Bangladesh University of Engineering and Technology

BUET Harmonica

Sadat Hossain, Faria Binta Awal, Shafin Ahmed

2024-12-06

BUET Harmonica

Sadat Hossain, Faria Binta Awal, Shafin Ahmed

Techniques (A)

techniques.txt 159 lines

```
Recursion
Divide and conquer
  Finding interesting points in N log N
Algorithm analysis
  Master theorem
  Amortized time complexity
Greedy algorithm
  Scheduling
  Max contiguous subvector sum
  Invariants
  Huffman encoding
Graph theory
  Dynamic graphs (extra book-keeping)
  Breadth first search
  Depth first search
  * Normal trees / DFS trees
  Dijkstra's algorithm
  MST: Prim's algorithm
  Bellman-Ford
  Konig's theorem and vertex cover
  Min-cost max flow
  Lovasz toggle
  Matrix tree theorem
  Maximal matching, general graphs
  Hopcroft-Karp
  Hall's marriage theorem
  Graphical sequences
  Floyd-Warshall
  Euler cycles
  Flow networks
  * Augmenting paths
  * Edmonds-Karp
  Bipartite matching
  Min. path cover
  Topological sorting
  Strongly connected components
  2-SAT
  Cut vertices, cut-edges and biconnected components
  Edge coloring
  * Trees
  Vertex coloring
  * Bipartite graphs (=> trees)
  * 3^n (special case of set cover)
  Diameter and centroid
  K'th shortest path
  Shortest cycle
Dynamic programming
  Knapsack
  Coin change
  Longest common subsequence
  Longest increasing subsequence
  Number of paths in a dag
  Shortest path in a dag
  Dynprog over intervals
  Dynprog over subsets
  Dynprog over probabilities
  Dynprog over trees
  3^n set cover
  Divide and conquer
  Knuth optimization
  Convex hull optimizations
  RMQ (sparse table a.k.a 2^k-jumps)
  Bitonic cycle
  Log partitioning (loop over most restricted)
Combinatorics
  Computation of binomial coefficients
  Pigeon-hole principle
  Inclusion/exclusion
  Catalan number
  Pick's theorem
Number theory
  Integer parts
  Divisibility
  Euclidean algorithm
  Modular arithmetic
  * Modular multiplication
  * Modular inverses
  * Modular exponentiation by squaring
  Chinese remainder theorem
  Fermat's little theorem
  Euler's theorem
  Phi function
  Frobenius number
  Quadratic reciprocity
  Pollard-Rho
  Miller-Rabin
  Hensel lifting
  Vieta root jumping
Game theory
  Combinatorial games
  Game trees
  Mini-max
  Nim
  Games on graphs
  Games on graphs with loops
  Grundy numbers
  Bipartite games without repetition
  General games without repetition
  Alpha-beta pruning
Probability theory
Optimization
  Binary search
  Ternary search
  Unimodality and convex functions
  Binary search on derivative
Numerical methods
  Numeric integration
  Newton's method
  Root-finding with binary/ternary search
  Golden section search
Matrices
  Gaussian elimination
  Exponentiation by squaring
Sorting
  Radix sort
Geometry
  Coordinates and vectors
  * Cross product
  * Scalar product
  Convex hull
  Polygon cut
  Closest pair
  Coordinate-compression
  Quadtrees
  KD-trees
  All segment-segment intersection
Sweeping
  Discretization (convert to events and sweep)
  Angle sweeping
  Line sweeping
  Discrete second derivatives
Strings
  Longest common substring
  Palindrome subsequences
  Knuth-Morris-Pratt
  Tries
  Rolling polynomial hashes
  Suffix array
  Suffix tree
  Aho-Corasick
  Manacher's algorithm
  Letter position lists
Combinatorial search
  Meet in the middle
  Brute-force with pruning
  Best-first (A*)
  Bidirectional search
  Iterative deepening DFS / A*
Data structures
  LCA (2^k-jumps in trees in general)
  Pull/push-technique on trees
  Heavy-light decomposition
  Centroid decomposition
  Lazy propagation
  Self-balancing trees
  Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
  Monotone queues / monotone stacks / sliding queues
  Sliding queue using 2 stacks
  Persistent segment tree
```

1 Contest 1

2 Mathematics 1

3 Data structures 2

4 Numerical 5

5 Number theory 8

6 Combinatorial 10

7 Graph 11

8 Geometry 16

9 Strings 21

10 Various 23

Contest (1)

.vimrc

14 lines

```
set nocompatible
set number
set relativenumber
set tabstop=4
set shiftwidth=4
set expandtab
set autoindent
syntax on
set clipboard=unnamedplus

inoremap jj <Esc>
nnoremap <F5> :w<CR>:!g++ -std=c++17 -O2 % -o %:r<CR>
nnoremap <F6> :!./%:r<CR>
inoremap {<CR> {<CR>}<CR><Esc>kO<Tab>
```

debug.h

49 lines

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/pb_ds/exception.hpp>
#include <ext/pb_ds/hash_policy.hpp>
#include <ext/pb_ds/list_update_policy.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/trie_policy.hpp>

void __print(int x) { cerr << x; }
void __print(long x) { cerr << x; }
void __print(long long x) { cerr << x; }
void __print(unsigned x) { cerr << x; }
void __print(unsigned long x) { cerr << x; }
void __print(unsigned long long x) { cerr << x; }
void __print(float x) { cerr << x; }
void __print(double x) { cerr << x; }
void __print(long double x) { cerr << x; }
void __print(char x) { cerr << '\'' << x << '\''; }
void __print(const char *x) { cerr << '\"' << x << '\"'; }
void __print(const string &x) { cerr << '\"' << x << '\"'; }
void __print(bool x) { cerr << (x ? "true" : "false"); }

template <typename T, typename V>
void __print(const pair<T, V> &x) {
    cerr << '{';
    __print(x.first);
    cerr << ',';
    __print(x.second);
    cerr << '}';
}
template <typename T>
void __print(const T &x) {
    int f = 0;
    cerr << '{';
    for (auto &i : x) cerr << (f++ ? "," : ""), __print(i);
    cerr << "}";
}
void _print() { cerr << "]\n"; }
template <typename T, typename... V>
void _print(T t, V... v) {
    __print(t);
    if (sizeof...(v)) cerr << ", ";
    _print(v...);
}

#define debug(x...)                \
    cerr << "[" << #x << "] = ["; \
    _print(x)
// add a newline at the end
```

template.cpp

24 lines

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

// g++ -fsanitize=address -static-libasan -g -DLOCAL -O3 -std=c
    ++14 b.cpp
#ifdef LOCAL
#include "debug.h"
#else
#define debug(...)
#endif

#define endl "\n"
typedef long long ll;

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

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

troubleshoot.txt

52 lines

```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a teammate do it.

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

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your teammates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all data structures between test cases?
```

Mathematics (2)

2.1 Equations

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The extremum is given by $x = -b/2a$.

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned} \Rightarrow \begin{aligned} x &= \frac{ed - bf}{ad - bc} \\ y &= \frac{af - ec}{ad - bc} \end{aligned}$$

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

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

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

2.2 Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k - c_1 x^{k-1} - \cdots - c_k$, there are $d_1, \ldots, d_k$ s.t.

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

2.3 Trigonometry

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

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

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

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

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

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

2.4 Geometry

2.4.1 Triangles

Side lengths: a, b, c

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

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

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

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

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

Length of bisector (divides angles in two):

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

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

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

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

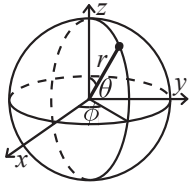
2.4.2 Quadrilaterals

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

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

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

2.4.3 Spherical coordinates



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

2.5 Derivatives/Integrals

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

Integration by parts:

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

2.6 Sums

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

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$
$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$
$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$
$$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

2.7 Series

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

2.8 Probability theory

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

Expectation is linear:

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

For independent X and Y,

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

Data structures (3)

OrderStatisticTree.h (22 lines)

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;

template <typename T> using o_set = tree<T, null_type, less<T>,
     rb_tree_tag, tree_order_statistics_node_update>;
template <typename T, typename R> using o_map = tree<T, R, less
    <T>, rb_tree_tag, tree_order_statistics_node_update>;
int main() {
  int i, j, k, n, m;
  o_set<int>se;
  se.insert(1);
  se.insert(2);
  cout << *se.find_by_order(0) << endl; ///k th element
  cout << se.order_of_key(2) << endl; ///number of elements
       less than k
  o_map<int, int>mp;
  mp.insert({1, 10});
  mp.insert({2, 20});
  cout << mp.find_by_order(0)->second << endl; ///k th element
  cout << mp.order_of_key(2) << endl; ///number of first
       elements less than k
  return 0;
}
```

Segtree.h

Description: Basic segment tree with range min and point update requires proper modification
Time: $\mathcal{O}(\log N)$

eab4c0, 47 lines

```
const int MX = 2e5 + 5;
vector<int> numbers(MX);

struct seg_tree {
    vector<int> vec;
    const int NEUTRAL_ELEMENT = INT_MAX;
    inline int lc(int x) { return (x << 1); }        // left
        child
    inline int rc(int x) { return ((x << 1) | 1); }  // right
        child

    seg_tree(int n) { vec.assign(4 * n, NEUTRAL_ELEMENT); }
    inline int combine(int a, int b) { return min(a, b); }
    // call build(1, 1, n)
    void build(int at, int start, int end) {
        if (start == end) {
            vec[at] = numbers[start];
            return;
        }
        int mid = (start + end) >> 1;
        build(lc(at), start, mid);
        build(rc(at), mid + 1, end);
        vec[at] = combine(vec[lc(at)], vec[rc(at)]);
        return;
    }
    // for updating index i, call update(1, 1, n, i)
    void update(int at, int start, int end, int update_index) {
        if (start == end && start == update_index) {
            vec[at] = numbers[update_index];
            return;
        }
        int mid = (start + end) >> 1;
        if (update_index <= mid) {
            update(lc(at), start, mid, update_index);
        } else
            update(rc(at), mid + 1, end, update_index);
        vec[at] = combine(vec[lc(at)], vec[rc(at)]);
        return;
    }
    // for query [l, r] call query(1, 1, n, l, r)
    int query(int at, int start, int end, int q_left, int
        q_right) {
        if (start > q_right || end < q_left) return
            NEUTRAL_ELEMENT;
        if (start >= q_left && end <= q_right) return vec[at];
        int mid = (start + end) >> 1;
        int l = query(lc(at), start, mid, q_left, q_right);
        int r = query(rc(at), mid + 1, end, q_left, q_right);
        return combine(l, r);
    }
};
```

SegtreeBeats.h

Description: Segment tree beats, supports a[i] := a[i] mod x can be modified by finding proper break condition and tag condition
Time: $\mathcal{O}(\log N)$

4fd31d, 47 lines

```
const int MAXN = 100001;

int N, Q;
long long tsum[MAXN * 4], tmax[MAXN * 4];

void update_mod(int l, int r, long long v, int t = 1, int tl =
    1, int tr = N) {
  if (r < tl || tr < l || tmax[t] < v) {
    return;
  } else if (tl == tr) {
    int val = tmax[t] % v;
    tsum[t] = tmax[t] = val;
    return;
  }

  int tm = (tl + tr) / 2;
  update_mod(l, r, v, t * 2, tl, tm);
  update_mod(l, r, v, t * 2 + 1, tm + 1, tr);
  tsum[t] = tsum[t * 2] + tsum[t * 2 + 1];
  tmax[t] = max(tmax[t * 2], tmax[t * 2 + 1]);
}

void update_set(int i, long long v, int t = 1, int tl = 1, int
    tr = N) {
  if (tl == tr) {
    tsum[t] = tmax[t] = v;
    return;
  }

  int tm = (tl + tr) / 2;
  if (i <= tm) {
    update_set(i, v, t * 2, tl, tm);
  } else {
    update_set(i, v, t * 2 + 1, tm + 1, tr);
  }
  tsum[t] = tsum[t * 2] + tsum[t * 2 + 1];
  tmax[t] = max(tmax[t * 2], tmax[t * 2 + 1]);
}

long long query(int l, int r, int t = 1, int tl = 1, int tr = N
    ) {
  if (r < tl || tr < l) {
    return 0;
  } else if (l <= tl && tr <= r) {
    return tsum[t];
  }

  int tm = (tl + tr) / 2;
  return query(l, r, t * 2, tl, tm) + query(l, r, t * 2 + 1, tm
      + 1, tr);
}
```

LazySegmentTree.h

Description: Segment tree with ability to add or set values of large intervals, and compute max of intervals. Can be changed to other things. Use with a bump allocator for better performance, and SmallPtr or implicit indices to save memory.
Usage: `Node* tr = new Node(v, 0, sz(v));`
Time: $\mathcal{O}(\log N)$.

"../various/BumpAllocator.h" 34ecf5, 50 lines

```
const int inf = 1e9;
struct Node {
  Node *l = 0, *r = 0;
  int lo, hi, mset = inf, madd = 0, val = -inf;
  Node(int lo,int hi):lo(lo),hi(hi){} // Large interval of -inf
  Node(vi& v, int lo, int hi) : lo(lo), hi(hi) {
    if (lo + 1 < hi) {
      int mid = lo + (hi - lo)/2;
      l = new Node(v, lo, mid); r = new Node(v, mid, hi);
      val = max(l->val, r->val);
    }
    else val = v[lo];
  }
  int query(int L, int R) {
    if (R <= lo || hi <= L) return -inf;
    if (L <= lo && hi <= R) return val;
    push();
    return max(l->query(L, R), r->query(L, R));
  }
  void set(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) mset = val = x, madd = 0;
    else {
      push(), l->set(L, R, x), r->set(L, R, x);
      val = max(l->val, r->val);
    }
  }
  void add(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) {
      if (mset != inf) mset += x;
      else madd += x;
      val += x;
    }
    else {
      push(), l->add(L, R, x), r->add(L, R, x);
      val = max(l->val, r->val);
    }
  }
  void push() {
    if (!l) {
      int mid = lo + (hi - lo)/2;
      l = new Node(lo, mid); r = new Node(mid, hi);
    }
    if (mset != inf)
      l->set(lo,hi,mset), r->set(lo,hi,mset), mset = inf;
    else if (madd)
      l->add(lo,hi,madd), r->add(lo,hi,madd), madd = 0;
  }
};
```

UnionFindRollback.h

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

de4ad0, 21 lines

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

Matrix.h

Description: Basic operations on square matrices.
Usage: `Matrix<int, 3> A;`

c43c7d, 26 lines

```
A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};
vector<int> vec = {1,2,3};
vec = (A^N) * vec;
```

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

LineContainer.h

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

8ec1c7, 30 lines

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

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

Treap.h

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

9556fc, 55 lines

```
struct Node {
  Node *l = 0, *r = 0;
  int val, y, c = 1;
  Node(int val) : val(val), y(rand()) {}
  void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template<class F> void each(Node* n, F f) {
  if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
  if (!n) return {};
  if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound(k)
    auto pa = split(n->l, k);
    n->l = pa.second;
    n->recalc();
    return {pa.first, n};
  } else {
    auto pa = split(n->r, k - cnt(n->l) - 1); // and just "k"
    n->r = pa.first;
    n->recalc();
    return {n, pa.second};
  }
}

Node* merge(Node* l, Node* r) {
  if (!l) return r;
  if (!r) return l;
  if (l->y > r->y) {
    l->r = merge(l->r, r);
    l->recalc();
    return l;
  } else {
    r->l = merge(l, r->l);
    r->recalc();
    return r;
  }
}

Node* ins(Node* t, Node* n, int pos) {
  auto pa = split(t, pos);
  return merge(merge(pa.first, n), pa.second);
}

// Example application: move the range [l, r) to index k
void move(Node*& t, int l, int r, int k) {
  Node *a, *b, *c;
  tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
  if (k <= l) t = merge(ins(a, b, k), c);
  else t = merge(a, ins(c, b, k - r));
}
```

FenwickTree.h

Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value.
Time: Both operations are $\mathcal{O}(\log N)$.

e62fac, 22 lines

```
struct FT {
  vector<ll> s;
  FT(int n) : s(n) {}
  void update(int pos, ll dif) { // a[pos] += dif
    for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
  }
  ll query(int pos) { // sum of values in [0, pos)
    ll res = 0;
    for (; pos > 0; pos &= pos - 1) res += s[pos-1];
    return res;
  }
  int lower_bound(ll sum) {// min pos st sum of [0, pos] >= sum
    // Returns n if no sum is >= sum, or -1 if empty sum is.
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {
      if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
        pos += pw, sum -= s[pos-1];
    }
    return pos;
  }
};
```

FenwickTree2d.h

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

"FenwickTree.h" 157f07, 22 lines

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

RMQ.h

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

510c32, 16 lines

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

MoQueries.h

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

a12ef4, 49 lines

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

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

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

SQRT.h

Description: Basic square root decomposition
Time: $\mathcal{O}\left(sqrt(N)\right)$ per query

8c140f, 27 lines

```
int n, q;
vector<int> a(n);
int main() {
    // input data
    //  preprocessing
    const int len =
        (int)sqrt(n + .0) + 1;  // size of the block and the
              number of blocks
    vector<int> b(len);
    for (int i = 0; i < n; ++i) b[i / len] += a[i];

    // answering the queries
    while (q--) {
        int l, r;
        // read input data for the next query
        cin >> l >> r;
        int sum = 0;
        int c_l = l / len, c_r = r / len;
        if (c_l == c_r)
            for (int i = l; i <= r; ++i) sum += a[i];
        else {
            for (int i = l, end = (c_l + 1) * len - 1; i <= end
                 ; ++i)
                sum += a[i];
            for (int i = c_l + 1; i <= c_r - 1; ++i) sum += b[i
                 ];
            for (int i = c_r * len; i <= r; ++i) sum += a[i];
        }
    }
}
```

Eertree.h

Description: Palindromic tree
Time: $\mathcal{O}\left(N\right)$

34ba40, 32 lines

```
const int maxn = 1e5, sigma = 26;

int s[maxn], len[maxn], link[maxn], to[maxn][sigma];

int n, last, sz;

void init()
{
    s[n++] = -1;
    link[0] = 1;
    len[1] = -1;
    sz = 2;
}

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

void add_letter(int c)
{
    s[n++] = c;
    last = get_link(last);
    if(!to[last][c])
    {
        len [sz] = len[last] + 2;
        link[sz] = to[get_link(link[last])][c];
        to[last][c] = sz++;
    }
    last = to[last][c];
}
```

Trie.h

Description: Bitwise Trie, can be changed to accommodate strings
Time: $\mathcal{O}\left(\sum N\right)$

a5e4a5, 56 lines

```
struct Trie {
  static const int B = 31;
  struct node {
    node* nxt[2];
    int sz;
    node() { nxt[0] = nxt[1] = NULL; sz = 0; }
  } *root;
  Trie() { root = new node(); }
  void insert(int val) {
    node* cur = root;
    cur -> sz++;
    for (int i = B - 1; i >= 0; i--) {
      int b = val >> i & 1;
      if (cur -> nxt[b] == NULL) cur -> nxt[b] = new node();
      cur = cur -> nxt[b];
      cur -> sz++;
    }
  }
  int query(int x, int k) { // number of values s.t. val ^ x <
       k
    node* cur = root;
    int ans = 0;
    for (int i = B - 1; i >= 0; i--) {
      if (cur == NULL) break;
      int b1 = x >> i & 1, b2 = k >> i & 1;
      if (b2 == 1) {
        if (cur -> nxt[b1]) ans += cur -> nxt[b1] -> sz;
        cur = cur -> nxt[!b1];
      } else cur = cur -> nxt[b1];
    }
    return ans;
  }
  int get_max(int x) { // returns maximum of val ^ x
    node* cur = root;
    int ans = 0;
    for (int i = B - 1; i >= 0; i--) {
      int k = x >> i & 1;
      if (cur -> nxt[!k]) cur = cur -> nxt[!k], ans <<= 1, ans
           ++;
      else cur = cur -> nxt[k], ans <<= 1;
    }
    return ans;
  }
  int get_min(int x) { // returns minimum of val ^ x
    node* cur = root;
    int ans = 0;
    for (int i = B - 1; i >= 0; i--) {
      int k = x >> i & 1;
      if (cur -> nxt[k]) cur = cur -> nxt[k], ans <<= 1;
      else cur = cur -> nxt[!k], ans <<= 1, ans++;
    }
    return ans;
  }
  void del(node* cur) {
    for (int i = 0; i < 2; i++) if (cur -> nxt[i]) del(cur ->
         nxt[i]);
    delete(cur);
  }
} t;
```

Numerical (4)

4.1 Polynomials

Polynomial.h

c9b7b0, 17 lines

```
struct Poly {
  vector<double> a;
  double operator()(double x) const {
    double val = 0;
    for (int i = sz(a); i--;) (val *= x) += a[i];
    return val;
  }
  void diff() {
```

```
    rep(i,1,sz(a)) a[i-1] = i*a[i];
    a.pop_back();
  }
  void divroot(double x0) {
    double b = a.back(), c; a.back() = 0;
    for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
    a.pop_back();
  }
};
```

PolyRoots.h

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

"Polynomial.h" b00bfe, 23 lines

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

PolyInterpolate.h

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

08bf48, 13 lines

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

4.2 Matrices

Determinant.h

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

bd5cec, 15 lines

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

MatrixInverse.h

Description: Invert matrix A. Returns rank; result is stored in A unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1})$ (mod p^k) where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.
Time: $\mathcal{O}\left(n^3\right)$

ebfff6, 35 lines

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

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

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

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

SolveLinear.h

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

44c9ab, 38 lines

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

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

  rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
      rep(j,i,n) if (fabs(b[j]) > eps) return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }

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

SolveLinear2.h

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

"SolveLinear.h" 08e495, 7 lines

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

SolveLinearBinary.h

Description: Solves $Ax = b$ over $\mathbb{F}_2$. If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.
Time: $\mathcal{O}\left(n^2 m\right)$

fa2d7a, 34 lines

```
typedef bitset<1000> bs;

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

```
    }
    rank++;
  }

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

4.3 Fourier transforms

FastFourierTransform.h

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

Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ ($\sim 1s$ for $N = 2^{22}$)

00ced6, 35 lines

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

FastFourierTransformMod.h

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

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

"FastFourierTransform.h" b82773, 22 lines

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

NumberTheoreticTransform.h

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

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

"../number-theory/ModPow.h" ced03d, 35 lines

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

FastSubsetTransform.h

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

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

464cf3, 16 lines

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

Subset.h

Description: Various subset convolutions

Time: $\mathcal{O}\left(2^K * K^2\right)$ for conv, $\mathcal{O}\left(2^K * K\right)$ for others

bf4921, 91 lines

```
vector<LL> XorTransform(vector<LL> p, bool inverse) {
    int n = p.size();
    assert((n&(n-1))==0);
    for (int len = 1; 2*len <= n; len <<= 1) {
        for (int i = 0; i < n; i += len+len) {
            for (int j = 0; j < len; j++) {
                LL u = p[i+j], v = p[i+len+j];
                if (!inverse)   p[i+j] = u+v, p[i+len+j] = u-v;
                else            p[i+j] = (u+v)/2, p[i+len+j] =
                                    (u-v)/2;
            }
        }
    }
    return p;
}

vector<LL> SOS(vector<LL> p, bool inverse, bool subset) {
    int k = __builtin_ctz(p.size());
    assert(p.size() == (1<<k));
    for (int i=0; i<k; i++)
        for (int mask=0; mask<(1<<k); mask++)
            if (bool(mask & (1<<i)) == subset) {
                if (!inverse) p[mask] += p[mask^(1<<i)];
                else          p[mask] -= p[mask^(1<<i)];
            }
    return p;
}

vector<LL> product(const vector<LL> &a, const vector<LL> &b) {
    assert(a.size() == b.size());
    vector<LL> ans(a.size());
    for (int i=0; i<a.size(); i++)  ans[i] = a[i] * b[i];
    return ans;
}

vector<LL> XorConvolution(vector<vector<LL>> vs) {
    int n = vs.size();
    for (int i=0; i<n; i++) vs[i] = XorTransform(vs[i], 0);
    vector<LL> ans = vs[0];
    for (int i=1; i<n; i++) ans = product(ans, vs[i]);
```

```
    ans = XorTransform(ans, 1);
    return ans;
}

vector<LL> ORConvolution(vector<vector<LL>> vs) {
    int n = vs.size();
    for (int i=0; i<n; i++) vs[i] = SOS(vs[i], 0, 1);

    vector<LL> ans = vs[0];
    for (int i=1; i<n; i++) ans = product(ans, vs[i]);
    ans = SOS(ans, 1, 1);
    return ans;
}

vector<LL> AndConvolution(vector<vector<LL>> vs) {
    int n = vs.size();
    for (int i=0; i<n; i++) vs[i] = SOS(vs[i], 0, 0);

    vector<LL> ans = vs[0];
    for (int i=1; i<n; i++) ans = product(ans, vs[i]);
    ans = SOS(ans, 1, 0);
    return ans;
}

vector<LL> SubsetConvolution(const vector<LL> &a, const vector<
    LL> &b) {
    int k = __builtin_ctz(a.size());
    assert(a.size() == (1<<k) && b.size() == (1<<k));

    vector<LL> Z(1<<k);
    vector<vector<LL>> A(k+1, Z), B(k+1, Z), C(k+1, Z);

    for (int mask=0; mask<(1<<k); mask++) {
        A[__builtin_popcount(mask)][mask] = a[mask];
        B[__builtin_popcount(mask)][mask] = b[mask];
    }

    for (int i=0; i<=k; i++) {
        A[i] = SOS(A[i], 0, 1);
        B[i] = SOS(B[i], 0, 1);
        for (int j=0; j<=i; j++)
            for (int mask = 0; mask < (1<<k); mask++)
                C[i][mask] += A[j][mask]*B[i-j][mask];
        C[i] = SOS(C[i], 1, 1);
    }

    vector<LL> ans(1<<k);
    for (int mask=0; mask<(1<<k); mask++) {
        ans[mask] = C[__builtin_popcount(mask)][mask];
    }
    return ans;
}
```

Number theory (5)

5.1 Modular arithmetic

BasicMath.h

Description: Basic math and modular arithmetic

268112, 93 lines

```
const int MX = 2e5 + 10;
array<int, MX> factorial;
const int MOD1 = 1e9 + 7;
inline int add(int a, int b) {
    a += b;
    while (a >= MOD1) a -= MOD1;
    while (a < 0) a += MOD1;
    return a;
}
inline void add_self(int& a, int b) { a = add(a, b); }
inline int sub(int a, int b) { return add(a, -b); }
inline void sub_self(int& a, int b) { a = sub(a, b); }
inline int mul(int a, int b) { return (a * 1LL * b) % MOD1; }
inline void mul_self(int& a, int b) { a = mul(a, b); }
int power(int a, int b) {
    int ans = 1;
    while (b) {
        if (b & 1) mul_self(ans, a);
        mul_self(a, a);
        b >>= 1;
    }
    return ans;
}
int gcd(int a, int b, int& x, int& y) {
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
    return a1;
}
int modInverse(int a, int m) {
    int x, y;
    int g = gcd(a, m, x, y);
    while (x < 0) x += m;
    return x;
}

inline int divide(int a, int b) { return mul(a, modInverse(b,
    MOD1)); }
inline void divide_self(int& a, int b) { a = divide(a, b); }
void precompute() {
    factorial[0] = 1;
    for (int i = 1; i < MX; i++) {
        factorial[i] = mul(factorial[i - 1], i);
    }
}
inline int nCr(int n, int r) {
    static bool isPrecomputationDone = false;
    if (!isPrecomputationDone) {
        precompute();
        isPrecomputationDone = true;
    }
    if (r > n || n < 0 || r < 0) return 0;
    if (n == r || r == 0) return 1;
    return divide(factorial[n], mul(factorial[r], factorial[n -
        r]));
}
inline int nPr(int n, int r) {
    static bool isPrecomputationDone = false;
    if (!isPrecomputationDone) {
        precompute();
        isPrecomputationDone = true;
    }
    if (r > n || n < 0 || r < 0) return 0;
    if (r == 0) return 1;
    if (n == r) return factorial[n];
    return divide(factorial[n], factorial[n - r]);
}

__int128 read() {
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}
void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}
bool cmp(__int128 x, __int128 y) { return x > y; }
```

Congruence.h

Description: Necessary congruence templates. well commented

e810ad, 173 lines

```
typedef pair<LL, LL> PLL;

inline LL modit(LL x, LL m) {
  LL z = x%m; return z<0 ? z+m : z;
}

LL gcd(LL u, LL v) {
    if (u == 0) return v;
    if (v == 0) return u;
    int shift = __builtin_ctzll(u | v);
    u >>= __builtin_ctzll(u);
    do {
        v >>= __builtin_ctzll(v);
        if (u > v) swap(u, v);
        v = v-u;
    } while (v);
    return u << shift;
}

LL power(LL a, LL b, LL m) {
    a = modit(a, m);
  LL ans = 1;
  while (b) {
    if (b & 1) ans = (ans*a)%m;
    a = (a*a)%m;
    b >>= 1;
  }
  return ans;
}

LL egcd(LL a, LL b, LL &x, LL &y) {
  LL xx = y = 0;
  LL yy = x = 1;
  while (b) {
    LL q = a/b;
    LL t = b; b = a%b; a = t;
    t = xx; xx = x-q*xx; x = t;
    t = yy; yy = y-q*yy; y = t;
  }
  return a;
}
LL inverse(LL a, LL m) {
  LL x, y;
  LL g = egcd(a, m, x, y);
  if (g > 1) return -1;
  return modit(x, m);
}
```

```
PLL CRT(LL m1, LL r1, LL m2, LL r2) {
  LL s, t;
  LL g = egcd(m1, m2, s, t);
  if (r1%g != r2%g) return PLL(0, -1);
  LL M = m1*m2;
  LL ss = ((s*r2)%m2)*m1;
  LL tt = ((t*r1)%m1)*m2;
  LL ans = modit(ss+tt, M);
  return PLL(ans/g, M/g);
}

pair<LL, LL> SolveCongruence(LL a, LL b, LL m) {
  LL x, y;
  LL g = egcd(a, m, x, y);
  if (b%g == 0) {
        LL d = m/g;
    x = modit(x*(b/g), d);
    return {x, d};
  }
  return {-1, -1};
}

bool LinearDiophantine(LL a, LL b, LL c, LL &x, LL &y) {
  if (!a && !b) {
    if (c)          return false;
    x = y = 0;      return true;
  }
  if (!a) {
    if (c%b)        return false;
    x = 0; y = c/b; return true;
  }
  if (!b) {
    if (c%a)        return false;
    x = c/a; y = 0; return true;
  }
  LL g = gcd(a, b);
  if (c%g) return false;
  x = c/g*inverse(a/g, b/g);
  y = (c-a*x)/b;
  return true;
}

LL primitive_root(LL p) {
    if (p == 2) return 1;
    LL phi = p-1, n = phi;

    vector<LL> factor;
    for (int i=2; i*i<=n; ++i)
        if (n%i == 0) {
            factor.push_back (i);
            while (n%i==0)  n/=i;
        }

    if (n>1)  factor.push_back(n);
    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (int i=0; i<factor.size() && ok; ++i)
            ok &= power(res, phi/factor[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}

int discreteLog(int a, int b, int M) {
    a %= M, b %= M;
    int k = 1, add = 0, g;
    while ((g = gcd(a, M)) > 1) {
        if (b == k)     return add;
        if (b%g)        return -1;
        b/=g, M/=g, ++add;
        k = (1LL*k*a/g)%M;
    }

    int RT = sqrt(M)+1, aRT = 1;
    for (int i=0; i<RT; i++) aRT = (aRT*1LL*a)%M;

    unordered_map<int, int> vals;
    for (int i=0, cur=b; i<=RT; i++) {
        vals[cur] = i;
        cur = (cur*1LL*a)%M;
    }

    for (int i=1, cur=k; i<=M/RT+1; i++) {
        cur = (cur*1LL*aRT)%M;
        if (vals.find(cur) != vals.end())    return RT*i-vals[
            cur]+add;
    }
    return -1;
}

int discreteRoot(int a, int b, int P) {
    if (b%P == 0)     return    a == 0 ? -1 : 0;
    int g = primitive_root(P);
    int y = discreteLog(power(g, a, P), b, P);
    return y == -1 ? -1 : power(g, y, P);
}

// Tonnelli Shanks: finds x st x^2 = a (mod p)
// p must be prime. returns -1 if none.
// complexity: O(log^2 p) worst case, O(log p) on average

int SQRT(int a, int p) {
  a = modit(a, p);
  if (a == 0) return 0;
  if (power(a, (p-1)/2, p) != 1)  return -1;
  if (p%4==3)           return power(a, (p+1)/4, p);
  int s = p-1, n = 2;
  int r = 0, m;
  while (s%2 == 0) ++r, s/=2;
  // find a non-square mod p
  while (power(n, (p-1)/2, p) != p-1) ++n;
  int x = power(a, (s+1)/2, p);
  int b = power(a, s, p), g = power(n, s, p);
  for (;; r = m) {
    int t = b;
    for (m = 0; m<r && t!=1; ++m) t = 1LL*t*t%p;
    if (m == 0) return x;
    int gs = power(g, 1LL << (r-m-1), p);
    g = 1LL*gs*gs%p;
    x = 1LL*x*gs%p;
    b = 1LL*b*g%p;
  }
}
```

5.2 Primality

FastEratosthenes.h

Description: Prime sieve for generating all primes smaller than LIM.
Time: LIM=1e9 ≈ 1.5s

6b2912, 20 lines

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

LinearSieve.h

Description: Linear sieve with least prime factor, 1e7

055a87, 14 lines

```
const int N = 1e8;
int lp[N];
vector<int> pr;

void pre() {
    for (int i=2; i<N; ++i) {
        if (lp[i] == 0) {
            lp[i] = i;
            pr.push_back (i);
        }
        for (int j=0; j<(int)pr.size() && pr[j]<=lp[i] && i*pr[
            j]<N; ++j)
            lp[i * pr[j]] = pr[j];
    }
}
```

SegmentedSieve.h

Description: Segmented sieve, well commented

91fe31, 20 lines

```
vector<char> segmentedSieve(long long L, long long R) {
    // generate all primes up to sqrt(R)
    long long lim = sqrt(R);
    vector<char> mark(lim + 1, false);
    vector<long long> primes;
    for (long long i = 2; i <= lim; ++i) {
        if (!mark[i]) {
            primes.emplace_back(i);
            for (long long j = i * i; j <= lim; j += i) mark[j]
                = true;
        }
    }

    vector<char> isPrime(R - L + 1, true);
    // 0 denotes L, 1 denotes L + 1, so on...
    for (long long i : primes)
        for (long long j = max(i * i, (L + i - 1) / i * i); j
            <= R; j += i)
            isPrime[j - L] = false;
    if (L == 1) isPrime[0] = false;
    return isPrime;
}
```

Factor.h

Description: Miller rabin + pollard rho, well commented

cd0d20, 53 lines

```
namespace Rho {
    ULL mult(ULL a, ULL b, ULL mod) {
        LL ret = a * b - mod * (ULL)(1.0L / mod * a * b);
        return ret + mod * (ret < 0) - mod * (ret >= (LL) mod);
    }

    ULL power(ULL x, ULL p, ULL mod){
        ULL s=1, m=x;
```

```
        while(p) {
            if(p&1) s = mult(s, m, mod);
            p>>=1;
            m = mult(m, m, mod);
        }
        return s;
    }

    vector<LL> bases = {2, 325, 9375, 28178, 450775, 9780504,
         1795265022};
    bool isprime(LL n) {
        if (n<2)    return 0;
        if (n%2==0)   return n==2;

        ULL s = __builtin_ctzll(n-1), d = n>>s;
        for (ULL x: bases) {
            ULL p = power(x%n, d, n), t = s;
            while (p!=1 && p!=n-1 && x%n && t--) p = mult(p, p,
                 n);
            if (p!=n-1 && t != s)               return 0;
        }
        return 1;
    }

    mt19937_64 rng(chrono::system_clock::now().time_since_epoch
         ().count());
    ULL FindFactor(ULL n) {
        if (n == 1 || isprime(n))    return n;
        ULL c = 1, x = 0, y = 0, t = 0, prod = 2, x0 = 1, q;
        auto f = [&](ULL X) { return mult(X, X, n) + c;};

        while (t++ % 128 or gcd(prod, n) == 1) {
            if (x == y) c = rng()%(n-1)+1, x = x0, y = f(x);
            if ((q = mult(prod, max(x, y) - min(x, y), n)))
                 prod = q;
            x = f(x), y = f(f(y));
        }
        return gcd(prod, n);
    }

    vector<ULL> factorize(ULL x) {
        if (x == 1)     return {};
        ULL a = FindFactor(x), b = x/a;
        if (a == x) return {a};
        vector<ULL> L = factorize(a), R = factorize(b);
        L.insert(L.end(), R.begin(), R.end());
        return L;
    }
}
```

Mobius.h

Description: Mobius sieve, tested

1529cc, 10 lines

```
int mob[N];
void mobius() {
  mob[1] = 1;
  for (int i = 2; i < N; i++){
    mob[i]--;
    for (int j = i + i; j < N; j += i) {
      mob[j] -= mob[i];
    }
  }
}
```

5.3 Divisibility

euclid.h

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

33ba8f, 5 lines

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

CRT.h

Description: Chinese Remainder Theorem.

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

Time: $\log(n)$

"euclid.h" 04d93a, 7 lines

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

5.3.1 Bézout's identity

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

$$ax + by = d$$

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

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

phiFunction.h

Description: *Euler's* ϕ function is defined as $\phi(n) :=$ # of positive integers $\le n$ that are coprime with n. $\phi(1) = 1$, p prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, m, n coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1} p_2^{k_2} ... p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1 - 1} ... (p_r - 1)p_r^{k_r - 1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$.
$\sum_{d|n} \phi(d) = n$, $\sum_{1 \le k \le n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$

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

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

cf7d6d, 8 lines

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

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

5.4 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

5.5 Primes

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

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

5.6 Estimates

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

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

5.7 Mobius Function

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

Mobius Inversion:

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

Other useful formulas/forms:

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

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

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

Combinatorial (6)

6.1 Permutations

6.1.1 Factorial

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

IntPerm.h

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

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

044568, 6 lines

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

6.1.2 Cycles

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

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

6.1.3 Derangements

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

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

6.1.4 Burnside's lemma

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

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

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

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

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

6.2 Partitions and subsets

6.2.1 Partition function

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

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

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

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

6.2.2 Lucas' Theorem

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

6.2.3 Binomials

multinomial.h

Description: Computes $\binom{k_1+\cdots+k_n}{k_1,k_2,\ldots,k_n} = \frac{(\sum k_i)!}{k_1!k_2!\ldots k_n!}$. a0a312, 5 lines

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

6.3 General purpose numbers

6.3.1 Bernoulli numbers

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

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

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

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

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

6.3.3 Eulerian numbers

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

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

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

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

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.5 Bell numbers

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

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

6.3.6 Labeled unrooted trees

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

6.3.7 Catalan numbers

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

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

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

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

Graph (7)

7.1 Fundamentals

Dijkstra.h
Description: Basic Dijkstra
Time: $\mathcal{O}(ElogV)$ e1776c, 41 lines

```
typedef long long ll;
struct Edge {
    int to;
    ll weight;
    bool operator>(const Edge& other) const {
        if (weight != other.weight) return weight > other.
            weight;
        return to > other.to;
    }
};

const ll INF = 2e15;
vector<vector<Edge>> adj;

void dijkstra(int source, vector<ll>& dist, vector<int>& parent
    ) {
    const int n = adj.size();
    dist.assign(n, INF);
    parent.assign(n, -1);
    vector<bool> visited(n + 2, false);

    // auto Compare = [&](const auto& a, const auto& b){
    //     return a.second > b.second;
    // };
    // priority_queue<pair<int, ll>, vector<pair<int, ll>>,
        decltype(Compare)> pq(Compare);

    dist[source] = 0;
    priority_queue<Edge, vector<Edge>, greater<Edge>> pq;
    pq.push({source, 0});
    while (!pq.empty()) {
        int from = pq.top().to;
        pq.pop();
```

```
        if (visited[from]) continue;
        visited[from] = true;
        for (auto& edge : adj[from]) {
            if (dist[edge.to] > dist[from] + edge.weight) {
                dist[edge.to] = dist[from] + edge.weight;
                parent[edge.to] = from;
                pq.push({edge.to, dist[edge.to]});
            }
        }
    }
}
```

BellmanFord.h

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

830a8f, 23 lines

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

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

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

FloydWarshall.h

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

531245, 12 lines

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

TopoSort.h

Description: Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than n – nodes reachable from cycles will not be returned.
Time: $\mathcal{O}(|V| + |E|)$

d678d8, 8 lines

```
vi topoSort(const vector<vi>& gr) {
  vi indeg(sz(gr)), q;
  for (auto& li : gr) for (int x : li) indeg[x]++;
  rep(i,0,sz(gr)) if (indeg[i] == 0) q.push_back(i);
  rep(j,0,sz(q)) for (int x : gr[q[j]])
    if (--indeg[x] == 0) q.push_back(x);
  return q;
}
```

7.2 Network flow

MinCostMaxFlow.h

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

58385b, 79 lines

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

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

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

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

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

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

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

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

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

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

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

Dinic.h

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

d7f0f1, 42 lines

```
struct Dinic {
  struct Edge {
    int to, rev;
    ll c, oc;
    ll flow() { return max(oc - c, 0LL); } // if you need flows
  };
  vi lvl, ptr, q;
  vector<vector<Edge>> adj;
  Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
  void addEdge(int a, int b, ll c, ll rcap = 0) {
    adj[a].push_back({b, sz(adj[b]), c, c});
    adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap});
  }
  ll dfs(int v, int t, ll f) {
    if (v == t || !f) return f;
    for (int& i = ptr[v]; i < sz(adj[v]); i++) {
      Edge& e = adj[v][i];
      if (lvl[e.to] == lvl[v] + 1)
        if (ll p = dfs(e.to, t, min(f, e.c))) {
          e.c -= p, adj[e.to][e.rev].c += p;
          return p;
        }
    }
    return 0;
  }
  ll calc(int s, int t) {
    ll flow = 0; q[0] = s;
    rep(L,0,31) do { // 'int L=30' maybe faster for random data
      lvl = ptr = vi(sz(q));
      int qi = 0, qe = lvl[s] = 1;
      while (qi < qe && !lvl[t]) {
        int v = q[qi++];
        for (Edge e : adj[v])
          if (!lvl[e.to] && e.c >> (30 - L))
            q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
      }
```

```
      while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
    } while (lvl[t]);
    return flow;
  }
  bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

MinCut.h

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

7.3 Matching

hopcroftKarp.h

Description: Fast bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and *btoa* should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.
Usage: `vi btoa(m, -1); hopcroftKarp(g, btoa);`
Time: $\mathcal{O}\left(\sqrt{V}E\right)$ f612e4, 42 lines

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

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

WeightedMatching.h

Description: Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and returns (min cost, match), where L[i] is matched with R[match[i]]. Negate costs for max cost. Requires $N \leq M$.
Time: $\mathcal{O}\left(N^2M\right)$ 1e0fe9, 31 lines

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

GeneralMatching.h

Description: Matching for general graphs. Fails with probability N/mod.
Time: $\mathcal{O}\left(N^3\right)$
"../numerical/MatrixInverse-mod.h" cb1912, 40 lines

```
vector<pii> generalMatching(int N, vector<pii>& ed) {
  vector<vector<ll>> mat(N, vector<ll>(N)), A;
  for (pii pa : ed) {
    int a = pa.first, b = pa.second, r = rand() % mod;
    mat[a][b] = r, mat[b][a] = (mod - r) % mod;
  }

  int r = matInv(A = mat), M = 2*N - r, fi, fj;
  assert(r % 2 == 0);

  if (M != N) do {
    mat.resize(M, vector<ll>(M));
    rep(i,0,N) {
      mat[i].resize(M);
      rep(j,N,M) {
        int r = rand() % mod;
        mat[i][j] = r, mat[j][i] = (mod - r) % mod;
      }
    }
  } while (matInv(A = mat) != M);

  vi has(M, 1); vector<pii> ret;
  rep(it,0,M/2) {
    rep(i,0,M) if (has[i])
      rep(j,i+1,M) if (A[i][j] && mat[i][j]) {
        fi = i; fj = j; goto done;
    } assert(0); done:
    if (fj < N) ret.emplace_back(fi, fj);
    has[fi] = has[fj] = 0;
    rep(sw,0,2) {
      ll a = modpow(A[fi][fj], mod-2);
      rep(i,0,M) if (has[i] && A[i][fj]) {
        ll b = A[i][fj] * a % mod;
        rep(j,0,M) A[i][j] = (A[i][j] - A[fi][j] * b) % mod;
      }
      swap(fi,fj);
    }
  }
  return ret;
}
```

7.4 DFS algorithms

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.
Usage: `scc(graph, [&](vi& v) { ... })` visits all components in reverse topological order. `comp[i]` holds the component index of a node (a component only has edges to components with lower index). `ncomps` will contain the number of components.
Time: $\mathcal{O}\left(E+V\right)$ 76b5c9, 24 lines

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

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

BiconnectedComponents.h

Description: Finds all biconnected components in an undirected graph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle.
Usage: `int eid = 0; ed.resize(N);`
`for each edge (a,b) {`
`ed[a].emplace_back(b, eid);`
`ed[b].emplace_back(a, eid++); }`
`bicomps([&](const vi& edgelist) {...});`
Time: $\mathcal{O}\left(E+V\right)$ c6b7c7, 32 lines

```
vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
```

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

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

2sat.h

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

5f9706, 56 lines

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

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

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

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

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

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

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

EulerWalk.h

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

780b64, 15 lines

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

Italiano.h

Description: Reachability in DAG
Time: $\mathcal{O}(N)$ Amortized per update

003f6e, 24 lines

```
struct Italiano {
    int n;
    vector<vector<int>> par;
    vector<vector<vector<int>>> child;
    Italiano(int n)
        : n(n), par(n, vector<int>(n, -1)), child(n, vector<
            vector<int>>(n)) {}
    bool is_reachable(int s, int t) { return s == t || par[s][t
        ] >= 0; }
    bool add_edge(int s, int t) {
        if (is_reachable(t, s)) return false;  // break DAG
            condition
        if (is_reachable(s, t)) return true;   // no-
            modification performed
        for (int p = 0; p < n; ++p) {
            if (is_reachable(p, s) && !is_reachable(p, t)) meld
                (p, t, s, t);
        }
        return 1;
    }
    void meld(int root, int sub, int u, int v) {
        par[root][v] = u;
        child[root][u].push_back(v);
        for (int c : child[sub][v]) {
            if (!is_reachable(root, c)) meld(root, sub, v, c);
        }
    }
};
// add edges one by one. if it breaks DAG law then print it
```

7.5 Trees

BinaryLifting.h

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

bfce85, 25 lines

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

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

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

LCA.h

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

"../data-structures/RMQ.h" 0f62fb, 21 lines

```
struct LCA {
  int T = 0;
  vi time, path, ret;
  RMQ<int> rmq;

  LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C,0,-1), ret)) {}
  void dfs(vector<vi>& C, int v, int par) {
    time[v] = T++;
    for (int y : C[v]) if (y != par) {
      path.push_back(v), ret.push_back(time[v]);
```

```
      dfs(C, y, v);
    }
  }

  int lca(int a, int b) {
    if (a == b) return a;
    tie(a, b) = minmax(time[a], time[b]);
    return path[rmq.query(a, b)];
  }
  //dist(a,b){return depth[a] + depth[b] - 2*depth[lca(a,b)];}
};
```

CompressTree.h

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

"LCA.h" 9775a0, 21 lines

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

HLD.h

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most log(n) light edges. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. VALS_EDGES being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0.
Time: $\mathcal{O}\left((\log N)^2\right)$

"../data-structures/LazySegmentTree.h" 03139d, 46 lines

```
template <bool VALS_EDGES> struct HLD {
  int N, tim = 0;
  vector<vi> adj;
  vi par, siz, rt, pos;
  Node *tree;
  HLD(vector<vi> adj_)
    : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
      rt(N),pos(N),tree(new Node(0, N)){ dfsSz(0); dfsHld(0); }
  void dfsSz(int v) {
    if (par[v] != -1) adj[v].erase(find(all(adj[v]), par[v]));
    for (int& u : adj[v]) {
      par[u] = v;
      dfsSz(u);
      siz[v] += siz[u];
      if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
    }
  }
  void dfsHld(int v) {
    pos[v] = tim++;
    for (int u : adj[v]) {
      rt[u] = (u == adj[v][0] ? rt[v] : u);
      dfsHld(u);
    }
  }
  template <class B> void process(int u, int v, B op) {
    for (; rt[u] != rt[v]; v = par[rt[v]]) {
      if (pos[rt[u]] > pos[rt[v]]) swap(u, v);
      op(pos[rt[v]], pos[v] + 1);
    }
    if (pos[u] > pos[v]) swap(u, v);
    op(pos[u] + VALS_EDGES, pos[v] + 1);
  }
  void modifyPath(int u, int v, int val) {
    process(u, v, [&](int l, int r) { tree->add(l, r, val); });
  }
  int queryPath(int u, int v) { // Modify depending on problem
    int res = -1e9;
    process(u, v, [&](int l, int r) {
        res = max(res, tree->query(l, r));
    });
    return res;
  }
  int querySubtree(int v) { // modifySubtree is similar
    return tree->query(pos[v] + VALS_EDGES, pos[v] + siz[v]);
  }
};
```

LinkCutTree.h

Description: Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.
Time: All operations take amortized $\mathcal{O}(\log N)$.

0fb462, 90 lines

```
struct Node { // Splay tree. Root's pp contains tree's parent.
  Node *p = 0, *pp = 0, *c[2];
  bool flip = 0;
  Node() { c[0] = c[1] = 0; fix(); }
  void fix() {
    if (c[0]) c[0]->p = this;
    if (c[1]) c[1]->p = this;
    // (+ update sum of subtree elements etc. if wanted)
  }
  void pushFlip() {
    if (!flip) return;
    flip = 0; swap(c[0], c[1]);
    if (c[0]) c[0]->flip ^= 1;
    if (c[1]) c[1]->flip ^= 1;
  }
  int up() { return p ? p->c[1] == this : -1; }
  void rot(int i, int b) {
    int h = i ^ b;
    Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
    if ((y->p = p)) p->c[up()] = y;
    c[i] = z->c[i ^ 1];
    if (b < 2) {
      x->c[h] = y->c[h ^ 1];
      y->c[h ^ 1] = x;
    }
    z->c[i ^ 1] = this;
    fix(); x->fix(); y->fix();
    if (p) p->fix();
    swap(pp, y->pp);
  }
  void splay() {
    for (pushFlip(); p; ) {
      if (p->p) p->p->pushFlip();
      p->pushFlip(); pushFlip();
      int c1 = up(), c2 = p->up();
      if (c2 == -1) p->rot(c1, 2);
      else p->p->rot(c2, c1 != c2);
    }
  }
  Node* first() {
    pushFlip();
    return c[0] ? c[0]->first() : (splay(), this);
  }
};

struct LinkCut {
  vector<Node> node;
  LinkCut(int N) : node(N) {}

  void link(int u, int v) { // add an edge (u, v)
    assert(!connected(u, v));
    makeRoot(&node[u]);
    node[u].pp = &node[v];
  }
  void cut(int u, int v) { // remove an edge (u, v)
    Node *x = &node[u], *top = &node[v];
    makeRoot(top); x->splay();
    assert(top == (x->pp ?: x->c[0]));
    if (x->pp) x->pp = 0;
    else {
      x->c[0] = top->p = 0;
      x->fix();
    }
  }
  bool connected(int u, int v) { // are u, v in the same tree?
    Node* nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
  }
  void makeRoot(Node* u) {
    access(u);
    u->splay();
    if(u->c[0]) {
      u->c[0]->p = 0;
      u->c[0]->flip ^= 1;
      u->c[0]->pp = u;
      u->c[0] = 0;
      u->fix();
    }
  }
  Node* access(Node* u) {
    u->splay();
    while (Node* pp = u->pp) {
      pp->splay(); u->pp = 0;
      if (pp->c[1]) {
        pp->c[1]->p = 0; pp->c[1]->pp = pp; }
      pp->c[1] = u; pp->fix(); u = pp;
    }
    return u;
  }
};
```

DSUonTree.h

Description: Sack/Small to large trick. Let st[v] dfs starting time of vertex v, ft[v] be it's finishing time and ver[time] is the vertex which it's starting time is equal to time.
Time: $\mathcal{O}(VlogV)$

26dcba, 21 lines

```
int cnt[maxn];
void dfs(int v, int p, bool keep){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p && sz[u] > mx)
```

```
        mx = sz[u], bigChild = u;
    for(auto u : g[v])
        if(u != p && u != bigChild)
            dfs(u, v, 0);  // run a dfs on small childs and
                 clear them from cnt
    if(bigChild != -1)
        dfs(bigChild, v, 1);  // bigChild marked as big and not
             cleared from cnt
    for(auto u : g[v])
  if(u != p && u != bigChild)
      for(int p = st[u]; p < ft[u]; p++)
    cnt[ col[ ver[p] ] ]++;
    cnt[ col[v] ]++;
    //now cnt[c] is the number of vertices in subtree of vertex
         v that has color c. You can answer the queries easily
        .
    if(keep == 0)
        for(int p = st[v]; p < ft[v]; p++)
      cnt[ col[ ver[p] ] ]--;
}
```

CentroidDecomp.h

Description: Centroid decomposition on nodes p[u] = parent of u in centroid tree
d[x][u] = distance from u to a parent of u at level x of centroid tree
if u is in subtree of centroid c, then d[lvl[c]][u] = dist(c, l)

Time: $\mathcal{O}(VlogV)$ d0a95f, 37 lines

```
const int maxn = 1e5 + 10;
vector<int> adj[maxn];
int lvl[maxn], sub[maxn], p[maxn], vis[maxn], d[18][maxn], ans[
     maxn];

void calc(int u, int par) { sub[u] = 1;
  for(int v : adj[u]) if(v - par && !vis[v])
    calc(v, u), sub[u] += sub[v];
}
int centroid(int u, int par, int r) {
  for(int v : adj[u]) if(v - par && !vis[v])
    if(sub[v] > r) return centroid(v, u, r);
  return u;
}
void dfs(int l, int u, int par) {
  if(par + 1) d[l][u] = d[l][par] + 1;
  for(int v : adj[u]) if(v - par && !vis[v])
    dfs(l, v, u);
}
void decompose(int u, int par) {
  calc(u, -1);
  int c = centroid(u, -1, sub[u] >> 1);
  vis[c] = 1, p[c] = par, lvl[c] = 0;
  if(par + 1) lvl[c] = lvl[par] + 1;
  dfs(lvl[c], c, -1);
  for(int v : adj[c]) if(v - par && !vis[v])
    decompose(v, c);
}
void update(int u) {
  for(int v = u; v + 1; v = p[v])
    ans[v] = min(ans[v], d[lvl[v]][u]);
}
int query(int u) {
  int ret = 1e9;
  for(int v = u; v + 1; v = p[v])
    ret = min(ret, ans[v] + d[lvl[v]][u]);
  return ret;
}
```

Kruskal.h

Description: Kruskal's algorithm with Union by Rank, rank + path compression

Time: $\mathcal{O}(MlogN)$ 6762da, 52 lines

```
vector<int> parent, rank;

void make_set(int v) {
  parent[v] = v;
  rank[v] = 0;
}

int find_set(int v) {
  if (v == parent[v])
    return v;
  return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b) {
  a = find_set(a);
  b = find_set(b);
  if (a != b) {
    if (rank[a] < rank[b])
      swap(a, b);
    parent[b] = a;
    if (rank[a] == rank[b])
      rank[a]++;
  }
}

struct Edge {
  int u, v, weight;
  bool operator<(Edge const& other) {
    return weight < other.weight;
  }
};

  int n;
  vector<Edge> edges;

  int cost = 0;
  vector<Edge> result;
  parent.resize(n);
  rank.resize(n);

  for (int i = 0; i < n; i++)
    make_set(i);

  sort(edges.begin(), edges.end());

  for (Edge e : edges) {
    if (find_set(e.u) != find_set(e.v)) {
      cost += e.weight;
      result.push_back(e);
      union_sets(e.u, e.v);
    }
  }
```

7.6 Math

7.6.1 Number of Spanning Trees

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

.....

7.6.2 Erdős–Gallai theorem

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

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

Geometry (8)

8.1 Geometric primitives

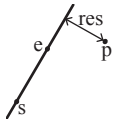
Point.h

Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.) 47ec0a, 28 lines

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

lineDistance.h

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

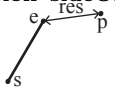


"Point.h" f6bf6b, 4 lines

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

SegmentDistance.h

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



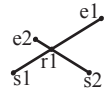
Usage:
```
Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;
```

"Point.h" 5c88f4, 6 lines

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

SegmentIntersection.h

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



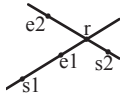
Usage:
```
vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;
```

"Point.h", "OnSegment.h" 9d57f2, 13 lines

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

lineIntersection.h

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



Usage:
```
auto res = lineInter(s1,e1,s2,e2);
if (res.first == 1)
cout << "intersection point at " << res.second << endl;
```

"Point.h" a01f81, 8 lines

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

sideOf.h

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

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

"Point.h" 3af81c, 9 lines

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

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

OnSegment.h

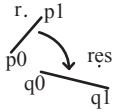
Description: Returns true iff p lies on the line segment from s to e. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

"Point.h" c597e8, 3 lines

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

linearTransformation.h

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



"Point.h" 03a306, 6 lines

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

LineProjectionReflection.h

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

"Point.h" b5562d, 5 lines

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

PolarSort.h

Description: Sort points, works even if three points are collinear

a8acc6, 10 lines

```
inline int quad (point p) {
  if (p.x < 0 and p.y < 0) return 0;
  if (p.x >= 0 and p.y < 0) return 1;
  if (p.x >= 0 and p.y >= 0) return 2;
  if (p.x < 0 and p.y >= 0) return 3;
}

sort(v.begin(), v.end(), [] (point a, point b) {
  return quad(a) == quad(b) ? a.x * b.y > a.y * b.x : quad(a) <
        quad(b);
});
```

Angle.h

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

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

0f0602, 35 lines

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

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

8.2 Circles

CircleIntersection.h

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

"Point.h" 84d6d3, 11 lines

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

CircleTangents.h

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

"Point.h" b0153d, 13 lines

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

CircleLine.h

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

"Point.h" e0cfba, 9 lines

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

CirclePolygonIntersection.h

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

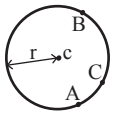
"../../content/geometry/Point.h" a1ee63, 19 lines

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

circumcircle.h

Description:

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



"Point.h" 1caa3a, 9 lines

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

MinimumEnclosingCircle.h

Description: Computes the minimum circle that encloses a set of points.
Time: expected $\mathcal{O}(n)$

"circumcircle.h" 09dd0a, 17 lines

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

8.3 Polygons

InsidePolygon.h

Description: Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.
Usage: `vector<P> v = {P{4,4}, P{1,2}, P{2,1}};`
`bool in = inPolygon(v, P{3, 3}, false);`
Time: $\mathcal{O}(n)$

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

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

PolygonArea.h

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

"Point.h" f12300, 6 lines

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

PolygonCenter.h

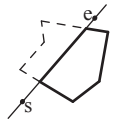
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"Point.h" 9706dc, 9 lines

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

PolygonCut.h

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



Usage: `vector<P> p = ...;`
`p = polygonCut(p, P(0,0), P(1,0));`

"Point.h", "lineIntersection.h" f2b7d4, 13 lines

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

PolygonUnion.h

Description: Calculates the area of the union of n polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)
Time: $\mathcal{O}\left(N^2\right)$, where N is the total number of points

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

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

```
    sort(all(segs));
    for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
    double sum = 0;
    int cnt = segs[0].second;
    rep(j,1,sz(segs)) {
      if (!cnt) sum += segs[j].first - segs[j - 1].first;
      cnt += segs[j].second;
    }
    ret += A.cross(B) * sum;
  }
  return ret / 2;
}
```

ConvexHull.h

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

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

"Point.h" 310954, 13 lines

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

HullDiameter.h

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

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

"Point.h" c571b8, 12 lines

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

PointInsideHull.h

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

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

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

```
typedef Point<ll> P;

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

LineHullIntersection.h

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

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

"Point.h" 7cf45b, 39 lines

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

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

PolygonTangent.h

Description: Calculates tangents to a polygon from an external point dmcp : sgn, getCross : cross

cc27d5, 59 lines

```
// Calculate [ACW, CW] tangent pair from an external point
#define CW    -1
#define ACW    1

int direction(Point st, Point ed, Point q) {
  return dcmp(getCross(ed - st, q - ed));
}

bool isGood(Point u, Point v, Point Q, int dir) {
  return direction(Q, u, v) != -dir;
}

Point better(Point u, Point v, Point Q, int dir) {
  return direction(Q, u, v) == dir ? u : v;
}

Point tangents(Point* pt, Point Q, int dir, int lo, int hi) {
  while (hi - lo > 1) {
    int mid = (lo + hi) / 2;
    bool pvs = isGood(pt[mid], pt[mid - 1], Q, dir);
    bool nxt = isGood(pt[mid], pt[mid + 1], Q, dir);

    if (pvs && nxt) return pt[mid];
    if (!(pvs || nxt)) {
      Point p1 = tangents(pt, Q, dir, mid + 1, hi);
      Point p2 = tangents(pt, Q, dir, lo, mid - 1);
      return better(p1, p2, Q, dir);
    }

    if (!pvs) {
      if (direction(Q, pt[mid], pt[lo]) == dir)
        hi = mid - 1;
      else if (better(pt[lo], pt[hi], Q, dir) == pt[lo])
        hi = mid - 1;
      else
        lo = mid + 1;
    }
    if (!nxt) {
      if (direction(Q, pt[mid], pt[lo]) == dir)
        lo = mid + 1;
      else if (better(pt[lo], pt[hi], Q, dir) == pt[lo])
        hi = mid - 1;
      else
        lo = mid + 1;
    }
  }

  Point ret = pt[lo];
  for (int i = lo + 1; i <= hi; i++)
    ret = better(ret, pt[i], Q, dir);
  return ret;
}

// [ACW, CW] Tangent
pair<Point, Point> get_tangents(Point* pt, int n, Point Q) {
  Point acw_tan = tangents(pt, Q, ACW, 0, n - 1);
  Point cw_tan = tangents(pt, Q, CW, 0, n - 1);
  return make_pair(acw_tan, cw_tan);
}
```

Star.h

Description: Struct for star area, less headache

f297d5, 14 lines

```
struct Star{
    int n;    // number of side of the star
    double r; // radius of the circum-circle
    Star(int n,double r) {this->n=n; this->r=r;}

    double getArea(){
        double theta=pi/n;
        double s=2*r*sin(theta);
        double R=0.5*s/tan(theta);
        double a=0.5*n*s*R;
        double a2=0.25*s*s/tan(1.5*theta);
        return a-n*a2;
    }
};
```

8.4 Misc. Point Set Problems

ClosestPair.h

Description: Finds the closest pair of points.
Time: $\mathcal{O}(n \log n)$

"Point.h" ac41a6, 17 lines

```
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set<P> S;
  sort(all(v), [](P a, P b) { return a.y < b.y; });
  pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
  int j = 0;
  for (P p : v) {
    P d{1 + (ll)sqrt(ret.first), 0};
    while (v[j].y <= p.y - d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
    for (; lo != hi; ++lo)
      ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
    S.insert(p);
  }
  return ret.second;
}
```

ManhattanMST.h

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

"Point.h" df6f59, 23 lines

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

kdTree.h

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

"Point.h" bac5b0, 63 lines

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

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

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

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

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

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

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

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

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

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

FastDelaunay.h

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

"Point.h" eefdf5, 88 lines

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

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

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

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

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

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

vector<P> triangulate(vector<P> pts) {
```

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

8.5 3D

PolyhedronVolume.h

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

3058c3, 6 lines

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

Point3D.h

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

8058ae, 32 lines

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

3dHull.h

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

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

"Point3D.h" 5b45fc, 49 lines

```
typedef Point3D<double> P3;

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

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

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

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

sphericalDistance.h

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

611f07, 8 lines

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

Strings (9)

KMP.h

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

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

d4375c, 16 lines

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

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

Zfunc.h

Description: z[i] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)

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

ee09e2, 12 lines

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

Manacher.h

Description: For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).

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

e7ad79, 13 lines

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

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.

Usage: `rotate(v.begin(), v.begin()+minRotation(v), v.end());`

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

d07a42, 8 lines

```
int minRotation(string s) {
```

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

SuffixArray.h

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

bc716b, 22 lines

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

SuffixTree.h

Description: Ukkonen's algorithm for online suffix tree construction. Each node contains indices [l, r) into the string, and a list of child nodes. Suffixes are given by traversals of this tree, joining [l, r) substrings. The root is 0 (has l = -1, r = 0), non-existent children are -1. To get a complete tree, append a dummy symbol – otherwise it may contain an incomplete path (still useful for substring matching, though).
Time: $\mathcal{O}(26N)$

aae0b8, 50 lines

```
struct SuffixTree {
  enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10
  int toi(char c) { return c - 'a'; }
  string a; // v = cur node, q = cur position
  int t[N][ALPHA],l[N],r[N],p[N],s[N],v=0,q=0,m=2;

  void ukkadd(int i, int c) { suff:
    if (r[v]<=q) {
      if (t[v][c]==-1) { t[v][c]=m;  l[m]=i;
        p[m++]=v; v=s[v]; q=r[v];  goto suff; }
      v=t[v][c]; q=l[v];
    }
    if (q==-1 || c==toi(a[q])) q++; else {
      l[m+1]=i;  p[m+1]=m;  l[m]=l[v];  r[m]=q;
      p[m]=p[v];  t[m][c]=m+1;  t[m][toi(a[q])]=v;
      l[v]=q;  p[v]=m;  t[p[m]][toi(a[l[m]])]=m;
      v=s[p[m]];  q=l[m];
      while (q<r[m]) { v=t[v][toi(a[q])];  q+=r[v]-l[v]; }
      if (q==r[m])  s[m]=v;  else s[m]=m+2;
      q=r[v]-(q-r[m]);  m+=2;  goto suff;
    }
  }

  SuffixTree(string a) : a(a) {
    fill(r,r+N,sz(a));
    memset(s, 0, sizeof s);
    memset(t, -1, sizeof t);
    fill(t[1],t[1]+ALPHA,0);
    s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;
    rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
  }

  // example: find longest common substring (uses ALPHA = 28)
  pii best;
  int lcs(int node, int i1, int i2, int olen) {
    if (l[node] <= i1 && i1 < r[node]) return 1;
    if (l[node] <= i2 && i2 < r[node]) return 2;
    int mask = 0, len = node ? olen + (r[node] - l[node]) : 0;
    rep(c,0,ALPHA) if (t[node][c] != -1)
      mask |= lcs(t[node][c], i1, i2, len);
    if (mask == 3)
      best = max(best, {len, r[node] - len});
    return mask;
  }
  static pii LCS(string s, string t) {
    SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));
    st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
    return st.best;
  }
};
```

SuffixArrayCPAlgo.h

Description: Suffix Array and lcp from cp-algo
Usage: If necessary
Time: $\mathcal{O}(nlogn)$

a367bb, 58 lines

```
vector<int> sort_cyclic_shifts(string const& s) {
  int n = s.size();
  const int alphabet = 256;
  vector<int> pn(n), cn(n);
  for (int h = 0; (1 << h) < n; ++h) {
    for (int i = 0; i < n; i++) {
      pn[i] = p[i] - (1 << h);
      if (pn[i] < 0)
        pn[i] += n;
    }
    fill(cnt.begin(), cnt.begin() + classes, 0);
    for (int i = 0; i < n; i++)
      cnt[c[pn[i]]]++;
    for (int i = 1; i < classes; i++)
      cnt[i] += cnt[i-1];
    for (int i = n-1; i >= 0; i--)
      p[--cnt[c[pn[i]]]] = pn[i];
    cn[p[0]] = 0;
    classes = 1;
    for (int i = 1; i < n; i++) {
      pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
      pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << h)) %
           n]};
      if (cur != prev)
        ++classes;
      cn[p[i]] = classes - 1;
    }
    c.swap(cn);
  }
  return p;
}
vector<int> suffix_array_construction(string s) {
  s += "$";
  vector<int> sorted_shifts = sort_cyclic_shifts(s);
  sorted_shifts.erase(sorted_shifts.begin());
  return sorted_shifts;
}
vector<int> lcp_construction(string const& s, vector<int> const
    & p) {
  int n = s.size();
  vector<int> rank(n, 0);
  for (int i = 0; i < n; i++)
    rank[p[i]] = i;

  int k = 0;
  vector<int> lcp(n-1, 0);
  for (int i = 0; i < n; i++) {
    if (rank[i] == n - 1) {
      k = 0;
      continue;
    }
    int j = p[rank[i] + 1];
    while (i + k < n && j + k < n && s[i+k] == s[j+k])
      k++;
    lcp[rank[i]] = k;
    if (k)
      k--;
  }
  return lcp;
}
```

Hashing-codeforces.h

Description: Various self-explanatory methods for string hashing. Use on Codeforces, which lacks 64-bit support and where solutions can be hacked.

<sys/time.h> 678247, 52 lines

```
typedef uint64_t ull;
static int C; // initialized below

// Arithmetic mod two primes and 2^32 simultaneously.
// "typedef uint64_t H;" instead if Thue-Morse does not apply.
template<int M, class B>
struct A {
  int x; B b; A(int x=0) : x(x), b(x) {}
  A(int x, B b) : x(x), b(b) {}
  A operator+(A o){int y = x+o.x; return{y - (y>=M)*M, b+o.b};}
  A operator-(A o){int y = x-o.x; return{y + (y< 0)*M, b-o.b};}
  A operator*(A o) { return {(int)(1LL*x*o.x % M), b*o.b}; }
  explicit operator ull() { return x ^ (ull) b << 21; }
  bool operator==(A o) const { return (ull)*this == (ull)o; }
  bool operator<(A o) const { return (ull)*this < (ull)o; }
};
typedef A<1000000007, A<1000000009, unsigned>> H;

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

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

```
    ret.push_back(h = h * C + str[i] - pw * str[i-length]);
  }
  return ret;
}

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

int main() {
  timeval tp;
  gettimeofday(&tp, 0);
  C = (int)tp.tv_usec; // (less than modulo)
  assert((ull)(H(1)*2+1-3) == 0);
  // ...
}
```

SuffixAutomaton.h

Description: Suffix automaton template from cp-algo
Time: $\mathcal{O}(n)$

646f9e, 41 lines

```
struct state {
    int len, link;
    map<char, int> next;
};
const int MAXLEN = 100000;
state st[MAXLEN * 2];
int sz, last;
void sa_init() {
    st[0].len = 0;
    st[0].link = -1;
    sz++;
    last = 0;
}
void sa_extend(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p = last;
    while (p != -1 && !st[p].next.count(c)) {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if (p == -1) {
        st[cur].link = 0;
    } else {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len) {
            st[cur].link = q;
        } else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            while (p != -1 && st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}
```

Tokenize.h

Description: Tokenizer wrt delimiter
Time: $\mathcal{O}(n)$

1e4c32, 30 lines

```
string trim(const string& str) {
  size_t first = str.find_first_not_of(" \t\n\r");
  size_t last = str.find_last_not_of(" \t\n\r");
  if (first == string::npos || last == string::npos)
    return "";
  return str.substr(first, (last - first + 1));
}
bool isDelimiter(char ch, const set<char>& delimiters) {
  return delimiters.find(ch) != delimiters.end();
}
vector<string> tokenize(const string& input, const string&
    delimiters) {
  set<char> delimiterSet(delimiters.begin(), delimiters.end());
  vector<string> tokens;
  string token;

  for (char ch : input) {
    if (isDelimiter(ch, delimiterSet)) {
      if (!token.empty()) {
        tokens.push_back(token);
        token.clear();
      }
      tokens.push_back(string(1, ch));
    } else {
      token += ch;
    }
  }
  if (!token.empty()) tokens.push_back(token);

  return tokens;
}
```

Lyndon.h

Description: A string is called simple (or a Lyndon word), if it is strictly smaller than any of its own nontrivial suffixes The Lyndon factorization of the string s is a factorization S = wi where wi are simple, and in non-increasing order
Time: $\mathcal{O}(n)$

0e6ce6, 18 lines

```
vector<string> duval(string const& s) {
    int n = s.size();
    int i = 0;
    vector<string> factorization;
    while (i < n) {
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j]) k = i;
            else k++;
            j++;
        }
        while (i <= k) {
            factorization.push_back(s.substr(i, j - k));
            i += j - k;
        }
    }
    return factorization;
}
```

Various (10)

10.1 Intervals

IntervalContainer.h

Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).
Time: $\mathcal{O}(\log N)$

edce47, 23 lines

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

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

IntervalCover.h

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

9e9d8d, 19 lines

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

ConstantIntervals.h

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

753a4c, 19 lines

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

```
  if (to <= from) return;
  int i = from; auto p = f(i), q = f(to-1);
  rec(from, to-1, f, g, i, p, q);
  g(i, to, q);
}
```

10.2 Misc. algorithms

TernaryDouble.h

Description: Ternary serach for finding minima/maxima of a unimodal continuous function
200-300 iterations are sufficient
Usage: `If necessary`
Time: $\mathcal{O}(logn)$

373e7a, 14 lines

```
double ternary_search(double l, double r) {
    double eps = 1e-9;          //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);  //evaluates the function at m1
        double f2 = f(m2);  //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);  //return the maximum of f(x) in [l, r]
}
```

TernarySearch.h

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

9155b4, 11 lines

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

LIS.h

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

2932a0, 17 lines

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

FastKnapsack.h

Description: Given N non-negative integer weights w and a non-negative target t, computes the maximum S <= t such that S is the sum of some subset of the weights.
Time: $\mathcal{O}(N \max(w_i))$

b20ccc, 16 lines

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

Hackenbush.h

Description: Green Hackenbush on undirected graph/tree Time : O(V + E)

5cb9d6, 42 lines

```
vector<int> g[N];
int n, T, low[N], dis[N];
int dfs (int u, int pre = 0) {
  dis[u] = low[u] = ++T;
  int ans = 0;
  for (auto v : g[u]) {
    if (v == pre) {
      pre += 2 * n;
      continue;
    }
    if (dis[v] == 0) {
      int res = dfs(v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] > dis[u])
        ans ^= (1 + res) ^ 1;
      else
        ans ^= res;
    } else low[u] = min(low[u], dis[v]);
  }
  if (pre > n) pre -= 2 * n;
  for (auto v : g[u]) if (v != pre && dis[u] <= dis[v]) ans ^=
      1;
  return ans;
}
int ground[N];
for(int i = 1; i <= n; i++) ground[i] = 0;
    int root = 1;
    ground[root] = 1;
    int ans = 0;
    T = 0;
    for (int i = 0; i < m; ++i) {
      int u, v;
      scanf("%d %d", &u, &v);
      if (ground[u]) u = root;
      if (ground[v]) v = root;
      int dummy;
      if (u == v) ans ^= 1;
      else {
        g[u].push_back(v);
        g[v].push_back(u);
      }
    }
    ans ^= dfs(root);
```

10.3 Dynamic programming

KnuthDP.h

Description: When doing DP on intervals: $a[i][j] = \min_{i<k<j}(a[i][k] + a[k][j]) + f(i,j)$, where the (minimal) optimal k increases with both i and j, one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$. This is known as Knuth DP. Sufficient criteria for this are if $f(b,c) \leq f(a,d)$ and $f(a,c) + f(b,d) \leq f(a,d) + f(b,c)$ for all $a \leq b \leq c \leq d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.
Time: $\mathcal{O}\left(N^2\right)$

DivideAndConquerDP.h

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

d38d2b, 18 lines

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

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

10.4 Optimization tricks

10.4.1 Bit hacks

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