牛客寒假营第四场题解

签到：A、B、I

easy：H

mid：C、F

hard：D、G

防AK：E、J

shit：A、B、C、D、E、F、G、H、I、J

前言

本场比赛致敬了2025年由（已删除）出题组命题的（已删除）区域赛。

本场比赛已经几乎可以确定是 6 场中最烂的一场了。

虽然还没有写最后一场，但是知道出题人是（已删除），感觉很容易猜，而第五场的出题人是（已删除），大家可以期待一下最后两场。

原本第 3 至第 10 题的题目名字为：

构造王国

（已删除）构造帝国

构造（已删除）帝国

构造及字符串（已删除）王国

构造（已删除）（已删除）国

构造（已删除）联盟

构造（已删除）国

构造（已删除）（已删除）国

但由于第二题写到的原因，已改名。

现在除了第一、二题，每一道题的主角都对应了一个人物，并且开头的笑话大部分也对应了之前的题目名字。

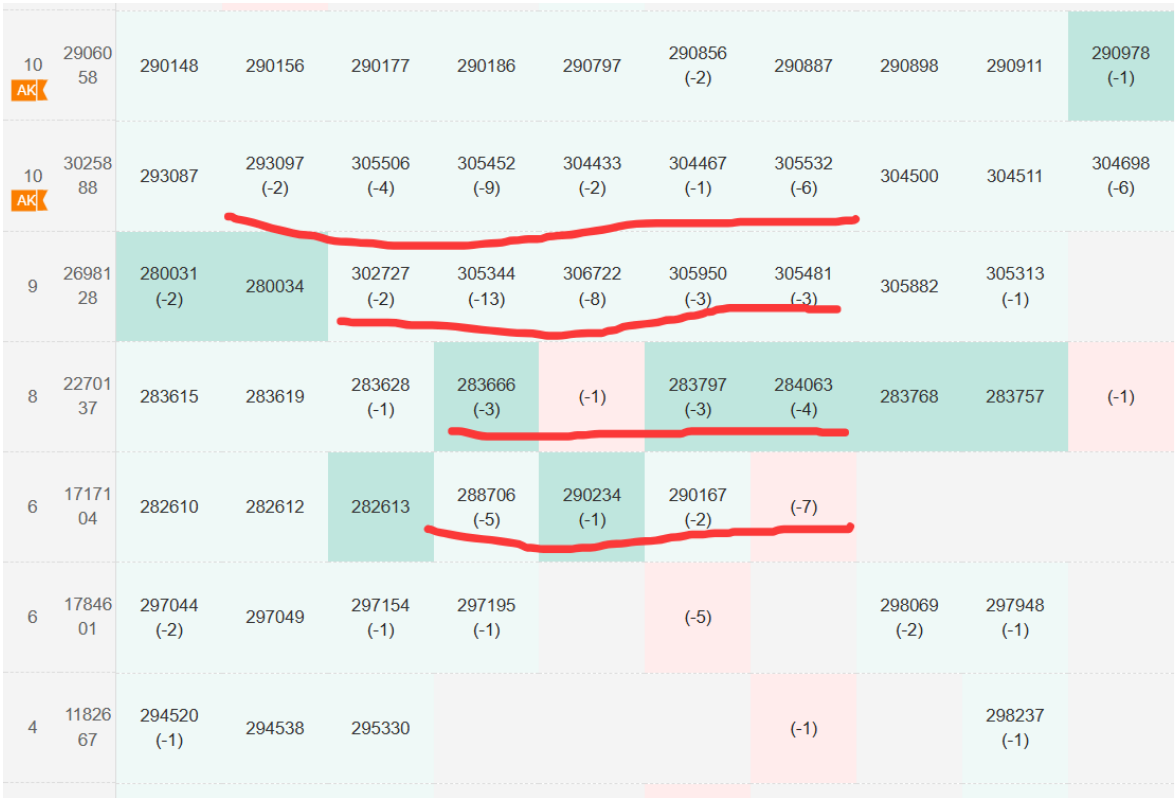
H题有几个数字彩蛋。

让我们恭喜蔡光获得冠军！

虽然表面上有 8 道构造题，但实际上真正的构造题只有 C、F 两题，剩下的都是披着构造皮的输出方案题。

大部分题都比较符合预期的顺序，不过 C 和 H 有了一点错位，似乎是大佬把榜带歪了点。

下图是内测的榜单，大家都非常痛苦，基本上总有这么几题在坐牢。



10 AK	2906058	290148	290156	290177	290186	290797	290856 (-2)	290887	290898	290911	290978 (-1)
10 AK	3025888	293087	293097 (-2)	305506 (-4)	305452 (-9)	304433 (-2)	304467 (-1)	305532 (-6)	304500	304511	304698 (-6)
9	2698128	280031 (-2)	280034	302727 (-2)	305344 (-13)	306722 (-8)	305950 (-3)	305481 (-3)	305882	305313 (-1)	
8	2270137	283615	283619	283628 (-1)	283666 (-3)	(-1)	283797 (-3)	284063 (-4)	283768	283757	(-1)
6	1717104	282610	282612	282613	288706 (-5)	290234 (-1)	290167 (-2)	(-7)			
6	1784601	297044 (-2)	297049	297154 (-1)	297195 (-1)		(-5)		298069 (-2)	297948 (-1)	
4	1182667	294520 (-1)	294538	295330				(-1)		298237 (-1)	

A 本场比赛灵感来源于树状数组出题组

签到、枚举

按题意模拟，可以直接使用双重循环判断有多少个数小于等于 a_i 。

时间复杂度： $O(n^2)$ 。

```
#include<bits/stdc++.h>

using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        int cnt = 0;
        for (int j = 1; j <= n; j++) {
            if (j == i) continue;
            if (a[j] <= a[i]) cnt++;
        }
        if (cnt * 10 >= (n - 1) * 8) sum += a[i];
    }
    cout << sum << endl;
}
```

B 构造部落

前缀和

我们可以用一个前缀和数组记录第 i 名首领在位的第 1 年是公元几年。

然后每次查询就可以直接知道第 x 名首领在位的第 1 年是公元几年，然后再加上 $y-1$ 就可以得到答案。

时间复杂度： $O(n+q)$ 。

```
#include<bits/stdc++.h>

using namespace std;

int main() {
    int n, q, s;
    cin >> n >> q >> s;
    vector<int> a(n + 1), b = a;
    b[0] = s;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        b[i] = b[i - 1] + a[i - 1];
    }
    while (q--) {
        int x, y;
        cin >> x >> y;
        cout << b[x] + y - 1 << endl;
    }
```

```
}
```

C 墨提斯的排列

构造、位运算、格雷码、分治

实际上最终的排列就是格雷码。

格雷码有一个性质是：相邻两个数字在二进制中有且仅有一位不同。

我们考虑一下排列的性质：排列相邻两个数字在二进制中至少会有 1 位不同，长度为 n 的排列至少会有 $n-1$ 位不同，答案就是 $\sum_{i=1}^{n-1} 2^t (t \in [0, n-1])$ 。

最优的想法当然是将每一个 t 都取 0 ，但很显然这是不可能的，因为这样一定会出现 $a, a \oplus 1, a$ 子区间，这样构造的就不是排列了。

如果我们首先将第 k 位为 0 的数放在前面，第 k 位为 1 的数放在后面，这样第 k 位实际上就只会有 1 次 2^k 的代价。而左右两边，在去除第 k 位之后实际上是完全一样的两个排列，也就是说在左边第 t 位至少会变化 1 次，在右边第 t 位也至少会变化 1 次，总共至少会变化 2 次。

只考虑 k, t 两位至少会产生 $2^k + 2 \times 2^t$ 的代价，很显然 k 小于 t 一定是不优的。因此首先我们把最高位分成两边是比较好的选择，在此之后就是分治的过程。

因此最后的代价应该是： $\sum_{i=0}^{n-1} 2^{n-i-1} \times 2^i = \sum_{i=0}^{n-i-1} 2^{n-1} = n \times 2^{n-1}$ 。

这个代价实际上就是答案的下界，我们需要构造一个排列使得代价尽可能接近这个下界，而格雷码的代价恰好为这个下界。

一种基于分治的构造方式，以 $n = 4$ 为例，将排列的最高位按 0 和 1 分成左右两部分，然后往下分治递归，每一位可以分成：

```
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1
0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0
0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
4 5 7 6 2 3 1 0 8 9 B A E F D C
```

另一种贪心的构造方式，首先将 0 放进排列尾部，然后每次从小到大尝试改变尾部的二进制位，如果得到的是新数就更新排列：

```
0
0 1（尝试改变0的倒数1位，得到1，成功）
0 1 3（尝试改变1的倒数1位，得到0，但0在排列中，失败；尝试改变1的倒数第2位，得到3，成功）
0 1 3 2（尝试改变3的倒数1位，得到2，成功）
0 1 3 2 6（尝试改变2的倒数1位，得到3，失败；尝试改变2的倒数第2位，得到0，失败，尝试改变2的倒数第3位，得到6，成功）
0 1 3 2 6 7 5 4
实际上这也是格雷码。
```

基于格雷码性质的构造方式：格雷码第 i 项就是 $i \oplus (i >> 1)$ 。

时间复杂度： $O(2^n)$ 。

有趣的事：这题在验题时，验题人发现是写了格雷码，我才知道我贪心构造的居然是格雷码。

```
#include<bits/stdc++.h>

using namespace std;

int main(){
    int n;
    cin >> n;
    for(int i = 0; i < 1 << n; i++){
        cout << (i ^ (i >> 1)) << " ";
    }
    cout << endl;
}
```

D 东风谷早苗与博丽灵梦

数学、扩展欧几里得（EXGCD）、二分

设早苗走了 c_1 步，灵梦走了 c_2 步，可以得出方程 $ac_1 + sc_2 = x$ 。

由于 a, s, x 我们已知，因此这是一个二元一次方程。

根据初中数学可知：一个二元一次方程有无数组解。而我们需要找到其中一组非负整数的特解，且最小化 $max(c_1, c_2)$ 。

我们可以使用扩展欧几里得算法判断这个方程是否有整数解，并求出非负整数通解的形式。

在通解中，由于 a, s, x 都是正整数，因此随着 c_1 增大， c_2 一定会减小，那么 $c_1 - c2$ 一定是**单调递增**的。

当 $c_1 - c_2$ 接近 0 时， c_1 和 c_2 最平均，也就是说可以令 $max(c_1, c_2)$ 最小。

因此我们可以二分求第一个 $c_1 - c_2 \geq 0$ 的 c_1', c_2' 和最后一个 $c_1 - c_2 \leq 0$ 的 c_1'', c_2'' ，最终答案就是 $min(max(c_1', c_2'), max(c_1'', c_2''))$ 。

注意最终的 c_1', c_2', c_1'', c_2'' 不可以为负数。

时间复杂度： $O(T \times logA)$ 。

```
#include<bits/stdc++.h>

using namespace std;

using LL = long long;

namespace Exgcd {
using LL = long long;
using LLL = __int128;

array<LLL, 4> get(LL a, LL b, LL c) {       //ax + by = c
    array<LLL, 4> ans = {1, 0, 0, 1};
    auto &[x, y, g, f] = ans;
    auto dfs = [&](auto dfs, LL a, LL b) {
        if (b == 0) {
            ans[2] = a;
            return;
        }
        dfs(dfs, b, a % b);
```

```
        ans = {ans[1], ans[0] - (a / b)* ans[1], ans[2], ans[3]};
    };
    dfs(dfs, a, b);
    if (g < 0) {
        g = -g;
        x = -x;
        y = -y;
    }
    if (c % g) f = 0;
    else {
        x *= c / g;
        y *= c / g;
    }
    return ans;
}

array<LLL, 2> minx(LL a, LL b, LL c) {           // x 的最小非负整数解
    auto [x, y, g, f] = get(a, b, c);
    if (!f) return {-1, -1};
    if (!c) return {0, g};
    auto bg = b / g;
    auto k = -x / bg;
    while (x + k * bg < 0) k += 1;
    while (x + (k - 1) * bg >= 0) k -= 1;            //似乎不太需要
    return {x + k * bg, g};
}
};

int main() {
    int T;
    cin >> T;
    while (T--) {
        LL x, a, s;
        cin >> x >> a >> s;
        auto [_, g] = Exgcd::minx(a, s, x);
        auto X = _;
        auto Y = (x - X * a) / s;
        if (X < 0 || Y < 0) {
            cout << "No" << endl;
            continue;
        }
        cout << "Yes" << endl;
        auto sg = s / g;
        __int128 l = 0, r = 1e18;
        auto check = [&](auto mid) {
            auto x1 = X + mid * sg;
            auto y1 = (x - x1 * a) / s;
            return pair(x1, y1);
        };
        while (l < r) {
            auto mid = l + r >> 1;
            auto [x1, y1] = check(mid);
            if (x1 > y1) r = mid;
            else l = mid + 1;
        }
        auto [x1, y1] = check(l);
        auto [x0, y0] = check(l - 1);
        int tar = 1;
```

```
        assert(min(x1, y1) >= 0 || min(x0, y0) >= 0);
        if (min(x1, y1) < 0) tar = 0;
        else if (min(x0, y0) < 0) tar = 1;
        else {
            if (max(x1, y1) <= max(x0, y0)) tar = 1;
            else tar = 0;
        }
        if (tar) cout << format("{} {}", x1, y1) << endl;
        else cout << format("{} {}", x0, y0) << endl;
    }
}
```

E 立希的扫雷构造

状压DP、路径还原、打表

原本这题的数据范围是 $n \times m = 10^6$ ，但是似乎大家都不会做，于是改成了状压 DP 板子。

如果本题只求一组解，我们可以使用简单的状压DP解决。

$dp_{i,j,k}$ 表示第 i 行布置雷的状态为 j ，使用了 k 个雷的最大危险值。

转移就是 $dp_{i,j,k} = max(dp_{i,j,k}, dp_{i-1,l,k-count(j)} + calc(j,l))$ ，其中 l 是上一行的状态， $count(j)$ 是 j 状态中雷的数量， $calc(j,l)$ 是 j 状态和 l 状态相邻产生的危险值之和。

其中，我们可以预处理 $count(j)$ 和 $calc(j,l)$ 加速转移的过程。并且可以发现 k 个雷产生的危险值实际上等价于将所有位置取反后 $n \times m - k$ 个雷产生的危险值（本质上就是在全是雷的地图上放空格产生的"危险值"），可以将雷数的枚举再优化一半。顺便记录一下每个状态的前驱，就可以还原方案了。

我们可以将所有询问中 n, m 相同的询问放在一起，本质上我们只要处理最大的雷数 k ，剩下所有雷数小于 k 的询问，实际上都已经计算好了。

再进行归类，将所有 m 相同的询问放在一起，处理最大的行数 n ，剩下所有行数小于 n 的询问，实际上也都已经计算好了。

因此我们实际上只需要使用 $9(m \leq 9)$ 次状压DP就可以预处理所有的答案了， 9 次状压DP的总时间复杂度是 $\sum_{t=1}^{9} \frac{t^4}{2} \times 2^{2t}$ 。

虽然时间复杂度非常抽象，看上去完全过不了，但细心的你一定可以发现，本质不同的询问实际只有 2000 多种，然后每种询问的答案不会太长，因此只要打一份包含所有答案的表交上去，这题就能过了。

下图是某验题人的代码：

```
f[9][9][71] = {80, 511, 511, 511, 351, 511, 341, 511, 341, 511};
f[9][9][72] = {72, 511, 511, 511, 383, 511, 341, 511, 341, 511};
f[9][9][73] = {64, 511, 511, 511, 511, 511, 341, 511, 341, 511};
f[9][9][74] = {56, 511, 511, 511, 511, 511, 343, 511, 341, 511};
f[9][9][75] = {48, 511, 511, 511, 511, 511, 351, 511, 341, 511};
f[9][9][76] = {40, 511, 511, 511, 511, 511, 383, 511, 341, 511};
f[9][9][77] = {32, 511, 511, 511, 511, 511, 511, 511, 341, 511};
f[9][9][78] = {24, 511, 511, 511, 511, 511, 511, 511, 343, 511};
f[9][9][79] = {16, 511, 511, 511, 511, 511, 511, 511, 351, 511};
f[9][9][80] = {8, 511, 511, 511, 511, 511, 511, 511, 383, 511};
f[9][9][81] = {0, 511, 511, 511, 511, 511, 511, 511, 511, 511};


    int t = 1;
    std::cin >> t;

    while (t--) {
        solve();
    }

    return 0;
}
```

如果你不会算时间复杂度 or 你觉得牛客是神机 or 你是剪枝战神 or 你是卡常大师，导致你非常敢写代码，直接按上面的暴力状压 DP 去写，你大概率也是可以通过的。

在完全不考虑常数的情况下，实际耗时 $< 150ms$ 。

```
int n = 9, k = n * n / 2;
vector dp(n + 1, vector(n + 1, vector(1 << n, vector<int>(k + 1, -1e9))));
vector pre(n + 1, vector(n + 1, vector(1 << n, vector<int>(k + 1, -1))));
for(int m = 1; m <= n; m++){
    vector dis(1 << m, vector<int>(1 << m));
    k = m * n / 2;
    for(int i = 0; i < 1 << m; i++){ ...
    }
    for(int i = 0; i < 1 << m; i++){ ...
    }
    for(int i = 2; i <= n; i++){
        for(int j = 0; j < 1 << m; j++){
            int cnt = __builtin_popcount(j);
            for(int l = 0; l < 1 << m; l++){
                for(int o = cnt; o <= k; o++){
                    if(dp[m][i][j][o] < dp[m][i - 1][l][o - cnt] + dis[l][j]){
                        dp[m][i][j][o] = max(dp[m][i][j][o], dp[m][i - 1][l][o - cnt] + dis[l][j]);
                        pre[m][i][j][o] = l;
                    }
                }
            }
        }
    }
}
```

时间复杂度： $O(\sum_{t=1}^{9} \frac{t^4}{2} \times 2^{2t})/O(T)$ 。

```
#include<bits/stdc++.h>

using namespace std;

int main() {
    int n = 9, k = n * n / 2;
    vector dp(n + 1, vector(n + 1, vector(1 << n, vector<int>(k + 1, -1e9))));
    vector pre(n + 1, vector(n + 1, vector(1 << n, vector<int>(k + 1, -1))));
    for(int m = 1; m <= n; m++){
        vector dis(1 << m, vector<int>(1 << m));
        k = m * n / 2;
        for(int i = 0; i < 1 << m; i++){
            for(int j = 0; j < 1 << m; j++){
                auto &t = dis[i][j];
```

```
                for(int p = 0; p < m; p++){
                    if(!(i >> p & 1)){
                        t += j >> p & 1;
                        t += j >> (p - 1) & 1;
                        t += j >> (p + 1) & 1;
                    }
                    if(!(j >> p & 1)){
                        t += i >> p & 1;
                        t += i >> (p - 1) & 1;
                        t += i >> (p + 1) & 1;
                        t += j >> (p - 1) & 1;
                        t += j >> (p + 1) & 1;
                    }
                }
            }
        }
        for(int i = 0; i < 1 << m; i++){
            int cnt = __builtin_popcount(i);
            dp[m][1][i][cnt] = 0;
            for(int p = 0; p < m; p++){
                if(!(i >> p & 1)){
                    int t = 0;
                    t += i >> (p - 1) & 1;
                    t += i >> (p + 1) & 1;
                    dp[m][1][i][cnt] += t;
                }
            }
        }
        for(int i = 2; i <= n; i++){
            for(int j = 0; j < 1 << m; j++){
                int cnt = __builtin_popcount(j);
                for(int l = 0; l < 1 << m; l++){
                    for(int o = cnt; o <= k; o++){
                        if(dp[m][i][j][o] < dp[m][i - 1][l][o - cnt] + dis[l]
[j]){
                            dp[m][i][j][o] = max(dp[m][i][j][o], dp[m][i - 1][l]
[o - cnt] + dis[l][j]);
                            pre[m][i][j][o] = l;
                        }
                    }
                }
            }
        }
    }
    int T;
    cin >> T;
    while(T--){
        int n, m, k;
        cin >> n >> m >> k;
        int flag = 0;
        if(k * 2 >= n * m){
            k = n * m - k;
            flag = 1;
        }
        pair<int, int> ans = {-1, -1};
        for(int i = 0; i < 1 << m; i++){
            ans = max(ans, pair(dp[m][n][i][k], i));
        }
```

```
        auto [x, y] = ans;
        cout << x << endl;
        for(int i = n, j = y, cnt = k; i > 0; i--){
            char a = '*', b = '.';
            if(flag) swap(a, b);
            for(int o = 0; o < m; o++){
                if(j >> o & 1) cout << a;
                else cout << b;
            }
            cout << endl;
            int t = __builtin_popcount(j);
            j = pre[m][i][j][cnt];
            cnt -= t;
        }
    }
}
```

F 爱音的01串构造

构造

如果 a, b 都不为 0，那么同时包含 0 和 1 的子串的 mex 一定为 2。

最后的权值就是 mex 为 2 的子串数量 $\times 2 +$ 全 0 子串数量，mex 为 2 的子串数量 $+$ 全 0 子串数量 $+$ 全 1 子串数量 $=$ 所有子串数量。

全 1 子串显然是没有贡献的，因此我们需要最小化全 1 子串的数量。

如果一个子串为 11110，此时全 1 子串有 $1+2+3+4=10$ 个，如果将 0 换至中间，可以使得全 1 子串降为$(1+2)\times 2=6$。

也就是说，在 1 多 0 少的时候，让全 1 子串的最大长度最小一定是优的，具体操作就是先将 1 铺好，然后将 1 平均分成 $a+1$ 块，在块与块中间插入 0。这样我们就最小化了全 1 子串的个数，此时全 0 子串的个数在此时恰好为 a。

如果 0 多 1 少呢？先将 0 铺好，然后我们要往里面填 1，怎么填最优呢？

如果我们将所有的 1 都分开填在 0 的中间，那显然此时全 1 子串的数量恰好为 b，达到了全 1 子串最小值。那全 0 子串数量 $+ mex$ 为 2 的子串数量 $=$ 所有子串数量 $- b$，而最后的权值计算中 mex 为 2 的子串数量肯定是越多越好的。

因此我们现在需要最小化全 0 子串的数量，那这样就和上述的在 1 中填 0 的方案完全一致了。

所以我们的构造方式就是，将多的字符平均分成少的字符数量 $+1$ 块，然后在块与块之间填数。

时间复杂度：$O(n)$。

```
#include<bits/stdc++.h>

using namespace std;

int main() {
    int T = 1;
    cin >> T;
    while (T--) {
        int a, b;
        cin >> a >> b;
```

```
        string t;
        if (a == b) {
            for (int i = 1; i <= a; i++) {
                t += "10";
            }
        } else {
            char c0 = '0', c1 = '1';
            if (a < b) {
                swap(a, b);
                swap(c0, c1);
            }
            int x = a / (b + 1);
            int k = a % (b + 1);
            for (int i = 1; i <= b + 1; i++) {
                t += string(x + (k > 0), c0);
                if (k > 0) k -= 1;
                t += c1;
            }
            t.pop_back();
        }
        cout << t << endl;
    }
}
```

G 真白的幻觉

数学、打表、质因数、搜索

如果数字中有 0，那很显然至多变化一次就结束了。

如果数字中有 1，实际上不会对答案造成任何影响。

如果数字中有 5，那么它后续变化中一定都有 $5/0$，且数位中一旦出现偶数，就会变 0，导致快速结束。

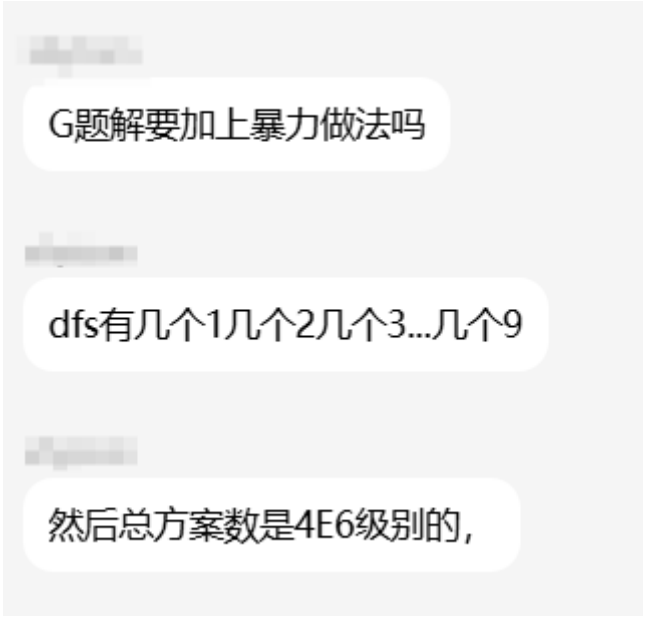
如果数字中有 4、6、8、9，实际上它们对乘积的贡献等价于 $2\times 2, 2\times 3, 2\times 2\times 2, 3\times 3$。

因此实际上一个数我们只需要看它有几个因子 2、3、7 即可，我们可以枚举这三个因子的个数，然后暴力检验变化的次数。

最后将这些因子组成数字即可，为了让数字位数最少，就是将每 3 个 2 变成一个 8，每 2 个 3 变成一个 9，多出来的 2 和 3 组成 2、3、4、6，所有的 7 都单独作为一个数字。

如果没发现 5 的性质，或者感觉 5 对答案影响不大，也可以顺便枚举因子 5 的个数，实际上对做法没有影响。

当然也有其他的做法，比如：



时间复杂度： $O(logA^3)$ 。

```
#include<bits/stdc++.h>

using namespace std;

int main() {
    cout << "27777789999999999 277777788888899" << endl;
}
```

H 时不时使使用玉米加农炮掩饰害羞的邻座艾莉同学

STL、暴力

这题比较典型，本质上就是单点修改，区间查询最大值。

可以使用 set ，每次修改前从 set 中将方格删除，修改后再将方格放进 set 。

然后每次查询在 set 中找到最大值即可。

一个更优秀的做法：由于每次只会增加敌人，所以最大值只会增大不会变小，我们可以用一个变量维护最大值，每次修改后更新最大值即可，就不需要 set 了。

不过请注意这种方式无法维护会减小的单点修改 + 区间最值。

时间复杂度： $O(n^2+q)$ 。

```
#include<bits/stdc++.h>

using namespace std;

using LL = long long;

int main() {
    int n, m, q;
    cin >> n >> m >> q;
    vector b(n + 1, vector<LL>(m + 1));
    array<LL, 3> ma = {0, 0, 0};
    for (int i = 1; i <= n; i++) {
```

```
        for (int j = 1; j <= m; j++) {
            int t;
            cin >> t;
            for (int dx = -2; dx <= 2; dx++) {
                int k = 2 - abs(dx);
                for (int dy = -k; dy <= k; dy++) {
                    int x = i + dx;
                    int y = j + dy;
                    if (x < 1 || x > n || y < 1 || y > m) continue;
                    b[x][y] += t;
                    ma = max(ma, {b[x][y], x, y});
                }
            }
        }
    }
    while (q--) {
        int x, y, z;
        cin >> x >> y >> z;
        for (int dx = -2; dx <= 2; dx++) {
            int k = 2 - abs(dx);
            for (int dy = -k; dy <= k; dy++) {
                int xx = x + dx;
                int yy = y + dy;
                if (xx < 1 || xx > n || yy < 1 || yy > m) continue;
                b[xx][yy] += z;
                ma = max(ma, {b[xx][yy], xx, yy});
            }
        }
        cout << format("{} {}", ma[1], ma[2]) << endl;
    }
}
```

I 初华的扭蛋机

概率、期望、诈骗

戒赌题！

如果我们下注 1 枚筹码，那么中 1 个小球的概率是：$\frac{1\times5^2\times3}{6^3}=\frac{75}{216}$。

如果我们下注 1 枚筹码，那么中 2 个小球的概率是：$\frac{1^2\times5\times3}{6^3}=\frac{15}{216}$。

如果我们下注 1 枚筹码，那么中 3 个小球的概率是：$\frac{1^3}{6^3}=\frac{1}{216}$。

期望的收益是 $1-\frac{75\times2+15\times3+1\times10}{6^3}=1-\frac{205}{216}=-\frac{11}{216}$。

也就是说我们在某种颜色上下注 1 枚筹码，期望是负的。而如果在多种颜色的筹码下注的话，必然会导致事件冲突（即不可能同时产生中两种颜色各两个小球的事件）也就是说期望会进一步降低。

因此不赌的期望收益为 0，大于 $-\frac{11}{216}$，因此所有筹码都不下注是最优解。

时间复杂度：$O(1)$。

```cpp
#include<bits/stdc++.h>

using namespace std;

int main() {
    cout << "######" << endl;
}
```

J 立希坐地铁

最短路、路径还原、模拟

很显然只有换乘站、起点、终点站有意义，因此建图时只需要考虑这些站就可以了。

同一条地铁路线的点两两可达，因此可以对同一条线路的点两两连边，边权为两点距离即可。

但这样如果一条地铁路线上有很多点，会导致边数太多从而超时。而实际上由于路线是连续的，因此我们只需要将一条线路中相邻的点连边就可以了。

那不同线路上的同一个点怎么办呢？我们可以将 1 个点拆成 3 个点，分别代表不同的线路，然后将这 3 个点两两连边，表示换乘。

接下来只需要跑一边单源最短路就可以得到最短的距离了。

然后如何得到方案呢？类似 E 题，每次转移时，我们记录前驱即可。然后由于我们是对相邻的点进行连边，构造出来的方案可能会有同站同线路换乘，因此我们还需要对构造出来的方案进行一些优化。

时间复杂度：$O(n \times logn)$ 。

有趣的事：本题数据中，输出的构造方式是错的，似乎把南北写反了（哈基米南北路多），但是并不会影响 SPJ 的判题，因此也没改数据。

```cpp
#include<bits/stdc++.h>

using namespace std;

int main() {
    int n, m;
    cin >> n >> m;
    vector vx(n + 1, vector<int>()), vy = vx;
    vector vr(n + 1, vector<pair<int, int>>());
    map<pair<int, int>, int> id;
    vector<pair<int, int>> node(m + 3);
    for (int i = 1; i <= m + 2; i++) {
        int u, v;
        cin >> u >> v;
        if (i > 2 && pair(u, v) == node[1]) continue;
        if (i > 2 && pair(u, v) == node[2]) continue;
        vx[u].push_back(v);
        vy[v].push_back(u);
        int r = min({u, v, n - u + 1, n - v + 1});
        vr[r].push_back({u, v});
        id[ {u, v}] = i;
        node[i] = {u, v};
    }
    int sz = (m + 3) * 3;
```

```
    vector ve(sz, vector<pair<int, int>>());
    for (int i = 1; i <= m + 2; i++) {
        ve[i * 3].push_back({i * 3 + 1, 1});
        ve[i * 3].push_back({i * 3 + 2, 1});
        ve[i * 3 + 1].push_back({i * 3, 1});
        ve[i * 3 + 1].push_back({i * 3 + 2, 1});
        ve[i * 3 + 2].push_back({i * 3, 1});
        ve[i * 3 + 2].push_back({i * 3 + 1, 1});
    }
    auto get = [&](auto pr) {
        auto &[x, y] = pr;
        int l = min({x, y, n - x + 1, n - y + 1});
        int ans = 0;
        int r = n - l + 1;
        int len = r - l;
        if (x == l) ans += y - l;
        else if (x == r) ans += len * 2 + r - y;
        else {
            if (y == r) ans += len + x - l;
            else ans += len * 3 + r - x;
        }
        return ans;
    };
    auto link = [&](int x1, int y1, int x2, int y2, int t) {
        int u = id[ {x1, y1}] * 3 + t;
        int v = id[ {x2, y2}] * 3 + t;
        int d = 1e9;
        if (t != 2) d = (abs(x1 - x2) + abs(y1 - y2)) * 2;
        else {
            auto d0 = get(pair(x1, y1));
            auto d1 = get(pair(x2, y2));
            int len = min({x1, y1, n - x1 + 1, n - y1 + 1});
            int M = (n - len + 1 - len) * 4;
            d = min((d1 - d0 + M) % M, (d0 - d1 + M) % M) * 2;
            ve[u].push_back({v, d});
            ve[v].push_back({u, d});
        }
        ve[u].push_back({v, d});
        ve[v].push_back({u, d});
    };
    for (int i = 1; i <= n; i++) {
        sort(vx[i].begin(), vx[i].end());
        sort(vy[i].begin(), vy[i].end());
        sort(vr[i].begin(), vr[i].end(), [&](auto & l, auto & r) {
            return get(l) < get(r);
        });
        for (int j = 0; j < vx[i].size(); j++) {
            if (j > 0) link(i, vx[i][j], i, vx[i][j - 1], 0);
            if (j + 1 < vx[i].size()) link(i, vx[i][j], i, vx[i][j + 1], 0);
        }
        for (int j = 0; j < vy[i].size(); j++) {
            if (j > 0) link(vy[i][j], i, vy[i][j - 1], i, 1);
            if (j + 1 < vy[i].size()) link(vy[i][j], i, vy[i][j + 1], i, 1);
        }
        if (vr[i].size() > 1) {
            auto& v = vr[i];
            int M = v.size();
            for (int j = 0; j < M; j++) {
```

```
            int f = (j - 1 + M) % M;
            link(v[j].first, v[j].second, v[f].first, v[f].second, 2);
            int b = (j + 1) % M;
            link(v[j].first, v[j].second, v[b].first, v[b].second, 2);
        }
    }
}
vector<int> dis(sz, 1e9), vis(sz), pre(sz);
dis[3] = 0;
dis[4] = 0;
dis[5] = 0;
set<pair<int, int>> st;
st.insert({0, 3});
st.insert({0, 4});
st.insert({0, 5});
while (st.size()) {
    auto [d, u] = *st.begin();
    st.erase(st.begin());
    if (vis[u]) continue;
    vis[u] = 1;
    for (auto &[v, w] : ve[u]) {
        if (dis[v] <= d + w) continue;
        dis[v] = d + w;
        st.insert({dis[v], v});
        pre[v] = u;
    }
}
pair<int, int> ans = {1e9, 1e9};
auto &[x, y] = ans;
ans = min(ans, {dis[6], 6});
ans = min(ans, {dis[7], 7});
ans = min(ans, {dis[8], 8});
if (x >= 9e8) cout << "Impossible!" << endl;
else {
    cout << x << endl;
    vector<array<int, 3>> v;
    for (int i = y; i; i = pre[i]) {
        v.push_back({node[i / 3].first, node[i / 3].second, i % 3});
    }
    reverse(v.begin(), v.end());
    int fx = 0, fy = 0;
    vector<pair<int, int>> ans;
    string s;
    for (auto &[x, y, t] : v) {
        ans.push_back({x, y});
        if (pair(x, y) != node[2]) s += "RCO"[t];
        while (s.size() >= 2 && s.back() == s[s.size() - 2]) {
            s.pop_back();
            ans.pop_back();
        }
        fx = x;
        fy = y;
    }
    cout << s.size() << endl;
    for (int i = 0; i < ans.size(); i++) {
        cout << ans[i].first << " " << ans[i].second << endl;
        if (i + 1 < ans.size()) {
            char ch = '#';
```

```
                if (s[i] == 'R') {
                    if (ans[i + 1].second > ans[i].second) ch = 'E';
                    else ch = 'W';
                }
                if (s[i] == 'C') {
                    if (ans[i + 1].first > ans[i].first) ch = 'S';
                    else ch = 'N';
                }
                if (s[i] == 'O') {
                    int d0 = get(ans[i]);
                    int d1 = get(ans[i + 1]);
                    int in = min({ans[i].first, ans[i].second,
                                  n - ans[i].first + 1, n - ans[i].second + 1});
                    int len = n - in + 1 - in;
                    if (d1 > d0) {
                        if (d1 - d0 <= len * 2) ch = 'C';
                        else ch = 'I';
                    } else {
                        if (d0 - d1 <= len * 2) ch = 'I';
                        else ch = 'C';
                    }
                }
                cout << ch << endl;
            }
        }
    }
}
```