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

1.1 设备规格

设备规格

Legion R7000P2020H

设备名称	PC
处理器	AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz
机带 RAM	16.0 GB (15.9 GB 可用)
设备 ID	370678B7-1FA2-4C35-8BAB-F92D3429406B
产品 ID	00342-35932-44511-AAOEM
系统类型	64 位操作系统, 基于 x64 的处理器
笔和触控	为 10 触摸点提供笔和触控支持

1.2 操作系统

Windows 规格

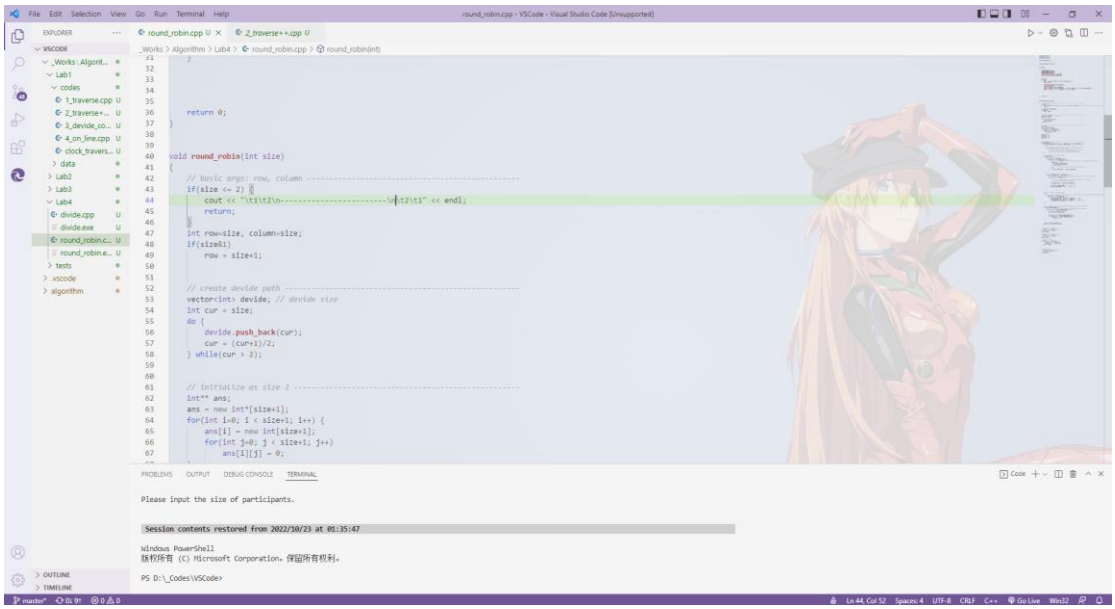
版本	Windows 10 家庭中文版
版本号	21H2
安装日期	2022/4/5
操作系统内部版本	19044.2006
序列号	PF2660CA
体验	Windows Feature Experience Pack 120.2212.4180.0

1.3 编程语言&编译器

C++11

```
version : MinGW-W64-builds-4.3.5
user    : nixman
date    : 05.12.2018-10:29:36 AM
```

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

```cpp
int main()
{
    // time clock
    LARGE_INTEGER cpuFreq;
    LARGE_INTEGER startTime;
    LARGE_INTEGER endTime;
    QueryPerformanceFrequency(&cpuFreq);
    QueryPerformanceCounter(&startTime);


    int size;
    while(1) {
        cout << "Please input the size of participants.\n";
        cin >> size;

        QueryPerformanceCounter(&startTime);
        round_robin(size);
        QueryPerformanceCounter(&endTime);
        double runTime = ((endTime.QuadPart - startTime.QuadPart)*1.0f / cpuFreq.QuadPart);
        cout << "Calculate in " << runTime << "s.\n" << endl;
    }


    return 0;
}
```

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）

```cpp
    // basic args: row, column ------------------------------------------------
    if(size <= 2) {
        cout << "\t1\t2\n----------------------\n\t2\t1" << endl;
        return;
    }
    int row=size, column=size;
    if(size&1)
        row = size+1;


    // create devide path -----------------------------------------------------
    vector<int> devide; // devide size
    int cur = size;
    do {
        devide.push_back(cur);
        cur = (cur+1)/2;
    } while(cur > 2);
```

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。

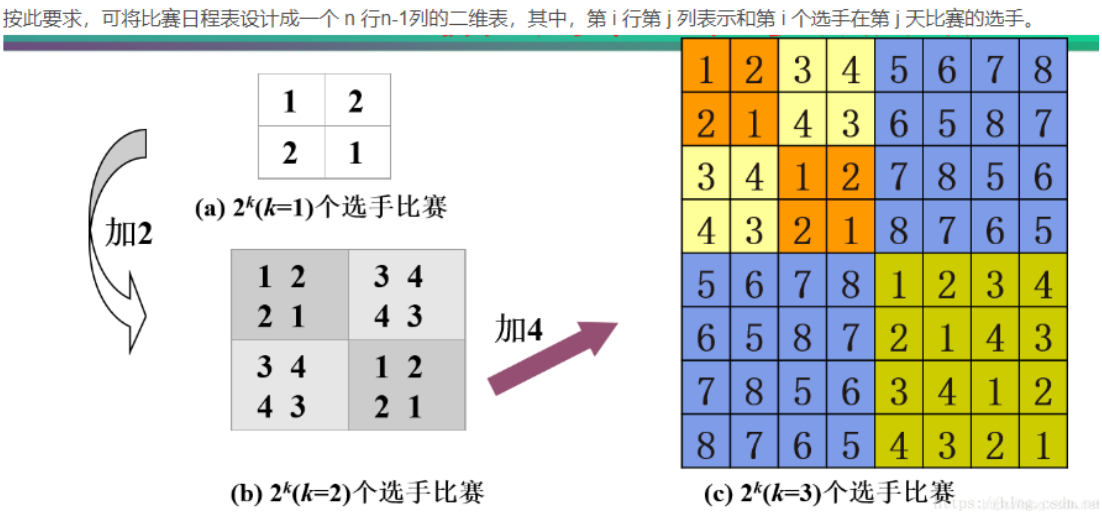
```
59      // initialize as size 2 ----------------------------------------
60      int** ans;
61      ans = new int*[size+1];
62      for(int i=0; i < size+1; i++) {
63          ans[i] = new int[size+1];
64          for(int j=0; j < size+1; j++)
65              ans[i][j] = 0;
66      }
67      ans[0][0] = 1;
68      ans[0][1] = 2;
69      ans[1][0] = 2;
70      ans[1][1] = 1;
```

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。

```
// 2 -- even size to combine as even
else {
    // 2.1 -- generate right, under, right-under (scopy to the three square perhaps with adding tmp_size)
    for(int i=0; i < tmp_size; i++) {
        for(int j = 0; j < tmp_size; j++) {
            ans[i][tmp_size+j] = ans[i][j] + tmp_size;  // right
            ans[tmp_size+i][j] = ans[i][j] + tmp_size;  // under
            ans[tmp_size+i][tmp_size+j] = ans[i][j];    // right-under
        }
    }
}
```

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。

```
85          // 1 -- odd size to combine as even
86          if(tmp_size&1) {
87              // 1.1 -- generate right
88              for(int i=0; i < tmp_size+1; i++) {
89                  for(int j = 0; j < tmp_size; j++) {
90                      // 1.1.1 -- 0 to insert the rest rival( left 0-insert with relative right insert
91                      if(ans[i][j] == 0) {
92                          ans[i][j] = j+1+tmp_size;
93                          ans[i][tmp_size+j] = j+1;
94                      }
95                      // 1.1.2 -- normal add tmp_size moving to relative location
96                      else
97                          ans[i][tmp_size+j] = ans[i][j]+tmp_size;
98                  }
99              }
```

- 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。

```
101             // 1.2 -- generate under, right-under
102             for(int i=tmp_size+1; i < tmp_size*2; i++) {
103                 // 1.2.1 -- traverse into line = [tmp_size+2, tmp_size*2] for inserting all units
104                 for(int j=0; j < tmp_size; j++) {
105                     /*
106                         1.2.1.1 -- left index is i, calculate right index to swap_insert
107                         THIS IS THE MOST IMPORTANT SOLUTION
108                     */
109                     int right_index = (i+j)%tmp_size+tmp_size;
110 
111                     ans[i][j] = right_index+1;  // left insert
112                     ans[i][right_index] = j+1;  // right insert
113                 }
114             }
115         }
```

这个强推论就是图中描述的：

n = 6时的最后两行的数据的填充即：

```
第五行：1 -> 5, 2 -> 6, 3 -> 4
第六行：1 -> 6, 2 -> 4, 3 -> 5
```

n = 10时的最后两行的数据的填充即：

```
第七行：1 -> 7, 2 -> 8, 3 -> 9, 4 -> 10, 5 -> 6
第八行：1 -> 8, 2 -> 9, 3 -> 10, 4 -> 6, 5 -> 7
第九行：1 -> 9, 2 -> 10, 3 -> 6, 4 -> 7, 5 -> 8
第十行：1 -> 10, 2 -> 6, 3 -> 7, 4 -> 8, 5 -> 9
```

那么，很明显的：最后几行的呈现的规律是：
结论：前n / 2列的数据与后n/2列的数据在轮转的对应赋值。

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。

```
129         // 3 -- dgenerate size is even and delete something to match odd devide size
130         if(devide[target_index]&1) {
131             // 3.1 -- traverse into each line=devide[target_index]+1 for deleting
132             for(int i=0; i < devide[target_index]+1; i++) {
133                 // 3.1.1 -- clear last column
134                 ans[i][devide[target_index]] = 0;
135
136                 // 3.1.2 -- clear number need[target_index]
137                 for(int j=0; j < devide[target_index]+1; j++)
138                     if(ans[i][j] == devide[target_index]+1)
139                         ans[i][j] = 0;
140             }
141         }
```

2.4.5 combine 逻辑

```
73      // start generate ------------------------------------------------------------
74      int tmp_size = 2;
75      for(int target_index=devide.size()-1; target_index >= 0; target_index--) {
76          //cout << "devide = " << devide[target_index] << endl;
77          /*
78              generate answer is always even, but what we need perhaps be odd
79              so generate odd devide, we should do:
80                  1. generate right && 0 unit inserting the rest couple-rival
81                  2. insert the rest lines with INPORTANT swap_insert
82                  3. clear last column and clear number need[target_index]
83          */
84
85          // 1 -- odd size to combine as even
86          if(tmp_size&1) {…
116
117         // 2 -- even size to combine as even
118         else {…
128
129         // 3 -- dgenerate size is even and delete something to match odd devide size
130         if(devide[target_index]&1) {…
142
143         // 4 -- refresh tmp_size
144         tmp_size = devide[target_index];
145     }
```

2.4.6 输出函数

```
148     // output -- first line
149     for(int i=0; i < column; i++)
150         cout << "\t" << i+1;
151     cout << endl;
152 
153     // output -- "--------"
154     for(int i=0; i < column+1; i++)
155         cout << "--------";
156     cout << endl;
157 
158     // output -- table
159     for(int i=1; i < row; i++) {
160         for(int j=0; j < column; j++)
161             cout << "\t" << ans[i][j];
162         cout << endl;
163     }
```

终端运行大概长这样:

```
PS D:\_Codes\VSCode\_Works\Algorithm\Lab4> cd "d:\_Codes\VSCode\_Works\Algorithm\Lab4\" ; if ($?) { g++ round_robin.cpp -o round_robin } ; if ($?)
{ .\round_robin }
Please input the size of participants.
13
        1       2       3       4       5       6       7       8       9       10      11      12      13
---------------------------------------------------------------------------------------------------------------
        2       1       4       3       6       5       0       9       8       11      10      13      12
        3       4       1       2       7       13      5       10      11      8       9       0       6
        4       3       2       1       12      7       6       11      10      9       8       5       0
        5       6       7       11      1       2       3       12      13      0       4       8       9
        6       5       10      7       2       1       4       13      12      3       0       9       8
        7       9       5       6       3       4       1       0       2       12      13      10      11
        8       7       6       5       4       3       2       1       0       13      12      11      10
        9       10      11      12      13      0       8       7       1       2       3       4       5
        10      11      12      13      0       8       9       6       7       1       2       3       4
        11      12      13      0       8       9       10      5       6       7       1       2       3
        12      13      0       8       9       10      11      4       5       6       7       1       2
        13      0       8       9       10      11      12      3       4       5       6       7       1
        0       8       9       10      11      12      13      2       3       4       5       6       7
Calculate in 0.024718s.

Please input the size of participants.
```

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

```
Please input the size of participants.
6
        1       2       3       4       5       6
-----------------------------------------------------
        2       1       6       5       4       3
        3       5       1       6       2       4
        4       3       2       1       6       5
        5       6       4       3       1       2
        6       4       5       2       3       1
Calculate in 0.0062821s.
```

7：4+4-1，测试第二个和第三个逻辑分支

```
Please input the size of participants.
7
        1       2       3       4       5       6       7
---------------------------------------------------------------
        2       1       4       3       6       5       0
        3       4       1       2       7       0       5
        4       3       2       1       0       7       6
        5       6       7       0       1       2       3
        6       5       0       7       2       1       4
        7       0       5       6       3       4       1
        0       7       6       5       4       3       2
Calculate in 0.0074466s.
```

8：4+4，测试第三个逻辑分支

```
Please input the size of participants.
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
Calculate in 0.009871s.
```

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。

```
Please input the size of participants.
1
        1       2
------------------------
        2       1
Calculate in 0.0003832s.

Please input the size of participants.
2
        1       2
------------------------
        2       1
Calculate in 0.0003466s.

Please input the size of participants.
3
        1       2       3
---------------------------------
        2       1       0
        3       0       1
        0       3       2
Calculate in 0.0034427s.

Please input the size of participants.
4
        1       2       3       4
---------------------------------------
        2       1       4       3
        3       4       1       2
        4       3       2       1
Calculate in 0.0038547s.
```

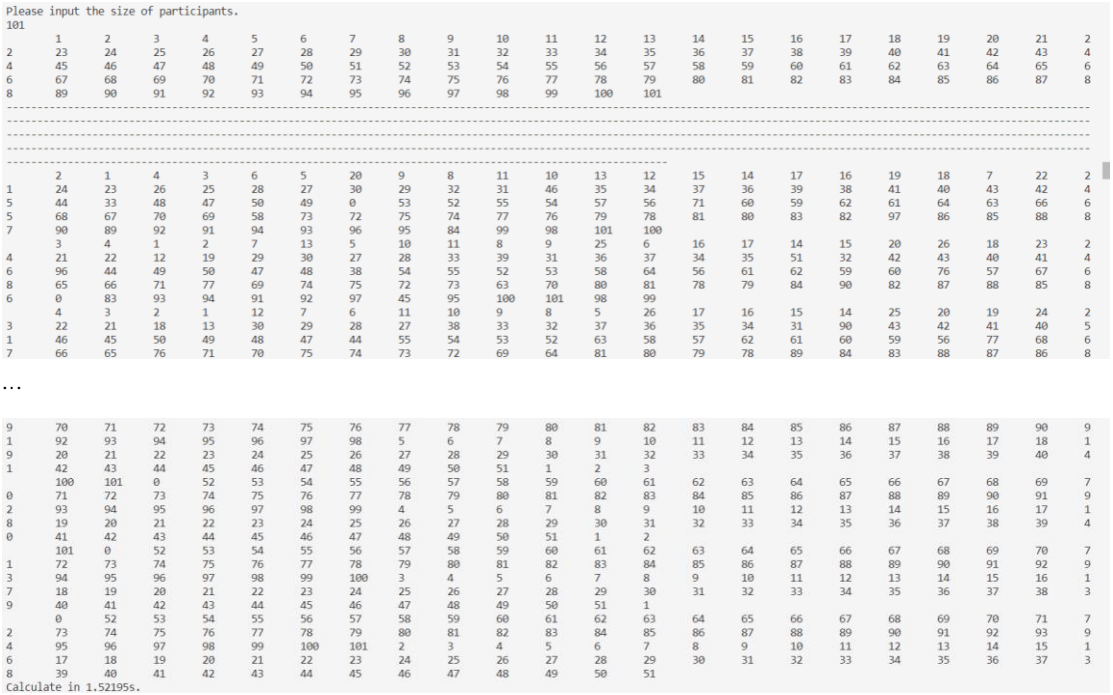
```
        1       2       3       4       5
-----------------------------------------------
        2       1       0       5       4
        3       5       1       0       2
        4       3       2       1       0
        5       0       4       3       1
        0       4       5       2       3
Calculate in 0.005137s.

Please input the size of participants.
6
        1       2       3       4       5       6
-------------------------------------------------------
        2       1       6       5       4       3
        3       5       1       6       2       4
        4       3       2       1       6       5
        5       6       4       3       1       2
        6       4       5       2       3       1
Calculate in 0.0060447s.

Please input the size of participants.
7
        1       2       3       4       5       6       7
---------------------------------------------------------------
        2       1       4       3       6       5       0
        3       4       1       2       7       0       5
        4       3       2       1       0       7       6
        5       6       7       0       1       2       3
        6       5       0       7       2       1       4
        7       0       5       6       3       4       1
        0       7       6       5       4       3       2
Calculate in 0.0082565s.

Please input the size of participants.
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
Calculate in 0.0091285s.
```

- 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。

```
Please input the size of participants.
101
Calculate in 0.0002694s.

Please input the size of participants.
```

发现计算复杂度很小。

- 测试 10000

```
Please input the size of participants.
10000
Calculate in 0.480561s.

Please input the size of participants.
```

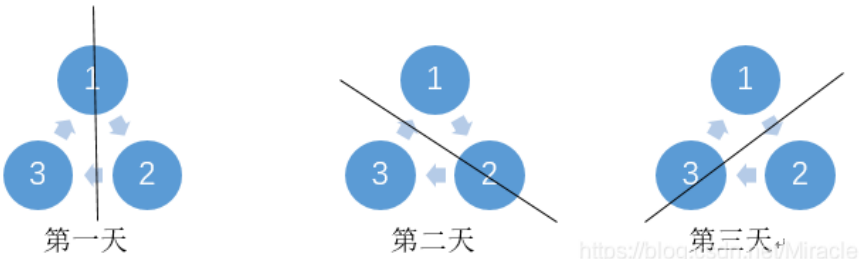
没有 IO 显得计算时间很小。

- 测试 20000

```
Please input the size of participants.
20000
Calculate in 2.14507s.
```

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）

```cpp
// odd size
if(size&1) {
    int index_left, index_right;
    for(int i=0; i < size; i++) {
        index_left = i;
        index_right = rest(i+1, column);

        while(index_left != index_right) {
            ans[i+1][index_left] = index_right+1;
            ans[i+1][index_right] = index_left+1;
            cout << index_left << " " << index_right << endl;
            index_left = rest(index_left-1, size);
            index_right = rest(index_right+1, size);
        }
    }
}

// even size
else {
    int index_left, index_right;
    for(int i=0; i < size-1; i++) {
        index_left = i;
        index_right = rest(i+1, size);

        do {
            ans[i+1][index_left] = index_right+1;
            ans[i+1][index_right] = index_left+1;
            cout << index_left << " " << index_right << endl;
            index_left = rest(index_left-1, size);
            index_right = rest(index_right+1, size);
        } while(index_left != index_right-1);
    }
}
```

- 测试 13

```
       1       2       3       4       5       6       7       8       9       10      11      12      13
-------------------------------------------------------------------------------------------------------------
       2       1       13      12      11      10      9       0       7       6       5       4       3
       4       3       2       1       13      12      11      10      0       8       7       6       5
       6       5       4       3       2       1       13      12      11      0       9       8       7
       8       7       6       5       4       3       2       1       13      12      0       10      9
       10      9       8       7       6       5       4       3       2       1       13      0       11
       12      11      10      9       8       7       6       5       4       3       2       1       0
       0       13      12      11      10      9       8       7       6       5       4       3       2
       3       0       1       13      12      11      10      9       8       7       6       5       4
       5       4       0       2       1       13      12      11      10      9       8       7       6
       7       6       5       0       3       2       1       13      12      11      10      9       8
       9       8       7       6       0       4       3       2       1       13      12      11      10
       11      10      9       8       7       0       5       4       3       2       1       13      12
       13      12      11      10      9       8       0       6       5       4       3       2       1
Calculate in 0.0431997s.
```

差不多可以。

- 测试 10000（屏蔽输出）

```
Please input the size of participants.
10000
Calculate in 0.75883s.
```

多边形运行时间居然比分治长！

- 测试 20000（屏蔽输出）

```
Please input the size of participants.
20000
Calculate in 3.07178s.
```

- 测试 30000（屏蔽输出）

```
Please input the size of participants.
30000
Calculate in 6.90664s.
```

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 $T(n) = (n/2)^2 + (n/4)^2 \cdots + 1$

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）