Project 5 Report - Angela Kan

1. OBSTACLES: I found the logic for writing cleanupRules to be a lot more difficult than determineScore. Most notably, I had a lot of trouble determining how to update the i/j indices after removing a match rule for each specific removal. It involved quite a bit of tricky logic, but I managed to work through it and figure it out by drawing out match rule scenarios on a piece of paper and talking through what should happen. I also got pretty stuck with removing all duplicates of the w-in word of a one-word rule, as originally I only checked the rules after the specified one-word rule at i we were checking the other rules against, but I forgot that two-word rules before index i could also have the same w-in word and would thus need to be removed. After realizing this error, I went back and updated my code for this, making sure to update the index of i as well for this specific situation.
2. PSEUDOCODE:

rotateLeft:

for the length of n:

shift all elements after specified pos one over to left for both char arrays

cleanupRules:

if nRles negative or zero, return 0

otherwise:

for the number of rules:

for each word in w-in/w-out, change uppercase to lower-case

iterate through each letter in each word:

if not alpha/space char:

remove match rule and update numbers/index

then break from loop

iterate through each string:

if w-in word equals empty string

remove match rule and update numbers/index

iterate through each string:

check all other rules against the string

if other rules w-in match one-word rule w-in:

remove match rule and update numbers/index

if removed rule is less than comparison string:

update i

iterate through each string:

check all rules after the specified string

if duplicate two-word rule:

remove match rule and update numbers/index

return number of clean rules

determineScore:

if nRules negative or zero, return 0

make a copy of the document, max size +1:

add spaces to beginning and end

iterate through the doc:

convert all upper case letters to lower case

remove all non-letters/non-spaces by shifting one over to left

check for empty doc:

return 0

iterating through for the number of rules in match rules:

create a temp string with spaces at beginning and end for w-in word

if wordin word is present as a substring in the document:

if no second rule:

add one to score

if second rule:

create a temp string with spaces at beginning and end for w-in word

if wordout word is NOT present as a substring in the document:

add one to score

return score of doc

1. TEST CASES:

*cleanupRules:*

const int TEST1\_NRULES = 12;

char test1win[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = {

"confusion", "FAMILY", "charm", "hearty", "house", "worn-out", "family", "charm", "ties", "", "charm", "FaMiLy"

};

char test1wout[TEST1\_NRULES][MAX\_WORD\_LENGTH+1] = {

"", "TIES", "confusion", "hearty", "intrigue", "younger", "first", "", "family", "frightened", "", "tIeS"

};

Tests for all 5 conditions in varying order: assert(cleanupRules(test1win, test1wout, 12) == 6);

const int TEST2\_NRULES = 12;

char test2win[TEST2\_NRULES][MAX\_WORD\_LENGTH+1] = {

"hearty", "confusion", "charm", "FAMILY", "house", "worn-out", "family", "charm", "ties", "", "charm", "FaMiLy"

};

char test2wout[TEST2\_NRULES][MAX\_WORD\_LENGTH+1] = {

"hear-ty", "", "confusion", "TIES", "intrigue", "younger", "first", "", "family", "frightened", "", "tIeS"

};

Non-alpha character at i=0: assert(cleanupRules(test2win, test2wout, 12) == 6);

Negative nRules: assert(cleanupRules(test2win, test2wout, -1) == 0);

nRules=0: assert(cleanupRules(test2win, test2wout, 0) == 0);

char test3win[7][MAX\_WORD\_LENGTH+1] = {

"hOW", "how", "how", "FAMILY", "house", "how", "house"

};

char test3wout[7][MAX\_WORD\_LENGTH+1] = {

"hearty", "confusion", "", "TIES", "intrigue", "hearty", ""

};

Multiple rules with same w-in word BEFORE the one-word rule, uppercase char at i=0, one-word rule at last position: assert(cleanupRules(test3win, test3wout, 7) == 3);

char test4win[6][MAX\_WORD\_LENGTH+1] = {

"who", "confusion", "charm", "FAMILY", "house", "charm"

};

char test4wout[6][MAX\_WORD\_LENGTH+1] = {

"who", ";h", "", "TIES", "intrigue", "younger"

};

Duplicate w-in word at last position, w-in = w-out word at i=0, non-alpha char at start of string: assert(cleanupRules(test4win, test4wout, 6) == 3);

char test5win[3][MAX\_WORD\_LENGTH+1] = {

"", "mzs:", "charm"

};

char test5wout[3][MAX\_WORD\_LENGTH+1] = {

"", "", "charm"

};

No set of match rules is clean: assert(cleanupRules(test5win, test5wout, 3) == 0);

*determineScore:*

const int TESTa\_NRULES = 3;

char testawin[TESTa\_NRULES][MAX\_WORD\_LENGTH+1] = {

"family", "unhappy", "horse",

};

char testawout[TESTa\_NRULES][MAX\_WORD\_LENGTH+1] = {

"", "horse", "",

};

Tests for one-word rule, two-word rule, absence of one-word ruel: assert(determineScore("Happy families are all alike; every unhappy family is unhappy in its own way.",

testawin, testawout, TESTa\_NRULES) == 2);

Absence of one-word rule, two word rule w-in and w-out present: assert(determineScore("Happy horses are all alike; every unhappy horse is unhappy in its own way.",

testawin, testawout, TESTa\_NRULES-1) == 0);

Presence of one one-word rule: assert(determineScore("Happy horses are all alike; every unhappy horse is unhappy in its own way.",

testawin, testawout, TESTa\_NRULES) == 1);

Non-alpha characters present, one word rule applies: assert(determineScore("A horse! A horse! My kingdom for a horse!",

testawin, testawout, TESTa\_NRULES) == 1);

Numbers/non-alpha char present in between one-word rule word: assert(determineScore("horse:stable ratio is 10:1",

testawin, testawout, TESTa\_NRULES) == 0);

Non-alpha characters only: assert(determineScore("\*\*\*\* 2020 \*\*\*\*",

testawin, testawout, TESTa\_NRULES) == 0);

nRules = 0: assert(determineScore("d", testawin, testawout, 0) == 0);

nRules = negative: assert(determineScore("d", testawin, testawout, -1) == 0);

char testbwin[4][MAX\_WORD\_LENGTH+1] = {

"family", "hi", "wow", "whoooooooooooooooooo"

};

char testbwout[4][MAX\_WORD\_LENGTH+1] = {

"", "horse", "", "",

};

Capitalized one-word rule as first word: assert(determineScore("FAMILy first", testbwin, testbwout, 4) == 1);

Extra spaces, one-word rule: assert(determineScore("family family family first", testbwin, testbwout, 3) == 1);

Empty string: assert(determineScore("", testbwin, testbwout, 4) == 0);

Substring of word exists but not as own word + special characters in one-word rule: assert(determineScore("famiLya fir;;;;d wow;;;;;", testbwin, testbwout, 4) == 1);

Extra spaces, one word rule is first word: assert(determineScore("hi man", testbwin, testbwout, 4) == 1);

Doc is just spaces: assert(determineScore(" ", testbwin, testbwout, 4) == 0);

250 char doc with special char: assert(determineScore("----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------", testbwin, testbwout, 4) == 0);

20 char word that matches one-word rule: assert(determineScore("whoooooooooooooooooo", testbwin, testbwout, 4) == 1);