Team notebook

December 24, 2019

Contents

1 ClosestPair

```
typedef Point<ll> P;
pair<P, P> closest(vector<P> v)
{
	assert(sz(v) > 1);
	set<P> S;
	sort(all(v), [](P a, P b) { return a.y < b.y; });
	pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
	int j = 0;
	trav(p, v)
	{
		P d{1 + (ll)sqrt(ret.first), 0};
		while (v[j].y <= p.y - d.x)
			S.erase(v[j++]);
		auto lo = S.lower_bound(p - d), hi =
		    S.upper_bound(p + d);
		for (; lo != hi; ++lo)
			ret = min(ret, {(*lo - p).dist2(),
			    {*lo, p}});
		S.insert(p);
	}
	return ret.second;
}
```

2 ConvexHull

```
// Returns a vector of indices of the convex hull in counter-
// clockwise order. Points on the edge of the hull between
    two
// other points are not considered part of the hull.
// Time: O (n log n)

typedef Point<ll> P;
vector<P> convexHull(vector<P> pts)
{
	if (sz(pts) <= 1)
		return pts;
	sort(all(pts));
	vector<P> h(sz(pts) + 1);
	int s = 0, t = 0;
	for (int it = 2; it--; s = --t, reverse(all(pts)))
		trav(p, pts)
		{
			while (t >= s + 2 && h[t -
			    2].cross(h[t - 1], p) <= 0)
				t--;
			h[t++] = p;
		}
	return {h.begin(), h.begin() + t - (t == 2 && h[0]
	    == h[1])};
}
```

3 DS

```
// Example :
    https://www.hackerearth.com/challenges/competitive/october-easy
// Enables online insertion of (key, value) pairs and
    querying of maximum value over keys less than a given
    limit.
```

```
// To query minimums instead, set maximum_mode = false.
template <typename T_key, typename T_value, T_value V_INF,
    bool maximum_mode = true>
struct online_prefix_max
{
	map<T_key, T_value> optimal;

	bool better(T_value a, T_value b)
	{
		return (a < b) ^ maximum_mode;
	}

	// Queries the maximum value in the map over all
	    entries with key < `key_limit`.
	T_value query(T_key key_limit) const
	{
		auto it = optimal.lower_bound(key_limit);

		if (it == optimal.begin())
			return maximum_mode ?
			    (is_signed<T_value>::value ?
			    -V_INF : 0) : V_INF;

		it--;
		return it->second;
	}

	// Adds an entry to the map and discards entries
	    that are now obsolete.
	void insert(T_key key, T_value value)
	{
		auto it = optimal.lower_bound(key);

		// Quit if value is suboptimal.
		if (it != optimal.end() && it->first == key)
		{
			if (!better(value, it->second))
				return;
		}
		else if (it != optimal.begin())
		{
			auto prev_it = it;
			prev_it--;

			if (!better(value, prev_it->second))
				return;
		}

		while (it != optimal.end() &&
		    !better(it->second, value))
			it = optimal.erase(it);

		optimal.insert(it, {key, value});
	}
};
```

4 DS1

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

#define IOS                                \
	ios_base::sync_with_stdio(false); \
	cin.tie(0);                       \
	cout.tie(0);
#define int long long int
#define endl "\n"
#define sz(a) (int)((a).size())
#define all(a) a.begin(), a.end()
```

```
const int N = 500009;
const int mod = 1000000007;

// Template : Number of Elements less than equal to x in
    range l to r

vector<int> bit;

struct Query
{
        int l, r, x, idx;
};

struct ArrayElement
{
        int val, idx;
};

bool cmp1(Query q1, Query q2)
{
        return q1.x < q2.x;
}

bool cmp2(ArrayElement x, ArrayElement y)
{
        return x.val < y.val;
}

void update(vector<int> &bit, int idx, int val, int n)
{
        for (; idx <= n; idx += idx & -idx)
                bit[idx] += val;
}

int query(vector<int> &bit, int idx, int n)
{
        int sum = 0;
        for (; idx > 0; idx -= idx & -idx)
                sum += bit[idx];
        return sum;
}

void answerQueries(vector<int> &ans, vector<Query> &queries,
    vector<ArrayElement> &arr)
{
        int n = sz(arr);
        int q = sz(queries);
        bit.assign(n + 1, 0);
        sort(all(arr), cmp2);
        sort(all(queries), cmp1);
        int curr = 0;
        ans.resize(q);
        for (int i = 0; i < q; i++)
        {
                while (arr[curr].val <= queries[i].x && curr
                    < n)
                {
                        update(bit, arr[curr].idx + 1, 1, n);
                        curr++;
                }
                ans[queries[i].idx] = query(bit, queries[i].r
                    + 1, n) - query(bit, queries[i].l, n);
        }
}

int n, Q;
vector<Query> q;
vector<ArrayElement> a;

void pre()
{
}
```

```
int32_t main()
{
	IOS;
	pre();
	cin >> n >> Q;
	a.resize(n);
	q.resize(Q);
	for (int i = 0; i < n; i++)
	{
		cin >> a[i].val;
		a[i].idx = i;
	}
	for (int i = 0; i < Q; i++)
	{
		cin >> q[i].l >> q[i].r >> q[i].x;
		q[i].idx = i;
	}
	vector<int> ans;
	answerQueries(ans, q, a);
	for (auto i : ans)
		cout << i << " ";
	cout << endl;
}
```

5 DSU

```
struct UF
{
	vi e;
	UF(int n) : e(n, -1) {}
	bool same_set(int a, int b) { return find(a) ==
	   find(b); }
	int size(int x) { return -e[find(x)]; }
	int find(int x) { return e[x] < 0 ? x : e[x] =
	   find(e[x]); }
	bool join(int a, int b)
	{
		a = find(a), b = find(b);
		if (a == b)
			return false;
		if (e[a] > e[b])
			swap(a, b);
		e[a] += e[b];
		e[b] = a;
		return true;
	}
};
```

6 Determinant

```
// Description: Calculates determinant of a matrix. Destroys
   the matrix.
// Time: O N 3
double det(vector<vector<double>> &a)
{
	int n = sz(a);
	double res = 1;
	rep(i, 0, n)
	{
		int b = i;
		rep(j, i + 1, n) if (fabs(a[j][i]) >
		   fabs(a[b][i])) b = j;
		if (i != b)
			swap(a[i], a[b]), res *= -1;
		res *= a[i][i];
		if (res == 0)
			return 0;
```

```
                rep(j, i + 1, n)
                {
                        double v = a[j][i] / a[i][i];
                        if (v != 0)
                                rep(k, i + 1, n) a[j][k] -= v *
                                    a[i][k];
                }
        }
        return res;
}
```

7 DivideandConqDp

```
// DivideAndConquerDP.h
// Description: Given a[i] = min lo(i)k<hi(i) (f (i, k))
    where the (minimal)
// optimal k increases with i, computes a[i] for i = L..R  1.
// Time: O ((N + (hi   lo)) log N )
struct DP
{ // Modify at w i l l :
        int lo(int ind) { return 0; }
        int hi(int ind) { return ind; }
        ll f(int ind, int k) { return dp[ind][k]; }
        void store(int ind, int k, ll v) { res[ind] = pii(k,
            v); }
        void rec(int L, int R, int LO, int HI)
        {
                if (L >= R)
                        return;
                int mid = (L + R) >> 1;
                pair<ll, int> best(LLONG_MAX, LO);
                rep(k, max(LO, lo(mid)), min(HI, hi(mid)))
                    best = min(best, make_pair(f(mid, k), k));
                store(mid, best.second, best.first);
                rec(L, mid, LO, best.second + 1);
                rec(mid + 1, R, best.second, HI);
        }
        void solve(int L, int R) { rec(L, R, INT_MIN,
            INT_MAX); }
};
```

8 FracBinary

```
// Description: Given f and N , finds the smallest fraction
    p/q  [0, 1] such
// that f (p/q) is true, and p, q  N . You may want to throw
    an exception from
// f if it finds an exact solution, in which case N can be
    removed.
// Usage: fracBS([](Frac f) { return f.p>=3*f.q; }, 10); //
    {1,3}
// Time: O (log(N ))
struct Frac
{
        ll p, q;
};
template <class F>
Frac fracBS(F f, ll N)
{
        bool dir = 1, A = 1, B = 1;
        Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search
            (0 , N]
        if (f(lo))
                return lo;
        assert(f(hi));
        while (A || B)
        {
```

```
        ll adv = 0, step = 1; // move hi if dir, else lo
        for (int si = 0; step; (step *= 2) >>= si)
        {
            adv += step;
            Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
            if (abs(mid.p) > N || mid.q > N || dir == !f(mid))
            {
                adv -= step;
                si = 2;
            }
        }
        hi.p += lo.p * adv;
        hi.q += lo.q * adv;
        dir = !dir;
        swap(lo, hi);
        A = B;
        B = !!adv;
    }
    return dir ? hi : lo;
}
```

9 IntervalContainer

```
// Description: Add and remove intervals from a set of disjoint intervals.
// Will merge the added interval with any overlapping intervals in the set when
// adding. Intervals are [inclusive, exclusive).
// Time: O (log N )
set<pii>::iterator addInterval(set<pii> &is, int L, int R)
{
    if (L == R)
        return is.end();
    auto it = is.lower_bound({L, R}), before = it;
    while (it != is.end() && it->first <= R)
    {
        R = max(R, it->second);
        before = it = is.erase(it);
    }
    if (it != is.begin() && (--it)->second >= L)
    {
        L = min(L, it->first);
        R = max(R, it->second);
        is.erase(it);
    }
    return is.insert(before, {L, R});
}
void removeInterval(set<pii> &is, int L, int R)
{
    if (L == R)
        return;
    auto it = addInterval(is, L, R);
    auto r2 = it->second;
    if (it->first == L)
        is.erase(it);
    else
        (int &)it->second = L;
    if (R != r2)
        is.emplace(R, r2);
}
```

10 IntervalCover

```
// IntervalCover.h
```

```
// Description: Compute indices of smallest set of intervals
    covering another
// interval. Intervals should be [inclusive, exclusive). To
    support [inclusive,
// inclusive], change (A) to add || R.empty(). Returns empty
    set on failure
// (or if G is empty).
// Time: O (N log N )
template <class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I)
{
        vi S(sz(I)), R;
        iota(all(S), 0);
        sort(all(S), [&](int a, int b) { return I[a] < I[b];
            });
        T cur = G.first;
        int at = 0;
        while (cur < G.second)
        { // (A)
                pair<T, int> mx = make_pair(cur, -1);
                while (at < sz(I) && I[S[at]].first <= cur)
                {
                        mx = max(mx,
                            make_pair(I[S[at]].second, S[at]));
                        at++;
                }
                if (mx.second == -1)
                        return {};
                cur = mx.first;
                R.push_back(mx.second);
        }
        return R;
}
```

11 KMP

```
// Description: pi[x] computes the length of the longest
    prefix of s that ends
// at x, other than s[0...x] itself (abacaba -> 0010123).
    Can be used to find all
// occurrences of a string.
vi pi(const string &s)
{
        vi p(sz(s));
        rep(i, 1, sz(s))
        {
                int g = p[i - 1];
                while (g && s[i] != s[g])
                        g = p[g - 1];
                p[i] = g + (s[i] == s[g]);
        }
        return p;
}
vi match(const string &s, const string &pat)
{
        vi p = pi(pat + \0   + s), res;
        rep(i, sz(p) - sz(s), sz(p)) if (p[i] == sz(pat))
            res.push_back(i - 2 * sz(pat));
        return res;
}
```

12 LIS

```
// Description: Compute indices for the longest increasing
    subsequence.
// Time: O (N log N )

template<class I> vi lis(vector<I> S)
```

```
{
	vi prev(sz(S));
	typedef pair<I, int> p;
	vector<p> res;
	rep(i, 0, sz(S))
	{
		p el{S[i], i};
		//S[ i]+1 for nondecreasing
		auto it = lower_bound(all(res), p{S[i], 0});
		if (it == res.end())
			res.push_back(el), it = --res.end();
		*it = el;
		prev[i] = it == res.begin() ? 0 : (it -
		    1)->second;
	}
	int L = sz(res), cur = res.back().second;
	vi ans(L);
	while (L--)
		ans[L] = cur, cur = prev[cur];
	return ans;
}
```

---

13 LazySeg

```
// Usage: Node* tr = new Node(v, 0, sz(v));

const int inf = 1e9;
struct Node
{
	Node *l = 0, *r = 0;
	int lo, hi, mset = inf, madd = 0, val = -inf;
	Node(int lo, int hi) : lo(lo), hi(hi) {} // Large
	    interval of in f
	Node(vi &v, int lo, int hi) : lo(lo), hi(hi)
	{
		if (lo + 1 < hi)
		{
			int mid = lo + (hi - lo) / 2;
			l = new Node(v, lo, mid);
			r = new Node(v, mid, hi);
			val = max(l->val, r->val);
		}
		else
			val = v[lo];
	}
	int query(int L, int R)
	{
		if (R <= lo || hi <= L)
			return -inf;
		if (L <= lo && hi <= R)
			return val;
		push();
		return max(l->query(L, R), r->query(L, R));
	}
	void set(int L, int R, int x)
	{
		if (R <= lo || hi <= L)
			return;
		if (L <= lo && hi <= R)
			mset = val = x, madd = 0;
		else
		{
			push(), l->set(L, R, x), r->set(L, R,
			    x);
			val = max(l->val, r->val);
		}
	}
	void add(int L, int R, int x)
	{
```

```
        if (R <= lo || hi <= L)
                return;
        if (L <= lo && hi <= R)
        {
                if (mset != inf)
                        mset += x;
                else
                        madd += x;
                val += x;
        }
        else
        {
                push(), l->add(L, R, x), r->add(L, R,
                    x);
                val = max(l->val, r->val);
        }
    }
    void push()
    {
        if (!l)
        {
                int mid = lo + (hi - lo) / 2;
                l = new Node(lo, mid);
                r = new Node(mid, hi);
        }
        if (mset != inf)
                l->set(lo, hi, mset), r->set(lo, hi,
                    mset), mset = inf;
        else if (madd)
                l->add(lo, hi, madd), r->add(lo, hi,
                    madd), madd = 0;
    }
};
```

14 MO

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

#define IOS                             \
       ios_base::sync_with_stdio(false); \
       cin.tie(0);                      \
       cout.tie(0);
// #define int long long int
#define endl "\n"
#define sz(a) (int)((a).size())
#define all(a) a.begin(), a.end()

const int N = 500009;
const int mod = 1000000007;

//  MO's Template :-
   https://cp-algorithms.com/data_structures/sqrt_decomposition.ht

long long int ans = 0;
void remove(int idx);    // TODO: remove value at idx from
   data structure
void add(int idx);      // TODO: add value at idx from data
   structure
long long int get_answer(); // TODO: extract the current
   answer of the data structure

const int block_size = 500;

struct Query
{
       int l, r, idx;
       bool operator<(Query other) const
       {
              return make_pair(l / block_size, r) <
```

```
                make_pair(other.l / block_size,
                    other.r);
        }
};

vector<long long int> mo_s_algorithm(vector<Query> &queries)
{
        vector<long long int> answers(queries.size());
        sort(queries.begin(), queries.end());

        // TODO: initialize data structure

        int cur_l = 0;
        int cur_r = -1;
        // invariant: data structure will always reflect the
            range [cur_l, cur_r]
        for (Query q : queries)
        {
                while (cur_l > q.l)
                {
                        cur_l--;
                        add(cur_l);
                }
                while (cur_r < q.r)
                {
                        cur_r++;
                        add(cur_r);
                }
                while (cur_l < q.l)
                {
                        remove(cur_l);
                        cur_l++;
                }
                while (cur_r > q.r)
                {
                        remove(cur_r);
                        cur_r--;
                }
                answers[q.idx] = get_answer();
        }
        return answers;
}

int t, n;
vector<int> a;
vector<int> cnt((1e6) + 9, 0);

void remove(int idx)
{
        ans -= (long long int)((long long int)(a[idx]) *
            cnt[a[idx]] * cnt[a[idx]]);
        cnt[a[idx]]--;
        ans += (long long int)((long long int)(a[idx]) *
            cnt[a[idx]] * cnt[a[idx]]);
}

void add(int idx)
{
        ans -= (long long int)((long long int)(a[idx]) *
            cnt[a[idx]] * cnt[a[idx]]);
        cnt[a[idx]]++;
        ans += (long long int)((long long int)(a[idx]) *
            cnt[a[idx]] * cnt[a[idx]]);
}

long long int get_answer()
{
        return ans;
}

void pre()
{
```

```
}

int32_t main()
{
	IOS;
	pre();
	cin >> n >> t;
	a.resize(n);
	for (int i = 0; i < n; i++)
		cin >> a[i];
	vector<Query> q;
	q.resize(t);
	for (int i = 0; i < t; i++)
	{
		cin >> q[i].l >> q[i].r;
		q[i].l--;
		q[i].r--;
		q[i].idx = i;
	}
	vector<long long int> A;
	A.resize(t);
	A = mo_s_algorithm(q);
	for (auto i : A)
		cout << i << endl;
	return 0;
}
```

15 MST

```
struct Edge
{
	int u, v, weight;
	bool operator<(Edge const &other)
	{
		return weight < other.weight;
	}
};

int n;
vector<Edge> edges;

int cost = 0;
vector<int> tree_id(n);
vector<Edge> result;
for (int i = 0; i < n; i++)
	tree_id[i] = i;

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

for (Edge e : edges)
{
	if (tree_id[e.u] != tree_id[e.v])
	{
		cost += e.weight;
		result.push_back(e);

		int old_id = tree_id[e.u], new_id =
		    tree_id[e.v];
		for (int i = 0; i < n; i++)
		{
			if (tree_id[i] == old_id)
				tree_id[i] = new_id;
		}
	}
}
```

16 Manacher

```
// Description: For each position in a string, computes
    p[0][i] = half length
// of longest even palindrome around pos i, p[1][i] =
    longest odd (half rounded
// down).
// Time: O (N )

array<vi, 2> manacher(const string &s)
{
	int n = sz(s);
	array<vi, 2> p = {vi(n + 1), vi(n)};
	rep(z, 0, 2) for (int i = 0, l = 0, r = 0; i < n;
	    i++)
	{
		int t = r - i + !z;
		if (i < r)
			p[z][i] = min(t, p[z][l + t]);
		int L = i - p[z][i], R = i + p[z][i] - !z;
		while (L >= 1 && R + 1 < n && s[L - 1] == s[R
		    + 1])
			p[z][i]++, L--, R++;
		if (R > r)
			l = L, r = R;
	}
	return p;
}
```

17 MinRot

```
// Description: Finds the lexicographically smallest
    rotation of a string.
// Usage: rotate(v.begin(), v.begin()+min rotation(v),
    v.end());
// Time: O (N )

int min_rotation(string s)
{
	int a = 0, N = sz(s);
	s += s;
	rep(b, 0, N) rep(i, 0, N)
	{
		if (a + i == b || s[a + i] < s[b + i])
		{
			b += max(0, i - 1);
			break;
		}
		if (s[a + i] > s[b + i])
		{
			a = b;
			break;
		}
	}
	return a;
}
```

18 MyLca

```
int n;
int a, b;
int tree[4 * N + 9];
bool isRoot[N];
vector<int> v[N];
vector<int> tour;
vector<int> height;
int first[N];

void build(int l, int r, int idx)
{
	if (l == r)
```

```
	{
		tree[idx] = l;
		return;
	}
	int mid = (l + r) / 2;

	build(l, mid, 2 * idx + 1);
	build(mid + 1, r, 2 * idx + 2);

	// tree[idx] = max(tree[2 * idx + 1], tree[2 * idx +
	    2]);

	if (height[tree[2 * idx + 1]] < height[tree[2 * idx
	    + 2]])
	{
		tree[idx] = tree[2 * idx + 1];
	}
	else
	{
		tree[idx] = tree[2 * idx + 2];
	}
}

int query(int l, int r, int s, int e, int idx)
{
	if (l > e || r < s)
	{
		return INT_MIN;
	}

	if (l >= s && r <= e)
	{
		return tree[idx];
	}

	int mid = (l + r) / 2;

	int a = query(l, mid, s, e, 2 * idx + 1);
	int b = query(mid + 1, r, s, e, 2 * idx + 2);

	// Hard-Code corner case
	if (a == INT_MIN)
		return b;
	else if (b == INT_MIN)
		return a;
	else if (height[a] < height[b])
		return a;
	else
		return b;
}

void pre()
{
	memset(tree, 0, sizeof(tree));
	memset(first, -1, sizeof(first));
	memset(isRoot, true, sizeof(isRoot));
	tour.clear();
	height.clear();
	for (int i = 0; i < N; i++)
	{
		v[i].clear();
	}
}

void euler(int x, int p, int hgh)
{
	tour.push_back(x);
	first[x] = (tour.size() - 1);
	height.push_back(hgh);
	for (auto i : v[x])
	{
		if (i == p)
```

```
                continue;
            euler(i, x, hgh + 1);
            tour.push_back(x);
            height.push_back(hgh);
        }
}

int32_t main()
{
        IOS;
        int t, q;
        cin >> t;
        for (int test = 1; test <= t; test++)
        {
            cout << "Case " << test << ":\n";
            pre();
            cin >> n;
            for (int i = 1; i <= n; i++)
            {
                cin >> a;
                while (a--)
                {
                    cin >> b;
                    v[i].push_back(b);
                    isRoot[b] = false;
                }
            }
            int root = -1;
            for (int i = 1; i <= n; i++)
            {
                if (isRoot[i])
                {
                    root = i;
                }
            }
            euler(root, 0, 1);
            build(0, tour.size() - 1, 0);
            cin >> q;
            while (q--)
            {
                cin >> a >> b;
                if (first[a] > first[b])
                    swap(a, b);
                cout << tour[query(0, tour.size() - 1,
                    first[a], first[b], 0)] << endl;
            }
        }
}
```

19 Optimizations

```
// x & -x is the least bit in x.
// for (int x = m; x; ) { --x &= m; ... }
// loops over all subset masks of m (except m itself).
// c = x&-x, r = x+c; (((rx) >> 2)/c) | r
// is the next number after x with the same number of
// bits set.
// rep(b,0,K) rep(i,0,(1 << K))
// if (i & 1 << b) D[i] += D[i(1 << b)];
// computes all sums of subsets.

typedef unsigned long long ull;
typedef __uint128_t L;
struct FastMod
{
        ull b, m;
        FastMod(ull b) : b(b), m(ull((L(1) << 64) / b)) {}
        ull reduce(ull a)
        {
```

```
        ull q = (ull)((L(m) * a) >> 64), r = a - q *
            b;
        return r >= b ? r - b : r;
    }
};

// Description: When you need to dynamically allocate many
    objects and
//  dont  care about freeing them. new  X  otherwise has an
    overhead of some-
// thing like 0.05us + 16 bytes per allocation.

// Either g l o b a l l y or in a single class :
static char buf[450 << 20];
void *operator new(size_t s)
{
    static size_t i = sizeof buf;
    assert(s < i);
    return (void *)&buf[i -= s];
}
void operator delete(void *) {}
```

20 OrderStats

```
// order_of_key (k) : Number of items strictly smaller than
    k .
// find_by_order(k) : K-th element in a set (counting from
    zero).

#include <bits/extc++.h>
using namespace std;
using namespace __gnu_pbds;
template <class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
        tree_order_statistics_node_update>;
void example()
{
    Tree<int> t, t2;
    t.insert(8);
    auto it = t.insert(10).first;
    assert(it == t.lower_bound(9));
    assert(t.order_of_key(10) == 1);
    assert(t.order_of_key(11) == 2);
    assert(*t.find_by_order(0) == 8);
    t.join(t2); // assuming T < T2 or T > T2, merge t2
        into t
}

// Description: Hash map with the same API as unordered map,
    but
// faster. Initial capacity must be a power of 2 (if
    provided).

#include <bits/extc++.h>
// To use most b i t s rather than j u s t the lowest ones :
struct chash
{
    const uint64_t C = ll(2e18 * M_PI) + 71; // large
        odd number
    ll operator()(ll x) const { return
        __builtin_bswap64(x * C); }
};
__gnu_pbds::gp_hash_table<ll, int, chash> h({}, {}, {}, {},
    {1 << 16});
```

21 Point

```
template <class T>
```

```
int sgn(T x) { return (x > 0) - (x < 0); }
template <class T>
struct Point
{
	typedef Point P;
	T x, y;
	explicit Point(T x = 0, T y = 0) : x(x), y(y) {}
	bool operator<(P p) const { return tie(x, y) <
	    tie(p.x, p.y); }
	bool operator==(P p) const { return tie(x, y) ==
	    tie(p.x, p.y); }
	P operator+(P p) const { return P(x + p.x, y + p.y);
	    }
	P operator-(P p) const { return P(x - p.x, y - p.y);
	    }
	P operator*(T d) const { return P(x * d, y * d); }
	P operator/(T d) const { return P(x / d, y / d); }
	T dot(P p) const { return x * p.x + y * p.y; }
	T cross(P p) const { return x * p.y - y * p.x; }
	T cross(P a, P b) const { return (a - *this).cross(b
	    - *this); }
	T dist2() const { return x * x + y * y; }
	double dist() const { return sqrt((double)dist2()); }
	// angle to  xaxis  in interval [ pi  , pi ]
	double angle() const { return atan2(y, x); }
	P unit() const { return *this / dist(); } // makes d
	    i s t ()=1
	P perp() const { return P(-y, x); }   // rotates +90
	    degrees
	P normal() const { return perp().unit(); }
	// returns point rotated a     radians ccw around the
	    origin
	P rotate(double a) const
	{
		return P(x * cos(a) - y * sin(a), x * sin(a)
		    + y * cos(a));
	}
	friend ostream &operator<<(ostream &os, P p)
	{
		return os << "(" << p.x << "," << p.y << ")";
	}
};
```

22 RMQ

```
// Description: Range Minimum Queries on an array. Returns
    min(V[a], V[a
// + 1], ... V[b - 1]) in constant time.
// Usage: RMQ rmq(values);
// rmq.query(inclusive, exclusive);
// Time: O (|V | log |V | + Q)
template <class T>
struct RMQ
{
	vector<vector<T>> jmp;
	RMQ(const vector<T> &V)
	{
		int N = sz(V), on = 1, depth = 1;
		while (on < N)
			on *= 2, depth++;
		jmp.assign(depth, V);
		rep(i, 0, depth - 1) rep(j, 0, N)
		    jmp[i + 1][j] = min(jmp[i][j],
		                        jmp[i][min(N - 1, j + (1
		                            << i))]);
	}
	T query(int a, int b)
	{
		assert(a < b); // or return i n f i f a == b
		int dep = 31 - __builtin_clz(b - a);
```

```
                return min(jmp[dep][a], jmp[dep][b - (1 <<
                    dep)]);
        }
};
```

23 SPOJLCA

```
// LCA.h
// Description: Data structure for computing lowest common
    ancestors in a
// tree (with 0 as root). C should be an adjacency list of
    the tree, either directed
// or undirected. Can also find the distance between two
    nodes.
// Usage: LCA lca(undirGraph);
// lca.query(firstNode, secondNode);
// lca.distance(firstNode, secondNode);
// Time: O (N log N + Q)
// "../data-structures/RMQ.h"
typedef vector<pii> vpi;
typedef vector<vpi> graph;

struct LCA
{
        vi time;
        vector<ll> dist;
        RMQ<pii> rmq;
        LCA(graph &C) : time(sz(C), -99), dist(sz(C)),
            rmq(dfs(C)) {}
        vpi dfs(graph &C)
        {
                vector<tuple<int, int, int, ll>> q(1);
                vpi ret;
                int T = 0, v, p, d;
                ll di;
                while (!q.empty())
                {
                        tie(v, p, d, di) = q.back();
                        q.pop_back();
                        if (d)
                                ret.emplace_back(d, p);
                        time[v] = T++;
                        dist[v] = di;
                        trav(e, C[v]) if (e.first != p)
                            q.emplace_back(e.first, v, d + 1,
                                di + e.second);
                }
                return ret;
        }
        int query(int a, int b)
        {
                if (a == b)
                        return a;
                a = time[a], b = time[b];
                return rmq.query(min(a, b), max(a, b)).second;
        }
        ll distance(int a, int b)
        {
                int lca = query(a, b);
                return dist[a] + dist[b] - 2 * dist[lca];
        }
};
```

24 Zfunc

```
// Description: z[x] computes the length of the longest
    common prefix of s[i:]
// and s, except z[0] = 0. (abacaba -> 0010301)
```

```
vi Z(string S)
{
	vi z(sz(S));
	int l = -1, r = -1;
	rep(i, 1, sz(S))
	{
		z[i] = i >= r ? 0 : min(r - i, z[i - l]);
		while (i + z[i] < sz(S) && S[i + z[i]] ==
		    S[z[i]])
			z[i]++;
		if (i + z[i] > r)
			l = i, r = i + z[i];
	}
	return z;
}
```

25 articulation

```
int n;                  // number of nodes
vector<vector<int>> adj; // adjacency list of graph
//The implementation needs to distinguish three cases: when
    we go down the edge in DFS tree, when we find a back
    edge to an ancestor of the vertex and when we return to
    a parent of the vertex. These are the cases:
// visited[to]=false - the edge is part of DFS tree;
// visited[to]=true && toparent - the edge is back edge to
    one of the ancestors;
// to=parent - the edge leads back to parent in DFS tree.


vector<bool> visited;
vector<int> tin, low;
int timer;
```

```
void dfs(int v, int p = -1)
{
	visited[v] = true;
	tin[v] = low[v] = timer++;
	int children = 0;
	for (int to : adj[v])
	{
		if (to == p)
			continue;
		if (visited[to])
		{
			low[v] = min(low[v], tin[to]);
		}
		else
		{
			dfs(to, v);
			low[v] = min(low[v], low[to]);
			if (low[to] >= tin[v] && p != -1)
				IS_CUTPOINT(v);
			++children;
		}
	}
	if (p == -1 && children > 1)
		IS_CUTPOINT(v);
}

void find_cutpoints()
{
	timer = 0;
	visited.assign(n, false);
	tin.assign(n, -1);
	low.assign(n, -1);
	for (int i = 0; i < n; ++i)
	{
		if (!visited[i])
```

```
                        dfs(i);
        }
}
```

26 bridge

```
// The implementation needs to distinguish three cases: when
    we go down the edge in DFS tree, when we find a back
    edge to an ancestor of the vertex and when we return to
    a parent of the vertex. These are the cases:

// visited[to]=false - the edge is part of DFS tree;
// visited[to]=true && toparent - the edge is back edge to
    one of the ancestors;
// to=parent - the edge leads back to parent in DFS tree.
// To implement this, we need a depth first search function
    which accepts the parent vertex of the current node.

int n;                  // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1)
{
        visited[v] = true;
        tin[v] = low[v] = timer++;
        for (int to : adj[v])
        {
                if (to == p)
                        continue;
                if (visited[to])
                {
                        low[v] = min(low[v], tin[to]);
                }
                else
                {
                        dfs(to, v);
                        low[v] = min(low[v], low[to]);
                        if (low[to] > tin[v])
                                IS_BRIDGE(v, to);
                }
        }
}

void find_bridges()
{
        timer = 0;
        visited.assign(n, false);
        tin.assign(n, -1);
        low.assign(n, -1);
        for (int i = 0; i < n; ++i)
        {
                if (!visited[i])
                        dfs(i);
        }
}
```

27 lazyPropogation

```
int a[N], tree[4 * N + 2], lazy[4 * N + 2];

void build(int l, int r, int idx)
{
        if (l == r)
        {
```

```
                tree[idx] = a[l];
                return;
        }
        int mid = (l + r) / 2;

        build(l, mid, 2 * idx + 1);
        build(mid + 1, r, 2 * idx + 2);

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

void update(int l, int r, int s, int e, int idx, int val)
{

        if (lazy[idx] != 0)
        {
                tree[idx] += lazy[idx];
                if (l != r)
                {
                        lazy[2 * idx + 1] += lazy[idx];
                        lazy[2 * idx + 2] += lazy[idx];
                }
                lazy[idx] = 0;
        }

        if (l > e || r < s)
        {
                return;
        }
        int mid = (l + r) / 2;

        if (l >= s && r <= e && (l != r))
        {
                tree[idx] += val;
                lazy[2 * idx + 1] += val;
                lazy[2 * idx + 2] += val;
                return;
        }

        else if (l >= s && r <= e)
        {
                tree[idx] += val;
                return;
        }

        update(l, mid, s, e, 2 * idx + 1, val);
        update(mid + 1, r, s, e, 2 * idx + 2, val);

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

int query(int l, int r, int s, int e, int idx)
{
        if (l > e || r < s)
        {
                return INT_MIN;
        }

        if (lazy[idx] != 0)
        {
                tree[idx] += lazy[idx];
                int val = lazy[idx];
                lazy[idx] = 0;
                if (l != r)
                {
                        lazy[2 * idx + 1] += val;
                        lazy[2 * idx + 2] += val;
                }
        }
```

```
        if (l >= s && r <= e)
        {
                return tree[idx];
        }

        int mid = (l + r) / 2;

        int a = query(l, mid, s, e, 2 * idx + 1);
        int b = query(mid + 1, r, s, e, 2 * idx + 2);

        return max(a, b);
}
```

28 slidingWindow

```
// k is the window size
deque<pair<int, int>> q;

void del()
{
        while (!q.empty())
        {
                q.pop_back();
        }
}

void insertion(pair<int, int> p)
{
        while (!q.empty())
        {
                pair<int, int> lst = q.back();
                if (lst.first >= p.first)
                {
                        q.push_back(p);
                        return;
                }
                else
                {
                        q.pop_back();
                }
        }

        q.push_back(p);
}

int getMax(int j)
{
        int per = j - k + 1;

        while (q.front().second < per)
        {
                q.pop_front();
        }

        return q.front().first;
}
```