回溯

**【The Settlers of Catan】**

Nubia的苏丹没有子女，所以她决定，在她去世的时候，把她的国家分成*k*个不同的部分，每个部分将由在一些测试中表现最好的人来继承，有可能某个人继承多个部分或者全部。为了确保最终只有智商最高的人成为她的继承者，苏丹设计了一个巧妙的测试。在一个喷泉飞溅和充满异香的大厅里，放着*k*个国际象棋棋盘。每一个棋盘的方格用从1到99范围内的数字进行编号，并提供8个宝石做的皇后棋子。每一个潜在的继承人的任务是将8个皇后放置在棋盘上，使得没有一个皇后可以攻击另一个皇后，并且对于棋盘上所选择的皇后所占据的方格，要求方格内的数字的总和要和苏丹选择的数字一样高。（如果您不熟悉国际象棋的规则，这就是说，在棋盘上的每一行和每一列只能有一个皇后，并且在每条对角线上，最多只能由一个皇后。）

请您编写一个程序，输入棋盘的数量以及每个棋盘的详细情况，并确定在这些条件下每个棋盘可能的最高得分。（苏丹是一个好的棋手，也是一个优秀的数学家，她给出的数字是最高的。）

**输入**

输入首先在一行中给出棋盘的数量*k*，然后给出*k*个64个数字组成的集合，每个集合由8行组成，每行8个数字，每个数字是小于100的正整数。棋盘的数量不会多于20。

**输出**

输出给出*k*个数字，表示你的*k*个得分，每个得分一行，向右对齐，5个字符的宽度。

|  |  |
| --- | --- |
| **样例输入** | **样例输出** |
| 1  1 2 3 4 5 6 7 8  9 10 11 12 13 14 15 16  17 18 19 20 21 22 23 24  25 26 27 28 29 30 31 32  33 34 35 36 37 38 39 40  41 42 43 44 45 46 47 48  48 50 51 52 53 54 55 56  57 58 59 60 61 62 63 64 | 260 |

**试题来源：ACM South Pacific Regionals 1991**

**在线测试：UVA 167**

试题解析

对8\*8的棋盘来说，八皇后的置放方案有多种（共92种）。每种方案中置放八皇后的位置不尽相同，由于在每个棋盘的数据中，各个格中的数字不同，因此每种置放方案的得分也不尽相同。所以，在输入棋盘数据前，应该先离线计算出八皇后在棋盘上所有可能的放置方式。然后对每一个棋盘数据，计算每种置放方式中的得分，找出其中的最高得分。

1、回溯搜索8个皇后在棋盘上的所有可能放置方式

我们从上而下搜索每一行。由于棋盘上每一行和每一列只能有一个皇后，并且在每条对角线上，最多只能由一个皇后，因此需要标志每一列、每一条对角线是否被皇后选中。8\*8的棋盘共有8列、16条左对角线和16条右对角线。设：

*col*[*i*]为*i*列选中的标志（*0*≤*i*≤7）；

*left*[*ld*]为左对角线*ld*选中的标志（0≤*ld*≤15）；*right*[*rd*]为右对角线*rd*选中的标志（0≤*rd*≤15）。

经过（*r*，*c*）的右对角线序号*rd=r+c*，左对角线序号*ld=c-r*+7(见图3.4-1)，这样做，可使得16条左对角线和16条右对角线的序号互不相同。

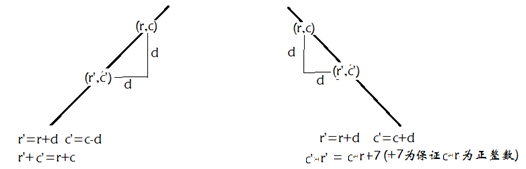


图3.4-1

当前皇后在*r*行的列位置为*tmp*[*r*]（0≤*r*≤7）；

*p*[*n*][*i*]存储第*n*种方式中8皇后的列位置（0≤*n*≤91，0≤*i*≤7）；

我们采用回溯法计算8皇后的所有可能放置方式。考虑的因素：

1. **状态：**当前行序号*r*作为递归过程的值参；*col*[*c*]、*left*[*ld*]和*right*[*rd*]亦反映了求解过程中每一步的状况，但这些参数的存储量大，为避免内存溢出，则将其设为全局变量，回溯时需恢复其递归前的值。

②**边界条件*r* *==* 8：**若搜索了所有行，则记下第*n*种方式中8个皇后的列位置（*P*[*n*][*i*]*= tmp*[*i*]，0≤*i*≤7），方式序号*n*+1，回溯；

③**搜索范围0≤*c*≤7：**若当前状态不满足边界条件，则依次搜索*r*行的每一列*c*，计算（*r*，*c*)的左右对角线序号（*ld=*(*c-r*)*+*7，*rd=c+r*）；

④**约束条件（!*col*[*c*] && !*left*[*ld*] && !*right*[*rd*]）：**若满足约束条件，则选中*c*列和左对角线*ld*和右对角线*rd*（*col*[*c*]=1,*left*[*ld*]=1,*right*[*rd*]=1），(*r*, *c*)置放皇后（*tmp*[*r*]=*c*），递归（*r*+1）。千万注意，递归回溯时需恢复递归前的参数（*col*[*c*] *=* 0, *left*[*ld*] = 0, *right*[*rd*] = 0）。

显然，从0行出发递归，便可计算出所有方案中8皇后的位置。

2、主程序

递归计算8个皇后的*n*种放置方式*P*[*i*][*j*]（0≤*i*≤*n*-1，0≤*j*≤7）；

依次处理*k*个棋盘：

读入当前棋盘数据*board*[ ][ ]；

计算和输出当前棋盘的最高得分**；

**参考程序**

#include <cstdio>

#include <vector>

using namespace std;

int P[1000][9]; //方式*i*中第*j*行皇后的列位置为*P*[*i*][*j*]

int tmp[8]; //当前方式中第*i*行皇后的列位置为*tmp*[*i*]

int n = 0; //方式数初始化

bool col[8] = {0}, left[15] = {0}, right[15] = {0}; //所有列和左右对角线未被选中

void func(int r) //从*r*行出发，递归计算所有方案中8皇后的位置

{

if (r == 8) { //若搜索了所有行

for (int i = 0; i < 8; ++i) //记下当前方式中8个皇后的列位置

P[n][i] = tmp[i];

++n; //方式数+1

return; //回溯

}

for (int c = 0; c < 8; ++c) { //依次搜索*r*行的每一列

int ld = (c - r) + 7; //计算(*r*, *c*)的左右对角线序号

int rd = c + r;

if (!col[c] && !left[ld] && !right[rd]) { //若第*c*列和左右对角线未选中，则选中*c*列和左右对角线

col[c] = 1, left[ld] = 1, right[rd] = 1;

tmp[r] = c; //(*r*, *c*)置放皇后

func(r + 1); //递归下一行

col[c]=0, left[ld]=0, right[rd]=0; //撤去第*c*列和左右对角线的选中标志

}

}

}

int main()

{

func(0); //从0行出发，自上而下递归计算所有方案中的8皇后位置

int Case;

int board[8][8];

scanf("%d", &Case); //输入棋盘数

while (Case--) {

for (int i = 0; i < 8; ++i) //输入当前棋盘中每格的数字

for (int j = 0; j < 8; ++j)

scanf("%d", &board[i][j]);

int ans = 0;

for (int i = 0; i < n; ++i) { //依次搜索每个方式

int sum = 0; //第*i*个方式的得分初始化

for (int j = 0; j < 8; ++j) //搜索每一行，累计第*i*个方式的得分

sum += board[j][P[i][j]];

if (sum>ans) ans=sum //若当前方式的得分目前最高，则调整棋盘最高得分

}

printf("%5d\n", ans); //输出当前棋盘的最高得分

}

}

**【The Settlers of Catan】**

在1995年，德国在Settlers of Catan举办了一场游戏，玩家们要在一座岛屿的未知的荒野上，通过构建道路、定居点和城市来控制这个岛屿。

您受雇于一家软件公司，公司刚刚决定开发这款游戏的电脑版，要求您实现这款游戏的一条特殊规则：

当游戏结束时，建造了最长的道路的玩家将获得额外的两分。

问题是，玩家通常建造复杂的道路网络，而不是一条线性的路径。因此，确定最长的路不是容易的（虽然玩家们通常可以马上看出）。

和原始的游戏相比，我们仅要解决一个简化了的问题：给出一个节点（城市）的集合和一个边（路段）的集合，这些连接节点的边的长度为1。

最长的道路被定义为网络中每条边都不会经过两次的最长的路径，虽然节点可以被经过超过一次。

例如，图3.4-2中的网络包含了一条长度为12的道路。

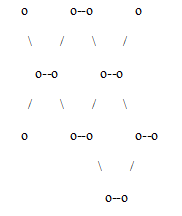


图3.4-2

**输入**

输入包含一个或多个测试用例。

每个测试用例的第一行给出两个整数：节点数*n* (2≤*n*≤25) 和边数*m* (1≤*m*≤25)。接下来的*m*行描述*m*条边，每条边由这条边所连接的两个节点的编号表示，节点编号从0 到*n*-1。边是无向边。节点的度最多为3。道路网不一定是连通的。

输入以*n*和*m*取0为结束。

**输出**

对每个测试用例，在单独的一行中输出最长路的长度。

|  |  |
| --- | --- |
| **样例输入** | **样例输出** |
| 3 2  0 1  1 2  15 16  0 2  1 2  2 3  3 4  3 5  4 6  5 7  6 8  7 8  7 9  8 10  9 11  10 12  11 12  10 13  12 14  0 0 | 2  12 |

**试题来源：University of Ulm Local Contest 1998**

**在线测试：POJ 2258，ZOJ 1947，UVA 539**

试题解析

道路网络是一个无向图。由于最长的道路中的每条边仅允许经过一次，由此设定无向图的相邻矩阵*a*[*i*][*j*]=

由于试题并未规定路径的起始点，因此需要将每个节点设为路径起始点计算路长，从中调整最长路长*best*。那么，如何计算以*i*节点为首的路径长度，调整*best*呢？显然采用回溯法是比较适宜的。

**状态：**求解过程中每一步的状况为目前路径的尾节点*i*和路长*l*；各条边的访问标志*a*[ ][ ]。为了避免内存溢出，我们设递归过程的值参为(*i*，*l*)，将*a*[ ][ ]设为全局变量，但回溯时需恢复其递归前的未访问标志。

**注：**由于是求最长路，路径能长则长，因此不设边界条件。

**搜索范围0≤*j*≤*n*-1:**可能与*i*节点关联的所有节点。

**约束条件*a*[*i*][*j*]==1：**若(*i*，*j*)为未访问边，则撤去（*i*，*j*）的访问标志（*a*[*i*][*j*]= *a*[*j*][*i*]=0），递归(*j*，*l*+1)；回溯时恢复其递归前的未访问标志（*a*[*i*][*j*]= *a*[*j*][*i*]=1）。

搜索完与*i*节点关联的所有节点后，便得出以*i*节点为首的路径长度*l*。此时，调整最长路长*best=max*{*best*，*l*}。

显然，在主程序中设*best*=0，依次计算以每个节点为首的路径（即递归（*i*，0），0≤*i*≤*n*-1），便可以计算出最长路长*best*。

**参考程序**

#include <stdio.h>

#include <assert.h>

#define DBG(x)

FILE \*input;

int n,m,best; //节点数*n*，边数*m*，最长路的长度*best*

int a[32][32]; //无向图的邻接矩阵

int read\_case() //输入测试用例信息，构造无向图的邻接矩阵

{

int i,j;

fscanf(input,"%d %d",&n,&m); //输入节点数和边数

DBG(printf("%d nodes, %d edges\n",n,m));

if (m==0 && n==0) return 0; //输入以*n*和*m*取0为结束

for (i=0; i<n; i++) //邻接矩阵初始化为空

for (j=0; j<n; j++) a[i][j] = 0;

while (m--) //依次输入*m*条边信息， 构造无向图的邻接矩阵

{

fscanf(input,"%d %d",&i,&j);

a[i][j] = a[j][i] = 1;

}

return 1;

}

void visit (int i, int l) //递归计算当前路径（目前行至节点*i*，路长为*l*），调整最次路长

{

int j;

for (j=0; j<n; j++) //搜索*i*节点相关的未访问边

if (a[i][j]) //若（*i*, *j*）未访问，则设访问标志

{ a[i][j]= a[j][i]=0;

visit(j,l+1); //沿*j*节点递归下去

a[i][j] = a[j][i] = 1; //恢复（*i*, *j*）未访问标志

}

if (l>best) best=l; //若当前路长为最长，则调整最长路的长度

}

void solve\_case() //计算和输出当前测试用例的解

{

int i;

best = 0; //最长路长度初始化

for (i=0; i<n; i++) //依次递归以每个节点为首的路径，调整最长路长

visit(i,0);

printf("%d\n",best); //输出最长路长

}

int main()

{

input = fopen("catan.in","r"); //输入文件初始化

assert(input!=NULL);

while (read\_case()) solve\_case(); //反复输入测试用例，计算和输出解，直至程序结束

fclose(input); //关闭输入文件

return 0;

}

**【Transportation】**

Ruratania刚刚进入资本主义社会，在包括运输业在内的多个领域中，正在建立新型的企业化的运作机制。运输公司TransRuratania要开设从城市A到城市B的一趟新的特快列车，途中要停若干站。车站依次编号，城市A编号为0，城市B编号为*m*。公司要进行一项实验，以改进乘客的运输量，并增加其收入。这列火车乘客的最大容量为*n*位乘客。火车票的价格等于出发站和目的地站（包括目的地站）之间的停车次数（也就是站的数量）。在列车从城市A出发之前，从所有在线路上的站的订票信息已经被获取完毕。一份车站S的订票订单是从车站S出发，且到一个确定的目的地站的所有预订。如果因为乘客容量的限制，公司有可能不能接受所有的订单，在这种情况下，公司的策略是，对于每个站的每一份订票订单，要么完全接受，要么完全拒绝。

请您编写一个程序，基于城市A到城市B的路线上的车站给出的订单，确定TransRuratania公司最大可能的公司总收入。一个被接受订单的收入是订单中乘客的数量和他们的车票价格的乘积。公司总收入则是所有被接受的订单的收入的总和。

**输入**

输入被划分为若干个测试用例。每个测试用例的第一行包含3个整数：列车乘客的最大容量为*n*，城市B的编号，所有车站被预订的车票总数。下面的行给出订票订单。每份订单包含3个整数：出发站，目的地站，乘客数量。在每个测试用例中，最多22份订单。城市B编号最多为7。测试用例以第一行的3个整数全部取0作为输入结束。

**输出**

除输入结束标志以外，对每个测试用例，输出一行，每行给出最大可能的总的收入。

|  |  |
| --- | --- |
| **样例输入** | **样例输出** |
| 10 3 4  0 2 1  1 3 5  1 2 7  2 3 10  10 5 4  3 5 10  2 4 9  0 2 5  2 5 8  0 0 0 | 19  34 |

**试题来源：ACM Central European Regional Contest 1995**

**在线测试：POJ 1040，UVA 301**

提示

显而易见，本题应采用回溯法求解。

**1、回溯搜索前的预处理**

设最大收入为*best*，初始时为0；订单数，即所有车站被预订的车票总数为*count*；

订单序列*orders*[ ]，其中第*i*份订单的起始站为*orders*[*i*].*from*，目的地站为*orders*[*i*].*to*，旅客数为*orders*[*i*].*passangers*。若公司接受该订单，则收入为*orders*[*i*].*price*=(*orders*[*i*].*to- orders*[*i*].*from*) \**orders*[*i*].*passangers*（0≤*i*≤*count*-1）。

我们以*price*域为关键字，按递增顺序排列订单序列*orders*[ ]。计算接受剩余订单的收入总和*orders*[*i*].*remaining*=，该域在回溯算法中列为最优性条件。回溯搜索时，我们按照*order*s[ ]的顺序枚举每份订单。

车站序列*train*[ ]，其中车站*i*的旅客数为*train*[*i*]，初始时*train*[ ] 清零。

**2、回溯搜索**

我们按照接受订单的收入递增顺序搜索每份订单：

**状态：**求解过程中每一步的状况为当前订单*start*；已接受的订单收入*earnings*；这两个参数列为递归过程的值参（*start*，*earnings*）。另外，各车站的人数*train*[ ]亦为状态，因为公司一旦接受订单，则需要累计途径火车站的人数，判断是否超载；回溯时需要恢复订单接受前途径火车站的人数。为了避免内存溢出，*train*[ ]列为全局变量。

**最优目标状态的条件*earnings > best*：**若当前收入最大，则调整最大收入(*best=earnings*)；

**搜索范围*start≤k≤count*-1。**

订单*k*是否被接受，需要同时满足约束条件和最优性要求：

**最优性要求*earnings+orders*[*k*].*remaining*<*best***：若即使按收入最大化要求，全部接受剩余订单，也不可能更优，则拒绝*k*订单，回溯；否则判断

**约束条件：**订单*k*途径的每个火车站*i*(*orders*[*k*].*from≤i≤orders*[*k*].*to*-1)增加了订单*k*的旅客后没有超载，即*train*[*i*] += *orders*[*k*].*passangers*)*≤n*。

若不满足约束条件，则拒绝订单*k*，恢复先前订单*k*途径的每个火车站人数（for( ; *i* >= *orders*[*k*].*from*; *i*--) *train*[*i*] -= *orders*[*k*].*passangers*）；

若满足约束条件，则接受订单*k*，递归子状态(*k* + 1, *earnings + orders*[*k*].*price*)。回溯时，恢复订单*start*接受前各车站的人数（for( ; *i >= orders*[*k*].*from*; *i*--) *train*[*i*] -= *orders*[*k*].*passangers*）。

显然，递归状态（0，0）后得出的*best*即为问题解。

**【Don't Get Rooked】**

在国际象棋中，车是一个可以纵向或横向移动任意个方格的棋子。本题我们仅考虑小棋盘（最多4× 4），棋盘上还设置了若干堵墙，车无法通过墙。本题的目标是要使得在棋盘上的车两两之间不能相互攻击对方。在棋盘上，一个合法的车的放置是没有两个车在同一水平行或垂直行，除非至少有一堵墙将它们分隔开。

图3.5-1给出了五张相同的棋盘。第一张图是空的棋盘，第二和第三张图给出了合法放置车的棋盘，而第四和第五张图则是非法放置车的棋盘。这张棋盘，可以合法放置的车的最大数量是5；第二张图给出了一种放置方法，当然还有一些其他的放置方法。

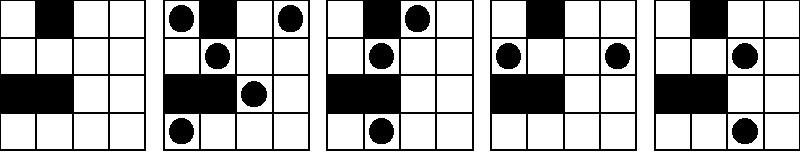


图3.5-1

请您编写一个程序，给出一张棋盘，计算在棋盘上可以合法放置车的最大数量。

**输入**

输入给出一张或多张棋盘的描述，然后在一行中给出数字0表示输入结束。每张棋盘的描述先在一行中给出一个正整数***n***，表示棋盘的大小；***n***最多为4。接下来的***n***行每行描述棋盘的一行，一个‘.’表示一个可以放置车的方格，而大写字母‘X’表示墙。输入中没有空格。

**输出**

对于每一个测试用例，输出一行，给出在棋盘上可以合法放置的车的最大数量。

|  |  |
| --- | --- |
| **样例输入** | **样例输出** |
| 4  .X..  ....  XX..  ....  2  XX  .X  3  .X.  X.X  .X.  3  ...  .XX  .XX  4  ....  ....  ....  ....  0 | 5  1  5  2  4 |

**试题来源：ACM Mid-Central USA 1998**

**在线测试：POJ 1315，UVA 639**

提示

我们给每个棋格定义3个标志：

1. ‘.’标志；
2. 合法放置车的‘P’标志；
3. 墙标志‘X’。

在计算棋盘上合法放置的最大车的数量前，做一个预处理：在棋盘的外围（0行、*size*+1行和0列）添加一堵墙（见图3.5-2）。

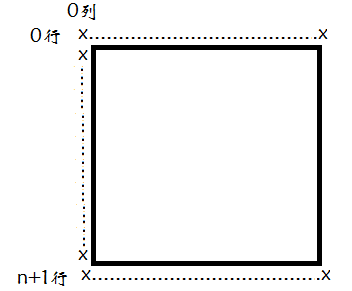


图3.5-2

按由上而下、由左而右的顺序搜索每个棋格。显然，若越过(*x*, *y*)左方和上方连续个‘.’格后遇到的第1个是‘X’格，则(*x*, *y*)是车的安全放置位置，标志(*x*, *y*)是‘P’格；否则(*x*, *y*)不是车的安全放置位置。

我们可以通过回溯法计算棋盘上可合法放置的最大车数。设*nPlaced*为当前合法放置的车数；*mostPlaced*为可放置的最多车数，初始时*mostPlaced*为0。

**状态：**求解过程中每一步的状况包括

1. 当前格(*x*, *y*)，该参数列为递归过程的值参；
2. 棋盘*board*[ ][ ]，为了避免溢出，将其设为全局变量，回溯时需恢复递归前的状态；

**搜索范围*x≤n*：**即按照自上而下顺序搜索*n*行

**约束条件(*x*, *y*)是‘.’格且为车的安全放置位置：**若满足该约束条件，则(*x*, *y*)置‘P’标志；递归(*x*, *y*+1)后得出的车数+1即为当前方案合法放置的车*nPlaced*。若该车数为目前最大（*nPlaced>mostPlaced*），则调整*mostPlaced=nPlaced*。回溯时，需恢复递归前的棋盘状态，即恢复(*x*, *y*)递归前的‘.’标志；

无论(*x*, *y*)是否满足约束条件，需要计算下一个搜索位置：若*y≥n*，则下一个搜索位置为(*x*+1,1)；否则下一个搜索位置为(*x*, *y*+1)。

显然，从(1, 1)格出发进行回溯搜索，便可计算出可放置的最多车数*mostPlaced*。

**【8 Queens Chess Problem】**

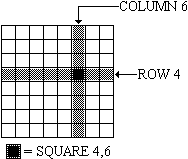
在国际象棋的棋盘上，可以放置8个皇后，使得没有一个皇后会被其他的皇后攻击。给出一个皇后的初始位置，请编写一个程序，确定所有可能的8个皇后的放置。

不要去试图写一个程序计算8个皇后在棋盘的每一个可能的8种放置。这将要进行88次的计算，会使得系统崩溃。对于您的程序，会有一个合理的运行时间约束。

**输入**

您的程序的输入是用一个空格分隔开的两个数字。这些数字表示在棋盘上，8个皇后中的一个所占据的位置。给出的棋盘是合法的，程序不需要对输入进行验证。

为了规范我们的表示，本题设定棋盘的最左上角的位置是(1, 1)。水平行的最上面的行是第1行，垂直列的最左边是第1列。如下图，方格(4,6)意味着第4行，第6列。



**输出**

您的程序输出是一行一个解答。

每个解答是1…*N*的数字序列。每个解由8个数字组成。这8个数字每个都是解答的行坐标。列坐标按8个数字的顺序给出。也就是说，第1个数字是在第1列的皇后的行坐标，第2个数字是在第2列的皇后的行坐标，以此类推。

下述的样例输入产生4个解答，下面给出每个解答的完整的8×8的表示。

SOLUTION 1 SOLUTION 2 SOLUTION 3 SOLUTION 4

1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0

0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0

0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0

0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0

0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0

如上所述，每个解答仅输出一行，用8个数字表示。解答1将皇后放置在第1行，第1列；第5行，第2列；第8行，第3列；第6行，第4列；第3行，第5列；......，第4行，第8列。

按样例输出所示，输出两行列标题，并按字典序输出解答。

|  |  |
| --- | --- |
| **样例输入** | **样例输出** |
| 1 1 | SOLN COLUMN  # 1 2 3 4 5 6 7 8  1 1 5 8 6 3 7 2 4  2 1 6 8 3 7 4 2 5  3 1 7 4 6 8 2 5 3  4 1 7 5 8 2 4 6 3 |

**试题来源：ACM East Central 1988**

**在线测试：UVA 750**

提示

**1、离线计算八皇后的所有置放方案**

在确定一个皇后位置的情况下，其余七个皇后的置放方案有多个。我们不妨先离线计算出八皇后的所有置放方案，将这些方案存放在一个记录表中。以后，每输入一个皇后位置（*r*，*c*），就在记录表中查询所有含（*r*，*c*）的方案。设

方案数为*l*；

方案记录表*sol*[ ][ ]，其中*sol*[*k*][*j*]为第*k*个方案中位于第*j*列上皇后的行位置（0≤*k*≤*l*-1，0≤*j*≤7）；

当前方案*temp*[ ]，其中第*i*列上皇后的行位置为*temp* [*i*]（0≤*i*≤7）；

行标志*row*[ ]，其中*row*[*i*]=true标志第*i*行目前未置放皇后；

左对角线标志*leftDiag*[ ]，其中*leftDiag*[*k*]=true标志第*k*条左对角线目前未置放皇后。经过（*r*，*c*）的左对角线序号*k=r+c*；右对角线标志*rightDiag*[ ]，其中*rightDiag*[*k*] =true标志第k条右对角线目前未置放皇后。经过（*r*，*c*）的右对角线序号*k=c-r*+8。计算左右对角线序号的思想方法可参阅例题**【3.4.2 The Settlers of Catan】**。

我们采用回溯法计算八皇后的所有置放方案：

**状态：**求解过程中每一步的状况包括

1. 当前列*c*，该参数列为递归过程的值参；
2. 行标志*row*[ ]、左对角线标志*leftDiag*[ ]和右对角线标志*rightDiag*[ ]。为了避免溢出，将其设为全局变量，回溯时需恢复递归前的状态；

**边界条件c == 8：**若8列搜索完，则第9列皇后的行位置设为0（*temp*[8] = 0），当前方案记为第*l*个方案（strcpy (*sol*[*l*], *temp*)），下一个方案的序号*l*++，回溯（return）；

**搜索范围0≤*r*≤7：**若8列未搜索完，则按自上而下顺序搜索c列的每一格；

**约束条件（*row*[*r*] && *rightDiag*[*c-r*+8] && *leftDiag*[*c+r*])=true：**若*r*行和经过（*r*, *c*）的左右对角线目前没有皇后，则设定*r*行和经过（*r*，*c*）的左右对角线有皇后标志(*row*[*r*] = *rightDiag*[*c-r*+8] = *leftDiag*[*r+c*]=false)，（*r*，*c*）置放皇后(*temp*[*c*] =*r*)；递归*c*+1列。回溯时，恢复递归前*row*[ ]、*leftDiag*[ ]和*rightDiag*[ ]的值(*row*[*r*]=*rightDiag*[*c-r*+8]=*leftDiag*[*r+c*]=true)。

显然从0列出发进行递归，便可得出八皇后置放的方案记录表*sol*[ ][ ]。

**2、从方案记录表*sol*[ ][ ]中找出含皇后位置(r，c)的置放方案**

每输入1个皇后位置（*r*，*c*），则检索方案记录表*sol*[ ][ ]：

若*sol*[*i*]满足条件（*sol*[*i*][*c*-1] == *r*-1）=true，则*sol*[*i*][0]…*sol*[*i*][7]即为含皇后位置(*r*，*c*)的一个置放方案（0≤*i*≤*l*-1）。