# 算法分析与设计

## Analysis and Design of Algorithm

第8次课

# 学习要点

- 理解动态规划算法的概念
- 掌握动态规划算法的基本要素
  - 最优子结构性质
  - 重叠子问题性质
- 掌握设计动态规划算法的步骤。
  - 找出最优解的性质，并刻划其结构特征
  - 递归地定义最优值
  - 以自底向上的方式计算出最优值
  - 根据计算最优值时得到的信息，构造最优解

# 要点回顾

- 动态规划法的实例
  - 最短路径问题
  - 矩阵连乘问题

# 矩阵连乘问题动态规划法

## 优化函数的递推方程：

**$m[i,j]$：得到$A[i{:}j]$的最少的相乘次数。可递归定义为：**

$$m[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+P_{i-1}P_kP_j\} & i<j\end{cases}$$

$(A_iA_{i+1}\cdots A_k)$ $(A_{k+1}A_{k+2}\cdots A_j)$

$P_{i-1}\times P_k$ $\quad$ $P_k\times P_j$

**该问题满足优化原则！**

# 矩阵连乘问题最优子结构证明

令最优解$m[1, n]=m[1, k]+m[k+1, n]+P_0P_kP_n$，即矩阵链在$k$的位置断开。证明$m[1, k]$和$m[k+1, n]$分别是子问题$A[1：k]$和$A[k+1：n]$的最优解。

$$(A_1A_{i+1}\cdots A_k)(A_{k+1}A_{k+2}\cdots A_n)$$

**反证法：** 假设$m[1, k]$不是$A[1：k]$的最优解，则$A[1：k]$存在一个最优解$m'[1, k]<m[1, k]$。此时，可以得到原问题$A[1：n]$的一个解

$$m'[1, n]=m'[1, k]+m[k+1, n]+P_0P_kP_n < m[1, n]$$

与假设矛盾！

# 矩阵连乘的实例

- **输入：** $P=<30, 35, 15, 5, 10, 20>$

  $n=5$

- **矩阵链：** $A_1 A_2 A_3 A_4 A_5$，其中

  $A_1:30\times35,\ A_2:35\times15,\ A_3:15\times5,$

  $A_4:5\times10,\ A_5:10\times20$

- **中间结果：** 存储所有子问题的最小乘法次数$m[i, j]$及得到这个值的划分位置$s[i, j]$

# 子问题最优解 $m[i, j]$

- $P$=<30, 35, 15, 5, 10, 20>

| | | | | | |
|---|---|---|---|---|---|
| $r$=1 | m[1,1]=0 | m[2,2]=0 | m[3,3]=0 | m[4,4]=0 | m[5,5]=0 |
| $r$=2 | | | | | |
| $r$=3 | | | | | |
| $r$=4 | | | | | |
| $r$=5 | | | | | |

举例：如何计算$m[2, 5]=\min\begin{Bmatrix} m[2,2]+m[3,5]+P_1P_2P_5 \\ m[2,3]+m[4,5]+P_1P_3P_5 \\ m[2,4]+m[5,5]+P_1P_4P_5 \end{Bmatrix}$

# 子问题最优解 $m[i, j]$

- $P$=<30, 35, 15, 5, 10, 20>

| | | | | | |
|---|---|---|---|---|---|
| $r$=1 | m[1,1]=0 | m[2,2]=0 | m[3,3]=0 | m[4,4]=0 | m[5,5]=0 |
| $r$=2 | m[1,2]=15750 | m[2,3]=2625 | m[3,4]=750 | m[4,5]=1000 | |
| $r$=3 | m[1,3]=7875 | m[2,4]=4375 | m[3,5]=2500 | | |
| $r$=4 | m[1,4]=9375 | m[2,5]=7125 | | | |
| $r$=5 | m[1,5]=11875 | | | | |

1. $m[2,5]= m[2,2]+ m[3,5]+ P_1P_2P_5 =$

$0+2500+35\times15\times20=13000$

# 子问题最优解 $m[i, j]$

- $P$=<30, 35, 15, 5, 10, 20>

| | | | | | |
|---|---|---|---|---|---|
| $r$=1 | m[1,1]=0 | m[2,2]=0 | m[3,3]=0 | m[4,4]=0 | m[5,5]=0 |
| $r$=2 | m[1,2]=15750 | m[2,3]=2625 | m[3,4]=750 | m[4,5]=1000 | |
| $r$=3 | m[1,3]=7875 | m[2,4]=4375 | m[3,5]=2500 | | |
| $r$=4 | m[1,4]=9375 | m[2,5]=7125 | | | |
| $r$=5 | m[1,5]=11875 | | | | |

2. $m[2,5]= m[2,3]+ m[4,5]+ P_1P_3P_5 =$

$2625+1000+35\times5\times20=7125$

# 子问题最优解 $m[i, j]$

- $P$=<30, 35, 15, 5, 10, 20>

| | | | | | |
|---|---|---|---|---|---|
| $r$=1 | m[1,1]=0 | m[2,2]=0 | m[3,3]=0 | m[4,4]=0 | m[5,5]=0 |
| $r$=2 | m[1,2]=15750 | m[2,3]=2625 | m[3,4]=750 | m[4,5]=1000 | |
| $r$=3 | m[1,3]=7875 | m[2,4]=4375 | m[3,5]=2500 | | |
| $r$=4 | m[1,4]=9375 | m[2,5]=7125 | | | |
| $r$=5 | m[1,5]=11875 | | | | |

3. $m[2,5]= m[2,4]+ m[5,5]+ P_1P_4P_5 =$

$4375+0+35\times10\times20=11375$

# 子问题最优解 $m[i, j]$

- $P$=<30, 35, 15, 5, 10, 20>

| | | | | | |
|---|---|---|---|---|---|
| $r$=1 | m[1,1]=0 | m[2,2]=0 | m[3,3]=0 | m[4,4]=0 | m[5,5]=0 |
| $r$=2 | m[1,2]=15750 | m[2,3]=2625 | m[3,4]=750 | m[4,5]=1000 | |
| $r$=3 | m[1,3]=7875 | m[2,4]=4375 | m[3,5]=2500 | | |
| $r$=4 | m[1,4]=9375 | m[2,5]=7125 | | | |
| $r$=5 | m[1,5]=11875 | | | | |

$m$[2,5]=min{13000, 7125, 11375}=7125

# 标记函数 $s[i, j]$

| | | | | |
|---|---|---|---|---|
| $r$=2 | s[1,2]=1 | s[2,3]=2 | s[3,4]=3 | s[4,5]=4 |
| $r$=3 | s[1,3]=1 | s[2,4]=3 | s[3,5]=3 | |
| $r$=4 | s[1,4]=3 | s[2,5]=3 | | |
| $r$=5 | s[1,5]=3 | | | |

解的追踪：s[1,5]=3➔$(A_1 A_2 A_3)(A_4 A_5)$

s[1,3]=1➔$A_1(A_2 A_3)$

输出

计算顺序为：$(A_1(A_2 A_3))(A_4 A_5)$

最少的乘法次数：m[1,5]=11875

# 小结：动态规划的基本要素

## 一、最优子结构

- 矩阵连乘计算次序问题的最优解包含着其子问题的最优解。这种性质称为最优子结构性质。
- 在分析问题的最优子结构性质时，通常可采用反证法。
- 利用问题的最优子结构性质，以自底向上的方式递归地从子问题的最优解逐步构造出整个问题的最优解。最优子结构是问题能用动态规划算法求解的前提。

**同一个问题可以有多种方式刻划它的最优子结构，有些表示方法的求解速度更快(空间占用小，问题的维度低)。**

# 小结：动态规划的基本要素

## 二、重叠子问题

- 递归算法求解问题时，每次产生的子问题并不总是新问题，有些子问题被反复计算多次。这种性质称为子问题的重叠性质。
- 动态规划算法，对每一个子问题只解一次，而后将其解保存在一个表格中，当再次需要解此子问题时，只是简单地用常数时间查看一下结果。
- 通常不同的子问题个数随问题的大小呈多项式增长。因此用动态规划算法只需要多项式时间，从而获得较高的解题效率。

# 基于备忘录的递归方法

# 回顾：基于递归的算法

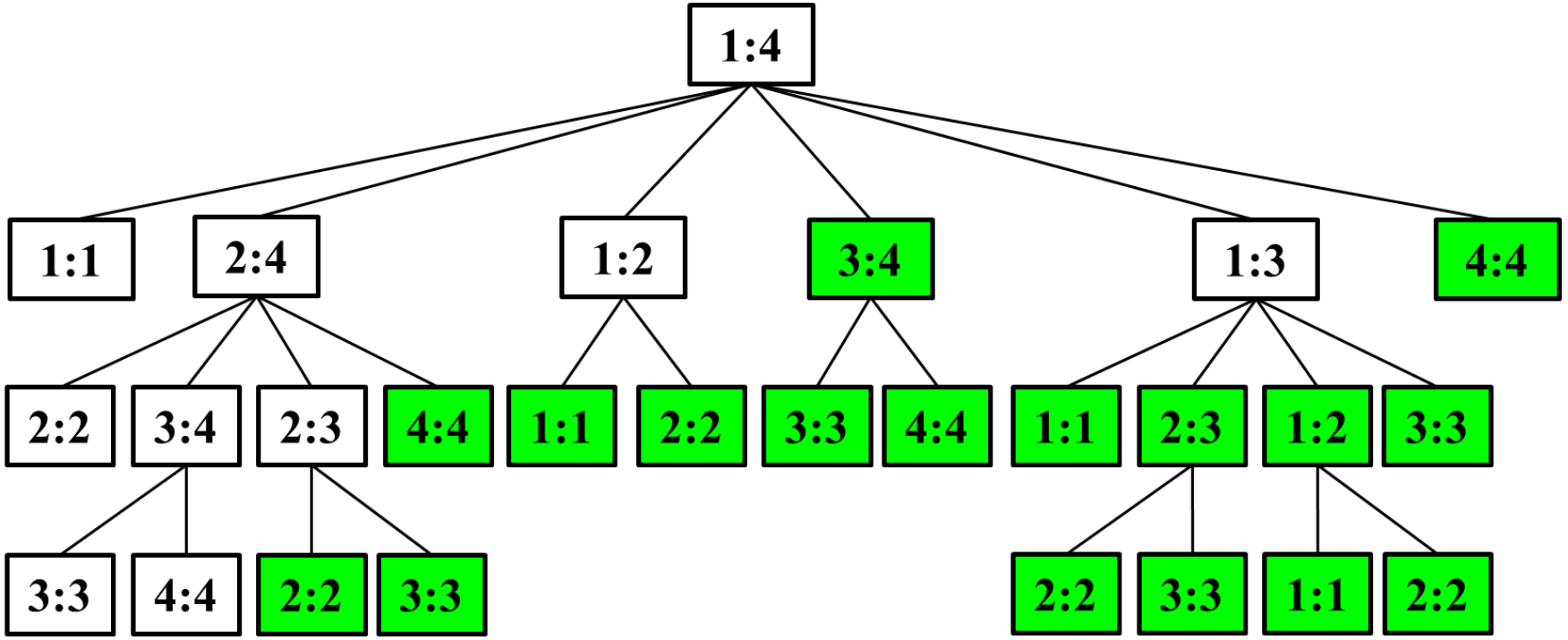
- **RecurMatrixChain(*P*, *i*, *j*)**

```
1.  m[i,j]←∞
2.  s[i, j]←i
3.  for k←i to j-1 do
4.      q←RecurMatrixChain(P, i, k)
           + RecurMatrixChain(P, k+1, j) + P_{i-1}P_kP_j
5.      if q < m[i,j]
6.      then m[i,j]←q
7.           s[i,j]←k
8.  return m[i,j]
```

**递归算法的时间复杂度为$\Omega(2^n)$。如何改进？**

# 动态规划算法的变形-备忘录法

用上述算法RecurMatrixChain(1,4)计算A[1:4]的递归树：



# 动态规划算法的变形-备忘录法

## 备忘录方法

- 该方法的控制结构与直接递归方法的控制结构相同，区别在于备忘录方法为每个解过的子问题建立了备忘录以备需要时查看，避免了相同子问题的重复求解。

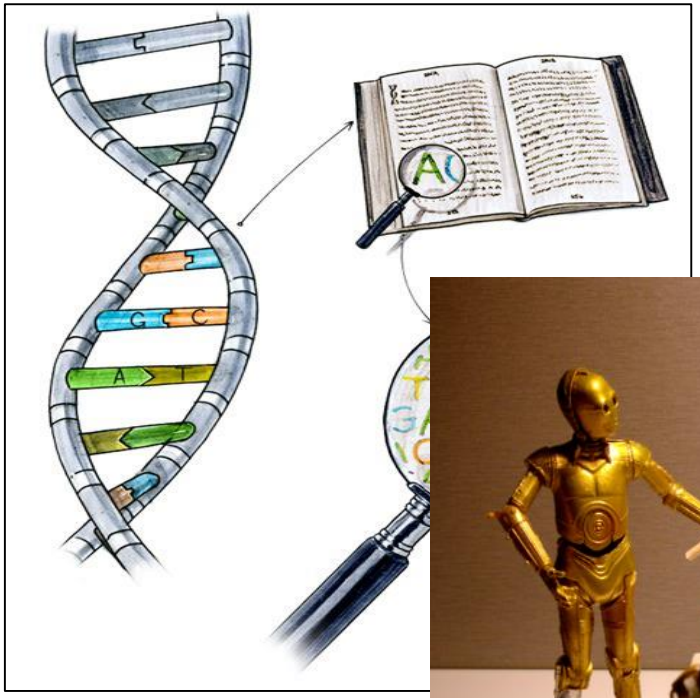
```
int LookupChain(int i，int j)
{
    if (m[i][j] > 0) return m[i][j];
    if (i == j) return 0;
    int u = LookupChain(i，i) + LookupChain(i+1，j) + p[i-1]*p[i]*p[j];
    s[i][j] = i;
    for (int k = i+1; k < j; k++) {
      int t = LookupChain(i，k) + LookupChain(k+1，j) + p[i-1]*p[k]*p[j];
      if (t < u) { u = t; s[i][j] = k;}
      }
    m[i][j] = u;
    return u;
}
```

# 动态规划法和备忘录法的异同

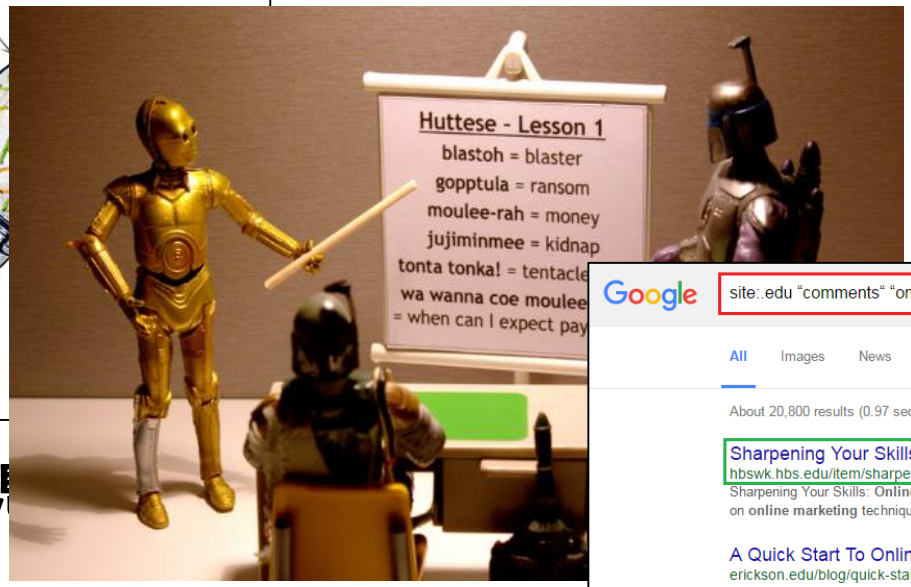
- 相同点：
  - 最优子结构
  - 重叠子问题

- 不同点
  - 动态规划：自底向上（从最小的问题开始解，不断填充子问题解的矩阵）
  - 备忘录：自顶向下（从最大的问题开始解，并查看子问题解的矩阵，若矩阵中有值，则无需额外计算）

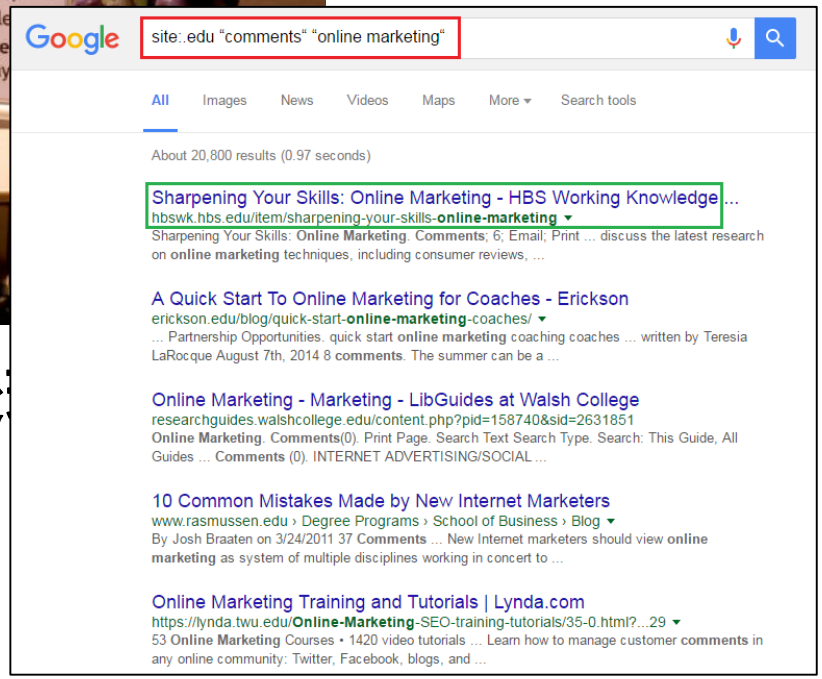
# 序列/字符串相关问题



生物信息




自然语言处




搜索引擎

# 理综（生物部分）试题

下列关于真核细胞中转录的叙述，错误的是

A．tRNA、rRNA和mRNA都从DNA转录而来

B．同一细胞中两种RNA和合成有可能同时发生

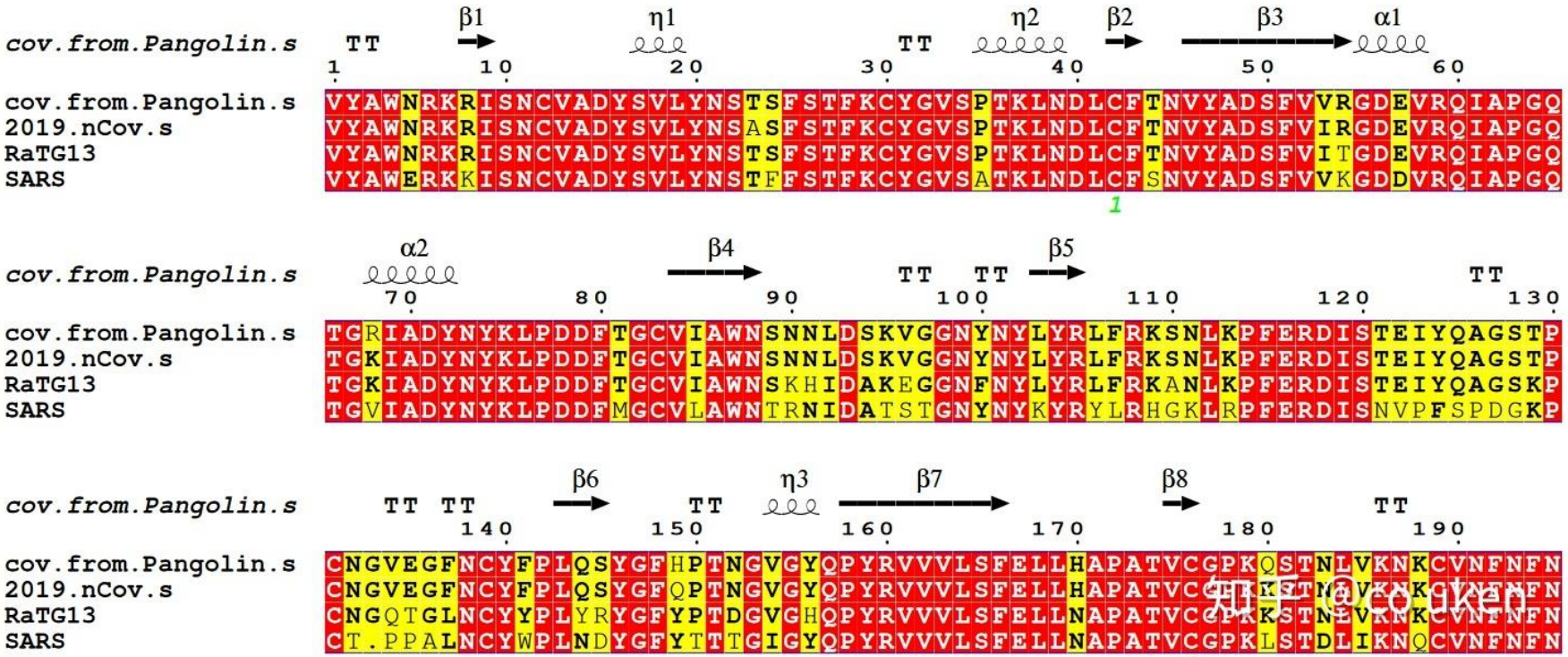
C．细胞中的RNA合成过程不会在细胞核外发生

D．转录出的RNA链与模板链的相应区域碱基互补

# 新冠病毒的基因序列比对

ORIGIN

```
        1 attaaaggtt tataccttcc caggtaacaa accaaccaac tttcgatctc ttgtagatct
       61 gttctctaaa cgaaatttaa aatctgtgtg gctgtcactc ggctgcatg  ttaaaact
      121 cacgcagtat aattaataac taattactgt cgttgacagg acacgagtaa ctcgtctatc
```



# 最长公共子序列

**子序列：** 若给定序列$X=\{x_1,x_2,\ldots,x_m\}$，则另一序列$Z=\{z_1,z_2,\ldots,z_k\}$，是$X$的子序列是指：存在一个**严格递增**下标序列$\{i_1,i_2,\ldots,i_k\}$使得对于所有$j=1,2,\ldots,k$有：$z_j=x_{i_j}$。

**例：** 序列$Z=\{B,C,D,B\}$是$X=\{A,B,C,B,D,A,B\}$的子序列，相应的递增下标序列为$\{2,3,5,7\}$。

给定两个序列$X$和$Y$，当另一序列$Z$既是$X$的子序列又是$Y$的子序列时，称$Z$是序列$X$和$Y$的**公共子序列**。

中国 MOOC 5-8 最长公共子序列

# 最长公共子序列

- **问题：** 给定两个序列$X=\{x_1,x_2,\ldots,x_m\}$和$Y=\{y_1,y_2,\ldots,y_n\}$，找出$X$和$Y$的最长公共子序列(Longest Common Subsequence, LCS)
- **实例：**

  X: A B C B D A B

  Y: B D C A B A

  最长公共子序列：B C B A，长度4

➔不是唯一的，长度相等情况下，可能有多个不同公共子序列，只需给出一个即可

# 穷举法

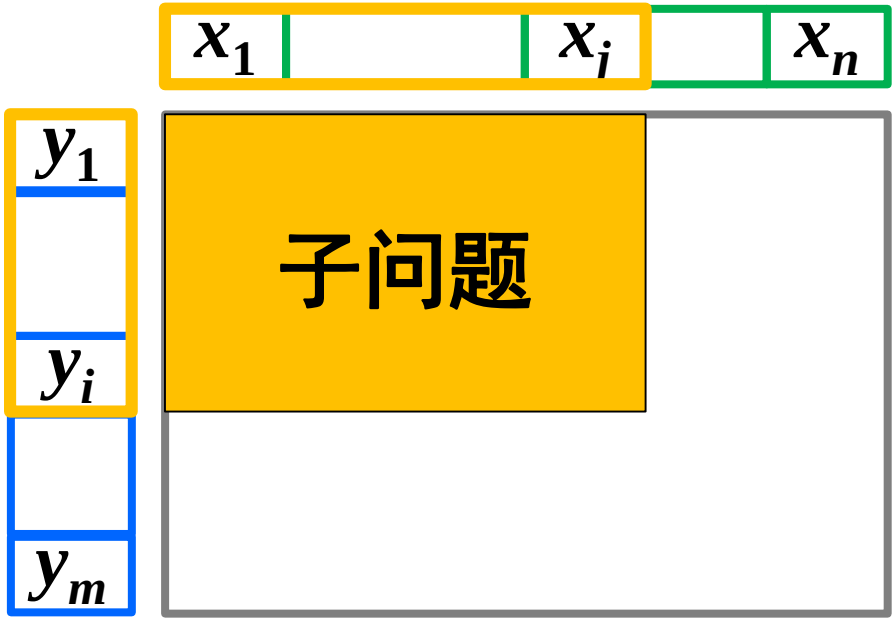
- 不妨设$m \leqslant n$,$|X|=m$, $|Y|=n$
- 算法：依次检查X的每个子序列在Y中是否出现

- **时间复杂度：**
  - X有$2^m$个子序列
  - 每个子序列$O(n)$时间

**最坏情况下时间复杂度：$O(n2^m)$**

# 子问题界定

- 参数 $i$ 和 $j$ 界定子问题
- X的终止位置是 $i$，Y的终止位置是 $j$
- $X_i=\{x_1,x_2,\ldots,x_i\}$ ， $Y_j=\{y_1,y_2,\ldots,y_j\}$

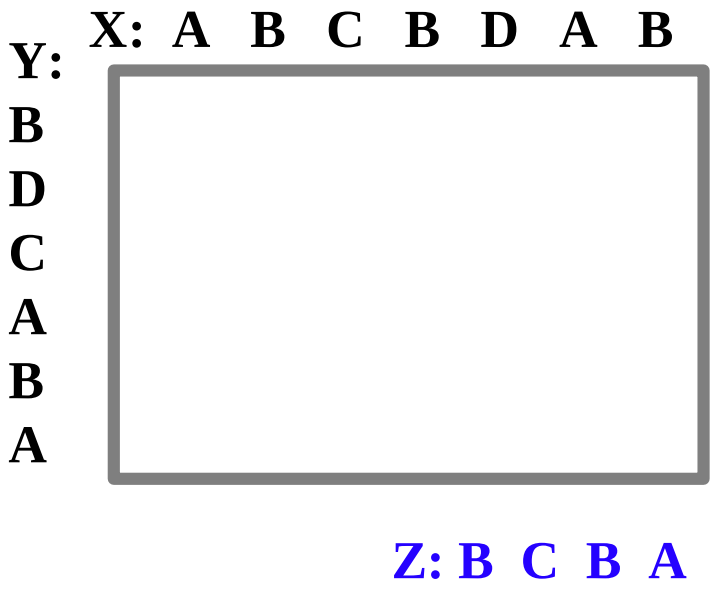



X: A B C B D A B

Y: B D C A B A

# 子问题间的依赖关系

- 假设两个序列X={$x_1,x_2,\ldots,x_m$}，Y={$y_1,y_2,\ldots,y_n$}，Z={$z_1,z_2,\ldots,z_k$}为X和Y的LCS，那么
  - 若$x_m = y_n$➔ $z_k = x_m = y_n$，且$Z_{k-1}$是$X_{m-1}$与$Y_{n-1}$的LCS
  - 若$x_m \neq y_n$，$z_k \neq x_m$ ➔ Z是$X_{m-1}$与Y的LCS
  - 若$x_m \neq y_n$，$z_k \neq y_n$ ➔ Z是X与$Y_{n-1}$的LCS




满足最优子结构性质和子问题重叠性

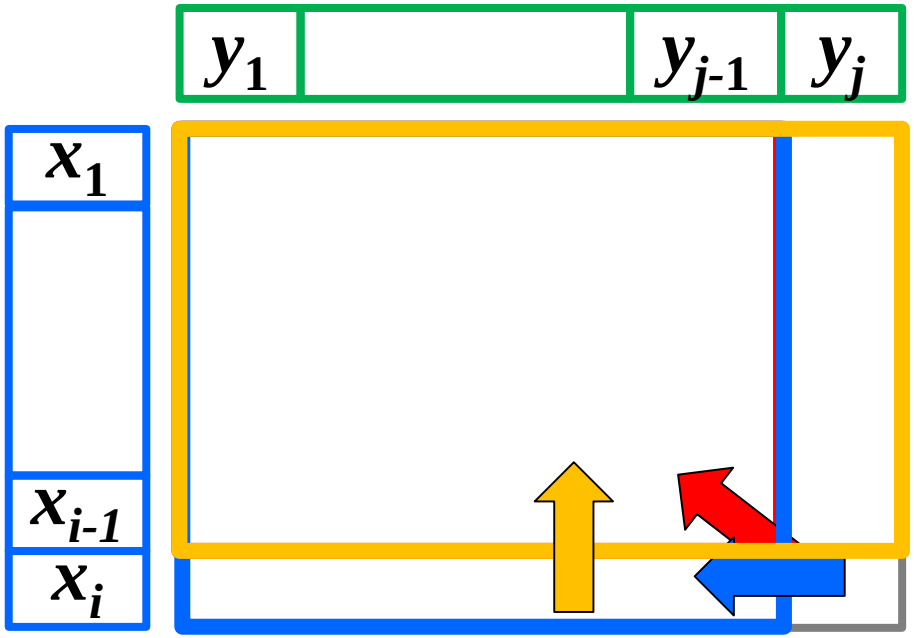
# 优化函数的递推方程

- 令X和Y的子序列
- $X_i=\{x_1,x_2,\cdots,x_i\}$，$Y_j=\{y_1,y_2,\cdots,y_j\}$
- $C[i,j]$：$X_i$与$Y_j$的LCS的长度

$$C[i,j]=\begin{cases}0 & \text{若}i=0\text{或}j=0\\ C[i-1,j-1]+1 & \text{若}i,j>0,\ x_i=y_j\\ \max\{C[i,j-1],C[i-1,j]\} & \text{若}i,j>0,\ x_i\neq y_j\end{cases}$$

# 标记函数

- 标记函数： $B[i,j]$，值为↖、←、↑
- $C[i,j]= C[i-1,j-1]+1$： ↖
- $C[i,j]= C[i,j-1]$： ←
- $C[i,j]= C[i-1,j]$： ↑



# 算法的伪码

- 算法 LCS(X, Y, *m*,*n*)

```
1.  for i←1 to m do  C[i,0] ← 0
2.  for i ← 1 to n do  C[0,i] ← 0
3.  for i ← 1 to m do
4.      for j ← 1 to n do
5.          if X[i]=Y[j]
6.          then  C[i,j] ← C[i-1,j-1]+1
7.                B[i,j] ←"↖"
8.          else if C[i-1,j]≥C[i,j-1]
9.               then C[i,j] ← C[i-1,j]
10.                   B[i,j] ←"↑"
11.              else C[i,j] ← C[i,j-1]
12.                   B[i,j] ←"←"
```

初值

子问题

# 追踪解

- 算法StructureSequence(B,*i*,*j*)
  - 输入：B[*i*,*j*]
  - 输出：X与Y的最长公共子序列

```
1.  if i=0 or j=0 then return  //序列为空
2.  if B[i,j]="↖"
3.  then  输出X[i]
4.        StructureSequence(B,i-1,j-1)
5.  else if  B[i,j]="↑"
6.        then StructureSequence(B,i-1,j)
7.        else StructureSequence(B,i,j-1)
```

# 标记函数的实例

- 输入：X=〈 A, B, C, B, D, A, B 〉
  Y=〈 B, D, C, A, B, A 〉

| X \ Y | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | B[1,1]=↑ | B[1,2]=↑ | B[1,3]=↑ | B[1,4]=↖ | B[1,5]=← | B[1,6]=↖ |
| 2 | B[2,1]=↖ | B[2,2]=← | B[2,3]=← | B[2,4]=↑ | B[2,5]=↖ | B[2,6]=← |
| 3 | B[3,1]=↑ | B[3,2]=↑ | B[3,3]=↖ | B[3,4]=← | B[3,5]=↑ | B[3,6]=↑ |
| 4 | B[4,1]=↑ | B[4,2]=↑ | B[4,3]=↑ | B[4,4]=↑ | B[4,5]=↖ | B[4,6]=← |
| 5 | B[5,1]=↑ | B[5,2]=↑ | B[5,3]=↑ | B[5,4]=↑ | B[5,5]=↑ | B[5,6]=↑ |
| 6 | B[6,1]=↑ | B[6,2]=↑ | B[6,3]=↑ | B[6,4]=↖ | B[6,5]=↑ | B[6,6]=↖ |
| 7 | B[7,1]=↑ | B[7,2]=↑ | B[7,3]=↑ | B[7,4]=↑ | B[7,5]=↖ | B[7,6]=↑ |

解：X[2], X[3], X[4], X[6], 即B, C, B, A

# 算法的时空复杂度

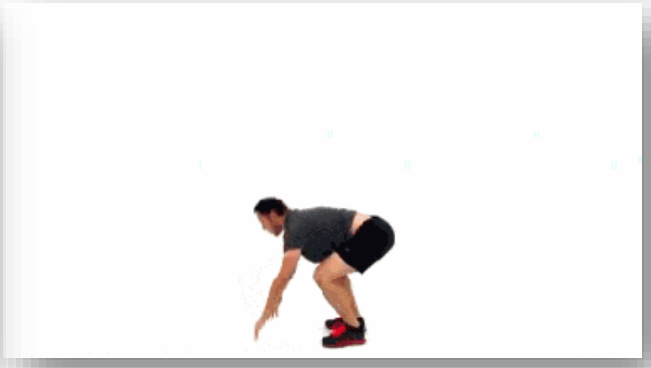
- 计算优化函数和标记函数
  - 赋初值，为$O(m)+O(n)$
  - 计算优化、标记函数迭代次$\Theta(mn)$
  - 循环体内常数次运算，时间为$\Theta(mn)$
- 构造解
  - 每步缩小X或Y的长度，时间$\Theta(m+n)$

**算法时间复杂度：$\Theta(mn)$**

**空间复杂度：$\Theta(mn)$**

# 序列/字符串相关问题

# 金老师的健身之路




**哪一段时间体重减得最多？**

# 最大子段和

**问题：** $n$个整数序列$a_1 \ldots a_n$求该序列形如 $\sum_{k=i}^{j} a_k$ 的子段和的最大值。当所有整数均为负整数时定义其最大子段和为0 。

- 所以 $\max\left\{0, \max_{1\le i\le j\le n}\sum_{k=i}^{j} a_k\right\}$
- 例：当$(a_1, a_2, \ldots, a_6)$= (-2,11 -4,13,-5,-2)

此最大子段和为：$\sum_{k=2}^{4} a_k = 20$

中国 MOOC 6-3 最大子段和

# 最大子段和简单算法

用数组$a$[]存储$n$个整数$a_1 \ldots a_n$。

```
void MaxSum(int n，int *a，int& besti，int& bestj)
{
    int sum=0;
    for (int i = 1; i <= n; i++)
      for (int j = i; j <= n; j++){
        int thissum=0;
        for (int k = i; k<=j; k++) thissum+=a[k]; //i→j
        if (thissum>sum) { //记录i，j
            sum=thissum;
            besti=i;
            bestj=j;
        }
    }
    return sum;
}
```

可进行改进

显然计算时间是$O(n^3)$

# 最大子段和简单算法的改进

```
void MaxSum(int n，int *a，int& besti，int& bestj)
{
      int sum=0;
      for (int i = 1; i <= n; i++){
        int thissum=0;
        for (int j = i; j <= n; j++){
            thissum+=a[j];                    //i→n的和
            if (thissum>sum) {
                sum=thissum;
                besti=i;
                bestj=j;
            }
        }
      }
      return sum;
}
```

从算法设计技巧上的改进，
计算时间是$O(n^2)$

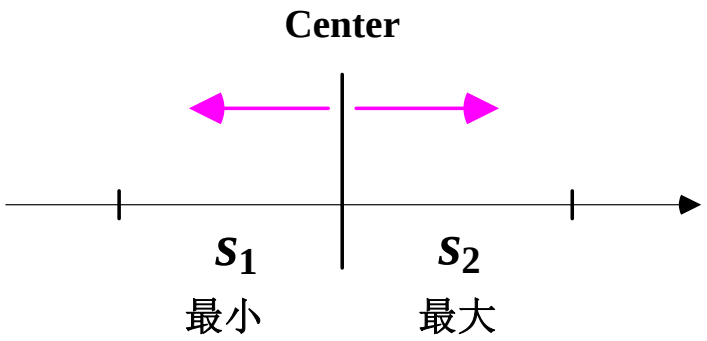
# 最大子段和的分治算法

- 如果将所给的序列$a[1..n]$分为长度相等的两段$a[1:n/2]$和$a[n/2+1:n]$,分别求出这两段最大子段和，则$a[1..n]$的最大子段和有三种情形：
  - *1*、$a[1:n]$的最大子段和与$a[1:n/2]$的最大子段和相同；
  - *2*、$a[1:n]$的最大子段和与$a[n/2+1:n]$的最大子段和相同；
  - *3*、$a[1:n]$的最大子段和为$\sum_{k=i}^{j} a_k$，且$1 \le i \le n/2$, $n/2+1 \le j \le n$

# 最大子段和的分治算法

- 对于1和2两种情况可以递归求解。
- 对于情形3， $a[n/2]$与$a[n/2+1]$在最优子序列中。
  - 可以在$a[1:n/2]$中计算出： $s_1 = \max\limits_{1\le i\le n/2}\sum\limits_{k=i}^{n/2} a[k]$
  - 可以在$a[n/2+1:n]$中计算出：$s_2 = \max\limits_{n/2+1\le i\le n}\sum\limits_{k=\frac{n}{2}+1}^{i} a[k]$

$s_1+s_2$即为最优

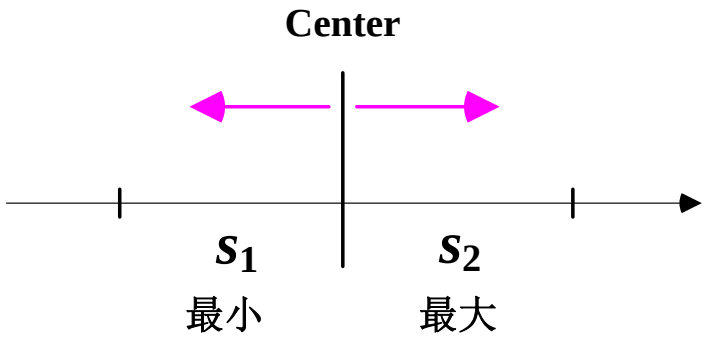


# 最大子段分治算法

## 复杂度分析

**算法所需的计算时间T(*n*)满足典型的分治算法递归式：**

$$T(n)=\begin{cases} O(1) & n \le c \\ 2T(n/2)+O(n) & n > c \end{cases}$$




➔ $T(n)=O(n\log n)$

# 最大子段和动态规划算法

- 令$b[j]$为a[1]，a[2]，…，a[$j$]的最大子段和（该子段的最后一个元素必须为a[$j$]），则

$$b[j]=\max_{1\le i\le j}\left\{\sum_{k=i}^{j}a[k]\right\}\quad 1\le j\le n$$

  - 所以，整个序列的最大子段和为最大的$b[j]$

$$\max_{1\le i\le j\le n}\sum_{k=i}^{j}a[k]=\max_{1\le j\le n}\max_{1\le i\le j}\sum_{k=i}^{j}a[k]=\max_{1\le j\le n}b[j]$$

# 最大子段和动态规划算法

- 由$b[j]$定义：
  - 当$b[j\text{-}1]>0$, $b[j]= b[j\text{-}1]+ a[j]$
  - 否则 , $b[j]= a[j]$

$$\left.\begin{array}{l}\text{当}b[j\text{-}1]>0,\ b[j]=b[j\text{-}1]+a[j]\\ \text{否则}\quad ,\ b[j]=a[j]\end{array}\right\}\Rightarrow b[j]=\max_{1\le j\le n}\{b[j-1]+a[j],a[j]\}$$

据此，可设计出求最大字段和的动态规划算法

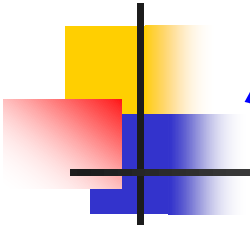
# 最大子段和动态规划算法

```
int MaxSum(int n, int *a)
{
    int sum=0, b=0;
    for (int i = 1; i <= n; i++){
        if (b>0)  b+=a[i];
        else  b=a[i];
        if (b>sum) sum=b;
    }
    return sum;
}
```

**算法复杂度$O(n)$**

# 最大子段和

- 最大字段和问题有不同的解法
- 时间复杂度各不相同
  - 最直观的做法 $O(n^3)$
  - 改进的做法$O(n^2)$
  - 分治法$O(n\log n)$
  - 动态规划法$O(n)$



# 小结

- 矩阵连乘问题
  - 最优子结构、递推方程
  - 自底向上的计算过程、构造解的过程
- 最长公共子序列
  - 最优子结构、递推方程
  - 自底向上的计算过程、构造解的过程
- 最大子段和
  - 如何依次改进算法