动态规划

东南大学计算机学院 方效林

本章内容

- 动态规划原理
- 矩阵连乘
- 钢条切割
- 最长公共子序列
- 最优二叉搜索树
- 流水作业调度
- 0/1背包问题

动态规划原理

- 与分治法类似，动态规划法也是把问题一层一层地分解为规模逐渐减小的同类型的子问题
- 分治法
  - 子问题是相互独立的
  - 若不独立，将重复计算
- 动态规划
  - 可分为多个相关子问题
  - 子问题的解被重复使用
  - 子问题只求解一次，结果保存在表中，以后用到时直接存取

动态规划原理

- 动态规划的条件
  - 最优子结构
    - 当一个问题的最优解包含了子问题的最优解时，称这个问题具有最优子结构
  - 重叠子问题
    - 在问题的求解过程中，很多子问题的解将被多次使用

矩阵连乘

- **两矩阵A和B，其维数分别是p×q和q×r，这两矩阵相乘需进行p×q×r次乘法**

矩阵连乘

- 3个矩阵相乘$M_1M_2M_3$，其维数分别为10×100, 100×5和5×50
  - 可按$M_1(M_2M_3)$的方法计算,
    - 计算M_2M_3：$100\times5\times50 = 25000$
    - 计算$M_1(M_2M_3)$：$10\times100\times50 = 50000$
    - 乘法运算总共25,000+50,000= 75,000
  - 也可按$(M_1M_2)M_3$的方法计算
    - 计算M_1M_2：$10\times100\times5 = 5000$
    - 计算$(M_1M_2)M_3$：$10\times5\times50 = 2500$
    - 乘法运算总共5,000+2,500=7,500

不同的计算顺序计算代价不同

矩阵连乘

- n个矩阵相乘，最小化乘法运算次数?
  - 解空间大小
    - 令p(n)为n个矩阵相乘不同计算方法的总数，则有
    - p(n) = 1 if n=1
    - p(n) = $\sum_{k=1}^{n-1} p(k)p(n-k)$ if n>1

$$(M_1M_2 \ldots M_k)(M_{k+1}M_{k+2} \ldots M_n)$$

    - p(n)正好是catalan数=$\frac{1}{n}C_{2(n-1)}^{n-1}$

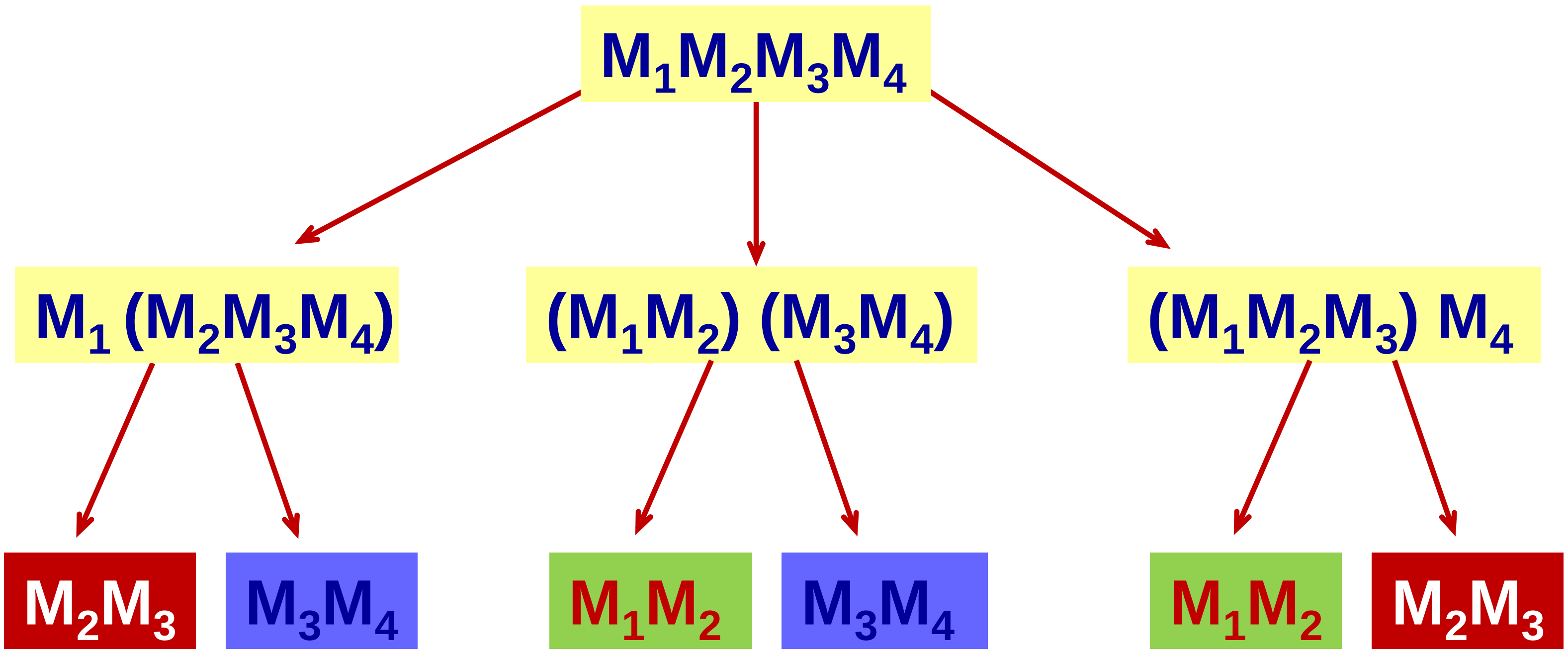
解空间巨大无法枚举

矩阵连乘

- n个矩阵相乘，最小化乘法运算次数?
  - 但是，若可分别得到$M_1M_2 \ldots M_k$和$M_{k+1}M_{k+2} \ldots M_n$的最小乘法次数，则可以得到在 k处断开的连乘方法$(M_1M_2 \ldots M_k)(M_{k+1}M_{k+2} \ldots M_n)$的最小乘法次数
    - 令$M_1M_2 \ldots M_k$的最小乘法次数为m[1,k]
    - $M_{k+1}M_{k+2} \ldots M_n$的最小乘法次数为m[k+1,n]
    - 则k处断开的最少乘法数$m[1,k] + m[k+1,n] + r_1 \times c_k \times c_n$

具有最优子结构：
问题的最优解包括子问题最优解

矩阵连乘



具有子问题重叠性

矩阵连乘

- 令m[i,j]表示$M_i M_{i+1} \ldots M_j$的最小乘法次数
- 则m[1,n]表示$M_1 M_2 \ldots M_n$的最小乘法次数
- 在k处断开$m[i,j] = m[i,k] + m[k+1,j]+r_i \times c_k \times c_j$
- 考虑所有k，则有
  - $m[i,j] = \min_{i \le k < j}\{m[i,k] + m[k+1,j] + r_i \times c_k \times c_j\}$, if $i<j$
  - $m[i,j] = 0$, if $i=j$

矩阵连乘

- $m[i,j] = \min_{i \le k < j}\{m[i,k] + m[k+1,j] + r_i \times c_k \times c_j\}$, if $i<j$
- $m[i,j] = 0$, if $i=j$

m[1,1]	m[1,2]	m[1,3]	m[1,4]	m[1,5]
	m[2,2]	m[2,3]	m[2,4]	m[2,5]
		m[3,3]	m[3,4]	m[3,5]
			m[4,4]	m[4,5]
				m[5,5]

矩阵连乘

自底向上方法

```
Matrix-Chain-Order(r)
    n=length(r);
    for  i=1 to n do
         m[i, i]=0;
    for  x=1  to  n-1  do
         for  i=1  to  n-x  do
              j = i+x;
              m[i, j] = ∞;
              for  k=i  to  j-1  do
                   q = m[i, k]+m[k+1, j]+r_{i-1}c_k c_j
                   if q<m[i, j] then
                        m[i,j]=q;
                        s[i,j]=k;
    return m and s
```

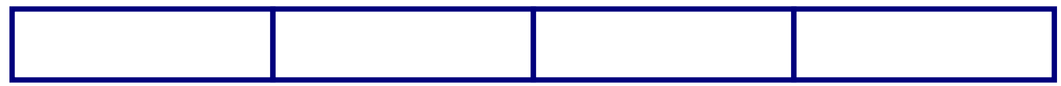
矩阵连乘

递归备忘录方法

```
m[N,N], s[N,N] 赋-1;
MatrixChain (i, j)
    if i=j then
        m[i, j]=0;
        return;
    m[i, j] = ∞;
    for  k=i  to  j-1  do
        a = m[i, k]; b = m[k+1, j];
        if a == -1 then  a = MatrixChain(i, k);
        if b == -1 then  b = MatrixChain(k+1, j);
        q = a + b + r_{i-1}c_kc_j
        if q<m[i, j] then
            m[i,j]=q;
            s[i,j]=k;
```

钢条切割

- 长度为n英寸的钢条进行切割，可有很多切法
  - 1段4英寸
  - 4段1英寸
  - 2段2英寸
  - 1段3英寸1段1英寸



- 假设有一张价格表

长度i	1	2	3	4
价格pi	1	5	8	9

将这4英寸的钢条切成2段2英寸的，收益最大，为10

钢条切割

- 问题定义
  - 给定一段长度为n英寸的钢条和一个价格表pi (i=1,2,…,n)，给定切割方案，使收益最大

钢条切割

- 令r(n)为长度为n英寸的钢条的最大收益，则

$$r(n) = \max_{1\leq i\leq n}\{p_i + r(n-i)\}$$

含义：从一端切一段长度为i的钢条下来后，可获得的最大收益

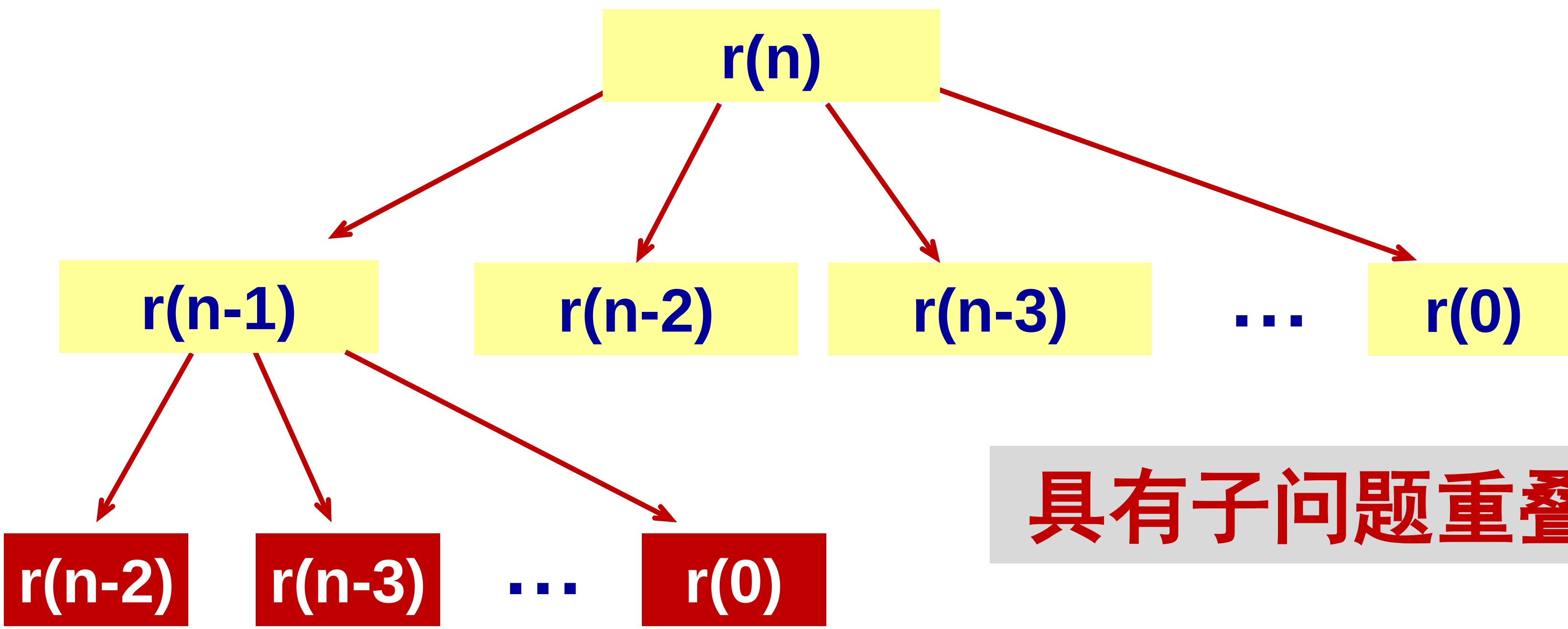
即切下来的钢条价格pi，加上剩下长度为n-i的钢条的最大收益

具有最优子结构：
问题的最优解包括子问题最优解

钢条切割

- 令r(n)为长度为n英寸的钢条的最大收益，则

$$r(n) = \max_{1 \le i \le n}\{p_i + r(n-i)\}$$



具有子问题重叠性

钢条切割

自底向上方法

```
CUT(p, n, r)
    r[0] = 0
    for j=1 to n
        temp = -∞
        for i=1 to j
            temp = max(temp, p[i]+r[j-i])
        r[j] = temp
```

r[0]不用计算，依次计算r[1],r[2],r[3],r[4]

钢条切割

递归备忘录方法

预处理r[ ]，使r[i]=-∞

```
CUT(p, n, r)
    if r[n]≥0 then return r[n]   //r[n]不用重复计算了
    if n == 0 then temp = 0
    else
        temp = -∞
        for i=1 to n do
            temp = max(temp , p[i]+CUT(p, n-i))
    r[n] = temp
    return temp
```

最长公共子序列

- 子序列
  - $X=<x_1, x_2, \ldots, x_m>$, 若$1 \le i_1 < i_2 < \cdots < i_k \le m$，使$Z=<z_1, z_2, \cdots, z_k> = <x_{i1}, x_{i2}, \cdots, x_{ik}>$, 称Z是X的子序列
    - X=(A, B, C, D, E, F, G)
    - Z=(B, C, E, F)是 X 的子序例
    - W=(B, D, A)不是X的子序例
- 公共子序列
  - Z 是序列 X 与 Y 的公共子序列
    - Z 是X的子序列
    - Z 也是Y的子序列

最长公共子序列

- 最长公共子序列（LCS）问题
  - 输入：$X = (x_1,x_2,\ldots,x_n)$，$Y = (y_1,y_2,\ldots y_m)$
  - 输出：Z = X与Y的最长公共子序列

最长公共子序列

- **第i前缀**
  - 设 $X=(x_1, x_2, ..., x_n)$ 是一个序列
  - X的第i前缀 X(i)，定义为$X(i)=(x_1, ..., x_i)$
    - 例如：X=(A, B, D, C, A),
    - X(1)=(A), X(2)=(A, B), X(3)=(A, B, D)

最长公共子序列

- **最优子结构性质**
  - 设$X=(x_1, \ldots, x_m)$、$Y=(y_1, \ldots, y_n)$是两个序列，$Z=(z_1, \ldots, z_k)$是X与Y的LCS，则有：
    - （1）如果$x_m=y_n$，
      - 则$z_k=x_m=y_n$, Z(k-1)是X(m-1)和Y(n-1)的LCS，
      - 即LCS(m,n) = LCS(m-1,n-1) + 1.
    - （2）如果$x_m \neq y_n$
      - 若最终$z_k \neq x_m$，则 LCS(m,n)= LCS(m-1,n)
      - 若最终$z_k \neq y_n$，则 LCS(m,n)= LCS(m,n-1)
      - LCS(m,n)=max{LCS(m-1,n), LCS(m,n-1)}

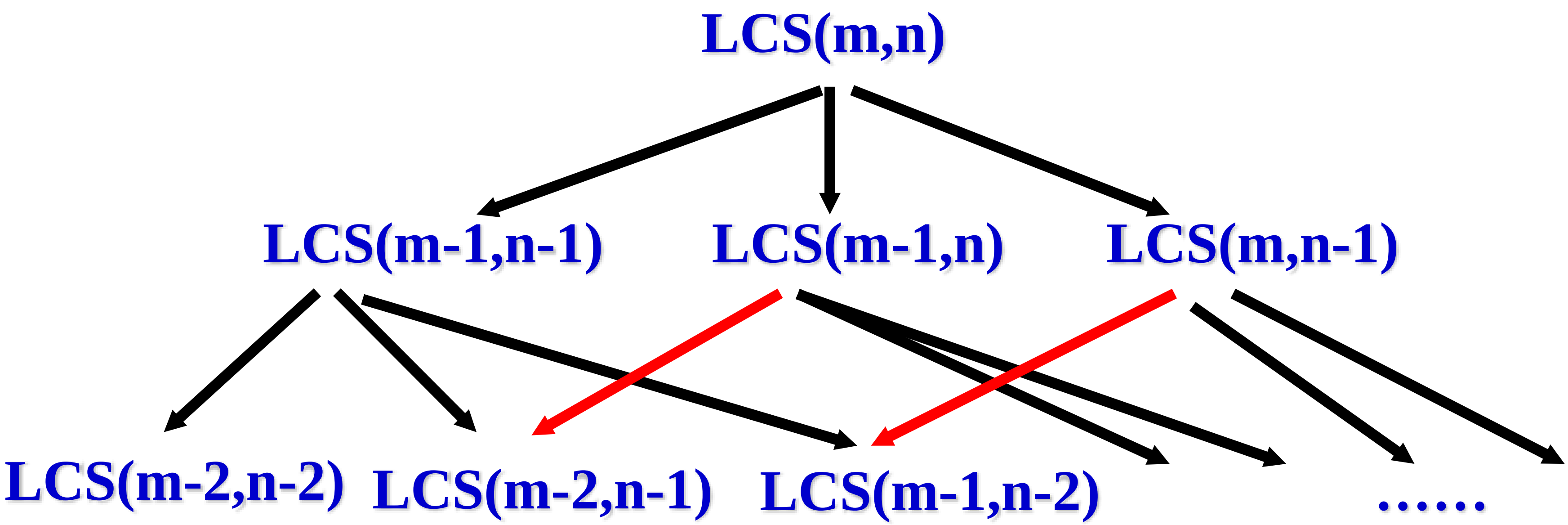
$X=(x_1, x_2, \ldots, x_{m-1}, x_m)$

$Y=(y_1, y_2, \ldots, y_{n-1}, y_n)$

问题最优解
包括子问题最优解

最长公共子序列

- 子问题重叠性质



最长公共子序列

- LCS递归方程
  - $LCS[i,j]=0$ if $i=0$ or $j=0$
  - $LCS[i,j]=LCS[i-1, j-1] + 1$ if $i, j>0, x_i=y_j$
  - $LCS[i,j]=Max(LCS[i,j-1], LCS[i-1,j])$ if $i,j>0, x_i \neq y_j$

最长公共子序列

- **基本思想**

	LCS[i-1, j-1]	LCS[i-1,j]	
	LCS[i, j-1]	LCS[i, j]	

最长公共子序列

- 自底向上计算过程

LCS[0,0]	LCS[0,1]	LCS[0,2]	LCS[0,3]	LCS[0,4]
LCS[1,0]	LCS[1,1]	LCS[1,2]	LCS[1,3]	LCS[1,4]
LCS[2,0]	LCS[2,1]	LCS[2,2]	LCS[2,3]	LCS[2,4]
LCS[3,0]	LCS[3,1]	LCS[3,2]	LCS[3,3]	LCS[3,4]

最长公共子序列

- 递归计算过程

LCS[0,0]	LCS[0,1]	LCS[0,2]	LCS[0,3]	LCS[0,4]
LCS[1,0]	LCS[1,1]	LCS[1,2]	LCS[1,3]	LCS[1,4]
LCS[2,0]	LCS[2,1]	LCS[2,2]	LCS[2,3]	LCS[2,4]
LCS[3,0]	LCS[3,1]	LCS[3,2]	LCS[3,3]	LCS[3,4]

最长公共子序列

```
for i=0 to m do LCS[i,0]←0
for j=1 to n do LCS[0,j]←0
for i=1 to m do
    for j=1 to n do
        if X[i]=Y[j] then
            LCS[i,j] = LCS[i-1,j-1]+1;
            b[i,j] = “↖” ;
        else if LCS[i-1,j]≥LCS[i,j-1] then
            LCS[i,j] = LCS[i-1,j];
            b[i,j] = “↑” ;
        else
            LCS[i,j] = LCS[i,j-1];
            b[i,j] = “←” ;
```

时间复杂度O(mn)

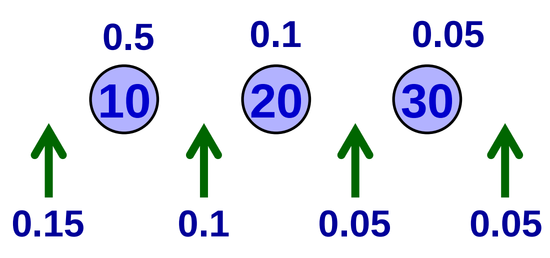
空间复杂度O(mn)

最长公共子序列

		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

最优二叉搜索树

- 问题
  - 假设有n个key，每个key搜索概率不同，除key之外的值(即搜索不成功情况)搜索概率也不同，构造平均搜索长度最小的二叉树
    - 例如，3个key ={10，20，30}，每个key搜索概率为P={0.5, 0.1, 0.05}，除key之外(空隙中)的值搜索概率为Q={0.15, 0.1, 0.05, 0.05}

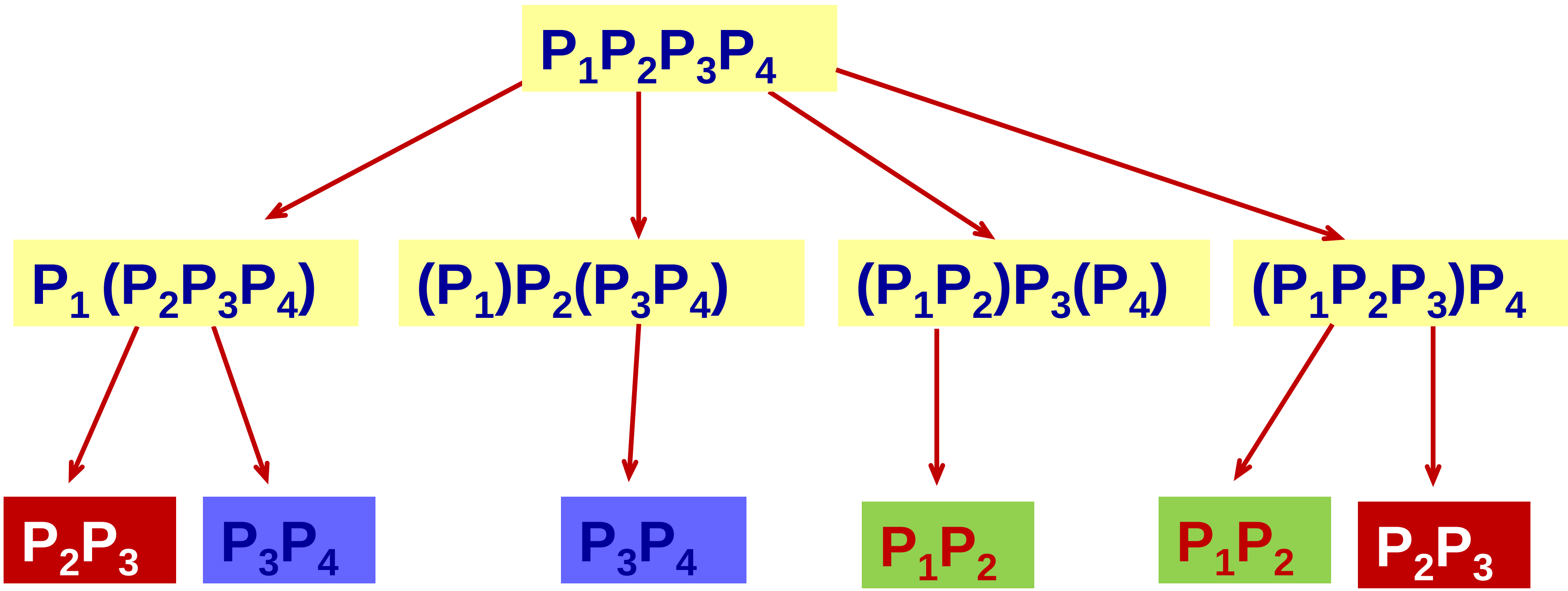


最优二叉搜索树

- 具有最优子结构性质
  - 如果优化二叉搜索树T具有包含关键字集合{i,i+1, …, j}子树T’，则T’必是关于关键字集合{i,i+1, …, j}子问题的最优解
  - 证明:
    - 若不然，必有关键字集{i,i+1, …, j}子树T’’，T’’的搜索代价低于T’。用T’’替换T中的T’，可以得到一个期望搜索代价比T小的原始问题的二叉搜索树，与T是最优解矛盾.

问题的最优解包括子问题最优解

最优二叉搜索树

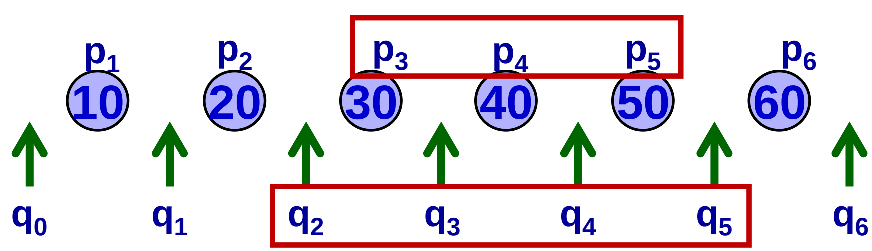


具有子问题重叠性

最优二叉搜索树

- 求解方法
  - c(i,j)表示第 i+1 到 j 个key构造的最优二叉树的代价(平均搜索长度)，则C(0,n)是最后结果
  - w(i,j)表示第 i+1 到 j 个key权值及第i到j个空隙权值和
    - $w(i,j) = (q_i+\ldots+q_j) + (p_{i+1}+\ldots+ p_j)$

$$w(i, j)=w(i, j-1)+p_j+q_j$$



$W(2,5)=(q_2+q_3+q_4+q_5) + (p_3+p_4+p_5)$

最优二叉搜索树

- 求解方法
  - c(i,j)表示第 i+1 到 j 个key构造的最优二叉树的代价(平均搜索长度)，则C(0,n)是最后结果
  - w(i,j)表示第 i+1 到 j 个key权值及第i到j个空隙权值和
    - $w(i,j) = (q_i+\ldots+q_j) + (p_{i+1}+\ldots+p_j)$
  - i+1, … , j以k为根的最优二叉树代价：

$$= p_k + c(i,k-1)+w(i,k-1) + c(k,j)+w(k,j)$$

左子树的代价

右子树的代价

根的代价

左子树加一层的代价

右子树加一层的代价

最优二叉搜索树

- 求解方法
  - c(i,j)表示第 i+1 到 j 个key构造的最优二叉树的代价(平均搜索长度)，则C(0,n)是最后结果
  - w(i,j)表示第 i+1 到 j 个key权值及第i到j个空隙权值和
    - $w(i,j) = (q_i+\ldots+q_j) + (p_{i+1}+\ldots+ p_j)$
  - i+1, … , j以k为根的最优二叉树代价：

$= p_k + c(i,k-1)+w(i,k-1) + c(k,j)+w(k,j)$

$= w(i,j) + c(i,k-1) + c(k,j)$

左子树的代价

树上权值和

右子树的代价

最优二叉搜索树

- 求解方法
  - c(i,j)表示第 i+1 到 j 个key构造的最优二叉树的代价(平均搜索长度)，则C(0,n)是最后结果
  - w(i,j)表示第 i+1 到 j 个key权值及第i到j个空隙权值和
    - $w(i,j) = (q_i+\dots+q_j) + (p_{i+1}+\dots+p_j)$
  - i+1, … , j以k为根的最优二叉树代价：

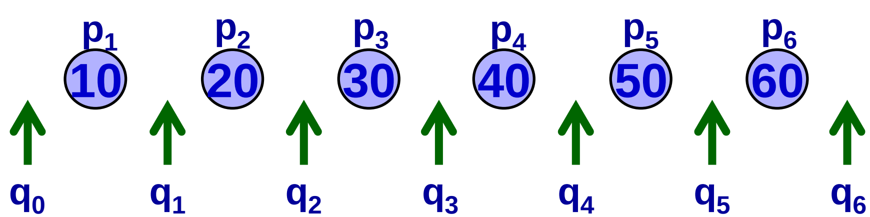
    $= p_k + c(i,k-1)+w(i,k-1) + c(k,j)+w(k,j)$

    $= w(i,j) + c(i,k-1) + c(k,j)$
  - i+1, … , j的最优二叉树代价：

$$c(i,j) = \underbrace{w(i,j)}_{\text{树上权值和}} + \underbrace{\min\{ c(i,k-1) + c(k,j) \}}_{\text{左右子树代价和最小值}}, \text{其中} i < k \le j$$

最优二叉搜索树

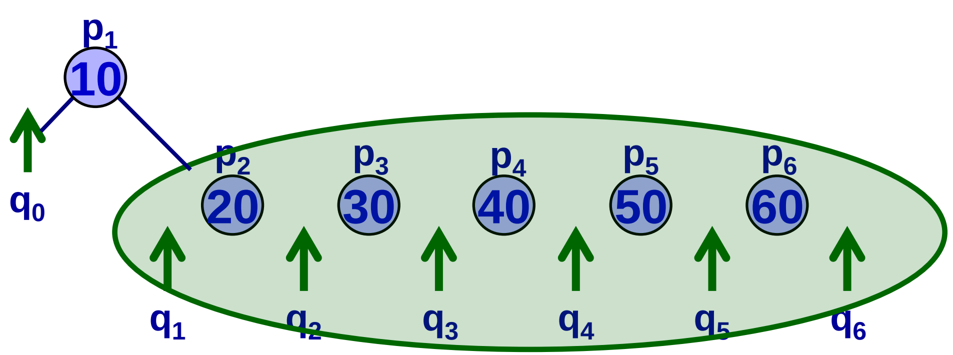
- **求解方法**
  - c(i,j)表示第 i+1 到 j 个key构造的最优二叉树的代价(平均搜索长度)， C(0,n)是最后结果
  - $c(i,j) = w(i,j) + \min\{ c(i,k-1) + c(k,j) \}$, 其中$i < k \leq j$



$c(0,6)=w(0,6)+\min\{c(0,k)+c(k,6)\}, 0 < k \leq 6$

最优二叉搜索树

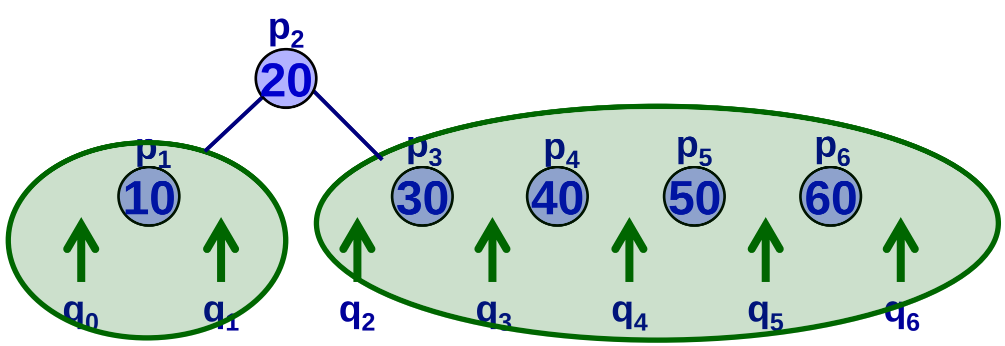
- 求解方法
  - c(i,j)表示第 i+1 到 j 个key构造的最优二叉树的代价(平均搜索长度)， C(0,n)是最后结果
  - $c(i,j) = w(i,j) + \min\{ c(i,k-1) + c(k,j) \}$, 其中$i < k \le j$



$c(0,6)=w(0,6)+\min\{c(0,k)+c(k,6)\}, 0 < k \le 6$

最优二叉搜索树

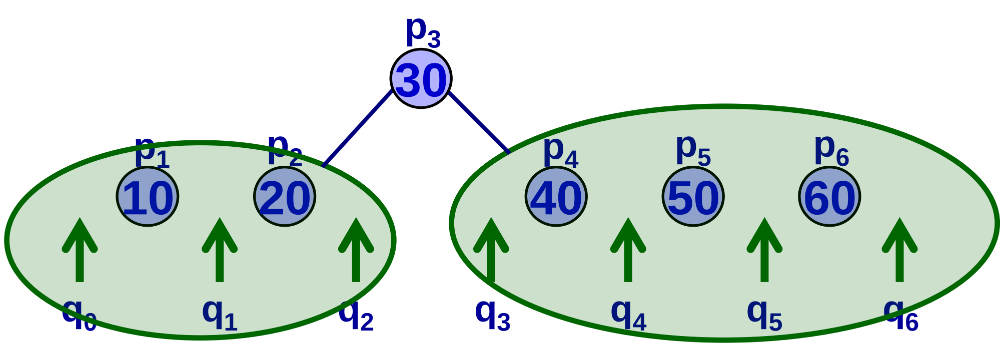
- 求解方法
  - c(i,j)表示第 i+1 到 j 个key构造的最优二叉树的代价(平均搜索长度)， C(0,n)是最后结果
  - $c(i,j) = w(i,j) + \min\{ c(i,k-1) + c(k,j) \}$, 其中$i < k \le j$



c(0,6)=w(0,6)+min{c(0,k)+c(k,6)}, 0 < k ≤ 6

最优二叉搜索树

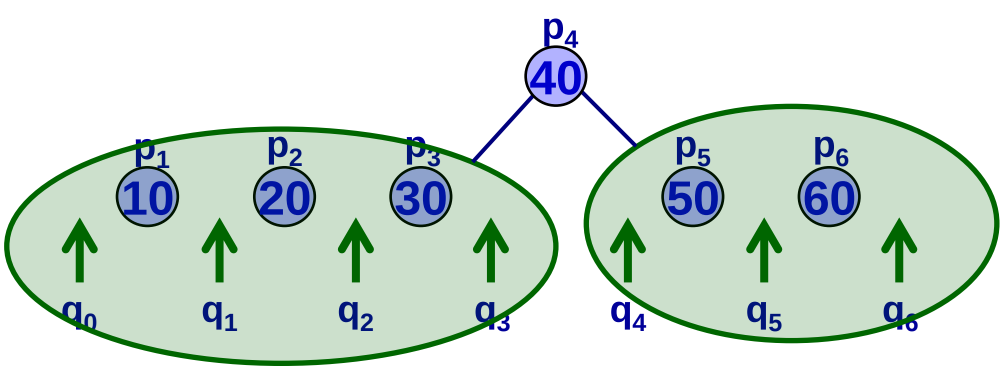
- 求解方法
  - c(i,j)表示第 i+1 到 j 个key构造的最优二叉树的代价(平均搜索长度)， C(0,n)是最后结果
  - $c(i,j) = w(i,j) + \min\{ c(i,k-1) + c(k,j) \}$, 其中$i < k \le j$



$c(0,6)=w(0,6)+\min\{c(0,k)+c(k,6)\}, 0 < k \le 6$

最优二叉搜索树

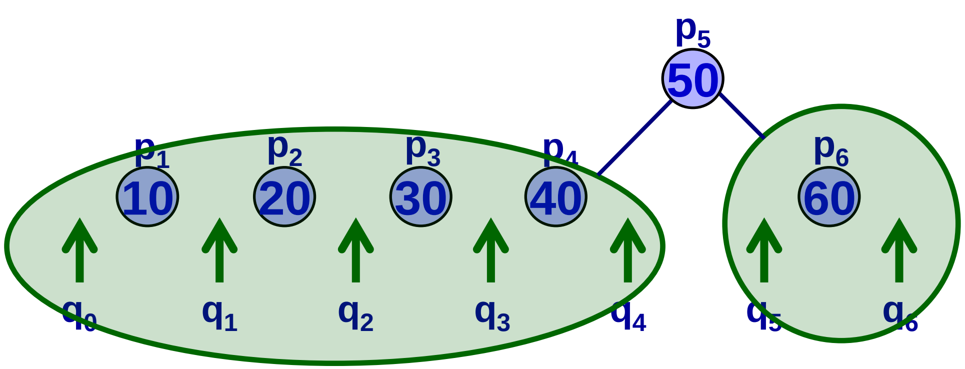
- 求解方法
  - c(i,j)表示第 i+1 到 j 个key构造的最优二叉树的代价(平均搜索长度)， C(0,n)是最后结果
  - $c(i,j) = w(i,j) + \min\{ c(i,k-1) + c(k,j) \}$, 其中$i < k \le j$



$c(0,6)=w(0,6)+\min\{c(0,k)+c(k,6)\}, 0 < k \le 6$

最优二叉搜索树

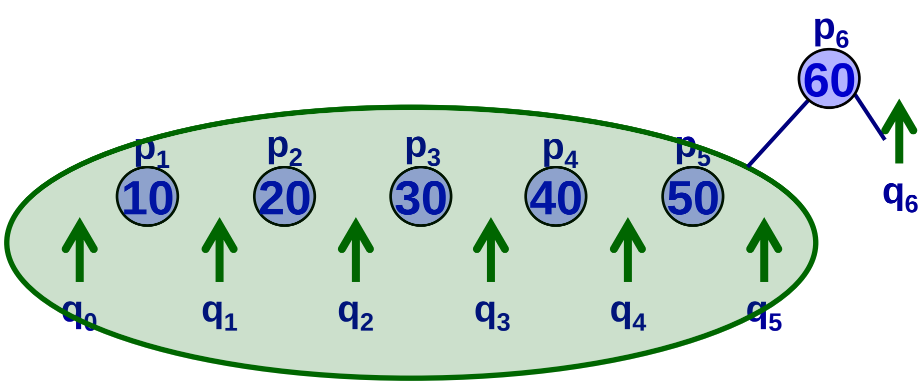
- 求解方法
  - c(i,j)表示第 i+1 到 j 个key构造的最优二叉树的代价(平均搜索长度)， C(0,n)是最后结果
  - $c(i,j) = w(i,j) + \min\{ c(i,k-1) + c(k,j) \}$, 其中$i < k \le j$



$c(0,6)=w(0,6)+\min\{c(0,k)+c(k,6)\}, 0 < k \le 6$

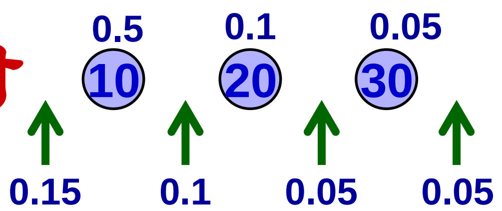
最优二叉搜索树

- 求解方法
  - c(i,j)表示第 i+1 到 j 个key构造的最优二叉树的代价(平均搜索长度)， C(0,n)是最后结果
  - $c(i,j) = w(i,j) + \min\{ c(i,k-1) + c(k,j) \}$, 其中$i < k \le j$

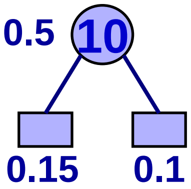


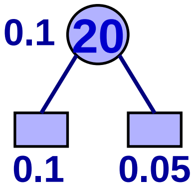
$c(0,6)=w(0,6)+\min\{c(0,k)+c(k,6)\}, 0 < k \le 6$

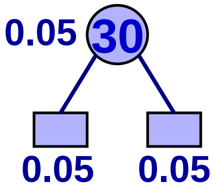
最优二叉搜索树



- 第1步：各key自成二叉树，计算平均搜索长度c，保留最小值







W=0.75

c(0,1)=0.75

W=0.25

c(1,2)=0.25

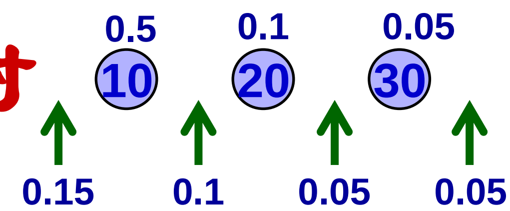
W=0.15

c(2,3)=0.15

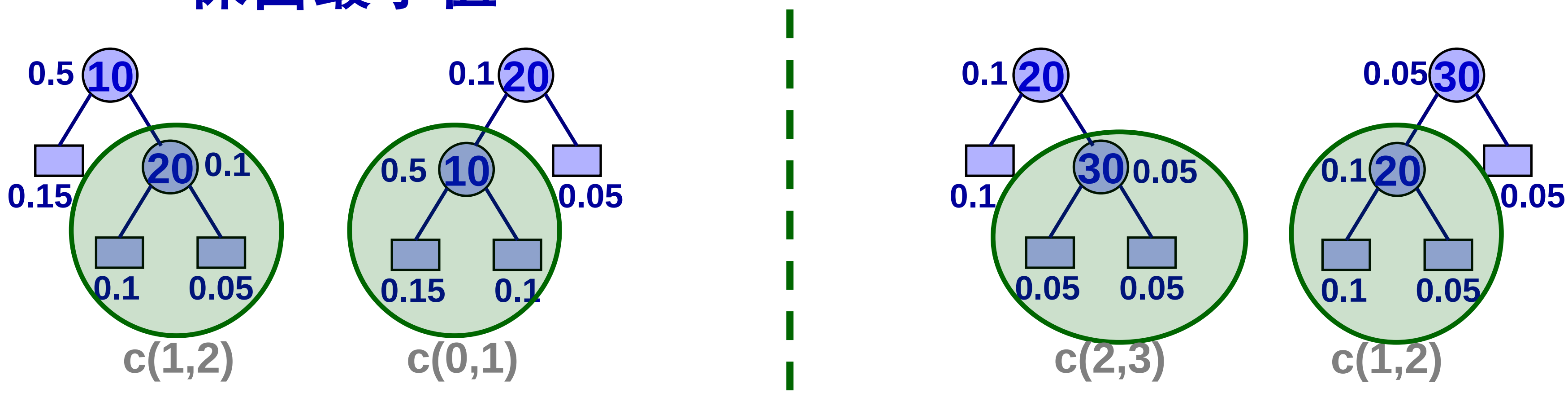
w等于树上所有结点(内部和外部)的权值和，
c等于树上所有结点(内部和外部)的权值和

c(0,1)=0.75		
	c(1,2)=0.25	
		c(2,3)=0.15

最优二叉搜索树



- 第2步：相邻2个key成二叉树, 计算平均搜索长度c保留最小值



c(0,2)=w+c(1,2)
=1.15

c(0,2)=w+c(0,1)
=1.65

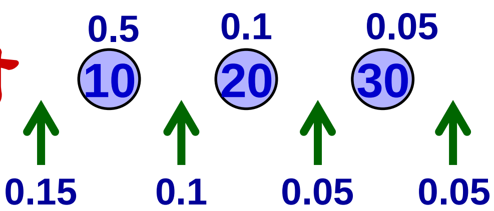
c(1,3)=w+c(2,3)
=0.5

c(1,3)=w+c(0,1)
=0.6

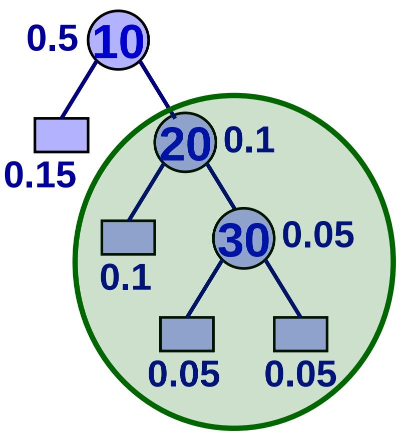
c(0,1)=0.75	c(0,2)=1.15	
	c(1,2)=0.25	c(1,3)=0.5
		c(2,3)=0.15

w等于树上所有结点(内部和外部)的权值和，
c等于w加上左右子树的代价

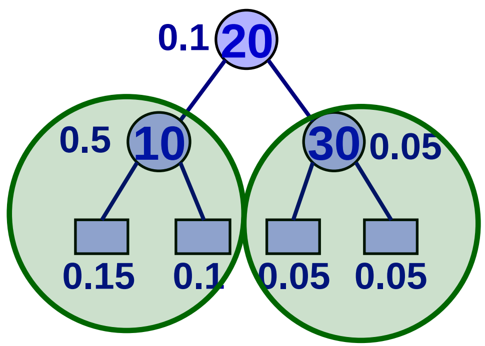
最优二叉搜索树



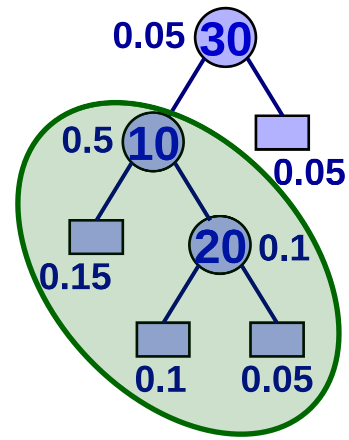
- 第3步：相邻3个key成二叉树, 计算平均搜索长度c保留最小值



c(0,3)=w+c(1,3)
=1.5



c(0,3)=w+c(0,1)+c(2,3)
=1.9



c(0,3)=w+c(0,2)
=2.15

w等于树上所有结点(内部和外部)的权值和,
c等于w加上左右子树的代价

c(0,1)=0.75	c(0,2)=1.15	c(0,3)=1.5
	c(1,2)=0.25	c(1,3)=0.5
		c(2,3)=0.15

最优二叉搜索树

```
Optimal-BST(p, q, n)
    for  i=0  to  n   do
        C(i, i) = 0;
        W(i, i) = q_i;
    for  x=1  to  n   do
        for   i=0   to   n-x   do
            j=i+x;
            C(i, j)=∞;
            W(i, j)=W(i, j-1)+p_j+q_j;
            for  k=i  to  j   do
                t = W(i,j);
                if k-1≥i  then t += C(i, k-1);
                if k+1≤j then t += C(k+1, j);
                if   t < C(i, j)  then  C(i, j)=t;  Root(i, j)=r;
```

时间复杂度$O(n^3)$

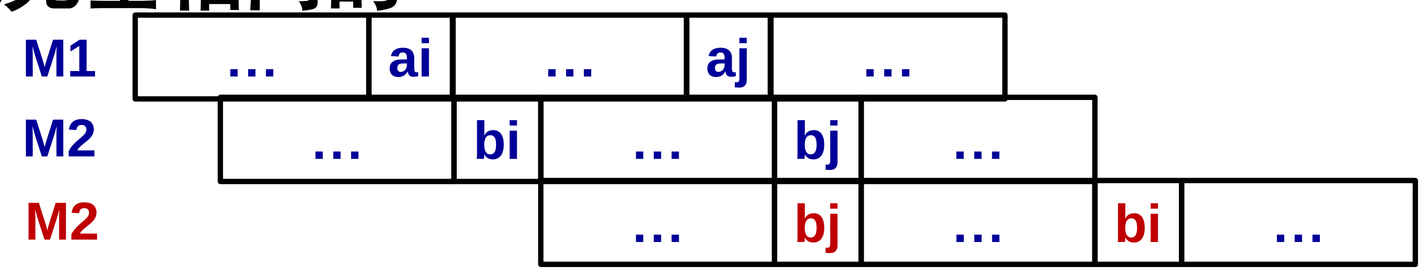
空间复杂度$O(n^2)$

流水作业调度问题

- 问题
  - n个作业 N={1，2，…，n}要在2台机器M1和M2组成的流水线上完成加工。每个作业须先在M1上加工，然后在M2上加工。M1和M2加工作业 i 所需的时间分别为 ai 和bi，每台机器同一时间最多只能执行一个作业。
  - 流水作业调度问题要求确定这n个作业的最优加工顺序，使得所有作业在两台机器上都加工完成所需最少时间

流水作业调度问题

- 一定存在最优调度使M1上的加工是无间断的
  - 即M1上的总加工时间是所有ai之和
  - M2上不一定是bi之和
- 一定存在最优调度使作业在两台机器上的加工次序是完全相同的



若是bi和bj交换，则bj需要aj结束，等待时间更长

仅需考虑在两台机上上加工次序完全相同的调度

流水作业调度问题

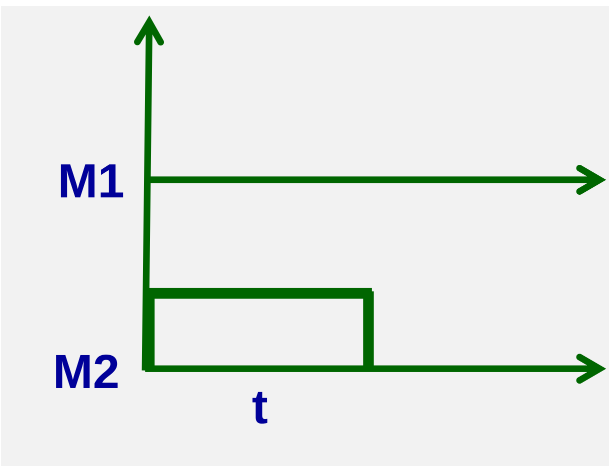
- $N=\{1,2,\ldots,n\}$，子集$S \subseteq N$
  - 机器M1开始加工S中作业时，机器M2还在加工其他作业，要等时间 t 后才可利用
    - 则完成S中作业所需的最短时间记为T(S,t)
    - 完成所有作业所需的最短时间记为T(N,0)
    - $T(N,0)=\min\{a_i + T(N-\{i\}, b_i)\}, \ i \in N$
      - ai：选一个作业 i 先加工，在M1的加工时间
      - T(N-{i},bi}：剩下的作业等bi才能在M2加工

流水作业调度问题

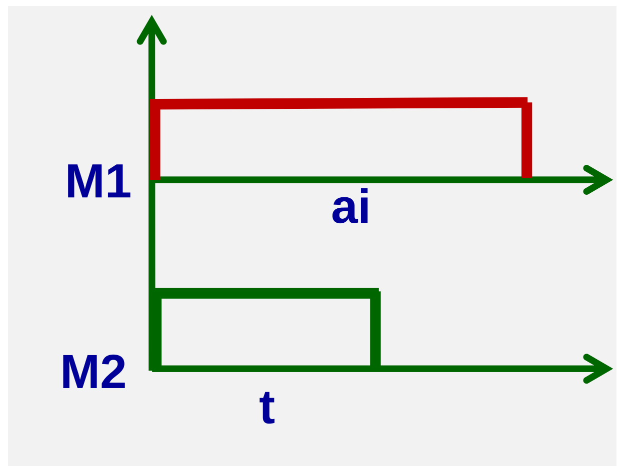
- $N=\{1,2,\ldots,n\}$，子集$S\subseteq N$
  - 机器M1开始加工S中作业时，机器M2还在加工其他作业，要等时间 t 后才可利用
    - 则完成S中作业所需的最短时间记为$T(S,t)$
    - 完成所有作业所需的最短时间记为$T(N,0)$
    - $T(S,t)=\{ai + T(S-\{i\}, bi+\max\{t-ai,0\})\}, i\in S$
      - ai：选一个作业 i 先加工，在M1的加工时间
      - $T(S-\{i\}, bi+\max\{t-ai,0\})$：剩下的作业等$bi+\max\{t-ai,0\}$才能在M2加工

流水作业调度问题

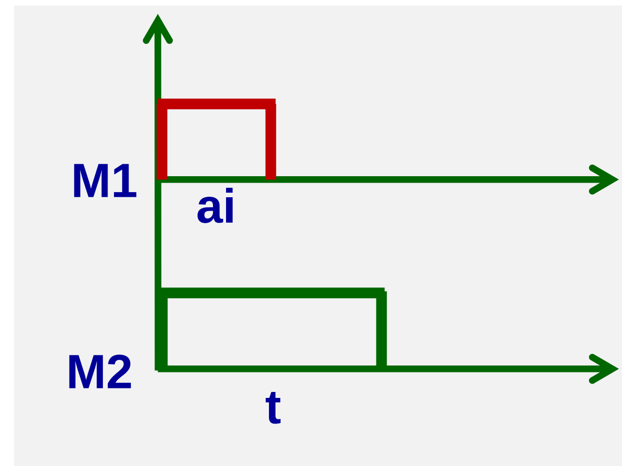
- $T(S,t)=\{a_i + T(S-\{i\}, b_i+\max\{t-a_i,0\})\}, i\in S$



$T(S,t)$



$a_i + T(S-\{i\}, b_i)$



$a_i + T(S-\{i\}, b_i+t-a_i)$

max{t-ai,0} 是由于在机器**M2**上，作业**i**必须在**max{t,ai}**时间之后才能加工，因此，在机器**M1**上完成作业加工**i**之后，在机器上还需**bi+max{t,ai}-ai=bi+max{t-ai,0}**

流水作业调度问题

- 最优子结构性质 **问题最优解包括子问题最优解**
  - 设π是N的一个最优调度，其加工顺序为π1,…, πn，其所需的加工时间为 $a_{\pi 1}$+T’。
  - 记S=N-{π1}，则T’=T(S, $b_{\pi 1}$)。
  - 证明：由T的定义知T(S, $b_{\pi 1}$)是对S最优的，故T’>=T(S, $b_{\pi 1}$)。若T’>T(S, $b_{\pi 1}$),设π’是作业集S在机器M2的等待时间为$b_{\pi 1}$情况下的一个最优调度。则π1,π’2,…,π’n是N的一个调度，且该调度所需的时间为$a_{\pi 1}$+T’ > $a_{\pi 1}$+T(S, $b_{\pi 1}$)。这与π是N的最优调度矛盾。故T’<=T(S, $b_{\pi 1}$), 从而T’=T(S, $b_{\pi 1}$)。最优子结构的性质得证。

流水作业调度问题

- 子问题重叠性质

T(N,0)

T(N-{1},b1)　　T(N-{2},b2)　　……　　T(N-{n},bn)

?

T(N-{1,4},b4+max{b1-a4,0})

……

T(N-{1,3},b3+max{b1-a3,0})

T(N-{1,2},b2+max{b1-a2,0})

流水作业调度问题

- 虽然满足最优子结构性质
- 也在一定程度满足子问题重叠性质
- 但是N的每个非空子集都计算一次，共2^n-1次，指数级的

流水作业调度问题

- **Johnson不等式**
  - 设一个最优调度中最前面的两个作业是i和j
  - $T(S,t) = ai + T(S-\{i\}, bi+\max\{t-ai,0\})$
  - $\quad\quad = ai + aj + T(S-\{i,j\}, tij)$
  - $tij = bj + \max\{bi+\max\{t-ai,0\}-aj,0\}$
  - $\quad = bj+bi-aj + \max\{\max\{t-ai,0\}, aj-bi\}$
  - $\quad = bj+bi-aj + \max\{t-ai,aj-bi,0\}$
  - $\quad = bj+bi-aj-ai + \max\{t,ai+aj-bi, ai\}$
  - 如果作业 i 和 j 满足$\min\{bi,aj\} \geq \min\{bj,ai\}$,则称作业 i 和 j 满足Johnson不等式

流水作业调度问题

tij ≤ tji。可得满足Johnson不等式为最优调度，不能交换

- 设一个最优调度中最前面的两个作业是i和j
- T(S,t) = ai + T(S-{i}, bi+max{t-ai,0})
- = ai + aj + T(S-{i,j}, tij)
- tij = bj+bi-aj-ai + max{t,ai+aj-bi, ai}，交换i和j后
- tji = bj+bi-aj-ai + max{t,ai+aj-bj, aj}
- 若满足Johnson不等式：min{bi,aj} ≥ min{bj,ai},
  - max{-bi,-aj} ≤ max{-bj,-aj}
  - ai+aj+max{-bi,-aj} ≤ ai+aj+max{-bj,-ai}
  - max{ai+aj-bi,ai} ≤ max{ai+aj-bj,aj}
  - max{t,ai+aj-bi,ai} ≤ max{t,ai+aj-bj,aj}

流水作业调度问题

- 计算过程
  - 满足Johnson不等式：min{bi,aj} ≥ min{bj,ai}
  - 则任务 i 应在 j 之前
  - 推广到一般情况：
    - 当min{a_1, a_2, … , a_n , b_1, b_2,…, b_n }=ai时，任何k≠i，都有min{bi,ak} ≥ min{bk,ai}，应将 i 安排在最前面
    - 当min{a_1, a_2,… , a_n , b_1, b_2,… , b_n }=bj时，任何k≠j，都有min{bk,aj} ≥ min{bj,ak}，应将 j 安排在最后面

流水作业调度问题

- 计算过程
  - (a1, a2, a3, a4) = (5, 12, 4, 8)
  - (b1, b2, b3, b4) = (6, 2, 14, 7)

作业	1	2	3	4
a	5	12	4	8
b	6	2	14	7

调度结果				
	3	1	4	2

0/1背包问题

- 问题定义
  - 给定 n 个物品和一个背包，物品 i 的重量是 w_i，价值v_i，背包容量为C，问如何选择装入背包的物品，使装入背包中的物品的总价值最大？

物品只能选择不装或者装入背包，而不分切一小部分装入，即0或者1

0/1背包问题

- 形式描述
  - 输入：$C > 0, w_i > 0, v_i > 0, 1 \le i \le n$
  - 输出：$(x_1, x_2, \ldots, x_n), x_i \in \{0, 1\}$, 满足$\Sigma_{1 \le i \le n} w_i x_i \le C$,
  - 使得$\Sigma_{1 \le i \le n} v_i x_i$最大
  - 这是一个整数规划问题

$$\max\{\Sigma_{1 \le i \le n} v_i x_i\}$$
$$\Sigma_{1 \le i \le n} w_i x_i \le C$$
$$x_i \in \{0, 1\}, 1 \le i \le n$$

0/1背包问题

- 最优子结构
  - 如果$(x_1, x_2, \ldots, x_n)$是0/1背包问题的最优解，
  - 则$(x_2, x_3, \ldots, x_n)$是如下子问题的最优解

$$\max\{\Sigma_{2 \le i \le n} v_i x_i\}$$
$$\Sigma_{2 \le i \le n} w_i x_i \le C - w_1 x_1$$
$$x_i \in \{0, 1\}, 2 \le i \le n$$

- 证明
  - 若$(x_2, x_3, \ldots, x_n)$不是该子问题的最优解，则存在子问题的最优解$(z_2, z_3, \ldots, z_n)$，那么$(x_1, z_2, z_3, \ldots, z_n)$是原问题的最优解，矛盾

0/1背包问题

- 设子问题为

$$\max\{\textstyle\sum_{i \le k \le n} v_k x_k\}$$
$$\textstyle\sum_{i \le k \le n} w_k x_k \le j$$
$$x_k \in \{0, 1\},\ i \le k \le n$$

- 其最优解为m(i, j)

含义：m(i, j)是背包容量为 j, 可选物品为i, i+1, …, n时问题的最优解

0/1背包问题

- 递归方程
  - $m(i, j) = m(i+1, j)$ $\quad 0 \le j < w_i$
  - $m(i, j) = \max\{m(i+1, j), m(i+1, j-w_i)+v_i\}$ $\quad j \ge w_i$
  - $m(n, j) = 0$ $\quad 0 \le j < w_n$
  - $m(n, j) = v_n$ $\quad j \ge w_n$

m(1,C)

m(2,C-w_1) m(2,C)

m(3, C-w_1-w_2) m(3,C-w_2) m(3,C-w_1) m(3,C)

0/1背包问题

for $j = 0$ **to** $\min(w_n-1, C)$ **do**

  $m[n, j] = 0;$

for $j = w_n$ **to** C **do**

  $m[n, j] = v_n;$

for $i = n-1$ **to** 2 **do**

  for $j=0$ **to** $\min(w_i-1, C)$ **do**

    $m[i, j] = m[i+1, j];$

  for $j=w_i$ **to** C **do**

    $m[i, j]=\max\{m[i+1, j], m[i+1, j-w_i]+v_i\};$

if $C < w_1$ **then** $m[1, C]=m[2, C];$

else $m[1, C]=\max\{m[2, C], m[2, C-w_1]+v_1\};$

0/1背包问题

- **构造最优解**
  - m(1, C)是最优解代价值，相应解计算如下：
  - if m(1, C)=m(2, C) then
  - $x_1=0$
  - 由m(2, C)继续构造最优解
  - Else
  - $x_1=1$;
  - 由m(2, C-w_1)继续构造最优解

小结

- 问题一般采用动态规划法，当具有：
  - 1）最优子结构性质时
  - 2）高度重复性
- 若问题不是NP-hard问题
  - 进一步分析后就有可能获得效率较高的算法。
- 若问题本身就是NP-hard问题
  - 那么与其它的精确算法相比，动态规划法性能一般不算太坏

- （1）给出N个1-9的数字 $(v_1,v_2,\ldots,v_N)$，不改变它们的相对位置，在中间加入K个乘号和N-K-1个加号，（括号随便加）使最终结果尽量大。因为乘号和加号一共就是N-1个了，所以恰好每两个相邻数字之间都有一个符号。并说明其具有优化子结构性质及子问题重叠性质。
  - 例如： N=5, K=2，5个数字分别为1、2、3、4、5，可以加成：
  - 1*2*(3+4+5)=24
  - 1*(2+3)*(4+5)=45
  - (1*2+3)*(4+5)=45

- （2）给定一长度为N的整数序列$(a_1,a_2,\ldots,a_N)$，将其划分成多个子序列（此问题中子序列是连续的一段整数），满足每个子序列中整数的和不大于一个数B，设计一种划分方法，最小化所有子序列中最大值的和。说明其具有优化子结构及子问题重叠性质
  - 例如： 序列长度为8的整数序列(2,2,2,8,1,8,2,1)，B=17，可将其划分成三个子序列(2,2,2)，(8,1,8)以及(2,1)，则可满足每个子序列中整数和不大于17，所有子序列中最大值的和12为最终结果。

- （3）对一棵树进行着色，每个结点可着黑色或白色，相邻结点不能着相同黑色，但可着相同白色。令树的根为r，请设计一种算法对树中尽量多的节点着黑色。

- （4）在自然语言处理中一个重要的问题是分词，例如句子“他说的确实在理”中“的确”“确实”“实在”“在理”都是常见的词汇，但是计算机必须为给定的句子准确判断出正确分词方法。一个简化的分词问题如下：给定一个长字符串$y=y_1y_2\ldots y_n$，分词是把y切分成若干连续部分，每部分都单独成为词汇。我们用函数quality(x)判断切分后的某词汇$x=x_1x_2\ldots x_k$的质量，函数值越高表示该词汇的正确性越高。分词的好坏用所有词汇的质量的和来表示。例如对句子“确实在理”分词，quality(确实) + quality(在理) > quality(确)+quality(实在)+quality(理)。请设计一个动态规划算法对字符串y分词，要求最大化所有词汇的质量和。（假定你可以调用quality(x)函数在一步内得到任何长度的词汇的质量）

- （5）给定 n个活动，活动a_i表示为一个三元组(s_i, f_i, v_i)，其中s_i表示活动开始时间，f_i表示活动的结束时间，v_i表示活动的权重。带权活动选择问题是选择一些活动，使得任意被选择的两个活动a_i和a_j执行时间互不相交，即区间$[s_i, f_i]$与$[s_j, f_j]$互不重叠，并且被选择的活动的权重和最大。请设计一种方法求解带权活动选择问题。

- （6）受限最短路径长度问题：给定一无向图G=(V, E, A, B)，A(e)表示边e的长度，B(v)表示顶点v的花费，计算小明从顶点s到顶点d的最短路径长度，满足以下限制，初始时小明随身携带M元钱，每经过一个顶点v，须交B(v)的过路费，若身上有大于B(v)的钱则可以通过，否则不可以通过。求顶点s到顶点d的最短路径

- （7）给定n个物品，每个物品有大小s_i，价值v_i。背包容量为C。要求找到一组物品，这些物品整包完全占满背包容量C，且总体价值最大。请写出动态规划迭代公式。

- （8）最大子数组问题：一个包含n个整数（有正有负）的数组A，设计一O(nlogn)算法找出和最大的非空连续子数组。（例如：[0, -2, 3, 5, -1, 2]应返回9，[-9, -2, -3, -5, -3]应返回-2。）

- （9）最长非降子序列：一个序列有N个数：A[1],A[2],…,A[N]，求出最长非降子序列的长度。