1. Notable obstacles
2. Trying to understand the spec and figuring out the limit and condition.
3. Finding a way to recognize the same w1 and w2 rules and remove the lesser one, especially the condition in my if statement, it took me a long time to be aware of my mistake.
4. Finding a way to figure out that I need two separate function to see if the rules are valid and if there are overlaps, and then coordinate them into if statement to remove the wrong one.
5. Figuring out how to find the range of n to prevent the case of negative position in an array.
6. Finding the way to accumulate the number of right rules without making repeated count.
7. Finding a way to transfer the words in document into a two dimensional array.
8. Debug!!!!!
9. Description(pseudocode)

Function1:

if “nRules” is less than or equal zero, consider it as zero, and then return 0 because there is no rules.

Change all the rules into lower case using “void lowerized”

Using “int isValidRule” and “int notHaveSamerule” to see if the rule in concern of “n” is valid or overlaps with former rules.

If there is, move the problematic rule to the end, and scoot all the rest forward for one position.

Keep doing the same n, until there is a right rule, then skipping it and n++;

The right time is equal to n.

Function2:

if “nRules” is less than or equal zero, consider it as zero, and then return 0 because there is no rules.

Create a new array “doc” to hold the lowerized document.

Create a new array “sim”, and put the words separately into each row, and count the position of rows.

Find the number of satisfaction, and if it satisfies, jump out of the loop and excute for the next rule.

1. Test data
2. char word1[20][MAX\_WORD\_LENGTH + 1] = { "FI","VE" };
3. char word2[20][MAX\_WORD\_LENGTH + 1] = { "SX","SIX" };
4. int distance[20] = { 0,-1 };
5. assert(normalizeRules(word1, word2, distance, 2) == 0);

IF THE DISTANCE IS LESS THAN ZERO

1. char word3[20][MAX\_WORD\_LENGTH + 1] = { "QWe","JIJIji","WhatI" };
2. char word4[20][MAX\_WORD\_LENGTH + 1] = { "MUN","Is","Me" };
3. int distance1[20] = { 1,2,3 };
4. assert(normalizeRules(word3, word4, distance1, 3) == 3);
5. assert(strcmp(word3[0], "qwe") == 0);
6. assert(strcmp(word3[1], "jijiji") == 0);
7. assert(strcmp(word3[2], "whati") == 0);
8. assert(strcmp(word4[0], "mun") == 0);
9. assert(strcmp(word4[1], "is") == 0);
10. assert(strcmp(word4[2], "me") == 0);

IF THE DISTANCE IS LESS THAN ZERO

1. char word5[20][MAX\_WORD\_LENGTH + 1] = { "341","","cheM","math","physics\*\*\*","","!!finance" };
2. char word6[20][MAX\_WORD\_LENGTH + 1] = { "Ryan&" ,"oi io xxee909","omg this is har","weird","evil","","paper" };
3. int distance2[20] = { 2,4,63,6,43,2,5,65,4,3 };
4. assert(normalizeRules(word5, word6, distance2, 6) == 1);
5. assert(strcmp(word5[0], "math") == 0);
6. assert(strcmp(word6[0], "weird") == 0);

INVALID RULE WITH NON-LETTER CHARACTER

1. char word7[20][MAX\_WORD\_LENGTH + 1] = { "WHat","Man","man","shittY","JuMpy","man" ,"WhaT","mAn","whAt","what","shitty" };
2. char word8[20][MAX\_WORD\_LENGTH + 1] = { "mAN","Venture","venture","what","wHat","what","jumpY","venTUre","jUmpy" ,"man","what" };
3. int distance3[20] = { 5,4,2,4,7,9,3,2,3,4 };
4. assert(normalizeRules(word7, word8, distance3,11) == 4);

const int TEST1\_NRULES = 4;

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

"one", "one", "three", "six"

};

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

"two", "four", "one", "two"

};

int test1dist[TEST1\_NRULES] = {

3, 4, 1, 13

};

assert(calculateSatisfaction(test1w1, test1w2, test1dist, TEST1\_NRULES,

"one two Three foUr five Six seven EIGHT nine ten eleven twelve.") == 3);

assert(calculateSatisfaction(test1w1, test1w2, test1dist, TEST1\_NRULES,

"one two Three foUr five Six.") == 3);

assert(calculateSatisfaction(test1w1, test1w2, test1dist, TEST1\_NRULES,

"------ 9324 ------") == 0);

assert(calculateSatisfaction(test1w1, test1w2, test1dist, TEST1\_NRULES,

" one two : three FOUR!!!!") == 2);

//DIDN’T DO WELL, WHEN THE DOC STARTS WITH ‘ ‘, I WOULD CONSIDER IT AS A CSTRING

assert(calculateSatisfaction(test1w1, test1w2, test1dist, TEST1\_NRULES,

"one two one TWO three three Four!") == 2);

//DIDN’T DO WELL, FIND OUT THAT EACH CSTRING NEEDS A \0 IN ORDER NOT TO HAVE SOME WEIRD CHARACTER

assert(calculateSatisfaction(test1w1, test1w2, test1dist, TEST1\_NRULES,

"one&two&three&four&five&six is equal to 0000000") == 0);

cout << "All tests succeeded" << endl;

const int TEST2\_NRULES = -2;

assert(calculateSatisfaction(test1w1, test1w2, test1dist, TEST2\_NRULES,

"one two one TWO three three Four!") == 0);