Report on Project 5

a)Found it difficult to keep incrementing and decrementing while a row was deleted in normalizeRules. It was difficult to debug it because of the length. Took a long time to find logical errors in calculateSatisfaction.

b)Pseudocode: normalizeRules

{if (nRules <= 0)

return 0

Repeatedly check for empty strings in word1

{ If found then use bubble sort technique to delete row, increment nRules accordingly}

Repeatedly check for empty strings in word2

{ If found then use bubble sort technique to delete row, increment nRules accordingly}

Repeatedly convert all chars in word1 to lower case and check if any special characters exist

{ If found then delete row, increment nRules accordingly}

Repeatedly convert all chars in word2 to lower case and check if any special characters exist

{If found then delete row, increment nRules accordingly}

Repeatedly check if two words in the same array are the same

{ If found then check if the corresponding words in the other array match

{ If found delete row depending on distance and increment nRules accordingly}

}

Repeatedly check if two words in different arrays match

{ If found then check if the corresponding words in the other arrays match

{ If found delete row depending on distance and increment accordingly}

}

return nRules;}

Pseudocode: calculateSatisfaction

{

if (nRules < 0) return 0;

create a new 1-D array new doc without non alphabetic characters

By repeatedly checking the document

Add to doc when char is alphabet or space, increment word count after every space

Create a new 2-D array doc with only the normal words and no special characters

By repeatedly adding to the doc by checking the 1D array new doc

Add a new word will one word till you hit zero byte

Repeatedly:

Check is word1 exist in the doc

If found check for word2

If found calculate the distance between them

If distancebtw <= distance

Increment satisfactionscore;

return satisfactionscore;

}

c)

Test data for calculateSatisfaction:

const int TEST1\_NRULES = 4;

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

"mad", "deranged", "nefarious", "have"

};

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

"scientist", "robot", "plot", "mad"

};

int test1dist[TEST1\_NRULES] = {

2, 4, 1, 13

};

1. (test1w1, test1w2, test1dist, TEST1\_NRULES, "The mad UCLA scientist unleashed a deranged evil giant robot.") - checks basic working of the function with special characters
2. (test1w1, test1w2, test1dist, TEST1\_NRULES, "The mad UCLA scientist unleashed a deranged robot.") - checks basic working of the function with multiple spaces in between
3. (test1w1, test1w2, test1dist, TEST1\_NRULES, "\*\*\*\* 2016 \*\*\*\*") - checks function when all special characters
4. (test1w1, test1w2, test1dist, TEST1\_NRULES, " That plot: NEFARIOUS!") - check the function when starts with spaces and has special characters
5. (test1w1, test1w2, test1dist, TEST1\_NRULES,"deranged deranged robot deranged robot robot") - checks the function when function is satisfied multiple times with the same rule
6. (test1w1, test1w2, test1dist, TEST1\_NRULES, "That scientist said two mad scientists suffer from deranged-robot fever.") - basic check with special characters in between words
7. (test1w1, test1w2, test1dist, TEST1\_NRULES, "have scientist") - checks when document is has no satisfaction
8. (test1w1, test1w2, test1dist, TEST1\_NRULES, "scientist mad deranged robot nefarious plot have mad") - checks when all rules are satisfied
9. (test1w1, test1w2, test1dist, 0, "The mad UCLA scientist unleashed a deranged evil giant robot") - tests when nRules is 0
10. (test1w1, test1w2, test1dist, -21, "The mad UCLA scientist unleashed a deranged evil giant robot.") - tests when nRules is negative
11. (test1w1, test1w2, test1dist, TEST1\_NRULES, "I am prof. nefarious") - checks for out of range when first word is found

Test data for normalizeRules:

int testdid[5] = { 1, 2, 3, 4, 5};

char word1[5][MAX\_WORD\_LENGTH+1] = {"cat", "dog", "cow", "pig", "chicken"};

char word2[5][MAX\_WORD\_LENGTH+1] = {"dog", "cat", "cow", "horse", "sheep"};

int test2dist[TEST1\_NRULES] = {

2, 4, 1, 13, 12

};

char word3[5][MAX\_WORD\_LENGTH+1] = {"cat", "dog", "cow", "pig", "chicken"};

char word4[5][MAX\_WORD\_LENGTH+1] = {"dog", "cat", "cow", "horse", "sheep"};

int test3dist[5] = {

2, 4, 1, 13,12

};

char word5[5][MAX\_WORD\_LENGTH+1] = {"c-at", "dog", "co!w", "pig", "chicken"};

char word6[5][MAX\_WORD\_LENGTH+1] = {"dog", "cat", "cow", "horse", "sheep"};

int testdis[10] = {1, 3, -1, 9, 8};

char word9[5][MAX\_WORD\_LENGTH+1] = {"cat", "dog", "cow", "pig", "chicken"};

char word10[5][MAX\_WORD\_LENGTH+1] = {"dog", "cat", "cow", "horse", "sheep"};

int testdis1[10] = {1, 3, -1, 9, 8};

char word11[5][MAX\_WORD\_LENGTH+1] = {"", "dog", "cow", "pig", "chicken"};

char word12[5][MAX\_WORD\_LENGTH+1] = {"dog", "cat", "cow", "horse", "sheep"};

1. assert(normalizeRules(word1, word2, testdid, 5) == 4); // repeating rules & same rules
2. assert(normalizeRules(word3, word4, test2dist, 0) == 0); // nRules is 0
3. assert(normalizeRules(word3, word4, test2dist, -1) == 0); // nRules is negative
4. assert(normalizeRules(word5, word6, test3dist, 5) == 3); // special characters
5. assert(normalizeRules(word7, word8, test4dist, 7) == 4); // basic check
6. assert(normalizeRules(word9, word10, testdis, 5) == 3) // when dis is negative
7. assert(normalizeRules(word11, word12, testdis1, 5) == 3); // empty string