University of London International Programmes CO2226
Software engineering, algorithm design and analysis

Coursework assignment 2 (2016–17)

Submission details

What to hand in

Marks will be awarded for correct code (*i.e.* for code that produces results; if your code does not produce the correct result, no marks will be awarded for that part of the question).

You must submit exactly one Java file called Ass226<StudentID>.java. For example, if your student number ID is 101031722, your file will be named: `Ass226101031722.java`. When this file is compiled it must produce a file Ass226<StudentID>.class *e.g.* `Ass226101031722.class`, which, when run, must produce the answers to all the coursework assignment questions by implementing the appropriate algorithms.

Your java file may contain other classes, but they **must** all be in the same single java file. You must write your code assuming all data files are in the same directory as the program. Failure to do this will lead to a mark of zero! If you cannot complete a particular question, then the answer should say 'NOT DONE'. Your program should take the text files described below as command-line arguments.

To run your program, the examiner will type as follows (note that the filenames are referenced without the file extension):

```
java Ass226101031722 country_codes randomGraph
```

Your output should look like this:

```
Name: Joe Doe
Student ID: 101031722

 Question 1: 2
 Question 2: 200 205
 Question 3: 4
 Question 4: 15
 Question 5: [350, 352]
 Question 6: 5.12
 Question 7: 29.47

Execution Time: 32094 milliseconds
```

These are just sample answers to give you an idea of what output format is expected from your program. The examiner will change the data files to test your programs so make sure your program works with files containing less/more cities and country codes. Try deleting some lines from the files and see if your program gives different answers.

Efficiency

You will be penalised if your program runs too slowly (5 marks for every minute over 7 minutes on a machine with Intel Core i7 vPro processor with12 gigabytes of RAM).

Try to speed up your program, by not recomputing values that you have already computed. Instead, store them (rather than recomputing).

Use `System.nanoTime();` to time your program. (Read the value at the beginning and end of your program and subtract and divide by a million.)

IF YOU DO NOT INCLUDE THE EXECUTION TIME OF YOUR PROGRAM YOU WILL SCORE ZERO!

IF YOU DO NOT USE THE DATA THE EXAMINER HAS PROVIDED YOU WILL SCORE ZERO!

ALL SOLUTIONS SHOULD INVOLVE CALLS TO YOUR GRAPH INSTANCE METHODS; DO NOT TRY TO CHEAT BY FINDING ANSWERS ELSEWHERE!

Finding the shortest paths in unweighted graphs (breadth first search)

Find out about adjacency matrices for representing graphs. Here is a program to help you become familiar with them:

```
import java.util.HashSet; import java.util.ArrayList;


public   class graph
{


double [] [] adj;


graph (double [] [] a)
{
adj= new double [a.length][a.length]; for (int i=0;i<a.length;i++)
for (int j=0;j<a.length;j++) adj[i][j]=a[i][j];
}

public HashSet <Integer> neighbours(int v)
{
HashSet <Integer> h = new HashSet <Integer> () ;
for (int i=0;i<adj.length;i++) if (adj[v][i]!=0) h.add(i); return h;
}


public   HashSet <Integer> vertices()
{
HashSet <Integer> h = new HashSet <Integer>(); for (int
i=0;i<adj.length;i++) h.add(i); return h;
}


ArrayList <Integer> addToEnd (int i, ArrayList <Integer> path) //
returns a new path with i at the end of path
{
ArrayList <Integer> k; k=(ArrayList<Integer>)path.clone(); k.add(i);
return k;
}

public   HashSet <ArrayList <Integer>> shortestPaths1(HashSet <ArrayList
<Integer>> sofar,   HashSet <Integer> visited, int end)
{
HashSet <ArrayList <Integer>> more = new HashSet <ArrayList
<Integer>>();
HashSet <ArrayList <Integer>> result = new HashSet <ArrayList
<Integer>>();
HashSet <Integer> newVisited = (HashSet <Integer>) visited.clone();
boolean done = false;
boolean carryon = false;
for         (ArrayList <Integer> p: sofar)
{
for (Integer z: neighbours(p.get(p.size()-1)))
{

if (!visited.contains(z))
```

```
 {
 carryon=true; newVisited.add(z);
 if (z==end) {done=true; result.add(addToEnd(z,p));} else
                   more.add(addToEnd(z,p));
 }
 }
 }
 if (done) return result; else
 if (carryon) return shortestPaths1(more,newVisited,end); else return new
 HashSet <ArrayList <Integer>>();
 }


 public   HashSet <ArrayList <Integer>> shortestPaths( int first,
 int end)
 {
 HashSet <ArrayList <Integer>> sofar = new HashSet <ArrayList
 <Integer>>();
 HashSet <Integer> visited = new HashSet <Integer>(); ArrayList <Integer>
 starting = new ArrayList<Integer>();
 starting.add(first
 );
 sofar.add(starting
 ); if (first==end)
 return sofar; visited.add(first);
 return shortestPaths1(sofar,visited,end);
 }

 public static void main(String [] args)
 {
double  [ ] [] a = {
                              {0.0, 1.0, 1.0,
{0.0, 0.0, 1.0, 1.0},              0.0},
{0.0, 1.0, 0.0, 1.0},
{0.0, 1.0, 1.0, 0.0}
  };


   graph g = new graph(a);

 for (int i=0;i<a.length;i++)
 {for (int j=0;j<a.length;j++)
 if (i!=j) System.out.println(i + " to " + j +": "+g.shortestPaths(i,j));

 }
 }
 }
```

Draw a picture of the graph and see if you agree with the output. Play with the program and alter the graph in order to check that you understand how the program works.

The Capital Cities Distance Problem

Study the following files of data about capital cities:

- **countries_lon_lat.csv**. This file has five fields in the following order: country_id, longitude_1, latitude_1, longitude_2, latitude_2 and it will be used for calculating the distance between cities. We will be using the first

three fields of this file.

- **randomGraph.csv**. This file contains three fields, the first two being country codes and the last one being the weight if we decide to take this edge (*e.g.* 2,3,12 means that the cost for moving from vertex 2 to vertex 3 is 12).
- **country_codes.csv**. This file has a code followed by the short name and finally the full name of the country whose capital we are looking at (*e.g*. 2,USA, United States of America would mean that the country with country code 2 has a short name of USA and a full name of United States of America). We will be using only the first two columns of this file.

Study the following program:

```
import java.io.*;
import java.util.Scanner;
import java.util.*;
class capital_cities
{
static int N= 500;
static              double [][] edges = new double[N][N];
static TreeMap <Integer,String> countryNames = new
TreeMap
                   <Integer,String>();
static ArrayList<String> convert (ArrayList<Integer> m)
{
ArrayList<String> z= new ArrayList<String>(); for (Integer i:m)
 z.add(countryNames.get(i));
  return z;
}

static HashSet<ArrayList<String>> convert (HashSet<ArrayList<Integer>>
paths)
{
HashSet <ArrayList <String>> k= new HashSet
 <ArrayList<String>>();
for (ArrayList <Integer> p:paths)
k.add(convert(p));
          return k;
}

public static void main(String[] args) throws Exception
{
for(int i=0;i<N;i++)for(int j=0;j<N;j++)
 edges[i][j]=0.0;
Scanner s = new Scanner(new FileReader("random_graph"));
String z =s.nextLine();
while (s.hasNext())
{
z =s.nextLine();
String[] results = z.split(",");
edges[Integer.parseInt(results[0])][Integer.parseInt(results[1])]=1.0;
 edges[Integer.parseInt(results[1])][Integer.parseInt(results[0])]=1.0;
}
s = new Scanner(new FileReader("country_codes"));
z =s.nextLine();
while (s.hasNext())
{
z =s.nextLine();
```

```
String[] results = z.split(",");
countryNames.put(Integer.parseInt(results[0]),results[1]);
}
graph G= new graph (edges);
int st =Integer.parseInt(args[0]); int fin =
Integer.parseInt(args[1]); System.out.println("Shortest path from " +

       countryNames.get(st) + "     to " +
       countryNames.get(fin)
+ " is"  + convert(G.shortestPaths(st,fin)));
}
}
```

Dijkstra's algorithm (Finding the shortest path in a weighted graph)

Watch Dijkstra's Algorithm (YouTube video) and Dijkstra's Algorithm again!

Study Dijkstra's algorithm MIT Lecture 17 Video.

Please study the pseudo code below for Dijkstra's Algorithm to find a shortest path from `start` to `end`:

```
Set S = {start}; //S is the set of vertices for which the shortest paths
                    from start have already been found

HashMap <Integer,Double> Q = Map each Vertex to Infinity
        (Double.POSITIVE_INFINITY), except map start -> 0;
// Q.get(i) represents the shortest distance found from start to i found
so far
ArrayList <Integer> []   paths;
For each i
set path[i] to be the path just containing start.
while (Q is not empty)
{
let v be the key of Q with the smallest value;
//I've given you a method int findSmallest(HashMap <Integer,Double> t) for this
if (v is end and Q does not map v to infinity)
return paths[end]; let w be the value of v in Q;
add v to S;
for (each neighbour u of v) do
{
if (u       not in S)
{
let w1 be the the weight of the (v,u) edge + w;
if w1 < the value of u in Q, then do the following:
{
update Q so now the value of u is w1
update paths(u) to be paths(v) with u stuck on the end
}
}
  remove v from Q;
}
}
```

1. Implement Dijkstra's Algorithm using the examiner's pseudo-code above; namely, put a function `dijkstra` into the `graph` class.

Here are some hints:

```
int findSmallest(HashMap <Integer,Double> t)
{
Object [] things= t.keySet().toArray();
double val=t.get(things[0]);
int least=(int) things[0]; Set <Integer> k = t.keySet(); for (Integer i
: k)
{
if (t.get(i) < val)
 {
    least=i;
    val=t.get(i);}
}
return least;

}
```

Fill in these bits:

```
public       ArrayList <Integer> dijkstra (int start, int end)
{
int N= ...;
HashMap <Integer,Double> Q = new HashMap <Integer,Double>();

ArrayList <Integer> [] paths = new ArrayList [N]; for (int i=0;i<N;i++)
{
Q.put(i,...);
paths[i]=new ArrayList <Integer>(); paths[i].....;
}
HashSet <Integer> S= new HashSet(); S.add(...);
Q.put(start,....); while (!Q.isEmpty())
{
int v = findSmallest(...);
if (v==end && ...) return ....; double w = Q.get(...);
S.add(...);
for(int u: neighbours(v)) if (...)
{
double w1= ....;
if (w1 < Q.get(u))
{
Q.put(u,...);
paths[u]= addToEnd(...);
}
}
Q.remove(...);
}
return new ArrayList <Integer> ();
}
```

Test your implementation using the following test program

```
class testDijk
{
public static void main(String [] args) throws Exception
{
 int N=1000;
 double edges[][]=new double[N][N];
```

```
 for(int i=0;i<N;i++)
for(int j=0;j<N;j++)
edges[i][j]=0.0;
Scanner s = new Scanner(new FileReader("randomGraph"));
String z;
while (s.hasNext())
{
 z =s.nextLine();
String[] results = z.split(",");
edges[Integer.parseInt(results[0])][Integer.parseInt(results[1])]=Double
.parseDouble(results[2]);
      }
graph G= new graph (edges);
System.out.println(G.dijkstra(Integer.parseInt(args[0]),
   Integer.parseInt(args[1])));
}
}
```

Use this randomGraph file (please note that this is the example from last year which refers to tube stations so you may need to make some adjustments when reading in the data).

Each line of the file has three values: the first two are vertices and the third is the weight of the edge between them.

When we run:

```
java testDijk 0 999
```

we should get:

```
[0, 492, 665, 114, 452, 999]
```

Do you?

Coursework assignment 2 questions

Please note that the countries in the questions are referred to by their name; as part of the assignment, you will need to resolve them into their ids (*e.g.* 200 for United Kingdom). Note: in the case of a direct link, please ignore this option and look for a route that includes at least one intermediate link.

1. How many shortest paths are there between the capital cities of the United Kingdom (UK) and Greece (GRC)? A shortest path here means a path with a minimal number of vertices. (Note: Use the shortestPaths method above.)

   [15 marks]

2. Which pair of capital cities have the highest number of shortest paths between them? Just give the country ids.

   [10 marks]

3. How many shortest paths do they have?

   [10 marks]

4. How long are each of these shortest paths?
   Hint: You may wish to use the following method.

```
static ArrayList<Integer>      firstElement (HashSet <ArrayList
<Integer>>      s)
{
return ( ArrayList<Integer>)s.toArray()[0];
}
```

[10 marks]

5. Which set of capital cities are furthest away from the capital city of Denmark (DEN) in terms of number of stops? (Just print out a set of numbers corresponding to the countries.)

[15 marks]

6. What is the length in terms of sum of the weights of the edges of the shortest path between the capital cities of Italy (ITA) and Poland (POL)? Note: use Dijkstra's Algorithm.

[20 marks]

7. What is the length (in Km) of the shortest path (in terms of distance) between the capital cities of Germany (GMY) and United Kingdom (UK)? Note: Use Dijkstra's Algorithm.

   You will need to use the following method (and the relevant data from the countries_lon_lat.csv file).

```
static double realDistance(double lat1, double lon1, double lat2,
double lon2)
{
int R = 6371; // km (change this constant to get miles) double
dLat = (lat2-lat1) * Math.PI / 180;
double dLon = (lon2-lon1) * Math.PI / 180; double a =
Math.sin(dLat/2) * Math.sin(dLat/2) +
Math.cos(lat1 * Math.PI / 180 ) * Math.cos(lat2 * Math.PI / 180 )
       *  Math.sin(dLon/2)  * Math.sin(dLon/2);
double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a)); double d
= R * c;
return d;
}
```

   For finding the distance in Km between any two points on the Earth's surface with given latitude and longitude, the latitude and longitude of each capital city is given in the countries_lon_lat file. Use this to compute the adjacency matrix for the weighted graph representation of the capital cities problem. We need the adj[i][j] to be the distance from city i to city j now.

   You will also need to write a method for finding the length of path by adding up all the weights of the edges in the path.

[20 marks]

[END OF COURSEWORK ASSIGNMENT 2]