

八皇后问题(N皇后问题)

题目标签

- 题目难度：困难
- 知识点标签：算法
- 课程时长：30分钟

题目描述

八皇后问题是在一个 8*8 的棋盘上放置皇后，要求其放置后满足同一行，同一列，同一对角线上没有重复的皇后出现。试问有多少种摆盘方式？

解题思路

我们的主要思路是通过一行一行的放置皇后，来使得每一行都有一个皇后。当然，这些皇后在放置时都必须满足规定的要求才行。

因此就会出先如下情况:

- 放置时不符合规则，继续检索同一行的下一列位置是否合理
- 如果符合规则就将其放置，然后进行下一行的尝试（递归）
- 如果有某一行没有可行的解，则退回上一行，消除上一行摆放的皇后，检索剩余的列，看是否有合理的位置，然后继续进行。(回溯)
- 直到所有的行都被放置为止。

注意的情况

需要注意的是，我们在放置皇后时需要检测其防止和理性的判断条件为:

1. 同一列的上方所有行中是否有皇后
2. 左上方对角线上是否有皇后
3. 右上方对角线上是否有皇后

算法实现(java)

```
public class EightQueen {
    private static final int num = 8; // 可以拓展为N皇后问题
    private static int[][] item = new int[num][num];
    private static int methods = 0; // 总方法数
    public static void main(String[] args) {
        buildQueen(0);
        System.out.println(methods);
    }
    /**
     * 构建棋盘的第row行
     *
     * @param row
     */
    private static void buildQueen(int row) {
        if (row == num) {
```

```

        methods++;
        return;
    } else {
        for (int col = 0; col < num; col++) { // 每一列进行检查，试探性放置
            if (issatisfy(row, col)) {
                item[row][col] = 1;
                buildQueen(row + 1);
                item[row][col] = 0;
            }
        }
    }
}

/**
 * 检查row行col列元素是否满足要求
 * 因为是一行行的放置皇后，所以不需要检测同一行是否存在重复皇后
 * 在判断重复元素时，只需要判断上半部分的区域即可
 *
 * @param row
 * @param col
 * @return
 */
private static boolean issatisfy(int row, int col) {
    for (int i = 0; i < row; i++) {
        if (item[i][col] == 1) { // 同一列的上方元素
            return false;
        }
    }
    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) { // 左上方斜对角线
        if (item[i][j] == 1) {
            return false;
        }
    }
    for (int i = row, j = col; i >= 0 && j < num; i--, j++) { // 右上方斜对角
        if (item[i][j] == 1) {
            return false;
        }
    }
    return true;
}
}

```

优雅的位运算解法

	0	1	2	3	4	5	6	7
0	Q							
1					Q			
2								Q
3	A		C	B	A		BIC	A
4								

我们可以看到，前三行已经放置了皇后Q，我们需要在第四行选择放置皇后的点。阴影部分表示会出现冲突的格子，而冲突我们主要分为三种：同列冲突、右下方冲突和左下方冲突。

而就这对这种情况而言(此例为八皇后问题，可拓展到N皇后)，一行刚好8个格子，对应8位二进制数字。因此我们可以首先定义冲突：

同列冲突: $A = 1000\ 1001$;

右下冲突: $B = 0001\ 0010$;

左下冲突: $C = 0010\ 0010$;

其中1表示冲突的格子，0表示可以放置皇后的格子。因此我们可以轻松得出综合的冲突情况：

$D = (A \mid B \mid C) = 1011\ 1011$;

对于我们将要放置的第四行而言，现在有两个0，意味着有两个可以放置皇后的位置，我们需要将所有情况都考虑到，这里有一个神奇的式子： $\text{bit} = (D + 1) \& \sim D$ ；它计算得出的结果是: 0000 0100;

其实它能够得到最右边一个可以放置皇后的位置，并用1来表示，其余位是0。这样做是有好处的...

我们现在得出 $\text{bit} = 0000\ 0100$ ，便能够轻松得到下一行的冲突 $A' = (A \mid \text{bit})$; $B' = (B \mid \text{bit}) \gg 1$; $C' = (C \mid \text{bit}) \ll 1$ ；便能够很轻易地写出递归式了。

而我们的第4行试探其实并没有结束，只是从左向右的第一个可以放置的位置进行了试探，那想要取到第二个可以放置的位置怎么办呢？很简单，只需要做如下运算：

$D = D + \text{bit}$ 将刚才试过的那一位设置为不能放置皇后状态，然后继续做 $\text{bit} = (D + 1) \& \sim D$ 即可。

一直循环的试探，知道D全部为1为止。

下面是整个程序的代码：

```
public class NQueen {

    private static final int N = 8; // 皇后数量，可拓展为N皇后
    private static int count = 0; // 总方法数
    private static int limit;

    public static void main(String[] args) {
        limit = (1 << N) - 1;
        backtracking(0, 0, 0, 0);
    }
}
```

```

        System.out.println(count);
    }

    private static void backtracking(int a, int b, int c, int depth) {
        if (depth == N) {
            count++;
            return;
        }
        int d = a | b | c;
        while (d < limit) {
            int bit = (d + 1) & ~d;
            backtracking(a | bit, limit & ((b | bit) >> 1), limit & ((c | bit)
<< 1), depth + 1);
            d |= bit;
        }
    }
}

```

C语言实现

```

#include<stdio.h>

#define line 8

void queen(int i,int j);

int check(int i,int j);

int chess[line][line];

int cas=0;
int xx,yy;

int main(){

queen(0,0);

printf("%d\n",cas);

return 0;

}

void queen(int i,int j){

if(j>=line){

return ;

}

```

```

if(check(i,j)==1){//如果能放

    chess[i][j]=1;//放皇后

    if(i==line-1){//如果是最后一行，记录情况

        cas++;
        /*下面是输出每种棋盘结果，供测试
        for (xx=0;xx<8;xx++)

            for(yy=0;yy<8;yy++){

                printf("%d",chess[xx][yy]);

                if(yy==7)

                    printf("\n");

            }

            printf("\n");

        上面是输出结果*/

    }

    else{

        queen(i+1,0);//不是最后一行就分析下一行

    }

}

chess[i][j]=0;//如果此位置不能放，就置空（0），判断旁边的格子。

    //如果此位置能放，走到这里就意味着上面的代码全部执行了，把皇后拿走（置零），再讨论其他情况，拿
    旁边位置试探。

    queen(i,j+1);

}

int check(int i,int j){

    int k;

    for(k=0;k<line;k++){

        if(chess[i][k]==1) return 0;//0=不能放

    }

```

```
for(k=0;k<line;k++){  
  
    if(chess[k][j]==1) return 0;  
  
}  
  
for(k=-line;k<=line;k++){//两对角线  
  
    if(i+k>=0&&i+k<line&&j+k>=0&&j+k<line)//从左上到右下对角线  
  
        if(chess[i+k][j+k]==1) return 0;  
  
    if(i-k>=0&&i-k<line&&j+k>=0&&j+k<line)//从左下到右上对角线  
  
        if(chess[i-k][j+k]==1) return 0;  
  
}  
  
return 1;  
  
}
```