Data Structures and Algorithms 1
Assessment Unit 1 – Software Project

Adam Sampson and Ruth Falconer
School of Design and Informatics

Introduction

In this assessment, you will demonstrate, using a software project and presentation, your achievement of the module's learning outcomes:

- Describe abstract and concrete requirements for data structures and algorithms.
- Describe a range of standard data structures and algorithms, in terms of both functionality and performance characteristics.
- By reasoning about behaviour and performance, be able to critically select appropriate data structures and algorithms for a given application within a software project.

If you have any questions, please ask on Teams, or contact Adam (**a.sampson@abertay.ac.uk**) or Ruth (**r.falconer@abertay.ac.uk**).

The application

Your application must:

- implement **two different standard algorithms** that solve the same real-world problem;
- make use of **appropriate data structures** for the application's needs;
- allow you to **compare the performance** of the two algorithms as you vary the **size of the input data**.

Here are some suggestions:

- Shortest-path algorithms – e.g. Lee, Dijkstra, A* – applied on a 2D grid or an arbitrary graph. You can use the shortest-path lab exercise as a starting point for this. For example, you might implement pathfinding around a simulated world in a game, or selecting paths for packets being sent around a network.
- String searching algorithms – e.g. Boyer-Moore, Aho-Corasick – to find the occurrences of a particular string within a large body of text. You can use the text-searching lab exercise as a starting point for this. For example, you might take a webserver log file and use an appropriate data structure to track how many times each page has been accessed.
- Tree searching and update algorithms – e.g. for different types of binary trees. For example, you might build a tree representing a complex directory structure on disk, and use search algorithms to efficiently find files that have particular properties, or identify duplicate files.

To meet the learning outcomes for the module, you need to demonstrate that you can implement reasonably complex algorithms by yourself. The following algorithms are **not allowed** because they're too simple to demonstrate this: brute-force string search, linear search, binary search, any sorting algorithm. (See the module FAQ for more details.)

If none of the ideas above appeal, have a look at the books on the module's reading list (e.g. Sedgewick's “Algorithms”), which describe a wide range of standard algorithms. If you're unsure whether your idea would meet the requirements, please talk to us.

We suggest that you use C++ for this assessment. If you'd like to use a different programming language, please check with Adam or Ruth to make sure that your project will meet the assessment's requirements. Your application may run on any operating system or platform.

If your project makes use of code from an external source – for example, libraries such as Boost or SFML, or examples from online sources or textbooks – then this must be **clearly acknowledged and referenced** in your submission. Please bear in mind that when we mark your project, we can only give you credit for **your own work**, not for code you've obtained from elsewhere – for example, if you're comparing pathfinding algorithms, using the A* implementation from the Boost library would not get you any marks.

The presentation

During week 14, you will give a short online presentation with slides to one of the teaching staff, covering the following topics:

- What problem you're solving, and which algorithms you've chosen to implement. You **don't** need to describe how these algorithms work during the presentation.
- Which data structures you've chosen to use in your application, and why, justifying your choice with reference to time/memory complexity of their operations and the needs of the algorithm. You should highlight places where choice of an appropriate data structure has simplified the code or improved the performance of the application.
- A comparison of the performance characteristics of your two algorithms as the size of the input data is varied. You should state the theoretical time complexity of the algorithms in the size of the input, and then show the results of timing measurements on a chart to show how your implementations compare to the theoretical performance.

Your presentation should last no longer than ten minutes; there will be approximately five minutes for questions and discussion afterwards.

We will assign you a presentation time during **the week starting Monday 4th January 2021**; a list of presentation times will be available on MyLearningSpace. If the time we suggest isn't possible for you, please get in touch with Adam as soon as possible to arrange an alternative.

You must submit your slides **in PDF format** to the “Presentation slides” assignment on MyLearningSpace **before your presentation**.

Some hints

You should aim to make your application perform as well as possible by the time you submit it – when doing your performance measurements, you should remove debugging/display code, and compile the project in release mode. It's good practice to use a profiler in order to identify where performance bottlenecks exist.

Here's an example of what we mean by justifying your choice of data structures: all pathfinding algorithms eventually build a data structure to represent the final path. If you chose to implement a pathfinding algorithm that worked by inserting items into the middle of the path as it ran (i.e. the cost of updating the path was a major contributor to the overall performance of the pathfinding process), then a linked list would be an appropriate choice because it's an ordered collection where insertion is O(1) in the length of the path; a vector would be an inappropriate choice because insertion is O(N), which would increase the overall time complexity of the pathfinding algorithm.

“I used arrays because I've used them before” is not an appropriate justification!

When presenting your application's performance, you should follow the best practices for performance measurement and comparison described in the lectures. In particular, you should consider the possible sources of error in the measurements you're taking, and take appropriate measures to avoid them. You should present your results graphically, including the variation as well as the typical value (e.g. use box plots), and you should use appropriate statistics to compare them (do you have a significant difference? what's the effect size?).

Submission

You must submit a ZIP file to the “Project” assignment on MyLearningSpace by **23:59 on Tuesday 5th January 2021**. Your ZIP file must contain the following:

- the complete source code for your application (at least the `.cpp` and `.h` files, for C++);
- a ready-to-run executable for your application (e.g. a Windows `.exe` file).

Please also make sure you've submitted your PDF slides to the “Presentation slides” assignment.

Feedback will be returned on **Tuesday 26th January 2021**.

To reduce the size of your ZIP file, please ensure that you have cleaned out any temporary files from your application's source code before submission – if you've used Visual Studio, then delete any `.obj`, `.ipch` and `.sdf` files, and any files from the profiler. For external libraries or large data files (e.g. password dictionaries), give us a download link rather than including a copy of it.

Grading criteria

This is a **summative** assessment: 80% of your final grade for CMP201 will be determined by your performance at the end of the module as demonstrated in this assessment.

Grade	Implementation quality	Algorithm implementation	Choice of data structures	Performance evaluation
A	High-quality application that operates flawlessly and follows best practice for code quality	Application demonstrates comprehensive understanding of the chosen algorithms, with correct implementations of both	Entirely appropriate argument about choice of data structures for the application, with clear justification of choice in terms of complexity and/or architectural concerns	Outstanding evaluation, considering sources of error and providing appropriate graphical presentation and statistical evaluation of results with reference to expected performance
B	High-quality application, with appropriate structure and only minor flaws in code quality or operation	Application demonstrates good understanding of chosen algorithms with mostly correct implementations of both	Mostly appropriate argument about choice of data structures, with clear justification of choice but some minor weaknesses	Very good evaluation, considering appropriate sources of error and presentating results appropriately
C	Acceptable-quality application (lab-exercise quality) with some flaws in code quality or operation	Application meets all requirements, with some flaws in algorithm implementation	Data structures used are generally reasonable, but argument contains flaws or omissions	Competent performance evaluation, with some mitigation for error and a reasonable presentation of the results
D	Application has substantial flaws in code quality or operation	Meets all requirements but has substantial flaws in implementation of the algorithms	Adequate choice of data structures based on application's requirements, with serious flaws in justification	Adequate performance measurement and evaluation
MF	Unsatisfactory application quality	Fails to meet all requirements	Inappropriate choice of data structures, or inadequate justification of choice	Fails to consider sources of error or provides inadequate presentation of results
F	Performance well below the threshold level, with only limited evidence of achievement			
NS	There is no submission, or the submission contains no relevant material			