Bangladesh Army University of Engineering and Technology

BAUET Twisted Minds

Md. Arik Rayhan, Md. Ohiduzaman Pranto, Ratul Hasan

IUT 11th National ICT Fest Programming Contest 2024

2024-04-26

Contents

Contest (1)

template.cpp

146 lines

```cpp
#pragma GCC optimization("O3")
#pragma GCC optimization("unroll-loops")

#include <bits/stdc++.h>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>

using namespace __gnu_pbds;
using namespace std;

const char nl = '\n';

typedef long long ll;
typedef unsigned long long ull;
typedef __uint128_t u128;
typedef long double ld;
typedef complex<ld> cd;

typedef pair<int, int> pi;
typedef pair<ll, ll> pl;
typedef pair<ld, ld> pd;

typedef vector<int> vi;
typedef vector<ld> vd;
typedef vector<ll> vl;
typedef vector<pi> vpi;
typedef vector<pl> vpl;
typedef vector<cd> vcd;

typedef map<int, int> mii;
typedef map<ll, ll> mll;
typedef map<ll, ll, greater<ll>> mllg;
typedef map<char, int> mci;
typedef map<string, int> msi;

typedef unordered_map<int, int> umii;
typedef unordered_map<ll, ll> umll;
typedef unordered_map<char, int> umci;
typedef unordered_map<string, int> umsi;

#define ins insert
#define mp make_pair
#define pb push_back
#define ff first
#define ss second
#define lb lower_bound
#define ub upper_bound
#define all(x) x.begin(), x.end()
#define rall(x) x.rbegin(), x.rend()
#define sz(x) (int)(x).size()

#define rep(i, a, b) for (int i = a; i < (b)
    ; i++)
#define per(i, a, b) for (int i = (b)-1; i
    >= a; i--)
#define gcd(a, b) __gcd(a, b)
#define lcm(a, b) (a * (b / gcd(a, b)))

#define deb(x) cout << #x << "=" << x <<
    endl
#define deb2(x, y) cout << #x << "=" << x <<
     "," << #y << "=" << y << endl
#define deb3(x, y, z) cout << #x << "=" << x
     << "," << #y << "=" << y << "," << #z
    << "=" << z << endl

ull rangesum(ll L, ll R) { return ((L + R) *
    (abs(R - L) + 1)) / 2; }
bool isPalindrome(string S)
{
  string P = S;
  reverse(P.begin(), P.end());
  return S == P ? true : false;
}
bool isPowerof(ll num, ll base) { return (
    num > 0 && num % base == 0) ? isPowerof(
    num / base, base) : num == 1; }
bool isPowerofTwo(ll num) { return (num > 0
    && (num & (num - 1)) == 0) ? true :
    false; }
int isSubstring(string main, string sub) {
    return main.find(sub) != string::npos ?
    main.find(sub) : -1; }

// 128 bit input output
__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; }

// Custom Hash
struct custom_hash
{
  static uint64_t splitmix64(uint64_t x)
  {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9
        ;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb
        ;
    return x ^ (x >> 31);
  }
  size_t operator()(uint64_t x) const
  {
    static const uint64_t FIXED_RANDOM =
        chrono::steady_clock::now().
        time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};

template <class T>
bool ckmin(T &a, const T &b) { return b < a
    ? a = b, 1 : 0; }
template <class T>
bool ckmax(T &a, const T &b) { return a < b
    ? a = b, 1 : 0; }

template <typename T>
using ordered_set = tree<T, null_type, less<
    T>,
             rb_tree_tag,
                 tree_order_statistics_node_update
                 >;
// *(o_set.find_by_order(val), o_set.
    order_of_key(val)

// cout<<"2^15 =                         32,768\
    n2^16 =                         65,536"<<
    endl;
// cout<<"2^31 =                  2,147,483,648\
    n2^32 =               4,294,967,296"<<
    endl;
// cout<<"2^63 =   9,223,372,036,854,775,808
    \n2^64 = 18,446,744,073,709,551,616"<<
    endl;

int32_t main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cout.tie(NULL);
  int T = 1;
  cin >> T;
  for (int tc = 1; tc <= T; tc++)
  {

  }
  return 0;
}
```

sublime.txt

5 lines

```
#ifdef ONLINEJUDGE
        clock_t tStart = clock();
        freopen("input.txt", "r", stdin);
        freopen("output.txt", "w", stdout);
    #endif
```

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 $x = -b/2a$.

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

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

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

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

2.4 Geometry

2.4.1 Triangles

Side lengths: a, b, c

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

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

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

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

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

Length of bisector (divides angles in two):

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

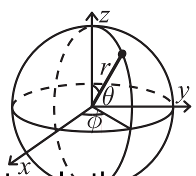
Law of sines: $\dfrac{\sin \alpha}{a} = \dfrac{\sin \beta}{b} = \dfrac{\sin \gamma}{c} = \dfrac{1}{2R}$

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

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

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

2.4.3 Spherical coordinates



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

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

2.5 Derivatives/Integrals

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

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) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ 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,

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

2.8.1 Discrete distributions

Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\mathrm{Bin}(n,p),\ n=1,2,\ldots,\ 0\leq p\leq 1$.

$$p(k) = \binom{n}{k}p^k(1-p)^{n-k}$$

$$\mu = np,\ \sigma^2 = np(1-p)$$

$\mathrm{Bin}(n,p)$ is approximately $\mathrm{Po}(np)$ for small p.

First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each which yields success with probability p is $\mathrm{Fs}(p),\ 0\leq p\leq 1$.

$$p(k) = p(1-p)^{k-1},\ k=1,2,\ldots$$

$$\mu = \frac{1}{p},\ \sigma^2 = \frac{1-p}{p^2}$$

Poisson distribution

The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\mathrm{Po}(\lambda),\ \lambda = t\kappa$.

$$p(k) = e^{-\lambda}\frac{\lambda^k}{k!}, k=0,1,2,\ldots$$

$$\mu = \lambda,\ \sigma^2 = \lambda$$

Data structures (3)

OrderStatisticTree.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change `null_type`.

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

782797, 16 lines

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

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

void example() {
  Tree<int> t, t2; t.insert(8);
  auto it = t.insert(10).first;
```

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

HashMap.h

Description: Hash map with mostly the same API as unordered_map, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

d77092, 7 lines

```
#include <bits/extc++.h>
// To use most bits rather than just the
   lowest ones:
struct chash { // large odd number for C
  const uint64_t C = ll(4e18 * acos(0)) |
     71;
  ll operator()(ll x) const { return
     __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h({}
   ,{},{},{},{1<<16});
```

SegmentTree.h

Description: Zero-indexed max-tree. Bounds are inclusive to the left and exclusive to the right. Can be changed by modifying T, f and unit.
Time: $\mathcal{O}(\log N)$

0f4bdb, 19 lines

```
struct Tree {
  typedef int T;
  static constexpr T unit = INT_MIN;
  T f(T a, T b) { return max(a, b); } // (
     any associative fn)
  vector<T> s; int n;
  Tree(int n = 0, T def = unit) : s(2*n, def
     ), n(n) {}
  void update(int pos, T val) {
    for (s[pos += n] = val; pos /= 2;)
      s[pos] = f(s[pos * 2], s[pos * 2 + 1])
         ;
  }
  T query(int b, int e) { // query [b, e)
    T ra = unit, rb = unit;
    for (b += n, e += n; b < e; b /= 2, e /=
       2) {
      if (b % 2) ra = f(ra, s[b++]);
      if (e % 2) rb = f(s[--e], rb);
    }
    return f(ra, rb);
  }
};
```

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;
  }
};
```

SubMatrix.h

Description: Calculate submatrix sums quickly, given upper-left and lower-right corners (half-open).
Usage: `SubMatrix<int> m(matrix);`
`m.sum(0, 0, 2, 2); // top left 4 elements`
Time: $\mathcal{O}\left(N^2 + Q\right)$

c59ada, 13 lines

```
template<class T>
struct SubMatrix {
  vector<vector<T>> p;
  SubMatrix(vector<vector<T>>& v) {
    int R = sz(v), C = sz(v[0]);
```

```
    p.assign(R+1, vector<T>(C+1));
    rep(r,0,R) rep(c,0,C)
      p[r+1][c+1] = v[r][c] + p[r][c+1] + p[
          r+1][c] - p[r][c];
  }
  T sum(int u, int l, int d, int r) {
    return p[d][r] - p[d][l] - p[u][r] + p[u
        ][l];
  }
};
```

Matrix.h

Description: Basic operations on square matrices.

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

c43c7d, 26 lines

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

LineContainer.h

Description: Container where you can add lines of the form kx+m, and query maximum values at points x. Useful for dynamic programming (“convex hull trick”).

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

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;
  }
};
```

Numerical (4)

4.1 Polynomials and recurrences

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;
}
```

BerlekampMassey.h

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

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

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

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

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

LinearRecurrence.h

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

f4e444, 26 lines

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

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

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

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

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

4.2 Optimization

GoldenSectionSearch.h

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

31d45b, 14 lines

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

```
      x2 = a + r*(b-a); f2 = f(x2);
    }
  return a;
}
```

4.3 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;
}
```

IntDeterminant.h

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

3313dc, 18 lines

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

SolveLinear.h

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

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)
}
```

MatrixInverse.h

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

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;
}
```

Tridiagonal.h

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

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

This is useful for solving problems on the type

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

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

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

Fails if the solution is not unique.

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

8f9fa8, 26 lines

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

Number theory (5)

5.1 Modular arithmetic

ModularArithmetic.h

Description: Operators for modular arithmetic. You need to set `mod` to some number first and then you can use the structure.

"euclid.h" 35bfea, 18 lines

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

ModInverse.h

Description: Pre-computation of modular inverses. Assumes LIM ≤ mod and that mod is a prime.

6f684f, 3 lines

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

ModPow.h

b83e45, 8 lines

```
const ll mod = 1000000007; // faster if
  const

ll modpow(ll b, ll e) {
  ll ans = 1;
  for (; e; b = b * b % mod, e /= 2)
    if (e & 1) ans = ans * b % mod;
  return ans;
}
```

ModLog.h

Description: Returns the smallest $x > 0$ s.t. $a^x = b \pmod m$, or -1 if no such x exists. modLog(a,1,m) can be used to calculate the order of a.

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

c040b8, 11 lines

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

ModSum.h

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

Time: $\log(m)$, with a large constant.

5c5bc5, 16 lines

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

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

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

ModMulLL.h

Description: Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for $0 \le a, b \le c \le 7.2 \cdot 10^{18}$.

Time: $\mathcal{O}(1)$ for modmul, $\mathcal{O}(\log b)$ for modpow

bbbd8f, 11 lines

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

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution).

Time: $\mathcal{O}(\log^2 p)$ worst case, $\mathcal{O}(\log p)$ for most p

"ModPow.h" 19a793, 24 lines

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

```
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

5.2 Primality

FastEratosthenes.h

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

6b2912, 20 lines

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

MillerRabin.h

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

"ModMulLL.h" 60dcd1, 12 lines

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

Factor.h

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

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

```
ull pollard(ull n) {
  auto f = [n](ull x) { return modmul(x, x, n) + 1; };
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  while (t++ % 40 || __gcd(prd, n) == 1) {
    if (x == y) x = ++i, y = f(x);
    if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

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;
}
```

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 $n = p_1^{k_1} p_2^{k_2} \ldots p_r^{k_r}$ then $\phi(n) = (p_1-1)p_1^{k_1-1} \ldots (p_r-1)p_r^{k_r-1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$.
$\sum_{d|n} \phi(d) = n$, $\sum_{1 \le k \le n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$
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 Fractions

ContinuedFractions.h

Description: Given N and a real number $x \geq 0$, finds the closest rational approximation p/q with $p, q \leq N$. It will obey $|p/q - x| \leq 1/qN$.
For consecutive convergents, $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$. ($p_k/q_k$ alternates between $> x$ and $< x$.) If x is rational, y eventually becomes ∞; if x is the root of a degree 2 polynomial the a's eventually become cyclic.
Time: $\mathcal{O}(\log N)$

dd6c5e, 21 lines

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

FracBinarySearch.h

Description: Given f and N, finds the smallest fraction $p/q \in [0, 1]$ such that $f(p/q)$ is true, and $p, q \leq N$. You may want to throw an exception from f if it finds an exact solution, in which case N can be removed.
Usage: `fracBS([](Frac f) { return f.p>=3*f.q; }, 10); // {1,3}`
Time: $\mathcal{O}(\log(N))$

27ab3e, 25 lines

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

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

5.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

5.6 Primes

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

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

5.7 Estimates

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

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

Combinatorial (6)

6.1.2 Binomials

multinomial.h

Description: Computes $\binom{k_1 + \dots + k_n}{k_1, k_2, \dots, k_n} = \frac{(\sum k_i)!}{k_1!k_2!...k_n!}$.

a0a312, 6 lines

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

Geometry (7)

7.1 Geometric primitives

Point.h

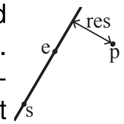
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

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.

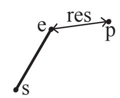


"Point.h" f6bf6b, 4 lines

```
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.



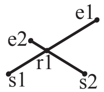
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

"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.



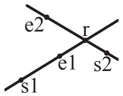
Usage:
```
vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;
```

"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 ⇔ left/on line/right. If the optional argument eps is given 0 is returned if p is within distance eps from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.

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

"Point.h" 3af81c, 9 lines

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

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

OnSegment.h

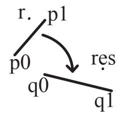
Description: Returns true iff p lies on the line segment from s to e. Use `(segDist(s,e,p)<=epsilon)` instead when using Point<double>.

"Point.h" c597e8, 3 lines

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

linearTransformation.h

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



"Point.h" 03a306, 6 lines

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

Angle.h

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

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

0f0602, 35 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
```

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

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

7.2 Circles

CircleIntersection.h

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

"Point.h" 84d6d3, 11 lines

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

CircleTangents.h

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

"Point.h" b0153d, 13 lines

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

CirclePolygonIntersection.h

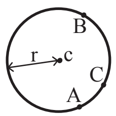
Description: Returns the area of the intersection of a circle with a ccw polygon.
Time: $\mathcal{O}(n)$

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

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

circumcircle.h

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



"Point.h" 1caa3a, 9 lines

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

MinimumEnclosingCircle.h

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

"circumcircle.h" 09dd0a, 17 lines

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

```
}
```

7.3 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

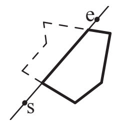
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"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;
}
```

Strings (8)

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

ee09e2, 12 lines

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

Manacher.h

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

e7ad79, 13 lines

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

MinRotation.h

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

d07a42, 8 lines

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

SuffixArray.h

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

38db9f, 23 lines

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

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;
}
};
```

Hashing.h

Description: Self-explanatory methods for string hashing.

2d2a67, 44 lines

```
// Arithmetic mod 2^64-1. 2x slower than mod
   2^64 and more
// code, but works on evil test data (e.g.
   Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash
   the same mod 2^64).
// "typedef ull H;" instead if you think
   test data is random,
// or work mod 10^9+7 if the Birthday
   paradox is not a problem.
typedef uint64_t ull;
struct H {
  ull x; H(ull x=0) : x(x) {}
  H operator+(H o) { return x + o.x + (x + o
     .x < x); }
  H operator-(H o) { return *this + ~o.x; }
  H operator*(H o) { auto m = (__uint128_t)x
     * o.x;
    return H((ull)m) + (ull)(m >> 64); }
  ull get() const { return x + !~x; }
  bool operator==(H o) const { return get()
     == o.get(); }
  bool operator<(H o) const { return get() <
      o.get(); }
};
static const H C = (ll)1e11+3; // (order ~ 3
   e9; random also ok)

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

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

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

AhoCorasick.h

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

Time: construction takes $\mathcal{O}(26N)$, where N = sum of length of patterns. find(x) is $\mathcal{O}(N)$, where N = length of x. findAll is $\mathcal{O}(NM)$.

f35677, 66 lines

```
struct AhoCorasick {
```

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

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

Various (9)

9.1 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);
}
```

9.2 Misc. algorithms

TernarySearch.h

Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \dots < f(i) \geq \dots \geq f(b)$. To reverse which of the sides allows non-strict inequalities, change the $<$ marked with (A) to <=, and reverse the loop at (B). To minimize f, change it to $>$, also at (B).

Usage: `int ind = ternSearch(0,n-1,[&](int i){return a[i];});`

Time: $\mathcal{O}(\log(b-a))$

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;
}
```

9.3 Dynamic programming

KnuthDP.h

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

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

DivideAndConquerDP.h

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

d38d2b, 18 lines

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

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

9.4 Debugging tricks

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

9.5 Optimization tricks

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

9.5.1 Bit hacks

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

9.5.2 Pragmas

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

FastMod.h

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

751a02, 8 lines

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

FastInput.h

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

7b3c70, 17 lines

```
inline char gc() { // like getchar()
  static char buf[1 << 16];
  static size_t bc, be;
  if (bc >= be) {
    buf[0] = 0, bc = 0;
    be = fread(buf, 1, sizeof(buf), stdin);
  }
  return buf[bc++]; // returns 0 on EOF
}

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

BumpAllocator.h

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

745db2, 8 lines

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

SmallPtr.h

Description: A 32-bit pointer that points into BumpAllocator memory.

"BumpAllocator.h" 2dd6c9, 10 lines

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

BumpAllocatorSTL.h

Description: BumpAllocator for STL containers.
Usage: vector<vector<int, small<int>>> ed(N);

bb66d4, 14 lines

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

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

SIMD.h

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

551b82, 43 lines

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

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

// High-level/specific methods:
// load(u)?_si256, store(u)?_si256,
    setzero_si256, _mm_malloc
// blendv_(epi8|ps|pd) (z?y:x),
    movemask_epi8 (hibits of bytes)
// i32gather_epi32(addr, x, 4): map addr[]
    over 32-b parts of x
// sad_epu8: sum of absolute differences of
    u8, outputs 4xi64
// maddubs_epi16: dot product of unsigned i7
    's, outputs 16xi15
// madd_epi16: dot product of signed i16's,
    outputs 8xi32
// extractf128_si256(, i) (256->128),
    cvtsi128_si32 (128->lo32)
// permute2f128_si256(x,x,1) swaps 128-bit
    lanes
// shuffle_epi32(x, 3*64+2*16+1*4+0) == x
    for each lane
// shuffle_epi8(x, y) takes a vector instead
     of an imm

// Methods that work with most data types (
    append e.g. _epi32):
// set1, blend (i8?x:y), add, adds (sat.),
    mullo, sub, and/or,
// andnot, abs, min, max, sign(1,x), cmp(gt|
    eq), unpack(lo|hi)

int sumi32(mi m) { union {int v[8]; mi m;} u
    ; u.m = m;
  int ret = 0; rep(i,0,8) ret += u.v[i];
      return ret; }
mi zero() { return _mm256_setzero_si256(); }
mi one() { return _mm256_set1_epi32(-1); }
bool all_zero(mi m) { return
    _mm256_testz_si256(m, m); }
bool all_one(mi m) { return
    _mm256_testc_si256(m, one()); }

ll example_filteredDotProduct(int n, short*
    a, short* b) {
  int i = 0; ll r = 0;
  mi zero = _mm256_setzero_si256(), acc =
      zero;
  while (i + 16 <= n) {
    mi va = L(a[i]), vb = L(b[i]); i += 16;
    va = _mm256_and_si256(_mm256_cmpgt_epi16
        (vb, va), va);
    mi vp = _mm256_madd_epi16(va, vb);
    acc = _mm256_add_epi64(
        _mm256_unpacklo_epi32(vp, zero),
      _mm256_add_epi64(acc,
          _mm256_unpackhi_epi32(vp, zero)));
  }
  union {ll v[4]; mi m;} u; u.m = acc; rep(i
      ,0,4) r += u.v[i];
  for (;i<n;++i) if (a[i] < b[i]) r += a[i]*
      b[i]; // <- equiv
  return r;
}
```

Our Snippets (10)

10.1 Md. Arik Rayhan

arik.cpp

185 lines

```
// Bitwise Sieve
const int pmxsz = 100000000;
int status[(pmxsz / 32) + 2];
int prime[5761455 + 5], noofprime = 0;
```

```
inline bool Bit_Check(int N, int pos) {
   return (bool)(N & (1 << pos)); }
inline int Bit_Set(int N, int pos) { return
   N = N | (1 << pos); }
inline bool PrimeCheck(int i) { return 1 ^ (
   bool)(Bit_Check(status[i >> 5], i & 31))
   ; }
inline void PrimeSet(int i) { status[i >> 5]
    = Bit_Set(status[i >> 5], i & 31); }
inline void Mark(int i, int N)
{
    for (int j = i * i; j <= N; j += (i <<
       1))
    {
        PrimeSet(j);
    }
}
void sieve(int N = 100000000)
{
    int i, j, sqrtN;
    sqrtN = int(sqrt(N));
    for (i = 5; i <= sqrtN; i += 6)
    {
        if (PrimeCheck(i))
        {
            Mark(i, N);
        }
        if (PrimeCheck(i + 2))
        {
            Mark(i + 2, N);
        }
    }
    prime[noofprime++] = 2;
    prime[noofprime++] = 3;
    for (i = 5; i <= N; i += 6)
    {
        if (PrimeCheck(i))
        {
            prime[noofprime++] = i;
        }
        if (PrimeCheck(i + 2))
        {
            prime[noofprime++] = i + 2;
        }
    }
}
```

```
// Single Prime Check using Miller Rabin
ull binpower(ull base, ull e, ull mod)
{
    ull result = 1;
    base %= mod;
    while (e)
    {
        if (e & 1)
            result = (u128)result * base %
               mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}
bool check_composite(ull n, ull a, ull d,
   int s)
{
    ull x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++)
    {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};
bool MillerRabin(ull n)
{
    if (n < 2)
        return false;
    int r = 0;
    ull d = n - 1;
    while ((d & 1) == 0)
    {
        d >>= 1;
        r++;
    }
    for (int a : {2, 3, 5, 7, 11, 13, 17,
       19, 23, 29, 31, 37})
    {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

```
// String Hashing
long long compute_hash(string const &s)
{
    const int p = 31;
    const int m = 1e9 + 9;
    long long hash_value = 0;
    long long p_pow = 1;
    for (char c : s)
    {
        hash_value = (hash_value + (c - 'a'
           + 1) * p_pow) % m;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}
```

```
// Trinary Search
double f(double x)
{
    // return some value
}
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]
}
```

```
}

// SPF using Sieve 10^6 in 280ms & 42MB
const int MAXN = 10e6 + 5;
int spf[MAXN];
vector<int> factor[MAXN];
inline vector<int> getFactorization(int x)
{
    vector<int> ret;
    while (x != 1)
    {
        ret.push_back(spf[x]);
        x = x / spf[x];
    }
    return ret;
}
void sievefactor()
{
    spf[1] = 1;
    for (int i = 2; i <= MAXN; i++)
    {
        spf[i] = i;
    }
    for (int i = 4; i <= MAXN; i += 2)
    {
        spf[i] = 2;
    }
    for (int i = 3; i * i < MAXN; i++)
    {
        if (spf[i] == i)
        {
            for (int j = i * i; j < MAXN; j
                += i)
                if (spf[j] == j)
                {
                    spf[j] = i;
                }
        }
    }
    for (int i = 1; i <= MAXN; i++)
    {
        factor[i] = getFactorization(i);
    }
}

// number conversion
long long n = stoll(str, nullptr, base);
ans = to_string(n);
string binary = bitset<64>(n).to_string();
stringstream ss;
ss << std::oct << n;
string octal = ss.str();
ans = octal;
stringstream ss;
ss << std::hex << n;
string hexa = ss.str();
transform(hexa.begin(), hexa.end(), hexa.
    begin(), ::toupper);
ans = hexa;
```

10.2 Ratul Hasan

ratul.cpp

233 lines

```
// FASTIO
import sys
ONLINE_JUDGE = __debug__
if ONLINE_JUDGE:
    import io,os
    input = io.BytesIO(os.read(0,os.fstat(0)
        .st_size)).readline

// binary to demical
x = '1000'
y = int(x, 2)
print(y)
// decimal to binary
n = 100
binary = format(n, 'b')
print(binary)

// 2D array
rows, cols = (5, 5)
arr = [[0]*cols]*rows
matrix = []
print("Enter the entries rowwise:")
R, C = map(int, input().split())
# matrix = [[int(input()) for x in range (C)
    ] for y in range(R)]
matrix = []
for i in range(R):
    array = list(map(int, input().split()))
    matrix.append(array)
# For printing the matrix
for i in range(R):
    for j in range(C):
        print(matrix[i][j], end = " ")
    print()
// sorting
array.sort()
array.sort(reverse=True)

a, b, c = map(int, input().split())
array = list(map(int, input().split()))
array = []
array.append(x,y/x)
from collections import defaultdict
Hash = defaultdict(int)
dp = [-1] * (n + 1)
specificRange = list(range(n + 1))
my_set = set()
my_set.add(value)
x = pow(a, b, c) //( a * a * a) % c)
x = a ** b

// check_if_string_is_a_subseq
    string a, b;
    cin >> a >> b;
    int n = a.size(), m = b.size();
    int dp[n + 1][m + 1];
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= m; j++) {
            if (i == 0 || j == 0) dp[i][j] =
                0;
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (a[i - 1] == b[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1]
                    + 1;
            } else {
                dp[i][j] = max(dp[i - 1][j],
                    dp[i][j - 1]);
            }
        }
    }
    if (dp[n][m] == a.size()......
```

```
// lcs
 string a, b;
    cin >> a >> b;
    int n = a.size(), m = b.size();
    int dp[n + 1][m + 1];
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= m; j++) {
            if (i == 0 || j == 0) dp[i][j] =
                0;
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (a[i - 1] == b[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1]
                    + 1;
            } else {
                dp[i][j] = max(dp[i - 1][j],
                     dp[i][j - 1]);
            }
        }
    }
    cout << dp[n][m] << endl;
    // minimum insertion and deletion to
       make b from a --> delete =  a.size()
        - dp[n][m]  insert = b.size() - dp[
       n][m]
    // print the lcs
    int i = n, j = m;
    string ans;
    while (i != 0 && j != 0) {
        if (a[i - 1] == b[j - 1]) {
            ans += a[i - 1];
            i--;
            j--;
        } else {
            if (dp[i][j - 1] > dp[i - 1][j])
                j--;
            else i--;
        }
    }
    reverse(ans.begin(), ans.end());

// lps
string a, b;
    cin >> a >> b;
    int n = a.size(), m = b.size();
    int dp[n + 1][m + 1];
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= m; j++) {
            if (i == 0 || j == 0) dp[i][j] =
                0;
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (a[i - 1] == b[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1]
                    + 1;
            } else {
                dp[i][j] = 0;
            }
        }
    }
    int mx = 0;
    int ci, cj;
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= m; j++) {
            if (dp[i][j] > mx) {
                mx = dp[i][j];
                ci = i;
                cj = j;
            }
        }
    }
    string ans;
    while (ci != 0 && cj != 0) {
        if (a[ci - 1] == b[cj - 1]) {
            ans += a[ci - 1];
            ci--;
            cj--;
        } else {
            break;
        }
    }
    reverse(ans.begin(), ans.end());

// lps
string a;
    cin >> a;
    int n = a.size();
    string b = a;
    reverse(b.begin(), b.end());
    int m = b.size();

    int dp[n + 1][m + 1];
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= m; j++) {
            if (i == 0 || j == 0) dp[i][j] =
                0;
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (a[i - 1] == b[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1]
                    + 1;
            } else {
                dp[i][j] = max(dp[i - 1][j],
                     dp[i][j - 1]);
            }
        }
    }
    // minimum deletion and insertion to
       make palindrome--> delete = b - dp[n
       ][m]   insert =  b - dp[n][m]
    int i = n, j = m;
    string ans;
    while (i != 0 && j != 0) {
        if (a[i - 1] == b[j - 1]) {
            ans += a[i - 1];
            i--;
            j--;
        } else {
            if (dp[i][j - 1] > dp[i - 1][j])
                j--;
            else i--;
        }
    }
    reverse(ans.begin(), ans.end());
// shortest common supersequence
string a, b;
    cin >> a >> b;
    int n = a.size();
    int m = b.size();
    int dp[n + 1][m + 1];
    for (int i = 0; i <= n; i++) {
```

```
        for (int j = 0; j <= m; j++) {
            if (i == 0 || j == 0) dp[i][j] =
                0;
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (a[i - 1] == b[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1]
                    + 1;
            } else {
                dp[i][j] = max(dp[i - 1][j],
                    dp[i][j - 1]);
            }
        }
    }
    cout << n + m - dp[n][m] << endl; // scs
        size
    // print section
    int i = n, j = m;
    string ans;
    while (i != 0 && j != 0) {
        if (a[i - 1] == b[j - 1]) {
            ans += a[i - 1];
            i--;
            j--;
        } else if (dp[i - 1][j] > dp[i][j -
            1]) {
            ans += a[i - 1];
            i--;
        } else {
            ans += b[j - 1];
            j--;
        }
    }
    while (i != 0) {
        ans += a[i - 1];
        i--;
    }
    while (j != 0) {
        ans += b[j - 1];
        j--;
    }
    reverse(ans.begin(), ans.end());
```

10.3 Md. Ohiduzaman Pranto

pranto.cpp

291 lines

```
//  A ^ 0 = A
//  A ^ A = 0
//  If A ^ B =C, then A ^ C =B
//  A ^ B ^ B = A
//  A & B <= min (A, B)
//  A | B >= max (A, B)
//  (A | B) + (A & B) = A + B
//  (A & 1) is 1 if A is odd, else 0
//  A & (A-1) is 0 if A is a power of 2 (
    except when A = 0)

// a ^ a = 0
// a ^ 0 = a
// a ^ b = 0 ==> a = b
// a ^ b = b ^ a
// (a ^ b) ^ c = a ^ (b ^ c)
// a ^ b ^ a = (a ^a) ^ b = 0 ^ b = b
// a ^ a ^..... ^ a = 0 (even number of a's)
// a ^ a ^..... ^ a = a (odd number of a's)
// a ^ b = c ==> a = b ^ c ==> a ^ b ^ c = 0

Left shift (a<<b = a*2^b)
Right shift (a>>b = a/2^b)
Bitwise AND (a&b)
Bitwise OR (a|b)
Bitwise XOR (a^b)
Bitwise NOT (~a = -a-1)
For all odd numbers the last bit is 1, and
    for even its 0
Odd/Even (n&1)? cout << "Odd" : cout << "
    Even";

Some properties of bitwise operations:
1. a|b=a^b+a&b
2. a^(a&b)=(a|b)^b
3. b^(a&b)=(a|b)^a
4. (a&b)^(a|b)=a^b

Addition:
1. a+b=a|b+a&b
2. a+b=a^b+2(a&b)

Subtraction:
1. a-b=(a^(a&b))-((a|b)^a)
2. a-b=((a|b)^b)-((a|b)^a)
3. a-b=(a^(a&b))-(b^(a&b))
4. a-b=((a|b)^b)-(b^(a&b))

// some bit operations

(n>>k)&1 -> kth bit on or off // needs
    modification
n|(a<<k) -> kth bit on
n&((1<<30)-1-(1<<k)) -> kth bit off
n&((1<<k)-1) -> last k bits on
__builtin_popcount(x) // number of set bits
__builtin_clz(x) // number of leading zeros
__builtin_ctz(x) // number of trailing zeros

int get_ith_bit(int n, int i)
{
    int mask = (1<<i);
    return (n&mask)>0?1:0;
}

int clear_ith_bit(int n, int i)
{
    int mask = ~(1<<i);
    n = (n&mask);
    return n;
}

int set_ith_bit(int n, int i)
{
    int mask = (1<<i);
    n=(n|mask);
    return n;
}

int update_ith_bit(int n, int i, int v)
{
    clearIthBit(n,i);
    int mask = (v<<i);
    n=(n|mask);
    return n;
}
```

```
int clear_last_i_bit(int n, int i)
{
     int mask = (-1<<i);
     n = (n&mask);
     return n;
}

int clear_bits_in_range(int n, int i, int j)
{
     int a = (-1<<j+1);
     int b = (i<<i-1);
     int mask = (a|b);
     n = (n&mask);
     return n;
}

int replace_bits_in_range(int n, int v, int
   i, int j)
{
     n = clear_bits_in_range(n,i,j);
     int mask = (v<<i);
     cout << (n|mask);
}

int count_set_bits(int n)
{
     int cont=0;
     while(n>0)
     {
         int last_bit = (n&1);
         cont+=last_bit;
         n = n>>1;
}
     return cont;
}

(max of max) or (min of min) is BINARY
   SEARCH you dumb STUPID fuck

find the position of something on a string
a = find(s.begin(), s.end() , '3') - s.begin
   () ;// we are finding the position of 3
   in this case

//Vector
vector<int> v(n); // we take a array of
    vector with fixed length
cin>>v[i];
v.sort(v.begin(), v.end());
v.begin() is inclusive (eta soho sort hobe)
And v.end() is exclusive (eta sara sort hobe
   )
always point the position after the last
   position of the vector -> 1,2,3,4[v.end
   ()]
cout<< (int)v.size() << nl; //v.size() e
    typecasting must

max element of a vector
*max_element(v.begin(), v.end())
removing a element (x) from vector
v.erase(find(v.bigin() , v.end () , x))

//map<pair<int,int>,string > m ;
cin can be done like this -> m[{x,y}]=s;
        int x , y ; cin>> x >> y ;
        pair<int , int> xx ;
        xx= make_pair(x,y);
        auto it= m.find(xx);
        if(it!=m.end())
        {
            cout<< (*it).second << nl;
        }

In map
cost += m[x]; // it will work but its not a
   good practice
because if x dosent exists then there will
   be a extra value named x inserted into
   the map
size will increase.
if( m.find(x) != m.end() ) // Good practice
{ cost += m[x]; }

//Stringstream
string s;
  getline(cin, s);
  stringstream ss;
  ss << s;
  string word;
  while (ss >> word) {
    cout << word << '\n';}

//find anything on a string
(find(s.begin() , s.end() , ' ') != s.end())
        {

        }

//check if all the char of the string is
   same
if (unique(s.begin(), s.end()) == s.begin()
   + 1 )
    {
        cout << " all are same bro "<< nl;
    }

//input string after int
problem first string dosent input in
cin.ignore();
getline(cin,s);

//string to int
int x = stoi(s); //string to int
ll x = stoll(s) ; //string to long long

//substring
s= "pranto" ; //indexing starts with 0 as
   usual
ans = s.substr(1,3); starts taking substring
    from 1 and takes 3 char from there
ans = "ran"
s.substr(1); starts taking substring from
   and 1 to the end
ans = "ranto"

// Print the number of times the character '
   e' appears in the string.
  cout << std::count(str.begin(), str.end(),
      ch) << endl;

/*Rearranges the elements in the range [
    first,last) in
  to the next lexicographically greater
      permutation.  */
void print(int a[], int n);
int main()
```

```
{
  int myints[] = {1, 4, 3}, n = 3;

  sort(myints, myints + n);
  /*sort to see all the combinations of
     lower to higher
  dont sort if you want to see only the next
      higher permutation*/

  cout << "The n! possible permutations with
      3 elements:\n";
  /*first it will print the original array
     then
    it will print the next higher
       permutation  */
  do
  {
    print(myints, n);
  } while (next_permutation(myints, myints +
      n));

  cout << "After loop: ";
  print(myints, n); // sorted array asending
      order

  return 0;
}

/*Rearranges the elements in the range [
   first,last) in
  to the next lexicographically lower
     permutation.  */
int main()
{
  int myints[] = {1, 4, 3}, n = 3;

  sort(myints, myints + n, greater<int>());
  /*sort to see all the combinations of
     higher to lower
  dont sort if you want to see only the next
      lower permutation*/

  cout << "The n! possible permutations with
      3 elements:\n";
  /*first it will print the original array
     then
    it will print the next lower permutation
          */
  do
  {
    print(myints, n);
  } while (prev_permutation(myints, myints +
     n));

  cout << "After loop: ";

  return 0;
}
```

```
Legendres formula
n! is multiplication of {1, 2, 3, 4,....n}.
How many numbers in {1, 2, 3, 4, .....n} are
    divisible by p?
Every pth number is divisible by p in {1, 2,
    3, 4, ...... n}. Therefore in n!, there
    are [n/p] numbers divisible by p. So we
    know that the value of x (largest power
    of p that divides n!) is at-least [n/p
   ].
Can x be larger than [n/p] ?
Yes, there may be numbers which are
   divisible by p2, p3, .....
How many numbers in {1, 2, 3, 4, ...... n}
   are divisible by p^2, p^3,...?
There are [n/(p2)] numbers divisible by p2 (
   Every p2th number would be divisible).
   Similarly, there are [n/(p3)] numbers
   divisible by p3 and so on.
What is the largest possible value of x?
So the largest possible power is [n/p] + [n
   /(p^2)] + [n/(p^3)] +.....
int Legendres formula(long long n, long long
    p) {
  int ans = 0;
  while (n) {
    ans += n / p; // now many times we can
       devide this by n , n^2 , n^3 how
       many times we can devide this like
       it
    n /= p;
  }
  return ans;
}
#num of digits
num_of_digit = floor(log10(n))+ 1 ; // 10
    base
Big gcd
gcd(a,b) == gcd(a%b , b ) ;
log(a*b) = log(a) + log(b)

I need in c++  logb(x)
but c++ only have log2 and log 10
 logb(x) = log2(x) / log2(b) ;

n^m prime factor
prime factor of n = 2^3 * 3^5 * 5^9
prime factor of n^m = 2^(3*m) * 3^(5*m) * 5^
    (9*m)

In some range some numbers are both
    divisible by x and y
Only divisible by x and y is (n/x) - n/(lcm(
    x,y))

partial_sum(a, a + 5, b, myfun); prefix sum
    can be calculated by this

how to clean the nodes used in graph
for (int i = 1 ; i <= n ; i++) {
  visited[i] = false;
  g[i].clear() ;
}
g[i].clear() empty all the data from the
    node
```