减少冗余与算法优化

长沙市长郡中学　胡伟栋

【摘要】

在信息学竞赛中，我们经常会遇到冗余，而冗余会造成算法、程序的效率不同程度的降低：有的是微不足道的，而有的会导致算法复杂度大大提高。本文针对后者，举例说明冗余对算法效率的影响和如何减少冗余。

【关键字】

冗余、算法优化

【正文】

一　引言

信息学竞赛中，我们所追求的目标之一，是使程序用最少的时间解决问题，也就是达到最高的效率。实际生活中也同样需要这样，高效率者往往会在竞争中取得优势。

冗余，是指多余的或重复的操作。在搜索、递推、动态规划等诸多的算法中，都会出现冗余。

由冗余的定义，要使算法达到最高的效率，必须去除算法中的冗余处理。但完全去除冗余是难以实现的，使程序用绝对最少的时间解决问题也是很难的，通常需要退一步，将目标改为：**使程序用尽量少的时间解决问题**。由冗余的定义，可以得到：**要提高算法的效率，必须减少算法中的冗余处理**。

要减少冗余，需要在大量的做题过程中，不断探索，不断积累经验。

下面就让我们通过两个具体的例子来研究冗余是如何影响算法效率的以及如何减少冗余。

二　整数拆分

2.1　问题描述[①]

将正整数 *N* 拆分成若干个整数的和，使拆分成的数是 2 的非负整数幂的形式。问有多少种拆分方案。如果两个方案仅有数的顺序不同，它们算作同一种方案。

如：

当 *N*=5 时，可以拆分成下面的形式：

5=1+1+1+1+1

5=1+1+1+2

5=1+2+2

5=1+4

①题目来源：金恺原创

所以，5 有 4 种拆分方案。

2.2　粗略分析

此题可用递推解决(为什么？请读者自己思考 ^_^)：

用 $F[i, j]$ 表示 i 拆分成若干个数，其中最大的数**不超过** 2^j 的拆分总数。则：

1° $F[0, j]=1$

2° $F[i,0]=1$，即 i 拆分成若干个数，其中最大的数不超过 $2^0=1$ 的拆分方案只有一种：把 i 拆分成 i 个 1。

3° 当 $i>0, j>0$ 时， $F[i, j]$ 由两类组成：

第一类：拆分成的最大数正好是 2^j，其总数为 $F[i-2^j, j]$；

第二类：拆分成的最大数小于 2^j，其总数为 $F[i, j-1]$。

所以 $F[i, j]=F[i-2^j, j]+F[i, j-1]$。

最后要计算的目标是 $F[N,M]$。

因为 $i-2^j$ 必须 ≥ 0，所以 $j \leq \lfloor \log_2 i \rfloor$，又有 $1 \leq i \leq N$。不难得出：总的时间复杂度是 $O(N \log_2 N)$[①]，空间复杂度也是 $O(N \log_2 N)$。看上去，这个复杂度已经很低了。但是，复杂度能不能再低一点儿呢？

2.3　减少冗余

为了便于研究，可以首先处理 N 是 2 的整数幂这种特殊情况，然后把 N 不是 2 的整数幂的情况化为 N 是 2 的整数幂的情况处理。

2.3.1　当 N 是 2 的整数次幂时

设 $N=2^M$ (M 为非负整数)。首先，为了讨论时更直观，把所有的 $F[i, j]$ 对应到以 I 为横轴，J 为纵轴的直角坐标系中的每一个整点上，将横坐标为 i 的点称为第 i 列的点、纵坐标为 j 的点称为第 j 行的点。($F[i, j]$ 是第 i 列，第 j 行的点)

若点 C 是点 A 与点 B 的和[②]，则连有向边 $\overrightarrow{AC}$ 和 $\overrightarrow{BC}$ (如图 A 所示)。

[①]此处所讲的时空复杂度都忽略了高精度的因素。因为当 N 达到 10000000 时答案也只有 60 位，这个数字是比较小的。

[②]当 C 为 $F[i, j]$ 时，由递推方程，A、B 分别为 $F[i-2^j, j]$、$F[i, j-1]$。

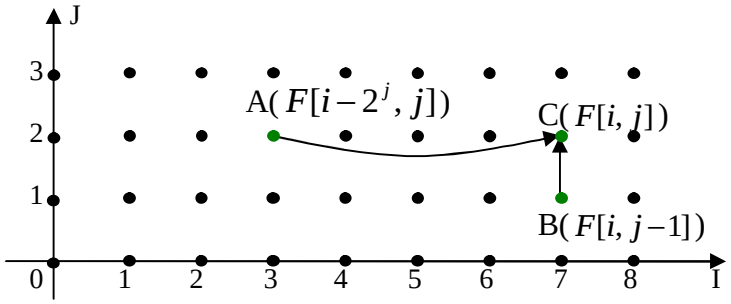

图 A　(*M*=3，*N*=8)

根据递推关系将所有的边都连出来，可以得到图 B。

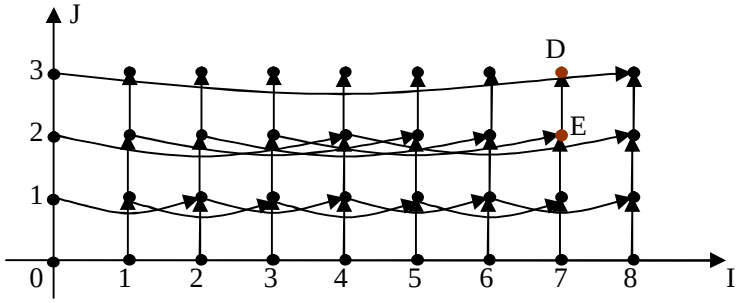

图 B　(*M*=3，*N*=8)

观察要计算的目标 $F[N,M]$，它是 $F[N-2^M,M]=F[0,M]$ 与 $F[N,M-1]$ 的和，而 $F[N,M-1]$ 是 $F[N-2^{M-1},M-1]$ 与 $F[N,M-2]$ 的和；$F[N,M-2]$ 是 $F[N-2^{M-2},M-2]$ 与 $F[N,M-3]$ 的和……可以看出，图中有很多的点(如图 B 中的 D，E)的值求出与不求出都不影响最后的答案，所以这些点都没必要求出，都是冗余的，在图中也没必要向这些点连边。将这些冗余的点和边删掉，只留下最后可能影响到目标点的点和边，图 B 变成了图 C 的样子，可以看出，图 C 比图 B 简洁多了，要计算的点数也少多了。

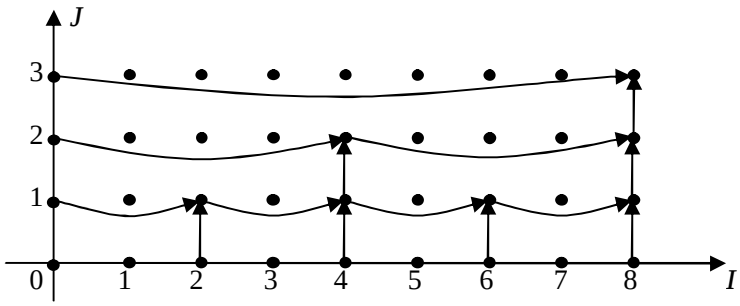

图 C　(*M*=3，*N*=8)

那么，到底要计算多少个点呢？

观察图 C，发现当 j 达到最大，即 $j=M$ 时第 j 行只需要计算 1 个值——$F[N,M]$；当 $j=M-1$ 时第 j 行要计算 2 个值——$F[N,M-1]$ 和 $F[\frac{N}{2},M-1]$；当

$j=M-2$ 时第 J 行要计算 4 个值——$F[\frac{N}{4},M-2]$、$F[\frac{N}{2},M-2]$、$F[\frac{3N}{4},M-2]$ 和 $F[N,M-2]$……由此可以猜想：

①当 $j=M-k$ 时第 j 行要且只要计算 2^k 个值。

②这些要计算的值是 $F[l\,2^j,j](1\le l\le 2^k)$ 这 2^k 个。

这两个猜想是否正确呢？答案是肯定的，下面用归纳法证明：

1° 当 $j=M$ 时，第 j 行只要计算 $F[N,M]$，这是显然的。猜想①、②此时都成立。

2° 假设 $j=M-k+1$ 时成立，第 j 行要计算的为 $F[l\,2^j,j](1\le l\le 2^{k-1})$ 这 2^{k-1} 个数。

取 $j'=j-1=M-k$，当 $l=l_0$ 时，由于第 j 行要计算 $F[l_0 2^j,j]$，由递推方程，第 j' 行要计算 $F[l_0 2^j,j']=F[2*l_0 2^{j'},j']$，而计算 $F[2*l_0 2^{j'},j']$ 又要用到 $F[(2l_0-1)*2^{j'},j']$，计算 $F[(2l_0-1)*2^{j'},j']$ 时要用到 $F[(2l_0-2)*2^{j'},j']$……，一直下去，最后所有的 $F[x*2^{j'},j'](1\le x\le 2l_0)$ 都要算出来。

当 l 取遍所有的 l_0 时，就可以得到第 j' 行要计算哪些值，它们是 $F[x*2^{j'},j'](1\le x\le 2^k)$ 这 2^k 个数，这个式子和②的是一样的。所以此时猜想①、②仍然成立。

综合 1° 和 2°，猜想①、②都是正确的。

由①，图中实际有用的点是 $1+2+2^2+2^3+\cdots\cdots+2^{M-1}=2^M-1=N-1$ 个，而开始时计算了 $N*M$ 个，可见计算过程中的冗余数目远远大于必须计算的数目。如果去掉这此冗余的计算，算法的时间复杂度可能降到 $O(N)$。

现在的问题变为：哪些点是要计算的呢？用②找要计算的点固然是一个方法，不过这里有两个巧妙的结论：

③图中每一列要计算的点必然是最下面的若干个点。

因为：如果要计算 $F[i,j]$，从递推方程看，必定要计算 $F[i,j-1]$，然后又必定要计算 $F[i,j-2]$……，直到要计算 $F[i,1]$，最后 $F[i,0]$ 已知。所以，如果要计算 $F[i,j]$，位于 $F[i,j]$ 正下方的所有点都要算出来，由此，图中每一列要计算的

点必然是最下面的若干个点。

④当 $i=X$ 时，第 i 列要计算的点的个数 $T_i= X$ 的二进制表示中最末的 0 的个数。

证明：

设 i 的二进制表示中最末有 T_i 个 0。

1° 当 $j<=T_i$ 时，$2^j \mid i$，即存在整数 l，使 $i = l\,2^j$。因 $1 \le i \le N$，可得 $1 \le l = \frac{i}{2^j} \le \frac{M}{2^j} = 2^{M-j}$，由②，$F[i, j]$ 要计算出来，即第 i 列至少要计算 j 个值。特别的，取 $j=T_i$，即可得到第 i 列至少要计算 T_i 个值。

2° 当 $j>=T_i+1$ 时，$2^j \nmid i$，即不存在整数 l，使 $i = l\,2^j$，由②，$F[i, j]$ 不必计算出来。所以第 i 列只需要计算 T_i 个值。

综合 1°,2°，命题④成立。

这样，时间复杂度降至 $O(N)$ 已不成问题。

再看一下空间复杂度。所有不必计算的数据都可以不分配内存，前面浪费了大量的空间。怎样才能不浪费呢？

假设程序实现时是按照外层循环从小到大的枚举 i 值、内层循环从小到大枚举 j 值的顺序，则对于 $F[i, j]$，所用到的值为 $F[i, j-1]$ 和 $F[i-2^j, j]$，其中 $F[i, j-1]$ 是第 j-1 行的已求出的点中最右边的点，即当前已求出的 $F[x, j-1]$ (x 是非负整数)中 x 最大的点；$F[i-2^j, j]$ 是第 j 行已求出的点中最右边的点，即当前已知的 $F[x, j]$ (x 是非负整数)中 x 最大的点(如图 D)，这是显然的。可以想到，对于某个 j，只要保存已求出的 $F[x, j]$ 中 x 最大的那个元素即可，这样可以减少浪费，而且能起到类似滚动数组减少空间的效果。空间复杂度可以降到 $O(\log_2 N)$。

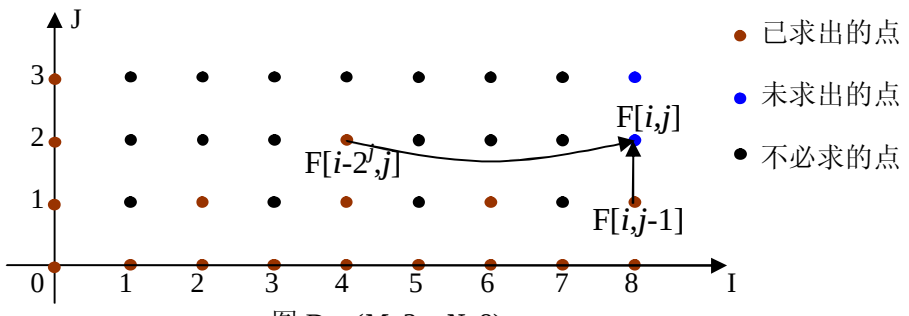

图 D　(M=3，N=8)

2.3.2　当 N 不是 2 的整数次幂时

可设 $N = 2^M - r$，其中 $2^{M-1} < N < 2^M$，这时，计算的目标是 $F[N, M-1]$，由

$F[N,M]=F[N,M-1]+F[N-2^M,M]$，又由于$F[N-2^M,M]=0$(或者说不存在)，$F[N,M]=F[N,M-1]$，可将所求的目标改为$F[N,M]$。

仍然将$F[i,j]$对应到坐标系中，连边，去除其中的冗余，得到图 E (其中 N=5)。添加$F[-r],F[-r+1],F[-r+2],\cdots\cdots,F[-1]$共 r 列，令它们的值全为 0(图 F)。然后将所有的点向右平移 r 个单位，得到的就是类似$N'=2^M$的情况了，但从第 0 列至第 r-1 列的值全为 0，其他列都满足$N'=2^M$的递推关系(如图 G)。变换之后可顺利的求出答案。

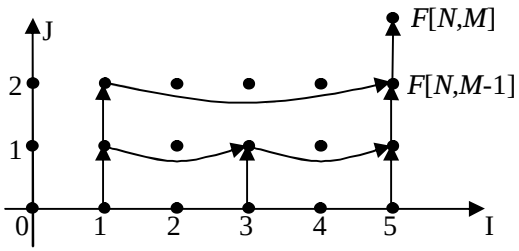

图 E

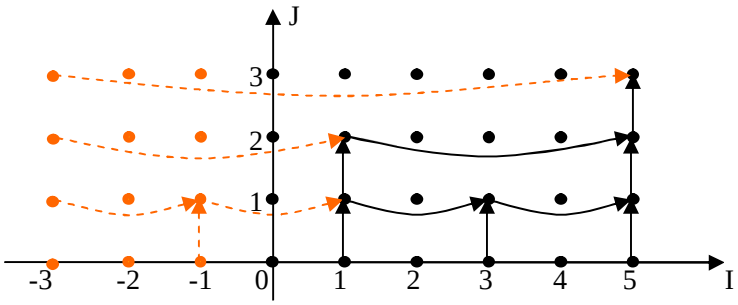

图 F

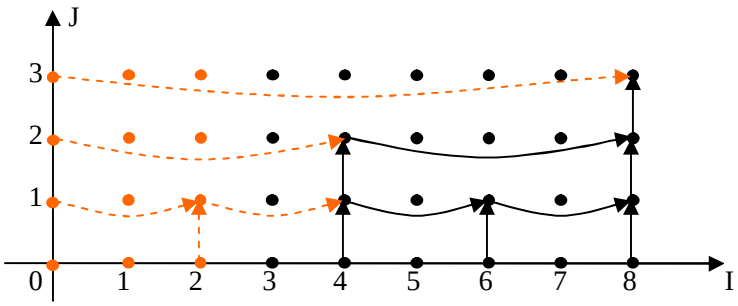

图 G

2.4　小结

回顾此题，由于开始时做了大量冗余的操作，使得时空复杂度相对于最后的

时空复杂度高了很多，当减少冗余后，时空复杂度低了很多。可见，减少冗余是优化本题的关键。

此题还有没有更好的算法呢？有直接的公式可用吗？

探索中 . . .

三　最大奖品价值

3.1　问题描述

作为今年年度消费最高奖获得者，你有机会参加一个促销性的抽奖游戏。该游戏是这样的：

有 N+2 级楼梯，从 0 至 N+1 标号，从第 1 级到第 N 级每级上都放着一个奖品，如果你从第 0 级走 M 步($M \leqslant N+1$)正好到达第 N+1 级楼梯，且不走出这 N+2 级楼梯，你所走过的楼梯上的奖品就是你的了，否则将得不到任何奖品。首先，你给所有的奖品打了一个分值，这些分值都是正整数。你希望得到的奖品的分值和最大，应该怎么走呢？

当然，一步不可能走太多级的楼梯，因为那样有可能摔着，假设每步最少走 0 级(即原地不动),最多走 K 级(即向楼梯标号增大的方向最多从第 i 级走到第 i+K 级，向楼梯标号减小的方向最多从第 i 级走到第 i-K 级)。

3.2　数学模型

原题可如下表示

有一个整数序列 $a_0, a_1, a_2, a_3, \cdots\cdots, a_N, a_{N+1}$，其中 $a_0 = a_{N+1} = 0$，$a_1, a_2, a_3, \cdots\cdots, a_N > 0$，对于一个 $M(M < N)$ 和 K，从序列中找出一个子序列 $a_{i_0}, a_{i_1}, a_{i_2}, \cdots\cdots, a_{i_M}$，使：

1) $0 = i_0$，$i_M = N+1$

2) $-K \leq i_1 - i_0 \leq K, -K \leq i_2 - i_1 \leq K, \cdots\cdots, -K \leq i_M - i_{M-1} \leq K$；

3) $a_{i_0} + a_{i_1} + a_{i_2} + a_{i_3} + \cdots\cdots + a_{i_M}$ 最大

3.3　粗略分析

原问题只说了每一步最多走的级数，没有规定是否能往回走。那么往回走有没有意义呢？假设往回走了，即出现 p，使 $i_{p+1} < i_p$，找出满足 $i_{p+1} < i_p$ 的最小的 p，因 $0 \leq i_p - i_{p+1} \leq K$，$-K \leq i_{p-1} - i_p \leq 0$，所以 $-K \leq i_{p-1} - i_{p+1} \leq K$，若将 i_{p-1}, i_p, i_{p+1} 变

为 i_{p-1}, i_{p+1}, i_p，即变换 i_p 和 i_{p+1} 的位置，仍会满足要求，且分值的和不变。按照这种方法调整，可以使 $i_0 \le i_1 \le i_2 \cdots\cdots \le i_M$。

因为除 $a_0 = a_{N+1} = 0$ 外其他的分值大于 0，在某级楼梯上停留不如走到一级未走过的楼梯，所以进一步可以使 $i_0 < i_1 < i_2 \cdots\cdots < i_M$。

做了上面的处理后，本题变成了一个最优子结构的问题，可以用动态规划解决。用 $F[i, j]$ 表示所选的第 i 个数为 a_j 所能得到子序列的和最大是多少。则 $F[i, j] = \max_{j-K \le x < j} \{F[i-1, x]\} + a_j$。

此算法的状态数为 $N * M$，决策数为 K，所以时间复杂度为 $O(NMK)$。这个算法还能优化吗？

3.4　减少冗余

当计算 $F[i, j-1]$ 和 $F[i, j]$ 时，分别要计算 $\max_{j-K-1 \le x < j-1} \{F[i-1, x]\}$ 和 $\max_{j-K \le x < j} \{F[i-1, x]\}$，而

$$\max_{j-K-1 \le x < j-1} \{F[i-1, x]\} = \max\{F[i-1, j-k-1], \max_{j-K \le x < j-1} \{F[i-1, x]\}\}$$

$$\max_{j-K \le x < j} \{F[i-1, x]\} = \max\{\max_{j-K \le x < j-1} \{F[i-1, x]\}, F[i-1, j-1]\}$$

可以看出：$\max_{j-K \le x < j-1} \{F[i-1, x]\}$ 计算了两次。如果能减少这种冗余计算，复杂度将能够降低。如何处理呢？通常，会有三类方法处理：

第一类方法：设计算 $F[i, j-1]$ 时所得的最大值为 X，当计算 $F[i, j]$ 时，将 X 与 $F[i-1, j-K-1]$ 比较，如果两个数不相等，则必有 $X = \max_{j-K \le x < j-1} \{F[i-1, x]\}$，所以计算 $F[i, j]$ 时的最大值为 $\max\{X, F[i-1, j-1]\}$，只要 $O(1)$ 即可转移；但当 $X = F[i-1, j-K-1]$ 时，就不得不计算 $\max_{j-K \le x < j} \{F[i-1, x]\}$。复杂度为 $O(K)$。

对于这一类方法，当数据比较特殊时，如 $a_1 > a_2 > a_3 > \cdots\cdots > a_N$，总时间复杂度是 $O(NMK)$。没有从根本优化。

第二类方法：用堆、线段树之类的数据结构优化。时间复杂度可降到 $O(NM \log_2 K)$。但编程复杂度将有所提高。

第三类方法：首先，分析动态规划的转移过程，对整个过程进行分析似乎难以下手，因此可以分析一部分，不妨分析从 $F[i-1]$ 到 $F[i]$ 移的这一步。注意到，若 $a<b<j$，且 $F[i-1,b]\geq F[i-1,a]$，当计算 $F[i,j]$ 时，因为有 $F[i-1,b]$ 在，最大的值不可能取 $F[i-1,a]$(当 $F[i-1,a]=F[i-1,b]$ 是最大值时，可以取 $F[i-1,b]$ 而不取 $F[i-1,a]$)，因此在计算 $F[i,j]$ 时可不必枚举 $F[i-1,a]$——$F[i-1,a]$ 是冗余的状态。如果用一个线性表存储 $F[i-1]$ 中所有要枚举的状态，且按标号(数组的第二维下标)从小到大排序，则所有这些状态的值将是单调递减的，这时所求的最大值将是线性表中的第一个元素的值。但怎么实现呢？把线性表改造一下[①]就可以了：当 j 变大时，线性表中所有的 $F[i-1,x](x<j-K)$ 都要删除(实际上，由于 j 是以 1 为增量的，最多将线性表的第一个元素删除)；然后将 $F[i-1,j-1]$ 插入线性表的最后，在插入的过程中，为了保持线性表中的数单调递减的特点，若 $F[i-1,x]<F[i-1,j-1]$，要将 $F[i-1,x]$ 删除，这些元素都是在线性表的末尾，可以从最后逐一删除。这个数据结构中，每个 $F[i-1,x](1\leq x\leq n)$ 最多进入一次队列，最多出一次队列，查找最大元素的复杂度为 $O(1)$。所以总的时间复杂度为 $O(NM)$。下面是一个转移的例子：

F[i-1]=(6 3 5 2 8)　　k=3

j	操作	队列	最大值
	开始	[]	
1	插入 6，直接插入	[6]	6
2	插入 3，直接插入	[6，3]	6
3	插入 5，5 比 3 大，将 3 从线性表最后删除，5 插入到最后位置	[6，5]	6
4	将 6 从线性表的第一个位置删除；插入 2，直接插入	[5，2]	5
5	插入 8，8 比 2、5 大，将 2、5 删除，8 插入到最后位置。	[8]	8

3.5　小结

这道题中，冗余并不像整数拆分那么容易去除，这时，可以考虑利用数据结构。同时也看到，数据结构并不时一尘不变的，只要灵活运用，它必能发挥很大的作用。**在减少冗余操作的过程中，往往要利用到数据结构。**

① 改造后的线性表在项荣璟的论文中出现过。

四　总结

从上面的例子可以看到：

在算法设计和编程过程中，冗余的出现是难以避免的。

冗余是高效率的天敌，**减少冗余，必然能使算法和程序效率提高很多**(例 1 时间复杂度由 $O(N\log_2 N)$ 降到 $O(N)$、空间复杂度由 $O(N\log_2 N)$ 降到 $O(\log_2 N)$，例 2 时间复杂度由 $O(NMK)$ 降到 $O(NM)$，程序的效率提高可想而知)。

减少冗余没有可套用的定理公式可用，只有认真分析、善于探索，并在做题中积累经验，才能得到减少冗余的好方法。

【感谢】

1、在我写这篇论文时，向期中老师、栗师、金恺都给了我很大的帮助，我向他们表示衷心的感谢。

2、衷心感谢任恺、王俊、易伟、康亮环、郑嬰……在临近期末考试时还能为我看论文，向我提很多宝贵的意见。

3、衷心感谢各位评委、老师能在百忙之中抽空组织此次活动。

【参考文献】

《充分利用问题性质——例析动态规划的“个性化”优化》——项荣璟 2003 论文。