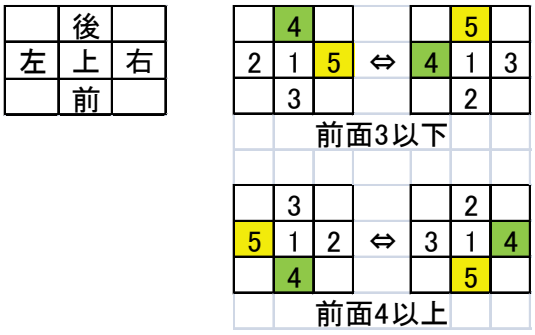
ACM ICPC 2012 国内予選問題 C 「偏りのあるサイコロ」

- この問題を解くためには、サイコロを倒せる方向と、倒した時のサイコロの状態(サイコロの上面と前面の数値)を知る必要がある。
- 倒せる方向は、{4, 5, 6}の面の方向であるが、{4, 5, 6}の内一つは上面または底面にくるので、倒せる方向の可能性は２通りしかない。その２通りのうち、大きい数値の面が優先される。これより、倒す側の面の数値の候補は次のようになる。

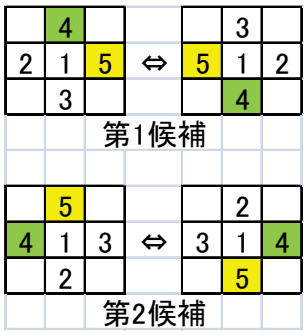
上面数値	1	2	3	4	5	6
第１候補	5	6	6	6	6	5
第２候補	4	4	5	5	4	4

第１候補 ＝(上面 == 1 || 上面 == 6)？5：6
第２候補 ＝(上面 == 3 || 上面 == 4)？5：6

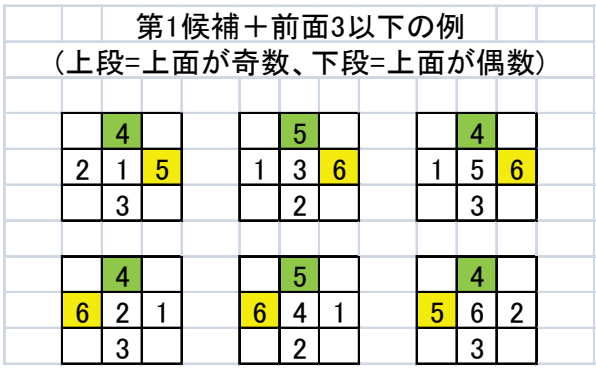
- 倒す方向は以下のように求めることができる。
  - ◆ 前面の数値＝倒す面の数値　⇒　前へ倒す
  - ◆ 前面の数値＝７−倒す面の数値　⇒　後へ倒す
  - ◆ 上記以外の場合、右または左へ倒すことになる。
    - ① 上面の数値が同じで、全面の数値が３以下か４以上のいずれかの場合、倒す方向が第１候補か第２候補かで左右が変わる。



② 上面の数値が同じで、倒す方向の候補(第 1 候補か第 2 候補)も同じ場合、前面の数値が 3 以下か 4 以上かにより倒す方向の左右が変わる。



③ 倒す方向の候補(第 1 候補か第 2 候補)が同じで、前面の数値が 3 以下か 4 以上のいずれかの場合、上面の数値が偶数か奇数かにより倒す方向の左右が変わる。



これらより、**右に倒れるのは、(第 1 候補 ⊕ 上面 = 奇数 ⊕ 前面 ≦ 3) が 1 の時**で、それ以外の時は左に倒れる。

- 回転前のサイコロの上面と前面の値を各々T, F とする。回転後のサイコロの上面と前面の値を各々T', F'とすると、T', F' は回転方向に応じて以下の表のようになる。

回転方向	前	後	右	左
T'=	B=7−F	F	L=7−R=7−倒す面の数値	R=7−L=7−倒す面の数値
F'=	T	7-T	F	F

※ 右または左へ倒す場合 T' = 6 – MIN(T, 7−T) – MIN(F, 7−F) で計算出来る。

∵ 倒す方向の面の数値の裏側の数値が上面に出てくるが、倒す方向の面の数値は 4 以上なので、上面に出てくる数値は 3 以下である。また、上面、前面、それらの裏側の面は、上面には出てこない。

- サイコロの転がり状況は２種類のマップを用いて管理
  - マップＡ：　その位置に既に置かれているサイコロの個数
  - マップＢ：　　その位置の一番上のサイコロの上面の数値

※ 倒す方向の位置に置かれているサイコロの個数が現在の位置のサイコロの個数より少なければ、そちらの方向に倒すことができる。

※ サイコロの個数は 100 以下なので、マップとしては 200x200 の 2 次元配列を用い、サイコロは(100, 100)の位置に置くものとする。

【プログラム例】

```
// ACM ICPC 2012 国内予選問題 C
// http://www.cs.titech.ac.jp/icpc2012/icpc2012-mondai/icpc2012/contest/C_ja.html
// Filename = pc.c
// Compile： cc pc.c
// Execution：./a.out < C0 > C0.result
// Check： diff C0.ans C0.result

#include <stdio.h>

#define MAX(a,b) ((a)>(b) ? (a) : (b))
#define MIN(a,b) ((a)<(b) ? (a) : (b))
// 方向の定義
#define F 0                     // 前
#define R 1                     // 右
#define B 2                     // 後
#define L 3                     // 左

// サイコロの状態を表す構造体
typedef struct {
  int top;                      // 上面の値
  int front;                    // 前面の値
} state_t;

int map[200][200];              // 各位置に置かれているサイコロの個数
int tv[200][200];               // 各位置の最上面の値
int count[6];                   // 上からメイル数値を数えるための配列
// 倒す方向に伴う座標の増減値
int dx[] = {0, 1, 0, -1};
int dy[] = {-1, 0, 1, 0};

// 第 1 候補の倒す方向を求める。s = サイコロの状態
int get_dir1(state_t s)
{
  int num;                                  // 倒す方向の面の数値
  num = (s.top==1 || s.top==6)?5:6;
  if(s.front == num) return F;              // 倒す面の数値＝前面の数値　→　前
```

```
  if((7-s.front) == num) return B;        // 倒す面の数値＝後面の数値　→　後
  if(s.top%2){                            // 上面の数値が奇数で
    if(s.front <= 3) return R;                    // 前面の数値が 3 以下　→　右
    else return L;                                // 前面の数値が 4 以上　→　左
  }
  else {                                  // 上面の数値が偶数で
    if(s.front <= 3) return L;                    // 前面の数値が 3 以下　→　左
    else return R;                                // 前面の数値が 4 以上　→　右
  }
}

// 第 2 候補の倒す方向を求める。s = サイコロの状態
int get_dir2(state_t s)
{
  int num;                                // 倒す方向の面の数値
  num = (s.top==3 || s.top==4)?5:4;
  if(s.front == num) return F;            // 倒す面の数値＝前面の数値　→　前
  if((7-s.front) == num) return B;        // 倒す面の数値＝後面の数値　→　後
  if(s.top%2){                            // 上面の数値が奇数で
    if(s.front <= 3) return L;                    // 前面の数値が 3 以下　→　左
    else return R;                                // 前面の数値が 4 以上　→　右
  }
  else {                                  // 上面の数値が偶数で
    if(s.front <= 3) return R;                    // 前面の数値が 3 以下　→　右
    else return L;                                // 前面の数値が 4 以上　→　左
  }
}

// 指定した方向へ倒した後の状態を求める。s = サイコロの状態, dir = 倒す方向
// 戻り値： 倒した後のサイコロの状態
state_t next_tf(state_t s, int dir)
{
  state_t ns;                     // 倒した後のサイコロの状態
  switch(dir){
  case F:             // 前
    ns.top = 7 - s.front;
    ns.front = s.top;
    break;
  case B:             // 後
    ns.top = s.front;
    ns.front = 7 - s.top;
    break;
  default:            // 左 or 右
    ns.front = s.front;
    ns.top = 6 - MIN(s.top, 7 - s.top) - MIN(s.front, 7 - s.front);
  }
  return ns;
}

int main()
```

```
{
  int n;                        // サイコロの個数
  int i,j;
  int x,y;                      // サイコロの位置
  int dir;                      // 倒す方向
  state_t s;                    // サイコロの状態

  while(1){
    scanf("%d", &n);            // サイコロの個数を読み込む
    if(n==0) break;             // 0 なら終了
    for(i=0; i<200; i++)        // 2 次元配列 map と tv の初期化
      for(j=0; j<200; j++){
          map[i][j] = 0;
          tv[i][j] = 0;
      }
    for(i=0; i<n; i++){                 // サイコロのデータを n 回読み込む
      scanf("%d %d", &s.top, &s.front);         // 上面と前面の値を読み込む
      x = y = 100;              // 最初に置く位置は(100,100)
      if(map[x][y] == 0){       // その位置にサイコロがなければ(最初のサイコロ)
          map[x][y] = 1;        // その位置のサイコロの個数を 1 とし
          tv[x][y] = s.top;     // その位置の最上面のサイコロの値をセット
          continue;             // 次のサイコロの処理へ
      }
      // サイコロの転がりをチェック
      while(1){
          dir = get_dir1(s);    // 倒れる方向の第 1 候補を求める
          if(map[x][y] > map[x+dx[dir]][y+dy[dir]]){        // その方向へ落下可能なら
            x = x + dx[dir]; y = y + dy[dir];       // その方向へ落下させた後の座標を計算
            s = next_tf(s, dir);                    // その時のサイコロの状態を求める
            continue;
          }
          dir = get_dir2(s);    // 倒れる方向の第 2 候補を求める
          if(map[x][y] > map[x+dx[dir]][y+dy[dir]]){        // その方向へ落下可能
            x = x + dx[dir]; y = y + dy[dir];       // その方向へ落下させた後の座標を計算
            s = next_tf(s, dir);                    // その時のサイコロの状態を求める
            continue;
          }
          // サイコロはこれ以上落下しない
          map[x][y]++;                  // その位置のサイコロ数を 1 増やす
          tv[x][y] = s.top;             // その位置の最上面の値を更新
          break;
      }
    }
    for(i=0; i<6; i++)                  // 最上面の値を求める配列を初期化
      count[i] = 0;
    for(i=0; i<200; i++)
      for(j=0; j<200; j++){
          if(tv[i][j] == 0) continue;   // その位置にはサイコロが無い
          count[tv[i][j]-1]++;          // その位置の最上面の数値の個数を 1 増やす
```

```
        }
    for(i=0; i<5; i++)                                  // 結果を出力
        printf("%d ", count[i]);
    printf("%d¥n", count[5]);
  }
}
```