Universidade de São Paulo

Lucas Harada, Jiang Zhi, Victor Lamarca

Contest (1)

template.cpp

14 lines

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

#define fr(i, n) for(int i = 0; i < n; i++)
#define all(v) (v).begin(), (v).end()
#define sz(v) (int)(v.size())
#define prin(a) cout << #a << " = " << a << endl
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int main() {
  ios::sync_with_stdio(0), cin.tie(0);
}
```

.bashrc

6 lines

```
comp() {
    g++ -std=c++17 -Ofast -Wall -Wshadow -fsanitize=address -
        D_GLIBCXX_DEBUG -W -Wextra -o $1 $1.cpp
}
run() {
    comp $1 && ./$1
}
```

.vimrc

3 lines

```
set nu sc ci si ai sw=4 ts=4 bs=2
set mouse=a
syntax on
```

hash.sh

3 lines

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]'| md5sum |cut -c-6
```

troubleshoot.txt

34 lines

```
VLAMARCA Misc:
cr
g++ -std=c++20 -Wshadow -fsanitize=address -D_GLIBCXX_DEBUG -W
     -Wall -Wextra $1 && ./a.out
crf
g++ -std=c++20 -O2 -w $1 && ./a.out
Large primes
  cabe int
  1.5e9+1
  1163926061
  long long
  1e15+37
Script run_mult
#Script chamado ri:
------------------------------
echo $RANDOM > auxin
./cr A.cpp < auxin
while true;
do
  echo $RANDOM > auxin
  ./a.out < auxin
done
-----------------------------
#O script acima deve ser usado para rodar um programas varias
     vezes para achar algum erro. Usar da seguinte forma:
    -PRINTAR SEED NO PROGRAMA EM Q SE BUSCA ERRO (SEED E
         RECEBIDA PELA VARIAVEL BASH $RANDOM)
    -CRIAR GERADOR DE INPUT NO PROPRIO PROGRAMA A PARTIR DA
         SEED
    -FAZER CHECKER, SE DER RUIM, GERAR LOOP INFINITO
    -USAR O SIMPLES SCRIPT ACIMA DE RODAR PROGRAMA VARIAS VEZES
  SE DER RUIM, ULTIMA SEED PRINTADA CAUSA O ERRO
Random:
//mt19937 rng(chrono::steady_clock::now().time_since_epoch().
     count());
mt19937 rng(time(0));
int seed = uniform_int_distribution(0, INT_MAX)(rng);
int x = rng()%n;
shuffle(all(v),rng);
```

Data structures (2)

OrderStatisticTree.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change null_type.
Time: $\mathcal{O}(\log N)$

782797, 14 lines

```
#include <bits/extc++.h>
using namespace __gnu_pbds;
template<class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
void example() {
  Tree<int> t, t2; t.insert(8);
  auto it = t.insert(10).first;
  assert(it == t.lower_bound(9));
  assert(t.order_of_key(10) == 1);
  assert(t.order_of_key(11) == 2);
  assert(*t.find_by_order(0) == 8);
  t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

SegmentTree2D.h

Description: SegTree2D
Time: $\mathcal{O}(X)$

899d6b, 39 lines

```
struct SEG2D{
 int n, m;
 vector<vector<int>> st;
 const int NEUT = MOD; // 0 para soma, INF para min
 int op(int a, int b){return min(a, b);}
 SEG2D(vector<vector<int>> a = {}){
  n = a.size(), m = (n ? a[0].size() : 0);
  st.resize(2 * n);
  fr(i, 2 * n) st[i].resize(2 * m);
  fr(i, n) fr(j, m) st[i + n][j + m] = MOD;
  fr(i, n) for(int j = m - 1 ; j > 0; j--)
   st[i + n][j] = op(st[i + n][j << 1], st[i + n][j << 1 | 1]);
  for(int i = n - 1; i > 0; i--) fr(j, 2*m)
   st[i][j]=op(st[i<<1][j],st[i<<1|1][j]);
 }
 void update(int x, int y, int v){ // intervalo aberto
  st[x + n][y + m]=v;
  for(int j = y + m;j > 1;j >>= 1)
   st[x + n][j >> 1] = op(st[x + n][j], st[x + n][j ^ 1]);
  for(int i = x + n; i > 1;i >>= 1)
   for(int j = y + m; j ; j >>= 1)
    st[i >> 1][j] = op(st[i][j], st[i^1][j]);
 }
 int query(int x0, int x1, int y0, int y1){// intervalo aberto
  int r = NEUT;
  for(int i0 = x0 + n,i1 = x1 + n; i0 < i1; i0 >>= 1, i1 >>= 1)
      {
   int t[4],q = 0;
   if(i0&1) t[q++] = i0++;
   if(i1&1) t[q++] = --i1;
   fr(k, q){
    for(int j0 = y0 + m,j1 = y1 + m;j0 < j1; j0 >>= 1,j1 >>= 1)
        {
     if(j0&1)r = op(r, st[t[k]][j0++]);
     if(j1&1)r = op(r, st[t[k]][--j1]);
    }
   }
  }
  return r;
 }
};
```

SegmentTreeBeats.h

Description: An special segment tree with the following operation
Time: $\mathcal{O}(\log N)$

41672b, 130 lines

```
/*
 * query(a, b) - {{min(v[a..b]), max(v[a..b])}, sum(v[a..b])}
 * updatemin(a, b, x) turn v[i] <- min(v[i], x), for each i in
      [a, b]
 * updatemax do the same operation with max, and updatesum add
      x
 * for each i in [a, b]
 * Complexity
 * build - O(n)
 * query - O(log(n))
 * update - O(log^2 (n)) amortizado (se nao usar updatesum,
      fica log(n) amortizado)
*/
#define f first
#define s second
namespace beats {
  struct node {
    int tam;
    ll sum, lazy; // lazy pra soma
    ll mi1, mi2, mi; // mi = #mi1
    ll ma1, ma2, ma; // ma = #ma1
    node(ll x = 0) {
      sum = mi1 = ma1 = x;
      mi2 = LINF, ma2 = -LINF;
      mi = ma = tam = 1;
      lazy = 0;
    }
```

```
  node(const node& l, const node& r) {
    sum = l.sum + r.sum, tam = l.tam + r.tam;
    lazy = 0;
    if (l.mi1 > r.mi1) {
      mi1 = r.mi1, mi = r.mi;
      mi2 = min(l.mi1, r.mi2);
    } else if (l.mi1 < r.mi1) {
      mi1 = l.mi1, mi = l.mi;
      mi2 = min(r.mi1, l.mi2);
    } else {
      mi1 = l.mi1, mi = l.mi+r.mi;
      mi2 = min(l.mi2, r.mi2);
    }
    if (l.ma1 < r.ma1) {
      ma1 = r.ma1, ma = r.ma;
      ma2 = max(l.ma1, r.ma2);
    } else if (l.ma1 > r.ma1) {
      ma1 = l.ma1, ma = l.ma;
      ma2 = max(r.ma1, l.ma2);
    } else {
      ma1 = l.ma1, ma = l.ma+r.ma;
      ma2 = max(l.ma2, r.ma2);
    }
  }
  void setmin(ll x) {
    if (x >= ma1) return;
    sum += (x - ma1)*ma;
    if (mi1 == ma1) mi1 = x;
    if (mi2 == ma1) mi2 = x;
    ma1 = x;
  }
  void setmax(ll x) {
    if (x <= mi1) return;
    sum += (x - mi1)*mi;
    if (ma1 == mi1) ma1 = x;
    if (ma2 == mi1) ma2 = x;
    mi1 = x;
  }
  void setsum(ll x) {
    mi1 += x, mi2 += x, ma1 += x, ma2 += x;
    sum += x*tam;
    lazy += x;
  }
};
node seg[4*MAX];
int n, *v;
node build(int p=1, int l=0, int r=n-1) {
  if (l == r) return seg[p] = {v[l]};
  int m = (l+r)/2;
  return seg[p] = {build(2*p, l, m), build(2*p+1, m+1, r)};
}
void build(int n2, int* v2) {
  n = n2, v = v2;
  build();
}
void prop(int p, int l, int r) {
  if (l == r) return;
  for (int k = 0; k < 2; k++) {
    if (seg[p].lazy) seg[2*p+k].setsum(seg[p].lazy);
    seg[2*p+k].setmin(seg[p].ma1);
    seg[2*p+k].setmax(seg[p].mi1);
  }
  seg[p].lazy = 0;
}
pair<pair<ll, ll>, ll> query(int a, int b, int p=1, int l=0,
    int r=n-1) {
  if (b < l or r < a) return {{LINF, -LINF}, 0};
  if (a <= l and r <= b) return {{seg[p].mi1, seg[p].ma1},
      seg[p].sum};
  prop(p, l, r);
  int m = (l+r)/2;
  auto L = query(a, b, 2*p, l, m), R = query(a, b, 2*p+1, m
      +1, r);
  return {{min(L.f.f, R.f.f), max(L.f.s, R.f.s)}, L.s+R.s};
}
node updatemin(int a, int b, ll x, int p=1, int l=0, int r=n
    -1) {
  if (b < l or r < a or seg[p].ma1 <= x) return seg[p];
  if (a <= l and r <= b and seg[p].ma2 < x) {
    seg[p].setmin(x);
    return seg[p];
  }
  prop(p, l, r);
  int m = (l+r)/2;
  return seg[p] = {updatemin(a, b, x, 2*p, l, m),
          updatemin(a, b, x, 2*p+1, m+1, r)};
}
node updatemax(int a, int b, ll x, int p=1, int l=0, int r=n
    -1) {
  if (b < l or r < a or seg[p].mi1 >= x) return seg[p];
  if (a <= l and r <= b and seg[p].mi2 > x) {
    seg[p].setmax(x);
    return seg[p];
  }
  prop(p, l, r);
  int m = (l+r)/2;
  return seg[p] = {updatemax(a, b, x, 2*p, l, m),
          updatemax(a, b, x, 2*p+1, m+1, r)};
}
node updatesum(int a, int b, ll x, int p=1, int l=0, int r=n
    -1) {
  if (b < l or r < a) return seg[p];
  if (a <= l and r <= b) {
    seg[p].setsum(x);
    return seg[p];
  }
  prop(p, l, r);
  int m = (l+r)/2;
  return seg[p] = {updatesum(a, b, x, 2*p, l, m),
          updatesum(a, b, x, 2*p+1, m+1, r)};
}
};
```

ImplicitTreap.h

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

79981c, 142 lines

```
struct Treap { // implicit key (key = index)
    int prior, size;
    int val; //value stored in the array
    int inc, mn;
    bool rev;
    Treap *left, *right;
    Treap() {}
    Treap(int v) {
        prior = rand();
        size = 1;
        val = v;
        inc = 0;
        mn = v;
        rev = false;
        left = right = NULL;
    }
};
inline int size(Treap* t) {
    return (t ? t->size : 0);
}
// flag t->inc is set,
// => the subtree of t (t not included) is not up-to-date
// flag t->rev is set,
// => every node in substree of t should
//    swap its 2 children
// Every Treap corresponds to a range in array
inline void push(Treap* t) {
    if (t->rev) {
        swap(t->left,t->right);
        if (t->left) {
            t->left->rev ^= 1;
        }
        if(t->right) {
            t->right->rev ^= 1;
        }
        t->rev = false;
    }
    if (t->inc) {
        if (t->left) {
            t->left->val += t->inc;
            t->left->inc += t->inc;
            t->left->mn += t->inc;
        }
        if (t->right) {
            t->right->val += t->inc;
            t->right->inc += t->inc;
            t->right->mn += t->inc;
        }
        t->inc = 0;
    }
}
inline void pull(Treap* t) {
    t->size = 1 + size(t->left) + size(t->right);

    t->mn = t->val;
    if (t->left) t->mn = min(t->mn, t->left->mn);
    if (t->right) t->mn = min(t->mn, t->right->mn);
}
int NN = 0;
Treap pool[200000];
inline Treap* new_treap(int val) {
    pool[NN] = Treap(val);
    return &pool[NN++];
}
Treap* merge(Treap* a, Treap* b) {
    if (!a || !b) return (a ? a : b);
    if (a->prior > b->prior) {
        push(a);
        a->right = merge(a->right, b);
        pull(a);
        return a;
    }
    else {
        push(b);
        b->left = merge(a, b->left);
        pull(b);
        return b;
    }
}
// size(a) will be k
// t is unable to use afterwards
void split(Treap* t, Treap*& a, Treap*& b, int k) {
    if (!t) { a = b = NULL; return; }
    push(t);
    if (size(t->left) < k) {
        a = t;
        split(t->right, a->right, b, k - size(t->left) - 1);
        pull(a);
    }
```

```
    else {
        b = t;
        split(t->left, a, b->left, k);
        pull(b);
    }
}
void add(Treap*& t, int x, int y, int inc) {
    Treap *a, *b, *c, *d;
    split(t, a, b, y); // t -> a, b
    split(a, c, d, x - 1); // a -> c, d
    d->inc += inc;
    d->val += inc;
    d->mn += inc;
    t = merge(merge(c, d), b);
}
void reverse(Treap*& t, int x, int y) {
    Treap *a, *b, *c, *d;
    split(t, a, b, y); // t -> a, b
    split(a, c, d, x - 1); // a -> c, d
    d->rev ^= 1;
    t = merge(merge(c, d), b);
}
void revolve(Treap*& t, int x, int y, int k) { // go left by k
    int len = y - x + 1;
    Treap *a, *b, *c, *d;
    split(t, a, b, y); // t -> a, b
    split(a, c, d, x - 1); // a -> c, d
    k = k % len;
    Treap *e, *f;
    split(d, e, f, len - k); // d -> e, f
    t = merge(merge(c, merge(f, e)), b);
}
void insert(Treap*& t, int x, int val) {
    Treap *a, *b;
    split(t, a, b, x);
    t = merge(merge(a, new_treap(val)), b);
}
void remove(Treap*& t, int x) {
    Treap *a, *b, *c, *d;
    split(t, a, b, x - 1); // t -> a, b
    split(b, c, d, 1); // b -> c, d
    t = merge(a, d);
}
int get_min(Treap*& t, int x, int y) {
    Treap *a, *b, *c, *d;
    split(t, a, b, y); // t -> a, b
    split(a, c, d, x - 1); // a -> c, d
    int ans = d->mn;
    t = merge(merge(c, d), b);
    return ans;
}
// Treap* root = NULL;
// root = merge(root, new_treap(val));
```

MoQueries.h

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

a12ef4, 47 lines

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

SlopeTrickH.h

Description: Slope Trick H
Time: $\mathcal{O}(N \log N)$

c4dafa, 16 lines

```
// Solves: Given a vector h, with a[i]/d[i] for cost increase/
    decrease
// entry i, what is the minimun cost to make it non decreasing?
ll slope = 0, linear = 0;
priority_queue< pair<ll, ll> > pq;
for (int i = 0; i < n; i++){
      slope += a[i], linear -= a[i] * h[i];
      pq.push({h[i], a[i] + d[i]});
      while (slope > 0){
          ll pnt, frq;
          tie(pnt, frq) = pq.top(), pq.pop();
          ll aux = min(slope, frq), slope -= aux, frq -= aux;
          linear += pnt * aux;
          if (frq > 0) pq.push({pnt, frq});
      }
}
cout << linear << endl;
```

SparseTableVL.h

Description: SparseTable - RMQ
Time: $\mathcal{O}(X)$

8b60f6, 35 lines

```
int log_floor(int n){
    return 31-__builtin_clz(n);
}
ll oper(ll a, ll b){
    return max(a,b);
}
/*
    Sparse table de maximo
        Ou definida de acordo com funcao oper acima
*/
struct sparse_table{
    int exp2;
    int n;
    vector<vector<ll>> mat;
sparse_table(){}
sparse_table(vector<ll> v){
    n = sz(v);
    exp2 = log_floor(n)+1;
    mat.resize(exp2);
    mat[0].resize(n);
    fr(i,n) mat[0][i] = v[i];
    for(int k = 1; k<exp2; k++){
        mat[k].resize(n);
        for(int i = 0; i+(1<<k)<=n; i++){
            mat[k][i] = oper(mat[k-1][i],mat[k-1][i+(1<<(k-1))
                ]);
        }
    }
}
//query fechada [l,r]
ll qry(int l, int r){
    assert(l<=r and l>=0 and r<n);
    int k = log_floor(r-l+1);
    return oper(mat[k][l],mat[k][r-(1<<k)+1]);
}
}; //end sparse_table
```

SegTreeIterativaVL.h

Description: SegTreeIterativa
Time: $\mathcal{O}(X)$

e1e675, 51 lines

```
struct node{
    ll val;
};
node oper(node a, node b){
    return node{a.val+b.val};
}
struct Seg{
node nulo(){
    return node{0};
}
//----------MUDAR ACIMA DISSO GERALMNT----------
int n;
vector<node> s;
Seg(){}
void build(){
    for(int i = n-1;i>0;i--){
        s[i] = oper(s[i<<1],s[i<<1|1]);
    }
}
Seg(int _n){
    n = _n;
    s = vector<node>(2*n);
    for(int i = n; i<2*n; i++) s[i] = nulo();
    build();
}
Seg(vector<ll> v){
    n = sz(v);
    s = vector<node>(2*n);
    for(int i = n; i<2*n; i++) s[i] = node{v[i-n]}; //mudar
        inicializacao de node a partir de v[i]
    build();
```

```
}
//pos 0-indexed (incrementa/faz operacao, nao atualiza/seta)
void upd(int pos, node val){
    pos+=n;
    s[pos] = oper(s[pos],val);
    for(;pos>1;pos>>=1)
        s[pos>>1] = oper(s[pos],s[pos^1]);
    //para atualizar/setar:
    //for(s[pos+=n]=val;pos>1;pos>>=1)
    //    s[pos>>1] = oper(s[pos],s[pos^1]);
}
//array eh abstraido para 0-indexed (nas folhas da seg) e [l,r)
node qry(int l, int r){
    node ans = nulo();
    for(l+=n,r+=n;l<r;l>>=1,r>>=1){
        if(l&1) ans = oper(ans,s[l++]);
        if(r&1) ans = oper(ans,s[--r]);
    }
    return ans;
}
}; // end seg
```

SegTreeGetFirstVL.h

Description: SegTree GetFirst
Time: $\mathcal{O}(X)$

e7be7b, 83 lines

```
struct node{
    ll soma, mx, mn;
    int l, r;
    ll lazy;
};
node nulo(){
    return node{0,-LLONG_MAX,LLONG_MAX,0,0,0};
}
node oper(node n1, node n2){
    return node{n1.soma+n2.soma,max(n1.mx,n2.mx),min(n1.mn,n2.
        mn),n1.l,n2.r,0};
}
struct Seg{
int n;
vector<node> s;
vector<ll> v;
// Seta o range. Para Incrementar mudar para +=
void updlazy(int no, ll x){
    if(x==0) return;
    s[no].soma = x*(s[no].r-s[no].l); // +=
    s[no].mx = x; // +=
    s[no].mn = x; // +=
    s[no].lazy = x;
}
//------------------MUDAR ACIMA DISSO (GERALMNT)
void build(int no, int l, int r){
    if(r-l==1){
        s[no] = node{v[l],v[l],v[l],l,r,0}; //mudar
            inicializacao a partir de v tmbm
        return;
    }
    int mid = (r+l)/2;
    build(2*no,l,mid);
    build(2*no+1,mid,r);
    s[no] = oper(s[2*no],s[2*no+1]);
}
Seg(vector<ll> _v){
    v = _v;
    n = sz(v);
    s = vector<node>(4*n);
    build(1,0,n);
}
void pass(int no){
    updlazy(2*no,s[no].lazy);
    updlazy(2*no+1,s[no].lazy);
    s[no].lazy = 0;
}
void upd(int lup, int rup, ll x, int no = 1){
    if(rup<=s[no].l or s[no].r<=lup) return;
    if(lup<=s[no].l and s[no].r<=rup){
        updlazy(no,x);
        return;
    }
    pass(no);
    upd(lup,rup,x,2*no);
    upd(lup,rup,x,2*no+1);
    s[no] = oper(s[2*no],s[2*no+1]);
}
node qry(int lq, int rq, int no = 1){
    if(rq<=s[no].l or s[no].r<=lq) return nulo();
    if(lq<=s[no].l and s[no].r<=rq){
        return s[no];
    }
    pass(no);
    return oper(qry(lq,rq,2*no), qry(lq,rq,2*no+1));
}
int get_first(int lq, int rq, const function<bool(const node&)>
    &f, int no = 1){
    if(rq<=s[no].l or s[no].r<=lq) return -1;
    if(!f(s[no])) return -1;
    if(s[no].l+1==s[no].r) return s[no].l;
    pass(no);
    int ans = get_first(lq,rq,f,2*no);
    if(ans!=-1) return ans;
    return get_first(lq,rq,f,2*no+1);
}
int get_last(int lq, int rq, const function<bool(const node&)>
    &f, int no = 1){
    if(rq<=s[no].l or s[no].r<=lq) return -1;
    if(!f(s[no])) return -1;
    if(s[no].l+1==s[no].r) return s[no].l;
    pass(no);
    int ans = get_last(lq,rq,f,2*no+1);
    if(ans!=-1) return ans;
    return get_last(lq,rq,f,2*no);
}
}; //end seg
```

Numerical (3)

3.1 Polynomials and recurrences

Polynomial.h

c9b7b0, 17 lines

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

PolyVL.h

Description: PolyVL
Time: $\mathcal{O}(X)$

b72e79, 459 lines

```
template<unsigned M_> struct modnum {
    static constexpr unsigned M = M_;
    using ll = long long; using ull = unsigned long long;
        unsigned x;
    constexpr modnum() : x(0U) {}
    constexpr modnum(unsigned x_) : x(x_ % M) {}
    constexpr modnum(int x_) : x(((x_ %= static_cast<int>(M)) <
        0) ? (x_ + static_cast<int>(M)) : x_) {}
    constexpr modnum(ull x_) : x(x_ % M) {}
    constexpr modnum(ll x_) : x(((x_ %= static_cast<ll>(M)) <
        0) ? (x_ + static_cast<ll>(M)) : x_) {}
    explicit operator int() const { return x; }
    modnum& operator+=(const modnum& a) { x = ((x += a.x) >= M)
        ? (x - M) : x; return *this; }
    modnum& operator-=(const modnum& a) { x = ((x -= a.x) >= M)
        ? (x + M) : x; return *this; }
    modnum& operator*=(const modnum& a) { x = unsigned((
        static_cast<ull>(x) * a.x) % M); return *this; }
    modnum& operator/=(const modnum& a) { return (*this *= a.
        inv()); }
    modnum operator+(const modnum& a) const { return (modnum(*
        this) += a); }
    modnum operator-(const modnum& a) const { return (modnum(*
        this) -= a); }
    modnum operator*(const modnum& a) const { return (modnum(*
        this) *= a); }
    modnum operator/(const modnum& a) const { return (modnum(*
        this) /= a); }
    modnum operator+() const { return *this; }
    modnum operator-() const { modnum a; a.x = x ? (M - x) : 0U
        ; return a; }
    modnum pow(ll e) const {
        if (e < 0) return inv().pow(-e);
        modnum x2 = x, xe = 1U;
        for (; e; e >>= 1) {
            if (e & 1) xe *= x2;
            x2 *= x2;
        }
        return xe;
    }
    modnum inv() const {
        unsigned a = x, b = M; int y = 1, z = 0;
        while (a) {
            const unsigned q = (b/a), c = (b - q*a);
            b = a, a = c; const int w = z - static_cast<int>(q)
                * y;
            z = y, y = w;
        } assert(b == 1U); return modnum(z);
    }
    friend modnum inv(const modnum& a) { return a.inv(); }
    template<typename T> friend modnum operator+(T a, const
        modnum& b) { return (modnum(a) += b); }
    template<typename T> friend modnum operator-(T a, const
        modnum& b) { return (modnum(a) -= b); }
    template<typename T> friend modnum operator*(T a, const
        modnum& b) { return (modnum(a) *= b); }
    template<typename T> friend modnum operator/(T a, const
        modnum& b) { return (modnum(a) /= b); }
    friend bool operator==(const modnum& a, const modnum& b) {
        return a.x == b.x; }
    friend bool operator!=(const modnum& a, const modnum& b) {
        return a.x != b.x; }
    friend ostream &operator<<(ostream& os, const modnum& a) {
        return os << a.x; }
```

```
    friend istream &operator>>(istream& in, modnum& n) { ull v_
        ; in >> v_; n = modnum(v_); return in; }
};

template<unsigned M_, unsigned G_, int K_ > struct FFT {
    static_assert(2U <= M_, "Fft: 2 <= M must hold.");
    static_assert(M_ < 1U << 30, "Fft: M < 2^30 must hold.");
    static_assert(1 <= K_, "Fft: 1 <= K must hold.");
    static_assert(K_ < 30, "Fft: K < 30 must hold.");
    static_assert(!((M_ - 1U) & ((1U << K_) - 1U)), "Fft: 2^K |
         M - 1 must hold.");
    static constexpr unsigned M = M_, M2 = 2U * M_, G = G_;
    static constexpr int K = K_;
    modnum<M> roots[K + 1], inv_roots[K + 1];
    modnum<M> ratios[K], inv_ratios[K];
    constexpr FFT() {
        const modnum<M> g(G);
        for (int k = 0; k <= K; ++k) {
            roots[k] = g.pow((M - 1U) >> k);
            inv_roots[k] = roots[k].inv();
        }
        for (int k = 0; k <= K - 2; ++k) {
            ratios[k] = -g.pow(3U * ((M - 1U) >> (k + 2)));
            inv_ratios[k] = ratios[k].inv();
        } assert(roots[1] == M - 1U);
    }
    void fft(modnum<M>* as, int n) const {
        assert(!(n & (n - 1))); assert(1 <= n); assert(n <= 1
             << K);
        int m = n;
        if (m >>= 1) {
            for (int i = 0; i < m; ++i) {
                const unsigned x = as[i + m].x;
                as[i + m].x = as[i].x + M - x;
                as[i].x += x;
            }
        }
        if (m >>= 1) {
            modnum<M> prod = 1U;
            for (int h = 0, i0 = 0; i0 < n; i0 += (m << 1)) {
                for (int i = i0; i < i0 + m; ++i) {
                    const unsigned x = (prod * as[i + m]).x;
                    as[i + m].x = as[i].x + M - x;
                    as[i].x += x;
                }
                prod *= ratios[__builtin_ctz(++h)];
            }
        }
        for (; m;) {
            if (m >>= 1) {
                modnum<M> prod = 1U;
                for (int h = 0, i0 = 0; i0 < n; i0 += (m << 1))
                     {
                    for (int i = i0; i < i0 + m; ++i) {
                        const unsigned x = (prod * as[i + m]).x
                            ;
                        as[i + m].x = as[i].x + M - x;
                        as[i].x += x;
                    }
                    prod *= ratios[__builtin_ctz(++h)];
                }
            }
            if (m >>= 1) {
                modnum<M> prod = 1U;
                for (int h = 0, i0 = 0; i0 < n; i0 += (m << 1))
                     {
                    for (int i = i0; i < i0 + m; ++i) {
                        const unsigned x = (prod * as[i + m]).x
                            ;
```

```
                        as[i].x = (as[i].x >= M2) ? (as[i].x -
                             M2) : as[i].x;
                        as[i + m].x = as[i].x + M - x;
                        as[i].x += x;
                    }
                    prod *= ratios[__builtin_ctz(++h)];
                }
            }
        }
        for (int i = 0; i < n; ++i) {
            as[i].x = (as[i].x >= M2) ? (as[i].x - M2) : as[i].
                x;
            as[i].x = (as[i].x >= M) ? (as[i].x - M) : as[i].x;
        }
    }
    void inverse_fft(modnum<M>* as, int n) const {
        assert(!(n & (n - 1))); assert(1 <= n); assert(n <= 1
             << K);
        int m = 1;
        if (m < n >> 1) {
            modnum<M> prod = 1U;
            for (int h = 0, i0 = 0; i0 < n; i0 += (m << 1)) {
                for (int i = i0; i < i0 + m; ++i) {
                    const unsigned long long y = as[i].x + M -
                        as[i + m].x;
                    as[i].x += as[i + m].x;
                    as[i + m].x = (prod.x * y) % M;
                }
                prod *= inv_ratios[__builtin_ctz(++h)];
            }
            m <<= 1;
        }
        for (; m < n >> 1; m <<= 1) {
            modnum<M> prod = 1U;
            for (int h = 0, i0 = 0; i0 < n; i0 += (m << 1)) {
                for (int i = i0; i < i0 + (m >> 1); ++i) {
                    const unsigned long long y = as[i].x + M2 -
                         as[i + m].x;
                    as[i].x += as[i + m].x;
                    as[i].x = (as[i].x >= M2) ? (as[i].x - M2)
                         : as[i].x;
                    as[i + m].x = (prod.x * y) % M;
                }
                for (int i = i0 + (m >> 1); i < i0 + m; ++i) {
                    const unsigned long long y = as[i].x + M -
                        as[i + m].x;
                    as[i].x += as[i + m].x;
                    as[i + m].x = (prod.x * y) % M;
                }
                prod *= inv_ratios[__builtin_ctz(++h)];
            }
        }
        if (m < n) {
            for (int i = 0; i < m; ++i) {
                const unsigned y = as[i].x + M2 - as[i + m].x;
                as[i].x += as[i + m].x;
                as[i + m].x = y;
            }
        }
        const modnum<M> invN = modnum<M>(n).inv();
        for (int i = 0; i < n; ++i) as[i] *= invN;
    }
    void fft(vector<modnum<M>>& as) const { fft(as.data(), int(
         as.size())); }
    void inverse_fft(vector<modnum<M>>& as) const { inverse_fft
         (as.data(), int(as.size())); }
    vector<modnum<M>> convolve(vector<modnum<M>> as, vector<
         modnum<M>> bs) const {
        if (as.empty() || bs.empty()) return {};
```

```
        const int len = int(as.size()) + int(bs.size()) - 1;
        int n = 1; for (; n < len; n <<= 1) {}
        as.resize(n); fft(as);
        bs.resize(n); fft(bs);
        for (int i = 0; i < n; ++i) as[i] *= bs[i];
        inverse_fft(as); as.resize(len); return as;
    }
    vector<modnum<M>> square(vector<modnum<M>> as) const {
        if (as.empty()) return {};
        const int len = int(as.size()) + int(as.size()) - 1;
        int n = 1; for (; n < len; n <<= 1) {}
        as.resize(n); fft(as);
        for (int i = 0; i < n; ++i) as[i] *= as[i];
        inverse_fft(as); as.resize(len); return as;
    }
};

using num = modnum<998244353U>;

FFT<998244353U, 3U, 23> fft_data;

// inv: integral, log, exp, pow
constexpr int LIM_INV = 1 << 20;  // @
num invs[LIM_INV], fac[LIM_INV], invFac[LIM_INV];
struct ModIntPreparator {
  ModIntPreparator() {
    invs[1] = 1;
    for (int i = 2; i < LIM_INV; ++i) invs[i] = -((num::M / i)
         * invs[num::M % i]);
    fac[0] = 1;
    for (int i = 1; i < LIM_INV; ++i) fac[i] = fac[i - 1] * i;
    invFac[0] = 1;
    for (int i = 1; i < LIM_INV; ++i) invFac[i] = invFac[i - 1]
         * invs[i];
  }
} preparator;

template<unsigned M> struct Poly : public vector<modnum<M>> {
  Poly() {}
  explicit Poly(int n) : vector<modnum<M>>(n) {}
  Poly(const vector<modnum<M>> &vec) : vector<modnum<M>>(vec) {
       }
  Poly(std::initializer_list<modnum<M>> il) : vector<modnum<M
       >>(il) {}
  int size() const { return vector<modnum<M>>::size(); }
  num at(long long k) const { return (0 <= k && k < size()) ?
       (*this)[k] : 0U; }
  int ord() const { for (int i = 0; i < size(); ++i) if ((*this
       )[i]) return i; return -1; }
  int deg() const { for (int i = size(); --i >= 0;) if ((*this)
       [i]) return i; return -1; }
  Poly mod(int n) const { return Poly(vector<modnum<M>>(this->
       data(), this->data() + min(n, size()))); }
  friend std::ostream &operator<<(std::ostream &os, const Poly
       &fs) {
    os << "[";
    for (int i = 0; i < fs.size(); ++i) { if (i > 0) os << ", "
         ; os << fs[i]; }
    return os << "]";
  }
  Poly &operator+=(const Poly &fs) {
    if (size() < fs.size()) this->resize(fs.size());
    for (int i = 0; i < fs.size(); ++i) (*this)[i] += fs[i];
    return *this;
  }
  Poly &operator-=(const Poly &fs) {
    if (size() < fs.size()) this->resize(fs.size());
    for (int i = 0; i < fs.size(); ++i) (*this)[i] -= fs[i];
    return *this;
```

```
}
Poly &operator*=(const Poly &fs) {
  if (this->empty() || fs.empty()) return *this = {};
  *this = fft_data.convolve(*this, fs);
  return *this;
}
Poly &operator*=(const num &a) {
  for (int i = 0; i < size(); ++i) (*this)[i] *= a;
  return *this;
}
Poly &operator/=(const num &a) {
  const num b = a.inv();
  for (int i = 0; i < size(); ++i) (*this)[i] *= b;
  return *this;
}
Poly &operator/=(const Poly &fs) {
    auto ps = fs;
    if (size() < ps.size()) return *this = {};
    int s = int(size()) - int(ps.size()) + 1;
    int nn = 1; for (; nn < s; nn <<= 1) {}
    reverse(this->begin(), this->end());
    reverse(ps.begin(), ps.end());
    this->resize(nn); ps.resize(nn);
    ps = ps.inv();
    *this = *this * ps;
    this->resize(s); reverse(this->begin(), this->end());
    return *this;
}
Poly &operator%=(const Poly& fs) {
    if (size() >= fs.size()) {
        Poly Q = (*this / fs) * fs;
        this->resize(fs.size() - 1);
        for (int x = 0; x < int(size()); ++x) (*this)[x] -= Q
             [x];
    }
    while (size() && this->back() == 0) this->pop_back();
    return *this;
}
Poly inv() const {
  if (this->empty()) return {};
  Poly b({(*this)[0].inv()}), fs;
  b.reserve(2 * int(this->size()));
  while (b.size() < this->size()) {
      int len = 2 * int(b.size());
      b.resize(2 * len, 0);
      if (int(fs.size()) < 2 * len) fs.resize(2 * len, 0);
      fill(fs.begin(), fs.begin() + 2 * len, 0);
      copy(this->begin(), this->begin() + min(len, int(this->
           size())), fs.begin());
      fft_data.fft(b);
      fft_data.fft(fs);
      for (int x = 0; x < 2*len; ++x) b[x] = b[x] * (2 - fs[x
           ] * b[x]);
      fft_data.inverse_fft(b);
      b.resize(len);
  }
  b.resize(this->size()); return b;
}
Poly differential() const {
    if (this->empty()) return {};
    Poly f(max(size() - 1, 1));
    for (int x = 1; x < size(); ++x) f[x - 1] = x * (*this)[x
         ];
    return f;
}
Poly integral() const {
    if (this->empty()) return {};
    Poly f(size() + 1);
    for (int x = 0; x < size(); ++x) f[x + 1] = invs[x + 1] *
         (*this)[x];
    return f;
}
Poly log() const {
    if (this->empty()) return {};
    Poly f = (differential() * inv()).integral();
    f.resize(size()); return f;
}
Poly exp() const {
    Poly f = {1};
    if (this->empty()) return f;
    while (f.size() < size()) {
        int len = min(f.size() * 2, size());
        f.resize(len);
        Poly d(len);
        copy(this->begin(), this->begin() + len, d.begin());
        Poly g = d - f.log();
        g[0] += 1;
        f *= g;
        f.resize(len);
    }
    return f;
}
Poly pow(int N) const {
    Poly b(size());
    if (N == 0) { b[0] = 1; return b; }
    int p = 0;
    while (p < size() && (*this)[p] == 0) ++p;
    if (1LL * N * p >= size()) return b;
    num mu = ((*this)[p]).pow(N), di = ((*this)[p]).inv();
    Poly c(size() - N*p);
    for (int x = 0; x < int(c.size()); ++x) {
        c[x] = (*this)[x + p] * di;
    }
    c = c.log();
    for (auto& val : c) val *= N;
    c = c.exp();
    for (int x = 0; x < int(c.size()); ++x) {
        b[x + N*p] = c[x] * mu;
    }
    return b;
}
Poly operator+() const { return *this; }
Poly operator-() const {
  Poly fs(size());
  for (int i = 0; i < size(); ++i) fs[i] = -(*this)[i];
  return fs;
}
Poly operator+(const Poly &fs) const { return (Poly(*this) +=
      fs); }
Poly operator-(const Poly &fs) const { return (Poly(*this) -=
      fs); }
Poly operator*(const Poly &fs) const { return (Poly(*this) *=
      fs); }
Poly operator%(const Poly &fs) const { return (Poly(*this) %=
      fs); }
Poly operator/(const Poly &fs) const { return (Poly(*this) /=
      fs); }
Poly operator*(const num &a) const { return (Poly(*this) *= a
     ); }
Poly operator/(const num &a) const { return (Poly(*this) /= a
     ); }
friend Poly operator*(const num &a, const Poly &fs) { return
     fs * a; }

// multipoint evaluation/interpolation
/* era friend */ static Poly eval(const Poly& fs, const Poly&
      qs) {
    int N = int(qs.size());
    if (N == 0) return {};
    vector<Poly> up(2 * N);
    for (int x = 0; x < N; ++x) {
        up[x + N] = Poly({0-qs[x], 1});
    }
    for (int x = N-1; x >= 1; --x) {
        up[x] = up[2 * x] * up[2 * x + 1];
    }
    vector<Poly> down(2 * N);
    down[1] = fs % up[1];
    for (int x = 2; x < 2*N; ++x) {
        down[x] = down[x / 2] % up[x];
    }
    Poly y(N);
    for (int x = 0; x < N; ++x) {
        y[x] = (down[x + N].empty() ? 0 : down[x + N][0]);
    }
    return y;
}
/* era friend */ static Poly interpolate(const Poly& fs,
     const Poly& qs) {
    int N = int(fs.size());
    vector<Poly> up(2 * N);
    for (int x = 0; x < N; ++x) {
        up[x + N] = Poly({0-fs[x], 1});
    }
    for (int x = N-1; x >= 1; --x) {
        up[x] = up[2 * x] * up[2 * x + 1];
    }
    Poly E = eval(up[1].differential(), fs);
    vector<Poly> down(2 * N);
    for (int x = 0; x < N; ++x) {
        down[x + N] = Poly({qs[x] * E[x].inv()});
    }
    for (int x = N-1; x >= 1; --x) {
        down[x] = down[2*x] * up[2*x+1] + down[2*x+1] * up[2*
             x];
    }
    return down[1];
}
/* era friend */ static Poly convolve_all(const vector<Poly>&
     fs, int l, int r) {
    if (r - l == 1) return fs[l];
    else {
        int md = (l + r) / 2;
        return convolve_all(fs, l, md) * convolve_all(fs, md,
             r);
    }
}
static Poly bernoulli(int N) {
    Poly fs(N);
    fs[1] = 1;
    fs = fs.exp();
    copy(fs.begin()+1, fs.end(), fs.begin());
    fs = fs.inv();
    for (int x = 0; x < N; ++x) fs[x] *= fac[x];
    return fs;
}
// x(x - 1)(x - 2)...(x - N + 1)
static Poly stirling_first(int N) {
    if (N == 0) return {1};
    vector<Poly> P(N);
    for (int x = 0; x < N; ++x) P[x] = {-x, 1};
    return convolve_all(P, 0, N);
}
static Poly stirling_second(int N) {
    if (N == 0) return {1};
    Poly P(N), Q(N);
```

```
        for (int x = 0; x < N; ++x) {
            P[x] = (x & 1 ? -1 : 1) * invFac[x];
            Q[x] = num(x).pow(N) * invFac[x];
        }
        return P * Q;
    }
};

/*
    tested in: https://judge.yosupo.jp/submission/102130

    Poly herda de vector de modnum<P>, acessos sao (*this)[i]
        Primo precisa ser fft friendly pra maioria das
            operacoes (mas posso usar 1 + 7*2^26 e 1 + 5*2^25
            e CRT pra recuperar pra outros mods)
    Ordem do vetor sao os coeficientes do menos pro mais
        significativo
        a[0]*x^0 + a[1]*x^1 + ...
    .deg() do Poly eh o indice do ultimo valor nao nulo (maior
        expoente)
    .ord() eh o indice do primeiro coef nao nulo
    tds functions a seguir retornam um Poly:

    Sao member functions (mas retornam, nao mudam o atual!)
        inv()
        differential()
        integral()
        log()
        exp()
        pow(int N)
        +-*(dividir)

    Poderiam ser statics
        eval(Poly a, Poly b)
        interpolate(Poly a, Poly b)
        convolve_all(vector<Poly>, l, r)
        multiplica tds os polys (nlog^2)
        por default usar de 0 a n
        bernoulli(int N)
        stirling_first(int n)
        stirling_second(int n)
*/

int main() {
    ios::sync_with_stdio(0); cin.tie(0);
    int n; cin >> n;
    vector<num> a(n); for(int i=0;i<n;i++) cin >> a[i];
    Poly b(a);
    Poly c = b.exp();
    for(int i=0;i<n;i++) cout << c[i] << " ";
}
```

PolyRoots.h

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

"Polynomial.h" b00bfe, 23 lines

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

LagrangeInterpolateVL.h

Description: Lagrange
Time: $\mathcal{O}(X)$

a36ebd, 15 lines

```
//pode mudar pra double ou mb
vector<frac> interpolate(vector<frac> x, vector<frac> y) {
    int n = sz(x);
    assert(sz(y)==sz(x));
    vector<frac> res(n), temp(n);
    fr(k,n-1) for(int i = k+1; i<n; i++)
        y[i] = (y[i] - y[k]) / (x[i] - x[k]);
    frac last(0,1); temp[0] = frac(1,1);
    fr(k,n) fr(i,n){
        res[i] = res[i] + y[k] * temp[i];
        swap(last, temp[i]);
        temp[i] = temp[i] - last * x[k];
    }
    return res;
}
```

BerlekampMasseyVL.h

Description: Berlekamp Massey
Time: $\mathcal{O}(X)$

8fdc7d, 92 lines

```
const int mod = 1e9+7;
ll mul(ll x, ll y, ll modc){ return (__int128) x * y % modc; }
ll ipow(ll x, ll y, ll p = mod){
 ll ret = 1, piv = x % p;
 while(y){
  if(y&1) ret = mul(ret, piv, p);
  piv = mul(piv, piv, p);
  y >>= 1;
 }
 return ret;
}
vector<int> berlekamp_massey(vector<int> x){
 vector<int> ls, cur;
 int lf, ld;
 fr(i,sz(x)){
  ll t = 0;
  fr(j,sz(cur)){
   t = (t + 1ll * x[i-j-1] * cur[j]) % mod;
  }
  if((t - x[i]) % mod == 0) continue;
  if(cur.empty()){
   cur.resize(i+1);
   lf = i;
   ld = (t - x[i]) % mod;
   continue;
  }
  ll k = -(x[i] - t) * ipow(ld, mod - 2) % mod;
  vector<int> c(i-lf-1);
  c.push_back(k);
  for(auto &j : ls) c.push_back(-j * k % mod);
  if(sz(c) < sz(cur)) c.resize(sz(cur));
  fr(j,sz(cur)){
   c[j] = (c[j] + cur[j]) % mod;
  }
  if(i-lf+sz(ls)>=sz(cur)){
   tie(ls, lf, ld) = make_tuple(cur, i, (t - x[i]) % mod);
  }
  cur = c;
 }
 for(auto &i : cur) i = (i % mod + mod) % mod;
 return cur;
}
int get_nth(vector<int> rec, vector<int> dp, ll n){
 int m = sz(rec);
 vector<int> s(m), t(m);
 s[0] = 1;
 if(m != 1) t[1] = 1;
 else t[0] = rec[0];
 auto mul = [&rec](vector<int> v, vector<int> w){
  vector<int> ans(2 * sz(v));
  fr(j,sz(v)){
   fr(k,sz(v)){
    ans[j+k] += 1ll * v[j] * w[k] % mod;
    if(ans[j+k] >= mod) ans[j+k] -= mod;
   }
  }
  for(int j=2*sz(v)-1; j>=sz(v); j--){
   for(int k=1; k<=sz(v); k++){
    ans[j-k] += 1ll * ans[j] * rec[k-1] % mod;
    if(ans[j-k] >= mod) ans[j-k] -= mod;
   }
  }
  ans.resize(sz(v));
  return ans;
 };
 while(n){
  if(n & 1) s = mul(s, t);
  t = mul(t, t);
  n >>= 1;
 }
 ll ret = 0;
 fr(i,m) ret += 1ll * s[i] * dp[i] % mod;
 return ret % mod;
}
vector<int> coef; //imprimir vetor coef na main
int guess_nth_term(vector<int> x, ll n){
 if(n < sz(x)) return x[n];
 coef = berlekamp_massey(x);
 if(coef.empty()) return 0;
 return get_nth(coef, x, n);
}
int main() {
    //f(n) = coef[0]*f(n-1) + coef[1]*f(n-2) + ...
    vector<int> va = {1,1,2,3,5,8};
    //fibonacci - n eh 0-indexado
    for(int n = sz(va)-2; n<=sz(va)+5; n++){
        assert(n>=0);
        ll fib = guess_nth_term(va, n);
        cout << "fib[" << n << "] = " << fib << endl;
    }
    prinv(coef);
}
```

RecLinearVL.h

Description: RecLinear
Time: $\mathcal{O}(X)$

cc8447, 30 lines

```
/*
    multipllica matriz - VALORES EM MODULO
    para matriz n x n complexidade n^3
*/
```

```
vector<vector<ll>> mm(vector<vector<ll>> a, vector<vector<ll>>
     b){
    int l = sz(a);
    int c = sz(b[0]);
    assert(sz(a[0])==sz(b));
    vector<vector<ll>> ans(l,vector<ll>(c));
    fr(i,l){
        fr(j,c){
            ll tot = 0;
            fr(k,a[0].size()){
                tot = (tot+a[i][k]*b[k][j])%MOD;
            }
            ans[i][j] = tot;
        }
    }
    return ans;
}
/*
    Eleva matriz a um expoente que deve ser >=1
    se for zero deveria retornar matriz identidade
*/
vector<vector<ll>> em(vector<vector<ll>> a, ll exp){
    if(exp==1) return a;
    vector<vector<ll>> mid = em(a,exp/2);
    if(exp%2) return mm(mm(mid,mid),a);
    return mm(mid,mid);
}
```

3.2 Optimization

Simplex.h

Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \le b$, $x \ge 0$. 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.

aa8530, 62 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)) {
      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;
    }
  void pivot(int r, int s) {
    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]);
  }
  bool simplex(int phase) {
    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);
    }
  }
  T solve(vd &x) {
    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;
  }
};
```

3.3 Matrices

Determinant.h

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

bd5cec, 15 lines

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

IntDeterminant.h

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

3313dc, 18 lines

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

GaussElimVL.h

Description: Gauss Elimination - SolveLinear
Time: $\mathcal{O}(X)$

83432c, 48 lines

```
/*
    retorno:
        0 - sem solucao
        1 - uma solucao
        2 - infinitas solucoes

    resolve sistema - acha X para
        a*X = b
    nos parametros da funcao, b eh a ultima coluna da matriz a
*/
int gauss (vector < vector<mb> > a, vector<mb> & ans) {
    int n = sz(a), m = sz(a[0])-1;
    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (a[i][col].val > a[sel][col].val)
                sel = i;
        if (a[sel][col].val==0)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                mb c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }
    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        mb sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if( sum.val != a[i][m].val)
            return 0;
    }
    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return 2;
    return 1;
}
```

MatrixInverse.h

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

ebfff6, 32 lines

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

Tridiagonal.h

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

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

This is useful for solving problems on the type

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

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

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

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

8f9fa8, 26 lines

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

3.4 Fourier transforms

FastFourierTransform.h

Description: fft(a) computes $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$ for all k. N must be a power of 2. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n, reverse(start+1, end), FFT back. Rounding is safe if $(\sum a_i^2 + \sum b_i^2)\log_2 N < 9 \cdot 10^{14}$ (in practice 10^{16}; higher for random inputs). Otherwise, use NTT/FFTMod.
Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ (~$1s$ for $N = 2^{22}$)

741afd, 136 lines

```
const double PI=acos(-1.0);
namespace fft {
    struct num {
        double x,y;
        num() {x = y = 0;}
        num(double x,double y): 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);}
    int base=1;
    vector<num> roots={{0,0}, {1,0}};
    vector<int> rev={0, 1};
    const double PI=acosl(-1.0);
    // always try to increase the base
    void ensure_base(int nbase) {
        if(nbase <= base) return;
        rev.resize(1 << nbase);
        for (int i = 0; i < (1 << nbase); i++)
            rev[i] = (rev[i>>1] >> 1) + ((i&1) << (nbase-1));
        roots.resize(1<<nbase);
        while(base<nbase) {
            double angle = 2*PI / (1<<(base+1));
            for(int i = 1<<(base-1); i < (1<<base); i++) {
                roots[i<<1] = roots[i];
                double angle_i = angle * (2*i+1-(1<<base));
                roots[(i<<1)+1] = num(cos(angle_i),sin(angle_i)
                    );
            }
            base++;
        }
    }
    void fft(vector<num> &a,int n=-1) {
        if(n==-1) n=a.size();
        assert((n&(n-1)) == 0);
        int zeros = __builtin_ctz(n);
        ensure_base(zeros);
        int shift = base - zeros;
        for (int i = 0; i < n; i++) {
            if(i < (rev[i] >> shift)) {
                swap(a[i],a[rev[i] >> shift]);
            }
        }
        for(int k = 1; k < n; k <<= 1) {
            for(int i = 0; i < n; i += 2*k) {
                for(int j = 0; j < k; j++) {
                    num z = a[i+j+k] * roots[j+k];
                    a[i+j+k] = a[i+j] - z;
                    a[i+j] = a[i+j] + z;
                }
            }
        }
    }
    vector<num> fa, fb;
    // multiply with less fft by using complex numbers.
    vector<int> multiply(vector<int> &a, vector<int> &b) {
        int need = a.size() + b.size() - 1;
        int nbase = 0;
        while((1 << nbase) < need) nbase++;
        ensure_base(nbase);
        int sz = 1 << nbase;
        if(sz > (int)fa.size()) fa.resize(sz);
        for(int i = 0; i < sz; i++) {
            int x = (i < (int)a.size() ? a[i] : 0);
            int y = (i < (int)b.size() ? b[i] : 0);
            fa[i] = num(x, y);
        }
        fft(fa, sz);
        num r(0,-0.25/sz);
        for(int i = 0; i <= (sz>>1); i++) {
            int j = (sz-i) & (sz-1);
            num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
            if(i != j) fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa
                [j])) * r;
            fa[i] = z;
        }
        fft(fa, sz);
        vector<int> res(need);
        for(int i = 0; i < need; i++) res[i] = fa[i].x + 0.5;
        return res;
    }
    vector<int> multiply_mod(vector<int> &a, vector<int> &b,
        int m, int eq=0) {
        int need = a.size() + b.size() - 1;
        int nbase = 0;
        while((1 << nbase) < need) nbase++;
        ensure_base(nbase);
        int sz = 1 << nbase;
        if(sz > (int)fa.size()) fa.resize(sz);
        for(int i = 0; i < (int)a.size(); i++) {
            int x = (a[i] % m + m) % m;
            fa[i] = num(x & ((1 << 15) - 1), x >> 15);
        }
        fill(fa.begin() + a.size(), fa.begin() + sz, num{0,0});
        fft(fa, sz);
        if(sz > (int)fb.size()) fb.resize(sz);
        if(eq) copy(fa.begin(), fa.begin() + sz, fb.begin());
        else {
            for(int i = 0; i < (int)b.size(); i++) {
                int x = (b[i] % m + m) % m;
                fb[i] = num(x & ((1 << 15) - 1), x >> 15);
            }
```

```
            fill(fb.begin() + b.size(), fb.begin() + sz, num{
                0,0});
            fft(fb,sz);
        }
        double ratio = 0.25 / sz;
        num r2(0, -1), r3(ratio, 0), r4(0, -ratio), r5(0,1);
        for(int i = 0; i <= (sz>>1); i++) {
            int j = (sz - i) & (sz - 1);
            num a1 = (fa[i] + conj(fa[j]));
            num a2 = (fa[i] - conj(fa[j])) * r2;
            num b1 = (fb[i] + conj(fb[j])) * r3;
            num b2 = (fb[i] - conj(fb[j])) * r4;
            if(i != j) {
                num c1 = (fa[j] + conj(fa[i]));
                num c2 = (fa[j] - conj(fa[i])) * r2;
                num d1 = (fb[j] + conj(fb[i])) * r3;
                num d2 = (fb[j] - conj(fb[i])) * r4;
                fa[i] = c1 * d1 + c2 * d2 * r5;
                fb[i] = c1 * d2 + c2 * d1;
            }
            fa[j] = a1 * b1 + a2 * b2 * r5;
            fb[j] = a1 * b2 + a2 * b1;
        }
        fft(fa, sz); fft(fb, sz);
        vector<int> res(need);
        for(int i = 0; i < need; i++) {
            ll aa = fa[i].x + 0.5;
            ll bb = fb[i].x + 0.5;
            ll cc = fa[i].y + 0.5;
            res[i] = (aa + ((bb%m) << 15) + ((cc%m) << 30))%m;
        }
        return res;
    }
    vector<int> square_mod(vector<int> &a, int m) {
        return multiply_mod(a, a, m, 1);
    }
};
```

NumberTheoreticTransform.h

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

"../number-theory/ModPow.h" ced03d, 33 lines

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

FastSubsetTransform.h

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

464cf3, 16 lines

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

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" 35bfea, 18 lines

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

ModInverse.h

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

6f684f, 3 lines

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

ModLog.h

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

c040b8, 11 lines

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

ModSum.h

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

5c5bc5, 14 lines

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }
ull divsum(ull to, ull c, ull k, ull m) {
  ull res = k / m * sumsq(to) + c / m * to;
  k %= m; c %= m;
  if (!k) return res;
  ull to2 = (to * k + c) / m;
  return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}
ll modsum(ull to, ll c, ll k, ll m) {
  c = ((c % m) + m) % m;
  k = ((k % m) + m) % m;
  return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution).
Time: $\mathcal{O}(\log^2 p)$ worst case, $\mathcal{O}(\log p)$ for most p

"ModPow.h" 19a793, 24 lines

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

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

4.2 Primality

FastEratosthenes.h

Description: Prime sieve.
Time: LIM=1e9 ≈ 1.5s

2a819b, 13 lines

```
const int MAXN = 10000010;
int lp[MAXN];
vector<int> pr;

void sieve(){
    for (int i = 2; i < MAXN; ++i) {
        if (lp[i] == 0) lp[i] = i, pr.push_back(i);
        for(auto p : pr){
            if(p > lp[i] || i * p >= MAXN) break;
            lp[i * p] = p;
        }
    }
}
```

PollardRhoFfao.h

Description: PollardRhoFfao
Time: $\mathcal{O}(X)$

5aba26, 92 lines

```
typedef unsigned long long ull;
ull gcd(ull u, ull v) {
    if (u == 0 || v == 0)
        return v ^ u;
    int shift = __builtin_ctzll(u | v);
    u >>= __builtin_ctzll(u);
    do {
        v >>= __builtin_ctzll(v);
        if (u > v) {
            ull t = v;
            v = u;
            u = t;
        }
        v -= u;
    } while (v);
    return u << shift;
}
ull modmul(ull a, ull b, ull M) {
    ll ret = a * b - M * ull(1.L / M * a * b);
    return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
    ull ans = 1;
    for (; e; b = modmul(b, b, mod), e /= 2)
        if (e & 1) ans = modmul(ans, b, mod);
    return ans;
}
bool isPrime(ull n) {
    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
    ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022
        },
        s = __builtin_ctzll(n-1), d = n >> s;
    for (ull a : A) {   // ^ count trailing zeroes
        ull p = modpow(a%n, d, n), i = s;
        while (p != 1 && p != n - 1 && a % n && i--)
            p = modmul(p, p, n);
        if (p != n-1 && i != s) return 0;
    }
    return 1;
}
typedef __uint128_t u128;
ull hi(u128 x) { return (x >> 64); }
ull lo(u128 x) { return (x << 64) >> 64; }
struct Mont {
    Mont(ull n) : mod(n) {
        inv = n;
        fr(i,6) inv *= 2 - n * inv;
        r2 = -n % n;
        fr(i,4) if ((r2 <<= 1) >= mod) r2 -= mod;
        fr(i,5) r2 = mul(r2, r2);
    }
    ull reduce(u128 x) const {
        ull y = hi(x) - hi(u128(lo(x) * inv) * mod);
        return ll(y) < 0 ? y + mod : y;
    }
    ull reduce(ull x) const { return reduce(x); }
    ull init(ull n) const { return reduce(u128(n) * r2); }
    ull mul(ull a, ull b) const { return reduce(u128(a) * b); }
    ull mod, inv, r2;
};
ull pollard(ull n) {
    if (n == 9)
        return 3;
    if (n == 25)
        return 5;
    if (n == 49)
        return 7;
    if (n == 323)
        return 17;
    Mont mont(n);
    auto f = [n, &mont](ull x) { return mont.mul(x, x) + 1; };
    ull x = 0, y = 0, t = 0, prd = 2, i = 1, q;
    while (t++ % 32 || gcd(prd, n) == 1) {
        if (x == y)
            x = ++i, y = f(x);
        if ((q = mont.mul(prd, max(x, y) - min(x, y))))
            prd = q;
        x = f(x), y = f(f(y));
    }
    return gcd(prd, n);
}
//Numeros fatorados neste map (primo -> frequencia)
unordered_map<ll, int> mp_fac;
void factor(ull n) {
    if (n == 1)
        return;
    if (isPrime(n))
        mp_fac[n]++;
    else {
        ull x = pollard(n);
        factor(x), factor(n / x);
    }
}
```

PollardRhoVL.h

Description: PollardRho
Time: $\mathcal{O}(X)$

4d5a84, 67 lines

```
mt19937 rnd(time(0));
ll grand(ll n){
    return uniform_int_distribution<ll>(0,n-1)(rnd);
}
ll mulmod(ll a, ll b, ll mod){
    if(b<0) return mulmod(a,(b%mod+mod)%mod,mod);
    if(b==0) return 0LL;
    ll ans = (2LL*mulmod(a,b/2,mod))%mod;
    if(b%2==0) return ans;
    return (ans+a)%mod;
}
ll exp_mod(ll a, ll x, ll m) {
    if (x == 0) return 1;
    ll res = exp_mod(a, x/2, m);
    res = mulmod(res, res, m); //(res * res) % m;
    if(x % 2 == 1) res = mulmod(res, a, m); // (res * a) % m
    return res;
}
//Rabin Miller
bool ispp(ll n){
    if(n<=1) return 0;
    if(n<=3) return 1;
    ll s = 0, d = n-1;
    while(d%2==0){
        d/=2;
        s++;
    }
    fr(k,64){
        ll a = grand(n-3)+2;
        ll x = exp_mod(a,d,n);
        if(x!=1 and x!=n-1){
            for(int r = 1;r<s;r++){
                x = mulmod(x,x,n);
                if(x==1) return 0;
                if(x==n-1) break;
            }
            if(x!=n-1) return 0;
        }
    }
    return 1;
}
ll rho(ll n){
    ll d, c = grand(n), x = grand(n),xx=x;
    if(n%2==0){
        return 2;
    }
    do{
        x = (mulmod(x,x,n)+c)%n;
        xx = (mulmod(xx,xx,n)+c)%n;
        xx = (mulmod(xx,xx,n)+c)%n;
        d = gcd(abs(x-xx),n);
    } while(d==1);
    return d;
}
//mapa de primo para frequencia
map<ll,int> F;
void factor(ll n){
    if(n==1) return;
    if(ispp(n)){
        F[n]++;
        return;
    }
    ll d = rho(n);
    factor(d);
    factor(n/d);
    return;
}
```

4.3 Divisibility

CRTgcdExtendidoVL.h

Description: Gcd extendido
Time: $\mathcal{O}(X)$

818f48, 53 lines

```
ll div(ll a, ll b, bool ceil){
    ll ans = abs(a/b);
    bool pos = (a<0)==(b<0);
    if(a%b and ceil==pos) ans++;
    if(!pos) ans*=-1;
    return ans;
```

```
}

ll gcd_ext(ll a, ll b, ll &xo, ll &yo){
    if(b==0){
        xo = 1, yo = 0;
        return a;
    }
    ll x1, y1;
    ll g = gcd_ext(b,a%b,x1,y1);
    xo = y1;
    yo = x1-(a/b)*y1;
    return g;
}
/*
Retorna qual o menor x positivo que satisfaz
a*x + b*y = c (obviamente o y correspondente eh negativo)
(ou -1 se nao existe)

Util em CRT para achar menor r positivo que
    r = ra (mod a)
    r = rb (mod b)
    ->
    a*x-b*y = rb-ra
    r = a*x + ra
*/
ll qual_sol(ll a, ll b, ll c){
    ll xo, yo;
    ll g = gcd_ext(a,b,xo,yo);
    if(c%g!=0) return -1;
    c/=g, a/=g,b/=g;
    xo*=c,yo*=c;

    ll k = div(-xo,b,b>0);

    return xo+k*b;
}
/*
    Return minimun r such that:
        r = ra (mod a)
        r = rb (mod b)
    Or -1 if no such r
*/
ll solve_crt(ll ra, ll a, ll rb, ll b){
    ll minx = qual_sol(a,-b,rb-ra);
    if(minx==-1) return minx;
    return a*minx+ra;
}
```

4.3.1 Bézout's identity

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

$$ax + by = d$$

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

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

phiFunction.h

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

Euler's thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod n$.
Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod p$ $\forall a$.

cf7d6d, 8 lines

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

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

4.4 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

4.5 Primes

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

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

4.6 Mobius Function

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

Mobius Inversion:

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

Other useful formulas/forms:

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

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

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

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.) Integer -> permutation can use a lookup table.
Time: $\mathcal{O}(n)$

044568, 6 lines

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

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 Lucas' Theorem

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

5.2.3 Binomials

multinomial.h

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

a0a312, 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;
}
```

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:

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

Euler-Maclaurin formula for infinite sums:

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

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

5.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k), \ c(0,0) = 1$$

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

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

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.

Graph (6)

6.1 Network flow

PushRelabel.h

Description: Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.

Time: $\mathcal{O}\left(V^2\sqrt{E}\right)$

0ae1d4, 48 lines

```
struct PushRelabel {
  struct Edge {
    int dest, back;
    ll f, c;
  };
  vector<vector<Edge>> g;
  vector<ll> ec;
  vector<Edge*> cur;
  vector<vi> hs; vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}

  void addEdge(int s, int t, ll cap, ll rcap=0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s])-1, 0, rcap});
  }

  void addFlow(Edge& e, ll f) {
    Edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
    e.f += f; e.c -= f; ec[e.dest] += f;
    back.f -= f; back.c += f; ec[back.dest] -= f;
  }
  ll calc(int s, int t) {
    int v = sz(g); H[s] = v; ec[t] = 1;
    vi co(2*v); co[0] = v-1;
    rep(i,0,v) cur[i] = g[i].data();
    for (Edge& e : g[s]) addFlow(e, e.c);

    for (int hi = 0;;) {
      while (hs[hi].empty()) if (!hi--) return -ec[s];
      int u = hs[hi].back(); hs[hi].pop_back();
      while (ec[u] > 0)  // discharge u
        if (cur[u] == g[u].data() + sz(g[u])) {
          H[u] = 1e9;
          for (Edge& e : g[u]) if (e.c && H[u] > H[e.dest]+1)
            H[u] = H[e.dest]+1, cur[u] = &e;
          if (++co[H[u]], !--co[hi] && hi < v)
            rep(i,0,v) if (hi < H[i] && H[i] < v)
              --co[H[i]], H[i] = v + 1;
          hi = H[u];
        } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
          addFlow(*cur[u], min(ec[u], cur[u]->c));
        else ++cur[u];
    }
  }
  bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

MinCostMaxFlowVlamarca.h

Description: MinCostMaxFlow

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

3f0a76, 140 lines

```
const int maxN = 310;
const double eps = 1e-6;
#define mset(v,x) memset(v,x,sizeof(v))

template<class T> bool lessT(const T &a, const T &b) { return a
      < b; }
template<> bool lessT(const double &a, const double &b) {
     return a < b - eps; }
template<class T> bool equalT(const T &a, const T &b) { return
     a == b; }
template<> bool equalT(const double &a, const double &b) {
     return fabs(a - b) < eps; }

template<typename T> struct costFlow {
  struct edge_t {
    int v, r; T w; int next;
    edge_t(int v, int r, T w, int next) : v(v), r(r), w(w),
         next(next) { }
  };
  vector<edge_t> edges;
  int h[maxN], vis[maxN];
  T d[maxN];

  void clear() {
    edges.clear(); mset(h, -1);
  }

    //r eh o flow e w o custo
  void addE(int u, int v, int r, T w) {
      u++,v++;
    edges.push_back(edge_t(v, r, w, h[u]));
    h[u] = sz(edges)-1;
    edges.push_back(edge_t(u, 0, -w, h[v]));
    h[v] = sz(edges)-1;
  }

  void spfa(int s, int t, int n) {
    queue<int> q;
```

```
    fill(d + 1, d + 1 + n, numeric_limits<T>::max());
    fill(vis + 1, vis + 1 + n, false);
    d[s] = 0, q.push(s), vis[s] = true;
    while (!q.empty()) {
      int u = q.front();
      q.pop(), vis[u] = false;
      for (int i = h[u]; i != -1; i = edges[i].next) {
        const edge_t &e = edges[i];
        if (e.r and lessT(d[u] + e.w, d[e.v])) {
          d[e.v] = d[u] + e.w;
          if (!vis[e.v]) {
            q.push(e.v);
            vis[e.v] = true;
          }
        }
      }
    }
    for (int i = 1; i <= n; ++i) {
      if (i != t) d[i] = d[t] - d[i];
    }
    d[t] = 0;
  }

  int augment(int u, int t, int flow) {
    if (u == t) return flow;
    vis[u] = true;
    int ret = 0;
    for (int i = h[u]; i != -1; i = edges[i].next) {
      int v = edges[i].v, r = edges[i].r; T w = edges[i].w;
      if (r and !vis[v] and equalT(d[v] + w, d[u])) {
        int temp = augment(v, t, min(flow, r));
        if (temp) {
          edges[i].r -= temp, edges[i ^ 1].r += temp;
          ret += temp, flow -= temp;
          if (flow == 0) break;
        }
      }
    }
    return ret;
  }

  bool adjust(int n) {
    T delta = numeric_limits<T>::max();
    for (int u = 1; u <= n; ++u) {
      if (!vis[u]) continue;
      for (int i = h[u]; i != -1; i = edges[i].next) {
        const edge_t &e = edges[i];
        if (e.r and !vis[e.v] and lessT(d[u], d[e.v] + e.w)) {
          delta = min(delta, d[e.v] + e.w - d[u]);
        }
      }
    }
    if (delta == numeric_limits<T>::max()) return false;
    for (int i = 1; i <= n; ++i) {
      if (vis[i]) d[i] += delta;
    }
    mset(vis,0);
    return true;
  }

    T getCost(){
        T cost = 0;
        for (int i = 1; i < (int) edges.size(); i += 2) cost +=
              edges[i].r * edges[i - 1].w;
        return cost;
    }

    /*returns a vector flow_to_cost such that
    flow_to_cost[i] is the minimun cost of a assignment having
        i+1 of flow*/
    vector<T> listAllCosts(int s, int t, int n) {
      s++,t++;
      int temp, flow = 0;
      spfa(s, t, n);
      vector<T> costs;
      do {
        while ((temp = augment(s, t, 1))) {
          flow += temp;
          costs.push_back(getCost());
          mset(vis,0);
        }
      } while (adjust(n));
      return costs;
    }

    //returns pair {maxflow,mincost}. n is the number of used
        vertices
    pair<int, T> minCostMaxFlow(int s, int t, int n) {
        s++,t++;
      int temp, flow = 0;
      spfa(s, t, n);
      do {
        while ((temp = augment(s, t, INT_MAX))) {
          flow += temp;
          mset(vis,0);
        }
      } while (adjust(n));
      T cost = getCost();
      return make_pair(flow, cost);
    }
};

int main(){
    costFlow<ll> cf;
    cf.clear();
    for(auto &[a,b,c] : vt){
        cf.addE(a,N+b,1,maxv-c);
    }
    //...
}
```

DinicVlamarca.h

Description: Dinic
Time: $\mathcal{O}(X)$ — 4b716b, 107 lines

```
const int MAXV = 3e3+10; // maximo numero de vertices
const int FINF = INT_MAX; // infinite flow

struct Edge {
    int to;
    int cap;
    Edge(int t, int c)
    {
        to = t;
        cap = c;
    }
};

vector<int> adj[MAXV];
vector<Edge> edge;
vector<Edge> eo;
int ptr[MAXV], dinic_dist[MAXV];

// Inserts an edge u->v with capacity c
inline void add_edge(int u, int v, int c)
{
    adj[u].push_back(edge.size());
```

```
    edge.push_back(Edge(v, c));
    adj[v].push_back(edge.size());
    edge.push_back(Edge(u, 0)); // modify to Edge(u,c) if graph
          is non-directed
}

bool dinic_bfs(int _s, int _t)
{
    memset(dinic_dist, -1, sizeof(dinic_dist));
    dinic_dist[_s] = 0;
    queue<int> q;
    q.push(_s);
    while (!q.empty() && dinic_dist[_t] == -1) {
        int v = q.front();
        q.pop();
        for (size_t a = 0; a < adj[v].size(); ++a) {
            int ind = adj[v][a];
            int nxt = edge[ind].to;
            if (dinic_dist[nxt] == -1 && edge[ind].cap) {
                dinic_dist[nxt] = dinic_dist[v] + 1;
                q.push(nxt);
            }
        }
    }
    return dinic_dist[_t] != -1;
}

int dinic_dfs(int v, int _t, int flow)
{
    if (v == _t)
        return flow;
    for (int& a = ptr[v]; a < (int)adj[v].size(); ++a) {
        int ind = adj[v][a];
        int nxt = edge[ind].to;
        if (dinic_dist[nxt] == dinic_dist[v] + 1 && edge[ind].
            cap) {
            int got = dinic_dfs(nxt, _t, min(flow, edge[ind].
                cap));
            if (got) {
                edge[ind].cap -= got;
                edge[ind ^ 1].cap += got;
                return got;
            }
        }
    }
    return 0;
}

int dinic(int _s, int _t)
{
    eo = edge; // qnd for fazer o fluxo, guardar como eram as
          capacidades originais (na vdd isto eh o grafo residual
           - quanto tem disponivel pra ir de fluxo) para poder
          recuperar a resposta
    int ret = 0, got;
    while (dinic_bfs(_s, _t)) {
        memset(ptr, 0, sizeof(ptr));
        while ((got = dinic_dfs(_s, _t, FINF)))
            ret += got;
    }
    return ret;
}

// Clears dinic structure
inline void dinic_clear(int n_vertices)
{
    for (int a = 0; a < n_vertices; ++a)
        adj[a].clear();
    edge.clear();
```

```
}

typedef tuple<int,int,int> tii;

/* rec_ans recupera resposta do fluxo do dinic
   returna tupla (u,v,c) quanto de fluxo (c) passa de u pra v (
        direcionado)
   (nao adiciona aresta se nao passa nd de fluxo nela)

   Lembrar de por em resposta apenas os vertices necessarios
   (geralmente tenho o source e sink a mais por exemplo)
*/
vector<tii> rec_ans(int n_vertices){
    vector<tii> ans;
    fr(i,n_vertices){
        for(auto &ide : adj[i]){
            if(eo[ide].cap>edge[ide].cap){
                ans.emplace_back(i,edge[ide].to,eo[ide].cap-
                    edge[ide].cap);
            }
        }
    }
    return ans;
}
```

GomoryHu.h

Description: Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.
Time: $\mathcal{O}(V)$ Flow Computations

"PushRelabel.h" 0418b3, 13 lines

```
typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
  vector<Edge> tree;
  vi par(N);
  rep(i,1,N) {
    PushRelabel D(N); // Dinic also works
    for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
    tree.push_back({i, par[i], D.calc(i, par[i])});
    rep(j,i+1,N)
      if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
  }
  return tree;
}
```

KuhnMunkras.h

Description: Weighted bipartite matching
Time: $\mathcal{O}(N^3)$

8f2214, 37 lines

```
//calculate
void km(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, INF);
    vector<char> used (m+1, false);
    do {
      used[j0] = true;
      int i0 = p[j0],  delta = INF,  j1;
      for (int j=1; j<=m; ++j)
        if (!used[j]) {
          int cur = a[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);
  }
}
//get answer, if there are extra edge remember to remove
vector<int> ans (n+1);
for (int j=1; j<=m; ++j)
  ans[p[j]] = j;
```

6.2 Matching

DFSMatching.h

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

522b98, 22 lines

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

MinimumVertexCover.h

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

"DFSMatching.h" da4196, 20 lines

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

WeightedMatching.h

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

1e0fe9, 31 lines

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

GeneralMatching.h

Description: Matching for general graphs. Fails with probability N/mod.
Time: $\mathcal{O}(N^3)$

"../numerical/MatrixInverse-mod.h" cb1912, 40 lines

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

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

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

```
      }
    }
  } while (matInv(A = mat) != M);

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

6.3 DFS algorithms

SCCVlamarca.h

Description: SCC
Time: $\mathcal{O}(X)$

1eb558, 47 lines

```
const int N = 5e5+10;
vector<int> g[N];
vector<int> comp_to_nos[N];
int tempo;
int disc[N]; //primeiro tempo em que noh foi descoberto
int low[N]; //minimo entre disc[no] e low[v] dos vizinhos

//stack e size of stack
int st[N], ss;

//componente do noh i (0 se ainda nao pertence a componente)
//comp[no] : [1,ncomp]
int comp[N], ncomp;

int dfs(int no){
    disc[no] = low[no] = ++tempo;
    st[ss++] = no;
    for(auto it : g[no]){
        if(!disc[it]) low[no] = min(low[no],dfs(it));
        else if(!comp[it]) low[no] = min(low[no],disc[it]);
    }
    if(low[no]==disc[no]){
        comp[no] = ++ncomp;
        while(st[ss-1]!=no) comp[st[--ss]] = comp[no];
        ss--;
    }
    return low[no];
}

/*
    Poe condicao (u or v) no 2sat
    se du==1, u eh 2*u+1 (impar) e significa
    que eh u normal (verdadeiro), do contrario eh not u
*/
void poe(int u, int v, int du, int dv){
    u = 2*u+du;
    v = 2*v+dv;
    g[u^1].push_back(v);
    g[v^1].push_back(u);
}

int main(){
    //rodar tarjan e definir comps de cada no
    fr(i,n) if(!disc[i]) dfs(i);
    //comp_to_nos, nem sempre necessario, comp 1-indexado
    fr(i,n) comp_to_nos[comp[i]].push_back(i);
}
```

BiconnectedComponents.h

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

2965e5, 33 lines

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

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

EulerCycleVlamarca.h

Description: Euler Cycle
Time: $\mathcal{O}(X)$

cb8f43, 39 lines

```
vector<pair<int,int>> g[N];
namespace eulerpath_space{
vector<int> path;
vector<int> idit, used_edge;

void dfs(int no){
    while(1){
        int &id = idit[no];
        while(id<sz(g[no]) and used_edge[g[no][id].second]) id
            ++;
        if(id==sz(g[no])) break;
        used_edge[g[no][id].second] = 1;
        int it = g[no][id++].first;
        dfs(it);
    }
    path.push_back(no);
}

/*
    For undirected graph g (adjacency list with of pair (it,
        id_edge))
    True if graph has eulerian cycle
    If true, path will have the nodes in the order of a cycle

    For directed version check if outdegree==indegree for every
        node
    (except initial and final node if eulerian path) and other
        changes
    submission: https://cses.fi/problemset/result/1981318/
*/
bool has_cycle(int n, int m){
    int inic = 0, nimp = 0;
    fr(i,n) if(sz(g[i])&1) nimp++, inic = i;
    //to change to eulerian path instead of cycle allow nimp==2
    if(nimp>0) return 0;
    path.clear();
    idit = vector<int>(n);
    used_edge = vector<int>(m);
    dfs(inic);
    if(sz(path)==m+1) return 1;
    return 0;
}
}; //end hpath_space
```

6.4 Coloring

EdgeColoring.h

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

e210e2, 31 lines

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

```
  rep(i,0,sz(eds))
    for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
  return ret;
}
```

6.5 Heuristics

MaximumClique.h

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

f7c0bc, 49 lines

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

MaximumIndependentSet.h

Description: To obtain a maximum independent set of a graph, find a max clique of the complement. If the graph is bipartite, see MinimumVertexCover.

6.6 Trees

LCAVlamarca.h

Description: LCA
Time: $\mathcal{O}(X)$

65450b, 59 lines

```
//LEMBRAR DE POR O MAKE DEPOIS DE MONTAR A ARVORE
const int N = 5e5+10;
namespace lca_space{
int nlog;
int n;
vector<int> *g;
int pai[N], dist[N]; //pai do no i (raiz = -1)
int st[N][25]; //sparse table - st[i][j] = pai 2^j niveis acima
    do no i
void dfs(int no, int from, int dac){
    dist[no] = dac;
    for(auto it : g[no]){
        if(it==from) continue;
        pai[it] = no;
        dfs(it,no,dac+1);
    }
}
void make(vector<int> _g[N], int _n, int root){
    g = _g;
    n = _n;
    pai[root] = -1;
    dfs(root,-1,0);
    nlog = 1;
    while((1<<nlog)<n) nlog++;
    assert(nlog<25);
    fr(i,n) fr(j,nlog+1) st[i][j] = -1;
    fr(i,n) st[i][0] = pai[i];
    for(int j = 1; j<=nlog; j++){
        fr(i,n){
            int ant_pai = st[i][j-1];
            if(ant_pai!=-1) st[i][j] = st[ant_pai][j-1];
        }
    }
}
int go_up(int no, int k){
    for(int i = nlog; i>=0; i--){
        if((1<<i)<=k and no!=-1){
            no = st[no][i];
            k-=(1<<i);
        }
    }
    if(k==0) return no;
    return -1;
}
int lca(int p, int q){
    if(dist[p]<dist[q]) swap(p,q);
    p = go_up(p,dist[p]-dist[q]);
    if(p==q) return p;
    for(int i = nlog; i>=0; i--){
        if(st[p][i]!=st[q][i]){
            p = st[p][i];
            q = st[q][i];
        }
    }
    return pai[q];
}
int get_dist(int u, int v){
    return dist[u]+dist[v]-2*dist[lca(u,v)];
}
}; //end lca_space
```

CompressTree.h

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

"LCA.h" 9775a0, 21 lines

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

HLDVlamarca.h

Description: HLD
Time: $\mathcal{O}(X)$

2629e5, 47 lines

```
template<int N, bool IN_EDGES> struct HLD {
    int t;
    vector<int> *g;
    int pai[N], sz[N], d[N];
    int root[N], pos[N];
    void dfsSz(int no) {
        if (~pai[no]) g[no].erase(find(all(g[no]),pai[no]));
        sz[no] = 1;
        for(auto &it : g[no]) {
            pai[it] = no; d[it] = d[no]+1;
            dfsSz(it); sz[no] += sz[it];
            if (sz[it] > sz[g[no][0]]) swap(it, g[no][0]);
        }
    }
    void dfsHld(int no) {
        pos[no] = t++;
        for(auto &it : g[no]) {
            root[it] = (it == g[no][0] ? root[no] : it);
            dfsHld(it); }
    }
    void init(int nor, vector<int> *_g) {
        g = _g;
        root[nor] = d[nor] = t = 0; pai[nor] = -1;
        dfsSz(nor); dfsHld(nor); }
    Seg<N> tree; //lembrar de ter build da seg sem nada
    void changeNode(int v, node val){
        tree.upd(pos[v],val);
    }
    node querySubtree(int v){
        node ans = tree.qry(pos[v]+IN_EDGES,pos[v]+sz[v]);
        return ans;
    }
    template <class Op>
    void processPath(int u, int v, Op op) {
        for (; root[u] != root[v]; v = pai[root[v]]) {
            if (d[root[u]] > d[root[v]]) swap(u, v);
```

```
            op(pos[root[v]], pos[v]); }
        if (d[u] > d[v]) swap(u, v);
        op(pos[u]+IN_EDGES, pos[v]);
    }
    node queryPath(int u, int v) { //modificacoes geralmente
        vem aqui (para hld soma)
        node res; processPath(u,v,[this,&res](int l,int r) {
            res = oper(tree.qry(l,r+1),res); });
        return res;
    }
};
// HLD<N,false> hld;
```

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

5909e2, 90 lines

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

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

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

6.7 Math

6.7.1 Number of Spanning Trees

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

6.7.2 Erdős–Gallai theorem

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

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

Geometry (7)

7.1 Geometric primitives

Point.h

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

47ec0a, 28 lines

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

PtRotForcaModVL.h

Description: PT rot

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

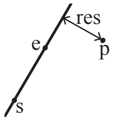
aa0fb5, 37 lines

```
const long double pi = acos(-1.0l);

struct pt{
    long double x, y;
    //... construtor
    long double mod(){
        return sqrt(sq(x)+sq(y));
    }
    pt operator -(pt b){
        return pt(x-b.x,y-b.y);
    }
    long double operator ^(pt b) const{
        return x*b.y-y*b.x;
    }
    int quad() const{
        int ans = 0;
        if(x<0) ans++;
        if(y<0) ans^=1, ans+=2;
        return ans;
    }
    bool operator <(pt b) const{
        if(this->quad()==b.quad()){
            return ((*this)^b)>0;
        }
        return this->quad()<b.quad();
    }
};
```

```
//rotaciona sentido anti horario
pt rot(pt p, double teta){
    return pt(p.x*cos(teta)-p.y*sin(teta),p.y*cos(teta)+p.x*sin
         (teta));
}

pt forca_mod(pt p, long double m){
    long double cm = p.mod();
    return pt(p.x*m/cm,p.y*m/cm);
}
```

lineDistance.h

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



"Point.h" f6bf6b, 4 lines

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

SegmentIntersectionVL.h

Description: SegmentIntersect
Time: $\mathcal{O}(X)$

afcef2, 25 lines

```
//checa se ponto esta dentro do segmento
bool inptseg(pt a1, pt b1, pt b2){
    pt v1 = (a1-b1), v2 = (b2-b1);
    if(v1^v2) return 0;
    v1 = b1-a1, v2 = b2-a1;
    return (v1*v2)<=0;
}

//checa se segmentos intersectam (bordas inclusas)
bool seg_intersect(pt a1, pt a2, pt b1, pt b2){
    fr(i,2){
        fr(j,2){
            if(inptseg(a1,b1,b2)) return 1;
            swap(a1,a2);
        }
        swap(a1,b1);
        swap(a2,b2);
    }
    fr(cor,2){
        pt v1 = (a1-b1), v2 = (a2-b1), vs = (b2-b1);
        if( 1.0l*(v1^vs)*(v2^vs) >= -0.5 ) return 0;
        swap(a1,b1), swap(a2,b2);
    }
    return 1;
}
```

7.2 Circles

CircleIntersection.h

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

"Point.h" 84d6d3, 11 lines

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

CircleTangents.h

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

"Point.h" b0153d, 13 lines

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

CirclePolygonIntersection.h

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

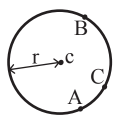
"../../content/geometry/Point.h" a1ee63, 19 lines

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

circumcircle.h

Description:

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



"Point.h" 1caa3a, 9 lines

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

MinimumEnclosingCircle.h

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

"circumcircle.h" 09dd0a, 17 lines

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

7.3 Polygons

ConvexHullVL.h

Description: ConvexHull
Time: $\mathcal{O}(X)$

e6c39b, 38 lines

```
struct pt //...
bool operator <(const pt &p1, const pt &p2){
    return pll(p1.x,p1.y)<pll(p2.x,p2.y);
}
bool operator ==(const pt &p1, const pt &p2){
    return pll(p1.x,p1.y)==pll(p2.x,p2.y);
}
/*
    gera convex hull em ordem ccw
    pontos colineares sao retirados
    no fr(cor,2) porimeiro se faz o lower hull
    depois o upper

    unico corner eh se pontos forem tds colineares,
    ai o ch eh degenerado
*/
vector<pt> mch(vector<pt> v){
    sort(all(v));
    v.resize(unique(all(v))-v.begin());
    vector<pt> ans;
    fr(cor,2){
        vector<pt> h;
        fr(i,v.size()){
            while(h.size()>=2){
                pt v1 = h.back()-h[h.size()-2];
                pt v2 = v[i]-h.back();
                if( (v1^v2) > 0 ) break;
                h.pop_back();
            }
            h.eb(v[i]);
        }
        fr(i,(int)h.size()-1){
            ans.eb(h[i]);
        }
        reverse(all(v));
    }
    return ans;
```

```
}
```

ConvexHullTrickVL.h

Description: CHT
Time: $\mathcal{O}(X)$

6fbc7b, 43 lines

```
struct Line{
    //reta da forma y=x*m+k
    // p eh onde a reta para, onde deixa de ser a maxima
    mutable ll m,k,p;
    // lembrar de fazer funcao const
    bool operator <(const Line &o) const { return m<o.m;}
    bool operator <(ll x) const { return p<x;}
};

struct Cht : multiset<Line,less<>> {
    const ll inf = LLONG_MAX;
    ll div(ll a, ll b){
        return a/b - ( (a^b)<0 and a%b);
    }
    bool bad2(iterator x, iterator y){
        if(y==end()){
            x->p = inf;
            return 0;
        }
        if(x->m==y->m) x->p = (x->k>y->k)? inf : -inf;
        else x->p = div(x->k-y->k, y->m-x->m);
        return x->p>=y->p;
    }
    //tenta adicionar a reta y=m*x+k ao CHTmaximo e o ajusto
        caso necessario
    void add(ll m, ll k){
        auto z = insert({m,k,0}), y = z++, x = y;
        while(bad2(y,z)) z = erase(z);
        if(x!=begin() and bad2(--x,y)) bad2(x,erase(y));  //no
            original se usa y=erase(y), mas nao precisa pois y
             sera redefinido na proxima linha msm
        while((y=x)!=begin() and (--x)->p >= y->p) bad2(x,erase
            (y));
    }
    //retorna o valor do qual do convex hull de maximo na
        coordenada x
    ll query(ll x){
        assert(!empty());
        auto l = *lower_bound(x);
        return x*l.m + l.k;
    }
};

int main(){
    Cht cht;
    cht.add(0,0);
    cht.query(y)
}
```

7.4 Misc. Point Set Problems

ClosestPair.h

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

"Point.h" ac41a6, 17 lines

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

kdTree.h

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

"Point.h" bac5b0, 63 lines

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

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

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

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

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

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

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

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

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

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

FastDelaunay.h

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

"Point.h" eefdf5, 88 lines

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

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

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

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

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

```
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;

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

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

HalfPlaneVL.h

Description: Half Place Intersection
Time: $\mathcal{O}(X)$

a0c844, 220 lines

```
/*
    Half plane intersection implementado com precisao inteira
    cuidado overflow
    coordenadas tais que 16*x^4<=1e18 (x<=1e4)
*/
struct pt{
    ll x, y;
    pt(){}
    pt(ll a, ll b){
        x = a, y = b;
    }

    pt operator -(pt p2){
        return pt(x-p2.x,y-p2.y);
    }

    pt esc(ll e){
        return pt(x*e,y*e);
    }
};

ll operator ^(const pt &p1, const pt &p2){
    return p1.x*p2.y-p1.y*p2.x;
}
struct seg{
    pt p2, p1;
    pt v;
    seg(){}
    seg(pt p11, pt p12){
        p2 = p12, p1 = p11;
        v = p2-p1;
    }
    seg esc(ll e){
        return seg(p1.esc(e),p2.esc(e));
    }
};
int quad(const pt &v){
    int ans = 0;
    if(v.x<0) ans++;
    if(v.y<0) ans+=2, ans^=1;
    return ans;
}

bool operator <(const seg &a, const seg &b){
    if(quad(a.v)!=quad(b.v)) return quad(a.v)<quad(b.v);
    return (a.v^b.v) > 0;
}

bool operator ==(const seg &a, const seg &b){
    return quad(a.v)==quad(b.v) and (a.v^b.v)==0;
}

/*
    para segmentos paralelos,
    retorna 1 se s1 esta a esquerda de s2
    (ou sao msm reta)
*/
int a_esquerda(seg s1, seg s2){
    pt v2 = s1.p2-s2.p1;
    return (s2.v^v2)>=0;
}

seg oposto(seg s){
    return seg(s.p2,s.p1);
}

auto prox(auto it, list<seg> &l){
    it++;
    if(it==l.end()) it = l.begin();
    return it;
}

auto prev(auto it, list<seg> &l){
    if(it==l.begin()) it = l.end();
    it--;
    return it;
}

struct ptd{
    long double x, y;
    ptd(){}
    ptd(long double a, long double b){
        x = a, y = b;
    }
    ptd operator -(ptd p2){
        return ptd(x-p2.x,y-p2.y);
    }
    long double operator ^(ptd p2){
        return x*p2.y-y*p2.x;
    }
};

int line_intersection(pt p1, pt p2, pt p3, pt p4, pt &p, ll &
 det){
    ll a1, a2, b1, b2, c1, c2;
    fr(cor,2){
        a1 = p1.y-p2.y, b1 = p2.x-p1.x;
        c1 = p1.x*a1 + p1.y*b1;
        swap(p1,p3);
        swap(p2,p4);
        swap(a1,a2);
        swap(b1,b2);
        swap(c1,c2);
    }
    det = a1*b2-a2*b1;
    assert(det);
    p = pt(c1*b2-b1*c2,a1*c2-a2*c1);
    return 1;
}

int line_intersection(pt p1, pt p2, pt p3, pt p4, ptd &p){
    ll a1, a2, b1, b2, c1, c2;
    fr(cor,2){
        a1 = p1.y-p2.y, b1 = p2.x-p1.x;
        c1 = p1.x*a1 + p1.y*b1;
        swap(p1,p3);
        swap(p2,p4);
        swap(a1,a2);
        swap(b1,b2);
        swap(c1,c2);
    }
    ll det = a1*b2-a2*b1;
    assert(det);
    p = ptd( (1.0l*c1*b2-b1*c2)/det,
             (1.0l*a1*c2-a2*c1)/det );
    return 1;
}

int tira(seg a, seg b, seg c){
    pt pi;
    ll e;
    line_intersection(a.p1,a.p2,c.p1,c.p2,pi,e);
    b = b.esc(e);
    pt v2 = pi-b.p1;
    return (b.v^v2) >= 0;
}

long double area;

void farea(vector<seg> v){
    vector<ptd> vd;
    fr(i,v.size()){
        int i1 = i+1;
        if(i1==v.size()) i1 = 0;
        ptd p;
        line_intersection(v[i].p1,v[i].p2,v[i1].p1,v[i1].p2,p);
        vd.eb(p);
    }
    area = 0;
    frr(i,1,(int)vd.size()-1){
        int i1 = i+1;
        if(i1==vd.size()) i1 = 0;
        ptd v1 = vd[i1]-vd[0];
        ptd v2 = vd[i]-vd[0];
        area += fabs(v1^v2);
    }
    area/=2;
}

/*
    retorno 0 -> area nula
            1 -> area finita
            2 -> area infinita
*/
int hp(vector<seg> v){
    sort(all(v));
    {
```

```
        vector<seg> aux;
        int i = 0;
        while(i<v.size()){
            seg cur = v[i];
            while(i+1<v.size() and (v[i+1]==v[i])){
                i++;
                if(a_esquerda(v[i],cur)) cur = v[i];
            }
            aux.eb(cur);
            i++;
        }
        v = aux;
    }

    {
        int i = 0, j = 0;
        while(1){
            j = max(j,i+1);
            while(j<v.size() and (v[i].v^v[j].v)>0) j++;
            if(j==v.size()) break;
            if((v[i].v^v[j].v)==0 and a_esquerda(v[i],oposto(v[
                j]))) return 0;
            i++;
        }
    }

    fr(i,v.size()) if((v[i].v^v[(i+1)%v.size()].v) <=0) return
        2;

    list<seg> l(all(v));
    assert(l.size()>=3);

    fr(tt,2) for(auto it = l.begin();it!=l.end();it++){
        fr(cor,2) while(1){
            auto it1 = it;
            auto it2 = prox(it1,l);
            auto it3 = prox(it2,l);
            if(cor){
                it3 = it;
                it2 = prev(it3,l);
                it1 = prev(it2,l);
            }
            if((it1->v^it3->v) <= 0){
                if(!cor and tira(*it2,oposto(*it1),*it3))
                    return 0;
                else if(cor and tira(*it1,oposto(*it3),*it2))
                    return 0;
                else break;
            } else{
                if(tira(*it1,*it2,*it3)) l.erase(it2);
                else break;
            }
        }
    }
    v = vector<seg>(all(l));
    farea(v);
    return 1;
}
```

AngleSweepVL.h

Description: AngleSweep
Time: $\mathcal{O}(X)$

c16500, 20 lines

```
struct pt //{...}
ll operator ^(const pt &a, const pt &b){
    return a.x*b.y-a.y*b.x;
}
int quad(const pt& p){
    int ans = 0;
    if(p.x<0) ans++;
    if(p.y<0) ans^=1, ans+=2;
    return ans;
}
bool operator ==(const pt &a, const pt &b){
    return quad(a)==quad(b) and (a^b)==0;
}
// Vector by angle comparator - return 0 if they are equal
bool operator <(const pt &a, const pt &b){
    if(quad(a)==quad(b)){
        return (a^b)>0;
    }
    return quad(a)<quad(b);
}
```

7.5 3D

PolyhedronVolume.h

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

3058c3, 6 lines

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

Point3D.h

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

8058ae, 32 lines

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

3dHull.h

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

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

"Point3D.h" 5b45fc, 49 lines

```
typedef Point3D<double> P3;

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

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

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

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

sphericalDistance.h

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

611f07, 8 lines

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

```
}
```

Strings (8)

KMP.h

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

d4375c, 16 lines

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

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

Zfunc.h

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

ee09e2, 12 lines

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

Manacher.h

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

e7ad79, 13 lines

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

SuffixArrayVL.h

Description: SufArray
Time: $\mathcal{O}(X)$

99f3f4, 61 lines

```
/*
    Retorna vector p, suffix array
    p[0] eh o indice do menor menor sufixo
    para usar em vector de inteiros, fazer compressao de
        coordenadas
    para [1,n] e o caracter especial adicionado sera o 0
    Note que n eh incrementado e com caracter especial valores
        sao [0,n-1]
*/
//vector<int> make_suf(vector<int> s){
vector<int> make_suf(string s){
    s+=(char)0;
    //s.push_back(0);
    int n = sz(s);
    vector<int> p(n), c(n), cnt(max(256,n));

    fr(i,n) cnt[s[i]]++;
    for(int i = 1; i<max(n,256); i++) cnt[i]+=cnt[i-1];
    fr(i,n) p[--cnt[s[i]]] = i;
    int nc = 1;
    for(int i = 1; i<n; i++){
        if(s[p[i]]!=s[p[i-1]]) nc++;
        c[p[i]] = nc-1;
    }

    vector<int> pn(n), cn(n);
    for(int k = 0; (1<<k)<n; k++){
        fr(i,n) pn[i] = (p[i]-(1<<k)+n)%n;
        fr(i,nc) cnt[i] = 0;
        fr(i,n) cnt[c[i]]++;
        for(int i = 1; i<nc; i++) cnt[i]+=cnt[i-1];
        for(int i = n-1; i>=0; i--) p[--cnt[c[pn[i]]]] = pn[i];
        nc = 1;
        cn[p[0]] = 0;
        for(int i = 1; i<n; i++){
            int v1 = n*c[p[i]] + c[(p[i]+(1<<k))%n];
            int v2 = n*c[p[i-1]] + c[(p[i-1]+(1<<k))%n];
            if(v1!=v2) nc++;
            cn[p[i]] = nc-1;
        }
        c = cn;
    }
    p.erase(p.begin());
    return p;
}
//vector<int> make_lcp(vector<int> &s, vector<int> &p){
vector<int> make_lcp(string &s, vector<int> &p){
    int n = sz(s);
    vector<int> rank(n), lcp(n-1);
    fr(i,n) rank[p[i]] = i;
    int k = 0;
    fr(i,n){
        if(rank[i]==n-1){
            k = 0;
            continue;
        }
        int j = p[rank[i]+1];
        while(i+k<n and j+k<n and s[i+k]==s[j+k]) k++;
        lcp[rank[i]] = k;
        if(k) k--;
    }
    return lcp;
}
```

SuffixTree.h

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

aae0b8, 50 lines

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

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

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

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

HashVlamarca.h

Description: Hash
Time: $\mathcal{O}(X)$

20f0f3, 55 lines

```
typedef unsigned long long ull;
ull fpull(ull x, ull e) {
    ull ans = 1;
    for(; e > 0; e /= 2) {
        if(e & 1) ans = ans * x;
        x = x * x;
```

```
    }
    return ans;
}
mt19937 rng(time(0));
vector<int> perm;
ull p27[N];
ull inv27[N];
void init_hash(int n){
    fr(i,26) perm.push_back(i+1);
    shuffle(all(perm),rng);
    p27[0] = inv27[0] = 1;
    for(int i = 1; i<n; i++){
        p27[i] = 27*p27[i-1];
        inv27[i] = fpull(p27[i],-1);
    }
}

/*
    Calcula hash de intervalos da string
        primeira letra eh digito menos significativo
        string de lowercase english letters
        Base 27 eh usada, cada letra eh mapeada para [1,26],
            nao tem 0
        Modulo eh (1<<64) - unsigned long long
*/
struct meuhash{
    vector<ull> pref;
    meuhash(){}
    meuhash(string &s){
        assert(sz(s)<N);
        assert(p27[1]*inv27[1]==1);
        pref.resize(sz(s));
        ull cur = 0;
        fr(i,sz(s)){
            cur += p27[i]*perm[s[i]-'a'];
            pref[i] = cur;
        }
    }
    //intervalo fechado [l,r]
    ull gethash(int l, int r){
        assert(l<=r and l>=0 and l<sz(pref) and r>=0 and r<sz(
            pref));
        l--;
        ull ans = pref[r];
        if(l>=0){
            ans -= pref[l];
            ans *= inv27[l+1];
        }
        return ans;
    }
}; //end hash
```

PrefAutomatonVL.h

Description: PrefAutomaton
Time: $\mathcal{O}(X)$

55c4f8, 35 lines

```
/*
    Constroi automato de sufixo da string (usa kmp)
        Para string de tamanho n, ha n+1 estados (de [0,n])
        estado 0 eh nada da string e n eh tudo da string (estou
            na ultima letra)

    prox[c][i] = proximo estado dado que estou no estado i apos
        adicionar letra c

        note que para string "aaaa"
        prox['a'][4] = 4 (continuo na string completa)
*/
int prox[26][N];
vector<int> fpref(string &s){
    vector<int> pref(s.size());
    for(int i = 1; i<sz(s); i++){
        int t = pref[i-1];
        while(t and s[i]!=s[t]) t = pref[t-1];
        if(s[i]==s[t]) t++;
        pref[i] = t;
    }
    return pref;
}
void build_aut(string &s){
    vector<int> pref = fpref(s);
    int n = sz(s);
    vector<int> v(n);
    fr(i,n) v[i] = s[i]-'a';
    fr(c,26) prox[c][0] = 0;
    prox[v[0]][0] = 1;
    for(int i = 1; i<=n; i++){
        fr(c,26){
            prox[c][i] = prox[c][pref[i-1]];
        }
        if(i<n) prox[v[i]][i] = i+1;
    }
}
```

AhoCorasick.h

Description: Aho-Corasick automaton, used for multiple pattern matching. Initialize with AhoCorasick ac(patterns); the automaton start node will be at index 0. find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(−, word) finds all words (up to $N\sqrt{N}$ many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. For large alphabets, split each symbol into chunks, with sentinel bits for symbol boundaries.
Time: construction takes $\mathcal{O}(26N)$, where N = sum of length of patterns. find(x) is $\mathcal{O}(N)$, where N = length of x. findAll is $\mathcal{O}(NM)$.

f35677, 66 lines

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

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

SuffixAutomata.h

Description: Build suffix automaton
Time: $\mathcal{O}(n\sigma)$

0ba915, 44 lines

```
struct State {
  State *par, *go[26];
  int val;
  int id;
  State(int val = 0) :
    par(NULL), val(0), id(0){
    memset(go, 0, sizeof go);
  }
};
State *root, *last;
State statePool[N * 2], *cur;
void init() {
  cur = statePool;
  root = last = cur++;
}
void extend(int w){
  State *p = last;
  State *np = cur++;
  np->val = p->val + 1;
  while(p && p->go[w] == NULL){
    p->go[w] = np, p = p->par;
  }
  if(p == NULL){
    np->par = root;
  }
  else{
    State *q = p->go[w];
    if(p->val + 1 == q->val){
      np->par = q;
    }
```

```
    else{
      State *nq = cur++;
      memcpy(nq->go, q->go, sizeof(q->go));
      nq->val = p->val+1;
      nq->par = q->par;
      q->par = nq;
      np->par = nq;
      while(p && p->go[w] == q){
        p->go[w] = nq, p = p->par;
      }
    }
  }
  last = np;
}
```

PalindromicTree.h

Description: Build palindromic tree
Time: $\mathcal{O}(n)$

af331b, 46 lines

```
struct node {
    int next[26];
    int len;
    int sufflink;
};
int len;
char s[MAXN];
node tree[MAXN];
int num;            // node 1 - root with len -1, node 2 - root
    with len 0
int suff;           // max suffix palindrome
bool addLetter(int pos) {
    int cur = suff, curlen = 0;
    int let = s[pos] - 'a';
    while (true) {
        curlen = tree[cur].len;
        if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[
             pos])
            break;
        cur = tree[cur].sufflink;
    }
    if (tree[cur].next[let]) {
        suff = tree[cur].next[let];
        return false;
    }
    num++;
    suff = num;
    tree[num].len = tree[cur].len + 2;
    tree[cur].next[let] = num;
    if (tree[num].len == 1) {
        tree[num].sufflink = 2;
        return true;
    }
    while (true) {
        cur = tree[cur].sufflink;
        curlen = tree[cur].len;
        if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[
             pos]) {
            tree[num].sufflink = tree[cur].next[let];
            break;
        }
    }
    return true;
}
void initTree() {
    num = 2; suff = 2;
    tree[1].len = -1; tree[1].sufflink = 1;
    tree[2].len = 0; tree[2].sufflink = 1;
}
```

Various (9)

9.1 Misc. algorithms

FastKnapsack.h

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

b20ccc, 16 lines

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

9.2 Dynamic programming

KnuthDP.h

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

DivideAndConquerDP.h

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

d38d2b, 18 lines

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

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

FastMod.h

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

751a02, 8 lines

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

FastInput.h

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

7b3c70, 17 lines

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

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