

algorithm

5.动态规划b

本节内容

- 4.5 投资问题
- 4.6 背包问题
- 4.7 最长公共子序列
- 4.8 图像压缩
- 4.9 最大子段和

投资问题

- 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): $(1,0), (0,1)$

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

- 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

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

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

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:

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

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

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

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

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
- 目标函数 $\max \sum_{i=1}^n v_i x_i$
- 约束条件 $\sum_{i=1}^n w_i x_i \leq b, \quad x_i \in \mathbb{N}$
- 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?

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

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:

$k \ y$	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

$k y$	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. $j \leftarrow i_k(y)$
 4. $x_k \leftarrow 1$
 5. $y \leftarrow y - w_k$
 6. while $i_k(y) = k$ do
 7. $y \leftarrow y - w_k$
 8. $x_k \leftarrow x_k + 1$
 9. if $i_k(y) \neq 0$ then goto 4

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

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 $z_j = x_j, j = 1, 2, \dots, k$
- 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

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

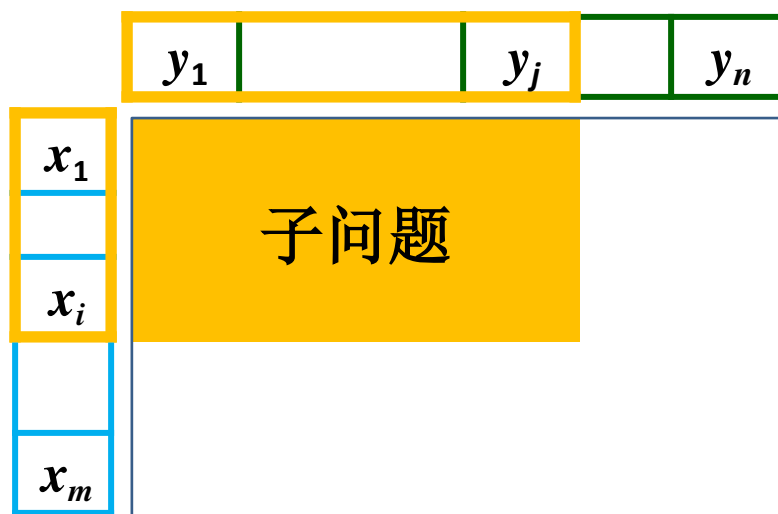
- 最长公共子序列: B C B A, 长度4

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

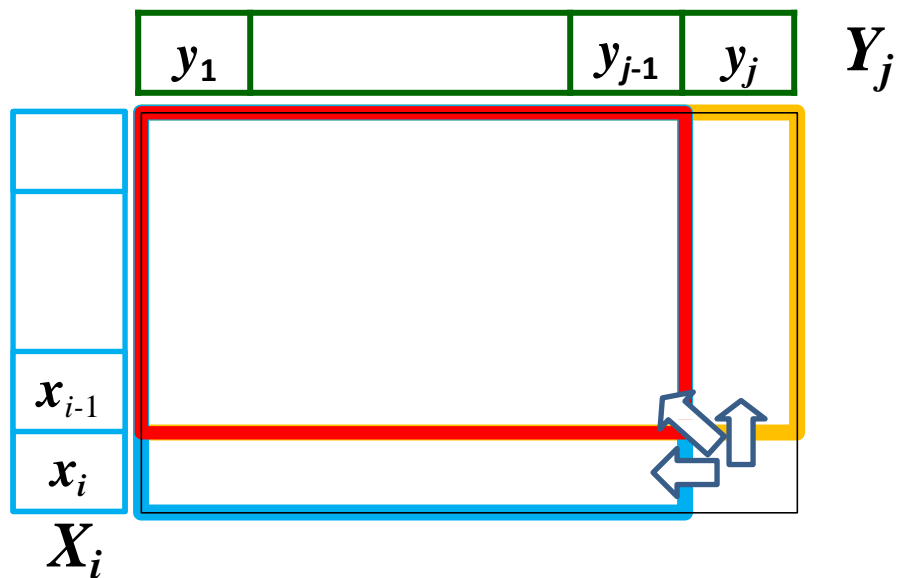
优化函数的递推方程

- 令 X 与 Y 的子序列
- $X_i = \langle x_1, x_2, \dots, x_i \rangle$, $Y_j = \langle y_1, y_2, \dots, y_j \rangle$
- $C[i, j]$: 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 $C[i, 0] \leftarrow 0$
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
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 $B[i, j] = “\searrow”$
 3. then 输出 $X[i]$
 4. Structure Sequence($B, i - 1, j - 1$)
 5. else if $B[i, j] = “\uparrow”$
 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	$B[1,1]=\uparrow$	$B[1,2]=\uparrow$	$B[1,3]=\uparrow$	$B[1,4]=\searrow$	$B[1,5]=\leftarrow$	$B[1,6]=\searrow$
2	$B[2,1]=\searrow$	$B[2,2]=\leftarrow$	$B[2,3]=\leftarrow$	$B[2,4]=\uparrow$	$B[2,5]=\searrow$	$B[2,6]=\leftarrow$
3	$B[3,1]=\uparrow$	$B[3,2]=\uparrow$	$B[3,3]=\searrow$	$B[3,4]=\leftarrow$	$B[3,5]=\uparrow$	$B[3,6]=\uparrow$
4	$B[4,1]=\uparrow$	$B[4,2]=\uparrow$	$B[4,3]=\uparrow$	$B[4,4]=\uparrow$	$B[4,5]=\searrow$	$B[4,6]=\leftarrow$
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	$B[6,1]=\uparrow$	$B[6,2]=\uparrow$	$B[6,3]=\uparrow$	$B[6,4]=\searrow$	$B[6,5]=\uparrow$	$B[6,6]=\searrow$
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： $\Theta(mn)$

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 $8n$
- 问题：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




$$\min_T \{ \sum (b[t] \times l[t] + 11) \},$$

$$T = \{S_1, S_2, \dots, S_m\} \text{ 为分段}$$

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

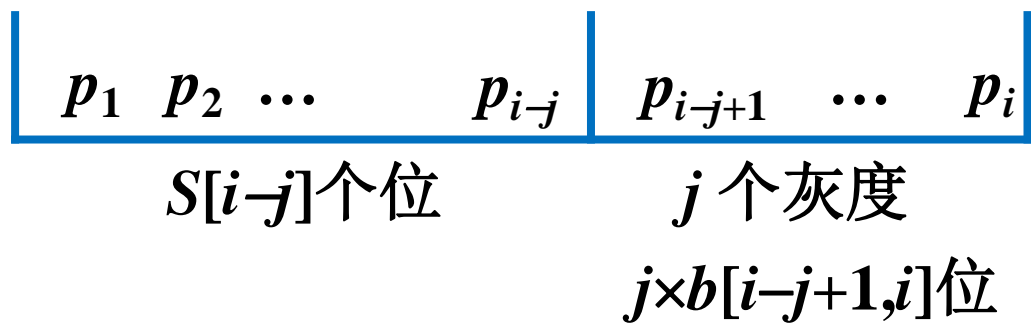
- 子问题前边界为1, 后边界为 i
- 对应像素序列为 $\langle p_1, p_2, \dots, p_i \rangle$
- 优化函数值 $S[i]$ 为最优分段存储位数
- 计算顺序
 - $i = 1$ 
 - $i = 2$ 
 - ...
 - $i = n$ 

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

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

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



伪码

1. Compress (P, n)
2. $Lmax \leftarrow 256; header \leftarrow 11; S[0] \leftarrow 0$
3. for $i \leftarrow 1$ to n do
4. $b[i] \leftarrow length(P[i])$
5. $bmax \leftarrow b[i]$
6. $S[i] \leftarrow S[i-1] + bmax$
7. $l[i] \leftarrow 1$
8. for $j \leftarrow 2$ to $\min\{i, Lmax\}$ do
9. if $bmax < b[i-j+1]$
10. then $bmax \leftarrow b[i-j+1]$
11. if $S[i] > S[i-j] + j * bmax$
12. then $S[i] \leftarrow S[i-j] + j * bmax$
13. $l[i] \leftarrow j$
14. $S[i] \leftarrow S[i] + header$

计算过程

- $P = \langle 10, 12, 15, 255, 1, 2 \rangle$
- $S[1]=15, S[2]=19, S[3]=23, S[4]=42, S[5]=50$
- $l[1]=1, l[2]=2, l[3]=3, l[4]=2, l[5]=1$

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

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

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

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

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

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

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

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

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

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

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

$$6 \times 8 + 11 \quad 59$$

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: $O(n)$

最大子段和

问题：给定 n 个数(可以为负数)的序列

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

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

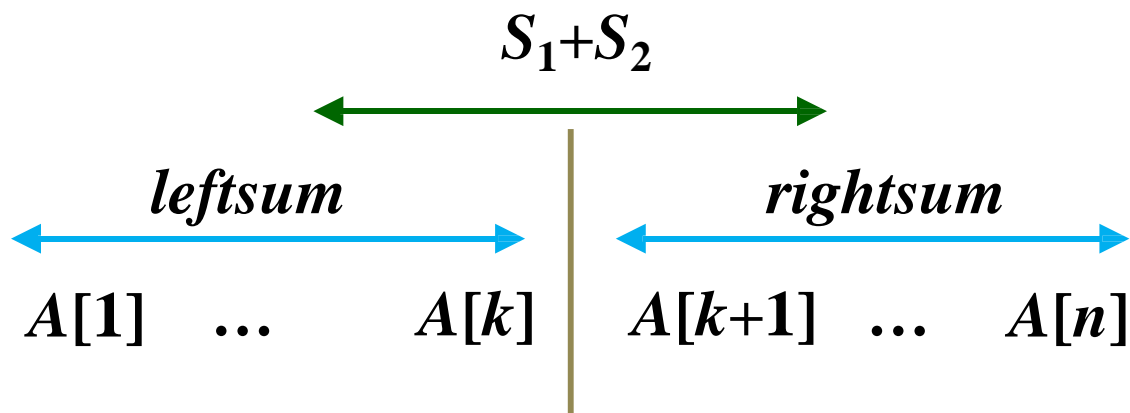
算法1

- 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. $thissum \leftarrow 0$
5. for $k \leftarrow i$ to j do
6. $thissum \leftarrow thissum + A[k]$
7. if $thissum > sum$
8. then $sum \leftarrow thissum$
9. $first \leftarrow i$
10. $last \leftarrow j$

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 $\text{MaxSubSum}(A, \text{left}, \text{right})$
 - 输入：数组 $A, \text{left}, \text{right}$ (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. $\text{center} \leftarrow \lfloor (\text{left} + \text{right}) / 2 \rfloor$
 3. $\text{leftsum} \leftarrow \text{MaxSubSum}(A, \text{left}, \text{center})$
 4. $\text{rightsum} \leftarrow \text{MaxSubSum}(A, \text{center} + 1, \text{right})$
 5. $S_1 \leftarrow A_1[\text{center}]$ //从 center 向左
 6. $S_2 \leftarrow A_2[\text{center} + 1]$ //从 $\text{center} + 1$ 向右
 7. $\text{sum} \leftarrow S_1 + S_2$
 8. if $\text{leftsum} > \text{sum}$ then $\text{sum} \leftarrow \text{leftsum}$
 9. if $\text{rightsum} > \text{sum}$ then $\text{sum} \leftarrow \text{rightsum}$

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

$$A[k],$$

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

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

$$\dots,$$

$$A[k]+\dots+A[1]$$

- 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

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



优化函数的递推方程

- 递推方程:

$$C[i] = \max_{i=2, \dots, n} \{ C[i-1] + A[i], A[i] \}$$

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

$$C[1] = 0 \quad \text{否则}$$

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

伪码

- 算法 $\text{MaxSum}(A, n)$
 - 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
- 时间复杂度： $O(n)$, 空间复杂度： $O(n)$

小结

- 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

