Wesly Lim

Rally Health Coding Challenge:

Setup Procedures:

Before doing any work on the algorithm, the text from text files must be parsed and organized. The function *readFile(filename)* takes in a file that is correctly formatted with the following criteria:

INTEGER INTEGER INTEGER INTEGER
START_STRING
END_STRING

The integers not be negative, or else they will loop forever to find a new minimum. There must also be exactly four integers. Error handling is done to make sure all of the integers are correctly formatted.

The string case is ignored, and must be a single word not separated by spaces. The words must also be in the dictionary. If there are too many lines on the text file, the program will also return -1. If there are any errors in input for strings, -1 will automatically be returned.

Once successfully parsed and error-checked, the weights are stored in a list and the start/end strings are stored in variables. The weights are then distributed in the word list dictionary. This is all done in *readFile(filename)*, where *dijkstra_shortest_path(graph, start, end)* is also called.

Algorithm:

For this problem, assume that a dictionary with the key as a string on a word list and the value as another dictionary containing related words as the key and weights as the value is provided (this is valid because the prompt allows the use of any word list). The dictionary would then be in the form of the following:

{ …
string: { path1: ins_weight, path2: del_weight, path3 : change_weight, path4: ana_weight },
… }

The next step is to perform Dijkstra's shortest path algorithm on the constructed weighted graph (word list). The inputs are four integers as weights separated by a space, a start string, and an end string which are all separated by newlines. A running dictionary of distances of words from the start is kept in track in conjunction, which is standard for Dijkstra's algorithm. The final smallest cost is returned by retrieving it from the dictionary of distances.

If there does not exist a path from one string to another within the graph, -1 is returned. Also, if the start and end strings are the same, -1 is also returned to prevent any cycling.

Proof:

For this proof, the same explanation is used in Dijkstra's algorithm. This is the fastest solution for finding weighted shortest path algorithms for directed graphs. The weights are fixed with the inputs in the text file, which is then inserted in the graph. It is known that this shortest path algorithm is correct; this program uses the same idea, but modifies a few variables. This can be found here: https://web.engr.oregonstate.edu/~glencora/wiki/uploads/dijkstra-proof.pdf

This is guaranteed to find the shortest path from the start string to the end string.

Time Complexity and Comparisons:

Dijkstra's shortest path algorithm runs in $O(|E| + |V|log|V|)$ time where E is the number of edges and V is the number of vertices. In this program's case, E is the number of connections there are for every word in the word list and V is the number of words in the word list. The space the dictionary takes is fixed because the word list is imported.

This method is chosen over the typical dynamic programming approach (the minimum edit distance) used for most Levenshtein distance problems because every interim step must also be a word in the word list. Usually this method takes $O(M * N)$ time to complete, where M, N are lengths of each string. However, each interim step in this method may not be a valid string in the word list. This makes this strategy invalid to use and the shortest path algorithm more favorable.

Every other method is uncontestable and therefore not considered in time complexity.

Technologies/How to Run Code:

I used Python 2 to code this project, and Sublime Text as my text editor. The code is ran on console with the following:

```
python rally.py
```

All text files must also be within the same directory if all unit tests want to be used as well. If not, remove the assert clauses at the bottom. The tests at the bottom tests all basic and edge cases of the code.