CS 31 – Project 5 – Jahan Kuruvilla Cherian 104436427

1. The most notable obstacle came with the second function, determineQuality, wherein figuring out how to do a cstring comparison within the document (which was only technically one large cstring) to each word from the match rules. I overcame this by formulating two local arrays, one which will be one large cstring that contains only relevant characters, and then copy over individual words from that string to a 2-Dimensional array. The other obstacle was being able to compare word1 and word2 both forwards and backwards with respect to their distances so that word2 could come either before or after word1. I accomplished this by implementing a helper function called determineMatchRule that checked for either word1 or word2 after the first word was detected, depending on which word was initially detected, that is if word1 was first detected then check for word2, but if word2 was first detected then check for word1.
2. Pseudocode:

***Function to print out the match rules***

***Go through the rows of the array***

***Print out the distance***

***Print out the word1***

***Print out the word2***

***Function to print out the revisedMultiDocument***

***Go through the rows of the array***

***Print out the word***

***Function to refresh the revisedDocument***

***Go through the document one character at a time***

***Replace it with the zero byte to make it officially empty***

***Function to refresh the revisedMultiDocument***

***Go through the rows of the 2-d array***

***Go through the individual columns of the 2-d array***

***Replace each character with zero byte to make it officially empty***

***Helper Function to determine whether the words match the match rule***

***Go through each row***

***Only if the distance is greater than zero***

***Check if the first word is equal to word1 or word2 and then check the word distance after to see if its same as word2 or word1***

***Otherwise decrease distance so as to check through 0 to distance away from the first word***

***Function standardizeRules to make sure the rules are valid***

***If we ever have the number of rules less than or equal to zero then give back zero***

***Store the number of rules that are valid in an integer***

***Go through the rows of the match rules***

***If we ever have negative distances, immediately deem that row invalid***

***Go through the columns of the words***

***If either of the words has uppercase characters***

***Make them lowercase***

***If either of the words has non alphabetic characters***

***Make their distance negative and deem that row invalid***

***Go through the rows once***

***Go through the rows again***

***If both words have valid lengths (not empty)***

***Get the value of the comparison of word1 with all word 2***

***If both words have valid lengths (not empty)***

***Get the value of the comparison of word2 with all word 1***

***If both word1 and word2 are equal but in the same row then do nothing***

***If then both word1 and word2 are equal in another match rule***

***Get rid of the one with the greater distance by making its distance -1***

***And by making that entire row empty***

***Go through the rows again***

***If we ever have a negative distance then we know its an invalid set from before***

***Make sure all of the invalid sets are empty***

***Starting at the point where we meet an invalid set go through the rows***

***past this point***

***If we meet a row with a valid set, then move it up and move***

***the invalid set down***

***then increment the initial loop to go through the entire loop again***

***Finally give out the number of valid match rules***

***Function determineQuality to check the number of match rules in a document***

***Create a 1-dimensional local array to hold the revised document***

***Create a 2- dimensional local array to hold the words from the revised document***

***Create an integer to hold the jumps (rows) for later use***

***Create an integer to hold the number of columns for later use***

***Create an integer to hold the number of matched rules in the document***

***Create a Boolean to determine if there was a space in the revised document***

***Refresh revised document to just zero bytes***

***Refresh revisedMultiDocument to just zero bytes***

***Simultaneously loop through both the document and the revisedDocument***

***If the character is an alphabet or a space***

***Store the character in the revisedDocument***

***Otherwise go back one character in the revisedDocument***

***If the document reaches a zero byte (that is the end) then exit out of the loop***

***Loop through the revisedDocument***

***If we have an uppercase character in the revisedDocument***

***Make it lowercase and store it in the exact same location***

***Loop through the revisedDocument***

***If the character is not a space***

***Store the character at the relevant row and column of the 2-d array***

***And make the Boolean of last space be false as no space was***

***encountered***

***Otherwise if it is a space and there was a space before this character***

***Do nothing***

***Otherwise if it’s a space and there was no space before this character***

***If the very first character of the column is not a zero byte***

***Make the column character a zero byte for that row***

***Reset the column counter to zero***

***Increment the jump to now store a word in the next row***

***And make the Boolean of the last space true as there was a***

***space encountered***

***Create an integer that holds the number of words in the 2-d array with respect to the***

***Rows***

***Loop through the set of match rules***

***Loop through the words row by row in the 2-d array***

***If the word in that row is equal to either word1 or word2***

***Use the helper function to determine whether the next word***

***n distance after this word is also equal to either word2 or***

***word1 and make sure the number of matchedRules is***

***incremented once***

***Refresh the revisedDocument***

***Refresh the 2-d array***

***Finally output the number of matched rules in the document***

1. Test Case

**Tests for StandardizeRules function:**

const int TEST1\_NCRITERIA = 4;

int test1dist[TEST1\_NCRITERIA] = {

2, 4, 1, 13

};

char test1w1[TEST1\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"mad", "half-witted", "nefarious", "have"

};

char test1w2[TEST1\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"scientist", "guy", "plot", "mad"

};

1.) assert(standardizeRules(test1dist,test1w1,test1w2,TEST1\_NCRITERIA) == 3) //tests for normal cases wherein we have rules with non alphabetical characters, which should be ignored.

const int TEST2\_NCRITERIA = 4;

int test2dist[TEST2\_NCRITERIA] = {

2, 4, 1, 13

};

char test2w1[TEST2\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"mad", "half-witted", "nefaRIous", "have"

};

char test2w2[TEST2\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"scientist", "guy", "plot", "MaD"

};

2.) assert(standardizeRules(test2dist,test2w1,test2w2,TEST2\_NCRITERIA) == 3) //tests for cases wherein we have rules with uppercase letters that should be converted to lowercase.

const int TEST3\_NCRITERIA = 4;

int test3dist[TEST3\_NCRITERIA] = {

2, 1, 1, 13

};

char test3w1[TEST3\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"mad", "PLOT", "nefarious", "scientist"

};

char test3w2[TEST3\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"scientist", "NEFARIOUS", "plot", "mad"

};

3.) assert(standardizeRules(test3dist,test3w1,test3w2,TEST3\_NCRITERIA) == 2) //tests for cases wherein we have repeated match rules, and those with a lower distance should be disregarded, and if they have equal distances then only one has to pass through.

const int TEST4\_NCRITERIA = 4;

int test4dist[TEST4\_NCRITERIA] = {

-3, 4, 1, 13

};

char test4w1[TEST4\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"mad", "half-witted", "nefari#$us", "have"

};

char test4w2[TEST4\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"scientist", "guy", "plot", "mad"

};

4.) assert(standardizeRules(test4dist,test4w1,test4w2,TEST4\_NCRITERIA) == 1) //tests for cases wherein we have negative distances, which should immediately disregard the entire match rule in that row.

const int TEST5\_NCRITERIA = 4;

int test5dist[TEST5\_NCRITERIA] = {

3, 4, 1, 13

};

char test5w1[TEST5\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"", "half-witted", "nefari#$us", "have"

};

char test5w2[TEST5\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"scientist", "guy", "plot", ""

};

5.) assert(standardizeRules(test5dist,test5w1,test5w2,TEST5\_NCRITERIA) == 0) //tests for cases wherein we have empty words, which should in turn disregard the match rule because standard form doesn’t decree empty characters.

const int TEST6\_NCRITERIA = 4;

int test6dist[TEST6\_NCRITERIA] = {

12, 4, 1, 13

};

char test6w1[TEST6\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"mad", "half-witted", "nefari#$us", "have"

};

char test6w2[TEST6\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"mad", "guy", "plot", "mad"

};

6.) assert(standardizeRules(test6dist,test6w1,test6w2,TEST6\_NCRITERIA) == 2) //tests for cases wherein we have word1 and word2 be the exact same in the same match rule, which is deemed standardized and should make sure that this case isn’t disregarded.

7.) Make sure to call standardizeRules multiple times on the same test case, to make sure the function continues to maintain the same number of rules returned, and doesn’t decrement by one.

const int TEST7\_NCRITERIA = 4;

int test7dist[TEST7\_NCRITERIA] = {

123455, 4, 1, 13

};

char test7w1[TEST7\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"mad", "asdfghjklmnbvcxzqwer", "nefari#$us", "have"

};

char test7w2[TEST7\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"scientist", "guy", "plot", "chicken"

};

8.) assert(standardizeRules(test7dist,test7w1,test7w2,TEST7\_NCRITERIA) == 3) //tests for cases wherein we have large integer values as the distances, but not above the max int value; and also makes sure that a match rule of length 20 characters is deemed valid, but one greater than 20 would not be valid.

9.) Check for a distance equal to zero, in which case that match rule mustn’t be valid, and as such that entire row should be eliminated.

**Tests for determineQuality function:**

const int TEST1\_NCRITERIA = 5;

int test1dist[TEST1\_NCRITERIA] = {

2, 4, 1, 13, 3

};

char test1w1[TEST1\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"mad", "deranged", "nefarious", "have", “deranged”

};

char test1w2[TEST1\_NCRITERIA][MAX\_WORD\_LENGTH+1] = {

"scientist", "robot", "plot", "mad", deranged”

};

1.) assert(determineQuality(test1dist, test1w1, test1w2, TEST1\_NCRITERIA,

"The mad UCLA scientist unleashed a deranged robot.") == 2) //tests for normal cases wherein there are some words from the test match rules in the document.

2.) assert(determineQuality(test1dist, test1w1, test1w2, TEST1\_NCRITERIA,

"The mad UCLA $scientist unle@asheD a deranged robot.") == 2) //tests for the case where the document may contain some unwanted non alphabetical characters that should be ignored and anything uppercase should be converted to lowercase.

3.) assert(determineQuality(test1dist, test1w1, test1w2, TEST1\_NCRITERIA,

"that is a Very Nefarious:: plot!!") == 1) //tests for cases wherein the match rule words may be before or after each other, and where the it should still find the match rule even if the words are not exactly distance apart, but within the distance.

4.) assert(determineQuality(test1dist, test1w1, test1w2, TEST1\_NCRITERIA,

"Two mad scientists suffer from deranged-robot fever.") == 0); //tests for when there is slight differences in words such as scientist not matching with scientists; and that if we have deranged-robot fever, it gets converted into a single word.

5.) assert(determineQuality(test1dist, test1w1, test1w2, TEST1\_NCRITERIA,

"deranged deranged deranged deranged deranged deranged") == 1); //tests for when we have multiple of the same match rules, but it should only be considered once, and return it once, and also checks for if word1 = word2 in the same match rule, this is still deemed correct.

6.) assert(determineQuality(test1dist, test1w1, test1w2, TEST1\_NCRITERIA,

"") == 0); //tests for when we have an empty document, and it contains no match rules at all, but an empty document is still valid.

7.) assert(determineQuality(test1dist, test1w1, test1w2, TEST1\_NCRITERIA,

"The deranged robot.") == 1); //tests for when there is a distance larger than the total number of words in the document, and so prevents undefined behavior by checking outside the bounds.