The Longest-Common-Subsequence Problem (1/3)

- Given a sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$, another sequence $Z = \langle z_1, z_2, \ldots, z_k \rangle$ is a subsequence of X if there exists a strictly increasing sequence $\langle i_1, i_2, \ldots, i_k \rangle$ of indices of X such that for all $j = 1, 2, \ldots, k$, we have $x_{i_j} = z_j$.
  - For example, *Z* = <*B*, *C*, *D*, *B*> is a subsequence of *X* = <*A*, *B*, *C*, *B*, *D*, *A*, *B*> with corresponding index sequence <2, 3, 5, 7>.
- Given two sequences *X* and *Y*, we say that a sequence *Z* is a common subsequence of *X* and *Y* if *Z* is a subsequence of both *X* and *Y*.

The Longest-Common-Subsequence Problem (2/3)

- For example, if *X* = <*A*, *B*, *C*, *B*, *D*, *A*, *B*> and *Y* = <*B*, *D*, *C*, *A*, *B*, *A* >, the sequence <*B*, *C*, *A*> is a common subsequence of both *X* and *Y*.
- The sequence <*B*, *C*, *A*> is not a longest common subsequence (LCS) of *X* and *Y*, however, since it has length 3 and the sequence <*B*, *C*, *B*, *A*>, which is also common to both *X* and *Y*, has length 4.
- The sequence <*B*, *C*, *B*, *A*> is an LCS of *X* and *Y*, as is the sequence <*B*, *D*, *A*, *B*>, since *X* and *Y* have no common subsequence of length 5 or greater.

The Longest-Common-Subsequence Problem (3/3)

- In the longest-common-subsequence problem, we are given two sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ and wish to find a maximum-length common subsequence of *X* and *Y*.
- **Example1:** S_1 = AGCGTAG, S_2 = GTCAGA, please find a maximum-length common subsequence.

Step 1: Characterizing a Longest Common Subsequence (1/2)

- In a brute-force approach to solving the LCS problem, we would
  - enumerate all subsequences of *X* (2^m subsequences in total) and
  - check each subsequence to see whether it is also a subsequence of *Y*,
  - keeping track of the longest subsequence we find.

➡ Requires exponential time, impractical for long sequences.

Step 1: Characterizing a Longest Common Subsequence (2/2)

- Given a sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$, we define the i th prefix of *X*, for $i = 0, 1, \ldots, m$, as $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.
  – For example, if X = <*A*, *B*, *C*, *B*, *D*, *A*, *B*>, then X_4 = <*A*, *B*, *C*, *B*> and X_0 is the empty sequence.

Theorem 15.1 (Optimal substructure of an LCS)

Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ 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 $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

Step 2: A Recursive Solution (1/2)

- **Theorem 15.1** implies that we should examine either one or two subproblems when finding an LCS of $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$.
  – If $x_m = y_n$, we must find an LCS of X_{m-1} and Y_{n-1}. Appending $x_m = y_n$ to this LCS yields an LCS of X and Y.
  – If $x_m \neq y_n$, then we must solve two subproblems: finding an LCS of X_{m-1} and Y and finding an LCS of X and Y_{n-1}. Whichever of these two LCSs is longer is an LCS of X and Y.

Step 2: A Recursive Solution (2/2)

- Let us define $c[i, j]$ to be the length of an LCS of the sequences X_i and Y_j.

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 , \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j , \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j . \end{cases} \quad (15.9)$$

Step 3: Computing the Length of an LCS (1/10)

- Procedure LCS-LENGTH takes two sequences X = <$x_1, x_2, \ldots, x_m$> and Y = <$y_1, y_2, \ldots, y_n$> as inputs.
  – It stores the $c[i, j]$ values in a table $c[0..m, 0..n]$, and it computes the entries in row-major order.
  – The procedure also maintains the table $b[1..m, 1..n]$ to help us construct an optimal solution.
  – The procedure returns the b and c tables; $c[m, n]$ contains the length of an LCS of X and Y.

Step 3: Computing the Length of an LCS (2/10)

LCS-LENGTH(X, Y)

```
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

Suppose X = <A, B, C, B, D, A, B> and Y = <B, D, C, A, B, A>

j		0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Step 3: Computing the Length of an LCS (3/10)

LCS-LENGTH(X, Y)

```
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

Suppose X = <A, B, C, B, D, A, B> and Y = <B, D, C, A, B, A>

j		0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Step 3: Computing the Length of an LCS (4/10)

LCS-LENGTH(X, Y)

```
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

Suppose X = <A, B, C, B, D, A, B> and Y = <B, D, C, A, B, A>

j		0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Step 3: Computing the Length of an LCS (5/10)

LCS-LENGTH(X, Y)

```
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

Suppose X = <A, B, C, B, D, A, B> and Y = <B, D, C, A, B, A>

j		0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Step 3: Computing the Length of an LCS (6/10)

Suppose X = <*A*, *B*, *C*, *B*, *D*, *A*, *B*> and Y = <*B*, *D*, *C*, *A*, *B*, *A*>

```
LCS-LENGTH(X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Step 3: Computing the Length of an LCS (7/10)

Suppose X = <*A*, *B*, *C*, *B*, *D*, *A*, *B*> and Y = <*B*, *D*, *C*, *A*, *B*, *A*>

```
LCS-LENGTH(X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0						
6	A	0						
7	B	0						

Step 3: Computing the Length of an LCS (8/10)

Suppose X = <*A*, *B*, *C*, *B*, *D*, *A*, *B*> and Y = <*B*, *D*, *C*, *A*, *B*, *A*>

```
LCS-LENGTH(X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0						
7	B	0						

Step 3: Computing the Length of an LCS (9/10)

Suppose X = <*A*, *B*, *C*, *B*, *D*, *A*, *B*> and Y = <*B*, *D*, *C*, *A*, *B*, *A*>

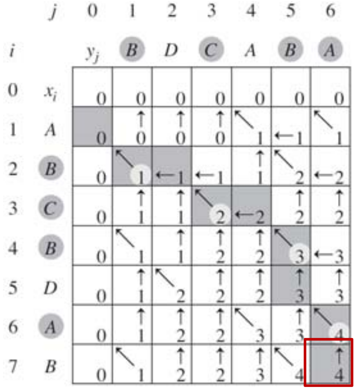
```
LCS-LENGTH(X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0						

Step 3: Computing the Length of an LCS (10/10)

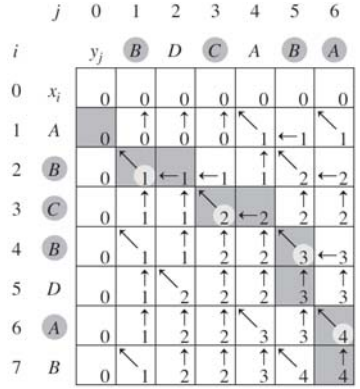
```
LCS-LENGTH(X, Y)
  m = X.length
  n = Y.length
  let b[1..m, 1..n] and c[0..m, 0..n] be new tables
  for i = 1 to m
      c[i, 0] = 0
  for j = 0 to n
      c[0, j] = 0
  for i = 1 to m
      for j = 1 to n
          if x_i == y_j
              c[i, j] = c[i − 1, j − 1] + 1
              b[i, j] = "↖"
          elseif c[i − 1, j] ≥ c[i, j − 1]
              c[i, j] = c[i − 1, j]
              b[i, j] = "↑"
          else c[i, j] = c[i, j − 1]
              b[i, j] = "←"
  return c and b
```

Suppose X = <A, B, C, B, D, A, B> and Y = <B, D, C, A, B, A>



Step 4: Constructing an LCS

```
PRINT-LCS(b, X, i, j)
  if i == 0 or j == 0
      return
  if b[i, j] == "↖"
      PRINT-LCS(b, X, i − 1, j − 1)
      print x_i
  elseif b[i, j] == "↑"
      PRINT-LCS(b, X, i − 1, j)
  else PRINT-LCS(b, X, i, j − 1)
```



Outine

- Rod Cutting
- Matrix-Chain Multiplication
- Elements of Dynamic Programming
- Longest Common Subsequence
- ***Optimal Binary Search Trees***

Introduction (1/2)

- Suppose that we are designing a program to translate text from English to Chinese.
  - We could perform lookup operations by building a binary search tree with n English words as keys.
  - Since the number of nodes visited when searching for a key in a binary search tree equals one plus the depth of the node containing the key, we want words that occur frequently in the text to be placed nearer the root.

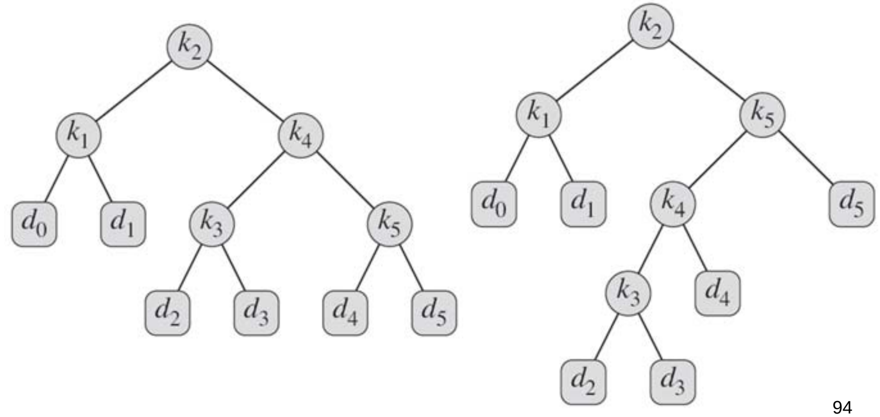
Introduction (2/2)

- We are given a sequence $K = \langle k_1, k_2, \ldots, k_n \rangle$ of n distinct keys in sorted order ($k_1 < k_2 < \ldots < k_n$), and we wish to build a binary search tree from these keys.
- For each key k_i, we have a probability p_i that a search will be for k_i.
- Some searches may be for values not in K, and so we also have $n + 1$ "dummy keys" $d_0, d_1, d_2, \ldots, d_n$ representing values not in K.
- For each dummy key d_i, we have a probability q_i that a search will correspond to d_i.

Two Binary Search Trees

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$$



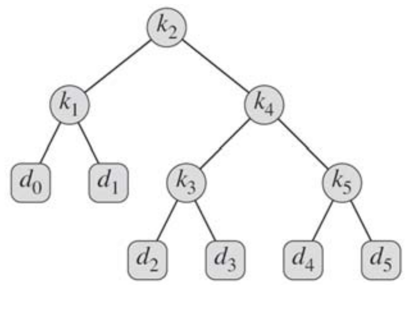
Expected Cost of a Search in T (1/2)

- Let us assume that the actual cost of a search equals the number of nodes examined, i.e., the depth of the node found by the search in T, plus 1. Then the expected cost of a search in T is

$$\begin{aligned} \mathrm{E}\left[\text{search cost in } T\right] &= \sum_{i=1}^{n} (\mathrm{depth}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^{n} (\mathrm{depth}_T(d_i) + 1) \cdot q_i \\ &= 1 + \sum_{i=1}^{n} \mathrm{depth}_T(k_i) \cdot p_i + \sum_{i=0}^{n} \mathrm{depth}_T(d_i) \cdot q_i \ , \end{aligned}$$

where depth_T denotes a node's depth in the tree T.

Expected Cost of a Search in T (2/2)



i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

node	depth	probability	contribution
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	2	0.05	0.15
k_4	1	0.10	0.20
k_5	2	0.20	0.60
d_0	2	0.05	0.15
d_1	2	0.10	0.30
d_2	3	0.05	0.20
d_3	3	0.05	0.20
d_4	3	0.05	0.20
d_5	3	0.10	0.40
Total			2.80