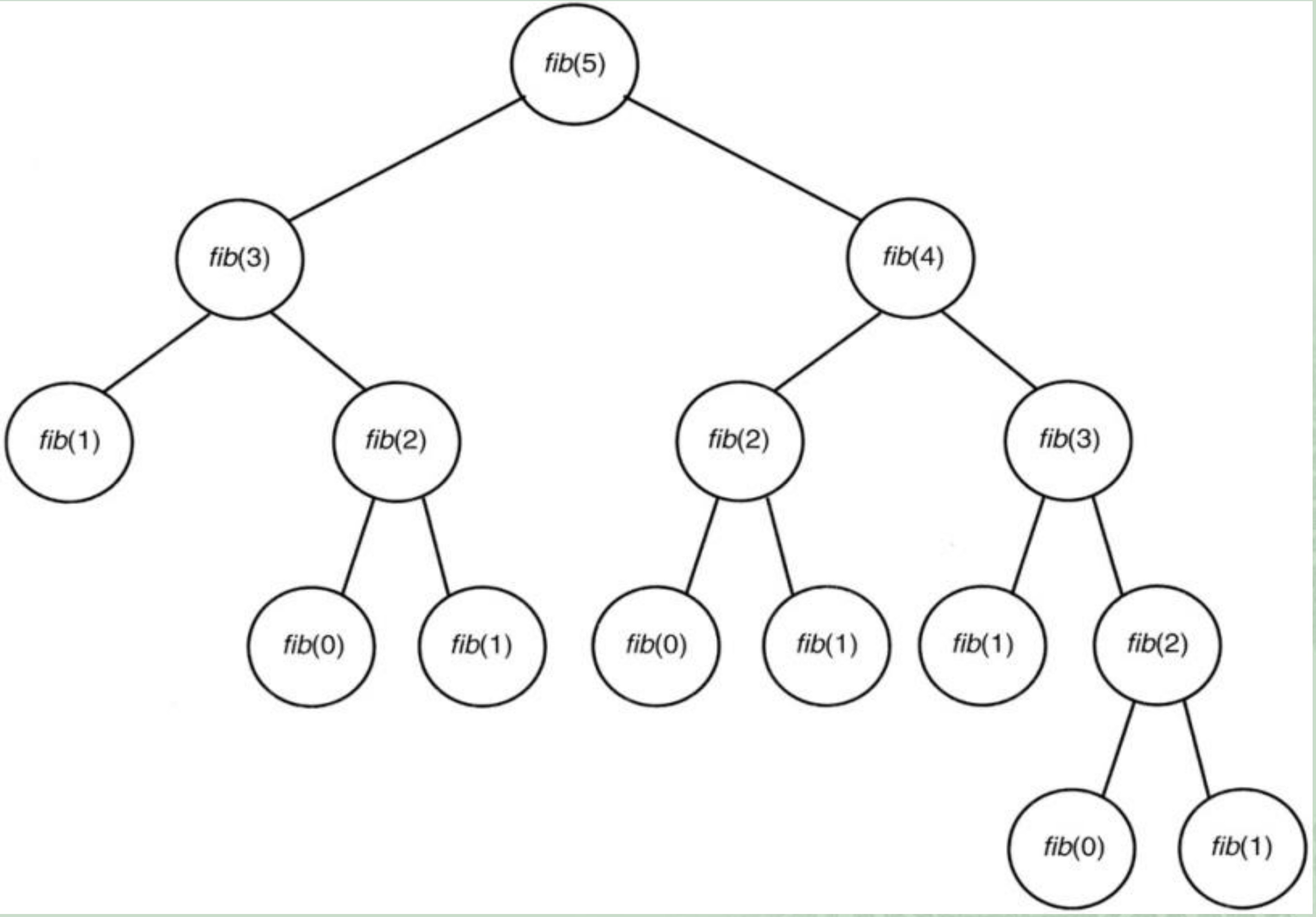
第四章 动态规划

✓目录

教学目标

- 理解动态规划的思想
- 掌握动态规划、分治法及贪心法的异同
- 掌握动态规划的基本要素
- 掌握动态规划的设计步骤
- 通过实例学习，掌握动态规划设计的策略

问题提出



fib(5)
fib(3)
fib(4)
fib(1)
fib(2)
fib(2)
fib(3)
fib(0)
fib(1)
fib(0)
fib(1)
fib(1)
fib(2)
fib(0)
fib(1)

学习动态规划的意义

- 动态规划问世以来，在经济管理、生产调度、工程技术和最优控制等方面得到了广泛的应用，例如最短路线、库存管理、资源分配、设备更新、排序、装载等问题，用动态规划方法比用其它方法求解更为方便。
- 虽然动态规划主要用于求解以时间划分阶段的动态过程的优化问题，但是一些与时间无关的静态规划（如线性规划、非线性规划），只要人为地引进时间因素，把它视为多阶段决策过程，也可以用动态规划方法方便地求解，因此研究该算法具有很强的实际意义。

动态规划的基本思想

- 基本思想
  - 问题具有类似子结构，各个子问题往往不是相互独立的。
  - 在求解过程中，将已解决的子问题的解进行保存，在需要时可以轻松找出。
  - 通常采用表的形式存放已解决的子问题的解，但是填表的过程非常重要。
  - 备忘录法可以不考虑填表顺序

动态规划的解题步骤

- **1.** 分析最优解的性质，刻画最优解的结构特征—考察是否适合采用动态规划法。
- **2.** 递归定义最优值（即建立递归式或动态规划方程）。
- **3.** 以*自底向上*的方式计算出最优值，并记录相关信息。
  - 一般说，分治法的思想是*自顶向下*
- **4.** 根据计算最优值时得到的信息，构造出最优解。

动态规划的基本要素

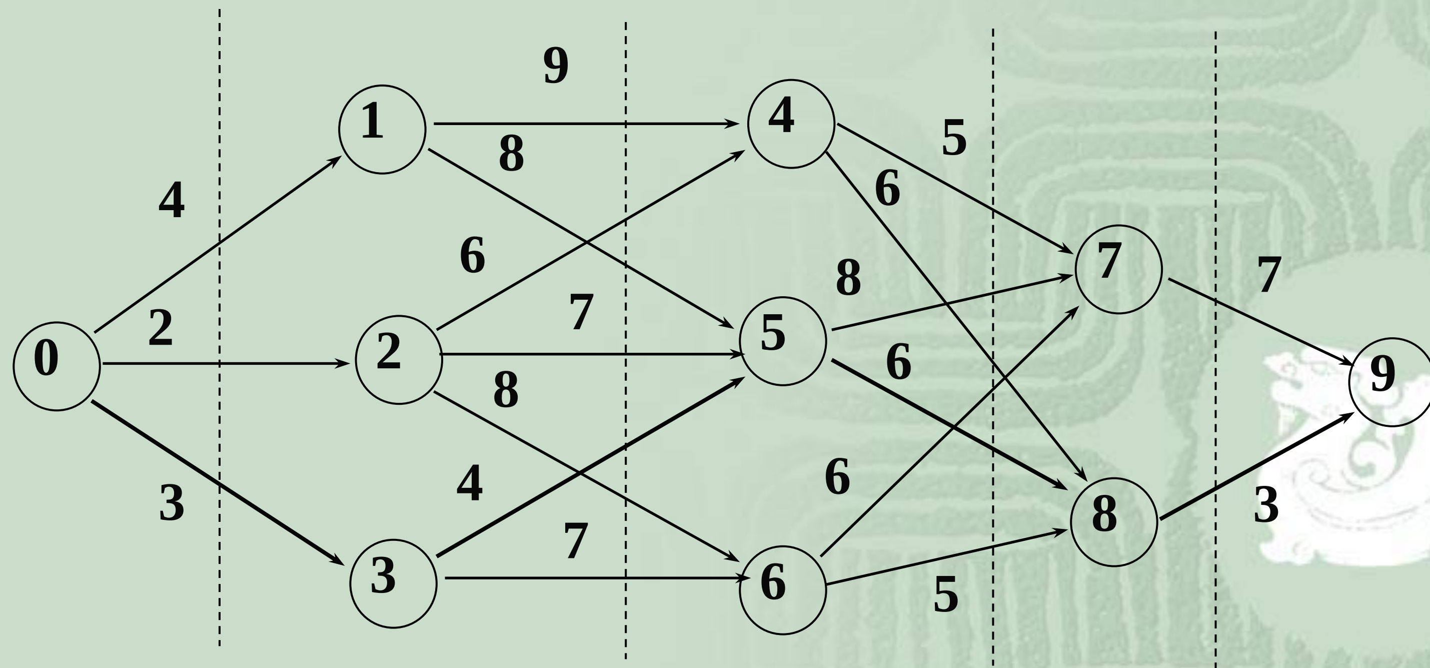
- 最优子结构性质
- 子问题重叠性质
  - 递归求解时，每次产生的子问题并不总是新问题，有些子问题出现多次，这种性质称为子问题的重叠性质。
  - 对于重复出现的子问题，只需在第一次遇到时就加以解决，并储存在表中，便于以后遇到时直接引用，可大大提高解题的效率。
- 自底向上的求解方式

动规解决斐波那契数列

- 仅从重叠性质考虑

多段图的最短路径问题

- 设图G=(V, E)是一个带权有向连通图，如果把顶点集合V划分成k个互不相交的子集Vi（2≤k≤n, 1≤i≤k），使得E中的任何一条边(u, v)，必有u∈Vi，v∈Vi+m（1≤i＜k, 1＜i+m≤k），则称图G为多段图，称s∈V1为源点，t∈Vk为终点。多段图的最短路径问题是求从源点到终点的最小代价路径。



多段图的最短路径问题

- 最优子结构性质
- 子问题重叠性质
- 自底向上的求解方式

矩阵连乘

- 问题描述
  - 给定n个矩阵{A_1,A_2,A_3,…,A_n}，其中A_i与A_{i+1}(i=1,2,3, …,n-1)是可乘的。用加括号的方法表示矩阵连乘的次序，不同加括号的方法所对应的计算次序是不同的。
  - 以【例4-2】为例讲述
  - 最优子结构性质分析

矩阵连乘

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 & 9 & 1 \\ 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 29 & 35 & 41 & 38 \\ 74 & 89 & 104 & 83 \end{bmatrix}$$

$$\underbrace{1 \times 7 + 2 \times 2 + 3 \times 6}_{\text{3 multiplications}}$$

$$2 \times 4 \times 3 = 24$$

矩阵连乘

M[i x j]×M[j x k] $i \times j \times k$ elementary multiplications

A	$\times$	B	$\times$	C	$\times$	D
20×2		2×30		30×12		12×8

$A((B(CD))$	$30 \times 12 \times 8 + 2 \times 30 \times 8 + 20 \times 2 \times 8 = 3,680$
$(AB)(CD)$	$20 \times 2 \times 30 + 30 \times 12 \times 8 + 20 \times 30 \times 8 = 8,880$
$A((BC)D)$	$2 \times 30 \times 12 + 2 \times 12 \times 8 + 20 \times 2 \times 8 = 1,232$
$((AB)C)D$	$20 \times 2 \times 30 + 20 \times 30 \times 12 + 20 \times 12 \times 8 = 10,320$
$(A(BC))D$	$2 \times 30 \times 12 + 20 \times 2 \times 12 + 20 \times 12 \times 8 = 3,120$

矩阵连乘

- 穷举法的时间复杂度是指数级的

$$A_1(\underbrace{A_2A_3\cdots A_n}_{t_{n-1}})$$

$$\underbrace{(A_1A_2A_3\cdots)}_{t_{n-1}}A_n$$

$$t_n \geq t_{n-1} + t_{n-1} = 2t_{n-1}$$

$$t_n \geq 2^{n-2}$$

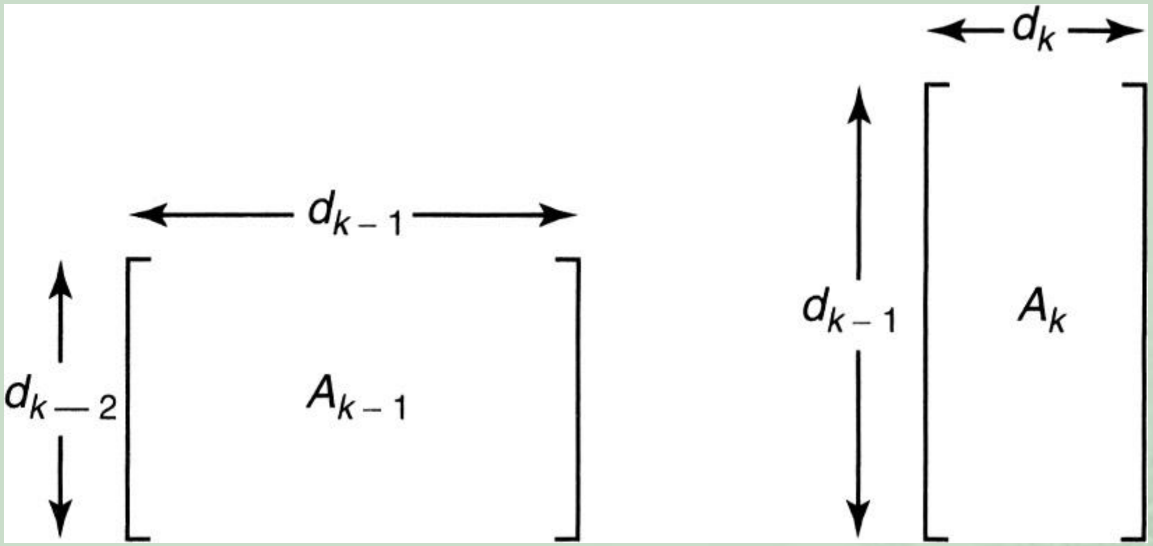
矩阵连乘

- 最优子结构性质

$$A_1((((A_2A_3)A_4)A_5)A_6)$$

$$(A_2A_3)A_4$$

矩阵连乘



A_1	$\times$	A_2	$\times$	A_3	$\times$	A_4	$\times$	A_5	$\times$	A_6
5×2		2×3		3×4		4×6		6×7		7×8
$d_0 \quad d_1$		$d_1 \quad d_2$		$d_2 \quad d_3$		$d_3 \quad d_4$		$d_4 \quad d_5$		$d_5 \quad d_6$

维度数组：　$d\,[\,0 \ldots n\,]$

A_k 的维度是：　$d_{k-1} \times d_k$

矩阵连乘

用二维数组来记录子问题的解。

$M[i][j] = A_i$ 到 A_j 的最小乘数 （$i<j$）

$M[i][i] = 0$

矩阵连乘

A_1	$\times$	A_2	$\times$	A_3	$\times$	A_4	$\times$	A_5	$\times$	A_6
5×2		2×3		3×4		4×6		6×7		7×8
$d_0 \quad d_1$		$d_1 \quad d_2$		$d_2 \quad d_3$		$d_3 \quad d_4$		$d_4 \quad d_5$		$d_5 \quad d_6$

$$
\begin{aligned}
(A_4A_5)A_6 \text{ Number of multiplications } &= d_3\times d_4\times d_5 + d_3\times d_5\times d_6 \\
&= 4\times 6\times 7 + 4\times 7\times 8 = 392 \\
A_4(A_5A_6) \text{ Number of multiplications } &= d_4\times d_5\times d_6 + d_3\times d_4\times d_6 \\
&= 6\times 7\times 8 + 4\times 6\times 8 = 528
\end{aligned}
$$

$$M[4][6] = minimum(392, 528) = 392$$

矩阵连乘

6个矩阵的最优连乘顺序一定是下面的某一种:

1. $A_1\ (A_2A_3A_4A_5A_6)$
2. $(A_1A_2)\ (A_3A_4A_5A_6)$
3. $(A_1A_2A_3)\ (A_4A_5A_6)$
4. $(A_1A_2A_3A_4)\ (A_5A_6)$
5. $(A_1A_2A_3A_4A_5)\ (A_6)$

矩阵连乘

1. $A_1 \ (A_2A_3A_4A_5A_6)$
2. $(A_1A_2)\ (A_3A_4A_5A_6)$
3. $(A_1A_2A_3)\ (A_4A_5A_6)$
4. $(A_1A_2A_3A_4)\ (A_5A_6)$
5. $(A_1A_2A_3A_4A_5)\ (A_6)$

$$M[1][k] + M[k+1][6] + d_0 d_k d_6$$

$$M[1][6] = \underset{1 \le k \le 5}{minimum}(M[1][k] + M[k+1][6] + d_0 d_k d_6)$$

矩阵连乘

For $1 \leq i \leq j \leq n$

$$M[i][j] = \min_{i \leq k \leq j-1} \left(M[i][k] + M[k+1][j] + d_{i-1} d_k d_j \right) \quad \text{if } i < j$$

$$M[i][i] = 0$$

矩阵连乘

M[1][6] → M[1][1], M[2][6] ; M[1][2] , M[3][6] ;

M[1][3] , M[4][6] ; M[1][4] , M[5][6] ;

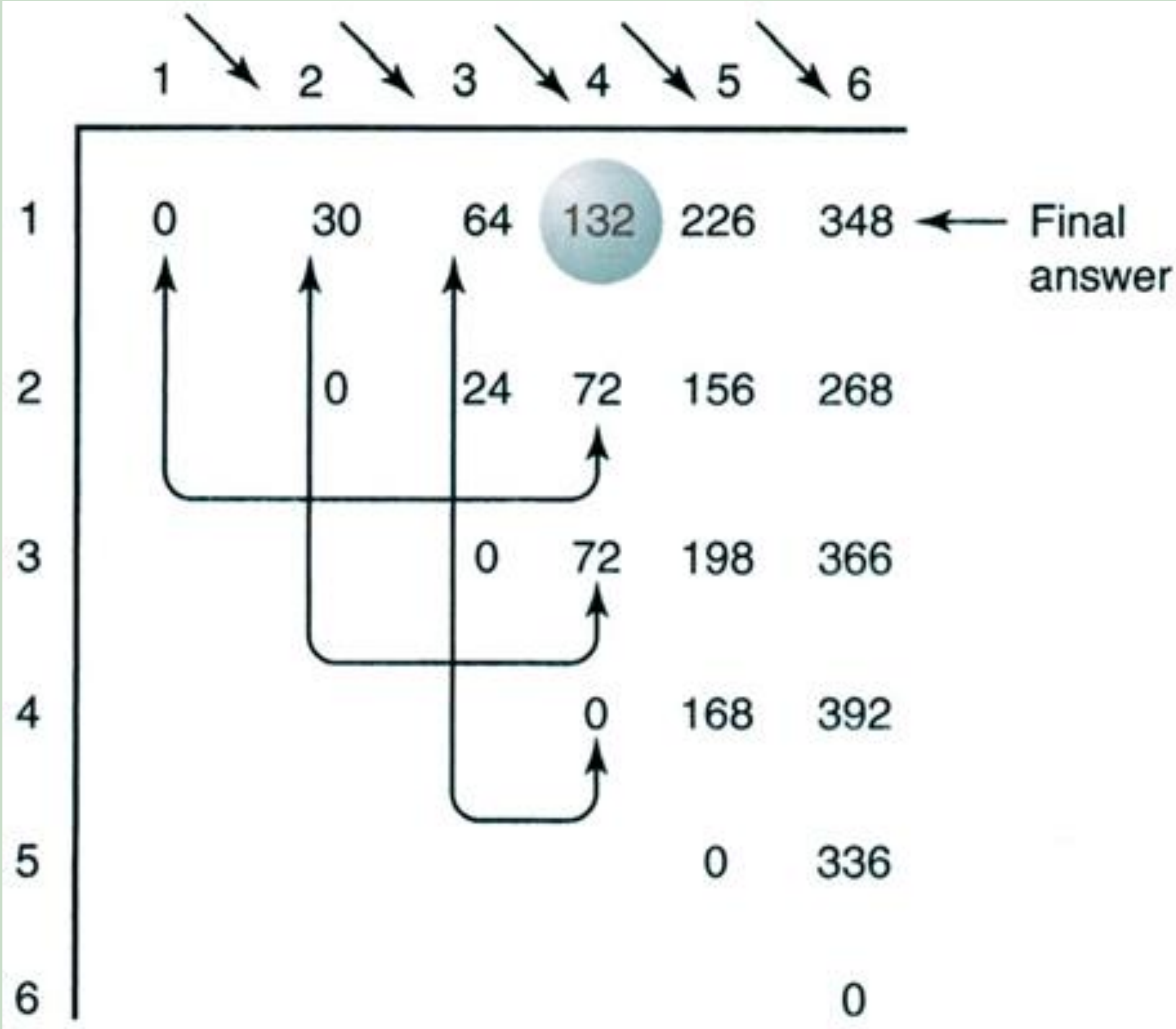
M[1][5] , M[6][6] ;

M[2][6] → M[2][2], M[3][6] ; M[2][3] , M[4][6] ;

M[2][4], M[5][6] ; M[2][5] , M[6][6] ;

……

避免重复计算的填表顺序.



矩阵连乘

计算第1条对角线:

$$
\begin{aligned}
M[1][2] &= \underset{1 \le k \le 1}{minimum}(M[1][k] + M[k+1][2] + d_0 d_k d_2) \\
&= M[1][1] + M[2][2] + d_0 d_1 d_2 \\
&= 0 + 0 + 5 \times 2 \times 3 = 30.
\end{aligned}
$$

The values of M[2][3], M[3][4], M[4][5], and M[5][6] are computed in the same way.

矩阵连乘

Compute diagonal 2:

$$
\begin{aligned}
M[1][3] &= \underset{1 \le k \le 2}{minimum}(M[1][k] + M[k+1][3] + d_0 d_k d_3) \\
&= minimum(M[1][1] + M[2][3] + d_0 d_1 d_3, \\
&\qquad M[1][2] + M[3][3] + d_0 d_2 d_3) \\
&= minimum(0 + 24 + 5 \times 2 \times 4, 30 + 0 + 5 \times 3 \times 4) = 64.
\end{aligned}
$$

The values of M[2][4], M[3][5], M[4][6] are computed in the same way.

矩阵连乘

Compute diagonal 3:

$$
\begin{aligned}
M[1][4] &= \underset{1 \le k \le 3}{minimum}(M[1][k] + M[k+1][4] + d_0 d_k d_4) \\
&= minimum(M[1][1] + M[2][4] + d_0 d_1 d_4, \\
&\qquad M[1][2] + M[3][4] + d_0 d_2 d_4) \\
&\qquad M[1][3] + M[4][4] + d_0 d_3 d_4) \\
&= minimum(0 + 72 + 5 \times 2 \times 6, 30 + 72 + 5 \times 3 \times 6, \\
&\qquad 64 + 0 + 5 \times 4 \times 6) = 132.
\end{aligned}
$$

The values of M[2][5], M[3][6] are computed in the same way.

矩阵连乘

Compute diagonal 4:

……

Compute diagonal 5:

……

M[1][6] = 348

矩阵连乘

diagonal 1: M[1][2],M[2][3], M[3][4], M[4][5], M[5][6]

diagonal 2: M[1][3],M[2][4], M[3][5], M[4][6]

diagonal 3: M[1][4],M[2][5], M[3][6]

diagonal 4: M[1][5],M[2][6]

diagonal 5: M[1][6]

矩阵连乘

```
int minmult (  int n,  const int d[ ],   index P[ ][ ] )
{   index   i,   j,   k,   diagonal;
    int M [1 .. n][1 .. n];

    for (i = 1; i < = n; i++)
        M[i][i] = 0;

    for (diagonal = 1; diagonal < = n - 1; diagonal ++)
        for (i = 1; i <= n - diagonal; i++)
        {    j = i + diagonal;
             M[i][j] =  minimum( M[i][k] + M[k + 1][j] + d[i-1]* d[k] * d[j] );
                          i ≤ k ≤ j - 1
             P[i][j] = a value of k that gave the minimum;
        }
   return M[1][n];
}
```

矩阵连乘

- Every-Case Time Complexity
  - The first two loops (for{ for{ } }) :

    n-1, n-2, n-3, ……., 2, 1
  - The minimum imply the third loop:

    j-i = (i+diagonal) – i = diagnoal

$$\sum_{diagonal=1}^{n-1} [(n - diagonal) \times diagonal]$$

$$\frac{n(n-1)(n+1)}{6} \in \Theta(n^3)$$

矩阵连乘

```
void order (index i,  index j)
{     if (i == j)
              cout << "A" << i;
      else
      {     k = P[i][j];
              cout << "(";
              order (i, k);
              order (k + 1, j);
              cout << ")";
       }
}
```

- Print Optimal Order

算法分析

- 语句int t=m[i][k]+m[k+1][j]+p[i-1]*p[k]*p[j]; 为算法**MatrixChain**的基本语句，最坏情况下该语句的执行次数为**O(n³)**，故该算法的最坏时间复杂性为**O(n³)**。
- 构造最优解的**Traceback**算法的时间主要取决于递归。最坏情况下时间复杂性的递归式为：

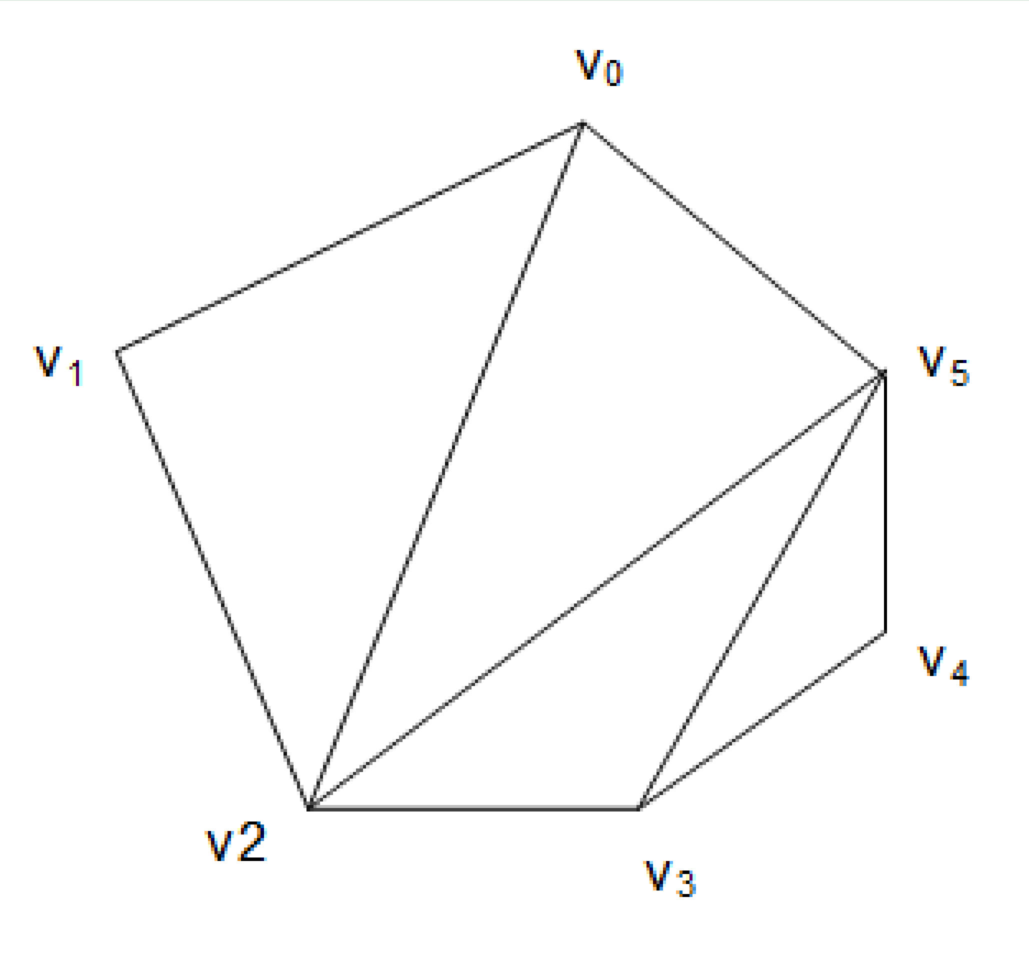
$$T(n)=\begin{cases} O(1) & n=1 \\ T(n-1)+O(1) & n>1 \end{cases}$$

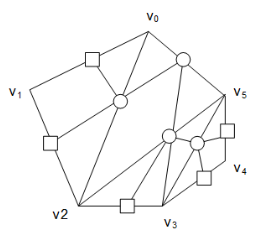
解此递归式得**T(n)=O(n)**。

凸多边形最优三角剖分

- 基本概念
  - 一个简单多边形及其内部构成一个闭凸集时，称该简单多边形为凸多边形 。
  - 凸多边形的不相邻的两个顶点连接的直线段称为凸多边形的弦 。
  - 凸多边形的三角剖分指将一个凸多边形分割成互不相交的三角形的弦的集合 。
  - 给定凸多边形及定义在边、弦构成的三角形的权函数，最优三角剖分即不同剖分方法所划分的各三角形上权函数之和最小的三角剖分。

三角剖分的结构





凸多边形最优剖分问题的解决方法和矩阵连乘问题相似。

(A1 A2) (A3 (A4 A5))

最优子结构性质分析

- 设$v_0v_kv_n$是将n+1边形P={v_0,v_1, …,v_n}分成三部分{v_0,v_1,…,v_k}、{v_k,v_{k+1},…,v_n}和{v_0,v_k,v_n}的最佳剖分方法，那么凸多边形{v_0,v_1,…,v_k}的剖分一定是最优的，{v_k,v_{k+1},…,v_n}的剖分也一定是最优的。
- 设{v_0,v_1, …,v_n}三角剖分的权函数之和为c，{v_0,v_1,…,v_k}三角剖分的权函数之和为a，{v_k,v_{k+1},…,v_n}三角剖分的权函数之和为b，三角形$v_0v_kv_n$的权函数为w($v_0v_kv_n$)，则c=a+b+ w($v_0v_kv_n$)。
- 如果c是最小的，则一定包含a和b都是最小的。如果a不是最小的，则它所对应的{v_0,v_1,…,v_k}的三角剖分就不是最优的。那么，对于凸多边形{v_0,v_1,…,v_k}来说，肯定存在最优的三角剖分，设{v_0,v_1,…,v_k}的最优三角剖分对应的权函数之和为a' (a'<a)，用a'代替a得到c'=a'+b+w($v_0v_k$vn)，则c'<c，这说明c对应的{v_0,v_1, …,v_n}的三角剖分不是最优的，产生矛盾。故a一定是最小的。同理，b也是最小的。最优子结构性质得证。

建立最优值的递归关系式

- 设**m[i][j]**表示$v_{i-1}v_i\ldots v_j$最优三角剖分权函数之和，**i=j**时表示一条直线段，将其看作退化多边形，其权函数为**0**。则

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

最长公共子序列问题

- ## 基本概念

  - ### （1）子序列

  给定序列 $X=\{x_1, x_2, \ldots, x_n\}$、$Z=\{z_1, z_2, \ldots, z_k\}$，若Z是X的子序列，当且仅当存在一个严格递增的下标序列 $\{i_1, i_2, \ldots, i_k\}$，对 $j\in\{1, 2, \ldots, k\}$ 有 $z_j=x_{i_j}$。

  - ### （2）公共子序列

  给定序列X和Y，序列Z是X的子序列，也是Y的子序列，则称Z是X和Y的公共子序列。

  - ### （3）最长公共子序列

  包含元素最多的公共子序列即为最长公共子序列。

最长公共子序列问题

- 例

X={A,B,C,B,D,A,B}

X的子序列：{A,B} {B,C,A} {A,B,C,D,A} …

Y={A,C,B,E,D,B}

XY的公共子序列：{A,B} {C,B,D} {A,C,B,D,B} …

XY的最长公共子序列：{A,C,B,D,B}

最长公共子序列问题

- 寻找具有递推关系的最优子结构
  - 从左向右切分，还是从右向左切分？与数据结构和代码方便程度有关。

X = {A,B,C,B,D,A,B}

↑
i

Y = {A,C,B,E,D,B}

↑
j

建立最优值的递归关系式

- 设c[i][j]表示序列X_i和Y_j的最长公共子序列的长度。则：

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

算法设计

- **步骤1**：确定合适的数据结构。采用数组**c**来存放各个子问题的最优值，数组**b**来存放各个子问题最优值的来源。数组**x[1:m]**和**y[1:n]**分别存放**X**序列和**Y**序列；
- **步骤2**：初始化。令**c[i][0]**＝**0**，**c[0][j]=0**，其中**0≤i≤m**，**0≤j≤n**；
- **步骤3**：循环阶段。根据递归关系式，确定序列**Xi**和**Y**的最长公共子序列长度；
- **步骤3-1**：**i=1**时，求出**c[1][j]**，同时记录**b[1][j]**，**1≤j≤n**；
- **步骤3-2**：**i=2**时，求出**c[2][j]**，同时记录**b[2][j]**，**1≤j≤n**；
- 依此类推，直到……
- **步骤3-m**：**i=m**时，求出**c[m][j]**，同时记录**b[m][j]**，**1≤j≤n**。此时，**c[m][n]**便是序列**X**和**Y**的最长公共子序列长度；
- **步骤4**：根据二维数组**b**记录的相关信息以自底向上的方式来构造最优解；
- **步骤4-1**：初始时，**i=m**，**j=n**；
- **步骤4-2**：如果**b[i][j]=1**，则输出**x[i]**，同时递推到**b[i-1][j-1]**; 如果**b[i][j]=2**，则递推到**b[i][j-1]**; 如果**b[i][j]=3**，则递推到**b[i-1][j]**；
- 重复执行步骤**4-2**，直到**i=0**或**j=0**，此时就可得到序列**X**和**Y**的最长公共子序列。

实例构造

- 【例4-6】给定序列X={A, B, C, B, D, A, B}和Y={B, D, C, A, B, A}，求它们的最长公共子序列。
  - 1. m=7，n=6，将停止条件填入数组c中，即c[i][0]＝0，c[0][j]=0，其中0≤i≤m，0≤j≤n。
  - 2. 当i=1时，X_1={A}，最后一个字符为A；Y_j的规模从1逐步放大到6，其最后一个字符分别为B、D、C、A、B、A；
  - 1. 依此类推，直到i=7。

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

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

从**i=7，j=6**处向前递推，找到**X**和**Y**的最长公共子序列为**:**

{B,C,B,A}　{B,C,A,B}　{B,D,A,B}　……

填表的时间复杂度 $O(n\lg n)$

构造最优解的时间复杂度 $O(\max(n,m))$

加工顺序问题

- 问题描述
- 设有n个工件需要在机器Ml和M2上加工，每个工件的加工顺序都是先在Ml上加工，然后在M2上加工。t_{1j}，t_{2j}分别表示工件j在M1，M2上所需的加工时间(j=1,2,…,n)。问应如何在两机器上安排生产，使得第一个工件从在M1上加工开始到最后一个工件在M2上加工完所需的总加工时间最短？

最优子结构性质分析

- 将n个工件的集合看作N={1,2,…,n}，设P是给定n个工件的一个最优加工顺序方案，P(i)是该调度方案的第i个要调度的工件(i=1,2,…,n)。

- 从集合S中的第一个工件开始在机器M1上加工到最后一个工件在机器M2上加工结束时所耗的时间为T(S，t)。设集合S的最优加工顺序中第一个要加工的工件为i，那么，经过的时间，进入的状态为第一台机器M1开始加工集合S-{i}中的工件时，第二台机器M2需要时间才能空闲下来，这种情况下机器M2加工完S-{i}中的工件所需的时间为T(S-{i}，t')，其中，即 $t'=t_{2i}+\max\{t-t_{1i}，0\}$，则

最优子结构性质分析

- $T(S，t)= t_{1i} +T(S-\{i\}，t_{2i} +max\{t- t_{1i}，0\})$ （4-1）
- 从式（4-1）可以看出，如果T(S，t)是最小的，那么肯定包含$T(S-\{i\}, t_{2i} +max\{t- t_{1i}，0\})$也是最小的。整体最优一定包含子问题最优。

建立最优值的递归关系式

- 设**T(S，t)**表示从集合**S**中的第一个工件开始在机器**M1**上加工到最后一个工件在机器**M2**上加工结束时所耗的最短时间，则：
- 当**S**为空集时，耗时为**M2**闲下来所需要的时间，即**T(S，t)=t**；
- 当**S**不为空集时，

$$T(S,t)=\min_{i\in S}\{t_{1i}+T(S-\{i\},t_{2i}+\max\{t-t_{1i},0\})\}$$

Johnson-Bellman's Rule

- 如果加工工件i和j满足min{ t_{1j}，t_{2i}}大于等于min{ t_{1i}，t_{2j}}不等式，称加工工件i和j满足Johnson Bellman's Rule。设最优加工顺序为P，则P的任意相邻的两个加工工件P(i)和P(i+1)满足
- 进一步可以证明，最优加工顺序的第i个和第j个要加工的工件，如果i<j，则

$$\min\{t_{1P(i+1)}, t_{2P(i)}\} \geq \min\{t_{1P(i)}, t_{2P(i+1)}\}$$

- 即：满足Johnson Bellman's Rule的加工顺序方案为最优方案。

$$\min\{t_{1P(j)}, t_{2P(i)}\} \geq \min\{t_{1P(i)}, t_{2P(j)}\}$$

算法设计

- 步骤1：令$N_1=\{i|t_{1i}<t_{2i}\}$,$N_2=\{i|t_{1i}t_{2i}\}$；
- 步骤2：将N_1中工件按t_{1i}非减序排序；将N_2中工件按t_{2i}非增序排序；
- 步骤3：N_1中工件接N_2中工件，即N_1N_2就是所求的满足**Johnson Bellman's Rule**的最优加工顺序

算法分析

- 显然，**FlowShop**算法的时间复杂性取决于**Sort**函数的执行时间，由于**Sort**函数的执行时间为**O(nlogn)**，因此**FlowShop**算法的时间复杂性为**O(nlogn)**。

0-1背包问题

- **问题描述**
  - **0-1背包问题可描述为：n个物品和1个背包。对物品i，其价值为v_i，重量为w_i，背包的容量为W。如何选取物品装入背包，使背包中所装入的物品的总价值最大？**
  - 约束条件：

$$\begin{cases} \sum_{i=1}^{n} w_i x_i \le W \\ x_i \in \{0,1\},(1 \le i \le n) \end{cases} \qquad \text{（4-7）}$$

  - 目标函数：

$$\max \sum_{i=1}^{n} v_i x_i \qquad \text{（4-8）}$$

  - 于是，问题归结为寻找一个满足约束条件（**4-7**），并使目标函数（**4-8**）达到最大的解向量**X=(x1, x2,…, xn)**。

最优子结构性质分析

- 假设（x1, x2,…, xn)是所给0-1背包问题的一个最优解，则（x2,…, xn）是下面相应子问题的一个最优解：
- 约束条件：

$$\begin{cases} \sum_{i=2}^{n} w_i x_i \le W - w_1 x_1 \\ x_i \in \{0,1\}, (2 \le i \le n) \end{cases}$$

- 目标函数：

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

运用反证法来证明

建立最优值的递归关系式

- 令C[i][j]= $\max \sum_{k=1}^{i} v_k x_k \leq j$
- C[0][j]=C[i][0]=0　　（4-10)

$$C[i][j]=\begin{cases} C[i-1][j] & j<w_i \\ \max\{C[i-1][j],C[i-1][j-w_i]+v_i\} & j \geq w_i \end{cases}$$

- （4-11)

算法设计

- 步骤1：设计算法所需的数据结构。采用数组w[n]来存放n个物品的重量；数组v[n]来存放n个物品的价值，背包容量为W，数组C[n+1][W+1]来存放每一次迭代的执行结果；数组x[n]用来存储所装入背包的物品状态；
- 步骤2：初始化。按式(4-10)初始化数组C；
- 步骤3：循环阶段。按式（4-11）确定前i个物品能够装入背包的情况下得到的最优值；
  - 步骤3-1：i=1时，求出C[1][j]，1≤j≤W；
  - 步骤3-2：i=2时，求出C[2][j]，1≤j≤W；
  - 依此类推，直到……
  - 步骤3-n：i=n时，求出C[n][W]。此时，C[n][W]便是最优值；

填表

- 例:5个物品，w(2,2,6,5,4)，v(6,3,5,4,6)，W=10，求0-1背包解。
- 行i表示物品，列j表示背包容量，表中数据表示C[i][j]

$$C[i][j]=\begin{cases} C[i-1][j] & j<w_i \\ \max\{C[i-1][j],C[i-1][j-w_i]+v_i\} & j\geq w_i \end{cases}$$

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6	6	6
2	0	0	6	6	9	9	9	9	9	9	9
3	0	0	6	6	9	9	9	9	11	11	14
4	0	0	6	6	9	9	9	10	11	13	14
5	0	0	6	6	9	9	12	12	15	15	15

构造解

- 从C[n][W]的值向前推，如果C[n][W]> C[n-1][W]，表明第n个物品被装入背包，则x_n=1，前n-1个物品被装入容量为W-w_n的背包中；否则，第n个物品没有被装入背包，则x_n=0，前n-1个物品被装入容量为W的背包中。依此类推，直到确定第1个物品是否被装入背包中为止。由此，得到下面关系式：
- $$\begin{cases} x_i = 0, j = j & 当C[i][j] = C[i-1][j] \\ x_i = 1, j = j - w_i & 当C[i][j] > C[i-1][j] \end{cases} \quad （4\text{-}12）$$
- 按照式（4-12），从C[n][W]的值向前倒推，即j初始为W，i初始为n，即可确定装入背包的具体物品。

确定装入背包的具体物品

- 从C[n][W]的值根据式（4-12）向前推，最终可求出装入背包的具体物品，即问题的最优解。
- 初始时，j=W，i=5。
- 如果C[i][j]=C[i-1][j]，说明第i个物品没有被装入背包，则xi =0；
- 如果C[i][j]>C[i-1][j]，说明第i个物品被装入背包，则xi =1，j=j-wi。
- 由于C[n][W]=C[5][10]=15>C[4][10]=14，说明物品5被装入了背包，因此x5=1，且更新j=j-w[5]=10-4=6。由于C[4][j]=C[4][6]=9=C[3][6]，说明物品4没有被装入背包，因此x4 =0；由于C[3][j]=C[3][6]=9=C[2][6]=9，说明物品3没有被装入背包，因此x3=0。由于C[2][j]=C[2][6]=9>C[1][6]=6，说明物品2被装入了背包，因此x2=1，且更新j=j-w[2]=6-2=4。由于C[1][j]=C[1][4]=6>C[0][4]=0，说明物品1被装入了背包，因此x1=1，且更新j=j-w[1]=4-2=2。最终可求得装入背包的物品的最优解X=(x1, x2,…, xn)=(1,1,0,0,1)。

构造解

$$\begin{cases} x_i = 0, j = j & 当C[i][j] = C[i-1][j] \\ x_i = 1, j = j - w_i & 当C[i][j] > C[i-1][j] \end{cases}$$

w(2,2,6,5,4)
V (6,3,5,4,6)

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6	6	6
2	0	0	6	6	9	9	9	9	9	9	9
3	0	0	6	6	9	9	9	9	11	11	14
4	0	0	6	6	9	9	9	10	11	13	14
5	0	0	6	6	9	9	12	12	15	15	15

算法分析

- 在算法KnapSack中，第3个循环是两层嵌套的for循环，为此，可选定语句if(j<w[i])作为基本语句，其运行时间为n*W，由此可见，算法KnapSack的时间复杂性为O(nW)。
- 该算法有两个较为明显的缺点：一是算法要求所给物品的重量w_i是整数；二是当背包容量W很大时，算法需要的计算时间较多，例如，当$W>2^n$时，算法需要$O(n2^n)$的计算时间。因此，在这里设计了对算法KnapSack的改进方法，采用该方法可克服这两大缺点。

改进算法

$$C[i][j]=\begin{cases} C[i-1][j] & j<w_i \\ \max\{C[i-1][j],C[i-1][j-w_i]+v_i\} & j\geq w_i \end{cases}$$

- 改进思路
  - 由**C[i][j]**的递归式（**4-11**）容易证明：在一般情况下，对每一个确定的**i(1≤i≤n)**，函数**C[i][j]**是关于变量**j**的阶梯状单调不减函数（事实上，计算**C[i][j]**的递归式在变量**j**是连续变量，即为实数时仍成立）。跳跃点是这一类函数的描述特征。在一般情况下，函数**C[i][j]**由其全部跳跃点唯一确定。

改进步骤

- （a）对每一个确定的i，用一个表p[i]来存储函数C[i][j]的全部跳跃点。对每一个确定的实数j，可以通过查找p[i]来确定函数C[i][j]的值。p[i]中的全部跳跃点(j，C[i][j])按j升序排列。由于函数C[i][j]是关于j的阶梯状单调不减函数，故p[i]中全部跳跃点的C[i][j]值也是递增排列的。
- （b）p[i]可通过计算p[i-1]得到。
- （c）清除受控点。
- （d）由此可得，在递归地由p[i-1]计算p[i]时，可先由p[i-1]计算出q[i-1]，然后合并p[i-1]和q[i-1]，并清除其中的受控跳跃点得到p[i]。
- 改进后算法的计算时间复杂性为$O(\min\{nW，2^n\})$

最优二叉查找树

- 定义
  - 最优二叉查找树是在所有表示有序序列**S**的二叉查找树中，具有最小平均比较次数的二叉查找树。
  - 注意：在查找概率不等的情况下，最优二叉树并不一定是高度最小的二叉查找树。

最优子结构性质分析

- 将由实结点{$s_1,s_2,\ldots,s_n$}和虚结点{$e_0,e_1, \ldots,e_n$}构成的二叉查找树记为T(1，n)。设定元素sk作为该树的根结点，1≤k≤n。则二叉查找树T(1，n)的左子树由实结点{$s_1,\ldots,s_{k-1}$}和虚结点$e_0,\ldots,e_{k-1}$组成，记为T(1，k-1)，而右子树由实结点{$s_{k+1},\ldots,s_n$}和虚结点$e_k,\ldots,e_n$组成，记为T(k+1，n)。

- 如果T(1，n)是最优二叉查找树，则左子树T(1，k-1)和右子树T(k+1，n)也是最优二叉查找树。如若不然，假设T' (k+1，n)是比T(k+1，n)更优的二叉查找树，则T' (k+1，n)的平均比较次数小于T(k+1，n)的平均比较次数，从而由T(1，k-1)、sk和T' (k+1，n)构成的二叉查找树T' (1，n)的平均比较次数小于T(1，n)的平均比较次数，这与T(1，n)是最优二叉树的前提相矛盾。因此，最优二叉查找树具有最优子结构性质得证。

建立最优值的递归关系式

- $$C(i, j) = w(i, j) + \min_{i \le k \le j}\{C(i, k-1) + C(k+1, j)\}$$ (4-21)
- 其中 $$w(i, j) = w(i, j-1) + p_j + q_j$$ (4-22)
- 初始时，C(i，i-1)=0; $w_{i(i-1)}=q_{i-1}$，其中1≤i≤n。 (4-23)
- 式（4-21）和（4-23）即为建立的最优值递归定义式。

算法设计

- 步骤1：设计合适的数据结构。设有序序列S={s1,…,sn}，数组s[n]存储序列S中的元素；数组p[n]存储序列S中相应元素的查找概率；二维数组C[n+1][n+1]，其中C[i][j]表示二叉查找树T(i，j)的平均比较次数；二维数组R[n+1][n+1]，其中R[i][j]表示二叉查找树T(i，j)中作为根结点的元素在序列S中的位置。数组q[n]存储虚结点$e_0,e_1,\ldots,e_n$的查找概率。为了提高效率，不是每次计算C(i，j)时都计算w_{ij}的值，而是把这些值存储在二维数组W[i][j]中；
- 步骤2：初始化。设置C[i][i-1]=0；W[i][i-1]=q_{i-1}；
- 步骤3：循环阶段。采用自底向上的方式逐步构造最优二叉查找树；
- 步骤3-1：字符集规模为1的时候，即$S_{ij}=\{s_i\}$，i=1,2, …,n且j=i，显然这种规模的子问题有n个，即首先要构造出n棵最优二叉查找树T(1，1), T(2，2),…,T(n，n)。依据公式(4-20)~(4-22)，很容易求得W[i][i]和C[i][i]。同时，对于所构造的n棵最优二叉查找树，它们的根分别记为：R[1][1]=1，R[2][2]=2,…,R[n][n]=n；

- 依此类推，构造出字符集Sij中含3个字符的最优二叉查找树、含4个字符的最优二叉查找树，直到……
- 步骤3-n: 字符集规模为n的时候，即S1n={s1,s2,…,sn}，显然这种规模的子问题有1个，即要构造出1棵最优二叉查找树T(1，n)。依据公式(4-21)，求得W[i][j]，然后在整数1,2…n中选择适当的k值，使得成立。同时，记录该树的根R[1][n]=k;
- 步骤4：最优解的构造。
- 从R[i][j]中保存的最优二叉查找子树T(i，j)的根结点信息，可构造出问题的最优解。当R[1][n]=k时，则元素sk即为所求的最优二叉查找树的根结点。此时，需要计算两个子问题：求左子树T(1，k-1)和右子树T(k+1，n)的根结点信息。如果R[1][k-1]=i，则元素si即为T(1，k-1)的根结点元素。依此类推，将很容易由R中记录的信息构造出问题的最优解。

构造实例

- 【例4-11】设5个有序元素的集合为{s1,s2,s3,s4,s5},查找概率p=<p1,p2,p3,p4,p5>= <0.15,0.1,0.05,0.1,0.2>；叶结点元素{e0,e1,e2,e3,e4,e5}，查找概率q=<q0,q1,q2,q3,q4,q5>= <0.05,0.1,0.05,0.05,0.05,0.1>。试构造5个有序元素的最优二叉查找树。

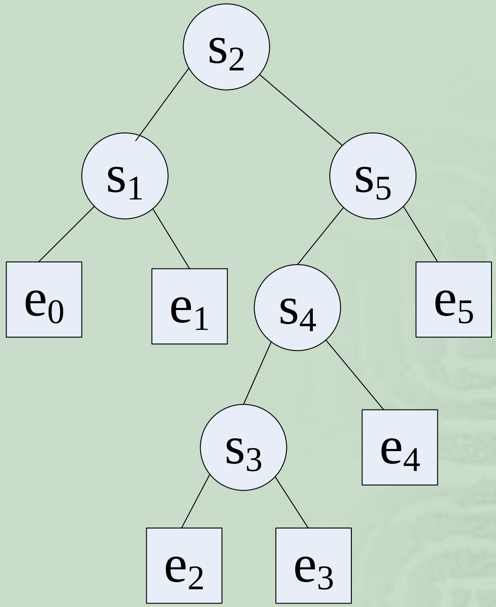
W (i \ j)	0	1	2	3	4	5
1	0.05	0.3	0.45	0.55	0.7	1.0
2		0.1	0.25	0.35	0.5	0.8
3			0.05	0.15	0.3	0.6
4				0.05	0.20	0.5
5					0.05	0.35
6						0.1

C (i \ j)	0	1	2	3	4	5
1	0	0.3	0.7	1.0	1.45	2.35
2		0	0.25	0.5	0.95	1.65
3			0	0.15	0.45	1.05
4				0	0.20	0.7
5					0	0.35
6						0

R (i \ j)	0	1	2	3	4	5
1		1	1	2	2	2
2			2	2	2	4
3				3	4	5
4					4	5
5						5

根据R中的信息构造最优解

- 步骤1：由于R[1][5]=2，即k=2，最优二叉查找树T(1，5)的根结点为s2；
- 步骤2：求出T(1，5)的左子树T(1，k-1)=T(1，1)的根结点为s1；
- 步骤3：求出T(1，5)的右子树T(k+1，5)=T(3，5)的根结点为s5；
- 步骤4：求出子树T(3，5)的左子树T(3，4)的根结点为s4；
- 由此构造出如图4-9所示的最优二叉查找树



算法分析

- 由算法描述容易看出，语句if((C[i][k-1]+C[k+1][j])＜C[i][j])对算法的运行时间贡献最大，因此，可选该语句作为基本语句。对于固定的t值，该语句需要的计算时间为O(j-i)=O(t)，因此，它总的运行时间为

$$\sum_{t=0}^{n-1}\sum_{i=1}^{n-t}O(t)=O(n^3)$$

备忘录法

- Fib数列

```
global Memo M[0 .. n];
M[0] = 0; M[1] = 1;
M[2 .. n] = -1;

int fib3(int n)  {
      if (M[n] == -1) {
            M[n] = fib3(n - 1) + fib3(n-2);
      }
      return M[n];
}
```

备忘录法

矩阵连乘

$$M[i][j] = \min_{i \leq k \leq j-1} \left(M[i][k] + M[k+1][j] + d_{i-1} d_k d_j \right) \quad \text{if } i < j$$

$$M[i][i] = 0$$

How ?

Memo Method

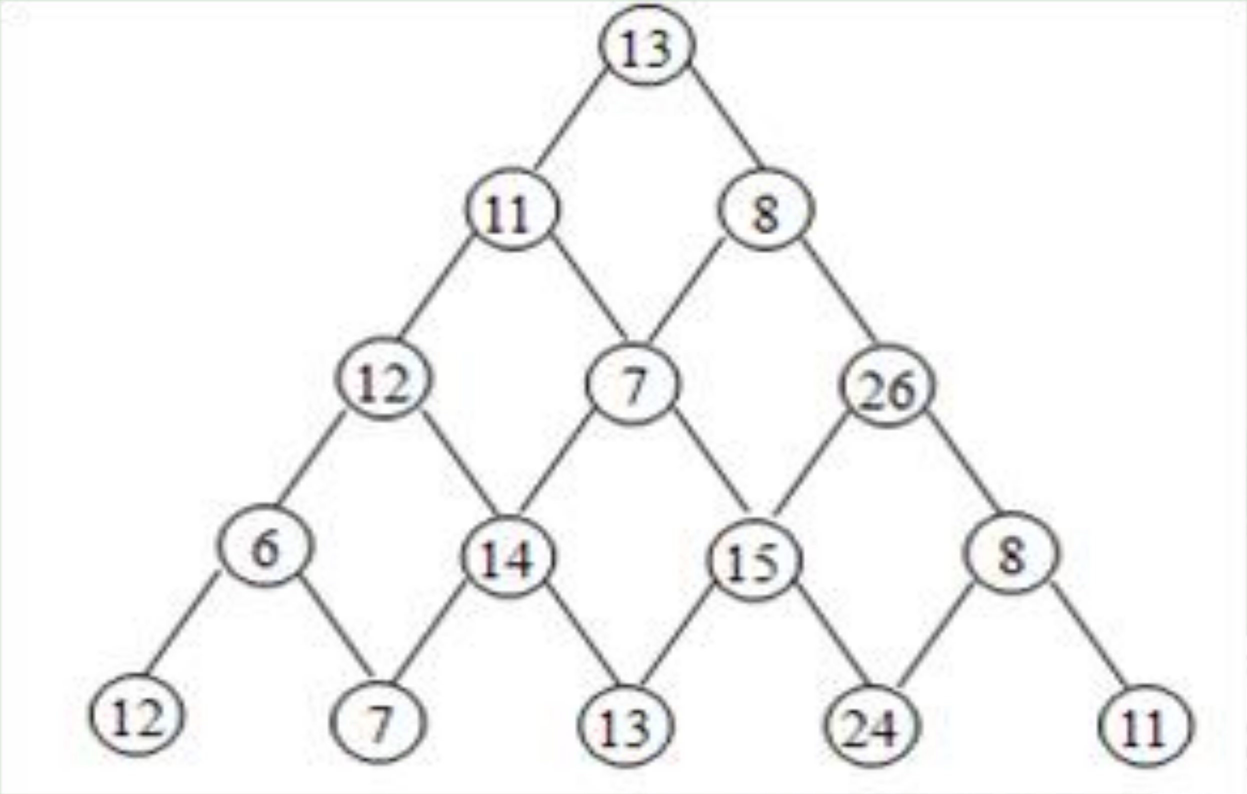
矩阵连乘

```
global Memo M[1..n][1..n]=-1;
global int d[0..n]={ … };
global index P[1..n][1..n];

int minmult ( int i,  int j )  {
    if( i==j )
        M[i][j]=0;
    else if ( M[i][j] == -1) {
        M[i][j] = min( minmult(i, k) + minmult(k+1, j) + d[i-1]* d[k] * d[j] );
                     i ≤ k ≤ j-1
        P[i][j] = a value of k that gave the minimum;
    }
    return M[i][j];
}
```

思考题

- 1. 数塔问题



思考题

- 2. 最大子段和

给定n个整数（可能为负数）组成的序列：

a[1],a[2],a[3],…,a[n]

求该序列如a[i]+a[i+1]+…+a[j]的子段和的最大值。当所有整数均为负值时定义其最大子段和为0。

例如：

（a1,a2,a3,a4,a4,a6）=(-2,11,-4,13,-5,-2)时，

最大子段和为20。

思考题

- 3. 神奇的双递归函数Ackerman

$A(1, 0) = 2$	
$A(0, m) = 1$	$m >= 0$
$A(n, 0) = n + 2$	$n >= 2$
$A(n, m) = A(A(n-1, m), m-1)$	$n, m >= 1$

$A(n,1)=2n$

$A(n,2)=2^n$

$A(n,3) = 2^{2^{2^{2\cdots^{2}}}}$ 其中2的层数为n

A(n,4) = 囧　增长太快，没有合适的数学表达方式

思考题

- 4. 找零钱问题