Contest (1)

template.cpp

28 lines

```
#include<bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
     tree_order_statistics_node_update> ordered_set;
typedef tree<pair<int, int>, null_type,less<pair<int, int> >,
     rb_tree_tag,tree_order_statistics_node_update>
     ordered_multiset;

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

signed main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    return 0;
    // -Wl,--stack=2147483648
}

{
  "cmd" : ["g++ -std=c++17 $file_name -o $file_base_name &&
      timeout 8s ./$file_base_name <inputf.in> outputf.in"],
  "selector" : "source.cpp",
  "shell": true,
  "working_dir" : "$file_path"
}
```

Mathematics (2)

2.1 Elimination

gauss.h

Description: Gauss elimination for solving linear system of equations.

Time: $\mathcal{O}(min(n, m) \cdot n \cdot m)$

bdfbab, 41 lines

```
template<typename T>
int gauss (vector < vector<T> > a, vector<T> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i){
            if ((a[i][col].val) > (a[sel][col].val))sel = i;
        }
        if ((a[sel][col]) == 0)continue; // abs(a[sel][col]) <
             eps, in case of doubles
        for (int i=col; i<=m; ++i){
            swap (a[sel][i], a[row][i]);
        }
        where[col] = row;

        for (int i=0; i<n; ++i){
            if (i != row) {
                T c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j){
                    a[i][j] -= a[row][j] * c;
                }
            }
        }
        ++row;
    }
    for (int i=0; i<m; ++i){
        if (where[i] != -1)ans[i] = a[where[i]][m] / a[where[i
             ]][i];
    }
    for (int i=0; i<n; ++i) {
        T sum = 0;
        for (int j=0; j<m; ++j)sum += ans[j] * a[i][j];
        if ((sum - a[i][m])  != 0 )return 0;  // No solution
    }

    for (int i=0; i<m; ++i){
        if (where[i] == -1)return 2;  // infinite solutions
    }
    return 1; // unique solution
}
```

xorbasis.h

Description: XOR basis

b82739, 29 lines

```
// check int or long long int
struct Basis {
    int bits = 30;       // check
    array<int, 30> b;    // basis
    Basis ()
    { for (int i = 0; i < bits; i++) b[i] = 0; }
    void add (int x) {
        for (int i = bits-1; i >= 0 && x > 0; --i)
            if (b[i]) x = min(x, x ^ b[i]);
            else b[i] = x, x = 0;
    }
    void merge (const Basis &other) {
        for (int i = bits-1; i >= 0; --i) {
            if (!other.b[i]) break;
            add(other.b[i]);
        }
    }
    int getmax () {
        int ret = 0;
        for (int i = bits-1; i >= 0; --i)
            ret = max(ret, ret ^ b[i]);
        return ret;
    }
    bool isPossible (int k) const {
        for (int i = bits-1; i >= 0; --i)
            k = min(k, k ^ b[i]);
        return k == 0;
    }
};
```

2.2 Trigonometry

$$\sin(A+B) = \sin A \cos B + \cos A \sin B$$
$$\cos(A+B) = \cos A \cos B - \sin A \sin B$$
$$\tan(A+B) = \frac{\tan A + \tan B}{1 - \tan A \tan B}$$
$$\sin A + \sin B = 2 \sin \frac{A+B}{2} \cos \frac{A-B}{2}$$
$$\cos A + \cos B = 2 \cos \frac{A+B}{2} \cos \frac{A-B}{2}$$
$$(a+b)\tan(A-B)/2 = (a-b)\tan(A+B)/2$$

2.3 Geometry

2.3.1 Triangles

Side lengths: a, b, c

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

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

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

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

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: $\dfrac{\sin\alpha}{a} = \dfrac{\sin\beta}{b} = \dfrac{\sin\gamma}{c} = \dfrac{1}{2R}$

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

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

2.3.2 Quadrilaterals

For cyclic quadrilaterals the sum of opposite angles is 180°, $ef = ac + bd$, and $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

2.4 Sums

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

2.5 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, \ (-\infty < x < \infty)$$
$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, \ (-1 < x \le 1)$$
$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, \ (-1 \le x \le 1)$$
$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, \ (-\infty < x < \infty)$$
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, \ (-\infty < x < \infty)$$

2.6 Lagrange Interpolation

A Lagrange polynomial is a polynomial that interpolates a given set of data, and has the lowest degree possible. It is written as $L(x)$ and has the property that $L(x_i) = y_i$ for every point in the data set. For a set of nodes $\{x_0, x_1, \ldots x_k\}$ the Lagrange basis $\{l_0, l_1, \ldots l_k\}$ is given as

$$l_j(x) = \prod_{i \neq j} \frac{x - x_i}{x_j - x_i}$$

The Lagrange interpolating polynomial corresponding to the values $\{y_0, y_1, \dots y_k\}$ is given as

$$L(x) = \sum_j y_j l_j(x)$$

2.7 Generating Functions

Catalan numbers $C(x) = \dfrac{1-\sqrt{1-4x}}{2x}$
Bell numbers (set partitions) $B(x) = e^{e^x-1}$
Stirling numbers of the 1^{st} kind (count permutations with exactly k cycles)

$$H(x,y) = (1+x)^y = \sum_n \sum_k \begin{bmatrix} n \\ k \end{bmatrix} \frac{x^n}{n!} y^k$$

$$H_k(x) = [y^k]H(x,y) = [y^k]\exp\left(y\log\left(1+x\right)\right) = \frac{(\log\left(1+x\right))^k}{k!}$$

Stirling numbers of the 2^{nd} kind (count partitions into k subsets)

$$B(x,y) = e^{y(e^x-1)} = \sum_n \sum_k \left\{ \begin{matrix} n \\ k \end{matrix} \right\} \frac{x^n}{n!} y^k$$

$$B_k(x) = [y^k]B(x,y) = \frac{(e^x-1)^k}{k!}$$

Data structures (3)

CustomHash.h

Description: Custom Hash for $umaps < pii, int >$.
Time: $\mathcal{O}(N)$

99f5e3, 13 lines

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(pair<uint64_t,uint64_t> x) const {
        static const uint64_t FIXED_RANDOM = chrono::
            steady_clock::now().time_since_epoch().count();
        return splitmix64(x.first + FIXED_RANDOM)^(splitmix64(x
            .second + FIXED_RANDOM) >> 1);
    }
};
```

SegtreeVaibhav.h

Description: $segtree < int, op, e > seg, seg.max_right < f > (p)$
Time: $\mathcal{O}(N \log N)$

1e22a4, 104 lines

```
template <class S, S (*op)(S, S), S (*e)()> struct segtree {
  public:
    segtree() : segtree(0) {}
    explicit segtree(int n) : segtree(std::vector<S>(n, e())) {
        }
    explicit segtree(const std::vector<S>& v) : _n(int(v.size()
        )) {
        size = 1;while(size < _n)size *= 2;
        log = __builtin_ctz(size);
        d = std::vector<S>(2 * size, e());
        for (int i = 0; i < _n; i++) d[size + i] = v[i];
        for (int i = size - 1; i >= 1; i--) {
            update(i);
        }
    }

    void set(int p, S x) {
        assert(0 <= p && p < _n);
        p += size;
        d[p] = x;
        for (int i = 1; i <= log; i++) update(p >> i);
    }

    S get(int p) const {
        assert(0 <= p && p < _n);
        return d[p + size];
    }

    S prod(int l, int r) const {
        assert(0 <= l && l <= r && r <= _n);
        S sml = e(), smr = e();
        l += size;
        r += size;

        while (l < r) {
            if (l & 1) sml = op(sml, d[l++]);
            if (r & 1) smr = op(d[--r], smr);
            l >>= 1;
            r >>= 1;
        }
        return op(sml, smr);
    }

    S all_prod() const { return d[1]; }

    template <bool (*f)(S)> int max_right(int l) const {
        return max_right(l, [](S x) { return f(x); });
        //first index that does not satisfy function i.e. true
            in [l,rval)
    }
    template <class F> int max_right(int l, F f) const {
        assert(0 <= l && l <= _n);
        assert(f(e()));
        if (l == _n) return _n;
        l += size;
        S sm = e();
        do {
            while (l % 2 == 0) l >>= 1;
            if (!f(op(sm, d[l]))) {
                while (l < size) {
                    l = (2 * l);
                    if (f(op(sm, d[l]))) {
                        sm = op(sm, d[l]);
                        l++;
                    }
                }
                return l - size;
            }
            sm = op(sm, d[l]);
            l++;
        } while ((l & -l) != l);
        return _n;
    }

    template <bool (*f)(S)> int min_left(int r) const {
        return min_left(r, [](S x) { return f(x); });
        //last index that satisfy function before r i.e true in
            [rval,r)
    }
    template <class F> int min_left(int r, F f) const {
        assert(0 <= r && r <= _n);
        assert(f(e()));
        if (r == 0) return 0;
        r += size;
        S sm = e();
        do {
            r--;
            while (r > 1 && (r % 2)) r >>= 1;
            if (!f(op(d[r], sm))) {
                while (r < size) {
                    r = (2 * r + 1);
                    if (f(op(d[r], sm))) {
                        sm = op(d[r], sm);
                        r--;
                    }
                }
                return r + 1 - size;
            }
            sm = op(d[r], sm);
        } while ((r & -r) != r);
        return 0;
    }

  private:
    int _n, size, log;
    std::vector<S> d;
    void update(int k) { d[k] = op(d[2 * k], d[2 * k + 1]); }
};
```

SegmentTree.h

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

1df7e7, 36 lines

```
int t[4*N]; // N is the number of elements

void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = t[v*2] + t[v*2+1];
    }
}

int sum(int v, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && r == tr) {
        return t[v];
    }
    int tm = (tl + tr) / 2;
    return sum(v*2, tl, tm, l, min(r, tm))
           + sum(v*2+1, tm+1, tr, max(l, tm+1), r);
}

void update(int v, int tl, int tr, int pos, int new_val) {
    if (tl == tr) {
        t[v] = new_val;
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos, new_val);
        else
            update(v*2+1, tm+1, tr, pos, new_val);
        t[v] = t[v*2] + t[v*2+1];
```

```
    }
}
```

lazySegVaibhav.h

Description: $lazy_segtree < S, op, e, F, mapping, composition, id > seg$
Time: $\mathcal{O}(N \log N)$

399c0c, 174 lines

```
template <class S,
          S (*op)(S, S),
          S (*e)(),
          class F,
          S (*mapping)(F, S),
          F (*composition)(F, F),
          F (*id)()>
struct lazy_segtree {

  public:
    lazy_segtree() : lazy_segtree(0) {}
    explicit lazy_segtree(int n) : lazy_segtree(std::vector<S>(
        n, e())) {}
    explicit lazy_segtree(const std::vector<S>& v) : _n(int(v.
        size())) {
        size = 1;while(size < _n)size *= 2;
        log = __builtin_ctz(size);
        d = std::vector<S>(2 * size, e());
        lz = std::vector<F>(size, id());
        for (int i = 0; i < _n; i++) d[size + i] = v[i];
        for (int i = size - 1; i >= 1; i--) {
            update(i);
        }
    }

    void set(int p, S x) {
        assert(0 <= p && p < _n);
        p += size;
        for (int i = log; i >= 1; i--) push(p >> i);
        d[p] = x;
        for (int i = 1; i <= log; i++) update(p >> i);
    }

    S get(int p) {
        assert(0 <= p && p < _n);
        p += size;
        for (int i = log; i >= 1; i--) push(p >> i);
        return d[p];
    }

    S prod(int l, int r) {
        assert(0 <= l && l <= r && r <= _n);
        if (l == r) return e();

        l += size;
        r += size;

        for (int i = log; i >= 1; i--) {
            if (((l >> i) << i) != l) push(l >> i);
            if (((r >> i) << i) != r) push((r - 1) >> i);
        }

        S sml = e(), smr = e();
        while (l < r) {
            if (l & 1) sml = op(sml, d[l++]);
            if (r & 1) smr = op(d[--r], smr);
            l >>= 1;
            r >>= 1;
        }

        return op(sml, smr);
    }

    S all_prod() { return d[1]; }

    void apply(int p, F f) {
        assert(0 <= p && p < _n);
        p += size;
        for (int i = log; i >= 1; i--) push(p >> i);
        d[p] = mapping(f, d[p]);
        for (int i = 1; i <= log; i++) update(p >> i);
    }
    void apply(int l, int r, F f) {
        assert(0 <= l && l <= r && r <= _n);
        if (l == r) return;

        l += size;
        r += size;

        for (int i = log; i >= 1; i--) {
            if (((l >> i) << i) != l) push(l >> i);
            if (((r >> i) << i) != r) push((r - 1) >> i);
        }

        {
            int l2 = l, r2 = r;
            while (l < r) {
                if (l & 1) all_apply(l++, f);
                if (r & 1) all_apply(--r, f);
                l >>= 1;
                r >>= 1;
            }
            l = l2;
            r = r2;
        }

        for (int i = 1; i <= log; i++) {
            if (((l >> i) << i) != l) update(l >> i);
            if (((r >> i) << i) != r) update((r - 1) >> i);
        }
    }

    template <bool (*g)(S)> int max_right(int l) {
        return max_right(l, [](S x) { return g(x); });
    }
    template <class G> int max_right(int l, G g) {
        assert(0 <= l && l <= _n);
        assert(g(e()));
        if (l == _n) return _n;
        l += size;
        for (int i = log; i >= 1; i--) push(l >> i);
        S sm = e();
        do {
            while (l % 2 == 0) l >>= 1;
            if (!g(op(sm, d[l]))) {
                while (l < size) {
                    push(l);
                    l = (2 * l);
                    if (g(op(sm, d[l]))) {
                        sm = op(sm, d[l]);
                        l++;
                    }
                }
                return l - size;
            }
            sm = op(sm, d[l]);
            l++;
        } while ((l & -l) != l);
        return _n;
    }

    template <bool (*g)(S)> int min_left(int r) {
        return min_left(r, [](S x) { return g(x); });
    }
    template <class G> int min_left(int r, G g) {
        assert(0 <= r && r <= _n);
        assert(g(e()));
        if (r == 0) return 0;
        r += size;
        for (int i = log; i >= 1; i--) push((r - 1) >> i);
        S sm = e();
        do {
            r--;
            while (r > 1 && (r % 2)) r >>= 1;
            if (!g(op(d[r], sm))) {
                while (r < size) {
                    push(r);
                    r = (2 * r + 1);
                    if (g(op(d[r], sm))) {
                        sm = op(d[r], sm);
                        r--;
                    }
                }
                return r + 1 - size;
            }
            sm = op(d[r], sm);
        } while ((r & -r) != r);
        return 0;
    }

  private:
    int _n, size, log;
    std::vector<S> d;
    std::vector<F> lz;

    void update(int k) { d[k] = op(d[2 * k], d[2 * k + 1]); }
    void all_apply(int k, F f) {
        d[k] = mapping(f, d[k]);
        if (k < size) lz[k] = composition(f, lz[k]);
    }
    void push(int k) {
        all_apply(2 * k, lz[k]);
        all_apply(2 * k + 1, lz[k]);
        lz[k] = id();
    }
};
```

lazyST.h

Description: Lazy segtree
Time: $\mathcal{O}(\log N)$ per query

74d877, 80 lines

```
template<typename T>
struct LazySegmentTree{
    vector<T> st;
    void assign(int n){
        st.resize(4*n+1);
    }
    ll combine(ll x,ll y){
        return x+y;          // check which opeartion is to be
            performed
    }
    void build(vector<ll> &v1, int v, int tl, int tr){
        if(tl==tr)    st[v].sum=v1[tl];     //  check
        else{
            int mid=((tl+tr)>>1);
            build( v1, (v<<1), tl, mid );
            build( v1, (v<<1)+1, mid+1, tr );
            st[v].sum = combine( st[(v<<1)].sum , st[(v<<1)+1].
                sum );
        }
```

```
    }
    void prop(int v, int tl, int tr){
        if(st[v].mark){
            st[v].sum=(tr-tl+1)*st[v].change; // check which
                opeartion is to be performed
            if(tl!=tr){
                st[(v<<1)].change=st[(v<<1)+1].change=st[v].
                    change;
                st[(v<<1)].mark=st[(v<<1)+1].mark=1;
                st[(v<<1)].lazy=st[(v<<1)+1].lazy=0;
            }
            st[v].change=st[v].mark=0;
        }
        if(st[v].lazy!=0){
            st[v].sum+=(tr-tl+1)*st[v].lazy; // check which
                opeartion is to be performed
            if(tl!=tr){
                st[(v<<1)].lazy+=st[v].lazy;
                st[(v<<1)+1].lazy+=st[v].lazy;
            }
            st[v].lazy=0;
// if st[v].lazy is != 0 at any point, it means from that
    vertex onwards we have to make updations
        }
    }
    ll query(int v, int tl, int tr, int l, int r){
        if(tr<l || r<tl) return 0;   // check which opeartion
            is to be performed
        prop(v,tl,tr);
        if(l<=tl && tr<=r)  return st[v].sum;
        int mid=((tl+tr)>>1);
        return combine( query((v<<1),tl,mid,l,min(r,mid)),
            query((v<<1)+1,mid+1,tr,max(l,mid+1),r) );
    }
    void update_many(int v, int tl, int tr, int l, int r, ll
        newVal){
        prop(v,tl,tr);
        if(tr<l || r<tl)    return;
        if(l==tl && r==tr){
            st[v].lazy+=newVal;
            prop(v,tl,tr);
            return;
        }else{
            int mid=((tl+tr)>>1);
            update_many( (v<<1), tl, mid, l, min(r,mid), newVal
                );
            update_many( (v<<1)+1, mid+1, tr, max(l,mid+1), r,
                newVal);
            st[v].sum = combine( st[(v<<1)].sum, st[(v<<1)+1].
                sum );
        }
    }
    void change_many(int v, int tl, int tr, int l, int r, ll
        newVal){
        prop(v,tl,tr);
        if(tr<l || r<tl)    return;
        if(l==tl && r==tr){
            st[v].lazy=0,st[v].mark=1,st[v].change=newVal;
            prop(v,tl,tr);
            return;
        }else{
            int mid=((tl+tr)>>1);
            change_many( (v<<1), tl, mid, l, min(r,mid), newVal
                );
            change_many( (v<<1)+1, mid+1, tr, max(l,mid+1), r,
                newVal);
            st[v].sum = combine( st[(v<<1)].sum, st[(v<<1)+1].
                sum );
        }
    }
};

struct Node{
    ll sum,lazy,change;
    bool mark;
};
LazySegmentTree<Node> lazyseg;
```

pertree.h

Description: Persistent segtree Inputs must be in [tl, tr).
Time: $\mathcal{O}(N \log N)$

42b5ec, 56 lines

```
struct Vertex {
    Vertex *l, *r;
    int sum;

    Vertex(int val) : l(nullptr), r(nullptr), sum(val) {}
    Vertex(Vertex *l, Vertex *r) : l(l), r(r), sum(0) {
        if (l) sum += l->sum;
        if (r) sum += r->sum;
    }
};

Vertex* build(int a[], int tl, int tr) {
    if (tl == tr)
        return new Vertex(a[tl]);
    int tm = (tl + tr) / 2;
    return new Vertex(build(a, tl, tm), build(a, tm+1, tr));
}

int query(Vertex* v, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && tr == r)
        return v->sum;
    int tm = (tl + tr) / 2;
    return get_sum(v->l, tl, tm, l, min(r, tm))
         + get_sum(v->r, tm+1, tr, max(l, tm+1), r);
}

Vertex* update(Vertex* v, int tl, int tr, int pos, int new_val)
    {
    if (tl == tr)
        return new Vertex(new_val);
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return new Vertex(update(v->l, tl, tm, pos, new_val), v
            ->r);
    else
        return new Vertex(v->l, update(v->r, tm+1, tr, pos,
            new_val));
}

int find_kth(Vertex* vl, Vertex *vr, int tl, int tr, int k) {
    if (tl == tr)
        return tl;
    int tm = (tl + tr) / 2, left_count = vr->l->sum - vl->l->
        sum;
    if (left_count >= k)
        return find_kth(vl->l, vr->l, tl, tm, k);
    return find_kth(vl->r, vr->r, tm+1, tr, k-left_count);
}

// int tl = 0, tr = MAX_VALUE + 1;
// std::vector<Vertex*> roots;
// roots.push_back(build(tl, tr));
// for (int i = 0; i < a.size(); i++) {
//     roots.push_back(update(roots.back(), tl, tr, a[i]));
// }

// find the 5th smallest number from the subarray [a[2], a[3],
    ..., a[19]]
// int result = find_kth(roots[2], roots[20], tl, tr, 5);
```

UnionFindRollback.h

Description: LCA Inputs must be in [0, mod).
Time: $\mathcal{O}(logN)$ per query

6f3f33, 40 lines

```
struct DSUrb {
    vi e; int comp;void init(int n) { e = vi(n,-1); comp = n;}
    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)]; }
    vector<array<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; }
        comp --;
        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.back(); mod.pop_back();
        if (a[0] != -1) e[a[0]] = a[2], e[a[1]] = a[3],comp++;
    }
};

template<int SZ> struct DynaCon {
    DSUrb D; vpii seg[2*SZ]; vi ans;
    void init(int n){D.init(n);ans.resize(SZ);}
    void upd(int l, int r, pii p) {  // add edge p to all times
        in interval [l, r]
        for (l += SZ, r += SZ+1; l < r; l /= 2, r /= 2) {
            if (l&1) seg[l++].pb(p);
            if (r&1) seg[--r].pb(p);
        }
    }
    void process(int ind) {
        for(auto t : seg[ind]) D.unite(t.ff,t.ss);
        if (ind >= SZ) {
            ans[ind-SZ] = D.comp;
            // do stuff with D at time ind-SZ
        } else process(2*ind), process(2*ind+1);
        for(auto t : seg[ind]) D.rollback();
    }
};

DynaCon<300001> dy;
```

LineContainer.h

Description: Container where you can add lines of the form kx+m, and query maximum values at points x. Useful for dynamic programming ("convex hull trick").
Time: $\mathcal{O}(\log N)$

8ec1c7, 30 lines

```
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
```

```
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

Treap.h

Description: implicit treap
Time: $\mathcal{O}(\log N)$ per operation

f4e7c0, 66 lines

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
     count());
using pt = struct tnode*;
pt root = NULL;
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() { rep(i,0,2) delete c[i]; }
};
int getsz(pt x) { return x?x->sz:0; }
ll getsum(pt x) { return x?x->sum:0; }
void prop(pt x) { // lazy propagation
    if (!x || !x->flip) return;
    swap(x->c[0],x->c[1]);
    x->flip = 0; rep(i,0,2) if (x->c[i]) x->c[i]->flip ^= 1;
}
pt calc(pt x) {
    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> splitsz(pt t, int sz) { // sz nodes go to left used
      for implicit
    if (!t) return {t,t};
    prop(t);
    if (getsz(t->c[0]) >= sz) {
        auto p = splitsz(t->c[0],sz); t->c[0] = p.ss;
        return {p.ff,calc(t)};
    } else {
        auto p=splitsz(t->c[1],sz-getsz(t->c[0])-1); t->c[1]=p.
             ff;
        return {calc(t),p.ss};
    }
}
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,int idx) { // insert v at idx(0 based
     indexing)
    auto a = splitsz(x,idx);
    return merge(a.ff,merge(new tnode(v),a.ss)); }
pt del(pt x, int idx) { // delete v at idx(0 based indexing)
    auto a = splitsz(x,idx), b = splitsz(a.ss,1);
    return merge(a.ff,b.ss); }
int find_kidx(pt t,int idx){//idx is 1 based
    assert(getsz(t) >= idx);
    prop(t);
    if(getsz(t->c[0]) == idx-1)return t->val;
    else if(getsz(t->c[0]) < idx)return find_kidx(t->c[1],idx-
         getsz(t->c[0])-1);
    else return find_kidx(t->c[0],idx);
}
//root = ins(root,a[i],i)
//auto a = splitsz(root,l);auto b = splitsz(a.ss,r-l);ll ans =
     b.ff->sum;
//root = merge(a.ff,merge(b.ff,b.ss)); sum l to r
```

FenwickTree.h

Description: Can be used for min/max operations for prefix queries.
Time: $\mathcal{O}(\log N)$

a2c32f, 30 lines

```
struct FenwickTree {
    vector<int> bit;
    int n;

    FenwickTree(int n) {
      this->n = n;
      bit.assign(n, 0);
    }

    int sum(int r) {
      int ret = 0;
      for (; r >= 0; r = (r & (r + 1)) - 1) ret += bit[r];
      return ret;
    }

    void add(int idx, int delta) {
      for(; idx < n; idx = idx | (idx + 1)) bit[idx] += delta;
    }

    // First index having prefix sum >= v
    int lower_bound(int v){
      int sum = 0, pos = 0;
      for(int i=25; i>=0; i--){
        if(pos + (1 << i) - 1 < n and sum + bit[pos + (1 << i)
             - 1] < v){
          sum += bit[pos + (1 << i) - 1], pos += (1 << i);
        }
      }
      return pos;
    }
};
```

FenwickTree2d.h

Description: 2-d range queries and point updates (0-indexed)
Time: $\mathcal{O}(\log N \log M)$. (Use persistent segment trees for $\mathcal{O}(\log N)$.)

55e85c, 25 lines

```
template <typename T>
struct BIT2D {
  const int n, m;
  vector<vector<T>> bit;

  BIT2D(int n, int m) : n(n), m(m), bit(n + 1, vector<T>(m + 1)
       ) {}
  void add(int r, int c, T val) {
    r++, c++;
    for (; r <= n; r += r & -r) {
      for (int i = c; i <= m; i += i & -i) { bit[r][i] += val;
           }
    }
  }
  T rect_sum(int r, int c) {
    r++, c++;
    T sum = 0;
    for (; r > 0; r -= r & -r) {
      for (int i = c; i > 0; i -= i & -i) { sum += bit[r][i]; }
    }
    return sum;
  }
  T rect_sum(int r1, int c1, int r2, int c2) {
    return rect_sum(r2, c2) - rect_sum(r2, c1 - 1) - rect_sum(
         r1 - 1, c2) +
           rect_sum(r1 - 1, c1 - 1);
  }
};
```

Segtree2d.h

Description: Call $build_x$, sum_x and $update_x$ for respective operations. Memory is $O(16 \cdot N \cdot M)$ and constant factor is too large, be careful.
Time: $\mathcal{O}(\log N \log M)$

ab2d1f, 61 lines

```
int t[4*N][4*M];

void build_y(int vx, int lx, int rx, int vy, int ly, int ry) {
    if (ly == ry) {
        if (lx == rx) t[vx][vy] = a[lx][ly];
        else t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
    } else {
        int my = (ly + ry) / 2;
        build_y(vx, lx, rx, vy*2, ly, my);
        build_y(vx, lx, rx, vy*2+1, my+1, ry);
        t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
    }
}

void build_x(int vx, int lx, int rx) {
    if (lx != rx) {
        int mx = (lx + rx) / 2;
        build_x(vx*2, lx, mx);
        build_x(vx*2+1, mx+1, rx);
    }
    build_y(vx, lx, rx, 1, 0, m-1);
}

int sum_y(int vx, int vy, int tly, int try_, int ly, int ry) {
    if (ly > ry)  return 0;
    if (ly == tly && try_ == ry) return t[vx][vy];
    int tmy = (tly + try_) / 2;
    return sum_y(vx, vy*2, tly, tmy, ly, min(ry, tmy))
         + sum_y(vx, vy*2+1, tmy+1, try_, max(ly, tmy+1), ry);
}

int sum_x(int vx, int tlx, int trx, int lx, int rx, int ly, int
     ry) {
    if (lx > rx) return 0;
    if (lx == tlx && trx == rx)
        return sum_y(vx, 1, 0, m-1, ly, ry);
```

```
    int tmx = (tlx + trx) / 2;
    return sum_x(vx*2, tlx, tmx, lx, min(rx, tmx), ly, ry)
        + sum_x(vx*2+1, tmx+1, trx, max(lx, tmx+1), rx, ly, ry
            );
}

void update_y(int vx, int lx, int rx, int vy, int ly, int ry,
    int x, int y, int new_val) {
    if (ly == ry) {
        if (lx == rx) t[vx][vy] = new_val;
        else t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
    } else {
        int my = (ly + ry) / 2;
        if (y <= my)
        update_y(vx, lx, rx, vy*2, ly, my, x, y, new_val);
        else update_y(vx, lx, rx, vy*2+1, my+1, ry, x, y,new_val
            );
        t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
    }
}

void update_x(int vx, int lx, int rx, int x, int y, int new_val
    ){
    if (lx != rx) {
        int mx = (lx + rx) / 2;
        if (x <= mx) update_x(vx*2, lx, mx, x, y, new_val);
        else update_x(vx*2+1, mx+1, rx, x, y, new_val);
    }
    update_y(vx, lx, rx, 1, 0, m-1, x, y, new_val);
}
```

SparseTable.h

Description: Sparse table
Time: $\mathcal{O}(N \log N)$ build, $\mathcal{O}(1)$ per query, each times combine cost. 701028, 27 lines

```
const int N=1e5+5; const int maxn=N;const int max_logn=20;
template<typename T>
struct SparseTable{
    int log[maxn];
    T dp[max_logn][maxn];
    T combine(T a,T b){return __gcd(a,b);}
    SparseTable(){
        log[1] = 0;
        for (int i = 2; i < maxn; i++)
            log[i] = log[i/2] + 1;
    }
    void build(vector<T> b)
    {
        int n=b.size();
        for (int i = 0; i < n; ++i){
            dp[0][i]=b[i];
        }
        for (int j = 1; j < max_logn; j++)
            for (int i = 0; i + (1 << j) <= n; i++)
                dp[j][i] = combine(dp[j - 1][i], dp[j - 1][i +
                    (1 << (j - 1))]);
    }
    T query(int l,int r)
    {
        int j=log[r-l+1];
        return combine(dp[j][l],dp[j][r-(1<<j)+1]);
    }
};SparseTable<int> sp;
```

MoQueries.h

Description: Answer interval or tree path queries by finding an approximate TSP through the queries, and moving from one query to the next by adding/removing points at the ends. If values are on tree edges, change `step` to add/remove the edge (a, c) and remove the initial `add` call (but keep `in`).
Time: $\mathcal{O}(N\sqrt{Q})$ a12ef4, 49 lines

```
void add(int ind, int end) { ... } // add a[ind] (end = 0 or 1)
void del(int ind, int end) { ... } // remove a[ind]
int calc() { ... } // compute current answer

vi mo(vector<pii> Q) {
  int L = 0, R = 0, blk = 350; // ~N/sqrt(Q)
  vi s(sz(Q)), res = s;
#define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1))
  iota(all(s), 0);
  sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
  for (int qi : s) {
    pii q = Q[qi];
    while (L > q.first) add(--L, 0);
    while (R < q.second) add(R++, 1);
    while (L < q.first) del(L++, 0);
    while (R > q.second) del(--R, 1);
    res[qi] = calc();
  }
  return res;
}

vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int root=0){
  int N = sz(ed), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
  vi s(sz(Q)), res = s, I(N), L(N), R(N), in(N), par(N);
  add(0, 0), in[0] = 1;
  auto dfs = [&](int x, int p, int dep, auto& f) -> void {
    par[x] = p;
    L[x] = N;
    if (dep) I[x] = N++;
    for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
    if (!dep) I[x] = N++;
    R[x] = N;
  };
  dfs(root, -1, 0, dfs);
#define K(x) pii(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] / blk & 1))
  iota(all(s), 0);
  sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
  for (int qi : s) rep(end,0,2) {
    int &a = pos[end], b = Q[qi][end], i = 0;
#define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
                  else { add(c, end); in[c] = 1; } a = c; }
    while (!(L[b] <= L[a] && R[a] <= R[b]))
      I[i++] = b, b = par[b];
    while (a != b) step(par[a]);
    while (i--) step(I[i]);
    if (end) res[qi] = calc();
  }
  return res;
}
```

Mos.h

Description: Mo's algorithm
Time: $\mathcal{O}\left((N+Q)\sqrt{N}\right)$ 6563d7, 36 lines

```
const int block=350;
struct Query {
    int l, r, idx;
    inline pair<int, int> toPair() const {
        return make_pair(l / block, ((l / block) & 1) ? -r : +r
            );
    }
};

inline bool operator<(const Query &a, const Query &b) {
    return a.toPair() < b.toPair();
}
void remove(int idx){
}
void add(int idx){
}
int get_answer(){
}

vector<int> mo_s_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());
    int cur_l = 0,cur_r = -1;
    for (Query q : queries) {
      while (cur_l > q.l)
          cur_l--, add(cur_l);
      while (cur_r < q.r)
          cur_r++, add(cur_r);
      while (cur_l < q.l)
          remove(cur_l), cur_l++;
      while (cur_r > q.r)
          remove(cur_r), cur_r--;
      answers[q.idx] = get_answer();
    }
    return answers;
}
```

HilbertOrder.h

Description: Returns hilbert curve order of (x, y) 372304, 12 lines

```
ll hilbertorder (int x, int y) {
    ll o = 0; const int mx = 1 << 20;
    for (int p = mx; p; p >>= 1){
        bool a = x&p, b = y&p;
        o = (o << 2) | (a * 3) ^ static_cast<int>(b);
        if (!b) {
            if (a) x = mx - x, y = mx - y;
            x ^= y ^= x ^= y;
        }
    }
    return o;
}
```

WaveletTree.h

Description: Finds kth smallest number in a range. l,r,k are 0-indexed.
Time: $\mathcal{O}((N+Q)\log N)$ b6e3be, 103 lines

```
size_t popcount64(uint_fast64_t x) {
  return __builtin_popcountll(x);
}

class bit_vector {
  using size_t = size_t;
  static constexpr size_t wordsize = numeric_limits<size_t>::
      digits;

  class node_type {
  public:
    size_t bit, sum;
    node_type() : bit(0), sum(0) {}
  };

  vector<node_type> v;

public:
  bit_vector() = default;
  explicit bit_vector(const vector<bool> a) : v(a.size() /
      wordsize + 1) {
    {
      const size_t s = a.size();
      for (size_t i = 0; i != s; i += 1)
```

```
        v[i / wordsize].bit |= static_cast<size_t>(a[i] ? 1 : 0)
                               << i % wordsize;
    }
    {
      const size_t s = v.size();
      for (size_t i = 1; i != s; i += 1)
        v[i].sum = v[i - 1].sum + popcount64(v[i - 1].bit);
    }
  }

  size_t rank0(const size_t index) const { return index - rank1(index); }
  size_t rank1(const size_t index) const {
    return v[index / wordsize].sum +
           popcount64(v[index / wordsize].bit &
                      ~(~static_cast<size_t>(0) << index % wordsize));
  }
};

template <class Key> class wavelet_matrix {
  using size_t = size_t;

public:
  using key_type = Key;
  using value_type = Key;
  using size_type = size_t;

private:
  static bool test(const key_type x, const size_t k) {
    return (x & static_cast<key_type>(1) << k) != 0;
  }
  static void set(key_type &x, const size_t k) {
    x |= static_cast<key_type>(1) << k;
  }

  size_t size_;
  vector<bit_vector> mat;

public:
  wavelet_matrix() = default;
  explicit wavelet_matrix(const size_t bit_length, vector<key_type> a)
    : size_(a.size()), mat(bit_length, bit_vector()) {
    const size_t s = size();
    for (size_t p = bit_length; p != 0;) {
      p -= 1;
      {
        vector<bool> t(s);
        for (size_t i = 0; i != s; i += 1) t[i] = test(a[i], p);
        mat[p] = bit_vector(t);
      }
      {
        vector<key_type> v0, v1;
        for (const size_t e : a)
          (test(e, p) ? v1 : v0).push_back(e);
        const auto itr = copy(v0.cbegin(), v0.cend(), a.begin());
        copy(v1.cbegin(), v1.cend(), itr);
      }
    }
  }

  size_t size() const { return size_; }

  key_type quantile(size_t first, size_t last, size_t k) const {
    key_type ret = 0;
    for (size_t p = mat.size(); p != 0;) {
      p -= 1;
      const bit_vector &v = mat[p];
      const size_t z = v.rank0(last) - v.rank0(first);
      if (k < z) {
        first = v.rank0(first), last = v.rank0(last);
      } else {
        set(ret, p), k -= z;
        const size_t b = v.rank0(size());
        first = b + v.rank1(first), last = b + v.rank1(last);
      }
    }
    return ret;
  }
};

// const wavelet_matrix<int> wm(30, a);
// int(wm.quantile(l, r, k)) for range [l,r)
```

Numerical (4)

4.1 Polynomials and recurrences

PolyInterpolate.h

Description: Given n points (x[i], y[i]), computes an n-1-degree polynomial p that passes through them: $p(x) = a[0] * x^0 + ... + a[n-1] * x^{n-1}$. For numerical precision, pick $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \dots n-1$.
Time: $\mathcal{O}(n^2)$

08bf48, 13 lines

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd res(n), temp(n);
  rep(k,0,n-1) rep(i,k+1,n)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  double last = 0; temp[0] = 1;
  rep(k,0,n) rep(i,0,n) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
}
```

BerlekampMassey.h

Description: Recovers any n-order linear recurrence relation from the first $2n$ terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size $\leq n$.
Usage: berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}
Time: $\mathcal{O}(N^2)$

"../number-theory/ModPow.h" 96548b, 20 lines

```
vector<ll> berlekampMassey(vector<ll> s) {
  int n = sz(s), L = 0, m = 0;
  vector<ll> C(n), B(n), T;
  C[0] = B[0] = 1;

  ll b = 1;
  rep(i,0,n) { ++m;
    ll d = s[i] % mod;
    rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
    if (!d) continue;
    T = C; ll coef = d * modpow(b, mod-2) % mod;
    rep(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
    if (2 * L > i) continue;
    L = i + 1 - L; B = T; b = d; m = 0;
  }

  C.resize(L + 1); C.erase(C.begin());
  for (ll& x : C) x = (mod - x) % mod;
  return C;
}
```

4.2 Optimization

4.3 Matrices

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

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)
}
```

4.4 Fourier transforms

FastFourierTransform.h

Description: 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: 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 $N = |A| + |B|$ (~1s 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<ll> 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;
}
```

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

027467, 16 lines

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

GcdConv.h

Description: Given arrays a and b (1-indexed), finds array c such that $c[z] = \sum_{z=gcd(x,y)} a[x] \cdot b[y]$,
Time: $\mathcal{O}(N \log N)$

947e33, 35 lines

```
const int MOD = 998244353;

void zeta(vector<ll>& a) {
  int n = a.size() - 1;
  for (int i = 1; i <= n; ++i) {
    for (int j = 2; j <= n / i; ++j) {
      a[i] += a[i * j];
      if (a[i] >= MOD) a[i] -= MOD;
    }
  }
}

void mobius(vector<ll>& a) {
  int n = a.size() - 1;
  for (int i = n; i >= 1; --i) {
    for (int j = 2; j <= n / i; ++j) {
      a[i] -= a[i * j];
      if (a[i] < 0) a[i] += MOD;
    }
  }
}

vector<ll> gcd_conv(vector<ll> a, vector<ll> b){
    zeta(a);
    zeta(b);

    int n = (int)a.size() - 1;
    vector<ll> c(n+1);
    for(int i=1;i<=n;i++){
        c[i]=(a[i]*b[i])%MOD;
    }

    mobius(c);
    return c;
}
```

LcmConv.h

Description: Given arrays a and b (1-indexed), finds array c such that $c[z] = \sum_{z=lcm(x,y)} a[x] \cdot b[y]$,
Time: $\mathcal{O}(N \log N)$

349f88, 35 lines

```
const int MOD = 998244353;

void zeta(vector<ll>& a) {
  int n = a.size() - 1;
  for (int i = n; i >= 1; --i) {
    for (int j = 2; j <= n / i; ++j) {
      a[i * j] += a[i];
      if (a[i * j] >= MOD) a[i * j] -= MOD;
    }
  }
}

void mobius(vector<ll>& a) {
  int n = a.size() - 1;
  for (int i = 1; i <= n; ++i) {
    for (int j = 2; j <= n / i; ++j) {
      a[i * j] -= a[i];
      if (a[i * j] < 0) a[i * j] += MOD;
    }
  }
}

vector<ll> lcm_conv(vector<ll> a, vector<ll> b){
    zeta(a);
    zeta(b);

    int n = (int)a.size() - 1;
    vector<ll> c(n+1);
    for(int i=1;i<=n;i++){
        c[i]=(a[i]*b[i])%MOD;
    }

    mobius(c);
    return c;
}
```

MinPlusConv.h

Description: Min-plus convolution for arbitrary + convex arrays. B is convex : $b_{i+1} - b_i \leq b_{i+2} - b_{i+1}$ Given arrays a and b (1-indexed), finds array c such that $c[k] = \min_{k=i+j} a[i] + b[j]$,
Time: $\mathcal{O}((N + M) \log N)$

2e9209, 37 lines

```
#define int long long

template <typename F>
vector<int> monotone_minima(int H, int W, F select) {
  vector<int> min_col(H);
  auto dfs = [&](auto& dfs, int x1, int x2, int y1, int y2) ->
      void {
    if (x1 == x2) return;
    int x = (x1 + x2) / 2;
    int best_y = y1;
    for (int y = y1 + 1; y < y2; ++y) {
      if (select(x, best_y, y)) best_y = y;
    }
    min_col[x] = best_y;
    dfs(dfs, x1, x, y1, best_y + 1);
    dfs(dfs, x + 1, x2, best_y, y2);
  };
  dfs(dfs, 0, H, 0, W);
  return min_col;
}

// B is convex
vector<int> min_plus_convolution(vector<int> A, vector<int> B)
    {
  int N = A.size(), M = B.size();
  for (int i = 0; i < M - 2; ++i) assert(B[i] + B[i + 2] >= 2 *
      B[i + 1]);
  auto select = [&](int i, int j, int k) -> bool {
    if (i < k) return false;
    if (i - j >= M) return true;
    return A[j] + B[i - j] >= A[k] + B[i - k];
  };
  vector<int> J = monotone_minima(N + M - 1, N, select);
  vector<int> C(N + M - 1);
```

```
  for (int i = 0; i < N + M - 1; ++i) {
    int j = J[i];
    C[i] = A[j] + B[i - j];
  }
  return C;
}
```

SubsetConv.h

Description: Subset convolution. $c[k] = \sum_{k=i|j,i\&j=0} a[i] \cdot b[j]$. The size of a and b must be same and a power of two.
Time: $\mathcal{O}\left(N(\log N)^2\right)$, works for $n = 2^{20}$ under 2 sec.
af2d58, 35 lines

```
typedef long long ll;
constexpr int MOD = 998244353;

auto sos (vector<ll>& a, const bool invert = false) {
    const size_t n = size(a);
    assert(__builtin_popcount(n) == 1);
    for (int i = 1; i < n; i <<= 1)
        for (int ms = 0; ms < n; ++ms)
            if ((ms & i) == 0)
                (a[ms | i] += (invert? MOD-a[ms]: a[ms])),
                a[ms | i] -= (a[ms | i] >= MOD ? MOD: 0);
}

// a contains the convoluted result
void subset_conv (vector<ll>& a, const vector<ll>& b) {
    const int n = size(a);
    assert(__builtin_popcount(n) == 1 and size(b) == n);
    const int p = __builtin_ctz(n) + 1;
    vector a_cap(p, vector(n, ll()));
    vector b_cap(p, vector(n, ll()));
    vector c_cap(p, vector(n, ll()));
    for (int i = 0; i < n ; ++i)
        a_cap[__builtin_popcount(i)][i] = a[i],
        b_cap[__builtin_popcount(i)][i] = b[i];
    for (int i = 0; i < p; ++i)
        sos(a_cap[i]), sos(b_cap[i]);
    for (int i = 0; i < p; ++i) {
        for (int j = 0; j <= i; ++j)
            for (int ms = 0; ms < n; ++ms)
                (c_cap[i][ms] += a_cap[j][ms] * b_cap[i - j][ms
                    ]) %= MOD;
        sos(c_cap[i], true);
    }
    for (int i = 0; i < n ; ++i)
        a[i] = c_cap[__builtin_popcount(i)][i];
}
```

NTT.h

Description: Use $RT = 5$ in case of 998244353 and 1000000007 and $RT = 62$ in case of any other MOD. Use conv for 998244353 and *conv_general* for any other MOD.
Time: $\mathcal{O}(N \log N)$
89d2bb, 58 lines

```
template<int MOD, int RT> struct mint {
    static const int mod = MOD; int v;
    static constexpr mint rt() { return RT; } // primitive root
    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<998244353,5>; // Check mod
template<class T> void fft(vector<T>& A, bool invert = 0) { //
    NTT
    int n = A.size(); assert((T::mod-1)%n == 0); vector<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) for(int j = 0;j
            < b;j++) {
            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+A.begin(),A.end());
        T z = inv(T(n)); for(auto &t : A) t *= z; }
} // for NTT-able moduli
template<class T> vector<T> conv(vector<T> A, vector<T> B) {
    if (!min(A.size(),B.size())) return {};
    int s = A.size()+B.size()-1, n = 1; for (; n < s; n *= 2);
    A.resize(n), fft(A); B.resize(n), fft(B);
    for(int i = 0;i < n;i++) A[i] *= B[i];
    fft(A,1); A.resize(s); return A;
}
template<class M, class T> vector<M> mulMod(const vector<T>& x,
    const vector<T>& y) {
    auto con = [](const vector<T>& v) {
        vector<M> w(v.size()); for(int i = 0;i < v.size();i++)
            w[i] = (int)v[i];
        return w; };
    return conv(con(x), con(y));
} // arbitrary moduli
template<class T> vector<T> conv_general(const vector<T>& A,
    const vector<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 = c0.size(); vector<T> res(n); m1 r01 = inv(m1(m0::
        mod));
    m2 r02 = inv(m2(m0::mod)), r12 = inv(m2(m1::mod));
    for(int i = 0;i < n;i++) { // a=remainder mod m0::mod, b
        fixes it mod m1::mod
        int a = c0[i].v, b = ((c1[i]-a)*r01).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;
}

// For mod M,using mi = mint<M,RT>
// vector<mi> a, b
// auto c = conv_general(a, b)
```

Number theory (5)

5.1 Modular arithmetic

FastMod.h

Description: Compute $a\%b$ about 5 times faster than usual, where b is constant but not known at compile time. Returns a value congruent to a (mod b) in the range $[0, 2b)$.
751a02, 8 lines

```
typedef unsigned long long ull;
struct FastMod {
  ull b, m;
  FastMod(ull b) : b(b), m(-1ULL / b) {}
  ull reduce(ull a) { // a % b + (0 or b)
    return a - (ull)((__uint128_t(m) * a) >> 64) * b;
  }
};
```

ModInverse.h

Description: Pre-computation of modular inverses. Assumes LIM $\leq$ mod and that mod is a prime.
6f684f, 3 lines

```
const ll mod = 1000000007, LIM = 200000;
ll* inv = new ll[LIM] - 1; inv[1] = 1;
rep(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;
```

ModLog.h

Description: Returns the smallest $x > 0$ s.t. $a^x = b \pmod m$, or -1 if no such x exists. modLog(a,1,m) can be used to calculate the order of a.
Time: $\mathcal{O}\left(\sqrt{m}\right)$
c040b8, 11 lines

```
ll modLog(ll a, ll b, ll m) {
  ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
  unordered_map<ll, ll> A;
  while (j <= n && (e = f = e * a % m) != b % m)
    A[e * b % m] = j++;
  if (e == b % m) return j;
  if (__gcd(m, e) == __gcd(m, b))
    rep(i,2,n+2) if (A.count(e = e * f % m))
      return n * i - A[e];
  return -1;
}
```

ModSum.h

Description: Sums of mod'ed arithmetic progressions.
modsum(to, c, k, m) = $\sum_{i=0}^{\mathrm{to}-1} (ki + c)\%m$. divsum is similar but for floored division.
Time: $\log(m)$, with a large constant.
5c5bc5, 16 lines

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }

ull divsum(ull to, ull c, ull k, ull m) {
  ull res = k / m * sumsq(to) + c / m * to;
  k %= m; c %= m;
  if (!k) return res;
  ull to2 = (to * k + c) / m;
  return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}

ll modsum(ull to, ll c, ll k, ll m) {
  c = ((c % m) + m) % m;
  k = ((k % m) + m) % m;
  return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

ModMulLL.h

Description: Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$.
Time: $\mathcal{O}(1)$ for modmul, $\mathcal{O}(\log b)$ for modpow
bbbd8f, 11 lines

```
typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
  ull ans = 1;
  for (; e; b = modmul(b, b, mod), e /= 2)
    if (e & 1) ans = modmul(ans, b, mod);
```

```
  return ans;
}
```

ModSqrt.h
Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution).
Time: $\mathcal{O}\left(\log^2 p\right)$ worst case, $\mathcal{O}\left(\log p\right)$ for most p

"ModPow.h" 19a793, 24 lines

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(modpow(a, (p-1)/2, p) == 1); // else no solution
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0)
    ++r, s /= 2;
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p), g = modpow(n, s, p);
  for (;; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m)
      t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

5.2 Primality

FastEratosthenes.h
Description: Prime sieve for generating all primes smaller than LIM.
Time: LIM=1e9 $\approx$ 1.5s

6b2912, 20 lines

```
const int LIM = 1e6;
bitset<LIM> isPrime;
vi eratosthenes() {
  const int S = (int)round(sqrt(LIM)), R = LIM / 2;
  vi pr = {2}, sieve(S+1); pr.reserve(int(LIM/log(LIM)*1.1));
  vector<pii> cp;
  for (int i = 3; i <= S; i += 2) if (!sieve[i]) {
    cp.push_back({i, i * i / 2});
    for (int j = i * i; j <= S; j += 2 * i) sieve[j] = 1;
  }
  for (int L = 1; L <= R; L += S) {
    array<bool, S> block{};
    for (auto &[p, idx] : cp)
      for (int i=idx; i < S+L; idx = (i+=p)) block[i-L] = 1;
    rep(i,0,min(S, R - L))
      if (!block[i]) pr.push_back((L + i) * 2 + 1);
  }
  for (int i : pr) isPrime[i] = 1;
  return pr;
}
```

MillerRabin.h
Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to $7 \cdot 10^{18}$; for larger numbers, use Python and extend A randomly.
Time: 7 times the complexity of $a^b \mod c$.

"ModMulLL.h" 60dcd1, 12 lines

```
bool isPrime(ull n) {
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
      s = __builtin_ctzll(n-1), d = n >> s;
  for (ull a : A) {   // ^ count trailing zeroes
    ull p = modpow(a%n, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n && i--)
      p = modmul(p, p, n);
    if (p != n-1 && i != s) return 0;
  }
  return 1;
}
```

Factor.h
Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).
Time: $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.

"ModMulLL.h", "MillerRabin.h" a33cf6, 18 lines

```
ull pollard(ull n) {
  auto f = [n](ull x) { return modmul(x, x, n) + 1; };
  ull 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);
}
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

Lehmer.h
Description: lehmer(n) gives the count of prime numbers <= n. Highly optimised, works well for $n <= 10^{12}$ under 1.5 sec.
Time: $\mathcal{O}\left(N^{\frac{2}{3}}\right)$

a4936d, 66 lines

```
#define ll long long

const int MAXN=100;
const int MAXM=100010;
const int MAXP=10000010;

int prime_cnt[MAXP];
ll dp[MAXN][MAXM];

vector<int> primes;
bitset<MAXP> is_prime;

void sieve(){
   is_prime[2]=true;
   for(int i=3;i<MAXP;i+=2){
      is_prime[i]=true;
   }
   for(int i=3;i*i<MAXP;i+=2){
    for(int j=i*i;is_prime[i] && j<MAXP; j+=(i<<1)){
           is_prime[j]=false;
     }
   }

   for(int i=1;i<MAXP;i++){
      prime_cnt[i]=prime_cnt[i-1]+is_prime[i];
      if(is_prime[i]){
         primes.push_back(i);
      }
   }
}

void gen(){
    sieve();
    for(int i=0;i<MAXM;i++){
         dp[0][i]=i;
    }
    for(int n=1;n<MAXN;n++){
        for(int m=0;m<MAXM;m++){
             dp[n][m]=dp[n - 1][m]-dp[n - 1][m/primes[n-1]];
        }
    }
}

ll phi(ll m,ll n){
    if (n==0) return m;
    if (m<MAXM && n<MAXN) return dp[n][m];
    if ((ll)primes[n-1]*primes[n - 1]>=m && m<MAXP)
        return prime_cnt[m]-n+1;

    return phi(m,n-1)-phi(m/primes[n - 1],n-1);
}

ll lehmer(ll m){
    if (m<MAXP) return prime_cnt[m];

    int s=sqrtl(0.5+m), y=cbrtl(0.5+m);
    int a=prime_cnt[y];

    ll res = phi(m,a)+a-1;
    for (int i=a;primes[i]<=s;i++){
        res=res-lehmer(m/primes[i])+lehmer(primes[i])-1;
    }
    return res;
}

// Call gen() in main
```

5.3 Divisibility

diophantine.h
Description: Solves the equation $ax+by=c$. Iterate through $x = l_x + k \cdot \frac{b}{g}$ for all $k \geq 0$ until $x = r_x$, and find the corresponding y values using the equation $ax + by = c$.
Time: $\mathcal{O}\left(\log(max(a,b))\right)$

668881, 71 lines

```
#define int long long

int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
      x = 1, y = 0;
      return a;
    }
    int x1, y1, d = gcd(b, a % b, x1, y1);
    x = y1, y = x1 - y1 * (a / b);
    return d;
}

bool find_any_solution(int a, int b, int c, int &x0, int &y0,
     int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }
    x0 *= c / g, y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
```

```
void shift_solution(int & x, int & y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}

// Returns the number of solutions
int find_all_solutions(int a, int b, int c, int minx, int maxx,
     int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g, b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

CRT.h

Description: Solves system of equations $x = a_i \pmod{m_i}$. When $m_1, m_2, \ldots$ are not coprime, we take $M = lcm(m_1, m_2, \ldots)$ and we break $a = a_i \pmod{m_i}$ into $a = a_i \pmod{{p_j}^{n_j}}$ for all prime factors p_j of m_i. and then proceed similarly. The congruence with the highest prime power modulus will be the strongest congruence of all congruences based on the same prime number. Either it will give a contradiction with some other congruence, or it will imply already all other congruences. If there are no contradictions, then the system of equation has a solution. We can ignore all congruences except the ones with the highest prime power moduli. c88c31, 16 lines

```
struct Congruence {
    long long a, m;
};

long long chinese_remainder_theorem(vector<Congruence> const&
     congruences) {
    long long M = 1, solution = 0;
    for (auto const& congruence : congruences) {
        M *= congruence.m;
    }
    for (auto const& congruence : congruences) {
        long long a_i = congruence.a, M_i = M / congruence.m;
        long long N_i = mod_inv(M_i, congruence.m);
        solution = (solution + a_i * M_i % M * N_i) % M;
    }
    return solution;
}
```

primitiveroots.h

Description: Primitive roots g is a primitive root modulo n if and only if the smallest integer k for which $g^k = 1 (mod n)$ is equal to phi(n). Primitive root modulo n exists if and only if: - n is 1, 2, 4, or - n is power of an odd prime number $(n = p^k)$, or - n is twice power of an odd prime number $(n = 2*(p^k))$ The number of primitive roots modulo n is equal to phi(phi(n)). 04ffae, 25 lines

```
int powmod (int a, int b, int p) {
    int res = 1;
    while (b)
        if (b & 1) {res = int (res * 1ll * a % p),  --b;}
        else {a = int (a * 1ll * a % p),  b >>= 1;}
    return res;
}
int find_primitive_root(int p) {
    vector<int> fact;
    int phi = phi(p); // find euler totient of p.
    int n = phi;
    for (int i=2; i*i<=n; ++i)
        if (n % i == 0) {
            fact.push_back(i);
            while (n % i == 0) n /= i;
        }
    if (n > 1) fact.push_back (n);
    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (size_t i=0; i<fact.size() && ok; ++i)
            ok &= powmod (res, phi / fact[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}
```

phiFunction.h

Description: $\sum_{d|n} \phi(d) = n$, $\sum_{1 \le k \le n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$

Euler's thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod n$.

Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod p \ \forall a$. cf7d6d, 8 lines

```
const int LIM = 5000000;
int phi[LIM];

void calculatePhi() {
  rep(i,0,LIM) phi[i] = i&1 ? i : i/2;
  for (int i = 3; i < LIM; i += 2) if(phi[i] == i)
    for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;
}
```

For $a \neq$, $b \neq 0$, then $d = gcd(a, b)$ is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If (x, y) is one solution, then all solutions are given by

$$\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)}\right), \quad k \in \mathbb{Z}$$

5.4 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

5.5 Estimates

$\sum_{d|n} d = O(n \log \log n)$.

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 $n < 1e19$.

5.6 Mobius Function

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d) g(n/d)$$

Other useful formulas/forms:

$\sum_{d|n} \mu(d) = [n = 1]$ (very useful)

$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n) g(d)$

$g(n) = \sum_{1 \le m \le n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \le m \le n} \mu(m) g(\lfloor \frac{n}{m} \rfloor)$

Combinatorial (6)

6.1 Partitions and subsets

6.1.1 Partition function

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

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

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

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	~2e5	~2e8

6.1.2 Lucas' Theorem

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

6.2 General purpose numbers

6.2.1 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

StirlingFirst.h

Description: Stirling numbers of first kind. Finds $S(n,k)$ for a fixed n and for all $k = 0, 1, \dots n$. Requires NTT. Take absolute values of $S(n,k)$ for use.
Time: $\mathcal{O}(N \log N)$ 33486f, 42 lines

```
template<class T>
vector<T> power_table(T a, ll N) {
  vector<T> f(N, 1);
  for(int i=0;i<N-1;i++) f[i + 1] = a*f[i];
  return f;
}

template<class T>
vector<T> poly_taylor_shift(vector<T> a, T c) {
  ll N = a.size();
  vector<T> fac(N,1), invFac(N,1);
  for(int i=1;i<N;i++){
    fac[i]=fac[i-1]*T(i);
    invFac[i]=invFac[i-1]*inv(T(i));
  }
  for(int i=0;i<N;i++) a[i] *= fac[i];
  auto b = power_table<T>(c, N);
  for(int i=0;i<N;i++) b[i] *= invFac[i];
  reverse(all(a));
  auto f = conv(a, b);
  f.resize(N);
  reverse(all(f));
  for(int i=0;i<N;i++) f[i] *= invFac[i];
  return f;
}

template<class T>
vector<T> stirling_first(int n) {
  if (n == 0) return {1};
  if (n == 1) return {0, 1};
  auto f = stirling_first<T>(n / 2);
  auto g = poly_taylor_shift(f, T(-(n / 2)));
  f = conv(f, g);
  if (n & 1) {
    g = {-(n - 1), 1};
    f = conv(f, g);
  }
  return f;
}

// using mi = mint<998244353,5>
// auto ans = stirling_first<mi>(n);
```

6.2.2 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. $\pi(j) > j$.

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

6.2.3 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

StirlingSecond.h

Description: Stirling numbers of second kind. Finds $S(n,k)$ for a fixed n and for all $k = 0, 1, \dots n$. Requires NTT.
Time: $\mathcal{O}(N \log N)$ ad0793, 19 lines

```
template<class T>
vector<T> stirling_second(int n){
  vector<T> as(n + 1), bs(n + 1), invFac(n + 1, 1);
  for (int i = 1; i <= n; ++i)
    invFac[i] = invFac[i - 1] * inv(T(i));
  for (int i = 0; i <= n; ++i)
   as[i] = invFac[i] * T(((i & 1) ? -1 : +1));
  for (int i = 0; i <= n; ++i)
   bs[i] = invFac[i] * pow(T(i), n);

  auto ans = conv(as, bs); // conv_general in case of MOD other
       than 998244353.
  while(ans.size() > n + 1){
     ans.pop_back();
  }
  return ans;
}

// using mi = mint<998244353,5>
// auto ans = stirling_second<mi>(n);
```

6.2.4 $N!$ using Stirling approximation

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Use $exp(x)$ in C++ to calculate e^x

6.2.5 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,

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

6.2.6 Labeled unrooted trees

\# on n vertices: n^{n-2}
\# on k existing trees of size n_i: $n_1 n_2 \cdots n_k n^{k-2}$
\# with degrees d_i: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

6.2.7 Catalan numbers

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

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

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

- 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.

Graph (7)

7.1 Fundamentals

BellmanFord.h

Description: Calculates shortest paths from s in a graph that might have negative edge weights. Unreachable nodes get dist = inf; nodes reachable through negative-weight cycles get dist = -inf. Assumes $V^2 \max|w_i| < {\sim}2^{63}$.
Time: $\mathcal{O}(VE)$ 830a8f, 23 lines

```
const ll inf = LLONG_MAX;
struct Ed { int a, b, w, s() { return a < b ? a : -a; }};
struct Node { ll dist = inf; int prev = -1; };

void bellmanFord(vector<Node>& nodes, vector<Ed>& eds, int s) {
  nodes[s].dist = 0;
  sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });

  int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled vertices
  rep(i,0,lim) for (Ed ed : eds) {
    Node cur = nodes[ed.a], &dest = nodes[ed.b];
    if (abs(cur.dist) == inf) continue;
    ll d = cur.dist + ed.w;
    if (d < dest.dist) {
      dest.prev = ed.a;
      dest.dist = (i < lim-1 ? d : -inf);
    }
  }
  rep(i,0,lim) for (Ed e : eds) {
    if (nodes[e.a].dist == -inf)
      nodes[e.b].dist = -inf;
  }
}
```

FloydWarshall.h

Description: Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m, where $m[i][j]$ = `inf` if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j, `inf` if no path, or `-inf` if the path goes through a negative-weight cycle.
Time: $\mathcal{O}(N^3)$ 531245, 12 lines

```
const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
  int n = sz(m);
  rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
  rep(k,0,n) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) {
      auto newDist = max(m[i][k] + m[k][j], -inf);
      m[i][j] = min(m[i][j], newDist);
    }
  rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
}
```

SmallToLarge.h

Description: sack on tree
Time: $\mathcal{O}(N \log N)$

cdc958, 12 lines

```
void add(int v, int p, int x){
    for(auto u: g[v])if(u != p && !big[u])add(u, v, x)
}
void dfs(int v, int p, bool keep){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])if(u != p && sz[u] > mx)mx = sz[u],
        bigChild = u;
    for(auto u : g[v])if(u != p && u != bigChild)dfs(u, v, 0);
    if(bigChild != -1)dfs(bigChild, v, 1), big[bigChild] = 1;
    add(v, p, 1);//answer queries now
    if(bigChild != -1)big[bigChild] = 0;
    if(keep == 0)add(v, p, -1);
}
```

CompCon.h

Description: complementary graph connected comp
Time: $\mathcal{O}(N \log N)$

5c992c, 12 lines

```
set<int> adj[N];set<int> unused;
void dfs(int current){
    unused.erase(current);if(unused.size()==0)return;
    auto it = unused.begin();
    while(it != unused.end()){
        int W = *(it);
        if(!adj[current].count(W)){
            union_sets(current,W);dfs(W);
        }if(unused.size()==0)return;
        it = unused.upper_bound(W);
    }
}
```

7.2 Network flow

PushRelabel.h

Description: Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.
Time: $\mathcal{O}\left(V^2\sqrt{E}\right)$

0ae1d4, 48 lines

```
struct PushRelabel {
  struct Edge {
    int dest, back;
    ll f, c;
  };
  vector<vector<Edge>> g;
  vector<ll> ec;
  vector<Edge*> cur;
  vector<vi> hs; vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}

  void addEdge(int s, int t, ll cap, ll rcap=0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s])-1, 0, rcap});
  }

  void addFlow(Edge& e, ll f) {
    Edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
    e.f += f; e.c -= f; ec[e.dest] += f;
    back.f -= f; back.c += f; ec[back.dest] -= f;
  }
  ll calc(int s, int t) {
    int v = sz(g); H[s] = v; ec[t] = 1;
    vi co(2*v); co[0] = v-1;
    rep(i,0,v) cur[i] = g[i].data();
    for (Edge& e : g[s]) addFlow(e, e.c);

    for (int hi = 0;;) {
      while (hs[hi].empty()) if (!hi--) return -ec[s];
      int u = hs[hi].back(); hs[hi].pop_back();
      while (ec[u] > 0)  // discharge u
        if (cur[u] == g[u].data() + sz(g[u])) {
          H[u] = 1e9;
          for (Edge& e : g[u]) if (e.c && H[u] > H[e.dest]+1)
            H[u] = H[e.dest]+1, cur[u] = &e;
          if (++co[H[u]], !--co[hi] && hi < v)
            rep(i,0,v) if (hi < H[i] && H[i] < v)
              --co[H[i]], H[i] = v + 1;
          hi = H[u];
        } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
          addFlow(*cur[u], min(ec[u], cur[u]->c));
        else ++cur[u];
    }
  }
  bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

MinCostMaxFlow.h

Description: Min-cost max-flow. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.
Time: $\mathcal{O}(FE\log(V))$ where F is max flow. $\mathcal{O}(VE)$ for setpi.

58385b, 79 lines

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

const ll INF = numeric_limits<ll>::max() / 4;

struct MCMF {
  struct edge {
    int from, to, rev;
    ll cap, cost, flow;
  };
  int N;
  vector<vector<edge>> ed;
  vi seen;
  vector<ll> dist, pi;
  vector<edge*> par;

  MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N), par(N) {}

  void addEdge(int from, int to, ll cap, ll cost) {
    if (from == to) return;
    ed[from].push_back(edge{ from,to,sz(ed[to]),cap,cost,0 });
    ed[to].push_back(edge{ to,from,sz(ed[from])-1,0,-cost,0 });
  }

  void path(int s) {
    fill(all(seen), 0);
    fill(all(dist), INF);
    dist[s] = 0; ll di;

    __gnu_pbds::priority_queue<pair<ll, int>> q;
    vector<decltype(q)::point_iterator> its(N);
    q.push({ 0, s });

    while (!q.empty()) {
      s = q.top().second; q.pop();
      seen[s] = 1; di = dist[s] + pi[s];
      for (edge& e : ed[s]) if (!seen[e.to]) {
        ll val = di - pi[e.to] + e.cost;
        if (e.cap - e.flow > 0 && val < dist[e.to]) {
          dist[e.to] = val;
          par[e.to] = &e;
          if (its[e.to] == q.end())
            its[e.to] = q.push({ -dist[e.to], e.to });
          else
            q.modify(its[e.to], { -dist[e.to], e.to });
        }
      }
    }
    rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
  }

  pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
      ll fl = INF;
      for (edge* x = par[t]; x; x = par[x->from])
        fl = min(fl, x->cap - x->flow);

      totflow += fl;
      for (edge* x = par[t]; x; x = par[x->from]) {
        x->flow += fl;
        ed[x->to][x->rev].flow -= fl;
      }
    }
    rep(i,0,N) for(edge& e : ed[i]) totcost += e.cost * e.flow;
    return {totflow, totcost/2};
  }

  // If some costs can be negative, call this before maxflow:
  void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1; ll v;
    while (ch-- && it--)
      rep(i,0,N) if (pi[i] != INF)
        for (edge& e : ed[i]) if (e.cap)
          if ((v = pi[i] + e.cost) < pi[e.to])
            pi[e.to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
  }
};
```

Dinic.h

Description: Flow algorithm with complexity $O(VE\log U)$ where $U = \max|\text{cap}|$. $O(\min(E^{1/2}, V^{2/3})E)$ if $U = 1$; $O(\sqrt{V}E)$ for bipartite matching.

d7f0f1, 42 lines

```
struct Dinic {
  struct Edge {
    int to, rev;
    ll c, oc;
    ll flow() { return max(oc - c, 0LL); } // if you need flows
  };
  vi lvl, ptr, q;
  vector<vector<Edge>> adj;
  Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
  void addEdge(int a, int b, ll c, ll rcap = 0) {
    adj[a].push_back({b, sz(adj[b]), c, c});
    adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap});
  }
  ll dfs(int v, int t, ll f) {
    if (v == t || !f) return f;
    for (int& i = ptr[v]; i < sz(adj[v]); i++) {
      Edge& e = adj[v][i];
      if (lvl[e.to] == lvl[v] + 1)
        if (ll p = dfs(e.to, t, min(f, e.c))) {
          e.c -= p, adj[e.to][e.rev].c += p;
          return p;
        }
    }
    return 0;
  }
```

```
  ll calc(int s, int t) {
    ll flow = 0; q[0] = s;
    rep(L,0,31) do { // 'int L=30' maybe faster for random data
      lvl = ptr = vi(sz(q));
      int qi = 0, qe = lvl[s] = 1;
      while (qi < qe && !lvl[t]) {
        int v = q[qi++];
        for (Edge e : adj[v])
          if (!lvl[e.to] && e.c >> (30 - L))
            q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
      }
      while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
    } while (lvl[t]);
    return flow;
  }
  bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

MinCut.h

Description: After running max-flow, the left side of a min-cut from s to t is given by all vertices reachable from s, only traversing edges with positive residual capacity.

GlobalMinCut.h

Description: Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.
Time: $\mathcal{O}\left(V^3\right)$

8b0e19, 21 lines

```
pair<int, vi> globalMinCut(vector<vi> mat) {
  pair<int, vi> best = {INT_MAX, {}};
  int n = sz(mat);
  vector<vi> co(n);
  rep(i,0,n) co[i] = {i};
  rep(ph,1,n) {
    vi w = mat[0];
    size_t s = 0, t = 0;
    rep(it,0,n-ph) { // O(V^2) -> O(E log V) with prio. queue
      w[t] = INT_MIN;
      s = t, t = max_element(all(w)) - w.begin();
      rep(i,0,n) w[i] += mat[t][i];
    }
    best = min(best, {w[t] - mat[t][t], co[t]});
    co[s].insert(co[s].end(), all(co[t]));
    rep(i,0,n) mat[s][i] += mat[t][i];
    rep(i,0,n) mat[i][s] = mat[s][i];
    mat[0][t] = INT_MIN;
  }
  return best;
}
```

GomoryHu.h

Description: Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.
Time: $\mathcal{O}(V)$ Flow Computations

"PushRelabel.h" 0418b3, 13 lines

```
typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
  vector<Edge> tree;
  vi par(N);
  rep(i,1,N) {
    PushRelabel D(N); // Dinic also works
    for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
    tree.push_back({i, par[i], D.calc(i, par[i])});
    rep(j,i+1,N)
      if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
  }
  return tree;
}
```

7.3 Matching

hopcroftKarp.h

Description: Fast bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and *btoa* should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.
Usage: vi btoa(m, -1); hopcroftKarp(g, btoa);
Time: $\mathcal{O}\left(\sqrt{V}E\right)$

f612e4, 42 lines

```
bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) {
  if (A[a] != L) return 0;
  A[a] = -1;
  for (int b : g[a]) if (B[b] == L + 1) {
    B[b] = 0;
    if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
      return btoa[b] = a, 1;
  }
  return 0;
}

int hopcroftKarp(vector<vi>& g, vi& btoa) {
  int res = 0;
  vi A(g.size()), B(btoa.size()), cur, next;
  for (;;) {
    fill(all(A), 0);
    fill(all(B), 0);
    cur.clear();
    for (int a : btoa) if(a != -1) A[a] = -1;
    rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
    for (int lay = 1;; lay++) {
      bool islast = 0;
      next.clear();
      for (int a : cur) for (int b : g[a]) {
        if (btoa[b] == -1) {
          B[b] = lay;
          islast = 1;
        }
        else if (btoa[b] != a && !B[b]) {
          B[b] = lay;
          next.push_back(btoa[b]);
        }
      }
      if (islast) break;
      if (next.empty()) return res;
      for (int a : next) A[a] = lay;
      cur.swap(next);
    }
    rep(a,0,sz(g))
      res += dfs(a, 0, g, btoa, A, B);
  }
}
```

DFSMatching.h

Description: Simple bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and *btoa* should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.
Usage: vi btoa(m, -1); dfsMatching(g, btoa);
Time: $\mathcal{O}(VE)$

522b98, 22 lines

```
bool find(int j, vector<vi>& g, vi& btoa, vi& vis) {
  if (btoa[j] == -1) return 1;
  vis[j] = 1; int di = btoa[j];
  for (int e : g[di])
    if (!vis[e] && find(e, g, btoa, vis)) {
      btoa[e] = di;
      return 1;
    }
  return 0;
}
int dfsMatching(vector<vi>& g, vi& btoa) {
  vi vis;
  rep(i,0,sz(g)) {
    vis.assign(sz(btoa), 0);
    for (int j : g[i])
      if (find(j, g, btoa, vis)) {
        btoa[j] = i;
        break;
      }
  }
  return sz(btoa) - (int)count(all(btoa), -1);
}
```

MinimumVertexCover.h

Description: Finds a minimum vertex cover in a bipartite graph. The size is the same as the size of a maximum matching, and the complement is a maximum independent set.

"DFSMatching.h" da4196, 20 lines

```
vi cover(vector<vi>& g, int n, int m) {
  vi match(m, -1);
  int res = dfsMatching(g, match);
  vector<bool> lfound(n, true), seen(m);
  for (int it : match) if (it != -1) lfound[it] = false;
  vi q, cover;
  rep(i,0,n) if (lfound[i]) q.push_back(i);
  while (!q.empty()) {
    int i = q.back(); q.pop_back();
    lfound[i] = 1;
    for (int e : g[i]) if (!seen[e] && match[e] != -1) {
      seen[e] = true;
      q.push_back(match[e]);
    }
  }
  rep(i,0,n) if (!lfound[i]) cover.push_back(i);
  rep(i,0,m) if (seen[i]) cover.push_back(n+i);
  assert(sz(cover) == res);
  return cover;
}
```

WeightedMatching.h

Description: Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and returns (min cost, match), where L[i] is matched with R[match[i]]. Negate costs for max cost. Requires $N \leq M$.
Time: $\mathcal{O}\left(N^2M\right)$

1e0fe9, 31 lines

```
pair<int, vi> hungarian(const vector<vi> &a) {
  if (a.empty()) return {0, {}};
  int n = sz(a) + 1, m = sz(a[0]) + 1;
  vi u(n), v(m), p(m), ans(n - 1);
  rep(i,1,n) {
    p[0] = i;
    int j0 = 0; // add "dummy" worker 0
    vi dist(m, INT_MAX), pre(m, -1);
    vector<bool> done(m + 1);
    do { // dijkstra
      done[j0] = true;
      int i0 = p[j0], j1, delta = INT_MAX;
      rep(j,1,m) if (!done[j]) {
        auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
        if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
        if (dist[j] < delta) delta = dist[j], j1 = j;
      }
```

```
      rep(j,0,m) {
        if (done[j]) u[p[j]] += delta, v[j] -= delta;
        else dist[j] -= delta;
      }
      j0 = j1;
    } while (p[j0]);
    while (j0) { // update alternating path
      int j1 = pre[j0];
      p[j0] = p[j1], j0 = j1;
    }
  }
  rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
  return {-v[0], ans}; // min cost
}
```

BlossomGeneral.h

Description: Blossom matching for general graph, $match[i]$ for i
Time: $\mathcal{O}(N^3)$

1b2a6f, 51 lines

```
vector<int> Blossom(vector<vector<int>>& graph) {
  int n = graph.size(), timer = -1;
  vector<int> mate(n, -1), label(n), parent(n),
              orig(n), aux(n, -1), q;
  auto lca = [&](int x, int y) {
    for (timer++; ; swap(x, y)) {
      if (x == -1) continue;
      if (aux[x] == timer) return x;
      aux[x] = timer;
      x = (mate[x] == -1 ? -1 : orig[parent[mate[x]]]);
    }
  };
  auto blossom = [&](int v, int w, int a) {
    while (orig[v] != a) {
      parent[v] = w; w = mate[v];
      if (label[w] == 1) label[w] = 0, q.push_back(w);
      orig[v] = orig[w] = a; v = parent[w];
    }
  };
  auto augment = [&](int v) {
    while (v != -1) {
      int pv = parent[v], nv = mate[pv];
      mate[v] = pv; mate[pv] = v; v = nv;
    }
  };
  auto bfs = [&](int root) {
    fill(label.begin(), label.end(), -1);
    iota(orig.begin(), orig.end(), 0);
    q.clear();
    label[root] = 0; q.push_back(root);
    for (int i = 0; i < (int)q.size(); ++i) {
      int v = q[i];
      for (auto x : graph[v]) {
        if (label[x] == -1) {
          label[x] = 1; parent[x] = v;
          if (mate[x] == -1)
            return augment(x), 1;
          label[mate[x]] = 0; q.push_back(mate[x]);
        } else if (label[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a); blossom(v, x, a);
        }
      }
    }
    return 0;
  };
  for (int i = 0; i < n; i++)
    if (mate[i] == -1)
      bfs(i);
  return mate;
}
```

7.4 DFS algorithms

bridgecuts.h

Description: Articulation points and bridges
Time: $\mathcal{O}(N+M)$

876fc5, 46 lines

```
// Bridges
int n, timer;
vector<vector<int>> adj;
vector<bool> vis;
vector<int> tin, low;
void dfs(int v, int p = -1) {
    vis[v] = true, tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (vis[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}
void find_bridges() {
    timer = 0, vis = vector(n, false);
    tin = low = vector(n, -1);
    for (int i = 0; i < n; ++i)
        if (!vis[i]) dfs(i);
}
// Articulation points:
void dfs(int v, int p = -1) {
    vis[v] = true;
    tin[v] = low[v] = timer++;
    int chs=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (vis[to]) low[v] = min(low[v], tin[to]);
        else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++chs;
        }
    }
    if(p == -1 && chs > 1) IS_CUTPOINT(v);
}
void find_cutpoints() {
     // same as findBridges()
}
```

BridgeTree.h

Description: bridge tree
Time: $\mathcal{O}(N+E)$

700276, 21 lines

```
const int N = 200'005;const int E = 200'005;
vector<int> bridgeTree[N],g[N];//edge list representation of
     graph
int U[E],V[E],vis[N],arr[N],T,dsu[N];
bool isbridge[E]; // if i'th edge is a bridge edge or not
int adj(int u,int e) { return U[e]^V[e]^u;}
int f(int x) { return dsu[x]=(dsu[x]==x?x:f(dsu[x]));}
void merge(int a,int b) { dsu[f(a)]=f(b);}
int dfs0(int u,int edge) { //mark bridges
    vis[u]=1;arr[u]=T++;int dbe = arr[u];
    for(auto e : g[u]) {int w = adj(u,e);
        if(!vis[w])dbe = min(dbe,dfs0(w,e));
        else if(e!=edge)dbe = min(dbe,arr[w]);
    }if(dbe == arr[u] && edge!=-1)isbridge[edge]=true;
    else if(edge!=-1)merge(U[edge],V[edge]);return dbe;
}
void buildBridgeTree(int n,int m) {
    for(int i=1; i<=n; i++)dsu[i]=i;
    for(int i=1; i<=n; i++)if(!vis[i])dfs0(i,-1);
    for(int i=1; i<=m; i++)if(f(U[i])!=f(V[i]))
        bridgeTree[f(U[i])].push_back(f(V[i])),bridgeTree[f(V[i
            ])].push_back(f(U[i]));
}
```

negativecyc.h

Description: Negative cycle detection
Time: $\mathcal{O}(V \cdot E)$

996b8f, 28 lines

```
vector<array<int,3>> edges;
bool negative_cycle(int n){
       vector<int> par(n,-1);
       vector<ll> d(n,1e18);
       int x;
       d[0]=0;
       bool any;
       for(int i=0;i<n;i++){
            any = false;
            for(auto [a,b,w] : edges){
                 if(d[b]>d[a]+w){
                      d[b]=d[a]+w,par[b]=a;
                      any=true,x=b;
                 }
            }
       }
       if(!any) return false;
       for(int i=0;i<n;i++){
          x=par[x];
       }
       vector<int> cyc={x}
       for(int i=par[x];i!=x;i=par[i]){
           cyc.push_back(i);
       }
       cyc.push_back(x);
       reverse(cyc.begin(),cyc.end());
       return true;
}
```

shortestcycle.h

Description: Shortest cycle
Time: $\mathcal{O}(V \cdot (V+E))$

5bd566, 22 lines

```
vector<vector<int>> adj;
int shortest_cycle(int n){
       int ans=1e9;
       for(int i=0;i<n;i++){
           vector<int> dis(n,-1),par(n,-1);
           queue<int> q;
           q.push(i),dis[i]=0;

           while(!q.empty()){
               int v=q.front(),q.pop();
               for(auto u : adj[v]){
                    if(dis[u]==-1)
                        dis[u]=dis[v]+1,par[u]=v,q.push(u);
                    else if(par[v]!=u && par[v]!=u)
                        ans=min(ans,dis[u]+dis[v]+1);
               }
           }
       }
       if(ans==1e9)
```

```
        ans=-1;
      return ans;
}
```

2sat.h

Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a||b)\&\&(!a||c)\&\&(d||!b)\&\&\ldots$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (~x).
Usage: `TwoSat ts(number of boolean variables);`
```
ts.either(0, ~3); // Var 0 is true or var 3 is false
ts.setValue(2); // Var 2 is true
ts.atMostOne({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true
ts.solve(); // Returns true iff it is solvable
ts.values[0..N-1] holds the assigned values to the vars
```
Time: $\mathcal{O}(N+E)$, where N is the number of boolean variables, and E is the number of clauses.

5f9706, 56 lines

```
struct TwoSat {
  int N;
  vector<vi> gr;
  vi values; // 0 = false, 1 = true

  TwoSat(int n = 0) : N(n), gr(2*n) {}

  int addVar() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
  }

  void either(int f, int j) {
    f = max(2*f, -1-2*f);
    j = max(2*j, -1-2*j);
    gr[f].push_back(j^1);
    gr[j].push_back(f^1);
  }
  void setValue(int x) { either(x, x); }

  void atMostOne(const vi& li) { // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i,2,sz(li)) {
      int next = addVar();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }

  vi val, comp, z; int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x; z.push_back(i);
    for(int e : gr[i]) if (!comp[e])
      low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
      x = z.back(); z.pop_back();
      comp[x] = low;
      if (values[x>>1] == -1)
        values[x>>1] = x&1;
    } while (x != i);
    return val[i] = low;
  }

  bool solve() {
    values.assign(N, -1);
    val.assign(2*N, 0); comp = val;
    rep(i,0,2*N) if (!comp[i]) dfs(i);
    rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
    return 1;
  }
};
```

EulerWalk.h

Description: Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret.
Time: $\mathcal{O}(V+E)$

780b64, 15 lines

```
vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0) {
  int n = sz(gr);
  vi D(n), its(n), eu(nedges), ret, s = {src};
  D[src]++; // to allow Euler paths, not just cycles
  while (!s.empty()) {
    int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
    if (it == end){ ret.push_back(x); s.pop_back(); continue; }
    tie(y, e) = gr[x][it++];
    if (!eu[e]) {
      D[x]--, D[y]++;
      eu[e] = 1; s.push_back(y);
    }}
  for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
  return {ret.rbegin(), ret.rend()};
}
```

7.5 Coloring

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;
}
```

7.6 Heuristics

MaxClique.h

Description: maximal clique, MIS = complement maxClique
Time: works for $n < 120$

09ac4a, 47 lines

```
template <int N, class E> struct MaxClique {
    using B = bitset<N>;int n;
    vector<B> g, col_buf;
    vector<int> clique, now;
    struct P { int id, col, deg;};
    vector<vector<P>> rems;
    void dfs(int dps = 0) {
        if (clique.size() < now.size()) clique = now;
        auto& rem = rems[dps];
        stable_sort(rem.begin(), rem.end(), [&](P a, P b) {
             return a.deg > b.deg; });
        int max_c = 1;
        for (auto& p : rem) {
            p.col = 0;
            while ((g[p.id] & col_buf[p.col]).any()) p.col++;
            max_c = max(max_c, p.id + 1);
            col_buf[p.col].set(p.id);
        }
        for (int i = 0; i < max_c; i++) col_buf[i].reset();
        stable_sort(rem.begin(), rem.end(), [&](P a, P b) {
             return a.col < b.col; });

        while (!rem.empty()) {
            auto p = rem.back();
            if (now.size() + p.col + 1 <= clique.size()) break;
            auto& nrem = rems[dps + 1];nrem.clear();
            B bs = B();
            for (auto q : rem) {
                if (g[p.id][q.id]) {
                    nrem.push_back({q.id, -1, 0});
                    bs.set(q.id);
                }
            }
            for (auto& q : nrem) {
                q.deg = (bs & g[q.id]).count();
            }
            now.push_back(p.id);
            dfs(dps + 1);
            now.pop_back();rem.pop_back();
        }
    }
    MaxClique(vector<vector<E>> _g) : n(int(_g.size())), g(n),
        col_buf(n), rems(n + 1) {
        for (int i = 0; i < n; i++) {
            rems[0].push_back({i, -1, int(_g[i].size())});
            for (auto e : _g[i]) g[i][e.to] = 1;
        }
        dfs();
    }
};struct E { int to; };
```

7.7 Trees

LCA.h

Description: binary lifting, computes lca
Time: $\mathcal{O}(\log N)$ per query

b71f7e, 34 lines

```
const int N = 2e5+5;
int depth[N],visited[N],up[N][20];
vi adj[N];vpii bkedge;

void dfs(int v) {
    visited[v]=true;
    rep(i,1,20)if(up[v][i-1]!=-1)up[v][i] = up[up[v][i-1]][i
        -1];
```

```
    for(int x : adj[v]) {
        if(!visited[x]) {
            depth[x] = depth[up[x][0] = v]+1;
            dfs(x);
        }
        else if(x!=up[v][0] && depth[v]>depth[x])bkedge.pb({v,x
             });
    }
}

int jump(int x, int d) {
    rep(i,0,20){
        if((d >> i) & 1)
            {if(x==-1)break;x = up[x][i];}
    }return x;
}

int LCA(int a, int b) {
    if(depth[a] < depth[b]) swap(a, b);
    a = jump(a, depth[a] - depth[b]);
    if(a == b) return a;
    rrep(i,19,0){
        int aT = up[a][i], bT = up[b][i];
        if(aT != bT) a = aT, b = bT;
    }
    return up[a][0];
}
```

CompressTree.h

Description: Given a rooted tree and a subset S of nodes, compute the minimal subtree that contains all the nodes by adding all (at most $|S|-1$) pairwise LCA's and compressing edges. Returns a list of (par, orig_index) representing a tree rooted at 0. The root points to itself.
Time: $\mathcal{O}(|S|\log|S|)$

"LCA.h" 9775a0, 21 lines

```
typedef vector<pair<int, int>> vpi;
vpi compressTree(LCA& lca, const vi& subset) {
  static vi rev; rev.resize(sz(lca.time));
  vi li = subset, &T = lca.time;
  auto cmp = [&](int a, int b) { return T[a] < T[b]; };
  sort(all(li), cmp);
  int m = sz(li)-1;
  rep(i,0,m) {
    int a = li[i], b = li[i+1];
    li.push_back(lca.lca(a, b));
  }
  sort(all(li), cmp);
  li.erase(unique(all(li)), li.end());
  rep(i,0,sz(li)) rev[li[i]] = i;
  vpi ret = {pii(0, li[0])};
  rep(i,0,sz(li)-1) {
    int a = li[i], b = li[i+1];
    ret.emplace_back(rev[lca.lca(a, b)], b);
  }
  return ret;
}
```

CentroidDecomposition.h

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

e20943, 35 lines

```
const int MX = 2e5+10;
template<int SZ> struct Centroid {
    int N; vi adj[SZ]; void ae(int a, int b) { adj[a].pb(b),
         adj[b].pb(a); }
    bool done[SZ]; int sub[SZ], par[SZ]; // processed as
         centroid yet, subtree size, current par
    void dfs(int x) {
        sub[x] = 1;
        for(auto y : adj[x]) if (!done[y] && y != par[x]) {
            par[y] = x; dfs(y); sub[x] += sub[y]; }
    }
    int centroid(int x) {
        par[x] = -1; dfs(x);
        for (int sz = sub[x];;) {
            pii mx = {0,0};
            for(auto y : adj[x]) if (!done[y] && y != par[x])
                mx=max(mx,{sub[y],y});
            if (mx.ff*2 <= sz) return x;
            x = mx.ss;
        }
    }
    int cen[SZ], lev[SZ]; //cen[x] : par,lev[x] : depth
    vector<vi> dist; // dists[i][x] gives distance to ith
         ancestor in centroid tree
    void genDist(int x, int p, int lev) {
        dist[lev][x] = dist[lev][p]+1;
        for(auto y : adj[x]) if (!done[y] && y != p) genDist(y,
             x,lev);
    } // CEN = {centroid above x, label of centroid subtree}
    void gen(int CEN, int x) {
        done[x = centroid(x)] = 1; cen[x] = CEN;
        lev[x] = (CEN == -1 ? 0 : lev[CEN]+1);
        if (lev[x] >= dist.size()) dist.emplace_back(N+1,-1);
        dist[lev[x]][x] = 0;
        for(auto y : adj[x]) if (!done[y]) genDist(y,x,lev[x]);
        for(auto y : adj[x]) if (!done[y]) gen(x,y);
    }
    void init(int _N) { N = _N; gen(-1,1); } // start with
         vertex 1
};Centroid<MX> ct;
```

dominator.h

Description: check reachability separately
Time: $\mathcal{O}(M\log N)$

3f2126, 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;
        for(auto &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);
        rrep(i,co,1) {
            for(auto &j : radj[i]) sdom[i] = min(sdom[i],sdom[
                 get(j)]);
            if (i > 1) sdomChild[sdom[i]].pb(i);
            for(auto &j : sdomChild[i]) {
                int k = get(j);
                if (sdom[j] == sdom[k]) dom[j] = sdom[j];
                else dom[j] = k;
            }
            for(auto &j : child[i]) par[j] = i;
        }
        for(int i = 2;i < co+1;i++) {
            if (dom[i] != sdom[i]) dom[i] = dom[dom[i]];
            ans[rlabel[dom[i]]].pb(rlabel[i]);
        }
    }
};
```

HLD.h

Description: LCA Inputs must be in [0, mod).
Time: $\mathcal{O}(logN)$ per query

0671a7, 60 lines

```
//euler,seg,hld combined
const int MX = 2e5+5;
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;
        for(auto& y : adj[x]) {
            par[y] = x; depth[y] = depth[x]+1;
            adj[y].erase(find(be(adj[y]),x));
            dfsSz(y); sz[x] += sz[y];
            if (sz[y] > sz[adj[x][0]]) swap(y,adj[x][0]);
        }
    }
    void dfsHld (int x) {
        pos[x] = ti++; rpos.pb(x);
        for(auto 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);
    }
    void clear () {
        rep(i,0,N+1){
            par[i]=0,root[i]=0,depth[i]=0,sz[i]=0,pos[i]=0;
            adj[i].clear();
        }
        ti=0;rpos.clear();
    }
    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;
    }
    int dist (int x, int y) { // # edges on path
        return depth[x]+depth[y]-2*depth[lca(x,y)]; }
    // [u, v)
    vector<pii> ascend (int u, int v) const {
      vector<pii> res;
      while (root[u] != root[v]) {
        res.emplace_back(pos[u], pos[root[u]]);
        u = par[root[u]];
      }
      if (u != v) res.emplace_back(pos[u], pos[v] + 1);
      return res;
    }
    // (u, v]
    vector<pii> descend (int u, int v) const {
      if (u == v) return {};
```

```
        if (root[u] == root[v]) return {{pos[u] + 1, pos[v]}};
        auto res = descend(u, par[root[v]]);
        res.emplace_back(pos[root[v]], pos[v]);
        return res;
    }
};
HLD<MX,0> hl;
```

LinkCut.h

Description: link cut tree
Time: $\mathcal{O}(\log N)$ per operation

5c875f, 102 lines

```
const int MX = 2e5+5;
typedef struct snode* sn;
struct snode {
    sn p, c[2]; // parent, children
    bool flip = 0; // subtree flipped or not
    int val, sz, v, sum; // 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] = NULL; calc(); }
    friend int getSz(sn x) { return x?x->sz:0; }
    friend int getSub(sn x) { return x?x->sub:0; }
    friend int getSum(sn x) { return x?x->sum:0; }
    void prop() { // lazy prop
        if (!flip) return;
        swap(c[0],c[1]); flip = 0;
        rep(i,0,2) if (c[i]) c[i]->flip ^= 1;
    }
    void calc() { // recalc vals
        rep(i,0,2) if (c[i]) c[i]->prop();
        sz = 1+getSz(c[0])+getSz(c[1]);
        sub = 1+getSub(c[0])+getSub(c[1])+vsub;
        sum = getSum(c[0])+getSum(c[1])+v;
    }
    int dir() {
        if (!p) return -2;
        rep(i,0,2) 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 = getSz(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 getSz(c[0]); }
    int sumRoot() { access(); return getSum(this); }
    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 = getSz(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); }
};
//Usage: FOR(i,1,N+1)LCT[i]=new snode(i); link(LCT[1],LCT[2],1)
    ;
//LCT[p]->access();LCT[p]->v += x;LCT[p]->calc(); update : a[p]
    += x;
//LCT[v]->makeRoot();LCT[u]->access();int ans = LCT[u]->vsub +
    LCT[u]->v; //subtree sum of u(par v)
//LCT[u]->sumRoot() + LCT[v]->sumRoot() - 2*l->sumRoot() + l->v
    ; // u to v path sum

sn LCT[MX];
```

DirectedMST.h

Description: Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.
Time: $\mathcal{O}(E \log V)$

"../data-structures/UnionFindRollback.h" 39e620, 60 lines

```
struct Edge { int a, b; ll w; };
struct Node {
  Edge key;
  Node *l, *r;
  ll delta;
  void prop() {
    key.w += delta;
    if (l) l->delta += delta;
    if (r) r->delta += delta;
    delta = 0;
  }
  Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
  if (!a || !b) return a ?: 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, vector<Edge>& g) {
  RollbackUF uf(n);
  vector<Node*> heap(n);
  for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
  ll res = 0;
  vi seen(n, -1), path(n), par(n);
  seen[r] = r;
  vector<Edge> Q(n), in(n, {-1,-1}), comp;
  deque<tuple<int, int, vector<Edge>>> cycs;
  rep(s,0,n) {
    int u = s, qi = 0, w;
    while (seen[u] < 0) {
      if (!heap[u]) return {-1,{}};
      Edge e = heap[u]->top();
      heap[u]->delta -= e.w, pop(heap[u]);
      Q[qi] = e, path[qi++] = u, seen[u] = s;
      res += e.w, u = uf.find(e.a);
      if (seen[u] == s) {
        Node* cyc = 0;
        int end = qi, time = uf.time();
        do cyc = merge(cyc, heap[w = path[--qi]]);
        while (uf.join(u, w));
        u = uf.find(u), heap[u] = cyc, seen[u] = -1;
        cycs.push_front({u, time, {&Q[qi], &Q[end]}});
      }
    }
    rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
  }

  for (auto& [u,t,comp] : cycs) { // restore sol (optional)
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : comp) in[uf.find(e.b)] = e;
    in[uf.find(inEdge.b)] = inEdge;
  }
  rep(i,0,n) par[i] = in[i].a;
  return {res, par};
}
```

7.8 Math

7.8.1 Erdős–Gallai theorem

A simple graph with node degrees $d_1 \geq \cdots \geq d_n$ exists iff $d_1 + \cdots + d_n$ is even and for every $k = 1 \ldots n$,

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k).$$

7.8.2 Number of Spanning Trees

Create an $N \times N$ matrix `mat`, and for each edge $a \to b \in G$, do `mat[a][b]--`, `mat[b][b]++` (and `mat[b][a]--`, `mat[a][a]++` if G is undirected). Remove the ith row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

Geometry (8)

8.1 Geometric primitives

Point.h

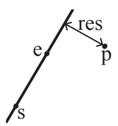
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

"Point.h" 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.

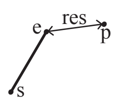


"Point.h" f6bf6b, 4 lines

```
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.



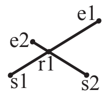
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

"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.



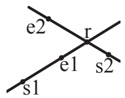
Usage: `vector<P> inter = segInter(s1,e1,s2,e2);`
`if (sz(inter)==1)`
`cout << "segments intersect at " << inter[0] << endl;`

"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 {1, 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

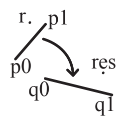
Description: Returns true iff p lies on the line segment from s to e. Use `(segDist(s,e,p)<=epsilon)` instead when using Point<double>.

"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;
}
```

linearTransformation.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();
}
```

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 * (ll)b.x) <
         make_tuple(b.t, b.half(), a.x * (ll)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

CircleIntersection.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;
}
```

CirclePolygonIntersection.h

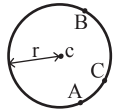
Description: Returns the area of the intersection of a circle with a ccw polygon.
Time: $\mathcal{O}(n)$

"../../content/geometry/Point.h" a1ee63, 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 circumcirle 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;
}
```

MinimumEnclosingCircle.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

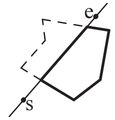
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"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:



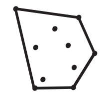
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.
Usage: `vector<P> p = ...;`
`p = polygonCut(p, P(0,0), P(1,0));`

"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;
}
```

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)
    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: • $(-1,-1)$ if no collision, • $(i,-1)$ if touching the corner i, • (i,i) if along side $(i,i+1)$, • (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 + !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;
}
```

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

ManhattanMST.h

Description: include Point.h, 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 form: (distance, src, dst). Use a standard MST algorithm on the result to find the final MST.
Time: $\mathcal{O}(N \log N)$

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);
  }
};
```

8.5 3D

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.

611f07, 8 lines

```
double sphericalDistance(double f1, double t1,
    double f2, double t2, double radius) {
  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);
}
```

8.6 Extra

HalfPlane.h

Description: include point.h

d22191, 79 lines

```
const long double eps = 1e-9, inf = 1e9;
struct Halfplane {
    // pq is the dir vector from p
    Point p, pq;
    long double angle;
    Halfplane() {}
    Halfplane(const Point& a, const Point& b) : p(a), pq(b - a)
        {
        angle = atan2l(pq.y, pq.x);
    }
    bool out(const Point& r) {
        return cross(pq, r - p) < -eps;
    }
    bool operator < (const Halfplane& e) const {
        return angle < e.angle;
    }
    //It is assumed they're never parallel.
    friend Point inter(const Halfplane& s, const Halfplane& t)
        {
        long double alpha = cross((t.p - s.p), t.pq) / cross(s.
            pq, t.pq);
        return s.p + (s.pq * alpha);
    }
};
vector<Point> hp_intersect(vector<Halfplane>& H) {
    Point box[4] = {
        Point(inf, inf),
        Point(-inf, inf),
        Point(-inf, -inf),
        Point(inf, -inf)
    };
    for(int i = 0; i<4; i++) {
        Halfplane aux(box[i], box[(i+1) % 4]);
        H.push_back(aux);
    }
    sort(H.begin(), H.end());
    deque<Halfplane> dq;
    int len = 0;
    for(int i = 0; i < int(H.size()); i++) {
        // Remove if last half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[len-1], dq[len-2]))
            ) {
            dq.pop_back();
            --len;
        }
        //Remove if first half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
            dq.pop_front();
            --len;
        }
        //Parallel half-planes
        if (len > 0 && fabsl(cross(H[i].pq, dq[len-1].pq)) <
            eps) {
            // Opposite parallel
            if (dot(H[i].pq, dq[len-1].pq) < 0.0)
                return vector<Point>();
            // Same direction half-plane: keep only the
                leftmost half-plane.
            if (H[i].out(dq[len-1].p)) {
                dq.pop_back();
                --len;
            }
            else continue;
        }
        dq.push_back(H[i]);
        ++len;
    }
    // front against the back
    while (len > 2 && dq[0].out(inter(dq[len-1], dq[len-2]))) {
        dq.pop_back();
        --len;
    }
    while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
        dq.pop_front();
        --len;
    }
    if (len < 3) return vector<Point>();
    //the convex polygon
    vector<Point> ret(len);
    for(int i = 0; i+1 < len; i++) {
        ret[i] = inter(dq[i], dq[i+1]);
    }
    ret.back() = inter(dq[len-1], dq[0]);
    return ret;
}
```

PointCount.h

Description: include Point.h

ca5d60, 51 lines

```
ll operator^(P r) const { return ll(x) * ll(r.y) - ll(y) * ll(r
    .x); }
static bool compareYX(P a, P b){ return a.y < b.y || (!(b.y < a
    .y) && a.x < b.x); }
static bool compareXY(P a, P b){ return a.x < b.x || (!(b.x < a
    .x) && a.y < b.y); }
typedef Point <ll> Vec;
using i64 = long long;
#define rep(i,n) for(int i=0; i<(int)(n); i++)

auto pointL = vector<int>(N); // bx < Ax
auto pointM = vector<int>(N); // bx = Ax
rep(i,N) rep(j,M) if(A[i].y == B[j].y){
    if(B[j].x < A[i].x) pointL[i]++;
    if(B[j].x == A[i].x) pointM[i]++;
}
auto edgeL = vector<vector<int>>(N, vector<int>(N)); // bx <
    lerp(Ax, Bx)
auto edgeM = vector<vector<int>>(N, vector<int>(N)); // bx =
    lerp(Ax, Bx)
rep(a,N){
    struct PointId { int i; int c; Vec v; };
    vector<PointId> points;
    rep(b,N) if(A[a].y < A[b].y) points.push_back({ b, 0, A[b]
        - A[a] });
    rep(b,M) if(A[a].y < B[b].y) points.push_back({ b, 1, B[b]
        - A[a] });
    rep(b,N) if(A[a].y < A[b].y) points.push_back({ b, 2, A[b]
        - A[a] });
    sort(points.begin(), points.end(), [&](const PointId& l,
        const PointId& r){
        i64 det = l.v ^ r.v;
        if(det != 0) return det < 0;
        return l.c < r.c;
    });
    int qN = points.size();
    vector<int> queryOrd(qN); rep(i,qN) queryOrd[i] = i;
    sort(queryOrd.begin(), queryOrd.end(), [&](int l, int r){
        return make_pair(points[l].v.y, points[l].c%2) <
            make_pair(points[r].v.y, points[r].c%2);
    });
    vector<int> BIT(qN);
    for(int qi=0; qi<qN; qi++){
        int q = queryOrd[qi];
        if(points[q].c == 0){
            int buf = 0;
            int p = q+1;
            while(p > 0){ buf += BIT[p-1]; p -= p & -p; }
            edgeL[a][points[q].i] = buf;
        } else if(points[q].c == 1) {
            int p = q+1;
            while(p <= qN){ BIT[p-1]++; p += p & -p; }
        } else {
            int buf = 0;
            int p = q+1;
            while(p > 0){ buf += BIT[p-1]; p -= p & -p; }
            edgeM[a][points[q].i] = buf;
        }
    }
    rep(b,N) edgeM[a][b] -= edgeL[a][b];
        }
```

Minkowski.h

5cd65e, 27 lines

```
void reorder_polygon(vector<pt> & P){
    size_t pos = 0;
    for(size_t i = 1; i < P.size(); i++){
        if(P[i].y < P[pos].y || (P[i].y == P[pos].y && P[i].x <
            P[pos].x))
            pos = i;
    }
    rotate(P.begin(), P.begin() + pos, P.end());
}
vector<pt> minkowski(vector<pt> P, vector<pt> Q){
    reorder_polygon(P);
    reorder_polygon(Q);
    P.push_back(P[0]);
    P.push_back(P[1]);
    Q.push_back(Q[0]);
    Q.push_back(Q[1]);
    vector<pt> result;
    size_t i = 0, j = 0;
    while(i < P.size() - 2 || j < Q.size() - 2){
        result.push_back(P[i] + Q[j]);
        auto cross = (P[i + 1] - P[i]).cross(Q[j + 1] - Q[j]);
```

```
        if(cross >= 0 && i < P.size() - 2)
            ++i;
        if(cross <= 0 && j < Q.size() - 2)
            ++j;
    }
    return result;
}
```

Strings (9)

Trie.h

Description: trie
Time: $\mathcal{O}(N * bits)$

8736d9, 23 lines

```
template<int SZ, int MXBIT> struct Trie {
    int nex[SZ][2], sz[SZ], num = 0; // num is last node in
        trie
    // change 2 to 26 for lowercase letters
    Trie() { memset(nex,0,sizeof nex); memset(sz,0,sizeof sz);
        }
    void ins(ll x, int a = 1) { // insert or delete
        int cur = 0; sz[cur] += a;
        rrep(i,MXBIT,0) {
            int t = (x>>i)&1;
            if (!nex[cur][t]) nex[cur][t] = ++num;
            sz[cur = nex[cur][t]] += a;
        }
    }
    ll test(ll x) { // compute max xor
        if (!sz[0]) return -INF; // no elements in trie
        int cur = 0;
        rrep(i,MXBIT,0) {
            int t = ((x>>i)&1)^1;
            if (!nex[cur][t] || !sz[nex[cur][t]]) t ^= 1;
            cur = nex[cur][t]; if (t) x ^= 1LL<<i;
        }
        return x;
    }
};
```

StringHashing.h

Description: string hash polynomial
Time: $\mathcal{O}(N)$

43d136, 24 lines

```
//0 based string hashing with closed intervals
class HashedString {
    private:
        static const long long M = 2e9+39;static const long
            long B = 9973;
        static vector<long long> pow;
        vector<long long> p_hash;
    public:
        HashedString(const string& s) : p_hash(s.size() + 1) {
            while (pow.size() < s.size()) {
                pow.push_back((pow.back() * B) % M);
            }
            p_hash[0] = 0;
            for (int i = 0; i < s.size(); i++) {
                p_hash[i + 1] = ((p_hash[i] * B) % M + s[i]) %
                    M;
            }
        }
        long long getHash(int start, int end) {
            long long raw_val = (
                p_hash[end + 1] - (p_hash[start] * pow[end -
                    start + 1])
            );
            return (raw_val % M + M) % M;
        }
};
vector<long long> HashedString::pow = {1};
```

KMP.h

Description: pi[x] computes the length of the longest prefix of s that ends at x, other than s[0...x] itself (abacaba -> 0010123). Can be used to find all occurrences of a string.
Time: $\mathcal{O}(n)$

d4375c, 16 lines

```
vi pi(const string& s) {
  vi p(sz(s));
  rep(i,1,sz(s)) {
    int g = p[i-1];
    while (g && s[i] != s[g]) g = p[g-1];
    p[i] = g + (s[i] == s[g]);
  }
  return p;
}

vi match(const string& s, const string& pat) {
  vi p = pi(pat + '\0' + s), res;
  rep(i,sz(p)-sz(s),sz(p))
    if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
  return res;
}
```

Zfunc.h

Description: z[i] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)
Time: $\mathcal{O}(n)$

ee09e2, 12 lines

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

Manacher.h

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

e7ad79, 13 lines

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

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.
Usage: `rotate(v.begin(), v.begin()+minRotation(v), v.end());`
Time: $\mathcal{O}(N)$

d07a42, 8 lines

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

SuffixArray.h

Description: Builds suffix array for a string. `sa[i]` is the starting index of the suffix which is i'th in the sorted suffix array. The returned vector is of size $n+1$, and `sa[0] = n`. The `lcp` array contains longest common prefixes for neighbouring strings in the suffix array: `lcp[i] = lcp(sa[i], sa[i-1])`, `lcp[0] = 0`. The input string must not contain any zero bytes.
Time: $\mathcal{O}(n \log n)$

38db9f, 23 lines

```
struct SuffixArray {
  vi sa, lcp;
  SuffixArray(string& s, int lim=256) { // or basic_string<int>
    int n = sz(s) + 1, k = 0, a, b;
    vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
    sa = lcp = y, iota(all(sa), 0);
    for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
      p = j, iota(all(y), n - j);
      rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
      fill(all(ws), 0);
      rep(i,0,n) ws[x[i]]++;
      rep(i,1,lim) ws[i] += ws[i - 1];
      for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
      swap(x, y), p = 1, x[sa[0]] = 0;
      rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
        (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
    }
    rep(i,1,n) rank[sa[i]] = i;
    for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
      for (k && k--, j = sa[rank[i] - 1];
          s[i + k] == s[j + k]; k++);
  }
};
```

SuffixAutomaton.h

Description: Suffix automaton.

058f57, 48 lines

```
template<size_t A_SZ = 26, char A_0 = 'a'>
struct SufAut {
    using map = array<int, A_SZ>;
    vector<int> len{0}, link{-1}, fst{-1};
    vector<bool> cln{0};
    vector<map> nx{{}};
    int l = 0;
    int push (int l, int sl, int fp, bool c, const map &adj) {
        len.push_back(l), link.push_back(sl);
        fst.push_back(fp), cln.push_back(c);
        nx.push_back(adj); return len.size() - 1;
    }
    void extend (const char c) {
        int cur = push(len[l]+1, -1, len[l], 0, {}), p = l;
        l = cur;
        while (~p and !nx[p][c - A_0])
            nx[p][c - A_0] = cur, p = link[p];
        if (p == -1) return void(link[cur] = 0);
        int q = nx[p][c - A_0];
        if (len[q] == len[p] + 1)
            return void (link[cur] = q);
        int cln = push(len[p] + 1, link[q], fst[q], true, nx[q
            ]);
        while (~p and nx[p][c - A_0] == q)
            nx[p][c - A_0] = cln, p = link[p];
```

```
        link[q] = link[cur] = cln;
    }
    int find (const string& s) {
        int u = 0;
        for (const auto c: s) {
            u = nx[u][c - A_0];
            if (!u) return -1;
        }
        return u;
    }
    vector<int> counts () {
        int n = size(len);
        vector c(n, 0);
        vector<vector<int>> inv(n);
        for (int i = 1; i < n; i++)
            inv[link[i]].push_back(i);
        auto dfs = [&](auto dfs, int u) -> void {
            c[u] = !cln[u];
            for (auto v: inv[u])
                dfs(dfs, v), c[u] += c[v];
        };
        dfs(dfs, 0); return c;
    }
};
```

AhoCorasick.h

Description: Aho-Corasick automaton, used for multiple pattern matching. Initialize with AhoCorasick ac(patterns); the automaton start node will be at index 0. find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(−, word) finds all words (up to $N\sqrt{N}$ many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. For large alphabets, split each symbol into chunks, with sentinel bits for symbol boundaries.
Time: construction takes $\mathcal{O}(26N)$, where N = sum of length of patterns. find(x) is $\mathcal{O}(N)$, where N = length of x. findAll is $\mathcal{O}(NM)$. f35677, 66 lines

```
struct AhoCorasick {
  enum {alpha = 26, first = 'A'}; // change this!
  struct Node {
    // (nmatches is optional)
    int back, next[alpha], start = -1, end = -1, nmatches = 0;
    Node(int v) { memset(next, v, sizeof(next)); }
  };
  vector<Node> N;
  vi backp;
  void insert(string& s, int j) {
    assert(!s.empty());
    int n = 0;
    for (char c : s) {
      int& m = N[n].next[c - first];
      if (m == -1) { n = m = sz(N); N.emplace_back(-1); }
      else n = m;
    }
    if (N[n].end == -1) N[n].start = j;
    backp.push_back(N[n].end);
    N[n].end = j;
    N[n].nmatches++;
  }
  AhoCorasick(vector<string>& pat) : N(1, -1) {
    rep(i,0,sz(pat)) insert(pat[i], i);
    N[0].back = sz(N);
    N.emplace_back(0);

    queue<int> q;
    for (q.push(0); !q.empty(); q.pop()) {
      int n = q.front(), prev = N[n].back;
      rep(i,0,alpha) {
        int &ed = N[n].next[i], y = N[prev].next[i];
        if (ed == -1) ed = y;
        else {
          N[ed].back = y;
          (N[ed].end == -1 ? N[ed].end : backp[N[ed].start])
            = N[y].end;
          N[ed].nmatches += N[y].nmatches;
          q.push(ed);
        }
      }
    }
  }
  vi find(string word) {
    int n = 0;
    vi res; // ll count = 0;
    for (char c : word) {
      n = N[n].next[c - first];
      res.push_back(N[n].end);
      // count += N[n].nmatches;
    }
    return res;
  }
  vector<vi> findAll(vector<string>& pat, string word) {
    vi r = find(word);
    vector<vi> res(sz(word));
    rep(i,0,sz(word)) {
      int ind = r[i];
      while (ind != -1) {
        res[i - sz(pat[ind]) + 1].push_back(ind);
        ind = backp[ind];
      }
    }
    return res;
  }
};
```

PalindromicTree.h

523eeb, 75 lines

```
/*
-> diff(v) = len(v) - len(link(v))
-> series link will lead from the vertex v to the vertex u
    corresponding
  to the maximum suffix palindrome of v which satisfies diff(v
      ) != diff(u)
-> path within series links to the root contains only O(log n)
    vertices
-> cnt contains the number of palindromic suffixes of the node
*/
struct PalindromicTree{
  struct node {
    int nxt[26], len, st, en, link, diff, slink, cnt, oc;
  };
  string s;vector<node> t;
  int sz, last;
  PalindromicTree() {}
  PalindromicTree(string _s) {
    s = _s;int n = s.size();
    t.clear();t.resize(n + 9);sz = 2, last = 2;
    t[1].len = -1, t[1].link = 1;t[2].len = 0, t[2].link = 1;
    t[1].diff = t[2].diff = 0;t[1].slink = 1;t[2].slink = 2;
  }
  int extend(int pos) {
    int cur = last, curlen = 0;
    int ch = s[pos] - 'a';
    while (1) {
      curlen = t[cur].len;
      if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos
          ]) break;
      cur = t[cur].link;
    }
    if (t[cur].nxt[ch]) {
      last = t[cur].nxt[ch];
      t[last].oc++;
      return 0;
    }
    sz++;last = sz;
    t[sz].oc = 1;t[sz].len = t[cur].len + 2;t[cur].nxt[ch] = sz
        ;
    t[sz].en = pos;t[sz].st = pos - t[sz].len + 1;
    if (t[sz].len == 1) {
      t[sz].link = 2;t[sz].cnt = 1;t[sz].diff = 1;t[sz].slink =
            2;
      return 1;
    }
    while (1) {
      cur = t[cur].link;
      curlen = t[cur].len;
      if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos
          ]) {
        t[sz].link = t[cur].nxt[ch];
        break;
      }
    }
    t[sz].cnt = 1 + t[t[sz].link].cnt;
    t[sz].diff = t[sz].len - t[t[sz].link].len;
    if (t[sz].diff == t[t[sz].link].diff) t[sz].slink = t[t[sz
        ].link].slink;
    else t[sz].slink = t[sz].link;
    return 1;
  }
  void calc_occurrences() {
    for (int i = sz; i >= 3; i--) t[t[i].link].oc += t[i].oc;
  }
  vector<array<int, 2>> minimum_partition() { //(even, odd), 1
      indexed
    int n = s.size();
    vector<array<int, 2>> ans(n + 1, {0, 0}), series_ans(n + 5,
        {0, 0});
    ans[0][1] = series_ans[2][1] = 1e9;
    for (int i = 1; i <= n; i++) {
      extend(i - 1);
      for (int k = 0; k < 2; k++) {
        ans[i][k] = 1e9;
        for (int v = last; t[v].len > 0; v = t[v].slink) {
          series_ans[v][!k] = ans[i - (t[t[v].slink].len + t[v
              ].diff)][!k];
          if (t[v].diff == t[t[v].link].diff) series_ans[v][!k]
              = min(series_ans[v][!k], series_ans[t[v].link
              ][!k]);
          ans[i][k] = min(ans[i][k], series_ans[v][!k] + 1);
        }
      }
    }
    return ans;
  }
};
```

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);
}
```

10.2 Misc. algorithms

TernarySearch.h

Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \ldots < f(i) \geq \cdots \geq f(b)$. To reverse which of the sides allows non-strict inequalities, change the < marked with (A) to <=, and reverse the loop at (B). To minimize f, change it to >, also at (B).
Usage: `int ind = ternSearch(0,n-1,[&](int i){return a[i];});`
Time: $\mathcal{O}(\log(b-a))$ 9155b4, 11 lines

```
template<class F>
int ternSearch(int a, int b, F f) {
  assert(a <= b);
  while (b - a >= 5) {
    int mid = (a + b) / 2;
    if (f(mid) < f(mid+1)) a = mid; // (A)
    else b = mid+1;
  }
  rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
  return a;
}
```

LIS.h

Description: Compute indices for the longest increasing subsequence.
Time: $\mathcal{O}(N \log N)$ 2932a0, 17 lines

```
template<class I> vi lis(const vector<I>& S) {
  if (S.empty()) return {};
  vi prev(sz(S));
  typedef pair<I, int> p;
  vector<p> res;
  rep(i,0,sz(S)) {
    // change 0 -> i for longest non-decreasing subsequence
    auto it = lower_bound(all(res), p{S[i], 0});
    if (it == res.end()) res.emplace_back(), it = res.end()-1;
    *it = {S[i], i};
    prev[i] = it == res.begin() ? 0 : (it-1)->second;
  }
  int L = sz(res), cur = res.back().second;
  vi ans(L);
  while (L--) ans[L] = cur, cur = prev[cur];
  return ans;
}
```

FastKnapsack.h

Description: Given N non-negative integer weights w and a non-negative target t, computes the maximum S <= 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: $\mathcal{O}(N^2)$ fac095, 21 lines

```
int solve() {
    int N;
    int dp[N][N], opt[N][N];
    auto C = [&](int i, int j) {
         // ... Implement cost function C.
    };
    for (int i = 0; i < N; i++) {
        opt[i][i] = i;
       //  ...  Initialize dp[i][i] according to the problem
    }
    for (int i = N-2; i >= 0; i--) {
        for (int j = i+1; j < N; j++) {
            int mn = INT_MAX, cost = C(i, j);
            for (int k = opt[i][j-1]; k <= min(j-1, opt[i+1][j
                ]); k++)
                if (mn >= dp[i][k] + dp[k+1][j] + cost)
                    opt[i][j] = k, mn = dp[i][k] + dp[k+1][j] +
                        cost;
            dp[i][j] = mn;
        }
    }
    return dp[0][N-1];
}
```

DivideAndConquerDP.h

Description: Given $a[i] = \min_{lo(i) \leq k < hi(i)}(f(i, k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R-1$.
Time: $\mathcal{O}((N + (hi - lo)) \log N)$ d38d2b, 18 lines

```
struct DP { // Modify at will:
  int lo(int ind) { return 0; }
  int hi(int ind) { return ind; }
  ll f(int ind, int k) { return dp[ind][k]; }
  void store(int ind, int k, ll v) { res[ind] = pii(k, v); }

  void rec(int L, int R, int LO, int HI) {
    if (L >= R) return;
    int mid = (L + R) >> 1;
    pair<ll, int> best(LLONG_MAX, LO);
    rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
      best = min(best, make_pair(f(mid, k), k));
    store(mid, best.second, best.first);
    rec(L, mid, LO, best.second+1);
    rec(mid+1, R, best.second, HI);
  }
  void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

SlopeTrick.h

Description: slopeTrick
Time: $\mathcal{O}(N \log N)$ 7355bd, 28 lines

```
const int MAXN = 100005;
int Z = 1;int ls, rs, lv, zp;map<int, int> MP;
void fix_left(ll s) {
    if (ls >= s) return;
    auto it = MP.begin();
    while (ls + it->second <= s) {
        ls += it->second;
        lv += ls * (next(it)->first - it->first);
        MP.erase(it++);
    }
    it->second -= s - ls;ls = s;
}
void fix_right(ll s) {
    if (rs <= s) return;
    auto it = --MP.end();
    while (rs - it->second >= s) {
        rs -= it->second;
        MP.erase(it--);
    }
    it->second += s - rs;rs = s;
}
void advance() {
    ll lo = MP.begin()->first;
    if (zp < lo) lv += ls * (zp - lo);
    else lv += Z * (zp - lo);
    ls -= Z, rs += Z;
    MP[zp] += 2 * Z;
}
```

10.4 Optimization tricks

10.4.1 Bit hacks

- `x & -x` is the least bit in `x`.
- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.
- `__builtin_popcountll(x)` returns the number of set bits in `x`
- `__builtin_clz(x)` counts the number of leading zeros in `x`
- `__builtin_ctz(x)` counts the number of trailing zeros in `x`