Introduction

This assignment has three parts: a written part (Q1), a programming part (Q2), and an optional written bonus part (Q3). You need to hand them in separately as stated above.

The goal is to design and implement (that is, write a working program for) a shortest path algorithm to solve a problem about word chains.

Please read these instructions carefully before posting any questions to Piazza. We hope all clarification questions you have will already be answered here. If not, please read previous questions on Piazza before posting new ones.

We will say two words (strings) $x$ and $y$ are linked from $x$ to $y$ if

- you can get $y$ from $x$ by deleting exactly one letter from $x$ (a “deletion”); or
- you can get $y$ from $x$ by inserting exactly one new letter (an “insertion”); or
- you can get $y$ from $x$ by interchanging two adjacent letters (this is called a twiddle); or
- you can get $y$ by reversing the order of the letters in $x$ (a “reversal”).

For example

- **loose** is linked to **lose** because you can delete one of the o’s in **loose** to get **lose**;
- **cat** is linked to **cart** because you can insert the letter **r**;
- **alter** is linked to **later** because you can get one from the other by a twiddle at positions 1 and 2;
- **drawer** is linked to **reward** because you can get one from the other by a reversal.

A **word chain** is a sequence of words $w_1, w_2, \ldots, w_n$, where for each $i$ with $1 \leq i < n$ we have that $w_i$ is linked to $w_{i+1}$. We say the chain links $w_1$ with $w_n$. For example,

\[
\text{spam, maps, map, amp, ramp}
\]

is a word chain because
• spam and maps are reversals;
• maps goes to map by a deletion;
• map goes to amp by a twiddle;
• amp goes to ramp by an insertion.

The cost of a word chain is obtained by summing the cost of each link of the chain, where

• a deletion costs 3;
• an insertion costs 1;
• a twiddle costs 2;
• a reversal of a $n$-letter word costs $n$.

If none of these cases apply, you can say that the link has infinite cost.

Thus, for example, the individual costs of the chain above are given as follows:

\[ \text{spam} \rightarrow \text{maps} \rightarrow \text{map} \rightarrow \text{amp} \rightarrow \text{ramp} \]

and the total cost of the chain is $4 + 3 + 2 + 1 = 10$.

In the (very rare) case where two words satisfy two different criteria, like tort and trot, the cost of the link is the lower of the two costs. Here it would be 2.

Given a word list of valid words, and two words $u, v$ from the word list, the word chain problem is to find the lowest-cost chain linking $u$ and $v$ using only words from the word list.

The Assignment

The goal is to create a working program to solve the word chain problem in one of C++, Java, or Python.

Q1. [10 marks] This is the written part. Hand it in via Crowdmark.

(a) Discuss how to efficiently solve the word chain problem using a graph algorithm. What do the vertices represent? What do the edges represent? How are they weighted? In what form will you store the graph? Is it directed or undirected? Do you expect the graph to be sparse or dense for a real word list, such as an English dictionary? Do you expect it to be connected? And, most importantly, how can you efficiently create the graph from a given word list?

Hint: if you are careful and think about the problem, you can probably create the graph from a word list rather efficiently.
words in the word list, in order to determine the weight of the edge connecting them, is not the best approach. It should be possible to create the graph in \( o(n^2) \) steps, where \( n \) is the number of words, but you might have to make some assumptions about what the word list looks like.

For this part, you are welcome to use techniques like tries, hashing, dictionaries, and so forth. Make any reasonable assumptions about the running time of anything you use, and state them. For example, it’s reasonable to assume that hashing a string of \( t \) letters costs \( O(t) \) time.

We are looking for high-level description here. There is no need to produce pages and pages of pseudocode.

(b) Next, explain how you can efficiently search the graph to find the solution to the word chain problem. Here you can assume the graph has been created, and you are given two words and want to find the lowest-weight chain linking them, or report that there is no such chain.

What graph algorithm is appropriate? These choices are for you to decide and explain. Here you should use one of the shortest-path algorithms we’ve covered.

There’s no need to provide pseudocode (unless you want to). Explain your ideas in English. Provide examples (if they are useful). Be sure to give as much detail as necessary to convince a skeptical TA.

(c) Finally, discuss the running time of both (i) creating the graph from the word list and (ii) given two words, finding the lowest-cost chain connecting them (if it exists).

When you do your analysis, be sure to specify if you’re obtaining a rigorous worst-case bound, or just a “heuristic” bound based on what you think happens in a typical example. You can express your running time in terms of the appropriate parameters of the word list, which might include \( n \), the total number of words in the word list, and \( q \), the total number of symbols of all words in the word list, and \( m \), the maximum length of a word in the word list.

We will be generous in marking this problem.

Q2. [20 marks] This is the programming part. Hand in via Marmoset.

Create a program to solve the word chain problem. Your program should have two distinct parts: a part that reads in a given word list and creates the graph from the word list, and a part that processes inputs (pairs of words) and determines the chain of lowest cost connecting the pairs.

Your program should read its word list from a file called `dict.txt`. Each line of `dict.txt` contains a nonempty lower-case word with no symbols other than the lowercase letters from a to z. Do not assume anything about the ordering of the word list. In particular,
it need not be in alphabetical order. The word list contains words of many different (positive) lengths.

Once your program has read in the word list, you should create the graph from it. For testing you can download a `dict.txt` from the course web page.

For the input of pairs of words, your program should read from standard input and write to standard output. Your program should expect, as input, the following: \( t \geq 1 \) lines of text, each containing two nonempty words \( u, v \), with the two words on each line separated by a single space.

There is no guarantee that the input words are actually in the word list. If either one is not, print the number \(-1\) for this pair.

(Don’t bother checking for inputs that don’t adhere to the specs.)

For example, an input might be

```
spam ramp
tippy sappy
```

You are to find the cost of the lowest-cost chain linking \( u \) to \( v \), for each of the \( t \) pairs \((u, v)\) of words given, and to give a chain with this cost. (There might be multiple chains of the given cost; you only have to find one.)

The output should be the following for each pair of words in the input: the cost of the chain, followed by a space, followed by the words of the chain from first to last, separated by one space. If there is no chain, the output should just be the number \(-1\).

For example, for the input above, if `dict.txt` were

```
spam
maps
map
amp
sap
sappy
tip
tippy
ramp
```

the output would be

```
10 spam maps map amp ramp
-1
```

Each line of the input and each line of the output has (of course) a newline `\n` at the end.
Other information

Your program should be called A9Q2.cpp if it is written in C++, or A9Q2.py, if it is written in Python.

If you use Java, the main method must be inside a public class called “A9Q2”. Note that capitalization and spelling of these names are extremely important for us to properly mark your assignment. If you don’t follow the naming conventions we list here, you risk getting a 0. You can use up to Java 10.

Your program must be original. However, you can use any standard built-in features or packages that go with the language you choose, (including, for example, the built-in arithmetic operations like +, −, *, /, <<, >>, %, etc.), vectors, strings, anything in Java.Util, built-in math functions, std::stack, NumPy, ceiling and max functions, etc.)

In particular, you are free to use in your code any standard packages for things like priority queues, binary heaps, dictionaries, hashing, tries, etc. Be aware that many implementations of binary heaps do not support the DECREASE-KEY operation, so you may have to consider how to handle this.

Tests

Once you get your program running on the big word list provided, try these queries to test your program and have some fun:

• How much does it cost to turn hate into love?
• Can you turn water into wine?
• Contrary to what some claim, you can get there from here. But you can’t get from here to eternity.
• Can you put the cart before the horse?
• Can a devil become an angel?
• Can the lost become found?
• A loser can’t become a winner, and there’s no way to turn chaos into order, but someone small can become large.

You don’t need to hand in any of these answers.

Submission

Only submit a program in one of the three programming languages; if you submit multiple times, we will only mark the last one. We will compile your program in the Linux environment using the command
g++ -std=c++11 A9Q2.cpp  (for C++)

or

javac A9Q2.java   (for Java).

For python we will use the default python on the student environment, which is currently Python 2.7.12.

Please make sure your code compiles and runs properly on the student environment before submitting it (preferably before the due date, in case there are any problems).

Submit your program through Marmoset: at https://marmoset.student.cs.uwaterloo.ca

Name your code file as discussed above.

Once you submit your program, you can see whether your program passes the public tests, which consist of 1 or 2 easy test cases. This is for you to test your output format. Don’t trust the public test cases exclusively! You are strongly recommended to test your code with your own test cases. Your final mark will be the marks for your last submission, tested by the public test cases and others (not public) that we design.

Marking

Your results will be automarked, so be sure that your output is precisely in the right form.

If your program takes more than 100 seconds of cpu time, it will time out and we will stop running it. Unless you are doing something very, very wrong, this is unlikely to happen.

Here’s how you will be evaluated on this part. We will test both correctness (14 marks) and efficiency (6 marks).

For correctness, we will run your program on some inputs and check that you answer correctly. Since, in general, there can be multiple lowest-cost chains for a given pair, you do not need to produce exactly the same one our program does; you just need to produce a correct one with the minimum weight. We won’t check things like empty input. You need to pass most of our test cases to get full marks. There will be no tricks, just tests on genuine inputs.

For efficiency, we will use cpu time as a proxy. You only get marks for efficiency on a problem if you received full marks for correctness on that problem.

Errors

If you get a Marmoset error like Assignment 9, Question 1 Compiling... Cannot find any C++, Java, or Python submission this probably means you submitted a program that was named wrong.
If you get a Marmoset error like

Your code crashed

that probably means you returned 1 after printing, instead of doing return 0 to exit the program.

**Bonus Question**

Q3. [Bonus question; 5 marks extra credit only]

Consider all pairs of words in the large word list (called `dict.txt` and available on the course home page) and consider the lowest-cost chain connecting them (if it exists). What is the largest of all of these? Provide the two words, the weight of the chain, and the words in the chain connecting them. No program is needed here, just the items of the previous sentence. Hand in via Crowdmark.