Competitive Programming Template

ethening

The Chinese University of Hong Kong

January 10, 2024

Contents

Start

Makefile

```
# ICPC
% : %.cpp
    g++ -Wall -Wfatal-errors -Wshadow -g -std=c++2a $< -o $@ -O2
```

Code Runner

```
g++ -g -std=c++20 -fsanitize=address,undefined -fno-sanitize-recover -fstack-protector-all -O2 $fileName -o
↪ $fileNameWithoutExt
```

Template

```cpp
#include "bits/stdc++.h"
using namespace std;
using ll = long long;
using pii = pair<int, int>;
using pil = pair<int, ll>;
using pli = pair<ll, int>;
using pll = pair<ll, ll>;

inline void yes() { cout  << "YES" << "\n"; return; }
inline void no() { cout << "NO" << "\n"; return; }

template<typename T>
bool chmin(T &x, T val) { if (val < x) { x = val; return true; } return false; }
template<typename T>
bool chmax(T &x, T val) { if (x < val) { x = val; return true; } return false; }

#define DEBUG 1
#define MULTI_TEST 1

void solve(int TC) {
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
}

int main() {
    cin.tie(0)->sync_with_stdio(0);
    cout << fixed << setprecision(9);
    int t = 1;
    if (MULTI_TEST) cin >> t;
    for (int _ = 1; _ <= t; _++) {
        solve(_);
    }
}
```

Pragma

```cpp
#pragma GCC optimize("Ofast")
#pragma GCC target("avx2")
```

Data Structure

Segment Tree

```cpp
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;

    SegmentTree() : n(0) {}
```

```
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(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());
        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 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;
        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;
        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);
    }
    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;
        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;
        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);
    }
};

constexpr ll inf = 1E18;

struct Info {
    ll cnt = 0;
    ll sum = 0;
    ll min = inf;
};

Info operator+(Info a, Info b) {
    Info c;
    c.cnt = a.cnt + b.cnt;
    c.sum = a.sum + b.sum;
    c.min = std::min(a.min, b.min);
    return c;
}
```

Lazy Segment Tree

```
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 {
    ll add = 0;

    void apply(Tag t) {
        add += t.add;
    }
};

struct Info {
    ll mn = 1E18;

    void apply(Tag t) {
        mn += t.add;
    }
};

Info operator+(Info a, Info b) {
    return {std::min(a.mn, b.mn)};
}
```

BIT

```
template <typename T>
struct BIT {
    int n;
    vector<T> a;

    BIT(int n = 0) {
        init(n);
    }

    void init(int n) {
```

```
        this->n = n;
        a.assign(n, T());
    }

    void add(int x, int v) {
        while (x < n) {
            a[x] += v;
            x += x & -x;
        }
    }

    T sum(int x) {
        auto ret = T();
        while (x > 0) {
            ret += a[x];
            x -= x & -x;
        }
        return ret;
    }

    T rangeSum(int l, int r) {
        if (l == 0) return sum(r);
        else return sum(r) - sum(l - 1);
    }

};
```

DSU

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

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

    void init(int n) {
        f.resize(n);
        iota(f.begin(), f.end(), 0);
        siz.assign(n, 1);
        rnk.assign(n, 0);
        cc = n;
    }

    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);
    }

    int merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) return -1;
        --cc;
        if (rnk[x] > rnk[y]) swap(x, y);
        siz[y] += siz[x];
        f[x] = y;
        if (rnk[x] == rnk[y]) rnk[y]++;
        return y;
    }

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

```
    }
};
```

KD Tree

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
    typedef Point P;
    T x, y;
    explicit Point(T x=0, T y=0) : x(x), y(y) {}
    bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
    bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
    P operator+(P p) const { return P(x+p.x, y+p.y); }
    P operator-(P p) const { return P(x-p.x, y-p.y); }
    P operator*(T d) const { return P(x*d, y*d); }
    P operator/(T d) const { return P(x/d, y/d); }
    T dot(P p) const { return x*p.x + y*p.y; }
    T cross(P p) const { return x*p.y - y*p.x; }
    T cross(P a, P b) const { return (a-*this).cross(b-*this); }
    T dist2() const { return x*x + y*y; }
    double dist() const { return sqrt((double)dist2()); }
    // angle to x-axis in interval [-pi, pi]
    double angle() const { return atan2(y, x); }
    P unit() const { return *this/dist(); } // makes dist()=1
    P perp() const { return P(-y, x); } // rotates +90 degrees
    P normal() const { return perp().unit(); }
    // returns point rotated 'a' radians ccw around the origin
    P rotate(double a) const {
        return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
    friend ostream& operator<<(ostream& os, P p) {
        return os << "(" << p.x << "," << p.y << ")"; }
};

using T = ll;
using P = Point<T>;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
    P pt; // if this is a leaf, the single point in it
    T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
    Node *first = 0, *second = 0;

    T distance(const P& p) { // min squared distance to a point
        T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
        T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
        return (P(x,y) - p).dist2();
    }

    Node(vector<P>&& vp) : pt(vp[0]) {
        for (P p : vp) {
            x0 = min(x0, p.x); x1 = max(x1, p.x);
            y0 = min(y0, p.y); y1 = max(y1, p.y);
        }
        if (vp.size() > 1) {
            // split on x if width >= height (not ideal...)
            sort(begin(vp), end(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
            // divide by taking half the array for each child (not
            // best performance with many duplicates in the middle)
            int half = (int)size(vp)/2;
            first = new Node({vp.begin(), vp.begin() + half});
            second = new Node({vp.begin() + half, vp.end()});
        }
    }
};

struct KDTree {
    Node* root;
```

```
    KDTree(const vector<P>& vp) : root(new Node({begin(vp), end(vp)})) {}

    pair<T, P> search(Node *node, const P& p) {
        if (!node->first) {
            // uncomment if we should not find the point itself:
            // if (p == node->pt) return {INF, P()};
            return make_pair((p - node->pt).dist2(), node->pt);
        }

        Node *f = node->first, *s = node->second;
        T bfirst = f->distance(p), bsec = s->distance(p);
        if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

        // search closest side first, other side if needed
        auto best = search(f, p);
        if (bsec < best.first)
            best = min(best, search(s, p));
        return best;
    }

    // find nearest point to a point, and its squared distance
    // (requires an arbitrary operator< for Point)
    pair<T, P> nearest(const P& p) {
        return search(root, p);
    }
};
```

Segment Tree Beats

```
#define REP(i, n) for (int i = 0; (i) < (int)(n); ++ (i))
#define REP3(i, m, n) for (int i = (m); (i) < (int)(n); ++ (i))
#define REP_R(i, n) for (int i = (int)(n) - 1; (i) >= 0; -- (i))
#define REP3R(i, m, n) for (int i = (int)(n) - 1; (i) >= (int)(m); -- (i))
#define ALL(x) std::begin(x), std::end(x)

/**
 * @brief a segment tree beats
 */
class segment_tree_beats {
    // MEMO: values for queries (max, min, lazy_add, and lazy_update) already apply to the current node; but not for children
    typedef struct {
        int64_t max;
        int64_t max_second;
        int max_count;
        int64_t min;
        int64_t min_second;
        int min_count;
        int64_t lazy_add;
        int64_t lazy_update;
        int64_t sum;
    } value_type;

    int n;
    std::vector<value_type> a;

public:
    segment_tree_beats() = default;
    segment_tree_beats(int n_) {
        n = 1; while (n < n_) n *= 2;
        a.resize(2 * n - 1);
        tag<UPDATE>(0, 0);
    }
    template <class InputIterator>
    segment_tree_beats(InputIterator first, InputIterator last) {
        int n_ = std::distance(first, last);
        n = 1; while (n < n_) n *= 2;
        a.resize(2 * n - 1);
        REP (i, n_) {
            tag<UPDATE>(n - 1 + i, *(first + i));
        }
```

```
        REP3 (i, n_, n) {
            tag<UPDATE>(n - 1 + i, 0);
        }
        REP_R (i, n - 1) {
            update(i);
        }
    }

    void range_chmin(int l, int r, int64_t value) {  // 0-based, [l, r)
        assert (0 <= l and l <= r and r <= n);
        range_apply<CHMIN>(0, 0, n, l, r, value);
    }
    void range_chmax(int l, int r, int64_t value) {  // 0-based, [l, r)
        assert (0 <= l and l <= r and r <= n);
        range_apply<CHMAX>(0, 0, n, l, r, value);
    }
    void range_add(int l, int r, int64_t value) {  // 0-based, [l, r)
        assert (0 <= l and l <= r and r <= n);
        range_apply<ADD>(0, 0, n, l, r, value);
    }
    void range_update(int l, int r, int64_t value) {  // 0-based, [l, r)
        assert (0 <= l and l <= r and r <= n);
        range_apply<UPDATE>(0, 0, n, l, r, value);
    }

    int64_t range_min(int l, int r) {  // 0-based, [l, r)
        assert (0 <= l and l <= r and r <= n);
        return range_get<MIN>(0, 0, n, l, r);
    }
    int64_t range_max(int l, int r) {  // 0-based, [l, r)
        assert (0 <= l and l <= r and r <= n);
        return range_get<MAX>(0, 0, n, l, r);
    }
    int64_t range_sum(int l, int r) {  // 0-based, [l, r)
        assert (0 <= l and l <= r and r <= n);
        return range_get<SUM>(0, 0, n, l, r);
    }

private:
    static constexpr char CHMIN = 0;
    static constexpr char CHMAX = 1;
    static constexpr char ADD = 2;
    static constexpr char UPDATE = 3;
    static constexpr char MIN = 10;
    static constexpr char MAX = 11;
    static constexpr char SUM = 12;

    template <char TYPE>
    void range_apply(int i, int il, int ir, int l, int r, int64_t g) {
        if (ir <= l or r <= il or break_condition<TYPE>(i, g)) {
            // break
        } else if (l <= il and ir <= r and tag_condition<TYPE>(i, g)) {
            tag<TYPE>(i, g);
        } else {
            pushdown(i);
            range_apply<TYPE>(2 * i + 1, il, (il + ir) / 2, l, r, g);
            range_apply<TYPE>(2 * i + 2, (il + ir) / 2, ir, l, r, g);
            update(i);
        }
    }
    template <char TYPE>
    inline bool break_condition(int i, int64_t g) {
        switch (TYPE) {
            case CHMIN: return a[i].max <= g;
            case CHMAX: return g <= a[i].min;
            case ADD: return false;
            case UPDATE: return false;
            default: assert (false);
        }
    }
    template <char TYPE>
```

```
    inline bool tag_condition(int i, int64_t g) {
        switch (TYPE) {
            case CHMIN: return a[i].max_second < g and g < a[i].max;
            case CHMAX: return a[i].min < g and g < a[i].min_second;
            case ADD: return true;
            case UPDATE: return true;
            default: assert (false);
        }
    }
    template <char TYPE>
    inline void tag(int i, int64_t g) {
        int length = n >> (32 - __builtin_clz(i + 1) - 1);
        if (TYPE == CHMIN) {
            if (a[i].max == a[i].min or g <= a[i].min) {
                tag<UPDATE>(i, g);
                return;
            }
            if (a[i].max != INT64_MIN) {
                a[i].sum -= a[i].max * a[i].max_count;
            }
            a[i].max = g;
            a[i].min_second = std::min(a[i].min_second, g);
            if (a[i].lazy_update != INT64_MAX) {
                a[i].lazy_update = std::min(a[i].lazy_update, g);
            }
            a[i].sum += g * a[i].max_count;
        } else if (TYPE == CHMAX) {
            if (a[i].max == a[i].min or a[i].max <= g) {
                tag<UPDATE>(i, g);
                return;
            }
            if (a[i].min != INT64_MAX) {
                a[i].sum -= a[i].min * a[i].min_count;
            }
            a[i].min = g;
            a[i].max_second = std::max(a[i].max_second, g);
            if (a[i].lazy_update != INT64_MAX) {
                a[i].lazy_update = std::max(a[i].lazy_update, g);
            }
            a[i].sum += g * a[i].min_count;
        } else if (TYPE == ADD) {
            if (a[i].max != INT64_MAX) {
                a[i].max += g;
            }
            if (a[i].max_second != INT64_MIN) {
                a[i].max_second += g;
            }
            if (a[i].min != INT64_MIN) {
                a[i].min += g;
            }
            if (a[i].min_second != INT64_MAX) {
                a[i].min_second += g;
            }
            a[i].lazy_add += g;
            if (a[i].lazy_update != INT64_MAX) {
                a[i].lazy_update += g;
            }
            a[i].sum += g * length;
        } else if (TYPE == UPDATE) {
            a[i].max = g;
            a[i].max_second = INT64_MIN;
            a[i].max_count = length;
            a[i].min = g;
            a[i].min_second = INT64_MAX;
            a[i].min_count = length;
            a[i].lazy_add = 0;
            a[i].lazy_update = INT64_MAX;
            a[i].sum = g * length;
        } else {
            assert (false);
        }
```

```
    }
    void pushdown(int i) {
        int l = 2 * i + 1;
        int r = 2 * i + 2;
        // update
        if (a[i].lazy_update != INT64_MAX) {
            tag<UPDATE>(l, a[i].lazy_update);
            tag<UPDATE>(r, a[i].lazy_update);
            a[i].lazy_update = INT64_MAX;
            return;
        }
        // add
        if (a[i].lazy_add != 0) {
            tag<ADD>(l, a[i].lazy_add);
            tag<ADD>(r, a[i].lazy_add);
            a[i].lazy_add = 0;
        }
        // chmin
        if (a[i].max < a[l].max) {
            tag<CHMIN>(l, a[i].max);
        }
        if (a[i].max < a[r].max) {
            tag<CHMIN>(r, a[i].max);
        }
        // chmax
        if (a[l].min < a[i].min) {
            tag<CHMAX>(l, a[i].min);
        }
        if (a[r].min < a[i].min) {
            tag<CHMAX>(r, a[i].min);
        }
    }
    void update(int i) {
        int l = 2 * i + 1;
        int r = 2 * i + 2;
        // chmin
        std::vector<int64_t> b { a[l].max, a[l].max_second, a[r].max, a[r].max_second };
        std::sort(b.rbegin(), b.rend());
        b.erase(std::unique(ALL(b)), b.end());
        a[i].max = b[0];
        a[i].max_second = b[1];
        a[i].max_count = (b[0] == a[l].max ? a[l].max_count : 0) + (b[0] == a[r].max ? a[r].max_count : 0);
        // chmax
        std::vector<int64_t> c { a[l].min, a[l].min_second, a[r].min, a[r].min_second };
        std::sort(ALL(c));
        c.erase(std::unique(ALL(c)), c.end());
        a[i].min = c[0];
        a[i].min_second = c[1];
        a[i].min_count = (c[0] == a[l].min ? a[l].min_count : 0) + (c[0] == a[r].min ? a[r].min_count : 0);
        // add
        a[i].lazy_add = 0;
        // update
        a[i].lazy_update = INT64_MAX;
        // sum
        a[i].sum = a[l].sum + a[r].sum;
    }

    template <char TYPE>
    int64_t range_get(int i, int il, int ir, int l, int r) {
        if (ir <= l or r <= il) {
            return 0;
        } else if (l <= il and ir <= r) {
            // base
            switch (TYPE) {
                case MIN: return a[i].min;
                case MAX: return a[i].max;
                case SUM: return a[i].sum;
                default: assert (false);
            }
        } else {
            pushdown(i);
```

```
            int64_t value_l = range_get<TYPE>(2 * i + 1, il, (il + ir) / 2, l, r);
            int64_t value_r = range_get<TYPE>(2 * i + 2, (il + ir) / 2, ir, l, r);
            // mult
            switch (TYPE) {
                case MIN: return std::min(value_l, value_r);
                case MAX: return std::max(value_l, value_r);
                case SUM: return value_l + value_r;
                default: assert (false);
            }
        }
    }
};

int main() {
    int n, q; scanf("%d%d", &n, &q);

    std::vector<long long> a(n);
    for (int i = 0; i < n; i++) {
        scanf("%lld", &a[i]);
    }
    segment_tree_beats beats(ALL(a));

    for (int ph = 0; ph < q; ph++) {
        int ty, l, r; scanf("%d%d%d", &ty, &l, &r);
        if (ty == 0) {
            long long b; scanf("%lld", &b);
            beats.range_chmin(l, r, b);
        } else if (ty == 1) {
            long long b; scanf("%lld", &b);
            beats.range_chmax(l, r, b);
        } else if (ty == 2) {
            long long b; scanf("%lld", &b);
            beats.range_add(l, r, b);
        } else {
            long long sum = beats.range_sum(l, r);
            printf("%lld\n", sum);
        }
    }
    return 0;
}
```

Link Cut Tree

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

using namespace std;

// modified from https://codeforces.com/blog/entry/75885
struct SplayTree {
  struct Node {
    int ch[2] = {0, 0}, p = 0;
    long long self = 0, path = 0;         // Path aggregates
    long long sub = 0, vir = 0;           // Subtree aggregates
    bool flip = 0;                        // Lazy tags
  };
  vector<Node> T;

  SplayTree(int n) : T(n + 1) {}

  void push(int x) {
    if (!x || !T[x].flip) return;
    int l = T[x].ch[0], r = T[x].ch[1];

    T[l].flip ^= 1, T[r].flip ^= 1;
    swap(T[x].ch[0], T[x].ch[1]);
    T[x].flip = 0;
  }

  void pull(int x) {
    int l = T[x].ch[0], r = T[x].ch[1]; push(l); push(r);

```

```
    T[x].path = T[l].path + T[x].self + T[r].path;
    T[x].sub = T[x].vir + T[l].sub + T[r].sub + T[x].self;
  }

  void set(int x, int d, int y) {
    T[x].ch[d] = y; T[y].p = x; pull(x);
  }

  void splay(int x) {
    auto dir = [&](int x) {
      int p = T[x].p; if (!p) return -1;
      return T[p].ch[0] == x ? 0 : T[p].ch[1] == x ? 1 : -1;
    };
    auto rotate = [&](int x) {
      int y = T[x].p, z = T[y].p, dx = dir(x), dy = dir(y);
      set(y, dx, T[x].ch[!dx]);
      set(x, !dx, y);
      if (~dy) set(z, dy, x);
      T[x].p = z;
    };
    for (push(x); ~dir(x); ) {
      int y = T[x].p, z = T[y].p;
      push(z); push(y); push(x);
      int dx = dir(x), dy = dir(y);
      if (~dy) rotate(dx != dy ? x : y);
      rotate(x);
    }
  }
};

struct LinkCut : SplayTree {
  LinkCut(int n) : SplayTree(n) {}

  int access(int x) {
    int u = x, v = 0;
    for (; u; v = u, u = T[u].p) {
      splay(u);
      int& ov = T[u].ch[1];
      T[u].vir += T[ov].sub;
      T[u].vir -= T[v].sub;
      ov = v; pull(u);
    }
    return splay(x), v;
  }

  void reroot(int x) {
    access(x); T[x].flip ^= 1; push(x);
  }

  int getroot(int x) {
    access(x);
    splay(x);
    while (T[x].ch[0]) x = T[x].ch[0];
    splay(x);
    return x;
  }

  void Link(int u, int v) {
    // check if originally not connected
    if (getroot(u) != getroot(v)) {
        reroot(u); access(v);
        T[v].vir += T[u].sub;
        T[u].p = v; pull(v);
    }
  }

  void Cut(int u, int v) {
    reroot(u); access(v);
    // check if there exist edge (u, v)
    if (T[v].ch[0] == u && !T[u].ch[1]) {
        T[v].ch[0] = T[u].p = 0; pull(v);
```

```
    }
  }

  // Rooted tree LCA. Returns 0 if u and v arent connected.
  int LCA(int u, int v) {
    if (u == v) return u;
    access(u); int ret = access(v);
    return T[u].p ? ret : 0;
  }

  // Query subtree of u where v is outside the subtree.
  long long Subtree(int u, int v) {
    reroot(v); access(u); return T[u].vir + T[u].self;
  }

  // Query path [u..v]
  long long Path(int u, int v) {
    reroot(u); access(v); return T[v].path;
  }

  // Update vertex u with value v
  void Update(int u, long long v) {
    access(u); T[u].self = v; pull(u);
  }
};

const int N = 2e5 + 11;
int a[N];
int32_t main(){
    int n, q; cin >> n >> q;
    for(int i = 1; i <= n; i++) cin >> a[i];

    LinkCut LCT(n + 1);
    for(int i = 1; i <= n; i++){
        LCT.Update(i, a[i]);
    }
    for(int i = 0; i < n - 1; i++){
        int u, v; cin >> u >> v;
        LCT.Link(++u, ++v);
    }
    for(int i = 0; i < q; i++){
        int type; cin >> type;
        if(type == 0){
            int u, v, w, x; cin >> u >> v >> w >> x;
            LCT.Cut(++u, ++v); LCT.Link(++w, ++x);
        }else if(type == 1){
            int p, x; cin >> p >> x;
            a[++p] += x;
            LCT.Update(p, a[p]);
        }else{
            int u, v; cin >> u >> v;
            ++u, ++v;
            cout << LCT.Path(u, v) << '\n';
        }
    }
}
```

Math

Mint

```
template<class T>
constexpr T power(T a, ll b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) res *= a;
    }
    return res;
}

```

```cpp
constexpr ll mul(ll a, ll b, ll p) {
    ll res = a * b - ll(1.L * a * b / p) * p;
    res %= p;
    if (res < 0) res += p;
    return res;
}

template<ll P>
struct MLong {
    ll x;
    constexpr MLong() : x{} {}
    constexpr MLong(ll x) : x{norm(x % getMod())} {}

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

    constexpr ll val() const { return x;}
    explicit constexpr operator ll() 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) { ll 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<>
ll MLong<0LL>::Mod = 1;

template<int P>
struct MInt {
    int x;
    constexpr MInt() : x{} {}
    constexpr MInt(ll 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) { ll 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 = 1;

template<int V, int P>
constexpr MInt<P> CInv = MInt<P>(V).inv();

constexpr int P = 1000000007;
using Z = MInt<P>;
```

Combin

```
template<class T>
constexpr T power(T a, ll b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) res *= a;
    }
    return res;
}

constexpr ll mul(ll a, ll b, ll p) {
    ll res = a * b - ll(1.L * a * b / p) * p;
    res %= p;
    if (res < 0) res += p;
    return res;
}

template<int P>
struct MInt {
    int x;
    constexpr MInt() : x{} {}
    constexpr MInt(ll 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) { ll 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 = 1;

template<int V, int P>
constexpr MInt<P> CInv = MInt<P>(V).inv();

constexpr int P = 1000000007;
using Z = MInt<P>;

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) {
        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);
    }
    Z catalan(int n) {
        return binom(2 * n, n) * inv(n + 1);
    }
    Z lucas(int n, int m) {
        if (n % P < m % P) return 0;
        if (n < P) return binom(n, m);
        return 1ll * lucas(n / P, m / P) * binom(n % P, m % P);
    }
} comb;
```

Sieve

```
std::vector<int> minp, primes;

void sieve(int n) {
    minp.assign(n + 1, 0);
    primes.clear();

    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            minp[i] = i;
            primes.push_back(i);
        }

        for (auto p : primes) {
            if (i * p > n) {
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                break;
            }
        }
    }
}
```

Polynomial

```
std::vector<int> rev;
template<int P>
std::vector<MInt<P>> roots{0, 1};

template<int P>
constexpr MInt<P> findPrimitiveRoot() {
    MInt<P> i = 2;
    int k = __builtin_ctz(P - 1);
    while (true) {
        if (power(i, (P - 1) / 2) != 1) {
```

```
            break;
        }
        i += 1;
    }
    return power(i, (P - 1) >> k);
}

template<int P>
constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();

template<>
constexpr MInt<998244353> primitiveRoot<998244353> {31};

template<int P>
constexpr void dft(std::vector<MInt<P>> &a) {
    int n = a.size();

    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; i++) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
        }
    }

    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            std::swap(a[i], a[rev[i]]);
        }
    }
    if (roots<P>.size() < n) {
        int k = __builtin_ctz(roots<P>.size());
        roots<P>.resize(n);
        while ((1 << k) < n) {
            auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1));
            for (int i = 1 << (k - 1); i < (1 << k); i++) {
                roots<P>[2 * i] = roots<P>[i];
                roots<P>[2 * i + 1] = roots<P>[i] * e;
            }
            k++;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                MInt<P> u = a[i + j];
                MInt<P> v = a[i + j + k] * roots<P>[k + j];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}

template<int P>
constexpr void idft(std::vector<MInt<P>> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    MInt<P> inv = (1 - P) / n;
    for (int i = 0; i < n; i++) {
        a[i] *= inv;
    }
}

template<int P = 998244353>
struct Poly : public std::vector<MInt<P>> {
    using Value = MInt<P>;

    Poly() : std::vector<Value>() {}
    explicit constexpr Poly(int n) : std::vector<Value>(n) {}
```

```
    explicit constexpr Poly(const std::vector<Value> &a) : std::vector<Value>(a) {}
    constexpr Poly(const std::initializer_list<Value> &a) : std::vector<Value>(a) {}

    template<class InputIt, class = std::_RequireInputIter<InputIt>>
    explicit constexpr Poly(InputIt first, InputIt last) : std::vector<Value>(first, last) {}

    template<class F>
    explicit constexpr Poly(int n, F f) : std::vector<Value>(n) {
        for (int i = 0; i < n; i++) {
            (*this)[i] = f(i);
        }
    }

    constexpr Poly shift(int k) const {
        if (k >= 0) {
            auto b = *this;
            b.insert(b.begin(), k, 0);
            return b;
        } else if (this->size() <= -k) {
            return Poly();
        } else {
            return Poly(this->begin() + (-k), this->end());
        }
    }
    constexpr Poly trunc(int k) const {
        Poly f = *this;
        f.resize(k);
        return f;
    }
    constexpr friend Poly operator+(const Poly &a, const Poly &b) {
        Poly res(std::max(a.size(), b.size()));
        for (int i = 0; i < a.size(); i++) {
            res[i] += a[i];
        }
        for (int i = 0; i < b.size(); i++) {
            res[i] += b[i];
        }
        return res;
    }
    constexpr friend Poly operator-(const Poly &a, const Poly &b) {
        Poly res(std::max(a.size(), b.size()));
        for (int i = 0; i < a.size(); i++) {
            res[i] += a[i];
        }
        for (int i = 0; i < b.size(); i++) {
            res[i] -= b[i];
        }
        return res;
    }
    constexpr friend Poly operator-(const Poly &a) {
        std::vector<Value> res(a.size());
        for (int i = 0; i < int(res.size()); i++) {
            res[i] = -a[i];
        }
        return Poly(res);
    }
    constexpr friend Poly operator*(Poly a, Poly b) {
        if (a.size() == 0 || b.size() == 0) {
            return Poly();
        }
        if (a.size() < b.size()) {
            std::swap(a, b);
        }
        int n = 1, tot = a.size() + b.size() - 1;
        while (n < tot) {
            n *= 2;
        }
        if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {
            Poly c(a.size() + b.size() - 1);
            for (int i = 0; i < a.size(); i++) {
```

```
                for (int j = 0; j < b.size(); j++) {
                    c[i + j] += a[i] * b[j];
                }
            }
            return c;
        }
        a.resize(n);
        b.resize(n);
        dft(a);
        dft(b);
        for (int i = 0; i < n; ++i) {
            a[i] *= b[i];
        }
        idft(a);
        a.resize(tot);
        return a;
    }
    constexpr friend Poly operator*(Value a, Poly b) {
        for (int i = 0; i < int(b.size()); i++) {
            b[i] *= a;
        }
        return b;
    }
    constexpr friend Poly operator*(Poly a, Value b) {
        for (int i = 0; i < int(a.size()); i++) {
            a[i] *= b;
        }
        return a;
    }
    constexpr friend Poly operator/(Poly a, Value b) {
        for (int i = 0; i < int(a.size()); i++) {
            a[i] /= b;
        }
        return a;
    }
    constexpr Poly &operator+=(Poly b) {
        return (*this) = (*this) + b;
    }
    constexpr Poly &operator-=(Poly b) {
        return (*this) = (*this) - b;
    }
    constexpr Poly &operator*=(Poly b) {
        return (*this) = (*this) * b;
    }
    constexpr Poly &operator*=(Value b) {
        return (*this) = (*this) * b;
    }
    constexpr Poly &operator/=(Value b) {
        return (*this) = (*this) / b;
    }
    constexpr Poly deriv() const {
        if (this->empty()) {
            return Poly();
        }
        Poly res(this->size() - 1);
        for (int i = 0; i < this->size() - 1; ++i) {
            res[i] = (i + 1) * (*this)[i + 1];
        }
        return res;
    }
    constexpr Poly integr() const {
        Poly res(this->size() + 1);
        for (int i = 0; i < this->size(); ++i) {
            res[i + 1] = (*this)[i] / (i + 1);
        }
        return res;
    }
    constexpr Poly inv(int m) const {
        Poly x{(*this)[0].inv()};
        int k = 1;
        while (k < m) {
```

```
            k *= 2;
            x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
        }
        return x.trunc(m);
    }
    constexpr Poly log(int m) const {
        return (deriv() * inv(m)).integr().trunc(m);
    }
    constexpr Poly exp(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
        }
        return x.trunc(m);
    }
    constexpr Poly pow(int k, int m) const {
        int i = 0;
        while (i < this->size() && (*this)[i] == 0) {
            i++;
        }
        if (i == this->size() || 1LL * i * k >= m) {
            return Poly(m);
        }
        Value v = (*this)[i];
        auto f = shift(-i) * v.inv();
        return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k);
    }
    constexpr Poly sqrt(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
        }
        return x.trunc(m);
    }
    constexpr Poly mulT(Poly b) const {
        if (b.size() == 0) {
            return Poly();
        }
        int n = b.size();
        std::reverse(b.begin(), b.end());
        return ((*this) * b).shift(-(n - 1));
    }
    std::vector<Value> eval(std::vector<Value> x) const {
        if (this->size() == 0) {
            return std::vector<Value>(x.size(), 0);
        }
        const int n = std::max(x.size(), this->size());
        std::vector<Poly> q(4 * n);
        std::vector<Value> ans(x.size());
        x.resize(n);
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                q[p] = Poly{1, -x[l]};
            } else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                q[p] = q[2 * p] * q[2 * p + 1];
            }
        };
        build(1, 0, n);
        std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly &num) {
            if (r - l == 1) {
                if (l < int(ans.size())) {
                    ans[l] = num[0];
                }
            } else {
```

```
                int m = (l + r) / 2;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).resize(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).resize(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
};

template<int P = 998244353>
Poly<P> berlekampMassey(const Poly<P> &s) {
    Poly<P> c;
    Poly<P> oldC;
    int f = -1;
    for (int i = 0; i < s.size(); i++) {
        auto delta = s[i];
        for (int j = 1; j <= c.size(); j++) {
            delta -= c[j - 1] * s[i - j];
        }
        if (delta == 0) {
            continue;
        }
        if (f == -1) {
            c.resize(i + 1);
            f = i;
        } else {
            auto d = oldC;
            d *= -1;
            d.insert(d.begin(), 1);
            MInt<P> df1 = 0;
            for (int j = 1; j <= d.size(); j++) {
                df1 += d[j - 1] * s[f + 1 - j];
            }
            assert(df1 != 0);
            auto coef = delta / df1;
            d *= coef;
            Poly<P> zeros(i - f - 1);
            zeros.insert(zeros.end(), d.begin(), d.end());
            d = zeros;
            auto temp = c;
            c += d;
            if (i - temp.size() > f - oldC.size()) {
                oldC = temp;
                f = i;
            }
        }
    }
    c *= -1;
    c.insert(c.begin(), 1);
    return c;
}


template<int P = 998244353>
MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
    int m = q.size() - 1;
    while (n > 0) {
        auto newq = q;
        for (int i = 1; i <= m; i += 2) {
            newq[i] *= -1;
        }
        auto newp = p * newq;
        newq = q * newq;
        for (int i = 0; i < m; i++) {
            p[i] = newp[i * 2 + n % 2];
        }
        for (int i = 0; i <= m; i++) {
            q[i] = newq[i * 2];
        }
        n /= 2;
```

```
    }
    return p[0] / q[0];
}
```

Graph

SCC

```
struct SCC {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> stk;
    std::vector<int> dfn, low, bel;
    int cur, cnt;

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

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        bel.assign(n, -1);
        stk.clear();
        cur = cnt = 0;
    }

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

    void dfs(int x) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);

        for (auto y : adj[x]) {
            if (dfn[y] == -1) {
                dfs(y);
                low[x] = std::min(low[x], low[y]);
            } else if (bel[y] == -1) {
                low[x] = std::min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
            cnt++;
        }
    }

    std::vector<int> work() {
        for (int i = 0; i < n; i++) {
            if (dfn[i] == -1) {
                dfs(i);
            }
        }
        return bel;
    }
};
```

Centroid Decomposition

```cpp
void solve() {
    int n, T;
    cin >> n >> T;

    vector<vector<array<int, 2>>> g(n);

    for (int i = 1; i < n; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        u--, v--;
        g[u].push_back({v, w});
        g[v].push_back({u, w});
    }

    int q;
    cin >> q;

    vector<vector<array<int, 3>>> qry(n);
    for (int i = 0; i < q; i++) {
        int a, b;
        cin >> a >> b;
        a--, b--;
        qry[0].push_back({a, b, i});
    }
    vector<ll> ans(q, inf);

    vector<bool> vis(n);
    vector<int> siz(n), bel(n);

    auto dfs = [&](auto self, int x, int p) -> void {
        siz[x] = 1;
        for (auto [y, w] : g[x]) {
            if (y == p) {
                continue;
            }
            self(self, y, x);
            siz[x] += siz[y];
        }
    };
    dfs(dfs, 0, -1);

    vector<ll> dep(n);
    auto solve = [&](auto &&self, int r) -> void {
        auto Q = std::move(qry[r]);

        auto find = [&](auto self, int x, int p, int s) -> int {
            for (auto [y, _] : g[x]) {
                if (y == p || vis[y] || 2 * siz[y] <= s) {
                    continue;
                }
                return self(self, y, x, s);
            }
            return x;
        };
        r = find(find, r, -1, siz[r]);
        vis[r] = true;

        auto dfs = [&](auto self, int x, int p) -> void {
            siz[x] = 1;
            for (auto [y, w] : g[x]) {
                if (y == p || vis[y]) {
                    continue;
                }
                dep[y] = dep[x] + w;
                bel[y] = x == r ? y : bel[x];
                self(self, y, x);
                siz[x] += siz[y];
            }
        };
```

```
        dfs(dfs, r, -1);

        for (auto [a, b, i] : Q) {
            ans[i] = min(ans[i], dep[a] + dep[b]);
            if (bel[a] == bel[b]) {
                qry[bel[a]].push_back({a, b, i});
            }
        }
        for (auto [y, _] : g[r]) {
            if (!vis[y]) {
                self(self, y);
            }
        }
    };
    solve(solve, 0);

    for (int i = 0; i < q; i++) {
        cout << ans[i] << "\n";
    }
}
```

Heavy Light Decomposition

```
struct HLD {
    int n;
    std::vector<int> siz, top, dep, parent, in, out, seq;
    std::vector<std::vector<int>> adj;
    int cur;

    HLD() {}
    HLD(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        siz.resize(n);
        top.resize(n);
        dep.resize(n);
        parent.resize(n);
        in.resize(n);
        out.resize(n);
        seq.resize(n);
        cur = 0;
        adj.assign(n, {});
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void work(int root = 0) {
        top[root] = root;
        dep[root] = 0;
        parent[root] = -1;
        dfs1(root);
        dfs2(root);
    }
    void dfs1(int u) {
        if (parent[u] != -1) {
            adj[u].erase(std::find(adj[u].begin(), adj[u].end(), parent[u]));
        }

        siz[u] = 1;
        for (auto &v : adj[u]) {
            parent[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v);
            siz[u] += siz[v];
            if (siz[v] > siz[adj[u][0]]) {
                std::swap(v, adj[u][0]);
            }
        }
```

```
    }
    void dfs2(int u) {
        in[u] = cur++;
        seq[in[u]] = u;
        for (auto v : adj[u]) {
            top[v] = v == adj[u][0] ? top[u] : v;
            dfs2(v);
        }
        out[u] = cur;
    }
    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;

        while (dep[top[u]] > d) {
            u = parent[top[u]];
        }

        return seq[in[u] - dep[u] + d];
    }

    bool isAncester(int u, int v) {
        return in[u] <= in[v] && in[v] < out[u];
    }

    int rootedParent(int u, int v) {
        std::swap(u, v);
        if (u == v) {
            return u;
        }
        if (!isAncester(u, v)) {
            return parent[u];
        }
        auto it = std::upper_bound(adj[u].begin(), adj[u].end(), v, [&](int x, int y) {
            return in[x] < in[y];
        }) - 1;
        return *it;
    }

    int rootedSize(int u, int v) {
        if (u == v) {
            return n;
        }
        if (!isAncester(v, u)) {
            return siz[v];
        }
        return n - siz[rootedParent(u, v)];
    }

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

Block Cut Tree

```cpp
struct BlockCutTree {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> dfn, low, stk;
    int cnt, cur;
    std::vector<std::pair<int, int>> edges;

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

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        stk.clear();
        cnt = cur = 0;
        edges.clear();
    }

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

    void dfs(int x) {
        stk.push_back(x);
        dfn[x] = low[x] = cur++;

        for (auto y : adj[x]) {
            if (dfn[y] == -1) {
                dfs(y);
                low[x] = std::min(low[x], low[y]);
                if (low[y] == dfn[x]) {
                    int v;
                    do {
                        v = stk.back();
                        stk.pop_back();
                        edges.emplace_back(n + cnt, v);
                    } while (v != y);
                    edges.emplace_back(x, n + cnt);
                    cnt++;
                }
            } else {
                low[x] = std::min(low[x], dfn[y]);
            }
        }
    }

    std::pair<int, std::vector<std::pair<int, int>>> work() {
        for (int i = 0; i < n; i++) {
            if (dfn[i] == -1) {
                stk.clear();
                dfs(i);
            }
        }
        return {cnt, edges};
    }
};
```

DSU On Tree

```cpp
struct FreqBuckets {
    vector<int> occ;
    vector<int> freq;
    FreqBuckets(int n, int maxC) : occ(maxC + 1, 0), freq(n + 1) { }

```

```
    void add(int x, int mul) {
        if (mul == +1) {
            occ[x]++;
            freq[occ[x]]++;
        }
        else if (mul == -1) {
            freq[occ[x]]--;
            occ[x]--;
        }
        else assert(false);
    }
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    int n,m; cin >> n >> m;

    vector<int> c(n);
    for (int i = 0; i < n; i++)
        cin >> c[i];

    vector<vector<int>> g(n);
    for (int i = 0; i + 1 < n; i++) {
        int u,v; cin >> u >> v; u--; v--;
        g[u].emplace_back(v);
        g[v].emplace_back(u);
    }

    vector<int> sz(n, 1);
    function<void(int, int)> dfs_hld = [&](int u, int p) {
        if (p != -1) {
            auto it = find(g[u].begin(), g[u].end(), p);
            assert(it != g[u].end());
            g[u].erase(it);
        }

        for (auto& v : g[u]) {
            dfs_hld(v, u);
            sz[u] += sz[v];
            if (sz[v] > sz[g[u][0]])
                swap(v, g[u][0]);
        }
    };
    dfs_hld(0, -1);

    vector<vector<pair<int, int>>> qry(n);
    for (int i = 0; i < m; i++) {
        int v,k; cin >> v >> k; v--;
        qry[v].emplace_back(k, i);
    }

    const int maxC = 100000;
    FreqBuckets cnt(n, maxC);
    vector<int> ans(m, -1);

    function<void(int, int)> dfs_addonly = [&](int u, int mul) {
        cnt.add(c[u], mul);
        for (auto& v : g[u])
            dfs_addonly(v, mul);
    };

    function<void(int)> dfs_solve = [&](int u) {
        for (auto& v : g[u]) {
            if (v == g[u][0]) continue;
            dfs_solve(v);
            dfs_addonly(v, -1);
        }

        if (!g[u].empty())
```

```
            dfs_solve(g[u][0]);
        cnt.add(c[u], +1);
        for (auto& v : g[u]) {
            if (v == g[u][0]) continue;
            dfs_addonly(v, +1);
        }

        for (auto& [k, i] : qry[u])
            ans[i] = (k <= n ? cnt.freq[k] : 0);
    };
    dfs_solve(0);

    for (int i = 0; i < m; i++)
        cout << ans[i] << "\n";

    return 0;
}
```

Edge Biconnected Component

```
using i64 = long long;

std::set<std::pair<int, int>> E;
struct EBCC {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> stk;
    std::vector<int> dfn, low, bel;
    int cur, cnt;

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

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        bel.assign(n, -1);
        stk.clear();
        cur = cnt = 0;
    }

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

    void dfs(int x, int p) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);

        for (auto y : adj[x]) {
            if (y == p) {
                continue;
            }
            if (dfn[y] == -1) {
                E.emplace(x, y);
                dfs(y, x);
                low[x] = std::min(low[x], low[y]);
            } else if (bel[y] == -1 && dfn[y] < dfn[x]) {
                E.emplace(x, y);
                low[x] = std::min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            do {
```

```
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
            cnt++;
        }
    }

    std::vector<int> work() {
        dfs(0, -1);
        return bel;
    }

    struct Graph {
        int n;
        std::vector<std::pair<int, int>> edges;
        std::vector<int> siz;
        std::vector<int> cnte;
    };
    Graph compress() {
        Graph g;
        g.n = cnt;
        g.siz.resize(cnt);
        g.cnte.resize(cnt);
        for (int i = 0; i < n; i++) {
            g.siz[bel[i]]++;
            for (auto j : adj[i]) {
                if (bel[i] < bel[j]) {
                    g.edges.emplace_back(bel[i], bel[j]);
                } else if (i < j) {
                    g.cnte[bel[i]]++;
                }
            }
        }
        return g;
    }
};
```

Two Sat

```
struct TwoSat {
    int n;
    std::vector<std::vector<int>> e;
    std::vector<bool> ans;
    TwoSat(int n) : n(n), e(2 * n), ans(n) {}
    void addClause(int u, bool f, int v, bool g) {
        e[2 * u + !f].push_back(2 * v + g);
        e[2 * v + !g].push_back(2 * u + f);
    }
    bool satisfiable() {
        std::vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
        std::vector<int> stk;
        int now = 0, cnt = 0;
        std::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] = std::min(low[u], low[v]);
                } else if (id[v] == -1) {
                    low[u] = std::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;
    }
    std::vector<bool> answer() { return ans; }
};

```

Flow

MaxFlow

```
constexpr int inf = 1E9;

template<class T>
struct MaxFlow {
    struct _Edge {
        int to;
        T cap;
        _Edge(int to, T cap) : to(to), cap(cap) {}
    };

    int n;
    std::vector<_Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<int> cur, h;

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

    void init(int n) {
        this->n = n;
        e.clear();
        g.assign(n, {});
        cur.resize(n);
        h.resize(n);
    }

    bool bfs(int s, int t) {
        h.assign(n, -1);
        std::queue<int> que;
        h[s] = 0;
        que.push(s);
        while (!que.empty()) {
            const int u = que.front();
            que.pop();
            for (int i : g[u]) {
                auto [v, c] = e[i];
                if (c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t) {
                        return true;
                    }
                    que.push(v);
                }
            }
        }
        return false;
    }

    T dfs(int u, int t, T f) {
        if (u == t) {
```

```
            return f;
        }
        auto r = f;
        for (int &i = cur[u]; i < int(g[u].size()); ++i) {
            const int j = g[u][i];
            auto [v, c] = e[j];
            if (c > 0 && h[v] == h[u] + 1) {
                auto a = dfs(v, t, std::min(r, c));
                e[j].cap -= a;
                e[j ^ 1].cap += a;
                r -= a;
                if (r == 0) {
                    return f;
                }
            }
        }
        return f - r;
    }
    void addEdge(int u, int v, T c) {
        g[u].push_back(e.size());
        e.emplace_back(v, c);
        g[v].push_back(e.size());
        e.emplace_back(u, 0);
    }
    T flow(int s, int t) {
        T ans = 0;
        while (bfs(s, t)) {
            cur.assign(n, 0);
            ans += dfs(s, t, std::numeric_limits<T>::max());
        }
        return ans;
    }

    std::vector<bool> minCut() {
        std::vector<bool> c(n);
        for (int i = 0; i < n; i++) {
            c[i] = (h[i] != -1);
        }
        return c;
    }

    struct Edge {
        int from;
        int to;
        T cap;
        T flow;
    };
    std::vector<Edge> edges() {
        std::vector<Edge> a;
        for (int i = 0; i < e.size(); i += 2) {
            Edge x;
            x.from = e[i + 1].to;
            x.to = e[i].to;
            x.cap = e[i].cap + e[i + 1].cap;
            x.flow = e[i + 1].cap;
            a.push_back(x);
        }
        return a;
    }
};
```

Min Cost Flow

```
constexpr int inf = 1E9;

template<class T>
struct MinCostFlow {
    struct _Edge {
        int to;
        T cap;
        T cost;
```

```
        _Edge(int to_, T cap_, T cost_) : to(to_), cap(cap_), cost(cost_) {}
    };
    int n;
    std::vector<_Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<T> h, dis;
    std::vector<int> pre;
    bool dijkstra(int s, int t) {
        dis.assign(n, std::numeric_limits<T>::max());
        pre.assign(n, -1);
        std::priority_queue<std::pair<T, int>, std::vector<std::pair<T, int>>, std::greater<std::pair<T, int>>> que;
        dis[s] = 0;
        que.emplace(0, s);
        while (!que.empty()) {
            T d = que.top().first;
            int u = que.top().second;
            que.pop();
            if (dis[u] != d) {
                continue;
            }
            for (int i : g[u]) {
                int v = e[i].to;
                T cap = e[i].cap;
                T cost = e[i].cost;
                if (cap > 0 && dis[v] > d + h[u] - h[v] + cost) {
                    dis[v] = d + h[u] - h[v] + cost;
                    pre[v] = i;
                    que.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != std::numeric_limits<T>::max();
    }
    MinCostFlow() {}
    MinCostFlow(int n_) {
        init(n_);
    }
    void init(int n_) {
        n = n_;
        e.clear();
        g.assign(n, {});
    }
    void addEdge(int u, int v, T cap, T cost) {
        g[u].push_back(e.size());
        e.emplace_back(v, cap, cost);
        g[v].push_back(e.size());
        e.emplace_back(u, 0, -cost);
    }
    std::pair<T, T> flow(int s, int t) {
        T flow = 0;
        T cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) {
                h[i] += dis[i];
            }
            T aug = std::numeric_limits<int>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                aug = std::min(aug, e[pre[i]].cap);
            }
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                e[pre[i]].cap -= aug;
                e[pre[i] ^ 1].cap += aug;
            }
            flow += aug;
            cost += aug * h[t];
        }
        return std::make_pair(flow, cost);
    }
    struct Edge {
        int from;
```

```
        int to;
        T cap;
        T cost;
        T flow;
    };
    std::vector<Edge> edges() {
        std::vector<Edge> a;
        for (int i = 0; i < e.size(); i += 2) {
            Edge x;
            x.from = e[i + 1].to;
            x.to = e[i].to;
            x.cap = e[i].cap + e[i + 1].cap;
            x.cost = e[i].cost;
            x.flow = e[i + 1].cap;
            a.push_back(x);
        }
        return a;
    }
};
```

String

KMP

```
int main(){
    string s;
    cin >> s;
    int n = (int)s.size();
    s = " " + s;

    vector<int> nxt(n + 1);
    for (int i = 2, p = 0; i <= n; i++) {
        while (p && s[p + 1] != s[i]) p = nxt[p];
        p = nxt[i] = p + (s[p + 1] == s[i]);
    }


    string t;
    cin >> t;
    int m = (int)t.size();
    t = " " + t;
    for (int i = 1, p = 0; i <= m; i++) {
        while (p && s[p + 1] != t[i]) p = nxt[p];
        p += (s[p + 1] == t[i]);
        if (p == n) {
            cout << i - n + 1 << "\n";
            p = nxt[p];
        }
    }

    for (int i = 1; i <= n; i++) {
        cout << nxt[i] << "\n";
    }

    return 0;
}
```

Trie

```
struct Trie {
    const int S = 26;
    int N;

    int cnt = 0; // root = 0

    vector<vector<int>> tr;
    vector<int> pos;

    Trie() : N(0) {}
```

```
    Trie(int n) : N(2 * n + 5), tr(N, vector<int>(S)), pos(N) {}

    void insert(string s, int id) {
        int p = 0;
        for (char it: s) {
            if (!tr[p][it - 'a']) tr[p][it - 'a'] = ++cnt;
            p = tr[p][it - 'a'];
        }
    }
};
```

GSAM

```
struct GSAM {
    const int S = 26;
    int N;

    vector<vector<int>> son;
    vector<int> len, fa;

    int cnt = 1; // root = 1

    GSAM() : N(0) {}
    GSAM(int n) : N(2 * n + 5), son(N, vector<int>(S)), len(N), fa(N) {}

    int insert(int p, int it) {
        int cur = ++cnt;
        len[cur] = len[p] + 1;
        while (!son[p][it]) son[p][it] = cur, p = fa[p];
        if (!p) return fa[cur] = 1, cur;
        int q = son[p][it];
        if (len[p] + 1 == len[q]) return fa[cur] = q, cur;
        int cl = ++cnt;
        son[cl] = son[q];
        len[cl] = len[p] + 1, fa[cl] = fa[q], fa[q] = fa[cur] = cl;
        while (son[p][it] == q) son[p][it] = cl, p = fa[p];
        return cur;
    }

    void build(Trie &T) {
        queue<int> Q;
        Q.push(0), T.pos[0] = 1;
        while (!Q.empty()) {
            int cur = Q.front();
            Q.pop();
            for (int i = 0; i < S; i++) {
                int p = T.tr[cur][i];
                if (!p) continue;
                T.pos[p] = insert(T.pos[cur], i), Q.push(p);
            }
        }
    }
};
```

ACA

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

using i64 = long long;

struct AhoCorasick {
    static constexpr int ALPHABET = 26;
    struct Node {
        int len;
        int link;
        std::array<int, ALPHABET> next;
        Node() : link{}, next{} {}
    };

    std::vector<Node> t;
```

```
15
16      AhoCorasick() {
17          init();
18      }
19
20      void init() {
21          t.assign(2, Node());
22          t[0].next.fill(1);
23          t[0].len = -1;
24      }
25
26      int newNode() {
27          t.emplace_back();
28          return t.size() - 1;
29      }
30
31      int add(const std::vector<int> &a) {
32          int p = 1;
33          for (auto x : a) {
34              if (t[p].next[x] == 0) {
35                  t[p].next[x] = newNode();
36                  t[t[p].next[x]].len = t[p].len + 1;
37              }
38              p = t[p].next[x];
39          }
40          return p;
41      }
42
43      int add(const std::string &a, char offset = 'a') {
44          std::vector<int> b(a.size());
45          for (int i = 0; i < a.size(); i++) {
46              b[i] = a[i] - offset;
47          }
48          return add(b);
49      }
50
51      void work() {
52          std::queue<int> q;
53          q.push(1);
54
55          while (!q.empty()) {
56              int x = q.front();
57              q.pop();
58
59              for (int i = 0; i < ALPHABET; i++) {
60                  if (t[x].next[i] == 0) {
61                      t[x].next[i] = t[t[x].link].next[i];
62                  } else {
63                      t[t[x].next[i]].link = t[t[x].link].next[i];
64                      q.push(t[x].next[i]);
65                  }
66              }
67          }
68      }
69
70      int next(int p, int x) {
71          return t[p].next[x];
72      }
73
74      int next(int p, char c, char offset = 'a') {
75          return next(p, c - 'a');
76      }
77
78      int link(int p) {
79          return t[p].link;
80      }
81
82      int len(int p) {
83          return t[p].len;
84      }
85
```

```
    int size() {
        return t.size();
    }
};
```

Manacher

```
using i64 = long long;
std::vector<int> manacher(std::string s) {
    std::string t = "#";
    for (auto c : s) {
        t += c;
        t += '#';
    }
    int n = t.size();
    std::vector<int> r(n);
    for (int i = 0, j = 0; i < n; i++) {
        if (2 * j - i >= 0 && j + r[j] > i) {
            r[i] = std::min(r[2 * j - i], j + r[j] - i);
        }
        while (i - r[i] >= 0 && i + r[i] < n && t[i - r[i]] == t[i + r[i]]) {
            r[i] += 1;
        }
        if (i + r[i] > j + r[j]) {
            j = i;
        }
    }
    return r;
}
```

SAM

```
struct SuffixAutomaton {
    static constexpr int ALPHABET_SIZE = 26, N = 1e6;
    struct Node {
        int len;
        int link;
        int next[ALPHABET_SIZE];
        Node() : len(0), link(0), next{} {}
    } t[2 * N];
    int cntNodes;
    SuffixAutomaton() {
        cntNodes = 1;
        std::fill(t[0].next, t[0].next + ALPHABET_SIZE, 1);
        t[0].len = -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 r = ++cntNodes;
            t[r].len = t[p].len + 1;
            t[r].link = t[q].link;
            std::copy(t[q].next, t[q].next + ALPHABET_SIZE, t[r].next);
            t[q].link = r;
            while (t[p].next[c] == q) {
                t[p].next[c] = r;
                p = t[p].link;
            }
            return r;
        }
        int cur = ++cntNodes;
        t[cur].len = t[p].len + 1;
        while (!t[p].next[c]) {
            t[p].next[c] = cur;
            p = t[p].link;
        }
        t[cur].link = extend(p, c);
        return cur;
```

```
    }
};
```

Suffix Array

```

#include <bits/stdc++.h>

using i64 = long long;
struct SuffixArray {
    int n;
    std::vector<int> sa, rk, lc;
    SuffixArray(const std::string &s) {
        n = s.length();
        sa.resize(n);
        lc.resize(n - 1);
        rk.resize(n);
        std::iota(sa.begin(), sa.end(), 0);
        std::sort(sa.begin(), sa.end(), [&](int a, int b) {return s[a] < s[b];});
        rk[sa[0]] = 0;
        for (int i = 1; i < n; ++i)
            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
        int k = 1;
        std::vector<int> tmp, cnt(n);
        tmp.reserve(n);
        while (rk[sa[n - 1]] < n - 1) {
            tmp.clear();
            for (int i = 0; i < k; ++i)
                tmp.push_back(n - k + i);
            for (auto i : sa)
                if (i >= k)
                    tmp.push_back(i - k);
            std::fill(cnt.begin(), cnt.end(), 0);
            for (int i = 0; i < n; ++i)
                ++cnt[rk[i]];
            for (int i = 1; i < n; ++i)
                cnt[i] += cnt[i - 1];
            for (int i = n - 1; i >= 0; --i)
                sa[--cnt[rk[tmp[i]]]] = tmp[i];
            std::swap(rk, tmp);
            rk[sa[0]] = 0;
            for (int i = 1; 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]);
            k *= 2;
        }
        for (int i = 0, j = 0; i < n; ++i) {
            if (rk[i] == 0) {
                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;
                lc[rk[i] - 1] = j;
            }
        }
    }
};
```

Geometry

Geometry (Jiangly)

```
using i64 = long long;
template<class T>
struct Point {
    T x;
    T y;
    Point(T x_ = 0, T y_ = 0) : x(x_), y(y_) {}

```

```
    template<class U>
    operator Point<U>() {
        return Point<U>(U(x), U(y));
    }
    Point &operator+=(Point p) & {
        x += p.x;
        y += p.y;
        return *this;
    }
    Point &operator-=(Point p) & {
        x -= p.x;
        y -= p.y;
        return *this;
    }
    Point &operator*=(T v) & {
        x *= v;
        y *= v;
        return *this;
    }
    Point &operator/=(T v) & {
        x /= v;
        y /= v;
        return *this;
    }
    Point operator-() const {
        return Point(-x, -y);
    }
    friend Point operator+(Point a, Point b) {
        return a += b;
    }
    friend Point operator-(Point a, Point b) {
        return a -= b;
    }
    friend Point operator*(Point a, T b) {
        return a *= b;
    }
    friend Point operator/(Point a, T b) {
        return a /= b;
    }
    friend Point operator*(T a, Point b) {
        return b *= a;
    }
    friend bool operator==(Point a, Point b) {
        return a.x == b.x && a.y == b.y;
    }
    friend std::istream &operator>>(std::istream &is, Point &p) {
        return is >> p.x >> p.y;
    }
    friend std::ostream &operator<<(std::ostream &os, Point p) {
        return os << "(" << p.x << ", " << p.y << ")";
    }
};

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 square(Point<T> p) {
    return dot(p, p);
}

template<class T>
double length(Point<T> p) {
    return std::sqrt(double(square(p)));
```

```cpp
}

long double length(Point<long double> p) {
    return std::sqrt(square(p));
}

template<class T>
Point<T> normalize(Point<T> p) {
    return p / length(p);
}

template<class T>
struct Line {
    Point<T> a;
    Point<T> b;
    Line(Point<T> a_ = Point<T>(), Point<T> b_ = Point<T>()) : a(a_), b(b_) {}
};

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

template<class T>
int sgn(Point<T> a) {
    return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
}

template<class T>
bool pointOnLineLeft(Point<T> p, Line<T> l) {
    return cross(l.b - l.a, p - l.a) > 0;
}

template<class T>
Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
    return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1.a - l1.b));
}

template<class T>
bool pointOnSegment(Point<T> p, Line<T> l) {
    return cross(p - l.a, l.b - l.a) == 0 && std::min(l.a.x, l.b.x) <= p.x && p.x <= std::max(l.a.x, l.b.x)
        && std::min(l.a.y, l.b.y) <= p.y && p.y <= std::max(l.a.y, l.b.y);
}

template<class T>
bool pointInPolygon(Point<T> a, std::vector<Point<T>> p) {
    int n = p.size();
    for (int i = 0; i < n; i++) {
        if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
            return true;
        }
    }

    int t = 0;
    for (int i = 0; i < n; i++) {
        auto u = p[i];
        auto v = p[(i + 1) % n];
        if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
            t ^= 1;
        }
        if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
            t ^= 1;
        }
    }

    return t == 1;
}

// 0 : not intersect
// 1 : strictly intersect
// 2 : overlap
```

```cpp
// 3 : intersect at endpoint
template<class T>
std::tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T> l2) {
    if (std::max(l1.a.x, l1.b.x) < std::min(l2.a.x, l2.b.x)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (std::min(l1.a.x, l1.b.x) > std::max(l2.a.x, l2.b.x)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (std::max(l1.a.y, l1.b.y) < std::min(l2.a.y, l2.b.y)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (std::min(l1.a.y, l1.b.y) > std::max(l2.a.y, l2.b.y)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
        if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
            return {0, Point<T>(), Point<T>()};
        } else {
            auto maxx1 = std::max(l1.a.x, l1.b.x);
            auto minx1 = std::min(l1.a.x, l1.b.x);
            auto maxy1 = std::max(l1.a.y, l1.b.y);
            auto miny1 = std::min(l1.a.y, l1.b.y);
            auto maxx2 = std::max(l2.a.x, l2.b.x);
            auto minx2 = std::min(l2.a.x, l2.b.x);
            auto maxy2 = std::max(l2.a.y, l2.b.y);
            auto miny2 = std::min(l2.a.y, l2.b.y);
            Point<T> p1(std::max(minx1, minx2), std::max(miny1, miny2));
            Point<T> p2(std::min(maxx1, maxx2), std::min(maxy1, maxy2));
            if (!pointOnSegment(p1, l1)) {
                std::swap(p1.y, p2.y);
            }
            if (p1 == p2) {
                return {3, p1, p2};
            } else {
                return {2, p1, p2};
            }
        }
    }
    auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
    auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
    auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
    auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);

    if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) || (cp3 < 0 && cp4 < 0)) {
        return {0, Point<T>(), Point<T>()};
    }

    Point p = lineIntersection(l1, l2);
    if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
        return {1, p, p};
    } else {
        return {3, p, p};
    }
}

template<class T>
bool segmentInPolygon(Line<T> l, std::vector<Point<T>> p) {
    int n = p.size();
    if (!pointInPolygon(l.a, p)) {
        return false;
    }
    if (!pointInPolygon(l.b, p)) {
        return false;
    }
    for (int i = 0; i < n; i++) {
        auto u = p[i];
        auto v = p[(i + 1) % n];
        auto w = p[(i + 2) % n];
        auto [t, p1, p2] = segmentIntersection(l, Line(u, v));

```

```
        if (t == 1) {
            return false;
        }
        if (t == 0) {
            continue;
        }
        if (t == 2) {
            if (pointOnSegment(v, l) && v != l.a && v != l.b) {
                if (cross(v - u, w - v) > 0) {
                    return false;
                }
            }
        } else {
            if (p1 != u && p1 != v) {
                if (pointOnLineLeft(l.a, Line(v, u))
                    || pointOnLineLeft(l.b, Line(v, u))) {
                    return false;
                }
            } else if (p1 == v) {
                if (l.a == v) {
                    if (pointOnLineLeft(u, l)) {
                        if (pointOnLineLeft(w, l)
                            && pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    } else {
                        if (pointOnLineLeft(w, l)
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    }
                } else if (l.b == v) {
                    if (pointOnLineLeft(u, Line(l.b, l.a))) {
                        if (pointOnLineLeft(w, Line(l.b, l.a))
                            && pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    } else {
                        if (pointOnLineLeft(w, Line(l.b, l.a))
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    }
                } else {
                    if (pointOnLineLeft(u, l)) {
                        if (pointOnLineLeft(w, Line(l.b, l.a))
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    } else {
                        if (pointOnLineLeft(w, l)
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    }
                }
            }
        }
    }
    return true;
}

template<class T>
std::vector<Point<T>> hp(std::vector<Line<T>> lines) {
    std::sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
        auto d1 = l1.b - l1.a;
        auto d2 = l2.b - l2.a;

        if (sgn(d1) != sgn(d2)) {
            return sgn(d1) == 1;
        }
```

```
        return cross(d1, d2) > 0;
    });

    std::deque<Line<T>> ls;
    std::deque<Point<T>> ps;
    for (auto l : lines) {
        if (ls.empty()) {
            ls.push_back(l);
            continue;
        }

        while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
            ps.pop_back();
            ls.pop_back();
        }

        while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
            ps.pop_front();
            ls.pop_front();
        }

        if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
            if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {

                if (!pointOnLineLeft(ls.back().a, l)) {
                    assert(ls.size() == 1);
                    ls[0] = l;
                }
                continue;
            }
            return {};
        }

        ps.push_back(lineIntersection(ls.back(), l));
        ls.push_back(l);
    }

    while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
        ps.pop_back();
        ls.pop_back();
    }
    if (ls.size() <= 2) {
        return {};
    }
    ps.push_back(lineIntersection(ls[0], ls.back()));

    return std::vector(ps.begin(), ps.end());
}

using i128 = __int128;
using P = Point<i128>;
```

Minkowski Sum

```
template<class P>
vector<P> minkowski(vector<P> C1, vector<P> C2){
    auto reorder_polygon = [](vector<P> &pts){
        size_t pos = 0;
        for (size_t i = 1; i < pts.size(); i++){
            if(pts[i].y < pts[pos].y || (pts[i].y == pts[pos].y && pts[i].x < pts[pos].x))
                pos = i;
        }
        rotate(pts.begin(), pts.begin() + pos, pts.end());
    };
    reorder_polygon(C1);
    reorder_polygon(C2);
    C1.push_back(C1[0]);
    C1.push_back(C1[1]);
    C2.push_back(C2[0]);
    C2.push_back(C2[1]);
```

```
    vector<P> ret;
    size_t i = 0, j = 0;
    while (i < C1.size() - 2 || j < C2.size() - 2){
        ret.push_back(C1[i] + C2[j]);
        auto cross = (C1[i + 1] - C1[i]).cross(C2[j + 1] - C2[j]);
        if (cross >= 0 && i < C1.size() - 2)
            ++i;
        if (cross <= 0 && j < C2.size() - 2)
            ++j;
    }
    return ret;
}
```