减少冗余与算法优化

湖南省长沙市长郡中学
胡伟栋

减少冗余与算法优化

算法的目标： 用最少的时间解决问题

最高的效率

要提高算法的效率，

冗余： 多余的或重复的操作

必须减少算法中的冗余

在搜索、递推、动态规划……中，都可能出现冗余

例 1：整数拆分——问题描述

将整数 N 拆分成若干个整数的和，要求所拆分成的数必须是 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，则它们算作同一种方案。

例 1：整数拆分——样例

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

5 有 4 种拆分方案。

例 1：整数拆分——递推的建立

递推的表示：

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 的拆分方案数。

递推方程：

$$F[i,0]=1 \qquad F[0,j]=1 \quad \text{（初始值）}$$

$$F[i,j]=F[i-2^j,j]+F[i,j-1]$$

最大数是 2^j　　　最大数小于 2^j

目标：　$F[N, \lfloor \log_2 N \rfloor]$

例 1：整数拆分——递推复杂度

$$F[i,j]=F[i-2^j,j]+F[i,j-1]$$

$1 \le i \le N \qquad 1 \le j \le \lfloor \log_2 i \rfloor \le \lfloor \log_2 N \rfloor$

复杂度：

时间复杂度： $O(N\log_2 N)$

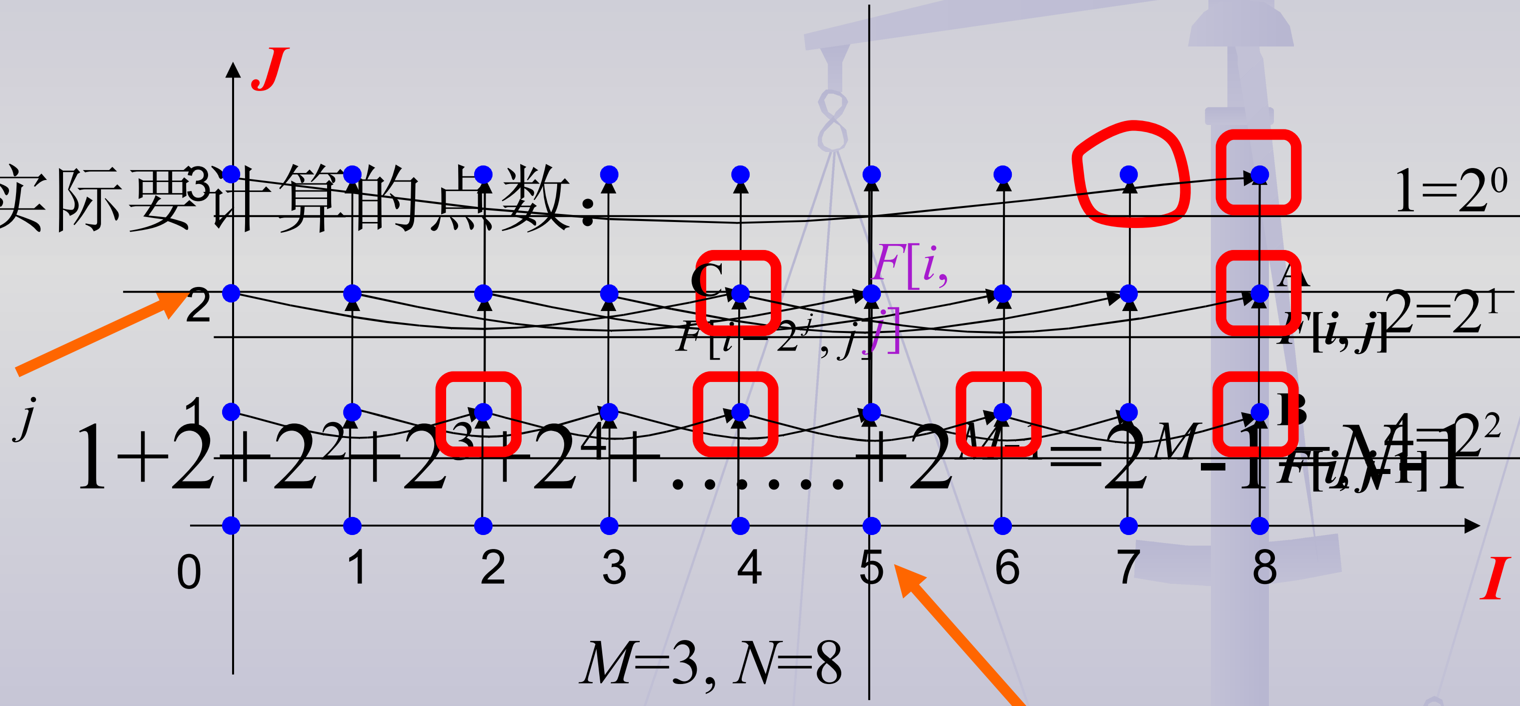
空间复杂度： $O(N\log_2 N)$

例 1：整数拆分—— 递推中的冗余

当 $N=2^M$ (M 是非负整数）时

实际要计算的点数：

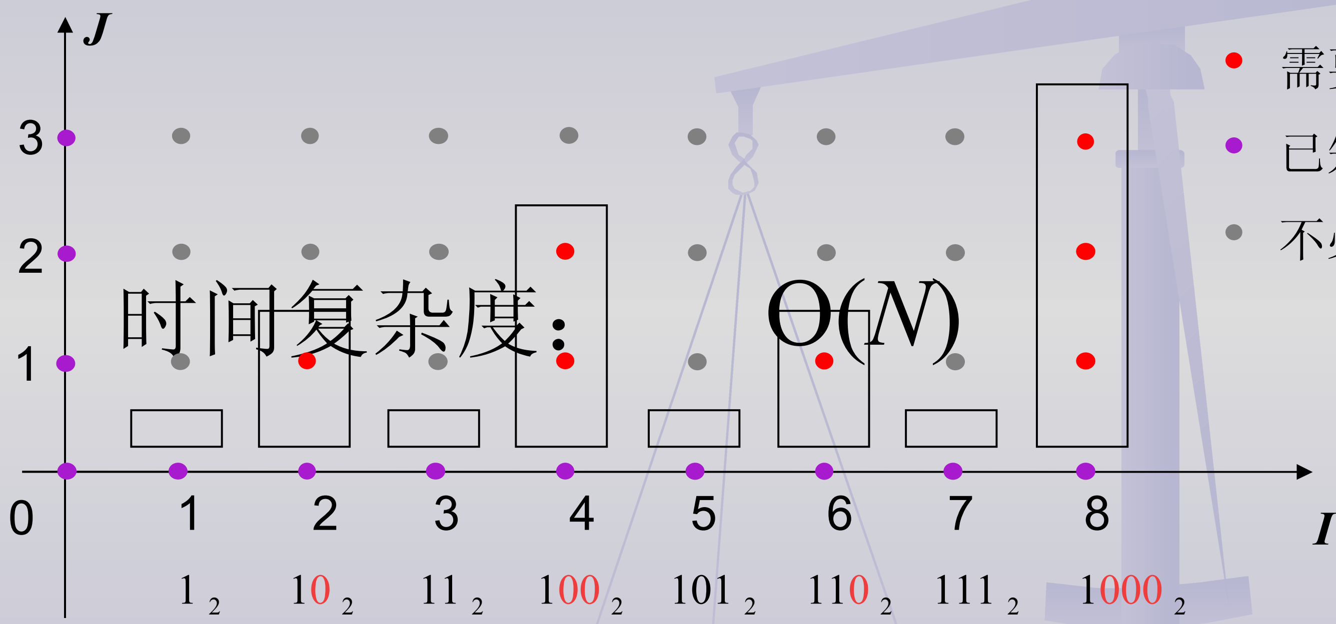
$$1+2+2^2+2^3+2^4+\cdots\cdots+2^{M-1}=2^M-1+M+1$$



当 $j=M-k$ 时，第 j 行要计算的点数为 2^k。

例 1：整数拆分——减少冗余

当 $N=2^M$ (M 是非负整数）时



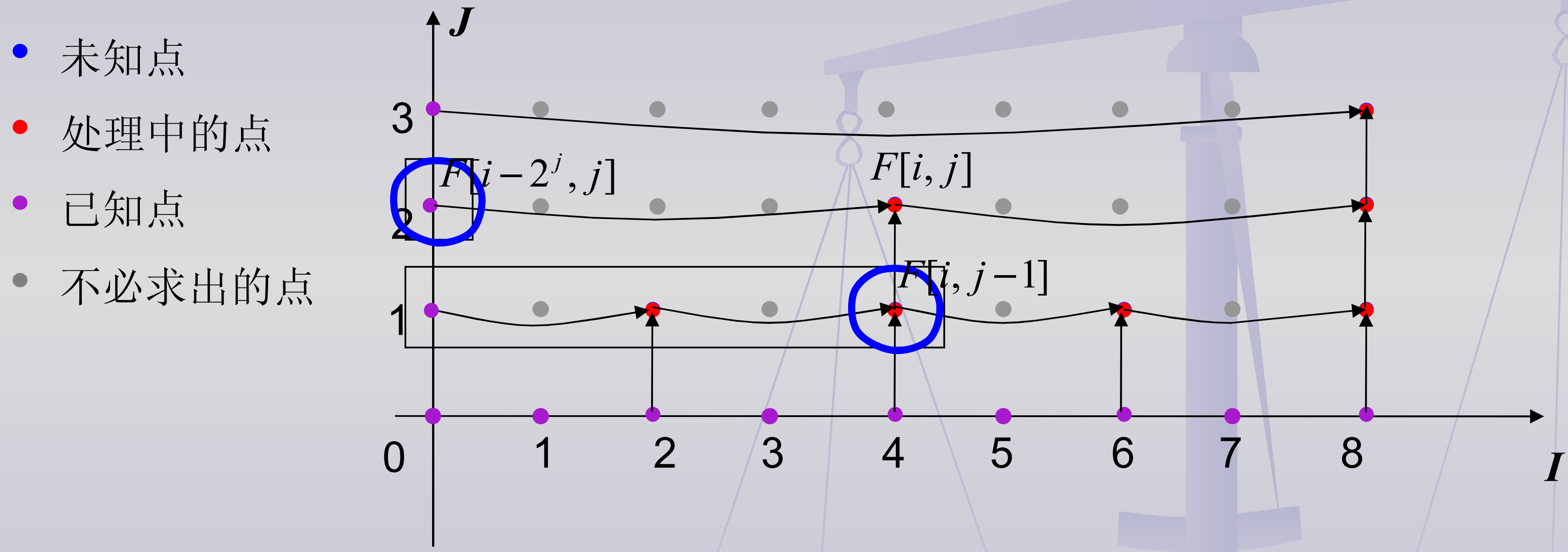
时间复杂度： $O(N)$

当 $i=x$ 时，第 i 列要计算的点数与 x 的二进制表示中最末的 0 的个数相等

每列要计算的点是最下方连续的若干个点

例 1：整数拆分——减少冗余

当 N=2^M (M 是非负整数) 时



在所有 $F[x, j]$ (j 一定 , x 为变量) 中，只要存储 x 最大的一个即可。

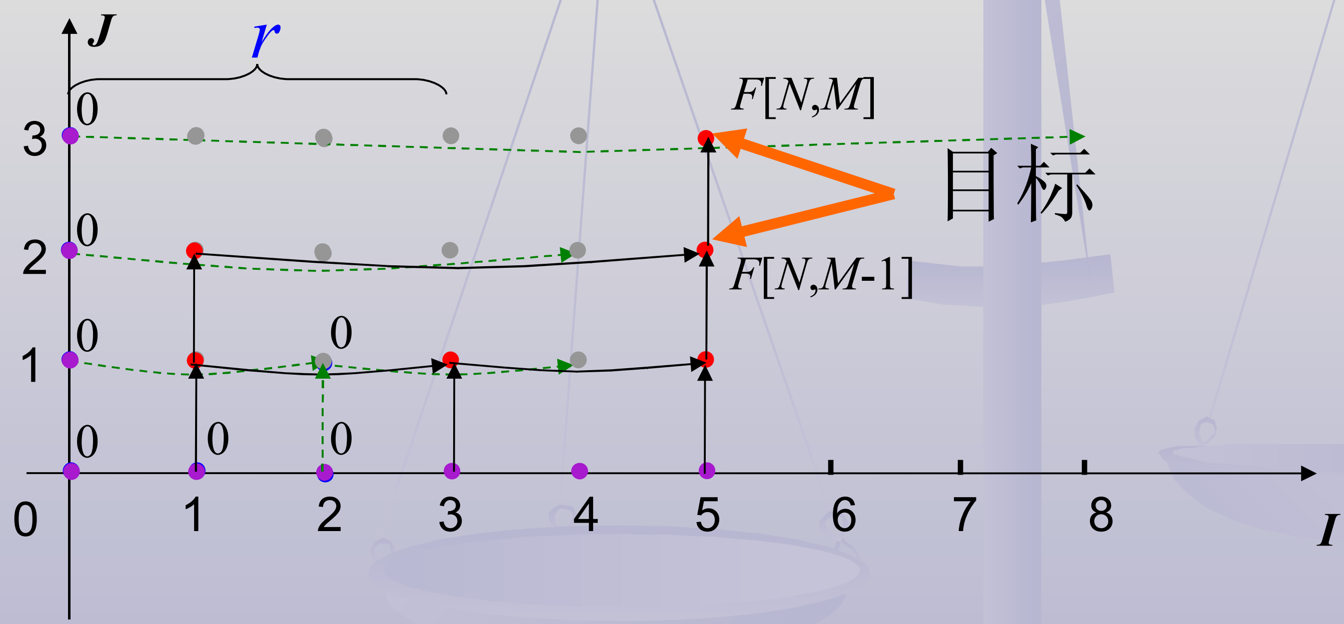
例 1：整数拆分——减少冗余

空间复杂度：$O(\log_2 N)$

例 1：整数拆分——减少冗余

当 $N\neq 2^M$ 时，可转化成 $N=2^M$ 的形式求解

设 $N=2^M-r$ $(2^{M-1}<N<2^M)$



例１：整数拆分——小结

冗余　时空复杂度较高

去除冗余后　时空复杂度相对很低

去除冗余　优化本题的关键

例 1：整数拆分——最后的思考

更优秀的算法？公式？

Exploring . . .

例 2：最大奖品价值——问题描述

有 N+2 级楼梯，分别用 0 至 N+1 编号，第 1 至 N 级楼梯上每级都放有一个奖品，每个奖品都有一个正的价值。如果某人从第 0 级开始，向上走 M 步正好到达第 N+1 级楼梯，他将得到所走过的楼梯上的所有奖品，否则他将一无所获。问能得到的奖品价值的和最大是多少？

当然，一步不可能走太多级楼梯，假设每步最多上 K 级，即最多从第 i 级走到第 i+K 级。

例 2：最大奖品价值——数学模型

有一列数 $a_0, a_1, a_2, \ldots, a_N, a_{N+1}$

其中 $a_0=0$　$a_1,a_2,a_3,\ldots,a_N>0$　$a_{N+1}=0$

从中选 $M+1$ 个数 $a_{i_0}, a_{i_1}, a_{i_2}, \ldots, a_{i_M}$，使

1) $0=i_0<i_1<i_2<\ldots<i_M=N+1$；

2) $i_1-i_0, i_2-i_1, i_3-i_2, \ldots, i_M-i_{M-1}\le K$

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

例 2：最大奖品价值——动态规划

状态表示：用 $F[i, j]$ 表示走 i 步到达第 j 级楼梯能得到的奖品的价值和的最大值

$$F[i, j]=\max_{j-k\leq x<j}\{F[i-1, x]\}+a_j$$

时间复杂度：$O(NMK)$

例 2：最大奖品价值——规划中的冗余

从 $F[i-1]$ 到 $F[i]$ 的转移

$f_1[j]$ 表示 $F[i-1, j]$　　　　$f_2[j]$ 表示 $F[i, j]$

$$f_1[j-k-1] \quad \overbrace{\underline{f_1[j-k] \quad f_1[j-k+1] \quad \cdots \quad f_1[j-3] \quad f_1[j-2]} \quad f_1[j-1]}^{\max \ f_2[j]}$$

$$\underbrace{f_1[j-k-1] \quad f_1[j-k] \quad f_1[j-k+1] \quad \cdots \quad f_1[j-3] \quad f_1[j-2]}_{\max \ f_2[j-1]} \quad f_1[j-1]$$

例 2：最大奖品价值——减少冗余

高

静态的考虑： 线段树 $O(NM\log_2 K)$

动态的考虑： 堆 $O(NM\log_2 K)$

每次都是找 f_1 中连续的一段

每次要求的 f_1 的一段都是变化的

编程复杂度

每次会加入一个新元素

每次会删除一个元素

例2：最大奖品价值——减少冗余

$a<b<j$ $\qquad f_1[a]\le f_1[b]$

$$\underbrace{f_1[j\text{-}k]\ f_1[j\text{-}k+1]\ \ldots\ \cancel{f_1[a]}\ \ldots\ \boxed{f_1[b]}\ \ldots\ f_1[j\text{-}1]}_{f_2[j]}\quad f_1[j]$$

对于任意 $a<b<j$

只有 $f_1[a]>f_1[b]$ 时，$f_1[a]$ 才有用

例2：最大奖品价值——减少冗余

$j\text{-}k \le a_1 < a_2 < a_3 < \ldots\ldots < a_r < j$

$f_1[a_1] > f_1[a_2] > f_1[a_3] > \ldots\ldots > f_1[a_r]$

max

数据结构： 线性表 *

队列 堆栈

删除第一个元素

$f_1[a_1]$	$f_1[a_2]$	$f_1[a_3]$	$f_1[a_4]$	$f_1[a_5]$	$f_1[a_6]$	$f_1[a_7]$	$f_1[a_8]$	x

新增元素到最后一个位置，并维护这个数据结构使它保持递减的性质

x

例 2：最大奖品价值——时间复杂度

时间复杂度　　　O(NM)

例 2：最大奖品价值——小结

$$
\begin{array}{l}
O(NM\log_2 K) \\
\nearrow_{\text{堆}} \\
O(NMK) \xrightarrow[\text{去除冗余}]{\text{线性表 *}} O(NM) \\
\searrow^{\text{线段树}} \\
O(NM\log_2 K)
\end{array}
$$

例 2：最大奖品价值——小结

去除冗余 $\xrightarrow[\text{数据结构}]{\text{探索}\quad\text{分析}}$ 降低复杂度

选取一个最合适的数据结构

总结

- 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，必然会使算法和程序效率提高很多
- 去除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，才能得到去除冗余的好方法

总结

如果在做题时和做题后，思考一下，能否有更好的方法解决此题，此题还有冗余能去吗，必然会得到意想不到的收获。

谢谢