A. 扫落叶

考虑对奇点的性质分类讨论：

- 如果奇点是叶子
  则先手在第一次直接拿走，此时，先手必胜
- 奇点不是叶子
  则双方一定要防止在自己的回合让奇点暴露出来，可以考虑奇点不暴露的最小情况：最少需要3个节点才能保证节点不暴露出来，因此3到谁手上就一定必输，此时因为Orange是先手，因此如果节点数是**偶数**，则Orange先手必胜。

因此，我们只需要存储每个节点的度，判断奇点所在的节点度数是否为1，否则判断n的奇偶性即可。

```
void solve()
{
        int n, x;
        cin >> n >> x;
        vector<int> d(n + 1);
        for(int i = 1; i < n; i ++)
        {
                int u, v;
                cin >> u >> v;
                d[u]++, d[v]++;
        }
        if(d[x] == 1) cout << "win\n";
        else
                if(n % 2 == 0) cout << "win\n";
                else cout << "lose\n";
}
```

B. 随机捕获

由于Orange和扑满的位置都是取遍 a 和 b 序列的所有值的，而且捕获只与 x 和 y 坐标单独的位置有关，因此我们可以分开考虑在单独一个维度。

对于一个维度，考虑枚举扑满的取值，那么对于扑满的当前位置 p，所有大于 p 的位置都是Orange可以捕获成功的。把这些位置加起来就是单个维度上所有可以成功捕获扑满的情况。

根据乘法原理，答案就是两个维度所有成功捕获扑满情况的乘积。

```
signed main()
{
    IOS;
    i64 n, m;
    cin >> n >> m;
    vector<int> a(n), b(m);
    map<int, int> cnta, cntb;
    for(int i = 0; i < n; i ++) cin >> a[i], cnta[a[i]]++;
    for(int i = 0; i < m; i ++) cin >> b[i], cntb[b[i]]++;
    i64 ansA = 0;
    for(auto[v, cnt] : cnta)
    {
        ansA += cnt * n;
        n -= cnt;
    }
    i64 ansB = 0;
    for(auto[v, cnt] : cntb)
    {
        ansB += cnt * m;
        m -= cnt;
    }
    cout << ansA * ansB << '\n';
    return 0;
}
```

C. 对立数组

钦定选择的位置为 p，那么实际上问题转化为，我们要找到一个 p 最小化如下函数的值：

$$f(p)=|\sum_{i=k}^{p}i-\sum_{i=p+1}^{k+n-1}i|$$

而根据等差数列求和公式，原式可以转化成两个等差数列求和：

$$f(p)=|\frac{(k+p)p}{2}-\frac{(p+1+k+n-1)(n-p)}{2}|$$

考虑去掉绝对值，则函数是一个根据 p 单调递增的函数，而且一定会穿过 x 轴由负到正，那么加上绝对值之后，负数部分的图像向上反转，则函数为一个单峰函数，存在极小值。

因此可以采用 **三分算法** 求出最小值。

```
void solve()
{
    i64 n, k;
    cin >> n >> k;
    auto sum = [&](i64 x, i64 y) -> i64 { return (x + y) * (y - x + 1) / 2;};
    auto f = [&](i64 x) -> i64 {return abs(sum(k, k + x - 1) - sum(k + x, k + n - 1));};
    i64 l = 0, r = n;
    while(l < r)
    {
        i64 ml = l + r >> 1ll;
        i64 mr = ml + 1ll;
        if(f(ml) > f(mr)) l = ml + 1;
        else r = ml;
    }
    cout << f(r) << '\n';
}
```

D. 幽谷传响

直接枚举所有子数组，求最小值再相加显然是不可做的。

我们可以考虑贡献法：**对于每个位置的每个数，求出它作为最小值的所有子数组。**

假设 a_p 作为子数组的最小值，那么他的子数组左端点可以取到左边第一个小于的 a_p 的前一个位置，右端点可以取到右边第一个小于 a_p 的位置的前方。这可以用单调栈简单求出。

同时需要考虑重复贡献问题：

对于如下序列，钦定 p 为其中的最小值，且 $a,b,...,i$ 均大于 p，则整个序列会被第一个 p 和第二个 p 重复计数。

$$a,b,c,\underline{p},d,e,f,\underline{p},g,h,i$$

为了避免这种情况，可以在计算后一个 p 的时候，把左端点范围设置到大于上一个 p 位置的前方。类似于 顾影自怜 。

```cpp
constexpr int mod = 1e9 + 7;
int Solve(vector<int>& arr) {
    vector<int> nxt_l(arr.size()), pre_l(arr.size());
    vector<vector<int>> pos(1e6 + 1);
    for(int i = 0; i < arr.size(); i ++) pos[arr[i]].push_back(i);
    stack<int> stk;
    for(int i = 0; i < arr.size(); i ++)
    {
        while(stk.size() and arr[stk.top()] >= arr[i]) stk.pop();
        pre_l[i] = stk.size() ? stk.top() : -1;
        stk.push(i);
    }
    stk = stack<int>();
    for(int i = arr.size() - 1; i >= 0; i --)
    {
        while(stk.size() and arr[stk.top()] >= arr[i]) stk.pop();
        nxt_l[i] = stk.size() ? stk.top() : arr.size();
        stk.push(i);
    }
    long long ans = 0;
    for(auto v : pos)
        for(int i = 0; i < v.size(); i ++)
            ans += 1ll * (v[i] - (i == 0 ? pre_l[v[i]] : max(pre_l[v[i]], v[i - 1]))) * (nxt_l[v[i]] - v[i]) % mod * arr[v[i]] % mod, ans %=
    return ans;
}

int main()
{
    cin.tie(0) -> sync_with_stdio(0);
    int n;
    cin >> n;
    vector<int> a(n);
    for(int i = 0; i < n; i ++) cin >> a[i];
    cout << solve(a) << '\n';
}
```

E. 收获日

> ps：真的看懂题意了吗?

事实上，第 i 天收走数量为 i 的所有颜色的羊毛，实际上可以得出第 i 天的总收获就是所有数量小于 i 的颜色的羊的总和。因此我们可以求出数量为 y 的羊的总数 $cnt[y]$，对其做前缀和即可快速求出所有回答。

```cpp
void solve() {
    int n, q;
    cin >> n >> q;
    map<int, int> mp; // 记录颜色为 x 的羊的数量
    for(int i = 1, x; i <= n; i++) {
        cin >> x;
        mp[x] ++ ;
    }
    vector<int> cnt(n + 1);
    for(auto & [x, y] : mp)
        cnt[y] += y; // 统计数量为y的所有颜色羊数量的和
    for(int i = 1; i <= n; i++) // 前缀和
        cnt[i] += cnt[i - 1];
    while(q -- ) {
        int d;
        cin >> d;
        d = min(d, n);
        cout << cnt[d] << " ";
    }
    cout << '\n';
}
```

F. 随机旅行

求随机选一个里程卡能去的不同城市的期望，实际上就是所有里程卡能到的城市除以里程卡的数量。

因此本题转化成每次询问从 s_i 出发，有多少个点到 s_i 的最短路小于等于 v_i。

可以对每个点跑一遍 $dijkstra$，求出所有点互相的最短路，然后对于每次询问，枚举所有点，累加最短路小于等于 v_i 的数量即可。最后除以 q 即可。

```
signed main()
{
    IOS;
    int n, m, q;
    cin >> n >> m >> q;
    vector<vector<pii>> edge(n + 1);
    while(m--)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edge[u].push_back({v, w});
        edge[v].push_back({u, w});
    }
    auto djikstra = [&](int st, vector<i64> &dist)
    {
        dist = vector<i64>(n + 1, 1e18);
        dist[st] = 0;
        vector<int> vis(n + 1);
        priority_queue<pii, vector<pii>, greater<pii>> heap;
        heap.push({0, st});
        while(heap.size())
        {
            auto [d, u] = heap.top();
            heap.pop();
            if(vis[u]) continue;
            vis[u] = 1;
            for(auto [v, w] : edge[u])
            {
                if(dist[v] > dist[u] + w)
                {
                    dist[v] = dist[u] + w;
                    heap.push({dist[v], v});
                }
            }
        }
    };
    vector<vector<i64>> dist(n + 1, vector<i64>(n + 1));
    for(int i = 1; i <= n; i ++)
        djikstra(i, dist[i]); // 求出所有点的单源最短路
    int sum = 0;
    for(int i = 1; i <= q; i ++)
    {
        int s, v;
        cin >> s >> v;
        for(int i = 1; i <= n; i ++)
            if(dist[s][i] <= v) sum ++;
    }
    cout << fixed << setprecision(2) << 1.0 * sum / q << '\n';
    return 0;
}
```