Heuristics

Heuristics that can be used for calculating text distance

reference: https://towardsdatascience.com/3-text-distances-that-every-data-scientist-should-know-7fcdf850e510

First approach: Hamming Distance

source: https://www.geeksforgeeks.org/hamming-distance-two-strings/

This worked pretty well for smaller length strings but took a lot of time for longer strings, thus leading me to find another approach

In [9]:
```
def heuristic(self,a, b):
        return sum(1 for i in range(len(a)) if a[i] != b[i])
```

Second approach: Levenshtien Distance/ Edit Distance,

source: https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0

problem reference: https://rosalind.info/problems/edta/,

code reference: https://github.com/zonghui0228/Rosalind-Solutions/blob/master/code/rosalind_edta.py

This was better than Hamming Distance since the time taken to calculate the heuristic for longer strings was way less than Hamming, though, it was not able to find the optimal solution for one of the given test case "THECLASSROOM" and "SCHOOLMASTER"

My Code

In [10]:
```
    def heuristic(self,str1, str2):
        m = len(str1)
        n = len(str2)
        dp = [[0] * (n + 1) for _ in range(m + 1)]

        for i in range(m + 1):
            for j in range(n + 1):
                if i == 0:
                    dp[i][j] = j
                elif j == 0:
                    dp[i][j] = i
                elif str1[i - 1] == str2[j - 1]:
                    dp[i][j] = dp[i - 1][j - 1]
                else:
                    dp[i][j] = 1 + min(dp[i][j - 1], dp[i - 1][j], dp[i - 1][j - 1])

        return dp[m][n]
```

Reference Code

In [11]:
```
def EditDistanceAlignment(s, t):
    m, n = len(s), len(t)
    if m*n==0:
        return m+n
    DP = [[0]*(n+1) for _ in range(m+1)]
    for i in range(m+1):
        DP[i][0] = i
    for j in range(n+1):
```

```
        DP[0][j] = j
    for i in range(1, m+1):
        for j in range(1, n+1):
            left = DP[i-1][j] + 1
            down = DP[i][j-1] + 1
            left_down = DP[i-1][j-1]
            if s[i-1] != t[j-1]:
                left_down += 1
            DP[i][j] = min(left, down, left_down)
    # pprint.pprint(DP)
    edit_distance = DP[m][n]
```

A Star

To compare the expanded combinations of a given string

Approach 1: Using List

This is the most basic approach, to add the exapnded combination set along with calculated f(n) to the list. sort the list according to f(n). But this approach was not the ideal or even the most effcient approach since it took a lot of time to sort the given list.

Approach 2: Using Heap

Heapq library in python automatically sorts the items as they are pushed in to the heap data structure, and using the pop method to get the item with the least f(n).

The time difference in sorting is nothing but time difference in linear sort vs heap sort.

About code generated by ChatGPT

Code generated by ChatGPT for the prompt: "Write a code in python for levenshtien distance"

First Generation:

In [1]:
```
def levenshtein_distance(s1, s2):
    # Get the length of the two input strings
    len_s1 = len(s1)
    len_s2 = len(s2)

    # Create a matrix to store the distances between each substring of the two input strings
    distance_matrix = [[0] * (len_s2 + 1) for _ in range(len_s1 + 1)]

    # Initialize the matrix
    for i in range(len_s1 + 1):
        distance_matrix[i][0] = i
    for j in range(len_s2 + 1):
        distance_matrix[0][j] = j

    # Compute the distances between the substrings of the two input strings
    for i in range(1, len_s1 + 1):
        for j in range(1, len_s2 + 1):
            if s1[i - 1] == s2[j - 1]:
                substitution_cost = 0
            else:
                substitution_cost = 1

            distance_matrix[i][j] = min(distance_matrix[i - 1][j] + 1,
                                        distance_matrix[i][j - 1] + 1,
                                        distance_matrix[i - 1][j - 1] + substitution_cost)
```

```
    # Return the distance between the two input strings
    return distance_matrix[len_s1][len_s2]
```

Second Generation:

In [2]:
```
def levenshtein_distance(s, t):
    m, n = len(s), len(t)
    if m < n:
        s, t = t, s
        m, n = n, m

    previous_row = range(n+1)
    for i, c1 in enumerate(s):
        current_row = [i+1]
        for j, c2 in enumerate(t):
            insertions = previous_row[j+1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row

    return previous_row[n]
```

Code generated by ChatGPT for the prompt : "Write a code to find longest common sequence between 2 words in python"

In [3]:
```
def longest_common_sequence(word1, word2):
    m = len(word1)
    n = len(word2)
    # initialize the dynamic programming table
    dp = [[0] * (n + 1) for _ in range(m + 1)]
    # fill in the table using dynamic programming
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if word1[i - 1] == word2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
    # trace back the longest common sequence
    lcs = ''
    i, j = m, n
    while i > 0 and j > 0:
        if word1[i - 1] == word2[j - 1]:
            lcs = word1[i - 1] + lcs
            i -= 1
            j -= 1
        elif dp[i - 1][j] > dp[i][j - 1]:
            i -= 1
        else:
            j -= 1
    return lcs
```

Code generated by ChatGPT for the prompt : "Write a code to find the distance between 2 words using the longest common sequence in python"

In [4]:
```
def LCS(s1, s2):
    # initialize a 2D array with 0's
    m = len(s1)
    n = len(s2)
    dp = [[0] * (n+1) for _ in range(m+1)]

    # fill the array with the lengths of LCS
    for i in range(1, m+1):
        for j in range(1, n+1):
```

```
            if s1[i-1] == s2[j-1]:
                dp[i][j] = dp[i-1][j-1] + 1
            else:
                dp[i][j] = max(dp[i-1][j], dp[i][j-1])

    # return the length of LCS
    return dp[m][n]

def distance(s1, s2):
    # find the length of the longest common subsequence
    lcs_len = LCS(s1, s2)

    # the distance is the sum of the lengths of s1 and s2
    # minus twice the length of LCS
    return len(s1) + len(s2) - 2 * lcs_len
```

As you can see, there is an inherent bias in the codex for ChatGPT in terms of generating a similar sort of code, if I had taken the code by using ChatGPT, it should've been similar to the prompts in the beginning than later. ChatGPT is not a reliable source to generate code, given enough regenerations, there is a similarity bound to happen between code written by humans and code generated by ChatGPT (which is trained on data written by humans :)). Just like Infinite Monkey Theorem :

The infinite monkey theorem states that a monkey hitting keys at random on a typewriter keyboard for an infinite amount of time will almost surely type any given text, such as the complete works of William Shakespeare. In fact, the monkey would almost surely type every possible finite text an infinite number of times.

source: https://en.wikipedia.org/wiki/Infinite_monkey_theorem

Another point to prove that there could be more than one way to do a certain problem is an example from Wikipedia showing the approaches that can be used to write the same algorithm. link: https://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Levenshtein_distance#:~:text=LANGUAGE%%5Bedit%20%7C

And according to the professor's email, and the code given above, the only reason there is a similarity between the given examples (My code, Student A's code and GPT's code) is the initialization of variables, matrix, and the comparison of string indices.

In [ ]:
```
# Common Code

m = len(str1)
n = len(str2)
dp = [[0] * (n + 1) for _ in range(m + 1)]

str1[i - 1] == str2[j - 1]:
```

Also, according to the professor's email, ChatGPT recommends Levenshtein Distance for A Star Algorithm where strings are anagrams.

Code generated by ChatGPT for the prompt : "Write a python program to give the minimum steps between two strings using a star algorithm where given strings are anagram"

First Generation

```
As an AI language model, I cannot write code in Python without a specific prompt or context. Please provide more details or a prompt for me to understand the specific problem that you want to solve.
```

Second Generation

In [6]:
```python
from collections import Counter

def string_distance(s1, s2):
    """Find the minimum number of steps to transform s1 into s2"""
    # Check if the two strings are anagrams
    if Counter(s1) != Counter(s2):
        return -1

    # Set up initial variables for A* algorithm
    nodes = [(0, s1)]
    visited = set()

    while nodes:
        # Sort the nodes based on estimated cost
        nodes.sort()

        # Pop the node with the lowest estimated cost
        cost, node = nodes.pop(0)

        # Check if we've reached the target node
        if node == s2:
            return cost

        # Add the node to the visited set
        visited.add(node)

        # Generate possible child nodes
        for i in range(len(node)):
            for j in range(i + 1, len(node)):
                # Swap the letters at positions i and j
                child = list(node)
                child[i], child[j] = child[j], child[i]
                child = ''.join(child)

                # Check if we've already visited this node
                if child in visited:
                    continue

                # Calculate the estimated cost of reaching the target node
                est_cost = sum(1 for a, b in zip(child, s2) if a != b)

                # Add the child node to the nodes list
                nodes.append((cost + 1 + est_cost, child))

    # If we've exhausted all possible nodes without reaching the target, the strings are not a
    return -1
```

Even if we go with the professor's claim that ChatGPT is legit. The first Generation does not even generate any code and in the second generation, we can clearly see no use of Levenshtien Distance.

In [ ]: