A mutsumi的质数合数

数学

做法一：观察数据范围为 1 到 100 ，因此我们可以直接手算或用程序算出所有的质数和合数，然后算出质数数量和合数数量，最后相加。

做法二：根据质数和合数的性质，只有 1 不是质数，也不是合数，因此质数数量加合数数量就是数字个数减 1 的个数。

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

using namespace std;

int main(){
    int n;
    cin >> n;
    int ans = 0;
    for(int i = 1; i <= n; i++){
        int x;
        cin >> x;
        if(x != 1) ans++;
    }
    cout << ans << endl;
}
```

B tomorin的字符串迷茫值

数数，字符串

首先，只有"mygo"，"m_ygo"，"my_go"，"myg_o"，"m_y_go"，"m_yg_o"，"my_g_o"，"m_y_g_o"这 8 种子串可以通过删除得到"mygo"子串。

那么，我们可以枚举上述 8 种子串，子串内的删除方法是唯一确定的，子串两边可以任意删除，这个子串对答案的贡献就是两边任意删除的方案数相乘。而子串两边任意删除的方案数可以通过动态规划预处理（其实是斐波那契数列）。

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

using namespace std;

const int M = 1e9 + 7;

int main(){
    string s;
    cin >> s;
    int n = s.size();
    s = " " + s + "BanG Dream! It's MyGO!!!!!";
    vector<int> f(n + 1, 1);
```

```
    f[1] = 2;
    for(int i = 2; i <= n; i++){
        f[i] = (f[i - 1] + f[i - 2]) % M;
    }
    vector<string> t = {"mygo", "m ygo", "my go", "myg o", "m y go", "m yg o",
"my g o", "m y g o"};
    int ans = 0;
    for(int i = 1; i <= n; i++){
        for(auto &j : t){
            auto x = s.substr(i, j.size());
            auto check = [&](auto s, auto t){
                for(int i = 0; i < s.size(); i++){
                    if(t[i] == ' ') continue;
                    if(s[i] != t[i]) return 0;
                }
                return 1;
            };
            if(check(x, j)) ans = (ans + 1ll * f[i - 1] * f[n - (i + j.size() -
1)] % M) % M;
        }
    }
    cout << ans << endl;
}
```

C anon的私货

贪心

一个贪心的想法是在第 1 个数字前面夹带 $a_1 - 1$ 个 0，然后 $a_1^{'}$ 变成 1 。接下来第两个数字开始，在两个数字之间夹带 $min(a_{i-1}^{'}, a_i) - 1$ 个 0 （计为 t ），并且将 a_i 修改为 $a_i^{'} = a_i - t$ 。最后在最后一个数字后夹带 $a_n^{'} - 1$ 个 0 。

证明：显然，易证，不会，看不懂，能过就是对的。

8:38:21

就是，贪心做法是 (这里 bi=ai-1, b(n+1)=+inf)
v0=b1
for i=1..n:
  vi=min(bi-v(i-1), b(i+1))
  ui = u(i-1)+vi
print un

然后有对于 1<=i<=n，x(i-1)+xi<=bi

对于 yi=\sum_{j<=i} xj，首先
y0=x0<=b1=u0
y1=x0+x1<=b1=u1

8:42:45

然后假设对 k<=i-1 都已经证明了 yk<=uk,
yi = y(i-2)+x(i-1)+xi = y(i-1)+xi <= min(y(i-2)+bi, y(i-1)+b(i+1))

y(i-2)<=u(i-2)=u(i-1)-v(i-1)
y(i-1)<=u(i-1)

所以从 min 里面提出一个 u(i-1)，就是
yi<=u(i-1)+min(bi-v(i-1), b(i+1))

注意到这个 min(..) 就是 vi 的定义，所以 yi<=u(i-1)+vi=ui

所以就证明了对于所有 i，yi<=ui，特别的 yn=x0+x1+...+xn<=un，也就是 un 是 0 的数量的上界

考虑到 v(i-1)+vi<=bi 所以 xi=vi 时可以达到上界，所以 un 就是最大值

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

using namespace std;

int main(){
    int n;
    cin >> n;
    vector<int> a(n + 2);
    auto ans = 0ll;
    for(int i = 1; i <= n; i++){
        cin >> a[i];
```

```
    }
    a[0] = a[n + 1] = 2e9;
    for(int i = 1; i <= n; i++){
        int l = min(a[i], a[i - 1]);
        ans += l - 1;
        a[i] -= l - 1;
        a[i - 1] -= l - 1;
        int r = min(a[i], a[i + 1]);
        ans += r - 1;
        a[i] -= r - 1;
        a[i + 1] -= r - 1;
    }
    cout << ans << endl;
}
```

D soyorin的树上剪切

贪心，图论，搜索，DP

首先算出 s 到 t 的经过的边数，计为 x 。

1) 在 k 小于 x 时，我们可以从大到小贪心删掉经过的边中边权最大的边。

2) 在 k 等于 x 时，我们留下了一条经过的边中，边权最小的边，并且剩余了一次操作。

3) 在 k 等于 $x+1$ 时，根据上一种情况，剩余了两次操作，我们可以通过两次操作把距离 s 或 t 不超过 1 的边置换到 s 和 t 之间。

如图，先选择 $(1,3)$ ，删除 $(1,2)$ 连接 $(3,2)$ ，再选择 $(3,2)$ ，删除 $(3,1)$ 连接 $(2,1)$ ，此时 1 到 2 的距离为 1 。

4) 当 k 大于 $x+1$ 时，我们可以留两次操作用于上述置换，多余操作（计为 y 次操作）可以将距离 s 或 t 不超过 $y+1$ 的节点缩到距离 s 或 t 不超过1的位置（类似第一类操作）。

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

using namespace std;

int main(){
    int n, s, t;
    cin >> n >> s >> t;
    vector ve(n + 1, vector<pair<int, int>>());
    for(int i = 1; i < n; i++){
        int u, v, w;
        cin >> u >> v >> w;
        ve[u].push_back({v, w});
        ve[v].push_back({u, w});
    }
    vector<int> c(n + 1);
    c[t] = 1;
    vector<int> dis;
    auto dfs = [&](auto dfs, int u, int fa) -> void{    //先染红必经之路
        if(c[u]) return;
        for(auto &[i, w] : ve[u]){
            if(i == fa) continue;
```

```
            dfs(dfs, i, u);
            if(c[i]){
                c[u] = 1;
                dis.push_back(w);
            }
        }
    };
    dfs(dfs, s, 0);
    for(int i = 1; i <= n; i++){            //把除s外所有红点的相邻非红色点加入s的邻边
        if(c[i] && i != s){
            for(auto &[j, w] : ve[i]){
                if(c[j]) continue;
                ve[s].push_back({j, w});
            }
        }
    }
    vector<int> dp(n + 1, 1e9), deep(n + 1);        //dp[i]为距离s不超过i，最短的路径长度
    auto dfs1 = [&](auto dfs1, int u, int fa) -> void{
        for(auto &[i, w] : ve[u]){
            if(i == fa) continue;
            if(c[i]) continue;
            deep[i] = deep[u] + 1;
            dp[deep[i]] = min(dp[deep[i]], w);
            dfs1(dfs1, i, u);
        }
    };
    dfs1(dfs1, s, 0);
    sort(dis.begin(), dis.end());
    auto ans = accumulate(dis.begin(), dis.end(), 0ll);
    int k = dis.size();
    dis.push_back(0);
    while(dis.size() > 1){          //先删必经路径上最大的
        ans -= dis.back();
        dis.pop_back();
        cout << ans << " ";
    }
    for(int i = k; i <= n; i++){            //经过两次额外交换，能换进来的最小的边
        ans = min(ans, 1ll * dp[i - k]);
        cout << ans << " ";
    }
    cout << endl;
}
```

E soyorin的数组操作（easy）

贪心

如果数组长度是偶数，则一定可以，每次选择整个数组就可以使相邻两项后一项减前一项的差值增加 1 。

如果数组长度是奇数，则倒数第二个数最多能被操作的次数可以被计算出来，操作后，倒数第四个数最多能被操作的次数也可以被计算出来，依次类推，所有偶数位最多能被操作的次数都能计算出来。所有操作完成后，判断数组是否有序即可。

当然，我们每次操作不能直接暴力加，而是要记录一个操作次数 cnt ，遍历到 a_i 时就是加上 $cnt \times i$ 。比较推荐从后往前操作，因为前面操作次数可能比后面多。

正着也能做，但似乎会爆 $long\,long$ ，要用 $\_\_int128$ 。

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

using namespace std;

using LL = long long;

int main() {
    int T;
    cin >> T;
    while (T--) {
        int n;
        cin >> n;
        vector<LL> a(n + 1);
        for (int i = 1; i <= n; i++) {
            cin >> a[i];
        }
        LL cnt = 0;
        if (n % 2 == 0) goto yes;
        for (int i = n - 1; i >= 2; i -= 2) {
            a[i] += cnt * i;
            a[i - 1] += cnt * (i - 1);
            if (a[i] > a[i + 1]) goto no;
            LL t = (a[i + 1] - a[i]) / i;
            a[i] += t * i;
            a[i - 1] += t * (i - 1);
            cnt += t;
        }
        if (is_sorted(a.begin(), a.end())) goto yes;
        goto no;
        if (0) yes: cout << "YES" << endl;
        if (0) no: cout << "NO" << endl;
    }
}
```

F soyorin的数组操作（hard）

贪心

如果数组长度是偶数，按上述结论一定可以，那么我们每次都操作整个数组即可，这个次数也很容易确定。

如果数组长度时奇数，那么在有解的情况下，上述操作求出的次数 cnt 是一个可行的最大值，也就是说，我们的操作次数最多为 cnt ，每个数能被加到的最大数字也能计算出来。那么，我们枚举到 i 时，只需要增加 $a_{i-1} \le a_i, a_i \le a_{i+1}$ 的必要的操作次数即可。

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

using namespace std;

```

```
using LL = long long;

int main() {
    int T;
    cin >> T;
    while(T--){
        int n;
        cin >> n;
        vector<LL> a(n + 1);
        for (int i = 1; i <= n; i++) {
            cin >> a[i];
        }
        if (n & 1) {
            LL cnt = 0;
            LL t = 0;
            auto b = a;
            {
                LL cnt = 0;
                for (int i = n - 1; i >= 2; i -= 2) {
                    b[i] += cnt * i;
                    b[i - 1] += cnt * (i - 1);
                    if (b[i] > b[i + 1]) goto no;
                    LL t = (b[i + 1] - b[i]) / i;
                    b[i] += t * i;
                    b[i - 1] += t * (i - 1);
                    cnt += t;
                }
                if (!is_sorted(b.begin(), b.end())) goto no;
            }
            for (int i = n - 1; i >= 1; i -= 2) {
                a[i] += i * t;
                a[i - 1] += (i - 1) * t;
                if (a[i + 1] < a[i]) {
                    LL x = a[i] - a[i + 1];
                    cnt -= x;
                    t += x;
                    a[i - 1] += x * (i - 1);
                    a[i] += x * i;
                    a[i + 1] += x * (i + 1);
                    a[i + 2] += x * (i + 2);
                }
                else cnt += (b[i+1] - a[i]) / i;
                LL x = a[i - 1] - a[i];
                if (x > 0) {
                    cnt -= x;
                    t += x;
                    a[i] += x * i;
                    a[i - 1] += x * (i - 1);
                }
                if (cnt < 0) goto no;
            }
            cout << t << endl;
        }
        else {
            LL cnt = 0;
            for (int i = n; i >= 1; i--) {
                LL r = a[i] + i * cnt;
                LL l = a[i - 1] + (i - 1) * cnt;
```

```
                if (r < l) cnt += l - r;
            }
            cout << cnt << endl;
        }
        if (0) no: cout << -1 << endl;
    }
}
```

G sakiko的排列构造（easy）

图匹配

我们可以把数和位置分成左右两个部分，就可以在数和位置之和为质数的两端连接一条边，之后我们直接接跑网络流进行最大匹配即可。

如果 n 为偶数，若存在合法的排列，那么一定是奇数和偶数位置、偶数和奇数位置进行配对。若存在奇数和奇数位置配对，那么一定会存在偶数和偶数位置配对，那么构造一定会失败。那么奇数和偶数位置、偶数和奇数位置进行配对一定是对称的，因此我们其实只需要将奇数和偶数分成两边，这样进行最大匹配会更快且更方便。

如果 n 为奇数，那么在奇数和偶数位置、偶数和奇数位置配对的前提下，一定会有一个奇数和奇数位置配对，这个奇数和位置之和一定是偶数，是质数的偶数有且仅有一个 2，因此这个奇数和奇数位置必须都是 1。在排除1的情况下，直接按 n 是偶数进行处理即可。

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

using namespace std;

template <typename T>
class Dinic{
public:
    struct edge{
        int to;         //到达点
        T flow;         //边权流量
        int re;         //反向边下标
    };
    vector<vector<edge>> ve;
    vector<vector<int>> true_edge;
    vector<int> deep, cur;      //深度，以及弧优化
    int n, S, E;
    const T inf = INT_MAX;

    Dinic(int n, int S, int E) : n(n) , S(S) , E(E){
        ve.resize(n + 1);
    }

    void add(int u, int v, T w){
        true_edge.push_back({u, v, (int)ve[u].size()}); //正向边及下标，用于求最小割
        ve[u].push_back({v, w, (int)ve[v].size()});     //正向边
        ve[v].push_back({u, 0, (int)ve[u].size()-1});       //反向边
    }

    bool bfs(){
```

```
        deep = move(vector<int>(n + 1, 0));         //初始化，因为每次 bfs 后 deep 不一定相同
        cur  = move(vector<int>(n + 1, 0));         //弧优化归 0

        queue<int> q;
        q.push(S);
        deep[S] = 1;
        while(q.size()){
            auto u = q.front();
            q.pop();
            if(u == E) return true;                 //可达，则有增广路，存在流量
            for(auto &[i, j, k] : ve[u]){
                if(deep[i] || !j) continue;
                deep[i] = deep[u] + 1;
                q.push(i);
            }
        }
        return false;
    }

    T dfs(int u, T in){
        if(u == E) return in;                           //增广路最终的有效流量

        T out = 0;                                  //流出的流量
        for(int i = cur[u]; i < ve[u].size(); i++){     //从还有流量的边开始走
            auto &[v, flow, re] = ve[u][i];
            cur[u] = i;                             //更新弧优化
            if(!flow || deep[v] != deep[u] + 1) continue;
            T res = dfs(v, min(in , flow));         //向下一层流量限制为获得的流量和可输出的流量中小的那一个
            flow -= res;                            //本条边可输出的流量减少了 res
            ve[v][re].flow += res;                  //反向路径可输出流量增加了 res
            in -= res;                              //获得的流量消耗了 res
            out += res;                             //当前结点可输出的流量增加了 res
            if(!in) break;                          //获得流量消耗完了，没必要往下走了
        }
        if(!out) deep[u] = 0;                       //当前结点输出不了流量了，下次不用进来了
        return out;                                 //返回输出的流量
    }

    T max_flows(){
        T sum = 0;
        while(bfs()){
            sum += dfs(S,inf);
        }
        return sum;
    }

    vector<pair<int, int>> min_cuts(){
        vector<pair<int, int>> ans;
        for(auto &v : true_edge){                   //v[0]~v[2] 分别记录有向边起点，终点，边的下标
            if(!ve[v[0]][v[2]].flow)                //当正向边流量为0时，说明是最小割
                ans.push_back({v[0], v[1]});
        }
        return ans;
```

```
    }
};

int main(){
    int n;
    cin >> n;
    Dinic<int> dinic(n + 2, 0, n + 1);
    int start = 1;
    int d = -1;
    if(n & 1){
        start = 3;
        d = 1;
    }
    for(int i = 2; i <= n; i += 2){
        dinic.add(0, i, 1);
        dinic.add(i + d, n + 1, 1);
        for(int j = start; j <= n; j += 2){
            int x = i + j;
            int isp = 1;
            for(int i = 3; i * i <= x; i++){
                if(x % i == 0) isp = 0;
            }
            if(isp) dinic.add(i, j, 1);
        }
    }
    assert(dinic.max_flows() == n / 2);
    auto v = dinic.min_cuts();
    vector<int> a(n + 1, 1);
    for(auto &[x, y] : v){
        if(x <= n && x > 0 && y <= n && y > 0){
            a[x] = y;
            a[y] = x;
        }
    }
    for(int i = 1; i <= n; i++){
        cout << a[i] << " ";
    }
    cout << endl;
}
```

H sakiko的排列构造（hard）

打表

1） 若 n 为 2 ，或 n 为 4 ，由于 3 和 5 都是质数，那么我们可以发现，只需要构造 n 到 1 即可；若 n 为 3 ，或 n 为 5 ，由于 5 和 7 都是质数，那么我们可以发现，在排除掉 1 后，只需要构造 n 到 2 即可。也就是说，如果 $n+1, n+2$ 中有一个质数，我们就可以直接得到答案。

2） 若 n 为 8 ，由于 3 和 11 都是质数，我们可以先构造 2 到 1 ，再构造 8 到 3 即可；若 n 为 7 ，由于 5 和 11 都是质数，排除掉 1 后，我们可以先构造 3 到 2 ，再构造 7 到 4 即可。也就是说，如果能找到一个断点 x ，使得 $x+1(x+2), n+x+1$ 两个数都是质数，则我们就可以直接得到答案。

根据打表可以发现，若 n 不满足第一种情况，则一定满足第二种情况，而且断点出现的位置不会超过1000，所以可以直接暴力判断是否为质数，不用使用筛法。

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

using namespace std;

int main(){
    int n;
    cin >> n;
    auto check = [&](int x){
        for(int i = 2; i * i <= x; i++){
            if(x % i == 0) return 0;
        }
        return 1;
    };
    int t = 1;
    if(n & 1){
        cout << 1 << " ";
        t++;
    }
    for(int i = t; i < n; i++){
        if(check(t + i) && check(i + 1 + n)){
            for(int j = i; j >= t; j--){
                cout << j << " ";
            }
            for(int j = n; j > i; j--){
                cout << j << " ";
            }
            cout << endl;
            return 0;
        }
    }
    for(int i = n; i >= 1; i--){
        cout << i << " ";
    }
    cout << endl;
}
```

I rikki的最短路

分类讨论

1） rikki走到 T 的过程看不到 A ，那么先走到 T ，再走到 A ，再走到 T 。距离为 $0-T, T-A, A-T$ 。

2） rikki走到 T 的过程看到了 A ，那么先走到 A ，再走到 T 。距离为 $0-A, A-T$ 。

注意不爆 $long\ long$ 即可。

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

using namespace std;

using LL = long long;

int main(){
    LL t, a, k;
```

```
    cin >> t >> a >> k;
    LL ans = 0;
    if(t * a >= 0){
        t = abs(t);
        a = abs(a);
        if(a <= t) ans = t;
        else ans = t + (a - t) * 2;
    }
    else{
        if(a > k || a < -k) ans = abs(t) * 3 + abs(a) * 2;
        else ans = abs(a) * 2 + abs(t);
    }
    cout << ans << endl;
}
```

J rikki的数组陡峭值

贪心+DP

如果所有区间的交集不为空，则在交集中任取一个值陡峭值都会为0。

如果所有区间的交集为空，那么可以将其拆分成一些连续且独立的交集，并且相邻两个交集之间一定不相交（否则这两个交集可以直接合并成一个交集），那么每个交集内部的陡峭值都是0。由于相邻交集是不相交的，因此我们可以用动态规划解决交集之间的陡峭值。

现在的问题是：怎么拆分成这些连续且独立的交集？我们可以贪心的按顺序选择交集。首先，第一个区间一定在第一个交集里，第二个区间如果和第一个交集相交，则我们更新第一个交集，否则我们将第二个区间变成第二个交集，以此类推。

证明：不会。结论是对于一个区间，如果前后两个交集都已知，这个区间既跟前一个交集相交又跟后一个交集相交，但前后两个交集不相交。那么这个区间一定包含了在前后两个交集之间空出的一部分，所以这个区间放在前面和后面都不会影响两个交集之间的陡峭值，但如果放在后面，可能会导致交集变小，导致后续增加区间的数量减少。

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

using namespace std;

int main(){
    int n;
    cin >> n;
    vector<pair<int, int>> ve(n);
    for(auto &[l, r] : ve){
        cin >> l >> r;
    }
    int L = 1, R = 1e9;
    vector<vector<int>> v;
    for(auto &[l, r] : ve){
        if(L > r || R < l){
            v.push_back({L, R});
            L = l;
            R = r;
        }
        L = max(L, l);
```

```
        R = min(R, r);
    }
    v.push_back({L, R});
    vector dp(v.size(), vector(2,0ll));
    for(int i = 1; i < v.size(); i++){
        dp[i][0] = min(dp[i - 1][0] + abs(v[i][0] - v[i - 1][0]), dp[i - 1][1] +
abs(v[i][0] - v[i - 1][1]));
        dp[i][1] = min(dp[i - 1][0] + abs(v[i][1] - v[i - 1][0]), dp[i - 1][1] +
abs(v[i][1] - v[i - 1][1]));
    }
    cout << min(dp.back()[0], dp.back()[1]) << endl;
}
```

K soyorin的通知

DP

由于人数只有1000，那么 b_i 实际有效的范围只有1000左右，并且，soyorin至少要花一次 p 的代价将消息通知给 1 个人，然后再让这个人去将消息通知给剩下的 $n-1$ 个人。

那么问题就转化成了：将消息通知给 $n-1$ 个人的最小代价，将消息通知给 b_i 个人需要花费 a_i 的代价，且 a_i, b_i 能用多次，也就是一个完全背包。

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

using namespace std;

int main() {
    int n, p;
    cin >> n >> p;
    vector<int> dp(n + 1, 1e9);
    dp[0] = 0;
    for (int i = 1; i <= n; i++) {
        int a, b;
        cin >> a >> b;
        if (b >= n) b = n - 1;
        for (int j = 0; j <= n; j++) {
            dp[j] = min(dp[j], dp[max(0, j - b)] + a);
        }
    }
    int ans = 1e9;
    for (int i = 0; i < n; i++) {
        ans = min(ans, dp[i] + (n - i) * p);
    }
    cout << ans << endl;
}
```

L anon的星星

数学

做法一：直接枚举赢和输的场数即可。

做法二：假设赢了 a 局，输了 b 局，那么 $a+b=n, a-b=x$ ，联立一下就是 $2a=n+x, a=(n+x)/2, b=n-a$ ，直接解方程即可。

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

using namespace std;

int main() {
    int n, x;
    cin >> n >> x;
    if((n + x) & 1) cout << -1 << endl;
    else{
        int t = n - abs(x) >> 1;
        if(x > 0) cout << n-t << " " << t << endl;
        else cout << t << " " << n - t <<endl;
    }
}
```

M mutsumi的排列连通

分类讨论

1）如果 $a_i=b_i$ ，且 i 不是 1 或 n ，那么只操作一次即可。

2）如果 $a_i=b_{i+1}$ 或 $a_i=b_{i-1}$ （都不越界），那么只操作一次即可。

3）如果上述两个条件都不满足，且 n 大于 2 ，则操作两次即可（把 a_2, b_2 都删除即可）。

4）如果 n 等于 2 ，若不满足第二个条件，则无解。

5）如果 n 等于 1 ，则无解。

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

using namespace std;

int main(){
    int T;
    cin >> T;
    while(T--){
        int n;
        cin >> n;
        vector<int> a(n + 2), b = a;
        for(int i = 1; i <= n; i++){
            cin >> a[i];
        }
        for(int i = 1; i <= n; i++){
            cin >> b[i];
        }
        if(n == 1) cout << -1 << endl;
        else if(n == 2){
            if(a[1] == b[1]) cout << -1 << endl;
            else cout << 1 << endl;
```

```
        }
        else{
            int tar = 0;
            for(int i = 1; i <= n; i++){
                if(a[i] == b[i] && i != 1 && i != n) tar++;
                if(a[i] == b[i - 1]) tar++;
                if(a[i] == b[i + 1]) tar++;
            }
            if(tar) cout << 1 << endl;
            else cout << 2 << endl;
        }
    }
}
```

花絮

这里是出题人——网恋粉毛被骗一万的世界第一可爱的毒瘤王嘤嘤

由于为了题目简洁，导致有些题用了专业名词，但是解释并不到位，非常抱歉，下次不解释了QAQ

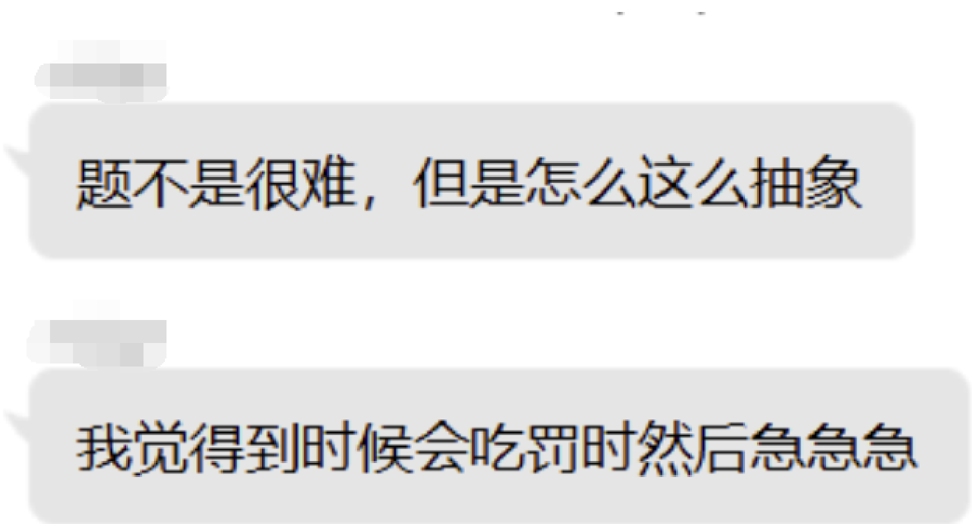
以及 I 题阅读理解有点抽象，非常抱歉TAT

本场比赛主要以思维为主，除D的DFS和K的DP外，几乎没有考察算法，更没有高深数据结构，对学习算法和数据结构较多的选手可能并不友好，反而是一些真正的萌新有着不错的发挥。然后榜就被带歪了，尤其是刚开始全在冲CEGH，没人做I和M，后期K的榜也有点歪。

很喜欢emo哥哥的一句话：选手一整场都能有事做，总有一款适合你的牢

求求你去看《BanG Dream! It's MyGO!!!!!》吧！只要是我能做的，我什么都愿意做！

本场比赛的出题费将资助嘤嘤买MyGO!!!!!的BD（什么？题出的太差不发出题费了？）



看了眼嘤嘤的防江莉题

我表示甘拜下风

以后要多向嘤嘤学习

小沙：智乃是我师傅
智乃：嘤嘤是我师傅

我毒瘤都是学的

嘤嘤是

浑然天成