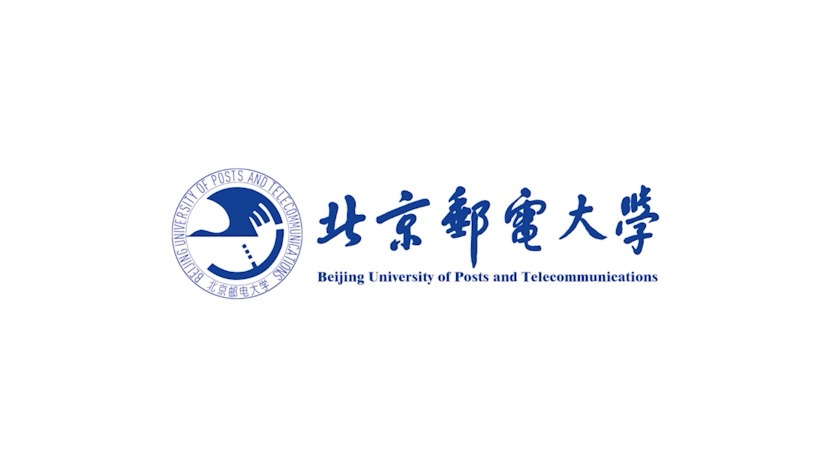
**算法设计与分析实验报告**



实验题目： 循环赛日程表算法的设计与分析

姓名： 马天成

学号： 2020211376

日期： 2022-11-24

目录

[一、实验环境 3](#_Toc118318307)

[1.1 设备规格 3](#_Toc118318308)

[1.2 操作系统 3](#_Toc118318309)

[1.3 编程语言&编译器 3](#_Toc118318310)

[1.4 开发工具 4](#_Toc118318311)

[二、实验内容 4](#_Toc118318312)

[2.1 实验目的 4](#_Toc118318313)

[2.2 实验内容及要求 4](#_Toc118318314)

[2.3 主函数功能设计 5](#_Toc118318315)

[2.4 子函数-记录分治归并路径 5](#_Toc118318316)

[2.4 子函数-Combine 6](#_Toc118318317)

[2.4.1 准备工作 6](#_Toc118318318)

[2.4.2 偶数归并成偶数 6](#_Toc118318319)

[2.4.3 奇数归并成偶数 7](#_Toc118318320)

[2.4.4 生成奇数结果矩阵 8](#_Toc118318321)

[2.4.5 combine逻辑 8](#_Toc118318322)

[2.4.6 输出函数 9](#_Toc118318323)

[2.5 正确性检验 9](#_Toc118318324)

[2.5.1 白盒测试 9](#_Toc118318325)

[2.5.2 黑盒测试 10](#_Toc118318326)

[三、出现问题及解决 13](#_Toc118318327)

[3.1 子函数逻辑思路理清 13](#_Toc118318328)

[3.2 奇数矩阵combine成偶数矩阵 13](#_Toc118318329)

[四、总结 13](#_Toc118318330)

[4.1 时间复杂度 13](#_Toc118318331)

[4.2 空间复杂度 14](#_Toc118318332)

[4.3 算法效率 14](#_Toc118318333)

[4.4 生成序列的规整性 14](#_Toc118318334)

[4.5 理解与思考 14](#_Toc118318335)

# 一、实验环境

## 1.1 设备规格

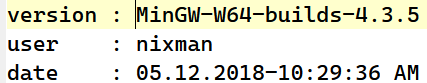


## 1.2 操作系统

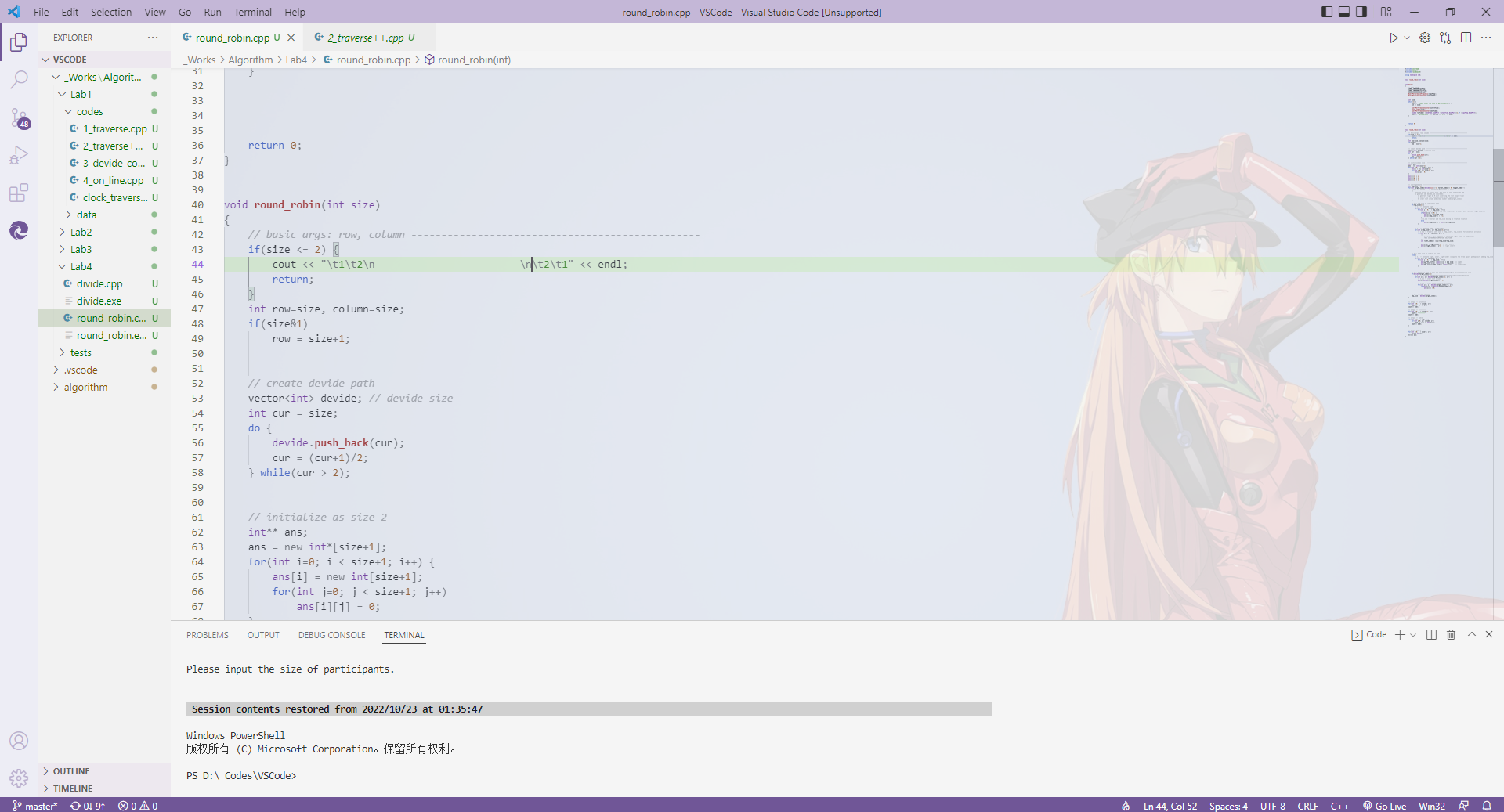


## 1.3 编程语言&编译器

C++11



## 1.4 开发工具



# 二、实验内容

## 2.1 实验目的

⚫ 理解分治法的策略，掌握基于递归的分治算法的实现方法；

⚫ 掌握基于数学模型建立算法模型的建模方法；

⚫ 理解并掌握在渐进意义下的算法复杂性的评价方法。

## 2.2 实验内容及要求

* 算法的设计与实现

问题描述 设有 n 个运动员要进行网球循环赛，设计一个满足下列条件的比 赛日程表： 1）每个选手必须与其他 n-1 个选手各赛一次； 2）每个选手一天只能赛一次 3）当 n 是偶数时，循环赛只能进行 n-1 天 4）当 n 是奇数时，循环赛只能进行 n 天测试要求

依据数学方法，解决选手人数不等于 2 k时，在偶数和奇数情况下， 依题目条件建立算法模型。

① 数据生成： 不同规模的数据集，用于测试算法的正确性及效率。

② 算法实现：实现能够满足题目要求的循环赛日程表算法及程序。

设计测试数据集，编写测试程序，用于测试：

a) 正确性：所实现算法的正确性；

b) 算法复杂性：分析评价各个算法在算法复杂性上的表现；（最差情 况、平均情况）

## 2.3 主函数功能设计



很简单的，就是进入一个while（1）死循环，然后从函数开始记录时间，运行完输出答案后进行时间统计，输出运行时间。

2.4 子函数-记录分治归并路径

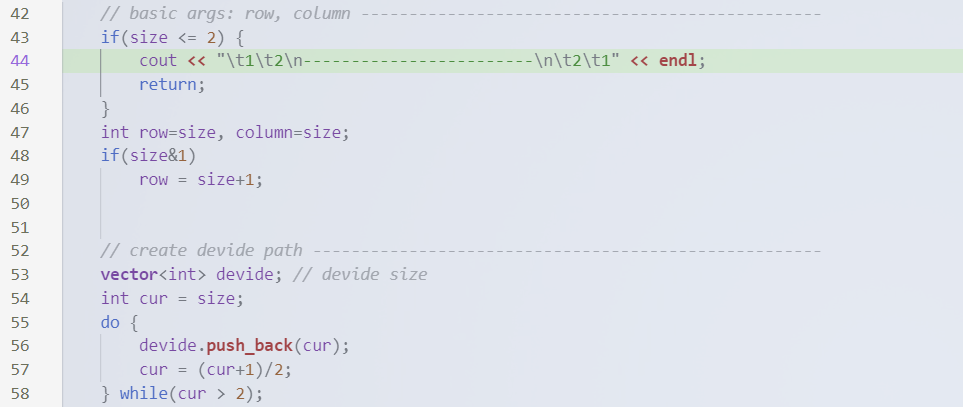
根据最简单的算法设计来说，我们合成最后的循环赛表其实就是为了将小的合成大的。 但是从上到下的分治需要递归调用。我们并不打算使用递归，要用消除递归的思想俩进行combine。这一步我们做的就是记录将小的合成大的路径，方便在每一步的combine中控制规模，成功合成需要的最终答案。

以下步骤是获得比赛人数并生成分治路径的过程：

1. 获得size（参加比赛的人数）。假如小于等于2则直接输出表格，没有必要进行输出。

2. 根据size进行分治，用vector存储需要合成的循环算赛人数大小。直到最后的cur（下一个需要存入的路径，即人数size）**小于等于2，到达初态，结束存储路径**。

*ep：size=13，存储 13 – 7 – 4 （combine从2->4->7->13）*



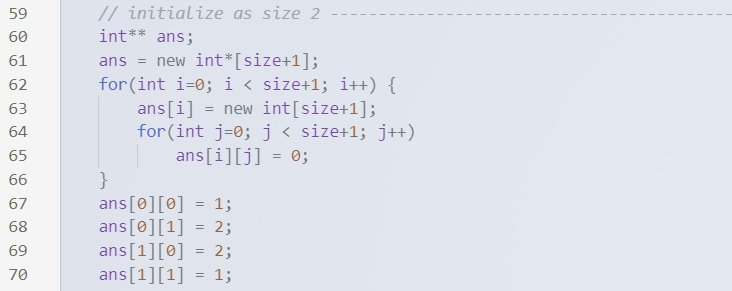
## 2.4 子函数-Combine

这里的归并是该算法最难的地方。对于偶数，这里非常简单。但是对于奇数，我们会显得非常复杂。在这里我也是花了4个小时进行代码编写，然后1个小时修改代码。

### 2.4.1 准备工作

进行了size的输入以及分治路径的生成，接下来就是准备工作。

首先生成最开始的存储规模。这里设定为size+1大小的矩阵。因为在奇数最后一步combine的时候会进行额外的一列的使用，所使用该空间。

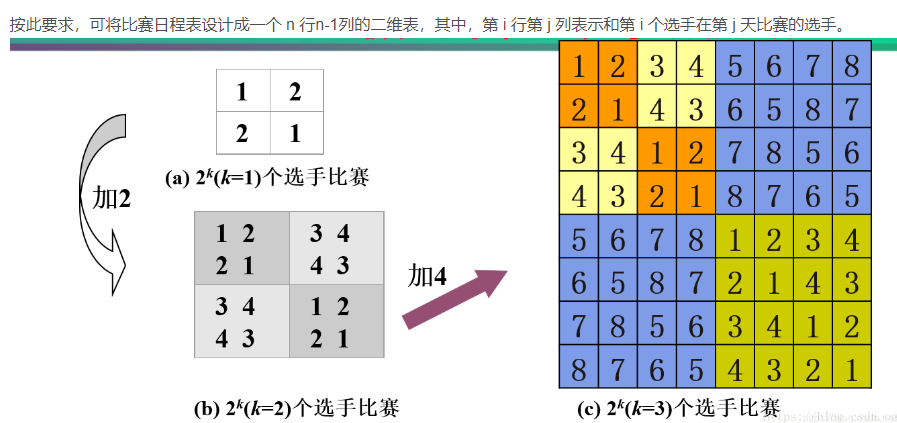


然后初始化为：

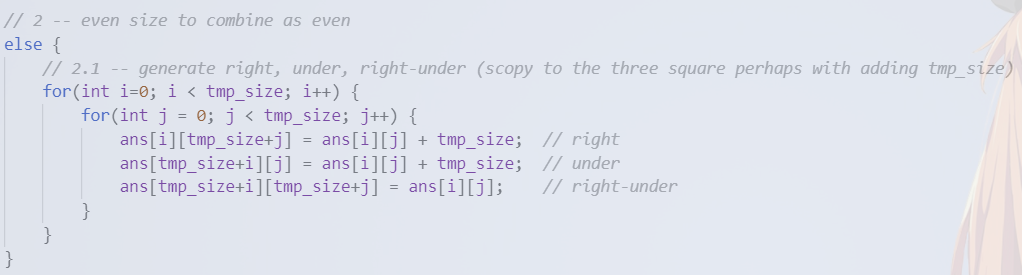
1 2

2 1

### 2.4.2 偶数归并成偶数



这就是非常简单的归并方式。将现有分区平移到另外四块，加上相应的size就可以了。



这里就是非常简单的生成剩下三个区域。

### 2.4.3 奇数归并成偶数

首先大体思路也是和奇数一样，归并成一个方块。但是这里面有太多的细节要讲。

**生成奇数要先生成偶数**

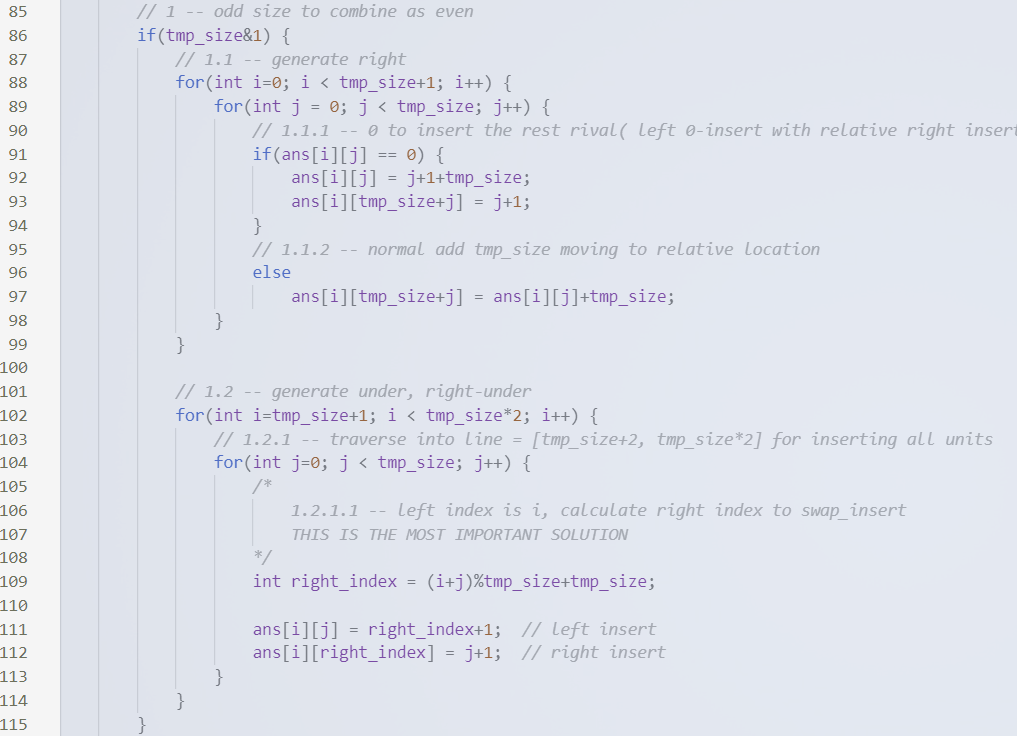
首先强调一些奇数的特征：

1. 有0存在。这代表轮空。

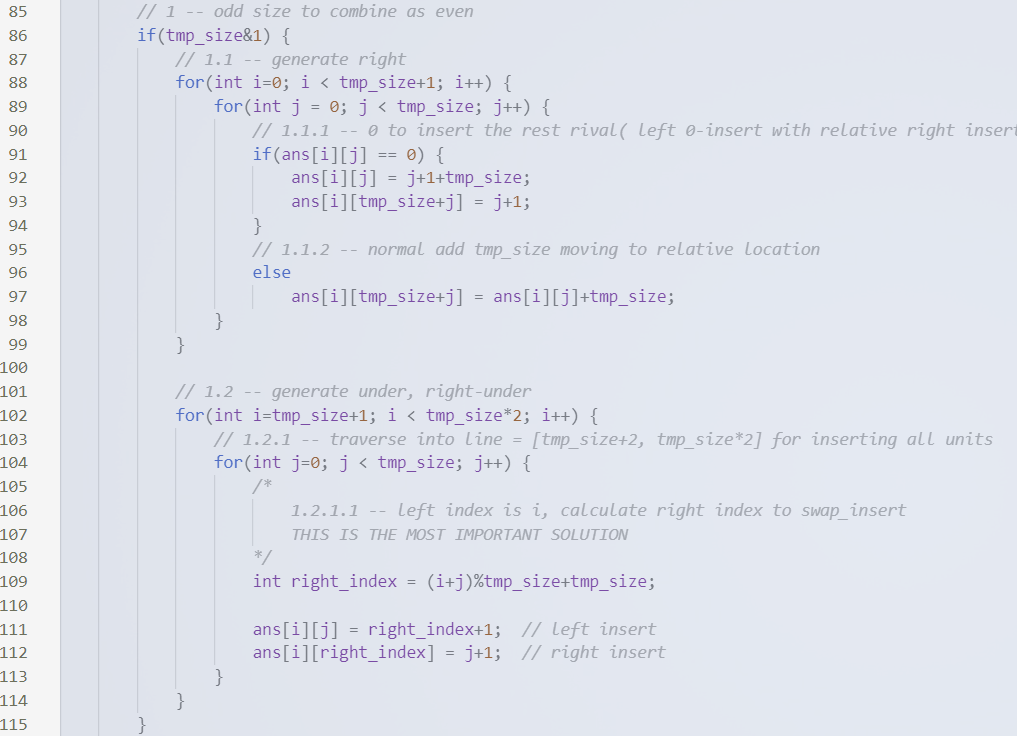
2. 行数是size+1（第一行代表选手列举）

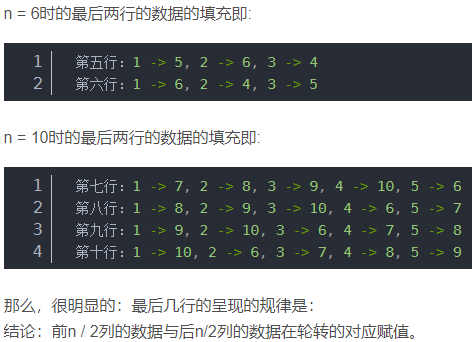
所以有以下注意要点：

* 首先填充右边方格。在扩充的时候，我们知道一行中至少会出现两个0。但是每一天在最优解中是不可能有轮空出现的，所以这两个空格就是当天需要对打的最后一对。



* 填充完右边方格之后，我们还有下边和右下边没有填充，这时候需要填充的行数是size-1行。这时候就要用到一个强推论。



 这个**强推论**就是图中描述的：

所以我们的赋值语句就是查找相应的右坐标，使得**左右坐标互换下标值**。

int right\_index = (i+j)%tmp\_size+tmp\_size;

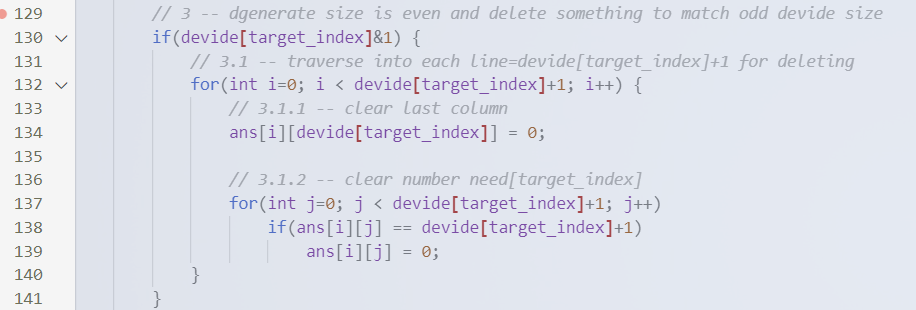
        ans[i][j] = right\_index+1; *// left insert*

        ans[i][right\_index] = j+1; *// right insert*

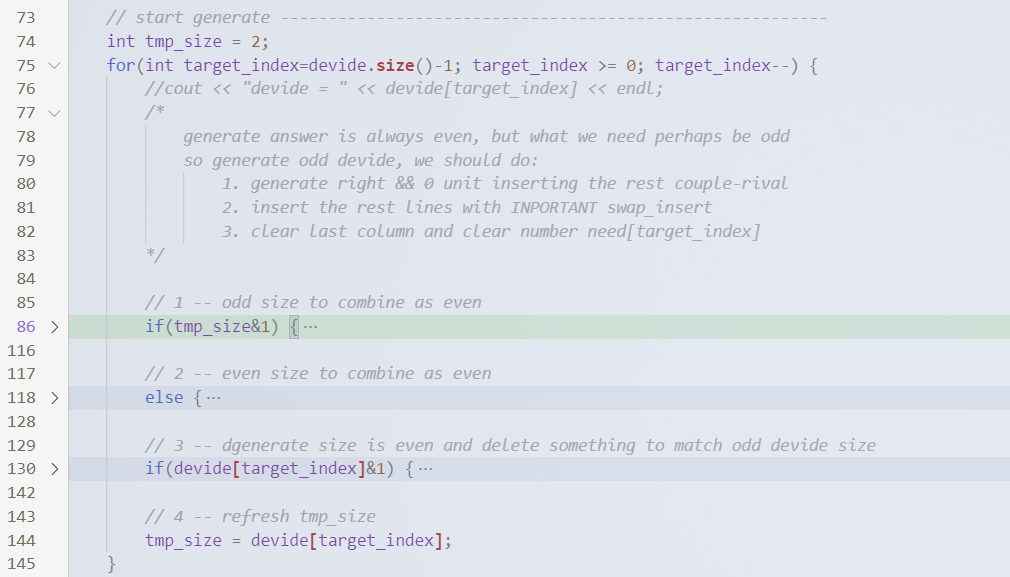
### 2.4.4 生成奇数结果矩阵

我们知道，combine之后生成的都是偶数矩阵。但是我们需要的可能是奇数矩阵。所以我们需要一个方式进行相应的转化。

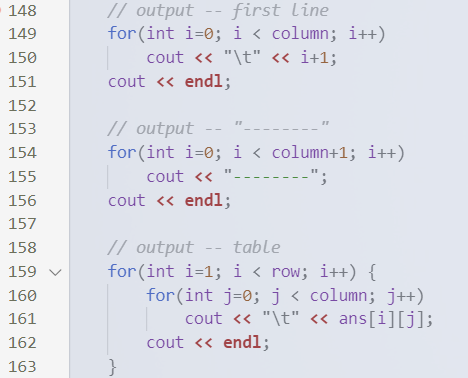
其实这也很简单。把最后一名当成空气人，其他人与他对战设为轮空，赋值为0，然后clear最后一行空气人的所有对战信息，这样就能生成所谓的奇数矩阵了。



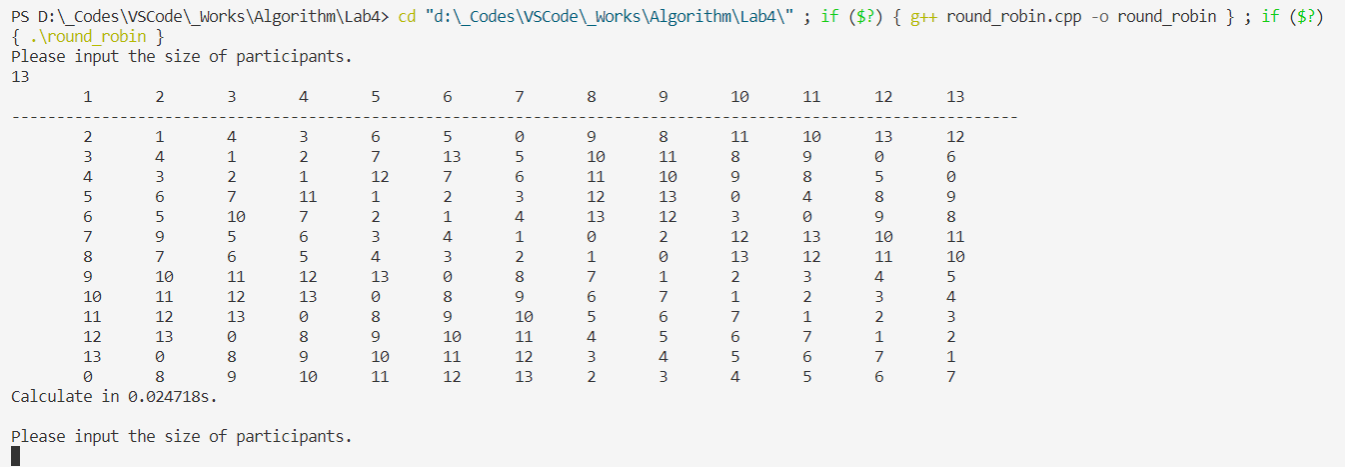
### 2.4.5 combine逻辑



### 2.4.6 输出函数



终端运行大概长这样：



## 2.5 正确性检验

这个测试很简单，主要是进行白盒测试，使得所有逻辑分支都能走正确；以及黑盒测试，使得该程序在大量数据测试下仍能正确运行。

### 2.5.1 白盒测试

主要逻辑分支就是三个：

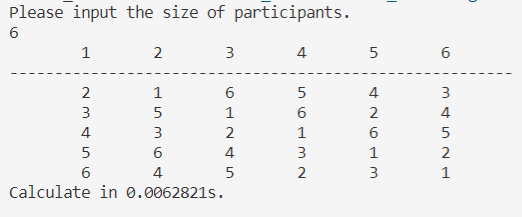
1. 奇矩阵combine成偶矩阵

2. 偶矩阵combine成偶矩阵

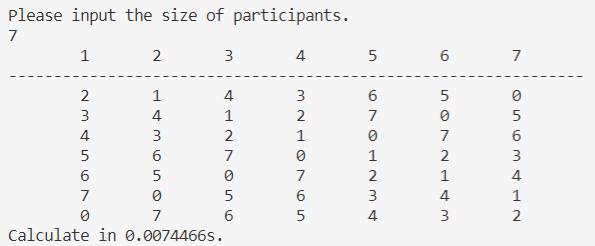
3. 偶数矩阵生成奇数矩阵

所以我们采用测试数据集为：

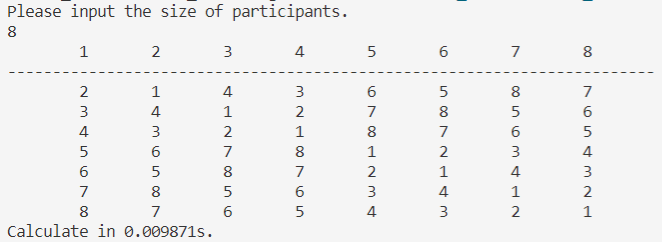
6：3+3，测试第一个逻辑分支



7：4+4-1，测试第二个和第三个逻辑分支



8：4+4，测试第三个逻辑分支

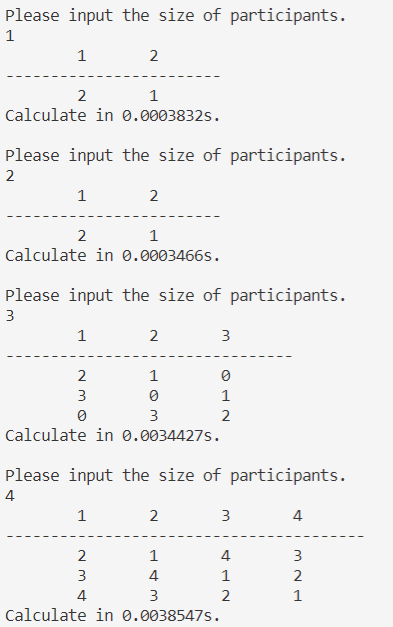
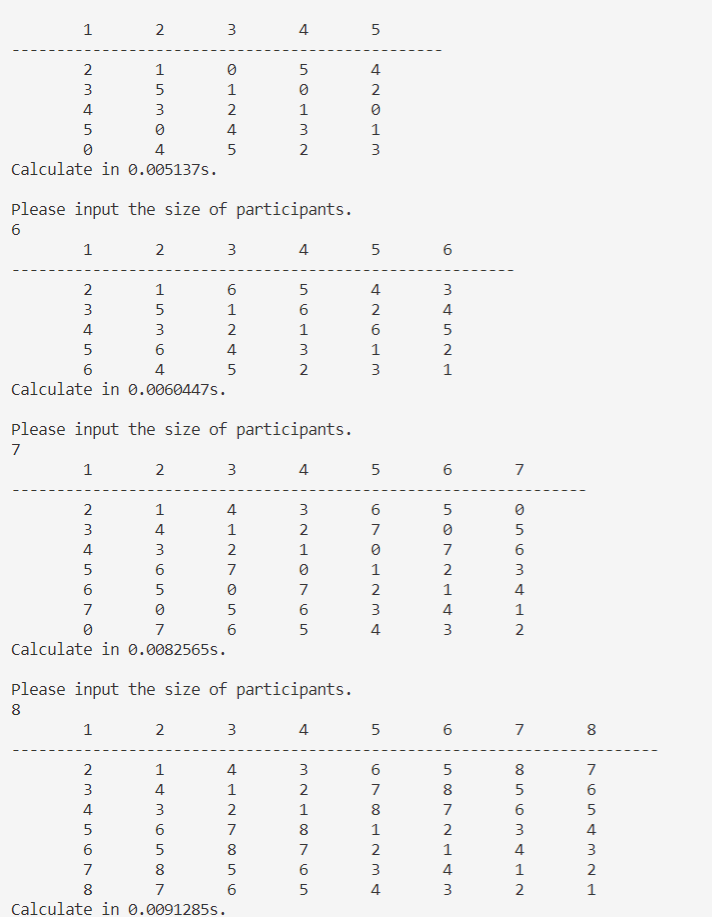


显然，这几个测试都没有问题。

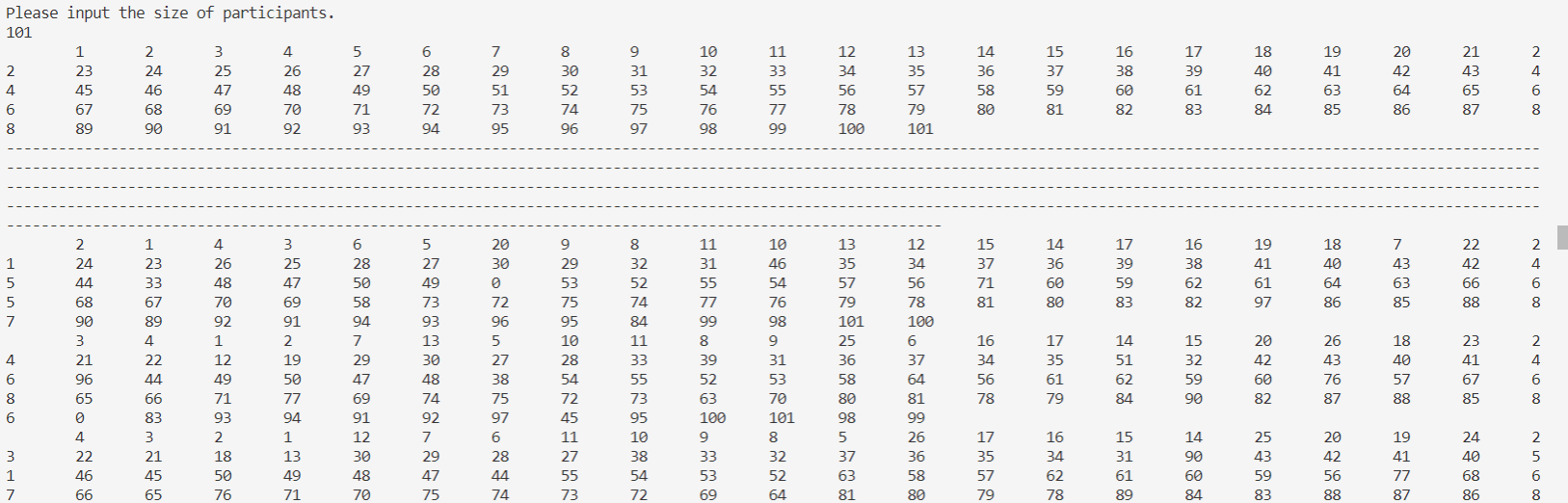
### 2.5.2 黑盒测试

* 测试了1-8的数据集

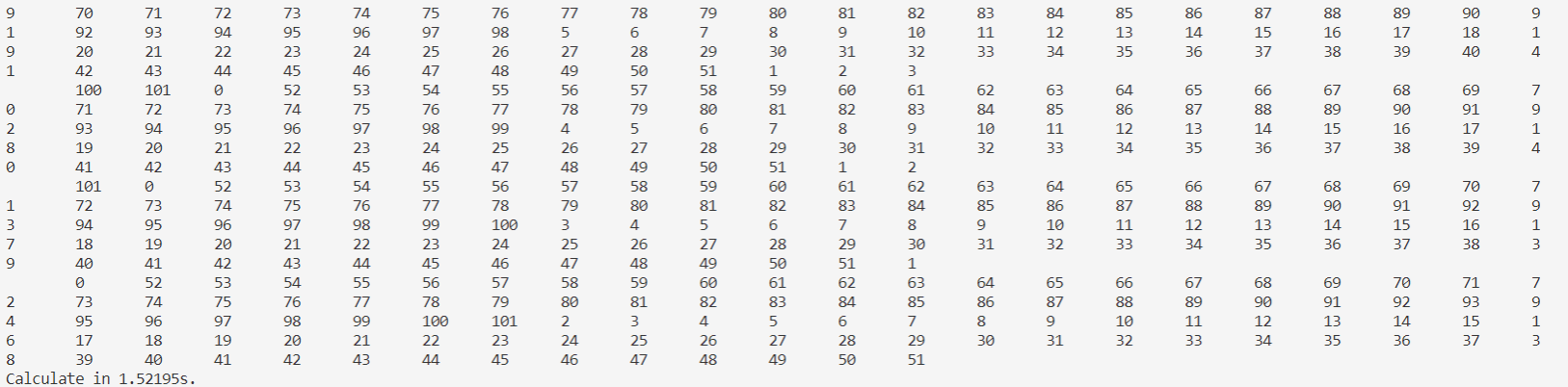
可以说是走完了所有可能的逻辑分支。都发现没有错误。



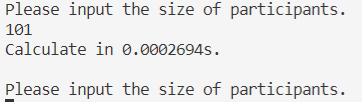
* 测试101



…

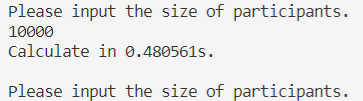


但是明显我们的输出一定占了大多数的运行时间。所以我们去掉输出试一试。



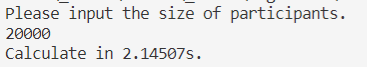
发现计算复杂度很小。

* 测试10000



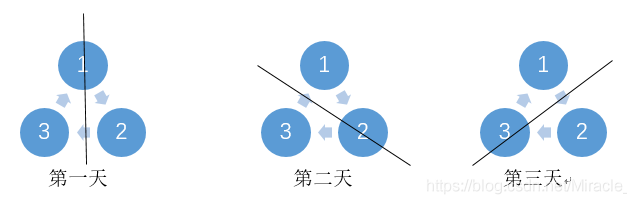
没有IO显得计算时间很小。

* 测试20000



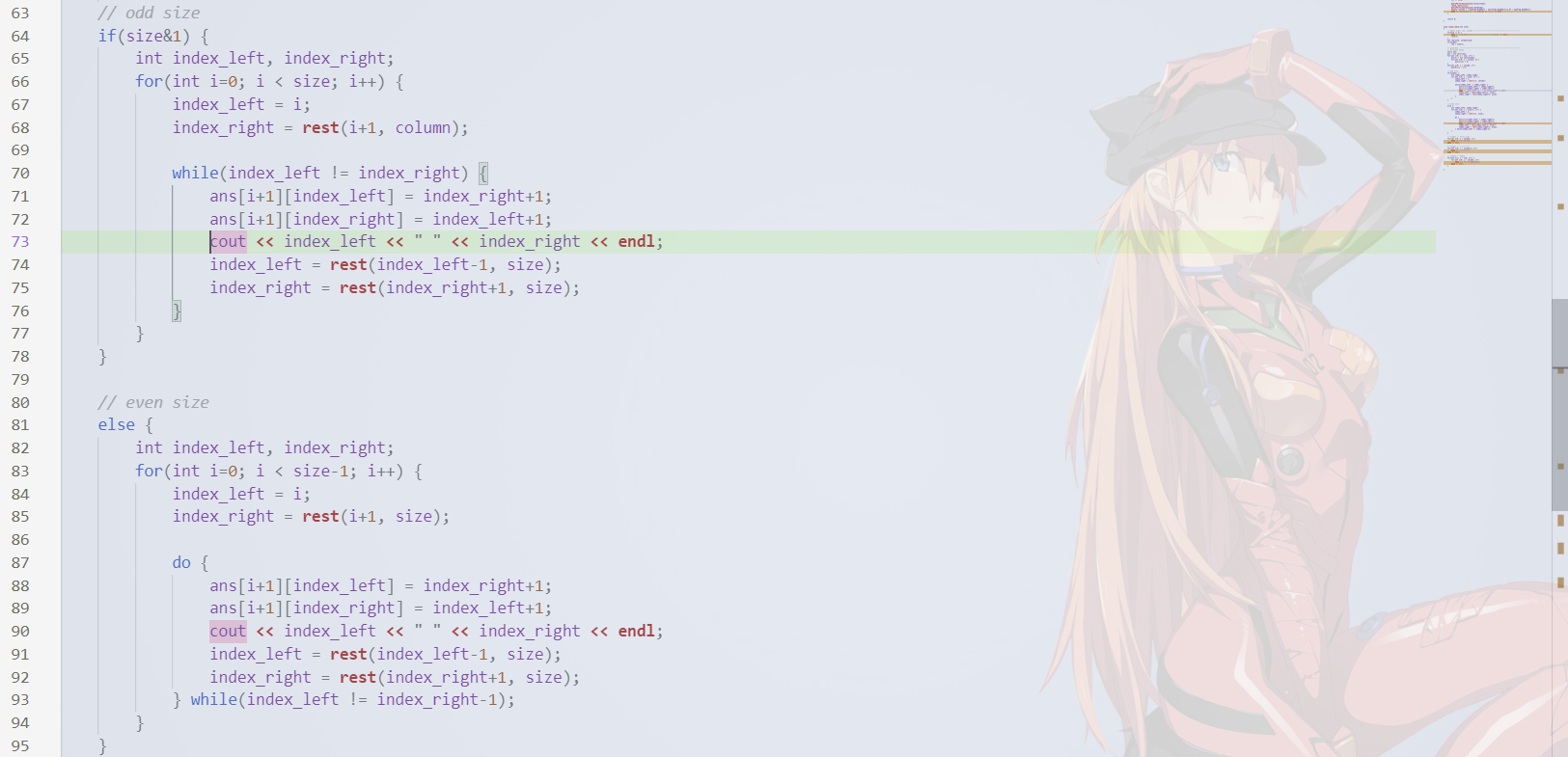
2.6 拓展：多边形算法

其实这是一个很简单的思路

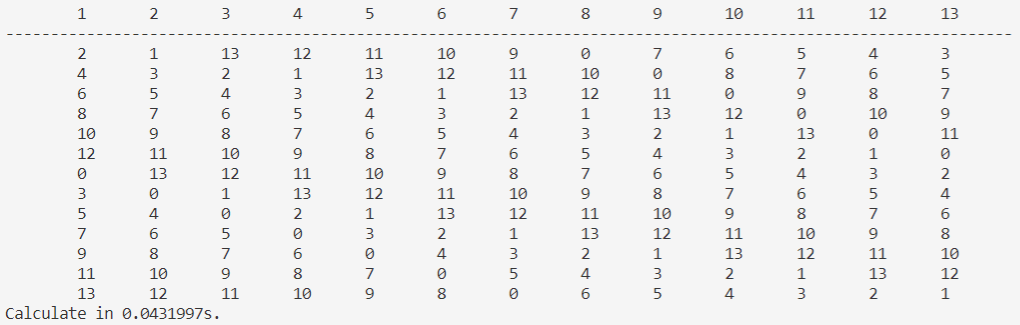


直接从中轴线开始分割，两边对称就是对战安排。

很简单奥，直接根据中轴线循环，左右遍历一下就是一天的结果。（分奇偶）

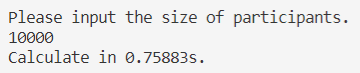


* 测试13



差不多可以。

* 测试10000（屏蔽输出）



**多边形运行时间居然比分治长！**

* 测试20000（屏蔽输出）

****

* 测试30000（屏蔽输出）

****

**多边形能运行，分治不能运行！**

# 三、出现问题及解决

## 3.1 子函数逻辑思路理清

分为三个策略：

1. 奇矩阵combine成偶矩阵

2. 偶矩阵combine成偶矩阵

3. 偶数矩阵生成奇数矩阵

因为对分治的把握和对过程的不理解，导致在编写过程中容易把这三个逻辑写岔（比如写奇矩阵combine成偶矩阵时，我们很容易写成了偶数矩阵生成奇数矩阵）

## 3.2 奇数矩阵combine成偶数矩阵

这个是最难的地方。我们处理这个问题的时候，需要考虑右边，下面，右下的数据生成方式。最后还是在细心地编写下得以解决。

# 四、总结

## 4.1 时间复杂度

* 分治法：

T(n) = T(n/2) + (n/2)^2

所以T(n) = (n/2)^2 + (n/4)^2 … +1

所以T(n) = O(n^2)

* 多边形法：

很简单，直接是一个二重循环逐个生成矩阵元素，所以是O(n^2)

## 4.2 空间复杂度

* 分治法：

很简单就是一个存储矩阵。所有的操作都是通过左上角拓展。

* 多边形法：
* 很简单就是一个存储矩阵。他直接生成整个矩阵。

## 4.3 算法效率

这很简单，因为算法效率他就是固定的。他没有最好最坏，输入什么就是什么。

所以不存在讨论因为数据集不同导致的算法效率问题。

## 4.4 生成序列的规整性

这里我的意思是生成矩阵是否有很好看的规律（一眼看出）。

分治法：我们生成的矩阵并不是有很强的对称性，所以最后也没有很好看。

多边形法：这里就有很强的规律性了。他就如算法描述的一样具有对称发散的性质。每一行都是上一层的循环位移。就有了很好看的矩阵

## 4.5 理解与思考

分治法花了我很长的时间，我认为这是有意义的。因为他让我知道了我们如何去应对一个如此复杂的分治问题。在思考过程中我也一直质疑算法的正确性，甚至在生成数组的时候我发现会出现比输入规模大的数字（这显然不对）。这一切的思考都是有意义和有价值的。

此外，多边形法就显得很简单。这就是算法建立在强数学推论上的好处（能用数学解决的就不要靠机器来一步步走哈哈）