Data Structure and Algorithms Projects

Please select one of the following projects to implement:

**Project 1: Spell Checking [50% of the class work]**

Spell checking is the process of verifying that a particular word is spelled properly according to some dictionary. Spell checkers are used in many applications, including word processors (such as Microsoft Word), electronic dictionaries, and optical character recognition (OCR) systems that need to turn images of printed text (or even handwriting) into coherent text.

Spell checking itself is trivial, requiring only a simple lookup in a dictionary. However, most applications of spell checking also require that the spell checker provide a list of potentially correct spellings (“near matches”) when the word was spelled improperly. For instance, if I type “speling” into an online dictionary, it will provide suggestions of similar words that I may have meant to type, including “spelling”, “spoiling”, “sapling”, and “splendid”.

Your task is to implement a spell checker that determines if a given word is spelled correctly based on a dictionary lookup. When the word is not spelled correctly, it will provide a list of similar-sounding words based on your implementation of the Metaphone algorithm, and ordered based on their edit distance from the string that the user typed.

We could use only the edit distance to find near matches, but by first using the Metaphone algorithm we achieve better results because it can find the word you are looking for even if you have no idea how to spell it. Additionally, we can isolate the set of near matches more quickly using the Metaphone algorithm.

Read on to learn about the Metaphone algorithm, which will be used to isolate the set of near matches; and an edit distance algorithm, which will be used to rank the results (notice that “spelling” comes before “spoiling”). Fig. 1 gives an overview of the spell-checking process.

![Figure 1: Overview: Sequence of Transformations for Spell Checking](image-url)
The Metaphone Algorithm

The Metaphone algorithm, created by Lawrence Philips, takes a word and returns a very rough approximation of the sound of that word. The rough approximation eliminates the distinction between some characters; for instance, ‘S’ and ‘Z’ sound very similar, so the Metaphone algorithm maps them to the same sound ("s"). Some letters may make many different types of sounds: ‘C’, for instance, may make the “sh” sound (denoted by ‘X’) when it is part of “cia” (as in “social”), the “s” sound when it is following by ‘C’, ‘I’, or ‘Y’ (as in “since”), or even the “k” sound when it is by itself, or followed by a ‘K’ (as in “clack”).

The Metaphone algorithm maps every sound in a word down to one of the following sounds: *, B, X, S, K, J, T, F, H, L, M, N, P, R, @, W, Y. The * symbol represents the sound of a vowel at the beginning of the word, ‘X’ represents the “sh” sound, and ‘@’ represents the “th” sound. All of the other consonant sounds in the list sound like the consonant. Vowels not at the beginning of the word are considered silent.

See Fig. 2(a) for some examples of strings generated by the Metaphone algorithm for some common words. We (arbitrarily) limit the length of the Metaphone output strings to 4 characters, which has been found to produce better results in many cases than using longer strings.
Consider a data communication network that must route data packets (email or MP3 files, for example). Such a network consists of routers connected by physical cables or links. A router can act as a source, a destination, or a forwarder of data packets. We can model a network as a graph with each router corresponding to a vertex and the link or physical connection between two routers corresponding to a pair of directed edges between the vertices.

A network that follows the OSPF (Open Shortest Path First) protocol routes packets using Dijkstra’s shortest path algorithm. The criteria used to compute the weight corresponding to a link can include the time taken for data transmission, reliability of the link, transmission cost, and available bandwidth. Typically each router has a complete representation of the network graph and associated information available to it.

For the purposes of this project, each link has associated with it the transmission time taken for data to get from the vertex at one end to the vertex at the other end. You will compute the best path using the criterion of minimizing the total time taken for data to reach the destination. The shortest time path minimizes the sum of the transmission times of the links along the path. The network topology can change dynamically based on the state of the links and the routers. For example, a link may go down when the corresponding cable is cut, and a vertex may go down when the corresponding router crashes. In addition to these transient changes, changes to a network occur when a link is added or removed.

Example

Consider a very simplified example of a network at RPI, with vertices at Amos Eaton, Lally, DCC, Troy, Sage, and Folsom. See Figure 1 for an illustration. For this example, the shortest time (or fastest) path from Amos Eaton to DCC is: Amos Eaton–Troy–DCC with a total transmission time of 0.9 milliseconds.
Figure 1: An example network graph. Each link shown represents two directed edges. Shown next to each link are the associated transmission times of its two edges in milliseconds.