Team notebook

December 1, 2023

Contents

1 Bignumremaster 1

2 Bridge+Articuno 2

3 ConvexHullTrick 2

4 DSUrollback 2

5 DinicLowerbound 3

6 Euclid$_Diophante$ 4

7 EulerCircle 4

8 Geometry 4

9 HLD 6

10 PersitentSegtri 7

11 Z$_algo$ 7

12 ahocorasick 7

13 bit2d 8

14 centroid 9

15 dinic 9

16 edmond$_karp$ 10

17 fft 11

18 gauss 12

19 hash$_NewSA_wip$ 12

20 kmp 13

21 lagrange$_interpolation$ 14

22 max$_match$ 14

23 mincostflow 15

24 pst$_lazy$ 15

25 pst$_new$ 16

26 rabin$_millerr$ 17

27 segtri2D 17

28 suffix$_array$ 18

29 tarjanSCC 20

1 Bignumremaster

```
const int BASE = 1e7, LBASE = 7;
void fix(big& a) {
	a.push_back(0);
	for (int i = 0; i < a.size() - 1; i++) {
			a[i + 1] += a[i] / BASE;
			a[i] %= BASE;
			if (a[i] < 0) a[i] += BASE, a[i + 1]--;
		}
		while (a.size() >= 2 && a.back() == 0) a.pop_back();
}
big operator + (big a, const big& b) {
		a.resize(max(a.size(), b.size()));
		for (int i = 0; i < b.size(); i++) a[i] += b[i];
		fix(a); return a;
}
big operator - (big a, const big& b) {
		for (int i = 0; i < b.size(); i++) a[i] -= b[i];
		fix(a); return a;
}
big operator * (const big& a, const big& b) {
		big c(a.size() + b.size());
		for (int i = 0; i < a.size(); i++)
		for (int j = 0; j < b.size(); j++) c[i + j] += a[i] * b[j];
		fix(c); return c;
}
big operator / (const big& a, int x) {
		big res;
		int r = 0;
		for (int i = a.size() - 1; i >= 0; i--) {
			r = r * BASE + a[i];
			res.push_back(r / x);
			r %= x;
		}
		reverse(res.begin(), res.end());
		fix(res); return res;
}
int operator % (const big& a, int x) {
		int r = 0;
		for (int i = a.size() - 1; i >= 0; i--) {
			r = r * BASE + a[i];
			r %= x;
		}
		return r;
}
bool operator < (const big& a, const big& b) {
		if (a.size() != b.size()) return a.size() < b.size();
		for (int i = a.size() - 1; i >= 0; i--)
		if (a[i] != b[i]) return a[i] < b[i];
		return 0;
```

```
}
big stringToBig(const string& s) {
		big a(s.size() / LBASE + 1);
		for (int i = 0; i < s.size(); i++) {
			int x = (s.size() - i - 1) / LBASE;
			a[x] = a[x] * 10 + s[i] - '0';
		}
		fix(a); return a;
}
void Print(const big& a) {
		cout << a.back();
		for (int i = (int) a.size() - 2; i >= 0; i--)
			cout << setw(LBASE) << setfill('0') << a[i];
}
```

2 Bridge+Articuno

```
void dfs(int u, int p) {
		tin[u] = low[u] = ++now;
		if (u != p) parts[u]++;
		for (int v : a[u]) {
			if (v == p) continue;
			if (tin[v]) low[u] = min(low[u], tin[v]);
			else {
				dfs(v, u);
				low[u] = min(low[u], low[v]);
				if (low[v] > tin[u]) bridge++;
				if (low[v] >= tin[u]) parts[u]++; // part >= 2 ->
				    articuno++
			}
		}
}
void dfs(int u,int p){
		check[u]=root.size();
		if(u!=p)h[u]=h[p]+1;
		sz[u]=1;
		for(int v:a[u]){
			if(v==p)continue;
			if(!check[v]){
				dfs(v,u);
				sz[u]+=sz[v];
				dp[u]+=dp[v];
```

```
            }
            else{
                if(h[u]<h[v])dp[u]--;
                else dp[u]++;
            }
        }
        if(u!=p&&dp[u]==0)bridge.push_back(u);
}
```

3 ConvexHullTrick

```
struct Line {
        mutable int k, m, p;
        bool operator < (const Line& oth) const {
                if (oth.k == INF && oth.m == INF) return p < oth.p;
                return k < oth.k;
        }
};

struct ConvexHull : multiset<Line> {
        int divi(int x, int y) {
                return x / y - ((x ^ y) < 0 && x % y);
        }
        bool isect(iterator x, iterator y) {
                if (y == end()) return x->p = INF, 0;
                if (x->k == y->k) x->p = x->m > y->m ? INF : -INF;
                else x->p = divi(y->m - x->m, x->k - y->k);
                return x->p >= y->p;
        }
        void add(int k, int m) {
                auto z = insert({k, m, 0}), y = z++, x = y;
                while (isect(y, z)) z = erase(z);
                if (x != begin() && isect(--x, y)) isect(x, erase(y));
                while ((y = x) != begin() && (--x)->p >= y->p)
                        isect(x, erase(y));
        }
        int query(int x) {
                assert(!empty());
                auto l = *lower_bound({INF, INF, x});
                return l.k * x + l.m;
        }
};
```

4 DSUrollback

```
int par[maxN];
int up[maxN];
int dist[maxN];
int cnt;
vector<pair<int*, int>> changes;
void rickroll(int id) {
        while (id < changes.size()) {
                *changes.back().first = changes.back().second;
                changes.pop_back();
        }
}
int Find(int u) {
        int res = par[u] >= 0 ? Find(par[u]) : u;
        if (res == u) up[u] = dist[u] = 0;
        else dist[u] = up[u] ^ dist[par[u]];
        return res;
}
void Unite(int u, int v) {
        int x = Find(u), y = Find(v);
        if (x == y) {
                if (dist[u] == dist[v]) {
                        changes.push_back({&cnt, cnt});
                        cnt++;
                }
                return;
        }
        if (par[x] > par[y]) swap(x, y);
        changes.push_back({&par[x], par[x]});
        changes.push_back({&par[y], par[y]});
        par[x] += par[y];
        par[y] = x;
        up[y] = dist[u] ^ dist[v] ^ 1;
}
```

5 DinicLowerbound

```
struct Dinic_lowerbound {
        struct edge {
                int v, rev;
```

```
		int cap, flow;
		int dem;
	};
	int S, T;
	vector<int> depth, ptr;
	vector<vector<edge>> adj;
	Dinic_lowerbound(int n) : S(n + 1), T(n + 2), depth(n + 5), ptr(n
	    + 5), adj(n + 5) {}
	Dinic_lowerbound() {}
	void addEdge(int u, int v, int cap, int dem) {
		adj[u].push_back({v, (int)adj[v].size(), cap, 0, dem});
		adj[v].push_back({u, (int)adj[u].size() - 1, 0, 0, 0});
	}
	bool bfs() {
		fill(depth.begin(), depth.end(), 0);
		queue<int> q;
		depth[S] = 1;
		q.push(S);
		while (!q.empty() && !depth[T]) {
			int u = q.front(); q.pop();
			for (edge& e : adj[u]) {
				if (depth[e.v] || e.cap - e.flow < 1)
				    continue;
				depth[e.v] = depth[u] + 1;
				q.push(e.v);
			}
		}
		return depth[T];
	}
	int dfs(int u, int flowIn) {
		if (u == T) return flowIn;
		int flowOut = 0;
		for (; ptr[u] < adj[u].size(); ptr[u]++) {
			edge& e = adj[u][ptr[u]];
			if (depth[u] + 1 != depth[e.v] || e.cap - e.flow ==
			    0) continue;
			int q = dfs(e.v, min(flowIn, e.cap - e.flow));
			flowOut += q;
			flowIn -= q;
			e.flow += q;
			adj[e.v][e.rev].flow -= q;
			if (flowIn == 0) break;
		}
		return flowOut;
	}
	int calc() {
		int maxFlow = 0;
		while (bfs()) {
			fill(ptr.begin(), ptr.end(), 0);
			maxFlow += dfs(S, INF);
		}
		return maxFlow;
	}
	int calc_with_lowerbound() {
		int n = S - 1;
		int s = n + 3;
		int t = n + 4;
		int sumDem = 0;
		vector<int> in(n + 3), out(n + 3);
		for (int u = 1; u <= n + 2; u++)
		for (edge& e : adj[u]) {
			sumDem += e.dem;
			out[u] += e.dem;
			in[e.v] += e.dem;
			e.cap -= e.dem;
		}
		for (int u = 1; u <= n; u++) {
			addEdge(s, u, in[u], 0);
			addEdge(u, t, out[u], 0);
		}
		addEdge(T, S, INF, 0);

		swap(s, S); swap(t, T);
		if (calc() != sumDem) return -1;
		swap(s, S); swap(t, T);

		for (int u = 1; u <= n + 2; u++)
		for (edge& e : adj[u]) {
			e.cap += e.dem;
			e.flow += e.dem;
		}
		for (int u = 1; u <= n; u++) {
			adj[u].pop_back();
			adj[u].pop_back();
		}
		adj[S].pop_back(); adj[T].pop_back();

		return calc() + sumDem;
	}
};
```

6 Euclid$_{Diophante}$

```
struct vl{
        long x,y;
};
long t;
long a,b,c,d;
long gcd(long a,long b){
        return b?gcd(b,a%b):a;
}
vl extended_euclid(long a,long b){
        long m=a,xm=1;
        long n=b,xn=0;
        while(n){
                long q=m/n;
                long r=m-q*n,xr=xm-q*xn;
                m=n,xm=xn;
                n=r,xn=xr;
        }
        // if m < 0 ?? (d = -1)
        return {xm,(m-a*xm)/b};
}
```

7 EulerCircle

```
void dfs(int u) {
         // all deg even, ans size = E + 1
    while (!a[u].empty()) {
        edge& p = e[a[u].back()];
        a[u].pop_back();
        if (p.used) continue;
        p.used = 1;
        int v = p.u ^ p.v ^ u;
        dfs(v);
        ans.push_back(v);
    }
}
```

8 Geometry

```
const double EPS = 1e-7;
const double DFT = 6969.69;
bool equal(const double& a, const double& b) {
        return abs(a - b) < EPS;
}
struct point {
        int x, y;
        point operator + (const point& oth) const {
                return {x + oth.x, y + oth.y};
        }
        point operator - (const point& oth) const {
                return {x - oth.x, y - oth.y};
        }
        // Dot product
        int operator % (const point& oth) const {
                return x * oth.x + y * oth.y;
        }
        // Cross product
        int operator * (const point& oth) const {
                // Equal to 2 * area(aOb)
                // Postive if slope(a) < slope(b)
                // Negative if slope(a) > slope(b)
                // |x1 y1|
                // |x2 y2|
                return x * oth.y - oth.x * y;
        }
        int sqrlen() const {
                return x * x + y * y;
        }
        double len() const {
                return sqrt(sqrlen());
        }
};
using Polygon = vector<point>;
point vect(const point& a, const point& b) {
        return b - a;
}
double turn(const point& a, const point& b) {
        // angle aOb in radian
        return atan2(a * b, a % b);
}
bool collinear(const point& a, const point& b, const point& c) {
        return vect(a, b) * vect(b, c) == 0;
}
bool between(const point& a, const point& b, const point& c) {
        return collinear(a, b, c) && vect(a, b) % vect(b, c) >= 0;
}
// a -> b -> c = left turn?
bool leftTurn(const point& a, const point& b, const point& c) {
        return vect(a, b) * vect(b, c) > 0;
}
int polygonArea2(const vector<point>& p) {
        // Two times the area of a polygon with integral vertices
        // Positive if counterclockwise
        // Negative if clockwise
        int area2 = 0;
        for(int i = 0; i < p.size(); i++) {
                int j = (i + 1) % p.size();
                area2 += p[i] * p[j];
        }
        return area2;
}
bool inside_polygon(const point& a, const Polygon& p) {
        double angle = 0;
        for (int i = 0; i < p.size(); i++) {
                int j = (i + 1) % p.size();
                if (between(p[i], a, p[j])) return 1;
                angle += turn(vect(a, p[i]), vect(a, p[j]));
        }
        return equal(abs(angle), 2 * PI);
}
struct segment {
        point a, b;
};
struct line {
        int a, b, c;
        line(const segment& l) {
                point n = vect(l.a, l.b);
                n = {n.y, -n.x};
                a = n.x;
                b = n.y;
                c = -(a * l.a.x + b * l.a.y);
        }
};
point line_intersection(const line& l1, const line& l2) {
        // |a1 b1|
        // |a2 b2|
        double d = l1.a * l2.b - l2.a * l1.b;
        // |b1 c1|
        // |b2 c2|
        double dx = l1.b * l2.c - l2.b * l1.c;
        // |c1 a1|
        // |c2 a2|
        double dy = l1.c * l2.a - l2.c * l1.a;
        if (equal(d, 0)) {
                // Coincide
                if (equal(dx + dy, 0)) return {DFT, 1};
                // Nope
                return {DFT, 0};
        }
        return {dx / d, dy / d};
}
point intersection(const segment& l1, const segment& l2) {
        point p = line_intersection(line(l1), line(l2));
        if (p.x == DFT || !between(l1.a, p, l1.b) || !between(l2.a, p,
            l2.b))
                return {DFT};
        return p;
}
void convex_hull {
        cin >> n; point p0 = {INF, INF};
        for (int i = 0; i < n; i++) {
                cin >> p[i].x >> p[i].y;
                if (p[i].y < p0.y || (p[i].y == p0.y && p[i].x < p0.x))
                        id = i, p0 = p[i];
        }
        // Lowest + Leftmost point
        swap(p[id], p[0]);
        // Graham Scan
        sort(p + 1, p + n, [](const point& a, const point& b) {
                // Sort by slope
                // Same slope -> sort by len
                point u = a - p0;
                point v = b - p0;
                int t = u * v;
                return t > 0 || (t == 0 && u.sqrlen() < v.sqrlen());
        });
        // Find convex hull
        vector<point> q;
        for (int i = 0; i < n; i++) {
                // If right turn -> goodbye fellow
            while (q.size() >= 2 && !leftTurn(q[q.size() - 2],
                q[q.size() - 1], p[i]))
                   q.pop_back();
            q.push_back(p[i]);
        }
}
```

9 HLD

```
void pre_dfs(int u, int p) {
        par[u] = p;
        if (u != p) depth[u] = depth[p] + 1;
        sz[u] = 1;
        for (int& i : adj[u]) {
                int v = e[i].u ^ e[i].v ^ u;
                if (v == p) continue;
                pre_dfs(v, u);
                sz[u] += sz[v];
        }
}
void hld(int u, int p, int is_heavy, int lastEdge) {
        if (!is_heavy) head[u] = u;
        else head[u] = head[p];
        treePos[u] = ++ti;
        edgePos[lastEdge] = ti;
        int maxsz = 0, heavy = -1, heavyEdge;
        for (int& i : adj[u]) {
                int v = e[i].u ^ e[i].v ^ u;
                if (v == p || sz[v] <= maxsz) continue;
                maxsz = sz[v], heavy = v, heavyEdge = i;
        }
        if (heavy != -1) hld(heavy, u, 1, heavyEdge);
        for (int& i : adj[u]) {
                int v = e[i].u ^ e[i].v ^ u;
                if (v == p || v == heavy) continue;
                hld(v, u, 0, i);
        }
}
int query(int u, int v) {
        int res = 0;
        for (; head[u] != head[v]; v = par[head[v]]) {
                if (depth[head[u]] > depth[head[v]]) swap(u, v);
                res = max(res, st.get(treePos[head[v]], treePos[v]));
        }
        if (depth[u] > depth[v]) swap(u, v);
        res = max(res, st.get(treePos[u] + 1, treePos[v]));
        return res;
}
```

10 PersitentSegtri

```
struct segment {
        int l, r, p;
        bool operator < (const segment& oth) const {
                return l < oth.l;
        }
};
int n, m, k;
vector<segment> vec;
struct PersistentSegmentTree {
        static const int maxNodes = maxN * 19;
        struct Node {
                int v, lc, rc;
        };
        int n;
        vector<Node> st;
        vector<int> ver;
        PersistentSegmentTree(int n) {
                this->n = n;
                st.reserve(maxNodes);
        }
        PersistentSegmentTree() {}
        int create(Node a) {
                st.push_back(a);
                return st.size() - 1;
        }
        int build(int l, int r) {
                if (l == r) {
                        return create({INF});
                }
                int mid = (l + r) / 2;
                int id = create({INF, build(l, mid), build(mid + 1, r)});
                return id;
        }
```

```
	int update(int i, int v, int oldId, int l, int r) {
		if (l == r) {
			return create({min(v, st[oldId].v)});
		}
		int mid = (l + r) / 2;
		int id = create(st[oldId]);
		if (i <= mid)
			st[id].lc = update(i, v, st[oldId].lc, l, mid);
		else
			st[id].rc = update(i, v, st[oldId].rc, mid + 1, r);
		st[id].v = max(st[st[id].lc].v, st[st[id].rc].v);
		return id;
	}
	int get(int L, int R, int id, int l, int r) {
		if (R < l || r < L) return -1;
		if (L <= l && r <= R) return st[id].v;
		int mid = (l + r) / 2;
		int lv = get(L, R, st[id].lc, l, mid);
		int rv = get(L, R, st[id].rc, mid + 1, r);
		return max(lv, rv);
	}
	void build() {
		ver.push_back(0);
		build(1, n);
	}
	void update(int i, int v) {
		ver.push_back(update(i, v, ver.back(), 1, n));
	}
	int get(int L, int R, int k) {
		return get(L, R, ver[k], 1, n);
	}
} pst;
```

11 Z*algo*

```
void Z() {
	s = ' ' + s;
	for (int i = 2, l, r = -1; i <= n; i++) {
		if (i <= r) z[i] = min(z[i - l + 1], r - i + 1);
		while (i + z[i] <= n && s[1 + z[i]] == s[i + z[i]]) z[i]++;
		if (r < i + z[i] - 1) {
			l = i;
			r = i + z[i] - 1;
		}
	}
}
```

12 ahocorasick

```
struct Trie {
	struct Node {
		int child[C];
		int par;
		int val;
		int link = 0;
		int nex[C];
		int ends = 0;
		int cnt = 0;
		Node(int par = -1, int val = -1) : par(par), val(val) {
			fill(child, child + C, 0);
			fill(nex, nex + C, 0);
		}
	};
	vector<Node> trie;
	Trie() {
		trie.reserve(maxLen);
		trie.push_back(Node());
	}
	void add(const string& s) {
		int v = 0;
		for (char c : s) {
			c -= 'A';
			if (!trie[v].child[c]) {
				trie[v].child[c] = trie.size();
				trie.push_back(Node(v, c));
			}
			v = trie[v].child[c];
		}
		trie[v].ends++;
	}
	void Ahchoo() {
		queue<int> q;
		q.push(0);
		while (!q.empty()) {
```

```
                    int u = q.front(); q.pop();
                    int p = trie[u].link;
                    trie[u].cnt = trie[p].cnt + trie[u].ends;
                    for (int c = 0; c < C; c++) {
                            if (trie[u].child[c]) {
                                    trie[trie[u].child[c]].link =
                                        trie[p].nex[c];
                trie[u].nex[c] = trie[u].child[c];
                                    q.push(trie[u].child[c]);
                            }
                            else {
                                    trie[u].nex[c] = trie[p].nex[c];
                            }
                    }
            }
    }
} trie;
```

13 bit2d

```
int BIT[N][M];
void add(int u, int v, int x){
    for(int i = u; i <= n; i += i&(-i)){
        for(int j = v; j <= m; j += j&(-j))BIT[i][j]+=x;
    }
}

int query(int u, int v){
    int sum = 0;
    for(int i = u; i > 0; i -= i&(-i)){
        for(int j = v; j > 0; j -= j&(-j))sum += BIT[i][j];
    }
    return sum;
}
void rectAdd(int a, int b, int u, int v, int x){
    add(a, b, x);
    add(u+1, v+1, x);
    add(u+1, b, -x);
    add(a, v+1, -x);
}
int BIT[4][N][M]; // {D[i][j]; i*D[i][j]; j*D[i][j]; i*j*D[i][j]}
void add(int u, int v, int x){
    for(int i = u; i <= n; i += i&(-i)){
        for(int j = v; j <= m; j += j&(-j)){
            BIT[0][i][j] += x;
            BIT[1][i][j] += u * x;
            BIT[2][i][j] += v * x;
            BIT[3][i][j] += u * v * x;
        }
    }
}

void rectAdd(int a, int b, int u, int v, int x){
    add(a, b, x);
    add(a, v + 1, -x);
    add(u + 1, b, -x);
    add(u + 1, v + 1, x);
}
// ben bit 2d
vector<int> pos[N];
vector<int> BIT[N];

void fakeAdd(int u, int v, int x){
    for(u; u <= n; u += u&(-u)){
        pos[u].push_back(v);
    }
}

void fakeQuery(int u, int v){
    for(u; u <= n; u += u&(-u)){
        pos[u].push_back(v);
    }
}
void compress(){
    for(int i = 1; i <= n; i++){
        pos[i].push_back(0);
        sort(pos[i].begin(), pos[i].end());
        pos[i].erase(unique(pos[i].begin(), pos[i].end()), pos[i].end());
        BIT[i].assign(pos[i].size(), 0);
    }
}
void add(int u, int v, int x){
    for(int i = u; i <= n; i += i&(-i)){
        for(int j = lower_bound(pos[i].begin(), pos[i].end(), v) -
            pos[i].begin(); j < BIT[i].size(); j += j&(-j)){
            BIT[i][j] += x;
        }
    }
}

void query(int u, int v){
    int sum = 0;
    for(int i = u; i > 0; i -= i&(-i)){
        for(int j = lower_bound(pos[i].begin(), pos[i].end(), v) -
            pos[i].begin(); j > 0; j -= j&(-j)){
            sum += BIT[i][j];
        }
    }
    return sum;
}
```

14 centroid

```
void pre_dfs(int u, int p) {
        sz[u] = 1;
        for (int& v : adj[u]) {
                if (v == p || ban[v]) continue;
                pre_dfs(v, u);
                sz[u] += sz[v];
        }
}
int centos(int u, int p, int root) {
        for (int& v : adj[u]) {
                if (v == p || ban[v]) continue;
                if (sz[v] * 2 > sz[root])
                        return centos(v, u, root);
        }
        return u;
}
void decompose(int u) {
        pre_dfs(u, u);
        ban[u = centos(u, u, u)] = 1;
        for (int& v : adj[u])
        if (!ban[v]) decompose(v);
}
```

15 dinic

```
struct Dinic {
        static const bool SCALING = false;
        int lim;
        struct edge {
                int v, rev;
                int cap, flow;
        };
        int S, T;
        vector<int> depth, ptr;
        vector<vector<edge>> adj;
        Dinic(int n) : S(n + 1), T(n + 2), depth(n + 3), ptr(n + 3), adj(n
            + 3) {}
        Dinic() {}
        void addEdge(int u, int v, int cap, int undirected = 0) {
                adj[u].push_back({v, (int)adj[v].size(), cap, 0});
                adj[v].push_back({u, (int)adj[u].size() - 1, cap *
                    undirected, 0});
        }
        bool bfs() {
                fill(depth.begin(), depth.end(), 0);
                queue<int> q;
                depth[S] = 1;
                q.push(S);
                while (!q.empty() && !depth[T]) {
                        int u = q.front(); q.pop();
                        for (edge& e : adj[u]) {
                                if (depth[e.v] || e.cap - e.flow < lim)
                                    continue;
                                depth[e.v] = depth[u] + 1;
                                q.push(e.v);
                        }
                }
                return depth[T];
        }
        int dfs(int u, int flowIn) {
                if (u == T) return flowIn;
                int flowOut = 0;
                for (; ptr[u] < adj[u].size(); ptr[u]++) {
                        edge& e = adj[u][ptr[u]];
                        if (depth[u] + 1 != depth[e.v] || e.cap - e.flow ==
                            0) continue;
                        int q = dfs(e.v, min(flowIn, e.cap - e.flow));
                flowIn -= q;
                flowOut += q;
                e.flow += q;
                adj[e.v][e.rev].flow -= q;
                if (flowIn == 0) break;
            }
            return flowOut;
        }
        int calc() {
            int maxFlow = 0;
            for (lim = SCALING ? (1 << 12) : 1; lim; lim >>= 3) {
                while (bfs()) {
                    fill(ptr.begin(), ptr.end(), 0);
                    maxFlow += dfs(S, INF);
                }
            }
            return maxFlow;
        }
};
```

16 edmond$_k arp$

```
void add_edge(int u, int v, int c) {
        a[u].push_back(v);
        a[v].push_back(u);
        cap[u][v] = c;
}
void inc_flow(int u, int v, int f) {
        flow[u][v] += f;
        flow[v][u] -= f;
        if (u == s) max_flow += f;
}
int residual(int u, int v) {
        return cap[u][v] - flow[u][v];
}
void bfs(int s, int t) {
        fill(par + 1, par + 1 + n, 0);
        queue<int> q;
        par[s] = s;
        q.push(s);
        while (!q.empty() && par[t] == 0) {
                int u = q.front(); q.pop();
                for (int& v : a[u]) {
                        if (par[v] != 0 || residual(u, v) == 0) continue;
                        par[v] = u;
                        q.push(v);
                }
        }
}
bool find_augmenting_path() {
        bfs(s, t);
        return par[t] != 0;
}
void augment() {
        int minn = INF;
        for (int v = t; v != s; v = par[v])
                minimize(minn, residual(par[v], v));

        for (int v = t; v != s; v = par[v])
                inc_flow(par[v], v, minn);
}
signed main() {
        while (find_augmenting_path())
                augment();
        cout << max_flow;
}
```

17 fft

```
using namespace std;

#define pb push_back
#define mp make_pair
#define sz(a) (int)(a).size()
#define all(a) (a).begin(), (a).end()

#define forn(i,n) for (int i=0; i<int(n); ++i)
#define fornd(i,n) for (int i=int(n)-1; i>=0; --i)
#define xrange(i,a,b) for (int i=int(a); i<int(b); ++i)

typedef long long ll;
typedef long double ld;
typedef unsigned long long ull;
```

```
const int INF = (int) 1e9;
const long long INF64 = (long long) 1e18;
const long double eps = 1e-9;
const long double pi = 3.14159265358979323846;

const int mod = 7340033;
const ll root = 5;
const ll root_1 = 4404020;
const ll root_pw = 1<<20;

int rev_element[7340033];

ll getmod(ll a, ll tmod){
    return ((a%tmod)+tmod)%tmod;
}

void fft (vector<ll> & a, bool invert) {
    int n = (int) a.size();

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

    for (int len=2; len<=n; len<<=1) {
        ll wlen = invert ? root_1 : root;
        for (int i=len; i<root_pw; i<<=1)
            wlen = ll (wlen * 1ll * wlen % mod);
        for (int i=0; i<n; i+=len) {
            ll w = 1;
            for (int j=0; j<len/2; ++j) {
                ll u = a[i+j], v = ll (a[i+j+len/2] * 1ll * w % mod);
                a[i+j] = getmod(u+v,mod);
                a[i+j+len/2] = getmod(u-v,mod);
                w = ll (w * 1ll * wlen % mod);
            }
        }
    }
    if (invert) {
        ll nrev = rev_element[n];
            for (int i=0; i<n; ++i)
                a[i] = int (a[i] * 1ll * nrev % mod);
    }
}

void precalc(){
    rev_element[1] = 1;
    for (int i=2; i<mod; i++)
        rev_element[i] = (mod - (mod/i) * rev_element[mod%i] % mod) % mod;
}


void multiply (const vector<ll> & a, const vector<ll> & b, vector<ll> &
     res) {
    vector <ll> fa (a.begin(), a.end()), fb (b.begin(), b.end());
    size_t n = 1;
    while (n < max (a.size(), b.size())) n <<= 1;
    n <<= 1;
    fa.resize (n), fb.resize (n);
    fft (fa, false), fft (fb, false);
    forn(i,n)
        fa[i] *= fb[i];
    fft (fa, true);
    res.resize (n);
    forn(i,n)
        res[i] = fa[i] % mod;
}

int MN = 29;
int MK = 1001;
vector <vector <ll> > dp(MN+2,vector <ll> (MK,0));
vector <ll> d;

void init(){
    int m;
    for (int i=0; i<=MN; i++){
        dp[i][0] = 1;
        multiply(dp[i],dp[i],d);
        d.resize(MK);
        multiply(d,d,dp[i+1]);
        dp[i+1].insert(dp[i+1].begin(),0);
        dp[i+1].resize(MK);
    }
}
```

18 gauss

```
void Gauss(int n) {
	for(int i = 0; i < n; i++) {
		for(int j = i; j < n; j++) {
			if(P[j][i] != 0) {
				for(int k = 0; k <= n; k++) {
					swap(P[j][k], P[i][k]);
				}
				break;
			}
		}

		if(P[i][i] == 0) continue;
		for(int j = 0; j < n; j++) {
			if(i == j) continue;
			num coef = P[j][i] / P[i][i];
			for(int k = 0; k <= n; k++) P[j][k] -= P[i][k] *
			    coef;
		}
	}

	for(int i = 0; i < n; i++) C[i] = P[i][n] / P[i][i];
	degree = n - 1;
}
```

19 hash$_NewSA_wip$

```
const int K = 269;
const int MOD1 = 1e9 + 7, MOD2 = 998244353;
int pw1[maxLen], pw2[maxLen];
void init_pw() {
	for (int i = 0; i < maxLen; i++) {
		pw1[i] = i ? pw1[i - 1] * K % MOD1 : 1;
		pw2[i] = i ? pw2[i - 1] * K % MOD2 : 1;
	}
}
struct Hash {
	int x, y;
	bool operator == (const Hash& oth) const {
		return x == oth.x && y == oth.y;
	}
};
struct Roll : vector<Hash> {
	string s;
	Roll(const string& s) : s(s) {
		assign(s.size(), {0, 0});
		for (int i = 1; i < s.size(); i++) {
			at(i).x = (at(i - 1).x * K + s[i]) % MOD1;
			at(i).y = (at(i - 1).y * K + s[i]) % MOD2;
		}
	}
	Roll() {}
	Hash get(int l, int r) {
		Hash res;
		res.x = (at(r).x - at(l - 1).x * pw1[r - l + 1] + MOD1 *
		    MOD1) % MOD1;
		res.y = (at(r).y - at(l - 1).y * pw2[r - l + 1] + MOD2 *
		    MOD2) % MOD2;
		return res;
	}
	int lcp(int i, int j) {
		int l = 0, r = size() - max(i, j);
		while (l < r) {
			int mid = (r - l) / 2 + l + 1;
			if (get(i, i + mid - 1) == get(j, j + mid - 1)) l =
			    mid;
			else r = mid - 1;
		}
		return l;
	}
	bool cmp(int i, int j) {
		int l = lcp(i, j);
		if (i + l - 1 == size() - 1) return 1;
		if (j + l - 1 == size() - 1) return 0;
		return s[i + l] < s[j + l];
	}
};const int K = 269;
const int MOD1 = 1e9 + 7, MOD2 = 998244353;
int pw1[maxLen], pw2[maxLen];
void init_pw() {
	for (int i = 0; i < maxLen; i++) {
		pw1[i] = i ? pw1[i - 1] * K % MOD1 : 1;
		pw2[i] = i ? pw2[i - 1] * K % MOD2 : 1;
	}
}
struct Hash {
        int x, y;
        bool operator == (const Hash& oth) const {
                return x == oth.x && y == oth.y;
        }
};
struct Roll : vector<Hash> {
        string s;
        Roll(const string& s) : s(s) {
                assign(s.size(), {0, 0});
                for (int i = 1; i < s.size(); i++) {
                        at(i).x = (at(i - 1).x * K + s[i]) % MOD1;
                        at(i).y = (at(i - 1).y * K + s[i]) % MOD2;
                }
        }
        Roll() {}
        Hash get(int l, int r) {
                Hash res;
                res.x = (at(r).x - at(l - 1).x * pw1[r - l + 1] + MOD1 *
                    MOD1) % MOD1;
                res.y = (at(r).y - at(l - 1).y * pw2[r - l + 1] + MOD2 *
                    MOD2) % MOD2;
                return res;
        }
        int lcp(int i, int j) {
                int l = 0, r = size() - max(i, j);
                while (l < r) {
                        int mid = (r - l) / 2 + l + 1;
                        if (get(i, i + mid - 1) == get(j, j + mid - 1)) l =
                            mid;
                        else r = mid - 1;
                }
                return l;
        }
        bool cmp(int i, int j) {
                int l = lcp(i, j);
                if (i + l - 1 == size() - 1) return 1;
                if (j + l - 1 == size() - 1) return 0;
                return s[i + l] < s[j + l];
        }
};
```

20 kmp

```
void kmp (string &s) {
        // KMP
        s = " " + s;
    for (int i = 2; i <= n; i++) {
        int j = p[i - 1];
        while (j > 0 && s[j + 1] != s[i])
            j = p[j];
        if (s[j + 1] == s[i]) p[i] = j + 1;
    }
    // Automation
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j < C; j++) {
            int t = i;
            while (t > 0 && s[t + 1] != 'a' + j)
                t = p[t];
            if(s[t + 1] == 'a' + j) f[i][j] = t + 1;
        }
    }
}
```

21 lagrange$_interpolation$

```
void add_poly (vector<ll> &a, vector<ll> &b) {
        for (int i = 0; i < (int) a.size(); i++) {
                a[i] = add (a[i], b[i]);
        }
}
void nhan (vector<ll> &a , ll chim) {
        vector<ll> tmp (k + 2, 0);
        for (int i=0;i<=k;i++) {
                tmp[i + 1] = a[i];
        }
        for (int i=0;i<=k + 1;i++) {
                tmp[i] = sub (tmp[i], 1ll * chim * a[i] % mod);
        }
        a = tmp;
}
void init () {
        dt.resize (k + 2);
        for (int i=1;i<=k + 1;i++) {
                p[i] = add (p[i - 1], cdt (i, k));
        }
        for (int i=0;i<=k + 1;i++) {
                vector<ll> tmp (k + 2, 0);
                tmp[0] = p[i];
                ll dak = 1;
                for (int j=0;j<=k + 1;j++) {
                        if (j==i) continue;
                        nhan (tmp, j);
                        dak = 1ll * dak * cdt ( sub (i, j), mod - 2) % mod;
                }
                for (ll &j : tmp) {
                        j = 1ll * j * dak % mod;

                }
                add_poly (dt, tmp);
        }

}
```

22 max$_m atch$

```
bool dfs (int u) {
        if (vis[u]) return 0;
        vis[u] = 1;
        for (int v : adj[u]) {
                if (ass[v]==0 || dfs (ass[v])) {
                        ass[v] = u;
                        return 1;
                }
        }
        return 0;
}
signed main () {
        random_shuffle (p + 1,p + 1 + n);
        for (int i=1;i<=n;i++) {
                memset (vis , 0 , sizeof (vis));
                match += dfs (p[i]);
        }
        cout<<match<<'\n';
        for (int i=1;i<=m;i++) {
                cout<<ass[i]<<" ";
        }
}
```

23 mincostflow

```
struct MinCostFlow {
        struct edge {
                int u, v, rev;
                int cap, flow;
                int cost;
        };
        int S, T;
        vector<int> dist, inq;
        vector<edge*> par;
        vector<vector<edge>> adj;
        MinCostFlow(int n) : S(n + 1), T(n + 2), dist(n + 3), inq(n + 3),
            par(n + 3), adj(n + 3) {}
        MinCostFlow() {}
        void addEdge(int u, int v, int cap, int cost) {
                adj[u].push_back({u, v, (int)adj[v].size(), cap, 0, cost});
                adj[v].push_back({v, u, (int)adj[u].size() - 1, 0, 0,
                    -cost});
        }
        bool spfa() {
                fill(dist.begin(), dist.end(), INF);
                queue<int> q;
                dist[S] = 0;
                q.push(S);
                inq[S] = 1;
                while (!q.empty()) {
                        int u = q.front(); q.pop();
                        inq[u] = 0;
                        for (edge& e : adj[u]) {
                                if (dist[u] + e.cost >= dist[e.v] || e.cap -
                                    e.flow == 0) continue;
                                dist[e.v] = dist[u] + e.cost;
                                par[e.v] = &e;
                                if (!inq[e.v]) {
                                        q.push(e.v);
                                        inq[e.v] = 1;
                                }
                        }
                }
```

```
		return dist[T] != INF;
	}
	pair<int, int> calc() {
		int flow = 0, cost = 0;
		while (spfa()) {
			int f = INF;
			for (int u = T; u != S; u = par[u]->u)
				f = min(f, par[u]->cap - par[u]->flow);
			for (int u = T; u != S; u = par[u]->u) {
				par[u]->flow += f;
				adj[u][par[u]->rev].flow -= f;
			}
			flow += f;
			cost += f * dist[T];
		}
		return {flow, cost};
	}
};
```

24 pst$_lazy$

```
struct PersistentSegmentTree {
	static const int maxNodes = maxN * (logN + 4) * 3;
	int n, p = 0;
	vector<int> lc, rc;
	vector<int> st;
	vector<int> lazy1, lazy2;
	vector<int> ver;
	PersistentSegmentTree(int n) : n(n), lc(maxNodes), rc(maxNodes),
	    st(maxNodes), lazy1(maxNodes), lazy2(maxNodes) {}
	PersistentSegmentTree() {}
	int leaf(int v) {
		st[++p] = v;
		return p;
	}
	int parent(int l, int r) {
		p++;
		if (p >= maxNodes) {cout << "br0"; exit(0);}
		lc[p] = l;
		rc[p] = r;
		st[p] = st[l] + st[r];
		return p;
	}
	void change(int v1, int v2, int id, int l, int r) {
		st[id] += v1 * (r - l + 1);
		lazy1[id] += v1;
		st[id] += v2 * (l + r) * (r - l + 1) / 2;
		lazy2[id] += v2;
	}
	int lazykid(int v1, int v2, int id, int l, int r) {
		p++;
		if (p >= maxNodes) {cout << "br0"; exit(0);}
		lc[p] = lc[id];
		rc[p] = rc[id];
		st[p] = st[id];
		lazy1[p] = lazy1[id];
		lazy2[p] = lazy2[id];
		change(v1, v2, p, l, r);
		return p;
	}
	void down(int id, int l, int r) {
		if (lazy1[id] == 0 && lazy2[id] == 0) return;
		int mid = (l + r) / 2;
		lc[id] = lazykid(lazy1[id], lazy2[id], lc[id], l, mid);
		rc[id] = lazykid(lazy1[id], lazy2[id], rc[id], mid + 1, r);
		lazy1[id] = 0;
		lazy2[id] = 0;
	}
	int build(int l, int r) {
		if (l == r) return leaf(0);
		int mid = (l + r) / 2;
		return parent(build(l, mid), build(mid + 1, r));
	}
	int update(int L, int R, int v1, int v2, int id, int l, int r) {
		if (R < l || r < L) return id;
		if (L <= l && r <= R) return lazykid(v1, v2, id, l, r);
		down(id, l, r);
		int mid = (l + r) / 2;
		return parent(update(L, R, v1, v2, lc[id], l, mid),
		    update(L, R, v1, v2, rc[id], mid + 1, r));
	}
	int get(int L, int R, int id, int l, int r) {
		if (R < l || r < L) return 0;
		if (L <= l && r <= R) return st[id];
		down(id, l, r);
		int mid = (l + r) / 2;
		int lv = get(L, R, lc[id], l, mid);
```

```
        int rv = get(L, R, rc[id], mid + 1, r);
        return lv + rv;
    }
    void build() {
        ver.push_back(build(1, n));
    }
    void update(int L, int R, int v1, int v2) {
        ver.push_back(update(L, R, v1, v2, ver.back(), 1, n));
    }
    int get(int L, int R, int k) {
        return get(L, R, ver[k], 1, n);
    }
} pst;
```

25 pst$_new$

```
struct vl {
    int x, y;
    bool operator < (const vl& oth) const {
        if (x != oth.x) return x > oth.x;
        return y < oth.y;
    }
};
int n, q;
int a[maxN];
vl b[maxN];
struct PersistentSegmentTree {
    static const int maxNodes = maxN * (logN + 4);
    int n, p = 0;
    vector<int> lc, rc;
    vector<int> st;
    vector<int> ver;
    PersistentSegmentTree(int n) : n(n), lc(maxNodes), rc(maxNodes),
        st(maxNodes) {}
    PersistentSegmentTree() {}
    int leaf(int v) {
        st[++p] = v;
        return p;
    }
    int parent(int l, int r) {
        p++;
        lc[p] = l;
        rc[p] = r;
        st[p] = st[l] + st[r];
        return p;
    }
    int build(int l, int r) {
        if (l == r) return leaf(0);
        int mid = (l + r) / 2;
        return parent(build(l, mid), build(mid + 1, r));
    }
    int update(int i, int v, int id, int l, int r) {
        if (i < l || r < i) return id;
        if (l == r) return leaf(st[id] + v);
        int mid = (l + r) / 2;
        return parent(update(i, v, lc[id], l, mid), update(i, v,
            rc[id], mid + 1, r));
    }
    int get(int L, int R, int id, int l, int r) {
        if (R < l || r < L) return 0;
        if (L <= l && r <= R) return st[id];
        int mid = (l + r) / 2;
        int lv = get(L, R, lc[id], l, mid);
        int rv = get(L, R, rc[id], mid + 1, r);
        return lv + rv;
    }
    void build() {
        ver.push_back(build(1, n));
    }
    void update(int i, int v) {
        ver.push_back(update(i, v, ver.back(), 1, n));
    }
    int get(int L, int R, int k) {
        return get(L, R, ver[k], 1, n);
    }
} pst;
```

26 rabin$_miller$r

```
bool isPrime(ull n) {
    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
    ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
        s = __builtin_ctzll(n-1), d = n >> s;
    for (ull a : A) { // ^ count trailing zeroes
```

```
            ull p = modpow(a%n, d, n), i = s;
            while (p != 1 && p != n - 1 && a % n && i--)
                    p = modmul(p, p, n);
            if (p != n-1 && i != s) return 0;
    }
    return 1;
}
```

27 segtri2D

```
struct io {
     template     <class T> inline static T next() {
          T            n; std::cin >> n;
          return            n;
          }
};

struct node {
     node     () : max(INT_MIN), min(INT_MAX) {}
     int      begin, end, mid, min, max;
};

template<int size> struct segment_tree {
     node      tree[size << 2];

     void      init_tree(int index, int left, int right, int values[]) {
          node          & now = tree[index];
          now           .begin = left, now.end = right;
          now           .mid = (now.begin + now.end) >> 1;
          if            (now.begin != now.end) {
               init_tree                 (L(index), now.begin, now.mid,
    values);
               init_tree                 (R(index), now.mid + 1, now.end,
    values);
               now                 .max = max(now.max,
    max(tree[L(index)].max, tree[R(index)].max));
               now                 .min = min(now.min,
    min(tree[L(index)].min, tree[R(index)].min));
                    } else
               now                 .min = now.max = values[now.begin];
          }

     state      update(int index, int at, int value, int values[]) {
          node           & now = tree[index];
          if             (now.begin == at && now.end == at) {
               now                  .max = now.min = value;
               values                 [at] = value;
               return                 state(value, value);
                    }
          if             (now.begin > at || now.end < at)
               return                 state(now.max, now.min);
          state              s1 = update(L(index), at, value, values);
          state              s2 = update(R(index), at, value, values);
          now            .max = max(s1.first, s2.first);
          now            .min = min(s1.second, s2.second);
          return             state(now.max, now.min);
          }

     state      query(int index, int left, int right) {
          node           & now = tree[index];
          if             (now.begin >= left && now.end <= right)
               return                 state(now.max, now.min);
          if             (now.begin > right || now.end < left)
               return                 state(INT_MIN, INT_MAX);
          state              result(INT_MIN, INT_MAX);
          state              s1 = query(L(index), left, right);
          state              s2 = query(R(index), left, right);
          result             .first = max(result.first, max(s1.first,
    s2.first));
          result             .second = min(result.second, min(s1.second,
    s2.second));
          return             result;
          }
};

segment_tree<MAXN> tree[MAXN];
int values[MAXN][MAXN];
```

28 suffix$_a rray$

```
#include<bits/stdc++.h>
using namespace std;
using ll = long long;
mt19937 rng(time(NULL));
```

```
int ran(int l, int r) {
        return rng() % (r - l + 1) + l;
}
bool is_prime(int x) {
        for (int i = 2; i * i <= x; i++)
        if (x % i == 0) return 0;
        return 1;
}
const int maxLen = 2e5 + 69;
const int K = 317;
const int MS = 1;
int MODS[MS];
ll MM[MS];
using hsh = array<int, MS>;
hsh powK[maxLen];
hsh operator + (hsh h, int c) {
        for (int i = 0; i < MS; i++)
                h[i] = (1ll * h[i] * K + c) % MODS[i];
        return h;
}
void init_hsh() {
        const int MINN = 8e8, MAXX = 1e9;
        for (int i = 0; i < MS; i++) {
                MODS[i] = ran(MINN, MAXX);
                while (!is_prime(MODS[i])) MODS[i]++;
                MM[i] = 1ll * MODS[i] * MODS[i];
                powK[0][i] = 1;
        }
        for (int i = 1; i < maxLen; i++) powK[i] = powK[i - 1] + 0;
}
struct Hash {
        string s;
        vector<hsh> h;
        Hash(const string& s) {
                this->s = s;
                h.resize(s.size());
                for (int i = 1; i < s.size(); i++) h[i] = h[i - 1] + s[i];
        }
        Hash() {}
        hsh get(int l, int r) {
                hsh res;
                for (int i = 0; i < MS; i++)
                res[i] = (h[r][i] - 1ll * h[l - 1][i] * powK[r - l + 1][i]
                    + MM[i]) % MODS[i];
                return res;
        }
        int lcp(int i, int j) {
                int l = 0, r = s.size() - max(i, j);
                while (l < r) {
                        int mid = (r - l) / 2 + l + 1;
                        if (get(i, i + mid - 1) == get(j, j + mid - 1)) l =
                            mid;
                        else r = mid - 1;
                }
                return l;
        }
        bool cmp(int i, int j) {
                int l = lcp(i, j);
                if (i + l - 1 == s.size() - 1) return 1;
                if (j + l - 1 == s.size() - 1) return 0;
                return s[i + l] < s[j + l];
        }
};

bool minimize(int& a, int b) {
        return a > b ? a = b, 1 : 0;
}

const int maxN = 2e5 + 69;
const int logN = 19;
struct vl {
        int x, y;
        bool operator < (const vl& oth) const {
                if (x != oth.x) return x < oth.x;
                return y < oth.y;
        }
};
int n;
string s;
Hash h;
int sa[maxN];
int lcp[maxN];
vl minn[maxN][logN];

set<int> se[maxN];

int x = 0, y = 1;
void Try(int a, int b) {
        // x / y vs a / b
        if (1ll * x * b < 1ll * a * y) x = a, y = b;
}
void build() {
	for (int i = 1; i < n; i++) minn[i][0] = {lcp[i], i};
	for (int j = 1; j < logN; j++)
	for (int i = 1; i + (1 << j) - 1 < n; i++)
		minn[i][j] = min(minn[i][j - 1], minn[i + (1 << (j -
		   1))][j - 1]);
}
vl get(int l, int r) {
	int j = 31 - __builtin_clz(r - l + 1);
	return min(minn[l][j], minn[r - (1 << j) + 1][j]);
}
int solve(int l, int r) {
	if (l == r) {
		se[l].insert(sa[l]);
		return l;
	}

	vl mid = get(l, r - 1);

	int left = solve(l, mid.y);
	int right = solve(mid.y + 1, r);

	if (se[left].size() > se[right].size()) swap(left, right);

	for (auto& u : se[left]) {
		auto it = se[right].lower_bound(u);
		if (it != se[right].end()) Try(mid.x, abs(u - *it));
		if (it != se[right].begin()) {
			it--;
			Try(mid.x, abs(u - *it));
		}
	}
	for (auto& u : se[left]) se[right].insert(u);

	return right;
}
signed main() {
	ios::sync_with_stdio(0);
	cin.tie(0); cout.tie(0);
	init_hsh();
	cin >> s;
	n = s.size();
	s = ' ' + s;
	h = Hash(s);
	for (int i = 1; i <= n; i++) sa[i] = i;
	stable_sort(sa + 1, sa + 1 + n, [&] (int i, int j) {
		return h.cmp(i, j);
	});

	for (int i = 1; i < n; i++) lcp[i] = h.lcp(sa[i], sa[i + 1]);
	build();

	solve(1, n);

	int d = __gcd(x, y);
	x /= d;
	y /= d;
	x += y;
	cout << x << '/' << y;
}
```

29 tarjanSCC

```
void dfs(int u) {
	tin[u] = low[u] = ++cnt;
	st.push_back(u);
	for (int v : a[u]) {
		if (tin[v]) low[u] = min(low[u], tin[v]);
		else {
			dfs(v);
			low[u] = min(low[u], low[v]);
		}
	}
	if (tin[u] == low[u]) {
		scc++;
		while (1) {
			int v = st.back();
			st.pop_back();
			tin[v] = low[v] = INF;
			root[v] = scc;
			sz[scc]++;
			if (v == u) break;
		}
	}
}
```