Major Difficulties

One of the major difficulties I had to overcome while doing this project was deleting things at corresponding positions if either distance, word1, or word2 was faulty at that position. In order to do this I shifted the position of everything after the erroneous input to the left in distance, word1, and word2. If the problem occurred in the last c-string of word1 or word2 or in the last number of the array *distance*, I simply decremented nRules because since it is in the last position, there is nothing to shift over it. Another problem I faced occurred in the second function determineQuality. In order to compare each “word” in the translation of document to each word in word1 and word2, I had to use string tokenizer in order to create new c-strings that held each word from the translation of the document.

My first function uses a for loop design along with several if statements in order to check what positions should be removed while simultaneously adjusting the value of nRules. The structure is as follows..

Pseudocode for function 1

Repeatedly check each number in distance and each word in word1 and word2

If the last number for distance is a problem, decrement nRules

If distance is neg or 0

Shift everything back by 1 in distance, word1, and word2

If word1[i] or word2[i] is blank then remove line and decrement nRules

Repeatedly check each character in each word of word1 and word2

If they are letter make them lowercase

If not then shift everything after the error to remove the line

Another loop repeatedly runs through each word after they have all been converted to lowercase

Check if two rules are equal, if so keep one with greater distance

If both rules are the same but in reverse order

Keep one with greater distance

Decrement appropriately

My second function uses a translation of the document. This translation is separated by word by tokenizer and then compared to the words in words1 and words2.

The structure is as follows…

Pseudocode for function 2

initialize new empty c-string

repeatedly check the position in document to see if it is a space or a letter

if so, set the position of the blank c-string to the according letter or space

change to lowercase

increment the position of the once blank c-string

when document reaches zero-byte, break.

Use string tokenizer

Create new c-string for each word

keep a counter of how many rules match

check each word in word1[] against every word in newDocument

if a word in newDocument matches a word in word1, check if there is another word

in the document that matches the word in the corresponding position of word2 and

is not greater than *distance* words away.

if so, increase counter by 1 and break

break again to outermost for loop

Within my outermost for-loop, I am incrementingthe position of word1, so I am checking each word in the newDocument against each word in word1.

In order to condense my code, I check if each word in the document matches a word in word2 by the same procedure. In fact, both the checks for matches in word1 and word2 are nested in the same outer for loop that increments the position of word1 or word2.

Test Cases

For Function standardizeRules:

If distance is negative: (-4, word1, word2, 5)

If distance is zero: (0, word1, word2, 5)

If c-string in word1 contains something other than a letter: (distance, word1, word2, 5) where word1 contains something other than a letter

If c-string in word2 contains something other than a letter: (distance, word1, word2, 5)

If c-string in word1 is empty: (distance, word1, word2, 5)

If c-string in word2 is empty: (distance, word1, word2, 5)

If two sets of rules in word1 and word2 are the same: (distance, word1, word2, 5)

If two sets of rules in word1 and word2 are the same but in reverse order: (distance, word1, word2, 5)

If the problem in the rules occurs in the last number of distance, or last word in word1 or word2: (distance, word1, word2, 5)

If c-string contains capital letters: (distance, word1, word2, 5)

Purpose: the purpose of these test cases is to make certain that in the above circumstances, the bad rules are removed and the proper value for nRules is obtained

For Function determineQuality:

If nRules is 0, return 0: (distance, word1, word2, 0, document)

If document contains non-letter characters:

(distance, word1, word2, 4, document)

If document has no matches: (distance, word1, word2, 3, document)

If a document has 1 match: (distance, word1, word2, 4, document)

If document matches the same rule more than once:

(distance, word1, word2, 3, document)

If document has multiple matches: (distance, word1, word2, 5, document)

If document has a rule that contains the same word twice:

ex word1[i] = mad

word2[i] = mad

(distance, word1, word2, 4, document)

If document matches a rule but in reverse order:

(distance, word1, word2, 3, document)

If distance goes beyond length of document: (distance, word1, word2, 6, document)

If the words in document appear to match the rule, but its outside the specified distance (distance, word1, word2, 5, document)

If document only contains one word (distance, word1, word2, 5, document) where document is something like “Hi”

Purpose: The purpose of the above tests is to make sure that the proper value for match rules is obtained.