Project: Word Games

In this project we are concerned with the problem of finding a chain (or sequence) of distinct words \( w_1, w_2, \ldots, w_k \) in a given set (dictionary) of words \( \{w_1, w_2, \ldots, w_n\} \) such that one can go from one word \( w_i \) in the sequence to the next \( (w_{i+1}) \) by a changing a single letter. For example, the following is such a chain of words:

\[
\text{pass} \rightarrow \text{past} \rightarrow \text{part} \rightarrow \text{wart} \rightarrow \text{wait} \rightarrow \text{bait} \rightarrow \text{bail} \rightarrow \text{fail}.
\]

The length of a chain of words is defined to be the number of words it consists of (e.g., the chain of words above has length 8). Furthermore, changing a letter incurs a certain nonnegative cost, which may depend on the letter and the word. For example, if \( w_i \) is the \( m \)-letter word \( a_1a_2\ldots a_m \), we might define the cost of changing \( a_j \) to \( a'_j \) to be one of the following:

- one, irrespective of the letter that is being changed and the letter it is being changed to, and irrespective of the position of the letter in the word;
- the distance of \( a_j \) from the end of the word (i.e., \( m - j + 1 \)); or
- the lexicographic distance between the two letters \( a_j \) and \( a'_j \) (e.g. the distance between the letters “a” and “c” is 2); or
- one, if changing a vowel to another vowel or a consonant to another consonant, and two otherwise.

Given a definition of the cost of changing a letter, we define the cost of a chain of words to be the sum of the costs of the letter changes that occur in the chain of words. For example, if the cost of changing a letter is defined to always be equal to one, then the cost of the chain of words given above is equal to 7. Finally, we define the distance between two words to be the minimum cost of a chain between the two words, or \( \infty \) if there is no such chain.

Given a set of \( n \) words from some dictionary, and a cost function \( f \), we are interested in efficient algorithms for answering the following queries/questions:

(i) Given two words, compute their distance, and find a least-cost chain between them.

(ii) Given a word \( w \), and an integer \( l \), find a least-cost chain starting with \( w \) that has length at least \( l \).

(iii) Given a word \( w \), and an integer \( c \), find a smallest-length chain starting with \( w \) that has cost at least \( c \).

(iv) From among all pairs of words which can be connected by a chain, find a pair whose distance is maximal, and find a least-cost chain between them.
In each query above, your algorithm(s) should return “No”, if there can be no chain of words, chosen from the given set of words, with the desired properties.

Your algorithm(s) are to be used in the following two modes:

**Static.** The set of words, once given, remains fixed and does not change. The user(s) will be asking any of the questions above many times.

**Dynamic.** Each time the user gives you with the set of words and the query s/he is interested in.

In designing your algorithms you need to take into consideration the mode in which your algorithm(s) will be used. In the static mode, you have the option to preprocess the input set of words in order to minimize the time it takes to respond to a user query. The time and space it takes to preprocess the input set is not of concern to the user. In the dynamic mode, where s/he provides you only with the query but also with the set of words, the amount of preprocessing your algorithm(s) may do is limited, since an excessive amount of preprocessing will adversely affect the response time to the user.

**What to do**

1. Design algorithm(s) for answering queries (i)–(iv). You may have different algorithms for the static and/or dynamic modes.

2. Develop appropriate software programs implementing your algorithm(s). You may use the programming language(s) of your choice. However, you must ensure that your programs can execute on the Unix machines (gl.umbc.edu) available to you on campus.

3. Analyze the running–time and memory requirements of your algorithm(s) for the intended usage mode (e.g. static or dynamic).

4. Test your algorithms using the set of words in the dictionary of the Unix spell program (/usr/share/lib/dict/words).

5. Experimentally evaluate your algorithms using subsets of the words of the Unix spell dictionary above, different costs for the operation “change a letter”, and different choices for the constants $c$ and $l$. Choose various subsets consisting of various numbers of words (e.g. subsets with 100 words, 1000 words, 5000 words, etc). Measure the time and memory required to answer queries (and preprocess the data).

6. Optimize your programs as much as you can and repeat the experiments if necessary.

7. Analyze your experimental results. This means that you do the following:

   - Statistically analyze the measurements you obtained (e.g. compute averages, medians, minima, maxima, standard deviations, etc). How do they depend on the number of words in the set? How do they depend on the cost of the “change a letter” operation that is used? How do they depend on the total number of characters of all the words in the set? Do they depend on any other input parameters or conditions?
• Plot the results of your statistical analyses.
• Discuss the results of your analyses (e.g. what conclusions can you draw from them?)

To measure the amount of computational resources your program(s) are using you should use the `getrusage` and `mallinfo` functions under the IRIX 6.x or Solaris 2.x operating systems, or the speedshop suite under IRIX 6.2+. See the Unix man pages on these functions for more details.

What to Hand in

Each student must hand in a written project report in the format of a computer science technical report. This report must explain what you did, why you did it, how you did it, what you found, and what is the significance of your findings. Do not simply list experimental findings; be sure also to interpret your findings. In addition, comment on the engineering aspects of your work: What difficulties did you encounter, how did you resolve these difficulties, and what were the consequences of your solutions to these difficulties?

Whenever possible, summarize your important experimental findings in appropriate graphs. As a separate appendix to your report, include a well-documented copy of all source code. For more about technical reports, see Alan Sherman’s guide to writing technical reports, which you can find at http://www.cs.umbc.edu/~sherman/Courses/documents/TR_how_to.html.