Advance Programming Assignment1

# Introduction

This assignment is about finding a path from one word of the dictionary to another and also return the hamming distance between those words. But there is a catch, while we go from one word to another, we have to make sure that each change in the first word makes it another valid word in the dictionary. So this requires us to modify the hamming distance code so that it fits our bespoke needs.

We further need to perform analysis on the paths returned.

Language used: JAVA

IDE used: Eclipse

# Problem

1. The starting challenge is to read the JSon stream or text file and parse it to extract the words of the dictionary. ( Put these words in an ArrayList aka wordList )
2. Splitting words into separate buckets based on their lengths. This optimize the code when will create graphs.
   1. Get length of the maximum words to that we know how many buckets to create.
   2. Place words into the buckets.
3. Creating a graph. To make graph we will use the words in the buckets. This will limit our search space to words of same length.
   1. Find out successors (or children). A function which returns true if the word being compared to parent word is one hamming distance away.
   2. Add children (or successors) for each GraphNode in the buckets (or HashMaps).
   3. Create a functions to make as many buckets as there are lengths of words and then populate those buckets.
   4. Create graph out of those words in that bucket.
4. Do graph search. I am going to use A\* search as it is an optimized approach.
   1. Make a heuristic for A\* search.
   2. Learn about PriorityQueues in java. Then implement a PriorityQueue.
   3. Make a new object to be placed in the PQ. This object has the path and the cost from source to destination.
5. Calculate the minimum chain for all of the solvable words.
6. Calculate the maximum chain for all of the solvable words.
7. Calculate the words with no chains at all.

# Solution (our approaches)

1. We first tried to read the json stream through python, but were unsuccessful. So next we tried Java and we don’t know why but it worked. We imported Json library and used it to read the file and store the words in an ArrayList.
2. Function Name: makeBuckets(). This function returns an ArrayList of buckets, where each bucket represents words of a specific length
   1. Function Name: getMaxWordLength()
   2. For each length size of the word, traverse the dictionary (wordList) and separate the words into buckets (based on their lengths).
3. Function Name: createGraph(). Iterate over all the buckets and create a graph in each one of them.
   1. Function Name: isWordChild(). Checking if hamming distance between the words is one or not.
   2. Access each bucket (or HashMap) and for all the words (or GraphNodes) in it compare it with the rest of the words. Words which have 1 hamming distance are joined.
   3. Done
   4. Done
4. Function Name: aStarSearch().
   1. Function Name: aStarHeuristic(). This is just a simple hamming distance calculation
   2. Done
   3. Class Name: PathClass()
5. A map data structure is used to hold the solvable words as key and the shortest path as the value.
6. Function Name: makeBuckets(). This function returns an ArrayList of Graphs. Each bucket in my convention represents a graph of a specific length
   1. Function Name: getMaxWordLength()
   2. For each length size of the word traverse the dictionary (wordList) and separate the words into buckets (based on their lengths). Each bucket represents a graph.