**暨南大学本科实验报告专用纸**

课程名称 算法分析与设计实验 成绩评定

实验项目名称 指导教师 李展 实验地点 N116

实验项目编号 5-0 实验项目类型 综合型

学生姓名 张瑞鹏 学号 2020101124

学院 信息科学技术学院 系 计算机 专业 计算机科学与技术

实验时间 2022 年 月 日 午～ 月 日 下 午

**（一）问题描述**

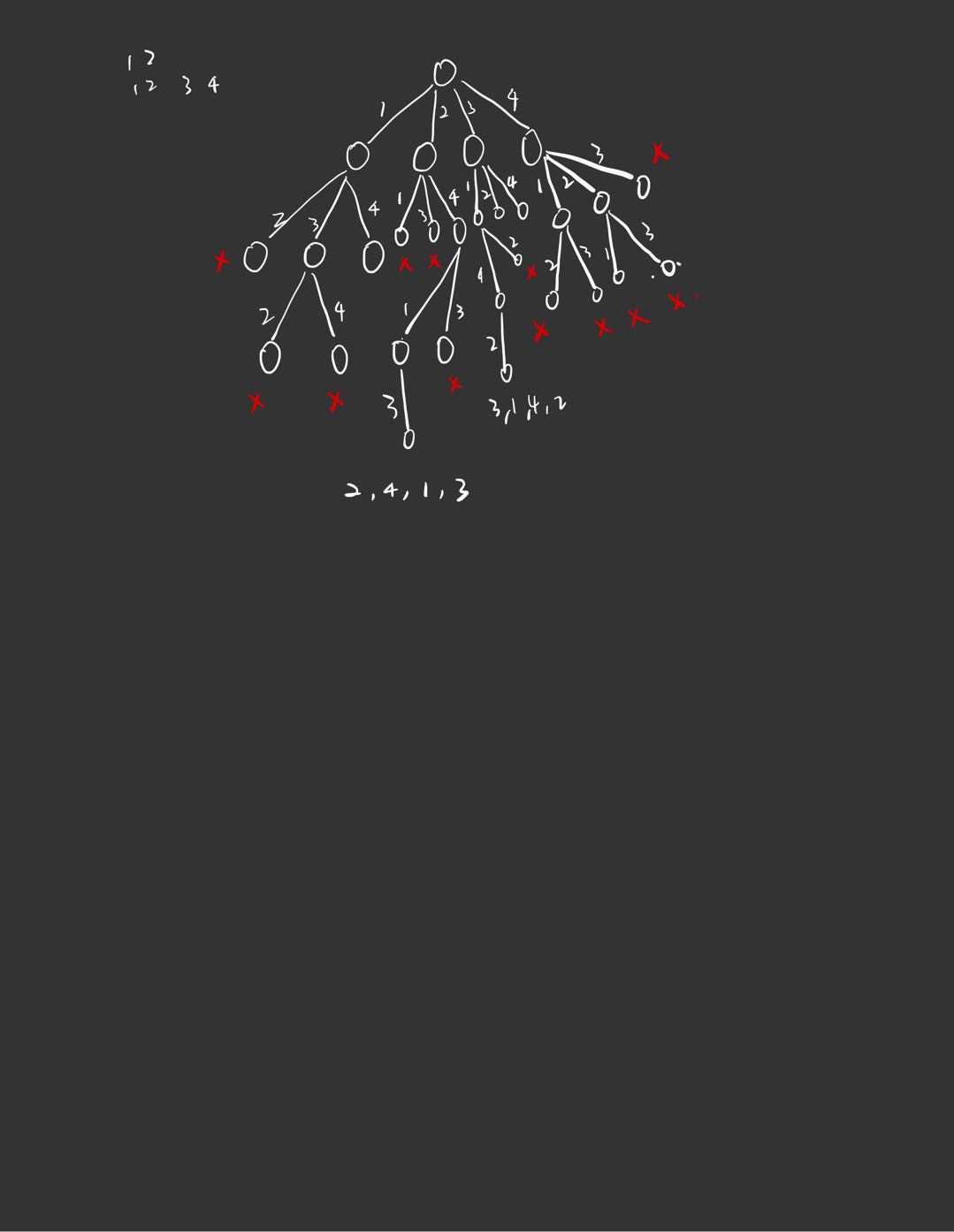
**N皇后问题用排列树的方式完成，其中包括了递归方式与非递归的方式**

**（二）算法思路**（用文字简单说明）

**回溯法，俗称暴力搜索。用排列树的方法的话实际上是枚举全排列，因为皇后不可能在同一列，因此列数不可能一样因此可以使用全排列的方式来做这道题目。**

**递归的思路很简单，对于一个顶点，for循环每一种选择方式，递归进入子树，直到出口，能够到达出口的，实际上已经是最优解，因为设计了减枝函数。**

**非递归的思路大同小异，也是深度优先搜索，不过需要定义一个t表示目前在第几层，并经常需要修改且回退，最后退到t<1结束算法，还需要另外设计一个数组来存储父亲节点上一次swap到哪里了，因为算法是逐个逐个的往后swap来产生排列数，因此需要存储步骤在哪。**



**（三）算法实施步骤和流程**（伪代码/流程图等方式描述）

**递归：**

void QueenBack(int t,int n,int \*x,int &result) {

if (t == n) {

if (place(x, n)) {

result++;

for (int i = 1; i <= n; i++) {

cout << x[i] << " ";

}

cout << endl;

}

}

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

swap(x, t, i);

if (place(x,t)) QueenBack(t + 1, n, x, result);

swap(x, i, t);//现场还原

}

}

**非递归：**

void QueenBack(int t, int n, int\* x, int& result) {

int help[n] = {0};

while (t >= 1) {

help[t] += 1;

if(help[t] <= n) swap(x, t, help[t]);

while (place(x, t) == 0 && help[t] <= n) {

swap(x, t, help[t]);

help[t] += 1;

if (help[t] <= n) swap(x, t, help[t]);

}

if (help[t]<=n) {

if (t == n) {

result++;

for (int i = 1; i <= n; i++) {

cout << x[i] << " ";

}

cout << endl;

t--;

swap(x, t, help[t]);

}

else {

t++;

help[t] = t - 1;

}

}

else {

t--;

if(t>=1) swap(x, t, help[t]);

}

}

return;

}

**（四）源代码**（通过了编译运行的正确程序）

#include<iostream>

using namespace std;

void swap(int\* x, int from, int to) {

//cout << endl <<"检查：" << from <<"\t" << to << endl;

int temp = x[to];

x[to] = x[from];

x[from] = temp;

}//交换函数

bool place(int\* x, int t) {

for (int i = 1; i < t; i++) {

if (abs(t - i) == abs(x[t] - x[i])) return false;

}//逐个检查对角线

return true;

}

void QueenBack(int t, int n, int\* x, int& result) {

int help[n] = {0};

while (t >= 1) {

help[t] += 1;

if(help[t] <= n) swap(x, t, help[t]);

while (place(x, t) == 0 && help[t] <= n) {

swap(x, t, help[t]);

help[t] += 1;

if (help[t] <= n) swap(x, t, help[t]);

}

if (help[t]<=n) {

if (t == n) {

result++;

for (int i = 1; i <= n; i++) {

cout << x[i] << " ";

}

cout << endl;

t--;

swap(x, t, help[t]);

}

else {

t++;

help[t] = t - 1;

}

}

else {

t--;

if(t>=1) swap(x, t, help[t]);

}

}

return;

}

int main(void) {

int n = 10;

int x[11] = { 0,1,2,3,4,5,6,7,8,9,10 };

int result = 0;

QueenBack(1, n, x, result);

if (result) {

cout << "the result is:" << result << endl;

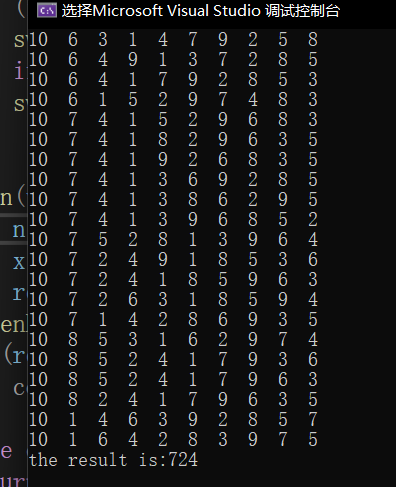
}

else cout << "no result" << endl;

return 0;

}

**（五）测试结果**（至少有两个以上算例及程序运行结果，截图贴进实验报告）

10个皇后

 4个皇后

**（六）实验总结**（至少三句话，可以写复杂度分析、遇到问题、可能的改进措施、心得体会等）

1.回溯算法的技术在于确定剪枝函数的设计。回溯法大致可以分成两种，一种是排列树，一种是子集树，而且子集树又包括了排列树，总体而言，子集树更加的一般。排列树相当于子集树又增加了一项限制，主要体现在最后一步更为复杂。

2.尝试了非递归写法，非递归的难度更加大，因为要更加熟悉模拟计算机压栈的过程，定义的t其实相当于一个栈，经常会出现++，--的操作，意思代表的就是栈的移动，整个设计思路都不太一样，难点在于需要分情况来恢复。

3.时间复杂度方面是O（n！）是一个相当大的时间复杂度。这是回溯的特点，复杂度感觉都会非常大，但是一般使用回溯的算法感觉都是比较复杂的算法，不具有最优子结构，甚至很多已经被证明是非线性时间能够解决的问题都可以解决，所以相对而言回溯其实实用性还是挺广的。

4.n皇后问题非常经典，排列树，子集树，递归非递归，这里就已经有4种方式了，4种方式都有写过，最简单的写法感觉就是递归子集树，最为复杂的应该就是非递归的排列树了。

5.还要多尝试非递归的算法，这种算法对整个理解更为透彻，感觉更加没有模板，很多东西都需要随机改变。

**暨南大学本科实验报告专用纸(附页)**