概述

	A. 落于现象	B. 大娱乐至上	C. 跳房子游戏	D. 好奇心宝贝	E. 在仙境之外	F. 可度量之心	G. 美丽新世界
题目参考难度	1200	800	1400	1300	1600	1400	1800
涉及知识点	模拟，递推	模拟	图论，最短路	结论题，模拟	差分	前缀和	数论，筛法

题解报告

> tips:如果有不懂的地方可以在评论区留言.

A. 落于现象

> tips：本题是斐波那契数改的题

我们假设 $f[i][0]$ 表示立方体第 i 秒后的奇偶，$f[i][1]$ 表示锥体第 i 秒后的奇偶，$f[i][2]$ 表示圆柱体第 i 秒后的奇偶。

那么有如下递推式：

$$f[i][0] = (2 \times f[i-1][0] + f[i][1] \times 2 + f[i][2]) mod\ 2$$

$$f[i][1] = (f[i-1][1] + f[i-1][0] + f[i-1][2]) mod\ 2$$

$$f[i][2] = (f[i-1][2] + f[i-1][0] + 2 \times f[i-1][1]) mod\ 2$$

注意到数据范围，我们可以通过递推式预处理出 1 到 10^6 内所有图形数量的奇偶。

然后每次 $O(1)$ 查询即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e6 + 5;
int f[maxn][3];
int main()
{
    ios::sync_with_stdio(false);
    f[0][0] = 1;
    f[0][1] = 1;
    f[0][2] = 1;
    for (int i = 0; i < 1e6; i++)
    {
        f[i + 1][0] = f[i][0];
        f[i + 1][1] = f[i][1];
        f[i + 1][2] = f[i][2];
        f[i + 1][0] += f[i][0];
        f[i + 1][0] %= 2;
        f[i + 1][1] += f[i][0];
        f[i + 1][1] %= 2;
        f[i + 1][2] += f[i][0];
        f[i + 1][2] %= 2;
        f[i + 1][0] += f[i][1] * 2 % 2;
        f[i + 1][0] %= 2;
        f[i + 1][2] += f[i][1] * 2 % 2;
        f[i + 1][2] %= 2;
        f[i + 1][0] += f[i][2];
        f[i + 1][0] %= 2;
        f[i + 1][1] += f[i][2];
        f[i + 1][1] %= 2;
    }
    int T;
    cin >> T;
    while (T--)
    {
        int x;
        cin >> x;
        cout << f[x][0] << ' ' << f[x][1] << ' ' << f[x][2] << '\n';
    }
    return 0;
}
```

B. 大娱乐至上

实际上，就是问你现在手上有多少种牌，再用52减掉你手上有的牌的种类数量就好了，直接用 `set` 去重即可。

```cpp
int main()
{
        set <string> st;
        int n;
    cin >> n;
        for (int i = 1; i <= n; i++)
    {
                string s;
                cin >> s;
                st.insert(s);
        }
        cout << 52 - st.size() << '\n';
        return 0;
}
```

C. 跳房子游戏

Dijkstra算法模板题，只不过是把图上的边换成了网格图。正常写即可。

```
int graph[510][510];
int dx[] = {0,1,0,-1};
int dy[] = {1,0,-1,0};
signed main()
{
    int n;
    cin >> n;
    for(int i = 0; i < n ; i ++)
        for(int j = 0; j < n; j ++)
            cin >> graph[i][j];
    priority_queue<pii, vector<pii>, greater<pii>> heap;
    vector<int> dist(n * n + 1, 1e9), vis(n * n + 1);
    dist[0] = graph[0][0];
    heap.push({dist[0], 0});
    while(heap.size())
    {
        int u = heap.top().second;
        heap.pop();
        int x = u / n, y = u % n;
        if(vis[u]) continue;
        vis[u] = 1;
        for(int i = 0; i < 4; i ++)
        {
            int px = x + dx[i], py = y + dy[i];
            if(px < 0 || py < 0 || px >= n || py >= n) continue;
            int v = px * n + py;
            if(dist[v] > dist[u] + graph[px][py])
            {
                dist[v] = dist[u] + graph[px][py];
                heap.push({dist[v], v});
            }
        }
    }
    cout << dist[(n - 1) * n + (n - 1)];
    return 0;
}
```

D. 好奇心宝贝

实际上，由于所有魔精都是相同的，我们可以把两只魔精撞到一起然后反向看成是两只魔精互相穿过，所以实际上就是所有魔精在沿着自身方向不断前进直到掉下去。因此，我们只要找到最右边的向左走的魔精和最左边的向右走的魔精，然后计算一下它们谁先掉下去即可。

```
signed main()
{
    IOS;
    int n, m;
    cin >> n >> m;
    vector<int> pos(m);
    string s;
    for(int i = 0; i < m; i ++) cin >> pos[i];
    cin >> s;
    int ans = 0;
    for(int i = 0; i < m; i ++)
    {
        if(s[i] == '1')
            ans = max(ans, n - pos[i] + 1);
        else
            ans = max(ans, pos[i]);
    }
    cout << ans << '\n';
    return 0;
}
```

E.在仙境之外

可以联想到Orange的区间修改I, 是一个很经典的区间加同一个数，最后询问修改的区间。本题与上一题不同点在于，本题是给区间加上一个从 1 开始的等差数列。

我们思考差分的本质，实际上就是让 $d_i = a_i - a_{i-1}$。那么我们从差分数组的角度来看问题，对于区间 $[l, r]$ 加上一个等差数列，实际上就是让 d_l 到 d_r 所有数都加上 1，那么这就转化成了上一题的区间修改问题。

因此，我们对差分数组再做一次差分，记 $dd_i = d_i - d_{i-1}$，那么我们对原区间 $[L, R]$ 加上一个等差数列，等价于对差分数组的区间加上 1，等价于对 $dd_L + 1$，那么我们现在应该思考，dd_{R+1} 应该如何修改呢，我们注意到，假设我们对 $[1, 5]$ 加上一个等差数列，那么可以转化成在 1 位置加上一个从 1 开始的等差数列，再在 6 位置减去一个从 6 开始的等差数列，我们把 $6, 7, 8, \cdots$ 做一次差分，得到 $6, 1, 1, 1 \cdots$，再次差分得到 $6, -5, 0, 0, 0, \ldots$。所以我们发现，要消除之前的影响，只需要在 dd_6 位置减去 6，再在 dd_7 位置加上 5 即可。

那么实际上对区间 $[L, R]$ 加上一个从 1 开始的等差数列，实际上就是对二阶差分数组 dd_L 加上 1，再对 dd_{R+1} 减去 $R - L + 2$ 以及 dd_{R+2} 加上 $R - L + 1$ 即可。

最后，我们只需要对差分数组做两次前缀和即可。

```
signed main()
{
    IOS;
    int n, m;
    cin >> n >> m;
    vector<i64> d(n + 3);
    while(m--)
    {
        int l, r;
        cin >> l >> r;
        d[l] ++;
        d[r + 1] -= r - l + 2;
        d[r + 2] += r - l + 1;
    }
    for(int i = 1; i <= n; i ++) d[i] += d[i - 1];
    for(int i = 1; i <= n; i ++) d[i] += d[i - 1];
    for(int i = 1; i <= n; i ++) cout << d[i] << ' ';
    return 0;
}
```

F. 可度量之心

一颗钻石等于两颗银子弹，一颗银子弹等于两颗钻石。我们直接把所有的银子弹转化成钻石来看就好。那么我们记 $\hat{} = 2$，$* = 1$。那么我们直接对两个数组作前缀和就好。每次验证两个字串是否等价实际上就是看他们权值是否相等，但是注意到，我们每次可以在任意地方加入三颗钻石，也就是说，我们还需要判断两个区间的权值之和是否差值是 3 的倍数，如果是，那么也可以看成他们是等价的。或者，从更一般的角度看，实际上就是看两个区间的权值之和是否对 3 同余。

```
const int N = 1e5 + 5;
typedef long long ll;

int pres[N];
int pret[N];
int main() {
    cin.tie(nullptr)->sync_with_stdio(false);
    string s, t;
    int a, b, c, d;
    int q;
    cin >> s >> t;
    cin >> q;

    pres[1] = (s[0] == '^' ? 2 : 1);
    pret[1] = (t[0] == '^' ? 2 : 1);
    for(int i = 1; i < s.size(); i ++) pres[i + 1] = pres[i] + (s[i] == '^' ? 2 : 1);
    for(int i = 1; i < t.size(); i ++) pret[i + 1] = pret[i] + (t[i] == '^' ? 2 : 1);

    while (q--) {
        cin >> a >> b >> c >> d;
        cout << ((pres[b] - pres[a - 1]) % 3 == (pret[d] - pret[c - 1]) % 3
                    ? "Yes\n"
                    : "No\n");
    }

    return 0;
}
```

G. 美丽新世界

x 包容 y，实际上就是 x 是 y 的倍数。那么一个数能包容 1 到 n 所有的数，实际上就是 1 到 n 所有数的公倍数。同时我们还要求这个数最小，因此题意就是在求 1 到 n 所有数的最小公倍数。

那么求最小公倍数，如果我们用质因数分解对每个数都操作一次，复杂度是 $O(n\sqrt{n})$ 的，显然会超时。我们考虑如何优化。

根据唯一质因数分解定义，对于任意的一个数 a，我们可以写成如下形式：

$$a = p_1^{c_1} \times p_2^{c_2} \cdots \times p_n^{c_n}$$

那么对于另一个数 b，我们也写成如下形式：

$$b = p_1^{d_1} \times p_2^{d_2} \cdots \times p_n^{d_n}$$

那么 a 和 b 的最小公倍数就是：

$$\operatorname{lcm}(a,b) = p_1^{\max(c_1,d_1)} \times p_2^{\max(c_2,d_2)} \cdots \times p_n^{\max(c_n,d_n)}$$

那么实际上，把上述结论推广到若干个数：

$$\operatorname{lcm}(a_1,a_2,\cdots,a_n) = p_1^{\max(c_{1,1},c_{2,1},\cdots,c_{n,1})} \times p_2^{\max(c_{1,2},c_{2,2},\cdots,c_{n,2})} \cdots \times p_n^{\max(c_{1,n},c_{2,n},\cdots,c_{n,n})}$$

实际上，我们只需要对每个质数 p_i，求出 1 到 n 中每个数中 p_i 的指数的最大值即可。

可以考虑先用质数筛，筛出所有的质数 p_i，然后，在 1 到 n 中，p_i 次数最高的那个数的次数一定是 $\lfloor \log_{p_i}^n \rfloor$。那么我们只需要对于每个质数 p_i，把他们的 $\lfloor \log_{p_i}^n \rfloor$ 次方累乘起来即可。

$$ans = \prod_{p_i}^{p_i \le n} p_i^{\lfloor \log_{p_i}^n \rfloor}$$

```cpp
const int mod = 998244353;
const int N = 1e7 + 10;
int vis[N];
vector<int> primes;
signed main()
{
    i64 lcm = 1;
    auto mylog = [&](int a, int b) -> int // 求以a为底数b的对数
    {
        return log(b) / log(a);
    };
    auto qmi = [&](int a, int k) // 快速幂
    {
        int res = 1;
        while (k)
        {
            if (k & 1)
                res = (i64)res * a % mod;
            a = (i64)a * a % mod;
            k >>= 1;
        }
        return res;
    };
    int n;
    cin >> n;
    for (int i = 2; i <= n; i++)
    {
        if (!vis[i])
        {
            primes.push_back(i);
            // 每次找到一个质数，就把他的log_p^n次幂累乘到lcm中
            lcm = (i64)lcm * qmi(i, mylog(i, n)) % mod;
        }
        for (int p : primes)
        {
            if (p * i > n)
                break;
            vis[p * i] = 1;
            if (i % p == 0)
                break;
        }
    }
    cout << lcm << '\n';
    return 0;
}
```