

题目四 N皇后问题

一.设计思路

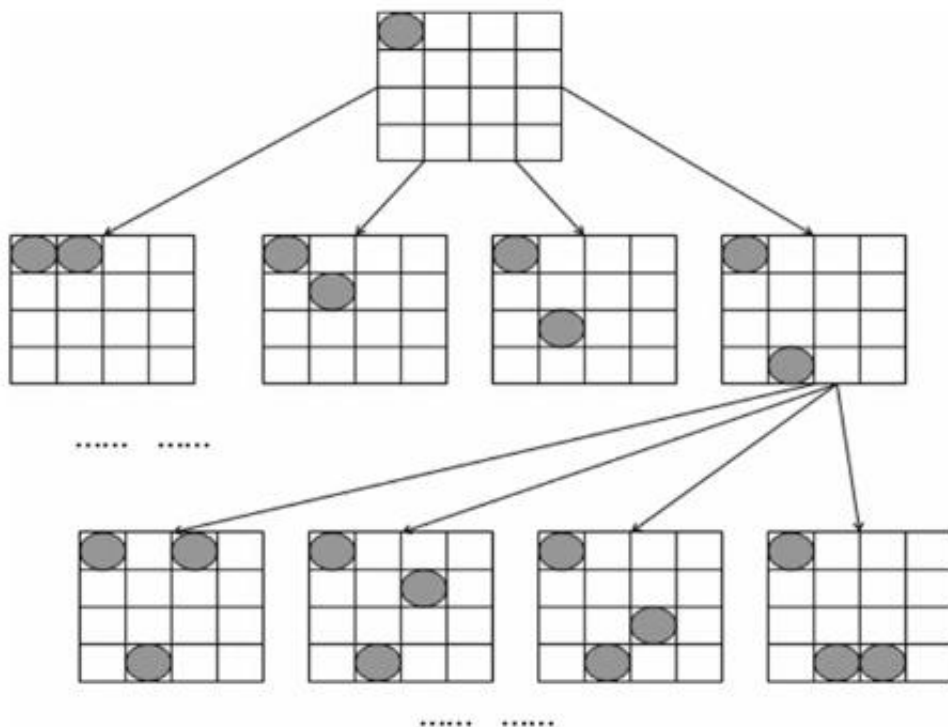
N皇后问题要求在 $N \times N$ 的棋盘上摆放N个皇后。要求没有一个皇后能够“吃掉”任何其它一个皇后，即任意两个皇后不能处于同一行，同一列或者同一条对角线上，求解有多少种摆法。

由于在棋盘上摆放N个皇后可能的摆法数量巨大，如果采用简单的循环枚举策略会导致高昂的时间开销，甚至根部的不到最终结果。因此在进行摆放尝试时，每一次的摆法应该参考之前棋子的位置，排除明显不可能的摆放位置。

为此，采用回溯法来解决N皇后问题。算法思路可简述如下。

- 将第一个皇后放置在第1行第1列
- 尝试在下一行上放置一个皇后，先尝试放在第一列是否与已放置的皇后冲突
- 若不冲突，则将皇后放置在该列
- 若发生冲突，则选择该行的下一列继续尝试
- 若果N列都冲突，则回到上一行，重新选择位置。
- 当所行都已经安排好合理位置的皇后，则生成一种摆法

其原理可表示如下：



在该算法的基础上，定义适当的数据结构来存储棋盘信息，不断探寻，即可得到N皇后问题的解。

二.数据结构实现

1.棋盘类型（map）

定义了棋盘 `int map[N]` 来表示棋盘信息。
其中 `N` 为棋盘边长，也是要摆放皇后的个数。
事实上，棋盘并没有被真正的存储，也没有必要被存储。因为在一个规模为 `N*N` 的棋盘中，我们只关心皇后摆放的位置，其他的点位都是没有意义的。所以只需要开辟一个长度为 `N` 的数组，数组下标 `i` 代表第 `i` 行中皇后被摆放的列号。

2.计数器（count）

定义了计数器 `int coutn` 用于记录已经生成的合理摆法的数量。

3.辅助函数

```
bool qualified(int* map,int m,int n);    // 合法性检验函数
```

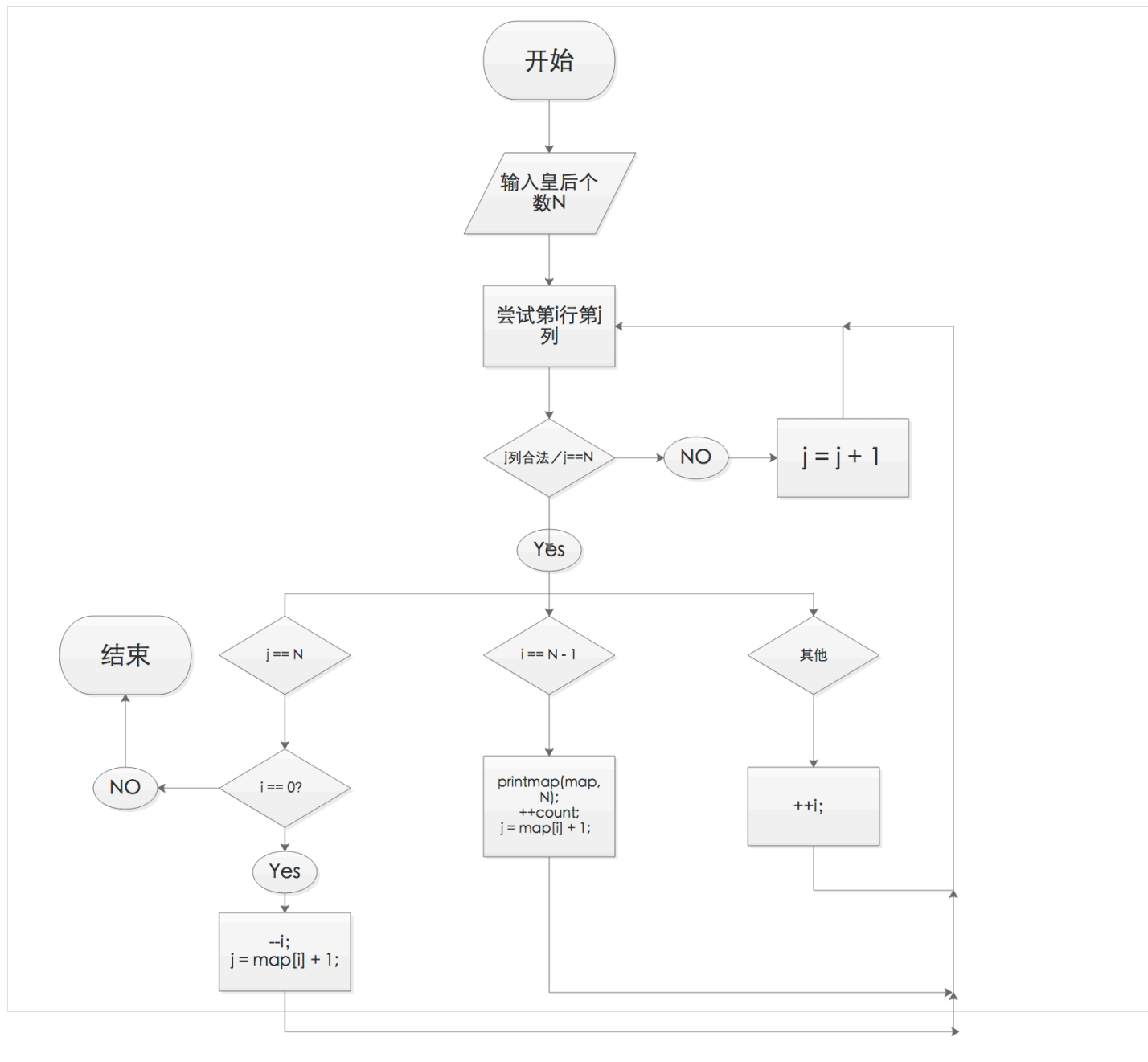
合法性检验函数，检验当前摆放位置是否合法，即是否其横竖斜向都不存在其他皇后。

```
void printmap(int* map,int n);          // 打印合理的N皇后布局
```

皇后位置打印函数，根据给定的皇后位置信息，格式化输出该图中皇后摆法。

三.系统实现

1.系统执行框架



首先提示用户输入皇后数量，用 `int n` 读入皇后数量。

然后定义常量 `const int N = n` 用于建立棋盘信息数组。

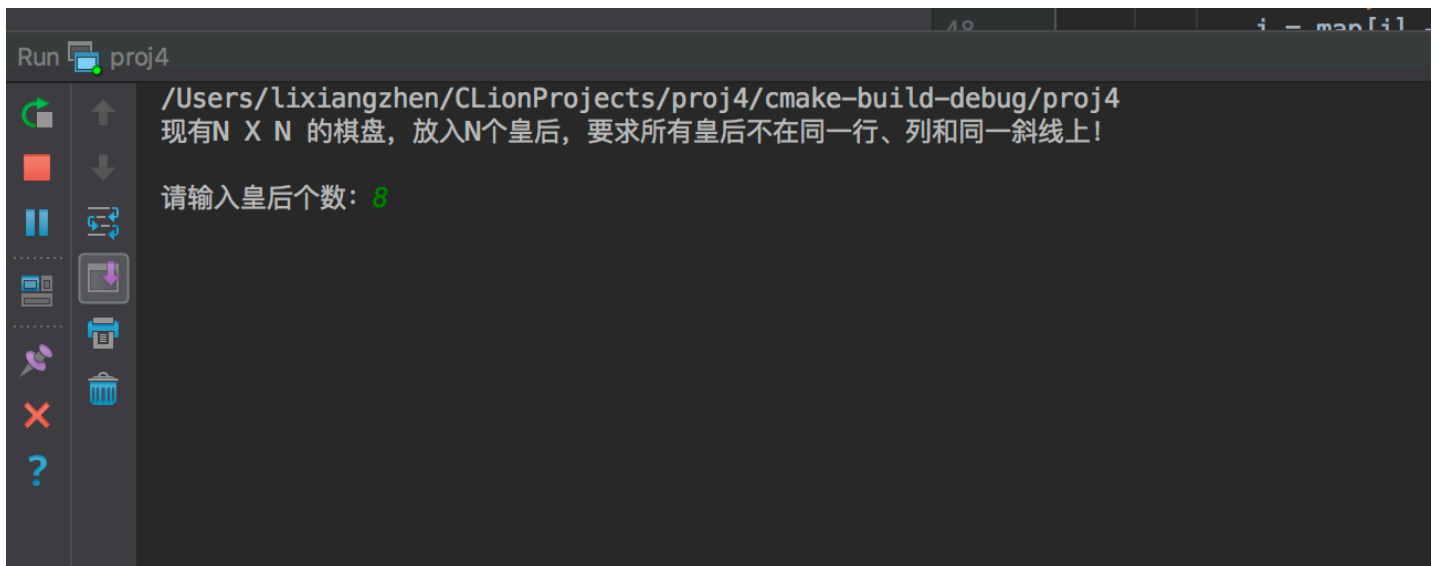
定义数组 `int map[N]` 作为棋盘信息数组，存储每行中摆放皇后的列号。

定义并初始化计数器 `int count = 0` 用于记录生成的方案数量。定义并初始化行号与列号 `int i = 0, j = 0` 为后续回溯做准备。

核心代码如下：

```
int n;
cout<<"现有N X N 的棋盘，放入N个皇后，要求所有皇后不在同一行、列和同一斜线上！\n\n";    /
/ 输出交互提示
cout<<"请输入皇后个数： ";
cin>>n;                                           // 读入地盘规模
cout<<"\n皇后摆法： \n\n";
const int N = n;                                // 定义常量N作为棋盘大小，用于建立辅助数组
int map[N];                                       // 辅助数组，存储每行中摆放皇后的列号
int count = 0;
int i = 0, j = 0;
```

程序执行情况如下：



```
Run proj4
/Users/lixiangzhen/CLionProjects/proj4/cmake-build-debug/proj4
现有N X N 的棋盘，放入N个皇后，要求所有皇后不在同一行、列和同一斜线上！

请输入皇后个数： 8
```

随后进入循环回溯，循环尝试皇后可能的摆放位置。对第*i*行的所有*j*列情况进行尝试。

如果尝试失败，继续尝试该行的下一列，直到 `j == N`

核心代码如下：

```

while(j < N){          // 对第i行的所有j列情况进行尝试
    if(qualified(map,i,j)){      // 判断是否为合法位置
        map[i] = j;            // 如果是合法位置就放置节点
        j = 0;                  // 列号归0
        break;
    }
    else                    // 如果不是合法位置，继续探索下一列
        ++j;
}

```

每次尝试停止后，根据 j 与 i 的值判断当前回溯到了哪一步。

如果 $j == N$ 表明当前行没有合适的位置，进一步查看 i ，如果 $i == 0$ 则表明已经无路可退，算法结束。反之，则 $--i$ 退回前一行，尝试改行之前摆放皇后位置的下一个位置。

核心代码如下：

```

if(j == N){          // 如果当前行没有合法位置
    if(i == 0)        // 如果行号为0，说明已经找到所有情况，循环结束
        break;
    else{             // 如果行号非0，就退回前一行
        --i;
        j = map[i] + 1;    // 探索前一行之前合法位置的下一个
    }
}

```

如果 $i == N - 1$ 表明已经探索到最后一行，并且产生了合适的皇后摆法，则将该

摆法输出，并继续对改行进行探索。

核心代码如下：

```
else if(i == N - 1){           // 如果已经探索到最后一行，说明已经产生一种方案
    printmap(map,N);           // 输出该方案
    ++count;                   // 记录方案个数
    j = map[i] + 1;             // 探索该行下一个位置
}
```

当所有摆法都探索结束后，输出摆法数量，程序退出。

核心代码如下：

```
cout<<"\n共有"<<count<<"种解法！"<<endl;
return 0;
```

2.位置合法性检验

遍历已经摆好的所有位置，比较其与当前位置的关系。由于每个皇后都摆在不同的行，所以只需要对列方向和斜线方向进行检验即可。

核心代码如下：

```

/*
 * 位置合法性检验函数
 * 检验摆放位置是否合法
 * 即其横竖斜向都不存在其他皇后
 * */
bool qualified(int* map,int m,int n){
    for(int i = 0;i < m;++i){          // 遍历已经摆好的每个皇后位置，检查是否有冲突
        if(map[i] == n || abs(m - i) == abs(n - map[i]))
            return false;
    }
    return true;
}

```

关联调用情况如下：

```

27 while(i < N){          // 循环尝试皇后可能的摆放位置
28     while(j < N){      // 对第i行的所有j列情况进行尝试
29         if(qualified(map,i,j)){ // 判断是否为合法位置
30             map[i] = j;      // 如果是合法位置就放置节点
31             j = 0;           // 列号归0
32             break;
33         }

```

3.打印N皇后布局

按照规定格式，打印N皇后的合理布局。
参数传入皇后位置信息已经棋盘规模。

核心代码如下：


```

/*
 * 皇后位置打印函数
 * 根据给定的皇后位置信息
 * 格式化输出该图中皇后摆法
 * */
void printmap(int* map,int n){
    for(int i = 0;i < n;++i){
        for(int j = 0;j < n;++j){           // 格式化输出棋盘各行
            if(map[i] == j)
                cout<<"X ";
            else
                cout<<"0 ";
        }
        cout<<endl;
    }
    cout<<endl;
}

```

关联调用情况如下：

```

43     }
44     }
45     else if(i == N - 1){                // 如果已经探索到最后一行，说明已经产生一种方案
46         printmap(map,N);                // 输出该方案
47         ++count;                         // 记录方案个数
48         j = map[i] + 1;                  // 探索该行下一个位置
49     }
50     else                                // 如果是中间行，则继续探索下一行
51         ++i;
52 }
53 cout<<"\n共有"<<count<<"种解法! "<<endl;
54 return 0;
55 }

```

四.测试

1.基本功能测试

测试用例

皇后数量： 8

程序执行情况如下：

```
Run proj4
/Users/lixiangzhen/CLionProjects/proj4/cmake-build-debug/proj4
现有N X N 的棋盘，放入N个皇后，要求所有皇后不在同一行、列和同一斜线上！

请输入皇后个数： 8

皇后摆法：
X 0 0 0 0 0 0 0
0 0 0 0 X 0 0 0
0 0 0 0 0 0 0 X
0 0 0 0 0 X 0 0
0 0 X 0 0 0 0 0
0 0 0 0 0 0 X 0
0 X 0 0 0 0 0 0
0 0 0 X 0 0 0 0

X 0 0 0 0 0 0 0
0 0 0 0 0 0 X 0
0 0 0 0 0 0 0 X
0 0 X 0 0 0 0 0
0 0 0 0 0 0 X 0
```

```
0 0 0 0 0 X 0 0
0 X 0 0 0 0 0 0
0 0 0 0 X 0 0 0
0 0 0 0 0 0 X 0
0 0 0 X 0 0 0 0

0 0 0 0 0 0 0 X
0 0 0 X 0 0 0 0
X 0 0 0 0 0 0 0
0 0 X 0 0 0 0 0
0 0 0 0 X 0 0 0
0 X 0 0 0 0 0 0
0 0 0 0 0 0 X 0
0 0 0 0 X 0 0 0

共有92种解法！

Process finished with exit code 0
```

测试用例

皇后数量: 11

程序执行情况如下:

13 bool qualified(int* map,int

Run proj4

/Users/lixiangzhen/CLionProjects/proj4/cmake-build-debug/proj4
现有N X N 的棋盘，放入N个皇后，要求所有皇后不在同一行、列和同一斜线上！

请输入皇后个数: 11

皇后摆法:

X 0 0 0 0 0 0 0 0 0 0
0 0 X 0 0 0 0 0 0 0 0
0 0 0 0 X 0 0 0 0 0
0 0 0 0 0 0 X 0 0 0
0 0 0 0 0 0 0 0 X 0
0 0 0 0 0 0 0 0 0 X
0 X 0 0 0 0 0 0 0 0
0 0 0 X 0 0 0 0 0 0
0 0 0 0 0 X 0 0 0 0
0 0 0 0 0 0 0 X 0 0
0 0 0 0 0 0 0 0 X 0

X 0 0 0 0 0 0 0 0 0 0
0 0 X 0 0 0 0 0 0 0 0

0 0 0 0 0 0 X 0 0 0 0
0 X 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 X
0 0 0 0 0 0 0 0 X 0 0
0 0 0 0 0 0 X 0 0 0 0
0 0 0 0 X 0 0 0 0 0 0
0 0 X 0 0 0 0 0 0 0 0
X 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 X
0 0 0 0 0 0 0 X 0 0
0 0 0 0 0 X 0 0 0 0 0
0 0 0 X 0 0 0 0 0 0 0
0 X 0 0 0 0 0 0 0 0 0

共有2680种解法!

Process finished with exit code 0

Process finished with exit code 0