Université d'Ottawa
Faculté de génie

École de science
d'informatique
et de génie électrique

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical
Engineering
and Computer Science

Assignment 4

CSI2120 Programming Paradigms

Winter 2018

Questions 1,2 due on March 6th before 11:00 pm in Virtual Campus

Question 3,4 due on April 2nd, 2018 before 11:00 pm in Virtual Campus

[12 marks in total]

Problem Description

The City of Ottawa owns and maintains many wading pools which need to be regularly maintained. The repair crew needs to find a route to visit the pools. You will have to help the crew to find a reasonable route to visit all pools. A route is reasonable if all pools are visited and the route is not too long. An optimal route is not required.

This is a graph problem where the nodes are the pools and the edges have weights corresponding to the distance between pools (We will just use the Euclidean distance and do not consider the road network).

We will build our solution by building a tree approximating the problem and you will need to implement the following solution strategy:

- Build a tree
  - Sort the pools from West to East.
  - Store the most Western pool as the root node
  - Connect the closest pool with an edge as the child of the root.
  - For each pool from West to East connect the node for the pool with an edge as the child of the closest node in the tree.
  - This will result in an n-ary tree.
- Search for a route
  - Given the above tree, perform a pre-order traversal to find a route that visits all nodes.
  - Make sure to calculate the distance travelled in kilometers.

The attached JSON file has all the specifications necessary for the task. Note that the locations of the pools are given in GPS coordinates.

Each program will have to save the solution found in the specified files in the following text format with one pool and the total distance traveled from the root so far in kilometers. (Note that the example below is arbitrary and not the correct solution).

```
Crestview 0
Bellevue Manor 3.5
```

...

The distance between two locations stored as latitude and longitude can be found as follows (great circle interpolation):

Input: lat_1,lon_1 and lat_2, lon_2 in radians

Output: $distance$ in kilometers

$$d_{radians} = 2 * asin\left(sqrt\left(\left(sin\left(\frac{lat_1 - lat_2}{2}\right)\right)^2 + cos(lat_1) * cos(lat_2) * \left(sin\left(\frac{lon_1 - lon_2}{2}\right)\right)^2 \right)\right)$$

$$distance = 6371.0 * d_{radians}$$

Note that the GPS location use angles in degree but the above formula expects angles in radians. The angles can be converted as usual:

$$angle_{radians} = 180.0 * \frac{angle_{degrees}}{pi}$$

Question 0. File preprocessing (Python or Java)

The file that contains the geographic positions of all pools is contained in a JSON file. You can use either Python or Java to preprocess this file and thus produce a simpler text file that your solutions under the different paradigms will use. For example, you can create a Python application that will create a prolog pl file containing the database of pool names and locations.

You are allowed to use a (freely and legally available) package of your choice to help you parse the file or you may to choose to simply read the JSON file and parse yourself. In the case of Java, you must submit the jar file of the package that you use.

Question 1. Object-oriented solution (Java) **[3 marks]**

- Implement the tree construction as described above.
- Find the solution with pre-order traversal.
- Save the solution in the file *solution.txt* in the current directory. This text file must be formatted as specified above.

Your program must be a Java application that takes a command line argument the data file name.

In addition, you must also provide an UML class diagram showing all the classes with their attributes, methods and associations. You cannot use any static methods (except main).

Question 2. Logic solution (Prolog) **[3 marks]**

Write a Prolog predicate `findRoute/1` with the solution returned in the argument in the form of a list. You must also provide the predicate `saveRoute/2` that takes as first argument the above solution and as a second argument a file name in which the solution will be stored.

In addition, you must provide the list of the defined predicates with a short definition for each of them.

Question 3. Functional solution (Scheme) **[3 marks]**

Write a Scheme function `findRoute` with the file name of your intermediate file format as argument. You must also provide the function `saveRoute` that takes as first argument the above solution and as a second argument a file name in which the solution is stored and will evaluate to true on success. Note that to build your solution, you are not allow to use any of the standard functions that end with `a`!

```
(findRoute "pools.txt")
  ⇨ '(("Crestview" 0) ("Bellevue Manor" 3.5) … )

(saveRoute (findRoute "pools.txt") "solution.txt")
  ⇨ #t
```

In addition, you must provide the list of the defined functions with a short definition for each of them.

Question 4. Concurrent programming (Go) **[3 marks]**

Write a Go function `findRoute` with the file name of the intermediate file as the first argument and the number of Go routines to be used. Your Go solution must make use of concurrent programming by using a configurable number of Go routines. You must also provide the function `saveRoute` that takes as first argument the above solution and as a second argument a file name in which the solution is stored and returns true on success.

```
func findRoute( filename string, num int) route []Edge
func saveRoute( route []Edge, filename string) bool
```

In addition, you must explain the strategy you used to design a concurrent solution.

Submission

Please submit a single archive as a zip file in Virtual Campus. Your archive must include the files

- Preprocessing (due on March 6th)
  - Either
    - `Preprocessing.py`

  - Or
    - `Prepocess.jar` (which must include the Java source code)
    - `YourChoiceOfJSONPackage.jar`
  - The file(s) produced in your preprocessing of the JSON file.
- Solution
  - `FindRoute.jar` (which must include the Java source code) (due on March 6th)
  - `findroute.pl` (due on March 6th)
  - `findroute.scm` (due on April 2nd)
  - `findroute.go` (due on April 2nd)
- Documentation
  - A pdf document including the descriptions of your solutions (due on April 2nd)