Comet OJ - 2019 六一欢乐赛 题解

赛前每题 AC 人数预测

$A \simeq D < F \simeq H < C \simeq B < G \simeq E$

这么猜测是因为 A,D 都是零基础的题目，F, H 则是间单的观察，C,B 则是简单的算法，最后的 G,E 是对新手来说可能实作比较不容易的模拟题。

problem A: 第001话 宝可梦,就决定是你了!

简化题意

给定 n，计算 $1+2+3+\ldots+n+(n-1)+(n-2)+\ldots+1$。

考察知识点

- 推导数学公式以及运用在算法竞赛上的能力。

解析

这题算是个阅读测验题吧，理解题目后，就知道你要计算的答案是
$1+2+3+\ldots+n+(n-1)+(n-2)+\ldots+1$。

使用求和级数公式可算出：原式 =
$\sum_{i=1}^{n-1} i + n + \sum_{i=1}^{n-1} i = (n*(n-1)/2)*2+n = n*(n-1)+n = n^2$。

所以只要用以下程序即可 AC。(请记得使用 long long)

```
#include<cstdio>
int main() {
    long long n;
    scanf("%lld", &n);
    printf("%lld\n",n*n);
    return 0;
}
```

现在假设你是个把求和级数忘光光不会推导公式，但已经学会 **for** 循环的初学者，那你就能够写出下列的程序：

```cpp
#include<cstdio>
int main() {
    int n;
    scanf("%d", &n);
    long long an = 0; // 储存答案的变量
    // 把答案加上 1 ~ n-1
    for(int i = 1; i < n; i++) {
        an += i;
    }
    // 把答案加上 n ~ 1
    for(int i = n; i > 0; i--) {
        an += i;
    }
    printf("%lld\n",an);
    return 0;
}
```

但传上 OJ 后会发现... 这个程序在最慢的情况可能会跑到 1 秒左右，非常逼近本题的时间限制 (笔者在赛前测试时大概有 1/2 的机率能够通过此题)，其原因是此份代码要执行非常多次加法运算。

那么我们可以减少一个 **for** 循环，变成如下：

```cpp
#include<cstdio>
int main() {
    int n;
    scanf("%d", &n);
    long long an = n;
    for(int i = 1; i < n; i++) {
        an += 2 * i;
    }
    printf("%lld\n",an);
    return 0;
}
```

就可以很稳的在时限内通过此题了。

Bonus: 图像化的背公式技巧

顺带一提，应该也有不少人曾经背过这个公式，所以一眼就看出答案是 n^2，这个公式有一个图像化技巧可以推得。

请看下图：

我们可以发现，用 $2 \times n - 1$ 条斜线可以不重复的贯穿所有 n^2 个正方形，且这些斜线贯穿的数量正好是 $1, 2, \ldots, (n-1), n, (n-1), (n-2), \ldots, 1$。

这样即可证明 $1 + 2 + 3 + \ldots + n + (n-1) + (n-2) + \ldots + 1 = n^2$

还有另一个公式如下也可以用类似方法证明：

$$1 + 3 + 5 + \ldots + (2 \times n - 1) = \sum_{i=1}^{n} (2 \times i - 1) = n^2$$

只要画 n 条折成直角的线如下即可，每条线由左上到右下贯穿的格子数量刚好是 $1, 3, \ldots$ 至 $2 \times n - 1$。

Problem B: 第002话 宝可梦中心大对决!

简化题意

给定 n ($n \leq 14$) 个数，请问最多能选几个数，使得选上的数满足任两数互质。

考察知识点

- 使用内建数学函式的能力 (这里是指 __gcd 的使用，这部分就不再解析里介绍了，请大家自寻在网路上搜寻。)
- 枚举含有 n 个元素的集合的所有子集的能力

解析

本题 n 至多只会有 14，所以 14 个数每个数选或不选(也就是枚举所有子集)共有 $2^14 = 16384$ 种选法，于是我们可以枚举所有选法，检查是否选上的任两数都互质，最后输出任两数都互质时最多能选几个数即可。

枚举 n 个元素的所有可能子集大致上有以下两种方法：

1. 使用递归来枚举(请直接参考以下代码及代码的注释)：

```
#include<cstdio>
#include<algorithm> // 要使用 __gcd 必须 include algorithm
using namespace std;
const int MAX_N = 14;
int n;
int a[MAX_N];      // 储存这 $n$ 个数
```

```cpp
bool used[MAX_N]; // 用来纪录一个数是否被选
int an;           // 储存当前已知最好答案
/*
 * 呼叫 dfs(id) 时，代表我们正在枚举第 id 个数选或不选
 */
void dfs(int id){
    // 若 id = n，代表已经枚举出一组选取的组合了，可以开始检测是否选取的数满足任两数都互质
    if(id == n){
        for(int i = 0;i < n; i++){
            for(int j = 0; j < i; j++) {
                // 若第 i 和第 j 个数都被选取且这两个数并不互质，则可以直接 return
                if(used[i] && used[j] && __gcd(a[i], a[j]) != 1) return;
            }
        }
        // 能执行到这里代表选取的数中任两数都互质，计算出 cnt 代表共选了几个数。
        int cnt = 0;
        for(int i = 0; i < n; i++) {
            if(used[i]) cnt++;
        }
        // 如果选的数的数量大于已知最大数量，则更新答案。
        if(cnt > an) an = cnt;
        return;
    }
    used[id] = true;
    dfs(id + 1);
    used[id] = false;
    dfs(id + 1);
}
int main() {
    int T;
    scanf("%d", &T);
    for(int i = 1; i <= T; i++) {
        an = 0; // 把答案初始化为 0
        scanf("%d", &n); // 读入 n
        for(int j = 0; j < n; j++) scanf("%d", &a[j]); // 读入 n 个数
        dfs(0); // 呼叫递归函式
        printf("%d\n",an);
    }
    return 0;
}
```

2. 使用数字 $0 \sim 2^N - 1$ 来代表所有子集：

使用以下方式来把数字 x 对应到一个子集：

把一个数写成 2 进位，并补齐到 n 位数，例如说 $5 = (0101)_2, 11 = (1011)_2$。那么若 x 由右边数来第 i 位若是 1，就代表 x 所对应到的子集有包含第 i 个数，但若右边数来第 i 位是 0，则代表 x 对应到的子集不包含第 i 个数。

所以 5 对应到的子集包含第 1 个数和第 3 个数，而 11 对应到的子集包含第 1、2、4 个数。

用此概念加上位运算，可以写出下列能在此题获得 AC 的代码：

```
#include<cstdio>
#include<algorithm> // 要使用 __gcd 必须 include algorithm
using namespace std;
const int MAX_N = 14;
int a[MAX_N];
// 此函式能获得 x 从最低位数来第 i 位的数字(位数从 0 开始编号)
int get_bit(int x, int i) {return (x >> i) & 1;}
// 检查 x 对应到的子集是否满足任两数都互质
bool coprime_in_subset(int n, int x){
    for(int j = 0; j < n; j++) {
        for(int k = 0; k < j; k++) {
            if(get_bit(x, j) && get_bit(x, k)) {
                if(__gcd(a[j], a[k]) != 1) return false;
            }
        }
    }
    return true;
}
void solve_one_test() {
    int n;
    int an = 0;
    scanf("%d", &n);
    for(int i = 0;i < n;i++) scanf("%d", &a[i]);
    for(int i = 0;i < (1<<n); i++){
        if(coprime_in_subset(n,i)){
            int cnt = 0;
            for(int j = 0;j < n; j++){
                cnt += get_bit(i,j);
            }
            if(an < cnt) an = cnt;
        }
    }
    printf("%d\n",an);
}
```

```
int main() {
    int T; // 储存有几组数据
    scanf("%d",&T);
    for(int _t = 1; _t <= T; _t++) {
        solve_one_test();
    }
}
```

时间复杂度估计

令 MAX_V 为 a_i 的最大值。

由于总共有 2^n 个子集，对于每个子集，要枚举任两个数字是否互质($O(n^2)$个组合)，计算是否互质的方法是采用求最大公因数是否为 1，这部分的时间复杂度为 $O(\log MAX\_V)$，故总时间复杂度为 $O(2^n \times n^2 \times \log MAX\_V)$。

但若更加聪明的使用递归，以及预处理任两个数是否互质，再加上用位元压缩来储存每个数和哪些数户质，本题可以使用普通的递归做到 $O(2^n)$。参考代码如下：

```
#include<cstdio>
#include<algorithm>
using namespace std;
const int MAX_N = 14;
int n;
int a[MAX_N];
int coprime[MAX_N];
int an;
void dfs(int id, int cnt,int choosen_mask){
    if(id == n){
        if(an < cnt) an = cnt;
        return;
    }
    if((coprime[id] & choosen_mask) == choosen_mask)
       dfs(id + 1, cnt + 1, choosen_mask | (1 << id));
    dfs(id + 1, cnt, choosen_mask);
}
void pre_do(){
    for(int i = 0; i < n; i++) {
        coprime[i] = 0;
        for(int j = 0; j < n; j++) {
            if(i != j && __gcd(a[i] , a[j]) == 1)
coprime[i] |= 1 << j;
        }
    }
}
```

```
int main() {
    int T;
    scanf("%d", &T);
    for(int i = 1; i <= T; i++) {
        an = 0;
        scanf("%d", &n);
        for(int j = 0; j < n; j++) scanf("%d", &a[j]);
        pre_do();
        dfs(0, 0, 0);
        printf("%d\n",an);
    }
    return 0;
}
```

关于递归的时间复杂度相关的学习资料可参考 从枚举到 K 短路 - dreamoon的文章 - 知乎

延伸学习：转换成图论问题 --- 最大团问题(Maximum Clique Problem)

若把每个数当成图论上的一个点，若两个数字互质，就把他们代表的点对间连上边。那么此问题其实就是最大团问题了，所以本题可以套用任何能解最大团的算法来解决，若用此方法，n 高达 50 时仍都可以在规定时限内算出答案。

problem C: 第003话 收服宝可梦吧!

简化题意

给你两个字符串 s 和 t，请问可不可以借由移除 s 中的两个字符后变成 t。

考察知识点

- dp 的概念
- 贪心算法的概念

解析

这题至少有两个可行的解题方向：**dp** 和 **贪心**，以下一一来分析。

使用 dp

熟悉 最长公共子序列(LCS, Longest Common Subsequence) 问题的人，或许能发现，此题就是在问，s 的长度是否为 t 的长度加 2 且 s 和 t 的 LCS 长度就是 t 的长度。

但这题直接使用 LCS 是会超时的，但我们可以用另外种思维去解它。

记 s_i 代表 s 的第 i 个字符，t_i 代表 t 的第 i 个字符。并令 $s_{prefix}[i]$ 代表 s 长度为 i 的前缀，$t_{prefix}[i]$ 同理。

使用 $dp[i][j] = true$ 代表 $s_{prefix}[i]$ 是否拿移除 j 个字符后变成 $t_{prefix}[i-j]$，否则 $dp[i][j] = false$。

如此一来我们可以列出关系式：

> $dp[i][j] = true$ 当且仅当 $(dp[i-1][j] = true$ 且 $s_i = t_{i-j})$ 或 $(j > 0$ 且 $dp[i-1][j-1] = true)$，

这是因为 $s_{prefix}[i]$ 若能移除 j 个字符变成 $t_{prefix}[i-j]$，那么可分为两种情况：

1. $s_{prefix}[i]$ 的最后一个字符没被移除，那么必须 $dp[i-1][j] = true$ 且 $s_i = t_{i-j}$
2. $s_{prefix}[i]$ 的最后一个字符被移除，那么 $s_{prefix}[i-1]$ 必须能移除 $j-1$ 个字符后变成 $s_{prefix}[i-j]$

题目所求为 $dp[length(s)][2]$， 所以我们只要依序计算出 dp[1..length(s)][0..2] 就行了。

参考代码如下：

```
#include<cstdio>
#include<cstring>
const int SIZE = 1000010;
char s[SIZE];
char t[SIZE];
int dp[SIZE][3];
void solve(){
    scanf("%s%s", s + 1, t + 1);
    int n = strlen(s + 1);
    int m = strlen(t + 1);
    if(m != n - 2){
        puts("0");
        return;
    }
    for(int i = 1;i <= n; i++) {
        for(int j = 0;j <= 2; j++) dp[i][j] = false;
    }
    dp[0][0] = true;
    for(int i = 1;i <= n; i++){
```

```
        for(int j=0;j<=2;j++){
            if(dp[i-1][j] && s[i] == t[i-j]) dp[i][j] =
true;
            if(j > 0 && dp[i-1][j-1]) dp[i][j] = true;
        }
    }
    if(dp[n][2]) puts("1");
    else puts("0");
}
int main() {
    int T;
    scanf("%d", &T);
    while(T--) {
        solve();
    }
}
```

虽然这个做法可以解出这题，但还是令人觉得有些可惜，怎么说呢？请想想看，如果现在题目改成问你 s 能否移除 k 个字符变成 t (k 可以是任意整数)，那这个做法还能做吗？并不，因为会超时。

现在我们就来看另一种思路的解法。

使用 贪心

我们可以使用以下贪心算法解这题：

> *对于字符串 t，由左而右依序在 s 中寻找最左边可以配对的合法字符去配对。*

可参考以下代码：

```
#include<cstdio>
#include<cstring>
const int SIZE = 1000010;
char s[SIZE];
char t[SIZE];
void solve(){
    scanf("%s%s", s + 1, t + 1);
    int n = strlen(s + 1);
    int m = strlen(t + 1);
    if(m != n - 2){
        puts("0");
        return;
    }
    int j = 1;
```

```
    for(int i = 1; i <= m; i++) {
        /*
         * 每一次进入循环都是在寻找 t[i] 可以配对的最左边的字符
         * 这个字符必须满足和 t[i] 相等以及位置比 t[i-1] 配对的
位置还要右边
         */
        while(j <= n && t[i] != s[j]) j++;
        // 若 j > n，代表找不到 s[i] 配对的字符。
        if(j > n) {
            puts("0");
            return;
        }
        j++;
    }
    // 程序能执行到这里，代表 t 的每一个字符都能在 s 中找到配对
    puts("1");
}
int main() {
    int T;
    scanf("%d", &T);
    while(T--) {
        solve();
    }
}
```

只要把 `if(m != n - 2)` 改成 `if(m != n - k)`，就能判断 s 是否能移除 k 个字符变成 t 了。

problem D: 第004话 武士少年的挑战!

简化题意

给你 18 张数字牌，每张牌上有 0 ～ 13 中的其中一个数字，若有两张一模一样的非零数字牌，就可丢弃，问最后最少能够剩下几张牌?

考察知识点

- 正确理解题意
- c++ 基础语法

解析

嗯...这题游戏的来源其实就是抽鬼牌游戏，但百度后发现抽鬼牌的变形好多，怕大家因为看到抽鬼牌这几个字而误会题目意思，就完全不提到抽鬼牌这几个字了。

直接附上参考代码：

```
#include<cstdio>
int cnt[14]; // 记录每个数字出现几次
int main() {
    for(int i = 0; i < 18; i++) {
        int x;
        scanf("%d", &x);
        cnt[x]++;
    }
    int an = 18;
    for(int i = 1; i <= 13; i++) {
        an -= cnt[i] / 2 * 2;
    }
    printf("%d\n", an);
    return 0;
}
```

problem E: 第005话 尼比市的决斗!

简化题意

有一个 $n \times n$ 迷宫，格子分成可走格子和墙壁，告诉你起点和终点位置，保证起点下方一定是墙壁。现在让你右手贴着起点下方的墙壁，一直往前走，保证一定能走到终点，请输出沿途中你右手贴着的每一面墙壁在地图的哪一个方向。

考察知识点

- 将问题转化为数学模型并且实际写代码模拟的能力
- ~~猜对出题者想问什么的能力~~

解析

这题乍看很复杂，但其实只要考虑三种情况，用 **while** 循环包住即可。

三种情况如下：(对于每种情况都有个已知前提：目前右手贴着墙，且头的朝向与右边的墙平行)

1. 前方的格子也是墙，此时继续走右手就会贴到前方的墙，身体仍在同一个格子，但左转了九十度。
2. 前方的格子不是墙，但右前方的格子是墙，此时身体会前进一格，方

向保持不变。

3. 前方的格子和右前方的格子都不是墙，此时就会身体先前进一格，再向右一格，并且过程中身体会向右转九十度。

下图是情况一的示例图(其他情况绘师懒得画了(>_<))

若把当前的身体朝向也考虑进去，实际上有 12 种情况，直觉的写法可能会写出这样的 12 种 case，但好一点的写法可以只用到三个 if else 的循环，请参考以下代码：

```
#include<cstdio>
#include<string>
using namespace std;
const int MAX_N = 15;
const int MIN_N = 5;
int n; // 地图大小
/*
 * 储存地读，格子座标是 (1,1) ~ (n,n)
 */
char s[MAX_N+1][MAX_N+2];
/*
 * 在这里我们把所有方向编号，依序是下 -> 右 -> 左 -> 上(编号0~3)
 * 由座标(x,y)移动到做座标(x+dx[i],y+dy[i])就是往方向 i 走
 */
int dx[4] = {1, 0,-1, 0};
int dy[4] = {0, 1, 0,-1};
char cc[5] = "DRUL"; //代表每个方向的字符
void read(){
    scanf("%d", &n);
    for(int i = 1; i <= n; i++) scanf("%s", s[i] + 1);
}
//若当前右手贴着墙的方向是 id，会回传向右转后的右手踢着的墙的方向
int turn_right(int id){
    return (id + 3) & 3;
}
//若当前右手贴着墙的方向是 id，会回传向左转后的右手踢着的墙的方向
int turn_left(int id){
    return (id + 1) & 3;
}
//已知起点在座标(x,y)
void solve(int x, int y){
    /* dir是目前脸朝向的方向
     * 初始时脸一定朝向右边，
     * 且右手贴着下方的墙
     */
    int dir = 1;
    putchar('D');
```

```
    while(true){
        /*
         * (head_x, head_y) 是前方的格子
         */
        int head_x = x + dx[dir];
        int head_y = y + dy[dir];
        if(s[head_x][head_y] == '#'){ // 第一种情形：前方是
墙壁
            // 印出前方的墙壁方向。
            putchar(cc[dir]);
            // 左转
            dir = turn_left(dir);
        }
        else{
            /*
             * (head_right_x, head_right_y) 是右前方格子的
座标
             */
            int head_right_x = x + dx[dir] +
dx[turn_right(dir)];
            int head_right_y = y + dy[dir] +
dy[turn_right(dir)];
            // 第二种情况：前方是格子，右前方是墙壁
            if(s[head_right_x][head_right_y] == '#') {
                // 印出右边的方向
                putchar(cc[turn_right(dir)]);
                // 前进一格
                x += dx[dir];
                y += dy[dir];
                // 如果前进到终点则程序结束
                if(s[x][y]=='T') return;
            }
            // 第三种情况：前方和右前方都是格子
            else{
                // 先前进一格
                x += dx[dir];
                y += dy[dir];
                // 如果前进到终点则程序结束
                if(s[x][y]=='T')return;
                // 右转 90 度
                dir=turn_right(dir);
                // 再前进一个
                x += dx[dir];
                y += dy[dir];
                // 印出当前右手贴着的方向
                putchar(cc[turn_right(dir)]);
                // 如果前进到终点则程序结束
```

```
                if(s[x][y] == 'T')return;
            }
        }
    }
}
int main(){
    read();
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            if(s[i][j] == 'S'){
                solve(i,j);
                puts("");
            }
    return 0;
}
```

problem F: 第006话 皮皮和月亮石!

简化题意

有 $n \times m$ 的格子，每次变化会多出一格不能种树，对于每次变化，请输出有多少个不同位置可以沿着边线种大小为 2×2 树。

考察知识点

- 简单的逻辑思考

解析

最直觉的做法应该是：对于每次变化后，都枚举还有哪些位置可以种树。，这样的话时间复杂度会是 $O(q \times n \times m)$，不足以应付这题。

我们能发现，每多一个格子不能用时，最多多 4 个位置不能种树，所以只要对于每次变化计算多出几个位置不可以种树即可。

参考代码如下：

```
#include<cstdio>
const int MAX_N = 1000;
int can[MAX_N + 1][MAX_N + 1]; // 记录哪些位置可以当做要种的树的左上角
int main(){
    int now_answer = 0;
    int n, m, q;
    scanf("%d%d%d", &n, &m, &q);
```

```
    for(int i = 1; i < n; i++)
        for(int j = 1; j < m; j++) {
            can[i][j] = 1;
            now_answer++;
        }
    for(int k = 1; k <= q; k++) {
        int x, y;
        scanf("%d%d", &x, &y);
        for(int i = 0; i < 2; i++) {
            for(int j = 0 ; j < 2; j++) {
                if(can[x - i][y - j]) {
                    now_answer--;
                    can[x - i][y - j] = 0;
                }
            }
        }
        printf("%d\n",now_answer);
    }
}
```

problem G: 第007话 华篮市的水中花!

简化题意

模拟两个人玩黑白棋，每个人会拥有一个 64 个位置的排列，每个人的下棋策略都是选择这个排列中，把棋摆在第一个能下的位置，求游戏终盘局面。

考察知识点

- 能够写出实用的小游戏程序
- ~~没玩过黑白棋的人可以当场学一下黑白棋怎么玩~~

解析

就是个大模拟，好像没什么好说的，可能要注意一夏祎些特殊情形，不要自己猜测一些没有根据假设，例如说，判断当前是否为终盘局面就请直接按照规则来：两个玩家都无处可下，如果猜测说终盘只可能发生在 64 格全下满，或是全部棋子都是清一色，那就错掉了。

直接附送代码:

```
#include<cstdio>
#include<set>
#include<algorithm>
#include<assert.h>
```

```
using namespace std;
const int EMPTY = 0;
const int BLACK = 1;
const int WHITE = 2;
const int N = 8;
struct ReversiSimulator{
    /* grid[x][y] 代表 棋盘上格子(x,y) 的当前状态
     * 对于 dir 从 0~7, (dx[dir],dy[dir]) 代表八个方向的位移
     */
    int grid[N+1][N+1];
    int dx[8] = {1,0,-1,0,1,1,-1,-1};
    int dy[8] = {0,1,0,-1,1,-1,1,-1};
    // 初始化盘面
    void init() {
        for(int i = 1; i <= 8; i++)
            for(int j = 1; j <= 8; j++) {
                grid[i][j] = EMPTY;
            }
        grid[4][4] = grid[5][5] = WHITE;
        grid[4][5] = grid[5][4] = BLACK;
    }
    // 翻转格子(x,y)里的旗子的颜色
    void rev(int x, int y){
        assert(grid[x][y] != EMPTY); // 确定不会翻到没有棋子
的格子
        grid[x][y] = 3 - grid[x][y];
    }
    // 判断一个座标是否超出边界
    bool out(int x, int y){
        return x <= 0 || y <= 0 || x > N || y > N;
    }
    // 判断颜色 col 可不可以下在格子 (x, y), 若可以, 就更新盘面
    bool put(int col, int x, int y){
        if(grid[x][y] != EMPTY)return 0;
        // 将纪录颜色 col 的棋若摆在位置 (x,y) 可以夹着对手哪些位
置的棋
        set<pair<int,int>> eaten_pos;
        //检查每个方向是否可以夹着对方旗子
        for(int dir = 0; dir < 8; dir++){
            int now_x = x + dx[dir];
            int now_y = y + dy[dir];
            set<pair<int,int>> enemy_pos;
            bool arrived = 0;
            // 遇到该方向第一个空格或超出边界则break
            while(!out(now_x,now_y) && grid[now_x][now_y]
!= EMPTY){
                if(grid[now_x][now_y] == col) {
```

```
                    // 遇到自己颜色的棋子则标另一端有自己的棋子并
break
                    arrived = 1;
                    break;
                }
                else{
                    // 遇到对手棋子则把此位置插入纪录对手棋子位置
的 set
                    enemy_pos.insert({now_x, now_y});
                }
                // 继续检查下一个旗子。
                now_x += dx[dir];
                now_y += dy[dir];
            }
            if(arrived){
                // 如果另一端有自己棋子，则把夹住的对手旗子位置都
加入被包住的位置 set
                eaten_pos.insert(enemy_pos.begin(),
enemy_pos.end());
            }
        }
        if(!eaten_pos.empty()){
            // 若有包住至少一个棋子，则(x,y)是可以摆的位置，并翻
转被夹住的棋子的颜色
            grid[x][y] = col;
            for(auto me: eaten_pos){
                rev(me.first,me.second);
            }
            return 1;
        }
        else return 0;
    }
}board;
int a[3][N * N + 1];
// (ids[x][0], ids[x][1]) 就是数字 x 对应到的棋盘座标
int ids[N * N + 1][2];
char cc[4] = ".BW";
void print(){
    for(int i = 1; i <= N; i++) {
        for(int j = 1; j <= N;
j++)putchar(cc[board.grid[i][j]]);
        puts("");
    }
}
bool can_move(int col){
    for(int i = 1; i <= N * N; i++){
        if(board.put(col, ids[a[col][i]][0], ids[a[col]
```

```
[i]][1]))return 1;
    }
    return 0;
}
int main(){
    // 计算每个数字对应的座标
    for(int x = 1; x <= 8; x++) {
        for(int y = 1; y <= 8; y++) {
            ids[8 * (x - 1) + y][0] = x;
            ids[8 * (x - 1) + y][1] = y;
        }
    }
    for(int i = 1; i <= 64; i++) {
        scanf("%d", &a[BLACK][i]);
    }
    for(int i = 1; i <= 64; i++) {
        scanf("%d", &a[WHITE][i]);
    }
    board.init();
    while(1){
        int move_cnt = 0;
        move_cnt += can_move(BLACK);
        move_cnt += can_move(WHITE);
        if(!move_cnt)break;
    }
    print();
    return 0;
}
```

problem H: 第008话 通向宝可梦擂台之路!

简化题意

有个弹力球放手后一直重复的落地又弹起来，每次从球在局部最高点到落地的时间区间内，求的高度是严格递减，并且每次从球落地到到求达到局部最高点时，球的高度是严格递增，依时序告诉你 n 个相异时间点的球的高度(这些时间点不会有恰好落地的时间)，请问求至少落地几次?

考察知识点

- 正确理解题意
- c++ 基础语法及简单模拟
- ~~大类的本质是贪心~~

解析

使用模拟即可，假设一刚开始球是处于落下的阶段，每读入一个新的高度，就判断是否球必须转向，模拟的过程中，计算有多少次转向是由落地变成

为什么会说考察知识点会提到人类的本质是贪心呢？这是因为，应该没有人会觉得若高度 $h_i > h_{i+1}$ 且在时间 t_i 时是落下的状态，在时间 t_i 至 t_{i+1} 之间球要落地一次以上答案会更理想吧 (>_<)。

若心中没有这个贪心的概念，本题也是可以用 dp 解的，这里就不赘述了。

贪心的参考代码如下：

```
#include<cstdio>
#include<limits.h>
const int INF = INT_MAX;
int main(){
    //  假定初始时球在无穷高的地方，且是朝下
    int last_h = INF;
    bool face_down = true;
    int N;
    scanf("%d", &N);
    int answer = 0;
    for(int i = 0; i < N; i++){
        int h;
        scanf("%d", &h);
        if(face_down){
            if(h >= last_h){
                face_down = false;
                answer++;
            }
        }
        else{
            if(h <= last_h){
                face_down = true;
            }
        }
        last_h = h;
    }
    printf("%d\n",answer);
    return 0;
}
```