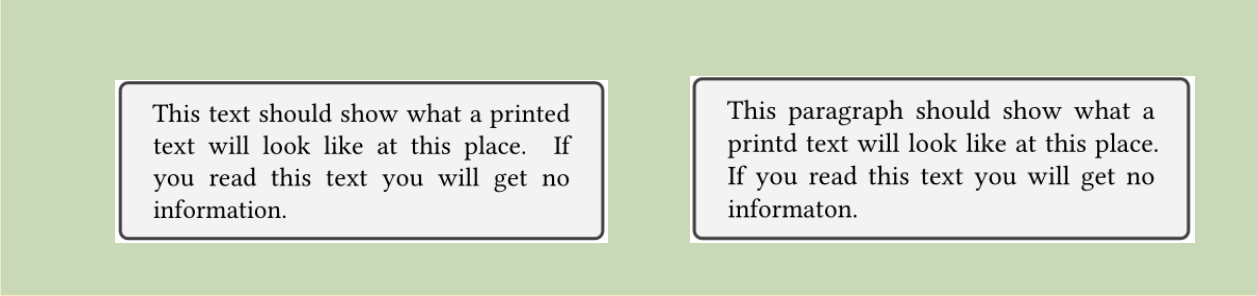
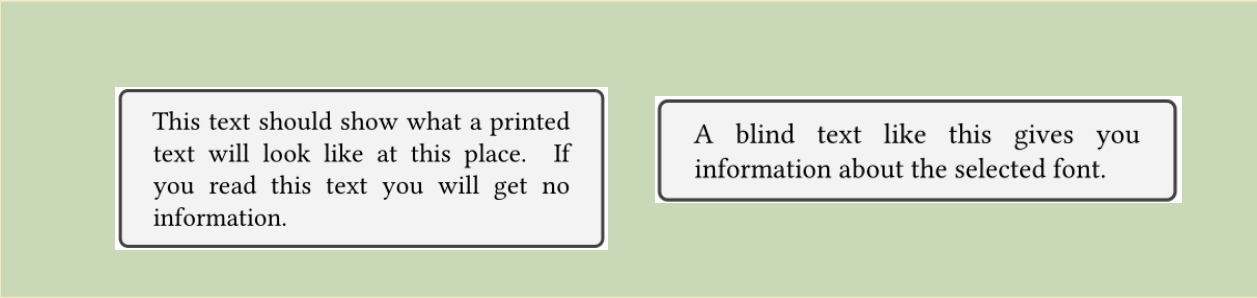
Final Project

Santiago Caballero Manzaneda

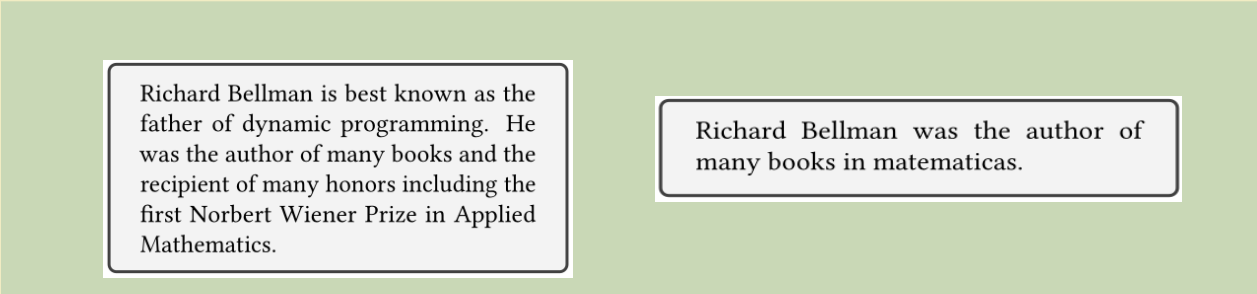
1. Make sure that you apply dynamic programming to your solution.



- The length of both texts is 24
- The misspelled word of “printd” it accurately has a similarity with “printed” more than 55% so it is considered as a word that would be included in the similarity percentage plagiarism.
- **23** represents the words found of the lcs addition of both sentences
- **24** represents the length of the min length between both texts
- **Formula**.- 23/24 * 100 = 95,83%



- The length of the first text is 24 and the second one is 12
- **4** represents the words of the longest common subsequence that was found in the second sentence of the first text “A, text, like, this” that is repeated
- **12** represents the length of the min length between both texts
- **Formula**.- 4/12 * 100 = 33,33%



- In this text the “Richard Bellman” is depreciated to calculate the plagiarism because the other words of the second text has a longer common subsequence with the words of the second sentences of the first text, It means that “8” words match with each-other in both texts so the first sentence is dismissed from the calculation.
- **8** represents the words of the longest common subsequence that was found
- **10** represents the length of the min length between both texts
- **Formula**.- 8/10 * 100 = 80,00%

2. Write a brief explanation of why you have chosen the DP algorithm to solve the problem.

I had chosen to implement the dynamic programming LCS (Longest Common Subsequence) algorithm for this problem because it efficiently computes the length of the longest common subsequence between two strings. This is crucial for comparing the textual content of sentences extracted from two files, aiding in the determination of plagiarism percentage. Additionally, the algorithm helps identify misspelled words by comparing if two words are at least 55% similar. By utilizing a bottom-up approach (tabulation), the LCS algorithm ensures effectiveness in calculating similarity percentages. While alternative approaches like the Levenshtain edit distance algorithm could be considered, they may not be as efficient for this specific task.

3. **Identify the time complexity of your solution.**

The algorithm implemented uses tabulation where a table of dp[l1][l2] has to be filled. has a complexity of O(lenghtA*lenghtB) where lenghtA is the length of the first text and lengthB is the length of the second one.

4. Answer the following question:
(a) Is there a non-dynamic programming solution? if so, please explain the idea and compare it with your solution.

Yes I think there are without using dynamic programming like the force-brute way you could just go through each word in one text and compare it with every word in the other text, checking if they're similar or not but that is slow, especially with big texts. The LCS thing is better because it's optimized and faster. It figures out the longest common subsequence between the two texts, making it way faster to compare and spot misspelled words, so I would use dp for this for sure.