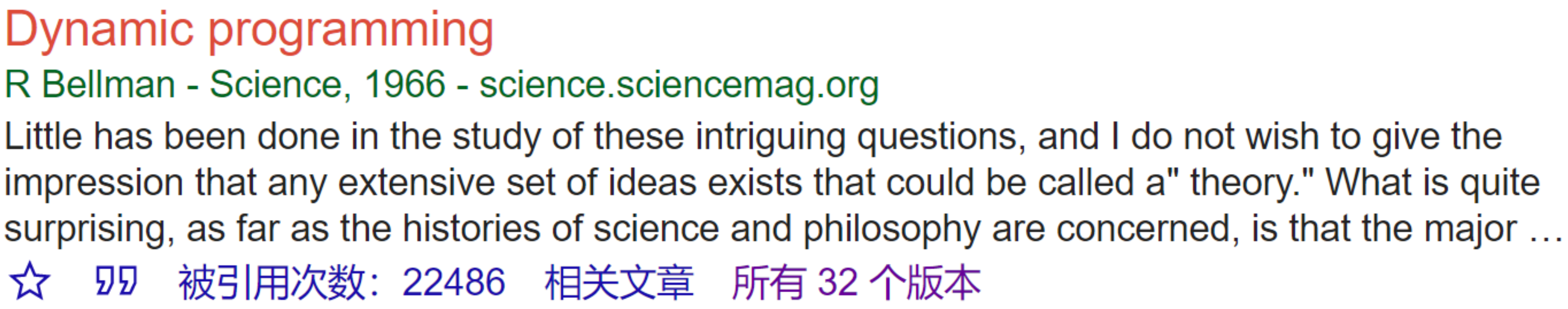
15 Dynamic Programming

Richard Bellman. Dynamic Programming. Princeton University Press, 1957.



Dynamic programming

R Bellman - Science, 1966 - science.sciencemag.org

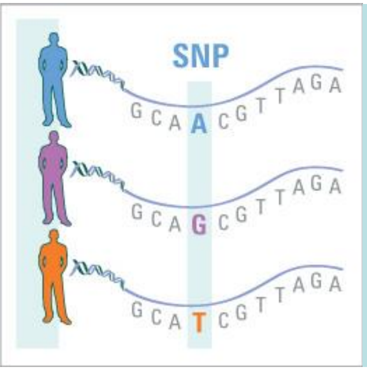
Little has been done in the study of these intriguing questions, and I do not wish to give the impression that any extensive set of ideas exists that could be called a" theory." What is quite surprising, as far as the histories of science and philosophy are concerned, is that the major …

☆ ” 被引用次数：22486 相关文章 所有 32 个版本

15 Dynamic Programming

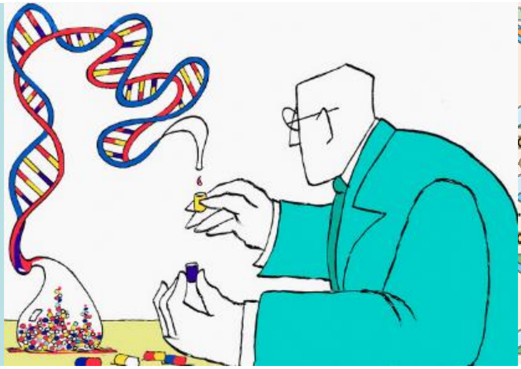
- Scheduling two automobile assembly lines
- Steel rod cutting
- Matrix-chain multiplication
- Characteristics of dynamic programming
- **Longest common subsequence**
  最长相同子序列
- Optimal binary search trees

15.4 Longest common subsequence

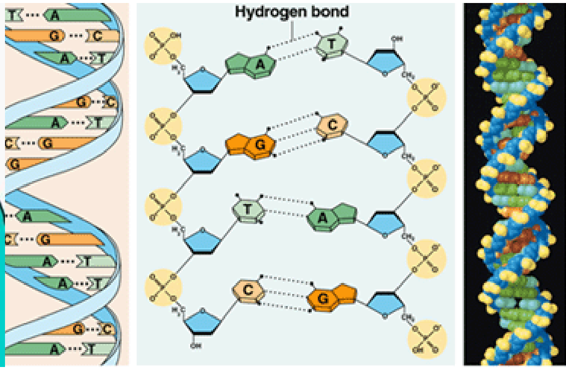


亲缘关系

转基因食物



制药



是否病变

- Compare the DNA of two (or more) different organisms.
  （比较两个不同生物体的 DNA）
- A strand of DNA: a string of molecules, *bases* (Adenine, Guanine, Cytosine, Thymine) （DNA的碱基对：A腺嘌呤,T胸腺嘧啶,C胞嘧啶,G鸟嘌呤）
  （多个分子bases以不同的组合方式构成一个 DNA 串）
  - a strand of DNA ∈ finite set {A, C, G, T}
  - for example, two organism’s DNA

    S_1= ACCGGTCGAGTGCGCGGAAGCCGGCCGAA

    S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA
- Goal: how “similar” the two strands are?（研究目标：确定两个DNA序列的相似程度）

15.4 Longest common subsequence

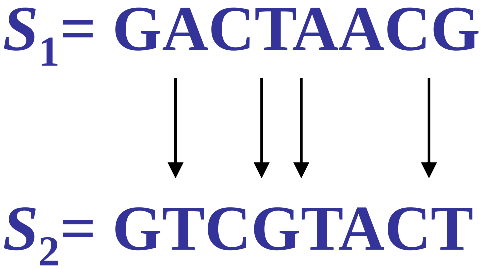
- **How to define the similarity of S_1 and S_2 ?**

  (1) S_1 and S_2 are similar if one is a substring of the other.

  （如果一个串是另一个串的子串，则S_1和S_2相似）

  S_1= GTCGTCGGAA　　S_2= GTCGTCG

  (2) If the number of changes needed to turn one into the other is small.（将一个串变换为另一个串，变换数最少）



S_1= GACTAACG

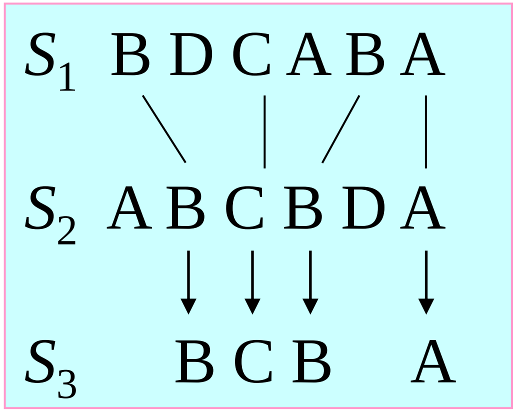
S_2= GTCGTACT

15.4 Longest common subsequence

- **How to define the similarity of S_1 and S_2 ?**

  (3) By finding a third strand S_3, in which the bases in S_3 appear in each of S_1 and S_2; these bases appear in the same order, but not necessarily consecutively. The longer the strand S_3 we can find, the more similar S_1 and S_2 are.

  （寻找第3个串S_3，S_3中的所有bases都包含在S_1和S_2中，这些bases在S_1和S_2中不一定连续排列，但必须是按顺序排列的。S_3越长，则S_1和S_2的相似度就越大。）



- **Formalize this notion of similarity as the longest-common-subsequence problem.（以这种相似性意义作为最长相同子序列问题的形式化定义）**

15.4 Longest common subsequence

- **Subsequence**: given a sequence $X = \langle x_1, x_2, ..., x_m \rangle$, another sequence $Z = \langle z_1, z_2, ..., z_k \rangle$ is a *subsequence* of X if there exists a strictly increasing sequence $\langle i_1, i_2, ..., i_k \rangle$ of indices of X such that for all $j = 1, 2, ..., k$, we have $x[i_j] = z_j$. （给定序列 X，如果存在 X 的索引的一个严格增序列 $< i_1, i_2, ..., i_k >$，使得对所有的 $j = 1, 2, ..., k$, 其中 $i_j \in \{1, 2, \dots, m\}$，且都有 $x[i_j] = z_j$，则称 $Z = < z_1, z_2, ..., z_k >$ 是 X 的子序列。）

For example, $Z =$ B, C, D, B

Z is a subsequence of $X = A, B, C, B, D, A, B$

with corresponding index sequence

$< i_1, i_2, i_3, i_4 > = 2, 3, 5, 7$

从 ***X*** 中以增序任意抽取一系列元素组成的新序列成为 ***X*** 的自序列

15.4 Longest common subsequence

- **Common subsequence: Given two sequences X and Y, a sequence Z is a *common subsequence* of X and Y if Z is a subsequence of both X and Y.**

  （给定两个序列 *X* 和 *Y*，如果序列 *Z* 是 *X* 的子序列同时也是 *Y* 的子序列，则称 *Z* 为 *X* 和*Y* 的相同子序列）

For example,

$X = A, B, C, B, D, A, B$	$X = A, B, C, B, D, A, B$
$Z_1 = B, C, A$	$Z_2 = B, C, B, A$
$Y = B, D, C, A, B, A$	$Y = B, D, C, A, B, A$

Z_1 is a common subsequence, but not a LCS of X and Y.

Z_2 is an LCS of X and Y (<B, D, A, B> is also an LCS).

15.4 Longest common subsequence

❒ LCS Problem

Given two sequences X= <x_1, x_2, ..., x_m> and Y= <y_1, y_2, ..., y_n>, how to find a maximum-length common subsequence of X and Y .

（给定两个子序列 X 和 Y，如何寻找 X 和 Y 的长度最大的相同子序列）

- **LCS problem can be solved efficiently using dynamic programming.**

Step 1: Characterizing an LCS

$$X = \langle x_1, x_2, ..., x_m \rangle \text{ and } Y = \langle y_1, y_2, ..., y_n \rangle$$

- **A brute-force approach**
  - **Enumerate all subsequences of *X* and check each subsequence to see if it is also a subsequence of *Y*.**（列举出 *X* 的所有子序列，逐项核查这些子序列是否为 *Y* 的子序列）
  - **Each subsequence of *X* corresponds to a subset of the indices {1, 2, ..., *m*} of *X*. There are 2^m subsequences of *X* . Exponential time, impractical for long sequences.**（*X* 的子序列对应*X* 的索引{1, 2, ..., *m*}的某个子集 。有2^m个 *X* 的子序列。需要指数运算时间，当序列较大时实际不可行）

1个元素：$C(1,m)$， $<x_1>, <x_2>, ..., <x_m>$;

2个元素：$C(2,m)$， $<x_1,x_2>, <x_1,x_3>, ..., <x_{m-1} ,x_m>$;

…… ；

m个元素：$C(m,m)$， $<x_1, x_2, ..., x_m>$

$$\sum_{i=1}^{m} C(i,m)=2^m$$

Step 1: Characterizing an LCS

$X = <x_1, x_2, ..., x_m>$ and $Y = <y_1, y_2, ..., y_n>$

- The LCS problem has an optimal-substructure property.

  （LCS问题具有最优子结构属性）

- Natural classes of subproblems: prefixes

  （子问题的自然分类：前缀）

  $X_i = <x_1, . . . , x_i >$, the ith prefix of X , for $i = 0, 1, ..., m$

  $Y_j = <y_1, . . . , y_j >$, the jth prefix of Y , for $j = 0, 1, ..., n$

  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.

Step 1: Characterizing an LCS

- **Theorem 15.1: (Optimal substructure of an LCS)**

  Let $Z = < z_1, ..., z_{k-1}\ , z_k >$ be any LCS of

  $X = < x_1, ..., x_{m-1}\ , x_m >$

  and $Y = < y_1, ..., y_{n-1}\ , \ y_n >$.

  （设 Z 是 X 和 Y 的任意 LCS）

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 $z_k \neq x_m \Rightarrow Z$ is an LCS of X_{m-1} and Y.**
3. **If $x_m \neq y_n$, then $z_k \neq y_n \Rightarrow Z$ is an LCS of X and Y_{n-1}.**

Step 1: Characterizing an LCS

$X = < x_1 , \ldots , x_{m-1} , x_m >$
$Z = < z_1 , \ \ldots , z_{k-1} , z_k >$
$Y = < y_1 , \ldots , y_{n-1} , y_n >$

❐ **Theorem 15.1: (Optimal substructure of an LCS)**

Let $Z = <z_1, \ldots , z_k>$ be any LCS of $X = <x_1, \ldots, x_m>$ and $Y = <y_1, \ldots, y_n>$.

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}.

Proof: **First show that $z_k=x_m=y_n$. Suppose $z_k \neq x_m$, Then $Z' = <z_1 , \ldots , z_k , x_m>$ is a common subsequence (CS) of X and Y , and has length k+1⇒ Z' is a longer CS than Z ⇒ contradicts Z being an LCS.**

（设$z_k \neq x_m$, 令$Z' = <z_1 , \ldots , z_k , x_m>$，则 Z' 是 X 和 Y 的相同子序列，且 length(Z')=k+1 ⇒ Z' 是比 Z 更长的子序列 ⇒ 与题设 Z 是LCS矛盾）

Now show Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}. Clearly, it's a CS. Now suppose there exists a CS W of X_{m-1} and Y_{n-1} that's longer than Z_{k-1} ⇒length(W) ≥ k. Make subsequence W' by appending x_m to W. W' is CS of X and Y , length(W')≥k+1 ⇒ contradicts Z being an LCS.

（显然，Z_{k-1} 是X_{m-1}和Y_{n-1}的 a CS，设W 是X_{m-1}和Y_{n-1}的 a CS，且 length(W)≥k, 将x_m附加到W后面得到W'，则W' 是X_m和Y_n的 a CS，且 length(W')≥k+1 ⇒ 与题设 Z 是LCS矛盾）

Step 1: Characterizing an LCS

$X = < x_1 , \dots , x_i , \dots , x_m >$
$Z = < z_1 , \dots , z_k >$
$Y = < y_1 , \dots , y_j , \dots , y_n >$

- **Theorem 15.1: (Optimal substructure of an LCS)**

  Let $Z = <z_1, \dots , z_k>$ be any LCS of $X = <x_1, \dots, x_m>$ and $Y = <y_1, \dots, y_n>$.

  2. If $x_m \neq y_n$, then $z_k \neq x_m \Rightarrow Z$ is an LCS of X_{m-1} and Y.

  3. If $x_m \neq y_n$, then $z_k \neq y_n \Rightarrow Z$ is an LCS of X and Y_{n-1}.

Proof:

2. If $z_k \neq x_m$, then Z is a CS of X_{m-1} and Y . Suppose there exists a subsequence W of X_{m-1} and Y with length > k. Then W is a CS of X and $Y \Rightarrow$ contradicts Z being an LCS.

（若$z_k \neq x_m$, 则 Z 是 X_{m-1} 和 Y 的 a CS. 设存在一个X_{m-1} 和 Y 的子序列 W，其 length(W)>k, 那么，W 是 X 和 Y 的 a CS $\Rightarrow$ 与题设 Z 是 an LCS 矛盾 ）

3. Symmetric to 2.

Step 1: Characterizing an LCS

- **Theorem 15.1: (Optimal substructure of an LCS)**

  Let $Z = <z_1, ... , z_{k-1} , z_k>$ be any LCS of $X = <x_1, x_2, ..., x_m>$ and $Y = <y_1, y_2, ..., y_n>$.

  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 $z_k \neq x_m \Rightarrow Z$ is an LCS of X_{m-1} and Y.**
  3. **If $x_m \neq y_n$, then $z_k \neq y_n \Rightarrow Z$ is an LCS of X and Y_{n-1}.**

- **Theorem 15.1 shows that an LCS of two sequences contains within it an LCS of prefixes of the two sequences.**

  两个序列的 an LCS 包含了这两个序列的前缀（子问题）的 an LCS

- **the LCS problem has an optimal-substructure property.**

Step 1: Characterizing an LCS

- **Optimal-substructure property: Theorem 15.1 shows that an LCS of two sequences contains within it an LCS of prefixes of the two sequences.**

  两个序列的 **an LCS** 包含了这两个序列的前缀（子问题）的 **an LCS**

- **For example**

$$X = < x_1 , x_2 , \ldots , x_m >$$
$$Z = < z_1 , z_2 , \ldots , z_k >$$
$$Y = < y_1 , y_2 , \ldots , y_n >$$

$$X_{m\text{-}1} = < x_1 , x_2 , \ldots , x_{m\text{-}1} >$$
$$Z_{k\text{-}1} = < z_1 , z_2 , \ldots , z_{k\text{-}1} >$$
$$Y_{n\text{-}1} = < y_1 , y_2 , \ldots , y_{n\text{-}1} >$$

$$x_m = y_n$$

$$X_{m\text{-}1} = <x_1 , x_2 , \ldots , x_{m\text{-}1} >$$
$$Z = < z_1 , z_2 , \ldots , z_k >$$
$$Y = < y_1 , y_2 ,\ldots , y_n >$$

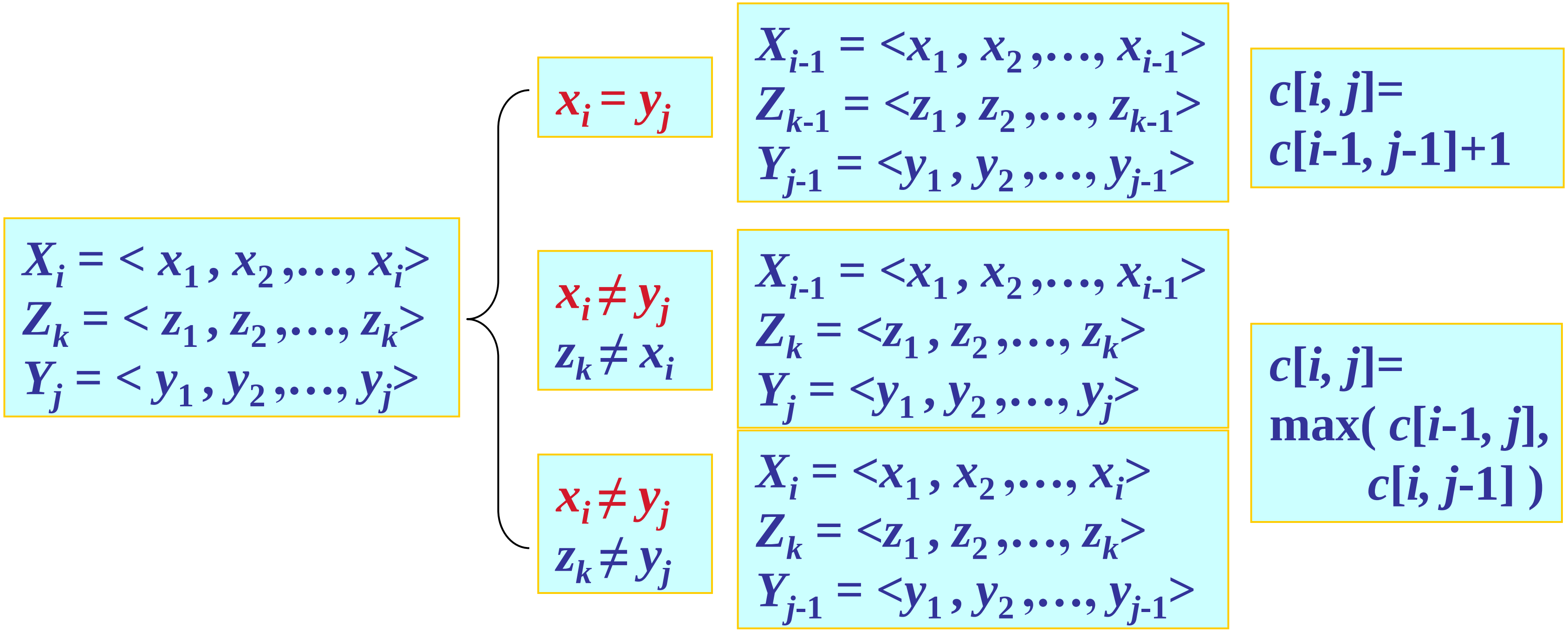
$$x_m \neq y_n$$
$$z_k \neq x_m$$

$$X = < x_1 , x_2 , \ldots , x_m >$$
$$Z = < z_1 , z_2 , \ldots , z_k >$$
$$Y_{n\text{-}1} = < y_1 , y_2 , \ldots , y_{n\text{-}1} >$$

$$x_m \neq y_n$$
$$z_k \neq y_n$$

Step 2: A recursive solution

- **Theorem 15.1 implies that there are either one or two subproblems to examine when finding an LCS of *X* and *Y*.**
  （当求 an LCS时，定理表明了有一个或两个子问题需要考虑）
- **Let $c[i, j]$ be the length of an LCS of X_i and Y_j ($c[i, j] = 0$, $i \cdot j = 0$).**



$X_i = < x_1 , x_2 ,\ldots, x_i>$
$Z_k = < z_1 , z_2 ,\ldots, z_k>$
$Y_j = < y_1 , y_2 ,\ldots, y_j>$

$x_i = y_j$

$X_{i\text{-}1} = <x_1 , x_2 ,\ldots, x_{i\text{-}1}>$
$Z_{k\text{-}1} = <z_1 , z_2 ,\ldots, z_{k\text{-}1}>$
$Y_{j\text{-}1} = <y_1 , y_2 ,\ldots, y_{j\text{-}1}>$

$c[i, j]=$
$c[i\text{-}1, j\text{-}1]+1$

$x_i \neq y_j$
$z_k \neq x_i$

$X_{i\text{-}1} = <x_1 , x_2 ,\ldots, x_{i\text{-}1}>$
$Z_k = <z_1 , z_2 ,\ldots, z_k>$
$Y_j = <y_1 , y_2 ,\ldots, y_j>$

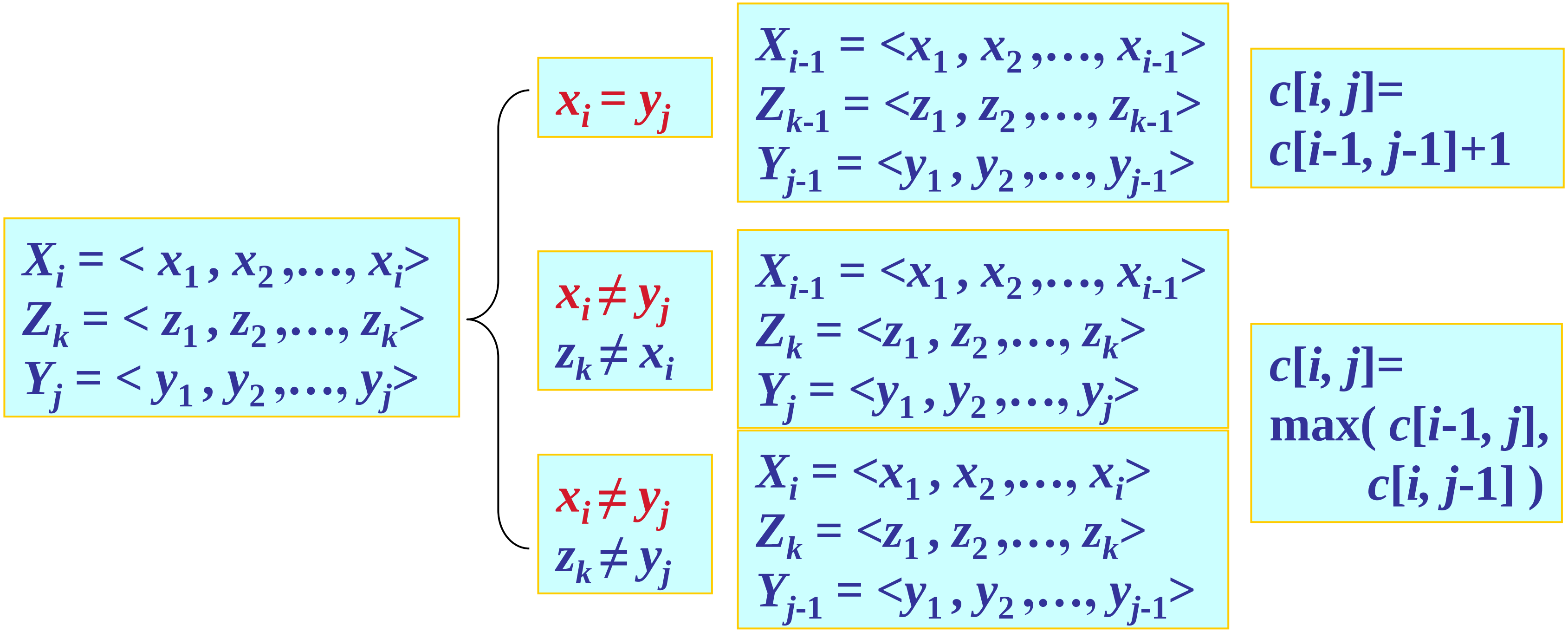
$x_i \neq y_j$
$z_k \neq y_j$

$X_i = <x_1 , x_2 ,\ldots, x_i>$
$Z_k = <z_1 , z_2 ,\ldots, z_k>$
$Y_{j\text{-}1} = <y_1 , y_2 ,\ldots, y_{j\text{-}1}>$

$c[i, j]=$
$\max(c[i\text{-}1, j],$
$c[i, j\text{-}1])$

Step 2: A recursive solution

- **The optimal substructure of the LCS problem gives the recursive formula.**（由LCS问题的最优子结构可导出递归公式）



$X_i = < x_1 , x_2 ,\ldots, x_i>$
$Z_k = < z_1 , z_2 ,\ldots, z_k>$
$Y_j = < y_1 , y_2 ,\ldots, y_j>$

$x_i = y_j$

$X_{i-1} = <x_1 , x_2 ,\ldots, x_{i-1}>$
$Z_{k-1} = <z_1 , z_2 ,\ldots, z_{k-1}>$
$Y_{j-1} = <y_1 , y_2 ,\ldots, y_{j-1}>$

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

$x_i \neq y_j$
$z_k \neq x_i$

$X_{i-1} = <x_1 , x_2 ,\ldots, x_{i-1}>$
$Z_k = <z_1 , z_2 ,\ldots, z_k>$
$Y_j = <y_1 , y_2 ,\ldots, y_j>$

$x_i \neq y_j$
$z_k \neq y_j$

$X_i = <x_1 , x_2 ,\ldots, x_i>$
$Z_k = <z_1 , z_2 ,\ldots, z_k>$
$Y_{j-1} = <y_1 , y_2 ,\ldots, y_{j-1}>$

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

$$c[i,j]=\begin{cases}0 & \text{if } i=0 \text{ or } j=0,\\ c[i-1,j-1]+1, & \text{if } i,j>0 \text{ and } x_i=y_j,\\ \max(c[i-1,j\ ],\ c[i,j-1]), & \text{if } i,j>0 \text{ and } x_i\neq y_j.\end{cases} \qquad (15.14)$$

Step 2: A recursive solution

- **The overlapping-subproblems property of LCS （重叠子问题）**

$X_{i-1} = < x_1 , x_2 , \dots , x_{i-1} >$
$Y_j \ \ = < y_1 , y_2 , \dots , y_j >$

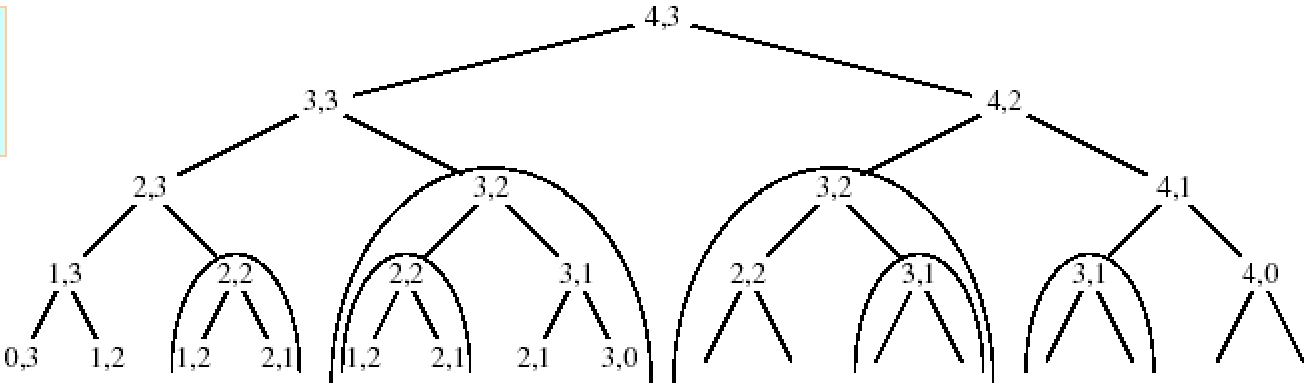
$X_i \ \ = < x_1 , x_2 , \dots , x_i >$
$Y_{j-1} = < y_1 , y_2 , \dots , y_{j-1} >$

⇒

$X_{i-1} = < x_1 , x_2 , \dots , x_{i-1} >$
$Y_{j-1} = < y_1 , y_2 , \dots , y_{j-1} >$

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ c[i-1, j-1]+1, & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max(c[i-1, j\ ],\ c[i, j-1]) \ , & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases} \qquad (15.14)$$

For example:
X_4 , Y_3



Step 3: Computing the length of an LCS

$$c[i,j]=\begin{cases}0 & \text{if } i=0 \text{ or } j=0,\\ c[i-1,j-1]+1, & \text{if } i,j>0 \text{ and } x_i=y_j,\\ \max(c[i-1,j],\ c[i,j-1]), & \text{if } i,j>0 \text{ and } x_i\neq y_j.\end{cases} \qquad (15.14)$$

- **Based on equation (15.14), we could easily write an recursive algorithm.**

$X_m = < x_1, x_2, \ldots, x_m >$
$Y_n = < y_1, y_2, \ldots, y_n >$

$X_{i\text{-}1} = < x_1, \ldots, x_{i\text{-}1} >$
$Y_j = < y_1, \ldots, y_j >$

$X_i = < x_1, \ldots, x_i >$
$Y_{j\text{-}1} = < y_1, \ldots, y_{j\text{-}1} >$

⇒

$X_{i\text{-}1} = < x_1, \ldots, x_{i\text{-}1} >$
$Y_{j\text{-}1} = < y_1, \ldots, y_{j\text{-}1} >$

- **Recursive algorithm?**

$T(m,n)=T(m\text{-}1,n)+T(m,n\text{-}1)+1$

- **Running time?**

$T(m,n)=\Omega(2^m+2^n)$

exponential-time

guess, then prove

Step 3: Computing the length of an LCS

- Based on equation (15.14), we could easily write an **exponential-time recursive** algorithm.

$$c[i,j]=\begin{cases}0 & \text{if } i=0 \text{ or } j=0,\\ c[i-1,j-1]+1, & \text{if } i,j>0 \text{ and } x_i=y_j,\\ \max(c[i-1,j],\ c[i,j-1])\ , & \text{if } i,j>0 \text{ and } x_i\neq y_j.\end{cases} \qquad (15.14)$$

$$X_m = < x_1, x_2, \ldots, x_m >$$
$$Y_n = < y_1, y_2, \ldots, y_n >$$

Recursive algorithm:

exponential-time

- Use dynamic programming to compute the solutions bottom up （使用动态规划法按自底向上方法进行求解）

Step 3: Computing the length of an LCS

$X_m = \langle x_1, x_2, \ldots, x_m \rangle$
$Y_n = \langle y_1, y_2, \ldots, y_n \rangle$

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

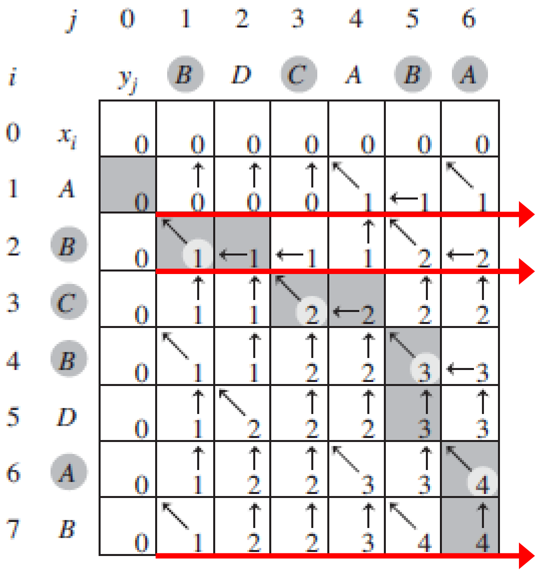
Use table $b[1..m, 1..n]$ to simplify construction of an optimal solution.
（使用*b*表构造最优解）

```
LCS-LENGTH($X$, $Y$)            // $X$ and $Y$ as inputs
$m \leftarrow length[X]$
$n \leftarrow length[Y]$
for $i \leftarrow 1$ to $m$        // Table $c[0..m, 0..n]$ stores $c[i, j]$,
    do $c[i, 0] \leftarrow 0$     // computed in row-major order.
for $j \leftarrow 0$ to $n$
    do $c[0, j] \leftarrow 0$
for $i \leftarrow 1$ to $m$
    for $j \leftarrow 1$ to $n$
        if $x_i = y_j$
            $c[i, j] \leftarrow c[i-1, j-1] + 1$
            $b[i, j] \leftarrow$ "↖"
        else if $c[i-1, j] \geq c[i, j-1]$  //use $X_{i-1}, Y_j$
                $c[i, j] \leftarrow c[i-1, j]$
                $b[i, j] \leftarrow$ "↑"
            else  $c[i, j] \leftarrow c[i, j-1]$
                $b[i, j] \leftarrow$ "←"
return $c$ and $b$
```

Step 3: Computing the length of an LCS

$X_i = \langle x_1, x_2, \ldots, x_i \rangle$
$Y_j = \langle y_1, y_2, \ldots, y_j \rangle$

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



i	j	0	1	2	3	4	5	6
	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	↖1	↑2	↑2	↑3	↖4	↑4

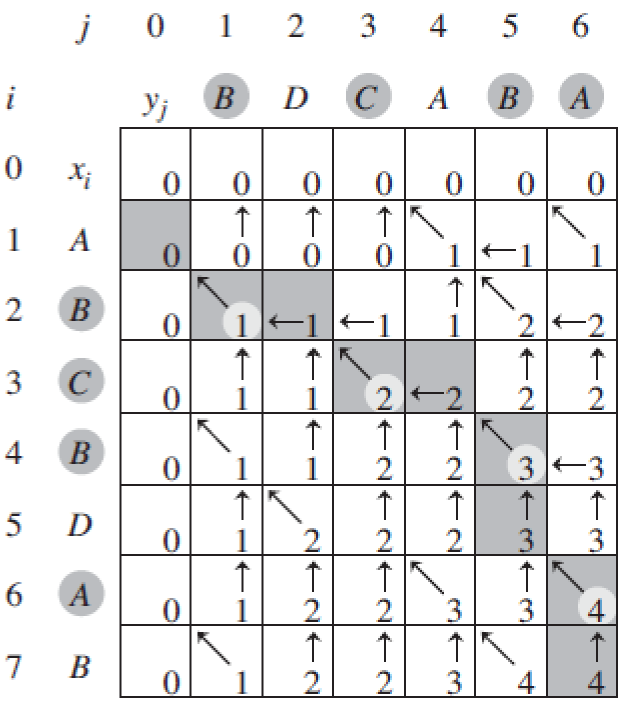
```
LCS-LENGTH(X, Y)                         // X and Y as inputs
$m \leftarrow length[X]$
$n \leftarrow length[Y]$
for $i \leftarrow 1$ to $m$              // Table c[0..m, 0..n] stores c[i,j],
    do $c[i, 0] \leftarrow 0$            // computed in row-major order.
for $j \leftarrow 0$ to $n$
    do $c[0, j] \leftarrow 0$
for $i \leftarrow 1$ to $m$
    for $j \leftarrow 1$ to $n$
        if $x_i = y_j$
            $c[i, j] \leftarrow c[i-1, j-1] + 1$
            $b[i, j] \leftarrow$ "↖"
        else if $c[i-1, j] \geq c[i, j-1]$            //use $X_{i-1}$, $Y_j$
            $c[i, j] \leftarrow c[i-1, j]$
            $b[i, j] \leftarrow$ "↑"
        else  $c[i, j] \leftarrow c[i, j-1]$          //use $X_i$, $Y_{j-1}$
            $b[i, j] \leftarrow$ "←"
return c and b
```

Running time?

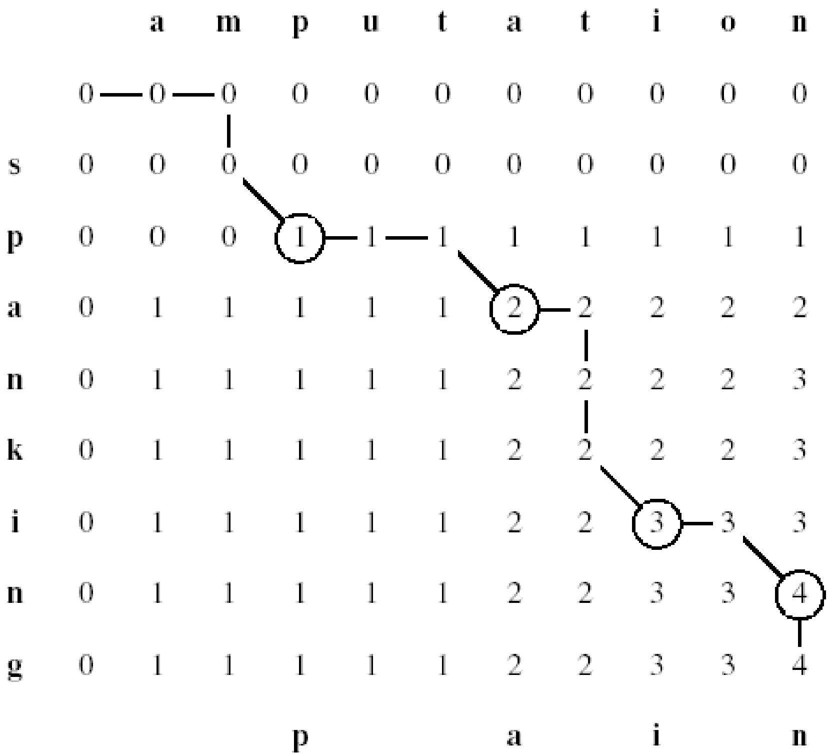
$T(m, n) = O(mn)$

Step 3: Computing the length of an LCS

- The running time of the procedure is *O*(*mn*), since each table entry takes *O*(1) time to compute.
- Use *b*[*i*, *j*] to reconstruct the elements of an LCS, the path is shaded or linked. （使用b表来重构 an LCS 的元素，路径用阴影或连线标注）



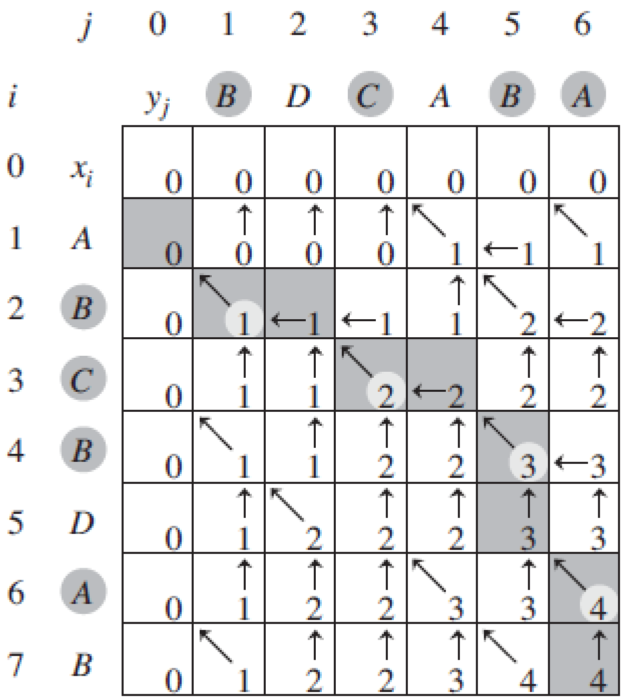
	j	0	1	2	3	4	5	6
i		y_j	*B*	*D*	*C*	*A*	*B*	*A*
0	x_i	0	0	0	0	0	0	0
1	*A*	0	↑0	↑0	↑0	↖1	←1	↖1
2	*B*	0	↖1	←1	←1	↑1	↖2	←2
3	*C*	0	↑1	↑1	↖2	←2	↑2	↑2
4	*B*	0	↖1	↑1	↑2	↑2	↖3	←3
5	*D*	0	↑1	↖2	↑2	↑2	↑3	↑3
6	*A*	0	↑1	↑2	↑2	↖3	↑3	↖4
7	*B*	0	↖1	↑2	↑2	↑3	↖4	↑4



		a	m	p	u	t	a	t	i	o	n
	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	1	1	1	1	1	1	1	1
a	0	1	1	1	1	1	2	2	2	2	2
n	0	1	1	1	1	1	2	2	2	2	3
k	0	1	1	1	1	1	2	2	2	2	3
i	0	1	1	1	1	1	2	2	3	3	3
n	0	1	1	1	1	1	2	2	3	3	4
g	0	1	1	1	1	1	2	2	3	3	4
				p			a		i		n

Step 4: Constructing an LCS

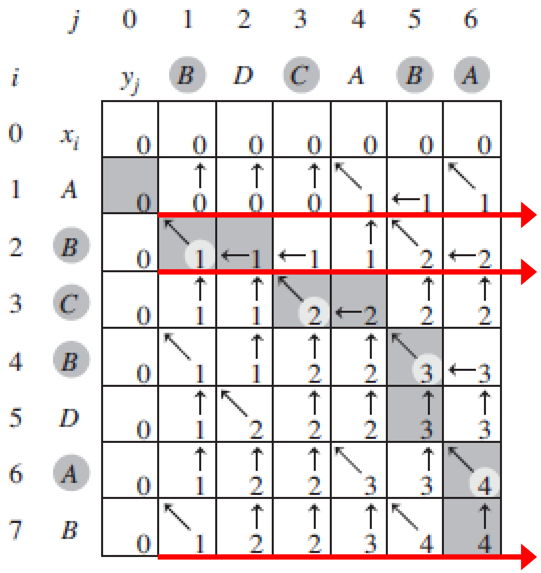
- **Initial call is PRINT-LCS(*b*, *X*, *m*, *n*).**
- **Whenever we encounter a “↖" in entry *b*[*i*, *j*], it implies that x_i=y_j is an element of the LCS.**
- **Procedure takes time *O*(*m*+*n*), since at least one of *i* and *j* is decremented in each stage of the recursion.**（每次递归时，*i* 和 *j* 至少有一个减值，算法的运行时间为*O*(*m*+*n*) ）



	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	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

```
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 $x_i$
else if b[i, j] = "↑"
    PRINT-LCS(b, X, i-1, j)
else PRINT-LCS(b, X, i, j-1)
```

Improving the code



	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	↖1	↑2	↑2	↑3	↖4	↑4

- Given an algorithm, you can improve on the time or space it uses.
- Some changes can simplify the code and improve constant factors but yield no asymptotic improvement in performance.
  （能简化代码，改进常数，但不能改进渐近性能）
- For example, we can eliminate the *b* table when constructing an LCS. Each *c*[*i*, *j*] entry depends on only: *c*[*i*-1, *j*-1], *c*[*i*-1, *j*], and *c*[*i*, *j*-1]. Given the value of *c*[*i*, *j*], we can determine in *O*(1) time which of these three values was used to compute *c*[*i*, *j*], without inspecting table *b*. (Exercise 15.4-2) （例如，重构an LCS 可以仅使用表 *c* , 而不用表 *b* ……）
- Save Θ(*mn*) space by this method, the space requirement does not asymptotically decrease, since we need Θ(*mn*) space for the *c* table anyway. （不使用表 *b* 可以节省Θ(*mn*)的空间开销，但算法的空间开销不会渐近减少，因为表 *c* 需要Θ(*mn*)的存储空间）

Improving the code

- Others can yield substantial asymptotic savings in time and space.

  （一些算法能产生实质性的、在时间和空间上的渐近性能的提高）
- Reduce the asymptotic space requirements for LCS-LENGTH, since it needs only two rows of table *c* at a time: the row being computed and the previous row. (In fact, we can use only slightly more than the space for one row of *c* to compute the length of an LCS. Exercise 15.4-4.)

  （可以减少LCS-LENGTH的空间的渐近开销。因为每次计算*c*[*i*,*j*]时仅需要表 *c* 的两行，正在计算的行和上一行。事实上，甚至可以仅使用比表 *c* 的一行稍多一点的空间来计算 *c* ）
- This improvement works if we need only the length of an LCS; if we need to reconstruct the elements of an LCS, the smaller table does not keep enough information.

  （如果我们仅需要求 an LCS 的长度时（最优值），上述改进的算法有效。若需要重构 an LCS 的每个元素（最优解），上述改进算法不能保留足够的信息）

Exercises

Problem 15-8
Image compression by seam carving

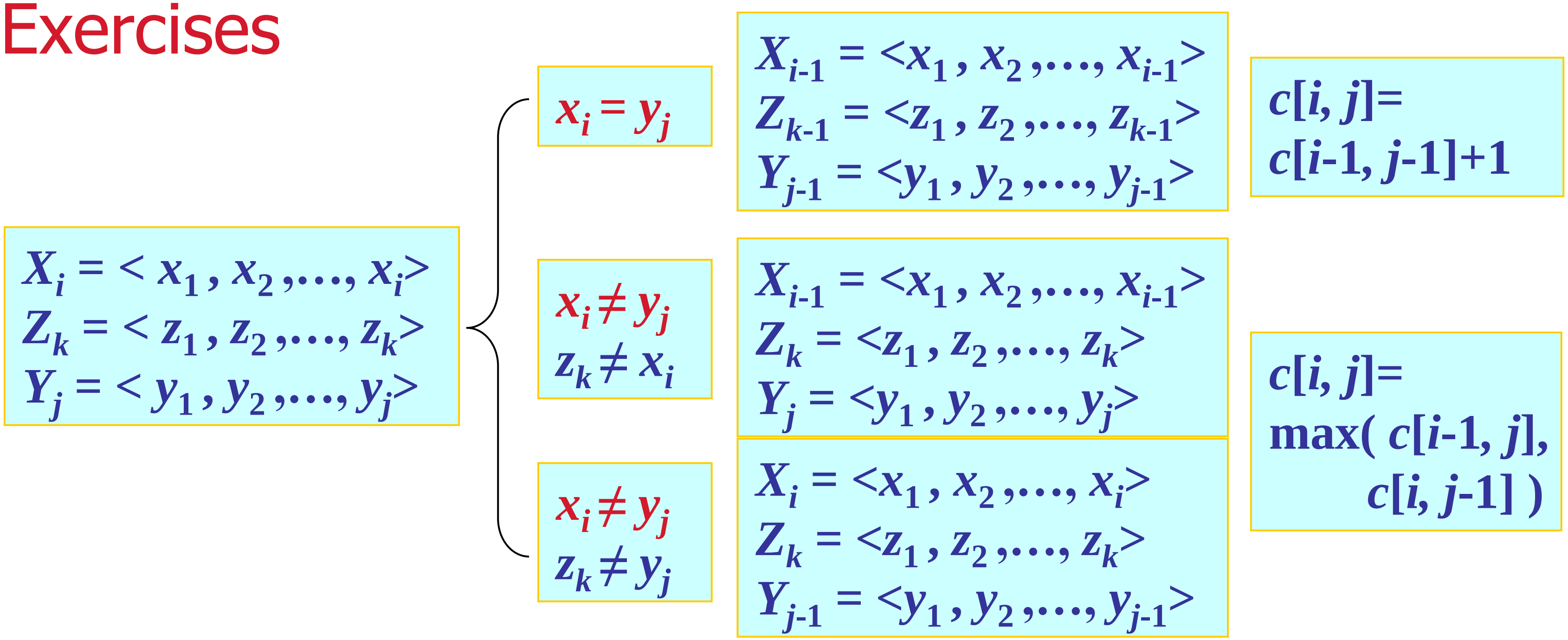
Seam carving for content-aware image resizing

S Avidan, A Shamir - ACM Transactions on graphics (TOG), 2007 - dl.acm.org

Abstract Effective resizing of images should not only use geometric constraints, but consider the image content as well. We present a simple image operator called seam carving that supports content-aware image resizing for both reduction and expansion. A seam is an …

☆ ❞ 被引用次数：1663 相关文章 所有 41 个版本

Exercises



$X_i = < x_1 , x_2 ,\ldots, x_i>$
$Z_k = < z_1 , z_2 ,\ldots, z_k>$
$Y_j = < y_1 , y_2 ,\ldots, y_j>$

$x_i = y_j$

$X_{i-1} = <x_1 , x_2 ,\ldots, x_{i-1}>$
$Z_{k-1} = <z_1 , z_2 ,\ldots, z_{k-1}>$
$Y_{j-1} = <y_1 , y_2 ,\ldots, y_{j-1}>$

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

$x_i \neq y_j$
$z_k \neq x_i$

$X_{i-1} = <x_1 , x_2 ,\ldots, x_{i-1}>$
$Z_k = <z_1 , z_2 ,\ldots, z_k>$
$Y_j = <y_1 , y_2 ,\ldots, y_j>$

$x_i \neq y_j$
$z_k \neq y_j$

$X_i = <x_1 , x_2 ,\ldots, x_i>$
$Z_k = <z_1 , z_2 ,\ldots, z_k>$
$Y_{j-1} = <y_1 , y_2 ,\ldots, y_{j-1}>$

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

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ c[i-1, j-1]+1, & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max(c[i-1, j], c[i, j-1]) , & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases} \qquad (15.14)$$

Based on equation (15.14) from the LCS, what is the

Recursive algorithm? Running time?

Exercises

15.4-2

仅使用表 *c*，如何重构 an LCS？(similar to PRINT-LCS)

15.4-5

对一个数序列，给出一个$O(n^2)$算法，求该序列的最长单调增子序列。

Sort in increasing order first, then find an LCS.

15.4-1

Determine an LCS of <1,0,0,1,0,1,0,1,0,0,1,1,0> and <0,1,0,1,1,0,1,1,0,1,0,1,0,1,1>. Implement by programming.

15.4-4

Implement by programming

Analyse the running time of the algorithm

15 Dynamic Programming

- **Why is DP effective ?**
- **When does DP apply ?**
- **Why do Time and Space conflict ?**