2025 CEIT 3月周赛II

A. 图形判断

注意到，面积是一个整数，然而，不存在边长为整数的等边三角形面积为整数，因此只需要判断是不是正方形即可。即：

$$\lfloor \sqrt{s} \rfloor^2 = s$$

```
void solve()
{
    int s;
    cin >> s;
    int t = sqrt(s);
    if(t * t == s) cout << "0\n";
    else cout << "3\n";
}
```

B. 卡牌游戏

本质上本题是要求给定序列的 **最大子段和**。

考虑动态规划：

我们设 $f[i]$ 表示以位置 i 结尾的最大子段和，则有：

$$f[i] = f[i-1] + a_i,\ f[i-1] \geq 0$$
$$f[i] = a_i,\ f[i-1] < 0$$

答案则为:

$$\text{ans} = \max_{i=1}^{n} f[i]$$

```
using i64 = long long;
void solve()
{
    int n;
    cin >> n;
    vector<int> a(n + 1);
    for(int i = 1; i <= n; i ++) cin >> a[i];
    i64 sum = a[1], ans = a[1];
    for(int i = 2; i <= n; i ++)
    {
        if(sum <= 0) sum = a[i];
        else sum += a[i];
        ans = max(ans, sum);
    }
    cout << ans << '\n';
}
```

C. 连携并发

考虑贡献法：

我们考虑每个 1 对后方 1 的贡献，对于位置 p 的 1，会给位置 $q(q > p)$ 贡献 $q - p$ 点能量。

在数轴上，我们可以看成从 $p + 1$ 位置开始，往后每个位置均为一个从 1 开始递增的公差为 1 的等差数列。其后方某个位置有 1，就累加上延申过来的等差数列的值。

因而最后转化为，每个 1 都会给当前位置的后一个位置一直到末尾加上一个等差数列。因此可以用 **二阶差分** 解决。

最后对还原后的原数组累加所有贡献即可。

```
signed main()
{
    IOS;
    int n;
    cin >> n;
    string s;
    cin >> s;
    s = '*' + s;
    vector<int> d(n + 1);
    auto presum = [&](vector<int> &a) // 对a进行一次前缀和
    {
        for(int i = 1; i < a.size(); i ++) a[i] = (a[i] + a[i - 1]) % mod;
    };
    for(int i = 1; i <= n; i ++)
        if(s[i] == '1')
            d[i + 1] ++;
    presum(d); presum(d);
    i64 ans = 0;
    for(int i = 1; i <= n; i ++)
    {
        if(s[i] == '1')
        {
            ans = ans + d[i];
            ans %= mod;
        }
    }
    cout << ans << '\n';
    return 0;
}
```

D. 首当其冲

我们要去找区间中第一个大于 v 的位置，可以考虑二分搜索。对于每个 mid，若 $a[l \sim mid]$ 的区间最大值大于 v，则答案在 $a[l \sim mid]$ 内，否则答案则在 $a[mid+1 \sim r]$ 内。同时，在最开始特判一下，整个区间的最大值是否大于 v，若不满足，则答案为 `-1`。

求区间最值可以考虑用 **ST表**。

```
struct ST {
    const int n, k;
    vector<int> in1, in2;
    vector<vector<int>> Max, Min;
    ST(int n) : n(n), in1(n + 1), in2(n + 1), k(__lg(n)) {
        Max.resize(k + 1, vector<int>(n + 1));
        Min.resize(k + 1, vector<int>(n + 1));
    }
    void init() {
        for (int i = 1; i <= n; i++) {
            Max[0][i] = in1[i];
            Min[0][i] = in2[i];
        }
        for (int i = 0, t = 1; i < k; i++, t <<= 1) {
            const int T = n - (t << 1) + 1;
            for (int j = 1; j <= T; j++) {
                Max[i + 1][j] = max(Max[i][j], Max[i][j + t]);
                Min[i + 1][j] = min(Min[i][j], Min[i][j + t]);
            }
        }
    }
    int getMax(int l, int r) {
        if (l > r) {
            swap(l, r);
        }
        int k = __lg(r - l + 1);
        return max(Max[k][l], Max[k][r - (1 << k) + 1]);
    }
    int getMin(int l, int r) {
        if (l > r) {
            swap(l, r);
        }
        int k = __lg(r - l + 1);
        return min(Min[k][l], Min[k][r - (1 << k) + 1]);
    }
};

void solve()
{
    int n, m;
    cin >> n >> m;
    ST st(n);
    for(int i = 1; i <= n; i ++) cin >> st.in1[i];
    st.init();
    while(m --)
    {
        int l, r, v;
        cin >> l >> r >> v;
        if(st.getMax(l, r) <= v)
        {
            cout << -1 << '\n';
            continue;
        }
        while(l < r)
        {
            int mid = l + r >> 1;
            if(st.getMax(l, mid) > v) r = mid;
            else l = mid + 1;
        }
        cout << r << '\n';
    }
}
```