Assignment 5 – Harshit Mawandia

Q1. Play with Grid:

Our main function is *gridPlay* and we use do not use any helper functions.

2. Time Complexity Analysis:

Let N be the number of rows and M be the number of columns.

The outer loop iterates N times, and the inner loops iterates M times for each iteration of the outer loop. We do O(1) calculations for each inner loop iteration. So for total N x M iteration of the inner loop, the time complexity of the algorithm will be O(NM)

3. Correctness Proof:

gridPlay(grid):

First, we initialise n with the number of rows and m with the number of columns. The outer loop runs from i = 0 to i = n-1 which gives us n iterations and for each such iteration inner loop runs m time from j = 0 to j= m-1 similarly.

At the start of the grid you have to take the Initial penalty, therefore S[0][0] is equal to grid[0][0].

If we are in the first row, there is only one way of reaching a tile, which is from the tile to its immediate right. So for reaching the j^{th} tile (index = j-1) in the first row, the penalty S[0][j-1] = penalty to reach S[0][j-2] + penalty of the j^{th} tile, i.e. S[0][j-1] = S[0][j-2] + grid[0][j-1]

Similarly, for the first column, S[i-1][0] = S[i-2][0] + grid[i-1][0]

For every other tile, we can either reach from above, from the left or from diagonally upper left tile, but we have to reach gaining minimum penalty, therefore, S[i][j] = min(S[i-1][j],S[i][j-1],S[i-1][j-1]) + penalty of $(i+1,j+1)^{th}$ tile, So we get

S[i][j] = min(S[i-1][j],S[i][j-1],S[i-1][j-1]) +grid[i][j]

After the loop terminates, we return the penalty to reach the $(n,m)^{th}$ tile which is S[n-1,m-1]

Assignment 5 – Harshit Mawandia

Q2. String Problem:

Our main function is *stringProblem,* and we use do not use any helper functions.

2. Time complexity analysis:

Let the length of first string be l_1 and second string me l_2

The time complexity of the algorithm is $O(3^{\min(l_1,l_2)})$ since we can do 3 operations for max of the $\min(l_1,l_2)$, we also do 2^k operations(for some k) but since that has lesser Order than 3^n, we don't need to count that.

3. Correctness Proof:

Let string a be written as $a_0a_1a_2 \ldots a_{n-1}$ where a_i are the letters that for the word a. Similarly, b be written as $b_0b_1b_2 \ldots . b_{m-1}$.

Base Cases:

If a=b, we need 0 steps

If a= "", we just need to insert all the letters in b, so len(b) steps

If b= "", we just need to delete all the letters in a, so len(a) steps

Recursion step:

If a_0 is not a vowel or if a_0 and b_0 both are vowels, we can do all the three operations, insert, replace, and delete.

If we want to insert a letter before a_0 we insert the first letter of b, i.e., $b_{0,}$ so we need to check the number of steps to convert the remaining string a which is still a with the remaining letters of b which become b[1:].

If we want to replace the first letter of a with be, we replace a_0 with b_0, and then check the number of steps to convert the remaining string, which is a[1:] to b[1:]

If we want to delete the first letter of a which is a_0, and then check the number of steps to convert the remaining string, which is a[1:] to b, which is still b.

Then we find the minimum no. of steps of the recursion after these three operations

If a_0 is a vowel while b_0 is not, we cannot replace, we just insert or delete, and check the minimum no. of steps to convert the resulting string after these two operations.