

# 探寻深度优先搜索中的优化技巧

——从正方形剖分问题谈起

长沙市长郡中学

金恺

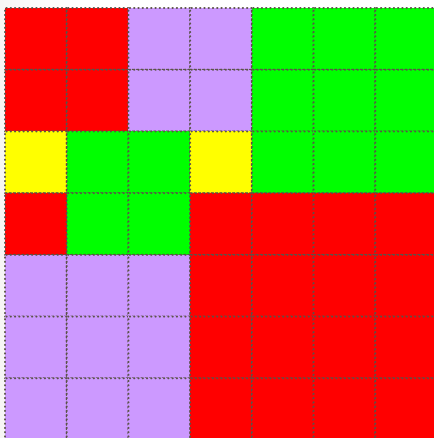
# 正方形分割问题

问题描述:

将  $n \times n$  个小格组成的大正方形分割成若干个较小的整数边长的正方形, 要求分成的小正方形数目最小。

范围:  $1 \leq n \leq 32$ 。

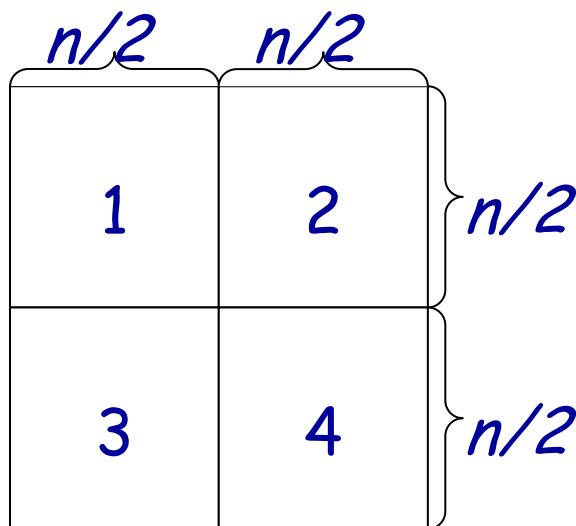
编程环境: *FreePascal*。可用 64MB 空间



$n=7$  时的一个最小数目的分割方案, 需要 9 个小正方形。

# 分析

当 $n$ 为偶数时最少需要4个小正方形：

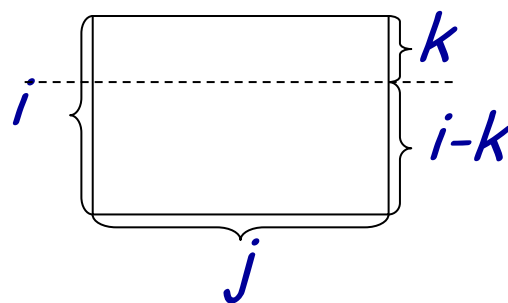
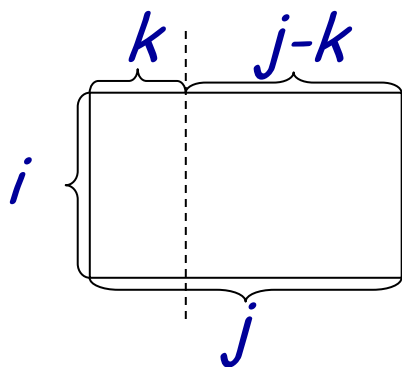


当 $n$ 为奇数时，很难发现有什么数学规律。



设  $f_{i,j}$  表示一个  $i \times j$  的矩形最少可以被剖分成多少个小正方形：

$$f_{i,j} = \begin{cases} 0 & i = 0 \text{ 或 } j = 0 \\ 1 & i = j, i < N \\ \min \begin{cases} f_{k,j} + f_{i-k,j} & (0 < k < i) \\ f_{i,k} + f_{i,j-k} & (0 < k < j) \end{cases} & i \neq j \text{ 或 } i = j = N \end{cases} \quad i > 0, j > 0$$



$n=7$  时，求出结果为 10，并不是最优值。原因：最优方案不一定能被某条直线分割。

$n$	7	11	13	17	19	23	29	31	其他
$f_{n,n}$	10	12	12	14	14	16	16	16	相同
最优值	9	11	11	13	13	13	14	15	
相差	1	1	1	1	1	3	2	1	0

$f_{n,n}$  仅是一个可行解，不过其值与最优解十分接近的。

目前只能用搜索！

优化：搜索之前先用上述动态规划方程求出一个较优值，  
限制搜索层次。

搜索量巨大，仅用这一条优化，效率十分低下。

如何搜索？

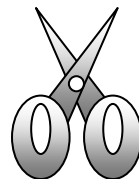
# 搜索的对象和顺序

从大量的搜索题(比如说IOI的Depot等)可以看出, 搜索的顺序和对象是十分重要的, 本题应该用什么作为搜索对象, 搜索顺序又是怎样呢?

搜索的对象: 每个小正方形的位置和边长。

一种最简单的搜索顺序为: 给大正方形中第 $i$ 行第 $j$ 列的小格编号 $(i-1)n+j$ , 每次选择编号最小的未覆盖的小格作为小正方形左上角的坐标, 然后枚举它的边长。

速度太慢, 需要大量剪枝!





# 剪枝

1、避重性剪枝：(对称性)规定左上角的那个小正方形的边长小于等于其他三个角上的小正方形的边长，右上角的小正方形边长大于等于左下角的小正方形边长。

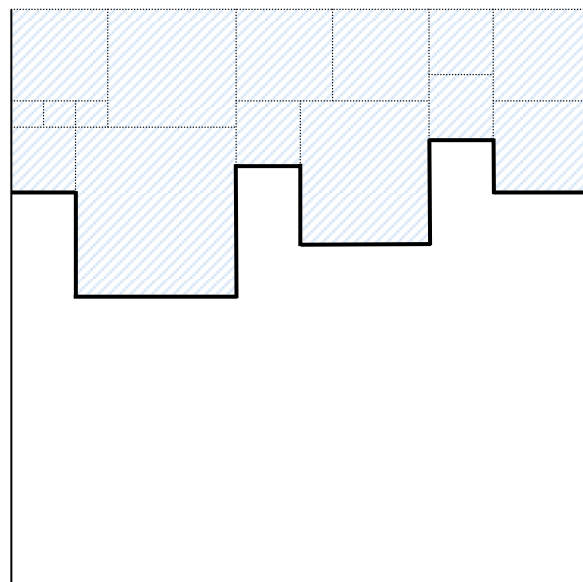
2、平方和剪枝： $11=1^2+1^2+3^2$ 。

设 $Sum_i$ 表示至少需要多少个整数才能使得这些整数的平方和等于 $i$ ,则

$$Sum_i = \begin{cases} 0 & i = 0 \\ \min\{Sum_{i-j^2}\} + 1 (j \leq \sqrt{i}) & i > 0 \end{cases}$$

# 改进搜索的顺序

按上述搜索顺序，每一步中大正方形被覆盖的区域都是凹凸不平的(如右图)，不方便剪枝，搜索效率也比较低。



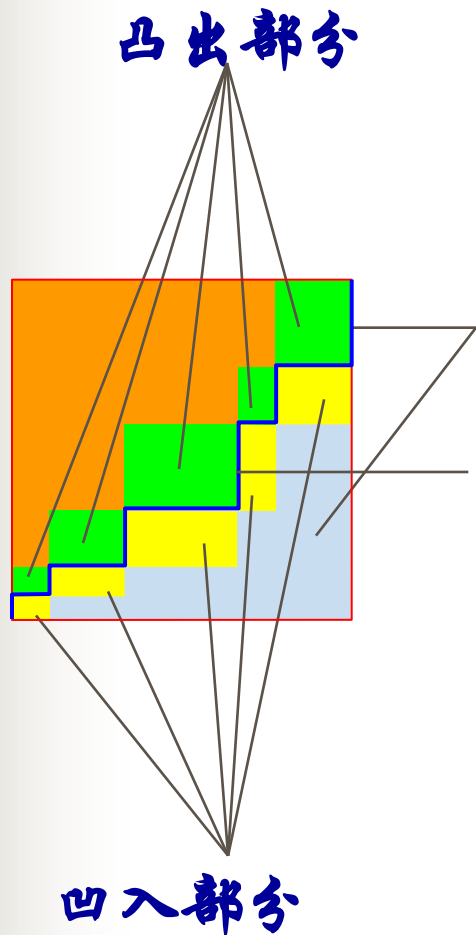
有没有更高效的搜索顺序呢？

既要不遗漏(正确性)、不重复(高效性)；  
又要能够方便的剪枝。



# 探寻新的搜索顺序

定义：



猜想：任何划分方案是否都能通过若干条划分线把一个个小正方形划分出来呢？

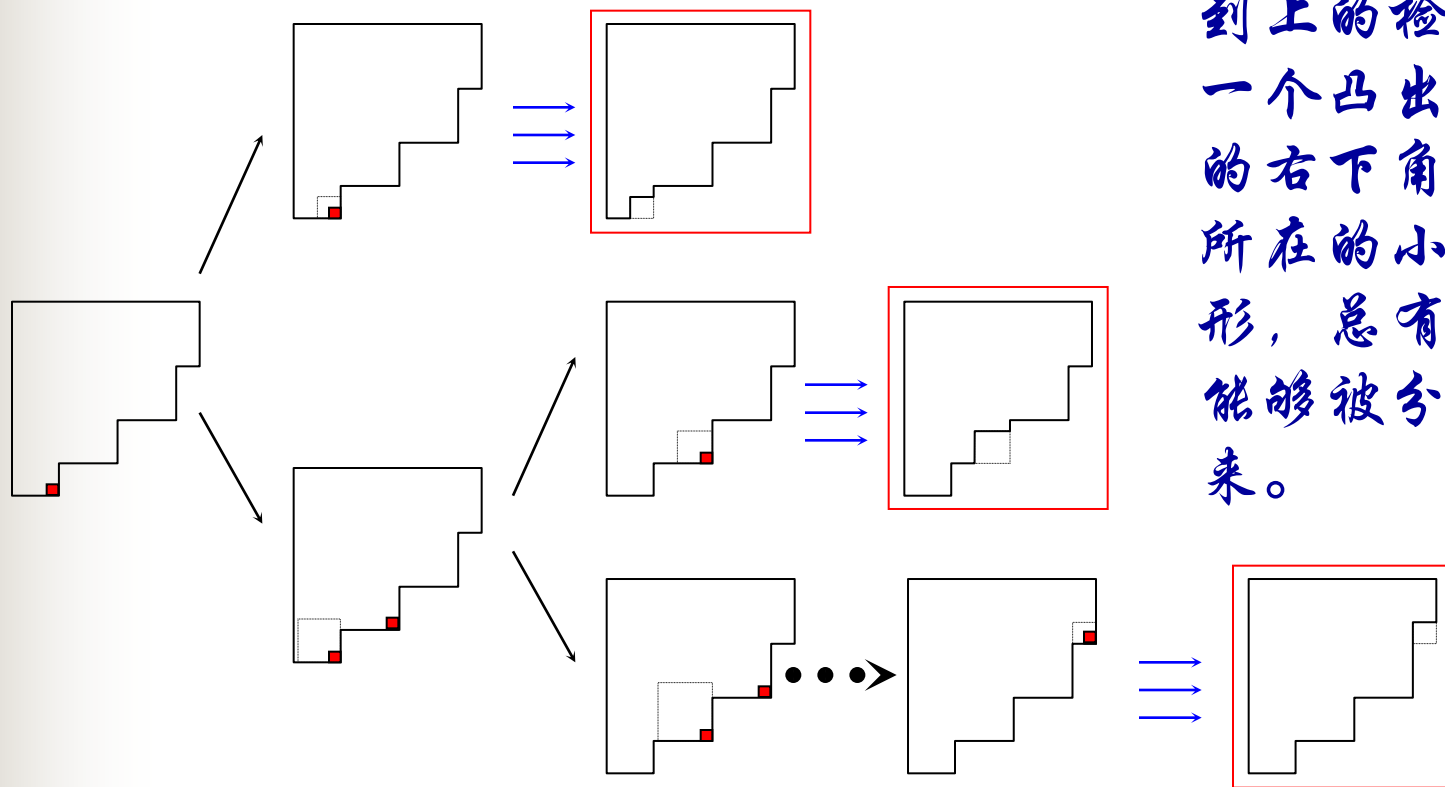
类矩形

划分线——  
一条从左下角到  
右上角的只往右  
或往上走的曲线

# 证明

大正方形就是一个类矩形，他的右边界和下边界连起来就是一条分割线；

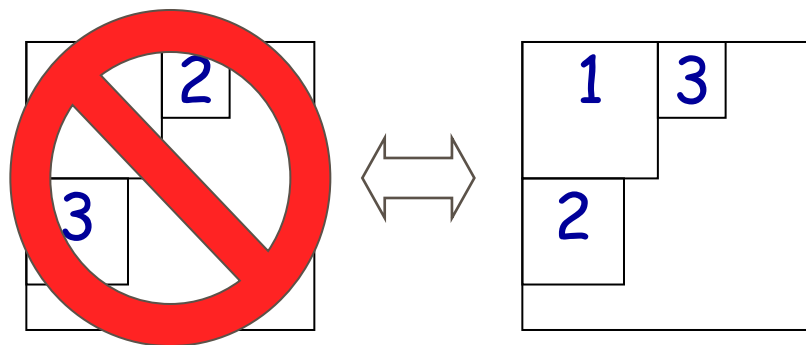
对于任意一个类矩形，从下到上的检查每一个凸出部分的右下角格子所在的小正方形，总有一个能够被分割出来。



# 新的搜索顺序

搜索时每次选择一个凹入部分的左上角作为小正方形的左上角，小正方形的边长小于等于这个凹入部分的宽，就能保证在任何时候已覆盖区域为一个类矩形。

缺点：相同的剖分方案可能会被搜索多次。



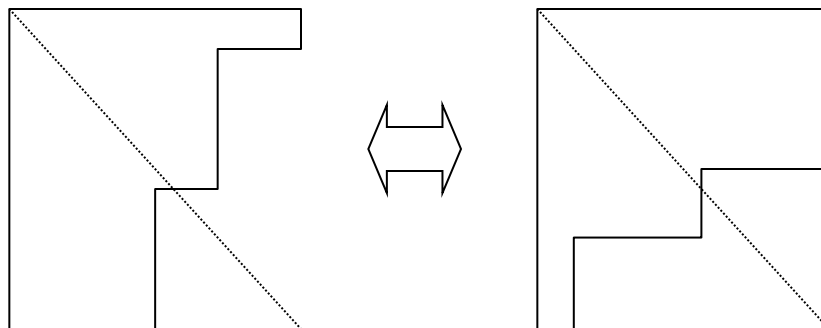
规定前一个小正方形不完全处于后一个的上部。

# 新增剪枝

1、一个类矩形若包含 $x$ 个凸出部分，则它至少需要 $x$ 个小正方形才能拼出，因为每个凸出部分右下角的小格都要属于不同的小正方形中。

实践证明：大多数情况下这个下界比 $Sum_i$ 要精确得多。

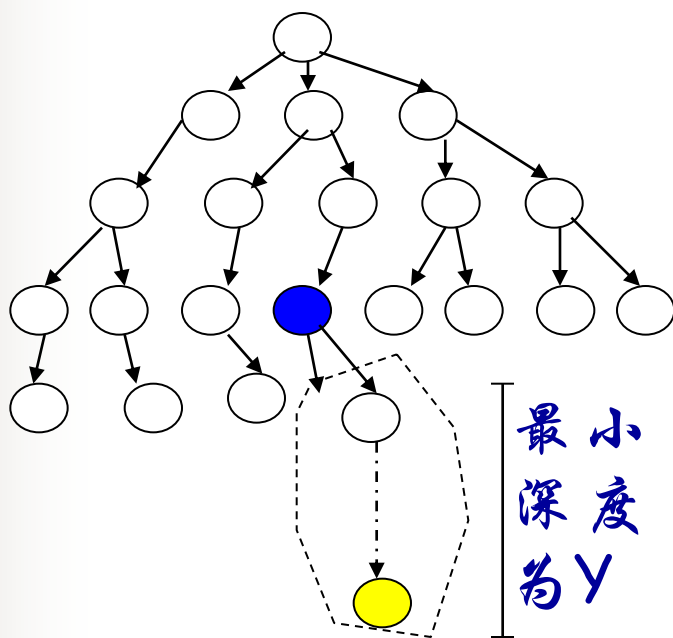
2、对右图所示的两种情况：它们沿“左上一右下”对角线是对称的，可以只搜它们中的任意一种情况。



搜索效率：在最慢时( $n=31$ )只要20s左右就能够出解了。(测试环境 (R)4 CPU 1.7GHz)

还有什么方法能够使搜索效率再大幅提高呢?

看下边这棵搜索树:



当搜到蓝色结点时, 若已经使用了 $x$ 个小正方形, 而这个节点到目标结点的最小深度为 $y$  (也就是说最少可以用 $y$ 个小正方形拼完此时的未覆盖区域), 那么若往下搜, 最优值显然刚好为 $x+y$ 。







# 状态的表示

首先，某个节点到目标节点的最优值只与当前未覆盖区域的形状有关，所以只要以未覆盖区域作为当前状态。

那么如何描述一个未覆盖区域呢？

可用逆序 $n$ 元组 $(a_1, a_2, \dots, a_n)$ ， $a_i \geq a_{i+1}$ ， $n \geq a_1$ 表示，其中 $a_i$ 表示第 $i+n-1$ 行未被覆盖的小格子数目。

分析：一个状态需要占用 $n$ 个字节，空间太多，而且比较两个状态是否相等的时间复杂度为 $O(n)$ ，速度也是非常慢的。

有没有更好的方法呢？

# 优化状态的表示

不妨令  $b_i = a_i - a_{i+1}$  ( $0 \leq i \leq n$ , 这里  $a_0 = n$ ,  $a_{n+1} = 0$ ), 则有  $b_i \geq 0$ ,  $\sum b_i = n$ 。由组合数学知识可知  $b$  的个数为: 把  $n$  个无标号的小球放入  $n+1$  个有标号的盒子中的方案数, 等于  $C_{2n}^n$ , 当  $n=31$  时也不超过  $1e18$ , 故可考虑用一个 `comp` 类型数对应一个多元组  $b$ 。

而  $b$  与  $a$  也是一一对应的, 所以可用一个 `comp` 类型整数直接描述一个状态。

如何将  $a$  与一个整数互相对应呢?

设  $Tot_{i,j}$  表示: 满足  $j \geq a_i \geq a_{i+1} \geq \dots \geq a_n \geq 0$  的序列  $\{a_k (i \leq k \leq n)\}$  的个数。

# 状态的对应的实现

将n元组a转化为整数s

$s \leftarrow 0;$

for  $i \leftarrow 1$  to  $n$  do  $s \leftarrow s + \text{Tot}[i, a[i] - 1];$

将整数s转化为n元组a:

$a[0] \leftarrow n;$

for  $i \leftarrow 1$  to  $n$  do

$a[i] \leftarrow a[i - 1];$

while  $s < \text{Tot}[i, a[i] - 1]$  do  $a[i] \leftarrow a[i] - 1$

$s \leftarrow s - \text{Tot}[i, a[i] - 1];$

$$\text{Tot}_{i,j} = \begin{cases} j+1 & i = n \\ 1 & j = 0 \\ \text{Tot}_{i,j-1} + \text{Tot}_{i+1,j} & j > 0 \end{cases} \quad i < n$$

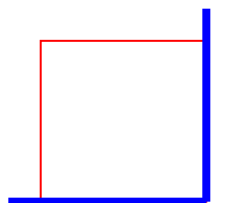
# 动态规划

由于优化了状态的表示，它可用一个整数方便的描述，而且问题很明显满足最优性和无后效性，所以很快就会想到用动态规划来求状态的最优值：

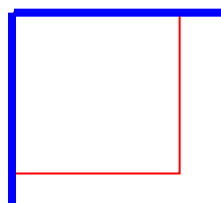
设  $f_i$  表示整数  $i$  所对应的未覆盖区域至少需要多少个小正方形拼成：

$$f_i = \begin{cases} 0 & i = 1 \\ \min\{f_j\} + 1 \{ \text{状态} j \text{ 到} i \text{ 只需添加} 1 \text{ 个正方形} \} & i > 1 \end{cases}$$

$i=1$  时对应的剖分线：



要求解的剖分线：



# 实现方法

显然无法求出所有的状态的 $f_i$ 值，但是我们只需求出其中的若干个就足够了。当然应该尽量多求一些，因为求出的状态越多搜索的效率就会越高。

观察上面的动态规划方程，假如把将每个状态看作一个点，若状态 $i$ 添加一个小正方形能变成状态 $j$ 就连一条有向边 $i \rightarrow j$ ，那么 $f_i$ 就是点1到点 $i$ 的最短距离。由于每条边长度都为1，可以从点1开始广搜，不断扩展直到产生出足够多的状态。



# 实现方法

为了搜索时查找方便，每扩展出一个节点，就将它放入 *Hash* 表中。

搜索时，每搜到一个节点，就查找当前的状态 *i* 是否在表中，如果找到了相应的 *f*<sub>*i*</sub> 值就可改进当前最优值然后直接回溯。

查找的复杂度仅为常数 ( $\leq 9$ )

算法缺陷：每搜索一个节点时都要进行一次查找，在多数情况下其实是找不到的，白白浪费了大量的时间。



# 实现方法

优化：只计算 $f_i$ 小于 $\leq c$ (自定的常数)的状态。搜索时，若已经使用了 $x$ 个小正方形，且 $x+c+1 \geq$ 当前最优解，那么就查找未覆盖区域对应的 $f_i$ 值：

1、若未找到，说明少于 $c+1$ 个小正方形不能拼成未覆盖区域，也就是说搜下去的话最少也要 $x+c+1$ 个，不比当前最优值更优；

2、如果找到了，那么搜下去的最优值显然为 $x+f_i$ ，就可直接令当前最优值 $=\min\{\text{当前最优值}, x+f_i\}$ ；

这样一来，避免了无谓的查找，效率大大提高。

# 搜索效率

经过以上的优化过程，搜索的层次能够减少了C层，效率得到了大大的提高。

如果结合各种各样剪枝条件，最大数据( $n=31$ )也仅要4秒多。

n	运行时间	n	运行时间
$\leq 17$	0.00s	25	0.22s
19	0.11s	27	0.38s
21	0.11s	29	2.86s
23	0.44s	31	4.12s



至此，已圆满解决了本题。

# 回顾解决本题的步骤

- 1、简单的动态规划  $F_{i,j}$ ，求较优解；
  - 2、根据本题“划分的对称性”等特点设计了一些简单的剪枝；
  - 3、通过合理的划分划分方案，得到新的搜索顺序，找到了新的剪枝方法；
  - 4、通过将每个状态对应于一个整数，方便的借助高效的动态规划算法来求某些状态值，从而降低了搜索的层数。
- 经过以上这4步，搜索的效率逐步提高！特别是第3步和第4步效率提高得十分明显。

# 总结

搜索，因其时间效率的“先天不足”，人们有针对性的研究出了很多优化技巧。

一、搜索的对象和顺序是优化中非常重要的一个环节，好的顺序往往是高效率的前提。但是，它也仅仅是一个前提，要保障高效率仅仅靠它是不够的；

二、“剪枝”是最常见的优化方法，几乎可以说，只要用搜索解决问题，就必须考虑剪枝优化。但是，无论剪枝多么巧妙，都不能从本质上降低搜索算法的时间复杂度。



# 总结

三、预求某些状态的最优值，降低搜索的层数。它能够起到降低搜索复杂度的功效。

由于FreePascal的开发使用，使得空间上的限制放松了上百倍，而题目对运行时间的限制历来都是十分苛刻的，这使得我们想到用牺牲空间的代价来换取更多的时间。若把它利用到搜索中来，就产生以上这种优化方法。


这一优化中最重要的一步就是状态的表示。

算法的优化是永无止境的，搜索的优化更是如此，需要我们具体问题具体分析，充分发挥思维的灵活性。只有积累了足够的经验，才能够熟练的应用各种优化技巧。



*Thank you!*





$n$	13	23	29
$F'n,n$	12	14	15
最优值	11	13	14