HIT
CS&E

第四章 动态规划

张炜
计算机科学与工程系

提要

4.1 动态规划原理
4.2 最长公共子序列
4.3 矩阵链乘法
4.4 0/1背包问题
4.5 最优二叉搜索树
4.6 凸多边形的三角剖分

参考资料

《Introduction to Algorithms》

- 第15章

4.1 动态规划技术的基本要素

为什么引入动态规划?
什么是动态规划?
如何进行动态规划?

Why?

- 分治技术的问题
  - 子问题是相互独立的
  - 如果子问题不是相互独立的，分治方法将重复计算公共子问题，效率很低
- 例如，计算斐波那契数列的第n项
  - $F(0)=F(1)=1$
  - $F(n)=F(n-1)+F(n-2)$

Why?

- 分治技术的问题
  - 子问题是相互独立的
  - 如果子问题不是相互独立的，分治方法将重复计算公共子问题，效率很低
- 分治算法

算法$F(n)$

输入:非负整数n

输出:斐波那契数列第n项

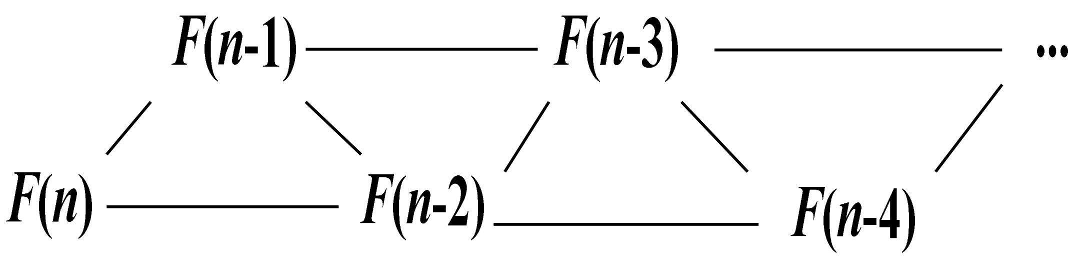
1. If n=0 或1 Then 输出1,算法结束
2. $f_1 \leftarrow F(n-1)$;
3. $f_2 \leftarrow F(n-2)$;
4. 输出f_1+f_2;

时间复杂性

$T(1)=T(0)=1$

$T(n)=T(n-1)+T(n-2)$

$T(n)$不是多项式有界的



Why?

- 提高效率的方法
  - 从规模最小的子问题开始计算
  - 用恰当数据结构存储子问题的解，供以后查询
  - 确保每个子问题只求解一次
- 算法

时间复杂性

$T(n)=\Theta(n)$

算法$F(n)$

输入:非负整数n

输出:斐波那契数列第n项

1. $A[0] \leftarrow 1$; $A[1] \leftarrow 1$;
2. For i=2 To n
3. $A[i] \leftarrow A[i-1]+A[i-2]$;
4. 输出$A[n]$;

Why?

- 分治技术的问题
  - 子问题是相互独立的
  - 如果子问题不是相互独立的，分治方法将重复计算公共子问题，效率很低
- 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
  - 很多优化问题可分为多个子问题，子问题相互关联，子问题的解被重复使用

What?

- 动态规划算法特点
  - 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，不重复计算，节省计算时间
  - 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

How?

- 使用Dynamic Programming的条件
  - **Optimal substructure**（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。
    - 缩小子问题集合，只需那些优化问题中包含的子问题，减低实现复杂性
    - 优化子结构使得我们能自下而上地完成求解过程
  - **Subteties**（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

- 动态规划算法的设计步骤
  - 分析优化解的结构
  - 递归地定义最优解的代价
  - 自底向上地计算优化解的代价保存之，并获取构造最优解的信息
  - 根据构造最优解的信息构造优化解

4.2 最长公共子序列

- 问题的定义
- 最长公共子序列 (LCS) 结构分析
- 建立求解LCS长度的递归方程
- 自底向上LCS长度的计算
- 构造优化解

问题的定义

- 子序列
  - $X=(A, B, C, B, D, B)$
  - $Z=(B, C, D, B)$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的公共子序列如果Z是X的子序也是Y的子序列。

最长公共子序列（*LCS*）问题

输入：$X=(x_1,x_2,\ldots,x_m)$，$Y=(y_1,y_2,\ldots y_n)$
输出：$Z=X$与Y的最长公共子序列

蛮力法

- 枚举X的每个子序列Z
- 检查Z是否为Y的子序列
- $T(n)=O(n2^m)$

最长公共子序列结构分析

- 第i前缀
  - 设$X=(x_1, x_2, ..., x_m)$是一个序列，X的第i前缀X_i是一个序列，定义为$X_i=(x_1, ..., x_i)$

例. $X=(A, B, D, C, A)$, $X_1=(A)$, $X_2=(A, B)$, $X_3=(A, B, D)$

- 优化子结构

  定理1（优化子结构）设$X=(x_1, ..., x_m)$、$Y=(y_1, ..., y_n)$是两个序列，$Z=(z_1, ..., z_k)$是X与Y的LCS，我们有：

  (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，$LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + <x_m=y_n>$.

  (2) 如果$x_m \neq y_n$，且$z_k \neq x_m$，则Z是X_{m-1}和Y的LCS，即 $LCS_{XY} = LCS_{X_{m-1}Y}$

  (3) 如果$x_m \neq y_n$,且$z_k \neq y_n$,则Z是X与Y_{n-1}的LCS，即 $LCS_{XY} = LCS_{XY_{n-1}}$

证明:

(1). $X=<x_1, \ldots, x_{m-1}, x_m>$, $Y=<y_1, \ldots, y_{n-1}, x_m>$，则

$$LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + <x_m=y_n>.$$

设$z_k \neq x_m$，则可加$x_m=y_n$到Z，得到一个长为$k+1$的X与Y的公共序列，与Z是X和Y的LCS矛盾。于是$z_k=x_m=y_n$。

现在证明Z_{k-1}是X_{m-1}与Y_{n-1}的LCS。显然Z_{k-1}是X_{m-1}与Y_{n-1}的公共序列。我们需要证明Z_{k-1}是LCS。

设不然，则存在X_{m-1}与Y_{n-1}的公共子序列W，W的长大于$k-1$。增加$x_m=y_n$到W，我们得到一个长大于k的X与Y的公共序列，与Z是LCS矛盾。于是，Z_{k-1}是X_{m-1}与Y_{n-1}的LCS.

(2) $X=<x_1, \ldots, x_{m-1}, x_m>$, $Y=<y_1, \ldots, y_{n-1}, y_n>$，

$x_m \neq y_n$，$z_k \neq x_m$，则 $LCS_{XY} = LCS_{X_{m-1}Y}$

由于$z_k \neq x_m$，Z是X_{m-1}与Y的公共子序列。我们来证Z是X_{m-1}与Y的LCS。设X_{m-1}与Y有一个公共子序列W，W的长大于k, 则W也是X与Y 的公共子序列，与Z是LCS矛盾。

(3) 同(2)可证。

X和Y的LCS的优化解结构为

$$LCS_{XY}=LCS_{X_{m-1}Y_{n-1}}+<x_m=y_n> \quad \text{if } x_m=y_n$$

$$LCS_{XY}=LCS_{X_{m-1}Y} \quad \text{if } x_m\neq y_n,\ z_k\neq x_m$$

$$LCS_{XY}=LCS_{XY_{n-1}} \quad \text{if } x_m\neq y_n,\ z_k\neq y_n$$

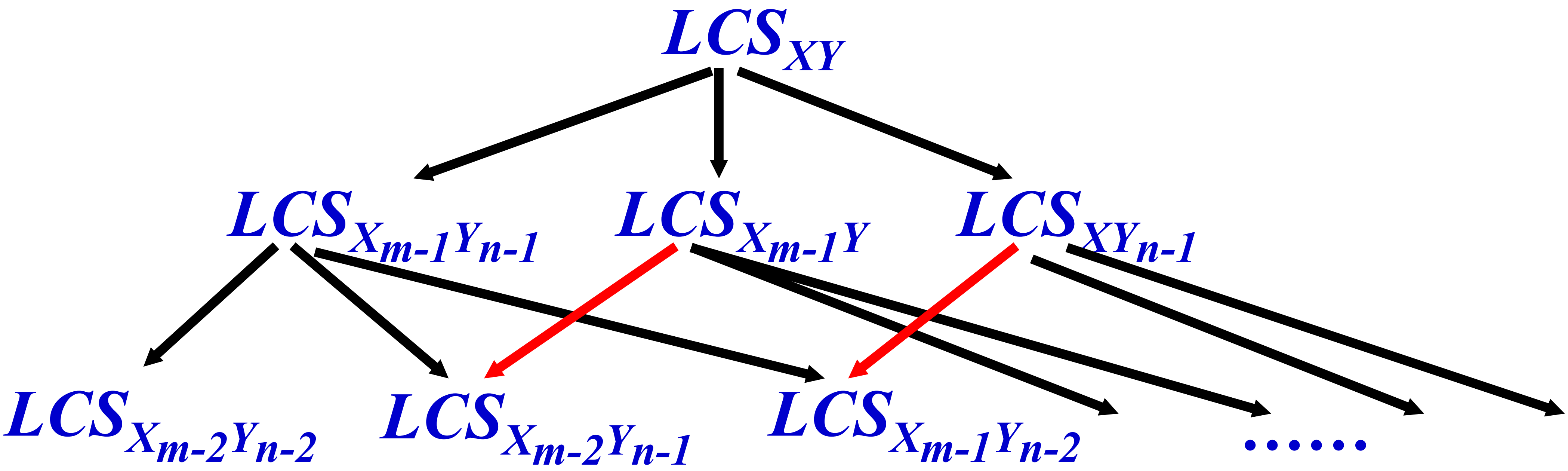
- 算法SimpleLCS(X,Y)

输入： $X=(x_1,x_2,\ldots,x_n)$， $Y=(y_1,y_2,\ldots y_m)$

输出： X,Y的最长公共子序列

1. If m=0 或 n=0 Then 输出空串,算法结束;
2. If $x_n=y_m$ Then
3. 输出SimpleLCS(X_{n-1},Y_{m-1})+<x_n>;
4. Else
5. $Z_1\leftarrow$SimpleLCS(X_{n-1},Y);
6. $Z_2\leftarrow$SimpleLCS(X,Y_{m-1});
7. 输出Z_1,Z_2中较长者;

- 子问题重叠性



LCS问题具有子问题重叠性

建立*LCS*长度的递归方程

- $C[i, j] = X_i$与Y_j的*LCS*的长度
- *LCS*长度的递归方程

$$C[i, j] = 0 \qquad \text{if } i=0 \text{ 或 } j=0$$

$$C[i, j] = C[i-1, j-1] + 1 \qquad \text{if } i, j>0 \text{ and } x_i = y_j$$

$$C[i, j] = Max(C[i, j-1], C[i-1, j]) \qquad \text{if } i, j>0 \text{ and } x_i \neq y_j$$

自底向上计算*LCS*的长度

- 基本思想

	C[i-1, j-1]	*C[i-1,j]*	
	C[i, j-1]	*C[i, j]*	

- 计算过程

C[0,0]	*C[0,1]*	*C[0,2]*	*C[0,3]*	*C[0,4]*
C[1,0]	*C[1,1]*	*C[1,2]*	*C[1,3]*	*C[1,4]*
C[2,0]	*C[2,1]*	*C[2,2]*	*C[2,3]*	*C[2,4]*
C[3,0]	*C[3,1]*	*C[3,2]*	*C[3,3]*	*C[3,4]*

- 计算*LCS*长度的算法
  - 数据结构

    C[0:m,0:n]: *C[i,j]*是X_i与Y_j的*LCS*的长度

    B[1:m,1:n]: *B[i,j]*是指针，指向计算*C[i,j]*时所选择的子问题的优化解所对应的*C*表的表项

```
LCS-length(X, Y)
 m←length(X);  n←length(Y);
 For  i←0  To  m  Do  C[i,0]←0;
 For  j←0  To  n  Do  C[0,j]←0;
 For  i←1  To  m  Do
     For  j←1  To  n  Do
         If  x_i = y_j
         Then C[i,j]←C[i-1,j-1]+1;  B[i,j]←"↖";
         Else If    C[i-1,j]≥C[i,j-1]  Then
                    C[i,j]←C[i-1,j]; B[i,j]←"↑";
              Else C[i,j]←C[i,j-1]; B[i,j]←"←";
Return C and B.
```

	j	0	1	2	3	4	5	6
i		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

构造优化解

- 基本思想
  - 从$B[m, n]$开始按指针搜索
  - 若$B[i, j]$=“↖”，则$x_i=y_j$是*LCS*的一个元素
  - 如此找到的“*LCS*”是X与Y的*LCS*的*Inverse*

Print-LCS(*B*, *X*, *i*, *j*)

IF $i=0$ **or** $j=0$ **THEN Return;**

IF $B[i, j]$=“↖”

THEN Print-LCS(*B*, *X*, *i-1*, *j-1*);

Print x_i;

ELSE If $B[i, j]$=“↑”

THEN Print-LCS(*B*, *X*, *i-1*, *j*);

ELSE Print-LCS(*B*, *X*, *i*, *j-1*).

	j	0	1	2	3	4	5	6
i		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

Print-LCS(B, X, length(X), length(Y))

可打印出*X*与*Y*的*LCS*。

4.3 矩阵链乘法

问题的定义

- 输入：$<A_1, A_2, \ldots, A_n>$, A_i是矩阵
- 输出：计算$A_1 \times A_2 \times \ldots \times A_n$的最小代价方法

矩阵乘法的代价/复杂性：乘法的次数

若A是$p \times q$矩阵，B是$q \times r$矩阵，则$A \times B$的代价是$O(pqr)$

Motivation

- 矩阵链乘法的实现
  - 矩阵乘法满足结合率。
  - 计算一个矩阵链的乘法可有多种方法:

例如，$(A_1 \times A_2 \times A_3 \times A_4)$

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

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

....

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

- 矩阵链乘法的代价与计算顺序的关系
  - 设A_1=10×100矩阵, A_2=100×5矩阵, A_3=5×50矩阵

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

    $T(A_1 \times (A_2 \times A_3)) = 100 \times 5 \times 50 + 10 \times 100 \times 50 = 750000$

结论: 不同计算顺序有不同的代价

- 矩阵链乘法优化问题的解空间
  - 设*p(n)*=计算*n*个矩阵乘积的方法数
  - *p(n)*的递归方程

如此之大的解空间是无法用枚举方法求出最优解的！

$$p(n) = 1 \qquad \text{if } n=1$$

$$p(n) = \sum_{k=1}^{n-1} p(k)\,p(n-k) \qquad \text{if } n>1$$

$$p(n)=C(n-1)=\text{Catalan数}=\frac{1}{n}\binom{2(n-1)}{n-1}=\Omega(4^n/n^{3/2})$$

设计求解矩阵链乘法问题的动态规划算法

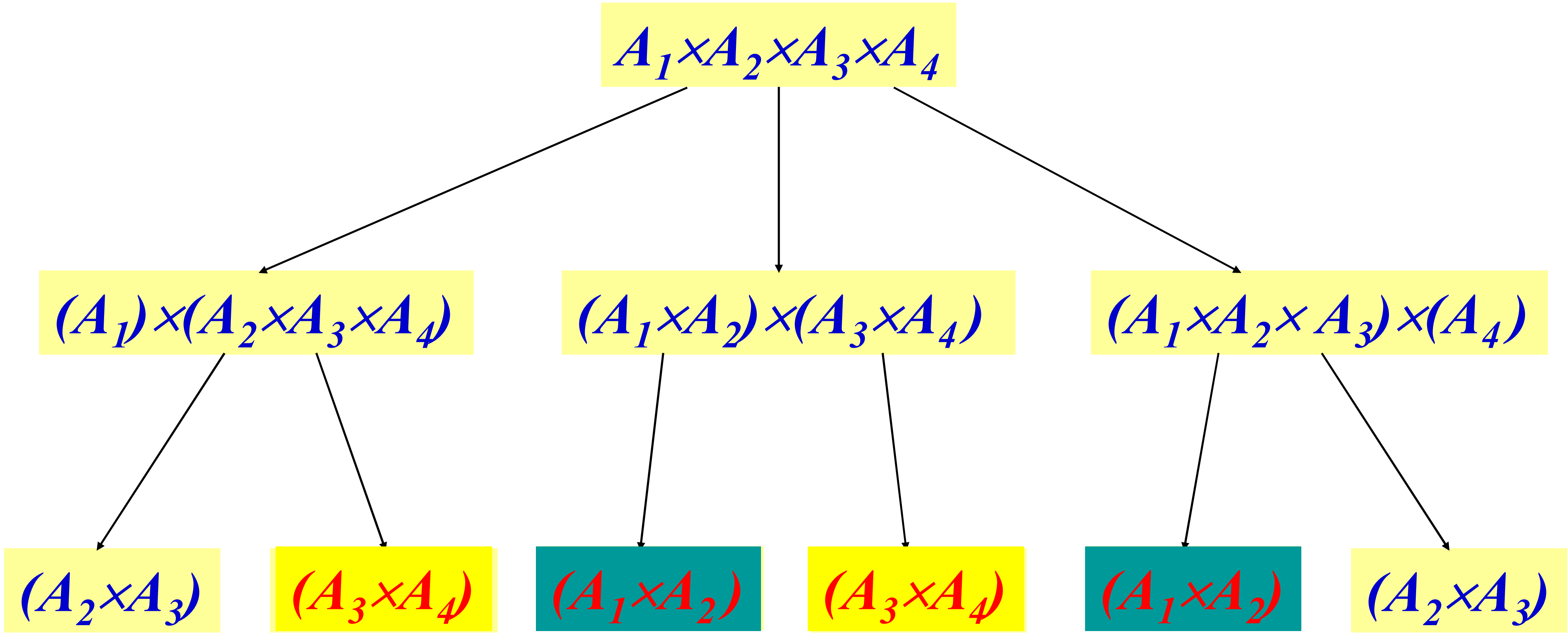
- 分析优化解的结构
- 递归地定义最优解的代价
- 自底向上地计算优化解的代价保存之，并获取构造最优解的信息
- 根据构造最优解的信息构造优化解

分析优化解的结构

- 两个记号
  - $A_{i\text{-}j}=A_i\times A_{i+1}\times\ldots\times A_j$
  - $cost(A_{i\text{-}j})$=计算$A_{i\text{-}j}$的代价
- 优化解的结构
  - 若计算$A_{1\sim n}$的优化顺序在k处断开矩阵链，即$A_{1\sim n}=A_{1\sim k}\times A_{k+1\sim n}$，则在$A_{1\sim n}$的优化顺序中，对应于子问题$A_{1\sim k}$的解必须是$A_{1\text{-}k}$的优化解，对应于子问题$A_{k+1\sim n}$的解必须是$A_{k+1\sim n}$的优化解

具有优化子结构：
问题的优化解包括子问题优化解

- 子问题重叠性



具有子问题重叠性

HIT
CS&E

递归地定义最优解的代价

- 假设
  - $m[i, j]$ = 计算$A_{i\sim j}$的最小乘法数
  - $m[1, n]$ = 计算$A_{1\sim n}$的最小乘法数
  - $A_1 \ldots A_k A_{k+1} \ldots. A_n$ 是优化解(*k*实际上是不可预知)
- 代价方程

$m[i, i]$ = 计算$A_{i\sim i}$的最小乘法数= 0

$$m[i, j] = m[i, k] + m[k+1, j] + p_{i-1}p_kp_j$$

其中, $p_{i-1}p_kp_j$是计算$A_{i\sim k}\times A_{k+1\sim j}$所需乘法数,

$A_{i\sim k}$和$A_{k+1\sim j}$分别是$p_{i-1}\times p_k$和$p_k\times p_j$矩阵.

考虑到所有的k，优化解的代价方程为

$$m[i,j] = 0 \qquad \text{if } i=j$$

$$m[i,j] = \min_{i \le k < j}\{ m[i,k] + m[k+1,j] + p_{i-1}p_kp_j \} \qquad \text{if } i<j$$

自底向上计算优化解的代价

$$m[i, j] = \min_{i \le k < j}\{ m[i, k] + m[k+1, j] + p_0 p_k p_5 \}$$

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

$$m[2,4] = \min\left\{\begin{array}{l} m[2,2]+m[3,4] \\ m[2,3]+m[4,4] \end{array}\right.$$

$$m[i, j] = \min_{i \le k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

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

```
Matrix-Chain-Order(p)
n=length(p)-1;
FOR i=1 TO n DO
    m[i, i]=0;
FOR l=2 TO n DO /* 计算l对角线 */
    FOR i =1 TO n-l+1 DO
        j=i+l-1;
        m[i, j]=∞;
        FOR k←i To j-1 DO /* 计算m[i,j] */
            q=m[i, k]+m[k+1, j]+p_{i-1}p_k p_j
            IF q<m[i, j] THEN m[i, j]=q;
Return m.
```

HIT
CS&E

获取构造最优解的信息

$S[i,j]$记录$A_iA_{i+1}\ldots A_j$的最优划分处在A_k与A_{k+1}之间

Matrix-Chain-Order(p)
n=length(p)-1；
FOR $i=1$ TO n DO
$m[i, i]=0$;
FOR $l=2$ TO n DO
FOR $i=1$ TO $n-l+1$ DO
$j=i+l-1$;
$m[i, j]=\infty$;
FOR $k\leftarrow i$ To j-1 DO
$q = m[i, k]+m[k+1, j]+p_{i-1}p_kp_j$
IF $q<m[i, j]$ THEN $m[i,j]=q$，$s[i,j]=k$;
Return m and s.

构造最优解

$S[i, j]$记录$A_i \ldots A_j$的最优划分处；
$S[i, S[i,j]]$记录$A_i \ldots A_{s[i,j]}$的最优划分处；
$S[S[i,j]+1, j]$记录$A_{s[i,j]+1} \ldots A_j$的最优划分处.

```
Print-Optimal-Parens(s, i, j)
 IF j=i
 THEN  Print "A"i;
 ELSE  Print "("
       Print-Optimal-Parens(s, i, s[i, j])
       Print-Optimal-Parens(s, s[i, j]+1, j)
       Print  ")"
```

调用**Print-Optimal-Parens(s, 1, n)**
即可输出$A_{1\sim n}$的优化计算顺序

算法复杂性

- 时间复杂性
  - 计算代价的时间
    - (l, i, k)三层循环, 每层至多n-1步
    - $O(n^3)$
  - 构造最优解的时间: $O(n)$
  - 总时间复杂性为: $O(n^3)$
- 空间复杂性
  - 使用数组m和S
  - 需要空间$O(n^2)$

4.4 0/1 背包问题

问题的定义

给定n种物品和一个背包，物品i的重量是w_i，价值v_i，背包承重为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？

对于每种物品只能选择完全装入或不装入，一个物品至多装入一次。

- 输入： $C>0,\ w_i>0,\ v_i>0,\ 1\leq i\leq n$
- 输出： $(x_1, x_2, \ldots, x_n),\ x_i\in\{0, 1\}$, 满足

$$\sum_{1\leq i\leq n} w_i x_i \leq C,\quad \sum_{1\leq i\leq n} v_i x_i \text{ 最大}$$

等价的整数规划问题

$$\max \sum_{1\leq i\leq n} v_i x_i$$

$$\sum_{1\leq i\leq n} w_i x_i \leq C$$

$$x_i\in\{0, 1\},\ 1\leq i\leq n$$

优化解结构的分析

定理 (优化子结构) 如果$(y_1, y_2, \ldots, y_n)$是0-1背包问题的优化解，则$(y_2, \ldots, y_n)$是如下子问题的优化解：

$$\max \sum_{2 \le i \le n} v_i x_i$$

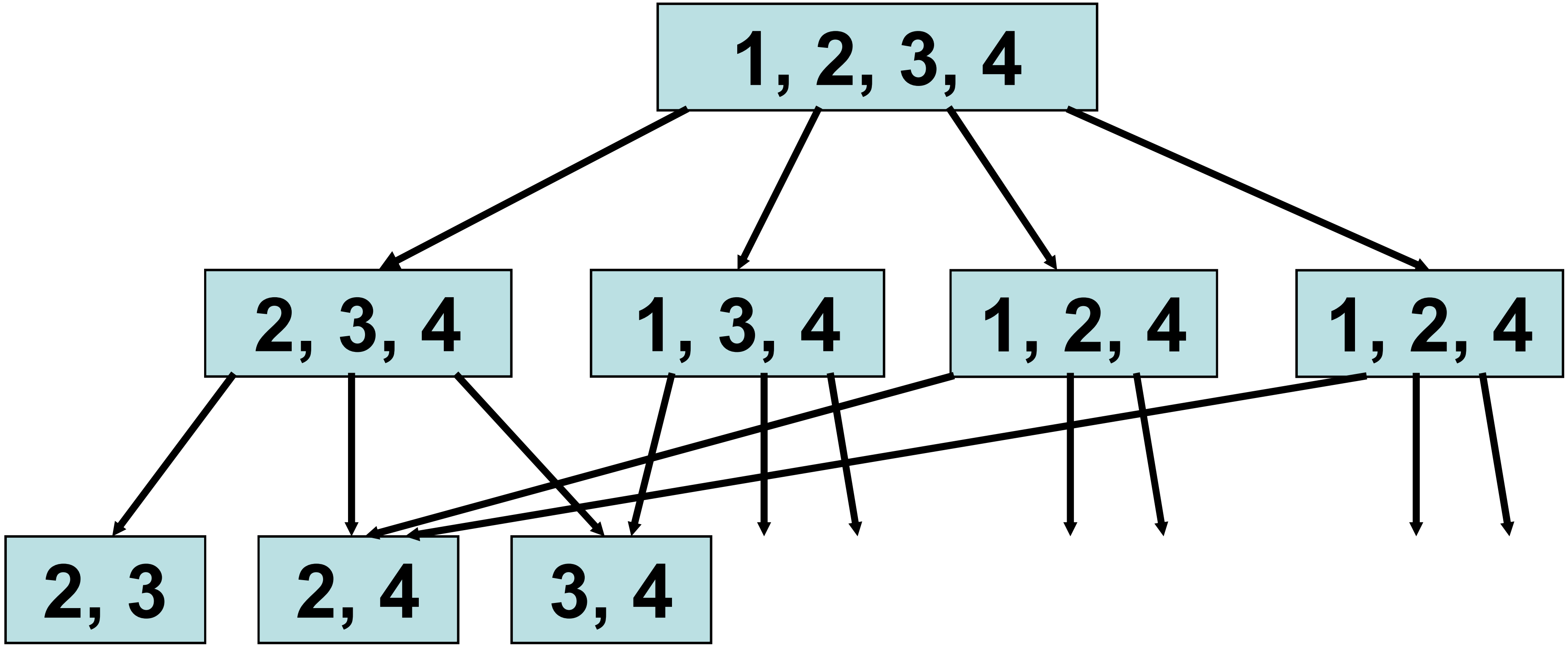
$$\sum_{2 \le i \le n} w_i x_i \le C - w_1 y_1$$

$$x_i \in \{0, 1\}, \quad 2 \le i \le n$$

证明： 如果$(y_2, \ldots, y_n)$不是子问题优化解, 则存在$(z_2, \ldots, z_n)$是子问题更优的解。于是, $(y_1, z_2, \ldots, z_n)$是原问题比$(y_1, y_2, \ldots, y_n)$更优解，矛盾。

HIT
CS&E

子问题重叠性



建立优化解代价的递归方程

- $m(i, j)$:
  背包容量为j, 可选物品为x_i, x_{i+1}, …, x_n时，问题的最优解的代价是$m(i, j)$.
- 形式地

  问题 $$\max \sum_{i\le k\le n} v_k x_k$$

  $$\sum_{i\le k\le n} w_k x_k \le j$$

  $$x_k \in \{0, 1\},\ i\le k \le n$$

  的最优解代价为$m(i, j)$.

- 递归方程

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

$$m(n, j) = 0 \qquad 0 \le j < w_n$$
$$m(n, j) = v_n \qquad j \ge w_n$$

自底向上计算优化解的代价

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

	$m(i,j)$
$m(i+1,j-w_i)$	$m(i+1,j)$

令w_i=整数, $n=4$

$m(1,C)$

$m(2,C-w_1)$ … $m(2,C)$

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

… $m(4,C-w_1-w_2-w_3)$ … $m(4,C-w_1-w_2)$ … $m(4,C-w_1)$ … $m(4,C)$

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

$$\begin{array}{r|l} & m(i, j) \\ \hline m(i+1, j-w_i) & m(i+1, j) \end{array}$$

令w_i=整数, $n=4$

						$m(1,C)$
$m(2, 0)$	…	$m(2,w_2-1)$	$m(2, w_2)$	…	$m(2,C-1)$	$m(2,C)$
$m(3, 0)$	…	$m(3,w_3-1)$	$m(3, w_3)$	…	$m(3,C-1)$	$m(3,C)$
$m(4, 0)$	…	$m(4,w_4-1)$	$m(4, w_4)$	…	$m(4,C-1)$	$m(4,C)$

•算法

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

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

构造优化解

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.

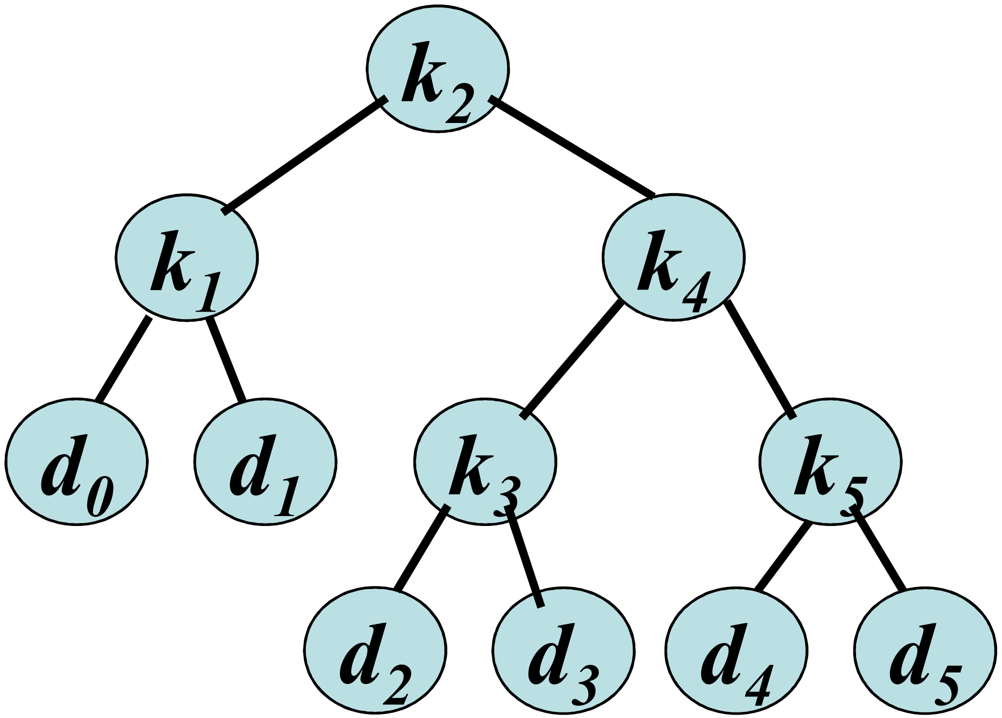
- 时间复杂性
  - 计算代价的时间
    - $O(Cn)$
  - 构造最优解的时间: $O(n)$
  - 总时间复杂性为: $O(Cn)$
- 空间复杂性
  - 使用数组 m
  - 需要空间 $O(Cn)$

4.5 最优二叉搜索树

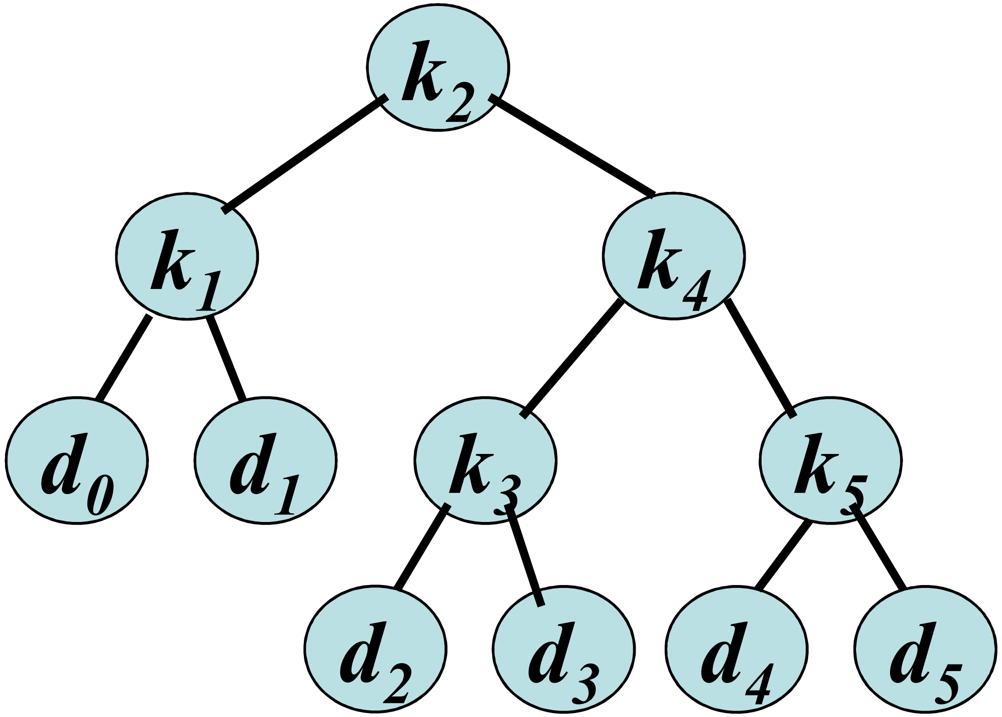
问题的定义

- 二叉搜索树T
  - 结点
    - $K=\{k_1, k_2, \ldots, k_n\}$
    - $D=\{d_0, d_1, \ldots, d_n\}$
    - d_i对应区间(k_i, k_{i+1})
      d_0对应区间$(-\infty, k_1)$
      d_n对应区间$(k_n, +\infty)$
  - 附加信息
    - 搜索k_i的概率为p_i
    - 搜索d_i的概率为q_i

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



- 搜索树的期望代价



$$E(T)=\sum_{i=1}^{n}(DEP_T(k_i)+1)p_i+\sum_{j=0}^{n}(DEP_T(d_i)+1)q_i$$

- 问题的定义

输入：$K=\{k_1, k_2, \ldots, k_n\}$, $k_1< k_2< \ldots < k_n$,

$P=\{p_1, p_2, \ldots, p_n\}$, p_i为搜索k_i的概率

$Q=\{q_0, q_1, \ldots, q_n\}$, q_i为搜索值d_i的概率

输出：构造K的二叉搜索树T, 使得

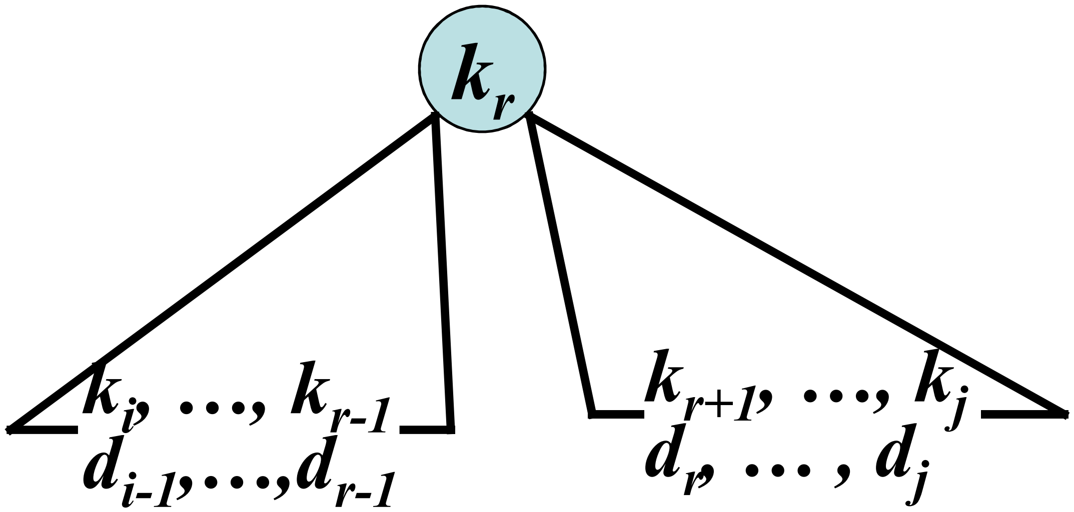
$$E(T)=\sum_{i=1}^{n}(DEP_T(k_i)+1)p_i+\sum_{j=0}^{n}(DEP_T(d_i)+1)q_i$$

最小

优化二叉搜索树结构的分析

- 划分子问题

$K=\{k_i, k_{i+1}, \ldots, k_j\}$的优化解的根必为$K$中某个$k_r$



如果$r=i$, 左子树$\{k_i, \ldots, k_{i-1}\}$仅包含d_{i-1}
如果$r=j$, 右子树$\{k_{r+1}, \ldots, k_j\}$仅包含d_j

HIT
CS&E

- **优化子结构**

定理. 如果优化二叉搜索树T具有包含关键字集合$\{k_i, k_{i+1}, \ldots, k_j\}$的子树$T'$, 则$T'$是关于关键字集合$\{k_i, k_{i+1}, \ldots, k_j\}$的子问题的优化解.

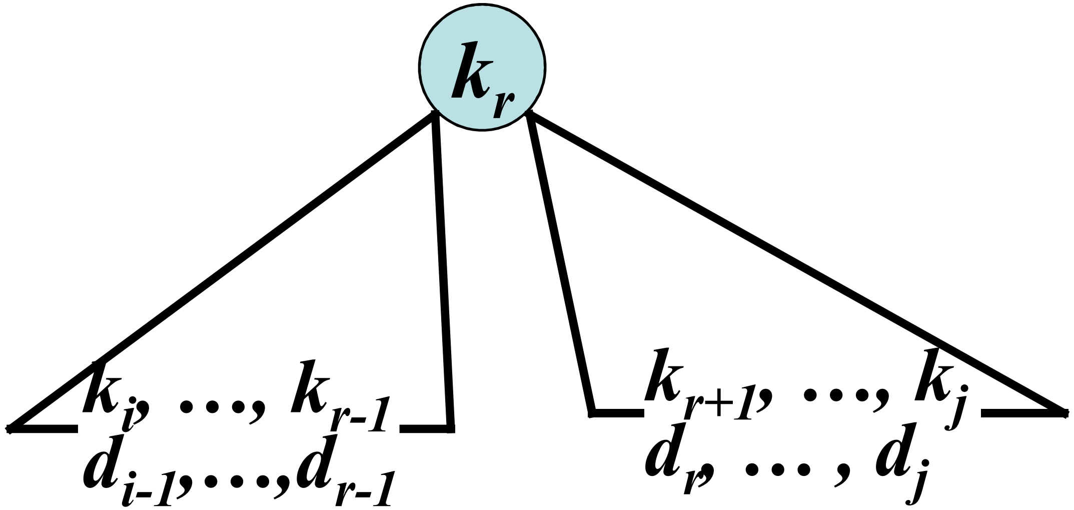
证明: 若不然，必有关键字集$\{k_i, k_{i+1}, \ldots, k_j\}$子树$T''$, T''的期望搜索代价低于T'.

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.

- 用优化子结构从子问题优化解构造优化解

$K=\{k_i, k_{i+1}, \ldots, k_j\}$的优化解的根必为$K$中某个$k_r$

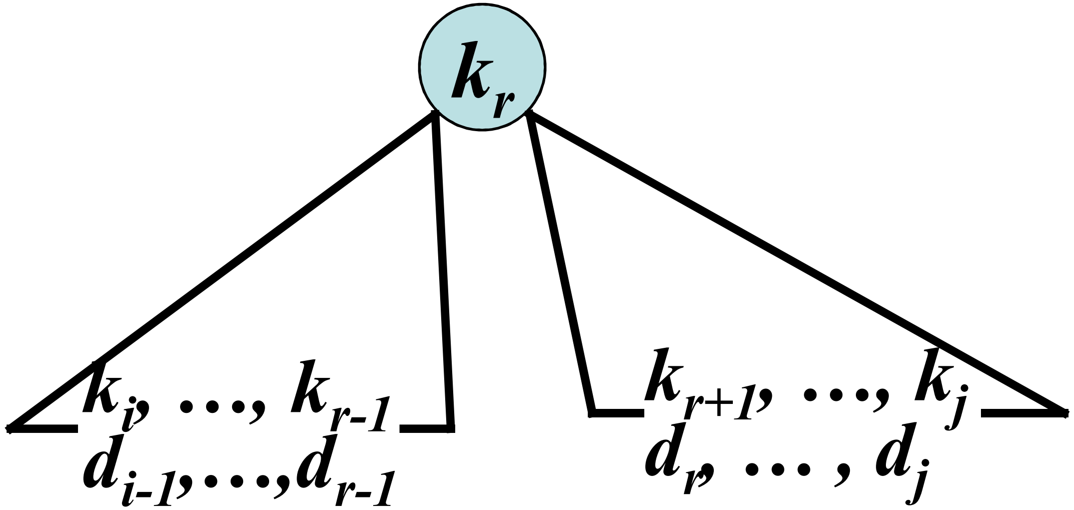


只要对于每个$k_r \in K$, 确定$\{k_i, \ldots, k_{r-1}\}$和$\{k_{r+1}, \ldots, k_j\}$的优化解，我们就可以求出K的优化解.

如果$r=i$, 左子树$\{k_i, \ldots, k_{i-1}\}$仅包含d_{i-1}
如果$r=j$, 右子树$\{k_{r+1}, \ldots, k_j\}$仅包含d_j

建立优化解的搜索代价递归方程

- 令$E(i, j)$为$\{k_i, \ldots, k_j\}$的优化解T_{ij}的期望搜索代价
  - 当$j=i-1$时，T_{ij}中只有叶结点d_{i-1}, $E(i, i-1)=q_{i-1}$
  - 当$j\geq i$时，选择一个$k_r\in\{k_i, \ldots, k_j\}$:



当把左右优化子树放进T_{ij}时, 每个结点的深度增加1

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

- 计算$W(i, r-1)$和$W(r+1, j)$

由 $$E(LT+1)=\sum_{l=i}^{r-1}(DEP_{左}(k_l)+2)p_l+\sum_{l=i-1}^{r-1}(DEP_{左}(d_l)+2)q_l$$

$$E(LT)=\sum_{l=i}^{r-1}(DEP_{左}(k_l)+1)p_l+\sum_{l=i-1}^{r-1}(DEP_{左}(d_l)+1)q_l$$

知 $$W(i,r-1)=E(LT+1)-E(LT)=\sum_{l=i}^{r-1}p_l+\sum_{l=i-1}^{r-1}q_l$$

同理，$W(r+1,j)=\sum_{l=r+1}^{j}p_l+\sum_{l=r}^{j}q_l$

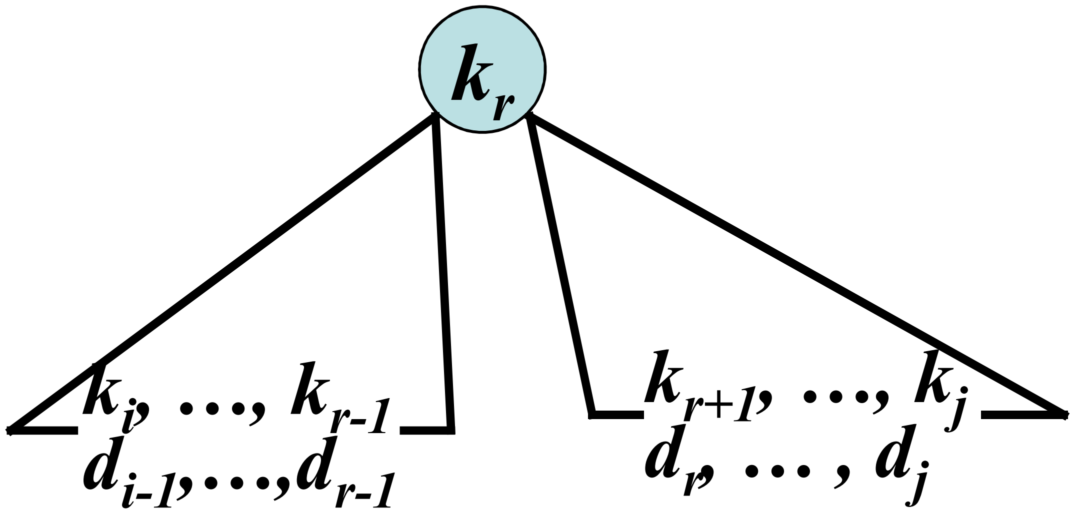
令 $W(i, j)=W(i, r-1)+W(r+1, j)+p_r=\sum_{l=i}^{j}p_l+\sum_{l=i-1}^{j}q_l$

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

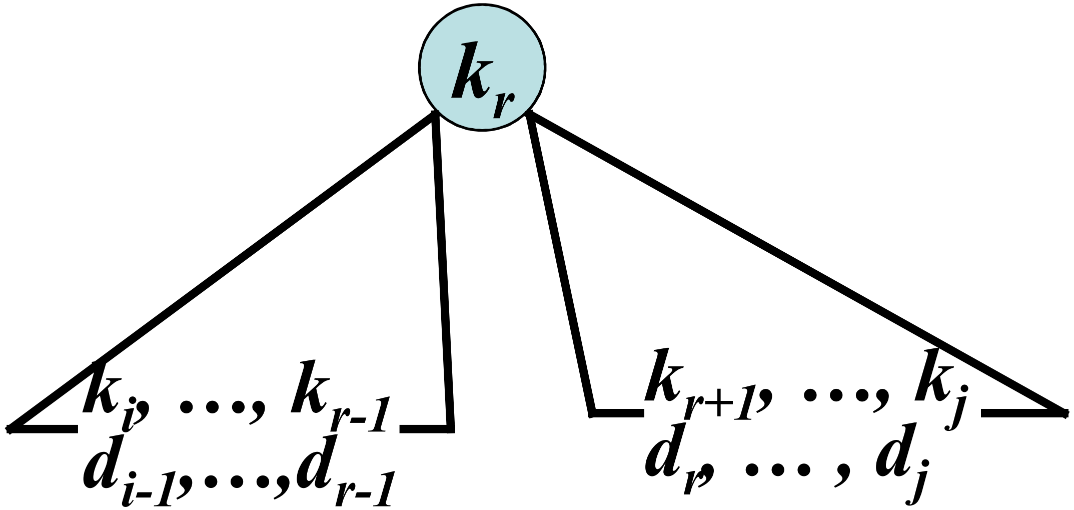
$W(i, i-1)=q_{i-1}$

$$W(i, j) = W(i, r-1) + W(r+1, j) + p_r = W(i, j-1) + p_j + q_j$$

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



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



In Summary

$$E(i, j) = q_{i-1} \quad \text{If } j = i-1$$

$$E(i, j) = \min_{i \le r \le j} \{E(i, r-1) + E(r+1, j) + W(i, j)\} \quad \text{If } j \ge i$$

$$W(i, i-1) = q_{i-1},$$

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

自下而上计算优化解的搜索代价

$$E(i, j)=q_{i-1} \quad If\ \ j=i-1$$

$$E(i, j)=\min_{i\leq r\leq j}\{E(i,r-1)+E(r+1,j)+W(i, j)\} \quad If\ j\geq i$$

$q_0=$	$E(1,0)$		$E(1,1)$		$E(1,2)$		$E(1,3)$		$E(1,4)$
		$q_1=$	$E(2,1)$		$E(2,2)$		$E(2,3)$		$E(2,4)$
				$q_2=$	$E(3,2)$		$E(3,3)$		$E(3,4)$
						$q_3=$	$E(4,3)$		$E(4,4)$
								$q_4=$	$E(5,4)$

- $W(i, i\text{-}1) = q_{i-1},\quad W(i, j) = W(i, j\text{-}1) + p_j + q_j$

$q_0 =$	$W(1,0)$	$W(1,1)$	$W(1,2)$	$W(1,3)$	$W(1,4)$
	$q_1 =$	$W(2,1)$	$W(2,2)$	$W(2,3)$	$W(2,4)$
		$q_2 =$	$W(3,2)$	$W(3,3)$	$W(3,4)$
			$q_3 =$	$W(4,3)$	$W(4,4)$
				$q_4 =$	$W(5,4)$

- 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 $\{k_i, \ldots, k_j\}$优化解的根

HIT
CS&E

Optimal-BST(*p*, *q*, *n*)

```
For i=1 To n+1 Do
    E(i, i-1) = q_{i-1};
    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
```

算法的复杂性

- 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
  - $O(n^2)$

4.6 凸多边形的三角剖分

- 弦

  多边形P上的任意两个不相邻结点v_i、v_{i+1}所对应的线段v_iv_{i+1}称为弦

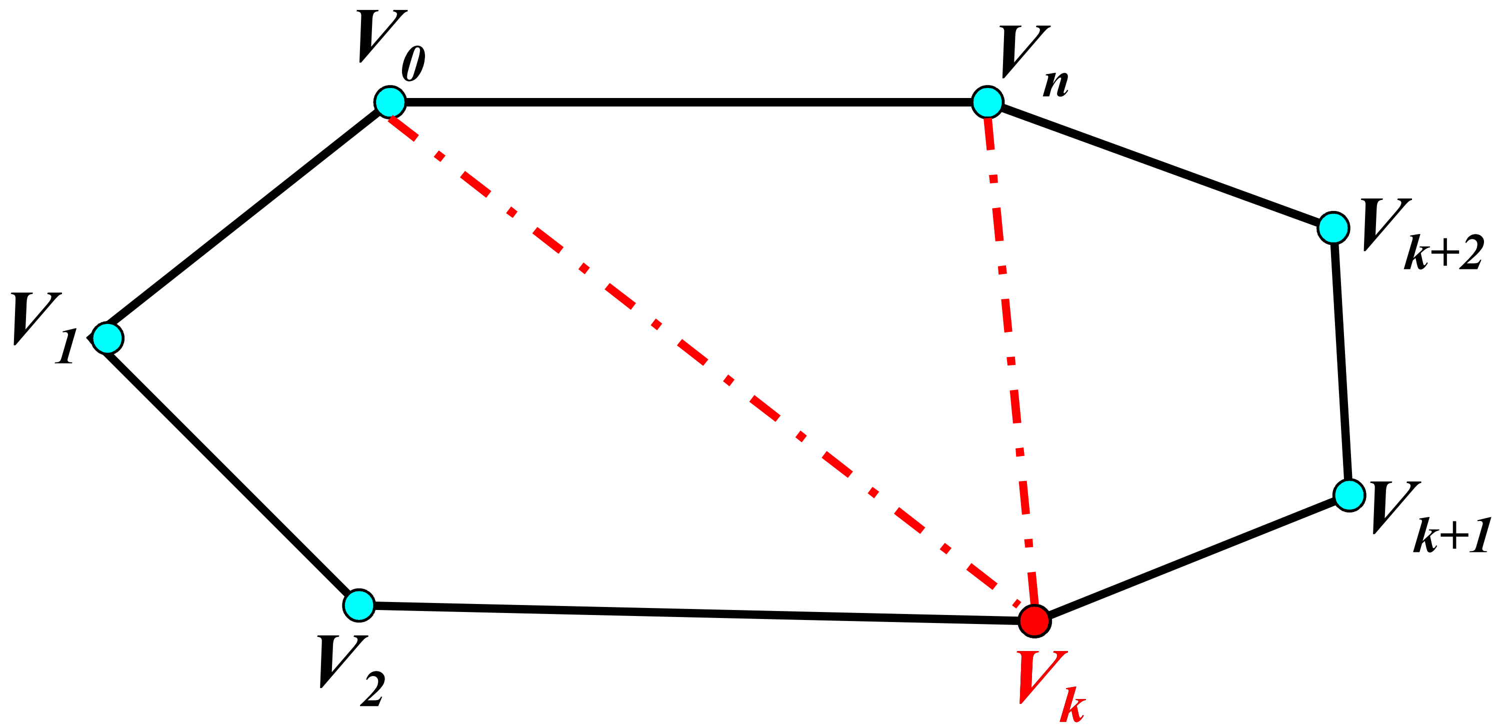
- 三角剖分

  一个多边形P的三角剖分是将P划分为不相交三角形的弦的集合

- 优化三角剖分问题
  - 输入：多边形P和代价函数W
  - 输出：求P的三角分T，使得代价$\sum_{s\in S_T}W(s)$最小，
    其中S_T是T所对应的三角形集合

优化解结构的分析

- 设
  - $P=(v_0, v_1, \ldots v_n)$是$n+1$个顶点的多边形
  - T_P是P的优化三角剖分，包含三角形$v_0v_kv_n$



  - $T_P=T(v_0, \ldots, v_k) \cup T(v_k, \ldots, v_n) \cup \{v_0v_k, v_kv_n\}$

优化三角剖分的代价函数

- 设 $t[i,j] = <v_{i-1}, v_i, \ldots\ldots, v_j>$的优化三角剖分代价

$$t[i, i] = t[j, j] = 0$$

$$t[i, j] = \min_{i \le k < j}\{t[i, k] + t[k+1, j] + w(\Delta_{v_{i-1}v_k v_j})\}$$

注意：

$t[i, k] = <v_{i-1}, v_i, \ldots\ldots, v_k>$的优化三角剖分代价

$t[k+1, j] = <v_k, v_i, \ldots\ldots, v_j>$的优化三角剖分代价

优化三角剖分动态规划算法

与矩阵链乘法问题一致，把算法

Matrix-chain-Order

Print-Optimal-Parens

略加修改即可计算*t[i,j]*并构造优化三角剖分解