University of Westminster
Department of Computer Science

5SENG001W	Algorithms – Referred/Deferred Coursework (2020/21)
Module leader	Klaus Draeger
Unit	Referred/Deferred Coursework
Weighting:	50%
Qualifying mark	30%
Description	Algorithmic analysis of Dijkstra's shortest path algorithm on the "Travelling Salesperson" problem
Learning Outcomes Covered in this Assignment:	This assignment contributes towards the following Learning Outcomes (LOs): - LO2: Be able to apply the theory for the effective design and implementation of appropriate data structures and algorithms in order to resolve the problem at hand; - LO3: Be able to analyse, predict, compare and contrast the performance of designed and implemented algorithms, particularly in the context of processing data; - LO4: Be able to use a range of typical data structures and collections as part of Application Programming Interfaces (APIs) offered by programming languages; - LO5: Be able to apply the theory for the definition and implementation of novel algorithms.
Handed Out:	June 2021
Due Date	13:00, Monday, 12th July 2021
Expected deliverables	- **A zip file containing the source code in Java or C++.** - **A PDF document (up to 2 A4 pages) discussing your algorithmic solution.**
Method of Submission:	Electronic submission on Blackboard via a provided link close to the submission time.
Type of Feedback and Due Date: BCS CRITERIA MEETING IN THIS ASSIGNMENT	Written feedback within 15 working days. **2.1.1 Knowledge and understanding of facts, concepts, principles & theories** **2.1.3 Problem solving strategies** **2.1.5 Deploy theory in design, implementation and evaluation of systems** **2.2.2 Evaluate systems in terms of quality and trade-offs** **2.3.2 Development of general transferable skills** **3.2.2 Defining problems, managing design process and evaluating outcomes** **4.1.1 Knowledge and understanding of scientific principles** **4.1.2 Knowledge and understanding of mathematical and statistical principles** **4.2.1 Use theoretical and practical methods in analysis and problem solving**

Assessment regulations

Refer to section 4 of the "How you study" guide for undergraduate students for a clarification of how you are assessed, penalties and late submissions, what constitutes plagiarism etc.

Penalty for Late Submission

If you submit your coursework late but within 24 hours or one working day of the specified deadline, 10 marks will be deducted from the final mark, as a penalty for late submission, except for work which obtains a mark in the range 40 – 49%, in which case the mark will be capped at the pass mark (40%). If you submit your coursework more than 24 hours or more than one working day after the specified deadline you will be given a mark of zero for the work in question unless a claim of Mitigating Circumstances has been submitted and accepted as valid.

It is recognised that on occasion, illness or a personal crisis can mean that you fail to submit a piece of work on time. In such cases you must inform the Campus Office in writing on a mitigating circumstances form, giving the reason for your late or non-submission. You must provide relevant documentary evidence with the form. This information will be reported to the relevant Assessment Board that will decide whether the mark of zero shall stand. For more detailed information regarding University Assessment Regulations, please refer to the following website:**http://www.westminster.ac.uk/study/current-students/resources/academic-regulations**

Coursework Description

1. The Problem

In this problem, we try to find the shortest route which visits each town from a given list before returning to the starting point. We want you to perform the following **tasks** (more details are given below):

1. Create a data structure to represent the states of the Travelling Salesperson (TSP) problem.
2. Write a parser to read the names and coordinates of a list of towns from a given input file (we will be providing benchmark examples for you to test your implementation on) and initialise your data structure.
2a. (**Not required**, but highly recommended) Test your base implementation on some (small!) examples to check if it creates the data structure properly.
3. Implement Dijkstra's algorithm in order to find a shortest solution.
4. Create a variation of Dijkstra's algorithm which prevents it from re-visiting towns.
5. Run a performance analysis of both versions on a selection of benchmark examples.
6. Write a short report about your solution and analysis.

2. The "Travelling Salesperson" (TSP) problem

An example input for this problem might look like this:

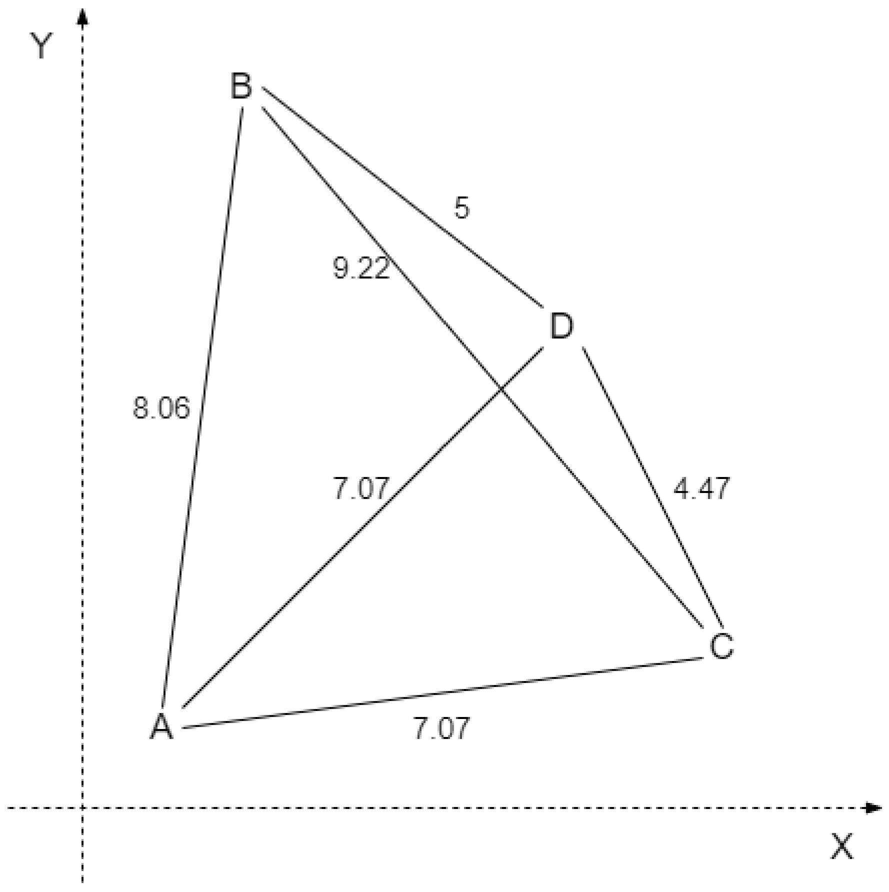
```
A 1 1
B 2 9
C 6 6
D 7 2
```

This represents the locations of four towns, A, B, C, and D, as pairs of integer (x, y) coordinates on a map of the area, with X and Y axes. For example, the town B is located at the point (2,9). The solution we are looking for is a tour which starts and ends at the first town (our hometown, which in the example is town A) and visits all the others in between, while being as short as possible. The output would be the names (or locations) of the towns in the order in which they are visited, and the total length of the tour. For example:

```
A                       1 1
C                       6 6
B              or       2 9
D                       7 2
A                       1 1
28.36168                28.36168
```

As usual, the distance between two towns is given by:
$\sqrt{(xdifference)^2 + (ydifference)^2}$.

For example, town B is at (2,9) and town C is at (6,6); their distance is $\sqrt{\mathbf{4^2 + 3^2}} = \mathbf{5}$. The full map including distances for the above example looks like this:



3. The data structure (Task 1)

In order to find a solution to the problem using Dijkstra's algorithm, we need a graph where the vertices represent all the possible states along a tour, and edges represent steps from one state to another. A state consists of

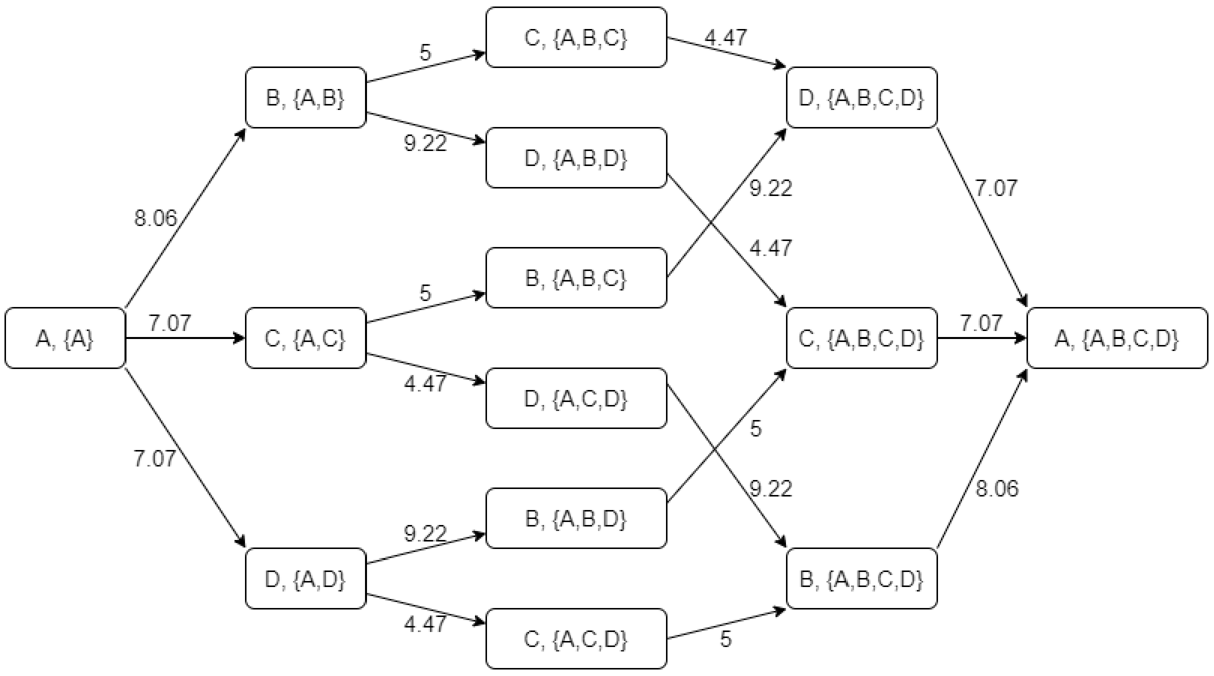
- a town (where we currently are)
- a set of towns (the ones we have visited so far)

For example, we could be in the state (D, {A, D}) where we are currently in town D and have so far visited towns A and D. If we next move to town C, we end up in the state (C, {A, C, D}).

In order to create a suitable data structure, you should:

1. choose a suitable **graph class,**
2. create **vertices** for all possible states (note that there can be a lot: for N towns there are $\mathbf{2^N}$ different subsets, so up to $\boldsymbol{O}(\boldsymbol{N2^N})$ vertices),
3. maintain a **lookup structure** which lets you look up the vertex representing a state (you will need this for the next item),
4. connect the vertices with all possible edges, each of which has the Euclidean distance between towns as its weight.

For the above example, the resulting graph (with some useless edges omitted) could look like this:



The lookup structure should help you build this graph in the following way: while adding the edges out of a vertex, e.g. (C, {A, C}), you would:

- look at each possible town to go to next (e.g., D),
- figure out what the current town and list of visited towns is in the resulting state (in this case D and {A, C, D}),
- use the lookup structure to find the vertex for this state.

One way of doing this could be an array of trees:

- first use the index of the current town to find the right tree in the array,
- then find the right vertex in the tree (going left/right based on which towns are/aren't in the visited set).

4. The parser (Task 2)

The purpose of the parser is to read names and locations from an input file which looks as described above. We have done this several times throughout the tutorial exercises, you may want to consult those examples. The main difference is that after reading the list of towns you need to figure out how to create all possible states. Your data structure should provide all the functions needed by the parser (like adding vertices and edges).

5. The algorithms (Tasks 3 and 4)

Dijkstra's algorithm has been covered in lecture 11. Implement it using your choice of data structure. We will only need a single-target version, because we always want to go from the state (A, {A}) (where we are in our hometown and have not visited any others yet) to the state (A, {A, B, ...}) (where we are back in our hometown, having visited all others).

For the variation (Task 5) we use the fact that for the Euclidean distance, a shortest tour will never re-visit a town (except at the end when we return home). This means that we ignore any edge to a state where the town is already in the current set, unless it is the goal state.

6. The analysis (Task 5)

In order to analyse the performance of your implementation, pick a suitable subset of input examples and create a table containing:

- the size of the example (i.e., the number of towns)
- the runtime of Dijkstra's algorithm
- the runtime of the variation

Make sure that you only measure the runtime of the algorithm itself, i.e., start the timer **after** creating the data structure. Also pick your examples so that the runtimes are meaningful (aim for at least 1 second. If none of the provided examples get you there, please let us know so that we can create some larger ones).
Based on the data, give an estimate of the complexity.

7. The report (Task 6)

Write a brief report (try to keep it below two A4 pages) containing the following:

a) A short explanation of your choice of data structure.
b) Output of your algorithm (i.e., a solution) on a small benchmark example.
c) The table of performance data and an analysis based on them. It should include a suggested order-of-growth classification (Big-O notation).

To be submitted:

- Your zipped source code in Java or C++. Your source code shall include header comments with your student ID and name.
- The report about the algorithmic performance analysis.

Coursework marking scheme:

Criterion and range	Indicative mark	Comments
Basics (0-10 marks)	10	A compilable and executable project has been created and follows programming guidelines for writing clear code.
	7	A compilable and executable project has been created but does not follow programming guidelines.
	3	A project has been created, but it is prone to compilation or runtime errors.
	0	Not done.
Task 1 (0-20 marks)	20	A data structure has been implemented, which has the following features: - Represents the states of the TSP problem. - Is equipped with a suitable lookup structure. - Correctly connects them with weighted edges. - Enables Dijkstra's algorithm via a suitable interface.
	5-15	A data structure has been implemented but lacks some or all of the features stated above.
	0	Not done.
Task 2 (0-20 marks)	20	A parser has been implemented and is able to create a graph representation from a given input file. It can handle any input file which has the format given in the problem description.
	10	A parser has been implemented and is able to create flow networks from the benchmark examples provided with the coursework specification, but not others.
	0	Not done.
Tasks 4 and 5 (0-30 marks)	30	Both versions of the algorithm done. A correct shortest solution (order of towns and length) can be calculated and output for any given input.
	10-20	Only one version implemented, or the implementation does not work for all possible inputs, or both.
	0	Not done.
Tasks 6 and 7 (0-20 marks)	20	The student has submitted a full report containing: - An explanation of the chosen data structure, - Output of the solution on a suitable example, - Performance data with accompanying analysis.
	5-15	The student has submitted a report, but some or all of the above are missing or lacking.
	0	Not done.