3.3.5 最大子段和

问题：给定n个整数（可以为负数）的序列

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

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

实例：(-2, 11, -4, 13, -5, -2)

解：最大子段和 $a_2+a_3+a_4= 20$

算法1---顺序求和+比较

算法2---分治策略

算法3---动态规划

算法1 顺序求和+比较

算法 Enumerate

输入：数组$A[1..n]$

输出：$sum, first, last$

```
sum ← 0
for i ← 1 to n do              // i为当前和的首位置
    for j ← i to n do          // j为当前和的末位置
        thissum ← 0            // thissum为A[i]到A[j]之和
        for k ← i to j do
            thissum ← thissum+A[k]
        if thissum > sum
        then sum ← thissum
            first ← i          // 记录最大和的首位置
            last ← j           // 记录最大和的末位置
```

时间复杂度：$O(n^3)$

算法2 分治策略

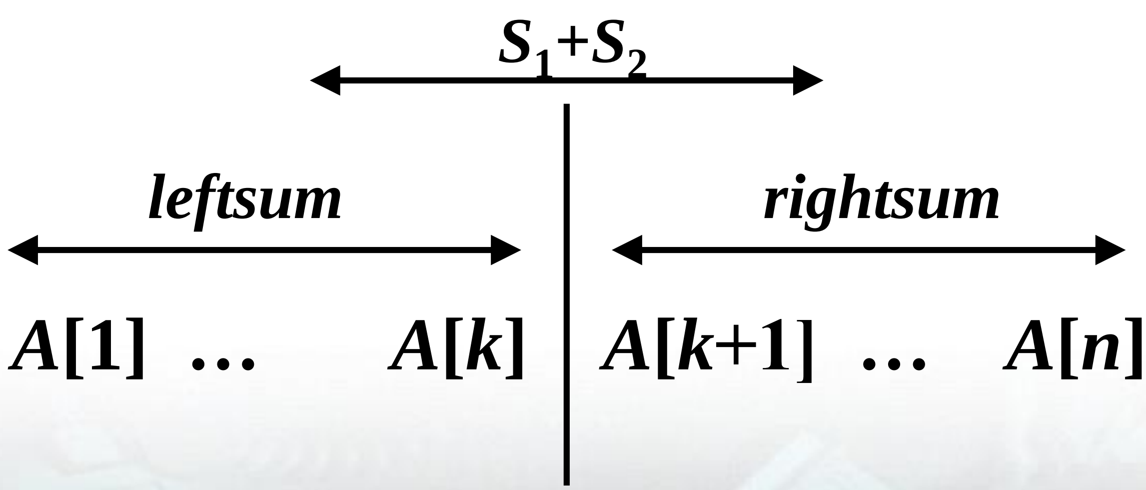
将序列分成左右两半，中间分点*center*

递归计算左段最大子段和 ***leftsum***

递归计算右段最大子段和 ***rightsum***

$a_{center} \to a_1$的最大和S_1，$a_{center+1} \to a_n$的最大和S_2

max { *leftsum*, *rightsum*, S_1+S_2}



分治算法

算法 MaxSubSum(A, *left*, *right*）

输入：数组A，*left*，*right*分别是A的左、右边界
输出：A的最大子段和*sum*及其子段的前后边界

1. if $|A| = 1$ then 输出元素值（当值为负时输出0）
2. $center \leftarrow \lfloor (left+right)/2 \rfloor$
3. $leftsum \leftarrow$ MaxSubSum(A,*left*,*center*)   //子问题A_1
4. $righsum \leftarrow$ MaxSubSum(A,*center*+1,*right*)   //子问题A_2
5. $S_1 \leftarrow A_1[center]$向左的最大和 //从center向左的最大和
6. $S_2 \leftarrow A_2[center+1]$向右的最大和 //从center+1向右的最大和
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：$T(n)=2T(n/2)+O(n)$, $T(c)=O(1)$
$T(n)=O(n\log n)$

算法3：动态规划

令$C[i]$ 是 $A[1..i]$中必须包含元素$A[i]$的最大子段和

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

$A[1]$ $A[2]$ $A[3]$ … $A[i\text{-}1]$ | $A[i]$ | … $A[n]$

最大子段和 $C[i\text{-}1]$

最大子段和 $C[i]$

递推方程: $C[i]= \max\{C[i-1]+A[i], A[i]\}$ $i=1,\dots,n$

$C[0]=0$

解: $\mathrm{OPT}(A) = \max_{0\le i\le n}\{C[i]\}$

北京大学

算法 MaxSum

算法3.10 MaxSum(A, n)

输入：数组A

输出：最大子段和sum, 子段的最后位置c

1. $sum \leftarrow 0$
2. $b \leftarrow 0$ // b是前一个最大子段和
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. $c \leftarrow i$ // 记录最大和的末项标号
10. return sum，c

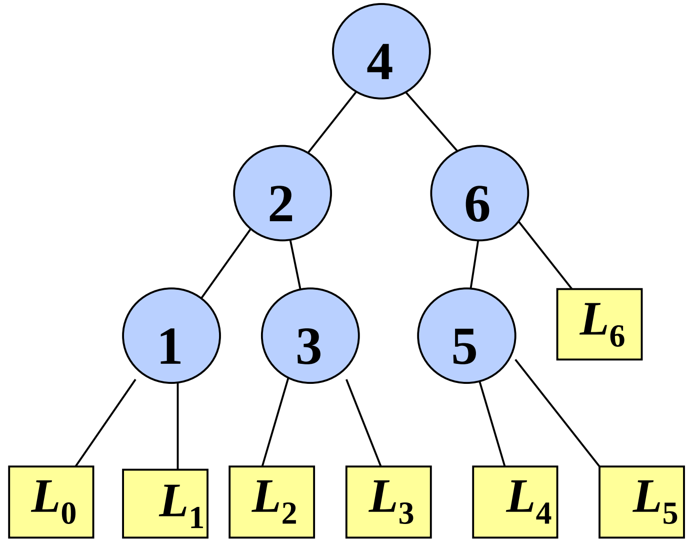
时间复杂度$O(n)$；空间复杂度$O(n)$

3.3.6 最优二叉检索树

设集合 S 为排序的 n 个元素 $x_1<x_2<\ldots<x_n$，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 在那个区间.

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到达叶结点，算法停止，输出x不在数组中.



$S=<1, 2, 3, 4, 5, 6>$

存取概率不等情况

区间： $(-\infty, x_1), (x_1, x_2), \ldots , (x_{n-1}, x_n), (x_n,+\infty)$,

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

给定序列 $S = <x_1, x_2, \ldots, x_n>$,

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：

$$P = <a_0, b_1, a_1, b_2, a_2, \ldots , b_n, a_n>$$

实例

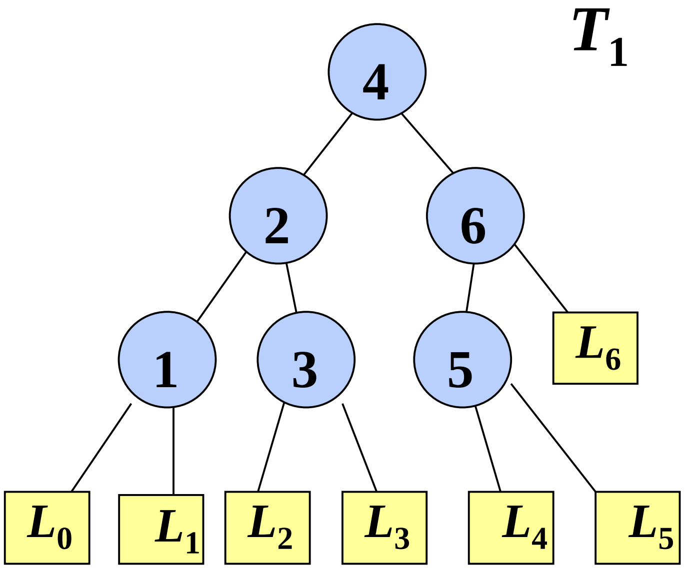
$S=<1,2,3,4,5,6>$

$P=<0.04,0.1,0.01,0.2,0.05,0.2,0.02,0.1,0.02,0.1,0.07, 0.05,0.04>$

实例

S = < 1, 2, 3, 4, 5, 6 >

P = < 0.04, 0.1, 0.01, 0.2, 0.05, 0.2,
0.02, 0.1, 0.02, 0.1, 0.07, 0.05,
0.04 >

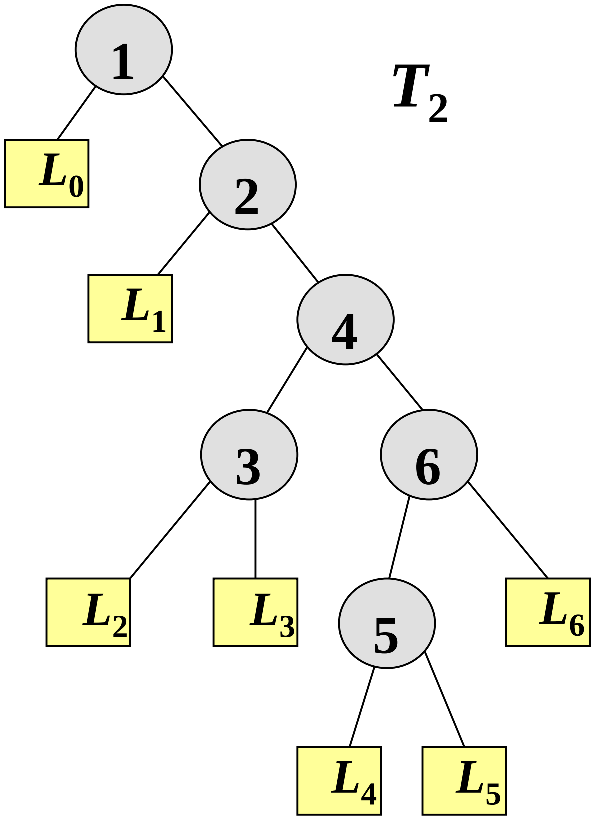


$$m(T_1)=[1*0.1+2*(0.2+0.05)+3*(0.1+0.2+0.1)]$$
$$+[3*(0.04+0.01+0.05+0.02+0.02+0.07)+2*0.04]$$
$$= 1.8+0.71=2.51$$

实例

S = < 1, 2, 3, 4, 5, 6 >

P = < 0.04, 0.1, 0.01, 0.2, 0.05, 0.2, 0.02, 0.1, 0.02, 0.1, 0.07, 0.05, 0.04 >



$$m(T_2) = [\,1*0.1+2*0.2 + 3*0.1 + 4*(0.2+0.05) + 5*0.1\,] + [\,1*0.04 + 2*0.01 + 4*(0.05 + 0.02 + 0.04) + 5*(0.02 + 0.07)] = 2.3 + 0.95 = 3.25$$

问题

数据集 $S=<x_1,x_2,\ldots,x_n>$

存取概率分布 $P=<a_0, b_1, a_1, b_2, \ldots, a_i, b_{i+1}, \ldots, b_n, a_n>$

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

区间 L_j 的深度为 $d(L_j)$, $j=0,1,\ldots,n$,

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 和相关存取概率分布 P，求一棵最优的(即平均比较次数最少的) 二分检索树.

算法设计:子问题划分

$S[i,j] = <x_i, x_{i+1}, \ldots, x_j>$ 是S 以 i 和 j 作为边界的子数据集

$P[i,j] = <a_{i-1}, b_i, a_i, b_{i+1}, \ldots, b_j, a_j>$是对应$S[i,j]$存取概率分布

例：$S=<A, B, C, D, E>$

$P=<0.04, 0.1, 0.02, 0.3, 0.02, 0.1, 0.05, 0.2, 0.06, 0.1, 0.01>$

$S[2,4]=<B, C, D>$

$P[2,4]=<0.02, 0.3, 0.02, 0.1, 0.05, 0.2, 0.06>$

子问题划分：以 x_k 作为根，划分成两个子问题：

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

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

例：以B为根，划分成以下子问题：

$S[1,1]=<A>$， $P[1,1]=<0.04, 0.1, 0.02>$

$S[3,5]=<C,D,E>$, $P[3,5]=<0.02, 0.1, 0.05, 0.2, 0.06, 0.1, 0.01>$

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，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（包括数据元素与空隙）之和

递推方程：

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

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

证明

$m[i,j]_k$：根为 x_k 时的二分检索树平均比较次数的最小值

$$
\begin{aligned}
& m[i,j]_k \\
&= (m[i,k-1]+w[i,k-1])+(m[k+1,j]+w[k+1,j])+1\times b_k \\
&= (m[i,k-1]+m[k+1,j])+(w[i,k-1]+b_k+w[k+1,j]) \\
&= (m[i,k-1]+m[k+1,j])+(\sum_{p=i-1}^{k-1} a_p+\sum_{q=i}^{k-1} b_q)+b_k+(\sum_{p=k}^{j} a_p+\sum_{q=k+1}^{j} b_q) \\
&= (m[i,k-1]+m[k+1,j])+\sum_{p=i-1}^{j} a_p+\sum_{q=i}^{j} b_q \\
&= m[i,k-1]+m[k+1,j]+w[i,j]
\end{aligned}
$$

平均比较次数：在所有 k 的情况下 $m[i,j]_k$ 的最小值，

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

实例

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

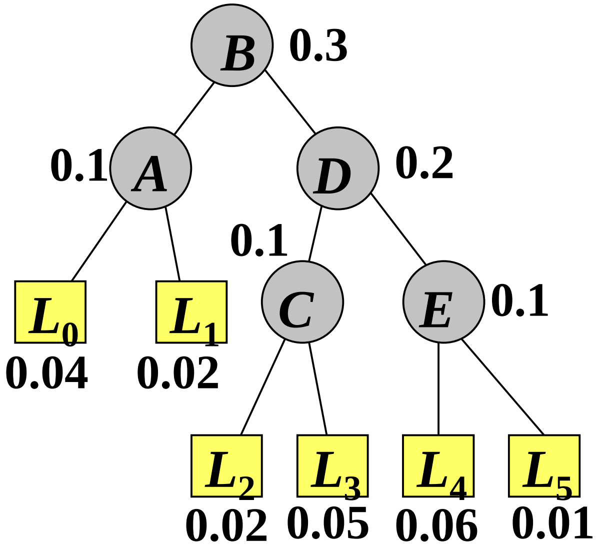
$$m[i,i-1]=0$$

$$m[1,1]=0.16,\quad m[3,5]=0.88$$

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

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

$$=1+\{0.16+0.88\}=2.04$$



复杂性估计：

$T(n)=O(n^3)$　　$S(n)=O(n^2)$

3.3.7生物信息学中的动态规划算法

RNA二级结构预测

一级结构：由字母 A,C,G,U 标记的核苷酸构成的一条链

二级结构：核苷酸相互匹配构成二级结构（平面图）

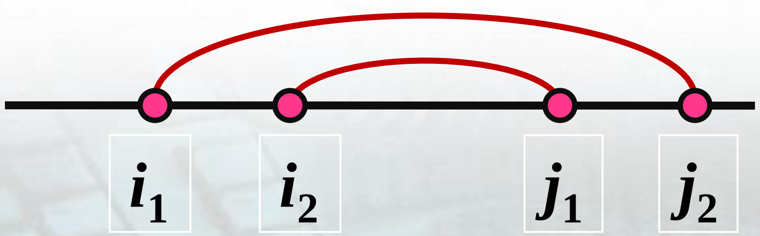
匹配原则：

(1) 配对U-A,C-G;

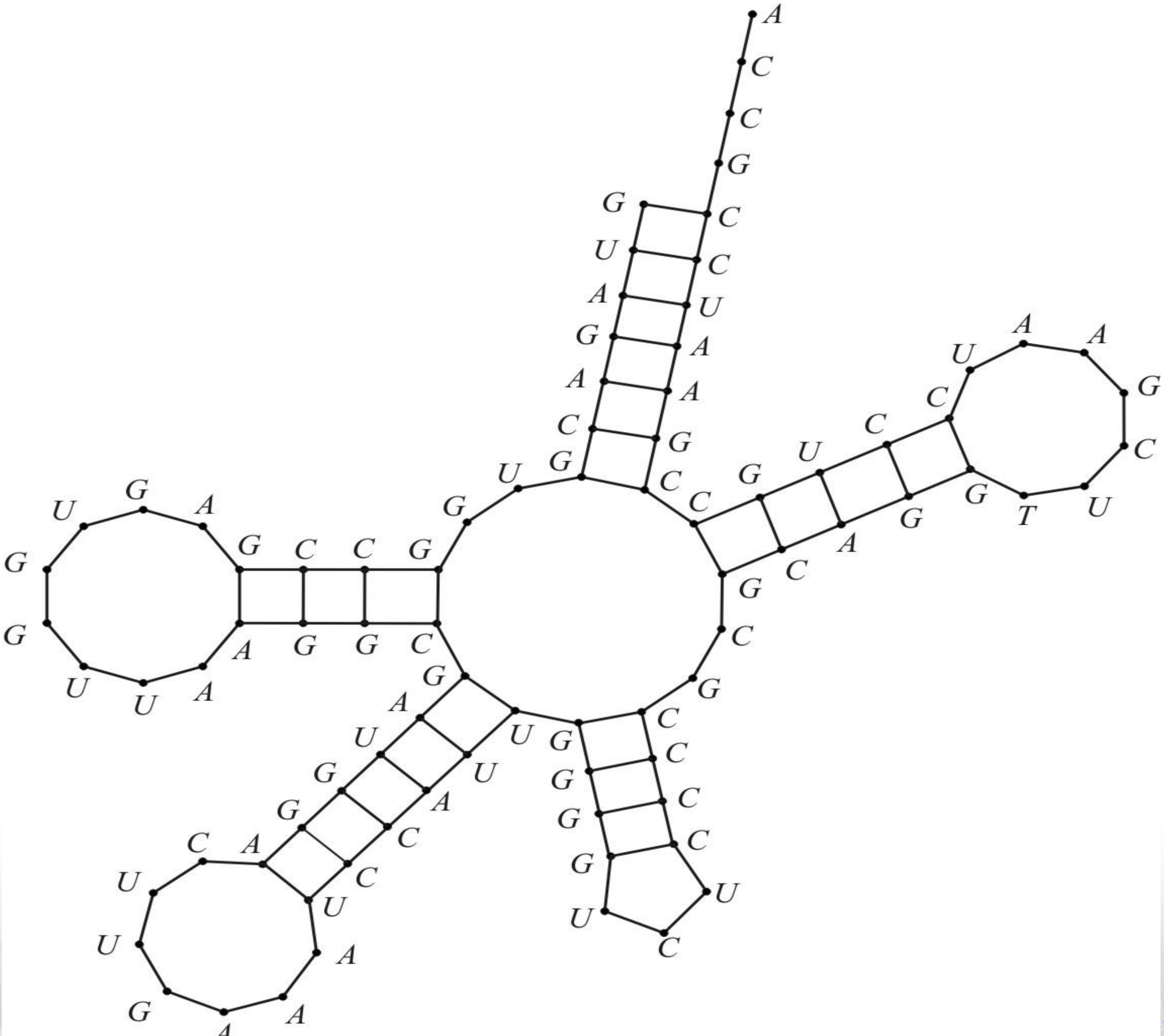
(2) 末端不出现“尖角”，位置i - j 配对，则 $i \le j-4$;

(3) 每个核苷酸只能参加一个配对;

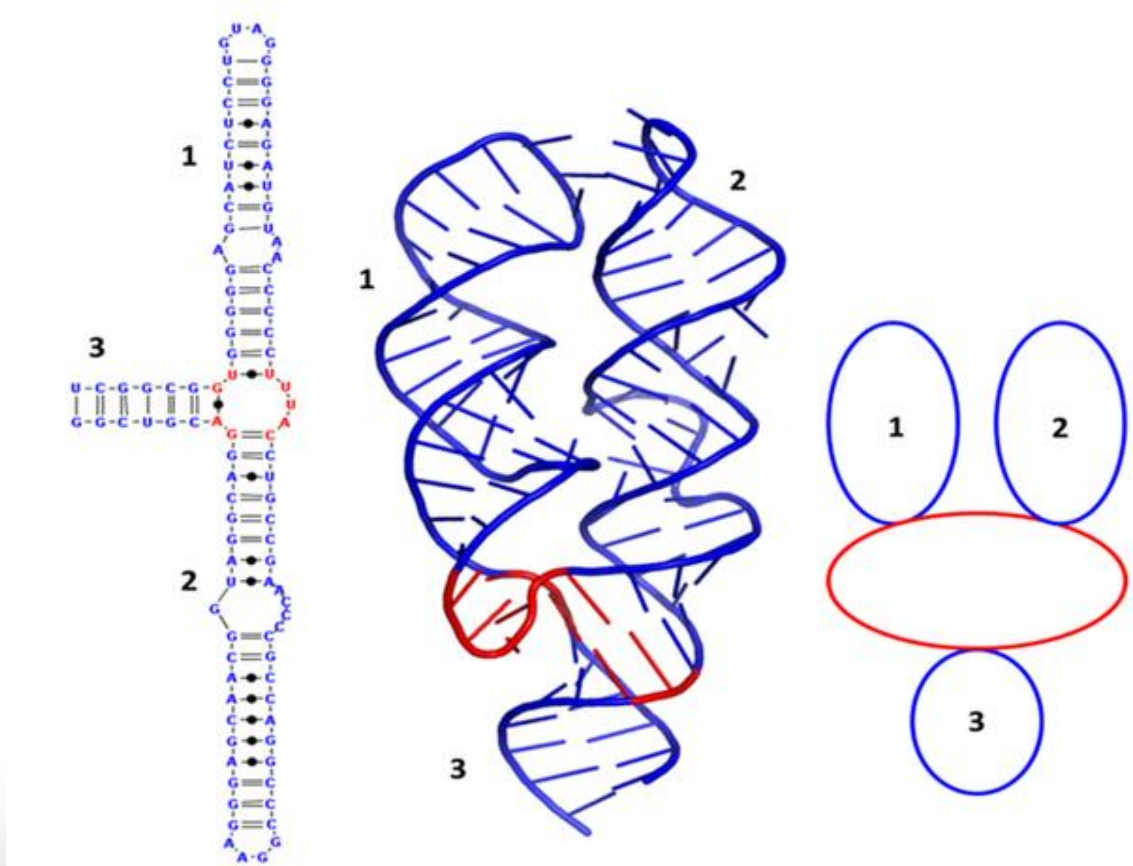
(4) 不允许交叉，即如果位置 i_1, i_2, j_1, j_2 满足$i_1<i_2<j_1<j_2$，不允许 i_1-j_1, i_2-j_2配对. 但可以允许i_1-j_2, i_2-j_1配对.



实例：4sRNA的二级结构

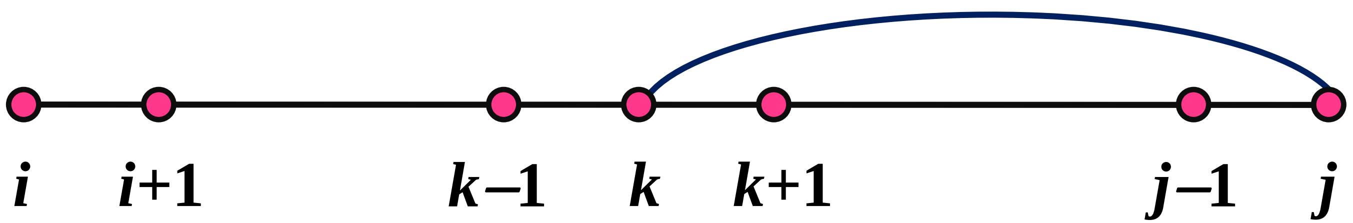


实例：RNA的多级结构



问题与算法设计

问题：给定RNA的一级结构：由*A*,*U*,*C*,*G* 构成的长为 *n* 的序列，寻找具有最大匹配对数的二级结构.



令C[*i*,*j*]是序列*S*[*i*..*j*]的最大匹配对数

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

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

算法时间复杂度是$O(n^3)$

序列比对

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）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≤6：

vintner

删除v：-intner

插入w：wintner

插入r：wrintner

删除n：wri-tner

删除n：writ-er

插入s：writers

序列比对

- 应用：生物信息学
  - 核苷酸的序列比对
    - AGGCTATCACCTGACCTCCAGGCCGATGCCC
    - TAGCTATCACGACCGCGGTCGATTTGCCCGAC
  - 比对结果
    - -**AG**G**CTATCAC**CT**GACC**T**C**CA**GG**C**CGA**--**TGCCC**---
    - T**AG**-**CTATCAC**--**GACC**GC--**GG**T**CGA**TT**TGCCC**GAC

算法设计

$S_1[1..n]$ 和 $S_2[1..m]$ 表示两个子序列

子问题划分：$S_1[1..i]$ 和 $S_2[1..j]$

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

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

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

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

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

算法的时间复杂度是$O(nm)$

Where did the name, dynamic programming, come from?

…The 1950s were not good years for mathematical research. [the] Secretary of Defense …had a pathological fear and hatred of the word, research…

I decided therefore to use the word, “programming”.

I wanted to get across the idea that this was dynamic, this was multistage… I thought, let’s … take a word that has an absolutely precise meaning, namely dynamic… it’s impossible to use the word, dynamic, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It’s impossible.

Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to.”

-- Richard Bellman, “Eye of the Hurricane: an autobiography” 1984.

source: Stanford CS124

小结

(1) 引入参数来界定子问题的边界.

(2) 判断该优化问题是否满足优化原则.

(3) 注意子问题的重叠程度.

(4) 给出带边界参数的优化函数定义与优化函数的递推关系考虑标记函数. 找到递推关系的初值.

(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.