1. 数据结构

1.并查集------------------------(1)

2.树状数组------------------------ (1)

3.哈希------------------------ (1)

4.线段树------------------------ (1)

5.st表------------------------(4)

二、图论

1.最短路------------------------ (4)

2.连通分量------------------------ (4)

3.LCA------------------------ (5)

4.最大流------------------------ (6)

5.费用流------------------------ (7)

6.快速费用流------------------------ (8)

7.树链剖分------------------------ (9)

8.k祖先+树上差分------------------------ (11)

三、数学

1.多项式------------------------ (11)

2.因数分解------------------------ (14)

3.线性筛------------------------ (15)

四、杂项

1.快读------------------------ (15)

2.二分------------------------ (16)

3.莫队------------------------ (16)

4.随机+计时------------------------ (16)

5.分数计算------------------------ (16)

6.对拍------------------------ (17)

一、数据结构：

1.并查集：

class dsu {

public:

vector<int> p;

int n;

vector<int> rank;

dsu(int \_n) : n(\_n) {

p.resize(n);

rank.resize(n);

fill(rank.begin(), rank.end(), 0);

iota(p.begin(), p.end(), 0);

}

inline int get(int x) {

return (x == p[x] ? x : (p[x] = get(p[x])));

}

inline bool unite(int x, int y) {

x = get(x), y = get(y);

if (x == y) return false;

if (rank[x] <= rank[y])

p[x] = y;

else

p[y] = x;

if (p[x] == p[y] && x != y)

rank[y]++;

return true;

}

};

2.树状数组

template<typename T>

class fenwick {

public :

int n;

vector<T> c;

vector<T> a;

int lowbit(int x) {

return x & (-x);

}

void add(int x, int d) {

a[x] += d;

for (int i = x; i <= n; i += lowbit(i))

c[i] += d;

}

T sum(int x) {

T ans = 0;

for (int i = x; i > 0; i -= lowbit(i))

ans += c[i];

return ans;

}

T sum(int l, int r) {

return sum(r) - sum(l - 1);

}

fenwick(const vector<T>& x) {

a = x;

n = (int)a.size() - 1;

c.resize(n + 1);

for (int i = 1; i <= n; i++) {

for (int j = i - lowbit(i) + 1; j <= i; j++) {

c[i] = c[i] + a[j];

}

}

}

//1-indexed

};

3.哈希

const int base = 323131;

for(int i = 1; i <= n; i++) {

hashb[i] = add(mul(base , hashb[i - 1]) , b[i]);

}

for(int i = 1; i <= 2 \* n; i++) {

hashc[i] = add(mul(base , hashc[i - 1]) , c[i]);

}

auto isequal = [&](int bl, int br, int cl, int cr) {

if(bl < 1 || br > n || cl < 1 || cr > 2 \* n) return false;

int xb = sub(hashb[br], mul(power(base, br - bl + 1), hashb[bl - 1]));

int xc = sub(hashc[cr], mul(power(base, cr - cl + 1), hashc[cl - 1]));

return xb == xc;

};

4.线段树

class segtree {

public:

struct node {

int sum = 0;

int add = 0;

void apply(int l, int r, int v) {

add += v;

sum += (r - l + 1) \* v;

}

};

node unite(const node& a, const node& b) const {

node res;

res.sum = a.sum + b.sum;

return res;

}

inline void push(int x, int l, int r) {

int y = (l + r) >> 1;

int z = x + ((y - l + 1) << 1);

if (tree[x].add != 0) {

tree[x + 1].apply(l, y, tree[x].add);

tree[z].apply(y + 1, r, tree[x].add);

tree[x].add = 0;

}

}

inline void pulL(int x, int z) {

tree[x] = unite(tree[x + 1], tree[z]);

}

int n;

vector<node> tree;

void build(int x, int l, int r) {

if (l == r) {

return;

}

int y = (l + r) >> 1;

int z = x + ((y - l + 1) << 1);

build(x + 1, l, y);

build(z, y + 1, r);

pulL(x, z);

}

template <typename M>

void build(int x, int l, int r, const vector<M>& v) {

if (l == r) {

tree[x].apply(l, r, v[l]);

return;

}

int y = (l + r) >> 1;

int z = x + ((y - l + 1) << 1);

build(x + 1, l, y, v);

build(z, y + 1, r, v);

pulL(x, z);

}

node get(int x, int l, int r, int lL, int rr) {

if (lL <= l && r <= rr) {

return tree[x];

}

int y = (l + r) >> 1;

int z = x + ((y - l + 1) << 1);

push(x, l, r);

node res{};

if (rr <= y) {

res = get(x + 1, l, y, lL, rr);

}

else {

if (lL > y) {

res = get(z, y + 1, r, lL, rr);

}

else {

res = unite(get(x + 1, l, y, lL, rr), get(z, y + 1, r, lL, rr));

}

}

pulL(x, z);

return res;

}

template <typename... M>

void modify(int x, int l, int r, int lL, int rr, const M&... v) {

if (lL <= l && r <= rr) {

tree[x].apply(l, r, v...);

return;

}

int y = (l + r) >> 1;

int z = x + ((y - l + 1) << 1);

push(x, l, r);

if (lL <= y) {

modify(x + 1, l, y, lL, rr, v...);

}

if (rr > y) {

modify(z, y + 1, r, lL, rr, v...);

}

pulL(x, z);

}

segtree(int \_n) : n(\_n) {

assert(n > 0);

tree.resize(2 \* n - 1);

build(0, 0, n - 1);

}

int find\_first\_knowingly(int x, int l, int r, const function<bool(const node&)> &f) {

if (l == r) {

return l;

}

push(x, l, r);

int y = (l + r) >> 1;

int z = x + ((y - l + 1) << 1);

int res;

if (f(tree[x + 1])) {

res = find\_first\_knowingly(x + 1, l, y, f);

} else {

res = find\_first\_knowingly(z, y + 1, r, f);

}

pulL(x, z);

return res;

}

int find\_first(int x, int l, int r, int ll, int rr, const function<bool(const node&)> &f) {

if (ll <= l && r <= rr) {

if (!f(tree[x])) {

return -1;

}

return find\_first\_knowingly(x, l, r, f);

}

push(x, l, r);

int y = (l + r) >> 1;

int z = x + ((y - l + 1) << 1);

int res = -1;

if (ll <= y) {

res = find\_first(x + 1, l, y, ll, rr, f);

}

if (rr > y && res == -1) {

res = find\_first(z, y + 1, r, ll, rr, f);

}

pulL(x, z);

return res;

}

int find\_last\_knowingly(int x, int l, int r, const function<bool(const node&)> &f) {

if (l == r) {

return l;

}

push(x, l, r);

int y = (l + r) >> 1;

int z = x + ((y - l + 1) << 1);

int res;

if (f(tree[z])) {

res = find\_last\_knowingly(z, y + 1, r, f);

} else {

res = find\_last\_knowingly(x + 1, l, y, f);

}

pulL(x, z);

return res;

}

int find\_last(int x, int l, int r, int ll, int rr, const function<bool(const node&)> &f) {

if (ll <= l && r <= rr) {

if (!f(tree[x])) {

return -1;

}

return find\_last\_knowingly(x, l, r, f);

}

push(x, l, r);

int y = (l + r) >> 1;

int z = x + ((y - l + 1) << 1);

int res = -1;

if (rr > y) {

res = find\_last(z, y + 1, r, ll, rr, f);

}

if (ll <= y && res == -1) {

res = find\_last(x + 1, l, y, ll, rr, f);

}

pulL(x, z);

return res;

}

template <typename M>

segtree(const vector<M>& v) {

n = v.size();

assert(n > 0);

tree.resize(2 \* n - 1);

build(0, 0, n - 1, v);

}

node get(int lL, int rr) {

assert(0 <= lL && lL <= rr && rr <= n - 1);

return get(0, 0, n - 1, lL, rr);

}

node get(int p) {

assert(0 <= p && p <= n - 1);

return get(0, 0, n - 1, p, p);

}

int find\_first(int ll, int rr, const function<bool(const node&)> &f) {

assert(0 <= ll && ll <= rr && rr <= n - 1);

return find\_first(0, 0, n - 1, ll, rr, f);

}

int find\_last(int ll, int rr, const function<bool(const node&)> &f) {

assert(0 <= ll && ll <= rr && rr <= n - 1);

return find\_last(0, 0, n - 1, ll, rr, f);

}

template <typename... M>

void modify(int lL, int rr, const M&... v) {

assert(0 <= lL && lL <= rr && rr <= n - 1);

modify(0, 0, n - 1, lL, rr, v...);

}

};

5.st表

template <typename T, class F = function<T(const T&, const T&)>>

class SparseTable {

public:

int n;

vector<vector<T>> mat;

F func;

SparseTable(const vector<T>& a, const F& f) : func(f) {

n = static\_cast<int>(a.size());

int max\_log = 32 - \_\_builtin\_clz(n);

mat.resize(max\_log);

mat[0] = a;

for (int j = 1; j < max\_log; j++) {

mat[j].resize(n - (1 << j) + 1);

for (int i = 0; i <= n - (1 << j); i++) {

mat[j][i] = func(mat[j - 1][i], mat[j - 1][i + (1 << (j - 1))]);

}

}

}

T get(int from, int to) const {

assert(0 <= from && from <= to && to <= n - 1);

int lg = 32 - \_\_builtin\_clz(to - from + 1) - 1;

return func(mat[lg][from], mat[lg][to - (1 << lg) + 1]);

}

};

二、图论（树）

1.最短路

template <typename T>

vector <T> dijkstra(const graph<T>& g, int start) {

assert(0 <= start && start < g.n);

vector<T> dist(g.n, numeric\_limits<T>::max());

priority\_queue<pair<T, int>, vector<pair<T, int> >, greater<pair<T, int> > > s;

dist[start] = 0;

s.emplace(dist[start], start);

while (!s.empty()) {

T expected = s.top().first;

int i = s.top().second;

s.pop();

if (dist[i] != expected) {

continue;

}

for (int id : g.g[i]) {

if (g.ignore != nullptr && g.ignore(id)) {

continue;

}

auto& e = g.edges[id];

int to = e.from ^ e.to ^ i;

if (dist[i] + e.cost < dist[to]) {

dist[to] = dist[i] + e.cost;

s.emplace(dist[to], to);

}

}

}

return dist;

// returns numeric\_limits<T>::max() if there's no path

}

2.联通分量+缩点

template <typename T> class graph {

public:

struct edge {

int from;

int to;

T cost;

};

vector<edge> edges;

vector< vector<int> > g;

vector<int> deg;

int n;

function<bool(int)> ignore;

graph(int \_n) : n(\_n) {

g.resize(n);

deg.resize(n);

ignore = nullptr;

}

int add(int from, int to, T cost = 1) {

int id = (int)edges.size();

g[from].push\_back(id);

deg[to]++;

//g[to].push\_back(id);

edges.push\_back({ from, to, cost });

return id;

}

graph<T> reverse() const {

graph<T> rev(n);

for (auto &e : edges) {

rev.add(e.to, e.from, e.cost);

}

return rev;

}

vector<int> topo() {

vector<int> ret(n);

queue<int> q;

for(int i = 0; i < n; i++) {

if(deg[i] == 0) {

q.push(i);

}

}

while(!q.empty()) {

int u = q.front(); q.pop();

for(int id : g[u]) {

int v = edges[id].to;

if(--deg[v] == 0) {

q.push(v);

ret[v] = ret[u] + 1;

}

}

}

return ret;

}

};

template <typename T> vector<int> find\_scc(const graph<T> &g) {

graph<T> g\_rev = g.reverse();

vector<int> order;

vector<bool> was(g.n, false);

function<void(int)> dfs1 = [&](int v) {

was[v] = true;

for (int id : g.g[v]) {

if (g.ignore != nullptr && g.ignore(id)) {

continue;

}

auto &e = g.edges[id];

int to = e.to;

if (!was[to]) {

dfs1(to);

}

}

order.push\_back(v);

};

for (int i = 0; i < g.n; i++) {

if (!was[i]) {

dfs1(i);

}

}

vector<int> c(g.n, -1);

function<void(int)> dfs2 = [&](int v) {

for (int id : g\_rev.g[v]) {

if (g\_rev.ignore != nullptr && g\_rev.ignore(id)) {

continue;

}

auto &e = g\_rev.edges[id];

int to = e.to;

if (c[to] == -1) {

c[to] = c[v];

dfs2(to);

}

}

};

int cnt = 0;

for (int id = g.n - 1; id >= 0; id--) {

int i = order[id];

if (c[i] != -1) {

continue;

}

c[i] = cnt++;

dfs2(i);

}

return c;

// c[i] <= c[j] for every edge i -> j

}

template <typename T> graph<T> Shrink(graph<T>& g) {

auto s = find\_scc(g);

int N = \*max\_element(s.begin(), s.end()) + 1;

graph<int> G(N);

vector<int> vis(N, 0);

for(int i = 0; i < g.n; i++) {

for(auto& id : g.g[i]) {

auto& [u, v, c] = g.edges[id];

if(s[u] == s[v] || vis[s[v]] ) continue;

G.add(s[u], s[v]);

vis[s[v]] = 1;

}

for(auto& id : g.g[i]) {

auto& [u, v, c] = g.edges[id];

vis[s[v]] = 0;

}

}

return G;

}

3. 求LCA

class Tree {

public:

struct edge {

int from;

int to;

};

vector<edge> edges;

vector< vector<int> > g;

int n, root;

vector<int> depth;

vector<int> lg;

vector<vector<int> > fa;

bool pre = 0;

Tree(int \_n, int \_r) : n(\_n), root(\_r) {

g.resize(n);

depth.resize(n);

lg.resize(n);

fa.assign(n, vector<int>(23, 0));

}

int add(int from, int to) {

int id = (int)edges.size();

g[from].push\_back(id);

g[to].push\_back(id);

edges.push\_back({ from, to });

return id;

}

int LCA(int x, int y) {

//1 - indexed

if(!pre) {

for(int i = 1; i <= n; i++) {

lg[i] = lg[i - 1] + (1 << lg[i - 1] == i);

}

function<void(int, int)> dfs = [&](int now, int fath) {

fa[now][0] = fath;

depth[now] = depth[fath] + 1;

for (int i = 1; i <= lg[depth[now]]; ++i) {

fa[now][i] = fa[fa[now][i - 1]][i - 1];

}

for(int id : g[now]) {

int v = edges[id].from ^ edges[id].to ^ now;

if (v != fath) dfs(v, now);

}

};

dfs(root, 0);

pre = 1;

}

if (depth[x] < depth[y]) swap(x, y);

while (depth[x] > depth[y]) {

x = fa[x][lg[depth[x] - depth[y]] - 1];

}

if (x == y) return x;

for (int k = lg[depth[x]] - 1; k >= 0; --k) {

if (fa[x][k] != fa[y][k]) {

x = fa[x][k], y = fa[y][k];

}

}

return fa[x][0];

}

};

4.最大流

template <class Cap> struct mf\_graph {

public:

mf\_graph() : \_n(0) {}

explicit mf\_graph(int n) : \_n(n), g(n) {}

int add\_edge(int from, int to, Cap cap) {

assert(0 <= from && from < \_n);

assert(0 <= to && to < \_n);

assert(0 <= cap);

int m = int(pos.size());

pos.push\_back({from, int(g[from].size())});

int from\_id = int(g[from].size());

int to\_id = int(g[to].size());

if (from == to) to\_id++;

g[from].push\_back(\_edge{to, to\_id, cap});

g[to].push\_back(\_edge{from, from\_id, 0});

return m;

}

struct edge {

int from, to;

Cap cap, flow;

};

edge get\_edge(int i) {

int m = int(pos.size());

assert(0 <= i && i < m);

auto \_e = g[pos[i].first][pos[i].second];

auto \_re = g[\_e.to][\_e.rev];

return edge{pos[i].first, \_e.to, \_e.cap + \_re.cap, \_re.cap};

}

std::vector<edge> edges() {

int m = int(pos.size());

std::vector<edge> result;

for (int i = 0; i < m; i++) {

result.push\_back(get\_edge(i));

}

return result;

}

void change\_edge(int i, Cap new\_cap, Cap new\_flow) {

int m = int(pos.size());

assert(0 <= i && i < m);

assert(0 <= new\_flow && new\_flow <= new\_cap);

auto& \_e = g[pos[i].first][pos[i].second];

auto& \_re = g[\_e.to][\_e.rev];

\_e.cap = new\_cap - new\_flow;

\_re.cap = new\_flow;

}

Cap flow(int s, int t) {

return flow(s, t, std::numeric\_limits<Cap>::max());

}

Cap flow(int s, int t, Cap flow\_limit) {

assert(0 <= s && s < \_n);

assert(0 <= t && t < \_n);

assert(s != t);

std::vector<int> level(\_n), iter(\_n);

queue<int> que;

auto bfs = [&]() {

std::fill(level.begin(), level.end(), -1);

level[s] = 0;

while(!que.empty()) que.pop();

que.push(s);

while (!que.empty()) {

int v = que.front();

que.pop();

for (auto e : g[v]) {

if (e.cap == 0 || level[e.to] >= 0) continue;

level[e.to] = level[v] + 1;

if (e.to == t) return;

que.push(e.to);

}

}

};

auto dfs = [&](auto self, int v, Cap up) {

if (v == s) return up;

Cap res = 0;

int level\_v = level[v];

for (int& i = iter[v]; i < int(g[v].size()); i++) {

\_edge& e = g[v][i];

if (level\_v <= level[e.to] || g[e.to][e.rev].cap == 0) continue;

Cap d =

self(self, e.to, std::min(up - res, g[e.to][e.rev].cap));

if (d <= 0) continue;

g[v][i].cap += d;

g[e.to][e.rev].cap -= d;

res += d;

if (res == up) return res;

}

level[v] = \_n;

return res;

};

Cap flow = 0;

while (flow < flow\_limit) {

bfs();

if (level[t] == -1) break;

std::fill(iter.begin(), iter.end(), 0);

Cap f = dfs(dfs, t, flow\_limit - flow);

if (!f) break;

flow += f;

}

return flow;

}

std::vector<bool> min\_cut(int s) {

std::vector<bool> visited(\_n);

queue<int> que;

que.push(s);

while (!que.empty()) {

int p = que.front();

que.pop();

visited[p] = true;

for (auto e : g[p]) {

if (e.cap && !visited[e.to]) {

visited[e.to] = true;

que.push(e.to);

}

}

}

return visited;

}

private:

int \_n;

struct \_edge {

int to, rev;

Cap cap;

};

std::vector<std::pair<int, int>> pos;

std::vector<std::vector<\_edge>> g;

};

5.普通费用流

template <typename T, typename C>

class mcmf {

public:

static constexpr T eps = (T) 1e-9;

struct edge {

int from;

int to;

T c;

T f;

C cost;

};

vector< vector<int> > g;vector<edge> edges;

vector<C> d;vector<int> q;

vector<bool> in\_queue;vector<int> pe;

int n, st, fin;

T flow; C cost;

mcmf(int \_n, int \_st, int \_fin) : n(\_n), st(\_st), fin(\_fin) {

assert(0 <= st && st < n && 0 <= fin && fin < n && st != fin);

g.resize(n);

d.resize(n);

in\_queue.resize(n);

pe.resize(n);

flow = 0;

cost = 0;

}

void clear\_flow() {

for (const edge &e : edges) {

e.f = 0;

}

flow = 0;

}

void add(int from, int to, T forward\_cap, T backward\_cap, C cost) {

assert(0 <= from && from < n && 0 <= to && to < n);

g[from].push\_back((int) edges.size());

edges.push\_back({from, to, forward\_cap, 0, cost});

g[to].push\_back((int) edges.size());

edges.push\_back({to, from, backward\_cap, 0, -cost});

}

bool expath() {

fill(d.begin(), d.end(), numeric\_limits<C>::max());

q.clear();

q.push\_back(st);

d[st] = 0;

in\_queue[st] = true;

int beg = 0;

bool found = false;

while (beg < (int) q.size()) {

int i = q[beg++];

if (i == fin) {

found = true;

}

in\_queue[i] = false;

for (int id : g[i]) {

const edge &e = edges[id];

if (e.c - e.f > eps && d[i] + e.cost < d[e.to]) {

d[e.to] = d[i] + e.cost;

pe[e.to] = id;

if (!in\_queue[e.to]) {

q.push\_back(e.to);

in\_queue[e.to] = true;

}

}

}

}

if (found) {

T push = numeric\_limits<T>::max();

int v = fin;

while (v != st) {

const edge &e = edges[pe[v]];

push = min(push, e.c - e.f);

v = e.from;

}

v = fin;

while (v != st) {

edge &e = edges[pe[v]];

e.f += push;

edge &back = edges[pe[v] ^ 1];

back.f -= push;

v = e.from;

}

flow += push;

cost += push \* d[fin];

}

return found;

}

pair<T, C> max\_flow\_min\_cost() {

while (expath()) {}

return make\_pair(flow, cost);

}

};

6.无负边快速费用流

template <class E> struct csr {

vector<int> start;

vector<E> elist;

explicit csr(int n, const vector<pair<int, E>>& edges)

: start(n + 1), elist(edges.size()) {

for (auto e : edges) {

start[e.first + 1]++;

}

for (int i = 1; i <= n; i++) {

start[i] += start[i - 1];

}

auto counter = start;

for (auto e : edges) {

elist[counter[e.first]++] = e.second;

}

}

};

template <class Cap, class Cost> struct mcf\_graph {

public:

mcf\_graph() {}

explicit mcf\_graph(int n) : \_n(n) {}

int add\_edge(int from, int to, Cap cap, Cost cost) {

assert(0 <= from && from < \_n);

assert(0 <= to && to < \_n);

assert(0 <= cap);

assert(0 <= cost);

int m = int(\_edges.size());

\_edges.push\_back({from, to, cap, 0, cost});

return m;

}

struct edge {

int from, to;

Cap cap, flow;

Cost cost;

};

edge get\_edge(int i) {

int m = int(\_edges.size());

assert(0 <= i && i < m);

return \_edges[i];

}

vector<edge> edges() { return \_edges; }

pair<Cap, Cost> flow(int s, int t) {

return flow(s, t, numeric\_limits<Cap>::max());

}

pair<Cap, Cost> flow(int s, int t, Cap flow\_limit) {

return slope(s, t, flow\_limit).back();

}

vector<pair<Cap, Cost>> slope(int s, int t) {

return slope(s, t, numeric\_limits<Cap>::max());

}

vector<pair<Cap, Cost>> slope(int s, int t, Cap flow\_limit) {

assert(0 <= s && s < \_n);

assert(0 <= t && t < \_n);

assert(s != t);

int m = int(\_edges.size());

vector<int> edge\_idx(m);

auto g = [&]() {

vector<int> degree(\_n), redge\_idx(m);

vector<pair<int, \_edge>> elist;

elist.reserve(2 \* m);

for (int i = 0; i < m; i++) {

auto e = \_edges[i];

edge\_idx[i] = degree[e.from]++;

redge\_idx[i] = degree[e.to]++;

elist.push\_back({e.from, {e.to, -1, e.cap - e.flow, e.cost}});

elist.push\_back({e.to, {e.from, -1, e.flow, -e.cost}});

}

auto \_g = csr<\_edge>(\_n, elist);

for (int i = 0; i < m; i++) {

auto e = \_edges[i];

edge\_idx[i] += \_g.start[e.from];

redge\_idx[i] += \_g.start[e.to];

\_g.elist[edge\_idx[i]].rev = redge\_idx[i];

\_g.elist[redge\_idx[i]].rev = edge\_idx[i];

}

return \_g;

}();

auto result = slope(g, s, t, flow\_limit);

for (int i = 0; i < m; i++) {

auto e = g.elist[edge\_idx[i]];

\_edges[i].flow = \_edges[i].cap - e.cap;

}

return result;

}

private:

int \_n;

vector<edge> \_edges;

struct \_edge {

int to, rev;

Cap cap;

Cost cost;

};

vector<pair<Cap, Cost>> slope(csr<\_edge>& g,int s,int t,Cap flow\_limit) {

vector<pair<Cost, Cost>> dual\_dist(\_n);

vector<int> prev\_e(\_n);

vector<bool> vis(\_n);

struct Q {

Cost key;

int to;

bool operator<(Q r) const { return key > r.key; }

};

vector<int> que\_min;

vector<Q> que;

auto dual\_ref = [&]() {

for (int i = 0; i < \_n; i++) {

dual\_dist[i].second = numeric\_limits<Cost>::max();

}

fill(vis.begin(), vis.end(), false);

que\_min.clear();

que.clear();

size\_t heap\_r = 0;

dual\_dist[s].second = 0;

que\_min.push\_back(s);

while (!que\_min.empty() || !que.empty()) {

int v;

if (!que\_min.empty()) {

v = que\_min.back();

que\_min.pop\_back();

} else {

while (heap\_r < que.size()) {

heap\_r++;

push\_heap(que.begin(), que.begin() + heap\_r);

}

v = que.front().to;

pop\_heap(que.begin(), que.end());

que.pop\_back();

heap\_r--;

}

if (vis[v]) continue;

vis[v] = true;

if (v == t) break;

Cost dual\_v = dual\_dist[v].first, dist\_v = dual\_dist[v].second;

for (int i = g.start[v]; i < g.start[v + 1]; i++) {

auto e = g.elist[i];

if (!e.cap) continue;

Cost cost = e.cost - dual\_dist[e.to].first + dual\_v;

if (dual\_dist[e.to].second - dist\_v > cost) {

Cost dist\_to = dist\_v + cost;

dual\_dist[e.to].second = dist\_to;

prev\_e[e.to] = e.rev;

if (dist\_to == dist\_v) {

que\_min.push\_back(e.to);

} else {

que.push\_back(Q{dist\_to, e.to});

}

}

}

}

if (!vis[t]) {

return false;

}

for (int v = 0; v < \_n; v++) {

if (!vis[v]) continue;

dual\_dist[v].first -= dual\_dist[t].second - dual\_dist[v].second;

}

return true;

};

Cap flow = 0;

Cost cost = 0, prev\_cost\_per\_flow = -1;

vector<pair<Cap, Cost>> result = {{Cap(0), Cost(0)}};

while (flow < flow\_limit) {

if (!dual\_ref()) break;

Cap c = flow\_limit - flow;

for (int v = t; v != s; v = g.elist[prev\_e[v]].to) {

c = min(c, g.elist[g.elist[prev\_e[v]].rev].cap);

}

for (int v = t; v != s; v = g.elist[prev\_e[v]].to) {

auto& e = g.elist[prev\_e[v]];

e.cap += c;

g.elist[e.rev].cap -= c;

}

Cost d = -dual\_dist[s].first;

flow += c;

cost += c \* d;

if (prev\_cost\_per\_flow == d) {

result.pop\_back();

}

result.push\_back({flow, cost});

prev\_cost\_per\_flow = d;

}

return result;

}

};

6.树链剖分

const int MaxN = 100005;

struct graph\_t

{

int cnte = 0;

int head[MaxN], to[MaxN \* 2 + 5], next[MaxN \* 2 + 5];

inline void add\_edge(int u, int v)

{

to[++cnte] = v;

next[cnte] = head[u];

head[u] = cnte;

}

};

int A[MaxN]; // i节点的点权

int Fa[MaxN], Dep[MaxN], Siz[MaxN], Wson[MaxN];

// i的父亲，i的深度，以i为根的子树的节点个数，i节点的重儿子

int Id[MaxN], Top[MaxN], Dfc;

// i的dfn序的下标，i所在的重链的起点

graph\_t Gr;

void dfs1(int u)

{

Siz[u] = 1;

for (int i = Gr.head[u]; i; i = Gr.next[i])

{

int v = Gr.to[i];

if (v == Fa[u])

continue;

Fa[v] = u;

Dep[v] = Dep[u] + 1;

dfs1(v);

Siz[u] += Siz[v];

if (Siz[v] > Siz[Wson[u]])

Wson[u] = v;

}

}

void dfs2(int u, int chain)

{

Top[u] = chain;

Id[u] = ++Dfc;

if (Wson[u] != 0)

dfs2(Wson[u], chain);

for (int i = Gr.head[u]; i; i = Gr.next[i])

{

int v = Gr.to[i];

if (v == Fa[u] || v == Wson[u])

continue;

dfs2(v, v);

}

}

inline int getLca(int u, int v)

{

while (Top[u] != Top[v])

{

if (Dep[Top[u]] < Dep[Top[v]])

std::swap(u, v);

u = Fa[Top[u]];

}

return Dep[u] < Dep[v] ? u : v;

}

void solve() {

int n, m, r;

cin >> n >> m >> r >> mod;

for(int i = 1; i <= n; i++) {

cin >> A[i];

}

segtree st(MaxN);

auto sum = [&](int x,int y) {

long long ans = 0;

int fx = Top[x], fy = Top[y];

while(fx != fy)

{

if(Dep[fx] >= Dep[fy])

{

ans = add(ans, st.get(Id[fx],Id[x]).sum);

x = Fa[fx], fx = Top[x];

}

else

{

ans = add(ans, st.get(Id[fy],Id[y]).sum);

y = Fa[fy], fy = Top[y];

}

}

ans = add(ans, st.get(min(Id[x], Id[y]), max(Id[x], Id[y])).sum);

return ans;

};

auto updates = [&](int x,int y,int c)

{

int fx = Top[x], fy = Top[y];

while(fx != fy)

{

if(Dep[fx] >= Dep[fy])

{

st.modify(Id[fx], Id[x], c);

x = Fa[fx], fx = Top[x];

}

else

{

st.modify(Id[fy], Id[y], c);

y = Fa[fy], fy = Top[y];

}

}

st.modify(min(Id[x], Id[y]), max(Id[x], Id[y]), c);

};

for(int i = 0; i < n - 1; i++) {

int u , v;

cin >> u >> v;

Gr.add\_edge(u, v);

Gr.add\_edge(v, u);

}

dfs1(r);

dfs2(r, r);

for(int i = 1; i <= n; i++) {

st.modify(Id[i], Id[i], A[i]);

}

for(int i = 0; i < m; i++) {

int op; cin >> op;

if(op == 1) {

int x, y, z;

cin >> x >> y >> z;

updates(x, y, z);

}

else if(op == 2) {

int x, y;

cin >> x >> y;

cout << sum(x, y) << '\n';

}

else if(op == 3) {

int x, z;

cin >> x >> z;

st.modify(Id[x], Id[x] + Siz[x] - 1, z);

}

else {

int x;

cin >> x;

cout << st.get(Id[x], Id[x] + Siz[x] - 1).sum << '\n';

}

}

}

7. 树上第K级祖先+树上差分

#include <bits/stdc++.h>

using namespace std;

#define maxn 2000005

#define ll long long

#define res register int

struct Node{

int to,next;

};

Node edge[maxn<<2]; //链式前向星要多开几倍数组

int head[maxn<<2],power[maxn],n,m,d[maxn],fa[maxn][23],num;

inline int read(){ //快读

int s=0;

char c=getchar();

while (c<'0' || c>'9') c=getchar();

while (c>='0' && c<='9') s=s\*10+c-'0',c=getchar();

return s;

}

//链式前向星

inline void add(int x,int y){edge[++num].to=y,edge[num].next=head[x],head[x]=num;}

//接下来是初始化

inline void work(int u,int fath){

d[u]=d[fath]+1,fa[u][0]=fath;

for (res i=0;fa[u][i];++i) fa[u][i+1]=fa[fa[u][i]][i];

for (res i=head[u];i;i=edge[i].next){

int e=edge[i].to;

if (e!=fath) work(e,u);

}

}

//倍增求LCA

inline int Lca(int u,int v){

if (d[u]>d[v]) swap(u,v);

for (res i=20;i>=0;--i) if (d[u]<=d[v]-(1<<i)) v=fa[v][i];

if (u==v) return u;

for (res i=20;i>=0;--i) if (fa[u][i]!=fa[v][i]) u=fa[u][i],v=fa[v][i];

return fa[u][0];

}

//累计

int ans[maxn];

inline void Get(int u,int fath){

for (res i=head[u];i;i=edge[i].next){

int e=edge[i].to;

if (e==fath) continue;

Get(e,u);

power[u]+=power[e];

}

ans[u] = power[u];

}

int getk(int x, int k) { //求k级祖先

if (!k) return x;

int index = log2(k); //得到最高位1的位置

//int index = log2(k & -k); 我们也可以这样得到最低位1的位置

return getk(fa[x][index], k - (1 << index));

}

int a[maxn];

int main(){

#ifdef LOCAL

freopen("in.txt", "r", stdin);

freopen("out.txt", "w", stdout);

#endif

n=read();

int x,y;

for (res i=1;i<n;++i){

x=read(),y=read();

add(x,y); add(y,x);

}

work(1,0);

for(int i = 1; i <= n; i++) {

a[i]= read();

}

for (res i=1; i<= n; ++i){

int x = i;

int y = getk(i, a[i]);

int lca=Lca(x,y);

++power[x];++power[y];--power[lca];--power[fa[lca][0]]; //树上差分

}

Get(1,0);

for(int i = 1; i <= n; i++)

printf("%d ", ans[i]);

return 0;

}

四、数论

1.多项式杂

constexpr int P = 998244353;

using i64 = long long;

// assume -P <= x < 2P

int norm(int x) {

if (x < 0) {

x += P;

}

if (x >= P) {

x -= P;

}

return x;

}

template<class T>

T power(T a, int b) {

T res = 1;

for (; b; b /= 2, a \*= a) {

if (b % 2) {

res \*= a;

}

}

return res;

}

struct Z {

int x;

Z(int x = 0) : x(norm(x)) {}

int val() const {

return x;

}

Z operator-() const {

return Z(norm(P - x));

}

Z inv() const {

assert(x != 0);

return power(\*this, P - 2);

}

Z &operator\*=(const Z &rhs) {

x = i64(x) \* rhs.x % P;

return \*this;

}

Z &operator+=(const Z &rhs) {

x = norm(x + rhs.x);

return \*this;

}

Z &operator-=(const Z &rhs) {

x = norm(x - rhs.x);

return \*this;

}

Z &operator/=(const Z &rhs) {

return \*this \*= rhs.inv();

}

friend Z operator\*(const Z &lhs, const Z &rhs) {

Z res = lhs;

res \*= rhs;

return res;

}

friend Z operator+(const Z &lhs, const Z &rhs) {

Z res = lhs;

res += rhs;

return res;

}

friend Z operator-(const Z &lhs, const Z &rhs) {

Z res = lhs;

res -= rhs;

return res;

}

friend Z operator/(const Z &lhs, const Z &rhs) {

Z res = lhs;

res /= rhs;

return res;

}

friend std::istream &operator>>(std::istream &is, Z &a) {

i64 v;

is >> v;

a = Z(v);

return is;

}

friend std::ostream &operator<<(std::ostream &os, const Z &a) {

return os << a.val();

}

};

std::vector<int> rev;

std::vector<Z> roots{0, 1};

void dft(std::vector<Z> &a) {

int n = a.size();

if (int(rev.size()) != n) {

int k = \_\_builtin\_ctz(n) - 1;

rev.resize(n);

for (int i = 0; i < n; i++) {

rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;

}

}

for (int i = 0; i < n; i++) {

if (rev[i] < i) {

std::swap(a[i], a[rev[i]]);

}

}

if (int(roots.size()) < n) {

int k = \_\_builtin\_ctz(roots.size());

roots.resize(n);

while ((1 << k) < n) {

Z e = power(Z(3), (P - 1) >> (k + 1));

for (int i = 1 << (k - 1); i < (1 << k); i++) {

roots[2 \* i] = roots[i];

roots[2 \* i + 1] = roots[i] \* e;

}

k++;

}

}

for (int k = 1; k < n; k \*= 2) {

for (int i = 0; i < n; i += 2 \* k) {

for (int j = 0; j < k; j++) {

Z u = a[i + j];

Z v = a[i + j + k] \* roots[k + j];

a[i + j] = u + v;

a[i + j + k] = u - v;

}

}

}

}

void idft(std::vector<Z> &a) {

int n = a.size();

std::reverse(a.begin() + 1, a.end());

dft(a);

Z inv = (1 - P) / n;

for (int i = 0; i < n; i++) {

a[i] \*= inv;

}

}

struct Poly {

std::vector<Z> a;

Poly() {}

Poly(const std::vector<Z> &a) : a(a) {}

Poly(const std::initializer\_list<Z> &a) : a(a) {}

int size() const {

return a.size();

}

void resize(int n) {

a.resize(n);

}

Z operator[](int idx) const {

if (idx < size()) {

return a[idx];

} else {

return 0;

}

}

Z &operator[](int idx) {

return a[idx];

}

Poly mulxk(int k) const {

auto b = a;

b.insert(b.begin(), k, 0);

return Poly(b);

}

Poly modxk(int k) const {

k = std::min(k, size());

return Poly(std::vector<Z>(a.begin(), a.begin() + k));

}

Poly divxk(int k) const {

if (size() <= k) {

return Poly();

}

return Poly(std::vector<Z>(a.begin() + k, a.end()));

}

friend Poly operator+(const Poly &a, const Poly &b) {

std::vector<Z> res(std::max(a.size(), b.size()));

for (int i = 0; i < int(res.size()); i++) {

res[i] = a[i] + b[i];

}

return Poly(res);

}

friend Poly operator-(const Poly &a, const Poly &b) {

std::vector<Z> res(std::max(a.size(), b.size()));

for (int i = 0; i < int(res.size()); i++) {

res[i] = a[i] - b[i];

}

return Poly(res);

}

friend Poly operator-(const Poly &a) {

std::vector<Z> res(a.size());

for (int i = 0; i < int(res.size()); i++) {

res[i] = -a[i];

}

return Poly(res);

}

friend Poly operator\*(Poly a, Poly b) {

if (a.size() == 0 || b.size() == 0) {

return Poly();

}

int sz = 1, tot = a.size() + b.size() - 1;

while (sz < tot) {

sz \*= 2;

}

a.a.resize(sz);

b.a.resize(sz);

dft(a.a);

dft(b.a);

for (int i = 0; i < sz; ++i) {

a.a[i] = a[i] \* b[i];

}

idft(a.a);

a.resize(tot);

return a;

}

friend Poly operator\*(Z a, Poly b) {

for (int i = 0; i < int(b.size()); i++) {

b[i] \*= a;

}

return b;

}

friend Poly operator\*(Poly a, Z b) {

for (int i = 0; i < int(a.size()); i++) {

a[i] \*= b;

}

return a;

}

Poly &operator+=(Poly b) {

return (\*this) = (\*this) + b;

}

Poly &operator-=(Poly b) {

return (\*this) = (\*this) - b;

}

Poly &operator\*=(Poly b) {

return (\*this) = (\*this) \* b;

}

Poly deriv() const {

if (a.empty()) {

return Poly();

}

std::vector<Z> res(size() - 1);

for (int i = 0; i < size() - 1; ++i) {

res[i] = (i + 1) \* a[i + 1];

}

return Poly(res);

}

Poly integr() const {

std::vector<Z> res(size() + 1);

for (int i = 0; i < size(); ++i) {

res[i + 1] = a[i] / (i + 1);

}

return Poly(res);

}

Poly inv(int m) const {

Poly x{a[0].inv()};

int k = 1;

while (k < m) {

k \*= 2;

x = (x \* (Poly{2} - modxk(k) \* x)).modxk(k);

}

return x.modxk(m);

}

Poly log(int m) const {

return (deriv() \* inv(m)).integr().modxk(m);

}

Poly exp(int m) const {

Poly x{1};

int k = 1;

while (k < m) {

k \*= 2;

x = (x \* (Poly{1} - x.log(k) + modxk(k))).modxk(k);

}

return x.modxk(m);

}

Poly pow(int k, int m) const {

int i = 0;

while (i < size() && a[i].val() == 0) {

i++;

}

if (i == size() || 1LL \* i \* k >= m) {

return Poly(std::vector<Z>(m));

}

Z v = a[i];

auto f = divxk(i) \* v.inv();

return (f.log(m - i \* k) \* k).exp(m - i \* k).mulxk(i \* k) \* power(v, k);

}

Poly sqrt(int m) const {

Poly x{1};

int k = 1;

while (k < m) {

k \*= 2;

x = (x + (modxk(k) \* x.inv(k)).modxk(k)) \* ((P + 1) / 2);

}

return x.modxk(m);

}

Poly mulT(Poly b) const {

if (b.size() == 0) {

return Poly();

}

int n = b.size();

std::reverse(b.a.begin(), b.a.end());

return ((\*this) \* b).divxk(n - 1);

}

std::vector<Z> eval(std::vector<Z> x) const {

if (size() == 0) {

return std::vector<Z>(x.size(), 0);

}

const int n = std::max(int(x.size()), size());

std::vector<Poly> q(4 \* n);

std::vector<Z> ans(x.size());

x.resize(n);

std::function<void(int, int, int)> build = [&](int p, int l, int r) {

if (r - l == 1) {

q[p] = Poly{1, -x[l]};

} else {

int m = (l + r) / 2;

build(2 \* p, l, m);

build(2 \* p + 1, m, r);

q[p] = q[2 \* p] \* q[2 \* p + 1];

}

};

build(1, 0, n);

std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly &num) {

if (r - l == 1) {

if (l < int(ans.size())) {

ans[l] = num[0];

}

} else {

int m = (l + r) / 2;

work(2 \* p, l, m, num.mulT(q[2 \* p + 1]).modxk(m - l));

work(2 \* p + 1, m, r, num.mulT(q[2 \* p]).modxk(r - m));

}

};

work(1, 0, n, mulT(q[1].inv(n)));

return ans;

}

};

2.因数分解

using u8 = uint8\_t;

using u32 = uint32\_t;

using u64 = uint64\_t;

using u128 = \_\_uint128\_t;

bool mr32(u32 n, u32 a) {

if (!(a %= n)) return 1;

uint8\_t r = \_\_builtin\_ctz(n - 1); u32 d = n >> r, p = 1;

for (; d; d >>= 1, a = u64(a) \* a % n) {if (d & 1) p = u64(p) \* a % n;}

if (p == 1) return 1;

while (p != n - 1 && --r) {p = u64(p) \* p % n;}

return p == n - 1;

}

bool mr64(u64 n, u64 a) {

if (!(a %= n)) return 1;

u8 r = \_\_builtin\_ctzll(n - 1); u64 d = n >> r, p = 1;

for (; d; d >>= 1, a = u128(a) \* a % n) {if (d & 1) p = u128(p) \* a % n;}

if (p == 1) return 1;

while (p != n - 1 && --r) {p = u128(p) \* p % n;}

return p == n - 1;

}

bool dmr32(u32 n) {

return mr32(n, 2) && mr32(n, 7) && mr32(n, 61);

}

bool dmr64(u64 n) {

if (n <= 0xFFFFFFFF) return dmr32(n);

return mr64(n, 2) && mr64(n, 325) && mr64(n, 9375) && mr64(n, 28178) &&

mr64(n, 450775) && mr64(n, 9780504) && mr64(n, 1795265022);

}

vector<u64> factorise(u64 n) {

// factorises integers 1 ≤ n < 2^62

vector<u64> pf;

for (u8 i : {2, 3, 5}) while (!(n % i)) pf.push\_back(i), n /= i;

for (u32 i = 7; i \* i <= n && i <= 211; i += 6) for (u32 j : {0, 4})

while (n % (i + j) == 0) pf.push\_back(i + j), n /= i + j;

if (n == 1) return pf;

vector<pair<u64, u8>> q{{n, 1}};

while (!q.empty()) {

auto n = q.back().first;

auto c = q.back().second;

q.pop\_back();

if (n < 49729) {

while (c--) pf.push\_back(n);

continue;

}

u64 r = sqrt(n) + 0.5;

if (r \* r == n) {

q.emplace\_back(r, c \* 2);

continue;

}

if (dmr64(n)) {

while (c--) pf.push\_back(n);

continue;

}

u64 ni = -n; for (u8 \_ = 5; \_--;) ni \*= 2 + ni \* n;

auto redc = [&](u128 x) -> u64 {

return u64(x) \* ni \* u128(n) + x >> 64;

};

auto f = [&](u64 x) {return redc(u128(x) \* x + 1);};

u64 g = n;

const u64 m = 1LL << (\_\_lg(n) / 5);

for (u64 x0 = 0; g == n; ++x0) {

u64 x, y = x0, q = 1, ys; g = 1;

for (u64 r = 1; g == 1; r <<= 1) {

x = y;

for (u64 \_ = r; \_--;) y = f(y);

for (u64 k = 0; k < r && g == 1; k += m) {

ys = y;

for (u64 \_ = min(m, r - k); \_--;)

y = f(y), q = redc(u128(max(x, y) - min(x, y)) \* q);

g = \_\_gcd(q, n);

}

}

if (g == n) {

for (; g == 1;) {

ys = f(ys);

g = \_\_gcd(max(x, ys) - min(x, ys), n);

}

}

}

q.emplace\_back(g, c); q.emplace\_back(n / g, c);

}

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

return pf;

}

3. 线性筛

const int M = (int) 1e6 + 8;

bool vis[M];

vector<int> p;

void Eulersiege(){

for(int i = 2; i <= M - 5; i++){

if(vis[i] == 0){

p.push\_back(i);

}

for(int j = 0; j < p.size(); j++){

if(i \* p[j] > M - 5){

break;

}

vis[i \* p[j]] = 1;

if(i % p[j] == 0){

break;

}

}

}

}

五、杂项

1.快读

struct IO {

#define MAXSIZE (1 << 20)

#define isdigit(x) (x >= '0' && x <= '9')

char buf[MAXSIZE], \*p1, \*p2;

char pbuf[MAXSIZE], \*pp;

#if DEBUG

#else

IO() : p1(buf), p2(buf), pp(pbuf) {}

~IO() { fwrite(pbuf, 1, pp - pbuf, stdout); }

#endif

inline char gc() {

#if DEBUG // 调试，可显示字符

return getchar();

#endif

if (p1 == p2) p2 = (p1 = buf) + fread(buf, 1, MAXSIZE, stdin);

return p1 == p2 ? ' ' : \*p1++;

}

inline bool blank(char ch) {

return ch == ' ' || ch == '\n' || ch == '\r' || ch == '\t';

}

template <class T>

inline void read(T &x) {

register double tmp = 1;

register bool sign = 0;

x = 0;

register char ch = gc();

for (; !isdigit(ch); ch = gc())

if (ch == '-') sign = 1;

for (; isdigit(ch); ch = gc()) x = x \* 10 + (ch - '0');

if (ch == '.')

for (ch = gc(); isdigit(ch); ch = gc())

tmp /= 10.0, x += tmp \* (ch - '0');

if (sign) x = -x;

}

inline void read(char \*s) {

register char ch = gc();

for (; blank(ch); ch = gc())

;

for (; !blank(ch); ch = gc()) \*s++ = ch;

\*s = 0;

}

inline void read(char &c) {

for (c = gc(); blank(c); c = gc())

;

}

inline void push(const char &c) {

#if DEBUG // 调试，可显示字符

putchar(c);

#else

if (pp - pbuf == MAXSIZE) fwrite(pbuf, 1, MAXSIZE, stdout), pp = pbuf;

\*pp++ = c;

#endif

}

template <class T>

inline void write(T x) {

if (x < 0) x = -x, push('-'); // 负数输出

static T sta[35];

T top = 0;

do {

sta[top++] = x % 10, x /= 10;

} while (x);

while (top) push(sta[--top] + '0');

}

template <class T>

inline void write(T x, char lastChar) {

write(x), push(lastChar);

}

} io;

2.二分

while(l < r) {

int mid = l + r >> 1;

if(check(mid)) {

r = mid;

}

else {

l = mid + 1;

}

}

ret = l;

// find the minimum feasible solution

while(l <= r) {

int mid = l + r >> 1;

if(check(mid)) {

l = mid + 1;

}

else {

r = mid - 1;

}

}

ret = r;

//find the maximum feasible solution

3.莫队

int up(int a, int b) {

return (a + b - 1) / b;

}

struct Q {

int l, r, id;

};

int a[M], belong[M], cnt[M];

void solve() {

int n;

cin >> n;

for(int i = 0; i < n; i++) {

cin >> a[i];

}

int siz = (int)sqrt(n);

for(int i = 0; i < n; i++) {

belong[i] = i / siz;

}

auto cmp = [&](Q a, Q b) {

return (belong[a.l] ^ belong[b.l]) ? belong[a.l] < belong[b.l] : ((belong[a.l] & 1) ? a.r < b.r : a.r > b.r);

};

int m;

cin >> m;

vector<Q> q(m);

for(int i = 0; i < m; i++) {

cin >> q[i].l >> q[i].r;

--q[i].l, --q[i].r;

q[i].id = i;

}

sort(all(q), cmp);

int now = 0;

int l = 1, r = 0;

auto add = [&](int pos) {

now += (cnt[a[pos]] == 0);

cnt[a[pos]]++;

};

auto del = [&](int pos) {

cnt[a[pos]]--;

now -= (cnt[a[pos]] == 0);

};

vi ans(m);

for(int i = 0; i < m; i++) {

while(l < q[i].l) del(l++);

while(l > q[i].l) add(--l);

while(r < q[i].r) add(++r);

while(r > q[i].r) del(r--);

ans[q[i].id] = now;

}

for(int i = 0; i < m; i++) {

cout << ans[i] << '\n';

}

}

4.随机+计时

mt19937\_64 rg(std::chrono::steady\_clock::now().time\_since\_epoch().count());

int rng(int l, int r){

return l + rg() % (r - l + 1);

}

clock\_t startTime;

double getCurrentTime() {

return (double)(clock() - startTime) / CLOCKS\_PER\_SEC;

}

5.分数计算

struct Fraction{

int up,down;

};

int gcb(int a,int b)

{

if(b==0)

return a;

else

return gcb(b,a%b);

}

Fraction reduction(Fraction &result)

{

if(result.down < 0)

{

result.down = -result.down;

result.up = - result.up;

}

else if(result.up == 0)

result.down = 1;

else

{

int x = gcb(abs(result.up),abs(result.down));

result.up /= x;

result.down /= x;

}

return result;

}

Fraction Add(Fraction a,Fraction b)

{

Fraction c;

c.up = a.up \* b.down + a.down \* b.up;

c.down = a.down \* b.down;

return reduction(c);

}

Fraction minu(Fraction a,Fraction b)

{

Fraction c;

c.up = a.up \* b.down - a.down \* b.up;

c.down = a.down \* b.down;

return reduction(c);

}

Fraction multi(Fraction a,Fraction b)

{

Fraction c;

c.up = a.up \* b.up;

c.down = a.down \* b.down;

return reduction(c);

}

Fraction divide(Fraction a,Fraction b)

{

Fraction c;

c.up = a.up \* b.down;

c.down = a.down \* b.up;

return reduction(c);

}

string to\_string(Fraction x) {

return to\_string(to\_string(x.up) + "/" + to\_string(x.down) );

}

bool cmp(Fraction a, Fraction b) {

return a.up \* b.down > b.up \* a.down; //>

}

Fraction to(int x) = {

return (Fraction){x, 1};

}

6. 对拍

#include <cstdlib>

#include <cstdio>

int main() {

int cnt = 0;

system("g++ data.cpp -o data");

system("g++ baoli.cpp -o baoli");

system("g++ std.cpp -o std");

while (true) {

system("data.exe");

system("std.exe");

system("baoli.exe");

if (system("fc out.txt baoli.txt")) break;

cnt++;

printf("%d cases AC\n", cnt);

}

system("pause");

return 0;

}