# 不甘心的皇后

印象最深刻的程序

<http://acm.scu.edu.cn/soj/problem.action?id=1082>

这是我做过印象最深刻的编程作品--皇后问题，我把对它的理解和分析过程整理在这里了，希望老师过目。我自己也经常关注ACM，并且希望在之后的学习生活中能够参加到ACM比赛中去锻炼思维，从中获得快乐。

#include<stdio.h>

#include<math.h>

#include<string.h>

int how,zero[10]; # 变量how表示放置的皇后种数，zero数组存放皇后的位置

int main() {

void count(int\* q, int length, int now);

while (1) {

int n, queen[10]; # n表示棋盘大小，queen是存放皇后的数组，最多十个皇后。

scanf\_s("%d", &n); # 输入n

if (n == 0) { # 如果棋盘边长为0，则无法放置皇后

break; # 跳出循环

}

memset(zero, -1, sizeof(zero)); # 将zero数组中的值都初始化为-1

how = 0;

for (int i = 0; i < n; i++) {

scanf\_s("%d",&queen[i]); # 输入每列已有的皇后位置

if (queen[i] == 0) { # 如果皇后位置为0，表示这一列还没有皇后

zero[i] = 0; # 那么zero数组中对应的列值也为0

}

}

count(queen, n, 0); # 从第一列开始计算可放置皇后的方法数

printf("%d\n", how); # 输出该棋盘可放置皇后的方法种数

}

return 0;

}

# 比较两个数大小，输出较大的一个值

int max(int a, int b) {

if (a > b) {

return a;

}

else

{

return b;

}

}

# 计算满足要求的情况数量

void count(int \*q,int length,int now) { # 函数变量：queen数组，棋盘大小，当前列值

if (now == length) { # 如果当前位置等于棋盘大小

int var; # 设置整数变量

for (var = 0; var < length-1&&pow(q[var]-q[var+1],2)<=1; var++) { # 从第一列开始遍历，如果每两个相邻列皇后之间的行距离大于1，则停止遍历

;

}

if (var == length - 1)how++; # 如果停止遍历后的var正好等于length-1，即已经遍历完所有列，则说明这种放置皇后的方法是正确的，方法数加1

return;

}

if (zero[now]==0) { # 如果当前列没有皇后

if (now==0) # 如果是第一列

{

for (int j = 1; j <= length; j++) { # 从第一列的每一行开始遍历

q[now] = j;

count(q, length, now + 1); # 计算第一列皇后在第j行时，棋盘可放置皇后种数

}

}

else { # 如果不是第一列，那么该列皇后的位置范围应为[前一列皇后位置-1，前一列皇后位置+1]

for (int j = max(1,q[now-1]-1); j <= length&&j<=q[now-1]+1; j++) { # max(1,q[now-1]-1) 是为了确保皇后位置最小是1

q[now] = j;

count(q, length, now + 1); # 计算当前列皇后在第j行时，棋盘可放置皇后种数

}

}

}

else

{

count(q, length, now + 1); # 当前列皇后已确定，计算后续列棋盘可放置皇后种数

}

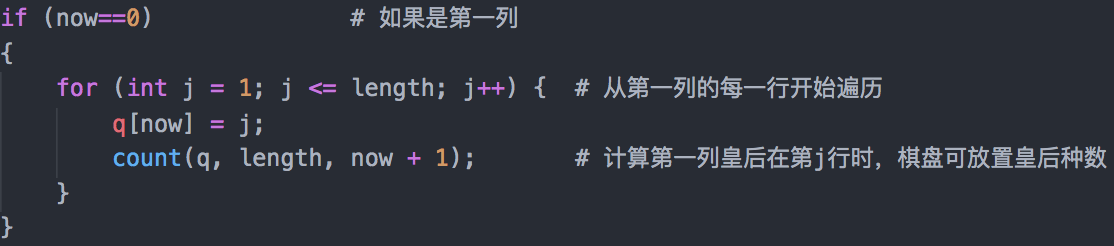
}

采用递归调用函数的思想，算法思路如下：

1. 从棋盘的第一列开始，判断当前列的皇后位置是否已经确定。若每一列皇后位置已经确定，则跳到下一行。若尚未确定，则根据上一列皇后的位置确定该列皇后位置的范围，然后在此范围内依次遍历，调用count函数计算棋盘可放置皇后种数。

注意：第一列比较特殊，需要单独处理。因为当第一列的皇后位置不确定时，它可以在

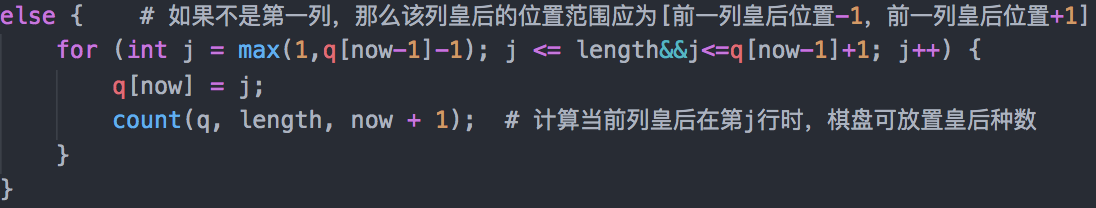
任意一行。所以单独计算。



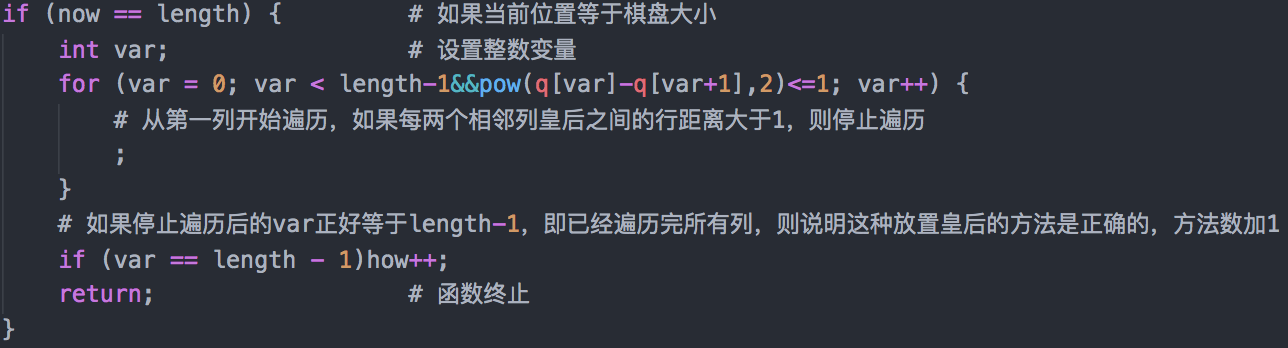
而对于其它列，该列皇后的位置范围应为[前一列皇后位置-1，前一列皇后位置+1]。

另外，为了避免前一列皇后位置-1=0的情况，这时该列皇后位置是无意义的，所以添

加了一个max函数，取“前一列皇后位置-1”的值和1的较大值。



1. 当遍历到最后一列时，所有皇后摆放完毕，判断当前摆放的棋盘是否满足题目要求。即采用for循环判断每两个相邻列皇后之间的行距离是否都小于等于1。如果满足题目要求，则摆放皇后种数增加1。函数终止。



理解和总结：

这道题和八皇后问题有些类似。但是条件不同，这道题需要确保棋盘中每两个相邻列的皇后之间的行距离最多只能差一格。所以只有先确定第一列的皇后位置，才能确定第二列的皇后位置，继而确定第三列、第四列等等。而每一列的皇后位置又有多种可能，因此想到使用深度优先搜索算法。对于每一列的可选择皇后位置中，选择一个位置作为出发点，然后依次从该点出发搜索下一列可能的皇后位置，直至棋盘中所有列都被搜索过为止，然后判断这条路径是否满足题目要求。

虽然我的编程水平还达不到做此类题目游刃有余，但我很享受沉浸其中思考解决编程思维问题的过程。