COMP 30070 Practical Exam 2018

Attempt Parts 1-4 in sequence. **Submit only one Ruby program finally**. Aim to get each part working correctly before moving on to the next. The bulk of the marks will be awarded for how much you get working, so be sure to submit a working solution, commenting out sections of code if necessary. Marks are also awarded for clean design and coding, formatting and (light) commenting. You may write your program in one file called `main.rb` if you wish, or use several `.rb` files.
Test cases: are not necessary to achieve an A+ grade. Bonus marks will be awarded if test cases are provided.
Git: As explained in the Rules, you must develop your solution under Git source code control, and commit every 10 to 15 minutes.
Quality: The main script may become messy as you proceed to the later questions. Don't worry overly about this, focus on the quality of the code in your classes.

After downloading the input files and desired output at:
http://csserver.ucd.ie/~meloc/oop.zip
**** turn off your wireless network and bluetooth before the exam commences ****

1. A bus stop comprises an id, a name, and a location, where a location is a point, comprising an x- and y-coordinate. The BusStops class represents a number of BusStop instances. Add a sort_by_name! method to BusStops, and suitable to_s methods to all the classes, so that invoking to_s on BusStops produces a string representing all the bus stops.

   Implement the Point, BusStop and BusStops classes. In main.rb, load data from bus_stops.txt and invoke sort_by_name! and to_s on the BusStops object to produce the desired output.

   bus_stops.txt comprises one or more lines, each line containing:
   stop_id stop_name x_coordinate y_coordinate

   (30 marks)

2. A Route comprises a route id and a sequence of bus stop ids. The Routes class represents a number of routes. Implement the Route and Routes classes. In main.rb, load a Routes object with data from routes.txt and invoke to_s on it to produce the desired output. Note that a Route object must ask the BusStops object from Part (1) to find the bus stop name that corresponds to a given bus stop id.

   (20 marks)

3. A LineSegment comprises a start point and an end point (both instances of the Point class) and has a length method that computes the Euclidean distance[1] between the two points. Add an iterator called each_line_segment to the Route

---

[1] Euclidean distance between two points is square_root($(x_1 - x_2)^2 + (y_1 - y_2)^2$). **`Math.sqrt`** and **`x**2`** is the Ruby syntax you'll need.

class that yields each of the line segments in the route in turn, and use this to write a length(route_id) method in the Routes class that returns the total length of the given route. Hence extend main.rb to output the lengths of all the routes.

(30 marks)

4. Create a new class PathFinder that has access to the routes and bus stops objects, and a method find_all_paths(origin) that computes all possible paths that start from the given origin stop id. Hence extend main.rb to output all paths starting from stops 1, 3 and 5.

   Advice: Do this any way you want, and use recursion if you wish; what I write here is just one possible approach based on a non-recursive, breadth-first traversal. The bus routes essentially form a directed acyclic graph, so a traversal starting from the origin will find the required paths. Create an array of paths (an array of arrays of bus stop ids) seeded initially with one path containing just the origin stop id. Iteratively process the paths, asking the Routes object to find the next stops accessible from the current stop id (the current terminal stop id in the path being considered). If this stop has no successor stops, skip to the next path. If it has one successor stop, add this stop to the path. If it has *n* successor stops, replace the current path with *n* paths, all identical up to the final stop, the final stop being one of the *n* successor stops. Terminate the process when no further terminal bus stops have successors. The paths object will contain the desired paths.

   (20 marks)

Submission

1. Your submission should contain:
   1.1. All the `.rb` files you wrote, including one called `main.rb` that runs the main script for the program. No folders/directories please.
   1.2. Your test cases (if any), including `all_tests.rb` that runs all your tests.
   1.3. A file called `statement.txt` containing your name, student ID and a short statement of what you achieved, e.g., "Parts 1-3 completed correctly, Part 4 attempted but unfinished."

2. If you finish early, call an invigilator to supervise your submission. Otherwise, when the end of the exam is announced (1) do a final commit to your Git repository, (2) switch your wireless on, (3) push your local repository to your exam GitHub repository and (4) check your GitHub repository to make sure all your files are there.

3. **Immediately after pushing to GitHub**, zip your working folder/directory and email it to mel.ocinneide@ucd.ie.