

## 冗繁削尽留清瘦

### ——浅谈信息的充分利用

长沙市雅礼中学 张一飞

#### 【摘要】

在算法设计中，人们往往不自觉地进行了大量多余的运算，这些累赘将大大降低算法的效率。作者认为，充分利用已知信息，是解决这一问题的一个有效方法。

所谓充分利用信息，就是在算法设计中，把已知信息尽可能充分地利用起来，以避免冗余运算，降低算法的时空复杂度，从而提高算法的效率。本文对充分利用信息，在优化回溯法、动态规划和数值计算中的应用作了初步的探讨。

#### 【关键字】

信息，算法优化

“冗繁削尽留清瘦”<sup>[1]</sup>虽然讲的是画竹，却包含着深刻的哲理。算法设计同画竹一样，也需要削尽冗繁。但在解题实践中，人们往往不自觉地做了一些多余的运算，而忽视了对已知信息的充分利用。

所谓充分利用信息，就是在算法设计中，把已知信息尽可能充分地利用起来，以避免冗余运算，降低算法的时空复杂度，从而提高算法的效率。限于篇幅，本文仅对这种方法提高回溯法、动态规划和数值计算的效率进行探讨。

### 1、提高回溯法的效率

我们知道，回溯法实质上是从树根出发，遍历一棵解答树的过程。如果解答树中存在一些性质相同的子树，那么，只要我们知道了其中一棵子树的性质，就可以根据这个信息，导出其它子树的性质。这就是自顶向下记忆化搜索<sup>[2]</sup>的基本思想。

记忆化搜索避免了一些多余的运算，因而比非记忆化搜索效率要高。但在有些记忆化搜索中，对信息的利用仍不够充分，还有进一步优化的余地。

下面，我们看一个例子：

#### 【序关系计数问题】

用关系‘<’和‘=’将3个数A、B和C依次排列有13种不同的关系：

$A < B < C, A < B = C, A < C < B, A = B < C, A = B = C, A = C < B,$

$B < A < C, B < A = C, B < C < A, B = C < A,$

$C < A < B, C < A = B, C < B < A.$

编程求出N个数依序排列时有多少种关系。

#### <1>. 枚举出所有的序关系表达式

我们可以采用回溯法枚举出所有的序关系表达式。 $N$ 个数的序关系表达式，是通过 $N$ 个大写字母和连接各字母的 $N-1$ 个关系符号构成。依次枚举每个位置上的大写字母和关系符号，直到确定一个序关系表达式为止。

由于类似于‘ $A=B$ ’和‘ $B=A$ ’的序关系表达式是等价的，为此，规定等号前面的大写字母在ASCII表中的序号，必须比等号后面的字母序号小。基于这个思想，我们很容易写出解这道题目的回溯算法。

算法 1-1，计算 $N$ 个数的序关系系数。

```

procedure Count(Step,First,Can);
{Step 表示当前确定第 Step 个大写字母;
 First 表示当前大写字母可能取到的最小值;
 Can 是一个集合，集合中的元素是还可以使用的大写字母}
begin
  if Step=N then begin{确定最后一个字母}
    for i:=First to N do if i in Can then Inc(Total); {Total 为统计的结果}
    Exit
  end;
  for i:=First to N do{枚举当前的大写字母}
    if i in Can then begin{i 可以使用}
      Count(Step+1,i+1,Can-[i]);{添等于号}
      Count(Step+1,1,Can-[i]);{添小于号}
    end
  end;
end;

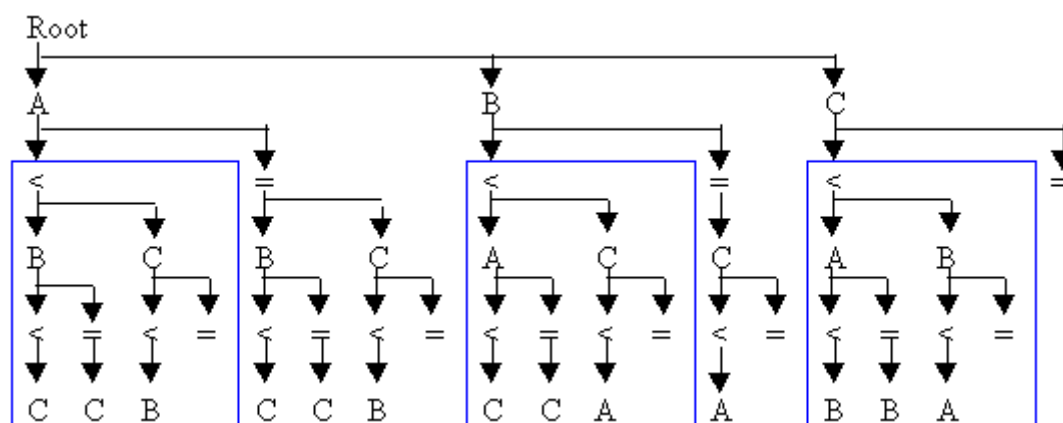
```

调用 Count(1,1,[1..N])后，Total 的值就是结果。该算法的时间复杂度是  $O(N!)$

## <2>. 粗略利用信息，优化算法 1-1

算法 1-1 中存在大量冗余运算。如图 1，三个方框内子树的形态完全一样。一旦我们知道了其中某一个方框内所产生的序关系系数，就可以利用这个信息，直接得到另两个方框内将要产生的序关系系数。

图 1  $N=3$  时的解答树



显然，在枚举的过程中，若已经确定了前 $k$ 个数，并且下一个关系符号是小于号，这时所能产生的序关系系数就是剩下的 $N-k$ 个数所能产生的序关系系数。

设 $i$ 个数共有 $F[i]$ 种不同的序关系，那么，由上面的讨论可知，在算法 1-1 中，调用一次 Count(Step+1,1,Can-[i])之后，Total 的增量应该是  $F[N-Step]$ 。这个

值可以在第一次调用  $\text{Count}(\text{Step}+1, 1, \text{Can}-[i])$  时求出。而一旦知道了  $F[\text{N-Step}]$  的值，就可以用  $\text{Total} := \text{Total} + F[\text{N-Step}]$  代替调用  $\text{Count}(\text{Step}+1, 1, \text{Can}-[i])$ 。这样，我们可以得到改进后的算法 1-2。

算法 1-2，计算  $N$  个数的序关系数。

*procedure Count(Step, First, Can);*

*{Step, First, Can 的含义同算法 1-1}*

*begin*

*if Step=N then begin {确定最后一个字母}*

*for i:=First to N do if i in Can then Inc(Total); {Total 为统计的结果}*

*Exit*

*end;*

*for i:=First to N do {枚举当前的大写字母}*

*if i in Can then begin {i 可以使用}*

*Count(Step+1, i+1, Can-[i]); {添等于号}*

*if F[N-Step]=-1 then begin {第一次调用}*

*F[N-Step]:=Total;*

*Count(Step+1, 1, Can-[i]); {添小于号}*

*F[N-Step]:=Total-F[N-Step] {F[N-Step]=Total 的增量}*

*end else Total:=Total+F[N-Step] {F[N-Step] 已经求出}*

*end*

*end;*

开始，将  $F[0], F[1], \dots, F[N-1]$  初始化为 -1

调用  $\text{Count}(1, 1, [1..N])$  之后，Total 的值就是结果

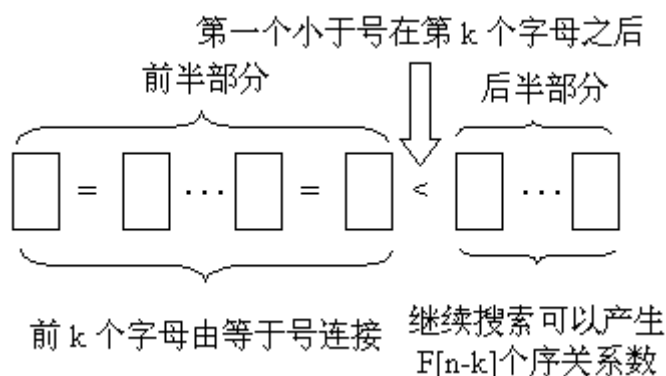
算法 1-2 与算法 1-1 的差别仅限于程序中的粗体部分。

算法 1-2 就是利用了  $F[0], F[1], \dots, F[N-1]$  的值，使得在确定添小于号以后，能够避免多余的搜索，尽快地求出所需要的方案数。该算法实质上就是自顶向下记忆化方式的搜索，它的时间复杂度为  $O(2^N)^{[3]}$ 。同算法 1-1 相比，效率虽然有所提高，但仍不够理想。

### <3>. 充分利用信息，进一步优化算法 1-2

在搜索的过程中，如果确定在第  $k$  个大写字母之后添加第一个小于号，则可得下面两条信息：

图 2 充分利用信息，进一步优化算法 1-2



第一条信息：前  $k$  个大写字母都是用等号连接的。

第二条信息：在此基础上继续搜索，将产生  $F[N-k]$  个序关系表达式。

如图 2 所示，序关系表达式中第一个小于号将整个表达式分成了两个部分。由乘法原理易知，图 2 所示的序关系表达式的总数，就是图中前半部分所能产生的序关系数，乘以后半部分所能产生的序关系数。算法 1-2 实质上利用了第二条信息，直接得到图中后半部分将产生  $F[n-k]$  个序关系数，并通过搜索得到前半部分将产生的序关系数。但如果我们利用第一条信息，就可以推知图中前半部分将产生的序关系数，就是  $N$  个物体中取  $k$  个的组合数，即  $C_n^k$ 。这样，我们可以得到  $F[n]$  的递推关系式：

$$\text{公式1: } F[n] = \sum_{k=1}^n C_n^k F[n-k], \text{ 其中 } F[0]=1$$

采用公式 1 计算  $F[n]$  的算法记为算法 1-3<sup>[4]</sup>，它的时间复杂度是  $O(N^2)$ 。

#### <4>. 小结

下面是三个算法的性能分析表<sup>[5]</sup>：

分析项目		算法 1-1	算法 1-2	算法 1-3
理论分析	时间复杂度	$O(N!)$	$O(2^N)$	$O(N^2)$
	空间复杂度	$O(1)$	$O(N)$	$O(N)$
实际运行情况	N=7	1s	<0.05s	<0.05s
	N=8	10s	<0.05s	<0.05s
	N=15	>10s	0.5s	<0.05s
	N=17	>10s	2s	<0.05s

在优化算法 1-1 的过程中，我们通过利用  $F[0], F[1], \dots, F[N-1]$  的信息，得到算法 1-2，时间复杂度也从  $O(N!)$  降到  $O(2^N)$ 。在算法 1-2 中，进一步消除冗余运算，就得到了  $O(N^2)$  的算法 1-3。

算法 1-3 计算  $F[n]$ ，体现了动态规划的思想。也就是说，我们通过充分利用信息，提高回溯法的效率，实质上是将搜索转化成了动态规划。

## 2、提高动态规划的效率

从上一节中可以看到，动态规划之所以高效，就在于它比较充分的利用了已知信息。但是，在有些动态规划中，仍存在冗余运算。这一节我们将进一步探讨如何充分利用信息，提高动态规划的效率。

下面我们看一个例子：

### 【理想收入问题】

理想收入是指在股票交易中，以 1 元为本金可能获得的最高收入，并且在理想收入中允许有非整数股票买卖。

已知股票在第  $i$  天每股价格是  $V[i]$  元， $1 \leq i \leq M$ ，求  $M$  天后的理想收入。

#### <1>. 一种动态规划的解法

解这道题目，很容易想到用动态规划。设  $F[i]$  表示在第  $i$  天收盘时能达到的最高收入，则有  $F[i]$  的递推关系式：

$$\text{公式2: } F[i] = \max_{(0 \leq j < i)} \{F[j] / V[k] * V[i]\}, \text{ 其中 } F[0] = 1, V[0] = 1$$

公式 2 的含义是：在第  $i$  天收盘时能达到的最高的收入，是将第  $j$  天收盘后

的收入，全部用于买入第  $k$  天的股票，再在第  $i$  天将所持的股票全部卖出所得的收入。采用公式 2，可以得到算法 2-1，其时间复杂度是  $O(M^3)$ ，空间复杂度是  $O(M)$ 。

算法 2-1

$F[0]:=1; V[0]:=1; F[1..M]:=0;$

for  $i:=1$  to  $M$  do

  for  $j:=0$  to  $i-1$  do

    for  $k:=j$  to  $i-1$  do

$F[i]:=Max\{F[i], F[j]/V[k]*V[i]\}$

### <2>. 改变状态表示的含义，优化算法 2-1

改变动态规划中状态表示的含义，是优化动态规划的常用方法。例如此题，我们可以采用两种不同的状态表示方法优化算法 2-1。

方法 1：设  $P[i]$  表示前  $i$  天能获得的最多股票数，可列出如下状态转移方程：

$$\text{公式3: } P[i] = \max_{(0 \leq j < i)} \{P[i-1], P[j] * V[j] / V[i]\}$$

这是因为前  $i$  天所能获得的最多股票数，或者是前  $i-1$  天获得的最多股票数，或者是在第  $j$  天将前  $j$  天所能获得的最多的股票全部卖出，再买入第  $i$  天的股票。显然，前  $i-1$  天能获得的最多股票数乘以第  $i$  天的股价，就是第  $i$  天能达到的最大收入。

方法 2：设  $Q[i]$  表示前  $i$  天能达到的最大收入，可列出如下状态转移方程：

$$\text{公式4: } Q[i] = \max_{(0 \leq j < i)} \{Q[i-1], Q[j] / V[j] * V[i]\}$$

就是说前  $i$  天所能达到的最大收入，或者是前  $i-1$  天所能达到的最大收入，或者是在第  $j$  天买入股票，再在第  $i$  天卖出，所能获得的最大收入。

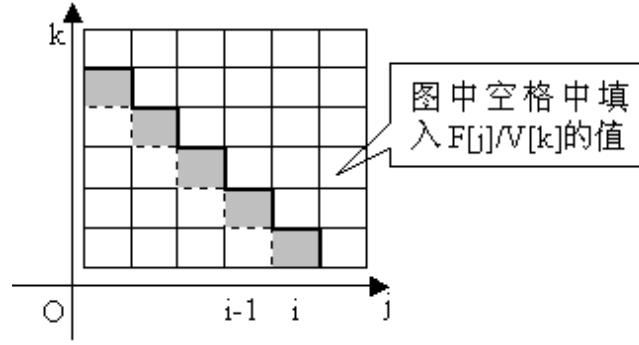
上述两种方法的时间复杂度都是  $O(M^2)$ 。这表明，改变状态表示的含义，在一定程度上提高了算法的效率。但对于这道题目，仅仅改变状态表示的含义，很难进一步优化算法。

### <3>. 粗略利用信息，优化算法 2-1

算法 2-1 粗体部分的功能是确定  $F[i]$  所能达到的最大值。由于  $V[i]$  不变，因此  $F[i]$  达到最大值，当且仅当  $F[j]/V[k]$  达到最大值，其中  $0 \leq j \leq k < i$ 。

算法 2-1 中，采用了二重循环来确定  $F[j]/V[k]$  的最大值。但在确定  $F[i-1]$  所能达到最大值的时候，我们实际上已经求出当  $0 \leq j \leq k < i-1$  时， $F[j]/V[k]$  所能达到的最大值。如果能充分利用这一信息，就可以更快地确定  $F[i]$  所能达到的最大值。如图 3 所示，要确定粗线下部的最大值，只需比较虚线下部的最大值和灰色部分的最大值即可。

图 3 初步利用信息，优化算法 2-1



为了表示出图 3 的思想，

设  $MaxFV[i] = \max_{(0 \leq j \leq k < i)} \{F[j]/V[k]\}$ ，则

$$\begin{aligned}
 MaxFV[i] &= \max_{(0 \leq j \leq k < i)} \{F[j]/V[k]\} \\
 &= \max \{ \max_{(0 \leq j \leq k < i-1)} \{F[j]/V[k]\}, \max_{(0 \leq j \leq k, k=i-1)} \{F[j]/V[k]\} \} \\
 &= \max \{ MaxFV[i-1], \max_{(0 \leq j \leq i-1)} \{F[j]/V[i-1]\} \} \\
 &= \max_{(0 \leq j < i)} \{ MaxFV[i-1], F[j]/V[i-1] \}
 \end{aligned}$$

$$\begin{aligned}
 \text{由公式 2, } F[i] &= \max_{(0 \leq j \leq k < i)} \{F[j]/V[k] * V[i]\} \\
 &= \max_{(0 \leq j \leq k < i)} \{F[j]/V[k]\} * V[i] \\
 &= MaxFV[i] * V[i]
 \end{aligned}$$

这样，我们得到如下递推关系式：

公式 5:

$$\begin{aligned}
 F[i] &= MaxFV[i] * V[i] \\
 MaxFV[i] &= \max_{(0 \leq j < i)} \{ MaxFV[i-1], F[j]/V[i-1] \}
 \end{aligned}$$

从公式 5 中可以看出，在确定  $MaxFV[i]$  时，较充分的利用了确定  $MaxFV[i-1]$  时产生的结果。采用公式 5 可得算法 2-2，它的时间复杂度为  $O(M^2)$ ，空间复杂度是  $O(M)$ 。

算法 2-2

```

F[0]:=1;MaxFV[0]:=0;V[0]:=1;
for i:=1 to M do begin
  MaxFV[i]:=MaxFV[i-1];
  for j:=0 to i-1 do
    MaxFV[i]:=Max{MaxFV[i],F[j]/V[i-1]}
  F[i]:=MaxFV[i]*V[i]
end;
```

将公式 5 化简，有

$$\begin{aligned}
 MaxFV[i] &= \max_{(0 \leq j < i)} \{ MaxFV[i-1], F[j]/V[i-1] \} \\
 &= \max_{(0 \leq j < i)} \{ MaxFV[i-1], MaxFV[j] * V[j]/V[i-1] \}
 \end{aligned}$$

在这个公式中， $MaxFV[i]$  可以看作是前  $i-1$  天所能获得的最多股票数。这种状态表示方法和公式 3 中  $P[i]$  的含义本质上是相同的。这样，我们通过对已知信息的利用，达到了改变动态规划中状态表示含义的效果。

#### <4>. 充分利用信息，进一步优化算法 2-2

在算法 2-2 中，进一步利用信息，很容易得到时间复杂度为  $O(M)$  的算法。

算法 2-2 的粗体部分的功能是确定  $MaxFV[i]$  所能达到的最大值。由于  $V[i-1]$

不变, 因此  $F[j]/V[i-1]$  达到最大值, 当且仅当  $F[j]$  达到最大值, 其中  $0 \leq j < i$ 。

算法 2-2 中内层循环实质上是在确定  $\text{MaxFV}[i]$  时, 找到  $F[0], F[1], \dots, F[i-1]$  中的最大值。而在确定  $\text{MaxFV}[i-1]$  时, 我们已经找到了  $F[0], F[1], \dots, F[i-2]$  中的最大值。如果把这个信息利用起来, 就可以更快的确定  $F[0], F[1], \dots, F[i-1]$  中的最大值。

设  $\text{MaxF}[i] = \max_{(0 \leq j < i)} \{F[j]\}$ , 则

$$\begin{aligned}\text{MaxF}[i] &= \max_{(0 \leq j < i)} \{F[j]\} \\ &= \max \{ \max_{(0 \leq j < i-1)} \{F[j]\}, F[i-1] \} \\ &= \max \{ \text{MaxF}[i-1], F[i-1] \} \\ \text{MaxFV}[i] &= \max_{(0 \leq j < i)} \{ \text{MaxFV}[i-1], F[j]/V[i-1] \} \\ &= \max \{ \text{MaxFV}[i-1], \text{MaxF}[i]/V[i-1] \}\end{aligned}$$

这样, 我们得到如下递推关系式:

公式 6:

$$\begin{aligned}F[i] &= \text{MaxFV}[i] * V[i] \\ \text{MaxFV}[i] &= \max \{ \text{MaxFV}[i-1], \text{MaxF}[i]/V[i-1] \} \\ \text{MaxF}[i] &= \max \{ \text{MaxF}[i-1], F[i-1] \}\end{aligned}$$

从公式 6 中可以看出, 在确定  $\text{MaxF}[i]$  时, 充分利用了确定  $\text{MaxF}[i-1]$  时所产生的信息。采用公式 6 可得算法 2-3, 它的时间复杂度是  $O(M)$ 。从公式 6 中的三个递推关系式可以看出, 当前状态都只与前一个状态有关, 因此, 空间复杂度可以进一步降到  $O(1)$ <sup>[6]</sup>。

算法 2-3

```
F:=1;MaxF:=0;MaxFV:=0;V[0]:=1;
for i:=1 to M do begin
  if F>MaxF then MaxF:=F;
  if MaxF/V[i-1]>MaxFV then MaxFV:=MaxF/V[i-1];
  F:=MaxFV*V[i]
end;
```

公式 6 中  $\text{MaxF}[i]$  可以看作是前  $i-1$  天能达到的最大收入<sup>[7]</sup>。虽然这种状态表示方法和公式 4 中  $Q[i]$  是类似的, 但算法的时间复杂度却从  $O(M^2)$  降到了  $O(M)$ , 空间复杂度也从  $O(M)$  降到了  $O(1)$ 。这样, 我们通过充分利用已知信息, 达到了改变状态表示难以达到的优化效果。

## <5>. 小结

下面是三个算法的性能分析表:

分析项目		算法 2-1	算法 2-2	算法 2-3
理论分析	时间复杂度	$O(M^3)$	$O(M^2)$	$O(M)$
	空间复杂度	$O(M)$	$O(M)$	$O(1)$
实际运行情况	M=200 的随机数据	4s	0.05s	<0.05s
	M=1000 的随机数据	>60s	1s	<0.05s
	M=2000 的随机数据	>60s	5s	<0.05s

在这道题中, 我们通过避免冗余运算, 将时间复杂度由  $O(M^3)$  先降到  $O(M^2)$ , 再进一步降到  $O(M)$ 。空间复杂度也从  $O(M)$  降到了  $O(1)$ 。

由此可见, 充分利用信息, 提高动态规划的效率, 是非常有效的。此外, 充分利用信息并不一定要以牺牲空间为代价, 同样可以优化算法的空间复杂度。

### 3、提高数值计算的效率

从前面的分析中，我们深深地体会到了动态规划的核心思想——充分利用已知信息。但这个思想并不仅限于动态规划。下面我们将看到，充分利用已知信息，对提高数值计算的效率，也十分有效。

我们看一道数值计算题：

**【高精度计算问题】** (湖南 1998 省赛试题)

输入  $n, m, k$ ，计算  $S_m(n)$  的后  $k$  位数。其中  $S_m(n) = 1^m + 2^m + \dots + n^m$ ， $1 \leq k \leq 99$ ， $1 \leq n, m \leq 9999$ 。

#### <1>. 分析

由于  $k$  可以达到 99，因此必须采用高精度计算。本题只要求出  $S_m(n)$  的后  $k$  位数，所以，每次计算只取结果的后  $k$  位数即可。显然，计算  $S_m(n)$  将耗费大量的时间用于乘幂运算。下面，我们分析乘幂运算的算法。

#### <2>. 朴素的乘幂算法

根据乘幂的定义，我们将  $i$  自乘  $m$  次即可。

算法 3-1，计算  $i^m$

Func Power( $i, m$ );

begin

SetValue( $x, 1$ ); { $x$  是高精度数}

for  $k := 1$  to  $m$  do

$x := \text{Mul}(x, i)$ ; {Mul 是高精度乘法，返回  $x * y$  的值}

Power :=  $x$

end;

采用算法 3-1 计算  $i^m$ ，需要进行  $M$  次高精度乘法。

#### <3>. 粗略利用信息，优化算法 3-1

算法 3-1 对信息的利用很不充分。例如，要计算  $i^5$ ，现在已经求出了  $i^3$  的值，算法 3-1 会利用  $i^3$  和  $i$  的值，求出  $i^4$ ，进而求出  $i^5$ 。而实际上，在计算  $i^3$  之前，我们已经求出了  $i^2$  的值，将  $i^3$  乘上  $i^2$ ，就可以求出  $i^5$ 。

我们用数组  $p[k]$  保存已经计算出的  $i^k$  的结果，在计算过程中，尽可能的使用已经计算过的信息。例如，我们可以这样计算  $i^{15}$ ：

$p[1] = i$

$p[2] = p[1] * p[1]$

$p[3] = p[2] * p[1]$

$p[6] = p[3] * p[3]$

$p[7] = p[6] * p[1]$

$p[14] = p[7] * p[7]$

$p[15] = p[14] * p[1]$

这样，我们只用了 6 次高精度乘法，就求出了  $i^{15}$ 。

从上面的例子可以看出，将一个已知结果平方，就可以只用一次乘法，求出一个比较大的指数幂，根据这个思想就可得到算法 3-2：



$$i^m = \begin{cases} (i^{\frac{m}{2}})^2 & m \text{为偶数} \\ i(i^{\frac{m-1}{2}})^2 & m \text{为奇数} \end{cases}$$

算法 3-2， 计算  $i^m$

```
Func Power(i,m);
begin
  if m=1 then Power:=i
  else begin
    Temp:=Power(i,m div 2);
    Temp:=Mul(Temp,Temp);
    if Odd(m) then Power:=Mul(Temp,i)
    else Power:=Temp
  end
end;
```

算法 3-2 计算  $i^m$  需要  $O(\log_2 m)$  次高精度乘法。这样，我们通过对信息的粗略利用<sup>[8]</sup>，将时间复杂度从  $O(m)$  降到了  $O(\log_2 m)$ 。

算法 3-2 实质上就是二分法。采用二分法求乘幂，之所以比累乘的方法高效，就在于它更加充分的利用了已知信息。

<4>. 充分利用信息，进一步优化算法 3-2

注意到在计算  $i^m$  时，我们已经知道了  $1^m, 2^m, \dots, (i-1)^m$  的值，而这些值在算法 3-2 计算  $i^m$  时没有起任何作用。如果能够建立  $i^m$  与  $1^m, 2^m, \dots, (i-1)^m$  的关系，就可更快计算  $i^m$ 。

例如，求  $120^m$ 。这时，我们已经计算了  $5^m$  和  $24^m$  的值。显然， $5^m$  和  $24^m$  相乘，就可以得到  $120^m$ 。

当  $i$  是合数时，设  $i=pq$ ，其中  $1 < p, q < i$ 。显然， $i^m = (pq)^m = p^m q^m$ 。由于  $p, q < i$ ，因此在计算  $i^m$  时， $p^m$  和  $q^m$  都是可以直接利用的信息。这样，对于合数  $i$ ，我们只需要一次高精度乘法，就可以求出  $i^m$ 。

采用这种方法得到算法记为算法 3-3。设  $1..n$  中，有  $x$  个合数，则算法 3-3 只做了  $(n-x)\log_2 m + x$  次高精度乘法。自然数中素数的分布十分稀疏，实践证明，当  $n=m=9999, k=99$  时，算法 3-3 的效率是算法 3-2 的 5 倍。

<5>. 小结

下面是三个算法的性能分析表：

分析项目		算法 3-1	算法 3-2	算法 3-3
理论分析	时间复杂度	$O(nm)$	$O(n\log_2 m)$	$O(n\log_2 m)$
	空间复杂度	$O(k)$	$O(k)$	$O(nk)$
实际运行情况	$N=M=100, K=99$	0.1	<0.05	<0.05
	$N=100, M=9999, K=99$	20s	0.5s	0.1s
	$N=M=1000, K=99$	15s	3s	1s
	$N=M=9999, K=99$	>60s	55s	10s

在这个例子中，我们首先通过对信息的粗略利用，优化朴素的乘幂算法 3-1，得到采用二分法的算法 3-2。进一步分析发现，二分法在计算  $i^m$  时，没有利用已经求出的  $1^m, 2^m, \dots, (i-1)^m$  等信息，充分利用这些信息，得到了更快的算法

3-3。由于算法 3-3 需要保存一些已经求过的值<sup>[9]</sup>，所以，该算法实质上是以空间为代价换取了时间。

#### 4、结束语

在搜索中加入记忆化信息提高回溯法的效率，改变动态规划中状态表示的含义提高动态规划的效率，采用二分法提高数值计算的效率，都不同程度的体现了对已知信息的充分利用。但仅使用这些常用技巧优化算法，仍可能存在多余的运算。如果能够更加充分的利用已知信息，就可以收到更好的效果。

在算法中削去冗繁，关键在于找到可利用的已知信息，一旦把它们充分利用起来，就可以大大提高算法效率。

#### 【参考文献】

1. 《信息学奥林匹克》. 1999(3)
2. 吴文虎，王健德 .《实用算法与程序设计》. 电子工业出版社 . 1998.01
3. 刘福生，王建德 .《青少年国际信息学（计算机）奥林匹克竞赛指导——人工智能搜索与程序设计》. 电子工业出版社 . 1993.04
4. 《郑板桥集》

## 【附录】

[1] 摘自《郑板桥集》第206页，全诗如下：

题画竹  
四十年来画竹枝  
日间挥写夜间思  
冗繁削尽留清瘦  
画到生时是熟时

[2] 见参考文献[2]。

[3] 在后面的分析中可以看到，调用粗体部分程序段的总次数是

$$\sum_{i=0}^n \sum_{j=0}^i C_i^j = \sum_{i=0}^n 2^i = 2^{n+1}$$

故算法的时间复杂度为  $O(2^{n+1})=O(2^n)$ 。

[4] 也可以直接采用数学方法推导公式1。但我认为，这比本文中使用的更难掌握。在参考文献[1]中，给出了一个形式上更简单的计算公式，但这个公式的设计也颇有难度。参考文献[1]中给出的公式是：

$$F(m, t) = [F(m-1, t-1) + F(m-1, t)] \times t$$

$$\text{边界条件: } F(1, 1) = 1, F(1, t) = 0 (t > 0)$$

$$n \text{ 个数的序关系系数是: } \sum_{i=1}^n F(n, i)$$

[5] 本文中所有程序的运行环境均为 Pentium 100MHz，BP7.0 编译。

[6] 如果我们边读数据边规划，就只要保存  $V[i-1]$  和  $V[i]$ ，而没有必要定义长度为  $M$  的数组  $V$ 。这样，算法的空间复杂度降到了  $O(1)$ 。

[7] 公式6可进一步化简：

$$\text{MaxF}[i] = \max \{ \text{MaxF}[i-1], F[i-1] \}$$

$$= \max \{ \text{MaxF}[i-1], \text{MaxFV}[i-1] * V[i-1] \}$$

$$\text{MaxF}[i] / V[i-1] = \max \{ \text{MaxF}[i-1], \text{MaxFV}[i-1] * V[i-1] \} / V[i-1]$$

$$= \max \{ \text{MaxF}[i-1] / V[i-1], \text{MaxFV}[i-1] * V[i-1] / V[i-1] \}$$

$$= \max \{ \text{MaxF}[i-1] / V[i-1], \text{MaxFV}[i-1] \}$$

$$\geq \text{MaxFV}[i-1]$$

$$\text{MaxFV}[i] = \max \{ \text{MaxFV}[i-1], \text{MaxF}[i] / V[i-1] \}$$

$$= \text{MaxF}[i] / V[i-1]$$

$$\text{MaxF}[i] = \max \{ \text{MaxF}[i-1], \text{MaxFV}[i-1] * V[i-1] \}$$

$$= \max \{ \text{MaxF}[i-1], \text{MaxF}[i-1] / V[i-2] * V[i-1] \}$$

这样可以得到如下递推关系式：

$$\text{MaxF}[i] = \max \{ \text{MaxF}[i-1], \text{MaxF}[i-1] / V[i-2] * V[i-1] \}$$

在源程序 Pro\_2.pas 中，给出了这个公式的算法2-4。算法2-4的时空复杂度

同算法 2-3，只是形式上更加简单。

<sup>[8]</sup> 值得说明的是，采用二分法求乘幂，并不是最充分利用已知信息的方法。例如，采用二分法计算  $i^{15}$  需要 6 次乘法，而实际上，我们只需 5 次乘法即可。

```
p[1]=i;  
p[2]=p[1]*p[1]  
p[3]=p[2]*p[1]  
p[6]=p[3]*p[3]  
p[9]=p[6]*p[3]  
p[15]=p[9]*p[6]
```

要得到求乘幂的最优方案，需要耗费大量的时间（见参考文献[3]）。由于二分法的计算次数，与最优的次数非常接近，因此，本文选择了二分法。

<sup>[9]</sup> 我们只需保存已算出的  $1^m, 2^m, \dots, 4999^m$  的值即可。当然，还可以进一步优化空间复杂度，详见程序 Pro\_3.pas。