Word Ladder Assignment

Zac Cripps s5133929, Thomas Sloman s5129352

Contents

Problem Statement

For this task we have already been given the Python program. However, the program has been edited resulting in it no longer working as efficiently as the original source code. The goal of 2810ICT assignment part 1 was to transform a given start word into the other given target word with the least number of steps required. Through each step one letter is only changed without changing positions of the other letters resulting in creating a new word. Each resulting word must also exist in a supplied dictionary file.

The program may accept a list of words in a single command, validate these words exist in the dictionary then append into a larger list to be used as a blacklist. The black list contains the words that must not appear in the pathing for each step when transforming the words. Additionally, an intermediate word can be supplied by the user and must appear on the path toward the target word. The intermediate word must also exist in the dictionary file.

User Requirements

The following outlines the user requirements for the program:

- The user shall be able to enter a dictionary name to be used for the program.
- The user shall be able to enter one word for both start and target word presented on two different lines.
- User can enter a number of words on the same line separated by whitespace when prompted to enter blacklisted words.
- Use can enter a word that should appear in the path towards the target word.
- All words should be of equal length.
- Words entered should be all alphanumeric with no special characters.
- Start word cannot equal the target word.

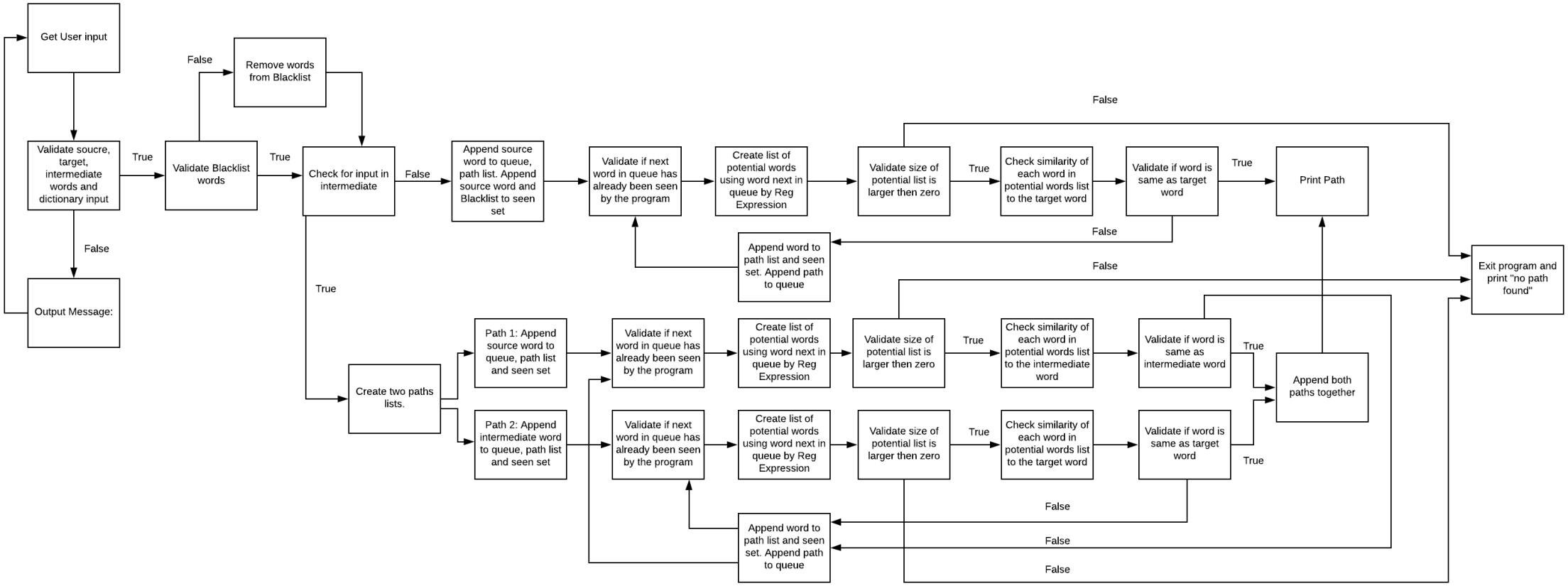
Software Requirements

The following outlines the software requirements for the program:

- The program shall accept only one input for dictionary file name, start, target, and intermediate word.
- The program shall accept multiple words on the same line for the blacklist.
- Program must validate that all words belong in the dictionary.txt file.
- Program must validate that all words entered have the same length.
- If blacklisted words are invalid they are individually removed from the list.
- The program shall display a message if any of the above inputs entered are wrong.
- The program should run using a dictionary file ending with the extension '.txt'.
- If all inputs are valid then execute the getPath() function.
- If no inputs are entered for blacklist and intermediate word, program shall still execute appropriately.
- Program shall display list containing the start word and all the steps taken to reach the target word.
- Program shall display a number next to the path list that displays the number of steps taken to reach target word.
- If no path is found the program shall display 'No path found'.

Software Design

High Level Design – Logical Block Diagram



Functions

There are four functions used in the program. ‘getPath’ is used as the entry point for the process of finding the best path based on the user’s inputs. ‘findPath’ uses a queue like structure to effectively iterate through many possible paths until one is found. ‘getNeighbours’ and ‘getSimilarity’ are both utility functions needed to perform different types of string comparisons.

- getPath(startWord, targetWord, blacklistWords, intermediateWord, viableWords)

This function initialises ‘searchQueue’ and ‘seenWords’ based on the user’s inputs, before calling findPath with these values until a path is found or there is no possible path.

- ‘startWord’ is the string entered by the user as the starting point of the path.
- ‘targetWord’ is a string entered by the user as the goal for the path to reach.
- ‘blacklistWords’ is a list of strings the algorithm must not use in a path from
- ‘startWord’ to ‘targetWord’.
- ‘intermediateWord’ is a string that must appear in the path.
- ‘viableWords’ is a list of strings containing all possible words that can appear in the path.

The returned value provided by this function is either the best path from ‘startWord’ to ‘targetWord’ as a list of strings or None if no path could be found.

- findPath(targetWord, queue, seen, viableWords)

This function finds all possible next steps in a path from the path provided in the 1st position in queue. It then adds these next steps to individual copies of this path and places them onto the end of the queue.

- ‘targetWord’ is the string entered by the user as the goal for the path to reach.
- ‘queue’ is a list of lists containing strings. The lists that this holds are all paths that have been found that may need to be checked by this function to find a complete path.
- ‘seen’ is a set of strings containing words that have already been found previously and should be ignored.
- ‘viableWords’ is a list of strings containing all possible words that can appear in the path.

The ‘queue’ and ‘seen’ input parameters are references to the variables ‘searchQueue’ and ‘seenWords’ from the function ‘getPath’ that may be modified by this function.

This function returns a path if one is found in the form of a list of strings, or None should there be no more possible steps to take from the path it got from the queue.

- getNeighbours(pattern, viableWords, seenWords, potentialWords)

This function gets all words from a list which match a regex pattern, where they have not already been included in ‘seenWords’ or ‘potentialWords’.

- ‘pattern’ is the pattern for comparing words which must be a string,
- ‘viableWords’ must be a list of strings that may be included in the result.
- ‘seenWords’ is a list of strings to be excluded from the result.
- ‘potentialWords’ is also a list of strings to be excluded from the result.

This function returns a list of strings containing all words matching the requirements provided.

- getSimilarity(word, targetWord)

This function compares each letter in ‘word’ and ‘targetWord’. If the letters are the same it adds a ‘1’ to a list in that letter’s position, otherwise it adds a ‘0’. Finally, the list is summed and that sum is returned.

- ‘Word’ is a string to be compared to ‘targetWord’.

- ‘targetWord’ is a string to be compared against.

The return value of this function is a number that is a whole integer representing the similarity of ‘word’ and ‘targetWord’.

Data Structures

The program uses both set and list data structures.

- Set

  A set is used to hold data that should not be iterated over but need to be queried for inclusivity.

  The sets used in the program are:

  - ‘seenWords’ holds words that have already been found and added to paths in the queue so that they are not added if they are found later..
  - ‘blacklistWords’ holds words that should not be added to paths if they are found when searching for a path

  Data members included in sets are:

  - ‘string’ holds characters in order.

  Functions in the program that use sets are:

  - getNeighbours()
  - findPath()
  - getPath()

- List

  Lists are used where many values must be iterated over.

  The lists in the program are:

  - ‘viableWords’ holds word that were found in the dictionary with the same length as the start word.
  - ‘searchQueue’ holds paths to be checked in findPath().
  - ‘path’ and ‘bestPath’ holds words in order of changes from start word.
  - ‘potentialWords’ holds words that could be added to the end of potential paths as the next step.

  Data members included in lists are:

  - ‘string’ holds characters in order.
  - ‘list’ holds any type in order.

  Functions in the program that use lists are:

  - getSimilarity()
  - getNeighbours()
  - findPath()
  - getPath()

Detailed Design

Function getSimilarity (word, targetWord) {

- Return sum of letters in word matching letters in targetWord

}

Function getNeighbours (pattern, viableWords, seenWords, potentialWords) {

- Return list of words from viableWords matching pattern that aren’t in seenWords or potentialWords

}

Function findPath (targetWord, queue, seen, viableWords) {

- Set word to last string from first list in queue
- Set path to copy of first list in queue
- Set potentialWords to empty list

For i in 0 to word length {

- Add to potentialWords result from getNeighbours with pattern being word with '.' in 'i'th position

}

If (potentialWords length is 0) {

- Return None

}

For word in potentialWords {

- Set match to getSimilarity of word and targetWord

If (match greater than or equal to target word length - 1) {

If (match equal to target word length - 1) {

- Append word to path

}

- Append targetWord to path
- Return Path

}

- Add word to seen
- Add word to path
- Add path to queue
- Remove word from path

}

}

Function getPath (startWord, targetWord, blacklistWords, intermediateWord, viableWords) {

- Set searchQueue to an empty list
- Create a list containing startWord, then append it to searchQueue
- Set seenWords to set of list of startWord
- Update seenWords to include blacklistWords
- Set bestPath to None

If (intermediateWord exists) {

- Add targetWord to seenWords

While searchQueue not empty and bestPath is None {

- Set bestPath to findPath
- Remove first list from searchQueue

}

- Set searchQueue to list of list of intermediateWord
- Set seenWords to set of list of intermediateWord
- Update seenWords to include blacklistWords
- Update seenWords to include bestPath
- Set secondPath to None

While searchQueue not empty and secondPath is None {

- Set secondPath to findPath
- Remove first list from searchQueue

}

- Add secondPath to bestPath

} else {

While searchQueue not empty and bestPath is None {
- Set bestPath to findPath
- Remove first list from searchQueue

}
}
- Return bestPath

}

While True {
- Get input dictionaryFile, strip
  If (dictionaryFile empty string or doesn't exist or empty file) {
  - Print error, continue

  }
- Get input startWord, strip, lower
  If (startWord length less than 1) {
  - Print error, continue

  }
- Set viableWords to empty list
  With dictionaryFile open as wordsFile {
  For wordsFileLine in wordsFile {
  - Set word to wordsFileLine, rstrip, lower
    If (word length equal to startWord length) {
    - Append word to viableWords

    }

  }
  }
  If (startWord not in viableWords) {
  - Print error, continue

  }
- Get input targetWord, strip, lower
  If (targetWord length not equal to startWord length
  or targetWord same as startWord
  or targetWord not in viableWords) {
  - Print error, continue

  }
- Get input intermediateWord, strip, lower
  If (intermediateWord length greater than 0) {
  If (intermediateWord length not equal to startWord length
  or intermediateWord same as targetWord
  or intermediateWord same as startWord
  or intermediateWord not in viableWords) {
  - Print error, continue

  }
  }
- Get input blackListWords , lower, split
  For word in blackListWords {
  If (word length not equal to startWord length
  or word not in viableWords
  or word same as startWord
  or word same as targetWord
  or word same as intermediateWord) {
  - Remove word from blacklistWords

- Print error message

}

}

- Set blacklistWords to a set
- Break

}

- Set path to getPath

If (path is not None) {

- Print path length - 1, path

} Else {

- Print “No path found.”

}

Configuration Management and Version Control

The version control used for this project was Github over the recommended version control, Bitbucket. This decision was made based on the existing knowledge of the version control and the features available on both software, which include allowing for users to have private collaboration and record all logs (commits) when interacting with the code for documentation. Development of the project was mostly done through using an extension available through Visual Studio Code called “Live Share”. This allowed for both members to simultaneously work on the source code with the initiator of the live collaboration committing and pushing the file to the central respiratory. Smaller implementations and changes made to the code were done separately in our own time and pushed to the repository with alerts made to either members about changes being applied to the source code. During the course of working on this project no conflicts of different branches from the repository being worked on occurred. For Git log please see Appendix 1.

Unit Tests

No	Test Case	Expected Results	Actual Results
1.0	Main Function		
1.1	Test normal case.	Return correct path.	Return correct path.
1.2	Test impossible case.	Return ‘No path found’.	Return ‘No path found’.
1.3	Test an empty start word.	Display error message and exit returning None.	Display error message and exit returning None.
1.4	Test start word not in dictionary.	Display error message and exit returning None.	Display error message and exit returning None.
1.5	Test an empty start word.	Display error message and exit returning None.	Display error message and exit returning None.
1.6	Test start word not in dictionary.	Display error message and exit returning None.	Display error message and exit returning None.
1.7	Test same start and target word.	Display error message and exit returning None.	Display error message and exit returning None.
2.0	Main Function - Dictionary		
2.1	Test empty dictionary.	Display error message and exit returning None.	Display error message and exit returning None.
2.2	Test bad dictionary.	Display error message and exit returning None.	Display error message and exit returning None.
3.0	Main Function - Intermediate		
3.1	Test normal with intermediate.	Display correct path and steps.	Display correct path and steps.
3.2	Test intermediate not in dictionary.	Display error message and exit returning None.	Display error message and exit returning None.
3.3	Test intermediate same as start word.	Display error message and exit returning None.	Display error message and exit returning None.
3.4	Test intermediate wrong length.	Display error message and exit returning None.	Display error message and exit returning None.
4.0	Main Function - Blacklist		
4.1	Test normal with blacklist.	Display error message and exit returning None.	Display error message and exit returning None.
4.2	Test blacklist word not in dictionary.	Display error message and exit returning None.	Display error message and exit returning None.
4.3	Test blacklist word same as start word.	Display error message and exit returning None.	Display error message and exit returning None.

4.4	Test blacklist word same as target word.	Display error message and exit returning None.	Display error message and exit returning None.
4.5	Test blacklist word same as intermediate word.	Display error message and exit returning None.	Display error message and exit returning None.
4.6	Test blacklist word wrong length.	Display error message and exit returning None.	Display error message and exit returning None.
5.0	GetPath Function		
5.1	Test normal.	Return correct path.	Return correct path.
6.0	FindPath Function		
6.1	Test normal.	Return correct path.	Return correct path.
6.2	Test no viable words.	Return None.	Return None.
7.0	GetSimilarity Function		
7.1	Test normal.	Return number of identical letters.	Return number of identical letters.
8.0	GetNeighbours Function		
8.1	Test normal.	Return list of words matching pattern that aren’t in seenWords, potentialWords.	Return list of words matching pattern that aren’t in seenWords, potentialWords.

Requirement Acceptance Test

Software Requirement No	Test	Implemented (Full /Partial/ None)	Test Results (Pass/ Fail)	Comments (for partial implementation or failed test results)
1	Accept dictionary file name with extension being .txt.	**Full**	**Pass**	N/A
2	Accept Start, Target and Intermediate word.	**Full**	**Pass**	N/A
3	Accept multiple words on the same line for blacklist.	**Full**	**Pass**	N/A
4	Display statement describing situation if input is invalid.	**Full**	**Pass**	N/A
5	Allow user to renter words if start, target or intermediate words are invalid.	**Full**	**Pass**	N/A
6	Accept no inputs for blacklist and intermediate word, program shall still execute appropriately.	**Full**	**Pass**	N/A
7	All words must be of the same length, alphanumeric, not equal to each other and exist in dictionary file.	**Full**	**Pass**	N/A
8	Display list of shortest path taken to reach from start word to target word.	**Full**	**Pass**	N/A
9	Display intermediate word if exists within the path list.	**Full**	**Pass**	N/A
10	Display number of steps taken to reach target word adjacent to path list.	**Full**	**Pass**	N/A

User Instructions

- Open program within an IDE or terminal that supports the python programming language.
- Run the file “word_lader.py” in the IDE or terminal.
- Enter a dictionary file name into terminal.
- Enter start word that exists in the selected dictionary file into the terminal.
- Enter target word that exists in the selected dictionary file into the terminal.
- (Optional) Enter an intermediate word that exists in the selected dictionary file into the terminal.
- (Optional) Enter blacklisted words separated by whitespace that exist in the selected dictionary file into the terminal.
- For words that are not valid, an error message will display asking the user to re-enter values from the start or continue with the program with changes made to the inputs to allow for continuation.

Appendix

1. Git Log

commit ba232892bdd49120100f773f959d83498301f04e
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Sun Aug 25 10:58:40 2019 +1000

Final version

commit ffef7b1cde851d8c1579e86d0368a53aaabe55ca
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Fri Aug 23 12:25:56 2019 +1000

Complete input tests, repo clean up

commit 3bc318dc1cbee85cd0c01cd71bab142bc4c16de9
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Thu Aug 22 19:10:46 2019 +1000

Added more tests, split into functions

commit 718b67ef8cd402fc792360bf8030b1a3bc051e57
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Thu Aug 22 18:39:50 2019 +1000

Test file clean up

commit fa1af25d6a9138106e75a196d91d671c0ca7b481
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Thu Aug 22 10:57:05 2019 +1000

More test coverage

commit 87f744a380dd9a280f6c837104cd118596f8eb87
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Tue Aug 20 22:04:24 2019 +1000

Added new functions to test, updated testable code

commit 9f08944a4b49152498d6e057a959dd3ec619d1e4
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Tue Aug 20 21:26:45 2019 +1000

Reorganised and expanded validation

commit 7309eab9367736d1dd3ced4f375dc5788e183a48
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Tue Aug 20 13:37:29 2019 +1000

Created testable code file

commit b9ef29f36696813c61c588d6dabd1a0bca7f3e41
Author: Zac <zachary.cripps@griffithuni.edu.au>
Date: Fri Aug 16 14:32:23 2019 +1000

Testing input values

commit 47bb5af99423c211935096e6d4c8f022ba7ea2b1
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Thu Aug 15 14:43:32 2019 +1000
    Testing set up and error handling
commit 54bc6f8723bcca21660a0885e9807bea62591c0b
Author: Zac <zachary.cripps@griffithuni.edu.au>
Date: Thu Aug 15 10:55:27 2019 +1000
    Error handling for inputs
commit b7d7f9953c3d0142496f245fd85cdef881404982
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Mon Aug 12 12:20:31 2019 +1000
    Improved findPath speed
commit 1f564679b9ede50401f2520f3a7d651cf69c4aeb
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Mon Aug 12 12:11:56 2019 +1000
    Reworked path()
commit 023d4064d0725270ecdeb9733482d20dc7fa4ba8
Author: Zac <zachary.cripps@griffithuni.edu.au>
Date: Mon Aug 12 11:56:22 2019 +1000
    Created function for intermediateWord path
commit 19dc9b228b4a5893ce4a0f77114449e5b6471b3f
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Mon Aug 12 11:42:23 2019 +1000
    Removed debug output
commit 75c062de3b9e7ce9cd763b7810afeb309cf2b080
Author: Zac <zachary.cripps@griffithuni.edu.au>
Date: Thu Aug 8 09:03:20 2019 +1000
    Added slicing to the secondPath list
commit edbd6b87937826ad638a6f71fdd773f9f046ac9a
Author: Zac <zachary.cripps@griffithuni.edu.au>
Date: Thu Aug 8 01:26:22 2019 +1000
    Added comments to code
commit a1a22d9d9802372afe4c0958b1d4ed3242042998
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Wed Aug 7 10:33:57 2019 +1000
    Added intermediate word
commit 63b4ddcca7311821b7939fee186086ad01feae9e
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Tue Aug 6 17:11:54 2019 +1000

Fixed viableWords not resetting
commit 56ad2de95ec9aba1fc2f35666604ebf270cebaca
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Tue Aug 6 14:29:23 2019 +1000
Changed blacklist to a set, added some validation
commit 2df7671e49f722c8b5125edf3d9c55146c6eeb21
Author: Zac Cripps <zachary.cripps@griffithuni.edu.au>
Date: Tue Aug 6 13:19:05 2019 +1000
Add blacklist words to seen set
commit dd96996092f220c2f2c18f6d5b2792136a851cd7
Author: Zac Cripps <zachary.cripps@griffithuni.edu.au>
Date: Tue Aug 6 10:30:21 2019 +1000
Added a blacklist
commit bc72a01da6ad52ed76d696b4688351c2d5882732
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Sun Aug 4 12:47:15 2019 +1000
Improved same function
commit 74ad02173c8a7d3792909d54e6e09fa3c7949a7d
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Sun Aug 4 12:17:20 2019 +1000
Allowed changes to correct letters
commit e0bcfa41e4e065cea8cc5ad8964c171b2dde1110
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Sun Aug 4 12:00:43 2019 +1000
Optimised queue for more difficult ladders
commit f0afba1310a8f78cfa84e6183efad9547a95ba1a
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Sat Aug 3 21:48:18 2019 +1000
Optimisation
commit d4b4f84685bfbc02e8c8d347620aecba04901e0d
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Sat Aug 3 19:29:04 2019 +1000
Code simplification and better debug logging
commit 37b0821338b5475f3ec332b0fd8c14acd26b09bf
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>
Date: Sat Aug 3 14:59:57 2019 +1000
Re added comments, skip useless loops
commit 463fbc757aac1267e7d050ba350ca2467bff6685
Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>

Date: Sat Aug 3 14:33:50 2019 +1000

    Implemented queue

commit 2d1f0c6f4ad05c5b041082c1195d2d1215473993

Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>

Date: Fri Aug 2 20:53:17 2019 +1000

    Minor fixes, comments, gitignore

commit a26e90cb7c4dde923308c5aa2703c3c3c51d10ef

Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>

Date: Fri Aug 2 15:36:11 2019 +1000

    Added plan comments

commit f5ed07a38ce6bb1825832deadd2e814d1c88eee4

Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>

Date: Fri Aug 2 14:55:15 2019 +1000

    Added minimum similarity

commit 064cf494dc67f6f2ffb613931ebfe2396172fdc6

Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>

Date: Fri Aug 2 14:07:42 2019 +1000

    Reversed sorted()

commit 101ec7df073f4e134a38d9c6d87262c4c87865fa

Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>

Date: Fri Aug 2 13:46:14 2019 +1000

    Added print steps

commit ea36896a59336618259b29cbba24185c292ece26

Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>

Date: Wed Jul 31 13:55:53 2019 +1000

    Added function comments, better variable names

commit 09d0c311a8cd1890d94ec28ad353fabde2d42395

Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>

Date: Wed Jul 31 12:50:15 2019 +1000

    Minor optimisations

commit 866b84afe6196e05444f39e8a7202f1aeb80640d

Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>

Date: Tue Jul 30 10:40:52 2019 +1000

    Renamed document

commit efcf01b7923ec561880489c7142de8bbecc81734

Author: Thomas Sloman <thomas.sloman@griffithuni.edu.au>

Date: Wed Jul 24 13:43:43 2019 +1000

    Original files