循环赛问题的四种解法

一、实验题目

设有n个运动员要进行网球循环赛，设计一个满足以下要求的比赛日程表：

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，循环赛进行n-1天。
4. 当n是奇数时，循环赛进行n天。

二、实验分析与设计

2.0 数据结构说明

用Table类存储赛程安排

`Table.date[i][j]`表示第i天，j运动员的对手。

`Table.has_fought[i][j]`表示i运动员与j运动员是否比赛过。

`vector <Fight> record[day]` 表示第day天的所有对战记录，其中，Fight结构体存储对战双方的编号：

```
struct Fight
{
    int x, y;
    Fight() : x(0), y(0) {};
    Fight(int _x, int _y) : x(_x), y(_y) {};
};
```

2.1 暴力深搜法

首先我们使用最朴素的暴力搜索算法。由于N为奇数和偶数时比赛总天数不同（N为奇数时循环赛进行**N**天，N为奇数时循环赛进行**N-1**天），且安排轮空的方式不同（N为奇数时**每天有一个运动员要轮空**，N为偶数时不需要安排轮空），于是用`n_is_odd`和`n_is_even`记录n的奇偶性，分情况进行遍历搜索。

- **N为奇数时**，从第1天搜索到第**N**天安排比赛，其中第一天被轮空的运动员为**N**，一直到第N天安排时被轮空的运动员为1
- **N为偶数时**，从第1天搜索到第**N - n_is_even**天安排比赛，不需要安排轮空

```
void DFS::Run()
{
    long long BegieTime = GetSysTimeMicros();
    n_is_odd = N & 1;
    n_is_even = 1 - n_is_odd;
    dfs_day(1, n_is_odd ? N : 0);        //从第一天开始搜索，如果是奇数，初始轮空运动员设为N，如果是偶数，轮空运动员设为0
    RunTime = (GetSysTimeMicros() - BegieTime) / 1000.0;
}
```

```
bool DFS::dfs_day(int day, int rest)    //day表示当前天数，rest表示当前轮空的运动员
{
    if (day > N - n_is_even)            //搜索到超过N - n_is_even天时说明完成搜索
        return true;
    return dfs_arrange(day, 1, rest);   //进入第day天搜索，第一个安排的运动员为1
}
```

在进行每一天的搜索时，遍历每一个运动员，考虑以下几种情况：

- 如果当前player在第day天被轮空，或在之前已经被安排好比赛，则直接考虑下一人
- 如果当前player没有轮空，且未被安排比赛，遍历编号大于他的运动员，若两人尚未对战过，则调用`fight(day, x, y)`记录运动员x与运动员y在第day天对战，进入递归搜索。若递归结果为Flase（最终没有找到可行解），则调用`cancel_fight(day, x, y)`清除对战记录，继续遍历。

```
bool DFS::dfs_arrange(int day, int player, int rest)
{
    if (player > N) {
        return dfs_day(day + 1, rest - 1);
    }
    if (player == rest || table.date[day][player])  //如果当前player
在第day天被轮空，或在之前已经被安排好比赛，则直接考虑下一人
        return dfs_arrange(day, player + 1, rest);
    FOR(i, player + 1, N)
        if (!table.date[day][i] && player != rest &&
!table.has_fought[player][i])
        {
            fight(day, player, i);          //安排player与i比赛
            if (dfs_arrange(day, player + 1, rest))     //如果在当前
的安排下深搜结果为TRUE，则不必继续深搜
                return true;
//直接返回可行结果
            cancel_fight(day, player, i);   //撤销player与i的比赛记录
        }
    return false;   //当前运动员遍历完其他所有人都没有找到合法对手，说明当前
安排不可行
}
```

```
inline void DFS::fight(int day, int x, int y)
{
    table.date[day][x] = y;
    table.date[day][y] = x;
    table.has_fought[x][y] = 1;
    table.has_fought[y][x] = 1;
}

inline void DFS::cancel_fight(int day, int x, int y)
{
    table.date[day][x] = 0;
    table.date[day][y] = 0;
    table.has_fought[x][y] = 0;
    table.has_fought[y][x] = 0;
}
```

N=4时输出结果

```
2 1 4 3
3 4 1 2
4 3 2 1
```

N=6时输出结果

```
2 1 4 3 6 5
3 5 1 6 2 4
4 6 5 1 3 2
5 4 6 2 1 3
6 3 2 5 4 1
```

N=15输出结果

```
2 1 4 3 6 5 8 7 10 9 12 11 14 13 0
3 4 1 2 7 8 5 6 11 12 9 10 15 0 13
4 3 2 1 8 7 6 5 12 11 10 9 0 15 14
5 6 7 8 1 2 3 4 13 14 15 0 9 10 11
6 5 8 7 2 1 4 3 14 13 0 15 10 9 12
7 8 5 6 3 4 1 2 15 0 13 14 11 12 9
8 7 6 5 4 3 2 1 0 15 14 13 12 11 10
9 10 11 12 13 14 15 0 1 2 3 4 5 6 7
10 9 12 11 14 13 0 15 2 1 4 3 6 5 8
11 12 9 10 15 0 13 14 3 4 1 2 7 8 5
12 11 10 9 0 15 14 13 4 3 2 1 8 7 6
13 14 15 0 9 10 11 12 5 6 7 8 1 2 3
14 13 0 15 10 9 12 11 6 5 8 7 2 1 4
15 0 13 14 11 12 9 10 7 8 5 6 3 4 1
0 15 14 13 12 11 10 9 8 7 6 5 4 3 2
```

由于其与$O(N!N!)$的复杂度，最多只能处理到N=27，需要考虑更优的算法

2.2 分治法

首先观察2.1 暴力深搜的到结果中$N = 2^k$的情况：

```
N=2
(1 2)
 2 1

N=4
(1 2 3 4)
 2 1 4 3
```

```
 3 4 1 2
 4 3 2 1

N=8
(1 2 3 4 5 6 7 8)
 2 1 4 3 6 5 8 7
 3 4 1 2 7 8 5 6
 4 3 2 1 8 7 6 5
 5 6 7 8 1 2 3 4
 6 5 8 7 2 1 4 3
 7 8 5 6 3 4 1 2
 8 7 6 5 4 3 2 1
```

在每一个答案中填补上`1 2 3 .. 2^k`的首行后，可以发现所有的**N=2^k**矩阵都可以由$N=2^{k-1}$的更小矩阵拷贝而来。

从左上角到右上角是内部比赛，可以完美复刻sub_n规模的比赛安排，因此拷贝矩阵，并加上子矩阵大小**sub_n**：

```
table.date[i][j] = table.date[i][j - sub_n] + sub_n;
```

1	2	3	4	5	6	7	8
2	1	4	3	6	5	8	7
3	4	1	2	7	8	5	6
4	3	2	1	8	7	6	5
5	6	7	8	1	2	3	4
6	5	8	7	2	1	4	3
7	8	5	6	3	4	1	2
8	7	6	5	4	3	2	1

从左上角到左下角：同样可以复刻sub_n规模的比赛安排，只不过此时是左子组与右子组的组间竞争，i与j+sub_n两位选手对战

```
table.date[i][j] = table.date[i- sub_n][j] + sub_n;
```

既然确定了左子组在右子组中的对战对手，那么对于右子组，在左子组中的对战对手也就是十分显然的了，在填充左下角的矩阵时，同时对同行填充值所在列赋当前列的值即可（即在自己对手的位置的比赛安排上填上自己）

```
table.date[i][j] = table.date[i- sub_n][j] + sub_n;
table.date[i][table.date[i- sub_n][j] + sub_n] = j;
```

由此，我们可以知道任意$N=2^k$时的比赛安排都可以被直接构造出来。

在N为非2^k的偶数时，由于该N可能曾经由奇数大小的子数组合并而来，存在不确定性，因此将合并算法稍作调整：

扩展内部竞争时，从左上角到右上角的算法不变，直接复制：

```
FOR(i, 1, sub_n - sub_n_is_even)                //若sub_n为偶数，子矩阵的比赛天数为sub_n-1
        FOR(j, sub_n + 1, n)
            table.date[i][j] = table.date[i][j - sub_n] + sub_n;
```

扩展组间竞争，从左上角到左下角时，按顺序与

$$i+sub_n, i+sub_n+1, \ldots, sub_n+sub_n, sub_n+1, sub_n+2, \ldots, sub_n-1$$

即从i+sub_n开始，到达右子组的结尾后返回右子组起点，这些运动员进行比赛。这样遍历循环的目的是使1~sub_n的每一个运动员安排组间比赛时彼此错开对手，不会冲突。此外，与之前$N=2^k$时填充右下角的方式相同，在安排1~sub_n时同时在自己对手的比赛安排上填上自己，即可完成右下角的填充。

从N=6扩展到N=12的例子如下：

1	2	3	4	5	6	7	8	9	10	11	12
2	1	6	5	4	3	8	7	12	11	10	9
3	5	1	6	2	4	9	11	7	12	8	10
4	3	2	1	6	5	10	9	8	7	12	11
5	6	4	3	1	2	11	12	10	9	7	8
6	4	5	2	3	1	12	10	11	8	9	7
7	8	9	10	11	12	1	2	3	4	5	6
8	9	10	11	12	7	6	1	2	3	4	5
9	10	11	12	7	8	5	6	1	2	3	4
10	11	12	7	8	9	4	5	6	1	2	3
11	12	7	8	9	10	3	4	5	6	1	2
12	7	8	9	10	11	2	3	4	5	6	1

在$N \neq 2^k$时，考虑N的奇偶进行分治：

- 若N为偶数
    - 若sub_n为偶数，仿照$N = 2^k$合并两个sub_n的赛事安排
    - 若sub_n为奇数，每一天每个子矩阵中都会各出现一个轮空的运动员，需要**对两个轮空运动员进行合并**（下文讨论）
- 若N为奇数

  此时需要添加一个哨兵，其编号N+1。添加哨兵后，总队伍数为偶数，情况转为N为偶数下的分治。在完成N+1个运动员的安排后，需要删去哨兵，具体操作为删去安排矩阵的第N+1列（对哨兵的安排显然不需要），而后在安排矩阵中所有出现N+1的位置替换为0（若在某天安排某位运动员与哨兵对战，则将其改为轮空），由此可以完成N为奇数下的分治。

  ```
  if (n_is_odd)
  {
      FOR(i, 1, n - n_is_even)
          FOR(j, 1, n)
          if (table.date[i][j] == n + n_is_odd)
              table.date[i][j] = 0;
  }
  ```

那么，回答刚才讨论的问题，若sub_n为奇数时，如何**合并两个轮空的运动员**？我们知道，在安排奇数个运动员的比赛时需要安排一个哨兵，即，那合并两个奇数个运动员的比赛时，岂不是会出现两个哨兵，这该如何解决呢？

既然每一天都有两个运动员轮空，那不如**让两个轮空的运动员对战！**

于是扩展内部比赛时，若发现当前运动员被轮空，则安排左右子矩阵中同一位置的运动员比赛：

```
FOR(i, 1, sub_n - sub_n_is_even)                //若sub_n为偶数，子矩阵的比赛天数为sub_n-1
        FOR(j, sub_n + 1, n)
        if (table.date[i][j - sub_n] == 0)      //发现当前运动员被轮空
        {
            table.date[i][j - sub_n] = j;
            table.date[i][j] = j - sub_n;
        }
        else
            table.date[i][j] = table.date[i][j - sub_n] + sub_n;    //没有轮空，正常复制
```

而在扩展子组间比赛时，本来1~sub_n的每一位运动员都需要与sub_n+1~n的所有另一个子组的运动员比赛，但由于每一位运动员 i 都在自己被轮空的那一天安排了与i+sub_n的比赛，因此在扩展子组间比赛不需要再与他比赛。运动员i即在扩展子组间比赛时，需要按顺序与

$$i + sub_n + 1, i + sub_n + 2, \ldots, sub_n + sub_n, sub_n + 1, sub_n + 2, \ldots, sub_n - 1$$

（比N为偶数时少了与i+sub_n的比赛）这些运动员进行比赛。

```
FOR(j, 1, sub_n)
    {
        int fight = j + sub_n;          //运动员j在组间竞争中的第一个对
手

        if (sub_n_is_odd)               //如果子组是奇数，安排过轮空互相
对战，则第一个对手要后移一位
            fight = (fight == n + n_is_odd) ? sub_n + 1 : fight +
1;    //实现循环遍历

        FOR(i, sub_n + sub_n_is_odd, n - n_is_even)
        {
            table.date[i][j] = fight;
            if (fight <= n) table.date[i][fight] = j;   //在自己对手
的位置的比赛安排上填上自己（如果这个对手是哨兵就不用填了）

            fight = (fight == n + n_is_odd) ? sub_n + 1 : fight +
1;    //实现循环遍历
        }
    }
```

最后，考虑临界情况。在N=2时只需要比赛一天，是最小规模的赛事，直接返回结果：

```
if (n == 2)
    {
        table.date[1][1] = 2;
        table.date[1][2] = 1;
        return;
    }
```

完整分治过程如下：

```
void Divide::work(int n)
{
    if (n == 2)
    {
        table.date[1][1] = 2;
        table.date[1][2] = 1;
```

```
        return;
    }
    int n_is_odd = n & 1;
    int n_is_even = 1 - n_is_odd;

    int sub_n = (n + n_is_odd) / 2;
    int sub_n_is_odd = sub_n & 1;
    int sub_n_is_even = 1 - sub_n_is_odd;

    work(sub_n);
    FOR(i, 1, sub_n - sub_n_is_even)
        FOR(j, sub_n + 1, n)
        if (table.date[i][j - sub_n] == 0)
        {
            table.date[i][j - sub_n] = j;
            table.date[i][j] = j - sub_n;
        }
        else
            table.date[i][j] = table.date[i][j - sub_n] + sub_n;

    FOR(j, 1, sub_n)
    {
        int fight = j + sub_n;
        if (sub_n_is_odd)
        {
            fight++;
            if (fight > n + n_is_odd)
                fight = sub_n + 1;
        }

        FOR(i, sub_n + sub_n_is_odd, n - n_is_even)
        {

            table.date[i][j] = fight;
            if (fight <= n) table.date[i][fight] = j;

            fight++;
            if (fight > n + n_is_odd)
                fight = sub_n + 1;
        }
    }

    if (n_is_odd)
    {
        FOR(i, 1, n - n_is_even)
            FOR(j, 1, n)
            if (table.date[i][j] == n + n_is_odd)
                table.date[i][j] = 0;
    }
}
```

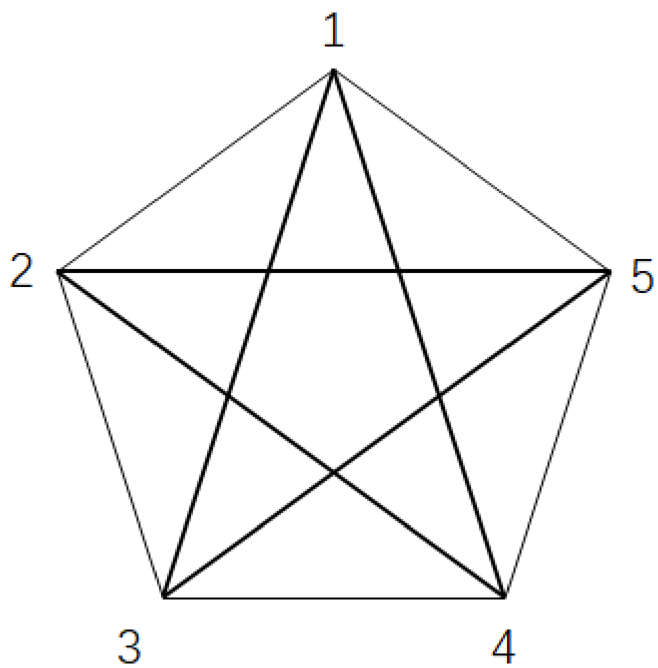
N=9测试结果

```
2 1 8 5 4 7 6 3 0
3 5 1 9 2 8 0 6 4
4 3 2 1 0 9 8 7 6
5 7 4 3 1 0 2 9 8
6 4 5 2 3 1 9 0 7
7 8 9 0 6 5 1 2 3
8 9 0 6 7 4 5 1 2
9 0 6 7 8 3 4 5 1
0 6 7 8 9 2 3 4 5
```

2.3 多边形旋转法

如果我们把问题换成：**N个运动员之间两两比赛，最多需要多少场**，并且用上小学生的思维，你会怎么做？

我会选择**画图求解**。例如，N=5时画出一个五边形，并连接所有顶点之间的线段，数数一共有多少条线段，便是答案：



答案为10场。

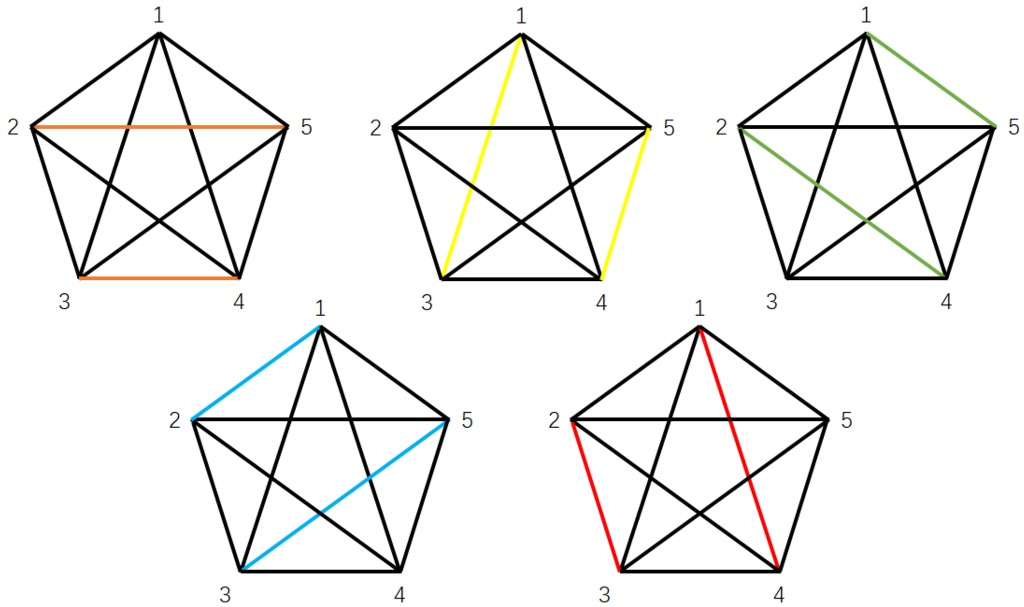
现在对着这个五边形思考我们的问题：既然每一条线段都是一场比赛，那么如何安排每一天的比赛？

答案是选择**尽可能多的顶点不重合线段**。

事实上，安排每一天的比赛时，需要满足的要求是什么？一是尽量让每个运动员都参加比赛，最多只能有一人轮空，而是一个运动员最多只能比一场。那么放到多边形里，就是每天选择的线段中，每个顶点至多出现一次，也就是**顶点不重合**。

那么如何选择尽可能多的满足要求的线段呢？

答案是**选择所有在同一水平线上的线段**。如下图所示：



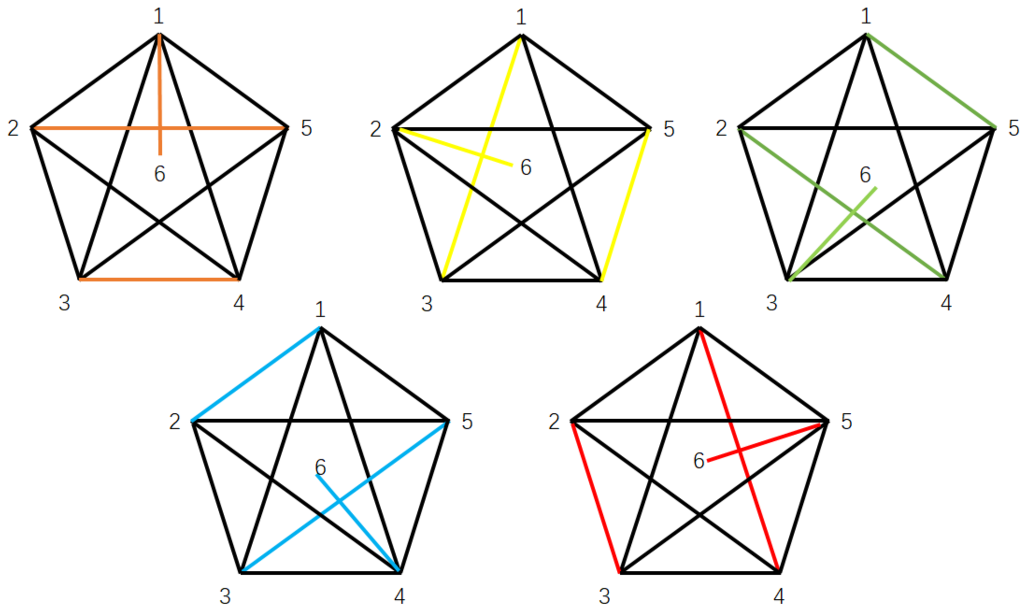
以上5天的安排分别为：

```
(4 , 3) (5 , 2) (1 , 0)
(5 , 4) (1 , 3) (2 , 0)
(1 , 5) (2 , 4) (3 , 0)
(2 , 1) (3 , 5) (4 , 0)
(3 , 2) (4 , 1) (5 , 0)
```

（和零号选手对战代表轮空）

由于N（N为奇数）边形最多只有$\lceil N/2 \rceil$条平行线段，且每次旋转360/N后线段不会重复。恰好满足N个运动员每天需要比$\lceil N/2 \rceil$场的需求，因此用多边形旋转来安排比赛是可行的。

然而，以上性质仅对N为奇数时成立，N为偶数时不成立。不过，在N为偶数时，我们可以构建N-1边形，每次将轮空的运动员与运动员N进行比赛，如下图所示：



此时5天的比赛安排为：

```
(4 , 3) (5 , 2) (1 , 6)
(5 , 4) (1 , 3) (2 , 6)
(1 , 5) (2 , 4) (3 , 6)
(2 , 1) (3 , 5) (4 , 6)
(3 , 2) (4 , 1) (5 , 6)
```

这样，我们就既能满足线段顶点不重合的要求，又能满足N个运动员N-1天结束比赛的要求了。

代码如下：

为维护一个vector `vertex`，每次取所有关于中心对称的点对安排比赛，若N为奇数，则中心点轮空；若N为偶数，则中心点与运动员N比赛。

每一天结束后，将`vertex`结尾的点挪到开头，实现多边形的旋转360/N度。

```
FOR(day, 1, N - n_is_even)
    {
        FOR(i, 0, (N - 1) / 2 - 1)      //取所有关于中心对称的点对安排比赛
        {
            int x = vertex[i];
            int y = vertex[N - n_is_even - 1 - i];
            table.date[day][x] = y;
            table.date[day][y] = x;
        }
        int mid = vertex[(N - 1) / 2];
        if (n_is_even)                  //若N为偶数，则中心点与运动员N比赛
        {
            table.date[day][mid] = N;
            table.date[day][N] = mid;
        }
```

```
        else                            //若N为奇数，则中心点轮空
        {
            table.date[day][mid] = 0;
        }
        int tail = vertex[N - n_is_even - 1];        //将vertex结尾的点挪到开头，实现多边形的旋转360/N度
        vertex.pop_back();
        vertex.insert(vertex.begin(), tail);
        cout << endl;
    }
```

2.4 直接构造法

我们观察 2.3 多边形旋转法 中的输出结果：

N=5

```
(1 , 5) (2 , 4) (3 , 0)
(5 , 4) (1 , 3) (2 , 0)
(4 , 3) (5 , 2) (1 , 0)
(3 , 2) (4 , 1) (5 , 0)
(2 , 1) (3 , 5) (4 , 0)
```

N=6

```
(1 , 5) (2 , 4) (3 , 6)
(5 , 4) (1 , 3) (2 , 6)
(4 , 3) (5 , 2) (1 , 6)
(3 , 2) (4 , 1) (5 , 6)
(2 , 1) (3 , 5) (4 , 6)
```

我们发现，除了N为奇数时轮空的0，或N为偶数时的N，其他每一行的编号都是上一行对应位置的循环减一。

如：从第一天到第二天，$1 \to 5, 5 \to 4, 2 \to 1, 4 \to 3, 3 \to 2$

于是，只要通过中心对称安排出第一天的比赛后，此后每一天将前一天的比赛记录中比赛双方各循环减一，即可得到当天安排。

代码如下（编写代码时使用每天加一，原理相同）：

第一天构造：

```
Struct::Struct(int _N) : table(_N), N(_N) {
```

```
    temp = new int[N + 2];
    FOR(i, 1, N + 1) temp[i] = 0;
    n_is_even = (N & 1) ? 0 : 1;
    int x = 1, y = N, i = 0;
    while (x <= y)
    {
        if (x == y)
        {
            temp[++i] = x;
            temp[++i] = 0;
            table.date[1][x] = 0;
        }
        else
        {
            temp[++i] = x;
            temp[++i] = y;
            table.date[1][x] = y;
            table.date[1][y] = x;
        }
        x++ ;
        y-- ;
    }
};
```

此后每一天构造：

```
FOR(j, 2, N - n_is_even)
{
    for (int i = 1; i <= N; i+= 2)
    {
        int x = temp[i];                        //取出前一天每场比赛的
x
        int y = temp[i + 1];                    //取出前一天每场比赛的
y
        x = (x == N - n_is_even) ? 1 : (x + 1); //x循环加1
        if (y != 0 && !(n_is_even && y == N))   //若y是轮空的0或是N为
偶数时的N，则不能改变
            y = (y == N - n_is_even) ? 1 : (y + 1);
        temp[i] = x;
        temp[i + 1] = y;
        table.date[j][x] = y;
        table.date[j][y] = x;
    }
}
```

三、正确性分析

正确性检验中，需要检测两点：

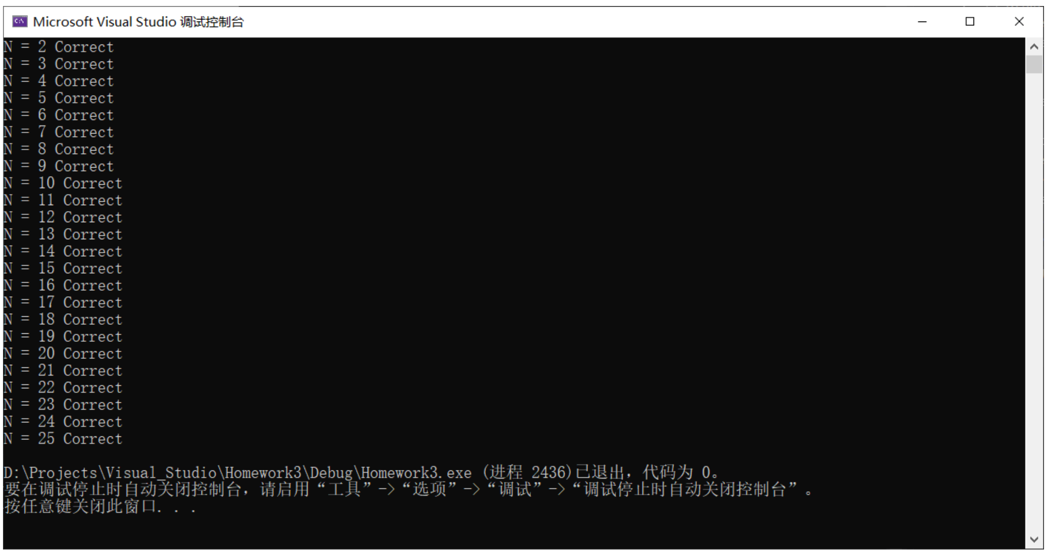
1. 比赛安排矩阵中，在x与y进行比赛时（$date[day][x] = y$），y是否也安排与x比赛（$date[day][y] = x$）
2. 是否在N为偶数的N-1天内，N为奇数的N天内完成所有运动员两两之间的比赛

校验代码如下：

```
FOR(i, 1, N)
    FOR(j, 1, N)
    {
    has_fought[i][j] =  (i == j);
    }

FOR(day, 1, N - n_is_even)
    FOR(x, 1, N)
    if (date[day][x] != 0)      //没有被轮空
    {
    int y = date[day][x];
    if (has_fought[x][y] || date[day][y] != x)
        return false;
    has_fought[x][y] = 1;
    }
FOR(i, 1, N)
    FOR(j, 1, N)
    if (has_fought[i][j] == 0)
        return false;
return true;
```

暴力搜索法测试结果：

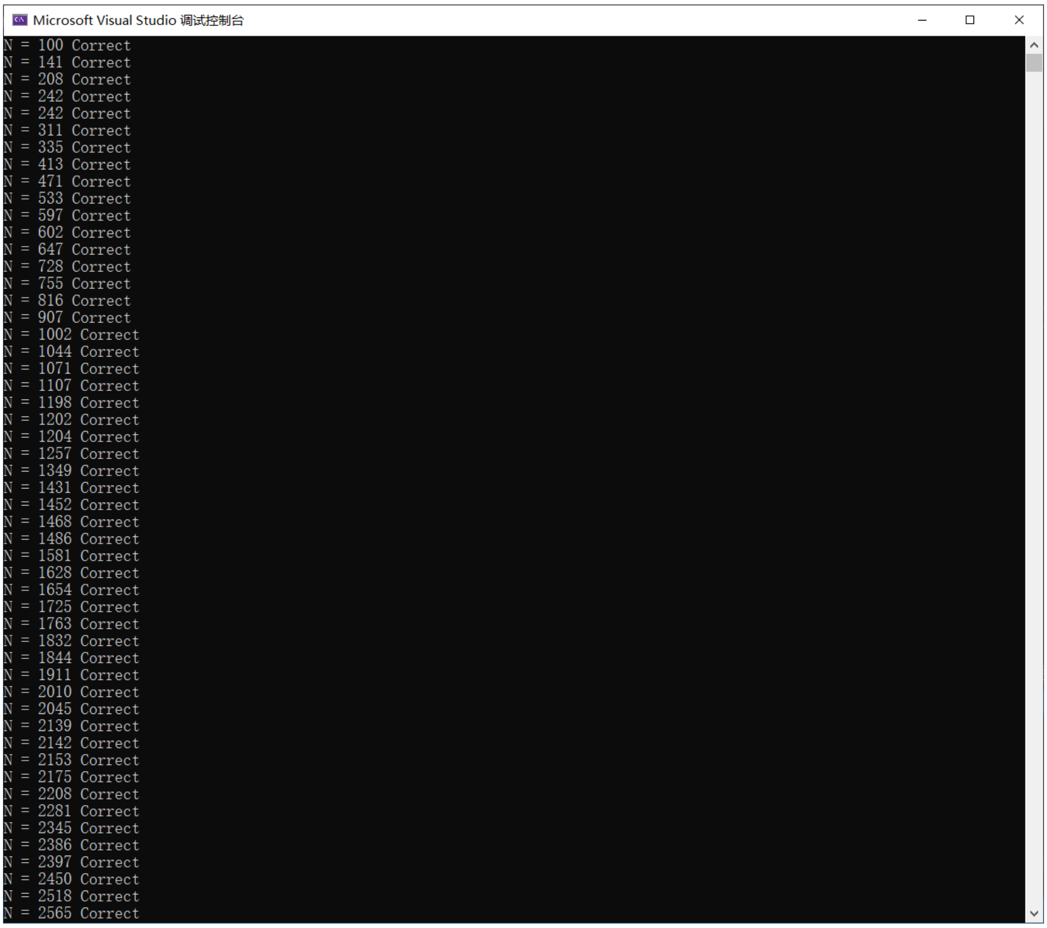


```
Microsoft Visual Studio 调试控制台
N = 2 Correct
N = 3 Correct
N = 4 Correct
N = 5 Correct
N = 6 Correct
N = 7 Correct
N = 8 Correct
N = 9 Correct
N = 10 Correct
N = 11 Correct
N = 12 Correct
N = 13 Correct
N = 14 Correct
N = 15 Correct
N = 16 Correct
N = 17 Correct
N = 18 Correct
N = 19 Correct
N = 20 Correct
N = 21 Correct
N = 22 Correct
N = 23 Correct
N = 24 Correct
N = 25 Correct

D:\Projects\Visual_Studio\Homework3\Debug\Homework3.exe (进程 2436)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

分治法测试结果：

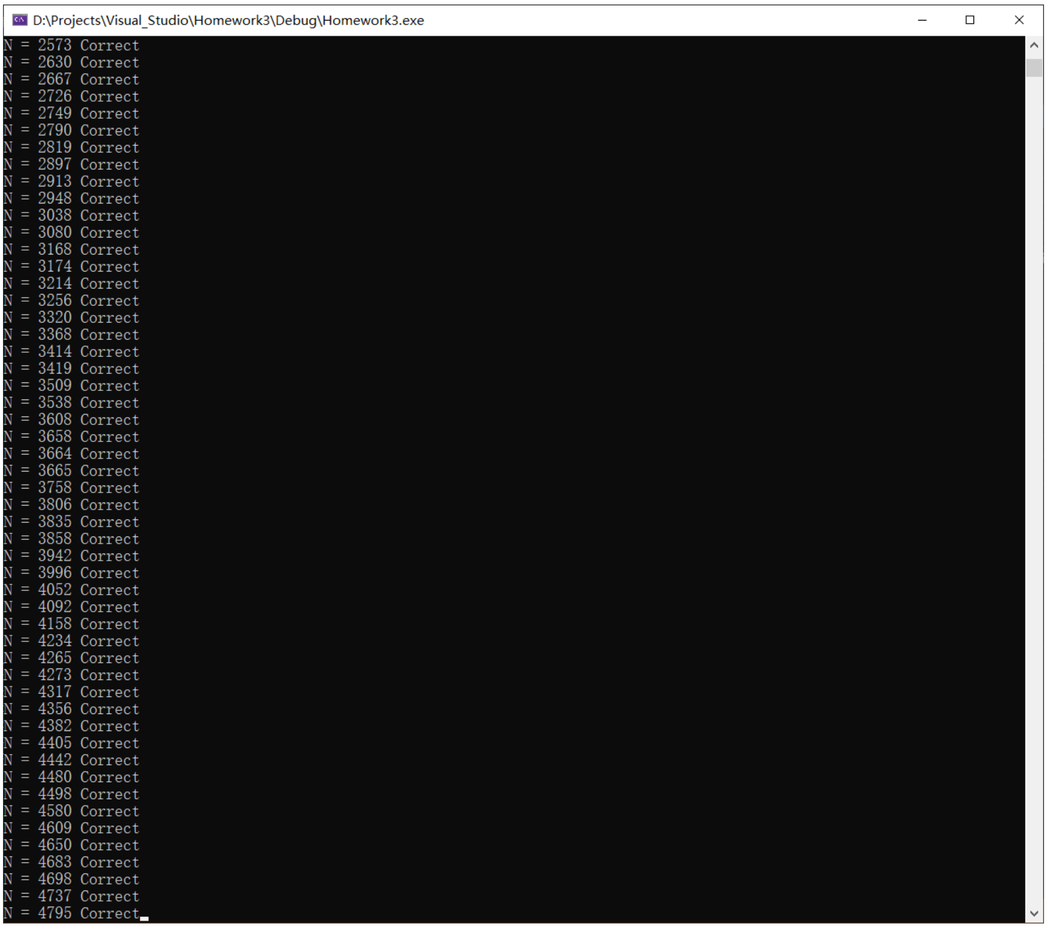

Microsoft Visual Studio 调试控制台
N = 2 Correct
N = 3 Correct
N = 4 Correct
N = 5 Correct
N = 6 Correct
N = 7 Correct
N = 8 Correct
N = 9 Correct
N = 10 Correct
N = 11 Correct
N = 12 Correct
N = 13 Correct
N = 14 Correct
N = 15 Correct
N = 16 Correct
N = 17 Correct
N = 18 Correct
N = 19 Correct
N = 20 Correct
N = 21 Correct
N = 22 Correct
N = 23 Correct
N = 24 Correct
N = 25 Correct
N = 26 Correct
N = 27 Correct
N = 28 Correct
N = 29 Correct
N = 30 Correct
D:\Projects\Visual_Studio\Homework3\Debug\Homework3.exe (进程 11072)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .


Microsoft Visual Studio 调试控制台
N = 100 Correct
N = 141 Correct
N = 208 Correct
N = 242 Correct
N = 242 Correct
N = 311 Correct
N = 335 Correct
N = 413 Correct
N = 471 Correct
N = 533 Correct
N = 597 Correct
N = 602 Correct
N = 647 Correct
N = 728 Correct
N = 755 Correct
N = 816 Correct
N = 907 Correct
N = 1002 Correct
N = 1044 Correct
N = 1071 Correct
N = 1107 Correct
N = 1198 Correct
N = 1202 Correct
N = 1204 Correct
N = 1257 Correct
N = 1349 Correct
N = 1431 Correct
N = 1452 Correct
N = 1468 Correct
N = 1486 Correct
N = 1581 Correct
N = 1628 Correct
N = 1654 Correct
N = 1725 Correct
N = 1763 Correct
N = 1832 Correct
N = 1844 Correct
N = 1911 Correct
N = 2010 Correct
N = 2045 Correct
N = 2139 Correct
N = 2142 Correct
N = 2153 Correct
N = 2175 Correct
N = 2208 Correct
N = 2281 Correct
N = 2345 Correct
N = 2386 Correct
N = 2397 Correct
N = 2450 Correct
N = 2518 Correct
N = 2565 Correct

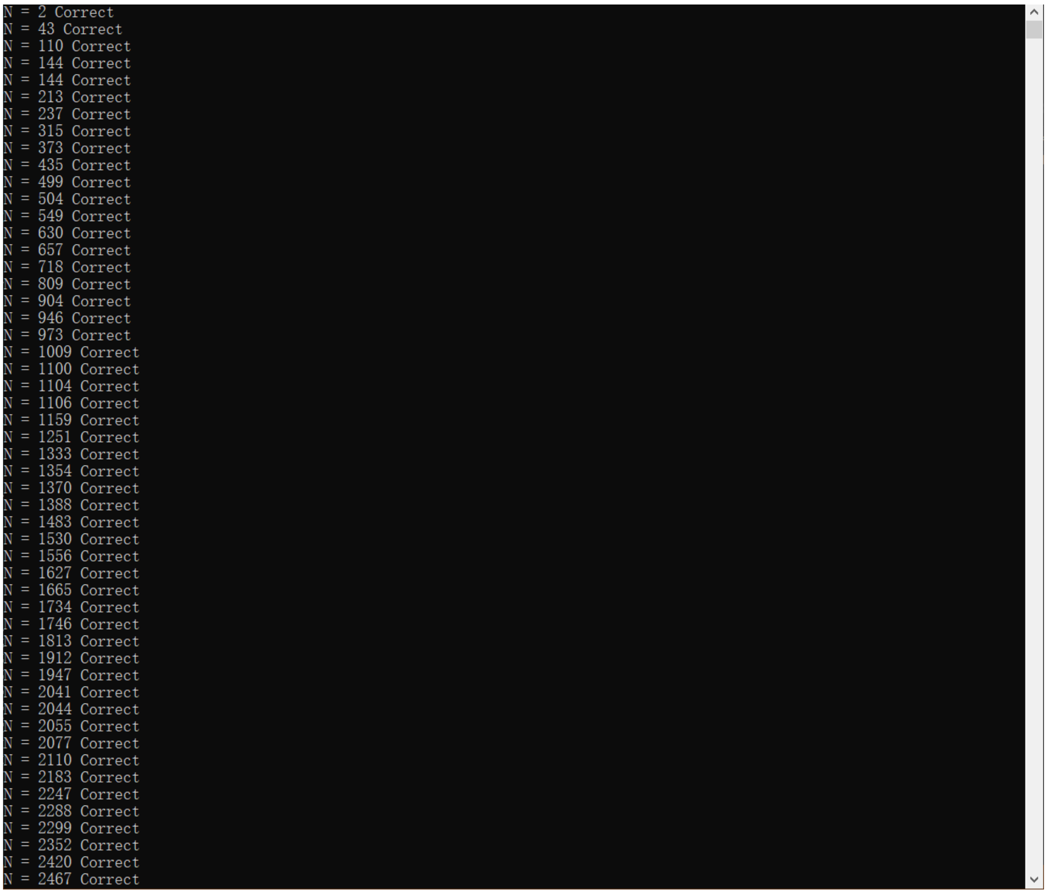
```
Microsoft Visual Studio 调试控制台
N = 2728 Correct
N = 2765 Correct
N = 2824 Correct
N = 2847 Correct
N = 2888 Correct
N = 2917 Correct
N = 2995 Correct
N = 3011 Correct
N = 3046 Correct
N = 3136 Correct
N = 3178 Correct
N = 3266 Correct
N = 3272 Correct
N = 3312 Correct
N = 3354 Correct
N = 3418 Correct
N = 3466 Correct
N = 3512 Correct
N = 3517 Correct
N = 3607 Correct
N = 3636 Correct
N = 3706 Correct
N = 3756 Correct
N = 3762 Correct
N = 3763 Correct
N = 3856 Correct
N = 3904 Correct
N = 3933 Correct
N = 3956 Correct
N = 4040 Correct
N = 4094 Correct
N = 4150 Correct
N = 4190 Correct
N = 4256 Correct
N = 4332 Correct
N = 4363 Correct
N = 4371 Correct
N = 4415 Correct
N = 4454 Correct
N = 4480 Correct
N = 4503 Correct
N = 4540 Correct
N = 4578 Correct
N = 4596 Correct
N = 4678 Correct
N = 4707 Correct
N = 4748 Correct
N = 4781 Correct
N = 4796 Correct
N = 4835 Correct
N = 4893 Correct
N = 4897 Correct
```

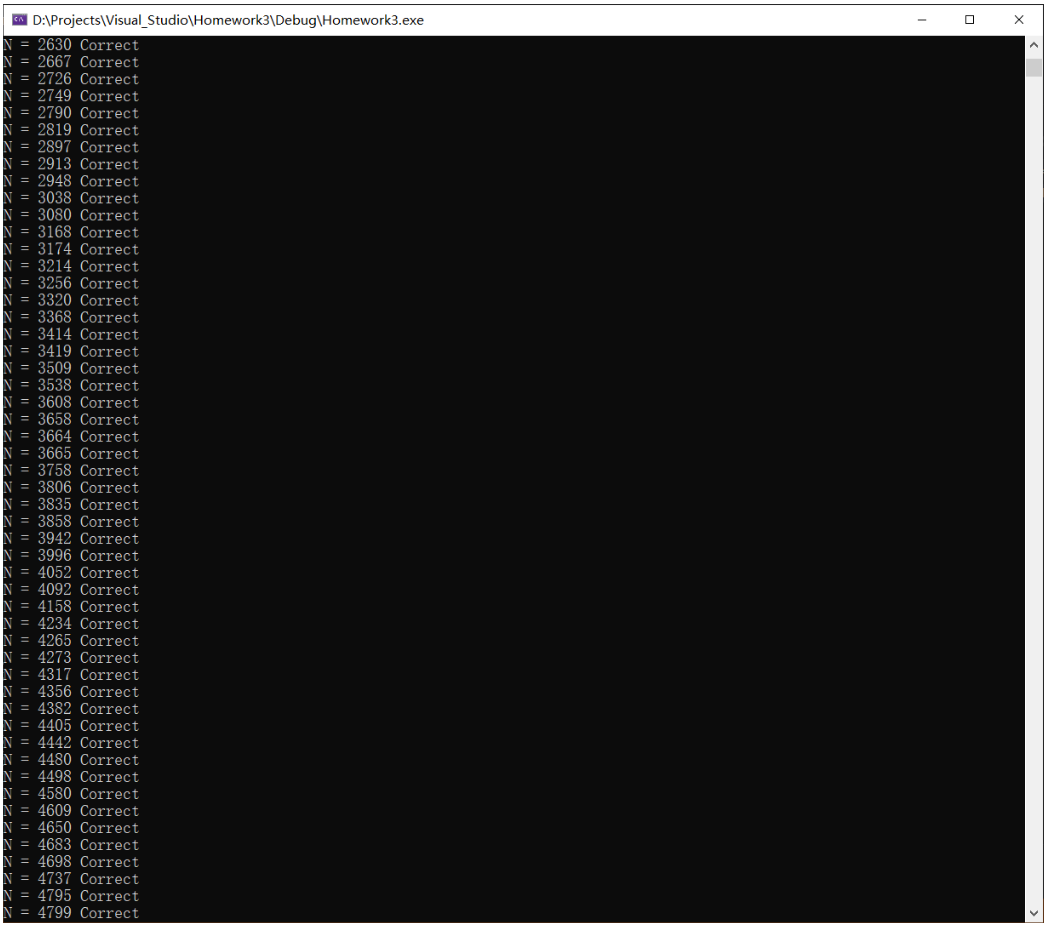
多边形旋转法测试结果：

```
D:\Projects\Visual_Studio\Homework3\Debug\Homework3.exe
N = 2 Correct
N = 43 Correct
N = 110 Correct
N = 144 Correct
N = 144 Correct
N = 213 Correct
N = 237 Correct
N = 315 Correct
N = 373 Correct
N = 435 Correct
N = 499 Correct
N = 504 Correct
N = 549 Correct
N = 630 Correct
N = 657 Correct
N = 718 Correct
N = 809 Correct
N = 904 Correct
N = 946 Correct
N = 973 Correct
N = 1009 Correct
N = 1100 Correct
N = 1104 Correct
N = 1106 Correct
N = 1159 Correct
N = 1251 Correct
N = 1333 Correct
N = 1354 Correct
N = 1370 Correct
N = 1388 Correct
N = 1483 Correct
N = 1530 Correct
N = 1556 Correct
N = 1627 Correct
N = 1665 Correct
N = 1734 Correct
N = 1746 Correct
N = 1813 Correct
N = 1912 Correct
N = 1947 Correct
N = 2041 Correct
N = 2044 Correct
N = 2055 Correct
N = 2077 Correct
N = 2110 Correct
N = 2183 Correct
N = 2247 Correct
N = 2288 Correct
N = 2299 Correct
N = 2352 Correct
N = 2420 Correct
N = 2467 Correct
```


D:\Projects\Visual_Studio\Homework3\Debug\Homework3.exe
N = 2573 Correct
N = 2630 Correct
N = 2667 Correct
N = 2726 Correct
N = 2749 Correct
N = 2790 Correct
N = 2819 Correct
N = 2897 Correct
N = 2913 Correct
N = 2948 Correct
N = 3038 Correct
N = 3080 Correct
N = 3168 Correct
N = 3174 Correct
N = 3214 Correct
N = 3256 Correct
N = 3320 Correct
N = 3368 Correct
N = 3414 Correct
N = 3419 Correct
N = 3509 Correct
N = 3538 Correct
N = 3608 Correct
N = 3658 Correct
N = 3664 Correct
N = 3665 Correct
N = 3758 Correct
N = 3806 Correct
N = 3835 Correct
N = 3858 Correct
N = 3942 Correct
N = 3996 Correct
N = 4052 Correct
N = 4092 Correct
N = 4158 Correct
N = 4234 Correct
N = 4265 Correct
N = 4273 Correct
N = 4317 Correct
N = 4356 Correct
N = 4382 Correct
N = 4405 Correct
N = 4442 Correct
N = 4480 Correct
N = 4498 Correct
N = 4580 Correct
N = 4609 Correct
N = 4650 Correct
N = 4683 Correct
N = 4698 Correct
N = 4737 Correct
N = 4795 Correct

直接构造法：


N = 2 Correct
N = 43 Correct
N = 110 Correct
N = 144 Correct
N = 144 Correct
N = 213 Correct
N = 237 Correct
N = 315 Correct
N = 373 Correct
N = 435 Correct
N = 499 Correct
N = 504 Correct
N = 549 Correct
N = 630 Correct
N = 657 Correct
N = 718 Correct
N = 809 Correct
N = 904 Correct
N = 946 Correct
N = 973 Correct
N = 1009 Correct
N = 1100 Correct
N = 1104 Correct
N = 1106 Correct
N = 1159 Correct
N = 1251 Correct
N = 1333 Correct
N = 1354 Correct
N = 1370 Correct
N = 1388 Correct
N = 1483 Correct
N = 1530 Correct
N = 1556 Correct
N = 1627 Correct
N = 1665 Correct
N = 1734 Correct
N = 1746 Correct
N = 1813 Correct
N = 1912 Correct
N = 1947 Correct
N = 2041 Correct
N = 2044 Correct
N = 2055 Correct
N = 2077 Correct
N = 2110 Correct
N = 2183 Correct
N = 2247 Correct
N = 2288 Correct
N = 2299 Correct
N = 2352 Correct
N = 2420 Correct
N = 2467 Correct



四、效率分析

4.1 理论分析

- 暴力深搜法

  由于嵌套两个深度为N的DFS，因此复杂度为$O(N!N!)$

- 分治法

  每次将原问题划分为两个N/2的子问题，合并时复杂度为$O(N^2)$

  则$T(N) = T(N/2) + O(N^2)$

  运用主定理可知，时间复杂度为$O(N^2)$

- 多边形旋转法

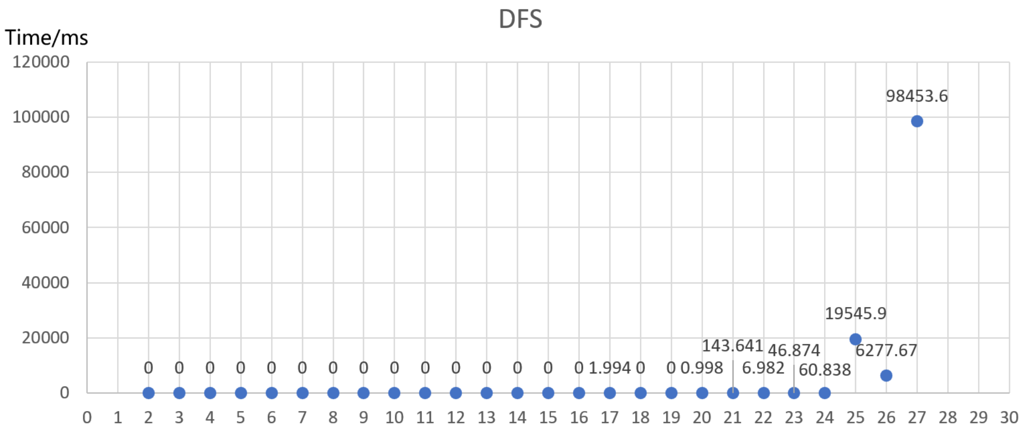
  遍历N天，每天遍历N位运动员，时间复杂度为$O(N^2)$

- 直接构造法

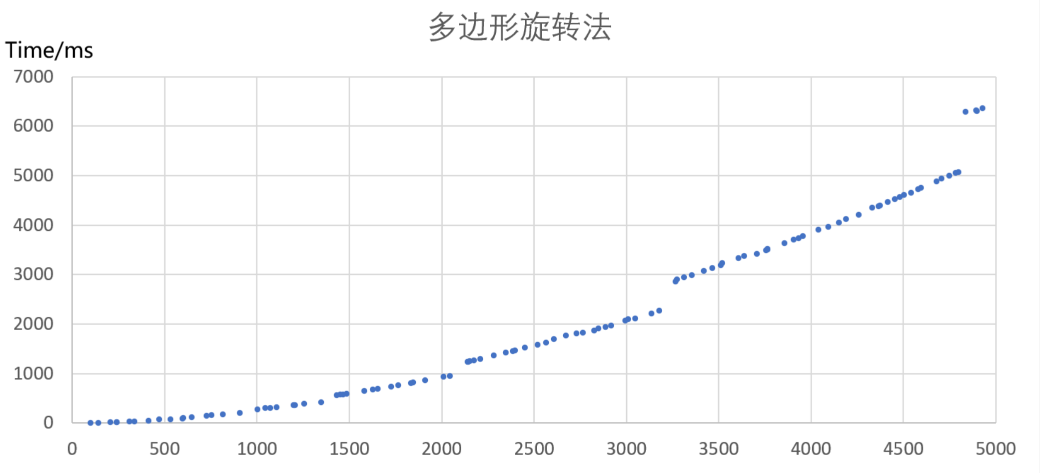
  遍历N天，每天遍历N位运动员，时间复杂度为$O(N^2)$

4.2 实际测试

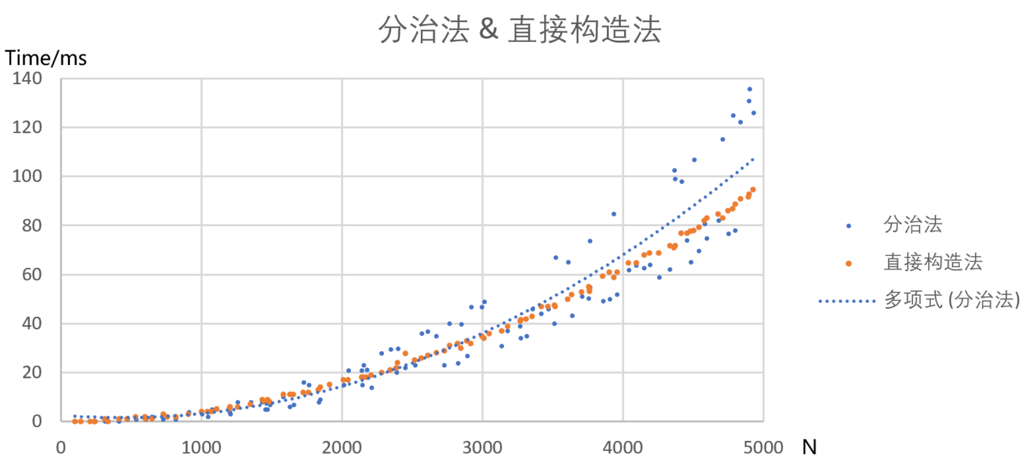
暴力深搜法



多边形旋转法



分治法和直接构造法



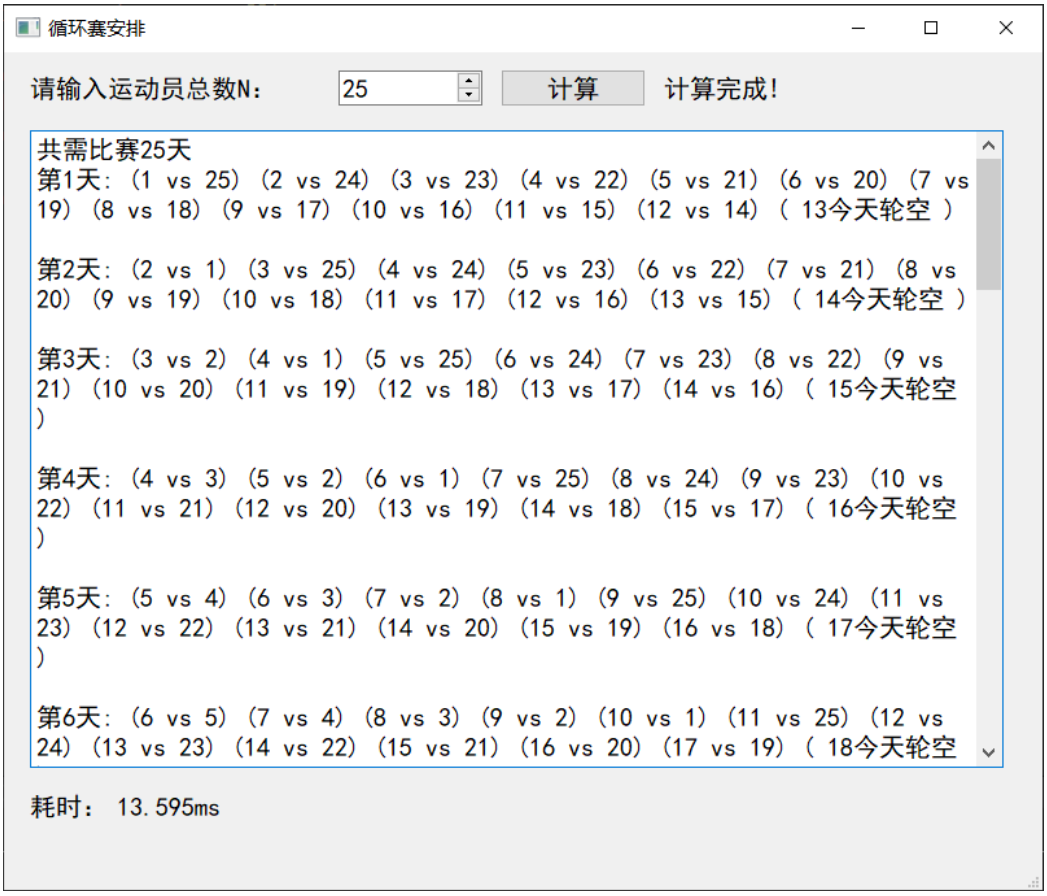
可以看出，效率最低的是暴力深搜法；

多边形旋转法与分治法、直接构造法同为$O(N^2)$算法，但效率低很多，推测是多边形旋转法中大量使用vector的插入、删除操作，导致效率低下（stl **慢慢慢！**）

在分治法与直接构造法的比较中，分治法会由于N的奇偶性产生波动，而直接构造法较稳定；用$F=N^2$曲线拟合分治法散点，得到的结构显示直接构造法略胜一筹。

五、设计应用程序

由测试结果可知，直接构造法效率最高。使用该算法，通过GUI设计软件**Qt**创建一个图形化的应用程序（作业目录下的**循环赛安排.exe**），供用户使用：



六、实验总结

在本次实验中，我探究了循环赛问题的四种解法。除去课堂上分析的分治法，我继续深入思考，通过几何构造，探究出多边形旋转构造法；在此基础上，进一步提出了直接构造法。测试结果表示，其效率略微超过了分治法。通过探究与思考，我设计的算法成功超越了传统的分治法，这令我成就感十足。