COMP 550
Algorithm and Analysis

Dynamic Programming

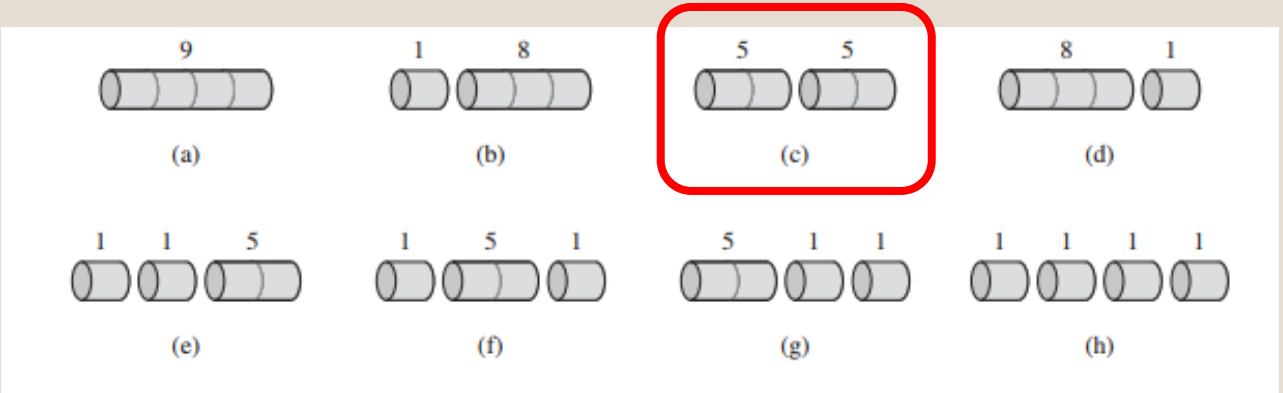
Based on CLRS Sec. 14

The Rod-Cutting Problem

- Buy long rods, cut them into shorter rods, and sell them
- Input: a rod of length n, a list of prices $P = \{p_1, p_2, \ldots, p_n\}$ (p_i denotes the price of length-i rod)
- Output: The maximum achievable revenue (and corresponding rod cuts)

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

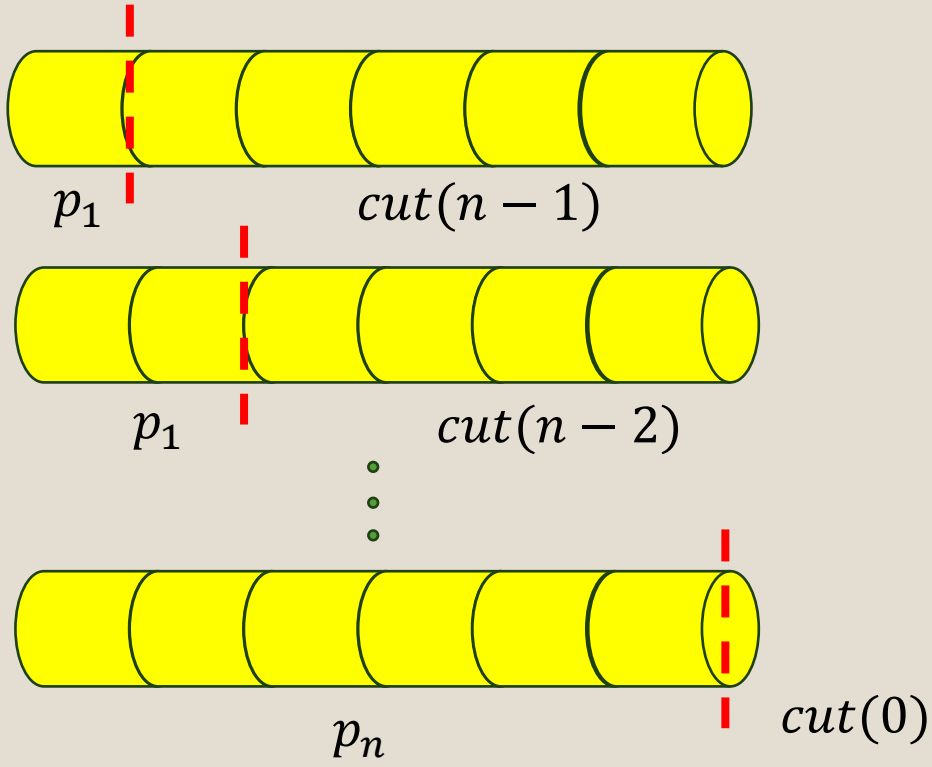
8 ways to cut a rod of length 4 (from CLRS)



Determine a Recursive Structure

- Let $cut(n)$ be the maximum revenue of cutting a rod of length n
- Express $cut(n)$ w.r.t smaller $cut(\cdot)$ and p_i values

$$cut(n) = \boldsymbol{max} \begin{cases} cut(n-1) + p_1 \\ cut(n-2) + p_2 \\ \vdots \\ cut(0) + p_n \end{cases}$$



Recursive Algorithm

```
CUT-ROD(p, n)
1  if n == 0
2      return 0
3  q = -∞
4  for i = 1 to n
5      q = max {q, p[i] + CUT-ROD(p, n - i)}
6  return q
```

$R(n)$: Recursive calls made for given value of n.

$$R(n) = 1 + \sum_{j=0}^{n-1} R(j)$$

$$R(n) = 2^n$$

Bottom-Up DP

$$cut(n) = \boldsymbol{max} \begin{cases} cut(n-1) + p_1 \\ cut(n-2) + p_2 \\ \vdots \\ cut(0) + p_n \end{cases}$$

- What should be the table dimension and size?
  - 1D array: $cut[0:n]$
- What are the base cases?
  - $Cut[0] = 0$
  - No price to sell 0-length rod.

Length i	0	1	2	3	4	5	6	7	8	9	10
Cut(i)	0										

Bottom-Up DP

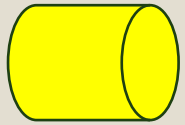
- Fill up the table iteratively using the recurrence relation

$$cut(n) = \boldsymbol{max} \begin{cases} cut(n-1) + p_1 \\ cut(n-2) + p_2 \\ \vdots \\ cut(0) + p_n \end{cases}$$

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

$$Cut(1) = \max(\, Cut(0) + p_1)$$

Length i	0	1	2	3	4	5	6	7	8	9	10
Cut(i)	0	1									



Bottom-UP DP

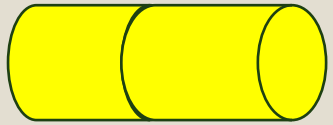
- Fill up the table iteratively using the recurrence relation

$$cut(n) = \boldsymbol{max} \begin{cases} cut(n-1) + p_1 \\ cut(n-2) + p_2 \\ \vdots \\ cut(0) + p_n \end{cases}$$

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

$$Cut(2) = \max(\, Cut(1) + p_1, Cut(0) + p_2)$$

Length i	0	1	2	3	4	5	6	7	8	9	10
Cut(i)	0	1	5								



Bottom-UP DP

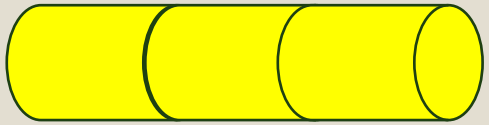
- Fill up the table iteratively using the recurrence relation

$$cut(n) = \boldsymbol{max} \begin{cases} cut(n-1) + p_1 \\ cut(n-2) + p_2 \\ \vdots \\ cut(0) + p_n \end{cases}$$

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

$$Cut(3) = \max(\, Cut(2) + p_1, Cut(1) + p_2, Cut(0) + p_3)$$

Length i	0	1	2	3	4	5	6	7	8	9	10
Cut(i)	0	1	5	8							



Bottom-UP DP

```
BOTTOM-UP-CUT-ROD(p, n)
1  let r[0:n] be a new array          // will remember solution values in r
2  r[0] = 0
3  for j = 1 to n                     // for increasing rod length j
4      q = -∞
5      for i = 1 to j                 // i is the position of the first cut
6          q = max{q, p[i] + r[j - i]}
7      r[j] = q                       // remember the solution value for length j
8  return r[n]
```

The inner loop executes for 1+2+…+n times

Running Time: $\Theta(n^2)$

Determining Rod Cuts

- Determine the rod lengths that produce the maximum revenue
- Two ways:
  - Store the best cut for each length (one additional table needed)
  - Trace back the computation to determine the best cut

Determining Rod Cuts

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
 1  let r[0..n] and s[0..n] be new arrays
 2  r[0] = 0
 3  for j = 1 to n
 4      q = -∞
 5      for i = 1 to j
 6          if q < p[i] + r[j - i]
 7              q = p[i] + r[j - i]
 8              s[j] = i
 9      r[j] = q
10  return r and s
```

```
PRINT-CUT-ROD-SOLUTION(p, n)
1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
2  while n > 0
3      print s[n]
4      n = n - s[n]
```

i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10

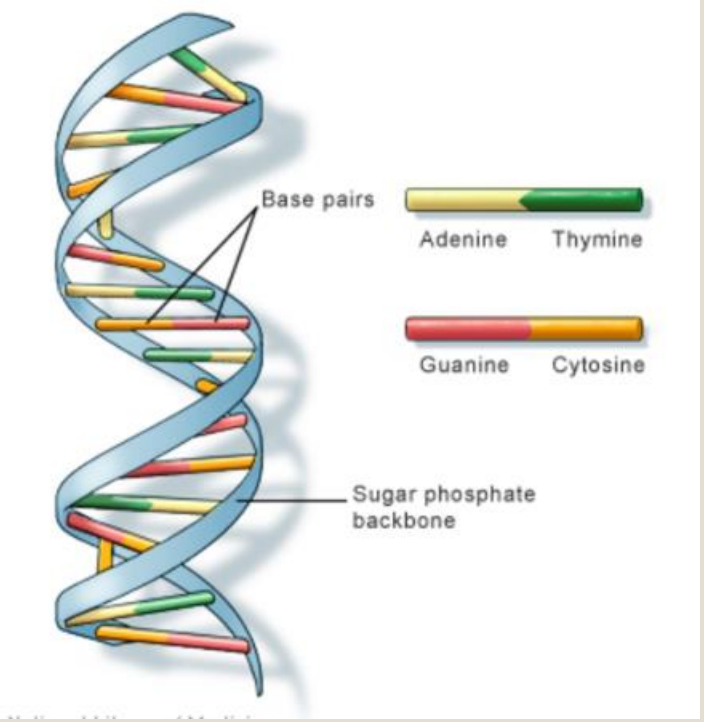
Longest Common Subsequence (LCS)

- Input: Two sequences, $X = \langle x_1, \ldots, x_m \rangle$ and $Y = \langle y_1, \ldots, y_n \rangle$
- Output: A common subsequence of X and Y whose length is the maximum.

- Subsequence need not be consecutive but must be in order

Longest Common Subsequence (LCS)

- Application:
  - Measuring similarity between DNA sequences of different organisms
  - DNA sequences for with letters: A, T, C, G



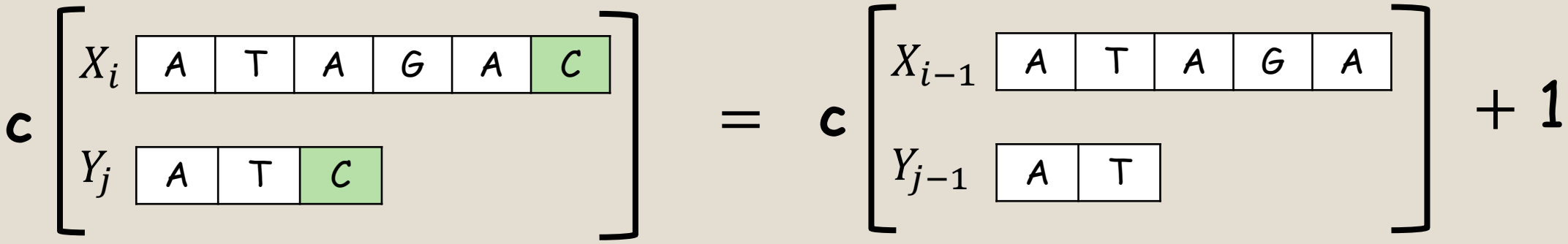
Source: https://medlineplus.gov/genetics/understanding/basics/dna/

Naïve Algorithm

- For every subsequence of X, check whether it's a subsequence of Y .
- Running Time: $\Theta(n2^m)$.
  - 2^m subsequences of X to check.
  - Each subsequence takes $\Theta(n)$ time to check: scan Y for first letter, for second, and so on

Determine A Recursive Structure

- Let $c[i,j]$ be the length of the LCS of $X_i = \langle x_1, \ldots, x_i \rangle$ and $Y_j = \langle y_1, \ldots, y_j \rangle$
  - X_i is a **prefix** of X
- Can we express $c[i,j]$ w.r.t $c[\cdot,\cdot]$ of smaller sequences?

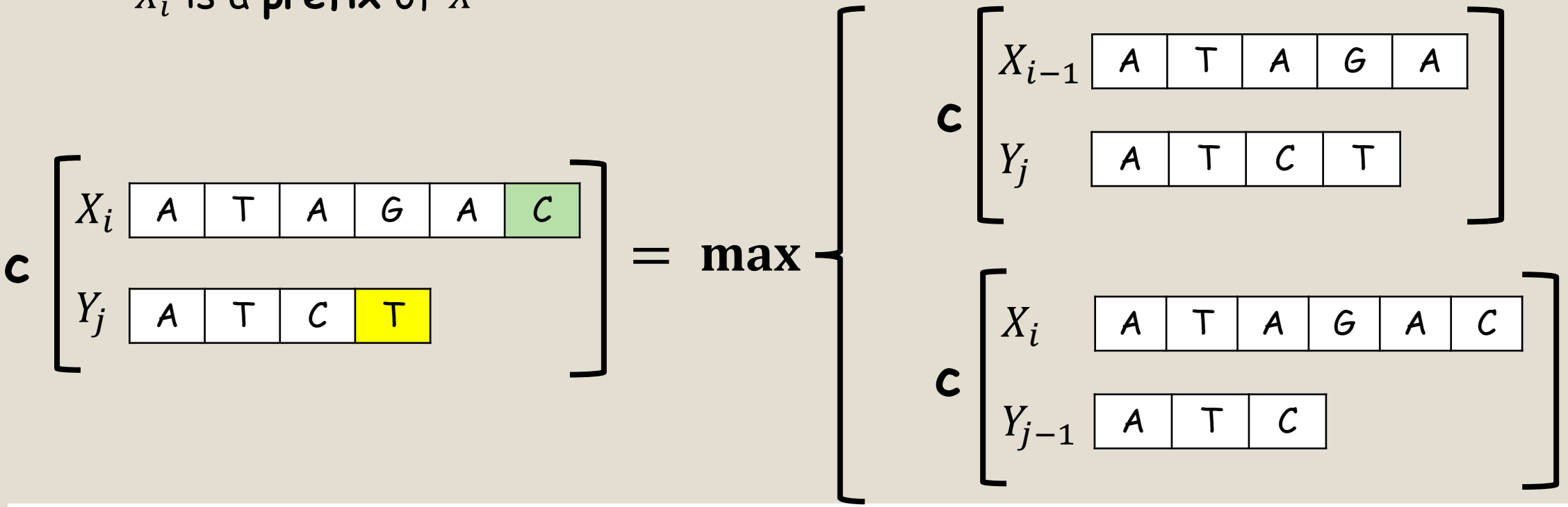


- If $x_i = y_j$, then

  LCS of X_i and Y_j must end in x_i and $c[i,j] = c[i-1, j-1] + 1$

Determine A Recursive Structure

- Let $c[i,j]$ be the length of the LCS of $X_i = \langle x_1, \ldots, x_i \rangle$ and $Y_j = \langle y_1, \ldots, y_j \rangle$
  - X_i is a **prefix** of X

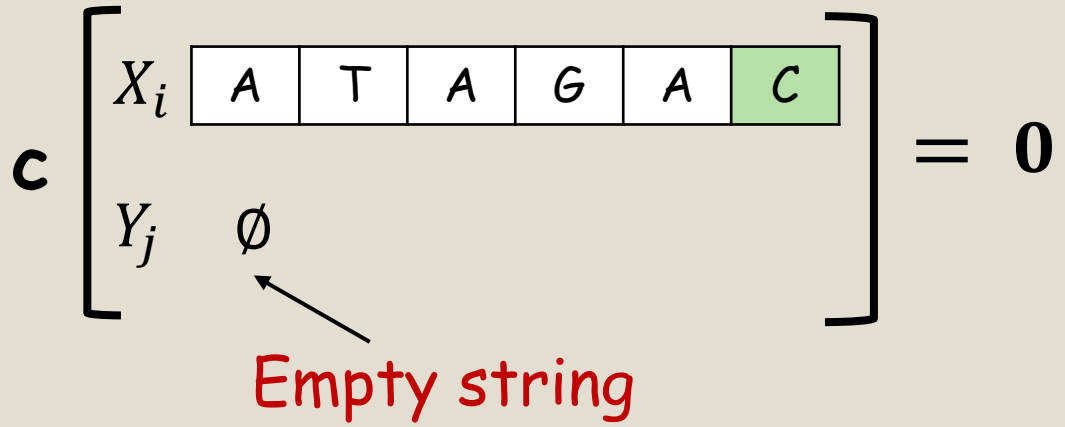


- If $x_i \neq y_j$, then

The end of LCS is either $\neq x_i$ or $\neq y_j$, and $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Determine A Recursive Structure

- Let $c[i, j]$ be the length of the LCS of $X_i = \langle x_1, \ldots, x_i \rangle$ and $Y_j = \langle y_1, \ldots, y_j \rangle$
  - X_i is a **prefix** of X



- If $i = 0$ or $j = 0$, then $c[i, j] = 0$

Determine A Recursive Structure

- Let $c[i,j]$ be the length of the LCS of $X_i = \langle x_1, \dots, x_i \rangle$ and $Y_j = \langle y_1, \dots, y_j \rangle$
  - X_i is a **prefix** of X

$$c[i,j] = \begin{cases} 0 & \text{, if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{, if } x_i = y_j \\ \max(\, c[i-1,j], c[i,j-1] \,) & \text{, if } x_i \neq y_j \end{cases}$$

Determine A Recursive Structure

- Optimal substructure property
  - We can give the following theorem based on the same idea
  - CLRS gives this Theorem before providing the recursive solution

Theorem

Let $Z = \langle z_1, \ldots, z_k \rangle$ be any LCS of X and Y.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then either $z_k \neq x_m$ and Z is an LCS of X_{m-1} and Y .
3. or $z_k \neq y_n$ and Z is an LCS of X and Y_{n-1}.

Please go through the proof before next class and
let me know if we need to cover this in class

Bottom-Up DP

$c[i,j]$: length of the LCS of

$X_i = \langle x_1, \dots, x_i \rangle$ and $Y_j = \langle y_1, \dots, y_j \rangle$

$$c[i,j] = \begin{cases} 0 & \text{, if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{, if } x_i = y_j \\ \max(c[i-1,j], c[i,j-1]) & \text{, if } x_i \neq y_j \end{cases}$$

- The original sequences are $X = \langle x_1, \dots, x_m \rangle$ and $Y = \langle y_1, \dots, y_n \rangle$
- So, our goal is to determine $c[m,n]$
- What should be our table dimension and size?
  - 2D table **c[0:m,0:n]**

Bottom-Up DP

$c[i,j]$: length of the LCS of

$X_i = \langle x_1, \ldots, x_i \rangle$ and $Y_j = \langle y_1, \ldots, y_j \rangle$

$$c[i,j] = \begin{cases} 0 & \text{, if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{, if } x_i = y_j \\ \max(c[i-1,j], c[i,j-1]) & \text{, if } x_i \neq y_j \end{cases}$$

- From recurrence, the base case is when "i=0" or "j=0"

		Y		T	G	C	A	G
X			0	1	2	3	4	5
	0		0	0	0	0	0	0
A	1		0					
T	2		0					
C	3		0					
G	4		0					

Bottom-Up DP

$c[i,j]$: length of the LCS of

$X_i = \langle x_1, \ldots, x_i \rangle$ and $Y_j = \langle y_1, \ldots, y_j \rangle$

$$c[i,j] = \begin{cases} 0 & \text{, if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{, if } x_i = y_j \\ \max(c[i-1,j], c[i,j-1]) & \text{, if } x_i \neq y_j \end{cases}$$

- Which order should the table be populated? (What are the dependencies?)

		Y		T	G	C	A	G
X			0	1	2	3	4	5
	0		0	0	0	0	0	0
A	1		0					
T	2		0					
C	3		0					
G	4		0					

Bottom-Up DP

```
LCS-LENGTH (X, Y)
1.  m = length[X]
2.  n = length[Y]
3.  for i = 1 to m
4.      c[i,0] = 0
5.  for j = 0 to n
6.      c[0,j] = 0
7.  for i = 1 to m
8.      for j ← 1 to n
9.          if x_i = y_j
10.             c[i, j ] ← c[i–1, j–1] + 1
11.             b[i, j ] ← "↖"
12.         else if c[i–1, j ] ≥ c[i, j–1]
13.             c[i, j ] ← c[i– 1, j ]
14.             b[i, j ] ← "↑"
15.         else
16.             c[i, j ] ← c[i, j–1]
17.             b[i, j ] ← "←"
18. return c and b
```

$$c[i,j] = \begin{cases} 0 & \text{, if } i = 0 \text{ or } j = 0 \\ c[i-1,j-1]+1 & \text{, if } x_i = y_j \\ \max(c[i-1,j], c[i,j-1]) & \text{, if } x_i \neq y_j \end{cases}$$

$b[i,j]$ points to table entry whose subproblem we used in solving LCS of X_i and Y_j.

$c[m,n]$ contains the length of an LCS of X and Y.

Running Time: $\Theta(m \cdot n)$

Bottom-Up DP

```
LCS-LENGTH (X, Y)
1.   m = length[X]
2.   n = length[Y]
3.   for i = 1 to m
4.       c[i,0] = 0
5.   for j = 0 to n
6.       c[0,j] = 0
7.   for i = 1 to m
8.       for j ← 1 to n
9.           if x_i = y_j
10.              c[i, j ] ← c[i–1, j–1] + 1
11.              b[i, j ] ← "↖"
12.          else if c[i–1, j ] ≥ c[i, j–1]
13.              c[i, j ] ← c[i– 1, j ]
14.              b[i, j ] ← "↑"
15.          else
16.              c[i, j ] ← c[i, j–1]
17.              b[i, j ] ← "←"
18.  return c and b
```

$$c[i,j] = \begin{cases} 0 & \text{, if } i = 0 \text{ or } j = 0 \\ c[i-1,j-1]+1 & \text{, if } x_i = y_j \\ \max(c[i-1,j], c[i,j-1]) & \text{, if } x_i \neq y_j \end{cases}$$

		Y		T	G	C	A	G
X			0	1	2	3	4	5
	0		0	0	0	0	0	0
A	1		0					
T	2		0					
C	3		0					
G	4		0					

Bottom-Up DP

```
LCS-LENGTH (X, Y)
1.  m = length[X]
2.  n = length[Y]
3.  for i = 1 to m
4.      c[i,0] = 0
5.  for j = 0 to n
6.      c[0,j] = 0
7.  for i = 1 to m
8.      for j ← 1 to n
9.          if x_i = y_j
10.             c[i, j ] ← c[i–1, j–1] + 1
11.             b[i, j ] ← "↖"
12.         else if c[i–1, j ] ≥ c[i, j–1]
13.             c[i, j ] ← c[i– 1, j ]
14.             b[i, j ] ← "↑"
15.         else
16.             c[i, j ] ← c[i, j–1]
17.             b[i, j ] ← "←"
18. return c and b
```

$$c[i,j] = \begin{cases} 0 & \text{, if } i = 0 \text{ or } j = 0 \\ c[i-1,j-1]+1 & \text{, if } x_i = y_j \\ \max(c[i-1,j], c[i,j-1]) & \text{, if } x_i \neq y_j \end{cases}$$

		Y		T	G	C	A	G
X			0	1	2	3	4	5
	0		0	0	0	0	0	0
A	1		0	0↑				
T	2		0					
C	3		0					
G	4		0					

Bottom-Up DP

```
LCS-LENGTH (X, Y)
1.  m = length[X]
2.  n = length[Y]
3.  for i = 1 to m
4.      c[i,0] = 0
5.  for j = 0 to n
6.      c[0,j] = 0
7.  for i = 1 to m
8.      for j ← 1 to n
9.          if x_i = y_j
10.             c[i, j ] ← c[i–1, j–1] + 1
11.             b[i, j ] ← "↖"
12.         else if c[i–1, j ] ≥ c[i, j–1]
13.             c[i, j ] ← c[i– 1, j ]
14.             b[i, j ] ← "↑"
15.         else
16.             c[i, j ] ← c[i, j–1]
17.             b[i, j ] ← "←"
18. return c and b
```

$$c[i,j] = \begin{cases} 0 & \text{, if } i = 0 \text{ or } j = 0 \\ c[i-1,j-1]+1 & \text{, if } x_i = y_j \\ \max(c[i-1,j], c[i,j-1]) & \text{, if } x_i \neq y_j \end{cases}$$

		Y		T	G	C	A	G
X			0	1	2	3	4	5
	0		0	0	0	0	0	0
A	1		0	0↑	0↑			
T	2		0					
C	3		0					
G	4		0					

Bottom-Up DP

```
LCS-LENGTH (X, Y)
1.  m = length[X]
2.  n = length[Y]
3.  for i = 1 to m
4.      c[i,0] = 0
5.  for j = 0 to n
6.      c[0,j] = 0
7.  for i = 1 to m
8.      for j ← 1 to n
9.          if x_i = y_j
10.             c[i, j ] ← c[i–1, j–1] + 1
11.             b[i, j ] ← "↖"
12.         else if c[i–1, j ] ≥ c[i, j–1]
13.             c[i, j ] ← c[i– 1, j ]
14.             b[i, j ] ← "↑"
15.         else
16.             c[i, j ] ← c[i, j–1]
17.             b[i, j ] ← "←"
18. return c and b
```

$$c[i,j] = \begin{cases} 0 & \text{, if } i = 0 \text{ or } j = 0 \\ c[i-1,j-1]+1 & \text{, if } x_i = y_j \\ \max(c[i-1,j], c[i,j-1]) & \text{, if } x_i \neq y_j \end{cases}$$

		Y		T	G	C	A	G
X			0	1	2	3	4	5
	0		0	0	0	0	0	0
A	1		0	0↑	0↑	0↑		
T	2		0					
C	3		0					
G	4		0					

Bottom-Up DP

```
LCS-LENGTH (X, Y)
1.   m = length[X]
2.   n = length[Y]
3.   for i = 1 to m
4.        c[i,0] = 0
5.   for j = 0 to n
6.        c[0,j] = 0
7.   for i = 1 to m
8.        for j ← 1 to n
9.             if x_i = y_j
10.                 c[i, j ] ← c[i–1, j–1] + 1
11.                 b[i, j ] ← "↖"
12.            else if c[i–1, j ] ≥ c[i, j–1]
13.                 c[i, j ] ← c[i– 1, j ]
14.                 b[i, j ] ← "↑"
15.            else
16.                 c[i, j ] ← c[i, j–1]
17.                 b[i, j ] ← "←"
18.  return c and b
```

$$c[i,j] = \begin{cases} 0 & \text{, if } i = 0 \text{ or } j = 0 \\ c[i-1,j-1]+1 & \text{, if } x_i = y_j \\ \max(c[i-1,j], c[i,j-1]) & \text{, if } x_i \neq y_j \end{cases}$$

		Y		T	G	C	A	G
	X		0	1	2	3	4	5
		0	0	0	0	0	0	0
A		1	0	0↑	0↑	0↑	1↖	
T		2	0					
C		3	0					
G		4	0					

Bottom-Up DP

```
LCS-LENGTH (X, Y)
1.   m = length[X]
2.   n = length[Y]
3.   for i = 1 to m
4.       c[i,0] = 0
5.   for j = 0 to n
6.       c[0,j] = 0
7.   for i = 1 to m
8.       for j ← 1 to n
9.           if x_i = y_j
10.              c[i, j ] ← c[i–1, j–1] + 1
11.              b[i, j ] ← "↖"
12.          else if c[i–1, j ] ≥ c[i, j–1]
13.              c[i, j ] ← c[i– 1, j ]
14.              b[i, j ] ← "↑"
15.          else
16.              c[i, j ] ← c[i, j–1]
17.              b[i, j ] ← "←"
18.  return c and b
```

$$c[i,j] = \begin{cases} 0 & \text{, if } i = 0 \text{ or } j = 0 \\ c[i-1,j-1]+1 & \text{, if } x_i = y_j \\ \max(c[i-1,j], c[i,j-1]) & \text{, if } x_i \neq y_j \end{cases}$$

		Y		T	G	C	A	G
	X		0	1	2	3	4	5
		0	0	0	0	0	0	0
A		1	0	0↑	0↑	0↑	1↖	1←
T		2	0					
C		3	0					
G		4	0					

Bottom-Up DP

```
LCS-LENGTH (X, Y)
1.   m = length[X]
2.   n = length[Y]
3.   for i = 1 to m
4.        c[i,0] = 0
5.   for j = 0 to n
6.        c[0,j] = 0
7.   for i = 1 to m
8.        for j ← 1 to n
9.             if x_i = y_j
10.                 c[i, j ] ← c[i–1, j–1] + 1
11.                 b[i, j ] ← "↖"
12.            else if c[i–1, j ] ≥ c[i, j–1]
13.                 c[i, j ] ← c[i– 1, j ]
14.                 b[i, j ] ← "↑"
15.            else
16.                 c[i, j ] ← c[i, j–1]
17.                 b[i, j ] ← "←"
18.  return c and b
```

$$c[i,j] = \begin{cases} 0 & \text{, if } i = 0 \text{ or } j = 0 \\ c[i-1,j-1]+1 & \text{, if } x_i = y_j \\ \max(c[i-1,j], c[i,j-1]) & \text{, if } x_i \neq y_j \end{cases}$$

		Y		T	G	C	A	G
X			0	1	2	3	4	5
	0		0	0	0	0	0	0
A	1		0	0↑	0↑	0↑	1↖	1←
T	2		0	1↖				
C	3		0					
G	4		0					

Bottom-Up DP

```
LCS-LENGTH (X, Y)
1.   m = length[X]
2.   n = length[Y]
3.   for i = 1 to m
4.       c[i,0] = 0
5.   for j = 0 to n
6.       c[0,j] = 0
7.   for i = 1 to m
8.       for j ← 1 to n
9.           if x_i = y_j
10.              c[i, j ] ← c[i–1, j–1] + 1
11.              b[i, j ] ← "↖"
12.          else if c[i–1, j ] ≥ c[i, j–1]
13.              c[i, j ] ← c[i– 1, j ]
14.              b[i, j ] ← "↑"
15.          else
16.              c[i, j ] ← c[i, j–1]
17.              b[i, j ] ← "←"
18.  return c and b
```

$$c[i,j] = \begin{cases} 0 & \text{, if } i = 0 \text{ or } j = 0 \\ c[i-1,j-1]+1 & \text{, if } x_i = y_j \\ \max(c[i-1,j], c[i,j-1]) & \text{, if } x_i \neq y_j \end{cases}$$

		Y		T	G	C	A	G
X			0	1	2	3	4	5
	0		0	0	0	0	0	0
A	1		0	0↑	0↑	0↑	1↖	1←
T	2		0	1↖	1←	1←	1←	1↑
C	3		0	1↑	1↑	2↖	2←	2←
G	4		0	1↑	2↖	2↑	2↑	3↖

Bottom-Up DP

```
PRINT-LCS (b, X, i, j)
1.  if i = 0 or j = 0
2.      return
3.  if b[i, j] = "↖"
4.      PRINT-LCS(b, X, i–1, j–1)
5.      print x_i
6.  else if b[i, j] = "↑"
7.      PRINT-LCS(b, X, i–1, j)
8.  else
9.      PRINT-LCS(b, X, i, j–1)
```

		Y		T	G	C	A	G
		X	0	1	2	3	4	5
	0		0	0	0	0	0	0
A	1		0	0↑	0↑	0↑	1↖	1←
T	2		0	1↖	1←	1←	1←	1↑
C	3		0	1↑	1↑	2↖	2←	2←
G	4		0	1↑	2↖	2↑	2↑	3↖

Print: TCG

- Initial call is PRINT-LCS(b, X, m, n)
- When $b[i, j]$ = "↖", we have extended LCS by one character.

So LCS = entries with "↖" in them

Running Time: $O(m + n)$

Thank You!