.bashrc

```
build() (
  g++ $@ -o $1.e -DLOC -std=c++17          \
      -Wall -W -Wfatal-errors -Wshadow \
      -Wlogical-op -Wconversion -Wfloat-equal
)

b() ( build $@ -O2 )

d() ( build $@ -fsanitize=address,undefined \
          -D_GLIBCXX_DEBUG -g )

run() ( $1 $2 && echo start >&2 && time ./$2.e )

loo() (
  set -e; $1 $2; $1 $3
  for ((;;)) {
    ./$3.e > gen.in
    time ./$2.e < gen.in > gen.out
  }
)

cmp() (
  set -e; $1 $2; $1 $3; $1 $4
  for ((;;)) {
    ./$4.e > gen.in;          echo -n 0
    ./$2.e < gen.in > p1.out; echo -n 1
    ./$3.e < gen.in > p2.out; echo -n 2
    diff p1.out p2.out;       echo -n Y
  }
)

# Other flags:
# -Wformat=2 -Wshift-overflow=2 -Wcast-qual
# -Wcast-align -Wduplicated-cond
# -D_GLIBCXX_DEBUG_PEDANTIC -D_FORTIFY_SOURCE=2
# -fno-sanitize-recover -fstack-protector

# Print optimization info: -fopt-info-all

# Stacktrace on STL assert:
#   g++ -D_GLIBCXX_DEBUG -H test.cpp 2>&1
#     >/dev/null | grep "debug/macros.h"
#   #ifdef __SANITIZE_ADDRESS__
#   extern "C"
#     void __sanitizer_print_stack_trace();
```

.vimrc

```
se ai aw cin cul ic is nocp nohls nu sc scs
se bg=dark sw=4 ts=4 so=7 ttm=9
sy on
vn _ :w !cpp -dD -P -fpreprocessed \|            \
  sed -z sg\\sggg \| md5sum \| cut -c-4 <cr>
```

template.cpp b927

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

using ll  = long long;
using Vi  = vector<int>;
using Pii = pair<int,int>;

#define mp make_pair
#define pb push_back
#define x  first
#define y  second

#define rep(i,b,e) for(int i=(b); i<(e); i++)
#define each(a,x)  for(auto& a : (x))
#define all(x)     (x).begin(),(x).end()
#define sz(x)      int((x).size())

int main() {
  cin.sync_with_stdio(0); cin.tie(0);
  cout << fixed << setprecision(12);

  // Don't call destructors:
  cout << flush; _Exit(0);
} // dd2c

// > Debug printer

#define tem \
  template<class t,class u,class...w> auto
#define pri(x,y,z) \
  tem operator<<(t&o,u a)->decltype(z,o) \
  { o << *#x; y; z; return o << #x+1; }

pri({},, a.print())
pri((),, o << a.x << ", " << a.y)

pri([], auto d=""; for (auto i : a)
  (o << d << i, d = ", "), all(a))

void DD(...) {}
tem DD(t s, u a, w... k) {
  for (int b=1; cerr << *s++, *s && *s - b*44;)
    b += 2 / (*s*2 - 81);
  cerr << ": " << a; DD(s, k...);
} // dbb7

#ifdef LOC
#define deb(x...) \
  DD("[,\b :] "#x, __LINE__, x), cerr << endl
#else
#define deb(...)
#endif

#define DBP(x...) void print() { DD(#x, x); }

// > Utils

// Return smallest k such that 2^k > n
// Undefined for n = 0!
int uplg(int n) { return 32-__builtin_clz(n); }
int uplg(ll n){ return 64-__builtin_clzll(n); }

// Compare with certain epsilon (branchless)
// Returns -1 if a < b; 1 if a > b; 0 if equal
// a and b are assumed equal if |a-b| <= eps
int cmp(double a, double b, double eps=1e-9) {
  return (a > b+eps) - (a+eps < b);
} // 81c1
```

template.java 4cea

```
import java.io.OutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.io.InputStreamReader;
import java.util.StringTokenizer;
import java.io.BufferedReader;
import java.io.InputStream;

public class Main {
  public static void main(String[] args) {
    try (PrintWriter out =
          new PrintWriter(System.out)) {
        new Task().solve(
          new InputReader(System.in), out);
    } // f642
  } // cbc2

  static class InputReader {
    BufferedReader r;
    StringTokenizer t = null;

    public InputReader(InputStream s) {
      r = new BufferedReader(
        new InputStreamReader(s), 32768);
    } // 0a0f

    public String next() {
      while (t == null || !t.hasMoreTokens())
        try {
          t = new StringTokenizer(
            r.readLine());
        } catch (IOException e) {
          throw new RuntimeException(e);
        } // fdda
      return t.nextToken();
    } // d64b

    public int nextInt() {
      return Integer.parseInt(next());
    } // 7737
  } // 94b9

  static class Task {
    public void solve(InputReader in,
                      PrintWriter out) {
      int n = in.nextInt();
      out.printf("hello %d", n);
    } // 477d
  } // 30cf
} // 75cd
```

various.h 1a4d

```
// If math constants like M_PI are not found
// add this at the beginning of file
#define _USE_MATH_DEFINES

// Pragmas
#pragma GCC optimize("Ofast,unroll-loops")
#pragma GCC target("popcnt,avx,tune=native")

// Clock
while (clock() < duration*CLOCKS_PER_SEC)

// Automatically implement operators:
// 1. != if == is defined
// 2. >, <= and >= if < is defined
using namespace rel_ops;

// Mersenne twister for randomization.
mt19937_64 rnd(chrono::steady_clock::now()
  .time_since_epoch().count());

// To shuffle randomly use:
shuffle(all(vec), rnd);

// To pick random integer from [A;B] use:
uniform_int_distribution<> dist(A, B);
int value = dist(rnd);

// To pick random real number from [A;B] use:
uniform_real_distribution<> dist(A, B);
double value = dist(rnd);

// Floats can represent integers up to 19*10^6
// Doubles can represent integers up to 9*10^15
```

various.py a9e9

```
input().split(' ') # Read and split into words
print('abc', end='') # Print without newline

>>> from fractions import *
>>> Fraction(16, -10)
Fraction(-8, 5)
>>> Fraction(123)
Fraction(123, 1)
>>> Fraction()
Fraction(0, 1)
>>> Fraction('3/7')
Fraction(3, 7)
>>> Fraction(' -3/7 ')
Fraction(-3, 7)
>>> Fraction('1.414213 \t\n')
>>> Fraction('7e-6')
Fraction(7, 1000000)
>>> Fraction(2.25)
Fraction(9, 4)
>>> Fraction(1.1)
Fraction(2476979795053773, 2251799813685248)
>>> Fraction('1/2') * Fraction('4/3')
Fraction(2, 3)
>>> Fraction(16, 5).numerator
16
>>> Fraction(16, 5).denominator
5

>>> from decimal import *
>>> getcontext().prec = 28
>>> Decimal(10)
Decimal('10')
>>> Decimal('3.14')
Decimal('3.14')
>>> Decimal(3.14)
Decimal('3.140000000000000124344978758017...')
>>> Decimal((0, (3, 1, 4), -2))
Decimal('3.14')
>>> Decimal(str(2.0 ** 0.5))
Decimal('1.4142135623730951')
>>> Decimal(2) ** Decimal('0.5')
Decimal('1.414213562373095048801688724')
>>> Decimal('NaN')
Decimal('NaN')
>>> Decimal('-Infinity')
Decimal('-Infinity')
```

geometry/convex_hull.h 511a

```
#include "vec2.h"

// Translate points such that lower-left point
// is (0, 0). Returns old point location; O(n)
vec2 normPos(vector<vec2>& points) {
  auto q = points[0].yx();
  each(p, points) q = min(q, p.yx());
  vec2 ret{q.y, q.x};
  each(p, points) p = p-ret;
  return ret;
} // ee96

// Find convex hull of points; time: O(n lg n)
// Points are returned counter-clockwise,
// first point is the lowest-left.
vector<vec2> convexHull(vector<vec2> points) {
  vec2 pivot = normPos(points);
  sort(all(points));
  vector<vec2> hull;

  each(p, points) {
    while (sz(hull) >= 2) {
      vec2 a = hull.back() - hull[sz(hull)-2];
      vec2 b = p - hull.back();
      if (a.cross(b) > 0) break;
      hull.pop_back();
    } // ad91
    hull.pb(p);
  } // 5908

  // Translate back, optional
  each(p, hull) p = p+pivot;
  return hull;
```

```
} // 62ed

// Find point p that minimizes dot product p*q.
// Returns point index in hull; time: O(lg n)
// If multiple points have same dot product
// one with smallest index is returned.
// Points are expected to be in the same order
// as output from convexHull function.
int minDot(const vector<vec2>& hull, vec2 q) {
  auto C = [](vec2 a, vec2 b) {
    return mp(a.dot(b), a.cross(b));
  }; // 27c4
  auto search = [&](int b, int e, vec2 p) {
    int f = b, g = e;
    while (b+1 < e) {
      int m = (b+e) / 2;
      (C(p, hull[m-1]) > C(p, hull[m])
        ? b : e) = m;
    } // 618b
    rep(i, 0, min(g-f, 2)) {
      if (C(p, hull[f+i]) < C(p, hull[b]))
        b = f+i;
      if (C(p, hull[g-i-1]) < C(p, hull[b]))
        b = g-i-1;
    } // 9413
    return b;
  }; // e993
  int m = search(0, sz(hull), {0, -1});
  int i = search(0, m, q);
  int j = search(m, sz(hull), q);
  return C(q, hull[i]) > C(q, hull[j])
    ? j : i;
} // 2fba
```

geometry/convex_hull_dist.h 2859

```
#include "vec2.h"

// Check if p is inside convex polygon. Hull
// must be given in counter-clockwise order.
// Returns 2 if inside, 1 if on border,
// 0 if outside; time: O(n)
int insideHull(vector<vec2>& hull, vec2 p) {
  int ret = 1;
  rep(i, 0, sz(hull)) {
    auto v = hull[(i+1)%sz(hull)] - hull[i];
    auto t = v.cross(p-hull[i]);
    ret = min(ret, cmp(t, 0)); // For doubles
    //ret = min(ret, (t>0) - (t<0)); // Ints
  } // 0d40
  return int(max(ret+1, 0));
} // 1f39

#include "segment2.h"

// Get distance from point to hull; time: O(n)
double hullDist(vector<vec2>& hull, vec2 p) {
  if (insideHull(hull, p)) return 0;
  double ret = 1e30;
  rep(i, 0, sz(hull)) {
    seg2 seg{hull[(i+1)%sz(hull)], hull[i]};
    ret = min(ret, seg.distTo(p));
  } // f3be
  return ret;
} // a00c

// Compare distance from point to hull
// with sqrt(d2); time: O(n)
// -1 if smaller, 0 if equal, 1 if greater
int cmpHullDist(vector<vec2>& hull,
                vec2 p, ll d2) {
  if (insideHull(hull,p)) return (d2<0)-(d2>0);
  int ret = 1;
  rep(i, 0, sz(hull)) {
    seg2 seg{hull[(i+1)%sz(hull)], hull[i]};
    ret = min(ret, seg.cmpDistTo(p, d2));
  } // 28cb
  return ret;
} // 30f3
```

geometry/convex_hull_sum.h 7f53

```
#include "vec2.h"

// Get edge sequence for given polygon
// starting from lower-left vertex; time: O(n)
// Returns start position.
vec2 edgeSeq(vector<vec2> points,
             vector<vec2>& edges) {
  int i = 0, n = sz(points);
  rep(j, 0, n)
    if (points[i].yx() > points[j].yx()) i = j;
  rep(j, 0, n) edges.pb(points[(i+j+1)%n] -
                        points[(i+j)%n]);
  return points[i];
} // 3aa7

// Minkowski sum of given convex polygons.
// Vertices are required to be in
// counter-clockwise order; time: O(n+m)
vector<vec2> hullSum(vector<vec2> A,
                     vector<vec2> B) {
  vector<vec2> sum, e1, e2, es(sz(A) + sz(B));
  vec2 pivot = edgeSeq(A, e1) + edgeSeq(B, e2);
  merge(all(e1), all(e2), es.begin());

  sum.pb(pivot);
  each(e, es) sum.pb(sum.back() + e);
  sum.pop_back();
  return sum;
} // f183
```

geometry/halfplanes.h 356a

```
#include "vec2.h"
#include "line2.h"

// Intersect halfplanes given by `lines`
// and output hull vertices to `out`
// in counter-clockwise order. Returns true
// if intersection is non-empty and bounded.
// Unbounded cases are not supported,
// add bounding-box if necessary. Works only
// with floating point vec2/line2; O(n lg n)
// PARTIALLY TESTED
bool intersectHalfplanes(vector<line2> in,
                         vector<vec2>& out) {
  sort(all(in), [](line2 a, line2 b) {
    return (a.v.angleCmp(b.v) ?:
            a.c*b.v.len() - b.c*a.v.len()) < 0;
  }); // 82fb

  int a = 0, b = 0, n = sz(in);
  vector<line2> dq(n+5);
  out.resize(n+5);
  dq[0] = in[0];

  rep(i, 1, n+1) {
    if (i == n) in.pb(dq[a]);
    if (!in[i].v.angleCmp(in[i-1].v)) continue;
    while (a < b && in[i].side(out[b-1]) > 0)
      b--;
    while (i!=n && a<b && in[i].side(out[a])>0)
      a++;
    if (in[i].intersect(dq[b], out[b]))
      dq[++b] = in[i];
  } // b9ba

  out.resize(b);
  out.erase(out.begin(), out.begin()+a);
  return b-a > 2;
} // f334
```

geometry/line2.h 9207

```
#include "vec2.h"

// 2D line/halfplane structure
// PARTIALLY TESTED

// Base class of versions for ints and doubles
template<class T, class P, class S>
struct bline2 {
  // For lines: v * point == c
  // For halfplanes: v * point <= c
  // (i.e. normal vector points outside)
  P v; // Normal vector [A; B]
  T c; // Offset (C parameter of equation)
  DBP(v, c);

  // Line through 2 points; normal vector
  // points to the right of ab vector
  static S through(P a, P b) {
    return { (a-b).perp(), a.cross(b) };
  } // 9ac7

  // Parallel line through point
  static S parallel(P a, S b) {
    return { b.v, b.v.dot(a) };
  } // 8e1c

  // Perpendicular line through point
  static S perp(P a, S b) {
    return { b.v.perp(), b.v.cross(a) };
  } // 7b75

  // Distance from point to line
  double distTo(P a) {
    return fabs(v.dot(a)-c) / v.len();
  } // 79e6
}; // ee4f

// Version for integer coordinates (long long)
struct line2i : bline2<ll, vec2i, line2i> {
  line2i() : bline2{{}, 0} {}
  line2i(vec2i a, ll b) : bline2{a, b} {}

  // Returns 0 if point a lies on the line,
  // 1 if on side where normal vector points,
  // -1 if on the other side.
  int side(vec2i a) {
    ll d = v.dot(a);
    return (d > c) - (d < c);
  } // 18a7
}; // fc9c

// Version for double coordinates
// Requires cmp() from template
struct line2d : bline2<double, vec2d, line2d> {
  line2d() : bline2{{}, 0} {}
  line2d(vec2d a, double b) : bline2{a, b} {}

  // Returns 0 if point a lies on the line,
  // 1 if on side where normal vector points,
  // -1 if on the other side.
  int side(vec2d a) { return cmp(v.dot(a),c); }

  // Intersect this line with line a, returns
  // true on success (false if parallel).
  // Intersection point is saved to `out`.
  bool intersect(line2d a, vec2d& out) {
    double d = v.cross(a.v);
    if (!cmp(d, 0)) return 0;
    out = (v*a.c - a.v*c).perp() / d;
    return 1;
  } // 2e68
}; // ab54
using line2 = line2d;
```

geometry/rmst.h 476a

```
#include "../structures/find_union.h"

// Rectilinear Minimum Spanning Tree
// (MST in Manhattan metric); time: O(n lg n)
// Returns MST weight. Outputs spanning tree
// to G, vertex indices match point indices.
// Edge in G is pair (target, weight).
ll rmst(vector<Pii>& points,
        vector<vector<Pii>>& G) {
  int n = sz(points);
  vector<pair<int, Pii>> edges;
  vector<Pii> close;
  Vi ord(n), merged(n);
  iota(all(ord), 0);

  function<void(int,int)> octant =
      [&](int begin, int end) {
    if (begin+1 >= end) return;

    int mid = (begin+end) / 2;
    octant(begin, mid);
    octant(mid, end);

    int j = mid;
    Pii best = {INT_MAX, -1};
    merged.clear();

    rep(i, begin, mid) {
      int v = ord[i];
      Pii p = points[v];

      while (j < end) {
        int e = ord[j];
        Pii q = points[e];
        if (q.x-q.y > p.x-p.y) break;
        best = min(best,make_pair(q.x+q.y, e));
        merged.pb(e);
        j++;
      } // 8576

      if (best.y != -1) {
        int alt = best.x-p.x-p.y;
        if (alt < close[v].x)
          close[v] = {alt, best.y};
      } // 4208
      merged.pb(v);
    } // f3ff

    while (j < end) merged.pb(ord[j++]);
    copy(all(merged), ord.begin()+begin);
  }; // a4e1

  rep(i, 0, 4) {
    rep(j, 0, 2) {
      sort(all(ord), [&](int l, int r) {
        return points[l] < points[r];
      }); // fe33
      close.assign(n, {INT_MAX, -1});
      octant(0, n);
      rep(k, 0, n) {
        Pii p = close[k];
        if (p.y != -1) edges.pb({p.x,{k,p.y}});
```

```
        points[k].x *= -1;
      } // 1c1d
    } // 9b38
    each(p, points) p = {p.y, -p.x};
  } // d06f

  ll sum = 0;
  FAU fau(n);
  sort(all(edges));
  G.assign(n, {});

  each(e, edges) if (fau.join(e.y.x, e.y.y)) {
    sum += e.x;
    G[e.y.x].pb({e.y.y, e.x});
    G[e.y.y].pb({e.y.x, e.x});
  } // b04a
  return sum;
} // f586
```

geometry/segment2.h 6504

```
#include "vec2.h"

// 2D segment structure; PARTIALLY TESTED

// Base class of versions for ints and doubles
template<class P, class S> struct bseg2 {
  P a, b; // Endpoints

  // Distance from segment to point
  double distTo(P p) const {
    if ((p-a).dot(b-a) < 0) return (p-a).len();
    if ((p-b).dot(a-b) < 0) return (p-b).len();
    return double(abs((p-a).cross(b-a)))
                  / (b-a).len();
  } // 62a2
}; // 85bc

// Version for integer coordinates (long long)
struct seg2i : bseg2<vec2i, seg2i> {
  seg2i() {}
  seg2i(vec2i c, vec2i d) : bseg2{c, d} {}

  // Check if segment contains point p
  bool contains(vec2i p) {
    return (a-p).dot(b-p) <= 0 &&
           (a-p).cross(b-p) == 0;
  } // c598

  // Compare distance to p with sqrt(d2)
  // -1 if smaller, 0 if equal, 1 if greater
  int cmpDistTo(vec2i p, ll d2) const {
    if ((p-a).dot(b-a) < 0) {
      ll l = (p-a).len2();
      return (l > d2) - (l < d2);
    } // d1a6
    if ((p-b).dot(a-b) < 0) {
      ll l = (p-b).len2();
      return (l > d2) - (l < d2);
    } // 9e65

    ll c = abs((p-a).cross(b-a));
    d2 *= (b-a).len2();
    return (c*c > d2) - (c*c < d2);
  } // 726d
}; // 4df2

// Version for double coordinates
// Requires cmp() from template
struct seg2d : bseg2<vec2d, seg2d> {
  seg2d() {}
  seg2d(vec2d c, vec2d d) : bseg2{c, d} {}

  bool contains(vec2d p) {
    return cmp((a-p).dot(b-p), 0) <= 0 &&
           cmp((a-p).cross(b-p), 0) == 0;
  } // b507
}; // 2036

using seg2 = seg2d;
```

geometry/vec2.h 6e47

```
// 2D point/vector structure; PARTIALLY TESTED

// Base class of versions for ints and doubles
template<class T, class S> struct bvec2 {
  T x, y;
  S operator+(S r) const {return{x+r.x,y+r.y};}
  S operator-(S r) const {return{x-r.x,y-r.y};}
  S operator*(T r) const { return {x*r, y*r}; }
  S operator/(T r) const { return {x/r, y/r}; }

  T dot(S r)   const { return x*r.x + y*r.y; }
  T cross(S r) const { return x*r.y - y*r.x; }
  T len2()     const { return x*x + y*y; }
  double len() const { return hypot(x, y); }
  S perp()     const { return {-y,x}; } // CCW

  pair<T, T> yx() const { return {y, x}; }

  double angle() const { //[0;2*PI] CCW from OX
    double a = atan2(y, x);
    return (a < 0 ? a+2*M_PI : a);
  } // 7095
}; // 17ed

// Version for integer coordinates (long long)
struct vec2i : bvec2<ll, vec2i> {
  vec2i() : bvec2{0, 0} {}
  vec2i(ll a, ll b) : bvec2{a, b} {}

  bool upper() const { return (y ?: x) >= 0; }

  int angleCmp(vec2i r) const {
    ll c = cross(r);
    return r.upper()-upper() ?: (c<0) - (c>0);
  } // b35f

  // Compare by angle, length if angles equal
  bool operator<(vec2i r) const {
    return (angleCmp(r) ?:
            len2() - r.len2()) < 0;
  } // 6f78

  bool operator==(vec2i r) const {
    return x == r.x && y == r.y;
  } // 136e
}; // d3f4

// Version for double coordinates
// Requires cmp() from template
struct vec2d : bvec2<double, vec2d> {
  vec2d() : bvec2{0, 0} {}
  vec2d(double a, double b) : bvec2{a, b} {}

  bool upper() const {
    return (cmp(y, 0) ?: cmp(x, 0)) >= 0;
  } // 086c

  int angleCmp(vec2d r) const {
    return r.upper() - upper() ?:
           cmp(0, cross(r));
  } // 12f3

  // Compare by angle, length if angles equal
  bool operator<(vec2d r) const {
    return (angleCmp(r) ?:
            cmp(len2(), r.len2())) < 0;
  } // f3d7

  bool operator==(vec2d r) const {
    return !cmp(x, r.x) && !cmp(y, r.y);
  } // 81cd

  vec2d unit() const { return *this / len(); }

  vec2d rotate(double a) const { // CCW
    return {x*cos(a) - y*sin(a),
            x*sin(a) + y*cos(a)}; // 1890
  } // 97e3
}; // 08e9

using vec2 = vec2d;
```

graphs/2sat.h 2443

```
// 2-SAT solver; time: O(n+m), space: O(n+m)
// Variables are indexed from 1 and
// negative indices represent negations!
// Usage: SAT2 sat(variable_count);
// (add constraints...)
// bool solution_found = sat.solve();
// sat[i] = value of i-th variable, 0 or 1
//          (also indexed from 1!)
// (internally: positive = i*2-1, neg. = i*2-2)
struct SAT2 : Vi {
  vector<Vi> G;
  Vi order, flags;

  // Init n variables, you can add more later
  SAT2(int n = 0) : G(n*2) {}

  // Add new var and return its index
  int addVar() {
    G.resize(sz(G)+2); return sz(G)/2;
  } // 98f3

  // Add (i => j) constraint
  void imply(int i, int j) {
    i = i*2 ^ i >> 31;
    j = j*2 ^ j >> 31;
    G[--i].pb(--j); G[j^1].pb(i^1);
  } // 8e25

  // Add (i v j) constraint
  void either(int i, int j) { imply(-i, j); }

  // Constraint at most one true variable
  void atMostOne(Vi& vars) {
    int y, x = addVar();
    each(i, vars) {
      imply(x, y = addVar());
      imply(i, -x); imply(i, x = y);
    } // 24aa
  } // 3ed7

  // Solve and save assignments in `values`
  bool solve() { // O(n+m), Kosaraju is used
    assign(sz(G)/2+1, -1);
    flags.assign(sz(G), 0);
    rep(i, 0, sz(G)) dfs(i);
    while (!order.empty()) {
      if (!propag(order.back()^1, 1)) return 0;
      order.pop_back();
    } // 5594
    return 1;
  } // 1e58

  void dfs(int i) {
    if (flags[i]) return;
    flags[i] = 1;
    each(e, G[i]) dfs(e);
    order.pb(i);
  } // d076

  bool propag(int i, bool first) {
    if (!flags[i]) return 1;
    flags[i] = 0;
    if (at(i/2+1) >= 0) return first;
    at(i/2+1) = i&1;
    each(e, G[i]) if (!propag(e, 0)) return 0;
    return 1;
  } // 4c1b
}; // d74c
```

graphs/bellman_ineq.h cd51

```
struct Ineq {
  ll a, b, c; // a - b >= c
}; // 663a

// Solve system of inequalities of form a-b>=c
// using Bellman-Ford; time: O(n*m)
bool solveIneq(vector<Ineq>& edges,
               vector<ll>& vars) {
  rep(i, 0, sz(vars)) each(e, edges)
    vars[e.b] = min(vars[e.b], vars[e.a]-e.c);
  each(e, edges)
    if (vars[e.a]-e.c < vars[e.b]) return 0;
  return 1;
} // 241e
```

graphs/biconnected.h 2b9e

```
// Biconnected components; time: O(n+m)
// Usage: Biconnected bi(graph);
// bi[v] = indices of components containing v
// bi.verts[i] = vertices of i-th component
// bi.edges[i] = edges of i-th component
// Bridges <=> components with 2 vertices
// Articulation points <=> vertices that belong
//                         to > 1 component
// Isolated vertex <=> empty component list
struct Biconnected : vector<Vi> {
  vector<Vi> verts;
  vector<vector<Pii>> edges;
  vector<Pii> S;

  Biconnected() {}

  Biconnected(vector<Vi>& G) : S(sz(G)) {
    resize(sz(G));
    rep(i, 0, sz(G)) S[i].x ?: dfs(G, i, -1);
    rep(c, 0, sz(verts)) each(v, verts[c])
      at(v).pb(c);
  } // cfce

  int dfs(vector<Vi>& G, int v, int p) {
    int low = S[v].x = sz(S)-1;
    S.pb({v, -1});

    each(e, G[v]) if (e != p) {
      if (S[e].x < S[v].x) S.pb({v, e});
      low = min(low, S[e].x ?: dfs(G, e, v));
    } // 446d

    if (p+1 && low >= S[p].x) {
      verts.pb({p}); edges.pb({});
      rep(i, S[v].x, sz(S)) {
        if (S[i].y == -1)
          verts.back().pb(S[i].x);
        else
          edges.back().pb(S[i]);
      } // 4fab
      S.resize(S[v].x);
    } // 6d66

    return low;
  } // 7fcc
}; // 4fa4
```

graphs/bip_edge_coloring.h 647c

```
// Bipartite edge coloring; time: O(nm)
// `edges` is list of (left vert, right vert),
// where vertices on both sides are indexed
// from 0 to n-1. Returns number of used colors
// (which is equal to max degree).
// col[i] = color of i-th edge [0..max_deg-1]
int colorEdges(vector<Pii>& edges,
               int n, Vi& col) {
  int m = sz(edges), c[2] = {}, ans = 0;
  Vi deg[2];
  vector<vector<Pii>> has[2];
  col.assign(m, 0);
  rep(i, 0, 2) {
    deg[i].resize(n+1);
    has[i].resize(n+1, vector<Pii>(n+1));
  } // b876

  function<void(int,int)> dfs =
    [&](int x, int p) {
      Pii i = has[p][x][c[!p]];
      if (has[!p][i.x][c[p]].y) dfs(i.x, !p);
      else has[!p][i.x][c[!p]] = {};
      has[p][x][c[p]] = i;
      has[!p][i.x][c[p]] = {x, i.y};
      if (i.y) col[i.y-1] = c[p]-1;
    }; // 19ad

  rep(i, 0, m) {
    int x[2] = {edges[i].x+1, edges[i].y+1};
    rep(d, 0, 2) {
      deg[d][x[d]]++;
      ans = max(ans, deg[d][x[d]]);
      for (c[d] = 1; has[d][x[d]][c[d]].y;)
        c[d]++;
    } // 9454
    if (c[0]-c[1]) dfs(x[1], 1);
    rep(d, 0, 2)
      has[d][x[d]][c[0]] = {x[!d], i+1};
    col[i] = c[0]-1;
  } // 5678
  return ans;
} // 1bfc
```

graphs/boski_matching.h dbd2

```
// Bosek's algorithm for partially online
// bipartite maximum matching - white vertices
// are fixed, black vertices are added
// one by one; time: O(E*sqrt(V))
// Usage: Matching match(num_white);
// match[v] = index of black vertex matched to
//            white vertex v or -1 if unmatched
// match.add(indices_of_white_neighbours);
// Black vertices are indexed in order they
// were added, the first black vertex is 0.
struct Matching : Vi {
  vector<Vi> adj;
  Vi rank, low, pos, vis, seen;
  int k{0};

  // Initialize structure for n white vertices
  Matching(int n = 0) : Vi(n, -1), rank(n) {}

  // Add new black vertex with its neighbours
  // given by `vec`. Returns true if maximum
  // matching is increased by 1.
  bool add(Vi vec) {
    adj.pb(move(vec));
    low.pb(0); pos.pb(0); vis.pb(0);
    if (!adj.back().empty()) {
      int i = k;
    nxt:
      seen.clear();
      if (dfs(sz(adj)-1, ++k-i)) return 1;
      each(v, seen) each(e, adj[v])
        if (rank[e] < 1e9 && vis[at(e)] < k)
          goto nxt;
      each(v, seen) each(w, adj[v])
        rank[w] = low[v] = 1e9;
    } // 6aec
    return 0;
  } // d2a7

  bool dfs(int v, int g) {
    if (vis[v] < k) vis[v] = k, seen.pb(v);
    while (low[v] < g) {
      int e = adj[v][pos[v]];
      if (at(e) != v && low[v] == rank[e]) {
        rank[e]++;
        if (at(e) == -1 || dfs(at(e), rank[e]))
          return at(e) = v, 1;
      } else if (++pos[v] == sz(adj[v])) {
        pos[v] = 0; low[v]++;
      } // e532
    } // 3d88
    return 0;
  } // 8561
}; // aac1
```

graphs/bridges_online.h 4124

```
// Dynamic 2-edge connectivity queries
// Usage: Bridges bridges(vertex_count);
// - bridges.addEdge(u, v); - add edge (u, v)
// - bridges.cc[v] = connected component ID
// - bridges.bi(v) = 2-edge connected comp ID
struct Bridges {
  vector<Vi> G; // Spanning forest
  Vi cc, size, par, bp, seen;
  int cnt{0};

  // Initialize structure for n vertices; O(n)
  Bridges(int n = 0) : G(n), cc(n), size(n, 1),
                       par(n, -1), bp(n, -1),
                       seen(n) {
    iota(all(cc), 0);
  } // ed70

  // Add edge (u, v); time: amortized O(lg n)
  void addEdge(int u, int v) {
    if (cc[u] == cc[v]) {
      int r = lca(u, v);
      for (int x : {u, v})
        while ((x = root(x)) != r)
          x = bp[bi(x)] = par[x];
    } else {
      G[u].pb(v); G[v].pb(u);
      if (size[cc[u]] > size[cc[v]]) swap(u,v);
      size[cc[v]] += size[cc[u]];
      dfs(u, v);
    } // abc7
  } // a6fd

  // Get 2-edge connected component ID
  int bi(int v) { // amortized time: < O(lg n)
    return bp[v] + 1 ? bp[v] = bi(bp[v]) : v;
  } // 3206

  int root(int v) {
    return par[v] == -1 || bi(par[v]) != bi(v)
      ? v : par[v] = root(par[v]);
  } // 2d27

  void dfs(int v, int p) {
    cc[v] = cc[par[v] = p];
    each(e, G[v]) if (e != p) dfs(e, v);
  } // 85f5

  int lca(int u, int v) { // Don't use this!
    for (cnt++;; swap(u, v)) if (u != -1) {
      if (seen[u = root(u)] == cnt) return u;
      seen[u] = cnt; u = par[u];
    } // afed
  } // 7f56
}; // bd70
```

graphs/dense_dfs.h 4fbd

```
#include "../math/bit_matrix.h"

// DFS over bit-packed adjacency matrix
// G = NxN adjacency matrix of graph
//     G(i,j) <=> (i,j) is edge
// V = 1xN matrix containing unvisited vertices
//     V(0,i) <=> i-th vertex is not visited
// Total DFS time: O(n^2/64)
struct DenseDFS {
  BitMatrix G, V; // space: O(n^2/64)

  // Initialize structure for n vertices
  DenseDFS(int n = 0) : G(n, n), V(1, n) {
    reset();
  } // 79e4

  // Mark all vertices as unvisited
  void reset() { each(x, V.M) x = -1; }

  // Get/set visited flag for i-th vertex
  void setVisited(int i) { V.set(0, i, 0); }
  bool isVisited(int i)  { return !V(0, i); }

  // DFS step: func is called on each unvisited
  // neighbour of i. You need to manually call
  // setVisited(child) to mark it visited
  // or this function will call the callback
  // with the same vertex again.
  template<class T>
  void step(int i, T func) {
    ull* E = G.row(i);
    for (int w = 0; w < G.stride;) {
      ull x = E[w] & V.row(0)[w];
      if (x) func((w<<6) | __builtin_ctzll(x));
      else   w++;
    } // 4c0a
  } // f045
}; // 8edb
```

graphs/directed_mst.h f935

```
#include "../structures/find_union_undo.h"

struct Edge {
  int a, b;
  ll w;
}; // 88dd

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;
  } // 3f0f
  Edge top() { prop(); return key; }
}; // 019f

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;
} // 34a3

void pop(Node*& a) {
  a->prop(); a = merge(a->l, a->r);
} // 666b

// Find directed minimum spanning tree
// rooted at vertex `root`; O(m log n)
// Returns weight of found spanning tree.
// out[i] = parent of i-th vertex in the tree,
// out[root] = -1
ll dmst(vector<Edge>& edges,
        int n, int root, Vi& out) {
  RollbackFAU uf(n);
  vector<Node*> heap(n);

  each(e, edges)
    heap[e.b] = merge(heap[e.b],
      new Node{e, 0, 0, 0});

  ll res = 0;
  Vi seen(n, -1), path(n);
  seen[root] = root;

  struct Cycle { int u, t; vector<Edge> e; };
  vector<Edge> Q(n), in(n, {-1, -1, 0}), comp;
  vector<Cycle> cycs;

  rep(s, 0, n) {
    int u = s, qi = 0, w;
    while (seen[u] < 0) {
      if (!heap[u]) return out.clear(), -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));
        heap[u = uf.find(u)] = cyc;
        seen[u] = -1;
        cycs.pb({u, time, {&Q[qi], &Q[end]}});
      } // 4ff6
    } // 05d8
    rep(i, 0, qi) in[uf.find(Q[i].b)] = Q[i];
  } // fc95

  reverse(all(cycs));
  each(c, cycs) {
    uf.rollback(c.t);
    Edge tmp = in[c.u];
    each(e, c.e) in[uf.find(e.b)] = e;
    in[uf.find(tmp.b)] = tmp;
  } // 957d
  out.resize(n);
  rep(i, 0, n) out[i] = in[i].a;
  return res;
} // b30d
```

graphs/dominators.h aec6

```
// Tarjan's algorithm for finding dominators
```

```
// in directed graph; time: O(m log n)
// Returns array of immediate dominators idom.
// idom[root] = root
// idom[v] = -1 if v is unreachable from root
Vi dominators(const vector<Vi>& G, int root) {
  int n = sz(G);
  vector<Vi> in(n), bucket(n);
  Vi pre(n, -1), anc(n, -1), par(n), best(n);
  Vi ord, idom(n, -1), sdom(n, n), rdom(n);

  function<void(int,int)> dfs =
    [&](int v, int p) {
      if (pre[v] == -1) {
        par[v] = p;
        pre[v] = sz(ord);
        ord.pb(v);
        each(e, G[v]) in[e].pb(v), dfs(e, v);
      } // 1182
    }; // ffd2

  function<Pii(int)> find = [&](int v) {
    if (anc[v] == -1) return mp(best[v], v);
    int b; tie(b, anc[v]) = find(anc[v]);
    if (sdom[b] < sdom[best[v]]) best[v] = b;
    return mp(best[v], anc[v]);
  }; // c07b

  rdom[root] = idom[root] = root;
  iota(all(best), 0);
  dfs(root, -1);

  rep(i, 0, sz(ord)) {
    int v = ord[sz(ord)-i-1], b = pre[v];
    each(e, in[v])
      b = min(b, pre[e] < pre[v] ? pre[e] :
              sdom[find(e).x]);
    each(u, bucket[v]) rdom[u] = find(u).x;
    sdom[v] = b;
    anc[v] = par[v];
    bucket[ord[sdom[v]]].pb(v);
  } // 54f4

  each(v, ord) idom[v] = (rdom[v] == v ?
    ord[sdom[v]] : idom[rdom[v]]);
  return idom;
} // 0656
```

graphs/edge_coloring.h 439b

```
// General graph edge coloring; time: O(nm)
// Finds (D+1)-edge-coloring of given graph,
// where D is max vertex degree.
// Returns vector of edge colors `col`.
// col[i] = color of i-th edge [0..D]
Vi vizing(vector<Pii>& edges, int n) {
  Vi cc(n+1), ret(sz(edges)),
     fan(n), fre(n), loc;
  each(e, edges) cc[e.x]++, cc[e.y]++;
  int u, v, cnt = *max_element(all(cc)) + 1;
  vector<Vi> adj(n, Vi(cnt, -1));
  each(e, edges) {
    tie(u, v) = e;
    fan[0] = v;
    loc.assign(cnt, 0);
    int at = u, end = u, d, c = fre[u],
        ind = 0, i = 0;
    while (d = fre[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 x = fan[i], y = fan[++i], f = cc[i];
      adj[u][f] = x; adj[x][f] = u;
      adj[y][f] = -1; fre[y] = f;
    } // 0024
    adj[u][d] = fan[i];
    adj[fan[i]][d] = u;
    for (int y : {fan[0], u, end})
      for (int& z = fre[y] = 0; adj[y][z]+1;)
        z++;
  } // 4240
  rep(i, 0, sz(edges))
    for (tie(u, v) = edges[i];
         adj[u][ret[i]] != v;) ++ret[i];
  return ret;
} // c259
```

graphs/edmonds_karp.h 8326

```
using flow_t = int;
constexpr flow_t INF = 1e9+10;

// Edmonds-Karp algorithm for finding
// maximum flow in graph; time: O(V*E^2)
struct MaxFlow {
  struct Edge {
    int dst, inv;
    flow_t flow, cap;
  }; // a53c

  vector<vector<Edge>> G;
  vector<flow_t> add;
  Vi prev;

  // Initialize for n vertices
  MaxFlow(int n = 0) : G(n) {}

  // Add new vertex
  int addVert() { G.pb({}); return sz(G)-1; }

  // Add edge from u to v with capacity cap
  // and reverse capacity rcap.
  // Returns edge index in adjacency list of u.
  int addEdge(int u, int v,
              flow_t cap, flow_t rcap = 0) {
    G[u].pb({ v, sz(G[v]), 0, cap });
    G[v].pb({ u, sz(G[u])-1, 0, rcap });
    return sz(G[u])-1;
  } // c96a

  // Compute maximum flow from src to dst.
  flow_t maxFlow(int src, int dst) {
    flow_t i, m, f = 0;
    each(v, G) each(e, v) e.flow = 0;

  nxt:
    queue<int> Q;
    Q.push(src);
    prev.assign(sz(G), -1);
    add.assign(sz(G), -1);
    add[src] = INF;

    while (!Q.empty()) {
      m = add[i = Q.front()];
      Q.pop();

      if (i == dst) {
        while (i != src) {
          auto& e = G[i][prev[i]];
          e.flow -= m;
          G[i = e.dst][e.inv].flow += m;
        } // 1f86
        f += m;
        goto nxt;
      } // 43a2

      each(e, G[i])
        if (add[e.dst] < 0 && e.flow < e.cap) {
          Q.push(e.dst);
          prev[e.dst] = e.inv;
          add[e.dst] = min(m, e.cap-e.flow);
        } // 4cdb
    } // 887e

    return f;
  } // cec0

  // Get flow through e-th edge of vertex v
  flow_t getFlow(int v, int e) {
    return G[v][e].flow;
  } // 0faf

  // Get if v belongs to cut component with src
  bool cutSide(int v) { return add[v] >= 0; }
}; // c5ef
```

graphs/flow_with_demands.h 0153

```
#include "edmonds_karp.h"
//#include "push_relabel.h" // if you need

// Flow with demands; time: O(maxflow)
struct FlowDemands {
  MaxFlow net;
  vector<vector<flow_t>> demands;
  flow_t total{0};

  // Initialize for k vertices
  FlowDemands(int k = 0) : net(2) {
    while (k--) addVert();
  } // 7bdf

  // Add new vertex
  int addVert() {
    int v = net.addVert();
    demands.pb({});
    net.addEdge(0, v, 0);
    net.addEdge(v, 1, 0);
    return v-2;
  } // 48b6

  // Add edge from u to v with demand dem
  // and capacity cap (dem <= flow <= cap).
  // Returns edge index in adjacency list of u.
  int addEdge(int u, int v,
              flow_t dem, flow_t cap) {
    demands[u].pb(dem);
    demands[v].pb(0);
    total += dem;
    net.G[0][v].cap += dem;
    net.G[u+2][1].cap += dem;
    return net.addEdge(u+2, v+2, cap-dem) - 2;
  } // a403

  // Check if there exists a flow with value f
  // for source src and destination dst.
  // For circulation, you can set args to 0.
  bool canFlow(int src, int dst, flow_t f) {
    net.addEdge(dst += 2, src += 2, f);
    f = net.maxFlow(0, 1);
    net.G[src].pop_back();
    net.G[dst].pop_back();
    return f == total;
  } // 6285

  // Get flow through e-th edge of vertex v
  flow_t getFlow(int v, int e) {
    return net.getFlow(v+2,e+2)+demands[v][e];
  } // 6cf6
}; // db37
```

graphs/general_matching.h f42c

```
// Edmond's Blossom algorithm for maximum
// matching in general graphs; time: O(nm)
// Returns matching size (edge count).
// match[v] = vert matched to v or -1
int blossom(vector<Vi>& G, Vi& match) {
  int n = sz(G), cnt = -1, ans = 0;
  match.assign(n, -1);
  Vi lab(n), par(n), orig(n), aux(n, -1), q;

  auto blos = [&](int v, int w, int a) {
    while (orig[v] != a) {
      par[v] = w; w = match[v];
      if (lab[w] == 1) lab[w] = 0, q.pb(w);
      orig[v] = orig[w] = a; v = par[w];
    } // 319e
  }; // ab9e

  rep(i, 0, n) if (match[i] == -1)
    each(e, G[i]) if (match[e] == -1) {
      match[match[e] = i] = e; ans++; break;
    } // a22a

  rep(root, 0, n) if (match[root] == -1) {
    fill(all(lab), -1);
    iota(all(orig), 0);
    lab[root] = 0;
    q = {root};
    rep(i, 0, sz(q)) {
      int v = q[i];
      each(x, G[v]) if (lab[x] == -1) {
        lab[x] = 1; par[x] = v;
        if (match[x] == -1) {
          for (int y = x; y+1;) {
            int p = par[y], w = match[p];
            match[match[p] = y] = p; y = w;
          } // 30c1
          ans++;
          goto nxt;
        } // 6fd0
        lab[match[x]] = 0; q.pb(match[x]);
      } else if (lab[x] == 0 &&
                 orig[v] != orig[x]) {
        int a = orig[v], b = orig[x];
        for (cnt++;; swap(a, b)) if (a+1) {
          if (aux[a] == cnt) break;
          aux[a] = cnt;
          a = (match[a]+1 ?
            orig[par[match[a]]] : -1);
        } // 2776
        blos(x, v, a); blos(v, x, a);
      } // 45a1
    } // d488
  nxt:;
  } // 8d8a

  return ans;
} // d872
```

graphs/general_matching_w.h 8fd8

```
// Edmond's Blossom algorithm for weighted
// maximum matching in general graphs; O(n^3)?
// Weights must be positive (I believe).
struct WeightedBlossom {
  struct edge { int u, v, w; };
  int n, s, nx;
```

```
vector<vector<edge>> g;
Vi lab, match, slack, st, pa, S, vis;
vector<Vi> flo, floFrom;
queue<int> q;

// Initialize for k vertices
WeightedBlossom(int k)
    : n(k), s(n*2+1),
      g(s, vector<edge>(s)),
      lab(s), match(s), slack(s), st(s),
      pa(s), S(s), vis(s), flo(s),
      floFrom(s, Vi(n+1)) {
  rep(u, 1, n+1) rep(v, 1, n+1)
    g[u][v] = {u, v, 0};
} // 5e51

// Add edge between u and v with weight w
void addEdge(int u, int v, int w) {
  u++; v++;
  g[u][v].w = g[v][u].w = max(g[u][v].w, w);
} // d296

// Compute max weight matching.
// `count` is set to matching size,
// `weight` is set to matching weight.
// Returns vector `match` such that:
// match[v] = vert matched to v or -1
Vi solve(int& count, ll& weight) {
  fill(all(match), 0);
  nx = n;
  weight = count = 0;
  rep(u, 0, n+1) flo[st[u] = u].clear();
  int tmp = 0;
  rep(u, 1, n+1) rep(v, 1, n+1) {
    floFrom[u][v] = (u-v ? 0 : v);
    tmp = max(tmp, g[u][v].w);
  } // a881
  rep(u, 1, n+1) lab[u] = tmp;
  while (matching()) count++;
  rep(u, 1, n+1)
    if (match[u] && match[u] < u)
      weight += g[u][match[u]].w;
  Vi ans(n);
  rep(i, 0, n) ans[i] = match[i+1]-1;
  return ans;
} // d611

int delta(edge& e) {
  return lab[e.u]+lab[e.v]-g[e.u][e.v].w*2;
} // 7b58
void updateSlack(int u, int x) {
  if (!slack[x] || delta(g[u][x]) <
    delta(g[slack[x]][x])) slack[x] = u;
} // 1f7f
void setSlack(int x) {
  slack[x] = 0;
  rep(u, 1, n+1) if (g[u][x].w > 0 &&
    st[u] != x && !S[st[u]])
      updateSlack(u, x);
} // ee9c
void push(int x) {
  if (x <= n) q.push(x);
  else rep(i, 0, sz(flo[x])) push(flo[x][i]);
} // 594d
void setSt(int x, int b) {
  st[x] = b;
  if (x > n) rep(i, 0, sz(flo[x]))
    setSt(flo[x][i],b);
} // c5c8
int getPr(int b, int xr) {
  int pr = int(find(all(flo[b]), xr) -
    flo[b].begin());
  if (pr % 2) {
    reverse(flo[b].begin()+1, flo[b].end());
    return sz(flo[b]) - pr;
  } else return pr;
} // 399f
void setMatch(int u, int v) {
  match[u] = g[u][v].v;
  if (u <= n) return;
  edge e = g[u][v];
  int xr = floFrom[u][e.u], pr = getPr(u,xr);
  rep(i, 0, pr)
    setMatch(flo[u][i], flo[u][i^1]);
  setMatch(xr, v);
  rotate(flo[u].begin(), flo[u].begin()+pr,
    flo[u].end());
} // f19d
void augment(int u, int v) {
  while (1) {
    int xnv = st[match[u]];
    setMatch(u, v);
    if (!xnv) return;
    setMatch(xnv, st[pa[xnv]]);
    u = st[pa[xnv]], v = xnv;
  } // bd23
} // e61a
int getLca(int u, int v) {
  static int t = 0;
  for (++t; u||v; swap(u, v)) {
    if (!u) continue;
    if (vis[u] == t) return u;
    vis[u] = t;
    u = st[match[u]];
    if (u) u = st[pa[u]];
  } // aa78
  return 0;
} // 9f28
void blossom(int u, int lca, int v) {
  int b = n+1;
  while (b <= nx && st[b]) ++b;
  if (b > nx) ++nx;
  lab[b] = S[b] = 0;
  match[b] = match[lca];
  flo[b].clear();
  flo[b].pb(lca);
  for (int x=u, y; x != lca; x = st[pa[y]]) {
    flo[b].pb(x);
    flo[b].pb(y = st[match[x]]);
    push(y);
  } // 63e0
  reverse(flo[b].begin()+1, flo[b].end());
  for (int x=v, y; x != lca; x = st[pa[y]]) {
    flo[b].pb(x);
    flo[b].pb(y = st[match[x]]);
    push(y);
  } // 63e0
  setSt(b, b);
  rep(x, 1, nx+1) g[b][x].w = g[x][b].w = 0;
  rep(x, 1, n+1) floFrom[b][x] = 0;
  rep(i, 0, sz(flo[b])) {
    int xs = flo[b][i];
    rep(x, 1, nx+1) if (!g[b][x].w ||
      delta(g[xs][x]) < delta(g[b][x]))
        g[b][x]=g[xs][x], g[x][b]=g[x][xs];
    rep(x, 1, n+1) if (floFrom[xs][x])
      floFrom[b][x] = xs;
  } // 5833
  setSlack(b);
} // 9000
void blossom(int b) {
  each(e, flo[b]) setSt(e, e);
  int xr = floFrom[b][g[b][pa[b]].u];
  int pr = getPr(b, xr);
  for (int i = 0; i < pr; i += 2) {
    int xs = flo[b][i], xns = flo[b][i+1];
    pa[xs] = g[xns][xs].u;
    S[xs] = 1; S[xns] = slack[xs] = 0;
    setSlack(xns); push(xns);
  } // f26f
  S[xr] = 1; pa[xr] = pa[b];
  rep(i, pr+1, sz(flo[b])) {
    int xs = flo[b][i];
    S[xs] = -1; setSlack(xs);
  } // a12a
  st[b] = 0;
} // f750
bool found(const edge& e) {
  int u = st[e.u], v = st[e.v];
  if (S[v] == -1) {
    pa[v] = e.u; S[v] = 1;
    int nu = st[match[v]];
    slack[v] = slack[nu] = S[nu] = 0;
    push(nu);
  } else if (!S[v]) {
    int lca = getLca(u, v);
    if (!lca) return augment(u, v),
      augment(v, u), 1;
    else blossom(u, lca, v);
  } // ddbb
  return 0;
} // 1c00
bool matching() {
  fill(S.begin(), S.begin()+nx+1, -1);
  fill(slack.begin(), slack.begin()+nx+1, 0);
  q = {};
  rep(x, 1, nx+1)
    if (st[x] == x && !match[x])
      pa[x] = S[x] = 0, push(x);
  if (q.empty()) return 0;
  while (1) {
    while (q.size()) {
      int u = q.front(); q.pop();
      if (S[st[u]] == 1) continue;
      rep(v, 1, n+1)
        if (g[u][v].w > 0 && st[u] != st[v]){
          if (!delta(g[u][v])) {
            if (found(g[u][v])) return 1;
          } else updateSlack(u, st[v]);
        } // b782
    } // 4d33
    int d = INT_MAX;
    rep(b, n+1, nx+1)
      if (st[b] == b && S[b] == 1)
        d = min(d, lab[b]/2);
    rep(x, 1, nx+1)
      if (st[x] == x && slack[x]) {
        if (S[x] == -1)
          d = min(d, delta(g[slack[x]][x]));
        else if (!S[x])
          d = min(d,delta(g[slack[x]][x])/2);
      } // 2a0e
    rep(u, 1, n+1) {
      if (!S[st[u]]) {
        if (lab[u] <= d) return 0;
        lab[u] -= d;
      } else if (S[st[u]] == 1) lab[u] += d;
    } // 4601
    rep(b, n+1, nx+1) if (st[b] == b) {
      if (!S[st[b]]) lab[b] += d*2;
      else if (S[st[b]] == 1) lab[b] -= d*2;
    } // e09b
    q = {};
    rep(x, 1, nx+1)
      if (st[x] == x && slack[x] &&
        st[slack[x]] != x &&
        !delta(g[slack[x]][x]) &&
        found(g[slack[x]][x])) return 1;
    rep(b, n+1, nx+1)
      if (st[b] == b && S[b] == 1 && !lab[b])
        blossom(b);
  } // a122
  return 0;
} // e966
}; // 3ee2
```

graphs/global_min_cut.h 0eac

```
// Find a minimum cut in an undirected graph
// with non-negative edge weights
// given its adjacency matrix M; time: O(n^3)
// `out` contains vertices on one side.
ll minCut(vector<vector<ll>> M, Vi& out) {
  int n = sz(M);
  ll ans = INT64_MAX;
  vector<Vi> co(n);
  rep(i, 0, n) co[i].pb(i);
  out.clear();
  rep(ph, 1, n) {
    auto w = M[0];
    size_t s = 0, t = 0;
    // O(V^2) -> O(E log V) with priority queue
    rep(it, 0, n-ph) {
      w[t] = INT64_MIN; s = t;
      t = max_element(all(w)) - w.begin();
      rep(i, 0, n) w[i] += M[t][i];
    } // 0831
    ll alt = w[t] - M[t][t];
    if (alt < ans) ans = alt, out = co[t];
    co[s].insert(co[s].end(), all(co[t]));
    rep(i, 0, n) M[s][i] += M[t][i];
    rep(i, 0, n) M[i][s] = M[s][i];
    M[0][t] = INT64_MIN;
  } // df69
  return ans;
} // 2a85
```

graphs/gomory_hu.h a520

```
#include "edmonds_karp.h"
//#include "push_relabel.h" // if you need

struct Edge {
  int a, b; // vertices
  flow_t w; // weight
}; // c331

// Build Gomory-Hu tree; time: O(n*maxflow)
// Gomory-Hu tree encodes minimum cuts between
// all pairs of vertices: mincut for u and v
// is equal to minimum on path from u and v
// in Gomory-Hu tree. n is vertex count.
// Returns vector of Gomory-Hu tree edges.
vector<Edge> gomoryHu(vector<Edge>& edges,
                      int n) {
  MaxFlow flow(n);
  each(e, edges) flow.addEdge(e.a,e.b,e.w,e.w);
```

```
  vector<Edge> ret(n-1);
  rep(i, 1, n) ret[i-1] = {i, 0, 0};

  rep(i, 1, n) {
    ret[i-1].w = flow.maxFlow(i, ret[i-1].b);
    rep(j, i+1, n)
      if (ret[j-1].b == ret[i-1].b &&
          flow.cutSide(j)) ret[j-1].b = i;
  } // 5ae4

  return ret;
} // afdb
```

graphs/kth_shortest.h 1f40

```
constexpr ll INF = 1e18;

// Eppstein's k-th shortest path algorithm;
// time and space: O((m+k) log (m+k))
struct Eppstein {
  using T = ll; // Type for edge weights
  using Edge = pair<int, T>;

  struct Node {
    int E[2] = {}, s{0};
    Edge x;
  }; // 013b

  T shortest; // Shortest path length
  priority_queue<pair<T, int>> Q;
  vector<Node> P{1};
  Vi h;

  // Initialize shortest path structure for
  // weighted graph G, source s and target t;
  // time: O(m log m)
  Eppstein(vector<vector<Edge>>& G,
           int s, int t) {
    int n = sz(G);
    vector<vector<Edge>> H(n);
    rep(i,0,n) each(e,G[i]) H[e.x].pb({i,e.y});

    Vi ord, par(n, -1);
    vector<T> d(n, -INF);
    Q.push({d[t] = 0, t});

    while (!Q.empty()) {
      auto v = Q.top();
      Q.pop();
      if (d[v.y] == v.x) {
        ord.pb(v.y);
        each(e, H[v.y]) if (v.x-e.y > d[e.x]) {
          Q.push({d[e.x] = v.x-e.y, e.x});
          par[e.x] = v.y;
        } // 5895
      } // 1b62
    } // 1a6d

    if ((shortest = -d[s]) >= INF) return;
    h.resize(n);

    each(v, ord) {
      int p = par[v];
      if (p+1) h[v] = h[p];
      each(e, G[v]) if (d[e.x] > -INF) {
        T k = e.y - d[e.x] + d[v];
        if (k || e.x != p)
          h[v] = push(h[v], {e.x, k});
        else
          p = -1;
      } // 5e05
    } // 31b9

    P[0].x.x = s;
    Q.push({0, 0});
  } // e00e

  int push(int t, Edge x) {
    P.pb(P[t]);
    if (!P[t = sz(P)-1].s || P[t].x.y >= x.y)
      swap(x, P[t].x);
    if (P[t].s) {
      int i = P[t].E[0], j = P[t].E[1];
      int d = P[i].s > P[j].s;
      int k = push(d ? j : i, x);
      P[t].E[d] = k; // Don't inline k!
    } // 10e1
    P[t].s++;
    return t;
  } // a2dc

  // Get next shortest path length,
  // the first call returns shortest path.
  // Returns -1 if there's no more paths;
  // time: O(log k), where k is total count
  // of nextPath calls.
  ll nextPath() {
    if (Q.empty()) return -1;
    auto v = Q.top();
    Q.pop();
    for (int i : P[v.y].E) if (i)
      Q.push({ v.x-P[i].x.y+P[v.y].x.y, i });
    int t = h[P[v.y].x.x];
    if (t) Q.push({ v.x - P[t].x.y, t });
    return shortest - v.x;
  } // 08af
}; // 5326
```

graphs/matroids.h 55ef

```
// Find largest subset S of [n] such that
// S is independent in both matroid A and B.
// A and B are given by their oracles,
// see example implementations below.
// Returns vector V such that V[i] = 1 iff
// i-th element is included in found set;
// time: O(r^2*init + r^2*n*add),
// where r is max independent set,
// `init` is max time of oracles init
// and `add` is max time of oracles canAdd.
template<class T, class U>
vector<bool> intersectMatroids(T& A, U& B,
                               int n) {
  vector<bool> ans(n);
  bool ok = 1;

  // NOTE: for weighted matroid intersection
  // find shortest augmenting paths
  // first by weight change, then by length
  // using Bellman-Ford, and skip this speedup:
  A.init(ans);
  B.init(ans);
  rep(i, 0, n) if (A.canAdd(i) && B.canAdd(i))
    ans[i] = 1, A.init(ans), B.init(ans);

  while (ok) {
    vector<Vi> G(n);
    vector<bool> good(n);
    queue<int> que;
    Vi prev(n, -1);

    A.init(ans);
    B.init(ans);
    ok = 0;

    rep(i, 0, n) if (!ans[i]) {
      if (A.canAdd(i)) que.push(i), prev[i]=-2;
      good[i] = B.canAdd(i);
    } // 9581

    rep(i, 0, n) if (ans[i]) {
      ans[i] = 0;
      A.init(ans);
      B.init(ans);
      rep(j, 0, n) if (i != j && !ans[j]) {
        if (A.canAdd(j)) G[i].pb(j);
        if (B.canAdd(j)) G[j].pb(i);
      } // bd2a
      ans[i] = 1;
    } // bf3e

    while (!que.empty()) {
      int i = que.front();
      que.pop();

      if (good[i]) {
        ans[i] = 1;
        while (prev[i] >= 0) {
          ans[i = prev[i]] = 0;
          ans[i = prev[i]] = 1;
        } // 51c8
        ok = 1;
        break;
      } // 384b

      each(j, G[i]) if (prev[j] == -1)
        que.push(j), prev[j] = i;
    } // 6eb6
  } // e092

  return ans;
} // ae8e

// Matroid where each element has color
// and set is independent iff for each color c
// #{elements of color c} <= maxAllowed[c].
struct LimOracle {
  Vi color; // color[i] = color of i-th element
  Vi maxAllowed; // Limits for colors
  Vi tmp;

  // Init oracle for independent set S; O(n)
  void init(vector<bool>& S) {
    tmp = maxAllowed;
    rep(i, 0, sz(S)) tmp[color[i]] -= S[i];
  } // 4dfb

  // Check if S+{k} is independent; time: O(1)
  bool canAdd(int k) {
    return tmp[color[k]] > 0;
  } // e312
}; // 7b5d

// Graphic matroid - each element is edge,
// set is independent iff subgraph is acyclic.
struct GraphOracle {
  vector<Pii> elems; // Ground set: graph edges
  int n; // Number of vertices, indexed [0;n-1]
  Vi par;

  int find(int i) {
    return par[i] == -1 ? i
      : par[i] = find(par[i]);
  } // b8b7

  // Init oracle for independent set S; ~O(n)
  void init(vector<bool>& S) {
    par.assign(n, -1);
    rep(i, 0, sz(S)) if (S[i])
      par[find(elems[i].x)] = find(elems[i].y);
  } // 1827

  // Check if S+{k} is independent; time: ~O(1)
  bool canAdd(int k) {
    return
      find(elems[k].x) != find(elems[k].y);
  } // 8ca4
}; // c506

// Co-graphic matroid - each element is edge,
// set is independent iff after removing edges
// from graph number of connected components
// doesn't change.
struct CographOracle {
  vector<Pii> elems; // Ground set: graph edges
  int n; // Number of vertices, indexed [0;n-1]
  vector<Vi> G;
  Vi pre, low;
  int cnt;

  int dfs(int v, int p) {
    pre[v] = low[v] = ++cnt;
    each(e, G[v]) if (e != p)
      low[v] = min(low[v], pre[e] ?: dfs(e,v));
    return low[v];
  } // 9d30

  // Init oracle for independent set S; O(n)
  void init(vector<bool>& S) {
    G.assign(n, {});
    pre.assign(n, 0);
    low.resize(n);
    cnt = 0;
    rep(i, 0, sz(S)) if (!S[i]) {
      Pii e = elems[i];
      G[e.x].pb(e.y);
      G[e.y].pb(e.x);
    } // 79a1
    rep(v, 0, n) if (!pre[v]) dfs(v, -1);
  } // 1200

  // Check if S+{k} is independent; time: O(1)
  bool canAdd(int k) {
    Pii e = elems[k];
    return max(pre[e.x], pre[e.y])
      != max(low[e.x], low[e.y]);
  } // 2550
}; // a5cc

// Matroid equivalent to linear space with XOR
struct XorOracle {
  vector<ll> elems; // Ground set: numbers
  vector<ll> base;

  // Init for independent set S; O(n+r^2)
  void init(vector<bool>& S) {
    base.assign(63, 0);
    rep(i, 0, sz(S)) if (S[i]) {
      ll e = elems[i];
      rep(j, 0, sz(base)) if ((e >> j) & 1) {
        if (!base[j]) {
          base[j] = e;
          break;
        } // 1df5
        e ^= base[j];
      } // 8495
    } // 655e
  } // b68c

  // Check if S+{k} is independent; time: O(r)
  bool canAdd(int k) {
    ll e = elems[k];
    rep(i, 0, sz(base)) if ((e >> i) & 1) {
```

```
    if (!base[i]) return 1;
    e ^= base[i];
   } // 49d1
   return 0;
  } // 66ff
}; // 4af3
```

graphs/min_cost_max_flow.h 1fa6

```
using flow_t = ll;
constexpr flow_t INF = 1e18;

// Min cost max flow using cheapest paths;
// time: O(nm + |f|*(m log n))
// or O(|f|*(m log n)) if costs are nonnegative
struct MCMF {
  struct Edge {
    int dst, inv;
    flow_t flow, cap, cost;
  }; // 20f7

  vector<vector<Edge>> G;
  vector<flow_t> add;

  // Initialize for n vertices
  MCMF(int n = 0) : G(n) {}

  // Add new vertex
  int addVert() { G.pb({}); return sz(G)-1; }

  // Add edge from u to v.
  // Returns edge index in adjacency list of u.
  int addEdge(int u, int v,
              flow_t cap, flow_t cost) {
    G[u].pb({ v, sz(G[v]), 0, cap, cost });
    G[v].pb({ u, sz(G[u])-1, 0, 0, -cost });
    return sz(G[u])-1;
  } // 1095

  // Compute minimum cost maximum flow
  // from src to dst. `f` is set to flow value,
  // `c` is set to total cost value.
  // Returns false iff negative cycle
  // is reachable from from source.
  bool maxFlow(int src, int dst,
               flow_t& f, flow_t& c) {
    flow_t i, m, d;
    f = c = 0;
    each(v, G) each(e, v) e.flow = 0;

    // [If costs are nonnegative]
    // vector<flow_t> pot(sz(G));
    // [/end]

    // [If costs can be negative] O(n*m)
    vector<flow_t> pot(sz(G), INF);
    pot[src] = 0;
    int it = sz(G), ch = 1;
    while (ch-- && it--)
      rep(s, 0, sz(G)) if (pot[s] != INF)
        each(e, G[s]) if (e.cap)
          if ((d = pot[s]+e.cost) < pot[e.dst])
            pot[e.dst] = d, ch = 1;
    if (it < 0) return 0;
    // [/end]

  nxt:
    Vi prev(sz(G), -1);
    vector<flow_t> dist(sz(G), INF);
    priority_queue<pair<flow_t, int>> Q;
    add.assign(sz(G), -1);
    Q.push({0, src});
    add[src] = INF;
    dist[src] = 0;

    while (!Q.empty()) {
      tie(d, i) = Q.top();
      Q.pop();
      if (d != -dist[i]) continue;
      m = add[i];

      if (i == dst) {
        f += m;
        c += m * (dist[i]-pot[src]+pot[i]);
        while (i != src) {
          auto& e = G[i][prev[i]];
          e.flow -= m;
          G[i = e.dst][e.inv].flow += m;
        } // 1f86
        rep(j, 0, sz(G))
          pot[j] = min(pot[j]+dist[j], INF);
        goto nxt;
      } // 36d4

      each(e, G[i]) if (e.flow < e.cap) {
        d = dist[i]+e.cost+pot[i]-pot[e.dst];
        if (d < dist[e.dst]) {
          Q.push({-d, e.dst});
          prev[e.dst] = e.inv;
          add[e.dst] = min(m, e.cap-e.flow);
          dist[e.dst] = d;
        } // 5ee6
      } // b6b2
    } // d47c
    return 1;
  } // 0bc3

  // Get flow through e-th edge of vertex v
  flow_t getFlow(int v, int e) {
    return G[v][e].flow;
  } // 0faf

  // Get if v belongs to cut component with src
  bool cutSide(int v) { return add[v] >= 0; }
}; // 1867
```

graphs/push_relabel.h 07d1

```
using flow_t = int;

// Push-relabel algorithm for maximum flow;
// O(V^2*sqrt(E)), but very fast in practice.
struct MaxFlow {
  struct Edge {
    int to, inv;
    flow_t rem, cap;
  }; // bc77

  vector<basic_string<Edge>> G;
  vector<flow_t> extra;
  Vi hei, arc, prv, nxt, act, bot;
  queue<int> Q;
  int n, high, cut, work;

  // Initialize for k vertices
  MaxFlow(int k = 0) : G(k) {}

  // Add new vertex
  int addVert() { G.pb({}); return sz(G)-1; }

  // Add edge from u to v with capacity cap
  // and reverse capacity rcap.
  // Returns edge index in adjacency list of u.
  int addEdge(int u, int v,
              flow_t cap, flow_t rcap = 0) {
    G[u].pb({ v, sz(G[v]), 0, cap });
    G[v].pb({ u, sz(G[u])-1, 0, rcap });
    return sz(G[u])-1;
  } // c96a

  void raise(int v, int h) {
    prv[nxt[prv[v]] = nxt[v]] = prv[v];
    hei[v] = h;
    if (extra[v] > 0) {
      bot[v] = act[h]; act[h] = v;
      high = max(high, h);
    } // d7ee
    if (h < n) cut = max(cut, h+1);
    nxt[v] = nxt[prv[v] = h += n];
    prv[nxt[nxt[h] = v]] = v;
  } // 5274

  void global(int s, int t) {
    hei.assign(n, n*2);
    act.assign(n*2, -1);
    iota(all(prv), 0);
    iota(all(nxt), 0);
    hei[t] = high = cut = work = 0;
    hei[s] = n;
    for (int x : {t, s})
      for (Q.push(x); !Q.empty(); Q.pop()) {
        int v = Q.front();
        each(e, G[v])
          if (hei[e.to] == n*2 &&
              G[e.to][e.inv].rem)
            Q.push(e.to), raise(e.to,hei[v]+1);
      } // 1901
  } // 3181

  void push(int v, Edge& e, bool z) {
    auto f = min(extra[v], e.rem);
    if (f > 0) {
      if (z && !extra[e.to]) {
        bot[e.to] = act[hei[e.to]];
        act[hei[e.to]] = e.to;
      } // 9d90
      e.rem -= f; G[e.to][e.inv].rem += f;
      extra[v] -= f; extra[e.to] += f;
    } // 0ffb
  } // da44

  void discharge(int v) {
    int h = n*2, k = hei[v];

    rep(j, 0, sz(G[v])) {
      auto& e = G[v][arc[v]];
      if (e.rem) {
        if (k == hei[e.to]+1) {
          push(v, e, 1);
          if (extra[v] <= 0) return;
        } else h = min(h, hei[e.to]+1);
      } // 87c1
      if (++arc[v] >= sz(G[v])) arc[v] = 0;
    } // 9741

    if (k < n && nxt[k+n] == prv[k+n]) {
      rep(j, k, cut) while (nxt[j+n] < n)
        raise(nxt[j+n], n);
      cut = k;
    } else raise(v, h), work++;
  } // b64f

  // Compute maximum flow from src to dst
  flow_t maxFlow(int src, int dst) {
    extra.assign(n = sz(G), 0);
    arc.assign(n, 0);
    prv.resize(n*3);
    nxt.resize(n*3);
    bot.resize(n);
    each(v, G) each(e, v) e.rem = e.cap;

    each(e, G[src])
      extra[src] = e.cap, push(src, e, 0);
    global(src, dst);

    for (; high; high--)
      while (act[high] != -1) {
        int v = act[high];
        act[high] = bot[v];
        if (v != src && hei[v] == high) {
          discharge(v);
          if (work > 4*n) global(src, dst);
        } // 7dcc
      } // 26d4

    return extra[dst];
  } // aa5e

  // Get flow through e-th edge of vertex v
  flow_t getFlow(int v, int e) {
    return G[v][e].cap - G[v][e].rem;
  } // 812c

  // Get if v belongs to cut component with src
  bool cutSide(int v) { return hei[v] >= n; }
}; // 2d6b
```

graphs/scc.h 1c43

```
// Tarjan's SCC algorithm; time: O(n+m)
// Usage: SCC scc(graph);
// scc[v] = index of SCC for vertex v
// scc.comps[i] = vertices of i-th SCC
// Components are in reversed topological order
struct SCC : Vi {
  vector<Vi> comps;
  Vi S;

  SCC() {}

  SCC(vector<Vi>& G) : Vi(sz(G),-1), S(sz(G)) {
    rep(i, 0, sz(G)) if (!S[i]) dfs(G, i);
  } // f0fa

  int dfs(vector<Vi>& G, int v) {
    int low = S[v] = sz(S);
    S.pb(v);

    each(e, G[v]) if (at(e) < 0)
      low = min(low, S[e] ?: dfs(G, e));

    if (low == S[v]) {
      comps.pb({});
      rep(i, S[v], sz(S)) {
        at(S[i]) = sz(comps)-1;
        comps.back().pb(S[i]);
      } // 8ed0
      S.resize(S[v]);
    } // ecc7

    return low;
  } // f3c6
}; // 215e
```

graphs/turbo_matching.h d400

```
// Find maximum bipartite matching; time: ?
// G must be bipartite graph!
// Returns matching size (edge count).
// match[v] = vert matched to v or -1
int matching(vector<Vi>& G, Vi& match) {
  vector<bool> seen;
  int n = 0, k = 1;
  match.assign(sz(G), -1);

  function<int(int)> dfs = [&](int i) {
```

```
    if (seen[i]) return 0;
    seen[i] = 1;
    each(e, G[i]) {
      if (match[e] < 0 || dfs(match[e])) {
        match[i] = e; match[e] = i;
        return 1;
      } // 893d
    } // 9532
    return 0;
  }; // d332

  while (k) {
    seen.assign(sz(G), 0);
    k = 0;
    rep(i, 0, sz(G)) if (match[i] < 0)
      k += dfs(i);
    n += k;
  } // 1128
  return n;
} // 0d38

// Convert maximum matching to vertex cover
// time: O(n+m)
Vi vertexCover(vector<Vi>& G, Vi& match) {
  Vi ret, col(sz(G)), seen(sz(G));

  function<void(int, int)> dfs =
      [&](int i, int c) {
    if (col[i]) return;
    col[i] = c+1;
    each(e, G[i]) dfs(e, !c);
  }; // 1f1b

  function<void(int)> aug = [&](int i) {
    if (seen[i] || col[i] != 1) return;
    seen[i] = 1;
    each(e, G[i]) seen[e] = 1, aug(match[e]);
  }; // 2465

  rep(i, 0, sz(G)) dfs(i, 0);
  rep(i, 0, sz(G)) if (match[i] < 0) aug(i);
  rep(i, 0, sz(G))
    if (seen[i] == col[i]-1) ret.pb(i);
  return ret;
} // 6f72
```

graphs/weighted_matching.h 8264

```
// Minimum cost bipartite matching; O(n^2*m)
// Input is n x m cost matrix, where n <= m.
// Returns matching weight.
// L[i] = right vertex matched to i-th left
// R[i] = left vertex matched to i-th right
ll hungarian(const vector<vector<ll>>& cost,
             Vi& L, Vi& R) {
  if (cost.empty())
    return L.clear(), R.clear(), 0;
  int b, c = 0, n = sz(cost), m = sz(cost[0]);
  assert(n <= m);

  vector<ll> x(n), y(m+1);
  L.assign(n, -1);
  R.assign(m+1, -1);

  rep(i, 0, n) {
    vector<ll> sla(m, INT64_MAX);
    Vi vis(m+1), prv(m, -1);
    for (R[b = m] = i; R[b]+1; b = c) {
      int a = R[b];
      ll d = INT64_MAX;
      vis[b] = 1;
      rep(j, 0, m) if (!vis[j]) {
        ll cur = cost[a][j] - x[a] - y[j];
        if (cur < sla[j])
          sla[j] = cur, prv[j] = b;
        if (sla[j] < d) d = sla[j], c = j;
      } // 6717
      rep(j, 0, m+1) {
        if (vis[j]) x[R[j]] += d, y[j] -= d;
        else sla[j] -= d;
      } // 8bb3
    } // 01c6
    while (b-m) c = b, R[c] = R[b = prv[b]];
  } // 71f5

  rep(j, 0, m) if (R[j]+1) L[R[j]] = j;
  R.resize(m);
  return -y[m];
} // 349d
```

math/berlekamp_massey.h 7d12

```
constexpr int MOD = 998244353;

ll modInv(ll a, ll m) { // a^(-1) mod m
  if (a == 1) return 1;
  return ((a - modInv(m%a, a))*m + 1) / a;
} // c437

// Find shortest linear recurrence that matches
// given starting terms of recurrence; O(n^2)
// Returns vector C such that for each i >= |C|
// A[i] = sum A[i-j-1]*C[j] for j = 0..|C|-1
vector<ll> massey(vector<ll>& A) {
  if (A.empty()) return {};
  int n = sz(A), len = 0, k = 0;
  ll s = 1;
  vector<ll> B(n), C(n), tmp;
  B[0] = C[0] = 1;

  rep(i, 0, n) {
    ll d = 0;
    k++;
    rep(j, 0, len+1)
      d = (d + C[j] * A[i-j]) % MOD;

    if (d) {
      ll q = d * modInv(s, MOD) % MOD;
      tmp = C;

      rep(j, k, n)
        C[j] = (C[j] - q * B[j-k]) % MOD;

      if (len*2 <= i) {
        B.swap(tmp);
        len = i-len+1;
        s = d + (d < 0) * MOD;
        k = 0;
      } // c350
    } // 79c7
  } // f70c

  C.resize(len+1);
  C.erase(C.begin());
  each(x, C) x = (MOD - x) % MOD;
  return C;
} // 20ce
```

math/bit_gauss.h 13eb

```
constexpr int MAX_COLS = 2048;

// Solve system of linear equations over Z_2
// time: O(n^2*m/W), where W is word size
// - A - extended matrix, rows are equations,
//       columns are variables,
//       m-th column is equation result
//       (A[i][j] - i-th row and j-th column)
// - ans - output for variables values
// - m - variable count
// Returns 0 if no solutions found, 1 if one,
// 2 if more than 1 solution exist.
int bitGauss(vector<bitset<MAX_COLS>>& A,
             vector<bool>& ans, int m) {
  Vi col;
  ans.assign(m, 0);

  rep(i, 0, sz(A)) {
    int c = int(A[i]._Find_first());
    if (c >= m) {
      if (c == m) return 0;
      continue;
    } // a6bb

    rep(k, i+1, sz(A)) if (A[k][c]) A[k]^=A[i];
    swap(A[i], A[sz(col)]);
    col.pb(c);
  } // a953

  for (int i = sz(col); i--;) if (A[i][m]) {
    ans[col[i]] = 1;
    rep(k,0,i) if(A[k][col[i]]) A[k][m].flip();
  } // 4ca1

  return sz(col) < m ? 2 : 1;
} // 996e
```

math/bit_matrix.h 2e3f

```
using ull = uint64_t;

// Matrix over Z_2 (bits and xor)
// TODO: arithmetic operations
struct BitMatrix {
  vector<ull> M;
  int rows, cols, stride;

  // Create matrix with n rows and m columns
  BitMatrix(int n = 0, int m = 0) {
    rows = n; cols = m;
    stride = (m+63)/64;
    M.resize(n*stride);
  } // 7ef0

  // Get pointer to bit-packed data of i-th row
  ull* row(int i) { return &M[i*stride]; }

  // Get value in i-th row and j-th column
  bool operator()(int i, int j) {
    return (row(i)[j/64] >> (j%64)) & 1;
  } // 28bd

  // Set value in i-th row and j-th column
  void set(int i, int j, bool val) {
    ull &w = row(i)[j/64], m = 1ull << (j%64);
    if (val) w |= m;
    else w &= ~m;
  } // 98a8
}; // 4df7
```

math/crt.h 8a85

```
using Pll = pair<ll, ll>;

ll egcd(ll a, ll b, ll& x, ll& y) {
  if (!a) return x=0, y=1, b;
  ll d = egcd(b%a, a, y, x);
  x -= b/a*y;
  return d;
} // 23c8

// Chinese Remainder Theorem; time: O(lg lcm)
// Solves x = a.x (mod a.y), x = b.x (mod b.y)
// Returns pair (x mod lcm, lcm(a.y, b.y))
// or (-1, -1) if there's no solution.
// WARNING: a.x and b.x are assumed to be
// in [0;a.y) and [0;b.y) respectively.
// Works properly if lcm(a.y, b.y) < 2^63.
Pll crt(Pll a, Pll b) {
  if (a.y < b.y) swap(a, b);
  ll x, y, g = egcd(a.y, b.y, x, y);
  ll c = b.x-a.x, d = b.y/g, p = a.y*d;
  if (c % g) return {-1, -1};
  ll s = (a.x + c/g*x % d * a.y) % p;
  return {s < 0 ? s+p : s, p};
} // 35a8
```

math/fft_complex.h 945b

```
using dbl = double;
using cmpl = complex<dbl>;

// Default std::complex multiplication is slow.
// You can use this to achieve small speedup.
cmpl operator*(cmpl a, cmpl b) {
  dbl ax = real(a), ay = imag(a);
  dbl bx = real(b), by = imag(b);
  return {ax*bx-ay*by, ax*by+ay*bx};
} // 3b78

cmpl operator*=(cmpl& a,cmpl b) {return a=a*b;}

// Compute DFT over complex numbers; O(n lg n)
// Input size must be power of 2!
void fft(vector<cmpl>& a) {
  static vector<cmpl> w(2, 1);
  int n = sz(a);

  for (int k = sz(w); k < n; k *= 2) {
    w.resize(n);
    rep(i,0,k) w[k+i] = exp(cmpl(0, M_PI*i/k));
  } // 92a9

  Vi rev(n);
  rep(i,0,n) rev[i] = (rev[i/2] | i%2*n) / 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 += k*2) rep(j,0,k) {
      auto d = a[i+j+k] * w[j+k];
      a[i+j+k] = a[i+j] - d;
      a[i+j] += d;
    } // b389
  } // 84bf
} // adf8

// Convolve complex-valued a and b,
// store result in a; time: O(n lg n), 3x FFT
void convolve(vector<cmpl>& a, vector<cmpl> b){
  int len = sz(a) + sz(b) - 1;
  if (len <= 0) return a.clear();
  int n = 1 << (32 - __builtin_clz(len));
  a.resize(n); b.resize(n);
  fft(a); fft(b);
  rep(i, 0, n) a[i] *= b[i] / dbl(n);
  reverse(a.begin()+1, a.end());
  fft(a);
  a.resize(len);
} // 8fe9

// Convolve real-valued a and b, returns result
// time: O(n lg n), 2x FFT
// Rounding to integers is safe as long as
// (max_coeff^2)*n*log_2(n) < 9*10^14
// (in practice 10^16 or higher).
vector<dbl> convolve(vector<dbl>& a,
                     vector<dbl>& b) {
```

```
  int len = max(sz(a) + sz(b) - 1, 0);
  int n = 1 << (32 - __builtin_clz(len));

  vector<cmpl> in(n), out(n);
  rep(i, 0, sz(a)) in[i].real(a[i]);
  rep(i, 0, sz(b)) in[i].imag(b[i]);

  fft(in);
  each(x, in) x *= x;
  rep(i,0,n) out[i] = in[-i&(n-1)]-conj(in[i]);
  fft(out);

  vector<dbl> ret(len);
  rep(i, 0, len) ret[i] = imag(out[i]) / (n*4);
  return ret;
} // 19ed

constexpr ll MOD = 1e9+7;

// High precision convolution of integer-valued
// a and b mod MOD; time: O(n lg n), 4x FFT
// Input is expected to be in range [0;MOD)!
// Rounding is safe if MOD*n*log_2(n) < 9*10^14
// (in practice 10^16 or higher).
vector<ll> convMod(vector<ll>& a,
                   vector<ll>& b) {
  vector<ll> ret(sz(a) + sz(b) - 1);
  int n = 1 << (32 - __builtin_clz(sz(ret)));
  ll cut = ll(sqrt(MOD))+1;

  vector<cmpl> c(n), d(n), g(n), f(n);

  rep(i, 0, sz(a))
    c[i] = {dbl(a[i]/cut), dbl(a[i]%cut)};
  rep(i, 0, sz(b))
    d[i] = {dbl(b[i]/cut), dbl(b[i]%cut)};

  fft(c); fft(d);

  rep(i, 0, n) {
    int j = -i & (n-1);
    f[j] = (c[i]+conj(c[j])) * d[i] / (n*2.0);
    g[j] =
      (c[i]-conj(c[j])) * d[i] / cmpl(0, n*2);
  } // e877

  fft(f); fft(g);

  rep(i, 0, sz(ret)) {
    ll t = llround(real(f[i])) % MOD * cut;
    t += llround(imag(f[i]));
    t = (t + llround(real(g[i]))) % MOD * cut;
    t = (t + llround(imag(g[i]))) % MOD;
    ret[i] = (t < 0 ? t+MOD : t);
  } // e75d

  return ret;
} // 513f
```

math/fft_mod.h 0a8a

```
// Number Theoretic Tranform (NTT)
// For functions below you can choose 2 params:
// 1. M - prime modulus that MUST BE of form
//       a*2^k+1, computation is done in Z_M
// 2. R - generator of Z_M

// Modulus often seen on Codeforces:
// M = (119<<23)+1, R = 62; M is 998244353

// Parameters for ll computation with CRT:
// M = (479<<21)+1, R = 62; M is > 10^9
// M = (483<<21)+1, R = 62; M is > 10^9

ll modPow(ll a, ll e, ll m) {
  ll t = 1 % m;
  while (e) {
    if (e % 2) t = t*a % m;
    e /= 2; a = a*a % m;
  } // 66ca
  return t;
} // 1973

// Compute DFT over Z_M with generator R.
// Input size must be power of 2; O(n lg n)
// Input is expected to be in range [0;MOD)!
// dit == true <=> inverse transform * 2^n
//                 (without normalization)
template<ll M, ll R, bool dit>
void ntt(vector<ll>& a) {
  static vector<ll> w(2, 1);
  int n = sz(a);

  for (int k = sz(w); k < n; k *= 2) {
    w.resize(n, 1);
    ll c = modPow(R, M/2/k, M);
    if (dit) c = modPow(c, M-2, M);
    rep(i, k+1, k*2) w[i] = w[i-1]*c % M;
  } // 0d98

  for (int t = 1; t < n; t *= 2) {
    int k = (dit ? t : n/t/2);
    for (int i=0; i < n; i += k*2) rep(j,0,k) {
      ll &c = a[i+j], &d = a[i+j+k];
      ll e = w[j+k], f = d;
      d = (dit ? c - (f=f*e%M) : (c-f)*e % M);
      if (d < 0) d += M;
      if ((c += f) >= M) c -= M;
    } // e4a6
  } // 8d38
} // 01f5

// Convolve a and b mod M (R is generator),
// store result in a; time: O(n lg n), 3x NTT
// Input is expected to be in range [0;MOD)!
template<ll M = (119<<23)+1, ll R = 62>
void convolve(vector<ll>& a, vector<ll> b) {
  int len = sz(a) + sz(b) - 1;
  if (len <= 0) return a.clear();
  int n = 1 << (32 - __builtin_clz(len));
  ll t = modPow(n, M-2, M);
  a.resize(n); b.resize(n);
  ntt<M,R,0>(a); ntt<M,R,0>(b);
  rep(i, 0, n) a[i] = a[i]*b[i] % M * t % M;
  ntt<M,R,1>(a);
  a.resize(len);
} // b413

ll egcd(ll a, ll b, ll& x, ll& y) {
  if (!a) return x=0, y=1, b;
  ll d = egcd(b%a, a, y, x);
  x -= b/a*y;
  return d;
} // 23c8

// Convolve a and b with 64-bit output,
// store result in a; time: O(n lg n), 6x NTT
// Input is expected to be non-negative!
void convLong(vector<ll>& a, vector<ll> b) {
  const ll M1 = (479<<21)+1, M2 = (483<<21)+1;
  const ll MX = M1*M2, R = 62;

  auto c = a, d = b;
  each(k, a) k %= M1;
  each(k, b) k %= M1;
  each(k, c) k %= M2;
  each(k, d) k %= M2;

  convolve<M1, R>(a, b);
  convolve<M2, R>(c, d);

  ll x, y; egcd(M1, M2, x, y);

  rep(i, 0, sz(a)) {
    a[i] += (c[i]-a[i])*x % M2 * M1;
    if ((a[i] %= MX) < 0) a[i] += MX;
  } // 2279
} // c493
```

math/fwht.h 3e6f

```
// Fast Walsh-Hadamard Transform; O(n lg n)
// Input must be power of 2!
// Uncommented version is for XOR.
// OR version is equivalent to sum-over-subsets
// (Zeta transform, inverse is Moebius).
// AND version is same as sum-over-supersets.
template<bool inv, class T>
void fwht(vector<T>& b) {
  for (int s = 1; s < sz(b); s *= 2) {
    for (int i = 0; i < sz(b); i += s*2) {
      rep(j, i, i+s) {
        auto &x = b[j], &y = b[j+s];
        tie(x, y) =
          mp(x+y, x-y);                   //XOR
        // inv ? mp(x-y, y) : mp(x+y, y); //AND
        // inv ? mp(x, y-x) : mp(x, x+y); //OR
      } // eea9
    } // a3d5
  } // 95ed

  // ONLY FOR XOR:
  if (inv) each(e, b) e /= sz(b);
} // 0779

// Compute convolution of a and b such that
// ans[i#j] += a[i]*b[j], where # is OR, AND
// or XOR, depending on FWHT version.
// Stores result in a; time: O(n lg n)
// Both arrays must be of same size = 2^n!
template<class T>
void bitConv(vector<T>& a, vector<T> b) {
  fwht<0>(a);
  fwht<0>(b);
  rep(i, 0, sz(a)) a[i] *= b[i];
  fwht<1>(a);
} // 7b82
```

math/gauss.h 7f0b

```
// Solve system of linear equations; O(n^2*m)
// - A - extended matrix, rows are equations,
//       columns are variables,
//       m-th column is equation result
//       (A[i][j] - i-th row and j-th column)
// - ans - output for variables values
// - m - variable count
// Returns 0 if no solutions found, 1 if one,
// 2 if more than 1 solution exist.
int gauss(vector<vector<double>>& A,
          vector<double>& ans, int m) {
  Vi col;
  ans.assign(m, 0);

  rep(i, 0, sz(A)) {
    int c = 0;
    while (c <= m && !cmp(A[i][c], 0)) c++;
    // For Zp:
    //while (c <= m && !A[i][c].x) c++;

    if (c >= m) {
      if (c == m) return 0;
      continue;
    } // a6bb

    rep(k, i+1, sz(A)) {
      auto mult = A[k][c] / A[i][c];
      rep(j, 0, m+1) A[k][j] -= A[i][j]*mult;
    } // 8dd5

    swap(A[i], A[sz(col)]);
    col.pb(c);
  } // ea2c

  for (int i = sz(col); i--;) {
    ans[col[i]] = A[i][m] / A[i][col[i]];
    rep(k, 0, i)
      A[k][m] -= ans[col[i]] * A[k][col[i]];
  } // 31b9

  return sz(col) < m ? 2 : 1;
} // 0b76
```

math/gauss_ortho.h c6ae

```
using Row = vector<double>;
using Matrix = vector<Row>;

// Given a system of n linear equations A
// over m variables, find dimensionality D
// of solution subspace, matrix M and vector t
// such that:
// - matrix M is orthogonal (i.e. M*M^T = I)
// - x is a solution <=> (Mx+t)[D..] = 0
// - x[D..] = 0 <=> M^T(x-t) is a solution
//   (in particular -M^T*t is a solution)
// Returns number of dimensions D, or -1 if
// there is no solution; time: O(n^2*m + n*m^2)
// Warning: numerical stability is kinda sus
int orthoGauss(Matrix& A, Matrix& M,
               Row& t, int m) {
  int d = m;
  t.assign(m, 0);
  M.assign(m, Row(m));
  rep(i, 0, m) M[i][i] = 1;

  rep(i, 0, sz(A)) {
    auto& w = A[i];
    double s = 0;
    rep(j, 0, d) s += w[j]*w[j];
    if (!cmp(s, 0)) {
      if (cmp(w[m], 0)) return -1;
      continue;
    } // 7462

    double r = sqrt(s);
    if (w[d-1] < 0) r = -r;
    s = sqrt((s + w[d-1]*r)*2);
    w[d-1] += r;
    rep(j, 0, d) w[j] /= s;
    r = w[m] / (w[d-1] * r * 2);

    rep(j, i+1, sz(A)) {
      s = 0;
      rep(k, 0, d) s += A[j][k] * w[k];
      s *= 2;
      rep(k, 0, d) A[j][k] -= s * w[k];
      A[j][m] -= s*r;
    } // 69fe

    rep(j, 0, m) {
      s = 0;
      rep(k, 0, d) s += M[k][j] * w[k];
      s *= 2;
      rep(k, 0, d) M[k][j] -= s * w[k];
    } // 692b
```

```
    s = -r;
    rep(k, 0, d) s += t[k] * w[k];
    s *= 2;
    rep(k, 0, d) t[k] -= s * w[k];
    d--;
  } // a688

  return d;
} // c093
```

math/linear_rec.h 60be

```
constexpr ll MOD = 998244353;

using Poly = vector<ll>;

// Compute k-th term of an n-order linear
// recurrence C[i] = sum C[i-j-1]*D[j],
// given C[0..n-1] and D[0..n-1]; O(n^2 log k)
ll linearRec(const Poly& C,
             const Poly& D, ll k) {
  int n = sz(D);

  auto mul = [&](Poly a, Poly b) {
    Poly ret(n*2+1);
    rep(i, 0, n+1) rep(j, 0, n+1)
      ret[i+j] = (ret[i+j] + a[i]*b[j]) % MOD;
    for (int i = n*2; i > n; i--) rep(j, 0, n)
      ret[i-j-1] =
        (ret[i-j-1] + ret[i]*D[j]) % MOD;
    ret.resize(n+1);
    return ret;
  }; // e722

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

  for (k++; k; k /= 2) {
    if (k % 2) pol = mul(pol, e);
    e = mul(e, e);
  } // 13af

  ll ret = 0;
  rep(i,0,n) ret = (ret + pol[i+1]*C[i]) % MOD;
  return ret;
} // 3fd1
```

math/linear_rec_fast.h 58a8

```
#include "polynomial.h"

// Compute k-th term of an n-order linear
// recurrence C[i] = sum C[i-j-1]*D[j],
// given C[0..n-1] and D[0..n-1];
// time: O(n log n log k)
Zp linearRec(const Poly& C,
             const Poly& D, ll k) {
  Poly f(sz(D)+1, 1);
  rep(i, 0, sz(D)) f[i] = -D[sz(D)-i-1];
  f = pow({0, 1}, k, f);
  Zp ret = 0;
  rep(i, 0, sz(f)) ret += f[i]*C[i];
  return ret;
} // 5b8d
```

math/matrix.h 9bf7

```
#include "modular.h"

using Row = vector<Zp>;
using Matrix = vector<Row>;

// Create n x n identity matrix
Matrix ident(int n) {
  Matrix ret(n, Row(n));
  rep(i, 0, n) ret[i][i] = 1;
  return ret;
} // ad1d

// Add matrices
Matrix& operator+=(Matrix& l, const Matrix& r){
  rep(i, 0, sz(l)) rep(k, 0, sz(l[0]))
    l[i][k] += r[i][k];
  return l;
} // b6bf
Matrix operator+(Matrix l, const Matrix& r) {
  return l += r;
} // d9b3

// Subtract matrices
Matrix& operator-=(Matrix& l, const Matrix& r){
  rep(i, 0, sz(l)) rep(k, 0, sz(l[0]))
    l[i][k] -= r[i][k];
  return l;
} // 90a1
Matrix operator-(Matrix l, const Matrix& r) {
  return l -= r;
} // dc4f

// Multiply matrices
Matrix operator*(const Matrix& l,
                 const Matrix& r) {
  Matrix ret(sz(l), Row(sz(r[0])));
  rep(i, 0, sz(l)) rep(j, 0, sz(r[0]))
    rep(k, 0, sz(r))
      ret[i][j] += l[i][k] * r[k][j];
  return ret;
} // 52ca
Matrix& operator*=(Matrix& l, const Matrix& r){
  return l = l*r;
} // da8a

// Square matrix power; time: O(n^3 * lg e)
Matrix pow(Matrix a, ll e) {
  Matrix t = ident(sz(a));
  while (e) {
    if (e % 2) t *= a;
    e /= 2; a *= a;
  } // 4400
  return t;
} // 65ea

// Transpose matrix
Matrix transpose(const Matrix& m) {
  Matrix ret(sz(m[0]), Row(sz(m)));
  rep(i, 0, sz(m)) rep(j, 0, sz(m[0]))
    ret[j][i] = m[i][j];
  return ret;
} // 5650

// Transform matrix to echelon form
// and compute its determinant sign and rank.
int echelon(Matrix& A, int& sign) { // O(n^3)
  int rank = 0;
  sign = 1;
  rep(c, 0, sz(A[0])) {
    if (rank >= sz(A)) break;
    rep(i, rank+1, sz(A)) if (A[i][c].x) {
      swap(A[i], A[rank]);
      sign *= -1;
      break;
    } // f98a
    if (A[rank][c].x) {
      rep(i, rank+1, sz(A)) {
        auto mult = A[i][c] / A[rank][c];
        rep(j, 0, sz(A[0]))
          A[i][j] -= A[rank][j]*mult;
      } // f519
      rank++;
    } // 4cd8
  } // 36e9
  return rank;
} // 6882

// Compute matrix rank; time: O(n^3)
#define rank rank_
int rank(Matrix A) {
  int s; return echelon(A, s);
} // c599

// Compute square matrix determinant; O(n^3)
Zp det(Matrix A) {
  int s; echelon(A, s);
  Zp ret = s;
  rep(i, 0, sz(A)) ret *= A[i][i];
  return ret;
} // b252

// Invert square matrix if possible; O(n^3)
// Returns true if matrix is invertible.
bool invert(Matrix& A) {
  int s, n = sz(A);
  rep(i, 0, n) A[i].resize(n*2), A[i][n+i] = 1;
  echelon(A, s);
  for (int i = n; i--;) {
    if (!A[i][i].x) return 0;
    auto mult = A[i][i].inv();
    each(k, A[i]) k *= mult;
    rep(k, 0, i) rep(j, 0, n)
      A[k][n+j] -= A[i][n+j]*A[k][i];
  } // 1e97
  each(r, A) r.erase(r.begin(), r.begin()+n);
  return 1;
} // 65b9
```

math/miller_rabin.h 7005

```
#include "modular64.h"

// Miller-Rabin primality test
// time O(k*lg^2 n), where k = number of bases

// Deterministic for p <= 10^9
// constexpr ll BASES[] = {
//   336781006125, 9639812373923155
// }; // d41d

// Deterministic for p <= 2^64
constexpr ll BASES[] = {
  2,325,9375,28178,450775,9780504,1795265022
}; // b8e0

bool isPrime(ll p) {
  if (p <= 2) return p == 2;
  if (p%2 == 0) return 0;

  ll d = p-1, t = 0;
  while (d%2 == 0) d /= 2, t++;

  each(a, BASES) if (a%p) {
    // ll a = rand() % (p-1) + 1;
    ll b = modPow(a%p, d, p);
    if (b == 1 || b == p-1) continue;
    rep(i, 1, t)
      if ((b = modMul(b, b, p)) == p-1) break;
    if (b != p-1) return 0;
  } // 9342

  return 1;
} // bec2
```

math/modinv_precompute.h 2882

```
constexpr ll MOD = 234567899;
vector<ll> modInv(MOD); // You can lower size

// Precompute modular inverses; time: O(MOD)
void initModInv() {
  modInv[1] = 1;
  rep(i, 2, sz(modInv)) modInv[i] =
    (MOD - (MOD/i) * modInv[MOD%i]) % MOD;
} // 22c1
```

math/modular.h 72a7

```
// Modulus often seen on Codeforces:
constexpr int MOD = 998244353;
// Some big prime: 15*(1<<27)+1 ~ 2*10^9

ll modInv(ll a, ll m) { // a^(-1) mod m
  if (a == 1) return 1;
  return ((a - modInv(m%a, a))*m + 1) / a;
} // c437

ll modPow(ll a, ll e, ll m) { // a^e mod m
  ll t = 1 % m;
  while (e) {
    if (e % 2) t = t*a % m;
    e /= 2; a = a*a % m;
  } // 66ca
  return t;
} // 1973

// Wrapper for modular arithmetic
struct Zp {
  ll x; // Contained value, in range [0;MOD-1]
  Zp() : x(0) {}
  Zp(ll a) : x(a%MOD) { if (x < 0) x += MOD; }

  #define OP(c,d) Zp& operator c##=(Zp r) { \
      x = x d; return *this; } \
    Zp operator c(Zp r) const { \
      Zp t = *this; return t c##= r; } // e986

  OP(+, +r.x - MOD*(x+r.x >= MOD));
  OP(-, -r.x + MOD*(0 > x-r.x));
  OP(*, *r.x % MOD);
  OP(/, *r.inv().x % MOD);
  Zp operator-() const { return Zp()-*this; }

  // For composite modulus use modInv, not pow
  Zp inv() const { return pow(MOD-2); }
  Zp pow(ll e) const{ return modPow(x,e,MOD); }
  void print() { cerr << x; } // For deb()
}; // f730

// Extended Euclidean Algorithm
ll egcd(ll a, ll b, ll& x, ll& y) {
  if (!a) return x=0, y=1, b;
  ll d = egcd(b%a, a, y, x);
  x -= b/a*y;
  return d;
} // 23c8
```

math/modular64.h 4b73

```
// Modular arithmetic for modulus < 2^62

ll modAdd(ll x, ll y, ll m) {
  x += y;
  return x < m ? x : x-m;
} // b653

ll modSub(ll x, ll y, ll m) {
  x -= y;
  return x >= 0 ? x : x+m;
} // b073
```

```
// About 4x slower than normal modulo
ll modMul(ll a, ll b, ll m) {
  ll c = ll((long double)a * b / m);
  ll r = (a*b - c*m) % m;
  return r < 0 ? r+m : r;
} // 1815

ll modPow(ll x, ll e, ll m) {
  ll t = 1;
  while (e) {
    if (e & 1) t = modMul(t, x, m);
    e >>= 1;
    x = modMul(x, x, m);
  } // bd61
  return t;
} // c8ba
```

math/modular_generator.h 845b

```
#include "modular.h" // modPow

// Get unique prime factors of n; O(sqrt n)
vector<ll> factorize(ll n) {
  vector<ll> fac;
  for (ll i = 2; i*i <= n; i++) {
    if (n%i == 0) {
      while (n%i == 0) n /= i;
      fac.pb(i);
    } // 6069
  } // a0cc
  if (n > 1) fac.pb(n);
  return fac;
} // 4a2a

// Find smallest primitive root mod n;
// time: O(sqrt(n) + g*log^2 n)
// Returns -1 if generator doesn't exist.
// For n <= 10^7 smallest generator is <= 115.
// You can use faster factorization algorithm
// to get rid of sqrt(n).
ll generator(ll n) {
  if (n <= 1 || (n > 4 && n%4 == 0)) return -1;

  vector<ll> fac = factorize(n);
  if (sz(fac) > (fac[0] == 2)+1) return -1;

  ll phi = n;
  each(p, fac) phi = phi / p * (p-1);
  fac = factorize(phi);

  for (ll g = 1;; g++) if (__gcd(g, n) == 1) {
    each(f, fac) if (modPow(g, phi/f, n) == 1)
      goto nxt;
    return g;
    nxt:;
  } // db24
} // 7641
```

math/modular_log.h ac62

```
#include "modular.h" // modInv

// Baby-step giant-step algorithm; O(sqrt(p))
// Finds smallest x such that a^x = b (mod p)
// or returns -1 if there's no solution.
ll dlog(ll a, ll b, ll p) {
  int m = int(min(ll(sqrt(p))+2, p-1));
  unordered_map<ll, int> small;
  ll t = 1;

  rep(i, 0, m) {
    int& k = small[t];
    if (!k) k = i+1;
    t = t*a % p;
  } // f1d0

  t = modInv(t, p);

  rep(i, 0, m) {
    int j = small[b];
    if (j) return i*ll(m) + j - 1;
    b = b*t % p;
  } // c7ed

  return -1;
} // 5c26
```

math/modular_sqrt.h db16

```
#include "modular.h" // modPow

// Tonelli-Shanks algorithm for modular sqrt
// modulo prime; O(lg^2 p), O(lg p) for most p
// Returns -1 if root doesn't exists or else
// returns square root x (the other one is -x).
ll modSqrt(ll a, ll p) {
  a %= p;
  if (a < 0) a += p;
  if (a <= 1) return a;
  if (modPow(a, p/2, p) != 1) return -1;
  if (p%4 == 3) return modPow(a, p/4+1, p);

  ll s = p-1, n = 2;
  int r = 0, j;
  while (s%2 == 0) s /= 2, r++;
  while (modPow(n, p/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 = j) {
    ll t = b;
    for (j = 0; j < r && t != 1; j++)
      t = t*t % p;
    if (!j) return x;
    ll gs = modPow(g, 1LL << (r-j-1), p);
    g = gs*gs % p;
    x = x*gs % p;
    b = b*g % p;
  } // f83f
} // 7a97
```

math/montgomery.h a4ba

```
#include "modular.h" // modInv

// Montgomery modular multiplication
// MOD < MG_MULT, gcd(MG_MULT, MOD) must be 1
// Don't use if modulo is constexpr; UNTESTED

constexpr ll MG_SHIFT = 32;
constexpr ll MG_MULT  = 1LL << MG_SHIFT;
constexpr ll MG_MASK  = MG_MULT - 1;
const ll MG_INV = MG_MULT-modInv(MOD, MG_MULT);

// Convert to Montgomery form
ll MG(ll x) { return (x*MG_MULT) % MOD; }

// Montgomery reduction
// redc(mg * mg) = Montgomery-form product
ll redc(ll x) {
  ll q = (x * MG_INV) & MG_MASK;
  x = (x + q*MOD) >> MG_SHIFT;
  return (x >= MOD ? x-MOD : x);
} // d0f5
```

math/nimber.h 474f

```
// Nimbers are defined as sizes of Nim heaps.
// Operations on nimbers are defined as:
// a+b = mex({a'+b : a' < a} u {a+b' : b' < b})
// ab  = mex({a'b+ab'+a'b' : a' < a, b' < b})
// Nimbers smaller than M = 2^2^k form a field.
// Addition is equivalent to xor, meanwhile
// multiplication can be evaluated
// in O(lg^2 M) after precomputing.

using ull = uint64_t;
ull nbuf[64][64]; // Nim-products for 2^i * 2^j

// Multiply nimbers; time: O(lg^2 M)
// WARNING: Call initNimMul() before using.
ull nimMul(ull a, ull b) {
  ull ret = 0;
  for (ull s = a; s; s &= (s-1))
    for (ull t = b; t; t &= (t-1))
      ret ^= nbuf[__builtin_ctzll(s)]
                 [__builtin_ctzll(t)];
  return ret;
} // 25be

// Initialize nim-products lookup table
void initNimMul() {
  rep(i, 0, 64)
    nbuf[i][0] = nbuf[0][i] = 1ull << i;
  rep(b, 1, 64) rep(a, 1, b+1) {
    int i = 1 << (63 - __builtin_clzll(a));
    int j = 1 << (63 - __builtin_clzll(b));
    ull t = nbuf[a-i][b-j];
    if (i < j)
      t = nimMul(t, 1ull << i) << j;
    else
      t = nimMul(t, 1ull << (i-1)) ^ (t << i);
    nbuf[a][b] = nbuf[b][a] = t;
  } // ca24
} // 1811

// Compute a^e under nim arithmetic; O(lg^3 M)
// WARNING: Call initNimMul() before using.
ull nimPow(ull a, ull e) {
  ull t = 1;
  while (e) {
    if (e % 2) t = nimMul(t, a);
    e /= 2; a = nimMul(a, a);
  } // da53
  return t;
} // c06c

// Compute inverse of a in 2^64 nim-field;
// time: O(lg^3 M)
// WARNING: Call initNimMul() before using.
ull nimInv(ull a) {
  return nimPow(a, ull(-2));
} // c6d9

// If you need to multiply many nimbers by
// the same value you can use this to speedup.
// WARNING: Call initNimMul() before using.
struct NimMult {
  ull M[64] = {0};

  // Initialize lookup; time: O(lg^2 M)
  NimMult(ull a) {
    for (ull t=a; t; t &= (t-1)) rep(i, 0, 64)
      M[i] ^= nbuf[__builtin_ctzll(t)][i];
  } // ea88

  // Multiply by b; time: O(lg M)
  ull operator()(ull b) {
    ull ret = 0;
    for (ull t = b; t; t &= (t-1))
      ret ^= M[__builtin_ctzll(t)];
    return ret;
  } // e480
}; // 1b80
```

math/phi_large.h 8703

```
#include "pollard_rho.h"

// Compute Euler's totient of large numbers
// time: O(n^(1/4)) <- factorization
ll phi(ll n) {
  each(p, factorize(n)) n = n / p.x * (p.x-1);
  return n;
} // 798e
```

math/phi_precompute.h 728b

```
Vi phi(1e7+1);

// Precompute Euler's totients; time: O(n lg n)
void calcPhi() {
  iota(all(phi), 0);
  rep(i, 2, sz(phi)) if (phi[i] == i)
    for (int j = i; j < sz(phi); j += i)
      phi[j] = phi[j] / i * (i-1);
} // 3c65
```

math/phi_prefix_sum.h 39a5

```
#include "phi_precompute.h"

constexpr int MOD = 998244353;

vector<ll> phiSum; // [k] = sum from 0 to k-1

// Precompute Euler's totient prefix sums
// for small values; time: O(n lg n)
void calcPhiSum() {
  calcPhi();
  phiSum.resize(sz(phi)+1);
  rep(i, 0, sz(phi))
    phiSum[i+1] = (phiSum[i] + phi[i]) % MOD;
} // bcf5

// Get prefix sum of phi(0) + ... + phi(n-1).
// WARNING: Call calcPhiSum first!
// For MOD > 4*10^9, answer will overflow.
ll getPhiSum(ll n) { // time: O(n^(2/3))
  static unordered_map<ll, ll> big;
  if (n < sz(phiSum)) return phiSum[n];
  if (big.count(--n)) return big[n];

  ll ret = (n%2 ? n%MOD * ((n+1)/2 % MOD)
                : n/2%MOD * (n%MOD+1)) % MOD;

  for (ll s, i = 2; i <= n; i = s+1) {
    s = n / (n/i);
    ret -= (s-i+1)%MOD*getPhiSum(n/i+1) % MOD;
  } // fa0b

  return big[n] = ret = (ret%MOD + MOD) % MOD;
} // 1d5f
```

math/pi_large.h fcbd

```
constexpr int MAX_P = 1e7;
vector<ll> pis, prl;

// Precompute prime counting function
// for small values; time: O(n lg lg n)
void initPi() {
  pis.assign(MAX_P+1, 1);
  pis[0] = pis[1] = 0;

  for (int i = 2; i*i <= MAX_P; i++)
    if (pis[i])
      for (int j = i*i; j <= MAX_P; j += i)
        pis[j] = 0;
```

```
  rep(i, 1, sz(pis)) {
    if (pis[i]) prl.pb(i);
    pis[i] += pis[i-1];
  } // 0672
} // 6d92

ll partial(ll x, ll a) {
  static vector<unordered_map<ll, ll>> big;
  big.resize(sz(prl));
  if (!a) return (x+1) / 2;
  if (big[a].count(x)) return big[a][x];
  ll ret = partial(x, a-1)
    - partial(x / prl[a], a-1);
  return big[a][x] = ret;
} // 774f

// Count number of primes <= x;
// time: O(n^(2/3) * log(n)^(1/3))
// Set MAX_P to be > sqrt(x) and call initPi
// before using!
ll pi(ll x) {
  static unordered_map<ll, ll> big;
  if (x < sz(pis)) return pis[x];
  if (big.count(x)) return big[x];

  ll a = 0;
  while (prl[a]*prl[a]*prl[a]*prl[a] < x) a++;

  ll ret = 0, b = --a;

  while (++b < sz(prl) && prl[b]*prl[b] < x) {
    ll w = x / prl[b];
    ret -= pi(w);
    for (ll j = b; prl[j]*prl[j] <= w; j++)
      ret -= pi(w / prl[j]) - j;
  } // a584

  ret += partial(x, a) + (b+a-1)*(b-a)/2;
  return big[x] = ret;
} // ea1d
```

math/pi_large_precomp.h 7fc0

```
#include "sieve.h"

// Count primes in given interval
// using precomputed table.
// Set MAX_P to sqrt(MAX_N) and run sieve()!
// Precomputed table will contain N_BUCKETS
// elements - check source size limit.

constexpr ll MAX_N = 1e11+1;
constexpr ll N_BUCKETS = 10000;
constexpr ll BUCKET_SIZE = (MAX_N/N_BUCKETS)+1;
constexpr ll precomputed[] = {/* ... */};

ll sieveRange(ll from, ll to) {
  bitset<BUCKET_SIZE> elems;
  from = max(from, 2LL);
  to = max(from, to);
  each(p, primesList) {
    ll c = max((from+p-1) / p, 2LL);
    for (ll i = c*p; i < to; i += p)
      elems.set(i-from);
  } // a29f
  return to-from-elems.count();
} // c646

// Run once on local computer to precompute
// table. Takes about 10 minutes for n = 1e11.
// Sanity check (for default params):
// 664579, 606028, 587253, 575795, ...
void localPrecompute() {
  for (ll i = 0; i < MAX_N; i += BUCKET_SIZE) {
    ll to = min(i+BUCKET_SIZE, MAX_N);
    cout << sieveRange(i, to) << ',' << flush;
  } // f6a7
  cout << endl;
} // 2b1e

// Count primes in [from;to) using table.
// O(N_BUCKETS + BUCKET_SIZE*lg lg n + sqrt(n))
ll countPrimes(ll from, ll to) {
  ll bFrom = from/BUCKET_SIZE+1,
     bTo = to/BUCKET_SIZE;
  if (bFrom > bTo) return sieveRange(from, to);
  ll ret = accumulate(precomputed+bFrom,
                      precomputed+bTo, 0);
  ret += sieveRange(from, bFrom*BUCKET_SIZE);
  ret += sieveRange(bTo*BUCKET_SIZE, to);
  return ret;
} // cced
```

math/pollard_rho.h ef01

```
#include "modular64.h"
#include "miller_rabin.h"

using Factor = pair<ll, int>;

void rho(vector<ll>& out, ll n) {
  if (n <= 1) return;
  if (isPrime(n)) out.pb(n);
  else if (n%2 == 0) rho(out,2), rho(out,n/2);
  else for (ll a = 2;; a++) {
    ll x = 2, y = 2, d = 1;
    while (d == 1) {
      x = modAdd(modMul(x, x, n), a, n);
      y = modAdd(modMul(y, y, n), a, n);
      y = modAdd(modMul(y, y, n), a, n);
      d = __gcd(abs(x-y), n);
    } // 3378
    if (d != n) return rho(out,d),rho(out,n/d);
  } // 047e
} // ba89

// Pollard's rho factorization algorithm
// Las Vegas version; time: n^(1/4)
// Returns pairs (prime, power), sorted
vector<Factor> factorize(ll n) {
  vector<Factor> ret;
  vector<ll> raw;
  rho(raw, n);
  sort(all(raw));
  each(f, raw) {
    if (ret.empty() || ret.back().x != f)
      ret.pb({ f, 1 });
    else
      ret.back().y++;
  } // 2ab1
  return ret;
} // 471c
```

math/polynomial.h a449

```
#include "modular.h"
#include "fft_mod.h"

using Poly = vector<Zp>;

// Cut off trailing zeroes; time: O(n)
void norm(Poly& P) {
  while (!P.empty() && !P.back().x)
    P.pop_back();
} // 8a8a

// Evaluate polynomial at x; time: O(n)
Zp eval(const Poly& P, Zp x) {
  Zp n = 0, y = 1;
  each(a, P) n += a*y, y *= x;
  return n;
} // d865

// Add polynomial; time: O(n)
Poly& operator+=(Poly& l, const Poly& r) {
  l.resize(max(sz(l), sz(r)));
  rep(i, 0, sz(r)) l[i] += r[i];
  norm(l);
  return l;
} // 656e
Poly operator+(Poly l, const Poly& r) {
  return l += r;
} // d9b3

// Subtract polynomial; time: O(n)
Poly& operator-=(Poly& l, const Poly& r) {
  l.resize(max(sz(l), sz(r)));
  rep(i, 0, sz(r)) l[i] -= r[i];
  norm(l);
  return l;
} // c68b
Poly operator-(Poly l, const Poly& r) {
  return l -= r;
} // dc4f

// Multiply by polynomial; time: O(n lg n)
Poly& operator*=(Poly& l, const Poly& r) {
  if (min(sz(l), sz(r)) < 50) {
    // Naive multiplication
    Poly p(sz(l)+sz(r));
    rep(i, 0, sz(l)) rep(j, 0, sz(r))
      p[i+j] += l[i]*r[j];
    l.swap(p);
  } else {
    // FFT multiplication
    // Choose appropriate convolution method,
    // see fft_mod.h and fft_complex.h
    using v = vector<ll>;
    convolve<MOD, 62>(*(v*)&l, *(const v*)&r);
  } // 30c9
  norm(l);
  return l;
} // e8b3
Poly operator*(Poly l, const Poly& r) {
  return l *= r;
} // 2de3

// Compute inverse series mod x^n; O(n lg n)
// Requires P(0) != 0.
Poly invert(const Poly& P, int n) {
  assert(!P.empty() && P[0].x);
  Poly tmp{P[0]}, ret = {P[0].inv()};
  for (int i = 1; i < n; i *= 2) {
    rep(j, i, min(i*2, sz(P))) tmp.pb(P[j]);
    (ret *= Poly{2} - tmp*ret).resize(i*2);
  } // 904e
  ret.resize(n);
  return ret;
} // 9293

// Floor division by polynomial; O(n lg n)
Poly& operator/=(Poly& l, Poly r) {
  norm(l); norm(r);
  int d = sz(l)-sz(r)+1;
  if (d <= 0) return l.clear(), l;
  reverse(all(l));
  reverse(all(r));
  l.resize(d);
  l *= invert(r, d);
  l.resize(d);
  reverse(all(l));
  return l;
} // cf5e
Poly operator/(Poly l, const Poly& r) {
  return l /= r;
} // 152d

// Remainder modulo a polynomial; O(n lg n)
Poly operator%(const Poly& l, const Poly& r) {
  return l - r*(l/r);
} // 4fc8
Poly& operator%=(Poly& l, const Poly& r) {
  return l -= r*(l/r);
} // 80bb

// Compute a^e mod x^n, where a is polynomial;
// time: O(n log n log e)
Poly pow(Poly a, ll e, int n) {
  Poly t = {1};
  while (e) {
    if (e % 2) (t *= a).resize(n);
    e /= 2; (a *= a).resize(n);
  } // d0c6
  norm(t);
  return t;
} // ada1

// Compute a^e mod m, where a and m are
// polynomials; time: O(|m| log |m| log e)
Poly pow(Poly a, ll e, const Poly& m) {
  Poly t = {1};
  while (e) {
    if (e % 2) t = t*a % m;
    e /= 2; a = a*a % m;
  } // 66ca
  return t;
} // 6f9c

// Derivate polynomial; time: O(n)
Poly derivate(Poly P) {
  if (!P.empty()) {
    rep(i, 1, sz(P)) P[i-1] = P[i]*i;
    P.pop_back();
  } // bd78
  return P;
} // c6c5

// Integrate polynomial; time: O(n)
Poly integrate(Poly P) {
  if (!P.empty()) {
    P.pb(0);
    for (int i = sz(P); --i;) P[i] = P[i-1]/i;
    P[0] = 0;
  } // eec1
  return P;
} // e2f3

// Compute ln(P) mod x^n; time: O(n log n)
Poly log(const Poly& P, int n) {
  Poly a = integrate(derivate(P)*invert(P,n));
  a.resize(n);
  return a;
} // 5d6b

// Compute exp(P) mod x^n; time: O(n lg n)
// Requires P(0) = 0.
Poly exp(Poly P, int n) {
  assert(P.empty() || !P[0].x);
  Poly tmp{P[0]+1}, ret = {1};
  for (int i = 1; i < n; i *= 2) {
```

```
    rep(j, i, min(i*2, sz(P))) tmp.pb(P[j]);
    (ret *= (tmp - log(ret, i*2))).resize(i*2);
  } // c28a
  ret.resize(n);
  return ret;
} // bd42

// Compute sqrt(P) mod x^n; time: O(n log n)
#include "modular_sqrt.h"
bool sqrt(Poly& P, int n) {
  norm(P);
  if (P.empty()) return P.resize(n), 1;

  int tail = 0;
  while (!P[tail].x) tail++;
  if (tail % 2) return 0;

  ll sq = modSqrt(P[tail].x, MOD);
  if (sq == -1) return 0;

  Poly tmp{P[tail]}, ret = {sq};
  for (int i = 1; i < n - tail/2; i *= 2) {
    rep(j, i, min(i*2, sz(P)-tail))
      tmp.pb(P[tail+j]);
    (ret += tmp * invert(ret,i*2)).resize(i*2);
    each(e, ret) e /= 2;
  } // 2d41

  P.resize(tail/2);
  P.insert(P.end(), all(ret));
  P.resize(n);
  return 1;
} // b9b3

// Compute polynomial P(x+c); time: O(n lg n)
Poly shift(Poly P, Zp c) {
  int n = sz(P);
  Poly Q(n, 1);
  Zp fac = 1;
  rep(i, 1, n) {
    P[i] *= (fac *= i);
    Q[n-i-1] = Q[n-i] * c / i;
  } // 1c20
  P *= Q;
  if (sz(P) < n) return {};
  P.erase(P.begin(), P.begin()+n-1);
  fac = 1;
  rep(i, 1, n) P[i] /= (fac *= i);
  return P;
} // b11f

// Compute values P(x^0), ..., P(x^{n-1});
// time: O(n lg n)
Poly chirpz(Poly P, Zp x, int n) {
  int k = sz(P);
  Poly Q(n+k);
  rep(i, 0, n+k) Q[i] = x.pow(i*(i-1)/2);
  rep(i, 0, k) P[i] /= Q[i];
  reverse(all(P));
  P *= Q;
  rep(i, 0, n) P[i] = P[k+i-1] / Q[i];
  P.resize(n);
  return P;
} // 4e8b

// Evaluate polynomial P in given points;
// time: O(n lg^2 n)
Poly eval(const Poly& P, Poly points) {
  int len = 1;
  while (len < sz(points)) len *= 2;

  vector<Poly> tree(len*2, {1});
  rep(i, 0, sz(points))
    tree[len+i] = {-points[i], 1};

  for (int i = len; --i;)
    tree[i] = tree[i*2] * tree[i*2+1];

  tree[0] = P;
  rep(i, 1, len*2)
    tree[i] = tree[i/2] % tree[i];

  rep(i, 0, sz(points)) {
    auto& vec = tree[len+i];
    points[i] = vec.empty() ? 0 : vec[0];
  } // c1c2
  return points;
} // 69b0

// Given n points (x, f(x)) compute n-1-degree
// polynomial f that passes through them;
// time: O(n lg^2 n)
// For O(n^2) version see polynomial_interp.h.
Poly interpolate(const vector<pair<Zp,Zp>>& P){
  int len = 1;
  while (len < sz(P)) len *= 2;

  vector<Poly> mult(len*2, {1}), tree(len*2);
  rep(i, 0, sz(P))
    mult[len+i] = {-P[i].x, 1};

  for (int i = len; --i;)
    mult[i] = mult[i*2] * mult[i*2+1];

  tree[0] = derivate(mult[1]);
  rep(i, 1, len*2)
    tree[i] = tree[i/2] % mult[i];

  rep(i, 0, sz(P))
    tree[len+i][0] = P[i].y / tree[len+i][0];

  for (int i = len; --i;)
    tree[i] = tree[i*2]*mult[i*2+1]
            + mult[i*2]*tree[i*2+1];
  return tree[1];
} // b706
```

math/polynomial_interp.h a4cc

```
// Interpolate set of points (i, vec[i])
// and return it evaluated at x; time: O(n)
template<class T>
T polyExtend(vector<T>& vec, T x) {
  int n = sz(vec);
  vector<T> fac(n, 1), suf(n, 1);

  rep(i, 1, n) fac[i] = fac[i-1] * i;
  for (int i=n; --i;) suf[i-1] = suf[i]*(x-i);

  T pref = 1, ret = 0;
  rep(i, 0, n) {
    T d = fac[i] * fac[n-i-1] * ((n-i)%2*2-1);
    ret += vec[i] * suf[i] * pref / d;
    pref *= x-i;
  } // 681d
  return ret;
} // dd92

// Given n points (x, f(x)) compute n-1-degree
// polynomial f that passes through them;
// time: O(n^2)
// For O(n lg^2 n) version see polynomial.h
template<class T>
vector<T> polyInterp(vector<pair<T, T>> P) {
  int n = sz(P);
  vector<T> ret(n), tmp(n);
  T last = 0;
  tmp[0] = 1;

  rep(k, 0, n-1) rep(i, k+1, n)
    P[i].y = (P[i].y-P[k].y) / (P[i].x-P[k].x);

  rep(k, 0, n) rep(i, 0, n) {
    ret[i] += P[k].y * tmp[i];
    swap(last, tmp[i]);
    tmp[i] -= last * P[k].x;
  } // af1c
  return ret;
} // 7c2c
```

math/sieve.h 3f3d

```
constexpr int MAX_P = 1e6;
bitset<MAX_P+1> primes;
Vi primesList;

// Erathostenes sieve; time: O(n lg lg n)
void sieve() {
  primes.set();
  primes.reset(0);
  primes.reset(1);

  for (int i = 2; i*i <= MAX_P; i++)
    if (primes[i])
      for (int j = i*i; j <= MAX_P; j += i)
        primes.reset(j);

  rep(i, 0, MAX_P+1) if (primes[i])
    primesList.pb(i);
} // d5ca
```

math/sieve_factors.h 312d

```
constexpr int MAX_P = 1e6;
Vi factor(MAX_P+1);

// Erathostenes sieve with saving smallest
// factor for each number; time: O(n lg lg n)
void sieve() {
  for (int i = 2; i*i <= MAX_P; i++)
    if (!factor[i])
      for (int j = i*i; j <= MAX_P; j += i)
        if (!factor[j])
          factor[j] = i;

  rep(i,0,MAX_P+1) if (!factor[i]) factor[i]=i;
} // 82b6

// Factorize n <= MAX_P; time: O(lg n)
// Returns pairs (prime, power), sorted
vector<Pii> factorize(ll n) {
  vector<Pii> ret;
  while (n > 1) {
    int f = factor[n];
    if (ret.empty() || ret.back().x != f)
      ret.pb({ f, 1 });
    else
      ret.back().y++;
    n /= f;
  } // 664c
  return ret;
} // bb65
```

math/sieve_segmented.h 849b

```
constexpr int MAX_P = 1e9;
bitset<MAX_P/2+1> primes; // Only odd numbers

// Cache-friendly Erathostenes sieve
// ~1.5s on Intel Core i5 for MAX_P = 10^9
// Memory usage: MAX_P/16 bytes
void sieve() {
  constexpr int SEG_SIZE = 1<<18;
  int pSqrt = int(sqrt(MAX_P)+0.5);
  vector<Pii> dels;
  primes.set();
  primes.reset(0);

  for (int i = 3; i <= pSqrt; i += 2) {
    if (primes[i/2]) {
      int j;
      for (j = i*i; j <= pSqrt; j += i*2)
        primes.reset(j/2);
      dels.pb({ i, j/2 });
    } // 9e62
  } // ff49

  for (int seg = pSqrt/2;
       seg <= sz(primes); seg += SEG_SIZE) {
    int lim = min(seg+SEG_SIZE, sz(primes));
    each(d, dels) for (;d.y < lim; d.y += d.x)
      primes.reset(d.y);
  } // 97ae
} // 6456

bool isPrime(int x) {
  return x == 2 || (x%2 && primes[x/2]);
} // 422c
```

math/simplex.h c4cf

```
using dbl = double;
using Row = vector<dbl>;
using Matrix = vector<Row>;

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

// Simplex algorithm; time: O(nm * pivots)
// Given m x n matrix A, vector b of length m,
// vector c of length n solves the following:
//   maximize c^T x, Ax <= b, x >= 0
// Output vector `x` contains optimal solution
// or some feasible solution in unbounded case.
// Returns objective value if bounded,
// +inf if unbounded, and -inf if no solution.
// You can test if double is inf using `isinf`.
// PARTIALLY TESTED
dbl simplex(const Matrix& A,
            const Row& b, const Row& c,
            Row& x, dbl eps = 1e-8) {
  int m = sz(b), n = sz(c);
  x.assign(n, 0);
  if (!n) return
    *min_element(all(b)) < -eps ? -1/.0 : 0;

  Vi N(n+1), B(m);
  Matrix D(m+2, Row(n+2));

  auto pivot = [&](int r, int s) {
    dbl inv = 1 / D[r][s];
    rep(i, 0, m+2)
      if (i != r && abs(D[i][s]) > eps) {
        dbl tmp = D[i][s] * inv;
        rep(j,0,n+2) D[i][j] -= D[r][j] * tmp;
        D[i][s] = D[r][s] * tmp;
      } // 5281
    each(k, D[r]) k *= inv;
    each(k, D) k[s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }; // f56b

  auto solve = [&](int phase) {
    for (int y = m+phase-1;;) {
      int s = -1, r = -1;
      rep(j, 0, n+1)
```

```
        if (N[j] != -phase) ltj(D[y]);
      if (D[y][s] >= -eps) return 1;
      rep(i, 0, m)
        if (D[i][s] > eps && (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 0;
      pivot(r, s);
    } // 3bef
  }; // 614a

  rep(i, 0, m) {
    copy(all(A[i]), D[i].begin());
    B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];
  } // b705
  rep(j, 0, n) D[m][N[j] = j] = -c[j];
  N[n] = -1; D[m+1][n] = 1;

  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 (!solve(2) || D[m+1][n+1] < -eps)
      return -1/.0;
    rep(i, 0, m) if (B[i] == -1) {
      int s = 0;
      rep(j, 1, n+1) ltj(D[i]);
      pivot(i, s);
    } // 78fd
  } // b52b
  bool ok = solve(1);
  rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
  return ok ? D[m][n+1] : 1/.0;
} // fe6f
```

structures/bitset_plus.h 6737

```
// Undocumented std::bitset features:
// - _Find_first() - returns first bit = 1 or N
// - _Find_next(i) - returns first bit = 1
//                   after i-th bit
//                   or N if not found

// Bitwise operations for vector<bool>
// UNTESTED

#define OP(x) vector<bool>& operator x##=(    \
    vector<bool>& l, const vector<bool>& r) { \
  assert(sz(l) == sz(r));                     \
  auto a = l.begin(); auto b = r.begin();     \
  while (a<l.end()) *a._M_p++ x##= *b._M_p++; \
  return l; } // f164
OP(&)OP(|)OP(^)
```

structures/fenwick_tree.h 8a44

```
// Fenwick tree (BIT tree); space: O(n)
// Default version: prefix sums
struct Fenwick {
  using T = int;
  static constexpr T ID = 0;
  T f(T a, T b) { return a+b; }

  vector<T> s;
  Fenwick(int n = 0) : s(n, ID) {}

  // A[i] = f(A[i], v); time: O(lg n)
  void modify(int i, T v) {
    for (; i < sz(s); i |= i+1) s[i]=f(s[i],v);
  } // a047

  // Get f(A[0], ..., A[i-1]); time: O(lg n)
  T query(int i) {
    T v = ID;
    for (; i > 0; i &= i-1) v = f(v, s[i-1]);
    return v;
  } // 9810

  // Find smallest i such that
  // f(A[0],...,A[i-1]) >= val; time: O(lg n)
  // Prefixes must have non-descreasing values.
  int lowerBound(T val) {
    if (val <= ID) return 0;
    int i = -1, mask = 1;
    while (mask <= sz(s)) mask *= 2;
    T off = ID;

    while (mask /= 2) {
      int k = mask+i;
      if (k < sz(s)) {
        T x = f(off, s[k]);
        if (val > x) i=k, off=x;
      } // de7f
    } // 929c
    return i+2;
  } // 4be9
}; // d9b0
```

structures/fenwick_tree_2d.h 9f31

```
// Fenwick tree 2D (BIT tree 2D); space: O(n*m)
// Default version: prefix sums 2D
// Change s to hashmap for O(q lg^2 n) memory
struct Fenwick2D {
  using T = int;
  static constexpr T ID = 0;
  T f(T a, T b) { return a+b; }

  vector<T> s;
  int w, h;

  Fenwick2D(int n = 0, int m = 0)
    : s(n*m, ID), w(n), h(m) {}

  // A[i,j] = f(A[i,j], v); time: O(lg^2 n)
  void modify(int i, int j, T v) {
    for (; i < w; i |= i+1)
      for (int k = j; k < h; k |= k+1)
        s[i*h+k] = f(s[i*h+k], v);
  } // d46b

  // Query prefix; time: O(lg^2 n)
  T query(int i, int j) {
    T v = ID;
    for (; i>0; i&=i-1)
      for (int k = j; k > 0; k &= k-1)
        v = f(v, s[i*h+k-h-1]);
    return v;
  } // 08cf
}; // e570
```

structures/find_union.h b4de

```
// Disjoint set data structure; space: O(n)
// Operations work in amortized O(alfa(n))
struct FAU {
  Vi G;
  FAU(int n = 0) : G(n, -1) {}

  // Get size of set containing i
  int size(int i) { return -G[find(i)]; }

  // Find representative of set containing i
  int find(int i) {
    return G[i] < 0 ? i : G[i] = find(G[i]);
  } // 5bc1

  // Union sets containing i and j
  bool join(int i, int j) {
    i = find(i); j = find(j);
    if (i == j) return 0;
    if (G[i] > G[j]) swap(i, j);
    G[i] += G[j]; G[j] = i;
    return 1;
  } // c721
}; // 62a4
```

structures/find_union_undo.h ce37

```
// Disjoint set data structure
// with rollback; space: O(n)
// Operations work in O(log(n)) time.
struct RollbackFAU {
  Vi G;
  vector<Pii> his;

  RollbackFAU(int n = 0) : G(n, -1) {}

  // Get size of set containing i
  int size(int i) { return -G[find(i)]; }

  // Find representative of set containing i
  int find(int i) {
    return G[i] < 0 ? i : find(G[i]);
  } // e478

  // Current time (for rollbacks)
  int time() { return sz(his); }

  // Rollback all operations after time `t`
  void rollback(int t) {
    for (int i = time(); t < i--;)
      G[his[i].x] = his[i].y;
    his.resize(t);
  } // 3ef3

  // Union sets containing i and j
  bool join(int i, int j) {
    i = find(i); j = find(j);
    if (i == j) return 0;
    if (G[i] > G[j]) swap(i, j);
    his.pb({i, G[i]});
    his.pb({j, G[j]});
    G[i] += G[j]; G[j] = i;
    return 1;
  } // 1491
}; // da8c
```

structures/hull_offline.h 3030

```
constexpr ll INF = 2e18;
// constexpr double INF = 1e30;
// constexpr double EPS = 1e-9;

// MAX of linear functions; space: O(n)
// Use if you add lines in increasing `a` order
// Default uncommented version is for int64
struct Hull {
  using T = ll; // Or change to double

  struct Line {
    T a, b, end;
    T intersect(const Line& r) const {
      // Version for double:
      //if (r.a-a < EPS) return b>r.b?INF:-INF;
      //return (b-r.b) / (r.a-a);
      if (a==r.a) return b > r.b ? INF : -INF;
      ll u = b-r.b, d = r.a-a;
      return u/d + ((u^d) >= 0 || !(u%d));
    } // f27f
  }; // 10dc

  vector<Line> S;
  Hull() { S.pb({ 0, -INF, INF }); }

  // Insert f(x) = ax+b; time: amortized O(1)
  void push(T a, T b) {
    Line l{a, b, INF};
    while (true) {
      T e = S.back().end=S.back().intersect(l);
      if (sz(S) < 2 || S[sz(S)-2].end < e)
        break;
      S.pop_back();
    } // 044f
    S.pb(l);
  } // 978e

  // Query max(f(x) for each f): time: O(lg n)
  T query(T x) {
    auto t = *upper_bound(all(S), x,
      [](int l, const Line& r) {
        return l < r.end;
      }); // de77
    return t.a*x + t.b;
  } // b8de
}; // 1d64
```

structures/hull_online.h 2a7b

```
constexpr ll INF = 2e18;

// MAX of linear functions online; space: O(n)
struct Hull {
  static bool modeQ; // Toggles operator< mode

  struct Line {
    mutable ll a, b, end;

    ll intersect(const Line& r) const {
      if (a==r.a) return b > r.b ? INF : -INF;
      ll u = b-r.b, d = r.a-a;
      return u/d + ((u^d) >= 0 || !(u%d));
    } // f27f

    bool operator<(const Line& r) const {
      return modeQ ? end < r.end : a < r.a;
    } // cfab
  }; // 6046

  multiset<Line> S;
  Hull() { S.insert({ 0, -INF, INF }); }

  // Updates segment end
  bool update(multiset<Line>::iterator it) {
    auto cur = it++; cur->end = INF;
    if (it == S.end()) return false;
    cur->end = cur->intersect(*it);
    return cur->end >= it->end;
  } // 63b8

  // Insert f(x) = ax+b; time: O(lg n)
  void insert(ll a, ll b) {
    auto it = S.insert({ a, b, INF });
    while (update(it)) it = --S.erase(++it);
    rep(i, 0, 2)
      while (it != S.begin() && update(--it))
        update(it = --S.erase(++it));
  } // 4f69

  // Query max(f(x) for each f): time: O(lg n)
  ll query(ll x) {
    modeQ = 1;
    auto l = *S.upper_bound({ 0, 0, x });
    modeQ = 0;
    return l.a*x + l.b;
  } // 7533
}; // 037e
```

```
bool Hull::modeQ = false;
```

structures/li_chao_tree.h d960

```
// Extended Li Chao tree; space: O(n)
// Let F be a family of functions,
// closed under function addition, such that
// for every f != g from the family F
// there exists x such that:
// f(z) <= g(z) for z <= x, else f(z) >= g(z)
// or
// g(z) <= f(z) for z <= x, else g(z) >= f(z).
// Typically F is family of linear functions.
// DS maintains a sequence c[0], ..., c[n-1]
// under operations max, add, query
// (see comments below for explanations).
// Configure by modifying:
// - T - type of sequence elements
// - Func - represents function from family F
// - ID_ADD - function f that is neutral
//            element for function addition
// - ID_MAX - function f that is neutral
//            element for function max
// TESTED ON RANDS
struct LiChao {
  using T = ll;

  struct Func {
    T a, b; // a*x + b

    // Evaluate function in point x
    T operator()(int x) const { return a*x+b; }

    // Sum of two functions
    Func operator+(Func r) const {
      return {a+r.a, b+r.b};
    } // f911
  }; // 0bed

  static constexpr Func ID_ADD{0, 0};
  static constexpr Func ID_MAX{0, T(-1e9)};

  vector<Func> val, lazy;
  int len;

  // Initialize tree for n elements; time: O(n)
  LiChao(int n = 0) {
    for (len = 1; len < n; len *= 2);
    val.resize(len*2, ID_MAX);
    lazy.resize(len*2, ID_ADD);
  } // c0ba

  void push(int i) {
    if (i < len) rep(j, 0, 2) {
      lazy[i*2+j] = lazy[i*2+j] + lazy[i];
      val[i*2+j] = val[i*2+j] + lazy[i];
    } // 54fc
    lazy[i] = ID_ADD;
  } // 1777

  // For each x in [vb;ve)
  // set c[x] = max(c[x], f(x));
  // time: O(log^2 n) in general case,
  //       O(log n) if [vb;ve) = [0;len)
  void max(int vb, int ve, Func f,
           int i = 1, int b = 0, int e = -1) {
    if (e < 0) e = len;
    if (vb >= e || b >= ve || i >= len*2)
      return;

    int m = (b+e) / 2;
    push(i);

    if (b >= vb && e <= ve) {
      auto& g = val[i];
      if (g(m) < f(m)) swap(g, f);
      if ((g(b) < f(b)) != (g(m) < f(m)))
        max(vb, ve, f, i*2, b, m);
      else
        max(vb, ve, f, i*2+1, m, e);
    } else {
      max(vb, ve, f, i*2, b, m);
      max(vb, ve, f, i*2+1, m, e);
    } // f2c0
  } // dec2

  // For each x in [vb;ve)
  // set c[x] = c[x] + f(x);
  // time: O(log^2 n) in general case,
  //       O(1) if [vb;ve) = [0;len)
  void add(int vb, int ve, Func f,
           int i = 1, int b = 0, int e = -1) {
    if (e < 0) e = len;
    if (vb >= e || b >= ve) return;

    if (b >= vb && e <= ve) {
      lazy[i] = lazy[i] + f;
      val[i] = val[i] + f;
    } else {
      int m = (b+e) / 2;
      push(i);
      max(b, m, val[i], i*2, b, m);
      max(m, e, val[i], i*2+1, m, e);
      val[i] = ID_MAX;
      add(vb, ve, f, i*2, b, m);
      add(vb, ve, f, i*2+1, m, e);
    } // bbe5
  } // 259f

  // Get value of c[x]; time: O(log n)
  T query(int x) {
    int i = x+len;
    T ret = val[i](x);
    while (i /= 2)
      ret = ::max(ret+lazy[i](x), val[i](x));
    return ret;
  } // 0c22
}; // 9530
```

structures/max_queue.h 3e9e

```
// Queue with max query on contained elements
struct MaxQueue {
  using T = int;
  deque<T> Q, M;

  // Add v to the back; time: amortized O(1)
  void push(T v) {
    while (!M.empty() && M.back() < v)
      M.pop_back();
    M.pb(v); Q.pb(v);
  } // 57a2

  // Pop from the front; time: O(1)
  void pop() {
    if (M.front() == Q.front()) M.pop_front();
    Q.pop_front();
  } // 101c

  // Get max element value; time: O(1)
  T max() const { return M.front(); }
}; // b6c4
```

structures/pairing_heap.h b2a7

```
// Pairing heap implementation; space O(n)
// Elements are stored in vector for faster
// allocation. It's MINIMUM queue.
// Allows to merge heaps in O(1)
template<class T, class Cmp = less<T>>
struct PHeap {
  struct Node {
    T val;
    int child{-1}, next{-1}, prev{-1};

    Node(T x = T()) : val(x) {}
  }; // 11ee

  using Vnode = vector<Node>;
  Vnode& M;
  int root{-1};

  int unlink(int& i) {
    if (i >= 0) M[i].prev = -1;
    int x = i; i = -1;
    return x;
  } // d9f6

  void link(int host, int& i, int val) {
    if (i >= 0) M[i].prev = -1;
    i = val;
    if (i >= 0) M[i].prev = host;
  } // 47d5

  int merge(int l, int r) {
    if (l < 0) return r;
    if (r < 0) return l;
    if (Cmp()(M[l].val, M[r].val)) swap(l, r);

    link(l, M[l].next, unlink(M[r].child));
    link(r, M[r].child, l);
    return r;
  } // fc42

  int mergePairs(int v) {
    if (v < 0 || M[v].next < 0) return v;
    int v2 = unlink(M[v].next);
    int v3 = unlink(M[v2].next);
    return merge(merge(v, v2), mergePairs(v3));
  } // 2eea

  // ---

  // Initialize heap with given node storage
  // Just declare 1 Vnode and pass it to heaps
  PHeap(Vnode& mem) : M(mem) {}

  // Add given key to heap, returns index; O(1)
  int push(const T& x) {
    int index = sz(M);
    M.emplace_back(x);
    root = merge(root, index);
    return index;
  } // e744

  // Change key of i to smaller value; O(1)
  void decrease(int i, T val) {
    assert(!Cmp()(M[i].val, val));
    M[i].val = val;

    int prev = M[i].prev;
    if (prev < 0) return;

    auto& p = M[prev];
    link(prev, (p.child == i ? p.child
          : p.next), unlink(M[i].next));

    root = merge(root, i);
  } // 1a67

  bool empty() { return root < 0; }
  const T& top() { return M[root].val; }

  // Merge with other heap. Must use same vec.
  void merge(PHeap& r) { // time: O(1)
    assert(&M == &r.M);
    root = merge(root, r.root); r.root = -1;
  } // 9623

  // Remove min element; time: O(lg n)
  void pop() {
    root = mergePairs(unlink(M[root].child));
  } // 5b13
}; // 09f3
```

structures/rmq.h 215c

```
// Range Minimum Query; space: O(n lg n)
struct RMQ {
  using T = int;
  static constexpr T ID = INT_MAX;
  T f(T a, T b)  { return min(a, b); }

  vector<vector<T>> s;

  // Initialize RMQ structure; time: O(n lg n)
  RMQ(const vector<T>& vec = {}) {
    s = {vec};
    for (int h = 1; h <= sz(vec); h *= 2) {
      s.pb({});
      auto& prev = s[sz(s)-2];
      rep(i, 0, sz(vec)-h*2+1)
        s.back().pb(f(prev[i], prev[i+h]));
    } // 7c37
  } // 14ed

  // Query f(s[b], ... ,s[e-1]); time: O(1)
  T query(int b, int e) {
    if (b >= e) return ID;
    int k = 31 - __builtin_clz(e-b);
    return f(s[k][b], s[k][e - (1<<k)]);
  } // caaa
}; // 5369
```

structures/segtree_config.h af25

```
// Segment tree configurations to be used
// in segtree_general and segtree_persistent.
// See comments in TREE_PLUS version
// to understand how to create custom ones.
// Capabilities notation: (update; query)

#if TREE_PLUS // (+; sum, max, max count)
  // time: O(lg n)
  using T = int; // Data type for update
                 // operations (lazy tag)
  static constexpr T ID = 0; // Neutral value
                             // for updates and lazy tags

  // This structure keeps aggregated data
  struct Agg {
    // Aggregated data: sum, max, max count
    // Default values should be neutral
    // values, i.e. "aggregate over empty set"
    T sum{0}, vMax{INT_MIN}, nMax{0};

    // Initialize as leaf (single value)
    void leaf() { sum = vMax = 0; nMax = 1; }

    // Combine data with aggregated data
    // from node to the right
    void merge(const Agg& r) {
      if (vMax < r.vMax) nMax = r.nMax;
      else if (vMax == r.vMax) nMax += r.nMax;
      vMax = max(vMax, r.vMax);
      sum += r.sum;
    } // 8850
```

```
    // Apply update provided in `x`:
    // - update aggregated data
    // - update lazy tag `lazy`
    // - `size` is amount of elements
    // - return 0 if update should branch
    //   (to be used in "segement tree beats")
    // - if you change value of `x` changed
    //   value will be passed to next node
    //   to the right during updates
    bool apply(T& lazy, T& x, int size) {
      lazy += x;
      sum += x*size;
      vMax += x;
      return 1;
    } // 4858
  }; // 9bf5
#elif TREE_MAX // (max; max, max count)
  // time: O(lg n)
  using T = int;
  static constexpr T ID = INT_MIN;

  struct Agg {
    // Aggregated data: max value, max count
    T vMax{INT_MIN}, nMax{0};
    void leaf() { vMax = 0; nMax = 1; }

    void merge(const Agg& r) {
      if (vMax < r.vMax) nMax = r.nMax;
      else if (vMax == r.vMax) nMax += r.nMax;
      vMax = max(vMax, r.vMax);
    } // f56b

    bool apply(T& lazy, T& x, int size) {
      if (vMax <= x) nMax = size;
      lazy = max(lazy, x);
      vMax = max(vMax, x);
      return 1;
    } // 8bd5
  }; // 15b6
#elif TREE_SET // (=; sum, max, max count)
  // time: O(lg n)
  // Set ID to some unused value.
  using T = int;
  static constexpr T ID = INT_MIN;

  struct Agg {
    // Aggregated data: sum, max, max count
    T sum{0}, vMax{INT_MIN}, nMax{0};
    void leaf() { sum = vMax = 0; nMax = 1; }

    void merge(const Agg& r) {
      if (vMax < r.vMax) nMax = r.nMax;
      else if (vMax == r.vMax) nMax += r.nMax;
      vMax = max(vMax, r.vMax);
      sum += r.sum;
    } // 8850

    bool apply(T& lazy, T& x, int size) {
      lazy = x;
      sum = x*size;
      vMax = x;
      nMax = size;
      return 1;
    } // 845b
  }; // 7488
#elif TREE_BEATS // (+, min; sum, max)
  // time: amortized O(lg n) if not using +
  //       amortized O(lg^2 n) if using +
  // Lazy tag is pair (add, min).
  // To add x: run update with {x, INT_MAX},
  // to min x: run update with {0, x}.
  // When both parts are provided addition
  // is applied first, then minimum.
  using T = Pii;
  static constexpr T ID = {0, INT_MAX};

  struct Agg {
    // Aggregated data: max value, max count,
    //                  second max value, sum
    int vMax{INT_MIN}, nMax{0}, max2{INT_MIN};
    int sum{0};
    void leaf() { sum = vMax = 0; nMax = 1; }

    void merge(const Agg& r) {
      if (r.vMax > vMax) {
        max2 = vMax;
        vMax = r.vMax;
        nMax = r.nMax;
      } else if (r.vMax == vMax) {
        nMax += r.nMax;
      } else if (r.vMax > max2) {
        max2 = r.vMax;
      } // b074
      max2 = max(max2, r.max2);
      sum += r.sum;
    } // 3124

    bool apply(T& lazy, T& x, int size) {
      if (max2 != INT_MIN && max2+x.x >= x.y)
        return 0;

      lazy.x += x.x;
      sum += x.x*size;
      vMax += x.x;
      if (max2 != INT_MIN) max2 += x.x;

      if (x.y < vMax) {
        sum -= (vMax-x.y) * nMax;
        vMax = x.y;
      } // 7025
      lazy.y = vMax;
      return 1;
    } // fe0c
  }; // 2924
#endif
```

structures/segtree_general.h 725a

```
// Highly configurable statically allocated
// (interval; interval) segment tree;
// space: O(n)
struct SegTree {
  // Choose/write configuration
  #include "segtree_config.h"

  // Root node is 1, left is i*2, right i*2+1
  vector<Agg> agg; // Aggregated data for nodes
  vector<T> lazy;  // Lazy tags for nodes
  int len{1};      // Number of leaves

  // Initialize tree for n elements; time: O(n)
  SegTree(int n = 0) {
    while (len < n) len *= 2;
    agg.resize(len*2);
    lazy.resize(len*2, ID);
    rep(i, 0, n) agg[len+i].leaf();
    for (int i = len; --i;)
      (agg[i] = agg[i*2]).merge(agg[i*2+1]);
  } // 4417

  void push(int i, int s) {
    if (lazy[i] != ID) {
      agg[i*2].apply(lazy[i*2], lazy[i], s/2);
      agg[i*2+1].apply(lazy[i*2+1],
                       lazy[i], s/2);
      lazy[i] = ID;
    } // 3ba9
  } // 5d19

  // Modify interval [vb;ve) with val; O(lg n)
  T update(int vb, int ve, T val, int i = 1,
           int b = 0, int e = -1) {
    if (e < 0) e = len;
    if (vb >= e || b >= ve) return val;

    if (b >= vb && e <= ve &&
        agg[i].apply(lazy[i], val, e-b))
      return val;

    int m = (b+e) / 2;
    push(i, e-b);
    val = update(vb, ve, val, i*2, b, m);
    val = update(vb, ve, val, i*2+1, m, e);
    (agg[i] = agg[i*2]).merge(agg[i*2+1]);
    return val;
  } // aa8e

  // Query interval [vb;ve); time: O(lg n)
  Agg query(int vb, int ve, int i = 1,
            int b = 0, int e = -1) {
    if (e < 0) e = len;
    if (vb >= e || b >= ve) return {};
    if (b >= vb && e <= ve) return agg[i];

    int m = (b+e) / 2;
    push(i, e-b);
    Agg t = query(vb, ve, i*2, b, m);
    t.merge(query(vb, ve, i*2+1, m, e));
    return t;
  } // 1a1e
}; // db5c
```

structures/segtree_persist.h dcfc

```
// Highly configurable (interval; interval)
// persistent segment tree;
// space: O(queries lg n)
// First tree version number is 0.
struct SegTree {
  // Choose/write configuration
  #include "segtree_config.h"

  vector<Agg> agg; // Aggregated data for nodes
  vector<T> lazy;  // Lazy tags for nodes
  vector<bool> cow; // Copy children on push?
  Vi L, R;         // Children links
  int len{1};      // Number of leaves

  // Initialize tree for n elements; O(lg n)
  SegTree(int n = 0) {
    int k = 1;
    while (len < n) len *= 2, k++;

    agg.resize(k);
    lazy.resize(k, ID);
    cow.resize(k, 1);
    L.resize(k);
    R.resize(k);
    agg[--k].leaf();

    while (k--) {
      (agg[k] = agg[k+1]).merge(agg[k+1]);
      L[k] = R[k] = k+1;
    } // 211f
  } // 83cf

  // New version from version `i`; time: O(1)
  // First version number is 0.
  int fork(int i) {
    L.pb(L[i]); R.pb(R[i]); cow.pb(cow[i] = 1);
    agg.pb(agg[i]); lazy.pb(lazy[i]);
    return sz(L)-1;
  } // a21b

  void push(int i, int s, bool w) {
    bool has = (lazy[i] != ID);
    if ((has || w) && cow[i]) {
      int a = fork(L[i]), b = fork(R[i]);
      L[i] = a; R[i] = b; cow[i] = 0;
    } // 1a3e
    if (has) {
      agg[L[i]].apply(lazy[L[i]],lazy[i],s/2);
      agg[R[i]].apply(lazy[R[i]],lazy[i],s/2);
      lazy[i] = ID;
    } // eca6
  } // 9f84

  // Modify interval [vb;ve) with val
  // in tree version `i`; time: O(lg n)
  T update(int i, int vb, int ve, T val,
           int b = 0, int e = -1) {
    if (e < 0) e = len;
    if (vb >= e || b >= ve) return val;

    if (b >= vb && e <= ve &&
        agg[i].apply(lazy[i], val, e-b))
      return val;

    int m = (b+e) / 2;
    push(i, e-b, 1);
    val = update(L[i], vb, ve, val, b, m);
    val = update(R[i], vb, ve, val, m, e);
    (agg[i] = agg[L[i]]).merge(agg[R[i]]);
    return val;
  } // 776e

  // Query interval [vb;ve)
  // in tree version `i`; time: O(lg n)
  Agg query(int i, int vb, int ve,
            int b = 0, int e = -1) {
    if (e < 0) e = len;
    if (vb >= e || b >= ve) return {};
    if (b >= vb && e <= ve) return agg[i];

    int m = (b+e) / 2;
    push(i, e-b, 0);
    Agg t = query(L[i], vb, ve, b, m);
    t.merge(query(R[i], vb, ve, m, e));
    return t;
  } // abf4
}; // 8a44
```

structures/segtree_point.h ff25

```
// Segment tree (point, interval)
// Configure by modifying:
// - T - stored data type
// - ID - neutral element for query operation
// - f(a, b) - combine results
struct SegTree {
  using T = int;
  static constexpr T ID = INT_MIN;
  T f(T a, T b) { return max(a,b); }

  vector<T> V;
  int len;

  // Initialize tree for n elements; time: O(n)
  SegTree(int n = 0, T def = 0) {
    for (len = 1; len < n; len *= 2);
    V.resize(len*2, ID);
```

```
    rep(i, 0, n) V[len+i] = def;
    for (int i = len; --i;)
      V[i] = f(V[i*2], V[i*2+1]);
  } // 459e

  // Set element `i` to `val`; time: O(lg n)
  void set(int i, T val) {
    V[i += len] = val;
    while (i /= 2)
      V[i] = f(V[i*2], V[i*2+1]);
  } // 4bcd

  // Query interval [b;e); time: O(lg n)
  T query(int b, int e) {
    b += len; e += len-1;
    if (b > e)  return ID;
    if (b == e) return V[b];
    T x = f(V[b], V[e]);

    while (b/2 < e/2) {
      if (~b&1) x = f(x, V[b^1]);
      if (e&1)  x = f(x, V[e^1]);
      b /= 2; e /= 2;
    } // 444a
    return x;
  } // de36
}; // c178
```

structures/treap.h 0da3

```
// "Set" of implicit keyed treaps; space: O(n)
// Nodes are keyed by their indices in array
// of all nodes. Treap key is key of its root.
// "Node x" means "node with key x".
// "Treap x" means "treap with key x".
// Key -1 is "null".
// Put any additional data in Node struct.
struct Treap {
  struct Node {
    // E[0] = left child, E[1] = right child
    // weight = node random weight (for treap)
    // size = subtree size, par = parent node
    int E[2] = {-1, -1}, weight{rand()};
    int size{1}, par{-1};
    bool flip{0}; // Is interval reversed?
  }; // c082

  vector<Node> G; // Array of all nodes

  // Initialize structure for n nodes
  // with keys 0, ..., n-1; time: O(n)
  // Each node is separate treap,
  // use join() to make sequence.
  Treap(int n = 0) : G(n) {}

  // Create new treap (a single node),
  // returns its key; time: O(1)
  int make() { G.pb({}); return sz(G)-1; }

  // Get size of node x subtree. x can be -1.
  int size(int x) { // time: O(1)
    return (x >= 0 ? G[x].size : 0);
  } // 81cf

  // Propagate down data (flip flag etc).
  // x can be -1; time: O(1)
  void push(int x) {
    if (x >= 0 && G[x].flip) {
      G[x].flip = 0;
      swap(G[x].E[0], G[x].E[1]);
      each(e, G[x].E) if (e>=0) G[e].flip ^= 1;
    } // + any other lazy operations
  } // ed19

  // Update aggregates of node x.
  // x can be -1; time: O(1)
  void update(int x) {
    if (x >= 0) {
      int& s = G[x].size = 1;
      G[x].par = -1;
      each(e, G[x].E) if (e >= 0) {
        s += G[e].size;
        G[e].par = x;
      } // f7a7
    } // + any other aggregates
  } // 46a3

  // Split treap x into treaps l and r
  // such that l contains first i elements
  // and r the remaining ones.
  // x, l, r can be -1; time: ~O(lg n)
  void split(int x, int& l, int& r, int i) {
    push(x); l = r = -1;
    if (x < 0) return;
    int key = size(G[x].E[0]);
    if (i <= key) {
      split(G[x].E[0], l, G[x].E[0], i);
      r = x;
    } else {
      split(G[x].E[1], G[x].E[1], r, i-key-1);
      l = x;
    } // fe19
    update(x);
  } // 8211

  // Join treaps l and r into one treap
  // such that elements of l are before
  // elements of r. Returns new treap.
  // l, r and returned value can be -1.
  int join(int l, int r) { // time: ~O(lg n)
    push(l); push(r);
    if (l < 0 || r < 0) return max(l, r);

    if (G[l].weight < G[r].weight) {
      G[l].E[1] = join(G[l].E[1], r);
      update(l);
      return l;
    } // 18c7

    G[r].E[0] = join(l, G[r].E[0]);
    update(r);
    return r;
  } // b559

  // Find i-th node in treap x.
  // Returns its key or -1 if not found.
  // x can be -1; time: ~O(lg n)
  int find(int x, int i) {
    while (x >= 0) {
      push(x);
      int key = size(G[x].E[0]);
      if (key == i) return x;
      x = G[x].E[key < i];
      if (key < i) i -= key+1;
    } // 054c
    return -1;
  } // 0b9b

  // Get key of treap containing node x
  // (key of treap root). x can be -1.
  int root(int x) { // time: ~O(lg n)
    while (G[x].par >= 0) x = G[x].par;
    return x;
  } // be8b

  // Get position of node x in its treap.
  // x is assumed to NOT be -1; time: ~O(lg n)
  int index(int x) {
    int p, i = size(G[x].E[G[x].flip]);
    while ((p = G[x].par) >= 0) {
      if (G[p].E[1] == x) i+=size(G[p].E[0])+1;
      if (G[p].flip) i = G[p].size-i-1;
      x = p;
    } // 3f81
    return i;
  } // ddad

  // Reverse interval [l;r) in treap x.
  // Returns new key of treap; time: ~O(lg n)
  int reverse(int x, int l, int r) {
    int a, b, c;
    split(x, b, c, r);
    split(b, a, b, l);
    if (b >= 0) G[b].flip ^= 1;
    return join(join(a, b), c);
  } // e418
}; // 73f2
```

structures/wavelet_tree.h 1133

```
// Wavelet tree ("merge-sort tree over values")
// Each node represent interval of values.
// seq[1]     = original sequence
// seq[i]     = subsequence with values
//              represented by i-th node
// left[i][j] = how many values in seq[0:j)
//              go to left subtree
struct WaveletTree {
  vector<Vi> seq, left;
  int len;

  WaveletTree() {}

  // Build wavelet tree for sequence `elems`;
  // time and space: O((n+maxVal) log maxVal)
  // Values are expected to be in [0;maxVal).
  WaveletTree(const Vi& elems, int maxVal) {
    for (len = 1; len < maxVal; len *= 2);
    seq.resize(len*2);
    left.resize(len*2);
    seq[1] = elems;
    build(1, 0, len);
  } // a5e9

  void build(int i, int b, int e) {
    if (i >= len) return;
    int m = (b+e) / 2;
    left[i].pb(0);
    each(x, seq[i]) {
      left[i].pb(left[i].back() + (x < m));
      seq[i*2 + (x >= m)].pb(x);
    } // ac25
    build(i*2, b, m);
    build(i*2+1, m, e);
  } // 8153

  // Find k-th smallest element in [begin;end)
  // [begin;end); time: O(log maxVal)
  int kth(int begin, int end, int k, int i=1) {
    if (i >= len) return seq[i][0];
    int x = left[i][begin], y = left[i][end];
    if (k < y-x) return kth(x, y, k, i*2);
    return kth(begin-x, end-y, k-y+x, i*2+1);
  } // 7861

  // Count number of elements >= vb and < ve
  // in [begin;end); time: O(log maxVal)
  int count(int begin, int end, int vb, int ve,
            int i = 1, int b = 0, int e = -1) {
    if (e < 0) e = len;
    if (b >= ve || vb >= e) return 0;
    if (b >= vb && e <= ve) return end-begin;
    int m = (b+e) / 2;
    int x = left[i][begin], y = left[i][end];
    return count(x, y, vb, ve, i*2, b, m) +
      count(begin-x, end-y, vb, ve, i*2+1,m,e);
  } // 71cf
}; // 3f09
```

structures/ext/hash_table.h 2d30

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
// gp_hash_table<K, V> = faster unordered_set

// Anti-anti-hash
const size_t HXOR = mt19937_64(time(0))();
template<class T> struct SafeHash {
  size_t operator()(const T& x) const {
    return hash<T>()(x ^ T(HXOR));
  } // 3a78
}; // 7d0e
```

structures/ext/rope.h 051f

```
#include <ext/rope>
using namespace __gnu_cxx;
// rope<T> = implicit cartesian tree
```

structures/ext/tree.h a3bc

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

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

// Standard set functions and:
// t.order_of_key(key) - index of first >= key
// t.find_by_order(i) - find i-th element
// t1.join(t2) - assuming t1<>t2 merge t2 to t1
```

structures/ext/trie.h 5cc2

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/trie_policy.hpp>
using namespace __gnu_pbds;

using pref_trie = trie<
  string, null_type,
  trie_string_access_traits<>, pat_trie_tag,
  trie_prefix_search_node_update
>;
```

text/aho_corasick.h 697d

```
constexpr char AMIN = 'a'; // Smallest letter
constexpr int ALPHA = 26;  // Alphabet size

// Aho-Corasick algorithm for linear-time
// multiple pattern matching.
// Add patterns using add(), then call build().
struct Aho {
  vector<array<int, ALPHA>> nxt{1};
  Vi suf = {-1}, accLink = {-1};
  vector<Vi> accept{1};

  // Add string with given ID to structure
  // Returns index of accepting node
  int add(const string& str, int id) {
```

```
    int i = 0;
    each(c, str) {
      if (!nxt[i][c-AMIN]) {
        nxt[i][c-AMIN] = sz(nxt);
        nxt.pb({}); suf.pb(-1);
        accLink.pb(1); accept.pb({});
      } // 5ead
      i = nxt[i][c-AMIN];
    } // ace9
    accept[i].pb(id);
    return i;
  } // 27c8

  // Build automata; time: O(V*ALPHA)
  void build() {
    queue<int> que;
    each(e, nxt[0]) if (e) {
      suf[e] = 0; que.push(e);
    } // c34d
    while (!que.empty()) {
      int i = que.front(), s = suf[i], j = 0;
      que.pop();
      each(e, nxt[i]) {
        if (e) que.push(e);
        (e ? suf[e] : e) = nxt[s][j++];
      } // 8521
      accLink[i] = (accept[s].empty() ?
          accLink[s] : s);
    } // 1e8a
  } // 2561

  // Append `c` to state `i`
  int next(int i, char c) {
    return nxt[i][c-AMIN];
  } // 6bb7

  // Call `f` for each pattern accepted
  // when in state `i` with its ID as argument.
  // Return true from `f` to terminate early.
  // Calls are in descreasing length order.
  template<class F> void accepted(int i, F f) {
    while (i != -1) {
      each(a, accept[i]) if (f(a)) return;
      i = accLink[i];
    } // c175
  } // 1f0d
}; // 2768
```

text/alcs.h 16fd

```
// All-substrings common sequences algorithm.
// Given strings A and B, algorithm computes:
//   C(i,j,k) = |LCS(A[:i), B[j:k))|
// in compressed form; time and space: O(n^2)
// To describe the compression, note that:
// 1. C(i,j,k-1) <= C(i,j,k) <= C(i,j,k-1)+1
// 2. If j < k and C(i,j,k) = C(i,j,k-1)+1,
//    then C(i,j+1,k) = C(i,j+1,k-1)+1
// 3. If j >= k, then C(i,j,k) = 0
// This allows us to store just the following:
//   ih(i,k) = min j s.t. C(i,j,k-1) < C(i,j,k)
struct ALCS {
  string A, B;
  vector<Vi> ih;

  ALCS() {}

  // Precompute compressed matrix; time: O(nm)
  ALCS(string s, string t) : A(s), B(t) {
    int n = sz(A), m = sz(B);
    ih.resize(n+1, Vi(m+1));
    iota(all(ih[0]), 0);
    rep(l, 1, n+1) {
      int iv = 0;
      rep(j, 1, m+1) {
        if (A[l-1] != B[j-1]) {
          ih[l][j] = max(ih[l-1][j], iv);
          iv = min(ih[l-1][j], iv);
        } else {
          ih[l][j] = iv;
          iv = ih[l-1][j];
        } // 7af8
      } // d115
    } // baff
  } // 3e1d

  // Compute |LCS(A[:i), B[j:k))|; time: O(k-j)
  // Note: You can precompute data structure
  // to answer these queries in O(log n)
  // or compute all answers for fixed `i`.
  int operator()(int i, int j, int k) {
    int ret = 0;
    rep(q, j, k) ret += (ih[i][q+1] <= j);
    return ret;
  } // dabf

  // Compute subsequence LCS(A[:i), B[j:k));
  // time: O(k-j)
  string recover(int i, int j, int k) {
    string ret;
    while (i > 0 && j < k) {
      if (ih[i][k--] <= j) {
        ret.pb(B[k]);
        while (A[--i] != B[k]);
      } // 9d77
    } // 1edf
    reverse(all(ret));
    return ret;
  } // 738c

  // Compute LCS'es of given prefix of A,
  // and all prefixes of given suffix of B.
  // Returns vector L of length |B|+1 s.t.
  // L[k] = |LCS(A[:i), B[j:k))|; time: O(|B|)
  Vi row(int i, int j) {
    Vi ret(sz(B)+1);
    rep(k, j+1, sz(ret))
      ret[k] = ret[k-1] + (ih[i][k] <= j);
    return ret;
  } // e5bd

  // Compute LCS'es of given prefix of A,
  // and all substrings of B; time: O(n^2)
  // Return matrix M such that:
  // M[j][k] = |LCS(A[:i), B[j:j+k))|
  vector<Vi> matrix(int i) {
    vector<Vi> ret;
    rep(j, 0, sz(B)+1) ret.pb(row(i, j));
    return ret;
  } // b4b4
}; // 30e0
```

text/kmp.h 5729

```
// Computes prefsuf array; time: O(n)
// ps[i] = max prefsuf of [0;i); ps[0] := -1
template<class T> Vi kmp(const T& str) {
  Vi ps; ps.pb(-1);
  each(x, str) {
    int k = ps.back();
    while (k >= 0 && str[k] != x) k = ps[k];
    ps.pb(k+1);
  } // 05aa
  return ps;
} // 8a6c

// Finds occurences of pat in vec; time: O(n)
// Returns starting indices of matches.
template<class T>
Vi match(const T& str, T pat) {
  int n = sz(pat);
  pat.pb(-1); // SET TO SOME UNUSED CHARACTER
  pat.insert(pat.end(), all(str));
  Vi ret, ps = kmp(pat);
  rep(i, 0, sz(ps)) {
    if (ps[i] == n) ret.pb(i-2*n-1);
  } // a1e9
  return ret;
} // f986
```

text/kmr.h ee8c

```
// KMR algorithm for O(1) lexicographical
// comparison of substrings.
struct KMR {
  vector<Vi> ids;

  KMR() {}

  // Initialize structure; time: O(n lg^2 n)
  // You can change str type to Vi freely.
  KMR(const string& str) {
    ids.clear();
    ids.pb(Vi(all(str)));

    for (int h = 1; h <= sz(str); h *= 2) {
      vector<pair<Pii, int>> tmp;

      rep(j, 0, sz(str)) {
        int a = ids.back()[j], b = -1;
        if (j+h < sz(str)) b = ids.back()[j+h];
        tmp.pb({ {a, b}, j });
      } // a210

      sort(all(tmp));
      ids.emplace_back(sz(tmp));

      int n = 0;
      rep(j, 0, sz(tmp)) {
        if (j > 0 && tmp[j-1].x != tmp[j].x)
          n++;
        ids.back()[tmp[j].y] = n;
      } // bd2e
    } // 969a
  } // a40e

  // Get representative of [begin;end); O(1)
  Pii get(int begin, int end) {
    if (begin >= end) return {0, 0};
    int k = 31 - __builtin_clz(end-begin);
    return {ids[k][begin], ids[k][end-(1<<k)]};
  } // 85f3

  // Compare [b1;e1) with [b2;e2); O(1)
  // Returns -1 if <, 0 if ==, 1 if >
  int cmp(int b1, int e1, int b2, int e2) {
    int l1 = e1-b1, l2 = e2-b2;
    int l = min(l1, l2);
    Pii x = get(b1, b1+l), y = get(b2, b2+l);

    if (x == y) return (l1 > l2) - (l1 < l2);
    return (x > y) - (x < y);
  } // bf42

  // Compute suffix array of string; O(n)
  Vi sufArray() {
    Vi sufs(sz(ids.back()));
    rep(i, 0, sz(ids.back()))
      sufs[ids.back()[i]] = i;
    return sufs;
  } // d98d
}; // 457e
```

text/lcp.h 0c65

```
// Compute Longest Common Prefix array for
// given string and it's suffix array; O(n)
// In order to compute suffix array use kmr.h
// or suffix_array_linear.h
template<class T>
Vi lcpArray(const T& str, const Vi& sufs) {
  int n = sz(str), k = 0;
  Vi pos(n), lcp(n-1);
  rep(i, 0, n) pos[sufs[i]] = i;
  rep(i, 0, n) {
    if (pos[i] < n-1) {
      int j = sufs[pos[i]+1];
      while (i+k < n && j+k < n &&
          str[i+k] == str[j+k]) k++;
      lcp[pos[i]] = k;
    } // 2cba
    if (k > 0) k--;
  } // 8b22
  return lcp;
} // d438
```

text/lyndon_factorization.h 688c

```
// Compute Lyndon factorization for s; O(n)
// Word is simple iff it's stricly smaller
// than any of it's nontrivial suffixes.
// Lyndon factorization is division of string
// into non-increasing simple words.
// It is unique.
vector<string> duval(const string& s) {
  int n = sz(s), i = 0;
  vector<string> ret;
  while (i < n) {
    int j = i+1, k = i;
    while (j < n && s[k] <= s[j])
      k = (s[k] < s[j] ? i : k+1), j++;
    while (i <= k)
      ret.pb(s.substr(i, j-k)), i += j-k;
  } // 3f17
  return ret;
} // 0e48
```

text/main_lorentz.h 70b7

```
#include "z_function.h"

struct Sqr {
  int begin, end, len;
}; // f012

// Main-Lorentz algorithm for finding
// all squares in given word; time: O(n lg n)
// Results are in compressed form:
// (b, e, l) means that for each b <= i < e
// there is square at position i of size 2l.
// Each square is present in only one interval.
vector<Sqr> lorentz(const string& s) {
  vector<Sqr> ans;
  Vi pos(sz(s)/2+2, -1);

  rep(mid, 1, sz(s)) {
    int part = mid & ~(mid-1), off = mid-part;
    int end = min(mid+part, sz(s));
    auto a = s.substr(off, part);
    auto b = s.substr(mid, end-mid);
```

```
    string ra(a.rbegin(), a.rend());
    string rb(b.rbegin(), b.rend());

    rep(j, 0, 2) {
      // Set # to some unused character!
      Vi z1 = prefPref(ra);
      Vi z2 = prefPref(b+"#"+a);
      z1.pb(0); z2.pb(0);

      rep(c, 0, sz(a)) {
        int l = sz(a)-c;
        int x = c - min(l-1, z1[l]);
        int y = c - max(l-z2[sz(b)+c+1], j);
        if (x > y) continue;

        int sb = (j ? end-y-l*2 : off+x);
        int se = (j ? end-x-l*2+1 : off+y+1);
        int& p = pos[l];

        if (p != -1 && ans[p].end == sb)
          ans[p].end = se;
        else
          p = sz(ans), ans.pb({sb, se, l});
      } // af4b

      a.swap(rb);
      b.swap(ra);
    } // d147
  } // a9bb

  return ans;
} // 0562
```

text/manacher.h 8680

```
// Manacher algorithm; time: O(n)
// Finds largest radiuses for palindromes:
// r[2*i] = for center at i (single letter = 1)
// r[2*i+1] = for center between i and i+1
template<class T> Vi manacher(const T& str) {
  int n = sz(str)*2, c = 0, e = 1;
  Vi r(n, 1);
  auto get = [&](int i) { return i%2 ? 0 :
    (i >= 0 && i < n ? str[i/2] : i); }; // 3d98

  rep(i, 0, n) {
    if (i < e) r[i] = min(r[c*2-i], e-i);
    while (get(i-r[i]) == get(i+r[i])) r[i]++;
    if (i+r[i] > e) c = i, e = i+r[i]-1;
  } // 0f87

  rep(i, 0, n) r[i] /= 2;
  return r;
} // a300
```

text/min_rotation.h e4d6

```
// Find lexicographically smallest
// rotation of s; time: O(n)
// Returns index where shifted word starts.
// You can use std::rotate to get the word:
// rotate(s.begin(), s.begin()+minRotation(s),
//        s.end());
int minRotation(string s) {
  int a = 0, n = sz(s); s += s;
  rep(b, 0, n) rep(i, 0, n) {
    if (a+i == b || s[a+i] < s[b+i]) {
      b += max(0, i-1); break;
    } // 865b
    if (s[a+i] > s[b+i]) {
      a = b; break;
    } // 7628
  } // 40be
  return a;
```

```
} // 9ed8
```

text/monge.h b5e3

```
// NxN matrix A is simple (sub-)unit-Monge
// iff there exists a (sub-)permutation
// (N-1)x(N-1) matrix P such that:
//   A[x,y] = sum i>=x, j<y: P[i,j]
// The first column and last row are always 0.
// We represent these matrices implicitly
// using permutations p s.t. P[i,p(i)] = 1.

// (min, +) product of simple unit-Monge
// matrices represented by permutations P, Q,
// is also a simple unit-Monge matrix.
// The permutation that describes the product
// can be obtained by the following procedure:
// 1. Decompose P, Q into minimal sequences of
//    elementary transpositions.
// 2. Concatenate the transposition sequences.
// 3. Scan from left to right and remove
//    transpositions that decrease
//    inversion count (i.e. second crossings).
// 4. The reduced sequence represents result.

// Invert sub-permutation with values [0;n).
// Missing values should have value `def`.
Vi invert(const Vi& P, int n, int def) {
  Vi ret(n, def);
  rep(i, 0, sz(P)) if (P[i] != def)
    ret[P[i]] = i;
  return ret;
} // b5a6

// Split permutation P into half `lo`
// with values less than `k`, and half `hi`
// with remaining values, shifted by `k`.
// Missing rows from `lo` and `hi` are removed,
// original indices are in `loInd` and `hiInd`.
void split(const Vi& P, int k, Vi& lo, Vi& hi,
           Vi& loInd, Vi& hiInd) {
  int i = 0;
  each(e, P) {
    if (e < k) lo.pb(e), loInd.pb(i++);
    else hi.pb(e-k), hiInd.pb(i++);
  } // c3a6
} // 7bb7

// Map sub-permutation into sub-permutation
// of length `n` on given indices sets.
Vi expand(const Vi& P, Vi& ind1, Vi& ind2,
          int n, int def) {
  Vi ret(n, def);
  rep(k, 0, sz(P)) if (P[k] != def)
    ret[ind1[k]] = ind2[P[k]];
  return ret;
} // 7f23

// Compute (min, +) product of square
// simple unit-Monge matrices given their
// permutation representations; time: O(n lg n)
// Permutation of second matrix is inverted!
Vi comb(const Vi& P, const Vi& invQ) {
  int n = sz(P);

  if (n < 100) {
    // 5s -> 1s speedup for ALIS for n = 10^5
    Vi ret = invert(P, n, -1);
    rep(i, 0, sz(invQ)) {
      int from = invQ[i];
      rep(j, 0, i) from += invQ[j] > invQ[i];
      for (int j = from; j > i; j--)
        if (ret[j-1] < ret[j])
          swap(ret[j-1], ret[j]);
    } // 7cd1
    return invert(ret, n, -1);
  } // a886

  Vi p1, p2, q1, q2, i1, i2, j1, j2;
  split(P, n/2, p1, p2, i1, i2);
  split(invQ, n/2, q1, q2, j1, j2);

  p1 = expand(comb(p1, q1), i1, j1, n, -1);
  p2 = expand(comb(p2, q2), i2, j2, n, n);
  q1 = invert(p1, n, -1);
  q2 = invert(p2, n, n);

  Vi ans(n, -1);
  int delta = 0, j = n;

  rep(i, 0, n) {
    ans[i] = (p1[i] < 0 ? p2[i] : p1[i]);
    while (j > 0 && delta >= 0)
      delta -= (q2[--j] < i || q1[j] >= i);

    if (p2[i] < j || p1[i] >= j)
      if (delta++ < 0)
        if (q2[j] < i || q1[j] >= i)
          ans[i] = j;
  } // c396

  return ans;
} // cbff

// Helper function for `mongeMul`.
void padPerm(const Vi& P, Vi& has, Vi& pad,
             Vi& ind, int n) {
  vector<bool> seen(n);
  rep(i, 0, sz(P)) if (P[i] != -1) {
    ind.pb(i);
    has.pb(P[i]);
    seen[P[i]] = 1;
  } // 157e
  rep(i, 0, n) if (!seen[i]) pad.pb(i);
} // 103b

// Compute (min, +) product of
// simple sub-unit-Monge matrices given their
// permutation representations; time: O(n lg n)
// Left matrix has size sz(P) x sz(Q).
// Right matrix has size sz(Q) x n.
// Output matrix has size sz(P) x n.
// NON-SQUARE MATRICES ARE NOT TESTED!
Vi mongeMul(const Vi& P, const Vi& Q, int n) {
  Vi h1, p1, i1, h2, p2, i2;
  padPerm(P, h1, p1, i1, sz(Q));
  padPerm(invert(Q, n, -1), h2, p2, i2, sz(Q));

  h1.insert(h1.begin(), all(p1));
  h2.insert(h2.end(), all(p2));
  Vi ans(sz(P), -1), tmp = comb(h1, h2);

  rep(i, 0, sz(i1)) {
    int j = tmp[i+sz(p1)];
    if (j < sz(i2)) {
      ans[i1[i]] = i2[j];
    } // 4d16
  } // c8a0
  return ans;
} // fda1

// Range Longest Increasing Subsequence Query;
// preprocessing: O(n lg^2 n), query: O(lg n)
#include "../structures/wavelet_tree.h"
struct ALIS {
  WaveletTree tree;

  ALIS() {}

  // Precompute data structure; O(n lg^2 n)
  ALIS(const Vi& seq) {
    Vi P = build(seq);
    each(k, P) if (k == -1) k = sz(seq);
    tree = {P, sz(seq)+1};
  } // aaed

  // Query LIS of s[b;e); time: O(lg n)
  int operator()(int b, int e) {
    return e - b -
      tree.count(b, sz(tree.seq[1]), 0, e);
  } // fb4a

  Vi build(const Vi& seq) {
    int n = sz(seq);
    if (!n) return {};
    int lo = *min_element(all(seq));
    int hi = *max_element(all(seq));

    if (lo == hi) {
      Vi tmp(n);
      iota(all(tmp), 1);
      tmp.back() = -1;
      return tmp;
    } // 16d1

    int mid = (lo+hi+1) / 2;
    Vi p1, p2, i1, i2;
    split(seq, mid, p1, p2, i1, i2);

    p1 = expand(build(p1), i1, i1, n, -1);
    p2 = expand(build(p2), i2, i2, n, -1);
    each(j, i1) p2[j] = j;
    each(j, i2) p1[j] = j;
    return mongeMul(p1, p2, n);
  } // fe71
}; // 5740
```

text/palindromic_tree.h 1207

```
constexpr int ALPHA = 26; // Set alphabet size

// Tree of all palindromes in string,
// constructed online by appending letters.
// space: O(n*ALPHA); time: O(n)
// Code marked with [EXT] is extension for
// calculating minimal palindrome partition
// in O(n lg n). Can also be modified for
// similar dynamic programmings.
struct PalTree {
  Vi txt; // Text for which tree is built

  // Node 0 = empty palindrome (root of even)
  // Node 1 = "-1" palindrome (root of odd)
  Vi len{0, -1}; // Lengths of palindromes
  Vi link{1, 0}; // Suffix palindrome links
  // Edges to next palindromes
  vector<array<int, ALPHA>> to{ {}, {} };
  int last{0}; // Current node (max suffix pal)

  Vi diff{0, 0};   // len[i]-len[link[i]] [EXT]
  Vi slink{0, 0};  // Serial links        [EXT]
  Vi series{0, 0}; // Series DP answer    [EXT]
  Vi ans{0};       // DP answer for prefix[EXT]

  int ext(int i) {
    while (len[i]+2 > sz(txt) ||
           txt[sz(txt)-len[i]-2] != txt.back())
      i = link[i];
    return i;
  } // d442

  // Append letter from [0;ALPHA); time: O(1)
```

```
  // (or O(lg n) if [EXT] is enabled)
  void add(int x) {
    txt.pb(x);
    last = ext(last);

    if (!to[last][x]) {
      len.pb(len[last]+2);
      link.pb(to[ext(link[last])][x]);
      to[last][x] = sz(to);
      to.pb({});

      // [EXT]
      diff.pb(len.back() - len[link.back()]);
      slink.pb(diff.back() == diff[link.back()]
        ? slink[link.back()] : link.back());
      series.pb(0);
      // [/EXT]
    } // 8c1b
    last = to[last][x];

    // [EXT]
    ans.pb(INT_MAX);
    for (int i=last; len[i] > 0; i=slink[i]) {
      series[i] = ans[sz(ans) - len[slink[i]]
                  - diff[i] - 1];
      if (diff[i] == diff[link[i]])
        series[i] = min(series[i],
                        series[link[i]]);
      // If you want only even palindromes
      // set ans only for sz(txt)%2 == 0
      ans.back() = min(ans.back(),series[i]+1);
    } // ab3b
    // [/EXT]
  } // 66d3
}; // 595d
```

text/suffix_array_linear.h 1341

```
#include "../util/radix_sort.h"

// KS algorithm for suffix array; time: O(n)
// Input values are assumed to be in [1;k]
Vi sufArray(Vi str, int k) {
  int n = sz(str);
  Vi suf(n);
  str.resize(n+15);

  if (n < 15) {
    iota(all(suf), 0);
    rep(j, 0, n) countSort(suf,
      [&](int i) { return str[i+n-j-1]; }, k);
    return suf;
  } // 5fcf

  // Compute triples codes
  Vi tmp, code(n+2);
  rep(i, 0, n) if (i % 3) tmp.pb(i);

  rep(j, 0, 3) countSort(tmp,
    [&](int i) { return str[i-j+2]; }, k);

  int mc = 0, j = -1;

  each(i, tmp) {
    code[i] = mc += (j == -1 ||
        str[i] != str[j] ||
        str[i+1] != str[j+1] ||
        str[i+2] != str[j+2]);
    j = i;
  } // bfdc

  // Compute suffix array of 2/3
  tmp.clear();
  for (int i=1; i < n; i += 3) tmp.pb(code[i]);
  tmp.pb(0);
  for (int i=2; i < n; i += 3) tmp.pb(code[i]);
  tmp = sufArray(move(tmp), mc);

  // Compute partial suffix arrays
  Vi third;
  int th = (n+4) / 3;
  if (n%3 == 1) third.pb(n-1);

  rep(i, 1, sz(tmp)) {
    int e = tmp[i];
    tmp[i-1] = (e < th ? e*3+1 : (e-th)*3+2);
    code[tmp[i-1]] = i;
    if (e < th) third.pb(e*3);
  } // f9f1

  tmp.pop_back();
  countSort(third,
    [&](int i) { return str[i]; }, k);

  // Merge suffix arrays
  merge(all(third), all(tmp), suf.begin(),
    [&](int l, int r) {
      while (l%3 == 0 || r%3 == 0) {
        if (str[l] != str[r])
          return str[l] < str[r];
        l++; r++;
      } // 2f8a
      return code[l] < code[r];
    }); // 4cb3

  return suf;
} // 9165

// KS algorithm for suffix array; time: O(n)
Vi sufArray(const string& str) {
  return sufArray(Vi(all(str)), 255);
} // 593f
```

text/suffix_automaton.h d00d

```
constexpr char AMIN = 'a'; // Smallest letter
constexpr int ALPHA = 26; // Set alphabet size

// Suffix automaton - minimal DFA that
// recognizes all suffixes of given string
// (and encodes all substrings);
// space: O(n*ALPHA); time: O(n)
// Paths from root are equivalent to substrings
// Extensions:
// - [OCC] - count occurences of substrings
// - [PATHS] - count paths from node
struct SufDFA {
  // State v represents endpos-equivalence
  // class that contains words of all lengths
  // between link[len[v]]+1 and len[v].
  // len[v] = longest word of equivalence class
  // link[v] = link to state of longest suffix
  //           in other equivalence class
  // to[v][c] = automaton edge c from v
  Vi len{0}, link{-1};
  vector<array<int, ALPHA>> to{ {} };
  int last{0}; // Current node (whole word)

  vector<Vi> inSufs; // [OCC] Suffix-link tree
  Vi cnt{0};         // [OCC] Occurence count
  vector<ll> paths;  // [PATHS] Out-path count

  SufDFA() {}

  // Build suffix automaton for given string
  // and compute extended stuff; time: O(n)
  SufDFA(const string& s) {
    each(c, s) add(c);
    finish();
  } // ec2e

  // Append letter to the back
  void add(char c) {
    int v = last, x = c-AMIN;
    last = sz(len);
    len.pb(len[v]+1);
    link.pb(0);
    to.pb({});
    cnt.pb(1); // [OCC]

    while (v != -1 && !to[v][x]) {
      to[v][x] = last;
      v = link[v];
    } // 4cfc

    if (v != -1) {
      int q = to[v][x];
      if (len[v]+1 == len[q]) {
        link[last] = q;
      } else {
        len.pb(len[v]+1);
        link.pb(link[q]);
        to.pb(to[q]);
        cnt.pb(0); // [OCC]
        link[last] = link[q] = sz(len)-1;
        while (v != -1 && to[v][x] == q) {
          to[v][x] = link[q];
          v = link[v];
        } // 784f
      } // 90aa
    } // af69
  } // 345a

  // Compute some additional stuff (offline)
  void finish() {
    // [OCC]
    inSufs.resize(sz(len));
    rep(i, 1, sz(link)) inSufs[link[i]].pb(i);
    dfsSufs(0);
    // [/OCC]

    // [PATHS]
    paths.assign(sz(len), 0);
    dfs(0);
    // [/PATHS]
  } // 3f75

  // Only for [OCC]
  void dfsSufs(int v) {
    each(e, inSufs[v]) {
      dfsSufs(e);
      cnt[v] += cnt[e];
    } // 2469
  } // 0c60

  // Only for [PATHS]
  void dfs(int v) {
    if (paths[v]) return;
    paths[v] = 1;
    each(e, to[v]) if (e) {
      dfs(e);
      paths[v] += paths[e];
    } // 22b3
  } // d004

  // Go using edge `c` from state `i`.
  // Returns 0 if edge doesn't exist.
  int next(int i, char c) {
    return to[i][c-AMIN];
  } // c363

  // Get lexicographically k-th substring
  // of represented string; time: O(|substr|)
  // Empty string has index 0.
  // Requires [PATHS] extension.
  string lex(ll k) {
    string s;
    int v = 0;
    while (k--) rep(i, 0, ALPHA) {
      int e = to[v][i];
      if (e) {
        if (k < paths[e]) {
          s.pb(char(AMIN+i));
          v = e;
          break;
        } // f307
        k -= paths[e];
      } // 29be
    } // 4600
    return s;
  } // e4af
}; // 7135
```

text/suffix_tree.h 40a4

```
constexpr int ALPHA = 26;

// Ukkonen's algorithm for online suffix tree
// construction; space: O(n*ALPHA); time: O(n)
// Real tree nodes are called dedicated nodes.
// "Nodes" lying on compressed edges are called
// implicit nodes and are represented
// as pairs (lower node, label index).
// Labels are represented as intervals [L;R)
// which refer to substrings [L;R) of txt.
// Leaves have labels of form [L;infinity),
// use getR to get current right endpoint.
// Suffix links are valid only for internal
// nodes (non-leaves).
struct SufTree {
  Vi txt; // Text for which tree is built
  // to[v][c] = edge with label starting with c
  //            from node v
  vector<array<int, ALPHA>> to{ {} };
  Vi L{0}, R{0}; // Parent edge label endpoints
  Vi par{0};     // Parent link
  Vi link{0};    // Suffix link
  Pii cur{0, 0}; // Current state

  // Get current right end of node label
  int getR(int i) { return min(R[i],sz(txt)); }

  // Follow edge `e` of implicit node `s`.
  // Returns (-1, -1) if there is no edge.
  Pii next(Pii s, int e) {
    if (s.y < getR(s.x))
      return txt[s.y] == e ? mp(s.x, s.y+1)
                           : mp(-1, -1);
    e = to[s.x][e];
    return e ? mp(e, L[e]+1) : mp(-1, -1);
  } // f430

  // Create dedicated node for implicit node
  // and all its suffixes
  int split(Pii s) {
    if (s.y == R[s.x]) return s.x;

    int t = sz(to); to.pb({});
    to[t][txt[s.y]] = s.x;
    L.pb(L[s.x]);
    R.pb(L[s.x] = s.y);
    par.pb(par[s.x]);
    par[s.x] = to[par[t]][txt[L[t]]] = t;
```

```
    link.pb(-1);

    int v = link[par[t]], l = L[t] + !par[t];
    while (l < R[t]) {
      v = to[v][txt[l]];
      l += getR(v) - L[v];
    } // 0393

    v = split({v, getR(v)-l+R[t]});
    link[t] = v;
    return t;
  } // 10bb

  // Append letter from [0;ALPHA) to the back
  void add(int x) { // amortized time: O(1)
    Pii t; txt.pb(x);
    while ((t = next(cur, x)).x == -1) {
      int m = split(cur);
      to[m][x] = sz(to);
      to.pb({});
      par.pb(m);
      L.pb(sz(txt)-1);
      R.pb(INT_MAX);
      link.pb(-1);
      cur = {link[m], getR(link[m])};
      if (!m) return;
    } // 60c2
    cur = t;
  } // 6f4e
}; // dbfb
```

text/z_function.h 0466

```
// Computes Z function array; time: O(n)
// zf[i] = max common prefix of str and str[i:]
template<class T> Vi prefPref(const T& str) {
  int n = sz(str), b = 0, e = 1;
  Vi zf(n);
  rep(i, 1, n) {
    if (i < e) zf[i] = min(zf[i-b], e-i);
    while (i+zf[i] < n &&
      str[zf[i]] == str[i+zf[i]]) zf[i]++;
    if (i+zf[i] > e) b = i, e = i+zf[i];
  } // e906
  zf[0] = n;
  return zf;
} // b88d
```

trees/centroid_decomp.h 5247

```
// Centroid decomposition; space: O(n lg n)
struct CentroidTree {
  // child[v] = children of v in centroid tree
  // par[v] = parent of v in centroid tree
  //          (-1 for root)
  // depth[v] = depth of v in centroid tree
  //            (0 for root)
  // ind[v][i] = index of vertex v in i-th
  //             centroid subtree from root
  // size[v] = size of centroid subtree of v
  // subtree[v] = list of vertices
  //              in centroid subtree of v
  // dists[v] = distances from v to vertices
  //            in its centroid subtree
  //            (in the order of subtree[v])
  // neigh[v] = neighbours of v
  //            in its centroid subtree
  // dir[v][i] = index of centroid neighbour
  //             that is first vertex on path
  //             from centroid v to i-th vertex
  //             of centroid subtree
  //             (-1 for centroid)
  vector<Vi> child, ind, dists, subtree,
             neigh, dir;
  Vi par, depth, size;
  int root; // Root centroid

  CentroidTree() {}

  CentroidTree(vector<Vi>& G)
      : child(sz(G)), ind(sz(G)), dists(sz(G)),
        subtree(sz(G)), neigh(sz(G)),
        dir(sz(G)), par(sz(G), -2),
        depth(sz(G)), size(sz(G)) {
    root = decomp(G, 0, 0);
  } // 026c

  void dfs(vector<Vi>& G, int v, int p) {
    size[v] = 1;
    each(e, G[v]) if (e != p && par[e] == -2)
      dfs(G, e, v), size[v] += size[e];
  } // bbed

  void layer(vector<Vi>& G, int v,
             int p, int c, int d) {
    ind[v].pb(sz(subtree[c]));
    subtree[c].pb(v);
    dists[c].pb(d);
    dir[c].pb(sz(neigh[c])-1);
    each(e, G[v]) if (e != p && par[e] == -2) {
      if (v == c) neigh[c].pb(e);
      layer(G, e, v, c, d+1);
    } // dc82
  } // 37ee

  int decomp(vector<Vi>& G, int v, int d) {
    dfs(G, v, -1);
    int p = -1, s = size[v];
  loop:
    each(e, G[v]) {
      if (e != p && par[e] == -2 &&
          size[e] > s/2) {
        p = v; v = e; goto loop;
      } // e0a5
    } // 3533

    par[v] = -1;
    size[v] = s;
    depth[v] = d;
    layer(G, v, -1, v, 0);

    each(e, G[v]) if (par[e] == -2) {
      int j = decomp(G, e, d+1);
      child[v].pb(j);
      par[j] = v;
    } // 70b5
    return v;
  } // 217c
}; // 71d6
```

trees/centroid_offline.h ac92

```
// Helper for offline centroid decomposition
// Usage: CentroidDecomp(G);
// Constructor calls method `process`
// for each centroid subtree.
struct CentroidDecomp {
  vector<Vi>& G; // Reference to target graph
  vector<bool> on; // Is vertex enabled?
  Vi size; // Used internally

  // Run centroid decomposition for graph g
  CentroidDecomp(vector<Vi>& g)
      : G(g), on(sz(g), 1), size(sz(g)) {
    decomp(0);
  } // 8677

  // Compute subtree sizes for subtree rooted
  // at v, ignoring p and disabled vertices
  void computeSize(int v, int p) {
    size[v] = 1;
    each(e, G[v]) if (e != p && on[e])
      computeSize(e, v), size[v] += size[e];
  } // 1c0d

  void decomp(int v) {
    computeSize(v, -1);
    int p = -1, s = size[v];
  loop:
    each(e, G[v]) {
      if (e != p && on[e] && size[e] > s/2) {
        p = v; v = e; goto loop;
      } // e0a5
    } // f31d
    process(v);
    on[v] = 0;
    each(e, G[v]) if (on[e]) decomp(e);
  } // f170

  // Process current centroid subtree:
  // - v is centroid
  // - boundary vertices have on[x] = 0
  // Formally: Let H be subgraph induced
  // on vertices such that on[v] = 1.
  // Then current centroid subtree is
  // connected component of H that contains v
  // and v is its centroid.
  void process(int v) {
    // Do your stuff here...
  } // d41d
}; // a923
```

trees/heavylight_decomp.h ebb8

```
#include "../structures/segtree_point.h"

// Heavy-Light Decomposition of tree
// with subtree query support; space: O(n)
struct HLD {
  // Subtree of v = [pos[v]; pos[v]+size[v])
  // Chain with v = [chBegin[v]; chEnd[v])
  Vi par;     // Vertex parent
  Vi size;    // Vertex subtree size
  Vi depth;   // Vertex distance to root
  Vi pos;     // Vertex position in "HLD" order
  Vi chBegin; // Begin of chain with vertex
  Vi chEnd;   // End of chain with vertex
  Vi order;   // "HLD" preorder of vertices
  SegTree tree; // Verts are in HLD order

  HLD() {}

  // Initialize structure for tree G
  // and given root; time: O(n lg n)
  // MODIFIES ORDER OF EDGES IN G!
  HLD(vector<Vi>& G, int root)
      : par(sz(G)), size(sz(G)),
        depth(sz(G)), pos(sz(G)),
        chBegin(sz(G)), chEnd(sz(G)) {
    dfs(G, root, -1);
    decomp(G, root, -1, 0);
    tree = {sz(order)};
  } // 8263

  void dfs(vector<Vi>& G, int v, int p) {
    par[v] = p;
    size[v] = 1;
    depth[v] = p < 0 ? 0 : depth[p]+1;

    if (G[v].empty()) return;
    int& fs = G[v][0];
    if (fs == p) swap(fs, G[v].back());

    each(e, G[v]) if (e != p) {
      dfs(G, e, v);
      size[v] += size[e];
      if (size[e] > size[fs]) swap(e, fs);
    } // 9872
  } // e25f

  void decomp(vector<Vi>& G,
              int v, int p, int chb) {
    pos[v] = sz(order);
    chBegin[v] = chb;
    chEnd[v] = pos[v]+1;
    order.pb(v);

    each(e, G[v]) if (e != p) {
      if (e == G[v][0]) {
        decomp(G, e, v, chb);
        chEnd[v] = chEnd[e];
      } else {
        decomp(G, e, v, sz(order));
      } // c84a
    } // f707
  } // eb89

  // Get root of chain containing v
  int chRoot(int v) {return order[chBegin[v]];}

  // Level Ancestor Query; time: O(lg n)
  int laq(int v, int level) {
    while (true) {
      int k = pos[v] - depth[v] + level;
      if (k >= chBegin[v]) return order[k];
      v = par[chRoot(v)];
    } // 8c18
  } // 675e

  // Lowest Common Ancestor; time: O(lg n)
  int lca(int a, int b) {
    while (chBegin[a] != chBegin[b]) {
      int ha = chRoot(a), hb = chRoot(b);
      if (depth[ha] > depth[hb]) a = par[ha];
      else b = par[hb];
    } // 5620
    return depth[a] < depth[b] ? a : b;
  } // c168

  // Call func(chBegin, chEnd) on each path
  // segment; time: O(lg n * time of func)
  template<class T>
  void iterPath(int a, int b, T func) {
    while (chBegin[a] != chBegin[b]) {
      int ha = chRoot(a), hb = chRoot(b);
      if (depth[ha] > depth[hb]) {
        func(chBegin[a], pos[a]+1);
        a = par[ha];
      } else {
        func(chBegin[b], pos[b]+1);
        b = par[hb];
      } // f9a5
    } // 563c

    if (pos[a] > pos[b]) swap(a, b);
    // Remove +1 from pos[a]+1 for vertices
    // queries (with +1 -> edges).
    func(pos[a]+1, pos[b]+1);
  } // 17e5

  // Query path between a and b; O(lg^2 n)
```

```
  SegTree::T queryPath(int a, int b) {
    auto ret = tree.ID;
    iterPath(a, b, [&](int i, int j) {
      ret = tree.f(ret, tree.query(i, j));
    }); // 1113
    return ret;
  } // 1bc9

  // Query subtree of v; time: O(lg n)
  SegTree::T querySubtree(int v) {
    return tree.query(pos[v], pos[v]+size[v]);
  } // 23db
}; // 59bf
```

trees/lca.h 294f

```
// LAQ and LCA using jump pointers
// space: O(n lg n)

struct LCA {
  vector<Vi> jumps;
  Vi level, pre, post;
  int cnt{0}, depth;

  LCA() {}

  // Initialize structure for tree G
  // and root r; time: O(n lg n)
  LCA(vector<Vi>& G, int root)
      : jumps(sz(G)), level(sz(G)),
        pre(sz(G)), post(sz(G)) {
    dfs(G, root, root);
    depth = int(log2(sz(G))) + 2;
    rep(j, 0, depth) each(v, jumps)
      v.pb(jumps[v[j]][j]);
  } // d6ce

  void dfs(vector<Vi>& G, int v, int p) {
    level[v] = p == v ? 0 : level[p]+1;
    jumps[v].pb(p);
    pre[v] = ++cnt;
    each(e, G[v]) if (e != p) dfs(G, e, v);
    post[v] = ++cnt;
  } // e286

  // Check if a is ancestor of b; time: O(1)
  bool isAncestor(int a, int b) {
    return pre[a] <= pre[b] &&
           post[b] <= post[a];
  } // 5514

  // Lowest Common Ancestor; time: O(lg n)
  int operator()(int a, int b) {
    for (int j = depth; j--;)
      if (!isAncestor(jumps[a][j], b))
        a = jumps[a][j];
    return isAncestor(a, b) ? a : jumps[a][0];
  } // 27d8

  // Level Ancestor Query; time: O(lg n)
  int laq(int a, int lvl) {
    for (int j = depth; j--;)
      if (lvl <= level[jumps[a][j]])
        a = jumps[a][j];
    return a;
  } // 75b3

  // Get distance from a to b; time: O(lg n)
  int distance(int a, int b) {
    return level[a] + level[b] -
        level[operator()(a, b)]*2;
  } // 07e0

  // Get k-th vertex on path from a to b,
  // a is 0, b is last; time: O(lg n)
  // Returns -1 if k > distance(a, b)
  int kthVertex(int a, int b, int k) {
    int c = operator()(a, b);
    if (level[a]-k >= level[c])
      return laq(a, level[a]-k);
    k += level[c]*2 - level[a];
    return (k > level[b] ? -1 : laq(b, k));
  } // 46c9
}; // 2254
```

trees/lca_linear.h 7aa5

```
// LAQ and LCA using jump pointers
// with linear memory; space: O(n)
struct LCA {
  Vi par, jmp, depth, pre, post;
  int cnt{0};

  LCA() {}

  // Initialize structure for tree G
  // and root v; time: O(n lg n)
  LCA(vector<Vi>& G, int v)
      : par(sz(G), -1), jmp(sz(G), v),
        depth(sz(G)), pre(sz(G)), post(sz(G)) {
    dfs(G, v);
  } // 94cf

  void dfs(vector<Vi>& G, int v) {
    int j = jmp[v], k = jmp[j], x =
      depth[v]+depth[k] == depth[j]*2 ? k : v;
    pre[v] = ++cnt;
    each(e, G[v]) if (!pre[e]) {
      par[e] = v; jmp[e] = x;
      depth[e] = depth[v]+1;
      dfs(G, e);
    } // b123
    post[v] = ++cnt;
  } // 3280

  // Level Ancestor Query; time: O(lg n)
  int laq(int v, int d) {
    while (depth[v] > d)
      v = depth[jmp[v]] < d ? par[v] : jmp[v];
    return v;
  } // f509

  // Lowest Common Ancestor; time: O(lg n)
  int operator()(int a, int b) {
    if (depth[a] > depth[b]) swap(a, b);
    b = laq(b, depth[a]);
    while (a != b) {
      if (jmp[a] == jmp[b])
        a = par[a], b = par[b];
      else
        a = jmp[a], b = jmp[b];
    } // fe08
    return a;
  } // 25ff

  // Check if a is ancestor of b; time: O(1)
  bool isAncestor(int a, int b) {
    return pre[a] <= pre[b] &&
           post[b] <= post[a];
  } // 5514

  // Get distance from a to b; time: O(lg n)
  int distance(int a, int b) {
    return depth[a] + depth[b] -
        depth[operator()(a, b)]*2;
  } // a340

  // Get k-th vertex on path from a to b,
  // a is 0, b is last; time: O(lg n)
  // Returns -1 if k > distance(a, b)
  int kthVertex(int a, int b, int k) {
    int c = operator()(a, b);
    if (depth[a]-k >= depth[c])
      return laq(a, depth[a]-k);
    k += depth[c]*2 - depth[a];
    return (k > depth[b] ? -1 : laq(b, k));
  } // 34ed
}; // a221
```

trees/link_cut_tree.h 6bd6

```
constexpr int INF = 1e9;

// Link/cut tree; space: O(n)
// Represents forest of (un)rooted trees.
struct LinkCutTree {
  vector<array<int, 2>> child;
  Vi par, prev, flip, size;

  // Initialize structure for n vertices; O(n)
  // At first there's no edges.
  LinkCutTree(int n = 0)
      : child(n, {-1, -1}), par(n, -1),
        prev(n, -1), flip(n, -1), size(n, 1) {}

  void push(int x) {
    if (x >= 0 && flip[x]) {
      flip[x] = 0;
      swap(child[x][0], child[x][1]);
      each(e, child[x]) if (e>=0) flip[e] ^= 1;
    } // + any other lazy path operations
  } // bae2

  void update(int x) {
    if (x >= 0) {
      size[x] = 1;
      each(e, child[x]) if (e >= 0)
        size[x] += size[e];
    } // + any other path aggregates
  } // 8ec0

  void auxLink(int p, int i, int ch) {
    child[p][i] = ch;
    if (ch >= 0) par[ch] = p;
    update(p);
  } // 0a9a

  void rot(int p, int i) {
    int x = child[p][i], g = par[x] = par[p];
    if (g >= 0) child[g][child[g][1] == p] = x;
    auxLink(p, i, child[x][!i]);
    auxLink(x, !i, p);
    swap(prev[x], prev[p]);
    update(g);
  } // 4c76

  void splay(int x) {
    while (par[x] >= 0) {
      int p = par[x], g = par[p];
      push(g); push(p); push(x);
      bool f = (child[p][1] == x);
      if (g >= 0) {
        if (child[g][f] == p) { // zig-zig
          rot(g, f); rot(p, f);
        } else { // zig-zag
          rot(p, f); rot(g, !f);
        } // 2ebb
      } else { // zig
        rot(p, f);
      } // f8a2
    } // 446b
    push(x);
  } // 55a7

  // After this operation x becomes the end
  // of preferred path starting in root;
  void access(int x) { // amortized O(lg n)
    while (true) {
      splay(x);
      int p = prev[x];
      if (p < 0) break;

      prev[x] = -1;
      splay(p);

      int r = child[p][1];
      if (r >= 0) swap(par[r], prev[r]);
      auxLink(p, 1, x);
    } // 2b87
  } // 30eb

  // Make x root of its tree; amortized O(lg n)
  void makeRoot(int x) {
    access(x);
    int& l = child[x][0];
    if (l >= 0) {
      swap(par[l], prev[l]);
      flip[l] ^= 1;
      update(l);
      l = -1;
      update(x);
    } // 0064
  } // b246

  // Find root of tree containing x
  int find(int x) { // time: amortized O(lg n)
    access(x);
    while (child[x][0] >= 0)
      push(x = child[x][0]);
    splay(x);
    return x;
  } // d78d

  // Add edge x-y; time: amortized O(lg n)
  // Root of tree containing y becomes
  // root of new tree.
  void link(int x, int y) {
    makeRoot(x); prev[x] = y;
  } // fb4f

  // Remove edge x-y; time: amortized O(lg n)
  // x and y become roots of new trees!
  void cut(int x, int y) {
    makeRoot(x); access(y);
    par[x] = child[y][0] = -1;
    update(y);
  } // 1908

  // Get distance between x and y,
  // returns INF if x and y there's no path.
  // This operation makes x root of the tree!
  int dist(int x, int y) { // amortized O(lg n)
    makeRoot(x);
    if (find(y) != x) return INF;
    access(y);
    int t = child[y][0];
    return t >= 0 ? size[t] : 0;
  } // ae69
}; // 4480
```

util/arc_interval_cover.h 7507

```
using dbl = double;
```

```
// Find size of smallest set of points
// such that each arc contains at least one
// of them; time: O(n lg n)
int arcCover(vector<pair<dbl, dbl>>& inters,
             dbl wrap) {
  int n = sz(inters);

  rep(i, 0, n) {
    auto& e = inters[i];
    e.x = fmod(e.x, wrap);
    e.y = fmod(e.y, wrap);
    if (e.x < 0) e.x += wrap, e.y += wrap;
    if (e.x > e.y) e.x += wrap;
    inters.pb({e.x+wrap, e.y+wrap});
  } // b87d

  Vi nxt(n);
  deque<dbl> que;
  dbl r = wrap*4;
  sort(all(inters));

  for (int i = n*2-1; i--;) {
    r = min(r, inters[i].y);
    que.push_front(inters[i].x);
    while (!que.empty() && que.back() > r)
      que.pop_back();
    if (i < n) nxt[i] = i+sz(que);
  } // 5e6c

  int a = 0, b = 0;
  do {
    a = nxt[a] % n;
    b = nxt[nxt[b]%n] % n;
  } while (a != b);

  int ans = 0;
  while (b < a+n) {
    b += nxt[b%n] - b%n;
    ans++;
  } // 7350
  return ans;
} // 7871
```

util/bit_hacks.h 599a

```
// __builtin_popcount - count number of 1 bits
// __builtin_clz - count most significant 0s
// __builtin_ctz - count least significant 0s
// __builtin_ffs - like ctz, but indexed from 1
//                 returns 0 for 0
// For ll version add ll to name

using ull = uint64_t;

#define T64(s,up)                            \
  for (ull i=0; i<64; i+=s*2)                \
    for (ull j = i; j < i+s; j++) {          \
      ull a = (M[j] >> s) & up;              \
      ull b = (M[j+s] & up) << s;            \
      M[j] = (M[j] & up) | b;                \
      M[j+s] = (M[j+s] & (up<<s)) | a;       \
    } // a290

// Transpose 64x64 bit matrix
void transpose64(array<ull, 64>& M) {
  T64(1,  0x5555555555555555);
  T64(2,  0x3333333333333333);
  T64(4,  0xF0F0F0F0F0F0F0F);
  T64(8,  0xFF00FF00FF00FF);
  T64(16, 0xFFFF0000FFFF);
  T64(32, 0xFFFFFFFFLL);
} // 6889

// Lexicographically next mask with same
// amount of ones.
int nextSubset(int v) {
  int t = v | (v - 1);
  return (t + 1) | (((~t & -~t) - 1) >>
      (__builtin_ctz(v) + 1));
} // 4c0c
```

util/bump_alloc.h 09f9

```
// Allocator, which doesn't free memory.

char mem[400<<20]; // Set memory limit
size_t nMem;

void* operator new(size_t n) {
  nMem += n; return &mem[nMem-n];
} // fba6
void operator delete(void*) {}
```

util/compress_vec.h bc5d

```
// Compress integers to range [0;n) while
// preserving their order; time: O(n lg n)
// Returns mapping: compressed -> original
Vi compressVec(vector<int*>& vec) {
  sort(all(vec),
    [](int* l, int* r) { return *l < *r; });
  Vi old;
  each(e, vec) {
    if (old.empty() || old.back() != *e)
      old.pb(*e);
    *e = sz(old)-1;
  } // 7eb0
  return old;
} // 2b60
```

util/deque_undo.h 404d

```
// Deque-like undoing on data structures with
// amortized O(log n) overhead for operations.
// Maintains a deque of objects alongside
// a data structure that contains all of them.
// The data structure only needs to support
// insertions and undoing of last insertion
// using the following interface:
// - insert(...) - insert an object to DS
// - time() - returns current version number
// - rollback(t) - undo all operations after t
// Assumes time() == 0 for empty DS.
struct DequeUndo {
  // Argument for insert(...) method of DS.
  using T = tuple<int, int>;
  DataStructure ds; // Configure DS type here.
  vector<T> elems[2];
  vector<Pii> his = {{0,0}};

  // Push object to front or back of deque,
  // depending on side parameter.
  void push(T val, bool side) {
    elems[side].pb(val);
    doPush(0, side);
  } // df9f

  // Pop object from front or back of deque,
  // depending on side parameter.
  void pop(int side) {
    auto &A = elems[side], &B = elems[!side];
    int cnt[2] = {};

    if (A.empty()) {
      assert(!B.empty());
      auto it = B.begin() + sz(B)/2 + 1;
      A.assign(B.begin(), it);
      B.erase(B.begin(), it);
      reverse(all(A));
      his.resize(1);
      cnt[0] = sz(A);
      cnt[1] = sz(B);
    } else {
      do {
        cnt[his.back().y ^ side]++;
        his.pop_back();
      } while (cnt[0]*2 < cnt[1] &&
               cnt[0] < sz(A));
    } // b4ef

    cnt[0]--;
    A.pop_back();
    ds.rollback(his.back().x);
    for (int i : {1, 0})
      while (cnt[i]) doPush(--cnt[i], i^side);
  } // 6eba

  void doPush(int i, bool s) {
    apply([&](auto... x) { ds.insert(x...); },
      elems[s].rbegin()[i]);
    his.pb({ds.time(), s});
  } // 4fed
}; // 5f3d
```

util/inversion_vector.h 01f9

```
// Get inversion vector for sequence of
// numbers in [0;n); ret[i] = count of numbers
// greater than perm[i] to the left; O(n lg n)
Vi encodeInversions(Vi perm) {
  Vi odd, ret(sz(perm));
  int cont = 1;

  while (cont) {
    odd.assign(sz(perm)+1, 0);
    cont = 0;

    rep(i, 0, sz(perm)) {
      if (perm[i] % 2) odd[perm[i]]++;
      else ret[i] += odd[perm[i]+1];
      cont += perm[i] /= 2;
    } // 4ed0
  } // a4f0
  return ret;
} // c2e1

// Count inversions in sequence of numbers
// in [0;n); time: O(n lg n)
ll countInversions(Vi perm) {
  ll ret = 0, cont = 1;
  Vi odd;

  while (cont) {
    odd.assign(sz(perm)+1, 0);
    cont = 0;

    rep(i, 0, sz(perm)) {
      if (perm[i] % 2) odd[perm[i]]++;
      else ret += odd[perm[i]+1];
      cont += perm[i] /= 2;
    } // 916f
  } // c9b5
  return ret;
} // 1aaf
```

util/longest_inc_subseq.h 0bd1

```
// Longest Increasing Subsequence; O(n lg n)
int lis(const Vi& seq) {
  Vi dp(sz(seq), INT_MAX);
  each(c, seq) *lower_bound(all(dp), c) = c;
  return int(lower_bound(all(dp), INT_MAX)
            - dp.begin());
} // d0e9
```

util/max_rects.h 2a16

```
struct MaxRect {
  // begin = first column of rectangle
  // end = first column after rectangle
  // hei = height of rectangle
  // touch = columns of height hei inside
  int begin, end, hei;
  Vi touch; // sorted increasing
}; // 41fe

// Given consecutive column heights find
// all inclusion-wise maximal rectangles
// contained in "drawing" of columns; time O(n)
vector<MaxRect> getMaxRects(Vi hei) {
  hei.insert(hei.begin(), -1);
  hei.pb(-1);
  Vi reach(sz(hei), sz(hei)-1);
  vector<MaxRect> ans;

  for (int i = sz(hei)-1; --i;) {
    int j = i+1, k = i;
    while (hei[j] > hei[i]) j = reach[j];
    reach[i] = j;

    while (hei[k] > hei[i-1]) {
      ans.pb({ i-1, 0, hei[k], {} });
      auto& rect = ans.back();

      while (hei[k] == rect.hei) {
        rect.touch.pb(k-1);
        k = reach[k];
      } // 6e7e
      rect.end = k-1;
    } // e03f
  } // 2796
  return ans;
} // 0e49
```

util/mo.h caeb

```
// Modified MO's queries sorting algorithm,
// slightly better results than standard.
// Allows to process q queries in O(n*sqrt(q))

struct Query {
  int begin, end;
}; // b76d

// Get point index on Hilbert curve
ll hilbert(int x, int y, int s, ll c = 0) {
  if (s <= 1) return c;
  s /= 2; c *= 4;
  if (y < s)
    return hilbert(x&(s-1), y, s, c+(x>=s)+1);
  if (x < s)
    return hilbert(2*s-y-1, s-x-1, s, c);
  return hilbert(y-s, x-s, s, c+3);
} // 0fb9

// Get good order of queries; time: O(n lg n)
Vi moOrder(vector<Query>& queries, int maxN) {
  int s = 1;
  while (s < maxN) s *= 2;

  vector<ll> ord;
  each(q, queries)
    ord.pb(hilbert(q.begin, q.end, s));

  Vi ret(sz(ord));
  iota(all(ret), 0);
  sort(all(ret), [&](int l, int r) {
```

```
    return ord[l] < ord[r];
  }); // 9aea
  return ret;
} // ecec
```

util/parallel_binsearch.h fdd4

```
// Run `n` binary searches on [b;e) parallely.
// `cmp` should be lambda with arguments:
// 1) vector<Pii>& - pairs (v, i)
//    which are queries if value for index i
//    is greater or equal to v;
//    pairs are sorted by v
// 2) vector<bool>& - output vector,
//    set true at index i if value
//    for i-th query is >= queried value
// Returns vector of found values;
// time: O((n+c) lg range), where c is cmp time
template<class T>
Vi multiBS(int b, int e, int n, T cmp) {
  if (b >= e) return Vi(n, b);
  vector<Pii> que(n), rng(n, {b, e});
  vector<bool> ans(n);

  rep(i, 0, n) que[i] = {(b+e)/2, i};

  for (int k = 32-__builtin_clz(e-b); k--;) {
    int last = 0, j = 0;
    cmp(que, ans);
    rep(i, 0, sz(que)) {
      Pii &q = que[i], &r = rng[q.y];
      if (q.x != last) last = q.x, j = i;
      (ans[i] ? r.x : r.y) = q.x;
      q.x = (r.x+r.y) / 2;
      if (!ans[i]) swap(que[i], que[j++]);
    } // 6c4e
  } // 622c

  Vi ret;
  each(p, rng) ret.pb(p.x);
  return ret;
} // 9e29
```

util/radix_sort.h 6fb4

```
// Stable countingsort; time: O(k+sz(vec))
// See example usage in radixSort for pairs.
template<class F>
void countSort(Vi& vec, F key, int k) {
  static Vi buf, cnt;
  vec.swap(buf);
  vec.resize(sz(buf));
  cnt.assign(k+1, 0);
  each(e, buf) cnt[key(e)]++;
  rep(i, 1, k+1) cnt[i] += cnt[i-1];
  for (int i = sz(vec)-1; i >= 0; i--)
    vec[--cnt[key(buf[i])]] = buf[i];
} // cc6f

// Compute order of elems, k is max key; O(n)
Vi radixSort(const vector<Pii>& elems, int k) {
  Vi order(sz(elems));
  iota(all(order), 0);
  countSort(order,
    [&](int i) { return elems[i].y; }, k);
  countSort(order,
    [&](int i) { return elems[i].x; }, k);
  return order;
} // e8f6
```

xyz/kactl.h b5ec

```
// --- POINT 3D
template<class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T a=0, T b=0, T c=0)
    : x(a), y(b), z(c) {}
  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); } // 6cad
  P operator-(R p) const {
    return P(x-p.x, y-p.y, z-p.z); } // 01e2
  P operator*(T d) const {
    return P(x*d, y*d, z*d); } // 071d
  P operator/(T d) const {
    return P(x/d, y/d, z/d); } // 40df
  T dot(R p) const {
    return x*p.x + y*p.y + z*p.z; } // 466c
  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);
  } // 923b
  T dist2() const { return x*x + y*y + z*z; }
  double dist() const {
    return sqrt((double)dist2()); } // de26
  //Azimuthal angle (longitude) to x-axis
  double phi() const { // in interval [-pi, pi]
    return atan2(y, x); }
  //Zenith angle (latitude) to the z-axis
  double theta() const { // in interval [0, pi]
    return atan2(sqrt(x*x+y*y),z); } // ed29
  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;
  } // bc6c
}; // ce68

// --- HULL 3D (requires POINT 3D) O(n^2)
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;
}; // 8ad1

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);
  }; // 51be
  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();
      } // 5d69
    } // eeb6
    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);
    } // 9578
  } // c66b
  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;
}; // 2116

// --- DELAUNAY (requires POINT 3D and HULL 3D)
// Calls trifun for every triangle
template<class P, class F> // O(n^2)
void delaunay(vector<P>& ps, F trifun) {
  if (sz(ps) == 3) { int d =
    ((ps[1]-ps[0]).cross(ps[2]-ps[0]) < 0);
    trifun(0,1+d,2-d); } // 1266
  vector<P3> p3;
  for (P p : ps)
    p3.emplace_back(p.x, p.y, p.len2());
  if (sz(ps) > 3) for(auto t:hull3d(p3))
    if ((p3[t.b]-p3[t.a]).
      cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) <0)
    trifun(t.a, t.c, t.b);
} // 95dc

// --- FAST DELAUNAY (requires our vec2)
// O(n log n)
#include "../geometry/vec2.h"
typedef vec2i P;
typedef struct Quad* Q;
// (can be ll if coords are < 2e4)
typedef __int128_t lll;
// not equal to any other point
P arb(LLONG_MAX,LLONG_MAX);

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

ll cross(P x, P a, P b) {
  return (a-x).cross(b-x);
} // 0647
bool circ(P p, P a, P b, P c) {
  lll p2 = p.len2(), A = a.len2()-p2,
      B = b.len2()-p2, C = c.len2()-p2;
  return cross(p,a,b)*C + cross(p,b,c)*A +
    cross(p,c,a)*B > 0;
} // fe20
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{
    new Quad{0}}}}; // 08c9
  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;
} // 8da8
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o);
  swap(a->o, b->o);
} // 6867
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
} // af0b

pair<Q,Q> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b =
      makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = cross(s[0], s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ?
      c : b->r() }; // d148
  } // d2fb

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

#define DEL(e, init, dir) Q e = init->dir; \
  if (valid(e)) \
    while (circ(e->dir->F(),H(base),e->F())) {\
      Q t = e->dir; \
      splice(e, e->prev()); \
      splice(e->r(), e->r()->prev()); \
      e->o = H; H = e; e = t; \
    } // a3f9
  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());
  } // 53db
  return { ra, rb };
} // 4baa
```

```
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 (cross(e->o->F(), e->F(), e->p)
    < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; \
    pts.push_back(c->p); \
    q.push_back(c->r()); c = c->next(); } \
    while (c != e); } // 889e
  ADD; pts.clear();
  while (qi < sz(q))
    if (!(e = q[qi++])->mark) ADD;
  return pts;
} // ef04

// --- CIRCUMCIRCLE (requires our vec2)
#include "../geometry/vec2.h"
typedef vec2d P;
double ccRadius(const P& A, const P& B,
    const P& C) {
  return (B-A).len()*(C-B).len()*
    (A-C).len()/abs((B-A).cross(C-A))/2;
} // 95ee
P ccCenter(const P& A, const P& B, const P& C){
  P b = C-A, c = B-A;
  return A + (b*c.len2()-c*b.len2()).perp()
    / b.cross(c)/2;
} // 2880

// --- MINIMUM ENCLOSING CIRCLE
// requires CIRCUMCIRCLE; expected O(n)
pair<P, double> mec(vector<P> ps) {
  shuffle(all(ps), mt19937(uint32_t(time(0))));
  P o = ps[0];
  double r = 0, EPS = 1 + 1e-8;
  rep(i,0,sz(ps)) if ((o - ps[i]).len() >
      r * EPS) {
    o = ps[i], r = 0;
    rep(j,0,i) if ((o - ps[j]).len() >
        r * EPS) {
      o = (ps[i] + ps[j]) / 2;
      r = (o - ps[i]).len();
      rep(k,0,j) if ((o - ps[k]).len() >
          r * EPS) {
        o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).len();
      } // b992
    } // 942d
  } // 0a5c
  return {o, r};
} // faa0

// --- POLYGON CENTER OF MASS
typedef vec2d P; // (requires our vec2d)
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]);
  } // eaf6
  return res / A / 3;
} // 405a
```

xyz/uj.h 33d1

```
// P = punkt 2D lub 3D, K = double
// --- GEOMETRIA OKREGOW W 2D ~Bartosz Walczak
struct circle { // okrag w 2D
  P c; K r; // srodek, promien
  circle(const P &ci, K ri=0) : c(ci),r(ri){}
  circle() {}
  K length() const { return 2*M_PI*r; }
  K area() const { return M_PI*r*r; }
}; // 2a8d
// Czy punkt lezy na okregu?
bool on_circle(const P &a, const circle &c)
  { return fabs((a-c.c).norm()-c.r*c.r) < EPS;}
// Czy kolo/punkt lezy wewnatrz lub na brzegu?
bool operator<(const circle &a,const circle &b)
  { return b.r+EPS > a.r && (a.c-b.c).norm() <
  (b.r-a.r)*(b.r-a.r)+EPS; } // 1f58
// Srodek okragu opisanego na trojkacie
circle circumcircle(P a, P b, P c) {
  if ((a-b).norm() > (c-b).norm()) swap(a,c);
  if ((b-c).norm() > (a-c).norm()) swap(a,b);
  if (fabs(det(b-a, c-b)) < EPS)
    throw "zdegenerowany";
  P v=intersection(median(a, b), median(b,c));
  return circle(v, sqrt((a-v).norm()));
} // 1327
// Przeciecie okregu i prostej.
// Zwraca liczbe punktow
int intersection(const circle &c,
      const line &p, P I[]/*OUT*/) {
  K d = p.n.norm(), a = (p.n*c.c-p.c)/d;
  P u = c.c-p.n*a; a *= a; K r = c.r*c.r/d;
  if (a >= r+EPS) return 0;
  if (a > r-EPS) { I[0]=u; return 1; }
  K h = sqrt(r-a);
  I[0] = u+cross(p.n)*h; I[1] = u-cross(p.n)*h;
  return 2;
} // 51f4
// Przeciecie dwoch okregow.
// Zwraca liczbe punktow. Zalozenie: c1.c!=c2.c
int intersection(const circle &c1,
    const circle &c2, P I[]/*OUT*/) {
  K d = (c2.c-c1.c).norm(), r1 =
    c1.r*c1.r/d, r2 = c2.r*c2.r/d;
  P u = c1.c*((r2-r1+1)*0.5) +
    c2.c*((r1-r2+1)*0.5);
  if (r1 > r2) swap(r1, r2);
  K a = (r1-r2+1)*0.5; a *= a;
  if (a >= r1+EPS) return 0;
  if (a > r1-EPS) { I[0] = u; return 1; }
  P v = cross(c2.c - c1.c); K h = sqrt(r1-a);
  I[0] = u+v*h; I[1] = u-v*h; return 2;
} // 45ea

// --- GEOMETRIA 3D ~Bartosz Walczak
// Kat pomiedzy dwoma wektorami. Zawsze >=0.
// Zalozenie: a,b!=0
K angle3(const xyz &a, const xyz &b)
  { return atan2(sqrt(cross(b,a).norm()),a*b);}
struct plane { // plaszczyzna {v: n*v=c}
  xyz n; K c; // (n - wektor normalny)
  plane(const xyz &ni, K ci) : n(ni), c(ci) {}
  plane() {}
}; // 93a3
// Czy punkt lezy na plaszczyznie?
bool on_plane(const xyz &a, const plane &p)
  { return fabs(p.n*a - p.c) < EPS; }
// Plaszczyzna rozpieta przez 3 punkty ccw.
// Zalozenie: a,b,c niezalezne
plane span3(const xyz &a, const xyz &b,
    const xyz &c)
  { xyz n = cross(c-a, b-a);
    return plane(n, n*a); } // 2d8e
// Plaszczyzna symetralna odcinka.
// Zalozenie: a!=b
plane median3(const xyz &a, const xyz &b)
  { return plane(b-a, (b-a)*(b+a)*0.5); }
// Plaszczyzna rownolegla przez punkt
plane parallel3(const xyz &a, const plane &p)
  { return plane(p.n, p.n*a); }
// Odleglosc punktu od plaszczyzny
K dist3(const xyz &a, const plane &p)
  { return fabs(p.n*a-p.c)/sqrt(p.n.norm()); }
struct line3 { // prosta {v: cross(v,u)=w}
    xyz u, w; // (u - wektor kierunku)
    // UWAGA! konstruktor dwuargumentowy
    // nie tworzy prostej przechodzacej
    // przez 2 punkty,
    // w tym celu nalezy uzyc span3!
    line3(const xyz &ui, const xyz &wi)
      : u(ui), w(wi) {}
    line3() {}
}; // 8ba4
// Czy punkt lezy na prostej?
bool on_line3(const xyz &a, const line3 &p)
  { return (cross(a,p.u)-p.w).norm() < EPS; }
// Prosta rozpieta przez 2 punkty. Za␣l.: a!=b
line3 span3(const xyz &a, const xyz &b)
  { return line3(b-a, cross(a, b-a)); }
// Plaszczyzna rozpieta przez prosta i
// punkt ccw. Zalozenie: cross(a,p.u)!=p.w
plane span3(const line3 &p, const xyz &a)
  { return plane(cross(a,p.u)-p.w, p.w*a); }
// Prosta przeciecia dwoch plaszczyzn
line3 intersection3(const plane &p,
    const plane &q) {
  xyz u=cross(q.n, p.n);
  if (u.norm() < EPS) throw "rownolegle";
  return line3(u, q.n*p.c-p.n*q.c);
} // 08c7
// Punkt przeciecia plaszczyzny i prostej
xyz intersection3(const plane &p,
    const line3 &q) {
  K d = q.u*p.n;
  if (fabs(d) < EPS) throw "rownolegle";
  return (q.u*p.c + cross(p.n, q.w))/d;
} // 7f03
// Prosta prostopadla do plaszczyzny
// przechodzaca przez punkt
line3 perp3(const xyz &a, const plane &p)
  { return line3(p.n, cross(a, p.n)); }
// Plaszczyzna prostopadla do prostej
// przechodzaca przez punkt
plane perp3(const xyz &a, const line3 &p)
  { return plane(p.u, p.u*a); }
// Odleglosc punktu od prostej
K dist3(const xyz &a, const line3 &p)
  { return sqrt((cross(a,p.u)-p.w).norm()) /
    sqrt(p.u.norm()); } // a713
// Odleglosc 2 prostych od siebie.
// Zalozenie: cross(q.u,p.u)!=0
// (niestabilne przy bliskim 0)
K dist3(const line3 &p, const line3 &q)
  { return fabs(p.u*q.w + q.u*p.w) /
    sqrt(cross(q.u, p.u).norm()); } // 88d8

// --- GEOMETRIA SFER W 3D ~Bartosz Walczak
struct sphere {
  xyz c; K r; // srodek, promien
  sphere(const xyz &ci, K ri=0)
    : c(ci), r(ri) {}
  sphere() {}
  // pole powierzchni
  K area() const { return 4*M_PI*r*r; }
  // objetosc kuli
  K volume() const { return 4*M_PI*r*r*r/3; }
}; // 029e
// Czy punkt lezy na sferze?
bool on_sphere(const xyz &a, const sphere &s)
  { return fabs((a-s.c).norm()-s.r*s.r) < EPS;}
// Czy sfera/punkt lezy wewnatrz lub na brzegu?
bool in_sphere(const sphere &a, const sphere&b)
  { return b.r+EPS > a.r && (a.c-b.c).norm() <
    (b.r-a.r)*(b.r-a.r)+EPS; } // 1f58
// Przeciecie sfery i prostej.
// Zwraca liczbe punktow przeciecia
int intersection3(const sphere &s,
  const line3 &p, xyz I[]/*OUT*/) {
  K d = p.u.norm(), a = (cross(s.c,p.u)-p.w).
    norm()/(d*d), r = s.r*s.r/d;
  if (a >= r+EPS) return 0;
  xyz u = (p.u*(p.u*s.c)+cross(p.u,p.w))/d;
  if (a > r-EPS) { I[0] = u; return 1; }
  K h = sqrt(r-a);
  I[0] = u+p.u*h; I[1] = u-p.u*h; return 2;
} // 14d7
// Przeciecie sfery i plaszczyzny.
// Zwraca true, jesli sie przecinaja. Wtedy u,r
// sa odp. srodkiem i promieniem okregu
// przeciecia. Zalozenie: s1.c!=s2.c
bool intersection3(const sphere &s,
    const plane &p, xyz &u, K &r) {
  K d = p.n.norm(), a = (p.n*s.c-p.c)/d;
  u = s.c-p.n*a; a *= a; K r1 = s.r*s.r/d;
  if (a >= r1+EPS) return false;
  r = a > r1-EPS ? 0 : sqrt(r1-a)*sqrt(d);
  return true;
} // d90d
// Przeciecie dwoch sfer.
// Zwraca true, jesli sie przecinaja.
// Wtedy u,r sa odp. srodkiem i promieniem
// okregu przeciecia. Zalozenie: s1.c!=s2.c
bool intersection3(const sphere &s1,
    const sphere &s2, xyz &u, K &r) {
  K d = (s2.c-s1.c).norm(), r1 = s1.r*s1.r/d,
    r2 = s2.r*s2.r/d;
  u = s1.c*((r2-r1+1)*0.5) +
    s2.c*((r1-r2+1)*0.5);
  if (r1 > r2) swap(r1, r2);
  K a = (r1-r2+1)*0.5; a *= a;
  if (a >= r1+EPS) return false;
  r = a > r1-EPS ? 0 : sqrt(r1-a)*sqrt(d);
  return true;
} // db4b

// --- GEOMETRIA NA SFERZE ~Bartosz Walczak
// Odleglosc dwoch punktow na sferze
K distS(const xyz &a, const xyz &b)
  { return atan2(sqrt(cross(b,a).norm()),a*b);}
struct circleS { // okrag na sferze
  xyz c; K r; // srodek, promien katowy
  circleS(const xyz &ci, K ri) : c(ci), r(ri){}
  circleS() {}
  K area() const { return 2*M_PI*(1-cos(r)); }
}; // faf2
// Okrag rozpiety przez 3 punkty.
```

```
// Zalozenie: punkty sa parami rozne
circleS spanS(xyz a, xyz b, xyz c) {
    int tmp = 1;
    if ((a-b).norm() > (c-b).norm())
      { swap(a, c); tmp = -tmp; }
    if ((b-c).norm() > (a-c).norm())
      { swap(a, b); tmp = -tmp; }
    xyz v = cross(c-b, b-a);
    v = v*(tmp/sqrt(v.norm()));
    return circleS(v, distS(a,v));
} // 7374
// Przeciecie 2 okregow na sferze.
// Zalozenie: cross(c2.c,c1.c)!=0
int intersectionS(const circleS &c1,
    const circleS &c2, xyz I[]/*OUT*/) {
  xyz n = cross(c2.c, c1.c),
    w = c2.c*cos(c1.r)-c1.c*cos(c2.r);
  K d = n.norm(), a = w.norm()/d;
  if (a >= 1+EPS) return 0;
  xyz u = cross(n,w)/d;
  if (a > 1-EPS) { I[0] = u; return 1; }
  K h = sqrt(1-a)/sqrt(d);
  I[0] = u+n*h; I[1] = u-n*h; return 2;
} // cbcb
```

Tutte Matrix. For a simple undirected graph G, Let M be a matrix with entries $A_{i,j} = 0$ if $(i,j) \notin E$ and $A_{i,j} = -A_{j,i} = X$ if $(i,j) \in E$. X could be any random value. If the determinants are non-zero, then a perfect matching exists, while other direction might not hold for very small probability.

Kirchhoff's Theorem. For a multigraph G with no loops, define Laplacian matrix as $L = D - A$. D is a diagonal matrix with $D_{i,i} = deg(i)$, and A is an adjacency matrix. If you remove any row and column of L, the determinant gives a number of spanning trees.

Burnside's lemma / Pólya enumeration theorem. let G and H be groups of permutations of finite sets X and Y. Let $c_m(g)$ denote the number of cycles of length m in $g \in G$ when permuting X. The number of colorings of X into $|Y| = n$ colors with exactly r_i occurrences of the i-th color is the coefficient of $w_1^{r_1} \dots w_n^{r_n}$ in the following polynomial:

$P(w_1, \dots, w_n) = \frac{1}{|H|} \sum_{h \in H} \frac{1}{|G|} \sum_{g \in G} \prod_{m \geq 1} (\sum_{h^m(b)=b} (w_b^m))^{c_m(g)}$

When $H = \{I\}$ (No color permutation):
$P(w_1, \dots, w_n) = \frac{1}{|G|} \sum_{g \in G} \prod_{m \geq 1} (w_1^m + \dots + w_n^m)^{c_m(g)}$

Without the occurrence restriction:
$P(1, \dots, 1) = \frac{1}{|G|} \sum_{g \in G} n^{c(g)}$
where $c(g)$ could also be interpreted as the number of elements in X that are fixed up to g.

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

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

Pick's Theorem. $A = i + \frac{b}{2} - 1$, where: P is a simple polygon whose vertices are grid points, A is area of P, i is # of grid points in the interior of P, and b is # of grid points on the boundary of P. If h is # of holes of P ($h+1$ simple closed curves in total), $A = i + \frac{b}{2} + h - 1$.

Xudyh Sieve. $F(n) = \sum_{d|n} f(d)$

$S(n) = \sum_{i \leq n} f(i) = \sum_{i \leq n} F(i) - \sum_{d=2}^{n} S\left(\lfloor \frac{n}{d} \rfloor\right)$

Preprocess $S(1)$ to $S(M)$ (Set $M = n^{\frac{2}{3}}$ for complexity)

$S(n) = \sum f(i) = \sum_{i \leq n} \left[F(i) - \sum_{j|i, j \neq i} f(j)\right] = \sum F(i) - \sum_{i/j=d=2}^{n} \sum_{dj \leq n} f(j)$

$S(n) = \sum i f(i) = \sum_{i \leq n} i \left[F(i) - \sum_{j|i, j \neq i} f(j)\right] = \sum i F(i) - \sum_{i/j=d=2}^{n} \sum_{dj \leq n} dj f(j)$

$\sum_{d|n} \varphi(d) = n$ $\quad \sum_{d|n} \mu(d) =$ if $(n > 1)$ then 0 else 1 $\quad \sum_{d|n} (\mu(\frac{n}{d}) \sum_{e|d} f(e)) = f(n)$

Labeled unrooted trees

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

$$x_1 x_2 x_3 \dots x_n (x_1 + x_2 + \dots x_n)^{n-2} = \sum_T x_1^{d_{T(1)}} x_2^{d_{T(2)}} \dots x_n^{d_{T(n)}}$$

where the sum is over all spanning trees T in K_n and $d_{T(i)}$ is the degree of i in T

Partition function

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

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

Taylor's theorem[4][5][6] — Let $k \geq 1$ be an integer and let the function $f : \mathbf{R} \to \mathbf{R}$ be k times differentiable at the point $a \in \mathbf{R}$. Then there exists a function $h_k : \mathbf{R} \to \mathbf{R}$ such that

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(k)}(a)}{k!}(x-a)^k + h_k(x)(x-a)^k,$$

Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1} - c^a}{c-1}, c \neq 1 \qquad 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6} \qquad 1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$

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

Triangles

Side lengths: a, b, c

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

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

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

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

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

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

Length of bisector (divides angles in two):

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

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

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

Symbol Newtona:

$$\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n} \qquad \sum_{k=1}^{n} k \binom{n}{k}^2 = n2^{n-1} \qquad \sum_{k=0}^{n} \binom{r}{k}\binom{s}{n-k} = \binom{r+s}{m+n}$$

Liczby Stirlinga I rodzaju (liczba permutacji n elementow o k cyklach):

$$\left[{n+1 \atop k}\right] = n\left[{n \atop k}\right] + \left[{n \atop k-1}\right] \qquad \sum_{p=k}^{n} \left[{n \atop p}\right]\binom{p}{k} = \left[{n+1 \atop k+1}\right]$$

Liczby Stirlinga II rodzaju (liczba podzialow zbioru n-elementowego na k klas)

$$\left\{{n \atop k}\right\} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j} j^n \qquad \left\{{n+1 \atop k}\right\} = k\left\{{n \atop k}\right\} + \left\{{n \atop k-1}\right\}$$

$$\left\{{n \atop k}\right\} \equiv \binom{z}{w} \pmod 2, \quad z = n - \left\lceil \frac{k+1}{2} \right\rceil, \; w = \left\lfloor \frac{k-1}{2} \right\rfloor$$

Jesli kazde 2 elementy zbioru musza byc odlegle o co najmniej d:

$S^d(n,k) = S(n-d+1, k-d+1), n \geq k \geq d$

Jesli $f(n) = \sum_{d|n} g(d)$, to $g(n) = \sum_{d|n} \mu(d) f\left(\frac{n}{d}\right)$

gdzie $\mu(1) = 1$, $\mu(p^2 \cdot a) = 0$, $\mu(p_1 \cdot p_2 \cdot \ldots \cdot p_k) = (-1)^k$ dla p, p_i pierwszych.

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

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

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

$$\frac{1}{1-x} = \sum_{n \geq 0} x^n \qquad \log\frac{1}{1-x} = \sum_{n \geq 1} \frac{x^n}{n}$$

$$\sin x = \sum_{n \geq 0} (-1)^n \frac{x^{2n+1}}{(2n+1)!} \qquad \cos x = \sum_{n \geq 0} (-1)^n \frac{x^{2n}}{(2n)!}$$

$$\tan^{-1} x = \sum_{n \geq 0} (-1)^n \frac{x^{2n+1}}{2n+1} \qquad \frac{1}{(1-x)^{k+1}} = \sum_n \binom{n+k}{n} x^n$$

$$\frac{1}{\sqrt{1-4x}}\left(\frac{1-\sqrt{1-4x}}{2x}\right)^k = \sum_n \binom{2n+k}{n} x^n$$

$$\frac{1}{2x}(1 - \sqrt{1-4x}) = \sum_n \frac{1}{n+1}\binom{2n}{n} x^n \qquad \frac{1}{\sqrt{1-4x}} = \sum_k \binom{2k}{k} x^k$$