

《校外实训（一）》（郑宇辉）

2022-2023学年 第3学期

本周课堂讨论主题

- 计算思维：枚举（穷举）

- 定义
- 应用场景
- 经典案例
- 注意事项
- 参考代码
- 推荐资源

- 协作文档：

<https://kdocs.cn/l/cvIcSx8wrQgU>

[金山文档] 计算思维（枚举）.docx

- 演示任务：通过一个具体案例，演示编码解决过程

- 1、案例说明
- 2、编码过程（问题分析、IPO设计、算法讲解）
- 3、结果演示

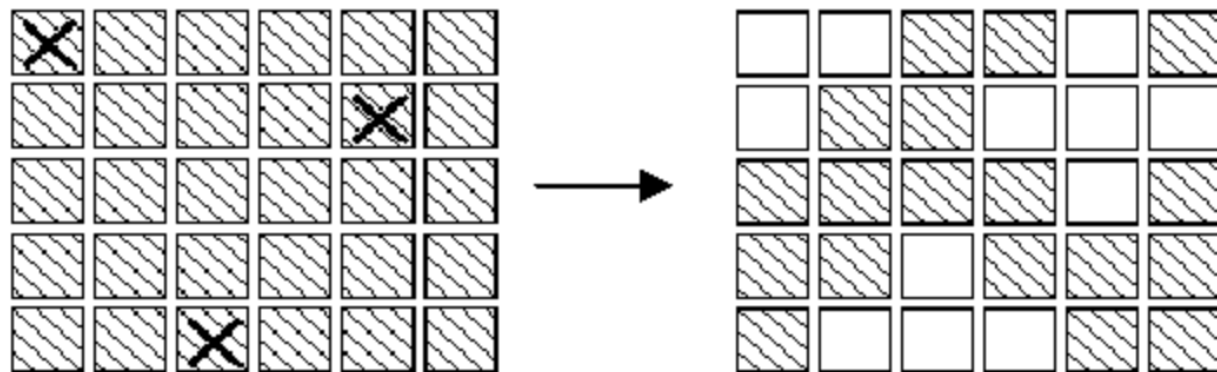
计算思维之常用算法设计：

<https://wenku.baidu.com/view/816e18796037ee06eff9aef8941ea76e59fa4a71.html>

熄灯问题

描述

有一个由按钮组成的矩阵，其中每行有6个按钮，共5行。每个按钮的位置上有一盏灯。当按下一个按钮后，该按钮以及周围位置(上边、下边、左边、右边)的灯都会改变一次。即，如果灯原来是点亮的，就会被熄灭；如果灯原来是熄灭的，则会被点亮。在矩阵角上的按钮改变3盏灯的状态；在矩阵边上的按钮改变4盏灯的状态；其他的按钮改变5盏灯的状态。



原文链接:

<http://www.xmuoj.com/contest/50/problem/w2p04>

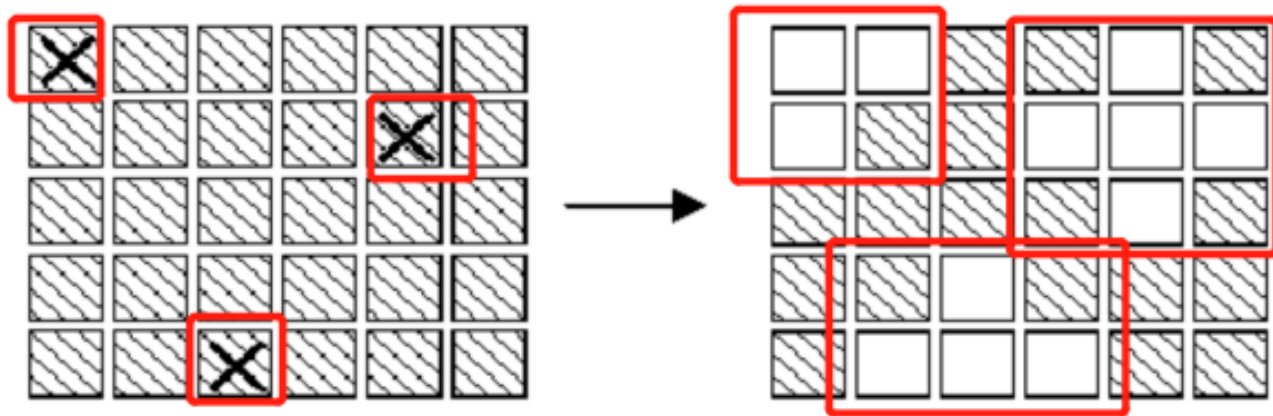
编程解决问题的步骤

- **分析问题**：分析问题的计算部分，**想清楚**；【分解问题】
- **划分边界**：划分问题的功能边界，**规划IPO**；【input、process、output】
- **设计算法**：设计问题的求解算法，**关注算法**；
- **编写程序**：编写问题的计算程序，**编程序**；
- **调试测试**：调试程序使正确运行，**运行调试**；
- **升级维护**：适应问题的升级维护，**更新完善**；

熄灯问题---分析问题

描述

有一个由按钮组成的矩阵，其中每行有6个按钮，共5行。每个按钮的位置上有一盏灯。当按下一个按钮后，该按钮以及周围位置(上边、下边、左边、右边)的灯都会改变一次。即，如果灯原来是点亮的，就会被熄灭；如果灯原来是熄灭的，则会被点亮。在矩阵角上的按钮改变3盏灯的状态；在矩阵边上的按钮改变4盏灯的状态；其他的按钮改变5盏灯的状态。



原文链接:

<http://www.xmuoj.com/contest/50/problem/w2p04>

熄灯问题---搜索资料

- 不要直接搜索“熄灯问题”：
- 建议搜索“枚举算法”，并做在线总结的学习笔记：
 - <https://kdocs.cn/l/cvIcSx8wrQgU>
 - [金山文档] 计算思维（枚举）.docx
- 判断搜索的质量：是否收录、转载数、作者、收藏数（点赞数）、阅读数等
- [OJ习题详解 1222: EXTENDED LIGHTS OUT, 2811: 熄灯问题 C++算法 枚举](#)

分析问题

- 每个按钮最多按一次
- 各个按钮被按下的顺序对最终的结果没有影响
- 对第1行中每盏点亮的灯，按下第2行对应的按钮，就可以熄灭第1行的全部灯如此重复下去，可以熄灭第1，2，3，4行的全部灯（剩下第5行的状态有待检测）。
- 同理，按下第1、2、3、4、5列的按钮灯，可以熄灭前5列的灯（剩下的第6列的状态有待检测）。

枚举——熄灯问题

<https://www.jianshu.com/p/972af4bbf2a6>

解题思路

1. 枚举所有可能的按钮状态，每种状态计算一下最后的情况，看是否都熄灭。所有状态数为 2^{30} ，因此这种方案不可行。
2. 如果存在某个局部，一旦这个局部的状态确定，那么剩下的其它状态只能是确定的一种，或不多的n种，则只需要枚举这个局部即可。以第一行为例，假设它就是那个局部，如果第一行的状态确定了，是不是第二行的状态就确定了呢？答案是是的，因为第一行按钮按过之后，亮的灯只有按第二行才能将其熄灭。同理，第二行按钮按下后，只能通过第三行按钮来控制灯熄灭。
3. 枚举第一行的所有可能状态，每个位置有0和1两种状态，共6个位置，因此第一行的所有可能状态为 $2^6=64$ 种，枚举状态可以通过递归实现。如果使用每个比特位代表一个灯的话，则可能的状态为数字0-63。

枚举——熄灯问题

<https://www.jianshu.com/p/972af4bbf2a6>

ACM题目中输入数据的处理 (C++版)

□ 模式1：最简单的输入，接受一组输入，针对这组输入计算出值即可。

- `cin >> a >> b;`

□ 模式2：要输入多组数据，直到读至输入文件末尾 (EOF) 为止。

- `while (cin >> a >> b) { }`

□ 模式3：一次运行，要输入多组数据，组数由第一个输入数据决定（在开始的时候输入一个N，接下来是N组数据）。

- `cin >> n; for (int i = 0; i < n; i++) { cin >> a >> b; }`

□ 模式4：输入不说明有多少组数据，但以某个特殊输入为结束标志。构造循环对数据进行处理，将是否遇到了要求结束的输入，作为循环是否结束的依据。

- `while (cin >> a >> b && (a || b)) { }`

原文链接：https://blog.csdn.net/sxhelijian/article/details/8978850?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_title-0&spm=1001.2101.3001.4242

处理输入和输出

输入

5行组成，每一行包括6个数字（0或1）。

相邻两个数字之间用单个空格隔开。

0表示灯的初始状态是熄灭的，1表示灯的初始状态是点亮的。

输出

5行组成，每一行包括6个数字（0或1）。

相邻两个数字之间用单个空格隔开。

其中的1表示需要把对应的按钮按下，0则表示不需要按对应的按钮。

输入样例 1

```
2
0 1 1 0 1 0
1 0 0 1 1 1
0 0 1 0 0 1
1 0 0 1 0 1
0 1 1 1 0 0
0 0 1 0 1 0
1 0 1 0 1 1
0 0 1 0 1 1
1 0 1 1 0 0
0 1 0 1 0 0
```

输出样例 1

```
PUZZLE #1
1 0 1 0 0 1
1 1 0 1 0 1
0 0 1 0 1 1
1 0 0 1 0 0
0 1 0 0 0 0
PUZZLE #2
1 0 0 1 1 1
1 1 0 0 0 0
0 0 0 1 0 0
1 1 0 1 0 1
1 0 1 1 0 1
```

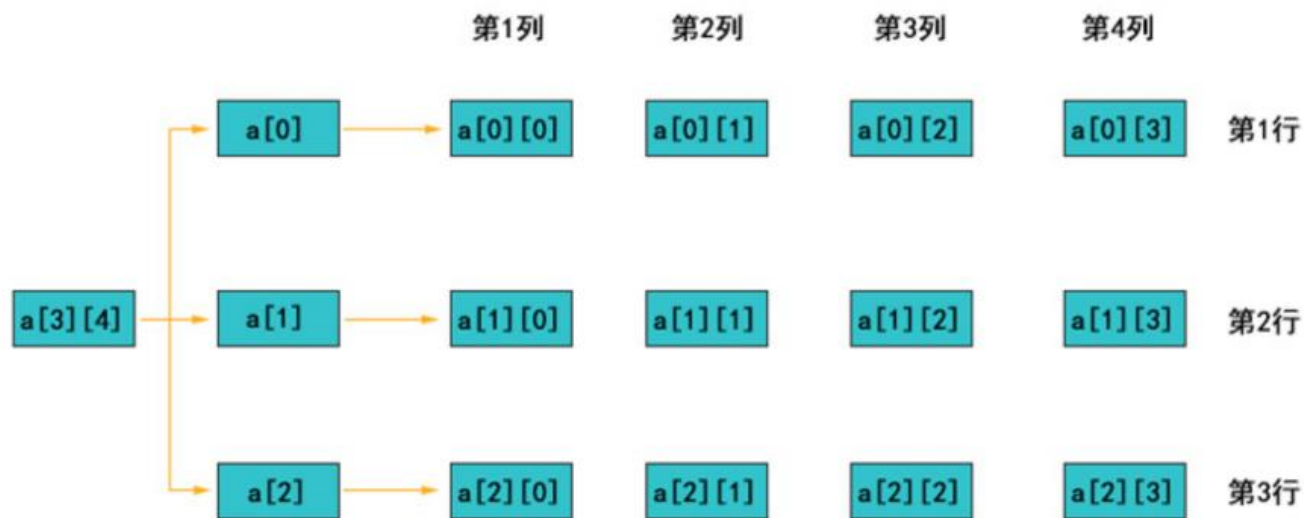
注意：PUZZLE行结尾没有空格，数字行最后有一个空格。

• 问题分解：IPO

数据结构：二维数组

```
int a[3][4];
```

在上述定义的二维数组中，共包含3*4个元素，即12个元素。接下来，通过一张图来描述二维数组a的元素分布情况。

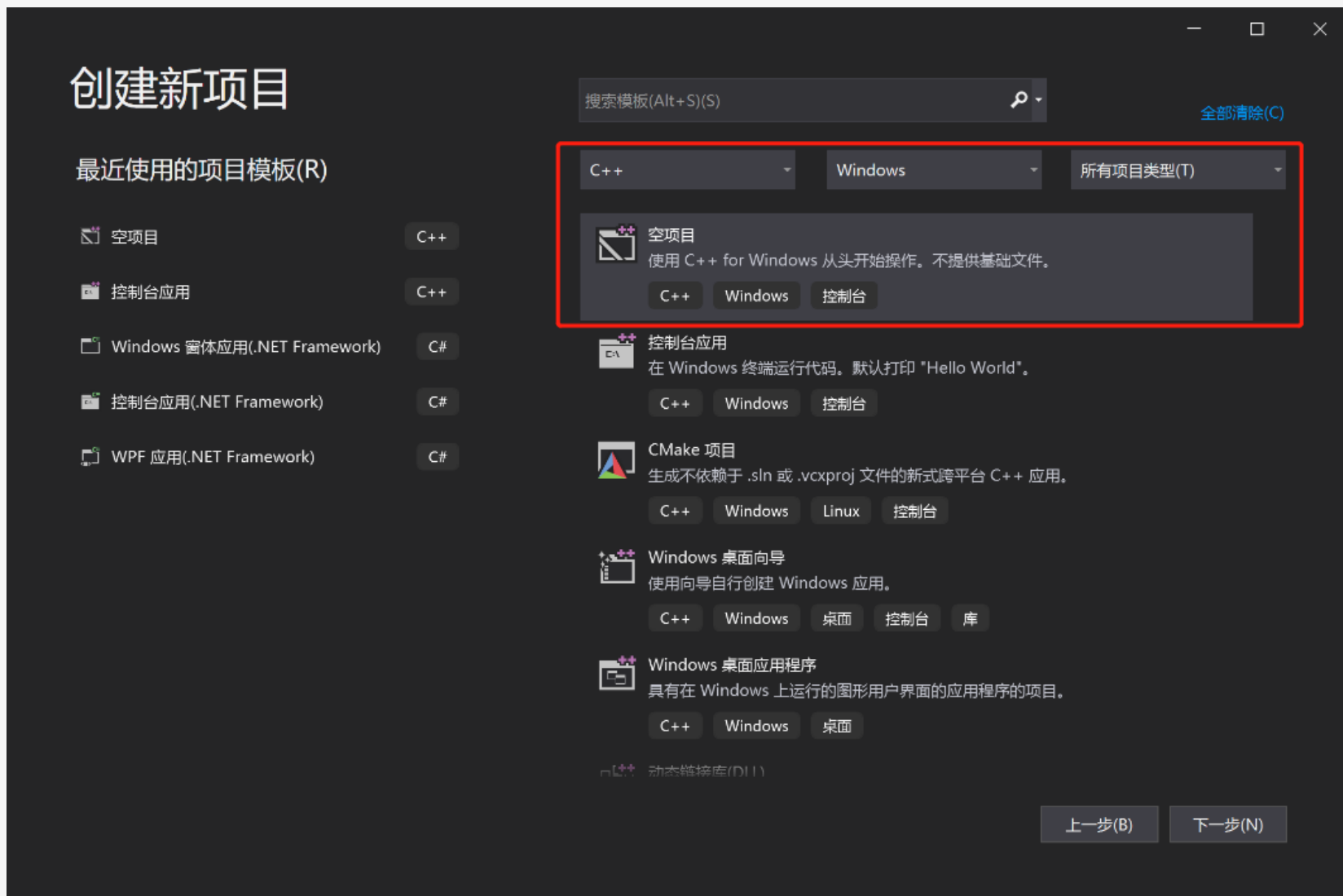


从图中可以看出，二维数组a是按行进行存放的，先存放a行，再存放a[1]行、a[2]行，并且每行有4个元素，也是依次存放的。

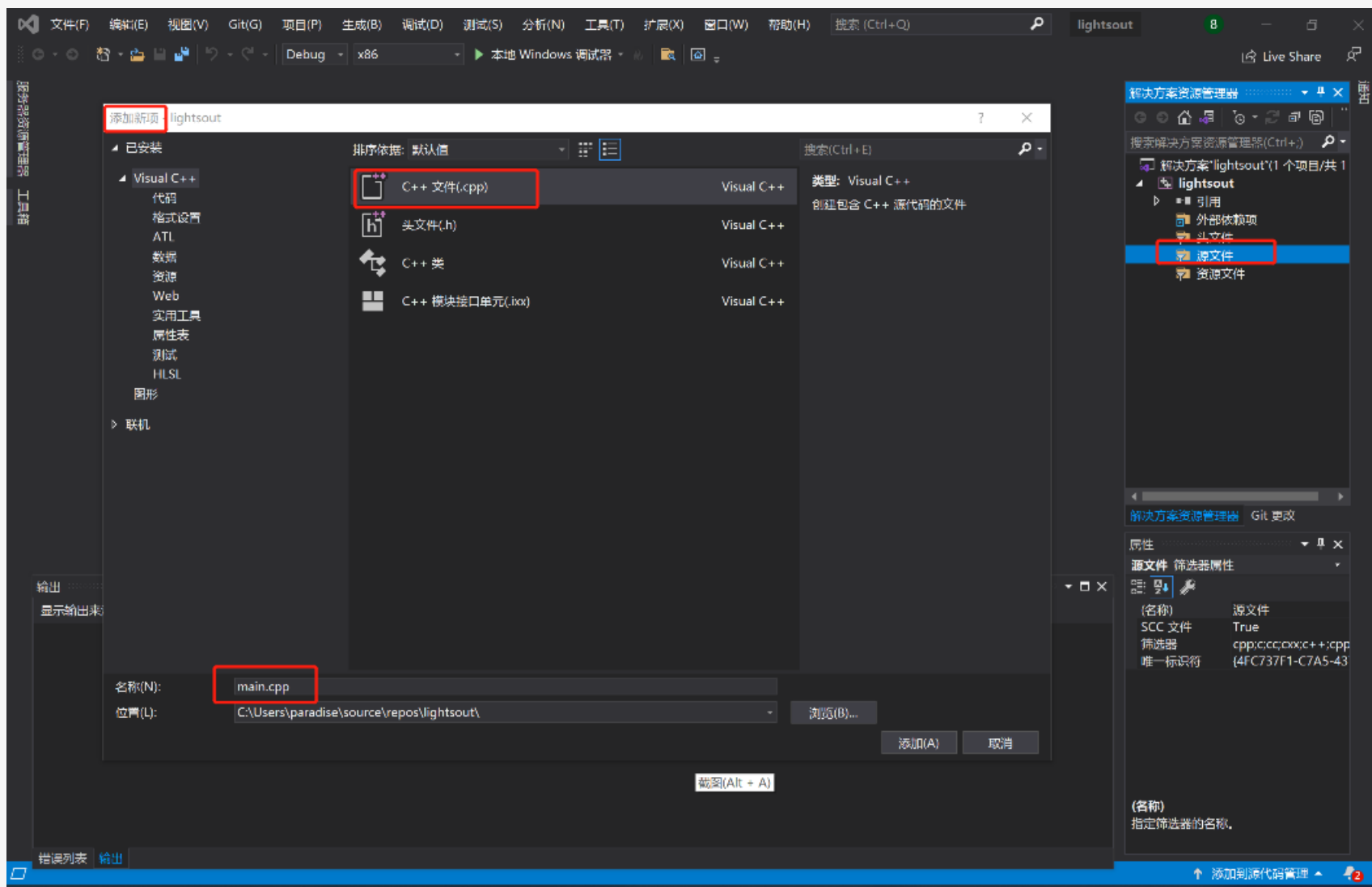
完成二维数组的定义后，需要对二维数组进行初始化，初始化二维数组的方式有4种，具体如下。

- 针对特定的问题，
- 寻找解决方案。

单文件方式



添加源文件



处理输入

- 程序=数据结构+算法

```
//定义数据结构
int N=0; //N组数据
int nResult=0; // 存储解的个数
int orgLights[5][6] ; //存储原始灯的状态, 0代表灭, 1代表亮;
int resultLights[5][6]; // 按下后的灯的状态, 用于检测是否全灭;
int buttonLights[5][6]; //存储是否被按下, 0, 代表没有; 1, 代表被按下; 也是最后的输出结果
```

问题分解

1.输入数据

- ① 根据输入数据，设置灯的状态。

2.枚举每一种按键情况，检查是否全灭

- ① 将数转换为二进制数，再转化为按键矩阵；
- ② 根据按键矩阵，设置灯的状态
- ③ 检查结果是否全灭

3.输出数据

- ① 根据输出格式要求，打印出按键情况

问题分解

1. 输入数据

① 根据输入数据，设置灯的状态。注意初始化变量。

```
138 //一次运行，要输入多组数据，组数由第一个输入数据决定
139 //（在开始的时候输入一个N，接下来是N组数据）。
140
141 cin >> N;
142 for (int m = 0; m < N; m++)
143 {
144     // 一组输入，处理一次，如果有结果就输出结果
145     //每次循环，初始化
146     memset(orgLights, 0, sizeof(orgLights));
147     memset(resultLights, 0, sizeof(resultLights));
148     memset(buttonLights, 0, sizeof(buttonLights));
149
150     for (int i = 0; i < 5; i++)
151     {
152         for (int j = 0; j < 6; j++)
153         {
154             cin >> orgLights[i][j]; //存储初始输入结果
155             resultLights[i][j] = orgLights[i][j]; //每次枚举开始之后，要重置为初始输入
156         }
157     }
```


问题分解

枚举每一种按键情况，检查是否全灭

① 将数转换为二进制数，再转化为按键矩阵；

```
23 // 将二进制数转化为按钮矩阵
24 //
25 int ButtonStatus(const int m, int (&buttonStatus)[5][6])
26 {
27     memset(buttonStatus, 0, sizeof(buttonStatus));
28     //除2取余，逆序排列
29     int index = 0;
30     int temp = m;
31
32     while (temp > 0)
33     {
34         buttonStatus[index / 6][ index % 6] = temp % 2;
35         index++;
36         temp = temp / 2;
37     }
38     return 0;
39 }
```

问题分解

枚举每一种按键情况，检查是否全灭

① 根据按键矩阵，设置灯的状态

```
64 // 根据操作灯的按钮矩阵，改变周边的灯的状态
65 int PressButton2(const int i,const int j, int (&buttonStatus)[5][6],int (&Result)[5][6])
66 {
67     buttonStatus[i][j] = 1;
68     //当i, j位置的按钮按下时，改变周边十字的灯的状态;
69     Result[i][j] = (Result[i][j] + 1) % 2;
70     if ((i - 1) >= 0) //判断是否越界
71         Result[i - 1][j] = (Result[i - 1][j] + 1) % 2;
72     if ((j - 1) >= 0)
73         Result[i][j - 1] = (Result[i][j - 1] + 1) % 2;
74     if ((i + 1) < 5)
75         Result[i + 1][j] = (Result[i + 1][j] + 1) % 2;
76     if ((j + 1) < 6)
77         Result[i][j + 1] = (Result[i][j + 1] + 1) % 2;
78
79     return 0;
80 }
```

问题分解

枚举每一种按键情况，检查是否全灭

① 检查结果是否全灭

```
64 // 检查是否全灭,如果是, 返回1; 否则, 返回0;  
65 int CheckResult(const int (&Result)[5][6])  
66 {  
67     for (int i = 0; i < 5; i++)  
68         for (int j = 0; j < 6; j++)  
69             if (Result[i][j] != 0)  
70                 return 0;  
71  
72     return 1;  
73 }
```

问题分解

输出数据

- ① 根据输出格式要求，打印出按键情况

```
6 //打印二维数组
7 int printResult(const int iGroupNumber,const int (&Result)[5][6])
8 {
9     cout << "PUZZLE #" << iGroupNumber << endl;
10
11     for (int i = 0; i < 5; i++)
12     {
13         for (int j = 0; j < 6; j++)
14         {
15             cout << Result[i][j] << " ";
16         }
17         cout << endl;
18     }
19
20
21     return 0;
22 }
```

暴力枚举

1.枚举每一种按键情况，检查是否全灭

- ① 将数转换为二进制数，再转化为按键矩阵；
- ② 根据按键矩阵，设置灯的状态
- ③ 检查结果是否全灭

```
166 //暴力枚举
167 for (int m = 0; m < pow(2,30); m++)
168 {
169     //每次枚举前，重置灯的初始状态；
170     ResetLights(orgLights, resultLights);
171
172     //将 m 转化为按键矩阵
173     ButtonStatus(m, buttonLights);
174     //根据按键情况，改变灯的状态
175     PressButton(buttonLights, resultLights);
176     //3.Output
177     //检查是否全灭，是，输出结果并退出枚举，处理下一组输入；否，接着枚举
178     if (CheckReult(resultLights))
179     {
180         printResult(nResult, buttonLights);
181         break;
182     }
183 }
```

算法设计（改进）

1. 输入数据

- ① 根据输入数据，设置灯的状态。

2. 处理数据

- ① 枚举第一行的所有按键情况，并执行以下 2-5 步
- ② 通过第二行按键关闭第一行亮的灯
- ③ 通过第三行按键关闭第二行亮的灯
- ④ 依此类推，通过第五行的按键关闭第四行亮的灯
- ⑤ 测试第五行的灯的状态，是否全灭，
 - a) 如果是，则成功；
 - b) 否则，改变第一行的按键，再次循环。


3. 输出数据

- ① 根据输出格式要求，打印出按键情况

枚举优化：部分枚举取代全局枚举

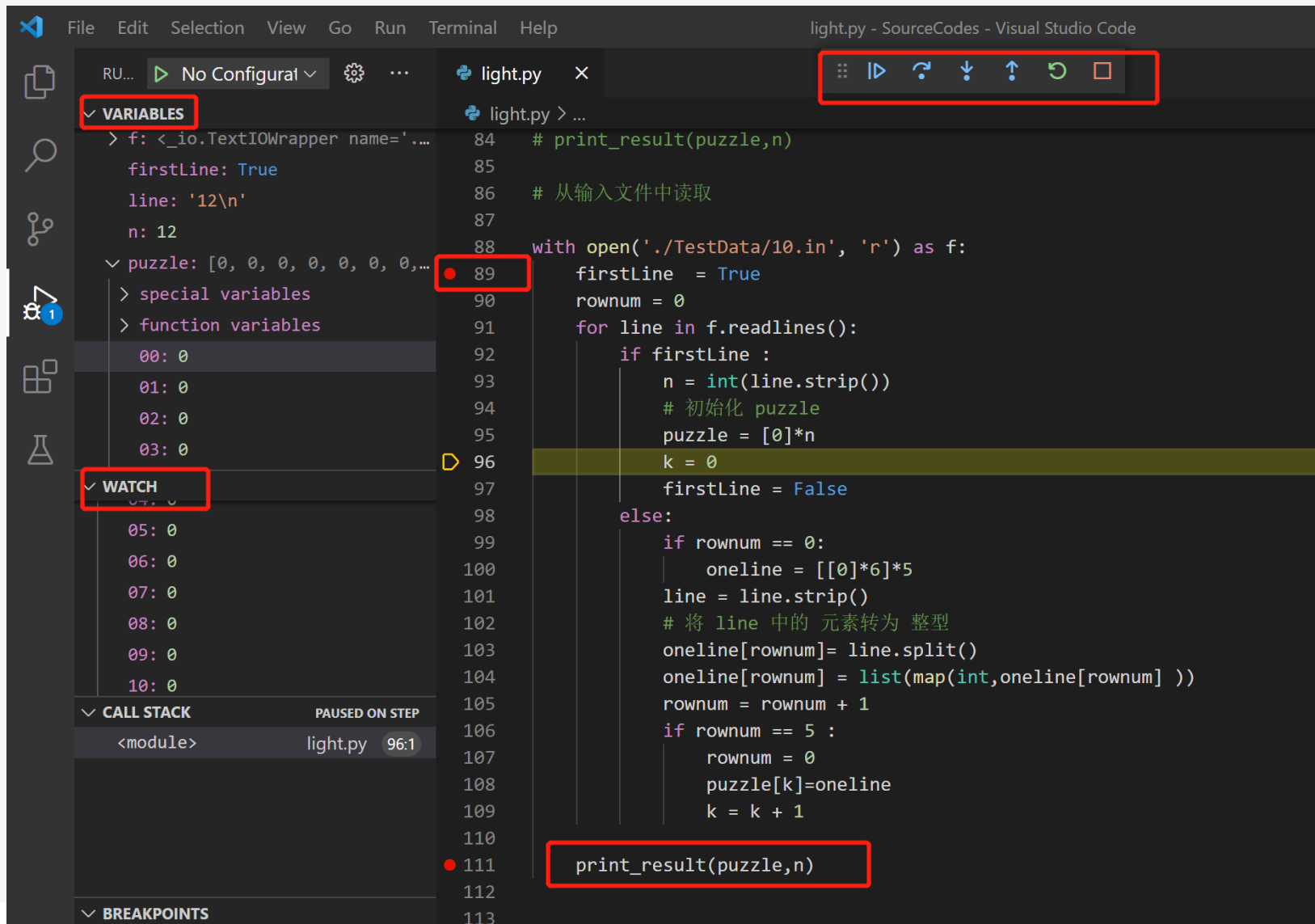
```
186 //优化：枚举局部，第一行状态确定后，后续的都可以确定。
187 for (int m = 0; m < 64; m++)
188 {
189     //每次枚举前，重置灯的初始状态；
190     ResetLights(orgLights, resultLights);
191     //将 m 转化为按键矩阵
192     ButtonStatus(m, buttonLights);
193     //根据按键情况，改变灯的状态
194     PressButton(buttonLights, resultLights);
195     //从第一行开始，通过下一行依次灭掉上一行的灯。
196     for (int i = 0; i < 4; i++)
197         for(int j=0;j<6;j++)
198         {
199             if (resultLights[i][j] == 1)
200             {
201                 PressButton2(i+1, j,buttonLights, resultLights);
202             }
203         }
204     // //3.Output
205     //检查是否全灭，是，输出结果并退出枚举，处理下一组输入；否，接着枚举
```

本地测试通过后，再在OJ平台上提交，测试最终结果

 已接受
时间: 2ms 内存: 3MB 语言: C++ 用户: yhzeng

ID	状态	内存	时间	分数	实际时间	信号
1	已接受	3MB	0ms	10	2ms	0
2	已接受	3MB	2ms	10	5ms	0
3	已接受	3MB	2ms	10	5ms	0
4	已接受	3MB	0ms	10	2ms	0
5	已接受	3MB	1ms	10	4ms	0
6	已接受	3MB	2ms	10	7ms	0
7	已接受	3MB	0ms	10	3ms	0
8	已接受	3MB	2ms	10	7ms	0
9	已接受	3MB	0ms	10	4ms	0
10	已接受	3MB	2ms	10	4ms	0

编码调试（设置断点F9，进入调试模式F5）



- 设置断点 F9
- 进入调试模式 F5
- 添加监视
- 单步执行 F10
- 运行到下一个断点 F5
- 查看本地变量
- 打印变量
- 注释相关代码
-
- 功能模块函数化（分解问题）

实训建议

【时间是最公平和最宝贵的资源】

- Learning By Doing : 在实践中学 【动手实践】
- 主动学习，拒绝依赖 【善于利用网络，自主解决问题】
- 请千万不要用“复制”-“粘贴”把代码从页面粘贴到你自己的电脑上。
- 善于总结、归纳 【利用在线文档】
- 积极参与讨论与演示 【表达】
- 制订长期目标与短期目标
- 学好语文、数学、英语、逻辑、..... 【计算机是一种工具】
- 【偏方】： 请准备几份大厂的面试题或者你心仪的专业考研试卷，放在桌面！