

本周教学内容

动态规划算法的 设计与分析

投资问题
动态规划
算法

背包问题
动态规划
算法

最长公共子
序列问题动
态规划算法

递归实现

迭代实现

动态规划算法的设计要素
例子：矩阵链相乘

以最短路径为例说明动态规划算法
的设计思想及必要条件

动态规划 算法的例子

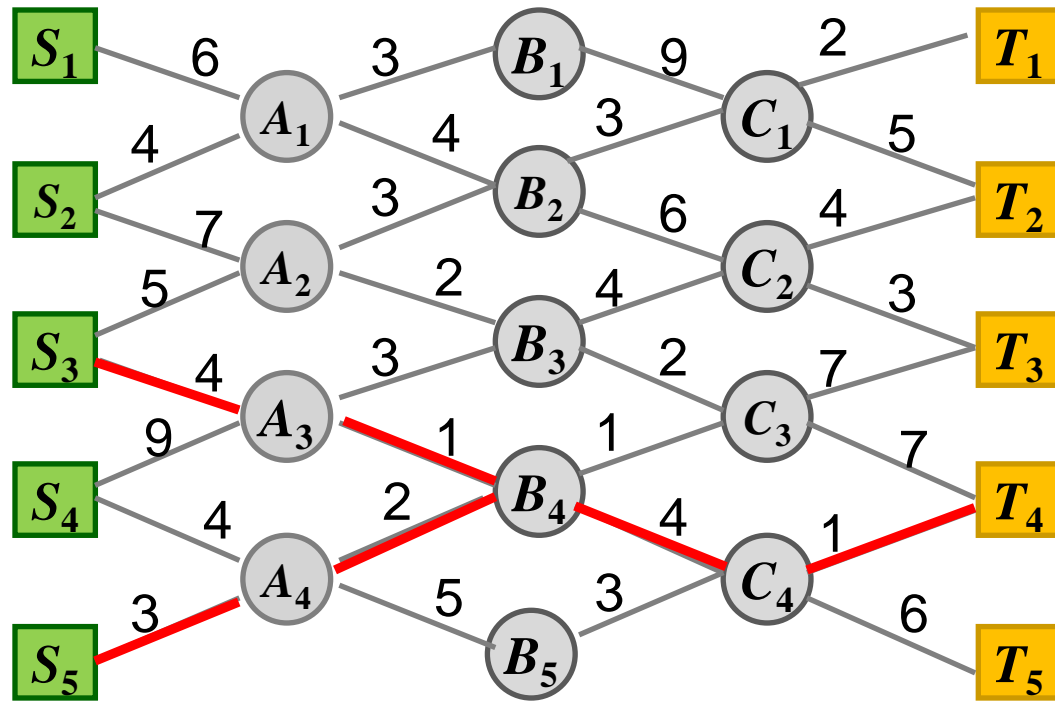
最短路径问题

问题:

输入: 起点集合 $\{S_1, S_2, \dots, S_n\}$,
终点集合 $\{T_1, T_2, \dots, T_m\}$,
中间结点集,
边集 E , 对于任意边 e 有长度

输出: 一条从起点到终点的最短路

一个实例



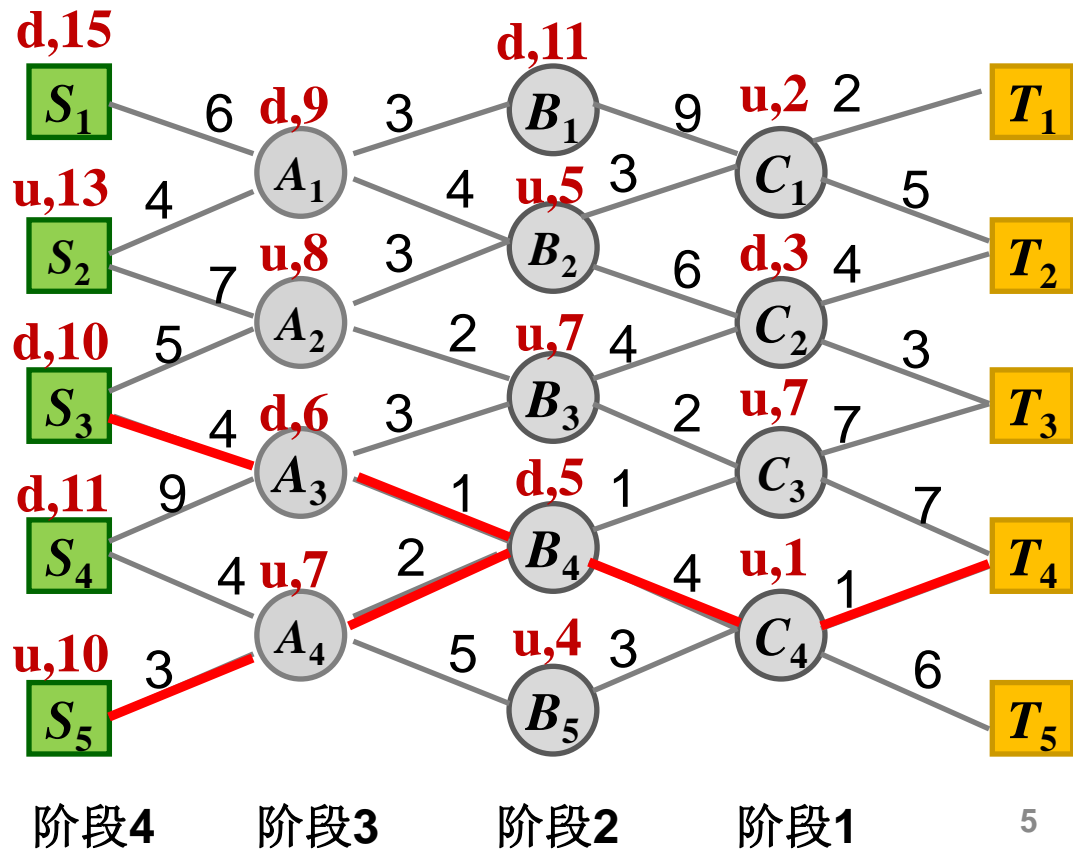
算法设计

蛮力算法：考察每一条从某个起点到某个终点的路径，计算长度，从其中找出最短路径。

在上述实例中，如果网络的层数为 k ，那么路径条数将接近于 2^k

动态规划算法：多阶段决策过程。每步求解的问题是后面阶段求解问题的子问题。每步决策将依赖于以前步骤的决策结果。

动态规划求解



子问题界定

后边界不变, 前边界前移

 决策 1

  决策 2

   决策 3

    决策 4

S_i A_j B_k C_l T_m

最短路长的依赖关系

$$\underline{F(C_l)} = \min_m \{C_l T_m\} \quad \text{决策 1}$$

$$F(B_k) = \min_l \{B_k C_l + \underline{F(C_l)}\} \quad \text{决策 2}$$

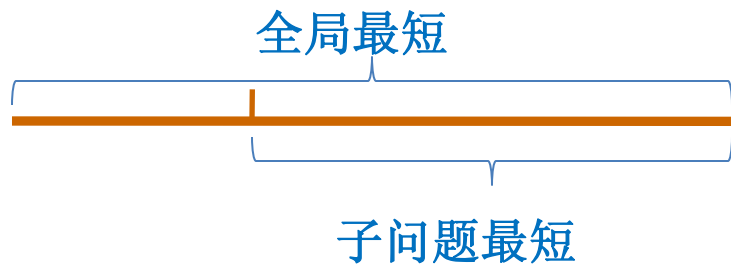
$$F(A_j) = \min_k \{A_j B_k + F(B_k)\} \quad \text{决策 3}$$

$$F(S_i) = \min_j \{S_i A_j + F(A_j)\} \quad \text{决策 4}$$

优化函数值之间存在依赖关系

优化原则:最优子结构性质

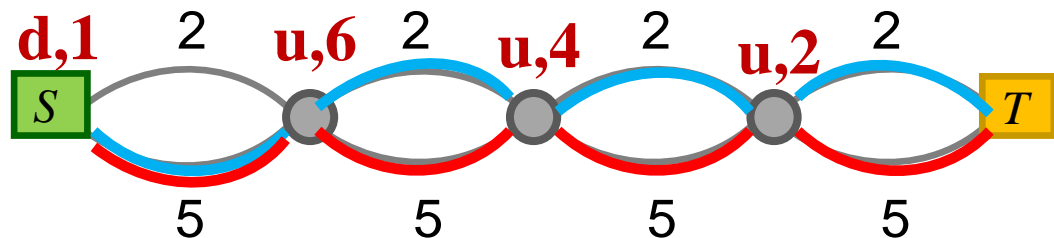
- **优化函数的特点:** 任何最短路的子路径相对于子问题始、终点最短



- **优化原则:** 一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优决策序列

一个反例

求总长模10的最小路径



动态规划算法的解：下，上，上，上

最优解：下，下，下，下

不满足优化原则，不能用动态规划

小结

动态规划(Dynamic Programming)

- 求解过程是多阶段决策过程，每步处理一个子问题，可用于求解组合优化问题
- 适用条件：问题要满足优化原则或最优子结构性质，即：一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优决策序列

动态规划 算法设计

动态规划设计要素

1. 问题建模，优化的目标函数是什么？约束条件是什么？
2. 如何划分子问题（边界）？
3. 问题的优化函数值与子问题的优化函数值存在着什么依赖关系？
(递推方程)
4. 是否满足优化原则？
5. 最小子问题怎样界定？其优化函数值，即初值等于什么？

矩阵链相乘

问题： 设 A_1, A_2, \dots, A_n 为矩阵序列， A_i 为 $P_{i-1} \times P_i$ 阶矩阵， $i = 1, 2, \dots, n$. 试确定矩阵的乘法顺序，使得元素相乘的总次数最少.

输入： 向量

$$P = \langle P_0, P_1, \dots, P_n \rangle,$$

其中 P_0, P_1, \dots, P_n 为 n 个矩阵的行数与列数

输出： 矩阵链乘法加括号的位置.

矩阵相乘基本运算次数

矩阵A: i 行 j 列, B : j 行 k 列, 以元素相乘作基本运算, 计算 AB 的工作量

$$\begin{bmatrix} \cdots \\ a_{t1} & a_{t2} & \cdots & a_{tj} \\ \cdots \end{bmatrix} \begin{bmatrix} \vdots \\ b_{1s} \\ b_{2s} \\ \vdots \\ b_{js} \end{bmatrix} = \begin{bmatrix} \vdots \\ c_{ts} \\ \vdots \end{bmatrix}$$

$$c_{ts} = a_{t1}b_{1s} + a_{t2}b_{2s} + \cdots + a_{tj}b_{js}$$

AB : i 行 k 列, 计算每个元素需要做 j 次乘法, 总计乘法次数 为 ijk

实例

实例: $P = \langle 10, 100, 5, 50 \rangle$

$A_1: 10 \times 100$, $A_2: 100 \times 5$, $A_3: 5 \times 50$,

乘法次序

$(A_1 A_2) A_3$: $10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$

$A_1 (A_2 A_3)$: $10 \times 100 \times 50 + 100 \times 5 \times 50 = 75000$

第一种次序计算次数最少.

蛮力算法

加 n 个括号的方法有 $\frac{1}{n+1}\binom{2n}{n}$ 种，
是一个Catalan数，是指数级别

$$W(n) = \Omega\left(\frac{1}{n+1} \frac{(2n)!}{n!n!}\right)$$

代入
Stirling
公式

$$= \Omega\left(\frac{1}{n+1} \frac{\sqrt{2\pi 2n} \left(\frac{2n}{e}\right)^{2n}}{\sqrt{2\pi n} \left(\frac{n}{e}\right)^n \sqrt{2\pi n} \left(\frac{n}{e}\right)^n}\right) = \Omega(2^{2n} / n^{\frac{3}{2}})$$

动态规划算法

- 子问题划分

$A_{i..j}$: 矩阵链 $A_i A_{i+1} \dots A_j$, 边界 i, j

输入向量: $\langle P_{i-1}, P_i, \dots, P_j \rangle$

其最好划分的运算次数: $m[i, j]$

- 子问题的依赖关系

最优划分最后一次相乘发生在矩阵 k 的位置, 即

$$A_{i..j} = A_{i..k} A_{k+1..j}$$

$A_{i..j}$ 最优运算次数依赖于 $A_{i..k}$ 与 $A_{k+1..j}$ 的最优运算次数

优化函数的递推方程

递推方程:

$m[i,j]$: 得到 $A_{i..j}$ 的最少的相乘次数

$m[i,j]$

$$= \begin{cases} 0 & i = j \\ \min_{i \leq k < j} \{ \underline{m[i,k]} + \underline{m[k+1,j]} + P_{i-1}P_kP_j \} & i < j \end{cases}$$

A_i	\dots	A_k	A_{k+1}	\dots	A_j
$P_{i-1} \times P_k$			$P_k \times P_j$		

该问题满足优化原则

小结

动态规划算法设计要素

- 多阶段决策过程，每步处理一个子问题，界定子问题的边界
- 列出优化函数的递推方程及初值
- 问题要满足优化原则或最优子结构性质，即：一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优决策序列

动态规划算法 的递归实现

部分伪码

子问题 $i-j$

算法1 RecurMatrixChain (P, i, j)

1. $m[i,j] \leftarrow \infty$
2. $s[i,j] \leftarrow i$
3. for $k \leftarrow i$ to $j-1$ do
4. $q \leftarrow$ RecurMatrixChain(P, i, k)
 + RecurMatrixChain($P, k+1, j$) + $p_{i-1}p_kp_j$
5. if $q < m[i,j]$
6. then $m[i,j] \leftarrow q$
7. $s[i,j] \leftarrow k$
8. return $m[i,j]$

划分位置 k

找到更好的解

这里没有写出算法的全部描述（递归边界）

算法分析

时间复杂度的递推方程

$$T(n) \geq \begin{cases} O(1) & n = 1 \\ \sum_{k=1}^{n-1} (T(k) + T(n-k) + O(1)) & n > 1 \end{cases}$$

$$T(n) \geq O(n) + \sum_{k=1}^{n-1} T(k) + \sum_{k=1}^{n-1} T(n-k)$$

$$T(n) \geq O(n) + 2 \sum_{k=1}^{n-1} T(k)$$

时间复杂度

数学归纳法证明: $T(n) \geq 2^{n-1}$

$n=2$, 显然为真.

假设对于任何小于 n 的 k , 命题为真

$$T(n) \geq O(n) + 2 \sum_{k=1}^{n-1} T(k)$$

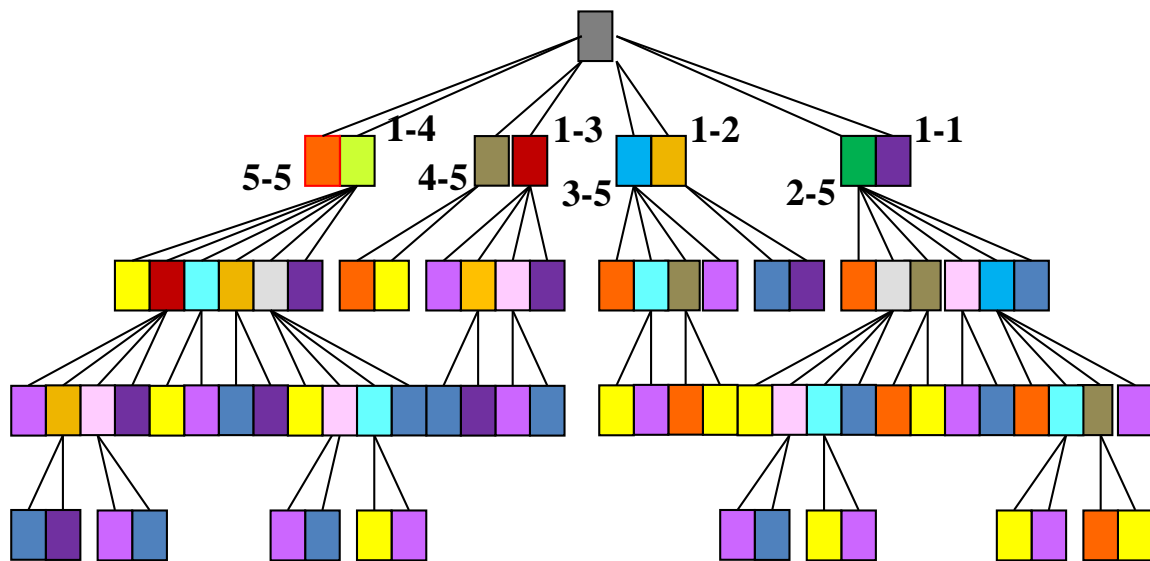
代入归纳假设

$$\geq O(n) + 2 \sum_{k=1}^{n-1} 2^{k-1}$$

等比数列求和

$$= O(n) + 2(2^{n-1} - 1) \geq 2^{n-1}$$

子问题的产生, $n=5$



划分: $A_{1..4}A_{5..5}$, $A_{1..3}A_{4..5}$, $A_{1..2}A_{3..5}$, $A_{1..1}A_{2..5}$

产生 8 个子问题, 即第一层的 8 个结点.

子问题的计数

边界	次数	边界	次数	边界	次数
1-1	8	1-2	4	2-4	2
2-2	12	2-3	5	3-5	2
3-3	14	3-4	5	1-4	1
4-4	12	4-5	4	2-5	1
5-5	8	1-3	2	1-5	1

边界不同的子问题：15 个

递归计算的子问题：81 个

结论

- 与蛮力算法相比较，动态规划算法利用了子问题优化函数间的依赖关系，时间复杂度有所降低
- 动态规划算法的递归实现效率不高，原因在于同一子问题多次重复出现，每次出现都需要重新计算一遍。
- 采用空间换时间策略，记录每个子问题首次计算结果，后面再用时就直接取值，每个子问题只算一次。

动态规划算法 的迭代实现

迭代计算的关键

- 每个子问题只计算一次
- 迭代过程
 - 从最小的子问题算起
 - 考虑计算顺序，以保证后面用到的值前面已经计算好
 - 存储结构保存计算结果——备忘录
- 解的追踪
 - 设计标记函数标记每步的决策
 - 考虑根据标记函数追踪解的算法

矩阵链乘法不同子问题

长度1: 只含1个矩阵, 有 n 个子问题
(不需要计算)

长度2: 含2个矩阵, $n-1$ 个子问题

长度3: 含3个矩阵, $n-2$ 个子问题

...

长度 $n-1$: 含 $n-1$ 个矩阵, 2个子问题

长度 n : 原始问题, 只有1个

矩阵链乘法迭代顺序

长度为1: 初值, $m[i, i] = 0$

长度为2: 1..2, 2..3, 3..4, ... , $n-1..n$

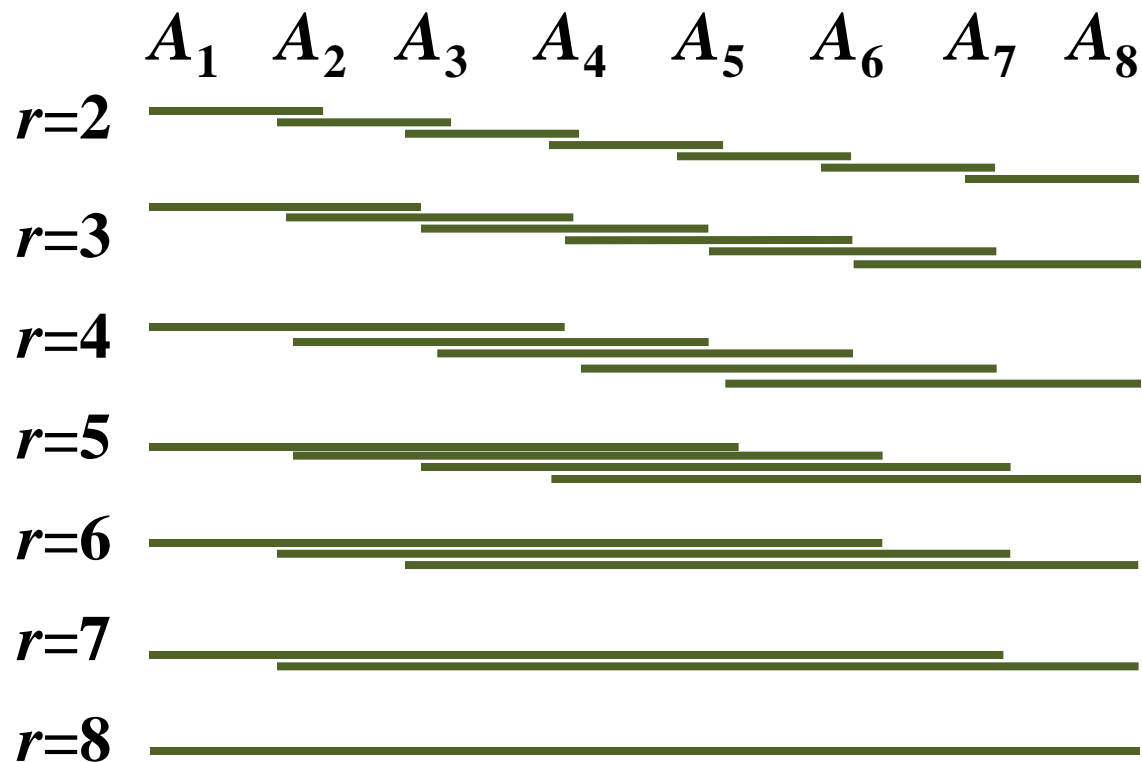
长度为3: 1..3, 2..4, 3..5, ... , $n-2..n$

...

长度为 $n-1$: 1.. $n-1$, 2.. n

长度为 n : 1.. n

$n=8$ 的子问题计算顺序



算法 **MatrixChain** (P, n)

1. 令所有的 $m[i,j]$ 初值为0
2. for $r \leftarrow 2$ to n do // r 为链长
3. for $i \leftarrow 1$ to $n-r+1$ do // 左边界 i
4. $j \leftarrow i+r-1$ // 右边界 j
5. $m[i,j] \leftarrow m[i+1,j] + p_{i-1}p_ip_j$ // $k=i$
6. $s[i,j] \leftarrow i$ // 记录 k
7. for $k \leftarrow i+1$ to $j-1$ do // 遍历 k
8. $t \leftarrow \underline{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j}$
9. if $t < m[i,j]$
10. then $m[i,j] \leftarrow t$ // 更新解
11. $s[i,j] \leftarrow k$

遍历
长 r 子
问题

遍历所
有划分

二维数组 m 与 s 为备忘录

时间复杂度

- **根据伪码**：行 2, 3, 7 都是 $O(n)$ ，循环执行 $O(n^3)$ 次，内部为 $O(1)$

$$W(n) = O(n^3)$$

- **根据备忘录**：估计每项工作量，求和。
子问题有 $O(n^2)$ 个，确定每个子问题的最少乘法次数需要对不同划分位置比较，需要 $O(n)$ 时间。

$$W(n) = O(n^3)$$

- 追踪解工作量 $O(n)$ ，总工作量 $O(n^3)$.

实例

输入: $P = \langle 30, 35, 15, 5, 10, 20 \rangle$,
 $n = 5$

矩阵链: $A_1 A_2 A_3 A_4 A_5$, 其中

$A_1: 30 \times 35$, $A_2: 35 \times 15$, $A_3: 15 \times 5$,

$A_4: 5 \times 10$, $A_5: 10 \times 20$

备忘录: 存储所有子问题的最小乘法次数及得到这个值的划分位置.

备忘录 $m[i,j]$ $P = \langle 30, \underline{35}, 15, 5, \underline{10}, 20 \rangle$

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

$$m[2,5] = \min\{ 0+2500+35 \times 15 \times 20, 2625+1000+35 \times 5 \times 20, \\ 4375+0+35 \times 10 \times 20 \} = 7125$$

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: $s[1,5]=3 \Rightarrow (A_1A_2A_3)(A_4A_5)$

$s[1,3]=1 \Rightarrow A_1(A_2A_3)$

输出

计算顺序: $(A_1(A_2A_3))(A_4A_5)$

最少的乘法次数: $m[1,5]=11875$

两种实现的比较

递归实现：时间复杂性高，空间较小

迭代实现：时间复杂性低，空间消耗多

原因：递归实现子问题多次重复计算，子问题计算次数呈指数增长. 迭代实现每个子问题只计算一次.

动态规划时间复杂度：

备忘录各项计算量之和 + 追踪解工作量

通常追踪工作量不超过计算工作量，是问题规模的多项式函数

动态规划算法的要素

- 划分子问题，确定子问题边界，将问题求解转 变成多步判断的过程。
- 定义优化函数,以该函数极大(或极小)值作为依据,确定是否满足优化原则。
- 列优化函数的递推方程和边界条件
- 自底向上计算，设计备忘录 (表格)
- 考虑是否需要设立标记函数
- 用递推方程或备忘录估计时间复杂度

投资问题

投资问题的建模

问题： m 元钱， n 项投资， $f_i(x)$: 将 x 元投入第 i 个项目的效益. 求使得总效益最大的投资方案.

建模：

问题的解是向量 $\langle x_1, x_2, \dots, x_n \rangle$,

x_i 是投给项目 i 的钱数, $i = 1, 2, \dots, n$.

目标函数 $\max\{f_1(x_1)+f_2(x_2)+\dots+f_n(x_n)\}$

约束条件 $x_1+x_2+\dots+x_n=m, x_i \in \mathbb{N}$

实例

- 实例：5万元钱，4个项目
效益函数如下表所示

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	0	0	0
1	11	0	2	20
2	12	5	10	21
3	13	10	30	22
4	14	15	32	23
5	15	20	40	24

子问题界定和计算顺序

子问题界定：由参数 k 和 x 界定

k ：考虑对项目 $1, 2, \dots, k$ 的投资

x ：投资总钱数不超过 x



这两个参数与矩阵链相乘问题的参数有什么区别？

原始输入： $k = n, x = m$

子问题计算顺序：

$k = 1, 2, \dots, n$

对于给定的 $k, x = 1, 2, \dots, m$

优化函数的递推方程

$F_k(x)$: x 元钱投给前 k 个项目最大效益

多步判断: 若知道 p 元钱 ($p \leq x$) 投给前 $k-1$ 个项目的最大效益 $F_{k-1}(p)$, 确定 x 元钱投给前 k 个项目的方案

递推方程和边界条件

$$F_k(x) = \max_{0 \leq x_k \leq x} \{f_k(x_k) + F_{k-1}(x - x_k)\} \quad k > 1$$

$$F_1(x) = f_1(x)$$

$k=1$ 时实例的计算

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	0	0	0
1	11	0	2	20
2	12	5	10	21
3	13	10	30	22
4	14	15	32	23
5	15	20	40	24

$k=1$ 为初值

$F_1(1)=11, F_1(2)=12, F_1(3)=13,$

$F_1(4)=14, F_1(5)=15,$

$k=2$ 时实例计算

方案(项目2,其他): (1,0), (0,1)

$$F_2(1) = \max\{f_1(1), f_2(1)\} = 11$$

x	$f_1(x)$	$f_2(x)$
0	0	0
1	11	0
2	12	5
3	13	10
4	14	15
5	15	20

方案: (2,0), (1,1), (0,2)

$$F_2(2) = \max\{f_2(2), F_1(1) + f_2(1), F_1(2)\} = 12$$

方案: (3,0), (2,1), (1,2), (0,3)

$$F_2(3) = \max\{f_2(3), F_1(1) + f_2(2), F_1(2) + f_2(1), F_1(3)\} = 16$$

类似地计算

$$F_2(4) = 21, F_2(5) = 26$$

备忘录和解

x	$F_1(x) \ x_1(x)$	$F_2(x) \ x_2(x)$	$F_3(x) \ x_3(x)$	$F_4(x) \ x_4(x)$
1	11 1	11 0	11 0	20 1
2	12 2	12 0	13 1	31 1
3	13 3	16 2	30 3	33 1
4	14 4	21 3	41 3	50 1
5	15 5	26 4	43 4	61 1

$$x_4(5)=1 \Rightarrow x_4=1, \ x_3(5-1) = x_3(4)$$

$$x_3(4)=3 \Rightarrow x_3=3, \ x_2(4-3) = x_2(1)$$

$$x_2(1)=0 \Rightarrow x_2=0, \ x_1(1-0) = x_1(1)$$

$$x_1(1)=1 \Rightarrow x_1=1$$

解: $x_1=1, x_2=0, x_3=3, x_4=1, F_4(5) = 61$

时间复杂度分析

备忘录表中有 m 行 n 列, 共计 mn 项

$$F_k(x) = \max_{0 \leq x_k \leq x} \{f_k(x_k) + F_{k-1}(x - x_k)\} \quad k > 1$$

$$F_1(x) = f_1(x)$$

x_k 有 $x+1$ 种可能的取值, 计算 $F_k(x)$ 项 ($2 \leq k \leq n, 1 \leq x \leq m$) 需要:

$x+1$ 次加法

x 次比较

时间复杂度分析

对备忘录中所有的项求和：

$$\text{加法次数} \sum_{k=2}^n \sum_{x=1}^m (x+1) = \frac{1}{2}(n-1)m(m+3)$$

$$\text{比较次数} \sum_{k=2}^n \sum_{x=1}^m x = \frac{1}{2}(n-1)m(m+1)$$

$$W(n) = O(nm^2)$$

小结

投资问题的动态规划算法

- 用两个不同类型的参数界定子问题
- 优化函数的递推方程及初值
- 根据备忘录中项的计算估计时间复杂度
- 时间复杂度为： $O(nm^2)$

动态规划算法 解背包问题

背包问题 (Knapsack Problem)

一个旅行者随身携带一个背包. 可以放入背包的物品有 n 种, 每种物品的重量和价值分别为 w_i, v_i . 如果背包的最大重量限制是 b , 每种物品可以放多个. 怎样选择放入背包的物品以使得背包的价值最大? 不妨设上述 w_i, v_i, b 都是正整数.

实例: $n = 4, b = 10$

$$v_1 = 1, \quad v_2 = 3, \quad v_3 = 5, \quad v_4 = 9,$$

$$w_1 = 2, \quad w_2 = 3, \quad w_3 = 4, \quad w_4 = 7, \quad 2$$

建模

解是 $\langle x_1, x_2, \dots, x_n \rangle$, 其中 x_i 是装入背包的第 i 种物品个数

$$\text{目标函数} \quad \max \sum_{i=1}^n v_i x_i$$

$$\text{约束条件} \quad \sum_{i=1}^n w_i x_i \leq b, \quad x_i \in \mathbb{N}$$

线性规划问题: 由线性条件约束的线性函数取最大或最小的问题

整数规划问题: 线性规划问题的变量 x_i 都是非负整数

子问题界定和计算顺序

子问题界定：由参数 k 和 y 界定

k : 考虑对物品 $1, 2, \dots, k$ 的选择

y : 背包总重量不超过 y

原始输入: $k = n, y = b$

子问题计算顺序:

$k = 1, 2, \dots, n$

对于给定的 $k, y = 1, 2, \dots, b$

优化函数的递推方程

$F_k(y)$: 装前 k 种物品, 总重不超过 y ,
背包达到的最大价值

$$F_k(y) = \max\{F_{k-1}(y), F_k(y - w_k) + v_k\}$$

$$F_0(y) = 0, \quad 0 \leq y \leq b, \quad F_k(0) = 0, \quad 0 \leq k \leq n$$

$$F_1(y) = \left\lfloor \frac{y}{w_1} \right\rfloor v_1, \quad F_k(y) = -\infty \quad y < 0$$



类似于投资问题, 如何写递推方程
这两种写法有什么区别

$$F_k(y) = \max_{0 \leq x_k \leq \lfloor y/w_k \rfloor} \{F_{k-1}(y - x_k w_k) + x_k v_k\}$$

标记函数

$i_k(y)$: 装前 k 种物品, 总重不超 y , 背包达到最大价值时装入物品的最大标号

$$i_k(y) = \begin{cases} i_{k-1}(y) & F_{k-1}(y) > F_k(y - w_k) + v_k \\ k & F_{k-1}(y) \leq F_k(y - w_k) + v_k \end{cases}$$

$$i_1(y) = \begin{cases} 0 & y < w_1 \\ 1 & y \geq w_1 \end{cases}$$

实例

输入: $v_1=1, v_2=3, v_3=5, v_4=9,$
 $w_1=2, w_2=3, w_3=4, w_4=7,$
 $b=10$

$F_k(y)$ 的计算表如下:

$k \backslash y$	1	2	3	4	5	6	7	8	9	10
1	0	1	1	2	2	3	3	4	4	5
2	0	1	3	3	4	6	6	7	9	9
3	0	1	3	5	5	6	8	10	10	11
4	0	1	3	5	5	6	9	10	10	12

追踪解

$k \backslash y$	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	1	1	1	1	1	1
2	0	1	2	2	2	2	2	2	2	2
3	0	1	2	3	3	3	3	3	3	3
4	0	1	2	3	3	3	4	3	4	4

$$i_4(10)=4 \Rightarrow x_4 \geq 1$$

$$i_4(10 - w_4) = i_4(3) = 2 \Rightarrow x_2 \geq 1, x_4 = 1, x_3 = 0$$

$$i_2(3 - w_2) = i_2(0) = 0 \Rightarrow x_2 = 1, x_1 = 0$$

解 $x_1=0, x_2=1, x_3=0, x_4=1$, 价值12

追踪算法

算法 Track Solution

输入: $i_k(y)$ 表, $k=1,2,\dots,n$, $y=1,2,\dots,b$

输出: x_1, x_2, \dots, x_n , n 种物品的装入量

1. for $k \leftarrow 1$ to n do $x_k \leftarrow 0$

2. $y \leftarrow b$, $k \leftarrow n$

3. $j \leftarrow i_k(y)$

初始追踪位置

4. $x_k \leftarrow 1$

5. $y \leftarrow y - w_k$

6. while $i_k(y) = k$ do

继续放 k
种物品

7. $y \leftarrow y - w_k$

8. $x_k \leftarrow x_k + 1$

9. if $i_k(y) \neq 0$ then goto 4

继续追
下一种

时间复杂度 $O(nb)$

根据公式

$$F_k(y) = \max\{F_{k-1}(y), F_k(y - w_k) + v_k\}$$

备忘录需计算 nb 项，每项常数时间，
计算时间为 $O(nb)$ 。

伪多项式时间算法：时间为参数 b 和 n 的多项式，不是输入规模的多项式。
参数 b 是整数，表达 b 需要 $\log b$ 位，
输入规模是 $\log b$ 。

背包问题的推广

物品数受限背包：第 i 种物品最多用 n_i 个

0-1背包问题： $x_i = 0, 1, i = 1, 2, \dots, n$

多背包问题： m 个背包，背包 j 装入最大重量 $B_j, j = 1, 2, \dots, m$. 在满足所有背包重量约束条件下使装入物品价值最大.

二维背包问题：每件物品有重量 w_i 和体积 $t_i, i = 1, 2, \dots, n$, 背包总重不超过 b , 体积不超过 V , 如何选择物品以得到最大价值.

小结

- 划分子问题，确定子问题边界，将问题求解转变成多步判断的过程.
- 定义优化函数,以该函数极大(或极小)值作为依据,确定是否满足优化原则.
- 列优化函数的递推方程和边界条件
- 自底向上计算，设计备忘录 (表格)
- 考虑是否需要设立标记函数
- 时间复杂度估计

最长公共子序列

子序列

设序列 X, Z ,

$$X = \langle x_1, x_2, \dots, x_m \rangle$$

$$Z = \langle z_1, z_2, \dots, z_k \rangle$$

若存在 X 的元素构成的严格递增序列
使得

$$z_j = x_{i_j}, j = 1, 2, \dots, k$$

则称 Z 是 X 的子序列

X 与 Y 的公共子序列 Z : Z 是 X 和 Y
的子序列

子序列的长度：子序列的元素个数

最长公共子序列

问题：给定序列

$$X = \langle x_1, x_2, \dots, x_m \rangle$$

$$Y = \langle y_1, y_2, \dots, y_n \rangle$$

求 X 和 Y 的最长公共子序列

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

蛮力算法

不妨设 $m \leq n$, $|X| = m$, $|Y| = n$

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：

每个子序列 $O(n)$ 时间

X 有 2^m 个子序列

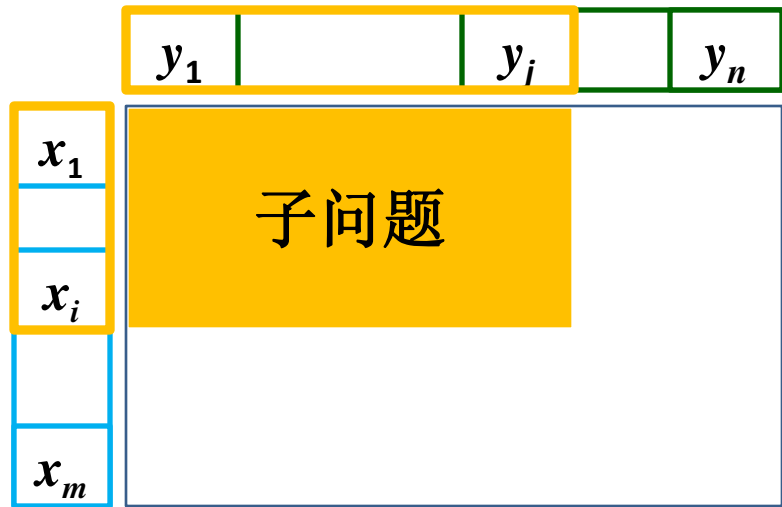
最坏情况下时间复杂度： $O(n 2^m)$

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

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



子问题间的依赖关系

设 $X=\langle x_1, x_2, \dots, x_m \rangle$, $Y=\langle y_1, y_2, \dots, y_n \rangle$,
 $Z=\langle z_1, z_2, \dots, z_k \rangle$ 为 X 和 Y 的 LCS, 那么

(1) 若 $x_m = y_n \Rightarrow z_k = x_m = y_n$, 且

Z_{k-1} 是 X_{m-1} 与 Y_{n-1} 的 LCS;

(2) 若 $x_m \neq y_n$, $z_k \neq x_m \Rightarrow$

Z 是 X_{m-1} 与 Y 的 LCS;

(3) 若 $x_m \neq y_n$, $z_k \neq y_n \Rightarrow$

Z 是 X 与 Y_{n-1} 的 LCS.

满足优化原则和子问题重叠性

优化函数的递推方程

令 X 与 Y 的子序列

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

$C[i,j]$: X_i 与 Y_j 的 LCS 的长度

$$C[i,j]$$

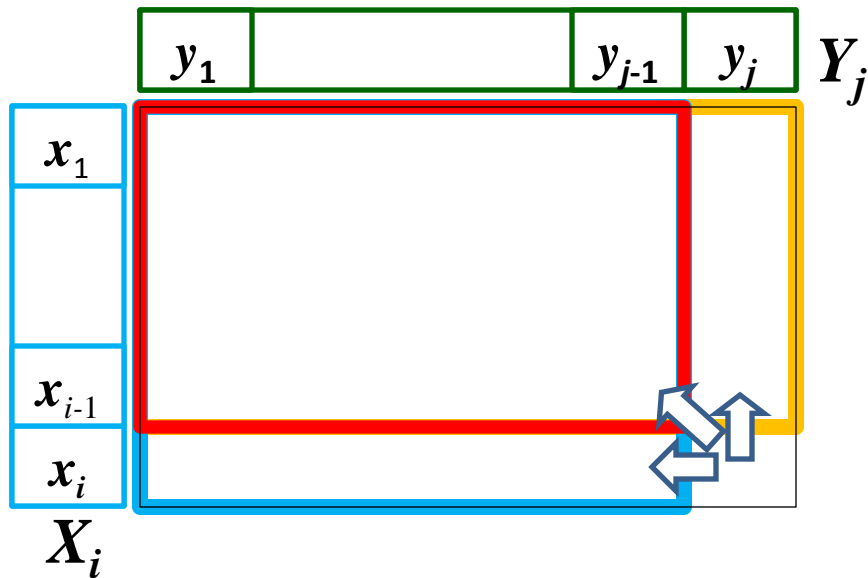
$$= \begin{cases} 0 & \text{若 } i=0 \text{ 或 } j=0 \\ C[i-1,j-1]+1 & \text{若 } i,j>0, x_i = y_j \\ \max\{C[i,j-1], C[i-1,j]\} & \text{若 } i,j>0, x_i \neq y_j \end{cases}$$

标记函数

标记函数: $B[i, j]$, 值为 \nwarrow 、 \leftarrow 、 \uparrow

$C[i, j] = C[i-1, j-1] + 1$: \nwarrow $C[i, j] = C[i, j-1]$: \leftarrow

$C[i, j] = C[i-1, j]$: \uparrow



伪码

算法 LCS (X, Y, m, n)

1. for $i \leftarrow 1$ to m do $C[i,0] \leftarrow 0$

初值

2. for $i \leftarrow 1$ to n do $C[0,i] \leftarrow 0$

3. for $i \leftarrow 1$ to m do

4. for $j \leftarrow 1$ to n do

子问题
 x_i, y_j

5. if $X[i]=Y[j]$

6. then $C[i,j] \leftarrow C[i-1,j-1]+1$

7. $B[i,j] \leftarrow "\nwarrow"$

8. else if $C[i-1,j] \geq C[i,j-1]$

9. then $C[i,j] \leftarrow C[i-1,j]$

10. $B[i,j] \leftarrow "\uparrow"$

11. else $C[i,j] \leftarrow C[i,j-1]$

12. $B[i,j] \leftarrow "\leftarrow"$

追踪解

算法 Structure Sequence(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 $B[i, j] = \text{“}\nwarrow\text{”}$
3. then 输出 $X[i]$
4. Structure Sequence($B, i-1, j-1$)
5. else if $B[i, j] = \text{“}\uparrow\text{”}$
6. then Structure Sequence ($B, i-1, j$)
7. else Structure Sequence ($B, i, j-1$)

标记函数的实例

输入: $X = \langle A, B, C, B, D, A, B \rangle$,
 $Y = \langle B, D, C, A, B, A \rangle$,

	1	2	3	4	5	6
1	$B[1,1]=\uparrow$	$B[1,2]=\uparrow$	$B[1,3]=\uparrow$	$B[1,4]=\nearrow$	$B[1,5]=\leftarrow$	$B[1,6]=\nwarrow$
2	$B[2,1]=\nwarrow$	$B[2,2]=\leftarrow$	$B[2,3]=\leftarrow$	$B[2,4]=\uparrow$	$B[2,5]=\nwarrow$	$B[2,6]=\leftarrow$
3	$B[3,1]=\uparrow$	$B[3,2]=\uparrow$	$B[3,3]=\nwarrow$	$B[3,4]=\leftarrow$	$B[3,5]=\uparrow$	$B[3,6]=\uparrow$
4	$B[4,1]=\uparrow$	$B[4,2]=\uparrow$	$B[4,3]=\uparrow$	$B[4,4]=\uparrow$	$B[4,5]=\nwarrow$	$B[4,6]=\leftarrow$
5	$B[5,1]=\uparrow$	$B[5,2]=\uparrow$	$B[5,3]=\uparrow$	$B[5,4]=\uparrow$	$B[5,5]=\uparrow$	$B[5,6]=\uparrow$
6	$B[6,1]=\uparrow$	$B[6,2]=\uparrow$	$B[6,3]=\uparrow$	$B[6,4]=\nwarrow$	$B[6,5]=\uparrow$	$B[6,6]=\nwarrow$
7	$B[7,1]=\uparrow$	$B[7,2]=\uparrow$	$B[7,3]=\uparrow$	$B[7,4]=\uparrow$	$B[7,5]=\uparrow$	$B[7,6]=\uparrow$

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 $\Theta(mn)$ 次，

循环体内常数运算，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： $\Theta(mn)$

小结

- 最长公共子序列问题的建模
- 子问题边界的界定
- 递推方程及初值，优化原则判定
- 伪码
- 标记函数与解的追踪
- 时间复杂度

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

Compress (P, n)

伪码

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

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

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

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 $center$

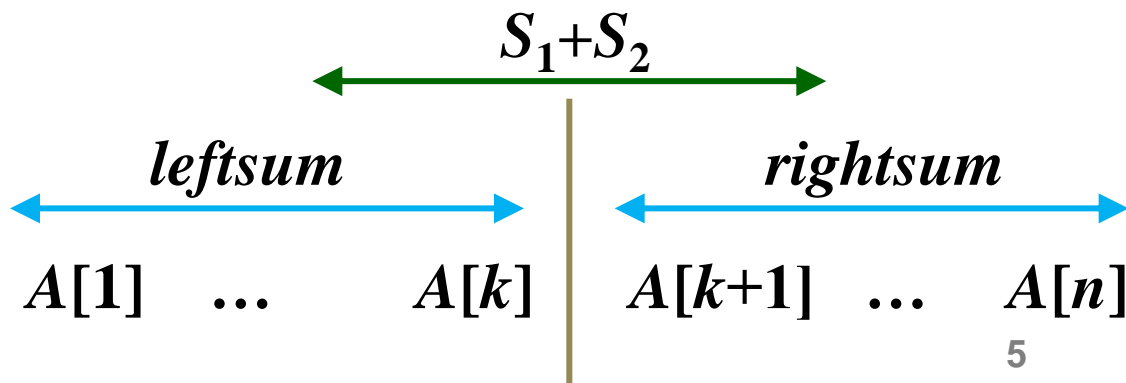
递归计算左段最大子段和 $leftsum$

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 $O(n)$

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

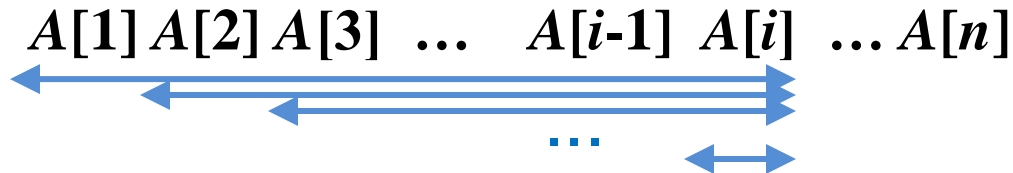
$$T(c) = O(1)$$

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

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

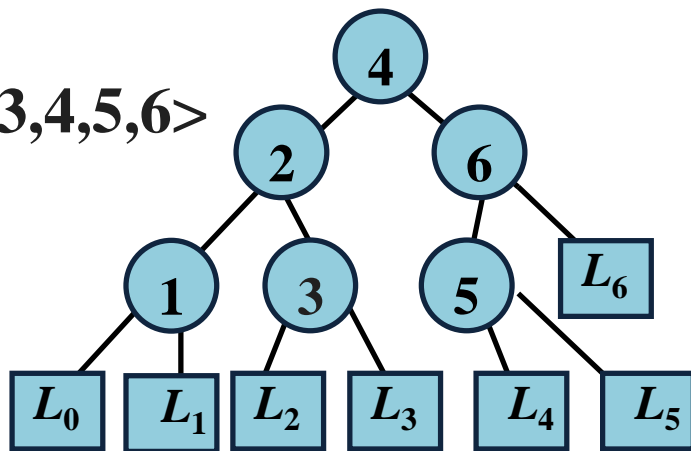
最优二叉检索树

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).

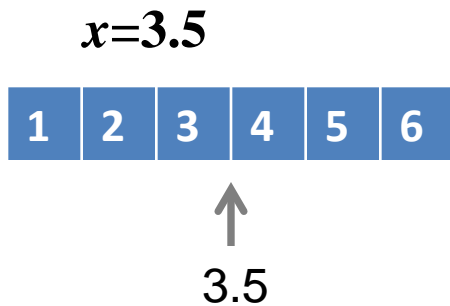
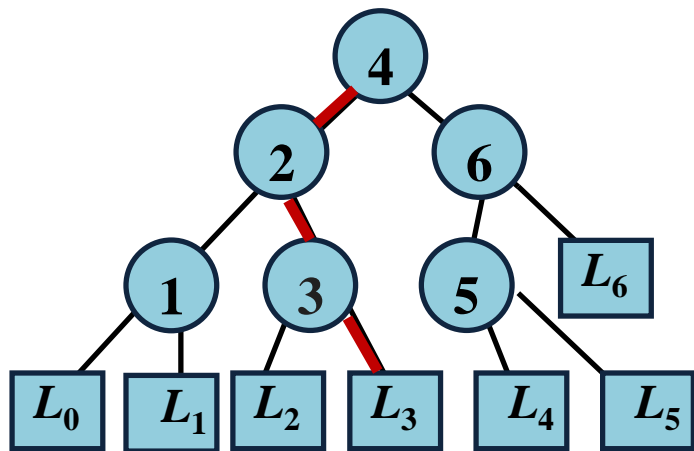
实例:

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



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:

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

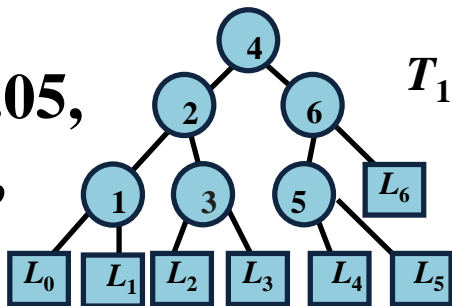
各个空隙的检索概率分别为:

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

检索数据的平均时间

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

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

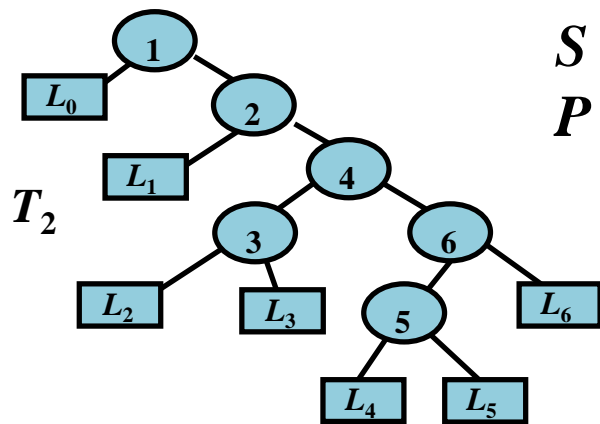


$m(T_1)$

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

$$= 1.8 + \mathbf{0.71} = \mathbf{\underline{2.51}}$$

检索数据的平均时间



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

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

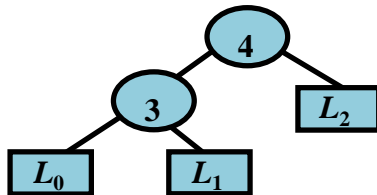
$m(T_2)$

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

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

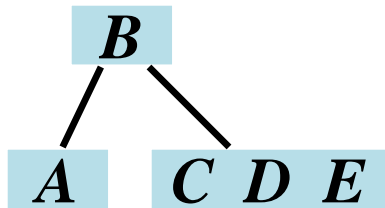
$$P[2, 4] = \langle 0.02, \mathbf{0.3}, 0.02, \mathbf{0.1}, 0.05, \mathbf{0.2}, 0.06 \rangle$$

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

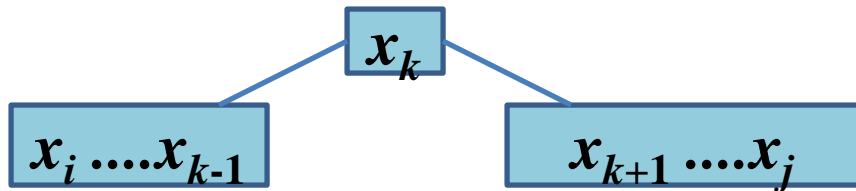
设 $m[i,j]$ 是相对于输入 $S[i,j]$ 和 $P[i,j]$ 的最优二叉搜索树的平均比较次数

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

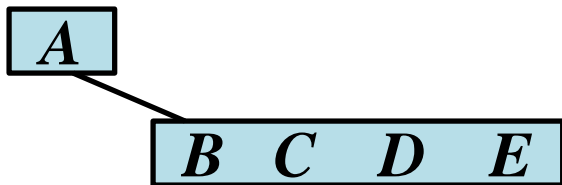
$$m[i, j] = \min \{ m[i, j]_k \mid i \leq k \leq j \}$$

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，

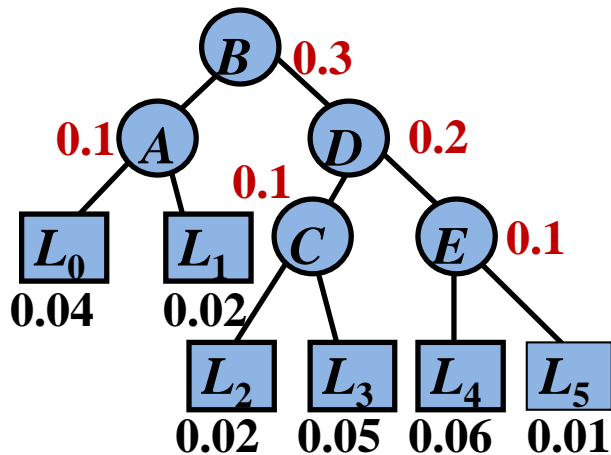
对应于： $S[1,0]$ ， $m[1,0]=0$ 的情况。



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: $T(n) = O(n^3)$

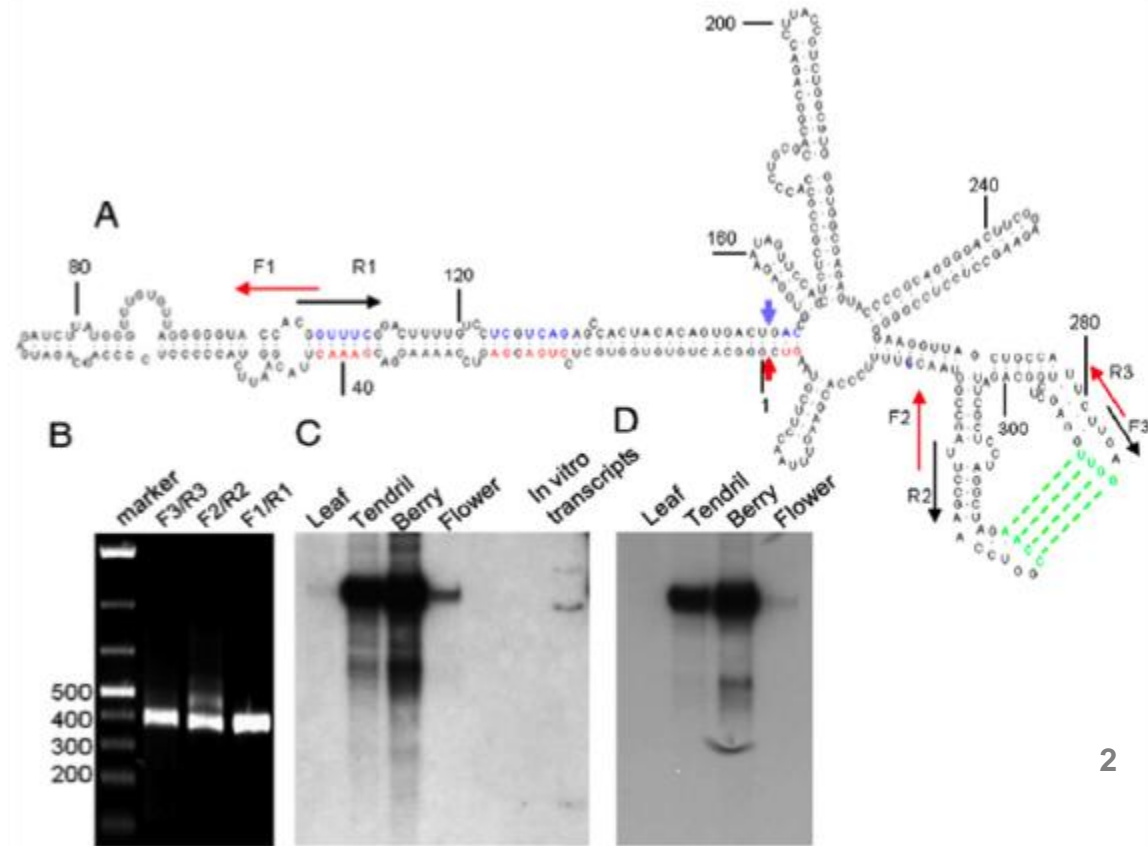
空间复杂度: $S(n) = O(n^2)$

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

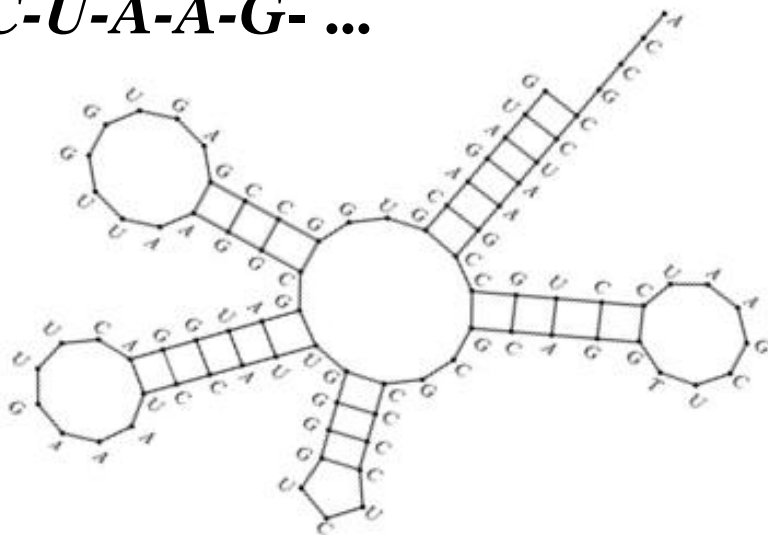


RNA二级结构

一级结构： 由字母 *A, C, G, U* 标记的核苷酸构成的一条链.

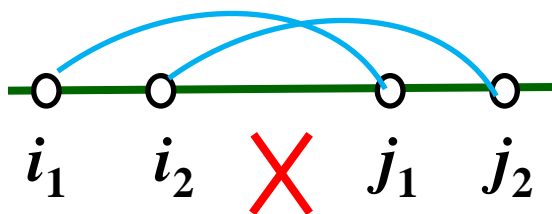
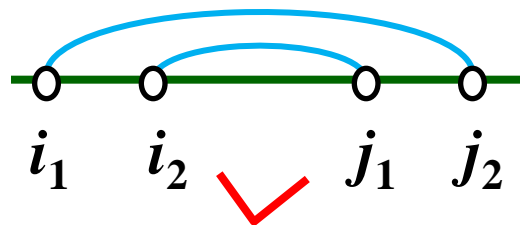
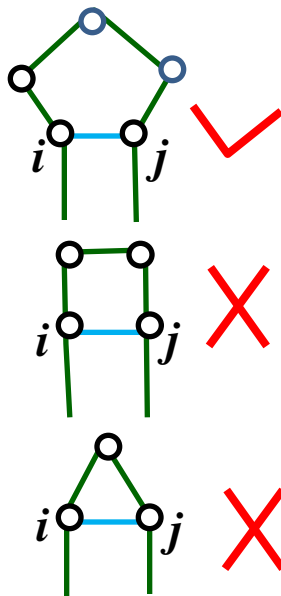
实例： *A-C-C-G-C-C-U-A-A-G-C-C-G-U-C-C-U-A-A-G- ...*

二级结构：
核苷酸相互
匹配构成的
平面结构

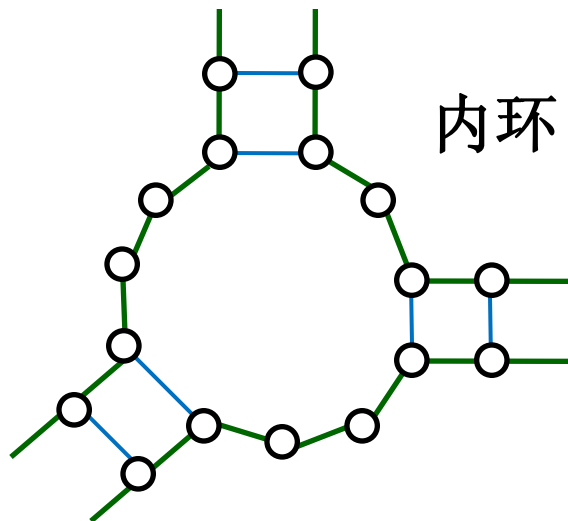
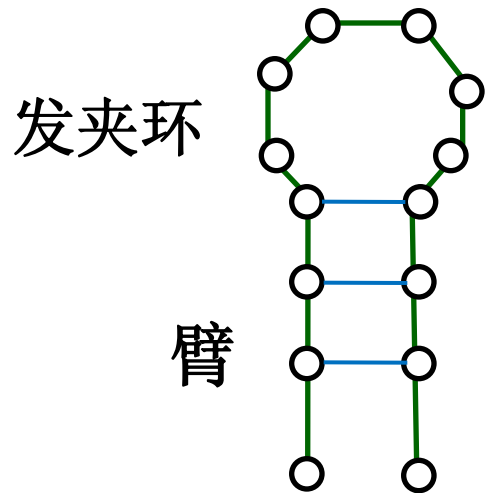


匹配原则

- 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

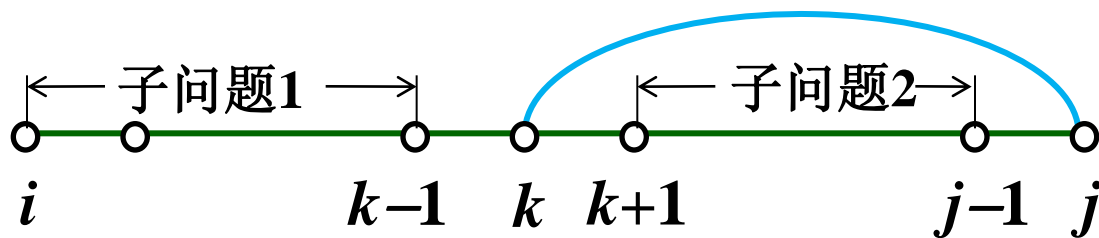
生物学条件：具有最小自由能

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 ≤ 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 $O(m \ n)$ 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 $O(n \ m)$

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。