# CSCI311 Final Project

Jim Campbell, Jack Goldberg, Tracy Li, Elsa Perelli

November 20, 2020

## 1 Algorithm Description

This project implements a spellcheck system with C language. The core algorithm for this system is fulfilled by a hash table and Longest-Common Subsequence (LCS).

1. **Storage of the dictionary word list**
   This project corrects the input word according to the provided dictionary word list. Our team stores the correct word list to a adjacent list. Each cell holds a pointer to the head of a doubly linked list ADT of all the words with same length.

2. **Rules to check spelling**
   This system first looks up the input word in the hash table our team created. If the correct word is not found, then apply the LCS algorithm to find the best fit. Our program calculates a range of word lengths to run the spell-check algorithm on. The bounds are a fourth shorter than the input up to a fourth longer than the input. So, the algorithm does not need to check an entire dictionary, just a subset.

   According to our research, people are likely to spell a word incorrectly, but the word length will remain close to the intended word.

3. **Hash table**
   Our hash table uses hash function shown below:

```
int hash( char *string) {
    unsigned int hash_val = 5381;
    int c;
    while (c = *string++)
        hash_val = ((hash_val<< 5) + hash_val) + c;
    return hash_val;
}
```

   By using this hash function, our team greatly lowered the number of collisions.

4. **LCS algorithm**
   Our team implements a standard dynamic programming LCS algorithm according to the class note. The `lcs()` function in the `lcs.c` will return the length of LCS of the input word and given reference to the dictionary word list.

   The searching range of the LCS follows the rule:

```
int lower_bound = (int) ((double) strlen(input) * (double) (.75));
int upper_bound = (int) ((double) strlen(input) * (double) (1.25));
```

   The upper bound has maximum value of 14 which is the longest word in the given dictionary.

## 2 Asymptotic Runtime Analysis

1. Generating the dictionary adjacent list: $O(n)$
2. Generating the hash table for the word list: $O(n)$
3. The hash table for checking the word is correct or not: $O(1)$
4. The dynamic programming LCS: $O(mn)$, where $m$ is the length of input word, and $n$ is the word length of the dictionary dlist we are looking at.

Therefore, summing up the above procedures, the LCS step dominates the asymptotic runtime of our algorithm is $O(mn)$.