搜 索 基 础

浙江省镇海中学 xt

目录

1.穷举法

[例题 1] 光光的困惑

光光的妈妈给光光一篮鸡蛋，让光光拿去给外婆。光光很高兴地接过鸡蛋，向外婆家走去。途中，光光想：这一篮鸡蛋有多少个呢？于是，光光就开始一个个地拿出鸡蛋，数了起来：一个、两个……一不小心，蛋全倒翻摔碎了。光光大哭起来……

例题中，光光数鸡蛋的方法是穷举法，他依次取出鸡蛋然后数，也就是遍历了所有的情况。这样的方法，在其他的题目中，此种方法也是适用的。这是一种最基础的搜索，它的本身不需要用到高深的知识，却是搜索算法的核心。

搜索算法的核心就是遍历。而穷举具备了这一点。所以，在搜索找不出很好的方法时，穷举是我们的唯一法宝。

下面，我们讲一道经典习题的穷举算法。这是一种经典的穷举。

[例题 2] 砝码称重（NOIP T96-4）

设有 1g、2g、3g、5g、10g、20g 的砝码各若干枚（其总重<=1000），输出用这些砝码能称出的不同重量的个数，但不包括一个砝码也不用的情况。

最容易想到的方法就是枚举出有几个 1g，几个 2g，几个 3g……几个 20g，然后统计有几种不同的重量。用数组 w[1]~w[6]表示重量，q[1]~q[6]表示选择方案。算法描述如下（Pascal 语言）：

```
for q[1]:=0 to a1 do
  for q[2]:=0 to a2 do
    for q[3]:=0 to a3 do
      for q[4]:=0 to a4 do
        for q[5]:=0 to a5 do
           for q[6]:=0 to a6 do begin
             sum:=0;
             for i:=1 to 6 do sum:=sum+q[i]*w[i];
           end;
```

利用 6 个 for 循环可以算出总重量 sum，剩下的工作就是要判断 sum 是否已经出现过即判重。要实现这点很简单，注意到条件：其总重<=1000，可以开一个[0..1000]的 boolean 数组 H，设初值为 false,然后 H[sum]:=true，最后统计在 H[1..1000]中有几个 true 即可。

总结：此枚举算法着实弱智，但事实上此算法可以通过所有的测试数据，在比赛中使用可以省时省力。因此我们要改变印象中枚举算法是低效的观念，在没有方法时，枚举往往是突破口。（By 光光）

另注：此题的动态规划（Dynamic Programming）算法此处不予讨论。

[例题 3] 逻辑判断（TOJ1130）

小卡卡从 Pasca1 神庙附近的居民那里还了解到，远古时候的 Pasca1 圣地居住着三种居民：永远说真话的神，永远说假话的恶魔和在白天说假话，在夜里说真话的 Pasca1 人。小卡卡想，是否能根据远古居民的话来判断他们都是那种类

型的居民。

有五个居民 A，B，C，D，E。他们说的话只有 3 种:

1.I am [not] divine/evil/human.

2.X is [not] divine/evil/human.

3.It is day/night.

居民们说话的总数不超过 50。你的任务就是通过居民们说的话来判断他们的种类以及现在是白天还是黑夜。

这道题目是一道经典的穷举题。在这道题目中，我们首先可以计算出，我们需要穷举的东西——人们的属性和白天或黑夜。显而易见，五个人的属性与白天黑夜至多需要穷举 3*3*3*3*3*2=486，因为人数可能不到 5 个。而我们又不能想出更好的方法。于是我们就采用了穷举，作为这道题的正确算法。

在穷举上，我们有一个经典的“加一算法”来遍历。具体方法是这样的：

我们设五个人的属性分别有 0/1/2 三种情况，则我们可以用“三进制”计算得到 243 种情况，具体方法我们列几个：

具体方法说明	五人状态				
开始	0	0	0	0	0
末位加一	0	0	0	0	1
末位加一	0	0	0	0	2
进位	0	0	0	1	0
末位加一	0	0	0	1	1
末位加一	0	0	0	1	2
进位	0	0	0	2	0
末位加一	0	0	0	2	1
末位加一	0	0	0	2	2
进位再进位	0	0	1	0	0
……					

依照这样的方法，我们可以穷举每个人的状态，并且判断是否与供词符合。参考程序留给读者自己完成。

[例题 4] 数字和问题（宁波 2005 小学组）

已知 n 个整数 x1, x2, ..., xn，以及一个整数 k (k<n)。从 n 个整数中任选 k 个整数相加，可分别得到一系列的和。例如当 n=4, k=3，4 个整数分别为 3，7，12，19 时，可得全部的组合与它们的和为:

3+7+12=22　　3+7+19=29　　7+12+19=38　　3+12+19=34

现在，要求你计算出和为素数的共有多少种。

本题限定了数据规模，故可以很快判断出本题适用于穷举法。穷举法的常见思路有加一算法，本题即可以穷举出k个进行相加，判断是否为素数。素数n判断的一种方法是：由 2 穷举至$\sqrt{n}$，如果没有n的约数，则可以判断出n是素数，反之不能。利用穷举法，我们可以有一种直接的思路：枚举x[1]~x[n]是否要取，

如果达到共k个，则进行求和，并且判断是否为素数，统计。此种方法在数据规模较大的时候不是很好用，也就是判断次数在n大时增长迅速，属于指数级渐进时间复杂度，时间耗费大。时间复杂度O（2^n）。

[例题 5] 等差数列（ZJOI2004）

给定 n(1<=n<=100)个数,从中找出尽可能多的数使得他们能够组成一个等差数列.求最长的等差数列的长度.

我们首先必须对这个数列进行排序，排序我们采用 QuickSort。然后我们就能够很快想到一系列的方法：枚举头部、第二个与尾部。

但是我们为什么要枚举三个呢？头部当然不用说，第二个是为了与第一个作差，得出公差。然后就可以递推地去枚举到最后。

这样我们就能够得出一个$O(n^3)$的方法，而在n<=100 的时候保证不会超时。

[例题 6] 敲七、敲七版本 2（TOJ1006/1044）

1.输出 7 和 7 的倍数，还有包含 7 的数字例如（17，27，37...70，71，72，73...），给出一个整数 N,（N 不大于 30000），输出从小到大排列的不大于 N 的与 7 有关的数字，每行一个。

2. 给定正整数 N，求数列{An}的前 N 项。数列{An}满足如下条件:

1.该数列包含自然数集中所有能被 7 整除的数

2.该数列包含自然数集中所有个位上含 7 的数，如 17,27,177

3.该数列不包含自然数集中其他的数

4.该数列中的数有小到大有序排列

5.该数列中任意两项互不相同

给出一个正整数 N(N<=100,000),输出数列{An}的前 N 项,每项占单独一行。

1.这道题按照题意，我们可以求得最简单的方法：从 1 枚举到 N，对每一个数字进行判断。当然这是对这个数据来说最好的方法，其实还有一种方法是预处理，用一些方法记录下来（方法很多）。至于判断是否含有 7，我想到的是一位一位看，而 Maigo 大牛有个方法，就是转成字符串用 POS，这是不是个好方法呢？

2.理清题意，发现这是个简单题。读入 N 后，我们从 1 开始依次枚举，对每个数字进行 2 次运算，如果运算成功，就可以输出并记录个数，直到最后。

这个题目还有一种对于大数据的高效方法——合并有序链表。我们可以建立 2 个链表，然后生成 1..N 中符合条件的数字。因为这两个链表是有序的，因此用两个指针（静态、动态皆可）去合并。这样是最优效果。

[例题 7] 猫老大数（TOJ1081，猫老大与彩虹的竞赛）

猫老大很喜欢研究数字，特别是喜欢质数。一天，猫老大发现有一些数字可以表示成两个质数相乘的形式。比如，10＝2×5. 2，5 都是质数，所以 10 是一个“猫老大数 ”。所以猫老大决定考考彩虹，他告诉彩虹一个数 n ，判断 n 是不是“猫老大数”？（1<=n<=2^{31}-1）

由于题目中含有条件 1<=n<=2^{31}-1= maxlongint（LONG_MAX），所以我们可以先计算出 1 到$\sqrt{2^{31}-1}$的所有质数，共有 4792 个。求质数的方法在例题 4 讲过。于是我们把这些质数存在数组中，完成了第一步的预处理工作。

有这样一个定理：如果一个数是合数，那么它一定会有一个小于等于这个数的开方的不是 1 的因数。根据这个理和题意，则如果这个数是合数，我们只须要枚举 2 到$\sqrt{n}$就可以找到一个质数。

然后根据唯一分解定理，我们可以得出猫老大数必定只能质数分拆为两个质数。所以我们只须判断被一个质数除下后是不是质数即可。由于两个质数是成反比例的（n 一定），根据函数曲线，得出一种好的方法——先从小到大找到一个较小的，然后继续判断，直到找到第二个，判断剩下的是不是 1。

[例题 8] 勇闯黄金十二宫.....金牛宫（TOJ1001，黄金十二宫）

现在给出一个数字n，如果n可以表示成 2 个合数相乘的形式，则n被称作“牛数”，否则n被称作“弱数”。比如 16 = 4×4，4 为合数，所以 16 是一个“牛数”。阿尔迪巴告诉星矢很多数字，叫星矢判断这些数字是“牛数”还是“弱数”。（1<=n<=2^{31}-1）

其实这道题目与上道题目相反，有很大的相通之处。首先，预处理做法相同。其次，如果能够表示为两个合数的乘积，必定能够表示成为 4 个或以上的质数的乘积。我们用预处理的质数表求出质数的个数，然后判断是否大于等于 4 个就可以了。具体做法参看上题。

[例题 9] 石子合并（TOJ1017）

你有一堆石头质量分别为 W1,W2,W3...WN.(W< =100000)现在需要你将石头合并为两堆，使两堆质量的差为最小。规定 1<=N<=20，需输出合并后两堆的质量差的最小值。

这种题目有个直白的方法——加一算法。由于 1<=N<=20，我们就可以直接把这 N 堆用穷举的方法分成两堆，每一次都需要计算，然后求出最小值。最多每组数据需要计算 1048576 种情况。具体实现起来非常简单，我也就不再这里多说了。我的程序的运行时间大约是 13xxms，光光的大约是 15xxms，一般来说用这种方法枚举不会超时。

[例题 10] 反正切函数的运用（NOI2001）

反正切函数可展开成无穷级数，有如下公式

$$\arctan(x)=\sum_{n=0}^{\infty}\frac{(-1)^n x^{2n+1}}{2n+1} \quad (\text{其中 } 0\le x\le 1) \qquad \text{公式 (1)}$$

使用反正切函数计算π是一种常用的方法。例如，最简单的计算π的方法:

$$\pi = 4\arctan(1)$$
$$= 4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\cdots\right)$$

公式 (2)

然而，这种方法的效率很低，但我们可以根据角度和的正切函数公式:

$$\tan(\alpha+\beta)=\frac{\tan(\alpha)+\tan(\beta)}{1-\tan(\alpha)\tan(\beta)}$$

公式 (3)

通过简单的变换得到:

$$\arctan(p)+\arctan(q)=\arctan(\frac{p+q}{1-pq})$$

公式 (4)

利用这个公式，令 $p=\frac{1}{2},q=\frac{1}{3}$，则 $\frac{p+q}{1-pq}=1$，有

$$\arctan\left(\frac{1}{2}\right)+\arctan\left(\frac{1}{3}\right)=\arctan\left(\frac{\frac{1}{2}+\frac{1}{3}}{1-\frac{1}{2}\cdot\frac{1}{3}}\right)=\arctan(1)$$

使用 $\frac{1}{2}$ 和 $\frac{1}{3}$ 的反正切来计算 $\arctan(1)$，速度就快多了。

我们将公式 (4) 写成如下形式

$$\arctan(\frac{1}{a})=\arctan(\frac{1}{b})+\arctan(\frac{1}{c})$$

其中 a、b 和 c 均为正整数。

我们的问题是：对于每一个给定的 a（$1\le a\le 60000$），求 $b+c$ 的值。我们保证对于任意的 a 都存在整数解。如果有多个解，要求你给出 $b+c$ 最小的解。

首先我们对题目中给出的式子进行转化：

$\because \arctan\left(\frac{1}{a}\right)=\arctan\left(\frac{1}{b}\right)+\arctan\left(\frac{1}{c}\right)$，

$\therefore \arctan\left(\frac{1}{a}\right)=\arctan\left(\frac{\frac{1}{b}+\frac{1}{c}}{1-\frac{1}{bc}}\right)=\arctan\left(\frac{b+c}{bc-1}\right)$，$\therefore \frac{1}{a}=\frac{b+c}{bc-1}$，

交叉相乘得 $bc-1=a(b+c)$，化简得 $bc-ab-ac=1$，等号左右两边同时加上 a^2 得 $bc-ab-ac+a^2=1+a^2$，合并同类项得 $(b-a)(c-a)=1+a^2$，移项得

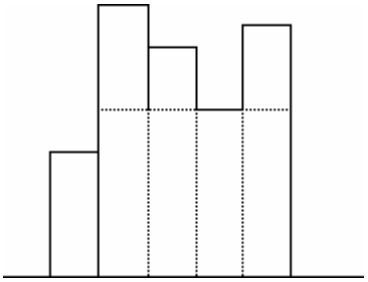
$c=\frac{1+a^2}{b-a}+a$。由于b、c次序无关紧要，所以我们不妨设$b\le c$，则有$b\le\frac{1+a^2}{b-a}+a$，

$b-a\le\frac{1+a^2}{b-a}$。由题意，显然得出$a<b$，$\therefore (b-a)^2\le 1+a^2$，$\therefore b-a\le\sqrt{1+a^2}$，

得到$b\le\sqrt{1+a^2}+a$，$\therefore b$取值范围为$\left[a+1,a+\sqrt{a^2+1}\right]$，枚举$b$即可。

[例题 11] 广告印刷（TOJ 一周年比赛）

最近，afy 决定给 TOJ 印刷广告，广告牌是刷在城市的建筑物上的，城市里有紧靠着的 N 个建筑。afy 决定在上面找一块尽可能大的矩形放置广告牌。我们假设每个建筑物都有一个高度，从左到右给出每个建筑物的高度 H1,H2...HN，且 0<Hi<=1,000,000,000，并且我们假设每个建筑物的宽度均为 1。N <= 1,000,000，要求输出广告牌的最大面积。



方法不难想到：如果要使涂刷的总面积是最大的，那么就必然有那么一个建筑物被刷满。那么哪一个建筑物能够被刷满才能够使总体涂刷的面积是最大的呢？沿袭上面的思路，我们穷举每一个建筑物，设它能够被刷满。于是我们可以用两个循环来枚举它的左方与右方各自能够延伸到哪里（也就是求出覆盖到哪里），然后这时的面积就可以用这个建筑物的高度*被覆盖建筑物的总个数，求每次求出来的面积中的最大值。

此题的路径压缩可以采取类似并查集的方法，实现起来也很简单。由于具体的实现与搜索关系不大，所以这里也就一笔带过了。

[例题 12] 仓库扩张（USACO Contest DEC05）

在 FJ 的农场里有 N（1<=N<=25000）个长方形仓库，这些仓库的四条边都分别与 x 轴、y 轴平行，且四角坐标均为正数，范围为 0..1000000。这些仓库都不互相重叠，但是可能有一些点或者一段边是它们共有的。

一次，FJ 得到了更多的奶牛去挤奶，因此他想扩张自己的仓库。一个仓库有能够扩张的条件是它不与别的仓库存在任何一个公共角或者任意一段公共边。请你计算有多少个仓库可以扩张。

这道题目枚举的方法有些特殊，需要一些枚举技巧。

首先在读入时，我们的处理方法是“拆边”，按横、纵两种拆。拆开以后，我们应该把所有的横边排序，所有的纵边也排序。

横边排序规则：1.按行排序。2.行相等时按这一行的头坐标（线段左边端点

纵坐标）排序。3.如果再相等就按线段右边端点纵坐标排序。这样就能够保证边的有序性，为下面做好铺垫。竖边类同。

然后就是把所有的横边、纵边进行重叠处理。我们可以把它们分开处理。如果重叠（或临接），则两个仓库都不能扩张。经过上述处理，我们可以发现我们需要比较的边数已经大大减少。因此我们可以加上一些细节处理来加快速度。

[例题 13] 行星队列（USACO Contest DEC05）

平面上给出 N（1<=N<=770）个点（坐标均为 1..15000 之间的整数），要求求出有多少组三个点在一条直线上。

这道题拿到后，首先我们会用枚举的方法，写出一个$O(N^3)$的程序。这可能可以过，但固然不是很好，在这里讲一下$O(N^2\log_2 N)$的方法。

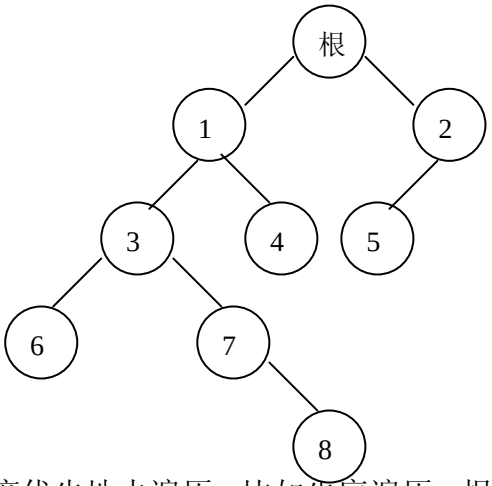
首先当然是读入操作。读入完毕后，我们这里用斜率来优化。所谓斜率，即为直线的倾斜角 α 的正切值。因为两点确定一条直线，故我们可以求出每两个点的直线的斜率。我们先用预处理，把斜率求出保存。当然，如求斜率时遇到除数为 0，则需另外处理（那就是与坐标轴平行的情况）。

然后把每一个点出发到所有点的直线按斜率排序。排序完毕，我们只需要开 i、j 二重循环去找这第三个点。因为两点确定一条直线，且每一点出发的直线都已经按斜率排序，故我们只需要用$O(\log_2 N)$的时间进行每一次查找。综合时间复杂度为$O((N^2)*(\log_2 N))=O(N^2\log_2 N)$。

2.深度优先搜索

深度优先搜索（Depth First Search），是一种常见的搜索方法。

我们在一般的搜索解题中，往往要找的是一棵答案树。比如某道题目中，我们有这样一棵答案树（文中是二叉的），我们要找到一条一条路径，使其能够到达节点 8。



我们可以深度优先地去遍历。比如先序遍历，根据它的递归定义，访问顺序为根-1-3-6-7-8，能很快地搜索出这样的一条路径。

下面我们来介绍一下回溯法。回溯法是一类适用广泛而形象的深度优先搜索算法。所谓回溯，就是“碰壁返回”。通俗的说，也就是你按照一定的顺序去产生一条路径，然后如果遇到了死路，就折回一部分然后试探下一条路径。每一次

生成的路径都需要经过判断是否到达最终目标。

还是以上一棵树为例，我们首先一路通畅，根-1-3-6，然后是死路。不符合，然后折回，到 3，然后试探右边 7，然后左边死路，右边 8 即为所求。

这样，我们就得出了一类问题的通用解法——深度优先搜索。这种方法由于答案树的递归定义，一般用递归来解决。

[例题 1] 四色图问题（宁波 2005 高中组）

在绘制区域地图时，要求以国家为单位着色，所有相邻国家着不同颜色。试编写一程序实现如下功能：使用至多四种不同颜色对地图进行涂色(每块涂一种颜色)，要求相邻区域的颜色互不相同，输出所有可能的涂色方案总数。国家数小于等于 10。

本题可用深度优先搜索来解决。

首先，我们读入的是一个**邻接矩阵**。由于国家数小于等于 10，所以我们最多要搜索 $4^{10}=1048576$ 次，保证了不会超时，所以我们在接下来的搜索过程中，可以用回溯的方法枚举，进行相邻节点是否同色的判断即可。搜索的时候，如果当前点与相邻点的颜色相同则回溯，否则继续深入搜索，如果找出一种方案，则记录并回溯。

流程是这样的：

1.当前节点为 1；
2.当前节点的颜色代号加 1；
3.如果当前节点不是 1 则与相邻节点判断是否相同
4.　　是：回溯
5.　　不是：如果当前节点为 n
6.　　　　是：记录并回溯；
7.　　　　不是：搜索下一层；
8.直到回溯到第 0 个国家为止

[例题 2] 外星生命（TOJ1062）

地球上的科学家收到了来自外星的信号：000023*000011=002093，科学家猜想这是某个外星人的年龄。但有人指出，这些外星人好像不怎么聪明，因为 23*11=253，而非 2093。但是科学家们发现，如果把 000011 改成 00091 的话算式就成立了。他们认为这是接收信号的时候出了差错的缘故。

现在给你这样一个算式，问最少改动几个数字就能使得算式成立？（格式是??????*??????=??????，忽略进位）

我们可以按位搜索，按照 DFS 的顺序，求出一种方案。然后本着动态规划中的最优原理，以这个答案为基础，把新的分支搜索进行下一位的搜索。

在这里我们引出**剪枝**的概念。在本题中，我们在第 n 步就已经比现有结果更差（需要更多的改进），这是我们就可以不用搜下去了，我们就退出本轮搜索，这个称为剪枝，我们剪的是答案树的枝。

在搜索中，剪枝一般都是很重要的，如果没有剪枝，我们就要搜索很多的节点，做很多的无用功以浪费时间，而剪枝避免了非常多的。不必要的搜索，大大提高了搜索的速度。所以，剪枝在搜索中的地位是非常高的。如果学好了基础的

搜索方法，再去掌握一定的剪枝，就能够用好搜索了。

[例题 3] 数的划分、放苹果（NOIP T2001-2、POJ1664）

数的划分: 将整数 n 分成 k 份，且每份不能为空，任意两种分法不能相同(不考虑顺序)，例如：n=7，k=3，下面三种分法被认为是相同的：1，1，5；1，5，1；5，1，1。问有多少种不同的分法。

放苹果: 把 M 个同样的苹果放在 N 个同样的盘子里，允许有的盘子空着不放，问共有多少种不同的分法？（用 K 表示）5，1，1 和 1，5，1 是同一种分法。
1<=M,N<=10

数的划分：很清楚，这是一道整数分解的问题。这种分解，较为直接的思路是**递归**。我们在本题可以采用深度优先搜索的方法。

我们定义一个过程，使其反复递归穷举第 1 份、第 2 份……第k份，然后寻找出可行的路径，时间复杂度$O(n^k)$。这种方法思路十分便捷，但也许会超时。

我们有两个很好的剪枝：

剪枝 1 如果剩余的够不上最小的则剪去。这是显而易见的常理，但是可以加快速度。**剪枝 2 枚举到剩余 1 个盘子时判断是否可行。**也是加快速度的方法，使在题目所描述的 n 和 k 的范围之中完全可行。

重复的只算一种，我们怎么样处理有关重复的呢？我们可以引入一个参数 min，作为至少每一个要达到 min，顺序由小而大，逐层递归。

综上所述，我们可以轻易写出一个优化过的递归搜索程序。

放苹果：我们只需要改变上题的一个小条件，把第一个盘子的最小搜索值设为 0。则就是从 0 搜索起，这个题目也迎刃而解了，毕竟几乎一样的题目～

[例题 4] 跳棋的挑战（USACO Training 1.1.4-1）

检查一个如下的 n×n 的跳棋棋盘，有 n 个棋子被放置在棋盘上，使得每行，每列，每条对角线(包括两条主对角线的所有对角线)上都至多有一个棋子。

下面的 6×6 的布局可以用序列 2 4 6 1 3 5 来描述，第 i 个数字表示在第 i 行的相应位置有一个棋子:

	1	2	3	4	5	6
1		○				
2				○		
3						○
4	○					
5			○			
6					○	

行号 1 2 3 4 5 6

列号 2 4 6 1 3 5

这只是跳棋放置的一个解。请遍一个程序找出所有跳棋放置的解。并把它们以上面的序列方法输出。解按字典顺序排列。请输出前 3 个解。最后一行是解的总个数。(6<=n<=13)

这是一道在 USACO 中经典的题——Checker Challenge。在这题中，搜索的优化和一些特性的运用显得十分重要。

分析这道题，我们首先看来就是一个经典的 N 皇后问题。还有的是这道题先定了算法，只能采用回溯算法。N 皇后有它的构造方法，但是不适用。所以我们来看如何就此问题运用最一般的方法求解。

我们对每一个限定条件进行分析。由于每一行都不能有两个跳棋，而跳棋有 N 个，所以每一行必有一个跳棋。由于它是要求每一行中跳棋的列，所以我们只需搜索每一个行就可以了。根据分析，横轴我们已经不需要考虑，纵轴一共有 N 条，＼型轴 2N-1 条，／型轴也有 2N-1 条。

然后我们可以递归求解。如果符合条件，就可以把这个棋子放上去，三条轴都须记上；当回溯的时候，就必须把它撤除，然后求得总数。

但是这种方法对于 N=13 时有致命弱点——慢！经过测试，大约需要 2 秒多才能出解。这个在时间上是承受不了的。这道题有个特性是值得我们利用的，那就是对称性。也就是说，需要搜 N，我们只需搜 N 的一半。比如 13，我们只需搜到 6，然后根据对称性把搜得的结果乘以 2，然后再加上搜到的 7 的结果。这样就轻松剪掉大约一半。

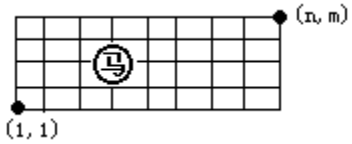
然后我们就有了解题的步骤：先搜到三个解—不记状态重新搜—输出。

这道题还有一个特性，那就是旋转。通过旋转还可以剪掉四分之一，但是似乎很麻烦，所以笔者没有去试验过。

关于此题的其他方法，详见其他书籍（比如**启发式修补法**《算法艺术与信息学竞赛》P195）。

[例题 5] 骑士的游历 1、2

设有一个 n*m 的棋盘，如下图，在棋盘上任一点有一个中国象棋马，



马走的规则为：1.马走日字 2.马只能向右走

即如下图所示：

任务 1: 当 N,M 输入之后,找出一条从左下角到右上角的路径。**(NOIP T97-3)**

任务 2: 如果是 n*n 的棋盘，取消马只能向右走的条件，找出一条路径使马能够不重复地遍历每一个点。**(《竞赛例题解析》)**

1.题目也就是说，马能够朝着四个方向前进。而题目要求的是一条路径，所以这道题是经典的深度优先的回溯算法。我们把四个方向用数组记录下来，然后就可以开始先用第一种方向生成一条路径，然后回溯，直到有一次点停留在目标顶点为止，输出路径。本题需要注意的是越界的判断与处理。

2.现在马摆脱了束缚，可以朝着八个方向走了。我们可以上小题所用的方法来处理这个问题。在这个问题中，我们所要研究的不仅仅是到达一个点，而是遍历棋盘。但是用哈希表判断是否到过的记录很麻烦，所以我们只要记录步数就可以了，直到走过 n^2-1 步为止。我们回溯之前，要先建立一张空棋盘，然后判断

要达到的是否是已经到过的，如果是则回溯，反之记录下来。最后输出这张棋盘就可以了。

[例题 7] 卫星照片（USACO Contest NOV05）

FJ 给他的农场买了 W*H 像素的卫星照片，希望找出最大的“连续的”（互相连接的）牧场。任何一对像素，一个像素如果能横向的或纵向的与属于这个牧场的另一个像素相连，这样的牧场称作是连续的。

每一张照片都数字化的抽象了，牧场区显示为“*”，非牧场区显示为“.”。下面是一个 10*5 的卫星照片样例:

..*.....**
.**..*****
.*...*....
..****.***
..****.***

这张照片显示了大小分别为 4、16、6 个像素的连续牧场区。帮助 FJ 在他的每张卫星照片中找到最大的连续牧场。

在这里，我们介绍一种方法，称为 FloodFill。

FloodFill 是一种简单的染色法，通常译为种子填充法，一般用来求连续区域的面积等。我们在这里先介绍一下这种方法。

FloodFill 可以有 2 种方法实现。

1. DFS。
   使用普通 DFS 架构。先从一个点出发，向相连的方向（一般是四周）探寻，如果四周有相连的点，则继续递归探查下一个点的四周方向。不过这个需要避免的是搜索的重复，避免陷入死循环。
2. BFS。
   使用普通 BFS 架构（下面会讲）。从一个点出发，向相连的方向扩展节点。这种方法有效避免了搜索的重复，且 BFS 适用范围较广（比如限制步数等），是可以推荐的。但是 BFS 编程比 DFS 略繁。

在这道题中，我们采用 DFS 来实现 FloodFill。

我们所需要做的一点细节处理是，每一个牧场区搜索到之后，就可以擦去，以避免搜索重复和死循环。

[例题 8] 房间问题（IOI94）

图示出了一张建筑平面图，编程计算

1、该建筑中有多少个房间;

2、最大的房间有多大;

3、拆除建筑中的某一堵墙，以形成一个尽可能大的房间。指出该墙。

该建筑分成 m*n 个方块（m≤50，n≤50），每个方块可有 0~4 堵墙（0 表示无墙）。

这道题也是典型的 FloodFill。

这里的 FloodFill 也是可以用 DFS/BFS 做的。为了避免重复计算，我们可以用一个二维数组记录是否探查过这个地方，分成的是第几个房间。

我们在 FloodFill 中就可以完美解决第 1、2 问，接下来就是第 3 个问题了。

我们可以枚举每一条边，来试验是否拆除的面积比原先记录的大（两边面积之和）。这里我们只需要用到穷举。

我们在这里能够采用 FloodFill 作为架构。试想，如果把 FloodFill 推广到无向图、有向图呢？有兴趣的读者可以自行研究，此处不再赘述。

[例题 9] 谷仓安全保护（USACO Contest NOV05）

FJ 给谷仓安装了一个新的安全系统，并且要给牛群中的每一个奶牛安排一个有效的密码。一个有效的密码由 L（3 <= L <= 15）个小写字母（来自传统的拉丁字母集'a'...'z'）组成，至少有一个元音（'a', 'e', 'i', 'o', 'u'），至少两个辅音（除去元音以外的音节），并且有按字母表顺序出现的字母(例如，'abc'是有效的，而'bac'不是）。

给定一个长度 L 和 C 个小写字母，写一个程序，打印出所有的长度为 L、能由这些字母组成的有效密码。密码必须按字母表顺序打印出来，一行一个。

这道题可以用比较基础的 DFS 回溯解决。

我们可以用一个字符串来当作答案数组（一个比较巧妙的方法），然后层层递归产生新的答案。当深度到达 L 时，判断是否满足元、辅音的要求，再输出。然后回溯（PASCAL 中每一次回溯要去掉“尾巴”，C/C++由于语言优势不必去除），以避免重复运算浪费时间。

这道题具体的搜索方法与前面我们所讲的一些题目别无二样，可见搜索算法的通用性很强。由于这道题比较简单，在此也不详细说明了。

[例题 10] 水碗（USACO Contest JAN06）

有 20 个碗，正反状态用 0、1 表示（正=0，反=1），给出一组碗的状况，要求用最少的操作次数使碗全部朝正面。一次操作就是以一个碗为中心，翻转它的左边、右边以及它自己。（如果一个碗的左（右）边没有则不需要翻转这个碗的左（右）边）

这道题拿到，需要一些敏锐的观察力。我们可以进行一些位处理。首先，我们可以把 20 个碗的 0、1 值从二进制转化为一个长整数。可以发现，这个长整数的范围在 0..1048575 之间，因此共 2^{20}种状态。这为我们下文的搜索做好了铺垫。

我们转换完毕，然后我们考虑怎样从一个状态值翻动以产生新的状态值。我们考虑多种位操作。其中异或（Pascal 中是 xor，C/C++中是^）操作无疑是最好的。我们可以举个例子来说明怎么样翻转。

就以样例数据作为例子。我们把原状态转化为二进制。

0	0	1	1	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	3484

然后，我们假设要以左起第三个为中心翻转。则做异或运算：

0	0	1	1	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	3484
0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14
0	1	0	0	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	3474

是不是很神奇呢？我们可以把每一个点为中心翻转的异或操作的数保存起来。产生规则：

$$Mask[i]=\begin{cases}3（i=1）\\7（i=2）\\Mask[i-1]*2（3<=i<=19）\\786432（i=20）\end{cases}$$

万事俱备，我们考虑怎么样搜索。我们首先想到的可能是 BFS。但是考虑到**以每一个点为中心翻 2 次等于不翻**，因此我们考虑到 DFS 可能也可以。具体框架不再叙述。

我们分析一下它们各自的时间效率。

DFS是稳定的，因为它不能直接求出最优解，因此我们必须遍历搜索树。每一次运算稳定于运算 2^{20}次。（平均时效 0.077s）

BFS时间效率差大，最少它一次就能够求出解，但是在队列中每一个节点必须先扩展 20 次（也就是做 20 次运算），然后才能判重。因此当最不利情况下，BFS时效低，至多运算可达到 $20*2^{20}$次。根据计算得最不利时效可达到 1.54s。

而时限是 1s 的，但是极端数据也估计是不存在的。（处理得当的 BFS 对于最大数据是 0.983s，处于危险边缘）我们经过如此的分析，还是推荐大家采用 DFS。

习题：

1. *《文件匹配》（NOI97）（*）*
2. *《智慧珠游戏》（NOI2005）（*）*
3. *《算符破译》（NOI2000）（**）*

3.广度优先搜索

广度优先搜索（Breadth First Search），是一种与深度优先搜索不同的搜索。这种搜索方法的通性是一般出现在最少多少步的情况。这种方法由于它搜索顺序的特殊，使广度优先搜索能够保证搜到的是最小的。

还是上述的一棵树，比如我们要求找到 5 的最少步数。我们按层遍历，找的顺序是根-1-2-3-4-5，是 3 步。而 DFS 的顺序则是根-1-3-6-7-8-4-2-5，需要遍历一圈才能够确定最少步数，即使找到了 5 也需要遍历整棵树。也就是说，BFS 其实是一种找出最优解的方法，而 DFS 则不是——只能找出一条路径，但不能保证最优。

但是 DFS 能够以递归的方式或者别的方式轻松进行（不包括递归死循环状态），而 BFS 似乎实现麻烦。我们可以以一种很常见的方法去实现它，那就是队列。我们可以从头扩展，按照队列 FIFO（First In First Out）的顺序加入到队尾。BFS 一般需要判断重复，如果是重复则不添加如队尾，这样有时候可以减少处理的数据。判断重复一般有 2 种方法：从头搜节点与 HASH 表。前者时间耗费巨大，后者空间耗费巨大。

判重、存节点导致了 BFS 有一些巨大的缺点。但是 BFS 不失为一种优秀的搜索方法，我们用例题来证明。

[例题 1] 救援行动（T0J1051 By Angel ForYou）

Angel 被传说中神秘的邪恶的 Moligpy 人抓住了！他被关在一个迷宫中。迷宫的长、宽不超过 200。迷宫中有不可以越过的墙以及监狱的看守。Angel 的朋

友带了一些救援队来到了迷宫中。他们的任务是：接近 Angel。我们假设接近 Angel 就是到达 Angel 所在的位置。假设移动需要 1 单位时间，杀死一个看守也需要 1 单位时间。到达一个格子以后，如果该格子有看守，则一定要杀死（否则会死的很难看的……只见那个看守开了 9 倍狙镜……）。交给你的任务是，最少要多少单位时间，才能到达 Angel 所在的地方？（只能向上、下、左、右 4 个方向移动）

本题有两种方法：由救援队出发去搜与由 Angel 出发倒找救援队（这变成谁救谁了？似乎救援队被困^_^）。这两种方法的好坏不能比较（完全看数据），只是由于数据的关系（传说有数据错误，有两支救援队），我们只能采用后者的方法。

我们可以首先根据数据建图，然后从 Angel 出发，向四个方向探索，然后建队列。由于位置一共只有 200*200=40000 种，所以我们只需要有数组模拟链表，态做一个队列，然后由列表扩展。

应该说本题不难，但是请采用由 Angel 出发倒找救援队的方法才能 AC。（这个是不是给我们提供了反向 BFS 的一种思路呢？）

[例题 2] 瑰丽华尔兹（NOI2005）

题目太长，略。请到http://www.noi.cn看题目。

本题的标准做法是动态规划（Dynamic Programming），效率很高，但是这里我们要介绍的是 BFS 算法。

算法的实质是求出每一个时间内是否走动而求出最长的不违反规定的时间。所以我们可以用宽度优先来求出。作者比较推崇使用链表。我们可以新建一个队列，做出起始，然后依次往后推出一步。由于没有无解，故可以不必判断重复，其实并没有重复不重复可言。这种方法不是很好，算法时间复杂度是 O(2n)，只能过掉 1 组数据。

我不推荐这种做法，若要正确解此题，请使用**动态规划**。

[例题 3] 倒水问题（经典问题）

有 2 个没有刻度的杯子，容积分别是 V1、V2，另有一个无限大的水缸，里面有无限多水。对这两个杯子可以进行如下操作:

1、从水缸里往一个杯子加满水;

2、把一个杯子里的水全部倒进水缸;

3、从一个杯子往另一个杯子里倒水，直到另一个杯子满或一个杯子空为止。

现在我们需要通过一定顺序的操作，使杯子 1、杯子 2 或杯子 1+杯子 2 中的水的体积是 V3。（V1、V2、V3，0＜V1、V2＜2^7，0＜V3＜2^8）

这道题很经典的算法是 BFS。我们必须从开始状态推出后面状态。

有每一种状态，可以得出 6 种状态：（1）甲杯装满水；（2）乙杯装满水；（3）甲杯倒空水；（4）乙杯倒空水；（5）甲->乙；（6）乙->甲。其中（5）（6）必须判断是一个先倒空还是一个先倒满。

在这里面只有 128*128=16384 种状态，所以我们设计 HASH，为

P(I,J)=128*I+J

于是我们用静态数组模拟队列，用 HASH 来进行判断，来进行 BFS。

[例题 4] 麻将游戏（SGOI）

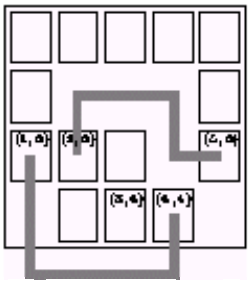
在一种"麻将"游戏中，游戏是在一个有 W*H 格子的矩形平板上进行的。

每个格子可以放置一个麻将牌，也可以不放（如图所示）。玩家的目标是将平板上的所有可通过一条路径相连的。两张相同的麻将牌，从平板上移去。最后如果能将所有牌移出平板，则算过关。

这个游戏中的一个关键问题是：两张牌之间是否可以被一条路径所连接，该路径满足以下两个特性：

1. 它由若干条线段组成，每条线段要么是水平方向，要么是垂直方向。
2. 这条路径不能横穿任何一个麻将牌（但允许路径暂时离开平板）。

这是一个例子：



在（1，3）的牌和在(4，4)的牌可以被连接。(2，3)和(3，4)不能被连接。

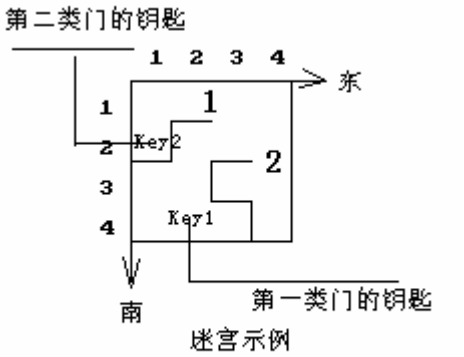
你的任务是编一个程序，检测两张牌是否能被一条符合以上规定的路径所连接。

这道题是不难的 BFS。

读入数据后，我们可以先构造两个外框——内外框是可以通行的，最外框是不可通行的。这样简化了问题本质。然后有一种方法——一路向外直接往外扩展（似乎可以节省时间？），往队列后添加节点。这道题当然也要构造 HASH，但是方法与上题相同。抓住了问题的本质，问题也就轻松解决了。

[例题 5] 拯救大兵瑞恩（CTSC99）

1944 年，特种兵麦克接到国防部的命令，要求立即赶赴太平洋上的一个孤岛，营救被敌军俘虏的大兵瑞恩。瑞恩被关押在一个迷宫里，迷宫地形复杂，但是幸好麦克得到了迷宫的地形图。



迷宫示例

迷宫的外形是一个长方形，其在南北方向被划分为 N 行，在东西方向被划分为 M 列，于是整个迷宫被划分为 N*M 个单元。我们用一个有序数对(单元的行号，单元的列号）来表示单元位置。南北或东西方向相邻的两个单元之间可以互通，或者存在一扇锁着的门，又或者存在一堵不可逾越的墙。迷宫中有一些单元存放着钥匙，并且所有的门被分为 P 类，打开同一类的门的钥匙相同，打开不同类的门的钥匙不同。

大兵瑞恩被关押在迷宫的东南角，即（N,M）单元里，并已经昏迷。迷宫只有一个入口，在西北角，也就是说，麦克可以直接进入(1,1)单元。另外，麦克从一个单元移动到另一个相邻单元的时间为 1，拿取所在单元的钥匙的时间以及用钥匙开门的时间忽略不计。

你的任务是帮助麦克以最快的方式抵达瑞恩所在单元，营救大兵瑞恩。

这道题目的 BFS 类型与往常所做的 BFS 类型大有不同，它属于典型的比较简单的**分层图 BFS**。由于钥匙最多有 10 种，一把钥匙有得到和没有得到这两种情况，所以我们可以得到这张图最多有 1024 种情况，也就是把这张图分成了 1024 层，在不同的情况下，我们要访问不同的图。由于一层图最多有 15*15=225 个节点，所以时空上面应该说问题不大。

首先我们根据题目的意思，按照输入文件的格式（按原题）去构造一张迷宫图。我们采用边读边构造的方法。我们设

$$g[x,y,d]=\begin{cases}0 & (x,y)\text{与它 d 方向的格子互通}\\ -1 & (x,y)\text{与它 d 方向的格子不通}\\ t & (x,y)\text{与它 d 方向的格子之间有 t 类门}\end{cases}$$

这样可以方便我们构造迷宫地图，具体的构造方法就不详细叙述了。构造之前，我们为了处理方便还通常在最外圈加上一圈围墙，方便判断越界的情况。

然后我们可以读入钥匙的坐标，并另外开一张地图记录钥匙存放情况。然后这道题就已经转化为普通的 BFS 了，但是在访问节点的过程中遇到了钥匙要及时地转化到另外的一张图上去，记得此时携带的钥匙是不同的。

[例题 6] 补丁 VS 错误（CTSC99）

一个程序总有个错误，公司经常发布补丁来修正这些错误，遗憾的是，每用一个补丁，在修正某些错误的时候，同时会加入某些错误，每个补丁都有一定运行时间。某公司发表了一个字处理软件，出现了 n 个错误 B={b1,b2,b3,.....bn}，于是该公司发布了 m 个补丁，每个补丁的应用都是有条件的（即哪些错误必须

存在，哪些错误不能存在）。求最少需要多少时间可全部修正这些错误。（1<=n<=20，1<=m<=100）（题目叙述采用 TOJ1275 的叙述）

这道题目拿到后，我们首先想到的是动态规划的方法。仔细分析以后，就会很容易发现，这道题目的叙述使者到题目的本质不符合动态规划的最优化递推原理，所以不能使用动态规划。由于这道题目求的是最少所需要的时间，所以我们可以使用的唯一方法就是广度优先搜索。

这道题中，我们先要对每一个补丁的适用环境与使用效果进行处理。我们要把适用环境、使用效果中的“+”与“-”分别放在四个数组里，以便于今后的使用。然后就是普通的 BFS 了。我们用新建一个链表当作搜索队列。则当出现 2 种情况的时候搜索才能够结束：1.错误集合为空，说明错误已经修复；2.队列为空，表示错误不能被修复，也就是无解。

这道题队列结点扩展也特别的有讲究。我们在插入节点的时候，有四种情况：

1.如果有一个节点的错误集合与扩展节点的错误集合相同，且那个节点的所用时间比当前的扩展节点少，则当前节点不需要入队；

2.如果有一个节点的所用时间比当前的扩展节点少或相等，则继续搜索下一节点；

3.如果有一个节点的所用时间比当前的扩展节点多，则把新的节点按照时间顺序插入队列，并且用指针操作删除在此之后的所有的错误集合与新节点的错误集合相等的节点，因为这个节点更优。

4.如果队列中所有的节点都比当前的扩展节点少或相等，则把新的节点加入在队列的末尾。

然后就可以按照普通的 BFS 的方法，来进行搜索。

[例题 7] 穿越封锁线（OIBH20051113《抗日英雄传》）

已知有一个有向图，有n个节点和m条边。在一个长度为k（0<=k<=10）的周期内每一分钟都可能有不能访问的边，求从节点 1 到节点n的最短时间消耗（每个时间段内可以选择停留）。

这道题目我们不能使用图论的最短路算法。因此根据题目叙述，我们可以选择 k 层的分层图 BFS。下面我来叙述一下具体的方法。

我们可以新建一个地图 map，map[i,j]来表示从 i 到 j 是否有一条边。因为每一分钟可以停留，为了处理方便，我们给每一条 map[i,i]连通。

然后，我们就可以运用 HASH 表判重。我们只需开最大 100*100*10=100000 个元素的布尔数组，来判每一层图的这个节点是否访问过。然后这样处理，就使整个程序变成了一个简单的 BFS，我们只需采用标准 BFS 算法即可轻松求得答案。

[例题 8] 最后的战犯（OIBH20051113《抗日英雄传》）

Feli 来到岩洞入口，发现岩洞其实是一个巨大的迷宫。迷宫地形极为复杂，为一个正方形，其中布满了障碍物。迷宫可以分为 N*N（2≤N≤100）个区域，每个区域或者是空地，或者是不可逾越的障碍物。小犬就躲藏在其中某一个区域内。由于小犬已经忍受了几天的饥饿，Feli 进入迷宫时他已经失去思维处于迷乱状态。小犬每秒钟只会沿着他的方向直线前进，如果遇到障碍物或者迷宫边界，

他会立刻向右转 90 度（不会花去时间），继续沿直线前进（初始方向向北）。Feli 每秒钟可以自主决定往哪个方向走。如果同一时刻 Feli 与小犬位于同一个区域，或者相邻的区域（非对角线相邻），Feli 可以立刻将小犬抓住。**（节选）**

关于这道题目，我们很快就可以想到一种算法，就是把从所有节点到所有节点的最短路径算出来。然而，这样需要 100*100*100*100=10^8 的内存。

但是我们可以有一点优化。由于 Feli 到达所有点的最短距离都是从他所在点开始的，因此我们只需要求出单源→全图的最短路径，使用 BFS 应该是一个不错的选择。于是，算法所需要的内存就成功地降为 100*100=10000 了。

由于 Feli 可以到任意一个点的距离最大只有 10000，然后，我们只需要模拟 10000 步小犬的动作即可。然后我们只需要判断每一步小犬有没有可能被 Feli 捉住。

需要有一点注意的是：小犬有可能一开始就被困在一个地方，四面不得动弹。因此，如果有一个地方小犬转 4 次向都不能行动，应该及时准确判无解。

[例题 9] Ni 骑士（USACO Contest DEC05）

给出一张 W*H（1<=W,H<=1000）的地图，每个位置都标有 0..4 的数字。其中 0 表示可以通过，1 表示不能通过，2 表示 Bessie 的起始位置，3 表示骑士起始位置，4 表示树丛。现在要求求出一条最短的路，使骑士经过树丛至少一次，且能够到达 Bessie 所在位置。

这道题目其实非常简单。我们可以以 Bessie 起始点为起始位置和以骑士起始点为起始位置分别做 BFS，求得他们到达地图中各个点的最少时间 b[i,j]和 k[i,j]。然后，根据原题，我们易得出他们的最短路径是骑士→某一棵树丛→Bessie。因为在原题中无向，故点 A 到点 B 的距离等于点 B 到点 A 的距离。因此我们可以枚举中继点——树丛。枚举整张地图，若一个点是树丛，则计算如果在[x,y]这棵树丛会合所需要的最短时间=b[x,y]+k[x,y]，然后更新答案。

查了一下 SACO（The South African Computer Olympiad）原题，发现这道题限时 8s，是因为它需要输出路径。因为在哪个树丛会合我们已经求得，所以这时我们可以再用一次 BFS，搜出路径然后输出即可。

习题：

4. *《镜子迷宫》（TOJ1074）*
5. *《聪明的打字员》（NOI2001）（*）*
6. *《聪聪和可可》（NOI2005）（*）*

4.双向广度优先搜索

我们还是在上述条件中找 5，求最少的步数，前提是这棵搜索树已经告诉你。我们就可以使用双向广度优先的算法，具体的方法是：

1. 由根节点扩展 1、2，存在队列 1 中；
2. 由节点 5 倒找上来，反向倒循，得到 2；
3. 则发现 2 已经在队列 1 中，搜索完成。

双向广度优先搜索有很多限制，前提是知道开始和结束的状态，与是否有逆推性有很大关联。具体使用方法无异于两次BFS（只是一顺一倒，先一步后一步）。但是这样效率的差别是很大的。例如二叉树，需要n步搜到。单向BFS是$O(2^n)$的，而双向BFS是$O(\sqrt{2^n})$的。

[例题 1] 九数码问题（ZJOI2005）

这是一个很古老的游戏了：有一个 3 × 3 的活动拼盘（如下图），方格上写有 0~8 这九个数字。利用拼盘背后的旋钮，游戏者每次可以进行以下两种操作之一：1.将拼盘外围的 8 个方格按顺时针挪一个位置。2.将中间一行向右移动一个位置，最右边的方格被移到最左边。给你一个拼盘的初始状态，你能用最少的操作次数把拼盘变成下图所示的目标状态吗？

0	1	2
3	4	5
6	7	8

这道题显然是使用 BFS。这道题选择 BFS 的同时，需要考虑到双向 BFS 的存在。这道题应该采用双向 BFS+HASH。HASH 的方法请自行设计，或参看我的另一份解题报告。至于双向 BFS，方法应该也是与 BFS 相近的，整个方案并无技巧性可言，但是纯靠 HASH 的帮助。

[例题 2] 字串变换（NOIP T2002-2）

已知有两个字串 A$, B$ 及一组字串变换的规则（至多 6 个规则）:

A1$ -> B1$

A2$ -> B2$

规则的含义为：在 A$ 中的子串 A1$ 可以变换为 B1$、A2$ 可以变换为 B2$ …。

例如：A$ = 'abcd'，B$ = 'xyz'

变换规则为：‘abc’->‘xu’，‘ud’->‘y’，‘y’->‘yz’

则此时，A$ 可以经过一系列的变换变为 B$，其变换的过程为：

‘abcd’->‘xud’->‘xy’->‘xyz’

共进行了三次变换，使得 A$ 变换为 B$。

这道题的正确方法是双向 BFS。这道题也不难，可以由 A$出发去替换，把正向队列中的串中的一段 Ai$替换成为另一段 Bi$，然后加入正向队列，并且判断是否重复和是否在逆向队列中。由于它的目标是 B$，故又可以把逆向队列中的串中的 Bi$替换为 Ai$，然后加入逆向队列，跟正向队列一样的方法判断。

由于这道题目不好构造 HASH，所以我们判断重复只能采用最原始的方法，就是从队首开始向后搜，依次判断（虽然很慢，但不得不用）。

5.迭代加深 DFS

BFS 可以有效求出一定情况下的最优解。不过 BFS 也有它的强力不足之处，那就是空间上的问题。原因是，BFS 需要建列表，而扩展节点众多时，可能会浪费巨大，链表在这个时候也无能为力。

这里我们就要引出一个概念，迭代加深搜索（Iterative Deepening）。

所谓迭代加深，作者的理解就是在一棵搜索树中，先假设结果的深度，然后都所到所需的深度，然后再迭代增加。

迭代加深搜索有效地节省了搜索所需要的空间，缺点是空间的代价是被大量重复计算所需要的时间所节省的。因此我们要合理选用 DFS-ID 和 BFS。

为了方便理解，我们先在下文举一个简单的例子。

[例题 1] 跳房子（USACO Contest NOV05）

奶牛们按不太传统的方式玩起了小孩子们玩的跳房子游戏。奶牛们创造了一个 5x5 的、由与 x，y 轴平行的数字组成的直线型网格，而不是用来在里面跳的、线性排列的、带数字的方格。

然后他们熟练地在网格中的数字中跳：向前跳、向后跳、向左跳、向右跳（从不斜过来跳），跳到网格中的另一个数字上。他们再这样跳啊跳（按相同规则），跳到另外一个数字上（可能是已经跳过的数字）。一共在网格内跳过五次后，他们的跳跃构建了一个六位整数（可能以 0 开头，例如 000201）。

求出所有能被这样创造出来的不同整数的总数。

这道题目已经告诉你深度，因此我们只需要普通 DFS，加上一个深度限制就可以了。我们这里给出一个深度限制的结构。

```
DFS（参数，参数，……，深度）
    当深度=限定深度时，退出
    DFS（……，深度+1）
    DFS（……，深度+1）
    ……
```

这道题目，我们可以从**每一个点**出发，出发点深度为 0，然后 DFS 限定深度搜索。本题相当好做。

[例题 2] 埃及分数（OIBH）

在古埃及，人们使用单位分数的和（形如 1/a 的，a 是自然数）表示一切有理数。如：2/3=1/2+1/6，但不允许 2/3=1/3+1/3，因为加数中有相同的。

对于一个分数 a/b，表示方法有很多种，但是哪种最好呢？

首先，加数少的比加数多的好，其次，加数个数相同的，最小的分数越大越好。如 19/45 分解成 1/5+1/6+1/18 是最好的。

编程计算最好的表达方式。

这道题目，用DFS乱搜是<u>绝对不可行</u>的。加数少的比加数多的好，因此我们直接就能想到BFS。但是事实上，BFS在空间上是绝对吃不消的。因此我们可以考虑DFS-ID。

DFS-ID 有效解决空间问题。我们可以先设搜索深度为 1 搜索，然后逐渐迭加深度，不过每次迭加需要重新搜索一次（这就是耗时的根源）。

具体的方法已经有了，我们便可以开始考虑下一步了。一个很重要的步骤就是考虑每个分数所扩展出来的范围。根据数学分析，我们就可以求得，这个分数分母最小值可以取到（上一个分母）与（剩余数字的倒数取整+1）的最大值；分母最大值可以取到（上一个分母-1）与（剩余数字的倒数取整）的最小值。因此我们就可以很快搜出解。

因此可以看出，我们需要合理选择搜索方式，来提高自己搜索的速度。

6.随机化法

一般的，我们在有些时候，很难找出一种完全精确的算法，这样的场合下，我们有了一种随机化法。随机化法的核心是从差到好，逐步逼近。这属于一种修补的方法。但是在有些场合下，这种方法可能会因为不精确而导致错误！所以说这种方法是一种近似算法。由于这种方法灵活多变，今年来在 OI 的方面也卓见成效。具体的使用方法我们在下文讲述。

[例题 1] 线型网络（OIBH）

有 N(N<=18)台 PC 放在机房内，现在要求由你选定一台 PC，用共 N-1 条网线从这台机器开始一台接一台地依次连接他们，最后接到哪个以及连接的顺序也是由你选定的，为了节省材料，网线都拉直。求最少需要一次性购买多长的网线。（说白了，就是找出 N 的一个排列 P1 P2 P3...PN 然后 P1->P2->P3->...->PN 找出 |P1P2|+|P2P3|+...+|PN-1PN| 长度的最小值)题目中给出的是各台机器的坐标。

这道题目确定的算法，非常容易超时，普通的搜索要经过很好的优化才可能过。所以，我们采取随机化的算法。

题目中告诉你的，是各台机器的坐标。所以我们第一步工作，就是把每两台机器之间的距离都计算出来，方法是勾股定理（不会不知道吧）。

然后，我们可以先随机生成一条当前的最优路径。当然这条路径就是最优路径的情况是少见的，我们必须要对此进行修补。然后采用交换的方法试，具体方法是用 Bubble Sort 的方法，去枚举每两个点，如果第 a-1 个点到第 a 个点的距离与第 b+1 个点到第 b 个点的距离之和大于第 a-1 个点到第 b 个点的距离与第 b+1 个点到第 a 个点的距离之和，则把这一段路的所有节点用反向。（为什么？）然后就这样对一条路径不断优化，直到不能优化。

然而一次生成修补的路径并不能代表最优，我们还需要把上述过程重复一定的次数，以达到最优。

[例题 2] 勇气的挑战（TOJ1073）

给定 n 个点的坐标(x，y，z)，且 n<=50，从点 1 出发，怎么样才能走一条路径，访问每个点一次且仅一次，使走过的距离和最小？

经过实验证明，这道题目的算法必须是随机化，所以说这道题在规定的时间范围内只有近似算法。这道题比上一道题稍微难一些，但是核心思想是相同的，也是求最短的路径。但是这也只不过是类似于上题的方法，所以我们只需在上题的基础上修改一下算法，使其针对于三维立体坐标轴。

还有我们要提示一下大家，经过测试，修补的次数大约可以设定为 15N 次，但是还是希望大家谨慎地去对待这类问题，毕竟正确第一。

[例题 3] 虫食算（NOIP T2004-4）

所谓虫食算，就是原先的算式中有一部分被虫子啃掉了，需要我们根据剩下的数字来判定被啃掉的字母。来看一个简单的例子:

```
  43#98650#45
+   8468#6633
-------------
  44445506978
```

其中#号代表被虫子啃掉的数字。根据算式，我们很容易判断：第一行的两个数字分别是 5 和 3，第二行的数字是 5。

现在，我们对问题做两个限制:

首先，我们只考虑加法的虫食算。这里的加法是 N 进制加法，算式中三个数都有 N 位，允许有前导的 0。

其次，虫子把所有的数都啃光了，我们只知道哪些数字是相同的。我们将相同的数字用相同的字母表示，不同的数字用不同的字母表示。如果这个算式是 N 进制的，我们就取英文字母表中的前 N 个大写字母来表示这个算式中的 0 到 N-1 这 N 个不同的数字（但是这 N 个字母并不一定顺序地代表 0 到 N-1）。输入数据保证 N 个字母分别至少出现一次。

```
  BADC
+ CBDA
------
  DCCC
```

上面的算式是一个 4 进制的算式。很显然，我们只要让 ABCD 分别代表 0123，便可以让这个式子成立了。你的任务是，对于给定的 N 进制加法算式，求出 N 个不同的字母分别代表的数字，使得该加法算式成立。输入数据保证有且仅有一组解。

关于这道题目，解法我具体也不说了，就是深度优先搜索。但是搜索的顺序是很重要的。举个例子，关于官方的第 8、9 组数据。官方的#8 数据是完全逆序的，所以对于这组数据，倒序搜索是很好的。但是#9 数据是完全升序，不用搜索便可以得出结果。

关于搜索的顺序有很多，有些数据是专门针对顺序搜索，有些则专门针对逆序搜索的，所以搜索的顺序影响了成绩。但是在比赛中评委不可能把数据拿给你吧？所以在不知道数据的情况下，有 2 种方法：双向与随机。这是比较好的方法。双向的搜索就是 1、n、2、n-1……地搜索，这个方法对于所有数据还是比较有效的，但是大都是中间的数据正好克制了它。

所以笔者也与大家一样比较偏向于随机的顺序。理由很简单：随机序的效率一般是很稳定的。

搜索的效率取决于搜索的算法与顺序，所以随机化在搜索不确定顺序的情况下是非常有用的。

7.总结

搜索是很博大精深的，方法远不止我说的几种。

使用搜索有一个很重要的问题，那就是算法的判断。一般要根据题目给出的最大值，去判断算法的可行性与估计得的分数（对于 OI 来说）。而算法时空复杂度与编程复杂度都是需要权衡的。

关于搜索的优化：

1.搜索剪枝。这是搜索最有力的优化方法，可以避免许多不必要的搜索，而且可以使搜索变得有力、有效，迅速出解。

2.随机化。搜索可以按照随机的顺序搜索，这样可以应对一些 BT 的数据，达到最稳定的效果。这种方法在 OI 上有很广泛的作用，虽然这是一种近似方法，在某些场合可能有误。

总之，搜索是无穷无限的，也是所谓的万能解决方法。

本文中所述算法不一定保证最优。