N皇后问题

问题描述

八皇后问题是一个古老而著名的问题，是回溯算法的经典问题。该问题是十九世纪著名的数学家高斯在1850年提出的：在8*8的国际象棋棋盘上，安放8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。

高斯认为有76种方案。1854年在柏林的象棋杂志上不同的作者发表了40种不同的解，后来有人用图论的方法得到结论，有92中摆法。

本实验拓展了N皇后问题，即皇后个数由用户输入。

项目要求

八皇后在棋盘上分布的各种可能的格局数目非常大，约等于2的32次方种，但是，可以将一些明显不满足问题要求的格局排除掉。由于任意两个皇后不能同行，即每行只能放置一个皇后，因此将第i个皇后放在第i行上，这样在放置第i个皇后时，只要考虑它与前i-1个皇后处于不同列和不同对角线位置上即可。

解决这个问题采用回溯法，首先将第一个皇后放置在第一行第一列，然后，依次在下一行上放置一个皇后，直到八个皇后全部放置安全。在放置每个皇后时，都依次兑每一列进行检测，首先检测放在第一列是否与已放置的皇后冲突，如不冲突，则将皇后放置在该列，否则，选择改行的下一列进行检测。如整行的八列都冲突，则回到上一行，重新选择位置，依次类推

算法思想

以栈为数据结构实现的非递归回溯算法。

功能实现及代码分析

1、类设计

定义栈并实现相关成员函数。

```
template<class T>
class stack                  //定义顺序栈模板类
{
public:
    stack() { top = -1; };   //构造函数，初始化一个空栈
    void push(T q);          //入栈
    void pop();              //出栈
    bool judgement();        //判断是否在同一行、同一列、同一斜线
    void display();          //输出棋盘
    bool isEmpty();          //判断栈是否为空
    void placeQueen(int row);//摆皇后的递归函数

private:
    T data[maxSize];         //定义栈的数组
    int top;                 //定义栈顶指针
};
```

2、栈设计

函数	返回值	描述
push(T q)	void	将棋盘的列推入栈
pop()	void	该列不能放下皇后，将栈顶列推出栈
isEmpty()	bool	判断栈是否为空

```
template<class T>
void stack<T>::push(T q) {    //入栈
    if (top >= maxSize - 1) throw "error";
```

```
    else {
        top++;                  //栈顶指针上移
        data[top] = q;
    }
}

template<class T>
void stack<T>::pop() {          //出栈
    if (isEmpty()) throw "error";
    else top--;                 //站定指针下移
}

template<class T>
bool stack<T>::isEmpty() {      //判断栈是否为空
    if (top == -1)return true;
    else return false;
};
```

3、进入项目

进入项目，引导用户输入皇后个数并求解。

```
int N;              //存储皇后个数的变量
int num = 0;        //初始化种数为0

int main() {
    stack<int> queen;

    cout << "现有N*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请输入皇后个数 N:";
    cin >> N;                                   //输入皇后的个数
    queen.placeQueen(0);                        //开始放置
    cout << "共有" << num << "种解法!" << endl;  //输出解法总数
    return 0;
}
```

4、求解

遍历棋盘

遍历棋盘，找到点后入栈，判断是否合法，若合法，到下一行继续搜索，直到得到完整方案并输出结果，若不合法，将该点出栈，继续遍历棋盘到下一个点。直接进入下一行搜索，相当于让该行的位置都标记为被占领状态，可以简化搜索的过程。

```
template<class T>
void stack<T>::placeQueen(int row) {
    for (int i = 0; i < N; i++)     //对棋盘的列进行遍历
    {
        push(i);                    //入栈
        if (judgement())            //判断能否放下
        {
            if (row < N - 1)        //是否是最后一个皇后
                placeQueen(row + 1);//还未到最后一个皇后，  则递归
            else {
                num++;              //方案个数计数加一
                display();          //打印棋盘
            }
        }
        pop();                      //若不能放下则出栈
    }
}
```

判断函数

通过遍历并比较与之前皇后的行列值差值，判断该落点是否合法，若合法返回true继续搜索，若不合法返回false将该点出栈。

```
template<class T>
bool stack<T>::judgement() {
    for (int i = 0; i < top; i++)//遍历前面各个皇后的位置
        if (data[top] == data[i] || abs(data[top] - data[i]) == top - i)//如果列值相等或者行差与列差相等
            return false;
    return true;
}
```

5、输出结果

```
template<class T>
void stack<T>::display() {
    cout << "第" << num << "种解法:" << endl;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < data[i]; j++)
            cout << "0\t";      //不放置皇后
        cout << "x\t";          //放置皇后
        for (int j = N - 1; j > data[i]; j--)
            cout << "0\t";      //不放置
        cout << '\n' << endl;
    }
    cout << endl;
}
```

用例演示

```
现有N*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 N:6
第1种解法:
0   x   0   0   0   0

0   0   0   x   0   0

0   0   0   0   0   x

x   0   0   0   0   0

0   0   x   0   0   0

0   0   0   0   x   0


第2种解法:
0   0   x   0   0   0

0   0   0   0   0   x

0   x   0   0   0   0

0   0   0   0   x   0

x   0   0   0   0   0

0   0   0   x   0   0


第3种解法:
0   0   0   x   0   0

x   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   x   0   0   0


第4种解法:
0   0   0   0   x   0

0   0   x   0   0   0

x   0   0   0   0   0

0   0   0   0   0   x

0   0   0   x   0   0
```

```
0   x   0   0   0   0


共有4种解法!

Process finished with exit code 0
```