afy
Codelibrary

目录

1 Ds

1.1 HLD

```
struct HLD {
    int n;
    vector<int> siz, top, dep, parent, l, r, hson, seq;
    vector<vector<int>> adj;
    int idx;

    HLD() {}
    HLD(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        siz.resize(n + 1);
        top.resize(n + 1);
        dep.resize(n + 1);
        parent.resize(n + 1);
        l.resize(n + 1);
        r.resize(n + 1);
        hson.resize(n + 1);
        seq.resize(n + 1);
        idx = 0;
        adj.assign(n + 1, {});
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
    }
    void work(int root = 1) {
        top[root] = root;
        dep[root] = 1;
        parent[root] = -1;
        dfs1(root, -1);
        dfs2(root, root);
    }
    void dfs1(int u, int f) {  // 搞fa,dep,son
        if (parent[u] != -1) {
            adj[u].erase(find(alls(adj[u]), parent[u]));
        }
        siz[u] = 1;
        for (int v : adj[u]) {
            if (v == f)
                continue;
            parent[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v, u);
            siz[u] += siz[v];
            if (siz[hson[u]] < siz[v])
                hson[u] = v;
        }
    }
    void dfs2(int u, int t) {  // 搞top
        top[u] = t;            // 记录链头
        l[u] = idx++;
        seq[l[u]] = u;
        if (!hson[u])
            return;        // 无重儿子
        dfs2(hson[u], t);  // 搜重儿子
        for (int v : adj[u]) {
            if (v == parent[u] || v == hson[u])
                continue;
            dfs2(v, v);  // 搜轻儿子
        }
        r[u] = idx;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = parent[top[u]];
            } else {
                v = parent[top[v]];
            }
        }
        return dep[u] < dep[v] ? u : v;
    }

    int dist(int u, int v) {
        return dep[u] + dep[v] - 2 * dep[lca(u, v)];
    }

    int jump(int u, int k) {
        if (dep[u] < k) {
            return -1;
        }
```

```
    int d = dep[u] - k;  // 目标节点的深度

    while (dep[top[u]] > d) {  // 不在当前链上
        u = parent[top[u]];    // 跳链头的父亲
    }
    // 同一条链上dfs序连续
    return seq[l[u] - dep[u] + d];
}

bool isAncester(int u, int v) {  // 判断u是不是v的祖先
    return l[u] <= l[v] && l[v] < r[u];
}

int rootedParent(int u, int v) {  // u为根的时候，v的父亲节点
    swap(u, v);
    if (u == v) {
        return u;
    }
    // v为根的时候，u的父亲节点
    if (!isAncester(u, v)) {  // u不是v的祖先
        return parent[u];
    }
    // u是v的祖先。现在v为根，u的父亲是(dfs序>=v)的那个节点
    auto it = upper_bound(adj[u].begin(), adj[u].end(), v, [&](int x, int y)
        {
        return l[x] < l[y];
    });
    it--;  // 找到第最后一个小于vd的dfs序节点
    return *it;
}

int rootedSize(int u, int v) {  // u为根的时候，v的子树大小
    if (u == v) {
        return n;
    }
    if (!isAncester(v, u)) {  // 如果v不是u的祖先
        return siz[v];
    }
    // v是u祖先。找到u为根的时候v的父亲。总数减去包含u的那部分
    return n - siz[rootedParent(u, v)];
}

int rootedLca(int a, int b, int c) {
    return lca(a, b) ^ lca(b, c) ^ lca(c, a);
}
```

```
    }
};
```

1.2 LCA

```
struct Blca {
    struct edge {
        int v = 0, w = 1;
    };
    int n;
    const static int len = __lg(N);
    vector<vector<edge>> e;
    vector<int> dep, dw, sz;
    vector<array<int, len + 1>> st;
    Blca() {};
    Blca(int n) {
        init(n);
    }
    void init(int n1) {
        n = n1;
        e.resize(n + 1);
        dep.resize(n + 1);
        dw.resize(n + 1);
        sz.resize(n + 1);
        st.resize(n + 1);
    }
    void add(int u, int v, int w = 1) {
        e[u].emplace_back(v, w);
    }
    void dfs(int u, int fa) {
        dep[u] = dep[fa] + 1;
        st[u][0] = fa;
        sz[u] = 1;
        for (int i = 1; i <= len; i++) st[u][i] = st[st[u][i - 1]][i - 1];
        for (auto [v, w] : e[u]) {
            deb(u, v);
            if (v == fa)
                continue;
            dw[u] = dw[v] + w;
            dfs(v, u);
            sz[u] += sz[v];
        }
```

```
    }
    int lca(int x, int y) {
        if (dep[x] < dep[y])
            swap(x, y);
        for (int i = len; i >= 0; i--) {
            if (dep[st[x][i]] >= dep[y])
                x = st[x][i];
        }
        // 跳到相同深度
        if (x == y)
            return y;
        // 提提前判本身就是祖先关系
        for (int i = len; i >= 0; i--) {
            if (st[x][i] != st[y][i]) {
                x = st[x][i];
                y = st[y][i];
            }
        }
        // 倍增一起向上跳，直到父亲就是答案
        return st[x][0];
    }
    int dis(int u, int v) {
        return dep[u] + dep[v] - 2 * dep[lca(u, v)];
    }
    int jump(int x, int k) {  // k级祖先
        for (int i = len; i >= 0; i--)
            if ((k >> i) & 1)
                x = st[x][i];
        return x;
    }
};
```

1.3 LCARMQ

```
struct LcaRmq {
    int n;
    vector<vector<int>> &adj;
    int root, tot;
    vector<int> dfn, ol, lg2, dep;
    vector<vector<int>> st;
    LcaRmq(int n_, auto &g_, auto r = 1) : n(n_), adj(g_), root(r) {
        ol.resize(n << 1);
        dfn.resize(n + 1);
        dep.resize(n + 1);
        lg2.resize(n << 1);
        tot = 0;
        auto dfs = [&](int x, int fa, auto dfs) -> void {
            ol[++tot] = x;
            dfn[x] = tot;
            dep[x] = dep[fa] + 1;
            for (auto y : adj[x]) {
                if (y == fa)
                    continue;
                dfs(y, x, dfs);
                ol[++tot] = x;
            }
        };
        dfs(root, 0, dfs);
        lg2[0] = -1;
        for (int i = 1; i <= tot; i++) lg2[i] = lg2[i >> 1] + 1;
        st.assign(lg2[tot] + 1, vector<int>(n * 2, 0));
        for (int i = 1; i <= tot; i++) st[0][i] = ol[i];
        for (int j = 1; j <= lg2[tot]; j++) {
            for (int i = 1; i + (1 << j) - 1 <= tot; i++) {
                st[j][i] = dep[st[j - 1][i]] < dep[st[j - 1][i + (1 << (j - 1))
                  ]]
                               ? st[j - 1][i]
                               : st[j - 1][i + (1 << (j - 1))];
            }
        }
    }
    int lca(int u, int v) {
        u = dfn[u], v = dfn[v];
        if (u > v)
            swap(u, v);
        int d = lg2[v - u + 1];
        return dep[st[d][u]] < dep[st[d][v - (1 << d) + 1]]
                   ? st[d][u]
                   : st[d][v - (1 << d) + 1];
    }
};
```

1.4 LCA_HLD

```
struct edge {
    int v, w;
};
struct HLD {
    int n;
    vector<int> siz, top, parent, l, r, hson, dep;
    vector<vector<edge>> adj;
    int idx;
    HLD() {}
    HLD(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        siz.resize(n + 1), hson.resize(n + 1), top.resize(n + 1);
        parent.resize(n + 1);
        l.resize(n + 1), r.resize(n + 1);
        idx = 0;
        adj.resize(n + 1), dep.resize(n + 1);
        // 根据题目要求加数据结构
    }
    void addEdge(int u, int v, int w) {
        adj[u].push_back({v, w});
    }
    void work(int root = 1) {
        top[root] = root;
        parent[root] = -1;
        dep[root] = 1;
        dfs1(root, -1);
        dfs2(root, root);
    }
    void dfs1(int u, int f) {  // 搞fa,dep,son
        siz[u] = 1;
        for (auto [v, w] : adj[u]) {
            if (v == f)
                continue;
            parent[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v, u);
            siz[u] += siz[v];
            if (siz[hson[u]] < siz[v])
                hson[u] = v;
        }
    }
    void dfs2(int u, int t) {  // 搞top
        top[u] = t;            // 记录链头
        l[u] = ++idx;
        if (!hson[u]) {
            r[u] = idx;
            return;
        }  // 无重儿子
        dfs2(hson[u], t);  // 搜重儿子
        for (auto [v, w] : adj[u]) {
            if (v == parent[u] || v == hson[u])
                continue;
            dfs2(v, v);  // 搜轻儿子
        }
        r[u] = idx;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = parent[top[u]];
            } else {
                v = parent[top[v]];
            }
        }
        return dep[u] < dep[v] ? u : v;
    }
    bool isAncester(int u, int v) {  // 判断u是不是v的祖先
        return l[u] <= l[v] && r[v] <= r[u];
    }
};
```

1.5 Miller_Robin

```
using i64 = long long;
i64 mul(i64 a, i64 b, i64 m) {
    return static_cast<__int128>(a) * b % m;
}
i64 power(i64 a, i64 b, i64 m) {
    i64 res = 1 % m;
    for (; b; b >>= 1, a = mul(a, a, m))
        if (b & 1)
            res = mul(res, a, m);
    return res;
```

```
}
bool isprime(i64 n) {//log^3(n)
    if (n < 2)
        return false;
    static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    int s = __builtin_ctzll(n - 1);
    i64 d = (n - 1) >> s;
    for (auto a : A) {
        if (a == n)
            return true;
        i64 x = power(a, d, n);
        if (x == 1 || x == n - 1)
            continue;
        bool ok = false;
        for (int i = 0; i < s - 1; ++i) {
            x = mul(x, x, n);
            if (x == n - 1) {
                ok = true;
                break;
            }
        }
        if (!ok)
            return false;
    }
    return true;
}
```

1.6 divide_CDQ

```
void solve() {
    int n, m;
    cin >> n >> m;
    vector<int> a(n + 1);
    vector<int> pos(n + 1);
    for (int i = 1; i <= n; i++) cin >> a[i], pos[a[i]] = i;
    vector<int> t(n + 1);
    for (int i = 1; i <= m; i++) {
        int x;
        cin >> x;
        t[pos[x]] = i;  // x这个元素在pos[x]，我们研究下标，所以下标pos[x]在i时
          刻被删除
    }
    int cur = m;
    deb(a);
    for (int i = 1; i <= n; i++) {
        if (t[i] == 0) {
            cur++;
            t[i] = cur;
        }
    }
    deb(t);
    auto cmpx = [&](array<int, 4>& c, array<int, 4>& d) {
        return c[1] < d[1];
    };
    auto cal = [&](vector<array<int, 4>>& q) {
        vector<int> ans(n + 1);
        Fwk<int> c(n + 1);
        sort(q.begin() + 1, q.end());
        auto cdq = [&](auto self, int l, int r) {
            if (l == r)
                return;
            int mid = (l + r) >> 1;
            self(self, l, mid);
            self(self, mid + 1, r);
            int pl = l, pr = mid + 1;
            while (pl <= mid && pr <= r) {
                if (q[pl][1] < q[pr][1]) {
                    c.add(q[pl][2], 1);
                    pl++;
                } else {
                    ans[q[pr][3]] += c.sum(q[pr][2]);
                    pr++;
                }
            }
            while (pl <= mid) {
                c.add(q[pl][2], 1);
                pl++;
            }
            while (pr <= r) {
                ans[q[pr][3]] += c.sum(q[pr][2]);
                pr++;
            }
            for (int i = l; i <= mid; i++) c.add(q[i][2], -1);
            sort(q.begin() + l, q.begin() + r + 1, cmpx);
        };
        cdq(cdq, 1, n);
```

```
        return ans;
    };
    auto re = [&](int x) {
        return n + 1 - x;
    };
    vector<ll> res(n + 2);
    vector<array<int, 4>> q(n + 1);
    for (int i = 1; i <= n; i++) {
        q[i] = {re(t[i]), i, re(a[i]), i};
    }
    auto ans1 = cal(q);
    for (int i = 1; i <= n; i++) res[t[i]] += ans1[i];
    for (int i = 1; i <= n; i++) {
        q[i] = {re(t[i]), re(i), a[i], i};
    }
    auto ans2 = cal(q);
    for (int i = 1; i <= n; i++) res[t[i]] += ans2[i];
    for (int i = n; i >= 1; i--) res[i] += res[i + 1];
    for (int i = 1; i <= m; i++) cout << res[i] << endl;
}
```

1.7 divide_dot

```
struct edge {
    int v, w;
};
void solve() {
    int n, m;
    cin >> n >> m;
    vector<vector<edge>> e(n + 1);
    for (int i = 1; i <= n - 1; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        e[u].push_back({v, w});
        e[v].push_back({u, w});
    }
    vector<int> q(m + 1);
    for (int i = 1; i <= m; i++) cin >> q[i];
    //----------------------相关数据
    int mxsz = inf, rt = 0;
    vector<int> sz(n + 1);
    vector<int> d(n + 1);    // 距离
    vector<int> del(n + 1);  // 标记
    vector<int> ans(n + 1);
    map<int, int> have;
    // 求重心函数，每次需要传参数：当前剩余点数sum
    // 每次使用getroot以前要初始化mxsz
    auto getroot = [&](auto self, int u, int fa, int sum) -> void {
        sz[u] = 1;
        int tmp = 0;
        for (auto [v, w] : e[u]) {
            if (v == fa || del[v])
                continue;
            self(self, v, u, sum);
            sz[u] += sz[v];
            tmp = max(tmp, sz[v]);
        }
        tmp = max(tmp, sum - sz[u]);
        if (tmp < mxsz) {
            mxsz = tmp;
            rt = u;
        }
    };
    auto upd = [&](auto self, int u, int fa, vector<int>& tmp) -> void {
        tmp.push_back(d[u]);
        for (auto [v, w] : e[u]) {
            if (v == fa || del[v])
                continue;
            d[v] = d[u] + w;
            self(self, v, u, tmp);
        }
    };
    auto cal = [&](int u) -> void {  // 计算经过u的答案
        have[0] = 1;
        vector<int> alltmp;
        for (auto [v, w] : e[u]) {
            if (del[v])
                continue;
            d[v] = w;
            vector<int> tmp;
            upd(upd, v, u, tmp);
            // 更新答案
            for (auto ndis : tmp) {
                for (int k = 1; k <= m; k++) {
                    if (q[k] >= ndis) {
                        if (have.count(q[k] - ndis)) {
```

```
                        ans[k] = 1;
                    }
                }
            }
        }
        // 更新have
        for (auto ndis : tmp) {
            if (ndis <= inf) {  // 这里的距离带边权
                alltmp.push_back(ndis);
                have[ndis] = 1;
            }
        }
    }
    // 及时清空这一次的计算，未来还要用
    for (auto x : alltmp) have.erase(x);
};
// 分治
auto divide = [&](auto self, int u) -> void {
    cal(u);
    del[u] = 1;
    for (auto [v, w] : e[u]) {
        if (del[v])
            continue;
        mxsz = sz[v];                   // 分治递归下去
        getroot(getroot, v, v, mxsz);  // 找到新的递归重心
        self(self, rt);
    }
};
//-----------------------具体调用
mxsz = n;
getroot(getroot, 1, 1, n);    // rt更新
getroot(getroot, rt, rt, n);  // 重构sz[]
divide(divide, rt);
// 输出
for (int i = 1; i <= m; i++) {
    if (ans[i])
        cout << "AYE" << endl;
    else
        cout << "NAY" << endl;
}
}
```

1.8 dsu

```
struct DSU {
    vector<int> f, siz;

    DSU() {}
    DSU(int n) {
        init(n);
    }

    void init(int n) {
        f.resize(n + 1);
        std::iota(f.begin(), f.end(), 0);
        siz.assign(n + 1, 1);
    }

    int find(int x) {
        while (x != f[x]) {
            x = f[x] = f[f[x]];
        }
        return x;
    }
    bool same(int x, int y) {
        return find(x) == find(y);
    }

    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int size(int x) {
        return siz[find(x)];
    }
};
```

1.9 dsu_del

```
struct DSU {
    vector<int> siz;
    vector<int> f;
    vector<array<int, 2>> his;
    DSU(int n) : siz(n + 1, 1), f(n + 1) {
        iota(f.begin(), f.end(), 0);
    }
    int find(int x) {
        while (f[x] != x) {
            x = f[x];
        }
        return x;
    }
    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        if (siz[x] < siz[y]) {
            swap(x, y);
        }
        his.push_back({x, y});
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int time() {
        return his.size();
    }

    void revert(int tm) {
        while (his.size() > tm) {
            auto [x, y] = his.back();
            his.pop_back();
            f[y] = y;
            siz[x] -= siz[y];
        }
    }
};
void solve() {
    int n, m;
    cin >> n >> m;
    DSU dsu(n + 1);
    for (int i = 1; i <= m; i++) {
        int op;
        cin >> op;
        if (op == 1) {//合并x,y
            int x, y;
            cin >> x >> y;
            dsu.merge(x, y);
        } else if (op == 2)//撤回上一次操作
            dsu.revert(dsu.his.size() - 1);
        else {
            int x, y;
            cin >> x >> y;//查询x,y连通性
            if (dsu.find(x) == dsu.find(y))
                cout << "YES" << endl;
            else
                cout << "NO" << endl;
        }
    }
}
```

1.10 dsu_per

```
struct DSU {
    vector<int> siz;
    vector<int> f;
    vector<array<int, 2>> his;

    DSU(int n) : siz(n + 1, 1), f(n + 1) {
        iota(f.begin(), f.end(), 0);
    }

    int find(int x) {
        while (f[x] != x) {
            x = f[x];
        }
        return x;
    }

    bool merge(int x, int y) {
```

```
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        if (siz[x] < siz[y]) {
            swap(x, y);
        }
        his.push_back({x, y});
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int time() {
        return his.size();
    }

    void revert(int tm) {
        while ((int)his.size() > tm) {
            auto [x, y] = his.back();
            his.pop_back();
            f[y] = y;
            siz[x] -= siz[y];
        }
    }
    void backlast() {
        assert(his.size());
        // if (his.size() == 0)
        //     return;
        revert(his.size() - 1);
    }
};

void solve() {
    int n, m;
    cin >> n >> m;
    vector<vector<int>> e(m + 1);
    DSU dsu(n);
    vector<array<int, 3>> type(m + 1);
    for (int i = 1; i <= m; i++) {
        cin >> type[i][0];
        if (type[i][0] == 1 || type[i][0] == 3) {
            e[i - 1].push_back(i);
            cin >> type[i][1] >> type[i][2];
        } else {
            int k;
            cin >> k;
            assert(k < i);
            e[k].push_back(i);
        }
    }
    vector<int> ans(m + 1);
    function<void(int)> dfs = [&](int u) {
        int tmp = dsu.time();

        for (auto v : e[u]) {
            // int tmp = dsu.time();
            auto [op, x, y] = type[v];
            bool flag = 0;
            if (op == 1)
                flag = dsu.merge(x, y);  // 必须合并成功才能在下面撤销
            else if (op == 3)
                ans[v] = (dsu.find(x) == dsu.find(y)) ? 1 : 0;
            dfs(v);
            // dsu.revert(tmp);
            if (op == 1 && flag)
                dsu.backlast();
        }
    };
    dfs(0);
    for (int i = 1; i <= m; i++) {
        if (type[i][0] == 3) {
            deb(i);
            cout << ans[i] << endl;
        }
    }
}
```

1.11 factorize_i64

```
using i64 = long long;//数学
vector<i64> factorize(i64 n) {  // 期望O(n^(1/4))找到非平凡有因子
    vector<i64> p;
    function<void(i64)> f = [&](i64 n) {
        if (n <= 10000) {
```

```
        for (int i = 2; i * i <= n; ++i)
            for (; n % i == 0; n /= i)
                p.push_back(i);
        if (n > 1)
            p.push_back(n);
        return;
    }
    if (isprime(n)) {
        p.push_back(n);
        return;
    }
    auto g = [&](i64 x) {
        return (mul(x, x, n) + 1) % n;
    };
    i64 x0 = 2;
    while (true) {
        i64 x = x0;
        i64 y = x0;
        i64 d = 1;
        i64 power = 1, lam = 0;
        i64 v = 1;
        while (d == 1) {
            y = g(y);
            ++lam;
            v = mul(v, abs(x - y), n);
            if (lam % 127 == 0) {
                d = gcd(v, n);
                v = 1;
            }
            if (power == lam) {
                x = y;
                power *= 2;
                lam = 0;
                d = gcd(v, n);
                v = 1;
            }
        }
        if (d != n) {
            f(d);
            f(n / d);
            return;
        }
        ++x0;
    }
    };
    f(n);
    sort(p.begin(), p.end());
    return p;
}
```

1.12 fenwick

```
template <typename T>
struct Fwk {
    int n;
    std::vector<T> a;

    Fwk(int n_ = 0) {
        init(n_);
    }

    void init(int n_) {
        n = n_;
        a.assign(n + 1, T{});
    }

    void add(int x, const T &v) {
        assert(x > 0);
        for (int i = x; i <= n; i += i & -i) {
            a[i] = a[i] + v;
        }
    }

    T sum(int x) {
        T ans{};
        assert(x <= n);
        for (int i = x; i > 0; i -= i & -i) {
            ans = ans + a[i];
        }
        return ans;
    }

    T rangeSum(int l, int r) {  // 要传入1-1
        return sum(r) - sum(l);
    }
```

```
    int select(const T &k) {  // 寻找最后一个使得前缀和小于等于 k 的位置。
        int x = 0;
        T cur{};
        for (int i = 1 << std::__lg(n); i; i /= 2) {  // GCC
            if (x + i <= n && cur + a[x + i] <= k) {
                x += i;
                cur = cur + a[x];
            }
        }
        return x;
    }
};
```

1.13 hashtree

```
struct treehash {
    int n;
    int rt = 0;
    vector<ll> h1, h2;
    vector<int> siz;
    vector<vector<int>> e;
    treehash(int n_) : n(n_), h1(n_ + 1), h2(n_ + 1), siz(n_ + 1), e(n_ + 1) {}
    ll h(ll x) {
#pragma GCC diagnostic ignored "-Woverflow"
        return x * x * x * 1237123 + 19260817;
    }

    ll f(ll x) {
        ll cur = h(x & ((1 << 31) - 1)) + h(x >> 31);
        return cur;
    }
    void add(int u, int v) {
        e[u].push_back(v);
    }
    int pos = 0, pos2 = 0, ans = 1e9;
    void getroot(int u, int fa) {
        siz[u] = 1;
        int mx = 0;
        for (auto v : e[u]) {
            if (v == fa)
                continue;
            getroot(v, u);
            siz[u] += siz[v];
            mx = max(mx, siz[v]);
        }
        mx = max(mx, n - siz[u]);
        // 维护了重心是pos
        if (mx < ans) {
            ans = mx, pos = u, pos2 = 0;
        } else if (mx == ans) {
            pos2 = u;
        }
    };
    void dfs1(int u, int fa, auto &h) {
        h[u] = 1;
        for (auto v : e[u]) {
            if (v == fa)
                continue;
            dfs1(v, u, h);
            h[u] += f(h[v]);
        }
    };
    int work() {
        getroot(rt, 0);
        dfs1(pos, 0, h1);
        if (pos2)
            dfs1(pos2, 0, h2);
        ll val = h1[pos];
        if (pos2) {
            val = max(val, h2[pos2]);
        }
        return val;
    }
};
//m颗无根无标号，n个节点的树。两个哈希值取最大的那个作为key
void solve() {
    map<ll, int> mp;
    int n, m;
    cin >> m;
    for (int j = 1; j <= m; j++) {
        cin >> n;
        treehash th(n);
        for (int i = 1; i <= n; i++) {
            int x;
            cin >> x;
            if (x == 0)
```

```
            th.rt = i;
        else
            th.add(i, x), th.add(x, i);
    }
    ll val = th.work();
    if (mp.count(val) == 0)
        mp[val] = j;
    cout << mp[val] << endl;
  }
}
```

1.14 hashtree_RT

```
#include <bits/stdc++.h>
#ifdef LOCAL
#include "debug.h"
#else
#define deb(...)
#endif
using namespace std;
#define ll long long
// #define int long long
#define ull unsigned long long
#define pii pair<int, int>
#define db double
#define baoliu(x, y) cout << fixed << setprecision(y) << x
#define endl "\n"
#define alls(x) (x).begin(), (x).end()
#define fs first
#define sec second
#define bug(x) cerr << #x << "␣=␣" << x << endl
const int N = 2e5 + 10;
const int M = 1e6 + 10;
const int inf = 0x3f3f3f3f;
const int mod = 998244353;
const double eps = 1e-8;
const double PI = acos(-1.0);
struct treehash {
    int n;
    int rt = 0;          // 注意无向树的根怎么给
    vector<ll> h1, h2;  // hash的值在longlong范围内
    vector<int> siz;
    vector<vector<int>> e;
    treehash(int n_) : n(n_), h1(n_ + 1), h2(n_ + 1), siz(n_ + 1), e(n_ + 1) {}
    ll h(ll x) {
#pragma GCC diagnostic ignored "-Woverflow"
        return x * x * x * 1237123 + 19260817;
    }

    ll f(ll x) {
        ll cur = h(x & ((1 << 31) - 1)) + h(x >> 31);
        return cur;
    }
    void add(int u, int v) {
        e[u].push_back(v);
    }
    int pos = 0, pos2 = 0, ans = 1e9;
    void getroot(int u, int fa) {  // 找树的1-2重心
        siz[u] = 1;
        int mx = 0;
        for (auto v : e[u]) {
            if (v == fa)
                continue;
            getroot(v, u);
            siz[u] += siz[v];
            mx = max(mx, siz[v]);
        }
        mx = max(mx, n - siz[u]);
        // 维护了重心是pos
        if (mx < ans) {
            ans = mx, pos = u, pos2 = 0;
        } else if (mx == ans) {
            pos2 = u;
        }
    };
    void dfs1(int u, int fa, auto &h) {
        h[u] = 1;
        for (auto v : e[u]) {
            if (v == fa)
                continue;
            dfs1(v, u, h);
            h[u] += f(h[v]);
        }
    };
    ll work() {  // 无根树先找重心
        getroot(rt, 0);
```

```
        dfs1(pos, 0, h1);
        if (pos2)
            dfs1(pos2, 0, h2);
        ll val = h1[pos];
        if (pos2) {
            val = max(val, h2[pos2]);
        }
        return val;
    }
};
void solve() {
    int n;
    cin >> n;
    treehash h1(n), h2(n);
    for (int i = 1; i <= n - 1; i++) {
        int u, v;
        cin >> u >> v;
        h1.add(u, v);
        h1.add(v, u);
    }
    for (int i = 1; i <= n - 1; i++) {
        int u, v;
        cin >> u >> v;
        h2.add(u, v);
        h2.add(v, u);
    }
    h1.dfs1(1, 0, h1.h1);
    h2.dfs1(1, 0, h2.h1);

    if (h1.h1[1] == h2.h1[1])
        cout << "Isomorphism" << endl;
    else
        cout << "No" << endl;
}
signed main() {
    cin.tie(0);
    ios::sync_with_stdio(false);
#ifdef LOCAL
    double starttime = clock();
    // freopen("in.txt", "r", stdin);
    // freopen("out.txt", "w", stdout);
#endif
    int t = 1;
    cin >> t;
    while (t--) solve();
#ifdef LOCAL
    double endtime = clock();
    cerr << "Time Used: " << (double)(endtime - starttime) / CLOCKS_PER_SEC *
      1000 << " ms" << endl;
#endif
    return 0;
}
```

1.15 k_ancestor_O(1)

```
void solve() {
    int n, q;
    cin >> n >> q;
    int rt = 0;
    vector<int> l(n + 1), r(n + 1), node(n + 1);
    int idx = 0;
    vector<int> fa(n + 1), dep(n + 1);
    vector<int> hson(n + 1), top(n + 1), len(n + 1);
    vector<vector<int>> e(n + 1);
    for (int i = 1; i <= n; i++) {
        int x;
        cin >> x;
        if (x == 0) {
            rt = i;
            continue;
        }
        e[x].push_back(i);
    }
    auto dfs1 = [&](auto self, int u, int curd = 1) -> void {
        dep[u] = curd;
        len[u] = 1;
        for (auto v : e[u]) {
            fa[v] = u;
            self(self, v, curd + 1);
            if (len[v] + 1 > len[u]) {
                hson[u] = v;
                len[u] = len[v] + 1;
            }
        }
    };
    auto dfs2 = [&](auto self, int u, int tp) -> void {
```

```
        l[u] = ++idx;
        node[idx] = u;
        top[u] = tp;
        if (hson[u])
            self(self, hson[u], tp);
        for (auto v : e[u]) {
            if (top[v])
                continue;
            self(self, v, v);
        }
        r[u] = idx;
    };
    int bei = __lg(n);
    vector<vector<int>> st(bei + 1, vector<int>(n + 1));
    vector<vector<int>> anc(n + 1), des(n + 1);
    auto work = [&](int rt) {
        dfs1(dfs1, rt);
        dfs2(dfs2, rt, rt);
        for (int i = 1; i <= n; i++) st[0][i] = fa[i];
        for (int i = 1; i <= bei; ++i) {
            for (int j = 1; j <= n; ++j) {
                st[i][j] = st[i - 1][st[i - 1][j]];
            }
        }
        for (int i = 1; i <= n; ++i) {
            if (top[i] == i) {
                for (int j = 0, p = i; j < len[i]; ++j, p = fa[p])
                    anc[i].push_back(p);
                for (int j = 0; j < len[i]; ++j)
                    des[i].push_back(node[l[i] + j]);
            }
        }
    };
    auto query = [&](int p, int k) {
        if (k == 0)
            return p;  // 特判
        int i = __lg(k), q = st[i][p];
        int tp = top[q];
        // q的k-(1<<i)级祖先小于链长，预处理了两倍链长的信息
        int d = k - (1 << i) - (dep[q] - dep[tp]);
        if (d > 0)
            return anc[tp][d];
        else
            return des[tp][-d];
    };

    for (int i = 1; i <= q; i++) {
        int x, k;
        cin >> x >> k;
        int res = query(x, k);
        cout << res << endl;
    }
}
```

1.16 segtree

```
template <class Info>
struct Segtree {
#define ls(x) x << 1
#define rs(x) (x << 1) | 1
    int n;
    vector<Info> info;
    Segtree() : n(0) {}
    Segtree(int n_, Info v_ = Info()) { init(vector<Info>(n_ + 1, v_)); }
    Segtree(vector<Info> t_) { init(t_); }
    void init(vector<Info> a)  //[1,n]
    {
        n = a.size() - 1;
        info.assign((n << 2) + 1, Info());
        function<void(int, int, int)> build = [&](int x, int l, int r) -> void {
            if (l == r) {
                info[x] = a[l];
                return;
            }
            int mid = (l + r) >> 1;
            build(ls(x), l, mid);
            1 build(rs(x), mid + 1, r);
            pushup(x);
        };
        build(1, 1, n);
    }
    void pushup(int x) { info[x] = info[ls(x)] + info[rs(x)]; }
    void update(int x, int l, int r, int p, const Info& v) {
        if (l == r) {
            info[x] = v;
            return;
```

```
    }
    int mid = (l + r) >> 1;
    if (p <= mid)
        update(ls(x), l, mid, p, v);
    else
        update(rs(x), mid + 1, r, p, v);
    pushup(x);
}
void update(int p, const Info& v) { update(1, 1, n, p, v); }
Info query(int x, int l, int r, int ql, int qr) {
    if (l > qr || r < ql)
        return Info();
    if (ql <= l && r <= qr)
        return info[x];
    int mid = (l + r) >> 1;
    return query(ls(x), l, mid, ql, qr) +
           query(rs(x), mid + 1, r, ql, qr);
}
Info query(int ql, int qr) { return query(1, 1, n, ql, qr); }
template <class F>
int findFirst(int x, int l, int r, int ql, int qr, F pred) {
    if (l > qr || r < ql || !pred(info[x]))
        return -1;
    if (l == r)
        return l;
    int mid = (l + r) >> 1;
    int res = findFirst(x << 1, l, mid, ql, qr, pred);
    if (res == -1)
        res = findFirst(x << 1 | 1, mid + 1, r, ql, qr, pred);
    return res;
}
template <class F>
int findFirst(int l, int r, F pred) {
    return findFirst(1, 1, n, l, r, pred);
}
template <class F>
int findLast(int x, int l, int r, int ql, int qr, F pred) {
    if (l > qr || r < ql || !pred(info[x]))
        return -1;
    if (l == r)
        return l;
    int mid = (l + r) >> 1;
    int res = findLast(x << 1 | 1, mid + 1, r, ql, qr, pred);
    if (res == -1)
        res = findLast(x << 1, l, mid, ql, qr, pred);
    return res;
}
template <class F>
int findLast(int l, int r, F pred) {
    return findLast(1, 1, n, l, r, pred);
}
};
struct Info {
    ll sum = 0, len = 1;
};
Info operator+(const Info& a, const Info& b) {  // 维护的信息怎么合并
    Info c = Info();
    c.sum = a.sum + b.sum;
    c.len = a.len + b.len;
    return c;
};
```

1.17 segtree_ps

```
template <class Info>
struct Segment {
    struct Node {
        int left, right;
        Info info;
        Node() : left{0}, right{0}, info{} {}
    };
    int n;
    vector<Node> t;
    Segment(int n = 0) { init(n); }
    void init(int n) {
        this->n = n;
        t.assign(1, {});
    }
    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }
    int modify(int prev, int l, int r, int x, const Info &v) {
        int curr = newNode();
        t[curr] = t[prev];
        t[curr].info.apply(v);
```

```
        if (r - l == 1) {
            return curr;
        }
        int m = (l + r) / 2;
        if (x < m) {
            t[curr].left = modify(t[prev].left, l, m, x, v);
        } else {
            t[curr].right = modify(t[prev].right, m, r, x, v);
        }
        return curr;
    }
    int modify(int prev, int x, const Info &v) { return modify(prev, 0, n, x, v)
      ; }
    Info query(int L, int R, int l, int r, int x) {
        if (r - l == 1) {
            return t[R].info - t[L].info;
        }
        int m = (l + r) / 2;
        if (x < m) {
            return query(t[L].left, t[R].left, l, m, x);
        } else {
            return query(t[L].right, t[R].right, m, r, x);
        }
    }
    Info query(int L, int R, int x) { return query(L, R, 0, n, x); }
    Info rangeQuery(int L, int R, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return t[R].info - t[L].info;
        }
        int m = (l + r) / 2;
        return rangeQuery(t[L].left, t[R].left, l, m, x, y) + rangeQuery(t[L].
          right, t[R].right, m, r, x, y);
    }
    Info rangeQuery(int L, int R, int l, int r) { return rangeQuery(L, R, 0, n,
      l, r); }
};

struct Info {
    int x;
    Info(int x = 0) : x{x} {}
    void apply(const Info &v) { x += v.x; }
};
Info operator+(Info lhs, Info rhs) {
    Info res = lhs;
    res.x += rhs.x;
    return res;
}
Info operator-(Info lhs, Info rhs) {
    Info res = lhs;
    res.x -= rhs.x;
    return res;
}
```

1.18 segtreelazy0base

```
template<class Info, class Tag>
struct LazySegmentTree {
    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
    LazySegmentTree() : n(0) {}
    LazySegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    LazySegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        tag.assign(4 << std::__lg(n), Tag());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
```

```
        build(2 * p + 1, m, r);
        pull(p);
    };
    build(1, 0, n);
}
void pull(int p) {
    info[p] = info[2 * p] + info[2 * p + 1];
}
void apply(int p, const Tag &v) {
    info[p].apply(v);
    tag[p].apply(v);
}
void push(int p) {
    apply(2 * p, tag[p]);
    apply(2 * p + 1, tag[p]);
    tag[p] = Tag();
}
void modify(int p, int l, int r, int x, const Info &v) {
    if (r - l == 1) {
        info[p] = v;
        return;
    }
    int m = (l + r) / 2;
    push(p);
    if (x < m) {
        modify(2 * p, l, m, x, v);
    } else {
        modify(2 * p + 1, m, r, x, v);
    }
    pull(p);
}
void modify(int p, const Info &v) {
    modify(1, 0, n, p, v);
}
Info rangeQuery(int p, int l, int r, int x, int y) {
    if (l >= y || r <= x) {
        return Info();
    }
    if (l >= x && r <= y) {
        return info[p];
    }
    int m = (l + r) / 2;
    push(p);
    return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y)
      ;
}
Info rangeQuery(int l, int r) {
    return rangeQuery(1, 0, n, l, r);
}
void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
    if (l >= y || r <= x) {
        return;
    }
    if (l >= x && r <= y) {
        apply(p, v);
        return;
    }
    int m = (l + r) / 2;
    push(p);
    rangeApply(2 * p, l, m, x, y, v);
    rangeApply(2 * p + 1, m, r, x, y, v);
    pull(p);
}
void rangeApply(int l, int r, const Tag &v) {
    return rangeApply(1, 0, n, l, r, v);
}
template<class F>
int findFirst(int p, int l, int r, int x, int y, F pred) {
    if (l >= y || r <= x || !pred(info[p])) {
        return -1;
    }
    if (r - l == 1) {
        return l;
    }
    int m = (l + r) / 2;
    push(p);
    int res = findFirst(2 * p, l, m, x, y, pred);
    if (res == -1) {
        res = findFirst(2 * p + 1, m, r, x, y, pred);
    }
    return res;
}
template<class F>
int findFirst(int l, int r, F pred) {
    return findFirst(1, 0, n, l, r, pred);
}
template<class F>
int findLast(int p, int l, int r, int x, int y, F pred) {
```

```
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        push(p);
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};

struct Tag {
    i64 a = 0, b = 0;
    void apply(Tag t) {
        a = std::min(a, b + t.a);
        b += t.b;
    }
};

int k;

struct Info {
    i64 x = 0;
    void apply(Tag t) {
        x += t.a;
        if (x < 0) {
            x = (x % k + k) % k;
        }
        x += t.b - t.a;
    }
};
Info operator+(Info a, Info b) {
    return {a.x + b.x};
}
```

1.19 segtreelazy1base

```
template <class Info, class Tag>
struct LazySegmentTree {
    int n;
    vector<Info> info;
    vector<Tag> tag;
    LazySegmentTree() : n(0) {}
    LazySegmentTree(int n_, Info v_ = Info()) { init(vector<Info>(n_ + 1, v_));
      }
    LazySegmentTree(vector<Info> t_) { init(t_); }
    void init(vector<Info> a)  //[1,n]
    {
        n = a.size() - 1;
        info.assign((n << 2) + 1, Info());
        tag.assign((n << 2) + 1, Tag());
        function<void(int, int, int)> build = [&](int x, int l, int r) -> void {
            if (l == r) {
                info[x] = a[l];
                return;
            }
            int mid = l + r >> 1;
            build(x << 1, l, mid);
            build(x << 1 | 1, mid + 1, r);
            pushup(x);
        };
        build(1, 1, n);
    }
    void pushup(int x) { info[x] = info[x << 1] + info[x << 1 | 1]; }
    void apply(int p, const Tag& v) {
        info[p].apply(v);  // 标记更新自己
        tag[p].apply(v);   // 下传标记
    }
    void pushdown(int x) {
        apply(x << 1, tag[x]);
        apply(x << 1 | 1, tag[x]);
        tag[x] = Tag();
    }
    void update(int x, int l, int r, int p, const Info& v) {
        if (l == r) {
            info[x] = v;
            return;
        }
        int mid = l + r >> 1;
```

```
    pushdown(x);
    if (p <= mid)
        update(x << 1, l, mid, p, v);
    else
        update(x << 1 | 1, mid + 1, r, p, v);
    pushup(x);
}
void update(int p, const Info& v) { update(1, 1, n, p, v); }
Info query(int x, int l, int r, int ql, int qr) {
    if (l > qr || r < ql)
        return Info();
    if (ql <= l && r <= qr)
        return info[x];
    int mid = l + r >> 1;
    pushdown(x);
    return query(x << 1, l, mid, ql, qr) +
           query(x << 1 | 1, mid + 1, r, ql, qr);
}
Info query(int ql, int qr) { return query(1, 1, n, ql, qr); }
void rangeupdate(int x, int l, int r, int ql, int qr, const Tag& v) {
    if (l > qr || r < ql)
        return;
    if (ql <= l && r <= qr) {
        apply(x, v);
        return;
    }
    int mid = l + r >> 1;
    pushdown(x);
    rangeupdate(x << 1, l, mid, ql, qr, v);
    rangeupdate(x << 1 | 1, mid + 1, r, ql, qr, v);
    pushup(x);
}
void rangeupdate(int ql, int qr, const Tag& v) {
    rangeupdate(1, 1, n, ql, qr, v);
}
template <class F>
int findFirst(int x, int l, int r, int ql, int qr, F pred) {
    if (l > qr || r < ql || !pred(info[x]))
        return -1;
    if (l == r)
        return l;
    int mid = l + r >> 1;
    pushdown(x);
    int res = findFirst(x << 1, l, mid, ql, qr, pred);
    if (res == -1)
        res = findFirst(x << 1 | 1, mid + 1, r, ql, qr, pred);
    return res;
}
template <class F>
int findFirst(int l, int r, F pred) {
    return findFirst(1, 1, n, l, r, pred);
}
template <class F>
int findLast(int x, int l, int r, int ql, int qr, F pred) {
    if (l > qr || r < ql || !pred(info[x]))
        return -1;
    if (l == r)
        return l;
    int mid = l + r >> 1;
    pushdown(x);
    int res = findLast(x << 1 | 1, mid + 1, r, ql, qr, pred);
    if (res == -1)
        res = findLast(x << 1, l, mid, ql, qr, pred);
    return res;
}
template <class F>
int findLast(int l, int r, F pred) {
    return findLast(1, 1, n, l, r, pred);
}
};
struct Tag {
    ll add = 0;
    void apply(const Tag& v) { add += v.add; }  // 标记怎么合并
};
struct Info {
    ll sum = 0, len = 1;
    void apply(const Tag& v) { sum += len * v.add; }  // 标记怎么更新节点信息
};
Info operator+(const Info& a, const Info& b) {  // 维护的信息怎么合并
    Info c = Info();
    c.sum = a.sum + b.sum;
    c.len = a.len + b.len;
    return c;
}
```

1.20 segtreelazy_xkm

```
template <class info, class tag>
class LSGT {
    std::vector<info> node;
    std::vector<tag> ta;
    int siz;
    void build(int idx, int l, int r) {
        if (l == r)
            return;
        int mid = (l + r) >> 1;
        build(idx << 1, l, mid), build(idx << 1 | 1, mid + 1, r);
        node[idx] = node[idx << 1] + node[idx << 1 | 1];
    }
    template <typename T>
    void build(int idx, int l, int r, const std::vector<T> &vec) {
        if (l == r) {
            node[idx] = vec[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(idx << 1, l, mid, vec), build(idx << 1 | 1, mid + 1, r, vec);
        node[idx] = node[idx << 1] + node[idx << 1 | 1];
    }
    void apply(int idx) {
        if (ta[idx].empty())
            return;
        ta[idx << 1].apply(ta[idx]);
        ta[idx << 1 | 1].apply(ta[idx]);
        node[idx << 1].apply(ta[idx]);
        node[idx << 1 | 1].apply(ta[idx]);
        ta[idx] = {};
    }
    void modify(int idx, int l, int r, int ql, int qr, const tag &add) {
        if (ql <= l && qr >= r) {
            ta[idx].apply(add);
            node[idx].apply(add);
            return;
        }
        apply(idx);
        int mid = (l + r) >> 1;
        if (ql <= mid)
            modify(idx << 1, l, mid, ql, qr, add);
        if (qr > mid)
            modify(idx << 1 | 1, mid + 1, r, ql, qr, add);
        node[idx] = node[idx << 1] + node[idx << 1 | 1];
    }
    info query(int idx, int l, int r, int ql, int qr) {
        if (ql <= l && qr >= r)
            return node[idx];
        apply(idx);
        int mid = (l + r) >> 1;
        if (qr <= mid)
            return query(idx << 1, l, mid, ql, qr);
        else if (ql > mid)
            return query(idx << 1 | 1, mid + 1, r, ql, qr);
        else
            return query(idx << 1, l, mid, ql, qr) + query(idx << 1 | 1, mid +
              1, r, ql, qr);
    }

public:
    LSGT(const int size) : node(size << 2), ta(size << 2), siz(size) {
        build(1, 1, siz);
    }
    template <typename T>
    LSGT(const std::vector<T> &vec) : node(vec.size() << 2), ta(vec.size() << 2)
      , siz(vec.size() - 1) {
        build(1, 1, siz, vec);
    }
    void modify(int ql, int qr, const tag &add) {
        modify(1, 1, siz, ql, qr, add);
    }
    info query(int ql, int qr) {
        return query(1, 1, siz, ql, qr);
    }
};
//区间加/区间和
struct tag {
    long long add;
    tag() : add(0) {}
    tag(long long x) : add(x) {}
    bool empty() const {
        return !add;
    }
    void apply(const tag &o) {
        add += o.add;
    }
};
```

```
struct info {
    int len;
    long long sum;
    info() : len(1), sum(0) {}
    info(long long x) : len(1), sum(x) {}
    info(int len, long long sum) : len(len), sum(sum) {}
    info operator+(const info &o) const {
        return info{len + o.len, sum + o.sum};
    }
    void apply(const tag &o) {
        sum += o.add * len;
    }
};

//区间最小值计数
struct tag {
    int add;
    tag() : add(0) {}
    tag(int x) : add(x) {}
    bool empty() const {
        return !add;
    }
    void apply(const tag &o) {
        add += o.add;
    }
};
struct info {
    int minn;
    long long sum;
    info() : minn(0), sum(0) {}
    info(long long sum) : minn(0), sum(sum) {}
    info(int minn, long long sum) : minn(minn), sum(sum) {}
    info operator+(const info &o) const {
        if (minn == o.minn) {
            return {minn, sum + o.sum};
        } else {
            return minn < o.minn ? *this : o;
        }
    }
    void apply(const tag &o) {
        minn += o.add;
    }
};
```

1.21 st

```
template <typename T, class F = function<T(const T&, const T&)>>
struct SparseTable {
    int n;
    vector<vector<T>> st;
    F func;

    SparseTable(const vector<T>& a, const F& f) : func(f) {
        n = (int)a.size() - 1;
        int max_log = __lg(n) + 1;
        st.resize(max_log + 1);
        st[0] = a;
        for (int j = 1; j <= max_log; j++) {
            st[j].resize(n + 1);
            for (int i = 1; i + (1 << (j - 1)) <= n; i++) {
                st[j][i] = func(st[j - 1][i], st[j - 1][i + (1 << (j - 1))]);
            }
        }
    }
    T get(int l, int r) const {
        int len = __lg(r - l + 1);
        return func(st[len][l], st[len][r - (1 << len) + 1]);
    }
};
void solve() {
    int n, m;
    cin >> n >> m;
    vector<int> a(n + 1, 0);
    for (int i = 1; i <= n; i++) cin >> a[i];
    SparseTable<int> qmx(a, [](int i, int j) { return max(i, j); });
    for (int i = 1; i <= m; i++) {
        int l, r;
        cin >> l >> r;
        cout << qmx.get(l, r) << endl;
    }
}
```

1.22 st_o(n)

```
template <class T,
          class Cmp = std::less<T>>
```

```cpp
struct RMQ {
    const Cmp cmp = Cmp();
    static constexpr unsigned B = 64;
    using u64 = unsigned long long;
    int n;
    std::vector<std::vector<T>> a;
    std::vector<T> pre, suf, ini;
    std::vector<u64> stk;
    RMQ() {}
    RMQ(const std::vector<T> &v) {
        init(v);
    }
    void init(const std::vector<T> &v) {
        n = v.size();
        pre = suf = ini = v;
        stk.resize(n);
        if (!n) {
            return;
        }
        const int M = (n - 1) / B + 1;
        const int lg = std::__lg(M);
        a.assign(lg + 1, std::vector<T>(M));
        for (int i = 0; i < M; i++) {
            a[0][i] = v[i * B];
            for (int j = 1; j < B && i * B + j < n; j++) {
                a[0][i] = std::min(a[0][i], v[i * B + j], cmp);
            }
        }
        for (int i = 1; i < n; i++) {
            if (i % B) {
                pre[i] = std::min(pre[i], pre[i - 1], cmp);
            }
        }
        for (int i = n - 2; i >= 0; i--) {
            if (i % B != B - 1) {
                suf[i] = std::min(suf[i], suf[i + 1], cmp);
            }
        }
        for (int j = 0; j < lg; j++) {
            for (int i = 0; i + (2 << j) <= M; i++) {
                a[j + 1][i] = std::min(a[j][i], a[j][i + (1 << j)], cmp);
            }
        }
        for (int i = 0; i < M; i++) {
            const int l = i * B;
            const int r = std::min(1U * n, l + B);
            u64 s = 0;
            for (int j = l; j < r; j++) {
                while (s && cmp(v[j], v[std::__lg(s) + l])) {
                    s ^= 1ULL << std::__lg(s);
                }
                s |= 1ULL << (j - l);
                stk[j] = s;
            }
        }
    }
    T operator()(int l, int r) {  // 左闭右开
        if (l / B != (r - 1) / B) {
            T ans = std::min(suf[l], pre[r - 1], cmp);
            l = l / B + 1;
            r = r / B;
            if (l < r) {
                int k = std::__lg(r - l);
                ans = std::min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
            }
            return ans;
        } else {
            int x = B * (l / B);
            return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + l];
        }
    }
};
void solve() {
    int n, m;
    cin >> n >> m;
    vector<int> a(n);
    for (int i = 0; i < n; i++) cin >> a[i];
    RMQ<int, greater<int>> qmx(a);
    for (int i = 1; i <= m; i++) {
        int l, r;
        cin >> l >> r;
        l--;
        r--;
        cout << qmx(l, r + 1) << endl;
    }
}
```

1.23 tree_virtual

```
struct edge {
    int v, w;
};
struct HLD {
    int n;
    vector<int> siz, top, parent, l, r, hson, dep;
    vector<vector<edge>> adj;
    int idx;
    vector<int> mn;  // 1—u的最小边权

    HLD() {}
    HLD(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        siz.resize(n + 1), hson.resize(n + 1), top.resize(n + 1);
        parent.resize(n + 1);
        l.resize(n + 1), r.resize(n + 1);
        idx = 0;
        adj.resize(n + 1), dep.resize(n + 1);
        // 根据题目要求加数据结构
        mn.resize(n + 1, 1e9);
    }
    void addEdge(int u, int v, int w) {
        adj[u].push_back({v, w});
    }
    void work(int root = 1) {
        top[root] = root;
        parent[root] = -1;
        dep[root] = 1;
        dfs1(root, -1);
        dfs2(root, root);
    }
    void dfs1(int u, int f) {  // 搞fa,dep,son
        siz[u] = 1;
        for (auto [v, w] : adj[u]) {
            if (v == f)
                continue;
            mn[v] = min(mn[u], w);
            parent[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v, u);
            siz[u] += siz[v];
            if (siz[hson[u]] < siz[v])
                hson[u] = v;
        }
    }
    void dfs2(int u, int t) {  // 搞top
        top[u] = t;            // 记录链头
        l[u] = ++idx;
        if (!hson[u]) {
            r[u] = idx;
            return;
        } // 无重儿子
        dfs2(hson[u], t);  // 搜重儿子
        for (auto [v, w] : adj[u]) {
            if (v == parent[u] || v == hson[u])
                continue;
            dfs2(v, v);  // 搜轻儿子
        }
        r[u] = idx;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = parent[top[u]];
            } else {
                v = parent[top[v]];
            }
        }
        return dep[u] < dep[v] ? u : v;
    }
    bool isAncester(int u, int v) {  // 判断u是不是v的祖先
        return l[u] <= l[v] && r[v] <= r[u];
    }
};
void solve() {
    int n;
    cin >> n;
    HLD hld(n);
    for (int i = 1; i <= n - 1; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        hld.addEdge(u, v, w);
        hld.addEdge(v, u, w);
```

```
    }
    auto cmp = [&](int i, int j) {
        return hld.l[i] < hld.l[j];
    };
    hld.work(1);
    auto buildvt = [&](vector<int>& node, vector<vector<edge>>& e) {
        node.push_back(1);  // 保证根节点在虚树中存在
        sort(alls(node), cmp);
        node.erase(unique(alls(node)), node.end());
        set<int> tmp;
        for (auto x : node) tmp.insert(x);
        for (int i = 1; i < (int)node.size(); i++) tmp.insert(hld.lca(node[i -
          1], node[i]));
        node.clear();
        for (auto x : tmp) node.push_back(x);
        sort(alls(node), cmp);
        vector<int> st;  // 维护一个栈
        for (auto v : node) {
            while (!st.empty() && !hld.isAncester(st.back(), v))
                st.pop_back();
            if (!st.empty())
                e[st.back()].push_back({v, hld.mn[v]});
            st.push_back(v);
        }
    };
    int q;
    cin >> q;
    vector<vector<edge>> e(n + 1);
    vector<ll> dp(n + 1);  // 使得u子树内关键点与u不连通的代价
    vector<bool> vis(n + 1);
    auto cal = [&](auto self, int u, int fa) -> void {  // 计算答案
        for (auto [v, w] : e[u]) {
            if (v == fa)
                continue;
            self(self, v, u);
            if (vis[v])
                dp[u] += w;
            else
                dp[u] += min((ll)w, dp[v]);
        }
    };
    auto clear = [&](vector<int>& node) {  // 清空本次用的点的信息
        for (auto x : node) {
            vis[x] = 0;
            dp[x] = 0;
            e[x].clear();
        }
    };
    for (int i = 1; i <= q; i++) {
        int num;
        cin >> num;
        vector<int> node;
        for (int j = 1; j <= num; j++) {
            int x;
            cin >> x;
            node.push_back(x);
            vis[x] = 1;
        }
        buildvt(node, e);
        cal(cal, 1, 1);
        cout << dp[1] << endl;
        clear(node);
    }
}
```

1.24 trie_per

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

using namespace std;

const int N = 6e5 + 10, M = N * 25 + N;

// 数组大小应为，点数 × 层数（长度），然后第二维是看可能的分支数量。
int idx, tr[M][2], cnt[M], root[N], val[M];
int n, m, a[N];

void insert(int i, int x) {
    int p = root[i], q = root[i - 1];
    for (int k = 24; k >= 0; k--) {
        int t = x >> k & 1;
        if (tr[q][t ^ 1])
            tr[p][t ^ 1] = tr[q][t ^ 1];
        if (tr[p][t]) {
            p = tr[p][t];
            q = tr[q][t];
```

```
        } else {
            p = tr[p][t] = ++idx;
            q = tr[q][t];
        }
        cnt[p] = cnt[q] + 1;
    }
    val[p] = x;
}
int query(int l, int r, int x) {
    int p = root[r], q = root[max(0, l - 1)];
    for (int k = 24; k >= 0; k--) {
        int t = x >> k & 1;
        if (tr[p][t ^ 1] && cnt[tr[p][t ^ 1]] - cnt[tr[q][t ^ 1]] > 0) {
            p = tr[p][t ^ 1], q = tr[q][t ^ 1];
        } else {
            p = tr[p][t], q = tr[q][t];
        }
    }
    return x ^ val[p];
}
int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        root[i] = ++idx;
        a[i] ^= a[i - 1];
        insert(i, a[i]);
    }
    while (m--) {
        char op;
        int l, r, x;
        cin >> op;
        if (op == 'A') {
            cin >> x;
            root[++n] = ++idx;
            a[n] = a[n - 1] ^ x;
            insert(n, a[n]);
        } else {
            cin >> l >> r >> x;
            int y = a[n] ^ x;
            cout << query(l - 1, r - 1, y) << '\n';
        }
    }
    return 0;
}
```

1.25 twoheap

```
priority_queue<int> down;                                  // 大根堆
priority_queue<int, vector<int>, greater<int> > b;  // 前k大的元素在这里，第k大在堆顶
void insert(int x) {
    if (b.empty() || x >= b.top())
        b.push(x);  // 插入
    else {
        down.push(x);
    }
}
int stob(int sum, int small) {  // 第small小转化为第k大，当前总数意义下
    return sum + 1 - small;
}
void makebl(int k) {                                                    // 维护前k大性质
    while ((int)b.size() > k) down.push(b.top()), b.pop();  // 调整
    while ((int)b.size() < k) b.push(down.top()), down.pop();
}
int getkth() {
    return b.top();
}
```

1.26 twoheaplazy

```
#include <bits/stdc++.h>
using namespace std;
class MedianFinder {//可删除对顶堆，动态维护中位数
    priority_queue<int, vector<int>, greater<int>> minheap;
    priority_queue<int> maxheap;
    unordered_map<int, int> delayed;
    int minSize, maxSize;  // decrease delayed

    template <typename T>
    void prune(T &heap) {
        while (!heap.empty()) {
            int num = heap.top();
```

```
            if (delayed.count(num)) {
                delayed[num]--;
                if (delayed[num] == 0)
                    delayed.erase(num);
                heap.pop();
            } else
                break;
        }
    }

    void makebalance() {
        if (maxSize > minSize + 1) {
            minheap.push(maxheap.top());
            maxheap.pop();
            minSize++;
            maxSize--;
            prune(maxheap);
        } else if (maxSize < minSize) {
            maxheap.push(minheap.top());
            minheap.pop();
            maxSize++;
            minSize--;
            prune(minheap);
        }
    }

public:
    MedianFinder() : minSize(0), maxSize(0) {}

    void insert(int num) {
        if (minheap.empty() && maxheap.empty()) {
            maxheap.push(num);
            maxSize++;
        } else {
            int topnum = maxheap.top();
            if (topnum < num) {
                minheap.push(num);
                minSize++;
            } else {
                maxheap.push(num);
                maxSize++;
            }
        }
        makebalance();
    }

    void erase(int num) {
        delayed[num]++;
        if (num <= maxheap.top()) {
            maxSize--;
            if (num == maxheap.top())
                prune(maxheap);
        } else {
            minSize--;
            if (num == minheap.top())
                prune(minheap);
        }
        makebalance();
    }

    double getMedian() {
        if (minSize == maxSize)
            return ((double)minheap.top() + maxheap.top()) / 2;  // 防范int溢出
        else
            return (double)maxheap.top();
    }
};

int main() {
    MedianFinder mf;

    // 插入一些数据
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        mf.insert(a[i]);
    }
    // mf.insert(3);
    // mf.insert(1);
    // mf.insert(5);
    // mf.insert(8);
    // mf.insert(2);
    for (int i = 1; i <= n; i++) {
        mf.erase(a[i]);
        // cout<<mf.getMedian() << endl;
        baoliu(mf.getMedian(), 1);
        cout << endl;
        mf.insert(a[i]);
```

```
    }
    // // 输出当前中位数
    // cout << "Current median: " << mf.getMedian() << endl;
    //
    // // 删除一个元素
    // mf.erase(1);
    //
    // // 再次输出中位数
    // cout << "Updated median: " << mf.getMedian() << endl;

    return 0;
}
```

1.27 笛卡尔树

```
struct DKR {
    int n;
    vector<int> l, r;
    int root;
    stack<int> st;

    DKR(int nn) : n(nn), l(nn + 1), r(nn + 1), root(0) {}
    // 默认为小根堆，维护最小值所在区间
    // dkr.built(a,less<int>());大根堆
    int built(const vector<int>& a, function<bool(int, int)> cmp = greater<int
      >()) {
        while (!st.empty()) st.pop();  // 清空栈
        for (int i = 1; i <= n; i++) {
            int last = 0;

            while (!st.empty() && cmp(a[st.top()], a[i])) {
                last = st.top();
                st.pop();
            }
            if (!st.empty()) {
                r[st.top()] = i;
            } else {
                root = i;
            }
            l[i] = last;
            st.push(i);
        }
        return root;
    }
};
```

2 Graph

2.1 2-sat

```
struct TwoSat {
    int n;
    vector<vector<int>> e;
    vector<bool> ans;
    TwoSat(int n) : n(n), e(2 * n), ans(n) {}
    void add(int u, bool f, int v, bool g) {
        // 偶数是u取反的变量，奇数是正变量
        // 一般来说需要反变量向正变量连边。
        e[2 * u + !f].push_back(2 * v + g);
        if (u == v)
            return;  // 对于单变量指定特判
        e[2 * v + !g].push_back(2 * u + f);
    }
    bool judge() {
        vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
        vector<int> stk;
        int now = 0, cnt = 0;
        function<void(int)> tarjan = [&](int u) {
            stk.push_back(u);
            dfn[u] = low[u] = now++;
            for (auto v : e[u]) {
                if (dfn[v] == -1) {
                    tarjan(v);
                    low[u] = min(low[u], low[v]);
                } else if (id[v] == -1) {
                    low[u] = min(low[u], dfn[v]);
                }
            }
            if (dfn[u] == low[u]) {
                int v;
                do {
                    v = stk.back();
                    stk.pop_back();
                    id[v] = cnt;
```

```
                } while (v != u);
                ++cnt;
            }
        };
        for (int i = 0; i < 2 * n; ++i)
            if (dfn[i] == -1)
                tarjan(i);
        for (int i = 0; i < n; ++i) {
            if (id[2 * i] == id[2 * i + 1])
                return false;
            ans[i] = id[2 * i] > id[2 * i + 1];
        }
        return true;
    }
    vector<bool> answer() { return ans; }
};
void solve() {
    int n, m;
    cin >> n >> m;
    TwoSat sat2(n);
    for (int i = 1; i <= m; i++) {
        int u, v;
        bool f1, f2;
        //u为f1或v为f2
        cin >> u >> f1 >> v >> f2;
        u--;
        v--;
        sat2.add(u, f1, v, f2);
    }
    if (sat2.judge()) {
        cout << "POSSIBLE" << endl;
        for (int i = 0; i < n; i++) cout << sat2.ans[i] << " ";
    } else
        cout << "IMPOSSIBLE" << endl;
}
```

2.2 dfs 判环的具体路径

```
void get_cycle(const vector<vector<int>>& e, vector<vector<int>>& cycle) {
    int n = e.size() - 1;//适用于每个点只在一个环上的情况，不考虑环套环
    vector<int> vis(n + 1), pre(n + 1);
    auto dfs = [&](auto&& self, int u, int fa) -> void {
        vis[u] = 1;  // 标记当前节点正在访问
        for (auto v : e[u]) {
            if (v == fa)  // 无向图中跳过父节点
                continue;
            deb(u, v);
            if (vis[v] == 0) {          // 未访问过的节点
                pre[v] = u;             // 记录前驱节点
                self(self, v, u);       // 递归访问
            } else if (vis[v] == 1) {  // 找到一个环
                vector<int> temp;
                int tmp = u;
                while (tmp != v) {  // 回溯找到环中的节点
                    temp.push_back(tmp);
                    tmp = pre[tmp];
                }
                temp.push_back(v);
                cycle.push_back(temp);  // 将环存入cycle
            }
        }
        vis[u] = 2;  // 标记为已访问
    };

    for (int i = 1; i <= n; i++) {
        if (vis[i] == 0) {  // 对每个未访问节点进行 DFS
            dfs(dfs, i, 0);
        }
    }
}
void solve() {
    int n;
    cin >> n;
    vector<vector<int>> e(n + 1), cycle(n + 1);
    for (int i = 1; i <= n; i++) {
        int u, v;
        cin >> u >> v;
        e[u].push_back(v);
        e[v].push_back(u);
    }
    get_cycle(e, cycle);
    for (int i = 0; i < (int)cycle.size(); i++) {
        if (cycle[i].size()) {
            sort(alls(cycle[i]));//不排序才是环上正确的路径
            for (auto x : cycle[i]) cout << x << " ";
```

```
            cout << endl;
        }
    }
}
```

2.3 匈牙利

```
struct XYL {
    vector<int> vis, match;
    vector<vector<int>> e;
    int n1, n2, m;
    XYL(int n1_, int n2_, int m_) {
        n1 = n1_;
        n2 = n2_;
        m = m_;
        vis.resize(n2 + 1);
        match.resize(n2 + 1);
        e.resize(n1 + 1);
    }
    void addEdge(int u, int v) {
        e[u].emplace_back(v);
    }
    int dfs(int u) {
        for (auto v : e[u]) {
            // 妹子的编号v
            if (vis[v])
                continue;
            vis[v] = 1;  // 先标记这个妹子
            if (!match[v] || dfs(match[v])) {
                match[v] = u;  // 配成对
                return 1;
            }
        }
        return 0;
    }
    int work() {
        int ans = 0;
        for (int i = 1; i <= n1; i++) {
            fill(alls(vis), 0);  // 每轮找增广路以前清空vis
            ans += dfs(i);
        }
        return ans;
    }
    vector<int> fangan() {
        vector<int> res(max(n1, n2) + 1);
        for (int i = 1; i <= n2; i++) res[match[i]] = i;
        return res;  // res[i]表示匹配的女生编号
    };
};
```

2.4 无向图判环

```
bool iscycle_undirect(vector<vector<int>> &e, vector<int> &deg) {  // 有环返回
  true
    queue<int> q;
    int n = deg.size() - 1;
    vector<bool> vis(n + 1);
    for (int i = 1; i <= n; i++)
        if (deg[i] == 1) {
            vis[i] = true;
            q.push(i);
        }
    while (q.size()) {
        auto u = q.front();
        q.pop();
        for (auto v : e[u]) {
            deg[v]--;
            if (deg[v] <= 1 && vis[v] == 0) {
                q.push(v);
                vis[v] = true;
            }
        }
    }
    for (int i = 1; i <= n; i++)
        if (deg[i] > 1) {  // 最后度数大于1的点在环上
            return true;
        }
    return false;
}//还需要对不同连通块分别处理具体路径和环长度
```

2.5 最短路 __ 迪杰斯特拉

```
struct edge {
    int v, w;
};
vector<int> dijs(vector<vector<edge>>& e, int s) {
    priority_queue<pii, vector<pii>, greater<pii>> q;
    vector<int> d(n + 1, (1LL<<31)-1);
    vector<bool> vis(n + 1);
    d[s] = 0;
    q.push({d[s],s});
    while (q.size()) {
        auto t = q.top();
        q.pop();
        int u = t.sec;
        if (vis[u])
            continue;  // 再进队就直接跳过
        vis[u] = 1;    // 标记u已出队

        for (auto [v, w] : e[u]) {

            if (d[v] > d[u] + w) {
                d[v] = d[u] + w;
                q.push({d[v], v});  // 小根堆
            }
        }

    }
    return d;
}
```

2.6 有向图判环

```
bool iscycle_direct(vector<vector<int>> &e, vector<int> &deg) {//有环返回true
    queue<int> q;
    int n = deg.size() - 1;
    for (int i = 1; i <= n; i++)
        if (deg[i] == 0)
            q.push(i);
    while (q.size()) {
        auto u = q.front();
        q.pop();
        for (auto v : e[u]) {
            deg[v]--;
            if (deg[v] == 0)
                q.push(v);
        }
    }
    for (int i = 1; i <= n; i++)
        if (deg[i]) {
            return true;
        }
    return false;
}
```

3 Math

3.1 Frac

```
template <class T>
struct Frac {
    T zi;
    T mu;
    Frac(T num_, T den_) : zi(num_), mu(den_) {
        if (mu < 0) {
            mu = -mu;
            zi = -zi;
        }
    }
    Frac() : Frac(0, 1) {}  // (1/0)
    Frac(T num_) : Frac(num_, 1) {}
    void reduce() {
        T g = std::gcd(zi, mu);  // 调用 std::gcd 计算最大公约数
        zi /= g;                 // 将分子除以最大公约数
        mu /= g;                 // 将分母除以最大公约数
    }
    explicit operator double() const {
        return 1. * zi / mu;
    }
    Frac &operator+=(const Frac &rhs) {
        zi = zi * rhs.mu + rhs.zi * mu;
        mu *= rhs.mu;
        return *this;
    }
    Frac &operator-=(const Frac &rhs) {
        zi = zi * rhs.mu - rhs.zi * mu;
```

```
        mu *= rhs.mu;
        return *this;
    }
    Frac &operator*=(const Frac &rhs) {
        zi *= rhs.zi;
        mu *= rhs.mu;
        return *this;
    }
    Frac &operator/=(const Frac &rhs) {
        zi *= rhs.mu;
        mu *= rhs.zi;
        if (mu < 0) {
            zi = -zi;
            mu = -mu;
        }
        return *this;
    }
    friend Frac operator+(Frac lhs, const Frac &rhs) {
        return lhs += rhs;
    }
    friend Frac operator-(Frac lhs, const Frac &rhs) {
        return lhs -= rhs;
    }
    friend Frac operator*(Frac lhs, const Frac &rhs) {
        return lhs *= rhs;
    }
    friend Frac operator/(Frac lhs, const Frac &rhs) {
        return lhs /= rhs;
    }
    friend Frac operator-(const Frac &a) {
        return Frac(-a.zi, a.mu);
    }
    friend bool operator==(const Frac &lhs, const Frac &rhs) {
        return lhs.zi * rhs.mu == rhs.zi * lhs.mu;
    }
    friend bool operator!=(const Frac &lhs, const Frac &rhs) {
        return lhs.zi * rhs.mu != rhs.zi * lhs.mu;
    }
    friend bool operator<(const Frac &lhs, const Frac &rhs) {
        return lhs.zi * rhs.mu < rhs.zi * lhs.mu;
    }
    friend bool operator>(const Frac &lhs, const Frac &rhs) {
        return lhs.zi * rhs.mu > rhs.zi * lhs.mu;
    }
    friend bool operator<=(const Frac &lhs, const Frac &rhs) {
        return lhs.zi * rhs.mu <= rhs.zi * lhs.mu;
    }
    friend bool operator>=(const Frac &lhs, const Frac &rhs) {
        return lhs.zi * rhs.mu >= rhs.zi * lhs.mu;
    }
    friend std::ostream &operator<<(std::ostream &os, Frac x) {
        T g = std::gcd(x.zi, x.mu);
        if (x.mu == g) {
            return os << x.zi / g;
        } else {
            return os << x.zi / g << "/" << x.mu / g;
        }
    }
};
```

3.2 Gauss

```
int gauss(vector<vector<db>>& a, int n) {
    int r = 1;                          // 当前行
    for (int c = 1; c <= n; c++) {  // 消元进行到第c列
        // 1.找到c列的最大行t
        int t = r;
        for (int i = r; i <= n; i++)
            if (fabs(a[i][c]) > fabs(a[t][c]))
                t = i;
        if (fabs(a[t][c]) < eps)
            continue;  // c列已全为0

        // 2.把最大行换到上面
        for (int i = c; i <= n + 1; i++) swap(a[t][i], a[r][i]);

        // 3.把当前行r的第一个数，变成1
        for (int i = n + 1; i >= c; i--) a[r][i] /= a[r][c];

        // 4.把当前列c下面的所有数，全部消成0
        for (int i = r + 1; i <= n; i++)
            if (fabs(a[i][c]) > eps)
                for (int j = n + 1; j >= c; j--)
                    a[i][j] -= a[i][c] * a[r][j];
        r++;  // 从下一行开始消元下一列
    }
```

```
    if (r <= n) {  // 说明已经提前变成梯形矩阵
        for (int i = r; i <= n; i++) {
            if (fabs(a[i][n + 1]) > eps)
                return 0;
        }  // 左边=0，右边≠0,无解
        return 2;  // 0==0，无穷多解
    }
    // 5.唯一解，从下往上回代，得到方程的解
    for (int i = n; i >= 1; i--)
        for (int j = i + 1; j <= n; j++)
            a[i][n + 1] -= a[i][j] * a[j][n + 1];
    return 1;
}

void solve() {
    int n;
    cin >> n;
    vector b(n + 1, vector<db>(n + 2));
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n + 1; j++)
            cin >> b[i][j];
    int t = gauss(b, n);
    if (t == 0) {
        cout << "No solution" << endl;
    } else if (t == 2) {
        cout << "Infinite group solutions" << endl;
    } else {
        for (int i = 1; i <= n; i++) {
            baoliu(b[i][n + 1], 2);
            cout << endl;
        }
    }
}
```

3.3 Gauss_matrix

```
db b[N][N];  // 增广矩阵
int id[N];   // id[j]=i：表示第j列的答案最终在第i行被计算
int rid[N];  // 第i行可以算出第j列的主元
void print(int n, int m) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m + 1; j++) {
            cerr << setiosflags(ios::left) << setw(5) << b[i][j];
        }
        cerr << endl;
    }
    cerr << "----------" << endl;
}
int rksz;
int freenum;
vector<db> ans;
int gauss(db a[][N], int n, int m) {  // n个方程，m个未知数。默认多余自由变量为0，记录映射关系
    int r = 1;                        // 当前行
    for (int c = 1; c <= m; c++) {    // 消元进行到第c列
        // 1.找到c列的最大行t
        int t = r;
        for (int i = r; i <= n; i++)
            if (fabs(a[i][c]) > fabs(a[t][c]))
                t = i;
        if (t > n || fabs(a[t][c]) < eps)
            continue;  // c列已全为0
        assert(t <= n);
        // 2.把最大行换到上面
        for (int i = c; i <= m + 1; i++) swap(a[t][i], a[r][i]);
        // 3.把当前行r的第一个数，变成1
        for (int i = m + 1; i >= c; i--) a[r][i] /= a[r][c];
        // 4.把当前列c下面的所有数，全部消成0
        for (int i = r + 1; i <= n; i++)
            if (fabs(a[i][c]) > eps)
                for (int j = m + 1; j >= c; j--)
                    a[i][j] -= a[i][c] * a[r][j];
        id[c] = r;
        rid[r] = c;
        r++;  // 从下一行开始消元下一列
    }
    //----------
    // print(n, m);
    rksz = r - 1;
    freenum = m - r + 1;
    ans.resize(m + 1);
    //----------
    if (r <= m) {  // 说明已经提前变成梯形矩阵
        for (int i = r; i <= n; i++) {
            if (fabs(a[i][m + 1]) > eps)
                return 0;  // 左边=0，右边≠0,无解
```

```
        }
    }
    for (int i = 1; i <= m; i++) {
        if (id[i] == 0) {
            // deb(i);
            ans[i] = 1;  // 如果第i列的主元没有对应行，自由变量随机赋值
        }
    }
    // 5.唯一解，从下往上回代，得到方程的解
    for (int i = rksz; i >= 1; i--) {
        for (int j = 1; j <= m; j++) {  // 左侧自由变量残余，右侧已经算出来的以及右侧自由变量
            if (j == rid[i])
                continue;
            a[i][m + 1] -= a[i][j] * ans[j];
        }
        // deb(rid[i]);
        ans[rid[i]] = a[i][m + 1];  // 第i行的主元在rid[i]列
    }
    if (m > n) {
        return 2;
    } else {
        // m<=n;
        assert(rksz <= m);
        if (rksz == m)
            return 1;
        return 2;
    }
}
void solve() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> b[i][j];
        }
    }
    for (int i = 1; i <= n; i++) cin >> b[i][m + 1];
    int t = gauss(b, n, m);  // n个方程,m个未知数
    deb(t, rksz, freenum);
    if (t == 0) {
        cout << "NO" << endl;
    } else {
        cout << "YES" << endl;
        for (int i = 1; i <= m; i++) cout << fixed << setprecision(12) << ans[i]
            << endl;
    }
}
```

3.4 Gauss_mod

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 205;
int n, m;
int s[N], t[N], G[N][N];
int a[N][N];
void swap_line(int x, int y) {
    for (int i = 1; i <= n + 1; i++) swap(a[x][i], a[y][i]);
}
bool empty_line(int x, int y) {
    for (int i = x; i <= n; i++)
        if (a[i][y])
            return 0;
    return 1;
}
void exgcd(int n, int m, int &x, int &y) {
    if (m == 0) {
        x = 1, y = 0;
        return;
    }
    int x1, y1;
    exgcd(m, n % m, x1, y1);
    x = y1, y = x1 - n / m * y1;
}
int get_inv(int p) {
    int x, y;
    exgcd(p, m, x, y);
    return (x % m + m) % m;
}
void gauss() {
    int hang = 1, pos;
    for (int i = 1; i <= n; i++) {
        pos = hang;
        if (empty_line(pos, i))
```

```
            continue;
        for (int j = hang; j <= n; j++) {
            if (a[j][i] > a[pos][i])
                pos = j;
        }
        swap_line(hang, pos);
        int op = get_inv(a[hang][i]);
        for (int j = 1; j <= n; j++) {
            if (hang == j)
                continue;
            for (int k = n + 1; k >= i; k--) a[j][k] = (a[j][k] - (a[hang][k] *
              a[j][i] % m + m) % m * op % m + m) % m;
        }
        hang++;
    }
    if (hang <= n) {
        for (int i = hang; i <= n; i++) {
            bool flag = 0;
            for (int j = 1; j <= n; j++)
                if (a[i][j])
                    flag = 1;
            if (!flag && a[i][n + 1]) {
                cout << "niuza";
                return;
            }
        }
    }
    hang = 1;
    for (int i = 1; i <= n; i++) {
        cout << a[hang][n + 1] * get_inv(a[hang][i]) % m << "␣";
        a[hang][n + 1] = 0;
        if (!a[hang][i + 1])
            hang++;
    }
}
signed main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        int k;
        cin >> k;
        for (int j = 1; j <= k; j++) {
            int u;
            cin >> u;
            G[i][u] = 1;
        }
    }
    for (int i = 1; i <= n; i++) cin >> s[i];
    for (int i = 1; i <= n; i++) cin >> t[i];
    for (int u = 1; u <= n; u++) {
        a[u][u] = 1;  //!@!!!
        for (int v = 1; v <= n; v++) {
            if (G[v][u])
                a[u][v] = 1;
        }
        a[u][n + 1] = ((t[u] - s[u]) % m + m) % m;
    }
    gauss();
    return 0;
}
```

3.5 Gauss_xor

异或线性方程组就是常数项和各变量系数都是0/1，
并且各变量取值也是0/1

```
gauss(vector<vector<int>>& a, int n, vector<int>& solution) {
    vector<int> freevar;           // 记录自由变量对应的列
    vector<int> pivot(n + 1, -1);  // 记录每一行的主元所在的列
    //solution.assign(n + 1, 0);
    int r = 1;
    for (int c = 1; c <= n; c++) {
        int t = r;
        // 找到当前列中的主元
        for (int i = r; i <= n; i++) {
            if (a[i][c]) {
                t = i;
                break;
            }
        }
        if (!a[t][c]) {
            freevar.push_back(c);
            continue;  // 当前列没有主元，继续到下一列
        }
        pivot[r] = c;  // 第 r 行的主元在 c 列
        if (t != r) {  // 交换行，将主元行放在第 r 行
            for (int i = c; i <= n + 1; i++)
                swap(a[r][i], a[t][i]);
```

```
        }
        // 消去主元下方的所有行
        for (int i = r + 1; i <= n; i++) {
            if (a[i][c])
                for (int j = n + 1; j >= c; j--) a[i][j] ^= a[r][j];
        }
        r++;
    }

    // 检查是否有解
    for (int i = r; i <= n; i++) {
        if (a[i][n + 1])
            return 0;  // 无解
    }
    //int tot = 0;
    int rksz = r - 1;  // 这是系数矩阵的秩
    // 自由变量根据题目要求情况去赋值
    for (auto i : freevar) solution[i] =0;
    //------------------------------
    for (int i = rksz; i >= 1; i--) {
        int sum = a[i][n + 1];
        for (int j = 1; j <= n; j++) {
            if (j == pivot[i]) {
                continue;
            }  // 如果不是主元所在的列
            sum ^= (a[i][j] * solution[j]);  // 右边已经求出来的，左边自由变量遗
              留
        }
        solution[pivot[i]] = sum;  // 求解对应的主元变量
    }
    assert(rksz <= n);
    if (rksz < n)
        return 2;  // 无穷多解
    return 1;      // 唯一解
}
// int t = gauss(b, n, sol);
```

3.6 Gauss_xor_bitset

```
int rksz;
// 还需要处理m方程，n变量，m>n
#define bit(x) bitset<(x)>
int gauss(vector<bit(5002)>& a, int n, int m, vector<int>& solution) {
    vector<int> freevar;           // 记录自由变量对应的列
    vector<int> pivot(n + 1, -1);  // 记录每一行的主元所在的列
    int r = 1;
    for (int c = 1; c <= m; c++) {
        int t = r;
        // 找到当前列中的主元
        for (int i = r; i <= n; i++) {
            if (a[i][c]) {
                t = i;
                break;
            }
        }
        if (t > n || !a[t][c]) {
            freevar.push_back(c);
            continue;  // 当前列没有主元，继续到下一列
        }
        pivot[r] = c;  // 第 r 行的主元在 c 列
        if (t != r) {  // 交换行，将主元行放在第 r 行
            swap(a[r], a[t]);
        }
        // 消去主元下方的所有行
        for (int i = r + 1; i <= n; i++) {
            if (a[i][c])
                a[i] ^= a[r];
        }
        r++;
    }

    // 检查是否有解
    for (int i = r; i <= n; i++) {
        if (a[i][m + 1])
            return 0;  // 无解
    }
    // int tot = 0;
    rksz = r - 1;  // 这是系数矩阵的秩
    // 自由变量根据题目要求情况去赋值
    for (auto i : freevar) solution[i] = 0;
    //------------------------------
    for (int i = rksz; i >= 1; i--) {
        int sum = a[i][m + 1];
        for (int j = 1; j <= m; j++) {
            if (j == pivot[i]) {
                continue;
```

```
            } // 如果不是主元所在的列
            sum ^= (a[i][j] * solution[j]); // 右边已经求出来的，左边自由变量遗留
        }
        solution[pivot[i]] = sum; // 求解对应的主元变量
    }
    // assert(rksz <= m);
    if (rksz < m)
        return 2; // 无穷多解
    return 1;      // 唯一解
}
// int t = gauss(b, n, m, sol);
```

3.7 Gauss_xor_matrix

```
int rksz;
// 还没有处理n方程，m变量的情况 (n>m)
int gauss(vector<vector<int>>& a, int n, int m, vector<int>& solution) {
    vector<int> freevar;           // 记录自由变量对应的列
    vector<int> pivot(n + 1, -1); // 记录每一行的主元所在的列
    int r = 1;
    for (int c = 1; c <= m; c++) {
        int t = r;
        // 找到当前列中的主元
        for (int i = r; i <= n; i++) {
            if (a[i][c]) {
                t = i;
                break;
            }
        }
        // if (t > n)
        //     deb(t, n, c, m);
        // assert(t <= n);
        // assert(c <= m);
        if (t > n || !a[t][c]) {
            freevar.push_back(c);
            continue; // 当前列没有主元，继续到下一列
        }
        pivot[r] = c; // 第 r 行的主元在 c 列
        if (t != r) { // 交换行，将主元行放在第 r 行
            for (int i = c; i <= m + 1; i++)
                swap(a[r][i], a[t][i]);
        }
        // 消去主元下方的所有行
        for (int i = r + 1; i <= n; i++) {
            if (a[i][c])
                for (int j = m + 1; j >= c; j--) a[i][j] ^= a[r][j];
        }
        r++;
    }

    // 检查是否有解
    for (int i = r; i <= n; i++) {
        if (a[i][m + 1])
            return 0; // 无解
    }
    // int tot = 0;
    rksz = r - 1; // 这是系数矩阵的秩
    // 自由变量根据题目要求情况去赋值
    for (auto i : freevar) solution[i] = 0;
    //-----------------------------
    for (int i = rksz; i >= 1; i--) {
        int sum = a[i][m + 1];
        for (int j = 1; j <= m; j++) {
            if (j == pivot[i]) {
                continue;
            } // 如果不是主元所在的列
            sum ^= (a[i][j] * solution[j]); // 右边已经求出来的，左边自由变量遗留
        }
        solution[pivot[i]] = sum; // 求解对应的主元变量
    }
    // assert(rksz <= m);
    if (rksz < m)
        return 2; // 无穷多解
    return 1;      // 唯一解
}
// int t = gauss(b, n, m, sol);
```

3.8 Miller_Robin

```
using i64 = long long;
i64 mul(i64 a, i64 b, i64 m) {
    return static_cast<__int128>(a) * b % m;
```

```
}
i64 power(i64 a, i64 b, i64 m) {
    i64 res = 1 % m;
    for (; b; b >>= 1, a = mul(a, a, m))
        if (b & 1)
            res = mul(res, a, m);
    return res;
}
bool isprime(i64 n) {
    if (n < 2)
        return false;
    static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    int s = __builtin_ctzll(n - 1);
    i64 d = (n - 1) >> s;
    for (auto a : A) {
        if (a == n)
            return true;
        i64 x = power(a, d, n);
        if (x == 1 || x == n - 1)
            continue;
        bool ok = false;
        for (int i = 0; i < s - 1; ++i) {
            x = mul(x, x, n);
            if (x == n - 1) {
                ok = true;
                break;
            }
        }
        if (!ok)
            return false;
    }
    return true;
}
```

3.9 bigint

```
using uint = unsigned;
const int MOD = 998244353; // NTT模数

// 模加法
int Add(int x, int y) { return (x + y >= MOD) ? x + y - MOD : x + y; }
// 模减法
int Dec(int x, int y) { return (x - y < 0) ? x - y + MOD : x - y; }
// 模乘法
int mul(int x, int y) { return 1ll * x * y % MOD; }
// 快速幂计算
uint qp(uint a, int b) {
    uint res = 1;
    for (; b; b >>= 1, a = mul(a, a))
        if (b & 1)
            res = mul(res, a);
    return res;
}

namespace NTT {
    int sz; // FFT大小
    uint w[2500005], w_mf[2500005]; // 存储预计算的单位根及其乘法因子
    // 计算乘法因子
    int mf(int x) { return (1ll * x << 32) / MOD; }
    // 初始化NTT
    void init(int n) {
        for (sz = 2; sz < n; sz <<= 1);
        uint pr = qp(3, (MOD - 1) / sz);
        w[sz / 2] = 1;
        w_mf[sz / 2] = mf(1);
        for (int i = 1; i < sz / 2; i++) {
            w[sz / 2 + i] = mul(w[sz / 2 + i - 1], pr);
            w_mf[sz / 2 + i] = mf(w[sz / 2 + i]);
        }
        for (int i = sz / 2 - 1; i; i--) {
            w[i] = w[i << 1];
            w_mf[i] = w_mf[i << 1];
        }
    }
    // 前向NTT
    void ntt(vector<uint>& A, int L) {
        for (int d = L >> 1; d; d >>= 1) {
            for (int i = 0; i < L; i += (d << 1)) {
                for (int j = 0; j < d; j++) {
                    uint x = A[i + j] + A[i + d + j];
                    if (x >= 2 * MOD) x -= 2 * MOD;
                    ll t = A[i + j] + 2 * MOD - A[i + d + j];
                    ll q = t * w_mf[d + j] >> 32;
                    int y = t * w[d + j] - q * MOD;
                    A[i + j] = x;
                    A[i + d + j] = y;
```

```
                }
            }
        }
        for (int i = 0; i < L; i++) {
            if (A[i] >= MOD) A[i] -= MOD;
        }
    }
    // 逆NTT
    void intt(vector<uint>& A, int L) {
        for (int d = 1; d < L; d <<= 1) {
            for (int i = 0; i < L; i += (d << 1)) {
                for (int j = 0; j < d; j++) {
                    uint x = A[i + j];
                    if (x >= 2 * MOD) x -= 2 * MOD;
                    ll t = A[i + d + j];
                    ll q = t * w_mf[d + j] >> 32;
                    int y = t * w[d + j] - q * MOD;
                    A[i + j] = x + y;
                    A[i + d + j] = x + 2 * MOD - y;
                }
            }
        }
        int k = (L & (-L));
        reverse(A.begin() + 1, A.end());
        for (int i = 0; i < L; i++) {
            ll m = -A[i] & (L - 1);
            A[i] = (A[i] + m * MOD) / k;
            if (A[i] >= MOD) A[i] -= MOD;
        }
    }
}

struct bigint {
    vector<int> nums; // 存储大整数的每一位
    int operator[](const int& k)const { return nums[k]; }
    int& operator[](const int& k) { return nums[k]; }
    int size() { return nums.size(); }
    void push_back(int x) { nums.push_back(x); }
    // 从整数构造大整数
    bigint(int x = 0) {
        do {
            nums.push_back(x % 10);
            x /= 10;
        } while (x);
    }
    // 从字符串构造大整数
    bigint(string s) {
        for (int i = s.size() - 1; i >= 0; i--)
            nums.push_back(s[i] - '0');
        trim();
    }
    // 去掉多余的前导零
    void trim() {
        while (nums.size() > 1 && nums.back() == 0) {
            nums.pop_back();
        }
    }
    // 清空大整数
    void clear() {
        nums.clear();
    }
    // 输入大整数
    friend istream& operator>>(istream& cin, bigint& num) {
        string tnum;
        cin >> tnum;
        num = tnum;
        return cin;
    }
    // 输出大整数
    friend ostream& operator<<(ostream& cout, bigint num) {
        bool start = false;
        for (int i = num.size() - 1; i >= 0; i--) {
            if (!start && num[i] == 0)
                continue;
            start = true;
            cout << num[i];
        }
        if (!start)
            cout << 0;
        return cout;
    }
};

// 比较运算符重载
bool operator<(bigint a, bigint b) {
    if (a.size() != b.size())
        return a.size() < b.size();
    for (int i = a.size() - 1; i >= 0; i--)
```

```
        if (a[i] != b[i])
            return a[i] < b[i];
    return false;
}

bool operator>(bigint a, bigint b) {
    return b < a;
}

bool operator<=(bigint a, bigint b) {
    return !(a > b);
}

bool operator>=(bigint a, bigint b) {
    return !(a < b);
}

bool operator==(bigint a, bigint b) {
    return !(a < b) && !(a > b);
}

bool operator!=(bigint a, bigint b) {
    return a < b || a > b;
}

// 大整数加法
bigint operator+(bigint a, bigint b) {
    bigint res;
    res.clear();
    int t = 0;
    int mx = max(a.size(), b.size());
    for (int i = 0; i < mx || t; i++) {
        if (i < a.size()) {
            t += a[i];
        }
        if (i < b.size()) {
            t += b[i];
        }
        res.push_back(t % 10);
        t /= 10;
    }
    res.trim();
    return res;
}
```

```
// 大整数减法
bigint operator-(bigint a, bigint b) {
    bigint res(a);
    bigint sub(b);
    int flag = 0;
    int len = res.size();
    while (sub.size() < res.size())
        sub.push_back(0);
    for (int i = 0; i < len; i++) {
        if (res[i] + flag >= sub[i]) {
            res[i] = res[i] + flag - sub[i];
            flag = 0;
        }
        else {
            res[i] = res[i] + 10 + flag - sub[i];
            flag = -1;
        }
    }
    res.trim();
    return res;
}

// 大整数乘法（nlogn）
bigint operator*(bigint a, bigint b) {
    bigint res;
    res.nums.pop_back();
    int dega = a.size() - 1, degb = b.size() - 1;
    int n = dega + degb + 1;
    int lim;
    for (lim = 1; lim < n; lim <<= 1);
    NTT::init(lim);
    vector<uint> A(lim);
    for (int i = 0; i <= dega; i++) A[i] = a[i];
    vector<uint> B(lim);
    for (int i = 0; i <= degb; i++) B[i] = b[i];
    NTT::ntt(A, lim);
    NTT::ntt(B, lim);
    for (int i = 0; i < lim; i++) A[i] = mul(A[i], B[i]);
    NTT::intt(A, lim);
    for (int i = 0, t = 0; i < lim || t; i++) {
        if (i < lim) t += A[i];
        res.push_back(t % 10);
        t /= 10;
```

```
    }
    res.trim();
    return res;
}

// 大整数与长整数乘法
bigint operator*(bigint a, ll b) {
    bigint res(a);
    int carry = 0;
    for (int i = 0; i < a.size(); i++) {
        carry += a[i] * b;
        res[i] = carry % 10;
        carry /= 10;
    }
    while (carry > 0) {
        res.push_back(carry % 10);
        carry /= 10;
    }
    return res;
}

// 大整数除法
bigint operator/(bigint a, bigint b) {
    bigint tnum(a);
    if (a < b)
        return 0;
    int n = a.size() - b.size();
    b.nums.insert(b.nums.begin(), n, 0);
    if (tnum >= b) {
        n++;
        b.nums.insert(b.nums.begin(), 0);
    }
    bigint ans;
    ans.nums.assign(n, 0);
    int n2 = b.size();
    while (n--) {
        n2--;
        b.nums.erase(b.nums.begin());
        while (!(tnum < b)) {
            int n1 = tnum.size();
            for (int j = 0; j < n2; j++) {
                tnum[j] -= b[j];
                if (tnum[j] < 0) {
                    tnum[j + 1]--;
                    tnum[j] += 10;
                }
            }
            tnum.trim();
            ans[n]++;
        }
    }
    ans.trim();
    return ans;
}

// 大整数与长整数除法
bigint operator/(bigint a, ll b) {
    bigint ans;
    ans.clear();
    int r = 0;
    for (int i = a.size() - 1; i >= 0; i--) {
        r = r % b * 10 + a[i];
        ans.push_back(r / b);
    }
    reverse(ans.nums.begin(), ans.nums.end());
    ans.trim();
    return ans;
}

// 大整数取模
bigint operator%(bigint a, bigint b) {
    bigint div_res = a / b;
    return a - div_res * b;
}

// 大整数与长整数取模
bigint operator%(bigint a, ll b) {
    bigint div_res = a / b;
    return a - div_res * b;
}

// 大整数快速幂
bigint qp(bigint a, ll n) {
    bigint res(1);
    while (n) {
        if (n & 1) res = res * a;
        a = a * a;
        n >>= 1;
```

```
    }
    return res;
}

// 大整数组合数
bigint comb(bigint n, bigint m) {
    bigint res = 1;
    for (bigint up = n, down = 1; down <= m; up = up - 1, down = down + 1)
        res = res * up, res = res / down;
    return res;
}
```

3.10 comb

```
#define int long long
int fac[N], infac[N], inv[N];
int qmi(int a, int b) {
    int res = 1;
    while (b) {
        if (b & 1)
            res = (res * a) % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}
int askinv(int x) {
    return qmi(x, mod - 2);
}
void init(int n) {
    fac[0] = 1, infac[0] = 1;
    for (int i = 1; i <= n; i++) fac[i] = fac[i - 1] * i % mod;
    infac[n] = askinv(fac[n]);
    for (int i = n; i >= 1; i--) {
        infac[i - 1] = infac[i] * i % mod;
        inv[i] = fac[i - 1] * infac[i] % mod;
    }
}
int C(int n, int m) {
    if (n == 0 || m == 0)
        return 1;
    return fac[n] * infac[m] % mod * infac[n - m] % mod;
}

int A(int n, int m) {
    if (n == 0 || m == 0)
        return 1;
    return fac[n] * infac[n - m] % mod;
}
```

3.11 comb_Z

```
struct Comb {
    int n;
    std::vector<Z> _fac;
    std::vector<Z> _invfac;
    std::vector<Z> _inv;

    Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
    Comb(int n) : Comb() {
        init(n);
    }

    void init(int m) {
        m = std::min(m, Z::getMod() - 1);
        if (m <= n) return;
        _fac.resize(m + 1);
        _invfac.resize(m + 1);
        _inv.resize(m + 1);

        for (int i = n + 1; i <= m; i++) {
            _fac[i] = _fac[i - 1] * i;
        }
        _invfac[m] = _fac[m].inv();
        for (int i = m; i > n; i--) {
            _invfac[i - 1] = _invfac[i] * i;
            _inv[i] = _invfac[i] * _fac[i - 1];
        }
        n = m;
    }

    Z fac(int m) {
        if (m > n) init(2 * m);
        return _fac[m];
```

```
    }
    Z invfac(int m) {
        if (m > n) init(2 * m);
        return _invfac[m];
    }
    Z inv(int m) {
        if (m > n) init(2 * m);
        return _inv[m];
    }
    Z binom(int n, int m) {
        if (n < m || m < 0) return 0;
        return fac(n) * invfac(m) * invfac(n - m);
    }
} comb;
```

3.12 det

```
ll MOD;
int cal(vector<vector<int>>& a, int n) {
    ll flag = 1;
    // 转化成上三角矩阵
    for (int i = 1; i <= n; ++i) {  // 枚举行
        for (int k = i + 1; k <= n; ++k) {
            while (a[i][i]) {  // 辗转相除
                ll tim = a[k][i] / a[i][i];
                for (int j = i; j <= n; ++j)
                    a[k][j] = (a[k][j] - tim * a[i][j] % MOD + MOD) % MOD;
                swap(a[k], a[i]);  // 把较小的放上去
                flag = -flag;
            }
            swap(a[k], a[i]);
            flag = -flag;
        }
    }
    ll res = 1;
    for (int i = 1; i <= n; ++i)
        res = res * a[i][i] % MOD;
    res *= flag;
    return (res + MOD) % MOD;
}
void solve() {
    int n;
    cin >> n >> MOD;
    vector b(n + 1, vector<int>(n + 1));
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            cin >> b[i][j];
    ll ans = cal(b, n);
    cout << ans << endl;
}
```

3.13 int128

```
using uint = unsigned;
const int MOD = 998244353; // NTT模数

// 模加法
int Add(int x, int y) { return (x + y >= MOD) ? x + y - MOD : x + y; }
// 模减法
int Dec(int x, int y) { return (x - y < 0) ? x - y + MOD : x - y; }
// 模乘法
int mul(int x, int y) { return 1ll * x * y % MOD; }
// 快速幂计算
uint qp(uint a, int b) {
    uint res = 1;
    for (; b; b >>= 1, a = mul(a, a))
        if (b & 1)
            res = mul(res, a);
    return res;
}

namespace NTT {
    int sz; // FFT大小
    uint w[2500005], w_mf[2500005]; // 存储预计算的单位根及其乘法因子
    // 计算乘法因子
    int mf(int x) { return (1ll * x << 32) / MOD; }
    // 初始化NTT
    void init(int n) {
        for (sz = 2; sz < n; sz <<= 1);
        uint pr = qp(3, (MOD - 1) / sz);
        w[sz / 2] = 1;
        w_mf[sz / 2] = mf(1);
        for (int i = 1; i < sz / 2; i++) {
            w[sz / 2 + i] = mul(w[sz / 2 + i - 1], pr);
```

```
        w_mf[sz / 2 + i] = mf(w[sz / 2 + i]);
    }
    for (int i = sz / 2 - 1; i; i--) {
        w[i] = w[i << 1];
        w_mf[i] = w_mf[i << 1];
    }
}
// 前向NTT
void ntt(vector<uint>& A, int L) {
    for (int d = L >> 1; d; d >>= 1) {
        for (int i = 0; i < L; i += (d << 1)) {
            for (int j = 0; j < d; j++) {
                uint x = A[i + j] + A[i + d + j];
                if (x >= 2 * MOD) x -= 2 * MOD;
                ll t = A[i + j] + 2 * MOD - A[i + d + j];
                ll q = t * w_mf[d + j] >> 32;
                int y = t * w[d + j] - q * MOD;
                A[i + j] = x;
                A[i + d + j] = y;
            }
        }
    }
    for (int i = 0; i < L; i++) {
        if (A[i] >= MOD) A[i] -= MOD;
    }
}
// 逆NTT
void intt(vector<uint>& A, int L) {
    for (int d = 1; d < L; d <<= 1) {
        for (int i = 0; i < L; i += (d << 1)) {
            for (int j = 0; j < d; j++) {
                uint x = A[i + j];
                if (x >= 2 * MOD) x -= 2 * MOD;
                ll t = A[i + d + j];
                ll q = t * w_mf[d + j] >> 32;
                int y = t * w[d + j] - q * MOD;
                A[i + j] = x + y;
                A[i + d + j] = x + 2 * MOD - y;
            }
        }
    }
    int k = (L & (-L));
    reverse(A.begin() + 1, A.end());
    for (int i = 0; i < L; i++) {
        ll m = -A[i] & (L - 1);
        A[i] = (A[i] + m * MOD) / k;
        if (A[i] >= MOD) A[i] -= MOD;
    }
}
}

struct bigint {
    vector<int> nums; // 存储大整数的每一位
    int operator[](const int& k)const { return nums[k]; }
    int& operator[](const int& k) { return nums[k]; }
    int size() { return nums.size(); }
    void push_back(int x) { nums.push_back(x); }
    // 从整数构造大整数
    bigint(int x = 0) {
        do {
            nums.push_back(x % 10);
            x /= 10;
        } while (x);
    }
    // 从字符串构造大整数
    bigint(string s) {
        for (int i = s.size() - 1; i >= 0; i--)
            nums.push_back(s[i] - '0');
        trim();
    }
    // 去掉多余的前导零
    void trim() {
        while (nums.size() > 1 && nums.back() == 0) {
            nums.pop_back();
        }
    }
    // 清空大整数
    void clear() {
        nums.clear();
    }
    // 输入大整数
    friend istream& operator>>(istream& cin, bigint& num) {
        string tnum;
        cin >> tnum;
        num = tnum;
        return cin;
    }
    // 输出大整数
```

```
    friend ostream& operator<<(ostream& cout, bigint num) {
        bool start = false;
        for (int i = num.size() - 1; i >= 0; i--) {
            if (!start && num[i] == 0)
                continue;
            start = true;
            cout << num[i];
        }
        if (!start)
            cout << 0;
        return cout;
    }
};

// 比较运算符重载
bool operator<(bigint a, bigint b) {
    if (a.size() != b.size())
        return a.size() < b.size();
    for (int i = a.size() - 1; i >= 0; i--)
        if (a[i] != b[i])
            return a[i] < b[i];
    return false;
}

bool operator>(bigint a, bigint b) {
    return b < a;
}

bool operator<=(bigint a, bigint b) {
    return !(a > b);
}

bool operator>=(bigint a, bigint b) {
    return !(a < b);
}

bool operator==(bigint a, bigint b) {
    return !(a < b) && !(a > b);
}

bool operator!=(bigint a, bigint b) {
    return a < b || a > b;
}

// 大整数加法
bigint operator+(bigint a, bigint b) {
    bigint res;
    res.clear();
    int t = 0;
    int mx = max(a.size(), b.size());
    for (int i = 0; i < mx || t; i++) {
        if (i < a.size()) {
            t += a[i];
        }
        if (i < b.size()) {
            t += b[i];
        }
        res.push_back(t % 10);
        t /= 10;
    }
    res.trim();
    return res;
}

// 大整数减法
bigint operator-(bigint a, bigint b) {
    bigint res(a);
    bigint sub(b);
    int flag = 0;
    int len = res.size();
    while (sub.size() < res.size())
        sub.push_back(0);
    for (int i = 0; i < len; i++) {
        if (res[i] + flag >= sub[i]) {
            res[i] = res[i] + flag - sub[i];
            flag = 0;
        }
        else {
            res[i] = res[i] + 10 + flag - sub[i];
            flag = -1;
        }
    }
    res.trim();
    return res;
}

// 大整数乘法（nlogn）
bigint operator*(bigint a, bigint b) {
```

```
    bigint res;
    res.nums.pop_back();
    int dega = a.size() - 1, degb = b.size() - 1;
    int n = dega + degb + 1;
    int lim;
    for (lim = 1; lim < n; lim <<= 1);
    NTT::init(lim);
    vector<uint> A(lim);
    for (int i = 0; i <= dega; i++) A[i] = a[i];
    vector<uint> B(lim);
    for (int i = 0; i <= degb; i++) B[i] = b[i];
    NTT::ntt(A, lim);
    NTT::ntt(B, lim);
    for (int i = 0; i < lim; i++) A[i] = mul(A[i], B[i]);
    NTT::intt(A, lim);
    for (int i = 0, t = 0; i < lim || t; i++) {
        if (i < lim) t += A[i];
        res.push_back(t % 10);
        t /= 10;
    }
    res.trim();
    return res;
}

// 大整数与长整数乘法
bigint operator*(bigint a, ll b) {
    bigint res(a);
    int carry = 0;
    for (int i = 0; i < a.size(); i++) {
        carry += a[i] * b;
        res[i] = carry % 10;
        carry /= 10;
    }
    while (carry > 0) {
        res.push_back(carry % 10);
        carry /= 10;
    }
    return res;
}

// 大整数除法
bigint operator/(bigint a, bigint b) {
    bigint tnum(a);
    if (a < b)
        return 0;
    int n = a.size() - b.size();
    b.nums.insert(b.nums.begin(), n, 0);
    if (tnum >= b) {
        n++;
        b.nums.insert(b.nums.begin(), 0);
    }
    bigint ans;
    ans.nums.assign(n, 0);
    int n2 = b.size();
    while (n--) {
        n2--;
        b.nums.erase(b.nums.begin());
        while (!(tnum < b)) {
            int n1 = tnum.size();
            for (int j = 0; j < n2; j++) {
                tnum[j] -= b[j];
                if (tnum[j] < 0) {
                    tnum[j + 1]--;
                    tnum[j] += 10;
                }
            }
            tnum.trim();
            ans[n]++;
        }
    }
    ans.trim();
    return ans;
}

// 大整数与长整数除法
bigint operator/(bigint a, ll b) {
    bigint ans;
    ans.clear();
    int r = 0;
    for (int i = a.size() - 1; i >= 0; i--) {
        r = r % b * 10 + a[i];
        ans.push_back(r / b);
    }
    reverse(ans.nums.begin(), ans.nums.end());
    ans.trim();
    return ans;
}
```

```
// 大整数取模
bigint operator%(bigint a, bigint b) {
    bigint div_res = a / b;
    return a - div_res * b;
}

// 大整数与长整数取模
bigint operator%(bigint a, ll b) {
    bigint div_res = a / b;
    return a - div_res * b;
}

// 大整数快速幂
bigint qp(bigint a, ll n) {
    bigint res(1);
    while (n) {
        if (n & 1) res = res * a;
        a = a * a;
        n >>= 1;
    }
    return res;
}

// 大整数组合数
bigint comb(bigint n, bigint m) {
    bigint res = 1;
    for (bigint up = n, down = 1; down <= m; up = up - 1, down = down + 1)
        res = res * up, res = res / down;
    return res;
}
```

3.14 linerbasis

```
int qmi(int a, int b) {
    int res = 1;
    while (b) {
        if (b & 1)
            res = res * a % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return res;
}
struct linerbasis {
    static const int mxl = 30;
    int a[mxl + 1];
    int n = 0;          // 尝试插入次数
    int tot = 0;        // 线性基大小
    vector<int> tmp;    // 有效位集中

    linerbasis() {
        std::fill(a, a + mxl + 1, 0);
    }

    bool insert(int t) {
        n++;
        for (int j = mxl; j >= 0; j--) {
            int u = (t >> j) & 1;
            if (u == 0)
                continue;
            if (a[j])
                t ^= a[j];
            else {
                for (int k = 0; k < j; k++)
                    if ((t >> k) & 1)
                        t ^= a[k];
                for (int k = j + 1; k <= mxl; k++)
                    if ((a[k] >> j) & 1)
                        a[k] ^= t;
                a[j] = t;
                tot++;
                return true;
            }
        }
        return false;
    }

    int querymx(int x = 0) {  // 与x能异或出来的最大值，默认是x=0表示内部自己异或的最大值
        int ans = x;
        for (int i = mxl; i >= 0; i--) ans = max(ans, ans ^ a[i]);
        return ans;
    }
    int querymn(int x = 0) {  // 与x能异或出来的最小值，默认是x=0表示内部自己异或的最小值
        int ans = x;
```

```
    for (int i = mxl; i >= 0; i--) ans = min(ans, ans ^ a[i]);
    return ans;
}

void initkth() {
    static bool initialized = false;
    if (initialized)
        return;
    for (int i = 0; i <= mxl; i++) {
        if (a[i])
            tmp.push_back(a[i]);
    }
    deb(tmp);
    initialized = true;
}
// 第k小
int querekthmin(int k, bool tkzo = false) {  // 第0小开始算
    initkth();
    int res = 0;
    if (tkzo == 0) {
        // 如果题目没有考虑空集，我们需要考虑能不能非空子集出现0
        if (tot == n)
            k++;
    }
    if (k >= (1LL << tot))
        return -1;
    for (int j = 0; j < tot; j++) {
        if ((k >> j) & 1)
            res ^= tmp[j];
    }
    return res;
}
// 值为x的下标
int querypos(int x) {
    int l = 0, r = (1 << tot) - 1;
    while (l < r) {
        int mid = (l + r) >> 1;
        if (querekthmin(mid, true) >= x)
            r = mid;
        else
            l = mid + 1;
    }
    int res = qmi(2, n - tot) * l % mod + 1;
    res %= mod;
    return res;
}
};
```

3.15 matrix

```
struct Matrix {
    using LL = long long;
    std::vector<std::vector<LL>> mat;
    Matrix() : mat{} {}

    /// @brief 生成n行m列空矩阵
    /// @param n 行数
    /// @param m 列数
    Matrix(int n, int m) : mat(n, std::vector<LL>(m)) {}

    /// @brief 生成单位矩阵 E
    /// @param n size
    Matrix(int n) : mat(n, std::vector<LL>(n)) {
        for (int i = 0; i < n; i++) {
            mat[i][i] = 1;
        }
    }

    int size() const { return mat.size(); }
    auto &operator[](int n) { return mat[n]; }
    auto &operator[](int n) const { return mat[n]; }

    auto begin() { return mat.begin(); }
    auto begin() const { return mat.begin(); }
    auto end() { return mat.end(); }
    auto end() const { return mat.end(); }

    Matrix operator*(const Matrix &o) const {
        Matrix res(mat.size(), o.size());
        for (size_t i = 0; i < res.size(); i++) {
            for (size_t k = 0; k < mat[0].size(); k++) {
                if (!mat[i][k])
                    continue;
                for (size_t j = 0; j < res[0].size(); j++) {
                    res[i][j] += mat[i][k] * o[k][j];
                    res[i][j] %= Mod;
```

```
            }
        }
    }
    return res;
}

Matrix operator*(const LL &a) const {
    Matrix res;
    res.mat = mat;
    for (int i = 0; i < res.size(); i++) {
        for (int j = 0; j < res[0].size(); j++) {
            res[i][j] *= a;
        }
    }
    return res;
}

friend Matrix operator*(const LL &a, const Matrix &o) { return o * a; }

Matrix operator+(const Matrix &o) const {
    Matrix res = *this;
    for (int i = 0; i < res.size(); i++) {
        for (int j = 0; j < res[0].size(); j++) {
            res[i][j] = (mat[i][j] + o[i][j]) % Mod;
        }
    }
    return res;
}

Matrix operator-(const Matrix &o) const { return -1 * o + *this; }

Matrix pow(LL k) const {
    Matrix a = *this;
    Matrix ans(this->size());
    while (k) {
        if (k & 1)
            ans = ans * a;
        a = a * a;
        k >>= 1;
    }
    return ans;
}

Matrix reverse() const {
    Matrix res(mat[0].size(), mat.size());
    for (int i = 0; i < mat[0].size(); i++) {
        for (int j = 0; j < mat.size(); j++) {
            res.mat[i][j] = mat[j][i];
        }
    }
    return res;
}
};
```

3.16 matrix_tree

```
int cal(vector<vector<int>>& a, int n) {//针对没有逆元
    ll flag = 1;
    // 转化成上三角矩阵
    for (int i = 1; i <= n; ++i) {  // 枚举行
        for (int k = i + 1; k <= n; ++k) {
            while (a[i][i]) {  // 辗转相除
                ll tim = a[k][i] / a[i][i];
                for (int j = i; j <= n; ++j)
                    a[k][j] = (a[k][j] - tim * a[i][j]);
                swap(a[k], a[i]);  // 把较小的放上去
                flag = -flag;
            }
            swap(a[k], a[i]);
            flag = -flag;
        }
    }
    ll res = 1;
    for (int i = 1; i <= n; ++i)
        res = res * a[i][i];
    res *= flag;
    return res;
}
void solve() {
    int n, m;
    cin >> n >> m;
    vector<vector<int>> b(n + 1, vector<int>(n + 1));
    for (int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        b[u][u]++;
```

```
        b[v][v]++;
        b[u][v]--;
        b[v][u]--;
    }
    int ans = cal(b, n - 1);
    cout << ans << endl;
}
```

3.17 modint

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

using i64 = long long;
template <class T>
constexpr T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}
//-------------------------------------------//
template <int P>
struct MInt {
    int x;
    constexpr MInt() : x{} {}
    constexpr MInt(i64 x) : x{norm(x % getMod())} {}

    static int Mod;
    constexpr static int getMod() {
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(int Mod_) {
        Mod = Mod_;
    }
    constexpr int norm(int x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
        return x;
    }
    constexpr int val() const {
        return x;
    }
    explicit constexpr operator int() const {
        return x;
    }
    constexpr MInt operator-() const {
        MInt res;
        res.x = norm(getMod() - x);
        return res;
    }
    constexpr MInt inv() const {
        assert(x != 0);
        return power(*this, getMod() - 2);
    }
    constexpr MInt &operator*=(MInt rhs) & {
        x = 1LL * x * rhs.x % getMod();
        return *this;
    }
    constexpr MInt &operator+=(MInt rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }
    constexpr MInt &operator-=(MInt rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }
    constexpr MInt &operator/=(MInt rhs) & {
        return *this *= rhs.inv();
    }
    friend constexpr MInt operator*(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr MInt operator+(MInt lhs, MInt rhs) {
```

```
        MInt res = lhs;
        res += rhs;
        return res;
    }
    friend constexpr MInt operator-(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr MInt operator/(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr std::istream &operator>>(std::istream &is, MInt &a) {
        i64 v;
        is >> v;
        a = MInt(v);
        return is;
    }
    friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a) {
        return os << a.val();
    }
    friend constexpr bool operator==(MInt lhs, MInt rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MInt lhs, MInt rhs) {
        return lhs.val() != rhs.val();
    }
};

template <>
int MInt<0>::Mod = 998244353;
//-------------------------------------//
constexpr int P = 998244353;
using Z = MInt<P>;
 //constexpr Z CInv = Z(n).inv();
```

3.18 modll

```
using i64 = long long;
template <class T>
constexpr T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}
constexpr i64 mul(i64 a, i64 b, i64 p) {
    i64 res = a * b - i64(1.L * a * b / p) * p;
    res %= p;
    if (res < 0) {
        res += p;
    }
    return res;
}
template <i64 P>
struct MLong {
    i64 x;
    constexpr MLong() : x{} {}
    constexpr MLong(i64 x) : x{norm(x % getMod())} {}

    static i64 Mod;
    constexpr static i64 getMod() {
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(i64 Mod_) {
        Mod = Mod_;
    }
    constexpr i64 norm(i64 x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
        return x;
    }
    constexpr i64 val() const {
```

```
    return x;
}
explicit constexpr operator i64() const {
    return x;
}
constexpr MLong operator-() const {
    MLong res;
    res.x = norm(getMod() - x);
    return res;
}
constexpr MLong inv() const {
    assert(x != 0);
    return power(*this, getMod() - 2);
}
constexpr MLong &operator*=(MLong rhs) & {
    x = mul(x, rhs.x, getMod());
    return *this;
}
constexpr MLong &operator+=(MLong rhs) & {
    x = norm(x + rhs.x);
    return *this;
}
constexpr MLong &operator-=(MLong rhs) & {
    x = norm(x - rhs.x);
    return *this;
}
constexpr MLong &operator/=(MLong rhs) & {
    return *this *= rhs.inv();
}
friend constexpr MLong operator*(MLong lhs, MLong rhs) {
    MLong res = lhs;
    res *= rhs;
    return res;
}
friend constexpr MLong operator+(MLong lhs, MLong rhs) {
    MLong res = lhs;
    res += rhs;
    return res;
}
friend constexpr MLong operator-(MLong lhs, MLong rhs) {
    MLong res = lhs;
    res -= rhs;
    return res;
}
friend constexpr MLong operator/(MLong lhs, MLong rhs) {
    MLong res = lhs;
    res /= rhs;
    return res;
}
friend constexpr std::istream &operator>>(std::istream &is, MLong &a) {
    i64 v;
    is >> v;
    a = MLong(v);
    return is;
}
friend constexpr std::ostream &operator<<(std::ostream &os, const MLong &a)
  {
    return os << a.val();
}
friend constexpr bool operator==(MLong lhs, MLong rhs) {
    return lhs.val() == rhs.val();
}
friend constexpr bool operator!=(MLong lhs, MLong rhs) {
    return lhs.val() != rhs.val();
}
};
template <>
i64 MLong<0LL>::Mod = i64(1E18) + 9;
//-------------------------------//
constexpr int P = i64(1E18) + 9;
using Z = MLong<P>;
```

3.19 pre_linerbasis

```
struct prelinerbasis {
    static const int mxl = 20;
    vector<array<int, mxl + 1>> p;     // p[id][i]表示前id个数，第i位的线性基
    vector<array<int, mxl + 1>> pos;  // pos[id][i]表示构造基p[id][i]的元素的下
      标最大值
    prelinerbasis() {}
    prelinerbasis(int n) { init(n); }
    void init(int n) {
        p.resize(n + 1);
        pos.resize(n + 1);
    }
    void insert(int x, int id) {
```

```
        for (int i = 0; i <= mxl; i++) {  // 复制前一版
            p[id][i] = p[id - 1][i];
            pos[id][i] = pos[id - 1][i];
        }
        int cur = id;
        for (int i = mxl; i >= 0; i--) {
            if (x >> i & 1) {
                if (!p[id][i]) {  // 不存在则加入
                    p[id][i] = x;
                    pos[id][i] = cur;
                    break;
                }
                // 存在则先交换，后异或
                if (pos[id][i] < cur)
                    swap(p[id][i], x), swap(pos[id][i], cur);
                x ^= p[id][i];
            }
        }
    }
    int query(int l, int r, int x = 0) {  //[l,r]的线性基与x异或的最大值
        int ans = x;
        for (int i = mxl; i >= 0; i--)
            if (pos[r][i] >= l)
                ans = max(ans, ans ^ p[r][i]);
        return ans;
    }
};

void solve() {
    int n;
    cin >> n;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++) cin >> a[i];
    prelinerbasis plb(n);
    for (int i = 1; i <= n; i++) plb.insert(a[i], i);
    int q;
    cin >> q;
    for (int i = 1; i <= q; i++) {
        int l, r;
        cin >> l >> r;
        cout << plb.query(l, r) << endl;
    }
}
```

3.20 prelinerbasis_tree

```
struct edge {
    int v, w;
};
struct HLD {
    int n;
    vector<int> siz, top, parent, l, r, hson, dep;
    vector<vector<edge>> adj;
    int idx;
    // 加数据结构
    vector<int> a;
    HLD() {}
    HLD(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        siz.resize(n + 1), hson.resize(n + 1), top.resize(n + 1);
        parent.resize(n + 1);
        l.resize(n + 1), r.resize(n + 1);
        idx = 0;
        adj.resize(n + 1), dep.resize(n + 1);
        // 根据题目要求加数据结构
        a.resize(n + 1);
    }
    void addEdge(int u, int v, int w) {
        adj[u].push_back({v, w});
    }
    void work(auto& plb, int root = 1) {
        top[root] = root;
        dep[root] = 1;
        dfs1(root, 0, plb);
        dfs2(root, root);
    }
    void dfs1(int u, int f, auto& plb) {  // 搞fa,dep,son
        siz[u] = 1;
        plb.insert(a[u], u, f, *this);  // 继承父节点，插入当前节点
        for (auto [v, w] : adj[u]) {
            if (v == f)
                continue;
            parent[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v, u, plb);
```

```
            siz[u] += siz[v];
            if (siz[hson[u]] < siz[v])
                hson[u] = v;
        }
    }
    void dfs2(int u, int t) {  // 搞top
        top[u] = t;            // 记录链头
        l[u] = ++idx;
        if (!hson[u]) {
            r[u] = idx;
            return;
        } // 无重儿子
        dfs2(hson[u], t);  // 搜重儿子
        for (auto [v, w] : adj[u]) {
            if (v == parent[u] || v == hson[u])
                continue;
            dfs2(v, v);  // 搜轻儿子
        }
        r[u] = idx;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = parent[top[u]];
            } else {
                v = parent[top[v]];
            }
        }
        return dep[u] < dep[v] ? u : v;
    }
    bool isAncester(int u, int v) {  // 判断u是不是v的祖先
        return l[u] <= l[v] && r[v] <= r[u];
    }
};
struct prelinerbasis_tree {
    static const int mxl = 60;
    vector<array<int, mxl + 1>> p;    // p[id][i]表示前id个数，第i位的线性基
    vector<array<int, mxl + 1>> pos;  // pos[id][i]表示构造基p[id][i]的元素的下
      标最大值
    prelinerbasis_tree() {}
    prelinerbasis_tree(int n) { init(n); }
    void init(int n) {
        p.resize(n + 1);
        pos.resize(n + 1);
    }
    void insert(int x, int u, int f, auto& hld) {
        deb(x, u, f);
        for (int i = 0; i <= mxl; i++)  // 复制父版
            p[u][i] = p[f][i], pos[u][i] = pos[f][i];
        int cur = u;
        for (int i = mxl; i >= 0; --i) {
            if ((x >> i) & 1) {
                deb(x, i);
                if (!p[u][i]) {  // 不存在则加入
                    p[u][i] = x;
                    pos[u][i] = cur;
                    break;
                }
                // 存在则先交换，后异或
                if (hld.dep[pos[u][i]] < hld.dep[cur])
                    swap(x, p[u][i]), swap(pos[u][i], cur);
                x ^= p[u][i];
            }
        }
    }
    int querymx(const vector<int>& b, int x = 0) {
        int res = x;
        for (int i = mxl; i >= 0; i--) res = max(res, res ^ b[i]);
        return res;
    }
    vector<int> query(int x, int y, auto& hld) {  // 查询x到y简单路径构造的线性
      基
        int tmplca = hld.lca(x, y);
        deb(x, y, tmplca);
        vector<int> b(mxl + 1);
        for (int i = mxl; i >= 0; --i) {  // 从x到根的链中提取出x~lca的线性基
            if (hld.dep[pos[x][i]] >= hld.dep[tmplca])
                b[i] = p[x][i];
        }
        for (int i = mxl; i >= 0; --i) {  // 暴力合并y~lca的基
            if (hld.dep[pos[y][i]] < hld.dep[tmplca])
                continue;
            int x = p[y][i];  // 提取y~lca链的基
            for (int j = i; j >= 0; --j) {
                if (x >> j & 1) {
                    if (!b[j]) {
                        b[j] = x;
                        break;
```

```
                    }
                    x ^= b[j];
                }
            }
        }
        return b;
    }
};

void solve() {
    int n, q;
    cin >> n >> q;
    vector<int> a(n + 1);
    HLD hld(n);
    prelinerbasis_tree plb(n);
    for (int i = 1; i <= n; i++) cin >> hld.a[i];
    for (int i = 1; i <= n - 1; i++) {
        int u, v;
        cin >> u >> v;
        hld.addEdge(u, v, 1);
        hld.addEdge(v, u, 1);
    }
    hld.work(plb, 1);
    for (int i = 1; i <= n; i++) deb(i, hld.dep[i]);
    for (int i = 1; i <= q; i++) {
        int u, v;
        cin >> u >> v;
        auto b = plb.query(u, v, hld);
        deb(b);
        cout << plb.querymx(b) << endl;
    }
}
```

3.21 simpson

```
const double eps = 1e-10;
double a, b, c, d, l, r;
//时间复杂度：O(log(n/eps))
// tips:要注意保证给的初始区间的积分是收敛的并且不要出现无定义点
// 反常积分的发散部分特判
// 1.对于初始区间，有时候显然不能直接赋值0和无穷大，
// 2.左端点复制成eps。
// 3.考虑右端点，根据题目条件的取值，
//当x=20（一个具体值）的时候代入发现已经远小于eps了故右端点设计为20.
double f(double x) {  // 积分函数
    return (c * x + d) / (a * x + b);
}
double simpson(double l, double r) {  // 辛普森公式
    return (r - l) * (f(l) + f(r) + 4 * f((l + r) / 2)) / 6;
}  // 二次函数特性

double asr(double l, double r, double ans) {  // 自适应
    // 分段simpson，如果划分足够小，低于误差就可以
    auto m = (l + r) / 2, a = simpson(l, m), b = simpson(m, r);
    if (fabs(a + b - ans) < eps)
        return ans;
    return asr(l, m, a) + asr(m, r, b);
}
int main() {
    scanf("%lf%lf%lf%lf%lf%lf", &a, &b, &c, &d, &l, &r);
    printf("%.6lf", asr(l, r, simpson(l, r)));
    return 0;
}
```

4 Misc

4.1 坐标转换

```
int id(int x,int y,int m){
  //m列
  return m*(x-1)+y;
}
pii rid(int u,int m){
  int x=(u+m-1)/m;//m列
  int y=u%m;if(y==0)y+=m;
  return make_pair(x,y);
}
```

4.2 小数保留问题

```
#include <iostream>
```

```
using namespace std;
要得到四舍五入小数点后的结果，我们可以将小数转换为整数来处理，然后再转换为小数。
// 用于四舍五入
int round_0 (double n)
{
  // 若为负数，则先化为正数再进行四舍五入
  if (n > 0)
    return n - int(n) >= 0.5 ? int(n)+1 : int(n);
  else
    return -n - int(-n) >= 0.5 ? -(int(-n) + 1) : -int(-n);
}

int main()
{
  double a = 1.2345;
  double b = 1.2355;
  double n_a = -1.2345;
  double n_b = -1.2355;

  a = round_0(a * 100.0) / 100.0;
  b = round_0(b * 100.0) / 100.0;
  n_a = round_0(n_a * 100.0) / 100.0;
  n_b = round_0(n_b * 100.0) / 100.0;

  cout << a << endl;  // 1.23
  cout << b << endl;  // 1.24
  cout << n_a << endl;  //-1.23
  cout << n_b << endl;  // -1.24
  return 0;
}
```

4.3 日期问题

```
// Mon = 0, ... % 7
// days since 1/1/1
// 从公元1年1月1日到给定日期（年 y、月 m、日 d）的天数
int getday(int y, int m, int d) {
    if (m < 3)
        --y, m += 12;
    return (365 * y + y / 4 - y / 100 + y / 400 + (153 * (m - 3) + 2) / 5 + d -
      307);
}
// 自（公元1年1月1日）以来的n天数转换为y年m月d号
void date(int n, int& y, int& m, int& d) {
    n += 429 + ((4 * n + 1227) / 146097 + 1) * 3 / 4;
    y = (4 * n - 489) / 1461;
    n -= y * 1461 / 4;
    m = (5 * n - 1) / 153;
    d = n - m * 153 / 5;
    if (--m > 12)
        m -= 12, ++y;
}
// 已知年月日，求星期数。
int week(int y, int m, int d) {
    if (m <= 2)
        m += 12, y--;
    return (d + 2 * m + 3 * (m + 1) / 5 + y + y / 4 - y / 100 + y / 400) % 7 +
      1;
}
//------------------------------------------
//记忆版本
int months[13] = {
    0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
};

int is_leap(int year)//闰年判断
{
    if (year % 4 == 0 && year % 100 || year % 400 == 0)
        return 1;
    return 0;
}

int get_days(int y, int m)//给年月，输出日
{
    int s = months[m];
    if (m == 2) return s + is_leap(y);
    return s;
}
```

4.4 表达式求值

给定一个表达式，其中运算符仅包含 +,-,*,/（加 减 乘 整除），可能包含括号，请你求出表达式的最终值。
数据保证给定的表达式合法。

题目保证符号 – 只作为减号出现，不会作为负号出现，例如，–1+2,(2+2)*(–(1+1)+2) 之类表达式均不会出现。
题目保证表达式中所有数字均为正整数。
题目保证表达式在中间计算过程以及结果中，均不超过 int
题目中的整除是指向 0取整,也就是说对于大于的结果向下取整，例如 5/3=1对于小于 0的结果向上取整,例如 5/(1–4)=–1

```
stack<int> num;
stack<char> op;

void eval()
{
    auto b = num.top(); num.pop();
    auto a = num.top(); num.pop();
    auto c = op.top(); op.pop();
    int x;
    if (c == '+') x = a + b;
    else if (c == '–') x = a – b;
    else if (c == '*') x = a * b;
    else x = a / b;
    num.push(x);
}

int main()
{
    unordered_map<char, int> pr{{'+', 1}, {'–', 1}, {'*', 2}, {'/', 2}};
    string str;
    cin >> str;
    for (int i = 0; i < str.size(); i ++ )
    {
        auto c = str[i];
        if (isdigit(c))
        {
            int x = 0, j = i;
            while (j < str.size() && isdigit(str[j]))
                x = x * 10 + str[j ++ ] – '0';
            i = j – 1;
            num.push(x);
        }
        else if (c == '(') op.push(c);
        else if (c == ')')
        {
            while (op.top() != '(') eval();
            op.pop();
        }
        else
        {
            while (op.size() && op.top() != '(' && pr[op.top()] >= pr[c]) eval()
                ;
            op.push(c);
        }
    }
    while (op.size()) eval();
    cout << num.top() << endl;
    return 0;
}
```

4.5 魔方

本题的输入是一个魔方的展开图，我们以黄色为中心块的面作为顶面，红色为中心块的面作为前面，绿色为中心块的面作为右面。
定义六种魔方转动操作类型，分别如下:
1. "R1": 右面顺时针旋转90度
2. "R2": 右面逆时针旋转90度
3. "U1": 顶面顺时针旋转90度
4. "U2": 顶面逆时针旋转90度
5. "F1": 前面顺时针旋转90度
6. "F2": 前面逆时针旋转90度

Sol:
特殊限制: 存在 6 种不同操作，且一定存在步数小于等于 8 的正解
直接暴力 dfs 或 bfs 搜索答案即可。

考虑魔方在转动某一面时，转动面的 9 个颜色会进行顺时针或逆时针移位，与转动面相邻的 4 个侧面中，直接
与转动面相邻的 3 个颜色也会按顺时针或逆时针顺序循环移位。故可使用 struct Plane 定义一个面按顺序排列的九种
颜色，结构体内部实现单面的顺时针或逆时针移位。使用 struct Cube 定义整个魔方的状态，每次操作先转动单面，
再按顺序移位相邻 4 个侧面中的 3 个相邻颜色即可。

```
      O O O
      O O O
B B B Y Y Y G G G W W W
B B B Y Y Y G G G W W W
B B B Y Y Y G G G W W W
      R R R
      R R R
      R R R
```

```
#include <bits/stdc++.h>
#define Buff ios::sync_with_stdio(false), cin.tie(nullptr), cout.tie(nullptr)
using namespace std;
typedef vector<char> vchar;

struct Plane  // 面
{
    char c[9];

    Plane() {
        memset(c, 0, sizeof(c));
    }

    void set(char x)  // 设置颜色，本题非必要
    {
        memset(c, x, sizeof(c));
    }

    bool check() const  // 判断此面颜色是否相同
    {
        for (int i = 1; i < 9; i++) {
            if (c[i] != c[0])
                return false;
        }
        return true;
    }

    void CRotate()  // 此面顺时针旋转90度
    {
        char b = c[0];
        c[0] = c[6];
        c[6] = c[8];
        c[8] = c[2];
        c[2] = b;

        b = c[1];
        c[1] = c[3];
        c[3] = c[7];
        c[7] = c[5];
        c[5] = b;
    }

    void CCRotate()  // 此面逆时针旋转90度
    {
        char b = c[0];
        c[0] = c[2];
        c[2] = c[8];
        c[8] = c[6];
        c[6] = b;

        b = c[1];
        c[1] = c[5];
        c[5] = c[7];
        c[7] = c[3];
        c[3] = b;
    }
};

struct Cube  // 魔方
{
    Plane p[6];

    Cube() {
        init();
    }

    void init()  // 初始化颜色，本题非必要
    {
        p[0].set('R');
        p[1].set('G');
        p[2].set('Y');
        p[3].set('O');
        p[4].set('B');
        p[5].set('W');
    }

    bool check() const  // 判断是否已还原
    {
        for (int i = 0; i < 6; i++) {
            if (!p[i].check())
                return false;
        }
        return true;
    }

    void operate(int opid)  // 操作
    {
        if (opid == 1)
```

```
        R1();
    else if (opid == 2)
        R2();
    else if (opid == 3)
        U1();
    else if (opid == 4)
        U2();
    else if (opid == 5)
        F1();
    else if (opid == 6)
        F2();
}

void roperate(int opid)  // 逆操作
{
    if (opid == 1)
        R2();
    else if (opid == 2)
        R1();
    else if (opid == 3)
        U2();
    else if (opid == 4)
        U1();
    else if (opid == 5)
        F2();
    else if (opid == 6)
        F1();
}

void R1() {
    p[1].CRotate();  // 旋转面

    int updateplane[4] = {0, 5, 3, 2};  // 需要按顺序移位的相邻面
    int updateid[4][3] = {{2, 5, 8},    // 每面需要移位的颜色下标
                          {6, 3, 0},
                          {2, 5, 8},
                          {2, 5, 8}};

    update(updateid, updateplane);
}
void R2() {
    p[1].CCRotate();

    int updateplane[4] = {0, 2, 3, 5};
    int updateid[4][3] = {{2, 5, 8},
                          {2, 5, 8},
                          {2, 5, 8},
                          {6, 3, 0}};

    update(updateid, updateplane);
}

void U1() {
    p[2].CRotate();

    int updateplane[4] = {0, 1, 3, 4};
    int updateid[4][3] = {{0, 1, 2},
                          {6, 3, 0},
                          {8, 7, 6},
                          {2, 5, 8}};

    update(updateid, updateplane);
}
void U2() {
    p[2].CCRotate();

    int updateplane[4] = {0, 4, 3, 1};
    int updateid[4][3] = {{0, 1, 2},
                          {2, 5, 8},
                          {8, 7, 6},
                          {6, 3, 0}};

    update(updateid, updateplane);
}

void F1() {
    p[0].CRotate();

    int updateplane[4] = {1, 2, 4, 5};
    int updateid[4][3] = {{6, 7, 8},
                          {6, 7, 8},
                          {6, 7, 8},
                          {6, 7, 8}};

    update(updateid, updateplane);
}
void F2() {
    p[0].CCRotate();
```

```
    int updateplane[4] = {1, 5, 4, 2};
    int updateid[4][3] = {{6, 7, 8},
                          {6, 7, 8},
                          {6, 7, 8},
                          {6, 7, 8}};

    update(updateid, updateplane);
}

void update(int uid[4][3], int uplane[4])  // 循环移位相邻面颜色
{
    char buffer[3] = {p[uplane[0]].c[uid[0][0]],
                      p[uplane[0]].c[uid[0][1]],
                      p[uplane[0]].c[uid[0][2]]};

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++)
            p[uplane[i]].c[uid[i][j]] = p[uplane[i + 1]].c[uid[i + 1][j]];
    }

    p[uplane[3]].c[uid[3][0]] = buffer[0];
    p[uplane[3]].c[uid[3][1]] = buffer[1];
    p[uplane[3]].c[uid[3][2]] = buffer[2];
}

friend std::istream& operator>>(std::istream& os, Cube& cube)  // 输入
{
    std::string buffer;

    for (int i = 0; i < 9; i += 3) {
        for (int j = 0; j < 3; j++) {
            os >> buffer;
            cube.p[3].c[i + j] = buffer[0];
        }
    }

    for (int i = 0; i < 9; i += 3) {
        for (int j = 0; j < 3; j++) {
            os >> buffer;
            cube.p[4].c[i + j] = buffer[0];
        }
        for (int j = 0; j < 3; j++) {
            os >> buffer;
            cube.p[2].c[i + j] = buffer[0];
        }
        for (int j = 0; j < 3; j++) {
            os >> buffer;
            cube.p[1].c[i + j] = buffer[0];
        }
        for (int j = 0; j < 3; j++) {
            os >> buffer;
            cube.p[5].c[i + j] = buffer[0];
        }
    }

    for (int i = 0; i < 9; i += 3) {
        for (int j = 0; j < 3; j++) {
            os >> buffer;
            cube.p[0].c[i + j] = buffer[0];
        }
    }

    return os;
}

friend std::ostream& operator<<(std::ostream& os, const Cube& cube)  // 输出，本题非必要
{
    for (int i = 0; i < 9; i += 3) {
        os << "␣␣␣␣␣␣";
        for (int j = 0; j < 3; j++) {
            os << cube.p[3].c[i + j];
            if (j < 2)
                os << '␣';
        }
        os << '\n';
    }

    for (int i = 0; i < 9; i += 3) {
        for (int j = 0; j < 3; j++) {
            os << cube.p[4].c[i + j] << '␣';
        }
        for (int j = 0; j < 3; j++) {
            os << cube.p[2].c[i + j] << '␣';
        }
        for (int j = 0; j < 3; j++) {
            os << cube.p[1].c[i + j] << '␣';
```

```
            }
            for (int j = 0; j < 3; j++) {
                os << cube.p[5].c[i + j];
                if (j < 2)
                    os << '␣';
            }
            os << '\n';
        }

        for (int i = 0; i < 9; i += 3) {
            os << "␣␣␣␣␣␣";
            for (int j = 0; j < 3; j++) {
                os << cube.p[0].c[i + j];
                if (j < 2)
                    os << '␣';
            }
            os << '\n';
        }

        return os;
    }
};

//
  ////////////////////////////////////////////////////////////////////////////


void dfs(Cube& cube, vchar& way, bool& flag, size_t stop)  // dfs深搜
{
    if (way.size() >= stop)
        return;

    char ref = -1;  // 上次操作的逆操作序号
    if (!way.empty()) {
        ref = *(way.rbegin());
        if (ref & 1)
            ref++;
        else
            ref--;
    }

    for (char i = 1; i <= 6; i++) {
        if (i == ref)  // 剪枝
            continue;
        cube.operate(i);
        way.emplace_back(i);

        if (cube.check()) {
            flag = true;
            return;
        }

        dfs(cube, way, flag, stop);

        if (flag)
            return;

        cube.roperate(i);  // 回溯
        way.pop_back();
    }
}

vchar bfs(Cube icube)  // bfs宽搜
{
    using pcv = pair<Cube, vchar>;  // 保存的魔方状态以及操作顺序

    pcv ib;
    ib.first = icube;
    queue<pcv> q;
    q.push(ib);

    vchar ans;

    while (!q.empty()) {
        pcv b = q.front();
        q.pop();
        char ref = -1;  // 上次操作的逆操作序号
        if (!b.second.empty()) {
            ref = *(b.second.rbegin());
            if (ref & 1)
                ref++;
            else
                ref--;
        }

        for (char i = 1; i <= 6; i++) {
            if (i == ref)
```

```
                continue;

            pcv x = b;
            x.first.operate(i);
            x.second.emplace_back(i);

            if (x.first.check()) {
                ans = x.second;
                return ans;
            }

            q.push(x);
        }
    }

    return ans;
}

void solve() {
    Cube cube;
    cin >> cube;

    if (cube.check()) {
        cout << "0\n";
        return;
    }

    vchar ans;

    // ans = bfs(cube); // bfs调用

    bool flag = false;
    dfs(cube, ans, flag, 8);  // dfs调用

    cout << ans.size() << '\n';

    vector<string> map_op{"", "R1", "R2", "U1", "U2", "F1", "F2"};  // 操作映射
    for (char i : ans) {
        if (i >= 1 && i <= 6)
            cout << map_op[i] << '\n';
    }
}

int main() {
    Buff;
    int _N = 1;
    // cin >> _N;
    while (_N--)
        solve();
    return 0;
}
```

5 STL

5.1 ___int128_RW

```
// 需要开同步流！！！
__int128 read() {
    __int128 X = 0, f = 1;
    char c = getchar();
    while (!isdigit(c) && c != '-') c = getchar();
    if (c == '-')
        c = getchar(), f = -1;
    while (isdigit(c)) X = X * 10 + (c ^ 48), c = getchar();
    return X * f;
}
void write(__int128 x) {
    if (x < 0)
        x = -x, putchar('-');
    if (x > 9)
        write(x / 10);
    putchar(x % 10 + '0');
}
```

5.2 ___int128_gcd

```
using i128 = __int128;
i128 gcd(i128 a, i128 b) {
    return b ? gcd(b, a % b) : a;
}
```

5.3 ___int128_iostream

```
istream &operator>>(istream &is, __int128 &T) {
    char c;
    int f = 1;
    T = 0;
    c = is.get();
    while (c != '-' && !isdigit(c)) c = is.get();
    if (c == '-')
        f = -1, c = is.get();
    while (isdigit(c)) {
        T = T * 10 + (c - '0');
        c = is.get();
    }
    T = f * T;
    return is;
}
std::ostream &operator<<(std::ostream &os, __int128 &n) {
    std::string s;
    while (n) {
        s += '0' + n % 10;
        n /= 10;
    }
    std::reverse(s.begin(), s.end());
    return os << s;
}
```

5.4 chmax

```
template<class T>
void chmax(T &a, T b) {
    if (a < b) {
        a = b;
    }
}
```

5.5 custom_hash

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().
          time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
//unordered_map<int,int,custom_hash>mp;
```

5.6 div

```
using i64 = long long;
i64 ceilDiv(i64 n, i64 m) {
    if (n >= 0) {
        return (n + m - 1) / m;
    } else {
        return n / m;
    }
}

i64 floorDiv(i64 n, i64 m) {
    if (n >= 0) {
        return n / m;
    } else {
        return (n - m + 1) / m;
    }
}
```

5.7 pair_hash

```
struct pair_hash {
    template <class T1, class T2>
    std::size_t operator()(const std::pair<T1, T2> &p) const {
        auto hash1 = std::hash<T1>{}(p.first);
        auto hash2 = std::hash<T2>{}(p.second);
        return hash1 ^ (hash2 + 0x9e3779b9 + (hash1 << 6) + (hash1 >> 2));
    }
};
```

```
// std::unordered_map<std::pair<int, int>, int, pair_hash> mp;
```

5.8 sqrt

```
long long mysqrt(long long n) {
    long long s = std::sqrt(n);
    while (s * s > n) {
        s--;
    }
    while ((s + 1) * (s + 1) <= n) {
        s++;
    }
    return s;
}
```

6 String

6.1 AC

```
struct AC {                          // 定义了AhoCorasick结构体，用于实现Aho-Corasick字符串匹配算法
    static constexpr int asz = 26;   // 定义常量ALPHABET为26，表示字母表的大小（26个小写字母）
    struct Node {                    // 定义了内部结构体Node，表示Trie树的一个节点
        int len;                     // 节点对应的字符串的长度
        int fail;                    // 节点的后缀链接，指向最长的可以匹配的后缀节点
        array<int, asz> next;
        // 表示从当前节点到下一个节点的转换，数组大小为字母表大小
        Node() : len{0}, fail{0}, next{} {}  // 构造函数，初始化len为0，link为0，next数组全为0
    };

    vector<Node> t;  // 定义一个Node类型的向量，存储Trie树中的所有节点

    AC() {  // 构造函数，调用init函数初始化Trie树
        init();
    }

    void init() {              // 初始化Trie树，创建根节点和伪根节点
        t.assign(2, Node());   // 创建两个节点，分别是根节点和伪根节点
        t[0].next.fill(1);     // 将根节点的所有next指向伪根节点
        t[0].len = -1;         // 设置根节点的len为-1
    }

    int newNode() {            // 创建一个新节点，并返回其索引
        t.emplace_back();      // 向向量t中添加一个新的Node节点
        return t.size() - 1;   // 返回新节点的索引
    }

    int add(const string &a) {                          // 向Trie树中添加字符串，并返回最后一个字符对应的节点索引
        int p = 1;                                      // 从伪根节点开始
        for (auto c : a) {                              // 遍历字符串中的每个字符
            int x = c - 'a';                            // 计算字符在字母表中的索引
            if (t[p].next[x] == 0) {                    // 如果当前字符的路径不存在
                t[p].next[x] = newNode();               // 创建新节点，并更新next数组
                t[t[p].next[x]].len = t[p].len + 1;     // 设置新节点的len为当前节点len加1
            }
            p = t[p].next[x];  // 移动到下一个节点
        }
        return p;  // 返回最后一个字符对应的节点索引
    }

    void work() {      // 构建Aho-Corasick自动机的后缀链接
        queue<int> q;  // 创建队列，用于广度优先搜索
        q.push(1);     // 将伪根节点加入队列

        while (!q.empty()) {    // 当队列不为空时，进行循环
            int x = q.front();  // 取出队列头部的节点
            q.pop();            // 移除队列头部的节点

            for (int i = 0; i < asz; i++) {                         // 遍历所有可能的字符
                if (t[x].next[i] == 0) {                            // 如果当前节点没有对应字符的转移
                    t[x].next[i] = t[t[x].fail].next[i];            // 设置为后缀链接节点的对应转移
                } else {                                            // 如果有对应字符的转移
                    t[t[x].next[i]].fail = t[t[x].fail].next[i];    // 设置新节点
```

```
                    的后缀链接
                q.push(t[x].next[i]);                    // 将新节点加
                    入队列
            }
        }
    }
}

int next(int p, int x) {  // 获取节点p的字符x的转移
    return t[p].next[x];
}

int fail(int p) {  // 获取节点p的后缀链接
    return t[p].fail;
}

int len(int p) {  // 获取节点p对应的字符串长度
    return t[p].len;
}

int size() {  // 获取Trie树的节点总数
    return t.size();
}
};

void solve() {
    AC ac;
    cin >> n;
    vector<int> id(n + 1);
    for (int i = 1; i <= n; i++) {
        string s;
        cin >> s;
        id[i] = ac.add(s);
    }

    ac.work();
    string tt;
    int p = 1;
    cin >> tt;
    int tot = ac.size();
    vector<int> sz(tot);

    m = tt.size();
    for (int i = 0; i < m; i++) {
        int ch = tt[i] - 'a';
        p = ac.next(p, ch);
        sz[p] += 1;
        deb(p);
    }
    vector<vector<int>> e(tot);
    for (int i = 2; i < tot; i++) {
        deb(i, ac.fail(i));
        e[ac.fail(i)].push_back(i);
    }
    auto dfs = [&](auto self, int u) -> void {
        for (auto v : e[u]) {
            self(self, v);
            sz[u] += sz[v];
        }
    };
    dfs(dfs, 1);
    for (int i = 1; i <= n; i++) cout << sz[id[i]] << endl;
}
```

6.2 EXKMP

```
vector<int> exkmp(string s){
  int len=s.size();
  s="␣"+s;
  vector<int>z(len+1);

  z[1]=0;
  int l=1,r=0;
  for(int i=2;i<=len;i++){
    if(i>r)z[i]=0;
    else {//利用之前的信息
      int k=i-l+1;
      z[i]=min(z[k],r-i+1);
    }
    while(i+z[i]<=len&&s[z[i]+1]==s[i+z[i]])z[i]++;
    if(i+z[i]-1>r){
      l=i;r=i+z[i]-1;
    }
  }
  return z;
}
```

6.3 Hash

```
struct Hash {
    static int findprime() {
        random_device rd;
        mt19937 gen(rd());

        int n = gen() % 900000000 + 100000000;
        if (n % 2 == 0)
            n++;
        while (true) {
            bool ok = 1;
            for (int i = 3; i * i <= n; i += 2) {
                if (n % i == 0) {
                    ok = 0;
                    n += 2;
                    break;
                }
            }
            if (ok)
                return n;
        }
    }

    static const int Mod;
    static vector<int> pow1;
    static vector<int> pow2;
    const int B1 = 131;
    const int B2 = 13331;
    string s;
    int len = 0;
    vector<int> f1, f2;
    using LL = long long;

    Hash() {}
    Hash(const string &t, bool rfg = 0) {
        init(t, rfg);
    }
    // 默认前缀哈希
    void init(const string &t, bool rfg = 0) {
        s = "␣" + t;
        len = t.size();
        int cur = pow1.size();
        if (cur - 1 <= len) {
            pow1.resize(len + 1, 1);
            pow2.resize(len + 1, 1);
            for (int i = cur; i <= len; i++) {
                pow1[i] = (LL)pow1[i - 1] * B1 % Mod;
                pow2[i] = (LL)pow2[i - 1] * B2 % Mod;
            }
        }
        f1.resize(len + 2, 0);
        f2.resize(len + 2, 0);
        if (rfg == 0)
            insert1(s);
        else
            insert2(s);
    }

    // 1-base
    pair<int, int> getpre(int l, int r) const {
        int res1 = (f1[r] - (LL)f1[l - 1] * pow1[r - l + 1] % Mod + Mod) % Mod;
        int res2 = (f2[r] - (LL)f2[l - 1] * pow2[r - l + 1] % Mod + Mod) % Mod;
        return make_pair(res1, res2);
    }
    pair<int, int> getsuf(int l, int r) const {
        int res1 = (f1[l] - (LL)f1[r + 1] * pow1[r - l + 1] % Mod + Mod) % Mod;
        int res2 = (f2[l] - (LL)f2[r + 1] * pow2[r - l + 1] % Mod + Mod) % Mod;
        return make_pair(res1, res2);
    }
    // 前缀哈希
    void insert1(const string &t) {
        for (int i = 1; i <= len; i++) {
            f1[i] = ((LL)f1[i - 1] * B1 + t[i]) % Mod;
            f2[i] = ((LL)f2[i - 1] * B2 + t[i]) % Mod;
        }
    }
    // 后缀哈希
    void insert2(const string &t) {
        for (int i = len; i >= 1; i--) {
            f1[i] = ((LL)f1[i + 1] * B1 + t[i]) % Mod;
            f2[i] = ((LL)f2[i + 1] * B2 + t[i]) % Mod;
        }
    }

    void clear() {
        f1.resize(1);
        f2.resize(1);
```

```
    }
};
const int Hash::Mod = Hash::findprime();
vector<int> Hash::pow1(1, 1);
vector<int> Hash::pow2(1, 1);
```

6.4 KMP

```
struct KMP
{
    vector<int> nxt;
    string tt;
    int len;
    KMP() {}
    KMP(string t)
    {
        len = t.size();
        t = "␣" + t;
        tt = t;
        nxt.resize(len + 1);
        nxt[1] = nxt[0] = 0;
        init(tt);
    }

    void init(string t)
    {
        for (int i = 2; i <= len; i++)
        {
            nxt[i] = nxt[i - 1];
            while (nxt[i] && t[i] != t[nxt[i] + 1])nxt[i] = nxt[nxt[i]];
            nxt[i] += (t[i] == t[nxt[i] + 1]);
        }
    }
    vector<int> getnxt()
    {
        return nxt;
    }
    vector<int> match(string &s, bool oneonly = 0)
    {
        int lens = s.size();
        s = "␣" + s;
        vector<int> stpos;
        int j = 0;
        for (int i = 1; i <= lens; i++)
        {
            while (j == len || (j && s[i] != tt[j + 1])) j = nxt[j];
            if (s[i] == tt[j + 1])j++;
            if (j == len)stpos.push_back(i - len + 1);
        }
        return stpos;
    }
};
```

6.5 MINSHOW

```
string getmin(string s){
  int len=s.size();
  s+=s;
  s="␣"+s;
  int i=1,j=2;//i,j表示以其位置开头的循环串
  while(j<=len){
    int k=0;//时间复杂度线性
    while(k<len&&s[i+k]==s[j+k])k++;
    if(s[i+k]>s[j+k]){
      i+=k+1;
    }
    else j+=k+1;
    if(i==j)j++;
    if(i>j)swap(i,j);
  }
    //最终字典序最小的是以i开头的
  return s.substr(i,len);
}
```

6.6 Manacher

```
struct PAS {
    string s = "#";
    int len = 1;
    vector<int> p;
   // vector<pair<int, int>> all;
    PAS() {}
    PAS(string t) {
```

```
        for (auto c : t) {
            s += c;
            s += '#';
            len += 2;
        }
        s = "␣" + s;
        p.resize(len + 1);
        getp(s);
    }
    vector<int> getp(string t) {
        int mid = 0, r = 0;
        for (int i = 1; i <= len; i++) {
            if (i > r)
                p[i] = 1;
            else
                p[i] = min(p[2 * mid - i], r - i + 1);
            while (i - p[i] > 0 && i + p[i] <= len && t[i - p[i]] == t[i + p[i
              ]]) {
                p[i] += 1;
                // int ql, qr;
                // if ((i - p[i] + 1) % 2 == 0)
                //     ql = (i - p[i] + 1) / 2;
                // else
                //     ql = (i - p[i] + 2) / 2;
                // if ((i + p[i] - 1) % 2 == 0)
                //     qr = (i + p[i] - 1) / 2;
                // else
                //     qr = (i + p[i] - 2) / 2;
                // all.emplace_back(ql, qr);
            }
            if (i + p[i] - 1 > r)
                mid = i, r = i + p[i] - 1;
        }
        return p;
    }
    int getmax() {
        int ans = 0;
        for (int i = 1; i <= len; i++) {
            ans = max(ans, p[i]);
        }
        return (ans - 1);
    }
};
```

6.7 PAM

```
struct PAM {
    static constexpr int asz = 28;
    struct Node {
        int len;
        int fail;
        int dep;      // 以这个节点结尾的回文子串的数量(回文fail树的深度)
        int cnt = 0;  // 同样的回文结构出现次数
        array<int, asz> next;
        // int mask = 0;用了多少种字母
        Node() : len{}, fail{}, dep{}, next{} {}
    };
    vector<Node> t;
    vector<int> idpos;  // idpos表示字符串字符位置到后缀自动机节点编号
    int last;
    string s;
    PAM() {
        init();
    }
    void init() {
        t.assign(2, Node());
        t[0].len = -1;  // 0: 奇根
        last = 1;       // 1: 偶根
        s.clear();
        idpos.assign(1, 0);
    }
    int newNode() {
        t.emplace_back();  // Node()
        return t.size() - 1;
    }

    bool add(char c, char offset = 'a') {
        int pos = s.size();
        s += c;
        int ch = c - offset;
        int cur = last, curlen = 0;

        while (true) {
            curlen = t[cur].len;
            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])
                break;
            cur = t[cur].fail;
        }  // 找到在哪个节点后面建新点
```

```
        if (t[cur].next[ch]) {
            last = t[cur].next[ch];
            idpos.push_back(last);
            t[last].cnt += 1;
            return false;
        }

        int num = newNode();
        last = num;
        idpos.push_back(last);
        t[num].len = t[cur].len + 2;
        // 在这里加入题目需要维护的值
        // t[num].mask = t[cur].mask;
        // t[num].mask |= 1 << ch;
        t[cur].next[ch] = num;

        if (t[num].len == 1) {  // 如果为单字符，指向偶根
            t[num].fail = 1;
            t[num].dep = 1;
            t[num].cnt = 1;
            return true;
        }

        while (true) {  // 为新节点找fail，从父亲的fail开始找
            cur = t[cur].fail;
            curlen = t[cur].len;
            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
                t[num].fail = t[cur].next[ch];
                break;
            }
        }
        t[num].cnt = 1;
        t[num].dep = 1 + t[t[num].fail].dep;

        return true;
    }
    int tot = 0;
    void work(string tt) {
        for (auto x : tt) add(x);
        tot = t.size() - 1;
        for (int i = tot; i >= 0; i--) {
            int fa = t[i].fail;
            t[fa].cnt += t[i].cnt;
        }
    }
    int fail(int p) {
        return t[p].fail;
    }
    int len(int p) {
        return t[p].len;
    }
    int size() {
        return t.size();
    }
    int cnt(int p) {
        return t[p].cnt;
    }
};
```

6.8 SA

```
struct SA {
    int n;                    // 存储字符串的长度
    vector<int> sa, rk, lc;  // sa: 后缀数组, rk: 排名数组, lc: 最长公共前缀数组
        (LCP)

    SA(string &s) {
        n = s.length();    // 初始化字符串的长度
        sa.resize(n + 1);  // 调整 sa 的大小为 n + 1
        lc.resize(n + 1);  // 调整 lc 的大小为 n
        rk.resize(n + 1);  // 调整 rk 的大小为 n + 1
        s = "␣" + s;
        iota(sa.begin(), sa.end(), 0);  // 初始化 sa 为 [1, 2, ..., n]
        sort(sa.begin() + 1, sa.end(), [&](int a, int b) {
            return s[a] < s[b];  // 按照首字符对索引进行排序
        });

        // 初始化 rk 数组
        rk[sa[1]] = 1;
        for (int i = 2; i <= n; ++i)
            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);

        int k = 1;                    // 初始化 k 为 1，表示当前使用的字符串长度
        vector<int> tmp, cnt(n + 1);  // tmp: 临时数组, cnt: 计数排序的频率数组
        tmp.reserve(n + 1);           // 为 tmp 预留 n + 1 个元素的空间
```

```
    while (rk[sa[n]] < n) {  // 当排名最高的后缀排名小于 n时继续循环
        tmp.clear();
        tmp.push_back(0);  // 清空 tmp 数组

        for (int i = 1; i <= k; ++i)
            tmp.push_back(n - k + i);  // 越界部分默认为空字符

        for (auto i : sa)
            if (i >= k + 1)
                tmp.push_back(i - k);  // 按第二关键字排序

        fill(cnt.begin(), cnt.end(), 0);  // 清空 cnt 数组
        for (int i = 1; i <= n; ++i)
            ++cnt[rk[i]];  // 统计每个排名出现的频率

        for (int i = 1; i <= n; ++i)
            cnt[i] += cnt[i - 1];  // 计算计数排序中的前缀和

        for (int i = n; i >= 1; --i) {
            int tmprk = cnt[rk[tmp[i]]];
            sa[tmprk] = tmp[i];
            cnt[rk[tmp[i]]] -= 1;
        }  // 根据 tmp 中的排名重建后缀数组

        std::swap(rk, tmp);  // tmp的功能变为之前的rk桶数组
        rk[sa[1]] = 1;       // 重新初始化排名数组，首先将 sa[1] 的排名设为
          1

        for (int i = 2; i <= n; ++i)
            rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] ||
                                         sa[i - 1] + k > n || tmp[sa[i - 1]
                                           + k] < tmp[sa[i] + k]);  // 基于
                                           前后部分进行比较
        k *= 2;

          // 将 k 翻倍，以便在下一个循环中比较更长的前缀
    }

    for (int i = 1, j = 0; i <= n; ++i) {
        if (rk[i] == 1) {  // 如果当前后缀是字典序最小的，不需要计算 LCP
            j = 0;
        } else {
            for (j -= j > 0; i + j <= n && sa[rk[i] - 1] + j <= n &&
                             s[i + j] == s[sa[rk[i] - 1] + j];)
                ++j;         // 计算与前一个后缀的最长公共前缀长度
            lc[rk[i]] = j;  // 排名为 i 的后缀与排名为 i-1 的 LCP
        }
    }
};
```

6.9 SAM

```
struct SAM {
    static constexpr int asz = 26;
    struct Node {
        int len;
        int fail;
        int cnt = 0;
        array<int, asz> next;
        Node() : len{}, fail{}, next{} {}
    };
    vector<Node> t;
    int tot = 0;
    SAM() {
        init();
    }
    void init() {
        t.assign(2, Node());
        t[0].next.fill(1);
        t[0].len = -1;
    }
    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }
    int extend(int p, int c) {
        if (t[p].next[c]) {
            int q = t[p].next[c];
            if (t[q].len == t[p].len + 1) {
                return q;
            }
            int nq = newNode();
            t[nq].len = t[p].len + 1;
            t[nq].fail = t[q].fail;
            t[nq].next = t[q].next;
```

```
            t[q].fail = nq;
            while (t[p].next[c] == q) {
                t[p].next[c] = nq;
                p = t[p].fail;
            }
            return nq;
        }
        int np = newNode();
        t[np].len = t[p].len + 1;
        while (!t[p].next[c]) {
            t[p].next[c] = np;
            p = t[p].fail;
        }
        t[np].fail = extend(p, c);
        t[np].cnt += 1;
        return np;
    }
    int extend(int p, char c, char offset = 'a') {
        return extend(p, c - offset);
    }

    int next(int p, int x) {
        return t[p].next[x];
    }

    int next(int p, char c, char offset = 'a') {
        return next(p, c - 'a');
    }

    int fail(int p) {
        return t[p].fail;
    }

    int len(int p) {
        return t[p].len;
    }
    int size() {
        return t.size();
    }
    int &cnt(int p) {
        return t[p].cnt;
    }
    void work(string s) {
        int p = 1;
        // vector<int> pos(1, 0);
        for (auto x : s) {
            p = extend(p, x);
            // pos.push_back(p);
        }
        tot = t.size() - 1;
        // return pos;
    }
    void getcnt(int len) {
        vector<int> tong(len + 1);
        vector<int> id(tot + 1);
        for (int i = 1; i <= tot; i++) tong[t[i].len]++;
        // 按照len[x]从小到大基数排序，相当于对SAM图进行拓扑排序
        for (int i = 1; i <= n; i++) tong[i] += tong[i - 1];
        for (int i = 1; i <= tot; i++) id[tong[t[i].len]--] = i;  // 排名为j的节
          点是状态i
        for (int i = tot; i >= 1; i--) {
            auto cur = t[id[i]];
            t[cur.fail].cnt += cur.cnt;
        }
        // 从后往前for,自底向上更新parent的right大小
    }
};
```

6.10 Trie_01

```
struct Trie_bin {  // 保证第一次一定是先插入，查询不能先做
    static constexpr int ALPHA = 2;
    static constexpr int width = 21;  // 值域必须小于2的width次方
    struct Node {
        int cnt = 0;
        array<int, ALPHA> next;
        Node() : next{} {}
    };
    vector<Node> t;
    Trie_bin() { init(); }
    void init() {
        t.assign(2, {});
    }
    int newNode() {
        t.emplace_back();
        return t.size() - 1;
```

```
    }
    // 增加flag标志便于删除
    int add(const vector<int> &a, int flag) {
        int p = 1;
        for (auto x : a) {
            if (t[p].next[x] == 0) {
                t[p].next[x] = newNode();
            }
            p = t[p].next[x];
            t[p].cnt += flag;
        }
        return p;
    }
    // 数字转01串vector
    int add(int x, int flag = 1) {  // x必须小于2的width次方
        vector<int> a;
        for (int i = width - 1; i >= 0; i--) {
            a.push_back((x >> i) & 1);
        }
        return add(a, flag);
    }
    int querymx(int x) {
        int res = 0, p = 1;
        for (int i = width - 1; i >= 0; i--) {
            int u = (x >> i) & 1;
            int nxp = t[p].next[u ^ 1];
            if (nxp && t[nxp].cnt) {
                res |= 1 << i;
                u ^= 1;
            }
            p = t[p].next[u];
        }
        return res;
    }
    int next(int p, int x) { return t[p].next[x]; }
    int size() { return t.size(); }
    int cnt(int p) {
        return t[p].cnt;
    }
};
Trie_bin tr;
```

6.11 Trie_per

```
struct Trie_per {
    static constexpr int SIZE = 2;
    static constexpr int width = 24;  // 值域小于2的width次方
    struct Node {
        int cnt;
        array<int, SIZE> next;
        Node() : cnt{0}, next{} {}
    };
    vector<Node> t;
    vector<int> ver;
    Trie_per() { init(); }
    void init() {
        t.assign(2, {});
        ver.resize(1);
    }
    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }
    int add(int pre, const vector<int> &a) {
        int cur = newNode();
        int p = pre, q = cur;
        t[q] = t[p];
        for (auto x : a) {
            t[q].next[x] = newNode();
            p = next(p, x);
            q = next(q, x);
            t[q] = t[p];
            t[q].cnt++;
        }
        return cur;
    }
    int add(int pre, int x) {  // 转成01vector
        vector<int> a;
        for (int i = width - 1; i >= 0; i--) {
            a.push_back((x >> i) & 1);
        }
        return add(pre, a);
    }
    void add(int x) {  // 外部接口，加入一个数生成一个新版本
        int pos = add(ver.back(), x);
        ver.push_back(pos);
```

```
    }
    // 查询x在版本(l,r]中和哪个数异或最大
    int querymx(int l, int r, int x) {  // 传l-1进来
        int res = 0;
        int p = ver[l], q = ver[r];
        for (int i = width - 1; i >= 0; i--) {
            int u = (x >> i) & 1;
            int nxp = t[p].next[u ^ 1], nxq = t[q].next[u ^ 1];
            if (t[nxq].cnt - t[nxp].cnt > 0) {
                res |= 1 << i;
                u ^= 1;
            }
            p = t[p].next[u];
            q = t[q].next[u];
        }
        return res;
    }
    int size() { return t.size(); }
    int cnt(int p) { return t[p].cnt; }
    int next(int p, int x) { return t[p].next[x]; }
};
```

6.12 Trie_string

```
int id(char c) {  // 给出现的字符集编码，记得改offset部分
    if (c >= 'a' && c <= 'z')
        return c - 'a';
    else if (c >= 'A' && c <= 'Z')
        return c - 'A' + 26;
    else
        return c - '0' + 52;
}
struct Trie {  // 正常字母字符串trie
    static constexpr int ALPHA = 26;
    struct Node {
        int cnt;
        bool ended;
        array<int, ALPHA> next;
        Node() : cnt{0}, ended{false}, next{} {}
    };
    vector<Node> t;
    Trie() { init(); }
    void init() {
        t.assign(2, {});
    }
    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }
    int add(const vector<int> &a) {
        int p = 1;
        for (auto x : a) {
            if (t[p].next[x] == 0) {
                t[p].next[x] = newNode();
            }
            p = t[p].next[x];
            t[p].cnt++;
        }
        t[p].ended = true;
        return p;
    }
    int add(const string &s, char offset = 'a') {
        vector<int> a;
        for (auto c : s) {
            a.push_back(c - offset);
        }
        return add(a);
    }
    int cnt(int p) { return t[p].cnt; }
    bool ended(int p) { return t[p].ended; }
    int next(int p, int x) { return t[p].next[x]; }
    int next(int p, char c, char offset = 'a') { return next(p, c - offset); }
    int size() { return t.size(); }
};
Trie tr;
```

7 geom

7.1 dls

```
typedef double db;
const db EPS = 1e-9;
//由于硬件限制，浮点数运算有误差，eps用来消除误差
inline int sign(db a) { return a < -EPS ? -1 : a > EPS; }
```

```
//判断数符号，负数返回-1，0返回0，正数返回1
inline int cmp(db a, db b) { return sign(a - b); }
//比较两数大小
//点类，向量类
//因为有许多操作相似，所以并在一起
struct P
{
    db x, y;
    //点表示坐标，向量表示向量
    P() {}
    P(db _x, db _y) : x(_x), y(_y) {}
    //构造函数
    P operator+(P p) { return {x + p.x, y + p.y}; }
    P operator-(P p) { return {x - p.x, y - p.y}; }
    P operator*(db d) { return {x * d, y * d}; }
    P operator/(db d) { return {x / d, y / d}; }
    //向量加减乘除
    bool operator<(P p) const
    {
        int c = cmp(x, p.x);
        if (c)
            return c == -1;
        return cmp(y, p.y) == -1;
    }
    bool operator==(P o) const
    {
        return cmp(x, o.x) == 0 && cmp(y, o.y) == 0;
    }
    //比较字典序
    db dot(P p) { return x * p.x + y * p.y; }
    //点积
    db det(P p) { return x * p.y - y * p.x; }
    //叉积
    db distTo(P p) { return (*this - p).abs(); }
    //点距离
    db alpha() { return atan2(y, x); }
    void read() { cin >> x >> y; }
    void write() { cout << "(" << x << "," << y << ")" << endl; }
    db abs() { return sqrt(abs2()); }
    db abs2() { return x * x + y * y; }
    P rot90() { return P(-y, x); }
    P unit() { return *this / abs(); }
    int quad() const { return sign(y) == 1 || (sign(y) == 0 && sign(x) >= 0); }
    //判断点在极角坐标系上半边还是下半边，极点和极轴也算上半边
    P rot(db an) { return {x * cos(an) - y * sin(an), x * sin(an) + y * cos(an)
      }; }
    //向量旋转
};
//线类，半平面类
struct L
{ // ps[0] -> ps[1]
    P ps[2];
    P &operator[](int i) { return ps[i]; }
    P dir() { return ps[1] - ps[0]; }
    L(P a, P b)
    {
        ps[0] = a;
        ps[1] = b;
    }
    bool include(P p) { return sign((ps[1] - ps[0]).det(p - ps[0])) > 0; }
    L push()
    { // push eps outward
        const double eps = 1e-8;
        P delta = (ps[1] - ps[0]).rot90().unit() * eps;
        return {ps[0] + delta, ps[1] + delta};
    }
};

#define cross(p1, p2, p3) ((p2.x - p1.x) * (p3.y - p1.y) - (p3.x - p1.x) * (p2.y
  - p1.y))
#define crossOp(p1, p2, p3) sign(cross(p1, p2, p3))
//叉积，可以用来求三角形面积（输入参数是三个点）
bool chkLL(P p1, P p2, P q1, P q2)
{
    db a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
    return sign(a1 + a2) != 0;
}
//判断向量平行
P isLL(P p1, P p2, P q1, P q2)
{
    db a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
    return (p1 * a2 + p2 * a1) / (a1 + a2);
}
P isLL(L l1, L l2) { return isLL(l1[0], l1[1], l2[0], l2[1]); }
//求直线交点
bool intersect(db l1, db r1, db l2, db r2)
{
    if (l1 > r1)
```

```
        swap(l1, r1);
    if (l2 > r2)
        swap(l2, r2);
    return !(cmp(r1, l2) == -1 || cmp(r2, l1) == -1);
}
bool isSS(P p1, P p2, P q1, P q2)
{
    return intersect(p1.x, p2.x, q1.x, q2.x) && intersect(p1.y, p2.y, q1.y, q2.y
      ) &&
          crossOp(p1, p2, q1) * crossOp(p1, p2, q2) <= 0 && crossOp(q1, q2, p1)
             * crossOp(q1, q2, p2) <= 0;
}

bool isSS_strict(P p1, P p2, P q1, P q2)
{
    return crossOp(p1, p2, q1) * crossOp(p1, p2, q2) < 0 && crossOp(q1, q2, p1)
      * crossOp(q1, q2, p2) < 0;
}
//判断线段相交，交在端点算不算分为严格不严格
bool isMiddle(db a, db m, db b)
{
    return sign(a - m) == 0 || sign(b - m) == 0 || (a < m != b < m);
}
bool isMiddle(P a, P m, P b)
{
    return isMiddle(a.x, m.x, b.x) && isMiddle(a.y, m.y, b.y);
}
bool onSeg(P p1, P p2, P q)
{
    return crossOp(p1, p2, q) == 0 && isMiddle(p1, q, p2);
}
bool onSeg_strict(P p1, P p2, P q)
{
    return crossOp(p1, p2, q) == 0 && sign((q - p1).dot(p1 - p2)) * sign((q - p2
      ).dot(p1 - p2)) < 0;
}
//点在线段上判定
P proj(P p1, P p2, P q)
{
    P dir = p2 - p1;
    return p1 + dir * (dir.dot(q - p1) / dir.abs2());
}
P reflect(P p1, P p2, P q)
{
    return proj(p1, p2, q) * 2 - q;
}
db nearest(P p1, P p2, P q)
{
    if (p1 == p2)
        return p1.distTo(q);
    P h = proj(p1, p2, q);
    if (isMiddle(p1, h, p2))
        return q.distTo(h);
    return min(p1.distTo(q), p2.distTo(q));
}
//投影，反射，最近点
//最近点是线段外一点到线段上的点的最短距离
db disSS(P p1, P p2, P q1, P q2)
{
    if (isSS(p1, p2, q1, q2))
        return 0;
    return min(min(nearest(p1, p2, q1), nearest(p1, p2, q2)), min(nearest(q1, q2
      , p1), nearest(q1, q2, p2)));
}
//线段距离
db rad(P p1, P p2)
{
    return atan2l(p1.det(p2), p1.dot(p2));
}

db incircle(P p1, P p2, P p3)
{
    db A = p1.distTo(p2);
    db B = p2.distTo(p3);
    db C = p3.distTo(p1);
    return sqrtl(A * B * C / (A + B + C));
}

// polygon
//简单多边形的问题只有判断点在多边形内，和多边形面积简单，其他只做凸多边形
db area(vector<P> ps)
{
    db ret = 0;
    for(int i=0;i<ps.size();++i)
        ret += ps[i].det(ps[(i + 1) % ps.size()]);
    return ret / 2;
}
//多边形面积
```

```
int contain(vector<P> ps, P p)
{ // 2:inside,1:on_seg,0:outside
    int n = ps.size(), ret = 0;
    rep(i, 0, n)
    {
        P u = ps[i], v = ps[(i + 1) % n];
        if (onSeg(u, v, p))
            return 1;
        if (cmp(u.y, v.y) <= 0)
            swap(u, v);
        if (cmp(p.y, u.y) > 0 || cmp(p.y, v.y) <= 0)
            continue;
        ret ^= crossOp(p, u, v) > 0;
    }
    return ret * 2;
}
//判断点在多边形内
vector<P> convexHull(vector<P> ps)
{
    int n = ps.size();
    if (n <= 1)
        return ps;
    sort(ps.begin(), ps.end());
    vector<P> qs(n * 2);
    int k = 0;
    for (int i = 0; i < n; qs[k++] = ps[i++])
        while (k > 1 && crossOp(qs[k - 2], qs[k - 1], ps[i]) <= 0)
            --k;
    for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--])
        while (k > t && crossOp(qs[k - 2], qs[k - 1], ps[i]) <= 0)
            --k;
    qs.resize(k - 1);
    return qs;
}
vector<P> convexHullNonStrict(vector<P> ps)
{
    // caution: need to unique the Ps first
    int n = ps.size();
    if (n <= 1)
        return ps;
    sort(ps.begin(), ps.end());
    vector<P> qs(n * 2);
    int k = 0;
    for (int i = 0; i < n; qs[k++] = ps[i++])
        while (k > 1 && crossOp(qs[k - 2], qs[k - 1], ps[i]) < 0)
            --k;
    for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--])
        while (k > t && crossOp(qs[k - 2], qs[k - 1], ps[i]) < 0)
            --k;
    qs.resize(k - 1);
    return qs;
}
//凸包

db convexDiameter(vector<P> ps)
{
    int n = ps.size();
    if (n <= 1)
        return 0;
    int is = 0, js = 0;
    rep(k, 1, n) is = ps[k] < ps[is] ? k : is, js = ps[js] < ps[k] ? k : js;
    int i = is, j = js;
    db ret = ps[i].distTo(ps[j]);
    do
    {
        if ((ps[(i + 1) % n] - ps[i]).det(ps[(j + 1) % n] - ps[j]) >= 0)
            (++j) %= n;
        else
            (++i) %= n;
        ret = max(ret, ps[i].distTo(ps[j]));
    } while (i != is || j != js);
    return ret;
}
//凸包直径
vector<P> convexCut(const vector<P> &ps, P q1, P q2)
{
    vector<P> qs;
    int n = ps.size();
    rep(i, 0, n)
    {
        P p1 = ps[i], p2 = ps[(i + 1) % n];
        int d1 = crossOp(q1, q2, p1), d2 = crossOp(q1, q2, p2);
        if (d1 >= 0)
            qs.pb(p1);
        if (d1 * d2 < 0)
            qs.pb(isLL(p1, p2, q1, q2));
    }
    return qs;
```

```
}
//直线切割凸包，返回直线左边凸包的点
db min_dist(vector<P> &ps, int l, int r)
{
    if (r - l <= 5)
    {
        db ret = 1e18;
        for(int i=l;i<r;++i)
            for(int j=l;j<i;++j)
                ret = min(ret, ps[i].distTo(ps[j]));
        return ret;
    }
    int m = (l + r) >> 1;
    db ret = min(min_dist(ps, l, m), min_dist(ps, m, r));
    vector<P> qs;
    for(int i=l;i<r;++i)
        if (abs(ps[i].x - ps[m].x) <= ret)
            qs.push_back(ps[i]);
    sort(qs.begin(), qs.end(), [](P a, P b) -> bool
         { return a.y < b.y; });
    for(int i=1;i<qs.size();++i)
        for (int j = i - 1; j >= 0 && qs[j].y >= qs[i].y - ret; --j)
            ret = min(ret, qs[i].distTo(qs[j]));
    return ret;
}
//平面最近点对,[l,r)，要求ps按x升序
int type(P o1, db r1, P o2, db r2)//圆与圆的位置关系
{
    db d = o1.distTo(o2);
    if (cmp(d, r1 + r2) == 1)
        return 4;
    if (cmp(d, r1 + r2) == 0)
        return 3;
    if (cmp(d, abs(r1 - r2)) == 1)
        return 2;
    if (cmp(d, abs(r1 - r2)) == 0)
        return 1;
    return 0;
}

vector<P> isCL(P o, db r, P p1, P p2)
{
    if (cmp(abs((o - p1).det(p2 - p1) / p1.distTo(p2)), r) > 0)
        return {};
    db x = (p1 - o).dot(p2 - p1), y = (p2 - p1).abs2(), d = x * x - y * ((p1 - o
      ).abs2() - r * r);
    d = max(d, (db)0.0);
    P m = p1 - (p2 - p1) * (x / y), dr = (p2 - p1) * (sqrt(d) / y);
    return {m - dr, m + dr}; // along dir: p1->p2
}

vector<P> isCC(P o1, db r1, P o2, db r2)
{ // need to check whether two circles are the same
    db d = o1.distTo(o2);
    if (cmp(d, r1 + r2) == 1)
        return {};
    if (cmp(d, abs(r1 - r2)) == -1)
        return {};
    d = min(d, r1 + r2);
    db y = (r1 * r1 + d * d - r2 * r2) / (2 * d), x = sqrt(r1 * r1 - y * y);
    P dr = (o2 - o1).unit();
    P q1 = o1 + dr * y, q2 = dr.rot90() * x;
    return {q1 - q2, q1 + q2}; // along circle 1
}

vector<P> tanCP(P o, db r, P p)
{
    db x = (p - o).abs2(), d = x - r * r;
    if (sign(d) <= 0)
        return {}; // on circle => no tangent
    P q1 = o + (p - o) * (r * r / x);
    P q2 = (p - o).rot90() * (r * sqrt(d) / x);
    return {q1 - q2, q1 + q2}; // counter clock-wise
}

vector<L> extanCC(P o1, db r1, P o2, db r2)
{
    vector<L> ret;
    if (cmp(r1, r2) == 0)
    {
        P dr = (o2 - o1).unit().rot90() * r1;
        ret.pb(L(o1 + dr, o2 + dr)), ret.pb(L(o1 - dr, o2 - dr));
    }
    else
    {
        P p = (o2 * r1 - o1 * r2) / (r1 - r2);
        vector<P> ps = tanCP(o1, r1, p), qs = tanCP(o2, r2, p);
        rep(i, 0, min(ps.size(), qs.size())) ret.pb(L(ps[i], qs[i])); // c1
```

```
        counter-clock wise
    }
    return ret;
}

vector<L> intanCC(P o1, db r1, P o2, db r2)
{
    vector<L> ret;
    P p = (o1 * r2 + o2 * r1) / (r1 + r2);
    vector<P> ps = tanCP(o1, r1, p), qs = tanCP(o2, r2, p);
    rep(i, 0, min(ps.size(), qs.size())) ret.pb(L(ps[i], qs[i])); // c1 counter-
      clock wise
    return ret;
}

db areaCT(db r, P p1, P p2)
{
    vector<P> is = isCL(P(0, 0), r, p1, p2);
    if (is.empty())
        return r * r * rad(p1, p2) / 2;
    bool b1 = cmp(p1.abs2(), r * r) == 1, b2 = cmp(p2.abs2(), r * r) == 1;
    if (b1 && b2)
    {
        if (sign((p1 - is[0]).dot(p2 - is[0])) <= 0 &&
            sign((p1 - is[0]).dot(p2 - is[0])) <= 0)
            return r * r * (rad(p1, is[0]) + rad(is[1], p2)) / 2 + is[0].det(is
              [1]) / 2;
        else
            return r * r * rad(p1, p2) / 2;
    }
    if (b1)
        return (r * r * rad(p1, is[0]) + is[0].det(p2)) / 2;
    if (b2)
        return (p1.det(is[1]) + r * r * rad(is[1], p2)) / 2;
    return p1.det(p2) / 2;
}

bool parallel(L l0, L l1) { return sign(l0.dir().det(l1.dir())) == 0; }
bool cmp(P a, P b)
{
    if (a.quad() != b.quad())
    {
        return a.quad() < b.quad();
    }
    else
    {
        return sign(a.det(b)) > 0;
    }
}
//极角排序
bool sameDir(L l0, L l1) { return parallel(l0, l1) && sign(l0.dir().dot(l1.dir()
  )) == 1; }
bool operator<(L l0, L l1)
{
    if (sameDir(l0, l1))
    {
        return l1.include(l0[0]);
    }
    else
    {
        return cmp(l0.dir(), l1.dir());
    }
}
bool check(L u, L v, L w)
{
    return w.include(isLL(u, v));
}
vector<P> halfPlaneIS(vector<L> &l)
{
    sort(l.begin(), l.end());
    deque<L> q;
    for (int i = 0; i < (int)l.size(); ++i)
    {
        if (i && sameDir(l[i], l[i - 1]))
            continue;
        while (q.size() > 1 && !check(q[q.size() - 2], q[q.size() - 1], l[i]))
            q.pop_back();
        while (q.size() > 1 && !check(q[1], q[0], l[i]))
            q.pop_front();
        q.push_back(l[i]);
    }
    while (q.size() > 2 && !check(q[q.size() - 2], q[q.size() - 1], q[0]))
        q.pop_back();
    while (q.size() > 2 && !check(q[1], q[0], q[q.size() - 1]))
        q.pop_front();
    vector<P> ret;
    for (int i = 0; i < (int)q.size(); ++i)
        ret.push_back(isLL(q[i], q[(i + 1) % q.size()]));
```

```
    return ret;
}
//半平面交
P inCenter(P A, P B, P C)
{
    double a = (B - C).abs(), b = (C - A).abs(), c = (A - B).abs();
    return (A * a + B * b + C * c) / (a + b + c);
}
//内心，角平分线的交点
P circumCenter(P a, P b, P c)
{
    P bb = b - a, cc = c - a;
    double db = bb.abs2(), dc = cc.abs2(), d = 2 * bb.det(cc);
    return a - P(bb.y * dc - cc.y * db, cc.x * db - bb.x * dc) / d;
}
//外心，垂直平分线的交点
P orthoCenter(P a, P b, P c)
{
    P ba = b - a, ca = c - a, bc = b - c;
    double Y = ba.y * ca.y * bc.y,
           A = ca.x * ba.y - ba.x * ca.y,
           x0 = (Y + ca.x * ba.y * b.x - ba.x * ca.y * c.x) / A,
           y0 = -ba.x * (x0 - c.x) / ba.y + ca.y;
    return {x0, y0};
}
//垂心，垂线的交点
```

7.2 平面最近点对 __ 分治

```
#include <bits/stdc++.h>
#ifdef LOCAL
#include "debug.h"
#else
#define deb(...)
#endif
using namespace std;
#define ll long long
// #define int long long
#define ull unsigned long long
#define pii pair<int, int>
#define db long double
#define baoliu(x, y) cout << fixed << setprecision(y) << x
#define endl "\n"
#define alls(x) (x).begin(), (x).end()
#define fs first
#define sec second
#define bug(x) cerr << #x << " = " << x << endl
const int N = 2e6 + 10;
const int M = 1e6 + 10;
const int inf = 0x3f3f3f3f;
const int mod = 998244353;
const double eps = 1e-8;
const double PI = acos(-1.0);
int n;
struct node {
    db x, y;
    bool operator<(const node &A) const {
        return x < A.x;
    }
} a[N], c[N];
db dis(node c, node d) {
    db c1 = c.x - d.x, c2 = c.y - d.y;
    return sqrt(c1 * c1 + c2 * c2);
}
db cal(int l, int r) {
    if (l == r)
        return 1e12;
    int cnt = 0;
    int mid = (l + r) >> 1;
    db d = min(cal(l, mid), cal(mid + 1, r));
    for (int i = l; i <= r; i++) {
        if (fabs(a[i].x - a[mid].x) < d) {
            c[++cnt].y = a[i].x;
            c[cnt].x = a[i].y;
        }
    }
    sort(c + 1, c + 1 + cnt);
    for (int i = 1; i <= cnt; i++) {
        for (int j = i + 1; j <= cnt && fabs(c[j].y - c[i].y) < d; j++) {
            d = min(d, dis(c[i], c[j]));
        }
    }
    return d;
}
void solve() {
    cin >> n;
```

```
    for (int i = 1; i <= n; i++) cin >> a[i].x >> a[i].y;
    sort(a + 1, a + 1 + n);
    db ans = cal(1, n);
    baoliu(ans, 12);
}

signed main() {
    cin.tie(0);
    ios::sync_with_stdio(false);
#ifdef LOCAL
    double starttime = clock();
    // freopen("in.txt", "r", stdin);
    // freopen("out.txt", "w", stdout);
#endif
    int t = 1;
    // cin >> t;
    while (t--) solve();
#ifdef LOCAL
    double endtime = clock();
    cerr << "Time Used: " << (double)(endtime - starttime) / CLOCKS_PER_SEC *
      1000 << " ms" << endl;
#endif
    return 0;
}
```

8 other

8.1 Compile_cmd

```
"cpp": "cd $dir
&& g++ $fileName -Wall -Wextra
-fsanitize=undefined -DLOCAL -D_GLIBCXX_DEBUG
-std=c++17 -g -O2 -o $fileNameWithoutExt
&& $dir/$fileNameWithoutExt",
```

8.2 debug

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

using namespace std;

void __print(int x) { cerr << x; }
void __print(long x) { cerr << x; }
void __print(long long x) { cerr << x; }
void __print(unsigned x) { cerr << x; }
void __print(unsigned long x) { cerr << x; }
void __print(unsigned long long x) { cerr << x; }
void __print(float x) { cerr << x; }
void __print(double x) { cerr << x; }
void __print(long double x) { cerr << x; }
void __print(char x) { cerr << '\'' << x << '\''; }
void __print(const char *x) { cerr << '\"' << x << '\"'; }
void __print(const string &x) { cerr << '\"' << x << '\"'; }
void __print(bool x) { cerr << (x ? "true" : "false"); }

template <typename T, typename V>
void __print(const pair<T, V> &x)
{
    cerr << '{';
    __print(x.first);
    cerr << ',';
    __print(x.second);
    cerr << '}';
}
template <typename T>
void __print(const T &x)
{
    int f = 0;
    cerr << '{';
    for (auto &i : x)
        cerr << (f++ ? "," : ""), __print(i);
    cerr << "}";
}
void _print() { cerr << "]\n"; }//没有剩余参数时递归调用
template <typename T, typename... V>
void _print(T t, V... v)
{
    __print(t);
    if (sizeof...(v))
        cerr << ", ";
    _print(v...);
}
#ifndef ONLINE_JUDGE
#define deb(x...)                   \
```

```
    cerr << "[" << #x << "] = [" ; \
    _print(x)
#else
#define deb(x...)
#endif
```

8.3 duipai_linux

```
#!/bin/bash

while ./data > in.txt && ./a < in.txt > out.txt && ./std < in.txt > std.txt &&
  diff out.txt std.txt; do
    echo "no problem meow!"
done
```

8.4 duipai_win

```
#include <bits/stdc++.h>
using namespace std;
using LL = long long;
mt19937 rnd(chrono::system_clock::now().time_since_epoch().count());
mt19937_64 rnd_64(chrono::system_clock::now().time_since_epoch().count());
void gen() {
    ofstream fout("in.txt");
    /// 添加对应的输入——gen文件寻找
    fout.close();
}
int main() {
    system("g++ std.cpp -o std -std=c++20 -O2");
    system("g++ test.cpp -o test -std=c++20 -O2");
    double TL = 5000.0;
    for (int i = 1; i <= 100; i++) {
        printf("iteration: %d\n", i);
        gen();
        system("std.exe < in.txt > ans.txt");
        double begin = clock();
        system("test.exe < in.txt > out.txt");
        double end = clock();
        double t = (double)(begin - end) / CLOCKS_PER_SEC * 1000;
        // cout << "Time Used: " << t << " ms" << endl;
        if (system("fc ans.txt out.txt")) {
            printf("test#%d WA\n", i);
            break;
        } else if (t > TL) {
            printf("test#%d TLE time used  %.0lfms\n", i, t);
            break;
        } else {
            printf("test#%d AC  time used  %.0lfms\n", i, t);
        }
    }
    return 0;
}
```

8.5 gdbcmd

```
-g
-Wall -Wextra
-Wshadow #防止局部变量不小心遮盖其他变量
-Wformat=2 #防止printf/scanf 写错
-Wconversion #防止意外的类型转换
-Wstack-usage=1 #看栈空间使用情况
-fsanitize=undefined #查找未定义行为
-fsanitize=address #查数组越界
-D_GLIBCXX_DEBUG : STL debug mode
-Wl,--stack=1073741824
-fsanitize=undefined fsanitize-undefined-trap-on-error
# Windows
#define deb(x) (void)(cerr << "L" << __LINE__ << ": "<< #x << " = " << (x) <<
  endl)
建议直接使用GDB 的命令行，-g,建议禁用优化。GDB 的常用命令有：
> b (breakpoint) 行号/函数名
> r (run) [< 输入文件名]
> n (next)
> s (step)
> c (continue)
> p (print) 表达式
> d (disp) 表达式
> cond (condition) 断点编号表达式
> bt (backtrace)
> fr (frame) 栈帧编号
> gcov/-ftest-coverage -fprofile-arcs:代码覆盖率检测
可以看代码中每一行被执行的次数
> gprof/-pg:代码剖析,可以看函数执行时间占总时间的百分
```

比
> gprof 输出的是时间，但只能精确到函数
> gcov 精确到行，但只能输出调用次数

8.6 gen_data

```
mt19937 rnd(chrono::system_clock::now().time_since_epoch().count());
mt19937_64 rnd_64(chrono::system_clock::now().time_since_epoch().count());
int rndi(int r) { return rnd() % r; }                          // 随机生成0-(r-1)
int rndi(int l, int r) { return rnd() % (r - l + 1) + l; }     // 随机生成l-r
LL rndll(LL l, LL r) { return rnd_64() % (r - l + 1) + l; }    // 随机生成0-(r-1)
char rndc() { return rndi(-128, 127); }                        // 生成 ASCII 码在
  [-128, 127] 范围内的随机字符
char rndc(const string &s) { return s[rndi(s.length())]; }     // 从给定字符串 s
  中随机选择一个字符
char rnd_lower() { return rndi(26) + 'a'; }                    // 随机小写字母
char rnd_upper() { return rndi(26) + 'A'; }                    // 大写
char rnd_digit() { return rndi(10) + '0'; }                    // 数字
char rnd_alpha() { // 大小写
    int r = rndi(52);
    return r < 26 ? (r + 'a') : (r - 26 + 'A');
}
char rnd_alphadigit() {  // 大小写+数字
    int r = rndi(62);
    if (r < 10)
        return r + '0';
    if (r < 36)
        return r - 10 + 'a';
    return r - 36 + 'A';
}
template <typename T>  // n 个随机值的 vector
vector<T> rnd_vec(int n, const function<T(void)> &f) {
    vector<T> vec;
    while (n--) vec.push_back(f());
    return vec;
}
// n个[l, r] 范围内的随机int
vector<int> rnd_vii(int n, int l, int r) {
    return rnd_vec<int>(n, [=]() { return rndi(l, r); });
}
// n 个在 [l, r] 范围内的随机 long long
vector<LL> rnd_vll(int n, LL l, LL r) {
    return rnd_vec<LL>(n, [=]() { return rndll(l, r); });
}
// 一个长度为 n 的随机字符串。每个字符由函数 f 生成。
string rnds(int n, const function<char(void)> &f) {
    string s;
    while (n--) s += f();
    return s;
}
// cout << rnds(10, []() { return rndc("abc"); }) << endl;
// 生成并输出一个由 10 个从字符串 "abc" 中随机选择的字符组成的字符串。
```

8.7 random_real_prime

```
//随机素数
979345007 986854057502126921
935359631 949054338673679153
931936021 989518940305146613
984974633 972090414870546877
984858209 956380060632801307
static int findprime() {//随机生成质数
    random_device rd;
    mt19937 gen(rd());

    int n = gen() % 900000000 + 100000000;
    if (n % 2 == 0)
        n++;
    while (true) {
        bool ok = 1;
        for (int i = 3; i * i <= n; i += 2) {
            if (n % i == 0) {
                ok = 0;
                n += 2;
                break;
            }
        }
        if (ok)
            return n;
    }
}
//-------------------------------------------------------//
//伪随机数生成
#define u64 unsigned long long
```

```
#define u32 unsigned int
u64 xorshift(u64 x) { x ^= x << 13; x ^= x >> 7; x ^= x << 17; return x; }
u32 xorshift(u32 x) { x ^= x << 13; x ^= x >> 17; x ^= x << 5; return x; }
//真随机
#include <random>
#include <chrono>
mt19937 rnd(chrono::system_clock::now().time_since_epoch().count());
mt19937_64 rnd_64(chrono::system_clock::now().time_since_epoch().count());
```

8.8 template

```
#include <bits/stdc++.h>
#ifdef LOCAL
#include "debug.h"
#else
#define deb(...)
#endif
using namespace std;
#define ll long long
//#define int long long
#define ull unsigned long long
#define pii pair<int, int>
#define db double
#define baoliu(x, y) cout << fixed << setprecision(y) << x
#define endl "\n"
#define alls(x) (x).begin(), (x).end()
#define fs first
#define sec second
#define bug(x) cerr << #x << " = " << x << endl
const int N = 2e5 + 10;
const int M = 1e6 + 10;
const int inf = 0x3f3f3f3f;
const int mod = 998244353;
const double eps = 1e-8;
const double PI = acos(-1.0);
void solve() {

}
signed main() {
    cin.tie(0);
    ios::sync_with_stdio(false);
#ifdef LOCAL
    double starttime = clock();
    // freopen("in.txt", "r", stdin);
    // freopen("out.txt", "w", stdout);
#endif
    int t = 1;
    //cin >> t;
    while (t--) solve();
#ifdef LOCAL
    double endtime=clock();
    cerr << "Time Used: " << (double)(endtime - starttime) / CLOCKS_PER_SEC *
      1000 << " ms" << endl;
#endif
    return 0;
}
```

8.9 template_region

```
#include <bits/stdc++.h>
using namespace std;
#ifdef LOCAL
#define deb(x) (void)(cerr << "L" << __LINE__ << ": " << #x << " = " << (x) <<
  endl)
#else
#define deb(x)
#endif
#define ll long long
// #define int long long
#define baoliu(x, y) cout << fixed << setprecision(y) << x
#define endl "\n"
const int mod = 998244353;
const double eps = 1e-8;
const double PI = acos(-1.0);
void solve() {
    int n;
    cin >> n;
    cout << n << endl;
}
signed main() {
    cin.tie(0);
    ios::sync_with_stdio(false);
#ifdef LOCAL
    double starttime = clock();
```

```
    auto t1 = freopen("in.txt", "r", stdin);
    auto t2 = freopen("out.txt", "w", stdout);
    assert(t1 != nullptr);
    assert(t2 != nullptr);
#endif
    int t = 1;
    // cin >> t;
    while (t--) solve();
#ifdef LOCAL
    double endtime = clock();
    cerr << "Time Used: " << (double)(endtime - starttime) / CLOCKS_PER_SEC *
      1000 << " ms" << endl;
#endif
    return 0;
}
```

8.10 test_g++

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    // GNU C++11: Array
    array<int, 3> C = {1, 2, 3};
    for (int i : C) {
        cout << i << " ";
    }
    cout << endl;
    // GNU C++14: Recursive lambda with auto
    auto dfs = [&](auto self, int x) -> void {
        if (x > 10)
            return;
        cout << "DFS at x = " << x << endl;
        self(self, x + 1);
    };
    dfs(dfs, 1);
    // GNU C++17: Template argument deduction for vector
    vector in(2, vector<int>(2, 1));
    for (auto x : in) {
        for (auto y : x) {
            cout << y << " ";
        }
        cout << endl;
    }

    // GNU C++17: Structured bindings
    map<int, int> dic = {{1, 2}, {3, 4}};
    for (auto [u, v] : dic) {
        cout << "{" << u << ", " << v << "} ";
    }
    cout << endl;
    // GNU C++20: contains method for map
    if (dic.contains(1)) {
        cout << "contains" << endl;
    } else {
        cout << "not contain" << endl;
    }

    return 0;
}
```

8.11 test_speed

```
// #pragma GCC optimize("Ofast", "unroll-loops")
#include <bits/stdc++.h>
using namespace std;
signed main() {
    int n = 4E3;
    bitset<30> ans;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j += 2) {
            for (int k = 1; k <= n; k += 4) {
                ans |= i | j | k;
            }
        }
    }
    cout << ans.to_ullong() << "\n";
}
```

8.12 teststack

```
#include <bits/stdc++.h>
using namespace std;
int cur = 1;
```

```
// 需要关闭O2测试
void func() {
    cout << cur << "MB" << endl;
    char arr[1024 * 1024];  // 1MB
    // 使用数组以防止优化
    // if (cur > 1024 )
    //     return;
    int sum = 0;
    cur++;
    func();
}
int main() {
    func();
    cout << "Yes" << endl;
    cout << cur << "MB" << endl;
    return 0;
}
```