scan){
```

```
System.out.println("Please Wait ... Best Price Being Calculated,
    \n To stop, please type in STOP");

choice = scan.nextLine();

if(choice.equals("STOP")){
   return;
}//if

System.out.println("Please Type The City To Start In");

cityTwo = scan.nextLine();

if(city.indexOf(cityTwo) == -1){
   System.out.println("Sorry no flights recorded");
   return;
} else { //if
   v = city.indexOf(cityTwo);
}//else

System.out.println("Please Type the the Next City");

cityThree = scan.nextLine();

if(city.indexOf(cityThree) == -1) {

   System.out.println("Sorry no flights recorded");
   return;

} else {//if

   w = city.indexOf(cityThree);

}//else

//DFS = new DepthFirstSearch(EWG, v);
DAPSP= new DijkstraAllPairsSP(EWD);
for (int v = 0; v < EWG.V(); v++){
   if(BFS.hasPathTo(w)) {
      StdOut.printf("%s to %s(%d): ", cityTwo, cityThree,
          BFS.distTo(w));

      for(int x: BFS.pathTo(w)) {
         if (x == v) {
            StdOut.println(x);
         } else { //if
            StdOut.println("-" + x);

         } //else
```

```
            StdOut.println();
         } //for
      }  else {//if

         StdOut.printf("%s to %s (-): not connected\n", cityTwo,
              cityThree);
      } //else
   } //for
}//bestPrice method

public void fewestFlights(Scanner scan) {

}

public void add(Scanner scan){
   System.out.println("Are you Adding a New Edge?");
   response = scan.nextLine().toLowerCase();
   if(response.equals("no")){
      System.out.println();
      return;
   }else { //if
      System.out.println("Please enter city: ");
      cityTwo = scan.nextLine();
   if(city.indexOf(cityTwo) == -1){
      System.out.println("Sorry no flight recorded");
      return;
   } else {//if
      v = city.indexOf(cityTwo);
   }//else

   System.out.println("Please enter next city");

   cityThree = scan.nextLine();

   if(city.indexOf(cityThree) == -1) {
      System.out.println("Sorry no flight recorded");
         return;
      } else {//if
         w = city.indexOf(cityTwo);
      }//else
      System.out.println("Please Input the Distance of the Flight");

      d = scan.nextInt();

      if(d < 0){
         System.out.println("No negative disance, unless you can
              travel a negative distance");
         return;
      } //if
```

```
        System.out.println("Please input the cost of the flight");

        weight = scan.nextDouble();

        if(weight < 0){
           System.out.println("We could not care if you get paid of not
               to take a flight, we just want money");
           return;
        }// if

        DE = new DirectedEdge(v, cityTwo, w, cityThree, d, weight);
        EWD.addEdge(DE);

        E = new Edge(v, cityTwo, w, cityThree, d, weight);
        EWG.addEdge(E);

        DE = new DirectedEdge(v, cityTwo, w, cityThree, d, weight);
        EWD.addEdge(DE);

        E = new Edge(v, cityTwo, w, cityThree, d, weight);
        EWG.addEdge(E);
     } //if-else
     System.out.println("The Edge Has Been Created");
  }//add method

  public void remove(Scanner scan){

     System.out.println("Are you sure you want to remove the edge");

     response = scan.nextLine().toLowerCase();

     if(response.equals("no")) {
        System.out.println();
        return;
     }//if
     else{
        System.out.println("Please enter the first city");
        cityTwo = scan.nextLine();
        System.out.println("Please enter the second city");
        cityThree = scan.nextLine();

        String cityTwoBackUp = cityTwo;

        String cityThreeBackUp = cityThree;

        System.out.println("");
     }
  }//remove method
  public static void main(String[] args) {
    lab9airlines airlines = new lab9airlines();

    try {
      File file = new File("airlines.txt");
      Scanner scan = null;
      airlines.readTheTextFile(file, scan);

    } catch (Exception io) { //try

      io.printStackTrace();
      System.exit(0);
    } //catch

    try {
      Scanner sc = null;
      airlines.promptUser(sc);
    } //try
    catch (Exception io){
      io.printStackTrace();
    } //catch
  } //main method
} //lab9airlines class
```

Part 2: While You Have Some Downtime

1. Suppose you use a stack instead of a queue when running BFS. Does it still complete paths?

   Yes, it still still complete paths but then the algorithms would not be considered to be Breadth First Search (BFS). It would be considered to be DFS (Depth First Search). When DFS being used now, newly discovered vertices are visited right away, and vertices discovered longer ago are only returned to once the new vertices have been visited.

2. How many minimum spanning trees are possible in a generic DAG? Why?

   Technically and despite the name directed acyclic graphs may be represented in trees but not necessarily tress so there are no minimum spanning trees in a generic DAG. This is the case because of the possibility of marriages between distant relatives. Due to the fact that no one can become their own ancestor and due to merges, in a generic directed acyclic graph, there are no minimum spanning trees.

3. If we implemented Eager Prim without a priority queue, instead scanning through all entries in the distTo[] array to find the next vertex to add to the tree, what is the new order of growth for a graph with V vertices and E edges? Would this ever be an appropriate algorithm for Prim?

   Prim's algorithm would run in time proportional to V^2, which is optimal for dense graphs.