

北京科技大学实验报告

学院： 计算机与通信工程学院

专业： 计算机科学与技术

班级： 计 184

姓名： 王丹琳

学号： 41824179

实验日期： 2020 年 12 月 13 日

1. 实验名称：

N 皇后问题

2. 实验目的：

在 $N \times N$ 格的国际象棋上摆放 N 个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，求：

1. 给定一个 N ，输出满足 N 皇后条件的摆法数。
2. 给定一个 N ，输出满足 N 皇后条件的其中一种摆法。
3. 给定一个棋盘，棋盘上每格都有一个数字，求一种摆法，使得八皇后所在格子数字之和最大。

3. 实验环境搭建：学生填写，主要包括编程语言和华为云平台实验环境搭建的过程

3.1 实验环境：

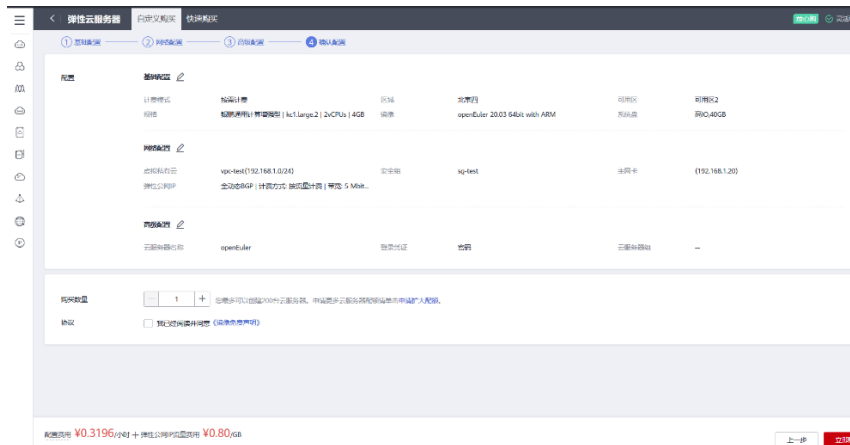
编程语言：C++

环境：华为云平台

3.2 搭建过程：

3.2.1 购买云服务器：

“自定义购买-4 确认配置”界面的截图



购买的服务器状态信息的截图

名称/ID	监控	可用...	状态	规格/镜像	IP地址	计费...	标签	操作
<input checked="" type="checkbox"/> openEuler 1f5a9bcc-fb2c-47...		可用区2	运行中	2vCPUs 4GB kc... openEuler 20.03 ...	121.36.3.2... 192.168.1....	按需计费 2020/12/1...	--	远程登录 更多 ▾
<input type="checkbox"/> ecs-67cd 7f1837dc-5b5a-4...		可用区1	关机	4vCPUs 8GB s6... CentOS 7.6 64bit	121.36.65... 192.168.0....	按需计费 2020/09/2...	--	远程登录 更多 ▾

< openEuler

开机 关机 重启 远程登录 更多 ▾

基本信息 云硬盘 网卡 安全组 弹性公网IP 监控 标签

云服务器信息

ID

1f5a9bcc-fb2c-478b-84cd-0bcfde11a116

名称

openEuler

区域

北京四

可用区

可用区2

规格

鲲鹏通用计算增强型 | kc1.large.2 | 2vCPUs | 4GB

镜像

openEuler 20.03 64bit with ARM

虚拟私有云

vpc-test

计费信息

计费模式

按需计费

创建时间

2020/12/12 21:44:08 GMT+08:00

启动时间

2020/12/12 21:44:47 GMT+08:00

管理信息

云服务器组

-- 新建云服务器组

关机

监控 监控中

云硬盘

系统盘

openEuler-volu... 高IO | 40 GB

网卡

主网卡

subnet-test 192.168.1.20 | 121.36.3.240

安全组

sg-test

弹性公网IP

121.36.3.240 | 5 Mbit/s

3.2.2 环境登录验证

成功登录华为云服务器界面的截图

```
121.36.3.240 - PuTTY
login as: root
Pre-authentication banner message from server:
| Authorized users only. All activities may be monitored and reported.
| End of banner message from server
root@121.36.3.240's password:
Welcome to Huawei Cloud Service

Last login: Mon May 18 15:35:37 2020

Welcome to 4.19.90-2003.4.0.0036.oel.aarch64

System information as of time: Sat Dec 12 23:09:19 CST 2020

System load: 0.51
Processes: 114
Memory used: 9.7%
Swap used: 0.0%
Usage On: 9%
IP address: 192.168.1.20
Users online: 1

[root@openeuler ~]# gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/aarch64-linux-gnu/7.3.0/lto-wrapper
Target: aarch64-linux-gnu
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --enable-shared --enable-threads=posix --enable-checking=release --with-system-zlib --enable-_cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-linker-build-id --with-linker-hash-style=gnu --enable-languages=c,c++,objc,obj-c++,fortran,lto --enable-plugin --enable-initfini-array --disable-libgcj --without-isl --without-cloog --enable-gnu-indirect-function --build=aarch64-linux-gnu --with-stagel-ldflags=' -Wl,-z,relro,-z,now' --with-bootstrap-ldflags=' -Wl,-z,relro,-z,now' --with-multilib-list=lp64
Thread model: posix
gcc version 7.3.0 (GCC)
[root@openeuler ~]# g++ -v
Using built-in specs.
COLLECT_GCC=g++
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/aarch64-linux-gnu/7.3.0/lto-wrapper
```

3.2.3 环境成功搭建验证

运行实验手册中的 1.3 C++语言开发斐波那契数列程序，验证环境搭建成功，实验的成功运行截图如下：

```
[root@openeuler test]# ./test
please input month:1
result =1
3
result =2
6
result =8
█
```

4. 实验原理：

本题 N 皇后问题为回溯思想的典型例题，所涉及的知识点也就是回溯法，回溯算法实际上就是一个类似枚举的搜索尝试方法，若不满足条件就“回溯”返回，尝试别的路径。区别于枚举法的地方在于：在搜索过程中，会采用约束方法或剪去得不到最优解的子树来避免无效搜索。在本题中，可以发现约束条件为不同列和不同正反对角线，用这一约束条件可以避免许多无效的枚举，提高效率。

5. 实验内容与步骤：

5.1 问题分析

根据题意，很明显本题所用到的思想为回溯思想。在试探过程中，我们可以将试探结果存放于一个大小为 N 的一维数组 s 中，其中下标 i 代表了棋盘第 i 行（第 i 个皇后），将这个皇后所处于行数保存在相应的数组位置 s[i] 中。在试探第 i 个皇后能否放时所用的约束条件为不同列 $s[i] \neq s[k]$ ($0 < k < i$) 和不同正反对角线 $|i - k| \neq |s[i] - s[k]|$ ($0 < k < i$)。若放到最后一个皇后还有位置能放下的话，则为一解；若还没放到最后一个，且有位置能放，则放下，继续放下一个皇后；若已经没地方能放了，则将该皇后拿出，回溯到上一个皇后，尝试别的路径。可以看出，N 皇后的尝试摆法实际上就是一个类似枚举的搜索尝试方法。若要求一种摆法，使得八皇后所在格子数字之和最大，则再加一个暂存 max 的一个数组（用来存放使八皇后所在格子数字之和最大的放法）。除此之外，算法上还需要在一种放法成功的时候加上和的比较，若结果大于之前的和，则更新最大值和最大数组 max。

5.2 算法设计

5.2.1 问题 1：给定一个 N，输出满足 N 皇后条件的摆法数。

```

num = 0;
for (int i = 1; i <= n; i++)
    x[i] = 0;
int k = 1;
while (k>=1)//(k, x[k])
{
    x[k] += 1;
    while (x[k] <= n && !Place(k))//找第k行所能放的位置
        x[k] = x[k] + 1;
    if (x[k] <= n && k == n) { //放完了
        num++;
    }
    else if (x[k] <= n && k < n) //还没全部放完
        k = k + 1;
    else {
        x[k] = 0;
        k = k - 1;
    }
}
cout << endl << "总计一共有" << num << "摆法";

```

5.2.2 问题 2: 给定一个 N，输出满足 N 皇后条件的其中一种摆法。

```

for (int i = 1; i <= n; i++)
    x[i] = 0;
k = 1;
while (k>=1)//(k, x[k])
{
    x[k] += 1;
    while (x[k] <= n && !Place(k))//找第k行所能放的位置
        x[k] = x[k] + 1;
    if (x[k] <= n && k == n) { //放完了
        //输出
        cout << "其中一种摆法为";
        show(); //输出一种摆法
        return; //返回
    }
    else if (x[k] <= n && k < n) //还没全部放完
        k = k + 1;
    else {
        x[k] = 0;
        k = k - 1;
    }
}

```

5.2.3 问题 3: 给定一个棋盘，棋盘上每格都有一个数字，求一种摆法，使得八

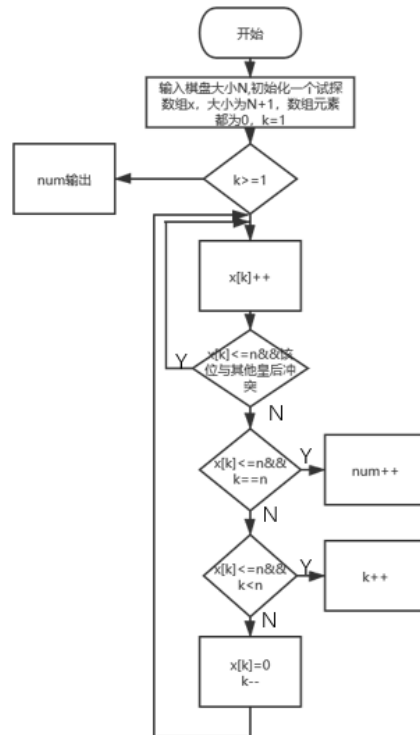
皇后所在格子数字之和最大。

```
max = 0;
for (int i = 1; i <= n; i++)
    x[i] = 0;
k = 1;
while (k>=1)//(k, x[k])
{
    x[k] += 1;
    while (x[k] <= n && !Place(k))//找第k行所能放的位置
        x[k] = x[k] + 1;
    if (x[k] <= n && k == n) { //放完了
        for (int i = 1; i <= n; i++)
            sum += eight[i][x[i]];
        if (sum > max) {
            max = sum;

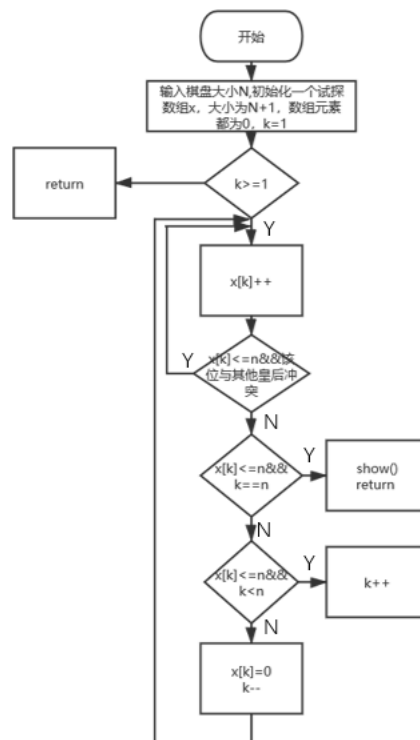
            for (int i = 1; i <= n; i++)
                max_eight[i] = x[i];
        }
        sum = 0;
    }
    else if (x[k] <= n && k < n) //还没全部放完
        k = k + 1;
    else {
        x[k] = 0;
        k = k - 1;
    }
}
cout << endl << "sum_max=" << max;
```

5.3 算法流程图

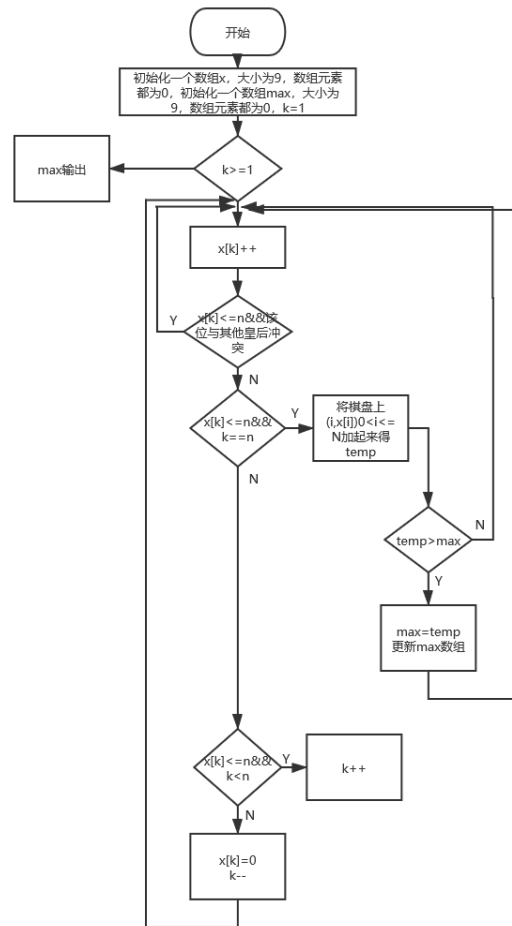
5.3.1 问题 1: 给定一个 N, 输出满足 N 皇后条件的摆法数。



5.3.2 问题 2: 给定一个 N, 输出满足 N 皇后条件的其中一种摆法。



5.3.3 问题 3: 给定一个棋盘, 棋盘上每格都有一个数字, 求一种摆法, 使得八皇后所在格子数字之和最大。



5.4 算法实现

```

bool Place(int k) { //判断
    for (int i = 1; i < k; i++) {
        if (x[k] == x[i] || abs(k - i) == abs(x[k] - x[i]))
            return false;
    }
    return true;
}

```

```

void show(int n) { //打印
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++)
            if (j != x[i]) cout << "-";
            else cout << "o";
        cout << endl;
    }
}

```

5.5 实验测试方案

实验测试方案：

输入 4，即棋盘大小为 4，放置 4 个皇后。经过计算可以得出只有 2 种放置方法。其中一种放置方法为

-0--
---0
0---
--0-

输入 8，即棋盘大小为 8，放置 8 个皇后。经过计算可以得出只有 92 种放置方法。其中一种放置方法为

0-----
----0---
-----0
----0--
--0-----
-----0-
-0-----
---0----

实验测试数据：

输入 4，预期输出：放置方法为 2，其中一种放置方法为 2 4 1 3

输入 8，预期输出：放置方法为 92，其中一种放置方法为 1 5 8 6 3 7 2 4

5.6 实验数据：

假定给定的 8*8 棋盘，棋盘上每格的数字：

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

样例输入 1

8

样例输出 1

One of the ways:
0-----
----0---
-----0


```

-----O--
--O-----
-----O-
-O-----
---O-----
sum_max=260
There are 92 ways in total.

```

5.7 实验数据处理：

输入 8，即棋盘大小为 8，放置 8 个皇后。经过一番努力，可以得出有 92 种放置方法。其中一种放置方法为

```

O-----
---O---
-----O
----O--
--O-----
-----O-
-O-----
---O---

```

且最大值为 260。

6. 实验结果与分析

6.1 测评结果

```

[root@openeuler test]# g++ -mabi=lp64 -march=armv8-a -o test queen.cpp
[root@openeuler test]# ./test
4
One of the ways:
-O--
---O
O---
--O-
There are 2 ways in total.

```

```

[root@openeuler test]# g++ -mabi=lp64 -march=armv8-a -o test queen.cpp
[root@openeuler test]# ./test
8
One of the ways:
O-----
---O---
-----O
----O--
--O-----
-----O-
-O-----
---O---
sum_max=260
There are 92 ways in total.

```

6.2 实验结果和算法分析

对于测试结果的分析：测试结果符合预期

时间复杂度和空间复杂度计算：时间复杂度为 $O(n!)$ ，因为用了一维数组来存放皇后所在位置，所以空间复杂度为 $O(n)$

7. 实验总结：

通过这次实验，我深刻理解了回溯算法的基本思想，与枚举法不同，回溯算法在遇到不满足条件的情况下就会“回溯”，尝试别的路径。由此，效率就大大提升。回溯法其实就像遍历一个树，树的根节点位于第一层（搜索的初始状态），第二层就相当于对于第一层做出选择后的第二个状态，以此类推，树到叶子的路径就是一个解。在遇到不满足约束条件时，就进行剪枝，减少无意义的遍历。

同时，在运用回溯算法进行解题时，我也熟悉了回溯算法的算法框架：

主算法：1.X{}

2.flag=false;

3.advance(1);

4.if(flag)输出解 X;

else 输出“无解”

advance(int k){

for(x_i, x_i 属于 S_k){

$x_k = x_i$;

将 x_k 加入 X;

if(X 为最终解){flag=true;return;}

else if(X 为部分解)advance(k+1);

}

}