Harsh Chobisa

David Smallberg

CS31

November 17, 2018

Project 5 Report

This project was clearly the most difficult project we have had thus far in CS31. I struggled a lot with both functions. For makeProper, my function would not work properly if I had to delete two consecutive items. I realized that this was because I was not properly controlling the iteration flow of the function, at that every time I deleted an item, I had to decrement the iteration counter. My next issue was conceptually understanding how to create makeProper. Eventually, I settled on created a new array with every word from document in it, and then using the positions of the words in that array to calculate the score.

My program has three main parts, the first of which is a deleteElement function. This function takes in all three arrays, the number of elements in each array, and the position to be deleted.

If position to be deleted is the last one in the array

Decrease variable representing length of array by one

Else

Shift everything in the array to the left by one position starting at the position to be deleted

Decrease variable representing length of array by one

Return new length of array

The next part of the program is the makeProper function. The function removes any element from the three arrays that does not fit the requirements. Pseudocode follows:

Make everything lowercase

If any separation is negative or any element in the arrays is not an array

Delete element

For every element in word1

Cycle through every element in word2

If word1 and word2 and equal for two different positions

Delete the one with the smaller separation

Now cycle through the two arrays again and check for repeats in the opposite order

Return the new size of the arrays

The final part of the program is the rate function which returns the score of a document given its pattern and separation arrays.

Make document lowercase

Remove every element of document that is not a letter or a space

Initialize array of 251 elements

Cycle through document

Store position of first letter of each word in document in this array

Initialize new array

Cycle through array of first letter positions

For every element, find the word after it

Store this word into the new array

Initialize score to zero

Cycle through document

Record position of word1 in array of words

Record position of word2 in array of words

Find absolute value of difference of positions of word1 and word2

Add this value to score

Return score

Test cases for rate:

**const** **int** RATE1\_NRULES = 5;

**char** rate1w1[RATE1\_NRULES][MAX\_WORD\_LENGTH+1] = {

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

};

**char** rate1w2[RATE1\_NRULES][MAX\_WORD\_LENGTH+1] = {

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

};

**int** rate1dist[RATE1\_NRULES] = {

1, 3, 0, 12 , 1

};

General test case to make sure mvp has been implemented

assert(rate("The mad UCLA scientist unleashed a deranged evil giant robot.", rate1w1, rate1w2, rate1dist, RATE1\_NRULES) == 2);

Another general test case:

assert(rate("The mad UCLA scientist unleashed a deranged robot.",

rate1w1, rate1w2, rate1dist, RATE1\_NRULES) == 2);

test case if there are no matching elements:

assert(rate("\*\*\*\* 2018 \*\*\*\*",

rate1w1, rate1w2, rate1dist, RATE1\_NRULES) == 0);

test case to ensure that not alphabetic characters are removed

assert(rate(" That plot: NEFARIOUS!",

rate1w1, rate1w2, rate1dist, RATE1\_NRULES) == 1);

Complicated test case to ensure accurate results when there are multiple repeated words.

assert(rate("deranged deranged robot deranged robot robot",

rate1w1, rate1w2, rate1dist, RATE1\_NRULES) == 1);

Complicated test case to ensure accurate results when there are no pattern matches.

assert(rate("That scientist said two mad scientists suffer from deranged-robot fever.",

rate1w1, rate1w2, rate1dist, RATE1\_NRULES) == 0);

Test case to ensure accuracy when a pattern has the same word for both words.

assert(rate("have james have.",

rate1w1, rate1w2, rate1dist, RATE1\_NRULES) == 1);

Test case to ensure accuracy when a pattern has the same word for both words.

assert(rate("have mad have.", rate1w1, rate1w2, rate1dist, RATE1\_NRULES) == 2);

Test cases for makeProper:

const int nPatterns = 9;

General test case to ensure success of an mvp.

char test1w1[9][MAX\_WORD\_LENGTH+1] =

{

"mad", "deranged", "jam-es", "half-witted", "robot", "plot", "have", "have", "mad"

};

char test1w2[9][MAX\_WORD\_LENGTH+1] =

{

"scientist", "robot", "PLOT", "assistant", "deranged", "Nefarious", "mad", "have", "scientist"

};

int test1dist[] =

{

1, 3, 0, 2, 1 , 0 , 12 , 0, 3

};

assert(makeProper(test1w1, test1w2, test1dist, nPatterns)==5);

Another general test case to ensure success of an mvp.

char test2w1[9][MAX\_WORD\_LENGTH+1] =

{

"mad", "deranged", "NEFARIOUS", "half-witted", "robot", "plot", "have", "mad" , "NefARiOUs"

};

char test2w2[9][MAX\_WORD\_LENGTH+1] =

{

"scientist", "robot", "PLOT", "assistant", "deranged", "Nefarious", "mad", "scientist" , "pLoT"

};

int test2dist[] =

{

1, 3, 0, 2, 1 , -1 , 12 , 3 , 6

};

assert(makeProper(test2w1, test2w2, test2dist, nPatterns)==4);

Test case to ensure only one of the following strings would be left.

char test3w1[5][MAX\_WORD\_LENGTH+1] =

{

"hello", "heLLO", "hello", "HEllo", "heLLO"

};

char test3w2[5][MAX\_WORD\_LENGTH+1] =

{

"world", "WORLD", "world", "worLD", "WOrld"

};

int test3dist[5] =

{

1, 3, 0, 2, 4

};

assert(makeProper(test3w1, test3w2, test3dist, 5) == 1);

Test to see if the function would work properly if there is nothing to be removed.

char test4w1[1][MAX\_WORD\_LENGTH+1] =

{

"hello"

};

char test4w2[1][MAX\_WORD\_LENGTH+1] =

{

"world"

};

int test4dist[1] =

{

1

};

assert(makeProper(test4w1, test4w2, test4dist, 1) == 1);

Test to see if the function would properly be able to delete every element without undefined behavior.

char test5w1[1][MAX\_WORD\_LENGTH+1] =

{

"hell-o"

};

char test5w2[1][MAX\_WORD\_LENGTH+1] =

{

"world"

};

int test5dist[1] =

{

1

};

assert(makeProper(test5w1, test5w2, test5dist, 1) == 0);

Test to see if the function would properly be able to delete the last element without undefined behavior.

char test6w1[2][MAX\_WORD\_LENGTH+1] =

{

"hello", "he-llo"

};

char test6w2[2][MAX\_WORD\_LENGTH+1] =

{

"world", "hello"

};

int test6dist[2] =

{

1 , 7

};

assert(makeProper(test6w1, test6w2, test6dist, 2) == 1);

Test to see if the function would properly be able to delete the every element without undefined behavior.

char test7w1[2][MAX\_WORD\_LENGTH+1] =

{

"hello(z", "he-llo"

};

char test7w2[2][MAX\_WORD\_LENGTH+1] =

{

"world", "hello"

};

int test7dist[2] =

{

1 , 7

};

assert(makeProper(test7w1, test7w2, test7dist, 2) == 0);