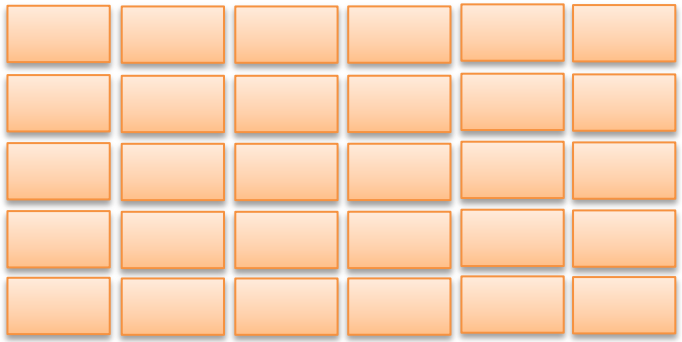
枚举 — 熄灯问题

郭 炜　刘家瑛

北京大学

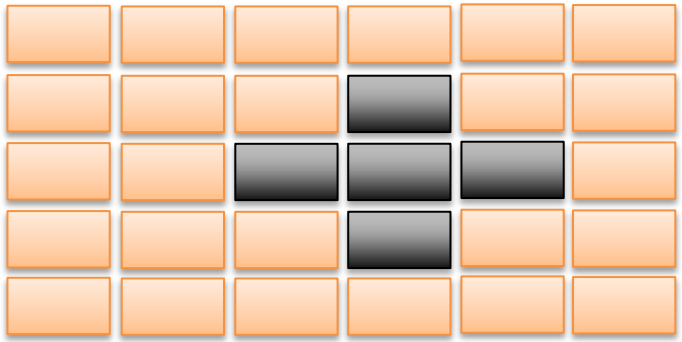
熄灯问题

- 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



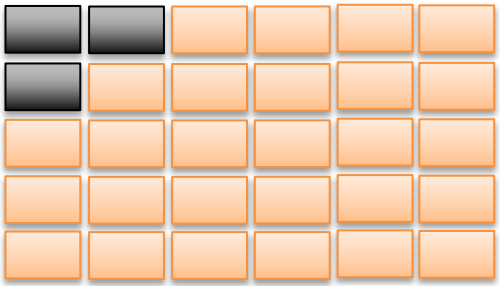
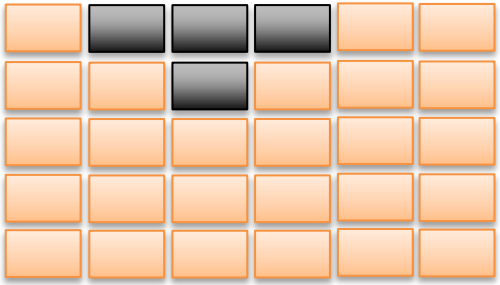
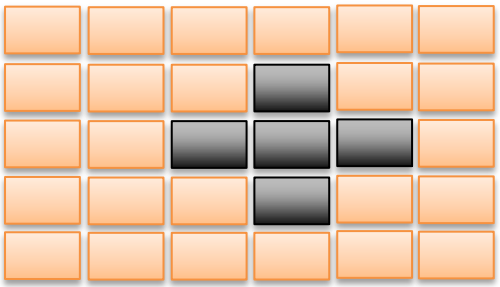
熄灯问题

- 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

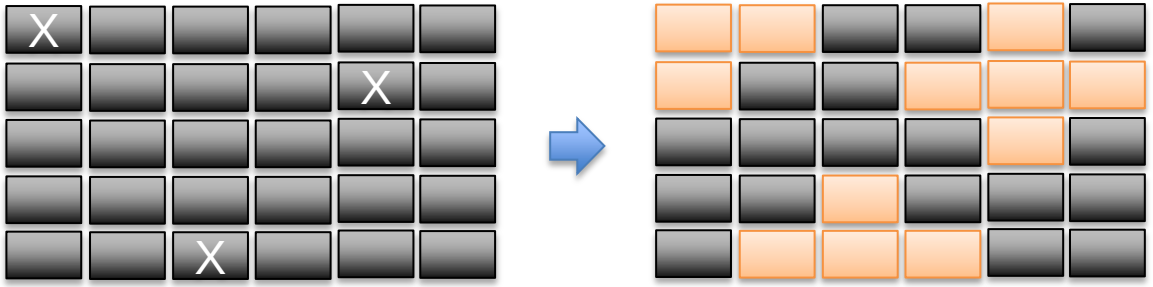


熄灯问题

- 问题描述
  - 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

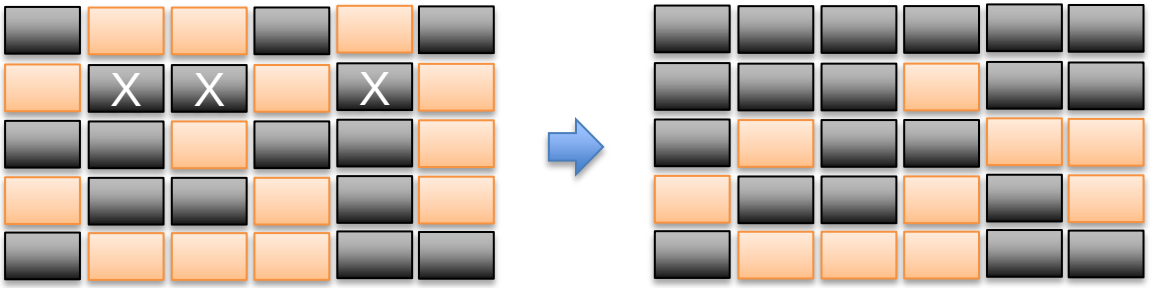




- 在下图中，左边矩阵中用X标记的按钮表示被按下，右边的矩阵表示灯状态的改变



- 与一盏灯毗邻的多个按钮被按下时，一个操作会抵消另一次操作的结果
- 第2行第3，5列的按钮都被按下

→ 第2行第4列的灯的状态就不改变



- 对矩阵中的每盏灯设置一个初始状态
- 请你写一个程序，确定需要按下哪些按钮，恰好使得所有的灯都熄灭

- 输入:
  - 第一行是一个正整数N, 表示需要解决的案例数
  - 每个案例由5行组成, 每一行包括6个数字
  - 这些数字以空格隔开, 可以是0或1
  - **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:
  - 对每个案例, 首先输出一行, 输出字符串 “PUZZLE #m”, 其中m是该案例的序号
  - 接着按照该案例的输入格式输出5行
    - **1** 表示需要把对应的按钮按下
    - **0** 表示不需要按对应的按钮
    - 每个数字以一个空格隔开

样例输入

```
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
```

样例输出

```
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
```

解题分析(1)

- 第2次按下同一个按钮时，将抵消第1次按下时所产生的结果

→每个按钮最多只需要按下一次

- 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

解题分析(2)

- 第一想法: 枚举所有可能的按钮(开关)状态, 对每个状态计算一下最后灯的情况, 看是否都熄灭
  - 每个按钮有两种状态(按下或不按下)
  - 一共有30个开关, 那么状态数是2^{30}, 太多, 会超时
- 如何减少枚举的状态数目呢?

  基本思路: 如果存在某个局部, 一旦这个局部的状态被确定, 那么剩余其他部分的状态只能是确定的一种, 或者不多的n种, 那么就只需枚举这个局部的状态即可

解题分析(3)

- 本题是否存在这样的 “局部” 呢?
- 经过观察, 发现第1行就是这样的一个 “局部”
  - 因为第1行的各开关状态确定的情况下, 这些开关作用过后, 将导致第1行某些灯是亮的, 某些灯是灭的

  →要熄灭第1行某个亮着的灯(假设位于第i列), 那么唯一的办法就是按下第2行第i列的开关

  (因为第1行的开关已经用过了, 而第3行及其后的开关不会影响到第1行)
  - 为了使第1行的灯全部熄灭, 第2行的合理开关状态就是唯一的

解题分析(4)

- 第2行的开关起作用后，

→ 为了熄灭第2行的灯，第3行的合理开关状态就也是唯一的

→ 以此类推，最后一行的开关状态也是唯一的

- 只要第1行的状态定下来，记作A，那么剩余行的情况就是确定唯一的了

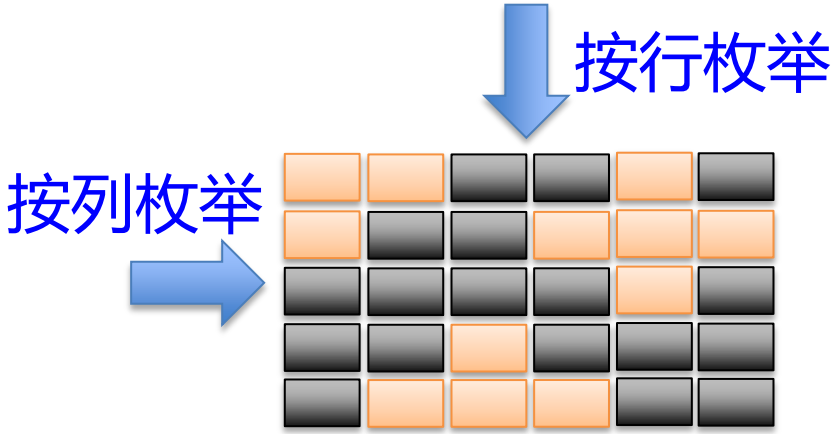
→ 推算出最后一行的开关状态，然后看看最后一行的开关起作用后，最后一行的所有灯是否都熄灭：

  - 如果是，那么A就是一个解的状态
  - 如果不是，那么A不是解的状态，第1行换个状态重新试试

- 只需枚举第1行的状态，状态数是$2^6 = 64$

有没有状态数更少的做法？

- 枚举第一列, 状态数是$2^5 = 32$
- 执行次数: $64 \times 5 \times 6$ vs. $32 \times 6 \times 5$



具体实现

- 用一个矩阵 puzzle[5][6] 表示灯的初始状态
  - puzzle[i][j]=1: 灯(i, j)初始时是被点亮的
  - puzzle [i][j]=0: 灯(i, j)初始时是熄灭的
- 用一个矩阵 press[5][6] 表示要计算的结果
  - press[i][j]=1: 需要按下按钮(i, j)
  - press[i][j]=0: 不需要按下按钮(i, j)

- press[0]里放着第1行开关的状态, 如何进行枚举呢?
- 可以使用六重循环:

```
for( int a0 = 0; a0 < 2; a0++ )
    for( int a1 = 0; a1 < 2; a1++ )
        for( int a2 = 0; a2 < 2; a2++ )
            for( int a3 = 0; a3 < 2; a3++ )
                for( int a4 = 0; a4 < 2; a4++ )
                    for( int a5 = 0; a5 < 2; a5++ ) {
                        press[0][0] = a0;
                        press[0][1] = a1;
                        press[0][2] = a2;   ……
                    }
```

实现方案

- 根据上面的分析，按钮矩阵的第1行元素的各种取值进行枚举，依次考虑如下情况：

0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	0	1	1
0	0	0	1	0	0
⋮					
1	1	1	1	1	1

- 枚举方式：
  - 将按钮矩阵的第1行看作一个二进制数
  - 通过实现++操作实现

实现方案

- 用一个6×8的数组来表示按钮矩阵:
  简化计算数组下一行的值的计算公式
  - 第0行, 第0列和第7列不属于PRESS矩阵范围, 可全置0

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)

- 给定PRESS的第1行取值, 计算出PRESS的其他行的值
  - press[2][c]=(puzzle[1][c]+press[1][c-1]
    +press[1][c]+press[1][c+1] + press[0][c]) %2
  - 0<r<5, 0<c<7

程序实现

```
#include <stdio.h>
int puzzle[6][8], press[6][8];
bool guess(){
    int c, r;
    for(r=1; r<5; r++)
        for(c=1; c<7; c++)
            press[r+1][c] = (puzzle[r][c]+press[r][c]+
            press[r-1][c]+ press[r][c-1]+press[r][c+1]) %2;
    for(c=1; c<7; c++)
        if ((press[5][c-1] + press[5][c] + press[5][c+1] +
            press[4][c]) %2 != puzzle[5][c] )
            return(false);
    return(true);
}
```

根据press第1行和puzzle数组, 计算press其他行的值

判断所计算的press数组能否熄灭第5行的所有灯

```
void enumerate (){
        int c;
        bool success;
        for ( c=1; c<7; c++)
            press[1][c] = 0;
        while(guess()==false){
            press[1][1]++;
            c = 1;
            while (press[1][c] > 1) {
                press[1][c] = 0;
                c++;
                press[1][c]++;
            }
        }
        return;
}
```

对press第1行的元素press[1][1]~press[1][6]的各种取值情况进行枚举，依次考虑：

0	0	0	0	0	0
1	0	0	0	0	0
0	1	0	0	0	0
1	1	0	0	0	0
0	0	1	0	0	0
⋮					
1	1	1	1	1	1

主程序

```
int main() {
    int cases, i, r, c;
    scanf("%d", &cases);
    for ( r=0; r<6; r++ )
        press[r][0] = press[r][7] = 0;
    for ( c=1; r<7; r++ )
        press[0][c] = 0;
    for (i=0; i<cases; i++){
        for(r=1; r<6; r++)
            for(c=1; c<7; c++)
                scanf("%d", &puzzle[r][c]);
    //读入输入数据
```

```
        enumerate ();
        printf("PUZZLE #%d\n",  i+1);
        for(r=1; r<6; r++){
            for(c=1; c<7; c++)
                printf("%d ", press[r][c]);
            printf("\n");
         }
    }
    return 0;
}
```

总结

- 枚举过程 enumerate( )
  - press[1][]中每一个元素表示一个二进制数0/1, 通过模拟二进制加法方式实现枚举
  - 需要处理进位
- 推测验证过程 guess( )
  - 用6×8按钮矩阵来简化下一行按钮值的计算公式
  - 根据press[1][]和Puzzle数组, 用公式来计算使得1-4行所有灯熄灭的press其他行的值, 再判断所计算的press数组能否熄灭矩阵第5行的所有灯