ACM算法与程序设计

第七讲

枚举与递归

什么是枚举

•枚举是基于已经有的知识进行答案猜测的一种问题求解策略。一般是根据建立的数学模型中的一组变量及其条件，在条件允许的范围内对变量依次取值，判断所取的值是否满足数学模型中的条件，直到找到(全部)符合条件的值为止。

使用时注意下列三方面的问题：

- 建立简洁的数学模型。数学模型中变量的数量要尽量少，它们之间相互独立。
- 减小搜索的空间。利用已经有的知识，缩小数学模型中各个变量的取值范围，防止不必要的计算。
- 采用合适的搜索顺序。对搜索空间的遍历顺序要与数学模型中的条件体现式一致。

称硬币

1、链接地址

2、问题描述

赛利有12枚银币。其中有11枚真币和1枚假币。假币看起来和真币没有区别，但是重量不同。但赛利不懂得假币比真币轻还是重。于是他向朋友借了一架天平。朋友希望赛利称三次就能找出假币而且拟定假币是轻是重。

例如：假如赛利用天平称两枚硬币，发觉天平平衡，阐明两枚都是真的。假如赛利用一枚真币与另一枚银币比较，发觉它比真币轻或重，阐明它是假币。经过精心安排每次的称量，赛利确保在称三次后拟定假币。

问题描述

输入格式

第一行有一种数字n，表达有n组测试用例。

对于每组测试用例：输入有三行，每行表达一次称量的成果。赛利事先将银币标号为A-L。每次称量的成果用三个以空格隔开的字符串表达：天平左边放置的硬币 天平右边放置的硬币 平衡状态。其中平衡状态用”up”, “down”, 或”even”表达, 分别为右端高、右端低和平衡。天平左右的硬币数总是相等的。

输出要求

输出哪一种标号的银币是假币，并阐明它比真币轻还是重(heavy or light)。

问题描述

输入样例

1

ABCD EFGH even

ABCI EFJK up

ABIJ EFGH even

输出样例

K is the counterfeit coin and it is light.

3、解题思绪

■ 此题中赛利已经设计了正确的称量方案，确保从三组称量数据中能得到唯一的答案。答案能够用两个变量表达：x-假币的标号、w-假币是比真币轻还是比真币重。x共有12种猜测；w有2种猜测。根据赛利设计的称量方案，(x，w)的24种猜测中，只有唯一的猜测与三组称量数据都不矛盾。所以，假如猜测(x，w)满足下列条件，这个猜测就是要找的答案：

- 在称量成果为"even" 的天平两边，没有出现x ；
- 假如w表达假币比真币轻，则在称量成果为"up" 的天平右边一定出现x、在称量成果为"down" 的天平左边一定出现x；
- 假如w表达假币比真币重，则在称量成果为"up" 的天平左边一定出现x、在称量成果为"down" 的天平右边一定出现x。

3、解题思绪

详细实现时，要注意两点：

(1) 选择合适的算法

对于每一枚硬币x 逐一试探：

- 假设 x 比真币轻的猜测是否成立？猜测成立则进行输出。
- 假设 x 比真币重的猜测是否成立？猜测成立则进行输出。

(2) 选择合适的数据构造

以字符串数组存储称量的成果。每次称量时，天平左右最多有6枚硬币。所以，字符串的长度需要为7，最终一位存储字符串的结束符'\0'，便于程序代码中使用字符串操作函数。

```
char left[3][7], right[3][7], result[3][7];
```

4、参照程序

```
#include <stdio.h>
#include <string.h>
char left[3][7], right[3][7], result[3][5];
bool isHeavy(char x);
bool isLight(char x);
int main(void)
{
    int i,n;
    char c;
    scanf("%d", &n);
    while ( n > 0 )
    {
        for ( i = 0; i < 3; i++ )
            scanf("%s %s %s", left[i], right[i], result[i]);
```

4、参照程序

```
        for ( c = 'A'; c <= 'L'; c++ )
        {
            if ( isLight(c) )
            {
                printf("%c is the counterfeit coin and it is light.\n", c);
                break;
            }
            if ( isHeavy(c) )
            {
                printf("%c is the counterfeit coin and it is heavy.\n", c);
                break;
            }
        }
        n--;
    }
    return 0;
}
```

4、参照程序

```
bool isLight( char x ) // 判断硬币x 是否为轻的代码
{
   int i;
   for ( i = 0; i < 3; i++ ) // 判断是否与三次称量成果矛盾
      switch( result[i][0] )
      {
         case 'u': if( strchr(right[i], x) == NULL )
                      return false;
                   break;
         case 'e': if(strchr(right[i], x) != NULL
                                        || strchr(left[i], x) != NULL)
                      return false;
                   break;
         case 'd': if(strchr(left[i], x) == NULL)
                      return false;
                   break;
      }
   return true;
}
```

4、参照程序

```
bool isHeavy( char x ) //判断硬币x 是否为重的代码
{
   int i;
   for ( i = 0; i < 3; i++ ) // 判断是否与三次称量成果矛盾
      switch( result[i][0] )
      {
         case 'u': if( strchr(left[i], x) == NULL)
                      return false;
                   break;
         case 'e': if(strchr(right[i], x) != NULL
                                          || strchr(left[i], x) != NULL)
                      return false;
                   break;
         case 'd': if(strchr(right[i], x) == NULL)
                      return false;
                   break;
      }
   return true;
}
```

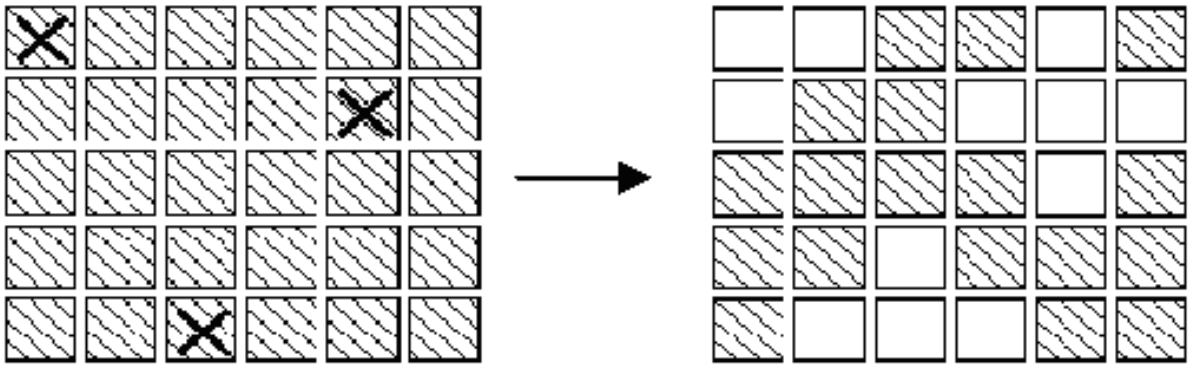
熄灯问题

1、链接地址

2、问题描述

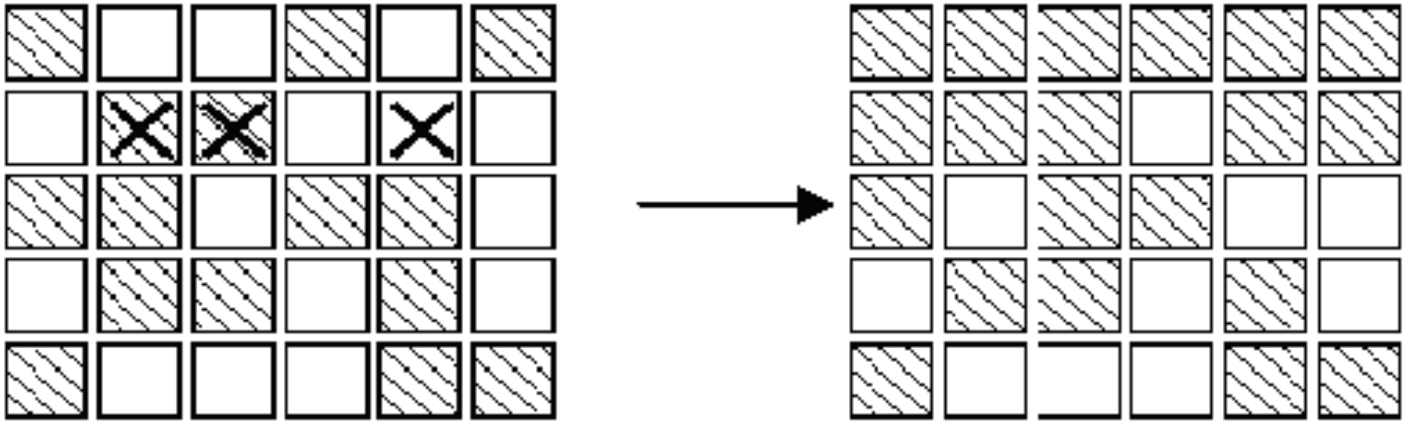
一种由按钮构成的矩阵，其中每行有6个按钮，共5行。每个按钮的位置上有一盏灯。当按下一种按钮后，该按钮以及周围位置(上边、下边、左边、右边)的灯都会变化一次。即，假如灯原来是点亮的，就会被熄灭；假如灯原来是熄灭的，则会被点亮。在矩阵角上的按钮变化3盏灯的状态；在矩阵边上的按钮变化4盏灯的状态；其他的按钮变化5盏灯的状态。

问题描述



在上图中，左边矩阵中用X标识的按钮表达被按下，右边的矩阵表达灯状态的变化。对矩阵中的每盏灯设置一种初始状态。请你按按钮，直至每一盏灯都熄灭。与一盏灯毗邻的多种按钮被按下时，一种操作会抵消另一次操作的成果。在下图中，第2行第3、5列的按钮都被按下，所以第2行、第4列的灯的状态就不变化。

问题描述



请你写一种程序，拟定需要按下哪些按钮，恰好使得全部的灯都熄灭。根据上面的规则，我们懂得：

(1)第2次按下同一种按钮时，将抵消第1次按下时所产生的成果。所以，每个按钮最多只需要按下一次；

(2)各个按钮被按下的顺序对最终的成果没有影响；

(3)对第1行中每盏点亮的灯，按下第2行相应的按钮，就能够熄灭第1行的全部灯。如此反复下去，能够熄灭第1、2、3、4行的全部灯。一样，按下第1、2、3、4、5列的按钮，能够熄灭前5列的灯。

问题描述

输入数据

第一行是一种正整数N，表达需要处理的案例数。每个案例由5行构成，每一行涉及6个数字。这些数字以空格隔开，能够是0或1。0表达灯的初始状态是熄灭的，1表达灯的初始状态是点亮的。

输出要求

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，其中的1表达需要把相应的按钮按下，0则表达不需要按相应的按钮。每个数字以一种空格隔开。

问题描述

输入样例

```
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

```
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
```

3、解题思绪

- 为了论述以便，按下图所示，为按钮矩阵中的每个位置分别指定一种坐标。用数组元素puzzle[i][j]表达位置(i, j)上灯的初始状态：1 表达灯是被点亮的；0 表达灯是熄灭的。用数组元素press[i][j]表达为了让全部的灯都熄灭，是否要按下位置(i, j)上的按钮：1 表达要按下；0 表达不用按下。

- 因为第0 行、第0 列和第7 列不属于按钮矩阵的范围，没有按钮，能够假设这些位置上的灯总是熄灭的、按钮也不用按下。其他30 个位置上的按钮是否需要按下是未知的。

- 所以数组press 共有2^30种取值。从这么大的一种空间中直接搜索我们要找的答案，显然代价太大、不合适。要从熄灯的规则中，发觉答案中的元素值之间的规律。不满足这个规律的数组press，就没有必要进行判断了。

3、解题思绪

(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)

- 根据熄灯规则，假如矩阵press 是寻找的答案，那么按照press 的第一行对矩阵中的按钮操作之后，此时在矩阵的第一行上：
- 假如位置(1, j)上的灯是点亮的，则要按下位置(2, j)上按钮，即press[2][j]一定取1；
- 假如位置(1, j)上的灯是熄灭的，则不能按位置(2, j)上按钮，即press[2][j]一定取0。
- 这么根据press 的第一、二行操作矩阵中的按钮，才干确保第一行的灯全部熄灭。而对矩阵中第三、四、五行的按钮不论进行什么样的操作，都不影响第一行各灯的状态。依此类推，能够拟定press 第三、四、五行的值。
- 所以，一旦拟定了press 第一行的值之后，为熄灭矩阵中第一至四行的灯，其他行的值也就随之拟定了。press 的第一行共有2^6 种取值，分别相应唯一的一种press 取值，使得矩阵中前四行的灯都能熄灭。只要对这2^6 种情况进行判断就能够了：假如按照其中的某个press对矩阵中的按钮进行操作后，第五行的全部灯也恰好熄灭，则找到了答案。

处理方案

(1)对press 第一行的元素press[1][1]~ press [1][6]的多种取值情况进行枚举，依次考虑如下情况：

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

(2) 对press 第一行每一种取值，根据熄灯规则计算出press的其他行的值。判断这个press 能否使得矩阵第五行的全部灯也恰好熄灭。

```
int main(void)
{
    int cases,i,r,c;
    scanf("%d", &cases);
    for (r=0;r<6;r++)
        press[r][0]=press[r][7]=0;
    for (c=1;c<7;c++)
        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]);
        enumate();
        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;
}
```

4、参照程序

```
void enumate(void)
{
    int c;
    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]++;
        }
    }
}
```

4、参照程序

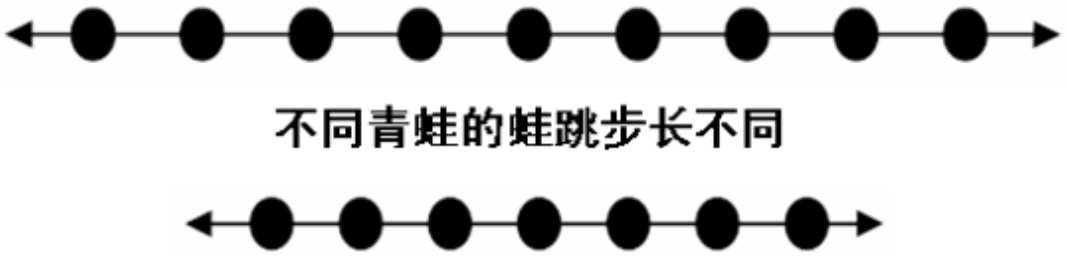
```
#include <stdio.h>
int puzzle[6][8],press[6][8];
bool guess(void)
{
    int c,r;
    for (r=1;r<5;r++ )
        for (c=1;c<7;c++)
            press[r+1][c]=(puzzle[r][c]+press[r][c]
                           +press[r-1][c]+press[r][c-1]
                           +press[r][c+1])%2;
    for(c=1;c<7;c++) //判断最终一行是否熄灭
        if ((press[5][c-1]+press[5][c]+press[5][c+1]
                          +press[4][c])%2!=puzzle[5][c])
            return(false);
    return true;
}
```

讨厌的青蛙

1、链接地址

2、问题描述

在韩国，有一种小的青蛙。每到晚上，这种青蛙会跳越稻田，从而踩踏稻子。农民在早上看到被踩踏的稻子，希望找到造成最大损害的那只青蛙经过的途径。每只青蛙总是沿着一条直线跳越稻田，而且每次跳跃的距离都相同，如图1所示。 稻田里的稻子构成一种栅格，每棵稻子位于一种格点上，如图2所示。而青蛙总是从稻田的一侧跳进稻田，然后沿着某条直线穿越稻田，从另一侧跳出去，如图3所示。



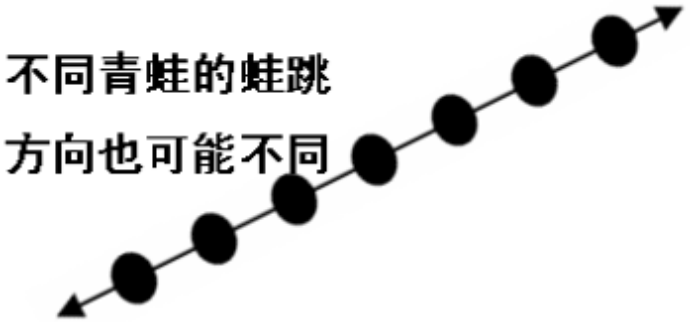


图 1 青蛙踩踏水稻示意图

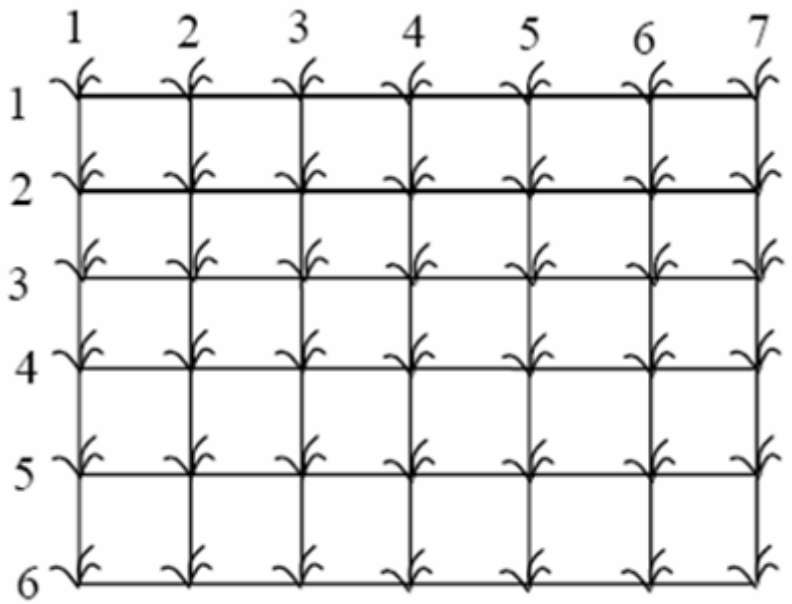


图 2 稻田栅格示意图

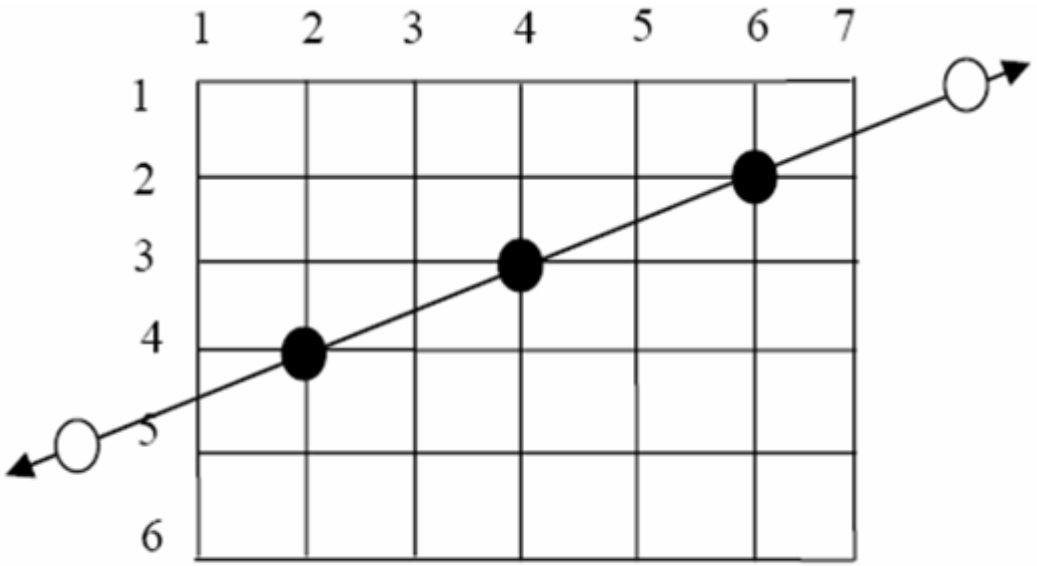


图 3 青蛙穿越稻田示意图

问题描述

青蛙的每一跳都恰好踩在一棵水稻上，将这棵水稻拍倒。可能会有多只青蛙从稻田穿越，有些水稻被多只青蛙踩踏，如图4所示。当然，农民所见到的是图5中的情形，看不到图4中的直线。

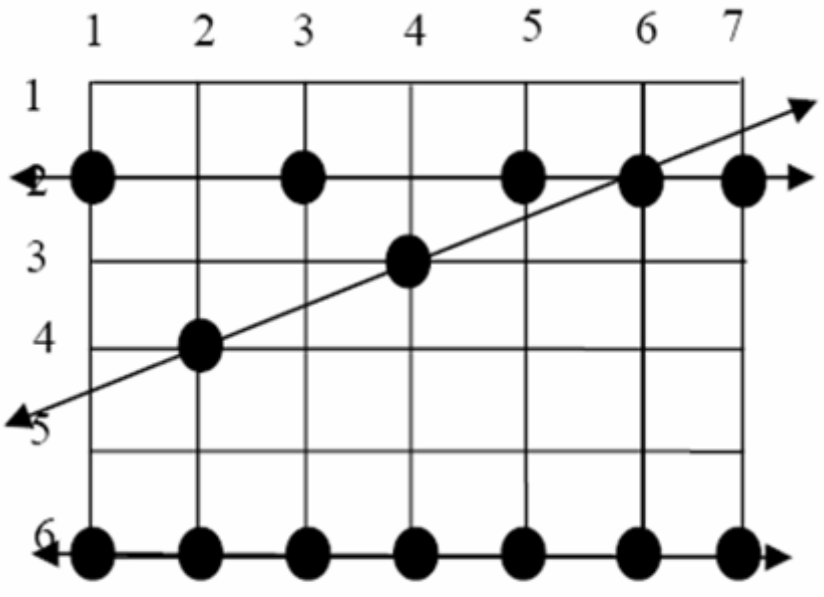


图 4 水稻被多只青蛙踩踏示意图

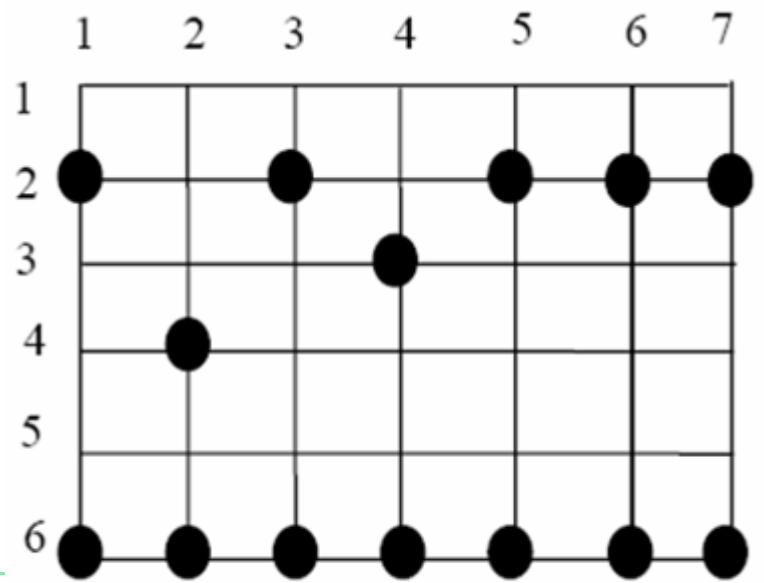


图 5 农民见到的稻田示意图