Plagrism Project Documentation

-> In this project we got 2(two) paragraphs and our program need to find similar sequence of words from those 2 paragraphs.

-> We will use dynamic programming approach to solve this problem.

-> We have created a class 'similarityInSentenseV3' and we will put all our methods in this class.

-> we have declared certain variables globally for ease and reduce complexity

-> In main method, first we got 2 String array which will store each word of the problem paragraph. Here I have used BufferedReader variable to take user input. Along with Arrays I have used 2 more String variable 'temp & 'piece'. -> If there are multiple line in paragraph then 'temp' will read lines one at a time along with converting it to lower case and removing all .(fullstop) and ,(commas) and will append it to 'piece' untill all lines are read.

-> after reading whole paragraph we will assign it to our first String array. Similar process for second paragraph.

-> After getting arrys we will pass this arrays to the method 'SIS'.

-> In SIS we created 2 Two-dimentional arrays 'path' and 'weight'. 'path' is used to keep track of directions(top, left, top-left). 'weight' is used to keep track of the length of sequence of words fill till any instance. The size of both this arrays will be (n x m) where n is length of paragraph 1 and m is length of paragraph 2.

-> Now we will call the method 'setupArrays' by passing array 'weight' and 'path'

-> Initially setupArrays method will fill the first column and row of 'weight' with (0) and 'path' with (#).

-> Now we will run a loop to traverse each index of array weight and path.

Outer loop will run for 1 to n and inner loop will run for 1 to m.

1. If St1(i) == St2(j) then we will put '/' in path[i][j] and weight[i][j] = weight[i-1][j-1] + 1;
2. else if weight[i][j-1] <= weight[i-1][j] then we will put '^' in path[i][j] and weight[i][j] = weight[i-1][j];
3. else weight[i][j] = weight[i][j-1] and we will put '<' in path[i][j]

-> This is how we will setup our arrays for further use.

-> Now we will call 'traverse' method to traverse our 2-D arrays from bottom to top.

if the index on which our pointer points has path[i][j] = '/' and also path[k-1][l-1] == '/' then will call another method 'findSeq' which will further look for sequence in same diagonal by decrementing row column index by 1.

-> I have created an array of ArrayList which stores the similar sequence found by using an index.

-> In 'findSeq' method, if the (i & j) index has reached (0) then we will simply return;

-> If the (i-1) & (j-1) index also contains '/' in path then we will recursively call the method findSeq with (i-1) & (j-1). Also after exploring the current index we will put '-' at path[i][j] to mark it as already visited to avoid redundancy. -> After returning from each recursive call we will append the word at ith index in our ArrayList[index].

-> if path[i-1][j-1] does not contains '/', then it means it's the end of the sequence and we should not call 'findSeq' further so we need to stop at current index. For this we will put '-' at current index and add word at this index to arraylist. After inserting whole sequence at an index of arrayList we will increment index for insertion of next sequence.

-> By the end of 'traverse' method we will have all the sequence of words in the arraylist which is needed to be printed so we will call 'print' method.

-> In print method I have created a String 'listString' which will hold whole sequence as a single string.

-> loop will run for arraylist's array length. At each index, if sequence is (et, gt, st) certain value then it will print the sequence.

-> For finding similarity I have divided the similar words found * 2 by total words in both string.

(similarWords * 2 / total_length) * 100