Algorithmics 3 Assessed Exercise

Status and Implementation Reports

Rafal Ciesielczuk
1104712c

February 8, 2014

Status report

The first program I had to create was a Word Ladder. The program produces the length of the shortest path and a path/ladder of shortest length that transforms the start word into the end word. If a ladder doesn't exist, then the program outputs that there is no possible ladder. In addition, the program outputs the total execution time in seconds and milliseconds. I believe this program is working as it's supposed to be working. This is based on the results that I've gotten, which are located in the **Empirical results** section.

The second program considers a weighted version of the word ladder problem. The program implements Dijkstra's algorithm for finding the shortest paths. Unfortunately, this program doesn't output what it's supposed to. When the program is compiled, it returns the words for each path (which, some of them are really close to the ideal one). In addition, it returns the weight of each path, which I am unsure if they are correct. I believe, the the most important thing, is that the Dijkstra's algorithm returns whether a ladder between the two words exists. In this case, this function works. The results are located in the **Empirical results** section.

Implementation report

(a) The implementation of the Dijkstra's algorithm was quite a diffuclt task, mainly because it didn't return all the necessary results. It was mainly based off of the modified BFS algorithm. The dijkstra's algorithm returned whether a ladder exists between the two given words supplied in the command line. This function was implemented in the Graph.java class in order to reduce scatter in the Main.java class. Since this aglorithm took into account a weighted version of the word ladder, I had to calculate the weight of each AdjListNode and then output it. The algorithm finds the shortest path between that vertex and every other vertex. It's supposed

to print out the correct and path weight for each of the path, but there seems to be a bug somewhere.

(b) For the backtrack search function, I've modified the given BFS algorithm from the previous lab excercise and implemented it in the Graph.java class. I did that, because I wanted to avoid scatter in the Main.java class.

What I've done, is that I've created an ArrayList based on the two parameters (word1 and word2) as vertices, which returns the arraylist of the correct word ladder, as well as the distance. The function carries out the the BFS from the first vertex and then resets everything. Then, the function sets up the queue, and starts BFS from the vertex. When visiting a new vertex for the first time, the distance gets assigned to be 1 + the distance of its predecessor.

In addition, there is an implemented backtrace function, which is a recursive function right when the vertex' word is equal to the last word.

Empirical results

Word Ladder

(a) Sample Output for a working solution:

(forty) managed to make it to (fifty)!

Ladder: forty - forth - firth - fifth - fifty

Ladder Distance: 4

Elapsed time: 2 seconds

Elapsed time: 1673 milliseconds

(b) Sample Output for a non-working ladder solution:

There is no ladder for: (money) to (greed)!

Elapsed time: 2 seconds

Elapsed time: 1629 milliseconds

Results:

(a) **print - paint**

Elapsed time: 1 seconds

Elapsed time: 1655 milliseconds

(b) **forty - fifty**

Elapsed time: 1 seconds

Elapsed time: 1705 milliseconds

(c) **cheat - solve**

Elapsed time: 2 seconds

Elapsed time: 1643 milliseconds

(d) **worry - happy**

Elapsed time: 2 seconds

Elapsed time: 1646 milliseconds

(d) **smile - frown**

Elapsed time: 1 seconds

Elapsed time: 1676 milliseconds

(e) **small - large**

Elapsed time: 2 seconds

Elapsed time: 1655 milliseconds

(f) **black - white**

Elapsed time: 1 seconds

Elapsed time: 1843 milliseconds

(f) **greed - money** - no ladder

Elapsed time: 2 seconds

Elapsed time: 1697 milliseconds

Dijkstra's Algorithm

(a) Sample Output for 'semi' working solution:

(forty) managed to make it to (fifty)!

Words: foray Words: forte Words: forth — Weight for this path: 17

Words: forty — Weight for this path: 19

Words: forty Words: forth Words: force Words: forge — Weight for this path: 13

Words: worth Words: north Words: forty Words: firth Words: forte — Weight for this path: 3

Words: forte Words: forge Words: farce — Weight for this path: 14

Words: gorge Words: forte Words: forgo Words: force — Weight for this path: 4

Words: forth Words: north — Weight for this path: 9

Words: worth Words: forth — Weight for this path: 8

Words: mirth Words: girth Words: forth Words: birth Words: fifth Words: filth — Weight for this path: 6

Words: force — Weight for this path: 14

Words: gouge Words: gorse Words: forge — Weight for this path: 1

Words: forge — Weight for this path: 10

Words: birth Words: firth Words: girth — Weight for this path: 6

Words: mirth Words: birth Words: firth — Weight for this path: 1

Words: mirth Words: girth Words: firth Words: berth Words: birch — Weight for this path: 17

Words: firth Words: filth

Elapsed time: 1 seconds Elapsed time: 1651 milliseconds

Results:
Results:

(a) **blare - blase**

Elapsed time: 2 seconds

Elapsed time: 1700 milliseconds

(b) **blond - blood**

Elapsed time: 1 seconds

Elapsed time: 1696 milliseconds

(c) **allow - alloy**

Elapsed time: 2 seconds

Elapsed time: 1701 milliseconds

(d) **cheat - solve**

Elapsed time: 2 seconds

Elapsed time: 1784 milliseconds

(e) **worry - happy**

Elapsed time: 1 seconds

Elapsed time: 1695 milliseconds

(f) **print - paint**

Elapsed time: 2 seconds

Elapsed time: 1705 milliseconds

(g) **small - large**

Elapsed time: 2 seconds

Elapsed time: 1795 milliseconds

(h) **black - white**

Elapsed time: 1 seconds Elapsed time: 1723 milliseconds

(i) **greed - money**

Elapsed time: 2 seconds Elapsed time: 1771 milliseconds