Programming Contest Templates

Senyang Su

October 29, 2025

v1.0.1

Contents

1 data-structure **1**
- 1.1 DSU.cpp . . . 1
- 1.2 Fenwick.cpp . . . 2
- 1.3 SegmentTree.cpp . . . 5

2 geometry **8**
- 2.1 geo2D.cpp . . . 8

3 graph **10**
- 3.1 Dijkstra.cpp . . . 10
- 3.2 eBcc.cpp . . . 12
- 3.3 Prim.cpp . . . 14
- 3.4 SCC.cpp . . . 16
- 3.5 SPFA.cpp . . . 19
- 3.6 vBCC.cpp . . . 21

4 math **23**
- 4.1 int128.cpp . . . 23
- 4.2 Matrix.cpp . . . 24
- 4.3 ModInt.cpp . . . 29
- 4.4 QuickPower.cpp . . . 31
- 4.5 Sieve.cpp . . . 32

5 others **33**
- 5.1 Gen.cpp . . . 33
- 5.2 test.sh . . . 35

6 string **36**
- 6.1 BinaryTrie.cpp . . . 36
- 6.2 exkmp.cpp . . . 39
- 6.3 hash.cpp . . . 41
- 6.4 kmp.cpp . . . 44
- 6.5 Trie.cpp . . . 46

7 tree **48**
- 7.1 LCA.cpp . . . 48

1 data-structure

1.1 DSU.cpp

用法示例:

```cpp
struct DSU {
    // ...
};

int main {
    DSU dsu(n); // 初始化一个并查集
    dsu.merge(x, y); // 合并第 x 个元素与第 y 个元素
    std::cout << dsu.same(x, y) << '\n'; // 判断第 x 个元素与第 y 个元素是否在同一个集合中
}
```

注意: 初始化的并查集下标从 1 开始。

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

using i64 = long long;

struct DSU {
    /*
    并查集
    DSU dsu(n);
    */
    int n;
    std::vector<int> fa, sz;

    explicit DSU(int n) {
        fa.assign(n + 1, 0), sz.assign(n + 1, 1);
        for (int i = 1; i <= n; ++i) {
            fa[i] = i;
        }
    }

    int find(int x) {
        return fa[x] == x ? x : fa[x] = find(fa[x]);
    }

    void merge(int x, int y) {
        int fx = find(x), fy = find(y);
        if (fx == fy) {
            return;
        }
        if (sz[fx] < sz[fy]) {
            std::swap(fx, fy);
        }
        fa[fy] = fx;
        sz[fx] += sz[fy];
    }
```

```cpp

    bool same(int x, int y) {
        return find(x) == find(y);
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int n, m;
    std::cin >> n >> m;

    DSU dsu(n);

    while (m--) {
        int o, x, y;
        std::cin >> o >> x >> y;
        if (o == 1) {
            dsu.merge(x, y);
        } else {
            std::cout << (dsu.same(x, y) ? "Y" : "N") << '\n';
        }
    }

    return 0;
}
```

1.2 Fenwick.cpp

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

using i64 = long long;

template<class T>
struct Fenwick {
    /*
    树状数组维护单点加，区间和
    使用 Fenwick<i64> tree(a); 初始化树状数组
    单点加: tree.add(pos, val);
    区间和: tree.sum(l, r)
    */
    int n;
    std::vector<T> tr;

    Fenwick() {
        n = 0;
        tr.assign(n + 1, T());
    }

    explicit Fenwick(const std::vector<T> &a) {
        n = a.size() - 1;
        tr.assign(n + 1, T());
```

```
        if (n) {
            for (int i = 1; i <= n; ++i) {
                tr[i] += a[i];
                int j = i + (i & -i);
                if (j <= n) tr[j] += tr[i];
            }
        }
    }

    void add(int pos, const T &val) {
        for (int i = pos; i <= n; i += i & -i) {
            tr[i] += val;
        }
    }

    T prefix(int pos) {
        T res = T();
        for (int i = pos; i; i -= i & -i) {
            res += tr[i];
        }
        return res;
    }

    T sum(int l, int r) {
        if (l > r) {
            return T();
        }
        return prefix(r) - prefix(l - 1);
    }
};

template<class T>
struct DoubleFenwick {
    /*
    树状数组维护区间加，区间和
    使用 DoubleFenwick<i64> tree(a); 初始化树状数组
    区间加: tree.add(l, r, val);
    区间和: tree.sum(l, r)
    */
    int n;
    Fenwick<T> tr1, tr2; // tr1 维护 d[i] 的前缀和, tr2 维护 d[i] * i 的前缀和

    explicit DoubleFenwick(const std::vector<T> &a) {
        n = a.size() - 1;
        std::vector<T> d(n + 1);
        for (int i = 1; i <= n; ++i) {
            d[i] = a[i] - a[i - 1];
        }
        tr1 = Fenwick<T>(d);
        for (int i = 1; i <= n; i++) {
            d[i] = d[i] * i;
        }
        tr2 = Fenwick<T>(d);
    }
```

```

    void add(int l, int r, const T &val) {
        if (l > r) {
            return;
        }
        tr1.add(l, val);
        if (r + 1 <= n) {
            tr1.add(r + 1, -val);
        }

        tr2.add(l, val * l);
        if (r + 1 <= n) {
            tr2.add(r + 1, -val * (r + 1));
        }
    }

    T prefix(int pos) {
        return tr1.prefix(pos) * (pos + 1) - tr2.prefix(pos);
    }

    T sum(int l, int r) {
        if (l > r) {
            return T();
        }
        return prefix(r) - prefix(l - 1);
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int n, m;
    std::cin >> n >> m;

    std::vector<i64> a(n + 1);
    for (int i = 1; i <= n; i++) {
        std::cin >> a[i];
    }

    DoubleFenwick<i64> tree(a);

    while (m--) {
        int o;
        std::cin >> o;

        if (o == 1) {
            int x, y;
            i64 k;
            std::cin >> x >> y >> k;
            tree.add(x, y, k);
        } else {
            int l, r;
            std::cin >> l >> r;
```

```
            std::cout << tree.sum(l, r) << '\n';
        }
    }

    return 0;
}
```

1.3 SegmentTree.cpp

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

using i64 = long long;

constexpr i64 INF = 1e18;

template<class Info, class Tag>
struct SegmentTree {
    /*
    线段树维护区间加，查询区间和，区间最值
    使用 SegmentTree<Info, Tag> st(a); 初始化线段树
    区间加: st.modify(L, R, val);
    查询区间和: st.query(L, R).sum;
    查询区间最大值: st.query(L, R).max;
    查询区间最小值: st.query(L, R).min;
    */
    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;

    explicit SegmentTree(const std::vector<i64>& a) {
        n = (int)a.size() - 1;
        info.assign(4 * n + 10, Info()), tag.assign(4 * n + 10, Tag());
        if (n) {
            build(1, 1, n, a);
        }
    }

    void modify(int L, int R, i64 val) {
        if (L > R || n == 0) {
            return;
        }
        modify(1, 1, n, L, R, Tag(val));
    }

    Info query(int L, int R) {
        if (L > R) {
            return Info();
        }
        return query(1, 1, n, L, R);
    }

private:
    void pushup(int u) {
```

```
        info[u] = info[u << 1] + info[u << 1 | 1];
    }

    void apply(int u, const Tag &t) {
        info[u].apply(t), tag[u].apply(t);
    }

    void pushdown(int u) {
        apply(u << 1, tag[u]), apply(u << 1 | 1, tag[u]);
        tag[u] = Tag();
    }

    void build(int u, int l, int r, const std::vector<i64>& a) {

        if (l == r) {
            info[u] = Info(a[l]);
            return;
        }
        int mid = (l + r) >> 1;
        build(u << 1, l, mid, a);
        build(u << 1 | 1, mid + 1, r, a);
        pushup(u);
    }

    void modify(int u, int l, int r, int L, int R, const Tag &t) {
        if (L <= l && r <= R) {
            apply(u, t);
            return;
        }
        pushdown(u);
        int mid = (l + r) >> 1;
        if (L <= mid) {
            modify(u << 1, l, mid, L, R, t);
        }
        if (R > mid) {
            modify(u << 1 | 1, mid + 1, r, L, R, t);
        }
        pushup(u);
    }

    Info query(int u, int l, int r, int L, int R) {
        if (L <= l && r <= R) {
            return info[u];
        }
        pushdown(u);
        int mid = (l + r) >> 1;
        Info res = Info();
        if (L <= mid) {
            res = res + query(u << 1, l, mid, L, R);
        }
        if (R > mid) {
            res = res + query(u << 1 | 1, mid + 1, r, L, R);
        }
        return res;
```

```cpp
    }
};

struct Tag {
    // Lazy tag，例如区间加法，区间乘法
    i64 add; // 区间加法

    Tag() : add(0) {}
    Tag(i64 x) : add(x) {}

    void apply(const Tag &oth) {
        // 如果更换维护的更改操作（例如区间加法，区间乘法，区间赋值），需要更改这里
        add += oth.add;
    }
};

struct Info {
    // 需要维护的信息，例如区间和，区间最值
    i64 min, max, sum;
    int cnt; // 区间长度

    Info() : min(INF), max(-INF), sum(0), cnt(0) {}
    Info(i64 x) : min(x), max(x), sum(x), cnt(1) {}
    Info(i64 min, i64 max, i64 sum, int cnt) : min(min), max(max), sum(sum), cnt(cnt) {}

    Info operator+(const Info &oth) const {
        return {std::min(min, oth.min), std::max(max, oth.max), sum + oth.sum, cnt + oth.cnt};
    }

    void apply(const Tag &tag) {
        // 如果更换维护的更改操作（例如区间加法，区间乘法，区间赋值），需要更改这里
        min += tag.add, max += tag.add, sum += cnt * 1ll * tag.add;
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int n, m;
    std::cin >> n >> m;

    std::vector<i64> a(n + 1);
    for (int i = 1; i <= n; ++i) {
        std::cin >> a[i];
    }

    SegmentTree<Info, Tag> seg(a);

    while (m--) {
        int op, l, r;
```

```
        i64 x;
        std::cin >> op;
        if (op == 1) {
            std::cin >> l >> r >> x;
            seg.modify(l, r, x);
        } else {
            std::cin >> l >> r;
            std::cout << seg.query(l, r).sum << '\n';
        }
    }

    return 0;
}
```

2 geometry

2.1 geo2D.cpp

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

using i64 = long long;

constexpr long double eps = 1e-12;

template<class T>
struct Point {
    T x, y;

    Point(T x = 0, T y = 0) : x(x), y(y) {}

    template<class U> operator Point<U>() const { return Point<U>(U(x), U(y));
    ↪ }

    Point& operator+=(const Point& o) {
        x += o.x, y += o.y;
        return *this;
    }

    Point& operator-=(const Point& o) {
        x -= o.x, y -= o.y;
        return *this;
    }

    Point& operator*=(T k) {
        x *= k, y *= k;
        return *this;
    }

    Point& operator/=(T k) {
        x /= k, y /= k;
        return *this;
    }
```

```

    friend Point operator+(Point a, const Point& b) { return a += b; }
    friend Point operator-(Point a, const Point& b) { return a -= b; }
    friend Point operator*(Point a, T k) { return a *= k; }
    friend Point operator*(T k, Point a) { return a *= k; }
    friend Point operator/(Point a, T k) { return a /= k; }

    friend std::istream& operator>>(std::istream& is, Point& p) {
        return is >> p.x >> p.y;
    }

    friend std::ostream& operator<<(std::ostream& os, const Point& p) {
        return os << "(" << p.x << ", " << p.y << ")";
    }

    Point<T> norm() const {
        T magnitude = abs(*this);
        return *this / magnitude;
    }
};

template<class T>
T dot(Point<T> a, Point<T> b) {
    return a.x * b.x + a.y * b.y;
}

template<class T>
T cross(Point<T> a, Point<T> b) {
    return a.x * b.y - a.y * b.x;
}

template<class T>
T sq(Point<T> a) {
    return dot(a, a);
}

template<class T>
long double abs(Point<T> a) {
    return sqrt((long double)sq(a));
}

template<class T>
T angle_to(const Point<T> &a, const Point<T> &b) {
    T angle_a = std::atan2(a.y, a.x);
    T angle_b = std::atan2(b.y, b.x);

    T angle_diff = angle_b - angle_a;

    if (angle_diff < 0) {
        angle_diff += 2 * M_PI;
    }

    return angle_diff; // 弧度
}
```

```cpp

template<class T>
struct Line {
    Point<T> a, b;

    Line(Point<T> a, Point<T> b) : a(a), b(b) {}
};

template<class T>
T distP2L(Point<T> p, Line<T> l) {
    Point<T> p1 = l.a, p2 = l.b;

    T nume = std::abs((p2.x - p1.x) * (p1.y - p.y) - (p1.x - p.x) * (p2.y - p1.y));
    T deno = std::sqrtl((p2.x - p1.x) * (p2.x - p1.x) + (p2.y - p1.y) * (p2.y - p1.y));

    return nume / deno;
}

// 判断两条直线是否平行
template<class T>
bool is_parallel(const Line<T>& l1, const Line<T>& l2) {
    return std::abs(cross(l1.b - l1.a, l2.b - l2.a)) < eps;
}

// 计算两条直线的交点
template<class T>
Point<T> intersection(const Line<T>& l1, const Line<T>& l2) {
    if (is_parallel(l1, l2)) {
        throw std::runtime_error("The lines are parallel, no intersection.");
    }

    return l1.a + (l1.b - l1.a) * cross(l2.b - l2.a, l1.a - l2.a) / cross(l1.b - l1.a, l2.b - l2.a);
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);


    return 0;
}
```

3 graph

3.1 Dijkstra.cpp

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

using i64 = long long;
```

```cpp

template <class T>
struct Graph {
    /*
    正权图, dijstra 求最短路
    Graph<i64> g(n);
    g.addEdge(u, v, w); // 无向边
    g.addArc(u, v, w); // 有向边

    std::vector<int> par;
    g.dijkstra(s, &par); // 求最短路, 并记录路径
    */
    int n;
    std::vector<std::vector<std::pair<int, T>>> adj;
    std::vector<T> dist;

    Graph(int n) {
        this->n = n;
        adj.assign(n + 1, {});
        dist.assign(n + 1, -1);
    }

    void addEdge(int u, int v, T w) {
        adj[u].emplace_back(v, w), adj[v].emplace_back(u, w);
    }

    void addArc(int u, int v, T w) {
        adj[u].emplace_back(v, w);
    }

    void dijkstra(int s, std::vector<int> *par = nullptr) {
        dist.assign(n + 1, -1);
        if (par) {
            par->assign(n + 1, -1);
        }

        using P = std::pair<T, int>;

        std::priority_queue<P, std::vector<P>, std::greater<P>> pq;
        dist[s] = 0;
        pq.emplace(0, s);

        while (!pq.empty()) {
            auto [d, u] = pq.top();
            pq.pop();
            if (dist[u] != -1 && d > dist[u]) {
                continue;
            }
            for (auto [v, w] : adj[u]) {
                T nd = d + w;
                if (dist[v] == -1 || nd < dist[v]) {
                    dist[v] = nd;
                    pq.emplace(nd, v);
                    if (par) {
```

```
                        (*par)[v] = u;
                    }
                }
            }
        }
    }

    static std::vector<int> get_path(int s, int t, std::vector<int> &par) {
        std::vector<int> path;
        for (int v = t; v != -1; v = par[v]) {
            path.push_back(v);
        }
        std::reverse(path.begin(), path.end());

        return path;
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int n, m, s;
    std::cin >> n >> m >> s;

    Graph<i64> g(n);
    for (int i = 0; i < m; i++) {
        int u, v;
        i64 w;
        std::cin >> u >> v >> w;
        g.addArc(u, v, w);
    }

    g.dijkstra(s);

    for (int i = 1; i <= n; i++) {
        std::cout << g.dist[i] << " \n"[i == n];
    }

    return 0;
}
```

3.2 eBcc.cpp

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

using i64 = long long;

struct eBCC {
    /*
    tarjan 求 eBCC，无向图
    bcc_id[i] 表示 i 所在的 BCC 编号
    bcc[i] 表示第 i 个 BCC 的顶点集合
```

```
    bridge 存储所有桥边
    */
    int n, m, ind, bcc_cnt, e_cnt;
    std::vector<std::vector<std::pair<int, int>>> adj;
    std::vector<int> dfn, low, is_bridge;
    std::vector<int> bcc_id; // 每个顶点所在的 BCC 编号
    std::vector<std::vector<int>> bcc; // 每个边双的顶点集合
    std::vector<std::pair<int, int>> bridge;

    eBCC(int n, int m) {
        this->n = n;
        this->m = m;
        ind = bcc_cnt = e_cnt = 0;
        adj.assign(n + 1, {});
        dfn.assign(n + 1, 0);
        low.assign(n + 1, 0);
        bcc_id.assign(n + 1, -1);
        is_bridge.assign(2 * m + 2, 0);
    }

    void addEdge(int u, int v) {
        adj[u].push_back({v, e_cnt++}), adj[v].push_back({u, e_cnt++});
    }

    void dfs(int u, int fid) {
        dfn[u] = low[u] = ++ind;
        for (auto [v, eid] : adj[u]) {
            if (!dfn[v]) {
                dfs(v, eid);
                low[u] = std::min(low[u], low[v]);
                if (low[v] > dfn[u]) {
                    is_bridge[eid] = is_bridge[eid ^ 1] = 1;
                    bridge.push_back({u, v});
                }
            } else if (eid != (fid ^ 1)) {
                low[u] = std::min(low[u], dfn[v]);
            }
        }
    }

    void comp(int u, int fid, int cid) {
        bcc_id[u] = cid;
        bcc[cid].push_back(u);
        for (auto [v, eid] : adj[u]) {
            if (!is_bridge[eid] && bcc_id[v] == -1 && eid != (fid ^ 1)) {
                comp(v, eid, cid);
            }
        }
    }

    int tarjan() {
        for (int i = 1; i <= n; i++) {
            if (!dfn[i]) {
                dfs(i, -1);
```

```
            }
        }
        for (int i = 1; i <= n; i++) {
            if (bcc_id[i] == -1) {
                bcc.push_back({});
                bcc_cnt++;
                comp(i, -1, bcc_cnt - 1);
            }
        }
        return bcc_cnt;
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int n, m;
    std::cin >> n >> m;

    eBCC graph(n, m);
    for (int i = 1; i <= m; i++) {
        int u, v;
        std::cin >> u >> v;
        if (u != v) {
            graph.addEdge(u, v);
        }
    }

    std::cout << graph.tarjan() << '\n';
    for (auto bcc : graph.bcc) {
        std::cout << bcc.size() << ' ';
        for (auto u : bcc) {
            std::cout << u << ' ';
        }
        std::cout << '\n';
    }

    return 0;
}
```

3.3 Prim.cpp

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

using i64 = long long;

template <class T>
struct Graph {
    /*
    Prim 求最小生成树
    */
    int n;
```

```cpp
    std::vector<std::vector<std::pair<int, T>>> adj;

    Graph(int n) {
        this->n = n;
        adj.assign(n + 1, {});
    }

    void addEdge(int u, int v, T w) {
        adj[u].emplace_back(v, w), adj[v].emplace_back(u, w);
    }

    void addArc(int u, int v, T w) {
        adj[u].emplace_back(v, w);
    }

    std::pair<T, std::vector<int>> prim(int s, int &reached) {
        constexpr T INF = std::numeric_limits<T>::max() / 4;
        std::vector<int> pre(n + 1, -1); // 树上节点的前驱
        std::vector<int> inMST(n + 1);
        std::vector<T> cost(n + 1, INF);

        using P = std::pair<T, int>;
        std::priority_queue<P, std::vector<P>, std::greater<P>> pq;

        T minCost = 0;
        reached = 0;

        pq.emplace(0, s);
        while (!pq.empty()) {
            auto [d, u] = pq.top();
            pq.pop();
            if (inMST[u]) {
                continue;
            }
            inMST[u] = 1, cost[u] = d, minCost += d, reached++;
            for (auto [v, w] : adj[u]) {
                if (!inMST[v] && cost[v] > w) {
                    cost[v] = w, pre[v] = u;
                    pq.emplace(w, v);
                }
            }
        }
        return {minCost, pre};
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int n, m;
    std::cin >> n >> m;

    Graph<i64> g(n);
```

```
    for (int i = 0; i < m; i++) {
        int u, v;
        i64 w;
        std::cin >> u >> v >> w;
        g.addEdge(u, v, w);
    }

    int r = -1;
    auto [ans, _] = g.prim(1, r);

    if (r < n) {
        std::cout << "orz" << '\n';
    } else {
        std::cout << ans << '\n';
    }

    return 0;
}
```

3.4 SCC.cpp

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

using i64 = long long;

struct SCC {
    /*
    tarjan 求 SCC，有向图
    scc_id[i] 表示 i 所在的 SCC 编号
    例如：对 scc 缩点之后跑 dp
    SCC scc(n);
    scc.addArc(u, v); ...
    scc.tarjan();

    std::vector<std::vector<int>> dag(scc.scc_cnt + 1);
    for (int u = 1; u <= n; u++) {
        for (auto v : scc.adj[u]) {
            if (scc.scc_id[u] != scc.scc_id[v]) {
                dag[scc.scc_id[u]].push_back(scc.scc_id[v]);
                // scc.scc_id[u] -> scc.scc_id[v]
            }
        }
    }
    */
    int n, ind, scc_cnt;
    std::vector<std::vector<int>> adj;
    std::vector<int> dfn, low, in_stk, scc_id;
    std::stack<int> stk;

    SCC(int n) {
        this->n = n;
        ind = scc_cnt = 0;
        adj.assign(n + 1, {});
```

```
        dfn.assign(n + 1, 0);
        low.assign(n + 1, 0);
        in_stk.assign(n + 1, 0);
        scc_id.assign(n + 1, 0);
    }

    void addArc(int u, int v) {
        adj[u].push_back(v);
    }

    void dfs(int u) {
        dfn[u] = low[u] = ++ind;
        stk.push(u), in_stk[u] = 1;

        for (auto v : adj[u]) {
            if (!dfn[v]) {
                dfs(v);
                low[u] = std::min(low[u], low[v]);
            } else if (in_stk[v]) {
                low[u] = std::min(low[u], dfn[v]);
            }
        }

        if (low[u] == dfn[u]) {
            ++scc_cnt;
            while (1) {
                int x = stk.top();
                stk.pop();
                in_stk[x] = 0;
                scc_id[x] = scc_cnt;
                if (x == u) {
                    break;
                }
            }
        }
    }

    int tarjan() {
        for (int i = 1; i <= n; i++) {
            if (!dfn[i]) {
                dfs(i);
            }
        }
        return scc_cnt;
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int n, m;
    std::cin >> n >> m;

```

```
    std::vector<int> a(n + 1);
    for (int i = 1; i <= n; i++) {
        std::cin >> a[i];
    }

    SCC graph(n);
    for (int i = 1; i <= m; i++) {
        int u, v;
        std::cin >> u >> v;

        graph.addArc(u, v);
    }

    graph.tarjan();

    std::vector<std::vector<int>> dag(graph.scc_cnt + 1);
    std::vector<int> in(graph.scc_cnt + 1);
    for (int u = 1; u <= n; u++) {
        for (auto v : graph.adj[u]) {
            if (graph.scc_id[u] != graph.scc_id[v]) {
                dag[graph.scc_id[u]].push_back(graph.scc_id[v]);
                in[graph.scc_id[v]]++;
            }
        }
    }

    std::vector<int> val(graph.scc_cnt + 1);
    for (int i = 1; i <= n; i++) {
        val[graph.scc_id[i]] += a[i];
    }

    std::vector<int> dp(graph.scc_cnt + 1);
    std::queue<int> q;

    for (int i = 1; i <= graph.scc_cnt; i++) {
        if (!in[i]) {
            q.push(i);
            dp[i] = val[i];
        }
    }

    while (!q.empty()) {
        auto now = q.front();
        q.pop();

        for (auto v : dag[now]) {
            dp[v] = std::max(dp[v], dp[now] + val[v]);
            in[v]--;
            if (in[v] == 0) {
                q.push(v);
            }
        }
    }

```

```
    int ans = *std::max_element(dp.begin() + 1, dp.end());
    std::cout << ans << '\n';

    return 0;
}
```

3.5 SPFA.cpp

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

using i64 = long long;

template <class T>
struct Graph {
    /*
    SPFA 求最短路
    g.addEdge(u, v, w); // 无向边
    g.addArc(u, v, w); // 有向边

    std::vector<int> par;
    g.SPFA(s, &par); // 求最短路, 并记录路径
    */
    int n;
    std::vector<std::vector<std::pair<int, T>>> adj;
    std::vector<T> dist;

    const T INF = std::numeric_limits<T>::max() / 4;

    Graph(int n) {
        this->n = n;
        adj.assign(n + 1, {});
        dist.assign(n + 1, INF);
    }

    void addEdge(int u, int v, T w) {
        adj[u].emplace_back(v, w), adj[v].emplace_back(u, w);
    }

    void addArc(int u, int v, T w) {
        adj[u].emplace_back(v, w);
    }

    bool SPFA(int s, std::vector<int> *par = nullptr) {

        std::vector<int> inq(n + 1);
        std::vector<int> cnt(n + 1); // 入队次数
        if (par) {
            par->assign(n + 1, -1);
        }

        std::deque<int> q;
        q.push_back(s);
        inq[s] = 1, dist[s] = 0;
```

```

        while (!q.empty()) {
            int u = q.front();
            q.pop_front();
            inq[u] = 0;

            T du = dist[u];
            if (du == INF) {
                continue;
            }

            for (auto [v, w] : adj[u]) {
                if (du + w < dist[v]) {
                    dist[v] = du + w;
                    if (par) {
                        (*par)[v] = u;
                    }

                    if (!inq[v]) {
                        if (!q.empty() && dist[v] < dist[q.front()]) {
                            q.push_front(v);
                        } else {
                            q.push_back(v);
                        }

                        inq[v] = 1;

                        if (++cnt[v] == n) {
                            return 0; // 存在负环
                        }
                    }
                }
            }
        }
        for (int i = 1; i <= n; ++i) {
            dist[i] = (dist[i] == INF ? (T)-1 : dist[i]);
        }
        return 1;
    }

    static std::vector<int> get_path(int s, int t, std::vector<int> &par) {
        std::vector<int> path;
        for (int v = t; v != -1; v = par[v]) {
            path.push_back(v);
        }
        std::reverse(path.begin(), path.end());

        return path;
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
```

```

    int n, m, s;
    std::cin >> n >> m >> s;

    Graph<i64> g(n);
    for (int i = 0; i < m; i++) {
        int u, v;
        i64 w;
        std::cin >> u >> v >> w;
        g.addArc(u, v, w);
    }

    g.SPFA(s);

    for (int i = 1; i <= n; i++) {
        std::cout << g.dist[i] << " \n"[i == n];
    }

    return 0;
}
```

3.6 vBCC.cpp

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

using i64 = long long;

struct vBCC {
    /*
    tarjan 求 vBCC，无向图
    bcc_id[i] 表示 i 所在的 BCC 编号
    bcc[i] 表示第 i 个 BCC 的顶点集合
    is_cut[i] 表示 i 是否是割点
    */
    int n, ind, bcc_cnt, root;
    std::vector<std::vector<int>> adj;
    std::vector<int> dfn, low, is_cut;
    std::vector<std::vector<int>> bcc; // 每个点双的顶点集合，编号从 0 开始
    std::stack<int> stk;

    vBCC(int n) {
        this->n = n;
        ind = bcc_cnt = 0;
        adj.assign(n + 1, {});
        dfn.assign(n + 1, 0);
        low.assign(n + 1, 0);
        is_cut.assign(n + 1, 0);
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v), adj[v].push_back(u);
    }

```

```
    void dfs(int u, int f) {
        dfn[u] = low[u] = ++ind;
        stk.push(u);

        if (u == root && adj[u].empty()) {
            bcc_cnt++;
            bcc.push_back({u});
            stk.pop();
            return;
        }

        int child = 0;
        for (auto v : adj[u]) {
            if (!dfn[v]) {
                child++;
                dfs(v, u);
                low[u] = std::min(low[u], low[v]);

                if (low[v] >= dfn[u]) {
                    if (u != root || child > 1) {
                        is_cut[u] = 1;
                    }
                    bcc_cnt++;
                    std::vector<int> tmp;
                    int z;
                    do {
                        z = stk.top();
                        stk.pop();
                        tmp.push_back(z);
                    } while (z != v);
                    tmp.push_back(u);
                    bcc.push_back(std::move(tmp));
                }
            } else if (v != f && dfn[v] < dfn[u]) {
                low[u] = std::min(low[u], dfn[v]);
            }
        }
    }

    int tarjan() {
        for (int i = 1; i <= n; i++) {
            if (!dfn[i]) {
                root = i;
                dfs(i, 0);
            }
        }
        return bcc_cnt;
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

```

```
    int n, m;
    std::cin >> n >> m;

    vBCC graph(n);
    for (int i = 1; i <= m; i++) {
        int u, v;
        std::cin >> u >> v;
        if (u != v) {
            graph.addEdge(u, v);
        }
    }

    std::cout << graph.tarjan() << '\n';

    for (auto bcc : graph.bcc) {
        std::cout << bcc.size() << ' ';
        for (auto u : bcc) {
            std::cout << u << ' ';
        }
        std::cout << '\n';
    }

    return 0;
}
```

4 math

4.1 int128.cpp

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

using i128 = __int128_t;
using u128 = __uint128_t;

inline std::ostream& operator<<(std::ostream& os, i128 x) {
    if (x == 0) return os << '0';
    if (x < 0) { os << '-'; x = -x; }
    char s[64]; int p = 0;
    while (x) { s[p++] = char('0' + x % 10); x /= 10; }
    while (p) os << s[--p];
    return os;
}
inline std::ostream& operator<<(std::ostream& os, u128 x) {
    if (x == 0) return os << '0';
    char s[64]; int p = 0;
    while (x) { s[p++] = char('0' + x % 10); x /= 10; }
    while (p) os << s[--p];
    return os;
}

inline std::istream& operator>>(std::istream& is, i128& x) {
    is >> std::ws;
```

```
    bool neg = false;
    int c = is.peek();
    if (c == '+' || c == '-') { neg = (c == '-'); is.get(); }
    if (!std::isdigit(is.peek())) { is.setstate(std::ios::failbit); return is; }
    u128 v = 0;
    while (std::isdigit(is.peek())) v = v * 10 + (is.get() - '0');
    x = neg ? -i128(v) : i128(v);
    return is;
}
inline std::istream& operator>>(std::istream& is, u128& x) {
    is >> std::ws;
    if (!std::isdigit(is.peek())) { is.setstate(std::ios::failbit); return is; }
    x = 0;
    while (std::isdigit(is.peek())) x = x * 10 + (is.get() - '0');
    return is;
}

int main() {
    i128 a, b;
    std::cin >> a >> b;
    std::cout << a / b << '\n';

    return 0;
}
```

4.2 Matrix.cpp

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

using i64 = long long;

template<typename T, T MOD>
class Matrix {
    /*
    using Mat = Matrix<i64, 1000000007>;

    int n;
    i64 k;
    std::cin >> n >> k;

    Mat A(n, n);
    for (int i = 1; i <= A.rows(); i++) {
        for (int j = 1; j <= A.cols(); j++) {
            std::cin >> A.at(i, j);
        }
    }

    auto res = A.pow(k);
    res.print();
    */
private:
```

```cpp
    int n, m;                   // 行列数
    std::vector<T> data;            // 一维存储

    // 工具：把值规范到模意义下
    static T norm(T x) {
        // 假设 T 是带符号整数类型或无符号整型可以做 % 运算
        x %= MOD;
        if (x < 0) x += MOD;
        return x;
    }

public:
    // 构造函数：默认 0 填充
    Matrix(int n, int m) : n(n), m(m), data(n * m, 0) {}

    // 访问元素（1-index，返回引用）
    T& at(int i, int j) {
        assert(1 <= i && i <= n && 1 <= j && j <= m);
        return data[(i - 1) * m + (j - 1)];
    }

    // 常量访问
    const T& at(int i, int j) const {
        assert(1 <= i && i <= n && 1 <= j && j <= m);
        return data[(i - 1) * m + (j - 1)];
    }

    // 模数
    static constexpr T mod() { return MOD; }

    // 行列数访问器
    int rows() const { return n; }
    int cols() const { return m; }

    // 矩阵加法（逐元素相加，模 MOD）
    Matrix operator+(const Matrix& oth) const {
        assert(n == oth.n && m == oth.m);
        Matrix res(n, m);
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= m; ++j) {
                T v = at(i, j) + oth.at(i, j);
                res.at(i, j) = norm(v);
            }
        }
        return res;
    }

    // 自加（A += B）
    Matrix& operator+=(const Matrix& oth) {
        assert(n == oth.n && m == oth.m);
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= m; ++j) {
                T v = at(i, j) + oth.at(i, j);
                at(i, j) = norm(v);
```

```
            }
        }
        return *this;
    }

    // 矩阵减法（逐元素相减，模 MOD）
    Matrix operator-(const Matrix& oth) const {
        assert(n == oth.n && m == oth.m);
        Matrix res(n, m);
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= m; ++j) {
                T v = at(i, j) - oth.at(i, j);
                res.at(i, j) = norm(v);
            }
        }
        return res;
    }

    // 自减（A -= B）
    Matrix& operator-=(const Matrix& oth) {
        assert(n == oth.n && m == oth.m);
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= m; ++j) {
                T v = at(i, j) - oth.at(i, j);
                at(i, j) = norm(v);
            }
        }
        return *this;
    }

    // 标量乘法（k * A 或 A * k，逐元素乘以 k，然后模 MOD）
    Matrix operator*(T k) const {
        Matrix res(n, m);
        k = norm(k); // 先把标量也规约一下
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= m; ++j) {
                // 这里用 __int128 保护溢出可以考虑，但为了通用 T，不先上
                T v = (at(i, j) % MOD) * k;
                res.at(i, j) = norm(v);
            }
        }
        return res;
    }

    // 自乘（A *= k）
    Matrix& operator*=(T k) {
        k = norm(k);
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= m; ++j) {
                T v = (at(i, j) % MOD) * k;
                at(i, j) = norm(v);
            }
        }
        return *this;
```

```
    }

    // k * A
    friend Matrix operator*(T k, const Matrix& mat) {
        return mat * k;
    }

    // 点乘 / Hadamard 逐元素相乘（同型矩阵）
    // res[i,j] = A[i,j] * B[i,j] (mod MOD)
    Matrix hadamard(const Matrix& oth) const {
        assert(n == oth.n && m == oth.m);
        Matrix res(n, m);
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= m; ++j) {
                T v = (at(i, j) % MOD) * (oth.at(i, j) % MOD);
                res.at(i, j) = norm(v);
            }
        }
        return res;
    }

    // 转置矩阵（返回新对象）
    Matrix transpose() const {
        Matrix res(m, n); // 维度反过来
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= m; ++j) {
                res.at(j, i) = at(i, j);
            }
        }
        return res;
    }

    // 矩阵乘法
    Matrix operator*(const Matrix& oth) const {
        assert(m == oth.n); // 左的列数 == 右的行数
        Matrix res(n, oth.m);

        for (int i = 1; i <= n; ++i) {
            for (int k = 1; k <= m; ++k) {
                // 先取出 a_ik
                T a_ik = at(i, k) % MOD;
                if (a_ik == 0) continue;
                for (int j = 1; j <= oth.m; ++j) {
                    __int128 mul = (__int128)a_ik * (__int128)(oth.at(k, j) %
                    ↪ MOD);
                    T add_val = (T)(mul % MOD);

                    T cur = res.at(i, j) + add_val;
                    if (cur >= MOD) cur -= MOD;
                    res.at(i, j) = cur;
                }
            }
        }

```

```
        return res;
    }

    Matrix& operator*=(const Matrix& oth) {
        assert(m == oth.n);

        int new_n = n;
        int new_m = oth.m;
        int mid   = m;

        std::vector<T> newData((size_t)new_n * new_m, (T)0);

        for (int i = 1; i <= new_n; ++i) {
            for (int k = 1; k <= mid; ++k) {
                T a_ik = at(i, k) % MOD;
                if (a_ik == 0) continue;
                for (int j = 1; j <= new_m; ++j) {
                    __int128 mul = (__int128)a_ik * (__int128)(oth.at(k, j) %
                    ↪  MOD);
                    T add_val = (T)(mul % MOD);

                    T &cell = newData[(size_t)(i - 1) * new_m + (j - 1)];
                    T cur = cell + add_val;
                    if (cur >= MOD) cur -= MOD;
                    cell = cur;
                }
            }
        }

        n = new_n;
        m = new_m;
        data.swap(newData);

        return *this;
    }

    static Matrix identity(int n) {
        Matrix I(n, n);
        for (int i = 1; i <= n; ++i) {
            I.at(i, i) = 1 % MOD;
        }
        return I;
    }

    Matrix pow(long long e) const {
        assert(n == m);
        assert(e >= 0);

        Matrix base = *this;
        Matrix res  = identity(n);

        long long k = e;
        while (k > 0) {
            if (k & 1LL) {
```

```
                res *= base;
            }
            k >>= 1LL;
            if (k > 0) {
                base *= base;
            }
        }
        return res;
    }

    void print() const {
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= m; ++j) {
                std::cout << at(i, j) << " \n"[j == m];
            }
        }
    }
};

using Mat = Matrix<i64, 1000000007>;

int main() {
    int n;
    i64 k;
    std::cin >> n >> k;

    Mat A(n, n);
    for (int i = 1; i <= A.rows(); i++) {
        for (int j = 1; j <= A.cols(); j++) {
            std::cin >> A.at(i, j);
        }
    }

    auto res = A.pow(k);
    res.print();

    return 0;
}
```

4.3 ModInt.cpp

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

template<int MOD>
struct ModInt {
    /*
    using Z = ModInt<998244353>;

    int main() {
        Z a = 2, b = 3;
        cout << a + b << "\n";    // 5
        cout << a * b << "\n";    // 6
        cout << a / b << "\n";    // 2 * inv(3)
```

```
        cout << a.pow(10) << "\n"; // 1024
    }
    */

    static_assert(MOD > 0, "MOD must be positive");

    int v;

    ModInt() : v(0) {}
    template<class T>
    ModInt(T x) {
        long long t = (long long)x % MOD;
        if (t < 0) t += MOD;
        v = (int)t;
    }

    static constexpr int mod() { return MOD; }
    int val() const { return v; }

    ModInt& operator+=(const ModInt& o) {
        v += o.v;
        if (v >= MOD) v -= MOD;
        return *this;
    }
    ModInt& operator-=(const ModInt& o) {
        v -= o.v;
        if (v < 0) v += MOD;
        return *this;
    }
    ModInt& operator*=(const ModInt& o) {
        v = (int)((__int128)v * o.v % MOD);
        return *this;
    }
    ModInt& operator/=(const ModInt& o) {
        return *this *= o.inv();
    }

    friend ModInt operator+(ModInt a, const ModInt& b) { return a += b; }
    friend ModInt operator-(ModInt a, const ModInt& b) { return a -= b; }
    friend ModInt operator*(ModInt a, const ModInt& b) { return a *= b; }
    friend ModInt operator/(ModInt a, const ModInt& b) { return a /= b; }
    ModInt operator-() const { return ModInt(v ? MOD - v : 0); }

    friend bool operator==(const ModInt& a, const ModInt& b) { return a.v == b.v; }
    friend bool operator!=(const ModInt& a, const ModInt& b) { return a.v != b.v; }

    ModInt pow(long long e) const {
        ModInt res = 1, base = *this;
        while (e > 0) {
            if (e & 1) res *= base;
            base *= base;
```

```
            e >>= 1;
        }
        return res;
    }

    ModInt inv() const {
        return pow(MOD - 2);
    }

    friend std::ostream& operator<<(std::ostream& os, const ModInt& a) {
        return os << a.v;
    }
    friend std::istream& operator>>(std::istream& is, ModInt& a) {
        long long x; is >> x;
        a = ModInt(x);
        return is;
    }
};

using Z = ModInt<998244353>;

int main() {
    Z a = 2, b = 3;
    std::cout << a + b << "\n";    // 5
    std::cout << a * b << "\n";    // 6
    std::cout << a / b << "\n";    // 2 * inv(3)
    std::cout << a.pow(10) << "\n"; // 1024
}

```

4.4 QuickPower.cpp

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

using i64 = long long;

i64 qpow(i64 a, i64 b, int mod) {
    i64 res = 1;
    while (b) {
        if (b & 1) {
            res = res * a % mod;
        }
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    // example
```

```
    i64 a = 2, b = 10, mod = 1e9 + 7;
    std::cout << qpow(a, b, mod) << '\n';

    return 0;
}
```

4.5 Sieve.cpp

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

using i64 = long long;

struct Sieve {
    /*
    Sieve sieve(n); // 0-indexed primes <= n

    // 对 n 分解质因数
    auto f = sieve.factorize(n);
    std::cout << n << " = ";
    for (int i = 0; i < (int)f.size(); i++) {
        std::cout << f[i].first << "^" << f[i].second;
        if (i + 1 < (int)f.size()) std::cout << " * ";
    }
    std::cout << '\n';
    */
    int n;
    std::vector<int> primes, minp;

    Sieve(int n_max = 0) {
        n = n_max;

        int pn = n / std::log(n);
        primes.reserve(pn);
        minp.assign(n + 1, 0);

        for (int i = 2; i <= n; i++) {
            if (!minp[i]) {
                minp[i] = i;
                primes.push_back(i);
            }
            for (int p : primes) {
                i64 v = i * 1ll * p;
                if (v > n) {
                    break;
                }
                minp[v] = p;
                if (p == minp[i]) {
                    break;
                }
            }
        }
    }

```

```cpp
    bool is_prime(int x) const {
        if (x < 2 || x > n) {
            return 0;
        }
        return minp[x] == x;
    }

    std::vector<std::pair<int,int>> factorize(int x) const {
        std::vector<std::pair<int,int>> res;
        while (x > 1) {
            int p = minp[x];
            int c = 0;
            while (x % p == 0) {
                x /= p;
                c++;
            }
            res.push_back({p, c});
        }
        return res;
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int n, q;
    std::cin >> n >> q;

    Sieve sieve(n);
    while (q--) {
        int k;
        std::cin >> k;

        std::cout << sieve.primes[k - 1] << '\n';
    }

    return 0;
}
```

5 others

5.1 Gen.cpp

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

using i64 = long long;

std::mt19937 rnd(std::chrono::steady_clock().now().time_since_epoch().count());

int rng(int l, int r) { // [l, r]
```

```
  return rnd() % (r - l + 1) + l;
}

void array(int n) { // Generates an array with n elements
  int m = 1E9;
  std::vector<int> a(n);
  for (int i = 0; i < n; i++) {
    a[i] = rng(0, m);
    std::cout << a[i] << " \n"[i == n - 1];
  }
}

void cand(int m, int n) { // Generate m cands from 1 to n
  for (int i = 0; i < m; i++) {
    int l = rng(1, n);
    int r = rng(1, n);

    if (l > r) {
      std::swap(l, r);
    }

    std::cout << l << " " << r << "\n";
  }
}

void tree(int n) { // Generate a tree with n vertices
  int m = 1E9;
  for (int i = 1; i < n; i++) {
    int p = rng(0, i - 1);
    int v = rng(1, m);
    std::cout << p + 1 << " " << i + 1 << " " << v << "\n";
  }
}

void graph(int n, int m) {
// Generate an undirected graph with n vertices and m edges. There are no
 ↪  double edges or self-rings in the graph, and must be connected.
  std::vector<std::pair<int, int>> e;
  std::map<std::pair<int, int>, bool> f;

  std::cout << n << " " << m << "\n";

  for (int i = 1; i < n; i++) {
    int p = rng(0, i - 1);
    e.push_back(std::make_pair(p, i));
    f[std::make_pair(p, i)] = f[std::make_pair(i, p)] = true;
  }

  // debug(f);
  for (int i = n; i <= m; i++) {
    int x, y;
    do {
      x = rng(0, n - 1);
      y = rng(0, n - 1);
```

```
    } while (x == y || f.count(std::make_pair(x, y)));

    e.push_back(std::make_pair(x, y));
    f[std::make_pair(x, y)] = f[std::make_pair(y, x)] = true;
  }

  std::shuffle(e.begin(), e.end(), rnd);

  for (auto [x, y] : e) {
    std::cout << x + 1 << " " << y + 1 << "\n";
  }
}

int main() {

  return 0;
}
```

5.2 test.sh

```
#!/usr/bin/env bash
set -euo pipefail

t=$1

g++ gen.cpp -o gen
g++ A.cpp -o A
g++ A__Good.cpp -o A__Good

for ((i=1; i<=t; i++)); do
    ./gen > in.txt
    ./A < in.txt > out.txt
    ./A__Good < in.txt > ans.txt

    if diff -q out.txt ans.txt > /dev/null; then
        echo "Accepted"
    else
        echo "Input"
        cat in.txt
        echo " "
        echo "Output"
        cat out.txt
        echo " "
        echo "Answer"
        cat ans.txt
        exit 1
    fi
done
```

6 string

6.1 BinaryTrie.cpp

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

using i64 = long long;

struct BinaryTrie {
    /*
    01-Trie（按位存 int）
    BinaryTrie bt;
    bt.insert(x, times=1);     // 插入
    bt.erase(x,  times=1);     // 删除
    bt.count(x);               // x 的出现次数
    bt.max_xor(x);             // 返回与 x 的最大异或值 (x XOR best)
    */

    static constexpr int A = 2;      // 只有 0/1 两个分支
    static constexpr int W = 31;     // 最高位下标（int 32 位: 31..0）

    struct Node {
        int ch[A];
        int pass;   // 经过该节点的数的个数
        int end;    // 在该节点结束的数的个数
        Node() {
            pass = end = 0;
            ch[0] = ch[1] = -1;
        }
    };

    std::vector<Node> t;
    int sz = 0; // 集合中数的总个数

    BinaryTrie() {
        t.push_back(Node());
    }
    void clear() {
        t.clear();
        t.push_back(Node());
        sz = 0;
    }

    void insert(int x, int times = 1) {
        // 插入 x
        uint32_t v = (uint32_t)x;
        int u = 0;
        for (int b = W; b >= 0; --b) {
            int bit = (v >> b) & 1u;
            if (t[u].ch[bit] == -1) {
                t[u].ch[bit] = (int)t.size();
                t.push_back(Node());
            }
```

```
            u = t[u].ch[bit];
            t[u].pass += times;
        }
        t[u].end += times;
        sz += times;
    }

    int count(int x) {
        // x 的出现次数
        uint32_t v = (uint32_t)x;
        int u = 0;
        for (int b = W; b >= 0; --b) {
            int bit = (v >> b) & 1u;
            int nx = t[u].ch[bit];
            if (nx == -1) {
                return 0;
            }
            u = nx;
        }
        return t[u].end;
    }

    bool erase(int x, int times = 1) {
        // 删除 x
        if (times <= 0) {
            return 1;
        }
        if (count(x) < times) {
            return 0;
        }

        uint32_t v = (uint32_t)x;
        int u = 0;
        for (int b = W; b >= 0; --b) {
            int bit = (v >> b) & 1u;
            u = t[u].ch[bit];
            t[u].pass -= times;
        }
        t[u].end -= times;
        sz -= times;

        return 1;
    }


    int max_xor(int x) {
        // 与 x 的最大异或值
        if (sz == 0) {
            return 0;
        }
        uint32_t v = (uint32_t)x, ans = 0;
        int u = 0;
        for (int b = W; b >= 0; --b) {
            int bit = (v >> b) & 1u;
```

```
            int want = bit ^ 1;
            int nx = t[u].ch[want];
            if (nx != -1 && t[nx].pass > 0) {
                ans |= (1u << b);
                u = nx;
            } else {
                u = t[u].ch[bit];
            }
        }
        return (int)ans;
    }

    int min_xor(int x) const {
        if (sz == 0) {
            return 0;
        }
        uint32_t v = (uint32_t)x, ans = 0;
        int u = 0;
        for (int b = W; b >= 0; --b) {
            int bit = (v >> b) & 1u;
            int same  = t[u].ch[bit];
            int other = t[u].ch[bit ^ 1];
            if (same != -1 && t[same].pass > 0) {
                u = same;
            } else {
                u = other;
                ans |= (1u << b);
            }
        }
        return (int)ans;
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int n;
    std::cin >> n;

    int max = 0;

    BinaryTrie tr;
    for (int i = 1; i <= n; i++) {
        int x;
        std::cin >> x;
        max = std::max(max, tr.max_xor(x));
        tr.insert(x);
    }

    std::cout << max << '\n';

    return 0;
}
```

6.2 exkmp.cpp

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

using i64 = long long;

struct Z_function {
    /*
    Z 函数
    z[i] 表示 s[i:] 与 s 的最长公共前缀长度
    Z_function z_str(str);
    */
    std::string p; // 模式串
    std::vector<int> z; // 失配数组

    explicit Z_function(const std::string &pat) {
        p = pat;
        z.assign(p.size(), 0);
        for (int i = 1, l = 0, r = 0; i < (int)p.size(); i++) {
            if (i <= r) {
                z[i] = std::min(r - i + 1, z[i - l]);
            }
            while (i + z[i] < (int)p.size() && p[z[i]] == p[i + z[i]]) {
                z[i]++;
            }
            if (i + z[i] - 1 > r) {
                l = i, r = i + z[i] - 1;
            }
        }
    }

    static std::vector<int> calc_z(const std::string &s) {
        int n = (int)s.size();
        std::vector<int> z(n);
        for (int i = 1, l = 0, r = 0; i < (int)s.size(); i++) {
            if (i <= r) {
                z[i] = std::min(r - i + 1, z[i - l]);
            }
            while (i + z[i] < (int)s.size() && s[z[i]] == s[i + z[i]]) {
                z[i]++;
            }
            if (i + z[i] - 1 > r) {
                l = i, r = i + z[i] - 1;
            }
        }
        return z;
    }

    std::vector<int> findAll(const std::string &text) {
        // 在 text 中查找所有 p 的出现
        std::vector<int> res;
        if (p.empty()) {
            return res;
        }
```

```

        std::string s = p;
        s.push_back('#');
        s += text;
        auto zz = calc_z(s);
        int m = (int)p.size();
        for (int i = m + 1; i < (int)s.size(); i++) {
            if (zz[i] >= m) {
                res.push_back(i - (m + 1));
            }
        }
        return res;
    }

    int findFirst(const std::string &text) {
        if (p.empty()) {
            return -1;
        }

        std::string s = p;
        s.push_back('#');
        s += text;
        auto zz = calc_z(s);
        int m = (int)p.size();
        for (int i = m + 1; i < (int)s.size(); i++) {
            if (zz[i] >= m) {
                return i - (m + 1);
            }
        }
        return -1;
    }

    int findCount(const std::string &text) {
        if (p.empty()) {
            return 0;
        }

        std::string s = p;
        s.push_back('#');
        s += text;
        auto zz = calc_z(s);
        int m = (int)p.size(), cnt = 0;
        for (int i = m + 1; i < (int)s.size(); i++) {
            if (zz[i] >= m) {
                cnt++;
            }
        }
        return cnt;
    }

    std::vector<int> borders() {
        std::vector<int> res;
        int n = (int)p.size();
        for (int i = 1; i < n; i++) {
```

```
            if (i + z[i] == n) {
                res.push_back(z[i]);
            }
        }
        std::sort(res.begin(), res.end());
        return res;
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    std::string a, b;
    std::cin >> a >> b;

    Z_function zb(b);
    int nb = b.size();
    i64 ans1 = 0;
    for (int i = 0; i < nb; i++) {
        if (i == 0) {
            ans1 ^= (nb + 1);
            continue;
        }
        ans1 ^= (i + 1) * 1ll * (zb.z[i] + 1);
    }

    std::string ab = b + "#" + a;
    Z_function zab(ab);
    int nab = ab.size();
    i64 ans2 = 0;
    for (int i = 1; i <= (int)a.size(); i++) {
        ans2 ^= i * 1ll * (zab.z[i + nb] + 1);
    }

    std::cout << ans1 << '\n' << ans2 << '\n';
    return 0;
}
```

6.3 hash.cpp

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

using i64 = long long;

int find(int n) {
    auto check = [&](int x) -> bool {
        if (x <= 1) {
            return 0;
        }
        for (int i = 2; i * i <= x; i++) {
            if (x % i == 0) {
                return 0;
```

```
            }
        }
        return 1;
    };

    while (!check(n)) {
        ++n;
    }
    return n;
}

static std::mt19937 rng((uint32_t)std::chrono::steady_clock::now().time_since_epoch().count());

static inline int rnd(int lo, int hi) {
    return (int)(rng() % (uint32_t)(hi - lo + 1)) + lo;
}

static inline int random_prime() {
    return find(rnd(100000000, 1000000000));
}

static inline int random_base(int P) {
    int lo = 256, hi = std::max(256, P - 2);
    return rnd(lo, hi);
}

using Hash = std::array<uint32_t, 2>;

struct StrHash {
    /*
    随机双模数字符串哈希
    StrHash h(s);
    auto hash = s.get(l, r);
    常数较大
    */
    static int P1, P2, B;

    int n;
    std::vector<int> h1, h2, p1, p2;

    StrHash() = default;

    explicit StrHash(const std::string &s) {
        n = s.size();
        h1.assign(n + 1, 0), h2.assign(n + 1, 0), p1.assign(n + 1, 0), p2.assign(n + 1, 0);
        if (P1 == 0 && P2 == 0 && B == 0) {
            P1 = random_prime();
            do {
                P2 = random_prime();
            } while (P1 == P2);
            B = random_base(std::min(P1, P2));
```

```
        }

        p1[0] = 1, p2[0] = 1;

        for (int i = 0; i < n; i++) {
            int x = (unsigned char)s[i] + 1;
            p1[i + 1] = (1ll * p1[i] * B) % P1;
            p2[i + 1] = (1ll * p2[i] * B) % P2;
            h1[i + 1] = (1ll * h1[i] * B + x) % P1;
            h2[i + 1] = (1ll * h2[i] * B + x) % P2;
        }
    }

    Hash get(int l, int r) {
        ++r;

        int a = (int)((h1[r] - (i64)h1[l] * p1[r - l]) % P1);
        if (a < 0) {
            a += P1;
        }

        int b = (int)((h2[r] - (i64)h2[l] * p2[r - l]) % P2);
        if (b < 0) {
            b += P2;
        }

        return {a, b};
    }
};

int StrHash::P1 = 0;
int StrHash::P2 = 0;
int StrHash::B = 0;

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int n;
    std::cin >> n;

    // std::vector<StrHash> h(n + 1);

    std::set<Hash> st;
    for (int i = 1; i <= n; i++) {
        std::string s;
        std::cin >> s;
        StrHash h(s);

        st.insert(h.get(0, (int)s.size() - 1));
    }

    std::cout << st.size() << "\n";

```

```
    return 0;
}
```

6.4 kmp.cpp

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

using i64 = long long;

struct KMP {
    /*
    KMP kmp(str); // 使用模式串初始化
    auto kmp.findAll(text) -> std::vector<int>; // 在 text 中找 所有 p 出现的位置
    */
    std::string p; // 模式串
    std::vector<int> pi; // 失配数组

    explicit KMP(const std::string &pat) {
        p = pat;
        pi.assign(p.size(), 0);
        for (int i = 1; i < (int)p.size(); i++) {
            int j = pi[i - 1];
            while (j > 0 && p[i] != p[j]) {
                j = pi[j - 1];
            }
            if (p[i] == p[j]) {
                j++;
            }
            pi[i] = j;
        }
    }

    std::vector<int> findAll(const std::string &text) {
        // 在 text 中找 所有 p 出现的位置
        std::vector<int> res;
        if (p.empty()) {
            return res;
        }

        int j = 0;
        for (int i = 0; i < (int)text.size(); i++) {
            while (j > 0 && text[i] != p[j]) {
                j = pi[j - 1];
            }
            if (text[i] == p[j]) {
                j++;
            }
            if (j == (int)p.size()) {
                res.push_back(i - (int)p.size() + 1);
                j = pi[j - 1];
            }
        }
```

```
        return res;
    }

    int findFirst(const std::string &text) {
        // 在 text 中找 p 第一次出现的位置
        if (p.empty()) {
            return -1;
        }

        int j = 0;
        for (int i = 0; i < (int)text.size(); i++) {
            while (j > 0 && text[i] != p[j]) {
                j = pi[j - 1];
            }
            if (text[i] == p[j]) {
                j++;
            }
            if (j == (int)p.size()) {
                return i - (int)p.size() + 1;
            }
        }
        return -1;
    }

    int findCount(const std::string &text) {
        // 在 text 中找 p 出现的次数
        if (p.empty()) {
            return 0;
        }

        int j = 0, res = 0;
        for (int i = 0; i < (int)text.size(); i++) {
            while (j > 0 && text[i] != p[j]) {
                j = pi[j - 1];
            }
            if (text[i] == p[j]) {
                j++;
            }
            if (j == (int)p.size()) {
                res++;
                j = pi[j - 1];
            }
        }
        return res;
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    std::string t, p;
    std::cin >> t >> p;

```

```
    KMP kmp(p);
    auto res = kmp.findAll(t);

    for (auto i : res) {
        std::cout << i << '\n';
    }

    for (auto l : kmp.pi) {
        std::cout << l << " ";
    }
    std::cout << '\n';

    return 0;
}
```

6.5 Trie.cpp

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

using i64 = long long;

struct Trie {
    /*
    字典树
    Trie tr;
    tr.insert(s, times=1)     插入
    tr.erase(s,  times=1)     删除
    tr.count_word(s)          单词 s 的出现次数
    tr.count_prefix(pre)      以 pre 为前缀的单词数量
    */
    static constexpr int A = 65;

    struct Node {
        int ch[A];
        int pass;   // 经过该节点的单词数
        int end;    // 在该节点结束的单词数
        Node() {
            pass = end = 0;
            std::fill(ch, ch + A, -1);
        }
    };

    std::vector<Node> t;

    Trie() {
        t.push_back(Node());
    }
    void clear() {
        t.clear();
        t.push_back(Node());
    }

    static inline int id(char c) {
```

```
        if ('0' <= c && c <= '9') {
            return c - '0';
        } else if ('a' <= c && c <= 'z') {
            return c - 'a' + 10;
        } else if ('A' <= c && c <= 'Z') {
            return c - 'A' + 36;
        } else {
            return -1;
        }
    }

    void insert(const std::string& s, int times = 1) {
        int u = 0;
        for (char c : s) {
            int k = id(c);
            if (t[u].ch[k] == -1) {
                t[u].ch[k] = (int)t.size();
                t.push_back(Node());
            }
            u = t[u].ch[k];
            t[u].pass += times;
        }
        t[u].end += times;
    }

    int count_word(const std::string& s) const {
        // 查找字符串集合中 s 的数量
        int u = 0;
        for (char c : s) {
            int k = id(c), nx = t[u].ch[k];
            if (nx == -1) return 0;
            u = nx;
        }
        return t[u].end;
    }

    int count_prefix(const std::string& pre) const {
        // 查找字符串集合中前缀 pre 的数量
        int u = 0;
        for (char c : pre) {
            int k = id(c), nx = t[u].ch[k];
            if (nx == -1) return 0;
            u = nx;
        }
        return t[u].pass;
    }

    bool erase(const std::string& s, int times = 1) {
        // 从字典树中删除 s
        if (times <= 0) {
            return 1;
        }
        if (count_word(s) < times) {
            return 0;
```

```
        }

        int u = 0;
        for (char c : s) {
            u = t[u].ch[id(c)];
            t[u].pass -= times;
        }
        t[u].end -= times;

        return 1;
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int t;
    std::cin >> t;

    while (t--) {
        int n, q;
        std::cin >> n >> q;

        Trie tr;
        for (int i = 1; i <= n; i++) {
            std::string s;
            std::cin >> s;
            tr.insert(s);
        }

        while (q--) {
            std::string t;
            std::cin >> t;
            std::cout << tr.count_prefix(t) << "\n";
        }
    }

    return 0;
}
```

7 tree

7.1 LCA.cpp

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

using i64 = long long;

struct HLD {
    /*
    使用树链剖分求 LCA, HLD tree(edge, 1) -> 初始化
```

```
    tree.lca(u, v) -> 求 u, v 的最近公共祖先
    */
    const std::vector<std::vector<int>> &edge;

    std::vector<int> fa, dep, son, sz, top;

    HLD(const std::vector<std::vector<int>> &edge_, int root = 1) : edge(edge_) {
        int n = (int)edge.size() - 1;
        fa.assign(n + 1, 0), dep.assign(n + 1, 0), son.assign(n + 1, 0), sz.assign(n + 1, 0), top.assign(n + 1, 0);

        dfs1(root, 0);
        dfs2(root, root);
    }

    int lca(int u, int v) const {
        while (top[u] != top[v]) {
            if (dep[top[u]] < dep[top[v]]) {
                std::swap(u, v);
            }
            u = fa[top[u]];
        }
        return dep[u] < dep[v] ? u : v;
    }

private:
    void dfs1(int u, int f) {
        fa[u] = f, dep[u] = dep[f] + 1, sz[u] = 1, son[u] = 0;
        for (auto v : edge[u]) {
            if (v == f) {
                continue;
            }
            dfs1(v, u);
            sz[u] += sz[v];
            if (sz[son[u]] < sz[v]) son[u] = v;
        }
    }

    void dfs2(int u, int t) {
        top[u] = t;
        if (son[u] == 0) {
            return;
        }
        dfs2(son[u], t);
        for (auto v : edge[u]) {
            if (v == fa[u] || v == son[u]) {
                continue;
            }
            dfs2(v, v);
        }
    }
};
```

```

void solve() {
  int n;
  std::cin >> n;

  std::vector<std::vector<int>> edge(n + 1);
  for (int i = 1; i < n; i++) {
    int u, v;
    std::cin >> u >> v;

    edge[u].push_back(v), edge[v].push_back(u);
  }

    HLD tree(edge, 1); // 使用邻接表和指定的根节点初始化 tree

    std::cout << tree.lca(1, 2) << '\n';

    // lca(u, v)
}

int main() {
  std::ios::sync_with_stdio(false);
  std::cin.tie(nullptr);

  int t = 1;
  std::cin >> t;
  while (t--) {
    solve();
  }

  return 0;
}
```