Madeleine Murphy

004991129

CS 31

11/19/18

Project 5 Report

1. The major obstacles I overcame with this project had mostly to do with understanding what exactly I had to do, tracing through the spec to know how the function exactly needed to be implemented. For example, it took me a while to understand what exactly a proper pattern was, and how it would work in a document. For makeProper, I just needed to figure out how to get started and how exactly to work with C strings. I struggled with “deleting” the improper words in the string but then learned that I could just make everything in the string the ‘\0\. Visualizing the 2D array was also difficult for me at first, but once I figured out how to implement makeProper for a small part of the parameters, for example checking if word1 was in proper form, or checking if any word1 == another word1. Once I figured these out, it was easy to then apply it to anything involving word2.

In the rate function, I had a similar problem of just not quite knowing how to go about implementing the function. I decided that the best way was to create a new array that excluded non-letter characters and terminated words at each space so that I could later go through and check each word in the new “document”. Making the many imbedded for loops and if/else statements was very complicated and I had to go through line by line many times to make sure that it was checking the right words, characters, and breaking at the right places to get out and move on through the document. So essentially, my biggest obstacles were just figuring out how to tackle the problem and going little problem by little problem, debugging an deciphering exactly what needed to happen at each step.

1. **Design of makeProper:**

*If nPatterns negative, treat it as 0*

*Repeatedly for word1[0], word2[0], separation[0]...word1[i]...:*

*bool isBad = false*

*if a word1 contains no characters, isBad = true*

*if a word1 contains something other than a letter, isBad = true*

*else*

*make all letters in word1 lowercase*

*null terminate word1*

*repeat exact steps for word2*

*if separation is negative, isBad = true*

*if isBad = true*

*make all characters in word1 and word 2 ‘\0’*

*repeat until i = nPatterns(go through all patterns)*

*//now check if any patterns have same words:*

*repeatedly*

*if the first character is a null character, already know it’s a bad word, so break*

*if word1[i] equals any other word1 in pattern & word2 = that word2*

*if second one has lower separation*

*delete second (by making all characters ‘\0’)*

*else (if they have same separation or first has lower separation)*

*delete first*

*if word1 = another word2 in pattern, and first word2 = second word1 (they swap order but have same pattern)*

*if second one has lower separation*

*delete second*

*else (if they have same separation or first has lower separation)*

*delete first*

*create temp c string array 1*

*create temp c string array 2*

*create temp int array*

*repeatedly*

*copy all valid arrays into new c string arrays (first character not ‘\0’)*

*copy valid numbers that pair with valid strings into temp int array*

*count function holds place in temp array, increment count each time you add a valid string*

*if all strings are invalid (count = 0)*

*return 0*

*repeatedly*

*copy new arrays with valid strings into original arrays*

*return count*

**Design of rate:**

*if nPatterns is negative, treat as if it were 0*

*Create a temporary 2D array document*

*Repeatedly (for document[k])*

*if document[k] is a letter*

*put it in the new temp doc*

*Make all letters lowercase*

*Count characters incremented*

*if it’s a space*

*make it a nullbyte in the 2D array*

*increment word count - to the next word in the 2D array*

*start character count over*

*if it’s a null byte*

*make tempDoc character a nullbyte*

*increment word count (last word still counts)*

*break*

*set rate = 0*

*repeatedly*

*if word in the temp array starts with ‘\0’, it’s an empty word*

*count the empty words*

*if all the words are empty words, return 0*

*//now we calculate the rate*

*Repeatedly*

*Bool foundPattern = false*

*Start at the first word in the pattern*

*Repeatedly*

*If foundPattern = true, break (we want to move on to next word set in pattern)*

*If word1 equals a word in the document*

*If the word2 follows that word and it’s separated by less than or equal to the separation value that goes with the word*

*Add 1 to rate*

*foundPattern = true*

*break*

*If word2 equals a word in the document*

*If the word1 follows that word and it’s separated by less than or equal to the separation value that goes with the word*

*Add 1 to rate*

*foundPattern = true*

*break*

*return rate*

1. **TEST CASES:**

 //test with given complicated example that it reduces down to four

    char firstTest1[8][MAX\_WORD\_LENGTH+1] = {"mad", "deranged", "NEFARIOUS", "half-witted", "robot", "plot", "have", "NeFaRiOuS"};

    char firstTest2[8][MAX\_WORD\_LENGTH+1] = {"scientist", "robot", "PLOT", "assistant", "deranged", "Nefarious", "mad", "pLoT"};

    int separation1[8] = {1, 3, 0, 2, 1, 0, 12, 0};

    int nPatterns1 = 8;

    assert(makeProper(firstTest1, firstTest2, separation1, nPatterns1) == 4);

    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] = {

        1,           3,          0,           12

    };

    int z = rate("  That plot: NEFARIOUS!",

                 test1w1, test1w2, test1dist, TEST1\_NRULES);

            cout << z << endl;

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

                test1w1, test1w2, test1dist, TEST1\_NRULES) == 2);

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

                test1w1, test1w2, test1dist, TEST1\_NRULES) == 2);

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

                test1w1, test1w2, test1dist, TEST1\_NRULES) == 0);

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

                test1w1, test1w2, test1dist, TEST1\_NRULES) == 1);

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

                test1w1, test1w2, test1dist, TEST1\_NRULES) == 1);

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

                test1w1, test1w2, test1dist, TEST1\_NRULES) == 0);

    assert(rate("The mad UCLA scientist unleashed a deranged evil giant robot.", test1w1, test1w2, test1dist, TEST1\_NRULES) == 2);

    int test2dist[TEST1\_NRULES] = {

        1, -1, 0, 12};

    assert(makeProper(test1w1, test1w2, test2dist, TEST1\_NRULES) == 3); //check that it gets rid of negative values

     int test4dist[TEST1\_NRULES] ={

        -1, -1, -1, -12};

    int test5dist[TEST1\_NRULES] ={

        -1, -1, 0, -12};

    assert(makeProper(test1w1, test1w2, test5dist, TEST1\_NRULES) == 1); //check if one value is still proper

    assert(makeProper(test1w1, test1w2, test4dist, TEST1\_NRULES) == 0); //check that it gets rid of negative values

    const int TEST2\_NRULES = 7;

    char test2w1[TEST2\_NRULES][MAX\_WORD\_LENGTH+1] = { "rick", "morty", "beth", "summer", "birdperson", "mr", "wubbalubbadubdub" };

    char test2w2[TEST2\_NRULES][MAX\_WORD\_LENGTH+1] = { "morty", "jessica", "husband", "popular", "tammy", "poopybutthole", "rick"};

    int test3dist[TEST2\_NRULES] = {1, 2, 3, 3, 2, 0, 5};

    assert(makeProper(test2w1, test2w2, test3dist, TEST2\_NRULES) == 7);

    assert(rate("Rick and morty love it when rick says wubbalubbadubdub", test2w1, test2w2, test3dist, TEST2\_NRULES) == 2);

    assert(rate("Summer is the most popular girl in school", test2w1, test2w2, test3dist, TEST2\_NRULES) == 1);

    assert(rate("Morty is in love with Jessica", test2w1, test2w2, test3dist, TEST2\_NRULES) == 0);

    assert(rate("Beth hates her husband. Mr. Poopybutthole says 'oooweee' a lot!", test2w1, test2w2, test3dist, TEST2\_NRULES) == 2);    //check with periods and other non-letter characters in between

    assert(rate("rick morty morty jessica beth husband summer popular birdperson tammy mr poopybutthole rick wubbalubbadubdub", test2w1, test2w2, test3dist, TEST2\_NRULES) == 7);//check if all words are in there

    assert(rate("Hello, how are you rick?", test2w1, test2w2, test3dist, TEST2\_NRULES) == 0); //check if one word is in there

    assert(rate("Rick says wubba lubba dub dub", test2w1, test2w2, test3dist, TEST2\_NRULES) == 0);  //check that recognizes spaces, doesn't cound as one word

    assert(rate("Morty morty", test2w1, test2w2, test3dist, TEST2\_NRULES) == 0); //both word1s and word2s, but for different pattern

    assert(rate("Birdperson married tam", test2w1, test2w2, test3dist, TEST2\_NRULES) == 0); //checks that it doesn't cound half of the word as the word

    assert(rate("rick is mortylicious. summer wants popularity", test2w1, test2w2, test3dist, TEST2\_NRULES) == 0); //checks if word is within another word, still doesn't count

    assert(rate("asdjhfadfajhdfahfrmortyasdjkfkadrick", test2w1, test2w2, test3dist, TEST2\_NRULES) == 0); //tests if two words are within a bunch of random characters

    assert(rate("87^\*rick\*4723984 morty", test2w1, test2w2, test3dist, TEST2\_NRULES) == 1); //tests that it ignores other random characters

    assert(rate("", test2w1, test2w2, test3dist, TEST2\_NRULES) == 0); //test empty string

    assert(rate("%^&\*()#!@$", test2w1, test2w2, test3dist, TEST2\_NRULES) == 0); //test bad string

    assert(rate("", test2w1, test2w2, test3dist, -1) == 0); //treats negative nPatterns as if its 0

    assert(rate("rick and morty", test2w1, test2w2, test3dist, -12) == 0); //treats negative nPatterns as if its 0

    const int TEST3\_NRULES = 4;

    char test3w1[TEST3\_NRULES][MAX\_WORD\_LENGTH+1] = {"a", "b\*b%& b7", "c", "a" };

    char test3w2[TEST3\_NRULES][MAX\_WORD\_LENGTH+1] = {"b", "b", "a", "c" };

    int test6dist[TEST3\_NRULES]={1, 1, 3, 1};

    assert(makeProper(test3w1, test3w2, test6dist, TEST3\_NRULES) == 2);

    assert(makeProper(test3w1, test3w2, test6dist, -1) == 0); //treat negative nPatterns as if it were 0

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