



南京邮电大学
Nanjing University of Posts and Telecommunications

《算法设计与分析》课内实验报告

实验题目：回溯法

实验日期：2024 年 1 月 3 日

指导教师：张博

学生姓名：单家俊

学生学号：B21080526

实验成绩：

一、实验目的

本实验的主要目的是通过学习和实践，掌握回溯法在解决 24 点问题或 N 皇后问题中的应用。具体目标包括：

1. 理解回溯法的基本思想
2. 掌握在算法设计中如何使用回溯法解决具体问题
3. 通过编写代码实现，加深对回溯法的理解
4. 分析回溯法在不同问题中的应用效果

二、实验内容

用回溯法求解 N 皇后问题。

三、实验过程描述

算法思想：

要解决 N 皇后问题，其实就是要解决好怎么放置这 n 个皇后，每一个皇后与前面的所有皇后不能在同一行、同一列、同一对角线，在这里我们可以以行优先，就是说皇后的行号按顺序递增，只考虑第 i 个皇后放置在第 i 行的哪一列，所以在放置第 i 个皇后的时候，可以从第 1 列判断起，如果可以放置在第 1 个位置，则跳到下一行放置下一个皇后。如果不能，则跳到下一列...直到最后一列，如果最后一列也不能放置，则说明此时放置方法出错，则回到上一个皇后向之前放置的下一列重新放置。此即是回溯法的精髓所在。当第 n 个皇后放置成功后，即得到一个可行解，此时再回到上一个皇后重新放置寻找下一个可行解...如此后，即可找出一个 n 皇后问题的所有可行解。

计算时间复杂度：

在 N 皇后问题中，因为每个节点（最下一层除外）都要遍历包括当前层在内的上面每一层的选择，才能得到当前可行的全部直接子孙节点，且绝大多数节点都集中在层数靠下的位置，故可以估计出平均每个节点的生成时间复杂度 $p(n)$ 都是 n。

例如上例中，根节点生成时要检验它的 8 个真子孙节点是否可行；对于第一层的每一个节点，也都需要判断其全部的 8 个真子孙节点是否可行；第二层、第三层 直到倒数第二层，也都需要判断其全部的 8 个真子孙节点是否可行；最后一层虽比较特殊，无需判断，但最后一层（已经抵达可行解）的节点在 N 皇后问题（排列树）中和上一层的节点数相同，故与其他节点一视同仁也不影响时间复杂度的计算。

所以 N 皇后的时间复杂度为 $O(n \times \text{实际生成的节点数})$ 。

代码：

```
#include <iostream>
using namespace std;
```

```

#include <math.h>
#include <vector>

static int ii = 0;

bool Place(int k, int i, int* x) //判定两个皇后是否在同一列或在同一斜线上
{
    for (int j = 0; j < k; j++)
        if ((x[j] == i) || (abs(x[j] - i) == abs(j - k))) return false;
    return true;
}

void NQueens(int k, int n, int* x) //递归函数（求解 n 皇后问题）
{
    for (int i = 0; i < n; i++)
    {
        if (Place(k, i, x))
        {
            x[k] = i;
            if (k == n - 1)
            {
                ii++;
                cout << "(" << ii << ") ";
                for (i = 0; i < n; i++) {
                    cout << x[i] << " ";
                }
                cout << endl;
            }
            else
            {
                NQueens(k + 1, n, x);
            }
        }
    }
}

void NQueens(int n, int* x)
{
    NQueens(0, n, x);
}

int main()
{
    int x4[4]; // 用于解决 4 皇后问题
    int x8[8]; // 用于解决 8 皇后问题

```

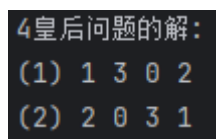
```
// 解决 4 皇后问题
cout << "4 皇后问题的解:\n";
for (int i = 0; i < 4; i++) x4[i] = -1;
NQueens(4, x4); // 将棋盘大小作为 n 传递

// 解决 8 皇后问题
cout << "\n8 皇后问题的解:\n";
for (int i = 0; i < 8; i++) x8[i] = -1;
ii = 0;
NQueens(8, x8); // 将棋盘大小作为 n 传递

return 0;
}
```

四、实验结果

四皇后问题：



```
4皇后问题的解：
(1) 1 3 0 2
(2) 2 0 3 1
```

八皇后问题：

部分解如下

```
(58) 4 6 0 3 1 7 5 2
(59) 4 6 1 3 7 0 2 5
(60) 4 6 1 5 2 0 3 7
(61) 4 6 1 5 2 0 7 3
(62) 4 6 3 0 2 7 5 1
(63) 4 7 3 0 2 5 1 6
(64) 4 7 3 0 6 1 5 2
(65) 5 0 4 1 7 2 6 3
(66) 5 1 6 0 2 4 7 3
(67) 5 1 6 0 3 7 4 2
(68) 5 2 0 6 4 7 1 3
(69) 5 2 0 7 3 1 6 4
(70) 5 2 0 7 4 1 3 6
(71) 5 2 4 6 0 3 1 7
(72) 5 2 4 7 0 3 1 6
(73) 5 2 6 1 3 7 0 4
(74) 5 2 6 1 7 4 0 3
(75) 5 2 6 3 0 7 1 4
(76) 5 3 0 4 7 1 6 2
(77) 5 3 1 7 4 6 0 2
(78) 5 3 6 0 2 4 1 7
(79) 5 3 6 0 7 1 4 2
(80) 5 7 1 3 0 6 4 2
(81) 6 0 2 7 5 3 1 4
(82) 6 1 3 0 7 4 2 5
(83) 6 1 5 2 0 3 7 4
(84) 6 2 0 5 7 4 1 3
(85) 6 2 7 1 4 0 5 3
(86) 6 3 1 4 7 0 2 5
(87) 6 3 1 7 5 0 2 4
(88) 6 4 2 0 5 7 1 3
(89) 7 1 3 0 6 4 2 5
(90) 7 1 4 2 0 6 3 5
(91) 7 2 0 5 1 4 6 3
(92) 7 3 0 2 5 1 6 4
```

五、实验小结

在此次实验过程中,运用回溯法通过搜索状态空间树中的每个问题状态来求解一个或全部可解的算法思想实现了 N 皇后求解的问题,学会了分析问题的具体特征并设计算法,基本了解运用回溯法的算法思想求解实际问题的具体实现操作,受益匪浅。