



Carnegie Mellon University

CMU 2

Zack Lee, Lawrence Chen, Howard Halim

adapted from KACTL and MIT NULL

2019-10-24

- 1 Contest
- 2 Mathematics
- 3 Data Structures
- 4 Number Theory
- 5 Combinatorial
- 6 Numerical
- 7 Graphs
- 8 Geometry
- 9 Strings

Contest (1)

template.cpp30 lines

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

using namespace std;

#define f first
#define s second
#define pb push_back
#define mp make_pair
#define sq(a) (a)*(a)
#define all(v) v.begin(), v.end()
#define sz(v) (int)v.size()

#define MOO(i, a, b) for(int i=a; i<b; i++)
#define M00(i, a) for(int i=0; i<a; i++)
#define MOOd(i,a,b) for(int i = (b)-1; i >= a; i--)
#define M00d(i,a) for(int i = (a)-1; i>=0; i--)

#define FAST ios::sync_with_stdio(0); cin.tie(0);
#define finish(x) return cout << x << '\n', 0;

typedef long long ll;
typedef long double ld;
typedef vector<int> vi;
typedef pair<int,int> pi;
typedef pair<ld,ld> pd;
typedef complex<ld> cd;

int main() { FAST

}

.bashrc6 lines
```

co() {
 g++ -std=c++11 -O2 -Wall -Wl,-stack_size -Wl,0x10000000 -o
 ↪\$1 \$1.cc
}
run() {
 co \$1 && ./\$1
}

1 .vimrc3 lines

```
set nosp backspace=indent,eol,start nu ru si ts=4 sw=4 is hls  
    ↪sm mouse=a  
syntax on  
filetype plugin indent on
```

3

5 cppreference.txt7 lines

where V, W are lengths of sides opposite angles v, w .

$$a \cos x + b \sin x = r \cos(x - \phi)$$
$$a \sin x + b \cos x = r \sin(x + \phi)$$

where $r = \sqrt{a^2 + b^2}, \phi = \operatorname{atan2}(b, a)$.

2.4 Geometry

2.4.1 Triangles

Side lengths: a, b, c

Semiperimeter: $p = \frac{a + b + c}{2}$

Area: $A = \sqrt{p(p - a)(p - b)(p - c)}$

Circumradius: $R = \frac{abc}{4A}$

Inradius: $r = \frac{A}{p}$

Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two):

$$s_a = \sqrt{bc \left[1 - \left(\frac{a}{b + c} \right)^2 \right]}$$

Law of sines: $\frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c} = \frac{1}{2R}$

Law of cosines: $a^2 = b^2 + c^2 - 2bc \cos \alpha$

Law of tangents: $\frac{a + b}{a - b} = \frac{\tan \frac{\alpha + \beta}{2}}{\tan \frac{\alpha - \beta}{2}}$

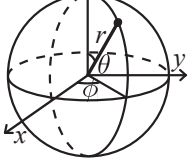
2.4.2 Quadrilaterals

With side lengths a, b, c, d , diagonals e, f , diagonals angle θ , area A and magic flux $F = b^2 + d^2 - a^2 - c^2$:

$$4A = 2ef \cdot \sin \theta = F \tan \theta = \sqrt{4e^2 f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is 180° ,

2.4.3 Spherical coordinates



$$x = r \sin \theta \cos \phi \qquad r = \sqrt{x^2 + y^2 + z^2}$$
$$y = r \sin \theta \sin \phi \quad \theta = \operatorname{acos}(z / \sqrt{x^2 + y^2 + z^2})$$
$$z = r \cos \theta \qquad \phi = \operatorname{atan2}(y, x)$$

2.5 Derivatives/Integrals

$$\frac{d}{dx} \arcsin x = \frac{1}{\sqrt{1 - x^2}} \qquad \frac{d}{dx} \arccos x = -\frac{1}{\sqrt{1 - x^2}}$$
$$\frac{d}{dx} \tan x = 1 + \tan^2 x \qquad \frac{d}{dx} \arctan x = \frac{1}{1 + x^2}$$
$$\int \tan ax = -\frac{\ln |\cos ax|}{a} \qquad \int x \sin ax = \frac{\sin ax - ax \cos ax}{a^2}$$
$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2} \operatorname{erf}(x) \qquad \int x e^{ax} dx = \frac{e^{ax}}{a^2} (ax - 1)$$

Integration by parts:

$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

2.6 Sums

$$c^a + c^{a+1} + \dots + c^b = \frac{c^{b+1} - c^a}{c - 1}, c \neq 1$$

$$1 + 2 + 3 + \dots + n = \frac{n(n + 1)}{2}$$

$$1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(2n + 1)(n + 1)}{6}$$

$$1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^2(n + 1)^2}{4}$$

$$1^4 + 2^4 + 3^4 + \dots + n^4 = \frac{n(n + 1)(2n + 1)(3n^2 + 3n - 1)}{30}$$

2.7 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots, (-\infty < x < \infty)$$

$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots, (-1 < x \leq 1)$$

$$\sqrt{1 + x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \dots, (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots, (-\infty < x < \infty)$$

2.8 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x . It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent X and Y ,

$$V(aX + bY) = a^2V(X) + b^2V(Y).$$

2.8.1 Discrete distributions

Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\operatorname{Bin}(n, p)$, $n = 1, 2, \dots$, $0 \leq p \leq 1$.

$$p(k) = \binom{n}{k} p^k (1 - p)^{n - k}$$

$$\mu = np, \sigma^2 = np(1 - p)$$

$\operatorname{Bin}(n, p)$ is approximately $\operatorname{Po}(np)$ for small p .

First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each wich yields success with probability p is $\operatorname{Fs}(p)$, $0 \leq p \leq 1$.

$$p(k) = p(1 - p)^{k - 1}, k = 1, 2, \dots$$

$$\mu = \frac{1}{p}, \sigma^2 = \frac{1 - p}{p^2}$$

Poisson distribution

The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\operatorname{Po}(\lambda)$, $\lambda = t\kappa$.

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}, k = 0, 1, 2, \dots$$

$$\mu = \lambda, \sigma^2 = \lambda$$

2.8.2 Continuous distributions

Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is $U(a,b)$, $a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2}, \sigma^2 = \frac{(b-a)^2}{12}$$

Exponential distribution

The time between events in a Poisson process is $\text{Exp}(\lambda)$, $\lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}$$

Normal distribution

Most real random values with mean μ and variance σ^2 are well described by $\mathcal{N}(\mu,\sigma^2)$, $\sigma > 0$.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If $X_1 \sim \mathcal{N}(\mu_1,\sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2,\sigma_2^2)$ then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

2.9 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let X_1, X_2, \dots be a sequence of random variables generated by the Markov process. Then there is a transition matrix $\mathbf{P} = (p_{ij})$, with $p_{ij} = \Pr(X_n = i | X_{n-1} = j)$, and $\mathbf{p}^{(n)} = \mathbf{P}^n \mathbf{p}^{(0)}$ is the probability distribution for X_n (i.e., $p_i^{(n)} = \Pr(X_n = i)$), where $\mathbf{p}^{(0)}$ is the initial distribution.

π is a stationary distribution if $\pi = \pi \mathbf{P}$. If the Markov chain is *irreducible* (it is possible to get to any state from any state), then $\pi_i = \frac{1}{\mathbb{E}(T_i)}$ where $\mathbb{E}(T_i)$ is the expected time between two visits in state i . π_j/π_i is the expected number of visits in state j between two visits in state i .

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors, π_i is proportional to node i 's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1). $\lim_{k \rightarrow \infty} \mathbf{P}^k = \mathbf{1}\pi$.

A Markov chain is an A-chain if the states can be partitioned into two sets \mathbf{A} and \mathbf{G} , such that all states in \mathbf{A} are absorbing ($p_{ii} = 1$), and all states in \mathbf{G} leads to an absorbing state in \mathbf{A} . The probability for absorption in state $i \in \mathbf{A}$, when the initial state is j , is $a_{ij} = p_{ij} + \sum_{k \in \mathbf{G}} a_{ik} p_{kj}$. The expected time until absorption, when the initial state is i , is $t_i = 1 + \sum_{k \in \mathbf{G}} p_{ki} t_k$.

Data Structures (3)

3.1 STL

MapComparator.h	
Description: custom comparator for map / set	8 lines
<pre>struct cmp { bool operator()(const int& l, const int& r) const { return l > r; } }; set<int,cmp> s; // FOR(i,10) s.insert(rand()); trav(i,s) ps(i); map<int,int,cmp> m;</pre>	

CustomHash.h	
Description: faster than standard unordered map	23 lines
<pre>struct chash { static uint64_t splitmix64(uint64_t x) { // http://xorshift.di.unimi.it/splitmix64.c x += 0x9e3779b97f4a7c15; x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9; x = (x ^ (x >> 27)) * 0x94d049b133111eb; return x ^ (x >> 31); } size_t operator()(uint64_t x) const { static const uint64_t FIXED_RANDOM = chrono::steady_clock::now(). time_since_epoch().count(); return splitmix64(x + FIXED_RANDOM); } }; template<class K, class V> using um = unordered_map<K, V, chash ⇨>; template<class K, class V> using ht = gp_hash_table<K, V, chash ⇨>; template<class K, class V> V get(ht<K,V>& u, K x) { return u.find(x) == end(u) ? 0 : u[x]; }</pre>	

OrderStatisticTree.h	
Description: A set (not multiset!) with support for finding the n 'th element, and finding the index of an element.	
Time: $\mathcal{O}(\log N)$	
<code><ext/pb_ds/tree_policy.hpp></code> , <code><ext/pb_ds/assoc.container.hpp></code>	18 lines
<pre>using namespace __gnu_pbds; template<class T> using Tree = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>; // to get a map, change null_type #define ook order_of_key #define fbo find_by_order void treeExample() { Tree<int> t, t2; t.insert(8); auto it = t.insert(10).f; assert(it == t.lb(9)); assert(t.ook(10) == 1); assert(t.ook(11) == 2); assert(*t.fbo(0) == 8); t.join(t2); // assuming T < T2 or T > T2, merge t2 into t }</pre>	

Rope.h	
Description: insert element at n -th position, cut a substring and re-insert somewhere else	
Time: $\mathcal{O}(\log N)$ per operation? not well tested	
<code><ext/rope></code>	13 lines
<pre>using namespace __gnu_cxx; void ropeExample() { rope<int> v(5, 0); FOR(i,sz(v)) v.mutable_reference_at(i) = i+1; // or push_back rope<int> cur = v.substr(1,2); v.erase(1,2); FOR(i,sz(v)) cout << v[i] << " "; // 1 4 5 cout << "\n"; v.insert(v.mutable_begin()+2,cur); for (rope<int>::iterator it = v.mutable_begin(); it != v. ⇨mutable_end(); ++it) cout << *it << " "; // 1 4 2 3 5 cout << "\n"; }</pre>	

LineContainer.h	
Description: Given set of lines, computes greatest y -coordinate for any x	
Time: $\mathcal{O}(\log N)$	
<pre>struct Line { mutable ll k, m, p; // slope, y-intercept, last optimal x ll eval(ll x) { return k*x+m; } bool operator<(const Line& o) const { return k < o.k; } bool operator<(ll x) const { return p < x; } }; struct LC : multiset<Line,less<>> { // for doubles, use inf = 1/.0, div(a,b) = a/b const ll inf = LLONG_MAX; ll div(ll a, ll b) { return a/b-((a^b) < 0 && a%b); } // ⇨floored division ll bet(const Line& x, const Line& y) { // last x such that ⇨first line is better if (x.k == y.k) return x.m >= y.m ? inf : -inf; return div(y.m-x.m,x.k-y.k); } bool isect(iterator x, iterator y) { // updates x->p, ⇨determines if y is unneeded if (y == end()) { x->p = inf; return 0; } }</pre>	

```

    x->p = bet(*x,*y); return x->p >= y->p;
}
void add(ll k, ll 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, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p) isect(x,
        ↪erase(y));
}
ll query(ll x) {
    assert(!empty());
    auto l = *lb(x);
    return l.k*x+l.m;
}
};
```

3.2 1D Range Queries

RMQ.h
Description: 1D range minimum query
Time: $\mathcal{O}(N \log N)$ build, $\mathcal{O}(1)$ query

```

template<class T> struct RMQ {
    constexpr static int level(int x) {
        return 31-__builtin_clz(x);
    } // floor(log2(x))
    vector<vi> jmp;
    vector<T> v;
    int comb(int a, int b) {
        return v[a] == v[b] ? min(a,b) : (v[a] < v[b] ? a : b);
    } // index of minimum

    void init(const vector<T>& _v) {
        v = _v; jmp = {vi(sz(v)); iota(all(jmp[0]),0);
        for (int j = 1; 1<<j <= sz(v); ++j) {
            jmp.pb(vi(sz(v)-(1<<j)+1));
            FOR(i,sz(jmp[j])) jmp[j][i] = comb(jmp[j-1][i],
                jmp[j-1][i+(1<<(j-1))]);
        }
    }

    int index(int l, int r) { // get index of min element
        int d = level(r-l+1);
        return comb(jmp[d][l], jmp[d][r-(1<<d)+1]);
    }
    T query(int l, int r) { return v[index(l,r)]; }
};
```

BIT.h
Description: N-D range sum query with point update
Time: $\mathcal{O}\left((\log N)^D\right)$

```

template <class T, int ...Ns> struct BIT {
    T val = 0;
    void upd(T v) { val += v; }
    T query() { return val; }
};

template <class T, int N, int... Ns> struct BIT<T, N, Ns...> {
    BIT<T,Ns...> bit[N+1];
    template<typename... Args> void upd(int pos, Args... args) {
        for (; pos <= N; pos += (pos&-pos)) bit[pos].upd(args...);
    }
    template<typename... Args> T sum(int r, Args... args) {
        T res = 0; for (; r; r -= (r&-r)) res += bit[r].query(args
            ↪...);
        return res;
    }
};
```

```

template<typename... Args> T query(int l, int r, Args... args
    ↪) {
    return sum(r,args...)-sum(l-1,args...);
}
}; // BIT<int,10,10> gives a 2D BIT
```

BITrange.h
Description: 1D range increment and sum query
Time: $\mathcal{O}(\log N)$

```

"BIT.h" 11 lines
template<class T, int SZ> struct BITrange {
    BIT<T,SZ> bit[2]; // piecewise linear functions
    // let cum[x] = sum_{i=1}^x a[i]
    void upd(int hi, T val) { // add val to a[1..hi]
        bit[1].upd(1,val), bit[1].upd(hi+1,-val); // if x <= hi,
            ↪cum[x] += val*x
        bit[0].upd(hi+1,hi*val); // if x > hi, cum[x] += val*hi
    }
    void upd(int lo, int hi, T val) { upd(lo-1,-val), upd(hi,val)
        ↪; }
    T sum(int x) { return bit[1].sum(x)*x+bit[0].sum(x); } // get
        ↪ cum[x]
    T query(int x, int y) { return sum(y)-sum(x-1); }
};
```

SegTree.h
Description: 1D point update, range query
Time: $\mathcal{O}(\log N)$

```

template<class T> struct Seg {
    const T ID = 0; // comb(ID,b) must equal b
    T comb(T a, T b) { return a+b; } // easily change this to min
        ↪ or max
    int n; vector<T> seg;
    void init(int _n) { n = _n; seg.rsz(2*n); }

    void pull(int p) { seg[p] = comb(seg[2*p],seg[2*p+1]); }
    void upd(int p, T value) { // set value at position p
        seg[p += n] = value;
        for (p /= 2; p; p /= 2) pull(p);
    }

    T query(int l, int r) { // sum on interval [l, r]
        T ra = ID, rb = ID; // make sure non-commutative operations
            ↪ work
        for (l += n, r += n+1; l < r; l /= 2, r /= 2) {
            if (l&1) ra = comb(ra,seg[l++]);
            if (r&1) rb = comb(seg[--r],rb);
        }
        return comb(ra,rb);
    }
};
```

SegTreeBeats.h
Description: supports modifications in the form ckmin(a,i,t) for all $l \leq i \leq r$, range max and sum queries
Time: $\mathcal{O}(\log N)$

```

65 lines
template<int SZ> struct SegTreeBeats {
    int N;
    ll sum[2*SZ];
    int mx[2*SZ][2], maxCnt[2*SZ];

    void pull(int ind) {
        FOR(i,2) mx[ind][i] = max(mx[2*ind][i],mx[2*ind+1][i]);
        maxCnt[ind] = 0;
        FOR(i,2) {
            if (mx[2*ind+i][0] == mx[ind][0])
```

```

            maxCnt[ind] += maxCnt[2*ind+i];
            else ckmax(mx[ind][1],mx[2*ind+i][0]);
        }
        sum[ind] = sum[2*ind]+sum[2*ind+1];
    }
    void build(vi& a, int ind = 1, int L = 0, int R = -1) {
        if (R == -1) { R = (N = sz(a))-1; }
        if (L == R) {
            mx[ind][0] = sum[ind] = a[L];
            maxCnt[ind] = 1; mx[ind][1] = -1;
            return;
        }
        int M = (L+R)/2;
        build(a,2*ind,L,M); build(a,2*ind+1,M+1,R); pull(ind);
    }

    void push(int ind, int L, int R) {
        if (L == R) return;
        FOR(i,2)
            if (mx[2*ind^i][0] > mx[ind][0]) {
                sum[2*ind^i] -= (ll)maxCnt[2*ind^i]*
                    (mx[2*ind^i][0]-mx[ind][0]);
                mx[2*ind^i][0] = mx[ind][0];
            }
    }
    void upd(int x, int y, int t, int ind = 1, int L = 0, int R =
        ↪-1) {
        if (R == -1) R += N;
        if (R < x || y < L || mx[ind][0] <= t) return;
        push(ind,L,R);
        if (x <= L && R <= y && mx[ind][1] < t) {
            sum[ind] -= (ll)maxCnt[ind]*(mx[ind][0]-t);
            mx[ind][0] = t;
            return;
        }
        if (L == R) return;
        int M = (L+R)/2;
        upd(x,y,t,2*ind,L,M); upd(x,y,t,2*ind+1,M+1,R); pull(ind);
    }
    ll qsum(int x, int y, int ind = 1, int L = 0, int R = -1) {
        if (R == -1) R += N;
        if (R < x || y < L) return 0;
        push(ind,L,R);
        if (x <= L && R <= y) return sum[ind];
        int M = (L+R)/2;
        return qsum(x,y,2*ind,L,M)+qsum(x,y,2*ind+1,M+1,R);
    }
    int qmax(int x, int y, int ind = 1, int L = 0, int R = -1) {
        if (R == -1) R += N;
        if (R < x || y < L) return -1;
        push(ind,L,R);
        if (x <= L && R <= y) return mx[ind][0];
        int M = (L+R)/2;
        return max(qmax(x,y,2*ind,L,M), qmax(x,y,2*ind+1,M+1,R));
    }
};
```

PersSegTree.h
Description: persistent segtree with lazy updates, assumes that lazy[cur] is included in val[cur] before propagating cur
Time: $\mathcal{O}(\log N)$

```

60 lines
template<class T, int SZ> struct pseg {
    static const int LIMIT = 10000000; // adjust
    int l[LIMIT], r[LIMIT], nex = 0;
    T val[LIMIT], lazy[LIMIT];

    int copy(int cur) {
        int x = nex++;
```

```

    val[x] = val[cur], l[x] = l[cur], r[x] = r[cur], lazy[x] =
        ↪ lazy[cur];
    return x;
}
T comb(T a, T b) { return min(a,b); }
void pull(int x) { val[x] = comb(val[l[x]],val[r[x]]); }
void push(int cur, int L, int R) {
    if (!lazy[cur]) return;
    if (L != R) {
        l[cur] = copy(l[cur]);
        val[l[cur]] += lazy[cur];
        lazy[l[cur]] += lazy[cur];

        r[cur] = copy(r[cur]);
        val[r[cur]] += lazy[cur];
        lazy[r[cur]] += lazy[cur];
    }
    lazy[cur] = 0;
}

T query(int cur, int lo, int hi, int L, int R) {
    if (lo <= L && R <= hi) return val[cur];
    if (R < lo || hi < L) return INF;
    int M = (L+R)/2;
    return lazy[cur]+comb(query(l[cur],lo,hi,L,M), query(r[cur]
        ↪),lo,hi,M+1,R));
}

int upd(int cur, int lo, int hi, T v, int L, int R) {
    if (R < lo || hi < L) return cur;

    int x = copy(cur);
    if (lo <= L && R <= hi) { val[x] += v, lazy[x] += v; return
        ↪ x; }
    push(x,L,R);

    int M = (L+R)/2;
    l[x] = upd(l[x],lo,hi,v,L,M), r[x] = upd(r[x],lo,hi,v,M+1,R
        ↪);
    pull(x); return x;
}

int build(vector<T>& arr, int L, int R) {
    int cur = nex++;
    if (L == R) {
        if (L < sz(arr)) val[cur] = arr[L];
        return cur;
    }

    int M = (L+R)/2;
    l[cur] = build(arr,L,M), r[cur] = build(arr,M+1,R);
    pull(cur); return cur;
}

vi loc;
void upd(int lo, int hi, T v) { loc.pb(upd(loc.back(),lo,hi,v
    ↪,0,SZ-1)); }
T query(int ti, int lo, int hi) { return query(loc[ti],lo,hi
    ↪,0,SZ-1); }
void build(vector<T>& arr) { loc.pb(build(arr,0,SZ-1)); }
};

```

Treap.h

Description: easy BBST, use split and merge to implement insert and delete
Time: $\mathcal{O}(\log N)$

77 lines

```
typedef struct tnode* pt;
```

```

struct tnode {
    int pri, val; pt c[2]; // essential
    int sz; ll sum; // for range queries
}

```

```

bool flip; // lazy update

tnode (int _val) {
    pri = rand()+(rand()<<15); val = _val; c[0] = c[1] = NULL;
    sz = 1; sum = val;
    flip = 0;
}

};

int getsz(pt x) { return x?x->sz:0; }
ll getsum(pt x) { return x?x->sum:0; }

pt prop(pt x) {
    if (!x || !x->flip) return x;
    swap(x->c[0],x->c[1]);
    x->flip = 0;
    FOR(i,2) if (x->c[i]) x->c[i]->flip ^= 1;
    return x;
}

pt calc(pt x) {
    assert(!x->flip);
    prop(x->c[0]), prop(x->c[1]);
    x->sz = 1+getsz(x->c[0])+getsz(x->c[1]);
    x->sum = x->val+getsum(x->c[0])+getsum(x->c[1]);
    return x;
}

void tour(pt x, vi& v) {
    if (!x) return;
    prop(x);
    tour(x->c[0],v); v.pb(x->val); tour(x->c[1],v);
}

pair<pt,pt> split(pt t, int v) { // >= v goes to the right
    if (!t) return {t,t};
    prop(t);
    if (t->val >= v) {
        auto p = split(t->c[0], v); t->c[0] = p.s;
        return {p.f, calc(t)};
    } else {
        auto p = split(t->c[1], v); t->c[1] = p.f;
        return {calc(t), p.s};
    }
}

pair<pt,pt> splitsz(pt t, int sz) { // leftmost sz nodes go to
    ↪left
    if (!t) return {t,t};
    prop(t);
    if (getsz(t->c[0]) >= sz) {
        auto p = splitsz(t->c[0], sz); t->c[0] = p.s;
        return {p.f, calc(t)};
    } else {
        auto p = splitsz(t->c[1], sz-getsz(t->c[0])-1); t->c[1] = p
            ↪.f;
        return {calc(t), p.s};
    }
}

pt merge(pt l, pt r) {
    if (!l || !r) return l ? l : r;
    prop(l), prop(r);
    pt t;
    if (l->pri > r->pri) l->c[1] = merge(l->c[1],r), t = l;
    else r->c[0] = merge(l,r->c[0]), t = r;
    return calc(t);
}

pt ins(pt x, int v) { // insert v
    auto a = split(x,v), b = split(a.s,v+1);
    return merge(a.f,merge(new tnode(v),b.s));
}

```

```

}
pt del(pt x, int v) { // delete v
    auto a = split(x,v), b = split(a.s,v+1);
    return merge(a.f,b.s);
}

```

SqrtDecomp.h

Description: 1D point update, range query

Time: $\mathcal{O}(\sqrt{N})$

44 lines

```

struct sqrtDecomp {
    const static int blockSZ = 10; //change this
    int val[blockSZ*blockSZ];
    int lazy[blockSZ];

    sqrtDecomp() {
        M00(i, blockSZ*blockSZ) val[i] = 0;
        M00(i, blockSZ) lazy[i] = 0;
    }
    void upd(int l, int r, int v) {
        int ind = l;
        while(ind%blockSZ && ind <= r) {
            val[ind] += v;
            lazy[ind/blockSZ] += v;
            ind++;
        }
        while(ind + blockSZ <= r) {
            lazy[ind/blockSZ] += v*blockSZ;
            ind += blockSZ;
        }
        while(ind <= r) {
            val[ind] += v;
            lazy[ind/blockSZ] += v;
            ind++;
        }
    }
    int query(int l, int r) {
        int res = 0;
        int ind = l;
        while(ind%blockSZ && ind <= r) {
            res += val[ind];
            ind++;
        }
        while(ind + blockSZ <= r) {
            res += lazy[ind/blockSZ];
            ind += blockSZ;
        }
        while(ind <= r) {
            res += val[ind];
            ind++;
        }
        return res;
    }
};

```

Number Theory (4)

4.1 Modular Arithmetic

Modular.h

Description: modular arithmetic operations

41 lines

```

template<class T> struct modular {
    T val;
    explicit operator T() const { return val; }
    modular() { val = 0; }
    modular(const ll& v) {

```



```

    val = (-MOD <= v && v <= MOD) ? v : v % MOD;
    if (val < 0) val += MOD;
}

// friend ostream& operator<<(ostream& os, const modular& a)
//   ↪{ return os << a.val; }
friend void pr(const modular& a) { pr(a.val); }
friend void re(modular& a) { ll x; re(x); a = modular(x); }

friend bool operator==(const modular& a, const modular& b) {
    ↪return a.val == b.val; }
friend bool operator!=(const modular& a, const modular& b) {
    ↪return !(a == b); }
friend bool operator<(const modular& a, const modular& b) {
    ↪return a.val < b.val; }

modular operator-() const { return modular(-val); }
modular& operator+=(const modular& m) { if ((val += m.val) >=
    ↪MOD) val -= MOD; return *this; }
modular& operator=(const modular& m) { if ((val -= m.val) <
    ↪0) val += MOD; return *this; }
modular& operator*=(const modular& m) { val = (ll)val*m.val%
    ↪MOD; return *this; }
friend modular pow(modular a, ll p) {
    modular ans = 1; for (; p; p /= 2, a *= a) if (p&1) ans *=
    ↪a;
    return ans;
}
friend modular inv(const modular& a) {
    assert(a != 0); return exp(a,MOD-2);
}
modular& operator/=(const modular& m) { return (*this) *= inv
    ↪(m); }

friend modular operator+(modular a, const modular& b) {
    ↪return a += b; }
friend modular operator-(modular a, const modular& b) {
    ↪return a -= b; }
friend modular operator*(modular a, const modular& b) {
    ↪return a *= b; }

friend modular operator/(modular a, const modular& b) {
    ↪return a /= b; }
};

```

```

typedef modular<int> mi;
typedef pair<mi,mi> pmi;
typedef vector<mi> vmi;
typedef vector<pmi> vpmi;

```

ModFact.h

Description: pre-compute factorial mod inverses for MOD , assumes MOD is prime and $SZ < MOD$

Time: $\mathcal{O}(SZ)$

```

10 lines
vl inv, fac, ifac;
void genInv(int SZ) {
    inv.rsz(SZ), fac.rsz(SZ), ifac.rsz(SZ);
    inv[1] = 1; FOR(i,2,SZ) inv[i] = MOD-MOD/i*inv[MOD%i]%MOD;
    fac[0] = ifac[0] = 1;
    FOR(i,1,SZ) {
        fac[i] = fac[i-1]*i%MOD;
        ifac[i] = ifac[i-1]*inv[i]%MOD;
    }
}

```

ModMulLL.h

Description: multiply two 64-bit integers mod another if 128-bit is not available works for $0 \leq a, b < \text{mod} < 2^{63}$

```

14 lines
typedef unsigned long long ul;

// equivalent to (ul)(__int128(a)*b%mod)
ul modMul(ul a, ul b, const ul mod) {
    ll ret = a*b-mod*(ul)((ld)a*b/mod);
    return ret+((ret<0)-(ret>=(ll)mod))*mod;
}
ul modPow(ul a, ul b, const ul mod) {
    if (b == 0) return 1;
    ul res = modPow(a,b/2,mod);
    res = modMul(res,res,mod);
    if (b&1) return modMul(res,a,mod);
    return res;
}

```

ModSqrt.h

Description: find sqrt of integer mod a prime

Time: ?

```

26 lines
"Modular.h"
template<class T> T sqrt(modular<T> a) {
    auto p = pow(a, (MOD-1)/2); if (p != 1) return p == 0 ? 0 :
    ↪-1; // check if zero or does not have sqrt
    T s = MOD-1, e = 0; while (s % 2 == 0) s /= 2, e++;
    modular<T> n = 1; while (pow(n, (MOD-1)/2) == 1) n = (T)(n)+1;
    ↪ // find non-square residue

    auto x = pow(a, (s+1)/2), b = pow(a, s), g = pow(n, s);
    int r = e;
    while (1) {
        auto B = b; int m = 0; while (B != 1) B *= B, m++;
        if (m == 0) return min((T)x, MOD-(T)x);
        FOR(i, r-m-1) g *= g;
        x *= g; g *= g; b *= g; r = m;
    }

/* Explanation:
 * Initially, x^2=ab, ord(b) = 2^m, ord(g) = 2^r where m<r
 * g = g^{2^{r-m-1}} -> ord(g) = 2^{m+1}
 * if x'=x*g, then b' = b*g^2
    (b')^{2^{m-1}} = (b*g^2)^{2^{m-1}}
    = b^{2^{m-1}}*g^{2^m}
    = -1*-1
    = 1
    -> ord(b')|ord(b)/2
 * m decreases by at least one each iteration
 */

```

ModSum.h

Description: Sums of mod'ed arithmetic progressions

```

15 lines
typedef unsigned long long ul;

ul sumsq(ul to) { return (to-1)*to/2; } // sum of 0..to-1

ul divsum(ul to, ul c, ul k, ul m) { // sum_{i=0}^{to-1} floor((
    ↪ki+c)/m)
    ul res = k/m*sumsq(to)+c/m*to;
    k %= m; c %= m; if (!k) return res;
    ul to2 = (to*k+c)/m;
    return res+(to-1)*to2-divsum(to2,m-1-c,m,k);
}

ll modsum(ul to, ll c, ll k, ll m) {

```

```

    c = (c%m+m)%m, k = (k%m+m)%m;
    return to*c+k*sumsq(to)-m*divsum(to,c,k,m);
}

```

4.2 Primality

PrimeSieve.h

Description: tests primality up to SZ

Time: $\mathcal{O}(SZ \log \log SZ)$

```

11 lines
template<int SZ> struct Sieve {
    bitset<SZ> isprime;
    vi pr;
    Sieve() {
        isprime.set(); isprime[0] = isprime[1] = 0;
        for (int i = 4; i < SZ; i += 2) isprime[i] = 0;
        for (int i = 3; i*i < SZ; i += 2) if (isprime[i])
            for (int j = i*i; j < SZ; j += i*2) isprime[j] = 0;
        FOR(i,2,SZ) if (isprime[i]) pr.pb(i);
    }
};

```

FactorFast.h

Description: Factors integers up to 2^{60}

Time: ?

```

46 lines
"PrimeSieve.h"
Sieve<1<<20> S = Sieve<1<<20>(); // should take care of all
    ↪primes up to n^(1/3)

bool millerRabin(ll p) { // test primality
    if (p == 2) return true;
    if (p == 1 || p % 2 == 0) return false;
    ll s = p - 1; while (s % 2 == 0) s /= 2;
    FOR(i,30) { // strong liar with probability <= 1/4
        ll a = rand() % (p - 1) + 1, tmp = s;
        ll mod = mod_pow(a, tmp, p);
        while (tmp != p - 1 && mod != 1 && mod != p - 1) {
            mod = mod_mul(mod, mod, p);
            tmp *= 2;
        }
        if (mod != p - 1 && tmp % 2 == 0) return false;
    }
    return true;
}

ll f(ll a, ll n, ll &has) { return (mod_mul(a, a, n) + has) % n
    ↪; }

```

```

vpl pollardsRho(ll d) {
    vpl res;
    auto& pr = S.pr;
    for (int i = 0; i < sz(pr) && pr[i]*pr[i] <= d; i++) if (d %
    ↪pr[i] == 0) {
        int co = 0; while (d % pr[i] == 0) d /= pr[i], co++;
        res.pb({pr[i],co});
    }
    if (d > 1) { // d is now a product of at most 2 primes.
        if (millerRabin(d)) res.pb({d,1});
        else while (1) {
            ll has = rand() % 2321 + 47;
            ll x = 2, y = 2, c = 1;
            for (; c == 1; c = __gcd(abs(x-y), d)) {
                x = f(x, d, has);
                y = f(f(y, d, has), d, has);
            } // should cycle in ~sqrt(smallest nontrivial divisor)
            ↪turns
            if (c != d) {
                d /= c; if (d > c) swap(d,c);
                if (c == d) res.pb({c,2});
            }
        }
    }
}

```

```
        else res.pb({c,1}), res.pb({d,1});
        break;
    }
}
return res;
}
```

4.3 Divisibility

Euclid.h

Description: Euclidean Algorithm 9 lines

```
pl euclid(ll a, ll b) { // returns {x,y} such that a*x+b*y=gcd(
    ↪a,b)
    if (!b) return {1,0};
    pl p = euclid(b,a%b);
    return {p.s,p.f-a/b*p.s};
}

ll invGeneral(ll a, ll b) {
    pl p = euclid(a,b); assert(p.f*a+p.s*b == 1);
    return p.f+(p.f<0)*b;
}
```

CRT.h

Description: Chinese Remainder Theorem 7 lines

"Euclid.h"

```
pl solve(pl a, pl b) {
    auto g = __gcd(a.s,b.s), l = a.s/g*b.s;
    if ((b.f-a.f) % g != 0) return {-1,-1};
    auto A = a.s/g, B = b.s/g;
    auto mul = (b.f-a.f)/g*invGeneral(A,B) % B;
    return {(mul*a.s+a.f)%l+1}%l,1};
}
```

Combinatorial (5)

IntPerm.h

Description: convert permutation of {0,1,...,N-1} to integer in [0,N!]

Usage: assert (encode (decode (5,37)) == 37);

Time: $\mathcal{O}(N)$ 20 lines

```
vi decode(int n, int a) {
    vi el(n), b; iota(all(el),0);
    FOR(i,n) {
        int z = a%sz(el);
        b.pb(el[z]); a /= sz(el);
        swap(el[z],el.back()); el.pop_back();
    }
    return b;
}

int encode(vi b) {
    int n = sz(b), a = 0, mul = 1;
    vi pos(n); iota(all(pos),0); vi el = pos;
    FOR(i,n) {
        int z = pos[b[i]]; a += mul*z; mul *= sz(el);
        swap(pos[el[z]],pos[el.back()]);
        swap(el[z],el.back()); el.pop_back();
    }
    return a;
}
```

MatroidIntersect.h

Description: computes a set of maximum size which is independent in both graphic and colorful matroids, aka a spanning forest where no two edges are of the same color

Time: $\mathcal{O}(GI^{1.5})$ calls to oracles, where G is the size of the ground set and I is the size of the independent set 108 lines

"DSU.h"

```
int R;
map<int,int> m;

struct Element {
    pi ed;
    int col;
    bool in_independent_set = 0;
    int independent_set_position;
    Element(int u, int v, int c) { ed = {u,v}; col = c; }
};

vi independent_set;
vector<Element> ground_set;
bool col_used[300];

struct GBasis {
    DSU D;
    void reset() { D.init(sz(m)); }
    void add(pi v) { assert(D.unite(v.f,v.s)); }
    bool independent_with(pi v) { return !D.sameSet(v.f,v.s); }
};

GBasis basis, basis_wo[300];

bool graph_oracle(int inserted) {
    return basis.independent_with(ground_set[inserted].ed);
}

bool graph_oracle(int inserted, int removed) {
    int wi = ground_set[removed].independent_set_position;
    return basis_wo[wi].independent_with(ground_set[inserted].ed)
    ↪;
}

void prepare_graph_oracle() {
    basis.reset();
    FOR(i,sz(independent_set)) basis_wo[i].reset();
    FOR(i,sz(independent_set)) {
        pi v = ground_set[independent_set[i]].ed; basis.add(v);
        FOR(j,sz(independent_set)) if (i != j) basis_wo[j].add(v);
    }
}

bool colorful_oracle(int ins) {
    ins = ground_set[ins].col;
    return !col_used[ins];
}

bool colorful_oracle(int ins, int rem) {
    ins = ground_set[ins].col;
    rem = ground_set[rem].col;
    return !col_used[ins] || ins == rem;
}

void prepare_colorful_oracle() {
    FOR(i,R) col_used[i] = 0;
    trav(t,independent_set) col_used[ground_set[t].col] = 1;
}

bool augment() {
    prepare_graph_oracle();
    prepare_colorful_oracle();

    vi par(sz(ground_set),MOD);
    queue<int> q;
    FOR(i,sz(ground_set)) if (colorful_oracle(i)) {
        assert(!ground_set[i].in_independent_set);
        par[i] = -1; q.push(i);
    }

    int lst = -1;
```

```
while (sz(q)) {
    int cur = q.front(); q.pop();
    if (ground_set[cur].in_independent_set) {
        FOR(to,sz(ground_set)) if (par[to] == MOD) {
            if (!colorful_oracle(to,cur)) continue;
            par[to] = cur; q.push(to);
        }
    } else {
        if (graph_oracle(cur)) { lst = cur; break; }
        trav(to,independent_set) if (par[to] == MOD) {
            if (!graph_oracle(cur,to)) continue;
            par[to] = cur; q.push(to);
        }
    }
}

if (lst == -1) return 0;
do {
    ground_set[lst].in_independent_set ^= 1;
    lst = par[lst];
} while (lst != -1);
independent_set.clear();
FOR(i,sz(ground_set)) if (ground_set[i].in_independent_set) {
    ground_set[i].independent_set_position = sz(independent_set
    ↪);
    independent_set.pb(i);
}
return 1;
}

void solve() {
    re(R); if (R == 0) exit(0);
    m.clear(); ground_set.clear(); independent_set.clear();
    FOR(i,R) {
        int a,b,c,d; re(a,b,c,d);
        ground_set.pb(Element(a,b,i));
        ground_set.pb(Element(c,d,i));
        m[a] = m[b] = m[c] = m[d] = 0;
    }
    int co = 0;
    trav(t,m) t.s = co++;
    trav(t,ground_set) t.ed.f = m[t.ed.f], t.ed.s = m[t.ed.s];
    while (augment());
    ps(2*sz(independent_set));
}
```

PermGroup.h

Description: Schreier-Sims, count number of permutations in group and test whether permutation is a member of group

Time: ? 51 lines

```
const int N = 15;
int n;

vi inv(vi v) { vi V(sz(v)); FOR(i,sz(v)) V[v[i]] = i; return V;
    ↪ }

vi id() { vi v(n); iota(all(v),0); return v; }
vi operator*(const vi& a, const vi& b) {
    vi c(sz(a)); FOR(i,sz(a)) c[i] = a[b[i]];
    return c;
}

struct Group {
    bool flag[N];
    vi sigma[N]; // sigma[t][k] = t, sigma[t][x] = x if x > k
    vector<vi> gen;
    void clear(int p) {
        memset(flag,0, sizeof flag);
        flag[p] = 1; sigma[p] = id();
        gen.clear();
    }
}
```



```
    }
} g[N];

bool check(const vi& cur, int k) {
    if (!k) return 1;
    int t = cur[k];
    return g[k].flag[t] ? check(inv(g[k].sigma[t])*cur,k-1) : 0;
}

void updateX(const vi& cur, int k);
void ins(const vi& cur, int k) {
    if (check(cur,k)) return;
    g[k].gen.pb(cur);
    FOR(i,n) if (g[k].flag[i]) updateX(cur*g[k].sigma[i],k);
}

void updateX(const vi& cur, int k) {
    int t = cur[k];
    if (g[k].flag[t]) ins(inv(g[k].sigma[t])*cur,k-1); // fixes k
    ↪ -> k
    else {
        g[k].flag[t] = 1, g[k].sigma[t] = cur;
        trav(x,g[k].gen) updateX(x*cur,k);
    }
}

ll order(vector<vi> gen) {
    assert(sz(gen)); n = sz(gen[0]); FOR(i,n) g[i].clear(i);
    trav(a,gen) ins(a,n-1); // insert perms into group one by one
    ll tot = 1;
    FOR(i,n) {
        int cnt = 0; FOR(j,i+1) cnt += g[i].flag[j];
        tot *= cnt;
    }
    return tot;
}
```

Numerical (6)

6.1 Matrix

Matrix.h

Description: 2D matrix operations36 lines

```
template<class T> struct Mat {
    int r,c;
    vector<vector<T>> d;
    Mat(int _r, int _c) : r(_r), c(_c) { d.assign(r,vector<T>(c))
    ↪; }
    Mat() : Mat(0,0) {}
    Mat(const vector<vector<T>>& _d) : r(sz(_d)), c(sz(_d[0])) {
    ↪d = _d; }
    friend void pr(const Mat& m) { pr(m.d); }

    Mat& operator+=(const Mat& m) {
        assert(r == m.r && c == m.c);
        FOR(i,r) FOR(j,c) d[i][j] += m.d[i][j];
        return *this;
    }
    Mat& operator-=(const Mat& m) {
        assert(r == m.r && c == m.c);
        FOR(i,r) FOR(j,c) d[i][j] -= m.d[i][j];
        return *this;
    }
    Mat operator*(const Mat& m) {
        assert(c == m.r); Mat x(r,m.c);
        FOR(i,r) FOR(j,c) FOR(k,m.c) x.d[i][k] += d[i][j]*m.d[j][k]
    ↪;
        return x;
    }
}
```

```
Mat operator+(const Mat& m) { return Mat(*this)+=m; }
Mat operator-(const Mat& m) { return Mat(*this)-=m; }
Mat& operator*=(const Mat& m) { return *this = (*this)*m; }

friend Mat pow(Mat m, ll p) {
    assert(m.r == m.c);
    Mat r(m.r,m.c);
    FOR(i,m.r) r.d[i][i] = 1;
    for (; p; p /= 2, m *= m) if (p&1) r *= m;
    return r;
}

};

MatrixInv.h
Description: calculates determinant via gaussian elimination
Time:  $\mathcal{O}(N^3)$ 
"Matrix.h"31 lines
template<class T> T gauss(Mat<T>& m) { // determinant of 1000
    ↪x1000 Matrix in ~1s
    int n = m.r;
    T prod = 1; int nex = 0;
    FOR(i,n) {
        int row = -1; // for 1d use EPS rather than 0
        FOR(j,nex,n) if (m.d[j][i] != 0) { row = j; break; }
        if (row == -1) { prod = 0; continue; }
        if (row != nex) prod *= -1, swap(m.d[row],m.d[nex]);
        prod *= m.d[nex][i];
        auto x = 1/m.d[nex][i]; FOR(k,i,m.c) m.d[nex][k] *= x;
        FOR(j,n) if (j != nex) {
            auto v = m.d[j][i];
            if (v != 0) FOR(k,i,m.c) m.d[j][k] -= v*m.d[nex][k];
        }
        nex ++;
    }
    return prod;
}
```

```
template<class T> Mat<T> inv(Mat<T> m) {
    int n = m.r;
    Mat<T> x(n,2*n);
    FOR(i,n) {
        x.d[i][i+n] = 1;
        FOR(j,n) x.d[i][j] = m.d[i][j];
    }
    if (gauss(x) == 0) return Mat<T>(0,0);
    Mat<T> r(n,n);
    FOR(i,n) FOR(j,n) r.d[i][j] = x.d[i][j+n];
    return r;
}
```

MatrixTree.h

Description: Kirchhoff's Matrix Tree Theorem: given adjacency matrix, calculates # of spanning trees13 lines

```
mi numSpan(Mat<mi> m) {
    int n = m.r;
    Mat<mi> res(n-1,n-1);
    FOR(i,n) FOR(j,i+1,n) {
        mi ed = m.d[i][j];
        res.d[i][i] += ed;
        if (j != n-1) {
            res.d[j][j] += ed;
            res.d[i][j] -= ed, res.d[j][i] -= ed;
        }
    }
    return gauss(res);
}
```

```
};

6.2 Polynomials
VecOp.h
Description: arithmetic + misc polynomial operations with vectors73 lines
namespace VecOp {
    template<class T> vector<T> rev(vector<T> v) { reverse(all(v)
    ↪); return v; }
    template<class T> vector<T> shift(vector<T> v, int x) { v.
    ↪insert(v.begin(),x,0); return v; }
    template<class T> vector<T> integ(const vector<T>& v) {
        vector<T> res(sz(v)+1);
        FOR(i,sz(v)) res[i+1] = v[i]/(i+1);
        return res;
    }
    template<class T> vector<T> dif(const vector<T>& v) {
        if (!sz(v)) return v;
        vector<T> res(sz(v)-1); FOR(i,1,sz(v)) res[i-1] = i*v[i];
        return res;
    }
    template<class T> vector<T>& remLead(vector<T>& v) {
        while (sz(v) && v.back() == 0) v.pop_back();
        return v;
    }
    template<class T> T eval(const vector<T>& v, const T& x) {
        T res = 0; R0F(i,sz(v)) res = x*res+v[i];
        return res;
    }

    template<class T> vector<T>& operator+=(vector<T>& l, const
    ↪vector<T>& r) {
        l.rsz(max(sz(l),sz(r))); FOR(i,sz(r)) l[i] += r[i]; return
    ↪l;
    }
    template<class T> vector<T>& operator-=(vector<T>& l, const
    ↪vector<T>& r) {
        l.rsz(max(sz(l),sz(r))); FOR(i,sz(r)) l[i] -= r[i]; return
    ↪l;
    }
    template<class T> vector<T>& operator*=(vector<T>& l, const T
    ↪& r) { trav(t,l) t *= r; return l; }
    template<class T> vector<T>& operator/=(vector<T>& l, const T
    ↪& r) { trav(t,l) t /= r; return l; }

    template<class T> vector<T> operator+(vector<T> l, const
    ↪vector<T>& r) { return l += r; }
    template<class T> vector<T> operator-(vector<T> l, const
    ↪vector<T>& r) { return l -= r; }
    template<class T> vector<T> operator*(vector<T> l, const T& r
    ↪) { return l *= r; }
    template<class T> vector<T> operator*(const T& r, const
    ↪vector<T>& l) { return l*r; }
    template<class T> vector<T> operator/(vector<T> l, const T& r
    ↪) { return l /= r; }

    template<class T> vector<T> operator*(const vector<T>& l,
    ↪const vector<T>& r) {
        if (min(sz(l),sz(r)) == 0) return {};
        vector<T> x(sz(l)+sz(r)-1); FOR(i,sz(l)) FOR(j,sz(r)) x[i+j]
    ↪ += l[i]*r[j];
        return x;
    }
    template<class T> vector<T>& operator*=(vector<T>& l, const
    ↪vector<T>& r) { return l = l*r; }

    template<class T> pair<vector<T>,vector<T>> qr(vector<T> a,
    ↪vector<T> b) { // quotient and remainder

```

```
    assert(sz(b)); auto B = b.back(); assert(B != 0);
    B = 1/B; trav(t,b) t *= B;

    remLead(a); vector<T> q(max(sz(a)-sz(b)+1,0));
    while (sz(a) >= sz(b)) {
        q[sz(a)-sz(b)] = a.back();
        a -= a.back()*shift(b,sz(a)-sz(b));
        remLead(a);
    }

    trav(t,q) t *= B;
    return {q,a};
}

template<class T> vector<T> quo(const vector<T>& a, const
    ↪vector<T>& b) { return qr(a,b).f; }
template<class T> vector<T> rem(const vector<T>& a, const
    ↪vector<T>& b) { return qr(a,b).s; }

template<class T> vector<T> interpolate(vector<pair<T,T>> v)
    ↪{
    vector<T> ret, prod = {1};
    FOR(i,sz(v)) prod *= vector<T>({-v[i].f,1});
    FOR(i,sz(v)) {
        T todiv = 1; FOR(j,sz(v)) if (i != j) todiv *= v[j].f-v[j]
            ↪.f;
        ret += qr(prod,{-v[i].f,1}).f*v[i].s/todiv;
    }
    return ret;
}

using namespace VecOp;
```

PolyRoots.h

Description: Finds the real roots of a polynomial.
Usage: poly_roots({{2,-3,1}},-1e9,1e9) // solve x²-3x+2 = 0
Time: $\mathcal{O}(N^2 \log(1/\epsilon))$

```
"VecOp.h" 19 lines
vd polyRoots(vd p, ld xmin, ld xmax) {
    if (sz(p) == 2) { return {-p[0]/p[1]}; }
    auto dr = polyRoots(dif(p),xmin,xmax);
    dr.pb(xmin-1); dr.pb(xmax+1); sort(all(dr));
    vd ret;
    FOR(i,sz(dr)-1) {
        auto l = dr[i], h = dr[i+1];
        bool sign = eval(p,l) > 0;
        if (sign ^ (eval(p,h) > 0)) {
            FOR(it,60) { // while (h - l > 1e-8)
                auto m = (l+h)/2, f = eval(p,m);
                if ((f <= 0) ^ sign) l = m;
                else h = m;
            }
            ret.pb((l+h)/2);
        }
    }
    return ret;
}
```

Karatsuba.h

Description: multiply two polynomials
Time: $\mathcal{O}(N^{\log_2 3})$

```
int size(int s) { return s > 1 ? 32-__builtin_clz(s-1) : 0; }

void karatsuba(ll *a, ll *b, ll *c, ll *t, int n) {
    int ca = 0, cb = 0; FOR(i,n) ca += !!a[i], cb += !!b[i];
    if (min(ca, cb) <= 1500/n) { // few numbers to multiply
        if (ca > cb) swap(a, b);
```

```
        FOR(i,n) if (a[i]) FOR(j,n) c[i+j] += a[i]*b[j];
    } else {
        int h = n >> 1;
        karatsuba(a, b, c, t, h); // a0*b0
        karatsuba(a+h, b+h, c+n, t, h); // a1*b1
        FOR(i,h) a[i] += a[i+h], b[i] += b[i+h];
        karatsuba(a, b, t, t+n, h); // (a0+a1)*(b0+b1)
        FOR(i,h) a[i] -= a[i+h], b[i] -= b[i+h];
        FOR(i,n) t[i] -= c[i]+c[i+n];
        FOR(i,n) c[i+h] += t[i], t[i] = 0;
    }
}

vl conv(vl a, vl b) {
    int sa = sz(a), sb = sz(b); if (!sa || !sb) return {};
    int n = 1<<size(max(sa,sb)); a.rsz(n), b.rsz(n);
    vl c(2*n), t(2*n); FOR(i,2*n) t[i] = 0;
    karatsuba(&a[0], &b[0], &c[0], &t[0], n);
    c.rsz(sa+sb-1); return c;
}
```

FFT.h

Description: multiply two polynomials
Time: $\mathcal{O}(N \log N)$

```
"Modular.h" 40 lines
typedef complex<db> cd;
const int MOD = (119 << 23) + 1, root = 3; // = 998244353
// NTT: For p < 2^30 there is also e.g. (5 << 25, 3), (7 << 26,
    ↪3),
// (479 << 21, 3) and (483 << 21, 5). The last two are > 10^9.

constexpr int size(int s) { return s > 1 ? 32-__builtin_clz(s
    ↪-1) : 0; }
void genRoots(vmi& roots) { // primitive n-th roots of unity
    int n = sz(roots); mi r = pow(mi(root), (MOD-1)/n);
    roots[0] = 1; FOR(i,1,n) roots[i] = roots[i-1]*r;
}
void genRoots(vcd& roots) { // change cd to complex<double>
    ↪instead?
    int n = sz(roots); double ang = 2*PI/n;
    FOR(i,n) roots[i] = cd(cos(ang*i),sin(ang*i)); // is there a
        ↪way to do this more quickly?
}
```

```
template<class T> void fft(vector<T>& a, const vector<T>& roots
    ↪, bool inv = 0) {
    int n = sz(a);
    for (int i = 1, j = 0; i < n; i++) { // sort by reverse bit
        ↪representation
        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)
        for (int i = 0; i < n; i += len)
            FOR(j,len/2) {
                int ind = n/len*j; if (inv && ind) ind = n-ind;
                auto u = a[i+j], v = a[i+j+len/2]*roots[ind];
                a[i+j] = u+v, a[i+j+len/2] = u-v;
            }
    if (inv) { T i = T(1)/T(n); trav(x,a) x *= i; }
```

```
template<class T> vector<T> mult(vector<T> a, vector<T> b) {
    int s = sz(a)+sz(b)-1, n = 1<<size(s);
    vector<T> roots(n); genRoots(roots);
    a.rsz(n), fft(a,roots);
    b.rsz(n), fft(b,roots);
```

```
    FOR(i,n) a[i] *= b[i];
    fft(a,roots,1); return a;
}
```

FFTmod.h

Description: multiply two polynomials with arbitrary MOD ensures precision by splitting in half

```
"FFT.h" 27 lines
vl multMod(const vl& a, const vl& b) {
    if (!min(sz(a),sz(b))) return {};
    int s = sz(a)+sz(b)-1, n = 1<<size(s), cut = sqrt(MOD);
    vcd roots(n); genRoots(roots);

    vcd ax(n), bx(n);
    FOR(i,sz(a)) ax[i] = cd((int)a[i]/cut, (int)a[i]%cut); // ax(
        ↪x)=a1(x)+i*a0(x)
    FOR(i,sz(b)) bx[i] = cd((int)b[i]/cut, (int)b[i]%cut); // bx(
        ↪x)=b1(x)+i*b0(x)
    fft(ax,roots), fft(bx,roots);

    vcd v1(n), v0(n);
    FOR(i,n) {
        int j = (i ? (n-i) : i);
        v1[i] = (ax[i]+conj(ax[j]))*cd(0.5,0)*bx[i]; // v1 = a1*b1
            ↪+b0*cd(0,1));
        v0[i] = (ax[i]-conj(ax[j]))*cd(0,-0.5)*bx[i]; // v0 = a0*(
            ↪b1+b0*cd(0,1));
    }
    fft(v1,roots,1), fft(v0,roots,1);

    vl ret(n);
    FOR(i,n) {
        ll V2 = (ll)round(v1[i].real()); // a1*b1
        ll V1 = (ll)round(v1[i].imag())+(ll)round(v0[i].real()); //
            ↪a0*b1+a1*b0
        ll V0 = (ll)round(v0[i].imag()); // a0*b0
        ret[i] = ((V2%MOD*cut+V1)%MOD*cut+V0)%MOD;
    }
    ret.rsz(s); return ret;
} // ~0.8s when sz(a)=sz(b)=1<<19
```

PolyInv.h

Description: ?
Time: ?

```
"FFT.h" 11 lines
template<class T> vector<T> inv(vector<T> v, int p) { //
    ↪compute inverse of v mod x^p, where v[0] = 1
    v.rsz(p); vector<T> a = {T(1)/v[0]};
    for (int i = 1; i < p; i *= 2) {
        if (2*i > p) v.rsz(2*i);
        auto l = vector<T>(begin(v),begin(v)+i), r = vector<T>(
            ↪begin(v)+i,begin(v)+2*i);
        auto c = mult(a,l); c = vector<T>(begin(c)+i,end(c));
        auto b = mult(a*T(-1),mult(a,r)+c); b.rsz(i);
        a.insert(end(a),all(b));
    }
    a.rsz(p); return a;
}
```

PolyDiv.h

Description: divide two polynomials
Time: $\mathcal{O}(N \log N)$?

```
"PolyInv.h" 7 lines
template<class T> pair<vector<T>,vector<T>> divi(const vector<T>
    ↪& f, const vector<T>& g) { // f = q*g+r
    if (sz(f) < sz(g)) return {{},f};
    auto q = mult(inv(rev(g),sz(f)-sz(g)+1),rev(f));
```

```
q.rsz(sz(f)-sz(g)+1); q = rev(q);
auto r = f-mult(q,g); r.rsz(sz(g)-1);
return {q,r};
}
```

PolySqrt.h
Description: find sqrt of polynomial
Time: $\mathcal{O}(N \log N)$?

"PolyInv.h"8 lines

```
template<class T> vector<T> sqrt(vector<T> v, int p) { // S*S =
    ↪ v mod x^p, p is power of 2
    assert(v[0] == 1); if (p == 1) return {1};
    v.rsz(p);
    auto S = sqrt(v,p/2);
    auto ans = S+mult(v,inv(S,p));
    ans.rsz(p); ans *= T(1)/T(2);
    return ans;
}
```

6.3 Misc

LinRec.h
Description: Berlekamp-Massey: computes linear recurrence of order n for sequence of $2n$ terms
Time: ?

using namespace vecOp;

struct LinRec {

vmi x; // original sequence

vmi C, rC;

void init(const vmi& _x) {

x = _x; int n = sz(x), m = 0;

vmi B; B = C = {1}; // B is fail vector

mi b = 1; // B gives 0,0,0,...,b

FOR(i,n) {

m ++;

mi d = x[i]; FOR(j,1,sz(C)) d += C[j]*x[i-j];

if (d == 0) continue; // recurrence still works

auto _B = C; C.rsz(max(sz(C),m+sz(B)));

mi coef = d/b; FOR(j,m,m+sz(B)) C[j] -= coef*B[j-m]; //

↪recurrence that gives 0,0,0,...,d

if (sz(_B) < m+sz(B)) { B = _B; b = d; m = 0; }

}

rC = C; reverse(all(rC)); // polynomial for getPo

C.erase(begin(C)); trav(t,C) t *= -1; // x[i]=sum_{j=0}^{sz

↪(C)-1}C[j]*x[i-j-1]

}

vmi getPo(int n) {

if (n == 0) return {1};

vmi x = getPo(n/2); x = rem(x*x,rC);

if (n&1) { vmi v = {0,1}; x = rem(x*v,rC); }

return x;

}

mi eval(int n) {

vmi t = getPo(n);

mi ans = 0; FOR(i,sz(t)) ans += t[i]*x[i];

return ans;

}

};

Integrate.h
Description: ?

8 lines

```
// db f(db x) { return x*x+3*x+1; }
```

```
db quad(db (*f)(db), db a, db b) {
    const int n = 1000;
    db dif = (b-a)/2/n, tot = f(a)+f(b);
    FOR(i,1,2*n) tot += f(a+i*dif)*(i&1?4:2);
    return tot*dif/3;
}
```

IntegrateAdaptive.h
Description: ?

19 lines

```
// db f(db x) { return x*x+3*x+1; }
```

```
db simpson(db (*f)(db), db a, db b) {
    db c = (a+b) / 2;
    return (f(a) + 4*f(c) + f(b)) * (b-a) / 6;
}
```

```
db rec(db (*f)(db), db a, db b, db eps, db S) {
    db c = (a+b) / 2;
    db S1 = simpson(f, a, c);
    db S2 = simpson(f, c, b), T = S1 + S2;
    if (abs(T - S) <= 15*eps || b-a < 1e-10)
        return T + (T - S) / 15;
    return rec(f, a, c, eps/2, S1) + rec(f, c, b, eps/2, S2);
}
```

```
db quad(db (*f)(db), db a, db b, db eps = 1e-8) {
    return rec(f, a, b, eps, simpson(f, a, b));
}
```

Simplex.h
Description: Simplex algorithm for linear programming, maximize $c^T x$ sub-ject to $Ax \leq b, x \geq 0$
Time: ?

73 lines

```
typedef double T;
typedef vector<T> vd;
typedef vector<vd> vvd;
const T eps = 1e-8, inf = 1/.0;
```

```
#define ltj(X) if (s == -1 || mp(X[j],N[j]) < mp(X[s],N[s])) s =
    ↪j
```

```
struct LPSolver {
    int m, n;
    vi N, B;
    vvd D;
```

```
    LPSolver(const vvd& A, const vd& b, const vd& c) :
        m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
        FOR(i,m) FOR(j,n) D[i][j] = A[i][j];
        FOR(i,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
            ↪// B[i] -> basic variables, col n+1 is for constants
            ↪, why D[i][n]=-1?
        FOR(j,n) { N[j] = j; D[m][j] = -c[j]; } // N[j] -> non-
            ↪basic variables, all zero
        N[n] = -1; D[m+1][n] = 1;
    }
```

```
void print() {
    ps("D");
    trav(t,D) ps(t);
    ps();
    ps("B",B);
    ps("N",N);
    ps();
}
```

```
void pivot(int r, int s) { // row, column
```

```
T *a = D[r].data(), inv = 1/a[s]; // eliminate col s from
    ↪consideration
FOR(i,m+2) if (i != r && abs(D[i][s]) > eps) {
    T *b = D[i].data(), inv2 = b[s]*inv;
    FOR(j,n+2) b[j] -= a[j]*inv2;
    b[s] = a[s]*inv2;
}
FOR(j,n+2) if (j != s) D[r][j] *= inv;
FOR(i,m+2) if (i != r) D[i][s] *= -inv;
D[r][s] = inv; swap(B[r], N[s]); // swap a basic and non-
    ↪basic variable
}
```

```
bool simplex(int phase) {
    int x = m+phase-1;
    for (;;) {
        int s = -1; FOR(j,n+1) if (N[j] != -phase) ltj(D[x]); //
            ↪find most negative col
        if (D[x][s] >= -eps) return true; // have best solution
        int r = -1;
        FOR(i,m) {
            if (D[i][s] <= eps) continue;
            if (r == -1 || mp(D[i][n+1] / D[i][s], B[i])
                < mp(D[r][n+1] / D[r][s], B[r])) r = i; // find
                ↪smallest positive ratio
        }
        if (r == -1) return false; // unbounded
        pivot(r, s);
    }
}
```

```
T solve(vd &x) {
    int r = 0; FOR(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) { // x=0 is not a solution
        pivot(r, n); // -1 is artificial variable, initially set
            ↪to smth large but want to get to 0
        if (!simplex(2) || D[m+1][n+1] < -eps) return -inf; // no
            ↪solution
        // D[m+1][n+1] is max possible value of the negation of
            ↪artificial variable, starts negative but should get
            ↪to zero
        FOR(i,m) if (B[i] == -1) {
            int s = 0; FOR(j,1,n+1) ltj(D[i]);
            pivot(i,s);
        }
    }
    bool ok = simplex(1); x = vd(n);
    FOR(i,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
};
```

Graphs (7)

7.1 Fundamentals

DSU.h
Description: ?
Time: $\mathcal{O}(N\alpha(N))$

29 lines

```
template<int SZ> struct DSU {
    int par[SZ];
    int size[SZ];
    DSU() {
        M00(i, SZ) par[i] = i, size[i] = 1;
    }
    int get(int node) {
        if(par[node] != node) par[node] = get(par[node]);
```

```

        return par[node];
    }
    bool connected(int n1, int n2) {
        return (get(n1) == get(n2));
    }
    int sz(int node) {
        return size[get(node)];
    }
    void unite(int n1, int n2) {
        n1 = get(n1);
        n2 = get(n2);
        if(n1 == n2) return;
        if(rand()%2) {
            par[n1] = n2;
            size[n2] += size[n1];
        } else {
            par[n2] = n1;
            size[n1] += size[n2];
        }
    }
};

```

ManhattanMST.h

Description: Compute minimum spanning tree of points where edges are manhattan distances

Time: $\mathcal{O}(N \log N)$

"MST.h" 60 lines

```

int N;
vector<array<int,3>> cur;
vector<pair<ll,pi>> ed;
vi ind;

struct {
    map<int,pi> m;
    void upd(int a, pi b) {
        auto it = m.lb(a);
        if (it != m.end() && it->s <= b) return;
        m[a] = b; it = m.find(a);
        while (it != m.begin() && prev(it)->s >= b) m.erase(prev(it)
            ⇨));
    }
    pi query(int y) { // for all a > y find min possible value of
        ⇨ b
        auto it = m.ub(y);
        if (it == m.end()) return {2*MOD,2*MOD};
        return it->s;
    }
} S;

void solve() {
    sort(all(ind),[](int a, int b) { return cur[a][0] > cur[b]
        ⇨][0]; });
    S.m.clear();
    int nex = 0;
    trav(x,ind) { // cur[x][0] <= ?, cur[x][1] < ?
        while (nex < N && cur[ind[nex]][0] >= cur[x][0]) {
            int b = ind[nex++];
            S.upd(cur[b][1],{cur[b][2],b});
        }
        pi t = S.query(cur[x][1]);
        if (t.s != 2*MOD) ed.pb({(ll)t.f-cur[x][2],{x,t.s}});
    }
}

ll mst(vpi v) {
    N = sz(v); cur.resz(N); ed.clear();
    ind.clear(); FOR(i,N) ind.pb(i);
    sort(all(ind),[&v](int a, int b) { return v[a] < v[b]; });
}

```

```

FOR(i,N-1) if (v[ind[i]] == v[ind[i+1]]) ed.pb({0,{ind[i],ind
    ⇨[i+1]}});
FOR(i,2) { // it's probably ok to consider just two quadrants
    ⇨?
    FOR(i,N) {
        auto a = v[i];
        cur[i][2] = a.f+a.s;
    }
    FOR(i,N) { // first octant
        auto a = v[i];
        cur[i][0] = a.f-a.s;
        cur[i][1] = a.s;
    }
    solve();
    FOR(i,N) { // second octant
        auto a = v[i];
        cur[i][0] = a.f;
        cur[i][1] = a.s-a.f;
    }
    solve();
    trav(a,v) a = {a.s,-a.f}; // rotate 90 degrees, repeat
}
return kruskal(ed);
}

```

Dijkstra.h

Description: Dijkstra's algorithm for shortest path

Time: $\mathcal{O}(E \log V)$

31 lines

```

template<int SZ> struct dijkstra {
    vector<pair<int, ll>> adj[SZ];
    bool vis[SZ];
    ll d[SZ];

    void addEdge(int u, int v, ll l) {
        adj[u].PB(MP(v, l));
    }
    ll dist(int v) {
        return d[v];
    }
    void build(int u) {
        M00(i, SZ) vis[i] = 0;
        priority_queue<pair<ll, int>, vector<pair<ll, int>>,
            ⇨greater<pair<ll, int>>> pq;
        M00(i, SZ) d[i] = 1e17;
        d[u] = 0;
        pq.push(MP(0, u));
        while(!pq.empty()) {
            pair<ll, int> t = pq.top(); pq.pop();
            while(!pq.empty() && vis[t.S]) t = pq.top(), pq.pop()
                ⇨();
            vis[t.S] = 1;

            for(auto& v: adj[t.S]) if(!vis[v.F]) {
                if(d[v.F] > d[t.S] + v.S) {
                    d[v.F] = d[t.S] + v.S;
                    pq.push(MP(d[v.F], v.F));
                }
            }
        }
    }
};

```

DijkstraV2.h

Description: Dijkstra's algorithm for shortest path

Time: $\mathcal{O}(V^2)$

27 lines

```

template<int SZ> struct dijkstra {
    vector<pair<int, ll>> adj[SZ];
}

```

```

bool vis[SZ];
ll d[SZ];

void addEdge(int u, int v, ll l) {
    adj[u].PB(MP(v, l));
}
ll dist(int v) {
    return d[v];
}
void build(int u) {
    M00(i, SZ) vis[i] = 0;
    M00(i, SZ) d[i] = 1e17;
    d[u] = 0;
    while(1) {
        pair<ll, int> t = MP(1e17, -1);
        M00(i, SZ) if(!vis[i]) t = min(t, MP(d[i], i));
        if(t.S == -1) return;
        vis[t.S] = 1;

        for(auto& v: adj[t.S]) if(!vis[v.F]) {
            if(d[v.F] > d[t.S] + v.S) d[v.F] = d[t.S] + v.S
                ⇨;
        }
    }
};

```

7.2 Trees

LCAjumps.h

Description: calculates least common ancestor in tree with binary jumping

Time: $\mathcal{O}(N \log N)$

44 lines

```

template<int SZ> struct tree {
    vector<pair<int, ll>> adj[SZ];
    const static int LGSZ = 32-__builtin_clz(SZ-1);
    pair<int, ll> ppar[SZ][LGSZ];
    int depth[SZ];
    ll distfromroot[SZ];

    void addEdge(int u, int v, int d) {
        adj[u].PB(MP(v, d));
        adj[v].PB(MP(u, d));
    }
    void dfs(int u, int dep, ll dis) {
        depth[u] = dep;
        distfromroot[u] = dis;
        for(auto& v: adj[u]) if(ppar[u][0].F != v.F) {
            ppar[v.F][0] = MP(u, v.S);
            dfs(v.F, dep + 1, dis + v.S);
        }
    }
    void build() {
        ppar[0][0] = MP(0, 0);
        M00(i, SZ) depth[i] = 0;
        dfs(0, 0, 0);
        M00(i, 1, LGSZ) M00(j, SZ) {
            ppar[j][i].F = ppar[ppar[j][i-1].F][i-1].F;
            ppar[j][i].S = ppar[j][i-1].S + ppar[ppar[j][i-1].F]
                ⇨[i-1].S;
        }
    }
    int lca(int u, int v) {
        if(depth[u] < depth[v]) swap(u, v);
        M00d(i, LGSZ) if(depth[ppar[u][i].F] >= depth[v]) u =
            ⇨ppar[u][i].F;
        if(u == v) return u;
        M00d(i, LGSZ) {
            if(ppar[u][i].F != ppar[v][i].F) {

```

```

        u = ppar[u][i].F;
        v = ppar[v][i].F;
    }
    return ppar[u][0].F;
}
ll dist(int u, int v) {
    return distfromroot[u] + distfromroot[v] - 2*
        ↪ distfromroot[lca(u, v)];
}
};

```

CentroidDecomp.h

Description: can support tree path queries and updates

Time: $\mathcal{O}(N \log N)$

45 lines

```

template<int SZ> struct CD {
    vi adj[SZ];
    bool done[SZ];
    int sub[SZ], par[SZ];
    vl dist[SZ];
    pi cen[SZ];
    void addEdge(int a, int b) { adj[a].pb(b), adj[b].pb(a); }

    void dfs (int x) {
        sub[x] = 1;
        trav(y, adj[x]) if (!done[y] && y != par[x]) {
            par[y] = x; dfs(y);
            sub[x] += sub[y];
        }
    }
    int centroid(int x) {
        par[x] = -1; dfs(x);
        for (int sz = sub[x];;) {
            pi mx = {0,0};
            trav(y, adj[x]) if (!done[y] && y != par[x])
                ckmax(mx, {sub[y], y});
            if (mx.f*2 <= sz) return x;
            x = mx.s;
        }
    }

    void genDist(int x, int p) {
        dist[x].pb(dist[p].back()+1);
        trav(y, adj[x]) if (!done[y] && y != p) {
            cen[y] = cen[x];
            genDist(y, x);
        }
    }

    void gen(int x, bool fst = 0) {
        done[x = centroid(x)] = 1; dist[x].pb(0);
        if (fst) cen[x].f = -1;
        int co = 0;
        trav(y, adj[x]) if (!done[y]) {
            cen[y] = {x, co++};
            genDist(y, x);
        }
        trav(y, adj[x]) if (!done[y]) gen(y);
    }
    void init() { gen(1, 1); }
};

```

HLD.h

Description: Heavy Light Decomposition

Time: $\mathcal{O}(\log^2 N)$ per path operations

50 lines

```

template<int SZ, bool VALUES_IN_EDGES> struct HLD {
    int N; vi adj[SZ];
    int par[SZ], sz[SZ], depth[SZ];

```

```

    int root[SZ], pos[SZ];
    LazySegTree<ll, SZ> tree;
    void addEdge(int a, int b) { adj[a].pb(b), adj[b].pb(a); }

    void dfs_sz(int v = 1) {
        if (par[v]) adj[v].erase(find(all(adj[v]), par[v]));
        sz[v] = 1;
        trav(u, adj[v]) {
            par[u] = v; depth[u] = depth[v] + 1;
            dfs_sz(u); sz[v] += sz[u];
            if (sz[u] > sz[adj[v][0]]) swap(u, adj[v][0]);
        }
    }
    void dfs_hld(int v = 1) {
        static int t = 0;
        pos[v] = t++;
        trav(u, adj[v]) {
            root[u] = (u == adj[v][0] ? root[v] : u);
            dfs_hld(u);
        }
    }
    void init(int _N) {
        N = _N; par[1] = depth[1] = 0; root[1] = 1;
        dfs_sz(); dfs_hld();
    }

    template <class BinaryOperation>
    void processPath(int u, int v, BinaryOperation op) {
        for (; root[u] != root[v]; v = par[root[v]]) {
            if (depth[root[u]] > depth[root[v]]) swap(u, v);
            op(pos[root[v]], pos[v]);
        }
        if (depth[u] > depth[v]) swap(u, v);
        op(pos[u] + VALUES_IN_EDGES, pos[v]);
    }

    void modifyPath(int u, int v, int val) { // add val to
        ↪ vertices/edges along path
        processPath(u, v, [this, &val](int l, int r) { tree.upd(l,
            ↪ r, val); });
    }
    void modifySubtree(int v, int val) { // add val to vertices/
        ↪ edges in subtree
        tree.upd(pos[v] + VALUES_IN_EDGES, pos[v] + sz[v] - 1, val);
    }
    ll queryPath(int u, int v) { // query sum of path
        ll res = 0; processPath(u, v, [this, &res](int l, int r) {
            ↪ res += tree.qsum(l, r); });
        return res;
    }
};

```

7.3 DFS Algorithms

SCC.h

Description: Kosaraju's Algorithm: DFS two times to generate SCCs in topological order

Time: $\mathcal{O}(N + M)$

24 lines

```

template<int SZ> struct SCC {
    int N, comp[SZ];
    vi adj[SZ], radj[SZ], todo, allComp;
    bitset<SZ> visit;
    void addEdge(int a, int b) { adj[a].pb(b), radj[b].pb(a); }

    void dfs(int v) {
        visit[v] = 1;
        trav(w, adj[v]) if (!visit[w]) dfs(w);
        todo.pb(v);
    }

```

```

    }
    void dfs2(int v, int val) {
        comp[v] = val;
        trav(w, radj[v]) if (comp[w] == -1) dfs2(w, val);
    }

    void init(int _N) { // fills allComp
        N = _N;
        FOR(i, N) comp[i] = -1, visit[i] = 0;
        FOR(i, N) if (!visit[i]) dfs(i);
        reverse(all(todo)); // now todo stores vertices in order of
            ↪ topological sort
        trav(i, todo) if (comp[i] == -1) dfs2(i, i), allComp.pb(i);
    }
};

```

2SAT.h

Description: ?

"SCC.h"

38 lines

```

template<int SZ> struct TwoSat {
    SCC<2*SZ> S;
    bitset<SZ> ans;
    int N = 0;
    int addVar() { return N++; }

    void either(int x, int y) {
        x = max(2*x, -1-2*x), y = max(2*y, -1-2*y);
        S.addEdge(x^1, y); S.addEdge(y^1, x);
    }
    void implies(int x, int y) { either(~x, y); }
    void setVal(int x) { either(x, x); }
    void atMostOne(const vi& li) {
        if (sz(li) <= 1) return;
        int cur = ~li[0];
        FOR(i, 2, sz(li)) {
            int next = addVar();
            either(cur, ~li[i]);
            either(cur, next);
            either(~li[i], next);
            cur = ~next;
        }
        either(cur, ~li[1]);
    }

    bool solve(int _N) {
        if (_N != -1) N = _N;
        S.init(2*N);
        for (int i = 0; i < 2*N; i += 2)
            if (S.comp[i] == S.comp[i^1]) return 0;
        reverse(all(S.allComp));
        vi tmp(2*N);
        trav(i, S.allComp) if (tmp[i] == 0)
            tmp[i] = 1, tmp[S.comp[i^1]] = -1;
        FOR(i, N) if (tmp[S.comp[2*i]] == 1) ans[i] = 1;
        return 1;
    }
};

```

EulerPath.h

Description: Eulerian Path for both directed and undirected graphs

Time: $\mathcal{O}(N + M)$

30 lines

```

template<int SZ, bool directed> struct Euler {
    int N, M = 0;
    vpi adj[SZ];
    vpi::iterator its[SZ];
    vector<bool> used;

```

```

void addEdge(int a, int b) {
    if (directed) adj[a].pb({b,M});
    else adj[a].pb({b,M}), adj[b].pb({a,M});
    used.pb(0); M ++;
}

vpi solve(int _N, int src = 1) {
    N = _N;
    FOR(i,1,N+1) its[i] = begin(adj[i]);
    vector<pair<pi,int>> ret, s = {{src,-1},-1};
    while (sz(s)) {
        int x = s.back().f.f;
        auto& it = its[x], end = adj[x].end();
        while (it != end && used[it->s]) it ++;
        if (it == end) {
            if (sz(ret) && ret.back().f.s != s.back().f.f) return
                ⇨ {}; // path isn't valid
            ret.pb(s.back()), s.pop_back();
        } else { s.pb({it->f,x,it->s}); used[it->s] = 1; }
    }
    if (sz(ret) != M+1) return {};
    vpi ans; trav(t,ret) ans.pb({t.f.f,t.s});
    reverse(all(ans)); return ans;
}
};

```

BCC.h

Description: computes biconnected components

Time: $\mathcal{O}(N + M)$

37 lines

```

template<int SZ> struct BCC {
    int N;
    vpi adj[SZ], ed;
    void addEdge(int u, int v) {
        adj[u].pb({v,sz(ed)}), adj[v].pb({u,sz(ed)});
        ed.pb({u,v});
    }

    int disc[SZ];
    vi st; vector<vi> fin;
    int bcc(int u, int p = -1) { // return lowest disc
        static int ti = 0;
        disc[u] = ++ti; int low = disc[u];
        int child = 0;
        trav(i,adj[u]) if (i.s != p)
            if (!disc[i.f]) {
                child ++; st.pb(i.s);
                int LOW = bcc(i.f,i.s); ckmin(low,LOW);
                // disc[u] < LOW -> bridge
                if (disc[u] <= LOW) {
                    // if (p != -1 || child > 1) -> u is articulation
                    ⇨ point
                    vi tmp; while (st.back() != i.s) tmp.pb(st.back()),
                        ⇨ st.pop_back();
                    tmp.pb(st.back()), st.pop_back();
                    fin.pb(tmp);
                }
            } else if (disc[i.f] < disc[u]) {
                ckmin(low,disc[i.f]);
                st.pb(i.s);
            }
        }
        return low;
    }

    void init(int _N) {
        N = _N; FOR(i,N) disc[i] = 0;
        FOR(i,N) if (!disc[i]) bcc(i); // st should be empty after
            ⇨ each iteration
    }
}

```

};

7.4 Flows

Dinic.h

Description: faster flow

Time: $\mathcal{O}(N^2M)$ flow, $\mathcal{O}(M\sqrt{N})$ bipartite matching

45 lines

```

template<int SZ> struct Dinic {
    typedef ll F; // flow type
    struct Edge { int to, rev; F flow, cap; };

    int N,s,t;
    vector<Edge> adj[SZ];
    typename vector<Edge>::iterator cur[SZ];
    void addEdge(int u, int v, F cap) {
        assert(cap >= 0); // don't try smth dumb
        Edge a{v, sz(adj[v]), 0, cap}, b{u, sz(adj[u]), 0, 0};
        adj[u].pb(a), adj[v].pb(b);
    }

    int level[SZ];
    bool bfs() { // level = shortest distance from source
        // after computing flow, edges {u,v} such that level[u] \
            ⇨ neq -1, level[v] = -1 are part of min cut
        M00(i,N) level[i] = -1, cur[i] = begin(adj[i]);
        queue<int> q({s}); level[s] = 0;
        while (sz(q)) {
            int u = q.front(); q.pop();
            for (Edge e: adj[u]) if (level[e.to] < 0 && e.flow <
                ⇨ e.cap)
                q.push(e.to), level[e.to] = level[u]+1;
        }
        return level[t] >= 0;
    }

    F sendFlow(int v, F flow) {
        if (v == t) return flow;
        for (; cur[v] != end(adj[v]); cur[v]++) {
            Edge& e = *cur[v];
            if (level[e.to] != level[v]+1 || e.flow == e.cap)
                ⇨ continue;
            auto df = sendFlow(e.to,min(flow,e.cap-e.flow));
            if (df) { // saturated at least one edge
                e.flow += df; adj[e.to][e.rev].flow -= df;
                return df;
            }
        }
        return 0;
    }

    F maxFlow(int _N, int _s, int _t) {
        N = _N, s = _s, t = _t; if (s == t) return -1;
        F tot = 0;
        while (bfs()) while (auto df = sendFlow(s,numeric_limits<F>
            ⇨ ::max())) tot += df;
        return tot;
    }
}
};

```

MCMF.h

Description: Min-Cost Max Flow, no negative cycles allowed

Time: $\mathcal{O}(NM^2 \log M)$

53 lines

```

template<class T> using pqg = priority_queue<T,vector<T>,
    ⇨ greater<T>>;
template<class T> T poll(pqg<T>& x) {
    T y = x.top(); x.pop();
    return y;
}

```

```

template<int SZ> struct mcmf {
    typedef ll F; typedef ll C;
    struct Edge { int to, rev; F flow, cap; C cost; int id; };
    vector<Edge> adj[SZ];
    void addEdge(int u, int v, F cap, C cost) {
        assert(cap >= 0);
        Edge a{v, sz(adj[v]), 0, cap, cost}, b{u, sz(adj[u]), 0, 0,
            ⇨ -cost};
        adj[u].pb(a), adj[v].pb(b);
    }

    int N, s, t;
    pi pre[SZ]; // previous vertex, edge label on path
    pair<C,F> cost[SZ]; // tot cost of path, amount of flow
    C totCost, curCost; F totFlow;
    void reweight() { // makes all edge costs non-negative
        // all edges on shortest path become 0
        FOR(i,N) trav(p,adj[i]) p.cost += cost[i].f-cost[p.to].f;
    }
    bool spfa() { // reweight ensures that there will be negative
        ⇨ weights
        // only during the first time you run this
        FOR(i,N) cost[i] = {INF,0}; cost[s] = {0,INF};
        pqg<pair<C,int>> todo; todo.push({0,s});
        while (sz(todo)) {
            auto x = poll(todo); if (x.f > cost[x.s].f) continue;
            trav(a,adj[x.s]) if (x.f+a.cost < cost[a.to].f && a.flow
                ⇨ < a.cap) {
                // if costs are doubles, add some EPS to ensure that
                // you do not traverse some 0-weight cycle repeatedly
                pre[a.to] = {x.s,a.rev};
                cost[a.to] = {x.f+a.cost,min(a.cap-a.flow,cost[x.s].s)
                    ⇨ };
                todo.push({cost[a.to].f,a.to});
            }
        }
        curCost += cost[t].f; return cost[t].s;
    }

    void backtrack() {
        F df = cost[t].s; totFlow += df, totCost += curCost*df;
        for (int x = t; x != s; x = pre[x].f) {
            adj[x][pre[x].s].flow -= df;
            adj[pre[x].f][adj[x][pre[x].s].rev].flow += df;
        }
    }

    pair<F,C> calc(int _N, int _s, int _t) {
        N = _N; s = _s, t = _t; totFlow = totCost = curCost = 0;
        while (spfa()) reweight(), backtrack();
        return {totFlow, totCost};
    }
}
};

```

GomoryHu.h

Description: Compute max flow between every pair of vertices of undirected graph

56 lines

```

template<int SZ> struct GomoryHu {
    int N;
    vector<pair<pi,int>> ed;
    void addEdge(int a, int b, int c) { ed.pb({{a,b},c}); }

    vector<vi> cor = {}; // groups of vertices
    map<int,int> adj[2*SZ]; // current edges of tree
    int side[SZ];

    int gen(vector<vi> cc) {
        Dinic<SZ> D = Dinic<SZ>();
        vi comp(N+1); FOR(i,sz(cc)) trav(t,cc[i]) comp[t] = i;
    }
}

```



```

    trav(t,ed) if (comp[t.f.f] != comp[t.f.s]) {
        D.addEdge(comp[t.f.f],comp[t.f.s],t.s);
        D.addEdge(comp[t.f.s],comp[t.f.f],t.s);
    }
    int f = D.maxFlow(0,1);
    FOR(i,sz(cc)) trav(j,cc[i]) side[j] = D.level[i] >= 0; //
        ↪ min cut
    return f;
}

void fill(vi& v, int a, int b) {
    trav(t,cor[a]) v.pb(t);
    trav(t,adj[a]) if (t.f != b) fill(v,t.f,a);
}

void addTree(int a, int b, int c) { adj[a][b] = c, adj[b][a]
    ↪= c; }
void delTree(int a, int b) { adj[a].erase(b), adj[b].erase(a)
    ↪; }

vector<pair<pi,int>> init(int _N) { // returns edges of
    ↪ Gomory-Hu Tree
    N = _N;
    FOR(i,1,N+1) cor[0].pb(i);
    queue<int> todo; todo.push(0);
    while (sz(todo)) {
        int x = todo.front(); todo.pop();
        vector<vi> cc; trav(t,cor[x]) cc.pb({t});
        trav(t,adj[x]) {
            cc.pb({});
            fill(cc.back(),t.f,x);
        }
        int f = gen(cc); // run max flow
        cor.pb({}), cor.pb({});
        trav(t,cor[x]) cor[sz(cor)-2+side[t]].pb(t);
        FOR(i,2) if (sz(cor[sz(cor)-2+i]) > 1) todo.push(sz(cor)
            ↪ -2+i);
        FOR(i,sz(cor)-2) if (i != x && adj[i].count(x)) {
            addTree(i,sz(cor)-2+side[cor[i][0]],adj[i][x]);
            delTree(i,x);
        } // modify tree edges
        addTree(sz(cor)-2,sz(cor)-1,f);
    }
    vector<pair<pi,int>> ans;
    FOR(i,sz(cor)) trav(j,adj[i]) if (i < j.f)
        ans.pb({{cor[i][0],cor[j.f][0]},j.s});
    return ans;
}
};

```

7.5 Matching

DFSmatch.h

Description: naive bipartite matching

Time: $O(NM)$

26 lines

```

template<int SZ> struct MaxMatch {
    int N, flow = 0, match[SZ], rmatch[SZ];
    bitset<SZ> vis;
    vi adj[SZ];
    MaxMatch() {
        memset(match,0,sizeof match);
        memset(rmatch,0,sizeof rmatch);
    }

    void connect(int a, int b, bool c = 1) {
        if (c) match[a] = b, rmatch[b] = a;
        else match[a] = rmatch[b] = 0;
    }
}

```

```

bool dfs(int x) {
    if (!x) return 1;
    if (vis[x]) return 0;
    vis[x] = 1;
    trav(t,adj[x]) if (t != match[x] && dfs(rmatch[t]))
        return connect(x,t),1;
    return 0;
}

void tri(int x) { vis.reset(); flow += dfs(x); }
void init(int _N) {
    N = _N; FOR(i,1,N+1) if (!match[i]) tri(i);
}
};

```

Hungarian.h

Description: finds min cost to complete n jobs w/ m workers each worker is assigned to at most one job (n <= m)

Time: ?

28 lines

```

int HungarianMatch(const vector<vi>& a) { // cost array,
    ↪ negative values are ok
    int n = sz(a)-1, m = sz(a[0])-1; // jobs 1..n, workers 1..m
    vi u(n+1), v(m+1), p(m+1); // p[j] -> job picked by worker j
    FOR(i,1,n+1) { // find alternating path with job i
        p[0] = i; int j0 = 0;
        vi dist(m+1,MOD), pre(m+1,-1); // dist, previous vertex on
            ↪ shortest path
        vector<bool> done(m+1, false);
        do {
            done[j0] = true;
            int i0 = p[j0], j1; int delta = MOD;
            FOR(j,1,m+1) if (!done[j]) {
                auto cur = a[i0][j]-u[i0]-v[j];
                if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
                if (dist[j] < delta) delta = dist[j], j1 = j;
            }
            FOR(j,m+1) // just dijkstra with potentials
                if (done[j]) u[p[j]] += delta, v[j] -= delta;
            else dist[j] -= delta;
            j0 = j1;
        } while (p[j0]);
        do { // update values on alternating path
            int j1 = pre[j0];
            p[j0] = p[j1];
            j0 = j1;
        } while (j0);
    }
    return -v[0]; // min cost
}

```

UnweightedMatch.h

Description: general unweighted matching

Time: ?

79 lines

```

template<int SZ> struct UnweightedMatch {
    int vis[SZ], par[SZ], orig[SZ], match[SZ], aux[SZ], t, N; //
        ↪ 1-based index
    vi adj[SZ];
    queue<int> Q;
    void addEdge(int u, int v) {
        adj[u].pb(v); adj[v].pb(u);
    }

    void init(int n) {
        N = n; t = 0;
        FOR(i,N+1) {
            adj[i].clear();
            match[i] = aux[i] = par[i] = 0;
        }
    }
}

```

```

}

void augment(int u, int v) {
    int pv = v, nv;
    do {
        pv = par[v]; nv = match[pv];
        match[v] = pv; match[pv] = v;
        v = nv;
    } while(u != pv);
}

int lca(int v, int w) {
    ++t;
    while (1) {
        if (v) {
            if (aux[v] == t) return v; aux[v] = t;
            v = orig[par[match[v]]];
        }
        swap(v, w);
    }
}

void blossom(int v, int w, int a) {
    while (orig[v] != a) {
        par[v] = w; w = match[v];
        if (vis[w] == 1) Q.push(w), vis[w] = 0;
        orig[v] = orig[w] = a;
        v = par[w];
    }
}

bool bfs(int u) {
    fill(vis+1, vis+1+N, -1); iota(orig + 1, orig + N + 1, 1);
    Q = queue<int> (); Q.push(u); vis[u] = 0;
    while (sz(Q)) {
        int v = Q.front(); Q.pop();
        trav(x,adj[v]) {
            if (vis[x] == -1) {
                par[x] = v; vis[x] = 1;
                if (!match[x]) return augment(u, x), true;
                Q.push(match[x]); vis[match[x]] = 0;
            } else if (vis[x] == 0 && orig[v] != orig[x]) {
                int a = lca(orig[v], orig[x]);
                blossom(x, v, a); blossom(v, x, a);
            }
        }
    }
    return false;
}

int match() {
    int ans = 0;
    // find random matching (not necessary, constant
        ↪ improvement)
    vi V(N-1); iota(all(V), 1);
    shuffle(all(V), mt19937(0x94949));
    trav(x,V) if(!match[x])
        trav(y,adj[x]) if (!match[y]) {
            match[x] = y, match[y] = x;
            ++ans; break;
        }

    FOR(i,1,N+1) if (!match[i] && bfs(i)) ++ans;
    return ans;
}
};

```

7.6 Misc

MaximalCliques.h

Description: Finds all maximal cliques

Time: $\mathcal{O}(3^{n/3})$

19 lines

```
typedef bitset<128> B;
int N;
B adj[128];

void cliques(B P = ~B(), B X={}, B R={}) { // possibly in
    ↪ clique, not in clique, in clique
    if (!P.any()) {
        if (!X.any()) {
            // do smth with maximal clique
        }
        return;
    }
    auto q = (P|X)._Find_first();
    auto cands = P&~eds[q]; // clique must contain q or non-
        ↪ neighbor of q
    FOR(i,N) if (cands[i]) {
        R[i] = 1;
        cliques(eds, f, P & eds[i], X & eds[i], R);
        R[i] = P[i] = 0; X[i] = 1;
    }
}
```

LCT.h

Description: Link-Cut Tree, use vir for subtree size queries

Time: $\mathcal{O}(\log N)$

96 lines

```
typedef struct snode* sn;

struct snode {
    sn p, c[2]; // parent, children
    int val; // value in node
    int sum, mn, mx; // sum of values in subtree, min and max
        ↪ prefix sum
    bool flip = 0;
    // int vir = 0; stores sum of virtual children

    snode(int v) {
        p = c[0] = c[1] = NULL;
        val = v; calc();
    }

    friend int getSum(sn x) { return x?x->sum:0; }
    friend int getMn(sn x) { return x?x->mn:0; }
    friend int getMx(sn x) { return x?x->mx:0; }

    void prop() {
        if (!flip) return;
        swap(c[0],c[1]); tie(mn,mx) = mp(sum-mx,sum-mn);
        FOR(i,2) if (c[i]) c[i]->flip ^= 1;
        flip = 0;
    }
    void calc() {
        FOR(i,2) if (c[i]) c[i]->prop();
        int s0 = getSum(c[0]), s1 = getSum(c[1]); sum = s0+val+s1;
            ↪ // +vir
        mn = min(getMn(c[0]),s0+val+getMn(c[1]));
        mx = max(getMx(c[0]),s0+val+getMx(c[1]));
    }

    int dir() {
        if (!p) return -2;
        FOR(i,2) if (p->c[i] == this) return i;
    }
};
```

```
return -1; // p is path-parent pointer, not in current
        ↪ splay tree
}
bool isRoot() { return dir() < 0; }

friend void setLink(sn x, sn y, int d) {
    if (y) y->p = x;
    if (d >= 0) x->c[d] = y;
}
void rot() { // assume p and p->p propagated
    assert(!isRoot()); int x = dir(); sn pa = p;
    setLink(pa->p, this, pa->dir());
    setLink(pa, c[x^1], x);
    setLink(this, pa, x^1);
    pa->calc(); calc();
}
void splay() {
    while (!isRoot() && !p->isRoot()) {
        p->p->prop(), p->prop(), prop();
        dir() == p->dir() ? p->rot() : rot();
        rot();
    }
    if (!isRoot()) p->prop(), prop(), rot();
    prop();
}

void access() { // bring this to top of tree
    for (sn v = this, pre = NULL; v; v = v->p) {
        v->splay();
        // if (pre) v->vir -= pre->sz;
        // if (v->c[1]) v->vir += v->c[1]->sz;
        v->c[1] = pre; v->calc();
        pre = v;
        // v->sz should remain the same if using vir
    }
    splay(); assert(!c[1]); // left subtree of this is now path
        ↪ to root, right subtree is empty
}
void makeRoot() { access(); flip ^= 1; }
void set(int v) { splay(); val = v; calc(); } // change value
    ↪ in node, splay suffices instead of access because it
    ↪ doesn't affect values in nodes above it

friend sn lca(sn x, sn y) {
    if (x == y) return x;
    x->access(), y->access(); if (!x->p) return NULL; // access
        ↪ at y did not affect x, so they must not be connected
    x->splay(); return x->p ? x->p : x;
}
friend bool connected(sn x, sn y) { return lca(x,y); }
friend int balanced(sn x, sn y) {
    x->makeRoot(); y->access();
    return y->sum-2*y->mn;
}

friend bool link(sn x, sn y) { // make x parent of y
    if (connected(x,y)) return 0; // don't induce cycle
    y->makeRoot(); y->p = x;
    // x->access(); x->sz += y->sz; x->vir += y->sz;
    return 1; // success!
}
friend bool cut(sn x, sn y) { // x is originally parent of y
    x->makeRoot(); y->access();
    if (y->c[0] != x || x->c[0] || x->c[1]) return 0; // splay
        ↪ tree with y should not contain anything else besides x
    x->p = y->c[0] = NULL; y->calc(); return 1; // calc is
        ↪ redundant as it will be called elsewhere anyways?
}
};
```

DirectedMST.h

Description: computes minimum weight directed spanning tree, edge from $inv[i] \rightarrow i$ for all $i \neq r$

Time: $\mathcal{O}(M \log M)$

64 lines

```
"DSUrb.h"

struct Edge { int a, b; ll w; };
struct Node {
    Edge key;
    Node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
    if (!a || !b) return a ? a : b;
    a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

pair<ll,vi> dmst(int n, int r, const vector<Edge>& g) {
    DSUrb dsu; dsu.init(n); // DSU with rollback if need to
        ↪ return edges
    vector<Node> heap(n); // store edges entering each vertex in
        ↪ increasing order of weight
    trav(e,g) heap[e.b] = merge(heap[e.b], new Node(e));
    ll res = 0; vi seen(n,-1); seen[r] = r;
    vpi in(n,{-1,-1});
    vector<pair<int,vector<Edge>>> cycs;
    FOR(s,n) {
        int u = s, w;
        vector<pair<int,Edge>> path;
        while (seen[u] < 0) {
            if (!heap[u]) return {-1,{};};
            seen[u] = s;
            Edge e = heap[u]->top(); path.pb({u,e});
            heap[u]->delta -= e.w, pop(heap[u]);
            res += e.w, u = dsu.get(e.a);
            if (seen[u] == s) { // compress verts in cycle
                Node* cyc = 0; cycs.pb({u,{}});
                do {
                    cyc = merge(cyc, heap[w = path.back().f]);
                    cycs.back().s.pb(path.back().s);
                    path.pop_back();
                } while (dsu.unite(u, w));
                u = dsu.get(u); heap[u] = cyc, seen[u] = -1;
            }
        }
        trav(t,path) in[dsu.get(t.s.b)] = {t.s.a,t.s.b}; // found
            ↪ path from root
    }
    while (sz(cycs)) { // expand cycs to restore sol
        auto c = cycs.back(); cycs.pop_back();
        pi inEdge = in[c.f];
        trav(t,c.s) dsu.rollback();
        trav(t,c.s) in[dsu.get(t.b)] = {t.a,t.b};
        in[dsu.get(inEdge.s)] = inEdge;
    }
    vi inv;
    FOR(i,n) {
        assert(i == r ? in[i].s == -1 : in[i].s == i);
        inv.pb(in[i].f);
    }
}
```

```

    }
    return {res,inv};
}

```

DominatorTree.h

Description: a dominates b iff every path from 1 to b passes through a
Time: $\mathcal{O}(M \log N)$

46 lines

```

template<int SZ> struct Dominator {
    vi adj[SZ], ans[SZ]; // input edges, edges of dominator tree
    vi radj[SZ], child[SZ], sdomChild[SZ];
    int label[SZ], rlabel[SZ], sdom[SZ], dom[SZ], co;
    int root = 1;

    int par[SZ], bes[SZ];
    int get(int x) {
        // DSU with path compression
        // get vertex with smallest sdom on path to root
        if (par[x] != x) {
            int t = get(par[x]); par[x] = par[par[x]];
            if (sdom[t] < sdom[bes[x]]) bes[x] = t;
        }
        return bes[x];
    }

    void dfs(int x) { // create DFS tree
        label[x] = ++co; rlabel[co] = x;
        sdom[co] = par[co] = bes[co] = co;
        trav(y, adj[x]) {
            if (!label[y]) {
                dfs(y);
                child[label[x]].pb(label[y]);
            }
            radj[label[y]].pb(label[x]);
        }
    }

    void init() {
        dfs(root);
        ROF(i, 1, co+1) {
            trav(j, radj[i]) ckmin(sdom[i], sdom[get(j)]);
            if (i > 1) sdomChild[sdom[i]].pb(i);
            trav(j, sdomChild[i]) {
                int k = get(j);
                if (sdom[j] == sdom[k]) dom[j] = sdom[j];
                else dom[j] = k;
            }
            trav(j, child[i]) par[j] = i;
        }
        FOR(i, 2, co+1) {
            if (dom[i] != sdom[i]) dom[i] = dom[dom[i]];
            ans[rlabel[dom[i]]].pb(rlabel[i]);
        }
    }
};

```

EdgeColor.h

Description: naive implementation of Misra & Gries edge coloring, by Viz-ing's Theorem a simple graph with max degree d can be edge colored with at most $d+1$ colors
Time: $\mathcal{O}(MN^2)$

54 lines

```

template<int SZ> struct EdgeColor {
    int N = 0, maxDeg = 0, adj[SZ][SZ], deg[SZ];
    EdgeColor() {
        memset(adj, 0, sizeof adj);
        memset(deg, 0, sizeof deg);
    }
    void addEdge(int a, int b, int c) {
        adj[a][b] = adj[b][a] = c;
    }
};

```

```

    }
    int delEdge(int a, int b) {
        int c = adj[a][b];
        adj[a][b] = adj[b][a] = 0;
        return c;
    }
    vector<bool> genCol(int x) {
        vector<bool> col(N+1); FOR(i, N) col[adj[x][i]] = 1;
        return col;
    }
    int freeCol(int u) {
        auto col = genCol(u);
        int x = 1; while (col[x]) x++; return x;
    }
    void invert(int x, int d, int c) {
        FOR(i, N) if (adj[x][i] == d)
            delEdge(x, i), invert(i, c, d), addEdge(x, i, c);
    }
    void addEdge(int u, int v) { // follows wikipedia steps
        // check if you can add edge w/o doing any work
        assert(N); ckmax(maxDeg, max(++deg[u], ++deg[v]));
        auto a = genCol(u), b = genCol(v);
        FOR(i, 1, maxDeg+2) if (!a[i] && !b[i]) return addEdge(u, v, i);
    }

    // 2. find maximal fan of u starting at v
    vector<bool> use(N); vi fan = {v}; use[v] = 1;
    while (1) {
        auto col = genCol(fan.back());
        if (sz(fan) > 1) col[adj[fan.back()][u]] = 0;
        int i = 0; while (i < N && (use[i] || col[adj[u][i]])) i
            ++;
        if (i < N) fan.pb(i), use[i] = 1;
        else break;
    }

    // 3/4. choose free cols for endpoints of fan, invert cd_u
    auto path
    int c = freeCol(u), d = freeCol(fan.back()); invert(u, d, c);
    // 5. find i such that d is free on fan[i]
    int i = 0; while (i < sz(fan) && genCol(fan[i])[d]
        && adj[u][fan[i]] != d) i++;
    assert (i != sz(fan));
    // 6. rotate fan from 0 to i
    FOR(j, i) addEdge(u, fan[j], delEdge(u, fan[j+1]));
    // 7. add new edge
    addEdge(u, fan[i], d);
}
};

```

Geometry (8)

8.1 Primitives

Point.h

Description: Easy Geo

44 lines

```

typedef ld T;
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }

namespace Point {
    typedef pair<T, T> P;
    typedef vector<P> vP;

    P dir(T ang) {
        auto c = exp(ang*complex<T>(0, 1));
        return P(c.real(), c.imag());
    }
};

```

```

T norm(P x) { return x.f*x.f+x.s*x.s; }
T abs(P x) { return sqrt(norm(x)); }
T angle(P x) { return atan2(x.s, x.f); }
P conj(P x) { return P(x.f, -x.s); }

```

```

P operator+(const P& l, const P& r) { return P(l.f+r.f, l.s+r
    ↪s); }
P operator-(const P& l, const P& r) { return P(l.f-r.f, l.s-r
    ↪s); }
P operator*(const P& l, const T& r) { return P(l.f*r, l.s*r);
    ↪ }
P operator*(const T& l, const P& r) { return r*l; }
P operator/(const P& l, const T& r) { return P(l.f/r, l.s/r);
    ↪ }
P operator*(const P& l, const P& r) { return P(l.f*r.f-l.s*r
    ↪s, l.s*r.f+l.f*r.s); }
P operator/(const P& l, const P& r) { return l*conj(r)/norm(r
    ↪); }

```

```

P& operator+=(P& l, const P& r) { return l = l+r; }
P& operator-=(P& l, const P& r) { return l = l-r; }
P& operator*=(P& l, const T& r) { return l = l*r; }
P& operator/=(P& l, const T& r) { return l = l/r; }
P& operator*=(P& l, const P& r) { return l = l*r; }
P& operator/=(P& l, const P& r) { return l = l/r; }

```

```

P unit(P x) { return x/abs(x); }
T dot(P a, P b) { return (conj(a)*b).f; }
T cross(P a, P b) { return (conj(a)*b).s; }
T cross(P p, P a, P b) { return cross(a-p, b-p); }
P rotate(P a, T b) { return a*P(cos(b), sin(b)); }

```

```

P reflect(P p, P a, P b) { return a+conj((p-a)/(b-a))*(b-a);
    ↪ }
P foot(P p, P a, P b) { return (p+reflect(p, a, b))/(T)2; }
bool onSeg(P p, P a, P b) { return cross(a, b, p) == 0 && dot(p
    ↪ -a, p-b) <= 0; }
};

```

using namespace Point;

AngleCmp.h

Description: sorts points according to atan2

5 lines

```

template<class T> int half(pair<T, T> x) { return mp(x.s, x.f) >
    ↪ mp((T)0, (T)0); }
bool angleCmp(P a, P b) {
    int A = half(a), B = half(b);
    return A == B ? cross(a, b) > 0 : A < B;
}

```

LineDist.h

Description: computes distance between P and line AB

1 lines

```

T lineDist(P p, P a, P b) { return abs(cross(p, a, b))/abs(a-b);
    ↪ }

```

SegDist.h

Description: computes distance between P and line segment AB

5 lines

```

T segDist(P p, P a, P b) {
    if (dot(p-a, b-a) <= 0) return abs(p-a);
    if (dot(p-b, a-b) <= 0) return abs(p-b);
    return lineDist(p, a, b);
}

```

LineIntersect.h

Description: computes the intersection point(s) of lines AB, CD ; returns $-1,0,0$ if infinitely many, $0,0,0$ if none, $1,x$ if x is the unique point

```
"Point.h" 8 lines

P extension(P a, P b, P c, P d) {
    T x = cross(a,b,c), y = cross(a,b,d);
    return (d*x-c*y)/(x-y);
}

pair<int,P> lineIntersect(P a, P b, P c, P d) {
    if (cross(b-a,d-c) == 0) return {-(cross(a,c,d) == 0),P(0,0)}
    ⇨;
    return {1,extension(a,b,c,d)};
}
```

SegIntersect.h

Description: computes the intersection point(s) of line segments AB, CD

```
"Point.h" 11 lines

vP segIntersect(P a, P b, P c, P d) {
    T x = cross(a,b,c), y = cross(a,b,d);
    T X = cross(c,d,a), Y = cross(c,d,b);
    if (sgn(x)*sgn(y) < 0 && sgn(X)*sgn(Y) < 0) return {(d*x-c*y)
    ⇨/(x-y)};
    set<P> s;
    if (onSeg(a,c,d)) s.insert(a);
    if (onSeg(b,c,d)) s.insert(b);
    if (onSeg(c,a,b)) s.insert(c);
    if (onSeg(d,a,b)) s.insert(d);
    return {all(s)};
}
```

8.2 Polygons

Area.h

Description: computes area + the center of mass of a polygon with constant mass per unit area

Time: $\mathcal{O}(N)$

```
"Point.h" 16 lines

T area(const vP& v) {
    T area = 0;
    FOR(i,sz(v)) {
        int j = (i+1)%sz(v); T a = cross(v[i],v[j]);
        area += a;
    }
    return std::abs(area)/2;
}

P centroid(const vP& v) {
    P cen(0,0); T area = 0; // 2*signed area
    FOR(i,sz(v)) {
        int j = (i+1)%sz(v); T a = cross(v[i],v[j]);
        cen += a*(v[i]+v[j]); area += a;
    }
    return cen/area/(T)3;
}
```

InPoly.h

Description: tests whether a point is inside, on, or outside the perimeter of any polygon

Time: $\mathcal{O}(N)$

```
"Point.h" 10 lines

string inPoly(const vP& p, P z) {
    int n = sz(p), ans = 0;
    FOR(i,n) {
        P x = p[i], y = p[(i+1)%n];
        if (onSeg(z,x,y)) return "on";
        if (x.s > y.s) swap(x,y);
        if (x.s <= z.s && y.s > z.s && cross(z,x,y) > 0) ans ^= 1;
    }
}
```

```
    return ans ? "in" : "out";
}
```

ConvexHull.h

Description: Top-bottom convex hull

Time: $\mathcal{O}(N \log N)$

```
"ConvexHull.h" 48 lines

struct convexHull {
    set<pair<ld,ld>> dupChecker;
    vector<pair<ld,ld>> points;
    vector<pair<ld,ld>> dn, up, hull;

    convexHull() {}
    bool cw(pd o, pd a, pd b) {
        return ((a.f-o.f)*(b.s-o.s)-(a.s-o.s)*(b.f-o.f) <= 0);
    }
    void addPoint(pair<ld,ld> p) {
        if(dupChecker.count(p)) return;
        points.pb(p);
        dupChecker.insert(p);
    }
    void addPoint(ld x, ld y) {
        addPoint(mp(x,y));
    }
    void build() {
        sort(points.begin(), points.end());
        if(sz(points) < 3) {
            for(pair<ld,ld> p: points) {
                dn.pb(p);
                hull.pb(p);
            }
            M00d(i, sz(points)) {
                up.pb(points[i]);
            }
        } else {
            for(int i = 0; i < (int)points.size(); i++) {
                while(dn.size() >= 2 && cw(dn[dn.size()-2], dn[
                ⇨dn.size()-1], points[i])) {
                    dn.erase(dn.end()-1);
                }
                dn.push_back(points[i]);
            }
            for(int i = (int)points.size()-1; i >= 0; i--) {
                while(up.size() >= 2 && cw(up[up.size()-2], up[
                ⇨up.size()-1], points[i])) {
                    up.erase(up.end()-1);
                }
                up.push_back(points[i]);
            }
            sort(dn.begin(), dn.end());
            sort(up.begin(), up.end());

            for(int i = 0; i < up.size()-1; i++) hull.pb(up[i])
            ⇨;
            for(int i = sz(dn)-1; i > 0; i--) hull.pb(dn[i]);
        }
    }
};
```

PolyDiameter.h

Description: computes longest distance between two points in P

Time: $\mathcal{O}(N)$ given convex hull

```
"ConvexHull.h" 10 lines

ld diameter(vP P) { // rotating calipers
    P = hull(P);
    int n = sz(P), ind = 1; ld ans = 0;
    FOR(i,n)
        for (int j = (i+1)%n;;ind = (ind+1)%n) {
```

```
        ckmax(ans,abs(P[i]-P[ind]));
        if (cross(P[j]-P[i],P[(ind+1)%n]-P[ind]) <= 0) break;
    }
    return ans;
}
```

8.3 Circles

Circles.h

Description: misc operations with two circles

```
"Point.h" 46 lines

typedef pair<P,T> circ;
bool on(circ x, P y) { return abs(y-x.f) == x.s; }
bool in(circ x, P y) { return abs(y-x.f) <= x.s; }
T arcLength(circ x, P a, P b) {
    P d = (a-x.f)/(b-x.f);
    return x.s*acos(d.f);
}

P intersectPoint(circ x, circ y, int t = 0) { // assumes
    ⇨intersection points exist
    T d = abs(x.f-y.f); // distance between centers
    T theta = acos((x.s*x.s+d*d-y.s*y.s)/(2*x.s*d)); // law of
    ⇨cosines
    P tmp = (y.f-x.f)/d*x.s;
    return x.f+tmp*dir(t == 0 ? theta : -theta);
}

T intersectArea(circ x, circ y) { // not thoroughly tested
    T d = abs(x.f-y.f), a = x.s, b = y.s; if (a < b) swap(a,b);
    if (d >= a+b) return 0;
    if (d <= a-b) return PI*b*b;
    auto ca = (a*a+d*d-b*b)/(2*a*d), cb = (b*b+d*d-a*a)/(2*b*d);
    auto s = (a+b*d)/2, h = 2*sqrt(s*(s-a)*(s-b)*(s-d))/d;
    return a*a*acos(ca)+b*b*acos(cb)-d*h;
}

P tangent(P x, circ y, int t = 0) {
    y.s = abs(y.s); // abs needed because internal calls y.s < 0
    if (y.s == 0) return y.f;
    T d = abs(x-y.f);
    P a = pow(y.s/d,2)*(x-y.f)+y.f;
    P b = sqrt(d*d-y.s*y.s)/d*y.s*unit(x-y.f)*dir(PI/2);
    return t == 0 ? a+b : a-b;
}

vector<pair<P,P>> external(circ x, circ y) { // external
    ⇨tangents
    vector<pair<P,P>> v;
    if (x.s == y.s) {
        P tmp = unit(x.f-y.f)*x.s*dir(PI/2);
        v.pb(mp(x.f+tmp,y.f+tmp));
        v.pb(mp(x.f-tmp,y.f-tmp));
    } else {
        P p = (y.s*x.f-x.s*y.f)/(y.s-x.s);
        FOR(i,2) v.pb({tangent(p,x,i),tangent(p,y,i)});
    }
    return v;
}

vector<pair<P,P>> internal(circ x, circ y) { // internal
    ⇨tangents
    x.s *= -1; return external(x,y);
}
```

Circumcenter.h

Description: returns {circumcenter,circumradius}

```
"Point.h" 5 lines

pair<P,T> ccCenter(P a, P b, P c) {
    b -= a; c -= a;
    P res = b*c*(conj(c)-conj(b))/(b*conj(c)-conj(b)*c);
```

```
    return {a+res,abs(res)};
}
```

MinEnclosingCircle.h

Description: computes minimum enclosing circle

Time: expected $\mathcal{O}(N)$

"Circumcenter.h" 13 lines

```
pair<P, T> mec(vp ps) {
    shuffle(all(ps), mt19937(time(0)));
    P o = ps[0]; T r = 0, EPS = 1 + 1e-8;
    FOR(i,sz(ps)) if (abs(o-ps[i]) > r*EPS) {
        o = ps[i], r = 0;
        FOR(j,i) if (abs(o-ps[j]) > r*EPS) {
            o = (ps[i]+ps[j])/2, r = abs(o-ps[i]);
            FOR(k,j) if (abs(o-ps[k]) > r*EPS)
                tie(o,r) = ccCenter(ps[i],ps[j],ps[k]);
        }
    }
    return {o,r};
}
```

8.4 Misc

ClosestPair.h

Description: line sweep to find two closest points

Time: $\mathcal{O}(N \log N)$

21 lines

using namespace Point;

```
pair<P,P> solve(vp v) {
    pair<ld,pair<P,P>> bes; bes.f = INF;
    set<P> S; int ind = 0;

    sort(all(v));
    FOR(i,sz(v)) {
        if (i && v[i] == v[i-1]) return {v[i],v[i]};
        for (; v[i].f-v[ind].f >= bes.f; ++ind)
            S.erase({v[ind].s,v[ind].f});
        for (auto it = S.sub({v[i].s-bes.f,INF});
             it != end(S) && it->f < v[i].s+bes.f; ++it) {
            P t = {it->s,it->f};
            ckmin(bes,{abs(t-v[i]),{t,v[i]}});
        }
        S.insert({v[i].s,v[i].f});
    }

    return bes.s;
}
```

DelaunayFast.h

Description: Delaunay Triangulation, concyclic points are OK (but not all collinear)

Time: $\mathcal{O}(N \log N)$

"Point.h" 94 lines

typedef ll T;

```
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point
```

```
struct Quad {
    bool mark; Q o, rot; P p;
    P F() { return r()->p; }
    Q r() { return rot->rot; }
    Q prev() { return rot->o->rot; }
    Q next() { return r()->prev(); }
};
```

```
// test if p is in the circumcircle
bool circ(P p, P a, P b, P c) {
    ll ar = cross(a,b,c); assert(ar); if (ar < 0) swap(a,b);
    lll p2 = norm(p), A = norm(a)-p2,
        B = norm(b)-p2, C = norm(c)-p2;
    return cross(p,a,b)*C+cross(p,b,c)*A+cross(p,c,a)*B > 0;
}
```

```
Q makeEdge(P orig, P dest) {
    Q q[] = {new Quad{0,0,0,orig}, new Quad{0,0,0,arb},
            new Quad{0,0,0,dest}, new Quad{0,0,0,arb}};
    FOR(i,4) q[i]->o = q[-i & 3], q[i]->rot = q[(i+1) & 3];
    return *q;
}

void splice(Q a, Q b) {
    swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}

Q connect(Q a, Q b) {
    Q q = makeEdge(a->F(), b->p);
    splice(q, a->next());
    splice(q->r(), b);
    return q;
}
```

```
pair<Q,Q> rec(const vector<P>& s) {
    if (sz(s) <= 3) {
        Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
        if (sz(s) == 2) return { a, a->r() };
        splice(a->r(), b);
        auto side = cross(s[0], s[1], s[2]);
        Q c = side ? connect(b, a) : 0;
        return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
    }
}
```

```
#define H(e) e->F(), e->p
#define valid(e) (cross(e->F(),H(base)) > 0)
    Q A, B, ra, rb;
    int half = sz(s) / 2;
    tie(ra, A) = rec({all(s) - half});
    tie(B, rb) = rec({sz(s) - half + all(s)});
    while ((cross(B->p,H(A)) < 0 && (A = A->next())) ||
            (cross(A->p,H(B)) > 0 && (B = B->r()->o)));
    Q base = connect(B->r(), A);
    if (A->p == ra->p) ra = base->r();
    if (B->p == rb->p) rb = base;
```

```
#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
    while (circ(e->dir->F(), H(base), e->F())) { \
        Q t = e->dir; \
        splice(e, e->prev()); \
        splice(e->r(), e->r()->prev()); \
        e = t; \
    }
    for (;;) {
        DEL(LC, base->r(), o); DEL(RC, base, prev());
        if (!valid(LC) && !valid(RC)) break;
        if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
            base = connect(RC, base->r());
        else
            base = connect(base->r(), LC->r());
    }
    return {ra, rb};
}
```

```
vector<array<P,3>> triangulate(vector<P> pts) {
    sort(all(pts)); assert(unique(all(pts)) == pts.end());
    if (sz(pts) < 2) return {};

```

```
    Q e = rec(pts).f; vector<Q> q = {e};
```

```
    int qi = 0;
    while (cross(e->o->F(), e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
    q.push_back(c->r()); c = c->next(); } while (c != e); }
    ADD; pts.clear();
    while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;

    vector<array<P,3>> ret;
    FOR(i,sz(pts)/3) ret.pb({pts[3*i],pts[3*i+1],pts[3*i+2]});
    return ret;
}
```

8.5 3D

Point3D.h

Description: Basic 3D Geometry

45 lines

typedef ld T;

```
namespace Point3D {
    typedef array<T,3> P3;
    typedef vector<P3> vP3;
```

```
T norm(const P3& x) {
    T sum = 0; FOR(i,sz(x)) sum += x[i]*x[i];
    return sum;
}

T abs(const P3& x) { return sqrt(norm(x)); }
```

```
P3& operator+=(P3& l, const P3& r) { FOR(i,3) l[i] += r[i];
    ↪return l; }
P3& operator-=(P3& l, const P3& r) { FOR(i,3) l[i] -= r[i];
    ↪return l; }
P3& operator*=(P3& l, const T& r) { FOR(i,3) l[i] *= r;
    ↪return l; }
P3& operator/=(P3& l, const T& r) { FOR(i,3) l[i] /= r;
    ↪return l; }
```

```
P3 operator+(P3 l, const P3& r) { return l += r; }
P3 operator-(P3 l, const P3& r) { return l -= r; }
P3 operator*(P3 l, const T& r) { return l *= r; }
P3 operator*(const T& r, const P3& l) { return l*r; }
P3 operator/(P3 l, const T& r) { return l /= r; }
```

```
T dot(const P3& a, const P3& b) {
    T sum = 0; FOR(i,3) sum += a[i]*b[i];
    return sum;
}

P3 cross(const P3& a, const P3& b) {
    return {a[1]*b[2]-a[2]*b[1],
            a[2]*b[0]-a[0]*b[2],
            a[0]*b[1]-a[1]*b[0]};
}
```

```
bool isMult(const P3& a, const P3& b) {
    auto c = cross(a,b);
    FOR(i,sz(c)) if (c[i] != 0) return 0;
    return 1;
}
```

```
bool collinear(const P3& a, const P3& b, const P3& c) {
    ↪return isMult(b-a,c-a); }
bool coplanar(const P3& a, const P3& b, const P3& c, const P3
    ↪& d) {
    return isMult(cross(b-a,c-a),cross(b-a,d-a));
}

}
```

using namespace Point3D;

Hull3D.h

Description: 3D Convex Hull + Polyedron Volume

Time: $\mathcal{O}(N^2)$

"Point3D.h"	48 lines
<pre>struct ED { void ins(int x) { (a == -1 ? a : b) = x; } void rem(int x) { (a == x ? a : b) = -1; } int cnt() { return (a != -1) + (b != -1); } int a, b; }; struct F { P3 q; int a, b, c; }; vector<F> hull3d(const vP3& A) { assert(sz(A) >= 4); vector<vector<ED>> E(sz(A), vector<ED>(sz(A), {-1, -1})); #define E(x,y) E[f.x][f.y] vector<F> FS; // faces auto mf = [&](int i, int j, int k, int l) { // make face P3 q = cross(A[j]-A[i],A[k]-A[i]); if (dot(q,A[l]) > dot(q,A[i])) q *= -1; // make sure q ↪points outward F f{q, i, j, k}; E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i); FS.pb(f); }; FOR(i,4) FOR(j,i+1,4) FOR(k,j+1,4) mf(i, j, k, 6-i-j-k); FOR(i,4,sz(A)) { FOR(j,sz(FS)) { F f = FS[j]; if (dot(f.q,A[i]) > dot(f.q,A[f.a])) { // face is visible ↪, remove edges E(a,b).rem(f.c), E(a,c).rem(f.b), E(b,c).rem(f.a); swap(FS[j--], FS.back()); FS.pop_back(); } FOR(j,sz(FS)) { // add faces with new point F f = FS[j]; #define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, ↪ f.c); C(a, b, c); C(a, c, b); C(b, c, a); } } trav(it, FS) if (dot(cross(A[it.b]-A[it.a],A[it.c]-A[it.a]), ↪it.q) <= 0) swap(it.c, it.b); return FS; } // computes hull where no four are coplanar</pre>	

T signedPolyVolume(const vP3& p, const vector<F>& trilst) { T v = 0; trav(i,trilst) v += dot(cross(p[i.a],p[i.b]),p[i.c]); return v/6; }	
--	--

Strings (9)

9.1 Lightweight

KMP.h

Description: f[i] equals the length of the longest proper suffix of the i -th prefix of s that is a prefix of s

Time: $\mathcal{O}(N)$

vi kmp(string s) { int N = sz(s); vi f(N+1); f[0] = -1;	15 lines
--	----------

FOR(i,1,N+1) { f[i] = f[i-1]; while (f[i] != -1 && s[f[i]] != s[i-1]) f[i] = f[f[i]]; f[i] ++; } return f; } vi getOc(string a, string b) { // find occurrences of a in b vi f = kmp(a+"@"+b), ret; FOR(i,sz(a),sz(b)+1) if (f[i+sz(a)+1] == sz(a)) ret.pb(i-sz(a) ↪)); return ret; }	
---	--

Z.h

Description: for each index i , computes the the maximum len such that $s.substr(0,len) == s.substr(i,len)$

Time: $\mathcal{O}(N)$

vi z(string s) { int N = sz(s); s += '#'; vi ans(N); ans[0] = N; int L = 1, R = 0; FOR(i,1,N) { if (i <= R) ans[i] = min(R-i+1,ans[i-L]); while (s[i+ans[i]] == s[ans[i]]) ans[i] ++; if (i+ans[i]-1 > R) L = i, R = i+ans[i]-1; } return ans; } vi getPrefix(string a, string b) { // find prefixes of a in b vi t = z(a+b), T(sz(b)); FOR(i,sz(T)) T[i] = min(t[i+sz(a)],sz(a)); return T; } // pr(z("abcababcbabcaba"),getPrefix("abcab","uwetrabcerabcb")) ↪;	19 lines
--	----------

Manacher.h

Description: Calculates length of largest palindrome centered at each character of string

Time: $\mathcal{O}(N)$

vi manacher(string s) { string sl = "@"; trav(C,s) sl += c, sl += "#"; sl[sz(sl)-1] = '&'; vi ans(sz(sl)-1); int lo = 0, hi = 0; FOR(i,1,sz(sl)-1) { if (i != 1) ans[i] = min(hi-i,ans[hi-i+lo]); while (sl[i-ans[i]-1] == sl[i+ans[i]+1]) ans[i] ++; if (i+ans[i] > hi) lo = i-ans[i], hi = i+ans[i]; } ans.erase(begin(ans)); FOR(i,sz(ans)) if ((i&1) == (ans[i]&1)) ans[i] ++; // adjust ↪lengths return ans; } // ps(manacher("abacaba"))	18 lines
--	----------

MinRotation.h

Description: minimum rotation of string

Time: $\mathcal{O}(N)$

int minRotation(string s) { int a = 0, N = sz(s); s += s; FOR(b,N) FOR(i,N) { // a is current best rotation found up to ↪ b-1 if (a+i == b s[a+i] < s[b+i]) { b += max(0, i-1); break; ↪ } // b to b+i-1 can't be better than a to a+i-1 if (s[a+i] > s[b+i]) { a = b; break; } // new best found } return a; }	8 lines
---	---------

LyndonFactorization.h

Description: A string is "simple" if it is strictly smaller than any of its own nontrivial suffixes. The Lyndon factorization of the string s is a factorization $s = w_1w_2 \dots w_k$ where all strings w_i are simple and $w_1 \geq w_2 \geq \dots \geq w_k$

Time: $\mathcal{O}(N)$

vector<string> duval(const string& s) { int n = sz(s); vector<string> factors; for (int i = 0; i < n;) { int j = i + 1, k = i; for (; j < n && s[k] <= s[j]; j++) { if (s[k] < s[j]) k = i; else k ++; } for (; i <= k; i += j-k) factors.pb(s.substr(i, j-k)); } return factors; } int minRotation(string s) { // get min index i such that cyclic ↪ shift starting at i is min rotation int n = sz(s); s += s; auto d = duval(s); int ind = 0, ans = 0; while (ans+sz(d[ind]) < n) ans += sz(d[ind++]); while (ind && d[ind] == d[ind-1]) ans -= sz(d[ind--]); return ans; }	20 lines
--	----------

RabinKarp.h

Description: generates hash values of any substring in $\mathcal{O}(1)$, equal strings have same hash value

Time: $\mathcal{O}(N)$ build, $\mathcal{O}(1)$ get hash value of a substring

template<int SZ> struct rabinKarp { const ll mods[3] = {1000000007, 999119999, 1000992299}; ll p[3][SZ]; ll h[3][SZ]; const ll base = 1000696969; rabinKarp() {} void build(string a) { MOO(i, 3) { p[i][0] = 1; h[i][0] = (int)a[0]; MOO(j, 1, (int)a.length()) { p[i][j] = (p[i][j-1] * mods[i]) % base; h[i][j] = (h[i][j-1] * mods[i] + (int)a[j]) % ↪base; } } } tuple<ll, ll, ll> hsh(int a, int b) { if(a == 0) return make_tuple(h[0][b], h[1][b], h[2][b]) ↪; tuple<ll, ll, ll> ans;	25 lines
---	----------


```

get<0>(ans) = ((h[0][b] - h[0][a-1]*p[0][b-a+1]) %
    ↪base) + base) % base;
get<1>(ans) = ((h[1][b] - h[1][a-1]*p[1][b-a+1]) %
    ↪base) + base) % base;
get<2>(ans) = ((h[2][b] - h[2][a-1]*p[2][b-a+1]) %
    ↪base) + base) % base;
return ans;
}
};

```

9.2 Suffix Structures

ACfixed.h

Description: for each prefix, stores link to max length suffix which is also a prefix

Time: $\mathcal{O}(N \Sigma)$ 36 lines

```

struct ACfixed { // fixed alphabet
    struct node {
        array<int,26> to;
        int link;
    };
    vector<node> d;
    ACfixed() { d.eb(); }

    int add(string s) { // add word
        int v = 0;
        trav(C,s) {
            int c = C-'a';
            if (!d[v].to[c]) {
                d[v].to[c] = sz(d);
                d.eb();
            }
            v = d[v].to[c];
        }
        return v;
    }

    void init() { // generate links
        d[0].link = -1;
        queue<int> q; q.push(0);
        while (sz(q)) {
            int v = q.front(); q.pop();
            FOR(c,26) {
                int u = d[v].to[c]; if (!u) continue;
                d[u].link = d[v].link == -1 ? 0 : d[d[v].link].to[c];
                q.push(u);
            }
            if (v) FOR(c,26) if (!d[v].to[c])
                d[v].to[c] = d[d[v].link].to[c];
        }
    }
};

```

PalTree.h

Description: palindromic tree, computes number of occurrences of each palindrome within string

Time: $\mathcal{O}(N \Sigma)$ 25 lines

```

template<int SZ> struct PalTree {
    static const int sigma = 26;
    int s[SZ], len[SZ], link[SZ], to[SZ][sigma], oc[SZ];
    int n, last, sz;
    PalTree() { s[n++] = -1; link[0] = 1; len[1] = -1; sz = 2; }

    int getLink(int v) {
        while (s[n-len[v]-2] != s[n-1]) v = link[v];
        return v;
    }
};

```

```

void addChar(int c) {
    s[n++] = c;
    last = getLink(last);
    if (!to[last][c]) {
        len[sz] = len[last]+2;
        link[sz] = to[getLink(link[last])][c];
        to[last][c] = sz++;
    }
    last = to[last][c]; oc[last] ++;
}

void numOc() {
    vpi v; FOR(i,2,sz) v.pb({len[i],i});
    sort(rall(v)); trav(a,v) oc[link[a.s]] += oc[a.s];
}
};

```

SuffixArray.h

Description: ?

Time: $\mathcal{O}(N \log N)$ 43 lines

```

template<int SZ> struct suffixArray {
    const static int LGSZ = 33-__builtin_clz(SZ-1);
    pair<pi, int> tup[SZ];
    int sortIndex[LGSZ][SZ];
    int res[SZ];
    int len;

    suffixArray(string s) {
        this->len = (int)s.length();
        M00(i, len) tup[i] = MP(MP((int)s[i], -1), i);
        sort(tup, tup+len);
        int temp = 0;
        tup[0].F.F = 0;
        M00(i, 1, len) {
            if(s[tup[i].S] != s[tup[i-1].S]) temp++;
            tup[i].F.F = temp;
        }
        M00(i, len) sortIndex[0][tup[i].S] = tup[i].F.F;
        M00(i, 1, LGSZ) {
            M00(j, len) tup[j] = MP(MP(sortIndex[i-1][j], (j
                ↪+(1<<(i-1)<len)?sortIndex[i-1][j+(1<<(i-1))
                ↪]:-1), j));
            sort(tup, tup+len);
            int temp2 = 0;
            sortIndex[i][tup[0].S] = 0;
            M00(j, 1, len) {
                if(tup[j-1].F != tup[j].F) temp2++;
                sortIndex[i][tup[j].S] = temp2;
            }
        }
        M00(i, len) res[sortIndex[LGSZ-1][i]] = i;
    }

    int LCP(int x, int y) {
        if(x == y) return len - x;
        int ans = 0;
        M00d(i, LGSZ) {
            if(x >= len || y >= len) break;
            if(sortIndex[i][x] == sortIndex[i][y]) {
                x += (1<<i);
                y += (1<<i);
                ans += (1<<i);
            }
        }
        return ans;
    }
};

```

ReverseBW.h

Description: The Burrows-Wheeler Transform appends # to a string, sorts the rotations of the string in increasing order, and constructs a new string that contains the last character of each rotation. This function reverses the transform.

Time: $\mathcal{O}(N \log N)$ 8 lines

```

string reverseBW(string s) {
    vi nex(sz(s));
    vector<pair<char,int>> v; FOR(i,sz(s)) v.pb({s[i],i});
    sort(all(v)); FOR(i,sz(v)) nex[i] = v[i].s;
    int cur = nex[0]; string ret;
    for (; cur;cur = nex[cur]) ret += v[cur].f;
    return ret;
}

```

SuffixAutomaton.h

Description: constructs minimal DFA that recognizes all suffixes of a string

Time: $\mathcal{O}(N \log \Sigma)$ 73 lines

```

struct SuffixAutomaton {
    struct state {
        int len = 0, firstPos = -1, link = -1;
        bool isClone = 0;
        map<char, int> next;
        vi invLink;
    };

    vector<state> st;
    int last = 0;
    void extend(char c) {
        int cur = sz(st); st.eb();
        st[cur].len = st[last].len+1, st[cur].firstPos = st[cur].
            ↪len-1;
        int p = last;
        while (p != -1 && !st[p].next.count(c)) {
            st[p].next[c] = cur;
            p = st[p].link;
        }
        if (p == -1) {
            st[cur].link = 0;
        } else {
            int q = st[p].next[c];
            if (st[p].len+1 == st[q].len) {
                st[cur].link = q;
            } else {
                int clone = sz(st); st.pb(st[q]);
                st[clone].len = st[p].len+1, st[clone].isClone = 1;
                while (p != -1 && st[p].next[c] == q) {
                    st[p].next[c] = clone;
                    p = st[p].link;
                }
                st[q].link = st[cur].link = clone;
            }
        }
        last = cur;
    }

    void init(string s) {
        st.eb(); trav(x,s) extend(x);
        FOR(v,1,sz(st)) st[st[v].link].invLink.pb(v);
    }

    // APPLICATIONS
    void getAllOccur(vi& oc, int v) {
        if (!st[v].isClone) oc.pb(st[v].firstPos);
        trav(u,st[v].invLink) getAllOccur(oc,u);
    }

    vi allOccur(string s) {
        int cur = 0;
    }
};

```

```

    trav(x,s) {
        if (!st[cur].next.count(x)) return {};
        cur = st[cur].next[x];
    }
    vi oc; getAllOccur(oc,cur); trav(t,oc) t += 1-sz(s);
    sort(all(oc)); return oc;
}

v1 distinct;
ll getDistinct(int x) {
    if (distinct[x]) return distinct[x];
    distinct[x] = 1;
    trav(y,st[x].next) distinct[x] += getDistinct(y.s);
    return distinct[x];
}
ll numDistinct() { // # of distinct substrings, including
    ↪empty
    distinct.rsz(sz(st));
    return getDistinct(0);
}
ll numDistinct2() { // another way to get # of distinct
    ↪substrings
    ll ans = 1;
    FOR(i,1,sz(st)) ans += st[i].len-st[st[i].link].len;
    return ans;
}
};

```

SuffixTree.h

Description: Ukkonen's algorithm for suffix tree

Time: $\mathcal{O}(N \log \Sigma)$

61 lines

```

struct SuffixTree {
    string s; int node, pos;
    struct state {
        int fpos, len, link = -1;
        map<char,int> to;
        state(int fpos, int len) : fpos(fpos), len(len) {}
    };
    vector<state> st;
    int makeNode(int pos, int len) {
        st.pb(state(pos,len)); return sz(st)-1;
    }
    void goEdge() {
        while (pos > 1 && pos > st[st[node].to[s[sz(s)-pos]]].len)
            ↪{
                node = st[node].to[s[sz(s)-pos]];
                pos -= st[node].len;
            }
    }
    void extend(char c) {
        s += c; pos ++; int last = 0;
        while (pos) {
            goEdge();
            char edge = s[sz(s)-pos];
            int& v = st[node].to[edge];
            char t = s[st[v].fpos+pos-1];
            if (v == 0) {
                v = makeNode(sz(s)-pos,MOD);
                st[last].link = node; last = 0;
            } else if (t == c) {
                st[last].link = node;
                return;
            } else {
                int u = makeNode(st[v].fpos,pos-1);
                st[u].to[c] = makeNode(sz(s)-1,MOD); st[u].to[t] = v;
                st[v].fpos += pos-1; st[v].len -= pos-1;
                v = u; st[last].link = u; last = u;
            }
        }
    }
}

```

```

        if (node == 0) pos --;
        else node = st[node].link;
    }
}
void init(string _s) {
    makeNode(0,MOD); node = pos = 0;
    trav(c,_s) extend(c);
}
bool isSubstr(string _x) {
    string x; int node = 0, pos = 0;
    trav(c,_x) {
        x += c; pos ++;
        while (pos > 1 && pos > st[st[node].to[x[sz(x)-pos]]].len)
            ↪{
                node = st[node].to[x[sz(x)-pos]];
                pos -= st[node].len;
            }
        char edge = x[sz(x)-pos];
        if (pos == 1 && !st[node].to.count(edge)) return 0;
        int& v = st[node].to[edge];
        char t = s[st[v].fpos+pos-1];
        if (c != t) return 0;
    }
    return 1;
}
};

```

9.3 Misc

TandemRepeats.h

Description: Main-Lorentz algorithm, finds all (x,y) such that $s.substr(x,y-1) == s.substr(x+y,y-1)$

Time: $\mathcal{O}(N \log N)$

"z.h"

54 lines

```

struct StringRepeat {
    string S;
    vector<array<int,3>> al;
    // (t[0],t[1],t[2]) -> there is a repeating substring
    ↪starting at x
    // with length t[0]/2 for all t[1] <= x <= t[2]

    vector<array<int,3>> solveLeft(string s, int m) {
        vector<array<int,3>> v;

        vi v2 = getPrefix(string(s.begin()+m+1,s.end()),string(s.
            ↪begin(),s.begin()+m+1));
        string V = string(s.begin(),s.begin()+m+2); reverse(all(V))
            ↪; vi v1 = z(V); reverse(all(v1));

        FOR(i,m+1) if (v1[i]+v2[i] >= m+2-i) {
            int lo = max(1,m+2-i-v2[i]), hi = min(v1[i],m+1-i);
            lo = i-lo+1, hi = i-hi+1; swap(lo,hi);
            v.pb({2*(m+1-i),lo,hi});
        }

        return v;
    }

    void divi(int l, int r) {
        if (l == r) return;
        int m = (l+r)/2; divi(l,m); divi(m+1,r);

        string t = string(S.begin()+1,S.begin()+r+1);
        m = (sz(t)-1)/2;
        auto a = solveLeft(t,m);
        reverse(all(t));
        auto b = solveLeft(t,sz(t)-2-m);

        trav(x,a) al.pb({x[0],x[1]+1,x[2]+1});
    }
}

```

```

    trav(x,b) {
        int ad = r-x[0]+1;
        al.pb({x[0],ad-x[2],ad-x[1]});
    }
}

void init(string _S) {
    S = _S; divi(0,sz(S)-1);
}

vi genLen() { // min length of repeating substring starting
    ↪at each index
    priority_queue<pi,vpi,greater<pi>> m; m.push({MOD,MOD});
    vpi ins[sz(S)]; trav(a,al) ins[a[1]].pb({a[0],a[2]});
    vi len(sz(S));
    FOR(i,sz(S)) {
        trav(j,ins[i]) m.push(j);
        while (m.top().s < i) m.pop();
        len[i] = m.top().f;
    }
    return len;
}
};

```