PDPM Indian Institute of Information Technology Design & Manufacturing, Jabalpur

BLE

Aman Srivastava, Armin Patel, Divyansh Tripathi

2024-12-23

Contest (1)

template.cpp
<div align="right">31 lines</div>

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

#define rep(i, a, b) for (int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
// pbds
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;
template <typename T>
using ordset =
    tree<T, null_type, less<T>, rb_tree_tag,
        tree_order_statistics_node_update>; //
            order_of_key, find_by_order
// GNU hash table
gp_hash_table<int, int> table;
mt19937 rng(chrono::steady_clock::now().
    time_since_epoch().count());
// hash for pairs
struct chash {
  int operator()(pii x) const { return x.first *
    31 + x.second; }
};
gp_hash_table<pii, int, chash> table;

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

.bashrc
<div align="right">2 lines</div>

```
alias c='g++ -Wall -Wconversion -pedantic -Wfatal
    -errors -g -std=c++17 \
  -fsanitize=undefined,address -D ONPC'
```

stress.sh
<div align="right">21 lines</div>

```bash
#!/usr/bin/env bash

for ((testNum=0;testNum<$4;testNum++))
do
    echo $testNum
    ./$3 > input
    ./$2 < input > outSlow
    ./$1 < input > outWrong
    if !(cmp -s "outWrong" "outSlow")
    then
        echo "Error found!"
        echo "Input:"
        cat input
        echo "Wrong Output:"
        cat outWrong
        echo "Slow Output:"
        cat outSlow
        exit
    fi
done
echo Passed $4 tests
```

troubleshoot.txt
<div align="right">52 lines</div>

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

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

Go for a small walk, e.g. to the toilet.
Is your output format correct? (including
    whitespace)
Rewrite your solution from the start or let a
    teammate do it.

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

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

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

Mathematics (2)

2.1   Equations

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

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

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

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

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

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

2.2 Recurrences

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

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

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

2.3 Trigonometry

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

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

$$\sin v + \sin w = 2 \sin \frac{v + w}{2} \cos \frac{v - w}{2}$$
$$\cos v + \cos w = 2 \cos \frac{v + w}{2} \cos \frac{v - w}{2}$$

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

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

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

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

2.4 Geometry

2.4.1 Quadrilaterals

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

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

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

2.5 Sums

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

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

2.6 Series

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

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

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

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

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

2.7 Probability theory

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

Expectation is linear:

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

For independent X and Y,

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

Data structures (3)

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: $O(\log N)$ $Status : stress - tested$

08d511, 22 lines

```cpp
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"   4f2aa3, 50 lines

```cpp
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);
```

```cpp
    }
    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;
  }
};
```

UnionFind.h
Description: Disjoint-set data structure.
Time: $\mathcal{O}(\alpha(N))$

<div style="text-align:right">32c6e8, 14 lines</div>

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

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

<div style="text-align:right">ab6430, 30 lines</div>

```cpp
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;
  }
};
```

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

<div style="text-align:right">ca7d3d, 21 lines</div>

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

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

<div style="text-align:right">cad9a2, 55 lines</div>

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

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

template<class F> void each(Node* n, F f) {
```

```
if (n) { each(n->l, f); f(n->val); each(n->r, f
    ); }
}

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

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

Node* ins(Node* t, Node* n, int pos) {
  auto [l,r] = split(t, pos);
  return merge(merge(l, n), r);
}

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

FenwickTree.h

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

78bf26, 25 lines

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

FenwickTree2d.h

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

"FenwickTree.h"   aa9461, 22 lines

```
struct FT2 {
  vector<vi> ys; vector<FT> ft;
  FT2(int limx) : ys(limx) {}
  void fakeUpdate(int x, int y) {
    for (; x < sz(ys); x |= x + 1) ys[x].
        push_back(y);
  }
  void init() {
    for (vi& v : ys) sort(all(v)), ft.
        emplace_back(sz(v));
  }
```

```
  int ind(int x, int y) {
    return (int)(lower_bound(all(ys[x]), y) - ys[
        x].begin()); }
  void update(int x, int y, ll dif) {
    for (; x < sz(ys); x |= x + 1)
      ft[x].update(ind(x, y), dif);
  }
  ll query(int x, int y) {
    ll sum = 0;
    for (; x; x &= x - 1)
      sum += ft[x-1].query(ind(x-1, y));
    return sum;
  }
};
```

RMQ.h

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

ef1867, 16 lines

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

MoQueries.h

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

147849, 49 lines

```
void add(int ind, int end) { ... } // add a[ind]
    (end = 0 or 1)
```

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

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


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

```
                    else { add(c, end); in[c] = 1;
                    } a = c; }
    while (!(L[b] <= L[a] && R[a] <= R[b]))
      I[i++] = b, b = par[b];
    while (a != b) step(par[a]);
    while (i--) step(I[i]);
    if (end) res[qi] = calc();
  }
  return res;
}
```

Numerical (4)

4.1 Polynomials and recurrences

Polynomial.h

```
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: $\mathcal{O}\left(n^2 \log(1/\epsilon)\right)$

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

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

```
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 $S[0 \dots \geq n-1]$ and $tr[0 \dots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.
Usage: linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number
Time: $\mathcal{O}\left(n^2 \log k\right)$

<div align="right">5aa464, 26 lines</div>

```
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   Matrices

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

<div align="right">aca249, 15 lines</div>

```
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: $\mathcal{O}\left(N^3\right)$

<div align="right">4940fc, 18 lines</div>

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

<div align="right">d11d96, 38 lines</div>

```
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:

<div align="right">"SolveLinear.h"   6a6093, 7 lines</div>

```
rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
// ... then at the end:
```

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

<div align="right">f12c3a, 34 lines</div>

```cpp
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: $\mathcal{O}(n^3)$

<div align="right">685ba1, 35 lines</div>

```cpp
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;
}
```

4.3 Fourier transforms

FastFourierTransform.h

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

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

<div align="right">2f9c18, 35 lines</div>

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

```
}
```

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"`   6a0921, 21 lines

```cpp
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;
  }
};
```

ModArithSimple.h

Description: Use this for simplicity in place of modular arithematic c. *Status : Stress − tested*

6be54c, 16 lines

```cpp
ll madd(ll a, ll b) { return (a + b) % mod; }
ll msub(ll a, ll b) { return (((a - b) % mod) +
  mod) % mod; }
ll mmul(ll a, ll b) { return ((a % mod) * (b %
  mod)) % mod; }
ll mpow(ll base, ll exp) {
  ll res = 1;
  while (exp) {
    if (exp % 2 == 1) {
      res = (res * base) % mod;
```

```cpp
    }
    exp >>= 1;
    base = (base * base) % mod;
  }
  return res;
}
ll minv(ll base) { return mpow(base, mod - 2); }
ll mdiv(ll a, ll b) { return mmul(a, minv(b)); }
```

binaryexpinverse.h

f9d4ff, 15 lines

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

ll binpow(ll a, ll b) {
  ll ans = 1;
  while (b) {
    if (b & 1) {
      ans = (ans * a) % mod;
    }
    a = (a * a) % mod;
    b = b >> 1;
  }
  return (ans % mod);
}

ll modinverse(ll a, ll b) { return binpow(a, b -
  2) % mod; }
```

ModLog.h

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

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

6a655b, 13 lines

```cpp
// Description: Returns the smallest $x > 0$ s.t.
//    $a^x = b \pmod m$, or
// * $-1$ if no such $x$ exists.
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;
}
```

ModMuLL.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: O(1) for modmul, $O(\log b)$ for modpow *Status : stress − tested, proven correct Details : This runs* $2x$ *faster than the naive* $(_{int128_t})a * b$ *doc/modmul−proof.tex. A nearlier* $\le a, b \le c < 2^{52} \approx 4.5 \cdot 10^{15}$.

5f1d88, 12 lines

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

ModSqrt.h

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

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

`"ModPow.h"`   0675ba, 24 lines

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

5.2 Primality

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

ccd47a, 20 lines

```cpp
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 $a^b \bmod c$.

"ModMulLL.h"   ef9b53, 12 lines

```cpp
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"   d8f0f9, 18 lines

```cpp
ull pollard(ull n) {
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  auto f = [&](ull x) { return modmul(x, x, n) +
      i; };
  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 $a \pmod b$.

7e9eaf, 17 lines

```cpp
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;
}


ll generalized_mod_inverse(ll a, ll m) {
  ll x, y;
  int g = euclid(a, m, x, y);
  if (g != 1) {
    return -1;
  } else {
    x = (x % m + m) % m;
    return x;
  }
}
```

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 $0 \le x < \mathrm{lcm}(m,n)$. Assumes $mn < 2^{62}$.
Time: $\log(n)$

"euclid.h"   5fcb78, 7 lines

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

5.3.1 Bézout's identity
For $a \ne$, $b \ne 0$, then $d = gcd(a,b)$ is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

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

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

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

4724b2, 8 lines

```cpp
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

FracBinarySearch.h
Description: Given f and N, finds the smallest fraction $p/q \in [0,1]$ such that $f(p/q)$ is true, and $p,q \le N$. 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: $\mathcal{O}(\log(N))$

<div align="right">fa329d, 25 lines</div>

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

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

5.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

5.6 Primes

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

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

5.7 Estimates

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

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

Combinatorial (6)

6.1 Permutations

6.1.1 Factorial

IntPerm.h
Description: Permutation -> integer conversion. (Not order preserving.) Integer -> permutation can use a lookup table.
Time: $\mathcal{O}(n)$

<div align="right">0df637, 6 lines</div>

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

6.1.2 Cycles

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

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

6.1.3 Derangements

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

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

6.1.4 Burnside's lemma

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

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

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

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

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

6.2 Partitions and subsets

6.2.1 Partition function

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

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

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

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

6.2.2 Lucas' Theorem

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

6.2.3 Binomials

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

<div align="right">a86e09, 5 lines</div>

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

6.3 General purpose numbers

6.3.1 Bernoulli numbers

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

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

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

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

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

6.3.3 Eulerian numbers

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

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

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

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

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.5 Bell numbers

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

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

6.3.6 Labeled unrooted trees

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

6.3.7 Catalan numbers

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

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

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

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

Graph (7)

7.1 Fundamentals

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

0d1e77, 23 lines

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

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

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

```cpp
    if (d < dest.dist) {
      dest.prev = ed.a;
      dest.dist = (i < lim-1 ? d : -inf);
    }
  }
  rep(i,0,lim) for (Ed e : eds) {
    if (nodes[e.a].dist == -inf)
      nodes[e.b].dist = -inf;
  }
}
```

7.2 Network flow

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

733a12, 79 lines

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

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

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

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

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

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

    __gnu_pbds::priority_queue<pair<ll, int>> q;
```

```
  vector<decltype(q)::point_iterator> its(N);
  q.push({ 0, s });

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

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

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

// If some costs can be negative, call this
    before maxflow:
void setpi(int s) { // (otherwise, leave this
    out)
  fill(all(pi), INF); pi[s] = 0;
  int it = N, ch = 1; ll v;
  while (ch-- && it--)
```

```
    rep(i,0,N) if (pi[i] != INF)
      for (edge& e : ed[i]) if (e.cap)
        if ((v = pi[i] + e.cost) < pi[e.to])
          pi[e.to] = v, ch = 1;
  assert(it >= 0); // negative cost cycle
  }
};
```

7.3 Matching

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

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

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

```
vi cover(vector<vi>& g, int n, int m) {
  vi match(m, -1);
  int res = dfsMatching(g, match);
  vector<bool> lfound(n, true), seen(m);
```

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

7.4 DFS algorithms

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

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

  if (low == val[j]) {
    do {
      x = z.back(); z.pop_back();
      comp[x] = ncomps;
      cont.push_back(x);
    } while (x != j);
    f(cont); cont.clear();
    ncomps++;
  }
  return val[j] = low;
```

```
}
template<class G, class F> void scc(G& g, F f) {
  int n = sz(g);
  val.assign(n, 0); comp.assign(n, -1);
  Time = ncomps = 0;
  rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
}
```

FindingBridges.h
Description: Finds bridges in a graph. A bridge is an edge which when removed
Time: $\mathcal{O}(N + M)$

<div align="right">bae75b, 37 lines</div>

```
void IS_BRIDGE(int v, int to); // some function
    to process the found bridge
int n;                          // number of nodes
vector<vector<int>> adj;        // adjacency list
    of graph

vector<bool> visited;
vector<int> tin, low;
int timer;


void dfs(int v, int p = -1) {
  visited[v] = true;
  tin[v] = low[v] = timer++;
  bool parent_skipped = false;
  for (int to : adj[v]) {
    if (to == p && !parent_skipped) {
      parent_skipped = true;
      continue;
    }
    if (visited[to]) {
      low[v] = min(low[v], tin[to]);
    } else {
      dfs(to, v);
      low[v] = min(low[v], low[to]);
      if (low[to] > tin[v])
        IS_BRIDGE(v, to);
    }
  }
}
void find_bridges() {
  timer = 0;
  visited.assign(n, false);
  tin.assign(n, -1);
  low.assign(n, -1);
  for (int i = 0; i < n; ++i) {
    if (!visited[i])
      dfs(i);
  }
```

```
}
```

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

<div align="right">b7e39d, 58 lines</div>

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

  TwoSat(int n = 0) : N(n), gr(2 * n) {}
  int addVar() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
  }
  void either(int f, int j) {
    f = max(2 * f, -1 - 2 * f);
    j = max(2 * j, -1 - 2 * j);
    gr[f].push_back(j ^ 1);
    gr[j].push_back(f ^ 1);
  }
  void setValue(int x) { either(x, x); }
  void atMostOne(const vi &li) { // (optional)
    if (sz(li) <= 1)
      return;
    int cur = ~li[0];
    rep(i, 2, sz(li)) {
      int next = addVar();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }
  vi val, comp, z;
  int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x;
    z.push_back(i);
    for (int e : gr[i])
      if (!comp[e])
```

```
        low = min(low, val[e] ?: dfs(e));
    if (low == val[i])
      do {
        x = z.back();
        z.pop_back();
        comp[x] = low;
        if (values[x >> 1] == -1)
          values[x >> 1] = x & 1;
      } while (x != i);
    return val[i] = low;
  }

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

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

<div align="right">4b2b78, 15 lines</div>

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

7.5 Coloring

EdgeColoring.h
Description: Given a simple, undirected graph with max degree D, computes a $(D+1)$-coloring of the edges such that no neighboring edges share a color. (D-coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.)
Time: $\mathcal{O}(NM)$

<div align="right">fd39d8, 31 lines</div>

```cpp
vi edgeColoring(int N, vector<pii> eds) {
  vi cc(N + 1), ret(sz(eds)), fan(N), free(N),
      loc;
  for (pii e : eds) ++cc[e.first], ++cc[e.second
      ];
  int u, v, ncols = *max_element(all(cc)) + 1;
  vector<vi> adj(N, vi(ncols, -1));
  for (pii e : eds) {
    tie(u, v) = e;
    fan[0] = v;
    loc.assign(ncols, 0);
    int at = u, end = u, d, c = free[u], ind = 0,
         i = 0;
    while (d = free[v], !loc[d] && (v = adj[u][d
        ]) != -1)
      loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
    cc[loc[d]] = c;
    for (int cd = d; at != -1; cd ^= c ^ d, at =
        adj[at][cd])
      swap(adj[at][cd], adj[end = at][cd ^ c ^ d
          ]);
    while (adj[fan[i]][d] != -1) {
      int left = fan[i], right = fan[++i], e = cc
          [i];
      adj[u][e] = left;
      adj[left][e] = u;
      adj[right][e] = -1;
      free[right] = e;
    }
    adj[u][d] = fan[i];
    adj[fan[i]][d] = u;
    for (int y : {fan[0], u, end})
      for (int& z = free[y] = 0; adj[y][z] != -1;
          z++);
  }
  rep(i,0,sz(eds))
    for (tie(u, v) = eds[i]; adj[u][ret[i]] != v
        ;) ++ret[i];
  return ret;
}
```

7.6 Heuristics

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

<div align="right">a99bda, 12 lines</div>

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

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

<div align="right">2752a0, 49 lines</div>

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

7.7 Trees

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

<div align="right">7fc172, 25 lines</div>

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

int jmp(vector<vi>& tbl, int nod, int steps){
  rep(i,0,sz(tbl))
```

```
    if(steps&(1<<i)) nod = tbl[i][nod];
  return nod;
}


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

LCA.h

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

"../data-structures/RMQ.h"   e0881a, 21 lines

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

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

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

HLD.h

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

"../data-structures/LazySegmentTree.h"   e9afd6, 46 lines

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

```
    int res = -1e9;
    process(u, v, [&](int l, int r) {
        res = max(res, tree->query(l, r));
    });
    return res;
  }
  int querySubtree(int v) { // modifySubtree is
      similar
    return tree->query(pos[v] + VALS_EDGES, pos[v
        ] + siz[v]);
  }
};
```

DirectedMST.h

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

"../data-structures/UnionFindRollback.h"   f3d299, 60 lines

```
struct Edge { int a, b; ll w; };
struct Node {
  Edge key;
  Node *l, *r;
  ll delta;
  void prop() {
    key.w += delta;
    if (l) l->delta += delta;
    if (r) r->delta += delta;
    delta = 0;
  }
  Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
  if (!a || !b) return a ?: b;
  a->prop(), b->prop();
  if (a->key.w > b->key.w) swap(a, b);
  swap(a->l, (a->r = merge(b, a->r)));
  return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a
    ->r); }

pair<ll, vi> dmst(int n, int r, vector<Edge>& g)
     {
  RollbackUF uf(n);
  vector<Node*> heap(n);
  for (Edge e : g) heap[e.b] = merge(heap[e.b],
      new Node{e});
  ll res = 0;
  vi seen(n, -1), path(n), par(n);
  seen[r] = r;
  vector<Edge> Q(n), in(n, {-1,-1}), comp;
```

```cpp
deque<tuple<int, int, vector<Edge>>> cycs;
rep(s,0,n) {
  int u = s, qi = 0, w;
  while (seen[u] < 0) {
    if (!heap[u]) return {-1,{}};
    Edge e = heap[u]->top();
    heap[u]->delta -= e.w, pop(heap[u]);
    Q[qi] = e, path[qi++] = u, seen[u] = s;
    res += e.w, u = uf.find(e.a);
    if (seen[u] == s) {
      Node* cyc = 0;
      int end = qi, time = uf.time();
      do cyc = merge(cyc, heap[w = path[--qi]])
        ;
      while (uf.join(u, w));
      u = uf.find(u), heap[u] = cyc, seen[u] =
        -1;
      cycs.push_front({u, time, {&Q[qi], &Q[end
        ]}});
    }
  }
  rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
}

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

7.8   Math

7.8.1   Number of Spanning Trees

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

Geometry (8)

8.1   Geometric primitives

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

b7760b, 28 lines

```cpp
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"   aff9aa, 4 lines

```cpp
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"   9e0fa5, 6 lines

```cpp
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;
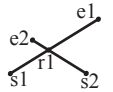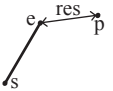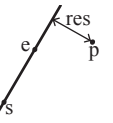
"Point.h", "OnSegment.h"   2f7799, 13 lines

```cpp
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 inter-
section 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;

<div align="right">"Point.h"   faee62, 8 lines</div>

```
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;

<div align="right">"Point.h"   c0ba2b, 9 lines</div>

```
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>.

<div align="right">"Point.h"   28a7e0, 3 lines</div>

```
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.

<div align="right">"Point.h"   ade292, 6 lines</div>

```
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();
}
```
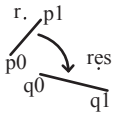
Angle.h
Description: A class for ordering angles (as represented by int points
and a number of rotations around the origin). Useful for rotational
sweeping. Sometimes also represents points or vectors.
Usage: vector<Angle> v = {w[0], w[0].t360() ...}; //
sorted
int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j;
}
// sweeps j such that (j-i) represents the number of
positively oriented triangles with vertices at 0 and i

<div align="right">3fa6eb, 35 lines</div>

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t)
      {}
  Angle operator-(Angle b) const { return {x-b.x,
      y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (half()
      && x >= 0)}; }
  Angle t180() const { return {-x, -y, t + half()
      }; }
  Angle t360() const { return {x, y, t + 1}; }
```

```
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare
      distances
  return make_tuple(a.t, a.half(), a.y * (ll)b.x)
      <
        make_tuple(b.t, b.half(), a.x * (ll)b.y)
        ;
}

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

8.2   Circles
CircleIntersection.h
Description: Computes the pair of points at which two circles inter-
sect. Returns false in case of no intersection.

<div align="right">"Point.h"   b2beb9, 11 lines</div>

```
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;
}
```

circumcircle.h

Description:

The circumcircle 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"   c6c722, 9 lines
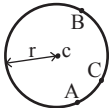
```
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P&
    C) {
  return (B-A).dist()*(C-B).dist()*(A-C).dist()/
      abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.
      cross(c)/2;
}
```

MinimumEnclosingCircle.h

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

"circumcircle.h"   3ed955, 17 lines

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

8.3 Polygons

InsidePolygon.h

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

"Point.h", "OnSegment.h", "SegmentDistance.h"   91725e, 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"   aa3c71, 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"   dd5f93, 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));
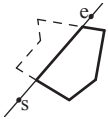
"Point.h", "lineIntersection.h"   e38095, 13 lines

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

ConvexHull.h

Description:

Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.
Time: $\mathcal{O}(n \log n)$

"Point.h"   5935f9, 13 lines

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

HullDiameter.h

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

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

"Point.h"   9bb7a3, 12 lines

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

PointInsideHull.h
Description: Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "OnSegment.h"   5bef6c, 14 lines

```cpp
typedef Point<ll> P;

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

8.4   Misc. Point Set Problems

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

"Point.h"   f85d3e, 17 lines

```cpp
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
```

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

8.5   3D

Point3D.h
Description: Class to handle points in 3D space. T can be e.g. double or long long.

                            14b717, 32 lines

```cpp
template<class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0) : x(x), y
    (y), z(z) {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z); }
  bool operator==(R p) const {
    return tie(x, y, z) == tie(p.x, p.y, p.z); }
  P operator+(R p) const { return P(x+p.x, y+p.y,
    z+p.z); }
  P operator-(R p) const { return P(x-p.x, y-p.y,
    z-p.z); }
  P operator*(T d) const { return P(x*d, y*d, z*d
    ); }
  P operator/(T d) const { return P(x/d, y/d, z/d
    ); }
  T dot(R p) const { return x*p.x + y*p.y + z*p.z
    ; }
  P cross(R p) const {
    return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y
      - y*p.x);
  }
  T dist2() const { return x*x + y*y + z*z; }
```

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

Strings (9)

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

                            3a707e, 16 lines

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

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

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

7a443b, 12 lines

```cpp
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: $\mathcal{O}(N)$

d823da, 13 lines

```cpp
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: $\mathcal{O}(N)$

18bdba, 8 lines

```cpp
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;
```

```cpp
}
```

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

077d68, 22 lines

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

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

2f0047, 50 lines

```cpp
struct SuffixTree {
  enum { N = 200010, ALPHA = 26 }; // N ~ 2*
      maxlen+10
```

```cpp
  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) {
```

```cpp
    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.
<div align="right">bf580e, 44 lines</div>

```cpp
// 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 ~ 3e9;
    random also ok)

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

```cpp
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 $\mathcal{O}(NM)$.
<div align="right">bdcee5, 66 lines</div>

```cpp
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);
```

```cpp
    N[n].end = j;
    N[n].nmatches++;
  }
  AhoCorasick(vector<string>& pat) : N(1, -1) {
    rep(i,0,sz(pat)) insert(pat[i], i);
    N[0].back = sz(N);
    N.emplace_back(0);

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

Various (10)

10.1 Intervals

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

0900be, 23 lines

```cpp
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 an-
other 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: $\mathcal{O}(N \log N)$

8c7b09, 19 lines

```cpp
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);
```

```cpp
    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: $\mathcal{O}\left(k \log \frac{n}{k}\right)$

a510b2, 19 lines

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

10.2 Misc. algorithms

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

fb4284, 11 lines

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

81ff17, 17 lines

```cpp
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: $\mathcal{O}(N \max(w_i))$

756daf, 16 lines

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

```
      rep(x,0,m) v[x+w[i]] = max(v[x+w[i]], u[x]);
      for (x = 2*m; --x > m;) rep(j, max(0,u[x]), v
          [x])
        v[x-w[j]] = max(v[x-w[j]], j);
  }
  for (a = t; v[a+m-t] < 0; a--) ;
  return a;
}
```

10.3   Dynamic programming

KnuthDP.h

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

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

DivideAndConquerDP.h

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

Time: $\mathcal{O}\left((N + (hi - lo)) \log N\right)$

<div align="right">bdae29, 18 lines</div>

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

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

10.4   Debugging tricks

- `signal(SIGSEGV, [](int) { _Exit(0); });` converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). `_GLIBCXX_DEBUG` failures generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).

- `feenableexcept(29);` kills the program on NaNs (`1`), 0-divs (`4`), infinities (`8`) and denormals (`16`).

10.5   Optimization tricks

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

10.5.1   Bit hacks

- `x & -x` is the least bit in `x`.

- `for (int x = m; x; ) { --x &= m; ... }` loops over all subset masks of `m` (except `m` itself).

- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.

- `rep(b,0,K) rep(i,0,(1 << K))`
  `if (i & 1 << b) D[i] += D[i^(1 << b)];`
  computes all sums of subsets.

10.5.2   Pragmas

- **#pragma** `GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.

- **#pragma** `GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.

- **#pragma** `GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).