Contents

Data structure

Ordered statistic tree

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds ;
using namespace std ;
typedef tree <int , null_type , less <int>, rb_tree_tag ,
tree_order_statistics_node_update> ordered_set ;

ordered_set myset;

int main() {
    myset.insert(1);
    myset.insert(4);
    myset.insert(5);
    // find the kth largest key 0-indexed
    cout << *myset.find_by_order(0) << endl;
    // order of the key, 0-index
    cout << myset.order_of_key(4) << endl;
    cout << *myset.find_by_order(1) << endl;
    cout << myset.order_of_key(6) << endl;
    // set size
    cout << myset.size() << endl;
    // find
    cout << (myset.find(4) != myset.end()) << endl;
    // remove
    myset.erase(4);
    tree <int , null_type , less <int>, rb_tree_tag ,
tree_order_statistics_node_update> :: iterator iter = myset.begin();
    while (iter != myset.end()) {
        cout << *iter << endl;
        iter++;
    }
    return 0;
}
```

Ordered statistic tree using range tree

```
#pragma GCC optimize(2)
#pragma GCC optimize(3)
#include <bits/stdc++.h>
```

```
#define LEFT (index << 1)
#define RIGHT (index << 1 | 1)
#define MID ((l+r)>>1)
#define MAX_SIZE 2097312
using namespace std;
int tree[MAX_SIZE];
int cnt[MAX_SIZE >> 1];
int n;

void build(int l, int r, int index) {
    if (l > r) return;
    if (l == r) {
        tree[index] = cnt[l];
        // cout << l << " " << cnt[l] << endl;
        return;
    }

    build(l, MID, LEFT);
    build(MID + 1, r, RIGHT);
    tree[index] = tree[LEFT] + tree[RIGHT];
}

void change(int pt, int l, int r, int index) {
    if (l > r) return;
    if (pt == l && l == r) {
        tree[index]++;
        return;
    }

    if (pt <= MID) {
        change(pt, l, MID, LEFT);
    } else {
        change(pt, MID + 1, r, RIGHT);
    }

    tree[index] = tree[LEFT] + tree[RIGHT];
}

void extractk(int l, int r, int index, int k) {
    if (l > r) return;
    // cout << "extract " << k << " " << l  << " " << r << endl;
    if (l == r) {
        tree[index]--;
        return;
    }

    if (tree[LEFT] >= k) {
        extractk(l, MID, LEFT, k);
    } else {
        extractk(MID + 1, r, RIGHT, k - tree[LEFT]);
    }
    tree[index] = tree[LEFT] + tree[RIGHT];
}

int get(int l, int r, int index) {
    if (l > r) return 0;
    if (l == r) {
        return l;
    }

    if (tree[LEFT] > 0) {
        return get(l, MID, LEFT);
    }
    return get(MID + 1, r, RIGHT);
}

int main() {
    int v, i;
    int q;
    scanf("%d", &n);
    scanf("%d", &q);
    for (i = 1; i <= n; ++i) {
        scanf("%d", &v);
        cnt[v]++;
    }

    build(1, n, 1);

    while (q > 0) {
        int v;
        scanf("%d", &v);
        if (v > 0) {
            change(v, 1, n, 1);
        } else {
            extractk(1, n, 1, -v);
        }
        --q;
    }

    if (tree[1] == 0) {
        printf("0\n");
    } else {
        printf("%d\n", get(1, n, 1));
    }
    return 0;
}
```

Dynamic range tree My implementation

```
struct dynamictree {
    struct node {
        int val;
        int l, r;
        struct node *left, *right;

        node(int l, int r, int val) {
            this->left = this->right = NULL;
            this->l = l, this->r = r, this->val = val;
        }
    };

    node *tree[MAX_SIZE];

    node *inc(node *rt, int pos, int l, int r, int val) {
        if (l > r || pos < l || pos > r) return rt;
        if (rt == NULL) {
            rt = new node(l, r, 0);
        }

        if (l == r && pos == r) {
            rt->val += val;
            return rt;
        }

        int mid = l + (r - l) / 2;
        if (pos <= mid) {
            rt->left = inc(rt->left, pos, l, mid, val);
        } else {
            rt->right = inc(rt->right, pos, mid + 1, r, val);
        }
        int v1 = rt->left == NULL ? 0 : rt->left->val;
        int v2 = rt->right == NULL ? 0 : rt->right->val;
        rt->val = v1 + v2;
        return rt;
    }

    int getans(node *rt, int l, int r) {
        if (rt == NULL) return 0;
        if (l <= rt->l && rt->r <= r) return rt->val;
        int mid = rt->l + (rt->r - rt->l) / 2;
        if (r <= mid) {
            return getans(rt->left, l, r);
        } else if (l >= mid + 1) {
            return getans(rt->right, l, r);
        } else {
            return getans(rt->left, l, r) + getans(rt->right, l, r);
        }
    }

    // increment pos by val in tree[version]
    void update(int pos, int val, int version) {
        tree[version] = inc(tree[version], pos, 0, 1000000000, val);
    }

    int query(int l, int r, int version) {
        return getans(tree[version], l, r);
    }
};

dynamictree tree;
```

Dynamic range tree Tom's implementation

```
#include <bits/stdc++.h>
#define MAXN (100005*100)
#define iii tuple<int, int, int>
#define lli long long int
using namespace std;

// range-update-range-query XOR tree with
// dynamic node creation

struct node {
    int s, e, v[2], lazy, p, c[2];
};

node tree[MAXN];
int used = 0;
int root;

int new_node(int s, int e, int p = 0, bool fill = false) {
    ++used;
    tree[used].s = s;
    tree[used].e = e;
    if (fill) {
        tree[used].v[0] = 0;
        tree[used].v[1] = (e-s+1);
    } else {
        tree[used].v[0] = (e-s+1);
        tree[used].v[1] = 0;
    }
    tree[used].lazy = 0;
    tree[used].p = p;
    tree[used].c[0] = 0;
    tree[used].c[1] = 0;
    return used;
}

void push(int at) {
    if (tree[at].s == tree[at].e) return;
    if (!tree[at].c[0] && !tree[at].c[1]) {
        assert(tree[at].v[0] == 0 || tree[at].v[1] == 0);
        int mid = (tree[at].s+tree[at].e)/2;
        tree[at].c[0] = new_node(tree[at].s, mid, at);
        tree[at].c[1] = new_node(mid+1, tree[at].e, at);
    }
    if (tree[at].lazy % 2 != 0) {
        swap(tree[tree[at].c[0]].v[0], tree[tree[at].c[0]].v[1]);
        swap(tree[tree[at].c[1]].v[0], tree[tree[at].c[1]].v[1]);
        tree[tree[at].c[0]].lazy += 1;
        tree[tree[at].c[1]].lazy += 1;
    }
    tree[at].lazy = 0;
}

void xor_range(int at, int s, int e) {
    if (s > tree[at].e || e < tree[at].s) return;
    push(at);
    if (s <= tree[at].s && e >= tree[at].e) {
        tree[at].lazy += 1;
        swap(tree[at].v[0], tree[at].v[1]);
        at = tree[at].p;
        while (at) {
            tree[at].v[0] = (
                tree[tree[at].c[0]].v[0] +
                tree[tree[at].c[1]].v[0]
            );
            tree[at].v[1] = (
                tree[tree[at].c[0]].v[1] +
                tree[tree[at].c[1]].v[1]
            );
            at = tree[at].p;
        }
    } else {
        xor_range(tree[at].c[0], s, e);
        xor_range(tree[at].c[1], s, e);
    }
}

void print_tree(int at, int level = 0) {
    for (int i = 0; i < level; i++) printf(" ");
    printf("#%d: [%d,%d] (%dx0,%dx1) ops = %d\n",
        at, tree[at].s, tree[at].e, tree[at].v[0], tree[at].v[1], tree[at].laz
y
    );
    if (tree[at].c[0]) print_tree(tree[at].c[0], level+1);
    if (tree[at].c[1]) print_tree(tree[at].c[1], level+1);
}
```

Range tree on arithmetic progression

```
#include <bits/stdc++.h>
#define MAX_SIZE 2097312
typedef long long ll;

using namespace std;

struct segt {
    ll lz, value;
};

segt tree[MAX_SIZE];
int n;


void pushdown(int l, int r, int index) {
    if (l >= r) return;
    if (tree[index].lz || tree[index].value) {
        int mid = l + (r - l) / 2;
        int len = mid - l + 1;
        tree[index * 2].lz += tree[index].lz;
        tree[index * 2 + 1].lz += tree[index].lz;
        tree[index * 2].value += tree[index].value;
        tree[index * 2 + 1].value += tree[index].value + tree[index].lz  * len
;
        tree[index].lz = 0;
        tree[index].value = 0;
    }
}

void update(int start, int end, int l, int r, int index, ll val) {
    if (start > end || l > r) return;
    int mid = l + (r - l) / 2;
    pushdown(l, r, index);
    if (start > r || l > end) return;
    if (start <= l && r <= end) {
        tree[index].value += val * (l - start + 1);
        tree[index].lz += val;
        return;
    }

    if (end <= mid) {
        update(start, end, l, mid, index * 2, val);
    } else if (start >= mid + 1) {
        update(start, end, mid + 1, r, index * 2 + 1, val);
    } else {
        update(start, end, l, mid, index * 2, val);
        update(start, end, mid + 1, r, index * 2 + 1, val);
    }
}

ll query(int pos, int l, int r, int index) {
    if (l > r || pos < l || pos > r) return 0;
    pushdown(l, r, index);
    int mid = l + (r - l) / 2;
    if (l == r && r == pos) return tree[index].value;
    if (pos <= mid) return query(pos, l, mid, index * 2);
    return query(pos, mid + 1, r, index * 2 + 1);
}
```

Range tree increase Fibonacci

```
#include <bits/stdc++.h>
#pragma GCC optimize(3)
#define MAX_SIZE 300011
#define MOD 1000000009
#define LEFT (index << 1)
#define RIGHT (index << 1 | 1)
typedef long long ll;
using namespace std;

/*
    Two properties are used in this problem
    F[n] = first * Fib[n-2] + second * Fib[n-1]
    S[n] = F[n+2] - second
*/
```

```
struct segt {
    // range sum
    ll rsum;
    // range first, second increment value
    ll lz1, lz2;
};

ll fib[MAX_SIZE];
int a[MAX_SIZE];
segt tree[MAX_SIZE << 2];

ll add(ll v1, ll v2) {
    return ((v1 + v2) % MOD + MOD) % MOD;
}

ll getn(ll first, ll second, int x) {
    if (x == 1) return first;
    if (x == 2) return second;
    return add(first * fib[x-2], second * fib[x-1]);
}

void pushdown(int l, int r, int index) {
    if (l > r) return;
    if (tree[index].lz1 || tree[index].lz2) {
        int mid = (l + r) >> 1;
        int len1 = mid - l + 1;
        int len2 = r - mid;
        if (l != r) {
            tree[LEFT].lz1 = add(tree[LEFT].lz1, tree[index].lz1);
            tree[LEFT].lz2 = add(tree[LEFT].lz2, tree[index].lz2);
            ll delta = getn(tree[index].lz1, tree[index].lz2, len1 + 2) - tree
[index].lz2;
            tree[LEFT].rsum = add(tree[LEFT].rsum, delta);
            ll v1 = getn(tree[index].lz1, tree[index].lz2, len1 + 1);
            ll v2 = getn(tree[index].lz1, tree[index].lz2, len1 + 2);
            tree[RIGHT].lz1 = add(tree[RIGHT].lz1, v1);
            tree[RIGHT].lz2 = add(tree[RIGHT].lz2, v2);
            delta = getn(v1, v2, len2 + 2) - v2;
            tree[RIGHT].rsum = add(tree[RIGHT].rsum, delta);
        }

        tree[index].lz1 = tree[index].lz2 = 0;
        // cout << l << " " << r << " = " << tree[index].rsum << endl;
    }
}

void build(int l, int r, int index) {
    if (l > r) return;
    if (l == r) {
        tree[index].rsum = a[l];
        return;
    }

    int mid = (l + r) >> 1;
    build(l, mid, LEFT);
    build(mid + 1, r, RIGHT);
    tree[index].rsum = add(tree[LEFT].rsum, tree[RIGHT].rsum);
}

void update(int start, int end, int l, int r, int index) {
    if (start > end || l > r) return;
    pushdown(l, r, index);
    if (start > r || l > end) return;
    if (start <= l && r <= end) {
        // TODO
        tree[index].lz1 = fib[l - start + 1];
        tree[index].lz2 = fib[l - start + 2];
        ll delta = getn(tree[index].lz1, tree[index].lz2, r - l + 3);
        tree[index].rsum = add(tree[index].rsum, delta - tree[index].lz2);
        return;
    }

    int mid = (l + r) >> 1;
    if (end <= mid) {
        update(start, end, l, mid, LEFT);
    } else if (start >= mid + 1) {
        update(start, end, mid + 1, r, RIGHT);
    } else {
        update(start, end, l, mid, LEFT);
        update(start, end, mid + 1, r, RIGHT);
    }
    tree[index].rsum = add(tree[LEFT].rsum, tree[RIGHT].rsum);
}

ll query(int start, int end, int l, int r, int index) {
    if (start > end || l > r || start > r || l > end) return 0;
    pushdown(l, r, index);
    if (start <= l && r <= end) {
        // cout << l << " " << r << " -> " << tree[index].rsum << endl;
        return tree[index].rsum;
    }
    int mid = (l + r) >> 1;
    if (end <= mid) {
        return query(start, end, l, mid, LEFT);
    }
    if (start >= mid + 1) {
        return query(start, end, mid + 1, r, RIGHT);
    }

    return add(query(start, end, l, mid, LEFT), query(start, end, mid + 1, r,
RIGHT));
}
```

Sqrt trick dynamic graph

```
#pragma GCC optimize(3)
#pragma GCC optimize(2)
#include <bits/stdc++.h>
#define MAX_SIZE 200011
using namespace std;

pair<int, int> edg[MAX_SIZE][2];
int f[MAX_SIZE];
int sz[MAX_SIZE];
vector<pair<int, int>> history;
// in the union-find relation, pending
set<pair<int, int>> in, pend, possible;
int tp[MAX_SIZE];
int ans[MAX_SIZE];
int N;
int find(int x, int cp=0) {
    if (f[x] == x) return x;
    if (cp) {
        return find(f[x], cp);
    }
    return f[x] = find(f[x], cp);
}

void cancel() {
    if (history.empty()) return;
    auto p = history.back();
    history.pop_back();
    f[p.first] = p.first;
    sz[p.second] -= sz[p.first];
}
```

```
void unionset(int x, int y, int tp=0) {
    int a = find(x, tp), b = find(y, tp);
    if (a != b) {
        if (sz[a] > sz[b]) swap(a, b);
        // join a, b
        sz[b] += sz[a];
        f[a] = b;
        if (tp) {
            history.emplace_back(a, b);
        }
    }
}

void init() {
    int i;
    for (i = 1 ; i <= N; ++i) {
        f[i] = i, sz[i] = 1;
    }
}

void reconstruct() {
    int i;
    for (i = 1 ; i <= N; ++i) {
        f[i] = i, sz[i] = 1;
    }

    for (auto &e : in) {
        unionset(e.first, e.second);
    }

    history.clear();
}

void addpend(pair<int, int> &e) {
    if (pend.find(e) != pend.end()) {
        pend.erase(e);
    } else {
        pend.insert(e);
    }
}

void addin(pair<int, int> &e) {
    if (in.find(e) != in.end()) {
        in.erase(e);
    } else {
        in.insert(e);
    }
}
```

```
int main() {
    int i, j, Q;
    scanf("%d%d", &N, &Q);
    init();
    int block_size = 5000;
    for (i = 1; i <= Q; ++i) {
        scanf("%d", &tp[i]);
        int x, y;
        scanf("%d%d", &x, &y);
        edg[i][0].first = (x - 1) % N + 1;
        edg[i][0].second = (y - 1) % N + 1;
        if (edg[i][0].first > edg[i][0].second) swap(edg[i][0].first, edg[i][0
].second);
        edg[i][1].first = x % N + 1;
        edg[i][1].second = y % N + 1;
        if (edg[i][1].first > edg[i][1].second) swap(edg[i][1].first, edg[i][1
].second);
    }

    for (i = 1; i <= Q; ++i) {
        int L = (i - 1) * block_size + 1, R = min(Q, block_size * i);
        if (L > Q || L > R) break;
        in.clear();
        pend.clear();
        possible.clear();
        for (j = L; j <= R; ++j) {
            if (tp[j] == 2) continue;
            possible.insert(make_pair(edg[j][0].first, edg[j][0].second));
            possible.insert(make_pair(edg[j][1].first, edg[j][1].second));
        }

        for (j = 1; j < L; ++j) {
            if (tp[j] == 2) continue;
            if (possible.find(edg[j][ans[j-1]]) == possible.end()) {
                addin(edg[j][ans[j-1]]);
            } else {
                addpend(edg[j][ans[j-1]]);
            }
        }
        reconstruct();
        // cout << "finish construct " << endl;
        for (j = L; j <= R; ++j) {
            if (tp[j] == 2) {
                while (!history.empty()) {
                    cancel();
                }
                for (auto &e : pend) {
                    // cout << "pending " << e.first << " " << e.second << end
l;
                    unionset(e.first, e.second, 1);
                }

                ans[j] = (find(edg[j][ans[j-
1]].first, 1) == find(edg[j][ans[j-1]].second, 1));
            } else {
                addpend(edg[j][ans[j-1]]);
                ans[j] = ans[j-1];
            }
        }
    }

    for (i = 1; i <= Q; ++i) {
        if (tp[i] == 2) {
            printf("%d", ans[i]);
        }
    }
    return 0;
}
```

Point model

Support operation: add point at (x, y), delete point at (x, y), get the point strictly on the right and up of (x, y)

```
#include <bits/stdc++.h>
#define MAX_SIZE 524312
#define ADD 0
#define REMOVE 1
#define FIND 2

using namespace std;

struct opt {
    int op, x, y;
};

set<int> pt[MAX_SIZE >> 1];
// stores the maximum y coordiate in the range
int tree[MAX_SIZE];
opt arr[MAX_SIZE >> 1];
vector<int> disc;
vector<int> tmp;
int q;
int n;
char s[10];

void build(int l, int r, int index) {
    if (l > r) return;
    if (l == r) {
        tree[index] = -1;
        return;
    }

    int mid = l + (r - l) / 2;
    build(l, mid, index * 2);
    build(mid + 1, r, index * 2 + 1);
    tree[index] = -1;
}

void update(int p, int l, int r, int index, int val, int tp) {
    if (l > r) return;
    if (l == r && p == l) {
        if (tp == ADD) {
            // cout << val << " " << tree[index] << endl;
            tree[index] = max(val, tree[index]);

        } else {
            // cout << "remove " << val << endl;
            pt[p].erase(val);
            if (pt[p].empty()) {
                tree[index] = -1;
            } else {
                tree[index] = *pt[p].rbegin();
            }
        }
        return;
    }

    int mid = l + (r - l) / 2;
    if (p <= mid) {
        update(p, l, mid, index * 2, val, tp);
    } else {
        update(p, mid + 1, r, index * 2 + 1, val, tp);
    }

    tree[index] = max(tree[index * 2], tree[index * 2 + 1]);
}

int query(int start, int end, int l, int r, int index, int val) {
    if (start > end || l > r || start > r || l > end) return MAX_SIZE;
    int mid = l + (r - l) / 2;
    if (start <= l && r <= end) {
        if (tree[index] <= val) return MAX_SIZE;
        if (l == r) return l;
        if (tree[index * 2] > val) {
            return query(start, end, l, mid, index * 2, val);
        }
        return query(start, end, mid + 1, r, index * 2 + 1, val);
    }

    return min(query(start, end, l, mid, index * 2, val), query(start, end, mi
d + 1, r, index * 2 + 1, val));
}

int main() {
    scanf("%d", &q);
    int i;
    for (i = 0 ; i < q; i++) {
        int x, y;
        scanf("%s%d%d", s, &x, &y);
        if (s[0] == 'r') {
            arr[i].op = REMOVE;
        } else if (s[0] == 'f') {
            arr[i].op = FIND;
        } else {
            arr[i].op = ADD;
        }

        arr[i].x = x;
        arr[i].y = y;
        tmp.push_back(arr[i].x);
    }

    sort(tmp.begin(), tmp.end());
    for (i = 0 ; i < (int) tmp.size(); i++) {
        if (i == 0 || tmp[i] != tmp[i-1]) {
            disc.push_back(tmp[i]);
        }
    }
    n = (int) disc.size();
    build(0, n - 1, 1);
    for (i = 0 ; i < q; i++) {
        int index;
        if (arr[i].op == ADD) {
            index = lower_bound(disc.begin(), disc.end(), arr[i].x) - disc.beg
in();
            // cout << arr[i].y << "*****" << (pt[index].empty() ? -
1 : *pt[index].rbegin()) << endl;
            if (pt[index].empty() || (*pt[index].rbegin() < arr[i].y)) {
                update(index, 0, n - 1, 1, arr[i].y, ADD);
            }
            pt[index].insert(arr[i].y);
        } else if (arr[i].op == REMOVE) {
            index = lower_bound(disc.begin(), disc.end(), arr[i].x) - disc.beg
in();
            update(index, 0, n - 1, 1, arr[i].y, REMOVE);
            // cout << pt[index].size() << endl;
        } else {
            vector<int> :: iterator iter = upper_bound(disc.begin(), disc.end(
), arr[i].x);
            if (iter == disc.end()) {
                printf("-1\n");
            } else {
                index = iter - disc.begin();
                int idk = query(index, n - 1, 0, n - 1, 1, arr[i].y);
                if (idk >= n) {
                    printf("-1\n");
                } else {
                    printf("%d %d\n", disc[idk], *pt[idk].upper_bound(arr[i].y
));
                }
            }
        }
    }
    return 0;
}
```

Range invert and range sum tree

```
typedef struct segTree {
    int value;
    // update information 1 for 1/0 for 0/-1 for up-to-date;
    int lazy;
    // invert information 1 for yes and 0 for no;
    int ivt;
} SegmentTree;

SegmentTree tree[MAX_SIZE * 8];

int n;

Long command[MAX_SIZE][3];

// set the lazy mark to be -1
void build(int l, int r, int index);
```

```
// a binary search in the segment tree
int minZero(int l, int r, int index);

void inverted(int start, int end, int l, int r, int index);

void lazyupdate(int l, int r, int index);

void update(int start, int end, int l, int r, int index, int v);

int main(int argc, char *argv[]) {
    int i, cnt = 0, m;
    mymap = unordered_map<Long, int>();
    cin >> m;
    // discreate the data into intervals, l, r and more importantly insert 1 i
nto data
    for (i = 0 ; i < m; i++) {
        cin >> command[i][0] >> command[i][1] >> command[i][2];
        // for the add command, insert l, r and r + 1
        order[cnt++] = command[i][1];
        order[cnt++] = ++command[i][2];
    }

    order[cnt++] = 1;
    sort(order, order + cnt);

    n = 0;
    for (i = 0 ; i < cnt; i++) {
        if (i == 0) {
            mymap[order[i]] = n;
            discrete[n++] = order[i];
        } else {
            if (order[i] != order[i-1]) {
                mymap[order[i]] = n;
                discrete[n++] = order[i];
            }
        }
    }
    /*
    for (i = 0 ; i < n; i++) {
        cout << discrete[i] << " ";
    }

    cout << endl;
    */
    build(0, n - 1, 1);

    for (i = 0 ; i < m; i++) {
        if (command[i][0] == 1) {
            update(mymap[command[i][1]], mymap[command[i][2]] - 1, 0, n - 1, 1
, 1);
        } else if (command[i][0] == 2) {
            update(mymap[command[i][1]], mymap[command[i][2]] - 1, 0, n - 1, 1
, 0);
        } else if (command[i][0] == 3) {
            inverted(mymap[command[i][1]], mymap[command[i][2]] - 1, 0, n - 1,
 1);
        }

        int index =  minZero(0, n - 1, 1);
        cout << discrete[index] << endl;
    }

    return 0;
}

void build(int l, int r, int index) {
    if (l > r) return;
    if (l == r) {
        tree[index].lazy = -1;
        return;
    }

    build(l, l + (r - l) / 2, index * 2);
    build(1 + l + (r - l) / 2, r, index * 2 + 1);
    tree[index].lazy = -1;
}


void lazyupdate(int l, int r, int index) {
    if (l != r) {
        int mid = l + (r - l) / 2;
        if (tree[index].lazy != -1) {
            tree[index * 2].lazy = tree[index * 2 + 1].lazy = tree[index].lazy
;
            tree[index * 2].value = (mid - l + 1) * tree[index].lazy;
            tree[index * 2 + 1].value = (r - mid) * tree[index].lazy;
            tree[index * 2].ivt = tree[index * 2 + 1].ivt = 0;
        }

        if (tree[index].ivt != 0) {
            tree[index * 2].ivt ^= 1;
            tree[index * 2 + 1].ivt ^= 1;
            tree[index * 2].value = (mid - l + 1) - tree[2 * index].value;
            tree[index * 2 + 1].value = (r - mid) - tree[2 * index + 1].value;
        }
    }
    tree[index].lazy = -1;
    tree[index].ivt = 0;
}

void update(int start, int end, int l, int r, int index, int v) {
    if (start > end || l > r) return;
    lazyupdate(l, r, index);
    if (start > r || end < l) return;
    if (start <= l && r <= end) {
        tree[index].value = (r - l + 1) * v;
        tree[index].lazy = v;
        tree[index].ivt = 0;
        return;
    }

    update(start, end, l, l + (r - l) / 2, index * 2, v);
    update(start, end, 1 + l + (r - l) / 2, r, index * 2 + 1, v);
    tree[index].value = tree[index * 2].value + tree[index * 2 + 1].value;
}

void inverted(int start, int end, int l, int r, int index) {
    if (start > end || l > r) return;
    lazyupdate(l, r, index);
    if (start > r || end < l) return;
    if (start <= l && r <= end) {
        tree[index].value = (r - l + 1) - tree[index].value;
        tree[index].ivt = 1 - tree[index].ivt;
        return;
    }

    inverted(start, end, l, l + (r - l) / 2, index * 2);
    inverted(start, end, 1 + l + (r - l) / 2, r, index * 2 + 1);
    tree[index].value = tree[index * 2].value + tree[index * 2 + 1].value;
}

int minZero(int l, int r, int index) {
    if (l > r) return 0;
    if (l == r) return l;
    lazyupdate(l, r, index);
    int mid = l + (r - l) / 2;
    if (tree[index * 2].value < (mid - l + 1)) {
        return minZero(l, mid, index * 2);
    }
    return minZero(mid + 1, r, index * 2 + 1);
}
```

Dsu on tree

```
multiset<int> st[MAX_SIZE];
vector<int> g[MAX_SIZE];
int deg[MAX_SIZE];
int dep[MAX_SIZE];
int f[MAX_SIZE];
int k;

void merge(int v1, int v2) {
    int d = dep[v2];
    int s1 = f[v1], s2 = f[v2];
    if (st[s1].size() < st[s2].size()) {
        // merge s1 to s2
        // condition for a merge is depth[s1_i]  + dep[s2_i] <= k + 2 * d
        // dep[s2_i] <= k + 2 * d - depth[s1_i]
        // we find the prev of upper_bound of k + 2 * d - depth[s1_i]
        // if the element exists we insert eliminate the element
        // and insert max(dep[s1_i], element) into s2
        // otherwise, we directly insert dep[s1_i] to s2
        for (auto v : st[s1]) {
            auto iter = st[s2].upper_bound(k + 2 * d - v);
            if (iter == st[s2].begin()) {
                st[s2].insert(v);
            } else {
                iter = prev(iter);
                int nv = max(*iter, v);
                st[s2].erase(iter);
                st[s2].insert(nv);
            }
        }
        f[v1] = s2;
    } else {
        // merge s2 to s1
        for (auto v : st[s2]) {
            auto iter = st[s1].upper_bound(k + 2 * d - v);
            if (iter == st[s1].begin()) {
                st[s1].insert(v);
            } else {
                iter = prev(iter);
                int nv = max(*iter, v);
                st[s1].erase(iter);
                st[s1].insert(nv);
            }
        }
        f[v2] = s1;
    }
}
```

```
void dfs(int v, int pre) {
    dep[v] = dep[pre] + 1;
    if (deg[v] == 1) {
        st[f[v]].insert(dep[v]);
        return;
    }

    for (auto nv : g[v]) {
        if (nv != pre) {
            dfs(nv, v);
        }
    }

    for (auto nv : g[v]) {
        if (nv != pre) {
            merge(nv, v);
        }
    }
}

int main() {
    int i, n;
    scanf("%d%d", &n, &k);
    REP (i,1,n-1) {
        f[i] = i;
```

Mo’s algorithm

```
#pragma GCC optimize(2)
#pragma GCC optimize(3)
#include <bits/stdc++.h>
typedef long long ll;
#define MAX_SIZE 262192
using namespace std;

int cnt[MAX_SIZE << 2], a[MAX_SIZE << 2];
ll ans[MAX_SIZE], sm = 0;
int l = 1, r = 0;
int block_size = 1;

struct qrs {
    int l, r, id;
    bool operator < (qrs other) const {
        int b1 = l / block_size, b2 = other.l / block_size;
        return (b1 ^ b2) ? b1 < b2 : ((b1 & 1) ? r < other.r : r > other.r);
    }
};

qrs q[MAX_SIZE];

inline void add(int idx) {
    // we want to add in a[idx]
    // current sum should increase by (cnt[a[idx]] + 1) ^2 - cnt[a[idx]] ^ 2
    // which is 2 * cnt[a[idx]] + 1 and cnt[a[idx]]++
    sm += 1ll * a[idx] * ((2ll * cnt[a[idx]]) + 1);
    cnt[a[idx]]++;
}

inline void del(int idx) {
    sm -= 1ll * a[idx] * ((2ll * cnt[a[idx]]) - 1);
    cnt[a[idx]]--;
}

int main() {
    int n, m, i;
    scanf("%d%d", &n, &m);
    for (i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
    }

    for (i = 0; i < m; i++) {
        scanf("%d%d", &q[i].l, &q[i].r);
        q[i].id = i;
    }
    block_size = sqrt(n);
    sort(q, q + m);

    for (i = 0 ; i < m; i++) {
        while (r < q[i].r) add(++r);
        while (r > q[i].r) del(r--);
        while (l > q[i].l) add(--l);
        while (l < q[i].l) del(l++);

        ans[q[i].id] = sm;
    }

    for (i = 0 ; i < m; i++) {
        printf("%lld\n", ans[i]);
    }
    return 0;
}
```

CDQ

```
#include <bits/stdc++.h>
#define MAX_SIZE 200111

using namespace std;

int tree[MAX_SIZE];
int n;

int shift(int x) {
    return x & (-x);
}

void update(int pos, int v) {
    if (pos < 1) return;
    while (pos <= n) {
        tree[pos] += v;
        pos += shift(pos);
    }
}

int query(int pos) {
    int ret = 0;
    while (pos > 0) {
        ret += tree[pos];
        pos -= shift(pos);
    }
    return ret;
}

struct qrs {
    // x, y, v, id
    int x, y, v, id, si;
};

qrs vc[MAX_SIZE << 3];
int ans[MAX_SIZE];
qrs tmp[MAX_SIZE << 3];
int cnt = 0;
// we need to support 2 types of queries
// increment(x, y) by 1, increment(x, y) by -1
// and query(1, 1, x, y)
// good news is the timestamp has already be sorted
// in ascending order, we don't need to worry about that
void cdq(int l, int r) {
    if (l >= r) return;
    int mid = (l + r) >> 1;
    cdq(l, mid);
    cdq(mid + 1, r);
    vector<pair<int, int>> hist;
    int i = l, j = mid + 1, k = 0;
    while (i <= mid && j <= r) {
        if (vc[i].x <= vc[j].x) {
            update(vc[i].y, vc[i].v);
            hist.emplace_back(vc[i].y, -vc[i].v);
            tmp[k++] = vc[i++];
        } else {
            ans[vc[j].id] += vc[j].si * query(vc[j].y);
            tmp[k++] = vc[j++];
        }
    }

    while (i <= mid) {
        tmp[k++] = vc[i++];
    }

    while (j <= r) {
        ans[vc[j].id] += vc[j].si * query(vc[j].y);
        tmp[k++] = vc[j++];
    }

    for (i = l; i <= r; ++i) {
        vc[i] = tmp[i-l];
    }

    for (auto h : hist) {
        update(h.first, h.second);
    }
}
```

2D BIT

```
int shift(int x) {
    return x & (-x);
}

void update(int x, int y, int val) {
    if (x <= 0 || y <= 0) return;
    int i, j;
    for (i = x; i <= n; i += shift(i)) {
        for (j = y; j <= m; j += shift(j)) {
            tree[i][j] += val;
        }
    }
}

ll query(int x, int y) {
    ll ret = 0;
    int i, j;
    for (i = x; i > 0; i -= shift(i)) {
        for (j = y; j > 0; j -= shift(j)) {
            ret += tree[i][j];
        }
    }

    return ret;
}
```

Ternary search

```
int low = 0, high = 1e9;
    for (i = 0 ; i < 70; ++i) {
        int l = low + (high - low) / 3;
        int r = high - (high - low) / 3;
        if (l > r) continue;
        if (f(l) <= f(r)) {
            high = r;
        } else {
            low = l;
        }
    }

    ll ans = f(low);
    for (i = low + 1; i <= high; ++i) {
        ans = min(ans, f(i));
    }
```

Union-find with deletion

```
#include <bits/stdc++.h>
#define MAX_SIZE 131092
#define MID ((l+r)>>1)
#define LEFT (index<<1)
```

```
#define RIGHT (index<<1|1)
using namespace std;

vector<int> f[MAX_SIZE];
vector<int> sz[MAX_SIZE];
// first -> second
vector<pair<int, int>> op;
// if first = second then it is a query of type 2
vector<pair<int, int>> tree[MAX_SIZE * 8];
// each edge's last occurrence time
map<pair<int, int>, int> mp;
// number of day and the corresponding add queries
vector<pair<pair<int, int>, int>> q[MAX_SIZE];
int ans[MAX_SIZE * 4];
int z[MAX_SIZE * 4];
int ed[MAX_SIZE * 4];

void update(int start, int end, int l, int r, int index, int v1, int v2) {
    if (start > end || l > r || start > r || l > end) return;
    if (start <= l && r <= end) {
        tree[index].emplace_back(v1, v2);
        return;
    }

    if (end <= MID) {
        update(start, end, l, MID, LEFT, v1, v2);
    } else if (start >= MID + 1) {
        update(start, end, MID + 1, r, RIGHT, v1, v2);
    } else {
        update(start, end, l, MID, LEFT, v1, v2);
        update(start, end, MID + 1, r, RIGHT, v1, v2);
    }
}

int find(int x) {
    if (x == f[x].back()) return x;
    return find(f[x].back());
}

void unionset(int v1, int v2) {
    int a = find(v1), b = find(v2);
    if (a == b) {
        op.emplace_back(-1, -1);
        return;
    }

    if (sz[a].back() > sz[b].back()) swap(a, b);
    f[a].push_back(b);
    sz[b].push_back(sz[a].back() + sz[b].back());
    op.emplace_back(a, b);
}

void del() {
    if (op.empty()) return;
    if (op.back().first == -1) {
        op.pop_back();
    } else {
        int v1 = op.back().first, v2 = op.back().second;
        op.pop_back();
        f[v1].pop_back();
        sz[v2].pop_back();
    }
}

void dfs(int l, int r, int index) {
    if (l > r) return;
    int cnt = 0;
    for (auto p : tree[index]) {
        unionset(p.first, p.second);
        ++cnt;
    }

    if (l == r) {
        if (z[l]) {
            ans[l] = sz[find(z[l])].back();
        }
        while (cnt-- > 0) del();
        return;
    }

    dfs(l, MID, LEFT);
    dfs(MID + 1, r, RIGHT);
    while (cnt-- > 0) del();
}
```

Persistent range tree 2d sum

```
#pragma GCC optimize(3)
#pragma GCC optimize(2)
#include <bits/stdc++.h>
#define MAX_SIZE 262192
#define MID ((l+r)>>1)
using namespace std;
```

```
struct segt {
    int left, right, value;
};

int cnt = 0;
int root[MAX_SIZE];
segt tree[MAX_SIZE * 30];
int n;

int build(int l, int r) {
    if (l > r) return 0;
    int curr = ++cnt;
    if (l == r) return curr;
    tree[curr].left = build(l, MID);
    tree[curr].right = build(MID + 1, r);
    return curr;
}

int update(int rt, int pos, int l, int r) {
    if (pos < l || pos > r || l > r || rt == 0) return rt;
    int curr = ++cnt;
    tree[curr] = tree[rt];
    if (pos == l && l == r) {
        tree[curr].value++;
        return curr;
    }

    if (pos <= MID) {
        tree[curr].left = update(tree[rt].left, pos, l, MID);
    } else {
        tree[curr].right = update(tree[rt].right, pos, MID + 1, r);
    }
    tree[curr].value = tree[tree[curr].left].value + tree[tree[curr].right].va
lue;
    return curr;
}

int query(int rt, int start, int end, int l, int r) {
    if (rt == 0 || start > end || l > r || start > r || l > end) return 0;
    if (start <= l && r <= end) return tree[rt].value;
    if (end <= MID) {
        return query(tree[rt].left, start, end, l, MID);
    } else if (start >= MID + 1) {
        return query(tree[rt].right, start, end, MID + 1, r);
    }
    return query(tree[rt].left, start, end, l, MID) + query(tree[rt].right, st
art, end, MID + 1, r);
}
```

Persistent range tree lazy update, memory warning, if less than 512MB don't use it

```
#include <bits/stdc++.h>
#define MAX_SIZE 300111
typedef long long ll;
using namespace std;

class persistentTree {
public:
    void build(int n) {
        this->n = n;
        root[0] = crt(0, n - 1);
    }

    void update(int version, int start, int end, ll dk, ll db) {
        root[version] = upd(root[version - 1], start, end, 0, n - 1, dk, db);
    }

    pair<ll, ll> query(int version, int x) {
        return getsum(root[version], x, 0, n-1);
    }

private:
    int n;
    struct node {
        int left, right;
        ll k, b;
    };
    node tree[MAX_SIZE * 140];
    int cnt = 0;
    int root[MAX_SIZE];

    void pushdown(int rt, int l, int r) {
        int mid = l + (r - l) / 2;
        if (rt != 0) {
            if (tree[rt].k != 0 || tree[rt].b != 0) {
                if (tree[rt].left && tree[rt].right) {
                    int nl = ++cnt;
                    int nr = ++cnt;
                    tree[nl].k = tree[tree[rt].left].k + tree[rt].k;
                    tree[nl].b = tree[tree[rt].left].b + tree[rt].b;
                    tree[nr].k = tree[tree[rt].right].k + tree[rt].k;
                    tree[nr].b = tree[tree[rt].right].b + tree[rt].b;
                    tree[nl].left = tree[tree[rt].left].left;
                    tree[nl].right = tree[tree[rt].left].right;
                    tree[nr].left = tree[tree[rt].right].left;
                    tree[nr].right = tree[tree[rt].right].right;
                    tree[rt].left = nl;
                    tree[rt].right = nr;
                    tree[rt].k = tree[rt].b = 0;
                }
            }
        }
    }

    int crt(int l, int r) {
        if (l > r) return 0;
        int curr = ++cnt;
        tree[curr].k = tree[curr].b = 0;
        if (l == r) {
            return curr;
        }

        int mid = l + (r - l) / 2;
        tree[curr].left = crt(l, mid);
        tree[curr].right = crt(mid + 1, r);
        return curr;
    }

    int upd(int rt, int start, int end, int l, int r, ll dk, ll db) {
        if (l > r || rt == 0) return rt;
        int curr = ++cnt;
        pushdown(rt, l, r);
        tree[curr] = tree[rt];
        if (start > r || l > end) return curr;
        if (start <= l && r <= end) {
            tree[curr].k += dk;
            tree[curr].b += db;
            return curr;
        }

        int mid = l + (r - l) / 2;
        if (end <= mid) {
            tree[curr].left = upd(tree[rt].left, start, end, l, mid, dk, db);
        } else if (start >= mid + 1) {
            tree[curr].right = upd(tree[rt].right, start, end, mid + 1, r, dk,
db);
        } else {
            tree[curr].left = upd(tree[rt].left, start, end, l, mid, dk, db);
            tree[curr].right = upd(tree[rt].right, start, end, mid + 1, r, dk,
 db);
        }

        return curr;
    }

    pair<ll, ll> getsum(int rt, int pos, int l, int r) {
        if (rt == 0 || l > r || pos < l || pos > r) return make_pair(0, 0);
        pushdown(rt, l, r);
        if (l == r && pos == l) {
            return make_pair(tree[rt].k, tree[rt].b);
        }

        int mid = l + (r - l) / 2;
        if (pos <= mid) {
            return getsum(tree[rt].left, pos, l, mid);
        }
        return getsum(tree[rt].right, pos, mid + 1, r);
    }
};

persistentTree tree;
```

Sparse Table

```
#pragma GCC optimize(2)
#include <bits/stdc++.h>
typedef long long ll;
#define MAX_SIZE 1000011
using namespace std;

int a[MAX_SIZE];
int table[20][MAX_SIZE];
int logs[MAX_SIZE];
int n, k;

void precompute() {
    int i;
    for (i = 2; i < MAX_SIZE; i++) {
        logs[i] = logs[i/2] + 1;
    }
}

int RMQ(int l, int r) {
    if (l > r) return 4 * MAX_SIZE;
    int len = r - l + 1;
    return min(table[logs[len]][l], table[logs[len]][r - (1 << logs[len]) + 1]
);
}

void build() {
    int i, j;
    for (i = 1; i <= n; i++) {
        table[0][i] = a[i];
    }

    for (i = 1; i <= 19; i++) {
        int prel = (1 << (i - 1));
        for (j = 1; j <= n; j++) {
            if (j + prel <= n) {
                table[i][j] = min(table[i-1][j], table[i-1][j+prel]);
            } else {
                table[i][j] = table[i-1][j];
            }
        }
    }
}
```

Rectangle area 2d plane (K is the times when things count)

```
#include <bits/stdc++.h>
#define MAX_SIZE 131122
#define LEFT (index << 1)
#define RIGHT (index << 1 | 1)
#define MID ((l+r) >> 1)
typedef long long ll;
using namespace std;

struct segt {
    // total cover
    int cover;
    // cover with at exactly i segments
    ll c[11];
};

struct event {
    int l, r, y, tp;
    bool operator < (event other) {
        if (y != other.y) return y < other.y;
        return tp > other.tp;
    }
};

vector<int> disc;
vector<event> evt;
segt tree[MAX_SIZE];
int tmp[MAX_SIZE];
int n, k;

void pullup(int l, int r, int index) {
    int i;
    for (i = 0 ; i <= k; ++i) tree[index].c[i] = 0;
    if (l == r) {
        tree[index].c[min(tree[index].cover, k)] = disc[l+1] - disc[l];
    } else {
        for (i = 0 ; i <= k; ++i) {
            tree[index].c[min(tree[index].cover + i, k)] += tree[LEFT].c[i] +
tree[RIGHT].c[i];
        }
    }
}

void build(int l, int r, int index) {
    if (l > r) return;
    int i;
    for (i = 0 ; i <= k; ++i) {
        tree[index].c[i] = 0;
    }
    tree[index].cover = 0;
    if (l == r) {
        tree[index].c[0] = disc[l+1] - disc[l];
        return;
    }

    build(l, MID, LEFT);
    build(MID + 1, r, RIGHT);
    tree[index].c[0] = tree[LEFT].c[0] + tree[RIGHT].c[0];
}

void update(int start, int end, int l, int r, int index, int tp) {
    if (start > end || l > r || start > r || l > end) return;
    if (start <= l && r <= end) {
        tree[index].cover += tp;
        pullup(l, r, index);
        return;
    }
    if (end <= MID) {
        update(start, end, l, MID, LEFT, tp);
    } else if (start >= MID + 1) {
        update(start, end, MID + 1, r, RIGHT, tp);
    } else {
        update(start, end, l, MID, LEFT, tp);
        update(start, end, MID + 1, r, RIGHT, tp);
    }
    pullup(l, r, index);
}

int main() {
    int T, I;
    int i, x1, x2, y1, y2;
    scanf("%d", &T);
    for (I = 1; I <= T; ++I) {
        ll ans = 0;
        int cnt = 0;
        evt.clear();
        disc.clear();
        scanf("%d%d", &n, &k);
        for (i = 0 ; i < n; ++i) {
            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
            x2++, y2++;
            tmp[cnt++] = x1, tmp[cnt++] = x2;
            evt.push_back(event{x1, x2, y1, 1});
            evt.push_back(event{x1, x2, y2, -1});
        }

        sort(evt.begin(), evt.end());
        sort(tmp, tmp + cnt);
        for (i = 0 ; i < cnt; ++i) {
            if (i == 0 || tmp[i] != tmp[i-1]) disc.push_back(tmp[i]);
        }

        int sz = (int) disc.size();
        build(0, sz - 2, 1);
        for (i = 0 ; i < (int) evt.size() - 1; ++i) {
            int l = lower_bound(disc.begin(), disc.end(), evt[i].l) - disc.beg
in();
            int r = lower_bound(disc.begin(), disc.end(), evt[i].r) - disc.beg
in() - 1;
            update(l, r, 0, sz - 2, 1, evt[i].tp);
            // cout << "cover " << disc[l] << " " << disc[r] << endl;
            // cout << tree[1].c[1] << "---" << tree[1].c[k]<< endl;
            ans = ans + 1ll * tree[1].c[k] * (evt[i+1].y - evt[i].y);
        }
        cout << "Case " << I << ": " << ans << endl;
    }
    return 0;
}
```

2D point counting offline

```
struct qrs {
    int x, y, id, si;
    bool operator < (qrs other) const {
        if (x != other.x) return x < other.x;
        return id < other.id;
    }
};

int cnt = 0;
qrs Q[MAX_SIZE << 3];

void addevent(int x, int y, int id, int si) {
    Q[cnt++] = qrs{x, y, id, si};
}

int main() {

    for (i = 1; i <= q; ++i) {
        int l, r, k;
        scanf("%d%d%d", &l, &r, &k);
        // cout << "query rectangle " << L[k] << " " << l << " " << R[k] << "
" << r << endl;
        addevent(R[k], r, i, 1);
        addevent(L[k] - 1, l - 1, i, 1);
        addevent(R[k], l - 1, i, -1);
        addevent(L[k] - 1, r, i, -1);
    }

    sort(Q, Q + cnt);

    for (i = 0; i < cnt; ++i) {
        if (Q[i].id == 0) {
            update(Q[i].y);
        } else {
            ans[Q[i].id] += Q[i].si * query(Q[i].y);
        }
    }

    for (i = 1; i <= q; ++i) printf("%d\n", ans[i]);
}
```

Interval set (add intervals)

```
set<pair<int, int>> s;

void add(int l, int r) {
    if (s.empty()) {
        s.insert(make_pair(l, r));
    } else {
        set<pair<int, int>> :: iterator iter = s.lower_bound(make_pair(l, 0));
        if (iter != s.begin()) {
            iter = prev(iter);
            if (iter->second >= l) {
                l = min(iter->first, l), r = max(iter->second, r);
                s.erase(iter);
            }
        }

        iter = s.lower_bound(make_pair(l, 0));
        while (iter != s.end() && iter->first <= r) {
            auto it = iter;
            iter++;
            r = max(it->second, r);
            s.erase(it);
        }
        s.insert(make_pair(l, r));
    }
}
```

Heavy light decomposition (change vertices from x to y)

```
rangetree tree;
vector<int> g[MAX_SIZE];
// id means the dfs order in the HLD of a node
// rid means the reverse of the id, i.e. v = rid[id[v]]
int id[MAX_SIZE], rid[MAX_SIZE];
// son is the heavy child of a node, son[i] = 0 if i is a leaf
int son[MAX_SIZE], parent[MAX_SIZE];
// top means the top node of the current heavy link
int top[MAX_SIZE], depth[MAX_SIZE];
// sz means the subtree size, use to determine the heavy child
int sz[MAX_SIZE];
int cnt = 0;
int n, Q;
```

```
void dfs1(int v, int pre) {
     parent[v] = pre;
     depth[v] = depth[pre] + 1;
     sz[v] = 1;
     int max_subsz = 0;
     for (auto nv : g[v]) {
         if (nv != pre) {
            dfs1(nv, v);
            sz[v] += sz[nv];
            if (max_subsz < sz[nv]) {
               son[v] = nv, max_subsz = sz[nv];
            }
         }
     }
}

void dfs2(int v, int tp) {
     top[v] = tp, id[v] = ++cnt, rid[id[v]] = v;
     if (son[v]) {
        dfs2(son[v], tp);
     }

     for (auto nv : g[v]) {
         if (nv != parent[v] && nv != son[v]) {
            dfs2(nv, nv);
         }
     }
}

bool hld(int x, int y) {
    pair<int, int> ret = {inf, -inf};
    int fx = top[x], fy = top[y];
    while (fx != fy) {
        if (depth[fx] >= depth[fy]) {
            auto curr = tree.query(id[fx], id[x], 1, n, 1);
            ret.first = min(ret.first, curr.rmin);
            ret.second = max(ret.second, curr.rmax);
            x = parent[fx], fx = top[x];
        } else {
            auto curr = tree.query(id[fy], id[y], 1, n, 1);
            ret.first = min(ret.first, curr.rmin);
            ret.second = max(ret.second, curr.rmax);
            y = parent[fy], fy = top[y];
        }
    }
    if (id[x] <= id[y]) {
        auto curr = tree.query(id[x], id[y], 1, n, 1);
```

```
            ret.first = min(ret.first, curr.rmin);
            ret.second = max(ret.second, curr.rmax);
        } else {
            auto curr = tree.query(id[y], id[x], 1, n, 1);
            ret.first = min(ret.first, curr.rmin);
            ret.second = max(ret.second, curr.rmax);
        }

        return ret.first == ret.second;
    }
    // change all edge's weight from x->y to min(original weight, v)
    void change(int x, int y, int v) {
        int fx = top[x], fy = top[y];
        while (fx != fy) {
            if (depth[fx] >= depth[fy]) {
                tree.update(id[fx], id[x], 1, n, 1, v);
                x = parent[fx], fx = top[x];
            } else {
                tree.update(id[fy], id[y], 1, n, 1, v);
                y = parent[fy], fy = top[y];
            }
        }

        if (id[x] <= id[y]) {
           tree.update(id[x], id[y], 1, n, 1, v);
        } else {
           tree.update(id[y], id[x], 1, n, 1, v);
        }
    }

    int LCA(int x, int y) {
        int fx = top[x], fy = top[y];
        while (fx != fy) {
            if (depth[fx] >= depth[fy]) {
                x = parent[fx], fx = top[x];
            } else {
                y = parent[fy], fy = top[y];
            }
        }
        if (x == y) return x;
        if (depth[x] < depth[y]) return x;
        return y;
    }
```

Heavy light decomposition (change edges from x to y)

You are given a connected weighted graph with *n* vertices and *m* edges. The graph doesn't contain loops nor multiple edges. Consider some edge with id *i*. Let's determine for this edge the maximum integer weight we can give to it so that it is contained in all minimum spanning trees of the graph if we don't change the other weights. You are to determine this maximum weight described above for each edge. You should calculate the answer for each edge independently, it means there can't be two edges with changed weights at the same time.

```
#include <bits/stdc++.h>
#define MAX_SIZE 524292
#define MID ((l+r)>>1)
#define LEFT (index << 1)
#define RIGHT (index << 1 | 1)
using namespace std;

const int inf = (1<<30);

struct rangemaxtree {
    struct segt {
        int value, lz;
    };

    segt tree[MAX_SIZE];

    void pushdown(int l, int r, int index) {
        if (l > r) return;
        if (tree[index].lz != inf) {
            if (l != r) {
                tree[LEFT].lz = min(tree[LEFT].lz, tree[index].lz);
                tree[RIGHT].lz = min(tree[RIGHT].lz, tree[index].lz);
                tree[LEFT].value = min(tree[LEFT].value, tree[index].lz);
                tree[RIGHT].value = min(tree[RIGHT].value, tree[index].lz);
            }
        }
        tree[index].lz = inf;
    }

    void update(int start, int end, int l, int r, int index, int val) {
        if (start > end || l > r || start > r || l > end) return;
        pushdown(l, r, index);
        if (start <= l && r <= end) {
            tree[index].value = min(tree[index].value, val);
            tree[index].lz = val;
            return;
        }
        if (end <= MID) {
            update(start, end, l, MID, LEFT, val);
        } else if (start >= MID + 1) {
            update(start, end, MID + 1, r, RIGHT, val);
        } else {
            update(start, end, l, MID, LEFT, val);
            update(start, end, MID + 1, r, RIGHT, val);
        }
        tree[index].value = max(tree[LEFT].value, tree[RIGHT].value);
    }

    int query(int start, int end, int l, int r, int index) {
        if (start > end || l > r || start > r || l > end) return -inf;
        pushdown(l, r, index);
        if (start <= l && r <= end) return tree[index].value;
        if (end <= MID) {
            return query(start, end, l, MID, LEFT);
        } else if (start >= MID + 1) {
            return query(start, end, MID + 1, r, RIGHT);
        }
        return max(query(start, end, l, MID, LEFT), query(start, end, MID + 1,
 r, RIGHT));
    }
};
// t1 maintains the range max of the mst-edges, t2 is used for updates
rangemaxtree t1, t2;
vector<int> g[MAX_SIZE];
// id means the dfs order in the HLD of a node
// rid means the reverse of the id, i.e. v = rid[id[v]]
int id[MAX_SIZE], rid[MAX_SIZE];
// son is the heavy child of a node, son[i] = 0 if i is a leaf
int son[MAX_SIZE], parent[MAX_SIZE];
// top means the top node of the current heavy link
int top[MAX_SIZE], depth[MAX_SIZE];
// sz means the subtree size, use to determine the heavy child
int sz[MAX_SIZE];
int cnt = 0;
int n;

int f[MAX_SIZE];

void init() {
    int i;
    for (i = 0 ; i < MAX_SIZE; ++i) {
        t1.tree[i].lz = t2.tree[i].lz
        = t1.tree[i].value = t2.tree[i].value = inf;
        f[i] = i;
    }
```

```
}

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

void unionset(int x, int y) {
    int a = find(x), b = find(y);
    f[a] = b;
}

struct edge {
    int from, to, w, id, select;
    bool operator < (edge other) const {
        return w < other.w;
    }
};

edge edg[MAX_SIZE];

void dfs1(int v, int pre) {
     parent[v] = pre;
     depth[v] = depth[pre] + 1;
     sz[v] = 1;
     int max_subsz = 0;
     for (auto nv : g[v]) {
         if (nv != pre) {
            dfs1(nv, v);
            sz[v] += sz[nv];
            if (max_subsz < sz[nv]) {
               son[v] = nv, max_subsz = sz[nv];
            }
         }
     }
}

void dfs2(int v, int tp) {
     top[v] = tp, id[v] = ++cnt, rid[id[v]] = v;
     if (son[v]) {
        dfs2(son[v], tp);
     }

     for (auto nv : g[v]) {
         if (nv != parent[v] && nv != son[v]) {
            dfs2(nv, nv);
         }
     }
```

```
}

int hld(int x, int y) {
    int ret = -inf;
    int fx = top[x], fy = top[y];
    while (fx != fy) {
        if (depth[fx] >= depth[fy]) {
            ret = max(ret, t1.query(id[fx], id[x], 1, n, 1));
            x = parent[fx], fx = top[x];
        } else {
            ret = max(ret, t1.query(id[fy], id[y], 1, n, 1));
            y = parent[fy], fy = top[y];
        }
    }

    if (x == y) return ret;
    if (id[x] <= id[y]) {
       ret = max(ret, t1.query(id[son[x]], id[y], 1, n, 1));
    } else {
       ret = max(ret, t1.query(id[son[y]], id[x], 1, n, 1));
    }
    return ret;
}
// change all edge's weight from x->y to min(original weight, v)
void change(int x, int y, int v) {
    int fx = top[x], fy = top[y];
    while (fx != fy) {
        if (depth[fx] >= depth[fy]) {
            t2.update(id[fx], id[x], 1, n, 1, v);
            x = parent[fx], fx = top[x];
        } else {
            t2.update(id[fy], id[y], 1, n, 1, v);
            y = parent[fy], fy = top[y];
        }
    }

    if (x == y) return;
    if (id[x] <= id[y]) {
       t2.update(id[son[x]], id[y], 1, n, 1, v);
    } else {
       t2.update(id[son[y]], id[x], 1, n, 1, v);
    }
}

int ans[MAX_SIZE];
int wt[MAX_SIZE];

void build(int l, int r, int index) {
    if (l > r) return;
    if (l == r) {
        t1.tree[index].value = wt[rid[l]];
        return;
    }

    build(l, MID, LEFT);
    build(MID + 1, r, RIGHT);
    t1.tree[index].value = max(t1.tree[LEFT].value, t1.tree[RIGHT].value);
}

int main() {
    init();
    int m, i;
    scanf("%d%d", &n, &m);
    for (i = 1; i <= m; ++i) {
        scanf("%d%d%d", &edg[i].from, &edg[i].to, &edg[i].w);
        edg[i].id = i;
    }

    sort(edg + 1, edg + 1 + m);

    for (i = 1; i <= m; ++i) {
        if (find(edg[i].from) != find(edg[i].to)) {
            edg[i].select = 1;
            unionset(edg[i].from, edg[i].to);
            g[edg[i].from].push_back(edg[i].to);
            g[edg[i].to].push_back(edg[i].from);
            // cout << "edge: " << edg[i].from << " " << edg[i].to << endl;
        }
    }

    dfs1(1, 0);
    dfs2(1, 1);

    for (i = 1; i <= m; ++i) {
        if (edg[i].select == 1) {
            if (depth[edg[i].from] > depth[edg[i].to]) swap(edg[i].from, edg[i
].to);
            wt[edg[i].to] = edg[i].w;
        }
    }

    build(1, n, 1);

    for  (i = 1; i <= m; ++i) {
        // this edge is not in the mst, the way to make it in all mst
        // is to make it smaller than the largest edge in the mst from edg[i].
from, to edg[i].to
        if (edg[i].select == 0) {
            ans[edg[i].id] = hld(edg[i].from, edg[i].to) - 1;
            change(edg[i].from, edg[i].to, edg[i].w - 1);
        }
    }

    for (i = 1; i <= m; ++i) {
        if (edg[i].select == 1) {
            ans[edg[i].id] = t2.query(id[edg[i].to], id[edg[i].to], 1, n, 1);
            if (ans[edg[i].id] >= inf) {
                ans[edg[i].id] = -1;
            }
        }
    }

    for (i = 1; i <= m; ++i) {
        printf("%d ", ans[i]);
    }
    printf("\n");
    return 0;
}
```

Expression tree

```
struct node {
    node *left, *right;
    char conn;
    int id;
    node(char conn, node* left, node* right) {
        this->conn = conn;
        this->left = left;
        this->right = right;
    }
};

node *root;

struct expression {

    node *build(char s[]) {
        return buildfromPrefix(convertToprefix(s));
    }

    int getpriority(char ep) {
```

```
    if (ep == '*' || ep == '/') return 2;
    if (ep == '+' || ep == '-') return 1;
    if (ep == '?') return 0;
    return -1;
}

node *buildfromPrefix(vector<char> s) {
    int len = s.size(), i;
    stack<node*> st;
    for (i = 0; i < len; ++i) {
        if (getpriority(s[i]) == -1) {
            st.push(new node(s[i], NULL, NULL));
        } else {
            if (getpriority(s[i]) == 2) {
                node *nd = new node(s[i], NULL, st.top());
                st.pop();
                st.push(nd);
            } else {
                node *f1 = st.top();
                st.pop();
                node *f2 = st.top();
                st.pop();
                node *f = new node(s[i], f2, f1);
                st.push(f);
            }
        }
    }

    return st.top();
}

vector<char> convertToprefix(char s[]) {
    vector<char> ret;
    int i, len = strlen(s);
    stack<char> st;
    for (i = 0 ; i < len; ++i) {
        if (getpriority(s[i]) == -1) {
            if (s[i] == '(') {
                st.push(s[i]);
            } else if (s[i] == ')') {
                while (!st.empty() && st.top() != '(') {
                    ret.push_back(st.top());
                    st.pop();
                }
                st.pop();
            } else {
                ret.push_back(s[i]);
            }
            } else {
                while (!st.empty() && (getpriority(s[i]) <= getpriority(st.top
()))) {
                    ret.push_back(st.top());
                    st.pop();
                }
                st.push(s[i]);
            }
        }

        while (!st.empty()) {
            ret.push_back(st.top());
            st.pop();
        }
        return ret;
    }
};

expression ep;
```

Convex hull

```
#include <bits/stdc++.h>
#define MOD 1000000007
#define MAX_SIZE 262192
using namespace std;
typedef long long ll;
struct line {
    ll m, b;
    int id;
};

ll floordiv (ll a, ll b) {
    return a / b - (a%b && ((a <0) ^ (b <0) )) ;
}

ll intersect ( line a, line b) {
    return floordiv (b.b - a.b, a.m - b.m);
}

pair<int, int> org[MAX_SIZE >> 1];
pair<pair<int, int>, int> ist[MAX_SIZE >> 1];

struct Convex {
    vector<line> cht;

    void insert(line l) {
        auto n = cht.size();
        while ((n >= 1 &&  l.m == cht[n-1].m && l.b >= cht[n-1].b)
        || (n >= 2 && intersect(cht[n-1], cht[n-2]) >= intersect(cht[n-
1], l))) {
            cht.pop_back();
            n = cht.size();
        }
        cht.push_back(l);
    }

    ll query(ll x) {
        int low = 0, high = (int) cht.size() - 2;
        int ret = (int) cht.size() - 1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (intersect(cht[mid], cht[mid + 1]) >= x) {
                ret = mid;
                high = mid - 1;
            } else {
                low = mid + 1;
            }
        }

        return cht[ret].id;
    }
};
```

Divide and conquer optimization

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 2e5 + 10;
const ll INF = 1e18 + 10;
int N , K , a[maxn];
int pos[maxn] , M = 0 , mp[maxn];
ll DP[maxn][102];
int cnt[105];
ll cost[maxn]; // # of inversions if we put j on pos[i]

ll cal(int l , int r){
    return cost[r] - cost[l - 1];
}

void solve(int l , int r , int x , int y , int k){
    if(l > r)return;
    int mid = (l + r) >> 1;
    int pos = -1;
    DP[mid][k] = DP[mid][k + 1]; // notice here, we inherent from layer k + 1,
 if not, just set to some dummy value
    for(int i = max(1 , x); i <= min(y , mid); ++i){
        ll C = cal(i , mid) + (ll)(i - 1) * (mid - i + 1);
        if(DP[mid][k] < DP[i - 1][k + 1] + C){
            DP[mid][k] = DP[i - 1][k + 1] + C;
            pos = i;
        }
    }
    solve(l , mid - 1 , x , pos , k);
    solve(mid + 1 , r , pos , y , k);
}

int main() {
    // init dp
    for(int j = K - 1; j >= 1; --j){
        solve(1 , M , 1 , M , j);
    }
    return 0;
}
```

Sweeping line

Geometry

Nearest 2 points on a plane

```
#pragma GCC optimize(3)
#include <bits/stdc++.h>
#define MAX_SIZE 100011
using namespace std;
typedef long long ll;
struct point {
    int x, y;
    bool operator < (point other) const {
        return y < other.y;
    }
};

const ll inf = 5e18;
```

```
point pt[MAX_SIZE];

int a[MAX_SIZE];

ll dist(point p1, point p2) {
    return 1ll * (p1.x - p2.x) * (p1.x - p2.x) +
           1ll * (p1.y - p2.y) * (p1.y - p2.y);
}

ll divc(int l, int r) {
    int i, j;
    ll ret = inf;
    if (l >= r) return inf;
    if (r - l == 1) {
        ret = dist(pt[l], pt[r]);
        return ret;
    }

    int mid = (l + r) >> 1;
    ll h1 = divc(l, mid), h2 = divc(mid + 1, r);
    ret = min(h1, h2);
    vector<point> can;
    for (i = mid; i >= l; --i) {
        if (1ll * (pt[mid].x - pt[i].x) * (pt[mid].x - pt[i].x) > ret) break;
        can.push_back(pt[i]);
    }

    for (i = mid + 1; i <= r; ++i) {
        if (1ll * (pt[mid].x - pt[i].x) * (pt[mid].x - pt[i].x) > ret) break;
        can.push_back(pt[i]);
    }

    sort(can.begin(), can.end());

    int sz = can.size();
    for (i = 0; i < sz; ++i) {
        for (j = i + 1; j < sz; ++j) {
            if (1ll * (can[i].y - can[j].y) * (can[i].y - can[j].y) > ret) bre
ak;
            ret = min(ret, dist(can[i], can[j]));
            // cout << i << " " << j << "  " << ret << endl;
        }
    }

    return ret;
}

int main() {
    int n, i;
    scanf("%d", &n);
    for (i = 1; i <= n; ++i) {
        scanf("%d",  &a[i]);
        pt[i].x = i, pt[i].y = a[i] + pt[i-1].y;
    }

    ll ans = divc(1, n);
    printf("%I64d\n", ans);
    return 0;
}
```

General template

```
#include <bits/stdc++.h>
const double Pi = acos(-1.0);
using namespace std;
const double eps = 1e-9;
int sgn(double x) {
    if (fabs(x) <= eps) return 0;
    if (x > 0) return 1;
    return -1;
}

struct point {
    double x, y;
    point(double x=0, double y=0) : x(x), y(y) {}
    point operator -(point other) {
        return point(x - other.x, y - other.y);
    }

    point operator +(point other) {
        return point(x + other.x, y + other.y);
    }

    bool operator < (point other) {
        if (x != other.x) return x < other.x;
        return y < other.y;
    }

    point operator *(double p) {
        return point(x*p, y*p);
    }
};

typedef point Vector;
```

```
struct segment {
    point v1, v2;
};

struct Line {
    // v is the point, p is the direction vector
    point v, p;
    Line(point v, point p):v(v), p(p) {}
};

double Dot(Vector A, Vector B){
    return A.x*B.x + A.y*B.y;
}

double Cross(Vector A, Vector B){
    return A.x*B.y-A.y*B.x;
}

double Dist(point x, point y) {
    return Length(x-y);
}


double Angle(Vector A, Vector B){
    return acos(Dot(A, B)/Length(A)/Length(B));
}

Vector Rotate(Vector A, double rad){ // rotation anti-clockwise with rad
    return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
}

Vector Normal(Vector A){ // left rotation 90 degree
    double L = Length(A);
    return Vector(-A.y/L, A.x/L);
}
// if c is on the left of a,b
bool ToLeftTest(point a, point b, point c){
    return Cross(b - a, c - b) > 0;
}

double DistanceToLine(point P, point A, point B){
    Vector v1 = B-A, v2 = P-A;
    return fabs(Cross(v1, v2)/Length(v1));
}

double DistanceToSegment(point P, point A, point B){
    Vector v1 = B-A, v2 = P-A, v3 = P-B;
    if(sgn(Dot(v1, v2)) < 0)
```

```
        return Length(v2);
    if(sgn(Dot(v1, v3)) > 0)
        return Length(v3);
    return DistanceToLine(P, A, B);
}

// line l1 and line l2 interection
vector<point> GetLineIntersection(Line l1, Line l2){
    vector<point> ret;
    point P = l1.v, Q = l2.v;
    Vector v = l1.p, w = l2.p;
    if (sgn(Cross(v, w)) == 0) return ret;
    Vector u = P-Q;
    double t = Cross(w, u)/Cross(v, w);
    point pp = P+v*t;
    ret.push_back(pp);
    return ret;
}
// check whether p is on the segment a1, a2
// the second equal sign is added only if p is the same as a1/a2 is allowed
bool OnSegment(point p, point a1, point a2){
    return sgn(Cross(a1-p, a2-p)) == 0 && sgn(Dot(a1-p, a2-p)) <= 0;
}

vector<point> GetSegmentIntersection(segment s1, segment s2) {
    Line l1 = Line(s1.v1, s1.v2 - s1.v1), l2 = Line(s2.v1, s2.v2 - s2.v1);
    auto vc = GetLineIntersection(l1, l2);
    if (vc.empty()) return vc;
    if (!OnSegment(vc.front(), s1.v1, s1.v2)
     || !OnSegment(vc.front(), s2.v1, s2.v2)) return vector<point>();
    return vc;
}

bool SegmentProperIntersection(point a1, point a2, point b1, point b2){
    double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1);
    double c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2-b1);
    //if 判断控制是否允许线段在端点处相交，根据需要添加
    if(!sgn(c1) || !sgn(c2) || !sgn(c3) || !sgn(c4)){
        bool f1 = OnSegment(b1, a1, a2);
        bool f2 = OnSegment(b2, a1, a2);
        bool f3 = OnSegment(a1, b1, b2);
        bool f4 = OnSegment(a2, b1, b2);
        bool f = (f1|f2|f3|f4);
        return f;
    }
    return (sgn(c1)*sgn(c2) < 0 && sgn(c3)*sgn(c4) < 0);
}
```

```
bool cmp(point a, point b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}

bool cw(point a, point b, point c) {
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) < 0;
}

bool ccw(point a, point b, point c) {
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) > 0;
}

void convex_hull(vector<point>& a) {
    if (a.size() == 1)
        return;

    sort(a.begin(), a.end(), &cmp);
    point p1 = a[0], p2 = a.back();
    vector<point> up, down;
    up.push_back(p1);
    down.push_back(p1);
    for (int i = 1; i < (int)a.size(); i++) {
        if (i == (int) a.size() - 1 || cw(p1, a[i], p2)) {
            while (up.size() >= 2 && !cw(up[up.size()-2], up[up.size()-
1], a[i]))
                up.pop_back();
            up.push_back(a[i]);
        }
        if (i == (int) a.size() - 1 || ccw(p1, a[i], p2)) {
            while(down.size() >= 2 && !ccw(down[down.size()-
2], down[down.size()-1], a[i]))
                down.pop_back();
            down.push_back(a[i]);
        }
    }

    a.clear();
    for (int i = 0; i < (int)up.size(); i++)
        a.push_back(up[i]);
    for (int i = down.size() - 2; i > 0; i--)
        a.push_back(down[i]);
}

double area(const vector<point>& fig) {
    double res = 0;
    for (unsigned i = 0; i < fig.size(); i++) {
        point p = i ? fig[i - 1] : fig.back();
        point q = fig[i];
        res += (p.x - q.x) * (p.y + q.y);
    }
    return fabs(res) / 2;
}

double Length(Vector A){
    return sqrt(Dot(A, A));
}

double polarangle(double y, double x) {
    return atan2(y, x);
}


double roting_caplter(vector<point> &pt) {
    int sz = pt.size(), q = 1;
    double ans = 1e18;
    pt.push_back(pt[0]);
    for (int i = 0 ; i < sz; ++i) {
        while(fabs(Cross(pt[i+1] - pt[i],pt[q] - pt[i])) < fabs(Cross(pt[i+1]
- pt[i], pt[q+1] - pt[i]))){
            q = (q + 1) % sz;
        }

        double tmp = DistanceToLine(pt[q], pt[i], pt[i+1]);
        ans = min(ans,tmp);
    }
    return ans;
}


/*  Line intersection integer range */
struct pt {
    long long x, y;
    pt() {}
    pt(long long _x, long long _y) : x(_x), y(_y) {}
    pt operator-(const pt& p) const { return pt(x - p.x, y - p.y); }
    long long cross(const pt& p) const { return x * p.y - y * p.x; }
    long long cross(const pt& a, const pt& b) const { return (a - *this).cross
(b - *this); }
};

int sgn(const long long& x) { return x >= 0 ? x ? 1 : 0 : -1; }

bool inter1(long long a, long long b, long long c, long long d) {
    if (a > b)
        swap(a, b);
    if (c > d)
        swap(c, d);
    return max(a, c) <= min(b, d);
}

bool check_inter(const pt& a, const pt& b, const pt& c, const pt& d) {
    if (c.cross(a, d) == 0 && c.cross(b, d) == 0)
        return inter1(a.x, b.x, c.x, d.x) && inter1(a.y, b.y, c.y, d.y);
    return sgn(a.cross(b, c)) != sgn(a.cross(b, d)) &&
           sgn(c.cross(d, a)) != sgn(c.cross(d, b));
}

/* circle template */

struct circle {
    double x, y, r;
};

int dcmp(double d1, double d2) {
    return sgn(d1-d2);
}

bool intersect(circle c1, circle c2) {
    double d = Dist(point{c1.x, c1.y}, point{c2.x, c2.y});
    double d1 = c1.r + c2.r, d2 = fabs(c1.r - c2.r);
    return dcmp(d, d1) <= 0 && dcmp(d, d2) >= 0;
}

// intersection of circle c with line Ax + By + C = 0
vector<point> linecircle(circle cr, double A, double B, double C) {
    vector<point> ret;
    double r = cr.r, a = A, b = B, c = C + A * cr.x + B * cr.y;
    double x0 = -a*c/(a*a+b*b), y0 = -b*c/(a*a+b*b);
    if (sgn(c * c - r * r * (a*a + b*b)) > 0) {
        return ret;
    } else if (sgn(c*c - r*r*(a*a+b*b)) == 0) {
        ret.push_back(point{x0, y0});
    } else {
        double d = r*r - c*c/(a*a+b*b);
        double mult = sqrt (d / (a*a+b*b));
        double ax, ay, bx, by;
        ax = x0 + b * mult;
        bx = x0 - b * mult;
        ay = y0 - a * mult;
        by = y0 + a * mult;
        ret.push_back(point{ax, ay});
        ret.push_back(point{bx, by});
    }
    return ret;
}
```

```
vector<point> circleintersect(circle c1, circle c2) {
    // reduce the problem to
    // find the line Ax + By + C = 0 intersect with c1
    circle c;
    c.r = c1.r;
    c.x = 0, c.y = 0;
    c2.x = c2.x - c1.x;
    c2.y = c2.y - c1.y;
    double A = -2.0 * c2.x;
    double B = -2.0 * c2.y;
    double C = c2.x * c2.x + c2.y * c2.y + c1.r * c1.r - c2.r * c2.r;
    vector<point> ret = linecircle(c, A, B, C);
    for (auto &r : ret) {
        r.x += c1.x;
        r.y += c1.y;
    }
    return ret;
}

/* tangent line of circle */
struct line {
    double a, b, c;
};


struct Circle : pt {
    double r;
};

const double EPS = 1E-9;

double sqr (double a) {
    return a * a;
}

void tangents (pt c, double r1, double r2, vector<line> & ans) {
    double r = r2 - r1;
    double z = sqr(c.x) + sqr(c.y);
    double d = z - sqr(r);
    if (d < -EPS)  return;
    d = sqrt (abs (d));
    line l;
    l.a = (c.x * r + c.y * d) / z;
    l.b = (c.y * r - c.x * d) / z;
    l.c = r1;
    ans.push_back (l);
}
```

```
vector<line> tangents (Circle a, Circle b) {
    vector<line> ans;
    for (int i=-1; i<=1; i+=2)
        for (int j=-1; j<=1; j+=2)
            tangents (b-a, a.r*i, b.r*j, ans);
    for (size_t i=0; i<ans.size(); ++i)
        ans[i].c -= ans[i].a * a.x + ans[i].b * a.y;
    return ans;
}
```

Pick’s theorem

$$S = Inner + \frac{Boarder}{2} - 1$$

String

KMP all indices (1-index)

You are given a string ss of length n. The **prefix function** for this string is defined as an array π of length n, where $\pi[i]$ is the length of the longest proper prefix of the substring $s[0 \dots i]$ which is also a suffix of this substring. A proper prefix of a string is a prefix that is not equal to the string itself. By definition, $\pi[0]=0$.

```
#include <bits/stdc++.h>
#define MAX_SIZE 8011
using namespace std;

vector<char> g[MAX_SIZE];
vector<int> p[MAX_SIZE];
int curr[MAX_SIZE];
char st[MAX_SIZE];
int dp[MAX_SIZE];
int numd[MAX_SIZE];

void upd(int idx, char ch) {
    g[idx].push_back(ch);
    // special case for index 1
    if ((int) g[idx].size() == 2) {
        p[idx].push_back(0);
        return;
    }
    while (curr[idx] > 0 && ch != g[idx][curr[idx]+1]) curr[idx] = p[idx][curr
[idx]];
    if (g[idx].back() == g[idx][curr[idx]+1]) ++curr[idx];
    p[idx].push_back(curr[idx]);
}

int main() {
    int i, n, j;
    scanf("%s", st + 1);
    n = strlen(st + 1);
    for (i = 1; i <= n; ++i) {
        numd[i] = (int) to_string(i).length();
        g[i].push_back('#');
        p[i].push_back(0);
    }

    for (i = 1; i <= n; ++i) {
        dp[i] = 2 * n + 2;
        for (j = 1; j <= i; ++j) {
            upd(j, st[i]);
            int period = (i - j + 1) - p[j].back();
            // cout << i << ": " << j << " " << period << endl;
            if ((i - j + 1) % period == 0) {
                dp[i] = min(dp[i], dp[j-
1] + numd[(i - j + 1) / period] + period);
            }

            dp[i] = min(dp[i], dp[j-1] + (i - j + 1) + 1);
        }
    }

    printf("%d\n", dp[n]);
    return 0;
}
```

Prefix function only once (1-index)

```
#include <bits/stdc++.h>
#define MAX_SIZE 100011
using namespace std;
typedef long long ll;
char st[MAX_SIZE];
int p[MAX_SIZE];
ll dp[MAX_SIZE];
vector<int> ans;

int main() {
    int N, i, j = 0;
    scanf("%s", st+1);
    N = strlen(st+1);
    for (i = 2; i <= N; ++i) {
        while (j > 0 && st[i] != st[j+1]) j = p[j];
        if (st[i] == st[j+1]) ++j;
        p[i] = j;
}
}
```

Z-function (0-index)

Suppose we are given a string ss of length n. The **Z-function** for this string is an array of length nn where the i-th element is equal to the greatest number of characters starting from the position ii that coincide with the first characters of ss.

In other words, z[i] is the length of the longest common prefix between ss and the suffix of ss starting at i. (0-index)

```
#include <bits/stdc++.h>
#define MAX_SIZE 100011
using namespace std;

char st[MAX_SIZE], q[1011];
int pre[1011], suf[1011];

vector<int> z_function(vector<char> &s) {
    int n = (int) s.size();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```

Suffix array

```
#pragma GCC optimize(3)
#pragma GCC optimize(2)
```

```
#pragma GCC optimize("Ofast")
#include <bits/stdc++.h>
#define MAX_SIZE 400111
#define MAXLOG 19
typedef long long ll;
using namespace std;

struct SA {
    int p[MAX_SIZE];
    int cnt[MAX_SIZE];
    int c[MAXLOG][MAX_SIZE];
    int logs[MAX_SIZE];
    int rank[MAX_SIZE];
    const int alphabet = (MAX_SIZE >> 1);
    int n;

    void init() {
        memset(p, 0, sizeof(p));
        memset(cnt, 0, sizeof(cnt));
        memset(c, 0, sizeof(c));
        memset(rank, 0, sizeof(rank));
    }

    void sort_cyclic_shifts(vector<int> &s) {
        int h, i;
        n = (int) s.size();
        for (i = 2; i < MAX_SIZE; ++i) logs[i] = logs[i/2]+1;
        for (i = 0; i < n; i++)
            cnt[s[i]]++;
        for (i = 1; i < alphabet; i++)
            cnt[i] += cnt[i-1];
        for (i = 0; i < n; i++)
            p[--cnt[s[i]]] = i;
        c[0][p[0]] = 0;
        int classes = 1;
        for (i = 1; i < n; i++) {
            if (s[p[i]] != s[p[i-1]])
                classes++;
            c[0][p[i]] = classes - 1;
        }

        vector<int> pn(n), cn(n);
        for (h = 0; (1 << h) < n; ++h) {
            for (i = 0; i < n; i++) {
                pn[i] = p[i] - (1 << h);
                if (pn[i] < 0)
                    pn[i] += n;
            }
            fill(cnt, cnt + classes, 0);
            for (i = 0; i < n; i++)
                cnt[c[h][pn[i]]]++;
            for (i = 1; i < classes; i++)
                cnt[i] += cnt[i-1];
            for (i = n-1; i >= 0; i--)
                p[--cnt[c[h][pn[i]]]] = pn[i];
            cn[p[0]] = 0;
            classes = 1;
            for (i = 1; i < n; i++) {
                pair<int, int> cur = {c[h][p[i]], c[h][(p[i] + (1 << h)) % n]}
;
                pair<int, int> prev = {c[h][p[i-1]], c[h][(p[i-
1] + (1 << h)) % n]};
                if (cur != prev)
                    ++classes;
                cn[p[i]] = classes - 1;
            }

            for (i = 0 ; i < n; ++i) {
                c[h+1][i] = cn[i];
            }
        }

        for (i = 0 ; i < n; ++i) rank[p[i]] = i;
    }

    int lcp(int i, int j) {
        int ans = 0, k;
        for (k = logs[n]; k >= 0; k--) {
            if (c[k][i] == c[k][j]) {
                ans += 1 << k;
                i += 1 << k;
                j += 1 << k;
            }
        }
        return ans;
    }
};

SA sa;
```

Hashing

```
#pragma GCC optimize(3)
#pragma GCC optimize(2)
#include <bits/stdc++.h>
#define MAX_SIZE 200111
```

```
using namespace std;
typedef long long ll;

struct rolling_hash {
    ll val[MAX_SIZE];
    ll ep[MAX_SIZE];
    ll mod = 1e9 + 9;
    int base = 47;
    int N;
    void init(int base=47, ll mod=1e9+9) {
        int i;
        this->base = base, this->mod = mod;
        ep[0] = 1;
        for (i = 1; i < MAX_SIZE; ++i) {
            ep[i] = ep[i-1] * base;
            ep[i] %= mod;
        }
    }

    void calc(char st[], int N) {
        this->N = N;
        for (int i = 1 ; i <= N; ++i) {
            val[i] = val[i-1] * base + (st[i] - 'a');
            val[i] %= mod;
        }
    }

    ll decode(int l, int r) {
        ll valL = val[l-1] * ep[r - l + 1];
        ll valR = val[r];
        return ((valR - valL) % mod + mod) % mod;
    }

    bool equal(int l1, int r1, int l2, int r2) {
        if (r1 - l1 != r2 - l2) return false;
        return decode(l1, r1) == decode(l2, r2);
    }
};

rolling_hash h;
```

Trie with deletion

```
class Trie {
public:
    Trie() {
        this->root = new node();
        add(0);
    }

    void add(int x) {
        vector<int> vc = parse(x);
        root = insert(root, vc, 0);
    }

    void del(int x) {
        vector<int> vc = parse(x);
        root = remove(root, vc, 0);
    }

    int query(int x) {
        int ret = 0, i;
        vector<int> vc = parse(x);
        node *curr = root;
        for (i = 0 ; i < 31; ++i) {
            if (vc[i] == 0) {
                if (curr->child[1] != NULL) {
                    ret = (ret << 1) + 1;
                    curr = curr->child[1];
                } else {
                    ret = ret << 1;
                    curr = curr->child[0];
                }
            } else {
                if (curr->child[0] != NULL) {
                    ret = (ret << 1) + 1;
                    curr = curr->child[0];
                } else {
                    ret = ret << 1;
                    curr = curr->child[1];
                }
            }
        }
        return ret;
    }

private:
    struct node {
        int cnt;
        node *child[2];
        node() {
            this->child[0] = this->child[1] = NULL;
            this->cnt = 0;
        }
```

```
};

node *root;

int size(node *rt) {
    return rt == NULL ? 0 : rt->cnt;
}

node *insert(node *rt, vector<int> &a, int pos) {
    int sz = a.size();
    if (pos > sz) return rt;
    if (pos == sz) {
        rt->cnt++;
        return rt;
    }

    if (rt->child[a[pos]] == NULL) {
        rt->child[a[pos]] = new node();
    }

    rt->child[a[pos]] = insert(rt->child[a[pos]], a, pos + 1);
    rt->cnt = size(rt->child[0]) + size(rt->child[1]);
    return rt;
}

node *remove(node *rt, vector<int> &a, int pos) {
    int sz = a.size();
    if (rt == NULL || pos > sz) return rt;
    if (pos == sz) {
        rt->cnt--;
        return rt;
    }

    rt->child[a[pos]] = remove(rt->child[a[pos]], a, pos + 1);
    if (!size(rt->child[a[pos]])) {
        delete rt->child[a[pos]];
        rt->child[a[pos]] = NULL;
    }

    rt->cnt = size(rt->child[0]) + size(rt->child[1]);
    return rt;
}

vector<int> parse(int x) {
    vector<int> ret = vector<int>(31, 0);
    int idx = 30;
    while (x > 0) {
        ret[idx--] = x & 1;
```

```
            x >>= 1;
        }
        return ret;
    }
};

Trie t = Trie();
```

Manchester algorithm

If the size of palindrome centered at i is x, then d2[i] stores (x+1)/2. If the size of palindrome centered at i is x, then d2[i] stores x/2.

```
int manchester(vector<int> &s) {
    int n = s.size();
    /*cout << "calculate ";
    for (auto ch : s) {
        cout << ch << " ";
    }
    cout << endl;*/
    vector<int> d1(n);
    for (int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
        while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
            k++;
        }
        d1[i] = k--;
        if (i + k > r) {
            l = i - k;
            r = i + k;
        }
    }

    vector<int> d2(n);
    for (int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
        while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
            k++;
        }
        d2[i] = k--;
        if (i + k > r) {
            l = i - k - 1;
            r = i + k ;
        }
    }

    int ret = 0;
    for (int i = 0 ; i < n; ++i) {
        if (s[i] >= 0) {
            ret += d1[i];
        }
    }
    for (int i = 0; i < n; ++i) {
        if (s[i] >= 0) {
            ret += d2[i];
        }
    }

    // cout << "get " << ret << endl;
    return ret;
}
```

Graph

2 Sat

```
#include <bits/stdc++.h>
#define MAX_SIZE 200011
using namespace std;

vector<int> g[MAX_SIZE];
int cmpid[MAX_SIZE], low[MAX_SIZE], visited[MAX_SIZE], instack[MAX_SIZE];
stack<int> st;
int cnt, N, cid = 1;

void tarjan(int v) {
    st.push(v);
    instack[v] = 1;
    visited[v] = low[v] = ++cnt;
    for (auto nv : g[v]) {
        if (!visited[nv]) {
            tarjan(nv);
            low[v] = min(low[v], low[nv]);
        } else if (instack[nv]) {
            low[v] = min(low[v], visited[nv]);
        }
    }
    if (low[v] == visited[v]) {
        while (st.top() != v) {
            instack[st.top()] = 0;
            cmpid[st.top()] = cid;
            st.pop();
        }

        cmpid[st.top()] = cid++;
        instack[st.top()] = 0;
        st.pop();

    }
}

int pos(int x) {
    return x * 2;
}

int neg(int x) {
    return x * 2 + 1;
}

int a, b;
int p[MAX_SIZE];
map<int, int> mp;

int main() {
    int i;
    scanf("%d%d%d", &N, &a, &b);
    for (i = 0; i < N; ++i) {
        scanf("%d", &p[i]);
        mp[p[i]] = i;
    }

    for (i = 0; i < N; ++i) {
        if (mp.find(a - p[i]) != mp.end()) {
            g[pos(i)].push_back(pos(mp[a - p[i]]));
            g[neg(i)].push_back(neg(mp[a - p[i]]));
        } else {
            g[pos(i)].push_back(neg(i));
        }

        if (mp.find(b - p[i]) != mp.end()) {
            g[neg(i)].push_back(neg(mp[b - p[i]]));
            g[pos(i)].push_back(pos(mp[b - p[i]]));
        } else {
            g[neg(i)].push_back(pos(i));
        }
    }

    for (i = 0; i < 2 * N; ++i) {
        if (!visited[i]) {
            tarjan(i);
        }
    }

    for (i = 0; i < N; ++i) {
        if (cmpid[pos(i)] == cmpid[neg(i)]) {
            printf("NO\n");
            return 0;
        }
    }

    printf("YES\n");
    for (i = 0; i < N; ++i) {
        if (cmpid[pos(i)] < cmpid[neg(i)]) {
            printf("0 "); // 0 means true
        } else {
            printf("1 "); // 1 means false
        }
    }
    printf("\n");
    return 0;
}
```

Biconnected component

```
#include <bits/stdc++.h>
#define MOD 1000000007
#define MAX_SIZE 401111

typedef long long ll;
using namespace std;

int cmpid[MAX_SIZE];
vector<pair<int, int> > g[MAX_SIZE];
int low[MAX_SIZE];
int visited[MAX_SIZE];
int instack[MAX_SIZE];
stack<int> s;
int k = 1;
// use tarjan's algorithm to extract the biconnected components
// and then use a dfs from the largest biconnected component
// and stores the edges in ans
```

```
pair<int, int> ans[MAX_SIZE];
int n, m;

void tarjan(int v, int pre) {
     visited[v] = k++;
     low[v] = visited[v];
     s.push(v);
     instack[v] = 1;
     for (auto np : g[v]) {
         int nv = np.first;
         if (nv != pre) {
            if (!visited[nv]) {
                tarjan(nv, v);
                low[v] = min(low[v], low[nv]);
            } else if (instack[nv]) {
                low[v] = min(low[v], visited[nv]);
            }
         }
     }

     if (low[v] == visited[v]) {
        while (s.top() != v) {
            cmpid[s.top()] = v;
            instack[s.top()] = 0;
            s.pop();
        }

        cmpid[s.top()] = v;
        instack[s.top()] = 0;
        s.pop();
     }
}

void dfs(int v, int pre) {
    visited[v] = low[v] = k++;
    for (auto np : g[v]) {
        if (np.first != pre) {
           if (!visited[np.first]) {
              dfs(np.first, v);
              low[v] = min(low[v], low[np.first]);
              ans[np.second].first = v;
              ans[np.second].second = np.first;
              if (low[np.first] > visited[v]) {
                  ans[np.second].first = np.first;
                  ans[np.second].second = v;
              }
           } else {
               low[v] = min(low[v], visited[np.first]);
```

```
                if (visited[v] > visited[np.first]) {
                   ans[np.second].first = v;
                   ans[np.second].second = np.first;
                }
            }
         }
     }
}
```

Euler path/cycle

```
#include <bits/stdc++.h>
#define MAX_SIZE 200011
using namespace std;

int b[MAX_SIZE], c[MAX_SIZE];
vector<int> disc;
int tmp[MAX_SIZE], f[MAX_SIZE];
int n;

int find(int x) {
    // cout << x << endl;
    if (x == f[x]) return x;
    return f[x] = find(f[x]);
}

void unionset(int x, int y) {
    int aa = find(x), bb = find(y);
    f[aa] = bb;
}

void preprocess() {
    int i;
    for (i = 0 ; i < MAX_SIZE; ++i) f[i] = i;
}

multiset<int> g[MAX_SIZE];
vector<pair<int, int>> ret;

void euler(int v) {
    // cout << "enter" << endl;
    while (!g[v].empty()) {
        int nv = *g[v].begin();
        g[nv].erase(g[nv].find(v));
        g[v].erase(g[v].find(nv));
        euler(nv);
```

```
            ret.emplace_back(v, nv);
        }
    }

    int main() {
        int i, n, j = 0;
        scanf("%d", &n);
        preprocess();
        for (i = 0; i < n-1; ++i) {
            scanf("%d", &b[i]);
            tmp[j++] = b[i];
        }

        for (i = 0 ; i < n-1; ++i) {
            scanf("%d", &c[i]);
            tmp[j++] = c[i];
        }

        for (i = 0 ; i < n-1; ++i) {
            if (b[i] > c[i]) {
                printf("-1\n");
                return 0;
            }
        }

        sort(tmp, tmp + j);

        for (i = 0 ; i < j; ++i) {
            if (i == 0 || tmp[i] != tmp[i-1]) {
                disc.push_back(tmp[i]);
            }
        }

        for (i = 0 ; i < n-1; ++i) {
            int idx1 = lower_bound(disc.begin(), disc.end(), b[i]) - disc.begin();
            int idx2 = lower_bound(disc.begin(), disc.end(), c[i]) - disc.begin();
            // add code here to add edges
            g[idx1].insert(idx2);
            g[idx2].insert(idx1);
            unionset(idx1, idx2);
        }

        int sz = disc.size();

        int x = find(0);

        for (i = 1; i < sz; ++i) {
            if (find(i) != x) {
```

```
            printf("-1\n");
            return 0;
        }
    }

    int start = 0;
    x = 0;
    for (i = 0 ; i < sz; ++i) {
        if ((int) g[i].size() % 2 == 1) {
            ++x;
            start = i;
        }
    }

    if (x > 2) {
        printf("-1\n");
        return 0;
    }
    // cout << "start= " << disc[start] << endl;
    euler(start);
    reverse(ret.begin(), ret.end());
    for (i = 0 ; i < (int) ret.size(); ++i) {
        if (i == 0) {
            printf("%d ", disc[ret[i].first]);
        }
        printf("%d ", disc[ret[i].second]);
    }
    printf("\n");
    return 0;

}
```

Kruskal tree

```
struct Construction {
    int tp;
    int k = 1;
    int f[MAX_SIZE];
    int parent[MAX_SIZE][19];
    vector<int> rt[MAX_SIZE >> 1];
    int dfsord[MAX_SIZE >> 1];
    int subsz[MAX_SIZE >> 1];
    Construction(int a) {
        this->tp = a;
    }
```

```
int find(int x) {
    if (f[x] == x) return x;
    return f[x] = find(f[x]);
}

void dfs(int v) {
    int i;
    dfsord[v] = k++;
    subsz[v] = 1;

    for (i = 1 ; i < 19; i++) {
        parent[v][i] = parent[parent[v][i-1]][i-1];
    }

    for (auto nv : rt[v]) {
        dfs(nv);
        subsz[v] = subsz[v] + subsz[nv];
    }
}

void build() {
    int i;
    for (int i = 1; i <= n; i++) {
        f[i] = i;
    }
    // the first tree
    if (tp == 0) {

        for (i = n; i >= 1; i--) {
            for (auto v : g[i]) {
                // only join from low to high
                if (i < v) {
                    v = find(v);
                    if (v == i) continue;
                    f[v] = i;
                    parent[v][0] = i;
                    rt[i].push_back(v);
                }
            }
        }

        dfs(1);
    } else {
    // the second tree
        for (i = 1; i <= n; i++) {
            for (auto v : g[i]) {
                // only join from low to high
```

```
                    if (i > v) {
                        v = find(v);
                        if (v == i) continue;
                        f[v] = i;
                        parent[v][0] = i;
                        rt[i].push_back(v);
                    }
                }
            }
            dfs(n);
        }
    }

    // give a point to start/end and an upper bound/lower bound
    // return to the uppermost point it can reach in its connected component
    int getrange(int x, int l) {
        int i;
        for (i = 18; i >= 0; i--) {
            if (parent[x][i] && ((tp == 0 && parent[x][i] >= l) || (parent[x][
i] <= l && tp == 1))) {
                x = parent[x][i];
            }
        }

        return x;
    }
};

Construction t1 = Construction(0);
Construction t2 = Construction(1);
```

Bellman ford K edges

```
dp[0][1] = 0;
    for (i = 1; i <= N; ++i) {
        for (j = 1; j <= N; ++j) {
            for (auto np : g[j]) {
                int nv = np.first;
                ll w = np.second;
                dp[i][nv] = min(dp[i-1][j] + w, dp[i][nv]);
            }
        }
    }
```

Floyd minimum cycle in undirected graph

```
while(scanf("%d %d",&n,&m)!=EOF)
    {
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                a[i][j]=a[j][i]=dis[i][j]=dis[j][i]=inf;
        while(m--)
        {
            int u,v,w;
            scanf("%d%d%d",&u,&v,&w);
            if(w<dis[u][v])
                a[u][v]=a[v][u]=dis[u][v]=dis[v][u]=w;
        }
        int ans=inf;
        for(int k=1;k<=n;k++)
        {
            for(int i=1;i<k;i++)
                for(int j=i+1;j<k;j++)//这里得是 i+1,防止重复
                    ans=min(ans,dis[i][j]+a[i][k]+a[k][j]);
            for(int i=1;i<=n;i++)
                for(int j=1;j<=n;j++)
                    dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
        }
        if(ans==inf)puts("It's impossible.");
        else printf("%d\n",ans);
    }
```

Smallest cycle in an undirected graph, BFS

```
int BFS(int src) {
    if (g[src].empty()) return MAX_SIZE;
    int ret = MAX_SIZE;
    memset(depth, -1, sizeof(depth));
    memset(visited, 0, sizeof(visited));
    queue<int> q;
    q.push(src);
    depth[src] = 0;
    while (!q.empty()) {
        auto p = q.front();
        q.pop();
```

```
            for (auto np : g[p]) {
                int nv = np.first, id = np.second;
                if (visited[id]) continue;
                if (depth[nv] == -1) {
                    depth[nv] = depth[p] + 1;
                    q.push(nv);
                    visited[id] = 1;
                } else {
                    ret = min(ret, depth[nv] + depth[p] + 1);
                }
            }
        }
        return ret;
}

int main() {
    for (i = 0 ; i < 1000; ++i) {
        ans = min(ans, BFS(i));
    }
    return 0;
}
```

Any Cycle in undirected graph

```
void dfs(int v) {
    if (flag) return;
    visited[v] = 1;
    st.push(v);
    instack[v] = 1;
    for (auto nv : g[v]) {
        if (instack[nv] && !flag) {
            flag = true;
            while (st.top() != nv) {
                instack[st.top()] = 0;
                cycle.push_back(st.top());
                st.pop();
            }
            cycle.push_back(st.top());
            instack[st.top()] = 0;
            st.pop();
            reverse(cycle.begin(), cycle.end());
        }
    }

    for (auto nv : g[v]) {
        if (!visited[nv]) {
            dfs(nv);
        }
    }
    if (!st.empty()) {
        st.pop();
    }
    instack[v] = 0;
}
```

Minimum Euler path

```
#include <bits/stdc++.h>
#define MOD 1000000007
#define MAXV 28
typedef long long ll;
using namespace std;

vector<pair<string, int>> g[MAXV];
int outdegree[MAXV], indegree[MAXV];
int f[MAXV];
bool visited[1011];
vector<string> ret;

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

void unionset(int a, int b) {
    int x = find(a), y = find(b);
    f[x] = y;
}

void init() {
     int i;
     memset(visited, false, sizeof(visited));
     for (i = 0 ; i < MAXV; i++) {
         g[i].clear();
         f[i] = i, outdegree[i] = indegree[i] = 0;
         ret.clear();
     }
}
```

```
void dfs(int v) {
    int i;
    for (i = 0; i < g[v].size(); i++) {
        if (!visited[g[v][i].second]) {
            visited[g[v][i].second] = true;
            int nv = (g[v][i].first[g[v][i].first.size() - 1]) - 'a';
            dfs(nv);
            ret.push_back(g[v][i].first);
        }
    }
}

char st[22];

int main() {
    int T, n, i;
    #ifdef DEBUG
    freopen("out.txt", "w", stdout);
    #endif
    scanf("%d", &T);
    while (T > 0) {
        scanf("%d", &n);
        init();
        set<char> h;
        for (i = 0 ; i < n; i++) {
            scanf("%s", st);
            string s = string(st);
            g[s[0] - 'a'].emplace_back(s, i);
            outdegree[s[0] - 'a']++, indegree[s[s.size() - 1] - 'a']++;
            h.insert(s[0]), h.insert(s[s.size() - 1]);
            unionset(s[0] - 'a', s[s.size() - 1] - 'a');
        }

        bool valid = true;

        int cng1 = 0, c1 = 0, sv = -1;
        for (i = 0 ; i < MAXV; i++) {
            if (outdegree[i] - indegree[i] == -1) {
                cng1++;
            } else if (outdegree[i] - indegree[i] == 1) {
                c1++;
                sv = i;
            } else if (outdegree[i] != indegree[i]) {
                valid = false;
            }
        }
        if (valid && ((c1 == 1 && cng1 == 1) || (c1 == 0 && cng1 == 0))) {
            valid = true;
        } else {
            valid = false;
        }

        unordered_set<int> prt;
        for (auto ch : h) {
            prt.insert(find(ch - 'a'));
        }

        if (prt.size() > 1) valid = false;
        if (sv == -1 && valid) {
            sv = *h.begin() - 'a';
        }


        if (!valid) {
            printf("***\n");
        } else {
            for (i = 0 ; i < MAXV; i++) sort(g[i].begin(), g[i].end());
            dfs(sv);
            reverse(ret.begin(), ret.end());
            printf("%s", ret[0].c_str());
            for (i = 1; i < ret.size(); i++) {
                printf(".%s", ret[i].c_str());
            }
            printf("\n");
        }
        T--;
    }
    return 0;

}
```

LCA

```
void dfs(int v, int pd, int dps) {
    height[v] = dps;
    parent[v][0] = pd;
    int i;
    for (i = 1; i <= logs[n]; i++) {
        parent[v][i] = parent[parent[v][i-1]][i-1];
        pmax[v][i] = max(pmax[parent[v][i-1]][i-1], pmax[v][i-1]);
    }
    for (auto nextp : graph[v]) {
        int nextv = nextp.first;
        ll wt = nextp.second;
        if (pd != nextv) {
            pmax[nextv][0] = wt;
            dfs(nextv, v, dps + 1);
        }
    }
}

ll LCA(int u, int v) {
    if (height[u] < height[v]) {
        swap(u, v);
    }

    int i;
    ll ans = 0ll;
    for (i = logs[n]; i >= 0; i--) {
        if (height[u] >= height[v] + (1 << i)) {
            ans = max(ans, pmax[u][i]);
            u = parent[u][i];
        }
    }

    if (u == v) {
        return ans;
    }

    for (i = logs[n]; i >= 0; i--) {
        if (parent[u][i] != parent[v][i]) {
            ans = max(ans, max(pmax[u][i], pmax[v][i]));
            u = parent[u][i];
            v = parent[v][i];
        }
    }

    ans = max(ans, max(pmax[u][0], pmax[v][0]));
    return ans;
}
```

Topsort least lexicographical order

```
#include <bits/stdc++.h>
#define MAX_SIZE 100011
using namespace std;
```

```
struct node {
    int id, v;
    bool operator < (node other) const {
        return v > other.v;
    }
};

int S, L, N;
char st[24], pt[24];
vector<string> vc;
int good[203][203], visited[MAX_SIZE];
vector<int> g[MAX_SIZE];
node nd[MAX_SIZE];
int ans[MAX_SIZE];
int pos[MAX_SIZE], indegree[MAX_SIZE];
map<string, int> mp;

int main() {
    int i, j;
    scanf("%d%d%d", &S, &L, &N);
    for (i = 0; i < S; ++i) {
        scanf("%s", st);
        vc.push_back(string(st));
    }

    sort(vc.begin(), vc.end());
    for (i = 0 ; i < S; ++i) {
        mp[vc[i]] = i;
    }

    for (i = 0 ; i < L; ++i) {
        scanf("%s%s", st, pt);
        int u = mp[string(st)], v = mp[string(pt)];
        good[u][v] = good[v][u] = 1;
    }

    for (i = 1; i <= N; ++i) {
        scanf("%s", st);
        int id = mp[string(st)];
        if (pos[id] != 0) {
            g[pos[id]].push_back(i);
        }
        pos[id] = i;
        nd[i].id = i, nd[i].v = id;
        for (j = 0 ; j < S; ++j) {
            if (j == id || !pos[j] || good[j][id]) continue;
            g[pos[j]].push_back(i);
        }
    }

    for (i = 1; i <= N; ++i) {
        for (auto nv : g[i]) {
            indegree[nv]++;
        }
    }

    priority_queue<node> q;
    for (i = 1; i <= N; ++i) {
        if (indegree[i] == 0) {
            q.push(nd[i]);
        }
    }

    vector<int> ret;
    while (!q.empty()) {
        auto curr = q.top();
        q.pop();
        ret.push_back(curr.v);
        for (auto nv : g[curr.id]) {
            indegree[nv]--;
            if (indegree[nv] == 0) {
                q.push(nd[nv]);
            }
        }
    }

    for (auto id : ret) {
        printf("%s ", vc[id].c_str());
    }
    printf("\n");
    return 0;
}
```

Tarjan articulation points

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

using namespace std;

class Graph {
public:
    Graph(int n) {
        this->graph = vector<vector<int> >(n, vector<int>());
        this->nV = n;
    }

    void addEdge(int v1, int v2) {
        assert(v1 < nV && v1 >= 0 && v2 < nV && v2 >= 0);
        graph[v1].push_back(v2);
        graph[v2].push_back(v1);
    }

    vector<int> tarjan_ap() {
        this->parent = vector<int>(nV, -1);
        this->visited = vector<int>(nV, 0);
        this->low = vector<int>(nV, 0);
        this->timest = 1;
        for (int i = 0 ; i < nV; i++) {
            if (!visited[i]) {
                dfs(i);
            }
        }

        vector<int> ret = vector<int>();
        for (auto nextv : arps) {
            ret.push_back(nextv);
        }
        return ret;
    }

private:
    vector<vector<int> > graph;
    int nV;
    // parent of a vertex
    vector<int> parent;
    // dfs order
    vector<int> visited;
    // low link value of a vertex
    vector<int> low;
    // articulation points
    unordered_set<int> arps;
    int timest;

    void dfs(int v) {
        visited[v] = timest++;
        low[v] = visited[v];
        int childcount = 0;
        bool isap = false;
        for (auto nextv : graph[v]) {
            if (!visited[nextv]) {
                parent[nextv] = v;
                childcount++;
                dfs(nextv);
                low[v] = min(low[v], low[nextv]);
                if (visited[v] <= low[nextv]) {
                    isap = true;
                }
            } else if (nextv != parent[v]) {
                low[v] = min(low[v], visited[nextv]);
            }
        }

        if (parent[v] == -1 && childcount >= 2) {
            arps.insert(v);
        }

        if (parent[v] != -1 && isap) {
            arps.insert(v);
        }
    }
};

int main(int argc, char *argv[]) {
    Graph g = Graph(8);
    g.addEdge(0,1);
    g.addEdge(0,2);
    g.addEdge(1,2);
    g.addEdge(2,3);
    g.addEdge(3,4);
    g.addEdge(4,5);
    g.addEdge(5,6);
    g.addEdge(4,6);
    g.addEdge(5,7);
    cout << "the articulation points are " << endl;
    for (auto v : g.tarjan_ap()) {
        cout << v << " ";
    }
    cout << endl;
    return 0;
}
```

BCC online

```
struct bcc_online {

    static const int size = 500005;

    void init() {
        for (int i = 0; i <= n; i++) {
            bcc[i] = comp[i] = i; // BCC and CC of a node
            link[i] = -1; // Link to parent in Connected Component
            sz[i] = 1; // Size of Connected Component
        }
        bridges = 0;
    }

    void add_edge(int u, int v) {
        u = get_bcc(u), v = get_bcc(v);
        if (u == v) return;
        int compu = get_comp(u), compv = get_comp(v);
        if (compu != compv) {
            bridges++;
            if(sz[compu] > sz[compv]) {
                swap(u, v);
                swap(compu, compv);
            }
            _make_root(u);
            link[u] = v;
            comp[u] = v;
            sz[compv] += sz[compu];
        }
        else _merge_path(u, v);
    }

    int get_bcc(int u) {
        if (u == -1) return -1;
        if (bcc[u] == u) return u;
        return bcc[u] = get_bcc(bcc[u]);
    }

    int get_comp(int u) {
        if (comp[u] == u) return u;
        return comp[u] = get_comp(comp[u]);
    }

private:

    int n, bridges, m;
    int bcc[size], comp[size], link[size], sz[size], vis[size];
    int current = 0;

    void _merge_path(int u, int v) {
        current++;
        vector<int> va, vb;
        int lca = -1;
        while (lca == -1) {
            if (u != -1) {
                u = get_bcc(u);
                va.push_back(u);
                if (vis[u] == current) lca = u;
                vis[u] = current;
                u = link[u];
            }
            if (v != -1) {
                vb.push_back(v);
                v = get_bcc(v);
                if (vis[v] == current) lca = v;
                vis[v] = current;
                v = link[v];
            }
        }
        for (auto &it : va) {
            bcc[it] = lca;
            if (it == lca) break;
            bridges--;
        }
        for (auto &it : vb) {
            bcc[it] = lca;
            if (it == lca) break;
            bridges--;
        }
    }

    void _make_root(int u) {
        u = get_bcc(u);
        int root = u, child = -1;
        while(u != -1)
        {
            int par = get_bcc(link[u]);
            link[u] = child;
            comp[u] = root;
            child = u;
            u = par;
        }
        sz[root] = sz[child];
    }
};
```

Flow

Dinics

```
#include <bits/stdc++.h>
#define MAX_SIZE 1222
typedef long long ll;
#define INF 100000000000
using namespace std;

struct Maxflow {
    struct edge {
        int from, to;
        ll flow, capacity;
    };

    // start and end point
    int s, t;

    // list array
    vector<edge> edg;

    // g reference to the ith vertex's edges
    vector<int> g[MAX_SIZE];

    // distance array and visited array
    int dist[MAX_SIZE], visited[MAX_SIZE];
    int cur[MAX_SIZE];

    void init() {
        edg.clear();
        int i;
        for (i = 0 ; i < MAX_SIZE; i++) {
            g[i].clear();
        }
    }

    void addedge(int from, int to, ll capacity) {
        edge e1 = edge{from, to, 0ll, capacity};
        edge e2 = edge{to, from, 0ll, 0ll};
        edg.push_back(e1), edg.push_back(e2);
        g[from].push_back((int) edg.size() - 2);
        g[to].push_back((int) edg.size() - 1);
    }

    // construct the level graph
    bool bfs() {
        memset(visited,0,sizeof(visited));
        memset(dist,0,sizeof(dist));
        queue<int> q;
        q.push(s);
        visited[s] = 1;
        dist[s] = 0;
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int i = 0 ; i < (int) g[v].size(); i++) {
                edge &e = edg[g[v][i]];
                int nxt = e.to;
                if (!visited[nxt] && e.capacity > e.flow) {
                    dist[nxt] = dist[v] + 1;
                    q.push(nxt);
                    visited[nxt] = 1;
                }
            }
        }

        return visited[t];
    }

    ll dfs(int x, ll cp) {
        if (x == t || cp == 0) {
            return cp;
        }

        ll flow = 0, newflow;
        for (int &y = cur[x]; y < (int) g[x].size(); y++) {
            edge &e = edg[g[x][y]];
            if (dist[x] + 1 == dist[e.to]) {
                ll minn = min(cp, e.capacity - e.flow);
                newflow = dfs(e.to, minn);
                if (newflow > 0) {
                    e.flow += newflow;
                    edg[g[x][y] ^1].flow -= newflow;
                    flow += newflow;
                    cp -= newflow;

                    if (cp == 0) {
                        break;
                    }
                }
            }
        }

        return flow;
    }

    ll Dinic(){
        ll flow=0;
```

```
        while(bfs()){
            memset(cur,0,sizeof(cur));
            flow += dfs(s,INF);
        }
        return flow;
    }
};

Maxflow mf;
```

MCMF (max cost change to negative sign)

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll INF = 10000000000;
const int maxn = 111;

struct Edge {
    int from, to;
    ll cap, flow, cost;
    Edge(int u, int v, ll c, ll f, ll w):from(u),to(v),cap(c),flow(f),cost(w)
    {}
};

struct MCMF {
    int n, m;
    int src, target;
    vector<Edge> edges;
    vector<int> G[maxn];
    int inq[maxn];
    ll d[maxn];
    int p[maxn];
    ll a[maxn];
    ll K;
    MCMF() {}

    void init(int n=maxn, ll K=INF) {
        this->n = n;
        for(int i = 0; i < n; i++) G[i].clear();
        edges.clear();
        this->K = K;
    }

    void addedge(int from, int to, ll cap, ll cost) {
        edges.push_back(Edge(from, to, cap, 0, cost));
        edges.push_back(Edge(to, from, 0, 0, -cost));
        m = edges.size();
        G[from].push_back(m-2);
        G[to].push_back(m-1);
    }

    bool SPFA(int s, int t, ll &flow, ll &cost) {
        for(int i = 0; i < n; i++) d[i] = INF;
        memset(inq, 0, sizeof(inq));
        d[s] = 0; inq[s] = 1; p[s] = 0; a[s] = INF;

        queue<int> Q;
        Q.push(s);
        while(!Q.empty()) {
            int u = Q.front(); Q.pop();
            inq[u] = 0;
            for(int i = 0; i < (int) G[u].size(); i++) {
                Edge& e = edges[G[u][i]];
                if(e.cap > e.flow && d[e.to] > d[u] + e.cost) {
                    d[e.to] = d[u] + e.cost;
                    p[e.to] = G[u][i];
                    a[e.to] = min(a[u], e.cap - e.flow);
                    if(!inq[e.to]) { Q.push(e.to); inq[e.to] = 1; }
                }
            }
        }
        if(d[t] == INF) return false;
        // note that everytime MCMF would find the cheapest path
        // for a unit flow
        if (cost + d[t] * a[t] > K) {
            flow += (K - cost) / d[t];
            cost += (K - cost) / d[t] * d[t];
            return false;
        }
        flow += a[t];
        cost += d[t] * a[t];
        for(int u = t; u != s; u = edges[p[u]].from) {
                edges[p[u]].flow += a[t];
                edges[p[u]^1].flow -= a[t];
        }
        return true;
    }

    ll mincostMaxflow(ll &cost) {
        ll flow = 0;
        cost = 0;
        while(SPFA(src, target, flow, cost));
        return flow;
    }
};
```

```
MCMF mf;
```

MCMF Dijkstra with potential

```
//MCF
#inclued <bits/stdc++.h>
namespace MCF {
    #define MAXN 100010
    #define MAXM 100010
    #define wint int
    #define cint int
    const wint wEPS = 0;
    const wint wINF = 1001001001;
    const cint cEPS = 0;
    const cint cINF = 1001001001;
    int n, m, ptr[MAXN], next[MAXM], zu[MAXM];
    wint capa[MAXM], tof;
    cint cost[MAXM], toc, d[MAXN], pot[MAXN];
    int vis[MAXN], pree[MAXN];
    void init(int _n) {
        n = _n; m = 0; memset(ptr, ~0, n << 2);
    }
    void ae(int u, int v, wint w, cint c) {
        next[m] = ptr[u]; ptr[u] = m; zu[m] = v; capa[m] = w; cost[m] = +c; ++
m;
        next[m] = ptr[v]; ptr[v] = m; zu[m] = u; capa[m] = 0; cost[m] = -
c; ++m;
    }
    bool spRep(int src, int ink, wint flo = wINF) {
        wint f;
        cint c, cc;
        int i, u, v;
        memset(pot, 0, n * sizeof(cint));
        for (bool cont = 1; cont; ) {
            cont = 0;
            for (u = 0; u < n; ++u) for (i = ptr[u]; ~i; i = next[i]) if (capa
[i] > wEPS) {
                if (pot[zu[i]] > pot[u] + cost[i] + cEPS) {
                    pot[zu[i]] = pot[u] + cost[i]; cont = 1;
                }
            }
        }
        for (toc = 0, tof = 0; tof + wEPS < flo; ) {
            typedef pair<cint,int> node;
            priority_queue< node,vector<node>,greater<node> > q;
            for (u = 0; u < n; ++u) { d[u] = cINF; vis[u] = 0; }
            for (q.push(make_pair(d[src] = 0, src)); !q.empty(); ) {
                c = q.top().first; u = q.top().second; q.pop();
                if (vis[u]++) continue;
                for (i = ptr[u]; ~i; i = next[i]) if (capa[i] > wEPS) {
                    cc = c + cost[i] + pot[u] - pot[v = zu[i]];
                    if (d[v] > cc) { q.push(make_pair(d[v] = cc, v)); pree[v]
= i; }
                }
            }
            if (!vis[ink]) return 0;
            f = flo - tof;
            for (v = ink; v != src; v = zu[i ^ 1]) { i = pree[v]; f = min(f,ca
pa[i]); }
            for (v = ink; v != src; v = zu[i ^ 1]) { i = pree[v]; capa[i] -
= f; capa[i ^ 1] += f; }
            tof += f;
            toc += f * (d[ink] - pot[src] + pot[ink]);
            for (u = 0; u < n; ++u) pot[u] += d[u];
        }
        return 1;
    }
}
// MCF::tof,toc will store the final flow and final cost
```

KM algorithm

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 1011;
const int inf = 0x3fffffff;

struct KM {
    // warning !! nx <= ny, otherwise, TLE!!!!!!
    int G[maxn][maxn];
    int m;
    int nx,ny;
    int link[maxn],lx[maxn],ly[maxn];
    int slack[maxn];
    bool visx[maxn],visy[maxn];

    bool dfs(int x) {
        visx[x]=1;
        for(int y=0;y<ny;y++){
            if(visy[y]) continue;
            int tmp=lx[x]+ly[y]-G[x][y];
            if(tmp==0){
                visy[y]=1;
                if(link[y]==-1||dfs(link[y])){
                    link[y]=x;
                    return true;
                }
            }
            else if(slack[y]>tmp) slack[y]=tmp;
        }
        return false;
    }

    int km() {
        memset(link,-1,sizeof(link));
        memset(ly,0,sizeof(ly));
        for(int i=0;i<nx;i++){
            lx[i]=-inf;
            for(int j=0;j<ny;j++){
                if(G[i][j]>lx[i]) lx[i]=G[i][j];
            }
        }
        for(int x=0;x<nx;x++){
            for(int i=0;i<ny;i++) slack[i]=inf;
            while(1){
                memset(visx,0,sizeof(visx));
                memset(visy,0,sizeof(visy));
                if(dfs(x)) break;
                int d=inf;
                for(int i=0;i<ny;i++){
                    if(!visy[i]&&d>slack[i]) d=slack[i];
                }
                for(int i=0;i<nx;i++){
                    if(visx[i]) lx[i]-=d;
                }
                for(int i=0;i<ny;i++){
                    if(visy[i]) ly[i]+=d;
                    else slack[i]-=d;
                }
            }
        }
        int res=0;
        for(int i=0;i<ny;i++){
            if(link[i]!=-1) res+=G[link[i]][i];
        }
        return res;
    }
};
```

```
KM mf;
```

Number theory

Matrix

```
struct Matrix{
    ll mat[300][300];
    int N;

    Matrix(int N_){
        N = N_;
        for(int i = 0; i < N; ++i){
            for(int j = 0; j < N; ++j){
                mat[i][j] = 0;
            }
        }
    }

    Matrix operator * (const Matrix& o) const {
        Matrix ret(N);
        for(int i = 0; i < N; ++i){
            for(int j = 0; j < N; ++j){
                for(int k = 0; k < N; ++k){
                    ret.mat[i][j] += (mat[i][k] * o.mat[k][j]) % MOD;
                    ret.mat[i][j] %= MOD;
                }
            }
        }
        return ret;
    }

    static Matrix identity(int N_){
        Matrix ret(N_);
        for(int i = 0; i < N_; ++i){
            ret.mat[i][i] = 1;
        }
        return ret;
    }

    Matrix operator ^ (ll k) const {
        Matrix ret = identity(N);
        Matrix a = *this;
        while(k){
            if(k & 1)ret = ret * a;
            a = a * a;
            k /= 2;
        }
        return ret;
    }

};
```

Period of some periodic function

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

using namespace std;

typedef long long ll;

ll f(ll v) {
    return (v + v / 1048576 + 12345) % (1ll << 40);
}

pair<ll, ll> cycle() {
    ll s0 = 1611516670;
    ll tortoise = f(s0), hare = f(tortoise);
    while (tortoise != hare) {
        tortoise = f(tortoise);
        hare = f(f(hare));
    }
    ll mu = 0;
    tortoise = s0;
    while (tortoise != hare) {
        tortoise = f(tortoise);
        hare = f(hare);
        mu += 1;
    }
    ll lam = 1;
    hare = f(tortoise);
    while (tortoise != hare) {
        hare = f(hare);
        lam += 1;
    }

    return make_pair(lam, mu);

}
```

```
int main() {
    auto p = cycle();
    cout << p.first << " " << p.second << endl;
    return 0;
}
```

FFT polynomial

```
#include <bits/stdc++.h>
#define ID if(0)
using namespace std;
const int maxn = 200005;
const double PI = acos(-1.0);
struct Complex{
    double x , y;
    Complex(){}
    Complex(double x_ , double y_) : x(x_) , y(y_) {}
    Complex operator - (const Complex& rhs) const {
        return Complex(x - rhs.x , y - rhs.y);
    }
    Complex operator + (const Complex& rhs) const {
        return Complex(x + rhs.x , y + rhs.y);
    }
    Complex operator * (const Complex& rhs) const {
        return Complex(x * rhs.x - y * rhs.y , x * rhs.y + y * rhs.x);
    }
};
char S[maxn] , T[maxn];
int Ssum[4][maxn];
vector<int> a[4] , b[4] , c[4];
int N , M , K;

void change(vector<Complex>& y , int len) {
    int i, j, k;
    for (i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) swap(y[i], y[j]);
        k = len / 2;
        while (j >= k) {
            j = j - k;
            k = k / 2;
        }
        if (j < k) j += k;
    }
}
```

```
void FFT(vector<Complex>& y , int len , int on) {
        change(y, len);
        for (int h = 2; h <= len; h <<= 1) {
            Complex wn(cos(2 * PI / h), sin(on * 2 * PI / h));
            for (int j = 0; j < len; j += h) {
                Complex w(1, 0);
                for (int k = j; k < j + h / 2; k++) {
                    Complex u = y[k];
                    Complex t = w * y[k + h / 2];
                    y[k] = u + t;
                    y[k + h / 2] = u - t;
                    w = w * wn;
                }
            }
        }
        if (on == -1) {
            for (int i = 0; i < len; i++) {
                y[i].x /= len;
            }
        }
    }

vector<int> multiply(vector<int> a , vector<int> b){
    int n = a.size() , m = b.size() , len = 1 , i;
    while(len < n + m)len <<= 1;
    vector<Complex> A(len) , B(len) , C(len);
    for(i = 0; i < n; ++i)A[i] = Complex(a[i] , 0);
    for(i = n; i < len; ++i)A[i] = Complex(0 , 0);
    for(i = 0; i < m; ++i)B[i] = Complex(b[i] , 0);
    for(i = m; i < len; ++i)B[i] = Complex(0 , 0);
    FFT(A , len , 1);
    FFT(B , len , 1);
    for(i = 0; i < len; ++i)C[i] = A[i] * B[i];
    FFT(C , len , -1);
    vector<int> c(len);
    for(i = 0; i < len; ++i){
        c[i] = (int)(C[i].x + 0.5);
    }
    return c;
}

void init(){
    int i , j;
    for(i = 0; i < N; ++i){
        if(S[i] == 'A')S[i] = '0';
        if(S[i] == 'T')S[i] = '1';
        if(S[i] == 'G')S[i] = '2';
        if(S[i] == 'C')S[i] = '3';
        for(j = 0; j <= 3; ++j){
            Ssum[j][i + 1] = Ssum[j][i];
        }
        ++Ssum[S[i] - '0'][i + 1];
    }
    for(i = 0; i < M; ++i){
        if(T[i] == 'A')T[i] = '0';
        if(T[i] == 'T')T[i] = '1';
        if(T[i] == 'G')T[i] = '2';
        if(T[i] == 'C')T[i] = '3';
    }
}

int main(){
    int i , j;
    scanf("%d %d %d" , &N , &M , &K);
    scanf("%s" , S);
    scanf("%s" , T);
    reverse(T , T + M);
    init();
    for(j = 0; j <= 3; ++j)a[j].clear() , b[j].clear() , c[j].clear();
    for(j = 0; j <= 3; ++j)a[j].resize(N) , b[j].resize(M);
    for(j = 0; j <= 3; ++j){
        for(i = 0; i < N; ++i){
            int l = max(0 , i - K) , r = min(N - 1 , i + K);
            a[j][i] = (Ssum[j][r + 1] > Ssum[j][l]);
        }
        for(i = 0; i < M; ++i){
            b[j][i] = (T[i] == '0' + j);
        }
    }
    for(j = 0; j <= 3; ++j){
        c[j] = multiply(a[j] , b[j]);
    }
    int ans = 0;
    for(i = 0; i <= N - M; ++i){
        int acc = 0;
        for(j = 0; j <= 3; ++j)acc += c[j][M + i - 1];
        ans += acc >= M;
    }
    printf("%d\n" , ans);
}
```

NTT arbitrary prime

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

```
using namespace std;
typedef long long ll;
const ll mod = 1e6 + 3;
const int maxn = 2e5 + 10;
int N , A , B , C;
ll l[maxn] , t[maxn] , fac[maxn * 2] , inv[maxn * 2];

// This is a NTT example, it computes the convolution of
// 2 polynomials 1e6 + 3
// this template runs NTT twice with 2 different primes and restore
// result with a larger prime MOD

namespace Polynomial_NTT {
    // we use 998244353 and 1004535809
    using ll = long long;
    int g = 3, P = 998244353;
    constexpr int MXLEN = 1 << 19;

    const ll MOD = 1002772198720536577ll;
    const ll check[] = {334257240187163831ll, 668514958533372747ll};

    inline ll mul(ll a, ll b) { // multiply a with b, modules MOD = 1002772198
720536577
        ll ans = 0;
        if(a < b) swap(a, b);
        while(b) {
            if(b & 1) ans = (ans + a) % MOD;
            a = a * 2 % MOD;
            b >>= 1;
        }
        return ans;
    }

    class poly_t : public vector<int> {
        public:
            poly_t() {}
            poly_t(initializer_list<int> list) {
                for(const int &val : list) push_back(val);
            }
            poly_t(const poly_t &A, int size) {*this = A, resize(size);}
    };

    inline int modAdd(int u, int v) {return (((u += v) >= P) && (u -= P)), u;}
    int getInv(int u) {
        return u == 1 ? 1 : static_cast<ll>(P - P / u) * getInv(P % u) % P;
    }
    int Pow(int u, int v) {
        int ans = 1;
        for(; v; v >>= 1, u = static_cast<ll>(u) * u % P)
            if(v & 1) ans = static_cast<ll>(u) * ans % P;
        return ans;
    }

    enum DFT_FLAG_t {NORMAL = 1, INVERSE = -1};
    void DFT(poly_t &A, int n, DFT_FLAG_t flag = NORMAL) {
        static int prevLen, rev[MXLEN + 1];
        if(prevLen != n) {
            prevLen = n;
            for(int i = 0; i < n; i++)
                rev[i] = (rev[i >> 1] >> 1) | ((i & 1) * (n >> 1));
        }
        if(static_cast<int>(A.size()) != n) A.resize(n);

        for(int i = 0; i < n; i++)
            if(i < rev[i]) swap(A[i], A[rev[i]]);
        for(int i = 1; i < n; i <<= 1) {
            ll wn = Pow(g, P - 1 + flag * (P - 1) / (i << 1));
            for(int j = 0; j < n; j += i << 1)
                for(int k = 0, w = 1; k < i; k++, w = wn * w % P) {
                    int x0 = A[j + k], x1 = static_cast<ll>(w) * A[i + j + k]
% P;
                    A[j + k] = modAdd(x0, x1);
                    A[i + j + k] = modAdd(x0, P - x1);
                }
        }
        if(flag == INVERSE) {
            ll tmp = getInv(n);
            for(int &val : A)
                val = tmp * val % P;
        }
    }

    poly_t multiply(poly_t A, poly_t B) {
        static poly_t C;
        int X = 1, n = A.size(), m = B.size();
        for(; X < n + m - 1; X <<= 1);
        C.resize(X);
        DFT(A, X, NORMAL), DFT(B, X, NORMAL);
        for(int i = 0; i < X; i++)
            C[i] = static_cast<ll>(A[i]) * B[i] % P;
        DFT(C, X, INVERSE), C.resize(n + m - 1);
        return C;
    }

}
Polynomial_NTT::poly_t h, g;
```

```
inline ll qpow(ll x , ll n){
    ll ret = 1ll;
    while(n){
        if(n % 2)ret = ret * x % mod;
        x = x * x % mod;
        n >>= 1;
    }
    return ret;
}

inline ll choose(int n , int r){
    if(n < r || n < 0 || r < 0)return 0;
    return fac[n] * (inv[n - r] * inv[r] % mod) % mod;
}

int main(){
    scanf("%d %d %d %d" , &N , &A , &B , &C);
    for(int i = 1; i <= N; ++i){
        scanf("%lld" , &l[i]);
    }
    for(int i = 1; i <= N; ++i){
        scanf("%lld" , &t[i]);
    }
    fac[0] = 1 , inv[0] = qpow(fac[0] , mod - 2);
    for(int i = 1; i < maxn * 2; ++i){
        fac[i] = fac[i - 1] * i % mod;
        inv[i] = qpow(fac[i] , mod - 2);
    }
    ll ans = 0;
    for(int i = 2; i <= N; ++i){
        // contribution of t[i] in F[N][N]
        ll v = qpow(A , N - i) * qpow(B , N - 1) % mod;
        v = v * t[i] % mod;
        v = v * choose(2 * N - 2 - i , N - i) % mod;
        ans += v;
        ans %= mod;
    }
    for(int i = 2; i <= N; ++i){
        // contribution of l[i] in F[N][N]
        ll v = qpow(A , N - 1) * qpow(B , N - i) % mod;
        v = v * l[i] % mod;
        v = v * choose(2 * N - 2 - i , N - i) % mod;
        ans += v;
        ans %= mod;
    }
    // calculate the coefficient of C in F[N][N]
    h.resize(N - 1); g.resize(N - 1);
    for(int i = 0; i <= N - 2; ++i){
        h[i] = qpow(A , i) * inv[i] % mod;
        g[i] = qpow(B , i) * inv[i] % mod;
    }
    // DO NTT twice for different primes
    Polynomial_NTT::P = 998244353;
    Polynomial_NTT::poly_t hg1 = Polynomial_NTT::multiply(h , g);
    Polynomial_NTT::P = 1004535809;
    Polynomial_NTT::poly_t hg2 = Polynomial_NTT::multiply(h , g);
    ll coe = 0;
    for(int i = 0; i <= 2 * N - 4; ++i){
        ll val = (Polynomial_NTT::mul(hg1[i] , Polynomial_NTT::check[0]) + Pol
ynomial_NTT::mul(hg2[i] , Polynomial_NTT::check[1])) % Polynomial_NTT::MOD % m
od;
        // restore (h * g)[i]
        coe += fac[i] * val % mod;
        coe %= mod;
    }
    coe = coe * C % mod;
    ans += coe;
    ans %= mod;
    printf("%lld\n" , ans);
}
```

NTT template normal

```
#include <bits/stdc++.h>
#define x first
#define y second
#define ID if(0)
#define sz(a) a.size()
using namespace std;
typedef long long ll;

const ll mod = 998244353;

inline ll qpow(ll x , ll n){
    ll ret = 1ll;
    while(n){
        if(n % 2)ret = ret * x % mod;
        x = x * x % mod;
        n >>= 1;
    }
    return ret;
}
vector<vector<ll>> poly;
```

```
void change(vector<ll>& y , int len) {
    int i, j, k;
    for (i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) swap(y[i], y[j]);
        k = len / 2;
        while (j >= k) {
            j = j - k;
            k = k / 2;
        }
        if (j < k) j += k;
    }
}

void NTT(vector<ll>& P , int op){
    int len = sz(P) , i , j , k;
    change(P , len);
    for(i = 1; i < len; i <<= 1){
        ll gn = qpow(3ll , (mod - 1) / (i << 1));
        if(op == -1)gn = qpow(gn , mod - 2);
        for(j = 0; j < len; j += (i << 1)){
            ll g = 1;
            for(k = 0; k < i; ++k , g = g * gn % mod){
                ll x = P[j + k] , y = g * P[j + k + i] % mod;
                P[j + k] = (x + y) % mod;
                P[j + k + i] = (x - y + mod) % mod;
            }
        }
    }
    if(op == -1){
        ll inv = qpow(len , mod - 2);
        for(i = 0; i < len; ++i)P[i] = P[i] * inv % mod;
    }
}

vector<ll> multiply(vector<ll> aa , vector<ll> bb){
    int n = sz(aa) , m = sz(bb) , len = 1 , i;
    while(len < n + m)len <<= 1;
    vector<ll> a(len) , b(len);
    for(i = 0; i < n; ++i)a[i] = aa[i];
    for(i = 0; i < m; ++i)b[i] = bb[i];
    NTT(a , 1);
    NTT(b , 1);
    vector<ll> c(len);
    for(i = 0; i < len; ++i)c[i] = a[i] * b[i] % mod;
    NTT(c , -1);
    while(!c.empty() && c.back() == 0)c.pop_back();
    return c;
}

vector<ll> solve(int l , int r){
    vector<ll> ret;
    if(l == r){
        for(ll i : poly[l])ret.push_back(i);
        return ret;
    }
    int mid = (l + r) >> 1;
    ret = multiply(solve(l , mid) , solve(mid + 1 , r));
    return ret;
}

int N;
map<int , int> cnt;

int main(){
    int i;
    scanf("%d" , &N);
    for(i = 1; i <= N; ++i){
        int x; scanf("%d" , &x); ++cnt[x];
    }
    for(auto& e : cnt){
        vector<ll> vec(e.y + 1 , 1);
        poly.push_back(vec);
        ID{
            printf("for %d\n" , e.x);
            for(ll i : poly.back()){
                printf("%lld " , i);
            }
            printf("\n");
        }
    }
    vector<ll> ans = solve(0 , sz(poly) - 1);
    ID for(int i = 0; i < sz(ans); ++i){
        printf("ans[%d] = %lld\n" , i , ans[i]);
    }
    int id = sz(ans) / 2;
    printf("%lld\n" , ans[id]);
}
```

NTT faster

```
#pragma GCC optimize(3)
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")
```

```cpp
#pragma comment(linker, "/stack:200000000")
#include <bits/stdc++.h>
#define sz(a) a.size()
using namespace std;
typedef long long ll;
typedef vector<int> vi;
const int mod = 998244353;
const int g = 3;
const int maxn = 1e5 + 10;
int fac[maxn * 2] , inv[maxn * 2] , N , LEN[maxn * 2];

inline int C(int n , int r){
    if(n < r || n < 0 || r < 0)return 0;
    return (ll)fac[n] * ((ll)inv[n - r] * inv[r] % mod) % mod;
}

inline ll qpow(ll b , ll p){
    ll res = 1;
    if (p < 0) p += mod - 1;
    while (p) {
        if (p & 1) (res *= b) %= mod;
        (b *= b) %= mod;
        p /= 2;
    }
    return res;
}

inline int add(int x , int y){
    int ret = x + y;
    if(ret > mod)ret -= mod;
    return ret;
}

inline int sub(int x , int y){
    int ret = x - y;
    if(ret < 0)ret += mod;
    return ret;
}

inline int mul(int x , int y){
    ll ret = (ll)x * y % mod;
    return ret;
}

int nearestPowerOfTwo (int n) {
    int ans = 1;
    while (ans < n) ans <<= 1;
    return ans;
}

void ntt (vector<int> &X, int inv) {
    int n = X.size();

    for (int i = 1, j = 0; i < n - 1; ++i) {
        for (int k = n >> 1; (j ^= k) < k; k >>= 1);
        if(i < j) swap(X[i], X[j]);
    }

    vector<ll> wp(n >> 1, 1);
    for (int k = 1; k < n; k <<= 1) {
        ll wk = qpow(g, inv * (mod - 1) / (k << 1));

        for (int j = 1; j < k; ++j)
            wp[j] = wp[j - 1] * wk % mod;

        for (int i = 0; i < n; i += k << 1) {
            for (int j = 0; j < k; ++j) {
                int u = X[i + j], v = X[i + j + k] * wp[j] % mod;
                X[i + j] = u + v < mod ? u + v : u + v - mod;
                X[i + j + k] = u - v < 0 ? u - v + mod : u - v;
            }
        }
    }

    if (inv == -1) {
        ll nrev = qpow(n, mod - 2);
        for(int i = 0; i < n; ++i)
            X[i] = X[i] * nrev % mod;
    }
}

vector<int> convolution (vector<int> A, vector<int> B){
    int sz = A.size() + B.size() - 1;
    int size = nearestPowerOfTwo(sz);

    A.resize(size), B.resize(size);
    ntt(A, 1), ntt(B, 1);

    for(int i = 0; i < size; i++)
        A[i] = 1ll * A[i] * B[i] % mod;

    ntt(A, -1);
    A.resize(sz);

    return A;
}
```

```
int main(){
    scanf("%d" , &N);
    vector<int> a(N) , b(N);
    for(int i = 0; i < N; ++i){
        scanf("%d %d" , &a[i] , &b[i]);
    }
    fac[0] = 1;
    inv[0] = qpow(fac[0] , mod - 2);
    for(int i = 1; i < maxn * 2; ++i){
        fac[i] = (ll)fac[i - 1] * i % mod;
        inv[i] = qpow(fac[i] , mod - 2);
        LEN[i] = qpow(i , mod - 2);
    }
    vector<int> ans(1 , 1);
    for(int i = 0; i < N; ++i){
        int M = sz(ans);
        int from = max(0 , b[i] - M + 1);
        int to = min(a[i] + b[i] , M + a[i] - 1);
        vector<int> ans2 , F(to - from + 1);
        for(int j = 0; j <= to - from; ++j){
            F[j] = C(a[i] + b[i] , from + j);
        }
        ans2 = convolution(F , ans);
        ans.resize(M + a[i] - b[i]);
        for(int j = 0 , k = b[i] - from; j < M + a[i] - b[i]; ++j , ++k){
            // start from term = C(a[i] + b[i] , b[i])
            ans[j] = ans2[k];
        }
    }
    int ret = 0;
    for(auto& x : ans){
        ret = add(ret , x);
    }
    printf("%d\n" , ret);
}
```

SIEVE partial sum

```
#include <bits/stdc++.h>
#define MAX_SIZE 10000011
using namespace std;

int p[MAX_SIZE], v[MAX_SIZE], visited[MAX_SIZE];
int main() {
    int i, j;
    int a, b, c, d;
    scanf("%d%d%d%d", &a, &b, &c, &d);
    for (i = 2; i < MAX_SIZE; ++i) {
        p[i] = 1;
    }

    for (i = 2; i < MAX_SIZE; ++i) {
        if (!visited[i]) {
            for (j = i; j < MAX_SIZE; j += i) {
                visited[j] = 1;
                p[j] *= i;
                v[j]++;
            }
        }
    }

    long long ans = 1ll * (d - c + 1) * (b - a + 1);
    for (i = 2; i < MAX_SIZE; ++i) {
        if (p[i] == i) {
            if (v[i] & 1) {
                ans = ans - 1ll * ((d / p[i]) - (c - 1) / p[i]) * ((b / p[i])
- (a - 1) / p[i]);
            } else {
                ans = ans + 1ll * ((d / p[i]) - (c - 1) / p[i]) * ((b / p[i])
- (a - 1) / p[i]);
            }
        }
    }

    printf("%lld\n", ans);
    return 0;
}
```

Gaussian elimination

The input to the function **gauss** is the system matrix aa. The last column of this matrix is vector bb.The function returns the number of solutions of the system $(0,1,\text{or } \infty)$. If at least one solution exists, then it is returned in the vector ans.

```
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity or a big number

int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

Extended Euclidian

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
```

Z2 basis

```
struct z2_basis {
    ll a[N];
    bool flag = false; // if the set is linearly dependent
    void insert(ll x) {
        int i;
        for(i = N - 1; i >= 0; i--) {
            if(x & (1ll << i)) {
                if(!a[i]) {
                    a[i] = x;
                    return;
                } else {
                    x ^=a[i];
                }
            }
        }

        flag = true;
    }

    bool check(ll x) {
        if (x == 0) return true;
        int i;
        for(i = N - 1; i >= 0; i--) {
            if (x & (1ll<<i)) {
                if(!a[i]) {
                    return false;
                } else {
                    x^=a[i];
                }
            }
        }

        return true;
    }

    ll qmax(ll res=0) {
        int i;
        for(i = N - 1; i >= 0; i--) {
            res = max(res, res ^ a[i]);
        }
        return res;
    }


    ll qmin() {
        int i;
        if(flag) {
            return 0;
        }

        for(i = 0; i <= N - 1; i++) {
            if(a[i]) return a[i];
        }
    }
    // return the dimension of the set
    int dim() {
        int i, cnt = 0;
        for (i = N - 1; i >= 0; --i) {
            if (a[i]) ++cnt;
        }

        return cnt;
    }
};
```

Euler totient

$n\cdot(1-1/p_1)\cdot(1-1/p_2)\cdots(1-1/p_k)$

```
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    phi[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
        phi[i] = i - 1;

    for (int i = 2; i <= n; i++)
        for (int j = 2 * i; j <= n; j += i)
              phi[j] -= phi[i];
}



int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}
```