1. Problems:

The notable obstacles that I had during normalizing the rules was having the entire program repeat checking a line after it was deemed not normal and replaced. Initially I had wanted it to break out of the if statement but that would require me to have it direct to another step. Instead, I ran all of the lines through each restriction separately. Another obstacle I faced was ensuring I was copying or comparing the right row (nRules-1 rather than nRules).

In calculating the satisfaction, I had a problem with a true value being any number and not just one for isalpha. Thus I had to set it equal to not 0. I also had problems with remembering which tests had already been satisfied.

Lastly, dealing with 2D arrays and remembering what each row represented was hard at first. Reading out rows was challenging. I think lastly creating arrays of proper sizes was difficult.

1. Pseudocode:

normalizeRules

Check each row

If the row word1 or word2 is empty (“”) then replace the entire row with the last of the rows.

Then repeat the loop for this row and decrease the number of rules accordingly.

Check each row

If the character is a letter then convert it to a lowercase letter.

If word1 contains a non-alphabet character, delete that entire row (word1/2/distance) and replace with the last of the rows.

Then repeat the loop for this row and decrease the number of rules accordingly.

Check each row

If the character is a letter then convert it to a lowercase letter.

If word2 contains a non-alphabet character, delete that entire row (word1/2/distance) and replace with the last of the rows.

Then repeat the loop for this row and decrease the number of rules accordingly.

Check each row

The distance must be greater than 0, if not delete that entire row (word1/2/distance) and replace with the last of the rows.

Then repeat the loop for this row and decrease the number of rules accordingly.

Check each row

Compare against each subsequent row

If the rows are identical then replace the row with the one that contains a larger distance and replace the smaller distance one with the very last row. Then repeat the loop for this row and decrease the number of rules accordingly.

Check each row \*\*Do above for the opposite strand if the word1 = word2 and vice versa for a row.

Return the rules total there are now.

CalculateSatisfaction

Create a copy of the document that we can edit.

Lowercase every letter in the document.

Remove every non-letter in the document.

Create a new array that can hold all the words of the document

Reasoning for the dimensions: the document can hold a 200 character word or 100 single words.

Create a temporary word that reads up to the space or end character

Copy this word into a new row of a new array.

\*this creates an array containing one row for every word\*

Create an array to add up how many times each rule has been satisfied.

Compare each row to the each word1

If the row = word1 then check if the word2 appears equal to row within the proper distance (measured by m).

Compare each row to the each word2

If the row = word2 then check if the word1 appears equal to row within the proper distance (measured by m).

Taking all the rules that were satisfied, only count the rule once if satisfied \*does this by counting which ones have more than 0 occurrences.

Return the number of rules satisfied.

1. Test cases:
   * Normalize Rules
     1. Test if all rules are normal
     2. Test if there is only one rule that isn’t normal
     3. Test if there are multiple rules that aren’t normal
     4. Abnormal rules:
        1. If there is an empty string
        2. If the string has non-letters
        3. If there are negative numbers
     5. Test if the program can convert all letters to lowercase
     6. Test if the program can differentiate between two identical strands and if they can properly delete the one with smaller distance.
     7. Test if the program can differentiate between two strands that have the same words but one in word1 and one in word2.
     8. Test for no rules.
     9. Test for one rule.
     10. Test for multiple rules.
   * calculateSatisfaction
     1. Test for a document with nothing.
     2. Test for a document that has only letters.
     3. Test for a document that only has non-letters.
     4. Test for a document that has a mix of letters and non-letters to see if it can properly read out only the letters.
     5. In the document:
        1. Test for when only one rule is satisfied.
        2. Test for when no rules are satisfied.
        3. Test when multiple rules are satisfied.
        4. Test when one rule is satisfied multiple times.