**CSE104 Coursework 1**

**(Due date: 5pm 05/20/2020 China Time, online ICE submission ONLY)**

# Learning Outcomes

On successful completion of this assignment, students are expected to:

* understand and be able to apply a variety of data structures, together with their internal representation and algorithms;
* be able to make informed choices between alternative ways of implementation, justifying choices on grounds such as time and space complexity;
* be able to select, with justification, appropriate data structures to ensure efficient implementation of an algorithm.

# Background information

In this coursework, you are required to design the data structure and algorithm for a problem called the travelling salesman problem (TSP). The TSP is a well-known combinatorial optimisation problem. In this problem, a salesman is initially located at a starting position and he has a list of cities that each must be visited exactly once. After visiting all these cities, he then needs to return to his starting position. In the example below, city 0 is the starting point and all other cities need to be visited.

A screenshot of a cell phone

Description automatically generated

To simplify the problem, each city is given an identifier, an x coordinate and a y coordinate. The travelling distance from one city to another is the Euclidean distance calculated based on their coordinates and there is a path from every city to every other. What you need to work out in this problem is to find a tour that visits all of the cities and has a total travelling distance as short as possible. The tour can be represented by a sequence of city identifiers. For example: 0->1->3->5->4->6->2->0

A picture containing photo, sitting, looking, table

Description automatically generated

For a small problem like above, it is possible to find out the optimal solution with brute force. But for large problems, it is very time consuming to get the optimal solution. Actually, an algorithm that solves this problem efficiently (in polynomial-time) is not known. Thus, don’t bother finding the best solution in this coursework.

# Coursework requirements

In this coursework, you will be given several java files. You need to fill in the missing parts to solve TSP problems.

## Problem instances

Several sample questions to test your algorithms will be given. Each sample instance is a text file with a list of cities. The first city is your starting point. An example of the format of the text file is like the one below:

0 0 0

1 5 7

2 -12 12

3 28 51

…

In these text files, the first column is the city name, the second column is the city’s x coordinate and the third column is the city’s y coordinate. Columns will be separated by tabs (‘\t’). Once you have read everything into your program, convert the text into your inner data structure. Note that the starting point is not always located at (0, 0).

**Requirement 1: Write a file reader that reads problem instance files into your java program and convert city data into the objects of the class “City”. You need to apply the knowledge you learned from CSE104.**

**public class** City {  
 *// provide members here*

**public double** distance(City b) {  
 *// returns the distance from this to city b*  
 }  
}

**public static** ArrayList<City> readFile(String filename) {  
 ArrayList<City> cities = **new** ArrayList<>();  
 */\*  
 Provide code here  
 \*/* **return** cities;  
}

Can you improve the efficiency of the “distance()” function by adopting a clever design?

## A simple city visit algorithm

Once the text file is read into your program and converted into your City array, you then need to develop a simple city visit algorithm that visits the closest (in terms of Euclidean distance) unvisited cities one by one. The algorithm starts with city 0 as the first “current city”. For each current city, you need to calculate the traveling distance between this city and all unvisited cities, then visit the most nearby city among them. The visited city then becomes the next “current city”. Your algorithm should repeat this process until no more cities are unvisited. Finally, the salesman will return back to city 0 (The first and the last city visited is city 0).

**Requirement 2: fill in the function block below. You are allowed to add additional functions or variables to support it. Make sure that this algorithm is as efficient as possible in terms of the computational time.**

**public static** ArrayList<City> solveProblem(ArrayList<City> citiesToVisit) {  
 ArrayList<City> routine = **new** ArrayList<City>();  
 */\*  
 provide code here. you are allowed to add your own functions outside  
 But you are NOT allowed to modify method name, parameters, return type.  
 \*/* **return** routine;  
}

## Printing out routines and get total distance

Now that you are able to solve TSPs and get a proper solution represented by an ArrayList. The next task is to write a routine printout function. The function should also return the total distance of the whole routine. The message output of this function should follow the following format: 0->1->3->5->2->4->0. Apart from this, your program should not emit any other outputs when running.

**Requirement 3: fill in the function blocks below.**

**public static double** printSolution(ArrayList<City> routine) {  
 **double** totalDistance = 0.0;  
 */\*  
 provide code here. you are NOT allowed to call your own functions.*

*But you can use your own variables defined somewhere else!   
 \*/* **return** totalDistance;  
}

# Coursework Submission

You will be given three template files for this coursework: “Main.java”, “City.java” and “TSPSolver.java”, add your code into these files according to this coursework specification. You are free to change the Main.java in order to test your code. But the Main.java code will NOT be examined. I will simply replace it with my version when marking the coursework. All of your implementations should be put into the City class and the TSPSolver class.

You should submit your Java program code files together with your report to the entry I provided on ICE. The submitted program solution should be well commented and include three files: “City.java”, “TSPSolver.java” and a 4-page report with a file name “Report.pdf” or “Report.doc” (or docx).

The report should not exceed 4 pages, without counting your .java files submitted separately from this report) with the module title and your name/student number & signed declaration for non-plagiarism shown on the title page. Your report should explain your data structure(s), what data structures/algorithms (rendered in pseudo code) you have designed and how they are tested, debugged and used in your programs. Explain how error handling is done in your program. You should also analyse the space complexity (i.e. memory costs) of your data structures and the complexity of your city-visit algorithm (i.e. cost of running time asymptotically) in “Requirement 2”.

# Marking

The mark distribution (100% total) is arranged as follows:

1. The readFile function worth up to 15%.
2. The solveProblem function worth up to 50%.
3. The printSolution function worth up to 10%.
4. Your report worth up to 25%.

This assignment: 50% of the overall marks.