WISCONSIN

UNIVERSITY OF WISCONSIN–MADISON

Longest SubRoutine

Mingrui Liu, Nitit Jongsawatsataporn, Ziyi Zhang

adapted from KTH ACM Contest Template Library

2024-04-04

Contest (1)

TODOs.txt

11 lines

```
- Geo add half-plan inter, ask sub what to add (section 7)
- test bit more for compressedTree.h
- Get familiar with blossom
- find PAM template
- vertex BCC, edge BCC

Sections
- Data structures (section 2)
- Graphs (section 6)
- Strings (section 8)
```

template.cpp

36 lines

```
// THINK/VERIFY about implementation before start to code,
   ↪maybe there is easier one!!
// read carefully! (input/output format, details, etc)

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

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

// sub
#include <bits/stdc++.h>
#define FR(i, n) for (int i=0; i<n;++i)
#define eb emplace_back
#define st first
#define nd second
using namespace std;
namespace R=ranges;
template<typename T>
using func=function<T>;
using ll = long long;
using ld = long double;
using i128=__int128;
using pii = pair<int, int>;
const int inf = 1e9 + 7;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

}
```

hash-cpp.sh

1 lines

```
cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum
```

clion.txt

3 lines

```
set(CMAKE_CXX_STANDARD 17)
set(GCC_COVERAGE_COMPILE_FLAGS "-g -O2 -std=gnu++20 -static
   ↪ -Wall -Werror")
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${
   ↪GCC_COVERAGE_COMPILE_FLAGS}")
```

minors.cpp

45 lines

```
// Define Hash Function for std::unordered_map
struct HASH{
    size_t operator()(const pii &x)const{
        return hash<ll>()((x.first)^(((ll)x.second)<<32));
    }
};
std::unordered_map<pii, int, HASH> mp;

// customize comparator for std::set
struct cmp {
    bool operator()(const edge &x, const edge &y) const {
       ↪return x.w < y.w; }
};
std::set<edge, cmp> S;

// multiply numbers up to 1e18 under some modulo
ll big_mul(ll a, ll b, ll mod)
{
    ll q = (ll)((ld) a * (ld) b / (ld) mod);
    ll r = a * b - q * mod;
    return (r + mod) % mod;
}
int main(){
    //random number
    mt19937 rng(chrono::steady_clock::now().
       ↪time_since_epoch().count());
    cout << rng() % 5 << endl;
    vi v;
    std::shuffle(v.begin(), v.end(), rng);

    //calculating sum of floor(n/i) in O(sqrt(n))
    int n, ans;
    for (int i = 1, j = 0; i <= n; i = j + 1) j = n/(n/i),
       ↪ans += 1ll*(j-i+1)*(n/i);

    // Iterate every submask
    for(int mask = 0; mask < (1 << n); mask++) {
        for(int sub = mask; ; sub = (sub - 1) & mask) {
            //...
            if(sub == 0) break;
        }
    }

    //Better hash map
    unordered_map<int, int> mp;
    mp.reserve(32768);
    mp.max_load_factor(0.25);
}
```

troubleshoot.txt

52 lines

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

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all datastructures 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 team mate.
Ask the team mate 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 team mate 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 team mates think about your algorithm?

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

Data structures (2)

Fenwick.h

Time: Both operations are $\mathcal{O}(\log N)$.

14 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 get(int pos) { // sum of values in [0, pos]
        ll res = 0;
        for (; pos >= 0; pos = (pos & (pos + 1)) - 1) res
           ↪+= s[pos];
        return res;
    }
};

// hash-cpp-all = 0dfaad1c17da97862ea61ff26651839a
```

2DBIT.cpp

29 lines

```
vector<int> vals[N], f[N];

void addupd(int x, int y) {
```

```
    for (int i = x; i < N; i |= i + 1) vals[i].push_back(y)
        ↪;
}

void addget(int x, int y) {
    if (x < 0 || y < 0) return;
    for (int i = x; i >= 0; i = (i & (i + 1)) - 1) vals[i].
        ↪push_back(y);
}

void upd(int x, int y, int v) {
    for (int i = x; i < N; i |= i + 1) {
        for (int j = lower_bound(vals[i].begin(), vals[i].
            ↪end(), y) - vals[i].begin();
             j < (int) f[i].size(); j |= j + 1) {
            f[i][j] += v;
        }
    }
}

int get(int x, int y) {
    if (x < 0 || y < 0) return 0;
    int res = 0;
    for (int i = x; i >= 0; i = (i & (i + 1)) - 1)
        for (int j = lower_bound(vals[i].begin(), vals[i].
            ↪end(), y) - vals[i].begin(); j >= 0;
             j = (j & (j + 1)) - 1)
            res += f[i][j];
    return res;
} // hash-cpp-all = a5676421701a8edc43b837632d70b2d2
```

HashMap.h

Time: Faster/better hash maps, taken from CF

<ext/pb_ds/assoc_container.hpp> 19 lines

```
using namespace __gnu_pbds;
gp_hash_table<int, int> table;
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::
            ↪steady_clock::now().time_since_epoch().count()
            ↪;
        return splitmix64(x + FIXED_RANDOM);
    }
};

gp_hash_table<int, int, custom_hash> safe_table;
// hash-cpp-all = eaa5ef366bbafa054e59a02f8ec40bc8
```

DynamicConvexHullTrick.cpp

Description: Query will return the maximum of the lines. 32 lines

```
typedef long long ll;

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)
    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()) { x->p = inf; return false; }
        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;
    }
}; // hash-cpp-all = 78fbd6d5c3d5351b8f2712bd6c535ce9
```

StaticConvexHullTrick.cpp

Description: Maintaining upper convex hull, querying the maximum. Need to put in lines in strictly increasing order of slope. 51 lines

```
struct Line {
    ll k, m;
    Line(ll _k, ll _m) {
        k = _k, m = _m;
    }
    Pll inter(Line o) {
        return {m - o.m, o.k - k};
    }
};

struct Hull {
    deque<Line> que;

    bool leq(Pll a, Pll b) {
        return a.first * b.second <= a.second * b.first;
    }

    // k needs to be strictly increasing!
    void add(ll k, ll m) {
        while(que.size() > 1) {
            int ls = que.size() - 1;
            if(leq(que[ls].inter(Line(k, m)), que[ls-1].
                ↪inter(que[ls]))) que.pop_back();
            else break;
        }
        que.push_back({k, m});
    }

    // Arbitrary x.
    ll query_bin(ll x) {
        if(que.empty()) return -INF;
        int l = 0, r = que.size() - 1;
        while(l < r) {
            int mi = (l + r) / 2;
            if(que[mi].k * x + que[mi].m < que[mi+1].k * x
                ↪+ que[mi+1].m) l = mi + 1;
            else r = mi;
        }
        return que[l].k * x + que[l].m;
    }

    // If querying increasing x.
    ll query(ll x) {
        if(que.empty()) return -INF;
        while(que.size() > 1) {
            if(que[0].k * x + que[0].m < que[1].k * x + que
                ↪[1].m) que.pop_front();
            else break;
        }
        return que[0].k * x + que[0].m;
    }

} hull;
// hash-cpp-all = 817ff0175d9f4dd826f400423d205fc4
```

PersistentSegTree.cpp

47 lines

```
#define lson(x) dat[x].ls
#define rson(x) dat[x].rs

struct PST {
    int ncnt = 1; //Need to initialize before every test
        ↪case!

    struct node{
        int ls, rs, sum;
    } dat[N * 30];

    int leaf(int val){
        dat[ncnt].ls = dat[ncnt].rs = 0;
        dat[ncnt].sum = val;
        return ncnt++;
    }

    int combine(int ls, int rs) {
        lson(ncnt) = ls;
        rson(ncnt) = rs;
        dat[ncnt].sum = (ls ? dat[ls].sum : 0) + (rs ? dat[
            ↪rs].sum : 0);
        return ncnt++;
    }

    int n;

    int build(int a[], int tl = 0, int tr = n-1){
        if(tl == tr) return leaf(a[tl]);
        int mid = (tl + tr) / 2;
        return combine(build(a, tl, mid), build(a, mid + 1,
            ↪ tr));
    }

    int get_sum(int v, int l, int r, int tl = 0, int tr = n
        ↪-1){
        if(tr < l || tl > r) return 0;
        if(l <= tl && tr <= r) return dat[v].sum;
        int tm = (tl + tr) / 2;

        return get_sum(lson(v), l, r, tl, tm)
               + get_sum(rson(v), l, r, tm + 1, tr);
    }

    int update(int v, int pos, int tl = 0, int tr = n - 1){
        if(tl == tr) return leaf(dat[v].sum + 1);
        int tm = (tl + tr) / 2;
```

```
        if(pos <= tm) return combine(update(lson(v), pos,
            ↪tl, tm), rson(v));
        else return combine(lson(v), update(rson(v), pos,
            ↪tm+1, tr));
    }
} // hash-cpp-all = 67c9c6a393624027d626740fd7104805
```

LazyPersistentSegTree.cpp

70 lines

```
int ncnt = 1; //Need to initialize before every test case!

struct node{
    int ls, rs, lazy;
    ll sum;
} ns[N * 100];

int newnode(int val){
    ns[ncnt].ls = ns[ncnt].rs = 0;
    ns[ncnt].sum = val;
    ns[ncnt].lazy = 0;
    return ncnt++;
}

int newnode(int ls, int rs){
    ns[ncnt].ls = ls;
    ns[ncnt].rs = rs;
    ns[ncnt].sum = (ls ? ns[ls].sum : 0) + (rs ? ns[rs].sum
        ↪ : 0);
    ns[ncnt].lazy = 0;
    return ncnt++;
}

int n, q;
int num[N];
int vs[N];
int tim = 0;

int newlazynode(int v, int val, int l, int r){
    ns[ncnt].ls = ns[v].ls;
    ns[ncnt].rs = ns[v].rs;
    ns[ncnt].lazy = ns[v].lazy + val;
    ns[ncnt].sum = ns[v].sum + (r - l + 1) * val;
    return ncnt++;
}

void push_down(int v, int tl, int tr){
    if(ns[v].lazy){
        if(tl != tr){
            int mid = (tl + tr) / 2;
            ns[v].ls = newlazynode(ns[v].ls, ns[v].lazy, tl
                ↪, mid);
            ns[v].rs = newlazynode(ns[v].rs, ns[v].lazy,
                ↪mid + 1, tr);
        }
        ns[v].lazy = 0;
    }
}

int build(int a[], int tl = 0, int tr = n-1){
    if(tl == tr) return newnode(a[tl]);
    int mid = (tl + tr) / 2;
    return newnode(build(a, tl, mid), build(a, mid + 1, tr)
        ↪);
}

ll get_sum(int v, int l, int r, int tl = 0, int tr = n-1){
    if(tr < l || tl > r) return 0;
    if(l <= tl && tr <= r) return ns[v].sum;
    push_down(v, tl, tr);
    int tm = (tl + tr) / 2;

    return get_sum(ns[v].ls, l, r, tl, tm)
           + get_sum(ns[v].rs, l, r, tm + 1, tr);
}

int update(int v, int l, int r, int val, int tl = 0, int tr
    ↪ = n-1){
    if(tr < l || tl > r) return v;
    if(l <= tl && tr <= r) return newlazynode(v, val, tl,
        ↪tr);
    push_down(v, tl, tr);
    int tm = (tl + tr) / 2;
    return newnode(update(ns[v].ls, l, r, val, tl, tm),
        ↪update(ns[v].rs, l, r, val, tm+1, tr));
} // hash-cpp-all = 8cd19de4c489c7ed299600e48c2be0ad
```

IterativeSegTree.cpp

Description: Iterative SegTree

20 lines

```
int n;
node t[2 * N];

void build() {  // build the tree
    for (int i = n; i < 2 * n; i++) t[i] = a[i];
    for (int i = n - 1; i > 0; --i) t[i] = combine(t[i<<1],
        ↪ t[i<<1|1]);
}

void modify(int p, const S& value) {
  for (t[p += n] = value; p >>= 1; ) t[p] = combine(t[p
      ↪<<1], t[p<<1|1]);
}

S query(int l, int r) {
  S resl, resr;
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l&1) resl = combine(resl, t[l++]);
    if (r&1) resr = combine(t[--r], resr);
  }
  return combine(resl, resr);
} // hash-cpp-all = 78e522e0dbc21cd07ad9d0fc8ba29435
```

SegmentTree.h

40 lines

```
#define lson(x) 2*x+1
#define rson(x) 2*x+2

struct node {
    ll sum;
    void add(int val) {
        sum += val;
    }
};

node merge(node L, node R) {
    return {L.sum + R.sum};
}

struct Tree {
    int n;
    vector<node> dat;
    Tree(int n): dat(n * 4) {}
    void init(int x, int l, int r, vi &a) {
        if(l == r) {dat[x] = {a[l]}; return ;}
        int mid = (l + r) / 2;
        init(lson(x), l, mid, a);
        init(rson(x), mid + 1, r, a);
        dat[x] = merge(dat[lson(x)], dat[rson(x)]);
    }
    void update(int pos, int x, int l, int r, int val) {
        if(l == r) {dat[x].add(val); return ;}
        int mid = (l + r) / 2;
        if(pos <= mid) update(pos, lson(x), l, mid, val);
        else update(pos, rson(x), mid + 1, r, val);
        dat[x] = merge(dat[lson(x)], dat[rson(x)]);
    }
    node query(int a, int b, int x, int l, int r) {
        if(r < a || b < l) return {0};
        int mid = (l + r) / 2;
        if(a <= l && r <= b) return dat[x];
        else return merge(query(a, b, lson(x), l, mid),
                          query(a, b, rson(x), mid + 1, r))
                            ↪;
    }
}; // hash-cpp-all = e61e995bbd5ae0395757a852bf93e3a5
```

LazySegmentTree.cpp

74 lines

```
#define lson(x) x * 2 + 1
#define rson(x) x * 2 + 2

struct node {
    ll sum, add;

    void add_val(ll x, ll len) {
        (add += x) %= mod;
        (sum += len * x) %= mod;
    }

    void merge(node &ls, node &rs) {
        sum = (ls.sum + rs.sum) % mod;
    }
};

struct Tree {
    int n;
    vector<node> dat;
    Tree(int n): dat(n * 4), n(n) {}

    void push_down(int x, int l, int r) {
        if(dat[x].add) {
            if(l < r) {
                int mid = (r + l) / 2;
                dat[lson(x)].add_val(dat[x].add, mid - l +
                    ↪1);
                dat[rson(x)].add_val(dat[x].add, r - mid);
            }
            dat[x].add = 0;
        }
    }

    void init(int x, int l, int r, vi &a) {
        if(l == r) {
            dat[x].sum = a[l] % mod, dat[x].add = 0;
            return ;
        }
        int mid = (l + r) / 2;
        init(lson(x), l, mid, a);
        init(rson(x), mid + 1, r, a);
        dat[x].add = 0;
        dat[x].merge(dat[lson(x)], dat[rson(x)]);
```

```
    }

    ll query(int a, int b, int x, int l, int r) {
        int mid = (l + r) / 2;
        if(r < a || l > b) return 0;
        push_down(x, l, r);
        if(l >= a && r <= b) return dat[x].sum;
        return (query(a, b, lson(x), l, mid) + query(a, b,
            ↪rson(x), mid+1, r)) % mod;
    }

    void update(int a, int b, int x, int l, int r, int
        ↪delta) {
        int mid = (l + r) / 2;
        if(r < a || l > b) return ;
        push_down(x, l, r);
        if(l >= a && r <= b) {
            dat[x].add_val(delta, r - l + 1);
            return ;
        }
        update(a, b, lson(x), l, mid, delta);
        update(a, b, rson(x), mid+1, r, delta);
        dat[x].merge(dat[lson(x)], dat[rson(x)]);
    }

    ll qry(int a, int b) {
        return query(a, b, 0, 0, n - 1);
    }
    void upd(int a, int b, int d) {
        update(a, b, 0, 0, n - 1, d);
    }
} ;

// hash-cpp-all = 6ff006de84588d52d298b1e6856e8d51
```

RMQ.cpp

Description: sparse table for range minimum, query is inclusive

19 lines

```
template<class T>
struct RMQ {
    vector<vector<T>> jmp;

    RMQ(const vector<T>& V) {
        int N = sz(V), on = 1, depth = 1;
        while (on < sz(V)) on *= 2, depth++;
        jmp.assign(depth, V);
        rep(i,0,depth-1) rep(j,0,N)
                jmp[i+1][j] = min(jmp[i][j],
                                  jmp[i][min(N - 1, j + (1
                                     ↪<< i))]);
    }

    T query(int a, int b) {
        assert(a <= b); // or return inf if a > b
        int dep = 31 - __builtin_clz(b + 1 - a);
        return min(jmp[dep][a], jmp[dep][b + 1 - (1 << dep)
            ↪]);
    }
}; // hash-cpp-all = 2cc350229a2ae135d4b6ebd4421df8ca
```

OrderStatisticTree.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element.

<bits/extc++.h> 15 lines

```
using namespace __gnu_pbds;

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

void example() {
    Tree<int> t, t2; t.insert(8);
    auto it = t.insert(10).first;
    assert(it == t.lower_bound(9));
    assert(t.order_of_key(10) == 1);
    assert(t.order_of_key(11) == 2);
    assert(*t.find_by_order(0) == 8);
    t.join(t2); // assuming T < T2 or T > T2, merge t2 into
        ↪ t
} // hash-cpp-all = 819d08f2c7f487c28758a3cf6fc13438
```

Treap.cpp

Description: Split-Merge Treap. Make sure to visit value after all the tags are pushed down!

89 lines

```
mt19937 rng(23333); // hash-cpp-1
struct Node {
  Node *l = 0, *r = 0;
  int y, c = 1; // y: random key
  pii v; // in case of duplicate key, use pair.
  int add; // int par; if wants to keep the parent, update
     ↪every time parent-child relationship, changes.
    Node(int v, int idx) : y(rng()), v({v, idx}), add(0) {
    }
    void recalc();
    void push_down();
};

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

void add_val(Node* n, int val) {
    if(!n) return ;
    n->add += val;
    n->v.first += val;
}

void Node::push_down() {
    if(add) {
        add_val(l, add);
        add_val(r, add);
        add = 0;
  }
}

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

Node* merge(Node* l, Node* r) {
    if (!l) return r;
    if (!r) return l;
    l->push_down();
    r->push_down();

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

Node* ins(Node* t, Node* n, int pos) {
    auto pa = split(t, pos);
    return merge(merge(pa.first, n), pa.second);
} // hash-cpp-1 = 6e5d471e88534f2a793013cfb260982c

// hash-cpp-2
Node* del(Node* t, int pos) {
    auto pa = split(t, pos);
    auto pb = split(pa.second, 1);
    return merge(pa.first, pb.second);
}
Node* find(Node *cur, int k) {
    if(!cur) return 0;
    if(cnt(cur->l) > k) return find(cur->l, k);
    else if(cnt(cur -> l) == k) return cur;
    else return find(cur->r, k - cnt(cur->l) - 1);
}
typedef pair<Node*, int> pni;
pni lower_bound(Node *cur, pii v, int pos) { // return {the
   ↪ Node*, position}; {NULL, n} in case of .end()
    if(!cur) return {0, pos};
    cur->push_down();
    if(cur->v >= v) {
        auto res = lower_bound(cur->l, v, pos);
        if(res.first) return res;
        else return {cur, pos + cnt(cur->l)};
  } else return lower_bound(cur->r, v, pos + cnt(cur->l) +
     ↪1);
} // hash-cpp-2 = 60c2dc01455aab4ddd306d730ca6e361
```

Numerical (3)

GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum. The maximum error in the result is eps. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.

Usage: `double func(double x) { return 4+x+.3*x*x; }`
`double xmin = gss(-1000,1000,func);`

Time: $\mathcal{O}\left(\log((b-a)/\epsilon)\right)$

14 lines

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

```
      x2 = a + r*(b-a); f2 = f(x2);
    }
  return a;
} // hash-cpp-all = 31d45b514727a298955001a74bb9b9fa
```

Polynomial.h

17 lines

```
struct Poly {
  vector<double> a;
  double operator()(double x) const {
    double val = 0;
    for(int i = sz(a); i--;) (val *= x) += a[i];
    return val;
  }
  void diff() {
    rep(i,1,sz(a)) a[i-1] = i*a[i];
    a.pop_back();
  }
  void divroot(double x0) {
    double b = a.back(), c; a.back() = 0;
    for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b,
      ↪b=c;
    a.pop_back();
  }
}; // hash-cpp-all = c9b7b07a5aae7b0a6df1b8cdb046375f
```

PolyRoots.h

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

"Polynomial.h" 23 lines

```
vector<double> poly_roots(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 = poly_roots(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;
} // hash-cpp-all = 2cf1903cf3e930ecc5ea0059a9b7fce5
```

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

13 lines

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd res(n), temp(n);
  rep(k,0,n-1) rep(i,k+1,n)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  double last = 0; temp[0] = 1;
  rep(k,0,n) rep(i,0,n) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
} // hash-cpp-all = 08bf48c9301c849dfc6064b6450af6f3
```

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}`

"../number-theory/ModPow.h" 20 lines

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

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

  C.resize(L + 1); C.erase(C.begin());
  trav(x, C) x = (mod - x) % mod;
  return C;
} // hash-cpp-all = 40387d9fed31766a705d6b2206790deb
```

LinearRecurrence.h

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

26 lines

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) { // hash-cpp-1
  int n = sz(S);

  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;
} // hash-cpp-1 = 261dd85251df2df60ee444e087e8ffc2
```

Integrate.h

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

8 lines

```
double quad(double (*f)(double), double a, double b) {
  const int n = 1000;
  double h = (b - a) / 2 / n;
  double v = f(a) + f(b);
  rep(i,1,n*2)
    v += f(a + i*h) * (i&1 ? 4 : 2);
  return v * h / 3;
} // hash-cpp-all = 65e2375b3152c23048b469eb414fe6b6
```

IntegrateAdaptive.h

Description: Fast integration using an adaptive Simpson's rule.
Usage: `double z, y;`

```
double h(double x) { return x*x + y*y + z*z <= 1; }
double g(double y) { ::y = y; return quad(h, -1, 1); }
double f(double z) { ::z = z; return quad(g, -1, 1); }
double sphereVol = quad(f, -1, 1), pi = sphereVol*3/4;
```

16 lines

```
typedef double d;
d simpson(d (*f)(d), d a, d b) {
  d c = (a+b) / 2;
  return (f(a) + 4*f(c) + f(b)) * (b-a) / 6;
}
d rec(d (*f)(d), d a, d b, d eps, d S) {
  d c = (a+b) / 2;
  d S1 = simpson(f, a, c);
  d S2 = simpson(f, c, b), T = S1 + S2;
  if (abs (T - S) <= 15*eps || b-a < 1e-10)
    return T + (T - S) / 15;
  return rec(f, a, c, eps/2, S1) + rec(f, c, b, eps/2, S2);
}
d quad(d (*f)(d), d a, d b, d eps = 1e-8) {
  return rec(f, a, b, eps, simpson(f, a, b));
} // hash-cpp-all = ad8a754372ce74e5a3d07ce46c2fe0ca
```

Determinant.h

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

15 lines

```
double det(vector<vector<double>>& a) {
  int n = sz(a); double res = 1;
  rep(i,0,n) {
    int b = i;
    rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
    if (i != b) swap(a[i], a[b]), res *= -1;
    res *= a[i][i];
    if (res == 0) return 0;
    rep(j,i+1,n) {
      double v = a[j][i] / a[i][i];
      if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
    }
  }
  return res;
} // hash-cpp-all = bd5cec161e6ad4c483e662c34eae2d08
```

IntDeterminant.h

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

18 lines

```
const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
  int n = sz(a); ll ans = 1;
  rep(i,0,n) {
    rep(j,i+1,n) {
      while (a[j][i] != 0) { // gcd step
        ll t = a[i][i] / a[j][i];
        if (t) rep(k,i,n)
          a[i][k] = (a[i][k] - a[j][k] * t) % mod;
        swap(a[i], a[j]);
        ans *= -1;
      }
    }
    ans = ans * a[i][i] % mod;
    if (!ans) return 0;
  }
  return (ans + mod) % mod;
} // hash-cpp-all = 3313dc3b38059fdf9f41220b469cfd13
```

Simplex.h

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

68 lines

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

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

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

  LPSolver(const vvd& A, const vd& b, const vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) { //
      ↪hash-cpp-1
      rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
      rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[
        ↪i];}
      rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
      N[n] = -1; D[m+1][n] = 1;
    } // hash-cpp-1 = 6ff8e92a6bb47fbd6606c75a07178914

  void pivot(int r, int s) { // hash-cpp-2
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
      T *b = D[i].data(), inv2 = b[s] * inv;
      rep(j,0,n+2) b[j] -= a[j] * inv2;
      b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  } // hash-cpp-2 = 9cd0a84b89fb678b2888e0defa688de2

  bool simplex(int phase) { // hash-cpp-3
    int x = m + phase - 1;
    for (;;) {
      int s = -1;
      rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
      if (D[x][s] >= -eps) return true;
      int r = -1;
      rep(i,0,m) {
        if (D[i][s] <= eps) continue;
        if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                     < MP(D[r][n+1] / D[r][s], B[r])) r = i
                       ↪;
      }
      if (r == -1) return false;
      pivot(r, s);
    }
  } // hash-cpp-3 = f156440bce4f5370ea43b0efa7de25ed

  T solve(vd &x) { // hash-cpp-4
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
      pivot(r, n);
      if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
      rep(i,0,m) if (B[i] == -1) {
        int s = 0;
        rep(j,1,n+1) ltj(D[i]);
        pivot(i, s);
      }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
  } // hash-cpp-4 = 396a95621f5e196bb87eb95518560dfb
};
```

math-simplex.cpp

Description: Simplex algorithm. WARNING- segfaults on empty (size 0) max cx st Ax<=b, x>=0 do 2 phases; 1st check feasibility; 2nd check boundedness and ans

40 lines

```
vector<double> simplex(vector<vector<double> > A, vector<
  ↪double> b, vector<double> c) {
  int n = (int) A.size(), m = (int) A[0].size()+1, r = n, s
    ↪ = m-1;
  vector<vector<double> > D = vector<vector<double> > (n+2,
    ↪ vector<double>(m+1));
  vector<int> ix = vector<int> (n+m);
  for (int i=0; i<n+m; i++) ix[i] = i;
  for (int i=0; i<n; i++) {
    for (int j=0; j<m-1; j++)D[i][j]=-A[i][j];
    D[i][m-1] = 1;
    D[i][m] = b[i];
    if (D[r][m] > D[i][m]) r = i;
  }
  for (int j=0; j<m-1; j++) D[n][j]=c[j];
  D[n+1][m-1] = -1; int z = 0;
  for (double d;;) {
    if (r < n) {
      swap(ix[s], ix[r+m]);
      D[r][s] = 1.0/D[r][s];
      for (int j=0; j<=m; j++) if (j!=s) D[r][j] *= -D[r][s
        ↪];
      for(int i=0; i<=n+1; i++)if(i!=r) {
        for (int j=0; j<=m; j++) if(j!=s) D[i][j] += D[r][j
          ↪] * D[i][s];
        D[i][s] *= D[r][s];
      }
    }
    r = -1; s = -1;
    for (int j=0; j <m; j++) if (s<0 || ix[s]>ix[j]) {
      if (D[n+1][j]>eps || D[n+1][j]>-eps && D[n][j]>eps) s
        ↪ = j;
    }
    if (s < 0) break;
    for (int i=0; i<n; i++) if(D[i][s]<-eps) {
      if (r < 0 || (d = D[r][m]/D[r][s]-D[i][m]/D[i][s]) <
        ↪-eps
        || d < eps && ix[r+m] > ix[i+m]) r=i;
    }
    if (r < 0) return vector<double>(); // unbounded
  }
  if (D[n+1][m] < -eps) return vector<double>(); //
    ↪infeasible
  vector<double> x(m-1);
  for (int i = m; i < n+m; i ++) if (ix[i] < m-1) x[ix[i]]
    ↪= D[i-m][m];
  printf("%.2lf\n", D[n][m]);
  return x; // ans: D[n][m]
} // hash-cpp-all = 70201709abdff05eff90d9393c756b95
```

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

38 lines

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

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

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

```
  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
} // hash-cpp-all = 44c9ab90319b30df6719c5b5394bc618
```

SolveLinear2.h

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

"SolveLinear.h" 8 lines

```
rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
  rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
  x[col[i]] = b[i] / A[i][i];
fail:; }
// hash-cpp-all = 08e495d9d51e80a183ccd030e3bf6700
```

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

34 lines

```
typedef bitset<1000> bs;

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

  x = bs();
  for (int i = rank; i--;) {
    if (!b[i]) continue;
    x[col[i]] = 1;
    rep(j,0,i) b[j] ^= A[j][i];
  }
  return rank; // (multiple solutions if rank < m)
} // hash-cpp-all = fa2d7a3e3a84d8fb47610cc474e77b4e
```

MatrixInverse.h

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

35 lines

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

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

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

  rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
  return n;
} // hash-cpp-all = ebfff64122d6372fde3a086c95e2cfc7
```

Tridiagonal.h

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

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

This is useful for solving problems on the type

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

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

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

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

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

26 lines

```
typedef double T;
vector<T> tridiagonal(vector<T> diag, const vector<T>&
    ↪super,
    const vector<T>& sub, vector<T> b) {
  int n = sz(b); vi tr(n);
  rep(i,0,n-1) {
    if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i]
        ↪== 0
      b[i+1] -= b[i] * diag[i+1] / super[i];
      if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];
      diag[i+1] = sub[i]; tr[++i] = 1;
    } else {
      diag[i+1] -= super[i]*sub[i]/diag[i];
      b[i+1] -= b[i]*sub[i]/diag[i];
    }
  }
  for (int i = n; i--;) {
    if (tr[i]) {
      swap(b[i], b[i-1]);
      diag[i-1] = diag[i];
      b[i] /= super[i-1];
    } else {
      b[i] /= diag[i];
      if (i) b[i-1] -= b[i]*super[i-1];
    }
  }
  return b;
} // hash-cpp-all = 8f9fa8b1e5e82731da914aed0632312f
```

ModInt.cpp

70 lines

```
const int MOD = (int)1e9 + 9;

struct mi {
    typedef decay<decltype(MOD)>::type T;
    T v; explicit operator T() const { return v; }
    mi() { v = 0; }
    mi(ll _v) {
        v = (-MOD < _v && _v < MOD) ? _v : _v % MOD;
        if (v < 0) v += MOD;
    }
    friend bool operator==(const mi& a, const mi& b) {
        return a.v == b.v; }
    friend bool operator!=(const mi& a, const mi& b) {
        return !(a == b); }
    friend bool operator<(const mi& a, const mi& b) {
        return a.v < b.v; }
//    friend void re(mi& a) { ll x; re(x); a = mi(x); }
//    friend str ts(mi a) { return ts(a.v); }

    mi& operator+=(const mi& m) {
        if ((v += m.v) >= MOD) v -= MOD;
        return *this; }
    mi& operator-=(const mi& m) {
        if ((v -= m.v) < 0) v += MOD;
        return *this; }
    mi& operator*=(const mi& m) {
        v = (ll)v*m.v%MOD; return *this; }
    mi& operator/=(const mi& m) { return (*this) *= inv(m);
        ↪ }
    friend mi pow(mi a, ll p) {
        mi ans = 1; assert(p >= 0);
        for (; p; p /= 2, a *= a) if (p&1) ans *= a;
        return ans;
    }
    friend mi inv(const mi& a) { assert(a.v != 0);
```

```
        return pow(a,MOD-2); }

    mi operator-() const { return mi(-v); }
    mi& operator++() { return *this += 1; }
    mi& operator--() { return *this -= 1; }
    friend mi operator+(mi a, const mi& b) { return a += b;
      ↪ }
    friend mi operator-(mi a, const mi& b) { return a -= b;
      ↪ }
    friend mi operator*(mi a, const mi& b) { return a *= b;
      ↪ }
    friend mi operator/(mi a, const mi& b) { return a /= b;
      ↪ }
};

const int N = (int)1e5 + 50;

mt19937 rng((uint32_t)chrono::steady_clock::now().
  ↪time_since_epoch().count());
typedef array<int,2> T; // bases not too close to ends
uniform_int_distribution<int> BDIST(0.1*MOD,0.9*MOD);
const T base = {BDIST(rng),BDIST(rng)};
//const T base = {10, 10};
const T ibase = {(int)inv(mi(base[0])),(int)inv(mi(base[1])
  ↪)};
T operator+(T l, T r) {
    rep(i,0,2) if ((l[i] += r[i]) >= MOD) l[i] -= MOD;
    return l; }
T operator-(T l, T r) {
    rep(i,0,2) if ((l[i] -= r[i]) < 0) l[i] += MOD;
    return l; }
T operator*(T l, T r) {
    rep(i,0,2) l[i] = (ll)l[i]*r[i]%MOD;
    return l; }

vector<T> pows = {{1,1}}, ipows = {{1,1}};

int main() {
    while (sz(pows) < N) pows.push_back(base*pows.back()),
      ↪ipows.push_back(ibase*ipows.back());
}
// hash-cpp-all = f421cc631d1d8eca46a7923bc768761a
```

3.1 Fourier transforms

fft.cpp

Description: FFT/NTT, polynomial mod/log/exp 303 lines

```
namespace fft {
#if FFT
// FFT
using dbl = double;
struct num { // hash-cpp-1
  dbl x, y;
  num(dbl x_ = 0, dbl y_ = 0) : x(x_), y(y_) { }
};
inline num operator+(num a, num b) { return num(a.x + b.x,
  ↪a.y + b.y); }
inline num operator-(num a, num b) { return num(a.x - b.x,
  ↪a.y - b.y); }
inline num operator*(num a, num b) { return num(a.x * b.x -
  ↪ a.y * b.y, a.x * b.y + a.y * b.x); }
inline num conj(num a) { return num(a.x, -a.y); }
inline num inv(num a) { dbl n = (a.x*a.x+a.y*a.y); return
  ↪num(a.x/n,-a.y/n); }
// hash-cpp-1 = d2cc70ff17fe23dbfe608d8bce4d827b
#else
// NTT
const int mod = 998244353, g = 3;
// For p < 2^30 there is also (5 << 25, 3), (7 << 26, 3),
// (479 << 21, 3) and (483 << 21, 5). Last two are > 10^9.
struct num { // hash-cpp-2
  int v;
  num(ll v_ = 0) : v(int(v_ % mod)) { if (v<0) v+=mod; }
  explicit operator int() const { return v; }
};
inline num operator+(num a,num b){return num(a.v+b.v);}
inline num operator-(num a,num b){return num(a.v+mod-b.v);}
inline num operator*(num a,num b){return num(1ll*a.v*b.v);}
inline num pow(num a, int b) {
  num r = 1;
  do{if(b&1)r=r*a;a=a*a;}while(b>>=1);
  return r;
}
inline num inv(num a) { return pow(a, mod-2); }
// hash-cpp-2 = 62f50e0b94ea4486de6fbc07e826040a
#endif

using vn = vector<num>;
vi rev({0, 1});
vn rt(2, num(1)), fa, fb;

inline void init(int n) { // hash-cpp-3
  if (n <= sz(rt)) return;
  rev.resize(n);
  rep(i,0,n) rev[i] = (rev[i>>1] | ((i&1)*n)) >> 1;
  rt.reserve(n);
  for (int k = sz(rt); k < n; k *= 2) {
    rt.resize(2*k);
#if FFT
    double a=M_PI/k; num z(cos(a),sin(a)); // FFT
#else
    num z = pow(num(g), (mod-1)/(2*k)); // NTT
#endif
    rep(i,k/2,k) rt[2*i] = rt[i], rt[2*i+1] = rt[i]*z;
  }
} // hash-cpp-3 = 408005a3c0a4559a884205d5d7db44e9

inline void fft(vector<num> &a, int n) { // hash-cpp-4
  init(n);
  int s = __builtin_ctz(sz(rev)/n);
  rep(i,0,n) if (i < rev[i]>>s) swap(a[i], a[rev[i]>>s]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
      num t = rt[j+k] * a[i+j+k];
      a[i+j+k] = a[i+j] - t;
      a[i+j] = a[i+j] + t;
    }
} // hash-cpp-4 = 1f0820b04997ddca9b78742df352d419

// Complex/NTT
vn multiply(vn a, vn b) { // hash-cpp-5
  int s = sz(a) + sz(b) - 1;
  if (s <= 0) return {};
  int L = s > 1 ? 32 - __builtin_clz(s-1) : 0, n = 1 << L;
  a.resize(n), b.resize(n);
  fft(a, n);
  fft(b, n);
  num d = inv(num(n));
  rep(i,0,n) a[i] = a[i] * b[i] * d;
  reverse(a.begin()+1, a.end());
  fft(a, n);
  a.resize(s);
  return a;
} // hash-cpp-5 = 7a20264754593de4eb7963d8fc3d8a15

// Complex/NTT power-series inverse
// Doubles b as b[:n] = (2 - a[:n] * b[:n/2]) * b[:n/2]
vn inverse(const vn& a) { // hash-cpp-6
  if (a.empty()) return {};
  vn b({inv(a[0])});
  b.reserve(2*a.size());
  while (sz(b) < sz(a)) {
    int n = 2*sz(b);
    b.resize(2*n, 0);
    if (sz(fa) < 2*n) fa.resize(2*n);
    fill(fa.begin(), fa.begin()+2*n, 0);
    copy(a.begin(), a.begin()+min(n,sz(a)), fa.begin());
    fft(b, 2*n);
    fft(fa, 2*n);
    num d = inv(num(2*n));
    rep(i, 0, 2*n) b[i] = b[i] * (2 - fa[i] * b[i]) * d;
    reverse(b.begin()+1, b.end());
    fft(b, 2*n);
    b.resize(n);
  }
  b.resize(a.size());
  return b;
} // hash-cpp-6 = 61660c4b2c75faa72062368a381f059f

#if FFT
// Double multiply (num = complex)
using vd = vector<double>;
vd multiply(const vd& a, const vd& b) { // hash-cpp-7
  int s = sz(a) + sz(b) - 1;
  if (s <= 0) return {};
  int L = s > 1 ? 32 - __builtin_clz(s-1) : 0, n = 1 << L;
  if (sz(fa) < n) fa.resize(n);
  if (sz(fb) < n) fb.resize(n);

  fill(fa.begin(), fa.begin() + n, 0);
  rep(i,0,sz(a)) fa[i].x = a[i];
  rep(i,0,sz(b)) fa[i].y = b[i];
  fft(fa, n);
  trav(x, fa) x = x * x;
  rep(i,0,n) fb[i] = fa[(n-i)&(n-1)] - conj(fa[i]);
  fft(fb, n);
  vd r(s);
  rep(i,0,s) r[i] = fb[i].y / (4*n);
  return r;
} // hash-cpp-7 = c2431bc9cb89b2ad565db6fba6a21a32

// Integer multiply mod m (num = complex) // hash-cpp-8
vi multiply_mod(const vi& a, const vi& b, int m) {
  int s = sz(a) + sz(b) - 1;
  if (s <= 0) return {};
  int L = s > 1 ? 32 - __builtin_clz(s-1) : 0, n = 1 << L;
  if (sz(fa) < n) fa.resize(n);
  if (sz(fb) < n) fb.resize(n);

  rep(i,0,sz(a)) fa[i] = num(a[i] & ((1<<15)-1), a[i] >>
    ↪15);
  fill(fa.begin()+sz(a), fa.begin() + n, 0);
  rep(i,0,sz(b)) fb[i] = num(b[i] & ((1<<15)-1), b[i] >>
    ↪15);
  fill(fb.begin()+sz(b), fb.begin() + n, 0);

  fft(fa, n);
  fft(fb, n);
  double r0 = 0.5 / n; // 1/2n
  rep(i,0,n/2+1) {
    int j = (n-i)&(n-1);
    num g0 = (fb[i] + conj(fb[j])) * r0;
    num g1 = (fb[i] - conj(fb[j])) * r0;
```

```
    swap(g1.x, g1.y); g1.y *= -1;
    if (j != i) {
      swap(fa[j], fa[i]);
      fb[j] = fa[j] * g1;
      fa[j] = fa[j] * g0;
    }
    fb[i] = fa[i] * conj(g1);
    fa[i] = fa[i] * conj(g0);
  }
  fft(fa, n);
  fft(fb, n);
  vi r(s);
  rep(i,0,s) r[i] = int((ll(fa[i].x+0.5)
        + (ll(fa[i].y+0.5) % m << 15)
        + (ll(fb[i].x+0.5) % m << 15)
        + (ll(fb[i].y+0.5) % m << 30)) % m);
  return r;
} // hash-cpp-8 = e8c5f6755ad1e5a976d6c6ffd37b3b22
#endif

} // namespace fft

// For multiply_mod, use num = modnum, poly = vector<num>
using fft::num;
using poly = fft::vn;
using fft::multiply;
using fft::inverse;
// hash-cpp-9
poly& operator+=(poly& a, const poly& b) {
  if (sz(a) < sz(b)) a.resize(b.size());
  rep(i,0,sz(b)) a[i]=a[i]+b[i];
  return a;
}
poly operator+(const poly& a, const poly& b) { poly r=a; r
    ↪+=b; return r; }
poly& operator-=(poly& a, const poly& b) {
  if (sz(a) < sz(b)) a.resize(b.size());
  rep(i,0,sz(b)) a[i]=a[i]-b[i];
  return a;
}
poly operator-(const poly& a, const poly& b) { poly r=a; r
    ↪-=b; return r; }
poly operator*(const poly& a, const poly& b) {
  // TODO: small-case?
  return multiply(a, b);
}
poly& operator*=(poly& a, const poly& b) {return a = a*b;}
// hash-cpp-9 = 61b8743c2b07beed0e7ca857081e1bd4
poly& operator*=(poly& a, const num& b) { // Optional
  trav(x, a) x = x * b;
  return a;
}
poly operator*(const poly& a, const num& b) { poly r=a; r*=
    ↪b; return r; }

// Polynomial floor division; no leading 0's plz
poly operator/(poly a, poly b) { // hash-cpp-10
  if (sz(a) < sz(b)) return {};
  int s = sz(a)-sz(b)+1;
  reverse(a.begin(), a.end());
  reverse(b.begin(), b.end());
  a.resize(s);
  b.resize(s);
  a = a * inverse(move(b));
  a.resize(s);
  reverse(a.begin(), a.end());
  return a;
} // hash-cpp-10 = a6589ce8fcf1e33df3b42ee703a7fe60
poly& operator/=(poly& a, const poly& b) {return a = a/b;}
poly& operator%=(poly& a, const poly& b) { // hash-cpp-11
  if (sz(a) >= sz(b)) {
    poly c = (a / b) * b;
    a.resize(sz(b)-1);
    rep(i,0,sz(a)) a[i] = a[i]-c[i];
  }
  return a;
} // hash-cpp-11 = 9af255f48abbeafd8acde353357b84fd
poly operator%(const poly& a, const poly& b) { poly r=a; r
    ↪%=b; return r; }

// Log/exp/pow
poly deriv(const poly& a) { // hash-cpp-12
  if (a.empty()) return {};
  poly b(sz(a)-1);
  rep(i,1,sz(a)) b[i-1]=a[i]*i;
  return b;
} // hash-cpp-12 = 94aa209b3e956051e6b3131bf1faafd1
poly integ(const poly& a) { // hash-cpp-13
  poly b(sz(a)+1);
  b[1]=1; // mod p
  rep(i,2,sz(b)) b[i]=b[fft::mod%i]*(-fft::mod/i); // mod p
  rep(i,1,sz(b)) b[i]=a[i-1]*b[i]; // mod p
  //rep(i,1,sz(b)) b[i]=a[i-1]*inv(num(i)); // else
  return b;
} // hash-cpp-13 = 6f13f6a43b2716a116d347000820f0bd
poly log(const poly& a) { // a[0] == 1 // hash-cpp-14
  poly b = integ(deriv(a)*inverse(a));
  b.resize(a.size());
  return b;
} // hash-cpp-14 = ce1533264298c5382f72a2a1b0947045
poly exp(const poly& a) { // a[0] == 0 // hash-cpp-15
  poly b(1,num(1));
  if (a.empty()) return b;
  while (sz(b) < sz(a)) {
    int n = min(sz(b) * 2, sz(a));
    b.resize(n);
    poly v = poly(a.begin(), a.begin() + n) - log(b);
    v[0] = v[0]+num(1);
    b *= v;
    b.resize(n);
  }
  return b;
} // hash-cpp-15 = f645d091e4ae3ee3dc2aa095d4aa699a
poly pow(const poly& a, int m) { // m >= 0 // hash-cpp-16
  poly b(a.size());
  if (!m) { b[0] = 1; return b; }
  int p = 0;
  while (p<sz(a) && a[p].v==0) ++p;
  if (1ll*m*p >= sz(a)) return b;
  num mu = pow(a[p], m), di = inv(a[p]);
  poly c(sz(a) - m*p);
  rep(i,0,sz(c)) c[i] = a[i+p] * di;
  c = log(c);
  trav(v,c) v = v * m;
  c = exp(c);
  rep(i,0,sz(c)) b[i+m*p] = c[i] * mu;
  return b;
} // hash-cpp-16 = 0f4830b9de34c26d39f170069827121f

// Multipoint evaluation/interpolation
// hash-cpp-17
vector<num> eval(const poly& a, const vector<num>& x) {
  int n=sz(x);
  if (!n) return {};
  vector<poly> up(2*n);
  rep(i,0,n) up[i+n] = poly({0-x[i], 1});
  per(i,1,n) up[i] = up[2*i]*up[2*i+1];
  vector<poly> down(2*n);
  down[1] = a % up[1];
  rep(i,2,2*n) down[i] = down[i/2] % up[i];
  vector<num> y(n);
  rep(i,0,n) y[i] = down[i+n][0];
  return y;
} // hash-cpp-17 = a079eba46c3110851ec6b0490b439931
// hash-cpp-18
poly interp(const vector<num>& x, const vector<num>& y) {
  int n=sz(x);
  assert(n);
  vector<poly> up(n*2);
  rep(i,0,n) up[i+n] = poly({0-x[i], 1});
  per(i,1,n) up[i] = up[2*i]*up[2*i+1];
  vector<num> a = eval(deriv(up[1]), x);
  vector<poly> down(2*n);
  rep(i,0,n) down[i+n] = poly({y[i]*inv(a[i])});
  per(i,1,n) down[i] = down[i*2] * up[i*2+1] + down[i*2+1]
    ↪* up[i*2];
  return down[1];
} // hash-cpp-18 = 74f15e1e82d51e852b321a1ff75ba1fd
```

FastSubsetTransform.h

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

16 lines

```
void FST(vi& a, bool inv) {
  for (int n = sz(a), step = 1; step < n; step *= 2) {
    for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step) {
      int &u = a[j], &v = a[j + step]; tie(u, v) =
        inv ? pii(v - u, u) : pii(v, u + v); // AND
        inv ? pii(v, u - v) : pii(u + v, u); // OR
        pii(u + v, u - v);                   // XOR
    }
  }
  if (inv) trav(x, a) x /= sz(a); // XOR only
}
vi conv(vi a, vi b) {
  FST(a, 0); FST(b, 0);
  rep(i,0,sz(a)) a[i] *= b[i];
  FST(a, 1); return a;
} // hash-cpp-all = 3de473e2c1de97e6e9ff0f13542cf3fb
```

Number theory (4)

4.1 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" 18 lines

```
const ll mod = 17; // change to something else
struct Mod {
  ll x;
  Mod(ll xx) : x(xx) {}
  Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
  Mod operator-(Mod b) { return Mod((x - b.x + mod) % mod);
    ↪ }
  Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
  Mod operator/(Mod b) { return *this * invert(b); }
  Mod invert(Mod a) {
    ll x, y, g = euclid(a.x, mod, x, y);
```

```
    assert(g == 1); return Mod((x + mod) % mod);
  }
  Mod operator^(ll e) {
    if (!e) return Mod(1);
    Mod r = *this ^ (e / 2); r = r * r;
    return e&1 ? *this * r : r;
  }
}; // hash-cpp-all = 35bfea8c111cb24c4ce84c658446961b
```

ModInverse.h

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

4 lines

```
const ll mod = 1000000007, LIM = 200000;
ll* inv = new ll[LIM] - 1; inv[1] = 1;
rep(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;
// hash-cpp-all = 6f684f0b9ae6c69f42de68f023a81de5
```

ModPow.h

6 lines

```
const ll mod = 1000000007; // faster if const
ll modpow(ll a, ll e) {
  if (e == 0) return 1;
  ll x = modpow(a * a % mod, e >> 1);
  return e & 1 ? x * a % mod : x;
} // hash-cpp-all = 2fa6d9ccac4586cba0618aad18cdc9de
```

ModSum.h

Description: Sums of mod'ed arithmetic progressions.

`modsum(to, c, k, m)` $= \sum_{i=0}^{to-1} (ki + c)\%m$. `divsum` is similar but for floored division.

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

19 lines

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

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

ll modsum(ull to, ll c, ll k, ll m) {
  c = ((c % m) + m) % m;
  k = ((k % m) + m) % m;
  return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
} // hash-cpp-all = 8d6e082e0ea6be867eaea12670d08dcc
```

ModMulLL.h

Description: Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for large c.

Time: $\mathcal{O}\left(64/bits \cdot \log b\right)$, where $bits = 64 - k$, if we want to deal with k-bit numbers.

19 lines

```
typedef unsigned long long ull;
const int bits = 10;
// if all numbers are less than 2^k, set bits = 64-k
const ull po = 1 << bits;
ull mod_mul(ull a, ull b, ull &c) {
  ull x = a * (b & (po - 1)) % c;
  while ((b >>= bits) > 0) {
    a = (a << bits) % c;
    x += (a * (b & (po - 1))) % c;
  }
  return x % c;
}
ull mod_pow(ull a, ull b, ull mod) {
  if (b == 0) return 1;
  ull res = mod_pow(a, b / 2, mod);
  res = mod_mul(res, res, mod);
  if (b & 1) return mod_mul(res, a, mod);
  return res;
} // hash-cpp-all = 40cd743544228d297c803154525107ab
```

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots.

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

"ModPow.h"

30 lines

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(modpow(a, (p-1)/2, p) == 1);
  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;
  int r = 0;
  while (s % 2 == 0)
    ++r, s /= 2;
  ll n = 2; // find a non-square mod p
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p);
  ll g = modpow(n, s, p);
  for (;;) {
    ll t = b;
    int m = 0;
    for (; m < r; ++m) {
      if (t == 1) break;
      t = t * t % p;
    }
    if (m == 0) return x;
    ll gs = modpow(g, 1 << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
    r = m;
  }
} // hash-cpp-all = 83e24bd39c8c93946ad3021b8ca6c3c4
```

BabyStepGiantStep.cpp

30 lines

```
//Given g^x=y(mod p), find x in O(log(p)*sqrt(p))
struct BSGS {
    const static int p = 998244353;
    const static int g = 3;
    int m;
    map<int, int> mp;
    vector<int> V;

    void pre() {
        m = (int)ceil(sqrt(p + 0.0) + 1);
        int cg = 1, revgm = 1;
        int invm = (int)(fp(fp(g, m), p - 2));
        for(int i=0;i<m;i++) {
            mp[cg] = i;
            V.push_back(revgm);
            cg = (int)(1LL * cg * g % p);
            revgm = (int)(1LL * revgm * invm % p);
        }
    }
    int find(int y) {
        for(int i=0;i<m;i++){
            int cur = (int)(1LL * V[i] * y % p);
            if(mp.count(cur)){
                return i * m + mp[cur];
            }
        }
        return -1;
    }
}; // hash-cpp-all = 27d863b1e9415002dd99f945abfeb344
```

4.2 Primality

eratosthenes.h

Description: Prime sieve for generating all primes up to a certain limit. isprime[i] is true iff i is a prime.

Time: lim=100'000'000 $\approx$ 0.8 s. Runs 30% faster if only odd indices are stored.

11 lines

```
const int MAX_PR = 5000000;
bitset<MAX_PR> isprime;
vi eratosthenes_sieve(int lim) {
  isprime.set(); isprime[0] = isprime[1] = 0;
  for (int i = 4; i < lim; i += 2) isprime[i] = 0;
  for (int i = 3; i*i < lim; i += 2) if (isprime[i])
    for (int j = i*i; j < lim; j += i*2) isprime[j] = 0;
  vi pr;
  rep(i,2,lim) if (isprime[i]) pr.push_back(i);
  return pr;
} // hash-cpp-all = 0564a3337fb69c0b87dfd3c56cdfe2e3
```

MillerRabin.h

Description: Miller-Rabin primality probabilistic test. Probability of failing one iteration is at most 1/4. 15 iterations should be enough for 50-bit numbers.

Time: 15 times the complexity of $a^b \mod c$.

"ModMulLL.h"

16 lines

```
bool prime(ull p) {
  if (p == 2) return true;
  if (p == 1 || p % 2 == 0) return false;
  ull s = p - 1;
  while (s % 2 == 0) s /= 2;
  rep(i,0,15) {
    ull a = rand() % (p - 1) + 1, tmp = s;
    ull mod = mod_pow(a, tmp, p);
    while (tmp != p - 1 && mod != 1 && mod != p - 1) {
      mod = mod_mul(mod, mod, p);
      tmp *= 2;
    }
    if (mod != p - 1 && tmp % 2 == 0) return false;
  }
  return true;
} // hash-cpp-all = ccddf18bab60a654ff4af45e95dd60b6
```

factor.h

Description: Pollard's rho algorithm. It is a probabilistic factorisation algorithm, whose expected time complexity is good. Before you start using it, run `init(bits)`, where bits is the length of the numbers you use. Returns factors of the input without duplicates.

Time: Expected running time should be good enough for 50-bit numbers.

"ModMulLL.h", "MillerRabin.h", "eratosthenes.h"

35 lines

```
vector<ull> pr;
ull f(ull a, ull n, ull &has) {
  return (mod_mul(a, a, n) + has) % n;
```

```
}
vector<ull> factor(ull d) {
  vector<ull> res;
  for (int i = 0; i < sz(pr) && pr[i]*pr[i] <= d; i++)
    if (d % pr[i] == 0) {
      while (d % pr[i] == 0) d /= pr[i];
      res.push_back(pr[i]);
    }
  //d is now a product of at most 2 primes.
  if (d > 1) {
    if (prime(d))
      res.push_back(d);
    else while (true) {
      ull has = rand() % 2321 + 47;
      ull x = 2, y = 2, c = 1;
      for (; c==1; c = __gcd((y > x ? y - x : x - y), d)) {
        x = f(x, d, has);
        y = f(f(y, d, has), d, has);
      }
      if (c != d) {
        res.push_back(c); d /= c;
        if (d != c) res.push_back(d);
        break;
      }
    }
  }
  return res;
}
void init(int bits) {//how many bits do we use?
  vi p = eratosthenes_sieve(1 << ((bits + 2) / 3));
  pr.assign(all(p));
} // hash-cpp-all = 67b304bd690b2a8445a7b4dbf93996d7
```

4.3 Divisibility

euclid.h

Description: Finds the Greatest Common Divisor to the integers a and b. Euclid also finds two integers x and y, such that $ax + by = \gcd(a, b)$. If a and b are coprime, then x is the inverse of $a \pmod b$.

7 lines

```
ll gcd(ll a, ll b) { return __gcd(a, b); }

ll euclid(ll a, ll b, ll &x, ll &y) {
  if (b) { ll d = euclid(b, a % b, y, x);
    return y -= a/b * x, d; }
  return x = 1, y = 0, a;
} // hash-cpp-all = 63e6f8d2f560b27cb800273d63d2102c
```

Euclid.java

Description: Finds {x, y, d} s.t. ax + by = d = gcd(a, b).

11 lines

```
static BigInteger[] euclid(BigInteger a, BigInteger b) {
  BigInteger x = BigInteger.ONE, yy = x;
  BigInteger y = BigInteger.ZERO, xx = y;
  while (b.signum() != 0) {
    BigInteger q = a.divide(b), t = b;
    b = a.mod(b); a = t;
    t = xx; xx = x.subtract(q.multiply(xx)); x = t;
    t = yy; yy = y.subtract(q.multiply(yy)); y = t;
  }
  return new BigInteger[]{x, y, a};
}
```

4.4 Fractions

ContinuedFractions.h

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

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

21 lines

```
typedef double d; // for N ~ 1e7; long double for N ~ 1e9
pair<ll, ll> approximate(d x, ll N) {
  ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x
    ↪;
  for (;;) {
    ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf
      ↪),
      a = (ll)floor(y), b = min(a, lim),
      NP = b*P + LP, NQ = b*Q + LQ;
    if (a > b) {
      // If b > a/2, we have a semi-convergent that gives
        ↪us a
      // better approximation; if b = a/2, we *may* have
        ↪one.
      // Return {P, Q} here for a more canonical
        ↪approximation.
      return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q)
        ↪) ?
        make_pair(NP, NQ) : make_pair(P, Q);
    }
    if (abs(y = 1/(y - (d)a)) > 3*N) {
      return {NP, NQ};
    }
    LP = P; P = NP;
    LQ = Q; Q = NQ;
  }
} // hash-cpp-all = dd6c5e1084a26365dc6321bd935975d9
```

FracBinarySearch.h

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

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

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

24 lines

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

template<class F>
Frac fracBS(F f, ll N) {
  bool dir = 1, A = 1, B = 1;
  Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N
    ↪]
  assert(!f(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;
} // hash-cpp-all = 214844f17d0c347ff436141729e0c829
```

4.5 Chinese remainder theorem

chinese.h

Description: Chinese Remainder Theorem.

`chinese(a, m, b, n)` returns a number x, such that $x \equiv a \pmod m$ and $x \equiv b \pmod n$. For not coprime n, m, use `chinese_common`. Note that all numbers must be less than 2^{31} if you have Z = unsigned long long.

Time: $\log(m + n)$

"euclid.h"

13 lines

```
template<class Z> Z chinese(Z a, Z m, Z b, Z n) {
  Z x, y; euclid(m, n, x, y);
  Z ret = a * (y + m) % m * n + b * (x + n) % n * m;
  if (ret >= m * n) ret -= m * n;
  return ret;
}

template<class Z> Z chinese_common(Z a, Z m, Z b, Z n) {
  Z d = gcd(m, n);
  if (((b -= a) %= n) < 0) b += n;
  if (b % d) return -1; // No solution
  return d * chinese(Z(0), m/d, b/d, n/d) + a;
} // hash-cpp-all = da3099704e14964aa045c152bb478c14
```

4.6 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

4.7 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 $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

4.8 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 (5)

5.1 Permutations

5.1.1 Factorial

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

IntPerm.h
Description: Permutation -> integer conversion. (Not order preserving.)
Time: $\mathcal{O}(n)$ 6 lines

```
int permToInt(vi& v) {
  int use = 0, i = 0, r = 0;
  trav(x,v)r=r * ++i + __builtin_popcount(use & -(1 << x)),
    use |= 1 << x;                  // (note: minus, not ~!)
  return r;
} // hash-cpp-all = e1b8eaea02324af14a3da94f409019b8
```

5.1.2 Cycles

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

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

5.1.3 Derangements

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

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

5.1.4 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

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

5.2 Partitions and subsets

5.2.1 Partition function

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

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

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

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

5.2.2 Binomials

binomialModPrime.h
Description: Lucas' thm: Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + \ldots + n_1 p + n_0$ and $m = m_k p^k + \ldots + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k}\binom{n_i}{m_i} \pmod p$. fact and invfact must hold precomputed factorials / inverse factorials, e.g. from ModInverse.h.
Time: $\mathcal{O}(\log_p n)$ 10 lines

```
ll chooseModP(ll n, ll m, int p, vi& fact, vi& invfact) {
  ll c = 1;
  while (n || m) {
    ll a = n % p, b = m % p;
    if (a < b) return 0;
    c = c * fact[a] % p * invfact[b] % p * invfact[a - b] %
      ↪ p;
    n /= p; m /= p;
  }
  return c;
} // hash-cpp-all = 81845faa6ecd635c391e4f0134f0676c
```

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

```
ll multinomial(vi& v) {
  ll c = 1, m = v.empty() ? 1 : v[0];
  rep(i,1,sz(v)) rep(j,0,v[i])
    c = c * ++m / (j+1);
  return c;
} // hash-cpp-all = a0a3128f6afa4721166feb182b82f130
```

5.3 General purpose numbers

5.3.1 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:

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

Euler-Maclaurin formula for infinite sums:

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

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

5.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

5.3.3 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. $\pi(j) > j$.

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

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

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

5.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

5.3.5 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,

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

5.3.6 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: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

5.3.7 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.

5.4 Other

nim-product.cpp

Description: Nim Product. 17 lines

```
using ull = uint64_t;
ull _nimProd2[64][64];
ull nimProd2(int i, int j) {
  if (_nimProd2[i][j]) return _nimProd2[i][j];
  if ((i & j) == 0) return _nimProd2[i][j] = 1ull << (i|j);
  int a = (i&j) & -(i&j);
  return _nimProd2[i][j] = nimProd2(i ^ a, j) ^ nimProd2((i
    ↪ ^ a) | (a-1), (j ^ a) | (i & (a-1)));
}
ull nimProd(ull x, ull y) {
  ull res = 0;
  for (int i = 0; (x >> i) && i < 64; i++)
    if ((x >> i) & 1)
      for (int j = 0; (y >> j) && j < 64; j++)
        if ((y >> j) & 1)
          res ^= nimProd2(i, j);
  return res;
} // hash-cpp-all = 9bba25d6ea05316a1be6cbff8d591d78
```

schreier-sims.cpp

Description: Check group membership of permutation groups 52 lines

```
struct Perm {
  int a[N];
  Perm() {
    for (int i = 1; i <= n; ++i) a[i] = i;
  }
  friend Perm operator* (const Perm &lhs, const Perm &rhs)
    ↪{
    static Perm res;
    for (int i = 1; i <= n; ++i) res.a[i] = lhs.a[rhs.a[i
      ↪]];
    return res;
  }
  friend Perm inv(const Perm &cur) {
    static Perm res;
    for (int i = 1; i <= n; ++i) res.a[cur.a[i]] = i;
    return res;
  }
};
class Group {
  bool flag[N];
  Perm w[N];
  std::vector<Perm> x;
public:
  void clear(int p) {
    memset(flag, 0, sizeof flag);
    for (int i = 1; i <= n; ++i) w[i] = Perm();
    flag[p] = true;
    x.clear();
  }
  friend bool check(const Perm&, int);
  friend void insert(const Perm&, int);
  friend void updateX(const Perm&, int);
} g[N];
bool check(const Perm &cur, int k) {
  if (!k) return true;
  int t = cur.a[k];
  return g[k].flag[t] ? check(g[k].w[t] * cur, k - 1) :
    ↪false;
}
void updateX(const Perm&, int);
void insert(const Perm &cur, int k) {
  if (check(cur, k)) return;
  g[k].x.push_back(cur);
  for (int i = 1; i <= n; ++i) if (g[k].flag[i]) updateX(
    ↪cur * inv(g[k].w[i]), k);
}
void updateX(const Perm &cur, int k) {
  int t = cur.a[k];
  if (g[k].flag[t]) {
    insert(g[k].w[t] * cur, k - 1);
  } else {
    g[k].w[t] = inv(cur);
    g[k].flag[t] = true;
    for (int i = 0; i < g[k].x.size(); ++i) updateX(g[k].x[
      ↪i] * cur, k);
  }
} // hash-cpp-all = 949a6e50dbdaea9cda09928c7eabedbc
```

Graph (6)

6.1 Euler walk

EulerWalk.h

Description: store the **reverse path** in the graph!!! 36 lines

```
struct Euler {
    struct edge {
        // c is orientation of the edge
        int to, id, c;
    };
    vector<vector<edge>> G;
    vi pt, evis, path;
    Euler(int n, int m): G(n), pt(n, 0), evis(m, 0){};

    void add_edge(int u, int v, int eid) {
        G[u].push_back({v, eid, 0});
        G[v].push_back({u, eid, 1});
    }

    void dfs(int v){
        int &i = pt[v];
        while(true){
            while(i < sz(G[v]) && evis[G[v][i].id]) i++; if
              ↪(i == sz(G[v])) break;
            auto e = G[v][i++];
            evis[e.id] = 1;
            dfs(e.to);
            // record the result
            res[e.id] = e.c;
            path.push_back(v);
        }
    }

    void get_path() {
        rep(i, 0, sz(G)) {
            if(pt[i] < sz(G[i])) {
                dfs(i);
            }
        }
    }
}; // hash-cpp-all = 6b832f15cb06417384241ca67e72ba36
```

6.2 Flows & Matching

Dinic.cpp 63 lines

```
struct edge {
    int to, cap, rev;
    edge(int _to, int _cap, int _rev) {
        to = _to, cap = _cap, rev = _rev;
    }
};

struct Dinic {
    vector<vector<edge>> G;
    vi level, iter;

    Dinic(int n) : G(n), level(n), iter(n) {}

    void add_edge (int from, int to, int cap) {
        G[from].push_back(edge(to, cap, sz(G[to])));
        G[to].push_back(edge(from, 0, sz(G[from]) - 1));
    }

    void bfs(int s) {
        fill(all(level), -1);
        queue<int> que;
        level[s] = 0;
        que.push(s);
        while(!que.empty()) {
            int v = que.front(); que.pop();
            for (auto &e : G[v]) {
                if(e.cap > 0 && level[e.to] < 0) {
                    level[e.to] = level[v] + 1;
                    que.push(e.to);
                }
            }
        }
    }

    int dfs(int v , int t, int f) {
        if(v == t) return f;
        for (int &i = iter[v]; i < sz(G[v]); i++) {
            edge &e = G[v][i];
```

```
            if (e.cap > 0 && level[v] < level[e.to]) {
                int d = dfs(e.to, t, min(e.cap, f));
                if (d > 0) {
                    e.cap -= d;
                    G[e.to][e.rev].cap += d;
                    return d;
                }
            }
        }
        return 0;
    }

    int max_flow(int s, int t) {
        int flow = 0;
        for (;;) {
            bfs(s);
            if(level[t] < 0) return flow;
            fill(all(iter), 0);
            int f ;
            while ((f = dfs(s, t, INF)) > 0) {
                flow += f;
            }
        }
    }
}; // hash-cpp-all = 8b741744942f4e5057e4b383f5b51738
```

FordFulkerson.cpp

46 lines

```
struct Flow {
    struct edge{
        int to, cap, rev;
        edge(int _to, int _cap, int _rev){
            to = _to, cap = _cap, rev = _rev;
        }
    };

    vector<vector<edge>> G;
    vector<int> used;
    Flow(int n): G(n), used(n) {}

    void add_edge(int from, int to, int cap){
        G[from].push_back(edge(to, cap, G[to].size()));
        G[to].push_back(edge(from, 0, G[from].size() - 1));
    }

    int dfs(int v, int t, int f){
        if(v == t) return f;
        used[v] = true;

        for(int i = 0; i < G[v].size(); i++){
            edge &e = G[v][i];
            if(!used[e.to] && e.cap > 0){
                int d = dfs(e.to, t, min(e.cap, f));
                if(d > 0){
                    e.cap -= d;
                    G[e.to][e.rev].cap += d;
                    return d;
                }
            }
        }
        return 0;
    }

    int max_flow(int s, int t){
        int f = 0;
        for(;;){
            fill(used.begin(), used.end(), 0);
            int d = dfs(s, t, INF);
            if(d == 0) return f;
            f += d;
        }
    }
}; // hash-cpp-all = a3de2b0f3581a5ad8de5433a3ba300e6
```

MaximumWeightMatching.cpp

47 lines

```
const int MAXN = 2010;
const int oo = 1000000007;

int dist[MAXN][MAXN];
// Finding the minimum weight prefect matching (of size n)
   ↪in O(N^3)
// The dist matrix is 1-indexed.
int hungarian(int n, int m){
  vector<int> u(n + 1), v(m + 1), p(m + 1), way(m + 1);
  for(int i = 1; i <= n; i++){
    p[0] = i;
    int j0 = 0;
    vector<int> minv(m + 1, oo);
    vector<char> used(m + 1, false);
    do{
      used[j0] = true;
      int i0 = p[j0], delta = oo, j1;
      for(int j = 1; j <= m; j++){
        if(!used[j]){
          int cur = dist[i0][j] - u[i0] - v[j];
          if(cur < minv[j]){
            minv[j] = cur;
            way[j] = j0;
          }
          if(minv[j] < delta){
            delta = minv[j];
            j1 = j;
          }
        }
      }
      for(int j = 0; j <= m; j++){
        if(used[j]){
          u[p[j]] += delta;
          v[j] -= delta;
        } else {
          minv[j] -= delta;
        }
      }
      j0 = j1;
    } while (p[j0] != 0);
    do{
      int j1 = way[j0];
      p[j0] = p[j1];
      j0 = j1;
    } while(j0);
  }
  return -v[0];
} // hash-cpp-all = 7aa16f18feade9a42ebdb04feae4632b
```

MinimumCostMaxFlow.cpp

60 lines

```
const int INF = (int)1e9;

typedef pair<int, int> P;
struct edge{int to, cap, cost, rev;};

struct MincostFlow {
    vector<vector<edge>> G;
    vi h, dist, prevv, preve;

    MincostFlow(int n = 0): G(n), h(n), dist(n), prevv(n),
       ↪preve(n) {}

    void add_edge(int from, int to, int cap, int cost){
        G[from].push_back((edge){to, cap, cost, (int)G[to].
           ↪size()});
        G[to].push_back((edge){from, 0, -cost, (int)G[from
           ↪].size() - 1});
    }

    int min_cost_flow(int s, int t, int f){
        int res = 0;
        // substitute with Bellman-ford/DAG DP for negative
           ↪ edges
        fill(all(h), 0);
        while(f > 0){
            priority_queue<P, vector<P>, greater<P> > que;
            fill(all(dist), INF);
            dist[s] = 0;
            que.push(P(0, s));
            while(!que.empty()){
                P p = que.top(); que.pop();
                int v = p.second;
                if(dist[v] < p.first) continue;
                for(int i = 0; i < G[v].size(); i++){
                    edge &e = G[v][i];
                    if(e.cap > 0 && dist[e.to] > dist[v] +
                       ↪e.cost + h[v] - h[e.to]){
                        dist[e.to] = dist[v] + e.cost + h[v
                           ↪] - h[e.to];
                        prevv[e.to] = v;
                        preve[e.to] = i;
                        que.push(P(dist[e.to], e.to));
                    }
                }
            }
            if(dist[t] == INF) {
              return -1;
            }
            for(int v = 0; v < sz(h); v++) h[v] += dist[v];

            int d = f;
            for(int v = t; v != s; v = prevv[v]){
                d = min(d, G[prevv[v]][preve[v]].cap);
            }
            f -= d;
            res += d * h[t];
            for(int v = t; v != s; v = prevv[v]){
                edge &e = G[prevv[v]][preve[v]];
                e.cap -= d;
                G[v][e.rev].cap += d;
            }
        }
        return res;
    }

} ; // hash-cpp-all = 9b745e74379363b3ef7ec1095927fc01
```

GomoryHu.cpp

Description: Finding a tree where the min edge on the path from a to b is the min a-b-cut. Complexity: O(n * c(max flow))

17 lines

```
struct GomoryHu {
    vector<pair<pii, ll>> ed;
    void ae(ll a, ll b, ll c) { ed.push_back({{a, b}, c});}
```

```
  vector<pair<pii, ll>> init(ll N) {
      vector<pii> ret(N + 1, {0, 0});
      rep(i, 1, N) {
          Dinic D(N);
          trav(t, ed) D.add_edge(t.first.first, t.first.
              ↪second, t.second),
              D.add_edge(t.first.second, t.first.first, t
                  ↪.second);
      ret[i].second = D.max_flow(i, ret[i].first);
    rep(j, i + 1, N) if (ret[j].first == ret[i].first &&
        ↪D.level[j] >= 0) ret[j].first = i;
      }
      vector<pair<pii, ll>> res;
      rep(i, 1, N) res.push_back({{i, ret[i].first}, ret[
          ↪i].second});
      return res;
  }
}; // hash-cpp-all = 64e83067e597c647bcc03dc8bdb47f20
```

blossom.cpp

Description: 1-indexed general matching
Time: $\mathcal{O}(NM)$, faster in practice

79 lines

```
struct MaxMatch {
    vi vis, par, orig, match, aux;
    int t, N;
    vector<vi> adj;
    queue<int> Q;

    MaxMatch(int n): vis(n + 1), par(n + 1), orig(n + 1),
        ↪match(n + 1), aux(n + 1), adj(n + 1), t(0), N(n)
        ↪{}

    void addEdge(int u, int v) {
        assert(u && v);
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void augment(int u, int v) {
        int pv = v, nv;
        do {
            pv = par[v]; nv = match[pv];
            match[v] = pv; match[pv] = v;
            v = nv;
        } while(u != pv);
    }

    int lca(int v, int w) {
        ++t;
        while (1) {
            if (v) {
                if (aux[v] == t) return v; aux[v] = t;
                v = orig[par[match[v]]];
            }
            swap(v, w);
        }
    }

    void blossom(int v, int w, int a) {
        while (orig[v] != a) {
            par[v] = w; w = match[v];
            if (vis[w] == 1) Q.push(w), vis[w] = 0;
            orig[v] = orig[w] = a;
            v = par[w];
        }
    }

    bool bfs(int u) {
        fill(vis.begin()+1, vis.begin()+1+N, -1);
        iota(orig.begin() + 1, orig.begin() + N + 1, 1);
        Q = queue<int> (); Q.push(u); vis[u] = 0;
        while (sz(Q)) {
            int v = Q.front(); Q.pop();
            trav(x,adj[v]) {
                if (vis[x] == -1) {
                    par[x] = v; vis[x] = 1;
                    if (!match[x]) return augment(u, x),
                        ↪true;
                    Q.push(match[x]); vis[match[x]] = 0;
                } else if (vis[x] == 0 && orig[v] != orig[x
                    ↪]) {
                    int a = lca(orig[v], orig[x]);
                    blossom(x, v, a); blossom(v, x, a);
                }
            }
        }
        return false;
    }

    int Match() {
        int ans = 0;
        // find random matching (not necessary, constant
            ↪improvement)
        vi V(N-1); iota(all(V), 1);
        shuffle(all(V), mt19937(233333));
        trav(x,V) if(!match[x])
            trav(y,adj[x]) if (!match[y]) {
            match[x] = y, match[y] = x;
            ++ans; break;
        }

        rep(i,1,N+1) if (!match[i] && bfs(i)) ++ans;
        return ans;
    }
}; // hash-cpp-all = ef6d7ea113cecafb8938a3943e758f22
```

6.3 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: $\mathcal{O}(E+V)$

24 lines

```
vi val, comp, z, cont;
int Time, ncomps;
template<class G, class F> int dfs(int j, G& g, F f) {
  int low = val[j] = ++Time, x; z.push_back(j);
  trav(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);
} // hash-cpp-all = 2c7a153ddd31436517cf3ad28efa4ac5
```

SCC2.h

Description: Finding the SCCs of the graph (Kosaraju's algorithm), cmp is the topo order of the SCCs

34 lines

```
struct SCC {
    vector<vi> G, rG;
    vi vs, used, cmp;
    int n;
    SCC(int n): G(n), rG(n), vs(n), used(n), cmp(n), n(n){}

    void add_edge(int from, int to){
        G[from].push_back(to);
        rG[to].push_back(from);
    }

    void dfs(int v){
        used[v] = true;
        for(int nxt : G[v]) if(!used[nxt]) dfs(nxt);
        vs.push_back(v);
    }

    void rdfs(int v, int k){
        used[v] = true;
        cmp[v] = k;
        for(int nxt : rG[v]) if(!used[nxt]) rdfs(nxt, k);
    }

    int scc(){
        fill(all(used), 0);
        vs.clear();
        rep(v, 0, n) if(!used[v]) dfs(v);
        fill(all(used), 0);
        int k = 0;
        reverse(all(vs));
        for(int v : vs) if(!used[v]) rdfs(v, k++);
        return k;
    }
}; // hash-cpp-all = 40039bc23103a6a7b277e3a6fad934a5
```

BiconnectedComponents.h

Time: $\mathcal{O}(E+V)$

37 lines

```
int cur, num[maxn], low[maxn];
int sz;
vector<int> com[maxn];
void tarjan(int u, int last) {
    num[u] = low[u] = ++cur;
    st.push(u); // vertex
    for(auto tmp : way[u]) {
        int v = tmp.first, id = tmp.second;
        if(v == last) continue; // if(id == last)
        if(!num[v]) {
            // st.push(id);
            tarjan(v, u); // tarjan(v, id)
            low[u] = min(low[u], low[v]);
            /* if(low[v] >= num[u]) {
                sz++;
```

```
                while(1) {
                    int x = st.top(); st.pop();
                    com[sz].push_back(x);
                    if(x == id) break;
                }
            } */
        }
        else low[u] = min(low[u], num[v]);
        /* else if(num[v] < num[u]) {
            st.push(id);
            low[u] = min(low[u], num[v]);
        } */
    }
    if(num[u] == low[u]) { // vertex
        sz++;
        while(1) {
            int x = st.top(); st.pop();
            com[sz].push_back(x);
            if(x == u) break;
        }
    }
} // hash-cpp-all = 3eaf49bccded1e3dca97f788e75bea4b
```

2sat.h

25 lines

```
struct SAT {
    SCC scc;
    int nvars;
    SAT (int nvars): scc(nvars * 2), nvars(nvars) {
    }
    void add(int a, int x, int b, int y) { // x/y is 1 if
       ↪true
        scc.add_edge(a + x * nvars, b + y * nvars);
        scc.add_edge(b + (!y) * nvars, a + (!x) * nvars);
    }

    void equiv(int a, int b) {
        add(a, 1, b, 1);
        add(a, 0, b, 0);
    }

    vi solve() {
        scc.scc();
        vi res(nvars);
        rep(i, 0, nvars) {
            if(scc.cmp[i] == scc.cmp[i + nvars]) return
               ↪{-1};
            else res[i] = scc.cmp[i + nvars] > scc.cmp[i];
               ↪// 1 if i is true
        }
        return res;
    }
}; // hash-cpp-all = c5493f91d22ac1339e4fb8960a5924c3
```

6.4 Heuristics

6.5 Trees

LCA.h

Description: LCA via binary lifting.
Time: $\mathcal{O}(N \log N)$ time to build, $\mathcal{O}(\log N)$ per query.

50 lines

```
typedef vector<pii> vpi;
typedef vector<vpi> graph;

struct LCA {
    vector<vi> parent;
    graph g;
    vi dep;
    vector<ll> dis;
    int lg, n;

    LCA(graph& g): n(sz(g)), dep(sz(g)), dis(sz(g)), g(g) {
        int on = 1;
        lg = 1;
        while (on < n) on *= 2, lg++;
        parent.assign(lg, dep);
        dfs(0, -1, 0, 0); // rooted at 0
        rep(k, 0, lg - 1) {
            rep(v, 0, n) {
                parent[k + 1][v] = parent[k][v] < 0 ? -1 :
                   ↪parent[k][parent[k][v]];
            }
        }
    }

    void dfs(int v, int p, int d, ll d2){
        parent[0][v] = p, dep[v] = d, dis[v] = d2;
        for(auto &e : g[v]){
            if(e.first != p) dfs(e.first, v, d + 1, d2 + e.
               ↪second);
        }
    }

    int lca(int u, int v){
        if(dep[u] > dep[v]) swap(u, v);
        rep(k, 0, lg) {
            if ((dep[v] - dep[u]) >> k & 1) v = parent[k][v
               ↪];
        }
        if(u == v) return u;
        for (int k = lg - 1; k >= 0; k--) {
            if(parent[k][u] != parent[k][v]) {
                u = parent[k][u];
                v = parent[k][v];
            }
        }
        return parent[0][u];
    }

    ll distance(int a, int b) {
        int ca = lca(a, b);
        return dis[a] + dis[b] - 2 * dis[ca];
    }
}; // hash-cpp-all = e68b2d5b1ef831f39e6c4012ef7b2d1d
```

LCA2.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. Can also find the distance between two nodes.
Usage: `LCA lca(undirGraph);`
`lca.query(firstNode, secondNode);`
`lca.distance(firstNode, secondNode);`
Time: $\mathcal{O}(N \log N)$ time to build, $\mathcal{O}(1)$ per query.

"../data-structures/RMQ.h" 37 lines

```
typedef vector<pii> vpi;
typedef vector<vpi> graph;

struct LCA {
    vi time;
    vector<ll> dist;
    RMQ<pii> rmq;
    LCA(graph& C) : time(sz(C), -99), dist(sz(C)), rmq(dfs(
       ↪C)) {}

    vpi dfs(graph& C) {
        vector<tuple<int, int, int, ll>> q(1);
        vpi ret;
        int T = 0, v, p, d; ll di;
        while (!q.empty()) {
            tie(v, p, d, di) = q.back();
            q.pop_back();
            if (d) ret.emplace_back(d, p);
            time[v] = T++;
            dist[v] = di;
            trav(e, C[v]) if (e.first != p)
                    q.emplace_back(e.first, v, d+1, di + e.
                       ↪second);
        }
        return ret;
    }

    int query(int a, int b) {
        if (a == b) return a;
        a = time[a], b = time[b];
        return rmq.query(min(a, b), max(a, b) - 1).second;
    }
    ll distance(int a, int b) {
        int lca = query(a, b);
        return dist[a] + dist[b] - 2 * dist[lca];
    }
}; // hash-cpp-all = 27bd760b53475aea296925a1f8dc7ba6
```

CompressTree.h

Time: $\mathcal{O}(|S| \log |S|)$

36 lines

```
LCA lca(g);
vi vir;
vi stk(n + 1, 0);

auto add_edge = [&](int u, int v, int w) {
    w = lca.distance(u, v);
    g2[u].push_back({v, w});
    g2[v].push_back({u, w});
};

auto build_virtual_tree = [&](vi S) {
    for(auto u : vir) g2[u].clear();
    vir.clear();
    vi S;
    sort(all(S), [&](int u,int v) {return lca.time[u] < lca
       ↪.time[v];});
    S.resize(unique(all(S)) - S.begin());
    int sz = 0;
    stk[++sz] = 0; // root
    for(auto u : S) {
        int x = lca.query(u, stk[sz]);
        vir.push_back(u); vir.push_back(x);
        if(u == stk[sz]) continue;
        if(x != stk[sz]) {
            while(sz >= 2 && lca.dep[stk[sz-1]] >= lca.dep[
               ↪x]) {
                add_edge(stk[sz-1], stk[sz]);
                sz--;
            }
            if(x != stk[sz]) {
                add_edge(x, stk[sz]);
                stk[sz] = x;
            }
```

```
        }
        stk[++sz] = u;
    }
    for(int i=1;i<=sz-1;i++) add_edge(stk[i], stk[i+1]);
}; // hash-cpp-all = 4d34e76e329b9efb631f916462602083
```

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. The function of the HLD can be changed by modifying T, LOW and f. f is assumed to be associative and commutative.
Usage: `HLD hld(G);`
`hld.update(index, value);`
`tie(value, lca) = hld.query(n1, n2);`

56 lines

```
typedef vector<vi> graph;

struct HLD {
    int n, chainNo, tim;
    Tree tree;
    vi top, din, dout, ord, dep, sub, par;
    graph g;
    HLD(graph &g, int rt = 0): g(g), tree(sz(g)) {
        n = sz(g);
        top = din = dout = ord = dep = sub = par = vi(n);
        chainNo = tim = 0;
        fill(all(din), -1);
        dfs(rt, -1, 0);
        hld(rt, -1, rt);
    }

    void dfs(int v, int p, int d){
        par[v] = p;
        sub[v] = 1;
        dep[v] = d;
        for(int &nxt : g[v]){
            if(nxt == p) continue;
            dfs(nxt, v, d + 1);
            sub[v] += sub[nxt];
            if(g[v][0] == p || sub[nxt] > sub[g[v][0]])
                swap(nxt, g[v][0]);
        }
    }

    void hld(int v, int p, int tp) {
        top[v] = tp;
        din[v] = tim;
        ord[tim++] = v;
        for (auto nxt: g[v]) {
            if (nxt == p) continue;
            if (nxt != g[v][0]) {
                chainNo++; hld(nxt, v, nxt);
            } else hld(nxt, v, tp);
        }
        dout[v] = tim - 1;
    }

    ll query(int a, int b) {
        ll res = 0;
        while(top[a] != top[b]) {
            if(dep[top[a]] < dep[top[b]]) swap(a, b);
            (res += tree.qry(din[top[a]], din[a])) %= mod;
            a = par[top[a]];
        }
        if(din[a] > din[b]) swap(a, b);
        // if querying edge, use tree.qry(din[a] + 1, din[b
           ↪])
        // lca is variable a
        (res += tree.qry(din[a], din[b])) %= mod;
        return res;
    }
}; // hash-cpp-all = 2bacac44cb38efe1f58a26a485c0d256
```

LinkCutTree.h

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

90 lines

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

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

  void link(int u, int v) { // add an edge (u, v)
    assert(!connected(u, v));
    make_root(&node[u]);
    node[u].pp = &node[v];
  }
  void cut(int u, int v) { // remove an edge (u, v)
    Node *x = &node[u], *top = &node[v];
    make_root(top); x->splay();
    assert(top == (x->pp ?: x->c[0]));
    if (x->pp) x->pp = 0;
    else {
      x->c[0] = top->p = 0;
      x->fix();
    }
  }
  bool connected(int u, int v) { // are u, v in the same
    ↪tree?
    Node* nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
  }
  void make_root(Node* u) {
    access(u);
    u->splay();
    if(u->c[0]) {
      u->c[0]->p = 0;
      u->c[0]->flip ^= 1;
      u->c[0]->pp = u;
      u->c[0] = 0;
      u->fix();
    }
  }
  Node* access(Node* u) {
    u->splay();
    while (Node* pp = u->pp) {
      pp->splay(); u->pp = 0;
      if (pp->c[1]) {
        pp->c[1]->p = 0; pp->c[1]->pp = pp; }
      pp->c[1] = u; pp->fix(); u = pp;
    }
    return u;
  }
}; // hash-cpp-all = 693483a825b93f7ad5f6ef219ba83e22
```

Centroid.cpp

29 lines

```
set<int> G[N];
int depth[N], par[N], sub[N];

int dfs1(int v, int p){
    sub[v] = 1;
    for(int nxt : G[v])
        if(nxt != p)
            sub[v] += dfs1(nxt, v);
    return sub[v];
}

int dfs2(int v, int p, int nn){
    for(int nxt : G[v]){
        if(nxt != p && sub[nxt] > nn/2) return dfs2(nxt, v,
           ↪ nn);
    }
    return v;
}

void decompose(int v, int p){
    dfs1(v, -1);
    int centroid = dfs2(v, -1, sub[v]);
    par[centroid] = p;
    for(int nxt : G[centroid]){
        G[nxt].erase(centroid);
        decompose(nxt, centroid);
```

```
    }
    G[centroid].clear();
} // hash-cpp-all = e5955539d9903a4299626c2df9ddc081
```

MatrixTree.h

Description: To count the number of spanning trees in an undirected graph G: create an $N \times N$ matrix `mat`, and for each edge $(a, b) \in G$, do `mat[a][a]++, mat[b][b]++, mat[a][b]--, mat[b][a]--`. Remove the last row and column, and take the determinant.

1 lines

```
// hash-cpp-all = d41d8cd98f00b204e9800998ecf8427e
```

6.6 Other

directed-MST.cpp

Description: Finds the minimum spanning arborescence from the root. (any more notes?)

73 lines

```
#define rep(i, n) for (int i = 0; i < n; i++)

#define N 110000
#define M 110000
#define inf 2000000000

struct edg {
    int u, v;
    int cost;
} E[M], E_copy[M];

int In[N], ID[N], vis[N], pre[N];

// edges pointed from root.
int Directed_MST(int root, int NV, int NE) {
  for (int i = 0; i < NE; i++)
    E_copy[i] = E[i];
    int ret = 0;
    int u, v;
    while (true) {
        rep(i, NV)   In[i] = inf;
        rep(i, NE) {
            u = E_copy[i].u;
            v = E_copy[i].v;
            if(E_copy[i].cost < In[v] && u != v) {
                In[v] = E_copy[i].cost;
                pre[v] = u;
            }
        }
        rep(i, NV) {
            if(i == root)   continue;
            if(In[i] == inf)    return -1; // no solution
        }

        int cnt = 0;
        rep(i, NV) {
          ID[i] = -1;
          vis[i] = -1;
        }
        In[root] = 0;

        rep(i, NV) {
            ret += In[i];
            int v = i;
            while(vis[v] != i && ID[v] == -1 && v != root)
                {
                vis[v] = i;
                v = pre[v];
            }
            if(v != root && ID[v] == -1) {
                for(u = pre[v]; u != v; u = pre[u]) {
                    ID[u] = cnt;
                }
                ID[v] = cnt++;
            }
        }
        if(cnt == 0)    break;
        rep(i, NV) {
            if(ID[i] == -1) ID[i] = cnt++;
        }
        rep(i, NE) {
            v = E_copy[i].v;
            E_copy[i].u = ID[E_copy[i].u];
            E_copy[i].v = ID[E_copy[i].v];
            if(E_copy[i].u != E_copy[i].v) {
                E_copy[i].cost -= In[v];
            }
        }
        NV = cnt;
        root = ID[root];
    }
    return ret;
}
// hash-cpp-all = 84815c2bfececf3575ecf663c0703643
```

graph-dominator-tree.cpp

Description: Dominator Tree.

75 lines

```
struct Dominator{
    struct min_DSU{
        vector<int> par, val;
        vector<int> const&semi;
        min_DSU(int N, vector<int> const&semi):par(N, -1),
            ↪val(N), semi(semi){
            iota(val.begin(), val.end(), 0);
        }
        void comp(int x){
            if(par[par[x]]!=-1){
                comp(par[x]);
                if(semi[val[par[x]]]<semi[val[x]])
                    val[x] = val[par[x]];
                par[x]=par[par[x]];
            }
        }
        int f(int x){
            if(par[x]==-1) return x;
            comp(x);
            return val[x];
        }
        void link(int x, int p){
            par[x] = p;
        }
    };
    int N;
    vector<vector<int> > G, rG;
    vector<int> idom, order;
    Dominator(int _N):N(_N), G(N), rG(N){}
    void add_edge(int a, int b){
        G[a].emplace_back(b);
        rG[b].emplace_back(a);
    }
    vector<int> calc_dominators(int S){
        idom.assign(N, -1);
        vector<int> par(N, -1), semi(N, -1);
        vector<vector<int> > bu(N);
        stack<int> s;
        s.emplace(S);
        while(!s.empty()){
            int a=s.top();s.pop();
            if(semi[a]==-1){
                semi[a] = order.size();
                order.emplace_back(a);
                for(int i=0;i<(int)G[a].size();++i){
                    if(semi[G[a][i]]==-1){
                        par[G[a][i]]=a;
                        s.push(G[a][i]);
                    }
                }
            }
        }
        min_DSU uni(N, semi);
        for(int i=(int)order.size()-1;i>0;--i){
            int w=order[i];
            for(int f:rG[w]){
                int oval = semi[uni.f(f)];
                if(oval>=0 && semi[w]>oval) semi[w] = oval;
            }
            bu[order[semi[w]]].push_back(w);
            uni.link(w, par[w]);
            while(!bu[par[w]].empty()){
                int v = bu[par[w]].back(); bu[par[w]].
                    ↪pop_back();
                int u=uni.f(v);
                idom[v] = semi[u] < semi[v] ? u : par[w];
            }
        }
        for(int i=1;i<(int)order.size();++i){
            int w=order[i];
            if(idom[w] != order[semi[w]])
                idom[w] = idom[idom[w]];
        }
        idom[S]=-1;
        return idom; // immediate dominator
    }
}; // hash-cpp-all = bae330fd4a4204248302ec5b7ab40095
```

Geometry (7)

7.1 Geometric primitives

Point.h

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

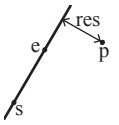
25 lines

```
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)); }
}; // hash-cpp-all = f698493d48eeeaa76063407bf935b5a3
```

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.

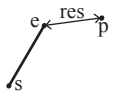


"Point.h" 4 lines

```
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
  return (double)(b-a).cross(p-a)/(b-a).dist();
} // hash-cpp-all = f6bf6b556d99b09f42b86d28d1eaa86d
```

SegmentDistance.h

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



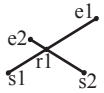
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

"Point.h" 6 lines

```
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
  if (s==e) return (p-s).dist();
  auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)))
    ↪;
  return ((p-s)*d-(e-s)*t).dist()/d;
} // hash-cpp-all = 5c88f46fb14a05a4f47bbd23b8a9c427
```

SegmentIntersection.h

Description:
If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists r1 is set to this point and 1 is returned. If no intersection point exists 0 is returned and if infinitely many exists 2 is returned and r1 and r2 are set to the two ends of the common line. The wrong position will be returned if P is Point<int> 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. Use segmentIntersectionQ to get just a true/false answer.



Usage: `Point<double> intersection, dummy;`
`if (segmentIntersection(s1,e1,s2,e2,intersection,dummy)==1)`
`cout << "segments intersect at " << intersection << endl;`

"Point.h" 27 lines

```
template<class P>
int segmentIntersection(const P& s1, const P& e1,
    const P& s2, const P& e2, P& r1, P& r2) {
  if (e1==s1) {
    if (e2==s2) {
      if (e1==e2) { r1 = e1; return 1; } //all equal
      else return 0; //different point segments
    } else return segmentIntersection(s2,e2,s1,e1,r1,r2);//
      ↪swap
  }
  //segment directions and separation
  P v1 = e1-s1, v2 = e2-s2, d = s2-s1;
  auto a = v1.cross(v2), a1 = v1.cross(d), a2 = v2.cross(d)
    ↪;
  if (a == 0) { //if parallel
    auto b1=s1.dot(v1), c1=e1.dot(v1),
         b2=s2.dot(v1), c2=e2.dot(v1);
    if (a1 || a2 || max(b1,min(b2,c2))>min(c1,max(b2,c2)))
      return 0;
    r1 = min(b2,c2)<b1 ? s1 : (b2<c2 ? s2 : e2);
    r2 = max(b2,c2)>c1 ? e1 : (b2>c2 ? s2 : e2);
    return 2-(r1==r2);
  }
  if (a < 0) { a = -a; a1 = -a1; a2 = -a2; }
  if (0<a1 || a<-a1 || 0<a2 || a<-a2)
    return 0;
  r1 = s1-v1*a2/a;
  return 1;
} // hash-cpp-all = 1181b7cc739b442c29bada6b0d73a550
```

SegmentIntersectionQ.h

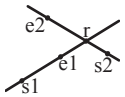
Description: Like segmentIntersection, but only returns true/false. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

"Point.h" 16 lines

```
template<class P>
bool segmentIntersectionQ(P s1, P e1, P s2, P e2) {
  if (e1 == s1) {
    if (e2 == s2) return e1 == e2;
    swap(s1,s2); swap(e1,e2);
  }
  P v1 = e1-s1, v2 = e2-s2, d = s2-s1;
  auto a = v1.cross(v2), a1 = d.cross(v1), a2 = d.cross(v2)
    ↪;
  if (a == 0) { // parallel
    auto b1 = s1.dot(v1), c1 = e1.dot(v1),
         b2 = s2.dot(v1), c2 = e2.dot(v1);
    return !a1 && max(b1,min(b2,c2)) <= min(c1,max(b2,c2));
  }
  if (a < 0) { a = -a; a1 = -a1; a2 = -a2; }
  return (0 <= a1 && a1 <= a && 0 <= a2 && a2 <= a);
} // hash-cpp-all = 1ff4ba22bd0aefb04bf48cca4d6a7d8c
```

lineIntersection.h

Description:
If a unique intersection point of the lines going through s1,e1 and s2,e2 exists r is set to this point and 1 is returned. If no intersection point exists 0 is returned and if infinitely many exists -1 is returned. If s1==e1 or s2==e2 -1 is returned. The wrong position will be returned if P is Point<int> 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: `point<double> intersection;`
`if (1 == LineIntersection(s1,e1,s2,e2,intersection))`
`cout << "intersection point at " << intersection << endl;`

"Point.h" 9 lines

```
template<class P>
int lineIntersection(const P& s1, const P& e1, const P& s2,
    const P& e2, P& r) {
  if ((e1-s1).cross(e2-s2)) { //if not parallell
    r = s2-(e2-s2)*(e1-s1).cross(s2-s1)/(e1-s1).cross(e2-s2
      ↪);
    return 1;
  } else
    return -((e1-s1).cross(s2-s1)==0 || s2==e2);
} // hash-cpp-all = aa1f17f0dbde5177e697038a420bb078
```

sideOf.h

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

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

"Point.h" 11 lines

```
template<class P>
int sideOf(const P& s, const P& e, const P& p) {
  auto a = (e-s).cross(p-s);
  return (a > 0) - (a < 0);
}
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);
} // hash-cpp-all = 2eb6fe62d7f3750fd3a0ec3d91329ed6
```

onSegment.h

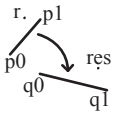
Description: Returns true iff p lies on the line segment from s to e. Intended for use with e.g. Point<long long> where overflow is an issue. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

"Point.h" 5 lines

```
template<class P>
bool onSegment(const P& s, const P& e, const P& p) {
  P ds = p-s, de = p-e;
  return ds.cross(de) == 0 && ds.dot(de) <= 0;
} // hash-cpp-all = 0b2b1c6866c98c2d2003acec0701e693
```

linearTransformation.h

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



"Point.h" 6 lines

```
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
  P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
  return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.
    ↪dist2();
} // hash-cpp-all = 03a3061b3ef024b4e29ea06169932b21
```

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`

37 lines

```
struct Angle {
```

```
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t
    ↪}; }
  int quad() const {
    assert(x || y);
    if (y < 0) return (x >= 0) + 2;
    if (y > 0) return (x <= 0);
    return (x <= 0) * 2;
  }
  Angle t90() const { return {-y, x, t + (quad() == 3)}; }
  Angle t180() const { return {-x, -y, t + (quad() >= 2)};
    ↪}
  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.quad(), a.y * (ll)b.x) <
         make_tuple(b.t, b.quad(), 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
    ↪)};
} // hash-cpp-all = 1856c5d371c2f8f342a22615fa92cd54
```

angleCmp.h

Description: Useful utilities for dealing with angles of rays from origin. OK for integers, only uses cross product. Doesn't support (0,0). 22 lines

```
template <class P>
bool sameDir(P s, P t) {
  return s.cross(t) == 0 && s.dot(t) > 0;
}
// checks 180 <= s..t < 360?
template <class P>
bool isReflex(P s, P t) {
  auto c = s.cross(t);
  return c ? (c < 0) : (s.dot(t) < 0);
}
// operator < (s,t) for angles in [base,base+2pi)
template <class P>
bool angleCmp(P base, P s, P t) {
  int r = isReflex(base, s) - isReflex(base, t);
  return r ? (r < 0) : (0 < s.cross(t));
}
// is x in [s,t] taken ccw? 1/0/-1 for in/border/out
template <class P>
int angleBetween(P s, P t, P x) {
  if (sameDir(x, s) || sameDir(x, t)) return 0;
  return angleCmp(s, x, t) ? 1 : -1;
} // hash-cpp-all = 6edd25f30f9c69989bbd2115b4fdceda
```

7.2 Circles

CircleIntersection.h

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

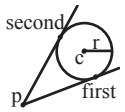
"Point.h" 14 lines

```
typedef Point<double> P;
bool circleIntersection(P a, P b, double r1, double r2,
    pair<P, P>* out) {
  P delta = b - a;
  assert(delta.x || delta.y || r1 != r2);
  if (!delta.x && !delta.y) return false;
  double r = r1 + r2, d2 = delta.dist2();
  double p = (d2 + r1*r1 - r2*r2) / (2.0 * d2);
  double h2 = r1*r1 - p*p*d2;
  if (d2 > r*r || h2 < 0) return false;
  P mid = a + delta*p, per = delta.perp() * sqrt(h2 / d2);
  *out = {mid + per, mid - per};
  return true;
} // hash-cpp-all = 828fbb1fff1469ed43b2284c8e07a06c
```

circleTangents.h

Description:

Returns a pair of the two points on the circle with radius r centered around c whos tangent lines intersect p. If p lies within the circle NaN-points are returned. P is intended to be Point<double>. The first point is the one to the right as seen from the p towards c.



Usage: `typedef Point<double> P;`
`pair<P,P> p = circleTangents(P(100,2),P(0,0),2);`

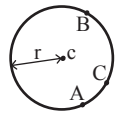
"Point.h" 6 lines

```
template<class P>
pair<P,P> circleTangents(const P &p, const P &c, double r)
  ↪{
  P a = p-c;
  double x = r*r/a.dist2(), y = sqrt(x-x*x);
  return make_pair(c+a*x+a.perp()*y, c+a*x-a.perp()*y);
} // hash-cpp-all = b70bc575e85c140131116e64926b4ce1
```

circumcircle.h

Description:

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



"Point.h" 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;
} // hash-cpp-all = 1caa3aea364671cb961900d4811f0282
```

MinimumEnclosingCircle.h

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

"circumcircle.h" 28 lines

```
pair<double, P> mec2(vector<P>& S, P a, P b, int n) {
  double hi = INFINITY, lo = -hi;
  rep(i,0,n) {
    auto si = (b-a).cross(S[i]-a);
    if (si == 0) continue;
    P m = ccCenter(a, b, S[i]);
    auto cr = (b-a).cross(m-a);
    if (si < 0) hi = min(hi, cr);
    else lo = max(lo, cr);
  }
  double v = (0 < lo ? lo : hi < 0 ? hi : 0);
  P c = (a + b) / 2 + (b - a).perp() * v / (b - a).dist2();
  return {(a - c).dist2(), c};
}
pair<double, P> mec(vector<P>& S, P a, int n) {
  random_shuffle(S.begin(), S.begin() + n);
  P b = S[0], c = (a + b) / 2;
  double r = (a - c).dist2();
  rep(i,1,n) if ((S[i] - c).dist2() > r * (1 + 1e-8)) {
    tie(r,c) = (n == sz(S) ?
      mec(S, S[i], i) : mec2(S, a, S[i], i));
  }
  return {r, c};
}
pair<double, P> enclosingCircle(vector<P> S) {
  assert(!S.empty()); auto r = mec(S, S[0], sz(S));
  return {sqrt(r.first), r.second};
} // hash-cpp-all = 9bf427c9626a72f805196e0b7075bda2
```

7.3 Polygons

insidePolygon.h

Description: Returns true if p lies within the polygon described by the points between iterators begin and end. If strict false is returned when p is on the edge of the polygon. Answer is calculated by counting the number of intersections between the polygon and a line going from p to infinity in the positive x-direction. The algorithm uses products in intermediate steps so watch out for overflow. If points within epsilon from an edge should be considered as on the edge replace the line "if (onSegment..." with the comment bellow it (this will cause overflow for int and long long).
Usage: `typedef Point<int> pi;`
`vector<pi> v; v.push_back(pi(4,4));`
`v.push_back(pi(1,2)); v.push_back(pi(2,1));`
`bool in = insidePolygon(v.begin(),v.end(), pi(3,4), false);`
Time: $\mathcal{O}(n)$

"Point.h", "onSegment.h", "SegmentDistance.h" 14 lines

```
template<class It, class P>
bool insidePolygon(It begin, It end, const P& p,
    bool strict = true) {
  int n = 0; //number of isects with line from p to (inf,p.
    ↪y)
  for (It i = begin, j = end-1; i != end; j = i++) {
    //if p is on edge of polygon
    if (onSegment(*i, *j, p)) return !strict;
    //or: if (segDist(*i, *j, p) <= epsilon) return !strict
      ↪;
    //increment n if segment intersects line from p
    n += (max(i->y,j->y) > p.y && min(i->y,j->y) <= p.y &&
        ((*j-*i).cross(p-*i) > 0) == (i->y <= p.y));
  }
  return n&1; //inside if odd number of intersections
} // hash-cpp-all = 0cadec56a74f257b8d1b25f56ba7ebad
```

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" 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;
} // hash-cpp-all = f123003799a972c1292eb0d8af7e37da
```

PolygonCenter.h

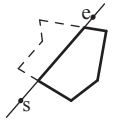
Description: Returns the center of mass for a polygon.

"Point.h" 10 lines

```
typedef Point<double> P;
Point<double> polygonCenter(vector<P>& v) {
  auto i = v.begin(), end = v.end(), j = end-1;
  Point<double> res{0,0}; double A = 0;
  for (; i != end; j=i++) {
    res = res + (*i + *j) * j->cross(*i);
    A += j->cross(*i);
  }
  return res / A / 3;
} // hash-cpp-all = d210bd2372832f7d074894d904e548ab
```

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" 15 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.emplace_back();
      lineIntersection(s, e, cur, prev, res.back());
    }
    if (side)
      res.push_back(cur);
  }
  return res;
} // hash-cpp-all = acf5106be46aa8f6f5d7a8d0ffdaae3c
```

ConvexHull.h

Description:
Returns a vector of indices 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.

Usage: vector<P> ps, hull;
trav(i, convexHull(ps)) hull.push_back(ps[i]);
Time: $\mathcal{O}(n \log n)$

"Point.h" 20 lines

```
typedef Point<ll> P;
pair<vi, vi> ulHull(const vector<P>& S) {
  vi Q(sz(S)), U, L;
  iota(all(Q), 0);
  sort(all(Q), [&S](int a, int b){ return S[a] < S[b]; });
  trav(it, Q) {
#define ADDP(C, cmp) while (sz(C) > 1 && S[C[sz(C)-2]].
  ↪cross(\
    S[it], S[C.back()]) cmp 0) C.pop_back(); C.push_back(it);
    ADDP(U, <=); ADDP(L, >=);
  }
  return {U, L};
}

vi convexHull(const vector<P>& S) {
  vi u, l; tie(u, l) = ulHull(S);
  if (sz(S) <= 1) return u;
  if (S[u[0]] == S[u[1]]) return {0};
  l.insert(l.end(), u.rbegin()+1, u.rend()-1);
  return l;
} // hash-cpp-all = d1b691dc7571b8460911ebe2e4023806
```

PolygonDiameter.h

Description: Calculates the max squared distance of a set of points.

"ConvexHull.h" 19 lines

```
vector<pii> antipodal(const vector<P>& S, vi& U, vi& L) {
  vector<pii> ret;
  int i = 0, j = sz(L) - 1;
  while (i < sz(U) - 1 || j > 0) {
    ret.emplace_back(U[i], L[j]);
    if (j == 0 || (i != sz(U)-1 && (S[L[j]] - S[L[j-1]])
          .cross(S[U[i+1]] - S[U[i]]) > 0)) ++i;
    else --j;
  }
  return ret;
}

pii polygonDiameter(const vector<P>& S) {
  vi U, L; tie(U, L) = ulHull(S);
  pair<ll, pii> ans;
  trav(x, antipodal(S, U, L))
    ans = max(ans, {(S[x.first] - S[x.second]).dist2(), x})
      ↪;
  return ans.second;
} // hash-cpp-all = 5596d386362874d2ebcf13cdb142574d
```

PointInsideHull.h

Description: Determine whether a point t lies inside a given polygon (counter-clockwise order). The polygon must be such that every point on the circumference is visible from the first point in the vector. It returns 0 for points outside, 1 for points on the circumference, and 2 for points inside.
Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "onSegment.h" 22 lines

```
typedef Point<ll> P;
int insideHull2(const vector<P>& H, int L, int R, const P&
  ↪p) {
  int len = R - L;
  if (len == 2) {
    int sa = sideOf(H[0], H[L], p);
    int sb = sideOf(H[L], H[L+1], p);
    int sc = sideOf(H[L+1], H[0], p);
    if (sa < 0 || sb < 0 || sc < 0) return 0;
    if (sb==0 || (sa==0 && L == 1) || (sc == 0 && R == sz(H
      ↪)))
      return 1;
    return 2;
  }
  int mid = L + len / 2;
  if (sideOf(H[0], H[mid], p) >= 0)
    return insideHull2(H, mid, R, p);
  return insideHull2(H, L, mid+1, p);
}
int insideHull(const vector<P>& hull, const P& p) {
  if (sz(hull) < 3) return onSegment(hull[0], hull.back(),
    ↪p);
  else return insideHull2(hull, 1, sz(hull), p);
} // hash-cpp-all = 1c16dba23109ced37b95769a3f1d19b7
```

LineHullIntersection.h

Description: Line-convex polygon intersection. The polygon must be ccw and have no colinear points. isct(a, b) returns a pair describing the intersection of a line with the polygon: $\bullet$ $(-1,-1)$ if no collision, $\bullet$ $(i,-1)$ if touching the corner i, $\bullet$ (i,i) if along side $(i,i+1)$, $\bullet$ (i,j) if crossing sides $(i,i+1)$ and $(j,j+1)$. In the last case, if a corner i is crossed, this is treated as happening on side $(i,i+1)$. The points are returned in the same order as the line hits the polygon.
Time: $\mathcal{O}(N + Q \log n)$

"Point.h" 63 lines

```
ll sgn(ll a) { return (a > 0) - (a < 0); }
typedef Point<ll> P;
struct HullIntersection {
  int N;
  vector<P> p;
  vector<pair<P, int>> a;

  HullIntersection(const vector<P>& ps) : N(sz(ps)), p(ps)
    ↪{
    p.insert(p.end(), all(ps));
    int b = 0;
    rep(i,1,N) if (P{p[i].y,p[i].x} < P{p[b].y, p[b].x}) b
      ↪= i;
    rep(i,0,N) {
      int f = (i + b) % N;
      a.emplace_back(p[f+1] - p[f], f);
    }
  }

  int qd(P p) {
    return (p.y < 0) ? (p.x >= 0) + 2
         : (p.x <= 0) * (1 + (p.y <= 0));
  }

  int bs(P dir) {
    int lo = -1, hi = N;
    while (hi - lo > 1) {
      int mid = (lo + hi) / 2;
      if (make_pair(qd(dir), dir.y * a[mid].first.x) <
        make_pair(qd(a[mid].first), dir.x * a[mid].first.y)
          ↪)
        hi = mid;
      else lo = mid;
    }
    return a[hi%N].second;
  }

  bool isign(P a, P b, int x, int y, int s) {
    return sgn(a.cross(p[x], b)) * sgn(a.cross(p[y], b)) ==
      ↪ s;
  }

  int bs2(int lo, int hi, P a, P b) {
    int L = lo;
    if (hi < lo) hi += N;
    while (hi - lo > 1) {
      int mid = (lo + hi) / 2;
      if (isign(a, b, mid, L, -1)) hi = mid;
      else lo = mid;
    }
    return lo;
```

```
  }

  pii isct(P a, P b) {
    int f = bs(a - b), j = bs(b - a);
    if (isign(a, b, f, j, 1)) return {-1, -1};
    int x = bs2(f, j, a, b)%N,
        y = bs2(j, f, a, b)%N;
    if (a.cross(p[x], b) == 0 &&
        a.cross(p[x+1], b) == 0) return {x, x};
    if (a.cross(p[y], b) == 0 &&
        a.cross(p[y+1], b) == 0) return {y, y};
    if (a.cross(p[f], b) == 0) return {f, -1};
    if (a.cross(p[j], b) == 0) return {j, -1};
    return {x, y};
  }
}; // hash-cpp-all = 79decd52fd801714ccebbaa6ab36151e
```

halfPlane.h

Description: Halfplane intersection area

"Point.h", "lineIntersection.h" 76 lines

```
#define eps 1e-8
typedef Point<double> P;

struct Line {
  P P1, P2;
  // Right hand side of the ray P1 -> P2
  explicit Line(P a = P(), P b = P()) : P1(a), P2(b) {};
  P intpo(Line y) {
    P r;
    assert(lineIntersection(P1, P2, y.P1, y.P2, r) == 1);
    return r;
  }
  P dir() {
    return P2 - P1;
  }
  bool contains(P x) {
    return (P2 - P1).cross(x - P1) < eps;
  }
  bool out(P x) {
    return !contains(x);
  }
};

template<class T>
bool mycmp(Point<T> a, Point<T> b) {
  // return atan2(a.y, a.x) < atan2(b.y, b.x);
  if (a.x * b.x < 0)  return a.x < 0;
  if (abs(a.x) < eps) {
    if (abs(b.x) < eps)   return a.y > 0 && b.y < 0;
    if (b.x < 0)  return a.y > 0;
    if (b.x > 0)  return true;
  }
  if (abs(b.x) < eps) {
    if (a.x < 0)  return b.y < 0;
    if (a.x > 0)  return false;
  }
  return a.cross(b) > 0;
}

bool cmp(Line a, Line b) {
  return mycmp(a.dir(), b.dir());
}

double Intersection_Area(vector <Line> b) {
  sort(b.begin(), b.end(), cmp);
  int n = b.size();
  int q = 1, h = 0, i;
  vector <Line> c(b.size() + 10);
  for (i = 0; i < n; i++) {
    while (q < h && b[i].out(c[h].intpo(c[h - 1]))) h--;
    while (q < h && b[i].out(c[q].intpo(c[q + 1]))) q++;
    c[++h] = b[i];
    if (q < h && abs(c[h].dir().cross(c[h - 1].dir())) <
      ↪eps) {
      if (c[h].dir().dot(c[h - 1].dir()) > 0) {
        h--;
        if (b[i].out(c[h].P1))  c[h] = b[i];
      }else {
        // The area is either 0 or infinite.
        // If you have a bounding box, then the area is
          ↪definitely 0.
        return 0;
      }
    }
  }
  while (q < h - 1 && c[q].out(c[h].intpo(c[h - 1]))) h--;
  while (q < h - 1 && c[h].out(c[q].intpo(c[q + 1]))) q++;
  // Intersection is empty. This is sometimes different
    ↪from the case when
  // the intersection area is 0.
  if (h - q <= 1) return 0;
  c[h + 1] = c[q];
  vector <P> s;
  for (i = q; i <= h; i++)  s.push_back(c[i].intpo(c[i +
    ↪1]));
  s.push_back(s[0]);
  double ans = 0;
  for (i = 0; i < (int) s.size() - 1; i++)  ans += s[i].
    ↪cross(s[i + 1]);
  return ans / 2;
} // hash-cpp-all = 5aff1aff2ef04bf0df442d6c353ea924
```

7.4 Misc. Point Set Problems

closestPair.h

Description: $i1$, $i2$ are the indices to the closest pair of points in the point vector p after the call. The distance is returned.
Time: $\mathcal{O}(n \log n)$

"Point.h" 58 lines

```
template<class It>
bool it_less(const It& i, const It& j) { return *i < *j; }
template<class It>
bool y_it_less(const It& i,const It& j) {return i->y < j->y
  ↪;}

template<class It, class IIt> /* IIt = vector<It>::iterator
  ↪ */
double cp_sub(IIt ya, IIt yaend, IIt xa, It &i1, It &i2) {
  typedef typename iterator_traits<It>::value_type P;
  int n = yaend-ya, split = n/2;
  if(n <= 3) { // base case
    double a = (*xa[1]-*xa[0]).dist(), b = 1e50, c = 1e50;
    if(n==3) b=(*xa[2]-*xa[0]).dist(), c=(*xa[2]-*xa[1]).
      ↪dist();
    if(a <= b) { i1 = xa[1];
      if(a <= c) return i2 = xa[0], a;
      else return i2 = xa[2], c;
    } else { i1 = xa[2];
      if(b <= c) return i2 = xa[0], b;
      else return i2 = xa[1], c;
  } }
  vector<It> ly, ry, stripy;
  P splitp = *xa[split];
  double splitx = splitp.x;
  for(IIt i = ya; i != yaend; ++i) { // Divide
    if(*i != xa[split] && (**i-splitp).dist2() < 1e-12)
      return i1 = *i, i2 = xa[split], 0;// nasty special
        ↪case!
    if (**i < splitp) ly.push_back(*i);
    else ry.push_back(*i);
  } // assert((signed)lefty.size() == split)
  It j1, j2; // Conquer
  double a = cp_sub(ly.begin(), ly.end(), xa, i1, i2);
  double b = cp_sub(ry.begin(), ry.end(), xa+split, j1, j2)
    ↪;
  if(b < a) a = b, i1 = j1, i2 = j2;
  double a2 = a*a;
  for(IIt i = ya; i != yaend; ++i) { // Create strip (y-
    ↪sorted)
    double x = (*i)->x;
    if(x >= splitx-a && x <= splitx+a) stripy.push_back(*i)
      ↪;
  }
  for(IIt i = stripy.begin(); i != stripy.end(); ++i) {
    const P &p1 = **i;
    for(IIt j = i+1; j != stripy.end(); ++j) {
      const P &p2 = **j;
      if(p2.y-p1.y > a) break;
      double d2 = (p2-p1).dist2();
      if(d2 < a2) i1 = *i, i2 = *j, a2 = d2;
  } }
  return sqrt(a2);
}

template<class It> // It is random access iterators of
  ↪point<T>
double closestpair(It begin, It end, It &i1, It &i2 ) {
  vector<It> xa, ya;
  assert(end-begin >= 2);
  for (It i = begin; i != end; ++i)
    xa.push_back(i), ya.push_back(i);
  sort(xa.begin(), xa.end(), it_less<It>);
  sort(ya.begin(), ya.end(), y_it_less<It>);
  return cp_sub(ya.begin(), ya.end(), xa.begin(), i1, i2);
} // hash-cpp-all = 42735b8e08701a3b73504ac0690e31df
```

kdTree.h

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

"Point.h" 63 lines

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

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

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

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

  Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
      x0 = min(x0, p.x); x1 = max(x1, p.x);
```

```
    y0 = min(y0, p.y); y1 = max(y1, p.y);
  }
  if (vp.size() > 1) {
    // split on x if the box is wider than high (not best
       ↪ heuristic...)
    sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
    // divide by taking half the array for each child (
       ↪not
    // best performance with many duplicates in the
       ↪middle)
    int half = sz(vp)/2;
    first = new Node({vp.begin(), vp.begin() + half});
    second = new Node({vp.begin() + half, vp.end()});
  }
 }
};

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

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

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

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

  // find nearest point to a point, and its squared
     ↪distance
  // (requires an arbitrary operator< for Point)
  pair<T, P> nearest(const P& p) {
    return search(root, p);
  }
}; // hash-cpp-all = bac5b0409b201c3b040301344a40dc31
```

DelaunayTriangulation.h

Description: Computes the Delaunay triangulation of a set of points. Each circumcircle contains none of the input points. If any three points are colinear or any four are on the same circle, behavior is undefined.
Time: $\mathcal{O}(n^2)$

"Point.h", "3dHull.h" 10 lines

```
template<class P, class F>
void delaunay(vector<P>& ps, F trifun) {
  if (sz(ps) == 3) { int d = (ps[0].cross(ps[1], ps[2]) <
     ↪0);
    trifun(0,1+d,2-d); }
  vector<P3> p3;
  trav(p, ps) p3.emplace_back(p.x, p.y, p.dist2());
  if (sz(ps) > 3) trav(t, hull3d(p3)) if ((p3[t.b]-p3[t.a])
     ↪.
      cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
    trifun(t.a, t.c, t.b);
} // hash-cpp-all = d173fc69317d23d87be99189086af6d2
```

FastDelaunay.h

Description: Fast Delaunay triangulation. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order {t[0][0], t[0][1], t[0][2], t[1][0], ...}, all counter-clockwise.
Time: $\mathcal{O}(n \log n)$

"Point.h" 90 lines

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

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

bool circ(P p, P a, P b, P c) { // is p in the circumcircle
   ↪?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B >
     ↪ 0;
}
Q makeEdge(P orig, P dest) {
  Q q0 = new Quad{0,0,0,orig}, q1 = new Quad{0,0,0,arb},
    q2 = new Quad{0,0,0,dest}, q3 = new Quad{0,0,0,arb};
  q0->o = q0; q2->o = q2; // 0-0, 2-2
  q1->o = q3; q3->o = q1; // 1-3, 3-1
  q0->rot = q1; q1->rot = q2;
  q2->rot = q3; q3->rot = q0;
  return q0;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

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

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
  Q A, B, ra, rb;
  int half = (sz(s) + 1) / 2;
  tie(ra, A) = rec({s.begin(), s.begin() + half});
  tie(B, rb) = rec({s.begin() + half, s.end()});
  while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
         (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;

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

vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  Q e = rec(pts).first;
  vector<Q> q = {e};
  int qi = 0;
  while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p
   ↪); \
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  return pts;
} // hash-cpp-all = bfb5deb6acc9a794f45978d08f765fbe
```

7.5 3D

PolyhedronVolume.h

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

6 lines

```
template<class V, class L>
double signed_poly_volume(const V& p, const L& trilist) {
  double v = 0;
  trav(i, trilist) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
  return v / 6;
} // hash-cpp-all = 1ec4d393ab307cedc3866534eaa83a0e
```

Point3D.h

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

32 lines

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

```
  P cross(R p) const {
    return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
  }
  T dist2() const { return x*x + y*y + z*z; }
  double dist() const { return sqrt((double)dist2()); }
  //Azimuthal angle (longitude) to x-axis in interval [-pi,
     ↪ pi]
  double phi() const { return atan2(y, x); }
  //Zenith angle (latitude) to the z-axis in interval [0,
     ↪pi]
  double theta() const { return atan2(sqrt(x*x+y*y),z); }
  P unit() const { return *this/(T)dist(); } //makes dist()
     ↪=1
  //returns unit vector normal to *this and p
  P normal(P p) const { return cross(p).unit(); }
  //returns point rotated 'angle' radians ccw around axis
  P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.unit
       ↪();
    return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
  }
}; // hash-cpp-all = 8058aeda36daf3cba079c7bb0b43dcea
```

3dHull.h

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

"Point3D.h" 49 lines

```
typedef Point3D<double> P3;

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

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

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

  rep(i,4,sz(A)) {
    rep(j,0,sz(FS)) {
      F f = FS[j];
      if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
        E(a,b).rem(f.c);
        E(a,c).rem(f.b);
        E(b,c).rem(f.a);
        swap(FS[j--], FS.back());
        FS.pop_back();
      }
    }
    int nw = sz(FS);
    rep(j,0,nw) {
      F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f
   ↪.c);
      C(a, b, c); C(a, c, b); C(b, c, a);
    }
  }
  trav(it, FS) if ((A[it.b] - A[it.a]).cross(
    A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
  return FS;
}; // hash-cpp-all = c172e9f2cb6b44ceca0c416fee81f1dc
```

sphericalDistance.h

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

8 lines

```
double sphericalDistance(double f1, double t1,
    double f2, double t2, double radius) {
  double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
  double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
  double dz = cos(t2) - cos(t1);
  double d = sqrt(dx*dx + dy*dy + dz*dz);
  return radius*2*asin(d/2);
} // hash-cpp-all = 611f0797307c583c66413c2dd5b3ba28
```

Strings (8)

AhoCorasick.cpp

Time: Init, and insert strings, and then build. Remember to propagate information to from fail link.

52 lines

```
const int M = (int)5e5 + 500;

struct Trie {
    static const int B = 26;
    int next[M][B], fail[M], end[M];
    int root, L;

    int newnode() {
        rep(i, 0, B) next[L][i] = -1;
        end[L++] = 0;
        return L - 1;
    }

    void init() { // Please do initalize it!!
        L = 0;
        root = newnode();
    }

   void insert(const string &buf) {
        int len = sz(buf);
        int now = root;
        rep(i, 0, len) {
            int &nxt = next[now][buf[i] - 'a'];
            if(nxt == -1) nxt = newnode();
            now = nxt;
        }
        end[now] = 1;
    }

    void build() {
        queue<int> Q;
        fail[root] = root;
        rep(i, 0, B) {
            if (next[root][i] == -1) next[root][i] = root;
            else {
                fail[next[root][i]] = root;
                Q.push(next[root][i]);
            }
        }
        while (!Q.empty()) {
            int now = Q.front();
            Q.pop();
            rep(i, 0, B) {
                if (next[now][i] == -1) next[now][i] = next
                   ↪[fail[now]][i];
                else{
                    fail[next[now][i]] = next[fail[now]][i
                       ↪];
                    Q.push(next[now][i]);
                }
            }
        }
    }
}Aho; // hash-cpp-all = 8e812a31b5ea14dc64eb7c7de956873c
```

KMP.cpp

Description: f[i]: the longest smaller prefix that is a suffix of t[1...i] (in length), one-index.
Time: $\mathcal{O}(|t| + |s|)$

29 lines

```
struct KMP {
    vi f;
    string t;

    KMP(string t): t(t) {
        calc_next(t);
    }

    void calc_next(string t) {
        f.resize(sz(t) + 1);
        f[0] = 0; f[1] = 0;
        for(int i = 1; i < sz(t); i++){
            int j = f[i];
            while(j && t[i] != t[j]) j = f[j];
            f[i+1] = t[i] == t[j] ? j + 1 : 0;
        }
    }

    int match(string s) {
        int n = sz(s), m = sz(t);
        int res = 0, j = 0;
        for(int i = 0; i < n; i++){
            while(j && t[j] != s[i]) j = f[j];
            if(t[j] == s[i]) j++;
            if(j == m) res ++, j = f[j];
        }
        return res;
    }
}; // hash-cpp-all = 264ddfa1a16d4f8efe98eb641f06f8a5
```

Manacher.cpp

44 lines

```
struct Manacher {
    string s, sn;
    int p[2*N];
```

```
    int Init() {
        int len = s.length();
        sn = "$#";

        for (int i = 0; i < len; i++) {
            sn.push_back(s[i]);
            sn.push_back('#');
        }

        sn.push_back('\0');

        return sn.length();
    }

    int run() {
        int len = Init();
        int max_len = -1;

        int id = 0;
        int mx = 0;

        for (int i = 1; i < len; i++)
        {
            if (i < mx)
                p[i] = min(p[2 * id - i], mx - i);
            else
                p[i] = 1;

            while (sn[i - p[i]] == sn[i + p[i]])
                p[i]++;

            if (mx < i + p[i]) id = i, mx = i + p[i];

            max_len = max(max_len, p[i] - 1);
        }

        return max_len;
    }
} mnc;
// hash-cpp-all = 7fe8cfcda626ca76d306f57435f7c0ef
```

SAM.cpp

Description: Build suffix automaton in $O(nB)$. For multi-string SAM, build a trie and do BFS, and set last as the last of parent in the trie.n the trie.

49 lines

```
struct state {
    int len, link;
    int next[B];
};

struct SAM {
    state st[MAXLEN * 2];
    int sz, last;
    int cnt[MAXLEN * 2];

    void sam_init() { // hash-cpp-1
        st[0].len = 0;
        st[0].link = -1;
        memset(st[0].next, -1, sizeof(st[0].next));
        sz = 1;
        last = 0;
    } // hash-cpp-1 = 88458c9c46d0594815f25aa78c9c9a6f

    void sam_extend(int c) { // hash-cpp-2
        int cur = sz++;
        st[cur].len = st[last].len + 1;
        memset(st[cur].next, -1, sizeof(st[cur].next));
        int p = last;
        while(p != -1 && st[p].next[c] == -1) {
            st[p].next[c] = cur;
            p = st[p].link;
        }
        if(p == -1) {
            st[cur].link = 0;
        } else {
            int q = st[p].next[c];
            if(st[p].len + 1 == st[q].len) {
                st[cur].link = q;
            } else {
                int clone = sz++;
                st[clone].len = st[p].len + 1;
                memcpy(st[clone].next, st[q].next, sizeof(
                    ↪st[q].next));
                st[clone].link = st[q].link;
                while(p != -1 && st[p].next[c] == q) {
                    st[p].next[c] = clone;
                    p = st[p].link;
                }
                st[q].link = st[cur].link = clone;
            }
        }
        last = cur;
    } // hash-cpp-2 = 8c1e20b0496de4b588c86bb364a6e5cf

} sam;
```

PAM.cpp

SuffixArray.cpp

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

24 lines

```
struct SuffixArray {
  vi sa, lcp, rank;
  SuffixArray(string& s, int lim=256) { // or basic_string<
      ↪int>
    int n = sz(s) + 1, k = 0, a, b;
    vi x(all(s)+1), y(n), ws(max(n, lim));
    x.back() = 0; // if basic_string<int>
    sa = lcp = rank = y, iota(all(sa), 0);
    for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim =
        ↪p) {
      p = j, iota(all(y), n - j);
      rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
      fill(all(ws), 0);
      rep(i,0,n) ws[x[i]]++;
      rep(i,1,lim) ws[i] += ws[i - 1];
      for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
      swap(x, y), p = 1, x[sa[0]] = 0;
      rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
        (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p
          ↪++;
    }
    rep(i,1,n) rank[sa[i]] = i;
    for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
      for (k && k--, j = sa[rank[i] - 1];
          s[i + k] == s[j + k]; k++);
  }
}; // hash-cpp-all = af6598a85079a59ffbf11b0175d8ff40
```

ZAlgorithm.cpp

Description: str = "aabaacd", z = 7, 1, 0, 2, 1, 0, 0

11 lines

```
vi get_z(string str) {
    int n = sz(str);
    vi z(n);
    z[0] = n;
    for(int i = 1, l = 0, r = 0; i < n; ++i){
        if(i <= r) z[i] = min(r - i + 1, z[i - l]);
        while(i + z[i] < n && str[z[i]] == str[i + z[i]])
            ↪++z[i];
        if(i + z[i] - 1 > r) l = i,  r = i + z[i] - 1;
    }
    return z;
} // hash-cpp-all = 3ccf4002389a292f5d076170fea26d62
```

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.

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

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

8 lines

```
int min_rotation(string s) {
  int a=0, N=sz(s); s += s;
  rep(b,0,N) rep(i,0,N) {
    if (a+i == b || s[a+i] < s[b+i]) {b += max(0, i-1);
      ↪break;}
    if (s[a+i] > s[b+i]) { a = b; break; }
  }
  return a;
} // hash-cpp-all = 358164768a20176868eba20757681e19
```

Various (9)

9.1 Misc. algorithms

Karatsuba.h

Description: Faster-than-naive convolution of two sequences: $c[x] = \sum a[i]b[x-i]$. Uses the identity $(aX+b)(cX+d) = acX^2 + bd + ((a+c)(b+d) - ac - bd)X$. Doesn't handle sequences of very different length well. See also FFT, under the Numerical chapter.

Time: $\mathcal{O}\left(N^{1.6}\right)$

1 lines

```
// hash-cpp-all = d41d8cd98f00b204e9800998ecf8427e
```

9.2 Dynamic programming

digitDP.cpp

25 lines

```
const int B = 20;

int dp[10000][B];
int bit[B], b;
int pw2[B];

int get(int rem, int d, bool flag){
    if(rem < 0) return 0;
    if(d == -1) return rem >= 0;
    if(!flag && ~dp[rem][d]) return dp[rem][d];
    int lim = flag ? bit[d] : 9;
    int res = 0;
    for(int i = 0; i <= lim; i++){
        res += get(rem - i * pw2[d], d - 1, flag && lim ==
            ↪i);
```

```
    }
    return flag ? res : dp[rem][d] = res;
}

int solve(int x){
    b = 0;
    int t = x;
    while(t > 0){bit[b++] = t % 10; t /= 10;}
    return get(A, b-1, true);
}
// hash-cpp-all = 562b39c346c3e127832d23b153971125
```

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

1 lines

```
// hash-cpp-all = d41d8cd98f00b204e9800998ecf8427e
```

9.3 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` violations generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).
- `feenableexcept(29);` kills the program on NaNs (1), 0-divs (4), infinities (8) and denormals (16).

9.4 Optimization tricks

9.4.1 Bit hacks

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

9.4.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize for loops and optimizes floating points better (assumes associativity and turns off denormals).
- `#pragma GCC target ("avx,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).

BumpAllocator.h

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

9 lines

```
// Either globally or in a single class:
static char buf[450 << 20];
void* operator new(size_t s) {
  static size_t i = sizeof buf;
  assert(s < i);
  return (void*)&buf[i -= s];
}
void operator delete(void*) {}
// hash-cpp-all = 745db225903de8f3cdfa051660956100
```

SmallPtr.h

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

"BumpAllocator.h"

10 lines

```
template<class T> struct ptr {
  unsigned ind;
  ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0) {
    assert(ind < sizeof buf);
  }
  T& operator*() const { return *(T*)(buf + ind); }
  T* operator->() const { return &**this; }
  T& operator[](int a) const { return (&**this)[a]; }
  explicit operator bool() const { return ind; }
}; // hash-cpp-all = 2dd6c9773f202bd47422e255099f4829
```

BumpAllocatorSTL.h

Description: BumpAllocator for STL containers.

Usage: vector<vector<int, small<int>>> ed(N);

14 lines

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

template<class T> struct small {
  typedef T value_type;
  small() {}
  template<class U> small(const U&) {}
  T* allocate(size_t n) {
    buf_ind -= n * sizeof(T);
    buf_ind &= 0 - alignof(T);
    return (T*)(buf + buf_ind);
  }
  void deallocate(T*, size_t) {}
}; // hash-cpp-all = bb66d4225a1941b85228ee92b9779d4b
```

Unrolling.h

6 lines

```
#define F {...; ++i;}
int i = from;
while (i&3 && i < to) F // for alignment, if needed
while (i + 4 <= to) { F F F F }
while (i < to) F
// hash-cpp-all = 520e76d6182da81d99aa0e67b36a0b3d
```

SIMD.h

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

43 lines

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

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

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

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

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

ll example_filteredDotProduct(int n, short* a, short* b) {
  int i = 0; ll r = 0;
  mi zero = _mm256_setzero_si256(), acc = zero;
  while (i + 16 <= n) {
    mi va = L(a[i]), vb = L(b[i]); i += 16;
    va = _mm256_and_si256(_mm256_cmpgt_epi16(vb, va), va);
    mi vp = _mm256_madd_epi16(va, vb);
    acc = _mm256_add_epi64(_mm256_unpacklo_epi32(vp, zero),
      _mm256_add_epi64(acc, _mm256_unpackhi_epi32(vp, zero)
        ↪));
```

```
  }
  union {ll v[4]; mi m;} u; u.m = acc; rep(i,0,4) r += u.v[
    ↪i];
  for (;i<n;++i) if (a[i] < b[i]) r += a[i]*b[i]; // <-
    ↪equiv
  return r;
} // hash-cpp-all = 551b820442570276f239d9d7e0800c65
```

9.5 Other languages

Main.java

Description: Basic template/info for Java

14 lines

```
import java.util.*;
import java.math.*;
import java.io.*;
public class Main {
  public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new
      ↪InputStreamReader(System.in));
    PrintStream out = System.out;
    StringTokenizer st = new StringTokenizer(br.readLine())
      ↪;
    assert st.hasMoreTokens(); // enable with java -ea main
    out.println("v=" + Integer.parseInt(st.nextToken()));
    ArrayList<Integer> a = new ArrayList<>();
    a.add(1234); a.get(0); a.remove(a.size()-1); a.clear();
  }
}
```