【第四十二课】傅里叶变换与信息隐写术（二）

2151. 基于陈述统计最多好人数

难度困难29

游戏中存在两种角色：

- **好人**：该角色只说真话。
- **坏人**：该角色可能说真话，也可能说假话。

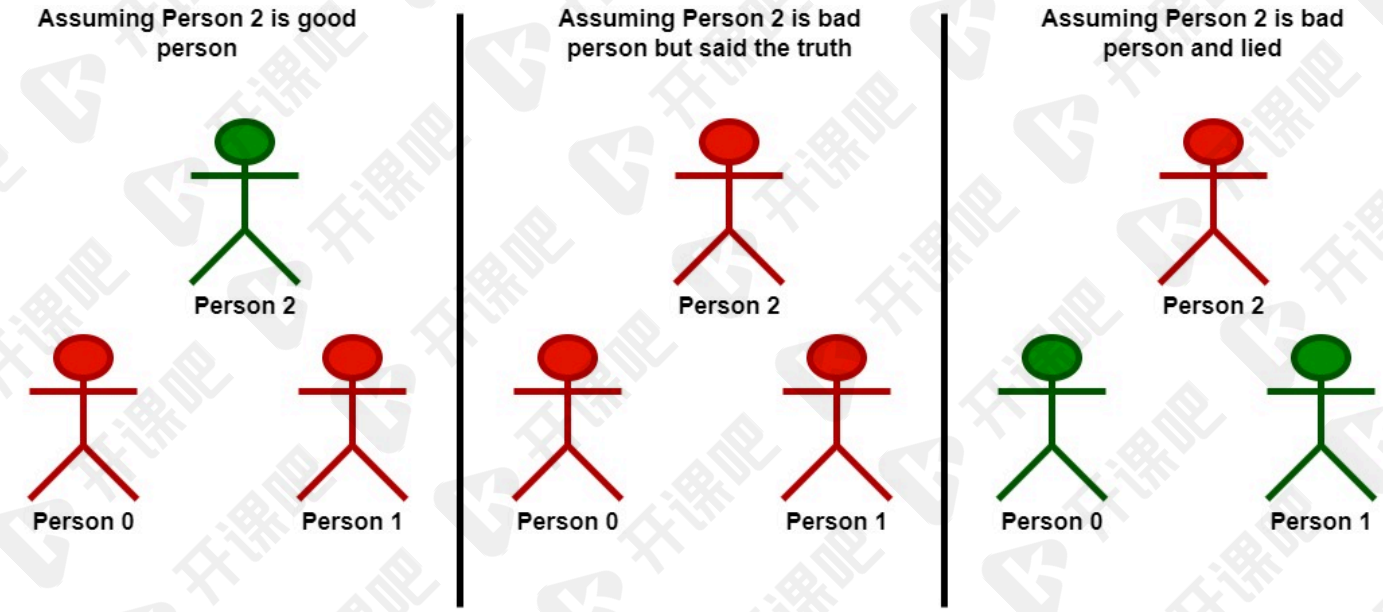
给你一个下标从 **0** 开始的二维整数数组 `statements` ，大小为 `n x n` ，表示 `n` 个玩家对彼此角色的陈述。具体来说， `statements[i][j]` 可以是下述值之一：

- `0` 表示 `i` 的陈述认为 `j` 是 **坏人** 。
- `1` 表示 `i` 的陈述认为 `j` 是 **好人** 。
- `2` 表示 `i` 没有对 `j` 作出陈述。

另外，玩家不会对自己进行陈述。形式上，对所有 `0 <= i < n` ，都有 `statements[i][i] = 2` 。

根据这 `n` 个玩家的陈述，返回可以认为是 **好人** 的 **最大** 数目。

示例 1：




```
输入：statements = [[2,1,2],[1,2,2],[2,0,2]]
输出：2
解释：每个人都做一条陈述。
- 0 认为 1 是好人。
- 1 认为 0 是好人。
- 2 认为 1 是坏人。
以 2 为突破点。
- 假设 2 是一个好人：
    - 基于 2 的陈述，1 是坏人。
    - 那么可以确认 1 是坏人，2 是好人。
```

```
    - 基于 1 的陈述，由于 1 是坏人，那么他在陈述时可能：
        - 说真话。在这种情况下会出现矛盾，所以假设无效。
        - 说假话。在这种情况下，0 也是坏人并且在陈述时说假话。
    -在认为 2 是好人的情况下，这组玩家中只有一个好人。
- 假设 2 是一个坏人：
    - 基于 2 的陈述，由于 2 是坏人，那么他在陈述时可能：
        - 说真话。在这种情况下，0 和 1 都是坏人。
            -在认为 2 是坏人但说真话的情况下，这组玩家中没有一个好人。
        - 说假话。在这种情况下，1 是好人。
            - 由于 1 是好人，0 也是好人。
            -在认为 2 是坏人且说假话的情况下，这组玩家中有两个好人。
在最佳情况下，至多有两个好人，所以返回 2 。
注意，能得到此结论的方法不止一种。
```

```cpp
class Solution {
public:
    int countOne(int x) {
        int cnt = 0;
        while (x) {
            x &= (x - 1);
            cnt += 1;
        }
        return cnt;
    }
    bool check(vector<vector<int>>& g, int mark, int n) {
        for (int i = 0; i < n; i++) {
            if ((mark & (1 << i)) == 0) continue;
            for (int j = 0; j < n; j++) {
                if (g[i][j] == 2) continue;
                if (g[i][j] != !!(mark & (1 << j))) return false;
            }
        }
        return true;
    }
    int maximumGood(vector<vector<int>>& statements) {
        int n = statements.size(), ans = 0;
        for (int i = 0, I = (1 << n); i < I; i++) {
            if (check(statements, i, n) == false) continue;
            ans = max(ans, countOne(i));
        }
        return ans;
    }
};
```

2163. 删除元素后和的最小差值

给你一个下标从 **0** 开始的整数数组 `nums` ，它包含 `3 * n` 个元素。

你可以从 `nums` 中删除 **恰好** `n` 个元素，剩下的 `2 * n` 个元素将会被分成两个 **相同大小** 的部分。

- 前面 `n` 个元素属于第一部分，它们的和记为 `sumfirst` 。
- 后面 `n` 个元素属于第二部分，它们的和记为 `sumsecond` 。

两部分和的 **差值** 记为 `sumfirst - sumsecond` 。

- 比方说， `sumfirst = 3` 且 `sumsecond = 2` ，它们的差值为 `1` 。
- 再比方， `sumfirst = 2` 且 `sumsecond = 3` ，它们的差值为 `1` 。

请你返回删除 `n` 个元素之后，剩下两部分和的 **差值的最小值** 是多少。

示例 1：

```
输入：nums = [3,1,2]
输出：-1
解释：nums 有 3 个元素，所以 n = 1 。
所以我们需要从 nums 中删除 1 个元素，并将剩下的元素分成两部分。
- 如果我们删除 nums[0] = 3 ，数组变为 [1,2] 。两部分和的差值为 1 - 2 = -1 。
- 如果我们删除 nums[1] = 1 ，数组变为 [3,2] 。两部分和的差值为 3 - 2 = 1 。
- 如果我们删除 nums[2] = 2 ，数组变为 [3,1] 。两部分和的差值为 3 - 1 = 2 。
两部分和的最小差值为 min(-1,1,2) = -1 。
```

```cpp
class Solution {
public:
    long long minimumDifference(vector<int>& nums) {
        int m = nums.size(), n = m / 3;
        long long lsum[m], ltotal = 0, rtotal = 0;
        multiset<long long> heap;
        for (int i = 0, sum = 0; i < m; i++) {
            ltotal += nums[i];
            lsum[i] = 0;
            heap.insert(-nums[i]);
            if (heap.size() < n) continue;
            if (heap.size() > n) {
                sum -= *(heap.begin());
                heap.erase(heap.begin());
            }
            lsum[i] = sum;
        }
        heap.clear();
        long long ans = INT_MAX;
        for (int i = m - 1, sum = 0; i >= n; i--) {
            ltotal -= nums[i];
            rtotal += nums[i];
            heap.insert(nums[i]);
```

```
            if (heap.size() < n) continue;
            if (heap.size() > n) {
                sum += *(heap.begin());
                heap.erase(heap.begin());
            }
            ans = min(ans, (ltotal - lsum[i - 1]) - (rtotal - sum));
        }
        return ans;
    }
};
```

2166. 设计位集

位集 Bitset 是一种能以紧凑形式存储位的数据结构。

请你实现 `Bitset` 类。

- `Bitset(int size)` 用 `size` 个位初始化 Bitset，所有位都是 `0` 。
- `void fix(int idx)` 将下标为 `idx` 的位上的值更新为 `1` 。如果值已经是 `1` ，则不会发生任何改变。
- `void unfix(int idx)` 将下标为 `idx` 的位上的值更新为 `0` 。如果值已经是 `0` ，则不会发生任何改变。
- `void flip()` 翻转 Bitset 中每一位上的值。换句话说，所有值为 `0` 的位将会变成 `1` ，反之亦然。
- `boolean all()` 检查 Bitset 中 **每一位** 的值是否都是 `1` 。如果满足此条件，返回 `true` ；否则，返回 `false` 。
- `boolean one()` 检查 Bitset 中 是否 **至少一位** 的值是 `1` 。如果满足此条件，返回 `true` ；否则，返回 `false` 。
- `int count()` 返回 Bitset 中值为 1 的位的 **总数** 。
- `String toString()` 返回 Bitset 的当前组成情况。注意，在结果字符串中，第 `i` 个下标处的字符应该与 Bitset 中的第 `i` 位一致。

示例：

```
输入
["Bitset", "fix", "fix", "flip", "all", "unfix", "flip", "one", "unfix", "count", "toString"]
[[5], [3], [1], [], [], [0], [], [], [0], [], []]
输出
[null, null, null, null, false, null, null, true, null, 2, "01010"]

解释
Bitset bs = new Bitset(5); // bitset = "00000".
bs.fix(3);     // 将 idx = 3 处的值更新为 1 ，此时 bitset = "00010" 。
bs.fix(1);     // 将 idx = 1 处的值更新为 1 ，此时 bitset = "01010" 。
bs.flip();     // 翻转每一位上的值，此时 bitset = "10101" 。
bs.all();      // 返回 False ，bitset 中的值不全为 1 。
bs.unfix(0);   // 将 idx = 0 处的值更新为 0 ，此时 bitset = "00101" 。
bs.flip();     // 翻转每一位上的值，此时 bitset = "11010" 。
bs.one();      // 返回 True ，至少存在一位的值为 1 。
bs.unfix(0);   // 将 idx = 0 处的值更新为 0 ，此时 bitset = "01010" 。
bs.count();    // 返回 2 ，当前有 2 位的值为 1 。
```

```
bs.toString(); // 返回 "01010" ，即 bitset 的当前组成情况。
```

```
class Bitset {
public:
    int size, base, *data, n, cnt;
    Bitset(int size) : size(size), base(30), cnt(0) {
        n = size / base + (size % base != 0);
        data = new int[n];
        memset(data, 0, sizeof(int) * n);
    }

    void fix(int idx) {
        int x = idx / base, y = idx % base;
        if ((data[x] & (1 << y)) == 0) {
            cnt += 1;
            data[x] |= (1 << y);
        }
        return ;
    }

    void unfix(int idx) {
        int x = idx / base, y = idx % base;
        if ((data[x] & (1 << y))) {
            cnt -= 1;
            data[x] ^= (1 << y);
        }
        return ;
    }

    void flip() {
        for (int i = 0; i < n; i++) data[i] = ~data[i];
        cnt = size - cnt;
        return ;
    }

    bool all() {
        return count() == size;
    }

    bool one() {
        return count() != 0;
    }

    int count() {
        return cnt;
    }

    string toString() {
        int m = size / base, rest = size % base;
```

```
        string s = "";
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < base; j++) {
                if ((data[i] & (1 << j))) s += '1';
                else s += '0';
            }
        }
        for (int i = 0; i < rest; i++) {
            if ((data[n - 1] & (1 << i))) s += '1';
            else s += '0';
        }
        return s;
    }
};

/**
 * Your Bitset object will be instantiated and called as such:
 * Bitset* obj = new Bitset(size);
 * obj->fix(idx);
 * obj->unfix(idx);
 * obj->flip();
 * bool param_4 = obj->all();
 * bool param_5 = obj->one();
 * int param_6 = obj->count();
 * string param_7 = obj->toString();
 */
```