University of Cambridge

Balloonatics

Tautvydas Jasiūnas, Vladimir Milenković, Yui Hin Arvin Leung

2022-11-05

Contest (1)

template.cpp

23 lines

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

#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()
#define trav(u, x) for (auto &u : x)
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
template<class A,class B>auto&operator<<(ostream&o,pair<A,B>p){return o<<
     '('<<p.first<<", "<<p.second<<')';}
template<class T>auto operator<<(ostream&o,T x)->decltype(x.end(),o){o<<'
     {';int i=0;for(auto e:x)o<<(", ")+2*!i++<<e;return o<<'}';}
#ifdef DEBUG
#define debug(x...) cerr<<"["#x"]: ",[](auto...$){((cerr<<$<<"; "),...);}
     (x),cerr<<'\n'
#else
#define debug(...) {}
#endif

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

.bashrc

14 lines

```bash
alias cmp='g++ -Wall -Wconversion -Wfatal-errors -g \
  -std=gnu++17 -DDEBUG -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps =<>

test() {
        cmp $1.cpp -o $1
        for i in $1*.in; do
                echo '===TEST===';
                cat $i;
                echo '===OUT===';
                ./$1 < $i;
                echo; echo;
        done
}
```

.vimrc

6 lines

```
set cin aw ai is ts=4 sw=4 tm=50 nu noeb bg=dark ru cul
sy on   |   im jk <esc>   |   im kj <esc>   |   no ; :
" Select region and then type :Hash to hash your selection.
" Useful for verifying that there aren't mistypes.
ca Hash w !cpp -dD -P -fpreprocessed \| tr -d '[:space:]' \
 \| md5sum \| cut -c-6
```

hash.sh

3 lines

```bash
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P | tr -d '[:space:]'| md5 -r |cut -c-6
```

Mathematics (2)

2.1 Equations

In general, given an equation $Ax = b$, the solution to a variable x_i is given by

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

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

2.2 Trigonometry

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

$$\tan(v+w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$
$$\sin v + \sin w = 2 \sin \frac{v+w}{2} \cos \frac{v-w}{2}$$
$$\cos v + \cos w = 2 \cos \frac{v+w}{2} \cos \frac{v-w}{2}$$
$$(V+W)\tan(v-w)/2 = (V-W)\tan(v+w)/2$$

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

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

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

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

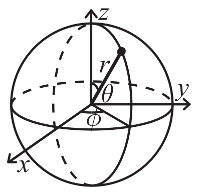
2.3.2 Quadrilaterals

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

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

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

2.3.3 Spherical coordinates



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

2.4 Derivatives/Integrals

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

Integration by parts:

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

2.5 Sums

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

2.6 Series

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

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

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

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

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

Data structures (3)

MapComparator.h

Description: example of function object (functor); keeps default behavior cmp(a, a) should always return FALSE(!)
Usage: `set<int,cmp> s; map<int,int,cmp> m;`
c31142, 1 lines

```
struct cmp{bool operator()(int l,int r)const{return l<r;}};
```

OrderStatisticTree.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change `null_type`.
Time: $\mathcal{O}(\log N)$
201e46, 16 lines

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

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

void example() {
  Tree<int> t, t2; t.insert(8);
  auto it = t.insert(10).first;
  assert(it == t.lower_bound(9));
  assert(t.order_of_key(10) == 1);
  assert(t.order_of_key(11) == 2);
  assert(*t.find_by_order(0) == 8);
  t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

HashMap.h

Description: Hash map with the same API as unordered_map, but ~3x faster. Initial capacity must be a power of 2 if provided.
Usage: `ht<int,int> h({},{},{},{},{1<<16});`
<ext/pb_ds/assoc_container.hpp> 650e7d, 10 lines

```
using namespace __gnu_pbds;
struct chash {
  const uint64_t C = ll(2e18*PI)+71; // large odd number
  const int RANDOM = rng();
  ll operator()(ll x) const {
    return __builtin_bswap64((x^RANDOM)*C); }
};
template<class K,class V> using ht = gp_hash_table<K,V,chash>;
template<class K,class V> V get(ht<K,V>& u, K x) {
  auto it = u.find(x); return it == end(u) ? 0 : it->s; }
```

SegmentTree.h

Description: Zero-indexed segtree. Queries [,).
Usage: `segtree<int, op, e> seg(vec);`
9d0568, 25 lines

```
template <class S, S (*op)(S, S), S (*e)()> struct segtree {
  private:
  int n, size, log;
  vector<S> d;
  void update(int k) {d[k] = op(d[2 * k], d[2 * k + 1]); }
  public:
  segtree(const vector <S>&v) : n(sz(v)) {
    log = (32 - __builtin_clz(n)); size = 1 << log;
    d = vector<S>(2*size, e());
    rep(i, 0, n) d[size+i] = v[i];
    for (int i = size - 1; i >= 1; --i) update(i);
  }
  void set(int pos, S val) {
    pos += size; d[pos] = val;
        rep(i, 1, log + 1) update(pos >> i);
  }
  S prod(int l, int r) {
    S sml = e(), smr = e();
    for (l += size, r += size; l < r; l >>= 1, r >>= 1) {
      if (l & 1) sml = op(sml, d[l++]);
      if (r & 1) smr = op(d[--r], smr);
    }
    return op(sml, smr);
  }
};
```

LazySegmentTree.h

Description: Segment tree with good lazy propagation templates. Queries [,). You give two monoids and one acts on the other by monoid homomorphisms.
Usage: Couting bitwise inversions on ranges:

```
struct S {mint a; int size;};
struct F {mint a, b;};
S op(S l, S r) { return S{l.a + r.a, l.size + r.size}; }
S e() { return S{0, 0}; }
S mapping(F l, S r) { return S{r.a * l.a + r.size * l.b, r.size}; }
F composition(F l, F r) { return F{r.a * l.a, r.b * l.a + l.b}; }
F id() { return F{1, 0}; }

lazy_segtree<S, op, e, F, mapping, composition, id> seg(vec);
```

Time: $\mathcal{O}(\log N)$.
58a104, 62 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 {
private:
  int n, size, log;
  vector <S> d; vector <F> lz;
  void update(int k) {d[k] = op(d[k << 1], d[k << 1 | 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(k << 1, lz[k]);
    all_apply(k << 1 | 1, lz[k]);
    lz[k] = id();
  }
  void make_pushes(int &l, int &r) {
    l += size; r += size;
    for (int i = log; i > 0; --i) {
      if (((l >> i) << i) != l) push(l >> i);
      if (((r >> i) << i) != r) push(r >> i);
    }
  }
public:
  lazy_segtree(const vector <S> &v) : n(sz(v)) {
    log = 32 - __builtin_clz(n); size = 1 << log;
    d = vector<S>(2*size, e());
    lz = vector<F>(size, id());
    rep(i, 0, n) d[size + i] = v[i];
    for (int i=size-1; i>0; --i) update(i);
  }
    void set(int p, S x) {
        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 prod(int l, int r) {
    if (l >= r) return e();
    make_pushes(l, r);
    S sml = e(), smr = e();
    for (; l < r; l >>= 1, r >>= 1) {
      if (l & 1) sml = op(sml, d[l++]);
      if (r & 1) smr = op(d[--r], smr);
    }
    return op(sml, smr);
  }
  void apply(int l, int r, F f) {
    if (l >= r) return;
    make_pushes(l, r);
    int initl = l, initr = r;
    for (; l < r; l >>= 1, r >>= 1) {
      if (l & 1) all_apply(l++, f);
      if (r & 1) all_apply(--r, f);
    }
    l = initl; r = initr;
    rep(i, 1, log+1) {
      if (((l >> i) << i) != l) update(l >> i);
      if (((r >> i) << i) != r) update((r - 1) >> i);
    }
  }
};
```

PersistentSegtree.h

Description: segment tree with historical data
Usage: figure it out
Time: $\mathcal{O}((\log N))$ per operation
83905e, 37 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 get_sum(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));
}
```

UnionFind.h

Description: Disjoint-set data structure.
Time: $\mathcal{O}(\alpha(N))$
356532, 14 lines

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

UnionFindRollback.h

Description: Disjoint-set data structure with undo. If undo is not needed, skip st, time() and rollback().
Usage: `int t = uf.time(); ...; uf.rollback(t);`
Time: $\mathcal{O}(\log(N))$ 68f1e8, 21 lines

```
struct RollbackUF {
  vi e; vector<pii> st;
  RollbackUF(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : find(e[x]); }
  int time() { return sz(st); }
  void rollback(int t) {
    for (int i = time(); i --> t;)
      e[st[i].first] = st[i].second;
    st.resize(t);
  }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back({a, e[a]});
    st.push_back({b, e[b]});
    e[a] += e[b]; e[b] = a;
    return true;
  }
};
```

Matrix.h

Description: Basic operations on square matrices.
Usage: `Matrix<int, 3> A;`
```
A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};
vector<int> vec = {1,2,3};
vec = (A^N) * vec;
```
a09d68, 26 lines

```
template<class T, int N> struct Matrix {
  typedef Matrix M;
  array<array<T, N>, N> d{};
  M operator*(const M& m) const {
    M a;
    rep(i,0,N) rep(j,0,N)
      rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];
    return a;
  }
  vector<T> operator*(const vector<T>& vec) const {
    vector<T> ret(N);
    rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
    return ret;
  }
  M operator^(ll p) const {
    assert(p >= 0);
    M a, b(*this);
    rep(i,0,N) a.d[i][i] = 1;
    while (p) {
      if (p&1) a = a*b;
      b = b*b;
      p >>= 1;
    }
    return a;
  }
};
```

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: $\mathcal{O}(\log N)$ 08a8ce, 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;
  }
};
```

Lichao.h

Description: LiChao tree for dynamic convex hull trick. can query at any point and insert lines/segments on ranges.
Time: $\mathcal{O}(n \log n)$ d36fd1, 66 lines

```
template <typename T>
struct LichaoTree {
  const T INF = std::numeric_limits<T>::max();
  struct line {
    T a, b;
    line(T a, T b) : a(a), b(b) {}
    T operator()(T x) const { return a * x + b; }
  };
  int n;
  std::vector<line> fs;
  std::vector<T> xs;

  int index(T x) const { return lower_bound(xs.begin(), xs.end(), x) - xs
       .begin(); }

  void update(T a, T b, int l, int r) {
    line g(a, b);
    for(l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if(l & 1) descend(g, l++);
      if(r & 1) descend(g, --r);
    }
  }

  void descend(line g, int i) {
    int l = i, r = i + 1;
    while(l < n) l <<= 1, r <<= 1;
    while(l < r) {
      int c = (l + r) >> 1;
      T xl = xs[l - n], xr = xs[r - 1 - n], xc = xs[c - n];
      line &f = fs[i];
      if(f(xl) <= g(xl) && f(xr) <= g(xr)) return;
      if(f(xl) >= g(xl) && f(xr) >= g(xr)) {
        f = g;
        return;
      }
      if(f(xc) > g(xc)) swap(f, g);
      if(f(xl) > g(xl))
        i = i << 1 | 0, r = c;
      else
        i = i << 1 | 1, l = c;
    }
  }

  public:
  Tree(const std::vector<T> &xs_) : xs(xs_) {
    sort(xs.begin(), xs.end());
    xs.erase(unique(xs.begin(), xs.end()), xs.end());
    n = xs.size();
    fs.assign(n << 1, line(T(0), INF));
  }

  // add f(x) = ax + b
  void add_line(T a, T b) { update(a, b, 0, n); }

  // add f(x) = ax + b (x in [xl, xr))
  void add_segment(T a, T b, T xl, T xr) {
    int l = index(xl), r = index(xr);
    update(a, b, l, r);
  }

  T get_min(T x) const {
    int i = index(x);
    T res = INF;
    for(i += n; i; i >>= 1) res = min(res, fs[i](x));
    return res;
  }
};
```

Treap.h

Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.
Time: $\mathcal{O}(\log N)$ ef60a6, 39 lines

```
struct Node {
  Node *c[2] = {0, 0};
  int prio, s = 1; // by default this just counts number of children
  Node() : prio(rand()) {}
  void recalc();
};

int sum(Node* n) { return n ? n->s: 0; }
void Node::recalc() { // push lazy propagation here
  s = sum(c[0]) + 1 + sum(c[1]);
}

Node* attach(Node *l, Node* n, Node *r){
  n->c[0] = l, n->c[1] = r;
  n->recalc();
  return n;
}

pair<Node*, Node*> split(Node* n, int k) {
  if (!n) return {};
  n->recalc();
  if (sum(n->c[0]) >= k) { // "n->val >= k" for lower_bound(k)
    auto [l, r] = split(n->c[0], k);
    return {l, attach(r, n, n->c[1])};
  } else {
    auto [l, r] = split(n->c[1], k - 1 - sum(n->c[0])); // and just "k"
    return {attach(n->c[0], n, l), r};
  }
}

Node* merge(Node* l, Node* r) {
  if (!l) return r;
  if (!r) return l;
  l->recalc(); // only needed for lazy propagation
  r->recalc();
  return l->prio > r->prio ?
    attach(l->c[0], l, merge(l->c[1], r)) :
    attach(merge(l, r->c[0]), r, r->c[1]);
}
```

FenwickTree2d.h

Description: Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j]. Requires that the elements to be updated are known in advance (call fakeUpdate() before init()).
Time: $\mathcal{O}\left(\log^2 N\right)$. (Use persistent segment trees for $\mathcal{O}(\log N)$.)
"FenwickTree.h" 527b24, 22 lines

```
struct FT2 {
  vector<vi> ys; vector<FT> ft;
  FT2(int limx) : ys(limx) {}
  void fakeUpdate(int x, int y) {
    for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
  }
  void init() {
    for (vi& v : ys) sort(all(v)), ft.emplace_back(sz(v));
  }
  int ind(int x, int y) {
    return (int)(lower_bound(all(ys[x]), y) - ys[x].begin()); }
  void update(int x, int y, ll dif) {
    for (; x < sz(ys); x |= x + 1)
      ft[x].update(ind(x, y), dif);
  }
  ll query(int x, int y) {
    ll sum = 0;
    for (; x; x &= x - 1)
      sum += ft[x-1].query(ind(x-1, y));
    return sum;
  }
};
```

BIT.h

Description: range sum queries and point updates for D dimensions
Usage: `{BIT<int,10,10>}` gives 2D BIT
Time: $\mathcal{O}\left((\log N)^D\right)$

aae4bd, 14 lines

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

RMQ.h

Description: Range Minimum Queries on an array. Returns min(V[a], V[a + 1], ... V[b - 1]) in constant time.
Usage: `RMQ rmq(values);`
`rmq.query(inclusive, exclusive);`
Time: $\mathcal{O}(|V|\log|V| + Q)$

1feea1, 16 lines

```
template<class T>
struct RMQ {
  vector<vector<T>> jmp;
  RMQ(const vector<T>& V) : jmp(1, V) {
    for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k) {
      jmp.emplace_back(sz(V) - pw * 2 + 1);
      rep(j,0,sz(jmp[k]))
        jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw]);
    }
  }
  T query(int a, int b) {
    assert(a < b); // or return inf if a == b
    int dep = 31 - __builtin_clz(b - a);
    return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
  }
};
```

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: $\mathcal{O}\left(N\sqrt{Q}\right)$

344404, 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;
}
```

Numerical (4)

4.1 Polynomials and recurrences

Polynomial.h

00c511, 17 lines

```
struct Poly {
  vector<double> a;
  double operator()(double x) const {
    double val = 0;
    for (int i = sz(a); i--;) (val *= x) += a[i];
    return val;
  }
  void diff() {
    rep(i,1,sz(a)) a[i-1] = i*a[i];
    a.pop_back();
  }
  void divroot(double x0) {
    double b = a.back(), c; a.back() = 0;
    for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
    a.pop_back();
  }
};
```

PolyRoots.h

Description: Finds the real roots to a polynomial.
Usage: `polyRoots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0`
Time: $\mathcal{O}\left(n^2\log(1/\epsilon)\right)$

"Polynomial.h" 43ba98, 23 lines

```
vector<double> polyRoots(Poly p, double xmin, double xmax) {
  if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
  vector<double> ret;
  Poly der = p;
  der.diff();
  auto dr = polyRoots(der, xmin, xmax);
  dr.push_back(xmin-1);
  dr.push_back(xmax+1);
  sort(all(dr));
  rep(i,0,sz(dr)-1) {
    double l = dr[i], h = dr[i+1];
    bool sign = p(l) > 0;
    if (sign ^ (p(h) > 0)) {
      rep(it,0,60) { // while (h - l > 1e-8)
        double m = (l + h) / 2, f = p(m);
        if ((f <= 0) ^ sign) l = m;
        else h = m;
      }
      ret.push_back((l + h) / 2);
    }
  }
  return ret;
}
```

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 $x[k] = c*\cos(k/(n-1)*\pi), k = 0\ldots n-1$.
Time: $\mathcal{O}\left(n^2\right)$

b3f87d, 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 $\le n$.
Usage: `berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}`
Time: $\mathcal{O}\left(N^2\right)$

"../number-theory/ModPow.h" 119a5a, 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;
}
```

LinearRecurrence.h

Description: Generates the k'th term of an n-order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0\ldots \ge n-1]$ and $tr[0\ldots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.
Usage: `linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number`
Time: $\mathcal{O}\left(n^2\log k\right)$

2e5ea3, 26 lines

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
  int n = sz(tr);

  auto combine = [&](Poly a, Poly b) {
    Poly res(n * 2 + 1);
    rep(i,0,n+1) rep(j,0,n+1)
      res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
    for (int i = 2 * n; i > n; --i) rep(j,0,n)
      res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
    res.resize(n + 1);
    return res;
  };

  Poly pol(n + 1), e(pol);
  pol[0] = e[1] = 1;

  for (++k; k; k /= 2) {
    if (k % 2) pol = combine(pol, e);
    e = combine(e, e);
  }

  ll res = 0;
  rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
  return res;
}
```

4.2 Optimization

GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum. The maximum error in the result is *eps*. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.
Usage: `double func(double x) { return 4+x+.3*x*x; }`
`double xmin = gss(-1000,1000,func);`
Time: $\mathcal{O}(\log((b-a)/\epsilon))$ 3279d7, 14 lines

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

HillClimbing.h

Description: Poor man's optimization for unimodal functions. 014cff, 14 lines

```
typedef array<double, 2> P;

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

Integrate.h

Description: Simple integration of a function over an interval using Simpson's rule. The error should be proportional to h^4, although in practice you will want to verify that the result is stable to desired precision when epsilon changes. 4aadf1, 7 lines

```
template<class F>
double quad(double a, double b, F f, const int n = 1000) {
  double h = (b - a) / 2 / n, v = f(a) + f(b);
  rep(i,1,n*2)
    v += f(a + i*h) * (i&1 ? 4 : 2);
  return v * h / 3;
}
```

IntegrateAdaptive.h

Description: Fast integration using an adaptive Simpson's rule.
Usage:
```
double sphereVolume = quad(-1, 1, [](double x) {
return quad(-1, 1, [&](double y) {
return quad(-1, 1, [&](double z) {
return x*x + y*y + z*z < 1; });});});
```
90b823, 15 lines

```
typedef double d;
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6

template <class F>
d rec(F& f, d a, d b, d eps, d S) {
  d c = (a + b) / 2;
  d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
  if (abs(T - S) <= 15 * eps || b - a < 1e-10)
    return T + (T - S) / 15;
  return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
}
template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
  return rec(f, a, b, eps, S(a, b));
}
```

Simplex.h

Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \le b$, $x \ge 0$. Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input vector is set to an optimal x (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that $x = 0$ is viable.
Usage: `vvd A = {{1,-1}, {-1,1}, {-1,-2}};`
`vd b = {1,1,-4}, c = {-1,-1}, x;`
`T val = LPSolver(A, b, c).solve(x);`
Time: $\mathcal{O}(NM * \#pivots)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}(2^n)$ in the general case. e23e0d, 68 lines

```
typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;

const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j

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

  LPSolver(const vvd& A, const vd& b, const vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
      rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
      rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];}
      rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
      N[n] = -1; D[m+1][n] = 1;
    }

  void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
      T *b = D[i].data(), inv2 = b[s] * inv;
      rep(j,0,n+2) b[j] -= a[j] * inv2;
      b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }

  bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
      int s = -1;
      rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
      if (D[x][s] >= -eps) return true;
      int r = -1;
      rep(i,0,m) {
        if (D[i][s] <= eps) continue;
        if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                     < MP(D[r][n+1] / D[r][s], B[r])) r = i;
      }
      if (r == -1) return false;
      pivot(r, s);
    }
  }

  T solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
      pivot(r, n);
      if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
      rep(i,0,m) if (B[i] == -1) {
        int s = 0;
        rep(j,1,n+1) ltj(D[i]);
        pivot(i, s);
      }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
  }
};
```

4.3 Matrices

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}\left(N^3\right)$ 4aa3ae, 15 lines

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

IntDeterminant.h

Description: Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.
Time: $\mathcal{O}\left(N^3\right)$ 03bfcf, 18 lines

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

SolveLinear.h

Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.
Time: $\mathcal{O}\left(n^2 m\right)$ 8cce9e, 38 lines

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

int solveLinear(vector<vd>& A, vd& b, vd& x) {
  int n = sz(A), m = sz(x), rank = 0, br, bc;
  if (n) assert(sz(A[0]) == m);
  vi col(m); iota(all(col), 0);
  rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
      rep(j,i,n) if (fabs(b[j]) > eps) return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }

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

SolveLinear2.h

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

"SolveLinear.h" 57df91, 7 lines

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

SolveLinearBinary.h

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

Time: $\mathcal{O}\left(n^2 m\right)$ 59331f, 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)
}
```

MatrixInverse.h

Description: Invert matrix A. Returns rank; result is stored in A unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.

Time: $\mathcal{O}\left(n^3\right)$ bf4fd7, 35 lines

```
int matInv(vector<vector<double>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<double>> tmp(n, vector<double>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;

  rep(i,0,n) {
    int r = i, c = i;
    rep(j,i,n) rep(k,i,n)
      if (fabs(A[j][k]) > fabs(A[r][c]))
        r = j, c = k;
    if (fabs(A[r][c]) < 1e-12) return i;
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n)
      swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
    swap(col[i], col[c]);
    double v = A[i][i];
    rep(j,i+1,n) {
      double f = A[j][i] / v;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] -= f*A[i][k];
      rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
    }
    rep(j,i+1,n) A[i][j] /= v;
    rep(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
  }

  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
  }

  rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
  return n;
}
```

Tridiagonal.h

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

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

This is useful for solving problems on the type

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

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

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

Fails if the solution is not unique.
If $|d_i| > |p_i| + |q_{i-1}|$ for all i, or $|d_i| > |p_{i-1}| + |q_i|$, or the matrix is positive definite, the algorithm is numerically stable and neither `tr` nor the check for `diag[i] == 0` is needed.

Time: $\mathcal{O}(N)$ 6ea502, 26 lines

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

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 $N = |A| + |B|$ ($\sim$1s for $N = 2^{22}$) c0608f, 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" cbfa3f, 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;
}
```

NumberTheoreticTransform.h

Description: ntt(a) computes $\hat{f}(k) = \sum_x a[x]g^{xk}$ for all k, where $g = \mathrm{root}^{(mod-1)/N}$. N must be a power of 2. Useful for convolution modulo specific nice primes of the form $2^a b + 1$, where the convolution result has size at most 2^a. For arbitrary modulo, see FFTMod. `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For manual convolution: NTT the inputs, multiply pointwise, divide by n, reverse(start+1, end), NTT back. Inputs must be in [0, mod).
Time: $\mathcal{O}(N \log N)$

"../number-theory/ModPow.h" 2e9e28, 33 lines

```
const ll mod = (119 << 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;
void ntt(vl &a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vl rt(2, 1);
  for (static int k = 2, s = 2; k < n; k *= 2, s++) {
    rt.resize(n);
    ll z[] = {1, modpow(root, mod >> s)};
    rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
  }
  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) {
      ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
      a[i + j + k] = ai - z + (z > ai ? mod : 0);
      ai += (ai + z >= mod ? z - mod : z);
    }
}
vl conv(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s), n = 1 << B;
  int inv = modpow(n, mod - 2);
  vl L(a), R(b), out(n);
  L.resize(n), R.resize(n);
  ntt(L), ntt(R);
  rep(i,0,n) out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
  ntt(out);
  return {out.begin(), out.begin() + s};
}
```

Number theory (5)

5.1 Modular arithmetic

ModularArithmetic.h

"euclid.h" 540148, 24 lines

```
ll safe_mod(ll x, ll m) {
  ll t = x%m;
  return t >= 0 ? t : t + m;
}

struct mint {
  static const ll mod = 17; // change to something else
  ll x;
  mint(ll xx) : x(safe_mod(xx, mod)) {}
  mint operator+(mint b) { return mint((x + b.x) % mod); }
  mint operator-(mint b) { return mint((x - b.x + mod) % mod); }
  mint operator*(mint b) { return mint((x * b.x) % mod); }
  mint operator/(mint b) { return *this * invert(b); }
  mint invert(mint a) {
    ll x, y, g = euclid(a.x, mod, x, y);
    assert(g == 1); return mint((x + mod) % mod);
  }
  mint operator^(ll e) {
    ll ans = 1, b = x;
    for (; e; b = b * b % mod, e /= 2)
    if (e & 1) ans = ans * b % mod;
    return ans;
  }
};
```

ModInverse.h

Description: Pre-computation of modular inverses. Assumes LIM $\leq$ mod and that mod is a prime. bd2317, 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: $\mathcal{O}(\sqrt{m})$ 054d83, 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. 1bdcb8, 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` e4489b, 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, $\mathcal{O}(\log p)$ for most p

"ModPow.h" 2bd7cb, 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 a725ac, 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" 523901, 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" 4da066, 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;
}
```

5.3 Divisibility

euclid.h

Description: Finds two integers x and y, such that $ax + by = \gcd(a, b)$. If you just need gcd, use the built in __gcd instead. If a and b are coprime, then x is the inverse of $a \pmod{b}$.

772961, 5 lines

```
ll euclid(ll a, ll b, ll &x, ll &y) {
  if (!b) return x = 1, y = 0, a;
  ll d = euclid(b, a % b, y, x);
  return y -= a/b * x, d;
}
```

Euclid2.h

Description: finds smallest $x \geq 0$ such that $L \leq Ax \pmod{P} \leq R$ 95edf8, 9 lines

```
ll cdiv(ll x, ll y) { return (x+y-1)/y; }
ll bet(ll P, ll A, ll L, ll R) {
  if (A == 0) return L == 0 ? 0 : -1;
  ll c = cdiv(L,A); if (A*c <= R) return c;
  ll B = P%A; // P = k*A+B, L <= A(x-Ky)-By <= R
  // => -R <= By % A <= -L
  auto y = bet(A,B,A-R%A,A-L%A);
  return y == -1 ? y : cdiv(L+B*y,A)+P/A*y;
}
```

CRT.h

Description: Chinese Remainder Theorem.

crt(a, m, b, n) computes x such that $x \equiv a \pmod{m}$, $x \equiv b \pmod{n}$. If $|a| < m$ and $|b| < n$, x will obey $0 \leq x < \text{lcm}(m, n)$. Assumes $mn < 2^{62}$.

Time: $\log(n)$

"euclid.h" 5e56b2, 7 lines

```
ll crt(ll a, ll m, ll b, ll n) {
  if (n > m) swap(a, b), swap(m, n);
  ll x, y, g = euclid(m, n, x, y);
  assert((a - b) % g == 0); // else no solution
  x = (b - a) % n * x % n / g * m + a;
  return x < 0 ? x + m*n/g : x;
}
```

phiFunction.h

Description: *Euler's* ϕ function is defined as $\phi(n) := \#$ of positive integers $\leq n$ that are coprime with n. $\phi(1) = 1$, p prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, m, n coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1} p_2^{k_2} ... p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1 - 1} ... (p_r - 1)p_r^{k_r - 1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$.

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

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

Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod{p}$ $\forall a$. eb1b94, 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;
}
```

5.4 Fractions

ContinuedFractions.h

Description: Given N and a real number $x \geq 0$, finds the closest rational approximation p/q with $p, q \leq N$. It will obey $|p/q - x| \leq 1/qN$.

For consecutive convergents, $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$. ($p_k/q_k$ alternates between $> x$ and $< x$.) If x is rational, y eventually becomes ∞; if x is the root of a degree 2 polynomial the a's eventually become cyclic.

Time: $\mathcal{O}(\log N)$ e5abc2, 21 lines

```
typedef double d; // for N ~ 1e7; long double for N ~ 1e9
pair<ll, ll> approximate(d x, ll N) {
  ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x;
  for (;;) {
    ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf),
       a = (ll)floor(y), b = min(a, lim),
       NP = b*P + LP, NQ = b*Q + LQ;
    if (a > b) {
      // If b > a/2, we have a semi-convergent that gives us a
      // better approximation; if b = a/2, we *may* have one.
      // Return {P, Q} here for a more canonical approximation.
      return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q)) ?
        make_pair(NP, NQ) : make_pair(P, Q);
    }
    if (abs(y = 1/(y - (d)a)) > 3*N) {
      return {NP, NQ};
    }
    LP = P; P = NP;
    LQ = Q; Q = NQ;
  }
}
```

FracBinarySearch.h

Description: Given f and N, finds the smallest fraction $p/q \in [0, 1]$ such that $f(p/q)$ is true, and $p, q \leq N$. You may want to throw an exception from f if it finds an exact solution, in which case N can be removed.

Usage: `fracBS([](Frac f) { return f.p>=3*f.q; }, 10); // {1,3}`

Time: $\mathcal{O}(\log(N))$ 9b0a9e, 25 lines

```
struct Frac { ll p, q; };

template<class F>
Frac fracBS(F f, ll N) {
  bool dir = 1, A = 1, B = 1;
  Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N]
  if (f(lo)) return lo;
  assert(f(hi));
  while (A || B) {
    ll adv = 0, step = 1; // move hi if dir, else lo
    for (int si = 0; step; (step *= 2) >>= si) {
      adv += step;
      Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
      if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
        adv -= step; si = 2;
      }
    }
    hi.p += lo.p * adv;
    hi.q += lo.q * adv;
    dir = !dir;
    swap(lo, hi);
    A = B; B = !!adv;
  }
  return dir ? hi : lo;
}
```

5.5 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.

5.6 Primes

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

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

5.7 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 $n < 1e19$.

5.8 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)

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

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

Combinatorial (6)

6.1 Permutations

6.1.1 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
n	11	12	13	14	15	16	17			
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14			
n	20	25	30	40	50	100	150	171		
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX		

IntPerm.h

Description: Permutation -> integer conversion. (Not order preserving.) Integer -> permutation can use a lookup table.

Time: $\mathcal{O}(n)$ eaf4ab, 6 lines

```
int permToInt(vi& v) {
  int use = 0, i = 0, r = 0;
  for(int x:v) r = r * ++i + __builtin_popcount(use & -(1<<x)),
    use |= 1 << x;                          // (note: minus, not ~!)
  return r;
}
```

6.1.2 Cycles

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

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

6.1.3 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1)+D(n-2)) = nD(n-1)+(-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$$

6.1.4 Burnside's lemma

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

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

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

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

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

6.2 Partitions and subsets

6.2.1 Partition function

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

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

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

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

6.2.2 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_kp^k + ... + n_1p + n_0$ and $m = m_kp^k + ... + m_1p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^k \binom{n_i}{m_i} \pmod{p}$.

6.2.3 Binomials

multinomial.h

Description: Computes $\binom{k_1+\cdots+k_n}{k_1,k_2,\ldots,k_n} = \frac{(\sum k_i)!}{k_1!k_2!...k_n!}$. e5331d, 6 lines

```
ll multinomial(vi& v) {
  ll c = 1, m = v.empty() ? 1 : v[0];
  rep(i,1,sz(v)) rep(j,0,v[i])
    c = c * ++m / (j+1);
  return c;
}
```

6.3 General purpose numbers

6.3.1 Bernoulli numbers

EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t-1}$ (FFT-able). $B[0,\ldots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \ldots]$

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^\infty f(x)dx - \sum_{k=1}^{\infty}\frac{B_k}{k!}f^{(k-1)}(m)$$

$$\approx \int_m^\infty f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles. Alternatively, number of permutations on n items with k prefix maxima. You can compute $c(n,k)$ in $O(nlog^2)$ using the second formula, D& C and FFT.

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

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

6.3.3 Eulerian numbers

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

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

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

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

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.5 Bell numbers

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

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

$$B(n) = \sum_{k=0}^{n}\binom{n}{k}\cdot B(k)$$

6.3.6 Labeled unrooted trees

# on n vertices: n^{n-2}
# on k existing trees of size n_i: $n_1n_2\cdots n_kn^{k-2}$
# with degrees d_i: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

6.3.7 Catalan numbers

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

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

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

- 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.

Catalan convolution: find the count of balanced parentheses sequences consisting of $n+k$ pairs of parentheses where the first k symbols are open brackets.

$$C^k = \frac{k+1}{n+k+1}\binom{2n+k}{n}$$

6.4 Other

DeBruijnSeq.h

Description: Recursive FKM, given alphabet $[0,k)$ constructs cyclic string of length k^n that contains every length n string as substr. 3f712e, 13 lines

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

PermGroup.h

Description: Used only once. Schreier-Sims lets you add a permutation to a group, count number of permutations in a group, and test whether a permutation is a member of a group.
Time: ? 6eb5d1, 43 lines

```
int n;
vi inv(vi v) { vi V(sz(v)); F0R(i,sz(v)) V[v[i]]=i; return V; }
vi id() { vi v(n); iota(all(v),0); return v; }
vi operator*(const vi& a, const vi& b) {
  vi c(sz(a)); F0R(i,sz(a)) c[i] = a[b[i]];
  return c; }

const int N = 15;
struct Group {
  bool flag[N];
  vi sigma[N]; // sigma[t][k] = t, sigma[t][x] = x if x > k
  vector<vi> gen;
  void clear(int p) {
    memset(flag,0,sizeof flag);
    flag[p] = 1; sigma[p] = id(); gen.clear();
  }
} g[N];
bool check(const vi& cur, int k) {
  if (!k) return 1;
  int t = cur[k];
  return g[k].flag[t] ? check(inv(g[k].sigma[t])*cur,k-1) : 0;
}
void updateX(const vi& cur, int k);
void ins(const vi& cur, int k) {
  if (check(cur,k)) return;
  g[k].gen.pb(cur);
  FOR(i,n) if (g[k].flag[i]) updateX(cur*g[k].sigma[i],k);
}
void updateX(const vi& cur, int k) {
  int t = cur[k]; // if flag, fixes k -> k
  if (g[k].flag[t]) ins(inv(g[k].sigma[t])*cur,k-1);
  else { g[k].flag[t] = 1, g[k].sigma[t] = cur;
    each(x,g[k].gen) updateX(x*cur,k); }
}
ll order(vector<vi> gen) {
  assert(sz(gen)); n = sz(gen[0]); FOR(i,n) g[i].clear(i);
  each(a,gen) ins(a,n-1); // insert perms into group one by one
  ll tot = 1;
  FOR(i,n) {
    int cnt = 0; FOR(j,i+1) cnt += g[i].flag[j];
    tot *= cnt; }
  return tot;
}
```

Graph (7)

7.1 Fundamentals

TopoSort.h

Description: Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than n – nodes reachable from cycles will not be returned.

Time: $\mathcal{O}(|V| + |E|)$ 156c0c, 14 lines

```
vi topoSort(const vector<vi>& gr) {
  vi indeg(sz(gr)), ret;
  for (auto& li : gr) for (int x : li) indeg[x]++;
  queue<int> q; // use priority_queue for lexic. largest ans.
  rep(i,0,sz(gr)) if (indeg[i] == 0) q.push(i);
  while (!q.empty()) {
    int i = q.front(); // top() for priority queue
    ret.push_back(i);
    q.pop();
    for (int x : gr[i])
      if (--indeg[x] == 0) q.push(x);
  }
  return ret;
}
```

7.2 Network flow

MinCostMaxFlow.h

Description: Min-cost max-flow. cap[i][j] != cap[j][i] is allowed; double edges are not. 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: Approximately $\mathcal{O}\left(E^2\right)$ 340364, 81 lines

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

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

struct MCMF {
  int N;
  vector<vi> ed, red;
  vector<VL> cap, flow, cost;
  vi seen;
  VL dist, pi;
  vector<pii> par;

  MCMF(int N) :
    N(N), ed(N), red(N), cap(N, VL(N)), flow(cap), cost(cap),
    seen(N), dist(N), pi(N), par(N) {}

  void addEdge(int from, int to, ll cap, ll cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
    ed[from].push_back(to);
    red[to].push_back(from);
  }

  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});

    auto relax = [&](int i, ll cap, ll cost, int dir) {
      ll val = di - pi[i] + cost;
      if (cap && val < dist[i]) {
        dist[i] = val;
        par[i] = {s, dir};
        if (its[i] == q.end()) its[i] = q.push({-dist[i], i});
        else q.modify(its[i], {-dist[i], i});
      }
    };

    while (!q.empty()) {
      s = q.top().second; q.pop();
      seen[s] = 1; di = dist[s] + pi[s];
      for (int i : ed[s]) if (!seen[i])
        relax(i, cap[s][i] - flow[s][i], cost[s][i], 1);
      for (int i : red[s]) if (!seen[i])
        relax(i, flow[i][s], -cost[i][s], 0);
    }
    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 (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
        fl = min(fl, r ? cap[p][x] - flow[p][x] : flow[x][p]);
      totflow += fl;
      for (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
        if (r) flow[p][x] += fl;
        else flow[x][p] -= fl;
    }
    rep(i,0,N) rep(j,0,N) totcost += cost[i][j] * flow[i][j];
    return {totflow, totcost};
  }

  // 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 (int to : ed[i]) if (cap[i][to])
          if ((v = pi[i] + cost[i][to]) < pi[to])
            pi[to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
  }
};
```

EdmondsKarp.h

Description: Flow algorithm with guaranteed complexity $O(VE^2)$. To get edge flow values, compare capacities before and after, and take the positive values only. bf2458, 35 lines

```
template<class T> T edmondsKarp(vector<unordered_map<int, T>>& graph, int
        source, int sink) {
  assert(source != sink);
  T flow = 0;
  vi par(sz(graph)), q = par;

  for (;;) {
    fill(all(par), -1);
    par[source] = 0;
    int ptr = 1;
    q[0] = source;

    rep(i,0,ptr) {
      int x = q[i];
      for (auto e : graph[x]) {
        if (par[e.first] == -1 && e.second > 0) {
          par[e.first] = x;
          q[ptr++] = e.first;
          if (e.first == sink) goto out;
        }
      }
    }
    return flow;
out:
    T inc = numeric_limits<T>::max();
    for (int y = sink; y != source; y = par[y])
      inc = min(inc, graph[par[y]][y]);

    flow += inc;
    for (int y = sink; y != source; y = par[y]) {
      int p = par[y];
      if ((graph[p][y] -= inc) <= 0) graph[p].erase(y);
      graph[y][p] += inc;
    }
  }
}
```

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. 00cbaf, 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: $\mathcal{O}\left(V^3\right)$ 65bba6, 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" f3fe9a, 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

DFSMatching.h

Description: Simple bipartite matching algorithm. Graph g should be given as normal adjacency list, no need to specify partitions explicitly
Time: $\mathcal{O}(VE)$ worst case but also for $n, m \leq 10^5$ in 0.5s 020391, 28 lines

```
struct matching {
  vector<vi> v;
  int n;
  vi odw, skoj;
  matching(vector <vi> &vec) : v(vec), n(sz(vec)), odw(n, 0), skoj(n, -1)
        {}
  bool dfs(int x) {
      if (odw[x]) return 0;
      odw[x] = 1;
      trav(u, v[x]) {
          if (skoj[u] == -1 || dfs(skoj[u])) {
              skoj[u] = x;
              skoj[x] = u;
              return 1;
          }
      }
      return 0;
  }
  int solve() {
    int ok = 1, res = 0;
    while (ok--) {
      fill(all(odw), 0);
      rep(i, 0, n) {
          if (skoj[i] == -1 && dfs(i)) res++, ok = 1;
      }
    }
    return res;
  }
};
```

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. Left vertices are [0,n), right: [n, n+m) (!!!!)
"DFSMatching.h" a11c58, 25 lines

```
vi cover(vector<vi>& g, int n, int m) {
  matching solver(g);
  int res = solver.solve();
  vector<bool> lfound(n, true), seen(m);
  vi q, cover;
  rep(i,0,n) {
    if (solver.skoj[i] != -1) lfound[i] = false;
  }
  rep(i,0,n) {
    if (lfound[i]) q.push_back(i);
  }
  while (!q.empty()) {
    int i = q.back(); q.pop_back();
    assert(0 <= i && i < n);
    lfound[i] = 1;
    trav(e, g[i]) if (!seen[e-n] && solver.skoj[e] != -1) {
      seen[e-n] = true;
      q.push_back(solver.skoj[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.
Time: $\mathcal{O}\left(N^2M\right)$ d09aa8, 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
}
```

GeneralMatching.h

Description: Matching for general graphs using Blossom algorithm.
Time: $\mathcal{O}(NM)$ surprisingly fast in practice 9411c8, 52 lines

```
vi Blossom(vector<vi> &graph) {
  int n = sz(graph), timer = -1;
  vi 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(all(label), -1);
    iota(all(orig), 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;
  };
  // Time halves if you start with (any) maximal matching.
  for (int i = 0; i < n; i++)
    if (mate[i] == -1)
      bfs(i);
  return mate;
}
```

7.4 DFS algorithms

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.
Usage: `scc(graph, [&](vi& v) { ... })` visits all components in reverse topological order. `comp[i]` holds the component index of a node (a component only has edges to components with lower index). `ncomps` will contain the number of components.
Time: $\mathcal{O}(E+V)$ 543453, 24 lines

```
vi val, comp, z, cont;
int Time, ncomps;
template<class G, class F> int dfs(int j, G& g, F& f) {
  int low = val[j] = ++Time, x; z.push_back(j);
  for (auto e : g[j]) if (comp[e] < 0)
    low = min(low, val[e] ?: dfs(e,g,f));

  if (low == val[j]) {
    do {
      x = z.back(); z.pop_back();
      comp[x] = ncomps;
      cont.push_back(x);
    } while (x != j);
    f(cont); cont.clear();
    ncomps++;
  }
  return val[j] = low;
}
template<class G, class F> void scc(G& g, F f) {
  int n = sz(g);
  val.assign(n, 0); comp.assign(n, -1);
  Time = ncomps = 0;
  rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
}
```

DominatorTree.h

Description: Calculates the dominator tree of a directed graph with given source. ans[i] is the immediate dominator of vertex i, ans[i] = i means vertex unreachable or source
Time: $\mathcal{O}(N\alpha)$ 13d5ab, 60 lines

```
struct dominator_tree {
  vector<basic_string<int>> g, rg, bucket;
  vi arr, par, rev, sdom, dom, dsu, label;
  int n, t;
  dominator_tree(int n) : g(n), rg(n), bucket(n), arr(n, -1),
    par(n), rev(n), sdom(n), dom(n), dsu(n), label(n), n(n), t(0) {}
  void add_edge(int u, int v) { g[u] += v; }
  void dfs(int u) {
    arr[u] = t;
    rev[t] = u;
    label[t] = sdom[t] = dsu[t] = t;
    t++;
    for (int w : g[u]) {
      if (arr[w] == -1) {
        dfs(w);
        par[arr[w]] = arr[u];
      }
      rg[arr[w]] += arr[u];
    }
  }
  int find(int u, int x=0) {
    if (u == dsu[u])
      return x ? -1 : u;
    int v = find(dsu[u], x+1);
    if (v < 0)
      return u;
    if (sdom[label[dsu[u]]] < sdom[label[u]])
      label[u] = label[dsu[u]];
    dsu[u] = v;
    return x ? v : label[u];
  }
  vi solve(int root) {
    dfs(root);
    iota(all(dom), 0);
    for (int i=t-1; i>=0; i--) {
      for (int w : rg[i])
        sdom[i] = min(sdom[i], sdom[find(w)]);
      if (i)
        bucket[sdom[i]] += i;
      for (int w : bucket[i]) {
        int v = find(w);
        if (sdom[v] == sdom[w])
          dom[w] = sdom[w];
        else
          dom[w] = v;
      }
      if (i > 1)
        dsu[i] = par[i];
    }
    for (int i=1; i<t; i++) {
      if (dom[i] != sdom[i])
        dom[i] = dom[dom[i]];
    }
    vi outside_dom(n);
    iota(all(outside_dom), 0);
    for (int i=0; i<t; i++)
      outside_dom[rev[i]] = rev[dom[i]];
    return outside_dom;
  }
};
```

BiconnectedComponents.h

Description: Finds all biconnected components in an undirected graph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle.
Usage: int eid = 0; ed.resize(N);
for each edge (a,b) {
ed[a].emplace_back(b, eid);
ed[b].emplace_back(a, eid++); }
bicomps([&](const vi& edgelist) {...});
Time: $\mathcal{O}(E+V)$ 3c178d, 33 lines

```
vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
  int me = num[at] = ++Time, e, y, top = me;
  for (auto pa : ed[at]) if (pa.second != par) {
    tie(y, e) = pa;
    if (num[y]) {
      top = min(top, num[y]);
      if (num[y] < me)
        st.push_back(e);
    } else {
      int si = sz(st);
      int up = dfs(y, e, f);
      top = min(top, up);
      if (up == me) {
        st.push_back(e);
        f(vi(st.begin() + si, st.end()));
        st.resize(si);
      }
      else if (up < me) st.push_back(e);
      else { /* e is a bridge */ }
    }
  }
  return top;
}

template<class F>
void bicomps(F f) {
  num.assign(sz(ed), 0);
  rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
}
```

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 $(a\|\|b)\&\&(!a\|\|c)\&\&(d\|\|!b)\&\&...$ 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. 86ce2c, 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: $\mathcal{O}(V+E)$ a741d8, 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: $\mathcal{O}(NM)$ 555013, 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

MaximalCliques.h

Description: Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.
Time: $\mathcal{O}\left(3^{n/3}\right)$, much faster for sparse graphs

b394f7, 12 lines

```
typedef bitset<128> B;
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R={}) {
  if (!P.any()) { if (!X.any()) f(R); return; }
  auto q = (P | X)._Find_first();
  auto cands = P & ~eds[q];
  rep(i,0,sz(eds)) if (cands[i]) {
    R[i] = 1;
    cliques(eds, f, P & eds[i], X & eds[i], R);
    R[i] = P[i] = 0; X[i] = 1;
  }
}
```

MaximumClique.h

Description: Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph.
Time: Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs.

d3b15f, 49 lines

```
typedef vector<bitset<200>> vb;
struct Maxclique {
  double limit=0.025, pk=0;
  struct Vertex { int i, d=0; };
  typedef vector<Vertex> vv;
  vb e;
  vv V;
  vector<vi> C;
  vi qmax, q, S, old;
  void init(vv& r) {
    for (auto& v : r) v.d = 0;
    for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
    sort(all(r), [](auto a, auto b) { return a.d > b.d; });
    int mxD = r[0].d;
    rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
  }
  void expand(vv& R, int lev = 1) {
    S[lev] += S[lev - 1] - old[lev];
    old[lev] = S[lev - 1];
    while (sz(R)) {
      if (sz(q) + R.back().d <= sz(qmax)) return;
      q.push_back(R.back().i);
      vv T;
      for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
      if (sz(T)) {
        if (S[lev]++ / ++pk < limit) init(T);
        int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
        C[1].clear(), C[2].clear();
        for (auto v : T) {
          int k = 1;
          auto f = [&](int i) { return e[v.i][i]; };
          while (any_of(all(C[k]), f)) k++;
          if (k > mxk) mxk = k, C[mxk + 1].clear();
          if (k < mnk) T[j++].i = v.i;
          C[k].push_back(v.i);
        }
        if (j > 0) T[j - 1].d = 0;
        rep(k,mnk,mxk + 1) for (int i : C[k])
          T[j].i = i, T[j++].d = k;
        expand(T, lev + 1);
      } else if (sz(q) > sz(qmax)) qmax = q;
      q.pop_back(), R.pop_back();
    }
  }
  vi maxClique() { init(V), expand(V); return qmax; }
  Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S) {
    rep(i,0,sz(e)) V.push_back({i});
  }
};
```

MaximumIndependentSet.h

Description: To obtain a maximum independent set of a graph, find a max clique of the complement. If the graph is bipartite, see MinimumVertexCover.

7.7 Trees

BinaryLifting.h

Description: Calculate power of two jumps in a tree, to support fast upward jumps and LCAs. Assumes the root node points to itself.
Time: construction $\mathcal{O}(N \log N)$, queries $\mathcal{O}(\log N)$

86af22, 25 lines

```
vector<vi> treeJump(vi& P){
  int on = 1, d = 1;
  while(on < sz(P)) on *= 2, d++;
  vector<vi> jmp(d, P);
  rep(i,1,d) rep(j,0,sz(P))
    jmp[i][j] = jmp[i-1][jmp[i-1][j]];
  return jmp;
}

int jmp(vector<vi>& tbl, int nod, int steps){
  rep(i,0,sz(tbl))
    if(steps&(1<<i)) nod = tbl[i][nod];
  return nod;
}

int lca(vector<vi>& tbl, vi& depth, int a, int b) {
  if (depth[a] < depth[b]) swap(a, b);
  a = jmp(tbl, a, depth[a] - depth[b]);
  if (a == b) return a;
  for (int i = sz(tbl); i--;) {
    int c = tbl[i][a], d = tbl[i][b];
    if (c != d) a = c, b = d;
  }
  return tbl[0][a];
}
```

LCA.h

Description: Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.
Time: $\mathcal{O}(N \log N + Q)$

"../data-structures/RMQ.h" 308eb1, 19 lines

```
struct LCA {
  int T = 0;
  vi time, path, ret;
  RMQ<int> rmq;
  LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C,0,-1), ret)) {}
  void dfs(vector<vi>& C, int v, int par) {
    time[v] = T++;
    for (int y : C[v]) if (y != par) {
      path.push_back(v), ret.push_back(time[v]);
      dfs(C, y, v);
    }
  }
  int lca(int a, int b) {
    if (a == b) return a;
    tie(a, b) = minmax(time[a], time[b]);
    return path[rmq.query(a, b)];
  }
  int dist(int a, int b){return time[a] + time[b] - 2*time[lca(a,b)];}
};
```

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, node) representing a tree. No index reordering occurs. The root points to itself.
Time: $\mathcal{O}(|S| \log |S|)$

"LCA.h" 981aab, 19 lines

```
typedef vector<pair<int, int>> vpi;
vpi compressTree(LCA& lca, const vi& subset) {
  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());
  vpi ret = {{li[0], li[0]}};
  rep(i,0,sz(li)-1) {
    int a = li[i], b = li[i+1];
    ret.emplace_back(lca.lca(a, b), b);
  }
  return ret;
}
```

Centroid.h

Description: Centroid decomposition template.

882f50, 73 lines

```
// override calc() to do what you need
// node indexes are preserved throughout invocations
// if something is protected, it is safe to use it in calc()
struct base_centroids {
    private:
    int n; //graph size
    vector <vi> v; //0-based
    vector <bool> odw, gold;
    vi sub, maxsub;
    protected:
    vi par; //current array of nodes'parents. par[root]= -1.
    vi get_subtrees(vi &pre) { //helper fn which finds ranges [, ) of
        root's subtrees.
        vi res = {};
        rep(i, 0, sz(pre)) {
            if (par[pre[i]] == pre[0]) res.push_back(i);
        }
        res.push_back(sz(pre));
        return res;
    }
    //calculate answers for the current centroid(root). Nodes given in
        PREORDER
    virtual void calc(int root, vi &nodes) = 0;
    private:
    void prep(int x, vi &nodes) {
        odw[x] = 1; sub[x] = 1;
        nodes.push_back(x);
        trav(u, v[x]) {
            if (!gold[u] && !odw[u]) {
                prep(u, nodes);
                sub[x] += sub[u];
                maxsub[x] = max(maxsub[x], sub[u]);
            }
        }
    }
    void cendfs(int x, int &PRE, vi &pre) {
        odw[x] = 1; pre[PRE++] = x;
        trav(u, v[x]) {
            if (!odw[u] && !gold[u]) {
                par[u] = x;
                cendfs(u, PRE, pre);
            }
        }
    }
    public:
    base_centroids(int N, vector <vi> graph) : n(N), v(graph), odw(n,
         false), gold(n, false), sub(n, 0), maxsub(n, 0), par(n, -1) {
    };
    void solve(int start=0) {
        vector <int> comp;
        prep(start, comp);
        int N = sz(comp), cen = -1;
        trav(node, comp) {
            maxsub[node] = max(maxsub[node], N - sub[node]);
            if (maxsub[node] <= N / 2) cen = node;
            odw[node] = 0, sub[node] = 0, maxsub[node] = 0;
        }
        int PRE = 0;
        vi pre(N, 0);
        par[cen] = -1;
        cendfs(cen, PRE, pre);
        calc(cen, pre);
        trav(node, comp) odw[node] = 0, par[node] = -1;
        gold[cen] = 1;
        trav(u, v[cen]) {
            if (!gold[u]) solve(u);
        }
    }
};

struct centroids : base_centroids {
    centroids(int N, vector <vi> graph) : base_centroids(N, graph) {}
    void calc(int root, vi &nodes) {
        trav(node, nodes) cerr << node << "\n";
    }
};
```

HLD.h

Description: Handles subtree and path queries simultaneously in one lazy_segtree. Each subtree is 1 segment, while path is $O(\log N)$ segments in the tree. $VALS\_EDGES$ being true means that values are stored in the edges, as opposed to the nodes. All values are initialized to the segtree default.
Time: $\mathcal{O}\left((\log N)^2\right)$, one logarithm for subtrees.

"../data-structures/LazySegmentTree.h" 163476, 56 lines

```
template <bool VALS_EDGES,
class S, S(*op)(S, S), S(*e)(),
class F, S (*mapping)(F, S), F(*composition)(F, F), F (*id)()>
struct HLD {
  int N, tim = 0;
  vector <vi> adj;
  vi par, siz, depth, rt, pos;
  lazy_segtree<S, op, e, F, mapping, composition, id> tree;
  HLD(vector <vi> adj_, int root=0) :
    N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1), depth(N),
    rt(N,root), pos(N), tree(vector<S>(N, e())) {dfsSz(root), dfsHld(root
         );}
  void dfsSz(int v) {
    if (par[v] != -1) adj[v].erase(find(all(adj[v]), par[v]));
    trav(u, adj[v]){
      par[u] = v, depth[u] = depth[v] + 1;
      dfsSz(u);
      siz[v] += siz[u];
      if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
    }
  }
  void dfsHld(int v) {
    pos[v] = tim++;
    trav(u, adj[v]){
      rt[u] = (u == adj[v][0] ? rt[v] : u);
      dfsHld(u);
    }
  }
  template<class B> int process(int u, int v, B query) {
    for (; rt[u] != rt[v]; v = par[rt[v]]) {
      if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
      query(pos[rt[v]], pos[v] + 1);
    }
    if (depth[u] > depth[v]) swap(u, v);
    query(pos[u] + VALS_EDGES, pos[v] + 1);
    return u;
  }
  int lca(int u, int v) {
    return process(u, v, [](int l, int r) {});
  }
  void path_apply(int u, int v, F func) {
    process(u, v, [&](int l, int r) {tree.apply(l, r, func); });
  }
  S path_prod(int u, int v) {
    S res = e();
    process(u, v, [&](int l, int r) {
      res = op(res, tree.prod(l, r));
    });
    return res;
  }
  void subtree_apply(int v, F func) {
    tree.apply(pos[v] + VALS_EDGES, pos[v] + siz[v], func);
  }
  S subtree_prod(int v) {
    return tree.prod(pos[v] + VALS_EDGES, pos[v] + siz[v]);
  }
};
```

LinkCutTree.h

Description: A dynamic data structure for rooted trees. 1-based(!!!) Path queries need commutativity + neutral element. Subtree queries need that + existence of inverse elements Lazy propagation is possible, too.
Time: All operations take amortized $\mathcal{O}(\log N)$.

f734dc, 91 lines

```
template<class T, T(*op)(T, T), T(*inv)(T), T(*e)()>
struct LCT {
  struct node {
    int ch[2] = {0, 0}, p = 0;
    T self = e(), path = e();        // Path aggregates
    T sub = e(), vir = e();          // Subtree aggregates
    bool flip = 0;                       // Lazy tags
  };
  vector<node> t;
  LCT(int n) : t(n + 1) {}
  void push(int x) {
    if (!x || !t[x].flip) return;
    int l = t[x].ch[0], r = t[x].ch[1];
    t[l].flip ^= 1, t[r].flip ^= 1;
    swap(t[x].ch[0], t[x].ch[1]);
    t[x].flip = 0;
  }
  void pull(int x) {
    int l = t[x].ch[0], r = t[x].ch[1]; push(l); push(r);

    t[x].path = op(op(t[l].path, t[x].self), t[r].path);
    t[x].sub = op(op(op(t[x].vir, t[l].sub), t[r].sub), t[x].self);
  }
  void set(int x, int d, int y) {
    t[x].ch[d] = y; t[y].p = x; pull(x);
  }
  void splay(int x) {
    auto dir = [&](int x) {
      int p = t[x].p; if (!p) return -1;
      return t[p].ch[0] == x ? 0 : t[p].ch[1] == x ? 1 : -1;
    };
    auto rotate = [&](int x) {
      int y = t[x].p, z = t[y].p, dx = dir(x), dy = dir(y);
      set(y, dx, t[x].ch[!dx]);
      set(x, !dx, y);
      if (~dy) set(z, dy, x);
      t[x].p = z;
    };
    for (push(x); ~dir(x); ) {
      int y = t[x].p, z = t[y].p;
      push(z); push(y); push(x);
      int dx = dir(x), dy = dir(y);
      if (~dy) rotate(dx != dy ? x : y);
      rotate(x);
    }
  }
  int access(int x) {
    int u = x, v = 0;
    for (; u; v = u, u = t[u].p) {
      splay(u);
      int& ov = t[u].ch[1];
      t[u].vir = op(t[u].vir, t[ov].sub);
      t[u].vir = op(t[u].vir, inv(t[v].sub));
      ov = v; pull(u);
    }
    return splay(x), v;
  }
  void reroot(int x) {
    access(x); t[x].flip ^= 1; push(x);
  }
  void link(int u, int v) {
    reroot(u); access(v);
    t[v].vir = op(t[v].vir, t[u].sub);
    t[u].p = v; pull(v);
  }
  void cut(int u, int v) {
    reroot(u); access(v);
    t[v].ch[0] = t[u].p = 0; pull(v);
  }
  // Rooted tree LCA. Returns 0 if u and v arent connected.
  int lca(int u, int v) {
    if (u == v) return u;
    access(u); int ret = access(v);
    return t[u].p ? ret : 0;
  }
  // Query subtree of u where v is outside the subtree.
  T subtree_prod(int u, int v) {
    reroot(v); access(u); return op(t[u].vir, t[u].self);
  }
  // Query path [u..v]
  T path_prod(int u, int v) {
    reroot(u); access(v); return t[v].path;
  }
  // Update vertex u with value v
  void set(int u, T v) {
    access(u); t[u].self = v; pull(u);
  }
  T get(int u) {
    return t[u].self;
  }
};
```

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" 68090b, 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 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).

7.8.2 Erdős–Gallai theorem

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

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

Geometry (8)

8.1 Geometric primitives

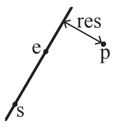
Point.h

Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)
710d96, 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.

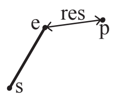


"Point.h" 57b36e, 4 lines

```
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.



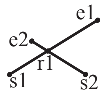
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

"Point.h" 9bf8f2, 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.



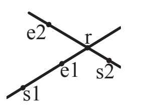
Usage: `vector<P> inter = segInter(s1,e1,s2,e2);`
`if (sz(inter)==1)`
`cout << "segments intersect at " << inter[0] << endl;`

"Point.h", "OnSegment.h" bf2daf, 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" 9e71e4, 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 ⇔ 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" b849e9, 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>.

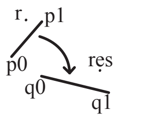
"Point.h" 63c809, 3 lines

```
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" 0976b6, 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`

4acf15, 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" 61a56b, 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" fd2a67, 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

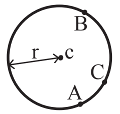
Description: Returns the area of the intersection of a circle with a ccw polygon.
Time: $\mathcal{O}(n)$

"../../content/geometry/Point.h" 2fa642, 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" e575d1, 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" 06500a, 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" e7f303, 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" fa890c, 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

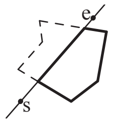
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"Point.h" aa22b5, 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:



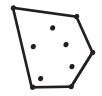
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.
Usage: vector<P> p = ...;
p = polygonCut(p, P(0,0), P(1,0));

"Point.h", "lineIntersection.h" d7b840, 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" d37f1a, 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).

"Point.h" a428f7, 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" 1a9c2e, 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" 5e9944, 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" ac1f6b, 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;
}
```

kdTree.h

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

"Point.h" 832f36, 63 lines

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

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

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

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

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

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

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

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

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

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

FastDelaunay.h

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

"Point.h" 767b99, 88 lines

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

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

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  H = r->o; r->r()->r() = r;
  rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
  r->p = orig; r->F() = dest;
  return r;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

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

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

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

vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  Q e = rec(pts).first;
  vector<Q> q = {e};
  int qi = 0;
  while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  return pts;
}
```

8.5 3D

PolyhedronVolume.h

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

b4a945, 6 lines

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

Point3D.h

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

8d3c78, 32 lines

```
template<class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z); }
  bool operator==(R p) const {
    return tie(x, y, z) == tie(p.x, p.y, p.z); }
  P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
  P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
  P operator*(T d) const { return P(x*d, y*d, z*d); }
  P operator/(T d) const { return P(x/d, y/d, z/d); }
  T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
  P cross(R p) const {
    return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
  }
  T dist2() const { return x*x + y*y + z*z; }
  double dist() const { return sqrt((double)dist2()); }
  //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
  double phi() const { return atan2(y, x); }
  //Zenith angle (latitude) to the z-axis in interval [0, pi]
  double theta() const { return atan2(sqrt(x*x+y*y),z); }
  P unit() const { return *this/(T)dist(); } //makes dist()=1
  //returns unit vector normal to *this and p
  P normal(P p) const { return cross(p).unit(); }
  //returns point rotated 'angle' radians ccw around axis
  P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.unit();
    return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
  }
};
```

3dHull.h

Description: Computes all faces of the 3-dimension hull of a point set. *No four points must be coplanar*, or else random results will be returned. All faces will point outwards.
Time: $\mathcal{O}\left(n^2\right)$

"Point3D.h" ec033c, 49 lines

```
typedef Point3D<double> P3;

struct PR {
  void ins(int x) { (a == -1 ? a : b) = x; }
  void rem(int x) { (a == x ? a : b) = -1; }
  int cnt() { return (a != -1) + (b != -1); }
  int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
  assert(sz(A) >= 4);
  vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
  vector<F> FS;
  auto mf = [&](int i, int j, int k, int l) {
    P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
    if (q.dot(A[l]) > q.dot(A[i]))
      q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
    FS.push_back(f);
  };
  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);

  rep(i,4,sz(A)) {
    rep(j,0,sz(FS)) {
      F f = FS[j];
      if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
        E(a,b).rem(f.c);
        E(a,c).rem(f.b);
        E(b,c).rem(f.a);
        swap(FS[j--], FS.back());
        FS.pop_back();
      }
    }
    int nw = sz(FS);
    rep(j,0,nw) {
      F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
      C(a, b, c); C(a, c, b); C(b, c, a);
    }
  }
  for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
    A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
  return FS;
};
```

sphericalDistance.h

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

76f0ab, 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);
}
```

Strings (9)

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: $\mathcal{O}(n)$

e441e1, 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[x] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)
Time: $\mathcal{O}(n)$

5128db, 12 lines

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

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: $\mathcal{O}(N)$

339ba7, 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: $\mathcal{O}(N)$

6c6480, 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. If you don't need get_lcp(a,b) then comment out RMQ lines.
Time: $\mathcal{O}(n \log n)$

b2e9a3, 35 lines

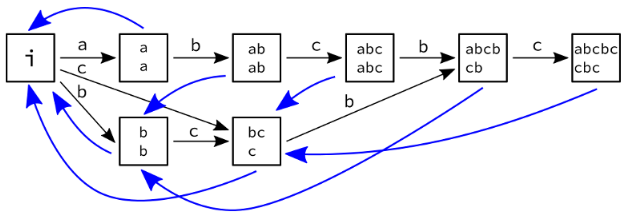
```
struct SuffixArray {
  vi sa, lcp, rank;
  int N;
  RMQ<int> *rmq;
  SuffixArray(string& s, int lim=256) : N(sz(s)){
    int n = sz(s) + 1, k = 0, a, b;
    vi x(all(s)+1), y(n), ws(max(n, lim));
    rank.resize(n);
    sa = lcp = y, iota(all(sa), 0);
    for (int j = 0, p = 0; p < n; j = max(1, 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++);

    rmq = new RMQ(lcp);
  }
  int get_lcp(int a,int b) {//lcp of suffixes starting at a,b
    if (max(a,b) >= N) return 0;
    if (a == b) return N-a;
    int t0 = rank[a] + 1, t1 = rank[b] + 1;
    if (t0 > t1) swap(t0,t1);
    return rmq->query(t0, t1);
  }
};
```

RunEnumerate.h

Description: Find all (i, p) such that `s.substr(i,p) == s.substr(i+p,p)`. No two intervals with the same period intersect or touch. Also look at comments below.
Time: $\mathcal{O}(N \log N)$

"SuffixArray.h" ff7b50, 14 lines

```
vector<array<int,3>> solve(string s) {
  int N = sz(s); SuffixArray A(s);
  reverse(all(s));
  SuffixArray B(s);
  vector<array<int,3>> runs;
  for (int p = 1; 2*p <= N; ++p) { // do in O(N/p) for period p
    for (int i = 0, lst = -1; i+p <= N; i += p) {
      int l = i-B.get_lcp(N-i-p,N-i), r = i-p+A.get_lcp(i,i+p);
      if (l > r || l == lst) continue;
      runs.pb({lst = l,r,p}); // for each i in [l,r],
    } // s.substr(i,p) == s.substr(i+p,p)
  }
  return runs;
}
```



SuffixAutomaton.h

Description: Suffix automaton. Constructs a DAG efficiently maintaining equivalence classes of string occurrences. LOOK AT THE PICTURE!!! Each distinct string is some path through the automaton. Each occurrence of string w is a path from its node to some terminal node. At most 2N states and 3N edges in the whole automaton. Many things done by DP, add calculations in init()
Time: $\mathcal{O}(n\alpha)$ or $\mathcal{O}(nlog\alpha)$ If you need suffix tree, use suffix links in SA for reversed string.

<bits/stdc++.h> 45cb4c, 92 lines

```
using namespace std;
struct state {
    int len, link;
    map<char, int> next;
    state() : len(0), link(-1) {}
};
struct suffix_automaton {
  string input;
  vector <state> st;
  int last, size;
  vi top; vector<ll> cnt; vector<bool> odw;
  suffix_automaton(const string &s) : input(s), last(0), size(1) {
    st.push_back(state());
    trav(c, s) add_letter(c);
    init();
  }
  void dfs(int x) {
    odw[x] = 1;
    for (auto [lett, node] : st[x].next)
      if (!odw[node]) dfs(node);
    top.push_back(x);
  }
  void init() {
    int p = last;
    cnt.resize(size, 0); odw.resize(size, 0);
    while (p > 0) cnt[p]++, p = st[p].link;
    dfs(0);
    reverse(all(top)); assert(top[0] == 0);
    for (int i = sz(top)-1; i>0; --i) {
      for (auto [lett, node] : st[top[i]].next) {
        cnt[top[i]] += cnt[node]; //dp calculations here
      }
    }
  }
  void add_letter(char c) {
    st.push_back(state());
    int cur = size++;
    st[cur].len = st[last].len + 1;
    int p = last;
    while (p != -1 && !st[p].next.count(c)) {
      st[p].next[c] = cur;
      p = st[p].link;
    }
    if (p == -1) {
      st[cur].link = 0;
    } else {
      int q = st[p].next[c];
      if (st[p].len + 1 == st[q].len) {
        st[cur].link = q;
      } else {
        st.push_back(state());
        int clone = size++;
        st[clone].len = st[p].len + 1;
        st[clone].next = st[q].next;
        st[clone].link = st[q].link;
        while (p != -1 && st[p].next[c] == q) {
          st[p].next[c] = clone;
          p = st[p].link;
        }
        st[q].link = st[cur].link = clone;
      }
    }
    last = cur;
  }
  int search(const string &s) {
    int q = 0;
    trav(c, s) {
      if (st[q].next.find(c) == st[q].next.end()) return 0;
      q = st[q].next[c];
    }
    return q;
  }
  ll count_occs(string &s) { return cnt[search(s)]; }
  string lcs(const string &T) {
    int v = 0, l = 0, best = 0, bestpos = 0;
    rep(i, 0, sz(T)) {
      while (v && !st[v].next.count(T[i])) {
        v = st[v].link;
        l = st[v].len;
      }
      if (st[v].next.count(T[i])) {
        v = st [v].next[T[i]];
        l++;
      }
      if (l > best) {
        best = l;
        bestpos = i;
      }
    }
    return T.substr(bestpos - best + 1, best);
  }
};
```

Hashing.h

Description: Self-explanatory methods for string hashing. f0e2ee, 44 lines

```
// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^64).
// "typedef ull H;" instead if you think test data is random,
// or work mod 10^9+7 if the Birthday paradox is not a problem.
struct H {
  typedef uint64_t ull;
  ull x; H(ull x=0) : x(x) {}
#define OP(O,A,B) H operator O(H o) { ull r = x; asm \
  (A "addq %%rdx, %0\n adcq $0,%0" : "+a"(r) : B); return r; }
  OP(+,,"d"(o.x)) OP(*,"mul %1\n", "r"(o.x) : "rdx")
  H operator-(H o) { return *this + ~o.x; }
  ull get() const { return x + !~x; }
  bool operator==(H o) const { return get() == o.get(); }
  bool operator<(H o) const { return get() < o.get(); }
};
static const H C = (ll)1e11+3; // (order ~ 3e9; random also ok)

struct HashInterval {
  vector<H> ha, pw;
  HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
    pw[0] = 1;
    rep(i,0,sz(str))
      ha[i+1] = ha[i] * C + str[i],
      pw[i+1] = pw[i] * C;
  }
  H hashInterval(int a, int b) { // hash [a, b)
    return ha[b] - ha[a] * pw[b - a];
  }
};

vector<H> getHashes(string& str, int length) {
  if (sz(str) < length) return {};
  H h = 0, pw = 1;
  rep(i,0,length)
    h = h * C + str[i], pw = pw * C;
  vector<H> ret = {h};
  rep(i,length,sz(str)) {
    ret.push_back(h = h * C + str[i] - pw * str[i-length]);
  }
  return ret;
}

H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return h;}
```

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 $\mathcal{O}(NM)$. 162257, 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;
  }
};
```

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: $\mathcal{O}(\log N)$ d58b62, 23 lines

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

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

IntervalCover.h

Description: Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add `|| R.empty()`. Returns empty set on failure (or if G is empty).
Time: $\mathcal{O}(N \log N)$ 3dcd0d, 19 lines

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

ConstantIntervals.h

Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.
Usage: `constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});`
Time: $\mathcal{O}\left(k \log \frac{n}{k}\right)$ c438a0, 19 lines

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

10.2 Misc. algorithms

TernarySearch.h

Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \dots < f(i) \geq \dots \geq f(b)$. 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: $\mathcal{O}(\log(b-a))$ 5aa241, 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: $\mathcal{O}(N \log N)$ a28917, 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;
}
```

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

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: $\mathcal{O}((N + (hi - lo)) \log N)$ 5d67e4, 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); }
};
```

10.4 Debugging tricks

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

10.5 Optimization tricks

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

10.5.1 Bit hacks

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

10.5.2 Pragmas

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

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 $a \pmod b$ in the range $[0, 2b)$.

510ea3, 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;
  }
};
```

FastInput.h

Description: Read an integer from stdin. Usage requires your program to pipe in input from file.
Usage: `./a.out < input.txt`
Time: About 5x as fast as cin/scanf.

0e82aa, 7 lines

```
int readInt() {
  int a, c;
  while ((a = getchar_unlocked()) < 40);
  if (a == '-') return -readInt();
  while ((c = getchar_unlocked()) >= 48) a = a * 10 + c - 480;
  return a - 48;
}
```

Kotlin (11)

11.1 Core

binarysearch.kt

Description: Binary search, because I skill issues
Usage: `bruh`

483e1c, 39 lines

```
inline fun BinarySearchFirstTrue(l:Int, r:Int, isTrue:(Int)->Boolean):Int
    ?{
    var L = l
    var R = r+1
    while(L < R){
        val m = (L + R) shr 1
        if(m == r+1 || isTrue(m)){
            R = m
        }else{
            L = m + 1
        }
    }
    return if(L == r+1) null else L
}
inline fun BinarySearchLastTrue(l:Int, r:Int, isTrue:(Int)->Boolean):Int?
    {
    var L = l-1
    var R = r
    while(L < R){
        val m = ((L + R) shr 1) + 1
        if(m == l-1 || isTrue(m)){
            L = m
        }else{
            R = m - 1
        }
    }
    return if(L == l-1) null else L
}
inline fun intTenarySearch(l:Int, r:Int, f:(Int)->Long):Int{
    var l = l-1
    var r = r
    while(r -l >1 ){
        val mid = (l + r ) shr 1
        if(f(mid) > f(mid+1)){
            r = mid
        }else{
            l = mid
        }
    }
    return l + 1
}
```

gcd.kt

Description: two types of gcd
Usage: `skill issue`

dab943, 26 lines

```
tailrec fun gcd(a: Int, b: Int): Int {
    if(b == 0) return a
    return if (a % b == 0) Math.abs(b) else gcd(b, a % b)
}
tailrec fun gcd(a: Long, b: Long): Long {
    if(b == 0L) return a
    return if (a % b == 0L) Math.abs(b) else gcd(b, a % b)
}
fun extendedGCD(p: Int, q: Int): IntArray {
    if (q == 0) return intArrayOf(p, 1, 0)
    val vals: IntArray = extendedGCD(q, p % q)
    val d = vals[0]
    val a = vals[2]
    val b = vals[1] - p / q * vals[2]
    return intArrayOf(d, a, b)
}
fun extendedInverse(a:Int,m:Int):Int?{
    val ans = extendedGCD(a,m)

    if(ans[0] != 1){
        return null
    }else{
        val ret = ans[1]
        return if(ret < 0) ret + m else ret
    }
}
```

integerdivision.kt

Description: integer divisions because it is hard
Usage: `bruh`

8b4777, 17 lines

```
infix fun Int.divceil(b:Int):Int{
    return (this/b) + if(this % b > 0) 1 else 0
}
infix fun Long.divceil(b:Long):Long{
    return (this/b) + if(this % b > 0) 1L else 0L
}
infix fun Int.div_neg_floor(b:Int):Int{
    return (this/b) - if(this % b < 0) 1 else 0
}
infix fun Long.div_neg_floor(b:Long):Long{
    return (this/b) - if(this % b < 0) 1L else 0L
}
fun modCount(x:Long, mod:Long, l:Long, r:Long):Long{
    val ans = ((r -x) div_neg_floor mod ) - ((l - x) divceil mod) + 1
    return maxOf(ans,0)
    //coun number of x mod mod in [l..r]
}
```

sieve.kt

Description: Very good sieve
Usage: `bruh`

7feab2, 73 lines

```
object sieve{

    const val sieveMx = 200005
    val primeOf = IntArray(sieveMx + 1)
    var primeCounter = 0
    val primeUpperBound = maxOf(25,(sieveMx.toDouble()/(Math.log(sieveMx.
        toDouble()) -4)).toInt() +3)
    val primes = IntArray(primeUpperBound)
    var sieveCalculated = false
    val nextPrime = IntArray(sieveMx+1)
    val nextPrimePower = IntArray(sieveMx+1)
    val afterPrimePowerDivison = IntArray(sieveMx+1)
    var mobius = IntArray(0)

    fun calculateSieveFast(){
        if(sieveCalculated){
            return
        }
        sieveCalculated = true
        for(i in 2..sieveMx){
            if(primeOf[i] == 0 ){
                primeOf[i] = i
                primes[primeCounter] = i
                primeCounter += 1
            }
            for(j in 0 until primeCounter){
                val p = primes[j]
                val pd = p * i
                if(p <= i && pd <= sieveMx){
                    primeOf[pd] = p
                }else{
                    break
                }
            }
        }
    }
    fun preparePrimePower(){
        nextPrime[1] = -1
        nextPrimePower[1] = -1
        afterPrimePowerDivison[1] = 1
        for(i in 2..sieveMx){
            val p = primeOf[i]
            val new = i / p
            nextPrime[i] = p
            if(nextPrime[new] == p){
                nextPrimePower[i] = nextPrimePower[new]
                afterPrimePowerDivison[i] = afterPrimePowerDivison[new]
            }else{
                afterPrimePowerDivison[i] = new
            }
            nextPrimePower[i] += 1
        }
    }
}
inline fun Int.eachPrimePower(act:(Int,Int)->Unit){
    assert(sieve.sieveCalculated)
    var here = this
    while(here > 1){
        act(sieve.nextPrime[here], sieve.nextPrimePower[here])
        here = sieve.afterPrimePowerDivison[here]
    }
}
fun Int.factors():List<Int>{
    val ret = mutableListOf(1)
    this.eachPrimePower { p, e ->
        val s = ret.toList()
        var now = 1
        repeat(e){
            now *= p
            ret.addAll(s.map{it * now})
        }
    }
    return ret
}
```

standard.kt
Description: my usual template
Usage: just use
495227, 184 lines

```
// 2022.11.03 at 13:22:33 GMT
import java.io.BufferedInputStream
import java.io.File
import java.io.PrintWriter
import kotlin.system.measureTimeMillis
import java.util.TreeMap
import java.util.TreeSet

// 1. Modded
const val p = 1000000007L
const val pI = p.toInt()
fun Int.adjust():Int{ if(this >= pI){ return this  - pI }else if (this <
      0){ return this + pI };return this }
fun Int.snap():Int{ if(this >= pI){return this - pI} else return this}
infix fun Int.mm(b:Int):Int{ return ((this.toLong() * b) % pI).toInt() }
infix fun Int.mp(b:Int):Int{ val ans = this + b;return if(ans >= pI) ans
      - pI else ans }
infix fun Int.ms(b:Int):Int{ val ans = this - b;return if(ans < 0) ans +
      pI else ans }
fun Int.inverse():Int = intPow(this,pI-2,pI)
fun Int.additiveInverse():Int{ return if(this == 0) 0 else pI - this }
infix fun Int.modDivide(b:Int):Int{ return this mm (b.inverse()) }
fun intPow(x:Int,e:Int,m:Int):Int{
    var X = x ; var E =e ; var Y = 1
    while(E > 0){
        if(E and 1 == 0){
            X = ((1L * X * X) % m).toInt()
            E = E shr 1
        }else{
            Y = ((1L * X * Y) % m).toInt()
            E -= 1
        }
    }
    return Y
}
// 2. DP initial values
const val plarge = 1_000_000_727
const val nlarge = -plarge
const val phuge = 2_727_000_000_000_000_000L
const val nhuge = -phuge
// 3. convenience conversions
val Boolean.chi:Int get() = if(this) 1 else 0 //characteristic function
val BooleanArray.chiarray:IntArray get() = IntArray(this.size){this[it].
     chi}
val Char.code :Int get() = this.toInt() -  'a'.toInt()
//3. hard to write stuff
fun IntArray.put(i:Int,v:Int){ this[i] = (this[i] + v).adjust() }
val mint:MutableList<Int> get() = mutableListOf<Int>()
val mong:MutableList<Long> get() = mutableListOf<Long>()
//4. more outputs
fun List<Char>.conca():String = this.joinToString("")
val CharArray.conca :String get() = this.joinToString("")
val IntArray.conca :String get() = this.joinToString(" ")
@JvmName("concaInt")
fun List<Int>.conca():String = this.joinToString(" ")
val LongArray.conca:String get() = this.joinToString(" ")
@JvmName("concaLong")
fun List<Long>.conca():String = this.joinToString(" ")
val String.size get() = this.length
const val randCount = 100
//7. bits
fun Int.has(i:Int):Boolean = (this and (1 shl i) != 0)
fun Long.has(i:Int):Boolean = (this and (1L shl i) != 0L)
//8 TIME
inline fun TIME(f:()->Unit){
    val t = measureTimeMillis(){
        f()
    }
    println("$t ms")
}
//9.ordered pair
const val interactive = false
object Reader{
    private const val BS = 1 shl 16
    private const val NC = 0.toChar()
    private val buf = ByteArray(BS)
    private var bId = 0
    private var size = 0
    private var c = NC

    var fakein = StringBuilder()

    private var IN: BufferedInputStream = BufferedInputStream(System.`in
          `, BS)
    val OUT: PrintWriter = PrintWriter(System.out)

    private val char: Char
        get() {
            if(interactive){
                return System.`in`.read().toChar()
            }
            while (bId == size) {
                size = IN.read(buf) // no need for checked exceptions
                if (size == -1) return NC
                bId = 0
            }
            return buf[bId++].toChar()
        }

    fun nextLong(): Long {
        var neg = false
        if (c == NC) c = char
        while (c < '0' || c > '9') {
            if (c == '-') neg = true
            c = char
        }
        var res = 0L
        while (c in '0'..'9') {
            res = (res shl 3) + (res shl 1) + (c - '0')
            c = char
        }
        return if (neg) -res else res
    }
    fun nextString():String{
        val ret = StringBuilder()
        while (true){ c = charif(!isWhitespace(c)){ break}}
        ret.append(c)
        while (true){
            c = char
            if(isWhitespace(c)){ break}
            ret.append(c)
        }
        return ret.toString()
    }
    fun isWhitespace(c:Char):Boolean{
        return c == ' ' || c == '\n' || c == '\r' || c == '\t'
    }
    fun rerouteInput(str:String){
        println("New Case ")
        IN = BufferedInputStream(str.byteInputStream(),BS)
    }
    fun flush(){
        OUT.flush()
    }
    fun takeFile(name:String){
        IN = BufferedInputStream(File(name).inputStream(),BS)
    }
}
fun eat(){ val st1 = TreeSet<Int>(); val st2 = TreeMap<Int,Int>()}
fun put(aa:Any){
    Reader.OUT.println(aa)
    if(interactive){ Reader.flush()}
}
fun done(){ Reader.OUT.close() }
fun share(aa:Any){
    if(aa is IntArray){Reader.fakein.append(aa.joinToString(" "))}
    else if(aa is LongArray){Reader.fakein.append(aa.joinToString(" "))}
    else if(aa is List<*>){Reader.fakein.append(aa.toString())}
    else{Reader.fakein.append(aa.toString())}
    Reader.fakein.append("\n")
}
val getint:Int get(){ val ans = getlong ; if(ans > Int.MAX_VALUE)
     IntArray(1000000000); return ans.toInt() }
val getlong:Long get() = Reader.nextLong()
val getstr:String get() = Reader.nextString()
fun getline(n:Int):IntArray{
    return IntArray(n){getint}
}
fun getlineL(n:Int):LongArray{
    return LongArray(n){getlong}
}
var dmark = -1
infix fun Any.dei(a:Any){
    dmark++
    val other = if(a is List<*>){ a.joinToString { " " }
    }else if(a is IntArray){ a.joinToString { " " }
    }else if(a is LongArray){ a.joinToString { " " }
    }else if(a is BooleanArray){ a.joinToString(" "){if(it)"1" else "0"}
    }else{a.toString()}
    println("<${dmark}>  ${this.toString()} : $other")
}
const val just = " "
fun crash(){
    throw Exception("Bad programme")}
fun assert(a:Boolean){
    if(!a){
        throw Exception("Failed Assertion")
    }}

const val singleCase = true
fun main(){
    repeat(1){

    }
    done()
}
```

11.2 DS

dsu.kt
Description: DSU
Usage: bruh
94b557, 32 lines

```
class DisjointUnionSets(val n: Int) {
    var size: IntArray = IntArray(n){1}
    var parent: IntArray = IntArray(n){it}
    var components:Int = n

    val successfulUnions:Int get() = n - components
    fun find(x: Int): Int {
        if (parent[x] != x) {
            parent[x] = find(parent[x])
        }
        return parent[x]
    }

    fun union(x: Int, y: Int):Boolean {
        var xRoot = find(x)
        var yRoot = find(y)
        if (xRoot == yRoot){
            return false
        }
        components--
        if(size[xRoot] < size[yRoot]){
            xRoot = yRoot.also { yRoot = xRoot }
        }
        parent[yRoot] = xRoot
        size[xRoot] += size[yRoot]
        return true
    }
    fun getsize(a:Int):Int{
        return size[find(a)]
    }
}
```

dsuparity.kt

Description: DSU parity, with path compression
Usage: bruh

e3329b, 55 lines

```
class DSU_parity_path(val n: Int) {
    var size: IntArray = IntArray(n){1}
    var parent: IntArray = IntArray(n){it}
    val changeParity = BooleanArray(n)
    var components:Int = n

    val successfulUnions:Int get() = n - components

    fun find(x: Int): Pair<Int,Boolean> {
        if (parent[x] != x) {
            val new = find(parent[x])
            parent[x] = new.first
            changeParity[x] = new.second xor changeParity[x]
        }
        return Pair(parent[x],changeParity[x])
    }
    fun canmerge(x:Int, y:Int, dif:Boolean):Boolean{
        val xx = find(x)
        val yy = find(y)
        if(xx.first == yy.first){
            val now = xx.second xor yy.second
            if(now != dif){
                return false
            }
            return true
        }
        return true
    }

    fun union(x: Int, y: Int,dif:Boolean):Boolean {
        val xRoot = find(x)
        val yRoot = find(y)
        if (xRoot.first == yRoot.first){
            val now = xRoot.second xor yRoot.second
            if(now != dif){
                return false
            }
            return true
        }
        components--
        var xx = xRoot.first
        var yy = yRoot.first
        val overdif = dif xor xRoot.second xor yRoot.second
        if(size[xx] < size[yy]){
            xx = yy.also { yy = xx }
        }
        parent[yy] = xx
        changeParity[yy] = overdif
        size[xx] += size[yy]
        return true
    }
    fun getsize(a:Int):Int{
        return size[find(a).first]
    }
}
```

graph.kt

Description: Linked list graph template
Usage: bruh

0a70fa, 130 lines

```
const val graphWeighed = false
class Graph(val n:Int, val m:Int, val directed:Boolean) {
    val maxedge = if (directed) m else m * 2

    var cnt = -1

    val edgecount:Int get() = cnt + 1

    val next = IntArray(maxedge)
    val head = IntArray(n) { -1 }
    val to = IntArray(maxedge)
    val from = IntArray(maxedge)
    val weights = IntArray(if (graphWeighed) m else 0)
    val Q = ArrayDeque<Int>()

    private fun primitive_add(u: Int, v: Int): Int {
        next[++cnt] = head[u]
        head[u] = cnt
        to[cnt] = v
        from[cnt] = u
        return cnt
    }

    fun add(u: Int, v: Int): Int {
        val e = primitive_add(u, v)
        if (!directed) {
            primitive_add(v, u)
        }
        return if (directed) e else e shr 1
    }

    fun addWeighted(u: Int, v: Int, w: Int):Int{
        val e = add(u, v)
        weights[e] = w
        return e
    }

    //Basic Transversals
    inline fun NS(a:Int, act:(Int)->Unit){
        var i= head[a]
        while(i != -1){
            act(to[i])
            i = next[i]
        }
    }
    inline fun NS_E(a:Int, act:(e:Int,v:Int)->Unit){
        var i= head[a]
        while(i != -1){
            act(i,to[i])
            i = next[i]
        }
    }

    // twice for undirected
    inline fun everyEdge(act:(a:Int, b:Int)->Unit){
        for(e in 0 until edgecount){
            act(from[e], to[e])
        }
    }

    //2 Basic Transversals
    var root = 0
    var preorder:IntArray = IntArray(0)
    var parent:IntArray = IntArray(0)
    val hasDFSorder:Boolean get() = preorder.size == n
    var parentEdge:IntArray = IntArray(0)

    //stores the order
    fun treeOrderDFS(withEdges:Boolean = false){
        parent = IntArray(n){-1}
        var pt = -1
        preorder = IntArray(n){-1}
        if(withEdges) parentEdge = IntArray(n){-1}

        Q.clear()
//          val Q = fastDeque(0,n)
        parent[root] = root
        Q.addLast(root)
        while(Q.isNotEmpty()){
            val a = Q.removeLast()
            val p = parent[a]
            preorder[++pt] = a
            NS_E(a){e,v ->
                if(v == p) return@NS_E
                if(withEdges) parentEdge[v] = if(directed) e else (e shr
                     1)
                parent[v] = a
                Q.addLast(v)
            }
        }
    }
    inline fun BFS(distRoot:Int, reached:(Int, Int)->Unit = {_,_ ->}):
        IntArray {
        Q.clear()
        val explored = IntArray(n+1){-1} // also store parents
        Q.addLast(distRoot)
        explored[distRoot] = -2
        val dist = IntArray(n){-1}
        dist[distRoot] = 0

        while(Q.size > 0){
            val x = Q.removeFirst()
            reached(x,explored[x])
            NS(x){ a->
                if(explored[a] == -1){
                    explored[a] = x
                    dist[a] = dist[x] + 1
                    Q.addLast(a)
                }
            }
        }
        return dist
    }
    //standard graph transversal orders
    inline fun leafFirst(act:(Int)->Unit){
        if(!hasDFSorder) treeOrderDFS()
       for(i in preorder.lastIndex downTo 0){
           act(preorder[i])
       }
    }
    inline fun rootFirst(act:(Int)->Unit){
        if(!hasDFSorder) treeOrderDFS()
        for(a in preorder){
            act(a)
        }
    }
    inline fun subs(v:Int, act:(Int)->Unit){
        NS(v){w ->
            if(w != parent[v]) act(w)
        }
    }
}
```

radix.kt

Description: radix sort for times getting kotlin issued
Usage: bruh

f7e0c8, 59 lines

```
private const val radixLog = 11
private const val radixBase = 1 shl radixLog
private const val radixMask = radixBase - 1
fun countSort(arr: IntArray, n: Int, expB: Int) {
    val output = IntArray(n)
    val count = IntArray(radixBase)
    for(i in 0 until n) {
        count[(arr[i] shr expB) and radixMask]++
    }
    for(i in 1 until radixBase){
        count[i] += count[i - 1]
    }
    for(i in n-1 downTo 0 ){
        val id = (arr[i] shr expB) and radixMask
        output[count[id] - 1] = arr[i]
        count[id]--
    }
    output.copyInto(arr)
}
fun IntArray.radixsort() {
    //Positives only!
    val n = this.size
    var b = 0
    repeat(3){
        countSort(this, n, b)
        b += radixLog
    }
}

fun countSortCarry(arr: IntArray, n: Int, expB: Int, extra:IntArray) {
    val output = IntArray(n)
    val newextra = IntArray(n)
    val count = IntArray(radixBase)

    for(i in 0 until n) {
        count[(arr[i] shr expB) and radixMask]++
    }
    for(i in 1 until radixBase){
        count[i] += count[i - 1]
    }
    for(i in n-1 downTo 0 ){
        val id = (arr[i] shr expB) and radixMask
        output[count[id] - 1] = arr[i]
        newextra[count[id] -1] = extra[i]
        count[id]--
    }
    output.copyInto(arr)
    newextra.copyInto(extra)
}
fun IntArray.radixCarry(carried:IntArray) {
    //Positives only!
    val n = this.size
    var b = 0
    repeat(3){
        countSortCarry(this, n, b,carried)
        b += radixLog
    }
}
```

rmq.kt

Description: range minimyum query, but can do max easily
Usage: bruh
52f3ed, 47 lines

```
typealias rmqType = Int
typealias rmqArrayType = IntArray
class rmq(val arr: rmqArrayType){
    val n = arr.size
    val store = mutableListOf<rmqArrayType>()
    init{
        preprocess()
    }
    companion object{
        const val max = false
    }
    private fun preprocess(){
        var s = 1
        var olds = 0
        while(s <= n){
            if(s == 1){
                val new = arr
                store.add(new)
            }else{
                val size = n-s+1
                val old = store.last()
                val new = rmqArrayType(size)
                for(i in 0 until size){
                    if(max){
                        new[i] = maxOf(old[i],old[i+olds])
                    }else{
                        new[i] = minOf(old[i],old[i+olds])
                    }

                }
                store.add(new)
            }
            olds = s
            s = s shl 1
        }
    }
    fun query(l:Int,r:Int):rmqType{
        val d = r - l + 1
        val i = 31 - d.countLeadingZeroBits()
        val s = 1 shl i
        val a1 = l
        val a2 = r - s + 1
        val ret1 = store[i][a1]
        val ret2 = store[i][a2]
        return if(max) maxOf(ret1,ret2) else minOf(ret1,ret2)
    }
}
```

st.kt

Description: Because no space
Usage: good luck, you cannot use
fabad5, 33 lines

```
class build_your_own(nSuggest:Int) {
    val n = if (nSuggest >= 2) (nSuggest - 1).takeHighestOneBit() shl 1
         else nSuggest
    inline fun segDivision(l: Int, r: Int, act: (index: Int, level: Int)
         -> Unit) {
        val l = maxOf(l, 0)
        val r = minOf(r, n - 1)
        var left = l + n
        var right = r + n + 1
        var level = 0
        while (left < right) {
            if (left and 1 != 0) {

                act(left, level)
                left += 1
            }
            if (right and 1 != 0) {
                right -= 1
                act(right, level)
            }
            left = left shr 1
            right = right shr 1
            level++
        }
    }

    fun query(l: Int, r: Int) {
        if (push) {
            pushPath(l + n)
            pushPath(r + n)
        }
        var ret = 0
        segDivision(l, r) { i, _ -> ret += sum[i] }
    }
}
```

11.3 tasks

Dijasktr.kt

Description: Need to have dijstraks prewritten because I skil issues
f4c17b, 30 lines

```
package bringtoICPC.Task
data class dijstrakitem(val a:Int, val x:Long):Comparable<dijstrakitem>{
    override fun compareTo(other:dijstrakitem):Int {
        return this.x.compareTo(other.x)
    }
}
fun Graph.dijstrak(d:LongArray):LongArray{
    val Q = PriorityQueue<dijstrakitem>()
    for(i in d.indices){
        if(d[i] == -1L){
            d[i] = phuge
        }else{
            Q.add(dijstrakitem(i,d[i]))
        }
    }
    while(Q.isNotEmpty()){
        val p = Q.poll()
        if(p.x != d[p.a]) continue
        val v = p.a
        NS_E(v){e,to ->
            val w = weights[if(directed) e else (e shr 1)]
            if(d[v] + w < d[to]){
                d[to] = d[v] + w
//                p[to] = v
                Q.add(dijstrakitem(to,d[to]))
            }
        }
    }
    return d
}
```

FFT.kt

Description: FFT in WF = free
f4eacc, 110 lines

```
class FFT {
    companion object{
        private const val maxPower = 20 // 262144
        private const val n = 1 shl maxPower
        private const val fftmod = 998244353
        private const val root = 15311432
        private const val root_1 = 469870224
        private const val root_level = 23
        private const val root_pw = 1 shl root_level

        const val FFTcut = 80

        private inline fun normalize(int: Int) = (int shr Int.SIZE_BITS -
             1 and fftmod) + int
        infix fun Int.modm(other:Int) = (this.toLong() * other % fftmod).
             toInt()
        infix fun Int.modplus(other: Int) = normalize(this + other -
             fftmod) // overflow-safe even if MOD>= 2^30
        infix fun Int.modminus(other: Int) = normalize(this - other)

        val quickpower = Array(maxPower+1){p ->
            val base = intPow(root, 1 shl (root_level - p), fftmod)
            val ret = IntArray(1 shl p)
            ret[0] = 1
            for(i in 1 until (1 shl p)){
                ret[i] = ret[i-1] modm base
            }
            ret
        }
        private val rev = IntArray(n).also { rev ->
            var bit = 1
            var rbit = n shr 1
            while(bit < n) {
                for(i in 0 until bit) {
                    rev[i or bit] = rbit or rev[i]
                }
                bit = bit shl 1
                rbit = rbit shr 1
            }
        }
        fun fft(a:IntArray,invert:Boolean){
            val n = a.size
            assert(n <= this.n)
            if(n <= 1) return
            val level = this.n.countTrailingZeroBits()
            val st = this.n / n
            for(i in 0 until n) {
                val j = rev[i * st]
                if(i < j) a[i] = a[j].also { a[j] = a[i] }
            }

            var len = 2
            var ang = 1 shl (level -1)
            if(invert) ang = -ang
            val pt = quickpower[level]
            while(len <= n){
                var i = 0
                val h = len shr 1
                while(i < n){
                    var k = 0
                    for(j in i until i+h){
                        val u = a[j]
                        val w = pt[k]
                        val v = a[j+h] modm w
                        a[j] = u modplus v
                        a[j+h] = u modminus v
                        k = k + ang and (1 shl level) - 1
                    }
                    i += len
                }
                len = len shl 1
                ang = ang shr 1
            }
            if(invert){
                val n_1 = intPow(n,fftmod-2,fftmod)
                for((i,x) in a.withIndex()){
                    a[i] = (x.toLong() * n_1 % fftmod).toInt()
                }
            }
        }
        fun fullconvolution(at:IntArray,bt:IntArray):IntArray{
            return fullconvolutionOpt(at,bt,at.size,bt.size)
        }
        fun brute(A:IntArray,B:IntArray):IntArray{
            val ret = IntArray(A.size + B.size -1)
            for(i in A.indices){
                for(j in B.indices){
                    ret[i+j] = ret[i+j] modplus (A[i] modm B[j])
                }
            }
            return ret
        }
        fun fullconvolutionOpt(at:IntArray,bt:IntArray,sizeA:Int,sizeB:
             Int):IntArray{
            // 1 shl 18 done in 77 ms
            if(sizeA <= FFTcut || sizeB <= FFTcut){
                return brute(at,bt)
            }
            val maxSize = (sizeA + sizeB - 1).takeHighestOneBit() * 2
            check(maxSize <= (1 shl maxPower ))
            val a = at.copyOf(maxSize)
            val b = bt.copyOf(maxSize)
            val expectedSize = at.size + bt.size - 1
            fft(a,false)
            fft(b,false)
            for(i in a.indices){
                a[i] = (a[i].toLong() * b[i] % fftmod).toInt()
            }
            fft(a,true)
            return a.copyOf(expectedSize)
        }
    }
}
```

series.kt

Description: Genfun in WF = free c47d6e, 115 lines

```
class series(val arr:IntArray){
    operator fun get(i:Int) = arr[i]
    operator fun set(i:Int, v:Int){ arr[i] = v }
    val size:Int get() = arr.size
    val degree:Int get(){
        var v = 0
        for(i in arr.indices){ if(arr[i] == 0) v++ else return v}
        return v
    }

    fun take(s:Int): series {
        val new = IntArray(s)
        arr.copyInto(new,0,0,minOf(s,arr.size))
        return series(new)
    }
    fun sliceSeries(l:Int,r:Int):series{
        val new = IntArray(r - l +1)
        arr.copyInto(new,0,l,minOf(r+1,arr.size))
        return series(new)
    }
    fun copyOf(len:Int):series{
        return series(arr.copyOf(len))
    }
    fun takefrom(i:Int):series{
        return series(arr.sliceArray(i until arr.size))
    }

    constructor(arr:IntArray,len:Int):this(arr.copyOf(len))
    fun invert():series{
        //300ms : 200000
        var Q = series(intArrayOf(arr[0].inverse()))
        while(Q.size < this.size){
            val newlen = Q.size * 2
            val A = this.take(newlen)
            val Q2 = Q.times(Q,newlen)
            val got = A.times(Q2,newlen)
            val flip = Q.multiply(2).take(newlen)
            val final = flip.minus(got)
            Q = final
        }
        return Q.take(this.size)
    }
    fun multiply(c:Int):series{
        return series(IntArray(arr.size){arr[it] mm c})
    }
    fun plus(other:series):series{
        val new = IntArray(this.size){this[it] mp other[it]}
        return series(new)
    }
    fun minus(other:series):series{
        val new = IntArray(this.size){this[it] ms other[it]}
        return series(new)
    }
    fun negate():series{
        return series(IntArray(this.size){this[it].additiveInverse()})
    }
    fun times(other:series, newsize:Int):series{
        //assume same size
        val new = FFT.fullconvolution(this.arr, other.arr).copyOf(newsize
             )
        return series(new)
    }
    fun differentiate():series{
        val new = IntArray(arr.size){if(it == arr.lastIndex) 0 else arr[
             it+1] mm (it+1)}
        return series(new)
    }
    fun integrate():series{
        val new = IntArray(arr.size + 1){if(it == 0) 0 else arr[it-1]
             modDivide (it)}
        return series(new)
    }
    fun log():series{
        require(this.arr[0] == 1)
        return (this.differentiate()).times(this.invert(),this.arr.size
             -1 ).integrate()
    }
    fun exp():series{
        check(arr[0] ==0)
        var Q = series(intArrayOf(1))
        while(Q.size < this.size){
            val newlen = Q.size * 2
            val s1 = series(intArrayOf(1),newlen)
            val s2 = this.take(newlen)
            val s3 = Q.take(newlen).log()
            val subtract = s1.plus(s2.minus(s3))
            val final = Q.times(subtract,newlen)
            Q = final
        }
        return Q.take(this.size)
    }
    fun raisePower(v:Long):series{
        val d = this.degree
        val sack = this.takefrom(d)
        if(d == this.size){
            return series(intArrayOf(),this.size)
        }
        val lead = this[d]
        val newlead = intPowEXP(lead,v, pI)
        val startfrom = (d * v)
        val first = (sack.multiply(lead.inverse()).log().multiply((v % pI
             ).toInt())).exp().multiply(newlead)
        return series(IntArray(this.size){if(it < startfrom) 0 else first
             [it - startfrom.toInt()]})
    }
    fun shift(v:Int):series{
        //this is slow log square method
        if(this.arr.size == 1) return this
        if(this.arr.size == 2){
            return series(intArrayOf(this[0] mp (v mm this[1]), this[1]))
        }else{
            val cut = this.arr.size shr 1
            val A = sliceSeries(0,cut-1).shift(v).copyOf(this.arr.size)
            val B = sliceSeries(cut,this.arr.lastIndex).shift(v)
            val X = series(IntArray(cut+1){ FACT.choose(cut, it) mm
                 intPow(v, cut - it, pI)})
            return A.plus(B.times(X,this.arr.size))
        }
        //replace x by x+v
    }
}
```

xorconvolution.kt

Description: fast hadamard transform in WF = free f15783, 22 lines

```
fun xorTransform(a:IntArray){
    val ret = a
    val n = a.size.countTrailingZeroBits()
    assert((1 shl n) == a.size)
    for(i in 0 until n){
        for(m in 0 until (1 shl n)){
            if(!m.has(i)){
                val k = m xor ( 1 shl i )
                ret[m] = (ret[m] mp ret[k]).also { ret[k] = ret[m] ms ret
                     [k]}
            }
        }
    }
}
fun xorConvolute(a:IntArray, b:IntArray):IntArray{
    val aa = a.copyOf()
    val bb = b.copyOf()
    xorTransform(aa)
    xorTransform(bb)
    val ret = aa.pointtimes(bb)
    xorTransform(ret)
    return (ret).eachdivide(ret.size)
}
```