```
/*
  0. 数组空间是否足够? 数组空间是否足够? 数组空间是否足够? 数据范围开int还是long long?

  1. 多想一点, 少错一点!
  是否有你的算法的反例?

  2. 注意边界条件!
  n = 1考虑了吗? p = 0考虑了吗? 是否对有关-1的东西有特判?

  3. 不要犯愚蠢的错误!
  时间复杂度有没有写假? 空间足够吗? 精度是否正确?
*/

#include <bits/extc++.h>
using namespace std;
using ll = long long;
using ull = unsigned long long;
namespace pbds = __gnu_pbds;

int main() {
        cin.tie(nullptr)->sync_with_stdio(false);

        return 0;
}
```

板子

目录

    - 单调栈、单调队列
- 图论
    - Dijkstra
    - Floyd
    - SPFA
    - 或值最短路
    - Tarjan全家桶
    - 同余最短路
    - Kruskal
    - Prim
    - Boruvka
    - Dinic
    - 匈牙利算法
    - 2-SAT
- 搜索
    - 迭代加深
- 动态规划
    - 背包DP
    - 区间DP
    - 数位DP
    - 计数DP
    - 线段树优化DP
- 数学
    - 快速幂和模整数类
    - 组合数
    - EXGCD
    - 中国剩余定理
    - 斯特林数
    - 质数筛
    - 多项式乘法
    - 线性基
    - 矩阵乘法、矩阵快速幂
    - 构造增广状态转移矩阵
    - Lehmer码
- 计算几何
    - 平面凸包
- 字符串
    - KMP
    - Z函数/扩展KMP
    - 字典树
- 贪心
    - 一组交叉直线经过所有点
- 二分

让代码跑起来

编译运行

```
g++ -std=c++20 test.cpp -o test.out
./test.out < input.txt
```

对拍器

```
# 请根据实际的C++版本修改-std版本
g++ generator.cpp -o generator.out -std=c++20 -O2 || exit 1
g++ solution.cpp -o solution.out -std=c++20 -O2 || exit 1
g++ brute_force.cpp -o brute_force.out -std=c++20 -O2 || exit 1
mkdir -p testcases
count=1
while true; do
    echo "Running test $count"
    ./generator.out > testcases/input.txt
    timeout 2s ./solution.out < testcases/input.txt > testcases/output_solution.txt
    exit_code=$?
    if [ $exit_code -eq 124 ]; then
        echo -e "\033[31mTime Limit Exceeded on solution\033[0m"
        break
    elif [ $exit_code -eq 139 ]; then
        echo -e "\033[31mRuntime Error (Segmentation fault) on solution\033[0m"
        cat testcases/input.txt
        break
    elif [ $exit_code -eq 134 ]; then
        echo -e "\033[31mRuntime Error (Aborted) on solution\033[0m"
        cat testcases/input.txt
        break
    elif [ $exit_code -eq 136 ]; then
        echo -e "\033[31mRuntime Error (Floating point exception) on solution\033[0m"
        cat testcases/input.txt
        break
    fi
    timeout 2s ./brute_force.out < testcases/input.txt > testcases/output_brute_force.txt
    if [ $exit_code -eq 124 ]; then
        echo -e "\033[31mTime Limit Exceeded on brute force\033[0m"
        break
```

```
    fi
    if ! diff -wB testcases/output_solution.txt testcases/output_brute_force.txt > /dev/null; then
        echo -e "\033[31mWrong Answer on test case $count\033[0m"
        echo "Input:"
        cat testcases/input.txt
        echo -e "\nSolution output:"
        cat testcases/output_solution.txt
        echo -e "\nBrute force output:"
        cat testcases/output_brute_force.txt
        break
    fi
    echo -e "\033[32mAccepted\033[0m"
    ((count++))
done
```

数据结构

线段树

```
struct SegTree {
        const int n;
        SegTree(const vector<ll> &v)
                : n(v.size()), s(v.size() * 4), l(v.size() * 4) {
                _b(v, 0, 0, n);
        }
        SegTree(int sz) : n(sz), s(sz * 4, 0), l(sz * 4, 0) {}
        ll query(int l, int r) {
                if(l == r) return 0;
                return _q(l, r, 0, 0, n);
        }
        void update(int l, int r, ll d) {
                if(l == r) return;
                _u(l, r, d, 0, 0, n);
        }
private:
        vector<ll> s, l;
        void _b(const vector<ll> &v, int r, int rl, int rr) {
                if(rr - rl == 1) {
                        s[r] = v[rl];
                        return;
                }
                int m = (rl + rr) / 2;
                _b(v, r * 2 + 1, rl, m);
                _b(v, r * 2 + 2, m, rr);
                s[r] = s[r * 2 + 1] + s[r * 2 + 2];
        }
        void _pd(int r, int rl, int rr) {
                if(l[r] == 0) return;
                int m = (rl + rr) / 2;
                s[r * 2 + 1] += l[r] * (m - rl);
                l[r * 2 + 1] += l[r];
```

```
                s[r * 2 + 2] += l[r] * (rr - m);
                l[r * 2 + 2] += l[r];
                l[r] = 0;
        }
        ll _q(int ql, int qr, int r, int rl, int rr) {
                if(ql <= rl && qr >= rr) return s[r];
                _pd(r, rl, rr);
                int m = (rl + rr) / 2;
                ll ans = 0;
                if(ql < m) {
                        ans += _q(ql, qr, r * 2 + 1, rl, m);
                }
                if(qr > m) {
                        ans += _q(ql, qr, r * 2 + 2, m, rr);
                }
                return ans;
        }
        // 区间修改
        void _u(int ul, int ur, ll d, int r, int rl, int rr) {
                if(ul <= rl && ur >= rr) {
                        s[r] += d * (rr - rl);
                        l[r] += d;
                        return;
                }
                _pd(r, rl, rr);
                int m = (rl + rr) / 2;
                if(ul < m) {
                        _u(ul, ur, d, r * 2 + 1, rl, m);
                }
                if(ur > m) {
                        _u(ul, ur, d, r * 2 + 2, m, rr);
                }
                s[r] = s[r * 2 + 1] + s[r * 2 + 2];
        }
        // 单点修改
        void __u(int x, ll v, int r, int rl, int rr) {
                if(rl == rr - 1) {
                        s[r] += v;
                        return;
                }
                int m = (rl + rr) / 2;
                if(x < m) {
                        __u(x, v, r * 2 + 1, rl, m);
                } else {
                        __u(x, v, r * 2 + 2, m, rr);
                }
                s[r] = s[r * 2 + 1] + s[r * 2 + 2];
        }
};
```

主席树

```
// persistent segment tree
struct PST {
```

```
        const int n;
        PST(int n_) : n(n_) {
                nodes.reserve(2 * n * (int)log2(n));
                vers.reserve(n + 1);
                vers.push_back(_build(0, n - 1));
        }
        void update(int k) {
                vers.push_back(_update(k, vers.back(), 0, n - 1));
        }
        // 这里为解决静态区间第k小问题设计，传入k, l, r
        int query(int k, int ver1, int ver2) const {
                return _query(k, vers[ver1 - 1], vers[ver2], 0, n - 1);
        }
private:
        struct Node {
                int l, r;
                int cnt;
        };
        vector<Node> nodes;
        vector<int> vers;
        int _build(int l, int r) {
                int ret = nodes.size();
                if(l == r) {
                        nodes.emplace_back(-1, -1, 0);
                        return ret;
                }
                nodes.emplace_back(-1, -1, 0);
                int mid = (l + r) >> 1;
                nodes[ret].l = _build(l, mid);
                nodes[ret].r = _build(mid + 1, r);
                return ret;
        }
        int _update(int k, int root, int l, int r) {
                int ret = nodes.size();
                nodes.emplace_back(nodes[root]);
                nodes[ret].cnt++;
                if(l == r) {
                        return ret;
                }
                int mid = (l + r) >> 1;
                if(k <= mid) {
                        nodes[ret].l = _update(k, nodes[root].l, l, mid);
                } else {
                        nodes[ret].r = _update(k, nodes[root].r, mid + 1, r);
                }
                return ret;
        }
        int _query(int k, int root1, int root2, int l, int r) const {
                if(l == r) return l;
                int mid = (l + r) >> 1;
                int cntl = nodes[nodes[root2].l].cnt - nodes[nodes[root1].l].cnt;
                if(k <= cntl) {
                        return _query(k, nodes[root1].l, nodes[root2].l, l, mid);
                } else {
                        return _query(k - cntl, nodes[root1].r, nodes[root2].r, mid + 1, r);
                }
```

```
        }
};
/*
struct PST {
        PST() : node_cnt(0), len(0) {}
        PST(const vector<ll> &a) {
                build(a);
        }
        void build(const vector<ll> &a) {
                ys = a;
                ind = a;
                sort(ind.begin(), ind.end());
                ind.erase(unique(ind.begin(), ind.end()), ind.end());
                len = ind.size();
                node_cnt = 0;
                ll n = a.size();
                sum.assign(n * 50, 0);
                lson.assign(n * 50, 0);
                rson.assign(n * 50, 0);
                roots.assign(n + 1, 0);
                roots[0] = _build(0, len - 1);
                for(ll i = 1; i <= n; i++) {
                        roots[i] = _update(_getid(a[i - 1]), 0, len - 1, roots[i - 1]);
                }
        }
        // 查询区间第k小
        ll qmink(int L, int R, int k) {
                return ind[_query(0, len - 1, roots[L - 1], roots[R], k)];
        }
        // 查询区间第k大
        ll qmaxk(int L, int R, int k) {
                int n = R - L + 1;
                return qmink(L, R, n - k + 1);
        }
        // 查询前缀[1,pos]中小于x的元素个数
        ll qltx(int pos, ll x) {
                int idx = _getid(x + 1) - 1; // 离散化位置
                if(roots[pos] == 0 || idx < 0) return 0;
                return _getcnt(roots[pos], 0, len - 1, idx);
        }
        // 查询区间[L,R]中值在[l_val, r_val]的元素个数
        ll qcnt(int L, int R, ll l_val, ll r_val) {
                ll r_high = qltx(R, r_val);
                ll r_low = qltx(R, l_val - 1);
                ll l_high = qltx(L - 1, r_val);
                ll l_low = qltx(L - 1, l_val - 1);
                return (r_high - r_low) - (l_high - l_low);
        }

private:
        vector<ll> ind;         // 离散化数组
        vector<ll> ys;          // 原始数组
        vector<ll> sum;         // 节点计数
        vector<ll> lson;        // 左子节点索引
        vector<ll> rson;        // 右子节点索引
        vector<ll> roots;       // 各版本根节点
```

```
        ll len;                         // 离散值域大小
        ll node_cnt;            // 节点总数
        // 获取离散化下标
        ll _getid(ll x) {
                return lower_bound(ind.begin(), ind.end(), x) - ind.begin();
        }
        // 构建空树
        ll _build(ll l, ll r) {
                ll id = ++node_cnt;
                if(l == r) return id;
                ll mid = (l + r) >> 1;
                lson[id] = _build(l, mid);
                rson[id] = _build(mid + 1, r);
                return id;
        }
        // 更新树版本
        ll _update(ll pos, ll l, ll r, ll pre) {
                ll id = ++node_cnt;
                lson[id] = lson[pre];
                rson[id] = rson[pre];
                sum[id] = sum[pre] + 1;
                if(l == r) return id;
                ll mid = (l + r) >> 1;
                if(pos <= mid)
                        lson[id] = _update(pos, l, mid, lson[pre]);
                else
                        rson[id] = _update(pos, mid + 1, r, rson[pre]);
                return id;
        }

        // 查询第k小
        ll _query(ll l, ll r, ll L_id, ll R_id, ll k) {
                if(l == r) return l;
                ll mid = (l + r) >> 1;
                ll left_count = sum[lson[R_id]] - sum[lson[L_id]];
                if(left_count >= k)
                        return _query(l, mid, lson[L_id], lson[R_id], k);
                else
                        return _query(mid + 1, r, rson[L_id], rson[R_id], k - left_count);
        }

        // 计算值<=x的元素个数
        ll _getcnt(ll id, ll l, ll r, ll x) {
                if(l == r) return sum[id];
                ll mid = (l + r) >> 1;
                if(x <= mid)
                        return _getcnt(lson[id], l, mid, x);
                else
                        return sum[lson[id]] + _getcnt(rson[id], mid + 1, r, x);
        }
};
*/
```

树状数组（单点、区间）

```
// 单点
struct Fenwick {
        const int n;
        Fenwick(int n)
                : n(n), nums(n + 1) {}
        ll query(int x) const {
                ll ans = 0;
                for(; x; x -= lbit(x)) {
                        ans += nums[x];
                }
                return ans;
        }
        void update(int x, ll v) {
                for(; x <= n; x += lbit(x)) {
                        nums[x] += v;
                }
        }
private:
        vector<ll> nums;
        static int lbit(int x) {
                return x & -x;
        }
};
// 区间
struct RangeFenwick {
        const int n;
        RangeFenwick(int n)
                : n(n), f1(n), f2(n) {}
        void update(int l, int r, ll v) {
                _update(l, v);
                _update(r + 1, -v);
        }
        ll query(int l, int r) const {
                return _query(r) - _query(l - 1);
        }
private:
        Fenwick f1, f2;
        void _update(int x, ll v) {
                f1.update(x, v);
                f2.update(x, v * (x - 1));
        }
        ll _query(int x) const {
                return f1.query(x) * x - f2.query(x);
        }
};
```

差分数组

使用差分数组区间加等差数列

```
// 二次差分
vector<ll> diff(n + 3, 0);
// [1,l]第i项加i
```

```
diff[1] += val;
diff[l + 1] -= val;
// [r+1,l+r]第i项减i-r
diff[r + 1] -= val;
diff[l + r + 1] += val;
// 还原
for(int _ = 0; _ < 2; ++_) {
        for(int i = 1; i < n + 3; ++i) {
                diff[i] += diff[i - 1];
        }
}
```

树状数组在线处理

```
struct ArithmeticFenwick {
        ArithmeticFenwick(int n)
                : f1(n + 2), f2(n + 2) {}
        // a*idx+b
        void update(int l, int r, ll a, ll b) {
                f1.update(l, b);
                f1.update(r + 1, -b - a * (r - l + 1));
                f2.update(l, a);
                f2.update(r + 1, -a);
        }
        ll query(ll idx) const {
                return f1.query(idx) + idx * f2.query(idx);
        }
private:
        Fenwick f1, f2;
};
```

分块

```
struct Block {
        Block() : l(-1), r(-1) {}
        // 注意左闭右开
        Block(int l, int r, const vector<ll> &nums_)
                : nums(r - l), lazy(0), sum(0), l(l), r(r) {
                for(int i = l; i < r; ++i) {
                        nums[i - l] = nums_[i];
                        sum += nums[i - l];
                }
        }
        // 注意左闭右开
        void update(int ul, int ur, ll dval) {
                if(ul > r || ur < l) {
                        return;
                }
                if(ul <= l && ur >= r) {
                        lazy += dval;
                        sum += dval * (r - l);
                        return;
```

```
                }
                for(int i = max(ul, l); i < min(ur, r); ++i) {
                        nums[i - l] += dval;
                }
                sum += dval * (min(ur, r) - max(ul, l));
        }
        // 注意左闭右开
        ll query(int ql, int qr) const {
                if(ql >= r || qr <= l) {
                        return 0;
                }
                if(ql <= l && qr >= r) {
                        return sum;
                }
                ll ans = lazy * (min(qr, r) - max(ql, l));
                for(int i = max(ql, l); i < min(qr, r); ++i) {
                        ans += nums[i - l];
                }
                return ans;
        }
private:
        vector<ll> nums;
        ll lazy;
        ll sum;
        int l, r;
};
struct BlockedArray {
        BlockedArray(const vector<ll> &nums) {
                int r = nums.size();
                int len = sqrt(r + 0.5);
                int size = (r + len - 1) / len;
                blocks.reserve(size);
                for(int i = 0; i < size; ++i) {
                        int ll = i * len, rr = min((i + 1) * len, r);
                        blocks.emplace_back(ll, rr, nums);
                }
        }
        void update(int l, int r, ll d) {
                for(auto &blk : blocks) {
                        blk.update(l, r, d);
                }
        }
        ll query(int l, int r) const {
                ll ans = 0;
                for(const auto &blk : blocks) {
                        ans += blk.query(l, r);
                }
                return ans;
        }
        vector<Block> blocks;
};
```

并查集

```
// 按秩合并
struct DSU {
	DSU(int n) : fa(n), rk(n, 1) {
		iota(fa.begin(), fa.end(), 0);
	}
	int find(int x) {
		if(fa[x] != x) {
			fa[x] = find(fa[x]);
		}
		return fa[x];
	}
	void connect(int x, int y) {
		x = find(x);
		y = find(y);
		if(x == y) return;
		if(rk[x] < rk[y]) {
			swap(x, y);
		}
		if(rk[x] == rk[y]) {
			rk[x]++;
		}
		fa[y] = x;
	}
	bool is_connected(int x, int y) {
		return find(x) == find(y);
	}
private:
	vector<int> fa, rk;
};
// 随机合并
struct DSU_Random {
	DSU_Random(int n) : fa(n) {
		iota(fa.begin(), fa.end(), 0);
	}
	int find(int x) {
		if(fa[x] != x) {
			fa[x] = find(fa[x]);
		}
		return fa[x];
	}
	bool is_connected(int x, int y) {
		return find(x) == find(y);
	}
	void connect(int x, int y) {
		int fx = find(x), fy = find(y);
		if(rnd() % 2) {
			fa[fx] = fy;
		} else {
			fa[fy] = fx;
		}
	}
private:
	static inline mt19937 rnd{ (unsigned int)chrono::steady_clock::now()
		.time_since_epoch().count() };
	vector<int> fa;
```

```
};
```

无旋Treap

```
struct Treap {
        Treap() : rt(nullptr) {}
        ~Treap() {
                if(rt) {
                        _dtor(rt);
                }
        }
        void insert(int v) {
                auto tmp = _sval(rt, v);
                auto l = _sval(tmp.first, v - 1);
                Node *node = l.second;
                if(!l.second) {
                        node = new Node(v);
                } else {
                        l.second->cnt++;
                        l.second->usize();
                }
                Node *lc = _merge(l.first, node);
                rt = _merge(lc, tmp.second);
        }
        void erase(int v) {
                auto tmp = _sval(rt, v);
                auto l = _sval(tmp.first, v - 1);
                if(l.second->cnt > 1) {
                        l.second->cnt--;
                        l.second->usize();
                        l.first = _merge(l.first, l.second);
                } else {
                        if(tmp.first == l.second) {
                                tmp.first = nullptr;
                        }
                        delete l.second;
                        l.second = nullptr;
                }
                rt = _merge(l.first, tmp.second);
        }
        // 注意1-based
        int qrv(int v) {
                return _qrv(rt, v);
        }
        // 注意1-based
        int qvr(int r) {
                return _qvr(rt, r);
        }
        // 注意是小于的，而不是大于等于的
        int lower_bound(int v) {
                auto tmp = _sval(rt, v - 1);
                int ret = _qvr(tmp.first, tmp.first->size);
                rt = _merge(tmp.first, tmp.second);
```

```
		return ret;
	}
	int upper_bound(int v) {
		auto tmp = _sval(rt, v);
		int ret = _qvr(tmp.second, 1);
		rt = _merge(tmp.first, tmp.second);
		return ret;
	}
private:
	struct Node {
		int val;
		int cnt;
		int size;
		int prio;
		Node *left, *right;
		Node(int v)
			: val(v), cnt(1), size(1), prio(random_device{}()), left(nullptr), right(nullptr) {}
		void usize() {
			size = cnt;
			if(left) {
				size += left->size;
			}
			if(right) {
				size += right->size;
			}
		}
	};
	Node *rt;
	static void _dtor(Node *ptr) {
		if(ptr->left) {
			_dtor(ptr->left);
			ptr->left = nullptr;
		}
		if(ptr->right) {
			_dtor(ptr->right);
			ptr->right = nullptr;
		}
		delete ptr;
	}
	static pair<Node *, Node *> _sval(Node *const ptr, int key) {
		if(!ptr) {
			return { nullptr, nullptr };
		}
		if(ptr->val <= key) {
			auto tmp = _sval(ptr->right, key);
			ptr->right = tmp.first;
			ptr->usize();
			return { ptr, tmp.second };
		} else {
			auto tmp = _sval(ptr->left, key);
			ptr->left = tmp.second;
			ptr->usize();
			return { tmp.first, ptr };
		}
	}
	static tuple<Node *, Node *, Node *> _srnk(Node *const ptr, int rnk) {
```

```
                if(!ptr) return { nullptr, nullptr, nullptr };
                int lsize = (ptr->left ? ptr->left->size : 0);
                if(rnk <= lsize) {
                        auto [lptr, mptr, rptr] = _srnk(ptr->left, rnk);
                        ptr->left = rptr;
                        ptr->usize();
                        return { lptr, mptr, ptr };
                } else if(rnk <= lsize + ptr->cnt) {
                        auto lptr = ptr->left, rptr = ptr->right;
                        ptr->left = ptr->right = nullptr;
                        return { lptr, ptr, rptr };
                } else {
                        auto [lptr, mptr, rptr] = _srnk(ptr->right, rnk - lsize - ptr->cnt);
                        ptr->right = lptr;
                        ptr->usize();
                        return { ptr, mptr, rptr };
                }
        }
        static Node *_merge(Node *const u, Node *const v) {
                if(!u) return v;
                if(!v) return u;
                if(u->prio < v->prio) {
                        u->right = _merge(u->right, v);
                        u->usize();
                        return u;
                } else {
                        v->left = _merge(u, v->left);
                        v->usize();
                        return v;
                }
        }
        int _qrv(Node *const ptr, int val) {
                auto [l, r] = _sval(ptr, val - 1);
                int ret = (l == nullptr ? 0 : l->size) + 1;
                rt = _merge(l, r);
                return ret;
        }
        int _qvr(Node *const ptr, int rnk) {
                auto [lptr, mptr, rptr] = _srnk(ptr, rnk);
                int ret = mptr->val;
                rt = _merge(_merge(lptr, mptr), rptr);
                return ret;
        }
};
```

树的倍增算法

```
constexpr int LOG = 20;
struct Increment {
        Increment(int n)
                : tree(n), anc(n, vector<int>(LOG, -1)), depth(n) {}
        void addedge(int u, int v) {
                tree[u].push_back(v);
```

```
                tree[v].push_back(u);
        }
        void build(int root) {
                dfs(root, -1);
        }
        int lca(int u, int v) {
                if(depth[u] < depth[v]) swap(u, v);
                for(int k = LOG - 1; k >= 0; --k) {
                        if(anc[u][k] != -1) {
                                if(depth[anc[u][k]] >= depth[v]) {
                                        u = anc[u][k];
                                }
                        }
                }
                if(u == v) return u;
                for(int k = LOG - 1; k >= 0; --k) {
                        if(anc[u][k] != anc[v][k]) {
                                u = anc[u][k];
                                v = anc[v][k];
                        }
                }
                return anc[u][0];
        }
        int kth_ancestor(int x, int k) {
                for(int i = 0; i < LOG; ++i) {
                        if((k >> i) & 1) {
                                x = anc[x][i];
                                if(x == -1) return -1;
                        }
                }
                return x;
        }
        vector<vector<int>> tree;
private:
        vector<vector<int>> anc;
        vector<int> depth;
        void dfs(int root, int fa) {
                depth[root] = fa != -1 ? depth[fa] + 1 : 0;
                anc[root][0] = fa;
                for(int k = 1; k < LOG; ++k) {
                        if(anc[root][k - 1] == -1) {
                                anc[root][k] = -1;
                        } else {
                                anc[root][k] = anc[anc[root][k - 1]][k - 1];
                        }
                }
                for(const int v : tree[root]) {
                        if(v != fa) {
                                dfs(v, root);
                        }
                }
        }
};
```

树链剖分

```
// 请拿一个单点修改（更新值）、区间查询的线段树
struct Node {
	ll w;
	vector<int> e;
};
struct HLD {
	HLD(const vector<Node> &g, int r = 0)
		: nodes(g.size()), nfd(g.size()) {
		dfs1(g, r, -1);
		int now_index = 0;
		dfs2(g, r, -1, now_index, r);
		vector<ll> nums(g.size());
		for(int i = 0; i < g.size(); ++i) {
			nums[nodes[i].dfn] = g[i].w;
		}
		seg.assign(nums);
	}
	int lca(int u, int v) const {
		while(nodes[u].toc != nodes[v].toc) {
			if(nodes[nodes[u].toc].d < nodes[nodes[v].toc].d) {
				v = nodes[nodes[v].toc].fa;
			} else {
				u = nodes[nodes[u].toc].fa;
			}
		}
		return nodes[u].d < nodes[v].d ? u : v;
	}
	void modify(int x, ll v) {
		seg.modify(nodes[x].dfn, v);
	}
	ll query_sum(int u, int v) const {
		int lca_node = lca(u, v);
		auto query_to_lca = [&](int point) -> ll {
			ll ans = 0;
			for(; nodes[point].toc != nodes[lca_node].toc;
				point = nodes[nodes[point].toc].fa) {
				int pos1 = nodes[point].dfn;
				int pos2 = nodes[nodes[point].toc].dfn;
				ans += seg.query_sum(pos2, pos1 + 1);
			}
			int pos1 = nodes[lca_node].dfn;
			int pos2 = nodes[point].dfn;
			ans += seg.query_sum(pos1 + 1, pos2 + 1);
			return ans;
		};
		ll ans = query_to_lca(u);
		ans += query_to_lca(v);
		int lca_pos = nodes[lca_node].dfn;
		ans += seg.query_sum(lca_pos, lca_pos + 1);
		return ans;
	}
private:
	struct _Node {
```

```
        int d;
        int fa;
        int toc;
        int dfn;
        int sz;
        int hs;
    };
    void dfs1(const vector<Node> &g, int x, int fa) {
        if(fa == -1) {
            nodes[x].d = 0;
        } else {
            nodes[x].d = nodes[fa].d + 1;
        }
        nodes[x].fa = fa;
        nodes[x].sz = 1;
        nodes[x].hs = -1;
        for(int i = 0; i < g[x].e.size(); ++i) {
            int next = g[x].e[i];
            if(next == fa) continue;
            dfs1(g, next, x);
            nodes[x].sz += nodes[next].sz;
            if(nodes[x].hs == -1 ||
                nodes[nodes[x].hs].sz < nodes[next].sz) {
                nodes[x].hs = next;
            }
        }
    }
    void dfs2(const vector<Node> &g, int x, int fa, int &dfn, int toc) {
        nodes[x].dfn = dfn++;
        nodes[x].toc = toc;
        if(nodes[x].hs != -1) {
            dfs2(g, nodes[x].hs, x, dfn, toc);
            for(int i = 0; i < g[x].e.size(); ++i) {
                int next = g[x].e[i];
                if(next != nodes[x].hs && next != fa) {
                    dfs2(g, next, x, dfn, next);
                }
            }
        }
    }
    vector<_Node> nodes;
    vector<int> nfd;
    SegTree seg;
};
```

CDQ分治

```
struct Data {
    int x, y, z;
    int cnt;
    int ans;
};
inline void kiana_1() noexcept {
```

```
int n, k;
cin >> n >> k;
Fenwick fwk(k);
vector<Data> _datas(n + 1, Data{ -1, -1, -1, 1 });
for(int i = 0; i < n; ++i) {
        cin >> _datas[i].x >> _datas[i].y >> _datas[i].z;
}
sort(_datas.begin(), _datas.end() - 1, [](const Data &l, const Data &r) {
        if(l.x != r.x) return l.x < r.x;
        if(l.y != r.y) return l.y < r.y;
        return l.z < r.z;
});
// 如果值域很大，考虑离散化数据
vector<Data> datas;
datas.reserve(n);
int cnt = 0;
for(int i = 0; i < n; ++i) {
        ++cnt;
        if((_datas[i].x != _datas[i + 1].x) ||
                (_datas[i].y != _datas[i + 1].y) ||
                (_datas[i].z != _datas[i + 1].z)) {
                datas.emplace_back(Data{ _datas[i].x, _datas[i].y, _datas[i].z, cnt });
                cnt = 0;
        }
}
int m = datas.size();
auto cdq = [&](auto &&cdq, int l, int r) -> void {
        if(r - l < 2) {
                return;
        }
        int mid = (l + r) >> 1;
        cdq(cdq, l, mid);
        cdq(cdq, mid, r);
        sort(datas.begin() + l, datas.begin() + mid, [](const Data &l, const Data &r) {
                if(l.y != r.y) return l.y < r.y;
                return l.z < r.z;
        });
        sort(datas.begin() + mid, datas.begin() + r, [](const Data &l, const Data &r) {
                if(l.y != r.y) return l.y < r.y;
                return l.z < r.z;
        });
        int j = l;
        for(int i = mid; i < r; ++i) {
                while(datas[j].y <= datas[i].y && j < mid) {
                        fwk.update(datas[j].z, datas[j].cnt);
                        ++j;
                }
                datas[i].ans += fwk.query(datas[i].z);
        }
        for(int k = l; k < j; ++k) {
                fwk.update(datas[k].z, -datas[k].cnt);
        }
};
cdq(cdq, 0, m);
vector<int> ans(n, 0);
for(int i = 0; i < m; ++i) {
```

```
                ans[datas[i].ans + datas[i].cnt - 1] += datas[i].cnt;
        }
        for(int i : ans) {
                cout << i << '\n';
        }
}
```

莫队

```
struct Query {
        int idx;
        // 左闭右开
        int l, r;
        int ans;
};
// 查询区间内满足元素出现次数等于自身的数的个数
void mo_algo(vector<Query> &queries, const vector<int> &nums) {
        int n = nums.size();
        int len = sqrt(n);
        sort(queries.begin(), queries.end(), [&](const Query &ql, const Query &qr) {
                int atl = ql.l / len, atr = qr.l / len;
                if(atl != atr) {
                        return atl < atr;
                }
                return (bool)((atl % 2 == 0) ^ (ql.r < qr.r));
        });
        int l = 0, r = 0, ans = 0;
        vector<int> count(n + 5, 0);
        auto add = [&](int x) {
                if(x >= n + 5) return;
                if(count[x] == x) --ans;
                count[x]++;
                if(count[x] == x) ++ans;
        };
        auto remove = [&](int x) {
                if(x >= n + 5) return;
                if(count[x] == x) --ans;
                count[x]--;
                if(count[x] == x) ++ans;
        };
        for(auto &q : queries) {
                // 抄板子时注意顺序，先加再更新还是先更新再加
                while(l > q.l) {
                        --l;
                        add(nums[l]);
                }
                while(l < q.l) {
                        remove(nums[l]);
                        ++l;
                }
                while(r < q.r) {
                        add(nums[r]);
                        ++r;
```

```
                }
                while(r > q.r) {
                        --r;
                        remove(nums[r]);
                }
                q.ans = ans;
        }
        sort(queries.begin(), queries.end(), [](const Query &l, const Query &r) {
                return l.idx < r.idx;
        });
}
```

单调栈、单调队列

```
// 单调栈：[l, r)，且有相同元素时，左边取最左端，右边取下一个自己
vector<pair<int, int>> max_range(const vector<int> &nums) {
        int n = nums.size();
        vector ret(n, pair(-1, -1));
        vector<pair<int, int>> stk; // 单调栈
        ret[0].first = 0;
        stk.emplace_back(nums[0], 0);
        for(int i = 1; i < n; ++i) {
                while(!stk.empty() && stk.back().first <= nums[i]) {
                        ret[stk.back().second].second = i;
                        stk.pop_back();
                }
                ret[i].first = stk.empty() ? 0 : stk.back().second + 1;
                stk.emplace_back(nums[i], i);
        }
        for(int i = n - 1; i >= 0; --i) {
                if(ret[i].second == -1) {
                        ret[i].second = n;
                }
        }
        return ret;
}
// 可以有相同元素
        vector ret(n, pair(0, n));
        vector<int> stk;
        for(int i = 0; i < n; ++i) {
                while(!stk.empty() && nums[stk.back()] <= nums[i]) {
                        stk.pop_back();
                }
                if(!stk.empty()) {
                        ret[i].first = stk.back() + 1;
                }
                stk.push_back(i);
        }
        stk.clear();
        for(int i = n - 1; i >= 0; --i) {
                while(!stk.empty() && nums[stk.back()] <= nums[i]) {
                        stk.pop_back();
                }
```

```
                if(!stk.empty()) {
                        ret[i].second = stk.back();
                }
                stk.push_back(i);
        }
// 最大滑动窗口：单调队列
vector<int> max_sliding_window(const vector<int> &nums, int k) {
        int n = nums.size();
        vector<int> ret(n - k + 1);
        vector<pair<int, int>> que;
        for(int i = 0; i < k - 1; ++i) {
                while(!que.empty() && que.back().first < nums[i]) {
                        que.pop_back();
                }
                que.emplace_back(nums[i], i);
        }
        int st = 0;
        for(int i = 0; i <= n - k; ++i) {
                int j = i + k - 1;
                if(que.size() > st && que[st].second < i) {
                        ++st;
                }
                while(!que.empty() && que.back().first < nums[j]) {
                        que.pop_back();
                }
                st = min(st, (int)que.size());
                que.emplace_back(nums[j], j);
                ret[i] = que[st].first;
        }
        return ret;
}
```

图论

Dijkstra

```
constexpr ll inf = 0x3f3f3f3f3f3f3f3f;
vector<ll> dijkstra(vector<vector<pair<int, ll>>> &graph, int start) noexcept {
        int v = graph.size();
        vector<ll> dist(v, inf);
        dist[start] = 0;
        vector<bool> visited(v, false);
        priority_queue<pair<ll, int>, vector<pair<ll, int>>, greater<>> pq;
        for(auto [vtx, w] : graph[start]) {
                dist[vtx] = w;
                pq.emplace(w, vtx);
        }
        while(!pq.empty()) {
                auto [w, vtx] = pq.top();
                pq.pop();
                if(visited[vtx]) continue;
```

```
                visited[vtx] = true;
                for(auto [vt, ww] : graph[vtx]) {
                        if(!visited[vt]) {
                                if(dist[vt] > dist[vtx] + ww) {
                                        dist[vt] = dist[vtx] + ww;
                                        pq.emplace(dist[vt], vt);
                                }
                        }
                }
        }
        return dist;
}
```

Floyd

```
// 操作后，graph就变成了最短路
void floyd(vector<vector<ll>> &graph) {
        for(int k = 0; k < graph.size(); ++k) {
                for(int i = 0; i < graph.size(); ++i) {
                        for(int j = 0; j < graph.size(); ++j) {
                                if(graph[i][j] > graph[i][k] + graph[k][j]) {
                                        graph[i][j] = graph[i][k] + graph[k][j];
                                }
                        }
                }
        }
}
```

SPFA

```
constexpr ll inf = 0x3f3f3f3f3f3f3f3f;
// 返回空vector说明有负环
vector<ll> spfa(const vector<vector<pair<int, ll>>> &graph, int s) {
        int n = graph.size();
        vector<ll> dist(n, inf);
        vector<int> count(n, 0);
        vector<bool> inqueue(n, false);
        queue<int> q;
        q.push(s);
        dist[s] = 0;
        inqueue[s] = true;
        while(!q.empty()) {
                int u = q.front();
                q.pop();
                inqueue[u] = false;
                for(auto [v, w] : graph[u]) {
                        if(dist[v] > dist[u] + w) {
                                dist[v] = dist[u] + w;
                                if(!inqueue[v]) {
                                        inqueue[v] = true;
                                        q.push(v);
```

```
                                        count[v] = count[u] + 1;
                                        if(count[v] > n) {
                                                return {};
                                        }
                                }
                        }
                }
        }
        return dist;
}
```

或值最短路

```
// 并查集谢谢喵
int ans = (1 << 30) - 1;
for(int i = 29; i >= 0; i--) {
        ans ^= (1 << i);
        DSU dsu(n);
        for(auto &[u, v, w] : edges) {
                if((w & ans) == w) {
                        dsu.connect(u, v);
                }
        }
        if(!dsu.is_connected(0, n - 1)) {
                ans ^= (1 << i);
        }
}
cout << ans << '\n';
```

Tarjan全家桶

```
// 强连通分量
struct Tarjan_SCC {
        Tarjan_SCC(int n) : nodes(n), graph(n) {}
        void addedge(int u, int v) {
                if(u == v) return;
                graph[u].push_back(v);
        }
        void solve() {
                int now_dfn = 0, scc_count = 0;
                for(int i = 0; i < nodes.size(); i++) {
                        if(nodes[i].dfn == -1) {
                                dfs(now_dfn, scc_count, i);
                        }
                }
                dag.assign(scc_count, {});
                set<pair<int, int>> edges;
                for(int u = 0; u < graph.size(); ++u) {
                        for(int v : graph[u]) {
                                int bu = nodes[u].inscc, bv = nodes[v].inscc;
                                if(bu != bv && !edges.contains({ bu, bv })) {
```

```
                                dag[bu].emplace_back(bv);
                                edges.insert({ bu, bv });
                        }
                }
        }
    }
    struct Node {
        int dfn;
        int low;
        bool instack;
        int inscc;
        Node() : dfn(-1), low(-1), instack(false), inscc(-1) {}
    };
    vector<Node> nodes;
    vector<vector<int>> graph;
    vector<vector<int>> sccs;
    vector<vector<int>> dag;
private:
    stack<int> scc_stack;
    void dfs(int &now_dfn, int &scc_count, int u) {
        nodes[u].dfn = now_dfn;
        nodes[u].low = now_dfn;
        ++now_dfn;
        scc_stack.push(u);
        nodes[u].instack = true;
        for(int v : graph[u]) {
            if(nodes[v].dfn == -1) {
                dfs(now_dfn, scc_count, v);
                nodes[u].low = min(nodes[u].low, nodes[v].low);
            } else if(nodes[v].instack) {
                nodes[u].low = min(nodes[u].low, nodes[v].low);
            }
        }
        if(nodes[u].dfn == nodes[u].low) {
            vector<int> scc;
            int v = -1;
            while(v != u) {
                v = scc_stack.top();
                scc_stack.pop();
                nodes[v].instack = false;
                nodes[v].inscc = scc_count;
                scc.emplace_back(v);
            }
            sccs.emplace_back(move(scc));
            ++scc_count;
        }
    }
};
// 边双
struct Tarjan_BCC {
    Tarjan_BCC(int n)
        : nodes(n), graph(n) {}
    void addedge(int u, int v) {
        if(u == v) return;
        graph[u].emplace_back(v);
        graph[v].emplace_back(u);
```

```
        }
        void solve() {
                int dfn_now = 0;
                for(int i = 0; i < nodes.size(); ++i) {
                        if(nodes[i].dfn == -1) {
                                dfs(i, -1, dfn_now);
                        }
                }
        }
        struct Node {
                int dfn;
                int low;
                bool instack;
                Node() :dfn(-1), low(-1), instack(false) {}
        };
        vector<Node> nodes;
        vector<vector<int>> graph;
        vector<vector<int>> dcc;
private:
        stack<int> stk;
        void dfs(int u, int father, int &dfn_now) {
                nodes[u].dfn = dfn_now;
                nodes[u].low = dfn_now;
                nodes[u].instack = true;
                dfn_now++;
                stk.push(u);
                for(int v : graph[u]) {
                        if(v == father) continue;
                        if(nodes[v].dfn == -1) {
                                dfs(v, u, dfn_now);
                                nodes[u].low = min(nodes[u].low, nodes[v].low);
                        } else if(nodes[v].instack) {
                                nodes[u].low = min(nodes[u].low, nodes[v].dfn);
                        }
                }
                if(nodes[u].dfn == nodes[u].low) {
                        vector<int> t;
                        int n = -1;
                        while(n != u) {
                                n = stk.top();
                                stk.pop();
                                t.emplace_back(n);
                                nodes[n].instack = false;
                        }
                        dcc.emplace_back(move(t));
                }
        }
};
// 点双
struct Tarjan_DCC {
        Tarjan_DCC(int n)
                : nodes(n), graph(n) {}
        void addedge(int u, int v) {
                if(u == v) return;
                graph[u].emplace_back(v);
                graph[v].emplace_back(u);
```

```
        }
        void solve() {
                int dfn_now = 0;
                for(int i = 0; i < nodes.size(); ++i) {
                        if(nodes[i].dfn == -1) {
                                dfs(i, -1, i, dfn_now);
                        }
                }
        }
        struct Node {
                int dfn;
                int low;
                Node() :dfn(-1), low(-1) {}
        };
        vector<Node> nodes;
        vector<vector<int>> graph;
        vector<vector<int>> dcc;
private:
        stack<int> stk;
        void dfs(int u, int parent, int root, int &dfn_now) {
                nodes[u].dfn = dfn_now;
                nodes[u].low = dfn_now;
                dfn_now++;
                stk.push(u);
                int child = 0;
                for(int v : graph[u]) {
                        if(v == parent) continue;
                        if(nodes[v].dfn == -1) {
                                dfs(v, u, root, dfn_now);
                                nodes[u].low = min(nodes[u].low, nodes[v].low);
                                if(nodes[v].low >= nodes[u].dfn) {
                                        ++child;
                                        vector<int> f = { u };
                                        while(!stk.empty()) {
                                                int x = stk.top();
                                                stk.pop();
                                                f.emplace_back(x);
                                                if(x == v) break;
                                        }
                                        dcc.emplace_back(move(f));
                                }
                        } else {
                                nodes[u].low = min(nodes[u].low, nodes[v].dfn);
                        }
                }
                if(u == root && graph[u].empty()) {
                        dcc.emplace_back(vector<int>{u});
                        return;
                }
        }
};
// 割点
struct Tarjan_Cutpoint {
        Tarjan_Cutpoint(int n)
                : nodes(n), graph(n) {}
        void addedge(int u, int v) {
```

```
                graph[u].emplace_back(v);
                graph[v].emplace_back(u);
        }
        void solve() {
                int dfn_now = 0;
                for(int i = 0; i < nodes.size(); ++i) {
                        if(nodes[i].dfn == -1) {
                                dfs(i, -1, dfn_now);
                        }
                }
                sort(cutpoints.begin(), cutpoints.end());
                cutpoints.erase(unique(cutpoints.begin(), cutpoints.end()), cutpoints.end());
        }
        struct Node {
                int dfn;
                int low;
                Node() : dfn(-1), low(-1) {}
        };
        vector<Node> nodes;
        vector<vector<int>> graph;
        vector<int> cutpoints;
private:
        void dfs(int u, int father, int &dfn_now) {
                nodes[u].dfn = dfn_now;
                nodes[u].low = dfn_now;
                dfn_now++;
                int child = 0;
                bool flag = false;
                for(int v : graph[u]) {
                        if(nodes[v].dfn == -1) {
                                child++;
                                dfs(v, u, dfn_now);
                                nodes[u].low = min(nodes[u].low, nodes[v].low);
                                if(father != -1) {
                                        flag |= (nodes[v].low >= nodes[u].dfn);
                                }
                        } else if(v != father) {
                                nodes[u].low = min(nodes[u].low, nodes[v].dfn);
                        }
                }
                if(father == -1) {
                        flag = (child > 1);
                }
                if(flag) {
                        cutpoints.emplace_back(u);
                }
        }
};
// 桥
struct Tarjan_Bridge {
        Tarjan_Bridge(int n)
                : nodes(n), graph(n) {}
        void addedge(int u, int v) {
                graph[u].emplace_back(v);
                graph[v].emplace_back(u);
        }
```

```
        void solve() {
                int dfn_now = 0;
                for(int i = 0; i < nodes.size(); ++i) {
                        if(nodes[i].dfn == -1) {
                                dfs(i, -1, dfn_now);
                        }
                }
        }
        struct Node {
                int dfn;
                int low;
                Node() : dfn(-1), low(-1) {}
        };
        vector<Node> nodes;
        vector<vector<int>> graph;
        vector<pair<int, int>> bridges;
private:
        void dfs(int u, int father, int &dfn_now) {
                nodes[u].dfn = nodes[u].low = dfn_now++;
                for(int v : graph[u]) {
                        if(nodes[v].dfn == -1) {
                                dfs(v, u, dfn_now);
                                nodes[u].low = min(nodes[u].low, nodes[v].low);
                                if(nodes[v].low > nodes[u].dfn) {
                                        bridges.emplace_back(min(u, v), max(u, v));
                                }
                        } else if(v != father) {
                                nodes[u].low = min(nodes[u].low, nodes[v].dfn);
                        }
                }
        }
};
```

同余最短路

```
// 跑的dijkstra
inline void kiana_1() noexcept {
        ll k;
        vector<ll> dist(4);
        cin >> k;
        for(int i = 0; i < 4; ++i) {
                cin >> dist[i];
        }
        ll modulo = 2 * min(dist[0], dist[3]);
        vector<vector<ll>> dp(4, vector<ll>(modulo, inf));
        vector<vector<bool>> visited(4, vector<bool>(modulo, false));
        dp[0][0] = 0;
        priority_queue<tuple<ll, ll, ll>, vector<tuple<ll, ll, ll>>, greater<>> pq;
        pq.emplace(0, 0, 0);
        auto get = [](ll a, ll b) -> ll {
                if(a == 3 && b == 0) {
                        return 3;
                }
```

```
                if(b == 3 && a == 0) {
                        return 3;
                }
                return min(a, b);
        };
        while(!pq.empty()) {
                auto [w, m, c] = pq.top();
                pq.pop();
                if(visited[c][m]) continue;
                visited[c][m] = true;
                ll d1 = (c + 1) % 4, d2 = (c + 3) % 4;
                ll w1 = dist[get(c, d1)], w2 = dist[get(c, d2)];
                if(chkmin(dp[d1][(w + w1) % modulo], w + w1)) {
                        pq.emplace(w + w1, (w + w1) % modulo, d1);
                }
                if(chkmin(dp[d2][(w + w2) % modulo], w + w2)) {
                        pq.emplace(w + w2, (w + w2) % modulo, d2);
                }
        }
        ll ans = inf;
        for(ll i = 0; i < modulo; ++i) {
                ll required = (k / modulo) * modulo + i;
                while(required < k) {
                        required += modulo;
                }
                chkmin(ans, max(required, dp[0][i]));
        }
        cout << ans << '\n';
}
```

Kruskal

```
constexpr ll inf = 0x3f3f3f3f3f3f3f3f;
struct Edge {
        int u, v;
        ll w;
};
// 拿个并查集板子下来谢谢喵
ll kruskal(vector<Edge> &edges, int n) {
        DSU dsu(n);
        ll ans = 0;
        sort(edges.begin(), edges.end(), [](const Edge &a, const Edge &b) {
                return a.w < b.w;
        });
        for(auto &e : edges) {
                if(!dsu.is_connected(e.u, e.v)) {
                        dsu.connect(e.u, e.v);
                        ans += e.w;
                }
        }
        for(int i = 1; i < n; ++i) {
                if(!dsu.is_connected(i, 0)) {
                        return inf;
```

```
                }
        }
        return ans;
}
```

Prim

```
constexpr ll inf = 0x3f3f3f3f3f3f3f3f;
ll prim(const vector<vector<pair<int, ll>>> &graph) {
        int n = graph.size();
        vector<bool> visited(n, false);
        vector<ll> dist(n, inf);
        dist[0] = 0;
        priority_queue<pair<ll, int>, vector<pair<ll, int>>, greater<>> pq;
        pq.emplace(0, 0);
        ll ret = 0;
        while(!pq.empty()) {
                auto [w, v] = pq.top();
                pq.pop();
                if(visited[v]) continue;
                visited[v] = true;
                ret += w;
                for(auto [u, w] : graph[v]) {
                        if(!visited[u] && dist[u] > w) {
                                dist[u] = w;
                                pq.emplace(w, u);
                        }
                }
        }
        for(bool b : visited) {
                if(!b) {
                        return inf;
                }
        }
        return ret;
}
```

Boruvka

```
constexpr ll inf = 0x3f3f3f3f3f3f3f3f;
struct Edge {
        int u, v;
        ll w;
};
ll boruvka(const vector<Edge> &edges, int n) {
        DSU dsu(n);
        ll ans = 0;
        int c = n;
        vector<Edge> mst(n);
        while(c > 1) {
                for(int i = 0; i < n; ++i) {
```

```
                        if(dsu.find(i) != i) continue;
                        mst[i] = Edge(-1, -1, inf);
                }
                for(auto [u, v, w] : edges) {
                        int fu = dsu.find(u), fv = dsu.find(v);
                        if(fu == fv) continue;
                        if(mst[fu].w > w) {
                                mst[fu] = Edge(u, v, w);
                        }
                        if(mst[fv].w > w) {
                                mst[fv] = Edge(u, v, w);
                        }
                }
                bool flag = false;
                for(int i = 0; i < n; ++i) {
                        if(dsu.find(i) != i) continue;
                        auto [u, v, w] = mst[i];
                        if(w == inf) return inf;
                        if(!dsu.is_connected(u, v)) {
                                dsu.connect(u, v);
                                ans += w;
                                --c;
                                flag = true;
                        }
                }
                if(!flag) {
                        return inf;
                }
        }
        return ans;
}
```

Dinic

```
constexpr ll inf = 0x3f3f3f3f3f3f3f3f;
struct Dinic {
        Dinic(int n, int s, int e)
                : graph(n), level(n), start(s), end(e), n(n) {}
        void addedge(int u, int v, ll w) {
                graph[u].emplace_back(Edge{ v, (int)graph[v].size(), w });
                graph[v].emplace_back(Edge{ u, (int)graph[u].size() - 1, 0 });
        }
        ll solve() {
                ll ans = 0;
                while(bfs()) {
                        ll flow = dfs(start, inf);
                        while(flow > 0) {
                                ans += flow;
                                flow = dfs(start, inf);
                        }
                }
                return ans;
        }
private:
```

```
        struct Edge {
                int to;
                int rev;
                ll cap;
        };
        vector<vector<Edge>> graph;
        vector<int> level;
        int n;
        int start, end;
        bool bfs() {
                fill(level.begin(), level.end(), -1);
                level[start] = 0;
                queue<int> q;
                q.emplace(start);
                while(!q.empty()) {
                        int u = q.front();
                        q.pop();
                        for(const auto &e : graph[u]) {
                                if(level[e.to] == -1 && e.cap > 0) {
                                        level[e.to] = level[u] + 1;
                                        q.emplace(e.to);
                                }
                        }
                }
                return level[end] != -1;
        }
        ll dfs(int u, ll max_flow) {
                if(u == end || max_flow == 0) return max_flow;
                ll total_flow = 0;
                for(auto &e : graph[u]) {
                        if(level[e.to] == level[u] + 1 && e.cap > 0) {
                                ll min_flow = min(max_flow, e.cap);
                                ll pushed = dfs(e.to, min_flow);
                                if(pushed > 0) {
                                        e.cap -= pushed;
                                        graph[e.to][e.rev].cap += pushed;
                                        total_flow += pushed;
                                        max_flow -= pushed;
                                        if(max_flow == 0) {
                                                return total_flow;
                                        }
                                }
                        }
                }
                return total_flow;
        }
};
```

匈牙利算法（不带权）

```
int hungarian(const vector<vector<int>> &graph, int vsz) {
        int usz = graph.size();
        vector<int> mu(usz, -1);
```

```
        vector<int> mv(vsz, -1);
        auto dfs = [&](auto &&dfs, int u, vector<bool> &visited) -> bool {
                for(int v : graph[u]) {
                        if(visited[v]) continue;
                        visited[v] = true;
                        if(mv[v] == -1 || dfs(dfs, mv[v], visited)) {
                                mv[v] = u;
                                mu[u] = v;
                                return true;
                        }
                }
                return false;
        };
        int ret = 0;
        for(int u = 0; u < usz; ++u) {
                if(mu[u] == -1) {
                        vector<bool> visited(vsz, false);
                        if(dfs(dfs, u, visited)) {
                                ret++;
                        }
                }
        }
        return ret;
}
```

2-SAT

```
// 本题给的是析取式，在这里我转化为了蕴含式求解
// 请打一个Tarjan_SCC下来
// 注意：蕴含式也要加另一条边，A->B要加B'->A'
int main() {
        cin.tie(nullptr)->sync_with_stdio(false);
        int n, m;
        cin >> n >> m;
        Tarjan_SCC solver(2 * n);
        for(int _ = 0; _ < m; ++_) {
                int i, flag_i, j, flag_j;
                cin >> i >> flag_i >> j >> flag_j;
                --i;
                --j;
                solver.addedge(2 * i + 1 - flag_i, 2 * j + flag_j);
                solver.addedge(2 * j + 1 - flag_j, 2 * i + flag_i);
        }
        solver.solve();
        for(int i = 0; i < n; ++i) {
                if(solver.nodes[2 * i].inscc == solver.nodes[2 * i + 1].inscc) {
                        cout << "IMPOSSIBLE\n";
                        return 0;
                }
        }
        cout << "POSSIBLE\n";
        vector<int> ans(n, -1);
        for(int i = 0; i < solver.sccs.size(); ++i) {
```

```
                for(int ii : solver.sccs[i]) {
                        int u = ii / 2;
                        int v = ii % 2;
                        if(ans[u] == -1) {
                                ans[u] = v;
                        }
                }
        }
        for(int i : ans) {
                cout << i << ' ';
        }
        return 0;
}
```

搜索

迭代加深

```
pair<int, stack<int>> iddfs(array<array<int, 3>, 3> src, const array<array<int, 3>, 3> &dst) {
        constexpr int dx[] = { -1, 1, 0, 0 };
        constexpr int dy[] = { 0, 0, -1, 1 };
        int ans = 0;
        stack<int> stk;
        pair<int, int> zeropos;
        for(int i = 0; i < 3; ++i) {
                for(int j = 0; j < 3; ++j) {
                        if(src[i][j] == 0) {
                                zeropos = { i, j };
                        }
                }
        }
        auto dfs = [&](auto &&dfs, int depth, int lastdir) -> bool {
                if(src == dst) {
                        for(int i = 0; i < 3; ++i) {
                                for(int j = 0; j < 3; ++j) {
                                        stk.push(src[i][j]);
                                }
                        }
                        return true;
                }
                if(depth >= ans) return false;
                auto revdir = [](int cur) -> int {
                        if(cur < 2) return 1 - cur;
                        return 5 - cur;
                };
                for(int nowdir = 0; nowdir < 4; ++nowdir) {
                        if(nowdir == revdir(lastdir)) {
                                continue;
                        }
                        auto [oldx, oldy] = zeropos;
                        int newx = oldx + dx[nowdir], newy = oldy + dy[nowdir];
```

```
                        if(newx < 0 || newx > 2 || newy < 0 || newy > 2) continue;
                        swap(src[oldx][oldy], src[newx][newy]);
                        zeropos = { newx, newy };
                        if(dfs(dfs, depth + 1, nowdir)) {
                                swap(src[oldx][oldy], src[newx][newy]);
                                zeropos = { oldx, oldy };
                                for(int i = 0; i < 3; ++i) {
                                        for(int j = 0; j < 3; ++j) {
                                                stk.push(src[i][j]);
                                        }
                                }
                                return true;
                        }
                        swap(src[oldx][oldy], src[newx][newy]);
                        zeropos = { oldx, oldy };
                }
                return false;
        };
        for(ans = 0; ans < 33; ++ans) {
                if(dfs(dfs, 0, 114514)) {
                        return { ans, stk };
                }
        }
        return { -1, stack<int>{} };
}
```

动态规划

背包DP

```
namespace OneD {
// 01背包
ll knapsack_01(const vector<pair<int, ll>> &objects, int maxw) {
        int n = objects.size();
        vector<ll> dp(maxw + 1, 0);
        for(auto [w, v] : objects) {
                for(int j = maxw; j >= w; --j) {
                        dp[j] = max(dp[j], dp[j - w] + v);
                }
        }
        return dp[maxw];
}
// 完全背包
ll knapsack_complete(const vector<pair<int, ll>> &objects, int maxw) {
        int n = objects.size();
        vector<ll> dp(maxw + 1, 0);
        for(auto [w, v] : objects) {
                for(int j = w; j <= maxw; ++j) {
                        dp[j] = max(dp[j], dp[j - w] + v);
                }
        }
```

```
        return dp[maxw];
}
struct Object {
        int cost, cnt;
        ll w;
};
// 多重背包
ll multi_knapsack(const vector<Object> &objects, int maxw) {
        vector<ll> dp(maxw + 1, 0);
        for(auto [cost, cnt, w] : objects) {
                for(int _ = 0; _ < cnt; ++_) {
                        for(int j = maxw; j >= cost; --j) {
                                dp[j] = max(dp[j], dp[j - cost] + w);
                        }
                }
        }
        return dp[maxw];
}
// 多重背包的二进制优化
ll multi_knapsack_binary(const vector<Object> &objects, int maxw) {
        vector<pair<int, ll>> objs;
        for(auto [cost, cnt, w] : objects) {
                int k = 1;
                while(cnt > 0) {
                        int take = min(k, cnt);
                        objs.emplace_back(cost * take, w * take);
                        cnt -= take;
                        k *= 2;
                }
        }
        return knapsack_01(objs, maxw);
}
// 缺少：多重背包的单调队列优化
}

namespace TwoD {
struct Object {
        int u, v;
        ll w;
};
// 二维01背包
ll knapsack_01(int U, int V, const vector<Object> &objects) {
        vector<vector<ll>> dp(U + 1, vector<ll>(V + 1, 0));
        for(auto [u, v, w] : objects) {
                for(int i = U; i - u >= 0; --i) {
                        for(int j = V; j - v >= 0; --j) {
                                dp[i][j] = max(dp[i][j], dp[i - u][j - v] + w);
                        }
                }
        }
        return dp[U][V];
}

}
```

区间DP

```
int main() {
	cin.tie(nullptr)->sync_with_stdio(false);
	int n;
	cin >> n;
	vector<ll> nums(n);
	for(int i = 0; i < n; ++i) {
		cin >> nums[i];
	}
	vector<int> s(2 * n + 1);
	for(int i = 0; i < 2 * n; ++i) {
		s[i + 1] = s[i] + nums[i];
	}
	vector<vector<int>> dp(2 * n + 1, vector<int>(2 * n + 1));
	for(int len = 2; len <= n; ++len) {
		for(int i = 0; i <= 2 * n - len; ++i) {
			int j = i + len;
			int mx = 0;
			for(int k = i + 1; k < j; ++k) {
				mx = max(mx, dp[i][k] + dp[k][j]);
			}
			dp[i][j] = mx + s[j] - s[i];
		}
	}
	int ans = 0;
	for(int i = 0; i < n; ++i) {
		ans = max(ans, dp[i][i + n]);
	}
	cout << ans << '\n';
	return 0;
}
```

数位DP

```
constexpr ll modulo = 10'0000'0007;
ll digit_dp(string k, int d) {
	vector<vector<ll>> memo(k.size(), vector<ll>(d, -1));
	auto dfs = [&](auto &&dfs, int idx, ll psum, bool isnum, bool islimit) -> ll {
		if(idx == k.size()) {
			return (ll)(isnum && (psum == 0));
		}
		if(memo[idx][psum] != -1 && isnum && !islimit) {
			return memo[idx][psum];
		}
		ll ans = 0;
		if(!isnum) {
			ans = (ans + dfs(dfs, idx + 1, 0, false, false)) % modulo;
		}
		int llim = isnum ? 0 : 1, hlim = islimit ? (k[idx] - '0') : 9;
		for(int i = llim; i <= hlim; ++i) {
			ans = (ans + dfs(dfs, idx + 1, ((psum + i) % d), true, (islimit && (i == hlim)
```

```
                }
                if(isnum && !islimit) {
                        memo[idx][psum] = ans;
                }
                return ans;
        };
        return dfs(dfs, 0, 0, false, true);
}
```

计数DP

```
constexpr ll modulo = 10'0000'0007;
ll count_dp(string s) {
        int n = s.size();
        vector<vector<ll>> dp(n, vector<ll>(n, 0));
        dp[0][0] = 1;
        vector<vector<ll>> sum(n, vector<ll>(n + 1, 0));
        sum[0][1] = 1;
        for(int i = 1; i < n; ++i) {
                for(int j = 0; j <= i; ++j) {
                        if(s[i - 1] == '<') {
                                dp[i][j] = sum[i - 1][j];
                        } else {
                                dp[i][j] = (sum[i - 1][i] - sum[i - 1][j] + modulo) % modulo;
                        }
                }
                sum[i][0] = 0;
                for(int j = 1; j <= i + 1; ++j) {
                        sum[i][j] = (sum[i][j - 1] + dp[i][j - 1]) % modulo;
                }
        }
        ll ans = 0;
        for(int j = 0; j < n; ++j) {
                ans = (ans + dp[n - 1][j]) % modulo;
        }
}
```

线段树优化DP

```
// SegTree区间最大值、单点加法线段树
int lis(const vector<int> &nums) {
        auto discrete = nums;
        sort(discrete.begin(), discrete.end());
        discrete.erase(unique(discrete.begin(), discrete.end()), discrete.end());
        unordered_map<int, int> mp;
        for(int i = 0; i < discrete.size(); i++) {
                mp[discrete[i]] = i;
        }
        SegTree seg(discrete.size());
        for(int i = 0; i < nums.size(); i++) {
                int x = mp[nums[i]];
```

```
                int v = seg.query(0, x) + 1;
                seg.update(x, v);
        }
        return seg.query(0, discrete.size());
}
```

数学

快速幂和模整数类

```
constexpr ll modulo = 9'9824'4353;
ll qpow(ll x, ll n) {
        ll ret = 1;
        while(n) {
                if(n & 1) ret = ret * x % modulo;
                x = x * x % modulo;
                n >>= 1;
        }
        return ret;
}
struct ModInt {
        ModInt(ll v = 0) : val(v % modulo) {}
        ModInt operator+(const ModInt &rhs) const {
                return ModInt((val + rhs.val) % modulo);
        }
        ModInt operator-(const ModInt &rhs) const {
                return ModInt((val - rhs.val + modulo) % modulo);
        }
        ModInt operator*(const ModInt &rhs) const {
                return ModInt((val * rhs.val) % modulo);
        }
        ModInt operator/(const ModInt &rhs) const {
                return ModInt(val * qpow(rhs.val, modulo - 2) % modulo);
        }
        ModInt power(int N) const {
                ModInt ret = 1;
                ModInt base = val;
                while(N) {
                        if(N & 1) ret = ret * base;
                        base = base * base;
                        N >>= 1;
                }
                return ret;
        }
        operator ll &() {
                return val;
        }
private:
        ll val;
};
// 底数相同时的快速幂，预处理O(sqrtn)，查询O(1)
```

```
constexpr ll maxn = 50005, modulo = 9'9824'4353;
ll power2[maxn], power2m[maxn];
int init = [] {
        power2[0] = power2m[0] = 1;
        for(int i = 1; i < maxn; ++i) {
                power2[i] = (power2[i - 1] * 2) % modulo;
        }
        power2m[1] = (power2[maxn - 1] * 2) % modulo;
        for(int i = 2; i < maxn; ++i) {
                power2m[i] = (power2m[i - 1] * power2m[1]) % modulo;
        }
        return 0;
}();
ll pow2(ll x) {
        ll i = x / maxn, j = x % maxn;
        return (power2m[i] * power2[j]) % modulo;
}
```

组合数

```
// 不预处理，一次性算一排
vector<ll> comb(ll n) {
        vector<ModInt> ret_(n + 1);
        ret_[0] = ret_[n] = 1;
        for(int i = 1, j = n - 1; i < j; ++i, --j) {
                ret_[i] = ret_[j] = ret_[i - 1] * ModInt(n - i + 1) / ModInt(i);
        }
        vector<ll> ret(ret_.begin(), ret_.end());
        return ret;
}
// 预处理版
constexpr ll maxn = 10'0005, modulo = 10'0000'0007;
ll qpow(ll x, ll n) {
        ll ret = 1;
        while(n) {
                if(n & 1) ret = ret * x % modulo;
                x = x * x % modulo;
                n >>= 1;
        }
        return ret;
}
ll fact[maxn], invfact[maxn];
int init = [] {
        fact[0] = 1;
        for(int i = 1; i < maxn; ++i) {
                fact[i] = (fact[i - 1] * i) % modulo;
        }
        invfact[maxn - 1] = qpow(fact[maxn - 1], modulo - 2);
        for(int i = maxn - 2; i >= 0; --i) {
                invfact[i] = (invfact[i + 1] * (i + 1)) % modulo;
        }
        return 0;
}();
```

```
ll comb(ll n, ll m) {
        if(m < 0 || m > n) return 0;
        return (((fact[n] * invfact[m]) % modulo) * invfact[n - m]) % modulo;
}
```

EXGCD

```
// @returns (gcd, x, y) so that gcd = ax + by
tuple<ll, ll, ll> exgcd(ll a, ll b) {
        if(b == 0) return tuple(a, 1ll, 0ll);
        auto [g, x, y] = exgcd(b, a % b);
        return tuple(g, y, x - (a / b) * y);
}
```

中国剩余定理

```
// @returns (a, b) so that answer is a + kb, k\in N_+
pair<ll, ll> crt(const vector<ll> &rem, const vector<ll> &mod) {
        int n = rem.size();
        ll modulo_ = 1, ans = 0;
        for(int i = 0; i < n; ++i) {
                modulo_ *= mod[i];
        }
        for(int i = 0; i < n; ++i) {
                ll m = modulo_ / mod[i];
                auto [_, b, __] = exgcd(m, mod[i]);
                ans = (ans + ((rem[i] * m) % modulo_ * b) % modulo_) % modulo_;
        }
        return pair((ans % modulo_ + modulo_) % modulo_, modulo_);
}
```

二项式反演公式

$$f(n)=\sum_{k=0}^{n}\binom{n}{k}g(k)\to g(n)=\sum_{k=0}^{n}(-1)^{n-k}\binom{n}{k}f(k)$$

$$f(n)=\sum_{k=n}^{m}\binom{k}{n}g(k)\to g(n)=\sum_{k=n}^{m}(-1)^{k-n}\binom{k}{n}f(k)$$

斯特林数

```
constexpr int maxn = 5005;
ll sterling[maxn][maxn];
int init = [] {
    sterling[0][0] = 1;
    for(int i = 1; i < maxn; ++i) {
```

```
        sterling[i][0] = 0;
        for(int j = 1; j <= i; ++j) {
            sterling[i][j] = (sterling[i - 1][j - 1] + sterling[i - 1][j] * j) % modulo;
        }
    }
    return 0;
}();
```

质数筛和它能干的事

```
constexpr ll maxn = 500005;
ll      phi[maxn], // 欧拉函数
        mu[maxn], // 莫比乌斯函数
        sigma0[maxn], // 因数个数
        sigma1[maxn], // 因数和
        lprime[maxn], // lowest prime, i==lprime[i]等价于i是质数
        lpower[maxn], // lprime[i]的次数
        powsum[maxn]; // \sum_{j=0}^{lpower[i]}(lprime[i])^j
vector<ll> primes;
ll qpow(ll x, ll n);
int init = [] {
        primes.reserve(maxn / 10);
        phi[1] = mu[1] = sigma0[1] = sigma1[1] = powsum[1] = 1;
        lpower[1] = 0;
        for(ll i = 2; i < maxn; ++i) {
                if(lprime[i] == 0) {
                        lprime[i] = i;
                        primes.push_back(i);
                        phi[i] = i - 1;
                        mu[i] = -1;
                        sigma0[i] = 2;
                        lpower[i] = 1;
                        powsum[i] = i + 1;
                        sigma1[i] = powsum[i];
                }
                for(ll p : primes) {
                        ll j = i * p;
                        if(j >= maxn) break;
                        lprime[j] = p;
                        if(i % p == 0) {
                                phi[j] = phi[i] * p;
                                mu[j] = 0;
                                lpower[j] = lpower[i] + 1;
                                powsum[j] = powsum[i] + qpow(p, lpower[j]);
                                sigma0[j] = sigma0[j / qpow(lprime[j], lpower[j])] * (lpower[j] + 1);
                                sigma1[j] = sigma1[j / qpow(lprime[j], lpower[j])] * powsum[j];
                                break;
                        } else {
                                phi[j] = phi[i] * (p - 1);
                                mu[j] = -mu[i];
                                lpower[j] = 1;
                                powsum[j] = 1 + p;
                                sigma1[j] = sigma1[i] * powsum[j];
```

```
                                sigma0[j] = sigma0[i] * 2;
                        }
                }
        }
        return 0;
}();
```

多项式乘法

FFT

```
using cd = complex<double>;
constexpr double pi = 3.14159265358979323846264338327950288;
vector<int> multiply(const vector<int> &a, const vector<int> &b) {
        int n = 1;
        while(n < (a.size() + b.size())) {
                n <<= 1;
        }
        vector<cd> fa(n), fb(n);
        for(int i = 0; i < a.size(); ++i) {
                fa[i] = a[i];
        }
        for(int i = 0; i < b.size(); ++i) {
                fb[i] = b[i];
        }
        auto fft = [](auto &&fft, vector<cd> &f, bool invert) -> void {
                int n = f.size();
                if(n == 1) return;
                vector<cd> f0(n / 2), f1(n / 2);
                for(int i = 0; i < n / 2; ++i) {
                        f0[i] = f[2 * i];
                        f1[i] = f[2 * i + 1];
                }
                fft(fft, f0, invert);
                fft(fft, f1, invert);
                double theta = 2.l * pi / n * (invert ? -1.l : 1.l);
                cd wt = 1, w(cos(theta), sin(theta));
                for(int t = 0; t < n / 2; ++t) {
                        cd u = f0[t], v = wt * f1[t];
                        f[t] = u + v;
                        f[t + n / 2] = u - v;
                        wt *= w;
                }
        };
        fft(fft, fa, false);
        fft(fft, fb, false);
        for(int i = 0; i < n; ++i) {
                fa[i] *= fb[i];
        }
        fft(fft, fa, true);
        for(int i = 0; i < n; ++i) {
                fa[i] /= n;
        }
```

```
        vector<int> ret(n);
        for(int i = 0; i < n; ++i) {
                ret[i] = int(fa[i].real() + (fa[i].real() > 0.1 ? 0.5l : -0.5l));
        }
        while(ret.size() > (a.size() + b.size() - 1)) {
                ret.pop_back();
        }
        return ret;
}
```

NTT

```
constexpr ll modulo = 9'9824'4353, g = 3;
// constexpr ll modulo = 10'0000'0007, g = 5; 不可以
ll qpow(ll x, ll n) {
        ll ret = 1;
        while(n) {
                if(n & 1) {
                        ret = ret * x % modulo;
                }
                x = x * x % modulo;
                n >>= 1;
        }
        return ret;
}
vector<ll> multiply(const vector<ll> &a, const vector<ll> &b) {
        int n = 1;
        while(n < a.size() + b.size()) {
                n <<= 1;
        }
        vector<ll> ca(n), cb(n);
        for(int i = 0; i < a.size(); ++i) {
                ca[i] = a[i];
        }
        for(int i = 0; i < b.size(); ++i) {
                cb[i] = b[i];
        }
        auto ntt = [](auto &&ntt, vector<ll> &f, bool invert) -> void {
                int n = f.size();
                if(n == 1) return;
                vector<ll> f0(n / 2), f1(n / 2);
                for(int i = 0; i < n / 2; ++i) {
                        f0[i] = f[2 * i];
                        f1[i] = f[2 * i + 1];
                }
                ntt(ntt, f0, invert);
                ntt(ntt, f1, invert);
                ll w = 1, wn = qpow(g, (modulo - 1) / n);
                if(invert) {
                        wn = qpow(wn, modulo - 2);
                }
                for(int t = 0; t < n / 2; ++t) {
                        ll u = f0[t], v = w * f1[t] % modulo;
                        f[t] = (u + v) % modulo;
```

```
                        f[t + n / 2] = (u - v + modulo) % modulo;
                        w = w * wn % modulo;
                }
        };
        ntt(ntt, ca, false);
        ntt(ntt, cb, false);
        for(int i = 0; i < n; ++i) {
                ca[i] = ca[i] * cb[i] % modulo;
        }
        ntt(ntt, ca, true);
        for(int i = 0; i < n; ++i) {
                ca[i] = ca[i] * qpow(n, modulo - 2) % modulo;
        }
        while(ca.size() > (a.size() + b.size() - 1)) {
                ca.pop_back();
        }
        return ca;
}
```

线性基

```
struct LinearBasis_XOR {
        LinearBasis_XOR() : base(61) {}
        void insert(ll val) {
                for(int i = 60; i >= 0; --i) {
                        if(((val >> i) & 1) == 0) continue;
                        if(base[i] == 0) {
                                base[i] = val;
                                return;
                        }
                        val ^= base[i];
                }
        }
        ll query_max() const {
                ll ans = 0;
                for(int i = 60; i >= 0; --i) {
                        if((ans ^ base[i]) > ans) {
                                ans ^= base[i];
                        }
                }
                return ans;
        }
private:
        vector<ll> base;
};
```

矩阵乘法和快速幂

```
template<class T>
vector<vector<T>> matmul(const vector<vector<T>> &lhs, const vector<vector<T>> &rhs) {
        int M = lhs.size(), N = lhs[0].size(), P = rhs[0].size();
```

```
        vector<vector<T>> ret(M, vector<T>(P, 0));
        for(int i = 0; i < M; ++i) {
                for(int j = 0; j < P; ++j) {
                        for(int k = 0; k < N; ++k) {
                                ret[i][j] = ret[i][j] + lhs[i][k] * rhs[k][j];
                        }
                }
        }
        return ret;
}
template<class T>
vector<vector<T>> matpow(vector<vector<T>> mat, ll N) {
        int M = mat.size();
        vector<vector<T>> ret(M, vector<T>(M, 0));
        for(int i = 0; i < M; ++i) {
                ret[i][i] = static_cast<T>(1);
        }
        while(N) {
                if(N & 1ll) {
                        ret = matmul(ret, mat);
                }
                mat = matmul(mat, mat);
                N >>= 1;
        }
        return ret;
}
```

高斯消元

```
vector<double> gauss(const vector<vector<double>> &A, const vector<double> &B) {
        int n = A.size();
        if(n == 0 || A[0].size() != n) return {};
        vector<vector<double>> aug(n, vector<double>(n + 1));
        for(int i = 0; i < n; ++i) {
                for(int j = 0; j < n; ++j) {
                        aug[i][j] = A[i][j];
                }
                aug[i][n] = B[i];
        }
        for(int i = 0; i < n; ++i) {
                int pivot = -1;
                double mx = 0;
                for(int j = i; j < n; ++j) {
                        if(abs(aug[j][i]) > mx) {
                                mx = abs(aug[j][i]);
                                pivot = j;
                        }
                }
                if(pivot == -1 || mx < 1e-10) return {};
                swap(aug[i], aug[pivot]);
                for(int j = i + 1; j < n; ++j) {
                        double factor = aug[j][i] / aug[i][i];
                        for(int k = i; k <= n; ++k) {
```

```
                                aug[j][k] -= factor * aug[i][k];
                        }
                }
        }
        vector<double> X(n);
        for(int i = n - 1; i >= 0; --i) {
                X[i] = aug[i][n];
                for(int j = i + 1; j < n; ++j) {
                        X[i] -= aug[i][j] * X[j];
                }
                X[i] /= aug[i][i];
        }
        return X;
}
```

构造增广状态转移矩阵

问题	转移矩阵	初始值	结果
$\sum_{k=0}^{n}(aB^k)_{i,j}$	$T=\begin{bmatrix}B & 0\\ e_j\cdot e_i^T & 1\end{bmatrix}$	$z_0=\begin{bmatrix}a\\ a_i\end{bmatrix}$	$z_n=T^nz_0$
$\sum_{k=0}^{n}\langle u,B^kv\rangle$	$T=\begin{bmatrix}B & 0\\ u^T & 1\end{bmatrix}$	$z_0=\begin{bmatrix}v\\ \langle u,v\rangle\end{bmatrix}$	$s_n=T^nz_0[back]$
$\sum_{i=0}^{n}A^i$	$T=\begin{bmatrix}A & I\\ 0 & I\end{bmatrix}$	无	T^{n+1} 的**右上角分块**

Lehmer码

`l[i] = k` 代表后面有k个数比 `a[i]` 小

```
// 均为0-based排列
vector<int> lehmer(const vector<int> &a) {
        int n = a.size();
        vector<int> l(n);
        Fenwick bit(n);
        for(int i = 1; i <= n; ++i) {
                bit.update(i, 1);
        }
        for(int i = 0; i < n; ++i) {
                int x = a[i] + 1;
                l[i] = bit.query(n) - bit.query(x);
                bit.update(x, -1);
        }
        return l;
}
vector<int> rev_lehmer(const vector<int> &l) {
        int n = l.size();
        Treap treap;
```

```
        for(int i = 0; i < n; ++i) {
                treap.insert(i);
        }
        vector<int> ret(n);
        for(int i = 0; i < n; ++i) {
                ret[i] = treap.query_val_by_rank(l[i] + 1);
                treap.erase(ret[i]);
        }
        return ret;
}
```

计算几何

平面凸包

```
struct Point {
        double x, y;
};
double cross(const Point &a, const Point &b, const Point &c) {
        return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
}
vector<Point> andrew(vector<Point> &points) {
        sort(points.begin(), points.end(), [](const Point &a, const Point &b) {
                return a.x == b.x ? a.y < b.y : a.x < b.x;
        });
        int n = points.size();
        vector<Point> stk;
        for(int i = 1; i < n; ++i) {
                while(stk.size() > 1 &&
                        cross(stk[stk.size() - 2], stk.back(), points[i]) <= 0) {
                        stk.pop_back();
                }
                stk.push_back(points[i]);
        }
        int t = stk.size();
        for(int i = n - 1; i >= 0; --i) {
                while(stk.size() > t &&
                        cross(stk[stk.size() - 2], stk.back(), points[i]) <= 0) {
                        stk.pop_back();
                }
                stk.push_back(points[i]);
        }
        return stk;
}
```

字符串

KMP

```
// 得到用于KMP的数组
vector<int> kmp_f(const string &str) {
	vector<int> f(str.size());
	int l = 0, r = 1;
	while(r < str.size()) {
		if(str[r] != str[l]) {
			if(l == 0) {
				f[r] = 0;
				++r;
			} else {
				l = f[l - 1];
			}
		} else {
			f[r] = l + 1;
			++r;
			++l;
		}
	}
	return f;
}
// 寻找第一次在off后面haystack中的needle
int kmp_find(const string &haystack, const string &needle, int off = 0) {
	int n = needle.size(), m = haystack.size();
	vector<int> f = kmp_f(needle);
	int i = 0, j = off;
	while(j < m) {
		if(haystack[j] == needle[i]) {
			++i;
			++j;
			if(i == n) {
				return j - n;
			}
		} else {
			if(i == 0) {
				++j;
			} else {
				i = f[i - 1];
			}
		}
	}
	return string::npos;
}
// 返回haystack中有多少个needle，可重（即10101有两个101）
int kmp_count(const string &haystack, const string &needle) {
	int n = needle.size(), m = haystack.size();
	int ret = 0;
	vector<int> f = kmp_f(needle);
	int i = 0, j = 0;
	while(j < m) {
		if(haystack[j] == needle[i]) {
			++i;
			++j;
			if(i == n) {
				++ret;
				i = f[i - 1];
			}
```

```
                } else {
                        if(i == 0) {
                                ++j;
                        } else {
                                i = f[i - 1];
                        }
                }
        }
        return ret;
}
```

Z函数（扩展KMP）

```
vector<int> zfn(const string &s) {
    int n = s.size();
    vector<int> z(n);
    for(int i = 1, l = 0, r = 0; i < n; ++i) {
        if(i <= r && z[i - l] < r - i + 1) {
            z[i] = z[i - l];
        } else {
            z[i] = max(0, r - i + 1);
            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;
}
```

字典树

```
struct Trie {
        Trie() : nodes(1) {}
        void insert(const string &str) {
                int rt = 0;
                for(int i = 0; i < str.size(); ++i) {
                        if((nodes[rt].nxt[getnum(str[i])]) == -1) {
                                nodes[rt].nxt[getnum(str[i])] = nodes.size();
                                nodes.emplace_back();
                        }
                        nodes[rt].cnt++;
                        rt = nodes[rt].nxt[getnum(str[i])];
                }
                nodes[rt].cnt++;
                nodes[rt].end = true;
        }
        bool find(const string &str) const {
```

```
                int rt = 0;
                for(int i = 0; i < str.size(); ++i) {
                        if((nodes[rt].nxt[getnum(str[i])]) == -1) {
                                return false;
                        }
                        rt = nodes[rt].nxt[getnum(str[i])];
                }
                return nodes[rt].end;
        }
        int find_prefix(const string &str) const {
                int rt = 0;
                for(int i = 0; i < str.size(); ++i) {
                        if((nodes[rt].nxt[getnum(str[i])]) == -1) {
                                return 0;
                        }
                        rt = nodes[rt].nxt[getnum(str[i])];
                }
                return nodes[rt].cnt;
        }
private:
        struct Node {
                array<int, 65> nxt;
                bool end;
                int cnt;
                Node() : end(false), cnt(0) {
                        nxt.fill(-1);
                }
        };
        vector<Node> nodes;
        static int getnum(char x) {
                if(x >= 'A' && x <= 'Z') {
                        return x - 'A';
                } else if(x >= 'a' && x <= 'z') {
                        return x - 'a' + 26;
                } else {
                        return x - '0' + 52;
                }
        }
};
```

贪心

一组交叉直线经过所有点

```
struct Point {
        int x, y;
};
bool cross_all(const vector<Point> &points) {
        bool okx = true, oky = true;
        int xb = -1, yb = -1;
        for(auto [x, y] : points) {
```

```
                if(x != points[0].x) {
                        if(yb == -1) {
                                yb = y;
                        } else if(yb != y) {
                                okx = false;
                        }
                }
                if(y != points[0].y) {
                        if(xb == -1) {
                                xb = x;
                        } else if(xb != x) {
                                oky = false;
                        }
                }
        }
        return okx || oky;
}
```

二分

最长上升子序列

```
int lis(const vector<int> &nums) {
        vector<int> lis;
        for(int i : nums) {
                auto it = lower_bound(lis.begin(), lis.end(), i);
                if(it == lis.end()) {
                        lis.push_back(i);
                } else {
                        *it = i;
                }
        }
        return lis.size();
}
```

二分答案

```
int binary(...) {
        auto check = [&](int val) -> bool {
                // 检查函数，有效返回true，否则false
        };
        int lo = 0, hi = 0x3f3f3f3f;
        int ans = -1;
        while(lo < hi) {
                int mid = (lo + hi) >> 1;
                if(check(mid)) {
                        ans = mid;
                        hi = mid - 1;
                } else {
```

```
                        lo = mid + 1;
                }
        }
        return ans;
}
```

三分

```
int peak(const vector<int> &nums) {
        int n = nums.size();
        int l = 0, r = n - 1;
        int ll = (r - l) / 3, rr = (r - l) * 2 / 3;
        while(l < r) {
                if(nums[ll] < nums[rr]) {
                        l = ll + 1;
                } else {
                        r = rr;
                }
                ll = l + (r - l) / 3;
                rr = l + (r - l) * 2 / 3;
        }
        return l;
}
```

杂项

高精

```
struct StrNum {
        StrNum() noexcept : number("0") {}
        StrNum(long long num) noexcept : number(to_string(num)) {}
        StrNum(const string &num) noexcept : number(num) {}
        StrNum(string &&num) noexcept : number(move(num)) {}
        StrNum operator+(const StrNum &r) const noexcept {
                return StrNum(move(add(number, r.number)));
        }
        StrNum &operator+=(const StrNum &r) noexcept {
                number = move(add(number, r.number));
                return *this;
        }
        StrNum operator-(const StrNum &r) const noexcept {
                return StrNum(move(minus(number, r.number)));
        }
        StrNum &operator-=(const StrNum &r) noexcept {
                number = move(minus(number, r.number));
                return *this;
        }
        StrNum operator*(const StrNum &r) const noexcept {
```

```
                return StrNum(move(mul(number, r.number)));
        }
        StrNum &operator*=(const StrNum &r) noexcept {
                number = move(mul(number, r.number));
                return *this;
        }
        StrNum operator-() const noexcept {
                if(number[0] == '-') return StrNum(number.substr(1, number.size() - 1));
                return StrNum('-' + number);
        }
        bool operator>(const StrNum &r) const noexcept {
                return gt(number, r.number);
        }
        bool operator<(const StrNum &r) const noexcept {
                return gt(r.number, number);
        }
        bool operator>=(const StrNum &r) const noexcept {
                return !gt(r.number, number);
        }
        bool operator<=(const StrNum &r) const noexcept {
                return !gt(number, r.number);
        }
        bool operator==(const StrNum &r) const noexcept {
                return number == r.number;
        }
        friend ostream &operator<<(ostream &os, const StrNum &r) noexcept {
                os << r.number;
                return os;
        }
        friend istream &operator>>(istream &is, StrNum &r) noexcept {
                is >> r.number;
                return is;
        }
private:
        string number;
        static bool gt(const string &a, const string &b) noexcept {
                if(a[0] == b[0] && a[0] == '-') {
                        string suba = a.substr(1, a.size() - 1);
                        string subb = b.substr(1, b.size() - 1);
                        return (gt(subb, suba));
                }
                if(a[0] == '-') return false;
                if(b[0] == '-') return true;
                if(a.size() > b.size()) return true;
                if(a.size() < b.size()) return false;
                int n = a.size();
                for(int i = 0; i < n; i++) {
                        if(a[i] > b[i]) return true;
                        if(a[i] < b[i]) return false;
                }
                return false;
        }
        static string add(const string &a, const string &b) noexcept {
                if(a[0] == '-' && b[0] == '-') {
                        string suba = a.substr(1, a.size() - 1);
                        string subb = b.substr(1, b.size() - 1);
```

```
                return ("-" + add(suba, subb));
        }
        if(a[0] == '-') {
                string suba = a.substr(1, a.size() - 1);
                return minus(b, suba);
        }
        if(b[0] == '-') {
                string subb = b.substr(1, b.size() - 1);
                return minus(a, subb);
        }
        if(a.size() < b.size()) return add(b, a);
        string ans;
        string stra = a, strb = b;
        ranges::reverse(stra);
        ranges::reverse(strb);
        int s = stra.size();
        int carry = 0;
        for(int i = 0; i < s; i++) {
                if(i < strb.size()) {
                        int m = (int)(stra[i] - '0'), n = (int)(strb[i] - '0');
                        int p = m + n + carry;
                        carry = 0;
                        if(p < 10) ans.push_back((char)(p + '0'));
                        else {
                                ans.push_back((char)(p - 10 + '0'));
                                carry = 1;
                        }
                } else {
                        int m = (int)(stra[i] - '0') + carry;
                        carry = 0;
                        if(m < 10) ans.push_back((char)(m + '0'));
                        else {
                                ans.push_back((char)('0'));
                                carry = 1;
                        }
                }
        }
        if(carry == 1) ans.push_back('1');
        ranges::reverse(ans);
        return ans;
}
static string mul(const string &a, const string &b) noexcept {
        if(a == "0" || b == "0") return "0";
        if(a == "1") return b;
        if(b == "1") return a;
        bool negative = (a[0] == '-') ^ (b[0] == '-');
        string stra = (a[0] == '-') ? a.substr(1) : a;
        string strb = (b[0] == '-') ? b.substr(1) : b;
        int n = stra.size();
        int m = strb.size();
        vector<int> result(n + m, 0);
        for(int i = n - 1; i >= 0; --i) {
                for(int j = m - 1; j >= 0; --j) {
                        int mul = (stra[i] - '0') * (strb[j] - '0');
                        int sum = mul + result[i + j + 1];
                        result[i + j + 1] = sum % 10;
```

```
                        result[i + j] += sum / 10;
                }
        }
        string ans;
        for(int num : result) {
                if(!(ans.empty() && num == 0)) {
                        ans.push_back(num + '0');
                }
        }
        return negative ? '-' + ans : ans;
    }
    static string minus(const string &a, const string &b) noexcept {
        if(a == b) return "0";
        if(b[0] == '-') {
                string str = b.substr(1, b.size() - 1);
                return add(a, str);
        }
        if(a[0] == '-') {
                string str = a.substr(1, a.size() - 1);
                return '-' + add(str, b);
        }
        if(!gt(a, b)) {
                return '-' + minus(b, a);
        }
        string ans = "", stra = a, strb = b;
        ranges::reverse(stra);
        ranges::reverse(strb);
        int s = stra.size();
        int subtract = 0;
        for(int i = 0; i < s; i++) {
                if(i < strb.size()) {
                        int k = stra[i] - strb[i] - subtract;
                        subtract = 0;
                        if(k < 0) {
                                subtract = 1;
                                k += 10;
                        }
                        ans.push_back((char)(k + '0'));
                } else {
                        int k = stra[i] - subtract - '0';
                        subtract = 0;
                        if(k < 0) {
                                subtract = 1;
                                k += 10;
                        }
                        ans.push_back((char)(k + '0'));
                }
        }
        while(ans[ans.size() - 1] == '0') {
                ans.pop_back();
        }
        ranges::reverse(ans);
        return ans;
    }
};
```

快读

```
struct Qread {
        Qread() : state(true) {}
        template<integral T> Qread &operator>>(T &val) {
                if(!state) {
                    val = 0;
                    return *this;
                }
                T x = 0, f = 1;
                char ch = getchar();
                while(ch < '0' || ch > '9') {
                        if(ch == EOF) {
                                state = false;
                                val = 0;
                                return *this;
                        }
                        if(ch == '-') {
                                f = -1;
                        }
                        ch = getchar();
                }
                while(ch >= '0' && ch <= '9') {
                        x = x * 10 + ch - '0';
                        ch = getchar();
                }
                val = x * f;
                return *this;
        }
        explicit operator bool() const {
                return state;
        }
private:
        bool state;
};
Qread qread;
```