



University of Virginia

Ground Zero

Edward Lue, Richard Wang, Nicholas Winschel

ICPC North America Championship 2023

May 29, 2023

1 Contest

2 Mathematics

3 Data structures

4 Numerical

5 Number Theory

6 Combinatorial

7 Graphs

8 Geometry

9 Strings

10Various

Contest (1)

template.cpp 44 lines

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

using ll = long long;
using db = long double; // or double if tight TL
using str = string;

using pi = pair<int,int>;
#define mp make_pair
#define f first
#define s second

#define tcT template<class T
tcT> using V = vector<T>;
tcT, size_t SZ> using AR = array<T,SZ>;
using vi = V<int>;
using vb = V<bool>;
using vpi = V<pi>;

#define sz(x) int((x).size())
#define all(x) begin(x), end(x)
#define sor(x) sort(all(x))
#define rsz resize
#define pb push_back
#define ft front()
#define bk back()

#define FOR(i,a,b) for (int i = (a); i < (b); ++i)
#define F0R(i,a) FOR(i,0,a)
#define ROF(i,a,b) for (int i = (b)-1; i >= (a); --i)
#define R0F(i,a) ROF(i,0,a)
#define rep(a) F0R(_,a)
#define each(a,x) for (auto& a: x)

const int MOD = 1e9+7;
const db PI = acos((db)-1);
mt19937 rng(0); // or mt19937_64

tcT> bool ckmin(T& a, const T& b) {
    return b < a ? a = b, 1 : 0; } // set a = min(a,b)
tcT> bool ckmax(T& a, const T& b) {
    return a < b ? a = b, 1 : 0; } // set a = max(a,b)

int main() { cin.tie(0)->sync_with_stdio(0); }
```

template .bashrc hash troubleshoot

```
1 .bashrc 3 lines
1 alias clr="printf '\33c'"
1 co() { g++ -std=c++17 -O2 -Wall -Wextra -Wshadow -Wconversion -o $1
    ↪$1.cpp; }
2 run() { co $1 && ./$1; }
3 hash.sh 1 lines
3 cpp -dD -P -fpreprocessed|tr -d '[:space:]'|md5sum|cut -c-6
7 troubleshoot.txt 75 lines
8 General:
8 Write down most of your thoughts, even if you're not sure
10 whether they're useful.
10 Give your variables (and files) meaningful names.
10 Stay organized and don't leave papers all over the place!
10 You should know what your code is doing ...
16 Pre-submit:
16 Write a few simple test cases if sample is not enough.
20 Are time limits close? If so, generate max cases.
20 Is the memory usage fine?
20 Could anything overflow?
22 Remove debug output.
22 Make sure to submit the right file.
```

```
Wrong answer:
Print your solution! Print debug output as well.
Read the full problem statement again.
Have you understood the problem correctly?
Are you sure your algorithm works?
Try writing a slow (but correct) solution.
Can your algorithm handle the whole range of input?
Did you consider corner cases (ex. n=1)?
Is your output format correct? (including whitespace)
Are you clearing all data structures between test cases?
Any uninitialized variables?
Any undefined behavior (array out of bounds)?
Any overflows or NaNs (or shifting ll by >=64 bits)?
Confusing N and M, i and j, etc.?
Confusing ++i and i++?
Return vs continue vs break?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some test cases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Rewrite your solution from the start or let a teammate do it.
```

```
Geometry:
Work with ints if possible.
Correctly account for numbers close to (but not) zero. Related:
for functions like acos make sure absolute val of input is not
(slightly) greater than one.
Correctly deal with vertices that are collinear, concyclic,
coplanar (in 3D), etc.
Subtracting a point from every other (but not itself)?
```

```
Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).
```

```
Time limit exceeded:
Do you have any possible infinite loops?
What's your complexity? Large TL does not mean that something
simple (like NlogN) isn't intended.
Are you copying a lot of unnecessary data? (References)
Avoid vector, map. (use arrays/unordered_map)
How big is the input and output? (consider FastIO)
What do your teammates think about your algorithm?
```

Calling count() on multiset?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all data structures between test cases?
If using pointers try BumpAllocator.

Mathematics (2)

Cramer's Rule: given an equation $Ax = b$, the solution to a variable x_i is given by

$$x_i = \frac{\det A'_i}{\det A}$$

where A'_i is A with the i 'th column replaced by b .

2.1 Trigonometry

$$\sin(v + w) = \sin v \cos w + \cos v \sin w$$

$$\cos(v + w) = \cos v \cos w - \sin v \sin w$$

$$\tan(v + w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$

$$\sin v + \sin w = 2 \sin \frac{v + w}{2} \cos \frac{v - w}{2}$$

$$\cos v + \cos w = 2 \cos \frac{v + w}{2} \cos \frac{v - w}{2}$$

$$(V + W) \tan(v - w)/2 = (V - W) \tan(v + w)/2$$

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

2.2 Geometry

2.2.1 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: $\frac{a + b}{a - b} = \frac{\tan \frac{\alpha + \beta}{2}}{\tan \frac{\alpha - \beta}{2}}$

2.2.2 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° , $ef = ac + bd$, and $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

2.3 Derivatives/Integrals

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

Integration by parts:

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

2.4 Sums

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

2.5 Series

$$\begin{array}{l} \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots, \quad (-1 < x \leq 1) \\ \sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \dots, \quad (-1 \leq x \leq 1) \\ \sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots, \quad (-\infty < x < \infty) \\ \cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots, \quad (-\infty < x < \infty) \end{array}$$

Data structures (3)

HashMap.h

Description: Hash map with similar API as unordered_map. Initial capacity must be a power of 2 if provided.

Usage: ht<int,int> h({},{},{},{},{1<<16});

Memory: ~1.5x unordered map

Time: ~3x faster than unordered map

<ext/pb_ds/assoc.container.hpp> 5872b2b, 9 lines

```
using namespace __gnu_pbds;
struct chash {
    const uint64_t C = 1l(4e18*acos(0))+71; // large odd number
    const int RANDOM = rng();
    ll operator()(ll x) const { return __builtin_bswap64((x^RANDOM)*C);
        ↪ }
};
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) {
    auto it = u.find(x); return it == end(u) ? 0 : it->s; }
```

OrderStatisticTree.h

Description: order_of_key, find_by_order (order = num less)

Time: $\mathcal{O}(\log N)$ cd2981, 6 lines

```
#include <bits/extc++.h>
using namespace __gnu_pbds;

template<class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
```

LineContainer.h

Description: Add lines of the form $ax+b$, query maximum y -coordinate for any x .

Time: $\mathcal{O}(\log N)$ df7386, 29 lines

```
using T = ll; const T INF = LLONG_MAX; // a/b rounded down
// ll fdiv(ll a, ll b) { return a/b-((a*b)<0&&a%b); }

bool _Q = 0;
struct Line {
    T a, b; mutable T lst;
    T eval(T x) const { return a*x+b; }
    bool operator<(const Line&o) const{return _Q?lst<o.lst:a<o.a;}
    T last_gre(const Line& o) const { assert(a <= o.a);
        // greatest x s.t. a*x+b >= o.a*x+o.b
        return lst=(a==o.a?(b>=o.b?INF:-INF):fdiv(b-o.b,o.a-a));}
};
```

```
struct LineContainer: multiset<Line> {
    bool isect(iterator it) { auto n_it = next(it);
        if (n_it == end()) return it->lst = INF, 0;
        return it->last_gre(*n_it) >= n_it->lst; }
    void add(T a, T b) {
        auto it = ins({a,b,0}); while (isect(it)) erase(next(it));
        if (it == begin()) return;
        if (isect(--it)) erase(next(it)), isect(it);
        while (it != begin()) {
            --it; if (it->lst < next(it)->lst) break;
            erase(next(it)); isect(it); }
    }
    T qmax(T x) { assert(!empty());
        _Q = 1; T res = lb({0,0,x})->eval(x); _Q = 0;
        return res; }
};
```

LineContainerDeque.h

Description: LineContainer assuming both slopes and queries monotonic.

Time: $\mathcal{O}(1)$

"LineContainer.h" e56463, 33 lines

```
struct LCdeque : deque<Line> {
    void addBack(Line L) { // assume nonempty
        while (1) {
            auto a = bk; pop_back(); a.lst = a.last_gre(L);
            if (size() && bk.lst >= a.lst) continue;
            pb(a); break;
        }
        L.lst = INF; pb(L);
    }
    void addFront(Line L) {
        while (1) {
            if (!size()) { L.lst = INF; break; }
            if ((L.lst = L.last_gre(ft)) >= ft.lst) pop_front();
            else break;
        }
        push_front(L);
    }
    void add(T a, T b) { // line goes to one end of deque
        if (!size() || a <= ft.a) addFront({a,b,0});
        else assert(a >= bk.a), addBack({a,b,0});
    }
    int ord = 0; // 1 = x's come in increasing order, -1 = decreasing
        ↪ order
    T query(T x) {
        assert(ord);
        if (ord == 1) {
            while (ft.lst < x) pop_front();
            return ft.eval(x);
        } else {
            while(size())&l&prev(prev(end())-<lst>=x)pop_back();
            return bk.eval(x);
        }
    }
};
```

```
}
}
};
```

RMQ.h

Description: 1D range minimum query. If TL is an issue, use arrays instead of vectors and store values instead of indices.

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

Time: $\mathcal{O}(1)$ a3f881, 19 lines

```
tcT> struct RMQ { // floor(log_2(x))
    int level(int x) { return 31-__builtin_clz(x); }
    V<T> v; V<vi> jmp;
    int cmb(int a, int b) {
        return v[a]==v[b]?min(a,b):(v[a]<v[b]?a:b); }
    void init(const V<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));
            F0R(i,sz(jmp[j])) jmp[j][i] = cmb(jmp[j-1][i],
                jmp[j-1][i+(1<<(j-1))]);
        }
    }
    int index(int l, int r) { // kat is rex instead
        assert(l <= r); int d = level(r-l+1);
        return cmb(jmp[d][l], jmp[d][r-(1<<d)+1]); }
    T query(int l, int r) { return v[index(l,r)]; }
};
```

SegmentTree.h

Description: 1D point update and range query where cmb is any associative operation. seg[1]==query(0,N-1).

Time: $\mathcal{O}(\log N)$ 1630f3, 18 lines

```
tcT> struct SegTree { // cmb(ID,b) = b
    const T ID{}; T cmb(T a, T b) { return a+b; }
    int n; V<T> seg;
    void init(int _n) { // upd, query also work if n = _n
        for (n = 1; n < _n; ) n *= 2;
        seg.assign(2*n,ID); }
    void pull(int p) { seg[p] = cmb(seg[2*p],seg[2*p+1]); }
    void upd(int p, T val) { // set val at position p
        seg[p += n] = val; for (p /= 2; p; p /= 2) pull(p); }
    T query(int l, int r) { // zero-indexed, inclusive
        T ra = ID, rb = ID;
        for (l += n, r += n+1; l < r; l /= 2, r /= 2) {
            if (l&1) ra = cmb(ra,seg[l++]);
            if (r&1) rb = cmb(seg[--r],rb);
        }
        return cmb(ra,rb);
    }
};
```

LazySegmentTree.h

Description: 1D range increment and sum query.

Time: $\mathcal{O}(\log N)$ 78a06d, 26 lines

```
tcT, int SZ> struct LazySeg {
    static_assert(pct(SZ) == 1); // SZ must be power of 2
    const T ID{}; T cmb(T a, T b) { return a+b; }
    T seg[2*SZ], lazy[2*SZ];
    LazySeg() { F0R(i,2*SZ) seg[i] = lazy[i] = ID; }
    void push(int ind, int L, int R) {
        seg[ind] += (R-L+1)*lazy[ind]; // dependent on operation
        if (L != R) F0R(i,2) lazy[2*ind+i] += lazy[ind];
        lazy[ind] = 0;
    } // recalc values for current node
    void pull(int ind){seg[ind]=cmb(seg[2*ind],seg[2*ind+1]);}
    void build() { ROF(i,1,SZ) pull(i); }
    void upd(int lo,int hi,T inc,int ind=1,int L=0, int R=SZ-1) {
        push(ind,L,R); if (hi < L || L < lo) return;
        if (lo <= L && R <= hi) {
            lazy[ind] = inc; push(ind,L,R); return; }
        int M = (L+R)/2; upd(lo,hi,inc,2*ind,L,M);
        upd(lo,hi,inc,2*ind+1,M+1,R); pull(ind);
    }
    T query(int lo, int hi, int ind=1, int L=0, int R=SZ-1) {
        push(ind,L,R); if (lo > R || L > hi) return ID;
        if (lo <= L && R <= hi) return seg[ind];
        int M = (L+R)/2; return cmb(query(lo,hi,2*ind,L,M),
```

```
        query(lo,hi,2*ind+1,M+1,R));
    }
};
```

PSeg.h

Description: Persistent min segtree with lazy updates, no propagation. If making d a vector then save the results of upd and build in local variables first to avoid issues when vector resizes in C++14 or lower.

Memory: $\mathcal{O}(N + Q \log N)$

8f37fa, 45 lines

```
tcT, int SZ> struct pseg {
    static const int LIM = 2e7;
    struct node {
        int l, r; T val = 0, lazy = 0;
        void inc(T x) { lazy += x; }
        T get() { return val+lazy; }
    };
    node d[LIM]; int nex = 0;
    int copy(int c) { d[nex] = d[c]; return nex++; }
    T cmb(T a, T b) { return min(a,b); }
    void pull(int c) { d[c].val =
        cmb(d[d[c].l].get(), d[d[c].r].get()); }
    T query(int c, int lo, int hi, int L, int R) {
        if (lo <= L && R <= hi) return d[c].get();
        if (R < lo || hi < L) return MOD;
        int M = (L+R)/2;
        return d[c].lazy+cmb(query(d[c].l,lo,hi,L,M),
            query(d[c].r,lo,hi,M+1,R));
    }
    int upd(int c, int lo, int hi, T v, int L, int R) {
        if (R < lo || hi < L) return c;
        int x = copy(c);
        if (lo <= L && R <= hi) { d[x].inc(v); return x; }
        int M = (L+R)/2;
        d[x].l = upd(d[x].l,lo,hi,v,L,M);
        d[x].r = upd(d[x].r,lo,hi,v,M+1,R);
        pull(x); return x;
    }
    int build(const V<T>& arr, int L, int R) {
        int c = nex++;
        if (L == R) {
            if (L < sz(arr)) d[c].val = arr[L];
            return c;
        }
        int M = (L+R)/2;
        d[c].l = build(arr,L,M), d[c].r = build(arr,M+1,R);
        pull(c); return c;
    }
    vi loc;
    void upd(int lo, int hi, T v) {
        loc.pb(upd(loc.bk,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(const V<T>&arr) {loc.pb(build(arr,0,SZ-1));}
};
```

Treap.h

Description: Easy BBST. Use split and merge to implement insert and delete.

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

bdb758, 65 lines

```
using pt = struct tnode*;
struct tnode {
    int pri, val; pt c[2]; // essential
    int sz; ll sum; // for range queries
    bool flip = 0; // lazy update
    tnode(int _val) {
        pri = rng(); sum = val = _val;
        sz = 1; c[0] = c[1] = nullptr;
    }
    ~tnode() { F0R(i,2) delete c[i]; }
};
int getsz(pt x) { return x?x->sz:0; }
ll getsum(pt x) { return x?x->sum:0; }
pt prop(pt x) { // lazy propagation
    if (!x || !x->flip) return x;
    swap(x->c[0],x->c[1]);
    x->flip = 0; F0R(i,2) if (x->c[i]) x->c[i]->flip ^= 1;
    return x;
}
pt calc(pt x) {
```

PSeg Treap BIT2DOff Matrix Determinant IntDeterminant

```
pt a = x->c[0], b = x->c[1];
assert(!x->flip); prop(a), prop(b);
x->sz = 1+getsz(a)+getsz(b);
x->sum = x->val+getsum(a)+getsum(b);
return x;
}
void tour(pt x, vi& v) { // print values of nodes,
    if (!x) return; // inorder traversal
    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) { // 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) { // keys in l < keys in r
    if (!l || !r) return 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); }
```

BIT2DOff.h

Description: point update and rectangle sum with offline 2D BIT. For each of the points to be updated, $x \in (0, SZ)$ and $y \neq 0$.

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

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

962052, 34 lines

```
template<class T, int SZ> struct OffBIT2D {
    bool mode = 0; // mode = 1 -> initialized
    vpi todo; // locations of updates to process
    int cnt[SZ], st[SZ];
    vi val; vector<T> bit; // store all BITs in single vector
    void init() { assert(!mode); mode = 1;
        int lst[SZ]; F0R(i,SZ) lst[i] = cnt[i] = 0;
        sort(all(todo),[](const pi& a, const pi& b) {
            return a.s < b.s; });
        each(t,todo) for (int x = t.f; x < SZ; x += x&-x)
            if (lst[x] != t.s) lst[x] = t.s, cnt[x] ++;
        int sum = 0; F0R(i,SZ) lst[i] = 0, st[i] = (sum += cnt[i]);
        val.rsz(sum); bit.rsz(sum); reverse(all(todo));
        each(t,todo) for (int x = t.f; x < SZ; x += x&-x)
            if (lst[x] != t.s) lst[x] = t.s, val[--st[x]] = t.s;
    }
    int rank(int y, int l, int r) {
        return ub(begin(val)+1,begin(val)+r,y)-begin(val)-1; }
    void UPD(int x, int y, T t) {
        for (y = rank(y,st[x],st[x]+cnt[x]); y <= cnt[x]; y += y&-y)
            bit[st[x]+y-1] += t; }
    void upd(int x, int y, T t) {
        if (!mode) todo.pb({x,y});
        else for (;x<SZ;x+=x&-x) UPD(x,y,t); }
    int QUERY(int x, int y) { T res = 0;
        for (y = rank(y,st[x],st[x]+cnt[x]); y; y -= y&-y) res += bit[st[
            ↵x]+y-1];
        return res; }
    T query(int x, int y) { assert(mode);
```

```
    T res = 0; for (;x;x-=x&-x) res += QUERY(x,y);
    return res; }
    T query(int xl, int xr, int yl, int yr) {
        return query(xr,yr)-query(xl-1,yr)
            -query(xr,yl-1)+query(xl-1,yl-1); }
};
```

Numerical (4)

4.1 Matrices

Matrix.h

Description: 2D matrix operations.

"../number-theory (11.1)/Modular Arithmetic/ModInt.h"

b18e29, 21 lines

```
using T = mi;
using Mat = V<V<T>>; // use array instead if tight TL

Mat makeMat(int r, int c) { return Mat(r,V<T>(c)); }
Mat makeId(int n) {
    Mat m = makeMat(n,n); F0R(i,n) m[i][i] = 1;
    return m;
}
Mat operator*(const Mat& a, const Mat& b) {
    int x = sz(a), y = sz(a[0]), z = sz(b[0]);
    assert(y == sz(b)); Mat c = makeMat(x,z);
    F0R(i,x) F0R(j,y) F0R(k,z) c[i][k] += a[i][j]*b[j][k];
    return c;
}
Mat& operator*=(Mat& a, const Mat& b) { return a = a*b; }
Mat pow(Mat m, ll p) {
    int n = sz(m); assert(n == sz(m[0]) && p >= 0);
    Mat res = makeId(n);
    for (; p; p /= 2, m *= m) if (p&1) res *= m;
    return res;
}
```

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.

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

bd5cec, 15 lines

```
double det(vector<vector<double>>& a) {
    int n = sz(a); double res = 1;
    rep(i,0,n) {
        int b = i;
        rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
        if (i != b) swap(a[i], a[b]), res *= -1;
        res *= a[i][i];
        if (res == 0) return 0;
        rep(j,i+1,n) {
            double v = a[j][i] / a[i][i];
            if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
        }
    }
    return res;
}
```

IntDeterminant.h

Description: Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.

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

3313dc, 18 lines

```
const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
    int n = sz(a); ll ans = 1;
    rep(i,0,n) {
        rep(j,i+1,n) {
            while (a[j][i] != 0) { // gcd step
                ll t = a[i][i] / a[j][i];
                if (t) rep(k,i,n)
                    a[i][k] = (a[i][k] - a[j][k] * t) % mod;
                swap(a[i], a[j]);
                ans *= -1;
            }
        }
        ans = ans * a[i][i] % mod;
        if (!ans) return 0;
    }
    return (ans + mod) % mod;
```

```
}
```

SolveLinear.h

Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.

Time: $\mathcal{O}\left(n^2m\right)$

44c9ab, 38 lines

```
typedef vector<double> vd;
const double eps = 1e-12;
```

```
int solveLinear(vector<vd>& A, vd& b, vd& x) {
    int n = sz(A), m = sz(x), rank = 0, br, bc;
    if (n) assert(sz(A[0]) == m);
    vi col(m); iota(all(col), 0);
```

```
    rep(i,0,n) {
        double v, bv = 0;
        rep(r,i,n) rep(c,i,m)
            if ((v = fabs(A[r][c])) > bv)
                br = r, bc = c, bv = v;
        if (bv <= eps) {
            rep(j,i,n) if (fabs(b[j]) > eps) return -1;
            break;
        }
```

```
        swap(A[i], A[br]);
        swap(b[i], b[br]);
        swap(col[i], col[bc]);
        rep(j,0,n) swap(A[j][i], A[j][bc]);
        bv = 1/A[i][i];
        rep(j,i+1,n) {
            double fac = A[j][i] * bv;
            b[j] -= fac * b[i];
            rep(k,i+1,m) A[j][k] -= fac*A[i][k];
        }
        rank++;
    }
```

```
    x.assign(m, 0);
    for (int i = rank; i--;) {
        b[i] /= A[i][i];
        x[col[i]] = b[i];
        rep(j,0,i) b[j] -= A[j][i] * b[i];
    }
    return rank; // (multiple solutions if rank < m)
}
```

SolveLinear2.h

Description: To get all uniquely determined values of x back from SolveLinear, make the following changes:

"SolveLinear.h" 08e495, 7 lines

```
rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
    rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
    x[col[i]] = b[i] / A[i][i];
fail:; }
```

SolveLinearBinary.h

Description: Solves $Ax = b$ over \mathbb{F}_2 . If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b .

Time: $\mathcal{O}\left(n^2m\right)$

fa2d7a, 34 lines

```
typedef bitset<1000> bs;
```

```
int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
    int n = sz(A), rank = 0, br;
    assert(m <= sz(x));
    vi col(m); iota(all(col), 0);
    rep(i,0,n) {
        for (br=i; br<n; ++br) if (A[br].any()) break;
        if (br == n) {
            rep(j,i,n) if(b[j]) return -1;
            break;
        }
        int bc = (int)A[br]._Find_next(i-1);
        swap(A[i], A[br]);
        swap(b[i], b[br]);
        swap(col[i], col[bc]);
        rep(j,0,n) if (A[j][i] != A[j][bc]) {
```

```
            A[j].flip(i); A[j].flip(bc);
        }
        rep(j,i+1,n) if (A[j][i]) {
            b[j] ^= b[i];
            A[j] ^= A[i];
        }
        rank++;
    }

    x = bs();
    for (int i = rank; i--;) {
        if (!b[i]) continue;
        x[col[i]] = 1;
        rep(j,0,i) b[j] ^= A[j][i];
    }
    return rank; // (multiple solutions if rank < m)
}
```

MatrixInv.h

Description: Uses gaussian elimination to convert into reduced row echelon form and calculates determinant. For determinant via arbitrary modulus, use a modified form of the Euclidean algorithm because modular inverse may not exist. If you have computed $A^{-1} \pmod{p^k}$, then the inverse $\pmod{p^{2k}}$ is $A^{-1}(2I - AA^{-1})$.

Time: $\mathcal{O}\left(N^3\right)$, determinant of 1000×1000 matrix of modints in 1 second if you reduce # of operations by half

73ec43, 38 lines

```
const db EPS = 1e-9; // adjust?
int getRow(V<V<db>>& m, int R, int i, int nex) {
    pair<db,int> bes{0,-1}; // find row with max abs value
    FOR(j,nex,R) ckmax(bes,{abs(m[j][i]),j});
    return bes.f < EPS ? -1 : bes.s; }
int getRow(V<vml>& m, int R, int i, int nex) {
    FOR(j,nex,R) if (m[j][i] != 0) return j;
    return -1; }
pair<T,int> gauss(Mat& m) { // convert to reduced row echelon form
    if (!sz(m)) return {1,0};
    int R = sz(m), C = sz(m[0]), rank = 0, nex = 0;
    T prod = 1; // determinant
    FOR(i,C) {
        int row = getRow(m,R,i,nex);
        if (row == -1) { prod = 0; continue; }
        if (row != nex) prod *= -1, swap(m[row],m[nex]);
        prod *= m[nex][i]; rank++;
        T x = 1/m[nex][i]; FOR(k,i,C) m[nex][k] *= x;
        FOR(j,R) if (j != nex) {
            T v = m[j][i]; if (v == 0) continue;
            FOR(k,i,C) m[j][k] -= v*m[nex][k];
        }
        nex++;
    }
    return {prod,rank};
}
Mat inv(Mat m) {
    int R = sz(m); assert(R == sz(m[0]));
    Mat x = makeMat(R,2*R);
    FOR(i,R) {
        x[i][i+R] = 1;
        FOR(j,R) x[i][j] = m[i][j];
    }
    if (gauss(x).s != R) return Mat();
    Mat res = makeMat(R,R);
    FOR(i,R) FOR(j,R) res[i][j] = x[i][j+R];
    return res;
}
```

MatrixTree.h

Description: Kirchhoff's Matrix Tree Theorem. Given adjacency matrix, calculates # of spanning trees.

"MatrixInv.h" 48363d, 11 lines

```
T numSpan(const Mat& m) {
    int n = sz(m); Mat res = makeMat(n-1,n-1);
    FOR(i,n) FOR(j,i+1,n) {
        mi ed = m[i][j]; res[i][i] += ed;
        if (j != n-1) {
            res[j][j] += ed;
            res[i][j] -= ed, res[j][i] -= ed;
        }
    }
    return gauss(res).f;
```

```
}
```

ShermanMorrison.h

Description: Calculates $(A + uv^T)^{-1}$ given $B = A^{-1}$. Not invertible if sum=0.

"MatrixInv.h" 3a3f34, 7 lines

```
void ad(Mat& B, const V<T>& u, const V<T>& v) {
    int n = sz(A); V<T> x(n), y(n);
    FOR(i,n) FOR(j,n)
        x[i] += B[i][j]*u[j], y[j] += v[i]*B[i][j];
    T sum = 1; FOR(i,n) FOR(j,n) sum += v[i]*B[i][j]*u[j];
    FOR(i,n) FOR(j,n) B[i][j] -= x[i]*y[j]/sum;
}
```

Tridiagonal.h

Description: $x = \text{tridiagonal}(d, p, q, b)$ solves the equation system

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} d_0 & p_0 & 0 & 0 & \cdots & 0 \\ q_0 & d_1 & p_1 & 0 & \cdots & 0 \\ 0 & q_1 & d_2 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & q_{n-3} & d_{n-2} & p_{n-2} \\ 0 & 0 & \cdots & 0 & q_{n-2} & d_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix}.$$

This is useful for solving problems on the type

$$a_i = b_i a_{i-1} + c_i a_{i+1} + d_i, 1 \leq i \leq n,$$

where a_0, a_{n+1}, b_i, c_i and d_i are known. a can then be obtained from

$$\{a_i\} = \text{tridiagonal}(\{1, -1, -1, \dots, -1, 1\}, \{0, c_1, c_2, \dots, c_n\}, \{b_1, b_2, \dots, b_n, 0\}, \{a_0, d_1, d_2, \dots, d_n, a_{n+1}\}).$$

Fails if the solution is not unique.

If $|d_i| > |p_i| + |q_{i-1}|$ for all i , or $|d_i| > |p_{i-1}| + |q_i|$, or the matrix is positive definite, the algorithm is numerically stable and neither tr nor the check for diag[] == 0 is needed.

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

8f9fa8, 26 lines

```
typedef double T;
vector<T> tridiagonal(vector<T> diag, const vector<T>& super,
    const vector<T>& sub, vector<T> b) {
    int n = sz(b); vi tr(n);
    rep(i,0,n-1) {
        if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i] == 0
            b[i+1] -= b[i] * diag[i+1] / super[i];
            if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];
            diag[i+1] = sub[i]; tr[i+1] = 1;
        } else {
            diag[i+1] -= super[i]*sub[i]/diag[i];
            b[i+1] -= b[i]*sub[i]/diag[i];
        }
    }
    for (int i = n; i--;) {
        if (tr[i]) {
            swap(b[i], b[i-1]);
            diag[i-1] = diag[i];
            b[i] /= super[i-1];
        } else {
            b[i] /= diag[i];
            if (i) b[i-1] -= b[i]*super[i-1];
        }
    }
    return b;
}
```

SparseDet.h

Description: Tries to find characteristic equation of matrix -> determinant.

"LinRec.h" a2ca78, 14 lines

```
mi sparseDet(int N, V<pair<pi,mi>> A) { // nonzero entries of matrix,
    ↪ no repeats
    auto gen = []() { return rng()%(MOD-1)+1; };
    vmi l(N), r(N), seq(2*N); FOR(i,N) l[i] = gen(), r[i] = gen();
    FOR(i,2*N) { // consider  $l * A^i * r$ , recurrence satisfies
        ↪ characteristic equation
        FOR(j,N) seq[i] += l[j]*r[j];
        vmi R(N); each(t,A) R[t.f.s] += r[t.f.f]*t.s;
        swap(r,R);
    }
    LinRec L; L.init(seq); // hopefully found characteristic equation
    if (L.C.bk == 0) return 0; // 0 is root of characteristic equation
```

```

    if (sz(L.C) != N) return sparseDet(N,ed); // keep trying ...
    mi res = L.C.bk; if (!(N&1)) res *= -1;
    return res;
};

```

4.2 Polynomials and recurrences

Poly.h

Description: Basic poly ops including division. Can replace T with double, complex.

"../number-theory (11.1)/Modular Arithmetic/ModInt.h" cd218a, 73 lines

```

using T = mi; using poly = V<T>;
void remz(poly& p) { while (sz(p)&&p.bk==T(0)) p.pop_back(); }
poly REMZ(poly p) { remz(p); return p; }
poly rev(poly p) { reverse(all(p)); return p; }
poly shift(poly p, int x) {
    if (x >= 0) p.insert(begin(p),x,0);
    else assert(sz(p)+x >= 0), p.erase(begin(p),begin(p)-x);
    return p;
}
poly RSZ(const poly& p, int x) {
    if (x <= sz(p)) return poly(begin(p),begin(p)+x);
    poly q = p; q.rsz(x); return q; }
T eval(const poly& p, T x) { // evaluate at point x
    T res = 0; R0F(i,sz(p)) res = x*res+p[i];
    return res; }
poly dif(const poly& p) { // differentiate
    poly res; FOR(i,1,sz(p)) res.pb(T(i)*p[i]);
    return res; }
poly integ(const poly& p) { // integrate
    static poly invs{0,1};
    for (int i = sz(invs); i <= sz(p); ++i)
        invs.pb(-MOD/i*invs[MOD%i]);
    poly res(sz(p)+1); F0R(i,sz(p)) res[i+1] = p[i]*invs[i+1];
    return res;
}

poly& operator+=(poly& l, const poly& r) {
    l.rsz(max(sz(l),sz(r))); F0R(i,sz(r)) l[i] += r[i];
    return l; }
poly& operator-=(poly& l, const poly& r) {
    l.rsz(max(sz(l),sz(r))); F0R(i,sz(r)) l[i] -= r[i];
    return l; }
poly& operator*=(poly& l, const T& r) { each(t,l) t *= r;
    return l; }
poly& operator/=(poly& l, const T& r) { each(t,l) t /= r;
    return l; }
poly operator+(poly l, const poly& r) { return l += r; }
poly operator-(poly l, const poly& r) { return l -= r; }
poly operator*(poly l) { each(t,l) t *= -1; return l; }
poly operator*(poly l, const T& r) { return l *= r; }
poly operator*(const T& r, const poly& l) { return l*r; }
poly operator/(poly l, const T& r) { return l /= r; }
poly operator*(const poly& l, const poly& r) {
    if (!min(sz(l),sz(r))) return {};
    poly x(sz(l)+sz(r)-1);
    F0R(i,sz(l)) F0R(j,sz(r)) x[i+j] += l[i]*r[j];
    return x;
}
poly& operator*=(poly& l, const poly& r) { return l = l*r; }

pair<poly,poly> quoRemSlow(poly a, poly b) {
    remz(a); remz(b); assert(sz(b));
    T lst = b.bk, B = T(1)/lst; each(t,a) t *= B;
    each(t,b) t *= B;
    poly q(max(sz(a)-sz(b)+1,0));
    for (int dif; (dif=sz(a)-sz(b)) >= 0; remz(a)) {
        q[dif] = a.bk; F0R(i,sz(b)) a[i+dif] -= q[dif]*b[i]; }
    each(t,a) t *= lst;
    return {q,a}; // quotient, remainder
}
poly operator%(const poly& a, const poly& b) {
    return quoRemSlow(a,b).s; }
T resultant(poly a, poly b) { // R(A,B)
    // =b_m^n*prod_{j=1}^m A(mu_j)
    // =b_m^n a_n^m * prod_{i=1}^n prod_{j=1}^m (mu_j-lambda_i)
    // =(-1)^m {a_n^m} a_n^m * prod_{i=1}^n B(lambda_i)
    // =(-1)^{nm} R(B,A)
    // Also, R(A,B)=b_m^{deg(A)-deg(A-CB)} R(A-CB,B)
    int ad = sz(a)-1, bd = sz(b)-1;
    if (bd <= 0) return bd < 0 ? 0 : pow(b.bk,ad);

```

```

    int pw = ad; a = a&b; pw -= (ad = sz(a)-1);
    return resultant(b,a)*pow(b.bk,pw)*T((bd&ad&1)?-1:1);
}

```

PolyRoots.h

Description: Finds the real roots of a polynomial.

Usage: poly_roots({{2,-3,1}},-1e9,1e9) // solve x²-3x+2 = 0

Time: $\mathcal{O}\left(N^2 \log(1/\epsilon)\right)$

"Poly.h" c9127a, 20 lines

```

typedef db T;
poly polyRoots(poly p, T xmin, T 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));
    poly ret;
    F0R(i,sz(dr)-1) {
        T l = dr[i], h = dr[i+1];
        bool sign = eval(p,l) > 0;
        if (sign^(eval(p,h) > 0)) {
            F0R(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;
}

```

PolyInterpolate.h

Description: n points determine unique polynomial of degree $\leq n-1$. For numerical precision pick $v[k].f = c * \cos(k/(n-1) * \pi)$, $k = 0 \dots n-1$.

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

"Poly.h" aada3a, 8 lines

```

poly interpolate(V<pair<T,T>> v) {
    poly res, tmp{1};
    F0R(i,sz(v)) { T prod = 1; // add one point at a time
        F0R(j,i) v[i].s -= prod*v[j].s, prod *= v[i].f-v[j].f;
        v[i].s /= prod; res += v[i].s*tmp; tmp *= poly{-v[i].f,1};
    } // add multiple of (x-v[0].f)*(x-v[1].f)*...*(x-v[i-1].f)
    return res;
}

```

LinearRecurrence.h

Description: Berlekamp-Massey. Computes linear recurrence C of order N for sequence s of $2N$ terms. $C[0] = 1$ and for all $i \geq sz(C)-1$, $\sum_{j=0}^{sz(C)-1} C[j]s[i-j] = 0$.

Usage: LinRec L; L.init({0,1,1,2,3}); L.eval(5); L.eval(6); // 5, 8

Time: init $\Rightarrow \mathcal{O}(N|C|)$, eval $\Rightarrow \mathcal{O}(|C|^2 \log p)$ or faster with FFT

"Poly.h" 39ea71, 29 lines

```

struct LinRec {
    poly s, C, rC;
    void BM() {
        int x = 0; T b = 1;
        poly B; B = C = {1}; // B is fail vector
        F0R(i,sz(s)) { // update C after adding a term of s
            ++x; int L = sz(C), M = i+3-L;
            T d = 0; F0R(j,L) d += C[j]*s[i-j]; // [D^i]C*s
            if (d.v == 0) continue; // [D^i]C*s=0
            poly _C = C; T coef = d*inv(b);
            C.rsz(max(L,M)); F0R(j,sz(B)) C[j+x] -= coef*B[j];
            if (L < M) B = _C, b = d, x = 0;
        }
    }
    void init(const poly& _s) {
        s = _s; BM();
        rC = C; reverse(all(rC));
        C.erase(begin(C)); each(t,C) t *= -1;
    } // now s[i]=sum_{j=0}^{sz(C)-1} C[j]*s[i-j-1]
    poly getPow(ll p) { // get x^p mod rC
        if (p == 0) return {1};
        poly r = getPow(p/2); r = (r*r)%rC;
        return p&1? (r*poly{0,1})%rC:r;
    }
    T dot(poly v) { // dot product with s
        T ans = 0; F0R(i,sz(v)) ans += v[i]*s[i];
    }
}

```

```

    return ans; } // get p-th term of rec
    T eval(ll p) { assert(p >= 0); return dot(getPow(p)); }
};

```

PolyInvSimpler.h

Description: computes A^{-1} such that $AA^{-1} \equiv 1 \pmod{x^n}$. Newton's method: If you want $F(x) = 0$ and $F(Q_k) \equiv 0 \pmod{x^a}$ then $Q_{k+1} = Q_k - \frac{F(Q_k)}{F'(Q_k)} \pmod{x^{2a}}$ satisfies $F(Q_{k+1}) \equiv 0 \pmod{x^{2a}}$. Application: if $f(n), g(n)$ are the #s of forests and trees on n nodes then $\sum_{n=0}^{\infty} f(n)x^n = \exp\left(\sum_{n=1}^{\infty} \frac{g(n)}{n!}\right)$.

Usage: vmi v{1,5,2,3,4}; ps(exp(2*log(v,9),9)); // squares v

Time: $\mathcal{O}(N \log N)$. For $N = 5 \cdot 10^5$, inv~270ms, log ~350ms, exp~550ms

"FFT.h", "Poly.h" 6e5362, 30 lines

```

poly inv(poly A, int n) { // Q-(1/Q-A)/(-Q^{f-2})
    poly B(inv(A[0]));
    for (int x = 2; x/2 < n; x *= 2)
        B = 2*B-RSZ(conv(RSZ(A,x),conv(B,B)),x);
    return RSZ(B,n);
}
poly sqrt(const poly& A, int n) { // Q-(Q^2-A)/(2Q)
    assert(A[0].v == 1); poly B{1};
    for (int x = 2; x/2 < n; x *= 2)
        B = inv(T(2))*RSZ(B+conv(RSZ(A,x),inv(B,x)),x);
    return RSZ(B,n);
}
// return {quotient, remainder}
pair<poly,poly> quoRem(const poly& f, const poly& g) {
    if (sz(f) < sz(g)) return {},f;
    poly q = conv(inv(rev(g),sz(f)-sz(g)+1),rev(f));
    q = rev(RSZ(q,sz(f)-sz(g)+1));
    poly r = RSZ(f-conv(q,g),sz(g)-1); return {q,r};
}
poly log(poly A, int n) { assert(A[0].v == 1); // (ln A)' = A'/A
    A.rsz(n); return integ(RSZ(conv(dif(A),inv(A,n-1)),n-1)); }
poly exp(poly A, int n) { assert(A[0].v == 0);
    poly B{1}, IB{1}; // inverse of B
    for (int x = 1; x < n; x *= 2) {
        IB = 2*IB-RSZ(conv(B,conv(IB,IB)),x);
        poly Q = dif(RSZ(A,x)); Q += RSZ(conv(IB,dif(B)-conv(B,Q)),2*x-1)
            <=;
        B = B+RSZ(conv(B,RSZ(A,2*x)-integ(Q)),2*x);
    }
    return RSZ(B,n);
}

```

PolyMultipoint.h

Description: Multipoint evaluation and interpolation

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

"PolyInv.h", "PolyConv.h" 9f6b18, 29 lines

```

void segProd(V<poly>& stor, poly& v, int ind, int l, int r) { // v ->
    <= places to evaluate at
    if (l == r) { stor[ind] = {-v[l],1}; return; }
    int m = (l+r)/2; segProd(stor,v,2*ind,l,m); segProd(stor,v,2*ind+1,
        <=m+1,r);
    stor[ind] = conv(stor[2*ind],stor[2*ind+1]);
}
void evalAll(V<poly>& stor, poly& res, poly v, int ind = 1) {
    v = quoRem(v,stor[ind]).s;
    if (sz(stor[ind]) == 2) { res.pb(sz(v)?v[0]:0); return; }
    evalAll(stor,res,v,2*ind); evalAll(stor,res,v,2*ind+1);
}

```

// evaluate polynomial v at points in p

```

poly multiEval(poly v, poly p) {
    V<poly> stor(4*sz(p)); segProd(stor,p,1,0,sz(p)-1);
    poly res; evalAll(stor,res,v); return res; }

```

```

poly combAll(V<poly>& stor, poly& dems, int ind, int l, int r) {
    if (l == r) return {dems[l]};
    int m = (l+r)/2;
    poly a = combAll(stor,dems,2*ind,l,m), b = combAll(stor,dems,2*ind
        <=+1,m+1,r);
    return conv(a,stor[2*ind+1])+conv(b,stor[2*ind]);
}

```

```

poly interpolate(V<pair<T,T>> v) {
    int n = sz(v); poly x; each(t,v) x.pb(t.f);
    V<poly> stor(4*n); segProd(stor,x,1,0,n-1);
}

```

```
poly dems; evalAll(stor,dems,dif(stor[1]));
F0R(i,n) dems[i] = v[i].s/dems[i];
return combAll(stor,dems,1,0,n-1);
}
```

4.3 Optimization

GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a,b]$ assuming f is unimodal on the interval, i.e. has only one local minimum. The maximum error in the result is ϵ ps. Works equally well for maximization with a small change in the code.

Usage: gss(-1000,1000,[](db x) { **return** 4+x+.3*x*x; }); // -5/3

Time: $\mathcal{O}(\log((b-a)/\epsilon))$ 544185, 14 lines

```
db gss(db a, db b, function<db>(db)> f) {
    db r = (sqrt(5)-1)/2, eps = 1e-7;
    db x1 = b - r*(b-a), x2 = a + r*(b-a);
    db f1 = f(x1), f2 = f(x2);
    while (b-a > eps)
        if (f1 < f2) { // change to > to find maximum
            b = x2; x2 = x1; f2 = f1;
            x1 = b - r*(b-a); f1 = f(x1);
        } else {
            a = x1; x1 = x2; f1 = f2;
            x2 = a + r*(b-a); f2 = f(x2);
        }
    return a;
}
```

HillClimbing.h

Description: Poor man's optimization for unimodal functions. 8eeaf, 14 lines

typedef array<double, 2> P;

```
template<class F> pair<double, P> hillClimb(P start, F f) {
    pair<double, P> cur(f(start), start);
    for (double jmp = 1e9; jmp > 1e-20; jmp /= 2) {
        rep(j,0,100) rep(dx,-1,2) rep(dy,-1,2) {
            P p = cur.second;
            p[0] += dx*jmp;
            p[1] += dy*jmp;
            cur = min(cur, make_pair(f(p), p));
        }
    }
    return cur;
}
```

Integrate.h

Description: Integration of a function over an interval using Simpson's rule, exact for polynomials of degree up to 3. The error should be proportional to dif^4 , although in practice you will want to verify that the result is stable to desired precision when epsilon changes.

Usage: quad([(db x) { **return** x*x+3*x+1; }, 2, 3] // 14.833 3ebaab, 6 lines

```
template<class F> db quad(F f, 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: Unused. Fast integration using adaptive Simpson's rule, exact for polynomials of degree up to 5.

Usage: db z, y;

```
db h(db x) { return x*x + y*y + z*z <= 1; }
db g(db y) { ::y = y; return quad(h, -1, 1); }
db f(db z) { ::z = z; return quad(g, -1, 1); }
db sphereVol = quad(f,-1,1), pi = sphereVol*3/4;
```

```
template<class F> db simpson(F f, db a, db b) {
    db c = (a+b)/2; return (f(a)+4*f(c)+f(b))*(b-a)/6; }
template<class F> db rec(F& f, db a, db b, db eps, db S) {
    db c = (a+b)/2;
    db S1 = simpson(f,a,c), 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);
}
template<class F> db quad(F f, db a, db b, db eps = 1e-8) {
    return rec(f,a,b,eps,simpson(f,a,b)); }
```

Simplex.h

Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \leq b$, $x \geq 0$. Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input vector x is set to an optimal x (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that $x = 0$ is viable.

Usage: vvd A{{1,-1}, {-1,1}, {-1,-2}};

vd b{1,1,-4}, c{-1,-1}, x;

T val = LPSolver(A, b, c).solve(x);

Time: $\mathcal{O}(NM \cdot \# \text{pivots})$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}(2^N)$ in the general case. c99f9c, 67 lines

using T = db; // double probably suffices

using vd = V<T>; **using** vvd = V<vd>;

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; // # m = constraints, # n = variables
    vi N, B; // N[j] = non-basic variable (j-th column), = 0
    vvd D; // B[j] = basic variable (j-th row)
    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)) {
        F0R(i,m) F0R(j,n) D[i][j] = A[i][j];
        F0R(i,m) B[i] = n+i, D[i][n] = -1, D[i][n+1] = b[i];
        // B[i]: basic variable for each constraint
        // D[i][n]: artificial variable for testing feasibility
        F0R(j,n) N[j] = j, D[m][j] = -c[j];
        // D[m] stores negation of objective,
        // which we want to minimize
        N[n] = -1; D[m+1][n] = 1; // to find initial feasible
    } // solution, minimize artificial variable
    void pivot(int r, int s) { // swap B[r] (row)
        T inv = 1/D[r][s]; // with N[r] (column)
        F0R(i,m+2) if (i != r && abs(D[i][s]) > eps) {
            T binv = D[i][s]*inv;
            F0R(j,n+2) if (j != s) D[i][j] -= D[r][j]*binv;
            D[i][s] = -binv;
        }
        D[r][s] = 1; F0R(j,n+2) D[r][j] *= inv; // scale r-th row
        swap(B[r],N[s]);
    }
}
```

```
bool simplex(int phase) {
    int x = m+phase-1;
    while (1) { // if phase=1, ignore artificial variable
        int s = -1; F0R(j,n+1) if (N[j] != -phase) ltj(D[x]);
        // find most negative col for nonbasic (NB) variable
        if (D[x][s] >= -eps) return 1;
        // can't get better sol by increasing NB variable
        int r = -1;
        F0R(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
        } // -> max increase in NB variable
        if (r == -1) return 0; // objective is unbounded
        pivot(r,s);
    }
}
```

```
T solve(vd& x) { // 1. check if x=0 feasible
    int r = 0; FOR(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) { // if not, find feasible start
        pivot(r,n); // make artificial variable basic
        assert(simplex(2)); // I think this will always be true??
        if (D[m+1][n+1] < -eps) return -inf;
        // D[m+1][n+1] is max possible value of the negation of
        // artificial variable, optimal value should be zero
        // if exists feasible solution
        F0R(i,m) if (B[i] == -1) { // artificial var basic
            int s = 0; FOR(j,1,n+1) ltj(D[i]); // -> nonbasic
            pivot(i,s);
        }
    }
    bool ok = simplex(1); x = vd(n);
    F0R(i,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
}
};
```

4.4 Fourier transforms

FastFourierTransform.h

Description: $\text{fft}(a)$ computes $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$ for all k . N must be a power of 2. Useful for convolution: $\text{conv}(a, b) = c$, where $c[x] = \sum a[i]b[x-i]$. For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n , reverse(start+1, end), FFT back. Rounding is safe if $(\sum a_i^2 + \sum b_i^2) \log_2 N < 9 \cdot 10^{14}$ (in practice 10^{16} ; higher for random inputs). Otherwise, use NTT/FFTMod.

Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ (~ 1 s for $N = 2^{22}$) 00ced6, 35 lines

```
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1); // (^ 10% faster if double)
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n); rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
    }
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
            C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
}
vd conv(const vd& a, const vd& b) {
    if (a.empty() || b.empty()) return {};
    vd res(sz(a) + sz(b) - 1);
    int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
    vector<C> in(n), out(n);
    copy(all(a), begin(in));
    rep(i,0,sz(b)) in[i].imag(b[i]);
    fft(in);
    for (C& x : in) x *= x;
    rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out);
    rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
    return res;
}
```

FastFourierTransformMod.h

Description: Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$ (in practice 10^{16} or higher). Inputs must be in $[0, \text{mod})$.

Time: $\mathcal{O}(N \log N)$, where $N = |A| + |B|$ (twice as slow as NTT or FFT)

"FastFourierTransform.h" b82773, 22 lines

```
typedef vector<l> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    vl res(sz(a) + sz(b) - 1);
    int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
    vector<C> L(n), R(n), outs(n), outl(n);
    rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
    rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
    fft(L), fft(R);
    rep(i,0,n) {
        int j = -i & (n - 1);
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
    }
    fft(outl), fft(outs);
    rep(i,0,sz(res)) {
        ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
        ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
        res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
    }
    return res;
}
```

FFT.h

Description: Multiply polynomials of ints for any modulus $< 2^{31}$. For XOR convolution ignore m within fft.

Time: $\mathcal{O}(N \log N)$. For $N = 10^6$, conv ~ 0.13 ms, conv-general ~ 320 ms.

"ModInt.h" 19ab20, 39 lines

```
// const int MOD = 998244353;
tcT> void fft(V<T>& A, bool invert = 0) { // NTT
    int n = sz(A); assert((T::mod-1)&n == 0); V<T> B(n);
    for(int b = n/2; b; b /= 2, swap(A,B)) { // w = n/b'th root
        T w = pow(T::rt(), (T::mod-1)/n*b), m = 1;
        for(int i = 0; i < n; i += b*2, m *= w) F0R(j,b) {
            T u = A[i+j], v = A[i+j+b]*m;
            B[i/2+j] = u+v; B[i/2+j+n/2] = u-v;
        }
    }
    if (invert) { reverse(1+all(A));
        T z = inv(T(n)); each(t,A) t *= z; }
} // for NTT-able moduli
tcT> V<T> conv(V<T> A, V<T> B) {
    if (!min(sz(A),sz(B))) return {};
    int s = sz(A)+sz(B)-1, n = 1; for (; n < s; n *= 2);
    A.rsz(n), fft(A); B.rsz(n), fft(B);
    F0R(i,n) A[i] *= B[i];
    fft(A,1); A.rsz(s); return A;
}
template<class M, class T> V<M> mulMod(const V<T>& x, const V<T>& y)
    ↪{
    auto con = [](const V<T>& v) {
        V<M> w(sz(v)); F0R(i,sz(v)) w[i] = (int)v[i];
        return w; };
    return conv(con(x), con(y));
} // arbitrary moduli
tcT> V<T> conv_general(const V<T>& A, const V<T>& B) {
    using m0 = mint<((119<<23)+1,62>); auto c0 = mulMod<m0>(A,B);
    using m1 = mint<(5<<25)+1, 62>; auto c1 = mulMod<m1>(A,B);
    using m2 = mint<(7<<26)+1, 62>; auto c2 = mulMod<m2>(A,B);
    int n = sz(c0); V<T> res(n); m1 r01 = inv(m1(m0::mod));
    m2 r02 = inv(m2(m0::mod)); r12 = inv(m2(m1::mod));
    F0R(i,n) { // a=remainder mod m0::mod, b fixes it mod m1::mod
        int a = c0[i].v, b = ((c1[i]-a)*r02-b)*r12.v,
            c = (((c2[i]-a)*r02-b)*r12).v;
        res[i] = (T(c)*m1::mod+b)*m0::mod+a; // c fixes m2::mod
    }
    return res;
}
```

FastSubsetTransform.h

Description: Transform to a basis with fast convolutions of the form $c[z] = \sum_{z=x\oplus y} a[x] \cdot b[y]$, where \oplus is one of AND, OR, XOR. The size of a must be a power of two.
Time: $\mathcal{O}(N \log N)$

```
void FST(vi& a, bool inv) {
    for (int n = sz(a), step = 1; step < n; step *= 2) {
        for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step) {
            int &u = a[j], &v = a[j + step]; tie(u, v) =
                inv ? pii(v - u, u) : pii(v, u + v); // AND
                inv ? pii(v, u - v) : pii(u + v, u); // OR
                pii(u + v, u - v); // XOR
        }
    }
    if (inv) for (int& x : a) x /= sz(a); // XOR only
}
vi conv(vi a, vi b) {
    FST(a, 0); FST(b, 0);
    rep(i,0,sz(a)) a[i] *= b[i];
    FST(a, 1); return a;
}
```

Number Theory (5)

5.1 Modular Arithmetic

Modular Arithmetic/ModIntShort.h

Description: Modular arithmetic. Assumes MOD is prime.

Usage: `mi a = MOD+5; inv(a);` // 400000003

```
template<int MOD, int RT> struct mint {
    static const int mod = MOD;
    static constexpr mint rt() { return RT; } // primitive root
    int v;
    explicit operator int() const { return v; }
    mint():v(0) {}
    mint(ll _v):v(int(_v%MOD)) { v += (v<0)*MOD; }
```

```
    mint& operator+=(mint o) {
        if ((v += o.v) >= MOD) v -= MOD;
        return *this; }
    mint& operator-=(mint o) {
        if ((v -= o.v) < 0) v += MOD;
        return *this; }
    mint& operator*=(mint o) {
        v = int((ll)v*o.v%MOD); return *this; }
    friend mint pow(mint a, ll p) { assert(p >= 0);
        return p==0?1:pow(a*a,p/2)*(p&1?a:1); }
    friend mint inv(mint a) { assert(a.v != 0); return pow(a,MOD-2); }
    friend mint operator+(mint a, mint b) { return a += b; }
    friend mint operator-(mint a, mint b) { return a -= b; }
    friend mint operator*(mint a, mint b) { return a *= b; }
};
using mi = mint<(int)1e9+7, 5>;
using vmi = V<mi>;
```

Modular Arithmetic/ModFact.h

Description: Combinations modulo a prime MOD . Assumes $2 \leq N \leq MOD$.

Usage: `F.init(10); F.C(6, 4);` // 15

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

```
"ModInt.h" 364271, 13 lines
struct {
    vmi invs, fac, ifac;
    void init(int N) { // idempotent
        invs.rsz(N), fac.rsz(N), ifac.rsz(N);
        invs[1] = fac[0] = ifac[0] = 1;
        FOR(i,2,N) invs[i] = mi((-1ll)MOD/i*(int)invs[MOD%i]);
        FOR(i,1,N) fac[i] = fac[i-1]*i, ifac[i] = ifac[i-1]*invs[i];
    }
    mi C(int a, int b) {
        if (a < b || b < 0) return 0;
        return fac[a]*ifac[b]*ifac[a-b];
    }
} F;
```

Modular Arithmetic/ModMulLL.h

Description: Primitive two 64-bit integers mod another if 128-bit is not available. `modMul` is equivalent to `(ul)((__int128(a)*b%mod)`. Works for $0 \leq a, b < mod < 2^{93}$.

```
530181, 9 lines
using ul = uint64_t;
ul modMul(ul a, ul b, const ul mod) {
    ll ret = a*b-mod*(ul)((db)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);
    return b&1 ? modMul(res,a,mod) : res;
}
```

Modular Arithmetic/FastMod.h

Description: Barrett reduction computes $a\%b$ about 4 times faster than usual where $b > 1$ is constant but not known at compile time. Division by b is replaced by multiplication by m and shifting right 64 bits.

```
aa19c9, 7 lines
using ul = uint64_t; using L = __uint128_t;
struct FastMod {
    ul b, m; FastMod(ul b) : b(b), m(-1ULL / b) {}
    ul reduce(ul a) {
        ul q = (ul)((__uint128_t(m) * a) >> 64), r = a - q * b;
        return r - (r >= b) * b; }
};
```

Modular Arithmetic/ModSqrt.h

Description: Tonelli-Shanks algorithm for square roots mod a prime. -1 if doesn't exist.

Usage: `sqr(mi((-1ll)1e10));` // 100000

Time: $\mathcal{O}\left(\log^2(MOD)\right)$

```
"ModInt.h" bcfa63, 14 lines
using T = int;
T sqrt(mi a) {
    mi p = pow(a, (MOD-1)/2);
    if (p.v != 1) return p.v == 0 ? 0 : -1;
    T s = MOD-1; int r = 0; while (s%2 == 0) s /= 2, ++r;
    mi n = 2; while (pow(n, (MOD-1)/2).v == 1) n = T(n)+1;
    // n non-square, ord(g)=2^r, ord(b)=2^m, ord(g)=2^r, m<r
```

```
for (mi x = pow(a,(s+1)/2), b = pow(a,s), g = pow(n,s);) {
    if (b.v == 1) return min(x.v,MOD-x.v); // x^2=ab
    int m = 0; for (mi t = b; t.v != 1; t *= t) ++m;
    rep(r-m-1) g *= g; // ord(g)=2^(m+1)
    x *= g, g *= g, b *= g, r = m; // ord(g)=2^m, ord(b)<2^m
}
}
```

Modular Arithmetic/ModSum.h

Description: Counts # of lattice points (x,y) in the triangle $1 \leq x, 1 \leq y, ax + by \leq s \pmod{2^{64}}$ and related quantities.

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

```
23cbf6, 20 lines
using ul = uint64_t;
ul sum2(ul n) { return n/2*((n-1)|1); } // sum(0..n-1)
// \return {f(x,y) | 1 <= x, 1 <= y, a*x+b*y <= S}
//      = sum_{i=1}^{qs} (S-a*i)/b
ul triSum(ul a, ul b, ul s) { assert(a > 0 && b > 0);
    ul qs = s/a, rs = s/a; // ans = sum_{i=0}^{qs-1}(i*a+rs)/b
    ul ad = a/b*sum2(qs)+rs*b*qs; a %= b, rs %= b;
    return ad+(a?triSum(b,a,a*qs+rs):0); // reduce if a >= b
} // then swap x and y axes and recurse
```

```
// \return sum_{x=0}^{n-1} (a*x+b)/m
//      = {f(x,y) | 0 < m*y <= a*x+b < a*n+b}
// assuming a*n+b does not overflow
ul divSum(ul n, ul a, ul b, ul m) { assert(m > 0);
    ul extra = b/m*n; b %= m;
    return extra+(a?triSum(m,a,a*n+b):0); }
// \return sum_{x=0}^{n-1} (a*x+b)%m
ul modSum(ul n, ll a, ll b, ul m) { assert(m > 0);
    a = (a%m+m)%m, b = (b%m+m)%m;
    return a*sum2(n)+b*n-m*divSum(n,a,b,m); }
```

5.2 Primality

5.2.1 Primes

$p = 962592769$ is such that $2^{21} \mid p - 1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power p^a , except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$ the group $\mathbb{Z}_{2^a}^\times$ is instead isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

5.2.2 Divisors

$\sum d \mid n \ d = \mathcal{O}(n \log \log n)$.

The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

Dirichlet Convolution: Given a function $f(x)$, let

$(f * g)(x) = \sum d \mid x \ g(d)f(x/d)$.

If the partial sums $s_{f*g}(n), s_g(n)$ can be computed in $\mathcal{O}(1)$ and $s_f(1 \dots n^{2/3})$ can be computed in $\mathcal{O}(n^{2/3})$ then all $s_f(\frac{n}{d})$ can as well. Use

$$s_{f*g}(n) = \sum_{d=1}^n g(d)s_f(n/d).$$

If $f(x) = \mu(x)$ then $g(x) = 1, (f * g)(x) = (x == 1)$, and $s_f(n) = 1 - \sum_{i=2}^n s_f(n/i)$.

If $f(x) = \phi(x)$ then $g(x) = 1, (f * g)(x) = x$, and $s_f(n) = \frac{n(n+1)}{2} - \sum_{i=2}^n s_f(n/i)$.

Primality/Sieve.h

Description: Tests primality up to SZ . Runs faster if only odd indices are stored.

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

41c6ed, 20 lines

```
template<int SZ> struct Sieve {
    bitset<SZ> is_prime; vi primes;
    Sieve() {
        is_prime.set(); is_prime[0] = is_prime[1] = 0;
        for (int i = 4; i < SZ; i += 2) is_prime[i] = 0;
        for (int i = 3; i*i < SZ; i += 2) if (is_prime[i])
            for (int j = i*i; j < SZ; j += i*2) is_prime[j] = 0;
        F0R(i,SZ) if (is_prime[i]) primes.pb(i);
    }
    // int sp[SZ]{}; // smallest prime that divides
    // Sieve() { // above is faster
    //     FOR(i,2,SZ) {
    //         if (sp[i] == 0) sp[i] = i, primes.pb(i);
    //         for (int p: primes) {
    //             if (p > sp[i] || i*p >= SZ) break;
    //             sp[i*p] = p;
    //         }
    //     }
    // }
```

Primality/MultiplicativePrefixSums.h

Description: $\sum_{i=1}^N f(i)$ where $f(i) = \prod \text{val}[e]$ for each p^e in the factorization of i . Must satisfy $\text{val}[1] = 1$. Generalizes to any multiplicative function with $f(p) = p^{\text{fixed power}}$.

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

"Sieve.h" 3151ea, 12 lines

```
vmi val;
mi get_prefix(ll N, int p = 0) {
    mi ans = N;
    for (; S.primes.at(p) <= N / S.primes.at(p); ++p) {
        ll new_N = N / S.primes.at(p) / S.primes.at(p);
        for (int idx = 2; new_N; ++idx, new_N /= S.primes.at(p)) {
            ans += (val.at(idx) - val.at(idx - 1))
                * get_prefix(new_N, p + 1);
        }
    }
    return ans;
}
```

Primality/PrimeCnt.h

Description: Counts number of primes up to N . Can also count sum of primes.

Time: $\mathcal{O}(N^{3/4} / \log N)$, 60ms for $N = 10^{11}$, 2.5s for $N = 10^{13}$

c04e96, 20 lines

```
ll count_primes(ll N) { // count_primes(1e13) == 346065536839
    if (N <= 1) return 0;
    int sq = (int)sqrt(N);
    vl big_ans((sq+1)/2), small_ans(sq+1);
    FOR(i,1,sq+1) small_ans[i] = (i-1)/2;
    F0R(i,sz(big_ans)) big_ans[i] = (N/(2*i+1)-1)/2;
    vb skip(sq+1); int prime_cnt = 0;
    for (int p = 3; p <= sq; p += 2) if (!skip[p]) { // primes
        for (int j = p; j <= sq; j += 2*p) skip[j] = 1;
        F0R(j,min((ll)sz(big_ans), (N/p/p+1)/2)) {
            ll prod = (ll)(2*j+1)*p;
            big_ans[j] -= (prod > sq ? small_ans[(double)N/prod]
                : big_ans[prod/2]) - prime_cnt;
        }
        for (int j = sq, q = sq/p; q >= p; --q) for (; j >= q*p; --j)
            small_ans[j] -= small_ans[q] - prime_cnt;
        ++prime_cnt;
    }
    return big_ans[0]+1;
}
```

Primality/MillerRabin.h

Description: Deterministic primality test, works up to 2^{64} . For larger numbers, extend A randomly.

"ModMulLL.h" 89df33, 11 lines

```
bool prime(ul n) { // not ll!
    if (n < 2 || n % 6 % 4 != 1) return n-2 < 2;
    ul A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
```

```
        s = __builtin_ctzll(n-1), d = n>s;
        each(a,A) { // ^ count trailing zeroes
            ul p = modPow(a,d,n), i = s;
            while (p != 1 && p != n-1 && a%n && i--) p = modMul(p,p,n);
            if (p != n-1 && i != s) return 0;
        }
        return 1;
    }
}
```

Primality/FactorFast.h

Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).

Time: $\mathcal{O}(N^{1/4})$, less for numbers with small factors

"MillerRabin.h", "../Modular Arithmetic/ModMulLL.h" 99cf33, 16 lines

```
ul pollard(ul n) { // return some nontrivial factor of n
    auto f = [n](ul x) { return modMul(x, x, n) + 1; };
    ul x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    while (t++ % 40 || gcd(prd, n) == 1) {
        if (x == y) x = ++i, y = f(x);
        if ((q = modMul(prd, max(x,y)-min(x,y), n)) prd = q;
            x = f(x), y = f(f(y));
        }
        return gcd(prd, n);
    }
}

void factor_rec(ul n, map<ul,int>& cnt) {
    if (n == 1) return;
    if (prime(n)) { ++cnt[n]; return; }
    ul u = pollard(n);
    factor_rec(u,cnt), factor_rec(n/u,cnt);
}
```

5.3 Euclidean Algorithm

Euclid/FracInterval.h

Description: Given fractions $a < b$ with non-negative numerators and denominators, finds fraction f with lowest denominator such that $a < f < b$. Should work with all numbers less than 2^{62} .

1860f3, 6 lines

```
pl bet(pl a, pl b) {
    ll num = a.f/a.s; a.f -= num*a.s, b.f -= num*b.s;
    if (b.f > b.s) return {1+num,1};
    auto x = bet({b.s,b.f},{a.s,a.f});
    return {x.s+num*x.f,x.f};
}
```

Euclid/Euclid.h

Description: Generalized Euclidean algorithm. euclid and invGeneral work for $A, B < 2^{62}$.

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

c7e528, 9 lines

```
// ceil(a/b)
// ll cdiv(ll a, ll b) { return a/b+((a^b)>0&&a%b); }
pl euclid(ll A, ll B) { // For A,B>0, finds (x,y) s.t.
    // Ax+By=gcd(A,B), |Ax|,|By|<=AB/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) { // find x in {0,B} such that Ax=1 mod B
    pl p = euclid(A,B); assert((p.f*A+p.s*B == 1);
    return p.f+(p.f<0)*B; } // must have gcd(A,B)=1
```

Euclid/CRT.h

Description: Chinese Remainder Theorem. $a.f \pmod{a.s}, b.f \pmod{b.s} \implies ? \pmod{\text{lcm}(a.s,b.s)}$. Should work for $ab < 2^{62}$.

"Euclid.h" 23df64, 10 lines

```
pl CRT(pl a, pl b) { assert(0 <= a.f && a.f < a.s && 0 <= b.f && b.f
    < b.s);
    if (a.s < b.s) swap(a,b); // will overflow if b.s^2 > 2^62
    ll x,y; tie(x,y) = euclid(a.s,b.s);
    ll g = a.s*x+b.s*y, l = a.s/g*b.s;
    if ((b.f-a.f)%g) return {-1,-1}; // no solution
    // ?*a.s+a.f \equiv b.f \pmod{b.s}
    // ?=(b.f-a.f)/g*(a.s/g)^{-1} \pmod{b.s/g}
    x = (b.f-a.f)%b.s*x%b.s/g*a.s+a.f;
    return {x+(x<0)*l,1};
}
```

Euclid/ModArith.h

Description: Statistics on mod'ed arithmetic series. minBetween and minRemainder both assume that $0 \leq L \leq R < B$, $AB < 2^{62}$

f68a6d, 40 lines

```
ll minBetween(ll A, ll B, ll L, ll R) {
    // min x s.t. exists y s.t. L <= A*x-B*y <= R
    A %= B;
    if (L == 0) return 0;
    if (A == 0) return -1;
    ll k = cdiv(L,A); if (A*k <= R) return k;
    ll x = minBetween(B,A,A-R%A,A-L%A); // min x s.t. exists y
    // s.t. -R <= Bx-Ay <= -L
    return x == -1 ? x : cdiv(B*x+L,A); // solve for y
}
```

```
// find min((Ax+C)%B) for 0 <= x <= M
// aka find minimum non-negative value of A*x-B*y+C
// where 0 <= x <= M, 0 <= y
ll minRemainder(ll A, ll B, ll C, ll M) {
    assert(A >= 0 && B > 0 && C >= 0 && M >= 0);
    A %= B, C %= B; ckmin(M,B-1);
    if (A == 0) return C;
    if (C >= A) { // make sure C<A
        ll ad = cdiv(B-C,A);
        M -= ad; if (M < 0) return C;
        C += ad*A-B;
    }
    ll q = B/A, new_B = B%A; // new_B < A
    if (new_B == 0) return C; // B-q*A
```

```
// now minimize A*x-new_B*y+C
// where 0 <= x,y and x+q*y <= M, 0 <= C < new_B < A
// q*y -> C-new_B*y
if (C/new_B > M/q) return C-M/q*new_B;
M -= C/new_B*q; C %= new_B; // now C < new_B
```

```
// given y, we can compute x = ceil((B-q*A)*y-C)/A
// so x+q*y = ceil((B*y-C)/A) <= M
ll max_Y = (M*A+C)/B; // must have y <= max_Y
ll max_X = cdiv(new_B*max_Y-C,A); // must have x <= max_X
if (max_X*A-new_B*max_Y+C >= new_B) --max_X;
// now we can remove upper bound on y
return minRemainder(A,new_B,C,max_X);
}
```

5.4 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2), \quad b = k \cdot (2mn), \quad c = k \cdot (m^2 + n^2),$$

with $m > n > 0$, $k > 0$, $m \perp n$, and either m or n even.

5.5 Lifting the Exponent

For $n > 0$, p prime, and ints x, y s.t. $p \nmid x, y$ and $p \mid x - y$:

$$p \neq 2 \text{ or } p = 2, 4 \mid x - y \implies v_p(x^n - y^n) = v_p(x - y) + v_p(n).$$

$$p = 2, 2 \nmid n \implies v_2(x^n - y^n) = v_2((x^2)^{n/2} - (y^2)^{n/2}).$$

Combinatorial (6)

6.1 Permutations

6.1.1 Cycles

Let $g_S(n)$ be the number of n -permutations whose cycle lengths all belong to the set S . Then

$$\sum_{n=0}^{\infty} g_S(n) \frac{x^n}{n!} = \exp \left(\sum_{n \in S} \frac{x^n}{n} \right)$$

6.1.2 Burnside’s lemma

Given a group G of symmetries and a set X , the number of elements of X *up to symmetry* equals

$$\frac{1}{|G|} \sum_{g \in G} |X^g|,$$

where X^g are the elements fixed by g ($g.x = x$).

If $f(n)$ counts “configurations” (of some sort) of length n , we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get

$$g(n) = \frac{1}{n} \sum_{k=0}^{n-1} f(\gcd(n, k)) = \frac{1}{n} \sum_{k|n} f(k) \phi(n/k).$$

6.2 Partitions and subsets

6.2.1 Partition function

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1, \quad p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k - 1)/2)$$

$$p(n) \sim 0.145 \cdot n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	$\sim 2e5$	$\sim 2e8$

6.2.2 Lucas’ Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + \dots + n_1 p + n_0$ and $m = m_k p^k + \dots + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^k \binom{n_i}{m_i} \pmod{p}$.

6.3 General purpose numbers

6.3.1 Bernoulli numbers

EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t - 1}$ (FFT-able).

$$B[0, \dots] = [1, -\tfrac{1}{2}, \tfrac{1}{6}, 0, -\tfrac{1}{30}, 0, \tfrac{1}{42}, \dots]$$

Sums of powers:

$$\sum_{i=1}^n i^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\begin{aligned} \sum_{i=m}^{\infty} f(i) &= \int_m^{\infty} f(x) dx - \sum_{k=1}^{\infty} \frac{B_k}{k!} f^{(k-1)}(m) \\ &\approx \int_m^{\infty} f(x) dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m)) \end{aligned}$$

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n, k) = c(n - 1, k - 1) + (n - 1)c(n - 1, k), \quad c(0, 0) = 1$$

$$\sum_{k=0}^n c(n, k) x^k = x(x + 1) \dots (x + n - 1)$$

$$\begin{aligned} c(8, k) &= 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1 \\ c(n, 2) &= 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \dots \end{aligned}$$

6.3.3 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j + 1)$, $k + 1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

$$E(n, k) = (n - k)E(n - 1, k - 1) + (k + 1)E(n - 1, k)$$

$$E(n, 0) = E(n, n - 1) = 1$$

$$E(n, k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$$

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

$$S(n, k) = S(n - 1, k - 1) + kS(n - 1, k)$$

$$S(n, 1) = S(n, n) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$$

6.3.5 Bell numbers

Total number of partitions of n distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \dots$ For p prime,

$$B(p^m + n) \equiv mB(n) + B(n + 1) \pmod{p}$$

6.3.6 Labeled unrooted trees

on n vertices: n^{n-2}

on k existing trees of size n_i : $n_1 n_2 \dots n_k n^{k-2}$

with degrees d_i : $(n - 2)! / ((d_1 - 1)! \dots (d_n - 1)!)$

6.3.7 Catalan numbers

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)n!}$$

$$C_0 = 1, \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n, \quad C_{n+1} = \sum C_i C_{n-i}$$

$$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \dots$$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n + 1$ leaves (0 or 2 children).
- ordered trees with $n + 1$ vertices.
- ways a convex polygon with $n + 2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

6.4 Young Tableaux

Let a **Young diagram** have shape $\lambda = (\lambda_1 \geq \dots \geq \lambda_k)$, where λ_i equals the number of cells in the i -th (left-justified) row from the top. A **Young tableau** of shape λ is a filling of the $n = \sum \lambda_i$ cells with a permutation of $1 \dots n$ such that each row and column is increasing.

Hook-Length Formula: For the cell in position (i, j) , let $h_{\lambda}(i, j) = |\{(I, J) | i \leq I, j \leq J, (I = i \text{ or } J = j)\}|$. The number of Young tableaux of shape λ is equal to $f^{\lambda} = \frac{n!}{\prod h_{\lambda}(i, j)}$.

Schensted’s Algorithm: converts a permutation σ of length n into a pair of Young Tableaux $(S(\sigma), T(\sigma))$ of the same shape. When inserting $x = \sigma_i$,

1. Add x to the first row of S by inserting x in place of the largest y with $x < y$. If y doesn’t exist, push x to the end of the row, set the value of T at that position to be i , and stop.
2. Add y to the second row using the same rule, keep repeating as necessary.

All pairs $(S(\sigma), T(\sigma))$ of the same shape correspond to a unique σ , so $n! = \sum (f^{\lambda})^2$. Also, $S(\sigma^R) = S(\sigma)^T$.

Let $d_k(\sigma), a_k(\sigma)$ be the lengths of the longest subseqs which are a union of k decreasing/ascending subseqs, respectively. Then $a_k(\sigma) = \sum_{i=1}^k \lambda_i, d_k(\sigma) = \sum_{i=1}^k \lambda_i^*$, where λ_i^* is size of the i -th column.

6.5 Other

DeBruijnSeq.h

Description: Given alphabet $[0, k)$ constructs a cyclic string of length k^n that contains every length n string as substr. a6961b, 13 lines

```
vi deBruijnSeq(int k, int n) {
    if (k == 1) return {0};
    vi seq, aux(n+1);
    function<void(int,int)> gen = [&](int t, int p) {
        if (t > n) { // +lyndon word of len p
            if (n%p == 0) FOR(i,1,p+1) seq.pb(aux[i]);
        } else {
            aux[t] = aux[t-p]; gen(t+1,p);
            while (++aux[t] < k) gen(t+1,t);
        }
    };
    gen(1,1); return seq;
}
```

NimProduct.h

Description: Product of numbers is associative, commutative, and distributive over addition (xor). Forms finite field of size 2^{2^k} . Defined by $ab = \text{mex}(\{a'b' + ab' + a'b' : a' < a, b' < b\})$. Application: Given 1D coin turning games G_1, G_2 $G_1 \times G_2$ is the 2D coin turning game defined as follows. If turning coins at x_1, x_2, \dots, x_m is legal in G_1 and y_1, y_2, \dots, y_n is legal in G_2 , then turning coins at all positions (x_i, y_j) is legal assuming that the coin at (x_m, y_n) goes from heads to tails. Then the Grundy function $g(x, y)$ of $G_1 \times G_2$ is $g_1(x) \times g_2(y)$. **Time:** 64^2 xors per multiplication, memorize to speed up. 5afe17, 46 lines

```
using ul = uint64_t;
struct Precalc {
    ul tmp[64][64], y[8][8][256];
    unsigned char x[256][256];
    Precalc() { // small nim products, all < 256
        FOR(i,256) FOR(j,256) x[i][j] = mult<8>(i,j);
        FOR(i,8) FOR(j,i+1) FOR(k,256)
            y[i][j][k] = mult<64>(prod2(8*i,8*j),k);
    }
    ul prod2(int i, int j) { // nim prod of 2^i, 2^j
        ul& u = tmp[i][j]; if (u) return u;
        if (!(i&j)) return u = 1ULL<<(i|j);
        int a = (i&j)&~(i&j); // a=2^k, consider 2^{2^k}
        return u=prod2(i^a,j)^prod2((i^a)|(a-1),(j^a)|(i&(a-1)));
        // 2^{2^k}*2^{2^k} = 2^{2^k}+2^{2^k-1}
    } // 2^{2^i}*2^{2^j} = 2^{2^i+2^j} if i<j
    template<int L> ul mult(ul a, ul b) {
        ul c = 0; FOR(i,L) if (a>>i&1)
            FOR(j,L) if (b>>j&1) c ^= prod2(i,j);
        return c;
    }
    // 2^{8*i}*(a>>(8*i)&255) * 2^{8*j}*(b>>(8*j)&255)
    // -> (2^{8*i}*2^{8*j})*(a>>(8*i)&255)*(b>>(8*j)&255))
    ul multFast(ul a, ul b) const { // faster nim product
        ul res = 0; auto f=[](ul c,int d) {return c>>(8*d)&255;};
        FOR(i,8) {
            FOR(j,i) res ^= y[i][j][x[f(a,i)][f(b,j)]]
                ^x[f(a,j)][f(b,i)]];
            res ^= y[i][i][x[f(a,i)][f(b,i)]]];
        }
        return res;
    }
}
```

```
const Precalc P;

struct nb { // nimber
    ul x; nb() { x = 0; }
    nb(ul _x): x(_x) {}
    explicit operator ul() { return x; }
    nb operator+(nb y) { return nb(x^y.x); }
    nb operator*(nb y) { return nb(P.multFast(x,y.x)); }
    friend nb pow(nb b, ul p) {
        nb res = 1; for (;p;p/=2,b=b*b) if (p&1) res = res*b;
        return res; } // b^{2^A-1}=1 where 2^A > b
    friend nb inv(nb b) { return pow(b,-2); }
};
```

MatroidIsect.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. In general, construct the exchange graph and find a shortest path. Can apply similar concept to partition matroid.

Usage: MatroidIsect<Gmat, Cmat> M(sz(ed), Gmat(ed), Cmat(col))

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

```
"../graphs (12)/DSU/DSU (7.6).h" d0051c, 51 lines

struct Gmat { // graphic matroid
    int V = 0; vpi ed; DSU D;
    Gmat(vpi _ed):ed(_ed) {
        map<int,int> m; each(t,ed) m[t.f] = m[t.s] = 0;
        each(t,m) t.s = V++;
        each(t,ed) t.f = m[t.f], t.s = m[t.s];
    }
    void clear() { D.init(V); }
    void ins(int i) { assert(D.unite(ed[i].f,ed[i].s)); }
    bool indep(int i) { return !D.sameSet(ed[i].f,ed[i].s); }
};

struct Cmat { // colorful matroid
    int C = 0; vi col; V<bool> used;
    Cmat(vi col):col(col) {each(t,col) ckmax(C,t+1); }
    void clear() { used.assign(C,0); }
    void ins(int i) { used[col[i]] = 1; }
    bool indep(int i) { return !used[col[i]]; }
};

template<class M1, class M2> struct MatroidIsect {
    int n; V<bool> iset; M1 m1; M2 m2;
    bool augment() {
        vi pre(n+1,-1); queue<int> q({n});
        while (sz(q)) {
            int x = q.ft; q.pop();
            if (iset[x]) {
                m1.clear(); F0R(i,n) if (iset[i] && i != x) m1.ins(i);
                F0R(i,n) if (!iset[i] && pre[i] == -1 && m1.indep(i))
                    pre[i] = x, q.push(i);
            } else {
                auto backE = [&]() { // back edge
                    m2.clear();
                    F0R(c,2)F0R(i,n)if((x==i|!iset[i])&&(pre[i]==-1)==c){
                        if (!m2.indep(i))return c?pre[i]=x,q.push(i),i:-1;
                        m2.ins(i); }
                    return n;
                };
                for (int y; (y = backE()) != -1;) if (y == n) {
                    for(; x != n; x = pre[x]) iset[x] = !iset[x];
                    return 1; }
            }
        }
        return 0;
    }
    MatroidIsect(int n, M1 m1, M2 m2):n(n), m1(m1), m2(m2) {
        iset.assign(n+1,0); iset[n] = 1;
        m1.clear(); m2.clear(); // greedily add to basis
        R0F(i,n) if (m1.indep(i) && m2.indep(i))
            iset[i] = 1, m1.ins(i), m2.ins(i);
        while (augment());
    }
};
```

Erdos-Gallai: $d_1 \geq \cdots \geq d_n$ can be degree sequence of simple graph on n vertices iff their sum is even and $\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k), \forall 1 \leq k \leq n$.

7.1 Basics

DSU/DSUrb (15.5).h

Description: Disjoint Set Union with Rollback	7d0297, 18 lines
------------------------------------------------------	------------------

```
struct DSUrb {
    vi e; void init(int n) { e = vi(n,-1); }
    int get(int x) { return e[x] < 0 ? x : get(e[x]); }
    bool sameSet(int a, int b) { return get(a) == get(b); }
    int size(int x) { return -e[get(x)]; }
    V<AR<int,4>> mod;
    bool unite(int x, int y) { // union-by-rank
        x = get(x), y = get(y);
        if (x == y) { mod.pb({-1,-1,-1,-1}); return 0; }
        if (e[x] > e[y]) swap(x,y);
        mod.pb({x,y,e[x],e[y]});
        e[x] += e[y]; e[y] = x; return 1;
    }
    void rollback() {
        auto a = mod.bk; mod.pop_back();
        if (a[0] != -1) e[a[0]] = a[2], e[a[1]] = a[3];
    }
};
```

Basics/NegativeCycle (7.3).h

Description: use Bellman-Ford (make sure no underflow)	688ec8, 11 lines
---------------------------------------------------------------	------------------

```
vi negCyc(int N, V<pair<pi,int>> ed) {
    vl d(N); vi p(N); int x = -1;
    rep(N) {
        x = -1; each(t,ed) if (ckmin(d[t.f.s],d[t.f.f]+t.s))
            p[t.f.s] = t.f.f, x = t.f.s;
        if (x == -1) return {};
    }
    rep(N) x = p[x]; // enter cycle
    vi cyc{x}; while (p[cyc.bk] != x) cyc.pb(p[cyc.bk]);
    reverse(all(cyc)); return cyc;
}
```

Basics/BellmanFord (7.3).h

Description: Shortest Path w/ negative edge weights Can be useful with linear programming Constraints of the form $x_i - x_j < k$

template<int SZ> struct BellmanFord {	f5ea68, 24 lines
----------------------------------------------------	------------------

```
    int n;
    vi adj[SZ];
    V<pair<pi,int>> ed;
    void ae(int u, int v, int w) {
        adj[u].pb(v), ed.pb({(u,v),w}); }
    ll dist[SZ];
    void genBad(int x) {
        // if x is reachable from negative cycle
        // -> update dists of all vertices which x can go to
        if (dist[x] == -INF) return;
        dist[x] = -INF;
        each(t,adj[x]) genBad(t);
    }
    void init(int _n, int s) {
        n = _n; F0R(i,n) dist[i] = INF;
        dist[s] = 0;
        F0R(i,n) each(a,ed) if (dist[a.f.f] < INF)
            ckmin(dist[a.f.s],dist[a.f.f]+a.s);
        each(a,ed) if (dist[a.f.f] < INF
            && dist[a.f.s] > dist[a.f.f]+a.s)
            genBad(a.f.s);
    }
};
```

7.2 Trees

Trees (10)/LCAjump (10.2).h

Description: Calculates least common ancestor in tree with verts $0 \dots N-1$ and root R using binary jumping.

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

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

struct LCA {	6b0ee9, 28 lines
---------------------	------------------

```
int N; V<vi> par, adj; vi depth;
void init(int _N) { N = _N;
    int d = 1; while ((1<<d) < N) ++d;
    par.assign(d,vi(N)); adj.rsz(N); depth.rsz(N);
}
void ae(int x, int y) { adj[x].pb(y), adj[y].pb(x); }
void gen(int R = 0) { par[0][R] = R; dfs(R); }
void dfs(int x = 0) {
    FOR(i,1,sz(par)) par[i][x] = par[i-1][par[i-1][x]];
    each(y,adj[x]) if (y != par[0][x])
        depth[y] = depth[par[0][y]=x]+1, dfs(y);
}
int jmp(int x, int d) {
    F0R(i,sz(par)) if ((d>>i)&1) x = par[i][x];
    return x; }
int lca(int x, int y) {
    if (depth[x] < depth[y]) swap(x,y);
    x = jmp(x,depth[x]-depth[y]); if (x == y) return x;
    R0F(i,sz(par)) {
        int X = par[i][x], Y = par[i][y];
        if (X != Y) x = X, y = Y;
    }
    return par[0][x];
}
int dist(int x, int y) { // # edges on path
    return depth[x]+depth[y]-2*depth[lca(x,y)]; }
};
```

Trees (10)/LCArmq (10.2).h

Description: Euler Tour LCA. Compress takes a subset S of nodes and computes the minimal subtree that contains all the nodes pairwise LCAs and compressing edges. Returns a list of (par, orig.index) representing a tree rooted at 0. The root points to itself.

Time: $\mathcal{O}(N \log N)$ build, $\mathcal{O}(1)$ LCA, $\mathcal{O}(|S| \log |S|)$ compress

"../data-structures/Static Range Queries (9.1)/RMQ (9.1).h"	e5a035, 28 lines
--------------------------------------------------------------------	------------------

```
struct LCA {
    int N; V<vi> adj;
    vi depth, pos, par, rev; // rev is for compress
    vpi tmp; RMQ<pi> r;
    void init(int _N) { N = _N; adj.rsz(N);
        depth = pos = par = rev = vi(N); }
    void ae(int x, int y) { adj[x].pb(y), adj[y].pb(x); }
    void dfs(int x) {
        pos[x] = sz(tmp); tmp.eb(depth[x],x);
        each(y,adj[x]) if (y != par[x]) {
            depth[y] = depth[par[y]=x]+1, dfs(y);
            tmp.eb(depth[x],x); }
    }
    void gen(int R = 0) { par[R] = R; dfs(R); r.init(tmp); }
    int lca(int u, int v){
        u = pos[u], v = pos[v]; if (u > v) swap(u,v);
        return r.query(u,v).s; }
    int dist(int u, int v) {
        return depth[u]+depth[v]-2*depth[lca(u,v)]; }
    vpi compress(vi S) {
        auto cmp = [&](int a, int b) { return pos[a] < pos[b]; };
        sort(all(S),cmp); R0F(i,sz(S)-1) S.pb(lca(S[i],S[i+1]));
        sort(all(S),cmp); S.erase(unique(all(S)),end(S));
        vpi ret({0,S[0]}); F0R(i,sz(S)) rev[S[i]] = i;
        FOR(i,1,sz(S)) ret.eb(rev[lca(S[i-1],S[i])],S[i]);
        return ret;
    }
};
```

Trees (10)/HLD (10.3).h

Description: Heavy-Light Decomposition, add val to verts and query sum in path/subtree.

Time: any tree path is split into $\mathcal{O}(\log N)$ parts

"../data-structures/LD Range Queries (9.2)/LazySeg (15.2).h"	1802e2, 48 lines
---------------------------------------------------------------------	------------------

```
template<int SZ, bool VALS_IN_EDGES> struct HLD {
    int N; vi adj[SZ];
    int par[SZ], root[SZ], depth[SZ], sz[SZ], ti;
    int pos[SZ]; vi rpos; // rpos not used but could be useful
    void ae(int x, int y) { adj[x].pb(y), adj[y].pb(x); }
    void dfsSz(int x) {
        sz[x] = 1;
        each(y,adj[x]) {
            par[y] = x; depth[y] = depth[x]+1;
            adj[y].erase(find(all(adj[y]),x));
            dfsSz(y); sz[x] += sz[y];
        }
    }
};
```

Graphs (7)

```

    if (sz[y] > sz[adj[x][0]]) swap(y,adj[x][0]);
}
}
void dfsHld(int x) {
    pos[x] = ti++; rpos.pb(x);
    each(y,adj[x]) {
        root[y] = (y == adj[x][0] ? root[x] : y);
        dfsHld(y);
    }
}
void init(int _N, int R = 0) { N = _N;
    par[R] = depth[R] = ti = 0; dfsSz(R);
    root[R] = R; dfsHld(R);
}
int lca(int x, int y) {
    for (; root[x] != root[y]; y = par[root[y]])
        if (depth[root[x]] > depth[root[y]]) swap(x,y);
    return depth[x] < depth[y] ? x : y;
}
LazySeg<ll,SZ> tree; // segtree for sum
template<class BinaryOp>
void processPath(int x, int y, BinaryOp op) {
    for (; root[x] != root[y]; y = par[root[y]]) {
        if (depth[root[x]] > depth[root[y]]) swap(x,y);
        op(pos[root[y]],pos[y]);
    }
    if (depth[x] > depth[y]) swap(x,y);
    op(pos[x]+VALS_IN_EDGES,pos[y]);
}
void modifyPath(int x, int y, int v) {
    processPath(x,y,[this,&v](int l, int r) {
        tree.upd(l,r,v); });
}
ll queryPath(int x, int y) {
    ll res = 0; processPath(x,y,[this,&res](int l, int r) {
        res += tree.query(l,r); });
    return res;
}
void modifySubtree(int x, int v) {
    tree.upd(pos[x]+VALS_IN_EDGES,pos[x]+sz[x]-1,v);
}
};

```

Trees (10)/Centroid (10.3).h

Description: The centroid of a tree of size N is a vertex such that after removing it, all resulting subtrees have size at most $\frac{N}{2}$. Supports updates in the form “add 1 to all verts v such that $dist(x, v) \leq y$.”

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

Time: $\mathcal{O}(N \log N)$ build, $\mathcal{O}(\log N)$ update and query

907e21, 54 lines

```

void add(vi& a, int b) { ckmin(b,sz(a)-1); if (b>=0) a[b]++; }
void prop(vi& a) { R0F(1,sz(a)-1) a[i] += a[i+1]; }
template<int SZ> struct Centroid {
    vi adj[SZ]; void ae(int a,int b){adj[a].pb(b),adj[b].pb(a);}
    bool done[SZ]; // processed as centroid yet
    int N,sub[SZ],cen[SZ],lev[SZ]; // subtree size, centroid anc
    int dist[32-__builtin_clz(SZ)][SZ]; // dists to all ancs
    vi stor[SZ], STOR[SZ];
    void dfs(int x, int p) { sub[x] = 1;
        each(y,adj[x]) if (!done[y] && y != p)
            dfs(y,x), sub[x] += sub[y];
    }
    int centroid(int x) {
        dfs(x,-1);
        for (int sz = sub[x];) {
            pi mx = {0,0};
            each(y,adj[x]) if (!done[y] && sub[y] < sub[x])
                ckmax(mx,{sub[y],y});
            if (mx.f*2 <= sz) return x;
            x = mx.s;
        }
    }
    void genDist(int x, int p, int lev) {
        dist[lev][x] = dist[lev][p]+1;
        each(y,adj[x]) if (!done[y] && y != p) genDist(y,x,lev);
    }
    void gen(int CEN, int _x) { // CEN = centroid above x
        int x = centroid(_x); done[x] = 1; cen[x] = CEN;
        sub[x] = sub[_x]; lev[x] = (CEN == -1 ? 0 : lev[CEN]+1);
        dist[lev[x]][x] = 0;
        stor[x].rsz(sub[x]),STOR[x].rsz(sub[x]+1);
        each(y,adj[x]) if (!done[y]) genDist(y,x,lev[x]);
        each(y,adj[x]) if (!done[y]) gen(x,y);
    }
    void init(int _N) { N = _N; FOR(i,1,N+1) done[i] = 0;
        gen(-1,1); } // start at vert 1
    void upd(int x, int y) {

```

Centroid (10 EulerPath (12 SCCT TwoSAT (12

```

int cur = x, pre = -1;
R0F(i,lev[x]+1) {
    ad(stor[cur],y-dist[i][x]);
    if (pre != -1) ad(STOR[pre],y-dist[i][x]);
    if (i > 0) pre = cur, cur = cen[cur];
}
} // call propAll() after all updates
void propAll() { FOR(i,1,N+1) prop(stor[i]), prop(STOR[i]); }
int query(int x) { // get value at vertex x
    int cur = x, pre = -1, ans = 0;
    R0F(i,lev[x]+1) { // if pre != -1, subtract those from
        ans += stor[cur][dist[i][x]]; // same subtree
        if (pre != -1) ans -= STOR[pre][dist[i][x]];
        if (i > 0) pre = cur, cur = cen[cur];
    }
    return ans;
}
};

```

7.2.1 SqrtDecompton

HLD generally suffices. If not, here are some common strategies:

- Rebuild the tree after every \sqrt{N} queries.
- Consider vertices with $>$ or $< \sqrt{N}$ degree separately.
- For subtree updates, note that there are $\mathcal{O}(\sqrt{N})$ distinct sizes among child subtrees of any node.

Block Tree: Use a DFS to split edges into contiguous groups of size \sqrt{N} to $2\sqrt{N}$.

Mo's Algorithm for Tree Paths: Maintain an array of vertices where each one appears twice, once when a DFS enters the vertex (st) and one when the DFS exists (en). For a tree path $u \leftrightarrow v$ such that $st[u] < st[v]$,

- If u is an ancestor of v , query $[st[u], st[v]]$.
- Otherwise, query $[en[u], st[v]]$ and consider $LCA(u, v)$ separately.

Solutions with worse complexities can be faster if you optimize the operations that are performed most frequently. Use arrays instead of vectors whenever possible. Iterating over an array in order is faster than iterating through the same array in some other order (ex. one given by a random permutation) or DFSing on a tree of the same size. Also, the difference between \sqrt{N} and the optimal block (or buffer) size can be quite large. Try up to 5x smaller or larger (at least).

7.3 DFS Algorithms

DFS/EulerPath (12.2).h

Description: Eulerian path starting at src if it exists, visits all edges exactly once. Works for both directed and undirected. Returns vector of {vertex,label of edge to vertex}. Second element of first pair is always -1.

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

9c222d, 23 lines

```

template<bool directed> struct Euler {
    int N; V<vpi> adj; V<vpi::iterator> its; vb used;
    void init(int _N) { N = _N; adj.rsz(N); }
    void ae(int a, int b) {
        int M = sz(used); used.pb(0);
        adj[a].eb(b,M); if (!directed) adj[b].eb(a,M);
    }
    vpi solve(int src = 0) {
        its.rsz(N); F0R(i,N) its[i] = begin(adj[i]);
        vpi ans, s({src,-1}); // {vert,prev vert},edge label
        int lst = -1; // ans generated in reverse order
        while (sz(s)) {
            int x = s.bk.f; auto& it=its[x], en=end(adj[x]);
            while (it != en && used[it->s]) ++it;
            if (it == en) { // no more edges out of vertex
                if (lst != -1 && lst != x) return {};
                // not a path, no tour exists

```

```

        ans.pb(s.bk); s.pop_back(); if (sz(s)) lst=s.bk.f;
        } else s.pb(*it), used[it->s] = 1;
    } // must use all edges
    if (sz(ans) != sz(used)+1) return {};
    reverse(all(ans)); return ans;
}
};

```

DFS/SCCT.h

Description: Tarjan's, DFS once to generate strongly connected components in topological order. a, b in same component if both $a \rightarrow b$ and $b \rightarrow a$ exist. Uses less memory than Kosaraju b/c doesn't store reverse edges.

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

a36e0c, 22 lines

```

struct SCC {
    int N, ti = 0; V<vi> adj;
    vi disc, comp, stk, comps;
    void init(int _N) { N = _N; adj.rsz(N);
        disc.rsz(N), comp.rsz(N,-1); }
    void ae(int x, int y) { adj[x].pb(y); }
    int dfs(int x) {
        int low = disc[x] = ++ti; stk.pb(x);
        each(y,adj[x]) if (comp[y] == -1) // comp[y] == -1,
            ckmin(low,disc[y]?dfs(y)); // disc[y] != 0 -> in stack
        if (low == disc[x]) { // make new SCC
            // pop off stack until you find x
            comps.pb(x); for (int y = -1; y != x;)
                comp[y = stk.bk] = x, stk.pop_back();
        }
        return low;
    }
    void gen() {
        F0R(i,N) if (!disc[i]) dfs(i);
        reverse(all(comps));
    }
};

```

DFS/TwoSAT (12.1).h

Description: Calculates a valid assignment to boolean variables a, b, c, \dots to a 2-SAT problem, so that an expression of the type $(a \vee b) \wedge (a \vee c) \wedge (d \vee b) \wedge \dots$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions ($\sim x$).

Usage: TwoSat ts;

```

ts.either(0, ~3); // Var 0 is true or var 3 is false
ts.setVal(2); // Var 2 is true
ts.atMostOne({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true
ts.solve(N); // Returns true iff it is solvable
ts.ans[0..N-1] holds the assigned values to the vars

```

"SCC (12.1).h"

ff0f3d, 31 lines

```

struct TwoSAT {
    int N = 0; vpi edges;
    void init(int _N) { N = _N; }
    int addVar() { return N++; }
    void either(int x, int y) {
        x = max(2*x,-1-2*x), y = max(2*y,-1-2*y);
        edges.eb(x,y);
    }
    void implies(int x, int y) { either(~x,y); }
    void must(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]);
    }
    vb solve() {
        SCC S; S.init(2*N);
        each(e,edges) S.ae(e.f^1,e.s), S.ae(e.s^1,e.f);
        S.gen(); reverse(all(S.comps)); // reverse topo order
        for (int i = 0; i < 2*N; i += 2)
            if (S.comp[i] == S.comp[i^1]) return {};
        vi tmp(2*N); each(i,S.comps) if (!tmp[i])
            tmp[i] = 1, tmp[S.comp[i^1]] = -1;
        vb ans(N); F0R(i,N) ans[i] = tmp[S.comp[2*i]] == 1;
        return ans;
    }
};

```

DFS/BCC (12.4).h

Description: Disconnected components of edges. Removing any vertex in BCC doesn't disconnect it. To get block-cut tree, create a bipartite graph with the original vertices on the left and a vertex for each BCC on the right. Draw edge $u \leftrightarrow v$ if u is contained within the BCC for v . Self-loops are not included in any BCC while BCCS of size 1 represent bridges.

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

0625a6, 35 lines

```
struct BCC {
    V<vpi> adj; vpi ed;
    V<vi> edgeSets, vertSets; // edges for each bcc
    int N, ti = 0; vi disc, stk;
    void init(int _N) { N = _N; disc.rsz(N), adj.rsz(N); }
    void ae(int x, int y) {
        adj[x].eb(y, sz(ed)), adj[y].eb(x, sz(ed)), ed.eb(x, y); }
    int dfs(int x, int p = -1) { // return lowest disc
        int low = disc[x] = ++ti;
        each(e, adj[x]) if (e.s != p) {
            if (!disc[e.f]) {
                stk.pb(e.s); // disc[x] < LOW -> bridge
                int LOW = dfs(e.f, e.s); ckmin(low, LOW);
                if (disc[x] <= LOW) { // get edges in bcc
                    edgeSets.eb(); vi& tmp = edgeSets.bk; // new bcc
                    for (int y = -1; y != e.s; )
                        tmp.pb(y = stk.bk), stk.pop_back();
                }
            } else if (disc[e.f] < disc[x]) // back-edge
                ckmin(low, disc[e.f]), stk.pb(e.s);
        }
        return low;
    }
    void gen() {
        F0R(i, N) if (!disc[i]) dfs(i);
        vb in(N);
        each(c, edgeSets) { // edges contained within each BCC
            vertSets.eb(); // so you can easily create block cut tree
            auto ad = [&](int x) {
                if (!in[x]) in[x] = 1, vertSets.bk.pb(x); };
            each(e, c) ad(ed[e].f), ad(ed[e].s);
            each(e, c) in[ed[e].f] = in[ed[e].s] = 0;
        }
    }
};
```

DFS/MaximalCliques.h

Description: Used only once. Finds all maximal cliques.

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

f5cd93, 16 lines

```
using B = bitset<128>; B adj[128];
int N;
// possibly in clique, not in clique, in clique
void cliques(B P = ~B(), B X={}, B R={}) {
    if (!P.any()) {
        if (!X.any()) // do smth with R
            return;
    }
    int q = (P|X)._Find_first();
    // clique must contain q or non-neighbor of q
    B cands = P&~adj[q];
    F0R(i, N) if (cands[i]) {
        R[i] = 1; cliques(P&adj[i], X&adj[i], R);
        R[i] = P[i] = 0; X[i] = 1;
    }
}
```

7.4 Flows

Konig's Theorem: In a bipartite graph, max matching = min vertex cover.

Dilworth's Theorem: For any partially ordered set, the sizes of the max antichain and of the min chain decomposition are equal. Equivalent to Konig's theorem on the bipartite graph (U, V, E) where $U = V = S$ and (u, v) is an edge when $u < v$. Those vertices outside the min vertex cover in both U and V form a max antichain.

Flows (12.3)/Dinic.h

Description: Fast flow. After computing flow, edges $\{u, v\}$ such that $lev[u] \neq -1, lev[v] = -1$ are part of min cut. Use reset and rcap for Gomory-Hu.

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

b7b370, 38 lines

```
struct Dinic {
    using F = ll; // flow type
    struct Edge { int to; F flo, cap; };
    int N; V<Edge> eds; V<vi> adj;
    void init(int _N) { N = _N; adj.rsz(N), cur.rsz(N); }
    void ae(int u, int v, F cap, F rcap = 0) { assert(min(cap, rcap) >=
        ↳0);
        adj[u].pb(sz(eds)); eds.pb({v, 0, cap});
        adj[v].pb(sz(eds)); eds.pb({u, 0, rcap});
    }
    vi lev; V<vi::iterator> cur;
    bool bfs(int s, int t) { // level = shortest distance from source
        lev = vi(N, -1); F0R(i, N) cur[i] = begin(adj[i]);
        queue<int> q({s}); lev[s] = 0;
        while (sz(q)) { int u = q.ft; q.pop();
            each(e, adj[u]) { const Edge& E = eds[e];
                int v = E.to; if (lev[v] < 0 && E.flo < E.cap)
                    q.push(v), lev[v] = lev[u] + 1;
            }
        }
        return lev[t] >= 0;
    }
    F dfs(int v, int t, F flo) {
        if (v == t) return flo;
        for (; cur[v] != end(adj[v]); cur[v]++) {
            Edge& E = eds[*cur[v]];
            if (lev[E.to] != lev[v] + 1 || E.flo == E.cap) continue;
            F df = dfs(E.to, t, min(flo, E.cap - E.flo));
            if (df) { E.flo += df; eds[*cur[v]^1].flo -= df;
                return df; } // saturated >= 1 one edge
        }
        return 0;
    }
    F maxFlow(int s, int t) {
        F tot = 0; while (bfs(s, t)) while (F df =
            dfs(s, t, numeric_limits<F>::max())) tot += df;
        return tot;
    }
};
```

Flows (12.3)/GomoryHu.h

Description: Returns edges of Gomory-Hu tree (second element is weight). Max flow between pair of vertices of undirected graph is given by min edge weight along tree path. Uses the fact that for any $i, j, k, \lambda_{ik} \geq \min(\lambda_{ij}, \lambda_{jk})$, where λ_{ij} denotes the flow between i and j .

Time: $N - 1$ calls to Dinic

"Dinic.h" 0d712e, 16 lines

```
template<class F> V<pair<pi, F>> gomoryHu(int N,
    const V<pair<pi, F>>& ed) {
    vi par(N); Dinic<F> D; D.init(N);
    vpi ed_locs; each(t, ed) ed_locs.pb(D.ae(t.f.f, t.f.s, t.s, t.s));
    V<pair<pi, F>> ans;
    F0R(i, 1, N) {
        each(p, ed_locs) { // reset capacities
            auto& e = D.adj.at(p.f).at(p.s);
            auto& e_rev = D.adj.at(e.to).at(e.rev);
            e.cap = e_rev.cap = (e.cap + e_rev.cap) / 2;
        }
        ans.pb({{i, par[i]}, D.maxFlow(i, par[i])});
        F0R(j, i + 1, N) if (par[j] == par[i] && D.lev[j]) par[j] = i;
    }
    return ans;
}
```

Flows (12.3)/MCMF.h

Description: Minimum-cost maximum flow, assumes no negative cycles. It is possible to choose negative edge costs such that the first run of Dijkstra is slow, but this hasn't been an issue in the past. Edge weights ≥ 0 for every subsequent run. To get flow through original edges, assign ID's during ae.

Time: Ignoring first run of Dijkstra, $\mathcal{O}(FM \log M)$ if caps are integers and F is max flow.

072c1b, 40 lines

```
struct MCMF {
    using F = ll; using C = ll; // flow type, cost type
    struct Edge { int to; F flo, cap; C cost; };
```

```
int N; V<C> p, dist; vi pre; V<Edge> eds; V<vi> adj;
void init(int _N) { N = _N;
    p.rsz(N), dist.rsz(N), pre.rsz(N), adj.rsz(N); }
void ae(int u, int v, F cap, C cost) { assert(cap >= 0);
    adj[u].pb(sz(eds)); eds.pb({v, 0, cap, cost});
    adj[v].pb(sz(eds)); eds.pb({u, 0, 0, -cost});
} // use asserts, don't try smth dumb
bool path(int s, int t) { // find lowest cost path to send flow
    ↳through
    const C inf = numeric_limits<C>::max(); F0R(i, N) dist[i] = inf;
    using T = pair<C, int>; priority_queue<T, vector<T>, greater<T>>
        ↳todo;
    todo.push({dist[s] = 0, s});
    while (sz(todo)) { // Dijkstra
        T x = todo.top(); todo.pop(); if (x.f > dist[x.s]) continue;
        each(e, adj[x.s]) { const Edge& E = eds[e]; // all weights
            ↳should be non-negative
            if (E.flo < E.cap && ckmin(dist[E.to], x.f + E.cost + p[x.s] - p[E.
                ↳to]))
                pre[E.to] = e, todo.push({dist[E.to], E.to});
        }
    } // if costs are doubles, add some EPS so you
    // don't traverse ~0-weight cycle repeatedly
    return dist[t] != inf; // return flow
}
pair<F, C> calc(int s, int t) { assert(s != t);
    F0R(_, N) F0R(e, sz(eds)) { const Edge& E = eds[e]; // Bellman-Ford
        if (E.cap) ckmin(p[E.to], p[eds[e^1].to] + E.cost); }
    F totFlow = 0; C totCost = 0;
    while (path(s, t)) { // p -> potentials for Dijkstra
        F0R(i, N) p[i] += dist[i]; // don't matter for unreachable nodes
        F df = numeric_limits<F>::max();
        for (int x = t; x != s; x = eds[pre[x]^1].to) {
            const Edge& E = eds[pre[x]]; ckmin(df, E.cap - E.flo); }
        totFlow += df; totCost += (p[t] - p[s]) * df;
        for (int x = t; x != s; x = eds[pre[x]^1].to)
            eds[pre[x]].flo += df, eds[pre[x]^1].flo -= df;
    } // get max flow you can send along path
    return {totFlow, totCost};
}
};
```

Flows (12.3)/GlobalMinCut.h

Description: Used only once. Stoer-Wagner, find a global minimum cut in an undirected graph as represented by an adjacency matrix.

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

6f4dcb, 25 lines

```
pair<int, vi> GlobalMinCut(V<vi> wei) {
    int N = sz(wei);
    vi par(N); iota(all(par), 0);
    pair<int, vi> bes(INT_MAX, {});
    R0F(phase, N) {
        vi w = wei[0]; int lst = 0;
        vector<bool> add(N, 1); F0R(i, 1, N) if (par[i] == i) add[i] = 0;
        F0R(i, phase) {
            int k = -1;
            F0R(j, 1, N) if (!add[j] && (k == -1 || w[j] > w[k])) k = j;
            if (i + 1 == phase) {
                if (w[k] < bes.f) {
                    bes = {w[k], {}};
                    F0R(j, N) if (par[j] == k) bes.s.pb(j);
                }
                F0R(j, N) wei[lst][j] += wei[k][j], wei[j][lst] = wei[lst][j];
                F0R(j, N) if (par[j] == k) par[j] = lst; // merge
            } else { // greedily add closest
                F0R(j, N) w[j] += wei[k][j];
                add[lst = k] = 1;
            }
        }
    }
    return bes;
}
```

7.5 Matching

Matching/Hungarian.h

Description: Given J jobs and W workers ($J \leq W$), computes the minimum cost to assign each prefix of jobs to distinct workers.

@tparam T a type large enough to represent integers on the order of $J * \max(\text{---})$ @param C a matrix of dimensions $J \times W$ such that $C[j][w] = \text{cost}$ to assign j -th job to w -th worker (possibly negative)

@return a vector of length J, with the j-th entry equaling the minimum cost to assign the first (j+1) jobs to distinct workers

Time: $\mathcal{O}(J^2W)$ f382f7, 36 lines

```
template<class T> vector<T> hungarian(const vector<vector<T>> &C) {
    const int J = (int)size(C), W = (int)size(C[0]);
    assert(J <= W);
    vector<int> job(W + 1, -1);
    vector<T> ys(J), yt(W + 1);
    vector<T> answers;
    const T inf = numeric_limits<T>::max();
    for (int j_cur = 0; j_cur < J; ++j_cur) {
        int w_cur = W;
        job[w_cur] = j_cur;
        vector<T> min_to(W + 1, inf);
        vector<int> prv(W + 1, -1);
        vector<bool> in_Z(W + 1);
        while (job[w_cur] != -1) {
            in_Z[w_cur] = true;
            const int j = job[w_cur];
            T delta = inf;
            int w_next;
            for (int w = 0; w < W; ++w) {
                if (!in_Z[w]) {
                    if (ckmin(min_to[w], C[j][w] - ys[j] - yt[w]))
                        prv[w] = w_cur;
                    if (ckmin(delta, min_to[w])) w_next = w;
                }
            }
            for (int w = 0; w <= W; ++w) {
                if (in_Z[w]) ys[job[w]] += delta, yt[w] -= delta;
                else min_to[w] -= delta;
            }
            w_cur = w_next;
        }
        for (int w; w_cur != -1; w_cur = w) job[w_cur] = job[w = prv[
            ↪w_cur]];
        answers.push_back(-yt[W]);
    }
    return answers;
}
```

Matching/GeneralMatchBlossom.h

Description: Variant on Gabow's Impl of Edmond's Blossom Algorithm. General unweighted max matching with 1-based indexing. If white[v] = 0 after solve(i) returns, v is part of every max matching.

Time: $\mathcal{O}(NM)$, faster in practice fd5cc7, 50 lines

```
struct MaxMatching {
    int N; V<vi> adj;
    V<int> mate, first; vb white; vpi label;
    void init(int _N) { N = _N; adj = V<vi>(N+1);
        mate = first = vi(N+1); label = vpi(N+1); white = vb(N+1); }
    void ae(int u, int v) { adj.at(u).pb(v), adj.at(v).pb(u); }
    int group(int x) { if (white[first[x]]) first[x] = group(first[x]);
        return first[x]; }
    void match(int p, int b) {
        swap(b, mate[p]); if (mate[b] != p) return;
        if (!label[p].s) mate[b] = label[p].f, match(label[p].f, b); //
            ↪vertex label
        else match(label[p].f, label[p].s), match(label[p].s, label[p].f);
            ↪// edge label
    }
    bool augment(int st) { assert(st);
        white[st] = 1; first[st] = 0; label[st] = {0,0};
        queue<int> q; q.push(st);
        while (!q.empty()) {
            int a = q.ft; q.pop(); // outer vertex
            each(b, adj[a]) { assert(b);
                if (white[b]) { // two outer vertices, form blossom
                    int x = group(a), y = group(b), lca = 0;
                    while (x||y) {
                        if (y) swap(x,y);
                        if (label[x] == pi{a,b}) { lca = x; break; }
                        label[x] = {a,b}; x = group(label[mate[x]].first);
                    }
                }
                for (int v: {group(a),group(b)}) while (v != lca) {
                    assert(!white[v]); // make everything along path white
                    q.push(v); white[v] = true; first[v] = lca;
                    v = group(label[mate[v]].first);
                }
            }
        }
    }
```

```
} else if (!mate[b]) { // found augmenting path
    mate[b] = a; match(a,b); white = vb(N+1); // reset
    return true;
} else if (!white[mate[b]]) {
    white[mate[b]] = true; first[mate[b]] = b;
    label[b] = {0,0}; label[mate[b]] = pi{a,0};
    q.push(mate[b]);
}
}
}
return false;
}
int solve() {
    int ans = 0;
    FOR(st,1,N+1) if (!mate[st]) ans += augment(st);
    FOR(st,1,N+1) if (!mate[st] && !white[st]) assert(!augment(st));
    return ans;
}
};
```

Matching/GeneralWeightedMatch.h

Description: General max weight max matching with 1-based indexing. Edge weights must be positive, combo of UnweightedMatch and Hungarian.

Time: $\mathcal{O}(N^3)$? 120873, 145 lines

```
template<int SZ> struct WeightedMatch {
    struct edge { int u,v,w; }; edge g[SZ*2][SZ*2];
    void ae(int u, int v, int w) { g[u][v].w = g[v][u].w = w; }
    int N,NX,lab[SZ*2],match[SZ*2],slack[SZ*2],st[SZ*2];
    int par[SZ*2],floFrom[SZ*2][SZ],S[SZ*2],aux[SZ*2];
    vi flo[SZ*2]; queue<int> q;
    void init(int _N) { N = _N; // init all edges
        FOR(u,1,N+1) FOR(v,1,N+1) g[u][v] = {u,v,0}; }
    int eDelta(edge e) { // >= 0 at all times
        return lab[e.u]+lab[e.v]-g[e.u][e.v].w*2; }
    void updSlack(int u, int x) { // smallest edge -> blossom x
        if (!slack[x] || eDelta(g[u][x]) < eDelta(g[slack[x]][x]))
            slack[x] = u; }
    void setSlack(int x) {
        slack[x] = 0; FOR(u,1,N+1) if (g[u][x].w > 0
            && st[u] != x && S[st[u]] == 0) updSlack(u,x); }
    void qPush(int x) {
        if (x <= N) q.push(x);
        else each(t,flo[x]) qPush(t); }
    void setSt(int x, int b) {
        st[x] = b; if (x > N) each(t,flo[x]) setSt(t,b); }
    int getPr(int b, int xr) { // get even position of xr
        int pr = find(all(flo[b]),xr)-begin(flo[b]);
        if (pr&1) { reverse(1+all(flo[b])); return sz(flo[b])-pr; }
        return pr; }
    void setMatch(int u, int v) { // rearrange flo[u], matches
        edge e = g[u][v]; match[u] = e.v; if (u <= N) return;
        int xr = floFrom[u][e.u], pr = getPr(u,xr);
        FOR(i,pr) setMatch(flo[u][i],flo[u][i^1]);
        setMatch(xr,v); rotate(begin(flo[u]),pr+all(flo[u])); }
    void augment(int u, int v) { // set matches including u->v
        while (1) { // and previous ones
            int xnv = st[match[u]]; setMatch(u,v);
            if (!xnv) return;
            setMatch(xnv,st[par[xnv]]);
            u = st[par[xnv]], v = xnv;
        }
    }
    int lca(int u, int v) { // same as in unweighted
        static int t = 0; // except maybe return 0
        for (++t;u||v;swap(u,v)) {
            if (!u) continue;
            if (aux[u] == t) return u;
            aux[u] = t; u = st[match[u]];
            if (u) u = st[par[u]];
        }
        return 0;
    }
    void addBlossom(int u, int anc, int v) {
        int b = N+1; while (b <= NX && st[b]) ++b;
        if (b > NX) ++NX; // new blossom
        lab[b] = S[b] = 0; match[b] = match[anc]; flo[b] = {anc};
        auto blossom = [&](int x) {
            for (int y; x != anc; x = st[par[y]])
                flo[b].pb(x), flo[b].pb(y = st[match[x]]), qPush(y);
        };
    }
```

```
blossom(u); reverse(1+all(flo[b])); blossom(v); setSt(b,b);
// identify all nodes in current blossom
FOR(x,1,NX+1) g[b][x].w = g[x][b].w = 0;
FOR(x,1,N+1) floFrom[b][x] = 0;
each(xs,flo[b]) { // find tightest constraints
    FOR(x,1,NX+1) if (g[b][x].w == 0 || eDelta(g[xs][x]) <
        eDelta(g[b][x])) g[b][x]=g[xs][x], g[x][b]=g[x][xs];
    FOR(x,1,N+1) if (floFrom[xs][x]) floFrom[b][x] = xs;
} // floFrom to deconstruct blossom
setSlack(b); // since didn't qPush everything
}
void expandBlossom(int b) {
    each(t,flo[b]) setSt(t,t); // undo setSt(b,b)
    int xr = floFrom[b][g[b][par[b]].u], pr = getPr(b,xr);
    for(int i = 0; i < pr; i += 2) {
        int xs = flo[b][i], xns = flo[b][i+1];
        par[xs] = g[xns][xs].u; S[xs] = 1; // no setSlack(xns)?
        S[xns] = slack[xs] = slack[xns] = 0; qPush(xns);
    }
    S[xr] = 1, par[xr] = par[b];
    FOR(i,pr+1,sz(flo[b])) { // matches don't change
        int xs = flo[b][i]; S[xs] = -1, setSlack(xs); }
    st[b] = 0; // blossom killed
}
bool onFoundEdge(edge e) {
    int u = st[e.u], v = st[e.v];
    if (S[v] == -1) { // v unvisited, matched with smth else
        par[v] = e.u, S[v] = 1; slack[v] = 0;
        int nu = st[match[v]]; S[nu] = slack[nu] = 0; qPush(nu);
    } else if (S[v] == 0) {
        int anc = lca(u,v); // if 0 then match found!
        if (!anc) return augment(u,v),augment(v,u),1;
        addBlossom(u,anc,v);
    }
    return 0;
}
bool matching() {
    q = queue<int>();
    FOR(x,1,NX+1) {
        S[x] = -1, slack[x] = 0; // all initially unvisited
        if (st[x] == x && !match[x]) par[x] = S[x] = 0, qPush(x);
    }
    if (!sz(q)) return 0;
    while (1) {
        while (sz(q)) { // unweighted matching with tight edges
            int u = q.ft; q.pop(); if (S[st[u]] == 1) continue;
            FOR(v,1,N+1) if (g[u][v].w > 0 && st[u] != st[v]) {
                if (eDelta(g[u][v]) == 0) { // condition is strict
                    if (onFoundEdge(g[u][v])) return 1;
                } else updSlack(u,st[v]);
            }
        }
        int d = INT_MAX;
        FOR(b,N+1,NX+1) if (st[b] == b && S[b] == 1)
            ckmin(d,lab[b]/2); // decrease lab[b]
        FOR(x,1,NX+1) if (st[x] == x && slack[x]) {
            if (S[x] == -1) ckmin(d,eDelta(g[slack[x]][x]));
            else if (S[x] == 0) ckmin(d,eDelta(g[slack[x]][x])/2);
        } // edge weights shouldn't go below 0
        FOR(u,1,N+1) {
            if (S[st[u]] == 0) {
                if (lab[u] <= d) return 0; // why?
                lab[u] -= d;
            } else if (S[st[u]] == 1) lab[u] += d;
        } // lab has opposite meaning for verts and blossoms
        FOR(b,N+1,NX+1) if (st[b] == b && S[b] != -1)
            lab[b] += (S[b] == 0 ? 1 : -1)*d*2;
        q = queue<int>();
        FOR(x,1,NX+1) if (st[x]==x && slack[x] // new tight edge
            && st[slack[x]] != x && eDelta(g[slack[x]][x]) == 0)
            if (onFoundEdge(g[slack[x]][x])) return 1;
        FOR(b,N+1,NX+1) if (st[b]==b && S[b]==1 && lab[b]==0)
            expandBlossom(b); // odd dist blossom taken apart
    }
    return 0;
}
pair<ll,int> calc() {
    NX = N; st[0] = 0; FOR(i,1,2*N+1) aux[i] = 0;
    FOR(i,1,N+1) match[i] = 0, st[i] = i, flo[i].clear();
    int wMax = 0;
    FOR(u,1,N+1) FOR(v,1,N+1)
```

```

        floFrom[u][v] = (u == v ? u : 0); ckmax(wMax,g[u][v].w);
FOR(u,1,N+1) lab[u] = wMax; // start high and decrease
int num = 0; ll wei = 0; while (matching()) ++num;
FOR(u,1,N+1) if (match[u] && match[u] < u)
    wei += g[u][match[u]].w; // edges in matching
return {wei,num};
}
};

```

Matching/MaxMatchLexMin.h

Description: lexicographically least matching wrt left vertices

Usage: solve(L,R,sz(L))

Time: $\log |L|$ times sum of complexities of gen, maxMatch 44bb49, 26 lines

```

vpi maxMatch(vi L, vi R); // return pairs in max matching
pair<vi,vi> gen(vi L, vi R); // return {Lp,Rp}, vertices on
// left/right that can be reached by alternating path from
// unmatched node on left after finding max matching

```

```

vpi res; // stores answer
void solve(vi L, vi R, int x) { // first |L|-x elements of L
    if (x <= 1) { // are in matching, easy if x <= 1
        vpi v = maxMatch(L,R);
        if (sz(v) != sz(L)) L.pop_back(), v = maxMatch(L,R);
        assert(sz(v) == sz(L));
        res.insert(end(res),all(v)); return;
    }
    vi Lp,Rp; tie(Lp,Rp)=gen(L,R); vi Lm=sub(L,Lp),Rm=sub(R,Rp);
    // Lp U Rm is max indep set, Lm U Rp is min vertex cover
    // Lp and Rm independent, edges from Lm to Rp can be ignored
    vpi v = maxMatch(Lm,Rm); assert(sz(v) == sz(Lm));
    res.insert(end(res),all(v));
    vi L2(all(L)-x/2); vi Lp2,Rp2; tie(Lp2,Rp2) = gen(L2,R);
    int cnt = 0; each(t,Lp2) cnt += t >= L[sz(L)-x];
    solve(Lp2,Rp2,cnt); // Rp2 covered by best matching
    vi LL = sub(Lp,Lp2), RR = sub(Rp,Rp2); // those in Lp but not
    // Lp2 that are < L[sz(L)-x/2] must be in answer, not cnt
    cnt = 0; each(t,LL) cnt += t >= L[sz(L)-x/2];
    solve(LL,RR,cnt); // do rest
} // x reduced by factor of at least two

```

Matching/MaxMatchFast.h

Description: Fast bipartite matching.

Time: $\mathcal{O}(M\sqrt{N})$ ec6c96, 31 lines

```

vpi maxMatch(int L, int R, const vpi& edges) {
    V<vi> adj = V<vi>(L);
    vi nxt(L,-1), prv(R,-1), lev, ptr;
    F0R(i,sz(edges)) adj.at(edges[i].f).pb(edges[i].s);
    while (true) {
        lev = ptr = vi(L); int max_lev = 0;
        queue<int> q; F0R(i,L) if (nxt[i]==-1) lev[i]=1, q.push(i);
        while (sz(q)) {
            int x = q.ft; q.pop();
            for (int y: adj[x]) {
                int z = prv[y];
                if (z == -1) max_lev = lev[x];
                else if (!lev[z]) lev[z] = lev[x]+1, q.push(z);
            }
            if (max_lev) break;
        }
        if (!max_lev) break;
        F0R(i,L) if (lev[i] > max_lev) lev[i] = 0;
        auto dfs = [&](auto self, int x) -> bool {
            for (;ptr[x] < sz(adj[x]);++ptr[x]) {
                int y = adj[x][ptr[x]], z = prv[y];
                if (z == -1 || (lev[z] == lev[x]+1 && self(self,z)))
                    return nxt[x]=y, prv[y]=x, ptr[x]=sz(adj[x]), 1;
            }
            return 0;
        };
        F0R(i,L) if (nxt[i] == -1) dfs(dfs,i);
    }
    vpi ans; F0R(i,L) if (nxt[i] != -1) ans.pb({i,nxt[i]});
    return ans;
}

```

7.6 Advanced

Advanced/MaxClique.h

Description: Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). To find maximum independent set consider complement.

Time: Runs in about 1s for $n = 155$ and worst case random graphs ($p = .90$). Faster for sparse graphs.

```

e80bc7, 41 lines

struct MaxClique {
    db limit = 0.025, pk = 0; // # of steps
    struct Vertex { int i, d=0; Vertex(int _i):i(_i){} };
    typedef vector<Vertex> vv; vv V;
    vector<bitset<200>> e; vector<vi> C; // colors
    vi qmax,q,S,old; // max/current clique, sum # steps up to lev
    void init(vv& r) { // v.d -> degree
        each(v,r) { v.d = 0; each(j,r) v.d += e[v.i][j.i]; }
        sort(all(r),[](Vertex a,Vertex b) { return a.d > b.d; });
        int mxD = r[0].d; F0R(i,sz(r)) r[i].d = min(i,mxD)+1;
    }
    void expand(vv& R, int lev = 1) {
        S[lev] += S[lev-1]-old[lev]; old[lev] = S[lev-1];
        while (sz(R)) {
            if (sz(q)+R.bk.d <= sz(qmax)) return; // no larger clique
            q.pb(R.bk.i); // insert node with max col into clique
            vv T; each(v,R) if (e[R.bk.i][v.i]) T.pb({v.i});
            if (sz(T)) {
                if (S[lev]++/+pk < limit) init(T); // recalc degs
                int j = 0, mxk = 1, mnk = max(sz(qmax)-sz(q)+1,1);
                C[1].clear(), C[2].clear();
                each(v,T) {
                    int k = 1; auto f = [&](int i) { return e[v.i][i]; };
                    while (any_of(all(C[k]),f)) k ++;
                    if (k > mxk) mxk = k, C[mxk+1].clear(); // new set
                    if (k < mnk) T[j++].i = v.i;
                    C[k].pb(v.i);
                }
                if (j > 0) T[j-1].d = 0; // >=1 vert >=j part of clique
                FOR(k,mnk,mxk+1) each(i,C[k]) T[j].i = i, T[j++].d = k;
                expand(T,lev+1);
            } else if (sz(q) > sz(qmax)) qmax = q;
            q.pop_back(), R.pop_back(); // R.bk not in set
        }
    }
    vi solve(vector<bitset<200>> conn) {
        e = conn; C.rsz(sz(e)+1), S.rsz(sz(C)), old = S;
        F0R(i,sz(e)) V.pb({i});
        init(V), expand(V); return qmax;
    }
};

```

Advanced/ChordalGraphRecognition.h

Description: Recognizes graph where every induced cycle has length exactly 3 using maximum adjacency search.

```

6cc97d, 58 lines

int N,M;
set<int> adj[MX];
int cnt[MX];
vi ord, rord;

vi find_path(int x, int y, int z) {
    vi pre(N,-1);
    queue<int> q; q.push(x);
    while (sz(q)) {
        int t = q.ft; q.pop();
        if (adj[t].count(y)) {
            pre[y] = t; vi path = {y};
            while (path.bk != x) path.pb(pre[path.bk]);
            path.pb(z);
            return path;
        }
        each(u,adj[t]) if (u != z && !adj[u].count(z) && pre[u] == -1) {
            pre[u] = t;
            q.push(u);
        }
    }
    assert(0);
}

int main() {
    setIO(); re(N,M);
    F0R(i,M) {

```

```

        int a,b; re(a,b);
        adj[a].insert(b), adj[b].insert(a);
    }
    rord = vi(N,-1);
    priority_queue<pi> pq;
    F0R(i,N) pq.push({0,i});
    while (sz(pq)) {
        pi p = pq.top(); pq.pop();
        if (rord[p.s] != -1) continue;
        rord[p.s] = sz(ord); ord.pb(p.s);
        each(t,adj[p.s]) pq.push({++cnt[t],t});
    }
    assert(sz(ord) == N);
    each(z,ord) {
        pi big = {-1,-1};
        each(y,adj[z]) if (rord[y] < rord[z])
            ckmax(big,mp(rord[y],y));
        if (big.f == -1) continue;
        int y = big.s;
        each(x,adj[z]) if (rord[x] < rord[y]) if (!adj[y].count(x)) {
            ps("NO");
            vi v = find_path(x,y,z);
            ps(sz(v));
            each(t,v) pr(t,' ');
            exit(0);
        }
    }
    ps("YES");
    reverse(all(ord));
    each(z,ord) pr(z,' ');
}

```

Advanced/DominatorTree.h

Description: Used only a few times. Assuming that all nodes are reachable from *root*, *a* dominates *b* iff every path from *root* to *b* passes through *a*.

Time: $\mathcal{O}(M \log N)$ 4b8836, 41 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 = 0;
    int par[SZ], bes[SZ];
    void ae(int a, int b) { adj[a].pb(b); }
    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;
        each(y,adj[x]) {
            if (!label[y]) {
                dfs(y); child[label[x]].pb(label[y]); }
            radj[label[y]].pb(label[x]);
        }
    }
    void init(int root) {
        dfs(root);
        ROF(i,1,co+1) {
            each(j,radj[i]) ckmin(sdom[i],sdom[get(j)]);
            if (i > 1) sdomChild[sdom[i]].pb(i);
            each(j,sdomChild[i]) {
                int k = get(j);
                if (sdom[j] == sdom[k]) dom[j] = sdom[j];
                else dom[j] = k;
            }
            each(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]);
        }
    }
};

```

Advanced/EdgeColor.h

Description: Used only once. Naive implementation of Misra & Gries edge coloring. By Vizing's Theorem, a simple graph with max degree d can be edge colored with at most $d + 1$ colors

Time: $\mathcal{O}(N^2M)$, faster in practice

cc2b29, 40 lines

```
template<int SZ> struct EdgeColor {
    int N = 0, maxDeg = 0, adj[SZ][SZ], deg[SZ];
    void init(int _N) { N = _N;
        F0R(i,N) { deg[i] = 0; F0R(j,N) adj[i][j] = 0; } }
    void ae(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; }
    V<bool> genCol(int x) {
        V<bool> col(N+1); F0R(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) {
        F0R(i,N) if (adj[x][i] == d)
            delEdge(x,i), invert(i,c,d), ae(x,i,c); }
    void ae(int u, int v) {
        // 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);
        F0R(i,1,maxDeg+2) if (!a[i] && !b[i])
            return ae(u,v,i);
        V<bool> use(N); vi fan = {v}; use[v] = 1;
        while (1) {
            auto col = genCol(fan.bk);
            if (sz(fan) > 1) col[adj[fan.bk][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;
        }
        int c = freeCol(u), d = freeCol(fan.bk); invert(u,d,c);
        int i = 0; while (i < sz(fan) && genCol(fan[i])[d]
            && adj[u][fan[i]] != d) i++;
        assert(i != sz(fan));
        F0R(j,i) ae(u,fan[j],delEdge(u,fan[j+1]));
        ae(u,fan[i],d);
    }
};
```

Advanced/DirectedMST.h

Description: Chu-Liu-Edmonds algorithm. Computes minimum weight directed spanning tree rooted at r , edge from $par[i] \rightarrow i$ for all $i \neq r$. Use DSU with rollback if need to return edges.

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

"../DSU/DSUrb (15.5).h"

5d5c10, 61 lines

```
struct Edge { int a, b; ll w; };
struct Node { // lazy skew heap 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 ? 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);
    vector<Node*> heap(n); // store edges entering each vertex
    // in increasing order of weight
    each(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}); // edge entering each vertex in MST
```

```
vector<pair<int,vector<Edge>>> cys;
F0R(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) { // found cycle, contract
            Node* cyc = 0; cys.pb(u);
            do {
                cyc = merge(cyc, heap[w = path.bk.f]);
                cys.bk.s.pb(path.bk.s);
                path.pop_back();
            } while (dsu.unite(u,w));
            u = dsu.get(u); heap[u] = cyc, seen[u] = -1;
            cys.bk.f = u;
        }
        each(t,path) in[dsu.get(t.s.b)] = {t.s.a,t.s.b};
    } // found path from root to s, done
    while (sz(cys)) { // expand cys to restore sol
        auto c = cys.bk; cys.pop_back();
        pi inEdge = in[c.f];
        each(t,c.s) dsu.rollback();
        each(t,c.s) in[dsu.get(t.b)] = {t.a,t.b};
        in[dsu.get(inEdge.s)] = inEdge;
    }
    vi par(n); F0R(i,n) par[i] = in[i].f;
    // i == r ? in[i].s == -1 : in[i].s == i
    return {res,par};
}
```

Advanced/LCT.h

Description: Link-Cut Tree. Given a function $f(1 \dots N) \rightarrow 1 \dots N$, evaluates $f^b(a)$ for any a, b . sz is for path queries; $sub, vsub$ are for subtree queries. $x \rightarrow access()$ brings x to the top and propagates it; its left subtree will be the path from x to the root and its right subtree will be empty. Then sub will be the number of nodes in the connected component of x and $vsub$ will be the number of nodes under x . Use $makeRoot$ for arbitrary path queries.

Usage: FOR(i,1,N+1)LCT[i]=new snode(i); link(LCT[1],LCT[2],1);

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

e24bf7, 110 lines

```
typedef struct snode* sn;
struct snode {
    sn p, c[2]; // parent, children
    sn extra; // extra cycle node for "The Applicant"
    bool flip = 0; // subtree flipped or not
    int val, sz; // value in node, # nodes in current splay tree
    int sub, vsub = 0; // vsub stores sum of virtual children
    snode(int _val) : val(_val) {
        p = c[0] = c[1] = extra = NULL; calc(); }
    friend int getSz(sn x) { return x?x->sz:0; }
    friend int getSub(sn x) { return x?x->sub:0; }
    void prop() { // lazy prop
        if (!flip) return;
        swap(c[0],c[1]); flip = 0;
        F0R(i,2) if (c[i]) c[i]->flip ^= 1;
    }
    void calc() { // recalc vals
        F0R(i,2) if (c[i]) c[i]->prop();
        sz = 1+getS(c[0])+getS(c[1]);
        sub = 1+getSub(c[0])+getSub(c[1])+vsub;
    }
    int dir() {
        if (!p) return -2;
        F0R(i,2) if (p->c[i] == this) return i;
        return -1; // p is path-parent pointer
    } // -> not in current splay tree
    // test if root of 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();
    }
```

```
}
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(); calc();
}
sn fbo(int b) { // find by order
    prop(); int z = getS(c[0]); // of splay tree
    if (b == z) { splay(); return this; }
    return b < z ? c[0]->fbo(b) : c[1] -> fbo(b-z-1);
}
void access() { // bring this to top of tree, propagate
    for (sn v = this, pre = NULL; v; v = v->p) {
        v->splay(); // now switch virtual children
        if (pre) v->vsub -= pre->sub;
        if (v->c[1]) v->vsub += v->c[1]->sub;
        v->c[1] = pre; v->calc(); pre = v;
    }
    splay(); assert(!c[1]); // right subtree is empty
}
void makeRoot() {
    access(); flip ^= 1; access(); assert(!c[0] && !c[1]); }
friend sn lca(sn x, sn y) {
    if (x == y) return x;
    x->access(), y->access(); if (!x->p) return NULL;
    x->splay(); return x->p?x: // y was below x in latter case
} // access at y did not affect x -> not connected
friend bool connected(sn x, sn y) { return lca(x,y); }
// # nodes above
int distRoot() { access(); return getS(c[0]); }
sn getRoot() { // get root of LCT component
    access(); sn a = this;
    while (a->c[0]) a = a->c[0], a->prop();
    a->access(); return a;
}
sn getPar(int b) { // get b-th parent on path to root
    access(); b = getS(c[0])-b; assert(b >= 0);
    return fbo(b);
} // can also get min, max on path to root, etc
void set(int v) { access(); val = v; calc(); }
friend void link(sn x, sn y, bool force = 0) {
    assert(!connected(x,y));
    if (force) y->makeRoot(); // make x par of y
    else { y->access(); assert(!y->c[0]); }
    x->access(); setLink(y,x,0); y->calc();
}
friend void cut(sn y) { // cut y from its parent
    y->access(); assert(y->c[0]);
    y->c[0]->p = NULL; y->c[0] = NULL; y->calc(); }
friend void cut(sn x, sn y) { // if x, y adj in tree
    x->makeRoot(); y->access();
    assert(y->c[0] == x && !x->c[0] && !x->c[1]); cut(y); }
};
sn LCT[MX];
```

```
void setNex(sn a, sn b) { // set f[a] = b
    if (connected(a,b)) a->extra = b;
    else link(b,a); }
void delNex(sn a) { // set f[a] = NULL
    auto t = a->getRoot();
    if (t == a) { t->extra = NULL; return; }
    cut(a); assert(t->extra);
    if (!connected(t,t->extra))
        link(t->extra,t), t->extra = NULL;
}
sn getPar(sn a, int b) { // get f^b[a]
    int d = a->distRoot(); if (b <= d) return a->getPar(b);
    b -= d+1; auto r = a->getRoot()->extra; assert(r);
    d = r->distRoot()+1; return r->getPar(b&d);
}
```

EdgeColoring.h

Description: Given a simple, undirected graph with max degree D , computes a $(D + 1)$ -coloring of the edges such that no neighboring edges share a color. (D -coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.)

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

e210e2, 31 lines


```
vi edgeColoring(int N, vector<pii> eds) {
    vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
    for (pii e : eds) ++cc[e.first], ++cc[e.second];
    int u, v, ncols = *max_element(all(cc)) + 1;
    vector<vi> adj(N, vi(ncols, -1));
    for (pii e : eds) {
        tie(u, v) = e;
        fan[0] = v;
        loc.assign(ncols, 0);
        int at = u, end = u, d, c = free[u], ind = 0, i = 0;
        while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
            loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
        cc[loc[d]] = c;
        for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd])
            swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
        while (adj[fan[i]][d] != -1) {
            int left = fan[i], right = fan[++i], e = cc[i];
            adj[u][e] = left;
            adj[left][e] = u;
            adj[right][e] = -1;
            free[right] = e;
        }
        adj[u][d] = fan[i];
        adj[fan[i]][d] = u;
        for (int y : {fan[0], u, end})
            for (int& z = free[y] = 0; adj[y][z] != -1; z++);
    }
    rep(i, 0, sz(eds))
        for (tie(u, v) = eds[i]; adj[u][ret[i]] != v; ) ++ret[i];
    return ret;
}
```

Geometry (8)

8.1 Geometric primitives

Point.h

Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

47ec0a, 28 lines

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
    typedef Point P;
    T x, y;
    explicit Point(T x=0, T y=0) : x(x), y(y) {}
    bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
    bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
    P operator+(P p) const { return P(x+p.x, y+p.y); }
    P operator-(P p) const { return P(x-p.x, y-p.y); }
    P operator*(T d) const { return P(x*d, y*d); }
    P operator/(T d) const { return P(x/d, y/d); }
    T dot(P p) const { return x*p.x + y*p.y; }
    T cross(P p) const { return x*p.y - y*p.x; }
    T cross(P a, P b) const { return (a-*this).cross(b-*this); }
    T dist2() const { return x*x + y*y; }
    double dist() const { return sqrt((double)dist2()); }
    // angle to x-axis in interval [-pi, pi]
    double angle() const { return atan2(y, x); }
    P unit() const { return *this/dist(); } // makes dist()==1
    P perp() const { return P(-y, x); } // rotates +90 degrees
    P normal() const { return perp().unit(); }
    // returns point rotated 'a' radians ccw around the origin
    P rotate(double a) const {
        return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
    friend ostream& operator<<(ostream& os, P p) {
        return os << "(" << p.x << ", " << p.y << ")"; }
};
```

lineDistance.h

Description:

Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.

"Point.h"

f6bf6b, 4 lines

```
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
    return (double) (b-a).cross(p-a)/(b-a).dist();
}
```

SegmentDistance.h

Description:

Returns the shortest distance between point p and the line segment from point s to e.

Usage: Point<double> a, b(2,2), p(1,1);

bool onSegment = segDist(a,b,p) < 1e-10;

"Point.h"

5c88f4, 6 lines

```
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
    if (s==e) return (p-s).dist();
    auto d = (e-s).dist2(), t = min(d,max(.0, (p-s).dot(e-s)));
    return ((p-s)*d-(e-s)*t).dist()/d;
}
```

SegmentIntersection.h

Description:

If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

Usage: vector<P> inter = segInter(s1,e1,s2,e2);

if (sz(inter)==1)

cout << "segments intersect at " << inter[0] << endl;

"Point.h", "OnSegment.h"

9d57f2, 13 lines

```
template<class P> vector<P> segInter(P a, P b, P c, P d) {
    auto oa = c.cross(d, a), ob = c.cross(d, b),
        oc = a.cross(b, c), od = a.cross(b, d);
    // Checks if intersection is single non-endpoint point.
    if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
        return {(a * ob - b * oa) / (ob - oa)};
    set<P> s;
    if (onSegment(c, d, a)) s.insert(a);
    if (onSegment(c, d, b)) s.insert(b);
    if (onSegment(a, b, c)) s.insert(c);
    if (onSegment(a, b, d)) s.insert(d);
    return {all(s)};
}
```

lineIntersection.h

Description:

If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {l, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.

Usage: auto res = lineInter(s1,e1,s2,e2);

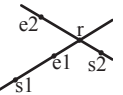
if (res.first == 1)

cout << "intersection point at " << res.second << endl;

"Point.h"

a01f81, 8 lines

```
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
    auto d = (e1 - s1).cross(e2 - s2);
    if (d == 0) // if parallel
        return {(s1.cross(e1, s2) == 0), P(0, 0)};
    auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
    return {1, (s1 * p + e1 * q) / d};
}
```



sideOf.h

Description: Returns where *p* is as seen from *s* towards *e*. $1/0/-1 \Leftrightarrow$ left/on line/right. If the optional argument *eps* is given 0 is returned if *p* is within distance *eps* from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.

Usage: bool left = sideOf(p1,p2,q)==1;

"Point.h"

3af81c, 9 lines

```
template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }
```

template<class P>

```
int sideOf(const P& s, const P& e, const P& p, double eps) {
    auto a = (e-s).cross(p-s);
    double l = (e-s).dist()*eps;
    return (a > l) - (a < -l);
}
```

OnSegment.h

Description: Returns true iff p lies on the line segment from s to e. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

"Point.h"

c597e8, 3 lines

```
template<class P> bool onSegment(P s, P e, P p) {
    return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

linearTrans.h

Description:

Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.

"Point.h"

03a306, 6 lines

```
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
    P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
    return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

LineProjRefl.h

Description: Projects point p onto line ab. Set refl=true to get reflection of point p across line ab insted. The wrong point will be returned if P is an integer point and the desired point doesn't have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow.

"Point.h"

b5562d, 5 lines

```
template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
    P v = b - a;
    return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
}
```

Angle.h

Description: A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.

Usage: vector<Angle> v = {w[0], w[0].t360() ...}; // sorted
int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }
// sweeps j such that (j-i) represents the number of positively oriented triangles with vertices at 0 and i

0f0602, 35 lines

```
struct Angle {
    int x, y;
    int t;
    Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
    Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
    int half() const {
        assert(x || y);
        return y < 0 || (y == 0 && x < 0);
    }
    Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
    Angle t180() const { return {-x, -y, t + half()}; }
    Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
    // add a.dist2() and b.dist2() to also compare distances
    return make_tuple(a.t, a.half(), a.y * (1ll)b.x <
        make_tuple(b.t, b.half(), a.x * (1ll)b.y);
}
```

```
// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
    if (b < a) swap(a, b);
    return (b < a.t180() ?
        make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
    Angle r(a.x + b.x, a.y + b.y, a.t);
    if (a.t180() < r) r.t--;
    return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
    int tu = b.t - a.t; a.t = b.t;
    return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

8.2 Circles

CircleIsect.h

Description: Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

```
"Point.h" 84d6d3, 11 lines

typedef Point<double> P;
bool circleInter(P a, P b, double r1, double r2, pair<P, P> out) {
    if (a == b) { assert(r1 != r2); return false; }
    P vec = b - a;
    double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
        p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
    if (sum*sum < d2 || dif*dif > d2) return false;
    P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
    *out = {mid + per, mid - per};
    return true;
}
```

CircleTangents.h

Description: Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

```
"Point.h" b0153d, 13 lines

template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
    P d = c2 - c1;
    double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
    if (d2 == 0 || h2 < 0) return {};
    vector<pair<P, P>> out;
    for (double sign : {-1, 1}) {
        P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
        out.push_back({c1 + v * r1, c2 + v * r2});
    }
    if (h2 == 0) out.pop_back();
    return out;
}
```

CircleLine.h

Description: Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points. P is intended to be Point<double>.

```
"Point.h" e0cfba, 9 lines

template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
    P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
    double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
    if (h2 < 0) return {};
    if (h2 == 0) return {p};
    P h = ab.unit() * sqrt(h2);
    return {p - h, p + h};
}
```

CPIsect.h

Description: Returns the area of the intersection of a circle with a ccw polygon.

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

```
"../content/geometry/Point.h" aleec63, 19 lines

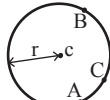
typedef Point<double> P;
```

```
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
    auto tri = [&](P p, P q) {
        auto r2 = r * r / 2;
        P d = q - p;
        auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
        auto det = a * a - b;
        if (det <= 0) return arg(p, q) * r2;
        auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
        if (t < 0 || 1 <= s) return arg(p, q) * r2;
        P u = p + d * s, v = p + d * t;
        return arg(p, u) * r2 + u.cross(v)/2 + arg(v, q) * r2;
    };
    auto sum = 0.0;
    rep(i, 0, sz(ps))
        sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
    return sum;
}
```

circumcircle.h

Description:

The circumcircle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.



```
"Point.h" 1caa3a, 9 lines

typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
    return (B-A).dist()*(C-B).dist()*(A-C).dist() /
        abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
    P b = C-A, c = B-A;
    return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

MECirc.h

Description: Computes the minimum circle that encloses a set of points.

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

```
"circumcircle.h" 09dd0a, 17 lines

pair<P, double> mec(vector<P> ps) {
    shuffle(all(ps), mt19937(time(0)));
    P o = ps[0];
    double r = 0, EPS = 1 + 1e-8;
    rep(i, 0, sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
        o = ps[i], r = 0;
        rep(j, 0, i) if ((o - ps[j]).dist() > r * EPS) {
            o = (ps[i] + ps[j]) / 2;
            r = (o - ps[i]).dist();
            rep(k, 0, j) if ((o - ps[k]).dist() > r * EPS) {
                o = ccCenter(ps[i], ps[j], ps[k]);
                r = (o - ps[i]).dist();
            }
        }
    }
    return {o, r};
}
```

8.3 Polygons

InsidePolygon.h

Description: Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.

Usage: vector<P> v = {P{4,4}, P{1,2}, P{2,1}};

bool in = inPolygon(v, P{3, 3}, false);

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

```
"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 11 lines

template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
    int cnt = 0, n = sz(p);
    rep(i, 0, n) {
        P q = p[(i + 1) % n];
        if (onSegment(p[i], q, a)) return !strict;
        //or: if (segDist(p[i], q, a) <= eps) return !strict;
        cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
    }
    return cnt;
```

PolygonArea.h

Description: Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

```
"Point.h" f12300, 6 lines

template<class T>
T polygonArea2(vector<Point<T>>& v) {
    T a = v.back().cross(v[0]);
    rep(i, 0, sz(v)-1) a += v[i].cross(v[i+1]);
    return a;
}
```

PolygonCenter.h

Description: Returns the center of mass for a polygon.

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

```
"Point.h" 9706dc, 9 lines

typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
    P res(0, 0); double A = 0;
    for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
        res = res + (v[i] + v[j]) * v[j].cross(v[i]);
        A += v[j].cross(v[i]);
    }
    return res / A / 3;
}
```

PolygonCut.h

Description:

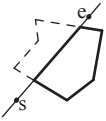
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.

Usage: vector<P> p = ...;

p = polygonCut(p, P(0,0), P(1,0));

```
"Point.h", "LineIntersection.h" f2b7d4, 13 lines

typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
    vector<P> res;
    rep(i, 0, sz(poly)) {
        P cur = poly[i], prev = i ? poly[i-1] : poly.back();
        bool side = s.cross(e, cur) < 0;
        if (side != (s.cross(e, prev) < 0))
            res.push_back(lineInter(s, e, cur, prev).second);
        if (side)
            res.push_back(cur);
    }
    return res;
}
```



PolygonUnion.h

Description: Calculates the area of the union of n polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)

Time: $\mathcal{O}(N^2)$, where N is the total number of points

```
"Point.h", "sideOf.h" 3931c6, 33 lines

typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
    double ret = 0;
    rep(i, 0, sz(poly)) rep(j, 0, sz(poly[i])) {
        P A = poly[i][v], B = poly[i][v + 1] % sz(poly[i]);
        vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
        rep(j, 0, sz(poly)) if (i != j) {
            rep(u, 0, sz(poly[j])) {
                P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
                int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
                if (sc != sd) {
                    double sa = C.cross(D, A), sb = C.cross(D, B);
                    if (min(sc, sd) < 0)
                        segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
                } else if (!sc && !sd && j<i && sgn((B-A).dot(D-C))>0) {
                    segs.emplace_back(rat(C - A, B - A), 1);
                    segs.emplace_back(rat(D - A, B - A), -1);
                }
            }
        }
        sort(all(segs));
        for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
        double sum = 0;
```

```

int cnt = segs[0].second;
rep(j,1,sz(segs)) {
    if (!cnt) sum += segs[j].first - segs[j - 1].first;
    cnt += segs[j].second;
}
ret += A.cross(B) * sum;
}
return ret / 2;
}

```

ConvexHull.h

Description:

Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.

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

"Point.h"	310954, 13 lines
-----------	------------------

```

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

```

HullDiameter.h

Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).

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

"Point.h"	c571b8, 12 lines
-----------	------------------

```

typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
    int n = sz(S), j = n < 2 ? 0 : 1;
    pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
    rep(i,0,j)
        rep(i,0,j)
            for (; j = (j + 1) % n) {
                res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
                if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
                    break;
            }
    return res.second;
}

```

PointInsideHull.h

Description: Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.

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

"Point.h", "sideOf.h", "OnSegment.h"	71446b, 14 lines
--------------------------------------	------------------

```

typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict = true) {
    int a = 1, b = sz(l) - 1, r = !strict;
    if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
    if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
    if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p) <= -r)
        return false;
    while (abs(a - b) > 1) {
        int c = (a + b) / 2;
        (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
    }
    return sgn(l[a].cross(l[b], p)) < r;
}

```

LineHullIntersection.h

Description: Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon: \bullet $(-1, -1)$ if no collision, \bullet $(i, -1)$ if touching the corner i , \bullet (i, i) if along side $(i, i + 1)$, \bullet (i, j) if crossing sides $(i, i + 1)$ and $(j, j + 1)$. In the last case, if a corner i is crossed, this is treated as happening on side $(i, i + 1)$. The points are returned in the same order as the line hits the polygon. extrVertex returns the point of a hull with the max projection onto a line.

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

"Point.h"	7cf45b, 39 lines
-----------	------------------

```

#define cmp(i, j) sgn(dir.perp().cross(poly[i]%n]-poly[j]%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
    int n = sz(poly), lo = 0, hi = n;
    if (extr(0)) return 0;
    while (lo + 1 < hi) {
        int m = (lo + hi) / 2;
        if (extr(m)) return m;
        int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
        (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
    }
    return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
    int endA = extrVertex(poly, (a - b).perp());
    int endB = extrVertex(poly, (b - a).perp());
    if (cmpL(endA) < 0 || cmpL(endB) > 0)
        return {-1, -1};
    array<int, 2> res;
    rep(i,0,2) {
        int lo = endB, hi = endA, n = sz(poly);
        while ((lo + 1) % n != hi) {
            int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
            (cmpL(m) == cmpL(endB) ? lo : hi) = m;
        }
        res[i] = (lo + 1 + cmpL(hi)) % n;
        swap(endA, endB);
    }
    if (res[0] == res[1]) return {res[0], -1};
    if (!cmpL(res[0]) && !cmpL(res[1]))
        switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
            case 0: return {res[0], res[0]};
            case 2: return {res[1], res[1]};
        }
    return res;
}

```

HalfPlaneIsect.h

Description: Returns vertices of half-plane intersection. A half-plane is the area to the left of a ray, which is defined by a point p and a direction dp. Area of intersection should be sufficiently precise when all inputs are integers with magnitude $\leq 10^9$. Intersection must be bounded. Probably works with floating point too (but EPS might need to be adjusted?).

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

"AngleCmp.h"	18f712, 52 lines
--------------	------------------

```

struct Ray {
    P p, dp; // origin, direction
    P isect(const Ray& L) const {
        return p+dp*(cross(L.dp,L.p-p)/cross(L.dp,dp)); }
    bool operator<(const Ray& L) const {
        return angleCmp(dp,L.dp); }
};

```

```

vP halfPlaneIsect(V<Ray> rays, bool add_bounds = false) {
    if (add_bounds) { // bound input by rectangle [0,DX] x [0,DY]
        int DX = 1e9, DY = 1e9;
        rays.pb({P{0,0},P{1,0}});
        rays.pb({P{DX,0},P{0,1}});
        rays.pb({P{DX,DY},P{-1,0}});
        rays.pb({P{0,DY},P{0,-1}});
    }
    sor(rays); // sort rays by angle
    { // remove parallel rays
        V<Ray> nrays;
        each(t, rays) {
            if (!sz(nrays) || cross(nrays.bk.dp, t.dp) > EPS) { nrays.pb(t);
                continue; }
            // last two rays are parallel, keep only one
            if (cross(t.dp, t.p-nrays.bk.p) > 0) nrays.bk = t;
        }
        swap(rays, nrays);
    }
    auto bad = [&](const Ray& a, const Ray& b, const Ray& c) {
        P p1 = a.isect(b), p2 = b.isect(c);
        if (dot(p2-p1,b.dp) <= EPS) {

```

```

        if (cross(a.dp,c.dp) <= 0) return 2; // isect(a,b,c) = empty
        return 1; // isect(a,c) == isect(a,b,c)
    }
    return 0; // all three rays matter
};
#define reduce(t) \
while (sz(poly) > 1) { \
    int b = bad(poly.at(sz(poly)-2),poly.bk,t); \
    if (b == 2) return {}; \
    if (b == 1) poly.pop_back(); \
    else break; \
}
deque<Ray> poly;
each(t, rays) { reduce(t); poly.pb(t); }
for(;;poly.pop_front()) {
    reduce(poly[0]);
    if (!bad(poly.bk,poly[0],poly[1])) break;
}
assert(sz(poly) >= 3); // expect nonzero area
vP poly_points; F0R(i,sz(poly))
    poly_points.pb(poly[i].isect(poly[(i+1)%sz(poly)]));
return poly_points;
}

```

HullTangents.h

Description: Given convex polygon with no three points collinear and a point strictly outside of it, computes the lower and upper tangents.

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

"../Primitives/Point.h"	85b807, 36 lines
-------------------------	------------------

```

bool lower;
bool better(P a, P b, P c) {
    T z = cross(a,b,c);
    return lower ? z < 0 : z > 0; }
int tangent(const vP& a, P b) {
    if (sz(a) == 1) return 0;
    int lo, hi;
    if (better(b,a[0],a[1])) {
        lo = 0, hi = sz(a)-1;
        while (lo < hi) {
            int mid = (lo+hi+1)/2;
            if (better(b,a[0],a[mid])) lo = mid;
            else hi = mid-1;
        }
        lo = 0;
    } else {
        lo = 1, hi = sz(a);
        while (lo < hi) {
            int mid = (lo+hi)/2;
            if (!better(b,a[0],a[mid])) lo = mid+1;
            else hi = mid;
        }
        hi = sz(a);
    }
    while (lo < hi) {
        int mid = (lo+hi)/2;
        if (better(b,a[mid],a[(mid+1)%sz(a)])) lo = mid+1;
        else hi = mid;
    }
    return lo%sz(a);
}
pi tangents(const vP& a, P b) {
    lower = 1; int x = tangent(a,b);
    lower = 0; int y = tangent(a,b);
    return {x,y};
}

```

8.4 Misc. Point Set Problems

ClosestPair.h

Description: Finds the closest pair of points.

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

"Point.h"	ac41a6, 17 lines
-----------	------------------

```

typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
    assert(sz(v) > 1);
    set<P> S;
    sort(all(v), [](P a, P b) { return a.y < b.y; });
    pair<ll, pair<P, P>> ret(LLONG_MAX, {P(), P()});
    int j = 0;
    for (P p : v) {

```

```
P d(1 + (1l)sqrt(ret.first()), 0);
while (v[j].y <= p.y - d.x) S.erase(v[j++]);
auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
for (; lo != hi; ++lo)
    ret = min(ret, {( *lo - p).dist2(), { *lo, p}});
S.insert(p);
}
return ret.second;
```

ManhattanMST.h

Description: Given N points, returns up to 4*N edges, which are guaranteed to contain a minimum spanning tree for the graph with edge weights $w(p, q) = -p.x - q.x - p.y - q.y$. Edges are in the form (distance, src, dst). Use a standard MST algorithm on the result to find the final MST.

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

```
"Point.h" df6f59, 23 lines

typedef Point<int> P;
vector<array<int, 3>> manhattanMST(vector<P> ps) {
    vi id(sz(ps));
    iota(all(id), 0);
    vector<array<int, 3>> edges;
    rep(k,0,4) {
        sort(all(id), [&](int i, int j) {
            return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
        map<int, int> sweep;
        for (int i : id) {
            for (auto it = sweep.lower_bound(-ps[i].y);
                 it != sweep.end(); sweep.erase(it++)) {
                int j = it->second;
                P d = ps[i] - ps[j];
                if (d.y > d.x) break;
                edges.push_back({d.y + d.x, i, j});
            }
            sweep[-ps[i].y] = i;
        }
        for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p.y);
    }
    return edges;
}
```

kdTree.h

Description: KD-tree (2d, can be extended to 3d)

```
"Point.h" bac5b0, 63 lines

typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();
```

```
bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }
```

```
struct Node {
    P pt; // if this is a leaf, the single point in it
    T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
    Node *first = 0, *second = 0;

    T distance(const P& p) { // min squared distance to a point
        T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
        T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
        return (P(x,y) - p).dist2();
    }
}
```

```
Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
        x0 = min(x0, p.x); x1 = max(x1, p.x);
        y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
        // split on x if width >= height (not ideal...)
        sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
        // divide by taking half the array for each child (not
        // best performance with many duplicates in the middle)
        int half = sz(vp)/2;
        first = new Node({vp.begin(), vp.begin() + half});
        second = new Node({vp.begin() + half, vp.end()});
    }
}
```

```
};

struct KDTree {
```

```
Node* root;
KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}

pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
        // uncomment if we should not find the point itself:
        // if (p == node->pt) return {INF, P()};
        return make_pair((p - node->pt).dist2(), node->pt);
    }
}
```

```
Node *f = node->first, *s = node->second;
T bfirst = f->distance(p), bsec = s->distance(p);
if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);
```

```
// search closest side first, other side if needed
auto best = search(f, p);
if (bsec < best.first)
    best = min(best, search(s, p));
return best;
```

```
// find nearest point to a point, and its squared distance
// (requires an arbitrary operator< for Point)
pair<T, P> nearest(const P& p) {
    return search(root, p);
}
};
```

FastDelaunay.h

Description: Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order $\{t[0][0], t[0][1], t[0][2], t[1][0], \dots\}$, all counter-clockwise.

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

```
"Point.h" eefdf5, 88 lines

typedef Point<ll> P;
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 {
    Q rot, o; P p = arb; bool mark;
    P& F() { return r()->p; }
    Q& r() { return rot->rot; }
    Q prev() { return rot->o->rot; }
    Q next() { return r()->prev(); }
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
    lll p2 = p.dist2(), A = a.dist2()-p2,
        B = b.dist2()-p2, C = c.dist2()-p2;
    return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}

Q makeEdge(P orig, P dest) {
    Q r = H ? H : new Quad(new Quad(new Quad{0}));
    H = r->o; r->r()->r() = r;
    rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
    r->p = orig; r->F() = dest;
    return r;
}
```

```
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 = s[0].cross(s[1], s[2]);
        Q c = side ? connect(b, a) : 0;
        return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
    }
}
```

```
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 = s[0].cross(s[1], s[2]);
        Q c = side ? connect(b, a) : 0;
        return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
    }
}
```

```
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 = s[0].cross(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) (e->F().cross(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 ((B->p.cross(H(A)) < 0 && (A = A->next()) ||
        (A->p.cross(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->o = H; H = e; 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<P> triangulate(vector<P> pts) {
    sort(all(pts)); assert(unique(all(pts)) == pts.end());
    if (sz(pts) < 2) return {};
    Q e = rec(pts).first;
    vector<Q> q = {e};
    int qi = 0;
    while (e->o->F().cross(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;
    return pts;
}
```

8.5 3D

PolyhedronVolume.h

Description: Magic formula for the volume of a polyhedron. Faces should point outwards.

```
template<class V, class L>
double signedPolyVolume(const V& p, const L& trilst) {
    double v = 0;
    for (auto i : trilst) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
    return v / 6;
}
```

Point3D.h

Description: Class to handle points in 3D space. T can be e.g. double or long long.

```
template<class T> struct Point3D {
    typedef Point3D P;
    typedef const P& R;
    T x, y, z;
    explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
    bool operator<(R p) const {
        return tie(x, y, z) < tie(p.x, p.y, p.z); }
    bool operator==(R p) const {
        return tie(x, y, z) == tie(p.x, p.y, p.z); }
    P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
    P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
    P operator*(T d) const { return P(x*d, y*d, z*d); }
    P operator/(T d) const { return P(x/d, y/d, z/d); }
    T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
    P cross(R p) const {
        return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
    }
    T dist2() const { return x*x + y*y + z*z; }
    double dist() const { return sqrt((double)dist2()); }
    //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
    double phi() const { return atan2(y, x); }
```

```
//zenith angle (latitude) to the z-axis in interval [0, pi]
double theta() const { return atan2(sqrt(x*x+y*y),z); }
P unit() const { return *this/(T)dist(); } //makes dist()==1
//returns unit vector normal to *this and p
P normal(P p) const { return cross(p).unit(); }
//returns point rotated 'angle' radians ccw around axis
P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.unit();
    return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
}
};
```

3dHull.h

Description: Computes all faces of the 3-dimension hull of a point set. *No four points must be coplanar*, or else random results will be returned. All faces will point outwards.

Time: $\mathcal{O}\left(n^2\right)$

"Point3D.h"	5b45fc, 49 lines
-------------	------------------

```
typedef Point3D<double> P3;
```

```
struct PR {
    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 vector<P3>& A) {
    assert(sz(A) >= 4);
    vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
    #define E(x,y) E[f.x][f.y]
    vector<F> FS;
    auto mf = [&](int i, int j, int k, int l) {
        P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
        if (q.dot(A[l]) > q.dot(A[i]))
            q = q * -1;
        F f{q, i, j, k};
        E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
        FS.push_back(f);
    };
    rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
        mf(i, j, k, 6 - i - j - k);

    rep(i,4,sz(A)) {
        rep(j,0,sz(FS)) {
            F f = FS[j];
            if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
                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();
            }
        }
        int nw = sz(FS);
        rep(j,0,nw) {
            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);
        }
        for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
            A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
        return FS;
    };
};
```

sphericalDistance.h

Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_2) from z axis (0 = north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. dx*radius is then the difference between the two points in the x direction and d*radius is the total distance between the points.

double sphericalDistance(double f1, double t1, double f2, double t2, double radius) {	611f07, 8 lines
---------------------------------------------------------------------------------------	-----------------

```
double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
double dz = cos(t2) - cos(t1);
double d = sqrt(dx*dx + dy*dy + dz*dz);
return radius*2*asin(d/2);
}
```

Strings (9)

9.1 Light

Light/KMP.h

Description: f[i] is 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(str s) { int N = sz(s); vi f(N+1); f[0] = -1; FOR(i,1,N+1) { for (f[i]=f[i-1];f[i]!=-1&&s[f[i]]!=s[i-1];)f[i]=f[f[i]]; ++f[i]; } return f; } vi getOc(str a, str 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; }	4538e4, 13 lines
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------

Light/Z (14.3).h

Description: f[i] is the max len such that s.substr(0,len) == s.substr(i,len)

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

vi z(str s) { int N = sz(s), L = 1, R = 0; s += '#'; vi ans(N); ans[0] = N; 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(str a, str b) { // find prefixes of a in b vi t = z(a+b); t = vi(sz(a)+all(t)); each(u,t) ckmin(u,sz(a)); return t; }	566170, 15 lines
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------

Light/Manacher.h

Description: length of largest palindrome centered at each character of string and between every consecutive pair

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

vi manacher(str _S) { str S = "0"; each(c,_S) S += c, S += "#"; S.bk = '&'; vi ans(sz(S)-1); int lo = 0, hi = 0; FOR(i,1,sz(S)-1) { if (i != 1) ans[i] = min(hi-i,ans[hi-i+lo]); while (S[i-ans[i]-1] == S[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%2 == ans[i]%2) ++ans[i]; return ans; }	fcc3f7, 13 lines
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------

Light/MinRotation.h

Description: minimum cyclic shift

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

int minRotation(str 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; } }	57b7f2, 10 lines
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------

```
// 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;
}
```

Light/LyndonFactor.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*₁*w*₂...*w*_{*k*} where all strings *w*_{*i*} are simple and *w*₁ ≥ *w*₂ ≥ ... ≥ *w*_{*k*}. Min rotation gets min index *i* such that cyclic shift of *s* starting at *i* is minimum.

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

vs duval(str s) { int N = sz(s); vs 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(str s) { int N = sz(s); s += s; vs 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; }	af38ba, 19 lines
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------

Light/HashRange (14.2).h

Description: Polynomial hash for substrings with two bases.

using H = AR<int,2>; // bases not too close to ends H makeH(char c) { return {c,c}; } uniform_int_distribution<int> BDIST(0.1*MOD,0.9*MOD); const H base{BDIST(rng),BDIST(rng)}; H operator+(H l, H r) { FOR(i,2) if ((l[i] += r[i]) >= MOD) l[i] -= MOD; return l; } H operator-(H l, H r) { FOR(i,2) if ((l[i] -= r[i]) < 0) l[i] += MOD; return l; } H operator*(H l, H r) { FOR(i,2) l[i] = (ll)l[i]*r[i]%MOD; return l; } };	fc0b90, 24 lines
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------

```
V<H> pows{{1,1}};
struct HashRange {
    str S; V<H> cum{};
    void add(char c) { S += c; cum.pb(base*cum.bk+makeH(c)); }
    void add(str s) { each(c,s) add(c); }
    void extend(int len) { while (sz(pows) <= len)
        pows.pb(base*pows.bk); }
    H hash(int l, int r) { int len = r+1-l; extend(len);
        return cum[r+1]-pows[len]*cum[l]; }
};
```

Light/ReverseBW (14.4).h

Description: Used only once. 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: $\mathcal{O}(N \log N)$

str reverseBW(str t) { vi nex(sz(t)); iota(all(nex),0); stable_sort(all(nex),[&t](int a,int b){return t[a]<t[b]}); str ret; for (int i = nex[0]; i;) ret += t[i = nex[i]]; return ret; }	e400d8, 7 lines
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------

Light/AhoCorasickFixed.h

Description: Aho-Corasick for fixed alphabet. For each prefix, stores link to max length suffix which is also a prefix.

Time: $\mathcal{O}(N \sum)$	96dfcc, 27 lines
<pre> template<size_t ASZ> struct ACfixed { struct Node { AR<int, ASZ> to; int link; }; V<Node> d[{}]; int add(str s) { // add word int v = 0; each(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.ft; q.pop(); F0R(c,ASZ) { 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) F0R(c,ASZ) if (!d[v].to[c]) d[v].to[c] = d[d[v].link].to[c]; } } }; </pre>	

Light/SuffixArray (14.4).h

Description: Sort suffixes. First element of sa is sz(S), isa is the inverse of sa, and lcp stores the longest common prefix between every two consecutive elements of sa.

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

"RMQ.h"	27a566, 30 lines
<pre> struct SuffixArray { str S; int N; vi sa, isa, lcp; void init(str _S) { N = sz(S = _S)+1; genSa(); genLcp(); } void genSa() { // sa has size sz(S)+1, starts with sz(S) sa = isa = vi(N); sa[0] = N-1; iota(1+all(sa),0); sort(1+all(sa), [&](int a, int b) { return S[a] < S[b]; }); FOR(i,1,N) { int a = sa[i-1], b = sa[i]; isa[b] = i > 1 && S[a] == S[b] ? isa[a] : i; } for (int len = 1; len < N; len *= 2) { // currently sorted // by first len chars vi s(sa), is(isa), pos(N); iota(all(pos),0); each(t,s) {int T=t-len;if (T>0) sa[pos[isa[T]]++] = T;} FOR(i,1,N) { int a = sa[i-1], b = sa[i]; isa[b] = is[a]==is[b]&&is[a+len]==is[b+len]?isa[a]:i; } } } void genLcp() { // Kasai's Algo lcp = vi(N-1); int h = 0; F0R(b,N-1) { int a = sa[isa[b]-1]; while (a+h < sz(S) && S[a+h] == S[b+h]) ++h; lcp[isa[b]-1] = h; if (h) h--; } R.init(lcp); } RMQ<int> R; int getLCP(int a, int b) { // lcp of suffixes starting at a,b if (a == b) return sz(S)-a; int l = isa[a], r = isa[b]; if (l > r) swap(l,r); return R.query(l,r-1); } }; </pre>	

Light/SuffixArrayLinear.h

Description: Linear-time suffix array.

Usage: sa.is(s, 26) // all entries must be in [0, 26)

Time: $\mathcal{O}(N)$, ~ 100 ms for $N = 5 \cdot 10^5$

```

vi sa_is(const vi& s, int upper) {
    int n = sz(s); if (!n) return {};
    vi sa(n); vb ls(n);
    R0F(i,n-1) ls[i] = s[i] == s[i+1] ? ls[i+1] : s[i] < s[i+1];
    vi sum_l(upper), sum_s(upper);
    F0R(i,n) (ls[i] ? sum_l[s[i]+1] : sum_s[s[i]]++)
    F0R(i,upper) {
        if (i) sum_l[i] += sum_s[i-1];
        sum_s[i] += sum_l[i];
    }
}

```

<pre> } auto induce = [&](const vi& lms) { fill(all(sa),-1); vi buf = sum_s; for (int d: lms) if (d != n) sa[buf[s[d]]++] = d; buf = sum_l; sa[buf[s[n-1]]++] = n-1; F0R(i,n) { int v = sa[i]-1; if (v >= 0 && !ls[v]) sa[buf[s[v]]++] = v; } buf = sum_l; R0F(i,n) { int v = sa[i]-1; if (v >= 0 && ls[v]) sa[--buf[s[v]+1]] = v; } }; vi lms_map(n+1,-1), lms; int m = 0; FOR(i,1,n) if (!ls[i-1] && ls[i]) lms_map[i]=m++, lms.pb(i); induce(lms); // sorts LMS prefixes vi sorted_lms;each(v,sa)if (lms_map[v]!==-1)sorted_lms.pb(v); vi rec_s(m); int rec_upper = 0; // smaller subproblem FOR(i,1,m) { // compare two lms substrings in sorted order int l = sorted_lms[i-1], r = sorted_lms[i]; int end_l = lms_map[l]+1 < m ? lms[lms_map[l]+1] : n; int end_r = lms_map[r]+1 < m ? lms[lms_map[r]+1] : n; bool same = 0; // whether lms substrings are same if (end_l-1 == end_r-r) { for (;l < end_l && s[l] == s[r]; ++l,++r); if (l != n && s[l] == s[r]) same = 1; } rec_s[lms_map[sorted_lms[i]]] = (rec_upper += !same); } vi rec_sa = sa_is(rec_s,rec_upper+1); F0R(i,m) sorted_lms[i] = lms[rec_sa[i]]; induce(sorted_lms); // sorts LMS suffixes return sa; } </pre>	
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

Light/TandemRepeats.h

Description: Find all (i, p) such that $s.substr(i, p) == s.substr(i+p, p)$.

No two intervals with the same period intersect or touch.

Usage: solve("aaabababa") // {{0, 1, 1}, {2, 5, 2}}

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

"SuffixArray.h"	661326, 13 lines
<pre> V<AR<int,3>> solve(str s) { int N = sz(s); SuffixArray A,B; A.init(s); reverse(all(s)); B.init(s); V<AR<int,3>> runs; for (int p = 1; 2*p <= N; ++p) { // do in $\mathcal{O}(N/p)$ for period p for (int i = 0, lst = -1; i+p <= N; i += p) { int l = i-B.getLCP(N-i-p,N-i), r = i-p+A.getLCP(i,i+p); if (l > r l == lst) continue; runs.pb({lst = l,r,p}); // for each i in [l,r], } // s.substr(i,p) == s.substr(i+p,p) } return runs; } </pre>	

9.2 Heavy

Heavy/PalTree.h

Description: Used infrequently. Palindromic tree computes number of occurrences of each palindrome within string. ans[i][0] stores min even x such that the prefix $s[1..i]$ can be split into exactly x palindromes, ans[i][1] does the same for odd x .

Time: $\mathcal{O}(N \sum)$ for addChar, $\mathcal{O}(N \log N)$ for updAns

```

struct PalTree {
    static const int ASZ = 26;
    struct node {
        AR<int,ASZ> to = AR<int,ASZ>();
        int len, link, oc = 0; // # occurrences of pal
        int slink = 0, diff = 0;
        AR<int,2> seriesAns;
        node(int _len, int _link) : len(_len), link(_link) {}
    };
    str s = ""; V<AR<int,2>> ans = {{0,MOD}};
    V<node> d = {{0,1},{-1,0}}; // dummy pals of len 0,-1
    int last = 1;
    int getLink(int v) {
        while (s[sz(s)-d[v].len-2] != s.bk) v = d[v].link;
    }
};

```

<pre> return v; } void updAns() { // serial path has $\mathcal{O}(\log n)$ vertices ans.pb({MOD,MOD}); for (int v = last; d[v].len > 0; v = d[v].slink) { d[v].seriesAns=ans[sz(s)-1-d[v].slink].len-d[v].diff; if (d[v].diff == d[d[v].link].diff) F0R(i,2) ckmin(d[v].seriesAns[i], d[d[v].link].seriesAns[i]); // start of previous oc of link[v]=start of last oc of v F0R(i,2) ckmin(ans.bk[i],d[v].seriesAns[i^1]+1); } } void addChar(char C) { s += C; int c = C-'a'; last = getLink(last); if (!d[last].to[c]) { d.eb(d[last].len+2,d[getLink(d[last].link)].to[c]); d[last].to[c] = sz(d)-1; auto z = d.bk; z.diff = z.len-d[z.link].len; z.slink = z.diff == d[z.link].diff ? d[z.link].slink : z.link; } // max suf with different dif last = d[last].to[c]; ++d[last].oc; updAns(); } void numOc() { R0F(i,2,sz(d)) d[d[i].link].oc += d[i].oc; } }; </pre>	
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

Heavy/SuffixAutomaton.h

Description: Used infrequently. Constructs minimal deterministic finite automaton (DFA) that recognizes all suffixes of a string. len corresponds to the maximum length of a string in the equivalence class, pos corresponds to the first ending position of such a string, lnk corresponds to the longest suffix that is in a different class. Suffix links correspond to suffix tree of the reversed string!

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

```

struct SuffixAutomaton {
    int N = 1; vi lnk(-1), len(0), pos(-1); // suffix link,
    // max length of state, last pos of first occurrence of state
    V<map<char,int>> nex{1}; V<bool> isClone{0};
    // transitions, cloned -> not terminal state
    V<vi> iLnk; // inverse links
    int add(int p, char c) { // ~p nonzero if p != -1
        auto getNext = [&]() {
            if (p == -1) return 0;
            int q = nex[p][c]; if (len[p]+1 == len[q]) return q;
            int clone = N++; lnk.pb(lnk[q]); lnk[q] = clone;
            len.pb(len[p]+1), nex.pb(nex[q]),
            pos.pb(pos[q]), isClone.pb(1);
            for (; ~p && nex[p][c] == q; p = lnk[p]) nex[p][c]=clone;
            return clone;
        };
        // if (nex[p].count(c)) return getNext();
        // ^ need if adding > 1 string
        int cur = N++; // make new state
        lnk.eb(), len.pb(len[p]+1), nex.eb(),
        pos.pb(pos[p]+1), isClone.pb(0);
        for (; ~p && !nex[p].count(c); p = lnk[p]) nex[p][c] = cur;
        int x = getNext(); lnk[cur] = x; return cur;
    }

    void init(str s) { int p = 0; each(x,s) p = add(p,x); }
    // inverse links
    void genILnk() {iLnk.rsz(N);FOR(v,1,N)iLnk[lnk[v]].pb(v);}
    // APPLICATIONS
    void getAllOccur(vi& oc, int v) {
        if (!isClone[v]) oc.pb(pos[v]); // terminal position
        each(u,iLnk[v]) getAllOccur(oc,u); }
    vi allOccur(str s) { // get all occurrences of s in automaton
        int cur = 0;
        each(x,s) {
            if (!nex[cur].count(x)) return {};
            cur = nex[cur][x]; }
        // convert end pos -> start pos
        vi oc; getAllOccur(oc,cur); each(t,oc) t += 1-sz(s);
        sort(all(oc)); return oc;
    }

    vl distinct;
    ll getDistinct(int x) {
        // # distinct strings starting at state x
        if (distinct[x]) return distinct[x];
    }
};

```

```

    distinct[x]=1;each(y,nex[x]) distinct[x]+=getDistinct(y.s);
    return distinct[x]; }
ll numDistinct() { // # distinct substrings including empty
    distinct.rsز(N); return getDistinct(0); }
ll numDistinct2() { // assert(numDistinct()==numDistinct2());
    ll ans = 1; FOR(i,1,N) ans += len[i]-len[lnk[i]];
    return ans; }
};

```

```

SuffixAutomaton S;
vi sa; str s;
void dfs(int x) {
    if (!S.isClone[x]) sa.pb(sz(s)-1-S.pos[x]);
    V<pair<char,int>> chr;
    each(t,S.iLnk[x]) chr.pb({s[pos[t]-S.len[x]],t});
    sort(all(chr)); each(t,chr) dfs(t.s);
}

int main() {
    re(s); reverse(all(s));
    S.init(s); S.genILnk();
    dfs(0); ps(sa); // generating suffix array for s
}

```

Heavy/SuffixTree.h

Description: Used infrequently. Ukkonen's algorithm for suffix tree. Longest non-unique suffix of s has length $\text{len}[p]+1$ after each call to add terminates. Each iteration of loop within add decreases this quantity by one.

Time: $\mathcal{O}(N \log \Sigma)$ 39751c, 51 lines

```

struct SuffixTree {
    str s; int N = 0;
    vi pos, len, lnk; V<map<char,int>> to;
    int make(int POS, int LEN) { // lnk[x] is meaningful when
        // x!=0 and len[x] != MOD
        pos.pb(POS); len.pb(LEN); lnk.pb(-1); to.eb(); return N++; }
    void add(int& p, int& lef, char c) { // longest
        // non-unique suffix is at node p with lef extra chars
        s += c; ++lef; int lst = 0;
        for (; lef; p? p=lnk[p]: lef--) { // if p != root then lnk[p]
            // must be defined
            while (lef>1 && lef>len[to[p][s[sz(s)-lef]]])
                p = to[p][s[sz(s)-lef]], lef -= len[p];
            // traverse edges of suffix tree while you can
            char e = s[sz(s)-lef]; int& q = to[p][e];
            // next edge of suffix tree
            if (!q) q = make(sz(s)-lef,MOD), lnk[lst] = p, lst = 0;
            // make new edge
            else {
                char t = s[pos[q]+lef-1];
                if (t == c) { lnk[lst] = p; return; } // suffix not unique
                int u = make(pos[q],lef-1);
                // new node for current suffix-1, define its link
                to[u][c] = make(sz(s)-1,MOD); to[u][t] = q;
                // new, old nodes
                pos[q] += lef-1; if (len[q] != MOD) len[q] -= lef-1;
                q = u, lnk[lst] = u, lst = u;
            }
        }
    }
    void init(str _s) {
        make(-1,0); int p = 0, lef = 0;
        each(c,_s) add(p,lef,c);
        add(p,lef,'$'); s.pop_back(); // terminal char
    }
    int maxPre(str x) { // max prefix of x which is substring
        for (int p = 0, ind = 0;;) {
            if (ind == sz(x) || !to[p].count(x[ind])) return ind;
            p = to[p][x[ind]];
            FOR(i,len[p]) {
                if (ind == sz(x) || x[ind] != s[pos[p]+i]) return ind;
                ind++;
            }
        }
    }
    vi sa; // generate suffix array
    void genSa(int x = 0, int Len = 0) {
        if (!sz(to[x])) sa.pb(pos[x]-Len); // found terminal node
        else each(t,to[x]) genSa(t.s,Len+len[x]);
    }
};

```

Various (10)

10.1 Intervals

IntervalContainer.h

Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).

Time: $\mathcal{O}(\log N)$ edce47, 23 lines

```

set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
    if (L == R) return is.end();
    auto it = is.lower_bound({L, R}), before = it;
    while (it != is.end() && it->first <= R) {
        R = max(R, it->second);
        before = it = is.erase(it);
    }
    if (it != is.begin() && (--it)->second >= L) {
        L = min(L, it->first);
        R = max(R, it->second);
        is.erase(it);
    }
    return is.insert(before, {L,R});
}

void removeInterval(set<pii>& is, int L, int R) {
    if (L == R) return;
    auto it = addInterval(is, L, R);
    auto r2 = it->second;
    if (it->first == L) is.erase(it);
    else (int&)it->second = L;
    if (R != r2) is.emplace(R, r2);
}

```

IntervalCover.h

Description: Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add || R.empty(). Returns empty set on failure (or if G is empty).

Time: $\mathcal{O}(N \log N)$ 9e9d8d, 19 lines

```

template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
    vi S(sz(I)), R;
    iota(all(S), 0);
    sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
    T cur = G.first;
    int at = 0;
    while (cur < G.second) { // (A)
        pair<T, int> mx = make_pair(cur, -1);
        while (at < sz(I) && I[S[at]].first <= cur) {
            mx = max(mx, make_pair(I[S[at]].second, S[at]));
            at++;
        }
        if (mx.second == -1) return {};
        cur = mx.first;
        R.push_back(mx.second);
    }
    return R;
}

```

ConstantIntervals.h

Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.

Usage: constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});

Time: $\mathcal{O}\left(k \log \frac{n}{k}\right)$ 753a4c, 19 lines

```

template<class F, class G, class T>
void rec(int from, int to, F& f, G& g, int& i, T& p, T q) {
    if (p == q) return;
    if (from == to) {
        g(i, to, p);
        i = to; p = q;
    } else {
        int mid = (from + to) >> 1;
        rec(from, mid, f, g, i, p, f(mid));
        rec(mid+1, to, f, g, i, p, q);
    }
}

```

```

template<class F, class G>
void constantIntervals(int from, int to, F f, G g) {
    if (to <= from) return;
    int i = from; auto p = f(i), q = f(to-1);
    rec(from, to-1, f, g, i, p, q);
    g(i, to, q);
}

```

10.2 Misc. algorithms

FastKnapsack.h

Description: Given N non-negative integer weights w and a non-negative target t , computes the maximum $S \leq t$ such that S is the sum of some subset of the weights.

Time: $\mathcal{O}(N \max(w_i))$ b20ccc, 16 lines

```

int knapsack(vi w, int t) {
    int a = 0, b = 0, x;
    while (b < sz(w) && a + w[b] <= t) a += w[b++];
    if (b == sz(w)) return a;
    int m = *max_element(all(w));
    vi u, v(2*m, -1);
    v[a+m-t] = b;
    rep(i,b,sz(w)) {
        u = v;
        rep(x,0,m) v[x+w[i]] = max(v[x+w[i]], u[x]);
        for (x = 2*m; --x > m;) rep(j, max(0,u[x]), v[x])
            v[x-w[j]] = max(v[x-w[j]], j);
    }
    for (a = t; v[a+m-t] < 0; a--);
    return a;
}

```

10.3 Dynamic programming

KnuthDP.h

Description: When doing DP on intervals: $a[i][j] = \min_{i < k < j} (a[i][k] + a[k][j]) + f(i, j)$, where the (minimal) optimal k increases with both i and j , one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$. This is known as Knuth DP. Sufficient criteria for this are if $f(b, c) \leq f(a, d)$ and $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$ for all $a \leq b \leq c \leq d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.

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

CircularLCS.h

Description: Used only twice. For strs A, B calculates longest common subsequence of A with all rotations of B

Time: $\mathcal{O}(|A| \cdot |B|)$ db21cf, 26 lines

```

int circular_lcs(str A, str B) {
    B += B;
    int max_lcs = 0;
    V<vb> dif_left(sz(A)+1, vb(sz(B)+1)), dif_up(sz(A)+1, vb(sz(B)+1));
    auto recalc = [&](int x, int y) { assert(x && y);
        int res = (A.at(x-1) == B.at(y-1)) |
            dif_up[x][y-1] | dif_left[x-1][y];
        dif_left[x][y] = res-dif_up[x][y-1];
        dif_up[x][y] = res-dif_left[x-1][y];
    };
    FOR(i,1,sz(A)+1) FOR(j,1,sz(B)+1) recalc(i,j);
    FOR(j,sz(B)/2) {
        // 1. zero out dp[.][j], update dif_left and dif_right
        if (j) for (int x = 1, y = j; x <= sz(A) && y <= sz(B); ) {
            int pre_up = dif_up[x][y];
            if (y == j) dif_up[x][y] = 0;
            else recalc(x,y);
            (pre_up == dif_up[x][y]) ? ++x : ++y;
        }
        // 2. calculate LCS[A[0:sz(A)], B[j:sz(B)/2])
        int cur_lcs = 0;
        FOR(x,1,sz(A)+1) cur_lcs += dif_up[x][j+sz(B)/2];
        ckmax(max_lcs, cur_lcs);
    }
    return max_lcs;
}

```

SMAWK.h

Description: Given negation of totally monotone matrix with entries of type D , find indices of row maxima (their indices increase for every submatrix). If tie, take lesser index. f returns matrix entry at (r, c) in $O(1)$. Use in place of divide & conquer to remove a log factor.

Time: $O(R + C)$, can be reduced to $O(C(1 + \log R/C))$ evaluations of f usage: 25 lines

```
template<class F, class D=ll> vi smawk (F f, vi x, vi y) {
    vi ans(sz(x),-1); // x = rows, y = cols
    #define upd() if (ans[i] == -1 || w > mx) ans[i] = c, mx = w
    if (min(sz(x),sz(y)) <= 0) {
        F0R(i,sz(x)) { int r = x[i]; D mx;
            each(c,y) { D w = f(r,c); upd(); } }
        return ans;
    }
    if (sz(x) < sz(y)) { // reduce subset of cols to consider
        vi Y; each(c,y) {
            for (;sz(Y);Y.pop_back()) { int X = x[sz(Y)-1];
                if (f(X,Y.bk) >= f(X,c)) break; }
            if (sz(Y) < sz(x)) Y.pb(c);
        } y = Y;
    } // recurse on half the rows
    vi X; for (int i = 1; i < sz(x); i += 2) X.pb(x[i]);
    vi ANS = smawk(f,X,y); F0R(i,sz(ANS)) ans[2*i+1] = ANS[i];
    for (int i = 0, k = 0; i < sz(x); i += 2){
        int to = i+1 < sz(ans) ? ans[i+1] : y.bk; D mx;
        for(int r = x[i];++k) {
            int c = y[k]; D w = f(r,c); upd();
            if (c == to) break; }
    }
    return ans;
};
```

10.4 Debugging tricks

- `signal(SIGSEGV, [](int) { _Exit(0); });` converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). `_GLIBCXX_DEBUG` failures generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).
- `feenableexcept(29);` kills the program on NaNs (1), 0-divs (4), infinities (8) and denormals (16).

10.5 Optimization tricks

`__builtin_ia32_ldmxcsr(40896);` disables denormals (which make floats 20x slower near their minimum value).

10.5.1 Bit hacks

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

10.5.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.
- `#pragma GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.
- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

BumpAllocator.h

Description: When you need to dynamically allocate many objects and don't care about freeing them. "new X" otherwise has an overhead of something like 0.05us + 16 bytes per allocation. 745db2, 8 lines

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

SmallPtr.h

Description: A 32-bit pointer that points into BumpAllocator memory. "BumpAllocator.h" 2dd6c9, 10 lines

```
template<class T> struct ptr {
    unsigned ind;
    ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0) {
        assert(ind < sizeof buf);
    }
    T& operator*() const { return *(T*)(buf + ind); }
    T* operator->() const { return &*this; }
    T& operator[](int a) const { return (&*this)[a]; }
    explicit operator bool() const { return ind; }
};
```

BumpAllocatorSTL.h

Description: BumpAllocator for STL containers. **Usage:** `vector<vector<int, small<int>>> ed(N);` bb66d4, 14 lines

```
char buf[450 << 20] alignas(16);
size_t buf_ind = sizeof buf;

template<class T> struct small {
    typedef T value_type;
    small() {}
    template<class U> small(const U&) {}
    T* allocate(size_t n) {
        buf_ind -= n * sizeof(T);
        buf_ind &= 0 - alignof(T);
        return (T*)(buf + buf_ind);
    }
    void deallocate(T*, size_t) {}
};
```

SIMD.h

Description: Cheat sheet of SSE/AVX intrinsics, for doing arithmetic on several numbers at once. Can provide a constant factor improvement of about 4, orthogonal to loop unrolling. Operations follow the pattern `"_mm(256)?name.(si(128|256)|epi(8|16|32|64)|pd|ps)".` Not all are described here; grep for `_mm_` in `/usr/lib/gcc/*/*4.9/include/` for more. If AVX is unsupported, try 128-bit operations, "emmintrin.h" and `#define _SSE_` and `_MMX_` before including it. For aligned memory use `_mm_malloc(size, 32)` or `int buf[N] alignas(32), but prefer loadu/storeu.` 551b82, 43 lines

```
#pragma GCC target ("avx2") // or sse4.1
#include "emmintrin.h"

typedef __m256i mi;
#define L(x) _mm256_loadu_si256((mi*)&(x))

// High-level/specific methods:
// load(u)?_si256, store(u)?_si256, setzero_si256, _mm_malloc
// blendv_(epi8|ps|pd) (z?y:x), movemask_epib (hibits of bytes)
// i32gather_epib32(addr, x, 4): map addr[] over 32-b parts of x
// sad_epu8: sum of absolute differences of u8, outputs 4xi64
// maddubs_epib16: dot product of unsigned i7's, outputs 16xi15
// madd_epib16: dot product of signed i16's, outputs 8xi32
// extractf128_si256(, i) (256->128), cvtsi128_si32 (128->lo32)
// permute2f128_si256(x,x,1) swaps 128-bit lanes
// shuffle_epib32(x, 3*64+2*16+1*4+0) == x for each lane
// shuffle_epib8(x, y) takes a vector instead of an imm

// Methods that work with most data types (append e.g. _epib32):
// set1, blend (i8?x:y), add, adds (sat.), mullo, sub, and/or,
// andnot, abs, min, max, sign(1,x), cmp(gt|eq), unpack(lo|hi)

int sumi32(mi m) { union {int v[8]; mi m;} u; u.m = m;
    int ret = 0; rep(i,0,8) ret += u.v[i]; return ret; }
mi zero() { return _mm256_setzero_si256(); }
```

```
mi one() { return _mm256_set1_epi32(-1); }
bool all_zero(mi m) { return _mm256_testz_si256(m, m); }
bool all_one(mi m) { return _mm256_testc_si256(m, one()); }

ll example_filteredDotProduct(int n, short* a, short* b) {
    int i = 0; ll r = 0;
    mi zero = _mm256_setzero_si256(), acc = zero;
    while (i + 16 <= n) {
        mi va = L(a[i]), vb = L(b[i]); i += 16;
        va = _mm256_and_si256(_mm256_cmpgt_epib16(vb, va), va);
        mi vp = _mm256_madd_epib16(va, vb);
        acc = _mm256_add_epib64(_mm256_unpacklo_epib32(vp, zero),
            _mm256_add_epib64(acc, _mm256_unpackhi_epib32(vp, zero)));
    }
    union {ll v[4]; mi m;} u; u.m = acc; rep(i,0,4) r += u.v[i];
    for (;i<n;++i) if (a[i] < b[i]) r += a[i]*b[i]; // <- equiv
    return r;
}
```


Techniques (A)

techniques.txt	159 lines
Recursion	
Divide and conquer	
Finding interesting points in N log N	
Algorithm analysis	
Master theorem	
Amortized time complexity	
Greedy algorithm	
Scheduling	
Max contiguous subvector sum	
Invariants	
Huffman encoding	
Graph theory	
Dynamic graphs (extra book-keeping)	
Breadth first search	
Depth first search	
* Normal trees / DFS trees	
Dijkstra's algorithm	
MST: Prim's algorithm	
Bellman-Ford	
Konig's theorem and vertex cover	
Min-cost max flow	
Lovasz toggle	
Matrix tree theorem	
Maximal matching, general graphs	
Hopcroft-Karp	
Hall's marriage theorem	
Graphical sequences	
Floyd-Warshall	
Euler cycles	
Flow networks	
* Augmenting paths	
* Edmonds-Karp	
Bipartite matching	
Min. path cover	
Topological sorting	
Strongly connected components	
2-SAT	
Cut vertices, cut-edges and biconnected components	
Edge coloring	
* Trees	
Vertex coloring	
* Bipartite graphs (=> trees)	
* 3^n (special case of set cover)	
Diameter and centroid	
K'th shortest path	
Shortest cycle	
Dynamic programming	
Knapsack	
Coin change	
Longest common subsequence	
Longest increasing subsequence	
Number of paths in a dag	
Shortest path in a dag	
Dynprog over intervals	
Dynprog over subsets	
Dynprog over probabilities	
Dynprog over trees	
3^n set cover	
Divide and conquer	
Knuth optimization	
Convex hull optimizations	
RMQ (sparse table a.k.a 2^k-jumps)	
Bitonic cycle	
Log partitioning (loop over most restricted)	
Combinatorics	
Computation of binomial coefficients	
Pigeon-hole principle	
Inclusion/exclusion	
Catalan number	
Pick's theorem	
Number theory	
Integer parts	
Divisibility	
Euclidean algorithm	
Modular arithmetic	
* Modular multiplication	
* Modular inverses	

* Modular exponentiation by squaring
Chinese remainder theorem
Fermat's little theorem
Euler's theorem
Phi function
Frobenius number
Quadratic reciprocity
Pollard-Rho
Miller-Rabin
Hensel lifting
Vieta root jumping
Game theory
Combinatorial games
Game trees
Mini-max
Nim
Games on graphs
Games on graphs with loops
Grundy numbers
Bipartite games without repetition
General games without repetition
Alpha-beta pruning
Probability theory
Optimization
Binary search
Ternary search
Unimodality and convex functions
Binary search on derivative
Numerical methods
Numeric integration
Newton's method
Root-finding with binary/ternary search
Golden section search
Matrices
Gaussian elimination
Exponentiation by squaring
Sorting
Radix sort
Geometry
Coordinates and vectors
* Cross product
* Scalar product
Convex hull
Polygon cut
Closest pair
Coordinate-compression
Quadtrees
KD-trees
All segment-segment intersection
Sweeping
Discretization (convert to events and sweep)
Angle sweeping
Line sweeping
Discrete second derivatives
Strings
Longest common substring
Palindrome subsequences
Knuth-Morris-Pratt
Tries
Rolling polynomial hashes
Suffix array
Suffix tree
Aho-Corasick
Manacher's algorithm
Letter position lists
Combinatorial search
Meet in the middle
Brute-force with pruning
Best-first (A*)
Bidirectional search
Iterative deepening DFS / A*
Data structures
LCA (2^k-jumps in trees in general)
Pull/push-technique on trees
Heavy-light decomposition
Centroid decomposition
Lazy propagation
Self-balancing trees
Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
Monotone queues / monotone stacks / sliding queues
Sliding queue using 2 stacks

Persistent segment tree