算法大题

迭代法

迭代法是一种常用算法设计方法。迭代是一个不断用新值取代变量的旧值，或由旧值递推出变量的新值的过程。迭代机制需要以下一些要素：

①迭代表达式；

②迭代变量；

③迭代初值；

④迭代终止条件。

基本格式如下：

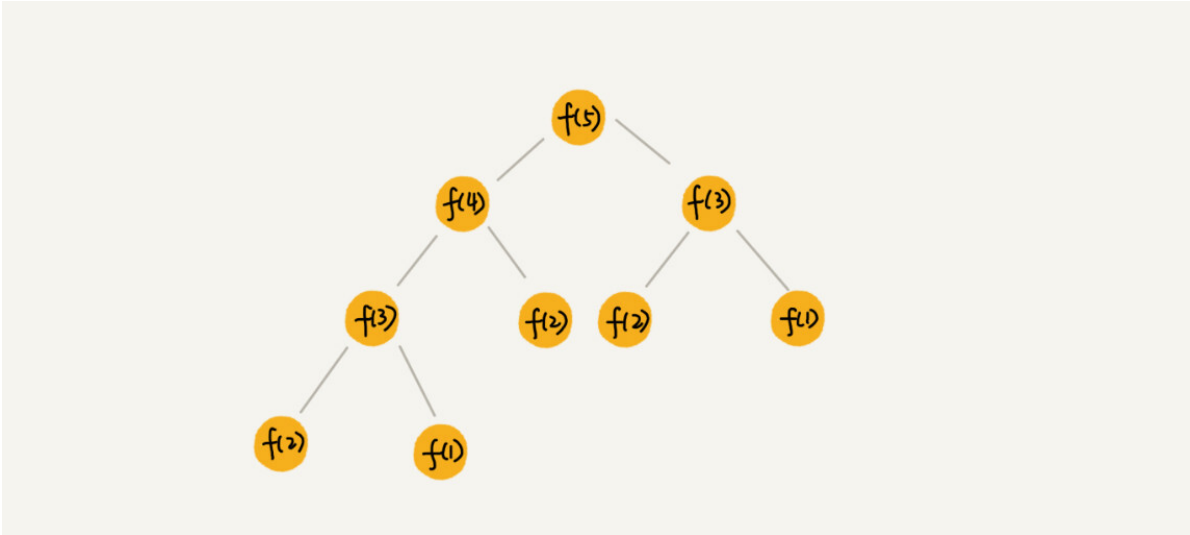
```
迭代变量赋初值；
循环语句
{
计算迭代式；
新值取代旧值；
}
```

递归树

递归的思想就是，将大问题分解为小问题来求解，然后再将小问题分解为小小问题。这样一层一层地分解，直到问题的数据规模被分解得足够小，不用继续递归分解为止。

如果我们把这个一层一层的分解过程画成图，它其实就是一棵树。我们给这棵树起一个名字，叫作**递归树**。

斐波那契数列的递归树如下图所示：



动态规划

最优子结构性质：某个问题的最优解包含着其子问题的最优解。这种性质称为最优子结构性质。

五条基本动态规划要素：

1. 问题建模，优化的目标函数是什么？约束条件是什么？
2. 问题的优化函数值与子问题的优化函数值存在 着什么依赖关系？(递推方程)
3. 如何划分子问题（边界）？
4. 是否满足优化原则？
5. 最小子问题怎样界定？其优化函数值，即初值等于什么？

最长公共子序列：

定义：从序列A中不改变顺序的取出部分元素，如A={a,b,c,d,e,f}，子序列可以为{b,d,e}

子问题的界定：

参数i 和 j 界定子问题

X 的终止位置是i，

Y 的终止位置是 j

Xi =<x1,x2,.....,xi>，Yj =<y1,y2,.....,yj>

也就是小一点的表(备忘录)。

标记函数：从最后往前推，最终解是从哪一格推出的就指向哪一格。

递推方程如下：

$$c[i,j]=\begin{cases}0, & i=0||j=0\\ c[i-1,j-1]+1, & i,j>0,x_i=y_j\\ Max\{c[i-1,j],c[i,j-1]\}, & i,j>0,x_i\neq y_j\end{cases}\qquad(1)$$

最长公共子序列

贪心法：

贪心法是从问题的某一个初始解出发，在每一个阶段都根据贪心策略来做出当前最优的决策，逐步逼近给定的目标，尽可能快地求得更好的解。当达到算法中的某一步不能再继续前进时，算法终止。贪心法可以理解为以逐步的局部最优，达到最终的全局最优。

思路和上面一致，由小到大。

贪心法的特点！！！！！！！

每个阶段贪心法都做出对眼前来讲是最有利的选择，不考虑该选择对将来是否有不良影响。
每个阶段的决策一旦做出，就不可更改，该算法不允许回溯。
贪心法是根据贪心策略来逐步构造问题的解。如果所选的贪心策略不同，则得到的贪心算法就不同，贪心解的质量当然也不同。该算法的好坏关键在于正确地选择贪心策略。
贪心法具有高效性和不稳定性，因为它可以非常迅速地获得一个解，但这个解不一定是最优解，即便不是最优解，也一定是最优解的近似解。

贪心法的基本要素！！！！！！！！

用贪心法求解的问题中，一般都具有两个重要的性质：最优子结构性质和贪心选择性质。

（1）最优子结构性质

一个问题能够分解成各个子问题来解决，通过各个子问题的最优解能递推到原问题的最优解。那么原问题的最优解一定包含各个子问题的最优解，这是能够采用贪心法来求解问题的关键。因为贪心法求解问题的流程是依序研究每个子问题，然后综合得出最后结果。而且，只有拥有最优子结构性质才能保证贪心法得到的解是最优解。

（2）贪心选择性质

贪心选择性质是指所求问题的整体最优解可以通过一系列局部最优的选择获得，即通过一系列的逐步局部最优选择使得最终的选择方案是全局最优的。其中每次所做的选择，可以依赖于以前的选择，但不依赖于将来所做的选择。

可见，贪心选择性质所做的是一个非线性的子问题处理流程，即一个子问题并不依赖于另一个子问题，但是子问题间有严格的顺序性。

贪心法的解题步骤！！！！！！

分解：将原问题分解为若干个相互独立的阶段。
解决：对于每个阶段依据贪心策略进行贪心选择，求出局部的最优解。
合并：将各个阶段的解合并为原问题的一个可行解

回溯

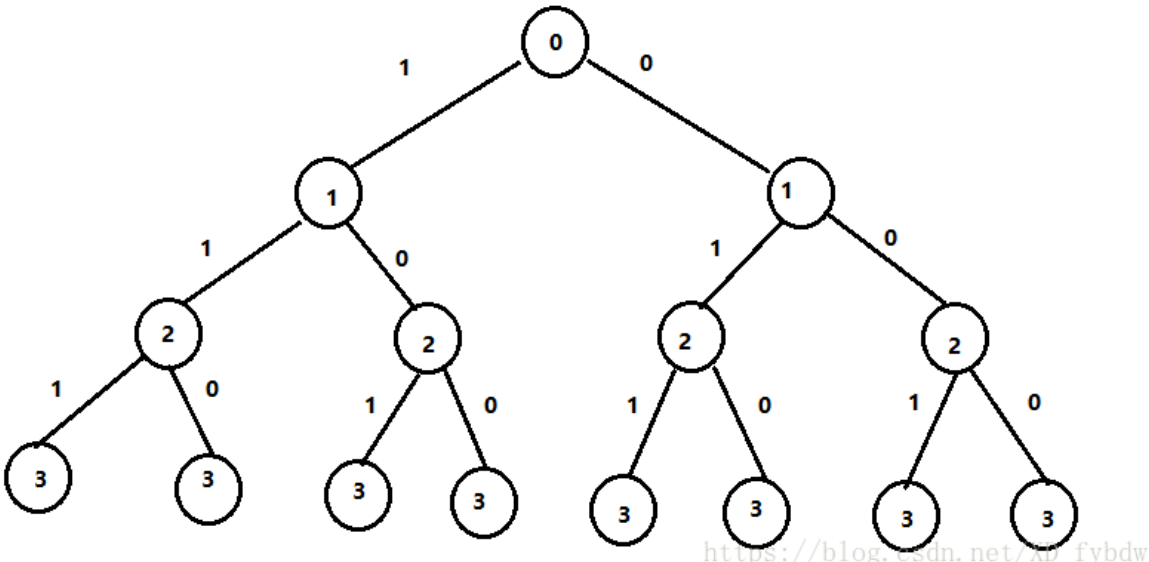
回溯法，又叫试探法，是一种寻找最优解的暴力搜寻法。但是，由于暴力，回溯法的**时间复杂度较高**，因此在比较一些数字较大的问题时，比如上次我们提到的最短路径问题等，运行时间一般比较长。在回溯法中，深度优先搜索是一种很重要的工具。

解空间：顾名思义，就是一个问题的所有解的集合。（但别忘了，这离我们要求的**最优解**还差很远！）

树的画法：

0-1背包：

左边就选择，右边不选择，节点值为背包内有的物品数量。



0-1背包问题：

背包8kg，可供选择的巧克力如下：

1	**费列罗**	**4kg**	**$4500**
2	好时之点	5kg	$5700
3	德芙	2kg	$2250
4	Cudie	1kg	$1100
5	自制	6kg	$6700

回溯法讲究“暴力”。我们从暴力的角度思考，想把**所有**的尽量装满背包的搭配都找出来，标记每一种装法（每一个解）**最大value**，从而找到最优解。

我们从第一种巧克力开始装，然后找下一个，判断能否装入，再**递归**，到达边界，比较，记录较优解，回溯，继续往下找。。。循环。

从**子集树**的角度将，我们**优先选择走左子树**，也就是**入包**；当走到叶结点或不符合约束的重量条件时，回溯到父结点，进入右结点，最后遍历全树。

判断能否装入后可以用一个**book**数组来标记是否选择入包。

01背包问题

实际问题

货郎问题

给定n个城市的无向带权图G(V,E)，顶点代表城市，权值代表城市之间的距离。若城市之间没有路径，则距离为无穷。

城市之间的距离存放在二维数组g[][]中。

从城市1出发，先到临近城市2，将走过的路程存放在变量 cl 中。

bestl代表当前找到的一种最短路径长度。如走法：1-2-3-4-5-1

显然，向城市深处走时，cl只会增加。因此当cl>bestl时，不必再往深处走。

限界条件为cl<bestl, cl 初值为0，bestf初值为∞

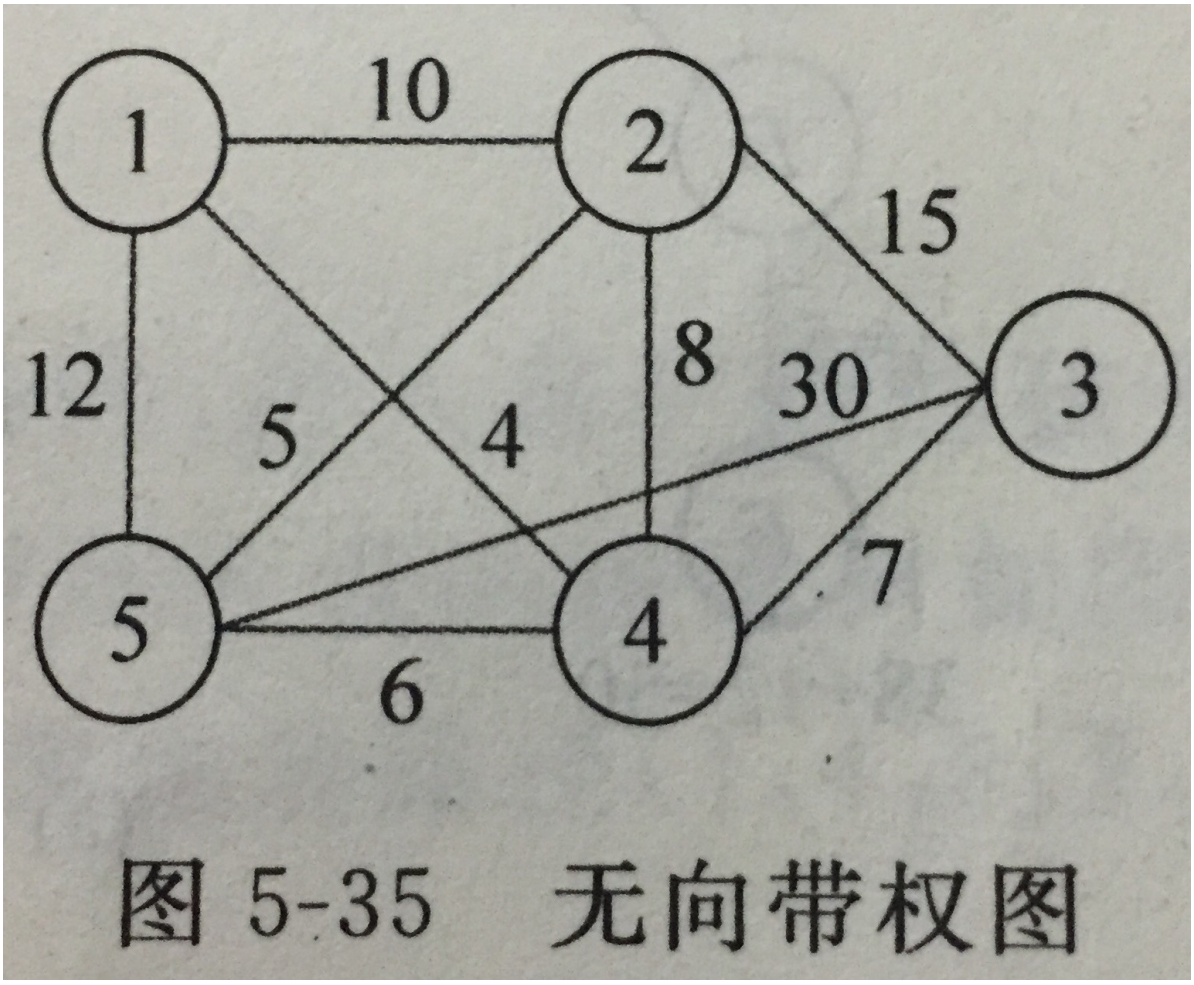


图 5-35　无向带权图

货郎问题

投资问题

m元钱，投资 n 个项目。效益函数 fi (x) 表示第 i 个项目投 x 元的效益(i =1, 2, ..., n)。求如何分配每个项目的钱数使得总效益最大?

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	0	0	0
1	11	0	2	20
2	12	5	10	21
3	13	10	30	22
4	14	15	32	23
5	15	20	40	24

输入：项目总数x；投资金额y；效益函数fi(k)
输出：最大利润maxProfit

投资问题

单源最短路径

dijkstra算法

最短路径

开灯问题

有n盏灯，编号为1……n，第一个人把所有灯打开，第二个人按下所有编号两倍的开关（这些灯被关掉），第三个人按下所有编号三倍的开关，以此类推，一共有k个人，问最后有哪些灯开着?

输入：n和k，输出开着的灯的编号，k<=n<=1000

分析见开灯问题

另一种更好的思路：

①被按了奇数次的灯泡是亮着的，被按了偶数次的灯泡是灭的
②可以发现，如果一个灯泡的编号具有偶数个因子，那么该灯泡就被按了偶数次，反之按了奇数次
③而只有完全平方数才有奇数个因子
④所以，原始问题可以转化为判断每个整数是否是完全平方数

币种问题

某单位给每个职工发工资(精确到元)。为了保证不要临时兑换零钱，且取款的张数最少，取工资前要统计出所有职工的工资所需各种币值(100,50,20,10,5,2,1元共七种)的张数。

	GZ\币值	100	50	20	10	5	2	1
z	252	2	1				1	
w	2686	26	1	1	1	1		1
	……							
total	2938	28	2	1	1	1	1	1

①将七种币值存储在数组B。七种币值就表示为B[i],i=1,2,3,4,5,6,7
为了能实现贪婪策略，七种币应从大面额的币种到小面额的币种依次存储
②用一个长度为7的数组S记录最终币值所需数量
③贪婪策略： 对每个人的工资，用"贪婪"的思想，先尽量多地取大面额的币种，由大面额到小面额币种逐渐统计。
④算法的时间复杂度：O(n)
币种问题

活动安排问题

n个活动E={1,2,..,n}，都要求使用同一公共资源。且在同一时间仅一个活动可使用该资源。
活动时间[si,fi), si为起始时间, fi为结束时间。si<fi
活动i和j相容: si>=fj或sj > =fi
求最大的相容活动子集合，尽可能多的活动兼容使用公共资源

①按结束时间非减序排序:f1<=f2 <= .. <= fn
②从第1个活动开始，按顺序放进集合A。放入活动i当且仅当与集合A中已有元素相容
与集合A中最后元素j比较：若si>=fj。则加入，否则不加入
fj=max(fk) ，k属于A，fj是集合A中的最大结束时间
③思想：选择具有最早结束时间的相容活动加入，使剩余的可安排时间最大，以安排尽可能多的活动。
由于输入的活动以其完成时间的非减序排列，所以每次总是选择具有最早完成时间的相容活动加入集合

A中。
④贪心算法并不总能求得问题的整体最优解。但对于这个问题总能求得最优解

活动安排问题

资源分配问题

设有资源n(n为整数),分配给m个项目,gi(x)为第i个项目分得资源x(x为整数)所得到的利润。
求总利润最大的资源分配方案，也就是解下列问题：
max z=g1(x1)+ g2(x2)+.....gm(xm)
x1+x2+x3+.....xm = n，0≤xi≤n，i=1,2,3,.....,m
函数gi(x)以数据表的形式给出。

例如：现有7万元投资到A，B，C 三个项目，利润见表,问题：求总利润最大的资源分配方案

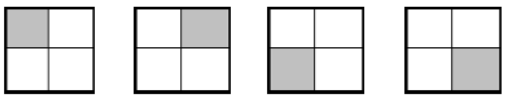
项 \ 额	1	2	3	4	5	6	7
A	0.11	0.13	0.15	0.21	0.24	0.3	0.35
B	0.12	0.16	0.21	0.23	0.25	0.24	0.34
C	0.08	0.12	0.2	0.24	0.26	0.3	0.35

①资源：n = 7(万元)；项目数：m = 3；gi(x)为第i个项目分得资源x所得到的利润。
②max z=g1(x1)+ g2(x2)+g3(x3)
x1+x2+x3= 7，0≤xi≤7，i=1,2,3
③划分阶段或找到子问题：每阶段增加一个项目，考察投资利润情况
④选择状态：每个阶段可以得到的最大利润;在第i阶段：前i个项目通过投资组合可得的最大利润；投资额不定：第i阶段总投资额为x时，设fi(x)为最大利润
⑤确定决策并写出状态转移方程
f1(x) = g1(x)
fi(x) = max{gi(xi) + fi-1(x-xi)}
gi(x):i项目投x后可得利润 fi(x):第i阶段投x的最大利润 0≤x≤n,0≤xi≤x

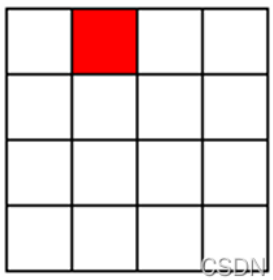
f1(x) = g1(x)
f2(x) = max{g2(x2) + f1(x-x2)}
f3(x) = max{g3(x3) + f2(x-x3)}

残缺棋盘(分治算法)

残缺棋盘是一个有2k×2k (k≥1)个方格的棋盘，其中恰有一个方格残缺。
下图给出k=1时各种可能的残缺棋盘，其中残缺的方格用阴影表示。
这样的棋盘称作“三格板”，残缺棋盘问题就是用这四种三格板覆盖更大的残缺棋盘。
在覆盖中要求：
1)两个三格板不能重叠
2)三格板不能覆盖残缺方格，但必须覆盖其他所有方格



1~4号

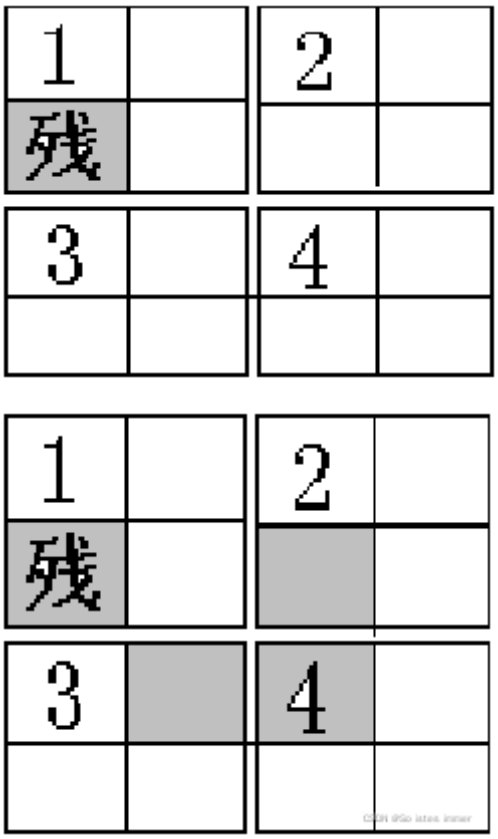


算法设计 （分而治之）

①以k=2时的问题为例，用二分法进行分解得到用双线划分的四个k=1的棋盘。

使用一个①号三格板(图中阴影)覆盖2、3、4号三个子棋盘的各一个方格，把覆盖后的方格，也看作是残缺方格，这时的2、3、4号子问题就是独立且与原问题相似的子问题

k=2其它情况也同理

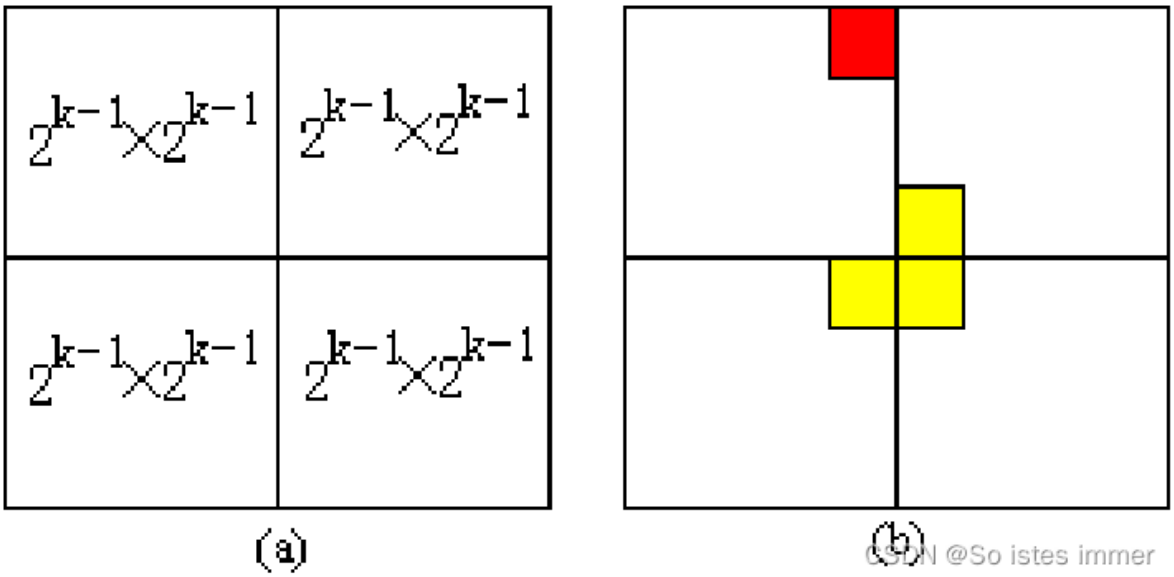


②推广至k=1,2,3,4,....

将2k×2k棋盘分割为4个2k-1×2k-1 子棋盘(a)所示。

特殊方格必位于4个较小子棋盘之一中，其余3个子棋盘中无特殊方格。

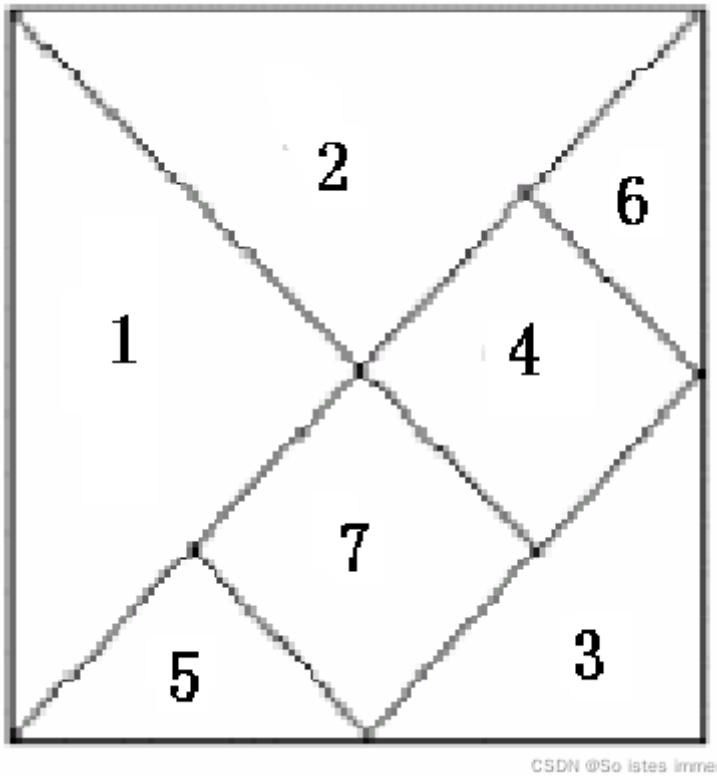
为将这3个无特殊方格的子棋盘转化为特殊棋盘，可用一个三格板覆盖这3个子棋盘的会合处，如 (b)所示，从而将原问题转化为4个较小规模的棋盘覆盖问题。



③ 递归地使用这种分割，直至棋盘简化为2×2的大小，以方便解决

图的m着色问题

有如下图所示的七巧板，试设计算法，使用至多4种不同颜色对七巧板进行涂色(每块涂一种颜色)，要求相邻区域的颜色互不相同，打印输出所有可能的涂色方案。



考虑：能否将七巧板转化为平面图？如何转化?
为了让算法能识别不同区域间的相邻关系，把七巧板上每一个区域看成一个顶点，若两个区域相邻，则相应的顶点间用一条边相连，这样该问题就转化为一个图的搜索问题。

①按顺序分别对1号,2号,.....,7号区域进行试探性涂色，用1,2,3,4号代表4种颜色。
涂色过程如下：
1)对某一区域涂上与其相邻区域不同的颜色。
2)若使用4种颜色进行涂色均不能满足要求，则回溯一步，更改前一区域的颜色。
3)转1)继续涂色，直到全部区域全部涂色为止，输出。
②区域之间邻接关系：邻接矩阵data[][]存储
涂色方案：数组color[]存储
图的m着色问题

8皇后（n皇后）

要在n*n的国际象棋棋盘中放n个皇后，使任意两个皇后都不能互相吃掉。
规则：皇后能吃掉同一行、同一列、同一对角线的任意棋子。
求所有的解。

①设n个皇后为xi，分别在第i行(i=1,2,...,n)
②问题的解状态：可以用(1,x1)，(2,x2)，.....，(n,xn)表示n个皇后的位置
由于行号固定，可简单记为：(x1,x2,...,xn)；
③ 问题的解空间：(x1,x2,...,xn)，1≤xi≤n(i=1,2,...,n)，共n^n个状态
④ 约束条件：n个(1,x1),(2,x2) ,... ,(n,xn)不在同一行、同一列和同一对角线上
原问题即在解空间中寻找符合约束条件的解状态

皇后