Cardiff School of Computer Science and Informatics

Coursework Assessment Pro-forma

Module Code:	CM2307
Module Title:	Object Orientation, Algorithms and Data Structures
Lecturer:	Bailin Deng
Assessment Title:	Implementation of a Road Map Data Structure
Assessment Number:	1
Date Set:	29 November 2021
Submission date and Time:	25 February 2022 at 9:30am
Return Date:	1 April 2022

This coursework is worth 50% of the total marks available for this module. If coursework is submitted late (and where there are no extenuating circumstances):

1. If the assessment is submitted no later than 24 hours after the deadline, the mark for the assessment will be capped at the minimum pass mark;

2. If the assessment is submitted more than 24 hours after the deadline, a mark of 0 will be given for the assessment.

Your submission must include the official Coursework Submission Cover sheet, which can be found here: `https://docs.cs.cf.ac.uk/downloads/coursework/Coversheet.pdf`.

Submission Instructions

Your coursework should be **submitted to Learning Central before 9:30am on the submission date**. It should be submitted **as a zip file** and should include **all** of the following files:

1. **One source file (`RoadMap.java`)**. Make sure to include (as comments) your student number at the top of the file. Also, follow this by any notes (as comments) regarding your submission. For instance, specify here if your program does not generate the proper output or does not do it in the correct manner.

2. **A text file (`report.txt`)** that explains the complexity of your code for determining if two places are connected by a path with charging stations (see the full coursework description for details).

3. **The official Coursework Submission Cover sheet** (**`[StudentNumber].pdf`**, with [StudentNumber] replaced by your student number).

Any deviation from the submission instructions above (including the number and types of files submitted) will result in a reduction 20% of the mark.

Staff reserve the right to invite students to a meeting to discuss coursework submissions.

Assignment

See the description starting at Page 4 for details.

Learning Outcomes Assessed

- Design algorithms and data structures.
- Implement fundamental data structures and algorithms.
- Test and evaluate performance of algorithms and data structures.

Criteria for assessment

Assessment and marking of your submitted program will be partly done by automatically checking the output of the submitted program against the solution program on a large set of test conditions. You can use the testbed at `https://testbed.cs.cf.ac.uk` to verify the correctness of your output: it allows you to upload your program and sample data files, and will run both your program and the solution program to compare their results. The testbed will indicate where and/or how much the results differ. It is highly recommended that you make extensive use of the testbed to check and refine your program. Although some simple test data is provided as part of the released coursework, you should create more data for more thorough testing.

The submitted file `report.txt` will be checked for correct analysis of the complexity of your algorithms.

Credit will be awarded according to the functionalities that you correctly implement (see instructions on the following page for details of each functionality), as well as time complexity of your algorithms. The total mark is a weighted sum of marks for different components, with the following percentage weights:

- 30% – loading and constructing the road map data structure
- 40% – determining if two places are connected by a path with charging stations: 25% for correct results, 15% for achieving the required complexity (for both the implementation and the analysis).
- 30% – finding the shortest path with charging stations.

The correctness of coding is evaluated based on the following criterion:

- The code produces correct results, or only requires minor changes to correct: 70%–100%.
- The code produces erroneous results, and can be fixed with small changes: 50%–69%.
- The code produces erroneous results, but can be fixed with major changes: 30%–49%.
- The code compiles, but with substantial errors in run-time that cannot be fixed easily: 10%–29%.
- No code submitted, or the code does not compile: 0%–9%.

The correctness of complexity is evaluated based on the following criterion:

- Correct implementation and correct analysis, or with minor error: 70%–100%.
- Correct implementation and incorrect analysis, or incorrect implementation and correct analysis, or with small changes required to achieve this correctness status: 40%–69%
- Correct implementation and incorrect analysis, or incorrect implementation and correct analysis, or with major changes required to achieve this correctness status: 20%–39%.
- No submission, or with both incorrect implementation and incorrect analysis that require substantial changes to fix: 0%–19%.

Feedback and suggestion for future learning

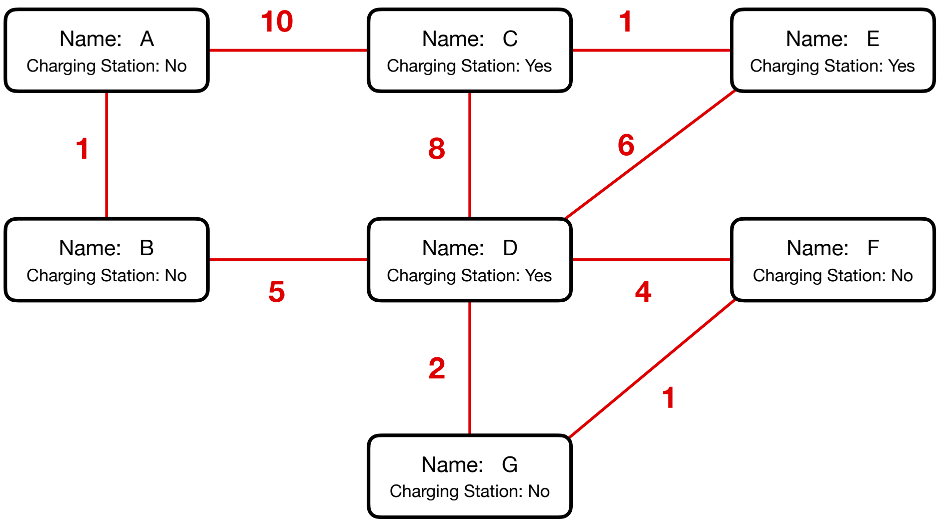
Feedback on your coursework will address the above criteria. Feedback and marks will be returned on 1 April 2022 via Learning Central.

Implementing a Road Map Data Structure

In the lectures you have learned that graph data structures can be used to represent road networks. In this coursework, you will implement such a data structure for road maps tailored for electric cars, providing functions that find paths between places.

The road map will contain a set of places, as well as road connections between them. It will be represented as a graph, where each place is a vertex and each road between two places is an edge. On each edge we store a number indicating the length of the road. Moreover, we assume that there is at most one road directly connecting two given places, so that there is no more than one edge between two vertices.

As currently electric cars are still limited in their range, when travelling in such cars it is important to ensure there are enough charging stations along the path taken. Therefore, we store in each vertex the availability of charging stations at the place. The following conceptual figure shows an example of such a road map graph. Here the rectangular frames represent places, and the red lines represent roads between them. The red number above each line is the length of the road.



In this coursework, you will implement a data structure for such graphs. As part of the handout materials, you are given a source file `RoadMap.java` that provides a skeleton for the class `RoadMap`. It is based on the adjacency list structure taught in the lectures. The class stores a list of vertices (represented using the `Vertex` class) and a list of edges (represented using the `Edge` class):

```
class RoadMap
{
    ...
    private ArrayList<Vertex> places;
    private ArrayList<Edge> roads;
    ...
}
```

The class `Vertex` contains a string for the name of the place, a dynamic array of edges that contain all incident roads for this place, a boolean variable `chargingStation` indicating the availability of charging station, and the integer `index` that is the 0-based index for this place within the dynamic array `places` of the `RoadMap` class.

```
private class Vertex
{
    ...
    private string name;
    private ArrayList<Edge> incidentRoads;
    private boolean chargingStation;
    private int index;
    ...
}
```

The class `Edge` stores the length of the road as an integer. It also has an array `incidentPlaces` that stores references to the two vertices that are incident with the edge:

```
private class Edge
{
    private int length;
    private Vertex[] incidentPlaces;
}
```

In this coursework, you will implementing the following functionalities:

1. **Loading a map from a file.** We will use a text file to store the road map. As an example, the file for the map figure on the previous page looks the following:

```
7 9
A 0
B 0
C 1
D 1
E 1
F 0
G 0
0 1 1
0 2 10
1 3 5
2 3 8
2 4 1
3 4 6
3 5 4
3 6 2
5 6 1
```

The file starts with a line that contains the number of vertices and the number of edges (7 and 9 respectively for this map). The content after that line contains two sections. The first section stores information about the vertices, with each line representing one vertex. For each vertex we store its name and availability of charging stations. The availability is written as an integer, with 1 meaning available, and 0 meaning unavailable. **The ordering of the lines within this section is the same as the ordering of their corresponding vertices in the `places` array.**

The second section stores information about the edges, with each line corresponding to one edge. Each line contains three numbers: the first two are 0-based indices of its two incident vertices within the `places` array; the last number is the length of the edge. For example, the line "`0 1 1`" means that the edge connects the vertices with index 0 and index 1 within the `places` array (i.e., A and B), and the length of this edge is 1.

The following method of `RoadMap` loads a map from a text file:

```
public void loadMap(String filename).
```

In the provided file `RoadMap.java`, there are already some codes inside this method. These codes read the content from the given file. You need to complete this method by using the read content to set up the data member `places` and `roads`. Specifically, you need to construct instances of `Vertex` and `Edge` classes, correctly set up their data members, and add them to the `places` and `roads` array in the correct order. Please read the comments in the method for hints.

2. **Determining if two places are connected by a path with charging stations.** When travelling between two places in an electric car, we want to ensure there are enough charging stations along the journey. Our data structure allows the user to query road connectivity subject to the availability of charging stations. This is to be implemented in the following method:

   ```
   public boolean isConnectedWithChargingStations(
           Vertex startVertex, Vertex endVertex).
   ```

   Given two vertices (`startVertex` and `endVertex`), this method determines whether there is a path that connects them while having a charging station on every intermediate vertex, i.e., **every vertex along the path, except for the given `startVertex` and `endVertex`, must have a charging station**. If such a path exists, then the method returns `true`. Otherwise, it returns `false`.

   You need to complete this method in `RoadMap.java`. **Your implementation needs to have worst-case time complexity no worse than $O(v+e)$, where v is the number of vertices, and e is the number of edges**. In your submitted file `report.txt`, please briefly explain your algorithm and analyse its worst-case time complexity.

3. **Shortest path with charging stations.** Finally, you need to complete the following method to find the **shortest** path with charging stations between the two given vertices.

   ```
   public ArrayList<Vertex> shortestPathWithChargingStations(
           Vertex startVertex, Vertex endVertex).
   ```

   The method returns the shortest path as a dynamic array containing all the vertices along the path, starting from `startVertex` and ending at `endVertex`. **Every vertex on the path, except for `startVertex` and `endVertex`, must have a charging station**. For this method, you can assume such a path exists, so that you do not need additional checking for its availability.

Testing Your Program

The provided `RoadMap` class already implements a `main` method that allows you to run the application using different command line options. According to the options, it will run the methods mentioned above to perform different operations, and output the result.

- To load a map and print its information:

  ```
  java RoadMap -i <MapFile>
  ```

An example file `Map.txt` is provided as part of the coursework handout. You can also create your own files for testing.

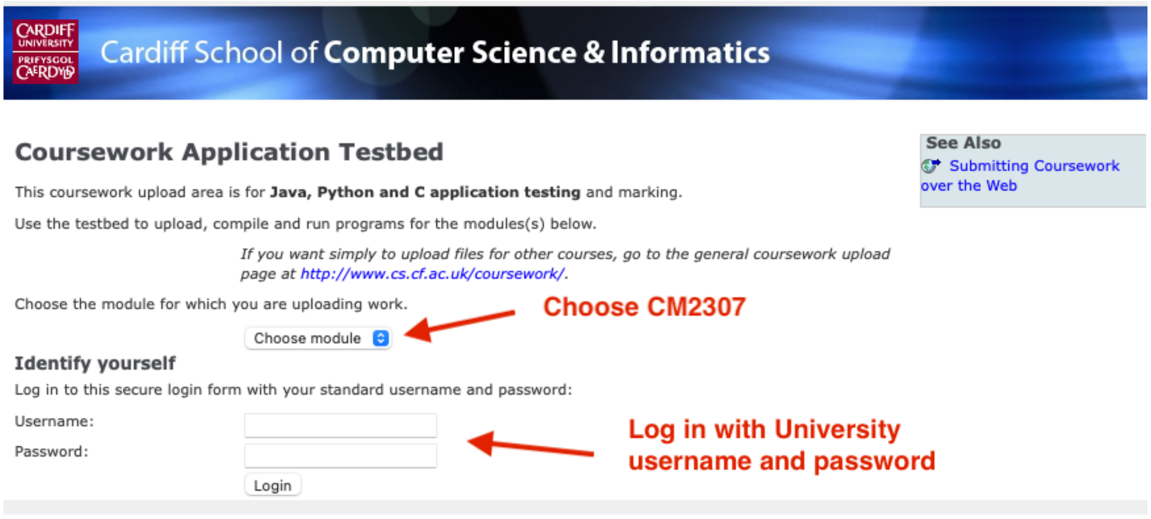
- To load a map file and find out the shortest path between two vertices:

```
java RoadMap -s <MapFile> <StartVertexIndex> <EndVertexIndex>
```

  Here `StartVertexIndex` and `EndVertexIndex` are 0-based indices for the two vertices within the `places` array. This command calls `isConnectedWithChargingStations(...)` to check if such a path exists. If it does, then the command continues to call `shortestPathWithChargingStations(...)` to find the shortest path with charging stations and print out the places along the path.

You can use the testbed platform https://testbed.cs.cf.ac.uk to upload your program and map files, and compare the output with the solution program. To do so:

1. Go to https://testbed.cs.cf.ac.uk, choose the module CM2307, and log in using your university username and password.


CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD
Cardiff School of Computer Science & Informatics
See Also
Submitting Coursework over the Web
Choose module
Login
Choose CM2307
Log in with University username and password

Coursework Application Testbed

This coursework upload area is for **Java, Python and C application testing** and marking.

Use the testbed to upload, compile and run programs for the modules(s) below.

If you want simply to upload files for other courses, go to the general coursework upload page at http://www.cs.cf.ac.uk/coursework/.

Choose the module for which you are uploading work.

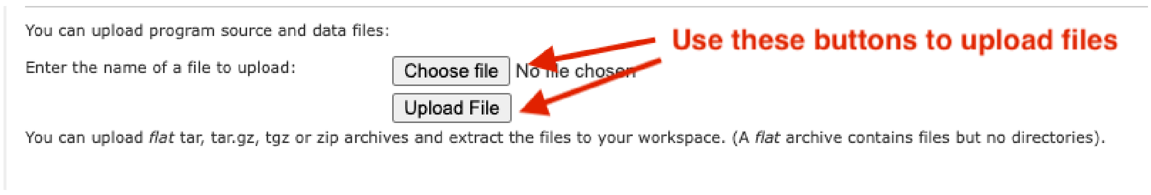
Identify yourself

Log in to this secure login form with your standard username and password:

Username:

Password:

2. Upload your source file(s) and data file(s) to the testbed.

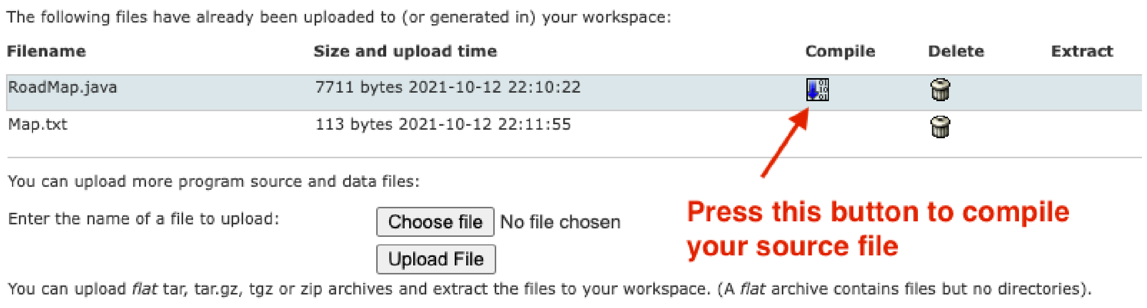


You can upload program source and data files:

Enter the name of a file to upload:

You can upload *flat* tar, tar.gz, tgz or zip archives and extract the files to your workspace. (A *flat* archive contains files but no directories).

3. Afterwards, you will see a list of your files, with a button in the ”compile” column for the source file. Press that button to compile your source code. If there is a compilation error, upload a new source file that fixes the issue and press the compile button again.


The following files have already been uploaded to (or generated in) your workspace:

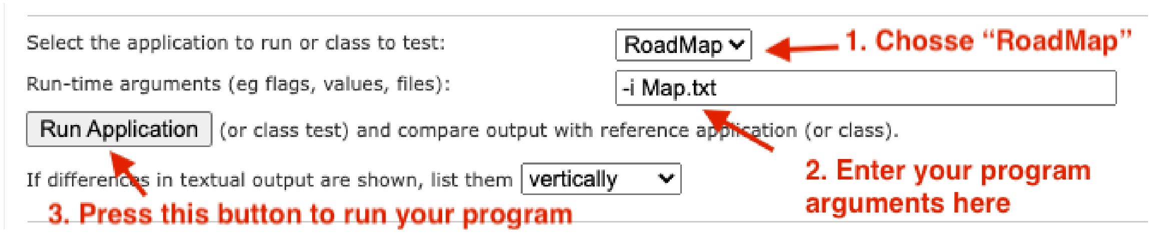
Filename	Size and upload time	Compile	Delete	Extract
RoadMap.java	7711 bytes 2021-10-12 22:10:22			
Map.txt	113 bytes 2021-10-12 22:11:55			

You can upload more program source and data files:

Enter the name of a file to upload:

You can upload *flat* tar, tar.gz, tgz or zip archives and extract the files to your workspace. (A *flat* archive contains files but no directories).

4. After the source code is compiled successfully, you can use the following interface to run your program and compare the result with the solution:


Select the application to run or class to test: RoadMap

Run-time arguments (eg flags, values, files): -i Map.txt

Run Application (or class test) and compare output with reference application (or class).

If differences in textual output are shown, list them vertically

1. Chosse "RoadMap"
2. Enter your program arguments here
3. Press this button to run your program

Please do not alter the `print` methods of the `RoadMap` class, as this may result in different printouts compared to the solution program even if your implementations are correct. Please also note that the provided example data files do not cover all the test cases that will be carried out when your submission is graded. Therefore, it is highly recommended that you test your program using more data.