算法设计与分析实验报告

实验题目：（期中）循环赛日程表算法的设计与分析

姓名：陈俊卉

学号：2020212256

日期：2022.10.26

一、实验环境

(列出实验的操作环境，如操作系统，编程语言版本等，更多信息可视各自实际情况填写)
① 操作系统：windows 10
② 编程语言：c++
③ 编程工具：vscode 及其组件

二、实验内容

具体要求请参照实验指导书中各实验的“实验内容及要求”部分。
(示例：1.描述你对实验算法的设计思路;2.给出算法关键部分的说明以及代码实现截图；3.对测试数据、测试程序(没有要求则非必须)进行说明，如测试覆盖程度，最好最坏平均三种情况等等，并给出测试结果截图等信息)

1.算法的设计与实现

(1) 分治法

由于本次任务需要考虑 n ＞ 1 的一切正整数，而不仅仅是为偶数的 n 或者为 2 的幂次的 n 值，所以需要使用一些参数确保在任何情况下分治都能够正确执行。用到的参数有(n 为选手个数)：

参数名	意义	公式
days	比赛天数	n（n 为奇数）；n-1（n 为偶数）
m	处理的选手的中间编号	(int)ceil(n / 2.0)【确保前半部分长于后半部分】
passed_days	前半部分（子问题）的天数	m（m 为奇数）；m-1（m 为偶数）

① 实验代码：

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

// 定义一个最大容纳选手人数，可修改
```

```
const int max_num = 100;
// 定义结果表，行为天数，列为选手
int res[max_num][max_num];




void deal(int n){
    // 设置参数

    // 确保前半部分长于后半部分
    int m = (int)ceil(n / 2.0);

    // 依据题意，n 为偶数时循环赛进行 n-1 天,n 为奇数时为 m 天
    int days;
    if (n % 2 == 0){
        days = n - 1;
    }
    else{
        days = n;
    }

    // 将天数分为两个部分 先算出前半部分再纵向构造:m 为偶数时为 m-1，m 为奇数时为 m
    int passed_days;
    if (m % 2 == 0){
        passed_days = m - 1;
    }
    else{
        passed_days = m;
    }

    // 横向构造
    for (int i = 1; i <= m; i++){
        for (int j = 1; j <= passed_days; j++){
            // 若本来有对手（不是虚拟对手，即上一个递归计算得到的子问题答案不为 0）
            if (res[i][j] != 0){
                res[i + m][j] = res[i][j] + m;
            }

            // 否则让它们互为对手
            else{
                res[i][j] = i + m;
                res[i + m][j] = i;
            }
```

```
        }
    }

    // 注意，如果上述情况出现了虚拟对手的情况（即为奇数），则 res[1][passed_days]的数引入了本不属于这个子问题的选手序号 m+1.
    // 此时使用公式 rvalue = (count + i - 1) % m + m + 1 会导致 res[1][passed_days] 与 res[1][passed_days + 1]一致
    // 所以需要 + 1 修正： rvalue = (count + fix + i - 1) % m + m + 1
    // fix 的值取决于 res[1][passed_days] 是否为 m+1
    int fix = 0;
    if (res[1][passed_days] == m + 1){
        fix = 1;
    }

    // 纵向构造
    for (int i = 1; i <= m; i++){
        int count = 0;
        for (int j = passed_days + 1;j <= days; j++){
            int rvalue = (count + fix + i - 1) % m + m + 1;
            res[i][j] = rvalue;
            // 赋值相对应的选手
            res[rvalue][j] = i;
            count++;
        }
    }

    // 此时已经构造完毕，但如果 n 为奇数，是设置了一个虚拟对手的。我们需要将其消除.
    if (n % 2 != 0){
        for (int i = 1; i <= 2 * m; i++)
        {
            for (int j = 1; j <= days; j++)
                if (res[i][j] == n + 1)
                    res[i][j] = 0;
        }
    }
}




void divide(int n){
    // 如果人数小于等于 1，没有意义，直接返回
    if (n == 1) return;
```

```
    // 如果问题人数为 2,直接返回正确答案
    else if (n == 2){
        res[1][1] = 2;
        res[2][1] = 1;
    }

    // 分治
    else{
        // 确保前半部分长于后半部分，可以成功对应赋值，所以使用 ceil
        divide((int)ceil(n / 2.0));
        // 根据前半部分横向、纵向构造
        deal(n);
    }
}

void get_res(int n){
    for (int i = 0; i <= n; i++){
        if (i == 0) cout << "pnum:" << ' ';
        else cout << i << ' ';
    }
    cout << endl;
    // cout << endl;


    int days;
    if (n % 2 == 0){
        days = n - 1;
    }
    else{
        days = n;
    }

    for (int j = 1; j <= days; j++){
        cout << "day" << j << ": ";
        for (int i = 1; i <= n; i++){
            cout << res[i][j] << ' ';
        }
        cout << endl;
    }
}


int main(){
    int n;
```

```
    cin >> n;
    divide(n);
    get_res(n);
}
```

② 原理：

实际上，我们的分治过程的每个阶段并不是独立的，而是有序的。我们将一个完整的子问题解决阶段分为两个部分：**横向构造**和**纵向构造**。

下文中的 i 表示选手编号，j 表示第 j 个比赛日。

我们以 n=4 为例：

我们设定的**初始状态是** n=2。显然，这个子问题只有一个比赛日。图表如下：

时间\编号	1	2
Day1	2	1

我们先进行横向构造。横向构造的公式为：

$$res[i+m][j] = res[i][j] + m$$

其中

$$1 \le i \le m$$
$$1 \le j \le passed\_days$$

显然，这样横向构造不会出现重复现象。但可以预见的是，$res[i][j]$并不一定都有确定值（可能为 0），因为在子问题中，**若 n 为奇数，则会出现轮空现象。我们需要让在子问题中轮空的对象与公式对应的$[i+m][j]$对应的选手互为对手**（这一点会在下文叙述）现在暂时按下不表。

横向构造后，就可以得出 n=4 时 Day1 的表：

时间\编号	1	2	3	4
Day1	2	1	4	3

接下来进行纵向构造。纵向构造的公式是：

$$res[i][j] = rvalue$$
$$res[rvalue][j] = i$$

其中

$$rvalue = (count + i + fix - 1)\%m + m + 1$$
$$passed\_days + 1 \le j \le days$$
$$1 \le i \le m$$

且 **count 初始值为 0，随着 j 的增加而增加。**

显然在 rvalue 的式子中，count 确保在 passed_days 之后的日子里，选手的对手都不同（列）；而 i 确保每一天每个选手的对手没有重复（行）。

而 fix 为判断是否要修正的参数：如果上述情况出现了虚拟对手的情况（即子问题为奇数），则 res[1][passed_days]的数引入了**本不属于子问题的选手序号** m+1。此时使用公式 rvalue = (count + i - 1) % m + m + 1 会导致 res[1][passed_days] 与 res[1][passed_days + 1]一致。所以需要 $fix = 1$ 修正，否则 $fix = 0$ 。（在 n=6 的例子会提及）

代入公式可以依次得到：

时间\编号	1	2	3	4
Day1	2	1	4	3
Day2	3		1	
Day3	4			1

时间\编号	1	2	3	4
Day1	2	1	4	3
Day2	3	4	1	2
Day3	4	3	2	1

在 n=3 时，我们仍以初始状态开始：

时间\编号	1	2
Day1	2	1

计算得到：

$$m = 2$$
$$passed\_days = 1$$
$$days = 3$$

所以横向构造、纵向构造会出现一个编号为 4 的**虚拟对手**，这使得结果与 n=4 一致：

时间\编号	1	2	3	4
Day1	2	1	4	3
Day2	3	4	1	2
Day3	4	3	2	1

我们需要将第四列去掉，并将前三列出现的虚拟对手置为 0，**表示轮空：**

时间\编号	1	2	3
Day1	2	1	**0**
Day2	3	**0**	1
Day3	**0**	3	2

考虑更加普适的 n=6 的情况：

计算可知，

$$m = 3$$
$$passed\_days = 3$$
$$days = 5$$

我们知道，n=6 的子问题是 n=3.而 n=3 的结果在上文已经算出。则：

时间\编号	1	2	3	4	5	6
Day1	2	1	**0**			
Day2	3	**0**	1			
Day3	**0**	3	2			
Day4						
Day5						

在进行横向构造时，代入公式时会出现虚拟对手。则需要让他们互相为对手。**即若 res[i] [j]=0,则我们令 res [i] [j] = i + m,res [i + m] [j] = i：**

时间\编号	1	2	3	4	5	6
Day1	2	1	**6**	5	4	**3**
Day2	3	**5**	1	6	**2**	4
Day3	**4**	3	2	**1**	6	5
Day4						
Day5						

在进行纵向构造时，因为其子问题为 n=3 为奇数，出现了虚拟对手，所以我们需要使用 fix=1 修正.构造完毕后得到的完整表格为：

时间\编号	1	2	3	4	5	6
Day1	2	1	6	5	4	3
Day2	3	5	1	6	2	4
Day3	4	3	2	1	6	5
Day4	5	6	4	3	1	2
Day5	6	4	5	2	3	1

综上，原问题被拆分为子问题的横向构造和纵向构造，通过 divide-conquer 可以解决。

④ **算法复杂度：**

由算法，我们能够得到以下递推式：

$$T(n) = T(\frac{n}{2}) + O(n^2)$$

实际上$O(n^2)$项为 $m \times passed\_days + m + 2m$ ．我们不妨将其直接视为n^2，这并不影响结论：

$$T(n) = T(\frac{n}{2}) + n^2$$

同理可得

$$T(\frac{n}{2}) = T(\frac{n}{4}) + (\frac{n}{2})^2$$

递推代入得

$$T(n) = T(\frac{n}{2^k}) + n^2 + (\frac{n}{2})^2 + \cdots + (\frac{n}{2^{k-1}})^2$$

令

$$k = \log_2 n$$

根据等比数列公式，以及$T(1) = 0$，得

$$T(n) = \left[\frac{4}{3}\left(1 - \left(\frac{1}{4}\right)^k\right)\right] \times n^2$$

令 n→+∞，$\left(\frac{1}{4}\right)^k$→0.最终有

$$T(n) = O(n^2)$$

空间复杂度：一个二维数组，即为$O(n^2)$.

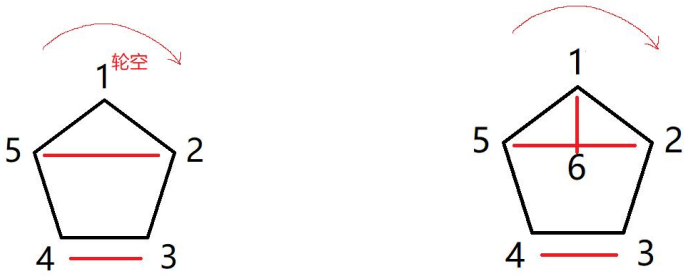
⑤ 正确性：

```
{ .\divide }HaRry_\Desktop\算法分析\实验四>
input competitor num
6
pnum: 1 2 3 4 5 6
day1: 2 1 6 5 4 3
day2: 3 5 1 6 2 4
day3: 4 3 2 1 6 5
day4: 5 6 4 3 1 2
day5: 6 4 5 2 3 1
```

```
PS C:\Users\HaRry_\Desktop\算法分析\实验四>
{ .\divide }
input competitor num
9
pnum: 1 2 3 4 5 6 7 8 9
day1: 2 1 8 5 4 7 6 3 0
day2: 3 5 1 9 2 8 0 6 4
day3: 4 3 2 1 0 9 8 7 6
day4: 5 7 4 3 1 0 2 9 8
day5: 6 4 5 2 3 1 9 0 7
day6: 7 8 9 0 6 5 1 2 3
day7: 8 9 0 6 7 4 5 1 2
day8: 9 0 6 7 8 3 4 5 1
day9: 0 6 7 8 9 2 3 4 5
```

(2) 多边形轮转法

主要思路：构造一个奇数边的多边形，最上方的顶点与中间的选手比赛（人数为奇数时中间没有，则轮空），其余在同一条平行线上的选手各自比赛，随后旋转多边形以达到循环的目的。这样，旋转次数恰好对应比赛天数，且保证每一位选手与其他所有选手都对战过。人数为奇数的情况具体如左图所示。**若选手有偶数个则将编号最大的放到中间，每次最顶上的顶点与中间编号的选手比赛，人数为偶数的情况如右图所示。**



① 实验代码：

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



void cycle(int n){
    // 平行线上的数
    int line_left;
    int line_right;

    // 多边形的边数
    int angle_num;
    // 奇数则不变，中间为虚拟对手（轮空）
    // 偶数则减 1，将编号最大的选手放在多边形中心，与多边形最顶上的点的选手比赛
    if (n % 2 == 1){
        angle_num = n;
    }
    else{
        angle_num = n - 1;
    }
    // 对顶点进行轮转
    for (int i = 1;i <= angle_num; i++){
        cout << "day" << i << ":" << endl;
        // 如果选手个数为奇数，则必然每一天都会有人轮空
```

```
        // 我们选取最顶上的这个顶点轮空
        if (n % 2 == 1){
            cout << "competitor " << i << " have a bye" << endl;
        }
        // 如果选手个数为偶数，则最顶上的顶点和中间得选手比赛
        else{
            cout << i << " vs " << n << endl;
        }

        // 计算平行线左边和右边的选手(第一组)
        line_left = (i-1) % angle_num ? (i-1) % angle_num : angle_num;
        line_right = (i+1) % angle_num ? (i+1) % angle_num : angle_num;
        cout << line_left << " vs " << line_right << endl;

        // 遍历所有平行线
        for (int j = 1; j <= angle_num / 2 - 1; j++){
            line_left = (line_left-1) % angle_num ? (line_left-1) % angle_num : angle_num;
            line_right = (line_right+1) % angle_num ? (line_right+1) % angle_num :
angle_num;
            cout << line_left << " vs " << line_right << endl;
        }

    }


}


int main()
{
    int num;
    cout << "input competitor num" << endl;
    cin >> num;
    cycle(num);
}
```

② 算法复杂度：

观察算法易得，时间复杂度为：

$$\left[k_1 + k_2 \times \left(\frac{n}{2} - 1\right)\right] \times n$$

k_1、k_2 为常数。所以时间复杂度即为$O(n^2)$.

空间复杂度显然为$O(1)$.

③ 正确性：

```
PS C:\Users\HaRry_\Desktop\算法分析\实验四>
.\cycle }
input competitor num
6
day1:
1 vs 6
5 vs 2
4 vs 3
day2:
2 vs 6
1 vs 3
5 vs 4
day3:
3 vs 6
2 vs 4
1 vs 5
day4:
4 vs 6
3 vs 5
2 vs 1
day5:
5 vs 6
4 vs 1
3 vs 2
```

```
.\cycle }rs\HaRry_\Desktop\算法分析\实验四>
input competitor num
7
day1:
competitor 1 have a bye
7 vs 2
6 vs 3
5 vs 4
day2:
competitor 2 have a bye
1 vs 3
7 vs 4
6 vs 5
day3:
competitor 3 have a bye
2 vs 4
1 vs 5
7 vs 6
day4:
competitor 4 have a bye
3 vs 5
2 vs 6
1 vs 7
day5:
competitor 5 have a bye
4 vs 6
3 vs 7
2 vs 1
day6:
competitor 6 have a bye
5 vs 7
4 vs 1
3 vs 2
day7:
competitor 7 have a bye
6 vs 1
5 vs 2
4 vs 3
```

两种算法都没有所谓的最好、最坏情况，硬要说的话，就是分治法的选手数如果为 2 的幂次，则不存在虚拟对手的 mask。最坏情况就是遇到足够多的奇数选手子问题。

三、出现问题及解决

(列出你在实验中遇到了哪些问题以及是如何解决的)

问题一：如何表示分治法中的“治”，也就是将分治数学模型转换为算法模型。

我考虑了几种实现无重复无遗漏分治的数学模型，但它们都会有具体的限制，如规定 n 必须是 2 的幂次，这并不符合题目的要求。后来经过考虑，放弃了直接对称赋值的想法。因为直接对称赋值本身就无法满足 n 为任意大于一的正整数这个要求（直接对称赋值要求二维数组的长和宽必须一一对应）。所以我把目光转移到了偏移赋值，通过横向构造和纵向构造最终实现了分治的方法。

问题二：是否能够运用该问题的几何特性来解决问题。

这个问题其实本来是在分治法时思考的，但考虑到分治法本身的二分特性，找不到与之几何特性类似的图形；而随后思考出的多边形轮转法正好解决了我这个问题。

四、总结

(对所实现算法的总结评价，如时间复杂度，空间复杂度，是否有能够进一步提升的空间，不同实现之间的比较，不同情况下的效率，通过实验对此算法的认识与理解等等)

通过本次实验，我尝试了先构建二分数学模型，随后建立分治算法模型解决循环赛日程表算法，也尝试了构建多边形轮转模型解决该问题。我意识到构建算法模型的第一步是构建问题对应的正确数学模型，思考该数学模型的正确表示方法，从而才能根据表示方法正确地构造出相应的算法模型。

同时，我意识到一些常规的模型可以套用以更好的时间空间复杂度解决这种类似的问题，如回溯法、分治法等。但更加惊艳的往往是使用几何性质解决问题的数学模型，如多边形轮转模型。