Algorithmics I – Assessed Exercise

Status and Implementation Reports

Adam Wong
2549779W

November 18, 2022

Status report

I believe that my implementation of both programs work correctly. When using the test data, both programs successfully output a word ladder from one word to another, where a path is physically possible. The first word ladder program correctly shows the path length. The weighted Dijkstra implementation correctly shows the weighted path distance.

They also do this quite efficiently. Both programs usually take between 200-300ms to run on my local machine.

Implementation report

(a) For the first program, I turned the list of words into an adjacency matrix, utilising and modifying: **Graph.java**, **AdjListNode.java**, and **Vertex.java** from the previous lab to implement this structure. I used Breadth-First-Search (BFS) to search through this matrix, to find the shortest path from the starting word to the target word. Tracing back using predecessors from the end word to the start results in the shortest path.

I chose BFS for this as it the breadth-first approach naturally produces the shortest path from a to b. Comparing this to other traversal algorithms e.g. Depth-First-Search, which would likely find a longer path first, I found BFS to be most efficient.

The files I provide for this implementation are in the **wordladder** folder. Within this folder there are:

- Main.java file
- worldLadderPack folder, which contains:
  - Graph.java
  - AdjListNode.java
  - Vertex.java

Running **javac Main.java** should successfully compile all this code.

(b) I implemented Dijkstra's algorithm in a slightly different way. I used a buckets data structure to gather adjacent words - giving each word a bucket and collecting its adjacents into them.

This is utilised later, rather than populating every vertex in the matrix with its edge weight to every other vertex, to efficiently traverse through the adjacency matrix.

I used a priority queue in my implementation of Dijkstra's algorithm. This ensures that each vertex is processed only once, thus resulting in greater efficiency

I had to create a Comparator class for Vertex so that my priority queue would always remove the Vertex with the minimum distance from the source word.

The files I provide for this implementation are in the **dijkstra** folder. Within this folder there are:

- Main.java file
- dijkstraPack folder, which contains:
  - Graph.java
  - AdjListNode.java
  - Vertex.java
  - VertexComparator.java

Running **javac Main.java** should successfully compile all this code.

Empirical results

Here are some example outputs from the first program using the test data:

```
word1 = forty
word2 = fifty
Shortest word ladder length: 4
Example of shortest word ladder:
forty
forth
firth
fifth
fifty

Elapsed time: 291 milliseconds
```

and where no path is found:

```
word1 = greed
word2 = money
No word ladder exists.

Elapsed time: 298 milliseconds
```

Here are some example outputs from the second Dijkstra's algorithm program:

```
Start Word: black
End Word: white
Min path length: 56
```

```
black
slack
shack
shank
thank
thane
thine
whine
white

Elapsed time: 220 milliseconds
```

and where no path is found:

```
Start Word: worry
End Word: happy
No word ladder exists.

Elapsed time: 209 milliseconds
```