CS 31 Project 5 Report

**A) A Brief Description of Notable Obstacles**

I thought that this project was one of the harder projects so far, and a lot of this difficulty stemmed from having to only use cstrings. A major obstacle for me was simply becoming familiar and used to cstrings as I was working on the project, and many times I had to rewrite multiple parts of my code because it would not work for cstrings. After I got familiar with cstrings writing a lot of the basic functionality for the functions was not too bad, although the second function did take a while to fully work out.

However, a major obstacle was in making sure my program could handle all of the various test cases I threw at it. There were multiple instances where a failed test case forced me to rethink the central logic of my functions, and change their design. All in all I changed the design of my functions numerous times in order to account for everything, and that was a very long and tedious process. For some reason the debugging portion for this project took substantially longer when compared to the projects of before. However, I do now believe that my program has been complete debugged, and it should handle all test cases fine.

**B) Description of Program Design**

*Overview*

My program has two main functions: normalizeRules, which takes a set of rules, removes invalid ones, and normalizes the rest and calculateSatisfaction, which applies a set of rules to a documents and returns the number of rules that document satisfies. Each of these two functions relies on three helper functions; the first relies on two void and one integer function, and the second relies on two boolean and one integer function.

*normalizeRules*

normalizeRules can be thought of as having five parts. The first part lowercases all alpha characters, the second part checks if any word in a rule has no characters, the third part checks if any rule has a nonpositive distance, the fourth part checks if any word1 or word2 in a rule has a non-alpha character, and the fifth part removes duplicates based on the spec. Here is the normalizeRules pseudocode:

*if nRules is less than or equal to zero*

*return 0*

*increment through the set ((nRules / 2) + 1) times*

*increment through each rule*

*lowercase all alpha characters*

*if a word in this rule has no characters*

*delete this rule*

*if this rule has a nonpositive distance*

*delete this rule*

*if a word1 in this rule has a nonalpha character*

*delete this rule*

*if a word2 in this rule has a nonalpha character*

*delete this rule*

*return removeDuplicates*

It may not be obvious from the pseudocode but normalizeRules calls three other functions: lowerCaseRules, deleteRule, and removeDuplicates and their functions should be evident from their names. The first function normalizeRules calls is lowerCaseRules

*lowerCaseRules*

lowerCaseRules will take as input word1, word2, and nRules and lowercase all the alpha characters of all the words for all the rules. It returns void and here is the lowerCaseRules pseudocode:

*increment through each rule*

*increment through each character of word1 till the end*

*if it is an alpha character*

*lowercase it and store it back in word1*

*increment through each character of word2 till the end*

*if it is an alpha character*

*lowercase it and store it back in word2*

This simple yet key function ensures that all words in the rules are standardized in lowercase for any future comparison.

*deleteRule*

deleteRule takes as input word1, word2, distance, ruleCount, and rowPos and will send the rule, as indicated by rowPos, to the end of the array by swapping word1, word2, and distance with the rule marked at ruleCount - 1, and then decrement ruleCount by 1. Here is the deleteRule pseudocode:

*increment through each character of word1 of the rule at rowPos*

*swap each character of word1 of this rule with the word1 of the last rule*

*increment through each character of word2 of the rule at rowPos*

*swap each character of word2 of this rule with the word2 of the last rule*

*swap the respective distances*

This function is called multiple times throughout normalizeRules whenever an invalid rule is found and we need to delete it. Also after this function is called within normalizeRule, ruleCount is immediately decremented by one so that in future iterations the rule that was just swapped into the last position will not be swapped back.

*removeDuplicates*

removeDuplicates takes as input word1, word2, distance and ruleCount and will check if there are any duplicates among the rules and if so, will delete that rule. This deletion is extremely similar to how deleteRule works. Here is the removeDuplicates pseudocode:

*increment through the set ruleCount times*

*start from the first rule and increment through all of them*

*start from one rule ahead of the above start rule, and increment through to the end*

*check if the word1 of these two rules if the same*

*check if the word2 of these two rules is the same*

*check which rule has the smaller distance*

*delete the rule with the smaller distance*

*if the distance is the same*

*delete the second rule*

*if the word1 of the two rules were not the same, check if word1 and word2 of two rules are the same*

*check if word2 and word1 are the same*

*check which rule has the smaller distance*

*delete the rule with the smaller distance*

*if the distance is the same*

*delete the second rule*

Each time this function deletes a rule, it decrements ruleCount by one and changes swapped to true, so that we can keep track of how many valid rules there are and that a rule cannot be deleted twice. The function will then end by returning ruleCount.

*calculateSatisfaction*

The other main function is calculateSatisfaction, which essentially works by incrementing through each rule while incrementing through the entire document looking for any matches from the document to words in the given rule. If there is a match, it will then check if that rule is satisfied and if so, will increment score by one. Here is the calculateSatisfaction pseudocode:

*if nRules is less than or equal to zero*

*return 0*

*normalize the document and store it in a comparison document*

*increment through each rule*

*increment through the comparison document*

*collect the next word*

*check if this word is the same as word1 or word2 in the given rule*

*store the other word of this rule in a comparison*

*store the distance of this rule in an integer check if distance was changed*

*check if any of the words ahead of this word by distance or less is equal to comparison*

*increment score by one*

*if above is not true, check if any of the words behind of this word by distance or less is equal to comparison*

*increment score by one*

*return score*

It may not be obvious from the pseudocode but calculateSatisfaction calls three other functions: createComparisonDoc, forwardChecker, and backwardChecker and their functions should be evident their names. The first function calculateSatisfaction calls is createComparisonDoc.

*createComparisonDoc*

createComparisonDoc takes as input document and an empty cstring. It will remove all non-alpha characters other than spaces, lowercase all alpha characters, and store this result in the empty cstring and return the size of this cstring. Here is the createComparisonDoc pseudocode:

*increment through each character of the document*

*if it is an alpha character*

*lowercase it and store it in the new doc*

*if it is a space character*

*store it in the new doc*

*store a null character in the last position of the new doc*

*return new doc size*

This simple yet key function provides a standardized document for future reference.

*forwardChecker*

fowardChecker takes as input comparison document, position, rule word, rule word comparison, and the distance. It will then increment through words of the comparison document while the word count is less than or equal to distance, and compare these words against the rule word. If there is a match, it will return true; if not, return false. Here is the forwardChecker pseudocode:

*repeatedly while number of words checked is less than or equal to distance*

*store first ahead word in a comparison cstring*

*check if word is same as the word from the rule*

*if so, break out*

*return true if a match was found*

This function will return a boolean value back to calculateSatisfaction, and if this boolean value is true, score is incremented by one and the function moves onto the next rule.

*backwardChecker*

backwardChecker takes as input comparison document, position, rule word, rule word comparison, and the distance. It will then increment through the previous words of the comparison document while the word count is less than or equal to distance, and position is not negative. Then it compares these words against the rule word and if there is a match, it will return true; if not, return false. Here is the backwardChecker pseudocode:

*repeatedly while number of words checked is less than or equal to distance and position is not negative*

*store first previous word in a comparison cstring*

*flip around this word since it was collected in reverse order*

*check if word is same as the word from the rule*

*if so, break out*

*return true if a match was found*

This function will return a boolean value back to calculateSatisfaction, and if this boolean value is true, score is incremented by one and the function moves onto the next rule.

**Those are all the functions and their relevant descriptions for my program.**

**C) Test Data (86 Test Cases in Total)**

*normalizeRules (55 Test Cases)*

nRules Test Data [4 in Total]:

**Number of rules is negative**

*(someArray, anotherArray, someDistances, -2)*

**Number of rules is zero**

*(someArray, anotherArray, someDistances, 0)*

**Number of rules is positive**

*(someArray, anotherArray, someDistances, 4)*

**Number of rules is some large number**

*(someArray, anotherArray, someDistances, 99999999)*

distance Array Test Data [7 in Total]:

**At least one of the distances is negative**

*(someArray, anotherArray, someNegativeDistance, 4)*

**All of the distances are negative**

*(someArray, anotherArray, allNegativeDistances, 4)*

**At least one of the distances is zero**

*(someArray, anotherArray, someZeroDistance, 4)*

**All of the distances are zero**

*(someArray, anotherArray, allZeroDistances, 4)*

**Some of the distances are negative, zero, or positive**

*(someArray, anotherArray, mixedDistances, 4)*

**All the distances are either negative or zero**

*(someArray, anotherArray, nonPositiveDistances, 4)*

**All of the distances are positive**

*(someArray, anotherArray, positiveDistances, 4)*

Same w1 and w1 values Test Data [14 in Total]:

**Two separate rules have the same w1 and w2 values, in the same order, with the first one having a smaller distance value**

*(firstArray, secondArray, someDistances, 4)*

**Two separate rules have the same w1 and w2 values, in the same order, with the second one having a smaller distance value**

*(firstArray, secondArray, someDistances, 4)*

**Two separate rules have the same w1 and w2 values, in different orders, with the first one having a smaller distance value**

*(firstArray, secondArray, someDistances, 4)*

**Two separate rules have the same w1 and w2 values, in different orders, with the second one having a smaller distance value**

*(firstArray, secondArray, someDistances, 4)*

**Two separate rules have the same w1 and w2 values, in the same order, with the same distance value**

*(firstArray, secondArray, someDistances, 4)*

**Two separate rules have the same w1 and w2 values, in different orders, with the same distance value**

*(firstArray, secondArray, someDistances, 4)*

**All the rules have the same w1 and w2 values, in the same order, with the first one having the smallest distance value**

*(firstArray, secondArray, someDistances, 4)*

**All the rules have the same w1 and w2 values, in the same order, with one in the middle having the smallest distance value**

*(firstArray, secondArray, someDistances, 4)*

**All the rules have the same w1 and w2 values, in the same order, with the last one having the smallest distance value**

*(firstArray, secondArray, someDistances, 4)*

**All the rules have the same w1 and w2 values, in differing orders, with the first one having the smallest distance value**

*(firstArray, secondArray, someDistances, 4)*

**All the rules have the same w1 and w2 values, in differing orders, with one in the middle having the smallest distance value**

*(firstArray, secondArray, someDistances, 4)*

**All the rules have the same w1 and w2 values, in differing orders, with the last one having the smallest distance value**

*(firstArray, secondArray, someDistances, 4)*

**All the rules have the same w1 and w2 values, in the same order, all with the same distance value**

*(firstArray, secondArray, someDistances, 4)*

**All the rules have the same w1 and w2 values, in differing orders, all with the same distance value**

*(firstArray, secondArray, someDistances, 4)*

Capitalization Test Data [12 in Total]:

**One word in the first array of one rule has at least one capital character**

*(capitalArray, secondArray, someDistances, 4)*

**One word in the first array of one rule has all capital characters**

*(capitalArray, secondArray, someDistances, 4)*

**One word in the second array of one rule has at least one capital character**

*(firstArray, capitalArray, someDistances, 4)*

**One word in the second array of one rule has all capital characters**

*(firstArray, capitalArray, someDistances, 4)*

**Both words of one rule have at least one capital character**

*(capitalArray, capitalArray2, someDistances, 4)*

**Both words of one rule have all capital characters**

*(capitalArray, capitalArray2, someDistances, 4)*

**All words in the first array have at least one capital character each**

*(capitalArray, secondArray, someDistances, 4)*

**All words in the first array have all capital characters**

*(capitalArray, secondArray, someDistances, 4)*

**All words in the second array have at least one capital character each**

*(firstArray, capitalArray, someDistances, 4)*

**All words in the second array have all capital characters**

*(firstArray, capitalArray, someDistances, 4)*

**All words in both arrays have at least one capital character each**

*(capitalArray, capitalArray2, someDistances, 4)*

**All words in both arrays have all capital characters**

*(capitalArray, capitalArray2, someDistances, 4)*

Differing Characters Test Data [18 in Total]:

**One word in the first array of one rule has at least one non-alpha character**

*(nonAlphaArray, secondArray, someDistances, 4)*

**One word in the first array of one rule has all non-alpha characters**

*(nonAlphaArray, secondArray, someDistances, 4)*

**One word in the first array of one rule has no characters**

*(emptyWordArray, secondArray, someDistances, 4)*

**One word in the second array of one rule has at least one non-alpha character**

*(firstArray, nonAlphaArray, someDistances, 4)*

**One word in the second array of one rule has all non-alpha characters**

*(firstArray, nonAlphaArray, someDistances, 4)*

**One word in the second array of one rule has no characters**

*(firstArray, emptyWordArray, someDistances, 4)*

**Both words of one rule have at least one non-alpha character**

*(nonAlphaArray, nonAlphaArray2, someDistances, 4)*

**Both words of one rule have all non-alpha characters**

*(nonAlphaArray, nonAlphaArray2, someDistances, 4)*

**Both words of one rule have all no characters**

*(emptyArray, emptyArray2, someDistances, 4)*

**All words in the first array have at least one non-alpha character each**

*(nonAlphaArray, secondArray, someDistances, 4)*

**All words in the first array have all non-alpha characters**

*(nonAlphaArray, secondArray, someDistances, 4)*

**All words in the first array have no characters**

*(emptyArray, secondArray, someDistances, 4)*

**All words in the second array have at least one non-alpha character each**

*(firstArray, nonAlphaArray, someDistances, 4)*

**All words in the second array have all non-alpha characters**

*(firstArray, nonAlphaArray, someDistances, 4)*

**All words in the second array have no characters**

*(firstArray, emptyArray, someDistances, 4)*

**All words in both arrays have at least one non-alpha character each**

*(nonAlphaArray, nonAlphaArray2, someDistances, 4)*

**All words in both arrays have all non-alpha characters**

*(nonAlphaArray, nonAlphaArray2, someDistances, 4)*

**All words in both arrays have all no characters**

*(emptyArray, emptyArray2, someDistances, 4)*

*calculateSatisfaction (31 Test Cases)*

nRules Test Data [4 in Total]:

**Number of rules is negative**

*(someArray, anotherArray, someDistances, -2, someDocument)*

**Number of rules is zero**

*(someArray, anotherArray, someDistances, 0, someDocument)*

**Number of rules is positive**

*(someArray, anotherArray, someDistances, 4, someDocument)*

**Number of rules is some large number**

*(someArray, anotherArray, someDistances, 999999, someDocument)*

document Test Data [14 in Total]:

**Document contains one word**

*(someArray, anotherArray, someDistances, 4, oneWordDocument)*

**Document contains multiple words**

*(someArray, anotherArray, someDistances, 4, manyWordDocument)*

**Document is less than 200 characters**

*(someArray, anotherArray, someDistances, 4, smallDocument)*

**Document is 200 characters**

*(someArray, anotherArray, someDistances, 4, largeDocument)*

**Document contains at least one non-alphabetic characters**

*(someArray, anotherArray, someDistances, 4, mixedDocument)*

**Document only contains non-alphabetic characters**

*(someArray, anotherArray, someDistances, 4, nonAlphaDocument)*

**Document contains at least one set of words with multiple spaces in between them**

*(someArray, anotherArray, someDistances, 4, someSpaceDocument)*

**All words within document have multiple spaces in between them**

*(someArray, anotherArray, someDistances, 4, lotsOfSpaceDocument)*

**Document contains at least one word with an uppercase letter**

*(someArray, anotherArray, someDistances, 4, uppercaseDocument)*

**All words within document are upper case**

*(someArray, anotherArray, someDistances, 4, allUppercaseDocument)*

**Document contains at least one word longer than 20 characters**

*(someArray, anotherArray, someDistances, 4, longDocument)*

**All words within document are longer than 20 characters**

*(someArray, anotherArray, someDistances, 4, allLongDocument)*

**Number of words in document is less than at least one distance**

*(someArray, anotherArray, someDistances, 4, shortDocument)*

**Number of words in document is less than all distances**

*(someArray, anotherArray, someDistances, 4, allShortDocument)*

satisfaction Test Data [13 in Total]:

**No rules are satisfied**

*(someArray, anotherArray, someDistances, 4, noSatisfyDocument)*

**At least one rule is satisfied**

*(someArray, anotherArray, someDistances, 4, oneSatisfyDocument)*

**All rules are satisfied**

*(someArray, anotherArray, someDistances, 4, allSatisfyDocument)*

**At least one rule is satisfied multiple times**

*(someArray, anotherArray, someDistances, 4, oneManySatisfyDocument)*

**All rules are satisfied multiple times**

*(someArray, anotherArray, someDistances, 4, allManySatisfyDocument)*

**At least one rule is satisfied by upcoming words in the document**

*(someArray, anotherArray, someDistances, 4, oneForwardSatisfyDocument)*

**All rules are satisfied by upcoming words in the document**

*(someArray, anotherArray, someDistances, 4, allForwardSatisfyDocument)*

**At least one rule is satisfied by previous words in the document**

*(someArray, anotherArray, someDistances, 4, oneBackwardSatisfyDocument)*

**All rules are satisfied by previous words in the document**

*(someArray, anotherArray, someDistances, 4, allBackwardSatisfyDocument)*

**A set of words in at least one rule are separated by a number of words less than distance**

*(someArray, anotherArray, someDistances, 4, oneQuickSatisfyDocument)*

**Words for all rules are separated by a number of words less than distance**

*(someArray, anotherArray, someDistances, 4, allQuickSatisfyDocument)*

**A set of words in at least one rule are separated by a word longer than 20 characters**

*(someArray, anotherArray, someDistances, 4, oneLongDocument)*

**Words for all rules are separated by a word longer than 20 characters**

*(someArray, anotherArray, someDistances, 4, allLongDocument)*