Algorithms: Assessed Exercise

2366908I

November 2020

1 Preamble

A key part of Computer Science history is the development of increasingly efficient algorithms. In doing so, we guarantee the processes that rely on executing said algorithm to perform in a less time-costly manner. This paper will go over a project that implements two algorithms to solve a problem: creating a Word Ladder. The premise of Word Ladder is as follows. First, one is provided with two words: a **starting** word and an **ending** word.

The words provided are assumed to be of the same size.

One must, from the **starting** word, form a sequence of words that reach the **ending** word. The sequence must be such that any two successive pairs of words can not differ by more than one corresponding character. Consider the following:

flood → **blood**
flour → **blood**

Note, the first example is legal, because only the corresponding letters in the first position of each string differ. In the second example, the words differ in the first, fourth and fifth positions of each string, which is illegal.

This paper will examine a Java program, which when provided with a starting and ending word and a dictionary of 5-letter words, will attempt to construct a word ladder between them. The result is printed to standard output. An example of a possible call to the program is as follows:

```
              java Main words5.txt flour bread
      java Main <words_dictionary> <start_word> <end_word>
```

The program is of two variations: a Breadth-First Search implementation and a Dijkstra's implementation. This paper will evaluate both and the reasoning behind the approach taken to implement each.

2 Implementations

To compile: javac Main.java. Issues may arise if your java version is more recent. In this event, use: javac -target 8 -source 8 Main.java

2.1 Program I: Breadth-First-Search

2.1.1 How it operates

The algorithm works as follows:

1. Initialize a queue, which will hold unvisited vertices.
2. Start at a vertex and push to the queue.
3. Visit the unvisited (adjacent) vertices and add them to the queue.
4. Now the first vertex added to the queue has been visited, since we have added all of its unvisited neighbours to the queue, so we can remove it from the queue.
5. Repeat the above process for the remainder of the graph.

2.1.2 State of the solution

The Breadth-First-Search implementation works correctly, finding the shortest sequence of words given a pair of words for which a ladder is in fact possible.

2.1.3 Implementation decisions

The warm-up lab solution provided good grounds for the BFS implementation particularly concerning the crux of the BFS' logic. The differences in implementations and the respective rationales are explained below:

- **Search scope:** The BFS does not span every vertex in the graph, and instead starts at the vertex corresponding to the start word in *wordList*, a Java ArrayList containing all words read from the dictionary file. The vertex is added to queue at the beginning and set as visited. It is not necessary to start at a non-starting-word vertex because we traverse vertices that are of no interest, thus causing inefficiency. Additionally, ultimately, if we start at some other point we will eventually have to arrive at the starting word's vertex, thus wasting resources.
- **Visiting adjacent nodes:** When visiting adjacent nodes, a check for the ending-word's index is done. If so, since we are updated the predecessors of the nodes throughout the BFS, then this will be the first (shortest) path to reach the target (end) word, beginning from the start word. Additionally, from here we can iteratively call the predecessors until we reach the start word to print the word ladder to standard output and return so that no

further resources are used. If we never reach the target (end) word, then it follows that it is not possible for the given start word.

2.1.4 Program output

Note that the following test cases are performed on a dictionary of size 1638, comprising solely 5-letter words. It is safely assumed that only 5-letter inputs are provided, thus there is no streamlined error handling for such illegal cases.

```
      java Main words5.txt print paint
                 print
                 paint
             Word count: 2
          Elapsed time: 108 ms
      --------------------------------
      java Main words5.txt forty fifty
                 fifty
                 fifth
                 firth
                 forth
                 forty
             Word count: 5
          Elapsed time: 124 ms
      --------------------------------
      java Main words5.txt cheat solve
                 solve
                 salve
                 halve
                 helve
                 heave
                 leave
                 lease
                 cease
                 chase
                 chasm
                 charm
                 chart
                 chert
                 cheat
             Word count: 14
          Elapsed time: 121 ms
      --------------------------------
      java Main words5.txt worry happy
Word ladder not possible for worry and happy
          Elapsed time: 106 ms
      --------------------------------
```

```
java Main words5.txt smile frown
            frown
            crown
            croon
            crook
            crock
            clock
            click
            slick
            slice
            spice
            spite
            smite
            smile
        Word count: 13
     Elapsed time: 107 ms
--------------------------------
java Main words5.txt small large
            large
            marge
            merge
            verge
            verse
            terse
            tease
            cease
            chase
            chasm
            charm
            chard
            shard
            share
            shale
            shall
            small
        Word count: 17
     Elapsed time: 141 ms
--------------------------------
java Main words5.txt black white
            white
            whine
            thine
            trine
            brine
            brink
```

```
                        blink
                        blank
                        black
                    Word count: 9
                 Elapsed time: 111 ms
            --------------------------------
            java Main words5.txt greed money
      Word ladder not possible for greed and money
                 Elapsed time: 106 ms
```

2.2 Program II: Dijkstra's Algorithm

2.2.1 How it operates

Start with a graph of vertices that are, by default, unvisited. Begin with a vertex, in this case the one corresponding to the start word, and and set it to visited. Push to the queue, indicating it is yet to be processed, i.e: neighbours are still unvisited. Then visit all unvisited vertices and remove the vertex from the queue. Repeat the process, starting with the vertex closest to the vertex preceding.

2.2.2 State of the solution

The program works as intended, returning the shortest path between two words, provided a path does exist.

2.2.3 Implementation decisions

- **MAX VALUE:** The program utilizes the integer class' max value field in order to describe the distance of a vertex as unknown (which changes throughout the life of the algorithm), or in the case of the end word vertex, impossible to reach via the starting word, thus stating a path does not exist.
- **Vertex processing:** Initially the start word's vertex is pushed to the queue. Then a while block is entered and maintain while there is still a vertex in the queue. In this block, all non-visited neighbour vertices are pushed to the queue and there distances updated if a shorter one is found via the current vertex. Then the current vertex is set to visited, and removed from the queue as it is processed. If the adjacent node being added to the unvisited queue corresponds to the ending word's index, then we have our path, and there is no need to waste resources with further processing; return the ladder and return from the method.

2.2.4 Program output

```
            java Main words5.txt blare blase
                        blase
                        blare
```

```
   Total weight of path: 1
     Elapsed time: 120 ms
--------------------------------
java Main words5.txt blond blood
            blood
            blond
   Total weight of path: 1
     Elapsed time: 141 ms
--------------------------------
java Main words5.txt allow alloy
            alloy
            allow
   Total weight of path: 2
     Elapsed time: 110 ms
--------------------------------
java Main words5.txt cheat solve
            solve
            salve
            halve
            helve
            heave
            leave
            lease
            cease
            chase
            chasm
            charm
            chart
            chert
            cheat
   Total weight of path: 96
     Elapsed time: 135 ms
--------------------------------
java Main words5.txt worry happy
Given words do not have a word ladder
     Elapsed time: 118 ms
--------------------------------
java Main words5.txt print paint
            paint
            print
   Total weight of path: 17
     Elapsed time: 146 ms
--------------------------------
java Main words5.txt small large
            large
            marge
            merge
            verge
            verse
            terse
            tease
            cease
            chase
            chasm
            charm
            chard
            shard
            share
            shale
            shall
            small
   Total weight of path: 118
     Elapsed time: 152 ms
--------------------------------
java Main words5.txt print paint
            white
            whine
            thine
            thane
            thank
            shank
            shack
            slack
            black
   Total weight of path: 56
     Elapsed time: 124 ms
--------------------------------
java Main words5.txt greed money
Given words do not have a word ladder
     Elapsed time: 122 ms
```