

# 本周教学内容

# 动态规划算法的 重要应用

图像压缩

最大子段和

最优二叉  
检索树  
算法

最优二叉  
检索树  
概念

RNA  
二级  
结构  
预测

序列  
比对

# 图像压缩

# 黑白图像存储

像素点灰度值：0~255，为8位二进制数

图像的灰度值序列： $\{p_1, p_2, \dots, p_n\}$ ， $p_i$ 为第 $i$ 个像素点灰度值

图像存储：每个像素的灰度值占8位，  
总计空间为  $8n$



有没有更好的存储  
方法



# 图像变位压缩的概念

变位压缩存储：将 $\{p_1, p_2, \dots, p_n\}$ 分成  $m$  段

$$S_1, S_2, \dots, S_m$$



同一段的像素占用位数相同

第  $t$  段有  $l[t]$  个像素，每个占用  $b[t]$  位

段头：记录  $l[t]$  (8位) 和  $b[t]$  (3位) 需要 11 位

总位数为

$$b[1] \cdot l[1] + b[2] \cdot l[2] + \dots + b[m] \cdot l[m] + 11m$$

# 图像压缩问题

约束条件：第  $t$  段像素个数  $l[t] \leq 256$

第  $t$  段占用空间： $b[t] \times l[t] + 11$

$$b[t] = \left\lceil \log(\max_{p_k \in S_t} p_k + 1) \right\rceil \leq 8$$

问题：给定像素序列  $\{p_1, p_2, \dots, p_n\}$ ，  
确定最优分段，即

$$\min_T \left\{ \sum_{t=1}^m (b[t] \times l[t] + 11) \right\},$$

$T = \{S_1, S_2, \dots, S_m\}$  为分段

# 实例

灰度值序列

$P=\{10,12,15,255,1,2,1,1,2,2,1,1\}$

分法1:  $S_1=\{10, 12, 15\}$ ,  $S_2=\{255\}$ ,  
 $S_3=\{1, 2, 1, 1, 2, 2, 1, 1\}$

分法2:  $S_1=\{10,12,15,255,1,2,1,1,2,2,1,1\}$

分法3: 分成12组, 每组一个数

存储空间

分法1:  $11 \times 3 + 4 \times 3 + 8 \times 1 + 2 \times 8 = 69$

分法2:  $11 \times 1 + 8 \times 12 = 107$

分法3:  $11 \times 12 + 4 \times 3 + 8 \times 1 + 1 \times 5 + 2 \times 3 = 163$

# 子问题界定与计算顺序

子问题前边界为1，后边界为  $i$

对应像素序列为  $\langle p_1, p_2, \dots, p_i \rangle$

优化函数值  $S[i]$  为最优分段存储位数

计算顺序

$i = 1$  

$i = 2$  

...

$i = n$  



# 算法设计

递推方程：设 $S[i]$ 是 $\{p_1, p_2, \dots, p_i\}$ 的最优分段需要的存储位数， $S_m$ 是最后分段

$$S[i] = \min_{1 \leq j \leq \min\{i, 256\}} \{ S[i-j] + j \times b[i-j+1, i] \} + 11$$

$$b[i-j+1, i] = \left\lceil \log(\max_{p_k \in S_m} p_k + 1) \right\rceil \leq 8$$



$S[i-j]$ 个位

$j$  个灰度

$j \times b[i-j+1, i]$ 位

# 伪码

Compress ( $P, n$ )

1.  $Lmax \leftarrow 256; header \leftarrow 11; S[0] \leftarrow 0$

2. for  $i \leftarrow 1$  to  $n$  do

3.      $b[i] \leftarrow \text{length}(P[i])$

4.      $bmax \leftarrow b[i]$

5.      $S[i] \leftarrow S[i-1] + bmax$

6.      $l[i] \leftarrow 1$

7.     for  $j \leftarrow 2$  to  $\min\{i, Lmax\}$  do

8.         if  $bmax < b[i-j+1]$

9.         then  $bmax \leftarrow b[i-j+1]$

10.        if  $S[i] > S[i-j] + j * bmax$

11.        then  $S[i] \leftarrow S[i-j] + j * bmax$

12.         $l[i] \leftarrow j$

13.      $S[i] \leftarrow S[i] + header$

子问题后  
边界  $i$

最后  
段长  $j$

找到更  
好分段

$P = \langle 10, 12, 15, 255, 1, 2 \rangle$ .

$S[1]=15, S[2]=19, S[3]=23, S[4]=42, S[5]=50$

$l[1]=1, l[2]=2, l[3]=3, l[4]=1, l[5]=2$

10	12	15	255	1	2
----	----	----	-----	---	---

$S[5]=50$   $1 \times 2 + 11$  63

10	12	15	255	1	2
----	----	----	-----	---	---

$S[4]=42$   $2 \times 2 + 11$  57

10	12	15	255	1	2
----	----	----	-----	---	---

$S[3]=23$   $3 \times 8 + 11$  58

10	12	15	255	1	2
----	----	----	-----	---	---

$S[2]=19$   $4 \times 8 + 11$  62

10	12	15	255	1	2
----	----	----	-----	---	---

$S[1]=15$   $5 \times 8 + 11$  66

10	12	15	255	1	2
----	----	----	-----	---	---


$6 \times 8 + 11$  59

# 追踪解

算法 Traceback ( $n, l$ )

输入：数组  $l$

输出：数组  $C$

1.  $j \leftarrow 1$       //  $j$  为正在追踪的段数
2. while  $n \neq 0$  do
3.     $C[j] \leftarrow l[n]$  
4.     $n \leftarrow n - l[n]$
5.     $j \leftarrow j + 1$

$C[j]$ : 从后向前追踪的第  $j$  段的长度

时间复杂度:  $O(n)$

# 小结

- 图像变位存储问题的建模
- 子问题边界的界定
- 递推方程及初值
- 伪码
- 标记函数与解的追踪
- 时间复杂度

# 最大子段和

# 最大子段和

**问题：** 给定 $n$ 个数(可以为负数)的序列

$$(a_1, a_2, \dots, a_n)$$

$$\text{求 } \max\{0, \max_{1 \leq i \leq j \leq n} \sum_{k=i}^j a_k\}$$

**实例**

$$(-2, 11, -4, 13, -5, -2)$$

**解：** 最大子段和为  $a_2+a_3+a_4=20$

# 算法

**算法1:** 对所有的  $(i, j)$  对, 顺序求和  $a_i + \dots + a_j$  并比较出最大的和

**算法2:** 分治策略, 将数组分成左右两半, 分别计算左边的最大和、右边的最大和、跨边界的最大和, 然后比较其中最大者

**算法3:** 动态规划



# 算法1

算法 Enumerate

输入：数组  $A[1..n]$ ,

输出：  $sum, first, last$

1.  $sum \leftarrow 0$
2. for  $i \leftarrow 1$  to  $n$  do
3.     for  $j \leftarrow i$  to  $n$  do
4.          $thisum \leftarrow 0$
5.         for  $k \leftarrow i$  to  $j$  do
6.              $thisum \leftarrow thisum + A[k]$
7.         if  $thisum > sum$
8.             then  $sum \leftarrow thisum$
9.              $first \leftarrow i$
10.             $last \leftarrow j$

和的  
边界  $i-j$

找到更  
大和

时间复杂度：  $O(n^3)$

# 算法2 分治策略

将序列分成左右两半，中间分点  $center$

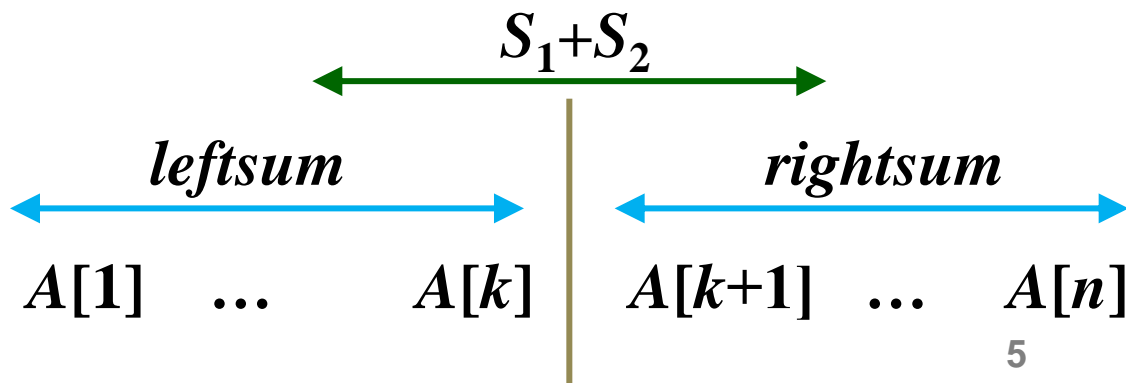
递归计算左段最大子段和  $leftsum$

递归计算右段最大子段和  $rightsum$

$center$  到  $a_1$  的最大和  $S_1$ ,  $k=center$

$center + 1$  到  $a_n$  的最大和  $S_2$

$\max \{ leftsum, rightsum, S_1+S_2 \}$



# 伪码

算法 MaxSubSum ( $A, left, right$ )

输入：数组  $A, left, right$  (左,右边界)

输出：最大子段和 $sum$ 及子段边界

1. if  $|A|=1$  then 输出元素(值为负输出0)
2.  $center \leftarrow \lfloor (left+right)/2 \rfloor$
3.  $leftsum \leftarrow \text{MaxSubSum}(A, left, center)$
4.  $rightsum \leftarrow \text{MaxSubSum}(A, center+1, right)$
5.  $S_1 \leftarrow A_1[center]$  //从 $center$ 向左
6.  $S_2 \leftarrow A_2[center+1]$  //从 $center+1$ 向右
7.  $sum \leftarrow S_1 + S_2$
8. if  $leftsum > sum$  then  $sum \leftarrow leftsum$
9. if  $rightsum > sum$  then  $sum \leftarrow rightsum$

# 时间复杂度

计算从  $k = center$  开始到  $a_1$  方向的最大和，每次加1个元素，得到

$A[k]$ ,  $A[k]+A[k-1]$ ,  $A[k]+A[k-1]+A[k-2]$ ,  
... ,  $A[k]+... +A[1]$

比较上述的最大和，时间为  $O(n)$  ,  
右半边也是  $O(n)$

$$T(n) = 2T(n/2) + O(n)$$

$$T(c) = O(1)$$

$$T(n) = O(n \log n)$$

# 算法3： 动态规划

子问题界定： 前边界为 1， 后边界  $i$ ，  
 $C[i]$  是  $A[1..i]$  中必须包含元素  $A[i]$  的  
向前连续延伸的最大子段和

$$C[i] = \max_{1 \leq k \leq i} \left\{ \sum_{j=k}^i A[j] \right\}$$



最大子段和  $C[i]$

# 优化函数的递推方程

递推方程:

$$\underline{C[i] = \max\{C[i-1] + A[i], A[i]\}}$$

$$i = 2, \dots, n$$

$$C[1] = A[1] \quad \text{若 } A[1] > 0$$

$$C[1] = 0 \quad \text{否则}$$

$$\text{解: } \text{OPT}(A) = \max_{1 \leq i \leq n} \{C[i]\}$$

# 伪码

算法 MaxSum ( $A, n$ )

输入：数组  $A$

输出：最大子段和  $sum$ , 子段最后位置  $c$

1.  $sum \leftarrow 0$
2.  $b \leftarrow 0$
3. for  $i \leftarrow 1$  to  $n$  do
4.   if  $b > 0$
5.   then  $b \leftarrow b + A[i]$
6.   else  $b \leftarrow A[i]$
7.   if  $b > sum$
8.   then  $sum \leftarrow b$
9.        $c \leftarrow i$
10. return  $sum, c$

子问题  
后边界  $i$

找到更  
大的和

时间复杂度：  $O(n)$ , 空间复杂度：  $O(n)$

# 小结

- 几个算法：蛮力, 分治, 动态规划
- 动态归划算法：
  - 子问题界定
  - 列优化函数的递推方程和边界条件
  - (不一定是原问题的优化函数)
  - 自底向上计算, 设计备忘录 (表格)
  - 如何根据动态规划的解找原问题的解
  - 时间复杂度估计



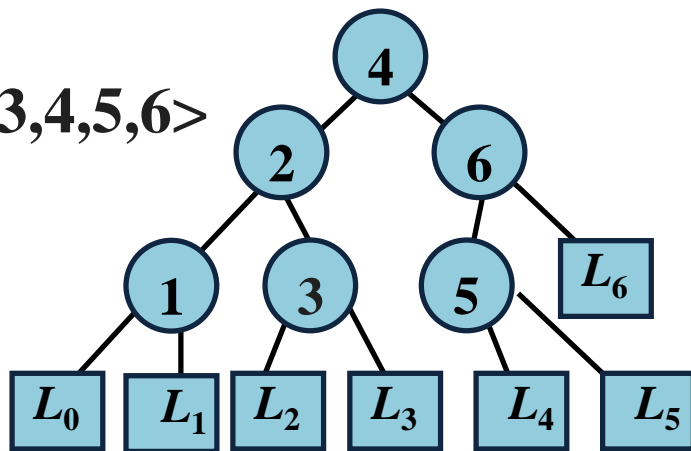
# 最优二叉检索树

# 二叉检索树

集合 $S$ 为排序的  $n$  个元素,  $x_1 < x_2 < \dots < x_n$ , 将这些元素存储在一棵二叉树的结点上, 以查找  $x$  是否在这些数中. 如果  $x$  不在, 确定  $x$  在那个空隙 (方结点).

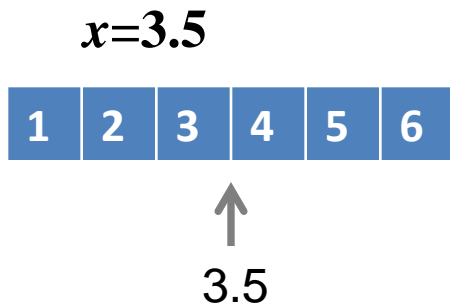
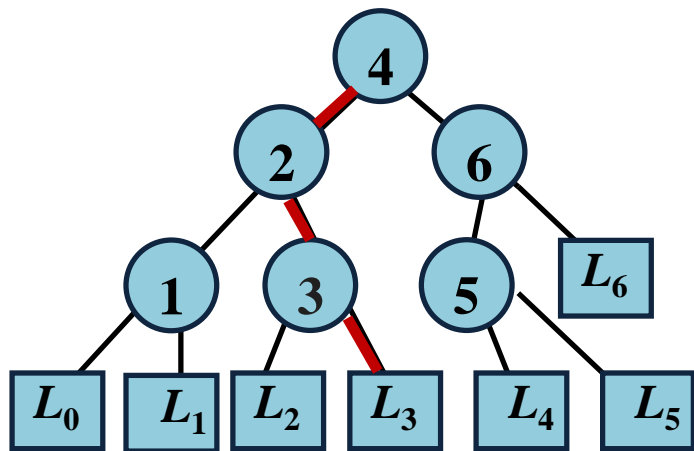
实例:

$S = \langle 1, 2, 3, 4, 5, 6 \rangle$



# 二叉树的检索方法

1. 初始,  $x$  与根元素比较;
2.  $x <$  根元素, 递归进入左子树;
3.  $x >$  根元素, 递归进入右子树;
4.  $x =$  根元素, 算法停止, 输出  $x$ ;
5.  $x$  到叶结点算法停止, 输出  $x$  不在数组.



# 数据元素存取概率分布

空隙:

$$(x_0, x_1), (x_1, x_2), \dots, (x_{n-1}, x_n), (x_n, x_{n+1}),$$

$$x_0 = -\infty, \quad x_{n+1} = +\infty$$

给定序列  $S = \langle x_1, x_2, \dots, x_n \rangle$ ,

$x$  在  $x_i$  的概率为  $b_i$ ,

$x$  在  $(x_i, x_{i+1})$  的概率为  $a_i$ ,

$S$  的存取概率分布如下:

$$P = \langle a_0, b_1, a_1, b_2, a_2, \dots, b_n, a_n \rangle$$

# 实例

实例:  $S = \langle 1, 2, 3, 4, 5, 6 \rangle$

$P = \langle 0.04, \mathbf{0.1}, 0.01, \mathbf{0.2}, 0.05, \mathbf{0.2}, 0.02, \mathbf{0.1}, 0.02, \mathbf{0.1}, 0.07, \mathbf{0.05}, 0.04 \rangle$

1, 2, 3, 4, 5, 6 检索的概率分别为:

$\mathbf{0.1}, \mathbf{0.2}, \mathbf{0.2}, \mathbf{0.1}, \mathbf{0.1}, \mathbf{0.05}$

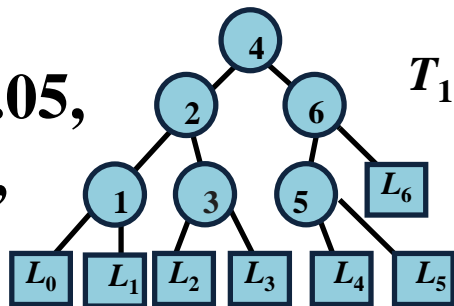
各个空隙的检索概率分别为:

$0.04, 0.01, 0.05, 0.02, 0.02, 0.07, 0.04$

# 检索数据的平均时间

$S = \langle 1, 2, 3, 4, 5, 6 \rangle$

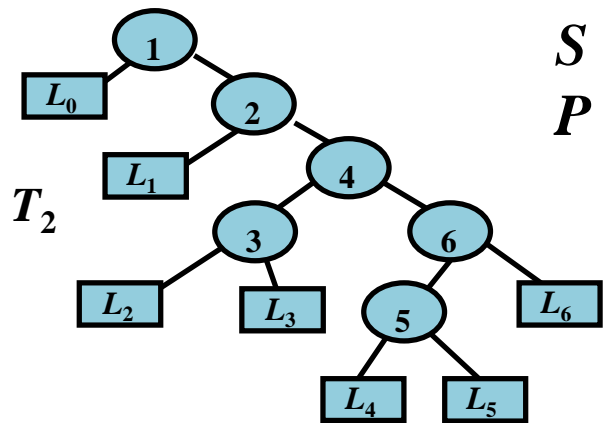
$P = \langle 0.04, \mathbf{0.1}, 0.01, \mathbf{0.2}, 0.05, \mathbf{0.2}, 0.02, \mathbf{0.1}, 0.02, \mathbf{0.1}, 0.07, \mathbf{0.05}, 0.04 \rangle$



$m(T_1)$

$$\begin{aligned} &= [1 \times 0.1 + 2 \times (0.2 + 0.05) + 3 \times (0.1 + 0.2 + 0.1)] \\ &\quad + [3 \times (0.04 + 0.01 + 0.05 + 0.02 + 0.02 + 0.07) \\ &\quad + 2 \times 0.04] \\ &= 1.8 + \mathbf{0.71} = \mathbf{\underline{2.51}} \end{aligned}$$

# 检索数据的平均时间



$S = \langle 1, 2, 3, 4, 5, 6 \rangle$

$P = \langle 0.04, \mathbf{0.1}, 0.01, \mathbf{0.2},$   
 $0.05, \mathbf{0.2}, 0.02, \mathbf{0.1},$   
 $0.02, \mathbf{0.1}, 0.07, \mathbf{0.05},$   
 $0.04 \rangle$

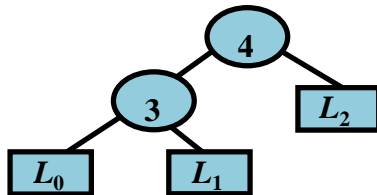
$m(T_2)$

$$\begin{aligned} &= [1 \times 0.1 + 2 \times 0.2 + 3 \times 0.1 + 4 \times (0.2 + 0.05) + 5 \times 0.1] \\ &\quad + [1 \times 0.04 + 2 \times 0.01 + 4 \times (0.05 + 0.02 + 0.04) \\ &\quad + 5 \times (0.02 + 0.07)] \\ &= 2.3 + 0.95 = \underline{\underline{3.25}} \end{aligned}$$

# 平均比较次数计算

数据集  $S = \langle x_1, x_2, \dots, x_n \rangle$

存取概率分布



$P = \langle a_0, b_1, a_1, b_2, \dots, a_i, b_{i+1}, \dots, b_n, a_n \rangle$

结点  $x_i$  在  $T$  中的深度是  $d(x_i)$ ,  $i=1,2,\dots,n$ ,

空隙  $L_j$  的深度为  $d(L_j)$ ,  $j=0,1,\dots,n$ ,

平均比较次数为:

$$t = \sum_{i=1}^n b_i (1 + d(x_i)) + \sum_{j=0}^n a_j d(L_j)$$



# 问题

给定数据集

$$S = \langle x_1, x_2, \dots, x_n \rangle,$$

及  $S$  的存取概率分布如下:

$$P = \langle a_0, b_1, a_1, b_2, a_2, \dots, b_n, a_n \rangle$$

求一棵最优的 ( 即平均比较次数最少的 ) 二分检索树.

# 小结

- 二叉检索树的构成
- 给定概率分布下，一棵二叉检索树的平均检索时间估计
- 什么是最优二叉检索树

# 最优二叉检索 树的算法

# 关键问题

子问题边界界定

如何将该问题归结为更小的子问题

优化函数的递推方程及初值

计算顺序

是否需要标记函数

时间复杂度分析

# 子问题划分

子问题边界为( $i, j$ )

数据集:  $S[i, j] = \langle x_i, x_{i+1}, \dots, x_j \rangle$

存取概率分布:

$$P[i, j] = \langle a_{i-1}, b_i, a_i, b_{i+1}, \dots, b_j, a_j \rangle$$

输入实例:  $S = \langle A, \underline{B, C, D}, E \rangle$

$$P = \langle 0.04, \mathbf{0.1}, \underline{0.02, \mathbf{0.3}, 0.02, \mathbf{0.1}}, \\ \underline{0.05, \mathbf{0.2}}, 0.06, \mathbf{0.1}, 0.01 \rangle$$

子问题:  $S[2, 4] = \langle B, C, D \rangle$

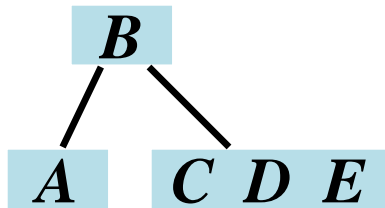
$$P[2, 4] = \langle 0.02, \mathbf{0.3}, 0.02, \mathbf{0.1}, 0.05, \mathbf{0.2}, 0.06 \rangle$$

# 子问题归约

以  $x_k$  作为根归结为子问题:

$$S[i, k-1], P[i, k-1]$$

$$S[k+1, j], P[k+1, j]$$



$$S[1,5] = \langle A, B, C, D, E \rangle$$

$$P[1,5] = \langle 0.04, \mathbf{0.1}, 0.02, \mathbf{0.3}, 0.02, \mathbf{0.1}, \\ 0.05, \mathbf{0.2}, 0.06, \mathbf{0.1}, 0.01 \rangle$$

$$S[1,1] = \langle A \rangle,$$

$$P[1,1] = \langle 0.04, \mathbf{0.1}, 0.02 \rangle$$

$$S[3,5] = \langle C, D, E \rangle,$$

$$P[3,5] = \langle 0.02, \mathbf{0.1}, 0.05, \mathbf{0.2}, 0.06, \mathbf{0.1}, 0.01 \rangle$$

# 子问题的概率之和

子问题界定  $S[i,j]$  和  $P[i,j]$ , 令

$$w[i,j] = \sum_{p=i-1}^j a_p + \sum_{q=i}^j b_q$$

是  $P[i,j]$  中所有概率(数据与空隙)之和

实例:  $S[2,4]=\langle B,C,D \rangle$

$$P[2,4]=\langle 0.02, \mathbf{0.3}, 0.02, \mathbf{0.1}, 0.05, \mathbf{0.2}, 0.06 \rangle$$

$$w[2,4]=(\mathbf{0.3}+\mathbf{0.1}+\mathbf{0.2})$$

$$+(\mathbf{0.02}+\mathbf{0.02}+\mathbf{0.05}+\mathbf{0.06})$$

$$= \mathbf{0.75}$$

# 优化函数的递推方程

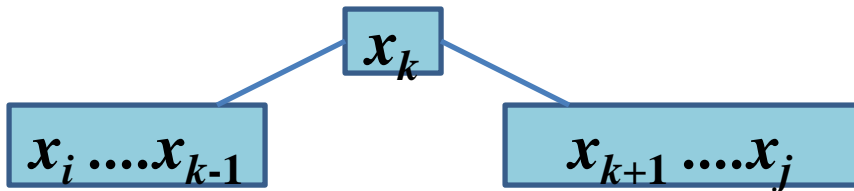
设  $m[i,j]$  是相对于输入  $S[i,j]$  和  $P[i,j]$  的最优二叉搜索树的平均比较次数

递推方程:

$$m[i,j] = \min_{i \leq k \leq j} \{ \underline{m[i, k-1]} + \underline{m[k+1, j]} + w[i, j] \},$$

$$1 \leq i \leq j \leq n$$

$$m[i, i-1] = 0, \quad i = 1, 2, \dots, n$$





# $m[i, j]_k$ 公式的证明

$m[i, j]_k$  : 根为  $x_k$  时平均比较次数的最小值

作为子  
树增加  
次数

$$m[i, j]_k$$

$$= (m[i, k-1] + \underline{w[i, k-1]}) + (m[k+1, j] + \underline{w[k+1, j]}) + 1 \times b_k$$

$$= (m[i, k-1] + m[k+1, j]) + (w[i, k-1] + b_k + w[k+1, j])$$

$$= (m[i, k-1] + m[k+1, j]) + (\underbrace{\sum_{p=i-1}^{k-1} a_p + \sum_{q=i}^{k-1} b_q}_{\text{代入概率公式}}) + b_k + (\underbrace{\sum_{p=k}^j a_p + \sum_{q=k+1}^j b_q}_{\text{代入概率公式}})$$

代入概  
率公式

$$= (m[i, k-1] + m[k+1, j]) + \sum_{p=i-1}^j a_p + \sum_{q=i}^j b_q$$

化简

$$= \boxed{m[i, k-1] + m[k+1, j] + w[i, j]}$$

# 递推方程

$$m[i, j] = \min_{i \leq k \leq j} \{m[i, k-1] + m[k+1, j] + w[i, j]\},$$

平均比较次数：在所有  $k$  的情况下

$m[i, j]_k$  的最小值

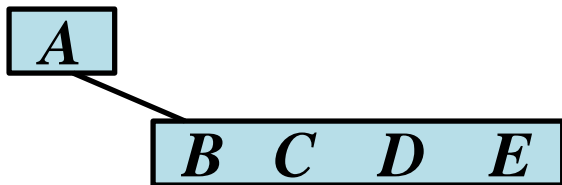
$$m[i, j] = \min \{ m[i, j]_k \mid i \leq k \leq j \}$$

初值  $m[i, i-1]=0$  对应于空的子问题，

例如  $S = \langle A, B, C, D, E \rangle$ ，取  $A$  作根，

$i = 1, k=1$ ，左边子问题为空树，

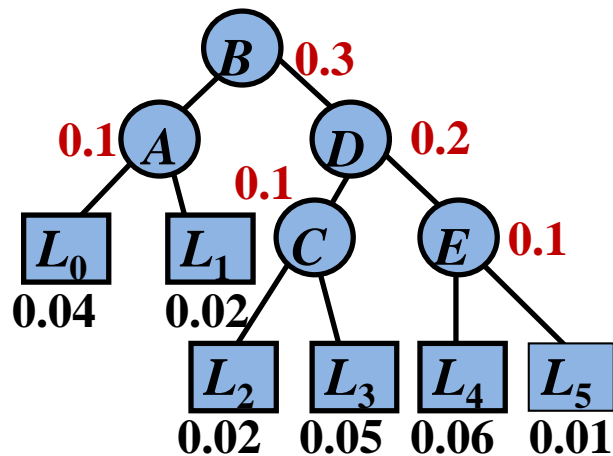
对应于： $S[1,0]$ ， $m[1,0]=0$  的情况。



# 实例

$$m[i, j] = \min_{i \leq k \leq j} \{ m[i, k-1] + m[k+1, j] + w[i, j] \}$$

$$m[i, i-1] = 0$$



以B为根

$k=2$

$$m[1,1]=0.16$$

$$m[3,5]=0.88$$

$$m[1,5] = 1 + \min_{k=2,3,4} \{ m[1, k-1] + m[k+1, 5] \}$$

$$= 1 + \{ m[1,1] + m[3,5] \} = 1 + \{ 0.16 + 0.88 \} = 2.04$$

# 计算复杂性估计

$$m[i, j] = \min_{i \leq k \leq j} \{m[i, k-1] + m[k+1, j] + w[i, j]\}$$

$$1 \leq i \leq j \leq n$$

$$m[i, i-1] = 0, \quad i = 1, 2, \dots, n$$

$i, j$  的所有组合  $O(n^2)$  种

每种要对不同的  $k$  进行计算,  $k = O(n)$

每次计算为常数时间

时间复杂性:  $T(n) = O(n^3)$

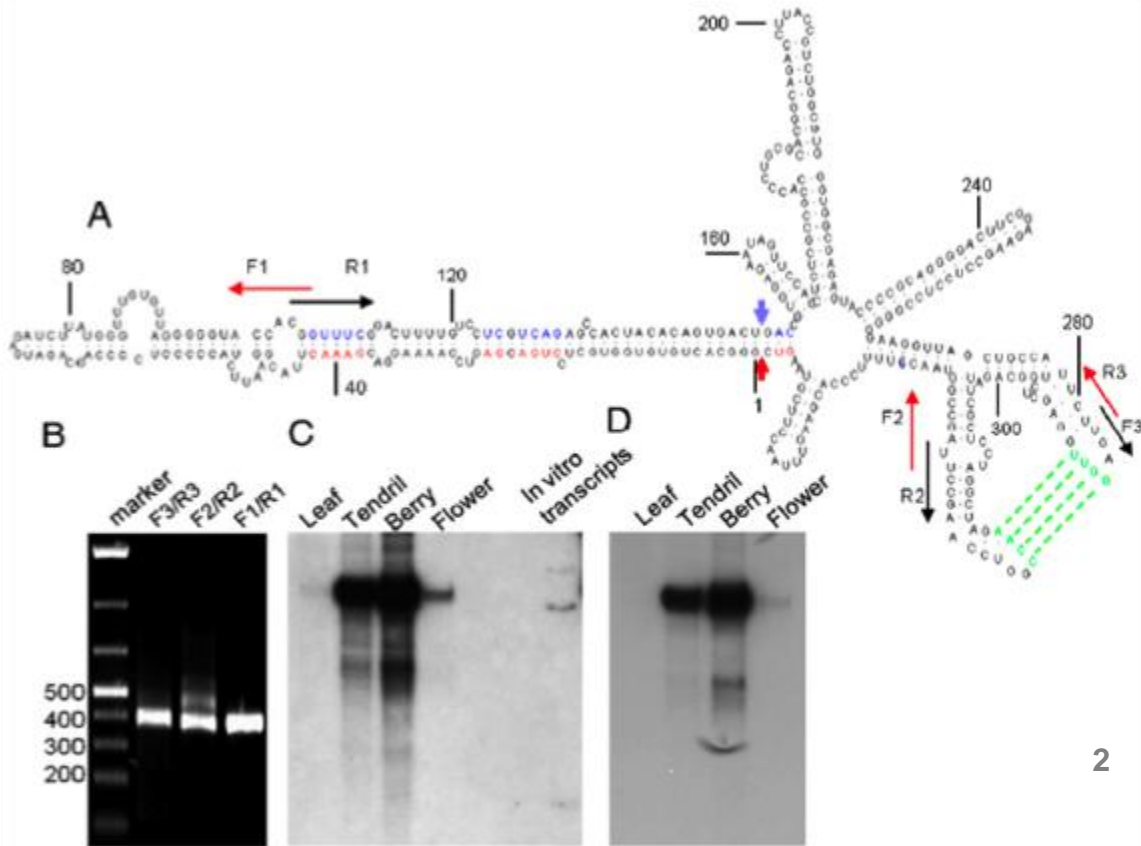
空间复杂度:  $S(n) = O(n^2)$

# 小结

- 划分子问题,以数据结点作为树根
- 定义优化函数,列出递推方程与边界条件
- 自底向上计算,设计备忘录(表格)
- 设立标记函数记录构成最优二叉搜索树或子树时根的位置.
- 时间复杂度估计

# RNA二级结构预测

# 375nt 的环形类病毒GHVd 的 RNA二级结构预测和RT-PCR、Northern blot检测结果

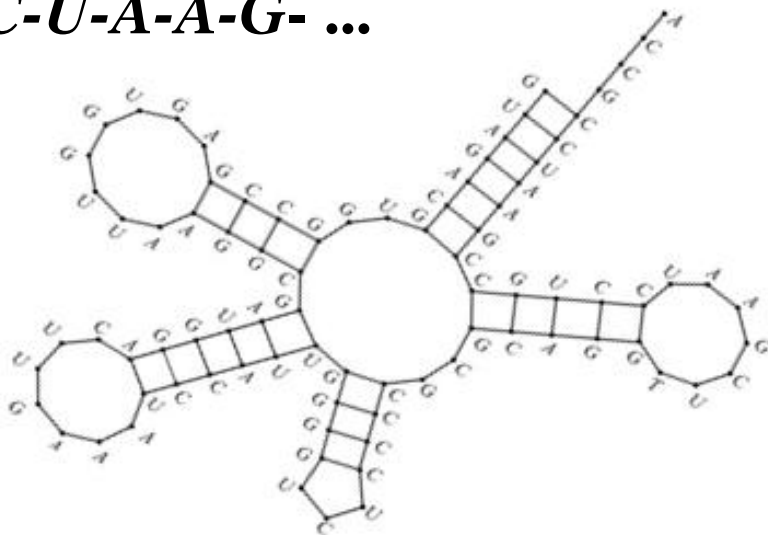


# RNA二级结构

**一级结构：** 由字母 *A, C, G, U* 标记的核苷酸构成的一条链.

实例： *A-C-C-G-C-C-U-A-A-G-C-C-G-U-C-C-U-A-A-G- ...*

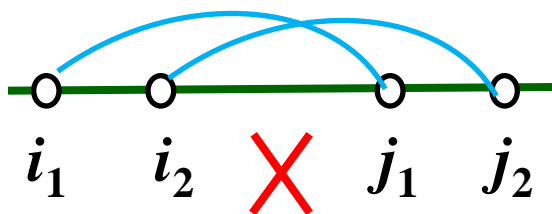
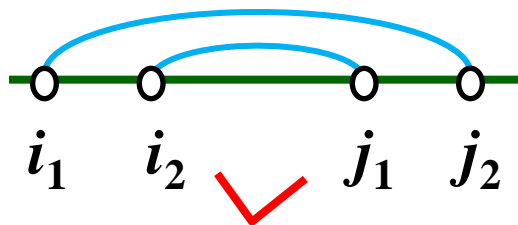
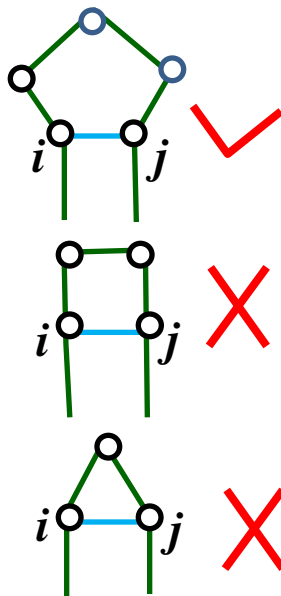
**二级结构：**  
核苷酸相互  
匹配构成的  
平面结构



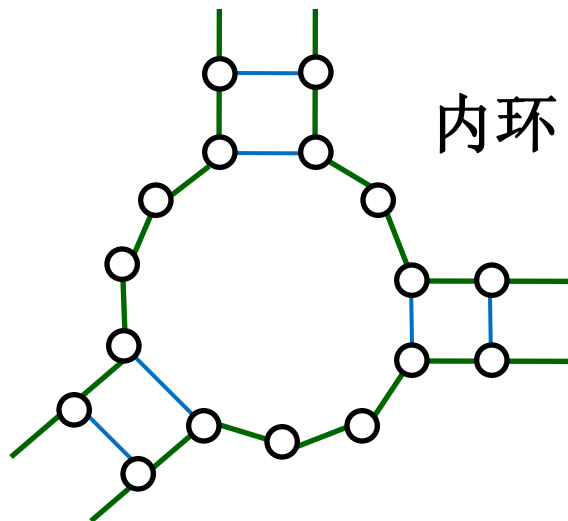
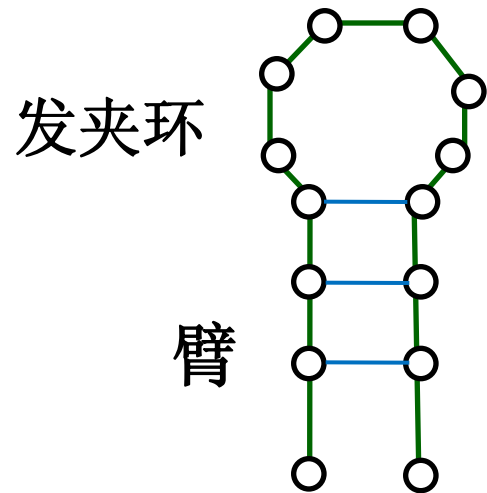


# 匹配原则

- 配对  $U-A$ ,  $C-G$
- 末端不出现“尖角”，位置  $i-j$  配对，则  $i \leq j-4$
- 每个核苷酸只能参加一个配对
- 不允许交叉，即如果位置  $i_1, i_2, j_1, j_2$  满足  $i_1 < i_2 < j_1 < j_2$ ，不允许  $i_1-j_1, i_2-j_2$  配对，但可以允许  $i_1-j_2, i_2-j_1$  配对。



# 匹配的结构



# RNA二级结构问题

给定RNA的一条链（一级结构），预测它的可能的稳定的二级结构

稳定二级结构满足的条件

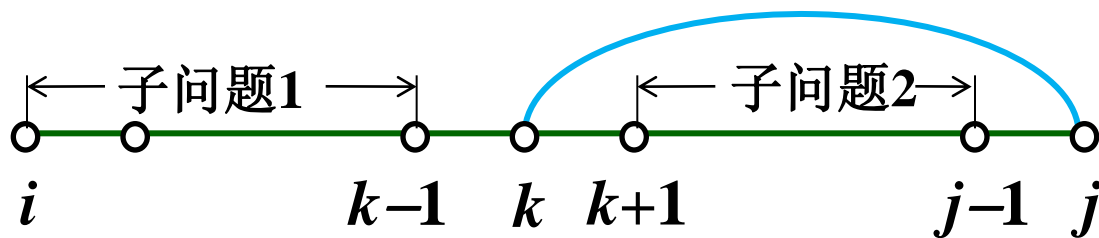
生物学条件：具有最小自由能

简化条件：具有最多的匹配对数

**问题：**给定RNA链，求具有最多匹配对数的二级结构，即最优结构。

# 建模

- 子问题界定：前边界  $i$ , 后边界  $j$
- 若  $j$  与  $k$  (所有可能) 位置匹配, 归约为  
子问题 1:  $i$  到  $k-1$  的链  
子问题 2:  $k+1$  到  $j-1$  的链



- 若  $j$  不参与匹配, 则原问题归约为  $i$  到  $j-1$  的子问题

# 优化函数的递推方程

令 $C[i,j]$ 是序列 $S[i..j]$ 的最大匹配对数

$$C[i,j] = \max\{C[i,j-1],$$
$$\max_{i \leq k \leq j-4} \{1 + C[i,k-1] + C[k+1,j-1]\}\}$$

$$1 \leq i, \quad j \leq n, \quad j - i \geq 4$$

$$C[i,j] = 0 \quad j - i < 4$$

满足优化原则

计算顺序：按照子问题长度计算

# 计算复杂度分析

子问题个数:  $i, j$  对的组合有  $O(n^2)$  个

对于给定的  $i$  和  $j$ ,  $j$  需要考察与所有可能的  $k$  是否匹配, 其中  $i \leq k \leq j-4$ , 需要  $O(n)$  时间.

算法时间复杂度是  $O(n^3)$ .

# 小结

- 划分子问题，确定子问题边界 $i, j$ 与归约方法.
- 定义优化函数,列递推方程和初值.
- 自底向上计算，设计备忘录 (表格)
- 设立标记函数，记下最优划分位置
- 时间复杂度估计

# 序列比对



# 序列比对

- 为确定两个序列之间的相似性或同源性，将它们按照一定的规律排列，进行比对.
- 应用：  
生物信息学中用于研究同源性，如蛋白质序列或 **DNA** 序列. 在比对中，错配与突变相对应，空位与插入或缺失相对应.  
计算语言学中用于语言进化或文本相似性的研究.

# 序列之间的编辑距离

编辑距离：

给定两个序列  $S_1$  和  $S_2$ ,

通过一系列字符编辑（插入、删除、替换）等操作，将  $S_1$  转变成  $S_2$ .

完成这种转换所需要的最少的编辑操作个数称为  $S_1$  和  $S_2$  的编辑距离.

# 实例

`vintner` 转变成 `writers`,  
编辑距离  $\leq 6$

	<code>v i n t n e r</code>
删除 v:	<code>- i n t n e r</code>
插入 w:	<code>w i n t n e r</code>
插入 r:	<code>w r i n t n e r</code>
删除 n:	<code>w r i - t n e r</code>
删除 n:	<code>w r i t - e r</code>
插入 s:	<code>w r i t e r s</code>

# 子问题界定和归约

$S_1[1..n]$  和  $S_2[1..m]$  表示两个序列

子问题:  $S_1[1..i]$  和  $S_2[1..j]$ , 边界  $(i, j)$

操作	归约子问题	编辑距离
删除 $S_1[i]$	$(i-1, j)$	+1
$S_1[i]$ 后插入 $S_2[j]$	$(i, j-1)$	+1
$S_1[i]$ 替换为 $S_2[j]$	$(i-1, j-1)$	+1
$S_1[i]=S_2[j]$	$(i-1, j-1)$	+0

# 优化函数的递推方程

$C[i,j]$ :  $S_1[1..i]$  和  $S_2[1..j]$  的编辑距离

$$C[i,j] = \min\{C[i-1,j]+1, C[i,j-1]+1, \\ C[i-1,j-1]+t[i,j]\}$$

$$t[i,j] = \begin{cases} 0 & S_1[i] = S_2[j] \\ 1 & S_1[i] \neq S_2[j] \end{cases}$$

$$C[0,j] = j,$$

$$C[i,0] = i$$

# 计算复杂度分析

- 子问题 由  $i, j$  界定,  
有  $O(mn)$  个子问题
- 每个子问题的计算  
为常数时间
- 算法的时间复杂度是  $O(nm)$

# 动态规划算法设计要点

- (1) 引入参数来界定子问题的边界. 注意子问题的重叠程度.
- (2) 给出带边界参数的优化函数定义与优化函数的递推关系, 找到递推关系的初值.
- (3) 判断该优化问题是否满足优化原则.
- (4) 考虑是否需要标记函数.

# 动态规划算法设计要点(续)

- (5) 采用自底向上的实现技术，从最小的子问题开始迭代计算，计算中用备忘录保留优化函数和标记函数的值。
- (6) 动态规划算法的时间复杂度是对所有子问题(备忘录)的计算工作量求和(可能需要追踪解的工作量)
- (7) 动态规划算法一般使用较多的存储空间，这往往成为限制动态规划算法使用的瓶颈因素。