计算机算法导引
——设计与分析

卢开澄　等编著

清华大学出版社

目　录

第1章 动态规划

1.1 最短路径问题

数学规划是研究最优化的一类数学问题，包含有线性规划，非线性规划等内容。和动态规划有什么关系呢？动态规划实际上是研究一类最优化问题的算法，应用范围十分广。下面通过若干实例，介绍如何将一个最优化问题通过动态规划来求解的基本原理。

如图 1.1.1，从 A_0 点要铺设一条管道到 A_6 点，中间必须经过 5 个中间站，第一站可以在 A_1，B_1 两地中任选一个，类似地，第二、三、四、五站可供选择的地点分别是：$\{A_2,B_2,C_2,D_2\}$，$\{A_3,B_3,C_3\}$，$\{A_4,B_4,C_4\}$，$\{A_5,B_5\}$。连接两地间管道的距离（或造价）用连线上的数字表示，要求选一条从 A_0 到 A_6 的铺管线路，使总距离最短（或总造价最小）。

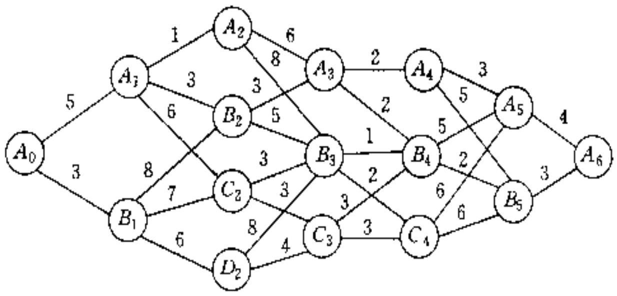


图 1.1.1

我们首先想到使用穷举法。在第一段，有两种路径选择：A_0A_1、A_0B_1。在第二段，若选 A_0A_1，第二段路径有三种选择：A_1A_2、A_1B_2、A_1C_2；若选 A_0B_1，也有三种选择：B_1B_2、B_1C_2、B_1D_2。所以两段共有 6 种选择。依次类推，从 A_0 到 A_6 共有 $2\times3\times2\times2\times2\times1=48$ 种不同路径。可通过 $48\times5=240$ 次加法，47 次比较，即通过各种可能方案的穷举，最后可求出从 A_0 到 A_6 的最短路径是：

$$A_0 \to A_1 \to B_2 \to A_3 \to B_4 \to B_5 \to A_6$$

相应的最短距离是 18。

但我们注意到最短路径有这样一个特性，即如果最短路径的第 k 站通过 P_k，则这一最短路径在由 P_k 出发到达终点的那一部分路径，对于始点为 P_k 到终点的所有可能的路径来说，必定也是距离最短的。这一特性很容易证明。读者可自己来完成它。

根据最短路径这一特性，启发我们计算时从最后一段开始，从后向前逐步递推的方法，求出各点到 A_6 的最短路径，最后求得从 A_0 到 A_6 的最短路径。步骤如下：

$k=6$ 时

设 $f(A_5)$ 表示由 A_5 到 A_6 的最短距离，$f(B_5)$ 表示由 B_5 到 A_6 的最短距离，显然有：

$$f(A_5) = 4, \qquad f(B_5) = 3。$$

$k=5$ 时

$$\begin{aligned} f(A_4) &= \min\{d(A_4,A_5) + f(A_5), d(A_4,B_5) + f(B_5)\} \\ &= \min\{\underline{3 + 4},\ 5 + 3\} = 7 \end{aligned}$$

这里括号里的底线____表示最小值所取的项。即 $f(A_4)$取的是 $A_4 \to A_5 \to A_6$，而不是 $A_4 \to B_5 \to A_6$。以后同此，不再说明。

$$\begin{aligned} f(B_4) &= \min\{d(B_4,A_5) + f(A_5), d(B_4,B_5) + f(B_5)\} \\ &= \min\{5 + 4, \underline{2 + 3}\} = 5 \\ f(C_4) &= \min\{d(C_4,A_5) + f(A_5), d(C_4,B_5) + f(B_5)\} \\ &= \min\{6 + 4, \underline{6 + 3}\} = 9 \end{aligned}$$

$k=4$ 时

$$\begin{aligned} f(A_3) &= \min\{d(A_3,A_4) + f(A_4), d(A_3,B_4) + f(B_4)\} \\ &= \min\{2 + 7, \underline{2 + 5}\} = 7 \\ f(B_3) &= \min\{d(B_3,B_4) + f(B_4), d(B_3,C_4) + f(C_4)\} \\ &= \min\{\underline{1 + 5}, 2 + 9\} = 6 \\ f(C_3) &= \min\{d(C_3,B_4) + f(B_4), d(C_3,C_4) + f(C_4)\} \\ &= \min\{\underline{3 + 5}, 3 + 9\} = 8 \end{aligned}$$

$k=3$ 时

$$\begin{aligned} f(A_2) &= \min\{d(A_2,A_3) + f(A_3), d(A_2,B_3) + f(B_3)\} \\ &= \min\{\underline{6 + 7}, 8 + 6\} = 13 \\ f(B_2) &= \min\{d(B_2,A_3) + f(A_3), d(B_2,B_3) + f(B_3)\} \\ &= \min\{\underline{3 + 7}, 5 + 6\} = 10 \\ f(C_2) &= \min\{d(C_2,B_3) + f(B_3), d(C_2,C_3) + f(C_3)\} \\ &= \min\{\underline{3 + 6}, 3 + 8\} = 9 \\ f(D_2) &= \min\{d(D_2,B_3) + f(B_3), d(D_2,C_3) + f(C_3)\} \\ &= \min\{8 + 6, \underline{4 + 8}\} = 12 \end{aligned}$$

$k=2$ 时

$$\begin{aligned} f(A_1) &= \min\{d(A_1,A_2) + f(A_2), d(A_1,B_2) + f(B_2), d(A_1,C_2) + f(C_2)\} \\ &= \min\{1 + 13, \underline{3 + 10}, 6 + 9\} = 13 \\ f(B_1) &= \min\{d(B_1,B_2) + f(B_2), d(B_1,C_2) + f(C_2), d(B_1,D_2) + f(D_2)\} \\ &= \min\{8 + 10, \underline{7 + 9}, 6 + 12\} = 16 \end{aligned}$$

$k=1$ 时

$$\begin{aligned} f(A_0) &= \min\{d(A_0,A_1) + f(A_1), d(A_0,B_1) + f(B_1)\} \\ &= \min\{\underline{5 + 13}, 3 + 16\} = 18 \end{aligned}$$

上述计算结果可表示如图 1.1.2，其中$\dfrac{A_5}{4}$表示从 A_5 出发到终点的最短路径长度为 4，即 $A_5 \xrightarrow{4} A_6$，余此类推。

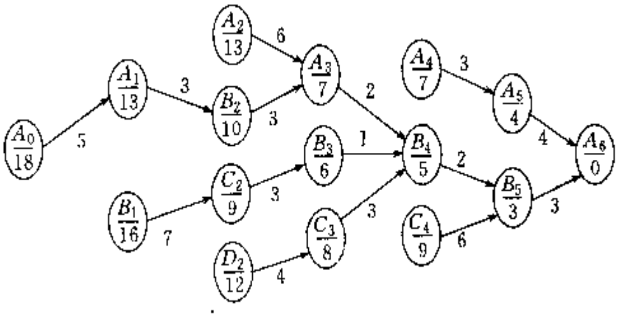


图 1.1.2

一共要用 15 次比较运算和 28 次加法运算就可得到从 A_0 到 A_6 的最短距离，而且在这过程中，还得到其它各点到 A_6 的最短路径和最短距离。

一般地，若我们考虑如图 1.1.3 所示从始点 $O(0,0)$ 到终点 $E(m,n)$ 的最短路径(也称格路)问题。若用穷举法，则需

$$(m+n-1)C(n+m,n)=\frac{(n+m-1)(n+m)!}{n!m!}$$

次加法及$\frac{(n+m)!}{n!\ m!}-1$ 次比较运算。组合数 $C(n+m,n)$ 为从 O 点到 E 点的路径数，这是考虑到图 1.1.3 的每一格路和 m 个 x，n 个 y 的任一排列一一对应。比如排列$\underbrace{x\ x\cdots x}_{m个}\underbrace{y\ y\cdots y}_{n个}$，对应于从 O 点先沿 x 方向走 m 块，后沿 y 方向走 n 块到 E 点。相当于由 m 个 y，n 个 x 构成的长度为 $m-n$ 的符号串数目，即从 $m+n$ 个格子中任选 m 个格子作为 x、其余为 y 的方案数。

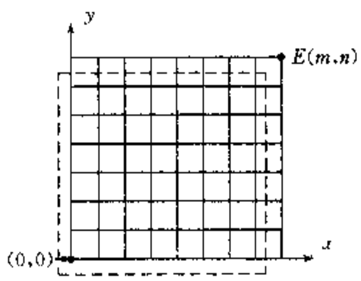


图 1.1.3

但用后一种方法只需 $2mn+m+n$ 次加法及 mn 次比较就够了。不难知道图 1.1.3 由虚线包围的矩形域内的点要作 2 次比较，2 次加法，在这以外的点只需 1 次加法，无需作比较。

当 $m=n$ 时，穷举法需进行 $(2n-1)\cdot(2n)!\ /(n!)^2$ 次加法，$(2n)!\ /(n!)^2-1$ 次比较。后一种方法只要作 $2(n^2+n)$次加法，n^2 次比较。由 Stirling 公式

$$n!\approx\sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n$$

可知穷举法的运算量是 n 的指数函数，后一种算法则只是 n^2 量级。

1.2 最佳原理

从前节例子知道：一个最短路径问题可变成多段判决问题，利用了最短路径的一个性质：从起点到终点的最短路径也是该路径上各点到终点的最短路径。与此类似的问题

很多，故可抽象成组合优化问题中的一个重要的最佳原理：假设为了解决某一优化问题，需要依次作出 n 个决策 $D_1, D_2, \cdots, D_n$，如若这个决策序列是最优的，对于任何一个整数 k，$1<k<n$，不论前面 k 个决策是怎样的，以后的最优决策只取决于由前面决策所确定的当前状态，即以后的决策 $D_{k+1}, D_{k+2}, \cdots, D_n$ 也是最优的。

本章的这节和以后各节主要举例说明如何灵活运用最佳原理。动态规划与线性、非线性规划不尽相同，实际它是

$$= \max_{0 \leq x_1 \leq 1000} \{50000x_1 + 40000000 - 40000x_1 + 70000(0.35x_1 + 600 - 0.60x_1)\}$$

$$= \max_{0 \leq x_1 \leq 1000} \{40000000 + 42000000 + 10000x_1 - 17500x_1\}$$

$$= \max_{0 \leq x_1 \leq 1000} \{82000000 - 7500x_1\}$$

$$= 82000000$$

即
$$x_1 = 0$$

故三年中生产计划要安排如下：

第一年 1000 台机器一律生产产品 P_2，

第二年把余下的机器继续生产产品 P_2，

第三年把所有的机器改为生产产品 P_1。

总收入为 82000000 元。

另外，有些数学问题也可以利用最佳原理将它转化为多段判决来解决。

例 2 在 $x_1+x_2+\cdots+x_n=a$ 的约束条件下，求 $x_1,x_2,\cdots,x_n$ 的值使

$$z = \sqrt{x_1} + \sqrt{x_2} + \cdots + \sqrt{x_n}$$

取极大值。

$$f_1(a) = \max_{0 \leq x \leq a} \sqrt{x} = \sqrt{a}$$

$$f_2(a) = \max_{0 \leq x \leq a} (\sqrt{x} + f_1(a - x)) = \max_{0 \leq x \leq a} (\sqrt{x} + \sqrt{a - x})$$

令 $y = \sqrt{x} + \sqrt{a-x}$

$$\frac{dy}{dx} = \frac{1}{2\sqrt{x}} - \frac{1}{2\sqrt{a-x}} = \frac{\sqrt{a-x} - \sqrt{x}}{2\sqrt{x(a-x)}} = 0$$

$$a - x = x,\quad x = \frac{a}{2},\quad 即\ x_1 = x_2 = x = \frac{a}{2}。$$

所以
$$f_2(a) = \left[\sqrt{\frac{a}{2}} + \sqrt{\frac{a}{2}}\right] = \sqrt{2a}$$

同样

$$f_3(a) = \max_{0 \leq x \leq a} (\sqrt{x} + f_2(a - x))$$
$$= \max_{0 \leq x \leq a} (\sqrt{x} + \sqrt{2(a - x)})$$

令 $y = \sqrt{x} + \sqrt{2(a-x)}$

$$\frac{dy}{dx} = \frac{1}{2\sqrt{x}} - \frac{\sqrt{2}}{2\sqrt{a-x}} = \frac{\sqrt{a-x} - \sqrt{2x}}{2\sqrt{2}\ \sqrt{x(a-x)}} = 0$$

$$a - x = 2x,\ x = \frac{a}{3},\quad 即\quad x_1 = x_2 = x_3 = \frac{a}{3}。$$

故
$$f_3(a) = \sqrt{\frac{1}{3}a} + 2\sqrt{\frac{1}{3}a} = \sqrt{3a}$$

用数学归纳法可以证明有：

$$f_n(a) = \sqrt{na}, x_1 = x_2 = \cdots = x_n = \frac{a}{n}$$

设 $f_{n-1}(a)=\sqrt{(n-1)a}$，$x_1=x_2=\cdots=x_{n-1}=\frac{a}{(n-1)}$，成立，

$$f_n(a) = \max_{0\leqslant x\leqslant a}\{\sqrt{x} + f_{n-1}(a-x)\}$$

$$= \max_{0\leqslant x\leqslant a}\{\sqrt{x} + \sqrt{(n-1)(a-x)}\}$$

令 $y=\sqrt{x}+\sqrt{(n-1)(a-x)}$

$$\frac{dy}{dx} = \frac{1}{2\sqrt{x}} - \frac{\sqrt{n-1}}{2\sqrt{a-x}} = \frac{\sqrt{a-x} - \sqrt{(n-1)x}}{2\sqrt{x(a-x)}} = 0$$

$$(n-1)x = a - x, \quad x = \frac{a}{n}$$

所以

$$x_1=x_2=\cdots=x_n=\frac{a}{n}$$

$$f_n(a) = \sqrt{\frac{a}{n}} + (n-1)\sqrt{\frac{a}{n}} = \sqrt{na}$$

例3 把正数 a 分成 n 个部分，使其乘积为最大。即 $x_1+x_2+\cdots+x_n=a$，使 $P_n(a)=x_1x_2\cdots x_n$ 达到最大。

设 $P_n(a)$为将 $a>0$ 分成 n 个部分的乘积的最大值。比如：

$$P_1(a) = a$$

$$P_2(a) = \max_{0\leqslant x\leqslant a}\{xP_1(a-x)\}$$

$$= \max_{0\leqslant x\leqslant a}\{x(a-x)\}$$

令 $y=x(a-x)$，$y'=a-2x$，即 $x_1=x_2=\frac{a}{2}$

所以，

$$P_2(a)=\left(\frac{a}{2}\right)^2$$

利用最佳原理得：

$$P_n(a) = \max_{0\leqslant x\leqslant a}\{xP_{n-1}(a-x)\}$$

并通过数学归纳法证明：

$$x_n^* = \frac{a}{n}, \quad P_n(a) = \left(\frac{a}{n}\right)^n$$

由于 $P_1(a)=a$ 成立。

设 $P_{n-1}(x)=\left(\frac{x}{n-1}\right)^{n-1}$，$x_{n-1}^*=\frac{a}{n-1}$成立，则

$$P_n(a) = \max_{0\leqslant x\leqslant a}\left\{x\left(\frac{a-x}{n-1}\right)^{n-1}\right\}$$

$$y = x\left(\frac{a-x}{n-1}\right)^{n-1}$$

$$\frac{dy}{dx} = \left(\frac{x}{n-1}\right)^{n-2}\left[\frac{a-nx}{n-1}\right] = 0$$

所以 $x_n^*=\frac{a}{n}$，即 $x_1=x_2=\cdots=x_n=\frac{a}{n}$

$$P_n(a)=\left(\frac{a}{n}\right)\left(\frac{a}{n}\right)^{n-1}=\left(\frac{a}{n}\right)^n$$

下面我们再来看一个离散型的问题，原理和上面的一样。

例 4 资源分配问题

设有资源 a 分配给 n 个项目，$g_i(x)$ 为将数量为 x 的资源分配给项目 i 所能得到的利润，$i=1,2,\cdots,n$ 最合理的资源分配导致解下面的问题：

$$\max z=g_1(x_1)+g_2(x_2)+\cdots+g_n(x_n)$$

$$x_1+x_2+\cdots+x_n=a$$

$$x_i\geqslant 0, i=1,2,\cdots,n$$

若 $g_i(x_i)$ 是 x_i 的线性函数，则是一般的线性规划问题。下面介绍如何利用最佳原理将其转化为多段判决问题：

设 $f_k(n)$ 为资源 n 分配给前 k 个项目所得的最大利润。

$$f_1(a)=\max_{0\leqslant x\leqslant a} g_1(x)$$

$$f_2(a)=\max_{0\leqslant x\leqslant a}\{g_2(x)+f_1(a-x)\}$$

$$f_3(a)=\max_{0\leqslant x\leqslant a}\{g_3(x)+f_2(a-x)\}$$

…… ……

$$f_n(a)=\max_{0\leqslant x\leqslant a}\{g_n(x)+f_{n-1}(a-x)\}$$

例如有 7 万元资本投资到 A、B、C 三种项目，其利润见表 1.2.1。

表 1.2.1

投资额 / 项目	1	2	3	4	5	6	7
A	0.12	0.15	0.20	0.21	0.24	0.30	0.36
B	0.22	0.24	0.26	0.28	0.30	0.33	0.34
C	0.18	0.22	0.26	0.28	0.30	0.34	0.36

1. 考虑投入产品 A 的生产资金 a 与利润 $f_1(a)$ 的关系见表 1.2.2。

表 1.2.2

a / f	1	2	3	4	5	6	7
$f_1(a)$	0.12	0.15	0.20	0.21	0.24	0.30	0.36

2. 若考虑投到 A、B 两种产品的生产，资金 a 与利润 $f_2(a)$ 关系如下，其中 x_2 为投到产品 B 的生产资金。

$$f_2(a) = \max_{0 \leqslant x_2 \leqslant a} \{g_2(x_2) + f_1(a - x_2)\}$$

其利润见表 1.2.3。其中 * 表示利润最大的状态。

表 1.2.3

a \ x_2	0	1	2	3	4	5	6	7	$f_2(a)$
1	0.12	0.22*							0.22
2	0.15	0.12+0.22 =0.34*	0.24						0.34
3	0.20	0.15+0.22 =0.37*	0.12+0.24 =0.36	0.26					0.37
4	0.21	0.20+0.22 =0.42*	0.15+0.24 =0.39	0.12+0.26 =0.38	0.28				0.42
5	0.24	0.21+0.22 =0.43	0.20+0.24 =0.44*	0.15+0.26 =0.41	0.12+0.28 =0.40	0.30			0.44
6	0.30	0.24+0.22 =0.46	0.21+0.24 =0.45	0.20+0.26 =0.46*	0.15+0.28 =0.43	0.12+0.30 =0.42	0.33		0.46
7	0.30	0.30+0.22 =0.52*	0.24+0.24 =0.48	0.21+0.26 =0.47	0.20+0.28 =0.48	0.15+0.30 =0.45	0.12+0.33 =0.45	0.34	0.52

3. 考虑投入 7 万元资金于三种产品 A、B、C 的生产，利润与资金关系如下：

$$f_3(7) = \max_{0 \leqslant x_3 \leqslant 7} \{g_3(x_3) + f_2(7 - x_3)\}$$

其中 x_3 为投入到产品 C 的生产资金。利润表如表 1.2.4。

表 1.2.4

x_3	0	1	2	3	4	5	6	7
$g_3(x)$	0.00	0.18	0.22	0.26	0.28	0.30	0.34	0.36
$f_2(7-x)$	0.52	0.46	0.44	0.42	0.37	0.34	0.22	0.00
$g_3(x)+f_2(7-x)$	0.52	0.64	0.66	0.68	0.65	0.64	0.56	0.36

所以

$$f_3(7) = \max_{0 \leqslant x_3 \leqslant 7} \{g_3(x) + f_2(7 - x)\} = 0.68$$

即可获得最大利润 0.68 万元。从表 1.2.4 知 $x_3=3$。

$$f_2(7 - x_3) = f_2(4)$$

从表 1.2.3 可知 $f_2(4)=0.42$，而且 $x_2=1$，故 $x_1=3$。

例 5 有 n 块银币，已知其中有一块是伪币，它比正常的银币轻。现有一天平，试求通过天平找出其中假币。要求在最坏情况下用天平的次数最少。

n 块银币取出 $2k$ 块来。天平两边各 k 块。有两种情况，一是天平两边不平衡，则伪币

必在轻的 k 个中，若天平两边相平衡，则伪币必在余下的 $n-2k$ 个中。故问题导至求动态规划问题：

$$S(n)=\min\{\max_{1\leq k\leq \lfloor \frac{n}{2}\rfloor}\{S(k),S(n-2k)\}\}+1$$

$$S(0)=0,\ S(1)=0$$

$n=2$ 时，

$$\begin{aligned}S(2)&=\min\{\max\{S(1),S(0)\}\}+1\\&=\min\{\max\{0,0\}\}+1\\&=1\end{aligned}$$

这说明 2 块银币用一次天平就够了。$S(1)=0$。据题意它就是伪币，无需用天平。

$n=3$ 时

$$\begin{aligned}S(3)&=\min\{\max\{S(1),S(1)\}\}+1\\&=1\end{aligned}$$

也就是说 3 块银币也只要用一次天平。有两种情况，若天平两边各 1 块，但不相等，则伪币为轻的一端；若相等，则伪币为余下的一块。

$n=4$ 时，

$$\begin{aligned}S(4)&=\min\{\max\{S(1),S(2)\},\max\{S(2),S(0)\}\}+1\\&=\min\{\max\{0,1\},\max\{1,0\}\}+1\\&=2\end{aligned}$$

此时要用两次天平。也有两种情况，一是取 2 块，天平两边各一块，若不等，伪币一次找到；若相等，伪币在余下的两块中。考虑到最坏情况，故需要作两次比较。另一种取 4 块，天平两端各两块，必有一边轻的，伪币在轻的一边故任何情况都得用两次天平。

$n=5$ 时，

$$\begin{aligned}S(5)&=\min\{\max\{S(1),S(3)\},\max\{S(2),S(1)\}\}+1\\&=\min\{\max\{0,1\},\max\{1,0\}\}+1\\&=2\end{aligned}$$

也有两种情况，但都要用 2 次天平。

$n=6$ 时，

$$\begin{aligned}S(6)&=\min\{\max\{S(1),S(4)\},\max\{S(2),S(2)\},\max\{S(3),S(0)\}\}+1\\&=\min\{2,1^*,1^*\}+1=2\end{aligned}$$

* 表示最小项。可见 $n=6$ 时可采用两种策略：一是取 4 块，天平两端各两块，余下两块。这时仍用两次天平。也可以在天平两端各 3 块，这也要用两次天平。但不能用天平两边各一块。若不等，一次便找出伪币，但若出现相等，伪币在余下的 4 块中，最坏情况要用 3 次天平。

$n=7$ 时，

$$\begin{aligned}S(7)&=\min\{\max\{S(1),S(5)\},\max\{S(2),S(3)\},\max\{S(3),S(1)\}\}+1\\&=\min\{\max\{0,2\},\max\{1,1\},\max\{1,0\}\}+1\\&=\min\{2,1^*,1^*\}+1=2\end{aligned}$$

故也有两种策略。

$n=8$ 时，

$$S(8)=\min\{\max\{S(1),S(6)\},\max\{S(2),S(4)\},\max\{S(3),S(2)\},\max\{S(4),S(0)\}\}+1$$
$$=\min\{\max\{0,2\},\max\{1,2\},\max\{1,1\},\max\{2,0\}\}+1$$
$$=\min\{2,2,1^*,2\}+1=2$$

可见最佳策略应该是取 6 个，分在两边天平上，每边 3 个。若不等，则伪币必在轻的 3 个中；若相等，则伪币必在余下的两个中，无论哪一种都只要两次比较，即用两次天平。

所讨论的结果用图 1.2.1 表示。

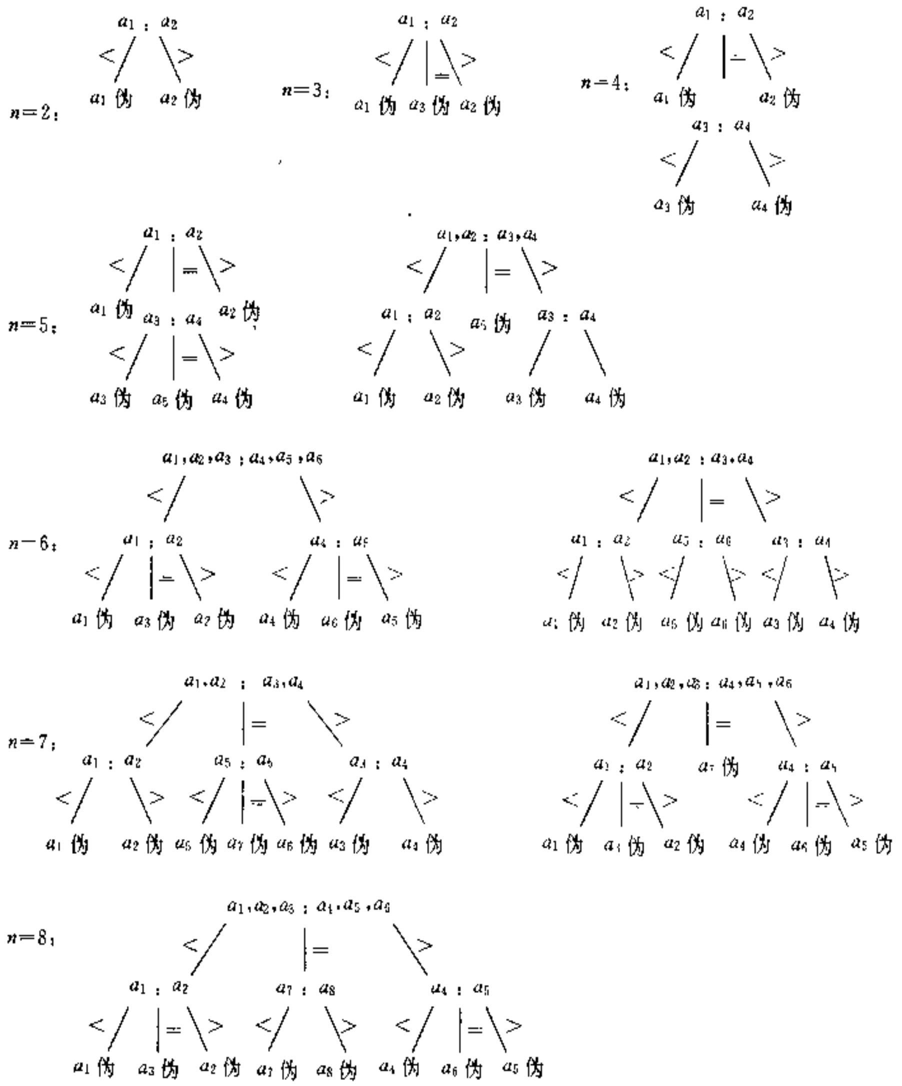


图 1.2.1

1.3 流动推销员(或旅行商)问题

1.3.1 算法及例题

已知一个由 n 个城市(或节点)组成的网络，这 n 个城市编号为 v_1，v_2、…，v_n，d_{ij}表示从 v_i 到 v_j 的距离(或时间、费用等)，一般 $d_{ij}\neq d_{ji}$，一个推销员要从 v_1 开始，访问每一城市一次且仅一次，最后返回 v_1。这个推销员应如何选择线路，才能使行程最短？通常称这个问题为流动推销员问题。有时也称旅行商问题。

我们知道这样的线路和 v_1，v_2，…，v_n 绕一圆圈排列一一对应，故有$(n-1)!$种不同方案，若使用穷举法，需作$(n-1)(n-1)!$次加法和$(n-1)!-1$次比较，当 n 很大时，这是不能接受的运算量，故流动推销员问题是典型的难解问题。下面我们介绍如何将流动推销员问题化为多段判决问题，用动态规划办法求解，并对其时间及空间复杂性作出估计。

令 $f(v_i;V)$表示从 v_i 点出发，遍历 V 中的点一次且仅一次，最后返回到 v_1 的最短距离，其中 V 是某些顶点构成的集合，且 $v_1\notin V$。这样有以下多段判决递推公式：

$$f(v_i;V)=\min_{v_j\in V}\{d_{ij}+f(v_j;V\setminus\{v_j\})\}$$

$V\setminus\{v_j\}$表示从 V 中除去 v_j。

设图 $G=(V,E)$，$V=\{v_1,v_2,\cdots,v_n\}$，关于图 G 的流动推销员问题即求

$$f(v_1;\{v_2,v_3,\cdots,v_n\})$$

下面看一个具体的例子。

距离矩阵：

$$D=\begin{array}{c}v_1\\v_2\\v_3\\v_4\\v_5\\ \\ \end{array}\begin{array}{c}\begin{pmatrix}0&2&1&3&1\\1&0&4&4&2\\5&4&0&2&2\\5&2&2&0&3\\4&2&4&2&0\end{pmatrix}\\ \begin{array}{ccccc}v_1&v_2&v_3&v_4&v_5\end{array}\end{array}=(d_{ij})_{5\times5}$$

$f(v_2;\varnothing)=1=d_{21}$,

$f(v_3;\varnothing)=5=d_{31}$,

$f(v_4;\varnothing)=5=d_{41}$,

$f(v_5;\varnothing)=4=d_{51}$,

$f(v_2;v_3)=d_{23}+f(v_3;\varnothing)=4+5=9$,

$f(v_2;v_4)=d_{24}+f(v_4;\varnothing)=4+5=9$,

$f(v_2;v_5)=d_{25}+f(v_5;\varnothing)=2+4=6$,

$f(v_3;v_2)=d_{32}+f(v_2;\varnothing)=4+1=5$,

$f(v_3;v_4)=d_{34}+f(v_4;\varnothing)=2+5=7$,

$f(v_3;v_5)=d_{35}+f(v_5;\varnothing)=2+4=6$,

$f(v_4;v_2)=d_{42}+f(v_2;\varnothing)=2+1=3$,

$$f(v_4;v_3) = d_{43} + f(v_3;\varnothing) = 2 + 5 = 7,$$
$$f(v_4;v_5) = d_{45} + f(v_5;\varnothing) = 3 + 4 = 7,$$
$$f(v_5;v_2) = d_{52} + f(v_2;\varnothing) = 2 + 1 = 3,$$
$$f(v_5;v_3) = d_{53} + f(v_3;\varnothing) = 4 + 5 = 9,$$
$$f(v_5;v_4) = d_{54} + f(v_4;\varnothing) = 2 + 5 = 7,$$
$$f(v_2;v_3,v_4) = \min\{d_{23} + f(v_3;v_4), d_{24} + f(v_4;v_3)\}$$
$$= \min\{\underline{4 + 7}, \underline{4 + 7}\} = 11$$

上式中的____线表示最小值的状态。

$$f(v_2;v_3,v_5) = \min\{d_{23} + f(v_3;v_5), d_{25} + f(v_5;v_3)\}$$
$$= \min\{\underline{4 + 6}, 2 + 9\} = 10$$
$$f(v_2;v_4,v_5) = \min\{d_{24} + f(v_4;v_5), d_{25} + f(v_5;v_4)\}$$
$$= \min\{4 + 7, \underline{2 + 7}\} = 9$$
$$f(v_3;v_2,v_4) = \min\{d_{32} + f(v_2;v_4), d_{34} + f(v_4;v_2)\}$$
$$= \min\{4 + 9, \underline{2 + 3}\} = 5$$
$$f(v_3;v_2,v_5) = \min\{d_{32} + f(v_2;v_5), d_{35} + f(v_5;v_2)\}$$
$$= \min\{4 + 6, \underline{2 + 3}\} = 5$$
$$f(v_3;v_4,v_5) = \min\{d_{34} + f(v_4;v_5), d_{35} + f(v_5;v_4)\}$$
$$= \min\{\underline{2 + 7}, \underline{2 + 7}\} = 9$$
$$f(v_4;v_2,v_3) = \min\{d_{42} + f(v_2;v_3), d_{43} + f(v_3;v_2)\}$$
$$= \min\{2 + 9, \underline{2 + 5}\} = 7$$
$$f(v_4;v_2,v_5) = \min\{d_{42} + f(v_2;v_5), d_{45} + f(v_5;v_2)\}$$
$$= \min\{2 + 6, \underline{3 + 3}\} = \min\{8, \underline{6}\} = 6$$
$$f(v_4;v_3,v_5) = \min\{d_{43} + f(v_3;v_5), d_{45} + f(v_5;v_3)\}$$
$$= \min\{\underline{2 + 6}, 3 + 9\} = 8$$
$$f(v_5;v_2,v_3) = \min\{d_{52} + f(v_2;v_3), d_{53} + f(v_3;v_2)\}$$
$$= \min\{2 + 9, \underline{4 + 5}\} = 9$$
$$f(v_5;v_2,v_4) = \min\{d_{52} + f(v_2;v_4), d_{54} + f(v_4;v_2)\}$$
$$= \min\{2 + 9, \underline{2 + 3}\} = 5$$
$$f(v_5;v_3,v_4) = \min\{d_{53} + f(v_3;v_4), d_{54} + f(v_4;v_3)\}$$
$$= \min\{4 + 7, \underline{2 + 7}\} = 9$$
$$f(v_2;v_3,v_4,v_5) = \min\{d_{23} + f(v_3;v_4,v_5), d_{24} + f(v_4;v_3,v_5), d_{25} + f(v_5;v_3,v_4)\}$$
$$= \min\{4 + 9, 4 + 8, \underline{2 + 9}\} = 11$$
$$f(v_3;v_2,v_4,v_5) = \min\{d_{32} + f(v_2;v_4,v_5), d_{34} + f(v_4;v_2,v_5), d_{35} + f(v_5;v_2,v_4)\}$$
$$= \min\{4 + 9, 2 + 6, \underline{2 + 5}\} = \min\{13, 8, \underline{7}\} = 7$$
$$f(v_4;v_2,v_3,v_5) = \min\{d_{42} + f(v_2;v_3,v_5), d_{43} + f(v_3;v_2,v_5), d_{45} + f(v_5;v_2,v_3)\}$$
$$= \min\{2 + 10, \underline{2 + 5}, 3 + 9\} = 7$$
$$f(v_5;v_2,v_3,v_4) = \min\{d_{52} + f(v_2;v_3,v_4), d_{53} + f(v_3;v_2,v_4), d_{54} + f(v_4;v_2,v_3)\}$$
$$= \min\{2 + 11, \underline{4 + 5}, \underline{2 + 7}\} = 9$$

$$
\begin{aligned}
f(v_1;v_2,v_3,v_4,v_5) &= \min\{d_{12}+f(v_2;v_3,v_4,v_5),d_{13}+f(v_3;v_2,v_4,v_5),d_{14} \\
&\qquad +f(v_4;v_2,v_3,v_5),d_{15}+f(v_5;v_2,v_3,v_4)\} \\
&= \min\{2+11,\underline{1+7},3+7,4+9\} \\
&= \min\{13,\underline{8},10,13\}=8
\end{aligned}
$$

故 v_1 出发先到 v_3，又从 $f(v_3;v_2,v_4,v_5)=7$ 可知 v_3 到 v_5，从而依次回溯过程知最短线路是：

$$v_1 \to v_3 \to v_5 \to v_4 \to v_2 \to v_1$$

除了穷举法外，动态规划方法给出了求流动推销员问题的方法。

1.3.2 复杂性估计

现在先把 $n=5$ 的前例作具体分析，由此可推到一般的情况。

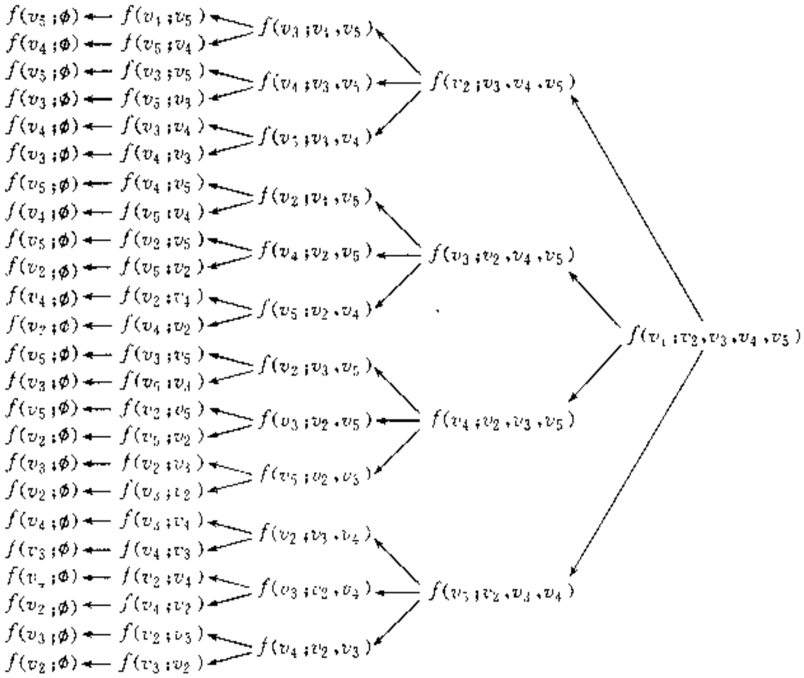


图 1.3.1

图 1.3.1 表示动态规划解法的一种计算关系或计算顺序。比如要计算 $f(v_2;v_3,v_4)$ 必须先计算 $f(v_3;v_4)$ 和 $f(v_4;v_3)$；要计算 $f(v_3,v_4)$ 必先计算 $f(v_4;\varnothing)$ 余此类推。计算过程自左向右，自下而上。

注意图 1.3.1 树状结构中元素并非完全不同，现只考虑其不同的元素。

不失一般情况，现在假定 $v=\{v_0,v_1,v_2,\cdots,v_n\}$，$\overline{V}=V\backslash\{v_0\}=\{v_1,v_2,\cdots,v_n\}$。从 v_0 出发遍历 $\overline{V}$ 中的点最后返回 v_0，求最短路径。则表 1.3.1 中

第 1 列：即右一列，显然只有一个 $f(v_0;\overline{V})$；

第 2 列：$f(v_{i_1};\overline{V}\backslash\{v_{i_1}\})$。由于 $v_{i_1}\in\overline{V}$，$\overline{V}\backslash\{v_{i_1}\}$ 的个数为 $n-1$，故第 2 列有 $nC(n-1,n-1)=n$ 个不同的 $f(v_i;\overline{V}\backslash\{v_i\})$；

第 3 列：$f(v_{i_2};\overline{V}\setminus\{v_{i_1},v_{i_2}\})$，其中 $v_{i_2}\in\overline{V}$，$\overline{V}\setminus\{v_{i_1},v_{i_2}\}$中的个数为 $n-2$，故第 3 列有 $nC(n-1,n-2)$个不同的 $f(v_i;\overline{V}\setminus\{v_i,v_j\})$。

⋮

一般第 k 列有：

$$nC(n-1,n-k+1)=nC(n-1,k-2)$$

个不同的元素 $f(v_{i_1};\overline{V}\setminus\{v_{i_1},v_{i_2},\cdots,v_{i_{k-1}}\})$。

故若利用一个存贮单元存贮一个 $f(v_i;v_i)$，则所需存贮单元数为：

$$S=1+\sum_{k=2}^{n+1}nC(n-1,k-2)=1+n\sum_{k=2}^{n+1}C(n-1,k-2)$$

$$=1+n\sum_{k=0}^{n-1}C(n-1,k)=1+n2^{n-1}=n2^{n-1}-1$$

下面对时间复杂性作出估计：

第 1 列为顶点 $f(v_0;\overline{V})$的计算要作 n 次加法和比较。

第 2 列：不同的 $f(v_{i_1};\overline{V}\setminus\{v_{i_1}\})$的个数为 $nC(n-1,n-1)$，每个要作 $n-1$ 次加法和比较运算。

第 3 列：不同的 $f(v_{i_2};\overline{V}\setminus\{v_{i_1},v_{i_2}\})$的个数为 $nC(n-1,n-2)$，每个要作 $n-2$ 次加法和比较运算。

⋮

一般第 k 列：不同的 $f(v_{i_{k-1}};\overline{V}\setminus\{v_{i_1},v_{i_2},\cdots,v_{i_{k-1}}\})$的数目是 $nC(n-1,k-2)$个，每个要作 $n-k+1$ 次加法和比较运算。$k=2,\cdots,n$，故加法和比较总数为

$$T=n+\sum_{k=2}^{n}(n-k-1)C(n-1,k-2)=n+\sum_{l=2}^{n}(n-k-1)C(n-1,n-k+1)$$

$$=n+n\{(n-1)C(n-1,n-1)+(n-2)C(n-1,n-2)+\cdots+C(n-1,1)\}$$

由于 $$(1+x)^{n-1}=1+C(n-1,1)x+C(n-1,2)x^2+\cdots+x^{n-1}$$

所以 $$(n-1)(1+x)^{n-2}=C(n-1,1)+2C(n-1,2)x+\cdots+(n-1)x^{n-2}$$

$x=1$ 代入等式两端得

$$C(n-1,1)+2C(n-1,2)+\cdots+(n-1)C(n-1,n-1)=(n-1)2^{n-2}$$

故得

$$T=n+n(n-1)2^{n-2}$$

从上面的分析可知，用动态规划求解流动推销员问题，它的时间复杂性虽然较穷举法有所下降，但时间复杂性和空间复杂性都仍然保持为规模 n 的指数函数，所以动态规划虽然提供了一种解法，但也不是一种可行的算法。

1.4 矩阵链乘问题

先以三个矩阵 A,B,C 的乘积为例。设 $A=(a_{ij})_{m\times n}$，$B=(b_{ij})_{n\times l}$，$C=(c_{ij})_{l\times r}$是给定的三个矩阵，求积 ABC。

由矩阵乘法的结合律知

$$ABC = (AB)C = A(BC)$$

即求 ABC 可以有两种步骤，一是 $(AB)C$，另一是 $A(BC)$，但所需乘法次数不同。计算 $(A_{m\times n}B_{n\times l})C_{l\times r}$ 所需的乘法次数为：

$$mnl + mlr$$

为方便起见，这里 $A_{m\times n}$ 用下标 $m\times n$ 表达矩阵 A 的阶。而计算 $A_{m\times n}(B_{n\times l}C_{l\times r})$ 所需的乘法次数为：

$$mnr + nlr$$

如当 $m=10, n=100, l=5, r=50$，则

$$mnl + mlr = 5000 + 2500 = 7500$$

$$mnr + nlr = 50000 + 25000 = 75000$$

可见 $(AB)C$ 和 $A(BC)$ 不同的乘积顺序，结果相同，而所作乘法的次数相差很悬殊。

所谓矩阵链乘问题，就是找出计算 $A_1A_2\cdots A_n$ 的运算量最小的乘积顺序。假定其中矩阵 A_i 是 $r_i\times r_{i+1}$ 阶的矩阵，即 $A_i=(a_{jk}^{(i)})_{r_i\times r_{i+1}}$。

可以将矩阵链乘问题化为多段判决问题。

设最佳的乘积方案形式为 $(A_1A_2\cdots A_i)(A_{i+1}A_{i+2}\cdots A_n)$，则 $A_1A_2\cdots A_i$ 和 $A_{i+1}A_{i+2}\cdots A_n$ 的乘积顺序也应该是最佳的。设 m_{ij} 为 $A_iA_{i+1}\cdots A_j$ 所需的最小乘法次数，

$$A_iA_{i+1}\cdots A_j = (A_iA_{i+1}\cdots A_k)(A_{k+1}\cdots A_j),$$

$$m_{ij} = \min_{i\le k<j}\{m_{ik} + m_{(k+1),j} + r_ir_{k+1}r_{j+1}\}$$

$$m_{ii} = 0, \qquad i,j = 1,2,\cdots,n, i < j$$

其中各式的意义读者自己思考。

如何找出矩阵链乘 $A_1A_2\cdots A_n$ 的最佳次序，我们以 $A_1=(a_{ij}^{(1)})_{35\times 40}, A_2=(a_{ij}^{(2)})_{40\times 20}, A_3=(a_{ij}^{(3)})_{20\times 10}, A_4=(a_{ij}^{(4)})_{10\times 15}$ 为例，使用多段判决求解如下：

$m_{12} = 35\times 40\times 20 = 28000$

$m_{23} = 40\times 20\times 10 = 8000$

$m_{34} = 20\times 10\times 15 = 3000$

$m_{13} = \min\{m_{12} + 35\times 20\times 10, m_{23} + 35\times 40\times 10\}$

$\quad = \min\{28000 + 7000, 8000 + 14000^*\} = 22000$

$m_{24} = \min\{m_{23} + 40\times 10\times 15, m_{34} + 40\times 20\times 15\}$

$\quad = \min\{8000 + 6000^*, 3000 + 12000\} = 14000$

$m_{14} = \min\{m_{24} + 35\times 40\times 15, m_{12} + m_{34} + 35\times 20\times 15, m_{13} + 35\times 10\times 15\}$

$\quad = \min\{14000 + 21000, 28000 + 3000 + 10500, 22000 + 5250\}$

$\quad = \min\{35000, 41500, 27250^*\} = 27250$

* 表示最小值的项，$m_{14}=27250$ 应是 $(A_1A_2A_3)A_4$，从 $m_{13}=22000$ 知最佳步骤应是 $A_1(A_2A_3)$，回溯上面的求解过程知：最佳方案为：

$$((A_1(A_2A_3))A_4)$$

上面的计算顺序，可以形象地用表 1.4.1 表示，顺序是沿着对角线逐层向上。若将表 1.4.1 画成图 1.4.2 的形状，即主对角线置于水平位置，则计算顺序自左往右，自下而上，

直到最顶层。

求最优的 m_{1n} 不是我们的主要目的，目的是找到最佳的矩阵乘法次序。实际上，上述求 m_{1n} 过程也给出了最佳次序，具体细节留给读者思考。

从上面通过简单观察可知，求 n 个矩阵链乘积的最佳顺序的计算等价于构造图 1.4.2，其时间复杂性为 $O(n^3)$，另外空间复杂性为 $O(n^2)$，用以存储 m_{ij}。

表 1.4.1

	$j=1$	2	3	4
$i=1$	0	m_{12}	m_{13}	m_{14}
2		0	m_{23}	m_{24}
3			0	m_{34}
4				0

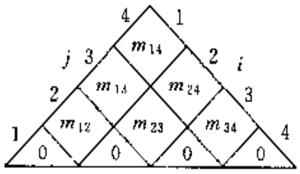


图 1.4.2

若采用穷举法，设 $P(n)$ 为 n 个矩阵乘积可能的运算顺序的数目，则有

$$P(n) = P(1)P(n-1) + P(2)P(n-2) + \cdots + P(n-1)P(1)$$

$$P(2) = 1$$

它是著名的 Catalan 数，

$$P(n) = \frac{1}{n+1}\binom{2n}{n}$$

从而可知：穷率法的时间复杂性为 n 的指数函数。

1.5 最长公共子序列

对给定的两个序列 $A=\{a_1,a_2,\cdots,a_m\}$ 和 $B=\{b_1,b_2,\cdots,b_n\}$，若存在单调增的整数序列：$i_1<i_2<\cdots<i_l$ 和 $j_1<j_2<\cdots<j_l$ 使得子序列 $\{a_{i_1},a_{i_2},\cdots,a_{i_l}\}$ 和 $\{b_{j_1},b_{j_2},\cdots,b_{j_l}\}$ 有 $a_{i_k}=b_{j_k}=c_k$，$k=1,2,\cdots,l$，记这个子序列为 $C=\{c_1,c_2,\cdots,c_l\}$，则称序列 C 是 A 和 B 的公共子序列。使 l 达到最大的公共子序列，称为序列 A 和 B 的最长公共子序列。用符号 $LCS(A,B)$ 表示。

例如：$A=\{a,b,c,b,d,a,c,b\}$

和 $B=\{b,d,c,a,b\}$

$C_1=\{b,a,b\}$ 是 A 和 B 的公共子序列，但不是最长公共子序列。$C_2=\{b,d,a,b\}$，$C_3=\{b,c,a,b\}$ 都是它们的公共子序列，长度都为 4，但找不出比 4 更长的公共子序列，所以 C_2 和 C_3 便是最长的公共子序列。最长公共子序列不唯一。

对序列 $A=\{a_1,a_2,\cdots,a_m\}$，定义 $A_0=\varnothing$，$A_1=\{a_1\}$，$A_2=\{a_1,a_2\}$，$\cdots$，$A_m=\{a_1,a_2,\cdots,a_m\}$。同样对序列 $B=\{b_1,b_2,\cdots,b_n\}$ 定义 $B_0=\varnothing$，$B_1=\{b_1\}$，$B_2=\{b_1,b_2\}$，$\cdots$，$B_n=\{b_1,b_2,\cdots,b_n\}$，则最长公共子序列问题可化为如下多段判决问题：

记 $A=A_m$，$B=B_n$

$$LCS(A_m,B_n)=\begin{cases}LCS(A_{m-1},B_{n-1})\cup\{a\}, 若\ a_m=b_n=a\\ \max\{LCS(A_{m-1},B_n),LCS(A_m,B_{n-1})\}, 其他\end{cases}$$

一般有

$$LCS(A_i,B_j)=\begin{cases}\varnothing, & i=0\ 或\ j=0\\ LCS(A_{i-1},B_{j-1})\cup\{a\} & 若\ a_i=b_j=a\\ \max\{LCS(A_{i-1},B_j),LCS(A_i,B_{j-1})\}, & 其他\end{cases}$$

若采用穷举法求得 $LCS(A,B)$，其时间复杂性是 m 或 n 的指数函数。下面介绍求 A 和 B 的最长子序列的动态规划算法。为方便起见，用 $l(i,j)$ 表示 $LCS(A_i,B_j)$ 的长度。

(1) 初始化操作，$l(i,0)\leftarrow 0, i=1,2,\cdots,m; l(0,j)\leftarrow 0, j=1,2,\cdots,n;\ i\leftarrow 1$。

(2) 若 $i\leqslant m$，则作【$j\leftarrow 1$，转(3)】。否则，转(7)。

(3) 若 $j\leqslant n$，转(4)。

否则，作【$i\leftarrow i+1$，转(2)】。

(4) 若 $a_i=b_j$，则作【$l(i,j)\leftarrow l(i-1,j-1)+1, j\leftarrow j+1$，转(3)】。

否则，转(5)。

(5) 若 $l(i-1,j)\geqslant l(i,j-1)$，则作【$l(i,j)\leftarrow l(i-1,j), j\leftarrow j+1$，转(3)】。

否则，转(6)。

(6) $l(i,j)\leftarrow l(i,j-1), j\leftarrow j+1$，转(3)。

(7) 输出 $l(m,n)$，停止。

算法中 $l(m,n)$ 给出了最长公共子序列的长度，适当修改算法，可以求得最长公共子序列本身。第 4 步处理 $a_i=b_j$ 的情况。第 5,6 步是对 $a_i\neq b_j$ 所采取的措施：$l(i,j)=\max\{l(i-1,j),l(i,j-1)\}$，其正确性可由多段判决公式看出。

现以 $A=\{d,b,c,b,a,d,b\}$ 和 $B=\{b,a,c,d,b,d\}$ 为例，$l(i,j)$ 计算如表 1.5.1 所示。

表 1.5.1

i \ j	0	1	2	3	4	5	6	
0	0	0	0	0	0	0	0	
1	0	0	0	0	1	1	1	d
2	0	1	1	1	1	2	2	b
3	0	1	1	2	2	2	2	c
4	0	1	1	2	2	3	3	b
5	0	1	2	2	2	3	3	a
6	0	1	2	3	3	3	4	d
7	0	1	2	2	3	4	4	b
		b	a	c	d	b	d	

计算按从上而下，从左向右的次序进行。

从算法中可以看出，对 $l(i,j)$ 的修改无非以下三处：

(1) 第 4 步 $a_i=a_j$ 时 $l(i,j)\leftarrow l(i-1,j-1)+1$。

(2) $a_i\neq b_j$ 但 $l(i-1,j)\geqslant l(i,j-1)$ 时，$l(i,j)\leftarrow l(i-1,j)$。

(3) $a_i\neq b_j$ 但 $l(i-1,j)<l(i,j-1)$ 时，$l(i,j)\leftarrow l(i,j-1)$。

这对于回溯找最大公共子序列时会有帮助的。读者还可以通过本例从 $l(i,j)$ 的变化找最长公共子序列。

1.6 图的任意两点间的最短距离

已知图 $G=(V,E)$，$V=\{v_1,v_2,\cdots,v_n\}$ 及距离矩阵

$$D=(d_{ij})_{n\times n}$$

$$d_{ij}=\begin{cases}(v_i,v_j)\text{ 的长度} & ,\text{若}(v_i,v_j)\in E\\ 0 & ,v_i=v_j\\ \infty & ,\text{其他}\end{cases}$$

$$i,j=1,2,\cdots,n$$

求图 G 的任意两点间的最短路径。

设矩阵 $A=(a_{ij})_{n\times n}$，$B=(b_{ij})_{n\times n}$，定义矩阵运算如下：

$$C\triangleq A*B=(c_{ij})_{n\times n}$$

其中

$$c_{ij}=\min_k\{a_{ik}+b_{kj}\},\ i,j=1,2,\cdots,n$$

令 $D^{(1)}=D$，$D^{(k+1)}=D^{(k)}*D^{(1)}$。

显然 $D^{(2)}=(d_{ij}^{(2)})$，$d_{ij}^{(2)}=\min\limits_k\{d_{ik}+d_{kj}\}$，表示从 v_i 出发经过某一中间点到达 v_j 点的最短距离。同样 $d_{ij}^{(3)}$ 表示从 v_i 经过两个中间点到 v_j 的最短距离。

定义

$$A\vee B\triangleq(\min(a_{ij},b_{ij}))_{n\times n}$$

令
$$D^*=D^{(1)}\vee D^{(2)}\vee\cdots\vee D^{(n)}=(d_{ij}^*)_{n\times n}$$

则 d_{ij}^* 表示从 v_i 点到 v_j 点的最短距离。

由于 $D^{(k)}$ 有 n^2 个元素，每个元素要作 n 次乘法运算，依以上办法可知求 D^* 的时间复杂性为 $\Theta(n^4)$。下面介绍动态规划的解法。

设
$$\overline{D}^{(k)}=(\overline{d}_{ij}^{(k)})$$

$\overline{d}_{ij}^{(k)}$ 为从 v_i 出发中途经过以 $(v_1,v_2,\cdots,v_k)$ 为中间点到 v_j 的最短距离。则可化为多段判决问题如下：

$$\overline{d}_{ij}^{(k)}=\min\{\overline{d}_{ij}^{(k-1)},\overline{d}_{ik}^{(k-1)}+\overline{d}_{kj}^{(k-1)}\}\qquad i,j=1,2,\cdots,n$$

其中 $\overline{d}_{ij}^{(0)}=d_{ij}$ 并且 $\overline{d}_{ij}^{(n)}$ 即为 i,j 之间的最短距离。

算法如下：

(1) $\overline{D}\leftarrow D$

(2) k 从 1 到 n 作

【i 从 1 到 n 作

【j 从 1 到 n 作

【$\bar{d}_{ij} \leftarrow \min\{\bar{d}_{ij}, d_{ik}+\bar{d}_{kj}\}$】】】

算法有 i,j,k 三重循环，故其时间复杂性为 $\Theta(n^3)$，而不是 $\Theta(n^4)$。

这个算法实际上是十分直观的，举例如下

$$D = \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ \end{array} \begin{bmatrix} \infty & 1 & 2 & \infty & \infty & \infty \\ \infty & \infty & 1 & 3 & \infty & 7 \\ \infty & \infty & \infty & 1 & 2 & \infty \\ \infty & \infty & \infty & \infty & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & 6 \\ \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$
$$\quad\quad v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \quad v_6$$

$k=2$ 时 $\qquad d_{ij} = \min\{d_{ij}, d_{i2}+d_{2j}\}$

显然 $(v_1,v_4)\notin E$，但 $v_1 \to v_2 \to v_4$，故 $d_{14}^{(2)}=4$

$$D^{(2)} = \begin{bmatrix} \infty & 1 & 2 & 4^* & \infty & \infty \\ \infty & \infty & 1 & 3 & \infty & 7 \\ \infty & \infty & \infty & 1 & 2 & \infty \\ \infty & \infty & \infty & \infty & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & 6 \\ \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

$k=3$ 时，$(v_1,v_5)\notin E$，但 $(v_1,v_3)\in E$，$(v_3,v_5)\in E$，且 $d_{15}^{(3)}=4$，类似有 $d_{14}^{(2)}=4$，但 $d_{13}=2$，$d_{34}=1$，故 $d_{14}^{(3)}=3$。类似有 $d_{24}^{(3)}=\{d_{24}^{(2)}, d_{23}^{(2)}+d_{34}\}=\min\{3,2\}=2$，$d_{25}^{(3)}=\min\{d_{25}^{(2)}, d_{23}^{(2)}+d_{35}^{(2)}\}=3$

$$\therefore \qquad D^{(3)} = \begin{bmatrix} \infty & 1 & 2 & 3^* & 4^* & \infty \\ \infty & \infty & 1 & 2^* & 3^* & \infty \\ \infty & \infty & \infty & 1 & 2 & 7 \\ \infty & \infty & \infty & \infty & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & 6 \\ \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

类似的步骤可得

$$D^{(4)} = \begin{bmatrix} \infty & 1 & 2 & 3 & 4 & 6^* \\ \infty & \infty & 1 & 2^* & 3 & 5^* \\ \infty & \infty & \infty & 1 & 2 & 4^* \\ \infty & \infty & \infty & \infty & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & 6 \\ \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix},$$

而且

$$D^{(4)} = D^{(5)} = D^{(6)}$$

1.7 整数规划问题

从前面的讨论可知动态规划实际上是一种技巧性极强的算法，它的理论基础最佳原理仅是十分一般的原则，如何应用它并不像原理本身那么容易简单。本节讨论它在整数规划上的应用。整数规划是规划论的一个侧面，它的求解难度很大，后面还将专题讨论它。

所谓整数规划，例如求下列问题的最优解。

$$\max z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

满足约束条件：

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leqslant b$$

$$x_i \text{ 为非负整数}, i=1,2,\cdots,n$$

令

$$f_n(b) = \max_{0\leqslant x_n\leqslant \left\lfloor \frac{b}{a_n}\right\rfloor} \{f_{n-1}(b - a_nx_n) + c_nx_n\}$$

一般有

$$f_i(x) = \max_{0\leqslant x_i\leqslant \left\lfloor \frac{x}{a_i}\right\rfloor} \{f_{i-1}(x - a_ix_i) + c_ix_i\} \quad i = 2,3,\cdots,n$$

计算从 $f_0(x)\equiv 0$ 开始。

举例说明步骤如下：

例 1

$$\max z=110x_1+160x_2+260x_3+210x_4$$

$$2x_1+3x_2+5x_3+4x_4\leqslant 20$$

$$x_i\geqslant 0 \qquad \text{整数}$$

令

$$f_1(k)= \max_{0\leqslant x_1\leqslant \left\lfloor \frac{k}{2}\right\rfloor} \{110x_1\}$$

故有

$$f_1(k) = \begin{cases} 55k & ,k \text{ 为偶数} \\ 55(k-1) & ,k \text{ 为奇数} \end{cases}$$

$$f_2(k) = \max_{0\leqslant x_2\leqslant \left\lfloor \frac{k}{3}\right\rfloor} \{f_1(k - 3x_2) + 160x_2\}$$

$k-3x_2$ 为偶数时

$$f_2(k) = \max_{0\leqslant x_2\leqslant \left\lfloor \frac{k}{3}\right\rfloor} \{160x_2 + 55(k - 3x_2)\}$$

$$= \max_{0\leqslant x_2\leqslant \left\lfloor \frac{k}{3}\right\rfloor} \{55k - 5x_2\} = 55k$$

$$\therefore \qquad x_2 = 0$$

$k-3x_2$ 为奇数时

$$f_2(k) = \max_{0\leqslant x_2\leqslant \left\lfloor \frac{k}{3}\right\rfloor} \{160x_2 + 55(k - 3x - 1)\}$$

$$= \max_{0\leqslant x_2\leqslant \left\lfloor \frac{k}{3}\right\rfloor} \{55k - 55 - 5x_2\}$$

$$=55k - 55$$

$$\therefore \qquad x_2 = 0$$

所以
$$f_2(k) = \begin{cases} 55k & ,k\text{ 为偶数} \\ 55k - 55 & ,k\text{ 为奇数} \end{cases}$$

$$f_3(k) = \max_{0 \leq x_3 \leq \left\lfloor \frac{k}{5} \right\rfloor} \{260x_3 + f_2(k - 5x_3)\}$$

$k-5x_3$ 为偶数时

$$\begin{aligned} f_3(k) &= \max_{0 \leq x_3 \leq \left\lfloor \frac{k}{5} \right\rfloor} \{260x_3 + 55(k - 5x_3)\} \\ &= \max_{0 \leq x_3 \leq \left\lfloor \frac{k}{5} \right\rfloor} \{260x_3 + 55k - 275x_3\} \\ &= \max_{0 \leq x_3 \leq \left\lfloor \frac{k}{5} \right\rfloor} \{55k - 15x_3\} \\ &= 55k \end{aligned}$$

$$x_3 = 0$$

$k-5x_3$ 为奇数时

$$\begin{aligned} f_3(k) &= \max_{0 \leq x_3 \leq \left\lfloor \frac{k}{5} \right\rfloor} \{260x_3 + 55(k - 5x_3) - 55\} \\ &= \max_{0 \leq x_3 \leq \left\lfloor \frac{k}{5} \right\rfloor} \{55k - 15x_3 - 55\} = 55k - 55 \end{aligned}$$

$$x_3 = 0$$

所以
$$f_3(k) = \begin{cases} 55k & ,k\text{ 为偶数} \\ 55k - 55 & ,k\text{ 为奇数} \end{cases}$$

$$\begin{aligned} f_4(20) &= \max_{0 \leq x_4 \leq 5} \{210x_4 + f_3(20 - 4x_4)\} \\ &= \max_{0 \leq x_4 \leq 5} \{210x_4 + 55(20 - 4x_4)\} \end{aligned}$$

由于 $20-4x_4$ 为偶数

故有

$$\begin{aligned} f_4(20) &= \max_{0 \leq x_4 \leq 5} \{1100 - 10x_4\} \\ &= 1100 \end{aligned}$$

$$x_4 = 0$$

故最优解为：

$$x_1 = 10,\ x_2 = x_3 = x_4 = 0,\ z = 1100$$

若将 $z_1(x)$，$z_2(x)$，$z_3(x)$ 分别用图 1.7.1 表示，可见 $z_1(x)$，$z_2(x)$，$z_3(x)$ 都是阶跃式的函数，而且 $z_i(x)$ 的阶跃点部分地包含了 $z_{i-1}(x)$ 的阶跃点，$i=1,2,3$。不仅本例如此，一般情况也有这样结果。函数 $z_i(x)$ 可由它的阶跃点完全确定，设阶跃点为 (X_j, Z_j)，其中 $Z_j = z_i(X_j)$。

令　属于 $J_i(x)$ 的阶跃点的全体为 J_i

$$H_i \triangleq \{(X, Z) \mid (X - a_i, Z - c_i) \in J_{i-1}\}$$

J_i 可由 H_i 和 J_{i-1} 归并而成。

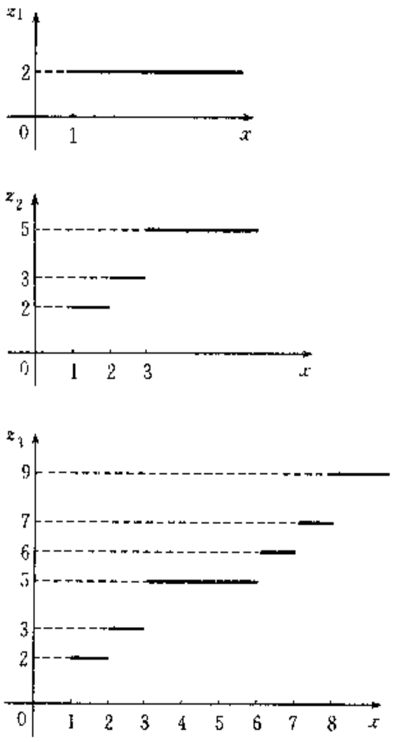


图 1.7.1

依本例

$$J_0 = \{(0,0)\}$$

$$H_1 = \{(1,2)\}$$

$$\therefore \quad J_1 = \{(0,0),(1,2)\}$$

$$H_2 = \{(2,3),(3,5)\}$$

$$\therefore \quad J_2 = \{(0,0),(1,2),(2,3),(3,5)\}$$

$$H_3 = \{(5,4),(6,6),(7,7),(8,9)\}$$

$$\therefore \quad J_3 = \{(0,0),(1,2),(2,3),(3,5),(6,6),(7,7),(8,9)\}$$

H_3 中的(5,4)没有进入 J_3,这是因为有(3,5)。

计算 $J_1,J_2,\cdots,J_n$ 需要的时间为 $O(2^{n-1})$。

若所有的 c_i 都是整数,所有 a_i 也都是整数,则属于 J_i 的阶跃点(X,Z)的数目不超过

$$1 + \min\left\{b, \sum_{j=1}^{i} c_j\right\}$$

$\sum_{i=1}^{n} |J_i|$ 的估计为 $O\left(\min\left\{2^n, nb, n\sum_{j=1}^{n} c_j\right\}\right)$

例 2 设有一旅客旅途可携带 10 重量单位的物品。有 3 种物品每件分别重 2,3,4 重量单位;每件价 11,16,20($\times 10^3$ 元)。试问应如何计划使所带的物品价值最大。

设 3 种物品携带的数量分别为 x_1,x_2,x_3 单位,则有

$$\max z = 11x_1 + 16x_2 + 20x_3$$

$$2x_1 + 3x_2 + 4x_3 \leqslant 10$$

$$x_i \text{ 为非负整数}, i=1,2,3$$

这是整数规划问题。但本问题也可以利用动态规划方法化为多段判决;步骤如下:

$$f_1(y_1) = \max_{0 \leqslant x_1 \leqslant \left\lfloor \frac{y_1}{2} \right\rfloor} \{11x_1\}$$

$$= 11\left\lfloor \frac{y_1}{2} \right\rfloor$$

表 1.7.2

y_1	0	1	2	3	4	5	6	7	8	9	10
$f_1(y_1)$	0	0	11	11	22	22	33	33	44	44	55
x_1^*	0	0	1	1	2	2	3	3	4	4	5

其中 x_1^* 为使 $f_1(y)$取极大值的 x_1 值。由

$$f_2(y_2) = \max_{0 \leqslant x_2 \leqslant \left\lfloor \frac{y_2}{3} \right\rfloor} \{16x_2 + f_1(y_2 - 3x_2)\}$$

及表 1.7.2 可知:

$y_2=0$ 时, $f_2(0)=0$, $x_1^*=x_2^*=0$,

$y_2=1$ 时, $f_2(1)=f_1(0)=0$, $x_1^*=x_2^*=0$,

$y_2=2$ 时, $f_2(2)=f_1(2)=11$, $x_1^*=1, x_2^*=0$,

$y_2=3$ 时, $f_2(3)=\max\{11,16\}=16$, $x_1^*=0, x_2^*=1$,

$y_2=4$ 时, $f_2(4)=\max\{22,16\}=22$, $x_1^*=2, x_2^*=0$,

$y_2=5$ 时, $f_2(5)=\max\{22,16+11\}=27$, $x_1^*=x_2^*=1$,

$y_2=6$ 时, $f_2(6)=\max\{33,16+11,32\}=33$, $x_1^*=3, x_2^*=0$,

$y_2=7$ 时, $f_2(7)=\max\{33,16+22,32\}=38$, $x_1^*=2, x_2^*=1$.

$y_2=8$ 时, $f_2(8)=\max\{44,16+22,32+11\}=44$, $x_1^*=4, x_2^*=0$,

$y_2=9$ 时, $f_2(9)=\max\{44,16+33,32+11\}=49$, $x_1^*=3, x_2^*=1$,

$y_2=10$ 时, $f_2(10)=\max\{55,16+33,32+22,48\}=55$, $x_1^*=5, x_2^*=0$.

这些数据列在表 1.7.3 上。

表 1.7.3

y_2	0	1	2	3	4	5	6	7	8	9	10
$f_2(y_2)$	0	0	11	16	22	27	33	38	14	49	55
x_1^*	0	0	1	0	2	1	3	2	4	3	5
x_2^*	0	0	0	1	0	1	0	1	0	1	0

$$f_3(10) = \max_{0 \leqslant x_3 \leqslant \left\lfloor \frac{10}{4} \right\rfloor} \{20x_3 + f_2(10 - 4x_3)\}$$

$$= \max\{55, 20 + 33, 40 + 11\} = 55$$

$$x_1^* = 5, x_2^* = x_3^* = 0$$

即携带的全部为第一种物品，这个结果并不奇怪，第一种物品每单位重量值 $5\frac{1}{2}\times 10^3$ 元，而其他两种物品每单位重量值 $16/3=5\frac{1}{3}\times 10^3$ 元和 5×10^3 元。

例 3 整数规划问题。即变量只取 0 或 1 两个值的整数规划问题。

$$\max z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leqslant b,$$

$$x_i = 0,1, \quad i = 1,2,\cdots,n$$

令
$$z_n(b) = \max\{z_{n-1}(b), z_{n-1}(b-a_n)+c_n\}$$

一般地有：

$$z_i(x) = \max\{z_{i-1}(x), z_{i-1}(x-a_i)+c_i\}$$

计算从 $z_0(a)=0$ 开始，并且当 $x<0$ 时令 $z_i(x)=-\infty$

$$z_1(x) = \max\{z_0(x), z_0(x-a_1)+c_1\},$$

$$z_2(x) = \max\{z_1(x), z_1(x-a_2)+c_2\},$$

$$\cdots\cdots$$

$$z_n(b) = \max\{z_{n-1}(b), z_{n-1}(b-a_{n-1})+c_n\}$$

先看一个例子

$$\max z = 2x_1 + 3x_2 + 4x_3,$$

$$x_1 + 2x_2 + 5x_3 \leqslant 6$$

$$x_1, x_2, x_3 = 0 \text{ 或 } 1$$

$$z_0(x) = \begin{cases} 0, & x \geqslant 0 \\ -\infty, & x < 0 \end{cases}$$

$$z_1(x) = \max\{z_0(x), z_0(x-1)+2\}$$

$$= \begin{cases} \max\{\underline{0}, -\infty\} = 0, & 0 \leqslant x < 1, \\ \max\{0, \underline{2}\} = 2, & x \geqslant 1 \end{cases}$$

$$z_2(x) = \max\{z_1(x), z_1(x-2)+3\}$$

$$= \begin{cases} \max\{\underline{0}, -\infty\} = 0, & 0 \leqslant x < 1, \\ \max\{\underline{2}, -\infty\} = 2, & 1 \leqslant x < 2, \\ \max\{2, \underline{3}\} = 3, & 2 \leqslant x < 3, \\ \max\{2, \underline{5}\} = 5, & x \geqslant 3。 \end{cases}$$

$$z_3(x) = \max\{z_2(x), z_2(x-5)+4\}$$

$$= \begin{cases} \max\{\underline{0}, -\infty\} = 0, & x < 1, \\ \max\{\underline{2}, -\infty\} = 2, & 1 \leqslant x < 2, \\ \max\{\underline{3}, -\infty\} = 3, & 2 \leqslant x < 3, \\ \max\{\underline{5}, 4\} = 5, & 3 \leqslant x < 6, \\ \max\{5, \underline{6}\} = 6, & 6 \leqslant x < 7, \\ \max\{5, \underline{7}\} = 7, & 7 \leqslant x < 8, \\ \max\{5, \underline{9}\} = 9, & x \geqslant 8。 \end{cases}$$

由于 $x_1+2x_2+5x_3\leqslant 6$,故 $x_3=1$,$z_2(x-5)+4=6$,$z_2(x-5)=2$,$x_2=0$,$x_1=1$。
故得最优解:

$$\max z = 6$$
$$x_1 = 1, x_2 = 0, x_3 = 1$$

1.8 同顺序流水作业的任务安排问题

设有 m 种加工用的工作母机:

$$M_1,\ M_2,\ \cdots,\ M_m$$

所谓同顺序流水作业是指它的加工顺序是相同的,不妨为

$$M_1 \to M_2 \to \cdots \to M_m$$

即先通过 M_1 加工,然后依次为 M_2,等等。

现有 n 项任务,其加工顺序一样,设为

$$J_1,\ J_2,\cdots,J_n$$

已知矩阵

$$T = (t_{ij})_{m\times n}$$

其中 t_{ij}=任务 J_j 每加工一单元所需 M_i 机器的时数。
求所用时间最短的任务加工顺序。

下面仅就 $m=2$ 的情形加以讨论。令

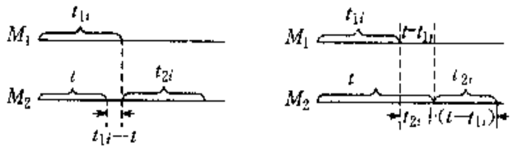
$$S_0 = \{J_1,J_2,\cdots,J_n\},\ N = \{1,2,\cdots,n\}$$

若 n 个任务的加工顺序不同,从第一个任务在机器 M_1 上加工开始,到最后一个任务在机器 M_2 上加工完毕为止,所需的时间也将迥异。从直观上我们知道最佳的安排是使得机器 M_2 的空闲时间达到最少,而对机器 M_1 不存在空闲等任务问题。当然 M_2 也存在任务等机器的状况,即 M_1 加工完毕,而 M_2 还在加工前面一个任务。

设 S 是任务的集合,若机器 M_1 开始加工 S 中的任务时,M_2 机器还在加工其它任务,t 时刻后才可利用,在这样的条件下,加工 S 中任务所需的最短时间设为 $T(S,t)$,则有:

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

其中 $t_{2i}+\max\{t-t_{1i},0\}$的意义可从图 1.8.1 中看出。



(a) $t<t_{1i}$　　(b) $t>t_{1i}$

$\max\{t-t_{1i},0\}=0$　　$\max\{t-t_{1i},0\}=t-t_{1i}$

图 1.8.1

设最佳的方案是 J_i 在前,J_j 在后,则有:

$T(S,t)=t_{1i}+T(S\backslash\{J_i\};t_{2i}+\max\{t-t_{1i},0\})$

$$=t_{1i}+t_{1j}+T(S\backslash\{J_i,J_j\};t_{2j}+\max\{t_{2i}+\max\{t-t_{1i},0\}-t_{1j},0\})$$
$$=t_{1i}+t_{1j}+T(S\backslash\{J_i,J_j\};T_{ij})$$

$$T_{ij}=t_{2j}+\max\{t_{2i}+\max\{t-t_{1i},0\}-t_{1j},0\}$$
$$=t_{2j}+t_{2i}-t_{1j}+\max\{\max\{t-t_{1i},0\},t_{1j}-t_{2i}\}$$
$$=t_{2i}+t_{2j}-t_{1j}+\max\{t-t_{1i},t_{1j}-t_{2i},0\}$$
$$=t_{2i}+t_{2j}-t_{1i}-t_{1j}+\max\{t,t_{1i},t_{1i}+t_{1j}-t_{2i}\}$$
$$=\begin{cases}t+t_{2i}+t_{2j}-t_{1i}-t_{1j}, & 若\ \max\{t,t_{1i},t_{1i}+t_{1j}-t_{2i}\}=t\\ t_{2i}+t_{2j}-t_{1j}, & 若\ \max\{t,t_{1i},t_{1i}+t_{1j}-t_{2i}\}=t_{1i};\\ t_{2j}, & 若\ \max\{t,t_{1i},t_{1i}+t_{1j}-t_{2i}\}=t_{1i}+t_{1j}-t_{2i}\end{cases}$$

如若最优次数 $J_i\to J_j$ 的加工顺序互换，则有：

$$\overline{T}(S;t)=t_{1i}+t_{1j}+T(S\backslash\{J_i,J_j\};T_{ji})$$

其中

$$T_{ji}=t_{2i}-t_{2j}-t_{1i}-t_{1j}+\max\{t,t_{1j},t_{1i}+t_{1j}-t_{2j}\}$$

如若：

$$\max\{t,t_{1i}+t_{1j}-t_{2i},t_{1i}\}\leqslant\max\{t,t_{1i}+t_{1j}-t_{2j},t_{1j}\}\tag{1.8.1}$$

则
$$T(S,t)\leqslant\overline{T}(S;t)$$

若下面条件成立，则式(1.8.1)成立

$$t_{1i}+t_{1j}+\max\{-t_{2i},-t_{1j}\}\leqslant t_{1i}+t_{1j}+\max\{-t_{2j},-t_{1i}\}$$

即

$$\min\{t_{2j},t_{1i}\}\leqslant\min\{t_{2i},t_{1j}\}\tag{1.8.2}$$

式(1.8.2)便是 Johnson 公式。也就是说(1.8.2)式成立时，则任务 J_i 安排在任务 J_j 之前加工。意思是在 M_1 上加工时间短的任务应优先，而在机器 M_2 上加工时间短的任务应排在后面。因而将 $t_{11},t_{12},t_{21},t_{22},\cdots,t_{n1},t_{n2}$ 按从小到大的顺序排列，若最小的是 t_{k1}，则 J_k 排在第一个，若 t_{k2} 为最小，则 J_k 排在最后一个。并从序列中排除 t_{k1} 和 t_{k2}，然后再依次观察余下的序数中的最小数直至 n 个任务都排完。

例 某印刷厂有 6 项加工任务 J_1,J_2,J_3,J_4,J_5,J_6，在印刷车间各需时间 3,12,5,2,9,11 单位，在装订车间需 8,10,9,6,3 和 1 单位。即

$$T=\begin{pmatrix}3 & 12 & 5 & 2 & 8 & 11\\ 7 & 10 & 9 & 6 & 4 & 1\end{pmatrix}_{2\times 6}$$

将矩阵 T 的元素从小到大按次序排列得：

$$\begin{array}{cccccccccccc}1< & 2< & 3< & 4< & 5< & 6< & 7< & 8< & 9< & 10< & 11< & 12\\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow\\ t_{26} & t_{14} & t_{11} & t_{23} & t_{13} & t_{24} & t_{21} & t_{15} & t_{23} & t_{22} & t_{16} & t_{12}\end{array}$$

这序列中最小元素为 t_{25}，故 J_6 是最后加工，从序列中删去 t_{16},t_{26} 剩下序列中求最小元素依此类推。

按上面所述算法得最佳的加工顺序应为：

$$J_1\to J_1\to J_3\to J_2\to J_5\to J_6$$

加工总时间为 43 单位。

1.9 可靠性问题

对于开关网络、计算机、大型精密仪器等复杂系统常提出这样的问题，如何利用已知可靠性的元件构成高可靠性的系统。一个系统可简单地看作是从输入经 n 级到达输出，见图 1.9.1。

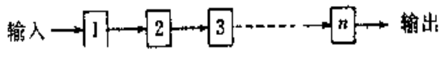


图 1.9.1

为提高可靠性，每一级可用多个相同元件并联。设每个元件出故障的概率为 $0<p<1$，两个元件并联同时出故障的概率将降为 p^2，三个元件并联同时出故障的概率将降到 $p^3<p^2<p$，如此等等，见图 1.9.2。

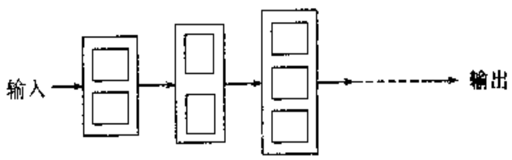


图 1.9.2

从理论上，可以通过增加元件的数量使每一级的可靠性达到任意高的程度，实际上这是办不到的。因为它将受到其他条件的约束，比如代价。因为重量、代价、大小都有限制。只能在某些限制条件允许的情况下使可靠性达到尽可能的高。

令 $p_j(x_j)$＝第 j 级有 x_j 个元件时该级正常运转的概率；

c_j 表示第 j 级每个元件的代价；

a_j 表示第 j 级每个元件的重量；

a 和 c 分别表示允许重量和代价的上限；

于是问题便成为如下的优化问题：

$$\max z = p_1(x_1)p_2(x_2)\cdots p_n(x_n)$$

约束条件为：

$$\sum_{j=1}^{n} c_j x_j \leqslant c,$$

$$\sum_{j=1}^{n} a_j x_j \leqslant a,$$

$$x_j \text{ 为正整数}, j=1,2,\cdots。$$

利用最佳原理可将其化为多段判决如下：

令 $f_i(u^{(i)},v^{(i)})$表示第 i 级允许重量上限为 $v^{(i)}$，代价上限为 $u^{(i)}$时的最高可靠性。则有：

$$f_1(u^{(1)},v^{(1)}) = \max_{0\leqslant x_1\leqslant y_1} p_1(x_1),$$

$$y_1=\min\left\{\left\lfloor\frac{u^{(1)}}{c_1}\right\rfloor,\left\lfloor\frac{v^{(1)}}{a_1}\right\rfloor\right\}$$

$$f_k(u^{(k)},v^{(k)})=\max_{1\leqslant x_k\leqslant y_k}\left[p_k(x_k)f_{k-1}(u^{(k)}-c_kx_k,v^{(k)}-a_kx_k)\right]$$

其中
$$y_k=\min\left\{\left\lfloor\frac{u^{(k)}}{c_k}\right\rfloor,\left\lfloor\frac{v^{(k)}}{a_k}\right\rfloor\right\}$$

例 试设计一个三级系统 S，分别用的三种元件 D_1,D_2,D_3 的价格与可靠性如下表：

元件	D_1	D_2	D_3
单价	40	30	60
可靠性	0.5	0.8	0.9

若系统 S 的代价不超过 220 元，试问应如何设计使可靠性达到最高。本例不考虑重量的限制。

$$f_3(u)=\max_{1\leqslant x_3\leqslant y_3}\{[1-(0.1)^{x_3}]f_2(u-30x_3)\}$$

其中
$$y_3=\left\lfloor\frac{220}{60}\right\rfloor=3$$

D_3 的可靠性为 0.9，失败的概率为 $1-0.9=0.1$，x_3 个元件并联全部失败的概率为 $(0.1)^{x_3}$，$1-(0.1)^{x_3}$ 是至少一个工作正常的概率。

所以

$$f_3(220)=\max_{1\leqslant x_3\leqslant 3}\{[1-(0.1)^{x_3}]f_2(220-60x_3)\}$$
$$=\max\{0.9f_2(160),0.99f_2(100),0.999f_2(40)\}$$

而

$$f_2(u)=\max_{1\leqslant x_2\leqslant y_2}\{(1-(0.2)^{x_2})f_1(u-30x_2)\}$$

其中
$$y_2=\left\lfloor\frac{u}{30}\right\rfloor$$

$$f_1(u)=\max_{1\leqslant x_1\leqslant y_1}\{1-(0.5)^{x_1}\}$$

其中
$$y_1=\left\lfloor\frac{u}{20}\right\rfloor$$

$u=160$ 时，$y_2=\left\lfloor\frac{160}{30}\right\rfloor=5$。

故 $f_2(160)=\max\{0.8f_1(130),0.96f_1(100),0.992f_1(70),0.9984f_1(40),$
$0.99982f_1(10)\}$

但

$$f_1(130)=\max\{0.5,0.75,0.875^*\}=0.875$$
$$f_1(100)=\max\{0.5,0.75\}=0.75$$
$$f_1(70)=0.5$$
$$f_1(40)=0.5$$
$$f_1(10)=0$$

所以 $f_2(160)=\max\{0.52,0.72^*,0.496,0.4992,0\}=0.72$

同理可得

$$\begin{aligned} f_2(100) &= \max\{0.8f_1(70), 0.96f_1(40), 0.992f_1(10)\} \\ &= \max\{0.4, 0.48^*, 0\} \\ &= 0.48 \\ f_2(40) &= 0.8f_1(10) = 0 \end{aligned}$$

所以

$$\begin{aligned} f_3(220) &= \max\{0.648, 0.4752, 0\} \\ &= 0.648 \end{aligned}$$

即当 $x_1=1, x_2=2, x_3=2$ 时代价为 200 元，可靠性达到 0.648。

1.10 设备更新问题

随着使用年限的增加，机器的使用效率降低，收入减少，维护费用增加，将旧的卖掉换新的设备，更新的费用也随之增加。见表 1.10.1。

表 1.10.1

产品年代	现有					第一年					第二年				第三年			第四年		第五年
机器岁数	1	2	3	4	5	0	1	2	3	4	0	1	2	3	0	1	2	0	1	0
收入	18	16	16	14	14	22	21	20	18	16	27	25	24	22	29	26	24	30	28	32
费用	8	8	9	9	10	6	6	8	8	10	5	6	8	9	5	5	6	4	5	4
更新费用	32	34	36	36	38	27	29	32	34	37	29	31	34	36	31	32	33	32	33	34

试问 5 年内机器更新的最佳策略应该如何？最佳策略指的是第一年开始买进机器后，决定在这 5 年内哪年设备更新，最后要使这 5 年内的总收入达到最大，或总支出达到最少。

表 1.10.1 需说明的，比如产品年代第一年指的是计划开始的第一年的产品。“现有”栏指的是计划开始的旧机器，有岁数为 1,2,…,5 的机器。机器岁数指的是用过的年数。

设备更新讨论的是 5 年内，各年分别对不同岁数的机器是更新还是留用作出论证。

引进符号：

$E_l(t)$ = 第 l 年岁数 t 的机器的收入；

$D_l(t)$ = 第 l 年岁数 t 的机器的费用；

$N_l(t)$ = 第 l 年岁数 t 的机器的更新费用；

$G_l(t)$ = 第 l 年开始时机器的岁数为 t 的最佳决策的收入，即在 $l, l+1, \cdots, l+N$ 年内的最佳收入。

显然有两种选择，一是继续留用，一是更新机器，即

$$G_l(t) = \max\{n_l(t), k_l(t)\}$$

其中

$$\begin{aligned} n_l(t) &= E_l(0) - D_l(0) - N_l(t) + G_{l+1}(1) \\ k_l(t) &= E_l(t) - D_l(t) + G_{l+1}(t+1) \end{aligned}$$

即 $n_l(t)$ 为第 l 年开始，把岁数 t 的机器更新为当年产品后的收入，而 $k_l(t)$ 为若上述机器继续留用的收入。更新还是留用取决于 $n_l(t)$ 和 $k_l(t)$ 的大小。

第 l 年开始时岁数为 t 的机器，其制造年代应为 $l-t$ 年。研究的是今后 N 年的计划，假定：

$$G_{N+1}(t) = 0$$

对于本题若设 $N=5$，即制定 5 年的设备更新政策。

1. $l=5$，从第 5 年开始的策略：

$$n_5(t) = E_5(0) - D_5(0) - N_5(t) + G_6(1)$$

$$k_5(t) = E_5(t) - D_5(t) + G_6(t+1)$$

$E_5(0)$ 为第 5 年新产品的收入，从表 1.10.1 可知 $E_5(0)=32$，同理 $D_5(0)=4$，但 $N_5(1)=33$，$N_5(2)=33$，$N_5(3)=36$，$N_5(4)=37$。第 5 年岁数为 1 的机器，应为第 4 年的产品。第 4 年产品岁数为 1 的更新费用为 33，故 $N_5(1)=33$。类似第 5 年岁数为 3 的机器应为第 2 年的产品，第 2 年的产品岁数为 3 的更新费用为 36。又如 $E_5(1)$ 应为第 4 年的产品，岁数为 1，故 $E_5(1)=28$，余此类推。

所以

$$\begin{aligned} n_5(1) &= 32 - 4 - 33 + 0 = -5 \\ k_5(1) &= 28 - 5 + 0 = 23 \\ G_5(1) &= \max\{n_5(1), k_5(1)\} \\ &= \max\{-5, 23\} \\ &= 23 \end{aligned}$$

由于 $n_5(1)<k_5(1)$，故第 5 年岁数为 1 的机器保留使用。

$$\begin{aligned} n_5(2) &= 32 - 4 - 33 + 0 = -5 \\ k_5(2) &= 24 - 6 + 0 = 18 \\ G_5(2) &= \max\{n_5(2), k_5(2)\} \\ &= \max\{-5, 18\} \\ &= 18 \end{aligned}$$

故岁数为 2 的机器保留使用。

$$\begin{aligned} n_5(3) &= 32 - 4 - 36 + 0 = -8 \\ k_5(3) &= 22 - 9 + 0 = 13 \\ G_5(3) &= \max\{n_5(3), k_5(3)\} \\ &= \max\{-8, 13\} \\ &= 13 \end{aligned}$$

故岁数为 3 的机器保留使用。

$$\begin{aligned} n_5(4) &= 32 - 4 - 37 + 0 = -9 \\ k_5(4) &= 16 - 10 + 0 = 6 \\ G_5(4) &= \max\{n_5(4), k_5(4)\} \\ &= \max\{-9, 6\} \\ &= 6 \end{aligned}$$

故岁数为 4 的机器保留使用。

$$n_5(5) = 32 - 4 - 38 + 0 = -10$$

$$k_5(5) = 14 - 10 + 0 = 4$$
$$G_5(5) = \max\{N_5(5), k_5(5)\} = \max\{-10, 4\} = 4$$

故岁数为 5 的机器保留使用。

2. $l=4$，第 4 年开始的策略：

$$n_4(t) = E_4(0) - D_4(0) - N_4(t) + G_5(1)$$

$\because$ $G_5(1) = 23$

$\therefore$ $k_4(t) = E_4(t) - D_4(t) + G_5(t+1)$

$$n_4(1) = 30 - 4 - 32 + 23 = 17$$

$\because$ $G_5(2) = 18$

$\therefore$ $k_4(1) = 26 - 5 + 18 = 39$

$$G_4(1) = \max\{n_4(1), k_4(1)\} = \max\{17, 39\} = 39$$

故岁数为 1 的机器继续使用。

$$n_4(2) = 30 - 4 - 34 + 23 = 15$$

$\because$ $G_5(3) = 13$

$\therefore$ $k_4(2) = 24 - 18 + 13 = 29$

$$G_4(2) = \max\{n_4(2), k_4(2)\} = \max\{15, 29\} = 29$$

故岁数为 2 的机器留用。

$$n_4(3) = 30 - 4 - 34 + 23 = 15$$

$\because$ $G_5(4) = 6$

$\therefore$ $k_4(3) = 18 - 8 + 6 = 16$

$$G_4(3) = \max\{n_4(3), k_4(3)\} = \max\{15, 16\} = 16$$

故岁数为 3 的机器留用。

$$n_4(4) = 30 - 4 - 36 + 23 = 13$$

$\because$ $G_5(5) = 4$

$\therefore$ $k_4(4) = 14 - 9 + 4 = 9$

$$G_4(4) = \max\{n_4(4), k_4(4)\} = \max\{13, 9\} = 13$$

故岁数为 4 的机器应更新。

3. $l=3$，第 3 年开始的策略：

$$n_3(t) = E_3(0) - D_3(0) - N_3(t) + G_4(1)$$
$$k_3(t) = E_3(t) - D_3(t) + G_4(t+1)$$

$$\because \quad G_4(1) = 39$$

$$n_3(1) = 29 - 5 - 31 + 39 = 32$$

$$\because \quad G_4(2) = 29$$

$$k_3(1) = 25 - 6 + 29 = 48$$

$$\begin{aligned} G_3(1) &= \max\{n_3(1), k_3(1)\} \\ &= \max\{32, 48\} \\ &= 48 \end{aligned}$$

故岁数为 1 的机器留用。

$$n_3(2) = 29 - 5 - 32 + 39 = 31$$

$$\because \quad G_4(3) = 16$$

$$\therefore \quad k_3(2) = 20 - 8 + 16 = 28$$

$$\begin{aligned} G_3(2) &= \max\{n_3(2), k_3(2)\} \\ &= \max\{31, 28\} \\ &= 31 \end{aligned}$$

故岁数为 2 的机器应更新。

$$n_3(3) = 29 - 5 - 36 + 39 = 27$$

$$\because \quad G_4(4) = 13$$

$$\therefore \quad k_3(3) = 16 - 9 + 13 = 20$$

$$\begin{aligned} G_3(3) &= \max\{n_3(3), k_3(3)\} \\ &= \max\{27, 20\} \\ &= 27 \end{aligned}$$

故岁数为 3 的机器应更新。

4. $l=2$，第 2 年开始的策略：

$$n_2(t) = E_2(0) - D_2(0) - N_2(t) + G_3(1)$$

$$k_2(t) = E_2(t) - D_2(t) + G_3(t + 1)$$

$$\because \quad G_3(1) = 48$$

$$n_2(1) = 27 - 5 - 29 + 48 = 41$$

$$\because \quad G_3(2) = 31$$

$$k_2(1) = 21 - 6 + 31 = 46$$

$$\begin{aligned} G_2(1) &= \max\{n_2(1), k_2(1)\} \\ &= \max\{41, 46\} \\ &= 46 \end{aligned}$$

故岁数为 1 的机器留用。

$$n_2(2) = 27 - 5 - 34 + 48 = 36$$

$$\because \quad G_3(3) = 27$$

$$k_2(2) = 16 - 8 + 27 = 35$$

$$\begin{aligned} G_2(2) &= \max\{n_2(2), k_2(2)\} \\ &= \max\{36, 35\} \\ &= 36 \end{aligned}$$

故岁数为 2 的机器应更新。

5. $l=1$，第 1 年开始的策略：

$$n_1(t) = E_1(0) - D_1(0) - N_1(t) + G_2(1)$$
$$k_1(t) = E_1(t) - D_1(t) + G_2(t+1)$$

$\because$
$$G_2(1) = 46$$
$\therefore$
$$n_1(1) = 22 - 6 - 32 + 46 = 30$$
$\because$
$$G_2(2) = 36$$
$\therefore$
$$k_1(1) = 18 - 8 + 36 = 46$$
$$\begin{aligned} G_1(1) &= \max\{n_1(1), k_1(1)\} \\ &= \max\{30, 46\} \\ &= 46 \end{aligned}$$

故岁数为 1 的机器留用。

习　　题

1. 利用最佳原理解：

$$\max z = 110x_1 + 160x_2 + 260x_3 + 210x_4$$
$$2x_1 + 3x_2 + 5x_3 + 4x_4 \leqslant 20$$
$$x_1, x_2, x_3, x_4 \text{ 为零或正整数}$$

2.
$$\min z = x_1^2 + x_2^2 + x_3^2$$
$$x_1 + x_2 + x_3 \geqslant 100$$
$$x_1, x_2, x_3 \geqslant 0$$

3. 利用最佳原理解下面流动推销员问题。

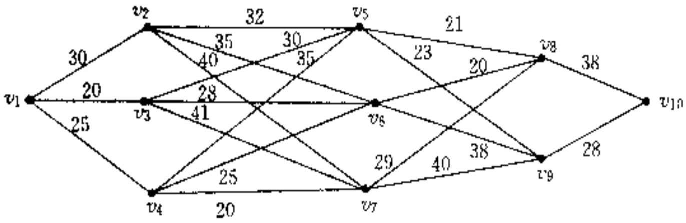
$$\begin{array}{rl} D = \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} & \begin{pmatrix} 0 & 10 & 8 & 18 & 14 \\ 10 & 0 & 7 & 11 & 4 \\ 8 & 7 & 0 & 6 & 5 \\ 18 & 11 & 6 & 0 & 9 \\ 14 & 4 & 5 & 9 & 0 \end{pmatrix} \\ & \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \end{array}$$

4.

$$\begin{array}{rl} D = \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} & \begin{pmatrix} 0 & 10 & 20 & 30 & 40 & 50 \\ 12 & 0 & 18 & 30 & 25 & 21 \\ 23 & 19 & 0 & 5 & 10 & 15 \\ 34 & 32 & 4 & 0 & 8 & 16 \\ 45 & 27 & 11 & 10 & 0 & 18 \\ 56 & 22 & 16 & 20 & 12 & 0 \end{pmatrix} \\ & \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \end{array}$$

5. 试对利用最佳原理解流动推销员问题所作的比较次数进行估计。

6. 求图题 1.1 中从 v_1 到 v_{10} 的最短路径。



图题 1.1

7.

$$\max z=g_1(x_1)+g_2(x_2)+g_3(x_3)$$

$$x_1^2+x_2^2+x_3^2\leqslant 20$$

$$x_1,x_2,x_3 \text{ 为零或正整数}$$

其中函数 $g_1(x_1),g_2(x_2),g_3(x_3)$ 见表题 1.1

表题 1.1

x_i	0	1	2	3	4	5	6	7	8	9	10
$g_1(x_i)$	2	4	7	11	13	15	18	22	18	15	11
$g_2(x_i)$	5	10	15	20	24	18	12	9	5	3	1
$g_3(x_i)$	8	12	17	22	19	16	14	11	9	7	4

8. 若有 4 项任务 J_1,J_2,J_3,J_4，要先后使用机器 M_1,M_2，使用 M_1,M_2 机器的时间见表题 1.2。

表题 1.2

J_i	J_1	J_2	J_3	J_4
M_1	3	4	8	10
M_2	6	2	9	15

求任务的最佳安排。

9. 有一个长度为 10 的背包，还有 4 类物品，权重分别为 $v_1=1,v_2=3,v_3=5,v_4=9$，其长度分别为 $w_1=2,w_2=3,w_3=4,w_4=7$，试问如何放置才能使背包中所装物品的权值最大。

10. 已知 $A_k=(a_{ij}^{(k)})_{r_k\times r_{k+1}},k=1,2,3,4,5,6,r_1=5,r_2=10,r_3=3,r_4=12,r_5=5,r_6=50,r_7=6$，求矩阵链积 A_1,A_2,A_3,A_4,A_5,A_6 的最佳求积顺序。

11. (*a*) 已知

$$A=(1,0,0,1,0,1,0,1)$$

$$B=(0,1,0,1,1,0,1,1,0)$$

求 $LCS(A,B)$。

(b) 已知

$$X=(A,B,C,B,D,A,B)$$
$$Y=(B,D,C,A,B,A)$$

求 $LCS(X,Y)$。

12. 已知序列 $a_1,a_2,\cdots,a_n$，试设计一算法从中找出一子序列

$$a_{i_1}<a_{i_2}<\cdots<a_{i_k}$$

使 k 达到最大，并讨论其复杂性。

13. 已知一凸 n 边形 K：$v_1v_2\cdots v_n$，利用互不交于 K 内的弦 $\overline{v_iv_j}$ 将 K 分割成 $n-2$ 个三角形，对于 $\Delta v_iv_jv_k$ 定义一权 $w(\Delta v_iv_jv_k)=|\overline{v_iv_j}|+|\overline{v_jv_k}|+|\overline{v_kv_i}|$，其中 $|\overline{v_iv_j}|$ 表示弦 $\overline{v_iv_j}$ 的长度，试设计一算法，使权达到最小，并讨论其复杂性。

14. 有金币 15 枚，它们重量一样。已知其中有一枚是假的，而且它的重量比真币为轻。要求用一天平将假的金币找出来，试设计一种算法（方案），使在最坏情况下用天平的次数最少。

15. 令 A 是由 n 个不同整数构成的序列，试设计一算法求 A 中最长的单调增（或减）子序列。

16. 已有 n 个整数，是否存在一种办法将它们分为两部分，使得各自的和相等？当已知整数序列 $a_1,a_2,\cdots,a_n$，是否存在 $\{1,2,\cdots,n\}$ 的子集 I，使得

$$\sum_{i\in I}a_i=\sum_{j\notin I}a_j$$

17. 试利用 1.6 即求图 $G=(V,E)$ 的任意两点间最短距离的方法，求图 G 的路径矩阵，

$$P=(p_{ij})_{n\times n}\qquad n=|V|$$

$$p_{ij}=\begin{cases}1, & \text{若 } v_i \text{ 到 } v_j \text{ 有路径相通}\\ 0, & \text{其它}\end{cases}$$

已知图 G 的邻接 $A=(a_{ij})_{n\times n}$，其中

$$a_{ij}=\begin{cases}1, & \text{若}(v_i,v_j)\in E\\ 0, & \text{其它。}\end{cases}\qquad i,j\in N$$

第2章 优先策略

优先策略顾名思义是“择优录取”，在某些方面的应用是非常成功的，也是我们设计算法时经常使用的一种策略。国外也叫做 Greedy method，意即见到好的就抓住不放。还是优先策略比较贴切。

2.1 最短树的 Kruskal 算法

设图 $G=(V,E)$ 是一简单连通图，$|V|=n$，$|E|=m$，每条边 e_i 都给以权 w_i，w_i 假定是边 e_i 的长度(也可以是其他)，$i=1,2,\cdots,m$。求图 G 的总长度最短的树。这就是最短树问题。

最短树的 Kruskal 算法的基本思想是：首先将赋权图 G 的边按权的递增顺序排列，不失一般性设为

$$e_1,e_2,\cdots,e_m$$

其中 $w_k\leqslant w_{k+1}$。然后在不构成回路的条件下，择优取进权最小的边。

求如图 2.1.1 所示的赋权图的最优树，数字表示该边的长度。把权按权的递增顺序排列。

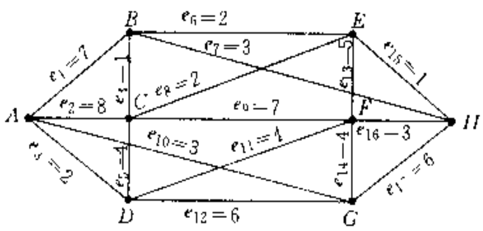


图 2.1.1

$e_3^{(1)}=1$，$e_{15}^{(2)}=1$，$e_4^{(3)}=2$，$e_6^{(4)}=2$，$e_8^{(5)}=2$，$e_7^{(6)}=3$，
$e_{10}^{(7)}=3$，$e_{16}^{(8)}=3$，$e_5^{(9)}=4$，$e_{11}^{(10)}=4$，$e_{14}^{(11)}=4$，$e_{13}^{(12)}=5$，
$e_{12}^{(13)}=6$，$e_{17}^{(14)}=6$，$e_1^{(15)}=7$，$e_9^{(16)}=7$，$e_2^{(17)}=8$。

右上肩括号里的数是排序的序号。按 Kruskal 算法的基本思想得到图 2.1.1 的最短树 T(图 2.1.2)。

T 中每条边括号里的数为进入 T 的序号，T 的权是 16。

下面给出求最短树的 Kruskal 算法，为方便起见不妨设 e_i 就是边的长度：

(1) 对属于 E 的边进行排序得

$$e_1\leqslant e_2\leqslant\cdots\leqslant e_m$$

(2) 初始化操作：$w\leftarrow 0$，$T\leftarrow\varnothing$，$k\leftarrow 0$，$t\leftarrow 0$。

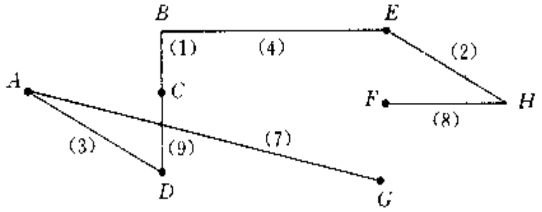


图 2.1.2

(3) 若 $t=n-1$,则转(6)。否则转(4)。

(4) 若 $T\cup\{e_k\}$ 构成一回路,则作

$$【k\leftarrow k+1,转(4)】。$$

(5) $T\leftarrow T\cup\{e_k\}, w\leftarrow w+w_k$,

$$t\leftarrow t+1, k\leftarrow k+1,转(3)。$$

(6) 输出 T,w,停止。

算法中 T 中边就是最短树的树枝,w 给出最短树的权。

Kruskal 算法首先要求对图 G 的边进行排序,以后我们将看到排序的时间复杂性为 $O(m\log_2 m)$。

2.2 求最短树的 Prim 算法

Kruskal 算法采取在不构造回路的条件下,优先选择长度最短的边作为最短树的边,而下面介绍的 Prim 算法采取另外一种优先策略:

已知图 $G=(V,E)$,$V=\{v_1,v_2,\cdots,v_n\}$,$D=(d_{ij})_{n\times n}$ 是图 G 的距离矩阵,若 $(v_i,v_j)\notin E$,则令 $d_{ij}=\infty$,并假定 $d_{ii}=\infty$。

Prim 算法的基本思想是:从某一顶点(设为 v_1)开始,令 $S\leftarrow\{v_1\}$,求 $V\backslash S$ 中点与 S 中点 v_1 距离最短的点,即从矩阵 D 的第一行元素中找到最小的元素,设为 d_{1j},则令 $S\leftarrow S\cup\{v_j\}$,继续求 $V\backslash S$ 中点与 S 的距离最短的点,设为 v_k,则令 $S\leftarrow S\cup\{v_k\}$,继续以上的步骤,直到 n 个顶点用 $n-1$ 条边连接起来为止。还是以图 2.1.1 为例(假定从 A 点出发)来进行说明。

图 2.2.1 给出了利用 Prim 算法对图 2.1.1 的求最短树的过程。

下面给出求最短树的 Prim 算法:

(1) 初始化操作:$T\leftarrow\varnothing$,$q(1)\leftarrow -1$,i 从 2 到 n 作【$p(i)\leftarrow 1$,$q(i)\leftarrow d_{i1}$】,$k\leftarrow 1$。

(2) 若 $k\geqslant n$ 则作【输出 T,结束】。

否则,作【$\min\leftarrow\infty$;

j 从 2 到 n 作

【若 $0<q(i)<\min$ 则作【$\min\leftarrow q(i)$,$h\leftarrow j$】】】

(3) $T\leftarrow T\cup\{(h,p(h))\}$,$q(h)\leftarrow -1$。

(4) j 从 2 到 n 作

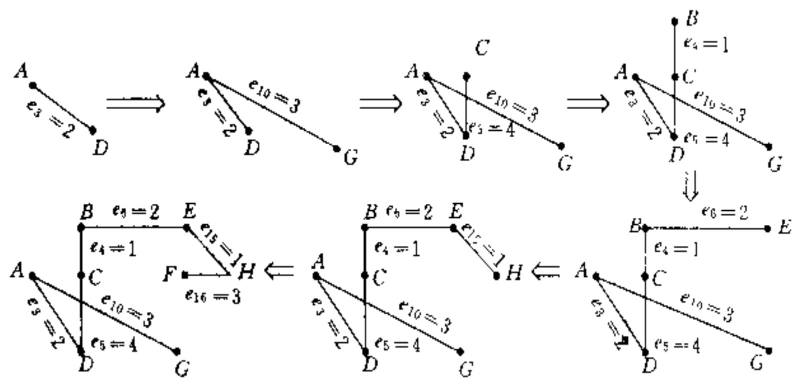


图 2.2.1

【若 $d_{hj}<q(j)$ 则作【$q(j)\leftarrow d_{hj}$, $p(j)\leftarrow h$】】

(5) $k\leftarrow k+1$，转(2)。

算法中数组 $p(i)$ 是用以记录和 v_i 点最接近的属于 S 的点，$q(i)$ 则是记录了 v_i 点和 S 中点的最短距离。$q(i)=-1$ 用以表示 v_i 点已进入集合 S。算法中第四步：v_h 点进入 S 后，对不属于 S 中的点 v_j 的 $p(j)$ 和 $q(j)$ 进行适当调整，使之分别记录了所有不属于 S 且和 S 距离最短的点和最短的距离。点 $v_1,v_2,\cdots,v_n$ 分别用 $1,2,\cdots,n$ 表示。

由于 Prim 算法没要求对边事先排序，所以其时间复杂性为 $O(n^2)$。

2.3 求最短路径的 Dijkstra 算法

已知图 $G=(V,E)$，$V=\{v_1,v_2,\cdots,v_n\}$，$D=(d_{ij})_{n\times n}$ 是距离矩阵，求 v_1 点到其它各点的最短路径的 Dijkstra 算法如下，其中 $l(v_i)$ 表 v_i 点与 S 集合中的点最短距离，S 是 V 的子集。

(1) 初始化操作：$S\leftarrow\{v_1\}$，$l(v_1)\leftarrow 0$，$l(v_i)\leftarrow\infty$，$i=2,3,\cdots,n$。$i\leftarrow 0$，$\bar{S}\leftarrow V\setminus\{v_1\}$。

(2) 若 $\bar{S}$ 是空集，则作【打印 S 后停止】。否则转(3)。

(3) 对 $v_i\in\bar{S}$ 的所有点计算：

$$l(v_i)=\min_{v_j\in S}\{l(v_i),l(v_j)+d_{ji}\}$$

(4) 令 $l(v_{i+1})=\min\limits_{v\in\bar{S}}\{l(v)\}$，$S\leftarrow S\cup\{v_{i+1}\}$

$$\bar{S}\leftarrow\bar{S}\setminus\{v_{i+1}\},\ i\leftarrow i+1,\ \text{转(2)}。$$

Dijkstra 算法的时间复杂性也为 $O(n^2)$。

例 见图 2.3.1。

求图 2.3.1 中 A 点到其它各点的最短路径的过程如下(见图 2.3.2)。

图 2.3.2 中( )里的数表从 A 点到该点的最短路径长度。

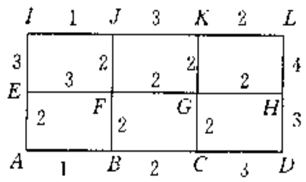


图 2.3.1

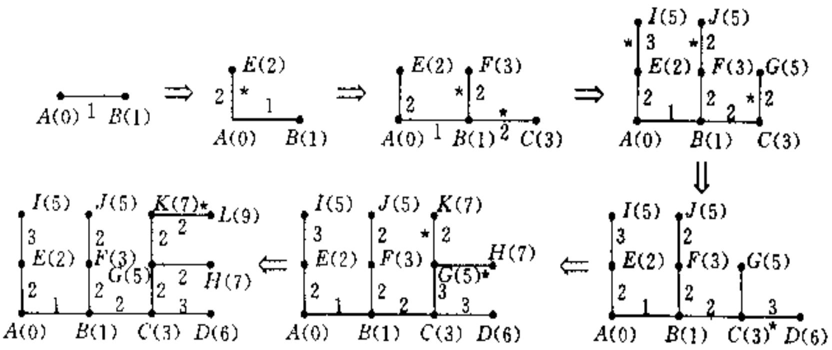


图 2.3.2

2.4 文件存储问题

文件存储问题是指文件若存放在磁带上的最佳存放顺序。设有 n 个文件 $f_1, f_2, \cdots, f_n$，长度分别为 $l_1, l_2, \cdots, l_n$，记录在一条或几条磁带上，调用的频率分别是 $h_1, h_2, \cdots, h_n(h_1+h_2+\cdots+h_n=1)$，问文件的记录次序应如何安排，使得查询时磁带平均转动的长度或时间最短？(磁带是顺序存储的)。

首先考虑一条磁带的存储安排问题：

1. 假定 $h_1=h_2=\cdots=h_n$ 的情形，即文件 $f_1, f_2, \cdots, f_n$ 被查找调用的频率是一样的。

设有 5 个文件 f_1, f_2, f_3, f_4, f_5 长度分别为 2,3,5,1,4。如若存放次序为 f_3, f_5, f_2, f_1, f_4，由于假定 $f_1, f_2, \cdots, f_5$ 查询的概率相同，故查阅文件一次要转动磁带的平均长度为：

$$\frac{1}{5}[5+(5+4)+(5+4+3)+(5+4+3+2)+(5+4+3+2+1)]$$

$$=\frac{1}{5}[5\times 5+4\times 4+3\times 3+2\times 2+1]$$

$$=\frac{1}{5}\times 55=11$$

如若存放次序为 f_4, f_1, f_2, f_5, f_3，则查阅文件一次平均需检索磁带的长度为：

$$\frac{1}{5}[1+(1+2)+(1+2+3)+(1+2+3+4)+(1+2+3+4+5)]$$

$$=\frac{1}{5}[5\times 1+4\times 2+3\times 3+2\times 4+5]$$

$$=\frac{1}{5}\times 35=7$$

不失一般性，设 $f_1, f_2, \cdots, f_n$ 是满足条件：

$$l_1\leqslant l_2\leqslant\cdots\leqslant l_n$$

的文件安排顺序，则可证明此时 $\sum\limits_{k=1}^{n}\sum\limits_{j=1}^{k}l_j$ 取最小值。

$$
\begin{array}{l}
S = \quad l_1 \\
\quad +l_1+l_2+ \\
\quad \quad \cdots\cdots \\
\quad +l_1+l_2+\cdots+l_h+ \\
\quad \quad \cdots\cdots \\
\quad +l_1+l_2+\cdots+l_h+\cdots+l_k+ \\
\quad \quad \cdots\cdots \\
\quad +l_1+l_2+\cdots+l_h+\cdots+l_k+\cdots+l_n \\
\hline
S=nl_1+(n-1)l_2+\cdots+(n-h+1)l_h+\cdots+(n-k+1)l_k+\cdots+l_n
\end{array}
$$

从上可见若 $l_h<l_k$，则 f_h 和 f_k 的位置互换将导致 S 增加为 S_1，增加的数量等于 S_1-S。

由于 $S_1=nl_1+(n-1)l_2+\cdots+(n-h+1)l_k+\cdots+(n-k+1)l_h+\cdots+l_n$

$$
\begin{aligned}
\Delta S &= S_1 - S \\
&= [nl_1 + (n-1)l_2 + \cdots + (n-h+1)l_k + \cdots + (n-k+1)l_h + \cdots + l_n] \\
&\quad - [nl_1 + (n-1)l_2 + \cdots + (n-h+1)l_h + \cdots + (n-k+1)l_k + \cdots + l_n] \\
&= [(n-h+1)l_k + (n-k+1)l_h] - [(n-h+1)l_h + (n-k+1)l_k] \\
&= (n-h+1)(l_k-l_h) - (n-k+1)(l_k-l_h) \\
&= (l_k-l_h)(k-h) > 0
\end{aligned}
$$

上面结果说明任何两个长度不等的文件的次序的改变都将导致平均检索时间的增加。也就证明了满足条件 $l_1\leqslant l_2\leqslant\cdots\leqslant l_n$ 的文件顺序是最优的存储顺序。

上述方法就是若查询的频率相同时文件长度短的优先排在前头。

2. $l_1=l_2=\cdots=l_n$ 但查找频率不同的情况。

同上面一样，n 个文件 $f_1,f_2,\cdots,f_n$ 的最佳的存储顺序 $f_1,f_2,\cdots,f_n$ 应使调用 f_i 的频率满足：

$$h_1 \geqslant h_2 \geqslant \cdots \geqslant h_n$$

即调用频率高的文件应记录在前。可以证明其使

$$S = (h_1 + 2h_2 + \cdots + nh_n)l$$

取得最小值；l 为文件长度。证明留作作业。

3. 一般情形，即假定文件 f_i 的长度为 l_i，调用频率为 h_i，可以证明满足：

$$\frac{h_1}{l_1} \geqslant \frac{h_2}{l_2} \geqslant \cdots \geqslant \frac{h_n}{l_n}$$

的文件顺序 $f_1,f_2,\cdots,f_n$，使

$$S = h_1l_1 + h_2(l_1+l_2) + \cdots + h_n(l_1+l_2+\cdots+l_n)$$

取最小值。因而是最佳的存储顺序。

因若 $h_i/l_i>h_{i+1}/l_{i+1}$，f_i 与 f_{i+1} 两个文件位置互换结果使得 S 改变为 S_1，则有

$$
\begin{aligned}
\Delta S = S_1 - S &= h_i(l_1+l_2+\cdots+l_i+l_{i+1}) + h_{i+1}(l_1+l_2+\cdots+l_{i-1}+l_{i+1}) \\
&\quad - h_i(l_1+l_2+\cdots+l_{i-1}+l_i) - h_{i+1}(l_1+l_2+\cdots+l_i+l_{i+1})
\end{aligned}
$$

$$=h_i l_{i+1} - h_{i+1} l_i$$
$$=l_i l_{i+1}\left(\frac{h_i}{l_i} - \frac{h_{i+1}}{l_{i+1}}\right) \geqslant 0$$

也就是说使得 S 增加。

采取的优先策略是 h_i/l_i 大的优先安排在前面。

现在考虑当一条磁带存储不下 n 个文件，要存储在多条(设为 m)磁带的情况。假定查找频率是相同的。

不失一般性，设 n 个文件 $f_1, f_2, \cdots, f_n$ 依其长短从小到大顺序排列，设 f_i 的长度为 l_i，设 $n=(h+1)m$，且

$$l_1 \leqslant l_2 \leqslant \cdots \leqslant l_n$$

若
$$k = pm + q \qquad (1 \leqslant q \leqslant m)$$

则 f_k 存放在第 q 条磁带上，而且在第 q 条磁带上的顺序是第 $p+1$ 个，即

第 1 条磁带：$f_1, f_{m+1}, \cdots, f_{hm+1}$

第 2 条磁带：$f_2, f_{m+2}, \cdots, f_{hm+2}$

…… … … …

第 m 条磁带：$f_m, f_{2m}, \cdots, f_{(h+1)m}$

检索各文件一次所需要的总长度：

$$S=(h+1)(l_1+l_2+\cdots+l_m)+h(l_{m+1}+l_{m+2}+\cdots+l_{2m})+\cdots+(l_{hm+1}+l_{hm+2}+\cdots+l_{(h+1)m})$$

从上式知若 $l_j \leqslant l_i$，则 f_i 和 f_j 的互换不会降低 S，最多使 S 保持不变。比如 $f_1, f_2, \cdots, f_m$ 之间互换不会改变 S 的值；$l_{m+1}, l_{m+2}, \cdots, l_{2m}$ 之间互换不改变 S；…，此外必将会引起 S 的增加。

上面讨论的是假设 $h_1=h_2=\cdots=h_n$ 的情况，取消这一假设的情形读者自己思考，采用的优先策略和上面相似。

2.5 有期限的任务安排问题

设有 n 项任务 $J_1, J_2, \cdots, J_n$，每项都有一个完成期限 $b_1, b_2, \cdots, b_n$，任务 J_i 在这期限 b_i 内完成可获利 c_i，自然不允许不在期限内完成了。假定每个任务的加工所用的机器时间都为一个，问应如何安排加工顺序，使得收益最多单位。

这里优先的标准除考虑加工的任务的利润 c_i 的大小外还需考虑到任务期限。

例如有四个任务 J_1, J_2, J_3, J_4，完成任务的期限分别为 2，1，3，4 个单位时间，在期限内完成可分别获利润 100，150，200，250 元。若仅考虑收入的大小，顺序为 J_4, J_3, J_2, J_1。但依此顺序，J_4 加工完毕时，J_2 已过期限，只好放弃 J_2 而加工 J_3，J_3 加工完毕时，J_1 又超过期限，只好放弃，结果总收入为 450 元。如若加工顺序改为 J_2, J_1, J_3, J_4 时，则总收入可达到 100＋150＋200＋250＝700 元。

现假定任务 $J_1, J_2, \cdots, J_n$ 满足

$$c_1 \geqslant c_2 \geqslant \cdots \geqslant c_n$$

安排的基本思想是按照 $b_1,b_2,\cdots,b_n$ 依序对其进行调整，排在后面的或提前或排除。现在对其采取的优先策略作非形式化叙述如下：

假设已安排了前面 s 个，即

$$J_{r_1},J_{r_2},\cdots,J_{r_s} \tag{2.5.1}$$

为最优安排的前 s 个；则应满足

$$b_{r_i}\leqslant i \qquad i=1,2,\cdots,s$$

即已安排的任务都能在规定期限内完成。

对后面等待安排的任务 J_k 来说，与之对应的 c_k 无疑满足 $c_k\leqslant c_{r_i}$，$i=1,2,\cdots,s$。如若 $b_k>s$，即任务 J_k 的利润比序列(2.5.1)中所有的都少，期限又长，则 J_k 可直接加入到序列(2.5.1)的后面。如若 $b_k\leqslant s$，即任务 J_k 期限较急，J_k 是否能插入到序列(2.5.1)中去？插入到什么地方？就得看序列(2.5.1)是否存在允许 J_k 插入的可能了。

首先要和 J_{r_s} 进行比较，若 $b_{r_s}=s$，则 J_k 只好被删除了。若 $b_{r_s}>s$，$b_k=s$，说明 J_{r_s} 这项任务可以往后推，则 J_k 置于 J_{r_s} 的位置，将 J_{r_s} 后移一位即可。若 $b_{r_s}>s$ 且 $b_k<s$，则将 J_k 和 $J_{r_{s-1}}$ 比较，方法同上。推而广之，若存在 J_{r_j}，$1\leqslant j\leqslant s$，满足 $b_{r_j}=j$，但 $b_k\leqslant b_{r_j}$，由于 $c_k<c_{r_j}$，故 J_k 也只好放弃了。

上述过程对 n 项任务都作出判断，最后得到的即为最佳的任务安排。正确性读者可自己思考。下面给出该算法。

(1) 初始化：对 $c_i,i=1,2,\cdots,n$ 进行排序，不妨假设满足：$c_1\geqslant c_2\geqslant\cdots\geqslant c_n$。

$$r(0)\leftarrow 0,\ b(0)\leftarrow 0,r(1)\leftarrow 1,\ k\leftarrow 1,i\leftarrow 2$$

(2) $s\leftarrow k$

(3) 若 $b(r(s))\geqslant b(i)$，则转(4)。否则，转(6)。

(4) 若 $b(r(s))\neq s$，则作【$s\leftarrow s-1$，转(3)】否则，作【若 $b(r(s))\geqslant b(i)$，则转(7)；否则，$l\leftarrow k$，转(5)】。

(5) 若 $l>s$，则作【$r(l+1)\leftarrow r(l)$，$l\leftarrow l-1$，转(5)】。否则，转(6)。

(6) $k\leftarrow k+1$，$r(s+1)\leftarrow i$。

(7) $i\leftarrow i+1$。

(8) 若 $i\leqslant n$ 则转(2)，否则，输出结果，停止。

算法中 k 和 i 都是用来记数，k 用来记录已安排的任务数目，i 是搜索到的任务数目，$r(i)$ 为最佳安排中的第 i 个任务。

该算法的时间复杂度为 $O(n^2)$。若对 $n!$ 种可能性使用穷举搜索寻找，则时间复杂度将是 n 的指数函数。

习　　题

1. 已知 n 个任务 $J_1,J_2,\cdots,J_n$。对应于每一 J_i 有完成所需的时间 t_i，限期 b_i，以及利润 c_i，$i=1,2,\cdots,n$，试讨论它的最佳安排。

2. 对于背包问题，求

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

满足约束条件：

$$\sum_{i=1}^{n} a_i x_i \leqslant b$$

$$x_i = 0 \text{ 或 } 1, 1 \leqslant i \leqslant n$$

若采取依 c_i/a_i 的大小作为优先策略，试讨论这样的策略的结果如何？

3. 证明 2.4 节中(2)的结论，即文件 $f_1, f_2, \cdots, f_n$，若 $l_1 = l_2 = \cdots = l_n$，其中 l_i 是文件 f_i 的长度，$h_1 \geqslant h_2 \geqslant \cdots \geqslant h_n$，$h_i$ 是 f_i 的查找频率，则序列 $f_1, f_2, \cdots, f_n$ 使

$$S = (h_1 + 2h_2 + \cdots + nh_n)l$$

达到最小。

4. 试证 2.5 节给出的背包问题

$$\max z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$
$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leqslant b$$
$$0 \leqslant x_i \leqslant 1, c_i > 0, a_i > 0$$
$$i = 1, 2, \cdots, n$$

的解是最优的。

5.（事件选择问题）

n 个事件需共享同一资源，而这一资源同一时刻只能被一个事件所使用，每个事件有一起始时间和终止时间。如何恰当地选择事件可以使最多的事件都能使用资源称为事件选择问题。试采用优先策略设计出寻找最优解的方法，并证明之。

6. 对于习题 5 中的问题，假若资源足够多，现在希望能够将所有的事件安排到尽可能少的资源上，试设计一个有效的算法来确定哪一个事件使用哪一个资源。

7. 在实数轴上放着 n 个点 $\{x_1, x_2, \cdots, x_n\}$，现在要求找出最少量的单位长线段来包含这 n 个点，试设计你的方案。

8. 考虑将 n 美分钱使用时的找钱问题，要求使用的硬币最少。

① 设计一个算法使得找钱时使用 25 美分，10 美分，5 美分和 1 美分。证明你的算法生成最优解。

② 假定可利用的硬币面值在 $c_0, c_1, \cdots, c_k$ 中，其中 $c_i > 1, i = 1, 2, \cdots, k, k \geqslant 1$，所有的 c_i 都是正整数，使生成的结果都是最优解。

9. (a) 若 e 是图 $G = (V, E)$ 的最短边，试证 e 必属于 G 的任一最短树。

(b) 若 e 是 G 的某一回路上最长边，试证 $G' = (V, E \setminus \{e\})$ 必有一最短树同时也是 G 的最短树。

10. A 是图 $G = (V, E)$ 的边的子集，且 A 将所有属于 V 的点联结起来，而且长度最短，试证 A 是树。

11. 若 T 和 T' 都是图 G 的最短树。试证属于 T 和 T' 的边，按其长度从小到大排列的序列是相同的。

12. 若 $G = (V, E)$ 的边长是从 1 到 $|V|$ 间的正整数，试讨论用 Kruskal 及 Prim 算法

求最短树算法的复杂性。

13. 对于 $G=(V,E)$图的边，权有负数时 Dijkstra 算法的正确性如何？

已知 G 的距离矩阵，

$$D=\begin{array}{cc}\begin{pmatrix} 0 & \infty & \infty & \infty & 9 & \infty \\ 1 & 0 & \infty & 12 & \infty & \infty \\ \infty & 12 & 0 & \infty & \infty & 2 \\ 6 & \infty & \infty & 0 & 13 & \infty \\ \infty & 17 & \infty & \infty & 0 & \infty \\ \infty & 15 & 20 & \infty & \infty & 0 \end{pmatrix} & \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \\ \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} & \end{array}$$

求两点间的最短路径。

14. $G=(V,E)$是一有向图，每条边(u,v)都对应一实数 $r(u,v)$，$0\leqslant r(u,v)\leqslant 1$，$r(u,v)$表示由 u 到 v 信道不出故障的概率。假定这些概率都是独立的，试设计一算法求两个已知点间最可靠位道的算法。并讨论其复杂性。

15. 7 项有期限的任务 $J_1,J_2,\cdots,J_7$，它们的利润依次为 7,6,5,4,3,2,1，它们的期限依次为

$$4,2,4,3,1,4,6$$

单位，试找它们的最佳加工方案。

16. 1.2 的例 5 中，若 $n=12$，试用最佳原理作出判定策略。

第3章 分治策略

分治策略是一种用得最多的一种有效方法，它的基本思想将问题分解成若干子问题，然后求解子问题。子问题较原问题无疑是要求容易些，由此得出原问题的解，就是所谓的“分而治之”的意思。分治策略还可以递归进行，即子问题仍然可以用分治策略来处理，最后的问题是非常基本而简单。

3.1 二分查找

这是分治策略的典型例子。

设有一组有序的 n 个数：

$$a_1 < a_2 < \cdots < a_n$$

现给定一数 Z，要查找 Z 是否在这序列中，若 Z 在序列中，则把它找出来，即给出它的位置，若不在也给出不在的信息。首先较易想到的是逐个比较的办法，算法如下：

(1) 初始化，$i \leftarrow 1$。

(2) 若 $a_i = Z$，则作【$k \leftarrow i$，转(4)】。否则作【$i \leftarrow i+1$，转(3)】。

(3) 若 $Z < a_{i+1}$ 或 $i > n$，则作【$k \leftarrow 0$，转(4)】。否则转(2)。

(4) 打印 k，停止。

若 $k=0$ 表示 Z 不在序列中，否则 k 给出的是 Z 在序列中的位置。

逐个查找的算法分析如下：

(1) 最好的情况：一次找到，即 $Z = a_1$。

(2) 最坏的情况：查 n 次才能找到或作出不存在的判断，即 $Z > a_n$。

(3) 若每个元素查找的机会均等，查找的平均次数为

$$m = \frac{1}{n}(1 + 2 + \cdots + n) = \frac{1}{2}(n + 1)$$

下面将讨论运用分治策略查找的另一种方法，即所谓的二分查找法，基本思想是将 n 个元素分成两半，取 $a_{\lfloor n/2 \rfloor}$ 与 Z 作比较，$\left(\left\lfloor \frac{n}{2} \right\rfloor\right.$表示不超过$\frac{n}{2}$的最大整数$\left.\right)$，若 $Z < a_{\lfloor n/2 \rfloor}$，则若 Z 在序列中，它一定存在子序列 $a_1, a_2, \cdots, a_{\lfloor n/2 \rfloor - 1}$ 中，同样若 $Z > a_{\lfloor n/2 \rfloor}$，则 Z 可能在序列 $a_{\lfloor n/2 \rfloor + 1}, a_{\lfloor n/2 \rfloor + 2}, \cdots, a_n$ 中，若 $Z = a_{\lfloor n/2 \rfloor}$，则一次找到。

算法如下：

(1) $i \leftarrow 1$，$j \leftarrow n$。

(2) $k \leftarrow \left\lfloor \frac{i+j}{2} \right\rfloor$，$y \leftarrow a(k)$。

(3) 若 $y = z$，则转(6)。否则，转(4)。

(4) 若 $i = j$，则作【$k = 0$，转(6)】。否则，转(5)。

(5) 若 $Z<y$,则作【$j \leftarrow k-1$,转(2)】。否则作【$i \leftarrow k+1$,转(2)】。

(6) 打印 k,停止。

同上,$k=0$ 是 Z 不在序列中的标志,否则给出 Z 在序列中的位置。

算法的复杂性分析:

(1) 最好的情况为一次查找到,即 $Z=a_{\lfloor n/2 \rfloor}$。

(2) 最坏的情况,设 $n=2^l$,则要查找 $l+1$ 次,即$(\log_2 n)+1$ 次。

从而可知二分查找的最坏情况比逐个查找的平均数少的多。它的复杂度前者为 $O(n)$,后者为 $O(\log_2 n)$。

3.2 整数乘法

两个 N 位数 A、B 的乘积,按正常的乘法要作 N^2 次一位数的乘法。为下面讨论简单起见,设 $N=2^n$,将 A、B 分解为两部分:

$$A = a_1 10^{N/2} + a_2$$

$$B = b_1 10^{N/2} + b_2$$

则

$$AB = (a_1 10^{N/2} + a_2)(b_1 10^{N/2} + b_2)$$

$$= a_1 b_1 10^N + (a_1 b_2 + a_2 b_1) 10^{N/2} + a_2 b_2$$

这里 a_1、a_2、b_1、b_2 都是 2^{n-1} 位数。

从这个公式可知:为了计算 AB,要作四次两个 $N/2$ 位数的乘积,即,a_1b_1、a_1b_2、a_2b_1、a_2b_2,此外还要作移位和加法。由于乘法用的时间比加法和移位所花的时间多,故只估计乘法次数。

设 M_n 表示两个 2^n 位数相乘所需作的一位数乘法的数量,则有,

$$M_n = 4M_{n-1}, \qquad M_0 = 1$$

令

$$G(x) = M_0 + M_1 x + M_2 x^2 + \cdots$$

$$G(x) = \frac{1}{1-4x} = 1 + 4x + (4x)^2 + \cdots$$

$$M_n = 4^n = 2^{2n}$$

故

$$M_n = N^2$$

说明上面一种分解乘法好处不大。都要作 N^2 次一位数的乘法。但 N^2 次一位数相乘并非天经地义,请看下面的算法:

设

$$p = (a_1 + a_2)(b_1 + b_2)$$

$$q = a_1 b_1$$

$$r = a_2 b_2$$

则有

$$AB = q10^N + (p - q - r)10^{N/2} + r$$

而上式计算 AB 只作了 3 次 $N/2$ 位数乘法和若干次加法及移位,与上面一样,若只考虑一位乘法运算的次数,设 T_n 为 $N=2^n$ 时所作的一位乘法的运算次数,则有

$$T_n = 3T_{n-1}, \qquad T_0 = 1$$

令 $$G(x) = T_0 + T_1x + T_2x^2 + \cdots$$

$$G(x) = \frac{1}{1-3x} = 1 + 3x + (3x)^2 + \cdots$$

$$T_n = 3^n$$

由于 $$N = 2^n$$

所以 $$T_n = O(N^{\log_2 3})$$

说明两个 N 位数相乘，并非一定要作 N^2 次一位数相乘。

例 $A=2348, B=3825$

$$a_1 = 23, a_2 = 48, b_1 = 38, b_2 = 25$$

$$p = (a_1 + a_2)(b_1 + b_2) = 71 \times 63 = 4473$$

$$q = a_1b_1 = 874, \qquad r = a_2b_2 = 1200$$

$$\begin{aligned} AB &= 8740000 + (4473 - 874 - 1200) \times 100 + 1200 \\ &= 8740000 + 239900 + 1200 \\ &= 8981100 \end{aligned}$$

上述算法还可以应用于 $2n$ 位 2 进制数的乘法。例如，u 和 v 都是 $2n$ 位的 2 进制数。

$$u = (u_{2n-1}u_{2n-2}\cdots u_1u_0)_2,$$

$$v = (v_{2n-1}v_{2n-2}\cdots v_1v_0)_2,$$

令

$$\bar{u}_1 = (u_{2n-1}u_{2n-2}\cdots u_n)_2, \quad \bar{v}_1 = (v_{2n-1}v_{2n-2}\cdots v_n)_2,$$

$$\bar{u}_0 = (u_{n-1}u_{n-2}\cdots u_0)_2, \quad \bar{v}_0 = (v_{n-1}v_{n-2}\cdots v_0)_2,$$

则

$$u = \bar{u}_1 \cdot 2^n + \bar{u}_0$$

$$v = \bar{v}_1 \cdot 2^n + \bar{v}_0$$

u 和 v 直接相乘得：

$$uv = (2^{2n} + 2^n)\underbrace{\bar{u}_1\bar{v}_1}_{R_2} + 2^n\underbrace{(\bar{u}_1 - \bar{u}_0)(\bar{v}_0 - \bar{v}_1)}_{R_1} + (2^n + 1)\underbrace{\bar{u}_0\bar{v}_0}_{R_0}$$

从上式可知：乘积 uv 可通过 3 次 n 位二进制数的乘法和若干次移位和加法来实现：

R_2(移 $2n$ 位)

$+R_2$(移 n 位)

$+R_1$(移 n 位)

$+R_0$(移 n 位)

$+R_0$

算法的复杂性和前面一样，读者自已考虑。

3.3 矩阵乘积的 Strassen 算法

1. 已知两个 n 阶方阵

$$A = (a_{ij})_{n\times n} \qquad B = (b_{ij})_{n\times n}$$

$$C = AB = (c_{ij})_{n\times n}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj},\ i,j = 1,2,\cdots,n。$$

求 $C=AB$ 即对 n^2 个元素 c_{ij} 进行计算，故要作 n^3 次乘法。相当时间内没有人怀疑过是否可以用少于 n^3 次乘法来完成。其实不然，先以 $n=2$ 的矩阵乘积为例。

对于矩阵

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix},\quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

则有

$$\begin{aligned} C = AB &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \\ &= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \end{aligned}$$

共需作 8 次乘法。

2. Strassen 提出的算法如下：

令

$$\begin{aligned} P &= (a_{11} + a_{22})(b_{11} + b_{22}) \\ Q &= (a_{21} + a_{22})b_{11} \\ R &= a_{11}(b_{12} - b_{22}) \\ S &= a_{22}(b_{21} - b_{11}) \\ T &= (a_{11} + a_{12})b_{22} \\ U &= (a_{21} - a_{11})(b_{11} + b_{12}) \\ V &= (a_{12} - a_{22})(b_{21} + b_{22}) \end{aligned}$$

则

$$\begin{aligned} c_{11} &= P + S - T + V \\ c_{12} &= R + T \\ c_{21} &= Q + S \\ c_{22} &= P + R - Q + U \end{aligned}$$

乘法次数从 8 次减为 7 次。上述方法可以推广到一般的矩阵乘积的情形。

设 A 和 B 是阶为 2^n 的方阵，只要将 $a_{11},a_{12},a_{21},a_{22}$ 看作是 2^{n-1} 阶的子矩阵，结论仍然是成立的。

下面对 Strassen 算法的复杂性分析如下：

设 P_n 表示两个 2^n 阶矩阵 A,B 的 Strassen 乘法所需的乘法运算量，则有

$$P_n = 7P_{n-1},\qquad P_1 = 7$$

$$G(x) = P_1 + P_2x + P_3x^2 + \cdots$$

$$G(x) = \frac{7}{1-7x} = 7(1 + 7x + 7^2x^2 + \cdots)$$

所以 $$P_n = 7^n$$

令 $$N = 2^n, \quad 则 \quad n = \log_2 N$$

所以 $$P_n = N^{\log_2 7} \approx N^{2.81} \tag{3.3.1}$$

也就是说 Strassen 乘法只需 $N^{2.81}$次乘法运算。

设 A_n 表示两个 $N(=2^n)$阶方阵 Strassen 乘法所需加法运算量，由算法可知除 7 次 2^{n-1}阶方阵乘法需要 $7A_{n-1}$加法外，还需要作 18 次 2^{n-1}阶的两个方阵的和。则有：

$$A_n = 7A_{n-1} + 18(2^{n-1})^2; \quad A_1 = 18$$

令 $$G(x) = A_1 + A_2 x + A_3 x^2 + \cdots$$

$$\begin{aligned}
x:&\quad A_2 = 7A_1 + 18 \cdot 2^2 \\
x^2:&\quad A_3 = 7A_2 + 18 \cdot 2^4 \\
+)&\quad \cdots\cdots\cdots \\
\hline
&G(x) - 18 = 7xG(x) + 18 \cdot \frac{4x}{1-4x}
\end{aligned}$$

可以求得

$$G(x) = 6\sum_{k=0}^{\infty}(7^{k+1} - 4^{k+1})x^k$$

所以 $$\begin{aligned} A_n &= 6(7^n - 4^n) \\ &= 6(N^{\log_2 7} - N^2) < 6N^{\log_2 7} = 6N^{2.81} \end{aligned} \tag{3.3.2}$$

从式(3.3.1)和(3.3.2)可知：Strassen 乘法比常规乘法的加法和乘法运算量都有减少，即由 N^3 降至 $N^{2.81}$，即 Strassen 算法的时间复杂性为 $O(N^{2.81})$。

另一方面，分析一下所需的存储单元，即空间复杂性。通常的矩阵乘法需要存放 A，B，C 三个 $N(=2^n)$阶矩阵，共 $3N^2(=3 \cdot 2^{2n})$个存储单元，但对于 Strassen 算法，除 A，B 矩阵本身外，还得存放 P，Q，R，S，T，U，V，用 S_n 表示 Strassen 算法存贮单元量，则：

$$S_n = 7S_{n-1} + 2(2^n)^2, \quad S_1 = 15,$$

令 $$S(x) = S_1 + S_2 x + S_3 x^2 + \cdots$$

$$\begin{aligned}
x:&\quad S_2 = 7S_1 + 2 \cdot 4^2 \\
x^2:&\quad S_3 = 7S_2 + 2 \cdot 4^3 \\
+)&\quad \cdots\cdots\cdots \\
\hline
&S(x) - 15 = 7xS(x) + 2 \cdot \frac{16x}{1-4x}
\end{aligned}$$

从而可得

$$S(x) = \frac{77}{3}\sum_{k=0}^{\infty} 7^k x^k - \frac{32}{3}\sum_{k=0}^{\infty} 4^k x^k$$

所以 $$\begin{aligned} S_n &= \frac{77}{3} \cdot 7^{n-1} - \frac{32}{3} \cdot 4^{n-1} \\ &= \frac{11}{3} \cdot 7^n - \frac{8}{3} \cdot 4^n \end{aligned}$$

因为 $$n = \log_2 N, \quad 7^n = N^{\log_2 7}, \quad 4^n = N^2$$

所以 $$S_n = \frac{11}{3}N^{2.81} - \frac{8}{3}N^2 = O(N^{2.81})$$

由此得出其空间复杂性为 $O(N^{2.81})$。这要高于一般的矩阵乘法。

3. 求矩阵的逆

由 Strassen 矩阵乘法,可以推广到矩阵的求逆运算,同前,设:$N=2^n$

$$A=\begin{bmatrix}A_{11} & A_{12}\\ A_{21} & A_{22}\end{bmatrix}\qquad A^{-1}=\begin{bmatrix}C_{11} & C_{12}\\ C_{21} & C_{22}\end{bmatrix}$$

令

$$\begin{aligned}
(Q_1) &= A_{11}^{-1}\\
(Q_2) &= A_{21}(Q_1)\\
(Q_3) &= (Q_1)A_{12}\\
(Q_4) &= A_{21}(Q_3)\\
(Q_5) &= (Q_4) - A_{22}\\
(Q_6) &= (Q_5)^{-1}\\
C_{12} &= (Q_3)(Q_6)\\
C_{21} &= (Q_6)(Q_2)\\
(Q_7) &= (Q_3)C_{21}\\
C_{11} &= (Q_1) - (Q_7)\\
C_{22} &= -(Q_6)
\end{aligned}$$

同 Strassen 矩阵乘法运算一样,求逆的时间复杂性也为 $O(N^{2.81})$。

3.4 矩阵乘积的 Winograd 算法

1. Winograd 等式

考虑等式

$$x_1y_1+x_2y_2=(x_1+y_2)(x_2+y_1)-x_1x_2-y_1y_2 \tag{3.4.1}$$

当 n 为偶数,即 $n=2k$ 时,可扩展如下(Winograd 等式):

$$\begin{aligned}
\sum_{i=1}^{2k}x_iy_i &= \sum_{u=1}^{k}(x_{2u-1}+y_{2u})(x_{2u}+y_{2u-1})-\sum_{u=1}^{k}x_{2u-1}x_{2u}-\sum_{u=1}^{k}y_{2u-1}y_{2u} \qquad (3.4.2)\\
&=(x_1+y_2)(x_2+y_1)+(x_3+y_4)(x_4+y_3)+\cdots+\\
&\quad (x_{2k-1}+y_{2k})\cdot(x_{2k}+y_{2k-1})-(x_1x_2+x_3x_4+\cdots+x_{2k-1}x_{2k})\\
&\quad -(y_1y_2+y_3y_4+\cdots+y_{2k-1}y_{2k})
\end{aligned}$$

由式(3.4.2),单从求 $\sum_{i=1}^{2k}x_iy_i$ 上来看,右式要更复杂,但是,$\sum_{u=1}^{k}x_{2u-1}x_{2u}$ 和 $\sum_{u=1}^{k}y_{2u-1}y_{2u}$ 可分别预先计算,因此,在通常情况下利用式(3.4.2)可改进两个矩阵的乘法。

2. 矩阵乘积的 Winograd 算法

设 A 和 B 是两个维数分别为 $m\times n$ 和 $n\times p$ 的矩阵,运用 Winograd 等式计算 AB 的算法如下:(设 $n=2k$)

(1) 计算

$$f_i=\sum_{u=1}^{k}a_{i,2u-1}a_{i,2u},\qquad i=1,2,\cdots,m$$

(2) 计算

$$g_j = \sum_{u=1}^{k} b_{2u-1,j} b_{2u,j}, \qquad j = 1,2,\cdots,p$$

(3) 计算

$$c_{ij} = \sum_{u=1}^{k} (a_{i,2u-1} + b_{2u,j})(a_{i,2u} + b_{2u-1,j}) - f_i - g_j \qquad i = 1,2,\cdots,m,\ j = 1,2,\cdots,p$$

则有：

$$C = AB = (c_{ij})_{m \times p}$$

从第 1 步到第 3 步所需运算量 $\frac{nmp}{2} + \frac{n}{2}(m+p)$ 次乘法和 $\frac{3}{2}nmp + mp + \left(\frac{n}{2} - 1\right)(m+p)$ 次加法，而一般的方法则需要 nmp 次乘法和 $(n-1)mp$ 次加法。故使用 Winograd 等式以后，乘法次数减少将近一半，而加法次数有所上升。所以 $n=m=p$ 时 Winograd 算法的时间复杂性仍为 $O(n^3)$。

3.5 布尔矩阵的乘法问题

1. 问题的提出

图的可达矩阵问题便导致求布尔矩阵的乘法。设图 $G=(V,E)$ 是简单图，$A=(a_{ij})_{n\times n}$ 是图 G 的邻接矩阵，$V=(v_1,v_2,\cdots,v_n)$，$P=(p_{ij})_{n\times n}$ 称为可达矩阵，其中

$$p_{ij} = \begin{cases} 0, & \text{从顶点 } v_i \text{ 不存在道路通往 } v_j \\ 1, & \text{否则} \end{cases}$$

这样的矩阵 P 的元素只有 0 或 1 两种，我们称之为布尔矩阵。用符号"$\vee$"表示布尔和，"$\wedge$"表示布尔积，则可达矩阵 P 可通过邻接矩阵 A 经布尔运算得到：

$$P = \bigvee_{k=1}^{n} A^{(k)} \tag{3.5.1}$$

其中

$$A^{(1)} = A, \qquad A^{(k)} = A^{(k-1)} \wedge A$$

另外对于布尔矩阵 $A=(a_{ij})_{n\times n}$，$B=(b_{ij})_{n\times n}$，$C=(c_{ij})_{n\times n}$，$D=(d_{ij})_{n\times n}$ 有如下结论：

若 $C=A\vee B$，则 $c_{ij}=a_{ij}\vee b_{ij}$

若 $D=A\wedge B$，则 $d_{ij}=\bigvee_{k=1}^{n}(a_{ik}\wedge b_{kj})$

则有

$$P = A \wedge (I \vee A \vee A^2 \vee \cdots \vee A^{n-1})$$

但

$$I \vee A \vee A^2 \vee \cdots \vee A^{n-1} = (I \vee A)^{n-1}$$

所以

$$P = A \wedge (I \vee A)^{n-1} \tag{3.5.2}$$

通过式(3.5.1)和(3.5.2)算得可达矩阵 P 的复杂度分析留给读者来进行。总之求 P 导致对布尔矩阵求布尔乘积，下面介绍求矩阵布尔积以往叫做"三个前苏联人"的布尔矩阵乘法。

2. 布尔矩阵乘积的算法

先来看一个例子：

$$A \wedge B = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \wedge \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

从中可以看出矩阵 C 各行有:

$$(1111) = (1011) \vee (1100) \vee (0100)$$
$$(1101) = (1100) \vee (0100) \vee (1101)$$
$$(1011) = (1011)$$
$$(1110) = (1100) \vee (0100) \vee (1110)$$

即矩阵 C 的各行是由矩阵 B 诸行作逻辑和而得来的。

一般地对 $C=A\wedge B$,设 $A=(a_{ij})_{l\times m}$,$B=(b_{ik})_{m\times n}$,$C=(c_{jk})_{l\times n}$,又 B,C 可写为:

$$B = \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_m \end{pmatrix}, \qquad C = \begin{pmatrix} C_1 \\ C_2 \\ \vdots \\ C_l \end{pmatrix}$$

其中 $B_i=(b_{i1} \quad b_{i2} \quad \cdots \quad b_{in})$, $C_j=(c_{j1} \quad c_{j2} \quad \cdots \quad c_{jn})$, $i=1,2,\cdots,m$, $j=1,2,\cdots,l$。

其实 $C_j=\bigvee_{k\in K_j} B_k$, $K_j=\{k\mid a_{jk}=1\}$, $j=1,2,\cdots,l$

不妨令 $n=pq$。原"三个苏联人"的算法是把矩阵 B 的 n 行分成 p 组如下:

$$\begin{array}{llll} 1: & B_1, & B_2, & \cdots, & B_q; \\ 2: & B_{q+1}, & B_{q+2}; & \cdots, & B_{2q}; \\ & \cdots & \cdots & & \cdots \\ p: & B_{n-q+1}, & B_{n-q+2}, & \cdots, & B_n \end{array}$$

对其中每一组的 q 个行向量,可作 2^q-q-1 次布尔和得 2^q-1 种可能的结合。以第一组 $B_1,B_2,\cdots,B_q$ 为例从表 3.5.1 可知:

表 3.5.1

0 0 ⋯ 0 0 0	$\varnothing$
0 0 ⋯ 0 0 1	B_q
0 0 ⋯ 0 1 0	B_{q-1}
0 0 ⋯ 0 1 1	$B_{q-1} \vee B_q$
0 0 ⋯ 1 0 0	B_{q-2}
⋯⋯⋯⋯	⋯⋯⋯⋯
1 1 ⋯ 1 1 1	$B_1 \vee B_2 \vee \cdots \vee B_q$

表的左端是长度为 q 的 0,1 符号串,从全 0 到全 1。似乎这样做是盲目的,做了许多多余的计算,其实不然。

计算每一个行向量 C_j,只是对这 p 组作至多 $p-1$ 次的布尔和的运算。

例如 $q=4, n=16$

$$A_1=(1010 \;\vdots\; 1100 \;\vdots\; 0110 \;\vdots\; 1001)$$

则 $$C_1=(B_1 \vee B_3) \vee (B_5 \vee B_6) \vee (B_{10} \vee B_{11}) \vee (B_{13} \vee B_{16})$$

而 $B_1 \vee B_3, B_5 \vee B_6, B_{10} \vee B_{11}, B_{13} \vee B_{16}$ 分别属于 4 个不同组。

3. 算法复杂性分析

综上所述，对每一组作一切可能的布尔和，共要对 n 维向量进行 $\frac{n}{q}(2^q-q-1)$ 次布尔和。另外为取得 $C_1, C_2, \cdots, C_n$ 共需进行 $n \cdot \frac{n}{q}$ 次 n 维向量的布尔和。略去低阶部分，共需进行

$$\frac{n^2}{q}+\frac{n \cdot 2^q}{q}$$

次 n 维向量的布尔和。

若 $n=2^q$，即取 $q=\log_2 n$，则其复杂性变为 $O\left(\frac{2n^2}{\log_2 n}\right)$。

习　题

1. 利用原"三个苏联人算法"求下面的矩阵 C。

$$C=\begin{pmatrix}1&0&0&1&0&1&1&0\\0&1&1&1&0&0&1&1\\1&1&0&0&0&0&0&1\\1&0&0&0&1&1&1&1\\1&1&0&1&1&0&1&0\\0&0&1&1&1&0&0&1\\0&1&1&0&0&0&1&0\\1&1&0&1&1&0&0&0\end{pmatrix} \wedge \begin{pmatrix}0&1&0&1&1&0&1&0\\1&0&0&1&0&1&1&1\\0&1&1&0&1&1&0&0\\0&1&0&1&1&0&1&0\\1&1&0&1&1&1&1&0\\1&1&1&0&0&0&0&1\\0&0&1&1&0&1&1&0\\0&1&0&1&0&0&1&1\end{pmatrix}$$

2. 试验证 Strassen 矩阵乘法是正确的。

3. 若改变二分法为三分法，即从 $a_1<a_2<\cdots<a_n$ 序列中寻找元素 Z，方法如下，先与 $a_{\lfloor \frac{n}{3} \rfloor}$ 元素比较，若 $Z>a_{\lfloor \frac{n}{3} \rfloor}$ 则与 $a_{\lfloor \frac{2}{3}n \rfloor}$ 比较，总之使余下的序列为 $\left\lfloor \frac{n}{3} \right\rfloor$。并讨论它的复杂性。

4. 证明有 n 个内点的二分树，下列等式成立。

$$E=I+2n$$

其中 E 是叶子到根节点的路长总和，I 是内点到根节点的路长和。

5. 证明两个 n-比特的整数可以用 $O(n^{\log 3})$ 步乘起来。

6. 给出两个分治策略的算法来计算两个次数为 n 的多项式的乘积。第一个算法将多项式系数按高一半和低一半分开。第二个算法将多项式系数按它们的位置的奇偶性分开。

7. 试修改 Strassen 的矩阵相乘方法，使得当 n 不是 2 的幂时能够计算 $n \times n$ 阶矩阵。并证明算法的时间复杂度是 $O(n^{\log 7})$

8. 使用 Strassen 方法做为子过程，求 $kn\times n$ 和 $n\times kn$ 的矩阵能在多少时间内完成？反过来（乘 $n\times kn$ 和 $kn\times n$ 的矩阵）又怎样呢？

9. 使用 Strassen 方法计算

$$\begin{pmatrix}1 & 3\\ 5 & 7\end{pmatrix}\begin{pmatrix}8 & 4\\ 6 & 2\end{pmatrix}$$

10. "格雷码"(Gray Code)是一个长度为 2^n 的序列，满足：

a) 每个元素都是长度为 n 比特的串。

b) 序列中无相同元素。

c) 连续的两个元素恰好只有 1 比特的不同。例如，$n=2$ 时{00,01,11,10}。

(1) 构造 $n=3$ 时的"格雷码"。

(2) 利用分治策略设计一个算法对任意的 n 构造相应的"格雷码"。

第4章 Huffman编码、FFT算法和数据压缩

在信息高速公路的时代，大量的信息存储在数据库里，信息的传输频繁，如何压缩数据成为极为关键的问题，因而数据压缩成为当今计算机科学中十分热门的课题。数据压缩的技术技巧性极强。在这一章里仅就其中最为成功的两个范例进行介绍。一个是Huffman编码，另一个是快速傅里叶(Fourier)变换FFT。Huffman编码利用了码字出现频率差异而设计长度不等的码，在数据压缩技术中至今仍扮演着重要的角色。FFT则利用图象的傅里叶系数的特性，即高频部分的系数很快趋近于零，而将它舍弃以达到压缩数据的目的。

一般技巧如将 $AAAABBBAAABBBBCCCCCCBBB$ 写成 $4A3B3A4B6C3B$ 从而达到压缩数据的目的，这里就不讨论了，其道理是显而易见的。

4.1 Huffman编码

Huffman树是一棵二分树，问题提出源于编码和文件检索，英文26个字母出现的频率不等，比如统计表明 e 出现的频率高达0.1304，而Z却只有0.0008，如若所有的字母不论频率多少，一律平等，都有一样的码长，这样的编码效率不高。在编码时若考虑频率高的字母码长较短一些，而频率低的字母码长一些，也就是码的长度不等，这样便提出了最佳编码问题，使数字压缩到最低程度。设 n 个字符

$$a_1, a_2, \cdots, a_n$$

出现的频率为

$$p_1, p_2, \cdots, p_n$$

满足

$$p_1 + p_2 + \cdots + p_n = 1$$

用一个0,1符号串来表达它们的编码，它们的码的长度分别为

$$l_1, l_2, \cdots, l_n$$

Huffman编码对应一有 n 个叶子的二分树，例如图4.1.1是有5个叶子的二分树。每一叶子对应一个码字，本例有5个码字(00,01,10,110,111)，编码方法既然对应一个二分

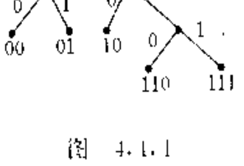


图 4.1.1

树，译码又是如何进行呢？以码文 1101000111 为例，译码的方法是从树根开始，沿着码字 0 或 1 所指的树枝走到树叶子，便算得一码字，接着再从树根开始继续搜索。比如本例不难发现它可分解为

$$110,10,00,111$$

由此可见一棵二分树对应一组编码，或编码策略。同样，一组编码方式也对应着这样一棵二分树。设计最佳编码实际上是归结为求一二分树使 $m(L)=\sum_i p_i l_i$ 达到最小，$m(L)$是码字长度的期望值或平均值。不妨假定 $p_1\leqslant p_2\leqslant\cdots\leqslant p_n$。首先要证明，若 T 是关于 $p_1\leqslant p_2\leqslant\cdots\leqslant p_n$ 的最佳二分树，则 p_1 和 p_2 可以是二分树上一对兄弟叶子，由于 p_1 是最小的数，故对应的 l_1 必须是最长的。而且不可能没有“兄弟”节点。如若不然，例如图 4.1.2 所示，可以修改使 $m(L)$减少，与最佳的假定矛盾，因(a)的 l_1 较(b)的 l_1 大。

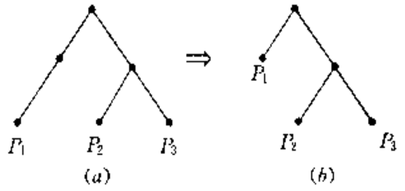


图 4.1.2

这就证明了叶子 p_1 不能没有“兄弟”节点，而且 p_1 的兄弟节点可能是 p_2。

设 T_n^* 是 $p_1\leqslant p_2\leqslant\cdots\leqslant p_n$ 的最佳二分树，p_1 和 p_2 是兄弟节点，T_{n-1}是去掉 p_1 和 p_2 两个叶子节点，并给 p_1,p_2 的父亲节点以权 p_1+p_2 的二分树(图 4.1.3)。显然有

$$m(T_{n-1})=m(T_n^*)-p_1-p_2$$

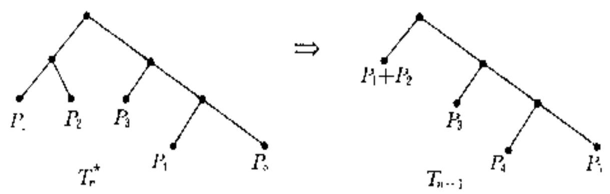


图 4.1.3

又设 T_{n-1}^*是权为 $p_1+p_2,p_3,\cdots,p_n$ 的最佳二分树，$\overline{T}_n$ 是 p_1+p_2 对应的叶子向下延伸以作为 p_1 和 p_2 的父亲节点的二分树，如图 4.1.4 所示。

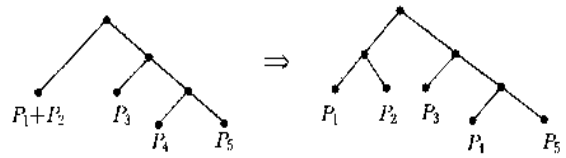


图 4.1.4

显然
$$m(T_{n-1}^*)=m(\overline{T}_n)-p_1-p_2$$

由于 T_n^* 是最佳二分树，故

$$m(T_n^*)\leqslant m(\overline{T}_n)$$

所以 $$m(\dot{T}_{n-1}) + p_1 + p_2 \leqslant m(T^*_{n-1}) + p_1 + p_2$$

即 $$m(\dot{T}_{n-1}) \leqslant m(T^*_{n-1})$$

由于 T^*_{n-1} 是 $p_1+p_2,p_3,\cdots,p_n$ 对应的最佳二分树，故

$$m(\dot{T}_{n-1}) = m(T^*_{n-1})$$

这就说明了可从 $p_1+p_2,p_3,\cdots,p_n$ 的最佳二分树 T^*_{n-1} 构造 T^*_n，$\dot{T}_{n-1}$ 本身就是关于 $p_1+p_2,p_3,\cdots,p_n$ 的最佳二分树，利用这个结论可以递推地运用来求最佳二分树 T^*_n。例如统计 8 个码字出现的次数分别为 1,3,5,8,13,20,24,26。试设计最佳的编码。

计算 Huffman 树过程见图 4.1.5。

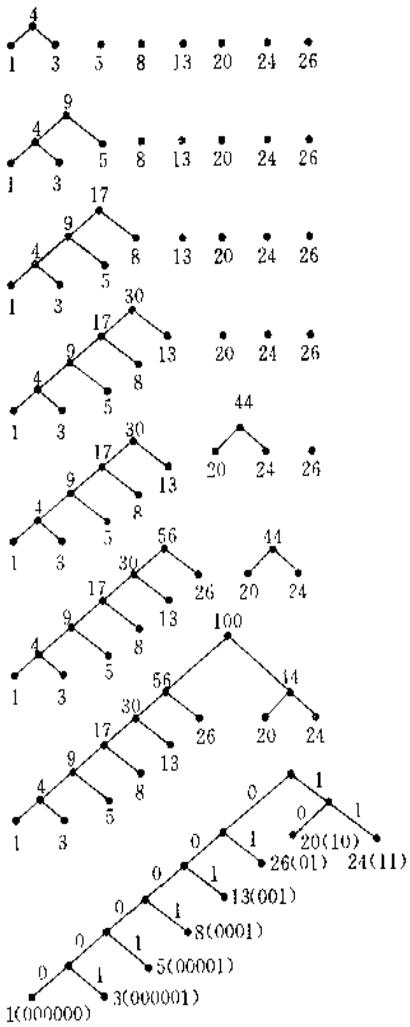


图 4.1.5

最后得一个二分树，8 个叶子分别对应于 8 个频数，括号里的数(0,1 符号串)便是它

的编码。例如频数最小的 1 和 3 的编码分别为 100000 和 100001。频数最高的 26 和 24，它们的编码分别为 01 和 11。

最后看一个实际例子，比如要对下面一段信息进行编码 *baabbbcbabbb* 共 12 个字符。若采用等长的码字，例如 a：00，b：01，c：10，则得码文 010000010101100100010101。由于 a，b，c 出现的频率不等，若采用 Huffman 编码 b：0，a：10，c：11，则同样的信息的码文是 0101000011010000，前者占了 24 位，而后者仅占 16 位。前者每一字符占两位，后者每一字符占 16/12=4/3 位。

4.2 快速傅里叶变换(FFT)

4.2.1 FFT 问题的背景

FFT 是英文 Fast Fourier Transform 的缩写，意为快速傅里叶变换。随着空间技术的发展，卫星拍摄的照片可以通过电波送回到地面，传送方法是将照片分成 $n\times m$ 个格子点，根据每格子上光的强弱变成波的强弱。要完整正确地表达一张照片，需要送回大量的数据，而且数据量大得惊人，在传输过程中还免不了受到外界的干扰而失真。为了压缩数据的需要，可将图象看作是一个二元函数，对其进行傅里叶变换，送回地面的不是照片数据本身，而是它的傅里叶系数。并且由于自然图象的特点，高频部分的系数很快接近于零可以大量略去。地面接收到的是傅里叶系数，可利用它恢复原来的图象，结果更加清晰。从照片转换为傅里叶系数，可以看作是作了一次傅里叶变换，地面上将接受到的信息还原为原来的照片，可以看作是对应的傅里叶逆变换。由于处理的数量大，而且要求能做到实时，因而算法速度要快，快速傅里叶变换 FFT 就是在这样的基础上提出的。FFT 的出现对当今科学技术的影响是深远的，不仅是数据压缩的成功范例，此外，在许多方面都有重大的影响。特别是并行算法，后面将看到 FFT 可以使许多计算同时进行，所以也是开创并行计算这个新领域的非常成功的例子。它在空间技术上的影响更是无法估量的。不论是并行算法还是数据压缩至今都是计算机科学的热门课题。在这两个方面 FFT 都是极其成功的，怎么估计都不为过分。

4.2.2 预备定理

在研究 FFT 算法之前，先给出几个重要的结论，为后面讨论作准备。

引理 若 r,m 都是整数，则

$$\sum_{k=0}^{n-1}e^{2\pi ik(r-m)/n}=\begin{cases}n, & r=m\\0, & r\neq m\end{cases}$$

其中 i 为虚数单位

证明 当 $r=m$ 时，

$$\sum_{k=0}^{n-1}e^{2\pi ik(r-m)/n}=\sum_{k=0}^{n-1}e^{0}=\sum_{k=0}^{n-1}1=n$$

显然成立。

当 $r\neq m$ 时

$$\sum_{k=0}^{n-1} e^{2\pi i k(r-m)/n} = \sum_{k=0}^{n-1} e^{2\pi i k a} = \sum_{k=0}^{n-1} \omega^k$$

其中 $\qquad \omega = e^{2\pi i a} = e^{2\pi i(r-m)/n}$

所以 $\qquad \sum_{k=0}^{n-1} e^{2\pi i k(r-m)/n} = \dfrac{1-\omega^n}{1-\omega}$

而 $\qquad \omega^n = e^{2\pi i(r-m)} = \cos 2(r-m)\pi + i\sin 2(r-m)\pi = 1$

故 $r \neq m$ 时，有

$$\sum_{k=0}^{n-1} e^{2\pi i k(r-m)/n} = 0$$

定理 若数列

$$x(0), x(1), \cdots, x(n-1) \tag{4.2.1}$$

和数列

$$X(1), X(2), \cdots, X(n-1) \tag{4.2.2}$$

满足下列关系：

$$X(k) = \frac{1}{n}\sum_{j=0}^{n-1} x(j) e^{-2\pi i k j/n} \tag{4.2.3}$$

$$k = 0, 1, 2, \cdots, n-1$$

则

$$x(j) = \sum_{k=0}^{n-1} X(k) e^{2\pi i j k/n} \tag{4.2.4}$$

$$j = 0, 1, 2, \cdots, n-1$$

证明 若将(4.2.3)看作是关于 $x(0), x(1), \cdots, x(n-1)$ 的方程组，(4.2.4)是它的解。同样若将(4.2.4)看作是关于 $x(0), x(1), \cdots, x(n-1)$ 的方程组，则(4.2.2)可以看作是它的解。

证明的办法是将(4.2.4)代入(4.2.3)的右端，看是否满足方程组(4.2.3)，若满足则成立。

$$\begin{aligned}
\frac{1}{n}\sum_{j=0}^{n-1} x(j) e^{-2\pi i k j/n} &= \frac{1}{n}\sum_{j=0}^{n-1}\left[\sum_{r=0}^{n-1} X(r) e^{2\pi i j r/n}\right] e^{-2\pi i j k/n} \\
&= \frac{1}{n}\sum_{j=0}^{n-1}\sum_{r=0}^{n-1} X(r) e^{2\pi i j(r-k)/n} \\
&= \frac{1}{n}\sum_{r=0}^{n-1} X(r)\sum_{j=0}^{n-1} e^{2\pi i j(r-k)/n} \\
&= X(k)
\end{aligned}$$

因为 $\qquad \sum_{j=0}^{n-1} e^{2\pi i j(r-k)/n} = \begin{cases} n, & r = k \\ 0, & r \neq k \end{cases}$

上述定理有着重要的意义，(4.2.3)是(4.2.4)的离散傅里叶变换，而(4.2.4)是(4.2.2)的傅里叶逆变换。(4.2.3)和(4.2.4)互为逆变换，而且形式十分相似。

由傅里叶分析可知，对于以 l 为周期的函数 $x(t)$，可展开成复数形式的傅里叶级数如下：

$$x(t)=\sum_{k=-\infty}^{\infty} c_k e^{2k\pi i t/l}$$

其中傅里叶系数

$$c_k=\frac{1}{l}\int_0^l x(\tau)e^{-i\frac{2k\pi}{l}\tau}d\tau, \tag{4.2.5}$$

$$k=0,\pm 1,\pm 2,\cdots$$

将区间$(0,l)n$等分：

$$\delta_t=\frac{l}{n},\ t_k=\frac{kl}{n},\ k=0,1,2,\cdots,n-1$$

对(4.2.5)作数值积分：

$$c_k\approx\frac{1}{l}\sum_{j=0}^{n-1}x(t_j)e^{-i2k\pi\cdot\frac{t_j}{l}}\cdot\frac{l}{n}$$

$$=\frac{1}{n}\sum_{j=0}^{n-1}x(t_j)e^{-2\pi ikj/n}$$

令

$$X(k)=\frac{1}{n}\sum_{j=0}^{n-1}x(t_j)e^{-2\pi ikj/n} \tag{4.2.6}$$

则有：$X(k)=c_k$，$\overline{X}(k)=c_{-k}$　　$k=0,\pm 1,\pm 2,\cdots$

从上我们可以看出$X(0),X(1),\cdots,X(n-1)$实际上是傅里叶系数近似积分值。根据定理从$X(0),X(1),\cdots,X(n-1)$可得：

$$x(k)=\sum_{j=0}^{n-1}X(j)e^{2\pi ikj/n}\quad k=0,1,2,\cdots,n-1 \tag{4.2.7}$$

其中　$x(k)$即为$x(t_k)$。

4.2.3　快速算法

从上面可知从$x(0),x(1),\cdots,x(n-1)$求得$X(0),X(1),\cdots,X(n-1)$和从$X(0),X(1),\cdots,X(n-1)$求得$x(0),x(1),\cdots x(n-1)$各自只要作n^2次乘法运算，$n(n-1)$次加法运算。下面我们以$n=2,4,8,16$为例，找出其计算规律，从而说明减少计算量是可能的。

1. $n=2$时，令

$$w_2=e^{\pi i}=\cos\pi+i\sin\pi=-1$$

$$x(0)=\sum_{k=0}^{1}X(k)=X(0)+X(1)$$

$$x(1)=\sum_{k=0}^{1}X(k)w_2^k=X(0)-X(1)$$

写成矩阵形式则有：

$$\begin{pmatrix}x(0)\\x(1)\end{pmatrix}=\begin{pmatrix}1&1\\1&-1\end{pmatrix}\begin{pmatrix}X(0)\\X(1)\end{pmatrix}$$

图4.2.1是其流程图的表示形式。

2. $n=2^2=4$时，令

$$w_4=e^{2\pi i/4}=e^{\frac{\pi}{2}i}=i,$$

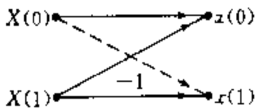


图 4.2.1

则 $w_4^0=1,\quad w_4^1=\mathrm{i},\quad w_4^2=-1,$

$w_4^3=-\mathrm{i},\quad w_4^4=1$

将其代入(4.2.7),并按 $x(0)$,$x(2)$,$x(1)$,$x(3)$的次序排列：

$$\begin{aligned}
x(0) &= X(0) + X(1) + X(2) + X(3) \\
x(2) &= X(0) + X(1)w_4^2 + X(2)w_4^4 + X(3)w_4^6 \\
&= X(0) + X(1)w_4^2 + X(2) + X(3)w_4^2 \\
x(1) &= X(0) + X(1)w_4 + X(2)w_4^2 + X(3)w_4^3 \\
x(3) &= X(0) + X(1)w_4^3 + X(2)w_4^6 + X(3)w_4^9 \\
&= X(0) + X(1)w_4^3 + X(2)w_4^2 + X(3)w_4
\end{aligned}$$

或写成矩阵形式：

$$\begin{pmatrix} x(0) \\ x(2) \\ x(1) \\ x(3) \end{pmatrix} = \left(\begin{array}{cc:cc} 1 & 1 & 1 & 1 \\ 1 & w_4^2 & 1 & w_4^2 \\ \hdashline 1 & w_4 & w_4^2 & w_4^3 \\ 1 & w_4^3 & w_4^2 & w_4 \end{array}\right) \begin{pmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{pmatrix}$$

但

$$\begin{pmatrix} 1 & w_4 \\ 1 & w_4^3 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & w_4^2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & w_4 \end{pmatrix}$$

$$\begin{pmatrix} w_4^2 & w_4^3 \\ w_4^2 & w_4 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & w_4^2 \end{pmatrix} \begin{pmatrix} -1 & 0 \\ 0 & -w_4 \end{pmatrix} = \begin{pmatrix} -1 & -i \\ -1 & i \end{pmatrix}$$

于是有

$$\begin{aligned}
&\left(\begin{array}{cc:cc} 1 & 1 & 1 & 1 \\ 1 & w_4^2 & 1 & w_4^2 \\ \hdashline 1 & w_4 & w_4^2 & w_4^3 \\ 1 & w_4^3 & w_4^2 & w_4 \end{array}\right) \\
&= \left(\begin{array}{cc:cc} 1 & 1 & 0 & 0 \\ 1 & w_4^2 & 0 & 0 \\ \hdashline 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & w_4^2 \end{array}\right) \cdot \left(\begin{array}{cc:cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ \hdashline 1 & 0 & -1 & 0 \\ 0 & w_4 & 0 & -w_4 \end{array}\right)
\end{aligned}$$

从而我们可得到：

$$\begin{pmatrix} X_1(0) \\ X_1(1) \\ X_1(2) \\ X_1(3) \end{pmatrix} = \begin{pmatrix} I_{(2)} & & I_{(2)} & \\ 1 & 0 & -1 & 0 \\ 0 & w_4 & 0 & -w_4 \end{pmatrix} \begin{pmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{pmatrix}$$

$$= \begin{pmatrix} X(0) + X(2) \\ X(1) + X(3) \\ X(0) - X(2) \\ [X(1) - X(3)]w_4 \end{pmatrix}$$

其中 $I(2)$ 表示 2 阶单位阵。

$$\begin{pmatrix} x(0) \\ x(2) \\ x(1) \\ x(3) \end{pmatrix} = \begin{pmatrix} 1 & 1 & & \\ 1 & w_4^2 & & 0 \\ & & 1 & 1 \\ & 0 & 1 & w_4^2 \end{pmatrix} \begin{pmatrix} X_1(0) \\ X_1(1) \\ X_1(2) \\ X_1(3) \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} X_1(0) \\ X_1(1) \\ X_1(2) \\ X_1(3) \end{pmatrix}$$

$$= \begin{pmatrix} X_1(0) + X_1(1) \\ X_1(0) - X_1(1) \\ X_1(2) + X_1(3) \\ X_1(2) - X_1(3) \end{pmatrix}$$

从 $X(0), X(1), X(2), X(3)$ 求 $x(0), x(1), x(2), x(3)$ 的计算过程可用流程图表示(图 4.2.2)。

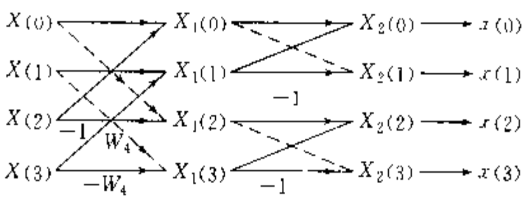


图 4.2.2

从图 4.2.2 可见 $n=2^2=4$ 时,经整理后分解成两个 $n=2$ 的快速傅里叶变换,共用 8 次加法,1 次乘法,而不是 16 次乘法,12 次加法。

3. $n=2^3$,令 $w_8=e^{2\pi i/8}=e^{\pi i/4}$

将其代入(4.2.7)并按照 $x(0), x(4), x(2), x(6), x(1), x(5), x(3), x(7)$ 的顺序排列。其中 $x(0), x(4), x(2), x(6)$ 的顺序是从 $n=4$ 时的顺序 $x(0), x(2), x(1), x(3)$ 下标乘 2 得到的。

$$x(0)=X(0)+X(1)+X(2)+X(3)+X(4)+X(5)+X(6)+X(7)$$

$$x(4)=X(0)+X(1)w_8^4+X(2)+X(3)w_8^4+X(4)+X(5)w_8^4+X(6)+X(7)w_8^4$$

$$x(2)=X(0)+X(1)w_8^2+X(2)w_8^4+X(3)w_8^6+X(4)+X(5)w_8^2+X(6)w_8^4+X(7)w_8^6$$

$$x(6)=X(0)+X(1)w_8^6+X(2)w_8^4+X(3)w_8^2+X(4)+X(5)w_8^6+X(6)w_8^4+X(7)w_8^2$$

$$x(1)=X(0)+X(1)w_8+X(2)w_8^2+X(3)w_8^3+X(4)w_8^4+X(5)w_8^5+X(6)w_8^6 \\ +X(7)w_8^7$$

$$x(5)=X(0)+X(1)w_8^5+X(2)w_8^2+X(3)w_8^7+X(4)w_8^4+X(5)w_8+X(6)w_8^6 \\ +X(7)w_8^3$$

$$x(3)=X(0)+X(1)w_8^3+X(2)w_8^6+X(3)w_8^7+X(4)w_8^4+X(5)w_8^7+X(6)w_8^2 \\ +X(7)w_8^5$$

$$x(7)=X(0)+X(1)w_8^7+X(2)w_8^6+X(3)w_8^5+X(4)w_8^4+X(5)w_8^3+X(6)w_8^2 \\ +X(7)w_8$$

写成矩阵有如下形式：

$$\begin{pmatrix} x(0)\\ x(4)\\ x(2)\\ x(6)\\ x(1)\\ x(5)\\ x(3)\\ x(7) \end{pmatrix}=\left(\begin{array}{cccc:cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\ 1 & w_8^4 & 1 & w_8^4 & 1 & w_8^4 & 1 & w_8^4\\ 1 & w_8^2 & w_8^4 & w_8^6 & 1 & w_8^2 & w_8^4 & w_8^6\\ 1 & w_8^6 & w_8^4 & w_8^2 & 1 & w_8^6 & w_8^4 & w_8^2\\ \hdashline 1 & w_8 & w_8^2 & w_8^3 & w_8^4 & w_8^5 & w_8^6 & w_8^7\\ 1 & w_8^5 & w_8^2 & w_8^7 & w_8^4 & w_8 & w_8^6 & w_8^3\\ 1 & w_8^3 & w_8^6 & w_8 & w_8^4 & w_8^7 & w_8^2 & w_8^5\\ 1 & w_8^7 & w_8^6 & w_8^5 & w_8^4 & w_8^3 & w_8^2 & w_8^1 \end{array}\right)\begin{pmatrix} X(0)\\ X(1)\\ X(2)\\ X(3)\\ X(4)\\ X(5)\\ X(6)\\ X(7) \end{pmatrix}$$

但

$$\begin{pmatrix} 1 & w_8 & w_8^2 & w_8^3\\ 1 & w_8^5 & w_8^2 & w_8^7\\ 1 & w_8^3 & w_8^6 & w_8\\ 1 & w_8^7 & w_8^6 & w_8^5 \end{pmatrix}$$

$$=\begin{pmatrix} 1 & 1 & 1 & 1\\ 1 & w_8^4 & 1 & w_8^4\\ 1 & w_8^2 & w_8^4 & w_8^6\\ 1 & w_8^6 & w_8^4 & w_8^2 \end{pmatrix}\begin{pmatrix} 1 & & & \mathbf{0}\\ & w_8 & & \\ & & w_8^2 & \\ \mathbf{0} & & & w_8^3 \end{pmatrix}$$

上式中 $1,w_8,w_8^2,w_8^3$ 分别是左端各列诸元素的公因子。同理有

$$\begin{pmatrix} w_8^4 & w_8^5 & w_8^6 & w_8^7 \\ w_8^4 & w_8 & w_8^6 & w_8^3 \\ w_8^4 & w_8^7 & w_8^2 & w_8^5 \\ w_8^4 & w_8^3 & w_4^2 & w_8^1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & w_8^4 & 1 & w_8^4 \\ 1 & w_8^2 & w_8^4 & w_8^6 \\ 1 & w_8^6 & w_8^4 & w_8^2 \end{pmatrix} \begin{pmatrix} -1 & & & \mathbf{0} \\ & -w_8 & & \\ & & -w_8^2 & \\ \mathbf{0} & & & -w_8^3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w_8^4 & 1 & w_8^4 & 1 & w_8^4 & 1 & w_8^4 \\ 1 & w_8^2 & w_8^4 & w_8^6 & 1 & w_8^2 & w_8^4 & w_8^6 \\ 1 & w_8^6 & w_8^4 & w_8^2 & 1 & w_8^6 & w_8^4 & w_8^2 \\ 1 & w_8 & w_8^2 & w_8^3 & w_8^4 & w_8^5 & w_8^6 & w_8^7 \\ 1 & w_8^5 & w_8^2 & w_8^7 & w_8^4 & w_8 & w_8^6 & w_8^3 \\ 1 & w_8^3 & w_8^6 & w_8 & w_8^4 & w_8^7 & w_8^2 & w_8^5 \\ 1 & w_8^7 & w_8^6 & w_8^5 & w_8^4 & w_8^3 & w_8^2 & w_8 \end{pmatrix}$$

$$= \left(\begin{array}{cccc:cccc} 1 & 1 & 1 & 1 & & & & \\ 1 & w_8^4 & 1 & w_8^4 & & \mathbf{0} & & \\ 1 & w_8^2 & w_8^4 & w_8^6 & & & & \\ 1 & w_8^6 & w_8^4 & w_8^2 & & & & \\ \hdashline & & & & 1 & 1 & 1 & 1 \\ & \mathbf{0} & & & 1 & w_8^4 & 1 & w_8^4 \\ & & & & 1 & w_8^2 & w_8^4 & w_8^6 \\ & & & & 1 & w_8^6 & w_8^4 & w_8^2 \end{array}\right) \left(\begin{array}{cccc:cccc} & I_{(4)} & & & & I_{(4)} & & \\ \hdashline 1 & & & \mathbf{0} & -1 & & & \mathbf{0} \\ & w_8 & & & & -w_8 & & \\ & & w_8^2 & & & & -w_8^2 & \\ \mathbf{0} & & & w_8^3 & \mathbf{0} & & & -w_8^3 \end{array}\right)$$

引进

$$\begin{pmatrix} X_1(0) \\ X_1(1) \\ X_1(2) \\ X_1(3) \\ X_1(4) \\ X_1(5) \\ X_1(6) \\ X_1(7) \end{pmatrix} = \left(\begin{array}{cccc:cccc} & I_{(4)} & & & & I_{(4)} & & \\ \hdashline 1 & & & \mathbf{0} & -1 & & & \mathbf{0} \\ & w_8 & & & & -w_8 & & \\ & & w_8^2 & & & & -w_8^2 & \\ \mathbf{0} & & & w_8^3 & \mathbf{0} & & & -w_8^3 \end{array}\right) \begin{pmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \end{pmatrix} = \begin{pmatrix} X(0)+X(4) \\ X(1)+X(5) \\ X(2)+X(6) \\ X(3)+X(7) \\ X(0)-X(4) \\ [X(1)-X(5)]w_8 \\ [X(2)-X(6)]w_8^2 \\ [X(3)-X(7)]w_8^3 \end{pmatrix}$$

则有

$$
\begin{pmatrix} x(0) \\ x(4) \\ x(2) \\ x(6) \\ x(1) \\ x(5) \\ x(3) \\ x(7) \end{pmatrix} = \left(\begin{array}{cccc:cccc}
1 & 1 & 1 & 1 & & & & \\
1 & w_8^4 & 1 & w_8^4 & & \multicolumn{2}{c}{\mathbf{0}} & \\
1 & w_8^2 & w_8^4 & w_8^6 & & & & \\
1 & w_8^6 & w_8^4 & w_8^2 & & & & \\
\hdashline
 & & & & 1 & 1 & 1 & 1 \\
 & \multicolumn{2}{c}{\mathbf{0}} & & 1 & w_8^4 & 1 & w_8^4 \\
 & & & & 1 & w_8^2 & w_8^4 & w_8^6 \\
 & & & & 1 & w_8^6 & w_8^4 & w_8^2
\end{array}\right) \begin{pmatrix} X_1(0) \\ X_1(1) \\ X_1(2) \\ X_1(3) \\ X_1(4) \\ X_1(5) \\ X_1(6) \\ X_1(7) \end{pmatrix}
$$

由于 $w_8^2 = e^{\frac{\pi}{2}i} = w_4$ 故

$$
\begin{pmatrix}
1 & 1 & 1 & 1 \\
1 & w_8^4 & 1 & w_8^4 \\
1 & w_8^2 & w_8^4 & w_8^6 \\
1 & w_8^6 & w_8^4 & w_8^2
\end{pmatrix} = \begin{pmatrix}
1 & 1 & 1 & 1 \\
1 & w_4^2 & 1 & w_4^2 \\
1 & w_4 & w_4^2 & w_4^3 \\
1 & w_4^3 & w_4^2 & w_4
\end{pmatrix}
$$

所以

$$
\begin{pmatrix} x(0) \\ x(4) \\ x(2) \\ x(6) \\ x(1) \\ x(5) \\ x(3) \\ x(7) \end{pmatrix} = \begin{pmatrix}
\begin{pmatrix}
1 & 1 & 1 & 1 \\
1 & w_4^2 & 1 & w_4^2 \\
1 & w_4 & w_4^2 & w_4^3 \\
1 & w_4^3 & w_4^2 & w_4
\end{pmatrix} \begin{pmatrix} X_1(0) \\ X_1(1) \\ X_1(2) \\ X_1(3) \end{pmatrix} \\
\begin{pmatrix}
1 & 1 & 1 & 1 \\
1 & w_4^2 & 1 & w_4^2 \\
1 & w_4 & w_4^2 & w_4^3 \\
1 & w_4^3 & w_4^2 & w_4
\end{pmatrix} \begin{pmatrix} X_1(4) \\ X_1(5) \\ X_1(6) \\ X_1(7) \end{pmatrix}
\end{pmatrix}
$$

从 $n=4$ 的情形可知 $n=8$ 的傅里叶变换可以分解为对 $X_1(0)$，$X_1(1)$，$X_1(2)$，$X_1(3)$，$X_1(4)$，$X_1(5)$，$X_1(6)$，$X_1(7)$的两次 $n=4$ 的傅里叶变换。令

$$
\begin{pmatrix} X_2(0) \\ X_2(1) \\ X_2(2) \\ X_2(3) \\ X_2(4) \\ X_2(5) \\ X_2(6) \\ X_2(7) \end{pmatrix} = \begin{pmatrix}
X_1(0) + X_1(2) \\
X_1(1) + X_1(3) \\
X_1(0) - X_1(2) \\
[X_1(1) - X_1(3)]w_4 \\
X_1(4) + X_1(6) \\
X_1(5) + X_1(7) \\
X_1(4) - X_1(6) \\
[X_1(5) - X_1(7)]w_4
\end{pmatrix}
$$

和

$$
\begin{pmatrix} x_3(0) \\ x_3(1) \\ x_3(2) \\ x_3(3) \\ x_3(4) \\ x_3(5) \\ x_3(6) \\ x_3(7) \end{pmatrix}
=
\begin{pmatrix} x_2(0)+x_2(1) \\ x_2(0)-x_2(1) \\ x_2(2)+x_2(3) \\ x_2(2)-x_2(3) \\ x_2(4)+x_2(5) \\ x_2(4)-x_2(5) \\ x_2(6)+x_2(7) \\ x_2(6)-x_2(7) \end{pmatrix}
=
\begin{pmatrix} x(0) \\ x(4) \\ x(2) \\ x(6) \\ x(1) \\ x(5) \\ x(3) \\ x(7) \end{pmatrix}
$$

计算结果列于图 4.2.3 上。

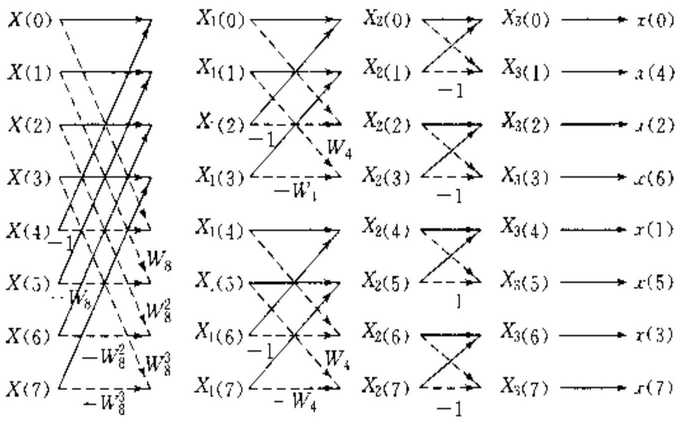


图 4.2.3

从图 4.2.3 可见 $n=8$ 时共用 5 次乘法、24 次加法，而不是 64 次乘法和 56 次加法。

类似以上的过程可以推得 $n=2^4=16$ 的快速傅里叶变换，其算法过程可用图 4.2.4 形象地表示出来。并且可以 $n=2^k$ 的算法递推出 $n=2^{k+1}$ 的算法。

4.2.4 傅里叶逆变换

以上给出的是以 $X(0)$、$X(1)$、…、$X(n-1)$ 到 $x(0)$、$x(1)$、…、$x(n-1)$ 的计算方法，也就是快速傅里叶变换。至于其逆变换，即从 $x(0),x(1),\cdots,x(n-1)$ 到 $X(0),X(1),\cdots,X(n-1)$，从式(4.2.3)和(4.2.4)可知有许多相似之处。所不同的是在流程图中用 w_n^{-1} 代替 w_n，比如 w_4 改为 w_4^{-1}、w_8 改为 w_8^{-1} 等等，最后结果除以 n。

4.2.5 计算结果的重排

FFT 算法的最后一个步骤是对计算结果重新排序。以 $n=16$ 为例，最后的结果并非是 $x(0),x(1),x(2),\cdots,x(n-1)$，而是 $x(0),x(8),x(4),x(12),x(2),x(10),x(6),x(14),x(1),x(9),x(5),x(13),x(3),x(11),x(7),x(15)$，但这个顺序是有规律可循的，如

$n=2$：$x(0),x(1)$

$n=4$：$x(0),x(2),x(1),x(3)$

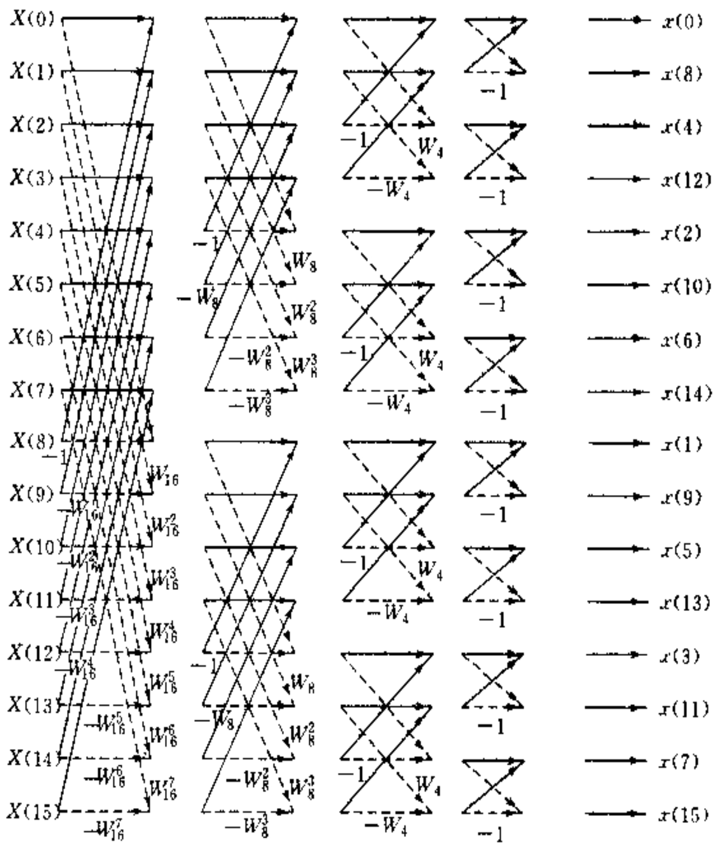


图 4.2

$n=8$: $x(0),x(4),x(2),x(6),x(1),x(5),x(3),x(7)$

$n=16$: $x(0),x(8),x(4),x(12),x(2),x(10),x(6),x(14)$

$x(1),x(9),x(5),x(13),x(3),x(11),x(7),x(15)$

下面我们给出两种排序的方法,原理留给读者思考。

1. 对于 0 到 2^n-1 的数 j 可依顺序表以 n 位二进制数:

$$j = j_1 j_2 \cdots j_n$$

则第 $j_1 j_2 \cdots j_n$ 个数的下标为:

$$j_n j_{n-1} \cdots j_1$$

以 $n=2^3$ 为例,列于表 4.2.1。

表 4.2.1

j / $j_1j_2j_3$	0	1	2	3	4	5	6	7
$j_1j_2j_3$	000	001	010	011	100	101	110	111
$j_3j_2j_1$	000	100	010	110	001	101	011	111
j	0	4	2	6	1	5	3	7

2. 假定 $N=2^n$

第 1 步：令 $n_s=\dfrac{N}{2^s}$，$s=1,2,\cdots,n$

第 2 步：作下列表格：

0

n_1

$n_2\quad n_1+n_2$

$n_3\quad n_1+n_3\quad n_2+n_3\quad n_1+n_2+n_3$

$n_4\quad n_1+n_4\quad n_2+n_4\quad n_1+n_2+n_4\quad n_3+n_4\quad n_1+n_3+n_4\quad n_2+n_3+n_4\quad n_1+n_2+n_3+n_4$

规律是 $0,n_1,n_2,n_3,n_4,\cdots$ 排在第一列，n_k 所在的行的元素依次是 n_k 和 $n_1,n_2,\cdots,n_{k-1}$ 行元素的和。

第 3 步：下列读数便是所求的顺序

$0,\quad n_1,\quad n_2,\quad n_1+n_2,\quad n_3,\quad n_1+n_3,\quad n_2+n_3,\quad n_1+n_2+n_3,$

$n_4,n_1+n_4,n_2+n_4,n_1+n_2+n_4,n_3+n_4,n_1+n_3+n_4,n_2+n_3+n_4,n_1+n_2+n_3+n_4,$

$\cdots$

例如　$N=2^4=16$ 时

0

8

4　12

2　10　6　14

1　9　5　13　3　11　7　15

故 0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,15 便是所求的顺序。

4.2.6　复杂性估计

综合以上所述的方法可见：$N=2^n$ 的 FFT 算法分解成两个 $N=2^{n-1}$ 的 FFT。

1. 令 M_n 为 $N=2^n$ 的 FFT 所需的乘法运算量，则有：

$$M_n=2M_{n-1}+2^{n-1}-1,\quad M_1=0$$

令
$$G_1(x)=M_1+M_2x+M_3x^2+\cdots,$$

$$x:\ M_2=2M_1+2-1$$

$$x^2:\ M_3=2M_2+2^2-1$$

$$+\cdots\cdots\cdots\cdots$$

$$G_1(x)-1=2xG_1(x)+\frac{2x}{1-2x}-\frac{x}{1-x}$$

$$(1-2x)G_1(x)=\frac{x}{(1-2x)(1-x)}$$

所以
$$G_1(x)=\frac{x}{(1-x)(1-2x)^2}=\frac{A}{1-x}+\frac{B}{1-2x}+\frac{C}{(1-2x)^2}$$

$$A(1-2x)^2+B(1-x)(1-2x)+C(1-x)=x$$

$$\begin{cases} A + B + C = 0 \\ 2A + B = 0 \\ 4A + 3B + C = -1 \end{cases}$$

$$A = C = 1, \quad B = -2$$

$$G_1(x) = \frac{1}{1-x} + \frac{1}{(1-2x)^2} + \frac{1}{1-2x}$$

$$= (1 + x + x^2 + \cdots) - 2(1 + 2x + 2^2x^2 + \cdots)$$

$$+ (1 + 2x + 2^2x^2 + \cdots)(1 + 2x + 2^2x^2 + \cdots)$$

所以
$$G_1(x) = \sum_{k=0}^{\infty}[1 - 2^{k+1} + 2^k(k+1)]x^k$$

$$M_n = 1 - 2^n + n2^{n-1}$$

例如
$$M_1 = 0, M_2 = 1, M_3 = 5, M_4 = 17$$

但
$$N = 2^n, \ n = \log_2 N$$

所以
$$M_n = 1 - N + \frac{1}{2}N\log_2 N$$

2. 令 A_n 表示 $N=2^n$ 的 FFT 所需的加法运算量。

$$A_n = 2A_{n-1} + 2^n, \quad A_1 = 2$$

令
$$G_2(x) = A_1 + A_2x + A_3x^2 + \cdots$$

$$\begin{array}{ll} x: & A_2 = 2A_1 + 2^2 \\ x^2: & A_3 = 2A_2 + 2^3 \\ + & \cdots\cdots\cdots\cdots \\ \hline \end{array}$$

$$G_2(x) - 2 = 2xG_2(x) + \frac{4x}{1-2x}$$

$$(1 - 2x)G_2(x) = \frac{2}{1-2x}$$

$$G_2(x) = \frac{2}{(1-2x)^2} = 2(1 + 2x + 2^2x^2 + \cdots)(1 + 2x + 2^2x^2 + \cdots)$$

$$A_n = n2^n = N\log_2 N$$

综合 1. 和 2. 可知 FFT 算法的时间复杂性为

$$O(N\log_2 N)$$

3. 空间复杂性估计

考虑到复数占用 2 个单元，故 FFT 只需要 $4N$ 个存储单元。直接利用数值积分法则需要 $2N^2$ 个存储单元。

4. 从 FFT 流程图可见，FFT 便于进行并行计算，利用 N 个处理器同时进行，速度可以大大提高，时间复杂性从 $O(N\log_2 N)$ 降至 $O(\log_2 N)$。

4.3 卷积及其应用

4.3.1 卷积

为表示方便起见，令 $x_i=x(i)$，$X_k=X(k)$

$$\boldsymbol{x}=(x_0\ x_1\cdots x_{n-1})^{\mathrm{T}},\ \boldsymbol{X}=(X_0\ X_1\cdots X_{n-1})^{\mathrm{T}}$$

并用

$$F\{\boldsymbol{x}\}=\boldsymbol{X}$$

表示 $\boldsymbol{X}$ 是 $\boldsymbol{x}$ 的离散傅里叶变换，$\boldsymbol{x}=F^{-1}\{\boldsymbol{X}\}$表 $\boldsymbol{x}$ 是 $\boldsymbol{X}$ 的离散傅里叶逆变换。

设
$$\boldsymbol{x}_1=(x_0^{(1)}\quad x_1^{(1)}\quad\cdots\quad x_{n-1}^{(1)})^{\mathrm{T}}$$
$$\boldsymbol{x}_2=(x_0^{(2)}\quad x_1^{(2)}\quad\cdots\quad x_{n-1}^{(2)})^{\mathrm{T}}$$

定义 令

$$x_k=\sum_{i=0}^{n-1}x_i^{(1)}x_{k-i}^{(2)},\quad k=0,1,2,\cdots,n-1$$
$$\boldsymbol{x}=(x_0\ x_1\cdots x_{n-1})^{\mathrm{T}}$$

称 $\boldsymbol{x}$ 为 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$ 的卷积，记作 $\boldsymbol{x}=\boldsymbol{x}_1*\boldsymbol{x}_2$。

设
$$x_{k+ln}^{(i)}=x_k^{(i)},\quad l=0,\pm1,\pm2,\cdots,i=1,2$$

则可表示为：

$$\begin{aligned}
x_0&=x_0^{(1)}x_0^{(2)}+x_1^{(1)}x_{-1}^{(2)}+x_2^{(1)}x_{-2}^{(1)}+\cdots+x_{n-1}^{(1)}x_{-n+1}^{(2)}\\
&=x_0^{(1)}x_0^{(2)}+x_1^{(1)}x_{n-1}^{(2)}+\cdots+x_{n-1}^{(1)}x_1^{(2)}\\
x_1&=x_0^{(1)}x_1^{(2)}+x_1^{(1)}x_0^{(2)}+\cdots+x_{n-1}^{(1)}x_{-n+2}^{(2)}\\
&=x_0^{(1)}x_1^{(2)}+x_1^{(1)}x_0^{(2)}+\cdots+x_{n-1}^{(1)}x_2^{(2)}\\
&\cdots\cdots\cdots\cdots\\
x_{n-1}&=x_0^{(1)}x_{n-1}^{(2)}+x_1^{(1)}x_{n-2}^{(2)}+\cdots+x_{n-1}^{(1)}x_0^{(2)}
\end{aligned}$$

或用矩阵形式表示如下：

$$\begin{pmatrix}x_0\\x_1\\\vdots\\x_{n-1}\end{pmatrix}=\begin{pmatrix}x_0^{(2)}&x_{n-1}^{(2)}&x_{n-2}^{(2)}&\cdots&x_1^{(2)}\\x_1^{(2)}&x_0^{(2)}&x_{n-1}^{(2)}&\cdots&x_2^{(2)}\\\cdots&\cdots&\cdots&\cdots&\cdots\\x_{n-1}^{(2)}&x_{n-2}^{(2)}&x_{n-3}^{(2)}&\cdots&x_0^{(2)}\end{pmatrix}\begin{pmatrix}x_0^{(1)}\\x_1^{(1)}\\\vdots\\x_{n-1}^{(1)}\end{pmatrix}$$

对 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$ 分别作快速傅里叶变换得：

$$X_j^{(1)}=\frac{1}{n}\sum_{k=0}^{n-1}x_k^{(1)}\mathrm{e}^{-i2\pi kj/n},\ j=0,1,2,\cdots,n-1$$

$$X_j^{(2)}=\frac{1}{n}\sum_{k=0}^{n-1}x_k^{(2)}\mathrm{e}^{-i2\pi kj/n},\ j=0,1,2,\cdots,n-1$$

定理 $\frac{1}{n}F\{\boldsymbol{x}_1*\boldsymbol{x}_2\}=F\{\boldsymbol{x}_1\}\cdot F\{\boldsymbol{x}_2\}$

即

$$\frac{1}{n}F\{\boldsymbol{x}_1 * \boldsymbol{x}_2\} = (X_0^{(1)}X_0^{(2)} \quad X_1^{(1)}X_2^{(2)} \quad \cdots \quad X_{n-1}^{(1)}X_{n-1}^{(2)})^{\mathrm{T}}$$

证明 令

$$Z = F\{\boldsymbol{x}_1\} \cdot F\{\boldsymbol{x}_2\} = (X_0^{(1)}X_0^{(2)} \quad X_1^{(1)}X_1^{(2)} \quad \cdots \quad X_{n-1}^{(1)}X_{n-1}^{(2)})^{\mathrm{T}}$$
$$= (Z_0 \quad Z_1 \quad Z_2 \quad \cdots \quad Z_{n-1})^{\mathrm{T}}$$

现求 Z 的傅里叶逆变换。

$$\sum_{j=0}^{n-1} Z_j \mathrm{e}^{\mathrm{i}2\pi kj/2} = \sum_{j=0}^{n-1} X_j^{(1)}X_j^{(2)}\mathrm{e}^{\mathrm{i}2\pi kj/n}$$
$$= \sum_{j=0}^{n-1}\left(\frac{1}{n}\sum_{l=0}^{n-1}x^1(l)\mathrm{e}^{-\mathrm{i}2\pi lj/n} - \frac{1}{n}\sum_{m=0}^{n-1}x^{(2)}(m)\mathrm{e}^{-\mathrm{i}2\pi mj/n}\right)\mathrm{e}^{\mathrm{i}2\pi kj/n}$$
$$= \sum_{l=0}^{n-1}\sum_{m=0}^{n-1}\left(x^{(1)}(l)x^{(2)}(m)\frac{1}{n^2}\sum_{j=0}^{n-1}\mathrm{e}^{-\mathrm{i}2\pi(l+m-k)j/n}\right)$$

但
$$\sum_{j=0}^{n-1}\mathrm{e}^{\mathrm{i}2\pi(l+m-k)/2} = \begin{cases} n, & l+m=k \\ 0, & \text{其他} \end{cases}$$

故
$$\sum_{j=0}^{n-1} Z_j \mathrm{e}^{\mathrm{i}2\pi kj/n} = \frac{1}{n}\sum_{l=0}^{n-1}x^{(1)}(l)x^{(2)}(k-l)$$

这就证明了

$$F\{\boldsymbol{x}_1\} \cdot F\{\boldsymbol{x}_2\} = \frac{1}{n}F\{\boldsymbol{x}_1 * \boldsymbol{x}_2\}$$

计算 $\boldsymbol{x}_1 * \boldsymbol{x}_2$ 的过程相当于作了三次快速傅里叶变换或逆变换。故时间复杂性仍为：$O(n\log_2 n)$

4.3.2 多项式的一种快速乘法

已知两个 n 次多项式

$F_1(Y) = a_0 + a_1Y + a_2Y^2 + \cdots + a_{n-1}Y^{n-1}$

$F_2(Y) = b_0 + b_1Y + b_2Y^2 + \cdots + b_{n-1}Y^{n-1}$

$$\begin{aligned} F_1(Y)F_2(Y) = {} & (a_0 + a_1Y + \cdots + a_{n-1}Y^{n-1})(b_0 + b_1Y + \cdots + b_{n-1}Y^{n-1}) \\ = {} & a_0b_0 + (a_0b_1 + a_1b_0)Y + (a_0b_2 + a_1b_1 + a_2b_0)Y^2 + \cdots + (a_0b_{n-1} + a_1b_{n-2} + \cdots \\ & + a_{n-2}b_1 + a_{n-1}b_0)Y^{n-1} + (a_1b_{n-1} + a_2b_{n-2} + \cdots + a_{n-2}b_2 + a_{n-1}b_1)Y^n + (a_2b_{n-1} \\ & + a_3b_{n-2} + \cdots + a_{n-1}b_2)Y^{n+1} + \cdots + (a_{n-2}b_{n-1} + a_{n-1}b_{n-2})Y^{2n-3} \\ & + a_{n-1}b_{n-1}Y^{2n-2} \end{aligned}$$

不难发现 $F_1(Y) \cdot F_2(Y)$ 的系数相当于下列两个列向量：

$$A = (\underbrace{a_0a_1a_2\cdots a_{n-1}}_{n\text{个}}\underbrace{0\ 0\cdots 0}_{n\text{个}})^{\mathrm{T}}$$
$$B = (\underbrace{b_0b_1b_2\cdots b_{n-1}}_{n\text{个}}\underbrace{0\ 0\cdots 0}_{n\text{个}})^{\mathrm{T}}$$

的卷积。

如若采用直接相乘的办法，乘法和加法的计算量均为 n^2。如若采用快速傅里叶变换来求积，其时间复杂性仅为 $O(n\log_2 n)$。

4.4 数论变换

设 $\boldsymbol{a}=\{a_0,a_1,\cdots,a_{N-1}\}$ 和 $\boldsymbol{c}=\{c_0,c_1,\cdots,c_{N-1}\}$ 是两个整数序列，若满足

$$c_k=\sum_{j=0}^{N-1}r^{kj}a_j \mod p \quad k=0,1,2,\cdots,N-1 \tag{4.4.1}$$

其中 r 是满足条件：$r^N\equiv1(\mathrm{mod}\ p)$ 的整数，p 是一素数。且 $N|p-1$，则称 $\boldsymbol{c}=\{c_0,c_1,\cdots,c_{N-1}\}$ 是序列 $\{a_0,a_1,\cdots,a_{N-1}\}$ 的数论变换。记作 $\boldsymbol{c}=NT\{\boldsymbol{a}\}$。

可以看出(4.4.1)的变换在形式上与 FFT 变换相似。r 相当于 $e^{2\pi i/N}$，$r^N\equiv1(\mathrm{mod}\ p)$ 替换 $(e^{2\pi i/N})^N=1$。

类似可得若 $c_0,c_1,\cdots,c_{N-1}$ 满足 4.4.1，则有：

$$a_j\equiv N^{-1}\sum_{k=0}^{N-1}c_kr^{-jk} \quad \mod p \tag{4.4.2}$$

$$j=0,1,2,\cdots,N-1$$

其中 N^{-1} 是满足 $NN^{-1}\equiv1(\mathrm{mod}\ p)$，$r^{-1}$ 满足 $rr^{-1}\equiv1(\mathrm{mod}\ p)$ 的整数，则公式(4.4.2)称为是(4.4.1)的数论逆变换，记作 $\boldsymbol{a}=NT^{-1}\{\boldsymbol{c}\}$。

卷积定理对数论变换也成立。

设序列 $\boldsymbol{a}_1=\{a_0^{(1)},a_1^{(1)},\cdots,a_{N-1}^{(1)}\}$ 的数论变换为

$\boldsymbol{c}_1=\{c_0^{(1)},c_1^{(1)},\cdots,c_{N-1}^{(1)}\}$，

序列 $\boldsymbol{a}_2=\{a_0^{(2)},a_1^{(2)},\cdots,a_{N-1}^{(2)}\}$ 的数论变换为

$\boldsymbol{c}_2=\{c_0^{(2)},c_1^{(2)},\cdots,c_{N-1}^{(2)}\}$，

即

$$c_k^{(i)}=\sum_{h=0}^{N-1}a_h^{(i)}r^{hk} \quad \mod p$$

$$k=0,1,2,\cdots,N-1,\quad i=1,2$$

令

$$Z_k=c_k^{(1)}\cdot c_k^{(2)}=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}a_i^{(1)}a_j^{(2)}r^{(i+j)k} \quad \mod p$$

$$k=0,1,2,\cdots,N-1$$

序列 $\{c_0^{(1)}c_0^{(2)},c_1^{(1)}c_1^{(2)},\cdots,c_{N-1}^{(1)}c_{N-1}^{(2)}\}$ 的逆变换记作 $NT^{-1}\{\boldsymbol{c}_1\cdot\boldsymbol{c}_2\}$。另记：

$$\boldsymbol{a}_1*\boldsymbol{a}_2=\{a_0,a_1,\cdots,a_{N-1}\}$$

为卷积。

其中：

$$a_j=N^{-1}\sum_{i=0}^{N-1}a_i^{(1)}a_{j-i}^{(2)} \quad \mod p \qquad j=0,1,2\cdots,N-1$$

现对 $\boldsymbol{c}_1\boldsymbol{c}_2$ 求其数论逆变换：

$$NT^{-1}(z_h)=N^{-1}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}a_i^{(1)}a_j^{(2)}\sum_{k=0}^{N-1}r^{(i+j-h)k} \quad \mod p$$

由于：

$$\sum_{k=0}^{N-1}r^{(i+j-h)k}\mathrm{mod}\ p=\begin{cases}N & i+j-h=0 \quad \mod N\\ 0 & \text{其他}\end{cases}$$

所以有 $$a_k = N^{-1}\sum_{i=0}^{N-1} a_i^{(1)} a_{k-i}^{(2)} \quad \mod p$$

即 $$NT\{\boldsymbol{a}_1\} \cdot NT\{\boldsymbol{a}_2\} = N^{-1} NT\{\boldsymbol{a}_1 * \boldsymbol{a}_2\}$$

形为 2^m-1 的素数称为 Mersenne 数，例如 $m=3$ 时的 7，$m=5$ 时的 31。$m=7,13,17,19,31$ 分别对应的素数 127，8191，131071，524287，2147483647 等都是 Mersenne 数。由这样的素数所定义的数论变换称为 Mersenne 数论变换。它要求 $N|2^m-2$。以 $m=13$ 为例，$2^{13}-2=2\cdot5\cdot7\cdot9\cdot13$，$N$ 的选择便很明显。r 应选择 $r^N=1 \mod(2^m-1)$。例如：

$$(-2)^{26}-1=[(-2)^{13}-1][(-2)^{13}+1]$$

而 $$(-2)^{13}+1=-(2^{13}-1)\equiv 0 \quad \mod(2^{13}-1)$$

所以 $$(-2)^{26}\equiv 1 \qquad \mod(2^{13}-1)$$

故 Mersenne 数论变换：

$$c_k=\sum_{i=0}^{25}(-2)^{ik}a_i,\ k=0,1,2,\cdots,25$$

一般，由 Fermat 定理 $2^{p-1}\equiv 1 \mod p$

$$p|2^{p-1}-1 \quad 故\ p|(2^p-2)$$

所以

$$\begin{aligned} & p[(2^p-1)-(2^p-2)/p] \\ \equiv & -(2^p-2) \qquad \mod(2^p-1) \\ \equiv & 1 \qquad\qquad \mod(2^p-1) \end{aligned}$$

即在 GF(2^p-1)中 $p^{-1}\equiv 2^p-1-(2^p-2)/p \qquad \mod(2^p-1)$

所以 $N=p$ 时，

$$c_k\equiv\sum_{j=0}^{N-1}a_j 2^{jk} \qquad \mod(2^p-1) \quad j=0,1,2,\cdots,p-1$$

还可推出 $N=2p$ 时有

$$c_k=\sum_{j=0}^{2p-1}a_j(-2)^{jk} \qquad \mod(2^p-1) \quad k=0,1,2,\cdots,2p-1$$

$$a_j=(2p)^{-1}\sum_{k=0}^{2p-1}c_k\cdot 2^{-jk} \qquad \mod(2^p-1) \quad j=0,1,2,\cdots,2p-1$$

而且还可证明：

$$(2p)^{-1}\equiv 2^p-1-(2^p-1)/p \qquad \mod(2^p-1)$$

例如：

$$\boldsymbol{a}=\{1,3,2,2,0\},\quad \boldsymbol{b}=\{3,0,2,1,2\}$$

求 $\boldsymbol{a}*\boldsymbol{b}$。

首先作 Mersenne 变换：

$$p=5,\ 2^5-1=31$$

$$A_k=\sum_{j=0}^{4}a_j 2^{jk} \qquad \mod 31,\quad k=0,1,2,3,4$$

得 $\boldsymbol{A}=\{8,0,18,30,31\}$

同样可得 $\boldsymbol{B}=\{8,20,22,0,27\}$

$$\boldsymbol{C}=\boldsymbol{A}\cdot\boldsymbol{B}=\{2,0,24,0,18\}$$

根据 $p^{-1}=(2^5-1)-(2^5-2)/5 \quad \mod 31$ 可得

$$p^{-1}=25 \quad \mod 31$$

$$2^{-1}=2^4 \quad \mod 31$$

对 $\boldsymbol{A}\cdot\boldsymbol{B}$ 作逆变换：

$$d_l=25\sum_{k=0}^{4}c_k(2)^{4kl} \quad \mod 31$$

有

$$\boldsymbol{a}*\boldsymbol{b}=\{15,15,12,13,9\}$$

若直接使用定义计算：

$$d_l\equiv\sum_{j=0}^{4}a_jb_{l-j},\ \mod 31 \qquad l=0,1,2,3,4$$

$$\boldsymbol{a}*\boldsymbol{b}=\begin{pmatrix}3&2&1&2&0\\0&3&2&1&2\\2&0&3&2&1\\1&2&0&3&2\\2&1&2&0&3\end{pmatrix}\begin{pmatrix}1\\3\\2\\2\\0\end{pmatrix}=\begin{pmatrix}15\\15\\12\\13\\9\end{pmatrix}$$

有相同结果。

当然，当 r 相当大，而卷积的结果所有元素不超过 r 时，所得结果便是正确的。即

$$\boldsymbol{a}*\boldsymbol{b}=N\cdot NT^{-1}\{NT(\boldsymbol{a})\cdot NT(\boldsymbol{b})\}$$

这儿就是这样的情形。

习　题

1. 运用快速傅里叶变换计算

$$x^3+x-3 \text{ 乘以 } 2x^2+2$$

2. 使用快速傅里叶变换计算(6,3,5,1)的傅里叶系数，并用傅里叶逆变换验证之。

3. 用快速傅里叶变换计算 $x_1=(1,3,2,1)$ 和 $x_2=(2,1,4,3)$ 的卷积。

4. 试证明对于 Huffman 编码，若将码按频率出现的大小非减排列时，则其码字的长度必定是按非增顺序排列的。

5. 如果一个文件中由 8-比特的字符构成，这不同的 256 个字符出现的概率几乎是相同的：即最高频率不超过最低频率的一倍。证明 Huffman 编码并不比 8-比特固定长的编码更有效。

6. 已知 8 个字符出现的频率正好依次是 Fibonacci 数 $F_1,F_2,\cdots,F_8$。试设计这 8 个字符的 Huffman 树，并讨论可否将你的结果推广到一般的情形。

7. 试证 n 个字符出现的频率排列成非增序列，则它们的 Huffman 编码的码长为非降序列。

8. 试将 Huffman 算法推广到每一位由 0,1,2 表示的三元码。

9. 试利用卷积计算下列两个多项式的乘法

$$F(x)=8x^3-6x+3,\ G(x)=7x^3-x^2+x+1$$

10. 已知 $A=(a_{ij})_{100\times100}$是 0,1 矩阵,而且每行都从 0 的符号串开始,而且每行 0 符号串和 1 符号串相间的数目不超过 5,试设计一种表达矩阵 A 的一种数据压缩办法,并举例说明之。

第 5 章 线性规划的分解原理

分治策略在线性规划求解的研究也有出色的表现。大的系统往往涉及到若干彼此独立的单位，因此有各单位内部的约束条件，当然各单位之间也有彼此关联的约束条件。下面讨论具有这样特点的问题的分解算法。

5.1 线性规划和单纯形法简介

为讨论方便起见，首先对线性规划问题和单纯形方法作简要的回顾。对已熟悉这部分内容的读者，它完全可以省略去。

设$\boldsymbol{c}=(c_1,c_2,\cdots,c_n)$，$\boldsymbol{b}=(b_1,b_2,\cdots,b_m)^T$

$$\boldsymbol{x}=(x_1,x_2,\cdots,x_n)^T$$

$\boldsymbol{A}=(a_{ij})_{m\times n}$

求

$$\begin{aligned}&\max z=\boldsymbol{cx}\\&\boldsymbol{Ax}=\boldsymbol{b}\\&\boldsymbol{x}\geqslant\boldsymbol{0}\end{aligned}\tag{5.1.1}$$

即在满足约束条件：$\boldsymbol{Ax}=\boldsymbol{b},\boldsymbol{x}\geqslant\boldsymbol{0}$下，使目标函数$z=\boldsymbol{cx}$达到极大值的问题便是典型的线性规划问题。设$\boldsymbol{x}=\begin{pmatrix}\boldsymbol{x}_B\\ \cdots\\ \boldsymbol{x}_N\end{pmatrix}$，$\boldsymbol{x}_B$是$m$维向量称为基变量，$\boldsymbol{x}_N$是$n-m$维向量称为非基变量。对应于$\boldsymbol{x}$，有

$$\begin{aligned}\boldsymbol{c}&=(\boldsymbol{c}_B\vdots\boldsymbol{c}_N)\\ \boldsymbol{A}&=(\boldsymbol{B}\vdots\boldsymbol{N})=(\boldsymbol{p}_1,\boldsymbol{p}_2,\cdots,\boldsymbol{p}_n)\\ \max z&=\boldsymbol{c}_B\boldsymbol{x}_B+\boldsymbol{c}_N\boldsymbol{x}_N\\ \boldsymbol{B}\boldsymbol{x}_B&+\boldsymbol{N}\boldsymbol{x}_N=\boldsymbol{b}\\ \boldsymbol{x}_B,&\boldsymbol{x}_N\geqslant\boldsymbol{0}\end{aligned}$$

则有：

$$\begin{aligned}\boldsymbol{x}_B&=\boldsymbol{B}^{-1}\boldsymbol{b}-\boldsymbol{B}^{-1}\boldsymbol{N}\boldsymbol{x}_N\\&=\boldsymbol{B}^{-1}\boldsymbol{b}-\sum_{j\in N}\boldsymbol{B}^{-1}\boldsymbol{p}_jx_j\end{aligned}\tag{5.1.2}$$

若$\boldsymbol{x}_N=\boldsymbol{0}$，则$\boldsymbol{x}_B=\boldsymbol{B}^{-1}\boldsymbol{b}$。

$$\begin{aligned}z&=\boldsymbol{c}_B(\boldsymbol{B}^{-1}\boldsymbol{b}-\boldsymbol{B}^{-1}\boldsymbol{N}\boldsymbol{x}_N)+\boldsymbol{c}_N\boldsymbol{x}_N\\&=\boldsymbol{c}_B\boldsymbol{B}^{-1}\boldsymbol{b}+\sum_{j\in N}(c_j-\boldsymbol{c}_B\boldsymbol{B}^{-1}\boldsymbol{p}_j)x_j\end{aligned}\tag{5.1.3}$$

故$\boldsymbol{x}_N=\boldsymbol{0}$时，$\boldsymbol{x}_B=\boldsymbol{B}^{-1}\boldsymbol{b}$，对应的目标函数为$z=\boldsymbol{c}_B\boldsymbol{B}^{-1}\boldsymbol{b}$。若非基变量$x_j$对应的$c_j-\boldsymbol{c}_B\boldsymbol{B}^{-1}\boldsymbol{p}_j>0$，则$x_j$从0变为某一正数，目标函数值$z$将随之增加。表达式$c_j-\boldsymbol{c}_B\boldsymbol{B}^{-1}\boldsymbol{p}_j$

$=c_j-z_j$ 称为检验数。不失一般性,假定基变量为 $x_1,x_2,\cdots,x_m$,$\boldsymbol{b}=(b_1,b_2,\cdots,b_m)^T$

即 $\boldsymbol{A}=(\boldsymbol{E}_{(m)}\vdots\boldsymbol{N})_{m\times n}$,$\boldsymbol{E}_{(m)}$表示 m 阶单位阵,则:

$$\boldsymbol{b}=b_1\boldsymbol{p}_1+b_2\boldsymbol{p}_2+\cdots+b_m\boldsymbol{p}_m$$

$$\boldsymbol{p}_j=a_{1j}\boldsymbol{p}_1+a_{2j}\boldsymbol{p}_2+\cdots+a_{mj}\boldsymbol{p}_m$$

$$\boldsymbol{b}-\beta\boldsymbol{p}_j=(b_1-\beta a_{1j})\boldsymbol{p}_1+(b_2-\beta a_{2j})\boldsymbol{p}_2+\cdots+(b_m-\beta \boldsymbol{p}_{mj})\boldsymbol{p}_m \quad (5.1.4)$$

若
$$\beta=\min_h\left\{\frac{b_h}{a_{hj}}\middle|a_{hj}>0\right\}=\frac{b_l}{a_{lj}} \quad (5.1.5)$$

β 的选择是保证由 x_j 的进入不至于使其它应变量出现负值。由于 x_j 由 0 逐步增加,基变量中第一个出现 0 的变量 x_l

则 $b_l-\beta a_{lj}=0$,从而 $\boldsymbol{p}_l$ 为退出基,$\boldsymbol{p}_j$ 为进入基,从原来的可行解

$$(b_1\,b_2\cdots b_m\,00\cdots0) \quad (5.1.6)$$

改变到另一可行解

$$(b_1-\beta a_{1j},b_2-\beta a_{2j},\cdots,\underset{\substack{\uparrow\\ \text{第}l\text{项}}}{0},\cdots,b_m,0,\cdots,\underset{\substack{\uparrow\\ \text{第}j\text{项}}}{\beta},\cdots,0) \quad (5.1.7)$$

由原来的基 $x_1,x_2,\cdots,x_m$ 改变为新的基,x_l 退出 x_j 进入。此时目标函数由原先的

$$z_1=c_1b+c_2b_2+\cdots+c_mb_m$$

变为

$$\begin{aligned}z_2&=c_1(b_1-\beta a_{1j})+c_2(b_2-\beta a_{2j})+\cdots+c_m(b_m-\beta a_{mj})+\beta c_j\\&=c_1b_1+c_2b_2+\cdots+c_mb_m+\beta(c_j-c_1a_{1j}-c_2a_{2j}-\cdots-c_ma_{mj})\\&=z_1+\beta(c_j-z_j)>z_1\end{aligned}$$

即从基 $\boldsymbol{p}_l$ 退出 $\boldsymbol{p}_j$ 进入目标函数上升,有所前进。这样的步骤继续下去,直到到达极值点。这就是单纯形法。

其中$(b_1b_2\cdots b_m\,00\cdots0)$和$(b_1-\beta a_{1j},b_2-\beta a_{2j},\cdots\underset{\text{第}l\text{个}}{0}\cdots,b_m-\beta a_{mj},0\cdots\underset{\text{第}j\text{个}}{\beta}\cdots)$

都是凸多面体:

$$\boldsymbol{Ax}=\boldsymbol{b},\boldsymbol{x}\geqslant\boldsymbol{0}$$

的顶点,举例如下:

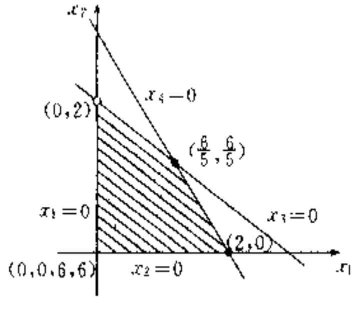


图 5.1.1

$$\begin{aligned}&\max z=x_1+x_2\\&2x_1+3x_2\leqslant 6\\&3x_1+2x_2\leqslant 6\\&x_1,x_2\geqslant 0\end{aligned}$$

本例的几何意义见图 5.1.1。

首先引进松弛变量 x_3,x_4 得

$$\begin{aligned}&\max z=x_1+x_2\\&2x_1+3x_2+x_3\quad=6\\&3x_1+2x_2\quad+x_4=6\\&x_1,x_2,x_3,x_4\geqslant 0\end{aligned}$$

这时 $x_3=x_4=6$ 是基变量，$\boldsymbol{B}=\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$

$c_1-z_1=c_1-\boldsymbol{c}_B\boldsymbol{B}^{-1}\boldsymbol{p}_1=1>0$，故选 x_1 作为进入基

$$\boldsymbol{A}=\begin{pmatrix}2 & 3 & 1 & 0\\ 3 & 2 & 0 & 1\end{pmatrix}=(\boldsymbol{p}_1\ \boldsymbol{p}_2\ \boldsymbol{p}_3\ \boldsymbol{p}_4)$$

故
$$\boldsymbol{p}_1=\begin{pmatrix}2\\ 3\end{pmatrix}=2\boldsymbol{p}_3+3\boldsymbol{p}_4$$

$$\boldsymbol{b}=\begin{pmatrix}6\\ 6\end{pmatrix}=6\boldsymbol{p}_3+6\boldsymbol{p}_4$$

$$\boldsymbol{b}-\beta\boldsymbol{p}_1=(6-2\beta)\boldsymbol{p}_3+(6-3\beta)\boldsymbol{p}_4$$

$\beta=2$ 时 $6-3\beta=0$。

$$\therefore \boldsymbol{b}=2\boldsymbol{p}_1+2\boldsymbol{p}_3$$

这个过程相当于原约束方程组通过消元得一组等价的约束方程组，即

$$2x_1+3x_2+x_3=6$$
$$③x_1+2x_2+x_4=6$$

以 $a_{21}(=3)$ 为主元素进行消元（圆圈标志主元素的位置），第 2 个等式两端除以 3，得

$$2x_1+3x_2+x_3=6$$
$$x_1+\frac{2}{3}x_2+\frac{1}{3}x_4=2$$

第 2 个等式乘以 -2 加到第 1 等式，消去 x_1 得

$$\frac{5}{3}x_2+x_3-\frac{2}{3}x_4=2$$
$$x_1+\frac{2}{3}x_2+\frac{1}{3}x_4=2$$

问题变为解一等价的问题：

$$\max z=x_1+x_2$$
$$\frac{5}{3}x_2+x_3-\frac{2}{3}x_4=2$$
$$x_1+\frac{2}{3}x_2+\frac{1}{3}x_4=2$$
$$x_1,x_2,x_3,x_4\geqslant 0$$

这时基变量变为 x_1 和 x_3，非基变量为 x_2，x_4。x_4 基退出 x_1 取而代之。从凸多面体的顶点 $(0,0,6,6)$ 移到 $(2,0,2,0)$，目标函数从 $z=0$，增至 $z=2$。

上面的计算过程还可以继续下去，直到所有的检验数 c_i-z_i 都非正为止，即

$$c_i-z_i\leqslant 0,\ i=1,2,3,4$$

计算过程可以用紧凑的表格来进行，变成表上作业，即单纯形表格（表 5.1.1）。

表 5.1.1

基	c_B	b \ p \ c	p_1	p_2	p_3	p_4	β
			1	1	0	0	
x_3	0	6	2	3	1	0	3
x_4	0	6	3	2	0	1	2
$c-z$		0	1	1	0	0	
x_3	0	2	0	5/3	1	−2/3	6/5
x_1	1	2	1	2/3	0	1/3	3
		2	0	1/3	0	−1/3	
x_2	1	6/5	0	1	3/5	−2/5	
x_1	1	6/5	1	0	−2/5	3/5	
		12/5	0	0	1/5	1/5	

表上作业(单纯形表格)也还有进一步改善的余地。读者知道消元过程相当于对整个矩阵左乘以 $\boldsymbol{B}^{-1}$，其中 $\boldsymbol{B}$ 是由基变量所对应于 $\boldsymbol{A}$ 的诸列构成的基矩阵。比如上表中以基变量 x_3,x_1 对应的基矩阵及其逆为

$$\boldsymbol{B}=(\boldsymbol{p}_3\ \boldsymbol{p}_1)=\begin{pmatrix}1 & 2\\ 0 & 3\end{pmatrix}$$

$$\boldsymbol{B}^{-1}=\begin{pmatrix}1 & -2/3\\ 0 & 1/3\end{pmatrix}$$

又如基变量 x_2,x_1 对应的基矩阵及其逆为

$$\boldsymbol{B}=(\boldsymbol{p}_2\ \boldsymbol{p}_1)=\begin{pmatrix}3 & 2\\ 2 & 3\end{pmatrix}$$

$$\boldsymbol{B}^{-1}=\begin{pmatrix}3/5 & -2/5\\ -2/5 & 3/5\end{pmatrix}$$

这些 $\boldsymbol{B}^{-1}$ 在单纯形表格中都有，分别在原来矩阵中单位子方阵的下面，这个道理是可以理解的。

单纯形法每次都有一个基变量退出，一个新的基取而代之，已知老的基矩阵 $\boldsymbol{B}_{\text{old}}$ 及其逆 $\boldsymbol{B}_{\text{old}}^{-1}$，新的基矩阵 $\boldsymbol{B}_{\text{new}}$ 和 $\boldsymbol{B}_{\text{old}}$ 只有一列之差。当然，求 $\boldsymbol{B}_{\text{new}}^{-1}$ 是有现成的方法的，问题在于可否利用 $\boldsymbol{B}_{\text{new}}$ 和 $\boldsymbol{B}_{\text{old}}$ 的关系及已知的 $\boldsymbol{B}_{\text{old}}^{-1}$，使求 $\boldsymbol{B}_{\text{new}}^{-1}$ 变得简单起来。答案是肯定的，只要对进入的基进行消元，$\boldsymbol{B}_{\text{old}}^{-1}$ 便可从消元过程变为 $\boldsymbol{B}_{\text{new}}^{-1}$，这从单纯形表格的运算过程中可以清楚地了解到。$\boldsymbol{B}_{\text{new}}^{-1}$ 一旦得到，其它部分自然不难求得。前面已经提到，消元的全过程等价于左乘以 $\boldsymbol{B}^{-1}$，所以求 $\boldsymbol{B}_{\text{new}}^{-1}$ 的计算无需对全部 $(n+m+1)\times m$ 阶矩阵进行，只要在 $(m+1)\times m$ 阶矩阵上进行。

其次，计算 $c_j-z_j=c_j-\boldsymbol{c}_B\boldsymbol{B}^{-1}\boldsymbol{p}_j$，也可以利用

$$\boldsymbol{c}_B\boldsymbol{B}^{-1}\boldsymbol{p}_j=\boldsymbol{c}_B(\boldsymbol{B}^{-1}\boldsymbol{p}_j)=(\boldsymbol{c}_B\boldsymbol{B}^{-1})\boldsymbol{p}_j$$

即结合律成立而得到简化。可先计算 $\boldsymbol{c}_B\boldsymbol{B}^{-1}=\boldsymbol{D}$，使

$$c_j - z_j = \boldsymbol{D}\boldsymbol{p}_j$$

现以

$$\max z = \boldsymbol{c}\boldsymbol{x}$$
$$\boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b}$$
$$\boldsymbol{x} \geqslant \boldsymbol{0}$$

为例叙述改善的单纯形法如下：

$$设\ \boldsymbol{A} = (a_{ij})_{m\times n}$$

(1) 计算 $\boldsymbol{B}^{-1}$，$\boldsymbol{B}^{-1}\boldsymbol{b}$。设 $\boldsymbol{B}^{-1}\boldsymbol{b}=(\tilde{b}_1,\tilde{b}_2,\cdots,\tilde{b}_m)^T$。

(2) 计算 $\boldsymbol{D}=\boldsymbol{c}_B\boldsymbol{B}^{-1}$。

(3) 计算 $c_j-z_j=c_j-\boldsymbol{D}\boldsymbol{p}_j$，若 x_j 为非基变量。若存在 $c_l-z_l>0$，则取 x_l 作为进入基转(4)，否则若对所有的 $c_j-z_j\leqslant 0$，$j=1,2,\cdots,m+n$ 则结束，给出结果。

(4) 计算 $\boldsymbol{B}^{-1}\boldsymbol{p}=(\tilde{a}_{1l},\tilde{a}_{2l},\cdots,\tilde{a}_{ml})^T$

$$求\ \beta = \min_j\left\{\frac{\tilde{b}_l}{\tilde{a}_{jl}}\middle|\tilde{a}_{jl}>0\right\} = \frac{\tilde{b}_h}{\tilde{a}_{hl}}$$

则 x_h 退出以 $\boldsymbol{p}_l$ 取代 $\boldsymbol{p}_h$ 转(1)。

改善的单纯形法对于 m 和 n 比较大时，方显出它的优越性，减少了运算量。而且改善的单纯形法不改变原来的矩阵 $\boldsymbol{A}$ 的内容。

还是以前面的例子再用改善单纯形法求解如下，望读者对照来比较它们的异同。

(1) x_3，x_4 依序为基变量，$\boldsymbol{B}=\boldsymbol{B}^{-1}=\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$，$\boldsymbol{c}_B=(0,0)$，$\boldsymbol{D}=\boldsymbol{c}_B\boldsymbol{B}^{-1}=(0,0)$。

$c_1-z_1=c_1-\boldsymbol{D}\boldsymbol{p}_1=1-(0,0)\begin{pmatrix}2\\3\end{pmatrix}=1>0$，故 x_1 作为进入基。

$\beta=\min\{6/2,6/3\}=b_2/a_{21}=2$，故 x_4 依序为第 2 基变量应退出 x_1。

(2) x_3，x_1 作为基变量，$\boldsymbol{B}^{-1}$ 计算如下

$$\left(\begin{array}{c|cc}2 & 1 & 0\\ (3) & 0 & 1\end{array}\right)\to\left(\begin{array}{c|cc}0 & 1 & -2/3\\ 1 & 0 & 1/3\end{array}\right)，故\ \boldsymbol{B}^{-1}=\begin{pmatrix}1 & -2/3\\ 0 & 1/3\end{pmatrix},$$

$$\boldsymbol{c}_B=(0\quad 1),\ \boldsymbol{D}=(0\quad 1)\begin{pmatrix}1 & -2/3\\ 0 & 1/3\end{pmatrix}=(0\quad 1/3),$$

$$c_2-z_2=1-(0\quad 1/3)\begin{pmatrix}3\\2\end{pmatrix}=1-2/3=1/3>0，故\ x_2\ 作为进入基$$

$$\boldsymbol{B}^{-1}\boldsymbol{b}=\begin{pmatrix}1 & -2/3\\ 0 & 1/3\end{pmatrix}\begin{pmatrix}6\\6\end{pmatrix}=\begin{pmatrix}2\\2\end{pmatrix},\ \tilde{\boldsymbol{p}}_2=\boldsymbol{B}^{-1}\boldsymbol{p}_2=\begin{pmatrix}1 & -2/3\\ 0 & 1/3\end{pmatrix}\begin{pmatrix}3\\2\end{pmatrix}=\begin{pmatrix}5/3\\2/3\end{pmatrix},$$

$$\beta=\min\left\{2\Big/\frac{5}{3},2\Big/\frac{2}{3}\right\}=\min\left\{\frac{6}{5},3\right\}=\frac{6}{5}=\frac{b_1}{a_{12}}$$

故第 1 个基变量 x_3 应退出，即 x_3 被 x_2 所取代。

现在的基变量依顺序为(x_2,x_1)。

(3) 求基变量(x_2,x_1)的 $\boldsymbol{B}^{-1}$ 如下

$$\begin{pmatrix} \frac{5}{3} & 1 & -\frac{2}{3} \\ \frac{2}{3} & 0 & \frac{1}{3} \end{pmatrix} \to \begin{pmatrix} 1 & \frac{3}{5} & -\frac{2}{5} \\ 0 & -\frac{2}{5} & \frac{3}{5} \end{pmatrix}$$

即
$$\boldsymbol{B}^{-1} = \begin{pmatrix} 3/5 & -2/5 \\ -2/5 & 3/5 \end{pmatrix}, \ \boldsymbol{c}_B = (1\ 1)$$

$$\boldsymbol{D} = \boldsymbol{c}_B\boldsymbol{B}^{-1} = (1,1)\begin{pmatrix} 3/5 & -2/5 \\ -2/5 & 3/5 \end{pmatrix} = (1/5\quad 1/5),$$

$$c_3 - z_3 = c_3 - \boldsymbol{D}\boldsymbol{p}_3 = -(1/5\quad 1/5)\begin{pmatrix} 1 \\ 0 \end{pmatrix} = -1/5 < 0$$

$$c_4 - z_4 = c_4 - \boldsymbol{D}\boldsymbol{p}_4 = -(1/5\quad 1/5)\begin{pmatrix} 0 \\ 1 \end{pmatrix} = -1/5 < 0$$

故已达到最优解

$$\bar{\boldsymbol{b}} = \begin{pmatrix} 3/5 & -2/5 \\ -2/5 & 3/5 \end{pmatrix}\begin{pmatrix} 6 \\ 6 \end{pmatrix} = \begin{pmatrix} 6/5 \\ 6/5 \end{pmatrix}$$

故基变量 $x_1=6/5, x_2=6/5$，

非基变量 $x_3=x_4=0$，

$$z = \boldsymbol{D}\bar{\boldsymbol{b}} = (1\ 1)\begin{pmatrix} 6/5 \\ 6/5 \end{pmatrix} = 12/5$$

最后还要强调：

(1) 改善单纯形在规模较大问题便显出它的优势；

(2) 改善单纯形法实际上就是单纯形法，过程和单纯形表格一致。

5.2 Dantzig-Wolfe 分解算法

对于许多单位协同工作的大系统，既有各单位间的相互约束条件，又有各单位内部的约束条件。许多大的线性规划问题可归结为如下类型。

目标函数 $Z=C_1X_1+C_2X_2+\cdots+C_NX_N$ 的极值

约束条件：

$$\begin{cases} A_1X_1 + A_2X_2 + \cdots + A_nX_n \leqslant (\geqslant)\boldsymbol{b} \\ B_1X_1 \qquad\qquad\qquad\qquad \leqslant (\geqslant)\boldsymbol{b}_1 \\ \qquad B_2X_2 \qquad\qquad\qquad \leqslant (\geqslant)\boldsymbol{b}_2 \\ \qquad \cdots\ \cdots\ \cdots \\ \qquad\qquad\qquad B_nX_n \leqslant (\geqslant)\boldsymbol{b}_n \\ X_1, X_2, \cdots, X_n \qquad \geqslant \boldsymbol{0} \end{cases}$$

约束条件用矩阵形式表示如下：

$$AX=\begin{pmatrix}A_1 & A_2 & \cdots & A_n\\ B_1 & 0 & \cdots & 0\\ 0 & B_2 & \cdots & 0\\ & \cdots & \cdots & \\ 0 & 0 & \cdots & B_n\end{pmatrix}\begin{pmatrix}X_1\\ X_2\\ \vdots\\ X_n\end{pmatrix}\leqslant(\geqslant)\begin{pmatrix}b_1\\ b_2\\ \vdots\\ b_n\end{pmatrix}$$

其中 $X_i, i=1,2,\cdots,n$ 为列向量。

约束矩阵 A 是稀疏阵，针对这种特点的线性规划问题，为了节省存储单元，减少计算量，可将问题分解为若干规模较小的子问题。下面以 $n=2$ 为例，介绍分解原理，一般的问题其步骤也是一样的。

$$\min z=C_1X_1+C_2X_2$$

$$\begin{cases}A_1X_1+A_2X_2=b\\ B_1X_1 \qquad\quad =b_1\\ \qquad\quad B_2X_2=b_2\\ \qquad X_1,X_2\geqslant 0\end{cases}\tag{P}$$

可将问题 (P) 看作是：

$$\min z=CX$$
$$AX=b$$
$$X\in S$$

其中 $X=\begin{pmatrix}X_1\\ X_2\end{pmatrix}$，$A=(A_1\mid A_2)$，$C=(C_1\mid C_2)$

$$S:\begin{cases}B_1X_1=b_1\\ B_2X_2=b_2\\ X_1,X_2\geqslant 0\end{cases}$$

S 是超凸多面体，设 $\overline{X}_1,\overline{X}_2,\cdots,\overline{X}_n$ 是 S 的所有顶点，故依据凸多面体的性质，存在 n 个非负数 $t_i, i=1,2,\cdots,n$，满足 $\sum\limits_{i=1}^{n}t_i=1$，使得

$$X=t_1\overline{X}_1+t_2\overline{X}_2+\cdots+t_n\overline{X}_n$$
$$CX=(C\overline{X}_1)t_1+(C\overline{X}_2)t_2+\cdots+(C\overline{X}_n)t_n$$

代入 (P)，于是得到关于变量 $t_1,t_2,\cdots,t_n$ 的线性规划问题 (T) 如下：

$$\min z=(C\overline{X}_1)t_1+(C\overline{X}_2)t_2+\cdots+(C\overline{X}_n)t_n$$

$$\begin{cases}(A\overline{X}_1)t_1+(A\overline{X}_2)t_2+\cdots+(A\overline{X}_n)t_n=b\\ t_1+t_2+\cdots+t_n=1\\ t_i\geqslant 0, i=1,2,\cdots,n\end{cases}\tag{T}$$

引进符号：$A\overline{X}_1=A_1, A\overline{X}_2=A_2,\cdots,A\overline{X}_n=A_n$

$C\overline{X}_1=c_1, C\overline{X}_2=c_2,\cdots,C\overline{X}_n=c_n$

于是问题 (T) 可以写成：

$$\min z=c_1t_1+c_2t_2+\cdots+c_nt_n$$

$$
\begin{aligned}
&\boldsymbol{A}_1 t_1 + \boldsymbol{A}_2 t_2 + \cdots + \boldsymbol{A}_n t_n = \boldsymbol{b} \\
&t_1 + t_2 + \cdots + t_n = 1 \\
&t_i \geqslant 0, i = 1,2,\cdots,n
\end{aligned} \tag{T}
$$

实际上，$\boldsymbol{A}_i$，c_i，$i=1,2,\cdots,n$ 客观上存在，但也都是尚未算出来的量。下面利用改进的单纯形法解问题(T)时，首先要确定进入的基 $\boldsymbol{P}_k=\begin{pmatrix}\boldsymbol{A}\overline{\boldsymbol{X}}_k \\ 1\end{pmatrix}$。根据单纯形法，可要求：

$$c_k - z_k = \min_j \{c_j - z_j\} < 0$$

而在问题(T)中：

$$c_j = \boldsymbol{C}\overline{\boldsymbol{X}}_j$$

$$z_j = \boldsymbol{C}_B \boldsymbol{B}^{-1} \boldsymbol{P}_j = \boldsymbol{C}_B \boldsymbol{B}^{-1} \begin{pmatrix}\boldsymbol{A}\overline{\boldsymbol{X}}_j \\ 1\end{pmatrix},$$

若进入基要求 $\min\{c_j - z_j\}$，则导致求解下列线性规划问题：

$$
\begin{aligned}
&\min w = \boldsymbol{C}\boldsymbol{X} - \boldsymbol{C}_B \boldsymbol{B}^{-1} \begin{pmatrix}\boldsymbol{A}\boldsymbol{X} \\ 1\end{pmatrix} \\
&\boldsymbol{X} \in \boldsymbol{S}
\end{aligned} \tag{S}
$$

求(S)的解的步骤就是单纯形法。但由于可行解域 $\boldsymbol{S}$ 是若干子凸多面体 $\boldsymbol{S}_i$ 的直积，即 $\boldsymbol{S}$： $\boldsymbol{B}_1\boldsymbol{X}_1=\boldsymbol{b}_1$，$\boldsymbol{X}_1\geqslant\boldsymbol{0}$，$\boldsymbol{B}_2\boldsymbol{X}_2=\boldsymbol{b}_2$，$\boldsymbol{X}_2\geqslant\boldsymbol{0}$，所以

$$\boldsymbol{S} = \boldsymbol{S}_1 \times \boldsymbol{S}_2$$

其中 $\boldsymbol{S}_1$： $\boldsymbol{B}_1\boldsymbol{X}_1=\boldsymbol{b}_1$，$\boldsymbol{X}_1\geqslant\boldsymbol{0}$；$\boldsymbol{S}_2$：$\boldsymbol{B}_2\boldsymbol{X}_2=\boldsymbol{b}_2$，$\boldsymbol{X}_2\geqslant\boldsymbol{0}$

故问题(S)可分解成若干个子问题。这里×是直积的意思。

举例说明就不难理解如何分解成若干子问题。

$$
\begin{aligned}
\min z = &-2x_1 - x_2 - 2x_3 + 2x_4 \\
&x_1 + x_3 \leqslant 4 \\
&x_1 + x_2 + 2x_4 \leqslant 4 \\
&x_1 \leqslant 4 \\
&x_1 + 2x_2 \leqslant 8 \\
&-x_3 + x_4 \leqslant 4 \\
&2x_3 + x_4 \leqslant 6 \\
&x_1, x_2, x_3, x_4 \geqslant 0
\end{aligned}
$$

将问题看作是

$$
\begin{aligned}
&\min z = \boldsymbol{C}\boldsymbol{X} \\
&\boldsymbol{A}\boldsymbol{X} \leqslant \boldsymbol{b} \\
&\boldsymbol{X} \in \boldsymbol{S}
\end{aligned}
$$

其中 $\boldsymbol{C}=(-2\quad -1\quad -2\quad 2)$，$\boldsymbol{A}=\begin{pmatrix}1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 2\end{pmatrix}$，

$$\boldsymbol{X} = (x_1\ x_2\ x_3\ x_4)^{\mathrm{T}},\ \boldsymbol{b} = \begin{pmatrix}4 \\ 6\end{pmatrix}$$

$$\boldsymbol{S} = \boldsymbol{S}_1 \times \boldsymbol{S}_2$$

$$\boldsymbol{S}_1: \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leqslant \begin{pmatrix} 4 \\ 8 \end{pmatrix}, \quad \boldsymbol{S}_2: \begin{pmatrix} -1 & 1 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} x_3 \\ x_4 \end{pmatrix} \leqslant \begin{pmatrix} 4 \\ 6 \end{pmatrix}$$

$$x_1, x_2 \geqslant 0, \quad x_3, x_4 \geqslant 0$$

$\boldsymbol{S}=\boldsymbol{S}_1\times\boldsymbol{S}_2$，即 $\boldsymbol{S}$ 分解为 $\boldsymbol{S}_1$ 和 $\boldsymbol{S}_2$ 的直积，其中 $\boldsymbol{S}_1$ 和 $\boldsymbol{S}_2$ 分别如图 5.2.1 所示。

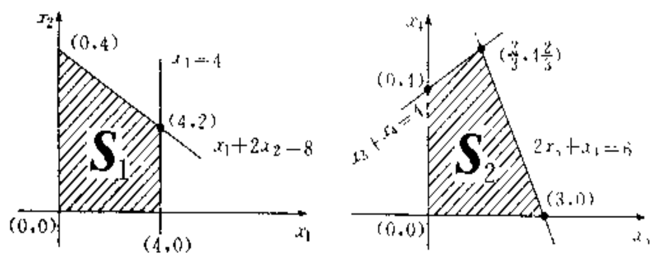


图 5.2.1

设 $\dot{\boldsymbol{X}}_1, \dot{\boldsymbol{X}}_2, \cdots, \dot{\boldsymbol{X}}_n$ 是 $\boldsymbol{S}$ 的顶点，其中 $\dot{\boldsymbol{X}}_1=(0\ 0\ 0\ 0)^{\mathrm{T}}$，本例实际上 $n=4\times4=16$，对所有 $\boldsymbol{X}\in\boldsymbol{S}$，有

$$\boldsymbol{X}=t_1\dot{\boldsymbol{X}}_1+t_2\dot{\boldsymbol{X}}_2+\cdots+t_n\dot{\boldsymbol{X}}_n=\sum_{i=1}^{n}t_i\dot{\boldsymbol{X}}_i,$$

其中 $t_1, t_2, \cdots, t_n\geqslant0, t_1+t_2+\cdots+t_n=1$

所以
$$\boldsymbol{C}\boldsymbol{X}=\sum_{i=1}^{n}t_i\boldsymbol{C}\dot{\boldsymbol{X}}_i=\sum_{i=1}^{n}\dot{c}_it_i$$

其中
$$\dot{c}_i=\boldsymbol{C}\dot{\boldsymbol{X}}_i$$

问题导致求 t 的线性规划问题

$$\min z=\sum_{i=1}^{n}\dot{c}_it_i$$

$$\sum_{i=1}^{n}\boldsymbol{A}_it_i\leqslant\begin{pmatrix}4\\6\end{pmatrix}$$

$$\sum_{i=1}^{n}t_i=1$$

$$t_i\geqslant0, i=1,2,\cdots,n$$

其中 $\boldsymbol{A}_i=\boldsymbol{A}\boldsymbol{X}_i, i=1,2,\cdots,n$，其中由于 $\boldsymbol{X}_1=(0\ 0\ 0\ 0)^{\mathrm{T}}$，

故
$$\boldsymbol{A}_1=\boldsymbol{A}\boldsymbol{X}_1=\begin{pmatrix}0\\0\end{pmatrix}, \quad c_1=\boldsymbol{C}\boldsymbol{X}_1=0$$

引进松弛变量 S_1, S_2 使得变成下列问题：

$$\min z=c_2t_2+\cdots+c_nt_n$$

$$\boldsymbol{A}_2t_2+\boldsymbol{A}_3t_3+\cdots+\boldsymbol{A}_nt_n+\boldsymbol{E}_1S_1+\boldsymbol{E}_2S_2=\begin{pmatrix}4\\6\end{pmatrix}$$

$$t_1+t_2+\cdots+t_n=1$$

$$t_i\geqslant0, i=1,2,\cdots,n$$

$$S_1 \geqslant 0,\ S_2 \geqslant 0$$

其中 $$\boldsymbol{E}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix},\boldsymbol{E}_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

取 t_1, S_1, S_2 作为基，$t=1, S_1=4, S_2=6$ 是一组可行解。故有矩阵(见表 5.2.2)：

表 5.2.2

基	C_B	b	B^{-1}		
S_1	0	4	1	0	0
S_2	0	6	0	1	0
t	0	1	0	0	1

由于 $\boldsymbol{B}$ 是单位阵，故 $\boldsymbol{B}^{-1}$ 也是单位阵，其中 $\boldsymbol{C}_B$ 为基变量的目标函数系数。$\boldsymbol{B}^{-1}$ 为基矩阵的逆矩阵。

$$\boldsymbol{C}_B\boldsymbol{B}^{-1} = (0\ 0\ 0)$$

寻找进入基的问题导至解下面的线性规划问题

$$\min w = -2x_1 - x_2 - 2x_3 + 2x_4 + (0\ 0\ 0)\begin{pmatrix} x_1 + x_3 \\ x_1 + x_2 + 2x_4 \end{pmatrix}$$

$$\min w = -2x_1 - x_2 - 2x_3 + 2x_4$$

$$\boldsymbol{X} \in \boldsymbol{S}$$

本问题分解为两个子问题：

$$\begin{aligned} &\min w_1 = -2x_1 - x_2 \\ &x_1 + x_2 \leqslant 4 \\ &x_1 + 2x_2 \leqslant 8 \\ &x_1, x_2 \geqslant 0 \end{aligned} \qquad \begin{aligned} &\min w_2 = -2x_3 + 2x_4 \\ &-x_3 + x_4 \leqslant 4 \\ &2x_3 + x_4 \leqslant 6 \\ &x_3, x_4 \geqslant 0 \end{aligned}$$

由图 5.2.1 可得解

$$x_1 = 4, x_2 = 2, x_3 = 3, x_4 = 0$$

即得 $\boldsymbol{X}_2 = (4\ 2\ 3\ 0)$ 是 $\boldsymbol{S}$ 的顶点

由于 $\boldsymbol{C}_B\boldsymbol{B}^{-1} = (0\ 0\ 0)$，故 $\boldsymbol{X}_2$ 点对应的

$$C_2 - Z_2 = \boldsymbol{C}\boldsymbol{X}_2 - \boldsymbol{C}_B\boldsymbol{B}^{-1} \cdot \begin{pmatrix} \boldsymbol{A}\boldsymbol{X}_2 \\ 1 \end{pmatrix}$$

$$= (-2\quad -1\quad -2\quad +2)\begin{pmatrix} 4 \\ 2 \\ 3 \\ 0 \end{pmatrix} = -16 < 0$$

即 $\boldsymbol{P}_2 = \begin{pmatrix} \boldsymbol{A}\boldsymbol{X}_2 \\ 1 \end{pmatrix}$ 可选作进入基

$$AX_2 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 2 \end{pmatrix}\begin{pmatrix} 4 \\ 2 \\ 3 \\ 0 \end{pmatrix} = \begin{pmatrix} 7 \\ 6 \end{pmatrix}$$

$$\overline{P}_2 = B^{-1}\begin{pmatrix} AX_2 \\ 1 \end{pmatrix} = \begin{pmatrix} 7 \\ 6 \\ 1 \end{pmatrix}$$

表 5.2.3 确定由 P_2 作为进入基后的退出基。

表 5.2.3

基	C_B	b	B^{-1}			P_2	β
S_1	0	4	1	0	0	7	4/7
S_2	0	6	0	1	0	6	1
t_1	0	1	0	0	1	1	1

由表 5.2.3 消元得表 5.2.4。

表 5.2.4

基	C_B	b	B^{-1}			P_2
t_2	-16	4/7	1/7	0	0	1
S_2	0	18/7	$-6/7$	1	0	0
t_1	0	3/7	$-1/7$	6	1	0

即 t_2 进入，S_1 退出。

$$t_1 = 3/7,\ t_2 = 4/7,\ S_2 = 18/7$$

$$X = t_1X_1 + t_2X_2 = \frac{4}{7}(4\ 2\ 3\ 0)^{\mathrm{T}} = \left(\frac{16}{7}\ \frac{8}{7}\ \frac{12}{7}\ 0\right)^{\mathrm{T}}$$

$$CX = (-2\quad -1\quad -2\quad 2)\left(\frac{12}{7}\ \frac{6}{7}\ \frac{9}{7}\ 0\right)^{\mathrm{T}} = -\frac{48}{7}$$

新的一轮计算开始

$$C_B = (-16\quad 0\quad 0)$$

$$C_BB^{-1} = (-16\quad 0\quad 0)\begin{pmatrix} \frac{1}{7} & 0 & 0 \\ -\frac{6}{7} & 1 & 0 \\ -\frac{1}{7} & 0 & 1 \end{pmatrix} = \left(-\frac{16}{7}\quad 0\quad 0\right)$$

找新的进入基导致解下面线性规划问题

$$\min w = CX - C_BB^{-1}\begin{pmatrix} AX \\ 1 \end{pmatrix}$$

$$X \in S$$

即

$$\min w = -2x_1 - x_2 - 2x_3 + 2x_4 - \left(-\frac{16}{7}\ 0\ 0\right)\begin{pmatrix} x_1 + x_3 \\ x_1 + x_2 + 2x_4 \\ 1 \end{pmatrix}$$

$$= \frac{2}{7}x_1 - x_2 + \frac{2}{7}x_3 + 2x_4$$

$$X \in S$$

导致解两个子问题

$$\min w_1 = \frac{2}{7}x_1 - x_2 \qquad \min w_2 = \frac{2}{7}x_3 + 2x_4$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in S_1 \qquad \begin{pmatrix} x_3 \\ x_4 \end{pmatrix} \in S_2$$

由图 5.2.1 可知 $x_1=0, x_2=4, x_3=0, x_4=0$ 是解。即得 S 的又一顶点

$$X_3 = \begin{pmatrix} 0 \\ 4 \\ 0 \\ 0 \end{pmatrix}$$

$$AX_3 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 2 \end{pmatrix}\begin{pmatrix} 0 \\ 4 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \end{pmatrix}$$

$$C_3 = CX_3 = (-2\ \ -1\ \ -2\ \ 2)\begin{pmatrix} 0 \\ 4 \\ 0 \\ 0 \end{pmatrix} = -4$$

$$z_3 = C_B B^{-1}\begin{pmatrix} AX_3 \\ 1 \end{pmatrix} = \left(-\frac{16}{7}\ 0\ 0\right)\begin{pmatrix} 0 \\ 4 \\ 1 \end{pmatrix} = 0$$

$$C_3 - z_3 = -4 < 0$$

故选 P_3 作为进入基

$$\overline{P}_3 = B^{-1}\begin{pmatrix} AX_3 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{7} & 0 & 0 \\ 6/7 & 1 & 0 \\ -\frac{1}{7} & 0 & 1 \end{pmatrix}\begin{pmatrix} 0 \\ 4 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \\ 1 \end{pmatrix}$$

确定退出基，由表 5.2.4 得表 5.2.5。

表 5.2.5

基	C_B	b	B^{-1}			P	β
t_2	-16	4/7	1/7	0	0	0	
S_2	0	18/7	$-6/7$	1	0	4	18/28
t_1	0	3/7	1/7	6	1	(1)	(3/7)

消元得表 5.2.6。

表 5.2.6

基	C_B	b	B^{-1}			P_3
t_2	-16	4/7	1/7	0	0	0
S_2	0	6/7	10/7	1	-4	0
t_1	-1	3/7	1/7	0	1	1

$$\boldsymbol{X} = t_2\boldsymbol{X}_2 + t_1\boldsymbol{X}_3$$

$$= \frac{4}{7}\begin{pmatrix}4\\2\\3\\0\end{pmatrix} + \frac{3}{7}\begin{pmatrix}0\\4\\0\\0\end{pmatrix} = \begin{pmatrix}16/7\\20/7\\12/7\\0\end{pmatrix}$$

$$\boldsymbol{CX} = (-2\ \ -1\ \ \ 2\ \ 2)\begin{pmatrix}16/7\\20/7\\12/7\\0\end{pmatrix} = -76/7$$

新的一轮找进入基重新开始

$$\boldsymbol{C}_B\boldsymbol{B}^{-1} = (-16\ \ 0\ \ -4)\begin{pmatrix}1/7 & 0 & 0\\ -10/7 & 1 & -4\\ 1/7 & 0 & 1\end{pmatrix} = \left(-\frac{20}{7}\ \ 0\ \ -4\right)$$

$$\min w = \boldsymbol{CX} - \boldsymbol{C}_B\boldsymbol{B}^{-1}\begin{pmatrix}\boldsymbol{AX}\\1\end{pmatrix}$$

$$= \frac{6}{7}x_1 - x_2 + \frac{6}{7}x_3 + 2x_4 + 4$$

$$\boldsymbol{X} \in \boldsymbol{S}$$

通过图 5.2.1 可知解为

$$x_1 = \begin{pmatrix}0\\4\\0\\0\end{pmatrix}$$

这时 $w=0$,即问题已得到最优解。

习　　题

1. 求解 $\min z=-x_1-3x_2+x_3-x_4$

$$\begin{aligned}
&x_1+x_2+x_3+x_4 &&\leqslant 8\\
&x_1+x_2 &&\leqslant 6\\
&\qquad\quad x_3+2x_4 &&\leqslant 10\\
&\qquad\quad -x_3+x_4 &&\leqslant 4\\
&x_i\geqslant 0,\ i=1,2,3,4
\end{aligned}$$

2. 求解 $\max z=x_1+8x_2+5x_3+6x_4$

$$\begin{aligned}
&x_1-4x_2+5x_3+2x_4\leqslant 7\\
&2x_1-3x_2 \leqslant 6\\
&5x_1-x_2 \leqslant 5\\
&\qquad 3x_3+4x_4\geqslant 12\\
&\qquad x_3 \leqslant 4\\
&\qquad\qquad x_4\leqslant 3\\
&x_i\geqslant 0,\ i=1,2,3,4
\end{aligned}$$

3. 求解 $\max z=x_1+x_2+3x_3-x_4$

$$\begin{aligned}
&x_1+x_2+x_3-x_4 \leqslant 12\\
&3x_1-4x_2 \leqslant 5\\
&-x_1+x_2 \leqslant 2\\
&\qquad x_3+x_4\leqslant 4\\
&\qquad x_3-x_4\leqslant 5\\
&x_i\geqslant 0,\ i=1,2,3,4
\end{aligned}$$

4. 求解 $\max z=6x_1+4x_2+3x_4$

$$\begin{aligned}
&x_1+x_2+x_4\leqslant 3\\
&\quad x_2+x_3+x_4 \leqslant 5\\
&2x_1+x_2 \leqslant 6\\
&x_1+x_3 \leqslant 4\\
&\qquad x_3+x_4 \leqslant 3\\
&\qquad x_3+2x_4 \leqslant 4\\
&x_1,x_2,x_3,x_4\geqslant 0
\end{aligned}$$

5. 求解 $\max z=x_1+4x_2-x_3+4x_4+2x_5+3x_6$

$$\begin{aligned}
&-x_1-x_2-x_3+x_4 \leqslant 0\\
&x_1+2x_2 \leqslant 3\\
&\qquad x_3-2x_4 \leqslant 3\\
&\qquad -x_3-x_4+x_5+x_6\leqslant 0
\end{aligned}$$

$$x_5+x_6\leqslant 3$$

$$x_i\geqslant 0,\ i=1,2,\cdots,6$$

6. 求解 $\min z=-x_1-3x_2+x_3-x_4$

$$\begin{aligned} x_1+x_2+x_3+x_4 &\leqslant 8\\ x_1+x_2 &\leqslant 6\\ x_3+2x_4 &\leqslant 10\\ -x_3+x_4 &\leqslant 4 \end{aligned}$$

$$x_i\geqslant 0,\ i=1,2,3,4$$

第6章 最佳二分树

6.1 二 分 树

6.1.1 二分树的定义

二分树是一种特殊的树，它有有限个节点，每个内节点(即非树的叶节点)有左右两个子二分树。当然这两个左右子二分树不相交(即左右两个子二分树没有共同的顶点)。

设二分树 BT 的"叶"节点为 $v_1,v_2,\cdots,v_n$。假定树中每条边的长度为1，从二分树 BT 的根节点到 v_i 的距离设为 l_i，也就是从根节点到叶节点 v_i 的路径上边的数目。非叶节点的点称之为内点。

6.1.2 二分树的性质

1. 具有 n 个叶节点的二分树有 $n-1$ 个内点。

这个结论可以用数学归纳法证明如下：

$n=2$ 时结论是显然的。

假设结论对于具有 $n-1$ 个叶节点的二分树是对的。即假定 $n-1$ 个叶子的二分树有 $n-2$ 个内点，这结论是对的。

现证明 n 的情形：

对于某一有 n 个节点的二分树 BT，必有一内节点 v_i' 与两个叶节点 v_k 和 v_j 相邻接，称 v_i' 是 v_k 和 v_j 的父亲节点。从 BT 中去掉 v_k 和 v_j 两叶子，使 v_i' 成为叶节点，从而得一新二分树 BT'，BT' 只有 $n-1$ 个叶节点。由归纳假设，BT' 有 $n-2$ 个内点，再加上 v_j 和 v_i 得 BT 树，从而得到二分树 BT 有 $n-1$ 个内点。

2. 设 $v_1,v_2,\cdots,v_n$ 为某二分树的叶节点，从各叶节点到树根的距离分别为 $l_1,l_2,\cdots,l_n$，则有：

$$\sum_{i=1}^{n} 2^{-l_i} = 1$$

从根节点出发，每个节点都有左右两种选择，设向左或向右的概率相等，各为$\frac{1}{2}$，所以到达叶节点 v_i 的概率为 2^{-l_i}，$1\leqslant i\leqslant n$。而从根节点出发，必到达某一叶节点而终止，故

$$\sum_{i=1}^{n} 2^{-l_i} = 1$$

3. 设 $v_1',v_2',\cdots,v_{n-1}'$ 是有 n 个叶节点的二分树的内点。属于内点 v_k' 的"后代"的叶节点数目令之为 n_k，$k=1,2,\cdots,n-1$。则：

$$\sum_{k=1}^{n-1} n_k = \sum_{i=1}^{n} l_i$$

在证明该性质之前先看一个例子，如图 6.1.1。

$n_1=4, n_2=3, n_3=2$

$l_1=2, l_2=3, l_3=3, l_4=1$

$n_1+n_2+n_3=4+3+2=9$

$l_1+l_2+l_3+l_4=2+3+3+1=9$

证明 使用数学归纳法，对 n 进行归纳。

显然当 $n=2$ 时，如图 6.1.2，结论是对的。

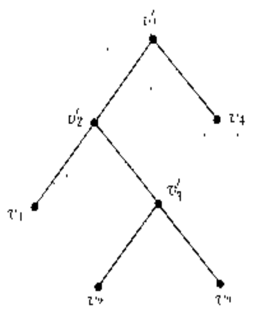


图 6.1.1

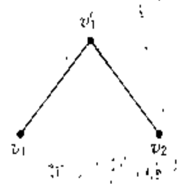


图 6.1.2

假设对于具有 n 个节点的二分树，结论是正确的，即

$$\sum_{k=1}^{n-1} n_k = \sum_{i=1}^{n} l_i$$

设 BT 是有 $n+1$ 个叶节点的二分树，如图 6.1.3(a)所示，不失一般性，假定 v_n 和 v_{n+1} 具有同一个"父亲"节点 v_n'。

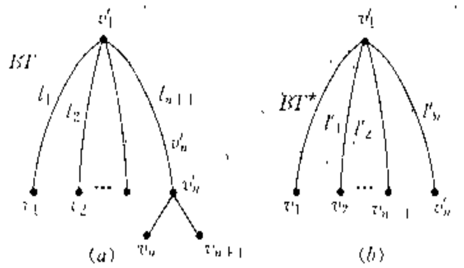


图 6.1.3

将叶子 v_n 和 v_{n+1} 剪去，于是 v_n' 便成为叶子节点，从而得到有 n 个叶节点的二分树 BT^*，如图 6.1.3(b)。设 BT^* 的 n 个叶节点到树根节点的距离为 $l_1', l_2', \cdots, l_n'$，其 $n-1$ 个内点的"后代"的叶节点数目分别为 $n_1', n_2', \cdots, n_{n-1}'$，由归纳假设，有：

$$\sum_{i=1}^{n-1} n_i' = \sum_{i=1}^{n} l_i'$$

现在来观察树 BT：从根节点 v_1' 到叶节点 $v_1, v_2, \cdots, v_{n-1}$ 的距离与 BT^* 相同，即

$$l_i = l_i', \ i = 1,2,\cdots,n-1$$

而
$$l_n = l_{n+1} - l'_n + 1$$
所以
$$\sum_{i=1}^{n+1} l_i = \sum_{j=1}^{n} l'_j + l'_n + 2$$
对于树 BT 来讲，从树根 v_1 到 v_n 或 v_{n+1} 的路径上的每个内点对应的 n_k 比 BT' 多 1，而且又多了一个内点 v'_n，$n_k=2$，其余的内点情况不变，从而有
$$\sum_{i=1}^{n} n'_i + l_n + 2 = \sum_{j=1}^{n+1} n_j$$
故得
$$\sum_{k=1}^{n} n_k = \sum_{j=1}^{n+1} l_j$$
4. $\min\limits_{T\in T^*}\max\limits_{v_i\in T}\{l_i^{(T)}\} = \lceil \log_2 n \rceil$

其中 T^* 为具有 n 个叶节点的二分树集合，$\lceil \log_2 n \rceil$ 为大于 $\log_2 n$ 的最小整数。

这个结论说明每一棵有 n 个叶节点的二分树 $T\in T^*$，对应 $l_{\max}^{(T)}=\max\limits_{v_i\in T}\{l_i^{(T)}\}$；对于所有 n 个叶节点的二分树的集合 T^*，$\min\limits_{T\in T^*}\{l_{\max}^{(T)}\}=\lceil \log_2 n \rceil$。

显然，所有的 n 个叶节点的二分树中，只有使与根的距离最短的叶节点的个数达到最多时，才能达到 $l_{\max}$ 取最小值 k。例如图 6.1.4 所示的那样。故 k 应满足 $2^k\geqslant n$，$k=\lceil \log_2 n \rceil$。一般当 $k=\log_2 n$ 时，图 6.1.4(a)的形状可使 $l_{\max}$ 达到最小。

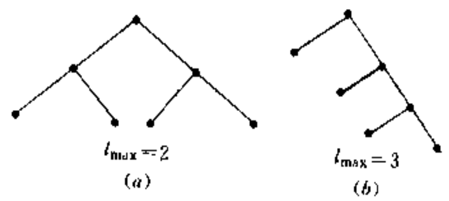


图 6.1.4

一般，若 $2^k<n<2^{k+1}$ 时，n 个叶节点的二分树在什么状态下做到树的高度最小，从下面将看到应该有：
$$k\leqslant l_i\leqslant k+1,\ i=1,2,\cdots,n$$
换句话说，就是任意两叶节点的距离 l_i 和 l_j 满足：
$$0\leqslant |l_i-l_j|\leqslant 1,\ i,j=1,2,\cdots,n$$
如若不然，设 v_i 和 v_j 对应的 l_i 和 l_j 满足：
$$l_i<l_j,\ l_j-l_i\geqslant 2$$
如图 6.1.5 所示，设(a)图为 T_1，(b)图为 T_2，又设 v_k 和 v_j 是兄弟节点，v'_h 是它们的"父亲"节点。若将 T_1 中的 v_k 和 v_j 叶节点剪接于 v_i 点，即以 v_i 作为 v_j 和 v_k 的"父亲"节点，得 T_2。从根节点到 v_j 的距离设为 l'_j，v'_h 成为叶节点，其距离(长度)设为 l'_h，则
$$l'_h=l_j-1\geqslant l'_j$$
而

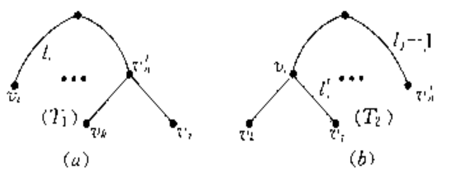


图 6.1.5

$$l_u'-l_j'=(l_j-1)-(l_i+1)=l_j-l_i-2$$

若 $l_j-l_i=2$，则 $l_u'-l_i'=0$

也就是说经过若干次调整，使得任意两叶节点 v_i 和 v_j，有

$$0\leqslant|l_i-l_j|\leqslant1$$

也就证明了

$$\min_{T\in T}\max_{v_i\in L}\{l_i^{(T)}\}=\lceil\log_2 n\rceil$$

从上面证明知，若 $n=2^k$，则 $l_{\max}=\log_2 n$。

所以
$$\sum_{i=1}^{n} l_i = nk = n\log_2 n$$

若
$$n=2^k+m,\ 1\leqslant m<2^k$$

根据使 $\{\sum_{i=1}^{n}l_i\}$ 取最小值的二分树的特点 $0\leqslant|l_i-l_j|\leqslant1$，故有

$$\min\{\sum_{i=1}^{n}l_i\}=(n-2m)k+2m(k+1)$$

请读者注意，当叶节点数 $n=2^k+m$ 时，与树根节点距离为 k 的内点必有 m 个，而距离为 $k+1$ 的叶节点有 $2m$ 个，与树根距离为 k 的叶节点应为 $2^k-m=n-2m$ 个，而且 $k=\lceil\log_2 n\rceil-1$，与树的根节点距离为 $\lceil\log_2 n\rceil$ 的节点最多可达 $2^{\lceil\log_2 n\rceil}$。

$$\begin{aligned}
&(n-2m)k+2m(k+1)=nk+2m\\
&=n(k+1)+2m-n=n(k+1)+n-2(n-m)\\
&=n(k+1)+n-2\cdot2^k=n\lceil\log_2 n\rceil+n-2^{\lceil\log_2 n\rceil}
\end{aligned}$$

这就是说：

5. 对于具有 n 个叶节点的二分树，下面的等式成立。

$$\min_{T}\{\sum_{i=1}^{n}l_i\}=n\lceil\log_2 n\rceil+n-2^{\lceil\log_2 n\rceil}$$

6.2 最佳二分树

在讲分治策略的时候已提到过二分查找的问题。已知序列：

$$x_1<x_2<\cdots<x_n$$

及 z，如若 z 在这序列中，设 $x_k=z$，找出 x_k，如若 z 不在该序列中，给出相应的信息。

序列 $x_1,x_2,\cdots,x_n$ 可以是各种类型的数据项，比如给出 Sun，Mon，Tue，Wed，Thu，Fri，Sat。如图 6.2.1 所示。图中左子树表示比树根小的数据项，右子树表示比树根大的数据项。则上述序列依其字典顺序可表示如图 6.2.2 所示。最后得二分树(图 6.2.3)。

图 6.2.3 提供了一种查找策略，图中的叶节点是查找失败的状态，内点是名字表。

给定 n 个数据项的序列，可用具有 n 个内点 $n+1$ 个叶节点的二分树来表示，但不唯一，比如图 6.2.4 是另外一种表示方法。

这一节看看 $x_1,x_2,\cdots,x_n$ 查找的几率各不相同的情况。

图 6.2.1

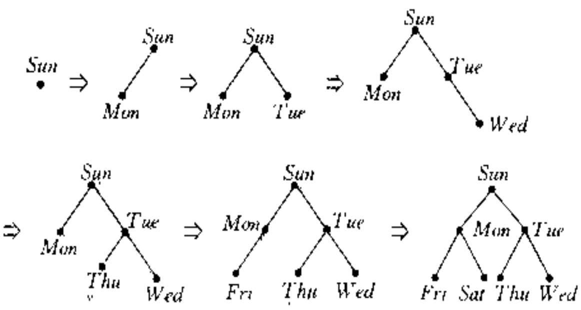


图 6.2.2

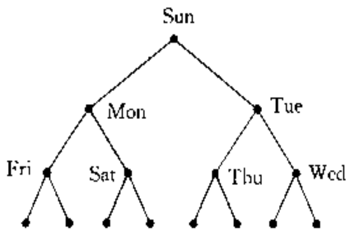


图 6.2.3

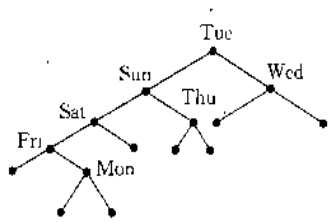


图 6.2.4

设 $x_1,x_2,\cdots,x_n$ 的查找频率分别为 $p_1,p_2,\cdots,p_n$。由于 $z<x_1$，$x_1<z<x_2$，$x_2<z<x_3,\cdots,x_{n-1}<z<x_n$，$x_n<z$ 而出现查找失败的几率分别为 $q_0,q_1,\cdots,q_n$。显然

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

序列 $x_1<x_2<\cdots<x_n$ 用一棵二分树 T 来表示，T 有 n 个内点正好分别是 $x_1,x_2,\cdots,x_n$，这棵二分树有 $n+1$ 个叶节点代表查找失败的 $n+1$ 种状态。有 n 个内点 $n+1$ 个叶节点的不同的二分树表达不同的查找策略。最佳策略对应的二分树叫做最佳二分树。在给出最佳的标准之前，先给出层次的定义如下：

点 x_i 到树根的距离用 $l(x_i)$ 表示，显然从树根开始查到 x_i 所作的比较次数应为 $l(x_i)+1$。

设 $y_0,y_1,\cdots,y_n$ 为二分树的叶节点，但确定出现查找失败现象 y_j，只须作 $l(y_j)$ 次比较。对应于二分树 T 定义 $C(T)$ 如下：

$$C(T) = \sum_{i=1}^{n}(l(x_i)+1)p_i + \sum_{j=0}^{n} q_j l(y_j)$$

$C(T)$ 称为 T 的带权的路径长度，也叫做平均查找次数。

给定 $x_1,x_2,\cdots,x_n$ 及其概率 $p_1,p_2,\cdots,p_n$ 以及 $y_0,y_1,\cdots,y_n$ 的概率 $q_0,q_1,\cdots,q_n$，存在一棵二分树 T^* 使 $C(T^*)$ 达到最小值时，称 T^* 为最佳(查找)二分树。

例 图 6.2.5 中 x_1,x_2,x_3 的查找频率分别为 0.20，0.25 和 0.15，失败的频率都是 0.10，则分别有：

$$C(T_1)=(0.15+0.10)+2(0.25+0.10)+3(0.20+0.10+0.10)=2.15$$

$C(T_2)=0.25+2(0.20+0.15)+2(0.10+0.10+0.10+0.10)=1.75$

$C(T_3)=(0.20+0.10)+2(0.15+0.10)+3(0.25+0.10+0.10)=2.15$

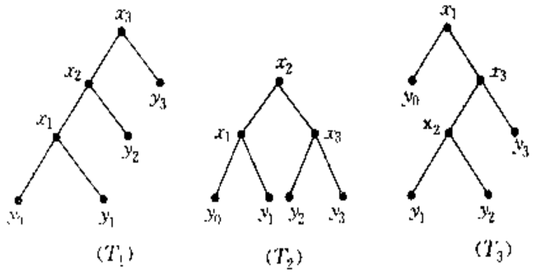


图 6.2.5

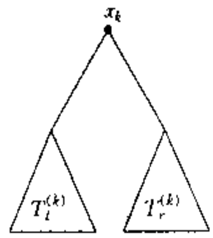


图 6.2.6

如图 6.2.6,设 $T_l^{(k)}$,$T_r^{(k)}$ 分别是最佳二分树的左右子二分树,这个最佳二分树的根节点设为 x_k,$T_l^{(k)}$ 包含了顶点 $x_1,x_2,\cdots,x_{k-1}$,$T_r^{(k)}$ 包含了顶点 $x_{k+1},x_{k+2},\cdots,x_n$。则有

$$C(T)=C(T_l^{(k)})+C(T_r^{(k)})+p_k+w_{0,k-1}+w_{k,n}$$

其中:

$$w_{0,k-1}=\sum_{i=1}^{k-1}p_i+\sum_{i=0}^{k-1}q_i$$

$$w_{k,n}=\sum_{i=k+1}^{n}p_i+\sum_{i=k}^{n}q_i$$

而且 $$C(T^*)=\min\{C(T_l^{(k)})+C(T_r^{(k)})\}+w_{0,k-1}+w_{k,n} \tag{6.1}$$

$T_l^{(k)}$ 表示以 k 点为根节点的左子二分树;同理 $T_r^{(k)}$ 为以 k 为根节点的右子二分树。$C(T_l^{(k)})$、$C(T_r^{(k)})$分别是顶点 x_k 的左右子二分树 $T_l^{(k)}$,$T_r^{(k)}$ 的带权路径长度。

从上可知若 T 为最佳二分树,则 $T_l^{(k)}$ 和 $T_r^{(k)}$ 也一定是最佳二分树。如若不然,只要代以对应的最优二分树,使 $C(T)$ 下降,与 T 是最佳二分树的假设矛盾。另一方面

$$\begin{aligned}
&p_k+w_{0,k-1}+w_{k,n}\\
&=p_k+\sum_{i=1}^{k-1}p_i+\sum_{i=0}^{k-1}q_i+\sum_{i=k+1}^{n}p_i+\sum_{i=k}^{n}q_i\\
&=\sum_{i=1}^{n}p_i+\sum_{i=0}^{n}q_i
\end{aligned}$$

可将(6.1)推广到一般的情况。若包含内点 $x_i,x_{i+1},\cdots,x_j$ 的最佳二分树是以 x_k 点为根节点。则有

$$C(T_{i,k-1}^{(k)})+C(T_{k,j}^{(k)})=\min_{i<k\leqslant j}\{C(\overset{*}{T}{}_{i,k-1}^{(k)})+C(\overset{*}{T}{}_{k,j}^{(k)})\} \tag{6.2}$$

其中假定$\overset{*}{T}{}_{i,k-1}^{(k)}$为包含 $x_{i+1},x_{i+2},\cdots,x_{k-1}$的最佳左子树(相应包含了叶子节点 $y_i,y_{i+1},\cdots,y_{k-1}$),$\overset{*}{T}{}_{k,j}^{(k)}$为包含点 $x_{k+1},x_{k+2},\cdots,x_j$ 的最佳右子树(相应包含了叶子节点 $y_k,y_{k+1},\cdots,y_j$)。式(6.3.2)提供了求出包含点 $x_{i+1},x_i,\cdots,x_j$ 的最佳二分树的顶点 x_k 的一种途径,$i<k\leqslant j$。当 x_k 求得后,则问题导致求左子树 $T_l^{(k)}(=T_{i,k-1})$和右子树 $T_r^{(k)}(=T_{k,j})$的最佳二分树。左子树为包含顶点 $x_{i+1},x_{i+2},\cdots,x_{k-1}$的二分树,右子树为包含顶点 $x_{k+1},x_{k+2},\cdots,x_j$

的二分树。

可利用动态规划中的最佳原理求最佳二分树,从含 1 个顶点,2 个顶点开始,…,直到含有 n 个顶点的最佳二分树。

设 $w_{ij} = \sum_{k=i}^{j} q_k + \sum_{k=i+1}^{j} p_k$

下面举例说明如何利用动态规划求解。

例 $n=4$ 的例子,设已知序列 $x_1<x_2<x_3<x_4$,

$p_1=1/16, p_2=3/16, p_3=2/16, p_4=2/16, q_0=2/16$

$q_1=1/16, q_2=3/16, q_3=q_4=1/16$。

为了方便起见,下面计算过程都只取分子,即 $p_1=1, p_2=3$,等等。

从 $C(T_{i,i})=0, w_{i,i}=q_i$ 开始

1. 有一个内点的情况

(1) 有一个内点 x_1 的是

$$T_{0,1}:\quad \begin{array}{ccc} & x_1(p_1) & \\ \diagup & & \diagdown \\ (q_0) & & (q_1) \end{array}$$

图中()里的数是对应于该点的查找几率,以后不再说明

即
$$w_{0,1}=p_1+q_0+q_1=4$$
$$C(T_{0,1})=w_{0,1}+[C(T_{0,0})+C(T_{1,1})]=4$$

(2) $w_{1,2}=p_2+q_1+q_2=7$
$$C(T_{1,2})=w_{1,2}+[C(T_{1,1})+C(T_{2,2})]=7$$

或

$$T_{1,2}:\quad \begin{array}{ccc} & x_2(p_2) & \\ \diagup & & \diagdown \\ (q_1) & & (q_2) \end{array}$$

(3)
$$w_{2,3}=p_3+q_2+q_3=6$$
$$C(T_{2,3})=w_{2,3}+[C(T_{2,2})+C(T_{3,3})]=6$$

$$T_{2,3}:\quad \begin{array}{ccc} & x_3(p_3) & \\ \diagup & & \diagdown \\ (q_2) & & (q_3) \end{array}$$

(4) 同理

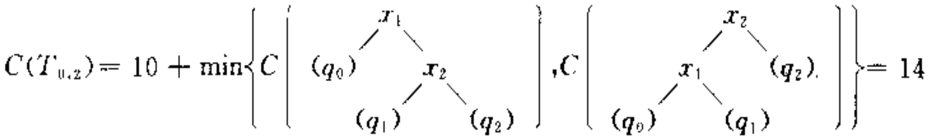
$$T_{3,4}:\quad \begin{array}{ccc} & x_4(p_4) & \\ \diagup & & \diagdown \\ (q_3) & & (q_4) \end{array}$$

$$C(T_{3,4}) = w_{3,4} = p_4 + q_3 + q_4 = 4$$

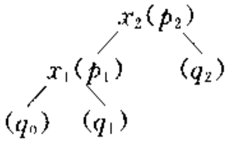
2. 有两个内点的情形

(1)
$$w_{0,2}=q_0+(p_1+p_2+q_1+q_2)=10$$
$$C(T_{0,2})=w_{0,2}+\min\{C(T^{(1)}_{0,0})+C(T^{(1)}_{1,2}), C(T^{(2)}_{0,1})+C(T^{(2)}_{2,2})\}$$
$$=10+\min\{7,4\}=14$$

或形象地表为



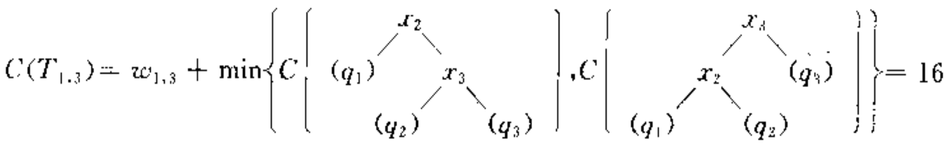
所以 $T_{0,2}$:



(2) $$w_{1,3} = q_1 + \sum_{i=2}^{3}(p_i + q_i) = 10$$

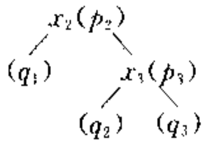
$$C(T_{1,3}) = w_{1,3} + \min\{C(T_{1,1}^{(2)}) + C(T_{2,3}^{(2)}), C(T_{1,2}^{(3)}) + C(T_{3,3}^{(3)})\}$$
$$= 10 + \min\{6^*, 7\} = 16$$

或形象地表为



故

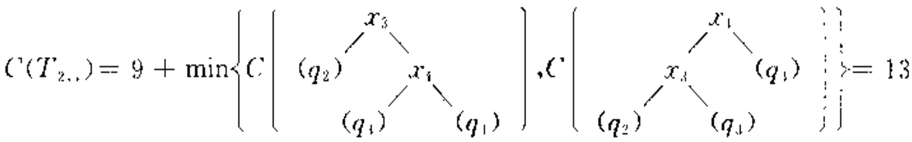
$T_{1,3}$:



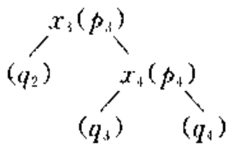
(3) $$w_{2,4} = q_2 + \sum_{i=3}^{4}(p_i + q_i) = 9$$

$$C(T_{2,4}) = w_{2,4} + \min\{C(T_{2,2}^{(3)}) + C(T_{3,4}^{(3)}), C(T_{2,3}^{(4)}) + C(T_{4,4}^{(4)})\}$$
$$= 9 + \min\{4, 6\} = 13$$

或形象地表为



$T_{2,4}$:

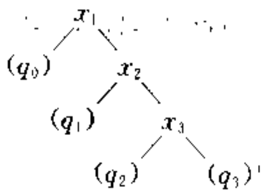


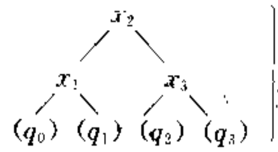
3. 含有三个内点的情形

(1) $$w_{0,3} = q_0 + \sum_{i=1}^{3}(p_i + q_i) = 13$$

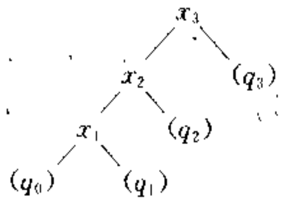
$$C(T_{0,3}) = w_{0,3} + \min\{C(T_{0,0}^{(1)}) + C(T_{1,3}^{(1)}), C(T_{0,1}^{(2)}) + C(T_{2,3}^{(2)}), C(T_{0,2}^{(3)}) + C(T_{3,3}^{(3)})\}$$
$$= 13 + \min\{16, 10^*, 14\}$$
$$= 23$$

或形象地表示为

$C(T_{0,3}) = 13 + \min\{C($


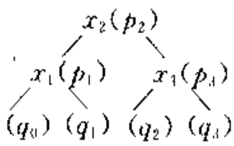
$), C($


$),$

$C($


$)\} = 23$

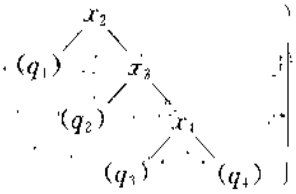
故

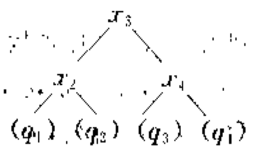
$T_{0,3}$:


(2)
$$w_{1,4} = q_1 + \sum_{i=2}^{4}(p_i + q_i) = 13$$

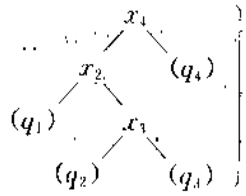
$$C(T_{1,4}) = w_{1,4} + \min\{C(T_{1,1}^{(2)}) + C(T_{2,4}^{(2)}), C(T_{1,2}^{(3)}) + C(T_{3,4}^{(3)}), C(T_{1,3}^{(4)}) + C(T_{4,4}^{(4)})\}$$
$$= 13 + \min\{13, 11, 16\}$$
$$= 24$$

或形象地表示为

$C(T_{1,4}) = 13 + \min\{C($


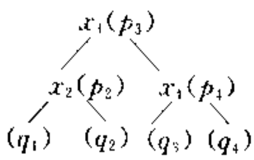
$), C($


$),$

$C($


$)\} = 24$

故

$T_{1,4}$:


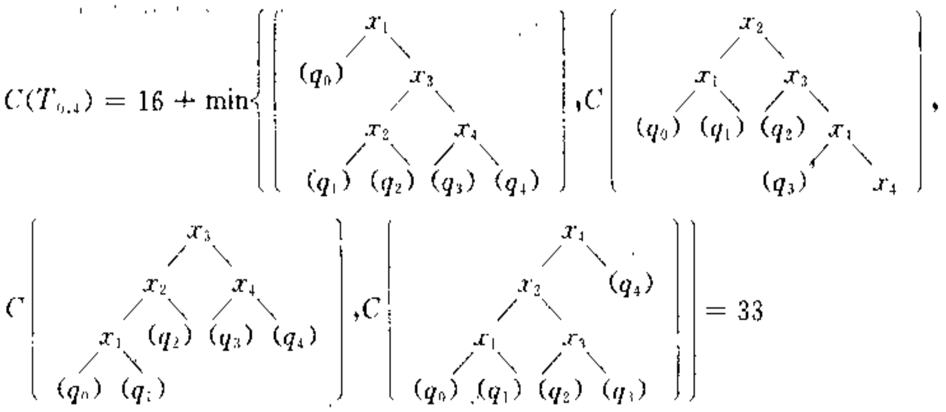
4. 含有四个内点的情形

$w_{0,4} = 16$

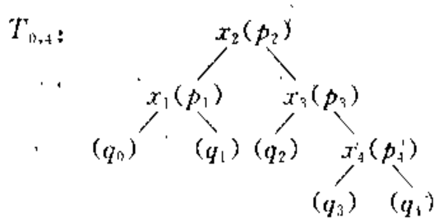
$$C(T_{0,4}) = 16 + \min\{C(T_{0,0}^{(1)}) + C(T_{1,4}^{(1)}), C(T_{0,1}^{(2)}) + C(T_{2,4}^{(2)}), C(T_{0,2}^{(3)}) + C(T_{3,4}^{(3)}),$$
$$C(T_{0,3}^{(4)}) + C(T_{4,4}^{(4)})\}$$

$$= 16 + \min\{24, 17^*, 18, 23\}$$
$$= 33$$

或形象地表示为



故



习　题

1. 利用动态规划原理构造最佳二分树。节点数 $n=5$，p_1, p_2, p_3, p_4, p_5 分别为 0.20，0.15，0.10，0.05，0.10，$q_0, q_1, q_2, q_3, q_4, q_5$ 分别为 0.05，0.10，0.10，0.05，0.05，0.05。

2. 构造最佳二分树，节点数 $n=4$，p_1, p_2, p_3, p_4 依次为 0.20，0.10，0.20，0.10；q_0, q_1, q_2, q_3, q_4 依次为 0.05，0.1，0.1，0.05，0.1。

3. 写出最佳二分树构成的算法，并分析其复杂度(用类 PASCAL 语言)。

4. 已知 15 个关键字 k_i，$i=1,2,\cdots,15$，其查找概率依次为 0.05，0.15，0.025，0.025，0.05，0.05，0.075，0.025，0.15，0.05，0.05，0.125，0.075，0.025，0.075，试求最佳二分树，使平均查找次数最小，并求平均查找次数。

第 7 章　内存分类法之一：插入分类法、Shell 分类法

7.1　分　　类

分类(有的也叫排序)是计算机科学的一种非常重要的问题。因为它是计算机经常遇到的工作，特别在一些特殊的数据处理系统中分类占有着相当大的运行时间，而且实际工作中要求各异，环境也千差万别，所以分类算法也因地而异。因此有效的算法一直是人们感兴趣的问题。

给定 n 个记录 $R_1, R_2, \cdots, R_n$，其相应的关键字分别是 $k_1, k_2, \cdots, k_n$，分类就是确定 1，2，…，n 的一种排列 $P_1, P_2, \cdots, P_n$，使其相应的关键字满足如下的非递减(或非递增)关系

$$k_{P_1} \leqslant k_{P_2} \leqslant \cdots \leqslant k_{P_n}$$

从而使 n 个记录成为一个按关键字有序的序列，

$$\{R_{P_1}, R_{P_2}, \cdots, R_{P_n}\}$$

由于待分类的记录数量不同，可将分类法分为两类，一类是内存分类，指的是待分类的关键字可放在计算机随机存储器中进行分类过程，当然关键字的数量有一定的限制。另一类是外存分类，指的是待分类的关键字的数量很大，以至内存一次不能容纳它们全部，在分类过程中尚需对外存，如磁带、磁盘，进行访问的分类过程。下面讨论内存分类。

内存分类的方法很多，但影响分类的因素比较多，所以就其全面性能而言，很难提出一种被认为是最好的方法，每种方法都有各自的优缺点，适合于各自特定的环境下使用。详细的总结可见于 Knuth 在他的 The Art of Computer Programming 第三卷。本书按分类过程中基本方法的不同，仅仅介绍内存分类方法中的插入分类、交换分类、选择分类、归并分类和基数分类等几种。仅就几个典型的算法并加以分析。读者在学习时，除了掌握算法本身外，更重要的是了解该算法在进行分类时所依据的原则，以利于学习和创造更新的算法。

通常，在分类的过程中需进行以下两种基本操作：① 比较两关键字大小；② 将记录从一个位置移到另一个位置。在分类算法中应同时考虑这两种操作的复杂性。也就是说在分类过程中必须同时考虑这两种基本操作。

7.2　分类的下界估计

在绝大多数情况下，分类算法都是基于“关键字间的比较”这个操作进行的，可用类似于图 7.2.1 所示的判定树来描述这些分类方法的过程。

图 7.2.1 的判定树表示三个关键字分别为 k_1，k_2 和 k_3 的记录进行排序的所有可能过程，树中每个非叶结点表示两个关键字间的一次比较，其左、右子树分别表示这次比较所

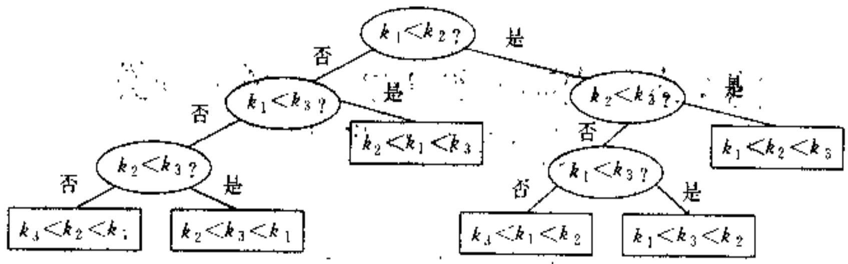


图 7.2.1

得的两种结果。假设 k_1,k_2,k_3 互不相等，它们之间只可能有下列 6 种关系：① $k_1<k_2<k_3$；② $k_1<k_3<k_2$；③ $k_3<k_1<k_2$；④ $k_2<k_1<k_3$；⑤ $k_2<k_3<k_1$，⑥ $k_3<k_2<k_1$，换句话说，这三个记录经过分类只可能得到下列六种可能结果：① $\{R_1,R_2,R_3\}$；② $\{R_1,R_3,R_2\}$；③ $\{R_3,R_1,R_2\}$；④ $\{R_2,R_1,R_3\}$；⑤ $\{R_2,R_3,R_1\}$；⑥ $\{R_3,R_2,R_1\}$。这里$\{R_1,R_2,R_3\}$表示三个记录 R_1,R_2,R_3 依次存放顺序，余此类推。而图 7.2.1 中的判定树上 6 个叶结点恰好表示这 6 种分类结果。并且，对每一个初始序列经分类达到有序序列所需进行的"比较"次数，恰为从树根到和该序列相应的叶节点的路径长度。由于图 7.2.1 的判定树的深度为 4，则对 3 个记录进行分类最坏的情况下至少要进行三次比较。

推广至一般情况下，对 n 个记录进行分类至少需进行多少次关键字间的比较？由于含 n 个关键字的序列可能出现的状态有 $n!$ 个，则描述 n 个记录分类过程的判定树必须有 $n!$ 个叶子节点。而我们知道，若二分树的高度为 h，则叶节点的个数不超过 2^{h-1}；反之若有 u 个叶节点，则二分树的高度至少为$\lceil \log_2 u \rceil$。对于描述 n 个记录分类的判定树上必定存在一条长度为$\lceil \log_2 n! \rceil$的路径。也就是说至少存在一种情况至少要作$\lceil \log_2 n! \rceil$次比较。这是排序的下界。说得确切些，在最坏情况下必须做$\lceil \log_2 n! \rceil$次比较。

上面的讨论可归结为这样一个模型，即有一个含有 $m=n!$ 个元素的集合 A，甲从中任取一个，让乙来猜，但允许乙提出 k 个"是"或"非"问题。问在最坏情况下 k 应该是多少？

乙提出第 1 个"是"或"非"问题，可将集合 A 分为 $A_1^{(1)}$ 和 $A_2^{(1)}$ 两个子集合，其中必有一个集合(设为 $A_1^{(1)}$)，它包含的元素个数(用$|A_1^{(1)}|$来表示)不少于$\frac{m}{2}$，即

$$|A_1^{(1)}|\geqslant\frac{m}{2}$$

若甲所取的元素正好在 $A_1^{(1)}$，乙提出第 2 个"是"或"非"问题后将集合 $A_1^{(1)}$ 分为 $A_1^{(2)}$ 和 $A_2^{(2)}$，其中之一设为 $A_1^{(2)}$ 有

$$|A_1^{(2)}|\geqslant\frac{1}{2}|A_1^{(1)}|\geqslant\frac{m}{2^2}$$

依此类推有

$$|A_1^{(k)}|\geqslant\frac{m}{2^k}$$

k 必须足够大，使得

$$\frac{m}{2^k}\leqslant 1$$

则乙可在最坏情况下通过 k 次提问题找到所要寻找的元素，即猜到 A 取的元素，这里"是"或"非"问题相当于作一次比较结果有＞＜两种状态。

$$2^k \geqslant n!$$

$$k \geqslant \log_2 n!$$

由此得到下述结论：

任何一个借助"比较"进行分类的算法，在最坏情况下所需的比较次数至少为 $\lceil \log_2 n! \rceil$。由 Stirling 公式：

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$\log_2\left[\sqrt{2n\pi}\left(\frac{n}{e}\right)^n\right] = n\left(\log_2 n - \log_2 e + \frac{1}{2}\log_2 n + \frac{1}{2}\log_2 2\pi\right)$$

$$= O(n\log_2 n)$$

即分类的下界为 $O(n\log_2 n)$。也就是说任何内存分类法在最坏情况下所需的比较次数不得少于 $O(n\log_2 n)$。

例 对序列 49，38，45，13，40，50，35 利用插入分类法进行分类，过程如下：

$-\infty$ 49 38 45 13 40 50 35

$j=2$：$-\infty$ 38 49 45 13 40 50 35

$j=3$：$-\infty$ 38 45 49 13 40 50 35

$j=4$：$-\infty$ 13 38 45 49 40 50 35

$j=5$：$-\infty$ 13 38 40 45 49 50 35

$j=6$：$-\infty$ 13 38 40 45 49 50 35

$j=7$：$-\infty$ 13 35 38 40 45 49 50

从上可见，简单插入分类的算法简洁，容易实现，那么它的效率如何呢？

设 n 个记录分类所作的比较次数为 $T(n)$，可记为 T_n，则对于简单插入分类有：

1. 最好的情况：(即 $x_1<x_2<x_3<\cdots<x_n$ 时)

$$T_n = T_{n-1}+1, T(1)=0,$$

令 $G(x)=T_1+T_2x+T_3x^2+\cdots$

$$(1-x)G(x)=x/1-x$$

$$\therefore\ G(x)=\frac{x}{(1-x)^2}=x(1+2x+3x^2+\cdots)$$

$$T_n = n-1$$

2. 最坏的情况，(即 $x_1>x_2>\cdots>x_n$ 时)

$$T_n=T_{n-1}+n, T(1)=1$$

令 $G(x)=T_1+T_2x+T_3x^2+\cdots$

$$(1-x)G(x)=1+2x+3x^2+\cdots\cdots$$

$$\therefore \quad G(x)=(1+2x+3x^2+\cdots\cdots)/(1-x)$$

$$=(1+2x+3x^2+\cdots)(1+x+x^2+\cdots)$$

$$T_n=1+2+\cdots\cdots+n=\frac{1}{2}n(n+1)$$

3. 平均估计

$$T_n=T_{n-1}+\frac{n+1}{2}$$

$$T_n=\frac{n}{4}(n+3)$$

故其时间复杂性为 $O(n^2)$。

7.3 二分插入分类法

已知一个序列 $x_1<x_2<\cdots<x_n$，给定一个 y，y 和所给的 $x_1,x_2,\cdots,x_n$ 均不相等，要求将 y 插入到它应有的位置。上面介绍的简单插入分类采取的是逐个比较的算法。假定 $x_1<x_2<\cdots<x_n$ 已排好，y 的插入策略是和一棵二分树相对应。以 $n=5$ 为例。已知 $x_1<x_2<x_3<x_4<x_5$。逐个比较法的策略对应图 7.3.1 所示的判决树。

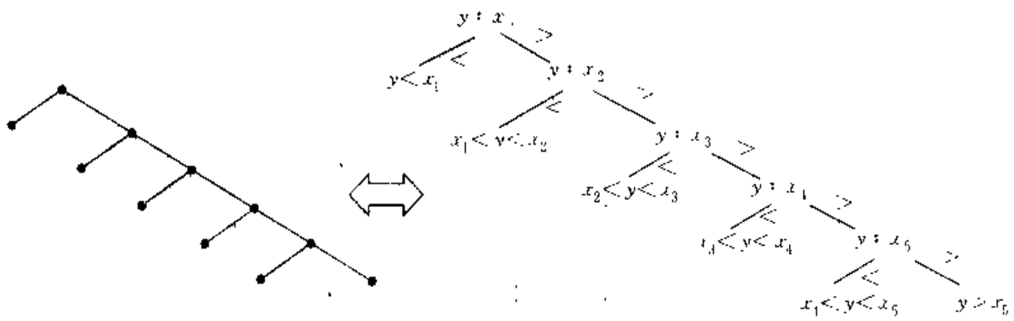


图 7.3.1

一般说来，n 个叶节点 $x_1,x_2,\cdots,x_n$ 的二分树，它的 $n-1$ 个内点可分别给以序号，使得每个顶点的序号大于它的"左儿子"，而小于它的"右儿子"，这样的一棵二分树对应一个查找插入策略。图 7.3.2 和图 7.3.3 便是不同策略对应的判决树。判决树的叶子节点给出 y 的插入位置。

在判决树已定的条件下利用插入法进行分类，在最坏的情况下要作 l_{max} 次的比较。二分插入分类是插入分类的一种。二分插入分类是从二分查找演变得来的。

一种算法要有一种相应的数据结构来保证。链表式的数据结构不适宜于二分查找，因中间元素的地址不容易得知，若采用顺序存储则二分查找易实现，但是对于二分插入分类

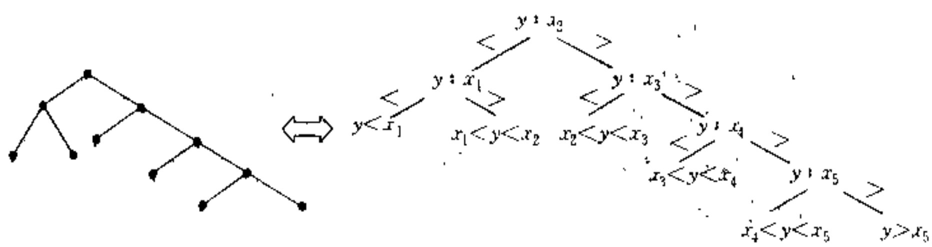


图 7.3.2

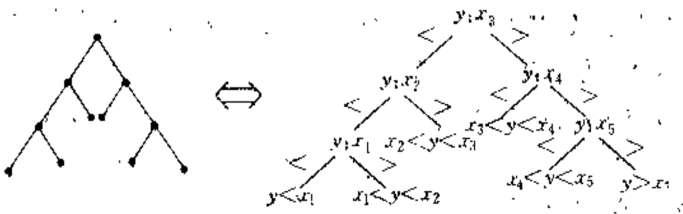


图 7.3.3

法，若采用顺序存储的方法，每次插入将会引起平均约一半的元素改变地址，主要的运算不是"比较"，而是"数据搬家"。

为了兼顾二分插入中间元素的寻找和少移动数据，可引入一种均衡树结构，关于均衡树的具体讨论留在以后。这里仅就二分插入分类法估计其所作比较次数的下界。二分插入排序是当前面 $k-1$ 个元素排列完毕后，第 k 个元素插入采用了二分查找的方法。在最坏情况下，第 k 元素插入需要作 $\lceil \log_2 k \rceil$ 次比较。故对于 n 个元素的分类排序采用二分插入分类法所需比较次数为：

$$S \leqslant \sum_{k=1}^{n} \lceil \log_2 k \rceil = n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1$$

证明 设 $n=2^m$ 时

$$\begin{aligned}
S = &\lceil \log_2 2 \rceil + (\lceil \log_2 3 \rceil + \lceil \log_2 4 \rceil) + (\lceil \log_2 5 \rceil + \lceil \log_2 6 \rceil + \lceil \log_2 7 \rceil \\
&+ \lceil \log_2 8 \rceil) + \cdots + \left(\left\lceil \log_2\left(\frac{n}{2}+1\right) \right\rceil + \left\lceil \log_2\left(\frac{n}{2}+2\right) \right\rceil\right. \\
&\left. + \cdots + \lceil \log_2 n \rceil\right) \\
= &1 + 2.2^1 + 3.2^2 + \cdots + \lceil \log_2 n \rceil \cdot 2^{\lceil \log_2 n \rceil - 1} \\
= &1 + 2.2^1 + 3.2^2 + \cdots + m2^{m-1}
\end{aligned}$$

由于

$$1 + x^1 + x^2 + \cdots + x^m = \frac{1-x^{m+1}}{1-x}$$

$$1 + 2x + 3x^2 + \cdots + mx^{m-1} = \left(\frac{1-x^{m+1}}{1-x}\right)'$$

$$= \frac{(m+1)(x-1)x^m + (1-x^{m+1})}{(1-x)^2} = \frac{mx^{m+1} - (m+1)x^m + 1}{(1-x)^2}$$

$$\therefore \quad S = \left.\frac{mx^{m+1} - (m+1)x^m + 1}{(1-x)^2}\right|_{x=2} = m2^{m+1} - (m+1)2^m + 1$$

$$= 2\lceil \log_2 n \rceil \cdot 2^{\lceil \log_2 n \rceil} - (\lceil \log_2 n \rceil + 1)2^{\lceil \log_2 n \rceil} + 1$$

$$= n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1$$

7.4 Shell 分类法

1. Shell 算法

希尔(Shell)分类法又称"缩小增量分类法"。它也是一种属插入分类的方法,但在时间效率上较前几种分类方法有较大的改进。

首先介绍 h-分类法,其基本思想是把 n 个关键字 $x_1,x_2,\cdots,x_n$ 分成 h 类如下:

$$S_k^{(h)} = \{x_k, x_{k+h}, x_{k+2h}, x_{k+3h}, \cdots\} \quad k=1,2,\cdots,h$$

对 $S_1^{(h)},S_2^{(h)},\cdots,S_h^{(h)}$ 各自进行排序的结果称为 h 分类。

现在我们通过举例非形式化说明 Shell 分类法。

若分类的序列有 16 个元素:

56,96,20,53,63,68,99,39,15,90,98,15,8,18,80,49,分步完成。

第 1 步 $h_1=6$,即距离为 6 的元素分成一组,共 6 组如下:

56,99,8; 96,39,18; 20,15,80; 53,90,49; 63,98; 68,15。

第 2 步为 $h_2=4$,在上述 6 组分别分类完毕得序列

8,18,15,49,63,15,56,39,20,53,98,68,99,96,80,90。

在此基础上,依距离为 4 的分为一组,共分 4 组:

8,63,20,99; 18,15,53,96; 13,56,98,80; 49,39,68,90。

各组排序得 8,15,13,39,20,18,56,49,63,53,80,68,99,96,98,90。

第 3 步为 $h_3=2$,最后 $h_4=1$。其过程如图 7.4.1 所示。

Shell 法是找一减序列

$$h_t > h_{t-1} > \cdots > h_2 > h_1 = 1$$

先对 $x_1,x_2,\cdots,x_n$ 作 h_t-分类,接着作 h_{t-1}-分类,…,h_1-分类。每一 h 分类用插入法较适宜,特别是后来序列已接近于排序完毕,仅需适当调整,更是如此。Shell 分类的算法如下:

(1) $S \leftarrow t$。

(2) 若 $S \geqslant 1$ 则转(3),否则结束。

(3) $h \leftarrow h_s, j \leftarrow h+1$。

(4) 若 $j \leqslant n$,则转(5);

否则作[$S \leftarrow S-1$,转(2)]。

(5) $i \leftarrow j-h$, $k \leftarrow X(j)$。

(6) 若 $k > X(i)$,则转(9);否则转(7)。

(7) $X(i+h) \leftarrow X(i), i \leftarrow i-h$。

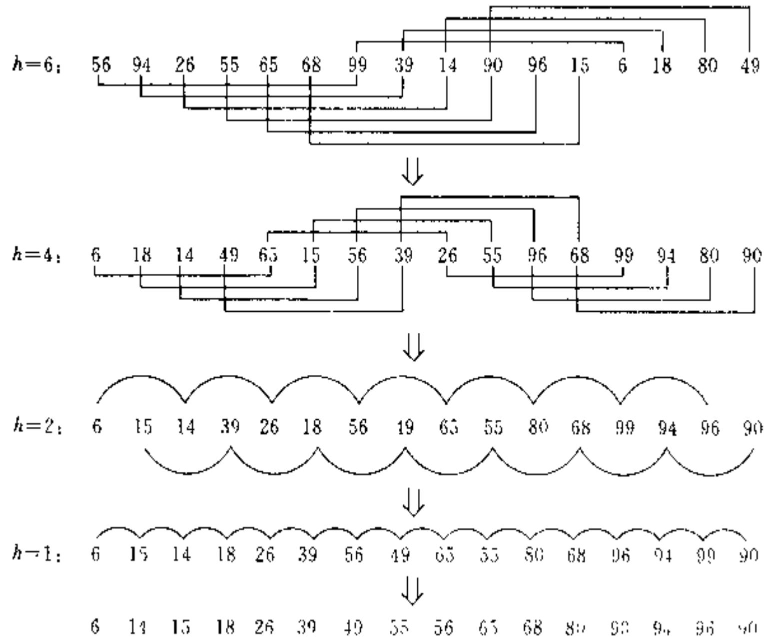


图 7.4.1

(8) 若 $i>0$ 则转(6)；否则转(9)。

(9) $X(i+h)\leftarrow k, j\leftarrow j+1$，转(4)。

2. Shell 算法的依据

Shell 算法的依据是：序列 $x_1,x_2,\cdots,x_n$ 在作 h-分类后，再进行 k-分类，$k<h$，下列关系保持不受影响。

$$x_k<x_{k+h}<x_{k+2h}<\cdots, k=1,2,\cdots,h$$

引理 序列 $x_1,x_2,\cdots,x_{m+r}$ 经排序得 $x_{(1)}<x_{(2)}<\cdots<x_{(m+r)}$，序列 $y_1,y_2,\cdots,y_{n+r}$ 经排序得：$y_{(1)}<y_{(2)}<\cdots<y_{(n+r)}$，若 $y_i<x_{m+i}$，$i=1,2,\cdots,r$，则

$$y_{(i)}<x_{(m+i)}, i=1,2,\cdots,r$$

这个引理可用图 7.4.2 表示如下。若 $r=4$，即若 $y_1<x_5$，$y_2<x_6$，$y_3<x_7$，$y_4<x_8$，则经过排序后的两个序列仍然有(图 7.4.3)：

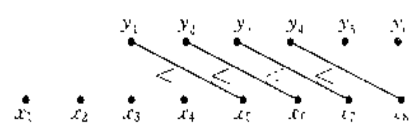


图 7.4.2

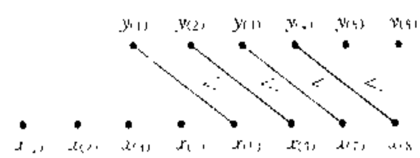


图 7.4.3

引理的证明

$y_1,y_2,\cdots,y_r$ 在排序后的序列 $y_{(1)}<y_{(2)}<\cdots<y_{(n+r)}$ 中依其从小到大的顺序为：

$$y_{(j_1)}<y_{(j_2)}<\cdots<y_{(j_r)} \tag{7.4.1}$$

显然有

$$y_{(1)}\leqslant y_{(j_1)},\ y_{(2)}\leqslant y_{(j_2)},\ \cdots\cdots,\ y_{(r)}\leqslant y_{(j_r)} \tag{7.4.2}$$

类似的道理，$x_{m+1}, x_{m+2}, \cdots, x_{m+r}$在排序后的序列 $x_{(1)}<x_{(2)}<\cdots<x_{(m+r)}$中依其顺序排列得

$$x_{(k_1)}<x_{(k_2)}<\cdots<x_{(k_r)} \tag{7.4.3}$$

显然有：

$$x_{(k_1)}\leqslant x_{(m+1)},\ x_{(k_2)}\leqslant x_{(m+2)},\ \cdots,\ x_{(k_r)}\leqslant x_{(m+r)} \tag{7.4.4}$$

由于(7.4.1)是 $y_1, y_2, \cdots, y_r$ 的排序，(7.4.3)是 $x_{m+1}, x_{m+2}, \cdots, x_{m+r}$的排序，根据假定 $y_i<x_{m+i}$，$i=1,2,\cdots,r$。故有

$$y_{(j_l)}<x_{(k_l)},\ l=1,2,\cdots,r$$

由(7.4.2)和(7.4.4)可得：

$$y_{(l)}<x_{(m+l)},\ l=1,2,\cdots,r$$

引理证毕。

例如在 5 分类的基础上，再进行 3-分类(图 7.4.4)：

由于已经通过了 5-分类，故 $y_1<x_1, y_2<x_2, y_3<x_3, \cdots$，在此基础上再进行 3-分类得(图 7.4.5)：

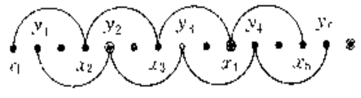


图 7.4.4

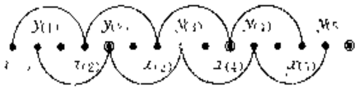


图 7.4.5

由引理得 $y_{(1)}<x_{(1)}, y_{(2)}<x_{(2)}, y_{(3)}<x_{(3)}$。即 3-分类的结果仍保持 5-分类后的特性。

满足 $h_t>h_{t-1}>\cdots>h_2>h_1=1$ 条件的序列

$$h_t, h_{t-1}, \cdots, h_2, h_1$$

的选择，Knuth 建议采用

$$h_1=1, h_{s+1}=3h_s+1$$

即 $\qquad h_j=(3^j-1)/2,\ 1\leqslant j\leqslant\lceil\log_3(2n-1)\rceil$

并得出时间复杂性经验公式为：$\alpha N(\log_2 N)^2+\beta N\log_2 N$。关于 Shell 算法的详细复杂性分析目前尚未完成。一般说来，对于基本有序的序列采用 Shell 算法来对付效果较好。

习　题

1. 我们看到当关键字按降序排列时，插入排序的效率是最坏的。试举出其它的初始排列次序使插入排序仍是最坏情况。

2. 考虑下面插入排序的变型：对于 $2\leqslant i\leqslant n$，插入 $L[i]$到 $L[1]\leqslant L[2]\leqslant\cdots\leqslant L[i-1]$时，做一个二分查找在找到 $L[i]$的正确位置。

(1) 最坏情况下有多少次比较。

(2) 最坏情况下有多少关键字的移动。

(3) 最坏情况下的关键字排列是什么?

(4) 能否通过将关键字从放在数组中改为放在联接表中来减少移动次数? 为什么?

3. 在分析插入排序时我们假定关键字是不同的,若考虑所有含相同关键字的序列时,平均情况会变好还是变坏? 为什么?

4. 给出一个直观的解释为什么 Shell 排序时间复杂性比 $\Theta(n^2)$ 低?

5. 为什么递减的 2 的幂为 Shell 排序参数是一个不好的选择?

6. 从二分查找树中删除的过程是否可交换,即先删除 x 再删除 y 与先删除 y 再删除 x 是否相等? 说明之或给出反例,对于插入操作又如何?

7. 假定在一个二分查找树中的树范围从 1 到 1000,现在要查找 363。下面哪一个序列不是真正的查找的序列。

(1) 2,252,401,398,330,344,397,363

(2) 924,220,911,244,898,258,362,363

(3) 925,202,911,240,912,245,363

(4) 2,399,387,219,266,382,381,278,363

(5) 935,278,347,621,299,392,358,363

8. 已知序列 $a_1, a_2, \cdots, a_n$,每个元素 a_i 都是正整数,并满足 $1 \leqslant a_i \leqslant k, i=1,2,\cdots,n$,其中 $k=O(n)$。试设计对这样的序列进行排序的算法,并讨论其复杂性。

9. 下列是针对问题 8 的算法,试举例说明并讨论其复杂性:

(1) i 从 1 到 k 作 $C[i] \leftarrow 0$。

(2) j 从 1 到 n 作

$C[A[j]] \leftarrow C[A[j]] + 1$。

(3) i 从 2 到 k 作

$C[i] \leftarrow C[i] + C[i-1]$

(4) j 从 n 降到 1 作

【$B[C[A[j]]] \leftarrow A[j]$,

$C[A[j]] \leftarrow C[A[j]] - 1$,】.

10. 假定 $a_1, a_2, \cdots, a_n$ 是均匀分布在[0,1)上的随机序列,试设计一分类算法,并讨论其复杂性。

11. 下面算法是对依序存贮于数组 A 中的序列,找出其中最大的元素和次大元素。

(1) 若 $A[1] > A[2]$ 则作

【max←$A[1]$,Sec←$A[2]$,i←3】,否则作

【max←$A[2]$,Sec←$A[1]$,i←3】,

(2) 若 $i \leqslant n$ 则作

【若 $A[i]>$Sec 则作

若 $A[i]>$max 则作

【Sec←max,max←$A[i]$】】否则作

Sec←$A[i]$。

(3) $i \leftarrow i+1$,转 2。

试讨论最坏情况下作多少次比较？平均比较数又是多少？

12. 试设计从 n 个元素中找出第三大元素的算法，并讨论其复杂性。

13. 已知 A 和 B 两个数组各为已排好序的 n 个元素，试设计一算法求这 $2n$ 个元素中的第 n 个元素，并讨论其复杂性。

14. 矩阵 $A=(a_{ij})_{n\times n}$ 的每一行元素都是单调增序列，每一列元素也是单调增序列。试设计一算法，判断一数 x 是否在 A 中？

第 8 章　内存分类法之二：递选分类法、堆集分类

8.1　递选分类法

递选分类法的基本思想十分直观，先从 n 个关键字中选出最小的一个，它自然是排序中的第一个元素。再从余下的 $n-1$ 个元素中找出最小的，它实际上是排序的第 2 个元素；依此类推反复进行，直到全部取走为止，排序结束。一般说来，从 $n-i$ 个余下的元素中，作 $n-i-1$ 次比较，可找出其中最小元素，它在排序中实为第 $i+1$ 个元素。所以递选分类法的步骤为，i 从 1 到 $n-1$ 进行如下的操作：通过 $n-i$ 次比较，从 $n-i+1$ 个元素中找出最小的，并将它和第 i 个元素互换，将它置于第 i 个元素的位置上。现描述算法如下，其中 j 是用以储存最小元素的下标。

(1) $i \leftarrow 1$。

(2) $j \leftarrow i$。

(3) $k \leftarrow j+1$。

(4) 若 $k \leqslant n$，则作

【若 $a(k) < a(j)$，则作 $j \leftarrow k$，转(5)】。否则，转(6)。

(5) $k \leftarrow k+1$，转(4)。

(6) 若 $j \neq i$，则作 $a(i) \leftrightarrow a(j)$。

(7) $i \leftarrow i+1$，若 $i \leqslant n$，则转(2)。否则结束。

算法(4)和(5)在于找出 $a(i)$，$a(i+1)$，$a(i+2)$，…，$a(n)$ 中的最小元素 $a(j)$，即将最小元素的下标存放在 j 中。(6)将 $a(j)$ 置于 $a(i)$ 的位置。

举例如下，已知序列

	3	1	6	0	5	4	8	2	9	7
$i=1$：	3	1	6	0	5	4	8	2	9	7
$i=2$：	0	1	6	3	5	4	8	2	9	7
$i=3$：	0	1	6	3	5	4	8	2	9	7
$i=4$：	0	1	2	3	5	4	8	6	9	7
$i=5$：	0	1	2	3	5	4	8	6	9	7
$i=6$：	0	1	2	3	5	4	8	6	9	7

$i=7$：0　1　2　3　4　5*　8　6　9　7

$i=8$：0　1　2　3　4　5　8　6　9　7

$i=9$：0　1　2　3　4　5　6　8　9　7

$i=10$：0　1　2　3　4　5　6　7　9　8

结果：0　1　2　3　4　5　6　7　8　9

其中＊表示$a(i)$本身就是$a(i),a(i+1),\cdots,a(n)$的最小元素。

容易看出简单递选分类过程中，所需进行移动存贮的操作次数较少，其最小值为“0”、最大值为n。然而无论关键字的初始排列如何，所需进行的关键字间比较次数相同，均为$n(n-1)/2$，因此，总的时间复杂度是$O(n^2)$。

由上述可见，分类的主要操作是进行关键字间的比较，因此改进该算法应从如何减少“比较”次数考虑。下面介绍树形递选分类。

8.2　二分树递选分类法

引进数据结构：作一个有n个叶节点的二分树，使从左到右每一叶节点分别对应于$x(1),x(2),\cdots,x(n)$。对于有同一“父亲”节点的两个节点进行比较，留下小的一个给它们的“父亲”节点，从左向右，自下而上继续这个过程，直到给树根一个数为止，这个数便是最小的一个，把这个被选的数取消，然后进行下一轮，直到所有元素都分类完毕为止。例如，从图8.2.1所示可以看出：

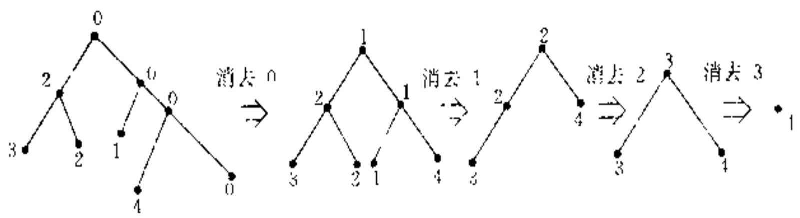


图　8.2.1

依次所得的结果为0，1，2，3，4。

复杂性分析：

首先是空间复杂性分析，利用树形递选分类法进行排序，所需要的存贮单元有：(1) n个对象即二分树的n个叶子节点；(2) $n-1$个内点；(3) 存放结果的n个单元，故约需$3n$个单元。

时间复杂性分析：每一个内点在分类的过程中都有先“蜕化”成树叶子，然后再“消

失"的过程。由于每个内点v_k有n_k个叶子，v_k蜕化成叶子节点，有n_k-1个叶子消失，故要作n_k-1次比较，最后留下一个。故全部比较总数为：

$$S = \sum_{k=1}^{n-1}(n_k - 1) = \sum_{k=1}^{n-1} n_k - (n-1)$$

但
$$\sum_{k=1}^{n-1} n_k = \sum_{i=1}^{n} l_i$$

且
$$\min \sum_{i=1}^{n} l_i = n\lceil \log_2 n \rceil + n - 2^{\lceil \log_2 n \rceil} + 1$$

所以
$$S \geqslant n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1$$

但前面讨论的简单递选分类时间复杂性为$O(n^2)$，其空间复杂性约为$2n$。与简单递选分类比较，二分树递选分类的时间复杂度较好，但空间复杂度较差($3n$)。

8.3 堆集分类法

二分树递选分类的时间复杂度为$O(n\log_2 n)$，但是这种分类方法辅助存贮空间较多。J. Willioms 在 1964 年提出了另一种形式的递选分类——堆集分类。堆集分类法利用了一种特定的数据结构——堆。下面首先介绍什么叫做堆。

1. 堆的构造

堆也是一种二分树，如图 8.3.1 所示。
每一个节点有两个数，一是"○"中的数，表示该节点存贮的内容，外面的数表示所存贮的数在数组中的序号。图 8.3.1 对应于一个数组

$$A = \{15, 12, 11, 7, 6, 5, 4, 2, 1, 0\}$$

例如$A(1)=15$，$A(7)=4$等，而且每个节点的"儿子"节点和"父亲"节点的地址可以通过简单计算而得到。假如一个节点的地址为n，则它的"父亲"节点的地址应为$\lfloor \frac{n}{2} \rfloor$，它的左"儿子"节点的地址为$2n$，右"儿子"节点的地址为$2n+1$。形象地用图 8.3.2 表示这种关系。$\lfloor \frac{n}{2} \rfloor$，$2n$，$2n+1$在计算机中用二进制数表示时运算是十分方便的。这样的数据结构，使地址能非常容易求得。

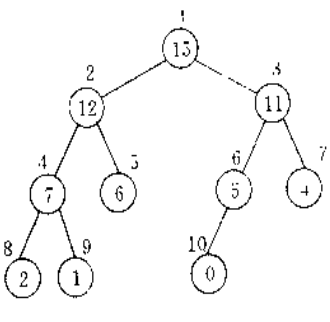


图 8.3.1

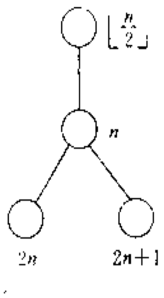


图 8.3.2

堆除了地址有如上的特点外，还要求每一个节点存贮的数比它的左右两儿子节点所存贮的数都要大。因此堆的根节点所存贮的数是堆中存贮的数中最大的。

堆集分类法首先要求是堆。由于最大元素一定在堆的顶(即堆的根节点)上。从堆中取走根节点后，为了保证堆状结构不变，需要进行调整。故堆集分类法的第一步要从二分树中构成堆。先通过一个例子作非形式化的说明(见图 8.3.3)，由虚线构成的矩形是当前着眼点的标志。调整的顺序是自右向左，自下向上；然后自上而下。为了方便起见"○"改为"•"。

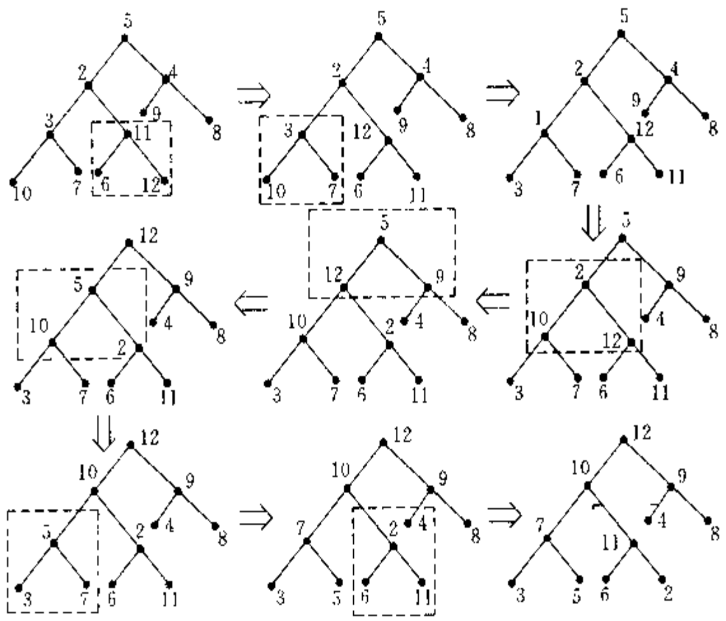


图 8.3.3

2. 建堆算法

设 $H(i)$ 表示堆集中第 i 个节点的地址，$L(i)$ 和 $R(i)$ 分别表示第 i 个节点的左"儿子"节点和右"儿子"节点的地址，$A(i)$ 表示第 i 个节点所存储的关键字。

对于二分树，内点数目比叶结点数目正好少一个。下面我们假定堆的节点个数为 n，即堆的规模为 n。则建堆算法可描述如下：

(1) $i \leftarrow \lfloor \frac{n}{2} \rfloor$。

(2) 若 $i \geqslant 1$，则转(3)。否则，结束。

(3) $l \leftarrow L(i)$，$r \leftarrow R(i)$。

(4) 若 $l \leqslant n$ 且 $A(l) > A(i)$，则作【 $\max \leftarrow l$，转(5)】。否则，作【 $\max \leftarrow i$，转(5)】。

(5) 若 $r \leqslant n$ 且 $A(r) > A(\max)$，则作【 $\max \leftarrow r$，转(6)】。否则转(6)。

(6) 若 $\max = i$，则转(7)。

否则作【 $A(i)$ 和 $A(\max)$ 互换，转(7)】。

(7) $i \leftarrow i - 1$，转(2)。

3. 建堆的时间复杂性

从堆的结构可知：

堆的顶部0层有 2^0 个结点；

1 层有 2^1 个结点；

2 层有 2^2 个结点；等等。

各新点都要往下"过滤"直到满足"堆"的要求为止。

设二分树高为 h，顶点数 $N=1+2+2^2+\cdots+2^h=2^{h+1}-1$。第 k 层一个数在最坏情况下要过滤到叶片必须作 $2(h-k)$ 次比较，即每下降一层必须作两次比较。但第 k 层有 2^k 个点，故构成堆所作的比较次数（指两个"儿子"中找最小的与其"父亲"再作的比较）为：

$$S=2\sum_{k=0}^{h}(h-k)2^k=2h\sum_{k=0}^{h}2^k-2\sum_{k=0}^{h}k2^k=2S_1-2S_2$$

$$S_1=h\sum_{k=0}^{h}2^k=h\frac{2^{h+1}-1}{2-1}=h(2^{h+1}-1)$$

$$S_2=\sum_{k=0}^{h}k2^k=\left[\sum_{k=0}^{h}kx^k\right]_{x=2}$$

而

$$\begin{aligned}\sum_{k=0}^{h}kx^k&=(x+2x^2+3x^3+\cdots+hx^h)\\&=x(1+2x+3x^2+\cdots+hx^{h-1})\\&=x(x+x^2+x^3+\cdots+x^h)'=x\left[\frac{x-x^{h+1}}{1-x}\right]'\\&=x\frac{(1-x)(1-(h+1)x^h)+x-x^{h+1}}{(1-x)^2}\\&=x\frac{1-(h+1)x^h+hx^{h+1}}{(1-x)^2}\end{aligned}$$

$$\begin{aligned}S_2&=2[1-(h+1)2^h+h2^{h+1}]=2+2h2^h-2^{h+1}\\&=2+(h-1)2^{h+1}\end{aligned}$$

∴

$$\begin{aligned}S&=2h(2^{h+1}-1)-2(2+(h-1)2^{h+1})\\&=2(2^{h+1}-h)-4\approx 4N\end{aligned}$$

即建堆的时间复杂度为 $O(N)$。

4. 堆集分类法

根据堆的结构特点，最大元素必在堆顶，即在堆二分树的根节点。将堆顶元素取走，用堆的最后一个位置（指地址序号）的元素置于堆顶，其结果必将不满足堆的要求。利用与建堆类似的步骤对之进行调整，使之恢复成为堆。如此反复，直到剩下两个元素为止。

举例说明：

从图 8.3.4 中可看出最后在堆的各节点依存储排好序的结果。

5. 堆集分类法的复杂性分析

具有 n 个节点的堆的高度 $h=\lfloor \log_2 n \rfloor$，在堆集分类法的过程中经常要将一元素置

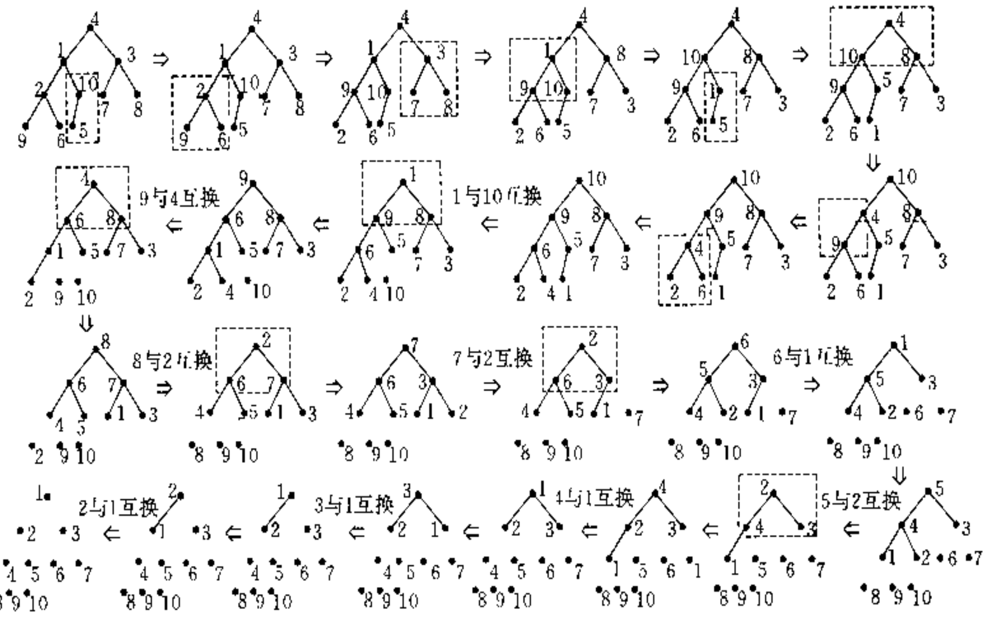


图 8.3.4

于堆的根节点，为此将失去作为堆的特点，必须调整使之恢复堆的性质。在这过程中在最坏情况下最多要作 $2\lfloor \log_2 n \rfloor$ 次比较，故堆集分类法的时间复杂性为：

$$\begin{aligned}2\sum_{k=1}^{n-1}\lfloor \log_2 k\rfloor &= 2\{(\lfloor\log_2 2\rfloor+\lfloor\log_2 3\rfloor)+(\lfloor\log_2 4\rfloor+\lfloor\log_2 5\rfloor\\ &\quad+\lfloor\log_2 6\rfloor+\lfloor\log_2 7\rfloor)++\cdots+(\cdots+\lfloor\log_2(n-1)\rfloor)\}\\ &= 2\{2\cdot 1+2\cdot 2^2+3\cdot 2^3+\cdots+(\lfloor\log_2 n\rfloor-1)2^{(\lfloor\log_2 n\rfloor-1)}\\ &\quad+\lfloor\log_2 n\rfloor(n-2^{\lfloor\log_2 n\rfloor})\}=2\{n\lfloor\log_2 n\rfloor+2^{\lfloor\log_2 n\rfloor+1}+2\}\end{aligned}$$

这是因为：

$$\begin{aligned}&1\cdot 2+2\cdot 2^2+3\cdot 2^3+\cdots+n2^n\\ &=\left\{x\frac{\mathrm{d}}{\mathrm{d}x}(1+x+x^2+\cdots+x^n)\right\}_{x=2}\\ &=\left\{x\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{1-x^{n+1}}{1-x}\right)\right\}_{x=2}\end{aligned}$$

所以堆集分类法的时间复杂性在最坏情况下为 $\Theta(n\log_2 n)$ 而空间复杂性为 n。

堆集分类是一种非常出色的算法，不仅如此，堆的本身也是一种有着广泛的应用的数据结构。比如由计算机参与任务安排，任务是按照它的优先级顺序进行处理的，任务经常随时加入，删除，故不需全部排序，只要将其优先数最高的关键字置于堆顶就可以。当任务完成或中断后，便将优先数最高的任务提出来，这样一种数据结构是“栈”、“队”数据结构的推广。当有新任务到达时，将其插入到堆中，堆的插入过程也很简单，留给读者自己

思考。

习　题

1. 高度为 h 的堆，其元素个数最多、最少各多少？

2. 试讨论由 n 个元素构成的堆的高度。

试证堆的子树中的最大元素也一定在该子树的根节点上。

3. 试通过下列例子详细陈述建堆的步骤，

5，2，4，3，18，10，11，16，9，8

即这些数依次自上而下，自左向右分布在一二分树的节点上。

4. 试证 n 个元素的堆，高为 h 的点至多为

$$\lceil n/2^{h+1} \rceil$$

5. 以序列 6，14，3，27，9，19，22，10，5 为例说明堆集分类法。

6. 若有一个堆自顶向下，自左向右的元素依次为 20，18，14，10，17，13，12，9，5，11，7，6，试讨论插入元素 8 的操作步骤。若删去元素 12，又将如何操作。

第9章　内存分类法之三：下溢分类法、快速分类法

9.1　下溢分类法

下溢分类又称Bubble分类，或称冒泡排序，是直接分类算法的一种，它对相邻两关键字进行比较。首先将第一个记录的关键字和第二个记录的关键字比较，若为逆序，则将两个记录交换，然后比较第二个记录和第三个记录的关键字，若必要就交换位置，反复进行，直至将最大的关键字换到最后一个，第二轮从头开始，从而将第二大的数送到它的合适位置，周而复始，直到排序完毕为止，显然判别分类结束的标志是“在一轮分类过程中没有进行关键字的交换动作”。其算法容易写出：

(1) $k \leftarrow n$。

(2) $j \leftarrow 1, lab \leftarrow 0$。

(3) 若 $x(j) > x(j+1)$，则转(4)。否则，转(5)。

(4) $r \leftarrow x(j), x(j) \leftarrow x(j+1), x(j+1) \leftarrow r, lab \leftarrow 1$。

(5) $j \leftarrow j+1$。

(6) 若 $j<k$，则转(3)，否则，转(7)。

(7) 若 $lab=1$，则作[$k \leftarrow k-1$，转(2)]。否则，结束。

算法中 lab 是一种标志数，n 个元素的序列一般最坏情况要进行 $n-1$ 轮，但只要出现最后 $lab=0$ 标志在这一轮中序列已不改变，说明已分类完毕，可以结束了。k 用来刻化排列的进程，开始时 $k=n$，依次 $k=n-1, n-2, \cdots, 2, 1$。

例　对初始序列49，38，65，97，76，13，27进行分类：

	(0)	(1)	(2)	(3)	(4)	(5)	(6)
	49	38	38	38	38	13	13
	38	49	49	49	13	27	27
	65	65	65	13	27	38	38
	97	76	13	27	49	49	49
	76	13	27	65	65	65	65
	13	27	76	76	76	76	76
	27	97	97	97	97	97	97
$lab=$		1	1	1	1	1	0
$k=$		7	6	5	4	3	2

表上()里的数标志着迭代次数。

复杂性分析：

1. 最坏情况下，n 个关键字 $x_1, x_2, \cdots, x_n$ 的分类必须反复进行直到 $k=1$ 为止。

设 T_n=n 个关键字的分类在最坏情况下所需的比较次数，则有关系式：

$$T_n = T_{n-1} + n - 1, T_2 = 1$$

令
$$G(x) = T_2 + T_3x + T_4x^2 + \cdots$$

$$(1-x)G(x) = \frac{1}{(1-x)^2}$$

$$G(x) = \frac{1}{(1-x)^3} = (1+2x+3x^2+\cdots)(1+x+x^2+\cdots)$$

∴
$$T_n = 1+2+3+\cdots+(n-1)$$

$$= \frac{1}{2}n(n-1)$$

2. 最好情况，就是初始序列已是正序：

$$T_n = n-1$$

3. 平均复杂度分析：

不相等的 n 个数 $x_1, x_2, \cdots, x_n$ 按其大小顺序分类得：

$$x_{i_1} < x_{i_2} < \cdots < x_{i_n}$$

令 $x_{i_k} \leftrightarrow k, k=1,2,\cdots,n$

即对分类所得的序列：

$$\begin{array}{ccccccccc} x_{i_1} & < & x_{i_2} & < & x_{i_3} & < & \cdots & < & x_{i_n} \\ \updownarrow & & \updownarrow & & \updownarrow & & & & \updownarrow \\ 1 & & 2 & & 3 & & \cdots & & n \end{array}$$

序号：1 2 3 … n

从而可知序列 $x_1, x_2, \cdots, x_n$ 对应 $1,2,\cdots,n$ 的一个排列：

$$\sigma_1\sigma_2\cdots\sigma_n$$

其中 σ_k 是 x_k 的序号。例如序列 17，23，15，7，3 依从小到大的顺序排列得：

$$\begin{array}{ccccccccc} 3 & < & 7 & < & 15 & < & 17 & < & 23 \\ \updownarrow & & \updownarrow & & \updownarrow & & \updownarrow & & \updownarrow \\ 1 & & 2 & & 3 & & 4 & & 5 \end{array}$$

序号：1 2 3 4 5

故原序列 17，23，15，7，3 对应一个排列 45321，即 $\sigma_1=4, \sigma_2=5, \sigma_3=3, \sigma_4=2, \sigma_5=1$。即 17 的序号为 4，23 的序号为 5，等等。以后 $\sigma_1\sigma_2\cdots\sigma_n$ 都指 $1,2,\cdots,n$ 的某一排列。某种意义上可以说对序列 $x_1, x_2, \cdots, x_n$ 的排序相当于对 $\sigma_1\sigma_2\cdots\sigma_n$ 的排序。在排列 $\sigma=\sigma_1\sigma_2\cdots\sigma_n$ 中，σ_k 的左边比 σ_k 大的数的个数用 δ_k 表示它，这样一个排列 σ 对应了一个 $\delta=\delta_1\delta_2\cdots\delta_n$。显然有：

$$0 \leqslant \delta_k < k, k=1,2,\cdots,n$$

反之，可以证明若 δ_k 满足关系 $0 \leqslant \delta_k < k, k=1,2,\cdots,n$，则 $\delta=\delta_1\delta_2\cdots\delta_n$ 对应了一个 $\sigma=\sigma_1\sigma_2\cdots\sigma_n$。例如 $\sigma=45321$，对应有 $\delta=00234$，比如 $\delta_3=2$，因 $\sigma_3=3, \sigma_2=5, \sigma_1=4$，即 σ_3 前面有两个比 σ_3 大的数，余此类推。

读者稍加注意不难发现：下溢法的每一个过程开始对应一个排列 σ，也对应地有一个 δ，而且每一轮结果使 δ 的非零元素减 1，而且向左移一位。比如：

$\sigma=582341967 \Leftrightarrow \delta=002225022$

$\sigma=523418679 \Leftrightarrow \delta=011140110$

$\sigma=234156789 \Leftrightarrow \delta=000300000$

$\sigma=231456789 \leftrightarrow \delta=002000000$

$\sigma=213456789 \leftrightarrow \sigma=010000000$

$\sigma=123456789 \leftrightarrow \sigma=000000000$

因而下溢法需要的过程次数取决于 δ 中最大元素的数值，即过程数目等于 δ 中最大元素的值加 1。

设 P_k 为 $\max\{\delta_i\}=k-1$ 的 δ 出现的概率，即

$$P_k=P_r\{\max_i\{\delta_i\}=k-1\}$$

但出现 δ 使 $\max_i\{\delta_i\}<k$ 的概率等于使所有元素 $\delta_i<k$ 的 δ 的数目除以 $n!$。

若 $\delta=\delta_1\delta_2\cdots\delta_n$ 满足 $\max_i\{\delta_i\}<k$，则必有 $\delta_1=0, 0\leqslant\delta_2<2, 0\leqslant\delta_3\leqslant 3, \cdots, 0\leqslant\delta_k<k$，$0\leqslant\delta_{k+1}<k, \cdots, 0\leqslant\delta_n<k$，故使 $\delta=\delta_1\delta_2\cdots\delta_n$ 满足条件 $\max_i\{\delta_i\}<k$ 的个数为 $k^{n-k}k!$，即：

$$P_r\{\max_i\{\delta_i\}<k\}=\frac{k!\,k^{n-k}}{n!},$$

$$\begin{aligned}
\therefore\quad P_k &= P_r\{\max_i\{\delta_i\}=k-1\}\\
&= P_r\{\max_i\{\delta_i\}<k\}-P_r\{\max_i\{\delta_i\}<k-1\}\\
&= \frac{1}{n!}\{k^{n-k}k!-(k-1)^{n-k+1}(k-1)!\}\\
m &= \sum_{k=1}^{n}kP_k\\
&= \sum_{k=1}^{n}\{k^{n-k}k!-(k-1)^{n-k+1}(k-1)!\}k/n!\\
&= \left\{\sum_{k=1}^{n-1}k\cdot k^{n-k}k!-\sum_{k=1}^{n-1}(k+1)!\,k^{n-k}\right\}\Big/n!\\
&= n-\frac{1}{n!}\sum_{k=1}^{n-1}k^{n-k}\cdot k! = n+1-\sum_{k=1}^{n}\frac{k^{n-k}\cdot k!}{n!}
\end{aligned}$$

而
$$\sum_{k=1}^{n}\frac{k^{n-k}k!}{n!}=\sqrt{\frac{n\pi}{2}}-\frac{2}{3}+O(n^{-\frac{1}{2}})$$

这个公式的证明略去。

前面讨论的算法是将最大的元素向后移，类似的理由，也可以将较小的元素向前移，通常称之为"下沉"和"上浮"，实际中我们可将"下沉"算法和"上浮"算法交替进行，从而可取得较好的效果。

(0)	(1)	(2)	(3)	(4)
49	38	13	13	13
38	49	38	38	27
65	65	49	49	38
97	76	65	65	49
76	13	76	27	65
13	27	27	76	76
27	97	97	97	97

还是以本节开始时的例子为例，第一轮最大的数 97 下沉到底；第二轮最小的数 13 上浮到顶，全部排序完成只用四轮而不是六轮。

9.2 快速分类法

1. 基本思想

不失一般性，假定把对 n 个对象的分类看作是对 1 到 n 的 n 个整数的排序。快速分类法的基本思想是从中取一合适的关键字 k，以 k 为标准把需要分类的 n 个对象分成两部分，一是比 k 小，一是比 k 大，即

$$\underline{(小于\ k\ 的部分)}k\underline{(大于\ k\ 的部分)}$$

然后对这两部分分别进行快速分类，对这两部分分类完毕后，简单地合并联接起来即可。算法还可递归进行。

快速分类法采用的算法设计技术还是分治策略。

我们先通过例子非形式化地介绍算法的思想实质：

3,9,1,6,5,4,8,2,10,7

(1) 引进指针 i 和 j 分别指向序列的开始和终端，并利用最左端的数作为 k。

```
3  9  1  6  5  4  8  2  10  7
↑                           ↑
i                           j
```

(2) 指针 j 向左移动，直到碰到比 3 小的数为止。本例为 2，2 与 3 互换得

```
2  9  1  6  5  4  8  3  10  7
↑                    ↑
i                    j
```

(3) 指针 i 向右移动，直到遇到比 3 大的数为止。本例为 9，9 与 3 互换得

```
2  3  1  6  5  4  8  9  10  7
   ↑                 ↑
   i                 j
```

这样指针 i 的左端的数比 3 小，指针 j 和它右端的数都比 3 大。

(4) 对指针 i 和 j 间的序列继续以上(1)—(3)的步骤直到指针 i 和指针 j 重合为止，算是一轮结束。把序列分成比 3 小的和比 3 大的两部分，现将第一轮的过程表示如下：

```
3  9  1  6  5  4  8  2  10  7
↑           ⇓               ↑  置指针 i 和 j
i                           j
3  9  1  6  5  4  8  2  10  7  指针 j 左移到 2(≤3)
↑           ⇓        ↑
i                    j
2  9  1  6  5  4  8  3  10  7  指针 i 右移到 9(>3)
   ↑                 ↑
   i        ⇓        j
```

2 3 1 6 5 4 8 9 10 7 3和9互换,
　↑ ↑
　i j ⇓ j左移至1(<3)

2 1 3 6 5 4 8 9 10 7 1和3互换
　　↑↑
　　i j

至此已将序列两部分,一是(2,1),经排序得(1,2)。另一部分利用快速分类法进行排序如下:

6 5 4 8 9 10 7
↑ ↑ 置指针i和j
i ⇓ j

6 5 4 8 9 10 7 指针j左移至4(<6)
↑ ↑
i j ⇓

4 5 6 8 9 10 7 4与6互换
↑ ↑
i j ⇓

4 5 6 8 9 10 7 指针i右移至6。
　↑↑
　i j

对8,9,10,7序列继续利用快速分类法如下:

8 9 10 7
↑ ↑ 置指针i和j
i ⇓ j

7 9 10 8 指针j由于7<8,故不能左移
↑ ↑ 7与8互换
i ⇓ j

7 9 10 8 指针i右移至9(>8)
　↑ ↑
　i ⇓ j

7 8 10 9 8与9互换
　↑ ↑
　i ⇓ j

7 8 10 9 指针j左移至8
　↑↑
　i j

故合并得经过分类的序列:

1 2 3 4 5 6 7 8 9 10

2. 快速分类算法

快速分类法的关键是分组,即分成小于k和大于k两组子序列。假定已知$X=\{x_1,x_2,\cdots,x_n\}$,$l\geqslant1$,$r\leqslant n$,不失一般性,假定要求对子序列

$$x_l,x_{l+1},\cdots,x_{r-1},x_r$$

进行分组,令分组算法为过程

Partition(x,l,r):

(1) $b \leftarrow x(l), i \leftarrow l, j \leftarrow r$。

(2) 若 $x(j) \leqslant b$,则转(3)。

否则【 $j \leftarrow j-1$,转(2)】。

(3) 若 $x(i) \geqslant b$,则转(4)。

否则作【 $i \leftarrow i+1$,转(3)】。

(4) 若 $i=j$,则转(5)。否则作【 $t \leftarrow x(i), a(i) \leftarrow a(j), a(j) \leftarrow t$,转(2)】。

(5) 返回 j 结束。

对 $X=\{x_1, x_2, \cdots, x_n\}$ 的快速分类只不过是递归调用过程 Partion,即

Quicksort(x,l,r):

(1) 若 $l=r$,则转(2)。

否则作【 $p \leftarrow$ Partition(x,l,r),

Quicksort$(x,l,p-1)$; Quicksort $(x,p+1,r)$】。

(2) 结束。

3. 复杂性分析:

前面例子中的快速分类过程可形象地如图 9.2.1 所示。

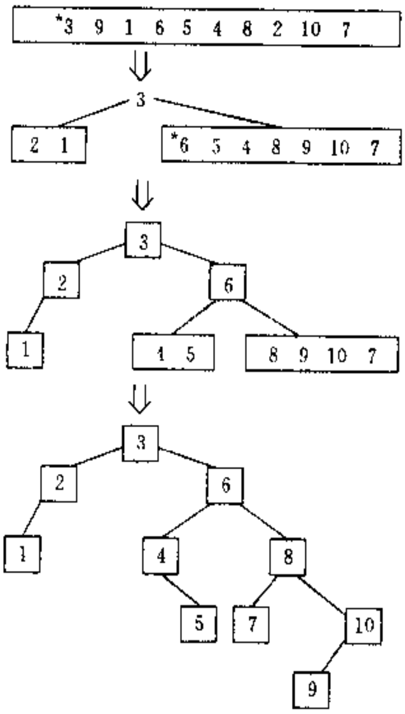


图 9.2.1

快速分类的全过程在于最后获得一棵树,比如以叶片“7”为例,它首先与 3 作比较,接着与 6 比较,然后与 8 比较,最后才确定其位置,共作了三次比较。一般说来所需的比较总数为对应的树从各点到树根的距离的总和。根据这个道理可得:

(1) 最坏情况

每个内点只有一个“儿子”,如图 9.2.2 所示。这时各点到树根的距离之和为:

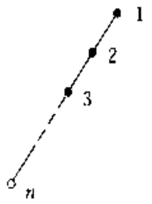


图 9.2.2

$$1+2+3+\cdots+(n-2)+(n-1)=\frac{n}{2}(n-1)$$

即
$$T_n=\frac{n}{2}(n-1)$$

(2) 最好的情况

每个内点都有两个儿子,各点到树根的距离总和最小。这时快速分类所对应的树是一棵二分树,对于这二分树有:

2(叶结点到根的距离之和)=各顶点到树根距离之和+顶点数−1

这个公式不难用数学归纳法加以证明,证明留作练习。若叶子数目为 n,则内点数目为 $n-1$。故顶点数目 $N=2n-1$。叶子到树根的距离之和的最小值约为 $n\log_2 n$。综合以上几点可得:

$$\min_{l}\sum(\text{各顶点到树根的距离})\approx 2n\log_2 n-(N-1)$$
$$\approx N\log_2 N$$

(3) 平均复杂度分析

记 T_n=利用快速分类法对 n 个对象进行分类所作的比较的平均数。

由于快速分类法取第 1 个数作为分割为两个序列的标准,n 个数的机率均等。若取第 k 个数,则一个序列有 $k-1$ 个数,另一个有 $n-k$ 个数,故有递归关系:

$$T_n=\frac{1}{n}\sum_{k=1}^{n}(n-1+T_{k-1}+T_{n-k})$$
$$=n-1+\frac{2}{n}\sum_{k=0}^{n-1}T_k,\qquad T_0=0 \qquad (9.2.1)$$

其中 $n-1$ 表示第一轮把 n 个数的序列一分为二时所作的比较次数。

式(9.2.1)是非常系数的递推关系:

$$nT_n=n(n-1)+2\sum_{k=0}^{n-1}T_k\quad T_0=0$$
$$(n+1)T_{n+1}=n(n+1)+2\sum_{k=0}^{n}T_k$$

∴
$$(n+1)T_{n+1}-nT_n=2n+2T_n$$

或
$$(n+1)T_{n+1}=(n+2)T_n+2n$$

令
$$S_n=T_n/n+1$$

则有

$$S_{n+1}-S_n=\frac{2n}{(n+1)(n+2)},\ S_0=0$$
$$S_n=\sum_{k=1}^{n-1}\frac{2k}{(k+1)(k+2)}=4\sum_{k=1}^{n-1}\frac{1}{k+2}-2\sum_{k=1}^{n-1}\frac{1}{k+1}$$

即

$$S_n = 4\sum_{k=2}^{n}\frac{1}{k+1} - 2\sum_{k=1}^{n-1}\frac{1}{k+1} = 2\sum_{k=2}^{n-1}\frac{1}{k+1} + \frac{4}{n+1} - 1$$

但

$$\sum_{k=2}^{n-1}\frac{1}{k+1} = \sum_{k=3}^{n}\frac{1}{k} < \int_2^n \frac{1}{x}\mathrm{d}x = \ln n - \ln 2$$

∴

$$S_n < 2\ln n - \left(2\ln 2 - \frac{4}{n+1} + 1\right) = 2\ln n + O(1)$$

$$T_n < 2(n+1)\ln n + O(n)$$

从上可知快速分类法的最坏情况的算法复杂度为 $O(n^2)$，但平均复杂性是 $O(n\ln n)$。为避免最坏情况的发生，在调用 Quicksort(x,l,r)时，可考虑引进随机的因素。即对子序列 $x_l,x_{l+1},\cdots,x_r$ 进行分组时随机产生一个整数 m：

$$l \leqslant m \leqslant r$$

作交换

$$x(l) \leftrightarrow x(m)$$

然后采用快速分类法，以避免出现最坏的情况。也就是将第一个元素 k 改为随机产生序列中某一元素。

习　题

1. 对于下溢分类法：

(1) 证明，经过一轮的交换之后，最大的元素必在最底部。

(2) 证明，如果没有连续的两个元素逆序排列，则整个表就是按序排列的。

2. 通过修改下溢分类法，可以避免在表的尾部常进行的无必要的比较，方法是跟踪循环中每一轮最后的数的位置。

(1) 证明如果最后的交换发生在第 j 和第 $j+1$ 个元素之间，则从$(j+1)$到 n 的元素都已在其正确的位置上。(注意，这比说明这们是有序的结论要强)。

(2) 修改算法使得若一轮中最后交换了第 j 和 $j+1$ 元素，则下一轮将不再检查 $j+1$ 以后位置的元素。

(3) 这种修改可以改善最坏情况下的效率吗？如果能，请说明之。

3. 对于快速分类来说，如果表已经是排序的，则要进行多少次比较，又有多少次对换操作？

4. 如果在快速分类中不用最左端的元素，而是用最左端值，中间值与最右端值三数居中的值来进行分治，试讨论在最坏情况下快速分类要进行多少次比较。(包括求中间值用的比较次数)。

5. 下面是一个快速分类中将元素分类过程的变型版本：

```
Procedure Split(first,last:Index;Var Splitpoint:Index);
{此过程需在数组 L 最后附加一个大于数组中任何数的最大值。}
Var
    x : Key;
```

```
    i,j : Index;
begin
    x : =L [first];
    i : =first; j : =last+1;
    repeat
        repeat j : =j-1 until L[j]≤x;
        repeat  i : =i+1  until  L[i]≥x;
        Interchange (L[i],L[j])
    until  i≥j;
    Interchange (L[i],L[j]);
    Interchange (L[first],L[j]);
    Splitpoint : =j
end  {Split}
```

(1) 若 L 有 k 个元素，则最坏情况下需多少次比较？(不是 $k-1$)。这对于快速分类的最坏情况有什么影响？

(2) 在什么情况下(初始情况)L 后附加的最大值会被检测到？

(3) 由于 Split 还要在较少的 L 的片断中运行，为什么不必在每个片断最后都加上最大值？

(4) 若 L 中有很多相同值，它们在分类后是否都排在 Splitpoint 的同一侧？

6. 证明快速分类在所有元素均相同的情况下时间复杂度是 $\Theta(n/g^n)$。

7. 证明当数组中的数以非增序排列时，快速分类的时间复杂性是 $\Theta(n^2)$。

第 10 章　内存分类法之四:归并分类法和基数分类法

10.1　归并分类法

和快速分类法一样,归并分类采用的技术也是分治策略。使得在最坏情况下其复杂度为 $O(n\log_2 n)$。归并分类法的基本思想是把要分类的序列:$x_1, x_2, \cdots, x_n$　一分为二。

$$x_1, x_2, \cdots, x_{\lfloor n/2 \rfloor}$$

$$x_{\lfloor n/2 \rfloor+1}, x_{\lfloor n/2 \rfloor+2}, \cdots, x_n$$

对它们分别加以分类,然后加以归并为统一的经过排序的序列。算法可递归进行,故归并分类法的主要工作在于归并。

例如对下面 8 个数进行分类(见图 10.1.1)

30,29,18,65,36,90,85,76

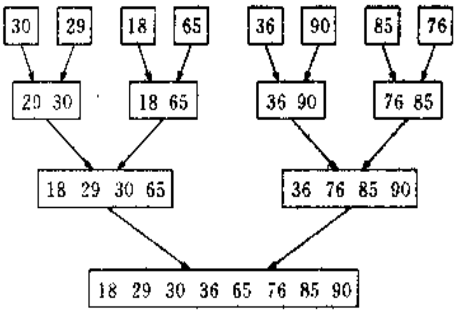


图　10.1.1

假定已知两序列 $a_1, a_2, \cdots, a_m$ 和 $b_1, b_2, \cdots, b_n$,已分类完毕:

$$a_1 < a_2 < a_3 < \cdots < a_m$$

$$b_1 < b_2 < b_3 < \cdots < b_n$$

其归并算法可描述如下:

(1) $h \leftarrow 1, i \leftarrow 1, j \leftarrow 1$。

(2) 若 $i \leqslant m$ 且 $j \leqslant n$,则转(3)。否则,转(6)。

(3) 若 $a_i < b_j$,则转 (4)。

否则作【$c_k \leftarrow b_j, j \leftarrow j+1$,转(5)】。

(4) $c_k \leftarrow a_i, i \leftarrow i+1$。

(5) $k \leftarrow k+1$,转 (2)。

(6) 若 $i > m$,则转 (9)。否则,转(7)。

(7) $c_k \leftarrow b_j, j \leftarrow j+1, k \leftarrow k+1$。

(8) 若 $j \leqslant n$,则转 (7)。否则转 (11)。

(9) $c_k \leftarrow a_i, i \leftarrow i+1, k \leftarrow k+1$。

(10) 若 $i \leqslant m$,则转 (9)。否则转 (11)。

(11) 停止。

最后所得 $c_1, c_2 \cdots, c_{m+n}$ 就是归并所得的结果。

例如已知 $a=\{18,29,30,65\}$,$b=\{36,76,85,90\}$ 是两组已分类好的序列,将其归并的过程可描述如下:

a:18	29	30	65	b:36	76	85	90	c:18								
18	29	30	65	36	76	85	90	18	29							
18	29	30	65	36	76	85	90	18	29	30						
18	29	30	65	36	76	85	90	18	29	30	36					
18	29	30	65	36	76	85	90	18	29	30	36	65				
18	29	30	65	36	76	85	90	18	29	30	36	65	76	85	90	

复杂性分析:

为讨论方便起见,考虑 $N=2^n$ 的情形。设 T_n 表示 2^n 个对象的分类在最好情况下所需的比较次数。则有递归关系:

$$T_n = 2T_{n-1} + 2^{n-1}, T_1 = 1$$

其中 2^{n-1} 指的是最后归并所作的比较次数。

设:

$$G(x) = T_1 + T_2 x + T_3 x^2 + \cdots$$

$$x: T_2 = 2T_1 + 2$$

$$X^2: T_3 = 2T_2 + 2^2$$

$$+) \cdots\cdots\cdots\cdots$$

$$G(x) - 1 = 2xG(x) + \frac{2x}{1-2x}$$

∴

$$G(x) = \frac{1}{(1-2x)^2} = (1 + 2x + 2^2x^2 + \cdots)^2$$

$$T_n = n2^{n-1} = \frac{1}{2} n 2^n$$

即

$$T_n = \frac{1}{2} N \log_2 N$$

即其时间复杂度为 $O(N\log_2 N)$

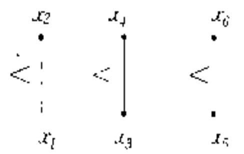
10.2 Ford-Johnson 归并插入分类法

1. 算法的非形式化描述

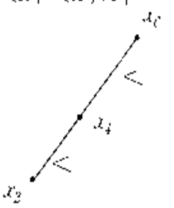
Ford-Johnson 算法的基本思想是将 n 个元素分成 $\left\lfloor \frac{n}{2} \right\rfloor$ 对，在 $\left\lfloor \frac{n}{2} \right\rfloor$ 对之间进行比较，取其“优胜者”进行分类排序，而在对这 $\left\lfloor \frac{n}{2} \right\rfloor$ 个“优胜者”分类时可递归调用该算法。然后将“失败者”分别插入“优胜者”分类完毕的序列中。插入可采用二分插入。从下面我们将看出 Ford-Johnson 分类法的关键在于它的插入顺序，不同的插入顺序将有不同的效率。例如：对序列：

$$x_1, x_2, x_3, x_4, x_5, x_6$$

进行分类。若 $x_1<x_2, x_3<x_4, x_5<x_6$。



又设对“优胜者”x_2, x_4, x_6，分类得：$x_2<x_4<x_6$，即



则 x_6 是当然的“冠军”，x_1 的插入无须作任何比较，接着插入 x_3 或 x_5 都只要作 2 次比较，但以先插入 x_5 较好。比如 x_3 先插入到序列 $x_1<x_2<x_4$ 中去，x_3 对 x_2 先作比较，若 $x_3>x_2$，则得 $x_4>x_3>x_2$，否则 x_3 对 x_1 作比较。总之两次比较是足够的。但在 x_3 插入后考虑 x_5 的插入，有时则需要作 3 次比较。比如 x_5 插入到 $x_1<x_2<x_3<x_4$ 中去便是一例。

若 x_5 先插入，只要作两次比较，而且无论 x_5 插入到什么地方，x_3 的插入最多只要 2 次比较就可以了。因 x_5 先插入不外以下几种情形：

(1) $x_1<x_2<x_4<x_5$；

(2) $x_1<x_2<x_5<x_4$；

(3) $x_1<x_5<x_2<x_4$；

(4) $x_5<x_1<x_2<x_4$.

因 $x_3<x_4$，故 x_3 插入不论什么状况都只要作两次比较，以(1)为例，x_3 先与 x_2 进行比较，若 $x_3>x_2$，则 x_3 插入到 x_2 与 x_4 之间；若 $x_3<x_2$，则再与 x_1 比较。便可决定其插入位置。又比如(3)，x_3 先与 x_5 比较，若 $x_3<x_5$ 则再与 x_1 比较，否则再与 x_2 比较便可确定 x_3 的插入位置。故一共作了 7 次比较，x_1 插入无需比较，x_3, x_5 插入时作 2 次比较。

将上面的讨论用于 10 个数的分类，如：

$$\begin{array}{ccccccccc} w_1 & < & w_2 & < & w_3 & < & w_4 & < & w_5 \\ \vert < & & \vert < & & \vert < & & \vert < & & \vert < \\ l_1 & & l_2 & & l_3 & & l_4 & & l_5 \end{array}$$

先利用上述的方法将 l_3,l_2 先后通过两次比较插入得：

$$\begin{array}{ccccccccccccccc} a_1 & < & a_2 & < & a_3 & < & a_4 & < & a_5 & < & a_6 & < & w_4 & < & w_5 \\ & & & & & & & & & & & & \vert < & & \vert < \\ & & & & & & & & & & & & l_4 & & l_5 \end{array}$$

l_5 首先插入，面对着已排序的 $a_1<a_2<\cdots<a_6<\omega_4$ 只要通过 3 次比较，例如 l_5 和 a_4 进行比较，若 $l_5>a_4$，则 l_5 和 a_6 作比较；否则 l_5 和 a_2 进行比较，若 $l_5>a_4$ 但 $l_5<a_6$，则 l_5 和 a_5 进行一次比较以确定其位置，其他情况可同样进行。先插入 l_4 后，由于 $l_4<w_4$，故不论什么情况，l_4 的插入也只要作三次比较。若 l_4 先插入，然后考虑 l_5 的插入，可能要四次比较。

将上面归并插入分类的基本思想推广到一般的情形。设有下列关系的 s 对关键字：

$$\begin{array}{ccccccc} w_1 & < & w_2 & < & \cdots & < & w_s \\ \vert < & & \vert < & & \cdots & & \vert < \\ l_1 & & l_2 & & \cdots & & l_s \end{array}$$

令 b_k 表示不超过 k 次比较完成插入的失败者 l_j 的数目，则 l_{b_k} 为“失败”者中插入时需要作不超过 k 次比较中序号最高的一个。由 2 分插入法知对于 l_{b_k} 来说可能有 2^k 个可能的位置。这 2^k 个元素包含了 b_{k-1} 个 l_j 全部插入完毕。

$$\therefore \qquad b_k+b_{k-1}=2^k, \qquad b_0=1$$

令
$$G(x)=b_0+b_1x+b_2x^2+\cdots$$

$$\begin{array}{r} x:b_1+b_0=2 \\ x^2:b_2+b_1=2^2 \\ x^3:b_3+b_2=2^2 \\ +)\ \cdots\ \cdots\cdots \\ \hline G(x)-1+xG(x)=\dfrac{2x}{1-2x} \end{array}$$

$$(1+x)G(x)=\frac{1}{1-2x}$$

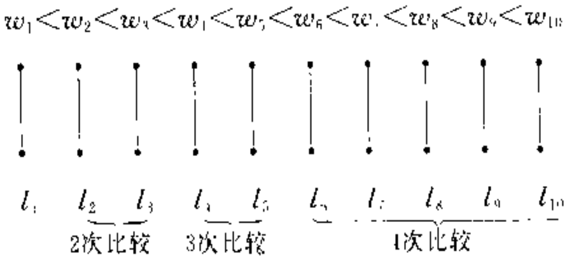
$$G(x)=\frac{1}{(1+x)(1-2x)}=\frac{1}{3}\left(\frac{2}{1-2x}+\frac{1}{1+x}\right)$$

$$=\frac{1}{3}\sum_{k=0}^{\infty}\left(2^{k+1}+(-1)^k\right)x^k$$

∴ $$b_k=\frac{1}{3}[2^{k+1}+(-1)^k]$$

$$b_0=1, b_1=1, b_2=3, b_3=5, b_4=11$$

例如 22 个关键字分类排序，1 个优胜者经过分类得：



l_1 插入无需比较；

l_2, l_3 需作 2 次比较；

l_4, l_5 需作 3 次比较；

$l_6, l_7, l_8, l_9, l_{10}, l_{11}$ 需作 4 次比较。

比较次数	0	2		3		4					
l_i	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	l_{11}
插入顺序	1	3	2	5	4	11	10	9	8	7	6

2. 算法分析

设 n 个关键字用 Ford-Johnson 法分类所作的比较次数用 c_n 表之，d_n 为 n 个"失败者"插入到 n 个"胜利者"中所作的比较次数 。则有：

$$c_n=C_{\lfloor \frac{n}{2} \rfloor}+d_{\lceil \frac{n}{2} \rceil, \lfloor \frac{n}{2} \rfloor}$$

$$c_1=0$$

在对 c_n 进行估计前先对 d_n 进行估计。设

$$b_{k-1}<l\leqslant b_k$$

由于 $$b_k=\frac{1}{3}[2^{k+1}+(-1)^k]$$

忽略 $(-1)^k$ 项得：

$$\frac{1}{3}2^k<l\leqslant \frac{1}{3}2^{k+1}$$

$$2^{k-1}<\frac{3}{2}l\leqslant 2^k$$

取对数得：

$$k-1<\log_2(\frac{3}{2}l)\leqslant k$$

$$k=\left\lceil \log_2 \frac{3}{2}l \right\rceil$$

可以证明正确的表达式应为

$$k=\left\lceil\log_2\left(\frac{3}{4}(2l-1)\right)\right\rceil$$

引理：(Hadian)　$c_n-c_{n-1}=\left\lceil\log_2\left(\frac{3}{4}n\right)\right\rceil$

证明：用数学归纳法证明。$n=2$ 时，$c_2=1$，$c_1=0$，$c_2-c_1=1=\left\lceil\log_2\left(\frac{3}{4}\cdot 2\right)\right\rceil$，引理成立。

设对 $k<n$ 时($n\geqslant 3$)引理成立，证 $k=n$ 时引理成立。

(1) $n=2s$ 时，这时$\left\lceil\frac{n}{2}\right\rceil=\left\lfloor\frac{n}{2}\right\rfloor=s$.

$$c_n=c_s+d_s+s$$

由于 $n=2s$，有$\left\lfloor\frac{n-1}{2}\right\rfloor=\left\lfloor\frac{n}{2}\right\rfloor-1=s-1$，

$$c_{n-1}=c_{s-1}+d_s+s-1$$

∴
$$c_n-c_{n-1}=1+c_s-c_{s-1}$$

根据假定：$c_s-c_{s-1}=\log_2\left(\frac{3}{4}s\right)$成立

$$c_n-c_{n-1}=1+\log_2\frac{3}{4}S$$

由于$\left\lceil\log_2\frac{3}{4}s\right\rceil=\left\lceil\log_2\left(\frac{3}{4}\cdot\frac{n}{2}\right)\right\rceil=\left\lceil\log_2\frac{3}{4}n\right\rceil-1$

∴
$$c_n-c_{n-1}=\left\lceil\log_2\frac{3}{4}n\right\rceil.$$

(2) $n=2s+1$ 时，由于

$$\left\lfloor\frac{n}{2}\right\rfloor=\left\lceil\frac{n}{2}\right\rceil-1=s,\quad\left\lceil\frac{n-1}{2}\right\rceil=\left\lfloor\frac{n}{2}\right\rfloor=s$$

∴
$$c_n=c_s+d_{s+1}+s,\quad c_{n-1}=c_s+d_s+s$$

$$c_n-c_{n-1}=d_{s+1}-d_s$$

但 d_s 是前 s 个“失败者”插入时所需要比较次数，故 $d_{s+1}-d_s$ 即为 t_{s+1} 插入时的比较次数：

$$d_{s+1}-d_s=\left\lceil\log_2\frac{3}{4}(2(s+1)-1)\right\rceil$$

$$=\left\lceil\log_2\frac{3}{4}(2s+1)\right\rceil$$

∴
$$c_n-c_{n-1}=\left\lceil\log_2\frac{3}{4}n\right\rceil$$

证毕。

由 Hadian 引理可得在最坏情况下 Ford-Johnson 算法需要：

$$\sum_{k=1}^{n}\left\lceil\log_2\frac{3}{4}k\right\rceil$$

次比较，

10.3 基数分类法

基数分类法与前面提到的分类法不同，以前的分类法都是以关键字间的比较进行分类。基数分类法则是依据关键字的表示形式进行。它的原理和卡片分类机类似。不妨设关键字是用二进制数构成，先按第一位数字分类，然后依顺序收集起来，再按第二位数字

A	0	0	0	0	1
S	1	0	0	1	1
O	0	1	1	1	1
R	1	0	0	1	0
T	1	0	1	0	0
I	0	1	0	0	1
N	0	1	1	1	0
G	0	0	1	1	1
E	0	0	1	0	1
X	1	1	0	0	0
A	0	0	0	0	1
M	0	1	1	0	1
P	1	0	0	0	0
L	0	1	1	0	0
E	0	0	1	0	1

(0)

R	1	0	0	1	0
T	1	0	1	0	0
N	0	1	1	1	0
X	1	1	0	0	0
P	1	0	0	0	0
L	0	1	1	0	0
A	0	0	0	0	1
S	1	0	0	1	1
O	0	1	1	1	1
I	0	1	0	0	1
G	0	0	1	1	1
E	0	0	1	0	1
A	0	0	0	0	1
M	0	1	1	0	1
E	0	0	1	0	1

(1) ↑

T	1	0	1	0	0
X	1	1	0	0	0
P	1	0	0	0	0
L	0	1	1	0	0
A	0	0	0	0	1
I	0	1	0	0	1
E	0	0	1	0	1
A	0	0	0	0	1
M	0	1	1	0	1
E	0	0	1	0	1
R	1	0	0	1	0
N	0	1	1	1	0
S	1	0	0	1	1
O	0	1	1	1	1
G	0	0	1	1	1

(2) ↑

X	1	1	0	0	0
P	1	0	0	0	0
A	0	0	0	0	1
I	0	1	0	0	1
A	0	0	0	0	1
R	1	0	0	1	0
S	1	0	0	1	1
T	1	0	1	0	0
L	0	1	1	0	0
E	0	0	1	0	1
M	0	1	1	0	1
E	0	0	1	0	1
N	0	1	1	1	0
O	0	1	1	1	1
G	0	0	1	1	1

(3) ↑

P	1	0	0	0	0
A	0	0	0	0	1
A	0	0	0	0	1
R	1	0	0	1	0
S	1	0	0	1	1
T	1	0	1	0	0
E	0	0	1	0	1
E	0	0	1	0	1
G	0	0	1	1	1
X	1	1	0	0	0
I	0	1	0	0	1
L	0	1	1	0	0
M	0	1	1	0	1
N	0	1	1	1	0
O	0	1	1	1	1

↑ (4)

A	0	0	0	0	1
A	0	0	0	0	1
E	0	0	1	0	1
E	0	0	1	0	1
G	0	0	1	1	1
I	0	1	0	0	1
L	0	1	1	0	0
M	0	1	1	0	1
N	0	1	1	1	0
O	0	1	1	1	1
P	1	0	0	0	0
R	1	0	0	1	0
S	1	0	0	1	1
T	1	0	1	0	0
X	1	1	0	0	0

↑ (5)

图 10.3.1

分类，依此类推。令

$$x_i = x_k^{(i)} x_{k-1}^{(i)} \cdots x_2^{(i)} x_1^{(i)}$$
$$x_j = x_k^{(j)} x_{k-1}^{(j)} \cdots x_2^{(j)} x_1^{(j)}$$

若存在 $t<k$，若存在 l 使得 $k \geqslant l > t$ 时 $x_l^{(j)} = x_l^{(i)}$，但 $x_t^{(j)} < x_t^{(i)}$，则 $x_j < x_i$。例如用 1 到 26 的 5 位二进制数表示 A 到 Z 的 26 个字母，例如 A 为 26 个字母之首，故为 00001，E 为 00101。对 $ASORTINGEXAMPLE$ 进行排序其过程如图 10.3.1，(1)是各个字母的序数用 5 位 0，1 符号串，按最后一位为 0 归并，依次得 $RTNXPL$，最后一位为 1 依次得 $ASOIGEAME$，右边是各自的 5 位 0，1 符号串(见图 10.3.1 中的(2))。(2)中的第 4 位依 0 及 1 归并，依次得(3)，…，最后得

$A\ A\ E\ E\ G\ I\ L\ M\ N\ O\ P\ R\ S\ T\ X$

基数分类法的时间复杂度与关键字的个数 N 及关键字的长度 K 成正比。故时间复杂度为 $O(KN)$。基数分类法时间复杂度的表现形式不同于前面的几种分类方法。但它仍然没有违背开始关于分类最坏复杂度下界为 $O(n\log_2 n)$ 的基本原则。这是因为假定以 b 为基(上例中 $b=2$)，则表示 n 个元素至少需 $\log_b n$ 位。这样总共需要 $nb\log_b n$ 次比较(需 $\log_b n$ 趟，每趟需 b 次比较——有 b 个关键字)，当 $b=e$ 时，取得最小值为 $en\ln n = O(n\log n)$

习　题

1. 如果序列已经是排序的，则归并分类要进行多少次比较？

2. 对下列英文字进行基数分类

COW，DOG，SEA，RUG，ROW，MOB，BOX，TAB，BAR，EAR，TAR，DIG，BIG，TEA，NOW，FOX。

3. 如若 n 是奇数，序列 $a_1, a_2, \cdots, a_n$ 如何利用 Ford-Johnson 归并分类法进行排序？试举例说明。

第 11 章　求第 k 个元素

实际应用中感兴趣的另一问题是寻找某一序列 $\{x_1,x_2,\cdots,x_n\}$ 中第 k 小的元素．特别地，寻找序列的中位数(可以想象，求中位数最困难)。这问题最直接了当的解法是先将序列排好序，从而可得出第 k 小的元素。从前面几章的讨论可知，最少需 $O(n\log_2 n)$ 次比较，即时间复杂度为 $O(n\log_2 n)$。只求其中一个元素能否有更有效的算法呢？事实上，在相当一段时间内，寻找求第 k 个元素的线性复杂度算法曾困扰了算法界，由 Rabin 所解决，并经 Knuth 简化得到了本书所要叙述的结果。

11.1　求最小及第二小元素

当 $k=1$ 和 $k=2$ 时问题便变成求最小和第 2 小的元素。

已知序列 $x_1,x_2,\cdots,x_n$，假定 n 个元素各不相等。在 n 个元素中要淘汰掉 $n-1$ 个元素保留最小的一个。所以无论采用什么方法都要作 $n-1$ 次比较，但若要继续求第 2 小元素却不必要再作 $n-2$ 次比较。因为第 2 小元素必然是在求最小元素的过程中被淘汰。若充分利用求最小元素时所获得的信息，则所需比较次数将会减少。以球赛来模拟两数的比较，产生冠军的过程可用一棵二分树来表示。n 个"叶结点"用以代表 n 位选手，两两比赛产生一位"胜利者"，作为它们的"父亲"，由 $\frac{n}{2}$ 位胜利者进入第二轮的比赛，如此反复直到产生冠军为止。可见图 11.1.1。

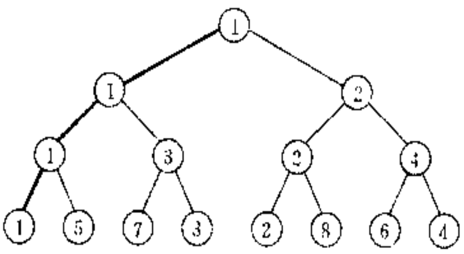
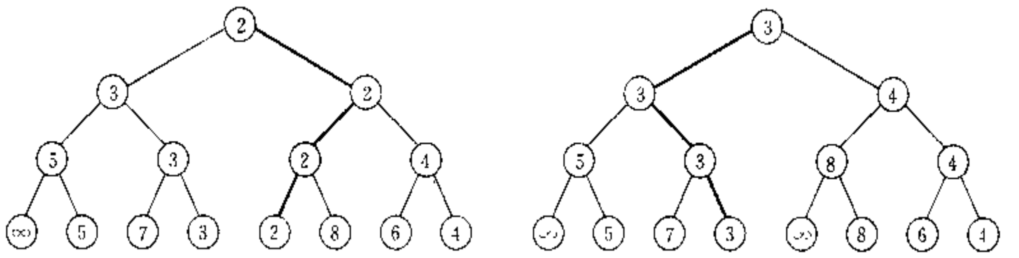


图　11.1.1

图中粗线表示"冠军"产生的过程。一般说来，n 位选手应进行 $\lceil \log_2 n \rceil$ 轮比赛方能产生"冠军"。"冠军"在最坏情况下必须战胜 $\lceil \log_2 n \rceil$ 位选手，"亚军"必然在这 $\lceil \log_2 n \rceil$ 位"失败者"中。故若最小元素的求得需要 $n-1$ 次比较，则求第 2 小元素仅需作 $\lceil \log_2 n \rceil-1$ 次比较。以图 11.1.1 为例，求第 2，3 小元素的过程可描述如图 11.1.2 所示。



(a) (b)

图 11.1.2

从上可知求最小和第 2 小元素共需进行：

$$(n-1)+(\lceil \log_2 n \rceil-1)=n+\lceil \log_2 n \rceil-2$$

次比较而不是 $2n-3$ 次比较。

11.2 求第 k 个元素

下面讨论不通过对序列 $x_1,x_2,\cdots,x_n$ 进行分类，而找出第 k 个元素 $x_{(k)}$ 的算法。

(1) 将 $x_1,x_2,\cdots,x_n$ 分成 $\left[\frac{n}{15}\right]$ 行，每行 15 个元素，对它们各自进行分类。由每行的第 8 个元素构成一序列 $C=\{c_1,c_2,\cdots,c_m\}$。这儿假定 $n=15m$。

(2) 求序列 C 的中间元素，设为 x。

(3) 图 11.2.1(b)中 A 部分的所有元素都比 x 小，B 部分的所有元素都比 x 大，C 和 D 两部分的元素有的比 x 大，有的比 x 小，但对 C、D 两部分，可通过 2 分法，每行作 3 次比较便可确定有多少元素比 x 小，从而确定元素 x 的序数。

(4) 如若 x 的序数正好是 k，则 x 便是所求的 $x_{(k)}$。若 x 的序数比 k 小，则 $x_{(k)}$ 不可能在 A 中出现。可将 A 部分从讨论的序列中除去。问题导致在余下的 $\frac{3}{4}n$ 个元素中求第 k_1 个元素。k_1 可通过简单的计算得出。如若 x 的序数比 k 大，则 $x_{(k)}$ 不可能出现在 B 中，将 B 部分从序列中去掉。

(5) A(或 B)被去除后，适当调整归并 C(或 D)部分，使每行仍为 15 个元素。

(6) 继续以上的步骤，直至元素个数少于 64 个为止。当元素个数少于 64 时，可采用适当的分类法加以分类，找出相应的元素。

具体请参看图 11.2.1。

设 $q(n)$ 表示从 n 个元素中找出第 k 个元素在最坏情况下所需的比较次数，$p(n)$ 为 n 个元素分成 15 个元素一行，而且每行已分类好的情况下，求第 k 个元素所需作的比较次数。

由于 15 个元素进行分类只要作 42 次比较，

则有：

$$q(n)=p(n)+42\left(\frac{n}{15}\right) \tag{11.2.1}$$

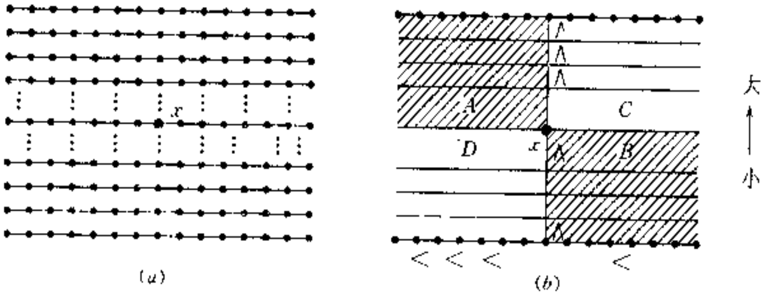


图 11.2.1

但

$$p(n)=q(\frac{n}{15})+3(\frac{n}{15})+\frac{13}{2}(\frac{n}{30})+p(\frac{3}{4}n) \qquad (11.2.2)$$

其中$q(\frac{n}{15})$为从每行第8个元素新组成的集合，求中间元素x所作的比较次数；$3(\frac{n}{15})$为确定C和D部分各行比x小的元素个数所作的比较次数；$\frac{13}{2}(\frac{n}{30})$为弃去$A$(或$B$)后，对$C$和$D$内各行进行归并调整使之每行仍旧为15个元素所作的比较次数，$p(\frac{3}{4}n)$为从余下的$\frac{3}{4}n$个元素中求第k_1个元素所需的比较次数。

由(11.2.1)得：

$$q(\frac{n}{15})=p(\frac{n}{15})+42(\frac{n}{225})$$

代入(11.2.2)得：

$$p(n)=p(\frac{3}{4}n)+p(\frac{n}{15})+42(\frac{n}{225})+\frac{3}{5}n+\frac{13}{60}n$$

即

$$p(n)=p(\frac{3}{4}n)+p(\frac{n}{15})+0.6033n \qquad (11.2.3)$$

式(11.2.3)是非线性递推关系，显然$p(0)=0$。可通过迭代法求解。有：

$$\begin{aligned}p(n)&=p(\frac{3}{4}n)+p(\frac{n}{15})+0.6033n\\&=p(\frac{9}{16}n)+p(\frac{n}{225})+2p(\frac{n}{20})+0.6033(\frac{3}{4}+\frac{1}{15})n\end{aligned}$$

即

$$p(n)=p(\frac{9}{16}n)+p(\frac{1}{225}n)+2p(\frac{n}{20})+0.6033\cdot\frac{49}{60}n$$

依此反复迭代，而且由于：

$$\lim_{n\to 0}p(n)=0$$

故令

$$p(n)=an$$

代入(11.2.2)得：

$$an = \frac{3}{4}an + \frac{a}{15}n + 0.6033n$$

$$a = 3.2913$$

$$\therefore \quad q(n) = 3.2913n + \frac{42}{15}n = 6.09n$$

$$q(n) = 6.09n$$

故其时间复杂度为 $O(n)$。是线性的。

习　题

1. 证明在最坏情况下，同时找出 n 个元素中的最大值和最小值必需要 $\lceil 3n/n \rceil$ 次比较。

2. 假定一个算法仅仅使用比较来寻找 n 个元素中的第 i 个元素，则它必能同时找到较少的 $i-1$ 个元素和 $n-i$ 个较大的元素而不用执行额外的操作。

3. 令 $X[1..n]$ 和 $Y[1..n]$ 是两个已排序的数组，试设计一个算法来找出这 $2n$ 个数的中间值，并讨论其时间复杂度。

4. 试用迭代法论证递推关系

$$p(n) = p(\frac{3}{4}n) + p(\frac{n}{15}) + 0.6033n$$

$$p(0) = 0$$

有解，存在常数 a 使得

$$p(n) = an$$

5. 设计一个算法，找出一个集合中的最大值和第二大值。当集合大小为 2 的幂时，并讨论算法需要多少次比较？

6. 证明即使在最坏情况下，六次比较也足以找出 5 个数中的中间值。而将 5 个数排序至少需要 7 次比较。

7. 设计一个算法找出 n 个数中的第 3 大值，并分析其复杂性。这类算法是否有必要确定最大值和第 2 大值？

8. 假定 l_1、l_2 是两个各包含 n 个值的按升序排列的数组：

(1) 设计算法找出这 $2n$ 个数中第 n 小的元素（假定这 $2n$ 个数各不相同），讨论其复杂度。

(2) 给出这个问题的下界。

9. 假设你有一台较小内存的计算机。现在有 n 个关键字存在外存（磁盘或磁带）上。关键字将被读到内存中处理，但只被读入一次。

(1) 找出其中的最大值需要的存贮单元至少是多少？

(2) 找出其中的中间值需要的存贮单元至少是多少？

第 12 章 外存分类法

前面讨论的分类技术都是针对有限量的关键字而言的，使得这些关键字在内存中容纳得下。而许多重要的分类问题涉及到非常大的量，不可能，也不允许全部驻留在内存中，必须存放在外存储器中。利用外存设备进行分类的方法称为外存分类法。

一般来说，外存分类花费在内外存之间数据调进与调出的时间比这些数据在内存里的处理与加工的时间要高出几个数量级，而且内外存调度方式又极大程度上依赖于外存设备本身(如磁带只能顺序存放)，所以外存分类所考虑的复杂度侧重面完全不同于内存分类。因此，在研究外存分类时，必须考虑存放在哪种外存上。由于磁带和磁盘是目前最广泛使用的外存储器，而且其存贮方法有代表性，所以下面主要研究磁带和磁盘的外存分类方法。

12.1 外存归并分类法

外存分类最常用的方法是归并分类(merge sorting)，这种分类方法既适用于磁盘，又适用于磁带，但又不完全一样。

归并分类的基本思想是：第一步，把待分类文件逐段(段的长度依内存空间而定)输入到内存，并逐段作内存分类，然后写回磁盘或磁带，这些已分类的段叫初始归并段。第二步，对这些初始归并段作多遍归并，每遍归并后，段长增加，段数减少，直至在外存上形成单一归并段为止。

假定开始时要分类的序列录在某一带上，长度为 n，内存分类只能容纳长度为 m 的一段。也就是说一次可以输出一段长度为 m 的已分类完毕的序列。现以 4 条带的外存分类算法为例描述如下(其中假定待分类的序列贮存于 T_1 带上)：

第 1 阶段：若 T_1 还有未分类的文件则作：

① 读进长度为 m 的记录(归并段)，并利用内存分类法进行分类；

② 将分类的结果交替地记录在 T_1 和 T_2 带上。

第 2 阶段：回绕磁带。

若 T_1 和 T_2 存在归并段，则作：

① T_1 带上的第一段和 T_2 带上的第一段进行归并，将结果记录在 T_3 带上；归并算法见第 10 章归并分类法。

② T_1 带上的第二段和 T_2 带上的第二段进行归并，将结果记录在 T_4 带上；

③ 重复步骤①和②直到 T_1 和 T_2 上的记录全部归并完毕为止。

④ 磁带回绕准备下面一轮的归并。将 T_1，T_2 由原来的输入带改为输出带，而 T_3，T_4 由原来的输出带改为输入带，再进入第二阶段，继续归并，直至产生单一的归并段。

例如： 设 $m=6$

T_4: 25 48 19 14 93 45 13 23 11 2 38 18 24 32 97 17 50 15 21 68 77 41 62 44 73 95 71 64

100 36 90 65 33 74 55 69 35 80 12 # #

其中# #为终止符号。

第一阶段后:$m=6$,故有长度$\leqslant 6$的段分别记在T_1和T_2上,即

T_1: 14 19 25 45 48 93 # 15 17 24 32 50 97 #

36 64 71 73 95 100 # 12 35 80 # #

T_2: 2 11 13 18 23 38 # 21 41 44 62 68 77 #

33 55 65 69 74 90 # #

磁带回绕后进行归并得长度$\leqslant 12$的段分别记录在T_3和T_4上。即

T_3: 2 11 13 14 18 19 23 25 38 45 48 93 #

33 36 55 64 65 69 71 73 74 90 95 100 # #

T_4: 15 17 21 24 32 41 44 50 62 68 77 97 #

12 35 80 # #

磁带回绕后再进行归并得

T_1: 2 11 13 14 15 17 18 19 21 23 24 25 32 38 41 44 45 48 50 62 68 77 93 77 # #

T_2: 12 33 35 36 55 64 65 69 71 73 74 80 90 95 100 # #

磁带回绕后进行第三轮的归并:

T_3:2 11 12 13 14 15 17 18 19 21 23 24 25 32 33 35 36 38 41 44 45 48 50 55 61 64 65 68

69 71 73 74 77 80 90 93 95 97 100 # #.

其归并过程可描述如图 12.1.1 所示。

$$
\begin{pmatrix} [1]^4\wedge \\ [1]^3\wedge \\ \wedge\phi \\ \wedge\phi \end{pmatrix}
\xrightarrow{\text{回绕}}
\begin{pmatrix} \wedge[1]^4 \\ \wedge[1]^3 \\ \wedge\phi \\ \wedge\phi \end{pmatrix}
\xrightarrow{\text{归并}}
\begin{pmatrix} \wedge[1]^2 \\ \wedge[1] \\ [2]\wedge \\ [2]\wedge \end{pmatrix}
\xrightarrow{\text{归并}}
\begin{pmatrix} \wedge\phi \\ \wedge\phi \\ [2]^2\wedge \\ [2][1]\wedge \end{pmatrix}
\xRightarrow{\text{回绕}}
$$

$$
\begin{pmatrix} \wedge\phi \\ \wedge\phi \\ \wedge[2]^2 \\ \wedge[2][1] \end{pmatrix}
\xrightarrow{\text{归并}}
\begin{pmatrix} [4]\wedge \\ [3]\wedge \\ \phi\wedge \\ \phi\wedge \end{pmatrix}
\xrightarrow{\text{回绕}}
\begin{pmatrix} \wedge[4] \\ \wedge[3] \\ \wedge\phi \\ \wedge\phi \end{pmatrix}
\xrightarrow{\text{归并}}
\begin{pmatrix} \phi\wedge \\ \phi\wedge \\ [7]\wedge \\ \phi\wedge \end{pmatrix}
\xRightarrow{\text{回绕}}
\begin{pmatrix} \wedge\phi \\ \wedge\phi \\ \wedge[7] \\ \wedge\phi \end{pmatrix}
$$

图 12.1.1

其中$\wedge$是读写头的标志,作为约定$[1]^4\wedge$表示在带的读写头前面有 4 个初始或长度为 1 的归并段。$\wedge[1]^4$表示回绕后读写头在前面;其余类似。

$$
\begin{pmatrix} \wedge[1]^4 \\ \wedge[1]^3 \\ \wedge\phi \\ \wedge\phi \end{pmatrix}
\Rightarrow
\begin{pmatrix} \wedge[1]^2 \\ \wedge[1] \\ [2]\wedge \\ [2]\wedge \end{pmatrix},
$$

表示T_1带上的第一归并段和T_2带上的第一归并段归并为一归并段记录在T_3带上,[2]表示为 2 个初始归并段归并而得长度为 2 的归并段。T_1和T_2的第二段归并后放在T_4带

上，故 T_1 带上保留 2 个初始归并段，T_2 带上留下一初始归并段，T_3 和 T_4 的读写头位于记录后面。余此类推。

一般地对 n 个元素序列，用 1 条磁带进行分类，如上所述，第一阶段共分类成 $\left\lceil \frac{n}{k} \right\rceil = r$ 段，k 是初始分段的长度。

第二阶段的第一轮归并成 $\lceil r/2 \rceil$ 段，第二轮归并为 $\left\lceil \frac{r}{4} \right\rceil$ 段，第三轮归并为 $\left\lceil \frac{r}{2^i} \right\rceil$ 段，故第二阶段要进行 $\lceil \log_2 r \rceil$ 轮的归并。带的回绕数为 $\lceil \log_2 r \rceil + 1$，设 m 是内存分类的段的长度 $k \leq m$。

12.2 置换选择段的构造

从前面可以看到初始段的长度 k 直接影响到带的回绕次数，机械速度较电子速度要慢得多得多。所以希望内存产生的段的长度尽可能长。下面介绍一种置换选择法。

设 m 是内存允许的分类记录的数目。先将 m 个记录读进内存，将最小关键字输出。此后再读进一关键字取代这输出的元素，再从中输出比前一个输出的关键字大的内存里最小关键字，如此反复，直到内存中所有的关键字都比已输出的小为止。还是以前一个例子为例：

25 48 19 14 93 45 13 23 11 2 38 18 24 32 97 17 50 ;15 21 68 77 41 …

先取前 $k(=6)$ 个元素进入内存：

25 48 19 14 93 45

⇓输出 14 读进 13

25 48 19 93 45 13

⇓输出 19 读进 23，13 留在内存

25 48 93 45 (13) 23

⇓输出 23 读进 11

25 48 93 45 (13) (11)

⇓输出 25，读进 2，11 留在内存

48 93 45 (13) (11) (2)

⇓输出 45，读进 38，2 留在内存

48 93 (13) (11) (2) (38)

⇓输出 48，读进 18，38 留在内存

93 (13)(11) (2) (38) (18)

⇓输出 93，读进 24，18 留在内存

(13) (11) (2) (38) (18) (24)

其中○内数表示输出的元素，()的数滞留在内存，故输出的段有 7 个元素：

14 19 23 25 45 48 93

新的一轮重新开始

$13\ 11\ (2)\ 38\ 18\ 24 \xRightarrow[\text{读进32}]{\text{输出2}} 13\ (11)\ 38\ 18\ 24\ 32 \xrightarrow[\text{读进97}]{\text{输出11}} (13)\ 38\ 18\ 24\ 32\ 97 \xrightarrow[\text{读进17}]{\text{输出13}} 38\ 18\ 24$
$32\ 97\ (17) \xRightarrow[\text{读进50}]{\text{输出17}} 38\ (18)\ 24\ 32\ 97\ 50 \xRightarrow[\text{读进(15)}]{\text{输出18}} 38\ (24)\ 32\ 97\ 50(15) \xrightarrow[\text{读进(21)}]{\text{输出24}} 38\ (32)\ 97\ 50\ (15)\ (21)$
$\xRightarrow[\text{读进68}]{\text{输出32}} (38)\ 97\ 50\ (15)\ (21)\ 68 \xRightarrow[\text{读进77}]{\text{输出38}} 97\ (50)\ (15)\ (21)\ 68\ 77 \xRightarrow[\text{读进(41)}]{\text{输出50}} 97\ (15)\ (21)\ (68)\ 77\ (41)$
$\xRightarrow[\text{读进(62)}]{\text{输出68}} 97\ (15)\ (21)\ (77)\ (41)\ (62) \xRightarrow[\text{读进(44)}]{\text{输出77}} (97)\ (15)\ (21)\ (41)\ (62)\ (44) \xrightarrow[\text{读进(73)}]{\text{输出97}} (15)\ (21)$
$(41)\ (62)\ (44)\ (73)$

至此输出第 2 段，共 12 个元素：

2　11　13　17　18　24　32　38　50　68　77　97。

此过程可继续下去，直至产生全部的初始归并段。

段的长度估计

对置换选择法产生的段的长度作精确的估计是非常困难的，只能作一些人为的假设，给出一个近似的结果。假定内存容许的数据量相当大，关键字是实数区间[0,1]中的某一个实数，输入到内存的数看作是雪花均匀地以常速落到环形的长度为 1 的跑道上，不妨设速率=1 单位/s。扫雪机沿着环形跑道运动，因内存保持 m 个关键字不变，相当于扫雪机的扫出的速率和雪花落到环形跑道的速率相等，都是 1 单位/s，这意味着扫雪机的前进速度和该点的雪的高度成反比，这些假定都是在 m 相当大时接近于正确。若扫雪机在 O 点出发，当扫雪机回到 o 点时，开始新的一段，如图 12.2.1。所以，段的长度=扫出的雪的数量。

开始时内存有 m 个随机数，相当于 m 个单位的雪花均匀地落在环形跑道上，即雪的初始高度为 m。随着扫雪机的前进，扫雪机前雪的高度在单调增加，开始后时刻 t，雪的高度 $h(t)$ 为 $m+t$，因假定下雪的速度为 1 单位/s，令 $h(t)$ 表 t 时刻雪的高度，$x(t)$ 表 t 时刻扫雪机的位置，下面给出初始段的估计，反复几次后渐趋稳定，稳定情况的讨论读者可自己思考。则

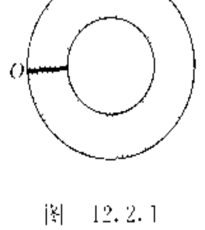


图　12.2.1

$$h(t)\frac{dx}{dt}=1 \quad 或 \quad \frac{dx}{dt}=\frac{1}{m+t} \quad x(0)=0$$

$$\therefore \quad x(t)=\ln c_1(m+t)$$

$$x(0)=\ln c_1 m=0 \quad c=\frac{1}{m}$$

即

$$x(t)=\ln\frac{m+t}{m}$$

当 $m+t/m=e$ 或 $t=m(e-1)$ 时，$x=1$。由于假定扫雪机的扫雪速率为 1 单位/s，故第一周期的扫雪量应为 $m(e-1)$，即初始段的长度应近似等于 $m(e-1)$。

置换选择法可以进一步改善，办法如下：即当内存里的最小数比刚已输出的数小时；可以认为它对这一段不起作用，而把它从内存中移到缓冲区去，再读进一个随机数。例如 $m=4$，随机序列为：

7 2 5 3 1 9 6 8 4

$\Downarrow$ 读进 4 个数到内存

(2 3 5 7)

$\Downarrow$ 输出 2，读进 1

(1 3 5 7)

$\Downarrow$ 输出 1 到缓冲区，读进 9。

(3 5 7 9)

$\Downarrow$ 输出 3，读进 6

(5 6 7 9)

$\Downarrow$ 输出 5，读进 8。

(6 7 8 9)

$\Downarrow$ 输出 6，读进 4。

(4 7 8 9)

$\Downarrow$ 输出 7，输出 4 到缓冲区。

(8 9)

$\Downarrow$ 输出 8。

(9)

$\Downarrow$ 输出 9

ϕ

故缓冲区中只有 1，4 两个数，得到经过分类的序列：

2 3 5 6 7 8 9

改善的置换选择法的长度为 em。证明从略。

12.3 三条带的外存归并分类法

1. 三条带的情形

对于三条带作 2 路归并的按 Fibonacci 序列做初始分布的思想，可以推广到一般更多带（$\geqslant 4$）的情形。先讨论三条带的情况。

三条带的归并利用到 Fibonacci 数的特性：

$$F_n=F_{n-1}+F_{n-2},\quad n=3,4,\cdots$$

$$F_1=F_2=1$$

即序列 1，1，2，3，5，8，13，14，…例如对于 21 个长度为 1 个单位的初始段的归并，可以如下进行：

$$\begin{pmatrix}[1]^{13}\\ [1]^{8}\\ \phi\end{pmatrix}\Rightarrow\begin{pmatrix}[1]^{5}\\ \phi\\ [2]^{8}\end{pmatrix}\Rightarrow\begin{pmatrix}\phi\\ [3]^{5}\\ [2]^{3}\end{pmatrix}\Rightarrow\begin{pmatrix}[5]^{3}\\ [3]^{2}\\ \phi\end{pmatrix}\Rightarrow\begin{pmatrix}[5]^{1}\\ \phi\\ [8]^{2}\end{pmatrix}\Rightarrow\begin{pmatrix}\phi\\ [13]^{1}\\ [8]^{1}\end{pmatrix}\Rightarrow\begin{pmatrix}[21]^{1}\\ \phi\\ \phi\end{pmatrix}$$

即 21 个初始段，其中 13 个记在 T_1 上，8 个记在 T_2 上，T_3 空着。

∵ $$F_7=21=F_6+F_5=13+8$$

第一个"⇒"符号两端表示T_1上的8段分别和T_2上的8段归并为长度为2单位的8段记在T_3；T_1余下长度为1个单位的5段。第二个"⇒"符号两端表明T_1上长度为1个单位的5段分别与T_3上长度为2单位的前面5段归并为长度为3单位的5段记在T_2；T_3余下长度为2单位的3段，余此类推。

一般的，若两条带上段的数目为F_{n-1}和F_n，则归并结果使短的一条带空出，新的带的段的数目为F_{n-2}和F_{n-1}。

2. 分析

由于通过三条带产生一长度为F_n的段必须由长度为F_{n-1}和F_{n-2}的两段归并而成，则作如图12.3.1的Fibonacci树T_n，利用它来估计工作量。

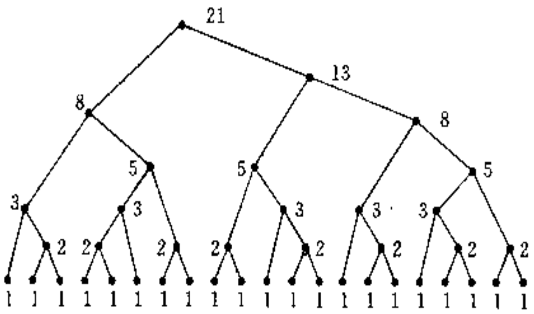


图 12.3.1

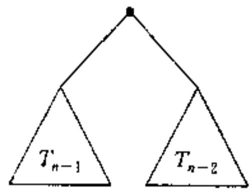


图 12.3.2

Fibonacci树T_n有F_n个叶子，以Fibonacci树的任一顶点为树根的子树仍是Fibonacci树。叶节点到树根的距离就是对应的初始段通过内存的次数。树T_n的结构如图12.3.2所示。Fibonacci树的叶节点到树根并非都是等距离，换句话说，各初始段通过内存的次数不相同。

令g_n表示有F_n个叶结点的Fibonacci树T_n的叶节点到树根的距离之和，则：

$$g_n=g_{n-1}+g_{n-2}+F_n,\quad n\geqslant 2$$

$$g_0=g_1=0$$

但 $$F_n=F_{n-1}+F_{n-2}\quad,\quad F_0=F_1=1$$

∴ $$F(x)=\sum_{k=0}F_kx^k=1/1-x-x^2$$

故

$$G(x)=\sum_{k=0}g_kx^k=\sum_{k=2}g_kx^k$$

$$=\sum_{k=2}g_{k-1}x^k+\sum_{k=2}g_{k-2}x^k+\sum_{k=2}F_kx^k$$

$$=x\sum_{k=2}g_kx^k+x^2\sum_{k=2}g_kx^k+\frac{1}{1-x-x^2}-1-x$$

∴ $$(1-x-x^2)G(x)=\frac{1}{1-x-x^2}-1-x$$

$$G(x)=\frac{1}{(1-x-x^2)^2}-\frac{1+x}{1-x-x^2}$$

但

$$1-x-x^2=\left(1-\frac{1+\sqrt{5}}{2}x\right)\left(1-\frac{1-\sqrt{5}}{2}x\right)$$

$$=(1-\alpha x)(1-\beta x)$$

$$\alpha=\frac{1+\sqrt{5}}{2}\quad,\quad\beta=\frac{1-\sqrt{5}}{2}$$

$$\frac{1}{1-x-x^2}=\frac{A}{1-\alpha x}+\frac{B}{1-\beta x}$$

不难求得 $\qquad A=\dfrac{\alpha}{\alpha-\beta}=\dfrac{1+\sqrt{5}}{2\sqrt{5}},B=\dfrac{-\beta}{\beta-\alpha}=-\dfrac{1-\sqrt{5}}{2\sqrt{5}}$

∴

$$G(x)=\frac{A^2}{(1-\alpha x)^2}+\frac{B^2}{(1-\beta x)^2}+\frac{2AB}{(1-\alpha x)(1-\beta x)}$$

$$-(1+x)\left(\frac{A}{1-\alpha x}+\frac{B}{1-\beta x}\right)$$

$$\frac{1}{(1-\alpha x)^2}=(1+\alpha x+\alpha^2x^2+\cdots)(1+\alpha x+\alpha^2x^2+\cdots)$$

$$=1+2\alpha x+3(\alpha x)^2+4(\alpha x)^3+\cdots$$

同理 $\qquad \dfrac{1}{(1-\beta x)^2}=\sum\limits_{k=1}k(\beta x)^{k-1},\quad$ 由于 $G(x)=\sum\limits_{k=0}g_kx^k$

∴ $\qquad g_n=A^2(n+1)\alpha^n+B^2(n+1)\beta^n+k_1\alpha^n+k_2\beta^n$

其中 $\qquad k_1=2AB-A-\dfrac{A}{\alpha},\quad k_2=2AB-B-\dfrac{B}{\beta}.$

n 充分大时 $\beta^n\to0$. 故

$$g_n\approx A^2(n+1)\alpha^n+k_1\alpha^n=A^2n\alpha^n+(A^2+k_1)\alpha^n$$

$$\alpha=\frac{1}{2}(1+\sqrt{5})=1.618034$$

$$A=\frac{1+\sqrt{5}}{2\sqrt{5}}=0.7236068,\ B=\frac{-(1-\sqrt{5})}{2\sqrt{5}},$$

$$2AB=2\frac{1-5}{4.5}=\frac{-4}{10}=-0.4$$

∴ $\qquad k_1=-0.4-0.7736068-0.4472042=-0.820811$

∴ $\qquad g_n\approx A^2n\alpha^n+(A^2+K_1)\alpha^n=0.5236068n\alpha^n+(0.5236068-0.820811)\alpha^n$

$$=0.5236068n\alpha^n-0.2972042\alpha^n$$

3. 非 Fibonacci 数的情形

先以三条带 13 段为例分析相应的归并次数如下，不难看到分段的归并次数不尽相同。

设 $\qquad$ 3 3 3

$\qquad$ (3 2 2)

表示长度为 3 的三段，其归并次数分别为 3，2，2 次．即括号里的数表示相应的归并次数．计算归并次数是自底向上回溯进行的，其过程可描述如表 12.3.1 所示．

表 12.3.1

T_1	T_2	T_3
1 1 1 1 1 ⋮ 1 1 1 (5)(4)(4)(4)(3) ⋮ (4)(3)(3)	1 1 1 1 1 (5) (4) (4) (4) (3)	
1 1 1 (4)(3)(3)		2 2 2 ⋮ 2 2 (4) (3) (3) ⋮ (3) (2)
	3 3 3 (3) (2) (2)	⋮ 2 2 ⋮ (3) (2)
5 ⋮ 5 (2) ⋮ (1)	5 (2)	
5 (1)		8 (1)
	13 (0)	

例如最后一行 T_2 上有 $\frac{13}{(0)}$，即长度为 13 单位的最后结果作为计数的始点，它是由 T_1 上的 5 单位长度的段和 T_3 上的 8 单位长度的段归并所得的；故倒数第二行有：

T_1	T_2	T_3
5 (1)		8 (1)

同样的理由 T_3 上 8 单位是由 T_1 的 5 单位一段和 T_2 上的 3 单位一段归并而得，故有倒数第三行，

T_1	T_2	T_3
5 5 (2)(1)	3 (2)	

余此类推．表中虚线右方表示归并后余下的部分．

表 12.3.1 中第一行 54443433　54443 括号里 13 个数正是 Fibonacci 树 13 个叶结点分别到树根的距离，参见图 12.3.1。也就分别是 13 段初始段归并过程通过内存的次数。

一般情况下，即当段的数目不是正好为某一 Fibonacci 数时，例如若有 10 段，可使其中 6 段在 T_1，4 段在 T_2，有 3 段是空的，这 3 个空白的段应使之正好是归并次数最多的三个。例如：

T_1	T_2
0 0 0 1 1 1 1 1	1 1 1 1 1
(5 4 4 4 3 4 3 3)	(5 4 4 4 3)

其中“0”表示空白段。显然任一分类好的段与空白段的归并就是该段本身,归并的工作当然可以省去。

通常各条带前面部分的各段归并次数较高,带的终端归并次数少一些,所以多余的空白段应均匀地分配在每条带的前面为好。

12.4 阶式归并法

当有多条带时,可将三条带的方法加以推广,得到阶式归并法。通过一个具体例子说明阶式归并方法1如图 12.4.1 所示。由此不难推广到一般的情形:

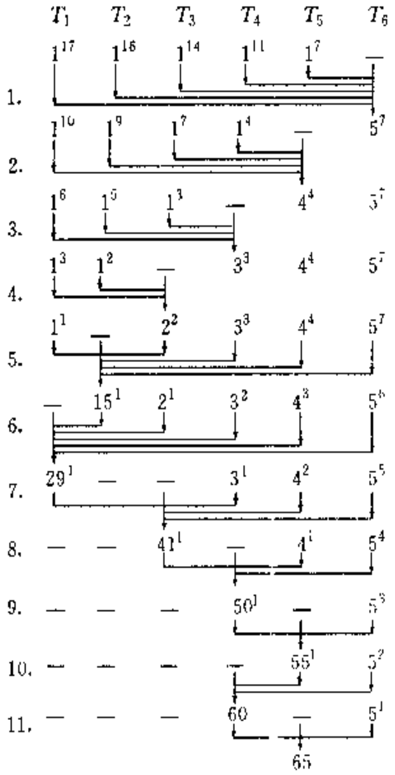


图 12.4.1

归并过程叙述如下:

(1) 1^{17}指的是长度为 1 单位的共 17 段,初始状态为 T_1 上记录了长度为 1 的段共 17 段,T_2 上记录了长度为 1 的段共 15 段(记作 1^{15}),…,T_6 为空。

(2) 从 T_1 到 T_5 每条带上各取 5 段归并成长度为 5 单位的段共 7 段(记作 5^7),记录在 T_6 上。此时 T_1 上记录为 1^{10},T_2 上为 1^9,…T_5 为空。

(3) 从 T_1 到 T_4 各取 4 段,归并成长度为 4 单位的段共 4 段(记作 4^4),记录在 T_5 上,T_4 空;

(4) 从 T_1 到 T_3 各取 3 段,归并成长度为 3 单位的段共 3 段,记录在 T_4 上(记为 3^3),T_3 为空。

此过程可继续下去，直到成为单一的归并段。

习　　题

1. 证明：令 F_j 是第 j 个 Fibonacci 数，则 $j\geqslant 5$ 时

$$\frac{F_{j+1}}{F_j}\leqslant\frac{13}{8}$$

2. 假设你有一量大的未排序文件在磁带上且只有两个磁带机，设计一个方法将文件排序。

3. 已知序列

25，47，19，14，93，45，13，23，10，8，38，18，24，30，97，17，50，15，21，68，77，41，62，44，73，95，71，64，102，36，90，65，33，74，55，69，35，80，12。

利用三条带的外存分类进行排序，设内存分类段的长度 $m=6$，试绘图说明其排序过程。

4. 试利用外存对下面序列进行段的构造，并叙述分类的过程。设 $m=6$

23，46，17，12，94，43，11，21，8，6，36，16，22，28，95，15

5. 如若有经过排序 18 段，试讨论如何利用三条带进行归并。叙述其归并过程。

6. 试利用 5 条带的归并算法对下面问题进行排序，已知四条带分别有 26，15，28，22 段已排序的段。

7. 假定有 31 段已排好序的段分别存储在三条带上，依次有 13，11，7 段。试用 4 条带进行归并。

8. 如何对存在一盘上的文件进行分类，除主内存外只有一条带可供使用。

第13章　分类网络

13.1　分类网络举例

前面讨论的内存和外存分类算法都是在串行计算机上进行，就是说某一时刻只能进行一个操作，这一章将讨论用硬件实现分类的问题，即分类专用装置——分类网络。

分类网络的基本元件是比较元件，输入经过有限时间的滞后，输出端分别给出两个输入量的最大、最小值，见图 13.1.1 所示。

图　13.1.1

为方便起见，以后的讨论中可用图 13.1.2 来表示。

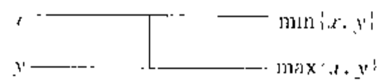


图　13.1.2

例如图 13.1.3 是一个具有 4 个输入端的分类网络。特别在 $A=3$、$B=2$、$C=1$、$D=4$ 的输入下，图 13.1.4 给出了上述输入通过分类网络，得到最后分类结果的过程。

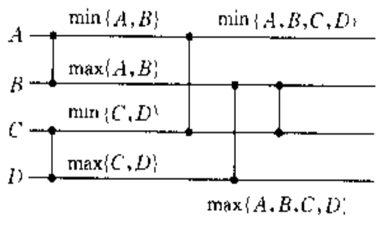


图　13.1.3

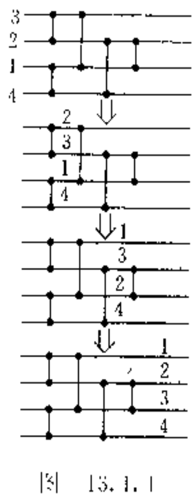


图　13.1.4

有些分类网络是依据分类算法来设计的，比如图 13.1.5 的网络是根据直接插入分类法而设计的。图中的虚线仅仅是对网络功能的划分，以便于理解。

图 13.1.6 是另一种依据 *Shell* 分类算法得到的分类网络。

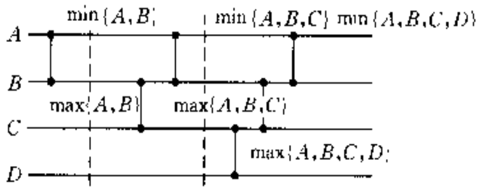


图 13.1.5

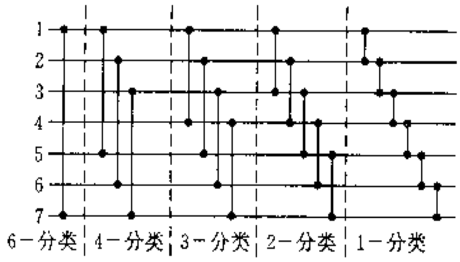


图 13.1.6

13.2 0-1 原 理

当我们研究一种分类网络时，如何验证其正确性呢？最简单的办法是将 $n!$ 种排列都输入，一一进行验证。但通过本节所讲述的 0-1 原理可将复杂性降到 2^n，重要的是采用 0-1 原理使得网络正确性的讨论分析更易进行。这在后面的讨论中将可看到。

0-1 原理指若分类网络对于任一输入序列 x_1，x_2，…，x_n 为{0,1}序列时正确，则对任意输入也一定正确，这样在我们构造分类网络后只需着眼于考虑 0-1 序列作用下的结果是否正确(见图 13.2.1)。

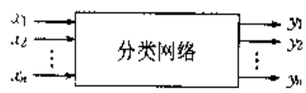


图 13.2.1

设函数 $h(x)$ 是单调增的，即若 $a<b$，则 $h(a)<h(b)$，则输入为 $h(x)$、$h(y)$ 时，比较元件的输出将分别为($\min\{h(x),h(y)\}$)，($\max\{h(x),h(y)\}$)。可用图 13.2.2 表示。

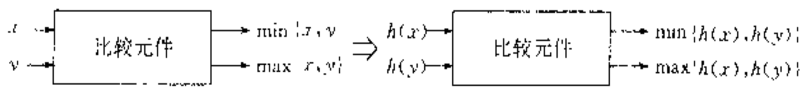


图 13.2.2

因 $h(x)$ 单调增，故

$$\min\{h(x),h(y)\}=h(\min\{x,y\})$$

$$\max\{h(x),h(y)\}=h(\max\{x,y\})$$

即有(图 13.2.3)：

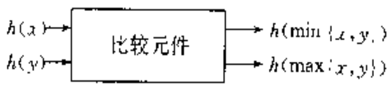


图 13.2.3

由此可得下面的结论：

若分类网络将输入序列$\{x_1,x_2,\cdots,x_n\}$转换为输出序列$\{y_1,y_2,\cdots,y_n\}$，则对于任一单

调增函数 h，分类网络将输入序列 $\{h(x_1),h(x_2),\cdots,h(x_n)\}$ 转换为输出序列 $\{h(y_1),h(y_2),\cdots,h(y_n)\}$。即对任一单调增函数 h，对于如图 13.2.1 所示的分类网络，将有如图 13.2.4 所示的结果。

图 13.2.4

定理(0-1 原理)　一个具有 n 个输入端的分类网络工作正确的充要条件是 n 个输入端为 0,1 时工作正确。

证明　若分类网络对任一 0-1 序列工作正确，但存在一个序列，分类网络工作失败，即存在一序列 $\{x_1,x_2,\cdots,x_n\}$，存在 x_i 和 x_j，$x_i<x_j$，但分类网络将 x_j 置于 x_i 前面。定义一单调增序列：

$$h(x)=\begin{cases}0 & x\leqslant x_i\\ 1 & x>x_i\end{cases}$$

从图 13.2.4 可知该网络输入为 $h(x_1)h(x_2)\cdots h(x_n)$ 时，输出时 $h(x_j)$ 将在 $h(x_i)$ 之前，也就是说发生 1 在 0 之前的情形这与假定（分类网络对任意 0,1 序列工作正确）相矛盾，对 0 1 序列工作正确是网络工作正确的充分条件得到了证明。

反之显然。因 0-1 序列是一种序列。分类网络若对一 0 1 序列工作不正确，网络工作一定不正确。必要性得到证明。

下面将通过例子，可以看到如何用 0 1 原理分析网络正确性。本节先介绍 B_n 型网络，如图 13.2.5 所示。这里不妨假定 n 为偶数，第 i 个输入 x_i 和第 $i-\frac{n}{2}$ 个输入 $x_{i-\frac{n}{2}}$ 进行比较，$i=1,2,\cdots,\frac{n}{2}$。这样构造的网络称之为 B_n 型网络。这 B_n 型网络有什么特点呢？先看一个例子，如图 13.2.6。

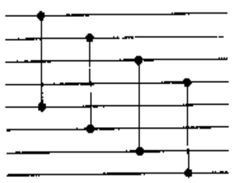


图 13.2.5

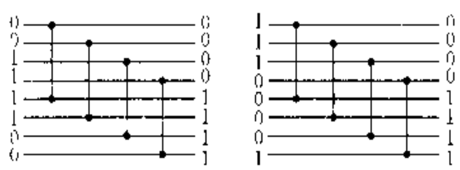


图 13.2.6

一个序列若是从单调增到单调减，或从单调减到单调增，称为双调序列。如 9　7　6　5　8　10　11　或 2　4　5　7　9　6　3　1 就是双调序列，特别，当序列为 0,1 序列时，双调序列有如下形式：

(a) $\underbrace{0\ \ 0\ \cdots 0}_{p}\ \underbrace{1\ \ 1\ \cdots 1}_{q}\ \underbrace{0\ \ 0\ \cdots 0}_{r}$ $\qquad p+q+r=n$

(b) $\underbrace{1\ \ 1\ \cdots 1}_{i}\ \underbrace{0\ \ 0\ \cdots 0}_{j}\ \underbrace{1\ \ 1\ \cdots 1}_{k}$ $\qquad i+j+k=n$

下面来讨论 B_n 型网络对双调序列的分类：

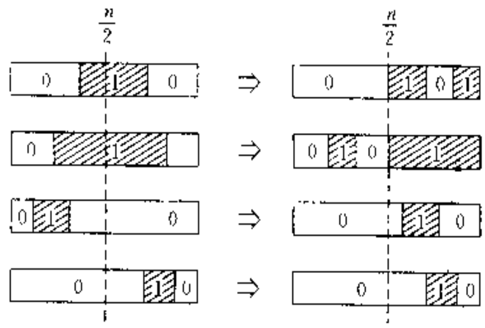


图 13.2.7

如图 13.2.7 左边为输入端的双调序列状态，右端为输出端的状态。讨论的输入序列都是(a)所示的双调序列，对于(b)，读者可自己讨论、总之双调序列输入 B_n 型分类网络得到的输出序列至少有一半是全 0 或全 1 的序列。即一半为

$$0\ \ 0\cdots0 \text{ 或 } 1\ \ 1\cdots1$$

另一半还是双调序列。

利用 0-1 原理和 B_n 型网络的性质可构造分类网络如图 13.2.8 所示。

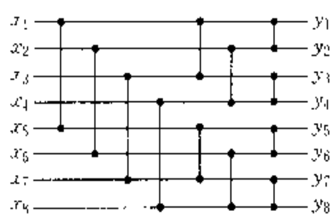


图 13.2.8

很容易知道，B_n 型网络对双调序列能正确分类。

令 $n=2^k$，分类网络所需的比较元件数为 a_k，则：

$$a_{k+1} = 2a_k + 2^k,\ a_1 = 1$$

令

$$A(x) = a_1 + a_2 x + a_3 x^2 + \cdots\cdots$$

$$A(x) - 1 = 2xA(x) + \frac{2x}{1-2x}$$

$$(1-2x)A(x) = \frac{1}{1-2x}$$

$$A(x) = \frac{1}{(1-2x)^2}$$

$$a_k = k2^{k-1}$$

故 n 个输入端由 B_n 型网络构成的分类网络需要

$$\frac{1}{2}n\log_2 n$$

个比较元件。

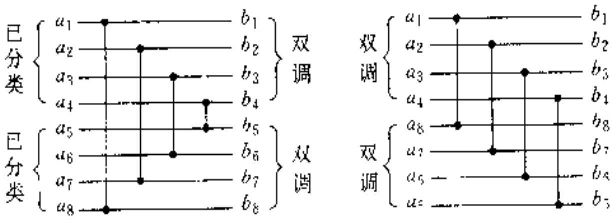
13.3 归并网络

这一节将研究一种将上节的双调分类网络加以修改得到的归并网络，它将已排序完毕的两个序列归并成一排好的序列。

设 $\boldsymbol{a}=\underbrace{0\ 0\ \cdots\ 0}_{h}\ \underbrace{1\ 1\ \cdots\ 1}_{k}$，$\boldsymbol{b}=\underbrace{0\ 0\ \cdots\ 0}_{p}\ \underbrace{1\ 1\ \cdots\ 1}_{q}$ 两组已分类好的序列，将 $\boldsymbol{b}$ 倒装在 $\boldsymbol{a}$ 的后面成为一个长为 $L=(p+q+h+k)$ 的双调序列：

$$\underbrace{0\ 0\ \cdots\ 0}_{h}\ \underbrace{1\ 1\ \cdots\ 1}_{k+q}\ \underbrace{0\ 0\ \cdots\ 0}_{p}$$

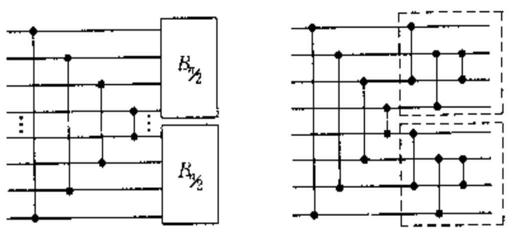
再利用上一节讨论的 B_n 型分类网络进行排序。这就是归并网络的思路。归并网络的第一步是将输入的第 i 个元素与第 $n-i+1$ 个元素进行比较，$i=1,2,\cdots,\frac{n}{2}$。输出至少有一半是全 0 或全 1，另一半是双调序列。图 13.3.1 给出归并网络与 B_n 网络的比较。理解了这一点，也就清楚了归并网络的原理。



(a) 归并网络第一步　　(b) 对应于(a)的 B_n 第一步

图 13.3.1

双调序列可通过 B_n 型网络进行分类。图 13.3.2 给出归并网络的构造的例子。

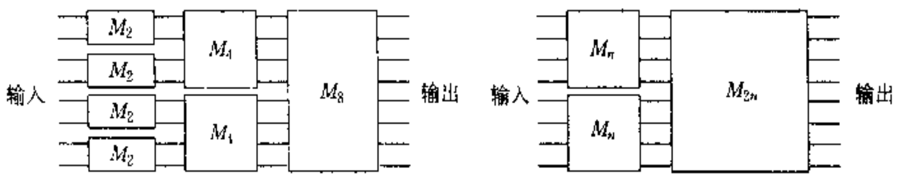


(a) M_n 分类网络　　(b) M_8 分类网络

图 13.3.2

由上所述，两个分类完毕、长度各为 $n/2$ 的序列可通过归并网络进行归并，最后输出的是由这两个序列分类完毕长度为 n 的序列。可以递归地利用这思想，前面长度为 $n/2$ 的两个序列(已排好序)也可以通过归并网络由两个长度各为 $n/4$ 的排好序的序列归并而成。依此类推。这些想法和归并排序法一样，不再重复。利用这思想构造分类网络如下：

利用归并网络构造分类网络如图 13.3.3，其中用 M_n 表示输入端为 n 的归并网络。



(a) (b)

图 13.3.3

归并网络的分析留给读者思考。

13.4 Batcher 奇偶归并网络

下面介绍的分类法采用的算法是基于分治策略，把两个序列各分成两半，前一半先进行分类，后一半也同样进行分类得两个有序的子序列，再利用归并网络进行归并。这样，Batcher 奇偶归并网络的算法可描述如下：

假定 $x_1<x_2<\cdots<x_m$，$y_1<y_2<\cdots<y_n$ 是两组已排好序的序列，则

第一步：$x_1,x_3,x_5\cdots$ 和 $y_1,y_3,y_5\cdots\cdots$ 归并得序列 $z_1,z_2,z_3,\cdots$；

$x_2,x_4,x_6\cdots$ 和 $y_2,y_4,y_6\cdots$ 归并得序列 $w_1,w_2,w_3\cdots$。

第二步：依顺序 $z_1,w_1,z_2,w_2,\cdots$ 利用(2,3)，(4,5)，…比较元件作用于这序列。其中(2,3)比较元件是指连序列的第 2 和第 3 元素间的比较元件，余此类推。

在证明算法的正确性之前，先看几个例子。当 $m=2$，$n=1$ 时，如图 13.4.1 所示。网络将 x_1,x_2,y_1 进行归并分类，解决了三个输入端的问题。图 13.4.2 是 $m=2$，$n=2$ 时的例子，图 13.4.3 是 $m=3$，$n=2$ 时的例子。图 13.4.4 是 $m=5$，$n=4$ 时的例子。

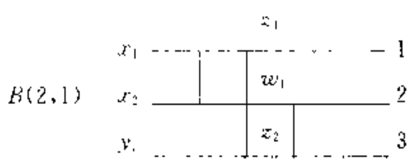


图 13.4.1

图中有用粗线刻划的，只不过用以标记对 $x_2,x_4,\cdots,y_2,y_4,\cdots$ 构成的偶数下标序列的分类。以图 13.4.4 为例，其中粗线所刻划的图实际上还是 $B(2,2)$ 的情形。当虚线右方 x_1，x_3,x_5,y_1,y_3 的分类用的是 $B(3,2)$ 型网络。最右端的标号 1,2,3,…,9，是根据顺序 z_1，$w_1,z_2,w_2,z_3,w_3,z_4,w_4,z_5$。明白了这些，便不难理解(2,3)，(4,5)，…比较元件的含义。

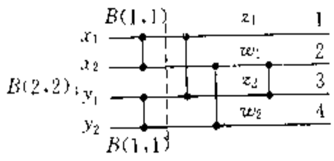


图 13.1.2

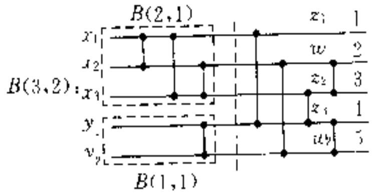


图 13.1.3

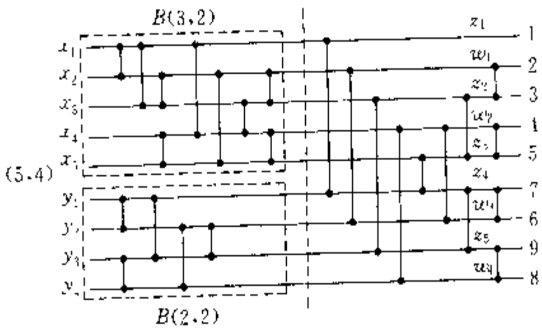


图 13.1.4

下面利用 0-1 原理对 Batcher 奇偶归并网络的正确性进行证明。

设
$$x_1x_2\cdots x_m=\underbrace{0\ \ 0\ \cdots\ 0}_{l}\ \ \underbrace{1\ \ 1\ \cdots\ 1}_{m-l}$$
$$y_1y_2\cdots y_n=\underbrace{0\ \ 0\ \cdots\ 0}_{k}\ \ \underbrace{1\ \ 1\ \cdots\ 1}_{n-k}$$

奇偶归并得:
$$z_1z_2\cdots z_p=\underbrace{0\ \ 0\ \cdots\ 0}_{r_1}\ \ \underbrace{1\ \ 1\ \cdots\ 1}_{s_1}$$
$$w_1w_2\cdots w_q=\underbrace{0\ \ 0\ \cdots\ 0}_{r_2}\ \ \underbrace{1\ \ 1\ \cdots\ 1}_{s_2}$$

其中 $r_1=\left\lceil \frac{l}{2} \right\rceil+\left\lceil \frac{k}{2} \right\rceil$, $r_2=\left\lfloor \frac{l}{2} \right\rfloor+\left\lfloor \frac{k}{2} \right\rfloor$

显然有 $r_1-r_2=0,1,2$ 三种可能,分别讨论如下:

(1)
$$\begin{array}{l} z_1z_2\cdots z_p=0\ 0\ \cdots\ 0\ 1\ 1\ 1\ 1 \\ w_1w_2\cdots w_q=0\ 0\ \cdots\ 0\ 1\ 1\ 1 \end{array}\qquad r_1-r_2=0$$

(2)
$$\begin{array}{l} z_1z_2\cdots z_p=0\ 0\ \cdots\ 0\ 0\ 1\ 1\ 1 \\ w_1w_2\cdots w_q=0\ 0\ \cdots\ 0\ 1\ 1\ 1 \end{array}\qquad r_1-r_2=1$$

(3)
$$\begin{array}{l} z_1z_2\cdots z_p=0\ 0\ \cdots\ 0\ 0\ 0\ 1\ 1 \\ w_1w_2\cdots w_q=0\ 0\ \cdots\ 0\ 1\ 1\ 1 \end{array}\qquad r_1-r_2=2$$

斜线表示比较元件，情况(图 13.4.4)可通过有(*)号比较元件调整之，从而可知 Batcher 算法是正确的。

习 题

1. 令 $n=2^t$ 试用三种以上方法设计一个 n 路输入、n 路输出的网络，使得最上边的输出总是最小值，而最下面的输出总是最大值。

2. 有人认为如果在排序网络中任意位置加上一个比较器，其结果仍是一个排序网络。试通过图 13.1.4 解释这个概念是错误的。

3. 试证明任何 n 输入排序网络中比较器的个数至少为 $O(n\lg n)$

4. 证明 n-输入的比较网络能够正确对输入 $\langle n,n-1,\cdots,1\rangle$ 排序的主要条件是它能够对 $n-1$ 个 0-1 序列 $\langle 1,0,0,\cdots,0,0\rangle$、$\langle 1,1,0,\cdots,0,0\rangle$、…、$\langle 1,1,\cdots,1,0\rangle$ 正确排序。

5. 证明 n 输入的排序网络中，在 i 和 $i-1$ 行之间至少有一个比较器，$i=1,2,\cdots,n-1$。

6. 若 B_n 网络中输入的双调序列是由任意整数构成的，试证明排序结果满足以下性质：上半部和下半部仍旧是双调序列，并且上半部的元素必不大于下半部元素。

7. 由 0 和 1 构成的 n 长双调序列有多少?

8. 证明类似于 0-1 原理有：若一个比较网络可以排序任意 0-1 双调序列，则能排序由任意整数构成的双调序列。

9. 为了验证一个比较网络是一个归并网络，最少应当测试多少个不同的 0-1 序列。

10. 证明对于任何的归并网络，需要 $O(n\lg n)$ 量级的比较器数。

11. 假设我们希望将 $2n$ 个元素 $\langle a_1,a_2,\cdots,a_{2n}\rangle$ 分为 n 个最小的和 n 个最大的两组，证明在对 $\langle a_1,a_2,\cdots,a_n\rangle$ 和 $\langle a_{n+1},a_{n+2},\cdots,a_{2n}\rangle$ 分别排序后只需经过常数级时间的排序就可完成。

12. 试构造 $B(5,5)$，$B(6,5)$ 网络。

第 14 章　查找及均衡树

许多计算机的任务中一个基本的操作是查找：从已经存储在内存（或外存）的一批记录中，按照关键字找出所需的一个记录。查找的目的是为了取得关键字所对应的记录，并进行某种处理。两个最常见的用于查找的数据形式是字典和链表。在一些大型数据库中字典和链表也许很大，并且常常使用。基于不同的查找要求设计不同的数据结构和算法。本章将逐一介绍。

如同前面介绍的二分树一样，我们可以将各种查找算法分解成若干基本操作的组合，最常见的基本操作有：建立起数据结构；查找一个或多个具有指定关键字的记录；插入一个新记录；删除一个指定的记录；另外还有分裂、合并等操作。

14.1　AVL 树——关于高度均衡的二分树

对 n 个元素进行查找，二分查找是非常有效的，而且顺序存储计算地址比较方便。但若考虑到新元素的插入、或删除一个元素，便暴露出顺序存储这种数据结构的弱点，因为它将引起平均 $\frac{n}{2}$ 个元素的移动。Adelson-Velski 和 landis 提出了一类均衡树，来解决这类问题，通常称之为 AVL 树。这节介绍 AVL 树之一——关于高度均衡的二分树。

设 T_l 和 T_r 分别是二分树 T 的左子树和右子树，称 T 为关于高度均衡的二分树（图 14.1.1），若满足

(1) $|h(T_l)-h(T_r)|\le 1$，

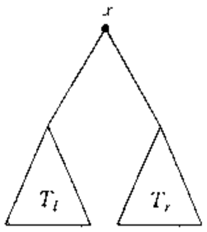


图 14.1.1

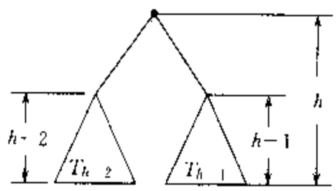


图 14.1.2

(2) T_l 和 T_r 也分别是关于高度均衡的二分树。其中 $h(T_l)$，$h(T_r)$ 分别指子树 T_l，T_r 的高度。条件(1)说明左、右子树的高度相差不超过 1；条件(2)是递归方式说明对任一内点其左右子树的高度差不超过 1。

下面对关于高度均衡的二分树的查找时间进行分析。设高度为 h 的二分树 T_h，其左子树是高度为 $h-1$ 的二分树 T_{h-1}，右子树是高度为 $h-2$ 的二分树 T_{h-2}（图 14.1.2）。设高度为 h 的关于高度均衡二分树在最坏情况下的节点数为 b_h，则有：

$$b_h = b_{h-1} + b_{h-2} + 1$$

$$b_0 = 1, b_1 = 2$$

令 $$G(x) = b_0 + b_1 x + b_2 x^2 + \cdots$$

$$x^2: b_2 = b_1 + b_0 + 1$$

$$x^3: b_3 = b_2 + b_1 + 1$$

$$+)\cdots\cdots\cdots\cdots\cdots\cdots$$

$$G(x) - 2x - 1 = x(G(x) - 1) + x^2G(x) + \frac{x^2}{1-x}$$

$$(1 - x - x^2)G(x) = 1 + x + \frac{x^2}{1-x} = \frac{1}{1-x}$$

$$G(x) = \frac{1}{(1-x-x^2)(1-x)}$$

令 $$\frac{1}{1-x-x^2} = \frac{A}{1-\alpha x} + \frac{B}{1-\beta x}$$

其中： $$\alpha = \frac{1+\sqrt{5}}{2},\quad \beta = \frac{1-\sqrt{5}}{2}$$

则 $$A = \frac{\sqrt{5}+1}{2\sqrt{5}}, B = 1 - A = \frac{\sqrt{5}-1}{2\sqrt{5}}$$

∴ $$G(x) = \frac{1}{\sqrt{5}}[(\alpha - \beta) + (\alpha^2 - \beta^2)x + \cdots](1 + x + x^2 + \cdots)$$

所以

$$b_k = \frac{1}{\sqrt{5}}[\alpha + \alpha^2 + \cdots + \alpha^{k+1} - (\beta + \beta^2 + \cdots + \beta^{k+1})]$$

$$= \frac{1}{\sqrt{5}}\left[\frac{\alpha - \alpha^{k+2}}{1-\alpha} - \frac{\beta - \beta^{k+2}}{1-\beta}\right]$$

$$= \frac{1}{\sqrt{5}}\left[\frac{\beta}{1-\beta}\beta^{k+1} - \frac{\alpha}{1-\alpha}\alpha^{k+1}\right] - 1$$

$$= \left(\frac{1}{2} + \frac{3}{2\sqrt{5}}\right)\alpha^{k+1} + \left(\frac{1}{2} - \frac{3}{2\sqrt{5}}\right)\beta^{k+1} - 1$$

由于 $\beta = \frac{1-\sqrt{5}}{2} = -0.618$，故 $\beta^k \to 0 (k \to \infty)$

所以 k 相当大时，

$$b_k \approx \left(\frac{1}{2} + \frac{3}{2\sqrt{5}}\right)\left(\frac{1+\sqrt{5}}{2}\right)^{k+1} = 1.1708\left(\frac{1+\sqrt{5}}{2}\right)^{k+1}$$

$$b_k \approx 1.89\left(\frac{1+\sqrt{5}}{2}\right)^k$$

$$\log_2 b_k \approx \log_2 1.89 + k\log_2\left(\frac{1+\sqrt{5}}{2}\right)$$

∴ $$k \approx (\log_2 b_k - \log_2 1.89)/\log_2\left(\frac{1+\sqrt{5}}{2}\right)$$

或写成：

$$h = \frac{\log_2 b_h}{\log_2\left(\frac{1+\sqrt{5}}{2}\right)} + O(1)$$

最后可得：

$$h \approx 1.4404\log_2 b_h$$

这公式给出关于高度均衡的二分树在最坏情况下，结点数与高度之间的关系。下面分析平均查找时间。

设 a_h 是高度为 h 的均衡二分树所有的内点查找所需比较次数的总和，即均衡树的所有内点到树根的距离总和，假定一子树的高度为 $h-1$，另一子树的高度为 $h-2$，则有

$$a_h = a_{h-1} + a_{h-2} + b_h - 1$$

$$a_0 = 0, a_1 = 1$$

令
$$c_h = \frac{a_h}{b_h}, c_0 = 0, c_1 = \frac{1}{2}$$

所以

$$\frac{a_h}{b_h} = \frac{b_{h-1}}{b_h}\cdot\frac{a_{h-1}}{b_{h-1}} + \frac{b_{h-2}}{b_h}\cdot\frac{a_{h-2}}{b_{h-2}} + 1 - \frac{1}{b_h}$$

由于

$$b_h \approx 1.89\left(\frac{1+\sqrt{5}}{2}\right)^h$$

$$\frac{b_{h-1}}{b_h} \approx \left(\frac{1+\sqrt{5}}{2}\right)^{-1}, \frac{b_{h-2}}{b_h} \approx \left(\frac{1+\sqrt{5}}{2}\right)^{-2}$$

有：

$$c_h = \left(\frac{1+\sqrt{5}}{2}\right)^{-1} c_{h-1} + \left(\frac{1+\sqrt{5}}{2}\right)^{-2} c_{h-2} + 1$$

令 $G(x) = c_0 + c_1 x + c_2 x^2 + \cdots\cdots$

$$x^2:\quad c_2 = \left(\frac{1+\sqrt{5}}{2}\right)^{-1} c_1 + \left(\frac{1+\sqrt{5}}{2}\right)^{-2} c_0 + 1$$

$$x^3:\quad c_3 = \left(\frac{1+\sqrt{5}}{2}\right)^{-1} c_2 + \left(\frac{1+\sqrt{5}}{2}\right)^{-2} c_1 + 1$$

+) ……………………………………………………………………

$$G(x) - \frac{1}{2}x = \left(\frac{1+\sqrt{5}}{2}\right)^{-1} xG(x) + \left(\frac{1+\sqrt{5}}{2}\right)^{-2} x^2G(x) + \frac{x^2}{1-x}$$

$$\left[1 - \left(\frac{1+\sqrt{5}}{2}\right)^{-1} x - \left(\frac{1+\sqrt{5}}{2}\right)^{-2} x^2\right]G(x) = \frac{x+x^2}{2(1-x)}$$

$$G(x) = \frac{x+x^2}{2(1-x)\left[1 - \left(\frac{1+\sqrt{5}}{2}\right)^{-1} x - \left(\frac{1+\sqrt{5}}{2}\right)^{-2} x^2\right]}$$

令 $y = \left(\frac{1+\sqrt{5}}{2}\right)^{-1} x$，则

$$\frac{1}{1 - \left(\frac{1+\sqrt{5}}{2}\right)^{-1} x - \left(\frac{1+\sqrt{5}}{2}\right)^{-2} x^2} = \frac{1}{1-y-y^2} = \frac{1}{(1-\alpha y)(1-\beta y)}$$

$$= \frac{A}{1-\alpha y} + \frac{B}{1-\beta y}$$

$$A = \frac{\sqrt{5}+1}{2\sqrt{5}}, B = \frac{\sqrt{5}-1}{2\sqrt{5}}, \alpha = \frac{1+\sqrt{5}}{2}, \beta = \frac{1-\sqrt{5}}{2}$$

所以：

$$\frac{1}{1-\left(\frac{1+\sqrt{5}}{2}\right)^{-1}x-\left(\frac{1+\sqrt{5}}{2}\right)^{-2}x^2}$$

$$= \frac{1}{\sqrt{5}}\left[(\alpha-\beta)+(\alpha^2-\beta^2)\alpha^{-1}x+(\alpha^3-\beta^3)\alpha^{-2}x^2+\cdots\right]$$

$$\frac{1}{(1-x)(1-\alpha^{-1}x-\alpha^{-2}x^2)}$$

$$= \frac{1}{\sqrt{5}}\left[(\alpha-\beta)+(\alpha^2-\beta^2)\alpha^{-1}x+(\alpha^3-\beta^3)\alpha^{-2}x^2+\cdots\right]\times(1+x+x^2+\cdots)$$

$$= \sum_{k=0} d_k x^k,$$

其中

$$d_k = \frac{1}{\sqrt{5}}\sum_{i=0}^{k}(\alpha^{i+1}-\beta^{i+1})\alpha^{-i}$$

$$= \frac{1}{\sqrt{5}}\left(\sum_{i=0}^{k}\alpha-\beta\sum_{i=0}^{k}\left(\frac{\beta}{\alpha}\right)^i\right) = \frac{1}{\sqrt{5}}\left[(k+1)\alpha-\beta\cdot\frac{1-\left(\frac{\beta}{\alpha}\right)^{k+1}}{1-\left(\frac{\beta}{\alpha}\right)}\right]$$

$$= \frac{1}{\sqrt{5}}\left[(k+1)\alpha-\beta\cdot\frac{\alpha}{\alpha-\beta}\cdot\frac{\alpha^{k+1}-\beta^{k+1}}{\alpha^{k+1}}\right]$$

$\therefore$
$$G(x) = \frac{x+x^2}{2(1-x)(1-\alpha^{-1}x-\alpha^{-2}x^2)} = \frac{1}{2}(x+x^2)\sum_{k=0}d_k x^k$$

$$= \frac{1}{2}\sum_{k=1}(d_k+d_{k-1})x^{k+1}$$

$\therefore$
$$c_h = \frac{1}{2}(d_{h-1}+d_{h-2})$$

$$= \frac{1}{2\sqrt{5}}\left[(2h-1)\alpha-\frac{\beta}{\alpha-\beta}\cdot\frac{2\alpha^h-(\alpha+\beta)\beta^{h-1}}{\alpha^{h-1}}\right]$$

$$= \frac{1}{2\sqrt{5}}\left[(2h-1)\alpha-\frac{1-\sqrt{5}}{2\sqrt{5}}\left(2\alpha-(\alpha+\beta)\frac{\beta^{h-1}}{\alpha^{h-1}}\right)\right]$$

当 h 充分大时，$\left(\frac{\beta}{\alpha}\right)^h\to 0$ 故有

$$c_h \approx 0.7236h$$
$$\approx 1.042\log_2 b_h$$

$c_h = a_h/b_h$ 实为平均查找所需的时间。

14.2 关于高度均衡的二分树的插入和删除

对于高度均衡的二分树，新的节点的插入，原有结点的删除，都有可能使其失去均衡性。以三个结点二分树为例，高度失去均衡的状态大致有以下几种：LL 型、LR 型、RL 型及 RR 型(图 14.2.1)，这几种构型具有典型性。

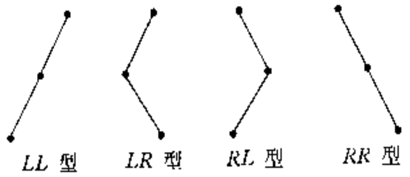


图 14.2.1

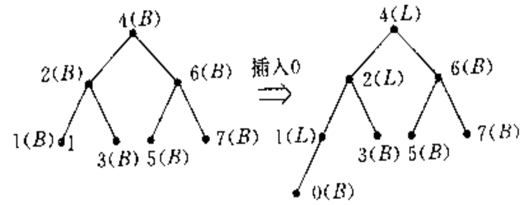


图 14.2.2

新结点的插入对均衡二分树的影响有以下 3 种情形：

(1) 原来是左、右高度相等，插入后左(右)比右(左)子树高出 1。如图 14.2.2。

(2) 原来是左(或右)高出 1。插入后使之左右高度相等，如图 14.2.3。

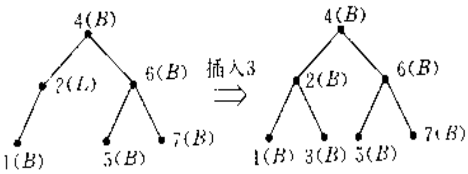


图 14.2.3

(3) 原来是左(右)高出 1 的高度均衡二分树，插入新结点使之失去均衡性。如图 14.2.4。

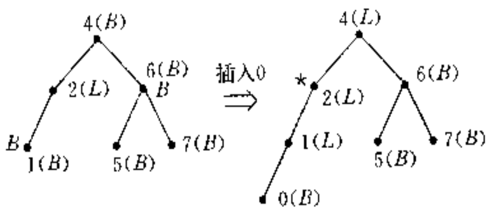


图 14.2.4

以上图中 B 表示平衡，L 表示左高，R 表示右高；* 表示失衡点。

下面介绍一般的调整失衡,使之恢复高度均衡的情况。

插入或删除一节点,判断哪些节点可能出现“失衡”现象?失衡时哪些结点出现“左高”或“右高”状态;这是算法的关键。至于调整使恢复关于高度均衡的步骤则是程式化的工作。当然假定在插入或删除前,二分树 T 是关于高度均衡的。

对结点 a,以它为根节点的子树又分为左、右两个子二分树,它们的高度记为 $h_l^{(a)}$,$h_r^{(a)}$,对每一节点 a 定义一标志数 $B_a=h_r^{(a)}-h_l^{(a)}$,如:

$$B_a=\begin{cases}0, & h_l^{(a)}=h_r^{(a)}\\ 1, & h_r^{(a)}=h_l^{(a)}+1\\ -1, & h_l^{(a)}=h_r^{(a)}+1\end{cases}$$

如图 14.2.5 所示。

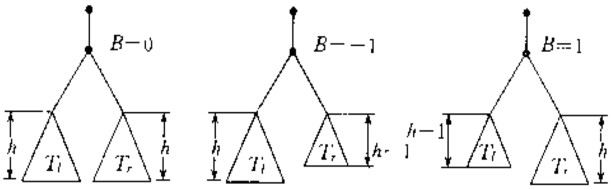


图 14.2.5

显然,若一内点 a,它的均衡标志数 B_a 为 1,新的插入点位于它的右子树改变了右子树的高度时,将出现不均衡现象,但需区分 RL 和 RR 两种情况。若插入点位于 a 点的右子树的左分子树,则出现 RL 不均衡现象。若插入点位于 a 点的右子树的右分子树,则出现 RR 不均衡现象。类似地可区分 LR 和 LL 不均衡现象。

从根节点到新插入点(新插入点本身除外)有一条路径 P。P 上最后一个均衡标志数不为零的点设为 b。有三种情况:

(1) 不存在这样的 b,即路径 P 所有的点均衡标志数均为零;

(2) 新插入的点为这样的 b 点的儿子结点;

(3) 新插入点和 b 点之间有一个或多个点。当然这些点的均衡标志数均为零。情况(1)和(2)都不会由于新插入的结点而引起不均衡现象发生,特别对于情况(2),插入前 b 点必有一儿子结点,均衡标志数非零,新插入点是 b 的另一个儿子,而 b 点的标志数改为零,不引起树的高度改变。只有情况(3)有此可能使原来均衡的树变成不均衡。

显然,若新插入的点改变了以某一点(设为 a)为根结点的子树的高度,将改变从树根通过该点 a 到插入点的高度。

如若 b 原来的均衡标志数为 1,而新插入点在它的右子树上,则新的标志数变为 2,当然这只是在情况(3)才会有的结果,而且有 RR 和 RL 之分。同样在情况(3),b 的标志数等于 -1,新插入点又在它的左子树,则标志数变为 -2,且有 LR 和 LL 之分。

下面介绍如何重建均衡树的方法。如图 14.2.6。

算法的形式化描述十分繁琐,但都是程式化的步骤,也就是说冗长但并不困难,这里略去。

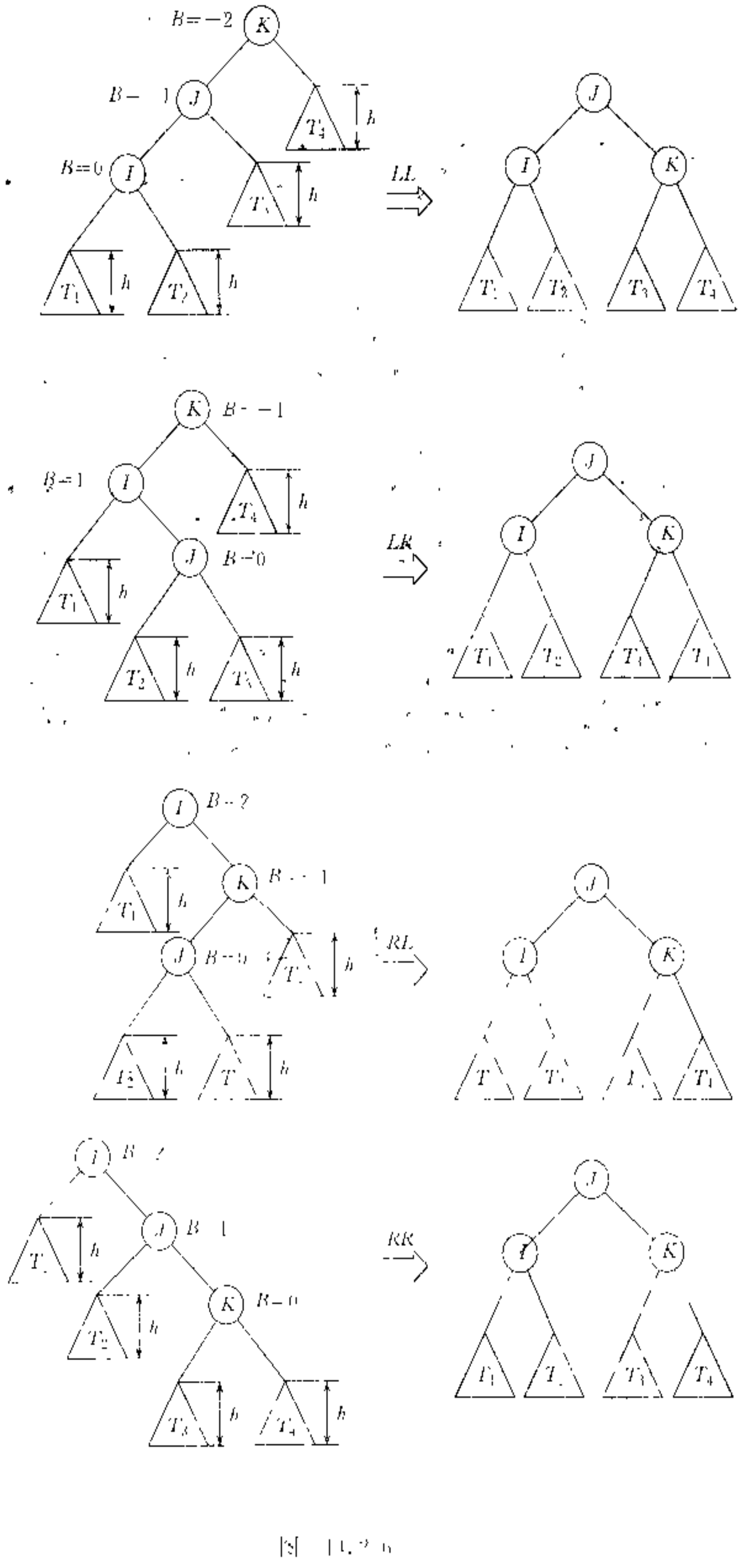


图 11.2.6

下面看一个例子。

已知一序列 8,9,10,2,1,5,3,6,4,7,11,12,将它按顺序插入,并调整保持关于高度

均衡二分树的全过程。如图 14.2.7。图中虚线围起的方格标明了失去均衡树特性的部分。

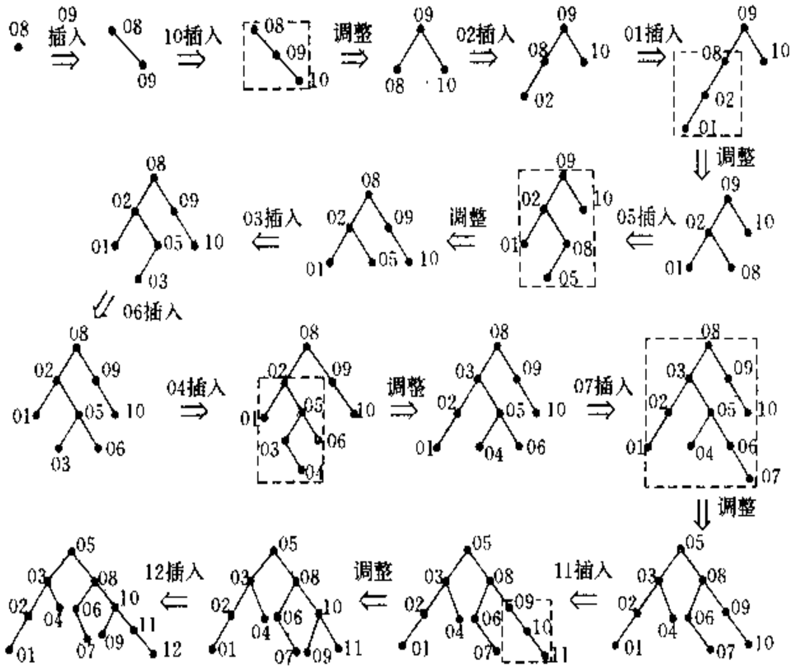
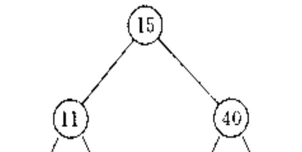


图 14.2.7

习　　题

1. 画出所有可能的 4 个节点的二分均衡树。
2. 将 30,5,70,80,50,依次插入到图题 11.1 的 AVL 树中,



图题 11.1

第 15 章　2-3 树和 2-3-4 树

15.1　2-3　树

2-3 树是一种关于高度均衡的树,它的任一内点有 2 个或 3 个"儿子"。所有叶片到根节点的距离都相等,只有叶节点才给出关键字,而内点只给出两个界和它相应的指针,见图 15.1.1。

图 15.1.1 给出了三个分支,x 是左分支的最大值,y 是中间分支的最大值,大于 y 的在右分支。当然也可能出现只有两个分支的情形,例如缺右分支。如图 15.1.2.

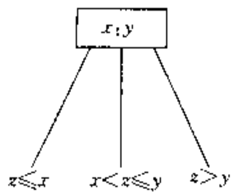


图　15.1.1

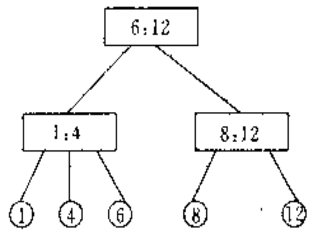


图　15.1.2

插入一个数可能有以下二种情况:

(1) 节点 $x:y$ 有两个"儿子"节点,若插入的值 $z \leqslant x$,则新入的点是它的"左儿子"节点(或称第一个儿子节点);若插入的值 $z>y$,则新插入的点是其"右儿子"节点(或称第 3 个儿子节点);此外插入的点是其"中间儿子节点"(第二个儿子节点),如图 15.1.3。

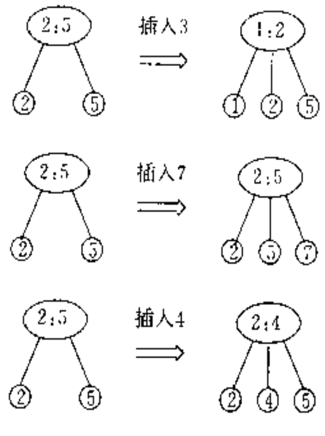


图　15.1.3

(2) 节点 $x:y$ 有三个"儿子"节点,则插入的 z 置于其适当的位置,这样 $x:y$ 点有 4 个"儿子",左边两个儿子和右边两个儿子各产生一节点,如图 15.1.4。

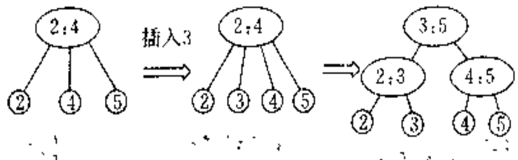


图 15.1.4

如若新增的内点又导致它的"父亲"节点有四个"儿子"节点，只需重复上述过程，如图15.1.5。

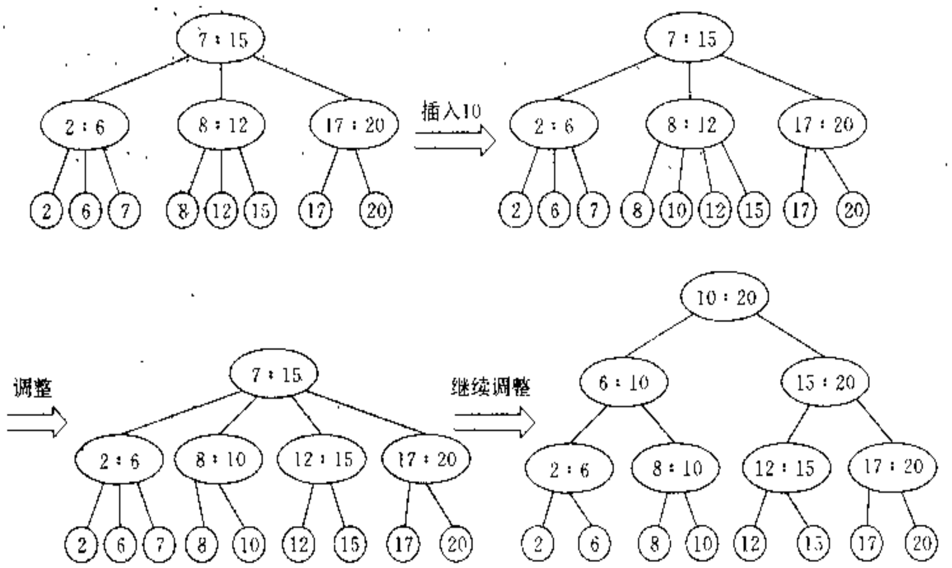


图 15.1.5

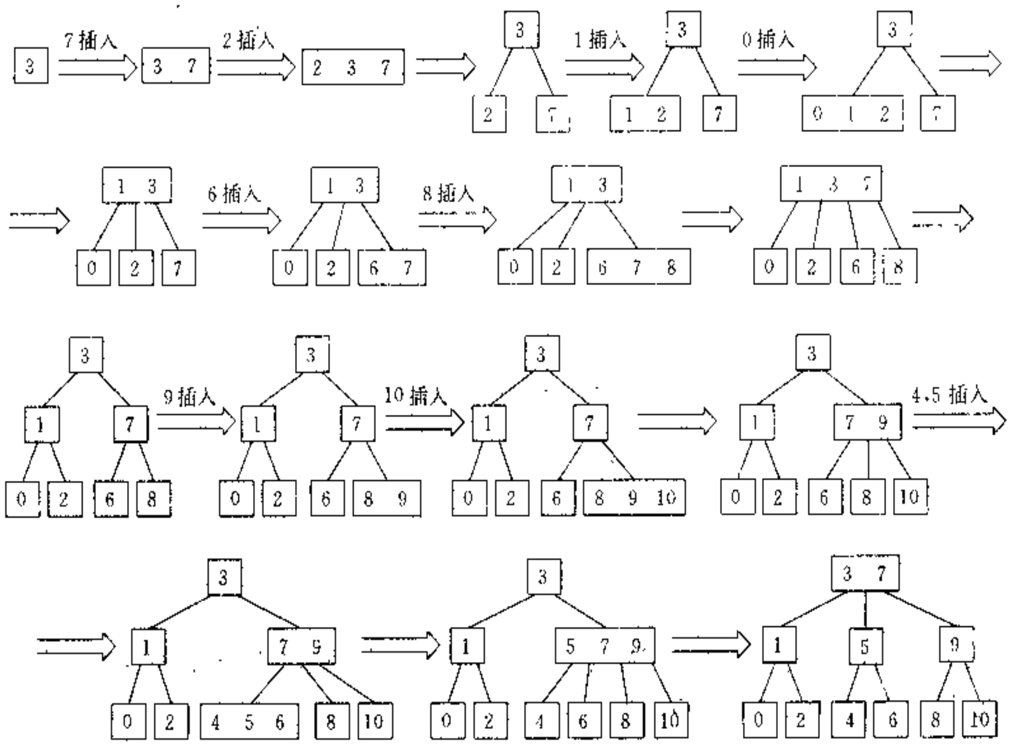


图 15.1.6

再举一例，如图 15.1.6 所示。

图 15.1.6 是序列

3，7，2，1，0，6，8，9，10，5，

插入初始为空的 2-3 树的全过程。

15.2 2-3-4 树

2-3-4 树和 2-3 树类似，是一种关于高度均衡树数据结构。它是具有下列性质的树：

(1) 每一个节点(根节点除外)具有 2，3 或 4 个“儿子”，有 k 个儿子的节点称为 k-节点。

(2) 所有的叶片具有相同的深度，也就是 2-3-4 树的高度

(3) 记录项存贮在叶子结点，每个内点包含以它的儿子为根结点的子树所存记录的关键字的最大值。

(4) 叶子节点所存贮的记录从左到右依次有序。

图 15.2.1 是 2-3-4 的一个例子。顺序是依英文字母表的顺序。

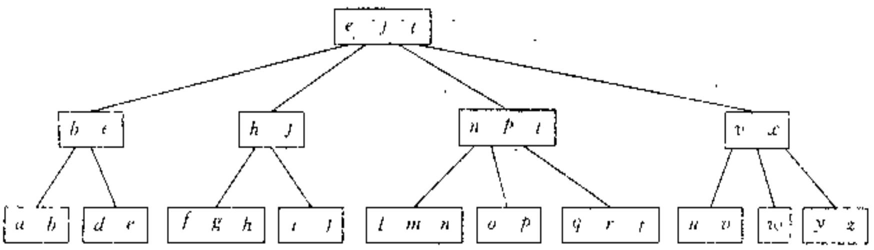


图 15.2.1

性质(3)和(4)使得在 2-3-4 树上查找比在二分树上查找仅稍微复杂，为了查找给定关键字的记录，我们从树根开始，使用节点内的信息确定查找路径。例如查找 r，由于 $j<r<t$ 故从根节点 (e,j,t) 向最右边的子树查找；下一步，由于 $p<r<t$，从节点 (n,p,t) 出发向其右边第 2 个子树查找，最后我们到达节点 (q,r,t) 的中间孩子，它包含关键字 r。

性质 1)说明高为 h 的 2-3-4 树，它的叶子可能有 2^h 到 4^h 项，另外由于查找时间与树的高度成正比，因此在含 n 项的 2-3-4 树中查找所需时间复杂度为 $O(\log_2 n)$。

2-3-4 树的插入

插入首先确定何处插入，须区分 2-节点，3-节点和 4-节点的情形。对于 2-节点和 3-节点只须更新内点的关键字信息。从而分别变为 3-节点和 4-节点。对于 4-节点的情况就不同了，举例说明如下(图 15.2.2)。

2-3-4 树的删除也必须区分节点类型，3-节点和 4-节点删除后分别变为 3-节点和 2

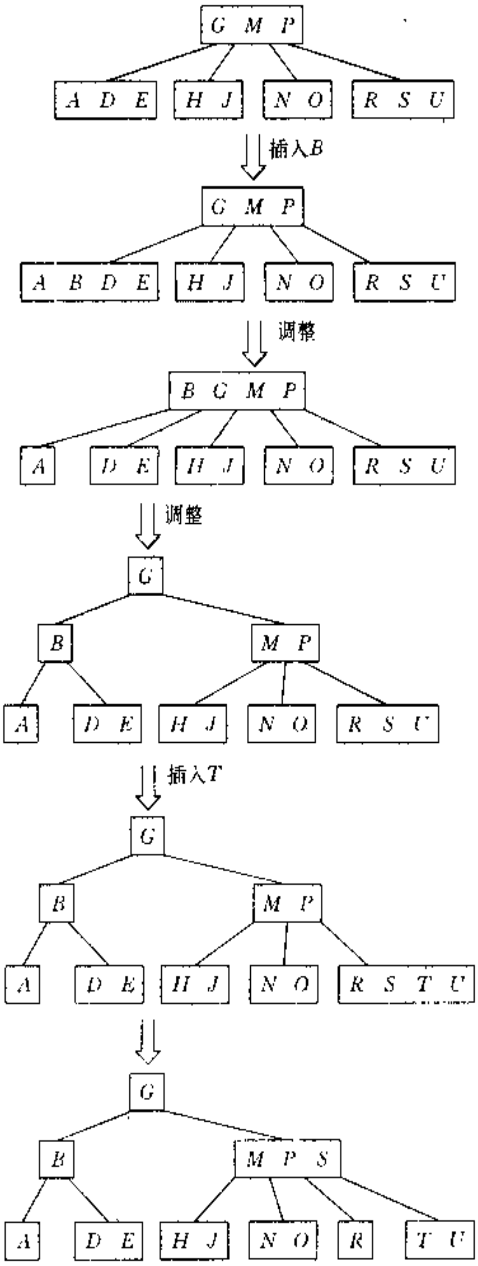


图 15.2.2

节点;但 2-节点删除后必须作相应处理。

删除后的处理留给读者作为习题。

最后举例说明一序列依次插入到 2-3-4 树中去的全过程(图 15.2.3),设序列为:

BJ,*HD*,*TH*,*RS*,*PR*,*FJ*,*AN*,*SD*,*GD*,*NJ*,*YN*,*ZJ*。

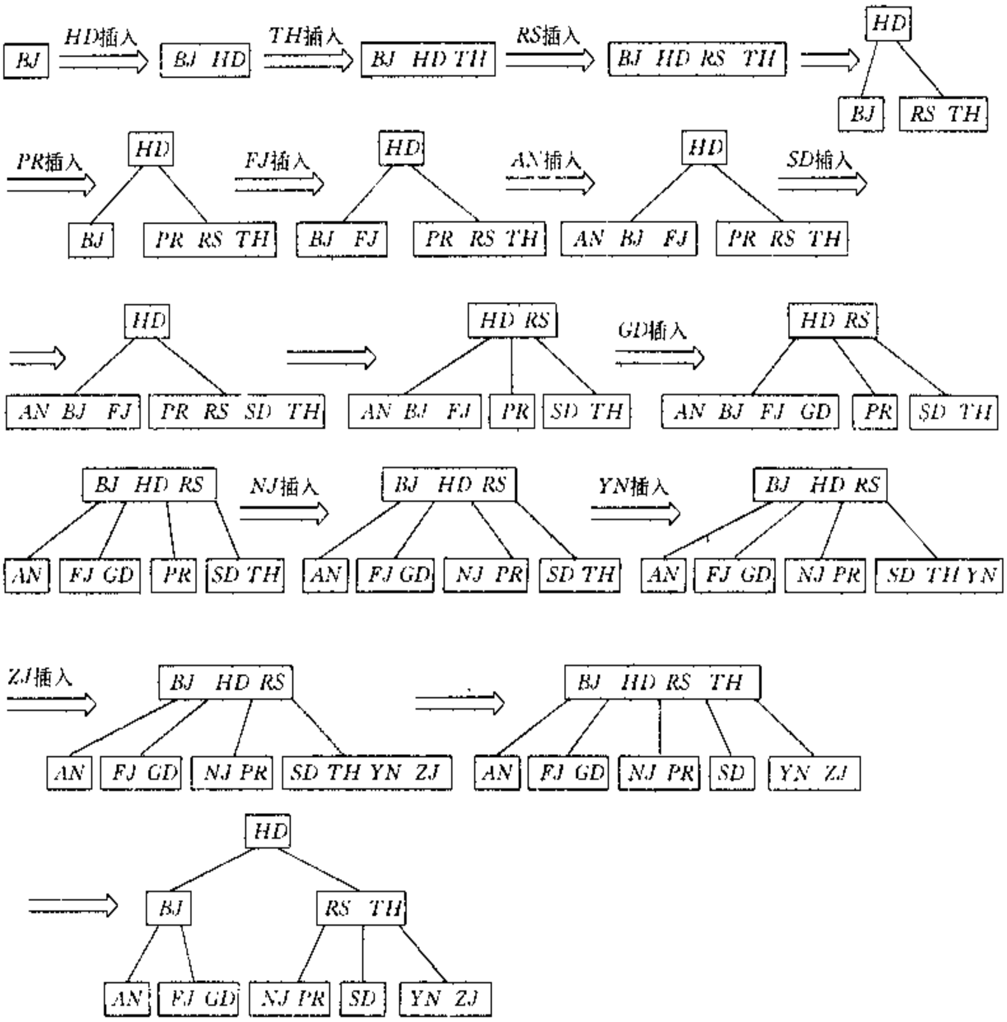


图 15.2.3

15.3 红 黑 树

一棵红黑树(RB 树)本身就是一棵二元搜索树,它的节点着以红色或黑色中的一种,所以每一节点必须附加一位用以存储颜色,并且满足下列性质:

1) 每一节点必须是红或黑的两者之一;

2) 所有叶节点是黑色的;

3) 若一节点是红色的,则它的"儿子"节点必是黑色的;

4) 每一节点到所有的叶节点的路径所经过的黑色节点的数目相同。

例如图 15.3.1 便是一棵红黑树,为方便起见,用"◯"表示黑色节点,"○"表示红色节点。

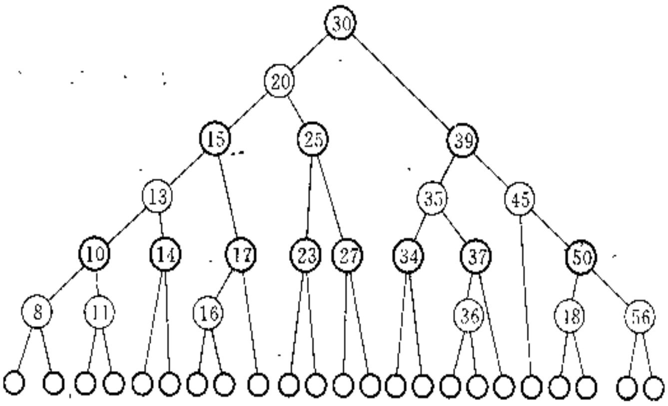


图 15.3.1

由于从每一节点到所有叶节点的路径上,它的后裔节点中黑色节点数相同,我们定义它为黑色高度 h_B。

从 RB 树的四个条件来看,从树根到叶节点的任何一条道路上黑色节点数相同,从而可知,最短的道路是所有的节点都是黑色。由性质 3)知最长的道路必是红黑相间,所以一条道路其长度最多是其它道路的两倍,因此 RB 树实际上是关于高度近似均衡的树。

一般来说,有 n 个节点的 RB 树的高度 h 满足:

$$h \leqslant 2\log_2(n+1)$$

这个不等式的证明留给读者思考。

和前面所讨论的均衡树一样,节点的插入到删除可能使其不再满足红黑树的条件,至于由于新的节点的插入或删除后如何调整指针和颜色,这里就不再讨论,留给读者自己研究,并作为作业。和高度均衡树类似,这些操作也都是程式化的。

习 题

1. 试将下列关键字依序插入到开始时为空的 2-3 树。

$F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E$。

2. 试将下列的关键字依序插入到开始时为空的 2-3-4 树。

$E, Z, D, Y, X, B, A, P, N, R, M, W, V, T, H, L, C, K, Q, S, F$。

3. 以 $TSINGHUAUNIVERSITY$ 为例,插入到初始为空的 2-3-4 树。

4. 以 $BEIJINGTSINGHUA$ 为例,说明插入到初始为空的 2-3 树。

5. 试证 n 个节点的 RB 树的高度 h,满足

$$h \leqslant 2\log_2(n+1)。$$

6. 试讨论从 2-3 树删除某一元素的算法。并举例说明。

7. 试讨论从 2-3-4 树删除某一元素的算法。并举例说明。

第16章 B-树

16.1 B-树 概 念

B-树是一种均衡的多路查找树，它在文件系统中很有用。首先介绍这种树的结构：

定义 B-树是一棵满足下列条件的根树：

(1) 每一节点 a 有以下的信息组：

① n_a 个关键字 $K_1^a, K_2^a, \cdots, K_{n_a}^a$ 满足：

$$K_1^a \leqslant K_2^a \leqslant \cdots \leqslant K_{n_a}^a$$

② 布尔量 l_a：

$$l_a = \begin{cases} 0, 若\ a\ 是内点 \\ 1, 叶节点 \end{cases}$$

(2) 若 a 是内点，它存在 n_a+1 个指针指向它的 n_a+1 个"儿子"子树，而叶节点不再有"儿子"；

(3) n_a 个关键字 K_i^a 将存贮在它的 n_a+1 个"儿子"子树中的关键字划分成区间：

$$K_1 \leqslant K_1^a \leqslant K_2 \leqslant K_2^a \leqslant \cdots \leqslant K_{n_a}^a \leqslant K_{(n_a+1)}$$

其中 K_i 是代表存贮在第 i 个"儿子"子树上的任意一个关键字；

(4) 每个叶节点到根节点的路径长度都相同；

(5) 每一节点有一关键字数目的上界和下界。

这些界可表示为关于固定整数 $t(\geqslant 2)$ 的关系如下：

① 除根节点外，每一节点至少有 $t-1$ 个关键字，每一内点至少有 t 个"儿子"。只要树非空，根节点也应至少有一个关键字；

② 每一节点最多有 $2t-1$ 个关键字，故一个内点可以有 $2t$ 个"儿子"节点。

从B-树的定义可以看出：它适用于磁盘或其它直接存取的外存设备。B-树最明显的特点在于它的每个节点可以有数目很多的"儿子"节点，故它更适合于磁盘的以页方式进行的存取。

特别当 $t=2$ 时，每一内点可能有2个，3个，或4个"儿子"，则类似于前面所讨论的2-3-4树。称 $t=2$ 为 B-树的最小级。但有一点不同的是。B-树的内点也可存放记录。

含有 n 个关键字的B-树，其中最高的是根节点可含有一个关键字，其余所有的内点都只有 $t-1$ 个关键字，此时高度为 h 的B-树含有的关键字数目至少是：

$$1 + 2(t-1)\sum_{i=1}^{h} t^{i-1} = 1 + 2(t-1)\frac{t^h - 1}{t-1} = 2t^h - 1$$

即关于 n 个关键字的B-树的高度为 h，则有

$$n \geqslant 2t^h - 1,$$

$$h \leqslant \log_t\left\lfloor \frac{n+1}{2} \right\rfloor。$$

由上可归纳成下面的定理。

定理 n 个关键字的 B-树的高度 h 有：

$$h \leqslant \log_t\left\lfloor \frac{n-1}{2} \right\rfloor$$

其中 $t(\geqslant 2)$ 是 B 树的最小级。

16.2 插入和删除

B 树的生成也是从空树起，逐个插入关键字而得。但由于 B-树节点中的关键字个数必须 $\geqslant t-1$，因此每次插入一个关键字不是在树中添加一个"叶子"节点，而是首先在最低层的某个非"叶子"节点中添加一个关键字，若该节点的关键字个数不超过 $2t-1$ 则插入完成，否则要产生节点的"分裂"。下面介绍一种 B-树的节点分裂算法。

当一个关键字插入到一个已装满 $2t-1$ 个关键字的节点(设为 a)时，"分裂"是一种最基本的运算。以中间元素 k_t 为界 $2t-1$ 个关键字分裂成左右两部分，每部分各 $t-1$ 个，将 k_t 提升至 a 的"父亲"节点中。如有必要再"分裂"其"父亲"节点。我们来看一个 $t=3$ 的例子(见图 16.2.1 和图 16.2.2)就完全清楚了。

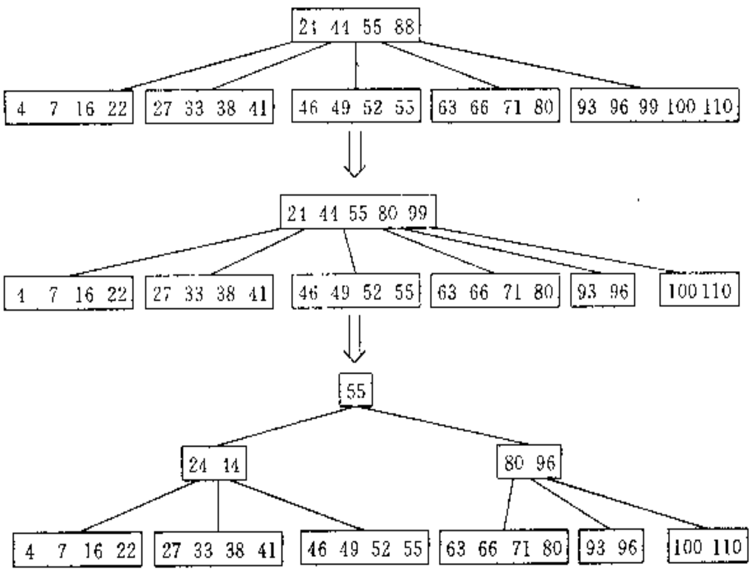


图 16.2.1

若需在 B-树上删除一个关键字，则首先应找到该关键字所在节点，并从中删除。若该节点为最下层的非叶节点且其中的关键字数目不少于 t，则删除完成，否则要进行"合并"节点的操作。删除的步骤和措施可具体描述如下：

(1) 若要删除的元素 d 是节点 a 所存储的元素，而且 a 是叶节点，则直接删去 d；

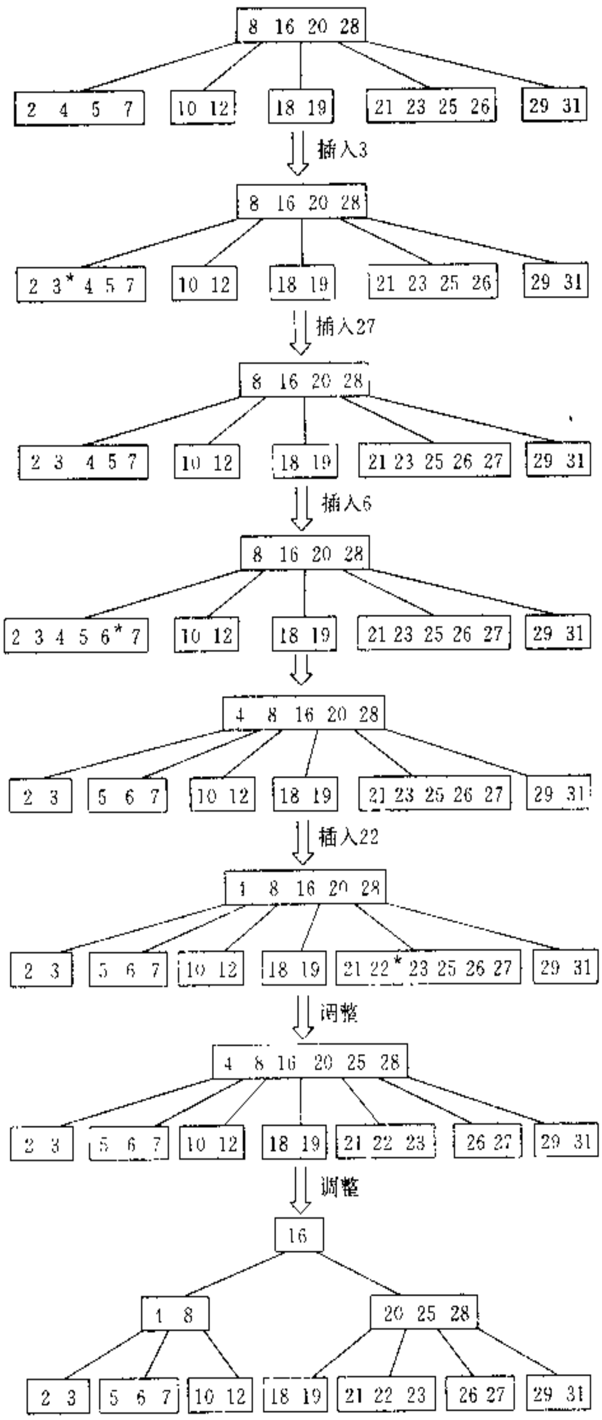


图 16.2.2

(2) 若 d 在结点 a，且 a 是内点，则：

① 若 a 有"儿子"b_1，至少有 t 个关键字在 d 的前面，则在 b_1 中找其中在 d 前面的最后一个，设为 d'，用 d' 取代 a 中的 d，并删去 b_1 中的 d'。

② 对称地若 a 有"儿子"b_2，至少有 t 个关键字在 d 的后面。则在 b_2 中找其中在 d 后

面的最前的一个，设为 d'，用 d' 取代 a 中的 d，并从 b_2 中删去 d'；

③ 如若 b_1 和 b_2 都只有 $t-1$ 个关键字，则将 d 和 b_2 中所有的关键字都归并到 b_1 中去，则 a 中失去 d 以及指向 b_2 的指针，b_1 含有 $2t-1$ 个关键字，取消 b_2，并从 b_1 中删去 d。

(3) 若 d 不出现在内点 a 中，确定包含有 d 的子树的根节点 r_i，若 r_i 只有 $t-1$ 个关键字，则按下述①或②处理。

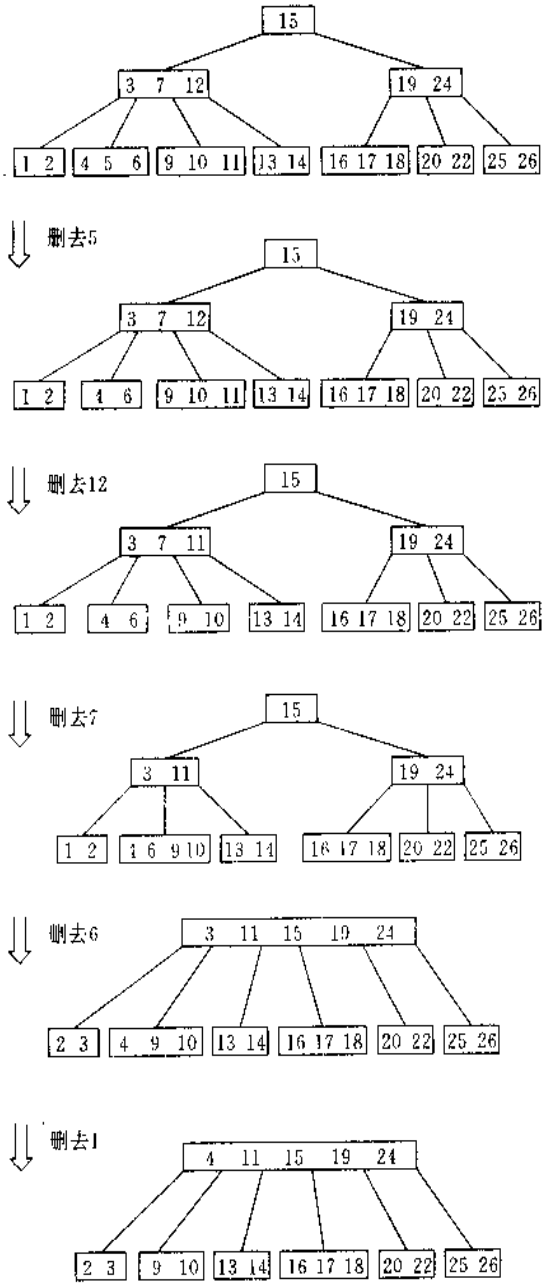


图 16.2.3

① r_i 只有 $t-1$ 个关键字，但它的兄弟有 t 个关键字。则从 a 取下一个到 r_i，从紧靠 r_i 的左或右兄弟中取一个到 a 中去，将兄弟的"儿子"中必须转到 r_i 的转到 r_i 来，；

② 若 r_i 和 r_i 的所有兄弟都只有 $t-1$ 个关键字，则将 r_i 和一兄弟归并，其中包含了一个从 a 取到下面来作为新的归并顶点的中间元素。

我们还是通过 $t=3$ 的实例(图 16.2.3)来说明这些删除准则：

习 题

1. 画出所有能表示{1,2,3,4,5}的最小度为 2 的 B-树。

2. 对于高度为 h 的 B 树，当最小度为 t 时，最多能够容纳多少关键字？

3. 将 $F,S,Q,K,C,L,H,T,V,N,M,R,N,P,A,B,X,Y,D,Z,E$ 插入一个空的 B-树中。只须画出节点须被分开前的情况和最后的情况。

4. 说明如何在 B-树中查找最小的关键字以及如何查找比给定关键字小的最大关键字。

5. 若将$\{1,2,\cdots,n\}$插入到最小度为 2 的 B-树中，最终 B-树将含有多少个节点。

6. B-树中级 $t=1$ 允许吗？为什么？

7. 用 $t=2$ 的 B-树表达序列 1,2,3,4,5,6,7,8,9

8. 设 $t=3$，以 *TSINGHUAUNIVERSITY* 为例说明插入到初始为空的 B-树。

9. 设 $t=4$，以 *BEIJING TSINGHUA* 为例说明插入到初始为空的 B-树。

第17章 哈 希 表

17.1 什么是哈希表

前面讨论的各种关键字存储，它们的地址是随机的，和关键字之间不存在确定的关系。在这样的结构中查找要进行一系列的比较，也就是说建立在比较的基础上。若采取根据关键字直接查找的办法，必然出现关键字可能出现的空间较实际上占用的要小得多。所以直接根据关键字存储势必要浪费大量的存储空间。

理想的情况是希望不经过任何比较，一次存取便能得到所要查找的对象，那就必须在存储位置和关键字之间建立一个确定的对应关系 h，使每个关键字和结构中一个唯一的存储位置相对应。因而在查找时，只要根据这个对应关系 h 找到给定值 k 的像 $h(k)$，若结构中存在关键字和 k 一致的记录，则必定在 $h(k)$ 的存储位置上。也就是说从文件的关键字 k，计算 $h(k)$ 便得它的存储位置。由此不需要比较便可直接取得所查记录。我们称这个对应关系 h 为哈希(Hash)函数。

由此所构造的表称为哈希表，又称散列或混列等。

17.2 哈希函数的构造方法

从上可看出，使用哈希表可能存在的问题是：可能存在 $k_1, k_2 \in K$，使得：

$$h(k_1) = h(k_2),$$

这就产生了冲突，故在研究哈希函数构造的同时，还必须研究解决冲突的方法。

什么样的哈希函数 h 才是比较好的呢？最好要使得每一个关键字均匀地分布在 m 个地址上，还要求计算简单。哈希函数 h 类似于随机数发生器。若关键字 k 出现的概率为 $p(k)$，则要求

$$\sum_{h(k)=j} p(k) = \frac{1}{m}, \quad j = 0,1,2,\cdots,m-1,$$

遗憾的是由于关键字序列的分布这个条件很难予以检测。

下面举例说明哈希函数的构造方法：

一般可将关键字 k 解释成某一自然数，若 k 是字符串，可通过 ASCII 码转化为自然数。

例 1

$$h(k) \equiv k \bmod m.$$

若 $m=13$，则 $k=14961$，有 $h(k) \equiv 11 \bmod m$。当然不能取 $m=2^l$。如若不然，则 $h(k)$ 正好是 k 的二进制表示中最低 l 比特(*bit*)。最好的情形是 $h(k)$ 的结果和 k 的各比特都有关系。

理想的 m 最好是素数，但又不太靠近 2 的幂，例如 $m=2^l-1$，则移位相同的两个字符串，其 Hash 函数的结果也相同。

例 2

$$h(k)=\lfloor m\cdot(kb \bmod 1)\rfloor \qquad 0<b<1$$

$kb \bmod 1$ 取的是 kb 的小数部分，即

$$kb-\lfloor kb\rfloor$$

其中 m 和 b 都是已知常数

若令 $b=(\sqrt{5}-1)/2=0.618033988$，$k=14960$，$m=15000$

$kb=9245.788467$，

$h(k)=\lfloor 15000\times 0.788467\rfloor=\lfloor 11827.005\rfloor=11827$

例 3 折叠法。通过例子介绍方法，比如 32 比特的符号串：

$k=10111100100100101000101010101100$

分成 6 比特一组。

$$\begin{array}{cccccc}
1 & 0 & 1 & 1 & 1 & 1\\
0 & 0 & 1 & 0 & 0 & 1\\
0 & 0 & 1 & 0 & 1 & 0\\
0 & 0 & 1 & 0 & 1 & 0\\
1 & 0 & 1 & 0 & 1 & 1\\
 & & & & 0 & 0\\
\hline
0 & 0 & 1 & 1 & 0 & 1
\end{array}$$

001101 是每位作不进位求和的结果。（当然也可以作进位加）这样得

$$h(k)=001101$$

例 4 平方取中法。

从 k^2 中提取 l 位。比较理想的是取中间 l 位。这是一种常用的构造哈希函数的方法。通常在选定哈希函数时，不一定能知道关键字的全部情况，取其中哪几位也不一定合适。而一个数平方后的中间几位数和数的每一位都相关，由此使随机分布的关键字得到的哈希地址也是随机的。

当然也可取前、后 l 位，但前面 l 位和 k 的前面若干位的关系比较密切，但和后面各位的关系则不密切。反之亦然。

17.3 解决冲突的方法

Hash 函数将关键字转换为地址，然而可能存在两个不同的关键字，但它们由 Hash 函数算出的地址是相同的。解决冲突的最简单的方法是当冲突发生时用某种方法寻找其他空地址，然后建立起地址链。例如 computerscience 通过 $h(k)\equiv k \bmod 17$ 的计算结果如表 17.3.1。

表 17.3.1

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
k_i	c	o	m	p	u	t	e	r	s	c	i	e	n	c	e
$h(k_i)$	3	15	13	16	4	3	5	1	2	3	9	5	14	3	5

上面是用 1 到 26 代表 A 到 Z 的 26 个字母。即 A 为 1，B 为 2，…，Z 为 26。假定开始时表的 0 到 16 的地址是空的，则依次插入得表 17.3.2。

表 17.3.2

地址	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		$r^{(8)}$	$s^{(9)}$	$c^{(1)}$	$u^{(5)}$	$t^{(6)}$				$i^{(11)}$				$m^{(3)}$	$n^{(13)}$	$o^{(2)}$	$p^{(4)}$
				$t^{(6)}$		$e^{(7)}$											
				$c^{(10)}$		$e^{(12)}$											
				$c^{(14)}$		$e^{(15)}$											

表 17.3.2 中字符右上方括号()的数字是插入的顺序。例如地址“5”所在列的 $t^{(6)}$，$e^{(7)}$等，分别表示表 17.3.1 中，$h(k_6)=h(k_7)=\cdots=5$，余此类推。

1. 自由地址法

解决冲突链地址法还可分为自由地址法和线性探测法，它们分别利用另外一组自由地址或者表中的空地址。

例如已知 Hash 函数，如下表 17.3.3。

表 17.3.3

keyword	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9
$h(k_i)$	7	5	8	5	9	8	6	0	9	7

关键字 k_0 按 $h(k_0)=7$ 存入地址号为 7 的单元，同样 $h(k_1)=5$，从而 k_1 存入地址为 5 的单元，k_2 存入 $h(k_2)=8$ 的单元。由于 $h(k_3)=5$ 且地址为 1 的单元被关键字 k_1 所占有，但自由存储单元 10 空着，故建立地址链，即在地址为 1 的后面建立一指向自由存储地址 10 的指针，并将 k_3 存入自由存储地址 10 的单元。同样的理由，k_4 存入地址 $h(k_4)=9$ 的单元，k_5 存入自由存储地址为 11 的单元，在单元 8 建立一指针，指向自由存储 11 单元，其他类推。可形象地用图 17.3.1 表示。

2. 线性探测法

若 $h(k_i)$地址发生了冲突，则探测表中下一单元，若该单元已占用，则继续依次探测，直到发现空单元为止。其过程可描述如下：

(1) $j \leftarrow h(k)$，$f \leftarrow 1$。

(2) 若 $a(j)$已被占领，则转(3)。

否则,[$a(j) \leftarrow k$,结束]。

(3) $j \leftarrow j+1 \bmod n$,$f \leftarrow f+1$。

若 $f > m$,则作[打印单元已满,结束]。否则,转(2)。

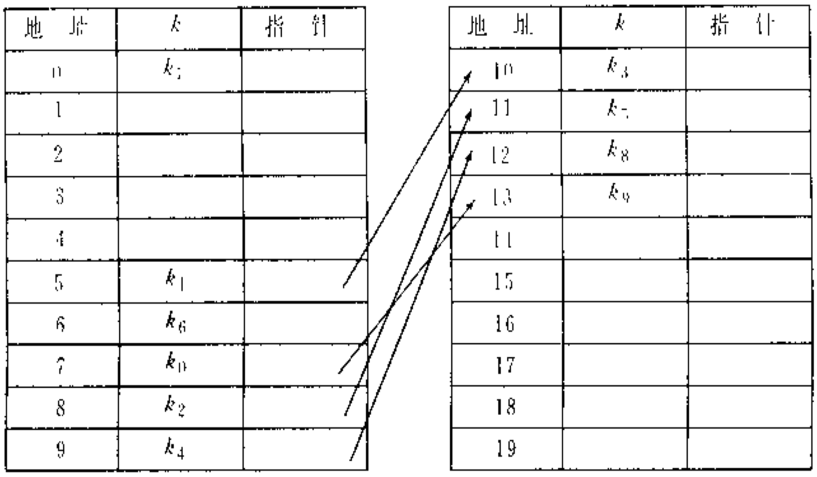


图 17.3.1

以前面表 17.3.1 的例子为例,假定开始表为空,即从 0 到 16 的地址都没有被占领。

表 17.3.4

i \ 地址	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1				$t^{(1)}$													
2																$o^{(2)}$	
3														$m^{(3)}$			
4																	$p^{(4)}$
5					$n^{(5)}$												
6						$t^{(6)}$											
7							$e^{(7)}$										
8		$r^{(8)}$															
9			$s^{(9)}$														
10				*	*	*	*	$c^{(10)}$									
11										$i^{(11)}$							
12						*	*	*	$e^{(12)}$								
13															$n^{(13)}$		
14				*	*	*	*	*	*	*	$c^{(14)}$						
15							*	*	*	*	*	$e^{(15)}$					

表 17.3.4 中字符右上角()里的数是顺序例如第一行的 $c^{(1)}$ 表示第一个字符是 c。$\times$ 表示冲突发生的状态，例如 $i=10$ 时 $c^{(10)}$ 左边有四个 $\times$，原来 $h(c)=3$，但地址 3,4,5,6 都分别被 c,u,t,e 所存放，故 $c^{(10)}$ 存放在地址 7。其他同理类推。

17.4 哈希算法的分析(线性探测法分析)

假定 m 个单元中已有 r 个元素，利用线性探测法究竟要作多少次探测呢？以表 17.3.4中的第 7 行第 5 个元素 d 为例，$h(k_7)=5$，但第 5 个元素被 d 占用，第 6 个单元也已被 e 所占用，第 7 单元空着，故探测次数为 3。同样第 11 行，$h(k_{11})=h(s)=2$，但从第 2 到第 7 单元都已被占用，第 8 单元空着，故探测次数为 7。总而言之，k 的 Hash 地址 $h(k)$ 已被占用，则需要探测次数为 $n_k=l_k+1$。l_k 是从 $h(k)$ 开始的链表长度。

从上面的简单分析可知在最坏情况下线性探测算法的效果是不好的，但我们感兴趣的是它的"平均特性"，下面先讨论哈希表规模为 m，已有 n 个项目的关键字地址记录在 n 个单元上，求第 $n+1$ 个项的关键字地址插入时作 r 次探测的概率 p_r。

首先作如下假定：第一个假定关键字存入地址是随机的；第二个假定哈希函数 h 是足够均匀的。

将 m 个单元对应一个 m 位的二进制数，即 m 位的 0,1 符号串，0 表示该单元空着，1 表示该单元被占用。m 个单元有 n 个单元被占用对应于有 n 个 1 的 m 位 0,1 符号串，这样的图像有 $\binom{m}{n}$ 个。第 $n+1$ 个关键字作 r 次探测后插入到哈希表，对应于 $r-1$ 个 1 组成的子串的图象，即：

$$\cdots 0\underbrace{1\ 1\cdots 1}_{r-1}0\cdots$$

设这样的数目为 s，则

$$p_r=s\Big/\binom{m}{n}$$

其中 $\binom{m}{n}$ 为有 n 个 1 的长为 m 的 0,1 符号串数目。同样的理由，其中有长度为 l 的 1 的子串的数目为：

$$\binom{m-l+1}{n-l-1}$$

即将 $\underbrace{1\ 1\cdots 1}_{n}$ 作为一个单元参加考虑。为了避免 $\underbrace{1\ 1\cdots 1}_{l}$ 和其他的 1 组成长度大于 l 的 1 的子串，实际上将 $\underbrace{1\ 1\cdots 1}_{l+1}0$ 作为一个单元参加组合，所以

$$p_r=\binom{m-r}{n-r+1}\Big/\binom{m}{n}$$

于是均匀哈希算法所需探测数的平均值为

$$\overline{S}_n=\sum_{r=1}^{n+1}rp_r=(m+1)\sum_{r=1}^{n+1}p_r-\sum_{r=1}^{n+1}(m-r+1)p_r=$$

$$m+1-\sum_{r=1}^{n}(m-1-r)\cdot\binom{m-r}{n-r-1}\Big/\binom{m}{n}$$

$$m+1-\sum_{r=1}^{n}(m-1-r)\cdot\frac{(m-r)!}{(n-r-1)!(m-n+1)!}\Big/\binom{m}{n}=$$

$$m+1-\sum_{r=1}^{n}(m-n)\cdot\frac{(m-r+1)!}{(n-r+1)!(m-n)!}\Big/\binom{m}{n}=$$

$$m+1-\sum_{r=1}^{n}(m-n)\binom{m-r-1}{n-r-1}\Big/\binom{m}{n}=$$

$$m+1-(m-n)\sum_{r=1}^{n}\binom{m-r+1}{n-r+1}\Big/\binom{m}{n}$$

而考：

$$\sum_{r=1}^{n}\binom{m-r+1}{n-r+1}=\sum_{r=1}^{n}\binom{m-r+1}{m-n}=$$

$$\binom{m}{m-n}+\binom{m-1}{m-n}+\cdots+\binom{m-n}{m-n}=$$

$$\frac{m!}{(m-n)!n!}+\frac{(m-1)!}{(m-n)!(n-1)!}+\frac{(m-2)!}{(m-n)!(n-2)!}+\cdots+1=$$

$$\frac{m!}{(m-n)!n!}\left[1+\frac{n}{m}+\frac{n(n-1)}{m(m-1)}+\cdots+\frac{n!(m-n)!}{m!}\right]$$

因为 $\frac{n-k}{m-k}<\frac{n}{m}$，$k>0$

所以：

$$\sum_{r=1}^{n}\binom{m-r+1}{n-r+1}<\frac{m!}{(m-n)!n!}\left[1+\frac{n}{m}+\frac{n^2}{m^2}+\cdots+\frac{n^n}{m^n}\right]$$

$$S_n<\left(m+1-(m-n)\frac{1-\alpha^{n+1}}{1-\alpha}\binom{m}{n}\right)\Big/\binom{m}{n}=$$

$$m+1-m(1-\alpha)\frac{1-\alpha^{n+1}}{1-\alpha}=$$

$$m+1-m(1-\alpha^{n+1})=$$

$$1+m\alpha^{n+1}$$

其中 $\alpha=\frac{n}{m}$

17.5 二重哈希法

二重哈希法也是解决冲突的一种方法。所谓二重哈希法是已知两个哈希函数 h_1 和 h_2，首先从 $h_1(k)$ 找开址，如若遇到冲突，则找

$$h_1(k)+h_2(k) \bmod m$$

如若依然冲突，则继续找

$$h_1(k)+2h_2(k) \bmod m$$

直到找到地址为止。要求 $h_2(k)$ 为 1 到 $m-1$ 间与 m 互素的数，见表 17.5.1。

表 17.5.1

关键字	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9
$h_1(k)$	4	6	5	0	6	3	3	5	1	2
$h_2(k)$	5	7	1	7	3	1	7	1	7	7

若关键字依顺序插入，利用二重哈希法可得表 17.5.2。

从表中看出 k_0,k_1,k_2,k_3 都只需一次探测便找到插入地址。然而 $h_1(k_4)=6$，地址 6 已被 k_1 占用，但

$$h_1(k_4)-h_2(k_4)=6-3=3 \mod 10$$

表 17.5.2

地址＼顺序	1	2	3	4	5	6	7	8	9	10
0				k_3			*		*	
1								k_7	*	
2						k_5		*		*
3					k_4	*	*	*	*	
4	k_0					*		*	*	
5			k_2			*		*		*
6		k_1			*				*	
7							k_6		*	
8										k_9
9									k_8	
探测次数	1	1	1	1	2	[illegible]	3	3	7	3

故 k_4 便占用地址 3，探测次数为 2。

经验说明二重哈希法使得分布比较均匀。

表中 * 是表示发生冲突的标志。

习 题

1. 证明探测序列 $h,h+c,h+2c,\cdots,h+(b-1)c$ 访问表中所有的位置当且仅当 $(b,c)=1$。

2. 将 5,28,19,15,20,33,12,17,10 插入一个哈希表中，$h(k)=k \mod 9$，采用链接的

方法解决冲突。

3. 对于采用链接方法的散列表，不论插入时放在最先还是最后，证明查找成功的期望值都是相同的。

4. $h(k)=\lfloor m(kA \bmod 1)\rfloor$，$m=1000$，$A=(\sqrt{5}-1)/2$，求 $k=61,62,63,64$ 的地址。

5. 已知 $h_1(k)=k \bmod m$，$h_2(k)=1+(k \bmod (m-1))$，$m=11$，求 $k=10,22,31,4,15,28,17,88,59$ 的二重散列地址。

第 18 章　DFS 算法和 BFS 算法

18.1　概　　述

在求解有限离散对象的数学问题时，我们最容易想到的是穷举法。也就是说将该问题的状态空间中的每一状态穷举出来，并依次对其作出判断，从而求出满足条件的解。例如：

1. 棋盘布局问题，也就是所谓的"皇后"问题。

将 n 个棋子布到 $n\times n$ 的棋盘上，每格只能放一个，要求任了两两不在同一行或同一列，也不在同一对角线上。以 $n=4$ 为例，假定第一行的棋子在 i 列，第二、三、四行的棋子分布在 j、k、l 列，所以一个布局可用 $ijkl$ 来表示。由于两两棋子不在同一行或同一列，$ijkl$ 等价于 1，2，3，4 的某一个排列。如图 18.1.1 所示的布局对应于排列 3142，总状态数（排列数）是 $4!=24$，它们是：

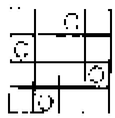


图 18.1.1

1234　1243　1324　1342　1423　1432

2134　2143　2314　2341　2413　2431

3124　3142　3214　3241　3412　3421

4123　4132　4213　4231　4312　4321

但并非每一排列都是问题的解，因为还要求任两个棋子不在同一条斜线上，还需要对以上 24 种状态一一进行检验，看是否满足条件的布局。因而一般说棋子布局问题的状态空间有 $n!$ 种状态，若对一般 $n\times n$ 的棋盘的布局问题，n 较大时穷举法将是不现实的。

2. 流动推销员问题（TSM 问题）或称旅行商问题

TSM 问题是这样的：某一推销员要到 n 个城市去推销产品，从某一城市出发，经过每一城市一次且仅一次，最后返回原出发地，要求所走的路线最短（或旅费最省）。

TSM 问题可以抽象成这样的问题：n 个城市用 n 个点表示，若两城市间有航线相连，则相应的两顶点之间用边相连，边上的权表示两城市间的距离（或旅费），以有向图 $G=(V,E)$，及距离矩阵 $D=(d_{ij})$，其中 $n=|V|$，$d_{ij}=\infty$ 表示 $(v_i,v_j)\notin E$（即 i，j 之间不存在直接的通路）。问题变成寻找从某一点 v 出发遍历各顶点一次且仅一次最后返回点 v 的最短回路。我们先假设任意两城市间的来回路程一样，则 $d_{ij}=d_{ji}$，$i,j=1,2,\cdots,n$，也就是 D 是对称方阵，则可能的不同方案数为 $\frac{1}{2}(n-1)!$。因回路 $i_1\to i_2\to\cdots\to i_n\to i_1$ 和 $i_1\to i_n\to\cdots\to i_2\to i_1$ 是相同的，否则不同的方案数为 $(n-1)!$

上面这两个问题的状态数分别为 $\frac{1}{2}(n-1)!$ 与 $(n-1)!$。由 Stirling 公式

$$n!\approx\sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

可知当 n 较小的时候，穷举法可在能接受的时间内进行，但当 n 不很大如 $n=20$ 时，

总有

$$26! \approx 4 \times 10^{26}$$

若每年以 365 天计，每年有 $365\times24\times3600=3.1536\times10^{7}$ 秒，若以每秒能搜索 10^{7} 个方案的超高速计算机来处理，需要 $\frac{4}{3.1536}\times10^{12}$ 年，这是事实上很难接受的时间。

综上所述在求解有限离散对象的数学问题的解时，只要问题的规模不十分大，穷举法不失是一种简单易行的方法，但当规模较大时，必须寻求效率较高的方法。

18.2 DFS 算 法

DFS 是英文"Depth First Search"的缩写，即深度优先搜索或纵深优先搜索的意思。

以上节的棋子布局问题来讨论 DFS 算法。以 $n=4$ 为例，可以发现没有必要逐一穷举每一状态，如上所示用 (h,i,j,k) 表示一种布局方案，即第一行的棋子在第 h 列，第二行棋子在第 i 列，第三行棋子在第 j 列，第四行棋子在第 k 列的一种布局。当 $h=1,i=2$ 是不允许的，故可一下子排除 $(1,2,j,k)$ 的所有状态。同样 $h=1,i=3$ 时未出现任何两个棋子在同列或同对角线的情况，故可进而考虑 $(1,3,j,k)$ 的各种可能状态。搜索过程可形象地表示如图 18.2.1。

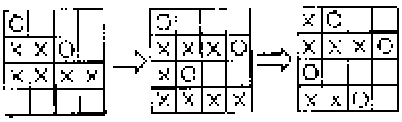


图 18.2.1

图中"○"表示棋子，"×"表示不符合布局条件的格子。

这种一旦发现前面已是此路不通，立即回头改换路径，而不是一条道走到底的策略，基本上还是一种穷举法，只不过设法减小穷举的搜索空间罢了。这就是所谓的 DFS 算法的基本思想。对 $n=4$ 的棋子布局问题，这种思想可形象地表示如图 18.2.2，这是所有可能的 24 种状态树，也就是问题的状态空间。同样的道理 $n\times n$ 棋盘问题的状态空间是具有 $n!$ 个叶子的 n 个元素的排列树。

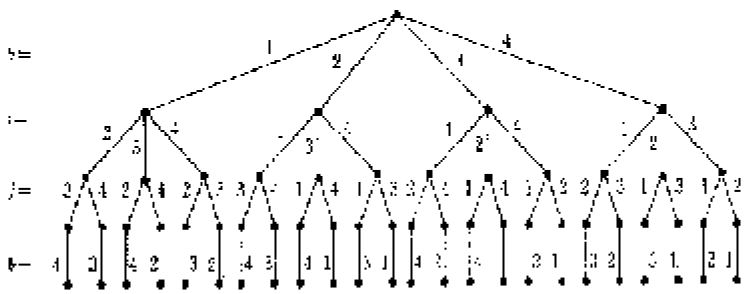


图 18.2.2

每一个叶子节点表示一个状态，而从根到叶子的路径则给出了一种排列方法。其对应于图18.2.2搜索过程可形象地在图18.2.3中体现出"向前走，碰壁回头"的思想。

图 18.2.3

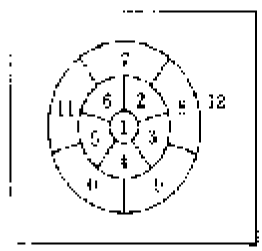


图 18.2.4

图18.2.3中的虚线边表示已碰壁，必须回退一步，准备重新向前搜索。图18.2.3实际是图18.2.2的一部分，表明它的便是在搜索过程中去掉的或称"被剪去的"部分。如果有一种算法，每次都剪去越多的树枝，那么该算法的效率一定是越高的。

下面是又一个应用DFS算法的问题：

试用4种颜色对如图18.2.4所示的12个区域进行着色，要求相邻的区域着不同的颜色。

设4种颜色分别用1,2,3,4表示，不失一般性，先假定域1,2,3分别着以颜色1,2,3，然后对域1,2,3以外的各域依序进行着色尝试，即以1色开始，不满足条件换以2色，依次类推，直到找到一种可行的着色方案，或试完为止。设矩阵 $A=(a_{ij})_{12\times12}$ 表示12个区域的邻接关系：

$$a_{ij}=\begin{cases}1, & 若域\ i\ 和域\ j\ 相邻\\ 0, & 其他\end{cases}$$

用 $c(i)$ 表示第 i 个域所着的颜色 $i=1,2,\cdots,12$，则该问题的算法如下描述：

(1) 初始化操作

$$c(1)\leftarrow 1, c(2)\leftarrow 2, c(3)\leftarrow 3$$

$$c(i)\leftarrow 1, 4\leqslant i\leqslant 12; \quad i\leftarrow 4$$

(2) $j\leftarrow c(i)$，若 $j<5$，则转(3)。

否则，作【$c(i)\leftarrow 1, i\leftarrow i-1$，若 $i<4$，则转(6)，否则，【$c(i)\leftarrow c(i)+1$，转(2)】】。

(3) k 从1到 $i-1$ 作

【若 $c(k)*a(i,k)=j$，则作【$c(i)\leftarrow c(i)+1$，转(2)】】。

(4) $i\leftarrow i+1$，若 $i<13$，则转(2)。

(5) 输出数组 c：

$$c(12)\leftarrow c(12)+1。$$

(6) 结束。

其中(3)步是对第 i 行域的第 j 种颜色是否符合要求进行判断，$c(k)*a(i,k)=j$ 是染色冲突，则换下一颜色。第(4)步是前进措施，第(2)步有后退措施。

DFS算法的关键在于处理向前走，碰壁判断及后退的策略。本例中当对某一区域四

[illegible]

18.3 无向图的 DFS 算法

设图 $G=(V,E)$ 是简单的无向图，要求遍历图 G 中所有的顶点及所有边。下面我们介绍无向图 G 的深度优先搜索遍历算法。简单图即无自环及平行边的图。假定图的存储结构中所有顶点及边未曾被访问过。DFS 遍历算法可以从图 G 中某个顶点 v 出发，访问此顶点，然后通过访问 v 的一条未被访问的关联边到达其相邻接点，再从这邻接点出发继续进行。若邻接点均已被访问，则后退，依此类推，直至返回到出发点 v，使图中所有和 v 有路径相通的顶点都已被访问为止。有时还可能从出发点 v 出发，图中尚有顶点未被访问，这时它们是非连通的，则另选图中一个未曾被访问的顶点作为起点，重复上述过程，直至图中所有顶点都被访问到为止。访问各边也可在此过程中完成。

从上面可知，不妨假设图 G 是连通的，否则只需分别遍历其各个连通分支即可。

在遍历过程中，设 v 点刚刚被访问过，现在 v 的邻接点中任选 w 点，且 w 点未被访问过，则经边 (v,w)，并规定边方向是从 v 到 w。这时称 v 是 w 的"父亲"节点，记作 $FATHER(w)=v$，称 w 为 v 的"儿子"。此时称边 (v,w) 被访问过了。

一般情况下，当访问某点 v 时，有两种可能：

(1) 如 v 的所有关联边都被访问过了，则 v 回到其父节点 $FATHER(v)$，从 $FATHER(v)$ 再继续搜索，此时称对 v 点的扫描完成了。

(2) 如果存在 v 的尚未被访问过的关联边，则任选其中一边，比如 (v,w)，它的方向为从 v 到 w，此时称边 (v,w) 现在被访问过。访问边 (v,w) 有两种情况：

(1) 如 w 点从未被访问过，则顺着边 (v,w) 访问 w，下一步从 w 开始继续搜索，此时将边 (v,w) 放入已访问边集合 T 中，且 $v=FATHER(w)$，图中用实线画出。

(2) 如 w 点已被访问过，则将 (v,w) 放入已访问边的另一边集合，并非当 v 的尚未被访问过的关联边，而原本是由 w 出发访问，此时边 (v,w) 称为后向边，意即由 v 回到已被访问过的 w 点，图中用虚线画出。

此时边 (v,w) 的方向已明确，(v,w) 就访问完毕。在 DFS 遍历期间，对顶点 v 按先后访问的次序安排了不同的序整数 $DFN(v)$，如 $DFN(v)=i$，则 v 是第 i 个被访问的顶点，$DFN(v)$ 称为 v 的深度优先搜索号数。当遍历所有顶点并返回到出发点搜索停止时，所有点及边都被访问过（假设是连通图）。遍历结果将图 G 的边分成两类：边集合 T 中的实线边及后向边（虚线）。边集合 T 中的所有有向实线边构成图 G 的一棵有向树，称为是 G 的 DFS 树。

先看一个例子。图 18.3.1 中 (a) 是所给的图 G；(b) 是 DFS 遍历的结果，顶点旁()的数字是深度优先搜索序号，其遍历过程可用图 18.3.2 描述。为了方便起见以()中的序号代表该点，比如 $v_1(1)$，在图 18.3.2 中记为 1。

下面将给出无向图的 DFS 遍历算法。在给出算法以前先给出几个记号，对每个点引入标记 $MARK(v)$，其值或为 0 或为 1，可以表示该点是否曾被访问过，开始时将点

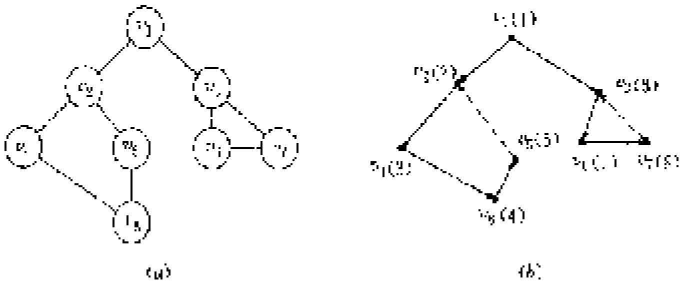


图 18.3.1

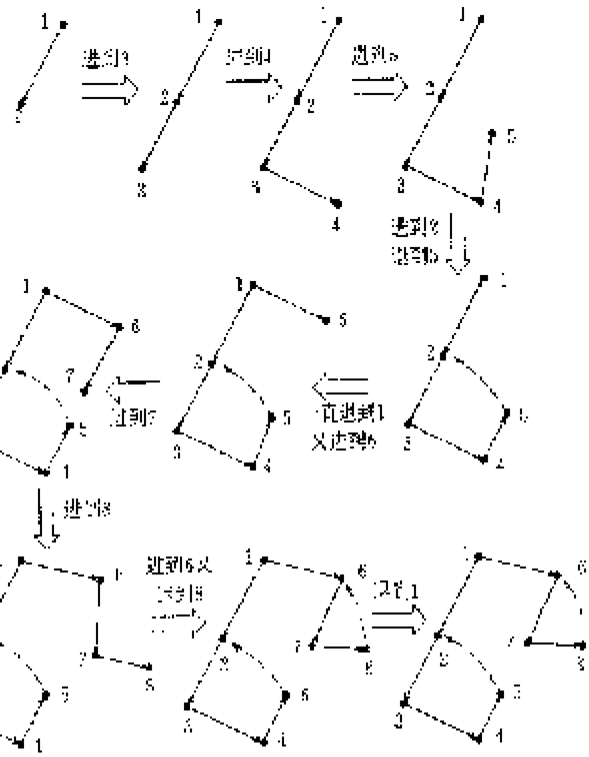


图 18.3.2

$MARK(v)=0$，表示它们未被访问过。无论何时，它一旦被访问，其 $MARK$ 值就改为 1。图 G 也不必要求是连通图。以后不特别说明都假定图 G 是简单图。

算法：无向图的 DFS 遍历算法

(1) 置 $T\leftarrow\varnothing$，$B\leftarrow\varnothing$，对每点 v，置 $FATHER(v)=0$，$MARK(v)=0$，$i\leftarrow1$

(2) (从任何一个图 G 的分支开始)，任选一个 $MARK$ 值为 0 的点，比如 r，有 $MARK(r)=0$，作[$DFN(r)\leftarrow i$，$MARK(r)\leftarrow1$，$v\leftarrow r$，(此 r 即为该分支的生成树的根)]。

(3) 如 v 的全部关联边已被标上"访问过"，则转(5)，此时 v 已完全扫描过。否则任选

未被访问的边(v,w)转(4)。

(4) 对于此边定向为从 v 到 w，标志成已访问的，实施下列步骤后转(3)。

① 如 $MARK(w)=0$ 作：

【$i\leftarrow i+1$，$DFN(w)\leftarrow i$，$T\leftarrow T\cup\{(v,w)\}$，

$MARK(w)\leftarrow 1$，$FATHER(w)\leftarrow v$，$v\leftarrow w$】。

② 如 $MARK(w)=1$，作 $B\leftarrow B\cup\{(v,w)\}$。

(5) 如 $FATHER(v)\neq 0$（即 v 不是该分支的生成树的根），则作【$v\leftarrow FATHER(v)$，转(3)】；否则，转(6)。

(6) 如对每个点 v，$MARK(v)=1$，则转(7)；否则，作【$i\leftarrow i+1$，且转(2)】。

(7) 停止，深度优先搜索遍历完毕。

算法中集合 T 给出了图 G 的 DFS 森林的所有树枝，而 B 则给出了后向边的全体。

在图 G 的 DFS 树中，若从 v 点出发，沿 DFS 树枝可以到达 w 点，则有 $DFN(v)<DFN(w)$，v 点称为 w 点的"祖先"，w 称为 v 的"后裔"。若 $(v,w)\in T$，则 v 是 w 的"父亲"节点，w 称为 v 的"儿子"节点。从 v 点不能沿 DFS 树枝到达 w 点，从 w 点也不能到达 v 点，这样的点 v 和 w 是可能存在的，这一点是连通图的性质所决定的。

DFS 法的时间复杂性为 $O(\max(m,n))$，其中 $n=|V|$，$m=|E|$。

18.4 有向图的 DFS 算法

有向图的深度优先搜索本质上与无向图相似，区别在于搜索只能顺着有向边的方向推进。这样有向图 $G=(V,E)$ 中的有向边根据被访问时的情况可以分成 4 种。一条未被访问的有向边 (v,w) 可按如下 4 种情况分类成：

情况 1：w 是未被访问过的点，则 (v,w) 归入图 G 的 DFS 森林的树枝 $TREE$。

情况 2：w 已被访问过：

a：在已形成的 DFS 森林中 w 是 v 的后裔节点，则 (v,w) 称为属于前向边集合 $FORWARD$；

b：在已形成的 DFS 森林中 w 是 v 的祖先节点，则 (v,w) 称为属于后向边集合 $BACK$。

c：在已形成的 DFS 森林中 v 和 w 不存在祖先与后裔的关系，并且有 $DFN(w)<DFN(v)$，则称 (v,w) 是属于横跨边集合 $CROSS$。注意无论如何不存在 $DFN(w)>DFN(v)$ 的这种类型横跨边 (v,w)。

从上可容易看出，一条边 (v,w)，如 $DFN(w)>DFN(v)$，则或为树枝边，或为前向边。细分一下，树枝边总是导向搜索一个新的（未访问）点；一条边 (v,w)，如 $DFN(w)<DFN(v)$，或叫做后向边（w 未访问完毕），或叫横跨边（w 已访问完）。显然在有向图中，不存在这样的横跨边 (v,w) 使得 $DFN(w)>DFN(v)$。

以上四种情况可形象地用图 18.4.1 表示。

顺便指出，有向图 G 的 DFS 森林可以看成由树枝边组成的原来 G 图的子图，可能会有这样的情况：图 G 去掉边的方向后是连通的，但有向图的 DFS 森林却不连通。

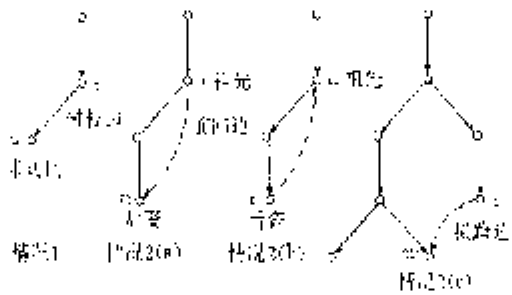


图 18.2.1

为了叙述有向图的 DFS 遍历算法，引进数组 $SCAN$，记录边的访问，对每点 v 设 $SCAN(v)=0$ 表示从 v 出发的有向边的访问还没有结束，若在它的后裔节点中，$SCAN(v)=1$ 表示该点的访问已完。这样满足 $DFN(v)<DFN(w)$ 的边 (v,w)，当 $SCAN(w)=0$ 时是后向边，当 $SCAN(w)=1$ 时则是横跨边。算法中用 $FOREST$、$FORWARD$、$BACK$、$CROSS$ 分别是树枝边、前向边、后向边、横跨边的集合。

算法：有向图的 DFS 遍历算法。

(1) G 是不含自环的有向图，令 $TREE\leftarrow\varnothing$，$FORWARD\leftarrow\varnothing$，$BACK\leftarrow\varnothing$，$CROSS\leftarrow\varnothing$，$i\leftarrow1$，对每个点 v 作【$MARK(v)\leftarrow0$，$FATHER(v)\leftarrow0$，$SCAN(v)\leftarrow0$】。

(2) 从一个新的根点开始 DFS 遍历：任选一个 $MARK$ 值为 0 的点，设为 r，$MARK(r)\leftarrow0$，令 $DFN(r)\leftarrow i$，$MARK(r)\leftarrow1$，$v\leftarrow r$。

(3) 如果所有以 v 为起点的有向边都被访问过，则置 $SCAN(v)\leftarrow1$，再转(5)；否则选取 v 点的任一条未被访问过的有向边 (v,w)，转(4)。

(4) 令 (v,w) 已被访问：

① 如 $MARK(w)=0$，则令【$i\leftarrow i+1$，$DFN(w)\leftarrow i$，$TREE\leftarrow TREE\cup\{(v,w)\}$，$MARK(w)\leftarrow1$，$FATHER(w)\leftarrow v$，$v\leftarrow w$，转(3)】。

② 如 $MARK(w)=1$，则令【当 $DFN(w)>DFN(v)$ 时，$FORWARD\leftarrow FORWARD\cup\{(v,w)\}$；当 $DFN(w)<DFN(v)$ 且 $SCAN(w)=0$ 时，$BACK\leftarrow BACK\cup\{(v,w)\}$；当 $DFN(w)<DFN(v)$ 且 $SCAN(w)=1$ 时，$CROSS\leftarrow CROSS\cup\{(v,w)\}$；转(3)】。

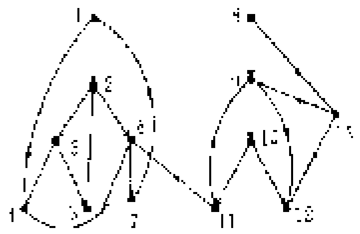
(5) 如 $FATHER(v)\neq0$，则 v 不是根，令【$v\leftarrow FATHER(v)$，转(3)】，否则令【$FATHER(v)\leftarrow0$，转(6)】。

(6) 如每一点都有 $MARK(v)=1$，则转(7)；否则，令【$i\leftarrow i+1$，转(2)】。

(7) 停止，遍历完毕。

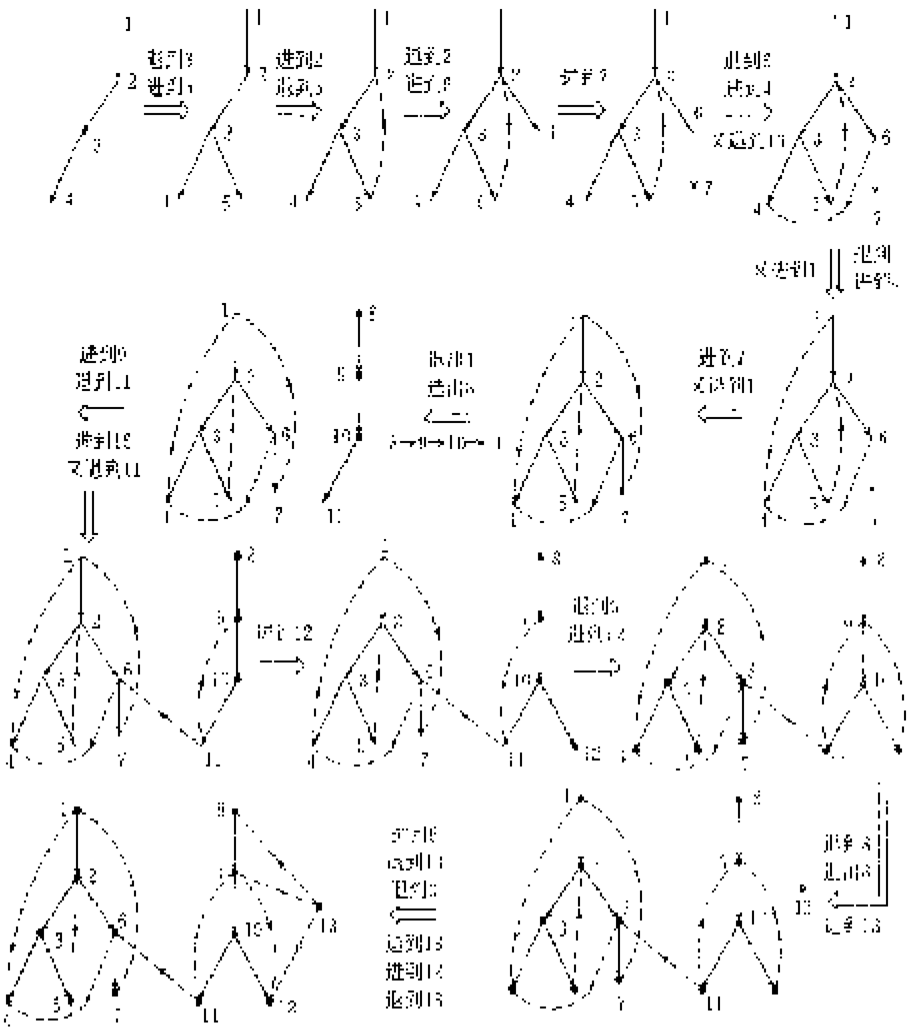
下面看一个例子(参 8.1.2)，图 18.2.2 的(a)是已知有向图，(b)是搜索的记录，(c)是过程，(d)是 DFS 森林，其 DFS 森林由 3 棵树组成。

同样有向图的 DFS 遍历算法的时间复杂性为 $O(\max(|V|,|E|))$。

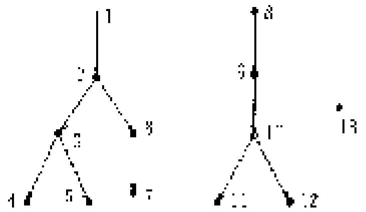


(a)

(b)



(c)



(d)

图 18.1.2

18.5 互连通块问题

设图 $G=(V,E)$ 是一无向连通图，若存在一点 $v\in V$，将 v 点从 G 图中消去，则 $G\setminus v$ 便分成互不连通的块，就称 v 是 G 的割切点。这里消去 v 点，即从图 G 中消去顶点 v 和与 v 关联的所有边。更一般地讲，若 $G\setminus v$ 比 G 的连通块多时就称 v 点为割切点，如图 18.5.1 中的顶点 v 便是割切点。不含割切点的图称为是互连的，连通最大的互连通子图称为是该图的互连通块。由于 DFS 遍历算法可求得图的互连通块，下面讨论之。

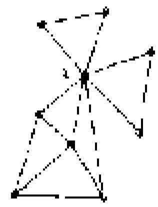


图 18.5.1

假定 $G=(V,E)$ 是连通的无向图，r 是其 DFS 树的根节点，则 r 点正好是割切点的充要条件是它有多于一个的“儿子”节点。这道理是显然的，其证明留给读者完成。

若割切点不是 DFS 树的根，将会出现什么现象呢？显然 v 点必存在某个“儿子”节点（记为 S），S 点的所有后裔节点均无后退边与 v 的“祖先”节点相连接，反之亦然。这些结论都是十分直观，证明也留给读者思考。

为此引进

$$L(v)=\min_{w\in R(v)}\{DFN(w)\}$$

其中 $R(v)$ 是从 v 点出发在 DFS 树，最多在后面有一条后退边所能到达的顶点的集合；$L(v)$ 是 $R(v)$ 中顶点的深度优先数 DFN 的最小值，鉴于这个道理，$L(v)$ 的计算是：

(1) 当首次访问 v 点时，令 $L(v)=DFN(v)$

(2) 当在 v 点出现后退边 (v,w) 时，令

$$L(v)=\min\{L(v),DFN(w)\}；$$

(3) 当从 v 的“儿子”节点 s 访问结束从 s 点退回到 v 点时，令

$$L(v)=\min\{L(v),L(s)\}。$$

只有对 v 的访问完全结束时，$L(v)$ 才能最后确定下来。利用 DFS 遍历算法访问图 G 的所有顶点和边，经递归下 $L(v)$ 的信息，从而可求去判定割切点和互连通块，割切点的充要条件如下：

若 v 点是割切点的充要条件是 v 有一“儿子”节点 s，使得

$$L(s)\geq DFN(v)。$$

为了判定互连通块的需要，下面的算法中设立了一个栈 $Stack$。

假定 $G=(V,E)$ 是连通图，则

(1) 对所有的 $v\in V$ 作【$Father(v)\leftarrow 0, Mark(v)\leftarrow 0$】，$i\leftarrow 1, Stack\leftarrow\Phi$。

(2) 任选一顶点 r，满足 $Mark(r)=0$，作【$DFN(r)\leftarrow i, L(r)\leftarrow i, Mark(r)\leftarrow 1, v\leftarrow r$】。

(3) 若所有和 v 关联的边均已被访问过，则转第 5 步；否则，任选一未访问过的边 (v,w)，给 (v,w) 以标志，并 $Push(Stack,(v,w))$，转(4)。

(4) (a) 若 $Mark(w)=0$，则作【$i\leftarrow i+1, DFN(w)\leftarrow i, L(w)\leftarrow i, Father(w)\leftarrow v$,
$Mark(w)\leftarrow 1, v\leftarrow w$，转(3)】。

(4) 若 $Mark(w)=1$，则作【$L(v)\leftarrow\min\{L(v),DFN(w)\}$，转(3)】.

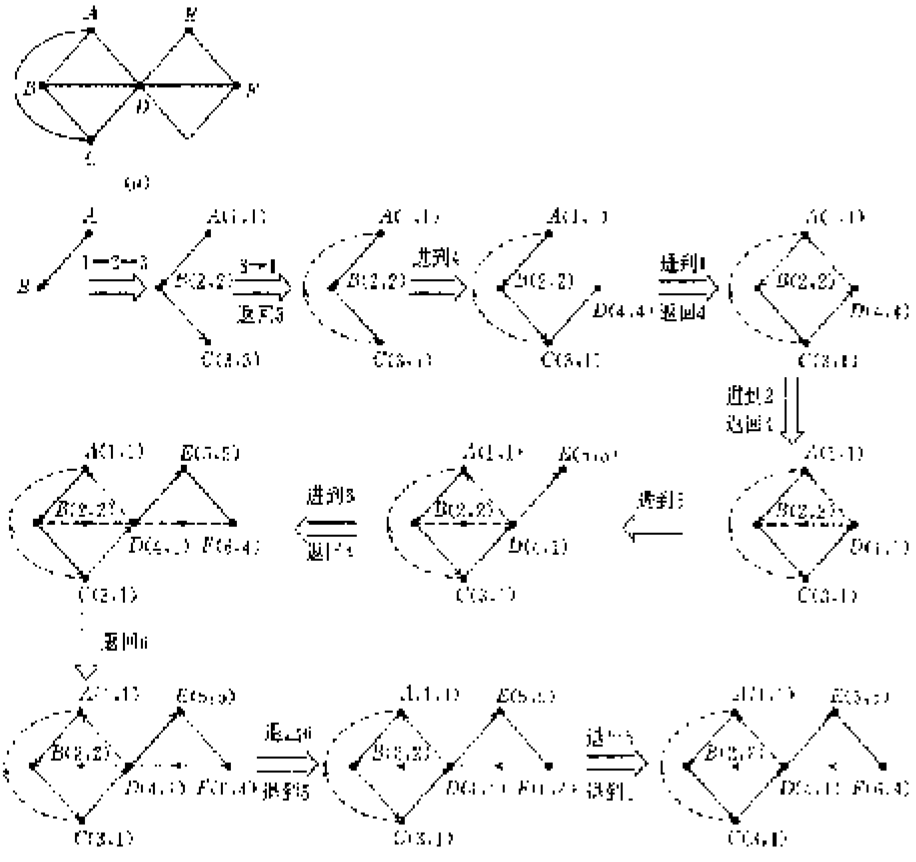
(5) 若 $Father(v)=0$ 则停止，否则作

【若 $L(v)\geqslant DFN(Father(v))$，则从栈 $Stack$ 中移出栈顶直至 $(Father(v),v)$ 边的所有边，它们构成双连通块。

令：$L(Father(v))\leftarrow\min\{L(v),L(Father(v))\}$，$v\leftarrow F(v)$，转(3)

】

例 见图 18.5.2。图 18.5.2(b) 每点标有两个数：第一个是深度优先数，第二个是 L 的值。从图 18.5.2 可知，顶点 5 作为 4 点的"儿子"结点，它的 $L(5)=4$，故顶点 4 是割顶点。该算法的复杂度为 $\max(|V|,|E|)$。



(b)

图 18.5.2

18.6 强连通块问题

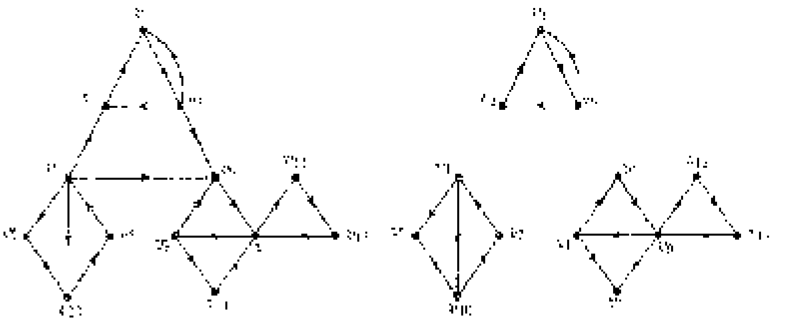
假定图 $G=(V,E)$ 是有向图，而且是强连通的，指的是对于它的任意两个顶点 u 和 v，图 G 存在一条从 u 到 v 的有向道路和一条从 v 到 u 的有向道路。

现代化城市的街道很多是单向的，也就是只允许向一个方向行车，但由街道和交叉点

向图等图必然是强连通的。图 G 的极大强连通子图叫做强连通块。

假定 $G_1=(V_1,E_1)$，$G_2=(V_2,E_2)$，…，$G_k=(V_k,E_k)$ 是图 $G=(V,E)$ 的强连通子图，T 是 G 的 DFS 森林，$T_1,T_2,\cdots,T_k$ 是 T 分别在 $G_1,G_2,\cdots,G_k$ 中导出的子图。在前面讨论了连通图的 DFS 森林可能是森林，未必是连通的，但对于强连通图，它的 DFS 树肯定是连通的。这一点读者可自己证明之。

图 18.6.1 是一个具有强连通子图的例子，(a) 是图 G，(b) 是它的强连通块。了解它们的结构对算法设计会有益处，算法的依据也就显得很直观的。



(a) (b)

图 18.6.1

强连通块可以收缩成一个点，产生一个新的有向图。比如图 18.6.1 可收缩成图 18.6.2 的形式。

若将图 G 之强连通块都收缩成点，形成有向图 G'，G' 不是强连通的，否则不然，则整个图 G 便是一个强连通块，它的收缩图将成为一个点。这些对我们了解有向图的 DFS 森林的构造是大有帮助的，各强连通块的 DFS 树是 G 的 DFS 森林的子图。

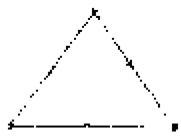


图 18.6.2

充分利用求连通块的思路，当用 DFS 搜索有向图 G，得到 DFS 森林，各顶点的 DFS 值记为 DFN，各有向边被分成树枝边、前向边、后向边、横跨边。在给出算法前，先讨论如下的几个定理。

定理 设 $G=(V,E)$ 是有向图，$G_i=(V_i,E_i)$ 是 G 的某一强连通块，$F=(V_F,E_F)$ 是该 G 的 DFS 森林，$T_i=(V_i,E_i\cap E_F)$，则：

(1) 若 u 和 v 是 V_i 上任意两点，则 u 和 v 在 F 有相同的"祖先"节点，即它们是同一个"祖先"节点的"后裔"节点。

(2) T_i 是一棵树。

证明 在证明前请注意一个事实，凡是后向边或横叉边都是从深度优先数 DFN 的值大的一端指向 DFN 的值小的一端。所以在强连通图之中从 DFN 的值较小的一点到 DFN 的值大的另一点存在一条通路，使得这通路都是由 DFS 树的树枝构成，也即从 DFN 值最小的一点沿 DFS 树到达 DFN 值大的另一点。现在来证明定理。

(1) 不失一般性，令 $DFN(u)<DFN(v)$，由于 u、v 同在一个连通块上，所以从 u 到 v 存在一条道路 R，R 上有一点 w，使对于 R 上其它各点 w' 可有：$DFN(w)\leq DFN(w')$，w' 点必然是 w 点的"后裔"节点。又由于 $DFN(w)\leq DFN(u)<DFN(v)$，所以 u 点也一定是点 w 的"后裔"节点，即 w 点是 u 和 v 的共同"祖先"。

(2) T_s 是各 V_s 各点在 DFS 森林的子树，r 是 T_s 的根节点，$v\in V_s$，若 u 是在从 r 到 v 的 DFS 树枝上，只要证明 $u\in V_s$ 就可以了。这是显然的，因为有一条从 r 沿 DFS 树枝到 u 的道路，从 u 通过 v 到 r 也有一条道路，所以 u 到 r、r 到 u 都有道路相通，即 r、u、v 同在一强连通块 G_s 上。

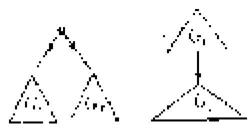

(a) (b)

图 18.6.3

树 T_s 的根节点 r 称为强连通块 G_s 的根节点。

若搜索的顺序从上往下，从左到右，假设 $r_1, r_2, \cdots, r_k$ 分别是依次得的强连通块 $G_1=(V_1,E_1)$，$G_2=(V_2,E_2)$，…，$G_k=(V_k,E_k)$ 的根节点。若 $i<j$，则 r_j 要访问 r_i 或属于 r_i，如图 18.6.3 所示。

所以 $G_i=(V_i,E_i)$ 强连通块是根 r_i 的"后裔"，但不是 $r_1, r_2, \cdots, r_{i-1}$ 的"后裔"。令

$$LL(v)=\min_{w\in S(v)}\{DFN(w)\}$$

其中 $S(v)$ 是从 v 出发在 DFS 树，最多经一后退边或一横跨边所能到达的顶点的集合。由定义立即可得，对强连通块的根节点有

$$LL(r_i)=DFN(r_i),\quad i=1,2,\cdots,k$$

若 $v\in V_i$，但 $v\neq r_i$，则至少存在一条道路 p 到 r_i，这道路必然包含一后退边或横跨边，故存在于 $DFN(v)>LL(v)$。从而得该可导出前面的定理。

定理 $G=(V,E)$ 是一有向图，v 是 G 的强连通块的根的充分必要条件是：$LL(v)=DFN(v)$。

求 G 的强连通块的根结点的方法并不困难，只要在对 G 进行 DFS 搜索过程中，对每个顶点进行访问的同时，计算并保留下 $LL(v)$ 的值，利用 $LL(v)=DFN(v)$ 作为判定强连通块的根结点的依据。

现在说明算法思路，主要是在利用 DFS 搜索法，计算出 $LL(v)$ 的值。

(1) 第一次访问 v 点时，置 $LL(v)\leftarrow DFN(v)$。

(2) 如果遇到后退边 (v,w) 时，修改 $LL(v)$ 如下：

$$LL(v)\leftarrow\min\{LL(v),DFN(w)\}$$

(3) 如果遇到横跨边 (v,w) 且处于同一强连通块中，则修改 $LL(v)$ 如下：

$$LL(v)\leftarrow\min\{LL(v),DFN(w)\}$$

(4) 当 v 的 DFS 搜索结束，由 S 回到 u 时，修改 $LL(u)$ 如下：

$$LL(u)=\min\{LL(u),LL(v)\}$$

在算法第(3)步，要判定 v、w 是否属同一强连通块，设计了一个字栈存点并来核 stack[ ]，同时对每一顶点设计一指针 point，开始时 $point(v)=0$，v 点没有进栈，当 v 进栈时，$point(v)=1$；出栈时，$point(v)=0$。

有向图的强连通分支算法：

(1) G 是给定的有向图，对 G 中每一顶点 $v \in V$，作【$Mark(v) \leftarrow 0$，$Father(v) \leftarrow 0$，$Point(v) \leftarrow 0$】（$Mark$ 标记点是否被访问，$Father$ 标定是否为根，$Point$ 标记进栈与否）；$i \leftarrow 1$；$Stack1 \leftarrow \varnothing$。

(2) 任选一个 $Mark$ 值为 0 的未访问点，记为 r 作为根，作【$DFN(r) \leftarrow i$，$LL(r) \leftarrow i$，$Mark(r) \leftarrow 1$，r 点压入栈 $Stack1$ 顶部，$point(r) \leftarrow 1$，$v \leftarrow r$】。

(3) 如果联于顶点 v 的全部出边都已检查过，则转(5)，否则任选一条未检查的出边，设为 (v,w)，现在将 (v,w) 标上已检查，并转(4)。

(4) 做完下列两种情况之一，再转第(3)步。

① 若 $Mark(w)=0$，（此时 (v,w) 是树枝边）

作【$i \leftarrow i+1$，$DFN(w) \leftarrow i$，$LL(w) \leftarrow i$，$Father(w) \leftarrow v$，$Mark(w) \leftarrow 1$，把 w 推到栈 $Stack1$ 顶部，且置 $point(w) \leftarrow 1$，$v \leftarrow w$】

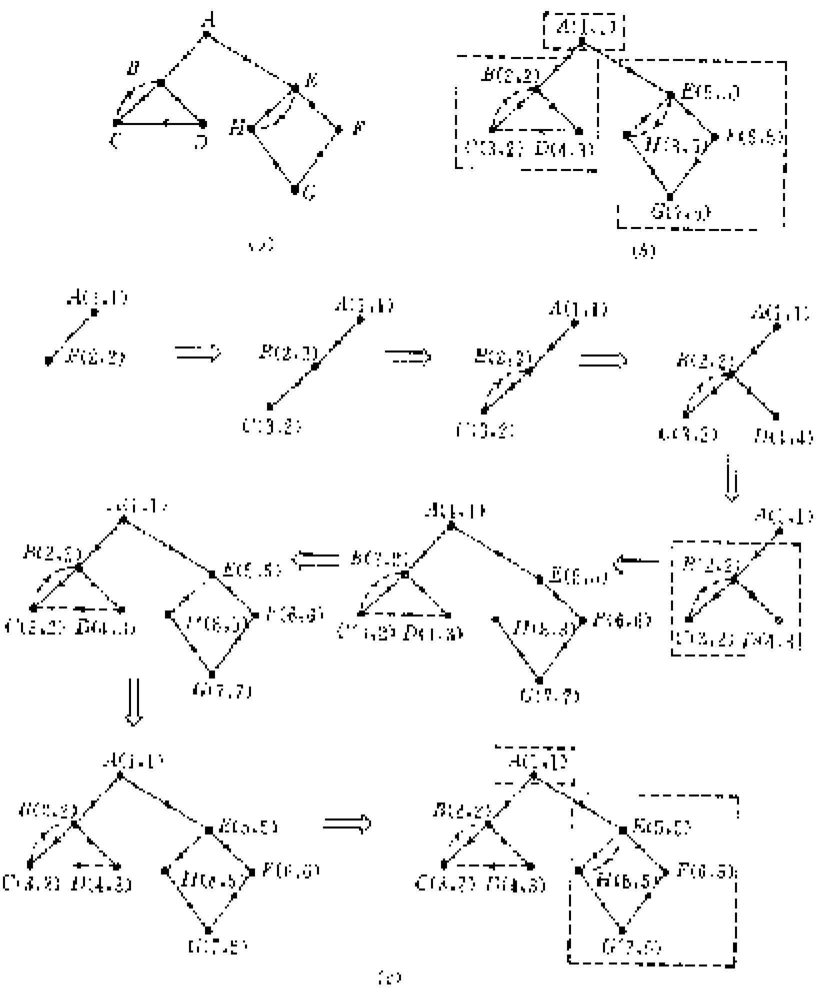


图 18.6.1

④ 若 $Mark(w)=1$，$DFN(w)<DFN(v)$ 且 $point(w)=1$ 则作【$LL(v)\leftarrow\min\{LL(v),DFN(w)\}$】

(5) 如 $LL(v)=DFN(v)$，则从栈 $stack$ 的顶部开始取出顶点，一直到取出 v 为止。(这些点组成强连通分支)。这些出栈的点，重新置 $point(x)\leftarrow 0$。

否则顺序执行下一步。

(6) 如 $Father(v)=0$ 转(7)。

否则作【$LL(Father(v))\leftarrow\min\{LL(Father(v)),LL(v)\}$；$v\leftarrow Father(v)$，转(3)】。

(7) 如每一顶点 x，有 $Mark(x)=1$，则转(8)，否则转(2)。

(8) 停止(所有的强连通分支均已找到)。

例如图 18.6.4(a)是有向图 G，(b)是 G 的强连通块，(c)是 DFS 算法的搜索过程。(b)中每一顶点都标有一对数($DFN(u)$,$LL(u)$)，例如 $G(7,5)$，即 $DFN(G)=7$，$LL(G)=5$。

18.7 BFS 算法

BFS 是英文"Breadth First Search"的缩写，意即广度优先搜索，它也是算法设计技术的一种，与 DFS 法相对应。对图 $G=(V,E)$ 进行 BFS 搜索的步骤可以概括叙述如下。从任意选定的 v 开始，依次检查所有与 v 点关联的边 $(v,a_1),(v,a_2),\cdots,(v,a_k)$，当上面这些边检查完毕后，再依次检查与 $a_1,a_2,\cdots,a_k$ 相关联的边 $(a_1,a_{11}),(a_1,a_{12}),\cdots,(a_1,a_{1j_1})$，$(a_2,a_{21}),(a_2,a_{22}),\cdots,(a_2,a_{2j_2}),\cdots,(a_k,a_{k1}),(a_k,a_{k2}),\cdots,(a_k,a_{kj_k})$。依此类推直到所有的边被检查，所有顶点均被访问为止。

举一个 BFS 搜索的例子来说明 BFS 方法。如图 18.7.1(a)的格局如何转变为(b)所示的状态，行走一步是空格周围的一个棋子和空格互换位置。

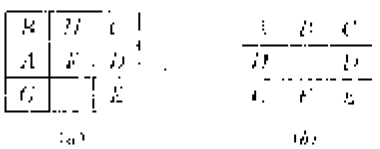


图 18.7.1

假定空格换位的顺序如图 18.7.2 所示，图 18.7.1(b)的搜索见图 18.7.2(b)。

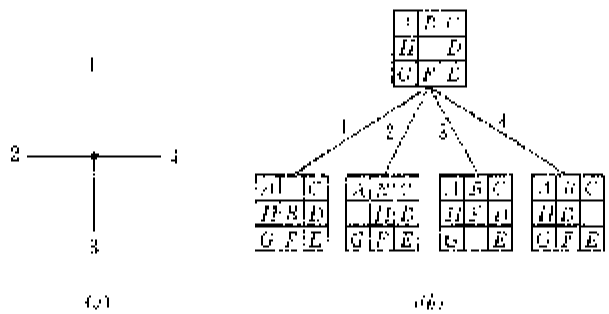


图 18.7.2

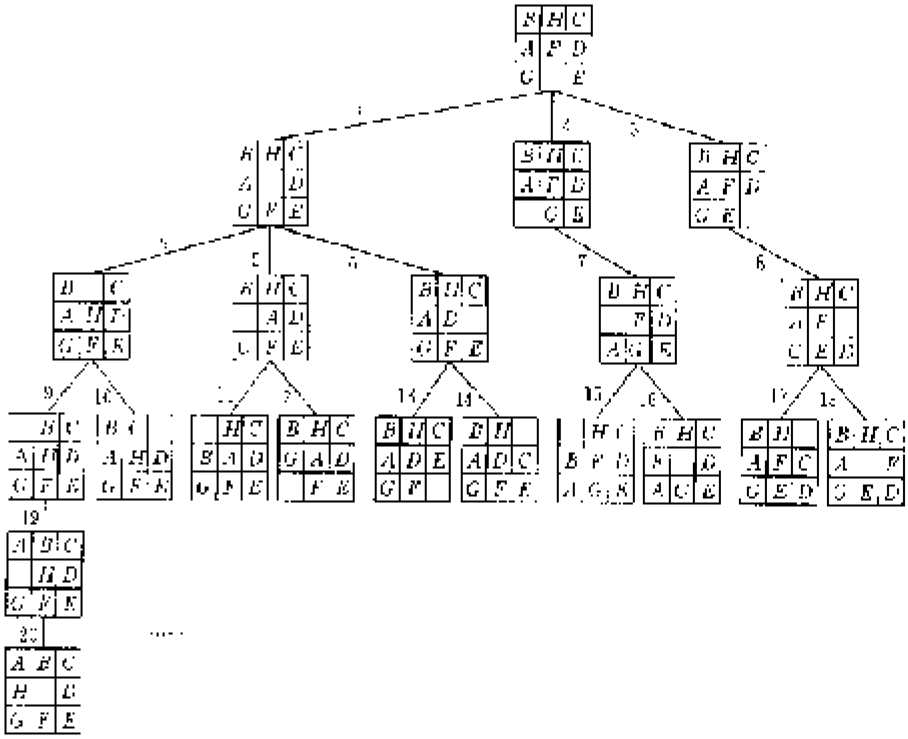


图 13.7.3

图 13.7.3 是搜索过程，从中可看出得到一种方案的具体步骤。用 BFS 算法同样能够从一个图，也能得到一棵 BFS 树（或 BFS 森林），其方法与 DFS 算法类似，在此就不一一叙述了。

习　　题

1. 已知有向图 $G=(V,E)$ 的邻接矩阵

$$A=(a_{ij})_{n\times n},\quad n=|V|$$

$$a_{ij}=\begin{cases}1 & (i,j)\in E\\ 0 & \text{其它}\end{cases}\qquad i,j=1,2,\cdots,n$$

(1) 试设计一算法计算各点的入度和出度，并讨论其复杂度

(2) 试设计一算法构造图

$$G_2=(V,E^2),$$

其中 $(v,w)\in E^2$，当且仅当存在 $v\in V$，使得

$$(u,v)\in E,(v,w)\in E$$

并讨论其复杂度

2. 已知图 $G=(V,E)$ 的树 T，求 T 的直径 d_T，

$$d_T=\max_{u,v\in T} d(u,v).$$

边(u,v)且u和v间无其他路。试设计一算法，并分析它的复杂度。

2. 已知有向图$G=(V,E)$，试设计一算法以判断对于任意两点u和v，是否存在一条从u到v的通路，并分析其复杂度。

3. 试证有向图$G=(V,E)$不存在回路的充要条件是进行一DFS时不存在后退边。

4. 试证无回路的有向图至少存在一入度为零的顶点，和一出度为零的顶点。

5. G是一无回路的有向图，试设计一将G的顶点的拓扑排序的算法，即若G有一有向边(u,v)，则u的序号必在v的前面。

6. 已知$G=(V,E)$是无回路的有向图，试讨论下面的算法给出图G顶点的拓扑排序，即反复地找出入度为零的顶点，打印它并删去以它为始点的所有边。并分析其复杂度。

7. 已知图$G=(V,E)$是有向图，G的强连通块用一个点来表示，得一点的集合V'。若u所代表的强连通块的点到v所代表的强连通块内的点，有边属于E，则构造有向边$(u,v)\in E'$，于是得图$G'=(V',E')$，G'称为G的收缩图，试证G'是一无回路的有向图。

8. 已知有向图$G=(V,E)$，试设计一算法以判断是否对于任意两点u和v，至少存在一条从u到v或v到u的通路，并讨论其复杂度。

第19章　α-β剪枝术和分支定界法

19-1　α-β剪枝术

最好的办法是通过例子来说明什么是α-β剪枝术，它本来就是一种技巧。比如有7根火柴，A、B两人依次从中取出一根或两根，但不能不取，当然也不能取多于2根，最后一个将火柴取尽的便是胜利者。用符号$\boxed{m_1}$表示轮到A时有m_1根火柴的状态，(m_2)表示轮到B时有m_2根火柴的状态，双方对弈时轮流进行，假定从A开始。图19.1.1中节点分别表示A取一根或两根，B所面对的两种状态。

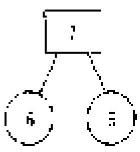


图 19.1.1

图19.1.2中节点边旁的括号里的数是它的搜索顺序号，括号外的数1或−1表示A取胜(或B取胜)。注意：各点的赋值是自下而上进行的。A的目的在于取胜，所以A的决策是取它的两个"儿子"节点值的最大者，同样B的目的在于让A失败，它的决策是取它的两个"儿子"节点值的最小者。

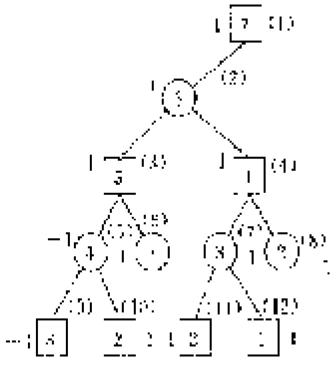


图 19.1.2

比如节点$\boxed{7}$的左儿子的值化1，它可以不问右"儿子"取什么值，决策是让B进入(5)的状态，不必顾虑其余，即对(5)的搜索可以省去，达到减少搜索的剪枝目的。同样道理，节点(5)的两个"儿子"都取1，故B就无法选择。节点$\boxed{5}$左儿子取值−1，但右儿子取值1，故A的决策应该让B进入(3)状态，而不取(4)，否则失败。请思考一下各节点的值为1或−1，是自下而上、自左而右地给出过程，若A走已能达到必胜地步，B次的搜索可省去，反之亦然。总之只要一边找到获胜的确定把握，另一边的搜索省去。

19.2　分支定界法和流动推销员问题

分支定界法是又一种用途十分广的算法，运用这方法的技巧性很强，不同类型的问题解法也各不相同，下面主要通过具体例子介绍它的思想。

例1　对称型的流动推销员问题，流动推销员问题是组合数学中的著名问题，在前面已有介绍。设$v_1,v_2,\cdots,v_n$是n个城市，

$$D=(d_{ij})_{n\times n}$$

是距离矩阵且满足$d_{ij}=d_{ji}$，$i,j=1,2,\cdots,n$，即从v_i到v_j的距离和从v_j到v_i的距离相等，

从某一城市出发，遍历各城市一次且仅一次，最后返回原地，求最短路径。

前面已讨论过由于对称 $d_{ij}=d_{ji}$，即从回路 $v_1\to v_2\to v_3\to v_4\to v_5\to v_1$ 和 $v_1\to v_5\to v_4\to v_3\to v_2\to v_1$ 有一样的路径长度。

例

$$
\begin{array}{c|ccccc|c}
 & & & & & & \\
\end{array}
D=\begin{pmatrix}
\infty & 14 & 1 & 15 & 2 \\
14 & \infty & 25 & 2 & 3 \\
1 & 25 & \infty & 9 & 9 \\
15 & 2 & 9 & \infty & 6 \\
2 & 3 & 9 & 6 & \infty
\end{pmatrix}
\begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix}
$$
$$\begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 \end{matrix}$$

对 $d_{ij}=d_{ji}$，故将 d_{ij} 和 d_{ji} 看作是相同的。将矩阵 D 对角线以上的元素从小到大排列有

$$d_{13},d_{15},d_{24},d_{25},d_{45},d_{35},d_{34},\cdots$$

取最小的 5 个求和，得

$$d_{13}+d_{24}+d_{15}+d_{25}+d_{45}=1+2+2+3+6=14$$

即

$$(1)\begin{pmatrix}13 & 24 & 15 & 25 & 45 \\ & & 14 & & \end{pmatrix}$$

表示，左肩上(1)表示搜索的顺序号，后面同此不另解说。显然下标中 5 出现了 3 次。若用 35 代替 45，则

$$d_{13}+d_{24}+d_{25}+d_{15}+d_{35}=1+2+3+6+9=21$$

即

$$(2)\begin{pmatrix}13 & 24 & 25 & 15 & 35 \\ & & 21 & & \end{pmatrix}$$

如排除 v_1v_5 这条路径起的要下标中 1、5 依然出现 3 次，现将搜索过程形象地表示如图 19.2.1 所示。如图 19.2.1 中(3)点表示在选择 v_1v_5 之后，但排斥了 $\overline{v_2v_5}$ 边的前提下的解为 20，依此类推。

图 19.2.1 中 15 表示选 v_1v_5 这条路径，$\overline{15}$ 为排除 $\overline{v_1v_5}$ 这条路径。搜索至(5)时发现（下式中(*)表示最佳路径）。

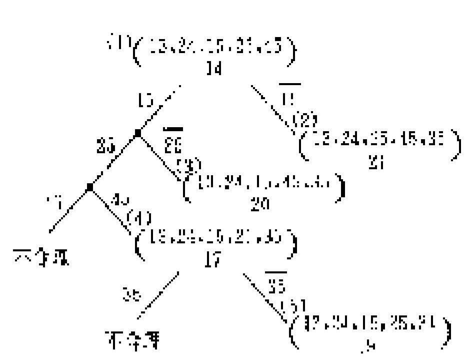


图 19.2.1

$$d_1 = d_{14} + d_{45} + d_{52} + d_{23} + d_{31} = 19 \qquad (4)$$

并得一条回路，而且是最佳旅行路：

$$v_1 \to v_4 \to v_5 \to v_2 \to v_3 \to v_1$$

因(2)式的下界为21，(3)式的下界为19，若大于19就没有进一步搜索的必要，所以(4)式便是最佳路径。

例 2 下述图的旅行推销员问题，其距离矩阵

$$D = (d_{ij})_{5\times5}$$

是非对称的，也就是 d_{ij} 不一定等于 d_{ji}。为了更好地理解下面的思想，不妨将 D 看成是旅费矩阵，即 d_{ij} 为从 v_i 到 v_j 的费用，从 v_i 到 v_j 的费用与从 v_j 到 v_i 的费用不同，这种可能性是存在的：

$$D = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ \\ \end{array} \begin{array}{c} \begin{bmatrix} \infty & 14 & 30 & 5 & 6 \\ 10 & \infty & 11 & 4 & 3 \\ 4 & 6 & \infty & 7 & 6 \\ 15 & 10 & 13 & \infty & 2 \\ 13 & 3 & 4 & 11 & \infty \end{bmatrix} \\ \begin{matrix} 1 & \quad 2 & \quad 3 & \quad 4 & \quad 5 \end{matrix} \end{array} \qquad (10.2.1)$$

为了方便起见，$v_1, v_2, \cdots, v_5$ 代以下标 $1, 2, \cdots, 5$。对 D 的每行减去该行的最小元素，或每列减去该列的最小元素，使得每行每列至少有一个零元素，比如

$$\begin{bmatrix} \infty & 14 & 30 & 5 & 6 \\ 10 & \infty & 11 & 4 & 3 \\ 4 & 6 & \infty & 7 & 6 \\ 15 & 10 & 13 & \infty & 2 \\ 13 & 3 & 4 & 11 & \infty \end{bmatrix} \begin{matrix} 5 \\ 3 \\ 4 \\ 2 \\ 3 \end{matrix} \xrightarrow{\text{行减 } 5,3,4,2,3} \begin{bmatrix} \infty & 9 & 25 & 0 & 1 \\ 7 & \infty & 8 & 1 & 0 \\ 0 & 2 & \infty & 3 & 2 \\ 13 & 8 & 11 & \infty & 0 \\ 10 & 0 & 1 & 8 & \infty \end{bmatrix} \xrightarrow{\text{第3列减去1}}$$

$$\begin{bmatrix} \infty & 9 & 24 & 0 & 1 \\ 7 & \infty & 7 & 1 & 0 \\ 0 & 2 & \infty & 3 & 2 \\ 13 & 8 & 10 & \infty & 0 \\ 10 & 0 & 0 & 8 & \infty \end{bmatrix}$$

第一行元素减去5，第二行元素减去3，……可以理解为从 v_1 出发的旅费一律降5个单位，从 v_2 出发的旅费一律降3个单位，……同样第三列元素一律减1，理解为进入 v_3 的旅费一律降1单位。由于旅游者对每一城市进入一次，出去也一次，故原来的旅游推销员问题的解也就是旅费矩阵为

$$
D' = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ \end{array}\begin{vmatrix} \infty & 8 & 24 & 0 & 1 \\ 7 & \infty & 7 & 1 & 0 \\ 0 & 2 & \infty & 1 & 2 \\ 13 & 8 & 10 & \infty & 0 \\ 10 & 0 & 0 & 3 & \infty \end{vmatrix}_{18} \qquad (19.2.2)
$$
$$
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array}
$$

的流动商人问题的解，矩阵右下角的 18(=5+3+4−2−3−1)是界。D' 矩阵每行每列都有至少一个"0"元素。由于 D' 第1行第4列元素为0，故若选取 $v_1\to v_4$ 的路径，排除从 v_1 到其它诸点的可能，以及排除从其它点进入 v_4 的可能，同时还要封锁从 v_4 直接返回 v_1，故划去第1行第4列，并将 d_{41} 的13改为∞，得

$$
\begin{array}{c} 2 \\ 3 \\ 4 \\ 5 \\ \end{array}\begin{vmatrix} 7 & \infty & 7 & 0 \\ 0 & 2 & \infty & 2 \\ \infty & 8 & 10 & 0 \\ 10 & 0 & 0 & \infty \end{vmatrix}_{18} \qquad (19.2.3)
$$
$$
\begin{array}{cccc} 1 & 2 & 3 & 5 \end{array}
$$

再从 v_4 出发到 v_5，因 D' 中 $d_{45}=0$，和上面类似可得

$$
\begin{array}{c} 2 \\ 3 \\ 5 \end{array}\begin{vmatrix} 7 & \infty & 7 \\ 0 & 2 & \infty \\ 10 & 0 & 0 \end{vmatrix} \xrightarrow{\text{第1行减7}} \begin{array}{c} 2 \\ 3 \\ 5 \end{array}\begin{vmatrix} 0 & \infty & 0 \\ 0 & 2 & \infty \\ 10 & 0 & 0 \end{vmatrix}_{25}
$$
$$
\begin{array}{ccc} 1 & 2 & 3 \end{array} \qquad\qquad \begin{array}{ccc} 1 & 2 & 3 \end{array}
$$

与取(v_1,v_4)边相反的，排除(v_1,v_4)边，只要令 $d_{14}\leftarrow\infty$ 即可，故得

$$
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{vmatrix} \infty & 8 & 24 & \infty & 1 \\ 7 & \infty & 7 & 1 & 0 \\ 0 & 2 & \infty & 1 & 2 \\ 13 & 8 & 10 & \infty & 0 \\ 10 & 0 & 0 & 3 & \infty \end{vmatrix}_{18} \xrightarrow{\text{第1行减1}} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{vmatrix} \infty & 7 & 23 & \infty & 0 \\ 7 & \infty & 7 & 1 & 0 \\ 0 & 2 & \infty & 1 & 2 \\ 13 & 8 & 10 & \infty & 0 \\ 10 & 0 & 0 & 3 & \infty \end{vmatrix}_{19}
$$
$$
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \qquad\qquad \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array}
$$

与取(v_4,v_5)边相反的是排除(v_4,v_5)边，即令(19.2.3)的矩阵中 $d_{45}\leftarrow\infty$，得

$$
\begin{vmatrix} 7 & \infty & 7 & 0 \\ 0 & 2 & \infty & 2 \\ \infty & 8 & 10 & \infty \\ 10 & 0 & 0 & \infty \end{vmatrix}_{18} \xrightarrow{\text{第3行减8}} \begin{vmatrix} 7 & \infty & 7 & 0 \\ 0 & 2 & \infty & 2 \\ \infty & 0 & 2 & \infty \\ 10 & 0 & 0 & \infty \end{vmatrix}_{26}
$$

其余依此类推。最后从右下方得一回路

$$
v_1\to v_4\to v_5\to v_2\to v_3\to v_1
$$

总费用为25单位，实际上也是最优解，搜索全过程如图19.2.7所示。凡是下界不低于25者一律不予继续搜索。

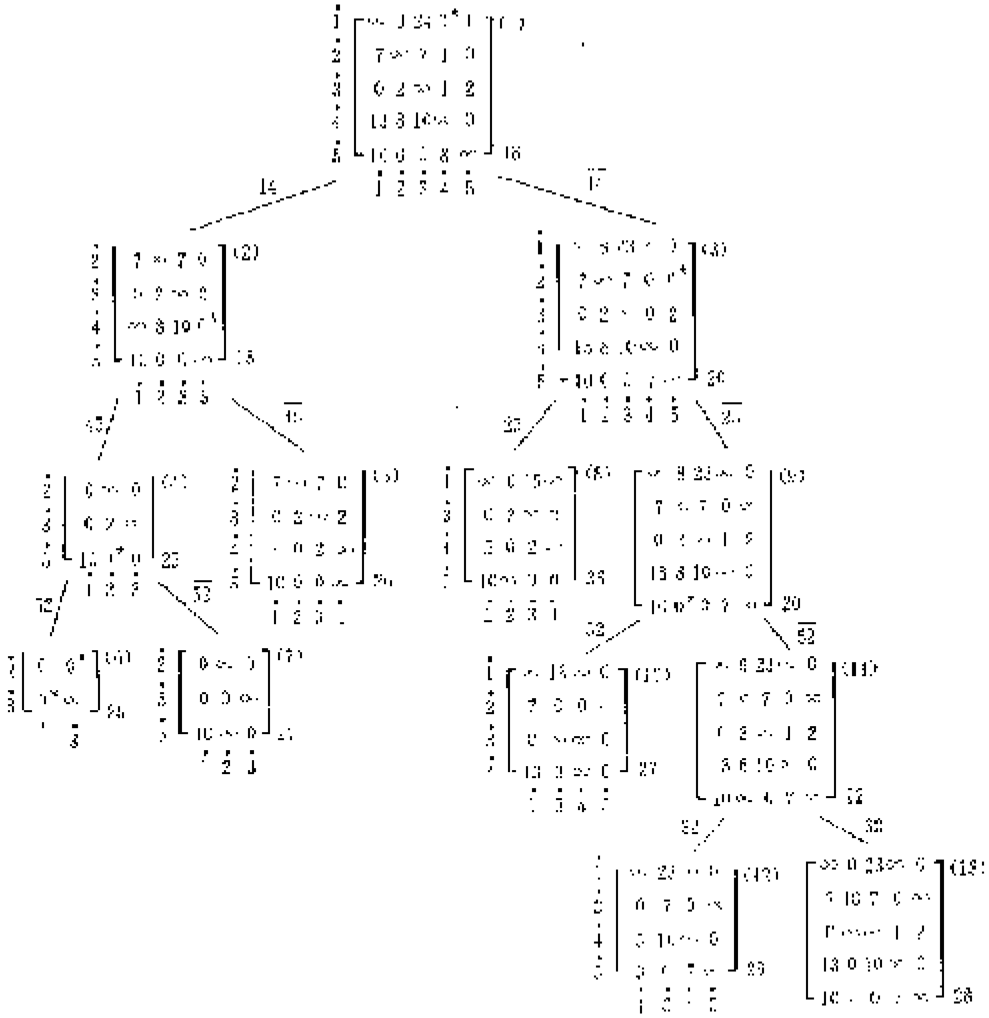


图 19.2.2

(5)和(7)的最低界均超过 25，故没有向下搜索的必要。图 19.2.2 中()里的数是顶点的标号，也可认作是搜索的顺序。搜索的策略是哪点的界最低，最先搜索哪点。

19.3 同顺序加工任务安排问题

设 J_1,J_2,J_3,J_4 是四项待加工的任务，它们的工序一样，即先在机器 m_1 上加工，然后在机器 m_2 上加工，最后在机器 m_3 上加工。加工时间矩阵是：

$$T=\begin{array}{c|ccc} J_1 & 5 & 7 & 9 \\ J_2 & 12 & 5 & 2 \\ J_3 & 9 & 8 & 5 \\ J_4 & 5 & 8 & 10 \\ & m_1 & m_2 & m_3 \end{array}=(t_{ij}).$$

t_{ij}即为任务J_i在机器m_j上加工所需时间，如若加工顺序是：

$$J_2 \to J_3 \to J_1 \to J_4$$

则从开始到结束所需的时间可计算如图 19.3.1。

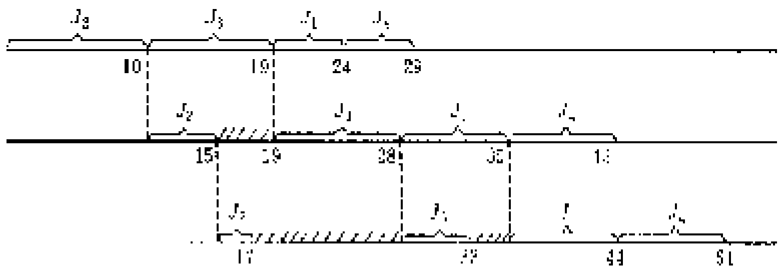


图 19.3.1

图中“///”表示机器空闲等待任务，比如机器m_2加工J_2于开始后15单位时间结束，但m_1加工J_3于19单位时间才能完结，所以机器m_2于15至19单位时间内空闲等待任务。加工顺序$J_2\to J_3\to J_1\to J_4$需51单位时间。现在要找一最佳的加工顺序，使完成任务的总时间达到最短。

分支定界法的关键在于估计下界，比如从J_1开始的加工顺序，估计加工所需的最短时间为：

$$t_{11} + \sum_{i=1}^{4} t_{i2} + \min_{k\neq 1} t_{k3}$$

即在理想状态下机器m_1加工完J_1，机器m_2无空闲，最后在机器m_3上加工的是任务J_k，其中k满足：

$$t_{k3} = \min_{i\neq 1} t_{i3}$$

即最后在机器m_3上加工的任务恰好是机器m_3上加工时间最短的任务J_k。当然J_k必须不是最先加工的任务J_1。因此式(*)给出了从任务J_1开始所需加工时间的下界，我们得图 19.3.2。

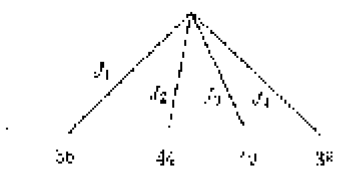


图 19.3.2

从图 19.3.2 可见，在这种理想状态下，以J_1或J_4开始的任务安排完成的时间可能最短，其中从J_1开始的任务安排为：

$$5 + (7 + 3 + 7 + 3) + 4 = 35$$

即完成的时间不可能少于35单位，其余同理类推。

一般从J_i开始，继以J_j的任务安排，理想完成时间应为：

$$f = t_{?} - \sum t_{i2} - \min t_{i3}$$

$J_? \to J_? \to J_?$ 的加工顺序所需的加工时间的最短界为：

$$f_? = f_? + t_? - \sum t_{i?} - t_?$$

利用分支定界法搜索最优解的过程表示如图 19.3.3：

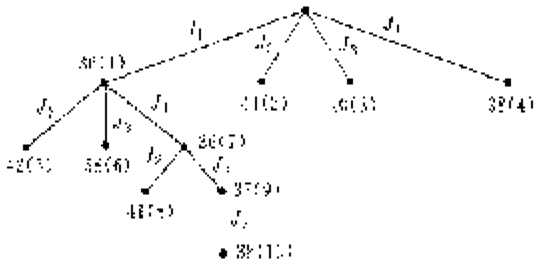


图 19.3.3

图中每个节点都有两个数，其中( )里的数表示搜索顺序的序号。最优的加工顺序是：

$$J_? \to J_1 \to J_? \to J_?$$

总时数为 38 单位，排序安排如图 19.3.4 所示。

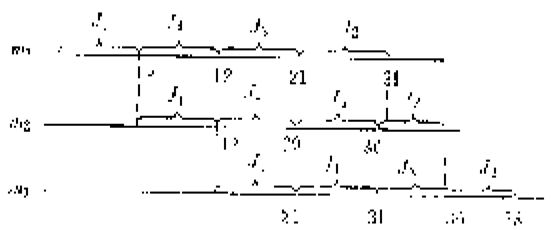


图 19.3.4

按估计值由小到大最佳优先搜索为原则，最先找到的是总时数为 40 单位，若估计的界不小于 38，就无搜索的必要。从图 19.3.4 可知剪去的搜索枝是非常可观的。分支定界的效果是很理想的。

综合以上几个例子，可以看到分支定界法的基本思想是采用 DFS 策略，并对每个结点估计一权值（在求极小值问题中对应于从该结点出发能得到的解的下界，若是最大值问题则反之），在搜索时利用权值来决定下一搜索结点的先后顺序（对于最小化问题，权值越小越优先），一旦搜索到一个解，利用这个解与权值可剪去那些不可能得到最优解的结点，使所有结点均被搜索或剪去，从而大大节省搜索的结点数。

分支定界法可应用于大量组合优化问题，其关键技术在于各节点权值如何估计，可以说，一个分支定界求解方法的效率基本上由值界方法所决定，若界估计不好，在极端情况下将与穷举搜索没有区别。

习　题

1. 试解下面距离矩阵的旅行商人问题

$$D=\begin{array}{c}v_1\\v_2\\v_3\\v_4\\v_5\\ \end{array}\begin{bmatrix}0 & 39 & 7 & 55 & 6\\39 & 0 & 41 & 73 & 72\\7 & 41 & 0 & 56 & 63\\55 & 73 & 56 & 0 & 18\\6 & 72 & 63 & 18 & 0\end{bmatrix}$$
$$\qquad\quad v_1\quad v_2\quad v_3\quad v_4\quad v_5$$

2. 试解下面距离矩阵的旅行商人问题

$$D=\begin{array}{c}v_1\\v_2\\v_3\\v_4\\v_5\\ \end{array}\begin{bmatrix}0 & 8 & 6 & 5 & 9\\8 & 0 & 7 & 13 & 12\\6 & 5 & 0 & 11 & 9\\7 & 9 & 11 & 0 & 4\\5 & 12 & 10 & 4 & 0\end{bmatrix}$$
$$\qquad\quad v_1\quad v_2\quad v_3\quad v_4\quad v_5$$

3. 试解下面距离矩阵的旅行商人问题

$$D=\begin{array}{c}v_1\\v_2\\v_3\\v_4\\v_5\\ \end{array}\begin{bmatrix}0 & 7 & 5 & 7 & 8\\8 & 0 & 5 & 13 & 12\\6 & 5 & 0 & 11 & 10\\5 & 14 & 12 & 0 & 5\\10 & 13 & 11 & 7 & 0\end{bmatrix}$$
$$\qquad\quad v_1\quad v_2\quad v_3\quad v_4\quad v_5$$

4. 解下面成本矩阵的最佳匹配问题

$$C=\begin{array}{c}A\\B\\C\\D\\ \end{array}\begin{bmatrix}11 & 13 & 13 & 8\\12 & 13 & 11 & 8\\13 & 12 & 12 & 10\\12 & 10 & 13 & 9\end{bmatrix}$$
$$\qquad\quad \mathrm{I}\quad \mathrm{II}\quad \mathrm{III}\quad \mathrm{IV}$$

其中 A,B,C,D 分别是四种材料，Ⅰ，Ⅱ，Ⅲ，Ⅳ 是四种产品。

5. α-β 剪枝技术中火柴的例子是这局的对策问题，也就是 n 个火柴的情况下，先下手的必胜或必败是确定的局势，除非对方失误，请找出它的规律。

6. 若 J_1,J_2,J_3,J_4 四个任务在 m_1,m_2,m_3 机器上以此序加工，加工的时间矩阵为

$$T=\begin{array}{c}J_1\\J_2\\J_3\\J_4\end{array}\begin{bmatrix}9 & 7 & 5\\10 & 5 & 3\\7 & 8 & 10\\5 & 7 & 9\end{bmatrix}$$

求最佳的加工顺序，使从开始到结束加工时间最短。

7. 10.3 的例子若要求求出所有最佳加工顺序，应该如何处理？并计算之。

第20章 整数规划

20.1 概 述

大量运筹学问题要求线性规划给出整数解，即归结为整数规划问题。

例1 背包问题：有一旅行者要从 n 件物品中选取不超过 b 公斤重的行李随身携带，要求总价值最大。

设第 j 种物品的重量为 a_j，价值为 c_j，$j=1,2,\cdots,n$。问题为：

$$\max z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leqslant b$$

$$x_j = 0 \text{ 或 } 1,\ j = 1,2,\cdots,n$$

即 $x_j=1$ 表示选取了第 j 种物品，否则 $x_j=0$。

例2 流动推销员问题：即有一商人要从 v_1 城出发，遍访 $v_2,v_3,\cdots,v_n$ 城各一次，最后返回 v_1，要求总路程最短。这个问题前面已给出分支定界解法。设从 v_i 城到 v_j 城的距离为 d_{ij}，问题为求 x_{ij} 及 u_i 使满足：

$$\min z = \sum_{i,j} d_{ij}x_{ij}$$

$$\sum_{i=1}^{n} x_{ij} = 1,\ j = 1,2,\cdots,n \tag{20.1.1}$$

$$\sum_{j=1}^{n} x_{ij} = 1,\ i = 1,2,\cdots,n \tag{20.1.2}$$

$$u_i - u_j + nx_{ij} \leqslant n-1,\ i,j = 1,2,\cdots,n,\ i \neq j \tag{20.1.3}$$

$$x_{ij} = 0 \text{ 或 } 1,\ u_i \text{ 为实数}$$

前两组约束条件(20.1.1)和(20.1.2)表示 $v_1,v_2,\cdots,v_n$ 各城市分别进入一次，后一组条件为使得不至于象图20.1.1所示那样，出现多于一个的非连通的回路。例如令 $x_{12}=x_{23}=x_{31}=1$，满足约束条件(20.1.1)，(20.1.2)，然而：

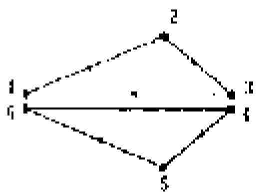


图 20.1.1

$$u_1 - u_2 + 6 \leqslant 5$$

$$u_2 - u_3 + 6 \leqslant 5$$

$$u_3 - u_1 + 6 \leqslant 5$$

故相加导致矛盾 $6 \leqslant 5$。

例3 资金分配问题：设有 n 个投资项目 $J_1,J_2,\cdots,J_n$，已知 $B=(b_1b_2\cdots b_m)^T$，$A=(a_{ij})_{m\times n}$，$C=(c_j)_{1\times n}$，其中 b_i 等于第 i 年的投资金额，a_{ij} 为第 i 年项目 J_j 所需的投资金额，c_j 为项目 J_j 的利润。资金的合理分配问题导致

$$\max z = CX$$

$$AX \leq B$$

$$x_i = 0 \text{ 或 } 1, i = 1,2,\cdots,n$$

其中 $X = (x_1, x_2, \cdots, x_n)^T$。

例4 下料问题：设有长度为 l 的钢材，要截成长度为 $l_1, l_2, \cdots, l_m$ 的材料，其数目分别为 $b_1, b_2, \cdots, b_m$ 根，已知 $A = (a_{ij})_{m \times n} = (P_1, P_2, \cdots, P_n)$，其中 $P_j = (a_{1j}, a_{2j}, \cdots, a_{mj})^T$ 是一种截割方案，即存在整数 $l_1, l_2, \cdots, l_m$ 使得：

$$a_{1j} l_1 + a_{2j} l_2 + \cdots + a_{mj} l_m \leq l, j = 1,2,\cdots,n$$

即矩阵 A 给出了 n 种截割方案，每一列是一种方案。

设 x_j 为按 P_j 列方案下料的钢材数目，合理的下料问题为：

$$\min z = x_1 + x_2 + \cdots + x_n$$

$$\begin{cases} a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n \geq b_1 \\ a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n \geq b_2 \\ \cdots\cdots \\ a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n \geq b_m \end{cases}$$

$$x_j \text{ 为非负整数 } j = 1,2,\cdots,n$$

上面例子说明运筹学中提出了大量的整数规划问题。计算机网络系统中也存在类似的情况，因为计算机网络系统拥有大量的文件（比如 n 个文件）如何将这些文件分配到 m 个数据库使得某些指标达到最优，如存储费用或通信费用最小，网络寿命或可靠性达到最高，这类称为文件分配问题。分布式数据库的出现使文件分配问题变得日益复杂而且重要，约束条件无非是每个数据库所分配文件总长度不超过它的容量，文件的存取时间不超过允许的限度，文件分配问题就导致 0-1 规划问题，这里不多讨论。

从以上诸例可见，整数规划实为线性规划问题，要求在变量取整数中求最优解。很容易想到利用线性规划求解，然后取整给出整数规划的解或给出近似解，但这么做并不能总是行得通的。比如许多问题变量只限 0、1，均可能表示“是”与“非”两种可能，用线性规划求解，然后取整的做法本身就缺少依据。如

例

$$\max z = x_1 + x_2$$

$$\begin{cases} 3x_1 - x_2 \leq 3/2 \\ -x_1 + x_2 \leq 1/2 \end{cases}$$

$$x_1, x_2 = \text{非负整数}$$

若将本问题改为一般线性规划问题，可得最优解：

$$x_1 = 1, x_2 = \frac{3}{2}$$

$$\max z = \frac{5}{2}$$

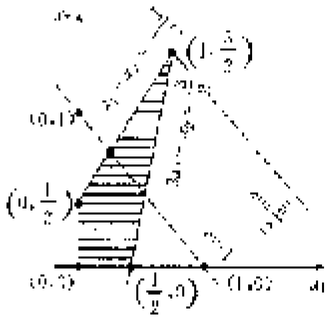


图 23.1.2

取整得 $x_1 = 1, x_2 = 1$ 或 $x_2 = 2$，具体参见图 23.1.2。

实际上问题的最优解是 $x_1 = x_2 = 0, \max z = 0$，甚至于 $(1,1)$ 点根本不是允许解域的

同节点。

下面我们将介绍搜索技术及分支定界法在整数规划中的应用。

20.2 0-1规划和它的DFS搜索(隐枚举)解法

整数规划中要求变元x_i只取0,1两种值时称为0-1规划。对于整数规划,除问题本身就只有两种状态外,已知变量的上界的整数规划问题都可以化为0-1规划问题。比如已知$0\leqslant x_i<2^{k+1}$,则令:

$$x_i = 2^k y_{ik} + 2^{k-1} y_{i,k-1} + \cdots + 2y_{i1} + y_{i0}$$

其中$y_{ij}=0$或1,$j=1,2,\cdots,k$。以之代入原问题,于是原问题导致求关于新引进的变量y_{ij}的0-1规划问题。

n个变元的0-1规划问题可以通过对2^n种状态进行穷举。

例

$$\max z = 2x_1 - x_2 - x_3$$

$$\begin{cases} x_1 + 3x_2 + x_3 \leqslant 2 & (20.2.1) \\ 4x_2 + x_3 \leqslant 5 & (20.2.2) \\ x_1 + 2x_2 - x_3 \leqslant 2 & (20.2.3) \\ x_1 + 4x_2 - x_3 \leqslant 4 & (20.2.4) \end{cases}$$

$$x_1, x_2, x_3 = 0 \text{或} 1$$

(x_1,x_2,x_3)共有$2^3=8$种状态,分别枚举如表20.2.1。

表 20.2.1

约束条件 \ $x_1x_2x_3$	(1)	(2)	(3)	(4)	$X\in S$	z
000	0	0	0	0	是	0
001	1	1	-1	-1	是	-1
010	(3)	4	2	4	非	
011	(4)	5	1	3	非	
100	1	0	1	1	是	2
101	2	1	0	0	是	1
110	(4)	4	(3)	(5)	非	
111	(5)	5	2	4	非	

表20.2.1中()表示括号里的数不满足该约束条件,S是允许解域。故得最优解:

$$x_1 = 1, x_2 = x_3 = 0, \max z = 2$$

n个变元的0-1规划,用穷举法要对2^n种状态进行一一检验,故其复杂度为$O(2^n)$,是典型的指数型算法,n较大时实际上是不可行的。下面结合整数规划问题介绍DFS搜

索法在解整数规划中的应用。

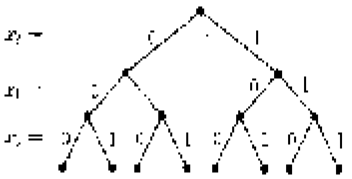


图 20.2.1

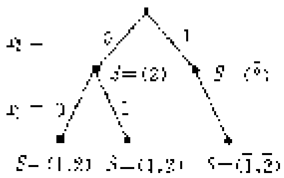


图 20.2.2

设问题的搜索空间如图 20.2.1 所示，为描述搜索过程的当前状态，引进集 S，例如：$S=\{x_1=1,x_2=0\}$ 表示先 $x_2=0$，后 $x_1=1$ 的顺序，或简单地记为 $S=\{\bar{1},2\}$，顶元素为 $x_i=1$，这作为一种约定。搜索树的每一顶点都对应一种状态，如图 20.2.2。搜索从树根开始，自上而下，自左而右，其步骤可叙述如下：

(1) $k\leftarrow 1$，$S\leftarrow\varnothing$。

(2) $x_k\leftarrow 0$，k 进入 S。

(3) 若按 S 的最后元素要放弃则转(4)，否则，转(5)。

(4) 若 $x_k=0$，则【$x_k\leftarrow 1$，转(3)】，

否则，【$k\leftarrow k-1$，若 $k\neq 0$，返转(4)

否则结束】。

(5) 若 $k=n$，则转(4)，

否则【$k\leftarrow k+1$，转(2)】。

例 $\max z = x_1+x_2+x_3$

$2x_1-5x_2+6x_3\leqslant -4$

$x_i=0$ 或 1 $\quad i=1,2,3$

设搜索空间如图 20.2.1。

(1) 对 $S=\{2\}$，$k=1$，由于 $x_1=0$ 不可能满足约束条件，即在 $x_2=0$ 的情况下，无论 x_1，x_3 如何取值约束条件都不成立。故这子树放弃。

(2) $S=\{\bar{2}\}\Rightarrow S=\{1,\bar{2}\}\rightarrow S=\{3,1,\bar{2}\}$，$k=3$，对应 $z=1$，

(3) $k=3$，故转变为 $S=\{\bar{3},1,\bar{2}\}$，由于 $-5+6=1>-4$，故 $x_3=1$ 应放弃，后退为 $S=\{\bar{1},\bar{2}\}$，不满足约束条件再后退，直后退到 $k=0$ 结束，搜索过程如图 20.2.3，其中 S 的下标标志着先后顺序。搜索过程可形象地描绘成流程图如图 20.2.4 所示。

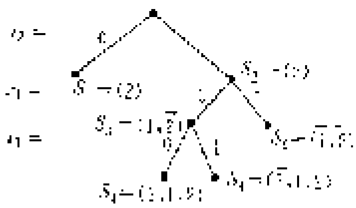


图 20.2.3

剪枝算法的关键在于决定放弃开发子树后退的策略，使得剪去的树枝越多，效果越好。剪枝的依据是什么？下面假定我们讨论的问题为

$$\max z=\sum_{j\in N} c_j x_j \qquad c_j\geqslant 0, j\in N$$

其中：$N=\{1,2,\cdots,n\}$

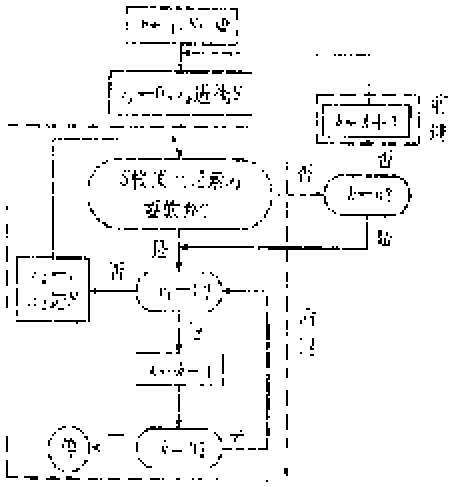


图 [illegible]

约束条件为：

$$\sum_{j\in N} a_{ij}x_j + s_i = b_i, \quad i \in M = \{1,2,\cdots,m\}$$

$$x_j = 0 \text{ 或 } 1, \ j = 1,2,\cdots,n$$

$$s_i \geqslant 0, \ i \in M$$

如若要求目标函数 z 取极大值，可设法将问题转化为极小值问题即将它转化为求 $\min(-z)$ 问题。

为方便，可假定所有的 c_j 都是非负的。如若 $z=\sum_{j\in N} c_jx_j$ 中有系数 $c_k<0$，可令 $y_k=1-x_k$ 代替 x_k，结果使得目标函数 y_k 的系数为正 $c_k'=-c_k$。

在搜索过程中为了避免盲目性，判定前进方向、决定后退的策略十分重要，其基本准则包括以下两点：

(1) 若已找到一允许解及其相应目标函数值 z_{min}，凡是前进会导致目标函数值 z 不小于 z_{min} 的，予以放弃；

(2) 前进一步不满足约束条件的应予以放弃。

对应于状态 S_k 有

$$s_i^{(k)} = b_i - \sum_{j\in S_k^+} a_{ij}, \quad i \in M,$$

$$z_k = \sum_{j\in S_k^+} c_j$$

其中

$$S_k^+ = \{j \mid j \in S_k, x_j = 1\}$$

若对于所有的 $i\in M$，恒有 $s_i^{(k)} \geqslant 0$，$s_i^{(k)}$ 实际上就是松弛变量，则相应地有：

$$x_j^{(k)} = \begin{cases} 1, & j \in S_k^+ \\ 0, & j \notin S_k^+, \ j \in N \end{cases}$$

作它们的 X_k，……一个或若干个违反的约束条件，且……不能给出可行解。反之若存在 $S_i^{(k)} < 0$，则 X_k 不在可行域内。

在 X_k 出发可依次向下搜索，步骤如下：

(1) 排除不在可行解域的点：

$A_k \triangleq \{j \in N \setminus S_k \mid$ 使得有 $S_i^{(k)} < 0$ 的 i 均有 $a_{ij} \geq 0\}$。$S_i^{(k)} < 0$ 表明 X_k 点不在可行解域上，此时 $j \in A_k$ 的变量 x_j 从原来的 $x_j = 0$ 改为 $x_j = 1$，不仅不能改善 X_k 的可行性，反而适得其反。故第1步就是集合 A_k，对于 $j \in A_k$ 的变量 x_j 予以放弃，从而使搜索范围缩小，从 N 中排除了 S_k 还排除了 A_k。

(2) 排除不能改进目标函数值的点：

向下搜索已不可能，则应从 S_k 后退，否则令：

$$C_k \triangleq \{j \in B_k \mid c_j + f(X_k) \geq z_{min}\}$$

C_k 是 B_k 的子集，对于 $j \in C_k$ 的 x_j，从 $x_j = 0$ 改为 $x_j = 1$，虽可改善可行性，但由于 $c_j + f(X_k) \geq z_{min}$，其目标函数值只会增加，不是我们所要求的，故 $j \in C_k$ 的 x_j 也不予考虑。从集合 B_k 中排除 C_k，搜索的范围进一步缩小。

(3) 进一步判断是否可能得到可行解：$D_k \triangleq B_k \setminus C_k$，若 $D_k = \phi$，则无可行解存在之可能，故应后退；否则令

$$E_k \triangleq \{i \in M \mid S_i^{(k)} < 0, \sum_{j \in D_k} \min\{0, a_{ij}\} > S_i^{(k)}\}$$

若 E_k 不是空集，说明存在 $i \in M$，使得 $S_i^{(k)} < 0$，而且绝对值甚如此之大，无法改善其可行性。因只有 $a_{ij} < 0$ 的 x_j 从 $x_j = 0$ 改为 $x_j = 1$ 方有助于使 $S_i^{(k)}$ 由负改为正的，或虽然 $S_i^{(k)}$ 依然为负，但绝对值减少。E_k 非空，即存在 i 使

$$\sum_{j \in D_k} \min\{0, a_{ij}\} > S_i^{(k)}$$

说明 $S_i^{(k)} < 0$ 绝对值之大，不存在从 $S_i^{(k)} < 0$ 改为正的可能性了。即对于 $j \in D_k$ 的 x_j 由 $x_j = 0$ 改为 $x_j = 1$，至少有一个约束条件无论如何无法满足，故应从 S_k 後退。故若 $E_k \neq \phi$，则后退，否则转(4)。

(4) 计算

$$v_j = \sum_{i=1}^{m} \min\{0, S_i^{(k)} - a_{ij}\}, \quad j \in D_k$$

若 v_j 中的最大者为 v_{j_k}，即

$$v_{j_k} = \max_j \{v_j\}$$

则令 $x_{j_k} = 1$，即 x_{j_k} 从0改为1。其中

$$\sum_{i=1}^{m} \min\{0, S_i^{(k)} - a_{ij}\}$$

是用以衡量由 $x_j = 0$ 改为 $x_j = 1$ 所引起的不可行性有所改善的程度。

$$v_{j_k} = \max_j \{v_j\}$$

是求出其中最大的，实际上是求

$$\min \sum_{S_i^{(k)} - a_{ij} < 0} |S_i^{(k)} - a_{ij}|$$

整个判断过程可用流程图描述如图 23.2.5.

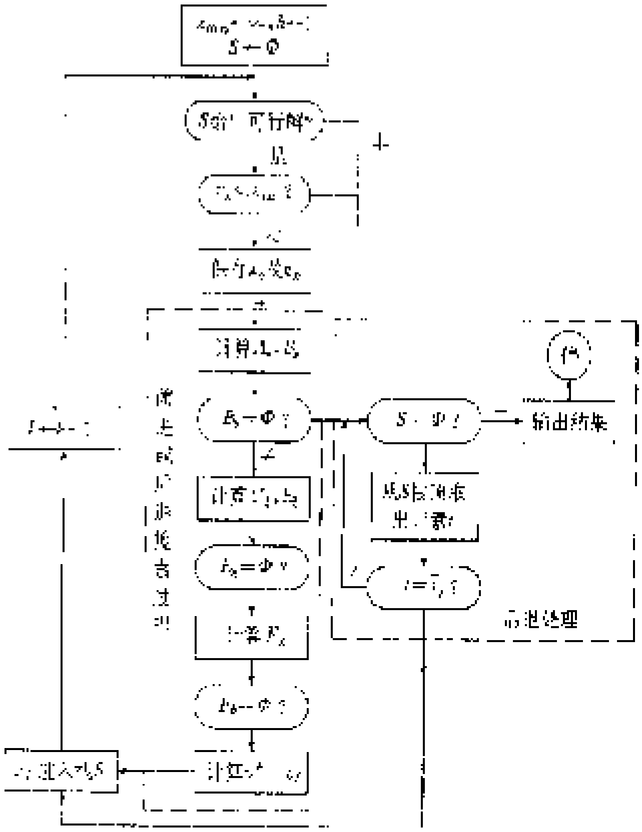


图 23.2.5

公式：

$A_k \triangleq \{j \in N \setminus S_k$，对 $S^{(k)}$ 的 i 有 $a_{ij} > 0\}$

$B_k \triangleq N \setminus (A_k \cup S_k)$

$C \triangleq \{j \in B_k \mid z_k + c_j \geq z_{\min}\}$

$E_k \triangleq B_k \setminus C$

$F_k \triangleq \{i \in M \mid S^{(k)} < 0, \sum_{j \in E_k} \min\{0, a_{ij}\} > S^{(k)}\}$

$v_j \triangleq \sum_{i=1}^{m} \min(0, S^{(k)} - a_{ij}), j \in E_k$

$v_{j^*} = \max_j (v_j)$

为了正确理解算法，应该耐心地分析一个例子是绝对必要的。只有这样才能了解它的含义。

例 $\min z = 5x_1 - 2x_2 + 5x_3 + 2x_4 + 3x_5$

$-x_1 - x_2 + x_3 + 2x_4 - x_5 \leq 2$

$7x_1 \quad - 3x_3 - 4x_4 - 3x_5 \leq -2$

$$11x_1-6x_2-3x_4-3x_5\leqslant -1$$

$$x_j=0,1,\quad j=1,2,\cdots,5$$

引进 $S_1,S_2,S_3\geqslant 0$，使得

$$\min\ z=3x_1-2x_2+5x_3+2x_4+3x_5$$

$$-x_1-x_2+x_3+2x_4-x_5+S_1=1\qquad(*)$$

$$-7x_1+3x_3-4x_4-3x_5+S_2=-2$$

$$11x_1-6x_2-3x_4-3x_5+S_3=-1$$

$$x_j=0,1,j=1,2,\cdots,5;\ S_i\geqslant 0,i=1,2,3$$

(1) $(S_1^0\ S_2^0\ S_3^0)^T=(1\ -2\ -1)^T$

故 $X=(0\ 0\ 0\ 0\ 0)^T$ 不是允许解故……

因存在 $j=3$，使 $a_{ij}\geqslant 0$，故 $A_1=\{3\}$

$B_1=\{1,2,4,5\}$，$C_1=\phi$，

$E_1=\{1,2,4,5\}$，

$F_1=\phi$，

$v_1=\min(0,1+1)+\min(0,-2+7)+\min(0,-1-11)=-12$

$v_2=\min(0,1+1)+\min(0,-2)+\min(0,-1+6)=-2$

$v_4=\min(0,1-2)+\min(0,-2+4)+\min(0,-1+3)=-1$

$v_5=\min(0,1+1)+\min(0,-2+3)+\min(0,-1+3)=0$

$v_j^*=\max\{-12,-2,-1,0\}=0$，$x_j^*=x_5$

(2) 令 $x_5=1$ 代入(*)得

$$\min\ z=3x_1-2x_2+5x_3+2x_4+3$$

$$-x_1-x_2+x_3+2x_4+S_1=2$$

$$-7x_1+3x_3-4x_4+S_2=1$$

$$11x_1-6x_2-3x_4+S_3=2$$

$$x_j=0,1,j=1,2,3,4;\ S_i\geqslant 0,i=1,2,3$$

$$S_2=(\bar{S}),A_2=\phi,B_2=\{1,2,3,4\},C_2=\phi$$

$$E_2=\{1,2,3,4\},F_2=\phi$$

实际上已获得一可行解 $x_1=x_2=x_3=x_4=0,x_5=1,z=3$.

$$v_1=\min(0,2+1)+\min(0,1+7)+\min(0,2-11)=-9$$

$$v_2=\min(0,2+1)+\min(0,1)+\min(0,2+6)=0$$

$$v_3=\min(0,2-1)+\min(0,1-3)+\min(0,2+3)=-2$$

$$v_4=\min(0,2-2)+\min(0,1+4)+\min(0,2+4)=0$$

$$v_j^*=\max\{-9,0,-2,0\}=0,\ x_j^*=x_2$$

(3) 令 $x_2=1,x_5=1$ 代入(*)得 $S_3=(\bar{S}_1,\bar{S}_2,\ldots)$，

$$\min\ z=3x_1+5x_3+2x_4+1$$

$$-x_1+x_3+2x_4+S_1=3$$

$$-7x_1+3x_3-4x_4+S_2=1$$

$$11x_1 \qquad 3x_4 \qquad S_3 = 8$$

$$x_i = 0, 1, i = 1, 3, 4; S_j \geq 0, j = 1, 2, 3.$$

由于 $z > z_{min} = 3$ 故应放弃前进；

(4) $S_4 = \{2, 5\}$，即

$$\min \quad z = 3x_1 + 5x_3 + 2x_4 - 5$$

$$-x_1 - x_3 + 2x_4 + S_1 = 2$$

$$-7x_1 + 3x_3 - 4x_4 - S_2 = -1$$

$$11x_1 - 3x_4 + S_3 = 2$$

$$x_i = 0, 1, S_j \geq 0, j = 1, 2, 3$$

由于 $z_4 > z_{min}$ 故放弃前进。

(5) $S_5 = \{5\}$

$$\min \quad z = 3x_1 + 2x_2 - 5x_3 - 2x_4$$

$$-x_1 - x_2 - x_3 + 2x_4 - S_1 = 1$$

$$-7x_1 + 3x_3 - 4x_4 + S_2 = -2$$

$$11x_1 - 6x_2 - 3x_4 + S_3 = -1$$

$$x_1, x_2, x_3, x_4 = 0, 1; S_1, S_2, S_3 \geq 0$$

$$A_5 = \{3\}, B_5 = \{1, 2, 4\}, C_5 = \phi, S_5 = \{1, 2, 4\}$$

$F_5 = \phi$

$v_1 = \min(0, 1 + 1) + \min(0, -2 + 7) + \min(0, -1 - 11) = -12$

$v_2 = \min(0, 1 + 1) + \min(0, -2) + \min(0, -1 + 6) = -2$

$v_4 = \min(0, 1 - 2) + \min(0, -2 - 4) + \min(0, -1 + 3) = -7$

$v^* = \max\{-12, -2, -7\} = -2 = v_2$，令 $x_2 = 1$

(6) $\min \quad z = 3x_1 + 2x_4 - 5x_3 + 2$

$$-x_1 - x_3 + 2x_4 + S_1 = 1$$

$$-7x_1 - 3x_4 + S_2 = 2$$

$$11x_1 - 3x_4 - S_3 = 2$$

$$x_1, x_3, x_4 = 0, 1; S_j \geq 0, j = 1, 2, 3$$

$S_6 = \{4, 5\}, A_6 = \{3\}, B_6 = \{1, 2\}, C_6 = \phi$

$D_6 = \{1, 2\}, F_6 = \phi$

$v_1 = \min(0, -1 + 1) + \min(0, 2 + 7) + \min(0, 2 - 11) = -9$

$v_2 = \min(0, -1 + 1) + \min(0, 2 + 5) = 0$

$v^* = \max\{-9, 0\} = 0 = v_2$

(7) 令 $x_2 = 1$

$$\min \quad z = 3x_1 + 5x_3 - 4$$

$$x_1 + x_3 - S_1 = 0$$

$$7x_1 + 3x_3 - S_2 = 2$$

$$11x_1 - S_3 = 8$$

$$x_j, x = 0, 1, S_i \geqslant 0, i = 1, 2, 3.$$

(8) $S_7 = \{1, 4, 5\}$。由于 $z > z_{\min}$ 故放弃前进

$S_8 = \{1, 4, 5\}$，即

$$\min \quad z = 2x_2 + 5x_3 - 2$$
$$x_2 + x_3 + S_1 = 1$$
$$3x_3 + S_2 = 2$$
$$-6x_3 - S_3 = 2$$
$$A_8 = \{3\}, B_8 = \{2\}, C_8 = \phi, E_8 = \{2\}, F_8 = \phi$$

(9) 令 $x_2 = 1, S_9 = \{2, 1, \bar{4}, 5\}$

$$\min \quad z = 5x_3 + 4$$
$$x_3 - S_1 = 0$$
$$5x_3 + S_2 = 2$$
$$S_3 = 8$$

显然 $x_3 = 0, x = x_4 = 0, x_2 = x_1 = 1$ 是可行解，但 $z > z_{\min}$，故应予放弃前进

(10) $S_{10} = \{2, 1, \bar{4}, 5\}$

$$\min \quad z = 5x_3 + 2$$
$$x_3 - S_1 = 1$$
$$3x_3 + S_2 = 2$$
$$S_3 = 2$$

无可行解，应后退。

(11) $S_{11} = \{4, 5\}$，

$$\min \quad z = 3x_1 + 2x_2 + 5x$$
$$x \quad x_2 + x + S_1 = 1$$
$$-7x_1 + 3x_2 - S_2 = 2$$
$$11x \quad 6x + S_3 =$$
$$x_i = 0, 1 \; i = 1, 2, 3. \; S_i \geqslant 0, j = 1, 2, 3.$$
$$A_{11} = \{3\}, B_{11} = \{1, 2\}, C_{11} = \phi, E_{11} = \{1, 2\}, F_{11} = \phi$$
$$v_1 = \min(0, 1 - 1) + \min(0, 2 + 7) = \min(0, 1 - 11) = -12$$
$$v_2 = \min(0, 1 - 1) + \min(0, 2) + \min(0, 1 + 6) = -2$$
$$v_l = \max(-12, -2) = -2 \quad v_7$$

(12) $S_{12} = \{\bar{2}, 4, 5\}$，

$$\min \quad z = 3x_1 + 5x_3 + 2$$
$$-x_1 + x_3 + S_1 = 2$$
$$-7x_1 - 2x_1 + S_2 = 2$$
$$11x \quad + S_3 = 7$$
$$x_1, x_3 = 0, 1 \; S_i \geqslant 0, i = 1, 2, 3.$$
$$A_{12} = \{3\}, B_{12} = \{1\}, z + c_l > z_{\min}$$ 应后退。

(13) $S_1=\{2,4,5\}$

$$\min\ z=3x_1+5x_2$$

$$x_1\quad x_1+S_1=1$$

$$-2x_1-3x_2+S_2=-2$$

$$11x_1\quad +S_3=-1$$

$$x_1,x_1=0,1\quad S_j\geqslant 0,\ j=1,2,3.$$

$A_{11}=\{3\},B_{11}=\{\ \},F_1\neq\phi$,故后退。

(14) $S_{11}\ \{4,5\}\to\{5\}\to\{\phi\}$

搜索完毕，最优解为

$$x_1\quad x_2\quad x_3\quad z\quad 0,\ 1,\ 5$$

搜索全过程见图20.2.6

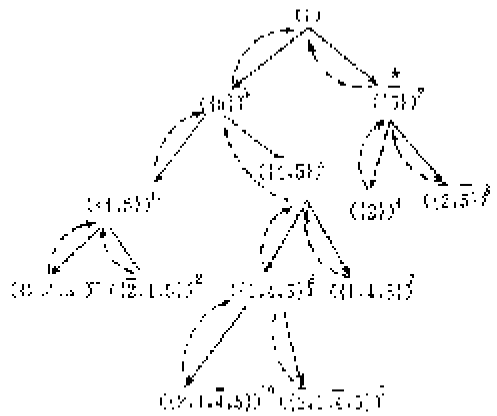


图 20.2.6

图中每个节点上注的数是栈S的状态，比如(5)表示$x_5=1$的状态，{2,5}表示$x_2=x_5=1$，右上角的数表示搜索过程的顺序，*表示最优解的点。

20.3 分支定界法在解整数规划中的应用

分支定界是搜索技术中应用较广的一种，下面通过一个例子介绍分支定界法在整数规划问题上求解的基本思想。

例

$$\max z=x_1+x_2,$$

$$\begin{cases}6x_1+5x_2\leqslant 13,\\ -2x_1+x_2\leqslant 1,\end{cases}\qquad (20.3.1)$$

$$x_1,x_2\geqslant 0\ \text{整数}$$

将问题作为线性规划求解，如图20.3.1所示。

易解可得

$$x_1=\frac{56}{33},x_2=\frac{145}{33},z=\frac{201}{33}$$

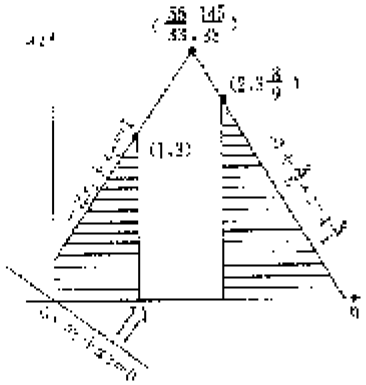


图 20.3

用
$$\begin{pmatrix} x_1\geq 0, & x_2\geq 0 \\ 56/33, & 145/33 \end{pmatrix}$$
表示线性规划问题(20.3.1)的解：

$$x_1=56/33,\ x_2=145/33,\ z=201/33$$

$z=201/33$ 是目标函数的上界，由于

$$1<x_1<2$$

故所求的最优解可能在域

$$0\leq x_1\leq 1,\ x_2\geq 0$$

或域

$$x_1\geq 2,\ x_2\geq 0$$

上取得。

$$\max\ z=x_1+x_2$$
$$5x_1+3x_2\leq 13$$
$$-2x_1+x_2\leq 1$$
$$0\leq x_1\leq 1,\ x_2\geq 0$$

的解为 $x_1=1$，$x_2=3$，目标函数为 [illegible]；

$$\max z=x_1+x_2$$
$$5x_1+3x_2\leq 13$$
$$-2x_1+x_2\leq 1$$
$$x_1\geq 2,\ x_2\geq 0$$

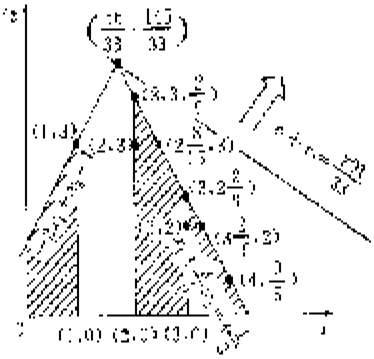


图 20.3.2

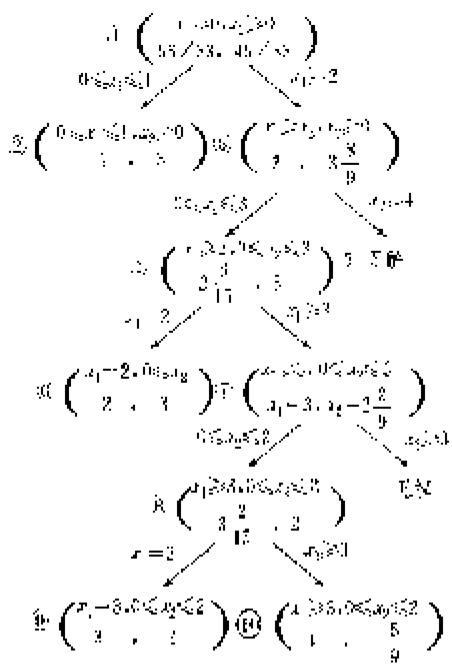


图 20.3.3

的解为 $x_1=2, x_2=3\frac{8}{9}$，目标函数 $z=5\frac{8}{9}$。分号，表以

$$\left\{\begin{matrix} 0 \leqslant x_1 \leqslant 1, & x_2 \geqslant 0 \\ 1, & 3 \end{matrix}\right\}$$

$$\left\{\begin{matrix} x_1 \geqslant 2, & x_2 \geqslant 0 \\ 2, & 3\frac{8}{9} \end{matrix}\right\}$$

具体数据图象可见图 20.3.2

用分支定界法求解过程可描述如图 20.3.3，图中( )完上角，里右数是搜索的顺序数。

习 题

用分支定界法求解下列整数规划问题

$$\max \quad z = 2x_1 + 3x_2$$

$$x_1 + \frac{3}{5}x_2 \leqslant 4\frac{1}{5}$$

$$-2x_1 - x_2 \leqslant 1$$

$$x_1, x_2 \geqslant 0$$

用 DFS 法求下列 0-1 规划问题的解

$$\min \quad z = 2x_1 + 5x_2 + 6x_3 + 8x_4 + 12x_5$$

$$3x_1 - x_2 + x_3 + x_4 - 5x_5 \leqslant -5$$

$$-6x_1 + 3x_2 - 2x_3 - x_5 \leqslant -2$$

$$2x_1 - 9x_2 - 3x_3 + 2x_4 - 3x_5 \leqslant -4$$

$$x_i \in \{0,1\}, i = 1,2,\cdots,5.$$

$$\min \quad z = 3x_1 + 2x_2 + 5x_3 + 2x_4 + 3x_5$$

$$-x_1 - x_2 - x_3 + 2x_4 - x_5 \leqslant 1$$

$$7x_1 - 3x_3 + 4x_4 - 3x_5 \leqslant -2$$

$$11x_1 - 6x_2 - 3x_4 - 3x_5 \leqslant -1$$

$$x_i \in \{0,1\}, i = 1,2,\cdots,5.$$

$$\min \quad z = 5x_1 + 7x_2 + 10x_3 + 3x_4 + x_5$$

$$x_1 + 3x_2 - 5x_3 + x_4 + 4x_5 \leqslant 0$$

$$2x_1 + 6x_2 - 3x_3 + 2x_4 + 2x_5 \geqslant 4$$

$$x_2 - 2x_3 - x_4 + x_5 \geqslant 2$$

$$x_1, x_2, \cdots, x_5 \in \{0,1\}$$

第21章 串 匹 配

21.1 概 述

有时候计算机处理的数据是以一种符号序列的形式出现，例如在文本编辑、图象处理中常会碰到这种类型的数据，通常称它们为串(字符串)。

串上的一个基本问题是模式匹配。所谓模式匹配是指已知一个长度为 n 的文本字符串：

$$a = a_1a_2\cdots a_n$$

和一个长度为 $m(\leq n)$ 的模式字符串：

$$b = b_1b_2\cdots b_m$$

问 a 中是否存在长度为 m 的子串

$$a_{i+1}a_{i+2}\cdots a_{i+m}$$

使得：

$$a_{i+j} = b_j,\ j = 1,2,\cdots,m,\ i \leq n - m。$$

以后通常称 a 为文本，b 为模式。

当 n 不大时，我们可采用一种最简单的算法，称之为强行搜索算法，叙述如下：

(1) $j \leftarrow 1, k \leftarrow 1$。

(2) 若 $(j \leq n)$ 且 $(k \leq m)$，则转(3)，否则，转(4)。

(3) 若 $a(j) = b(k)$，则【$j \leftarrow j+1, k \leftarrow k+1$】，

否则，【$j \leftarrow j-k+2, k \leftarrow 1$】。

转(2)。

(4) 若 $k > m$，则【输出有关匹配的信息，结束】

否则【输出匹配失败，结束】。

强行搜索算法的思想是将模式 b 和文本 a 从左至右逐个搜索，若在某一位匹配失败，则将模式 b 向右移动一位，再从模式 b 的第一位开始搜索。其下例清楚说明了这一点：

$$a = babcbabcaccaabb$$

$$b = babcac$$

强行搜索过程如下图所示("×"表示在该位匹配失败)：

```
a:    b a b c b a b c a c c a a b b
j=1   b a b c × a c
j=2     b × a b c a c
j=3       b a × b c a c
j=4         b × a b c a c
j=5           b a b c a c
```

在 $j=5$ 时搜索成功。在最坏情况下，要执行 $(n-m+1)*m$ 次字符的比较，因此其

时间复杂度为$O(mn)$。该算法没能进一步改善，但它是最基本的。下面我们将介绍KMP算法、BM算法和KR算法，它们在时间复杂性上都有所改善。

21.2 KMP（Knuth-Morris-Pratt）算法

KMP算法指的是由Knuth、Morris、Pratt合作的算法，这是他们三人各自独立得到的结果。

先通过例子对KMP算法作非形式化说明。假定文本a和模式b分别是

$$a = bcabcabcabbcabcabcabcab$$

$$b = bcabcabcabc$$

强行搜索没有充分利用模式b和已匹配部分的信息，例如上面例子a对b的匹配到第7个字符才失败，而强行搜索法只能以模式右移一个字符，从第一个字符开始，毫无疑问，我们有可能加快搜索的速度。

```
a = b c a b c a b c a b b c a b c a b c a b c a b
          b c a b c a b c a b c
```

右移3个字符后，可直接从原来匹配失败的字符位开始继续搜索（注意：不是从头开始），结果仍失败，再右移3个字符，即

```
a =   b c a b c a b c a b b c a b c a b c a b c a b
b =               b c a b c a b c a b c
```

“↕”表示上下匹配成功

再将b右移5个字符得：

```
a = b c a b c a b c a b b c a b c a b c a b c a b
b =                   b c a b c a b c a b c
```

在第2个字符匹配失败，b右移一个字符得：

```
A = b c a b c a b c a b b c a b c a b c a b c a b
B =                     b c a b c a b c a b c
```

匹配成功。从上可见模式右移几个字符与模式的结构及匹配失败的位置有关。下面来看模式串移动的依据。

假定匹配在b的第j位(b_j)处匹配失败，应将b右移$j-k$个字符，使得$b_1, b_2, \cdots, b_{k-1}$分别和$b_{j-k+1}, b_{j-k+2}, \cdots, b_{j-1}$一致，使$b_k$正好和原来匹配失败的$b_j$对齐，而且匹配可以从这一位开始继续搜索下去。换句话说，从b_{j-k+1}开始到b_{j-1}为止的$k-1$个字符和$b_1 b_2 \cdots b_{k-1}$完全一致，并且k必须是满足这一条件的最大值，否则有可能漏掉已成功的匹配。记k的最大值为$K(j)$，$j=1,2,\cdots,m$。若能事先对模式串进行预处理，求得每一$K(j)$，则匹配时利用$K(j)$可节省匹配的比较次数。

$b_1 b_2 \cdots\cdots b_{i-k+1} \cdots b_{i-1}$ ……

……向右推行 ……

基于以上的原则不难求得模式的每一位的 $K(j)$，以 $B=ababbaaba$ 为例

表 21.2.1

j	$K(j)$	
2	1	a b a b b a a b a
		a b a b b a a b a
3	1	a b a b b a a b a
		a b a b b a a b a
4	2	a b a b b a a b a
		a b a b b a a b a
5	3	a b a b b a a b a
		a b a b b a a b a
6	1	a b a b b a a b a
		a b a b b a a b a
7	2	a b a b b a a b a
		a b a b b a a b a
8	2	a b a b b a a b a
		a b a b b a a b a
9	3	a b a b b a a b a
		a b a b b a a b a

表21.2.1中的×号右边一位指出是匹配失败位置的标志，即在此位匹配失败，下面一行表示 $K(j)$ 的标志，则将模式 B 向右移动到表中所给出的位置，B 实际上向右移动的字符数为 $j-K(j)$。例 $j=5$，则 $K(5)=3$，即 B 向右移动2位。上述匹配过程可用一自动机来描述，该例的KMP算法可用图21.2.1表示。图21.2.1是自动机，每一结点表示一种状态，每点都有两条出弧，只有两条弧上标有字母，分别注以 a 或 b，表示当读入 a 或 b 后状态的改变。以状态6为例，6表示已有5位匹配成功，这时已完成匹配 $ababb$，若读入 a，则完成 $ababba$ 的匹配进入状态7；若读入 b，则成为 $ababbb$，此时下一个状态只匹配了 a、b 两字符，即将 B 向右移动3个字符，见表21.2.1中 $j=6$，$K=1$ 是匹配完毕状态。

总之，每一种状态都有前进和后退两种情况。所谓前进，即是维持针对状态的移动而言，比如从状态2到状态3的转移，便是前进；否则便是后退。后退就意味着匹配失败。后退指出匹配失败后应退到哪个状态。退回状态 j 相应于将 B 向右移动到合适的位置。从图21.2.1可知状态 j 表示 $b_1 b_2 \cdots b_j$ 均已匹配成功的字符。因此，算法的过程可以分成以下两步来进行：

（1）根据模式串构造上述自动机。

（2）从状态0开始，逐个读入文本串的字符，若能进入状态9，则表明一次匹配成功。KMP算法可以减少比较的次数。具体算法：

（1）$i \leftarrow 1, j \leftarrow 1$。

（2）若 $j > m$ 或 $i > n$ 转（4），否则转（3）。

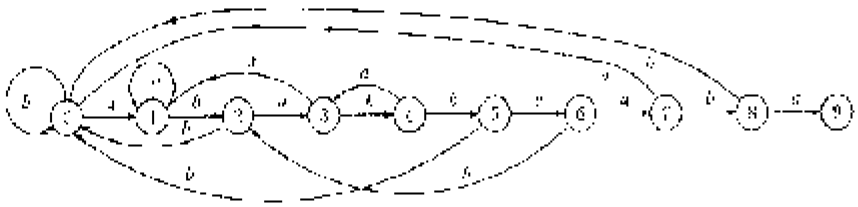


图 21.2.1

(3) 若 $j=0$ 或 $a(i)=b(j)$，则【$i\leftarrow i+1$，$j\leftarrow j+1$，转(2)】，

否则，【$j\leftarrow K(j)$，转(2)】。

(4) 若 $j>m$，则输出 $i-m$，表示从 a_{i-m} 开始匹配成功；

否则，表明匹配失败。

$K(j)$ 如何确定？从上面知道，其实它是模式串 b 自己进行匹配，算法如下：

(1) $i\leftarrow 1$，$j\leftarrow 0$，$K(1)\leftarrow 0$。

(2) 若 $i=m$，则停止；否则，转(3)。

(3) 若 $j=0$ 或 $b(i)=b(j)$，

则【$i\leftarrow i+1$，$j\leftarrow j+1$，$K(i)\leftarrow j$，转(2)】，

否则，【$j\leftarrow K(j)$，转(3)】。

(4) 结束

在最坏情况下，KMP 算法的时间复杂度为 $O(m+n)$，计算 $K(j)$ 的复杂度为 $O(m)$，[illegible]大大地[illegible]了。

21.3 BM(Boyer Moore) 算法

BM 算法和 KMP 算法的差别是对模式串的扫描方式自左至右变成自右至左。另一个差别是考虑文本中可能出现的字符在模式串中的位置。

例如设 $b=babaabab$，与文本 a 进行匹配时，比较方式由右向左进行，即先对第8位，然后第7位、第6位……。若通过自右向左比较第8位、第7位……，但在第7位匹配失败，则可将模式 b 向右移动[illegible]位，然后从文本的第[illegible]位开始下一次匹配。这是由于模式 b 前面不出现[illegible]的缘故。

[illegible] BM 算法的基本思想：

(1) 匹配自右向左进行。

(2) 若匹配失败发生在 $b_j\neq a_i$，且 a_i 不出现在模式 b 中，则将模式右移直到 b 的第一位与 [illegible] 对齐。若 a_i 在 b 中有若干地方出现，则应移动 $j-\max\{k\,|\,b_k=a_i\}$。

(3) 若模式 b 右部有 K 位和文本 a 中一致的部分，有一部分在 b 中其它部分出现，则可将 b 向右移动，直接使这部分对齐，当要求这一致部分尽可能的大。

以 $b=$ [illegible] 为例

表 21.3.1

j	$r(j)$																			
2	1	b	a	b	b	b	a	b	a	b	b									
			b	a	b	b	b	a	b	a	b	b								
3	5	b	a	b	b	b	a	b	a	b	b									
							b	a	b	b	b	a	b	a	b	b				
4	9	b	a	b	b	b	a	b	a	b	b									
											b	a	b	b	b	a	b	a	b	b
5	6	b	a	b	b	b	a	b	a	b	b									
								b	a	b	b	b	a	b	a	b	b			
6	6	b	a	b	b	b	a	b	a	b	b									
								b	a	b	b	b	a	b	a	b	b			
7	6	b	a	b	b	b	a	b	a	b	b									
								b	a	b	b	b	a	b	a	b	b			
8	6	b	a	b	b	b	a	b	a	b	b									
								b	a	b	b	b	a	b	a	b	b			
9	6	b	a	b	b	b	a	b	a	b	b									
								b	a	b	b	b	a	b	a	b	b			
10	6	b	a	b	b	b	a	b	a	b	b									
								b	a	b	b	b	a	b	a	b	b			

表21.3.1中$r(j)$表示当匹配从模式b的右端向左第j个字符发生匹配失败时，模式b应右移的字符数。移动后，比较仍然从模式b的新位置的右端开始。表中上下重叠部分便是最大的一致部分。以$j=6$这一行为例，重叠部分为babb。在上面一行表示b右面的4个字符的内容，下面一行表示babb也出现在b的前面，因babb在b的前面出现，但它和a的匹配部分在$j=6$出现不一致，故不能再扩大了，即最大的一致部分就是babb。

21.4 RK (Rabin-Karp)算法

在串匹配的简单算法中，把文本每m个字符构成的字符段作为一个字段，和模式进行匹配检查。但是如果能对一个长度为m的字符串赋以一个Hash函数，那么显然只有那些与模式具有相同Hash函数值的文本中的字符段才有可能与模式匹配，这是必要条件，而没有必要去考虑文本中所有长度为m的字段，因而大大提高了串匹配算法速度。RK算法就是基于这一思想，与KMP、BM算法迥然不同。

假设文本和模式中出现的字符集为Σ，设$S=|\Sigma|$，则可将长为m的字段数值化，例如将模式b数值化，看作是S进制数，即引入一个转换函数：

对应于

$$b = b_1b_2\cdots b_{m-1}b_m$$

有

$$b_w = b_{m-1}S + b_{m-2}S^2 + \cdots + b_1S^{m-1}$$

取其 Hash 函数：

$$B = b_m + b_{m-1}S + b_{m-2}S^2 + \cdots + b_1S^{m-1} \mod q$$
$$\equiv b_m + S(b_{m-1} + S(b_{m-2} + \cdots + S(b_2 + Sb_1)\cdots)) \mod q$$

q 是一素数。

文本 $a = a_1a_2\cdots a_n$，对于其中长度为 m 的一段

$$a_{k+1}a_{k+2}\cdots a_{k+m}$$

有

$$A_k = a_{k+m} + S(a_{k+m-1} + S(a_{k+m-2} + \cdots + S(a_{k+2} + Sa_{k+1})\cdots)) \mod q$$
$$k = 1,2,\cdots,n-m.$$

显然下列递推关系式成立

$$A_{k+1} = S(A_k - S^{m-1}a_{k+1}) + a_{k+m+1} \mod q$$

$k = 1,2,\cdots,n-m$.（其中 $A_0 = a_1S^{m-1} + a_2S^{m-2} + \cdots + a_{m-1}S + a_m$）

利用上式逐步计算 $A_1, A_2, \cdots, A_{n-m}$ 可在 $O(n+m)$ 时间内完成。$S^{m-1} \bmod q$ 应预先处理这一计算环节，以免反复计算。

匹配算法的基本想法是：如果文本的一个长度为 m 的串与模式具有相同 Hash 函数值，则进行匹配检查，否则，以及在匹配失败的情况下，继续计算下一个字符段的 Hash 函数值。

读者自己不难写出 RK 算法，这里略去。

对于 RK 算法，如果不计执行匹配检查的时间，则其剩余部分执行时间是 $O(n+m)$。不过若计算执行匹配检查的时间在内，则理论上，RK 算法的复杂度为 $O(mn)$。但我们适当选取 Hash 函数的 q 值，使得 mod 函数计算在计算机中既高效可执行，而冲突发生可能性又极小，从而该算法的实际执行时间为 $O(m+n)$。

Rabin-Karp 法可推广到其它有关问题，如二维图象匹配问题。

习　题

1. 试给出 BM 算法中 $\delta(j)$ 的算法。
2. 试给出 BM 算法。
3. $a = 31415926535897932$，$b = 26$，$q = 13$。
   试用 RK 法找出匹配。
4. 试讨论将 RK 法推广到讨论在 $n\times n$ 的文本 a 中寻找 $m\times m$ 的模式的问题。
5. 试用 KMP 及 BM 算法讨论问题 3 的匹配。
6. 编写可以推广的 Rabin-Karp 算法求二维的匹配。设文本为 n×n 字符阵，模式为 m×m 字符阵。
7. 已知模式为 1 0 1 0 0 1 0 0 1 0 1 0 0 1 0 1 0 0 1 0 0
   讨论 Knuth-Morris-Pratt 算法。
8. 讨论上题模式的 Boyer-Moore 算法。
9. 已知模式为 $a\ b\ a\ a\ b\ a\ c\ a$，试作出串匹配 KMP 算法的状态转移图(即有限自动

机)。

10. 用 Boyer-Moore 算法讨论上一问题。

11. 已知模式为 $a\ b\ a\ a\ b\ a$，试绘出 KMP 法的状态转移图(有限自动机)。

12. 用 Boyer-Moore 法讨论上面问题。

第22章 概率算法

前面所讨论的算法在算法的每一步都明确指定下一步该如何进行。本章所讨论的概率算法容许在算法执行的过程中可随机选择下一个步骤，这和传统的算法思想迥然不同。它的基本思想是将决定算法的随机性表现为偶然性，而算法的判定使得其成功的可能性较大，从而求解问题。许多复杂度较大的问题可从概率算法中找到一个较满意的近似解。

22.1 概率算法举例

在介绍概率算法之前，先举一个典型的概率统计问题作为先导，了解一下概率与统计的基本思想。

生日问题：有多少个人在一起使得其中至少有两个人生日相同的机会超过没有人生日相同的机会，即有相同生日的概率超过1/2。

若一年以365天计，366人在一起至少有两个人生日相同。可以按以下办法求出生日不同。假设所要求的人数为k，用$1,2,\cdots,k$表示这k个人，并假定生日是均匀分布的，故任何两个人i和j生日相同的概率为$1/365$。至少有两个人生日相同的概率等于1减去k个人的生日都不相同的概率。故现在转而讨论k个人生日都不相同的概率。

令D_i表示$1,2,\cdots,i$都和$i+1$生日不相同的事件，所有这k个人生日都不相同的事件记为B_k，则

$$B_k = D_1 \cap D_2 \cap \cdots \cap D_{k-1} = \bigcap_{i=1}^{k-1} D_i$$

$$B_k = \bigcap_{i=1}^{k-1} D_i$$

$$B_k = B_{k-1} \cap A_k,$$

这里A_k表示$1,2,\cdots,k$生日都不相同，$1,2,\cdots,k-1$中每个都和k的生日不相同。故

$$P(B_k) = P(B_{k-1})P(A_{k-1} \mid B_{k-1})$$

同理

$$P(B_{k-1}) = P(B_{k-2})P(A_{k-2} \mid B_{k-2})$$

一般地，

$$P(B_i) = P(B_{i-1})P(A_{i-1} \mid B_{i-1})$$

$$i = 2,3,\cdots,k$$

$$P(B_1) = 1$$

$$\therefore \quad P(B_k) = P(B_1)P(A_1 \mid B_1)P(A_2 \mid B_2)\cdots P(A_{k-1} \mid B_{k-1})$$

$P(A_1 \mid B_1)$是1和2生日不同的概率，它等于$\frac{365-1}{365} = \frac{364}{365}$，$P(A_2 \mid B_2)$是1和2生日不同，且和3生日也不同的概率，等于$\frac{365-2}{365} = \frac{363}{365}$。

一般地有

$$P(D_i|\overline{D_{i-1}}) = \frac{365-i}{365}$$

所以

$$P(D_k) = 1 \cdot \left(1 - \frac{1}{365}\right)\left(1 - \frac{2}{365}\right)\cdots\left(1 - \frac{k-1}{365}\right)$$

由 $e^x \geqslant 1+x$ 得

$$P(D_k) \leqslant e^{-\frac{1}{365}} \cdot e^{-\frac{2}{365}} \cdots e^{-\frac{k-1}{365}}$$

$$= e^{-\sum_{i=1}^{k-1} \frac{i}{365}} = e^{-\frac{k(k-1)}{730}}$$

若要求

$$e^{-\frac{k(k-1)}{730}} \leqslant \frac{1}{2}$$

则有

$$\frac{k(k-1)}{730} \geqslant \ln 2$$

$$k(k-1) \geqslant 730\ln 2$$

$$k \geqslant 23$$

从这题可知，只需23人就能使两个人生日相同的概率大于没有人生日相同的概率。

下面介绍概率算法，它作为数学方法已有较长的历史，这一节主要通过实例来说明它的想法。

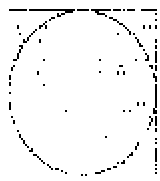


图 22.1.1

例1 设有一半径为 r 的圆及其外切四边形，如图22.1.1所示。向该正方形随机地投掷 n 个点，若落入圆内的点数为 k，试用此近似地计算 π 值。分析：

$$\frac{k}{n} \approx \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

所以

$$\pi \approx \frac{4k}{n}$$

这里考虑投入的点在正方形上均匀分布，因而正方形内

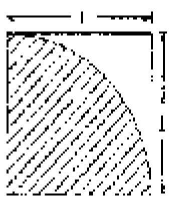


图 22.1.2

任一点落在圆上的概率为 $\frac{\pi r^2}{4r^2} = \frac{\pi}{4}$，所以当 n 足够大时，k 与 n 之比就逼近这一概率，即为 $\frac{\pi}{4}$，从而 $\pi \approx \frac{4k}{n}$。其算法描述如下：(该算法与上面描述稍有区别，它根据图22.1.2计算，这么做是为了方便描述，其落的概率仍为 $\frac{\pi}{4}$)。

(1) 初始化：$K \leftarrow 0$，$i \leftarrow 0$。

(2) 独立地产生一对在[0,1]区间上均匀分布的随机数 (x,y)，$i \leftarrow i+1$

(3) 若 $x^2+y^2 \leqslant 1$，则作【$K \leftarrow K+1$，转(4)】 否则，转(2)

(4) 若 $i\leqslant n$，则转(2)。

否则，作【$P_s \leftarrow 4K/n$，输出 P_s，结束】。

产生区间[0,1]上均匀分布的随机数，是可行的，至于产生[a,b]之间均匀分布的随机整数与它类似，读者可自己思考。另外，上述算法求得 π 的精度依赖于 n 的大小，当 n 愈大时，可以想像计算得到的值越精确的可能性越大。

例2 积分计算问题。设函数 $f(x)$：$0\leqslant f(x)\leqslant 1$，$0\leqslant x\leqslant 1$，计算

$$I=\int_0^1 f(x)\mathrm{d}x$$

上式相当于图 22.1.3 中阴影部分的面积。

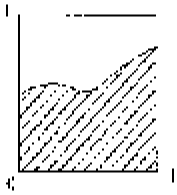


图 22.1.3

假定有边长为 1 的正方形如图 22.1.3，向该区域随机投掷 n 个点，设落在阴影部分的点的数目为 K，则

$$I\approx K/n$$

算法如下：

(1) $k\leftarrow 0$，$i\leftarrow 0$ /*初始化*/

(2) 独立地产生一分别在[0,1]区间均匀分布的随机数对 (x_i, y_i)，$i\leftarrow i+1$，

(3) 若 $y_i\leqslant f(x_i)$，则【$k\leftarrow k+1$，转(4)】。

否则，转(2)。

(4) 若 $i\leqslant n$，则转(2)。

否则，【$I\leftarrow k/n$，输出 I，结束】。

特别当

$$f(x)=\sqrt{1-x^2}$$

则有

$$I=\int_0^1\sqrt{1-x^2}\,\mathrm{d}x=\frac{\pi}{4}$$

$$\pi=4\cdot I$$

也可以通过它计算 π 的值。

例3 偏微分方程中的拉普拉斯方程狄里克莱问题是一极为重要的问题，因为许多数学和物理问题都导致解这样的问题。

以二维的拉普拉斯方程狄里克莱问题为例，即求二元函数 $u(x,y)$ 使满足

$$\begin{cases}\dfrac{\partial^2 u}{\partial x^2}+\dfrac{\partial^2 u}{\partial y^2}=0 & (x,y)\in D/\partial D\\ u|_\Gamma=f\end{cases}\qquad(22.1.1)$$

即 $u(x,y)$ 在域上满足 $\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0$，在 D 的边界 ∂D 即 Γ 上的值已知，求 $u(x,y)$。见图 22.1.4 和图 22.1.5。

狄里克莱问题的解存在而且唯一，但求它的解析解是十分困难的，通常是将它离散化，即将区域 D 网格化，对 $\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0$，对应于差分方程：

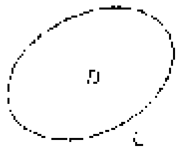


图 22.1.4

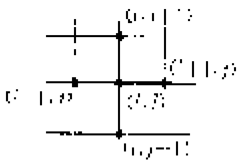


图 22.1.5

$$u_{ij} = \frac{1}{4}[u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}], \quad (i,j) \in D \tag{22.1.2}$$

D 域内每一网格点都对应这样的一个方程，未知数的数目和方程的数目一样。于是解拉普拉斯方程(22.1.1)变为解代数方程组(22.1.2)。

为使用概率算法求解，引进一随机变量 V_{ij}，设想有一醉汉从十字路口 (i,j) 点出发到处游荡，每经一个十字路口，它向东、西、南、北四个方向行走的概率都为 $\frac{1}{4}$，最终都将碰到某边界点，想象该点的边界值为该人被罚款的数目，记为 V_{ij}，我们知道 V_{ij} 是一个随机变量，则有

$$V_{ij} = \frac{1}{4}[V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1}]$$

和式(22.1.2)形式一样，设 V_{ij} 的数学期望为 m_{ij}，则有：

$$m_{ij} = \frac{1}{4}[m_{i+1,j} + m_{i-1,j} + m_{i,j+1} + m_{i,j-1}]$$

而且在边界上 m_{ij} 的值就是该点的边界值，m_{ij} 为(22.1.1)的解。概率求解是这样的：

对点 (i,j)，从 (i,j) 出发通过模拟"醉汉"行动 n 次，即"随机"游荡到边界上获得的 n 个结果：

$$V_{ij}^{(1)}, V_{ij}^{(2)}, \cdots, V_{ij}^{(n)},$$

则 $V_{ij} = \frac{1}{n}\sum_{k=1}^{n} V_{ij}^{(k)}$ 给出(22.1.2)的近似解。

22.2 随机数产生法

随机数在概率算法中有着重要的应用，真正的随机数在计算机上是不可能产生的，实际上都是一定程度上随机的，即伪随机数。

最简单的伪随机数产生器使用线性同余的办法来产生，即

$$a_{n+1} = b a_n + c \mod m$$

$$a_0 = d$$

其中 $b \geq 0, c \geq 0, d \leq m$，$d$ 称为种子。因为当 b, c, m 给定后，d 不同将产生不同的伪随机序列。如何选取 b, c 和 m 使得产生的随机数序列，其"随机"性能比较好？这是很难的理论问题。随机数的理论比较复杂，研究它已超出本书范围。但从直观来看，m 应取得充分大，因为由线性同余法产生的随机数的范围顶多是 $0, 1, \cdots, m-1$。另外应取 $\gcd(m, b) = 1$，通常

取 m 为素数。

当然也存在其它产生随机数的方法，在此不作讨论。

22.3 素数的概率判定算法

判定所给自然数 n 是否是素数，不仅在理论上有重要意义，而且由于密码学的发展，素数判定问题成为实际中十分有价值的研究课题。在这一节，将介绍一种素数的概率测试方法，该方法具有着很高的理论和应用价值，可以说是概率算法的重要代表。

素数的分布是稀疏的，据研究小于 10^4 的素数个数有 1229 个，小于 10^8 的素数个数有 5761455 个，小于 10^{12} 的素数个数有 37607912018 个。

设 $\pi(x)$ 为小于或等于 x 的全部素数个数，则

$$\lim_{x\to\infty}\pi(x)\Big/\frac{x}{\ln x}=1$$

也就是 $$\pi(x)\approx\frac{x}{\ln x}$$

和素数有关的重要定理有 Wilson 定理和 Fermat 定理。

定理（Wilson） n 是素数的充要条件是

$$(n-1)!\equiv -1 \bmod n$$

证明 必要性证明。

n 是素数，所以对 $1,2,\cdots,n-1$ 中的每一个数 a，必存在 a'，使得

$$aa'\equiv a'a\equiv 1 \bmod n$$

其中 1 的逆元素为 1，$n-1$ 的逆元素为 $n-1$，即

$$1\cdot 1\equiv 1 \bmod n,\ (n-1)(n-1)\equiv 1 \bmod n$$

除了 1 和 $n-1$ 外，其余 $n-3$ 个数有 $a\neq a'$，所以

$$(n-1)!\equiv(n-1)\equiv -1 \bmod n$$

充分性证明。

设 $(n-1)!\equiv -1 \bmod n$，但 n 不是素数，即 $a\mid n$，这里 $1<a<n$，则 $a\mid(n-1)!$，同样有

$$a\mid(n-1)!+1$$

即有

$$a\mid[(n-1)!+1-(n-1)!]$$

与 $a>1$ 矛盾。

定理得证明。

Wilson 定理在判定是否为素数的问题上有很高的理论价值，但问题在于 n 十分大时，计算量十分可观，而且几乎不可能。关于素数还有一个定理，即 Fermat 定理。

定理 若 p 是素数，则对于任意的整数 a，$p\nmid a$，有：

$$a^{p-1}\equiv 1 \bmod p$$

我们知道计算 a^n 不需要 n 次乘法，只需作 $O(\log n)$ 次乘法即可。Fermat 定理给出

作为判定素数的必要条件，从而不满足该条件的数肯定不是素数，但用这个条件判定素数就有可能出错。有人做过试验，n 从 1 到 1000，计算 $2^{n-1} \bmod n$，发现 $2^{n-1} \equiv 1 \bmod n$ 成立，但 n 不是素数的只有 22 个，即比率很小。

令 $n-1=2^t m$，其中 t 是非负整数，m 是正奇数，若 $b^m \equiv 1 \bmod n$ 或 $b^{2^j m} \equiv -1 \bmod n$，$0 \le j \le t-1$，则称 n 通过了以 b 为基的 Miller 测试。

定理 若 n 是素数，b 是正整数，且 $n \nmid b$，则 n 必然通过以 b 为基的 *Miller* 测试。

证 令

$$S_k = b^{2^{t-k} m} \bmod n = b^{(n-1)/2^k} \bmod n, k = 0, 1, \cdots, t$$

其中 $S_0 = b^{n-1} \bmod n$，$S_t = b^m \bmod n$。而且有 $(S_k - 1) \equiv (S_{k+1}-1)(S_{k+1}+1) \bmod n$ 中 $0 \le k \le t-1$。

若 n 是素数，则由 Fermat 定理 $S_0 = b^{n-1} \equiv 1 \bmod n$。故 $n | S_0 - 1$ 即 $n|(S_1-1)(S_1+1)$，由于 n 为素数，则 $n|S_1-1$ 或 $n|S_1+1$，若 $n|S_1+1$，则 $S_1 \equiv -1 \bmod n$ 即 n 通过了以 b 为基的 miller 测试，否则有 $n|S_1-1$，从而推出 $n|(S_2-1)(S_2+1)$，如此下去必定有存在某个 j 使得 $S_j \equiv -1 \bmod n (1 \le j \le t)$ 或是对所有 $j(0 \le j \le t)$ 有 $S_j \equiv 1 \bmod n$ 即 $S_t \equiv 1 \bmod n$。无论哪种情况，都有结论：n 通过了以 b 为基的 Miller 测试。通过 Miller 测试必通过 Fermat 定理，利用 Miller 测试可减少计算量，如果说利用 Fermat 定理作为判定一个数是否素数的必要条件，非充分条件，满足 Fermat 定理断定为素数，将用 miller 测试出错概率，特别是经过了多次 miller 测试后仍然未被排除掉的数不是素数的概率将会降低，下面这一关于 miller 测试的重要结果实证了上述结论的正确性。

定理 若 n 是奇合数，设 $1 \le b \le n-1$ 为任取的数，则 n 通过以 b 为基的 miller 测试的概率 $\le \frac{1}{4}$。

定理的证明从略。

基于上述的结论，若 n 是正整数，选 k 个小于 n 的正整数，以这 k 个数作为基进行 miller 测试，若 n 是合数，k 次测试全部通过的概率不超过 $\left(\frac{1}{4}\right)^k$。例如 $k=100$，n 是合数，但测试获得全部通过的概率不超过 $\left(\frac{1}{4}\right)^{100} \approx 6.22 \times 10^{-61}$，这是一个很小的数，从实践上看，一个事件发生的可能性这么小，可以说 miller 测试是完全可以信赖的。

习 题

试利用素数性质的测试法判定 [illegible] 是否素数？

第23章 并行算法

由于数值天气预报、计算空气动力学、空间技术以及核试验模拟等问题的实际需要，从各种途径设法提高计算机的速度，一直是计算机发展中的研究课题。它们计算量巨大，有的还要求实时，比如天气预报和空间技术。60、70年代电子技术的高速发展，使得计算机元件的运算速度获得很大的提高，现在已接近于电子传输的物理极限，进一步提高元件运算速度似乎将有待于新的技术的出现。在70年代初，计算机科学家们认识到，传统计算机的串行结构是阻碍运算速度提高的关键因素，从而找到了一条新路，开始并行算法的研究。

并行算法的思想在前面已接触过，比如流水线技术，特别是卫星图象处理等广泛应用的FFT算法和用于分类的专用装置，其中尤其是FFT专用机的应用极为成功。近几年来，并行机的研究是非常引人入胜的一个领域，但并行计算机的根本在于并行算法的设计，所以并行算法的研究工作是计算机科学家和数学工作者感兴趣的重要问题。特别应该提到的，计算机网络化，利用网络上各个点的资源联合进行分布式计算，是并行计算发展一个新的值得注目的领域。

由于大量算法是串行的，人脑的思维也是串行的，所以并行算法并不能轻而易举地获得，需要仔细研究它的方法。

23.1 并行计算机和并行算法的基本概念

并行计算机使用与一般计算机不同的随机存储器，是一种"并行的随机存储器"，记为PRAM为英文"Parallel Random-Acces Machine"缩写，如图23.1-1所示。处理元件$P_1, P_2, \cdots, P_n$可同时对公共的随机存储器进行读写，可以并行地完成算术的或逻辑的运算。因为任何两个处理元件之间通信都需通过公共的RAM，所以随着处理器件的增加，存取方式将趋于复杂，存取时间增长。故处理器件的个数不是毫无限制的，当然从理论上考虑，并不计较处理器的限制。

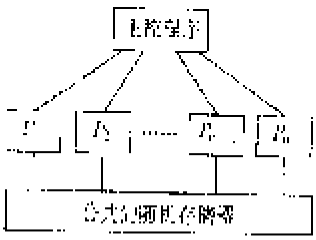


图 23.1-1

串行机和并行机都是依据指令对数据进行操作，有一指令流，告诉计算机每一步做什么工作。依据指令流和数据流的个数可将计算机分为如下四种类型：

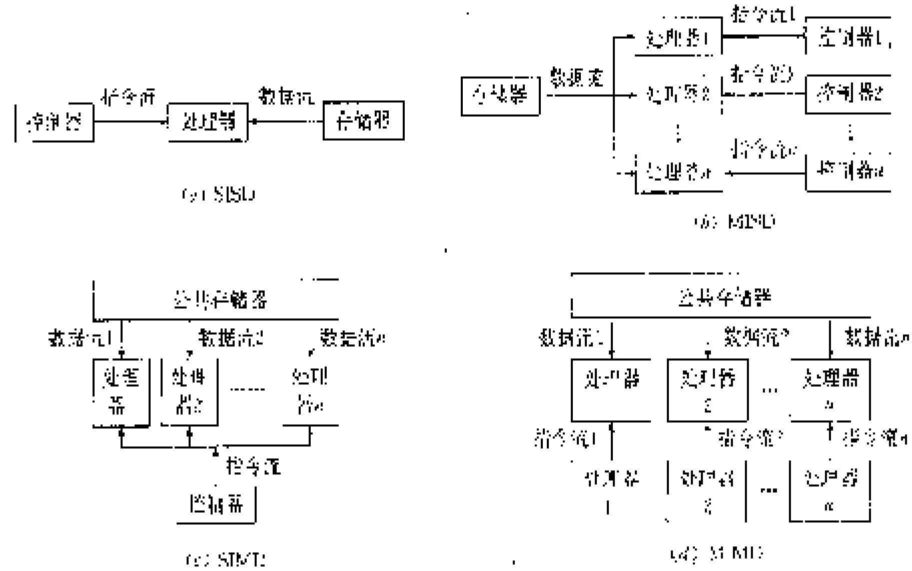
(1) 单指令流单数据流(Single Instruction Stream, Single Data Stream)，记为SISD；

(2) 多指令流单数据流(Multiple Instruction Stream, Single Data Stream)，用MISD表示；

(3) 单指令流多数据流，用SIMD表示；

④ 多指令流多数据流，用 MIMD 表示。

可用图 23.1.3 表示上述四种类型的结构。



(a) SISD　(b) MISD　(c) SIMD　(d) MIMD

图 23.1.3

其中 MISD 是有 n 个指令流，但只有一个数据流并行，这样的计算机比较特殊。对相同的数据流，各处理器用各自的指令流完成对数据的操作，比较适用于专用机。SIMD 是 n 个处理器由同一控制器来控制，各数据流不同，SIMD 模型是目前应用得最广泛的一种并行计算模型。MIMD 是较通用的并行机，但它由多个控制器控制，可看作是一种异步的计算，从复杂性分析及算法的设计上来说都较 SIMD 要复杂得多。

SIMD 和 MIMD 模型的处理机都使用了公共存储器，称之为并行的随机存储器(PRAM)。前面讲的两个处理器之间通过 PRAM 进行信息交流，比如处理器 i 要将一个数据传给处理器 j，则首先要将该数据写到 PRAM 的一个地址上，然后处理器 j 再从这个地址读这个数。而在并行算法的执行过程中，n 个处理器都在反复对 PRAM 进行存取，这样根据它们可否对同一地址作读写操作，又分为：

(1) EREW：EREW 是 Exclusive Read Exclusive Write，即不允许有两个处理器同时对同一地址进行读或写；

(2) CREW：C 为 Concurrent 的首字符，即允许同时读，但不允许同时写；

(3) ERCW：不允许同时读，但允许同时写；

(4) CRCW：允许同时读和写。

CRCW 型的 PRAM 当然可以执行 EREW 型的算法，反之则不然，CRCW 则需要更多的硬件支持。

若允许同时写，且有多个处理器要求对 PRAM 的同一地址写入不同的数，则出现冲突，哪一个数最终被允许写入？有几种解决问题的办法，这要看算法设计的需求：①按优先级，比如只写入其中最小的一个；②只允许写入的数值都相同，否则予以拒绝；③写入

它们的…，等等。

现介绍算法复杂性和对算法的评价，并行算法优劣的评价有以下几种标准：设问题的规模为 n。

(1) 运行时间 $t(n)$，即运行时间与问题规模 n 的关系。

(2) 处理器数目 $P(n)$，即所需的处理器数目与 n 的关系。

(3) 并行算法成本 $C(n)\triangleq P(n)t(n)$，即解题所需的步数。

(4) 加速比 $S_p(n)=T_s(n)/T_p(n)$，其中 $T_s(n)$ 是最快的串行算法最坏情况下所需的运行时间，$T_p(n)$ 是并行算法所需的运行时间，$T_p(n)$ 越小 $S_p(n)$ 就越大，故可以用来度量并行算法的改进程度。

显然有

$$1\leqslant S_p(n)\leqslant P(n)$$

这是因为 $T_s(n)\leqslant P(n)T_p(n)$

(5) 并行算法的效率 $E_p(n)=S_p(n)/p(n)$，$E_p(n)$ 可以用来度量处理器的效率。

后面举例说明并行算法。

下面举例说明并行算法。

例 1　从 n 个不同的数 $a_1,a_2,\cdots,a_n$ 的序列中找出最小的数。

若用串行办法处理，可在 $n-1$ 次比较中完成。若用 CRCW 型并行计算机并行处理，在同时写的数写进时录其中最小的，若计算机有 n 个处理器

$$P_1,P_2,\cdots,P_n$$

算法如下：

(1) i 从 1 到 n 并行作 P_i 读进 a_i。

(2) i 从 1 到 n P_i 向存储单元 M 并行作写进 a_i，则 M 最后得到的是

$$\min\{a_1,a_2,\cdots,a_n\}$$

又比如计算机有 n^2 个处理器 P_{ij}，$i,j=1,2,\cdots,n$。算法又如下：

(1) i 从 1 到 n 并行作 $M[i]\leftarrow 0$。

(2) i,j 从 1 到 n 并行作

【若 $a[i]<a[j]$ 则作 $M[j]\leftarrow 1$】。

(3) i 从 1 到 n 并行作

若 $M[i]=0$ 则输出 $a[i]$

又例如并行求和的问题。假定处理器 P_i 中存有数 a_i，$1\leqslant i\leqslant n$，要求在 P_i 里存和 $a_1+a_2+\cdots+a_i$。其并行算法可描述如下：

(1) $j\leftarrow 0$。

(2) i 从 2^j+1 到 n，处理器 P_i 并行作

【从 P_{i-2^j} 取得 a_{i-2^j}，$a_i\leftarrow a_i+a_{i-2^j}$】

(3) $j\leftarrow j+1$，若 $j\leqslant \log_2 n-1$，则转(2)。

否则，结束。

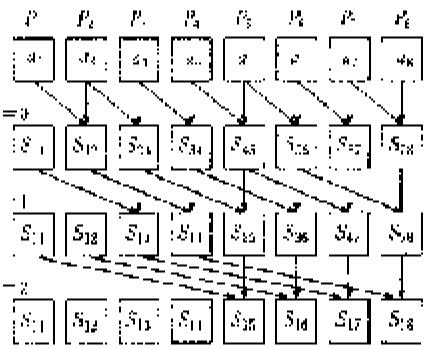


图 25.1.3

最后 P_i 即为所要求的。以 $n=8$ 为例，可形象表示如图 25.1.3。

从图23.1.3不难看出各步所做的工作：

$j=0$ 时：P_2 作 $P_2\leftarrow a_1+a_2$，P_3 作 $P_3\leftarrow a_2+a_3$，…

P_n 作 $P_n\leftarrow a_{n-1}+a_n$；

$j=1$ 时：P_3 作 $P_3\leftarrow a_1+(a_2+a_3)$，$P_4$ 作 $P_4\leftarrow(a_1+a_2)+(a_3+a_4)$，……

P_n 作 $P_n\leftarrow(a_{n-3}+a_{n-2})+(a_{n-1}+a_n)$；

$j=2$ 时：P_5 作 $P_5\leftarrow a_1+(a_2+a_3+a_4+a_5)$，…

$P_8\leftarrow(a_1+a_2)+(a_3+a_4+a_5+a_6)$…等等。

图中 S_{ij} 表示 $S_{ij}=a_i+a_{i+1}+\cdots+a_j$。

本章以后将分别就具体问题讨论其并行算法的设计。

23.2 递推关系的并行计算

由于实际中许多问题都可以归结为递推关系来求解，例如常微分方程初值问题数值解和矩阵求逆等，所以讨论递推关系的并行计算是有重要意义的。

设递推关系为

$$a_i=b_ia_{i-1}+c_i,\quad a_0=d$$

$$i=1,2,\cdots,n$$

其并行算法求解并非我们想像的那样 $a_1,a_2,\cdots,a_n$ 只能依次产生，即计算 a_i 前必须先计算 a_{i-1}，计算 a_{i-1} 必须先求出 a_{i-2} 等等。下面我们首先分析递推关系在并行方面的若干特性。

因为

$$a_i=b_ia_{i-1}+c_i$$

$$a_{i-1}=b_{i-1}a_{i-2}+c_{i-1}$$

则

$$\begin{aligned}a_i&=b_i(b_{i-1}a_{i-2}+c_{i-1})+c_i\\&=b_ib_{i-1}a_{i-2}+b_ic_{i-1}+c_i\\&=b_i^{(1)}a_{i-2}+c_i^{(1)}\end{aligned}$$

其中 $b_i^{(1)}=b_ib_{i-1}$

$c_i^{(1)}=b_ic_{i-1}+c_i$

依此以上各步骤可得：

$$a_i=b_i^{(j)}a_{i-2^j}+c_i^{(j)}$$

$$i=1,2,\cdots,n$$

$$j=0,1,\cdots,\log_2 n$$

其中

$$b_i^{(j)}=b_i^{(j-1)}b_{i-2^{j-1}}^{(j-1)}\tag{23.2.1}$$

$$c_i^{(j)}=b_i^{(j-1)}c_{i-2^{j-1}}^{(j-1)}+c_i^{(j-1)}\tag{23.2.2}$$

令

$$b_i^{(0)} = b_i, \qquad c_i^{(0)} = c_i$$

则(23.2.1)式计算如图 23.2.1 所示.

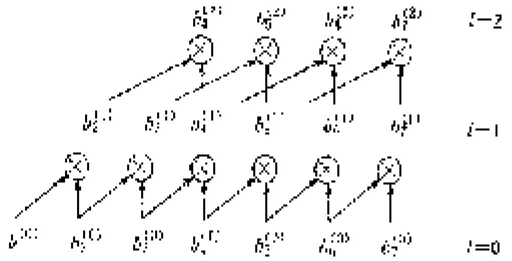


图 23.2.1

式(23.2.2)也可以并行计算如图 23.2.2 所示。

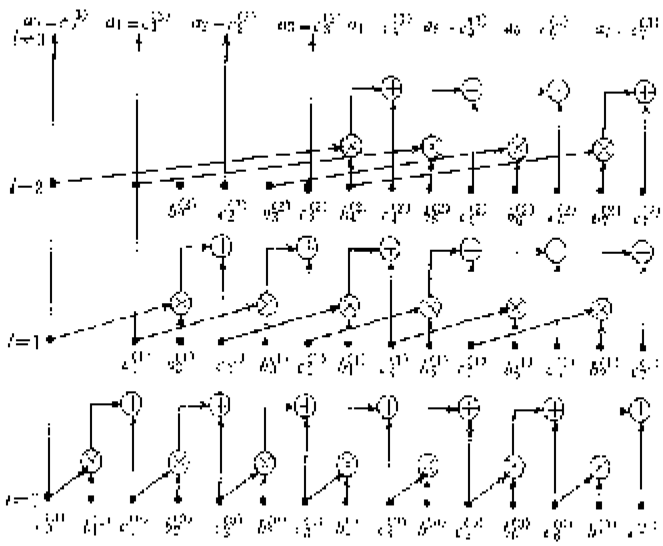


图 23.2.2

23.3 图的并行算法举例

1. 倍增技术

本节首先介绍一种并行算法中用得比较多的倍增技术，例如一个具有 n 个元素序列

$$a_1, a_2, \cdots, a_n$$

用链表形式存储，如图 23.3.1 所示

现要求统计每个元素在序列中的次序。若用串行算法，则可在 $O(n)$ 时间内完成，但若用 n 个处理器并行工作，可在 $O(\log_2 n)$ 时间内结束。

令最后一个元素 a_n 的指针为空，a_i 所指向下一个元素的地址用 $Ptr(i)$ 表示，则计算 a_i 到最后一元素的距离 $d(i)$ 的算法可描述如下：

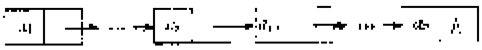


图 23.3.1

(1) i 从 1 到 n 并行作

【若 $Ptr(i)$ 不空则 $d(i)\leftarrow 1$，否则 $d(i)\leftarrow 0$】。

(2) 当存在一个 i 使得 $Ptr(i)$ 非空则作

【i 从 1 到 n 并行作

【若 $Ptr(i)$ 非空，则【$d(i)\leftarrow d(i)+d(Ptr(i))$，

$Ptr(i)\leftarrow Ptr(Ptr(i))$】】】。

上面算法可用例子表示如图 23.3.2 所示。

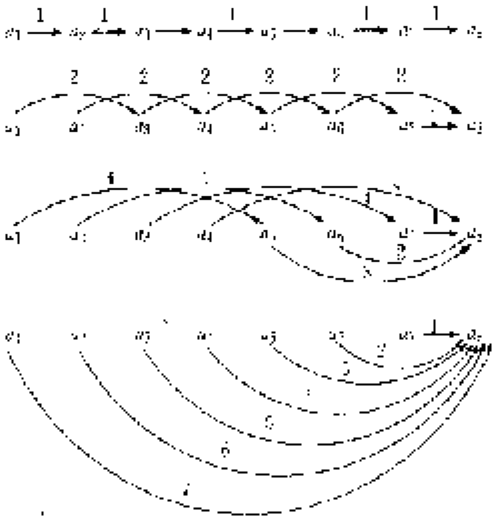


图 23.3.2

图 23.3.2 中箭头形象地表示指针，弧上的数值是 $d(i)$。再来以 n 个元素的序列

$$a_1, a_2, \cdots, a_n$$

为例，利用倍增技术求前 i 项和

$$S_i = a_1 + a_2 + \cdots + a_i,$$

和前面一样，元素 $a_i(i<n)$ 有一指针 $Ptr(i)$ 指向 a_{i+1}，a_n 的指针为空。

(1) i 从 1 到 n 并行作 $S(i)\leftarrow a(i)$。

(2) 只要存在一个 i，使得 $Ptr(i)$ 不为空，

则作【i 从 1 到 n 并行作

【$S(Ptr(i))\leftarrow S(i)+S(Ptr(i))$，

$Ptr(i)\leftarrow Ptr(Ptr(i))$】】。

例如(见图 23.3.3)。

2. Euler 链图法

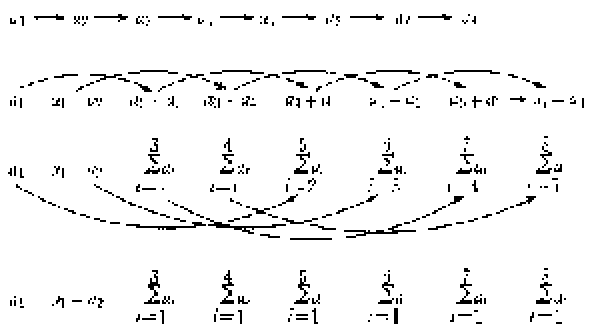
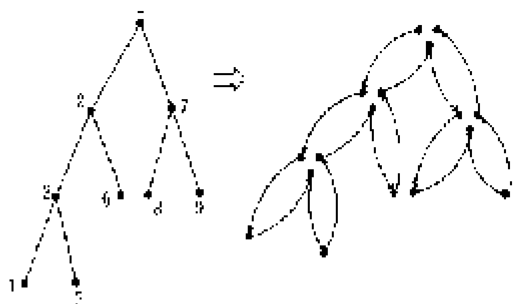


图 23.3.3

现在介绍一种求二分树各结点的深度的并行算法。对于任何一二分树可以构造一 Euler 链，如图 23.3.4 所示。



(a)　　　　(b)

图 23.3.4

将二分树的每一结点 i 对应一组指针 (l_i, r_i, p_i)，其中 l_i 对应于左指针，r_i 对应于右指针，p_i 为 i 的"父亲"指针，见图 23.3.5(a)。这样从图 23.3.4(a) 得一 Euler 链图，如图 23.3.5(b) 所示。

一般来说，l_i 应指向左儿子节点 j 的 l_j，但当为叶子节点或无左儿子时 l_i 应指向本节点的 r_i。同样的理由 r_i 应指向 i 的右儿子节点 k 的 l_k，但若无右儿子则指向本节点的 p_i。对于 p_i 而言，若 j 是其父亲的 p_j，则 p_i 应指向 p_j。

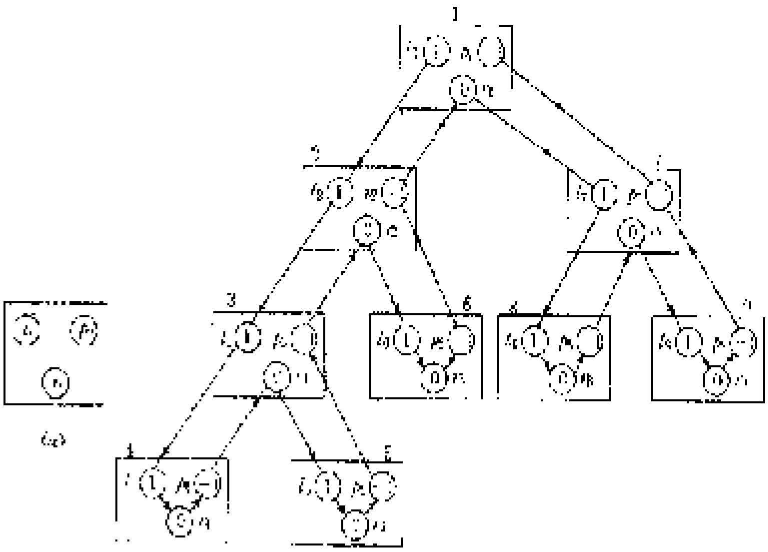
现在利用倍增技术从二分树根节点开始，沿 Euler 链，求节点"()"中数的和。其中所有 l_i 圆内数为 1，所有 r_i"()"里数均为 0，p_i 对应的"()"中数均为 −1。若将上述链拉直，可看作由 $3n$ 个数组成的链，若能计算得到每个节点的前缀和，则开始时为 l_i 的节点上的和就是该节点在树中的深度。对于图 23.3.5，通过计算，其结果如图 23.3.6 所示。

对于有 n 个节点的二分树，有 $3n$ 个运算对象，故可在 $O(\log_2 n)$ 时间内完成。当然算法不要求并发存取，故属于 EREW 型。

3. 已知连通图 $G=(V,E)$ 的距离矩阵为

$$D = (d_{ij})_{n\times n}$$

其中 $n = |V|$，且



(b)

图 23.3.5

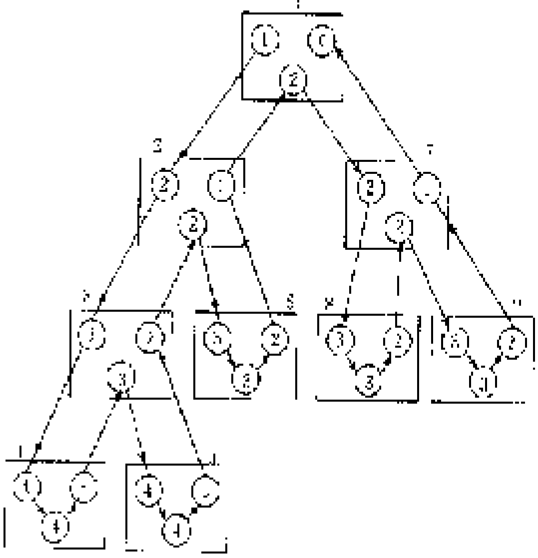


图 23.3.6

$$d_{ij}=\begin{cases}(i,j)\text{ 边的长度}, & (i,j)\in E\\ \infty & ,\text{其他}\end{cases}$$

求任意两点间的最短距离 p_{ij}。

计算路径从 i 出发经过 K 个中间点最后到达 j 的最短距离 p_{ij},可如下计算:

$$p_{ij}=\begin{cases}0, & i=j\\ \min\{d_{ii_1}+d_{i_1i_2}+\cdots+d_{i_kj}\}, & i\neq j\end{cases}$$

即计算

$$i \to i_1 \to i_2 \to \cdots \to i_k \to j$$

的长度。

求 p_{ij} 可用如下并行算法来求，算法的正确性可用归纳法证明。

第(1)步：对所有 i 和 j 并行作 $p(i,j) \leftarrow d(i,j)$；

第(2)步：下面的各操作执行 $\lceil \log n \rceil$ 次：

【对所有 i,j,k 并行作 $R(i,j,k) \leftarrow d(i,j)+d(j,k)$；对所有 i,j,k 并行作 $p(i,j) \leftarrow \min(d(i,j), R(i,1,j), R(i,2,j), \cdots, R(i,n,j))$；】

23.4 矩阵乘积的并行计算

两个矩阵的乘法是最基本的代数运算，设

$$A=(a_{ij})_{m\times n},\quad B=(b_{ij})_{n\times l}$$

令

$$C=(c_{ij})_{m\times l}=AB$$

$$c_{ij}=\sum_{k=1}^{n} a_{ik}b_{kj},\quad i=1,2,\cdots,m$$

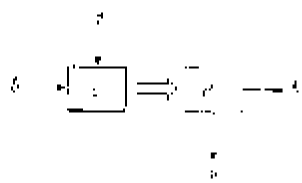


图 23.4.1

先介绍一种计算

$$y_i=\sum_{j=1}^{n} a_{ij}x_j,\quad i=1,2,\cdots,n$$

的并行计算处理办法，图 23.4.1 中的单元 □ 为处理器。

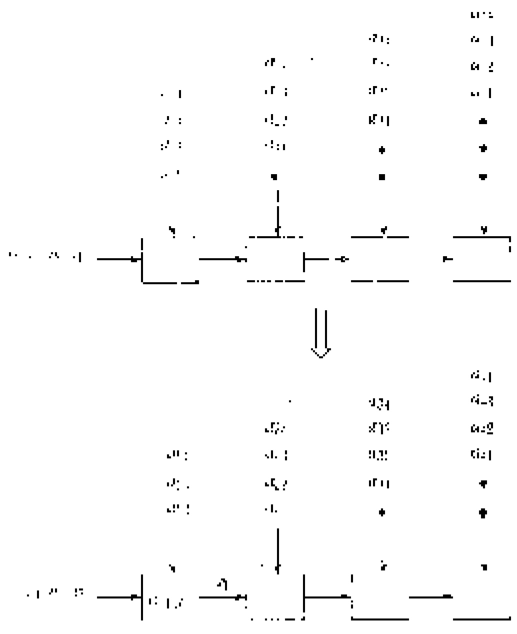


图 23.4.2

其中 $c'=ab+c$。

以 $n=4$ 为例叙述由 $n=4$ 个处理器的线性阵列如图 23.4.2。

计算 $2n-1$ 步结束，从第 i 单元中取出 y_i，$i=1,2,\cdots,n$，串行计算则需作 $2n^2$ 步。

类似可构造二维的阵列用以计算两个矩阵的乘积，$n=4$ 为例叙述如 23.4.3 所示。

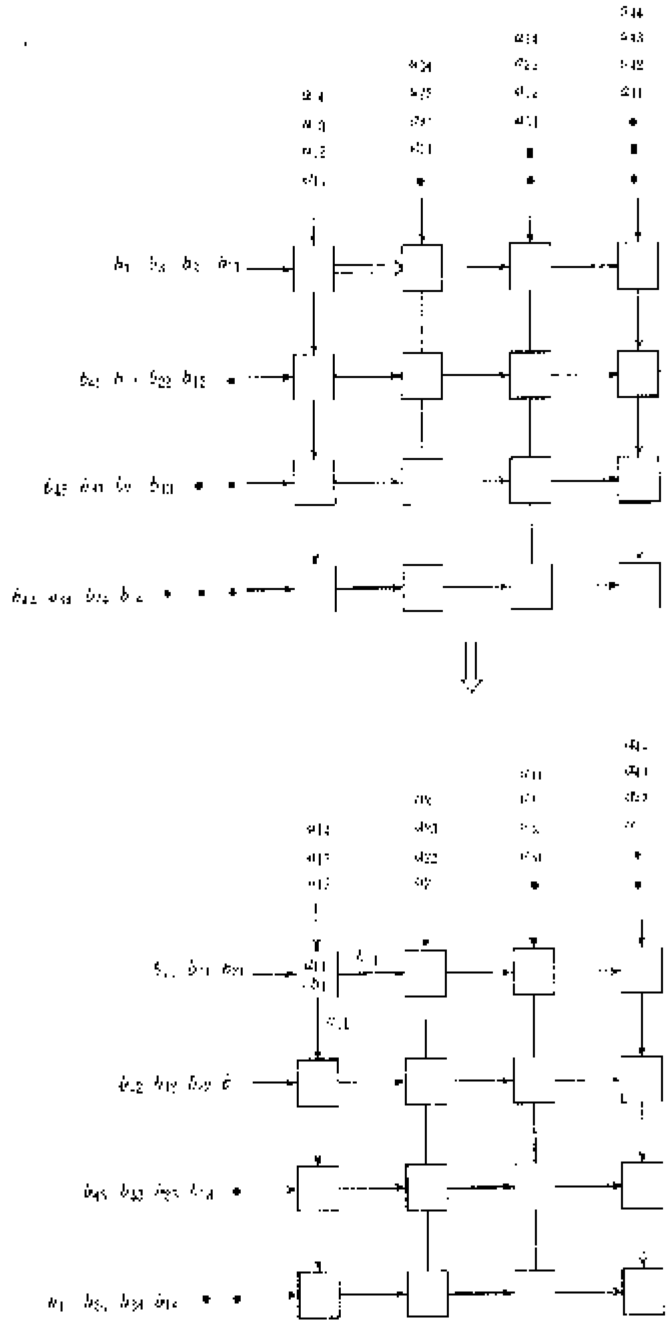


图 23.4.3

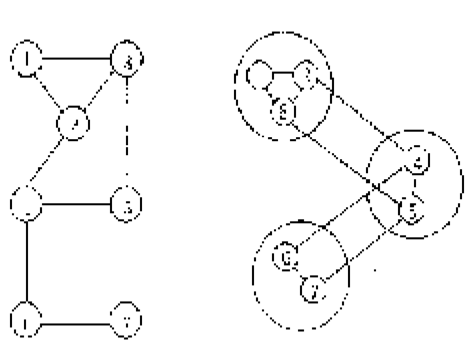
23.5 分布计算

首先要介绍什么叫分布式计算机系统，它是由多个相互联结的计算机组成的，这些计算机在物理上是相邻，也可以是分开的，通过网络进行通信，使得一个程序可以分散在各个计算机上并行运算。各计算机地位平等，不存在主从控制和集中控制环节，在这样的分布系统上进行并行计算叫做分布计算。它实际上是 *MIMD* 型并行计算机系统。

前面讨论的并行计算大都针对特定的问题设计由处理器组成相应的计算系统，而分布计算则根据客观实际的计算资源针对特定问题设计并行运算的流程。但并行计算主要着眼于各处理器的代价，而分布计算则必须更多考虑计算资源间的通信代价及其同步控制，这是因为它们之间松散耦合所引起的。

总而言之，并行算法是由算法确定计算环境，而分布计算则是由确定的环境确定算法，它与并行计算的联系和区别概括地说是：它们都采用多个处理器同时并行地计算以求得问题的解，但并行计算所关心的主要是各处理器的计算代价（这是由其紧耦合特性所决定的），而分布式处理关心得更多的是处理器之间的通信代价及同步控制（因之一般是松散耦合的）。

图 23.5.1(*a*) 中小圆表示由许多过程构成的模块，它们可以在不同的处理机执行，连线表示彼此模块之间存在传递控制。图 23.5.1(*b*) 中的大圆表示处理机。处理机间可以互相通信，这样的多机系统并不完全连接，在这样的环境下，要指定某一模块在某处理机，或要求程序运行的时间或费用达到最小，或要求资源的利用达到最佳标准。



(*a*)　　　　(*b*)

图 23.5.1

在串行程序模块进行分布式计算时，目的在于使得执行某种运算时充分利用某种处理机的特殊效果，比如某一程序的某过程的浮点运算量很大，就可以安排该过程在浮点运算能力很强的处理机上运行，如果处理机间的通信及控制传递不需要付出代价，这样的作法永远是容易被接受的，只不过有时候这样的费用是昂贵的，就需要综合考虑它的得失。如图 23.5.2(*a*) 表示串行程序在同一处理机上执行，所需时间为 T_1，图 23.5.2(*b*) 表示(1)在处理器 P_1 上执行，(2)、(3)在 P_2

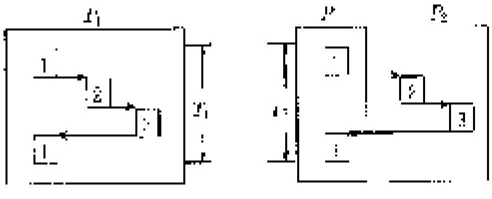


图 23.5.2

上执行。所需时间为 T_2。虽然处理机间通信付出了代价，但由于 P_2 执行(2)、(3)效率高，总时间比串行计算时要短。

习　　题

1. 已知序列 $a_1,a_2,\cdots,a_n$，试设计一 EREW 算法找出其中第 $\lfloor n/2 \rfloor$ 个元素，并分析其复杂度。

2. 已知 $a_1,a_2,\cdots,a_n$ 是一 0,1 序列，试设计一 EREW 算法，将它分为全零和全 1 两个子序列。

3. 已知一二元树，试设计一 EREW 算法，按左—右—中顺序遍历该二元树。

第 24 章　脉动阵列的并行处理

当今的大规模集成电路技术（VLSI）表明，各处理单元之间简单、规则的连接是导致高集成度和实现方便的关键因素。70 年代中后期提出的脉动 Systolic 网络方式正是在这一原则指导下得的并行处理结构，其基本思想是通过开发计算的可并行及流水特性来获取高速完成。脉动阵列的组成一般具有下述特点：网络中的处理器具有统一的结构，完成的是针对固定的且简单的操作，且只与几何邻近的处理器相连接，因而布局极其规则；对外数据传送在一种统一的时钟控制下以一种整齐、均匀、有节奏的方式进行。通常以脉动网络方式构成的专用计算设备，作为大型机的外接处理机完成某些专门计算（如矩阵乘法、快速傅里叶变换等），其计算速度往往非常高速。

24.1　矩阵和向量乘法的并行处理

首先我们介绍处理器的基本部件。如图 24.1.1 所示的基本部件，它有上、左、右三条输入线和左右两条输出线，其功能是将左边输入和上面输入进行乘法运算，将其结果和右方输入相加，最后从左方输出。利用多个简单基本部件组成的阵列可完成相当复杂的计算，例如求矩阵乘积。

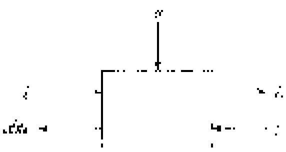


图 24.1.1

以 3×3 矩阵乘 3×1 列向量为例：

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

则该矩阵与向量相乘的“脉动”阵列可用图 24.1.2 表示。

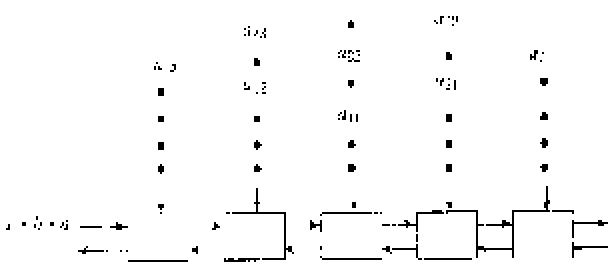


图 24.1.2

图 24.1.2 运行时脉动阵列的工作状态可动态表示如图 24.1.3。

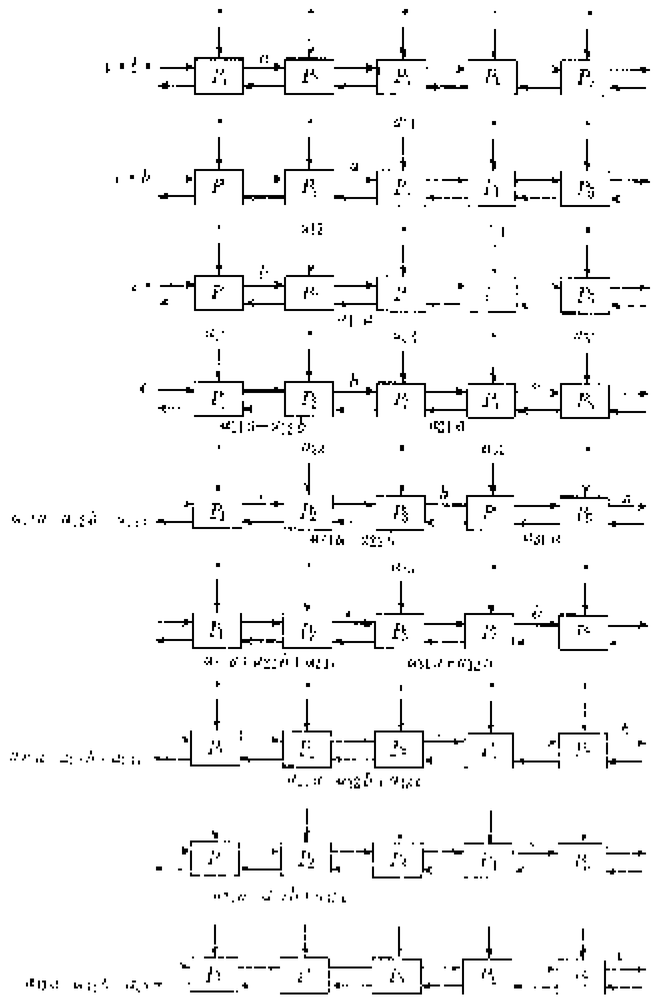


图 24.1.3

图中符号“·”为空格符。

24.2 矩阵乘法的并行处理

下面再介绍一种完成一般矩阵乘法的脉动阵列。同样先介绍一种网状结构的矩阵乘法的基本元件。其功能用图 24.2.1 来形象表示。

其中 C 为处理器 P 的寄存器，(C) 为寄存器所存储的内容，不引起混淆的情况下简记为 C。

设 $A=(a_{ij})_{n\times n}$，$B=(b_{ij})_{n\times n}$，$C=(c_{ij})_{n\times n}$，$C=AB$ 为例介绍如图 24.2.2 所示。

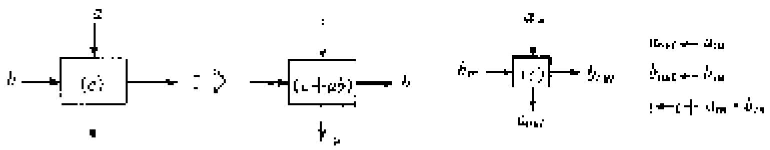


图 24.2.1

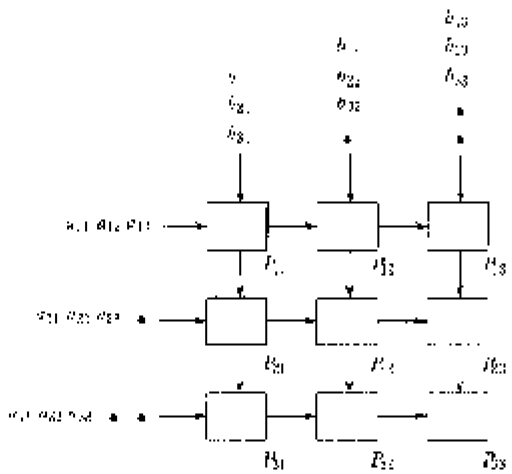


图 24.2.2

算法开始时，先将 C 置为 0。n 阶方阵的乘法需要 n^2 个处理器，其时间复杂度为 $O(n)$。

例如求：

$$C=\begin{pmatrix}2&1&3\\1&3&0\\2&0&3\end{pmatrix}\begin{pmatrix}-1&1&2\\3&2&0\\2&3&1\end{pmatrix}$$

其过程可表示如图 24.2.3。

最后得：

$$C=\begin{pmatrix}7&13&7\\8&7&2\\4&11&7\end{pmatrix}$$

从图 24.2.3 可看出共进行了 7 次乘法和加法。

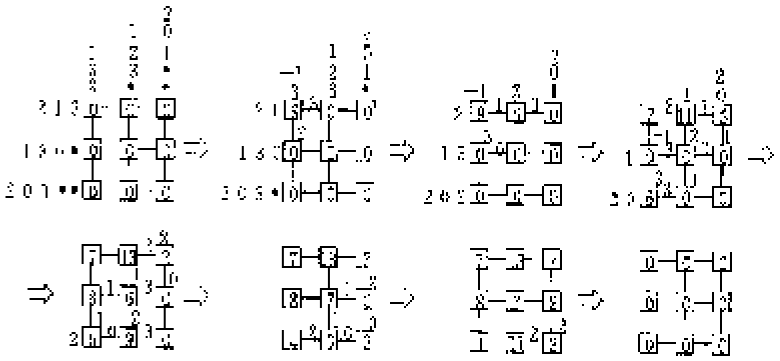


图 24.2.3

24.3 带状矩阵的并行乘法

在矩阵运算中有一类特殊而又应用广泛的带状矩阵乘法，所谓带状矩阵是指矩阵除主对角线附近的带状区域元素以外，其余元素均为零，因而若仍采用24.2节中的阵列，将有大量浪费，但利用下面介绍的阵列则可高效的计算这类运算。

带状矩阵的并行乘法采用一种六角状的基本运算部件如图24.3.1。

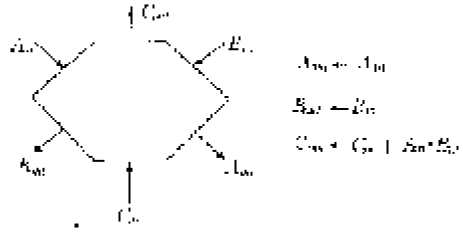


图 24.3.1

例如求下列矩阵之积

$$
\begin{bmatrix}
a_{11} & a_{12} & & & \\
a_{21} & a_{22} & a_{23} & & \\
a_{31} & a_{32} & a_{33} & a_{34} & \\
0 & a_{42} & a_{43} & a_{44} & a_{45} \\
 & \ddots & \ddots & \ddots & \ddots
\end{bmatrix}
\begin{bmatrix}
b_{11} & b_{12} & b_{13} & & & \\
b_{21} & b_{22} & b_{23} & b_{24} & & \\
 & b_{32} & b_{33} & b_{34} & b_{35} & \\
 & & b_{43} & b_{44} & b_{45} & b_{46} \\
 & & \ddots & \ddots & \ddots & \ddots
\end{bmatrix}
$$

$$
=\begin{bmatrix}
c_{11} & c_{12} & c_{13} & c_{14} & \cdots \\
c_{21} & c_{22} & c_{23} & c_{24} & \cdots \\
c_{31} & c_{32} & c_{33} & c_{34} & \\
c_{41} & c_{42} & c_{43} & c_{44} & \\
 & c_{52} & c_{53} & c_{54} & c_{55} \\
 & \ddots & \ddots & \ddots & \ddots
\end{bmatrix}
$$

上面是带宽为4的两个矩阵A和B相乘，计算可以通过一个4×4六角形连接的处理单元阵列来实现（如图24.3.2所示）。

$$c_{11} = a_{11}b_{11} + a_{12}b_{21}$$
$$c_{12} = a_{11}b_{12} + a_{12}b_{22}$$
$$c_{21} = a_{21}b_{11} + a_{22}b_{21}$$
$$c_{13} = a_{12}b_{23}$$
$$c_{14} = c_{15} = \cdots = 0$$
$$c_{22} = a_{21}b_{12} + a_{22}b_{22}$$
$$c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32}$$
$$\cdots\cdots\cdots\cdots$$

图24.3.2描绘了算法运行概貌（箭头表示数据流）：A、B和C的带中元素同步地从三个方向穿过网络，每个c_{ij}从底部边缘进入网络时取初值0。容易看出使用图24.3.1所示的处理单元，每个c_{ij}从顶部边缘离开时能累加到它的所有项。

图24.3.3给出了算法执行的几个连续步骤，帮助读者理解，从而就不难了解到该阵列的思想及其高速有效的原因。

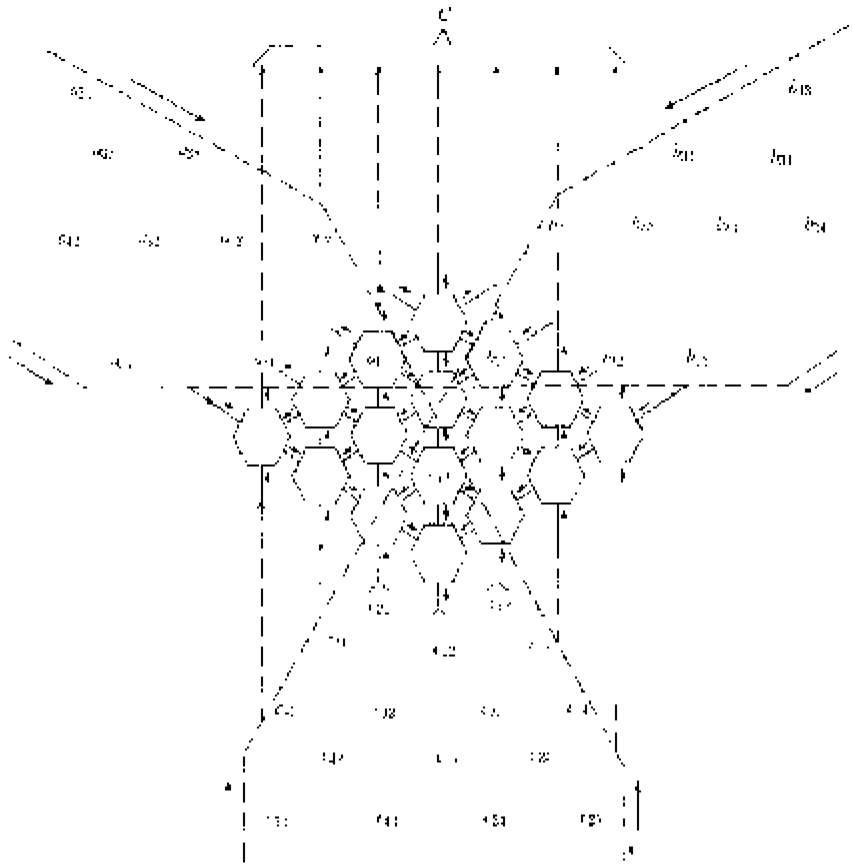


图 24.3.2

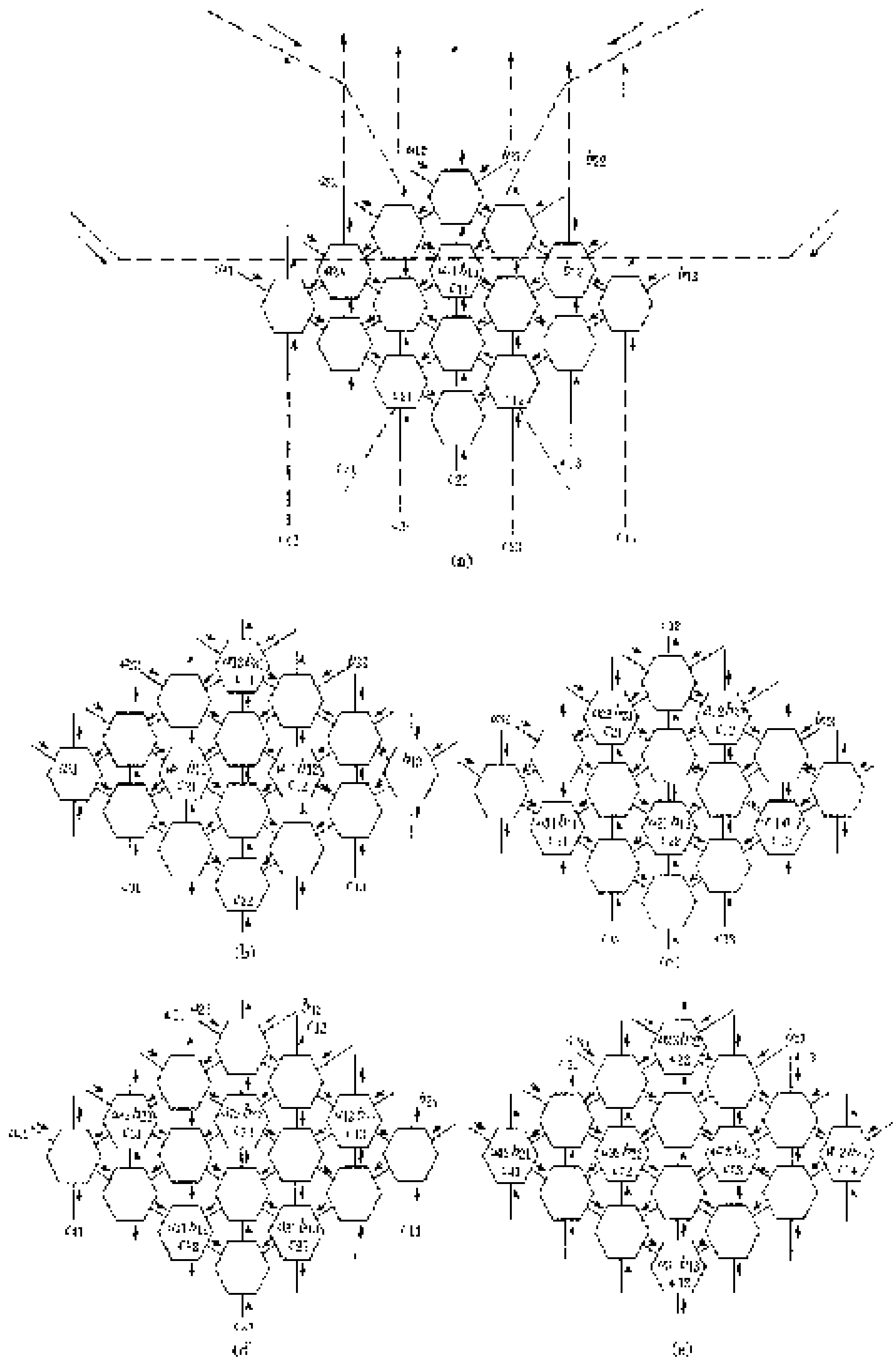


图 24-3-3

习　　题

利用矩阵积的并行算法，求下面矩阵之积并追踪下面例子的求积过程。

$$\begin{pmatrix} 1 & 3 & 2 \\ 3 & 2 & 1 \\ 2 & 3 & 1 \end{pmatrix} \begin{pmatrix} -1 & -1 & 1 \\ 0 & 2 & 1 \\ 3 & 1 & 0 \end{pmatrix}$$

第25章 计算几何

在计算机的应用领域，人们愈来愈频繁地处理本质上属于几何对象的问题，例如超大规模集成电路设计自动化问题、计算机辅助设计、计算机图形学、机器人等学科中的问题。

几何是一门古老的科学，计算几何算法则是新近才发展起来的新的研究领域，现代的图象显示系统集中精力去研究表示和处理几何物体的相交、显示等问题，促进了几何算法的产生和发展，大量有趣的新的算法被设计出来，形成计算几何新学科。

几何问题与其它问题的不同之处在于哪怕最简单的、最初等的几何问题也感到难以数学形式去处理它们，人们可以凭直观来判断一个初等的几何问题，但设计一个计算机程序来解决就有困难。因为人的观察和计算机算法采取的方法可能是完全不同的。

可以说计算几何算法的特点是首先它们要处理的是几何对象，另外虽然在具体处理时仍是数字处理，但利用几何直观可以为算法的设计提供指导和依据，从而降低了算法设计的难度。

本章简单介绍一些计算几何中的比较简单的问题，引导读者进一步去研究。

25.1 关于线段问题

首先讨论平面上线段的有关性质。已知平面上任意两点 $P_1(x_1,y_1)$、$P_2(x_2,y_2)$，则对线段 P_1P_2 上的任意一点 $P_3(x_3,y_3)$，存在一实数 α，$0\le\alpha\le1$，使得 $x_3=\alpha x_1+(1-\alpha)x_2$，$y_3=\alpha y_1+(1-\alpha)y_2$。

若将 P_1 和 P_2 两点分别看作是以(0,0)点为始点的向量 $\boldsymbol{P}_1$ 和 $\boldsymbol{P}_2$，则 $\boldsymbol{P}_1$ 和 $\boldsymbol{P}_2$ 的向量积 $\boldsymbol{P}_1\times\boldsymbol{P}_2$ 定义为：

$$\boldsymbol{P}_1\times\boldsymbol{P}_2=\det\begin{pmatrix}x_1 & x_2\\ y_1 & y_2\end{pmatrix}=x_1y_2-x_2y_1=-\boldsymbol{P}_2\times\boldsymbol{P}_1$$

不难知道 $x_1y_2-x_2y_1$ 的绝对值实际上是以 $\boldsymbol{P}_1$ 和 $\boldsymbol{P}_2$ 为两边的平行四边形的面积，见图 25.1.1。

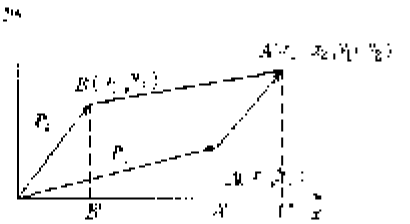


图 25.1.1

平行四边形 $OACB$ 的面积 S 等于 $\triangle OBB'$ 的面积加上梯形 $BB'C'C$ 的面积，减去 $\triangle OAA'$ 的面积再减去梯形 $AA'C'C$ 的面积。

$$\therefore S = \frac{1}{2}x_2y_2 + \frac{1}{2}(y_2 + y_1 + y_2)x_1 - \frac{1}{2}x_1y_1 - \frac{1}{2}(y_1 + y_1 + y_2)x_2$$

$$= x_1y_2 - x_2y_1$$

换句话说：$P_1 \times P_2$ 为正，则 P_1 反时针方向旋转小于 180°到 P_2；反之若 $P_1 \times P_2$ 为负，则 P_1 是顺时针旋转小于 180°到 P_2。它们的绝对值则都是以 P_1 和 P_2 为两边的平行四边形的面积。特别当 $P_1 \times P_2 = 0$，则 P_1 和 P_2 同向或反向共线。

一般地，以某一点 $A(x_1, y_1)$ 为始点的两个向量 AB 和 AC，它们的顺时针或反时针的关系也可用类似的计算来判断。设 B 点坐标为(x_2, y_2)，C 点为(x_3, y_3)，则 $AB = (x_2 - x_1, y_2 - y_1)$，$AC = (x_3 - x_1, y_3 - y_1)$，$AB \times AC = (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)$。

由于两个相邻的线段 AB 和 BC 关于 A 点的顺时针、逆时针旋转关系，如图 25.1.2 所示也可通过类似计算确定。这些留给读者自己思考。

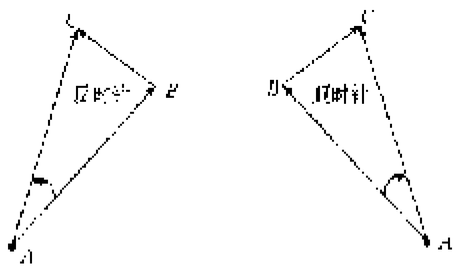


图 25.1.2

其次我们来讨论线段相交的问题：

判断两线段是否相交的最直接了当的方法是先求出两直线的交点，再判断该点是否在两线段中间。下面介绍较直观且简便的另一种方法。

假定有两线段 AB 和 CD，已知它们的坐标分别为 $A(x_1, y_1)$，$B(x_2, y_2)$，$C(x_3, y_3)$，$D(x_4, y_4)$。对每一条线段都存在以该线段为对角线的各边平行于坐标轴的矩形，如图 25.1.3 所示。

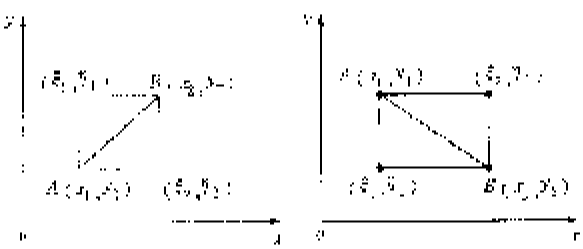


图 25.1.3

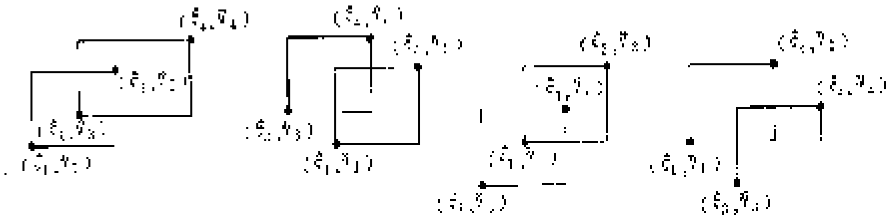
总之包含对角线 AB 的长方形的左下方顶点的坐标设为$(\bar{x}_1, \bar{y}_1)$，右上角的顶点坐标

记为(ξ_2,η_2)，令

$$\xi_1=\min\{x_1,x_2\},\quad \eta_1=\min\{y_1,y_2\}$$

$$\xi_2=\max\{x_1,x_2\},\quad \eta_2=\max\{y_1,y_2\}$$

这样的长方形记为$R((\xi_1,\eta_1),(\xi_2,\eta_2))$，两个长方形$R_1((\xi_1,\eta_1),(\xi_2,\eta_2))$和$R_2((\xi_3,\eta_3),(\xi_4,\eta_4))$相交的充要条件是满足下面4个条件(图25.1.4)：



(a)　(b)　(c)　(d)

图 25.1.4

$$\xi_2\geq\xi_3,\ \xi_4\geq\xi_1,\ \eta_2\geq\eta_3,\ \eta_4\geq\eta_1$$

若以线段$\overline{AB}$为对角线的长方形记为$R_1((\xi_1,\eta_1),(\xi_2,\eta_2))$，以线段$\overline{CD}$为对角线的长方形记为$R_2((\xi_3,\eta_3),(\xi_4,\eta_4))$，则若$R_1$和$R_2$不相交，则线段$\overline{AB}$和$\overline{CD}$不相交，但$R_1$和$R_2$相交线段$\overline{AB}$和$\overline{CD}$未必相交。

设定A、B2点对应2个向量P_A、P_B，即，如图25.1.5所示。

向量AB可表示为P_B-P_A。

现在我们来进一步分析一下图25.1.6中所示的各种情形。

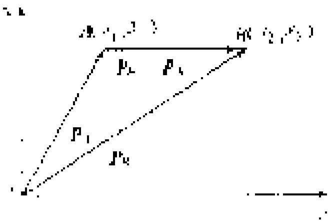


图 25.1.5

例如对于(a)有：$(P_C-P_A)\times(P_B-P_A)<0$，

$$(P_D-P_A)\times(P_B-P_A)<0$$

此时线段AB和CD不相交。

对于(b)有　$(P_C-P_A)\times(P_B-P_A)<0$

且$(P_D-P_A)\times(P_B-P_A)>0$

此时AB和CD相交。

对于(c)有　$(P_C-P_A)\times(P_B-P_A)<0$

且$(P_D-P_A)\times(P_B-P_A)=0$

对于(d)有　$(P_C-P_A)\times(P_B-P_A)=0$

$$(P_D-P_A)\times(P_B-P_A)=0$$

对于(d)，虽然P_A、P_B、P_C、P_D共线，但不相交。

在讨论了上面各种形状所满足的性质后，现讨论n个线段$S_1,S_2,\cdots,S_n$中有无相交的两线段，又哪些线段是相交的？设n个线段的端点分别为(a_i,b_i)和(c_i,d_i)，$i=1,2,\cdots,n$，且假定这n条线段无一垂直于x轴，没有三线共点的现象。

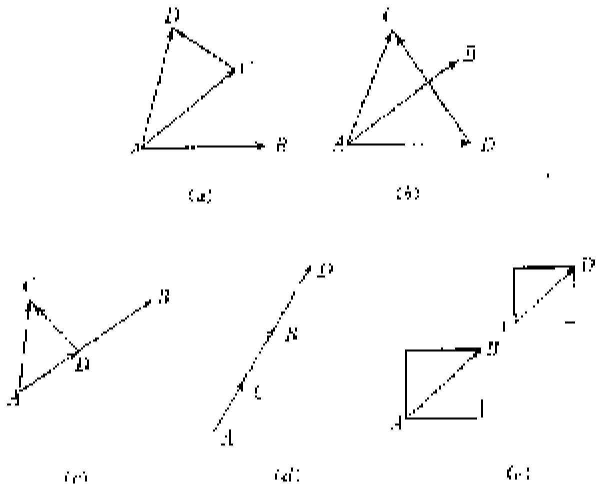


图 23.1.6

图 23.1.7 中过各线段的端点有一些平行的虚线，从左向右扫描。各虚线的下面有一组数，表明它和两线段相交自上向下的顺序，例如从左向右，自下而上依次有 1；3，2；1，2，4；……表明第一条虚线和 S_1 相交，第 2 条虚线依次和 S_3，S_2 相交，……等等。从这序列可以看出哪些线段是相交的。例如从第(5)条虚线下面的 3，2，4 和第(5)条线的 2，3，1，5 可见 S_2 和 S_3 相交。我们称这样一组虚线为扫描线。即 S_2 和 S_3 相交必然在扫描线上前后顺序颠倒。令每条扫描线下的序列记作 S'。下面判定线段是否相交的算法是很直观的。

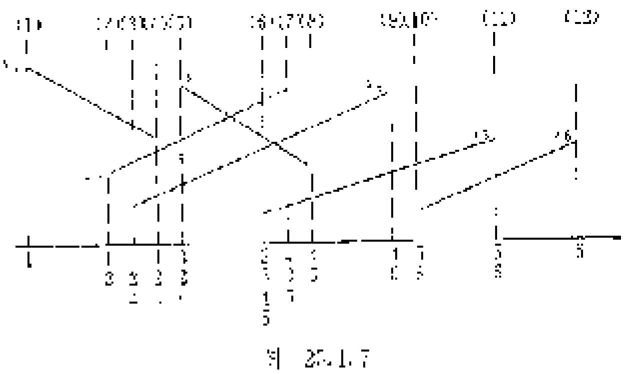


图 23.1.7

(1) 对 n 条线段的 $2n$ 个端点，从左到右，自下而上按坐标(x,y)的顺序排成队 Q。

(2) $A \leftarrow \phi$，$S' \leftarrow \phi$。

(3) 若队列 Q 非空，转(4)。否则，输出 A，停止。

(4) 从队列 Q 中取出最前面的元素 P；

(5) 若 P 是线段 S 的左端点，则作

【将 S 插入到 S' 中去。序列 S' 中在 S 前面的元素非空，设最后一个元素为 S_1；

序列 S' 中在 S 后面的元素非空，设其中最前一个元素为 S_2；

若 S 和 S_1 相交，则 $A \leftarrow \{(S_1,S)\} \cup A$；

若 S 和 S_2 相交，则 $A \leftarrow \{(S, S_2)\} \cup A$；】

否则，转(6)。

(6) 若 p 是线段 S 的右端点则作

【序列 S' 中在 S 前面的元素非空，设其中最后一元素为 S_1；序列 S' 中在 S 后面的元素非空，设其中最前一元素为 S_2；

若 S_1 和 S_2 相交，则 $A \leftarrow \{(S_1, S_2)\} \cup A$；

从 S' 中删去 S.】。

(7) 转(3)。

算法中 S' 是按前后顺序排列的线段序列。开始和终了，S' 都是空集。在插入和删除过程中可考虑使用 2-3 树或其它合适的数据结构。A 中存放的是一对对两两相交的线段。

25.2 求凸包问题

平面上一个点集的凸包是包含所有点的最小凸多边形，或说是围绕所有点的最短路径。它是一个基本几何计算。它在许多统计计算，特别是高维统计计算中起着重要的作用。

凸包有许多性质有助于设计算法，如凸包内任两点的连线必在凸包之内；如果在凸包外画一条线，然后将这条线绕某固定点旋转，那么它将首先接触到凸包边界的某顶点；等等。

求凸包是一个有趣的问题，而且许多计算几何的问题都要从求凸包开始，所以是计算几何的基础问题。下面介绍若干求凸包的算法。

1. 卷包裹法

从一个已经肯定在凸包上的点出发(例如 y 坐标最小的点)画一条水平直线，以该点为中心将该直线按逆时针方向(倾角增大方向)向上转动(以出发点为轴心)直至接触到点集中某点设为 v_1，它也必定在凸包边界上，再以 v_1 为轴心。这样继续下去……直至该直线转了 360°从而回到出发点位置。这样的过程类似于卷包裹直至完全把点集中点卷在内部为止。如图 25.2.1。

图 25.2.1

设 n 个点为 $P_1(x_1, y_1), P_2(x_2, y_2), \cdots\cdots, P_n(x_n, y_n)$，则求其凸包的算法可描述如下：

(1) 求 $P_i(x_i, y_i), i=1,2,\cdots,n$ 中 $y_1, y_2, \cdots, y_n$ 的最小值设为 y_m，即 $P_m(x_m, y_m)$ 为 y 坐标最小点。

(2) $M \leftarrow 0$，令 $P(n+1) \leftarrow P(m)$，$A \leftarrow 0$。

(3) $M \leftarrow M+1$，$P(M)$ 和 $P(m)$ 互换，$m \leftarrow n+1$，$v \leftarrow A$，$A \leftarrow 360$。

(4) i 从 $M+1$ 到 $n+1$ 作

【求 $P(M)$ 和 $P(i)$ 的倾斜角 θ，

若 $v < \theta < A$，则【$m \leftarrow i$，$A \leftarrow \theta$】

$i \leftarrow i+1$】。

(5) 若 $m=n-1$，则【输出结果，结束】。

否则，转(3)。

这个方法的诱人之处在于它容易推广到多维的情形。在"三维"的情形下，设想用平面"席卷"各点（绕着凸包上的棱）直至整个"包裹"包在内；这个方法的缺点是在最坏情况下执行时间正比例于 n^2。

2. Graham 扫描法

R. L. Graham 于 1972 年提出的另一算法，是对上述卷包裹法的一种改进，它的特点是在进行一次排序之后，就能将卷包裹的顺序确定下来，算法只需作一些少量的工作就可求得凸包。

Graham 扫描法是从凸包的一个角点开始，通常选 y 最小的点。不失一般性，令这一点为 $P_1(x_1,y_1)$，且使得闭 n 边形 $p_1p_2p_3\cdots p_np_1$ 满足 p_1p_i 线段和 x 轴正方向的夹角 θ_i，$i=1,2,\cdots,n$ 是一单调增序列。这可通过对 p_1 和其它点连线的倾斜角排序而得。

从多边形中依次对每一顶点进行检验，剔除不必要的顶点使得最后剩下的便是所求的凸包。那么什么样的点应该去除呢？先分析一下它们的特点(图 25.2.2)

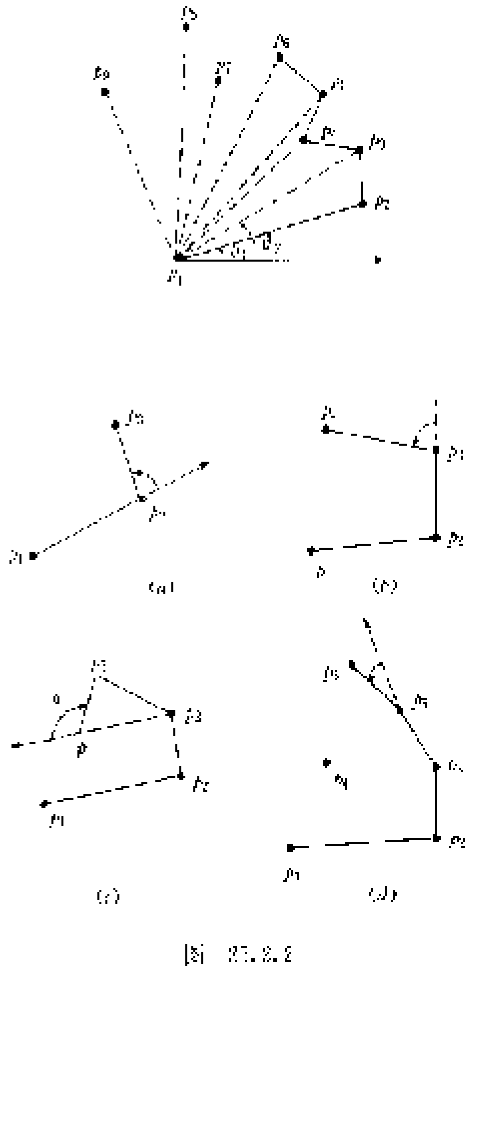


图 25.2.2

从图25.3.2(c)中看出 p_4 点加入后，从 p_3p_4 到 p_4p_5 顺时针旋转角 α，和其它点加入的情况相反，p_4 点应去除，即被剪 p_4 的 $p_3p_4p_5$，该算法描述如下：

(1) $m \leftarrow 1$，

i 从 2 到 n 作【若 $y_i < y_m$，则 $m \leftarrow i$】，

(2) i 从 1 到 n 作【若 $y_i = y_m$ 且 $x_i > x_m$，则 $m \leftarrow i$】，

(3) 以 $p(m)$ 作为极坐标原点，即作 $p(0) \leftarrow p(m)$，

(4) 对其余 $n-1$ 个点各点与 $p(0)$ 相连的线段关于极角和长度按字典序排列(若存在角度相同的两个，留下长度最大的)，并按顺序把它们连成 ω。对于点 v，指针 $R(v)$ 指向 v 的后继结点，$L(v)$ 指向 v 的前驱，$v \leftarrow p(0)$。

(5) 若 $R(v) \neq p(0)$，则

【若 v，$R(v)$，$R(R(v))$ 三点形成逆时针转动，

则【$v \leftarrow R(v)$，转(5)】，

否则，【删去 $R(v)$，$v \leftarrow L(L(v))$，转(5)】】。

否则，转(6)。

(6) 输出结果，结束。

算法第(1)、(2)步目的在于找出 y 坐标最小且最右边的点作为极坐标原点，第(5)步是判断 v 点是否应删除，还需另加说明的有：

① 若 $p_ip_{i+1}p_{i+2}$ 形成向顺时针方向旋转，则删去 p_{i+1}，进而对 $p_{i-1}p_ip_{i+2}$ 进行检测；

② $p_ip_{i+1}p_{i+2}$ 形成向逆时针方向旋转，则向前检测 $p_{i+1}p_{i+2}p_{i+3}$；

③ $v = p(0)$ 时扫描结束。

Graham 方法是"回溯法"的一个典型例子，就是说"先作尝试往下求，如果不行则作另外的尝试"。

另外，Graham 求凸包算法的计算复杂性为 $O(n\log n)$，这一点很有趣，因为这正是排序的计算复杂度，而其中求凸包过程的时间复杂度只为 $O(n)$，也就是说在该方法中排序的复杂性占主导地位。求凸包还有许多方法，它们中间不少都与排序方法存在着某种对应的关系(如卷包裹法与插入方法等对应，排序的归并法对应分治方法)，这也许不是一种巧合，但也有可能蕴含着它们之间的某种本质联系，而这已超出了本书的讨论范围，只能留给读者自己思考了。

习　题

1. 试设计一算法确定一点是否在已知凸多边形内部。
2. 试讨论在什么样的情况下，Graham 算法可能是低效率的。
3. 卷包裹法从凸包上一点开始是否完全必要？试讨论之。
4. 试证凸包内所有点间距离最长的两点必然是凸包的顶点。

第 26 章　NP 完备理论

并非所有问题都可以利用计算机来解决。可计算性理论讨论的是"什么样的问题可以利用计算机来解决，而什么样则不可"。理论上，一个可计算问题不一定是实际上可计算的。在什么情况下认为是满意地解决了？显然这个答案依据于解该问题的已知算法的实际效果。如果一个问题有一个算法，其所用时间不是不可允许的(这是应用的主要准则)，那么这个问题就认为是解决了，否则就认为是没有解决，虽然理论上是能解决。比如说一个问题可以计算，但要开动机器算 100 个世纪，就是实际上没解决的问题。不说 100 个世纪，一个世纪都不行。事实上我们已指出，决定一个算法的实际效率，要看它所需时间的增长速度，那么怎样的增长速度才认为是可被接受的呢？

现今计算机科学家们有一种共识，认为解决一个计算问题的算法，仅当其复杂性随问题规模的增加而多项式的增长时，这个算法才是实际有效的。按照这种观点，复杂性为 $O(n)$ 或 $O(n^2)$ 的算法是可以接受的(多项式的增长速度)。自然，其渐近复杂性自身不是多项式的，但它有一个多项式的上界，这样的算法也是可接受的，例如 $n^{1.5}$ 和 $n\log n$ 等。多项式算法之间还要比较 n 的幂，当算法复杂性为 $O(n^{10})$，这大概是不可接受的。它意味着运算次数为 kn^{10}，因解一个规模为 10 的例子，10^{10} 已经是一个天文数字了。所以多项式算法未必都有效，后面讨论的线性规划问题的椭球算法，理论上它是多项式算法，实际上它是不可行的，实践证明不是好算法。当然理论上证明指数型的算法，当然不是有效的了。

对于还未找到多项式算法的一类问题是计算机科学家更感兴趣的问题，其中不乏有实际背景的重要问题。它们是否存在有效的算法？经研究发现其中有一类问题性质相当，它们具有这样的性质：若有一个问题找到多项式算法，则它们全体都将得到解决；同样若证明了它们中任何一个肯定不存在有效算法，则对它们的全体也可放弃这方面的努力。1971 年 S. Cook 发表了"The Complexity of Theorem Proving Procedures"和 1972 年 R. Karp 发表的"Reducibility Among Combinatorial Problems"两篇著名的论文奠定了 NP 完备理论的基础。NP 是"Nondeterministic Polynomial"的缩写，译为"非确定型的多项式"的意思。为了说明什么是非确定型的多项式算法，先从什么是确定型的多项式算法谈起，所以要介绍确定型图灵机。

NP 完全理论不打算找出这一类问题的算法，仅着眼于证明这一类问题之等价性，以证明它们的困难程度相当。若其中一个问题获得多项式解法，则这一类问题全部获得解决，同样若能证明其中任何一问题没有"好"算法，则全体都没有。

26.1　确定型图灵机

一个问题的判定过程可形式地描述如下：

已知 $L\subseteq\{0,1\}^*$，对于 $x\in\{0,1\}^*$，若 $x\in L$，则给出答案"是"或称命题 x 为"真"，若

q_N，则给出答案"非"或命题 α 为"假"。其算法可归结为图灵机。

$\{0,1\}^*$ 指的是由有限个 0 和 1 组成的符号串集合。现实的计算机模型都可以用图灵机来模拟描述它，所以引进图灵机目的是使得我们的结果有普遍性，并简化模型。图灵机可以认为是计算机的数学模型。

定义 单带的图灵机包含有：

1) 有限状态集 Q，q_0, q_Y, q_N 属于 Q，其中 q_0 为初始状态，当图灵机进入状态 q_Y，则给出答案"是"然后停机；当图灵机进入状态 q_N，则给出答案"非"然后停机。即状态 q_0, q_Y, q_N 只是状态集 Q 的三个特殊状态。

2) 字符集 $\Sigma=\{0,1,\#\}$，其中 # 为空格。

3) 转移函数 f，

$$(Q\setminus\{q_Y,q_N\})\times\Sigma \xrightarrow{f} Q\times\Sigma\cup\{r,l\};$$

即机器处在正常的状态，即 $q\in Q\setminus\{q_Y,q_N\}$，当读到一个字符 a 时，有限状态控制器有以下三种可能的活动方式：

(1) $(q,a)\to(q',a')$，则机器从状态 q 改为 q'，而且读写头在读到的格子上写入符号 a'，读写头不动。

(2) $(q,a)\to(q',r)$，即机器从状态 q 改为 q'，读写头右移一格。

(3) $(q,a)\to(q',l)$，即机器状态从 q 改为 q'，读写头左移一格。

每当执行一次(1)、(2)或(3)就算完成"一步"计算，并为下一步进行准备。所以说图灵机是算法的一种数学模型。

单带的图灵机可以绘成图 26.1.1 所示。

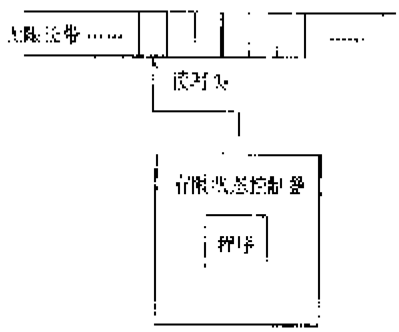


图 26.1.1

例

$Q=\{q_0,q_1,q_2,q_3,q_Y,q_N\}$

$\Sigma=\{0,1,\#\}$

f：$(q_0,0)\to(q_0,r)$、$(q_0,1)\to(q_0,r)$、$(q_0,\#)\to(q_1,l)$

$(q_1,0)\to(q_2,\#)$、$(q_1,1)\to(q_3,\#)$、$(q_1,\#)\to(q_N,l)$

$(q_2,0)\to(q_Y,r)$、$(q_2,1)\to(q_N,r)$、$(q_2,\#)\to(q_N,l)$

$(q_3,0)\to(q_N,r)$、$(q_3,1)\to(q_N,r)$、$(q_3,\#)\to(q_N,l)$

转移功能 f 可用表 26.1.1 表示，a 为输入字符。

对上述的确定型图灵机，在输入 $x=10100$ 的情形下，运行过程为图 26.1.2 所示。图的左端为状态标志，上方标以转移功能。

图 26.1.2 说明上例中确定型图灵机对输入 $x=10100$ 最后在状态 q_Y 停机。▽表示读写头。

表 26.1.1

状态 q / Γ	q_0	q_1	q_2	q_3
0	[illegible]	[illegible]	[illegible]	[illegible]
1	[illegible]	[illegible]	[illegible]	[illegible]
#	[illegible]	[illegible]	[illegible]	[illegible]

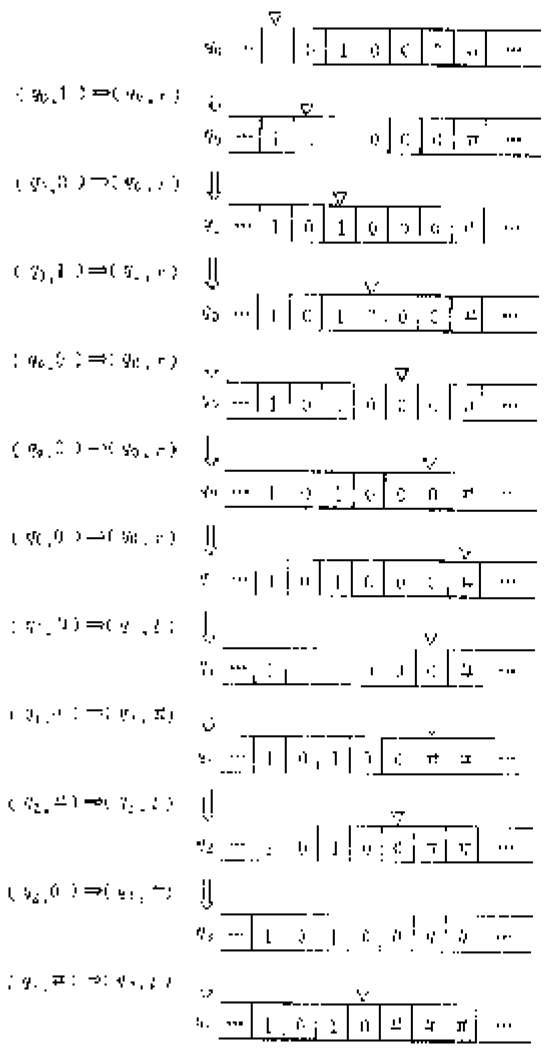


图 26.1.2

对于确定型的图灵机 TM，输入 x 后有三种可能：

(1) 图灵机 TM 接受输入 x，停机在状态 q_Y；

(2) 图灵机 TM 拒绝接受输入 x，停机在状态 q_N；

(3) 图灵机 TM 既不到达 q_Y，也不到达 q_N，则机器永不停机。

引进确定型的图灵机 TM 的计算复杂性作下一个形式化的定义。确定型的图灵机对输入 x 的时间复杂性指的是从开始到状态 q_Y 停机为止的运行步数。

设 $T(n)$ 为正整数 Z^+ 到 Z^+ 的映照，即给定正整数 n，对应一正整数 $T(n)$，如若对于任意长度 $\leqslant n$ 的输入 x，其一确定型的图灵机从初始状态 q_0 开始，在不超过 $T(n)$ 步下以接受而停机，则称机器在 $T(n)$ 时间内运行；或输入 x 在 $T(n)$ 时间内为图灵机所接受。

引进一个多项式时间问题类 P 如下：

$\mathrm{P} \triangleq \{L \mid L \subseteq \{0,1\}^*$，$L$ 为图灵机 TM 在多项式时间内所接受$\}$。即 P 为集合 $\{0,1\}^*$ 中能为图灵机在多项式时间内所接受的集合。

$\{0,1\}^*$ 表示 0 或 1 组成的符号串。

26.2 可满足性问题

已知问题 B 可在多项式时间内得到解，而问题 A 可在多项式时间内转换为问题 B，而且通过对 B 的求解得到 A 的解，可证明问题 A 也可在多项式时间内获得解：

$\pi \triangleq \{f \mid f: \{0,1\}^* \to \{0,1\}^*$ 可在多项式时间内完成$\}$。

即 π 为能在多项式时间内完成的从 $\{0,1\}^*$ 到 $\{0,1\}^*$ 转换的集合。

已知 $A \subseteq \{0,1\}^*$，$B \subseteq \{0,1\}^*$，若存在 $f \in \pi$ 使得

$$x \in A \Leftrightarrow f(x) \in B$$

则称 A 可转换为 B，并表以 $A \propto B$。

引理 若 $A \propto B$，$B \in \mathrm{P}$，则 $A \in \mathrm{P}$。

证明 $\forall x \in A \to f(x) \in B$。

设转换 f 有多项式界步数，设为 $q(|x|)$，故有 f 产生的输出的长度必须有多项式界，设为多项式 $p(x)$，即 $|f(x)| = p(|x|)$。

但求 B 的解的时间也有多项式的界，设为 h，对输入 x 先作用以 f，接着解属于 B 的问题，总共所需的时间：

$$q(|x|) + h(|f(x)|)$$

结果问题 A 转化为求 B 解，显然也是多项式的，所以 $A \in \mathrm{P}$。这个过程可用图 26.2.1 表示：

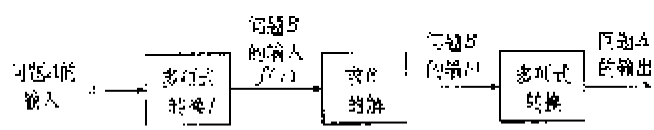


图 26.2.1

Cook 提出一个著名的“可满足性”问题，许多难解的问题都可以化为 Cook 的“可满足性”问题，从而可满足性问题可作为一类问题的难度标准。

先通过例子介绍几个必要的概念。

逻辑表达式

$$(\bar{A}\vee B\vee C)\wedge(\bar{A}\vee C\vee D)\wedge(B\vee\bar{C})\wedge(\bar{A}\vee C\vee\bar{D}) \qquad (*)$$

中 A,B,C,D 为逻辑变量，只取"真"和"伪"两个值。为方便起见用"T"表取"真"值，"F"表"伪"。$\bar{A}$ 为 A 的补，$\bar{B}$ 为 B 的补，等等。其中：

$$\bar{A}\vee B\vee C,\ \bar{A}\vee C\vee D,\ B\vee\bar{C},\ \bar{A}\vee C\vee\bar{D}$$

由逻辑变量或它们的补的逻辑和构成的项称为子句。每个子句也只取"F"和"T"两种值。

如果一个逻辑表达式 f 可表示为若干子句 $C_1,C_2,\cdots$ 的逻辑乘，即

$$f=C_1\wedge C_2\wedge\cdots\wedge C_i$$

而每一子句均为逻辑变量(或它的补)的逻辑和，则称逻辑表达式 f 的这种形式为合取范式。

只有当对所有的逻辑变量赋值后，使每个子句取值"T"，表达式 f 取值"T"。比如本例：

$$A=\mathrm{F},\ B=C=D=\mathrm{T},$$

则表达式(*)的各子句取值"T"，故表达式(*)取值 T。显然，从组合学的观点看，这问题相当于可在每一个子句中各取一文字组成一集合，使得所选取的文字集合中避免出现互补的一对。比如一个子句中选 $\bar{A}$，则另一子句中应避免选 A。

可满足性问题：

设 $L=\{A,B,\cdots,\bar{A},\bar{B},\cdots\}$，$C_1,C_2,\cdots,C_k$ 是 L 的有限子集，称为子句。每个 C_i 中不出现 L 中互补的一对(即 $x\in C_i$，则 $\bar{x}\notin C_i$)，$i=1,2,\cdots,k$。所谓可满足性问题，是确定是否存在一集合 $S\subseteq L$ 满足以下两个要求：

(1) S 中不包含互补的一对元素，即若 $x\in S$，则 $\bar{x}\notin S$；

(2) $S\cap C_i\neq\phi$，$i=1,2,\cdots,k$。

从数理逻辑看，每一个 C_i 是它所包含元素的逻辑和，若 S 中元素都取值 T，将使得 k 个子句都取值 T，因而 $C_1\wedge C_2\wedge\cdots\wedge C_k$ 取值 T。

可满足性问题通常用符号 SAT(Satisfiability 的缩写)来表示。

在介绍了可满足性问题定义之后，举一个图的着色问题例子，证明它可在多项式时间内转换为可满足性问题。

图的着色问题是这样的：已知有限图 G 及整数 m，问是否可用 m 种颜色对图 G 的顶点进行着色使得相邻的顶点有不同颜色？

定理 图的着色问题可转换为可满足性问题。

证明 已知图 G 及整数 m，令 $C_1,C_2,\cdots,C_m$ 表 m 种颜色。设

$$p_{ij}=\begin{cases}\mathrm{T}，若顶点\ i\ 着以颜色\ C_j\\ \mathrm{F}，其他\end{cases}$$

$$i=1,2,\cdots,n,\quad j=1,2,\cdots,m$$

这样，G 的每一种着色方案对应于给 mn 个逻辑变量 $\{p_{ij}\}$ 的一种指派。

但是：

(1) 每个顶点至少有一种颜色，故对于任一顶点 i，对应有子句 $P_{i1}\vee P_{i2}\vee\cdots\vee P_{im}$，$i=1,2,\cdots,n$；

(2) 相邻的顶点着不同颜色，故对图G的任意一对相邻顶点(r,s)必然有或r点不着以颜色C_j，或s点不着以颜色C_j，换句话说，对于图G的每条边(r,s)分别对应有子句$\bar{P}_{rj} \vee \bar{P}_{sj}$，$1 \leqslant j \leqslant m$，即$\bar{P}_{rj}=T$或$\bar{P}_{sj}=T$至少一个成立。可见$G$可以用$m$种颜色进行着色使相邻点不同色的充要条件是可对变量$P_{ij}$进行指派，$i=1,2,\cdots,n$；$j=1,2,\cdots,m$，使得由(1)、(2)构成的子句取值T。故图$G=(V,E)$的顶点$m$着色问题变为可满足性问题：

$$f=\bigwedge_{i=1}^{n}(P_{i1} \vee P_{i2} \vee \cdots \vee P_{im}) \wedge \bigwedge_{(r,s)\in E} \bigwedge_{j=1}^{m}(\bar{P}_{rj} \vee \bar{P}_{sj})$$

另外转换步骤(1)、(2)所需时间有多项式界，所以定理成立。

26.3 非确定型图灵机与Cook定理

1. 非确定型的图灵机

实际工作中存在一类属于组合数学研究范围的重要问题，至今尚未找到有效的算法，引起了组合数学家的兴趣。所谓有效的算法指的是在确定型的图灵机上用多项式时间获得解决的算法。为了描述这样的一类问题，引进非确定型图灵机概念如下：

非确定型图灵机完全是一种假想的机器，它和确定型的图灵机共同之点是有：

(1) 有限状态集Q，q_0,q_Y,q_N均属于Q，

(2) 输入字母表$A=\{0,1,B\}$。

不同之处在于控制功能是：

$(Q\setminus\{q_Y,q_N\})\times A \to Q\times\{A\cup\{R,L\}\}$的一个子集，也就是说它是多值的。其中符号的含义不再重复和确定型图灵机一致。

如果说确定型的图灵机在任一状态一次只能做一种运算，非确定型的图灵机则不然，它在同一时刻里可以同时做多种运算，这就是每步都有多种结果，故非确定型图灵机可以看做是多路搜索算法。有的把非确定型图灵机看做是除了多一个"猜想模块"以外，其余和确定型的图灵机结构一样，而这"猜想模块"带有"猜想头"，可对带写入猜想。若多路搜索看作是一棵树，即搜索树，其实这个"猜想头"可以看作是对多路搜索树的某一路的指引。即非确定型图灵机有两个阶段，一是猜想阶段，第二阶段是检验阶段

"猜想模块"也好，多值也好，都完全是一种假想，实际上还不存在这样的算法，可以想象每一步都要同时进行多种计算，即对于一切可能的情况并行地计算，并且各自独立地进行下去，直到一切可能都搜索完毕，非确定型图灵机方停机。

以用K种颜色对图G的n个顶点进行着色为例，猜想模块是在一切可能的K^n种方案中生成出一种方案。

第二阶段是对猜想模块提供的答案进行检验，若问题检验阶段所需的时间有多项式界，和确定型的图灵机对应有P类问题一样。下面引进NP类问题定义如下。

$NP \triangleq \{L \subseteq \{0,1\}^* \mid L$为非确定型图灵机在多项式时间内所接受$\}$。

即NP类是对解进行验证可在多项式时间内完成的一类问题。

已知图$G=(V,E)$，求哈密顿回路问题便是NP问题，因验证是否哈密顿回路可在多项式时间内完成，而最短的哈密顿回路(即流动售货员问题)并不是NP类问题，它不能在

在限时间内完成验证且为最短。

这里所谓"……在多项式时间内所接受"说的是对于输入 x，机器从初始状态 q_0，经过一系列的转换到达 q_f 停机状态，其步骤不超过 $T(|x|)$，$T(|x|)$ 是一个关于 $|x|$ 的多项式。

为方便起见可用符号 DTM 表示确定型图灵机，用 NDTM 表示非确定型图灵机。直观上容易理解为非确定型图灵机比确定型的图灵机功能要强。属于 DTM 的问题，必然属于 NDTM，即可用 DTM 在多项式时间内解决的问题也一定能用 NDTM 在多项式时间内解决，反之则不尽然。故

$$P \subseteq NP$$

设某 NDTM 的字母表 A 的元素个数为 K，即 $K=|A|$。对于输入 x，其长度 $|x|=n$，若 NDTM 在不超过 $p(n)$ 步内给出肯定的判定，则在第一阶段给出的"猜想"长度不超过 $p(n)$，每位的字母有 k 种可能，故"猜想"的全体不超过 $k^{p(n)}$ 个，每个猜想在 $p(n)$ 步内检验完毕，故用 DTM 串行方式进行，全体检验完毕的时间复杂性以

$$p(n)k^{p(n)}$$

为上界，这说明在 NDTM 上时间复杂性为 $p(n)$ 的判定问题可化为 DTM 上时间复杂性不超过 $p(n)k^{p(n)}$ 的问题相当。

2. Cook 定理

Cook 定理是 NP 完备理论的重要支柱之一。定理的证明用到数理逻辑的一个结论，即任一逻辑表达式可以通过下面几个基本法则化为合取范式，即表为若干子句之逻辑积

(1) $A\leftrightarrow B=(A\to B)\wedge(B\to A)$,

$A\to B=\bar{A}\vee B$

(2) $\bar{\bar{A}}=A,\ \overline{A\vee B}=\bar{A}\wedge\bar{B},\ \overline{A\wedge B}=\bar{A}\vee\bar{B}$

(3) $A\vee(B\wedge C)=(A\vee B)\wedge(A\vee C)$

例 $A\vee\overline{(B\vee C)}\vee D=(A\vee(\bar{B}\wedge\bar{C}))\vee D$

$=((A\vee\bar{B})\wedge(A\vee\bar{C}))\vee D$

$=D\vee((A\vee\bar{B})\wedge(A\vee\bar{C}))$

$=(D\vee(A\vee\bar{B}))\wedge(D\vee(A\vee\bar{C}))$

$=(D\vee A\vee\bar{B})\wedge(D\vee A\vee\bar{C})$

Cook 定理 若 $L\in NP$ 类，则 $L\propto SAT$。

证 由于可满足性问题属于 NP 类，故只要证任一属于 NP 的问题可以转换为可满足性问题，其时间有多项式的界。也就是证明 NDTM 在多项式界时间内予以接受的任一问题，可以通过多项式界时间化为可满足性问题。

设某一 NDTM，记为 M，在 $T(n)$ 多项式界内对输入 x 进行识别，只要找一有多项式界的方法把输入符号串 x 转换为一组子句 $f(x)$，而且

$$M \text{ 接受 } x \Leftrightarrow f(x) \text{ 为可满足的}$$

为了一般性，设 NDTM 的有限状态集合 Q：

$$Q=\{q_1,q_2,\cdots,q_s\}$$

设其中 q_1 为初始状态，q_2 为 q_Y，q_3 为 q_N。其字母表为：

$$A=\{a_1,a_2,\cdots,a_m\}$$

这其中 a_0 为空格。

机器在不超过 N 步内接受 x，每走一步，读写头至多左移一格或右移一格，故读写头的活动范围不超过以初始位置为中心的左右各 N 格，共 $2N+1$ 格。每行的 $2N+1$ 个方格上的符号、机器的当时状态，以及读写头的位置，这些信息完整地描绘了机器的瞬间图象。

t 从 1 到 N，各个时刻机器的瞬间图象是被接受的 x 的全部信息。

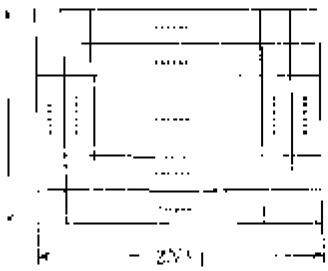


图 26.3.1

如图 26.3.1 所示，引进 $N\times(2N+1)$ 的方格，图中第 t 行记录 t 时刻带上的符号串，第 t 行第 j 列的方格用 (t,j) 表示。为了描述机器运行的全部过程，特引进以下命题变量：

$$A(i,t,j)=\begin{cases}T, & 若(t,j)的符号为 a_i\\ F, & 其他\end{cases}$$

$$H(t,j)=\begin{cases}T, & 若 t 时刻读写头在第 j 格\\ F, & 其他\end{cases}$$

$$S(t,k)=\begin{cases}T, & 若 t 时刻机器的状态为 q_k\\ F, & 其他\end{cases}$$

其中 $1\leqslant t\leqslant N$，$1\leqslant j\leqslant 2N+1$，$1\leqslant i\leqslant m$，$1\leqslant k\leqslant l$。

(1) 初始输入符号串为 $a_{i_1}a_{i_2}\cdots a_{i_n}$，即图 26.3.1 中第一行为：

$$\underbrace{空空\cdots空}_{N}\, a_{i_1}a_{i_2}\cdots a_{i_n}\,\underbrace{空空\cdots空}_{N-n}$$

可由子句联结如下：

$$\bigwedge_{j=1}^{N} A(0,1,j)\wedge\bigwedge_{j=N+1}^{N+n} A(i_{j-N},1,j)\wedge\bigwedge_{j=N+n+1}^{2N+1} A(0,1,j) \tag{26.3.1}$$

(2) 初始状态为 q_1，时刻 1 读写头位于第 $N+1$ 格，对应子句为：

$$S(1,1)\wedge H(1,N+1) \tag{26.3.2}$$

(3) 对于每一 t，$1\leqslant t\leqslant N$，机器有一种状态，而且只有一种状态，对应的子句为：

$$\bigwedge_{t=1}^{N}\Big(\big(\bigvee_{k=1}^{l} S(t,k)\big)\wedge\big(\bigwedge_{1\leqslant k<k'\leqslant l}\overline{(S(t,k)\wedge S(t,k'))}\big)\Big) \tag{26.3.3}$$

由于 $\overline{A\wedge B}=\bar{A}\vee\bar{B}$，故得对应子句：

$$\bigwedge_{t=1}^{N}\Big(\big(\bigvee_{k=1}^{l} S(t,k)\big)\wedge\bigwedge_{1\leqslant k<k'\leqslant l}\big(\overline{S(t,k)}\vee\overline{S(t,k')}\big)\Big) \tag{26.3.4}$$

(4) 对于 $1\leqslant t\leqslant N$，$1\leqslant j\leqslant 2N+1$ 的任一方格 (t,j) 有一个字符，且仅有一个字符，和 (26.3.3) 类似，对应的子句为：

$$\bigwedge_{t=1}^{N}\bigwedge_{j=1}^{2N+1}\Big(\big(\bigvee_{i} A(i,t,j)\big)\wedge\bigwedge_{0\leqslant i<i'\leqslant m}\big(\overline{A(i,t,j)}\vee\overline{A(i',t,j)}\big)\Big) \tag{26.3.5}$$

(5) 对于 $1\leqslant t\leqslant N$ 的任何时刻，读写头在一个方格上且仅在一个方格上，对应的子句为：

$$\bigwedge_{t=1}^{N}\left(\left(\bigvee_{i=1}^{2N} H(t,i)\right) \wedge \bigwedge_{1\le i_1<i_2\le 2N}\left(\overline{H(t,i_1)} \vee \overline{H(t,i_2)}\right)\right) \qquad (26.3.6)$$

(6) 从时刻 t 到时刻 $t+1$，只有 t 时刻读写头所在的方格，它的符号才有可能改变，即图 26.3.1 中第 t 和 $t+1$ 行，只有使 $H(t,i)=\mathrm{T}$ 的第 i 格符号可能不同，其余不变。

$A(t+1,i,j)=\mathrm{T}$ 表示在 $t+1$ 时刻第 i 个方格的字符为 a_j，它和 t 时刻的机器图象的关系有如下两种可能：一是 t 时刻，第 i 个方格的字符本来就是 a_j，从 t 时刻到 $t+1$ 时刻没有变化，即

$$A(t,i,j)=\mathrm{T}$$

如若不然，即 $A(t,i,j)=\mathrm{F}$，这时必然在 t 时刻读写头位置在第 i 个方格，用子句表示如下：

$$A(t,i,j) \leftrightarrow A(t+1,i,j) \vee H(t,i) \qquad (26.3.7)$$

但由

$$P \leftrightarrow Q = (P \wedge Q) \vee (\overline{P} \wedge \overline{Q})$$

而 $(P\wedge Q)\vee(\overline{P}\wedge\overline{Q})=(\overline{P}\vee Q)\wedge(P\vee\overline{Q})$

类似的理由，(26.3.7)可通过多项式的时间化为关于 $A(t,i,j)$、$A(t+1,i,j)$、$H(t,i)$ 及其补的合取范式。

(7) 对于 t 时刻机器的状态 $q\in Q\setminus\{q_Y,q_N\}$，则 $t+1$ 时刻机器状态、读写头所在的位置，以及原来 t 时刻读写头所在方格内的符号的改变，这三者均服从 NDTM 的控制功能，对应有：

$$\bigwedge_{t=1}^{N}\bigwedge_{i=1}^{2N}\bigwedge_{j}\bigwedge_{k}\left(\overline{A(t,i,j)} \vee \overline{H(t,i)} \vee \overline{S(t,k)} \vee \left(\bigvee\left(A(t+1,i,j') \wedge S(t+1,k') \wedge H(t+1,i')\right)\right)\right) \qquad (26.3.8)$$

式(26.3.8)中括号中的 $\bigvee$ 作为约定是指对所有可能的转移功能进行的，也就是说读写头位置 i、字符 j、状态 k 服从 NDTM 的转移功能 f 或 $(i',j',k')=f(i,j,k)$。

通过多项式界时间可将(26.3.8)化为合取范式。对(26.3.8)分析如下：对于 $t=1,2,\cdots,N$，$i=1,2,\cdots,2N+1$，$k=1,2,\cdots,l$，$j=1,2,\cdots,m$，$A(t,i,j)$、$H(t,i)$、$S(t,k)$ 有两种可能：一是全部满足 t 时刻的机器状态，则

$$\overline{A(t,i,j)} \vee \overline{H(t,i)} \vee \overline{S(t,k)} = \mathrm{F}; \qquad (26.3.9)$$

另一种可能 A、H、S 中至少有一个与机器在 t 时刻的状态不符合，这时

$$\overline{A(t,i,j)} \vee \overline{H(t,i)} \vee \overline{S(t,k)} = \mathrm{T}; \qquad (26.3.10)$$

而

$$\bigvee\left(A(t+1,i,j') \wedge S(t+1,k') \wedge H(t+1,i')\right) \qquad (26.3.11)$$

则描述 $t+1$ 时刻机器的可能状态。

若(26.3.9)成立，表达式(26.3.11)取"T"值，正好表达了 t 时刻到 $t+1$ 时刻机器状态的转变。否则(26.3.10)成立。

(8) 在 $t=N$ 时 NDTM 接受 x 而停机，有子句 $S(N,2)$。

对以上一组子句作 $\wedge$ 运算得一合取范式，该逻辑表达式为可满足的充要条件是 x 为 M 所接受。把输入符号串 x 化为这一组子句的过程，其时间复杂性有多项式界，这就证明

了所有 NP 类问题都可以化为可满足性问题。 证毕。

推论 P = NP 的充要条件是 SAT∈P。

因所有 NP 问题都可化为可满足性问题，故可满足性问题若有多项式界的解法，则所有 NP 类问题都将有多项式解法。故 SAT 是 NP 类中最难的问题。属于 NP 类中某问题 N，所有其它 NP 类问题都可在多项式时间内转换为该问题 N，则称问题 N 为 NP 完备，或称之为 N 属于 NPC。SAT 属于 NPC。NPC 后面的 C 是 *Complete* 的缩写。

26.4 几个 NP 完备的例子

从 Cook 定理及其重要推论可知，只要可满足性问题存在多项式解法，则所有 NP 类问题均可获得多项式解法，可满足性问题算是 NP 类中最困难的问题，称之为 NP 完备问题，或者称为 NPC 问题。

粗略地说，NPC 问题是 NP 类中最困难的问题，在 NP 类中没有比它更困难的了。可满足性问题可作为 NPC 问题的鉴定标准。

若要证明一问题属于 NP 完备，必须证它属于 NP，并证明某一已知的 NP 完备问题可在多项式时间内将它转换为该问题，则该问题便属于 NP 完备。NP 完备有时也称 NP 完全。

具体地说，要证明一个问题 P_1 属于 NP 完备，首先必须证明问题 P_1 属于 NP 类；其次要证明对一个已知属于 NPC 的问题 P_2 可转换为问题 P_1，最后证明这个转换过程有多项式界。

若问题 $L \in$ NP，所有 NP 问题都可以通过多项式时间转换为 L，则 L 称为属于 NP 完备类。

1. 团的问题

若完全图 G_1 是图 G 的子图，则称图 G_1 是图 G 的团。

例如图 26.4.1(a) 存在 2 个 3 个顶点的团，但没有超过 3 个顶点的团。(b) 有 4 个顶点的团。

团的问题：已知图 G 和整数 k，试判定图 G 是否存在 k 个顶点的团？

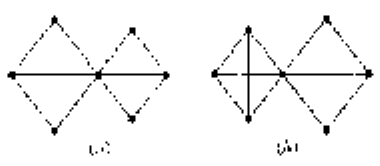


图 26.4.1

定理 团的问题属于 NPC。

证 可满足性问题可化为团的问题，办法如下：令图的顶点分别对应于表达式中出现的文字，下面条件成立的两顶点用边连结起来。

(1) 两个顶点对应的文字不出现于同一子句；

(2) 不相互补。

例如图 26.4.2 中第一子句的 A 与第 2 子句的 B 和 C 以及第 3 子句的 C 联以边，第一子句的 B 与第 2 子句的 C 和第 3 子句的 C、A 联以边，等等。可得图 26.4.3，对逻辑变量 A，B，C 的指派对应一求 $K=3$，图 26.4.3 有团。故存在 k 个顶点为团的充要条件是使

至少有k个逻辑变量子句集合是可满足的。显然，团的问题属于NP类，从SAT转换为团的问题可在多项式时间完成，故团的问题是NP完全问题。

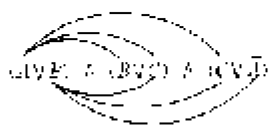


图 26.4.2

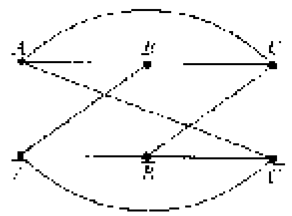


图 26.4.3

2. 3SAT问题

对于合取范式：

$$F = C_1 \wedge C_2 \wedge \cdots \wedge C_n$$

若每一子句C_i所含的文字数目都是3时，它的可满足性问题便称为3SAT问题。一般的SAT问题可以转换为3SAT问题。例如子句：

$$A \vee B \vee C \vee D \tag{26.4.1}$$

可引进新的变量x，使得：

$$(A \vee B \vee X) \wedge (\bar{X} \vee C \vee D) \tag{26.4.2}$$

式中每一个子句都只有三个文字，而且(26.4.1)和(26.4.2)等价，即满足式(26.4.1)必使满足(26.4.2)；反之，不论x被指派什么值，(26.4.2)被满足，(26.4.1)也得到满足。

一般说来，与上述例相类似，可把一个逻辑变量多于3的析取子句分解成若干子句，使得每个子句的全部逻辑变量都是3个，这样的合取范式的可满足性问题，称为3元可满足性问题，用3SAT表示。

定理 3SAT∈NPC

证明 因为3SAT是SAT的特殊情况，所以它属于NP。为了证明NP完全性，我们将证明可满足性问题多项式变换到3SAT。考虑由子句$C_1, C_2, \cdots, C_n$组成的任意表达式F，构造一个新的每个子句只有三个文字的表达式F'，使得F和F'等价。我们逐句检查F的子句，把每个C_i换成一个等价的子句集合，每个子句有三个文字。

我们分三种情况讨论：

(1) 若C_i有三个文字，则C_i不变。

(2) 若C_i有三个以上的文字，比方说$C_i = (\lambda_1 \vee \lambda_2 \vee \cdots \vee \lambda_k)$，$k>3$，我们把$C_i$换成$k-2$个子句$(\lambda_1 \vee \lambda_2 \vee x_1) \wedge (\bar{x}_1 \vee \lambda_3 \vee x_2) \wedge (\bar{x}_2 \vee \lambda_4 \vee x_3) \wedge \cdots \wedge (\bar{x}_{k-3} \vee \lambda_{k-1} \vee \lambda_k)$，其中$x_1, x_2, \cdots, x_{k-3}$是新变量。不难看出，这些新子句是可满足的当且仅当$C_i$是可满足的。

(3) 若$C_i = \lambda$，我们就用$\lambda \vee y \vee z$代替C_i；若$C_i = \lambda \vee \lambda'$，则用$\lambda \vee \lambda' \vee y$代替$C_i$。然后我们给公式添加上一些子句：

$$(\bar{z} \vee \alpha \vee \beta) \wedge (\bar{z} \vee \bar{\alpha} \vee \beta) \wedge (\bar{z} \vee \alpha \vee \bar{\beta}) \wedge (\bar{z} \vee \bar{\alpha} \vee \bar{\beta})$$
$$\wedge (\bar{y} \vee \alpha \vee \beta) \wedge (\bar{y} \vee \bar{\alpha} \vee \beta) \wedge (\bar{y} \vee \alpha \vee \bar{\beta}) \wedge (\bar{y} \vee \bar{\alpha} \vee \bar{\beta})$$

其中y, z, α, β都是新变量。这附加部分迫使变量z和y在任何满足F'的真值分配中都是

假的，从而子句 λ 和 $\lambda\vee\lambda'$ 分别等价于它们的替代部分。

于是消去了所有非三个文字的子句，也证明了，所得到的 F' 是可满足的当且仅当 F 是可满足的。

显然构造 F' 的过程可以在多项式时间内完成，因此前面所描述的就是一个由可满足性到 3SAT 的多项式转换。所以 3SAT 是 NP 完备的。

3. 图的着色问题

一个含有 n 个变量，m 个子句，每句不超过 3 个文字的合取范式可满足与否，可化为图 G 可否用 $n+1$ 种颜色进行着色使相邻的顶点不同色，故上述图的着色问题是 NPC。

设表达式有 m 个子句：

$$c_1,c_2,\cdots,c_m$$

n 个逻辑变量：

$$x_1,x_2,\cdots,x_n$$

及其补

$$\bar{x}_1,\bar{x}_2,\cdots,\bar{x}_n$$

下面假定 $n\geq 4$，可按下面叙述的不超过多项式时间的步骤，构造一个图 G，使得图 G 可用 $n+1$ 种颜色着色的充要条件是可满足性成立：

设图 G 的顶点 V 为：

$c_1,c_2,\cdots,c_m,x_1,x_2,\cdots,x_n,\bar{x}_1,\bar{x}_2,\cdots,\bar{x}_n,y_1,y_2,\cdots,y_n$，其中 $y_1,y_2,\cdots,y_n$ 是新引进的顶点。

设 G 的边 E 为：

$(x_i,\bar{x}_i)$，$i=1,2,\cdots,n$

(x_i,y_j)，$i\neq j$，$i,j=1,2,\cdots,n$

$(\bar{x}_i,y_j)$，$i\neq j$，$i,j=1,2,\cdots,n$

(y_i,y_j)，$i\neq j$，$i,j=1,2,\cdots,n$

(x_i,c_j)，若 $x_i\notin c_j$，$i=1,2,\cdots,n$，$j=1,2,\cdots,m$

$(\bar{x}_i,c_j)$，若 $\bar{x}_i\notin c_j$，$i=1,2,\cdots,n$，$j=1,2,\cdots,m$

设 $k=n+1$，给定了 k 种颜色，首先因 $y_1,y_2,\cdots,y_n$ 形成了完全图，$y_1,y_2,\cdots,y_n$ 中不能有两个顶点有相同颜色。

y_1 点着以第 1 种颜色；

y_2 点着以第 2 种颜色；

……

y_n 点着以第 n 种颜色；

余下第 $n+1$ 种颜色。由于除非 $i=j$，否则 x_i 和 y_j 不可同色，$\bar{x}_i$ 和 y_j 也一样，故补充着色如下。

从 x_k 和 $\bar{x}_k$ 中取一个令它为 $x_k^{(t)}$，$k=1,2,\cdots,n$

即取 $x_k^{(t)}$ 为 x_k 和 $\bar{x}_k$ 中任何一个。令

$x_1^{(t)}$ 着以第 1 种颜色；

$x_2^{(t)}$ 着以第 2 种颜色；

……

$x_n^{(n)}$ 着以第 n 种颜色，$x_n^{(n)}\in\{x_n,\bar{x}_n\}$

$\bar{x}_1^{(1)},\bar{x}_2^{(2)},\cdots,\bar{x}_n^{(n)}$ 着以第 $n+1$ 种颜色

每一个 c_i 不可以着以与它所对应的子句不含有的文字相同的颜色。具体地说，若 $x_j\notin c_i$，则 c_i 不能着与 x_j 相同的颜色。

其次，由于假定 $n\geq4$，而每一个 c_i 至多含有 3 个文字，所以至少存在一对 $(x_j,\bar{x}_j)$ 和 c_i 都相邻，即存在 (x_j,c_i)、$(\bar{x}_j,c_i)$ 边，所以每一个 c_i 都不能着以第 $n+1$ 种颜色。

由上可知 c_i 只能着与它所含的文字（比如 x_j 或 $\bar{x}_j$）相同的颜色，而且排除是第 $n+1$ 种颜色的可能性。如若着第 $n+1$ 种颜色的文字指派以值 F，其它指派以值 T，这就证明了所构造的图 G 能否用 $n+1$ 种颜色着色使相邻顶点不同色的充要条件是公式是否满足可满足性。

$n=3$ 时，(x_1,x_2,x_3) 共有 8 种可能的情况，可直接检验是否具有可满足性。

4. 独立集

图 $G=(V,E)$，S 是顶点 V 的子集，若集合 S 中的任意两个顶点都不相邻，则称 S 为图 G 的独立集。

独立集问题：已知图 $G=(V,E)$，$k\leq|V|$，求一独立集 $S\subseteq V$，使得 $|S|\geq k$？

定理 独立集问题属于 NP 完备问题。

证明 可证明团的问题∝独立集问题。因独立集问题属于 NP 类。

已知图 $G=(V,E)$ 及 k，作出 G 的补图 $\bar{G}=(V,\bar{E})$，其 $u\in V$，$v\in V$，若 $(u,v)\notin E$，则 $(u,v)\in\bar{E}$。

$S\subseteq V$ 是 G 的团的充要条件是：对于图 $\bar{G}$，S 是独立集，故团的问题∝独立集问题。

5. 顶点覆盖问题

图 $G=(V,E)$，若 C 是 V 的子集，G 的任一条边至少有一个顶点属于 C，则称 C 为图 G 的顶点覆盖。

顶点覆盖问题：已知图 $G=(V,E)$，正整数 $k\leq|V|$，问 G 是否存在 k 顶点的子集 $V'\subseteq V$，使得对每一条边 $e=(u,v)\in E$，u,v 中至少有一个属于 V'，而且 $|V'|\leq k$？

定理 顶点覆盖问题属于 NP 完备。

证明 顶点覆盖问题显然属于 NP 类，只要证独立集问题∝顶点覆盖问题。

已知图 $G=(V,E)$ 及整数 k，令 $l=|V|-k$。

如若有一独立集 S，使得 $|S|\geq k$，显然 $V\setminus S$ 是图 G 的顶点覆盖，$V\setminus S$ 的顶点数为 $|V|-|S|\leq|V|-k=l$。

反之，若 C 是图 G 的顶点覆盖集，$|C|\leq l$，则 $V\setminus C$ 是独立集，且 $V\setminus C$ 的顶点数为 $|V|-|C|\geq|V|-l=k$。证毕。

6. 哈密顿道路问题

已知有向图 $G=(V,E)$ 及两个顶点 u、v，求是否存在以 u 为始点，以 v 为终点遍历各顶点各一次的哈密顿道路？称之为有向图哈密顿道路问题。

定理 有向哈密顿道路问题属于 NP 完备

证明 只要证 k 个顶点覆盖问题∝有向哈密顿道路问题，后者属于 NP 类，验证是否

哈密顿道路可在多项式时间内完成。

已知无向图 $G=(V,E)$，k 是正整数，与 $v\in V$ 点相关联的边（即以 v 点为一端点的边）的数目设为 d_v，与 v 点关联的边记为 $e_1^v,e_2^v,\cdots,e_{d_v}^v$，现构造一有向图 $G'=(V',E')$ 如下：

顶点集合 V' 有以下两个组成部分：

(1) 新增加的 $k+1$ 个顶点：

$$a_0,a_1,\cdots,a_k$$

(2) 对于任一顶点 $v\in V$，对应有 $2d_v$ 个顶点：

$$v_1^{(1)},v_1^{(2)},v_2^{(1)},v_2^{(2)},\cdots,v_{d_v}^{(1)},v_{d_v}^{(2)}$$

边集合 E' 的组成部分有：

(1) 有向边 $(a_i,v_1^{(1)})$，$0\leqslant i<k$，$v\in V$；

(2) 有向边 $(v_{d_v}^{(2)},a_i)$，$1\leqslant i\leqslant k$，$v\in V$；

(3) 对于图 G 中相邻的顶点 u 和 v，$e_i^u=e_j^v$，则有：

$$(u_i^{(1)},v_j^{(1)}),(u_i^{(2)},v_j^{(2)}),(v_j^{(1)},u_i^{(1)}),(v_j^{(2)},u_i^{(2)})$$

(4) 对于 $v\in V$，对应有边：

$$(v_i^{(1)},v_i^{(2)}),1\leqslant i\leqslant d_v$$

$$(v_i^{(2)},v_{i+1}^{(1)}),1\leqslant i<d_v$$

故对应于顶点 $v\in V$，存在一条道路

$$H_v: v_1^{(1)}\to v_1^{(2)}\to v_2^{(1)}\to v_2^{(2)}\to\cdots\cdots\to v_{d_v}^{(1)}\to v_{d_v}^{(2)}$$

对应于边 $e_i^u=e_j^v=(u,v)$ 的两端点 u、v，道路 H_u 和 H_v 间存在边：

$$(u_i^{(1)},v_j^{(1)}),(u_i^{(2)},v_j^{(2)})$$

$$(v_j^{(1)},u_i^{(1)}),(v_j^{(2)},u_i^{(2)})$$

则边 (u,v) 对应一由 4 个顶点

$$u_i^{(1)},u_i^{(2)},v_j^{(1)},v_j^{(2)}$$

组成的子图，如图 26.4.4

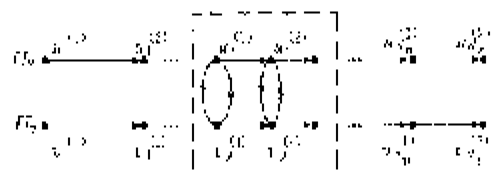


图 26.4.4

若在哈密顿道路进入 $u_i^{(1)}$ 点，则必须从 $u_i^{(2)}$ 点退出这子图。如若从 $v_j^{(2)}$ 点出，则始终无法经过 $v_j^{(1)}$ 点。因 $u_i^{(1)}$ 进入，由 $u_i^{(2)}$ 退出有两种可能，一是经过这 4 个顶点，一个是从 $u_i^{(1)}\to u_i^{(2)}$ 退出这子图。

从图 26.4.4 可见若图 G 中有 $(u,v)\in E$，则 G' 图中存在从 a_i 到 a_j 的道路：

$a_i\to u_1^{(1)}\to u_1^{(2)}\to\cdots\to u_i^{(1)}\to v_j^{(1)}\to v_j^{(2)}\to u_i^{(2)}\to\cdots\to u_{d_u}^{(1)}\to u_{d_u}^{(2)}\to a_j$

和 G

$a_1 \to v^{(1)} \to v^{(2)} \to \cdots \to v_k^{(1)} \to u_1^{(1)} \to u_2^{(1)} \to u^{(2)} \to \cdots \to v_k^{(1)} \to u_k^{(1)} \to a_k$

图 G 存在有哈密顿道路的必要条件是图 G 有 k 个顶点的顶点覆盖。设

$$C = \{u_1, u_2, \cdots, u_k\}$$

是图 G 的顶点覆盖，可构造图 G' 的从 a_0 到 a_k 的哈密顿道路，步骤如下：

(1) 构造一条从 a_0 到 a_k 的哈密顿道路 P，见图 26.4.5；

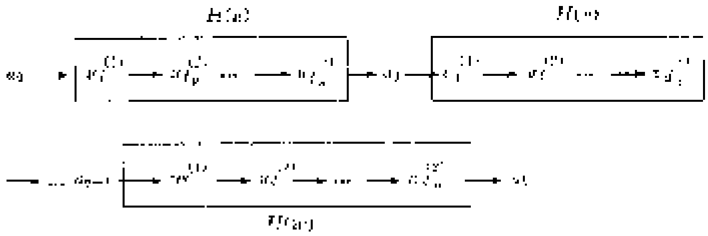


图 26.4.5

(2) 对于边 $(u,v)\in E$，若点 u 属于顶点覆盖集 C，但 v 不属于 C，则道路 P 中 H_u 的一段为绕道通过 $v^{(j)}$ 和 $v^{(j+1)}$ 两点一段所取代，其中 $(u,v)=e_i^u=e_j^v$，即 H_u 中 $u^{(i)}\to u^{(i+1)}$ 的一段改造为 $u^{(i)}\to v^{(j)}\to v^{(j+1)}\to u^{(i+1)}$，从而吸收了 $v^{(j)}$，$v^{(j+1)}$ 两个点。因 C 是顶点覆盖集，故可吸收 G' 的所有的顶点，与 u 点在 v 点通过 e 边相邻，从而吸收 $v^{(j)}$，$v^{(j+1)}$ 一样。反之，可从 G' 构造图 G 的顶点覆盖集 C。

由于哈密顿道路通过所有的顶点，包括所有的 a_i，$0\le i\le k$，已知 G' 中不存在从 a_i 到 a_j 的边，故哈密顿道路可以分解为若干段，从某一点 a_i 到另一点 a_j 的道路，道路中间不存在其它 a_l 点。对于每一条从 a_i 到 a_j 的道路对应一顶点 $v\in V$，使得道路中的顶点若不是 $v^{(\cdot)}$ 或 $u^{(\cdot)}$ 便是 $u^{(\cdot)}$ 或 $v^{(\cdot)}$，而 u 是 v 的相邻顶点，这样的 k 个顶点便是图 G 的顶点覆盖集。证毕。

图 26.4.6 给出一个 $k=2$ 的实例，G' 图中粗线标出从 a_0 到 a_2 的哈密顿道路。

推论 有向图 $G=(V,E)$ 是否存在有向的哈密顿回路的问题属于 NP 完备。

证 若能有向的哈密顿回路问题归约有向的哈密顿道路，只要对于有向图 $G=(V,E)$ 及两点 u 和 v，增加一点 s，及从 s 到 u 的边 (s,u) 和由 v 到 s 的边 (v,s) 得图 G'。显然图 G' 是否存在哈密顿回路的充要条件是图 G 是否存在从 u 到 v 的哈密顿道路。

定理 无向图 $G=(V,E)$ 是否存在哈密顿道路问题属于 NP 完备。

证明 只要将有向图的哈密顿道路问题变为无向图哈密顿道路问题。设已知有向图 $G=(V,E)$ 及两顶点 s,t，构造一无向图 $G'=(V',E')$，如下：

$$V'=\{v',v'',v'''\mid v\in V\}$$

$$E'=\{(v',v''),(v'',v''')\mid v\in V\}\cup\{(u''',v')\mid (u,v)\in E\}$$

即图 G 中每一顶点 v，对应在图 G' 上联接三个顶点 v',v'',v''' 的道路。当 u,v 两点在图 G 中是邻接点时，u''' 和 v' 相连。故当图 G 存在一条从 s 到 t 的哈密顿道路时，图 G' 存在一条从 s' 到 t''' 的哈密顿道路 P'，P' 包含了所有的边 (v',v'')，(v'',v''')，$v\in V$，从及属于哈密

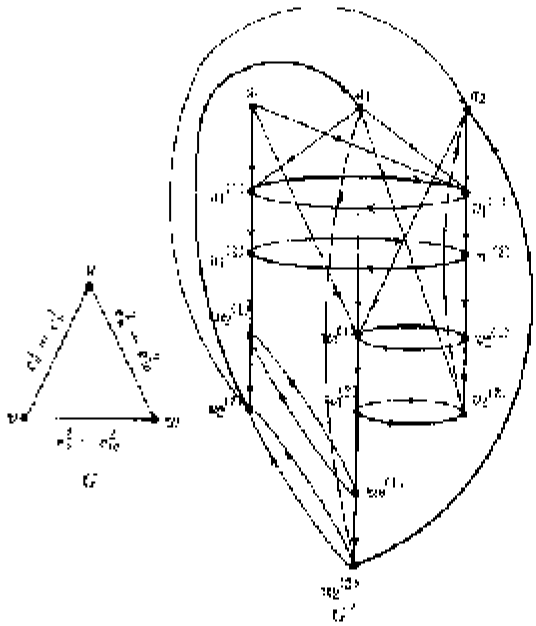


图 26.4.6

顿道路上的边(u,v)所对应的(u'',v')边，见图 26.4.7。

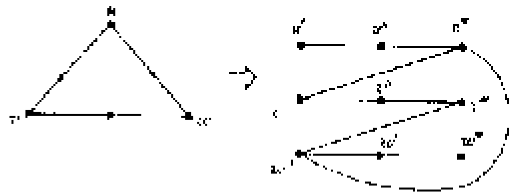


图 26.4.7

反之，图G'存在的哈密顿回路对应一有向图G的哈密顿道路。

7. 背包问题

已知一整数序列$S=\{a_1,a_2,\cdots,a_n\}$，其中$a_i\in N$，$i=1,2,\cdots,n$。给定$b\in N$，问是否存在一子集$S'\subseteq S$使得S'中元素之和等于b，即使得：

$$\sum_{a_i\in S'} a_i = b$$

背包问题显然属于NP类。下面证顶点覆盖问题可以转换为背包问题，从而证明背包问题属于NP完备。

设图$G=(V,E)$，存在一顶点集合$V'\subseteq V$，$|V'|=k$，使得任一边$(u,v)\in E$，必然有或$u\in V'$或$v\in V'$。即V'是图G的顶点覆盖；可以构造一整数序列$S=\{a_1,a_2,\cdots,a_n\}$，及$b\in N$的背包问题，使得图G有$|V'|=k$的顶点覆盖V'的充要条件是存在$S'\subseteq S$，使得

$$\sum_{a \in S} a = b$$

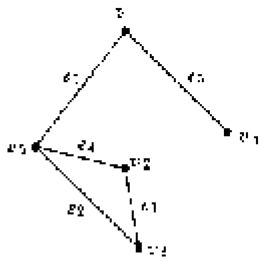
假设

$$B = (b_{ij})_{m \times n}$$

其中 $m=|E|, n=|V|$。

$$b_{ij} = \begin{cases} 1, 或 e_i 以 v_j 为端点 \\ 0, 其他 \end{cases}$$

下面通过一个例子来说明具体方法，见图 25.4.8



$$B = \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$
$$\quad\quad e_1 \ \ e_2 \ \ e_3 \ \ e_4 \ \ e_5$$

图 25.4.8

利用矩阵 B 的每一行构造一个 4 进制数如下，例如对矩阵 B 的第 1 行有：

$$(00101)_4 = 4^2 + 1 = 17$$

现构造序列 S 如表 26.4.1 所示。

表中虚线包围起来部分正好是 G 的关联矩阵 B。显然此表的各行分别由 5 个顶点 v_1, v_2, v_3, v_4, v_5 和 5 条边 e_1, e_2, e_3, e_4, e_5 来确定。一般地

$$a_i = 4^n + \sum_{j=1}^{m} b_{ij} 4^{j-1}, \quad i = 1, 2, \cdots, n$$

$$b_j = 4^{j-1}, \quad j = 1, 2, \cdots, m$$

$$S = \{a_1, a_2, \cdots, a_n, b_1, b_2, \cdots, b_m\}$$

点 v_1, v_2, v_5 为图 G 的顶点覆盖。表 26.4.1 中去掉 a_3, a_4 和 b_1 行使各列的和不产生向高位进位的情况，从而有去掉 a_3, a_4, b_1 行后各列的和依次为：

$$3, 2, 2, 2, 2, 2$$

本例中与 e_i 对应的 b_i 均端点都在顶点覆盖集上，第一个数 3 正好是覆盖集的规模。

$$b = (322222)_4 = 3754$$

表 26.4.1

	4^5	4^4 (e_1)	4^3 (e_2)	4^2 (e_3)	4^1 (e_4)	4^0 (e_5)	
a_1 =	1	0	0	1	0	1	=1041
a_2 =	1	1	0	0	1	0	=1284
a_3 =	1	1	1	0	0	0	=1344
a_4 =	1	0	0	1	0	0	=1040
a_5 =	1	0	1	0	1	1	=1093
b_1 =	0	1	0	0	0	0	=256
b_2 =	0	0	1	0	0	0	=64
b_3 =	0	0	0	1	0	0	=16
b_4 =	0	0	0	0		0	=4
b_5 =	0	0	0	0	0	1	=1

必须证明图 G 有 k 个顶点的顶点覆盖的充要条件是存在 $S'\subseteq S$，使得

$$\sum_{a\in S'} a = b$$

一般地假定图 G 的顶点覆盖 $V'=\{v_{i_1},v_{i_2},\cdots,v_{i_k}\}$，则

$$S'=\{a_{i_1},a_{i_2},\cdots,a_{i_k}\}\cup\{b_j \mid e_j\ \text{只有一个端点属于}\ V'\}$$

不难证明等式 $\sum_{a\in S'} a=b$ 成立。b 的 4 进制表示的首位 k 正好来自 k 个 $a_i\in S'$，而其余各位都是 2，这是由于每一条边 e_j 只有一端属于 V'，另一个 1 是由 b_j 提供的。一般地，

$$b=k4^m+2\sum_{j=0}^{m-1}4^j$$

反之，若存在一子集 $S'\subseteq S$，使得

$$\sum_{a\in S'} a=b$$

证明

$$S'=\{a_{i_1},a_{i_2},\cdots,a_{i_k}\}\cup\{b_{j_1},b_{j_2},\cdots,b_{j_l}\}$$

并且 $\{v_{i_1},v_{i_2},\cdots,v_{i_k}\}$ 是顶点覆盖。

每一条边 $e_j\in E$，对应于 e_j 这一列有元素 1，另一个 1 来自 b_j。由于是 4 进位的，所以不产生进位现象。除了最高位 4^m 位外，每一位对 a_i 对和的贡献至少为 1，最多为 2。对和的贡献至少为 1 正说明 V' 是顶点覆盖，b 的最高位 k 说明顶点覆盖的大小为 k。

26.5 复杂度类

(1) 算法复杂性理论是对问题的难度进行研究。比如 NP 类是可在多项式时间内判定是否是问题解的一类问题。P 类问题属于 NP，即 P⊆NP。NPC 是 NP 类中最难的一类问题，其中任何一个问题目前都尚未找到多项式算法，它们的难度相当，只要其中有一个问题找到多项式算法，则 NPC 类全部都有多项式解法。所以问题 P＝NP? 是计算机科学工作者十分感兴趣的待论问题。

若 P⊆NP，则 P、NP、NPC 三类的关系可用图 26.5.1 表示，即用椭圆表示 NP 类问题的集合，P 和 NPC 都是 NP 类的两个不相交子类，如若 P 和 NPC 有共同部分，则

$$P=NP$$

到目前为止还看不到这方面的一线曙光，如若已经肯定 P 不可能等于 NP，也可丢掉幻想，也就停止对它的努力。而目前已经肯定是 NPC 的问题不断增加，多达数以千计，上述几个是最基本的。

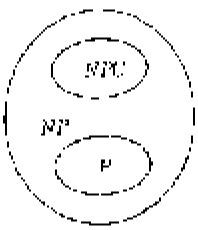


图 26.5.1

(2) Co-NP 问题

回忆下面熟悉的判定问题：

哈密顿回路问题：已知图 $G=(V,E)$，判定 G 存在哈密顿回路是 NP 问题。

哈密顿回路的余问题：

已知图 $G=(V,E)$，判定 G 不存在哈密顿回路的问题，即是哈密顿回路的余问题。

这个余问题是否是 NP 的还不清楚，事实上，我们很快会明白，它大概不是。要证明一个图是非哈密顿的，至今所知的唯一一般方法本质上是系统列举 G 的所有回路并验证没有回路包含 G 的所有点，这样的列举当然是个证明，但不幸是指数级的证明。

下面简单介绍 NP 类问题的余——Co-NP 问题：

从可满足性问题导出 NP 类，从可满足性问题的余导出 Co-NP 类，或称 NP 类的余。什么是 Co-NP？它可表示为 $\{0,1\}^*\setminus L$，其中 $L\in$ NP，$\{0,1\}^*$ 表示由 0、1 符号串构成的集。

NP 类问题指的是可在多项式时间内判定是的问题解的一类；Co-NP 类只是将问题中的"是"改为"非"。

例 判定图 $G=(V,E)$ 是连通的，这问题属于 NP，判定它不是连通的属于 Co-NP 类，但方法是一样的。

$$NP = Co\text{-}NP\ ?$$

也是一个未解决的问题。P=NP 的充要条件是 P=Co-NP，此时 NP=Co-NP。我们感兴趣的是 NP∩Co-NP 非空。例如"判定一整数 n 是合数"是属于 NP，而它的余"判定它不是合数"实际是同一问题。

各种不同复杂度类之间的关系可用图 26.5.2 来表示，其中 P 空间指的是要求有多项式空间但不需要多项式时间的一类问题。

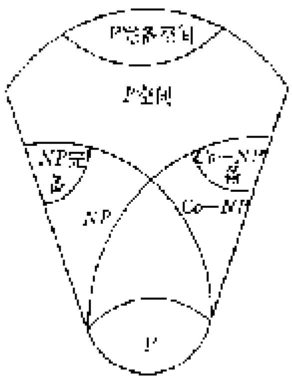


图 26.5.2

(3) NP 难题

虽然有时我们也能证明所有 NP 问题可在多项式时间内归结到某问题 A，但是我们不能证明 $A\in$ NP，所以 A 不能被称为 NP 完备的。然而毫无疑问，A 至少与任意 NP 问题有同样的难度，也即还不存在好算法的，正是为这些问题，我们引进 NP 难题的概念。

定义 若所有的 NP 问题都可以转换为 Π，则称 Π 为 NP 难题，简记为 NPH。

Π 是 NP 难题不要求 Π 属于 NP 类。NPC 问题一定是 NP 难题，但 NP 难题不一定是 NPC。比如流动推销员问题便不是 NP 完全的 NP 难题。

定理 对称型的流动推销员问题为 NP 难题。

所谓对称型的流动推销员问题指的是旅费矩阵 $C=(c_{ij})$ 是对称的。

证明 首先对称的流动推销员问题不属于 NP 类，因不能在多项式时间内对"猜想"进行检验，即对"是否是所有哈密顿回路中最优的？"作出肯定的或否定的判断。所以只要证明一个无向图 $G=(V,E)$ 是否存在哈密顿回路问题可在多项式时间内转换为对称的流动推销员问题，定理便得到了证明。

已知无向图 $G=(V,E)$，$|V|=n$，构造流动推销员问题如下。令

$$c_{ij}=\begin{cases}1, & (v_i,v_j)\in E\\ 2, & 其他\end{cases}$$

显然，图G的流动推销员问题有解的充要条件是图G有哈密顿回路，而且它的解为n。

注意：若G不含哈密顿回路，这时流动推销员问题的解至少为$n+1$。　　　　证毕。

一般说来，判定问题可能属于NP完备类，相应的最优问题则是NP难题，最明显的例子是判定图$G=(V,E)$是否存在哈密顿回路已证明属于NP完备类，然而求哈密顿回路中最短路径的流动推销员问题则是NP难题。又如判定图$G=(V,E)$是否存在k个顶点的团属于NP完全类，然而求图$G=(V,E)$的顶点数最多的团则属于NP难题。

习　题

1. 将下列逻辑表达式化为合取范式：

(1) $((x_1\vee x_2)\wedge(x_1\wedge x_2)\vee(x_1\wedge x_3))\vee x_4$

(2) $(\bar{x}_1\wedge\bar{x}_2\wedge x_3)\vee(x_1\wedge x_2\wedge\overline{(x_3\vee x_4)})\vee(x_1\wedge x_3)$

(3) $(x_1\wedge x_2)\vee(\overline{(x_1\vee x_2\vee x_3\vee x_4)}\wedge(x_1\vee x_2))$

(4) $\overline{(x_1\vee(x_2\wedge x_3)\vee x_4)}\wedge(x_1\vee\overline{(x_2\wedge x_3)})$

(5) $(x_1\vee\bar{x}_2\vee x_3\vee x_4)\wedge((x_2\wedge x_3)\vee x_1)$

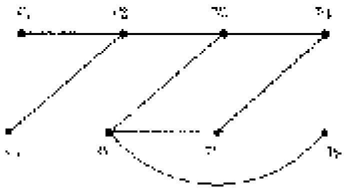
2. 上题中哪一个表达式是可满足的？

3. 求最少用多少种颜色对图题25.1的顶点着色，使得相邻顶点不同色。

4. 一图$G=(V,E)$称为是二分图，若顶点集合V可表为不相交的两个集合X和Y的和，即

$$V=X\cup Y,\ X\cap Y=\phi$$

而且图G所有的边将X的一个顶点和Y的一个顶点相连。试证判定一个图是否为二分图可在多项式时间内归约为判定该图为二染色问题，并证二染色问题属于P。



习题 25.1

5. S是整数集合，S是否存在一个子集C，使得C的元素之和等于已知数A的问题属于NP完备，并证它可在多项式时间内归约为0-1背包问题。

6. S是整数集合，S是否存在一子集U，使得U的元素之和等于$S-U$元素之和，这样的问题称之为分割，试证它可在多项式时间内归结为子集和问题，并可归结为0-1背包问题。

7.
$$\sum_{j=1}^{n} a_{ij}x_j\leqslant b_i,\ i=1,2,\cdots,m$$
$$\sum_{j=1}^{n} c_jx_j\geqslant B$$
$$x_j=0\ 或\ 1,\ j=1,2,\cdots,n$$

试证满足上面约束条件的0-1整数规划问题属于NPC。

8. 图$G=(V,E)$，U是V的子集，E中每一条边e至多和U中一个点关联，称U为G的独立集。试证求最大独立集问题属于NPC。试写一求最大独立集的算法，并讨论其复

杂性。

9. 已知无向图 $G=(V,E)$ 的邻接矩阵

$$A=(a_{ij})_{n\times n},\quad n=|V|,$$

$$a_{ij}=\begin{cases}1, & (i,j)\in E,\\ 0, & 其他,\end{cases}\qquad i,j=1,2,\cdots,n$$

试求可达矩阵 $P=(p_{ij})_{n\times n}$

$$p_{ij}=\begin{cases}1, & 从\ i\ 存在道路到\ j\\ 0, & 其他,\end{cases}\qquad i,j=1,2,\cdots,n$$

并证求 P 矩阵问题属于多项式类。

10. 已知无向图 $G=(V,E)$ 的距离矩阵

$$D=(d_{ij})_{n\times n},$$

其中

$$d_{ij}=\begin{cases}l(i,j), & 若(i,j)\in E,\\ \infty, & 若(i,j)\notin E,\end{cases}$$

试证求任意两点 u,v 间最短路径问题属于多项式类。其中 $l(i,j)$ 表示边 $(i,j)\in E$ 的长度。

11. 已知图 $G_1=(V_1,E_1)$，$G_2=(V_2,E_2)$，试证判断 G_1 和 G_2 同构问题属于 NP 类。

12. 试证无回路有向图的哈密顿道路问题是属于多项式类，并给出有效的算法。

13. 证 P⊆Co-NP。

14. 证 若 NP≠Co-NP，则 P≠NP。

15. $A=(a_{ij})_{m\times n}$，$b=(b_1,b_2,\cdots,b_m)^T$，判断是否存在 0-1 向量 $x=(x_1,x_2,\cdots,x_n)$ 满足 $x_i\in\{0,1\}$，$i=1,2,\cdots,n$，使

$$AX\leq b$$

这问题属于 NP 完备。

16. 试证下列问题是 NP 完全：

(1) 装箱问题；

(2) 哈密顿回路问题；

(3) 可满足性问题。

17. 流动推销员问题是否 NP 问题，为什么？

第27章 近似算法

不解决 P≠NP 猜想，一定得不出关于 NP 完备问题困难程度的定论来。尽管这问题有重大理论意义，并且许多计算机科学家对此有兴趣，但这个猜想的证明还很渺茫。这个问题的解决也许尚待时日，也许需要新的数学方法的出现才能作出回答。不过确实有许多著名的组合数学问题，并证明了它们属于 NP 完备或是 NP 难题，当然至今尚无有好的算法，因而求其有效的近似解法是必由之路。近似算法就是这样一些算法，它们给出的不是最优解，而是保证偏离真正最优解有固定误差的解。关于 NP 难题的近似解法还很年轻，尚不可能建立问题的近似解法。它有待于进一步发展。本文将就其中若干个问题讨论这种办法及其局限性。

27.1 任务安排的近似算法

设有 l 台完全相同的机床 $m_1, m_2, \cdots, m_l$ 加工 n 个彼此无关的任务 $j_1, j_2, \cdots, j_n$，所需时间分别为 $t_1, t_2, \cdots, t_n$，要求最后全部结束的时间最短。

l 台全同的机器的任务安排，当 $l=2$ 时实际上就是划分问题。设完成任务 j_i 所需的时间为 t_i，$1 \leqslant i \leqslant n$。任务安排问题相当于对集合

$$A = \{t_1, t_2, \cdots, t_n\}$$

的划分。$l>2$ 是它的自然推广。

1. 任务安排的近似算法 A

下面介绍一种实用的近似算法，对时间序列进行排序，不失一般性假定

$$t_1 \geqslant t_2 \geqslant \cdots \geqslant t_n$$

这个顺序也就是加工的先后顺序。当某台机器空闲时，立即将已剩下需要加工的时间最长的任务。

例 1 设 $l=3, n=6, t_1=10, t_2=9, t_3=8, t_4=7, t_5=6, t_6=5$。加工顺序如图 27.1.1，可见本例近似算法的结果也是最优的。

例 2 设 $l=3, n=7, t_1=6, t_2=6, t_3=t_4=t_5=4, t_6=t_7=3$。

这个问题的最佳方案应为图 27.1.2。

然而利用近似算法得图 27.1.3。

可见机床 m_1 在 $t=10$ 时结束工作，m_2 在 $t=9$ 时结束工作，然而 m_3 在 $t=11$ 时才结束，和最优方案相比完成任务的最后时间推迟了一个时间单位。如若说近似算法的绝对偏差 $\Delta=1$，则近似算法的相对偏差 δ 为 $\frac{1}{9}$。最佳的任务安排使完成 n 项任务所需的时间设为 T_l^*；近似算法所得任务安排，完成的时间设为 T_l，则绝对偏差 $\Delta=T_l-T_l^*$，相对偏差 $\delta=(T_l-T_l^*)/T_l^*$。

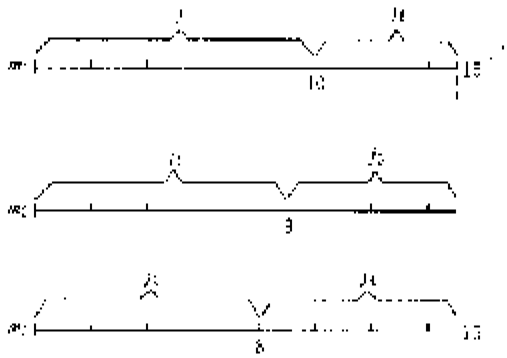


图 27.1.1

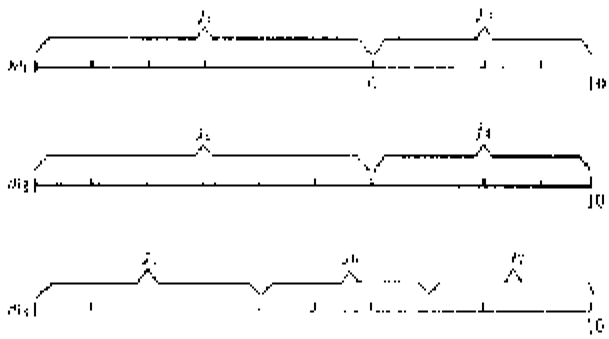


图 27.1.2

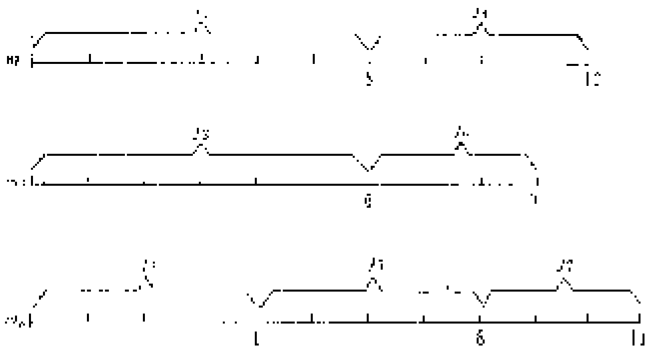


图 27.1.3

可以证明近似算法 A 的相对偏差 δ 满足

$$\delta \leqslant \frac{1}{3} - \frac{1}{3m} \tag{27.1.1}$$

当 $m=1$ 时公式(27.1.1)无疑是对的。

$l>1$时，设有使(27.1.1)不成立的数目最少的一组任务$(j_1,j_2,\cdots,j_n)$，完成任务j_i所需的时间为t_i。不失一般性，令

$$t_1\geqslant t_2\geqslant\cdots\geqslant t_n$$

首先可证当n是使(27.1.1)不成立的最小数目时，在近似算法A_1安排下最后完成的任务一定就是j_n。如若不然，设为j_k，$k<n$，即任务j_n在任务j_k之前结束。则依算法A_1完成任务

$$j_1,j_2,\cdots,j_k\quad(J_k)$$

的时间等于完成任务

$$j_1,j_2,\cdots,j_k,\cdots,j_n\quad(J_n)$$

的时间，设为T_n，即从j_{k+1}到j_n的$n-k$个任务都在完成任务j_k的过程中完成。完成(J_k)的最优方案所需的时间设为T_k^*，完成(J_n)的最优方案所需时间为T_n^*。显然有：

$$T_k^*\leqslant T_n^*$$

故$T_n-T_k^*\geqslant T_n-T_n^*\geqslant 0$。所以

$$\frac{T_n-T_k^*}{T_k^*}\geqslant\frac{T_n-T_n^*}{T_n^*}>\frac{1}{3}-\frac{1}{3l}$$

这与n是使(27.1.1)不成立的数目最少的一组任务之假定相矛盾，这就证明了$k=n$。

其次将证明若存在数目最小的任务序列$j_1,j_2,\cdots,j_n$使(27.1.1)不成立，则

$$n\leqslant 2l$$

因j_n必然是最后完成的一个任务。j_n开始的时间为T_n-t_n，这时所有机床无一空闲，故

$$T_n-t_n\leqslant\frac{1}{l}\sum_{i=1}^{n-1}t_i$$

$$T_n\leqslant\frac{1}{l}\sum_{i=1}^{n-1}t_i+t_n=\frac{1}{l}\sum_{i=1}^{n}t_i+\left(1-\frac{1}{l}\right)t_n$$

另一方面

$$T_n^*\geqslant\frac{1}{l}\sum_{i=1}^{n}t_i$$

所以

$$T_n-T_n^*\leqslant\frac{l-1}{l}t_n$$

$$\frac{T_n-T_n^*}{T_n^*}\leqslant\frac{l-1}{l}\frac{t_n}{T_n^*}$$

根据假设

$$\frac{T_n-T_n^*}{T_n^*}>\frac{1}{3}-\frac{1}{3l}$$

所以

$$\frac{l-1}{l}\frac{t_n}{T_n^*}>\frac{1}{3}-\frac{1}{3l}=\frac{l-1}{3l}$$

$$\frac{t_n}{T_n^*}>\frac{1}{3},\quad T_n^*<3t_n$$

由于 $t_1 \geq t_2 \geq \cdots \geq t_n$，所以 $n \leq 2l$，
但是对于 $n \leq 2l$ 的任务 $J_1, J_2, \cdots, J_n$，近似算法 A_1 给出的就是最优解，与假设矛盾，故(27.1.1)式无例外都成立。　　证毕。

$\frac{1}{3}-\frac{1}{3l}$ 是近似算法 A_1 在最坏情况下相对偏差的界。算法 A_1 的主要工作，一是对 $t_1, t_2, \cdots, t_n$ 进行排序，二是将排序结束的任务进行安排，所以其复杂性为 $O(n\log n)$。

2. 任务安排的近似算法 A_2

假定已经排序得 $t_1 \geq t_2 \geq \cdots \geq t_n$。确定整数 k，先对前 k 个任务求最优的安排，然后对后 $n-k$ 个任务应用算法 A_1。

例　$l=2, n=10, k=4, t_1=15, t_2=13, t_3=12, t_4=10, t_5=8, t_6=7, t_7=6, t_8=4, t_9=3, t_{10}=2$

此例的最佳安排应为：(如图 27.1.4)

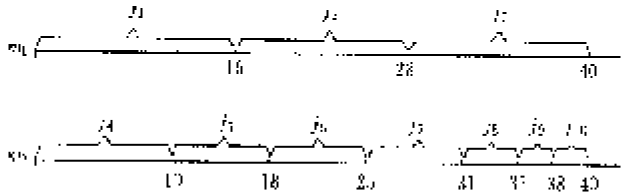


图　27.1.4

按算法 A_2，第 1 步找前 4 个任务的最佳安排，见图 27.1.5。

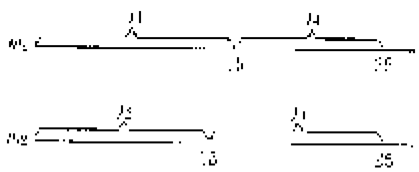


图　27.1.5

第 2 步按近似算法 A_1 进行如图 27.1.6。

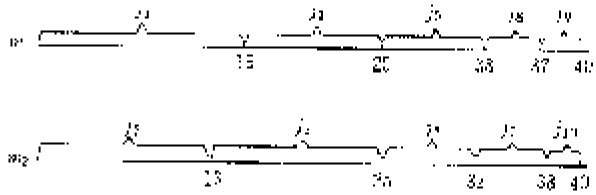


图　27.1.6

这例子说明用近似算法 A_2 得到的正好也是最优方案。

定理　算法 A_2 的相对偏差 δ 满足：

$$\delta \leqslant \frac{1-1/l}{1+\lfloor k/m \rfloor}$$

证明 令 t 为前 k 个最长的任务的最佳方案所需的时间，T_0 为依照算法 A_k 完成全部任务所需时间，显然对于 $n>k$ 有：

$$T_0 \geqslant t$$

令 n 个任务中最后在 T_k 时刻完成的是 t_j，$j>k$。则在时间间隔 0 到 T_k-t_j 内没有空闲的机床，不妨假定 $n>k$，$m>l$。由于 $t_{k+1}\geqslant t_j$，故机床在时间间隔 0 到 T_k-t_{k+1} 内都不空闲，所以

$$\sum_{i=1}^{n} t_i \geqslant l(T_k-t_{k+1})+t_{k+1}$$

$$T_0 \geqslant \frac{1}{l}\sum_{i=1}^{n} t_i \geqslant T_k-\frac{l-1}{l}t_{k+1}$$

$$T_k-T_0 \leqslant \frac{l-1}{l}t_{k+1}$$

但 $t_i \geqslant t_{k+1}$，$1\leqslant i\leqslant k+1$，根据鸽巢原理至少有一台机床完成这 $k+1$ 个任务中的数目不少于 $1+\lfloor k/l \rfloor$，故

$$T_0 \geqslant (1+\lfloor k/l \rfloor)t_{k+1}$$

合并上面两个不等式可得：

$$\frac{T_k-T_0}{T_0} \leqslant \frac{1-1/l}{1+\lfloor k/l \rfloor}$$

证毕。

27.2 装箱问题的近似算法

设有体积分别为 $v_1,v_2,\cdots,v_n$ 的 n 种物品 $u_1,u_2,\cdots,u_n$ 装到容量为 L 的箱子里。不同的装箱方案所需要的箱子数目可能不同，所谓装箱问题就是要使装完这 n 件物品的箱子数目达到最小。

装箱问题之实质也是一个划分问题，下面介绍若干近似算法，其法见 D. S. Johnson, "*Near-Optimal Bin Packing Algorithms.*"(1973)。

1. 算法 BP_1

设有 n 种物品 $u_1,u_2,\cdots,u_n$ 依次装箱，$u_i,u_{i+1},\cdots,u_j$ 已装进 B_k 箱，$L-(v_i+v_{i+1}+\cdots+v_j)<v_{j+1}$，即 u_{j+1} 已不能装进 B_k 箱，则 B_k 箱虽未装满但也封住并退出，u_{j+1} 装入另一个箱子 B_{k+1}。在以后的讨论中不妨令 $L=1$。

已装入箱子 B_i 的物品之体积用 $C(B_i)$ 表示之，则显然有：

$$C(B_i)+C(B_{i+1})>1$$

$$C(B_1)+C(B_2)+\cdots+C(B_m)>m/2,\ m \text{ 是任一偶数；}$$

$$C(B_1)+C(B_2)+\cdots+C(B_m)>(m-1)/2,\ m \text{ 是任一奇数}$$

对物品 $U=\{u_1,u_2,\cdots,u_n\}$，用 BP_1 法装箱所需的箱子数用 $BP_1(U)$ 表之，设最佳方案

所需的箱子数为 B^*，所以

$$B^* \geq \frac{m-1}{2}, \quad m < 2B^* + 1$$

$$BP_1(u) < 2B^* + 1 \qquad (27.2.1)$$

上式说明用 BP_1 法所用的箱子数不超过最佳方案的两倍。BP_1 法是最简单的方法。

2. BP_2 法

BP_1 法是最直观、最简易想到的一种，BP_2 由 BP_1 法略加修改而得。它的基本思想是 n 种物品 $u_1,u_2,\cdots,u_n$ 依次装箱，设箱的次序为 $B_1,B_2,\cdots,B_n$，对于某一个物品 u_i，它总是被装到第一个能装下 u_i 的箱子里，也就是说物品 u_i 被装到已装进的物品的体积小到足以容下 u_i 的下标为最小的一个箱子。对未装满的箱子既不封住，也不退出。对于物品 u_i，对所有的未装满的箱子从 $j=1$ 开始顺序检查，只要 $1-C(B_j)\geq v_i$，则将 u_i 装进 B_j 箱，即 B_j 满足：

$$1 - C(B_j) \geq v_i,\quad 1 - C(B_k) < v_i,\quad k < j$$

和 BP_1 算法一样，对于 n 种物品 $U=\{u_1,u_2,\cdots,u_n\}$，

$$BP_2(U) < 2\sum_{i=1}^{n} v_i$$

$$BP_2(U) < 2B^* \qquad (27.2.2)$$

公式(27.2.2)给出了 BP_2 算法最坏情况下的界。然而 BP_2 算法有可能达到 B^* 这个最佳的结果。

关于 $BP_2(u)$，下列不等式成立：

$$\frac{17}{10}B^* - 2 \leq BP_2(U) \leq \frac{17}{10}B^* + 2$$

其证明从略。关于 BP_2 算法可形象地用图 27.2.1 表示。

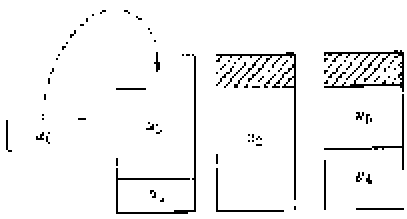


图 27.2.1

3. BP_3 算法

对近似算法 BP_2 作如下修改，物品 u_i 装进箱 B_j，要求：

$$1-C(B_j)-v_i = \min\{1-C(B_k)-v_i \mid 1-C(B_k) \geq v_i\}$$

选取 B_j 的目的是使得 u_i 装进以后留的空隙最小。关于算法 BP_3 有如下结果：

$$\frac{17}{10}B^* - 2 \leq BP_3(U) \leq \frac{17}{10}B^* + 2$$

4. BP_4 算法

第一步按体积大小将 n 个物品 $u_1,u_2,\cdots,u_n$ 从大到小排序得：

$$v_1 \geq v_2 \geq \cdots \geq v_n$$

然后再利用 BP_2 算法。关于 $BP_4(U)$ 有：

$$\frac{11}{9}B^* \leq BP_4(U) \leq \frac{11}{9}B^* + 4$$

5. BP_5 算法

与 BPf 算法类似，首先按体积从大到小将 n 个物品 $u_1,u_2,\cdots,u_n$ 排序得：

$$v_1 \geqslant v_2 \geqslant \cdots \geqslant v_n$$

然后再利用 BPf 算法。关于 $BP'(u)$ 有：

$$\frac{11}{9}B^* \leqslant BP'(U) \leqslant \frac{11}{9}B^* + 1$$

27.3 流动推销员问题的近似算法

前面我们讨论过本问题属于 NP 难题，不存在或至少到目前为止尚未找到多项式解法，这里也只是讨论它的几种近似解法。

1. 最近邻法

最近邻法的主要思想就是在尚未到达过的城市中选取与当前所在城市最邻近的城市作为下一个要访问的城市。具体地，设 $1,2,\cdots,n$ 表示 n 个城市，从城市 v_1 出发，不失一般性，假设前面的线路为：

$$v_1 \to v_2 \to \cdots \to v_k,\ k < n$$

下一个城市 v_{k+1} 应选取余下的 $n-k$ 个城市中离 v_k 最近的一个城市，从而最近邻法共需比较次数为：

$$(n-2)+(n-3)+\cdots+1=\frac{1}{2}(n-1)(n-2)$$

在最好情况下，最近邻法可以得到最佳路径。

下面讨论使用最近邻法时的误差：

用 T_n^* 表示问题规模为 n 时最佳回路，N_n 为用最近邻法所得的近似解的路径。

n 个城市间的距离矩阵记为：

$$D=(d_{ij})_{n\times n},\quad n \geqslant 2$$

并满足

(1) $d_{ij}=d_{ji}$

(2) $d_{ij}\leqslant d_{ik}+d_{kj}$

设用最近邻法得出的回路的 n 条边的长度元为：

$$l_1 \geqslant l_2 \geqslant \cdots \geqslant l_n$$

从 $d_{ij}=d_{ji}$，$d_{ij}\leqslant d_{ik}+d_{kj}$，$n\geqslant 2$ 立即可得

$$|T_n^*| \geqslant 2l_1 \tag{27.3.1}$$

定理
$$|T_n^*| \geqslant 2\sum_{i=k+1}^{2k} l_i,\quad 1 \leqslant k \leqslant \left\lfloor \frac{n}{2} \right\rfloor \tag{27.3.2}$$

证明 设根据最邻近法加进 l_i 边时，加进的城市为 a_i，记 $A_k=\{a_1,a_2,\cdots,a_{2k}\}$，访问 A_k 中诸城市的先后顺序和最佳路径一致的路径设为 T_k，由 $d_{ij}\leqslant d_{ik}+d_{kj}$ 可得：

$$|T_n^*| \geqslant T_k$$

下面转而证明
$$2\sum_{i=k+1}^{2k} l_i \leqslant |T_k|,\quad 1 \leqslant k \leqslant \left\lfloor \frac{n}{2} \right\rfloor$$

令 v_i, v_j 为 A_k 中的城市，而 (v_i, v_j) 为属于 T_k 的边。依据最近邻法，如果 v_i 在 v_j 前面加入到 N_n，则 $d_{v_iv_j} \geqslant l_i$；另一方面，若 v_j 在 v_i 前面加入到 N_n，则 $d_{v_iv_j} \geqslant l_j$。但是由于 $d_{v_iv_j} = d_{v_jv_i}$，v_i 和 v_j 中总有一个比另一个早加入到 N_n，总

$$d_{v_iv_j} \geqslant \min(l_i, l_j)$$

所以：

$$|T_k| = \sum_{(v_i,v_j) \in T_k} d_{v_iv_j} \geqslant \sum_{(v_i,v_j) \in T_k} \min(l_i, l_j)$$

但因为 $l_{k+1}, l_{k+2}, \cdots, l_{2k}$ 是 $l_1, l_2, \cdots, l_{2k}$ 中最小的 k 个，而且对于 $1 \leqslant i \leqslant 2k$ 的 l_i 在 T_k 中出现至多两次，故

$$|T_k| \geqslant \sum_{(v_i,v_j) \in T_k} \min(l_i, l_j)$$

$$\geqslant 2(l_{k+1} + l_{k+2} + \cdots + l_{2k}) = 2\sum_{i=k+1}^{2k} l_i$$

所以 T_k 的下界是 $2\sum_{i=k+1}^{2k} l_i$。

从而

$$|T_n^*| \geqslant 2\sum_{i=k+1}^{2k} l_i \qquad 1 \leqslant k \leqslant \left\lfloor \frac{n}{2} \right\rfloor$$

定理 $$2N_n \leqslant (\lceil \log_2 n \rceil + 1)T_n^* \tag{27.3.3}$$

证明 由上面定理

令 $A_k = (v_1, v_2, \cdots, v_n)$，$T_n^*$ 取代 T_k 得：

$$T_n^* \geqslant 2(l_{\lceil \frac{n}{2} \rceil + 1} + \cdots + l_n)$$

即：

$$|T_n^*| \geqslant 2\sum_{i=\lceil \frac{n}{2} \rceil + 1}^{n} l_i$$

对于式(27.3.2)，令 $k = 1, 2, 2^2, \cdots, 2^{\lceil \log_2 n \rceil - 2}$，得：

$$|T_n^*| \geqslant 2(l_2)$$

$$|T_n^*| \geqslant 2(l_3 + l_4)$$

$$|T_n^*| \geqslant 2(l_5 + l_6 + l_7 + l_8)$$

$$\cdots\cdots$$

$$+)\ |T_n^*| \geqslant 2(l_{2^{\lceil \log_2 n \rceil - 2} + 1} + \cdots + l_{2^{\lceil \log_2 n \rceil - 1}})$$

$$(\lceil \log_2 n \rceil - 1)T_n^* \geqslant 2(l_2 + l_3 + \cdots + l_{2^{\lceil \log_2 n \rceil - 1}})$$

已由(27.3.1)、(27.3.2)有

$$T_n^* \geqslant 2l_1,$$

$$T_n^* \geqslant 2(l_{\lceil \frac{n}{2} \rceil + 1} + l_{\lceil \frac{n}{2} \rceil + 2} + \cdots + l_n)$$

但

$$2^{\lceil \log_2 n \rceil - 1} \geqslant \left\lceil \frac{n}{2} \right\rceil$$

故 $$(\lceil \log_2 n \rceil - 1)T_n \geqslant 2(l_1 + l_2 + \cdots + l_n)$$
即得证 $$2N_n \leqslant (\lceil \log_2 n \rceil - 1)T_n.$$

下面先来看一个例子：

令

$$
D = D^{(1)} = \begin{array}{c} 1 \\ 2 \\ 3 \\ \\ \end{array}
\begin{array}{c}
\begin{pmatrix} \infty & 1 & 1 \\ 1 & \infty & 1 \\ 1 & 1 & \infty \end{pmatrix} \\
\begin{array}{ccc} 1 & 2 & 3 \end{array}
\end{array}
$$

其图象如图 27.8.1。

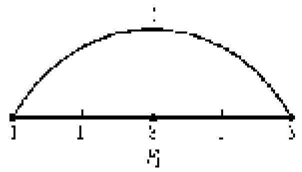


图 27.8.1

从矩阵

$$D^{(k)} = (d_{ij})_{(2^{k+1}-1)\times(2^{k+1}-1)}$$

其阶为 $(2^{k+2}-1)\times(2^{k+2}-1)$ 的矩阵 $D^{(k+1)}$ 可如下计算：

$$
D^{(k+1)} =
\begin{array}{c|ccccccc|cccccc}
1 & & & & & & & & \infty & \infty & \cdots & \infty & \cdots & \infty \\
2 & & & & & & & & \infty & \infty & \cdots & \infty & \cdots & \infty \\
\vdots & & & & & & & & \vdots & \vdots & & \vdots & & \vdots \\
2^k & & & & D^{(k)} & & & & \infty & L & \cdots & \infty & \cdots & \infty \\
\vdots & & & & & & & & \vdots & \vdots & & \vdots & & \vdots \\
2^{k+1}-1 & & & & & & & & 1 & \infty & \cdots & \infty & \cdots & \infty \\
2^{k+1} & \infty & \infty & \cdots & \infty & \cdots & 1 & & \infty & 1 & \cdots & L & \cdots & \infty \\
2^{k+1}+1 & \infty & \infty & \cdots & L & \cdots & \infty & & 1 & & & & & \\
\vdots & \vdots & \vdots & & \vdots & & \vdots & & \vdots & & & & & \\
2^{k+1}+2^k & \infty & \infty & \cdots & \infty & \cdots & \infty & & L & & & D^{(k)} & & \\
\vdots & \vdots & \vdots & & \vdots & & \vdots & & \vdots & & & & & \\
2^{k+2}-1 & \infty & \infty & \cdots & \infty & \cdots & \infty & & \infty & & & & & \\
\hline
 & 1 & 2 & \cdots & 2^k & \cdots & 2^{k+1}-1 & & 2^{k+1} & 2^{k+1}+1 & \cdots & 2^{k+1}+2^k & \cdots & 2^{k+2}-1
\end{array}
$$

其中

$$L = \frac{1}{6}(2^{k+2} + (-1)^{k+1} - 3)$$

设与 D_1 对应的图象为 F_1，则上面求 $D^{(k+1)}$ 的过程可形象地用图 27.8.2 表示。

令路径 p_1 对应于图 F_1 中的 1→3→2，p_{k+1} 对应于图 F_{k+1} 中的路径：

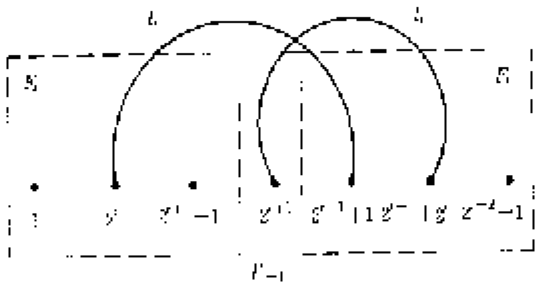


图 27.4.2

$$p_{i+1} = (p_i - 2^{i-1}) + 2^i$$

其中$(p_i + 2^{i-1})$的意义就是由p_i各点将下标加上2^{i-1}作为新的序号构成节点的路径。$p_i \to p_i + 2^{i-1}$表示首连接p_i和$p_i + 2^{i-1}$构成的路径，这从P_{i+1}图可看出。另外要注意的是：p_i中第一个点为1，最后一个点为2，并且P_i中所有点都在路径p_i上。P_{i+1}中从2^i到2^{i+1}点的距离与从2^i+2到2^i点的路径距离相等，为l_i。

设路径p_i的长度为S_i，则由上面的讨论可得如下的递推关系式：

$$S_{i+1} = 2S_i + 2l_i$$

代入l_i得：

$$S_{i+1} = 2S_i + \frac{1}{3}(2^{i+1} + (-1)^{i} + 3)$$

由初始条件$S_1 = 2$可得：

$$S_i = \frac{1}{9}[(3i + 4)2^{i} - (-1)^{i} - 5]$$

现在令$D^{(i)} = D^{(i)}$，对于$i > 1$时，$D^{(i)}$是由把$D^{(i-1)}$中元素$a_{1,2^{i-1}}$从∞改为$l_{i-1}+2$，把$a_{2^{i-1},1}$从∞改为1，所得到的对称矩阵。例如由

$$D^{(1)} = \begin{pmatrix} \infty & 1 & 1 \\ 1 & \infty & 1 \\ 1 & 1 & \infty \end{pmatrix}$$

可得 $l = \frac{1}{3}(2 + 1 - 3) = 12/6 = 2$，

$$D^{(2)} = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{pmatrix} \infty & 1 & 1 & \infty & \infty & \infty & \infty \\ 1 & \infty & 1 & \infty & 2 & \infty & \infty \\ 1 & 1 & \infty & 1 & \infty & \infty & \infty \\ 2 & \infty & 1 & \infty & 1 & 2 & \infty \\ \infty & 2 & \infty & 1 & \infty & 1 & 1 \\ \infty & \infty & \infty & 2 & 1 & \infty & 1 \\ 1 & \infty & \infty & \infty & 1 & 1 & \infty \end{pmatrix}$$

则

$$
D^{(3)}=\begin{array}{c|ccccccc}
1 & \infty & 1 & 1 & 2 & \infty & \infty & 1\\
2 & 1 & \infty & 1 & \infty & 2 & \infty & \infty\\
3 & 1 & 1 & \infty & 1 & \infty & \infty & \infty\\
4 & 2 & \infty & 1 & \infty & 1 & 2 & \infty\\
5 & \infty & 2 & \infty & 1 & \infty & 1 & 1\\
6 & \infty & \infty & \infty & 2 & 1 & \infty & 1\\
7 & 1 & \infty & \infty & \infty & 1 & 1 & \infty\\
\hline
 & 1 & 2 & 3 & 4 & 5 & 6 & 7
\end{array}
$$

可用图 27.3.3 表示。

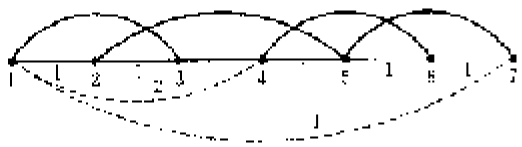


图 27.3.3

图中实线部分为 P_3，虚线部分为新加的边。

一般地，由 $D^{(k)}$ 推为 $D^{(k+1)}$ 相当于在图 P_{k+1} 中最左端的点和中间的点 $2^{k}-1$ 连以长为 $L_{k+1}-1$ 的边，最左端的点和最右端的点 $2^{k+1}-1$ 连以长度为 1 的边。

令 $S^{(k)}=(S_{ij})_{(2^{k+1}-1)\times(2^{k+1}-1)}$，其中 S_{ij} 为从 $D^{(k)}$ 求得的 i 到 j 的最短距离。显然 $S^{(k)}$ 与 $D^{(k)}$ 一样为对称矩阵，由上面的 $D^{(3)}$ 可得 $S^{(3)}$ 如下：

$$
S^{(3)}=\begin{array}{c|ccccccc}
1 & \infty & 1 & 1 & 2 & 2 & 2 & 1\\
2 & 1 & \infty & 1 & 2 & 2 & 3 & 3\\
3 & 1 & 1 & \infty & 1 & 2 & 3 & 3\\
4 & 2 & 2 & 1 & \infty & 1 & 2 & 2\\
5 & 2 & 2 & 2 & 1 & \infty & 1 & 1\\
6 & 2 & 3 & 3 & 2 & 1 & \infty & 1\\
7 & 1 & 3 & 3 & 2 & 1 & 1 & \infty\\
\hline
 & 1 & 2 & 3 & 4 & 5 & 6 & 7
\end{array}
$$

矩阵 $D^{(3)}$ 与 $S^{(3)}$ 的对应位置上有限数(非∞)相等。这性质具有一般性，对于 $k\geq 2$ 都成立。

矩阵 $S^{(k)}$ 的另一个重要性质是若从点 1 出发运用最近邻法将产生路径 P_k，最后返回点 1，可以求得，对于 $D^{(k)}$ 的图象 P_k 在中间点 2^k 和点 1 之间的最短路径为 L_k-1，类似可得点 2^k 和最右端的点 $2^{k+1}-1$ 间的最短距离也是 L_k-1。这个特性也说明，若从最佳路径的两个方向，最后结果一样。

由上面提供的例子可知，存在一个 $n(=2^{k+1}-1)$ 个点的最短线路：

$$1\to 2\to 3\to\cdots\to n\to 1$$

此路径的总长度为 n，即 $TC=n$，然而利用最近邻法可能得到：

$$N_n=S_n=L_k+1$$

$$= \frac{1}{9}[(3m-1)2^m - (-1)^m - 9] + \frac{1}{6}(2^{m+1} + (-1)^m - 3) - 1$$

$$= \frac{1}{3}m2^m - \frac{4}{9}2^m - \frac{3}{2} + \frac{1}{18}(-1)^m$$

$$= \frac{1}{3}n\log_2(n+1) - \frac{4}{9}n + \frac{1}{3}\log_2(n+1) - 1 - \frac{1}{18}[(-1)^{\log_2(n+1)} - 1]$$

所以

$$\frac{N_n}{T_n} = \frac{1}{3}\log_2(n+1) - \frac{4}{9} + \frac{\frac{1}{3}\log_2(n+1) - 1 + \frac{1}{18}[(-1)^{\log_2(n+1)} - 1]}{n}$$

$$> \frac{1}{3}\log_2(n+1) \qquad n \geqslant 2$$

即

$$\frac{N_n}{T_n} > \frac{1}{3}\log_2(n+1) \tag{*}$$

该例说明存在实实在在的例子，当 n 比较大时，定理给出了 N_n 很不坏的估计。也就是理论上最近邻法得出的路径可能是最佳结果的好几倍，总之以它作为近似解法，最好的情况得到的可能就是最优解，最坏情况也许是不妙的。

2. 最近插入法

最近插入法的基本思想是，在全 n 个点中的某 k 个点所构成的一个最佳回路 T_k 上，加进一个与 T_k 上的点最接近的点从而得到一个新的具有 $k+1$ 个顶点的最佳回路，这个过程可继续下去直到得出 n 个点的回路。那么哪一个点加进来？新加进的点放在回路的什么地方？比最近邻法复杂。但我们知道它在 $O(n^2)$ 时间内完成。

设最近插入法所得的回路的长度为 I_n，则有如下定理。

定理 $\frac{I_n}{T_n} < 2$。

证明 不失一般性，设 $1,2,\cdots,n$ 是最近插入法先后加入的点的顺序，并设点 i 插入在点 j 和 k 之间。令 $d_1 = 0$，则 i 加入后所引起的回路长度的改变量 δ_i 为

$$\delta_i = d_{ij} + d_{ik} - d_{jk}$$

可以证明点 $2,3,\cdots,n$ 可以分别与最优的哈密顿回路中除最长的边外的 $n-1$ 条边建立起一一对应关系，而且与点 i 对应的边的长度 l_i 不小于 $\frac{1}{2}\delta_i$，即

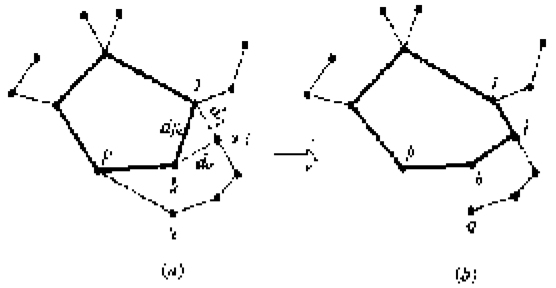
$$l_i \geqslant \frac{1}{2}\delta_i, \quad \delta_i \leqslant 2l_i$$

所以

$$\begin{aligned} I_n &= \delta_2 + \delta_3 + \cdots + \delta_n \\ &\leqslant 2(l_2 + l_3 + \cdots + l_n) = 2(T_n - l) \end{aligned}$$

l 为最短的哈密顿回路中的最长的边的长度。下面我们来证明这一一对应关系的存在性。

图 27.3.4(a) 中粗线为 T_k，细线为过 k 个点最佳回路中的边（但 l 除外），这些细线将尚未插入的点与 T_k 相连。



(a)　　　　　　(b)

图　27.3.4

设 i 为新插入点，插入位置在 j 和 k 之间，则

$$\delta_i = d_{ij} + d_{ik} - d_{jk}$$

p 是 T_i 上一点，q 是不在 T_i 上的但 p 邻近的点，也有可能 q 点就是 i 点，p、q 点在联 T_i 和 i 的道路上。(如(a)所示的那样)

$$d_{ji} \leqslant d_{pq}$$

$$d_{ki} \leqslant d_{jk} + d_{ji} \quad (\text{由假设}),$$

所以

$$d_{ki} \leqslant d_{jk} + d_{pq}$$

$$d_{ik} - d_{jk} \leqslant d_{jk} + 2d_{pq}$$

$$\delta_i = d_{ij} + d_{ik} - d_{jk} \leqslant 2d_{pq}$$

当 i 点插入在 j、k 之间时，则从图 27.3.4(a)中去掉 (p,q) 边，得图 27.3.4(b)。因此 i 点和 (p,q) 建立起一一对应关系而且满足

$$\delta_i \leqslant 2d_{pq}.$$　　　　证毕。

该定理说明使用最近插入法求得的旅行商回路在最坏情况下其长度不会超过最佳方案的两倍。

3. 最短树法

这里结合一个具体例子简单介绍求旅行商回路的近似算法——最短树法的具体步骤。

例　各点的具体位置如图 27.3.5 所示。

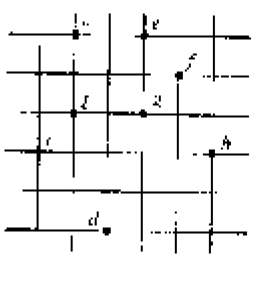


图　27.3.5

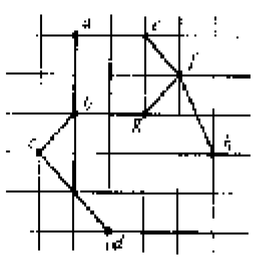


图　27.3.6

第1步：构造一最短树 T^*（图 27.3.6）。

第2步：任取一点 $r \in V$，以 r 作为树 T^* 的根结点，按 DFS 遍历法得出顶点的序列 P。选 a 作为树根，按 DFS 遍历法得顶点序列

$$a, b, c, d, e, f, g, h$$

第3步：按照顺序 P 构造哈密顿回路 H。

图 27.3.7 给出回路 H：

$$a \to b \to c \to d \to e \to f \to g \to h \to a.$$

回路 H 的总长度可计算如下：

$$7 + 5\sqrt{2} + \sqrt{5} + \sqrt{26} \approx 21.406$$

而我们不难求出实际上最短哈密顿回路 H^* 为

$$a \to b \to c \to d \to h \to g \to f \to e \to a$$

其总长度为：

$$4 + 5\sqrt{2} + \sqrt{5} + \sqrt{13} \approx 16.913$$

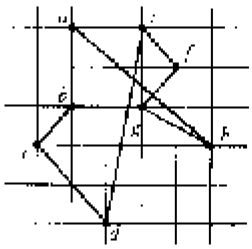


图 27.3.7

下面讨论运用最短树法求得的哈密顿回路有什么样的性质。

首先有

$$|H| \leqslant 2|T^*|$$

其中 $|H|$ 和 $|T^*|$ 分别表示哈密顿回路 H 和最短树 T^* 的长度。这由三角不等式：

$$d_{ij} \leqslant d_{ik} + d_{kj}$$

很容易得出。

另：
$$|T^*| \leqslant |H^*|$$

从而有：
$$|H| \leqslant 2|T^*| \leqslant 2|H^*|$$

也就是说使用最短树法求得的哈密顿回路在最坏情况下不超过最短哈密顿回路之长度的两倍，这与最近插入法很相似。

27.4 顶点覆盖问题的近似算法

由前知道求规模最小的顶点覆盖问题是 NP 难题，但求其近似解却优化问题不是很困难，其算法步骤可描述如下：

对无向图 $G=(V,E)$

(1) $C \leftarrow \phi, F \leftarrow E$。

(2) 若 $F=\phi$ 则转(3)。

否则，【任取 $(u,v) \in F$，$C \leftarrow C \cup \{u,v\}$，从 F 中删去所有与 u、v 关联的边，转(2)】

(3) 输出 C，停止。

关于由此得到的顶点覆盖集 C 有：

定理 $|C| \leqslant 2|C^*|$

其中 C^* 表示最小的顶点覆盖集。

证明 显然C是图$G=(V,E)$的一个顶点覆盖。

设算法中任意选取的边的集合为S，S中不存在两条边有共同的端点，所以每次吸收到C的端点有2个，即

$$|C| = 2|S| \qquad (27.4.1)$$

而最优的顶点覆盖集C^*至少含有边集合S中每一条边的一个端点，即

$$|S| \leqslant |C^*| \qquad (27.4.2)$$

由(27.4.1)、(27.4.2)可得：

$$|C| \leqslant 2|C^*|$$

例如图 27.4.1。

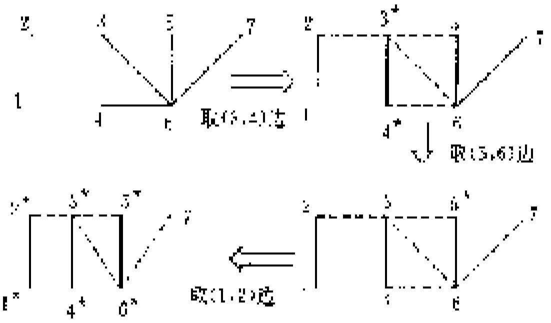


图 27.4.1

实际上最优顶点覆盖集为{1,5,6}.

习　题

1. 举例说明下列步骤是图顶点着色的近似算法

(1) $C \leftarrow \phi$.

(2) $E' \leftarrow E[G]$,

(3) 若 $E' \neq \phi$ 则作

【令(u,v)是E'的任意边，

$C \leftarrow C \cup \{u,v\}$,

从E'中消去和u或v关联的边】

(4)输出C。

2. 对上一问题提供的算法的近似程度进行讨论。

3. 举例说明下列步骤是图顶点着色的近似算法，并对它的近似程度进行讨论。

(1) $V' \leftarrow V, E' \leftarrow E, C \leftarrow \phi$。

(2) 在V'中找出度最大的顶点，设为u，作$C \leftarrow C \cup \{u\}$。

(3) $V' \leftarrow V' \setminus \{u\}$，从$E'$中消去与$u$关联所有的边。

(4) 若$E' \neq \phi$则转2，否则输出C。

4. 试对下面的背包问题给出求解方法

$$\max z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$
$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leqslant b$$
$$x_i \geqslant 0 \quad i = 1,2,\cdots,n$$

5. 下面是图顶点着色算法。

(1) $i \leftarrow 1$。

(2) $C \leftarrow 1$。

(3) 当与 v_i 邻接的顶点着颜色 C，则作

【$C \leftarrow C+1$，v_i 着以颜色 C】

(4) $i \leftarrow i+1$，当 $i \leqslant n$ 时，转 2，否则结束。

试对该算法进行分析。

6. 试设计一种下面背包问题的近似算法

$$\max z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$
$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leqslant b$$
$$x_i \geqslant 0 \qquad \text{整数}$$

第 28 章　密码学简介

密码作为一种技术源远流长，已有过它的不平凡的成就。但作为一门学科则是近若干年的事，并已经在许多领域得到广泛的应用。它也是算法复杂性理论引人入胜的新领域。随着计算机网络广泛的应用，信息时代的来临，信息的保护是至关重要的。密码是一种有效可行的方法。有效意味着使得窃取信息成为不可能，可行指的是代价比较低。算法复杂性理论是近代密码学的基础，所以密码学也成为复杂性理论引人入胜的一个分支。

28.1　什么是密码?

图 28.1.1 指出，若发信方要通过公共信道向收信方送去一明文 m，首先对 m 进行 E 变换，E 称为加密算法，k 称为密钥，密钥它是通信双方通过秘密信道私下约定的，也只有他们自己掌握，任何第三者都不知道。$C(=E_k(m))$ 称为密文。实际上，密文 C 是对明文 m 作含参数 k 的变换，收信方得到密文 C 以后，对 C 作 E_k 的逆变换 D_k，从而恢复为明文 m，即

$$m = D_k(C)$$

称 D_k 为解密算法。

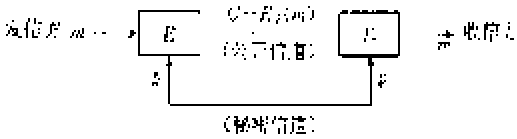


图 28.1.1

可见明文 m 变换为密文 C，一是加密算法 E，一是密钥 K。要求即使加密算法 E 公开了，只要密钥不知道，第三者即使窃取了密文 C，但仍然无法求得明文 m。

举例如下，设明文 m 为：

Secure message transmission is of extreme importance in information based society

最简单的例子是将明文 5 个字符分为一段，然后每段从右向左倒过来写得：

ruces sseme rtega imsna noiss efosi mertx opmie cnatr ninie amrof bnoit sdesa teico y，比如 secur 倒过来写成 ruces，余此类推，密钥是 5 这个数。

又如加密算法是作下列英文字母的变换：

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$
B	E	I	J	N	G	C	H	A	D	F	K	L	M	O	P	Q	R	S	T	U	V	W	X	Y	Z

这个变换表是根据密钥 K 为 Beijing China 而得到的，若每一个英文字母都只允许出现

一次，则记为BEIJNGCHA。后面就是25个英文字母除去密钥以外的字母依先后顺序写上。这样上面的明文经加密变换后为：

SNIURN MNSSBCN TRBMSMASSON AS OG NXTRNMN 这样形成的加密变换，称之为单表置换。

ALPORTBMIN. AM AMGORLBTAOM EBSNJ SOIANTY

我们知道这样的单表置换共有26!种，即26个英文字母的全排列，事实上$26!\approx4\times10^{26}$，若使用每秒10^7次超高速计算机，进行穷行搜索大约需要1.28×10^{12}年。若从这一标准来看，似乎这个加密方法可以说安全了。但是事实上并不是这样，若采用统计分析的方法分析这样一种单表置换的密码系统，不难破译它，由于英文字母出现的频率并不寻常地有规律，比如e出现的频率高达0.1308，还有t,a,o,n,r,i,s,h出现的频率分别大约为0.1045，0.0856，0.0797，0.0707，0.0677，0.0627，0.0607，0.0528。这几个字母以外频率明显下降，最低的为Z不到0.001，差别显著。所以由密文统计得出频率最高的字母(本例中是N)很可能就是e。英文字母的这样一种统计特性为破译提供了有力的帮助。另外还有可利用的特性为有些高频率的前后缀，比如he出现较多，eh很少见，th较多，ht较少等等。在破译中都被利用来作为依据使得单表置换显得极不安全了。

下面再介绍一种加密方法，其性能比单表置换的密码系统要好。它没有单表置换所具有的那种统计特性，如下表所示：

B	E	I	N	G
C	H	A	D	F
K	L	M	O	P
Q	R	S	T	U
V	W	X	Y	Z

译码过程如下：

(1) 将明文相邻字母写成一对，若遇到重复字母对，插进X将它们分开，使每一对字母都不相同。

(2) 若两个不同的字母a_1a_2在表的对角线上，由此对角线确定的一矩形的其它两个顶点，记为b_1b_2，b_1和a_1在同一行，b_2和a_2在同一行，则将a_1a_2加密为b_1b_2。

(3) 若a_1a_2在表内同一行，a_1的右邻字母为c_1，a_2的右邻字母为c_2，则将a_1a_2加密为c_1c_2。设第4列的右邻为第一列。

(4) 若a_1a_2在表的同一列，a_1的下方字母为d_1，a_2的下方字母为d_2，则将a_1a_2加密为d_1d_2。同上一样，第一行可看作第5行的下方。

还是以前面提到的明文为例，先将明文写成

Se cu re me sx sa ge tr an sm is si on is of ex tr em ei

mp or ta nc ei ni nf or ma ti on ba se ds oc ie ty

这儿对于*message*，改为*me sa sa ge*，密文如下：

RI FQ WU LI XI XM BI US IU XS AX XA TD AX. PD

IW US IL IN DK LT SD BD IN GN GD LT SM SN TD

IC RI AT KD NI YN.

可见一个字母它的密文是什么要看它和另一个字母如何组合了。

再介绍一种称之为 Vigenere 密码的。令 26 个英文字母依次对应于 0,1,2,…,25，即 a 为 0，b 为 1，…，z 为 25。

$m=m_1m_2\cdots m_n$ 是英文字符串的明文。

$k=k_1k_2\cdots k_r$ 是英文字符串的密钥。

则加密后的密文

$C=c_1c_2\cdots c_n$

其中

$$c_i=m_i+k_i \quad \mathrm{mod}\,26, \quad i=1,2,\cdots,n$$

若 $n>m$，可以考虑密钥 k 是 $k_1k_2\cdots k_r$ 的循环反复，还是以前面的例子来说明

m=S e c u r e m e s s a g e t r a n s m i s s i o n i s o f

k B E I J I N G C H I N A B E I J I N G C H I N A B E I J I

C=T I K D Z R S G Z A N G F X Z J Q F S K Z A V O P M A T N

e x t r e m e i m p o r t a n c e i n i n f o r m a t i o n

N G C H I N A B E I J I N G C H I N A B E I J I N G C H I N

R D V Y M Z E I Q X X Z G G P J M V N J R N X Z Z G V P W A

b a s e d s o c i e t y

A B E I J I N G C H I N

B B W M M A B I K L B L

这里密文是

T I K D Z R S G Z A N G F X Z J Q F S K Z A V O P M A T N R D V Y

M Z E I Q X X Z G G P J M V N J R N X Z Z G V P W A B B W M M A

B I K L B L

如明文 S 为 18，密钥 B 为 1，18+1=19，故对应的密文为 T。又 r 为 17，I 为 8，17+8=25，故对应的密文为 Z。又如 t 为 19，I 为 8，19+8=27≡1 mod26，故对应密文为 B，余此类推。

所以 Vigenere 密码明文字母的密码不是唯一的，因而都不具有单表置换那种统计特性。不过英文的一些固有的特性也被利用对 Vigenere 密码进行攻击。英文中如 *ed*、*ing*、*th*、*th*、*tion*、*sion* 出现的频率比较高，只要其中某一个前后的间隔字母数正好是密钥长度的倍数，则它们的密文将是相同。读者不难发现 message 中的 ss 和 transmission 中 ss 间的距离正好是 12，和密钥 *BEIJINGCHINA* 的长度相同，故密文 *ZA* 同时出现。这就成为破译 Vigenere 密码的突破口。总之 Vigenere 密码也不是安全的。

上面所介绍的几种方法是属于这样一种密码系统，加密用的密钥和解密用的密钥都是一样的，称其为对称密码或传统密码。这种系统有这样一种缺点，若在一块有 n 个用户

的公共网络上，两两用户之间都必须有自己的密钥，故共需 $C(n,2)$ 个，以 $n=1000$ 为例，$C(1000,2)=499500$ 故需要约 50 万个密钥，若考虑到密钥的更换，那将是不胜其繁的，另外每个用户必须记住与其通信的 $n-1$ 个用户的密钥，这也造成困难，若记在本子或其它载体上，这本身就带有极大不安全性。

1976 年 Diffe 和 Hellmein 在一篇文章中提出了公钥密码的想法。假定每一用户使用的加密与解密密钥是不相同的，则可将加密密钥公开，而解密密钥保密，只有用户自己知道，当然这样做必须保证加密密钥的公开不至于危及解密密钥的安全。设用户 A 有加密密钥 E_A 和解密密钥 D_A，同样用户 B 有加密密钥 E_B 和解密密钥 D_B，E_A 和 E_B 公布在公开的密钥文件上，若 A 要与 B 通信，可在密码本上查出 B 的加密密钥 E_B，用它来加密明文 m，即

$$C=E_B(m)$$

B 收到 C 后，以 D_B 对 C 进行解密以恢复为明文。即

$$D_B(c)=D_B(E_B(m))=m$$

当 Diffie 和 Hellman 提出上面公钥密码的构想时，还不存在实际的公钥例子，两年以后，几乎同时出现了背包公钥密码和 RSA 公钥密码，此后相继有若干公钥密码被提出，从此密码学的研究进入了发展阶段。

28.2 背包公钥密码

对子背包问题：

$$\sum_{i=1}^{n} a_i x_i = b,\ x_i=0 \text{ 或 } 1,\ i=1,2,\cdots,n$$

前面已证明它属于 NPC 类，但并非任何背包问题都是难解的，比如由超递增序列 $\{a_1,a_2,\cdots,a_n\}$ 构成的背包问题就不是难解的。所谓超递增即满足

$$\sum_{j=1}^{i-1} a_j < a_i,\ i=2,3,\cdots,n$$

的序列 $\{a_1,a_2,\cdots,a_n\}$。例如：2，5，11，23，47 就是超递增的，

$$2x_1+5x_2+11x_3+23x_4+47x_5=77$$

$$x_i=0 \text{ 或 } 1,\ i=1,2,3,4,5$$

不难看出它的解只能是 $x_5=1$，于是 $2x_1+5x_2+11x_3+23x_4=30$，所以只能是 $x_4=1$，∴ $2x_1+5x_2+11x_3=7$，故 $x_3=0$，取 $2x_1+5x_2=7$ 于是有解 $x_1=x_2=x_4=x_5=1$，$x_3=0$。

背包公钥密码是选一超递增序列：

$$a_1,a_2,\cdots,a_n$$

选取 $m>2a_n$ 和 w；$(w,m)=1$，作变换：

$$b_i=wa_i \mod m \qquad i=1,2,\cdots,n$$

于是得序列：

$$b_1,b_2,\cdots,b_n$$

将它作为公钥公开，$\{b_1,b_2,\cdots,b_n\}$并也不是超递增序列。令明文 m 为

$$m = m_1m_2\cdots m_n$$

$$m_i = 0 \text{ 或 } 1,\quad i = 1,2,\cdots,n$$

即 m 为 n 位二进制数。加密算法所得密文 C 为：

$$C = m_1b_1 + m_2b_2 + \cdots + m_nb_n$$

而从密文 C 求得 $m_1,m_2,\cdots,m_n$ 则是典型的背包问题。

若知道 w，由于 $(w,m)=1$，故存在 $\overline{w}$ 满足：

$$w\overline{w} \equiv 1 \mod m$$

则解密可如下进行：

$$\overline{w}C \equiv \overline{w}b_1m_1 + \overline{w}b_2m_2 + \cdots + \overline{w}b_nm_n \mod m$$

由于

$$\overline{w}b_i \equiv a_i \mod m \quad i = 1,2,\cdots,n$$

所以

$$m_1a_1 + m_2a_2 + \cdots + m_na_n \equiv \overline{w}C \mod m$$

这是超递增序列的背包问题。

由于背包公钥密码的公钥$\{b_1,b_2,\cdots,b_n\}$是由超递增序列$\{a_1,a_2,\cdots,a_n\}$经过变换得到的，所以并不是纯正的背包问题，虽然经过变换还是留有漏洞，所以也已被击破。但作为一种加密思想还在发展。下面我们来看一个例子。

例 已知$\{3,6,10,22,44\}$是超递增序列，取 $m=89$，$w=67$，$\overline{w}=4$，使得

$$67 \times 4 = 268 \equiv 1 \mod 89$$

$$b_i \equiv wa_i \mod 89$$

所以

$$b_1 \equiv 67 \times 3 \mod 89 = 23 \mod 89$$

$$b_2 \equiv 67 \times 6 \equiv 46 \mod 89$$

$$b_3 = 47, b_4 = 50, b_5 = 11$$

则对于明文 $m=11010$，密文 $C=23+46+50=119$。

在收到密文 119 后，作

$$4 \times 119 = 476 \equiv 31 \mod 89$$

$$31 = 3m_1 + 6m_2 + 10m_3 + 22m_4 + 44m_5$$

从而有

$$m_5 = 0, m_4 = 1, m_3 = 0, m_2 = 1, m_1 = 1$$

故 $m=11010$。

28.3 RSA 公钥密码

Rivest，Shamir 和 Adleman 提出的 RSA 公钥密码是基于数论中的 Euler 定理：

Euler 定理 若 a 和 m 互素，则

$$a^{\varphi(m)} \equiv 1 \mod m$$

其中 $\Phi(m)$ 为欧拉函数，若 $m=P_1^{a_1}P_2^{a_2}\cdots P_r^{a_r}$，则

$$\Phi(m)=m(1-\frac{1}{P_1})(1-\frac{1}{P_2})\cdots(1-\frac{1}{P_r})$$

Euler 定理的证明略去，仅介绍如何利用 Euler 定理构造公钥密码。

(1) 选择素数 p 和 q，(保密)；

(2) 计算 $n=pq$，(公开)；

(3) 计算 $\Phi(n)=(p-1)(q-1)$，(保密)；

(4) 选择加密密钥 e(公开)满足：

$$(e,\Phi(n))=1$$

(5) 计算解密密钥 d(保密)满足

$$d\cdot e\equiv 1 \mod \Phi(n)$$

加密算法 E：

$$E: C\equiv m^e \mod n$$

解密算法 D：$C^d \mod n$

$$C^d\equiv(m^e)^d \mod n$$

而

$$ed=k\Phi(n)+1$$

所以：

$$C^d\equiv m^{k\Phi(n)+1}\equiv(m^{\Phi(n)})^k m\equiv m \mod n$$

RSA 公钥密码系统采取公开 n，但对 $\Phi(n)=(p-1)(q-1)$ 进行保密的方法是基于因式分解 n 的困难性。到目前为止，最好的因素分解算法复杂度为 $O(e^{\sqrt{\ln n\ln\ln n}})$。若使用每秒 $10^{?}$ 次的高速计算机来分解一个 100 位的 10 进制数约需 74 年时间，若要分解 200 位的 10 进制数约需 3.8×10^9 亿年。为了确保密码系统的安全，要求 p 和 q 都是 100 位(10 进制数)的素数。由于大数的分解是一个热门的问题，最新的结果 129 位十进制数已被多国科学家合作在网络上通过分布计算被破解了。

从上可知若 n 被分解，则 RSA 公钥密码将被攻破，因此 RSA 的安全性并不是绝对的安全。

例 $p=43$，$q=59$，$n=pq=2537$，

$$\Phi(n)=42\times58=2436, e=13, d=937$$

若明文 m=public key encryptions

先将明文转化为数字，比如每组 4 位数

1520，0111，0802，1004，2404

1302，1724，1519，0814，1318

这样 1520 的密文为：

$$1520^{13} \mod 2537=0095$$

余此类推。

28.4 数字签名

在传统的密码通信双方用的密钥是私下约定的，比如A向B送去信息m，经加密得：

$$C = E_k(m)$$

若B对密文进行修改，或A否认向B送去m，因此所发生的争端将很难解决。RSA公钥可如下进行加密。若A欲向B发去明文m，则A要用自己的解密密钥对m作变换，再用B的加密密钥加密，即

$$C = E_B(D_A(m))$$

B收到密文C后，解密运算为：

$$E_A(D_B(C))$$

因

$$D_B(C) = D_B(E_B(D_A(m))) = D_A(m)$$

所以

$$E_A(D_B(C)) = E_A(D_A(m)) = m$$

因此若A否认向B送去m，B可向公证部门出示C，因只有A才掌握解密密钥D_A，B是无法篡改密文内容的。

对密码学就作这些介绍，进一步的知识可在相关的专业书中查到。

28.5 Hash 算法

上面介绍的数字签名，无疑是RSA公钥密码系统很引人入胜的特点，它的功能可用于确认。在信息时代，公共信道上的信息不一定都要求保密，更多只要求确认，确认收到的信息确实是发信人发的，没有任何篡改，这种功能还可以用在保护信息不被非法篡改的其他领域，比如储存在数据库的文件，为了避免被非法修改，也可在文件后面附着这个标志。密码学提供了这样的功能。

下面介绍Hash算法如下：

(1) Hash算法可用于任意大小的信息。

(2) Hash算法输出的是固定的大小。

(3) Hash算法的计算步骤是容易的。

(4) 已知信息M的$Hash(M)$，不可能找到M'使得

$$Hash(M') = Hash(M)$$

密码是如何构造这样的Hash算法呢？假定有一个加密算法E是一块密码，也就是对固定长度的比特位的明文进行加密的一种加密算法，加密后的密文也是固定长度的比特位。比如著名的数据加密标准DES就是这样的一种块码，它将64比特的明文m加密成64比特的密文C，密钥k也是64比特，其实真正起作用的只有56比特，其余8位是奇偶校验位。一般说来密文C的每一比特都和明文m以及密钥k的每个比特位有密切相关，以DES为例，不论密钥k还是明文m每改变一个比特都对密文C产生很普遍的变化，大致

约一半的比特位发生变化)。比如利用 DES 便可构造 Hash 算法，设 K 为密钥，用图 28.5.1 表示加密变换。对信息 $m=m_1m_2\cdots m_l$，其中 m_i 都是 64 比特的明文块码，$i=1,2,\cdots,l$。密钥 k，构造 $Hash(m)$ 如下(见图 28.5.2)：

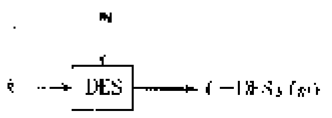


图 28.5.1

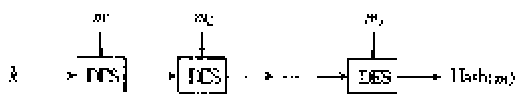


图 28.5.2

即令

$$k_1 = k,$$

$$k_i = DES_{k_{i-1}}(m_{i-1}),\quad i = 1,2,\cdots,l$$

$$Hash(m) = DES_{k_l}(m_l)$$

最后的 $Hash(m)$ 也只有 64 比特，但它和 $m=m_1m_2\cdots m_l$ 的每一比特，以及密钥 k 的每一比特都密切相关。也就是 m 的每一比特的改变将不能不对 $Hash(m)$ 发生明显的改变，而且不能构造 $m'\neq m$，而有

$$Hash(m') = Hash(m)$$

当然假定不掌握密钥 k。

上面是拿 DES 作为例子，其实对一般的固定长度的块码，上面的方法也都是对的。不过 DES 确定有这样的特性，使得密文 C 对明文 m 和密钥 k 都非常敏感，由它构造的 Hash 变换是很有用的。

习 题

1. 设 $p=37$，$q=43$，$n=37\times43=1591$，试求 d 与 e 构造一 RSA 公钥密码系统，并以 *Beijing China* 为例说明其加密解密过程。

2. 已知明文：

Secure message transmission is of extreme importance in information based society military diplomatic and corporate data transmissions must be safeguarded so also must the account of every individual who has an automatic teller bank account.

密钥为 *Tsinghua University*

试分别用(1)单表替换，(2)词组密钥，(3)*Play fair* 密码进行加密。

试用一高级语言编 DES 加密算法

第29章　LP问题的多项式算法

29.1　Klee和Minty举例

自从40年代提出单纯形法以来，它一直在线性规划成功地扮演着重要角色，应用的范围迅速地扩大，很少出现什么麻烦。但问题本身约束条件有矛盾者除外，这时问题本来就没有解，而单纯形法又没有无解的很好判断准则。72年Klee和Minty给出一个例子，说明利用单纯形法有可能时间复杂性是指数型的，也就是说单纯形法可能失效。于是提出了一个问题：线性规划本身究竟是不是属于NP困难问题？有的猜测它是。79年当时的苏联学者Хачиян利用一种叫做椭球算法求它的解，证明它的时间复杂性有多项式界。算法本身实际效果很不理想，但在西方主要是美国引起一场不小的震动，似乎NP=P的神话已隐隐约约在望，其实Хачиян的算法只有理论价值。证明了线性规划就是P类问题。而且理论上提供了一种判断无解的准则。它是理论上提供了一判断准则，因Хачиян算法的实际表现实在太差了。现在就从Klee和Minty的例子说起，而且仅就$n=2$和$n=3$详细说明，一般的推演就从略了。

$n=2$：

$$
\begin{aligned}
&\max z = x_2\\
&\quad x_1 \geq \varepsilon\\
&\quad x_1 \leq 1\\
&\quad x_2 - \varepsilon x_1 \geq 0\\
&\quad x_2 + \varepsilon x_1 \leq 1\\
&\quad x_1, x_2 \geq 0
\end{aligned}
\tag{29.1.1}
$$

引进松弛变量$\xi_1, \xi_2, \eta_1, \eta_2$使

$$
\begin{aligned}
&\max z = x_2\\
&\quad x_1 - \xi_1 = \varepsilon\\
&\quad x_1 + \eta_1 = 1\\
&\quad x_2 - \varepsilon x_1 - \xi_2 = 0\\
&\quad x_2 + \varepsilon x_1 + \eta_2 = 1\\
&\quad x_1, x_2, \xi_1, \xi_2, \eta_1, \eta_2 \geq 0,\ 0 < \varepsilon < 1
\end{aligned}
$$

图29.1.1的D域是问题的可行解域，V_1, V_2, V_3, V_4是可行解域的4个顶点。若单纯形法沿着

$$V_1 \to V_2 \to V_3 \to V_4$$

V_1V_2线：$x_2 - \varepsilon x_1 = 0$，$V_2V_3$线：$x_1 = 1$

V_3V_4线：$x_2 + \varepsilon x_1 = 1$，$V_4V_1$线：$x_1 = \varepsilon$

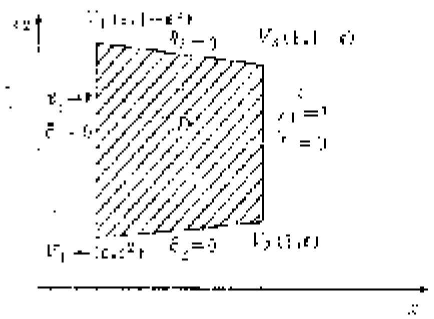


图 29.1.1

故V_1点：$x_1=\varepsilon, x_2=\varepsilon^2, \xi_1=0, \eta_1=1-\varepsilon, \xi_2=0, \eta_2=1-2\varepsilon^2$；

V_2点：$x_1=1, x_2=\varepsilon, \xi_1=1-\varepsilon, \eta_1=0, \xi_2=0, \eta_2=1-2\varepsilon$；

V_3点：$x_1=1, x_2=1-\varepsilon, \xi_1=1-\varepsilon, \eta_1=0, \xi_2=1-2\varepsilon, \eta_2=0$；

V_4点：$x_1=\varepsilon, x_2=1-\varepsilon^2, \xi_1=0, \eta_1=1-\varepsilon, \xi_2=1-2\varepsilon^2, \eta_2=0$。

$n=3$时，

$$
\begin{aligned}
\max z &= x_3 \\
x_1 &\geqslant \varepsilon \\
x_1 &\leqslant 1 \\
x_2 - \varepsilon x_1 &\geqslant 0 \\
x_2 + \varepsilon x_1 &\leqslant 1 \\
x_3 - \varepsilon x_2 &\geqslant 0 \\
x_3 + \varepsilon x_2 &\leqslant 1 \\
x_1, x_2, x_3 &\geqslant 0, \ 0 < \varepsilon < \frac{1}{2}
\end{aligned}
\tag{29.1.2}
$$

引入松弛变量$\xi_i, \eta_i, i=1,2,3$，将(29.1.2)式改写为

$$
\begin{aligned}
\max z &= x_3 \\
x_1 - \xi_1 &= \varepsilon \\
x_1 + \eta_1 &= 1 \\
x_2 - \varepsilon x_1 - \xi_2 &= 0 \\
x_2 + \varepsilon x_1 + \eta_2 &= 1 \\
x_3 - \varepsilon x_2 - \xi_3 &= 0 \\
x_3 + \varepsilon x_2 + \eta_3 &= 1 \\
x_i, \xi_i, \eta_i &\geqslant 0, i = 1,2,3
\end{aligned}
$$

它的可行解域见图 29.1.2，为：

$$
\begin{aligned}
&V_1: x_1=\varepsilon, \quad x_2=\varepsilon^2, \qquad x_3=\varepsilon^3 \\
&V_2: x_1=1, \quad x_2=\varepsilon, \qquad x_3=\varepsilon^2 \\
&V_3: x_1=1, \quad x_2=1-\varepsilon, \quad x_3=\varepsilon-\varepsilon^2
\end{aligned}
$$

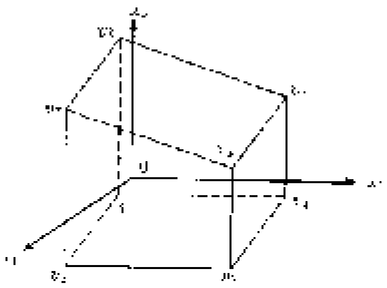


图 29.1.2

$$V_4: x_1=\varepsilon,\quad x_2=1-\varepsilon^2,\quad x_3=\varepsilon-\varepsilon^3$$
$$V_5: x_1=\varepsilon,\quad x_2=1-\varepsilon^2,\quad x_3=1-\varepsilon+\varepsilon^3$$
$$V_6: x_1=1,\quad x_2=1-\varepsilon,\quad x_3=1-\varepsilon+\varepsilon^2$$
$$V_7: x_1=1,\quad x_2=\varepsilon,\quad x_3=1-\varepsilon^2$$
$$V_8: x_1=\varepsilon,\quad x_2=\varepsilon^2,\quad x_3=1-\varepsilon^3$$

因 $0<\varepsilon<\frac{1}{2}$，故 $\varepsilon^3<\varepsilon^2<\varepsilon$，$\varepsilon^2<\varepsilon-\varepsilon^3$

故沿着 $V_1\to V_2\to V_3\to V_4\to V_5\to V_6\to V_7\to V_8$，

$$x_3^{(1)}<x_3^{(2)}<x_3^{(3)}<x_3^{(4)}<\cdots<x_3^{(7)}<x_3^{(8)}$$

其中 $x_3^{(i)}$ 表示 V_i 点的 x_3 坐标。

注意到 $V_1^{(3)}\to V_2^{(3)}\to V_3^{(3)}\to V_4^{(3)}$ 它们的 (x_1,x_2) 坐标正好和 $n=2$ 时的 $V_1^{(2)}\to V_2^{(2)}\to V_3^{(2)}\to V_4^{(2)}$ 完全一致，这里 $V_i^{(n)}$ 表示 n 维例子中的顶点 V_i，而后面的顶点 $V_5^{(3)}\to V_6^{(3)}\to V_7^{(3)}\to V_8^{(3)}$ 的 (x_1,x_2) 坐标和 $V_4^{(2)}\to V_3^{(2)}\to V_2^{(2)}\to V_1^{(2)}$ 的顺序相反。

下面给出一般 n 维空间的例子。

$$\begin{aligned}
&\max\ x_n\\
&x_1-\xi_1=\varepsilon\\
&x_1+\eta_1=1\\
&x_j-\varepsilon x_{j-1}-\xi_j=0\\
&\qquad\qquad j=2,3,\cdots,n\\
&x_j+\varepsilon x_{j-1}+\eta_j=1\\
&x_j,\xi_j,\eta_j\geqslant 0,\ j=1,2,\cdots,n
\end{aligned}\tag{29.1.3}$$

设 $0<\varepsilon<\frac{1}{2}$，可以证明问题(29.1.3)的可行解域有 2^n 个顶点，存在一条通过这 2^n 顶点的顺序：

$$V_1,V_2,\cdots,V_{2^n}$$

使得

$$x_n^{(1)}<x_n^{(2)}<\cdots<x_n^{(2^n)}$$

其中 $x_n^{(i)}$ 为 V_i 点的 x_n 坐标，这结论可通过数学归纳法来证明。高维情况不如 2 维到 3 维那样十分直观，但道理是一样的，证明从略。

29.2 Хачцян（哈奇扬）算法

关于哈奇扬算法的讨论应有三部分内容，一是将线性规划问题转化为一组不等式，哈奇扬算法是针对这组不等式进行求解，这是第二部分内容，最后是算法的正确性证明及复杂性估计，第三部分从略。

现在先介绍如何将线性规划问题转化为一组不等式。

线性规划转化为一组不等式

设原问题为：

$$\begin{aligned} \max z &= \boldsymbol{CX} \\ \boldsymbol{AX} &\leqslant \boldsymbol{b} \\ \boldsymbol{X} &\geqslant \boldsymbol{0} \end{aligned} \qquad (P)$$

其对偶问题为：

$$\begin{aligned} \min w &= \boldsymbol{Yb} \\ \boldsymbol{YA} &\geqslant \boldsymbol{C} \\ \boldsymbol{Y} &\geqslant \boldsymbol{0} \end{aligned} \qquad (D)$$

其中

$$\boldsymbol{A} = (a_{ij})_{m\times n} \qquad \boldsymbol{C} = (c_1\ c_2\ \cdots\ c_n)$$

$$\boldsymbol{X} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \qquad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

$$\boldsymbol{Y} = (y_1\ y_2\ \cdots\ y_m)$$

如若 $\boldsymbol{X}$ 和 $\boldsymbol{Y}$ 分别是问题(P)和(D)的解，则应有

$$\begin{aligned} \boldsymbol{CX} &= \boldsymbol{Yb} \\ \boldsymbol{YA} &\geqslant \boldsymbol{C} \\ \boldsymbol{AX} &\leqslant \boldsymbol{b} \\ \boldsymbol{X} &\geqslant \boldsymbol{0}, \boldsymbol{Y} \geqslant \boldsymbol{0} \end{aligned} \qquad (*)$$

根据对偶理论(P)的解 $\boldsymbol{X}$ 对(D)的解 $\boldsymbol{Y}$，必有 $\boldsymbol{CX} \leqslant \boldsymbol{Yb}$，若$(P)$有最优解 $\boldsymbol{X}^*$，(D)有最优解 $\boldsymbol{Y}^*$，则 $\boldsymbol{CX}^* = \boldsymbol{Y}^*\boldsymbol{b}$ 故问题导至求解一组不等式

$$\begin{aligned} \boldsymbol{CX} &\geqslant \boldsymbol{Yb} \\ \boldsymbol{AX} &\leqslant \boldsymbol{b} \\ \boldsymbol{YA} &\geqslant \boldsymbol{C} \\ \boldsymbol{X} &\geqslant \boldsymbol{0} \\ \boldsymbol{Y} &\geqslant \boldsymbol{0} \end{aligned}$$

如若 $\boldsymbol{CX}^* = \boldsymbol{Y}^*\boldsymbol{b}$，则 $\boldsymbol{X}^*$ 必是问题(P)的解。

前面已将线性规划问题转换成一组等价的不等式组，设该不等式组为

$$\boldsymbol{A}(\boldsymbol{X}) \leqslant \boldsymbol{b} \qquad (29.2.1)$$

式中 $A=(a_{ij})_{m\times n}$，$b=(b_1,b_2,\cdots,b_m)^T$。哈奇扬(Хачиян)算法是解这样的不等式组的方法，它实际上是将前苏联苏尔(Shor)用来解非线性规划的椭球算法推广到解线性规划上来。正如前面已提到过，它在理论上有很高的价值，实际效果很差。下面简要叙述它的主要成果，一般着重于理解它的几何的直观意义，而不一定给出它的证明。令

$$L = mn + \left\lceil \sum_{i=1}^{m}\sum_{j=1}^{n}\log_2(|a_{ij}|+1) + \sum_{i=1}^{m}\log_2(|b_i|+1) + \log_2(mn) \right\rceil + 1$$

L 可以看作是问题(29.2.1)输入所需机器代码的长度，也就是将 A，b 化为二进位数后 0 和 1 的位数。一个整数 a 所含的二进位数的位数为 $\lceil \log_2 a \rceil$。

定理 1 线性不等式组(29.2.1)有解的充要条件是

$$a_i X < b_i + \varepsilon \qquad i=1,2,\cdots,m \tag{29.2.2}$$

有解，式中 $\varepsilon=2^{-L}$，a_i 是矩阵 A 的第 i 行行向量，即

$$A=\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix}$$

设 Q 是 $n\times n$ 的非奇异方阵，$t\in R^n$，定义仿射变换 T：

$$Y = T(X) = t + QX$$

对于 n 维空间的单位球

$$S_n = \{X \mid X\in R^n, X^TX\leqslant 1\}$$

在 T 的作用下变为椭球体，即

$$T(S_n) = \{Y \mid Y\in R^n, (Y-t)^TB^{-1}(Y-t)\leqslant 1\}$$

式中 $B=QQ^T$ 是正定矩阵，而且 T 是可逆变换。

引理 1 设 $B=(A\vdots b)$，则 B 中任一 $k\times k$ 阶子方阵 D_k，都有

$$|\det(D_k)| < 2^L$$

假定 $1\leqslant k\leqslant \min(m,n)$。

引理 2 如若 $S_1\subseteq S_2\subseteq R^n$，则

$$T(S_1)\subseteq T(S_2)$$

引理 3 如若 $S\subseteq R^n$，S 有容积 V，则有

$$\mathrm{vol}(T(S)) = V\cdot|\det(Q)|$$

$\mathrm{vol}(T(S))$ 表 $T(S)$ 的体积。

引理 4 a 是 R^n 空间一向量，则存在一旋转变换 R，使

$$R(a) = (\|a\|,0,\cdots,0)^T$$

$\|a\|$ 是 a 的长度。

引理 5 对于 n 维空间有界的凸多面体

$$P=\{X \mid X\in R^n, |X|\leqslant C, AX\leqslant b\}$$

存在有限个向量 $a_i\in P$，$i=1,2,\cdots,k$，使得

$$P=\{\sum_{i=1}^{n}|\lambda_i\geqslant 0,\sum_{i}\lambda_i=1\}$$

其中 C 是某一常数。

引理 6 $V_0,V_1,\cdots,V_n\subset R^n$，且 $V_1-V_0,V_2-V_0,\cdots,V_n-V_0$ 线性无关，则 n 维空间的单纯形

$$P=\{\sum_{i=0}^{n}\lambda_i V_i\,|\,\lambda_i\geqslant 0,i=0,1,\cdots,n,\sum_{i=0}^{n}\lambda_i=1\}$$

的体积

$$\mathrm{vol}(P)=\frac{1}{n!}\left|\det\begin{pmatrix}1&1&\cdots&1\\V_0&V_1&\cdots&V_n\end{pmatrix}\right|$$

这个引理在 $n\leqslant 3$ 时不难证明。

引理 7 n 维空间球 $|X|\leqslant n2^L$ 内满足 $AX<b$ 的点集若体积至少为 $2^{-(n+1)L}$。

下面介绍哈奇扬椭球算法。

(1) 计算 $AX<b$ 的输入长度 L。

(2) $j\leftarrow 0,t_0\leftarrow 0,B_0\leftarrow n^2 2^{2L}I$。

(3) 若 t_j 是解则停止；否则若

$$j>16n(n+1)L$$

则 $AX<b$ 无解。

(4) 把 t_j 代入 $AX<b$ 任选一使 $a_it_j\geqslant b_i$ 成立的不等式，$a\leftarrow a_i$。

(5) 计算

$$t_{j+1}\leftarrow t_j-\frac{1}{n+1}\frac{B_ja^T}{\sqrt{aB_ja^T}}$$

$$B_{j+1}\leftarrow\frac{n^2}{n^2-1}\left[B_j-\frac{2}{n+1}\frac{(B_ja^T)(B_ja^T)^T}{aB_ja^T}\right]$$

$j\leftarrow j+1$　　转(3)

算法中构造的椭球 E_j

$$E_{j+1}=\{X|(X-t_{j+1})^TB_{j+1}^{-1}(X-t_{j+1})\leqslant 1\}$$

的迭代过程用图 29.2.1 表示。

而且

$$\frac{\mathrm{vol}(E_{j+1})}{\mathrm{vol}(E_j)}\leqslant a<1$$

设 E_j 是以 t 为中心的椭球包含有解集 S，$\mathrm{vol}(S)\geqslant 2^{-(n+1)L}$，如若 t 不满足某一不等式

$$a_iX<b_i$$

即 $a_iX\geqslant b_i$。超平面 $a_i(x-t)=0$ 将 E_j 分成两部分，设

$$\frac{1}{2}E_j[a_i]=E_j\cap\{x|a_i(x-t)\leqslant 0\}$$

$\frac{1}{2}E[a_i]$ 是 E_j 的半椭球。哈奇扬算法是由 E_j 及 t 去构造 E_{j+1}，每次所得的椭球都包含 S。

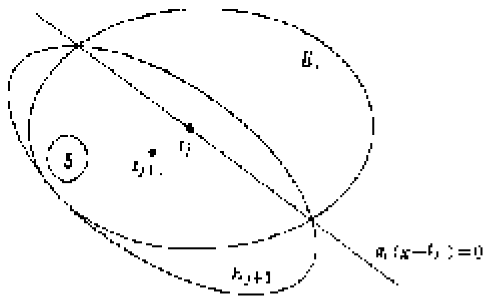


图 29.2.1

椭球的体积不断缩小，经过有限步后便可断定不等式组

$$AX < b$$

是否有解. 判定线性规划问题是否有解，也就是约束条件是否合适. 没有合适的办法，哈奇扬算法至少理论上提供了一种途径。

哈奇扬算法在理论上的便宜，而它的实际效果差，这也是不难理解的，首先将问题化为标准形将问题规模大大地扩大了，椭球在高维空间收敛得也慢。它的理论的价值在于，说明线性规划是属于P类，理论上给出判定无解的条件。证明从略。

29.3 Karmarkar 算法

单纯形法求解线性规划问题是从作为可行解域的凸多面体的一个顶点出发，沿着凸多面体的棱转移到邻近一个顶点，使目标函数有所改善，如此反复进行，最后达到最优解，哈奇扬算法也是迭代过程，开始用一足够大的球将线性规划解包含在内，迭代的过程，不断用较小的椭球取代前面较大的椭球，过程一直保持将线性规划的解包含在内，如此循环反复，椭球收敛于问题的解.

Karmarkar 也是一个迭代过程，它从可行解域的凸多面体内一点出发，沿着可行解域多面体内部向最优解.

Karmarkar 算法是从标准型出发：

$$\begin{cases} \min CX \\ X \in \Omega \cap S \end{cases}$$

其中 $C=(c_1,c_2,\cdots,c_n)$，$X=(x_1,x_2,\cdots,x_n)^T$

$$\Omega=\{X \mid AX=0\},\ S=\{X \mid \sum_{i=1}^{n} x_i = 1,\ x_i \geqslant 0, i=1,2,\cdots,n\}$$

$$A=(a_{ij})_{m\times n},\ r(A)=m$$

同时假定

H_1：目标函数的最优值为零。

H_2：单纯形 S 的中心 $X^0=\left(\frac{1}{n},\frac{1}{n},\cdots,\frac{1}{n}\right)^T \in \Omega$

先解决如何将线性规划问题转换为 Karmarkar 标准型，然后再讨论 Karmarkar 算法及其原理。

假定原始的线性规划问题为

$$\begin{cases}\min z = CX \\ AX \geqslant b \\ X \geqslant 0\end{cases} \tag{29.3.1}$$

它的对偶问题是

$$\begin{cases}\max z = Yb \\ YA \leqslant C \\ Y \geqslant 0\end{cases}$$

由对偶定理知线性规划问题 29.3.1 有解等价于下面不等式方程组有解：

$$\begin{cases}AX \geqslant b \\ YA \leqslant C \\ CX = Yb \\ X \geqslant 0, Y \geqslant 0\end{cases} \tag{29.3.2}$$

其中 $A=(a_{ij})_{m\times n}$，$C=(c_1,c_2,\cdots,c_n)$，$b=(b_1,b_2,\cdots,b_m)^T$，$X=(x_1,x_2,\cdots,x_n)^T$，$Y=(y_1,y_2,\cdots,y_m)$

引进变量 u 及 v 将(29.3.2)转换成

$$\begin{cases}AX - u = b \\ YA - v = C \\ CX - YA = 0 \\ X \geqslant 0, y \geqslant 0\end{cases} \tag{29.3.3}$$

引进线性规划问题

$$\begin{cases}\min z = \lambda \\ AX - u + (b - AX^{(0)} + u^{(0)})\lambda = b \\ YA + v + (C - Y^{(0)}A - v^{(0)})\lambda = C \\ CX - Yb + (-CX^{(0)} + Y^{(0)}b)\lambda = 0 \\ X \geqslant 0, Y \geqslant 0, u \geqslant 0, v \geqslant 0, \lambda \geqslant 0\end{cases} \tag{29.3.4}$$

其中 $u^{(0)}$，$v^{(0)}$，$X^{(0)}$，$Y^{(0)}$ 是任意正向量，可将(29.3.4)的约束条件写为

$$\begin{cases}\bar{A}\bar{X} = \bar{b} \\ \bar{X} \geqslant 0\end{cases}$$

显然 $X=X^{(0)}$，$Y=Y^{(0)}$，$u=u^{(0)}$，$v=v^{(0)}$，$\lambda=1$ 是(29.3.4)的一个解，故(29.3.4)是有解的线性规划问题，$\lambda=0$ 时(29.3.4)的约束条件就是(29.3.3)。

由于假定 $\lambda\geqslant0$，故(29.3.4)的最优解有两种可能：$\lambda=0$ 或 $\lambda>0$。若 $\lambda=0$，说明(29.3.3)有解，亦即(29.3.1)有解。故(29.3.4)有最优解 $\lambda=0$，等价于(29.3.1)有最优解。

若(29.3.3)的最优解 $\lambda>0$，这时(29.3.1)无最优解，如若不然，假定(29.3.1)有解，则(29.3.3)有解，设(29.3.3)的解为：

$$X = X^*, Y = Y^*, u = u^*, v = v^*, \lambda = 0$$

是(29.3.4)的一个可行解，且目标函数值为零，这跟 $z^*>0$ 的假定矛盾。

于是线性规划(29.3.1)可化为目标函数的最优值为零的线性规划(29.3.4)了。将(29.3.4)记作

$$\min z = CX$$
$$AX = b$$
$$X \geqslant 0$$

下面引进分式变换 T，设

$$a = (a_1, a_2, \cdots, a_n)^T > 0$$

是一可行解。

$$T: y_i = \frac{x_i}{a_i}\Big/\left[1+\sum_{j=1}^{n}\frac{x_j}{a_j}\right], \quad i = 1,2,\cdots,n$$

$$y_{n+1} = 1 - \sum_{i=1}^{n} y_i$$

上面的分式变换 T，将 R^n 空间的 $X\geqslant 0$ 映射为 R^{n+1} 的单纯型 S_{n+1} 上

$$S_{n+1} = \{Y \mid \sum_{j=1}^{n+1} y_j = 1, y_j \geqslant 0, j = 1,2,\cdots,n+1\}$$

这是由于

$$\sum_{i=1}^{n} y_i + y_{n+1} = 1, y_j \geqslant 0, j = 0,1,2,\cdots,n+1$$

而且不难验证变换 T 将 $a=(a_1,a_2,\cdots,a_n)^T>0$ 转换成

$$\bar{a} = \left(\frac{1}{n+1}, \frac{1}{n+1}, \cdots, \frac{1}{n+1}\right)^T \in R^{n+1}$$

根据 T 变换的定义

$$\sum_{i=1}^{n} y_i = 1 - y_{n+1}$$

$$\therefore \quad y_{n+1} = 1 - \sum_{i=1}^{n} y_i = 1 - \frac{\sum_{j=1}^{n}\frac{x_j}{a_j}}{1+\sum_{j=1}^{n}\frac{x_j}{a_j}}$$

$$= \frac{1}{1+\sum_{j=1}^{n}\frac{x_j}{a_j}}$$

$$1 + \sum_{j=1}^{n}\frac{x_j}{a_j} = \frac{1}{y_{n+1}}$$

以上代入 T 变换，可得：

T 的逆变换为

$$x_i = \frac{a_i y_i}{y_{n+1}}, \quad i = 1,2,\cdots,n$$

在 T 变换的作用下，满足约束条件

$$\begin{cases} AX = b, \\ X \geqslant 0, \end{cases} \qquad (29.3.5)$$

的任意点 $X=(x_1,x_2,\cdots,x_n)^T$ 将变为

$$Y=(y_1,y_2,\cdots,y_{n+1})^T \in S_{n+1},$$

其中
$$S_{n+1}=\left\{Y\in R^{n+1} \,\middle|\, \sum_{j=1}^{n+1} y_j = 1, y_j \geqslant 0, j=1,2,\cdots,n+1\right\}.$$

而且若 $a=(a_1,a_2,\cdots,a_n)^T>0$ 是满足(29.3.5)的可行解，则在 T 变换作用下，映射为 S_{n+1} 的中心。

$$a \to \frac{1}{n+1}e = \frac{1}{n+1}(1,1,\cdots,1)^T \in R^{n+1}.$$

约束条件 $AX=b, X\geqslant 0$ 在 T 变换作用下变为

$$A\begin{pmatrix} a_1 & & & \mathbf{0} \\ & a_2 & & \\ & & \ddots & \\ \mathbf{0} & & & a_n \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} - y_{n+1} b = 0,$$

$$\sum_{j=1}^{n+1} y_j = 1,$$

$$y_j \geqslant 0, j=1,2,\cdots,n+1.$$

若 $A=(P_1P_2\cdots P_n)$，P_i 是 A 矩阵的第 i 列，$i=1,2,\cdots,n$，则上式可写成

$$\sum_{j=1}^{n} a_j P_j y_j - y_{n+1} b = 0,$$

或
$$\sum_{j=1}^{n+1} \bar P_j y_j = 0,$$

其中
$$\bar P_j = a_j P_j, \bar P_{n+1} = P_{n+1} = -b$$

简记为
$$\bar A Y = 0,$$

其中
$$\bar A = (\bar P_1 \bar P_2 \cdots \bar P_n \bar P_{n+1}), Y=(y_1,y_2,\cdots,y_n,y_{n+1})^T$$

总之在 T 的作用下有

$$\{X \in R^n \mid AX=b, X\geqslant 0\} \xrightarrow{T} \{Y\in R^{n+1} \mid \bar A Y = 0, eY=1, Y\geqslant 0\}$$

目标函数

$$CX = \frac{1}{y_{n+1}} \sum_{j=1}^{n} a_j c_j y_j.$$

所以由 $CX=0$ 可导致

$$\sum_{j=1}^{n} \bar C_j y_j = 0.$$

综上所述问题(29.3.1)可以变换为 Karmarkar 典型问题

$$\min z = \bar C Y$$
$$\bar A Y = 0$$

$$e^TX = 1$$
$$X \geq 0$$

并满足

(1) $a_0 = \frac{1}{n}e^T$ 是一个可行解；

(2) 目标函数在可行解域上的最优值为 0。

Karmarkar 算法设 $\alpha \in \left(0, \sqrt{\frac{n-1}{n}}\right)$，及收敛参数 q。

(1) $X^0 \leftarrow \frac{1}{n}e, k \leftarrow 0$；

(2) 若 $CX^k \leq 2^{-q}CX^0$ 则结束；

(3) $D \leftarrow diag(x_1^k, x_2^k, \cdots, x_n^k)$，

$B \leftarrow \begin{bmatrix} AD \\ e^T \end{bmatrix}$，

计算

$C_p \leftarrow [I - B^T(BB^T)^{-1}B]DC^T$，

$\bar{C}_p \leftarrow \frac{C_p}{\|C_p\|}$；

(4) 计算

$b \leftarrow \frac{1}{n}e - \alpha r\bar{C}_p$，

$r \leftarrow \frac{1}{\sqrt{n(n-1)}}$。

(5) $X^{k+1} \leftarrow \frac{Db}{e^TDb}, k \leftarrow k+1$，转(2)。

先举例将算法各步骤说清楚，对算法中的各步作较明确的解释，这便于以后讨论算法的原理。

例 $\min z = x_1$，

$x_2 - x_3 = 0$，

$x_1 + x_2 + x_3 = 1$，

$x_1, x_2, x_3 \geq 0$。

本例中

$A = (0, 1, -1)$，而且从观察可知 $x_2 = \frac{1}{2}, x_3 = \frac{1}{2}, x_1 = 0$ 是一个解，而且属于 Karmarkar 标准型，

$$\alpha = \frac{1}{2}, r = \frac{1}{\sqrt{n(n-1)}} = \frac{1}{\sqrt{6}}。$$

第一次迭代

(1) $k \leftarrow 0$，

$X^{(0)} = a_0 = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)^T$。

(2) $D=\begin{pmatrix}\frac{1}{3} & 0 & 0\\ 0 & \frac{1}{3} & 0\\ 0 & 0 & \frac{1}{3}\end{pmatrix}$,

$$AD=(0\quad +1\quad -1)\begin{pmatrix}\frac{1}{3} & 0 & 0\\ 0 & \frac{1}{3} & 0\\ 0 & 0 & \frac{1}{3}\end{pmatrix}=\left(0,\frac{1}{3},-\frac{1}{3}\right),$$

$$B=\begin{pmatrix}0 & -\frac{1}{3} & \frac{1}{3}\\ 1 & 1 & 1\end{pmatrix}$$

$$DC^T=\begin{pmatrix}1/3 & 0 & 0\\ 0 & 1/3 & 0\\ 0 & 0 & 1/3\end{pmatrix}\begin{pmatrix}-1\\ 0\\ 0\end{pmatrix}=\begin{pmatrix}-1/3\\ 0\\ 0\end{pmatrix}$$

(3) $BB^T=\begin{pmatrix}0 & \frac{1}{3} & -\frac{1}{3}\\ 1 & 1 & 1\end{pmatrix}\begin{pmatrix}0 & 1\\ 1/3 & 1\\ 1/3 & 1\end{pmatrix}=\begin{pmatrix}2/9 & 0\\ 0 & 3\end{pmatrix}$

$$(BB^T)^{-1}=\begin{pmatrix}9/2 & 0\\ 0 & 1/3\end{pmatrix}$$

$$C_P=[I-B^T(BB^T)^{-1}B]DC^T$$

$$=\left[\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}-\begin{pmatrix}0 & 1\\ 1/3 & 1\\ -1/3 & 1\end{pmatrix}\begin{pmatrix}9/2 & 0\\ 0 & 1/3\end{pmatrix}\begin{pmatrix}0 & 1/3 & 1/3\\ 1 & 1 & 1\end{pmatrix}\right]\begin{pmatrix}-1/3\\ 0\\ 0\end{pmatrix}$$

$$=\left[\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}-\begin{pmatrix}0 & 1\\ 1/3 & 1\\ -1/3 & 1\end{pmatrix}\begin{pmatrix}0 & \frac{3}{2} & -\frac{3}{2}\\ 1/3 & 1/3 & 1/3\end{pmatrix}\right]\begin{pmatrix}1/3\\ 0\\ 0\end{pmatrix}$$

$$=\left[\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}-\begin{pmatrix}\frac{1}{3} & \frac{1}{3} & \frac{1}{3}\\ \frac{1}{3} & 5/6 & \frac{5}{6}\\ \frac{1}{3} & -1/6 & 5/6\end{pmatrix}\right]\begin{pmatrix}-\frac{1}{3}\\ 0\\ 0\end{pmatrix}$$

$$=\begin{pmatrix}\frac{2}{3} & -\frac{1}{3} & -1/3\\ \frac{1}{3} & \frac{1}{6} & \frac{5}{6}\\ -\frac{1}{3} & +\frac{1}{6} & \frac{1}{6}\end{pmatrix}\begin{pmatrix}-1/3\\ 0\\ 0\end{pmatrix}=\begin{pmatrix}-2/9\\ \frac{1}{9}\\ \frac{1}{9}\end{pmatrix}$$

$$|C_p| = \frac{1}{6}\sqrt{6},$$

$$\bar{C}_p = \frac{1}{\sqrt{6}}\begin{pmatrix}2\\-1\\-1\end{pmatrix}$$

(4) $b=\left(\frac{1}{3}\ \ \frac{1}{3}\ \ \frac{1}{3}\right)^T-\alpha \bar{C}_p$，若取 $r=1/\sqrt{6}$，$\alpha=\frac{1}{4}$.

$$\therefore b=\left(\frac{1}{3}\ \ \frac{1}{3}\ \ \frac{1}{3}\right)^T-\frac{1}{21}(2\ \ -1\ \ -1)^T=\left(\frac{10}{21}\ \ \frac{7}{21}\ \ \frac{7}{21}\right)^T.$$

(5) $$Db=\frac{1}{21}\begin{pmatrix}1/3&0&0\\0&1/3&0\\0&0&1/3\end{pmatrix}\begin{pmatrix}10\\7\\7\end{pmatrix}=\frac{1}{72}\begin{pmatrix}10\\7\\7\end{pmatrix}$$

$$CDb=\frac{1}{72}\cdot 24=\frac{1}{3}$$

$$X^1=Db/CDb=\frac{1}{21}\begin{pmatrix}10\\7\\7\end{pmatrix}$$

继续这样的过程可得到解 $X=(1\ \ 0\ \ 0)^T$，由观察法不难看出最优解是 $x_1=1$，$x_2=x_3=0$.

值得一提的是若选择 $\alpha=\sqrt{6}/3$ 时，则

$$b=\left(\frac{1}{3}\ \ \frac{1}{3}\ \ \frac{1}{3}\right)^T+\frac{1}{3}(2\ \ -1\ \ -1)^T=(1\ \ 0\ \ 0)^T$$

$$Db=\begin{pmatrix}\frac{1}{3}&0&0\\0&\frac{1}{3}&0\\0&0&\frac{1}{3}\end{pmatrix}\begin{pmatrix}1\\0\\0\end{pmatrix}=\frac{1}{3}\begin{pmatrix}1\\0\\0\end{pmatrix}.$$

$$CDb=1$$

$$X=\begin{pmatrix}1\\0\\0\end{pmatrix}$$

现在再回过头来看 Karmarkar 算法的原理.

Karmarkar 标准型

$$\min CX$$
$$X\in \Omega\cap S \qquad (29.3.6)$$

其中 $\Omega=\{X\mid X\in R^n, AX=0\}$，$S=\{X\mid X\in R^n, \sum_{i=1}^{n}x_i=1, x_i\geq 0, i=1,2,\cdots,n\}$ 在 T 作用下

$$Y=T(X)=\frac{D^{-1}X}{e^TD^{-1}X}$$

或
$$y_i = \frac{1}{a_i}x_i \Big/ \left[\sum_{j=1}^{n} \frac{x_j}{a_j}\right] \qquad i = 1,2,\cdots,n.$$

$$\sum_{i=1}^{n} y_i = \sum_{i=1}^{n} \frac{1}{a_i} x_i \Big/ \left[\sum_{j=1}^{n} \frac{x_j}{a_j}\right] = 1.$$

$$\left(\sum_{j=1}^{n} \frac{x_j}{a_j}\right) a_i y_i = x_i \qquad i = 1,2,\cdots,n.$$

$$\because \sum_{i=1}^{n} x_i = 1$$

$$\therefore \sum_{j=1}^{n} \frac{x_j}{a_j} \cdot \sum_{i=1}^{n} a_i y_i = 1$$

$$x_i = a_i y_i \Big/ \left(\sum_{j=1}^{n} a_j y_j\right) \qquad i = 1,2,\cdots,n$$

$$即\ T^{-1}: X = \frac{DY}{e^T DY} = T^{-1}(Y)$$

将(29.3.6)变为

$$\min \frac{CDY}{e^T DY}$$
$$ADY = 0$$
$$e^T Y = 1$$
$$Y \geqslant 0$$

下面来对算法中若干符号公式进行必要说明，对理解算法的思想不无好处。

$$S = \{X \mid X \in R^n, \sum_{i=1}^{n} x_i = 1, x_i \geqslant 0, i = 1,2,\cdots,n\}$$

从图 29.3.1 上可看出：

$n=2$ 时 S 域实际上是两端点为 $(0,1)$，$(1,0)$ 的线段，$(\frac{1}{2},\frac{1}{2})$ 是线段中点。

$n=3$ 时 S 域为以 $(0,0,1)$，$(0,1,0)$ 和 $(1,0,0)$ 三个顶点为顶点的三角形域，$(\frac{1}{3},\frac{1}{3},\frac{1}{3})$ 是三角形的中心。依此类推，在 R^n 中 S 域有 n 个顶点，$C(n,2)$ 条棱，$C(n,n-1)$ 个面的 $\frac{1}{n}(1,1,\cdots,1)$ 是它的中心。

可以证明包含外接于 S 域的球的半径为

$$R = \frac{\sqrt{n-1}}{\sqrt{n}}$$

以 $n=3$ 为例，R 即为从 $(\frac{1}{3},\frac{1}{3},\frac{1}{3})$ 点到 $(1,0,0)$ 点或到 $(0,1,0)$，$(0,0,1)$ 点的距离。

内切于 S 的球的半径为 $\frac{1}{\sqrt{n(n-1)}}$，$n=3$ 时即为边长为 $\sqrt{2}$ 的等边三角形高的 $\frac{1}{3}$。

设 $a=(a_1,a_2,\cdots,a_n)$ 是 S 一内点，则

$$\sum_{i=1}^{n}\alpha_i = 1,\ \alpha_i > 0,\ i = 1,2,\cdots,n$$

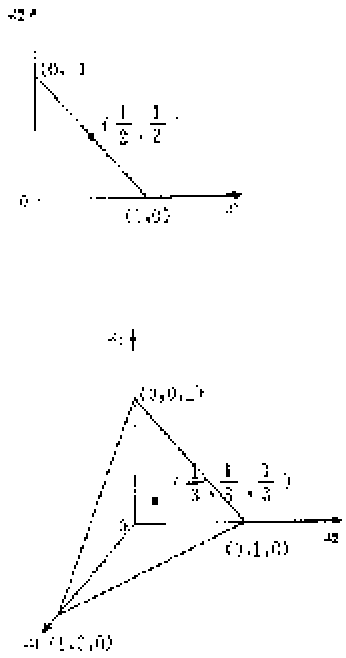


图 20.3.1

$$D = \mathrm{Diag}(\alpha_1,\alpha_2,\cdots,\alpha_n) = \begin{bmatrix} \alpha_1 & 0 & \cdots & 0 \\ 0 & \alpha_2 & & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \alpha_n \end{bmatrix}$$

$$D^{-1} = \mathrm{Diag}(\frac{1}{\alpha_1},\frac{1}{\alpha_2},\cdots,\frac{1}{\alpha_n})$$

$$T(x) = \frac{D^{-1}X}{eD^{-1}X} \quad \forall X \in S$$

所以$T(x)$是一向量，它的各分量经过其和等于1的规范化，则$T(X)$是由S到S的一个映射。

例如 $\alpha = \left(\frac{3}{10},\frac{1}{10},\frac{6}{10}\right)$

α是S的一个内点

$$D = \begin{bmatrix} \frac{3}{10} & 0 & 0 \\ 0 & \frac{1}{10} & 0 \\ 0 & 0 & \frac{6}{10} \end{bmatrix}$$

$$D^{-1} = \begin{bmatrix} \frac{10}{3} & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10/6 \end{bmatrix}$$

在T的变换下，有α点的象

$$T(\alpha) = \frac{1}{3}\begin{bmatrix} \frac{10}{3} & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & \frac{10}{6} \end{bmatrix}\begin{bmatrix} \frac{3}{10} \\ \frac{1}{10} \\ \frac{6}{10} \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

即将α变为S的中心。

不难验证S的三个顶点(1,0,0)、(0,1,0)、(0,0,1)在T的作用下，设$a=(1,0,0)^T$，$b=(0,1,0)^T$，$c=(0,0,1)^T$

$$T(a) = \frac{3}{10}\begin{bmatrix} 10/3 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 5/3 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \frac{3}{10}\begin{bmatrix} \frac{10}{3} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$T(b)=\frac{1}{10}\begin{bmatrix}\frac{10}{3}&0&0\\0&10&0\\0&0&[illegible]\end{bmatrix}\begin{bmatrix}3\\1\\0\end{bmatrix}=\begin{bmatrix}0\\1\\0\end{bmatrix}$$

同样可知道 $T(c)=\begin{bmatrix}0\\0\\1\end{bmatrix}$.

一般说 T 将 S 的边界点变为 S 的边界点，S 的顶点变为 S 的顶点，S 的内点的象也是 S 的内点。

在 T 变换作用下 Karmarkar 标准型变换如下：

$$\begin{aligned}&\min CX\\&AX=0\\&e^TX=1\\&X\geqslant 0\end{aligned}\quad\Rightarrow\quad\begin{aligned}&\min\frac{CDY}{e^TDY}\\&ADY=0\\&e^TY=1\\&Y\geqslant 0\end{aligned}$$

令 $B=\begin{pmatrix}AD\\e^T\end{pmatrix}$，$d$ 是满足 $Bd=0$ 的方向. Karmarkar 算法的思想就在于在满足 $Bd=0$ 的空间里寻找使目标函数下降的方向，即在不影响可行解的空间里寻找使目标函数下降的方向，得到的是使目标函数下降的可行解. 还要引进几个必要的概念.

$$L=\{u\mid u=A^Tx,\ x\text{ 是任意向量}\}$$

$$L^{\perp}=\{v\mid Av=0\}$$

其中 $A=(a_{ij})_{m\times n}$，$r(A)=m$，则 AA^T 可逆，L 和 $L^{\perp}$ 是正交互补子空间，对任一向量 x，必存在 $P=A^Tu$，$R\in L^{\perp}$，使得

$$X=P+R$$

$$AX=AA^Tu+AR=AA^Tu$$

∴ $u=(AA^T)^{-1}AX,\ P=A^Tu=A^T(AA^T)^{-1}AX$

又 $R=X-P$

∴ $R=[I-A^T(AA^T)^{-1}A]X$

定义 $I-A^T(AA^T)^{-1}A$ 为正交投影矩阵。

经过 T 的变换目标函数虽为分式，已不是线性函数，但可取分子 CDY 作为目标下降的指标，故取它的负梯度 $-DC^T$ 作为考虑方向，为了保持它的可行性，将它投影到约束矩阵的零空间。

故得

$$d=-[I-B^T(BB^T)^{-1}B]DC^T$$

现在已对应出 Karmarkar 算法，它的几何意义便一目了然了.

Karmarkar 算法的收敛性和复杂性从略.

Karmarkar 算法实际怎样呢？编者作过一些尝试，在一般维数比较低的情况，很不理想。这和单纯形法是对手，但作者报告在超大型问题时，较单纯形法有优势，所以还需要进一步看实践。理论分析上说这种算法好，实践也是这样，应该说是值得探索的途径，特别是近年

那一件大事。

习　　题

1. 已知 $a_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$，$B = \begin{pmatrix} 16 & 0 \\ 0 & 9 \end{pmatrix}$，
不等式为 $x_1 + x_2 \leqslant 1$。试用椭球切法作一次迭代，并绘出 E_1 和 E_2 进行比较。

2. 试证 Karmarkar 算法中变换 T

$$Y = T(X) = \frac{D^{-1}X}{e^TD^{-1}X}$$

有以下性质：

(1) T 将

$$S = \{X | e^TX = 1, x_j \geqslant 0, j = 1, 2, \cdots, n\}$$

变为 S，并求 T 的逆变换。

(2) T 将 a 变为 S 的中心 $\frac{1}{n}e$。

(3) 在 T 变换下 S 的顶点不变。

(4) 在 S 上，

$$\{X | AX = 0\} \xrightarrow{T} \{Y | ADY = 0\}.$$

3. 试对 28.3 中举的例子，再用 Karmarkar 算法迭代一次，试找它是否是最优解。

4. 试用 Karmarkar 算法解下列问题

$$\min z = -x_1$$
$$x_1 - x_2 = 0$$
$$x_1 + x_2 + x_3 = 1$$
$$x_1, x_2, x_3 \geqslant 0$$