RMIT UNIVERSITY

Algorithms and Analysis
COSC 1285/2123
Assignment 1: Implementing and Evaluating Data Structures for Maze Generation

Assessment Type	Individual assignment. Submit online via Canvas → Assignments → Assignment 1. Marks awarded for meeting requirements as closely as possible. Clarifications/updates may be made via announcements/assignment FAQ/relevant discussion forums.
Due Date	Week 6, April 18, Friday 11:59pm
Marks	20

Please read all the following information before attempting your assignment. This is an *individual* assignment. You may not collude with any other people and plagiarise their work. Everyone is expected to present the results of their own thinking and writing. Never copy other student's work (even if they "explain it to you first") and never give your written work to others. Keep any conversation high-level and never show your solution to others. Never copy from the Web or any other resource or use Generative AI like ChatGPT to generate solutions or the report. Remember you are meant to develop the solution by yourself - assessment is designed to encourage everyone to learn, and you are not doing yourself any favours by taking short cuts. Suspected collusion or plagiarism will be dealt with according to RMIT policy.

In the submission (your PDF file for the report of Task B) you will be required to certify that the submitted solution *represents your own work only* by agreeing to the following statement:

> *I certify that this is all my own original work. If I took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed in my submission. I will show I agree to this honor code by typing "Yes":*

1 Overview

In this assignment, you will implement a number of basic data structures for developing a maze generation program and evaluate their performance to determine what would be the best data structure in different situations. This assignment aims to help crystallise the analytical analysis parts of the course and familiarise you with certain data structures, how implementation decisions can have an influence on running times, and how to empirically evaluate different possible design choices (in this case, data structures).

2 Learning Outcomes

This assessment relates to 3 learning outcomes of the course which are:

- CLO 2: Compare, contrast, and apply key data structures: trees, lists, stacks, queues, hash tables and graph representations;
- CLO 4: Theoretically compare and analyse the time complexities of algorithms and data structures; and
- CLO 5: Implement, empirically compare, and apply fundamental algorithms and data structures to real-world problems.

3 Background

A maze is typically a 2D grid of cells and walls, an entrance and an exit. The aim is to find a path from the entrance to the exit. See Figure 1 for an example. There are strategies and algorithms to *solve* a maze, i.e., find such a solution path given a maze. There are also strategies and algorithms to *generate* a maze; in this assignment, we will focus on the generation of mazes.

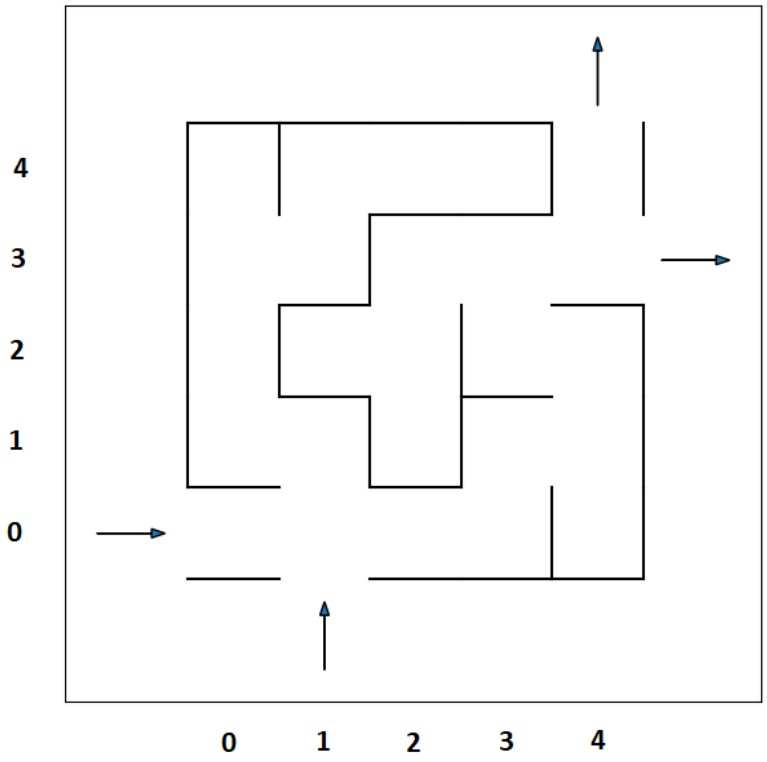

Figure 1: Sample 5 by 5 maze. The entrances are illustrated as arrows pointing towards the maze, and exits are illustrated as arrows pointing away from the maze. The cells of the maze are indexed from 0, and the row and column indices are drawn outside the maze. Note that the bottom left cell is (0,0), and top right cell is (4,4). This follows the convention that matplotlib follows (we use that for visualisation). Note the entrances are specified as (0,-1) and (-1,1) and exits as (5,4) and (3,5).

We are focused on *perfect* mazes, where there is always a path from entrance to exit (by the virtue that in a perfect maze, there is always a path from a cell in the maze to any other cell in the maze).

In addition, in this assignment, we focus on 2D, square cell mazes, i.e., each cell has 4 sides. There is no standard way to reference the cells, hence we adopted the convention issued in Figure 1, with (0,0) denoting the bottom left cell, and columns increase as we travel to the right of the maze (page), and rows increases as we travel up the maze (page). In order to specify the entrances and exits, we have added one row above, and one row below the maze, and one column to the left, and one column to the right of the maze. This means the row below the maze is referenced as -1, the row above by 5 (if we have 5 rows in the maze), the additional column to the left of the maze by -1, and the column to the right by 5 (if we have 5 columns in the maze). This allows us to still be able to use the original indexing, i.e., (0,0) refers to the bottom left cell of the maze, but still able to reference the location of entrances and exits.

4 Assessment details

The assignment is broken up into a number of tasks, to help you progressively complete the project.

Task A: Implement Mazes using Different Data Structures (6 marks)

In order to generate mazes, we need a way to represent them. In the assignment and this task, you will implement a Maze abstract data type using a number of data structures, namely *graphs*. Existing implementations using 2D arrays is provided as an example. Your implementation should complete the implementation of a number of operations in the provided Python skeleton code.

Data Structure Details

Mazes can be implemented using a number of data structures. We provide an implementation and you are to implement two others. We provide the implementations using the following data structures:

- 2D Array. Each element in the 2D array represents a cell or a wall in a 2D (square, 4 sided, cell) maze.

You are asked to implement the maze abstract data type using the following data structures:

- Graph (adjacency list). Adjacency list representation of a graph representing a 2D (square cell) maze.
- Graph (adjacency matrix). Adjacency matrix representation of a graph representing a 2D (square cell) maze.

file	description
`mazeTester.py`	Code that reads a configuration file then executes the generation using specified data structure. *No need to modify this file.*
`maze/maze.py`	Abstract class for mazes. All implementations of a maze should implement the Maze class. *No need to modify this file.*
`maze/util.py`	Coordinates class, and other utlity classes and methods. *No need to modify this file.*
`maze/graph.py`	Abstract class for graphs. All graph implementations should implement the Graph class. *No need to modify this file.*
`maze/arrayMaze.py`	2D array data structure implementation of a maze. *No need to modify this file.*
`maze/graphMaze.py`	Graph based data structure implementations of a maze. *Modify if need to.*
`maze/adjListGraph.py`	Code that implements an adjacent list (for maze). *Complete the implementation.*
`maze/adjMatGraph.py`	Code that implements an adjacent matrix (for maze). *Complete the implementation.*
`maze/maze_viz.py`	Modified code used to visualise generated mazes using matplotlib. *No need to modify this file.*
`generation/mazeGenerator.py`	Abstract class for maze generators. *No need to modify this file.*
`generation/recurBackGenerator.py`	Implementation of the recursive backtracking maze generator. *No need to modify this file.*
`README.txt`	Please read this first, it mentions how to run the code. *No need to modify this file.*

Table 1: Table of provided Python skeleton files.

For the above two data structures that we request you implement, you must program your own implementation, and not use libraries, e.g., networkx or numpy. Have a look at our implementations using a 2D array for an initial idea about how to possibly go about it. In the provided code, we make use of a limited number of packages. Apart from these cases in the provided skeleton code and built-in data types such as lists, dictionaries and arrays, this can be considered as an invalid implementation and attract 0 marks for that data structure. If in doubt, please ask first!

Code Framework

We provide Python skeleton code (see Table 1) to help you get started and ensure we have consistent interfacing so we can have consist correctness testing.

We also listed the files that you really need to modify/implement. You need to

complete the implementation of adjListGraph.py and adjMatGraph.py. We have provided the code to construct and provide functionaliy of a maze in the graphMaze file. It is mostly generic, but does assume a certain way of implementing the maze using the two graph data strucutres. If that doesn't align with your approach, please feel free to modify it, but please ensure the same functionality is still maintained and that you are able to generate mazes.

Note the first time you run this, please see the two provided configuration files, which will use the provided array data structure as the maze representation. If you modify and use one of the graph ones, there likely be errors returned when running, as the implementation for the graphs are empty. Please do not be alarmed by this - once the implementation is complete, we hopefully be able to output the correct graphs.

Note that for Task A, part of the assessment will be based on automated testing, which will feed a number of configuration files into your (completed) implementation of the provided Python skeleton. We will then check if the generated mazes are correct, e.g., whether they are perfect, have correct entrances and exits, didn't go into infinite loops etc. We would like you to do some testing about this - if you look at the code, there is a method to test if the generated maze is perfect, but no implementation. This implementation will not be part of your assessment, but it might be useful to consider how to implement this and do checking yourself, as part of practising evaluating your own code. Remember a perfect maze is one where any cell in the maze can reach any other cell, or another way to put it, there exists a path between any pair of cells in the maze. *If you do implement something to evaluate for perfect mazes, please do not submit that as part of your final code submission.*

Notes

- If you correctly implement the "Implement me!" parts of the provided skeleton, you in fact do not need to do anything else to get the correct output formatting. mazeTester.py will handle this.
- We will run your implementation on the university's core teaching servers, e.g., `titan.csit.rmit.edu.au`, `jupiter.csit.rmit.edu.au`, and `saturn.csit.rmit.edu.au`. If you develop on your own machines, please ensure your code compiles and runs on these machines. You don't want to find out last minute that your code doesn't compile on these machines. If your code doesn't run on these machines, we unfortunately do not have the resources to debug each one and cannot award marks for testing. *Please note this carefully, this is a firm requirement.*
- All submissions should be compiled with no warnings on **Python 3.6.8** when compiling the files specified in Table 1 - this is the default Python3 version on the Core teaching servers. Please ensure your code runs on the core teaching servers and with this version of Python. *Please note this carefully, this is a firm requirement.*

Task B: Evaluate your Data Structures (12 marks)

In this second task, you will evaluate your implemented structures both theoretically and empirically in terms of their time complexities for the different use case scenarios and different operations, e.g., removeWall(). Scenarios arise from different parameter settings for generating a maze.

Write a report on your analysis and evaluation of the different implementations. Consider and recommend in which scenarios each type of implementation would be most appropriate. The report should be **5 pages or less**, in font size 12. See the assessment rubric (Appendix A) for the criteria we are seeking in the report.

Use Case Scenarios

Typically, you may use real usage data to evaluate your data structures. However, for this assignment, you will write configuration file generators to enable testing over different scenarios of interest. There are many possibilities, hence to help you we suggest you consider the following factors.

- The dimensions of the maze.
- Data structure implementations.

Theoretical Analysis

At first, you are to conduct a theoretical analysis of the performance. Given the height h and width w (in terms of number of rows and columns) of the generated maze, report the best case and worse case running time estimate in terms of the exact asymptotic complexity (Θ) of each of your graph implementations when executing *updateWall()* and *neighbours()*. You will also need to provide an example scenario and explanation on the each of the best case and worse case running time estimate. This will help you when explaining your empirical analysis results. Put the results in the follow format of a table:

Theoretical Analysis		
Operations	Best Case	Worse Case
updateWall()	[asymptotic complexity] [example and explanation]	[asymptotic complexity] [example and explanation]
neighbours()	[asymptotic complexity] [example and explanation]	[asymptotic complexity] [example and explanation]

Empirical Analysis

Typically, you use real usage data to evaluate your data structures. However, for this assignment, you will write data generators to enable testing over different scenarios of interest.

Data Generation When generating different maze dimensions, you may want to write some configuration file generators (or generator directly within a copy of mazeTester.py). Due to the randomness of the data, you may wish to generate several datasets with the same parameters settings and take the average across a number of runs.

> Note, you may be generating and evaluating a significant number of datasets, hence we advise you to get started on this part relatively early.

Task 3: Video Interview

After the report and code is submitted, you will be asked to record your responses to a number of questions in Canvas. These questions will ask you about aspects of your implementation or report. You'll have a set time to consider the questions, make a recording then upload that recording. More details will be explained closer to submission.

5 Report Structure

As a guide, the report could contain the following sections:

- Theoretical analysis on running time and complexities of the different data structure implementation as outlined in Section 4.
- Explain your data and experimental setup. Things to include are (brief) explanations of the parameter configurations in your experiments, e.g., the range of maze dimensions sizes (add some brief explanation of why this range selection), describe how the evaluation parameters and data are generated (a paragraph and perhaps a figure or high level pseudo code suffice) and which approach(es) you decide to use for measuring the timing results.
- Evaluation of the data structures using the generated data. Analyse, compare and discuss your results. Provide your explanation on why you think the results are as you observed. You may consider using the known theoretical time complexities of the operations of each data structure to help in your explanation.
- Summarise your analysis as recommendations, e.g., for this range of maze dimensions, I recommend to use this data structure because... We suggest you refer to your previous analysis to help.

Clarification to Specifications

Please periodically check the assignment FAQ for further clarifications about specifications. In addition, the lecturer will go through different aspects of the assignment each week, so even if you cannot make it to the lectorials, be sure to check the course material page on Canvas to see if there are additional notes posted.

6 Submission

The final submission will consist of three parts:

- Your **Python source code** of your implementations. This must include all files, including the provided skeleton as well as any extra files you have created. Your source code should be placed into the same structure as the supplied skeleton code, and the root directory/folder should be named as `Assign1-<your student number>`. Specifically, if your student number is s12345, when `unzip Assign1-s12345.zip` is executed then all the source code files should be in directory Assign1-s12345. We use automated testing and compilation, and the testing script will expect this structure, so if is different, the script may not be able to compile your code. So please make sure not to change the structure.

- Your **written report for part B** in PDF format, called "assign1.pdf". Place this pdf within the Python source file directory/folder.
- Your **data generation code**. Create a sub-directory/sub-folder called "dataGen" within the Python source file directory/folder. Place your data generation code within that folder.
- Then, the Python source file folder (and files within) should be zipped up and named as `Assign1-<your student number>.zip`. E.g., if your student numbers is s12345, then your submission file should be called `Assign1-s12345.zip`, and when we unzip that zip file, then all the submission files should be in the folder Assign1-s12345.

Note: **submission of the report and code will be done via Canvas**. We will provide details closer to the submission deadline.

7 Assessment

The project will be marked out of 20.

The assessment in this project will be broken down into three parts. The following criteria will be considered when allocating marks.

Implementation (6/20):

- You implementation will be assessed based on the number of tests it passes in our automated testing. The tests will just look at things such as whether the generated graph is perfect, is of the right size, ran successfully (and didn't go into an infinite loop or take too long) etc.
- While the emphasis of this project is not programming, we would like you to maintain decent coding design, readability and commenting, hence commenting and coding style will make up a portion of your marks.

Report (12/20):

The marking sheet in Appendix A outlines the criteria that will be used to guide the marking of your evaluation report[1]. Use the criteria and the suggested report structure (Section 4) to inform you of how to write the report.

Video Interview 2/20:

This is a pass/fail assessment, and you'll be assessed based on your ability to answer some questions about the code and report.

Late Submission Penalty: Late submissions will incur a 10% penalty on the total marks of the corresponding assessment task per day or part of day late, i.e, 2 marks per day. Submissions that are late by 5 days or more are not accepted and will be awarded

[1]Note for the marking guide, if one of the criteria is not demonstrated, then 0 marks will be awarded for that criteria.

zero, unless special consideration has been granted. Granted Special Considerations with new due date set after the results have been released (typically 2 weeks after the deadline) will automatically result in an equivalent assessment in the form of a practical test, assessing the same knowledge and skills of the assignment (location and time to be arranged by the coordinator). Please ensure your submission is correct (all files are there, compiles etc), re-submissions after the due date and time will be considered as late submissions. The core teaching servers and Canvas can be slow, so please do double check ensure you have your assignments done and submitted a little before the submission deadline to avoid submitting late. We strongly advice you submit **at least one hour before the deadline**. *Late submissions due to slow processing of Canvas or slow Internet will not be looked upon favourly, even if it is a few minutes late. Slow processing of Canvas or slow Internet will require documentation and evidence submission attempts was made at least one hour before the deadline.*

Assessment declaration: By submitting this assessment, you agree to the assessment declaration - `https://www.rmit.edu.au/students/student-essentials/assessment-and-exams/assessment/assessment-declaration`

8 Academic integrity and plagiarism (standard warning)

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarised, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods
- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites. If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offence constituting misconduct. Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source
- Copyright material from the internet or databases
- Collusion between students

For further information on our policies and procedures, please refer to the following: `https://www.rmit.edu.au/students/student-essentials/rights-and-responsibilities/academic-integrity`.

9 Getting Help

There are multiple venues to get help. There are weekly consultation hours (see Canvas for time and location details). In addition, you are encouraged to discuss any issues you

have with your Tutor. We will also be posting common questions on the Assignment 1 Q&A section on Canvas and we encourage you to check and participate in the EdForum discussion forum. However, please **refrain from posting solutions**, particularly as this assignment is focused on algorithmic and data structure design.

A Marking Guide for the Report

Design of Evaluation (Maximum = 2 marks)	**Analysis of Results** (Maximum = 8 marks)	**Report Clarity** (Maximum = 2 marks)
2 marks Data generation is well designed, systematic and well explained. All suggested scenarios, data structures and a reasonable range of size of the maze were evaluated. Each type of test was run over a number of runs and results were averaged.	8 marks Analysis is thorough and demonstrates understanding and critical analysis. Well-reasoned explanations and comparisons are provided for all the data structures, scenarios and different input sizes of the mazes. All analysis, comparisons and conclusions are supported by empirical evidence and theoretical complexities. Well reasoned recommendations are given.	2 marks Very clear, well structured and accessible report, an undergraduate student can pick up the report and understand it with no difficulty.
1.4 marks Data generation is reasonably designed, systematic and explained. There are at least one obvious missing suggested scenarios, data structures or reasonable size of the maze. Each type of test was run over a number of runs and results were averaged.	5.5 marks Analysis is reasonable and demonstrates good understanding and critical analysis. Adequate comparisons and explanations are made and illustrated with most of the suggested scenarios and input sizes of the maze. Most analysis and comparisons are supported by empirical evidence and theoretical analysis. Reasonable recommendations are given.	1.4 marks Clear and structured for the most part, with a few unclear minor sections.
0.7 mark Data generation is somewhat adequately designed, systematic and explained. There are several obvious missing suggested scenarios, data structures or reasonable size of the maze. Each type of test may only have been run once.	3 marks Analysis is adequate and demonstrates some understanding and critical analysis. Some explanations and comparisons are given and illustrated with one or two scenarios and sizes of the maze. A portion of analysis and comparisons are supported by empirical evidence and theoretical analysis. Adequate recommendations are given.	0.7 mark Generally clear and well structured, but there are notable gaps and/or unclear sections.
0 marks Data generation is poorly designed, systematic and explained. There are many obvious missing suggested scenarios, data structures or reasonable size of the maze. Each type of test has only have been run once.	0-1 marks Analysis is poor and demonstrates minimal understanding and critical analysis. Few explanations or comparisons are made and illustrated with one scenario and size setting. Little analysis and comparisons are supported by empirical evidence and theoretical analysis. Poor or no recommendations are given.	0 marks The report is unclear on the whole and the reader has to work hard to understand.