Assignment 5

Akarsh Jain

2020CS10318

Problem 1.

1. done in .py file.

Dynamic Programming is used.

The matrix a will represent the penalties in reaching the corresponding points in the input grid after execution of program.

2. Worst Case Time Complexity.

The program consists of Three loops. I show that the time complexity of 1st and 2nd loop is less than that of third.

- 1st loop.

inside of the loop assignment statement is there which is $O(1)$ and since loop runs from 1 to $m-1$. its complexity is $O(m)$

- 2nd loop.

Inside assignment statement is O(1) and loop runs from 1 to n-1 so time comp. is O(n)

- 3rd loop.

inside of the innermost loop is O(1) because min function, calling a[i-1][j], a[i][j-1], a[i-1, j-1] and assignment is O(1)

since inside loop runs from 1 to m-1 its complexity = O(m) and then since this done in outer loop for all 1 to n-1

Hence total complexity = O(mn).

3. Proof

- Proof of invariant of first loop.

Invariant : entries from a[0][0] to a[0][j-1] represent penalty in reaching that position.

Base case: $j = 1$, Output will be $a[0][0] = grid[0][0]$

since going from $a[0][0]$ to $a[0][0]$.

Maintenance: let it be true for some $j = n-1$

then since reaching $j = n$ only way possible is by $j = n-1$ Hence $a[0][j] = a[0][j-1] + grid[0][j]$.

is correct and invariant holds for $j = n$ also.

Termination: $j = m-1$ and since after execution of loop all entries in 0^{th} row of a will represent penalties in reaching there, desired output is achieved.

- Proof of invariant of second loop is exactly similar because in this 0^{th} row was altered, in that 0^{th} coloumn is altered.

- Proof of 3^{rd} loop.

Invariant 1: Entries till $a[i-1]$ ($(i-1)^{th}$ row) represent penalties in reaching there. (outer loop)

Invariant 2: entries till a[i][j-1] represent penalties in reaching there. (inner loop)

- Proof of Invariant 1 assuming 2 to be correct.

Base case - $i=1$, ~~Hence~~ But 0^{th} row is already assigned correct penalties in previous loop. Hence, Base case is correct.

Maintenance: Since inv 2 is correct and since it correctly enters penalties in Matrix a for the i^{th} row Hence, invariant will hold after each Iteration.

Termination: $i=n$. Hence after execution of the loop, all entries in Matrix a will represent penalties associated in reaching that position which is the desired output.

- Proof of invariant 2

Base case: $j=1$. But 0^{th} column (a[j][0]) is already assigned correct penalties in previous loop. Hence, Base Case is correct.

Maintenance: Let it be true for j = n-1. Then since in reach j = n there are three possible ways ~~(top, bottom,~~ (top, left, diagonally) and in execution min penalty of all three is taken into account. By Principle of optimality and our initial assumption (true for j = n-1) it will be true for j = n also.

a[i-1][j] - from top

a[i-1][j-1] - diagonally

a[i][j-1] - from left.

Termination: j = m. Hence after execution of loop, penalties for entire ith row has been correctly set. This is desired output.

- Now for person to reach end of the grid, clearly by invariants, penalty will be a[n-1][m-1].

Problem 2

1. done in .py file.

Explanation. (informal)

let length (b) = m

length (a) = n

example let b = 'abc'

a = 'abd'

so the matrix A is constructed as follows

→ b ~~list~~ string

a ~~list~~ string ↓

X	Empty	a	b	c
Empty				
a				
b			*	
d				

Now here the marked box * will be filled by minimum number of changes for 'ab' (part of a) to be converted to 'ab' (part of b) for which Dynamic Programming is used.

The empty coloumn and row is made since deletion may lead to one ~~list~~ string being empty and subsequently number of changes to be done will be the length of the remaing list.

i.e. the empty colomns and rows are introduced as base cases.

Now observe deletion of an element from list a ~~shifts~~ ~~the~~ will mean changes to be calulated for, immidieate ~~right~~ top ~~left~~ block in the matrix.

Similarly replacement will be calculating changes for diagonally above block.

And insertion will be calculating changes for immidieate left block of the matrix, Since the inserted element and corresponding element in b string will be equal hence the substrings are to be seen as smaller sub problems.

2. Time complexity. (len(a) = m , len(b) = n)
clearly in the first 2 loops only assignment statements are there which are O(1) and loops run ~~m+1, n+1 ti~~ m, n times respectively Hence they are O(m), O(n).

next page

In the final loop.

~~The i~~ One step of the innermost loop contains constant number of conditionals, assignments and min evaluations. Also vowel(n) is $O(1)$ function. Hence one step is $O(1)$

But since loop runs m times and the outer loop runs n times, the total complexity is $O(mn)$.

Hence the time complexity of complete algo is $O(mn)$.

3. Proof.

Invariant 1: (outer loop) $A[x][y]$ represents min Number of changes to convert $a[0:x]$ to $b[0:y]$ for $x < i$

Invariant 2: (inner loop) $A[x][y]$ represents min no. of changes to convert $a[0:x]$ to $b[0:y]$ $x < i+1$; $y < j$

Proof of Invariant 1 assuming 2 is correct.

Base case: $i = 1$. Means $x < 1 \rightarrow x = 0$ but since we have set $A[0][y] = y$ in previous loop and $x = 0$ means empty string a. Hence, Base Case is correct.

Maintenance: By the invariant of loop 2 after it terminates, it will have set correct values for $A[i][j] \; \forall j$ Hence, invariant is valid for next iteration also.

Termination: $i = len(a)$ and after final iteration all entries in $A[x][y]$ will represent no. of changes to convert ~~a[n~~ $a[0:x]$ to $b[0:y]$ which is the desired output.

Proof of Invariant 2.

Base Case: $j = 1 \rightarrow y < 1 \rightarrow y = 0$ but we have set values of $A[x][0] = x$ since $y = 0$ represents empty string b. Hence Base Case is correct.

Maintenance: Let it be true for j = n. Then in the loop we check for all possibilities i.e.

- a[i-1] = b[j-1]

  A[i][j] = A[i-1][j-1]

  since the elements are equal we need to find the min changes to convert remaining elements of a to that of b. Because of Principle of optimality.

- a[i-1] ≠ b[j-1] :

  if both are vowels then all replacement, deletion and insertion are possible i.e.

  A[i][j] = ~~min~~ 1+ min (A[i-1][j-1], A[i-1][j], A[i][j-1])

  if a is vowel b is not then replacement is not possible.

  in other cases all three are possible.

Hence the invariant will be true for j = n+1 also.

Termination: j = len(b) Hence after execution all entries would be ~~made~~ made for A[i][j] ∀ j

Hence the required output is achieved.