FA Assignment 8

Q1.

$$Cost(T) = T.root.key + 2 \cdot Cost(T.root.left) + 2 \cdot Cost(T.root.right)$$

$K[1, \_\_, n]$.

1.(a)

let $Best[i, j]$ be the cost of the BST from i to j inclusive.

K - inorder traversal of T.

$Best[r, r] = r.key$

$Best[r, q]$ where $q < r$ is 0.

1.(b)

$$Best[i, j] =$$

$$\min \left(\forall_{i \le r \le j} \left(2 \cdot Best[i, r-1] + r.key + 2 \cdot Best[r+1, j] \right) \right)$$

- Boundaries :

$2Best[i, i-1] (\#0)$

$+ \; i \cdot key + 2 \cdot Best[i+1, j]$

all nodes in the right sub tree of i.

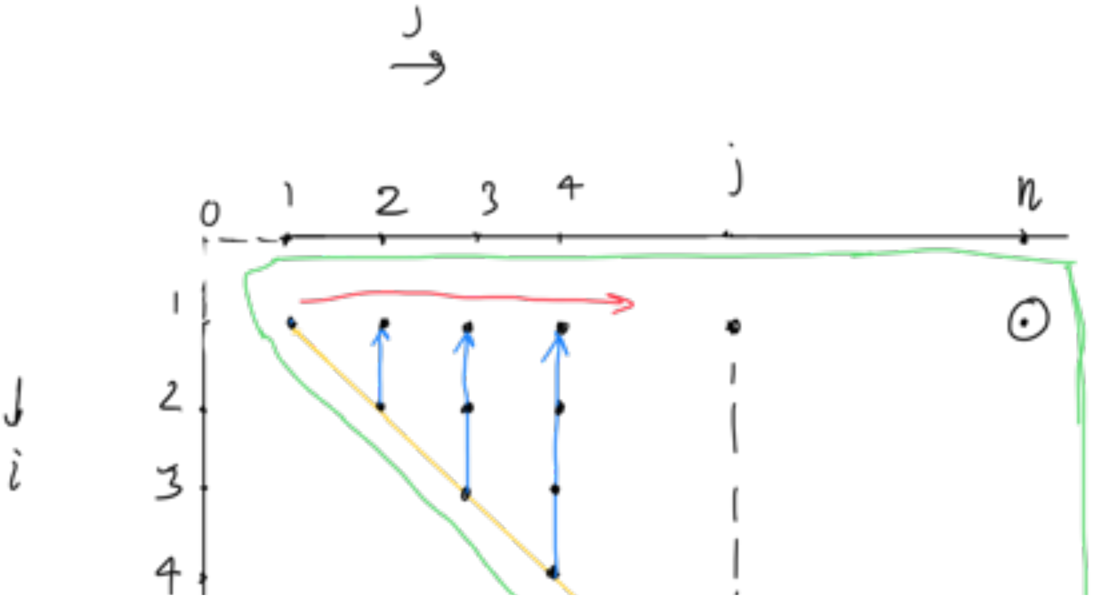
$2Best[j, j+1] (\#0)$

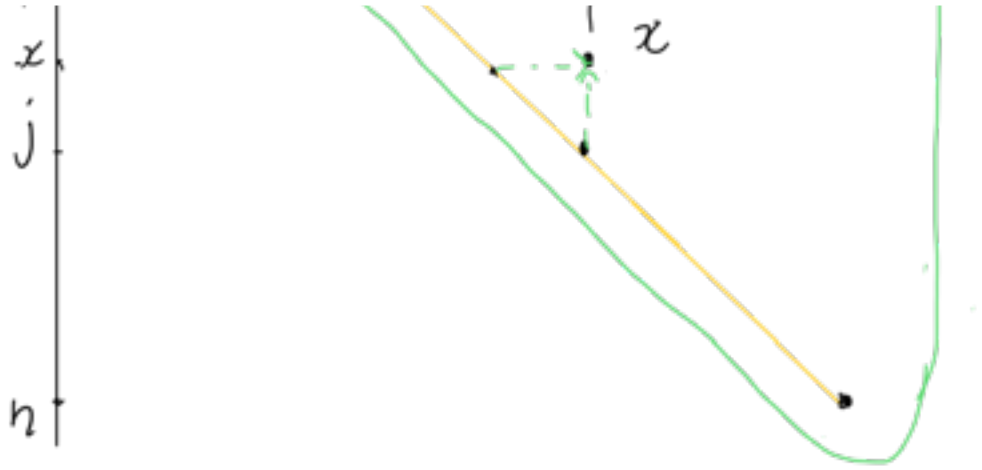
$+ j \cdot key + 2 Best[i, j-1]$

all nodes in the left sub tree of j.

In order traversal $K[i, \_, j]$, represents the nodes in sorted order and preserves the structure of the tree if for any root r, $(i \ldots r-1)$ on the left sub tree and $(r+1 \ldots j)$ on the right sub tree.

Final answer $Best[1, n]$





- The red arrow represents the outer index from 1 to n.
- for each value of the outer index, the inner index will go from that value to 1 as represented by the blue arrow.
- We can see that we have to fill the values inside the green triangle
- for iteration j, there will be j points, blue arrow going from j to 1. The amount of look ups needed to fill a point will be proportional to the length of the blue arrow. Follows from our equation for Best. This will be the number of conditions in max. (*2). Adjusting the indices of our summation,

$$\therefore T(n) \propto \sum_{j=1}^{n} \left(j \left(\sum_{k=1}^{j} \Theta(1) \right) \right)$$

$$\therefore T(n) = \Theta(n^3)$$

We are filling the array iteratively bottom up considering the ordering of the sub problems.

1.(b)

$K[1, \_\_, n]$ - post order traversal

Best$[i, j]$ - Optimal binary tree value such that $K[i, j]$ maintains the post order traveral.

Base Case:

$$\text{Best}[i, i] = i.\text{key}$$

$$\text{Best}(i, j) = 0 \text{ if } j < i$$

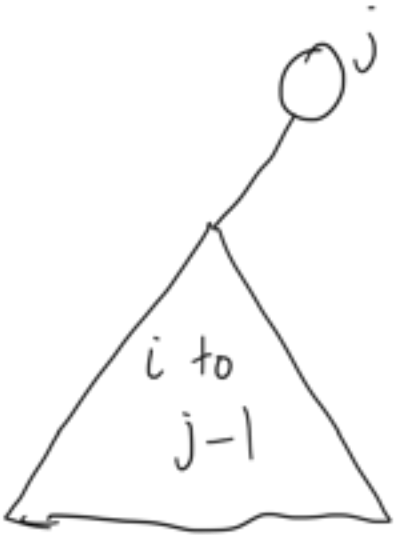
Recursive formulation:

$$\text{Best}[i, j] =$$

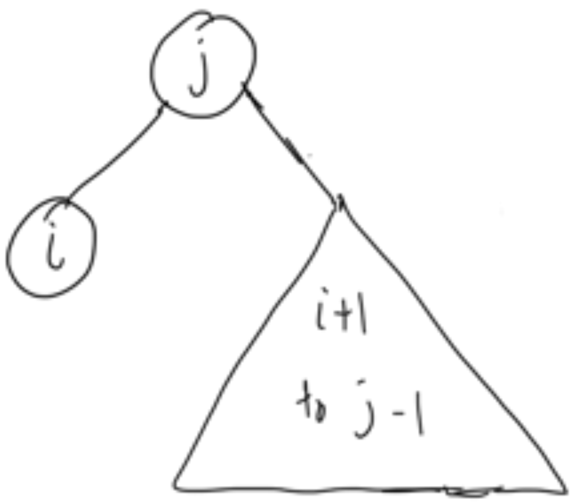
$$\min\left(\forall_{i-1 \le k \le j-1} \left(2.\text{Best}[i, k] + 2.\text{Best}[k+1, j-1] + j.\text{key}\right)\right)$$

Boundary conditions,

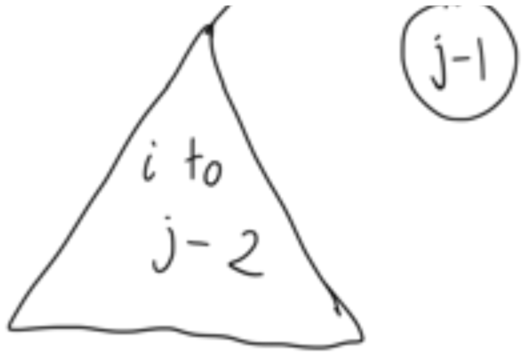
$k = i-1$,



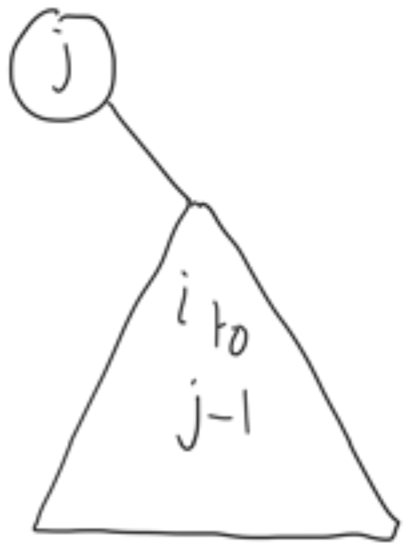
$k = i$,



$k = j-2$



k = j-1



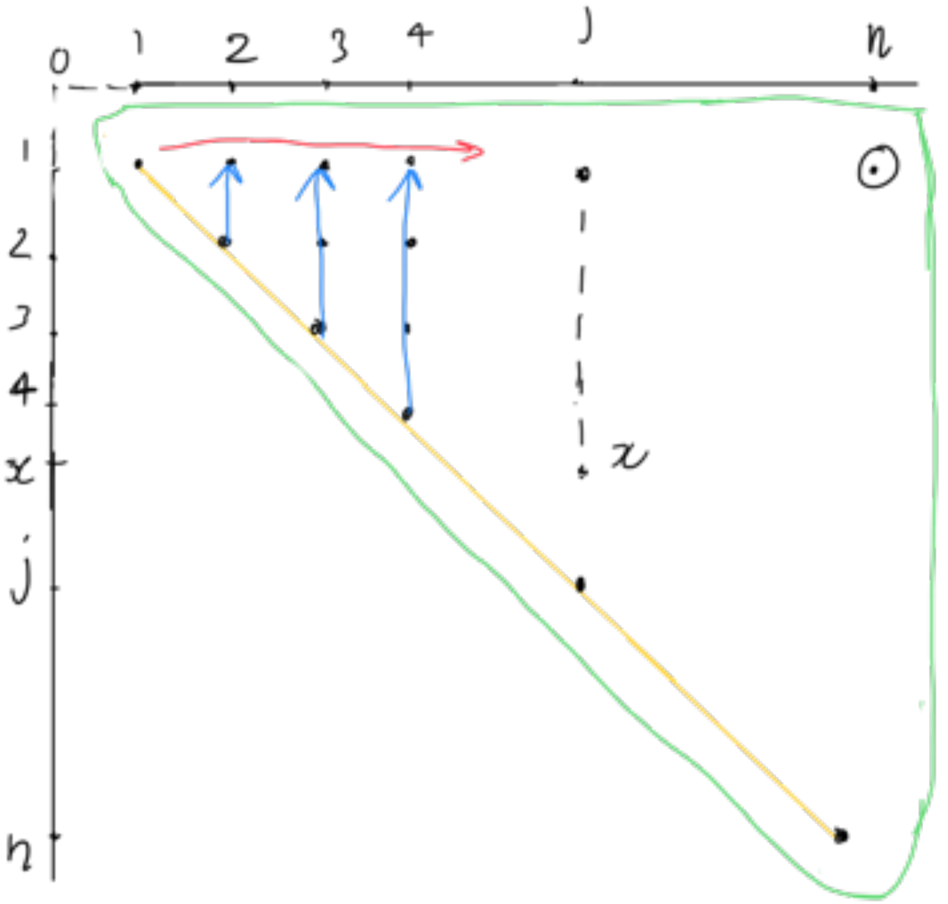
Note that our our cost function, placing an optimal sub tree on the left or right will not make a difference. The last diagram is just for notation.

For Post order traversal to be maintained, j must be at the root of the binary tree.

for any $i \le k < j$, the sub trees at k will be traversed before k, the elements ahead of k on which we are calling Best will also satisfy the post order traversal array properties recursively.

We are creating linear ordered cuts in the array in order to test for optimality and preserve the post order traversal property of array K.

$\rightarrow T(n) = \theta(n^3)$



We again have to fill the green triangle.
The zeros are just to be padded below the yellow diagonal.
for each iteration of the outer loop j, represented by the red arrow, the number of look ups for Best [x, j] will be proportional to the number of terms between x and j.
These numbers will already be present in our table. ie $k = i + \alpha$ $(i < \alpha < j)$
Best [$i+\alpha+1$, $j-1$] will be present below Best [i, $j-1$] and our blow arrow iterator will fill it before j. column $j-1$. Column $j-1$ comes before j.

Best [i, i+α] will be present to the left of Best [i,j] in some column before j and hence will have been already filled.

x will go from j to 1 to preserve the ordering of the sub problems.

$$\therefore \; T(n) \propto \sum_{j=1}^{n} \left(j \cdot \sum_{k=1}^{j} (\theta(1)) \right)$$

$$T(n) = \theta(n^3)$$

Q1.(c)

$K[1, \_\_, n]$

$\rightarrow$ $Cost(T) = T.root.key + \sum_{i \le i \le l} 2 \cdot cost(w_i)$

$Best[i, i] = i.key$

$Best[i, p] = 0$ if $p < i$

Recursive formulation,

$Best[i, j] =$

$$\min \left(\forall_{i \le k \le j} \left(2 \cdot Best[i, k-1] + Best[k, j] \right) \right)$$

→ Such a formulation enables us to maintain the post order traversal of the tree.

→ As we iterate the variable k, for some k, we consider that Best [i, k-1] as one child of j. Best [i, k-1] will represent the optimal cost of a sub tree rooted at k-1.

→ This will act as a single left child for our original tree i.e. Best [i, j], The rest of the children will be the result of an optimal tree from k to j. This is essentially our definition of Best.

- We show that $T(n) \equiv \Theta(n^3)$

→ For the Best table,

→ As we maintain the ordering of the sub problems in our implementation, for Best [i, j], look ups to Best for sub arrays will take constant time.

Therefore, the time taken will be proportional to the number of terms between i and j as we follow the correct ordering of the sub problems. As this follows the same pattern earlier,

→ Space Complexity(n) $\equiv$ $O(n^2)$

Table for Best,

→ $T(n) = \Theta(n^3)$

Q1.d.

- New_Best $[i, j, l]$ - best possible constrained tree such that post order traversal is maintained and root has exactly l children.
- Best $[i, j]$ - same as previous question.

- Base Cases:

New_Best $[1, 1, 1]$ = $K[1]$.key

for k from 2 to n:

$$New\_Best[1, k, 1] = K[k].key + 1 \cdot Best[1, k-1]$$

$$New\_Best[i, j, k] = 0$$

$\rightarrow$ if $(j - i + 1) < k$

or

$\rightarrow$ if $k = 0$

As the last time, the failure case at the boundary was used in conjunction with a boundary case as a subset of that valid boundary case and hence we used used zero for addition purposes based on our limits. The limits will ensure that a non valid case is not used in the comparision operation.

Recursive formulation:

$$New\_Best[i, j, l] =$$

$$\min\left(\forall_{i \le k < j-l} \left(l \cdot Best[i, k] + New\_Best[k+1, j, l-1] + i.key \right)\right)$$

For New_Best [i, j, l], we need l nodes at the root. We iterate over i to j-l via k. We obtain one from Best [i, k] based on our previous definition and (l-1) nodes through our recursive formulation of New_Best. For example, if k+k' is the next chosen k for the next New_Best look up, Best [k+1, k+k'] will be multiplied by (l-1), and then new_Best will be called upon the rest of the sub array with (l-2) as the parameter. This enables us to search through all valid k values and maintain the post order traversal of the tree.

For Time complexity,

For each j, we will need to do $\theta(j)$ new_Best with respect to the previous elements. Each calculation itself will take $\theta(j-l-1)$ time. We also have to do this for all l values.

$$T(n) = \left(\sum_{l=1}^{l} \left(\sum_{j=1}^{n} j \cdot \sum_{k=1}^{j-l-1} O(1) \right) \right)$$

$$\sum_{l=1}^{l}\left(\sum_{j=1}^{n} j\cdot(j-l-1)\right)$$

$$T(n) \propto \sum_{l=1}^{l}\sum_{j=1}^{n} j^2 - \sum_{l=1}^{l}\sum_{j=1}^{n} jl$$

$$\propto \quad \Theta(n^3 l) \qquad \# \; l \in \{0, 1, \_, n\}$$

$$T(n) = \Theta(n^3 l)$$

The old Best itself will take $\Theta(n^3)$ time
so this term will dominate.

For Space Complexity,

$$S(n) = \Theta(n^2 l)$$

The number of entires in our 3D table.
Old Best was $\Theta(n^2)$ space.

Q2.

(a)

Given $X[1, \_\_, m]$
$Y[1, \_\_, n]$

$$C : \Sigma \times \Sigma \rightarrow R^{+}$$

For LCS,

$$C(X[i], Y[j]) = 1 \quad \text{if } X[i] = Y[j]$$
$$= 0 \quad \text{otherwise}$$

Consider the recursive formulation,

$$LCS[i, j] =$$
$$\max \{ C(X[i], Y[j]) + LCS(i-1, j-1),$$
$$LCS(i-1, j),$$
$$LCS(i, j-1)$$

$LCS[i, j-1)$

}

(b)

$X[1, \_\_\_, m]$
$Y[1, \_\_\_, n]$

Base case :

$$Best[1,1] = C(X_1, Y_1)$$

$$Best[1,k] = \max\left(Best[1,k-1] + C(X_1, Y_k),\ Best[1,k-1] \right)$$

$\forall k \in \{2, n\}$

$$Best[p,1] = \max\left(Best[p-1,1] + C(X_p, Y_1),\ Best[p-1,1] \right)$$

$\forall p \in \{2, m\}$

→ Recursive formulation,

$$\begin{aligned} Best[i, j] = \\ \max\Big(& Best[i-1, j-1] + C(X_i, Y_j), \\ & Best[i-1, j], \\ & Best[i, j-1] \Big) \end{aligned}$$

→ At each stage, for the optimal cost of the sub sequence $X[1, \_, i]$, $Y[1, \_, j]$, we make a choice as to wether or not to include the pair (X_i, Y_j) in our sub sequence calculations.

→ If $(x_1, \_, x_k) \leftrightarrow (y_1, \_, y_k)$ are the optimal indices for our subsequence cost maximization process. They will be selected at some previous stage of the maximization process. Since that corresponding sub array must also be optimal. If there was some other optimal cost for the corresponding sub array, that would be selected with regards to the original array by our dynamic programming formulation.

→ $T(n,m) = O(nm)$

→ Analgous to filling the table in the LCS problem.

→ We are using the ordering of the sub problems, there are $O(nm)$ entries in the table.

Q2. c.

Best $[i, j, k]$ - maximum cost of subsequence for $X[1, \_, i]$ and $Y[1, \_, j]$ such that k elements are chosen.

$$Best[1,1,1] = c(X_1, Y_1) - 1^2$$

$$Best[1, k, 1] = \max \{ \; Best[1, k-1, 1], \; c(X_1, X_k) - 1^2 \; \}$$

$$Best[l, 1, 1] = \max \{ \; Best[l-1, 1, 1], \; c(X_l, X_1) - 1^2 \; \}$$

$$Best[i, j, k] = \max \{ Best[i-1, j-1, k-1] + c(X_i, Y_j)^2 - 2k + 1, \; Best[i-1, j, k], \; Best[i, j-1, k] \}$$

If the first option holds,

$B_{prev} = Cost_{prev} - (k-1)^2$

$$B_{new} = Cost_{prev} + C(X_i, Y_j) - k^2$$

$$= Cost_{prev} + c(X_i, Y_j) - k^2 + (k-1)^2$$

$- (k-1)^2$

$Bnew = Cost_{prev} - (k-1)^2$

$$+ C(X_i, Y_j) - k^2 + (k-1)^2$$

$(k-1)^2 - k^2 = (k-1-k)(k-1+k)$
$$= -2k + 1$$

→ From the recursive formulation, we make use of the ordering of the sub problems.

→ For Best $[i, j, k]$, the first option is the choice of selecting $C(X_i, Y_j)$ in our cost function, else we select the max of Best $[i, j-1, k]$ and Best $[i-1, j, k]$.

→ $T(n, m, k) = O(nmk)$

This is analogous to filling the entries of the $n \times m \times k$ cube.

For each entry, the entries on which it depends on are known and are constant time look ups, the number of parameters in consideration of one maximization is thus 3. Each entry in the table is filled in constant time as follows from the ordering of the sub problems.

There are $O(nmk)$ entries in the cube.

Q3.

(a)

To Prove :

$$LCS(X' \| b, Y' \| b) = LCS(X', Y') \| b$$

Set $A \equiv LCS(X' \| b, Y' \| b)$

Set $B \equiv LCS(X', Y') \| b$

For some element A' such that A' belongs to $LCS(X', Y')$,

$A' \equiv \{a_1, a_2, \_, a_k\}$

$\{a_1, a_2, \_, a_k\} \in X'$

$\{a_1, a_2 \_ a_k\} \in Y'$

After concatenation of X' and Y' with b, the LCS of $(X' \| b, Y' \| b)$ will go through the following recursive formulation,

$$LCS(X'\|b,\ Y'\|b) = LCS(X',Y') + \mathbb{1}\{b=b\}$$

$\therefore$ If $LCS(X',Y')$ contains string A', $LCS(X'\|b,\ Y'\|b)$ must contain $A'\|b$.

$A'\|b$ is by defition an element of set B which is $LCS(X',Y')\|b$.

$\therefore$ If A' is an element of set B, it must be present in Set A.

$\therefore$ Every element in Set B is contained in Set A.

If A'' is an element of set A, based on our recursive formulation, it must be obtained by concatenating b to some element of $LCS(X',Y')$.

$\rightarrow$ Suppose A'' didn't come from concatenating with some element in $LCS(X',Y')$. The first element of A'' is b. So the rest of the string must be an LCS of X', Y', if it wasn't A'' itself wouldn't be an LCS and therefore wouldn't be in Set A.

$\therefore$ If A'' is an element of set A, it must be present in set B.

∴ Every element of set A is present in Set B.

$\therefore A \subseteq B$ and $B \supseteq A$.

$\therefore A = B$

$\therefore LCS(X' \| b, Y' \| b) = LCS(X', Y') \| b$

Q3.
(b)

$L[i, j] =$

if $X[i] = Y[j]$:

$L(i-1, j-1) \| X[i]$

else :

$L(i-1, j) \cup L(i, j-1)$

↑ if $c[i-1, j] = c[i, j-1]$

$= L(i-1, j)$

↑ if $c[i-1, j] >$

$c[i, j-1]$

$= L(i, j-1)$

$\uparrow$ if $c[i-1, j] < c[i, j-1]$

(c)

$X_{3n} = [0, 1, 2, \ldots, 0, 1, 2]$

$= [(012)^n]$

$Y_{3n} = [(102)^n]$

Base case,

$n = 1,$

$\lambda[3, 3] = |\{\{0, 2\}, \{1, 2\}\}|$

$= 2$

Assume that the relation holds for $(n-1)$.

$\lambda(3(n-1), 3(n-1)) = \Omega(2^n)$

Inductive Step,

$$L(3n, 3n) = L(3(n-1), 3(n-1)) \, || \, \{0,2\}$$
$$+$$
$$L(3(n-1), 3(n-1)) \, || \, \{1,2\}$$

$$\therefore \lambda(3n, 3n) = 2\lambda(3(n-1), 3(n-1))$$
$$\therefore \lambda(3n, 3n) \leq 2.(c\,2^{n-1})$$
$$\therefore \lambda(3n, 3n) \leq c.2^{n}$$
$$\lambda(3n, 3n) = \Omega(2^{n})$$

(e)

Count$[i, j]$ - number of distinct pairs of LCS for $X[1, \_\, i]$, $Y[1, \_\_, j]$

$c[i,j]$ - standard LCS matrix.

Firstly, we consider what happens when X_i is not equal to X_j. We then talk about when X_i equals X_j.

Count [i, j] :=

{

Case 1.

1. if $(X_i \neq Y_j)$:

case 1.1

$c[i-1, j] > c[i, j-1]$

$c[i, j] = c[i-1, j]$

$\rightarrow$ Count [i, j] = Count [i-1, j]

Explanation :

if X_i is not equal to Y_j and the LCS is determined by (X_{i-1}, Y_j), we will consider the number of LCS strings from Count (i-1, j) as the number of LCS strings for Count (i, j)

case 1.2

$c[i, j-1] > c[i-1, j]$

$c[i, j] = c[i, j-1]$

$\rightarrow$ Count [i, j] = Count [i, j-1]

X_i does not equal Y_j, The LCS value $(c[i, j])$ is determined by $c[i, j-1]$. The LCS

strings will therefore consist of the LCS strings between $X[1,\_,i]$, $Y[1,\_,j-1]$.
So we put the value of $Count[i, j-1]$ in $Count[i,j]$.

case 1.3

$$c[i-1,j] = c[i,j-1] = c[i,j]$$
$$c[i,j] \neq c[i-1,j-1]$$

$$\rightarrow Count[i,j] = Count[i,j-1] + Count[i-1,j]$$

\#• If X_i doesn't equal Y_j and $c[i-1,j]$ equals $c[i,j-1]$. This will determine the value of $c[i,j]$. Also, $c[i-1,j]$ and $c[i,j-1]$ which determine $c[i,j]$, are not equal to $c[i-1,j-1]$.

- for $c[i-1,j]$, the value Y_j is somewhere inside $X[1,\_,i-1]$ and is therefore making a difference for the LCS $c[i-1,j]$ value as compared to $c[i-1,j-1]$. Similarly for X_i wrt comparing $c[i,j-1]$ and $c[i-1,j-1]$. Since this value is greater than $c[i-1,j-1]$, we consider the pairs $(i-1,j)$ and $(i,j-1)$ for our longest common subsequence.

- We will have two different sets of strings with the same LCS value. Therefore we add the count values for the two pairs $(i-1,j)$

and $(i,j-1)$, as the LCS value is the same, they will both constitute LCS strings for (i,j).

case 1.4

$$c[i-1,j] = c[i,j-1] = c[i,j]$$
$$c[i,j] = c[i-1,j-1]$$

$\rightarrow$ $Count[i,j] =$

$$Count[i-1,j-1] + (Count[i,j-1] - Count[i-1,j-1]) + ((Count[i-1,j] - Count[i-1,j-1]))$$

$\hat{\#}$ • When X_i doesn't equal Y_j and $c[i-1,j]$ equals $c[i,j-1]$, this determines the $c[i,j]$ value. Also, if $c[i-1,j]$ equaling $c[i,j]$ equaling $c[i,j-1]$ is also equal to $c[i-1,j-1]$.

- The LCS value is determined by $c[i-1,j-1]$.
- Since this is the same value as that for $c[i-1,j]$, there may be some strings in $Count[i-1,j]$ that are a result of the inclusion of j into the array. There will be strings in $Count[i-1,j]$ that are continuing from $Count[i-1,j-1]$

• Both sets of additional strings in Count[i-1,j] and Count[i,j-1] must be included in Count[i,j].

eg:

1	2	3	4	6	10	20	5	i
1	2	3	4	5	20	10	6	j

$(i-1, j-1)$

1 2 3 4 10
1 2 3 4 20

Count = 2

$(i-1, j)$

1 2 3 4 10
1 2 3 4 20
1 2 3 4 6

Count = 3

$(i, j-1)$

1 2 3 4 10
1 2 3 4 20
1 2 3 4 5

Count = 3

$$\text{Count}(i,j) = 2 + (3-2) + (3-2)$$
$$= 4$$

(i,j)

1 2 3 4 10
1 2 3 4 20
1 2 3 4 6
1 2 3 4 5

Case 2.

2. if $(X_i == Y_j)$: // $c[i,j] = c[i-1,j-1] + 1$

Case 2.1

$$c[i,j] = c[i-1,j] = c[i,j-1]$$

$\rightarrow$ $Count[i,j] =$

$$Count[i-1,j-1] +$$
$$Count[i-1,j] +$$
$$Count[i,j-1]$$

Explanation:

- For every string corresponding to $Count[i-1,j-1]$, the letter X_i will be concatenated.
- This will a set of strings different from those represented in $Count[i-1,j]$ and $Count[i,j-1]$, all three sets having the same LCS value.

Case 2.2.

$$c[i-1,j] < c[i,j-1]$$
$$c[i,j] = c[i,j-1]$$

$\rightarrow$ $Count[i,j] = Count[i,j-1]$

$$+ \ \mathrm{Count}[i-1, j-1]$$

following from our last explanation, we consider the valid sets of LCS strings. Since the c value for (i-1,j) is now lesser, it will not constitute into the longest common subsequence strings.

Case 2.3

$$c[i, j-1] < c[i-1, j]$$
$$c[i, j] = c[i, j-1]$$

$$\rightarrow \mathrm{Count}[i, j] = \mathrm{Count}[i-1, j] + \mathrm{Count}[i-1, j-1]$$

Symmetric to 2.2.

case 2.4

$$c[i, j] > \max(c[i-1, j], c[i, j-1])$$

$$\rightarrow \mathrm{Count}[i, j] = \mathrm{Count}[i-1, j-1]$$

For every string in LCS (i-1, j-1), the letter for X[i] will be appended. This will constitute the new longest common subsequence of string. The count thus remains equal to that of $(i-1, j-1)$.

$$T(n) = \theta(mn)$$

As we follow the ordering of the sub problems, each such count value to be filled in the table takes $\theta(1)$ time and there are $\theta(mn)$ such values in the table.

3.f.

$$\lambda(3n, 3n) = 2^n \quad \text{- part d}$$

$$\lambda(n, n) = 2^{n/3}$$

$$n = \min(m, n)$$

let k go from 1 to n.

for some k,

$$\lambda(k, k) = \Omega(2^{k/3})$$

$$\lambda(k, k) \leq c\, 2^{k/3}$$

$$\lambda(k-1, k) \geq \lambda(k-1, k-1)$$

$$\lambda(k-1, k) \geq c\,2^{k-1}$$

$$\lambda(k+1, k) \geq \lambda(k, k)$$

$$\lambda(k+2, k) \geq \lambda(k, k)$$

$$\sum_{i=1}^{m} \lambda(k, x_i) = \gamma(k, m)$$

$$= \lambda(k, 1) + \lambda(k, 2)$$

$$\ldots \lambda(k, m)$$

$$\leq \ldots \underbrace{c2^{k-2} + c2^{k-1} + c2^{k}}_{\alpha 2^{k}} + \underbrace{c(m-k)2^{k}}_{m2^{k}}$$

The LCS until k will be bounded by the size of k.

$m)$

Size of an integer at each level $= \theta(2^k)$

No. of bits needed to ... k integers

No. of bits needed to represent integers at level $k = \Theta(k)$

Total $= \Theta(mk)$

for total number of entries in the table,

$$= \sum_{k=1}^{n} (mk)$$

$$= \Theta(mn^2)$$

$$T(n, m) = \Theta(mn^2)$$