COMPSCI2026 Algorithmics

Assessed Exercise – Word Ladder

(2024 - 2025)

Notes for guidance

This is the only assessed practical exercise for Algorithmics. It carries 20% of the total assessment for the course. As a rough guide, it is intended that an average Level 2 Graduate Apprenticeship student should be able to obtain a B grade by putting in about 20 hours work and you are advised not to spend significantly more time than this on the exercise.

The exercise is to be done *individually*. Some discussion of the exercise among members of the class is to be expected, but close working together, or copying of code, in any form, is strictly forbidden – refer to the Plagiarism Policy and Guidelines in the Undergraduate Class Guide.

The use of AI tools for coding is permitted, but you must **explicitly disclose your AI usage** and critically evaluate AI-generated output as outlined in the marking scheme. **AI tools may only be used for assisting with the algorithmic solution and programming aspects of this coursework. AI must not be used to generate any part of the written report, including explanations, justifications, or reflections. All written report content must be your own work.** Failure to disclose AI assistance will be considered a breach of academic integrity. The marking scheme at the end of this document details the assessment criteria for both AI-assisted and non-AI-assisted submissions.

Deadline for submission. The hand-out date for the exercise is **11:00 Tuesday 18 March**, by which time all the relevant material should have been covered in lectures.

The deadline for submission is **22:00 Sunday 6 April**.

There will be a 2 hours lab session on **Wednesday 19 March** with a preparatory exercise on BFS graph traversal. There will be another 2 hour lab session on **Wednesday 26 March**, during which you will have the opportunity to ask questions on the exercise and discuss your progress with the course coordinator and the course tutors.

Specification

The purpose of the exercise is to write Java programs to investigate word ladders composed of five-letter words. A word ladder is a sequence of words where each word differs from its predecessor in exactly one position. For example, the following ladder, of length 6, ‘transforms’ the word *flour* into the word *bread*:

$$flour \rightarrow floor \rightarrow flood \rightarrow blood \rightarrow brood \rightarrow broad \rightarrow bread$$

A dictionary file `words5.txt` is provided, containing nearly 2000 five-letter words that should be used to construct ladders.

Program 1: Finding shortest word ladders. The first program should read in a dictionary file, together with two more five letter words, i.e. the program should take 3 command-line arguments:

1. a dictionary file;
2. a start word;
3. an end word.

The program should produce on the standard output channel the length of the shortest path and a path/ladder of shortest length that transforms the start word into the end word, or should report that no ladder is possible. The final line of output should report the execution time of the program in seconds.

(The code to generate this output is included in the skeleton programs provided. Note that it represents elapsed time, so may not be an accurate reflection of actual running time depending on other processes that may be executing on the computer.)

Program 2: Weighted word ladder using Dijkstra's algorithm. The second program considers a weighted version of the word ladder problem where the weight of a transformation (i.e. edge of the corresponding graph) is the absolute difference in the positions of the alphabet of the non-matching letter. For example, the weight of the edge between *angel* and *anger* equals the position of r minus the position of l which is 6.

This second program should implement Dijkstra's algorithm for finding the shortest paths. Similarly to the first case, the program should read in a dictionary file, together with two more five letter words the program and report on the standard output channel the minimum distance between the words together with a corresponding path, or should report that no ladder is possible. As for the first program, the final line of output should report the execution time of the program in seconds.

Clarifications

- the dictionary file (`words5.txt`) contains only words of 5 letters, all in lower-case, one word per line;
- no data validation is needed: you can assume that input to the first program is provided in the appropriate format, with all words in lower case;
- you can assume also that each word that is input is actually present in the given word file;

- a graph representation of the dictionary is the key to an efficient solution;
- graphs should be represented using adjacency lists and the program from the warm up laboratory exercise provides a very good basis for you programs.

Submission guidelines

The set up files for the exercise are available under Moodle (these include skeleton program files, a template report file, dictionary file of words and test data). (As a starting point, it would make sense to copy across the files for the classes `AdjListNode`, `Vertex` and `Graph` developed in the laboratory exercise.) You may wish to create additional small input files for test purposes.

Submit an `.tar.gz` archive of your work through Moodle. The archive should expand into a directory named after your 7-digit matriculation number. You can create such an archive by using the command:

```
tar cvzf 0123456.tar.gz 0123456/
```

This directory should contain the following.

- A pdf file `report.pdf` generated from the `report.tex` file, containing:
  - a status report, which should state whether you believe that your programs work correctly and if not what happens when the program is compiled (in the case of compile-time errors) or run (in the case of run-time errors or incorrect output);
  - a written discussion justifying your implementation decisions and the points listed in the marking criteria for the "Report, code quality, and presentation" for non-AI usage or the "Report, AI usage, and evaluation" for AI usage.
  - the output produced by your programs for the test data provided.
- Folders `wordladder` and `dijkstra` containing all your `.java` files for the two programs.
- In each folder there should be a class `Main.java` containing your main method; apart from this there can be any number of other `.java` files and other folders corresponding to packages if you wish. But ensure that any redundant files are removed. Both programs **must compile** from the command line when using the command `javac Main.java`.

Please make sure you follow the submission instructions - the penalty for non-adherence to Submission Instructions is 2 bands.

Marking Scheme (30 Marks, mapped to a band)

You must indicate whether you used AI tools for code generation or assistance. The marking scheme is adjusted accordingly. If AI was used only for debugging or for a specific part of the code, you must still provide the required AI disclosure and evaluation, explaining precisely how AI was used and how you verified or improved upon AI's suggestions. AI must not be used to generate any part of the written report, including explanations, justifications, or reflections

If AI is <u>NOT</u> used

- **Word ladder implementation (program 1)**: 10 marks (correctness, efficiency, and adherence to specifications).
- **Dijkstra's Shortest path algorithm (program 2)**: 10 marks (proper implementation of Dijkstra's algorithm for weighted word ladders).
- **Report, code quality, and presentation**: 8 marks in total broken down as follows:
  - **Code readability and structure** (3 marks): appropriate data structures, modularity, clarity, and proper commenting.
  - **Algorithm design explanation** (3 marks): justification of design choices, explanation of algorithmic approach, and consideration of alternatives.
  - **Debugging and problem-solving process** (2 marks): explanation of challenges faced, how they were resolved, and insights from debugging.
- **Outputs from test data**: 2 marks (correctness verification with the provided test cases).

If AI is used

- **Correctness of code (both programs)**: 10 marks (the solution must work correctly, but correctness alone is not enough).
- **Report, AI usage, and evaluation**: 18 marks in total broken down as follows:
  - **AI disclosure and summary** (3 marks): Clear statement of AI use (which parts were AI-generated, how it was used, which AI tools were used), and brief summary explaining how AI output was verified or modified.
  - **Code readability and structure** (3 marks): Ensure modularity, clarity, proper commenting, and appropriate data structures.
  - **Algorithm design explanation** (4 marks): Justify algorithmic choices, explain AI's design, and consider alternative approaches. Consider aspects such as:
    * Efficiency: Did you make any optimisations or was AI's solution already optimal? If unchanged, justify why.
    * Data structure selection: Was AI's choice appropriate or could a better one be used?
    * Readability and maintainability: Was AI's output clean or did it require you to restructure it?

    - ∗ Handling of edge cases: Did AI correctly account for special/edge cases or were fixes needed?
  - **AI errors and fixes <u>OR</u> AI solution analysis** (4 marks):
    - ∗ If AI-generated code contained errors, identify and fix at least one mistake or inefficiency, with explanation.
    - ∗ If AI-generated code was correct, analyse AI's design choices, potential inefficiencies, and possible improvements.
    - ∗ If AI was used only for debugging, explain the specific issue AI helped resolve and how the final solution was verified.
  - **Algorithm understanding and reflection** (4 marks):
    - ∗ Demonstrate understanding of the underlying algorithm, despite AI assistance.
    - ∗ Answer the reflection question: *If you had to implement this algorithm without AI, how would you structure it? What would be the most challenging part?*

- **Outputs from test data**: 2 marks (correctness verification with the provided test cases).

This is primarily an Algorithmics exercise, rather than a Software Engineering exercise, but you may be penalised, under the code readability and structure criteria above, for poor software engineering practice.