A. 一卡通

签到题，模拟即可。

B. Royale Bataille

对于每一行，每一列，所有出现的数字都必须是连续个 1，连续个 0，连续个 2 的模式，否则答案就是 0。

接下来，考虑存在填充方案的情况。

定义 $dp_{i,j,k}$ 代表到第 i 行，0 的最左侧取到 j，最右侧取到 k 的方案数。

那么对于每一行来说，必须满足 $j >$ [最左侧出现的非0数字] 且 $k <$ [最右侧出现的非0数字]。

根据题目的定义，只有在 $j \le j'$ 且 $k \le k'$ 的情况下才能从 $dp_{i-1,j',k'}$ 转移到 $dp_{i,j,k}$，转移的方式很简单，就是 $dp_{i,j,k} += dp_{i-1,j',k'}$。

此时暴力的转移应该是 $O(nm^4)$的，但是发现在遍历到 $dp_{i-1,j',k'}$ 时候可以使用二维前缀和的方法，在下一列所有满足 $j \le j'$ 且 $k \le k'$ 的位置加上 $dp_{i-1,j',k'}$，这样遍历到下一列的时候可以快速转移，时间复杂度 $O(nm^2)$。

发现取矩阵转置进行 dp 对于结果没有影响，所以当 $m > n$ 时候取转置进行相同 dp 即可，最终时间复杂度 $O(nm\min(n,m))$。

```
void solve() {
    int n, m;
    cin >> n >> m;
    vector<string> a(n);
    for (int i = 0; i < n; i++) cin >> a[i];

    const int mod = 998244353;

    auto process = [&](vector<tuple<int, int, int, int, int, int>> lim, int n, int m) -> int {
        int ans = 0;
        auto check = [&](int s, int t, int l, int r) -> int {
            return l <= s && r <= t && s <= r;
        };
        int flag1 = 1;
        int flag2 = 1;
        int last = 0;
        for (int i = 0; i < lim.size(); i++) {
            auto [left, right, left_inner, right_inner, all_r, all_g] = lim[i];
            if (!all_g) last = i + 1;
        }
        vector<vector<int>> f(m, vector<int>(m));
        for (int i = 0; i < lim.size(); i++) {
            auto [left, right, left_inner, right_inner, all_r, all_g] = lim[i];

            vector<vector<int>> tf(m, vector<int>(m));
            for (int l1 = 0; l1 < m; l1++) {
                for (int r1 = l1; r1 < m; r1++) {
                    //l1 <= r2 <= r1
```

```
                    //l2 <= l1
                    (tf[l1][r1] += f[l1][r1]) %= mod;
                    if (l1) (tf[l1][l1 - 1] -= f[l1][r1]) %= mod;
                }
            }
            for (int l = m - 1; l >= 0; l--) {
                for (int r = m - 1; r >= 0; r--) {
                    if (l + 1 < m) (tf[l][r] += tf[l + 1][r]) %= mod;
                    if (r + 1 < m) (tf[l][r] += tf[l][r + 1]) %= mod;
                    if (max(l, r) + 1 < m) (tf[l][r] -= tf[l + 1][r + 1]) %= mod;
                }
            }
            for (int l = 0; l < m; l++) {
                for (int r = l; r < m; r++) {
                    if (left <= l && l <= left_inner && right_inner <= r && r <= right) {

                    } else {
                        tf[l][r] = 0;
                    }
                }
            }
            if (flag1) {
                for (int l = left; l <= left_inner; l++) {
                    for (int r = max(l, right_inner); r <= right; r++) {
                        if (i && r + 1 < m) continue;
                        tf[l][r]++;
                        tf[l][r] %= mod;
                    }
                }
            }
            if (i + 1 >= last) {
                for (int l = 0; l < m; l++) {
                    for (int r = l; r < m; r++) {
                        if (i + 1 < n && l) continue;
                        ans += tf[l][r];
                        ans %= mod;
                    }
                }
            }
            flag1 &= all_r;
            flag2 &= all_g;
            f.swap(tf);
        }
        //out2(flag1, flag2);
        ans += flag1 + flag2;
        ans %= mod;
        ans += mod;
        ans %= mod;
        return ans;
    };
    auto solve_row = [&]() -> void {
        vector<tuple<int, int, int, int, int, int>> ans;
        for (int i = 0; i < m; i++) {
            int left = 0, right = n - 1;
            int left_inner = n - 1, right_inner = 0;
            int flag = 0;
```

```
            for (int j = 0; j < n; j++) {
                if (a[j][i] != '?') {
                    if (a[j][i] == '0') {
                        if (!flag) {
                            left_inner = right_inner = j;
                        } else {
                            right_inner = j;
                        }
                        flag = 1;
                    } else if (a[j][i] == '1') {
                        left = max(left, j + 1);
                    } else if (a[j][i] == '2') {
                        right = min(right, j - 1);
                    }
                }
            }
            ans.emplace_back(left, right, left_inner, right_inner, !flag && right == n - 1, !flag && left == 0);
        }
        cout << process(ans, m, n) << '\n';
    };
    auto solve_col = [&]() -> void {
        vector<tuple<int, int, int, int, int, int>> ans;
        for (int i = 0; i < n; i++) {
            int left = 0, right = m - 1;
            int left_inner = m - 1, right_inner = 0;
            int flag = 0;
            for (int j = 0; j < m; j++) {
                if (a[i][j] != '?') {
                    if (a[i][j] == '0') {
                        if (!flag) {
                            left_inner = right_inner = j;
                        } else {
                            right_inner = j;
                        }
                        flag = 1;
                    } else if (a[i][j] == '1') {
                        left = max(left, j + 1);
                    } else if (a[i][j] == '2') {
                        right = min(right, j - 1);
                    }
                }
            }
            ans.emplace_back(left, right, left_inner, right_inner, !flag && right == m - 1, !flag && left == 0);
        }
        cout << process(ans, n, m) << '\n';
    };
    if (m <= n) solve_col();
    else solve_row();
}
```

C,D. Kings Game

考虑没一个数字（每一堆石头的）sg函数，每个数字的 sg函数的结果是它的质因子的个数。

我们可以这样考虑，随便取 a_i 的一个因子 x 并且除以它，那么就相当于选择 a_i 的某一个质因子的组合并且使得 a_i 一个一个地除以质因子。假如我们一共选择了 k 个质因子，那么 a_i 被除的次数就减少了 k 次，就相当于从一堆石头中取出了 k 个石头。

然后注意题面中，**此时无法操作的国王赢得游戏的胜利**，说明谁进行最后一次操作谁就输，这是一个 anti-nim 游戏。anti-nim 游戏先手胜利的条件是所有数字 sg 值不全为 1 且异或和不为 0，或者所有数字的 sg 函数值全为 1 且异或和为 0。

但是，需要额外的注意的是本题中有 $sg=0$ 的数字且要求必须是非空的子序列。

那么，假设 $sg=0$ 的数字有 b 个，满足非空且 anti-nim 先手胜利的方法有 x 个,那么最后的答案就是 $2^b\times x+2^b-1$ 。

在简单版中每个询问直接暴力 dp 即可，时间复杂度 $O(32nq)$。

在困难版本中需要用到线性基去计算数字异或和为 0 的方案数，答案是 2^{x-rank}，$rank$ 是线性基中的数字数量，可以看作是除了线性基中的数字，外面的数字随便组合得到的异或值都可以用线性基中的数字组合出来。

那么对于每个询问，使用线性基并且容斥一下，时间复杂度 $O(32n+32\times\log(32)\times q)$ 或者离线也可以做到 $O(32n+(n+q)\times\log 32)$。

```
template <const int T>
struct ModInt
{
    const static int mod = T;
    int x;
    ModInt(int x = 0) : x(x % mod) {}
    ModInt(long long x) : x(int(x % mod)) {}
    int val()
    {
        return x;
    }
    ModInt operator+(const ModInt &a) const
    {
        int x0 = x + a.x;
        return ModInt(x0 < mod ? x0 : x0 - mod);
    }
    ModInt operator-(const ModInt &a) const
    {
        int x0 = x - a.x;
        return ModInt(x0 < 0 ? x0 + mod : x0);
    }
    ModInt operator*(const ModInt &a) const
    {
        return ModInt(1LL * x * a.x % mod);
    }
    ModInt operator/(const ModInt &a) const
    {
        return *this * a.inv();
    }
    bool operator==(const ModInt &a) const
    {
        return x == a.x;
    };
```

```
    bool operator!=(const ModInt &a) const
    {
        return x != a.x;
    };
    void operator+=(const ModInt &a)
    {
        x += a.x;
        if (x >= mod)
            x -= mod;
    }
    void operator-=(const ModInt &a)
    {
        x -= a.x;
        if (x < 0)
            x += mod;
    }
    void operator*=(const ModInt &a)
    {
        x = 1LL * x * a.x % mod;
    }
    void operator/=(const ModInt &a)
    {
        *this = *this / a;
    }
    friend ModInt operator+(int y, const ModInt &a)
    {
        int x0 = y + a.x;
        return ModInt(x0 < mod ? x0 : x0 - mod);
    }
    friend ModInt operator-(int y, const ModInt &a)
    {
        int x0 = y - a.x;
        return ModInt(x0 < 0 ? x0 + mod : x0);
    }
    friend ModInt operator*(int y, const ModInt &a)
    {
        return ModInt(1LL * y * a.x % mod);
    }
    friend ModInt operator/(int y, const ModInt &a)
    {
        return ModInt(y) / a;
    }
    friend ostream &operator<<(ostream &os, const ModInt &a)
    {
        return os << a.x;
    }
    friend istream &operator>>(istream &is, ModInt &t)
    {
        return is >> t.x;
    }
    ModInt pow(int64_t n) const
    {
        ModInt res(1), mul(x);
        while (n)
        {
            if (n & 1)
```

```
                res *= mul;

            mul *= mul;
            n >>= 1;
        }
        return res;
    }
    ModInt inv() const
    {
        int a = x, b = mod, u = 1, v = 0;
        while (b)
        {
            int t = a / b;
            a -= t * b;
            swap(a, b);
            u -= t * v;
            swap(u, v);
        }
        if (u < 0)
            u += mod;
        return u;
    }
};

vector<bool> isnotprime;
vector<int> primes, mindiv;
void getprime(int n)
{
    isnotprime.resize(n + 1, 0), mindiv.resize(n + 1, 1);
    isnotprime[1] = 1;
    for (int i = 2; i <= n; i++)
    {
        if (isnotprime[i] == 0)
        {
            primes.push_back(i);
            mindiv[i] = i;
        }

        for (size_t j = 0; j < primes.size() && i * primes[j] <= n; j++)
        {
            isnotprime[i * primes[j]] = 1, mindiv[i * primes[j]] = primes[j];
            if (i % primes[j] == 0)
                break;
        }
    }
}

class LinearBasis
{
    static constexpr int K = 5;
    std::array<int, K> a, t;

public:
    LinearBasis() : a{}, t{} {}

    // Insert vector x at time i.
```

```
    void insert(int x, int i)
    {
        for (int k = K - 1; ~k && x; --k)
        {
            if (((x >> k) & 1))
            {
                if (i > t[k])
                {
                    std::swap(a[k], x);
                    std::swap(t[k], i);
                }
                x ^= a[k];
            }
        }
    }

    // Find rank of LinearBasis
    int query(int i) const
    {
        int res = 0;
        for (int k = K - 1; ~k; --k)
        {
            if (t[k] >= i)
            {
                ++res;
            }
        }
        return res;
    }
};

const int mod = 998'244'353;
using Z = ModInt<mod>;
Z pow2[int(1e6 + 1)];

void pre()
{
    pow2[0] = 1;
    for (int i = 1; i <= 1e6; i++)
        pow2[i] = pow2[i - 1] * 2;
    getprime(1e7);
}

void solve()
{
    int n, q;
    cin >> n >> q;
    vector<int> a(n + 1), sg(n + 1);
    vector<array<int, 30>> pre(n + 1);
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = 1; i <= n; i++)
    {
        int x = a[i];
        while (x > 1)
        {
```

```
            x /= mindiv[x];
            sg[i]++;
        }
        pre[i][sg[i]]++;
        for (int j = 0; j < 30; j++)
            pre[i][j] += pre[i - 1][j];
    }
    vector<array<int, 2>> qur(q + 1);
    vector<int> idx(q + 1);
    for (int _ = 1; _ <= q; _++)
    {
        int l, r;
        cin >> l >> r;
        qur[_] = {l, r};
    }
    iota(notall(idx), 1);
    sort(notall(idx), [&](auto x, auto y)
         { return qur[x][1] < qur[y][1]; });
    LinearBasis lib;
    vector<Z> ans(q + 1);
    for (int i = 1, j = 1; i <= n; i++)
    {
        if (sg[i])
            lib.insert(sg[i], i);
        for (; j <= q && qur[idx[j]][1] == i; ++j)
        {
            int rank = lib.query(qur[idx[j]][0]);
            int sum0 = pre[qur[idx[j]][1]][0] - pre[qur[idx[j]][0] - 1][0];
            int sumnot0 = qur[idx[j]][1] - qur[idx[j]][0] + 1 - sum0;
            int sum1 = pre[qur[idx[j]][1]][1] - pre[qur[idx[j]][0] - 1][1];
            // 求失败且的方案数
            Z wayszero = pow2[sumnot0 - rank];
            if (sum1 == 0)
                wayszero -= 1;
            else
                wayszero -= pow2[sum1 - 1];
            Z waysone = sum1 == 0 ? Z(0) : pow2[sum1 - 1];
            Z failway = (wayszero + waysone) * pow2[sum0] + 1;
            Z success = pow2[sum0 + sumnot0] - failway;
            ans[idx[j]] = success;
        }
    }
    for (int i = 1; i <= q; i++)
        cout << ans[i] << '\n';
}
```

E. 十六行诗

贪心

因为 AABB，ABBA，ABAB 涵盖了所有出现两个数字的情况，所以一组句子能变成一首诗的充分必要条件是**至少存在两个数，使得这两个数均出现了两次及以上；或者有一个数出现了四次及以上**。

贪心的从前往后找，找到一组满足的就立刻加一组即可。

动态规划

不难发现如果一个前缀可以构成一首诗，那么会立刻断开。我的做法是记录 `m[v][stats]` 表示关键数字是 v，给 AABB，ABBA，ABAB 的所有转移状态编号为 $stats$ 的所有选择方案，方案总数是 $O(n^2)$ 的，转移是手写的每两种状态的转移，比较恶心。由于和 01 背包一样的问题，需要先转移填的数比较多的。

我的动态规划比较恶心，应该还有更好的动态规划解法，可以问一下场上做出这道题的其它选手。

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

using namespace std;

const int N = 10005;

int a[N];

int solve(int x, int n) {
    // start(Core 0) -> 0(Core A)/1(Core 0)====none -> A
    // 0(Core A) -> 2(Core 0)====A -> AA
    // 1(Core 0) -> 3(Core A)/4(Core B)====A -> AB/AB
    // 2(Core 0) -> 5(Core B)====AA -> AAB
    // 3(Core A) -> 6(Core B)====AB -> ABA
    // 4(Core B) -> 7(Core A)====AB -> ABB
    // 5(Core B) -> end====AAB -> AABB
    // 6(Core B) -> end====ABA -> ABAB
    // 7(Core A) -> end====ABB -> ABBA

    // pair<int, int>(stage_id, value_index) -> pair<int, int>(A, B);
    map<pair<int, int>, vector<pair<int, int>>> status;
    for (int i = x; i <= n; i++) {
        for (auto [A, B] : status[{ 7, a[i] }]) {  // 7 -> end
            return solve(i + 1, n) + 1;
        }
        for (auto [A, B] : status[{ 6, a[i] }]) {  // 6 -> end
            return solve(i + 1, n) + 1;
        }
        for (auto [A, B] : status[{ 5, a[i] }]) {  // 5 -> end
            return solve(i + 1, n) + 1;
        }
        for (auto [A, B] : status[{ 4, a[i] }]) {  // 4 -> 7
            status[{ 7, A }].push_back({ A, B });
        }
```

```
        for (auto [A, B] : status[{ 3, a[i] }]) {  // 3 -> 6
            status[{ 6, B }].push_back({ A, B });
        }
        for (auto [A, B] : status[{ 2, 0 }]) {  // 2 -> 5
            status[{ 5, a[i] }].push_back({ A, a[i] });
        }
        for (auto [A, B] : status[{ 1, 0 }]) {           // 1 -> *
            status[{ 3, A }].push_back({ A, a[i] });     // 1 -> 3
            status[{ 4, a[i] }].push_back({ A, a[i] });  // 1 -> 4
        }
        for (auto [A, B] : status[{ 0, a[i] }]) {  // 0 -> 2
            status[{ 2, 0 }].push_back({ A, 0 });
        }
        status[{ 1, 0 }].push_back({ a[i], 0 });     // start -> 1
        status[{ 0, a[i] }].push_back({ a[i], 0 });  // start -> 0
    }
    return 0;
}

/*
重新强调状态和阶段
*/

int main() {
    int n, k;
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    cout << solve(1, n) << '\n';
    return 0;
}

```

F. Yuhina City

首先使用最短路算法求出每个点辐射的最大值，然后对于每个询问二分回答。

假设当前二分的答案是 x，那么所有辐射值大于 x 的点就是黑色，其他点为白色。

当前答案正确的条件就是存在一条 u、v 之间的路径，经过的黑点的数量小于等于 k 个，使用 $0/1$最短路求一下 u,v 间最短路即可。

最终时间复杂度 $O(n+m\log m+q\times\log 10^{14}\times(n+m))$。

本题困难版，如何在 $q=10^5$ 的条件下解决问题，使用克鲁斯卡尔重构树和主席树解决问题。

```
void solve()
{
    ll n, m, q;
    cin >> n >> m >> q;
    vector<ll> r(n + 1);
    for (int i = 1; i <= n; i++)
        cin >> r[i];
    vector<vector<array<ll, 2>>> e(n + 1);
```

```cpp
    for (int i = 1; i <= m; i++)
    {
        ll u, v, w;
        cin >> u >> v >> w;
        e[u].push_back({v, w}), e[v].push_back({u, w});
    }
    {
        // dj
        priority_queue<array<ll, 2>> pq;
        for (int i = 1; i <= n; i++)
            pq.push({r[i], i});
        while (pq.size())
        {
            auto [d, u] = pq.top();
            pq.pop();
            if (r[u] > d)
                continue;
            for (auto [v, w] : e[u])
            {
                if (r[v] < d - w)
                {
                    r[v] = d - w;
                    pq.push({r[v], v});
                }
            }
        }
    }
    for (int _ = 1; _ <= q; _++)
    {
        ll u, v, k;
        cin >> u >> v >> k;
        ll lans = 0, rans = 1e14, ans = 1e14;
        auto check = [&](ll x)
        {
            vector<ll> color(n + 1);
            for (int i = 1; i <= n; i++)
                color[i] = (r[i] > x);
            vector<ll> d(n + 1, ll_inf), vis(n + 1);
            d[u] = color[u];
            deque<ll> que;
            que.push_back(u);
            while (que.size())
            {
                auto u = que.front();
                que.pop_front();
                if (vis[u])
                    continue;
                vis[u] = 1;
                for (auto [v, _] : e[u])
                {
                    if (d[v] > d[u] + color[v])
                    {
                        d[v] = d[u] + color[v];
                        color[v] == 0 ? que.push_front(v) : que.push_back(v);
                    }
                }
```

```
            }
            return d[v] <= k;
        };
        while (lans <= rans)
        {
            ll mid = (lans + rans) >> 1;
            if (check(mid))
                ans = mid, rans = mid - 1;
            else
                lans = mid + 1;
        }
        writeln(ans);
    }
}
```

G. Syl 和序列操作

答案的一个下界是序列的极差，即

$$\max_{i=1}^{n} a_i - \min_{i=1}^{n} a_i$$

因为一次操作最多要么让最大值减小 1，要么让最小值增大 1，而最终最小值和最大值必须相同，所以有上述的下界。下面我们考虑能否取到这个下界。

1. $a_1 \le a_n$

   最大值在 a_1 的右边，我们先对 $i = 1$ 进行 $\max - a_1$ 次操作二，这样最大值就变成 a_1，并且所有比 a_1 大的数都变成了 a_1，包括 a_n。然后对 $i = n$ 做 $a_1 - \min$ 次操作三，就能让所有数变成 a_1。这种情况能取到极差的下界。

2. $a_1 > a_n$

   最终 a_1 和 a_n 必然相等，但注意到只有操作一能让 a_1 变小和让 a_n 变大，所以至少要花 $a_1 - a_n$ 次操作让 a_1 和 a_n 相等。

   设 $A = \max_{i=2}^{n-1} a_i$, $B = \min_{i=2}^{n-1} a_i$ 分别为除了 a_1 和 a_n 以外的最大值和最小值。

   如果 $A \le a_n$，即中间所有数都不比 a_n 大，那肯定是要把 a_1 减小到 a_n 去，然后再花 $a_n - B$ 次操作二让所有数变成 a_n。这个讨论对 $B \ge a_1$ 也成立。这两种情况下都能取到极差下界。

   否则有 $A > a_n$ 且 $B < a_1$，那么区间 $[B, A]$ 与 区间 $[a_n, a_1]$ 有交集，显然最优解一定不会是让所有数变成这两个区间的交集之外的某个数。从交集中随便取一个数 C，$C \in [B, A]$ 且 $C \in [a_n, a_1]$，将所有数变成 C 的代价为 $a_1 - a_n + A - B$，此即最小代价，比极差下界大出交集的长度。

```
void solve() {
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) cin >> a[i];
    if (n <= 2) {
        cout << (n <= 1 ? 0 : abs(a[0] - a[1])) << endl;
        return;
    }
    ll ans = 0;
    auto [dn, up] = tie(*min_element(all(a)), *max_element(all(a)));
    ans = up - dn;
    if (a[0] > a[n - 1]) {
```

```
        auto [u, v] = tie(*min_element(a.begin() + 1, a.end() - 1), *max_element(a.begin() + 1, a.end() - 1));
        if (v >= a[n - 1] && u <= a[0])
            ans = (ll)a[0] - a[n - 1] + v - u;
    }
    cout << ans << endl;
}
```

H. Syl 和美丽树

一个直觉是至多的限制比至少的限制好满足。至少是这样的，至多限制的点只要无脑往 1 连就可以，可是至少的限制要考虑的事情就很多了，你需要保证存在一条从 1 开始的足够长的链，然后把至少的点往上面连才行。

一个显然的想法是，把点一个一个贪心地往 1 所在的树上去连，所有现在准备去连边的点都要尽可能使得连上去后 1 能到的最远的点的距离（记为 D）增加，这样才能让之后有着至少限制的点更容易连上去。

记每个点 v 的限制为至少 l_v，至多 r_v，如果 $l_v > r_v$ 当然无解。

假设你现在已经构造出来从 1 到 u 长度为 D 的链，现在能使得增加一条 (u, v) 的边后 D 增加 1 的点 v 需要满足的条件是

- v 没被用过
- $l_v \le D + 1 \le r_v$

如果有这样的点，选择 r_v 最小的点 v 去完成增加 D 的任务，这样 r_v 更大的点能够去增加更大的 D。不断重复增加 D 的操作，直到 D 不再能增加为止。

如果没有这样的点，那么 D 不再能够增加，这时如果存在 $l_w > D$ 的点 w，那么 w 不能被加入树里面，该情况无解。否则只需要将所有剩下的点 w 都连在我们构造的链上符合限制条件的任意一点上即可，这总是可行的，因为此时所有没用过的点 w 都满足 $r_w \le D$。

```
const int N = 2e5 + 3;
#define pii pair<int, int>
void solve() {
    int n, m;
    cin >> n >> m;
    vector<int> l(n + 1, 1), r(n + 1, n - 1);
    vector<vector<int>> S(n);
    while (m--) {
        int v, x, p;
        cin >> v >> x >> p;
        if (p == 0)
            r[v] = min(r[v], x);
        else
            l[v] = max(l[v], x);
    }
    for (int i = 2; i <= n; i++) {
        if (l[i] > r[i]) {
            cout << "Ugly\n";
            return;
        }
        S[l[i]].push_back(i);
    }
```

```
    vector<pii> e;
    priority_queue<pii, vector<pii>, greater<pii>> Q;
    vector<int> w(n, 0);
    w[0] = 1;
    for (int i = 1; i < n; i++) {
        for (auto v : S[i]) {
            Q.push({ r[v], v });
        }
        int rv, v;
        while (Q.size()) {
            tie(rv, v) = Q.top();
            if (rv >= i)
                break;
            Q.pop();
            e.emplace_back(w[rv - 1], v);
        }
        if (Q.empty())
            break;
        Q.pop();
        w[i] = v;
        e.emplace_back(w[i - 1], w[i]);
    }
    if (e.size() != n - 1) {
        cout << "Ugly\n";
        return;
    }
    cout << "Beautiful\n";
    for (auto [u, v] : e) {
        cout << u << " " << v << endl;
    }
}
```

I. So Far Away

Lemma1：只有值最小的 $k+1$ 个点产生实际作用。

Proof1：假如选定了一个点集 $a_u, a_v, a_1, a_2, \ldots, a_{k+1}$，除了之外选了一个点 a_i，在计算点集合内的任意两点 u, v 间的最短路时，**一定不会**出现边 (u, a_i) 和 (v, a_i)，那是因为即使删除 k 条边，点集合内一定存在某个点 j，满足 (u, a_j) 和 (v, a_j) 均存在。

所有只需在修改时对于点的权值从小到大排序，每次询问时取出前 $k+3$ 小的点和询问点 u, v，断开一些边，使用 floyd 算法求出询问点之间最短路即可，时间复杂度 $O((n+q)\log n + (k+3)^3 q)$。

```
void solve()
{
    int n, q;
    cin >> n >> q;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    set<array<int, 2>> mst;
    for (int i = 1; i <= n; i++)
        mst.insert({a[i], i});
    for (int _ = 1; _ <= q; _++)
```

```
    {
        int op;
        cin >> op;
        if (op == 1)
        {
            int x, b;
            cin >> x >> b;
            mst.erase({a[x], x});
            a[x] = b;
            mst.insert({a[x], x});
        }
        else
        {
            int qu, qv;
            cin >> qu >> qv;
            int k;
            cin >> k;
            set<array<int, 2>> block;
            for (int i = 1; i <= k; i++)
            {
                int u, v;
                cin >> u >> v;
                block.insert({u, v}), block.insert({v, u});
            }
            vector<int> candi;
            candi.push_back(qu), candi.push_back(qv);
            int cnt = 0;
            for (auto [d, u] : mst)
            {
                if (cnt >= 12)
                    break;
                ++cnt;
                if (u == qu || u == qv)
                    continue;
                candi.push_back(u);
            }
            vector<vector<int>> dp(candi.size(), vector<int>(candi.size()));
            for (int i = 0; i < candi.size(); i++)
                for (int j = i + 1; j < candi.size(); j++)
                {
                    if (block.count({candi[i], candi[j]}))
                        dp[i][j] = dp[j][i] = int_inf;
                    else
                        dp[i][j] = dp[j][i] = min(a[candi[i]], a[candi[j]]);
                }
            for (int k = 0; k < candi.size(); k++)
                for (int i = 0; i < candi.size(); i++)
                    for (int j = 0; j < candi.size(); j++)
                        dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j]);
            writeln(dp[0][1]);
        }
    }
}
```

J. MEX Should Be Same

鸽巢原理的简单应用，非 0 数和 0 的数字数量必定有一个数量小于等于 $\lfloor \frac{n\times n}{2} \rfloor$。

假如 0 的数量小于等于 $\lfloor \frac{n\times n}{2} \rfloor$，将所有 0 改成 1 即可，否则把所有非 0 数改为 0。

```
void solve()
{
    int n;
    cin >> n;
    vector<vector<int>> a(n + 1, vector<int>(n + 1));
    int zerocnt = 0;
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            cin >> a[i][j];
            zerocnt += (a[i][j] == 0);
        }
    }
    if (zerocnt <= (n * n) / 2)
    {
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                if (a[i][j] == 0)
                    a[i][j] = 1;
    }
    else
    {
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                if (a[i][j])
                    a[i][j] = 0;
    }
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            cout << a[i][j] << " \n"[j == n];
}
```

K. LRU is Best? (Easy Version)

简单版就是操作系统页替换的OPT算法，每次替换最远的出现页即可，假如当前的数字是最远出现的，不替换即可。

时间复杂度可以做到 $O(n \log n)$。

```
void solve()
{
    int n, m;
    cin >> n >> m;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++)
        cin >> a[i];
```

```
    for (int i = 1,x; i <= n; i++)
        cin >> x;
    for (int i = 1,x; i <= n; i++)
        cin >> x;
    for (int i = 1,x; i <= n; i++)
        cin >> x;
    vector<vector<int>> pos(n + 1);
    for (int i = 1; i <= n; i++)
        pos[a[i]].push_back(i);
    for (int i = 1; i <= n; i++)
        pos[i].push_back(n + 1);
    set<int> in;
    set<array<int, 2>, greater<>> pq;
    ll score = 0;
    for (int i = 1; i <= n; i++)
    {
        if (in.count(a[i]))
        {
            score += int(1);
        }
        else
        {
            // try_replace
            auto nxt = *upper_bound(all(pos[a[i]]), i);
            pq.insert({nxt, a[i]});
            in.insert(a[i]);
            if (pq.size() > m)
            {
                auto [d, u] = *pq.begin();
                in.erase(u), pq.erase(pq.begin());
            }
        }
        // update next
        if (in.count(a[i]))
        {
            auto nxt = *upper_bound(all(pos[a[i]]), i);
            pq.erase({i, a[i]});
            pq.insert({nxt, a[i]});
        }
    }
    writeln(score);
}
```

L. LRU is Best? (Hard Version)

首先 y 数组其实是没用的，可以先把得分设置成 $-\sum y$，然后使得每次命中的分数变成 $x_i + y_i$ 即可。

使用费用流解决这个问题，需要将每个点 i 拆成两个点 i, i'，连边方法如下。

首先，从源点向点 1 连接一条费用 0，流量 m 的边，从点 n 向汇点连接一条相同的边，代表最大只能同时取 m 个数字。

接着对于任意一对 $i, i+1$ ，连接一条费用 0，流量 m 的边。

接下来，在 i, i' 间，连接一条流量为 1，费用为 z_i 的边，代表将这个点加入到数组 b 中。

对于任意一对 i, j 满足 $a[i] == a[j]$ 的点，在 i', j' 间连接一条流量为 1 费用为 $-(x_i + y_i)$ 的边，代表获得得分。

最后，对于任意一对 $i', i+1$ 连接一条连接一条费用 0，流量 1 的边，代表把这个数字从数字 b 中替换。

答案就是 $-\sum y - min\_cost$，时间复杂度 $O(n^3)$。

```cpp
#define int ll
struct MCFGraph
{
    struct Edge
    {
        int v;
        long long c, f;
        Edge(int v, long long c, long long f) : v(v), c(c), f(f) {}
    };
    const int n;
    std::vector<Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<long long> h, dis;
    std::vector<int> pre;
    bool spfa(int s, int t)
    {
        std::queue<int> que;
        std::vector<bool> inQueue(n, false);
        dis.assign(n, std::numeric_limits<long long>::max());
        pre.assign(n, -1);
        h.assign(n, 0);
        dis[s] = 0;
        que.push(s);
        inQueue[s] = true;
        while (!que.empty())
        {
            int u = que.front();
            que.pop();
            inQueue[u] = false;
            for (int i : g[u])
            {
                int v = e[i].v;
                long long c = e[i].c;
                long long f = e[i].f;
                if (c > 0 && dis[v] > dis[u] + h[u] - h[v] + f)
                {
                    dis[v] = dis[u] + h[u] - h[v] + f;
                    pre[v] = i;
                    if (!inQueue[v])
                    {
                        que.push(v);
                        inQueue[v] = true;
                    }
                }
            }
        }
        return dis[t] != std::numeric_limits<long long>::max() && dis[t] < 0;
    }
```

```
    MCFGraph(int n) : n(n), g(n) {}
    void addEdge(int u, int v, int w, int c)
    {
        g[u].push_back(e.size());
        e.emplace_back(v, w, c);
        g[v].push_back(e.size());
        e.emplace_back(u, 0, -c);
    }
    std::pair<long long, long long> flow(int s, int t)
    {
        long long flow = 0;
        long long cost = 0;
        while (spfa(s, t))
        {
            for (int i = 0; i < n; ++i)
                h[i] += dis[i];
            long long aug = std::numeric_limits<long long>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].v)
                aug = std::min(aug, e[pre[i]].c);
            for (int i = t; i != s; i = e[pre[i] ^ 1].v)
            {
                e[pre[i]].c -= aug;
                e[pre[i] ^ 1].c += aug;
            }
            flow += aug;
            cost += (long long)(aug)*h[t];
        }
        return std::make_pair(flow, cost);
    }
};
void solve()
{
    ll n, m;
    cin >> n >> m;
    vector<ll> a(n + 1), x(n + 1), y(n + 1), z(n + 1);
    vector<vector<ll>> pos(n + 1);
    for (int i = 1; i <= n; i++)
        cin >> a[i], pos[a[i]].push_back(i);
    for (int i = 1; i <= n; i++)
        cin >> x[i];
    for (int i = 1; i <= n; i++)
        cin >> y[i];
    for (int i = 1; i <= n; i++)
        cin >> z[i];
    ll ans = -accumulate(notall(y), 0ll);
    for (int i = 1; i <= n; i++)
        x[i] += y[i];
    MCFGraph g(2 * n + 2);
    for (int i = 0; i < n; i++)
        g.addEdge(i, i + 1, m, 0);
    g.addEdge(n, 2 * n + 1, m, 0);
    for (int i = 1; i <= n; i++)
        g.addEdge(i, i + n, 1, z[i]);
    for (int i = 1; i <= n; i++)
        g.addEdge(i + n, i != n ? i + 1 : n * 2 + 1, 1, 0);
    for (int i = 1; i <= n; i++)
```

```
    {
        int sz = pos[i].size();
        for (int j = 0; j < sz - 1; j++)
        {
            int p1 = pos[i][j], p2 = pos[i][j + 1];
            g.addEdge(n + p1, n + p2, 1, -x[p2]);
        }
    }
    auto [f, c] = g.flow(0, n * 2 + 1);
    ans -= c;
    writeln(ans);
}
```

M. 猫娘部署

解法一

爆搜，但是得剪枝。

一个一个搜，需要先往放猫的方法搜，然后需要根据答案大小剪枝一下，如果当前的猫娘太少了，则这个状态就不行。

一行一行搜会比较好，可以先**预处理一行里面合法的状态**，搜下一行合法的状态，不行就退出，然后也需要先考虑猫娘比较多的方案，根据答案做剪枝。

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

using namespace std;

int n, m, total, ans;
int status[1024], terrian[1024], record[1024];

bool compatible(int a, int b) { return !(a & b); }

bool compatible(int a) { return compatible(a, a >> 1); }

void dfs(int dep, int last_mask, int count) {
    record[dep] = max(count, record[dep]);
    if (count + m / 2 + 1 < record[dep]) {
        return;
    }
    if (dep == n)
        return void(ans = max(ans, count));
    for (int i = 0; i < total; i++) {
        if (!compatible(status[i], last_mask))
            continue;
        if (!compatible(status[i], terrian[dep]))
            continue;
        dfs(dep + 1, status[i], count + __builtin_popcount(status[i]));
    }
}

int main() {
```

```
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        char ch[16];
        cin >> ch;
        for (int j = 0; j < m; j++) {
            terrian[i] |= (ch[j] == 'N') << j;
        }
    }

    for (int mk = 0; mk < (1 << m); mk++) {
        if (!compatible(mk))
            continue;
        status[total++] = mk;
    }

    sort(status, status + total, [](int a, int b) { return __builtin_popcount(a) > __builtin_popcount(b); });

    dfs(0, 0, 0);
    cout << ans << endl;
    return 0;
}
```

解法二

状压 DP，记录 $dp_{i,s}$ 表示解决到第 i 行，上一行填猫的状态是 s 时，最大能填多少猫娘。转移直接枚举一整行的猫娘状态，判断一下是否冲突即可。

解法三

对整个网格做**黑白染色**，构成一张**二分图**。要求猫娘不能相邻就是在二分图中选一些点，使得他们没有边连接。

经典的最大独立集问题，跑一个二分图匹配即可。

N. 混沌数字题解

降噪

噪声是随机翻转黑白像素，字是老老实实写的，可以用并查集算连通块大小来降噪。

其实更简单的办法是，如果相邻 8 个位置反色的更多（比如 6 个），那么就认为这个点的颜色被翻转了。

降噪后图长这样，下面哪个是原样的对比图：

解法一

先考虑把每个字符分开，观察到字符之间有一段全空的，对于每一列，设一个阈值，如果白点不超过阈值就认为是空的，再设一个阈值，如果连续这段都是空的，就从这里分割。

拿到每个字符的切割图像。

考虑 01 的**拓扑结构**，0 是有内外之分的，拿一条线从中间左到右穿过去，数交点个数就好。

也可以用别的拓扑结构性质来做。

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

using namespace std;

int n, m;
int a[105][1005], b[105][1005], cnt[1005], iskey[1005];

bool inside(int x, int y) {
    int o1 = 0, o2 = 0;
    int threshold = n / 16 + 1;
    for (int i = x; i >= 0; i--) {
        o1 += a[i][y];
    }
    for (int i = x; i <= n; i++) {
        o2 += a[i][y];
    }
    return o1 > threshold && o2 > threshold && a[x][y] == 0;
}

bool is_o_inside(int y) {
    int inside_threshold = n / 3 + 1, cnt = 0;
    for (int i = 1; i < n; i++) cnt += inside(i, y);
    // cerr<<"Test:"<<y<<' '<<cnt<<endl;
    return cnt >= inside_threshold;
}

bool legal(int x, int y) { return x >= 0 && x < n && y >= 0 && y < m; }

int solve(int l, int r) {
    int threshold = 12;
    int cnt = 0;
    for (int i = l; i <= r; i++) {
        cnt += is_o_inside(i);
```

```
    }
    if (cnt >= threshold)
        return 0;
    return 1;
}

int main() {
    scanf("%d%d", &n, &m);
    int key_col_threshold = n / 6 + 1;
    for (int i = 0; i < n; i++) {
        char s[1006];
        scanf("%s", s);
        for (int j = 0; j < m; j++) {
            a[i][j] = s[j] == '1', cnt[j] += a[i][j];
        }
    }

    for (int i = 0; i < m; i++) {
        iskey[i] = cnt[i] >= key_col_threshold;
    }

    int last = 0, length = 0;
    for (int i = 1; i < m; i++) {
        if (iskey[i] && !iskey[i - 1]) {
            last = i;
        }
        if (!iskey[i] && iskey[i - 1]) {
            if (i - last > 10) {
                printf("%d", solve(last, i - 1));
            }
        }
    }

    return 0;
}

```

解法二

还是把每个字符切开，然后看最中央的一部分，数一下白点数量，白点多就是 1，少就是 0。

解法三

并查集，0 的联通白点多，1 的联通白点少，自己估一下大概是多少就好，并查集太小的就是噪点，直接忽略掉。

总结

以上三个解法如果降噪之后基本不用太关心阈值，如果不降噪，那可能需要扣一下阈值。降噪还有种思路是跑高斯模糊，但都差不多。