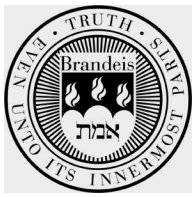

COMPUTER SCIENCE 12B (SPRING TERM, 2017)
ADVANCED PROGRAMMING TECHNIQUES

PROGRAMMING ASSIGNMENT 6
Due: Monday April 7th at 11:55pm

Programming with Collections - Edit distance -

You are to write a client program that computes the edit distance between two words, i.e., the **minimum** number of **operations** needed to transform one string into the other.

For this program, an operation is a substitution of as single character, such as from "brisk" to "brick". The edit distance between the words "dog" and "cat" is 3, because the chain of "dot", "cot" and "cat" transforms "dog" to "cat".

When computing the edit distance between two words, each intermediate word must be an actual valid word. Edit distances are useful in applications that need to determine how similar two strings are, such as spelling checkers.

Your program should ask the user for the name of a dictionary text file (we will provide this). From this file, you should compute a map from every word to its intermediate neighbors; that is, the words that have an edit distance of 1 from it. Once this map is built, you can ask the user to enter two words to compare and you "walk" the map to find paths from one word to another. This process should be done in a loop: once you compute the edit distance of two words, you prompt again the user for the next two words.

A good way to process paths to walk the map is to use a linked list of words to visit, starting from the beginning word, such as "dog" in the above example. Your algorithm should repeatedly remove the front word of the list and add all of its neighbors to the end of the list, until the ending word (such as "cat") is found or until the list becomes empty, which indicates that no path exists between the two words.

Special cases: (1) If a word is not included in the dictionary your program should print out the message "Word does not exist", (2) The edit distance of two words with different length is not defined. Therefore, if the two words given as an input by the user do not have the same length then your program should print the message "No solution", (3) if both cases (1) and (2) occur, you may print either error message.

General Guidelines:

You should download from LATTE two dictionary files. File `text.txt` has a dictionary of 8 words. You can use it to the your program with the "dog"/"cat" example (among

others). File `dict.txt` is a larger file and you should use it to test your program with more complex examples. We will be using this file for testing your programs.

You don't need to create any classes for this part. You only need to write one client program and you should name it `EditDistance.java`. You should however break your program into methods (e.g., one that builds the map, show transformation paths, calculates edit distance, etc)

You will be graded on correctness and programming style including the use of good variable names, comments on each class and each method, using local variables when possible, correct use of generics and the other standard style guidelines.

User Interaction:

While you choose how you want to break your code up into methods, we choose how your program needs to behave *externally*. You must follow the template we outline for you below.

Your program should first ask the user for the name of a dictionary file on one line. Your program should then loop through the following interactions, each of which should be on a separate line.

1. Ask the user for two words separated by a single space for which to compute the edit distance.

2. Print "Path = " followed by each word in the edit path (starting with the first word the user entered), a comma, a space, then the next word. A comma or a space should not follow the last word.

3. Print "Edit distance = x", where x is the edit distance.

Here is an example user interaction where everything in bold is input by the user, and everything not in bold is output by your program.

```
Enter name of dictionary file: dict.txt
Enter two words separated by a space: dog cat
Path = dog, dot, cot, cat
Edit distance = 3
Enter two words separated by a space: brisk brick
Path = brisk, brick
Edit distance = 1
Enter two words separated by a space: transportation cat
No Solution
Enter two words separated by a space: foobar whobar
Word does not exist
```

You may choose to implement a *quit* operating by listening for a single word from the user, such as "`quit`", if you would like, but you do not have to.

Design Hints:

There are better algorithms for computing edit distance, but this approach runs reasonably quickly even for a large dictionary like `dict.txt`. The slow part is computing the neighbors, but that could be pre-computed and stored in a separate file. The program uses a LinkedList as a queue to keep track of candidate words to explore. A set is used to keep track of previously explored words. The algorithm keeps a count of how many words are at each distance from the original word so that it can report the edit distance. It also constructs a map each time it explores that keeps track of how it got to each word. That allows the program to report the path between the two words. All in all, this involves a LOT of data structure manipulation using the standard Java collections classes.

You DON'T need to follow this design. You are free to design your own program as you wish.

Submission:

Your Java source code should be submitted via Latte the day it is due.

Grading:

You will be graded on

- **External Correctness:** The output of your program should match exactly what is expected. Programs that do not compile will not receive points for external correctness.
- **Internal Correctness:** Your source code should follow the stylistic guidelines shown in class. Also, remember to include the comment header at the beginning of your program.
- **One-on-one interactive grading:** By the end of the day that the assignment is due, please make an appointment with your TA for an interactive 10-15 minute grading session. You will receive an email notifying you of the name of the TA who has been assigned to you for this assignment with further instructions on setting up the appointment. (You will be meeting with a different TA for each assignment). One-on-one interactive grading will help you improve your programming skills and avoid repeating mistakes from one assignment to the next.