动态规划算法

学习要点

- 理解动态规划算法的概念。
- 掌握动态规划算法的适用条件及其证明

（1）最优子结构性质

（2）重叠子问题性质

- 掌握设计动态规划算法的步骤。

（ 1 ）找出最优解的性质，并刻划其结构特征。

（ 2 ）递归地定义最优值。

（ 3 ）以自底向上的方式计算出最优值。

（ 4 ）根据最优值的信息构造最优解

学习要点

- 通过应用范例学习动态规划算法设计策略。

（1）矩阵连乘问题；

（2）最长公共子序列；

（3）凸多边形最优三角剖分；

（4）0/1背包问题；

（5）最优二叉搜索树。

背景

- 解决优化问题
  - 给定一组约束条件和一个代价函数，在解空间中搜索具有最小或最大代价的优化解
- Why动态规划？
  - 对于一些优化问题，可以将其（递归）分解为若干子问题，但是经分解得到的子问题不是互相独立的。若用分治法来解决这些问题，分解得到的子问题数目太多，有些子问题被重复计算了多次
  - 如果我们能用一个表来记录所有已解决的子问题的答案，在需要时找出已求得的答案，就可以避免大量重复计算

基本步骤

- 找出最优解的性质，刻画其结构特征
- 递归地定义最优值
- 以自底向上的方式计算出最优值
- 根据最优值构造最优解（可选）

矩阵连乘问题

问题定义

- 输入：n个矩阵$A_1,A_2,\ldots,A_n$，其中A_i的维数为$p_{i-1} \times p_i$
  - A_i 和A_{i+1}是可乘的
  - A_i 的列p_i等于A_{i+1}的行p_i
- 输出：连乘积$A_1A_2\ldots A_n$
- 优化目标：最小的计算代价（最优的计算次序）
  - 矩阵乘法的代价：乘法次数
  - 例如，若A 是p ×q 矩阵，B 是q ×r 矩阵，则A ×B 的代价是pqr。
- 由于矩阵乘法满足结合律，所以计算矩阵的连乘可以有许多不同的计算次序。这种计算次序可以用加括号的方式来确定。

矩阵连乘问题

问题定义

- 若一个矩阵的计算次序确定，即该连乘积已完全加括号，反复调用2个矩阵乘法法则来计算
- 比如A_1 A_2 A_3 A_4有以下五种完全加括号方式：

  $((A_1\ A_2)\ (A_3\ A_4))$

  $(A_1\ ((A_2\ A_3)\ A_4))$

  $((A_1\ (A_2\ A_3))\ A_4)$

  $(A_1\ (A_2\ (A_3\ A_4)))$

  $(((A_1\ A_2)A_3)A_4)$
- 每一种完全加括号的方式，对应一种计算次序

矩阵连乘问题

例子

- 三个矩阵A_1: 10×100，A_2: 100×5，A_3: 5×50
- $(A_1A_2)A_3$
  - 代价：10×100×5＋10×5×50＝7500
- $A_1(A_2A_3)$
  - 代价：100×5×50＋10×100×50＝75000

计算量和计算次序有很密切的联系

如何找到最优的计算次序？？

矩阵连乘问题

- 穷举法
  - 对于n个矩阵的连乘，假设有P(n)个不同的计算次序
  - 在第k和k+1个矩阵之间将n个矩阵一分为二，k=1.2…n-1
  - 对两个矩阵子序列完全加括号

$$P(n)=\begin{cases}1 & n=1\\ \sum_{k=1}^{n-1}P(k)P(n-k) & n>1\end{cases}$$

$$n!=\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

卡特兰数(Catalan Number)：

$$P(n)=\frac{1}{n+1}C_{2n}^{n}=\frac{1}{n+1}*\frac{2n!}{n!\ (2n-n)!}$$

$$P(n)=\frac{(2n)!}{(n+1)!n!}=\Omega(4^n/n^{3/2})$$

矩阵连乘问题

分析最优解结构(1)

- 将矩阵连乘积$A_iA_{i+1}\ldots A_j$，简记为A[i:j]
- 设$A_iA_{i+1}\ldots A_j$的最优计算次序在矩阵A_k和A_{k+1}之间将矩阵链断开：$(A_i\ldots A_k)(A_{k+1}\ldots A_j)$
- 总计算量＝A[i:k]的计算量＋A[k+1: j]的计算量+A[i:k]和A[k+1:j]相乘的计算量
- 最优子结构
  - *假设A[i: j]的最优计算次序是从A_k处断开，那么该问题包含的子矩阵链A[i:k]和A[k+1:j]的计算次序也是最优的*
  - *原问题最优解包含子问题的最优解，称为最优子结构性质*
- 最优解：最优的计算次序（完全加括号的方式）
- 最优值：最优解下的计算代价（计算量）

矩阵连乘问题

- 建立递归关系（2）
  - **计算A[i : j]所需的最少乘法次数为m(i, j)**

$$m(i,j)=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m(i,k)+m(k+1,j)+p_{i-1}p_kp_j\} & i<j\end{cases}$$

  - 其中A_i是$p_{i-1}\times p_i$矩阵
  - **A[i:k]** 是 $p_{i-1}\times p_k$矩阵，**A[k+1:j]** 是 $p_k\times p_j$矩阵，

矩阵连乘问题

```
int matrixmultiply(i, j){
        if(i==j) return 0;
        int u=infinity;
        for(k=i; k<j; k++){
                int t=matrixmultiply (i, k) + matrixmultiply (k+1, j) + p[i-1]p[k]p[j];
                if(t<u) u=t;
        }
        return u;
}
```

❑ 时间复杂性

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

- 可以证明 $T(n)=n+2\sum_{k=1}^{n-1}T(k)=\Omega(2^n)$

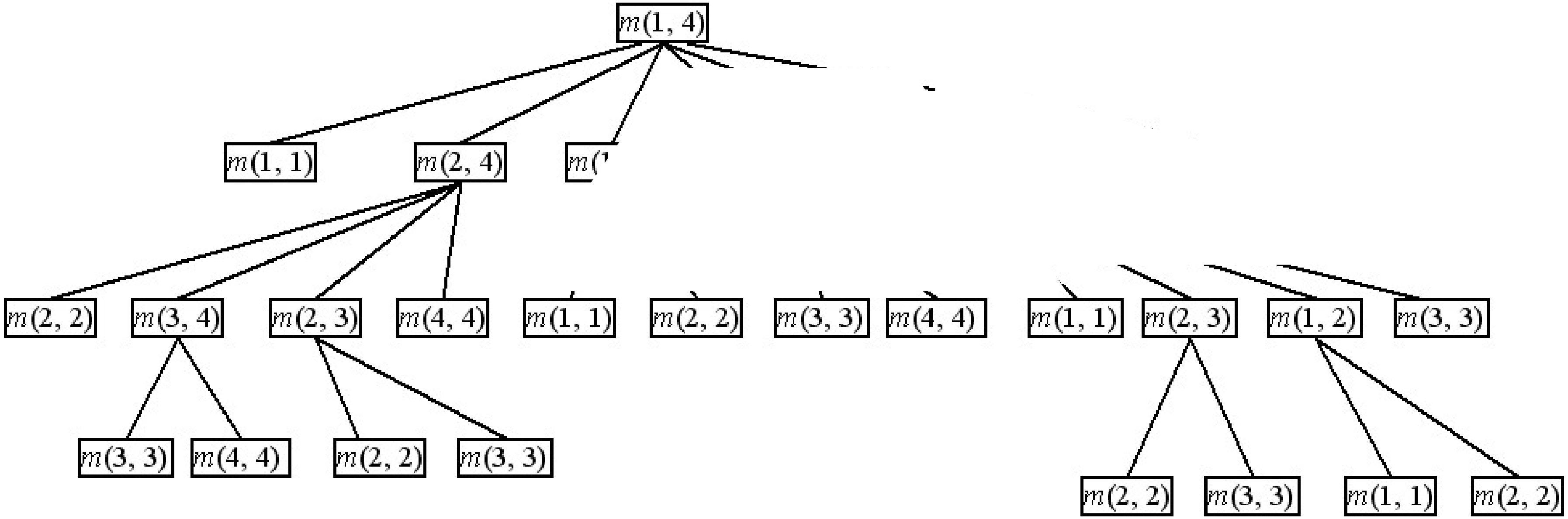
仅仅考虑最优子结构性质的递归求解并没有很好的效果

矩阵连乘问题

重叠子问题

$$m(i,j)=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m(i,k)+m(k+1,j)+p_{i-1}p_kp_j\} & i<j\end{cases}$$

计算m(1, 4)



对于1≤i≤j≤n,不同的有序对儿（i， j）对应不同的子问题,所以计算m(1， n)不同子问题的个数只有O(n^2)个（n个元素任意选两个不同或相同）

事实上，许多重复的子问题被计算多次（对比2^n）。

矩阵连乘问题（动态规划）

- 动态规划方法计算最优值（3）
  - 自底向上计算（从最简单的算起）
  - 用一个二维表保存已解决问题的答案
    - 每个子问题只计算一次，后面需要的时候直接查找结果

矩阵连乘问题（动态规划）

$$m(i,j)=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m(i,k)+m(k+1,j)+p_{i-1}p_kp_j\} & i<j\end{cases}$$

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]

计算m(1，5)

矩阵连乘问题（动态规划）

$$m(i,j)=\begin{cases} 0 & i=j \\ \min\limits_{i\le k<j}\{m(i,k)+m(k+1,j)+p_{i-1}p_kp_j\} & i<j \end{cases}$$

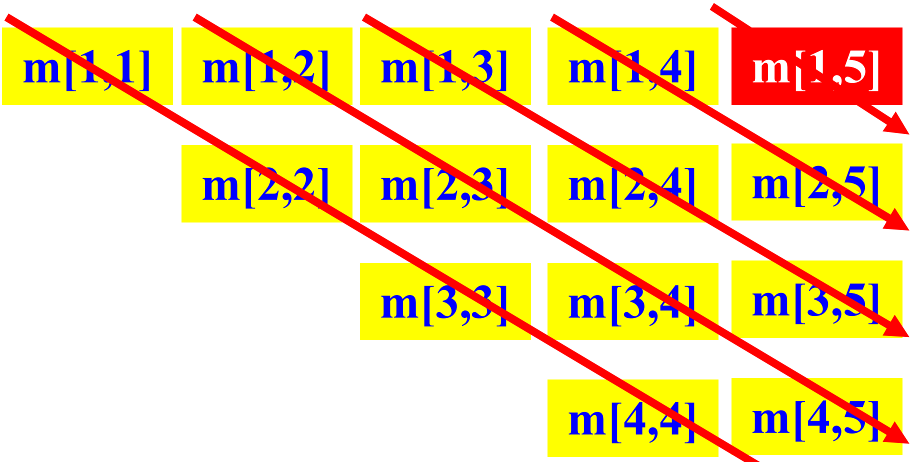
m[1,1]	m[1,2]	m[1,3]	m[1,4]	m[1,5]
	m[2,2]	m[2,3]	m[2,4]	m[2,5]
		m[3,3]	m[3,4]	m[3,5]
			m[4,4]	m[4,5]
				m[5,5]

计算m(1， 5)

矩阵连乘问题（动态规划）

$$m(i,j)=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m(i,k)+m(k+1,j)+p_{i-1}p_kp_j\} & i<j\end{cases}$$

```
MatrixChain(int *p, int n, int **m, int **s)
{
    for (int i = 1; i <= n; i++) m[i][i] = 0;
    for (int r = 2; r <= n; r++)
      for (int i = 1; i <= n - r+1; i++) {
        int j=i+r-1;
        m[i][j] = m[i+1][j]+ p[i-1]*p[i]*p[j];
        s[i][j] = i;
        for (int k = i+1; k < j; k++) {
          int t = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
}
```



算法复杂度分析：

算法的主要计算量取决于算法中对r，i和k的3重循环。

循环体内的计算量为O(1)，而3重循环的总次数为O(n^3)。

因此算法的计算时间上界为O(n^3)。算法所占用的空间显然为O(n^2)。

矩阵连乘问题（动态规划）4

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25

$$p=\{p_0,p_1,p_2,p_3,p_4,p_5,p_6\}=\{30,35,15,5,10,20,25\}$$

$j \longrightarrow$

$i \downarrow$

0	15750	7875	9375	11875	15125
	0	2625	4375	7125	10500
		0	750	2500	5375
			0	1000	3500
				0	5000
					0

$(A_1A_2A_3)(A_4A_5A_6)$

$(A_1(A_2A_3))((A_4A_5)A_6)$

$$m(i,j)=\begin{cases} 0 & i=j \\ \min\limits_{i\le k<j}\{m(i,k)+m(k+1,j)+p_{i-1}p_kp_j\} & i<j \end{cases}$$

S[i, j]	数值
s12	1
s13	1
s14	3
s15	3
s16	3
s23	2
s24	3
s25	3
s26	3
s34	3
s35	3
s36	3
s45	4
s46	5
s56	5

矩阵连乘问题（动态规划）

$$m(i,j)=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m(i,k)+m(k+1,j)+p_{i-1}p_kp_j\} & i<j\end{cases}$$

$j \longrightarrow$

$i \downarrow$

0	15750	7875	9375		
	0	2625	4375	7125 $m[2][5]$	
		0	750	2500	
			0	1000	3500
				0	5000
					0

$$k=3\rightarrow s[2][5]=3$$

$$p=\{p_0,p_1,p_3,p_4,p_5,p_6\}=\{30,35,15,5,10,20,25\}$$

$$m[2][5]=\min\begin{cases}m[2][2]+m[3][5]+p_1p_2p_5=0+2500+35\times15\times20=13000\\ m[2][3]+m[4][5]+p_1p_3p_5=2625+1000+35\times5\times20=7125\\ m[2][4]+m[5][5]+p_1p_4p_5=4375+0+35\times10\times20=11375\end{cases}$$

动态规划方法

- 基本思想
  - 把原始问题划分成一系列子问题
  - 求解每个子问题仅一次，并将其结果保存在一个表中，以后用到时直接存取，
  - 自底向上地计算
- 适用范围
  - 一类优化问题：可分为多个相关子问题，子问题的解被重复使用

动态规划方法

❑ 适用条件

1. 优化子结构

   ✓ 当一个问题的优化解包含了子问题的优化解时，我们说这个问题具有优化子结构。

   ✓ 优化子结构使得我们能自下而上地完成求解过程

   ✓ 证明：首先假设由问题的最优解导出的子问题的解不是最优的，然后再设法说明在这个假设下可构造出比原问题最优解更好的解，从而导致矛盾。

动态规划方法

❑ 适用条件

2. 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

✓ 对每一个子问题只解一次，而后将其解保存在一个表格中，当再次需要解此子问题时，只是简单地用常数时间查看一下结果。

最长公共子序列问题

子序列

- 序列：X=<A，B，C，B，D，A，B>
- Z=<B，C，D，B>是X的子序列
- W=<C, A, B, D>不是X的子序列
- 若给定序列$X=\{x_1,x_2,\ldots,x_m\}$，则另一序列$Z=\{z_1,z_2,\ldots,z_k\}$ 是X的子序列，是指存在一个严格递增下标序列$\{i_1,i_2,\ldots,i_k\}$, 使得对于所有$j=1,2,\ldots,k$有：$z_j=x_{i_j}$。

最长公共子序列问题

- 公共子序列
  - X=<A，B，C，B，D，A，B>
  - Y=<B，C，D，B，E>
  - Z={B, C, D, B}是X和Y的公共子序列
- 最长公共子序列问题
  - 输入： $X=<x_1,x_2,\ldots,x_m>$， $Y=<y_1,y_2,\ldots,y_n>$
  - 输出：X和Y的最长公共子序列Z
- 穷举法
  - 找到X的所有子序列，检查是否是Y的子序列，在检查过程中记录最长的公共子序列
  - 很明显，共2^m个不同子序列，需要指数时间。

最长公共子序列问题

优化子结构 1

- 设序列$X=<x_1,\ldots,x_m>$和$Y=<y_1,\ldots,y_n>$的最长公共子序列是$Z=<z_1, \ldots, z_k>$
- 如果$x_m=y_n$
  - $z_k=x_m=y_n$
  - $<z1,\ldots, z_{k-1}>$是$<x_1,\ldots,x_{m-1}>$和$<y_1,\ldots,y_{n-1}>$的最长公共子序列
- 如果$x_m \neq y_n$且$z_k \neq x_m$
  - $<z_1,\ldots, z_k>$是$<x_1,\ldots,x_{m-1}>$和$<y_1,\ldots,y_n>$的最长公共子序列
- 如果$x_m \neq y_n$且$z_k \neq y_n$
  - $<z_1,\ldots, z_k>$是$<x_1,\ldots,x_m>$和$<y_1,\ldots,y_{n-1}>$的最长公共子序列

最长公共子序列问题

- 找$<x_1,\ldots,x_m>$和$<y_1,\ldots,y_n>$的最长公共子序列
  - 如果$x_m=y_n$
    - 找$<x_1,\ldots,x_{m-1}>$和$<y_1,\ldots,y_{n-1}>$的最长公共子序列，在其尾部加上x_m
  - 如果$x_m\neq y_n$
    - 找$<x_1,\ldots,x_{m-1}>$和$<y_1,\ldots,y_n>$的最长公共子序列
    - 找$<x_1,\ldots,x_m>$和$<y_1,\ldots,y_{n-1}>$的最长公共子序列
    - 取这两个公共序列的较长者

最长公共子序列问题

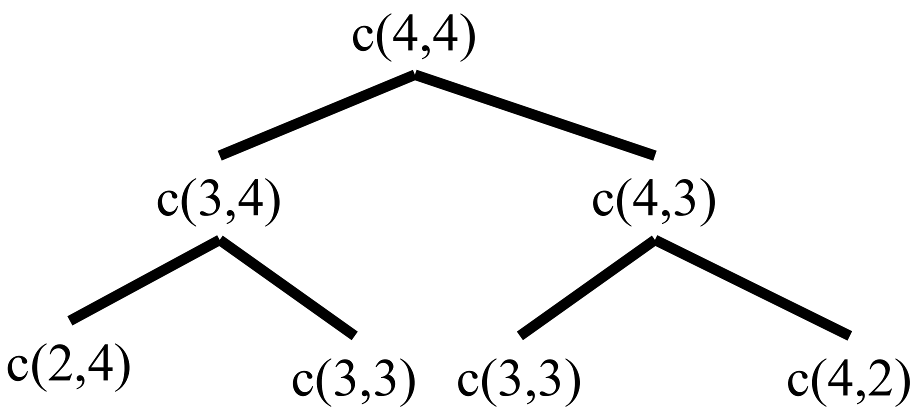
- 递归表达式 2
  - 用c(i, j)表示$<x_1,\ldots, x_i>$和$<y_1,\ldots,y_j>$的最长公共子序列的长度

$$
c(i,j)=\begin{cases} 0 & i=0\text{或}j=0 \\ c(i-1,j-1)+1 & i,j>0; x_i=y_j \\ \max\{c(i-1,j),c(i,j-1)\} & i,j>0; x_i\neq y_j \end{cases}
$$

最长公共子序列问题

- 重叠子问题

$$c(i,j)=\begin{cases}0 & i=0\text{或}j=0\\ c(i-1,j-1)+1 & i,j>0;x_i=y_j\\ \max\{c(i-1,j),c(i,j-1)\} & i,j>0;x_i\neq y_j\end{cases}$$



X长度m，Y长度n，那么总共有$\theta(mn)$个不同的子问题

最长公共子序列问题

求c[3,4]

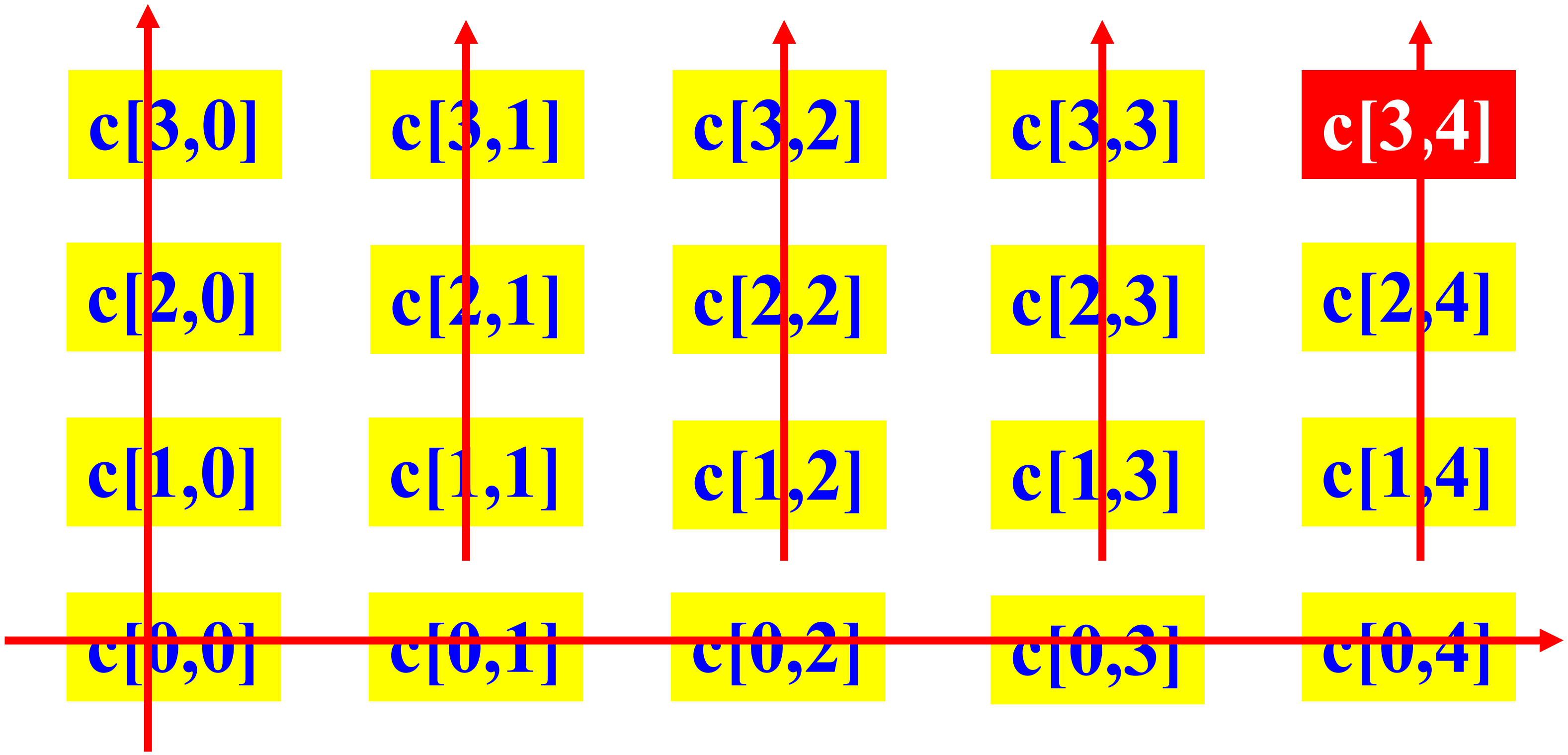
- 动态规划方法 3
  - 自底向上求解子问题
  - 子问题的解保存到表中

$$c(i,j)=\begin{cases}0 & i=0\text{或}j=0\\ c(i-1,j-1)+1 & i,j>0;x_i=y_j\\ \max\{c(i-1,j),c(i,j-1)\} & i,j>0;x_i\neq y_j\end{cases}$$

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

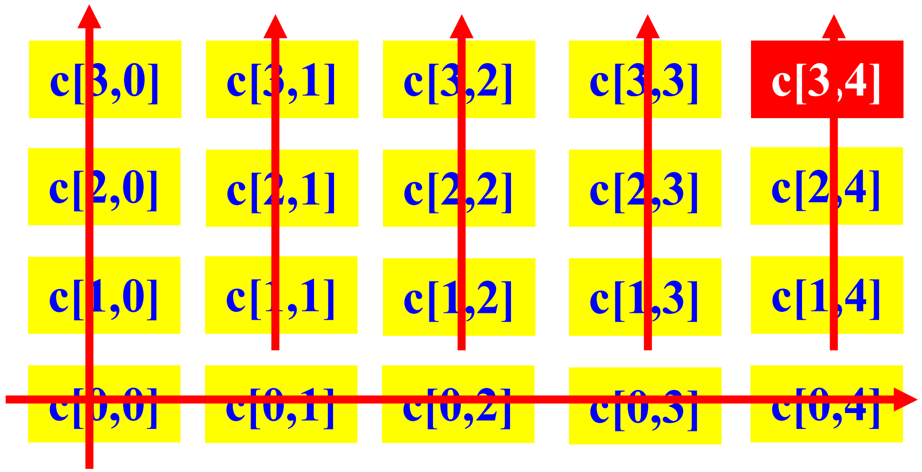
最长公共子序列问题

- 动态规划方法



最长公共子序列（计算最优值）

```
void LCSLength(int m， int n， char *x，
char *y， int **c， int **b)
{
      int i， j;
      for (i = 1; i <= m; i++) c[i][0] = 0;
      for (i = 1; i <= n; i++) c[0][i] = 0;
      for (i = 1; i <= m; i++)
        for (j = 1; j <= n; j++) {
          if (x[i]==y[j]) {
              c[i][j]=c[i-1][j-1]+1;
              b[i][j]=↙;}
          else if (c[i-1][j]>=c[i][j-1]) {
                  c[i][j]=c[i-1][j]; b[i][j]=↓;}
          else {c[i][j]=c[i][j-1]; b[i][j]=←;}
          }
}
```



$$c(i,j)=\begin{cases}0 & i=0\text{或}j=0\\ c(i-1,j-1)+1 & i,j>0;x_i=y_j\\ \max\{c(i-1,j),c(i,j-1)\} & i,j>0;x_i\neq y_j\end{cases}$$

算法复杂度分析：

算法的计算时间上界为 **O(mn)**。

最长公共子序列（构造最优解4）

```
void LCS(int i，int j，char *x，int **b)
{
    if (i ==0 || j==0) return;
    if (b[i][j]==↙){
      LCS(i-1，j-1，x，b);
      print x[i]; }
    else if (b[i][j]==↓)
      LCS(i-1，j，x，b);
    else
      LCS(i，j-1，x，b);
}
```

每次递归i、j减1，O（m+n）

➢例

X={B，D，C，A，B，A}

Y={A，B，C，B，D，A，B}

LCS= {B, C, B, A}

A	0	1	2	2	3	3	4	4
B	0	1	2	2	3	3	3	4
A	0	1	1	2	2	2	3	3
C	0	0	1	2	2	2	2	2
D	0	0	1	1	1	2	2	2
B	0	0	1	1	1	1	1	1
	0	0	0	0	0	0	0	0
		A	B	C	B	D	A	B

最长公共子序列（构造最优解）

```
void LCS(int i，int j，char *x，int **b)
{
    if (i ==0 || j==0) return;
    if (b[i][j]== ↙ ){
      LCS(i-1，j-1，x，b);
      print x[i]; }
    else if (b[i][j]== ↓)
      LCS(i-1，j，x，b);
    else
      LCS(i，j-1，x，b);
}
```

每次递归i、j减1，O（m+n）

➤例

X={B，D，C，A，B，A}

Y={A，B，C，B，D，A，B}

LCS= {B, C, B, A}

A	0	1	2	2	3	3	4	4
B	0	1	2	2	3	3	3	4
A	0	1	1	2	2	2	3	3
C	0	0	1	2	2	2	2	2
D	0	0	1	1	1	2	2	2
B	0	0	1	1	1	1	1	1
	0	0	0	0	0	0	0	0
		A	B	C	B	D	A	B

最长公共子序列（算法的改进）

哈尔滨工程大学

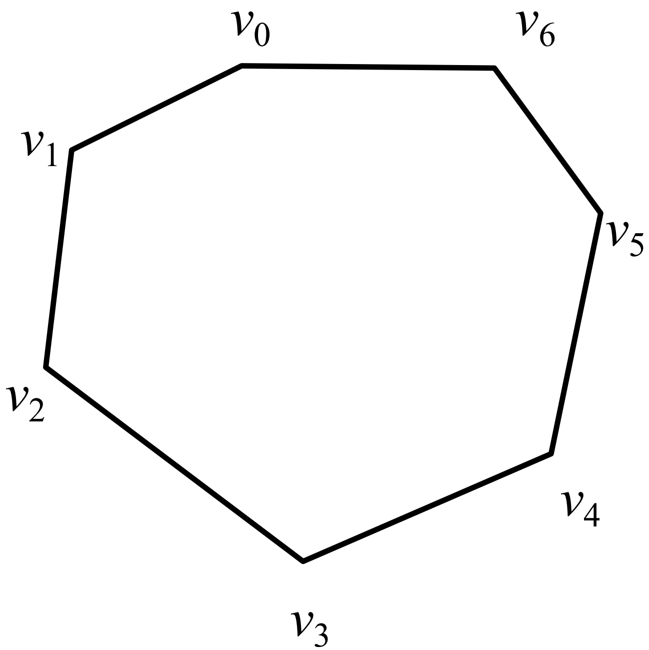
1. 将数组b省去，c(i, j)只和三个数组元素值有关，借助c本身在常数时间内确定到底是由哪个而确定。

$$c(i,j)=\begin{cases} 0 & i=0\text{或}j=0 \\ c(i-1,j-1)+1 & i,j>0;x_i=y_j \\ \max\{c(i-1,j),c(i,j-1)\} & i,j>0;x_i\neq y_j \end{cases}$$

2. 修改LCS，不使用数组b可以在O(m+n)时间构造最长公共子序列。从而节省Θ（mn）的空间。
3. 数组c仍然需要Θ（mn）的空间，实际只对常数因子优化。
4. 如果只求子序列长度，事实上c（i，j）只需要数组c的i和i-1行。空间可进一步减小到O（min{m，n}）

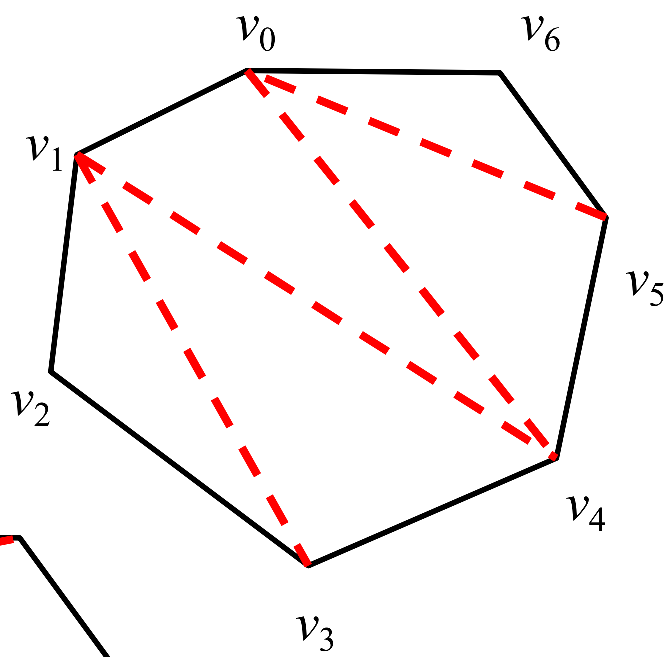
凸多边形最优三角剖分

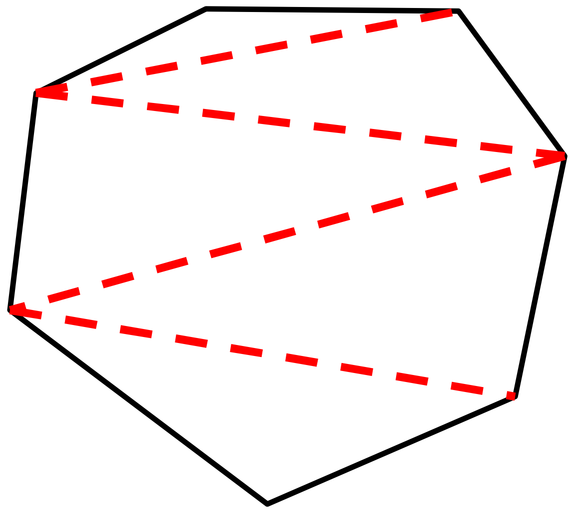
- 多边形
  - n个顶点的多边形P可表示为其顶点序列P={v_0, v_1,…, v_{n-1}}
  - 内部、边界、外部
- 凸多边形
  - 边界上或内部任意两点连成的线段都在其内部或边界上



凸多边形最优三角剖分

- 弦
  - 连接多边形上不相邻顶点v_i和v_j的线段
- 三角剖分
  - 将多边形划分为不相交的三角形的弦的集合



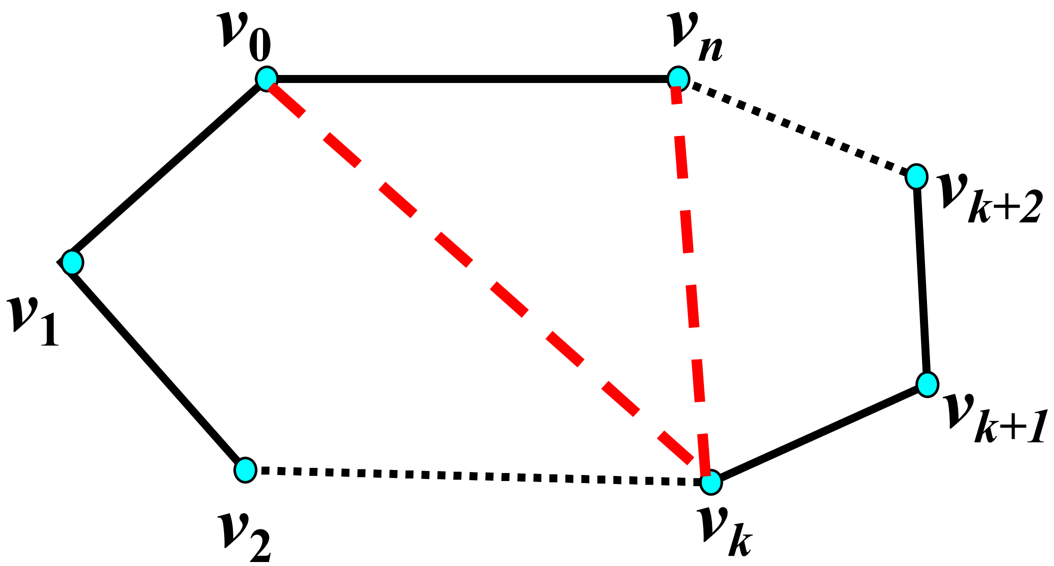


凸多边形最优三角剖分

- 凸多边形的最优三角剖分问题
  - 输入：凸多边形P和代价函数w
    - w指定了每个三角形的代价
    - *如*$w(v_i v_j v_k)=|v_i v_j|+|v_j v_k|+|v_k v_i|$
  - 输出：P的三角剖分T，使得 $\sum_{s \in S_T} w(s)$ 最小
    - S_T是T所对应的三角形集合

凸多边形最优三角剖分

- 优化子结构1
  - $P=(v_0,v_1,\ldots,v_n)$是$n+1$个顶点的凸多边形
  - T_P是P的优化三角剖分，包含三角形$v_0v_k v_n$



$$T_P=T(v_0, \ldots, v_k)\cup\ T(v_k, \ldots, v_n)\cup\ \{v_0v_kv_n\}$$

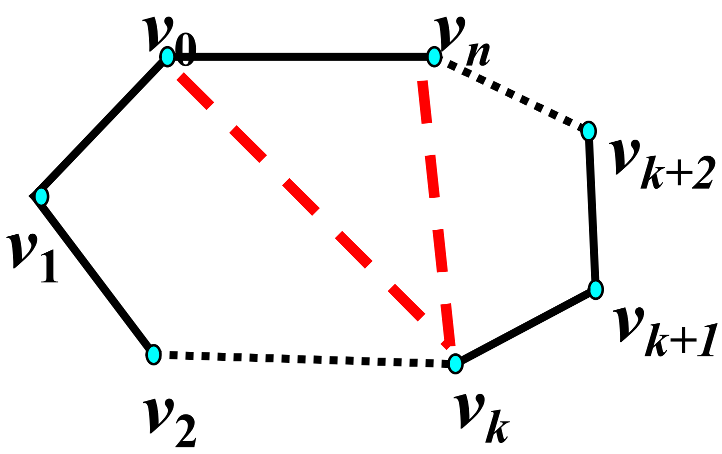
凸多边形最优三角剖分

- 定理（优化子结构）

  设$P=(v_0,v_1,\ldots v_n)$是$n+1$个顶点的凸多边形. 如果T_P是P的最优三角剖分且包含三角形$v_0v_k v_n$，即

  $T_P=T(v_0, \ldots, v_k)\cup T(v_k, \ldots, v_n)\cup \{v_0v_kv_n\}$ 则

  - (1). $T(v_0, \ldots, v_k)$是$P_1=(v_0,v_1, \ldots,v_k)$的优化三角剖分,
  - (2). $T(v_k, \ldots, v_n)$是$P_2=(v_k,v_{k+1},\ldots,v_n)$的优化三角剖分

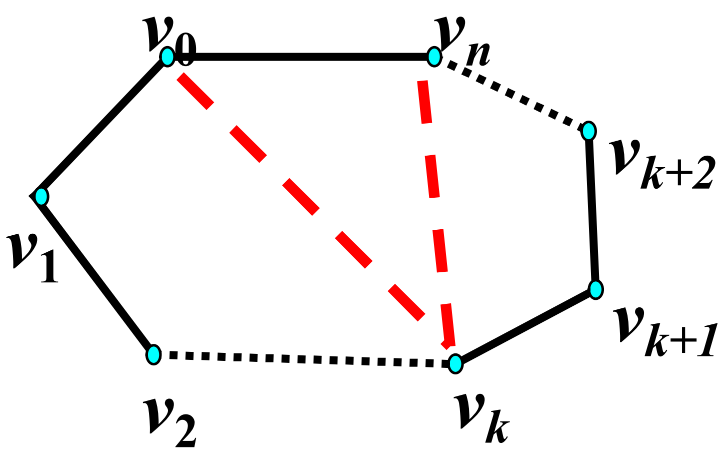


凸多边形最优三角剖分

最优三角剖分的递归结构2

- 设$t(i, j)$为凸多边形$\{v_{i-1}, v_i \dots, v_j\}$最优三角剖分的代价
- 为方便，设退化的多边形$\{v_{i-1}, v_i\}$具有权值0(i=j时)

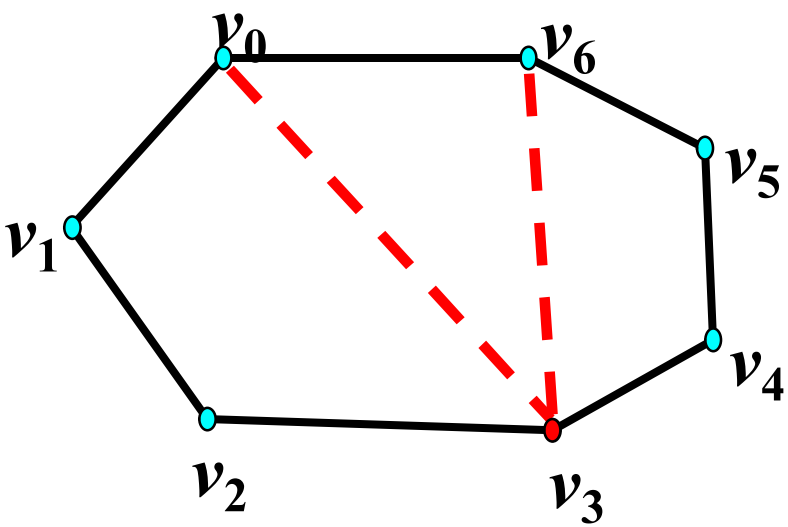
$$t(i,j)=\begin{cases} 0 & i=j \\ \min\limits_{i\le k<j}\{t(i,k)+t(k+1,j)+w(v_{i-1}v_kv_j)\} & i<j \end{cases}$$

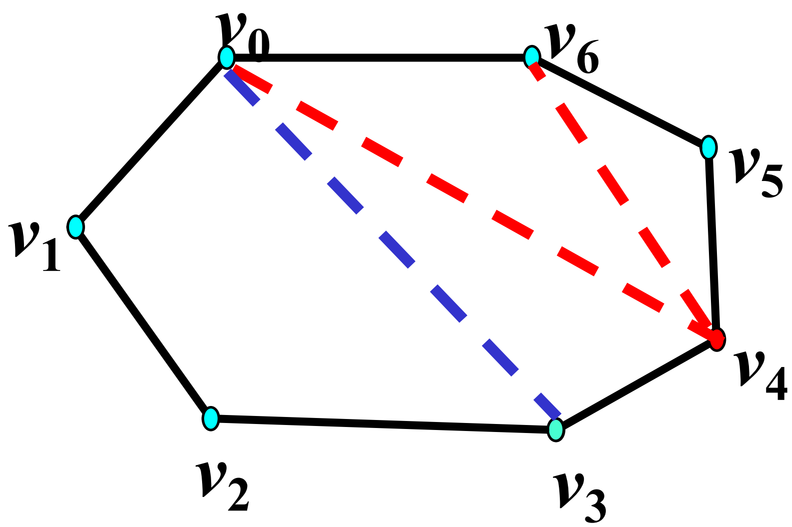


凸多边形最优三角剖分 $t(i, j) \to \{v_{i-1}, v_i \ \cdots, v_j\}$

$$t(i,j)=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{t(i,k)+t(k+1,j)+w(v_{i-1}v_kv_j)\} & i<j\end{cases}$$

❑ 重叠子问题





$i=1, k=3, j=6$

$t(1,6)=t(1,3)+t(4,6)+w\{v_0v_3v_6\}$

$i=1, k=4, j=6$

$t(1,6)=t(1,4)+t(5,6)+w\{v_0v_4v_6\}$

$i=1, k'=3, j=4$

$t(1,4)=t(1,3)+t(4,5)+w\{v_0v_4v_5\}$

凸多边形最优三角剖分

- 动态规划方法
  - 与矩阵连乘方法一致 34

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

计算t（1， 5）

$$t(i,j)=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{t(i,k)+t(k+1,j)+w(v_{i-1}v_kv_j)\} & i<j\end{cases}$$

凸多边形最优三角剖分

- 动态规划方法

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

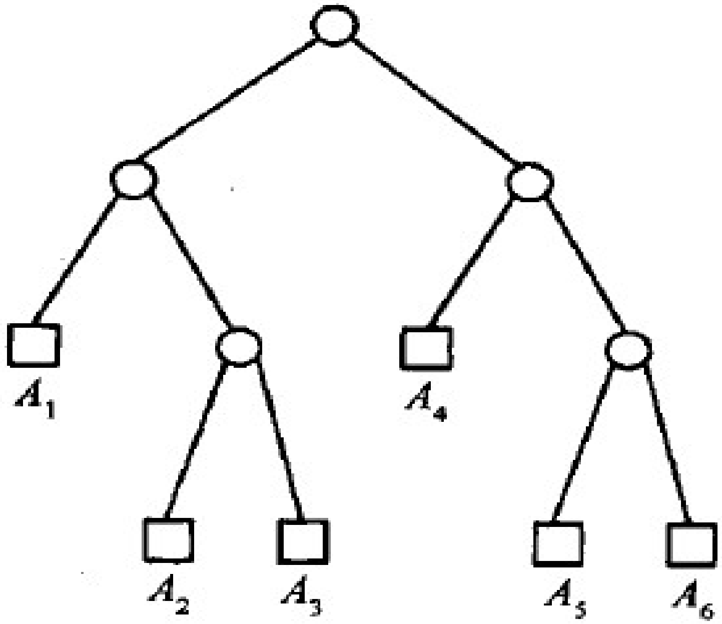
计算t(1，5)

凸多边形最优三角剖分

关系

- 一个表达式的完全加括号方式对应一颗完全二叉树，称为语法树（从哪里加括号，哪里就有一个结点），完全二叉树的n个叶结点表示表达式的n个原子。

$(A_1(A_2A_3))((A_4A_5)A_6)$

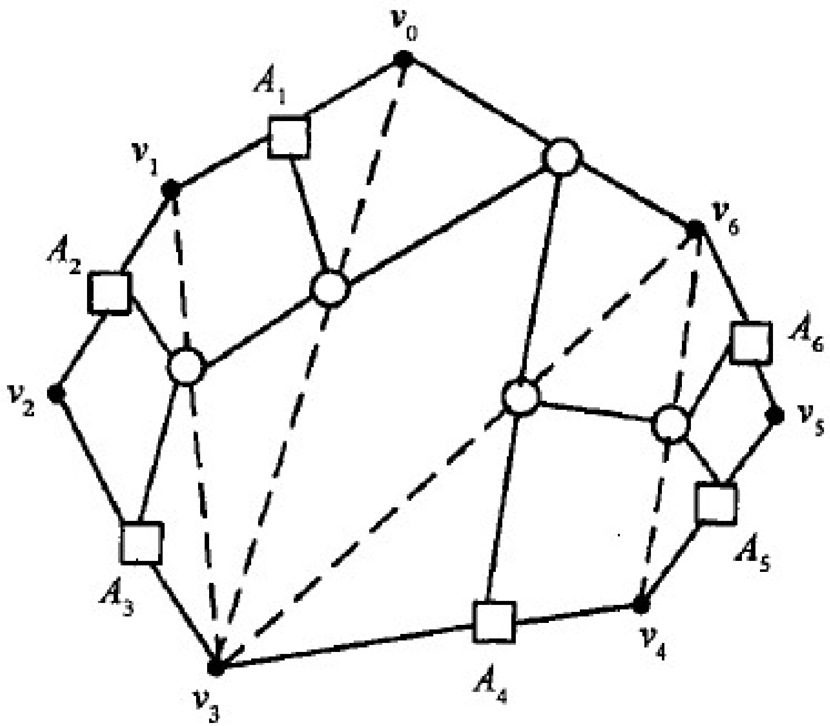


以某个结点为根的子树，表示为其左子树和右子树的乘积。

凸多边形最优三角剖分

关系

- 凸多边形$(v_0,v_1,\ldots,v_{n-1})$ 三角剖分问题也可以用语法树表示

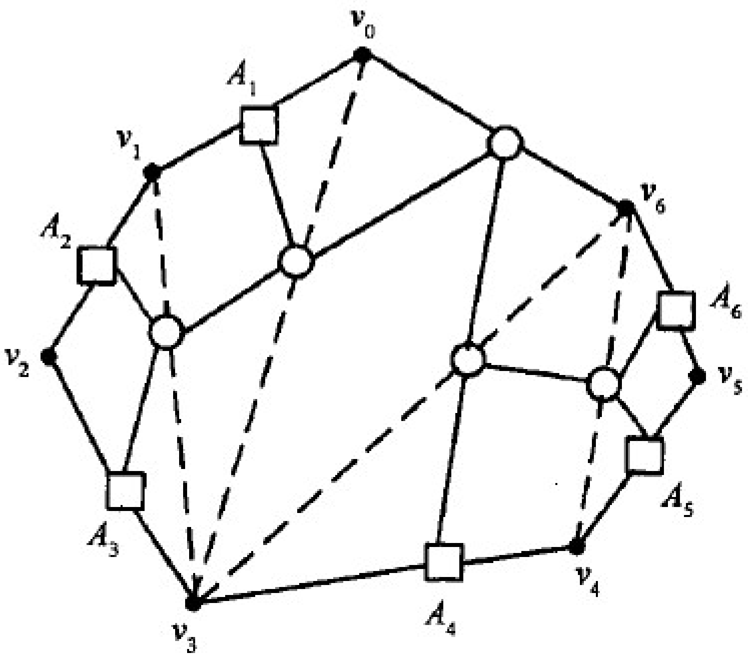


1. 根节点v_0v_6，三角剖分的弦组成其余内节点。
2. 多边形除v_0v_6的边都是语法树的一个叶结点，树根v_0v_6是三角形$v_0v_3v_6$的一条边，该三角形把原三角形分成三部分：三角形$v_0v_3v_6$，凸多边形$\{v_0v_1\cdots v_3\}$和凸多边形$\{v_3v_1\cdots v_6\}$。
3. 三角形$v_0v_3v_6$的另外两条边，即以弦v_0v_3弦v_3v_6为根的两个儿子，表示凸多边形$\{v_0v_1\cdots v_3\}$和凸多边形$\{v_3v_1\cdots v_6\}$的三角剖分。

凸多边形最优三角剖分

❑ 关系

- 凸多边形$(v_0,v_1,\ldots,v_{n-1})$ 三角剖分问题也可以用语法树表示



4. 对于凸n多边形的三角剖分与n-1个叶结点的语法树一一对应。
5. N个矩阵的完全加括号连乘积与n个叶结点的语法树一一对应。
6. N个矩阵的连乘积与凸n-1多边形一一对应。
7. 矩阵连乘的最优计算次序问题是凸多边形最优三角剖分问题的特殊情况。对于最优三角剖分问题，代价函数是任意的。

0-1背包问题

- 0-1背包问题
  - 输入：n 种物品和一个背包
    - 物品 i 的重量是 w_i，价值为 v_i
    - 背包的容量是 C
  - 输出：装入背包的物品
  - 优化目标：装入背包的物品**总价值**最大

0-1背包问题

形式化描述

- 输入：$\{<w_1, v_1>, <w_2, v_2>, \ldots, <w_n, v_n>\}$和$C$
- 输出：$(x_1, x_2, \ldots, x_n)$，$x_i \in \{0, 1\}$满足 $\sum_{i=1}^{n} w_i x_i \le C$
- 优化目标：$\max \sum_{i=1}^{n} v_i x_i$

等价于整数规划问题

$$\max \sum_{i=1}^{n} v_i x_i$$

规划中的变量（全部或部分，x_i）限制为整数。

$$\begin{cases} \sum_{i=1}^{n} w_i x_i \le C \\ x_i \in \{0,1\},\ 1 \le i \le n \end{cases}$$

0-1背包问题

最优子结构1

- 设 $(x_1, x_2, \ldots, x_n)$ 是0-1背包问题的一个最优解

如果$x_1=1$：则 $(x_2, \ldots, x_n)$ 是以下子问题的最优解

$$\max \sum_{i=2}^{n} v_i x_i$$

$$\begin{cases} \sum_{i=2}^{n} w_i x_i \le C - w_1 \\ x_i \in \{0,1\},\ 2 \le i \le n \end{cases}$$

如果$x_1=0$：则 $(x_2, \ldots, x_n)$ 是以下子问题的最优解

$$\max \sum_{i=2}^{n} v_i x_i$$

$$\begin{cases} \sum_{i=2}^{n} w_i x_i \le C \\ x_i \in \{0,1\},\ 2 \le i \le n \end{cases}$$

x_i = 0 or 1，将问题完整的分成两个子问题，原问题的最优解包含子问题的最优解。

0-1背包问题

递归关系2

- 设$m(i,j)$为背包容量为j，可选物品为$i, i+1, \ldots, n$是01背包问题的最优值，
- 有如下递归关系(分析第i个物品是否能装进去，到底装还是不装0 or 1)
  - $i<n$时（可选有i：n）

$$m(i,j)=\begin{cases}\max\{m(i+1,j),\ m(i+1,j-w_i)+v_i\} & j\geq w_i\\ m(i+1,j) & 0\leq j<w_i\end{cases}$$

  - $i=n$时（可选只有n）

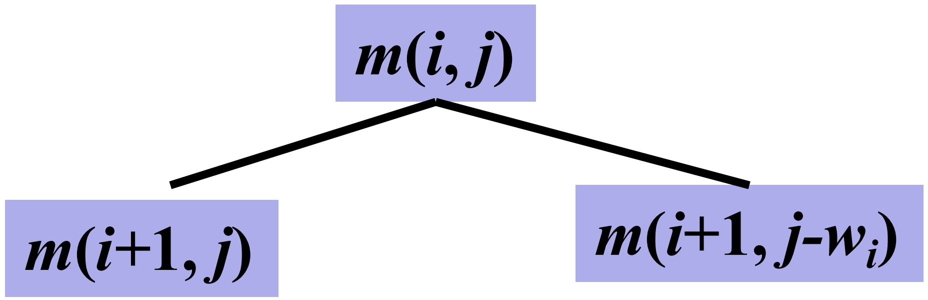
$$m(n,j)=\begin{cases}v_n & j\geq w_n\\ 0 & 0\leq j<w_n\end{cases}$$

0-1背包问题

递归关系

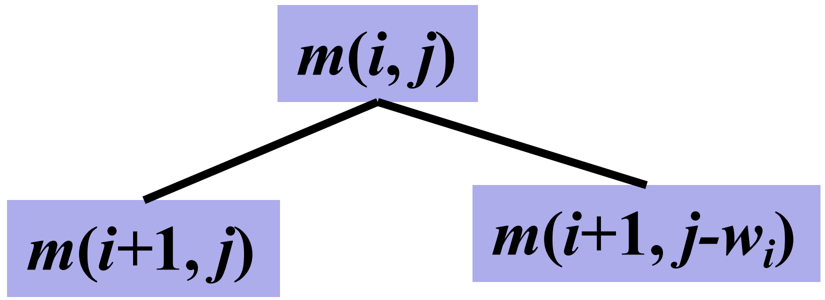
- $i<n$时

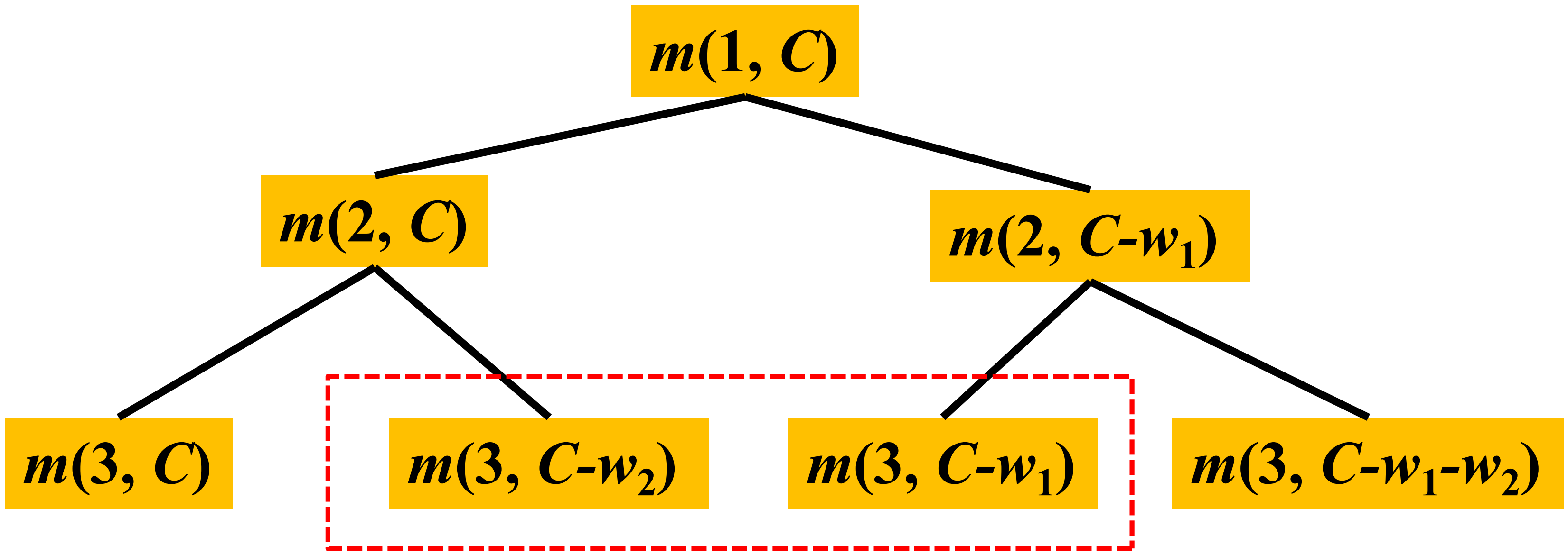
$$m(i,j)=\begin{cases}\max\{m(i+1,j),\ \ m(i+1,j-w_i)+v_i\} & j\ge w_i\\ m(i+1,j) & 0\le j<w_i\end{cases}$$



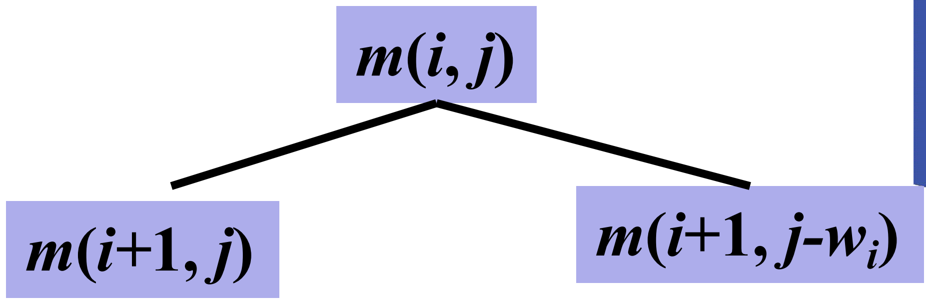
0-1背包问题

- 重叠子问题
  - 假设$w_1=w_2=w$



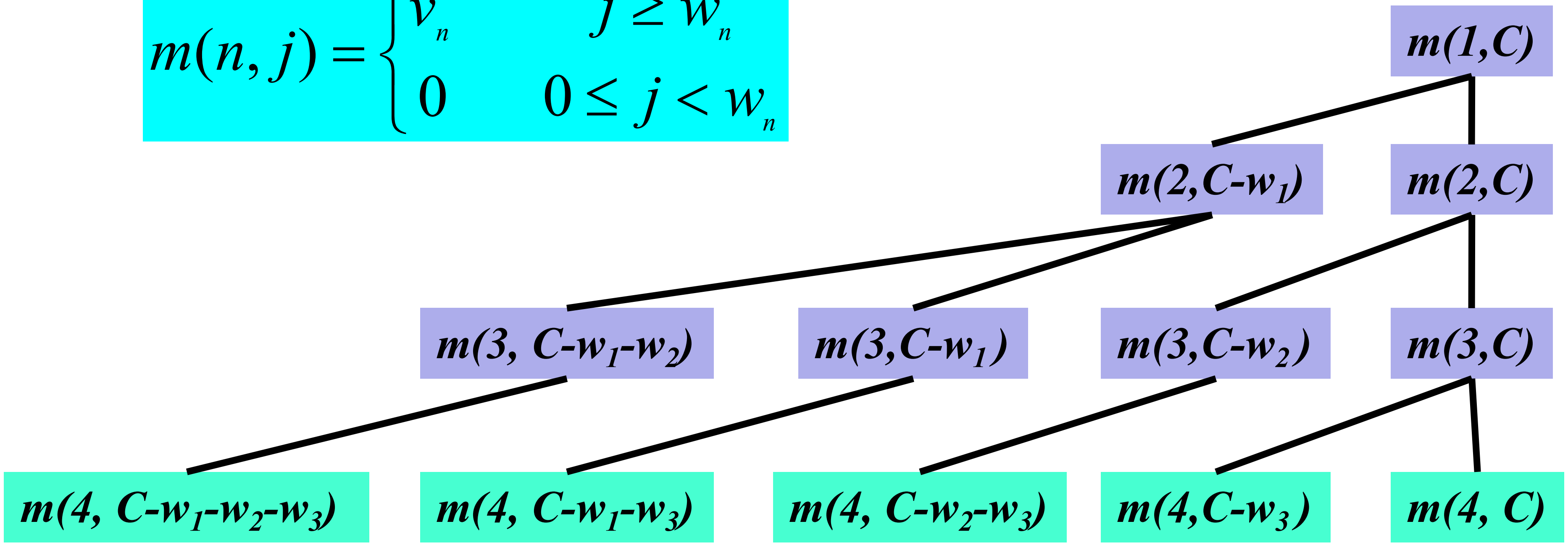


0-1背包问题



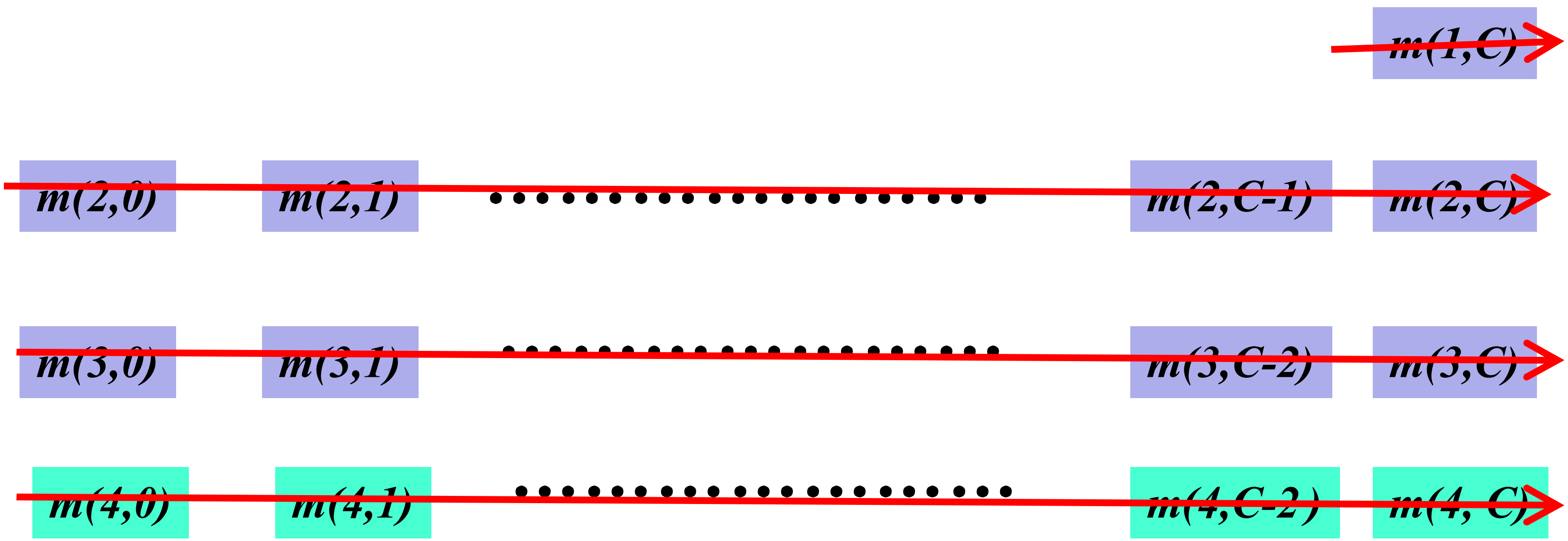
- 动态规划方法
  - 令w_i为整数，n=4

$$m(n, j) = \begin{cases} v_n & j \ge w_n \\ 0 & 0 \le j < w_n \end{cases}$$



0-1背包问题（动态规划方法）

- 令w_i为整数，n=4
- 自底向上计算最优解(i--, j++)



0-1背包问题（动态规划方法）

$$m(n,j)=\begin{cases} v_n & j \ge w_n \\ 0 & 0 \le j < w_n \end{cases}$$

算法：

```
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};
```

装入n
$C<= w_n-1$，装不下
$C>w_n-1$　　$(j=w_n)$

依次装入i=n-1….2
i--

j++

装入1,i=1

$$m(i,j)=\begin{cases} \max\{m(i+1,j),\ m(i+1,j-w_i)+v_i\} & j \ge w_i \\ m(i+1,j) & 0 \le j < w_i \end{cases}$$

0-1背包问题（动态规划方法）

❐ 构造最优解

1. $m(1, C)$是最优解代价值，相应解计算如下:

   If $m(1, C) = m(2, C)$

   Then $x_1 = 0$;

   Else $x_1 = 1$;

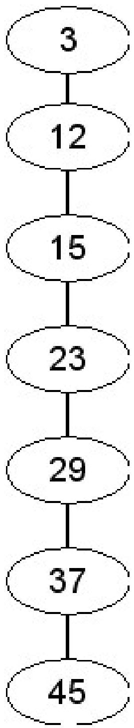
2. 如果 $x_1=0$，由$m(2, C)$继续构造最优解;
3. 如果 $x_1=1$，由$m(2, C-w_1)$继续构造最优解.

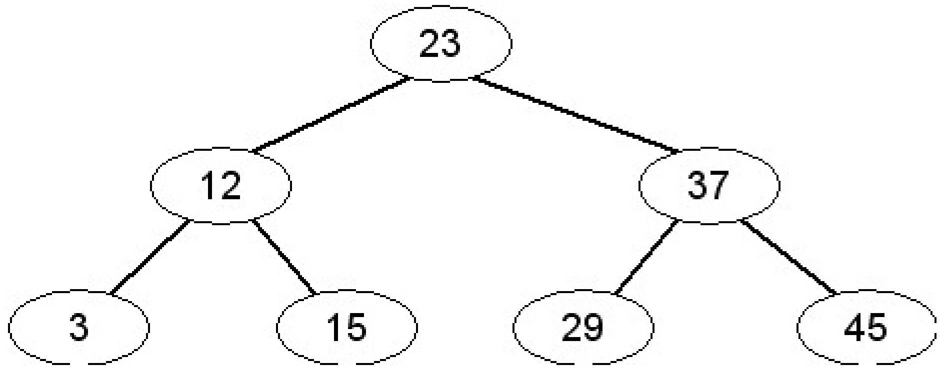
0-1背包问题（动态规划方法）

- 时间复杂性
  - 计算代价的时间
    - $O(Cn)$
  - 构造最优解的时间: $O(n)$
  - 总时间复杂性为：$O(Cn)$
- 空间复杂性
  - $O(Cn)$

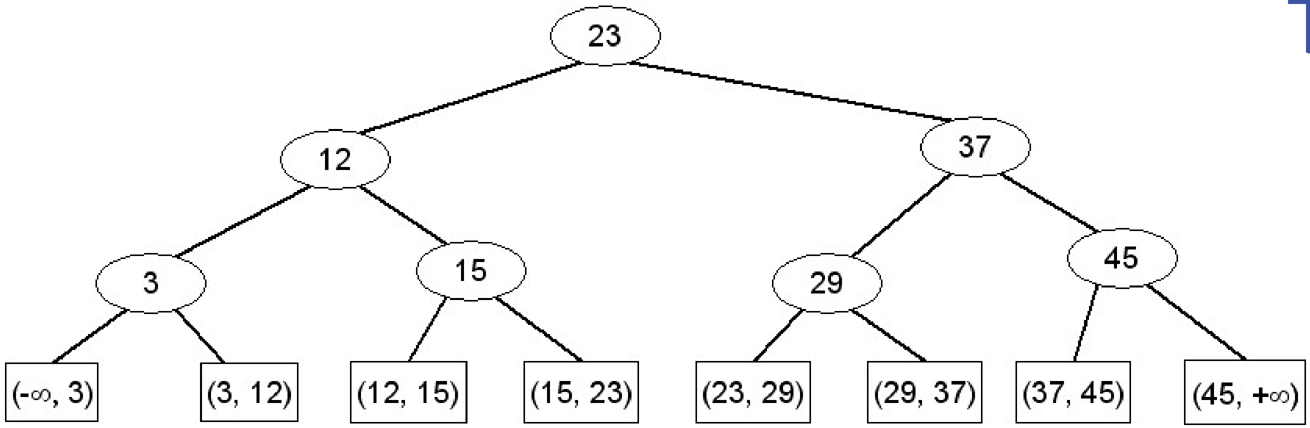
最优二叉搜索树

- 搜索问题
  - 给定有序数组，例如a[]=3, 12, 15, 23, 29, 37, 45
  - 判断元素x是否在数组中，及其在数组中（待插入）的位置





最优二叉搜索树



- 二叉搜索树
  - 内节点：$x_1, x_2, \ldots, x_n$
  - 叶节点：$(-\infty, x_1), (x_1, x_2), \ldots, (x_n, +\infty)$
  - 对于任意内节点x
    - 左子树中的元素都 小于x
    - 右子树中的元素都大于x

最优二叉搜索树

最优二叉搜索问题

$$\sum_{i=1}^{n} p_i + \sum_{j=0}^{n} q_j = 1$$

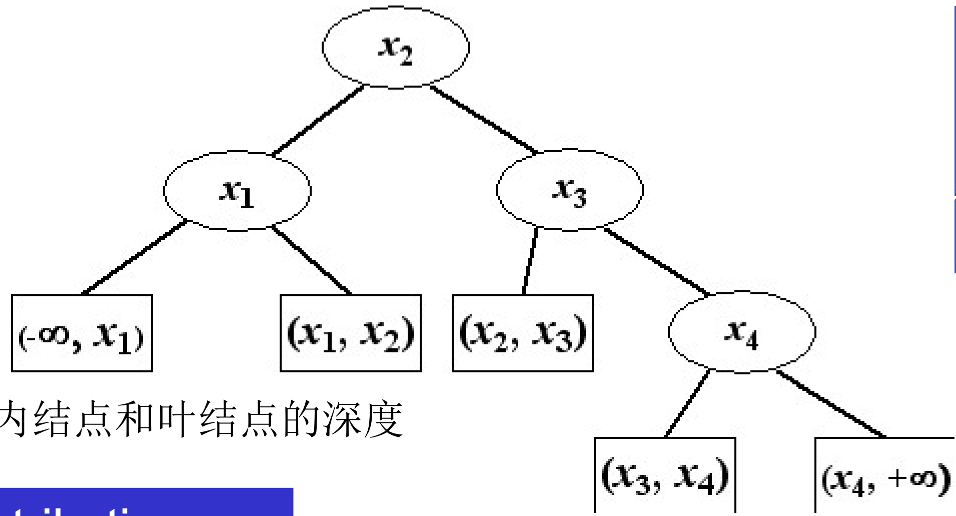
- 输入
  - 有序数集合 $\{x_1, x_2, \ldots, x_n\}$

Node	$(-\infty, x_1)$	x_1	(x_1, x_2)	x_2	…	x_n	$(x_n, +\infty)$
Probability	q_0	p_1	q_1	p_2	…	p_n	q_n

- 输出：最优二叉搜索树
  - 在二叉树中查找的平均路长（比较次数）最小

$$E(T) = \sum_{i=1}^{n} p_i(1+c_i) + \sum_{j=0}^{n} q_j d_j,$$ ci和di分别为内结点和叶结点的深度

最优二叉搜索树



$$E(T) = \sum_{i=1}^{n} p_i (1 + c_i) \ + \sum_{j=0}^{n} q_j d_j,$$ ci和di分别为内结点和叶结点的深度

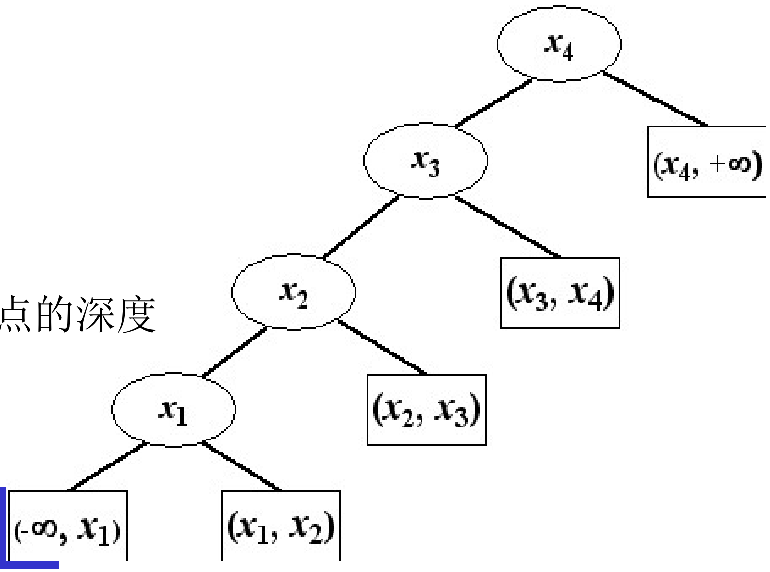
Node	Probability	Depth	Contribution
$x1$	0.05	1	0.1
$x2$	0.05	0	0.05
$x3$	0.1	1	0.2
$x4$	0.3	2	0.9
$(-\infty, x1)$	0.05	2	0.1
$(x1, x2)$	0.05	2	0.1
$(x2, x3)$	0.05	2	0.1
$(x3, x4)$	0.05	3	0.15
$(x4, +\infty)$	0.3	3	1.2

实结点贡献=（深度+1)*概率

虚结点贡献=深度*概率

Expectation
2.9

最优二叉搜索树



$$E(T)=\sum_{i=1}^{n} p_i(1+c_i)\ +\sum_{j=0}^{n} q_j d_j,$$ ci和di分别为内结点和叶结点的深度

Node	Probability	Depth	Contribution
$x1$	0.05	3	0.2
$x2$	0.05	2	0.15
$x3$	0.1	1	0.2
$x4$	0.3	0	0.3
$(-\infty, x1)$	0.05	4	0.2
$(x1, x2)$	0.05	4	0.2
$(x2, x3)$	0.05	3	0.15
$(x3, x4)$	0.05	2	0.1
$(x4, +\infty)$	0.3	1	0.3

实结点贡献=（深度+1)*概率

虚结点贡献=深度*概率

Expectation
1.8

Node	$(-\infty, x_1)$	x_1	(x_1, x_2)	x_2	…	x_n	$(x_n, +\infty)$
Probability	q_0	p_1	q_1	p_2	…	p_n	q_n

搜索树的代价

- 设搜索树T中
  - 树中x_i节点的深度为c_i
  - 叶节点(x_i, x_{i+1})的深度为d_i
- T的代价

$$E(T)=\sum_{i=1}^{n} p_i(1+c_i) + \sum_{j=0}^{n} q_j d_j$$

最优二叉搜索树

最优二叉搜索问题

- 输入
  - 有序数集合$\{x_1, x_2, \ldots, x_n\}$

Node	$(-\infty,\ x_1)$	x_1	(x_1, x_2)	x_2	…	x_n	$(x_n, +\infty)$
Weight	q_0	p_1	q_1	p_2	…	p_n	q_n

- 输出：二叉搜索树T
  - 树中x_i节点的深度为c_i
  - 叶节点$(x_i,\ x_i+1)$的深度为d_i
- 优化目标：最小化代价函数

$$E(T) = \sum_{i=1}^{n} p_i(1+c_i) + \sum_{j=0}^{n} q_j d_j$$

最优二叉搜索树

- 设T是针对输入 $E(T)=\sum_{i=1}^{n} p_i(1+c_i)+\sum_{j=0}^{n} q_j d_j$，ci和di分别为内结点和叶结点的深度

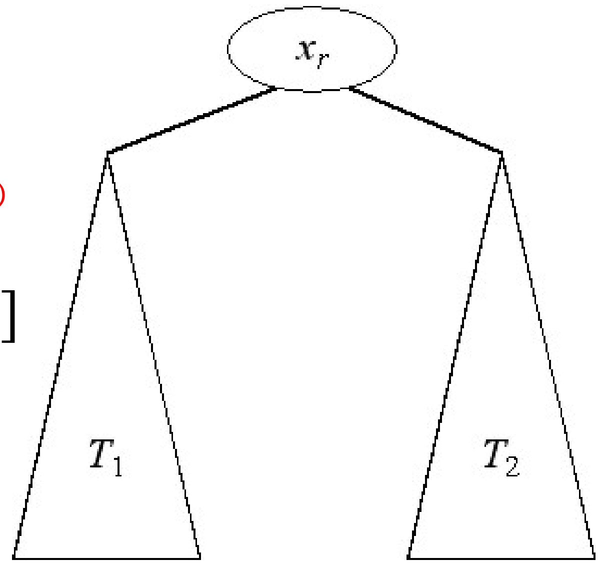
Node	(x_{i-1}, x_i)	x_i	…	x_j	(x_j, x_{j+1})
Weight	q_{i-1}	p_i	…	p_j	q_j

的最优二叉搜索树，且T以x_r为根

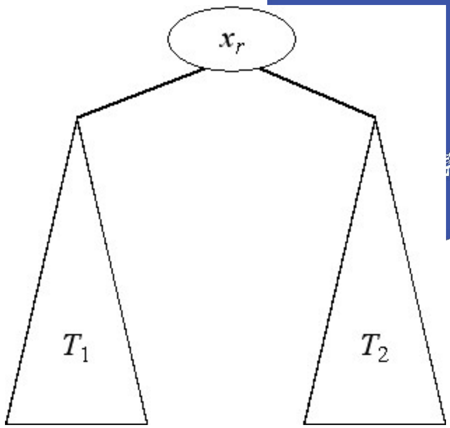
- 设 $w(i,j)=q_{i-1}+p_i+\ldots+p_j+q_j$
- 则，由于T1 和T2为子树，单独考虑时，结点深度减1（补）

$$E(T)=p_r+[E(T_1)+w(i,r-1)]+[E(T_2)+w(r+1,j)]$$

$$E(T)=w(i,j)+E(T_1)+E(T_2)$$



最优二叉搜索树



- 优化子结构
  - 如果T是针对以下输入的最优二叉搜索树

Node	(x_{i-1}, x_i)	x_i	…	x_j	(x_j, x_{j+1})
Weight	q_{i-1}	p_i	…	p_j	q_j

  - 那么
    - T_1是针对以下输入的最优二叉搜索树

Node	(x_{i-1}, x_i)	x_i	…	x_{r-1}	(x_{r-1}, x_r)
Weight	q_{i-1}	p_i	…	p_{r-1}	q_{r-1}

    - T_2是针对以下输入的最优二叉搜索树

Node	(x_r, x_{r+1})	x_{r+1}	…	x_j	(x_j, x_{j+1})
Weight	q_r	p_{r+1}	…	p_j	q_j

最优二叉搜索树

- 设$E(i, j)$是针对以下输入构造的最优二叉搜索树的代价

Node	(x_{i-1}, x_i)	x_i	…	x_j	(x_j, x_{j+1})
Weight	q_{i-1}	p_i	…	p_j	q_j

- 假设这棵树以x_r $(i \le r \le j)$为根，则

$$E(i,j) = p_r + [E(i,r-1) + w(i,r-1)] + [E(r+1,j) + w(r+1,j)]$$

$$E(i,j) = w(i,j) + E(i,r-1) + E(r+1,j)$$

最优二叉搜索树

❑ 设$E(i, j)$是针对以下输入构造的最优二叉搜索树的代价

Node	(x_{i-1}, x_i)	x_i	…	x_j	(x_j, x_{j+1})
Weight	q_{i-1}	p_i	…	p_j	q_j

❍ 如果不知道根节点，则

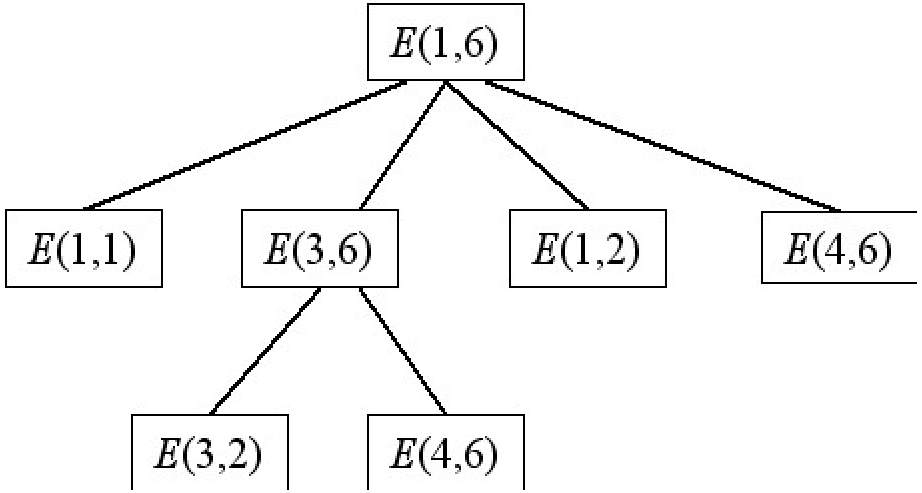
$$E(i,j)=\begin{cases} 0 & j=i-1 \\ \min\limits_{i\le r\le j}\{E(i,r-1)+E(r+1,j)+w(i,j)\} & i\le j \end{cases}$$

- 其中，$E(i, i\text{-}1)$是针对以下输入的最优二叉搜索树

Node	(x_{i-1}, x_i)
Weight	q_{i-1}

最优二叉搜索树

❑ 重叠子问题



$$E(i,j)=\begin{cases} 0 & j=i-1 \\ \min\limits_{i\le r\le j}\{E(i,r-1)+E(r+1,j)+w(i,j)\} & i\le j \end{cases}$$

最优二叉搜索树

- 动态规划方法
  - 求$E(1,4)$

$$E(i,j)=\begin{cases} 0 & j=i-1 \\ \min\limits_{i\le r\le j}\{E(i,r-1)+E(r+1,j)+w(i,j)\} & i\le j \end{cases}$$

$E(1,0)$	$E(1,1)$	$E(1,2)$	$E(1,3)$	$E(1,4)$
	$E(2,1)$	$E(2,2)$	$E(2,3)$	$E(2,4)$
		$E(3,2)$	$E(3,3)$	$E(3,4)$
			$E(4,3)$	$E(4,4)$
				$E(5,4)$

最优二叉搜索树

- 算法
  - 数据结构
    - $E[1:n+1;\ 0:n]$: 存储优化解搜索代价
    - $W[1: n+1;\ 0:n]$: 存储代价增量
    - $Root[1:n;\ 1:n]$: $Root(i, j)$记录子问题 $\{x_i, \ldots, x_j\}$优化解的根

最优二叉搜索树

```
Optimal-BST(p, q, n)
 For  i=1  To  n+1   Do
      E(i, i-1) = 0;
      W(i, i-1) = q_{i-1};
 For  l=1  To  n   Do
      For   i=1   To   n-l+1   Do
           j=i+l-1;
           E(i, j)=∞;
           W(i, j)=W(i, j-1)+p_j+q_j;
           For  r=i  To  j   Do
                t=E(i, r-1)+E(r+1, j)+W(i, j);
                If   t<E(i, j)
                Then  E(i, j)=t;  Root(i, j)=r;
 Return  E and  Root
```

最优二叉搜索树

- 时间复杂性
  - (l, i, r)三层循环，每层循环至多n步
  - 时间复杂性为$O(n^3)$
- 空间复杂性
  - 二个$(n+1)\times(n+1)$数组，一个$n\times n$数组
  - $O(n^2)$