程序设计实习

算法基础

张勤健
zqj@pku.edu.cn

北京大学信息科学技术学院

2024 年 5 月 8 日

枚举

- 基于逐个尝试答案的一种问题求解策略

枚举

- 基于逐个尝试答案的一种问题求解策略
- 例如: 求小于 N 的最大素数
  - 找不到一个数学公式, 使得根据 N 就可以计算出这个素数
  - N-1 是素数吗? N-2 是素数吗? ……

例题 1: 数字方格

- 描述

如上图，有 3 个方格，每个方格里面都有一个整数 a1，a2，a3。已知 0 <= a1, a2, a3 <= n，而且 a1 + a2 是 2 的倍数，a2 + a3 是 3 的倍数，a1 + a2 + a3 是 5 的倍数。你的任务是找到一组 a1，a2，a3，使得 a1 + a2 + a3 最大。

- 输入
一行，包含一个整数 n (0 <= n <= 100)。
- 输出
一个整数，即 a1 + a2 + a3 的最大值。
- 样例输入
3
- 样例输出
5

例题 1: 数字方格

```
1  #include <stdio.h>
2
3  int main() {
4    int n, a1, a2, a3, ans = 0;
5    scanf("%d", &n);
6    for (a1 = 0; a1 <= n; a1++) {
7      for (a2 = 0; a2 <= n; a2++) {
8        for (a3 = 0; a3 <= n; a3++) {
9          if ((a1 + a2) % 2 == 0
10           && (a2 + a3) % 3 == 0
11           && (a1 + a2 + a3) % 5 == 0) {
12             if (ans < a1 + a2 + a3)  ans = a1 + a2 + a3;
13         }
14       }
15     }
16   }
17   printf("%d", ans);
18   return 0;
19 }
```

例题 2: 完美立方

- 描述
  形如 $a^3 = b^3 + c^3 + d^3$ 的等式被称为完美立方等式。例如 $12^3 = 6^3 + 8^3 + 10^3$ 。编写一个程序，对任给的正整数 N ($N \leq 100$)，寻找所有的四元组 (a, b, c, d)，使得 $a^3 = b^3 + c^3 + d^3$，其中 a, b, c, d 大于 1，小于等于 N，且 $b \leq c \leq d$。
- 输入
  一个正整数 N ($N \leq 100$)。
- 输出
  每行输出一个完美立方。输出格式为：
  Cube = a, Triple = (b,c,d)
  其中 a,b,c,d 所在位置分别用实际求出四元组值代入。
  请按照 a 的值，从小到大依次输出。当两个完美立方等式中 a 的值相同，则 b 值小的优先输出、仍相同则 c 值小的优先输出、再相同则 d 值小的先输出。

例题 2: 完美立方

- 样例输入
  24
- 样例输出
  Cube = 6, Triple = (3,4,5)
  Cube = 12, Triple = (6,8,10)
  Cube = 18, Triple = (2,12,16)
  Cube = 18, Triple = (9,12,15)
  Cube = 19, Triple = (3,10,18)
  Cube = 20, Triple = (7,14,17)
  Cube = 24, Triple = (12,16,20)

例题 2: 完美立方

解题思路:
四重循环枚举 a,b,c,d

- a 在最外层，从小到大枚举，枚举范围 [2,N]
- b 范围 [2,a-1]
- c 范围 [b,a-1]
- d 范围 [c,a-1]

例题 2: 完美立方

```
1  #include <stdio.h>
2  int main() {
3    int N, a, b, c, d;
4    scanf("%d", &N);
5    for (a = 2; a <= N; a++)
6      for (b = 2; b < a; b++)
7        for (c = b; c < a; c++)
8          for (d = c; d < a; d++) {
9            if (a * a * a == b * b * b + c * c * c + d * d * d)
10             printf("Cube = %d, Triple = (%d,%d,%d)\n", a, b, c, d);
11         }
12   return 0;
13 }
```

例题 2: 完美立方-改进

```
1  #include <stdio.h>
2  int main() {
3    int i, N, a, b, c, d;
4    int cube[110] = {0};
5    scanf("%d",&N);
6    for (i = 1; i <= N; i++) {
7      cube[i] = i * i * i;
8    }
9    for(a = 2; a <= N; a++)
10     for(b = 2; b < a; b++)
11       for(c = b; c < a; c++)
12         for(d = c; d < a; d++) {
13           if (cube[a] == cube[b] + cube[c] + cube[d])
14             printf("Cube = %d, Triple = (%d,%d,%d)\n", a, b, c, d);
15         }
16   return 0;
17 }
```

例题 3: 生理周期

- 描述
  人生来就有三个生理周期，分别为体力、感情和智力周期，它们的周期长度为 23 天、28 天和 33 天。每一个周期中有一天是高峰。在高峰这天，人会在相应的方面表现出色。例如，智力周期的高峰，人会思维敏捷，精力容易高度集中。因为三个周期的周长不同，所以通常三个周期的高峰不会落在同一天。对于每个人，我们想知道何时三个高峰落在同一天。对于每个周期，我们会给出从当前年份的第一天开始，到出现高峰的天数（不一定是第一次高峰出现的时间）。你的任务是给定一个从当年第一天开始数的天数，输出从给定时间开始（不包括给定时间）下一次三个高峰落在同一天的时间（距给定时间的天数）。例如：给定时间为 10，下次出现三个高峰同天的时间是 12，则输出 2（注意这里不是 3）。

例题 3: 生理周期

- 输入
  一行，包含四个整数：p, e, i 和 d，相邻两个整数之间用单个空格隔开。p, e, i 分别表示体力、情感和智力高峰出现的时间（时间从当年的第一天开始计算）。d 是给定的时间，可能小于 p, e, 或 i。所有给定时间是非负的并且小于等于 365, 所求的时间小于等于 21252。
- 输出
  一个整数，即从给定时间起，下一次三个高峰同天的时间（距离给定时间的天数）。
- 样例输入
  4 5 6 7
- 样例输出
  16994

例题 3: 生理周期

解题思路:

- 从 d+1 天开始，一直试到第 21252 天，对其中每个日期 k, 看是否满足

```
(k - p) % 23 == 0
&& (k - e) % 28 == 0
&&(k - i) % 33 == 0
```

例题 3: 生理周期

解题思路:

- 从 d+1 天开始，一直试到第 21252 天，对其中每个日期 k, 看是否满足

```
(k - p) % 23 == 0
&& (k - e) % 28 == 0
&&(k - i) % 33 == 0
```

- 何试得更快?

例题 3: 生理周期

解题思路:

- 从 d+1 天开始，一直试到第 21252 天，对其中每个日期 k, 看是否满足

```
(k - p) % 23 == 0
&& (k - e) % 28 == 0
&&(k - i) % 33 == 0
```

- 何试得更快? 跳着试!

例题 3: 生理周期

```c
#include <stdio.h>
int main(){
  int p, e, i, d;
  scanf("%d%d%d%d", &p, &e, &i, &d);
  int k;
  for(k = d + 1; (k - p) % 23 != 0; k++) {
    ;
  }
  for(; (k - e) % 28 != 0; k = k + 23) {
    ;
  }
  for(; (k - i) % 33 != 0; k = k + 23 * 28) {
    ;
  }
  printf("%d", k - d);
  return 0;
}
```

例题 4: 称硬币

- 描述
  有 12 枚硬币。其中有 11 枚真币和 1 枚假币。假币和真币重量不同，但不知道假币比真币轻还是重。现在，用一架天平称了这些币三次，告诉你称的结果，请你找出假币并且确定假币是轻是重（数据保证一定能找出来）。

例题 4: 称硬币

- 输入
  第一行是测试数据组数。每组数据有三行，每行表示一次称量的结果。银币标号为 A-L。每次称量的结果用三个以空格隔开的字符串表示：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” 表示, 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
  输出哪一个标号的银币是假币，并说明它比真币轻还是重。
- 样例输入
  1
  ABCD EFGH even
  ABCI EFJK up
  ABIJ EFGH even
- 样例输出
  K is the counterfeit coin and it is light.

例题 4: 称硬币

解题思路:
对于每一枚硬币先假设它是轻的，看这样是否符合称量结果。如果符合，问题即解决。如果不符合，就假设它是重的，看是否符合称量结果。把所有硬币都试一遍，一定能找到特殊硬币

例题 4: 称硬币

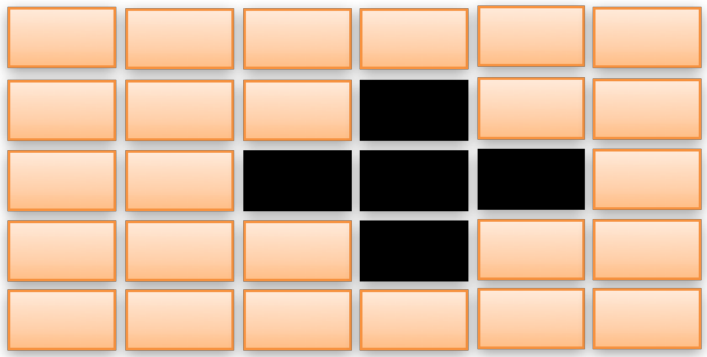
```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;
vector<string> sleft(3), sright(3), sresult(3);
bool isTrue(char c, bool light) ;//light 为 true 表示假设假币为轻，否则表示假设假币
int main() {
  int t;
  cin >> t;
  while(t--) {
  for(int i = 0;i < 3; ++i) cin >> sleft[i] >> sright[i] >> sresult[i];
    for(char c='A'; c <= 'L'; c++) {
      if (isTrue(c, true)) {
        cout << c << " is the counterfeit coin and it is light." << endl;
        break;
      } else if (isTrue(c, false)) {
        cout << c << " is the counterfeit coin and it is heavy." << endl;
        break;
      }
    }
  }
  return 0;
}

```

例题 4: 称硬币

```
24  //light 为 true 表示假设假币为轻，否则表示假设假币为重
25  bool isTrue(char c, bool light) {
26    for(int i = 0;i < 3; ++i) {
27      string pLeft, pRight; //存放天平两边的字符串
28      if (light) {
29        pLeft = sleft[i];
30        pRight = sright[i];
31      } else {
32        pLeft = sright[i];
33        pRight = sleft[i];
34      }
35      if (sresult[i] == "up") {
36        if (pRight.find(c) == pRight.npos)  return false;
37      } else if (sresult[i] == "even") {
38        if (pLeft.find(c) != pLeft.npos || pRight.find(c) != pRight.npos)
39          return false;
40      } else if(sresult[i]== "down") {
41        if (pLeft.find(c) == pRight.npos) return false;
42      }
43    }
44    return true;
45  }
46
```

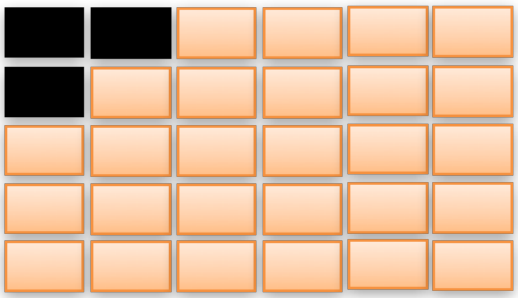
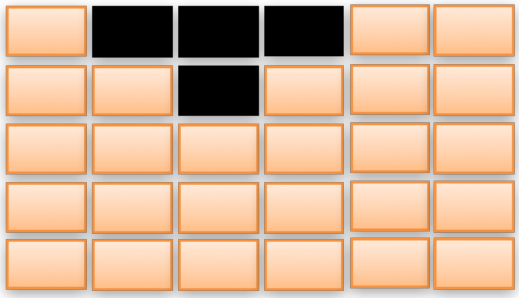
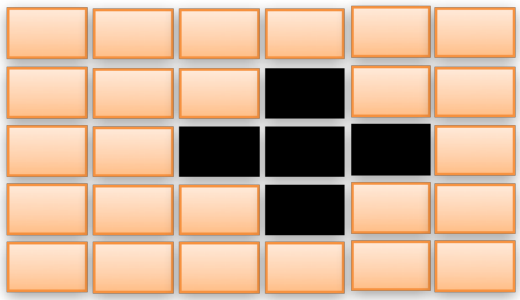
例题 5: 熄灯问题

- 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, 下, 左, 右) 的灯都会改变状态



例题 5: 熄灯问题

- 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

- 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
- 给定矩阵中每盏灯的初始状态，求一种按按钮方案，使得所有的灯都熄灭

例题 5: 熄灯问题

- 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。

例题 5: 熄灯问题

- 输入

  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

- 输出

  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

例题 5: 熄灯问题

解题思路:

- 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

例题 5: 熄灯问题

解题思路:

- 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

例题 5: 熄灯问题

解题思路:

- 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

例题 5: 熄灯问题

解题思路:

- 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

例题 5: 熄灯问题

解题思路:

- 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

例题 5: 熄灯问题

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

例题 5: 熄灯问题

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

例题 5: 熄灯问题

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

例题 5: 熄灯问题

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

例题 5: 熄灯问题

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 $2^6 = 64$

例题 5: 熄灯问题

```
1  #include <memory.h>
2  #include <string.h>
3  #include <stdio.h>
4  int getBit(char c,int i) {  //取 c 的第 i 位
5    return  (c >> i) & 1;
6  }
7  char setBit(char c, int i, int v) {//设置 c 的第 i 位为 v
8    if (v)  c |= ( 1 << i);
9    else  c &= ~( 1 << i);
10   return c;
11 }
12 char flip(char  c, int i) { //将 c 的第 i 位为取反
13   c ^= (1 << i);
14   return c;
15 }
16 void outputResult(char result[]) { //输出结果
17   int i, j;
18   for( int i = 0;i < 5; ++i ) {
19     for( int j = 0; j < 6; ++j ) {
20       printf("%d ", getBit(result[i],j));
21     }
22     printf("\n");
23   }
24 }
```

例题 5: 熄灯问题

```
25  int main()   {
26    int n, i, j;
27    char oriLights[5]; //最初灯矩阵，一个比特表示一盏灯
28    char lights[5];    //不停变化的灯矩阵
29    char result[5];    //结果开关矩阵
30    char switchs;    //某一行的开关状态
31    memset(oriLights, 0, sizeof(oriLights));
32    for(i = 0;i < 5; i++) { //读入最初灯状态
33      for(j = 0; j < 6; j++) {
34        int s;
35        scanf("%d", &s);
36        oriLights[i] = setBit(oriLights[i], j, s);
37      }
38    }
```

例题 5: 熄灯问题

```
39    for(n = 0; n < 64; n++) { //遍历首行开关的 64 种状态
40      memcpy(lights,oriLights,sizeof(oriLights));
41      switchs = n;
42      for (i = 0;i < 5; i++) {
43        result[i] = switchs; //第 i 行的开关方案
44        for (j = 0; j < 6; j++) {
45          if (getBit(switchs,j)) {
46            if (j > 0)  lights[i] = flip(lights[i],j-1);//改左灯
47            lights[i] = flip(lights[i],j);//改开关位置的灯
48            if (j < 5)  lights[i] = flip(lights[i],j+1);//改右灯
49          }
50        }
51        if (i < 4)
52          lights[i+1] ^= switchs;//改下一行的灯
53        switchs = lights[i]; //第 i+1 行开关方案和第 i 行灯情况同
54      }
55      if (lights[4] == 0) {
56        outputResult(result);
57        break;
58      }
59    }
60    return 0;
61  }
```

使用 bitset 的解法

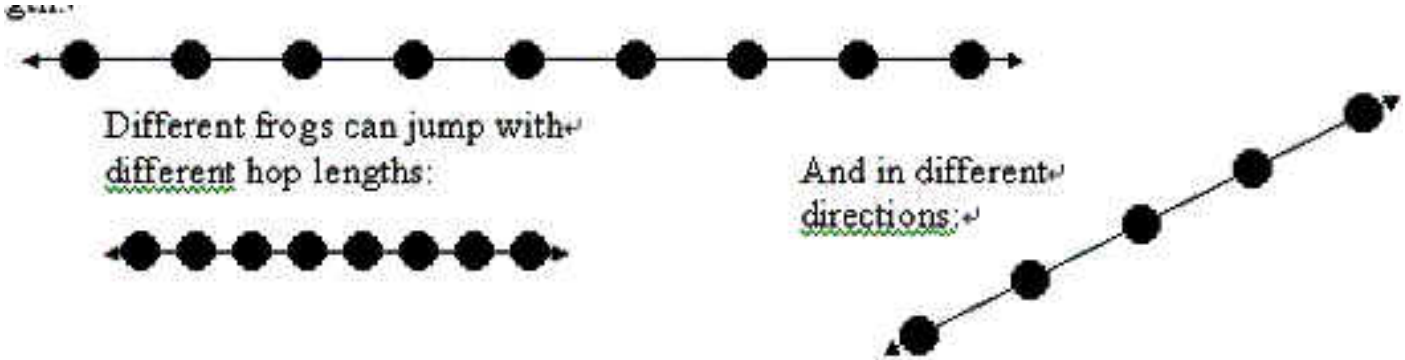
```
#include <string>
#include <cstring>
#include <iostream>
#include <bitset>
#include <algorithm>
using namespace std;
void OutputResult(bitset<6> result[]) { //输出结果
  for (int i = 0;i < 5; ++i) {
    for (int j = 0; j < 6; ++j) {
      cout << result[i][j];
      if (j < 5) cout << " ";
    }
    cout << endl;
  }
}
int main()   {
  bitset<6> oriLights[8]; //最初灯矩阵，一个比特表示一盏灯
  bitset<6> lights[8];    //不停变化的灯矩阵
  bitset<6> result[8];    //结果开关矩阵
  bitset<6> switchs;   //某一行的开关状态

    for (int i = 0;i < 5; ++i ) { //读入最初灯状态
      for (int j = 0; j < 6; ++j) {
        int s;
        cin >> s;
        oriLights[i][j] = s;
      }
    }
```

使用 bitset 的解法

```
32      for (int n = 0; n < 64; ++n) { //遍历首行开关的 64 种状态
33        copy(oriLights, oriLights+8, lights);
34        switchs = n; //第 i 行的开关状态
35        for (int i = 0;i < 5; ++i) {
36          result[i] = switchs; //第 i 行的开关方案
37          for (int j = 0; j < 6; ++j) {
38            if (switchs[j]) {
39              if (j > 0) lights[i].flip(j-1);  //改左灯
40              lights[i].flip(j); //改开关位置的灯
41              if (j < 5) lights[i].flip(j+1); //改右灯
42            }
43          }
44          if (i < 4) lights[i+1] ^= switchs;//改下一行的灯
45          switchs = lights[i]; //第 i+1 行开关方案和第 i 行灯情况同
46        }
47        if (lights[4] == 0) {
48          OutputResult(result);
49          break;
50        }
51      } // for( int n = 0; n < 64; n ++ )
52    return 0;
53  }
```

例题 6: 讨厌的青蛙

- 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路径。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



稻田里的稻子组成一个栅格，每棵稻子位于一个格点上。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

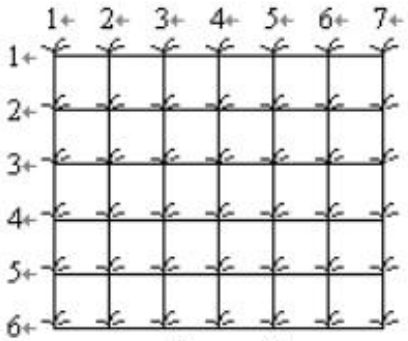


Figure-1

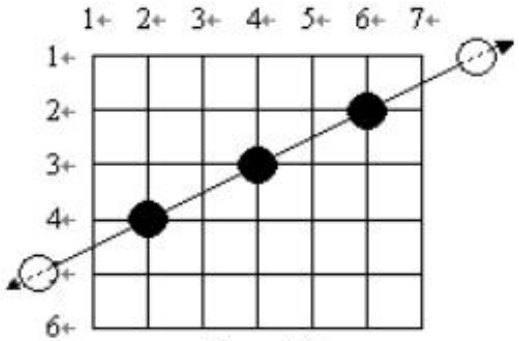


Figure-2

例题 6: 讨厌的青蛙

可能会有多只青蛙从稻田穿越。青蛙的每一跳都恰好踩在一棵水稻上，将这棵水稻拍倒。有些水稻可能被多只青蛙踩踏。当然，农民所见到的是图 4 中的情形，并看不到图 3 中的直线，也见不到别人家田里被踩踏的水稻。

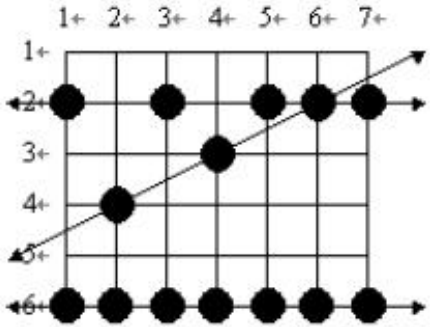


Figure-3

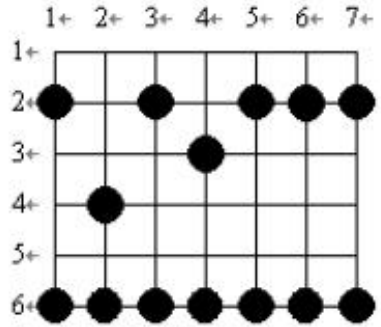


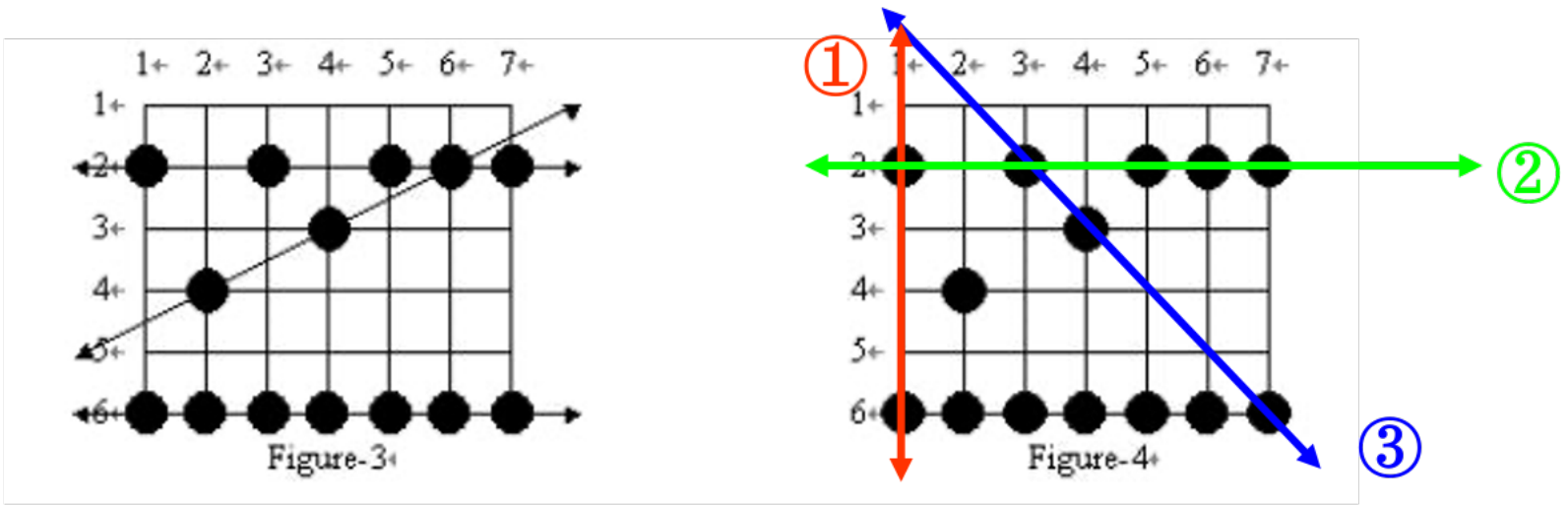
Figure-4

农民能够构造出青蛙穿越稻田时的行走路径，并且只关心那些在穿越稻田时至少踩踏了 3 棵水稻的青蛙。因此，每条青蛙行走路径上至少包括 3 棵被踩踏的水稻。

例题 6: 讨厌的青蛙

而在一条青蛙行走路径的直线上，也可能会有些被踩踏的水稻不属于该行走路径

1. 不是一条行走路径：只有两棵被踩踏的水稻；
2. 是一条行走路径，但不包括（2，6）上的水道；
3. 不是一条行走路径：虽然有 3 棵被踩踏的水稻，但这三棵水稻之间的距离间隔不相等。



请你写一个程序，确定：在一条青蛙行走路径中，最多有多少颗水稻被踩踏。例如，图 4 的答案是 7，因为第 6 行上全部水稻恰好构成一条青蛙行走路径。

例题 6: 讨厌的青蛙

- 输入
  从标准输入设备上读入数据。
  第一行上两个整数 R、C，分别表示稻田中水稻的行数和列数，$1 \leq R$、$C \leq 5000$。
  第二行是一个整数 N，表示被踩踏的水稻数量，$3 \leq N \leq 5000$。
  在剩下的 N 行中，每行有两个整数，分别是一颗被踩踏水稻的行号 (1∼R) 和列号 (1∼C)，两个整数用一个空格隔开。而且，每棵被踩踏水稻只被列出一次。
- 输出
  从标准输出设备上输出一个整数。如果在稻田中存在青蛙行走路径，则输出包含最多水稻的青蛙行走路径中的水稻数量，否则输出 0。

例题 6: 讨厌的青蛙

枚举什么?

例题 6: 讨厌的青蛙

枚举什么?

枚举青蛙踩踏的前两点（5000*5000)，步长和方向也就随之确定

假设一只青蛙进入稻田后踩踏的前两棵水稻分别是 $(X_1,_1),(X_2,Y_2)$. 那么: 青蛙每一跳在 X 方向上的步长 $d_X = X_2 - X_1$, 在 Y 方向上的步长 $d_Y = Y_2 - Y_1$;

例题 6: 讨厌的青蛙

枚举什么?

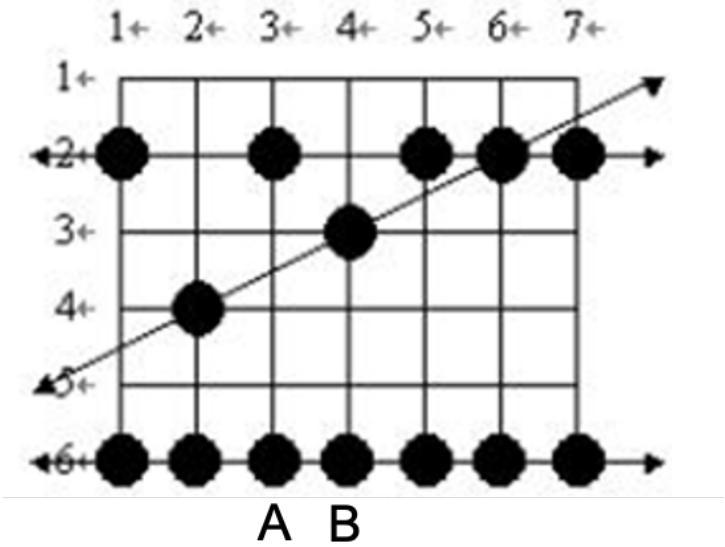
枚举青蛙踩踏的前两点（5000*5000)，步长和方向也就随之确定

假设一只青蛙进入稻田后踩踏的前两棵水稻分别是 $(X_1,_1),(X_2,Y_2)$. 那么: 青蛙每一跳在 X 方向上的步长 $d_X = X_2 - X_1$, 在 Y 方向上的步长 $d_Y = Y_2 - Y_1$;
接下来判断每走一步是否都会踩到水稻 $5000/max(d_X, d_Y) * T$，如果能，则记录一共走多少步后出稻田 (T 是判断时间)

例题 6: 讨厌的青蛙

及早排除不必要的尝试

- 要判断前两点的合法性

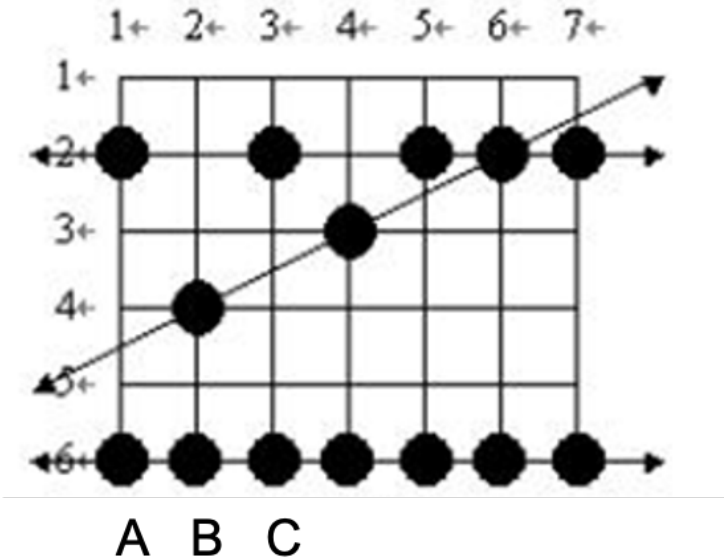


若选了 A,B 作为前两点，立即能判断这不成立。因步长为 1，而青蛙不可能从稻田外一步跳到 A。
直接否定这前两点的假设，不必进行后续判断

例题 6: 讨厌的青蛙

及早排除不必要的尝试

- 如果已经发现踩倒 n 棵水稻的路径，那么，最多不超过 n 步就会跳出稻田的前两点的方案，就可以直接否定。
- 有序，而非随机枚举前两点

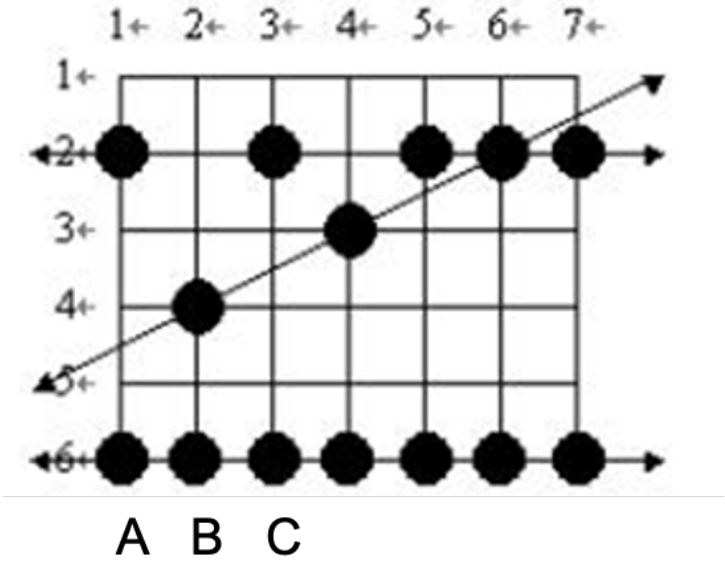


若先枚举了 A 和 B 作为前两点并发现可行，则枚举 AC 作为前两点就没有必要了。

例题 6: 讨厌的青蛙

及早排除不必要的尝试

- 有序，而非随机枚举前两点

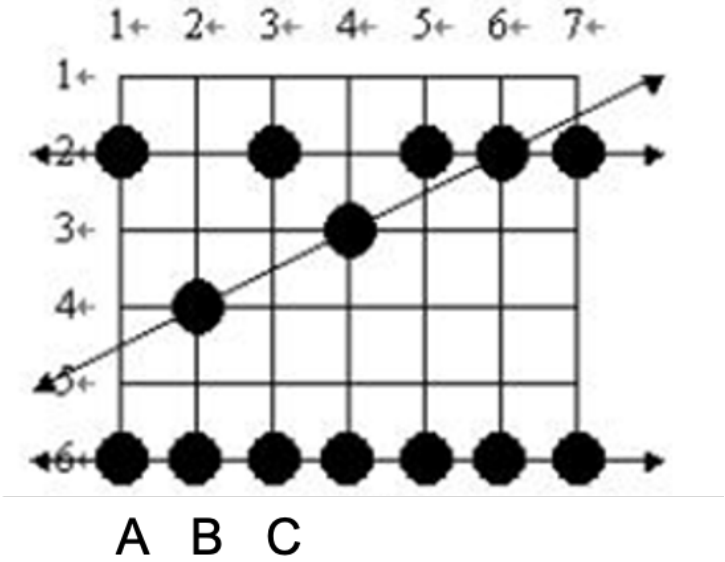


为避免对 AC 的没必要枚举，应该确定枚举顺序，即在第一点相同时，优先枚举 X 方向上步长短的（Y 方向亦可）。如何做到这个枚举顺序?

例题 6: 讨厌的青蛙

及早排除不必要的尝试

- 有序，而非随机枚举前两点



为避免对 AC 的没必要枚举，应该确定枚举顺序，即在第一点相同时，优先枚举 X 方向上步长短的（Y 方向亦可）。如何做到这个枚举顺序?

将所有水稻按 X 坐标从小到大排序，X 坐标相同则按 Y 坐标排序。枚举时确保第一点 X 坐标一定小于等于第二点 X 坐标。

例题 6: 讨厌的青蛙

如何判断某一步是否踩到水稻
猜测一条行走路径时，需要从当前位置 (X, Y) 出发上时，看看 $(X + d_X, Y + d_Y)$ 位置的水稻是否被踩踏

- 方案 1: 设置标记数组，常数时间即可知道某点上有无水稻。代价：5000*5000 的数组
- 方案 2: 用 binary_search() 从已经排好序的水稻数组中查找 $(X + d_X, Y + d_Y)$ 看是否存在

例题 6: 讨厌的青蛙

```cpp
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <cstring>
#include <bitset>
using namespace std;
int r, c, n;
struct Plant {//水稻排序规则，先按 x 坐标从小到大排，x 坐标相同则按 y 坐标从小到大排
  int x, y;
  bool operator < (const Plant & p) {
    if (x == p.x )
      return y < p.y;
    return x < p.x ;
  }
};
Plant plants[5010];
//char plantExists[5010][5010]; //plantExists[i][j] == 1 表示（i,j）处有水稻
bitset<5010> plantExists[5010]; //更节省空间
//判断以 secPlant 作为第 2 点，步长为 dx, dy，那么最多能走几步
int maxSteps(Plant secPlant, int dX, int dY) {
  Plant p;
  int steps = 2;
  p.x = secPlant.x + dX;
  p.y = secPlant.y + dY;
  while (p.x <= r && p.x >= 1 && p.y <= c && p.y >= 1) {
    if (!plantExists[p.x][p.y]) return 0; //每一步都必须踩倒水稻才算合理，否则这就不是一条行走路径
    //或: if(!binary_search(plants,plants+n,p))
    p.x += dX;
    p.y += dY;
    ++steps;
  }
  return steps;
}
```

例题 6: 讨厌的青蛙

```
34  int main() {
35    int i, j, dX, dY, pX, pY, steps, max = 2;
36    memset(plantExists,0,sizeof(plantExists));
37    scanf("%d %d", &r, &c);
38    //行数和列数, x 方向是上下, y 方向是左右
39    scanf("%d", &n);
40    for (i = 0; i < n; i++) {
41      scanf("%d %d", &plants[i].x, &plants[i].y);
42      plantExists[plants[i].x][plants[i].y] = 1;
43    }
44    sort(plants, plants + n);//将水稻按 x 坐标从小到大排序, x 坐标相同按 y 从小到大排序
45    for (i = 0; i < n - 2; i++)  //plants[i] 是第一个点
46      for ( j = i + 1; j < n - 1 ; j++) {  //plants[j] 是第二个点
47        dX = plants[ j ].x - plants[i].x;
48        dY = plants[ j ].y - plants[i].y;
49        pX = plants[ i ].x - dX; //(pX,pY) 是第一点之前的的点
50        pY = plants[ i ].y - dY;
51        if (pX <= r && pX >= 1 && pY <= c && pY >= 1) continue;
52        //第一点的前一点在稻田里, 说明本次选的第二点导致的 x 方向步长不合理 (太小), 取下一个点作为第二点
53        if (plants[i].x + (max - 1) * dX > r)  break;
54  //x 方向过早越界了. 说明本次选的第二点不成立
55  //如果换下一个点作为第二点, x 方向步长只会更大, 更不成立, 所以应该
56  //认为本次选的第一点必然是不成立的, 那么取下一个点作为第一点再试
57        pY = plants[ i ].y + (max - 1) * dY;
58        if (pY > c || pY < 1) continue; //y 方向过早越界了, 应换一个点作为第二点再试
59        steps = maxSteps(plants[j], dX, dY); //看看从这两点出发, 一共能走几步
60        if (steps > max) max = steps;
61      }
62    if (max == 2) max = 0;
63    printf("%d\n", max);
64    return 0;
65  }
66
```

例题 6: 讨厌的青蛙

小结

- 注意枚举顺序
- 尽早排除不可能的情况