Team Notebook

Pontificia Universidad Católica de Chile - Laranjas.clear()

April 10, 2024

Contents

1 data structures

1.1 dsu

```
struct Dsu {
    vector<int> p; Dsu(int N = 0) : p(N, -1) {}
    int get(int x) { return p[x] < 0 ? x : get(p[x]); }
    bool sameSet(int a, int b) { return get(a) == get(b); }
    int size(int x) { return -p[get(x)]; }
    vector<vector<int>> S;
    void unite(int x, int y) {
        if ((x = get(x)) == (y = get(y)))
            return S.push_back({-1});
        if (p[x] > p[y]) swap(x, y);
        S.push_back({x, y, p[x], p[y]});
        p[x] += p[y], p[y] = x;
    }
    void rollback() {
        auto a = S.back(); S.pop_back();
        if (a[0] != -1) p[a[0]] = a[2], p[a[1]] = a[3];
    }
};
```

1.2 fenwick-tree

```
int ft[MAXN+1]; // add dimension for multi-d
void upd(int i0, int v){ // add v to i0th element
    for(int i=i0+1;i<=MAXN;i+=i&-i)ft[i]+=v;//+ fors
}
int get(int i0){ // get sum of range [0,i0)
    int r=0; // add fors
    for(int i=i0;i;i-=i&-i)r+=ft[i];
    return r;
}
int get_sum(int i0,int i1){//sum of [i0,i1)
    return get(i1)-get(i0);
}
```

1.3 link-cut-tree

```
const int N_DEL = 0, N_VAL = 0; //delta, value
inline int mOp(int x, int y){return x+y;}//modify
inline int qOp(int lval, int rval){return lval + rval;}//
     query
inline int dOnSeg(int d, int len){return d==N_DEL ? N_DEL :
     d*len;}
//mostly generic
inline int joinD(int d1, int d2){
  if(d1==N_DEL)return d2;if(d2==N_DEL)return d1;return mOp(
      d1, d2);}
inline int joinVD(int v, int d){return d==N_DEL ? v : mOp(v,
      d);}
struct Node_t{
  int sz, nVal, tVal, d; bool rev;
  Node_t *c[2], *p;
  Node_t(int v) : sz(1), nVal(v), tVal(v), d(N_DEL), rev(0),
      p(0){
  c[0]=c[1]=0;
  }
  bool isRoot(){return !p || (p->c[0] != this && p->c[1] !=
      this);}
  void push(){
  if(rev){
    rev=0; swap(c[0], c[1]); fore(x,0,2)if(c[x])c[x]->rev^=1;
  }
  nVal=joinVD(nVal, d); tVal=joinVD(tVal, dOnSeg(d, sz));
  fore(x,0,2)if(c[x])c[x]->d=joinD(c[x]->d, d);
  d=N_DEL;
  }
  void upd();
};
typedef Node_t* Node;
int getSize(Node r){return r ? r->sz : 0;}
int getPV(Node r){
  return r ? joinVD(r->tVal, dOnSeg(r->d,r->sz)) : N_VAL;}
void Node_t::upd(){
  tVal = qOp(qOp(getPV(c[0]), joinVD(nVal, d)), getPV(c[1]))
      ;
  sz = 1 + getSize(c[0]) + getSize(c[1]);
}
void conn(Node c, Node p, int il){if(c)c->p=p;if(il>=0)p->c
     [!il]=c;}
void rotate(Node x){
  Node p = x->p, g = p->p;
  bool gCh=p->isRoot(), isl = x==p->c[0];
  conn(x->c[isl],p,isl); conn(p,x,!isl);
  conn(x,g,gCh?-1:(p==g->c[0])); p->upd();
}
void spa(Node x){//splay
  while(!x->isRoot()){
  Node p = x->p, g = p->p;
  if(!p->isRoot())g->push();
  p->push(); x->push();
  if(!p->isRoot())rotate((x==p->c[0])==(p==g->c[0])? p : x);
  rotate(x);
  }
  x->push(); x->upd();
}
Node exv(Node x){//expose
  Node last=0;
  for(Node y=x; y; y=y->p)spa(y),y->c[0]=last,y->upd(),last=
      y;
  spa(x);
  return last;
}
void mkR(Node x){exv(x);x->rev^=1;}//makeRoot
Node getR(Node x){exv(x);while(x->c[1])x=x->c[1];spa(x);
     return x;}
Node lca(Node x, Node y){exv(x); return exv(y);}
bool connected(Node x, Node y){exv(x);exv(y); return x==y?1:
     x->p!=0;}
void link(Node x, Node y){mkR(x); x->p=y;}
void cut(Node x, Node y){mkR(x); exv(y); y->c[1]->p=0; y->c
     [1]=0;}
Node father(Node x){
  exv(x); Node r=x->c[1];
  if(!r)return 0;
  while(r->c[0])r=r->c[0];
  return r;
}
void cut(Node x){ // cuts x from father keeping tree root
  exv(father(x));x->p=0;}
int query(Node x, Node y){mkR(x); exv(y); return getPV(y);}
void modify(Node x, Node y, int d){mkR(x);exv(y);y->d=joinD(
     y->d,d);}
Node lift_rec(Node x, int t){
  if(!x)return 0;
  if(t==getSize(x->c[0])){spa(x);return x;}
  if(t<getSize(x->c[0]))return lift_rec(x->c[0],t);
  return lift_rec(x->c[1],t-getSize(x->c[0])-1);
}
Node lift(Node x, int t){ // t-th ancestor of x (lift(x,1)
     is x's father)
  exv(x);return lift_rec(x,t);}
int depth(Node x){ // distance from x to its tree root
  exv(x);return getSize(x)-1;}
```

1.4 persistent-segment-tree-lazy

```
template <class T>
struct Node {
    T x, lz;
    int l = -1, r = -1;
};
```

```
template <class T>
struct Pstl {
    int N;
    vector<Node<T>> a;
    vector<int> head;

    T qneut() { return 0; }
    T merge(T l, T r) { return l + r; }
    T uneut() { return 0; }
    T accum(T u, T x) { return u + x; }
    T apply(T x, T lz, int l, int r) { return x + (r - l) *
         lz; }

    int build(int vl, int vr) {
        if (vr - vl == 1) a.push_back({qneut(), uneut()}); //
             node construction
        else {
            int vm = (vl + vr) / 2, l = build(vl, vm), r =
                 build(vm, vr);
            a.push_back({merge(a[l].x, a[r].x), uneut(), l, r
                 }); // query merge
        }
        return a.size() - 1;
    }

    T query(int l, int r, int v, int vl, int vr, T acc) {
        if (l >= vr || r <= vl) return qneut();
             // query neutral
        if (l <= vl && r >= vr) return apply(a[v].x, acc, vl,
             vr); // update op
        acc = accum(acc, a[v].lz);
             // update merge
        int vm = (vl + vr) / 2;
        return merge(query(l, r, a[v].l, vl, vm, acc), query(
             l, r, a[v].r, vm, vr, acc)); // query merge
    }

    int update(int l, int r, T x, int v, int vl, int vr) {
        if (l >= vr || r <= vl || r <= l) return v;
        a.push_back(a[v]);
        v = a.size() - 1;
        if (l <= vl && r >= vr) {
            a[v].x = apply(a[v].x, x, vl, vr); // update op
            a[v].lz = accum(a[v].lz, x);    // update merge
        } else {
            int vm = (vl + vr) / 2;
            a[v].l = update(l, r, x, a[v].l, vl, vm);
            a[v].r = update(l, r, x, a[v].r, vm, vr);
            a[v].x = merge(a[a[v].l].x, a[a[v].r].x); //
                 query merge
        }
        return v;
    }

    Pstl() {}
    Pstl(int N) : N(N) { head.push_back(build(0, N)); }

    T query(int t, int l, int r) {
        return query(l, r, head[t], 0, N, uneut()); // update
             neutral
    }
    int update(int t, int l, int r, T x) {
        return head.push_back(update(l, r, x, head[t], 0, N))
             , head.size() - 1;
    }
};
```

1.5 persistent-segment-tree

```
// usage:
// Pst<Node<ll>> pst;
// pst = {N};
// int newtime = pst.update(time, index, value);
// Node<ll> result = pst.query(newtime, left, right);

template <class T>
struct Node {
    T x;
    int l = -1, r = -1;

    Node() : x(0) {}
    Node(T x) : x(x) {}
    Node(Node a, Node b, int l = -1, int r = -1) : x(a.x + b.
         x), l(l), r(r) {}
};

template <class U>
struct Pst {
    int N;
    vector<U> a;
    vector<int> head;

    int build(int vl, int vr) {
        if (vr - vl == 1) a.push_back(U());
        else {
            int vm = (vl + vr) / 2, l = build(vl, vm),
                r = build(vm, vr);
            a.push_back(U(a[l], a[r], l, r));
        }
        return a.size() - 1;
    }

    U query(int l, int r, int v, int vl, int vr) {
        if (l >= vr || r <= vl) return U();
        if (l <= vl && r >= vr) return a[v];
        int vm = (vl + vr) / 2;
        return U(query(l, r, a[v].l, vl, vm),
                 query(l, r, a[v].r, vm, vr));
    }

    int update(int i, U x, int v, int vl, int vr) {
        a.push_back(a[v]);
        v = a.size() - 1;
        if (vr - vl == 1) a[v] = x;
        else {
            int vm = (vl + vr) / 2;
            if (i < vm) a[v].l = update(i, x, a[v].l, vl, vm)
                 ;
            else a[v].r = update(i, x, a[v].r, vm, vr);
            a[v] = U(a[a[v].l], a[a[v].r], a[v].l, a[v].r);
        }
        return v;
    }

    Pst() {}
    Pst(int N) : N(N) { head.push_back(build(0, N)); }

    U query(int t, int l, int r) {
        return query(l, r, head[t], 0, N);
    }
    int update(int t, int i, U x) {
        return head.push_back(update(i, x, head[t], 0, N)),
             head.size() - 1;
    }
};
```

1.6 rmq-lineal

```
typedef int tf; // O(n) construction, O(1) query
struct rmq{
    int n; tf INF=1e9;//change sign of INF for MAX
    vector<unsigned int> mk; vector<tf> bk,v;
    rmq(){}
    tf op(tf a, tf b){return min(a,b);}//change for maximum
    int f(int x){return x>>5;}
    rmq(vector<tf> &vv):n(SZ(vv)),mk(n),bk(n,INF),v(vv){
        unsigned int lst=0;
        for(int i=0;i<SZ(v);i++,lst<<=1){
```

```
        bk[f(i)]=op(bk[f(i)],v[i]);
        while(lst&&v[i-__builtin_ctz(lst)]>v[i]) lst^=lst
            &-lst;  //MIN
        //while(lst&&v[i-__builtin_ctz(lst)]<v[i]) lst^=
            lst&-lst; //MAX
        mk[i]=++lst;
      }
      for(int k=1,top=f(n);(1<<k)<=top;k++)fore(i,0,top)if(
          i+(1<<k)<=top)
        bk[top*k+i]=op(bk[top*(k-1)+i], bk[top*(k-1)+i
            +(1<<k-1)]);
    }
    tf get(int st, int en){
      return v[en-31+__builtin_clz(mk[en]&((1ll<<en-st+1)
          -1))];
    }
    tf query(int s, int e){ //[s,e]
      int b1=f(s),b2=f(e),top=f(n);
      if(b1==b2) return get(s,e);
      tf ans=op(get(s,(b1+1)*32-1), get(b2*32,e)); s=(b1+1)
          *32; e=b2*32-1;
      if(s<=e){
        int k=31-__builtin_clz(f(e-s+1));
        ans=op(ans,op(bk[top*k+f(s)],bk[top*k+f(e)-(1<<k)
            +1]));
      }
      return ans;
    }
};
```

1.7 segment-tree-2d

```
// #define MAXN 1024 #define op(a,b) (a+b) #define NEUT 0
int n,m; int a[MAXN][MAXN],st[2*MAXN][2*MAXN];
void build(){
    repx(i, 0, n) repx(j, 0, m) st[i+n][j+m] = a[i][j];
    repx(i, 0, n) for(int j = m-1; j; --j)
        st[i+n][j] = op(st[i+n][j<<1], st[i+n][j<<1|1]);
    for(int i = n-1; i; --i) repx(j, 0, 2*m)
        st[i][j] = op(st[i<<1][j], st[i<<1|1][j]);
}
void upd(int x, int y, int v){
    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){
    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;
        repx(k,0,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;
}
```

1.8 segment-tree-beats

```
struct Node {
    ll s, mx1, mx2, mxc, mn1, mn2, mnc, lz = 0;
    Node() : s(0), mx1(LLONG_MIN), mx2(LLONG_MIN), mxc(0),
        mn1(LLONG_MAX), mn2(LLONG_MAX), mnc(0) {}
    Node(ll x) : s(x), mx1(x), mx2(LLONG_MIN), mxc(1), mn1(x)
        , mn2(LLONG_MAX), mnc(1) {}
    Node(const Node &a, const Node &b) {
        // add
        s = a.s + b.s;
        // min
        if (a.mx1 > b.mx1) mx1 = a.mx1, mxc = a.mxc, mx2 =
            max(b.mx1, a.mx2);
        if (a.mx1 < b.mx1) mx1 = b.mx1, mxc = b.mxc, mx2 =
            max(a.mx1, b.mx2);
        if (a.mx1 == b.mx1) mx1 = a.mx1, mxc = a.mxc + b.mxc,
            mx2 = max(a.mx2, b.mx2);
        // max
        if (a.mn1 < b.mn1) mn1 = a.mn1, mnc = a.mnc, mn2 =
            min(b.mn1, a.mn2);
        if (a.mn1 > b.mn1) mn1 = b.mn1, mnc = b.mnc, mn2 =
            min(a.mn1, b.mn2);
        if (a.mn1 == b.mn1) mn1 = a.mn1, mnc = a.mnc + b.mnc,
            mn2 = min(a.mn2, b.mn2);
    }
};

// 0 - indexed / inclusive - inclusive
template <class node>
struct STB {
    vector<node> st; int n;

    void build(int u, int i, int j, vector<node> &arr) {
        if (i == j) {
            st[u] = arr[i];
            return;
        }
        int m = (i + j) / 2, l = u * 2 + 1, r = u * 2 + 2;
        build(l, i, m, arr), build(r, m + 1, j, arr);
        st[u] = node(st[l], st[r]);
    }
    void push_add(int u, int i, int j, ll v) {
        st[u].s += (j - i + 1) * v;
        st[u].mx1 += v, st[u].mn1 += v, st[u].lz += v;
        if (st[u].mx2 != LLONG_MIN) st[u].mx2 += v;
        if (st[u].mn2 != LLONG_MAX) st[u].mn2 += v;
    }
    void push_max(int u, ll v, bool l) { // for min op
        if (v >= st[u].mx1) return;
        st[u].s -= st[u].mx1 * st[u].mxc;
        st[u].mx1 = v;
        st[u].s += st[u].mx1 * st[u].mxc;
        if (l) st[u].mn1 = st[u].mx1;
        else if (v <= st[u].mn1) st[u].mn1 = v;
        else if (v < st[u].mn2) st[u].mn2 = v;
    }
    void push_min(int u, ll v, bool l) { // for max op
        if (v <= st[u].mn1) return;
        st[u].s -= st[u].mn1 * st[u].mnc;
        st[u].mn1 = v;
        st[u].s += st[u].mn1 * st[u].mnc;
        if (l) st[u].mx1 = st[u].mn1;
        else if (v >= st[u].mx1) st[u].mx1 = v;
        else if (v > st[u].mx2) st[u].mx2 = v;
    }
    void push(int u, int i, int j) {
        if (i == j) return;
        // add
        int m = (i + j) / 2, l = u * 2 + 1, r = u * 2 + 2;
        push_add(l, i, m, st[u].lz);
        push_add(r, m + 1, j, st[u].lz);
        st[u].lz = 0;
        // min
        push_max(l, st[u].mx1, i == m);
        push_max(r, st[u].mx1, m + 1 == j);
        // max
        push_min(l, st[u].mn1, i == m);
        push_min(r, st[u].mn1, m + 1 == r);
    }
    node query(int a, int b, int u, int i, int j) {
        if (b < i || j < a) return node();
        if (a <= i && j <= b) return st[u];
        push(u, i, j);
        int m = (i + j) / 2, l = u * 2 + 1, r = u * 2 + 2;
```

```
    return node(query(a, b, l, i, m), query(a, b, r, m +
        1, j));
}
void update_add(int a, int b, ll v, int u, int i, int j)
    {
    if (b < i || j < a) return;
    if (a <= i && j <= b) {
        push_add(u, i, j, v);
        return;
    }
    push(u, i, j);
    int m = (i + j) / 2, l = u * 2 + 1, r = u * 2 + 2;
    update_add(a, b, v, l, i, m);
    update_add(a, b, v, r, m + 1, j);
    st[u] = node(st[l], st[r]);
}
void update_min(int a, int b, ll v, int u, int i, int j)
    {
    if (b < i || j < a || v >= st[u].mx1) return;
    if (a <= i && j <= b && v > st[u].mx2) {
        push_max(u, v, i == j);
        return;
    }
    push(u, i, j);
    int m = (i + j) / 2, l = u * 2 + 1, r = u * 2 + 2;
    update_min(a, b, v, l, i, m);
    update_min(a, b, v, r, m + 1, j);
    st[u] = node(st[l], st[r]);
}
void update_max(int a, int b, ll v, int u, int i, int j)
    {
    if (b < i || j < a || v <= st[u].mn1) return;
    if (a <= i && j <= b && v < st[u].mn2) {
        push_min(u, v, i == j);
        return;
    }
    push(u, i, j);
    int m = (i + j) / 2, l = u * 2 + 1, r = u * 2 + 2;
    update_max(a, b, v, l, i, m);
    update_max(a, b, v, r, m + 1, j);
    st[u] = node(st[l], st[r]);
}

STB(vector<node> &v, int N) : n(N), st(N * 4 + 5) { build
    (0, 0, n - 1, v); }
node query(int a, int b) { return query(a, b, 0, 0, n -
    1); }
void update_add(int a, int b, ll v) { update_add(a, b, v,
     0, 0, n - 1); }
void update_min(int a, int b, ll v) { update_min(a, b, v,
     0, 0, n - 1); }
void update_max(int a, int b, ll v) { update_max(a, b, v,
     0, 0, n - 1); }
};
```

1.9 segment-tree-lazy

```
template <class T>
struct Stl {
    int n; vector<T> a, b;
    Stl(int n = 0) : n(n), a(4 * n, qneut()),
        b(4 * n, uneut()) {}

    T qneut() { return -2e9; }
    T uneut() { return 0; }
    T merge(T x, T y) { return max(x, y); }
    void upd(int v, T x, int l, int r)
        { a[v] += x, b[v] += x; }

    void push(int v, int vl, int vm, int vr) {
        upd(2 * v, b[v], vl, vm);
        upd(2 * v + 1, b[v], vm, vr);
        b[v] = uneut();
    }

    T query(int l, int r, int v=1, int vl=0, int vr=1e9) {
        vr = min(vr, n);
        if (l <= vl && r >= vr) return a[v];
        if (l >= vr || r <= vl) return qneut();
        int vm = (vl + vr) / 2;
        push(v, vl, vm, vr);
        return merge(query(l, r, 2 * v, vl, vm),
            query(l, r, 2 * v + 1, vm, vr));
    }

    void update(int l, int r, T x, int v = 1, int vl = 0,
            int vr = 1e9) {
        vr = min(vr, n);
        if (l >= vr || r <= vl || r <= l) return;
        if (l <= vl && r >= vr) upd(v, x, vl, vr);
        else {
            int vm = (vl + vr) / 2;
            push(v, vl, vm, vr);
            update(l, r, x, 2 * v, vl, vm);
            update(l, r, x, 2 * v + 1, vm, vr);
            a[v] = merge(a[2 * v], a[2 * v + 1]);
        }
    }
};
```

1.10 segment-tree

```
struct St {
    ll neut() { return 0; }
    ll merge(ll x, ll y) { return x + y; }

    int n; vector<ll> a;
    St(int n = 0) : n(n), a(2 * n, neut()) {}

    ll query(int l, int r) {
        ll x = neut(), y = neut();
        for (l += n, r += n; l < r; l /= 2, r /= 2) {
            if (l & 1) x = merge(x, a[l++]);
            if (r & 1) y = merge(a[--r], y);
        }
        return merge(x, y);
    }

    void update(int i, ll x) {
        for (a[i += n] = x; i /= 2;)
            a[i] = merge(a[2 * i], a[2 * i + 1]);
    }
};
```

1.11 sparse-table

```
template <class T>
struct Sparse {
    T op(T a, T b) { return max(a, b); }

    vector<vector<T>> st;
    Sparse() {}
    Sparse(vector<T> a) : st{a} {
        int N = st[0].size();
        int npot = N <= 1 ? 1 : 32 - __builtin_clz(N);
        st.resize(npot);
        repx(i, 1, npot) rep(j, N + 1 - (1 << i))
        st[i].push_back(
            op(st[i - 1][j], st[i - 1][j + (1 << (i - 1))])
        ); // query op
    }

    T query(int l, int r) { // range must be nonempty!
        int i = 31 - __builtin_clz(r - l);
        return op(st[i][l], st[i][r - (1 << i)]); // queryop
```

```
    }
};
```

1.12 treap-implicit

```
mt19937 gen(chrono::high_resolution_clock::now().
     time_since_epoch().count());
// 101 Implicit Treap //
struct Node {
    int p, sz = 0, v, acc, l = -1, r = -1;
    Node() : v(0), acc(0) {}
    Node(int x) : p(gen()), sz(1), v(x), acc(x) {}
    void recalc(const Node &a, const Node &b) {
        sz = a.sz + b.sz + 1; acc = v + a.acc + b.acc;
    }
};
template <class node>
struct Treap {
    vector<node> t;
    int n, r = -1;
    node get(int u) { return u != -1 ? t[u] : node(); }
    void recalc(int u){t[u].recalc(get(t[u].l),get(t[u].r));}
    int merge(int l, int r) {
        if (min(l, r) == -1) return l != -1 ? l : r;
        int ans = (t[l].p < t[r].p) ? l : r;
        if (ans == l) t[l].r = merge(t[l].r, r), recalc(l);
        if (ans == r) t[r].l = merge(l, t[r].l), recalc(r);
        return ans;
    }
    pii split(int u, int id) {
        if (u == -1) return {-1, -1};
        int szl = get(t[u].l).sz;
        if (szl >= id) {
            pii ans = split(t[u].l, id);
            t[u].l = ans.ss;
            recalc(u);
            return {ans.ff, u};
        }
        pii ans = split(t[u].r, id - szl - 1);
        t[u].r = ans.ff;
        recalc(u);
        return {u, ans.ss};
    }
    Treap(vi &v) : n(sz(v)) {
        for (int i=0; i<n; i++) t.eb(v[i]), r = merge(r, i);
    }
};
// Complete Implicit Treap with Lazy propagation //
struct Node {
    int p, sz = 0, v, acc, l = -1, r = -1, par = -1, lzv=0;
    bool lz = false, f = false;
    Node() : v(0), acc(0) {}
    Node(int x) : p(gen()), sz(1), v(x), acc(x) {}
    void recalc(const Node &a, const Node &b) {
        sz = a.sz + b.sz + 1; acc = v + a.acc + b.acc;
    }
    void upd_lazy(int x) { lz = 1, lzv += x; }
    void lazy() {v+=lzv, acc += sz * lzv, lz = 0, lzv = 0; }
    void flip() { swap(l, r), f = 0; }
};
template <class node> struct Treap {
    vector<node> t;
    int n, r = -1;
    node get(int u) { return u != -1 ? t[u] : node(); }
    void recalc(int u) {
        int l = t[u].l, r = t[u].r;
        push(l), push(r), flip(l), flip(r);
        t[u].recalc(get(l), get(r));
    }
    void push(int u) {
        if (u == -1 || !t[u].lz) return;
        int l = t[u].l, r = t[u].r;
        if (l != -1) t[l].upd_lazy(t[u].lzv);
        if (r != -1) t[r].upd_lazy(t[u].lzv);
        t[u].lazy();
    }
    void flip(int u) {
        if (u == -1 || !t[u].f) return;
        int l = t[u].l, r = t[u].r;
        if (l != -1) t[l].f ^= 1;
        if (r != -1) t[r].f ^= 1;
        t[u].flip();
    }
    int merge(int l, int r) { // (*) = only if parent needed
        if (min(l, r) == -1) return l != -1 ? l : r;
        push(l), push(r), flip(l), flip(r);
        int ans = (t[l].p < t[r].p) ? l : r;
        if (ans == l) t[l].r = merge(t[l].r, r), recalc(l);
        if (ans == r) t[r].l = merge(l, t[r].l), recalc(r);
        if (t[ans].l != -1) t[t[ans].l].par = ans; // (*)
        if (t[ans].r != -1) t[t[ans].r].par = ans; // (*)
        return ans;
    }
    pii split(int u, int id) {// (*) = only if parent needed
        if (u == -1) return {-1, -1};
        push(u);
        flip(u);
        int szl = get(t[u].l).sz;
        if (szl >= id) {
            pii ans = split(t[u].l, id);
            if (ans.ss != -1) t[ans.ss].par = u; // (*)
            if (ans.ff != -1) t[ans.ff].par = -1; // (*)
            t[u].l = ans.ss;
            recalc(u);
            return {ans.ff, u};
        }
        pii ans = split(t[u].r, id - szl - 1);
        if (ans.ff != -1) t[ans.ff].par = u; // (*)
        if (ans.ss != -1) t[ans.ss].par = -1; // (*)
        t[u].r = ans.ff;
        recalc(u);
        return {u, ans.ss};
    }
    int update(int u, int l, int r, int v) {
        pii a = split(u, l), b = split(a.ss, r - l + 1);
        t[b.ff].upd_lazy(v);
        return merge(a.ff, merge(b.ff, b.ss));
    }
    void print(int u) {
        if (u==-1) return; push(u), flip(u); print(t[u].l);
        cout << t[u].v << ' '; print(t[u].r);
    }
    Treap(vi &v) : n(sz(v)) {
        for (int i=0; i<n; i++) t.eb(v[i]), r = merge(r, i);
    }
};
```

1.13 treap

```
typedef struct item *pitem;
struct item {
    int pr,key,cnt; pitem l,r;
    item(int key):key(key),pr(rand()),cnt(1),l(0),r(0) {}
};
int cnt(pitem t){return t?t->cnt:0;}
void upd_cnt(pitem t){if(t)t->cnt=cnt(t->l)+cnt(t->r)+1;}
void split(pitem t, int key, pitem& l, pitem& r){ // l: <=
     key, r: > key
    if(!t)l=r=0;
    else if(key<t->key)split(t->l,key,l,t->l),r=t;
    else split(t->r,key,t->r,r),l=t;
    upd_cnt(t);
}
void insert(pitem& t, pitem it){
    if(!t)t=it;
    else if(it->pr>t->pr)split(t,it->key,it->l,it->r),t=it;
    else insert(it->key<t->key?t->l:t->r,it);
    upd_cnt(t);
```

```
}
void merge(pitem& t, pitem l, pitem r){
    if(!l||!r)t=l?l:r;
    else if(l->pr>r->pr)merge(l->r,l->r,r),t=l;
    else merge(r->l,l,r->l),t=r;
    upd_cnt(t);
}
void erase(pitem& t, int key){
    if(t->key==key)merge(t,t->l,t->r);
    else erase(key<t->key?t->l:t->r,key);
    upd_cnt(t);
}
```

2 dp

2.1 convex-hull-trick

```
struct Line {
    mutable ll a, b, c;

    bool operator<(Line r) const { return a < r.a; }
    bool operator<(ll x) const { return c < x; }
};
// dynamically insert `a*x + b` lines and query for maximum
// at any x all operations have complexity O(log N)
struct LineContainer : multiset<Line, less<>> {
    ll div(ll a, ll b) {
        return a / b - ((a ^ b) < 0 && a % b);
    }

    bool isect(iterator x, iterator y) {
        if (y == end()) return x->c = INF, 0;
        if (x->a == y->a) x->c = x->b > y->b ? INF : -INF;
        else x->c = div(y->b - x->b, x->a - y->a);
        return x->c >= y->c;
    }

    void add(ll a, ll b) {
        // a *= -1, b *= -1 // for min
        auto z = insert({a, b, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase
            (y));
        while ((y = x) != begin() && (--x)->c >= y->c) isect(
            x, erase(y));
    }

    ll query(ll x) {
        if (empty()) return -INF; // INF for min
        auto l = *lower_bound(x);
        return l.a * x + l.b;
        // return -l.a * x - l.b; // for min
    }
};
```

2.2 divide-and-conquer

```
// for every index i assign an optimal index j, such that
// cost(i, j) is minimal for every i. the property that if
// i2 >= i1 then j2 >= j1 is exploited (monotonic condition)
// calculate optimal index for all indices in range [l, r)
// knowing that the optimal index for every index in this
// range is within [optl, optr).
// time: O(N log N)
void calc(vector<int> &opt, int l, int r,int optl,int optr){
    if (l == r) return;
    int i = (l + r) / 2;
    ll optc = INF;
    int optj;
    repx(j, optl, optr) {
        ll c = i + j; // cost(i, j)
        if (c < optc) optc = c, optj = j;
    }
    opt[i] = optj;
    calc(opt, l, i, optl, optj + 1);
    calc(opt, i + 1, r, optj, optr);
}
```

3 geo2d

3.1 circle

```
struct C {
    P o; T r;

    // circle-line intersection, assuming it exists
    // points are sorted along the direction of the line
    pair<P, P> line_inter(L l) const {
        P c = l.closest_to(o); T c2 = (c - o).magsq();
        P e = l.d * sqrt(max(r*r - c2, T()) / l.d.magsq());
        return {c - e, c + e};
    }

    // check the type of line-circle collision
    // <0: 2 inters, =0: 1 inter, >0: 0 inters
    T line_collide(L l) const {
        T c2 = (l.closest_to(o) - o).magsq();
        return c2 - r * r;
    }

    // calculates the two intersections between two circles
    // the circles must intersect in one or two points!
    pair<P, P> inter(C h) const {
        P d = h.o - o;
        T c = (r * r - h.r * h.r) / d.magsq();
        return h.line_inter({(1 + c) / 2 * d, d.rot()});
    }

    // check if the given circles intersect
    bool collide(C h) const {
        return (h.o - o).magsq() <= (h.r + r) * (h.r + r);
    }

    // get one of the two tangents that go through the point
    // the point must not be inside the circle
    // a = -1: cw (relative to the circle) tangent
    // a = 1: ccw (relative to the circle) tangent
    P point_tangent(P p, T a) const {
        T c = r * r / p.magsq();
        return o + c*(p-o) - a*sqrt(c*(1-c))*(p-o).rot();
    }

    // get one of the 4 tangents between the two circles
    // a = 1: exterior tangents
    // a = -1: interior tangents (requires no area overlap)
    // b = 1: ccw tangent
    // b = -1: cw tangent
    // the line origin is on this circumference, and the
    // direction is a unit vector towards the other circle
    L tangent(C c, T a, T b) const {
        T dr = a * r - c.r;
        P d = c.o - o;
        P n = (d*dr+b*d.rot()*sqrt(d.magsq()-dr*dr)).unit();
        return {o + n * r, -b * n.rot()};
    }

    // circumcircle of a **non-degenerate** triangle
    static C thru_points(P a, P b, P c) {
        b = b - a, c = c - a;
        P p = (b*c.magsq() - c*b.magsq()).rot() / (b%c*2);
        return {a + p, p.mag()};
    }

    // find the two circles that go through the given point,
```

```
    // are tangent to the given line and have radius 'r'
    // the point-line distance must be at most 'r'!
    // the circles are sorted in the direction of the line
    static pair<C, C> thru_point_line_r(P a, L t, T r) {
        P d = t.d.rot().unit();
        if (d * (a - t.o) < 0) d = -d;
        auto p = C(a, r).line_inter({t.o + d * r, t.d});
        return {{p.first, r}, {p.second, r}};
    }

    // find the two circles that go through the given points
    // and have radius 'r'
    // circles sorted by angle from the first point
    // the points must be at most at distance 'r'!
    static pair<C, C> thru_points_r(P a, P b, T r) {
        auto p = C(a, r).line_inter({(a+b)/2, (b-a).rot()});
        return {{p.first, r}, {p.second, r}};
    }
};
```

3.2 closest-points

```
// sort by x
ll closest(vector<ii> &p) {
    int n = SZ(p);
    set<ii> s;
    ll best = 1e18;
    int j = 0;
    fore(i, 0, n) {
        ll d = ceil(sqrt(best));
        while(p[i].fst - p[j].fst >= best)
            s.erase({p[j].snd, p[j].fst}), j++;
        auto it1=s.lower_bound({p[i].snd-d,p[i].fst});
        auto it2=s.upper_bound({p[i].snd+d,p[i].fst});
        for(auto it = it1; it != it2; ++it) {
            ll dx = p[i].fst - it->snd;
            ll dy = p[i].snd - it->fst;
            best = min(best, dx * dx + dy * dy);
        }
        s.insert({p[i].snd, p[i].fst});
    }
    return best;
}
```

3.3 convex-hull

```
// ccw order, excludes collinear points by default
vector<P> chull(vector<P> p) {
    if (p.size() < 3) return p;
    vector<P> r; int m, k = 0;
    sort(p.begin(), p.end(), [](P a, P b) {
        return a.x != b.x ? a.x < b.x : a.y < b.y; });
    for (P q : p) { // lower hull
        while (k >= 2 && r[k - 1].left(r[k - 2], q) >= 0)
            r.pop_back(), k--; // >= to > to add collinears
        r.push_back(q), k++;
    }
    if (k == (int)p.size()) return r;
    r.pop_back(), k--, m = k;
    for (int i = p.size() - 1; i >= 0; --i) { // upper hull
        while (k >= m+2 && r[k-1].left(r[k-2], p[i]) >= 0)
            r.pop_back(), k--; // >= to > to add collinears
        r.push_back(p[i]), k++;
    }
    r.pop_back(); return r;
}
```

3.4 delaunay

```
typedef __int128_t lll; // if on a 64-bit platform

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

T cross(P a, P b, P c) { return (b - a) % (c - a); }

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
    lll p2 = p.magsq(), A = a.magsq() - p2,
        B = b.magsq() - p2, C = c.magsq() - p2;
    return cross(p, a, b) * C + cross(p, b, c) * A + cross(p,
        c, a) * B > 0;
}

Q *makeEdge(Q *&H, P orig, P dest) {
    Q *r = H ? H : new Q{new Q{new Q{new Q{0}}}};
    H = r->o; r->r()->r() = r;
    repx(i, 0, 4) r = r->rot, r->p = {INF, INF},
        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 *&H, Q *a, Q *b) {
    Q *q = makeEdge(H, a->F(), b->p);
    splice(q, a->next()); splice(q->r(), b); return q;
}

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

#define J(e) e->F(), e->p
#define valid(e) (cross(e->F(), J(base)) > 0)
    Q *A, *B, *ra, *rb; int half = s.size() / 2;
    tie(ra, A) = rec(H, {s.begin(), s.end() - half});
    tie(B, rb) = rec(H, {s.begin() + s.size() - half, s.end()
        });
    while ((cross(B->p, J(A)) < 0 && (A = A->next())) ||
           (cross(A->p, J(B)) > 0 && (B = B->r()->o)));
    Q *base = connect(H, 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(), J(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(J(RC), J(LC))))
            base = connect(H, RC, base->r());
        else base = connect(H, base->r(), LC->r());
    }
    return {ra, rb};
#undef J
```

```
#undef valid
#undef DEL
}

// there must be no duplicate points
// returns no triangles in the case of all collinear points
// produces counter-clockwise triangles ordered in triples
// maximizes the minimum angle across all triangulations
// the euclidean mst is a subset of these edges
// O(N log N)
vector<P> triangulate(vector<P> pts) {
    sort(pts.begin(), pts.end(), [](P a, P b) {
        return make_pair(a.x, a.y) < make_pair(b.x, b.y);
    });
    assert(unique(pts.begin(), pts.end()) == pts.end());
    if (pts.size() < 2) return {};
    Q *H = 0; Q *e = rec(H, pts).first;
    vector<Q *> q = {e}; int qi = 0;
    while (cross(e->o->F(), 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 < (int)q.size()) if (!(e = q[qi++])->mark) ADD;
    return pts;
#undef ADD
}
```

3.5 halfplane-intersect

```
// obtain the convex polygon that results from intersecting
    the given list
// of halfplanes, represented as lines that allow their left
    side
// assumes the halfplane intersection is bounded
vector<P> halfplane_intersect(vector<L> &H) {
    L bb(P(-INF, -INF), P(INF, 0));
    rep(k, 4) H.push_back(bb), bb.o = bb.o.rot(), bb.d = bb.d
        .rot();

    sort(begin(H), end(H), [](L a, L b) { return a.d.angcmp(b
        .d) < 0; });
    deque<L> q; int n = 0;
    rep(i, H.size()) {
        while (n >= 2 && H[i].side(q[n - 1].intersection(q[n
            - 2])) > 0)
            q.pop_back(), n--;
        while (n >= 2 && H[i].side(q[0].intersection(q[1])) >
            0)
            q.pop_front(), n--;
        if (n > 0 && H[i].parallel(q[n - 1])) {
            if (H[i].d * q[n - 1].d < 0) return {};
            if (H[i].side(q[n - 1].o) > 0) q.pop_back(), n--;
            else continue;
        }
        q.push_back(H[i]), n++;
    }

    while (n >= 3 && q[0].side(q[n - 1].intersection(q[n -
        2])) > 0)
        q.pop_back(), n--;
    while (n >= 3 && q[n - 1].side(q[0].intersection(q[1])) >
        0)
        q.pop_front(), n--;
    if (n < 3) return {};

    vector<P> ps(n);
    rep(i, n) ps[i] = q[i].intersection(q[(i + 1) % n]);
    return ps;
}
```

3.6 line

```
// a segment or an infinite line
// does not handle point segments correctly!
struct L {
    P o, d;

    static L from_eq(P ab, T c) {
        return L{ab.rot(), ab * -c / ab.magsq()};
    }
    pair<P, T> line_eq() { return {-d.rot(), d.rot() * o}; }

    // on which side of the line is the point
    // negative: left, positive: right
    T side(P r) const { return (r - o) % d; }

    // returns the intersection coefficient
    // in the range [0, d % r.d]
    // if d % r.d is zero, the lines are parallel
    T inter(L r) const { return (r.o - o) % r.d; }

    // get the single intersection point
    // lines must not be parallel
    P intersection(L r) const {return o+d*inter(r)/(d%r.d);}

    // check if lines are parallel
    bool parallel(L r) const { return abs(d % r.d) <= EPS; }

    // check if segments intersect
    bool seg_collide(L r) const {
        T z = d % r.d;
        if (abs(z) <= EPS) {
            if (abs(side(r.o)) > EPS) return false;
            T s = (r.o - o) * d, e = s + r.d * d;
            if (s > e) swap(s, e);
            return s <= d * d + EPS && e >= -EPS;
        }
        T s = inter(r), t = -r.inter(*this);
        if (z < 0) s = -s, t = -t, z = -z;
        return s>=-EPS && s<=z+EPS && t>=-EPS && t<=z+EPS;
    }

    // full segment intersection
    // makes a point segment if the intersection is a point
    // however it does not handle point segments as input!
    bool seg_inter(L r, L *out) const {
        T z = d % r.d;
        if (abs(z) <= EPS) {
            if (abs(side(r.o)) > EPS) return false;
            if (r.d * d < 0) r = {r.o + r.d, -r.d};
            P s = o * d < r.o * d ? r.o : o;
            P e = (o+d)*d < (r.o+r.d)*d ? o+d : r.o+r.d;
            if (s * d > e * d) return false;
            return *out = {s, e - s}, true;
        }
        T s = inter(r), t = -r.inter(*this);
        if (z < 0) s = -s, t = -t, z = -z;
        if (s>=-EPS && s<=z+EPS && t>=-EPS && t<=z+EPS)
            return *out = {o + d * s / z, {0, 0}}, true;
        return false;
    }

    // check if the given point is on the segment
    bool point_on_seg(P r) const {
        if (abs(side(r)) > EPS) return false;
        if ((r - o) * d < -EPS) return false;
        if ((r - o - d) * d > EPS) return false;
        return true;
    }

    // point in this line that is closest to a given point
```

```
    P closest_to(P r) const {
        P dr = d.rot(); return r + dr*((o-r)*dr)/d.magsq();
    }
};
```

3.7 minkowski

```
void reorder_polygon(vector<P> &ps) {
    int pos = 0;
    repx(i, 1, (int)ps.size()) {
        if (ps[i].y < ps[pos].y || (ps[i].y == ps[pos].y &&
             ps[i].x < ps[pos].x))
            pos = i;
    }
    rotate(ps.begin(), ps.begin() + pos, ps.end());
}
vector<P> minkowski(vector<P> ps, vector<P> qs) {
    // the first vertex must be the lowest
    reorder_polygon(ps); reorder_polygon(qs);
    ps.push_back(ps[0]); ps.push_back(ps[1]);
    qs.push_back(qs[0]); qs.push_back(qs[1]);
    vector<P> result; int i = 0, j = 0;
    while (i < ps.size() - 2 || j < qs.size() - 2) {
        result.push_back(ps[i] + qs[j]);
        auto z = (ps[i + 1] - ps[i]) % (qs[j + 1] - qs[j]);
        if (z >= 0 && i < ps.size() - 2) ++i;
        if (z <= 0 && j < qs.size() - 2) ++j;
    }
    return result;
}
```

3.8 point

```
struct P {
    T x, y;
    P(T x, T y) : x(x), y(y) {}
    P() : P(0, 0) {}

    friend ostream &operator<<(ostream &s, const P &r) {
        return s << r.x << " " << r.y;
    }
    friend istream &operator>>(istream &s, P &r) { return s
         >> r.x >> r.y; }

    P operator+(P r) const { return {x + r.x, y + r.y}; }
    P operator-(P r) const { return {x - r.x, y - r.y}; }
    P operator*(T r) const { return {x * r, y * r}; }
    P operator/(T r) const { return {x / r, y / r}; }
    P operator-() const { return {-x, -y}; }
    friend P operator*(T l, P r) { return {l * r.x, l * r.y};
         }

    P rot() const { return {-y, x}; }
    T operator*(P r) const { return x * r.x + y * r.y; }
    T operator%(P r) const { return rot() * r; }
    T left(P a, P b) { return (b - a) % (*this - a); }

    T magsq() const { return x * x + y * y; }
    T mag() const { return sqrt(magsq()); }
    P unit() const { return *this / mag(); }

    bool half() const { return abs(y) <= EPS && x < -EPS || y
         < -EPS; }
    T angcmp(P r) const { // like strcmp(this, r)
        int h = (int)half() - r.half();
        return h ? h : r % *this;
    }
    T angcmp_rel(P a, P b) { // like strcmp(a, b)
        P z = *this;
        int h = z % a <= 0 && z * a < 0 || z % a < 0;
        h -= z % b <= 0 && z * b < 0 || z % b < 0;
        return h ? h : b % a;
    }

    bool operator==(P r) const { return abs(x - r.x) <= EPS
         && abs(y - r.y) <= EPS; }

    double angle() const { return atan2(y, x); }
    static P from_angle(double a) { return {cos(a), sin(a)};
         }
};
```

3.9 polygon

```
// get TWICE the area of a simple polygon in ccw order
T area2(const vector<P> &p) {
    int n = p.size(); T a = 0;
    rep(i, n) a += (p[i] - p[0]) % (p[(i + 1) % n] - p[i]);
    return a;
}

// checks whether a point is inside a ccw simple polygon
// returns 1 if inside, 0 if on border, -1 if outside
int in_poly(const vector<P> &p, P q) {
    int w = 0;
    rep(i, p.size()) {
        P a = p[i], b = p[(i + 1) % p.size()];
        T k = (b - a) % (q - a);
        T u = a.y - q.y, v = b.y - q.y;
        if (k > 0 && u < 0 && v >= 0) w++;
        if (k < 0 && v < 0 && u >= 0) w--;
        if (k == 0 && (q - a) * (q - b) <= 0) return 0;
    }
    return w ? 1 : -1;
}

// check if point in ccw convex polygon, O(log n)
// + if inside, 0 if on border, - if outside
T in_convex(const vector<P> &p, P q) {
    int l = 1, h = p.size() - 2; assert(p.size() >= 3);
    while (l != h) { // collinear points are unsupported!
        int m = (l + h + 1) / 2;
        if (q.left(p[0], p[m]) >= 0) l = m;
        else h = m - 1;
    }
    T in = min(q.left(p[0], p[1]), q.left(p.back(), p[0]));
    return min(in, q.left(p[l], p[l + 1]));
}

int extremal(const vector<P> &p, P d) {
    int n = p.size(), l = 0, r = n - 1; assert(n);
    P e0 = (p[n - 1] - p[0]).rot();
    while (l < r) { // polygon must be convex
        int m = (l + r + 1) / 2;
        P e = (p[(m + n - 1) % n] - p[m]).rot();
        if (e0.angcmp_rel(d, e) < 0) r = m - 1;
        else l = m;
    }
    return l;
}

// square dist of most distant points of a ccw convex
// polygon with NO COLLINEAR POINTS
T callipers(const vector<P> &p) {
    int n = p.size();
    T r = 0;
    for (int i = 0, j = n < 2 ? 0 : 1; i < j; i++) {
        for (;; j = (j + 1) % n) {
            r = max(r, (p[i] - p[j]).magsq());
            if ((p[(i + 1) % n] - p[i]) % (p[(j + 1) % n] - p
                [j]) <= EPS) break;
        }
    }
    return r;
}
```

```
P centroid(const vector<P> &p) { // (barycenter)
    P r(0, 0); T t = 0; int n = p.size();
    rep(i, n) {
        r += (p[i] + p[(i+1)%n]) * (p[i] % p[(i+1)%n]);
        t += p[i] % p[(i+1)%n];
    }
    return r / t / 3;
}

// classify collision of a ray inside a ccw polygon vertex.
// ray is (o, d), vertex is b, previous vertex is a, next is
     c.
pair<bool, bool> inner_collide(P o, P d, P a, P b, P c) {
    T p = (a - o) % d;     // side of previous
    T n = (c - o) % d;     // side of next
    T v = (c - b) % (b - a); // is vertex convex?
    return {v > 0 ? n < 0 || (n == 0 && p < 0) : p > 0 || n <
         0,
          v > 0 ? p > 0 || (p == 0 && n > 0) : p > 0 || n <
              0};
}
```

3.10 sweep

```
#include "point.cpp"

// iterate over all pairs of points
// `op` is called with all ordered pairs of different
     indices `(i, j)`
// additionally, the `ps` vector is kept sorted by signed
     distance
// to the line formed by `i` and `j`
// for example, if the vector from `i` to `j` is pointing
     right,
// the `ps` vector is sorted from smallest `y` to largest `y
     `
// note that, because the `ps` vector is sorted by signed
     distance,
// `j` is always equal to `i + 1`
// this means that the amount of points to the left of the
     line is always `N - i`
template <class OP>
void all_pair_points(vector<P> &ps, OP op) {
    int N = ps.size();
    sort(ps.begin(), ps.end(), [](P a, P b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
    });
    vector<pair<int, int>> ss;
    rep(i, N) rep(j, N) if (i != j) ss.push_back({i, j});
    stable_sort(ss.begin(), ss.end(), [&](auto a, auto b) {
        return (ps[a.second] - ps[a.first]).angle_lt(ps[b.
            second] - ps[b.first]);
    });
    vector<int> p(N); rep(i, N) p[i] = i;
    for (auto [i, j] : ss)
        { op(p[i], p[j]); swap(ps[p[i]], ps[p[j]]); swap(p[i
            ], p[j]); }
}
```

3.11 theorems

```
// Pick's theorem
//     Simple polygon with integer vertices:
//     A = I + B / 2 - 1
//     A: Area of the polygon
//     I: Integer points strictly inside the polygon
//     B: Integer points on the boundary of the polygon
```

4 graph

4.1 artic-bridge-biconn

```
vector<int> g[MAXN];int n;
struct edge {int u,v,comp;bool bridge;};
vector<edge> e;
void add_edge(int u, int v){
    g[u].pb(e.size());g[v].pb(e.size());
    e.pb((edge){u,v,-1,false});
}
int D[MAXN],B[MAXN],T;
int nbc; // number of biconnected components
int art[MAXN]; // articulation point iff !=0
stack<int> st; // only for biconnected
void dfs(int u,int pe){
    B[u]=D[u]=T++;
    for(int ne:g[u])if(ne!=pe){
        int v=e[ne].u^e[ne].v^u;
        if(D[v]<0){
            st.push(ne);dfs(v,ne);
            if(B[v]>D[u])e[ne].bridge = true; // bridge
            if(B[v]>=D[u]){
                art[u]++; // articulation
                int last; // start biconnected
                do{last=st.top();st.pop();e[last].comp=nbc;}
                while(last!=ne);
                nbc++; // end biconnected
            }
            B[u]=min(B[u],B[v]);
        }
        else if(D[v]<D[u])st.push(ne),B[u]=min(B[u],D[v]);
    }
}
void doit(){
    memset(D,-1,sizeof(D));memset(art,0,sizeof(art));
    nbc=T=0; fore(i,0,n)if(D[i]<0)dfs(i,-1),art[i]--;
}
```

4.2 bellman-ford

```
struct Edge { int u, v; ll w; };

// find distance from source node to all nodes.
// supports negative edge weights.
// returns true if a negative cycle is detected.
//
// time: O(V E)
bool bellman_ford(int N, int s, vector<Edge> &E, vector<ll>
     &D, vector<int> &P) {
    P.assign(N, -1), D.assign(N, INF), D[s] = 0;
    rep(i, N - 1) {
        bool f = true;
        rep(ei, E.size()) {
            auto &e = E[ei];
            ll n = D[e.u] + e.w;
            if (D[e.u] < INF && n < D[e.v])
                D[e.v] = n, P[e.v] = ei, f = false;
        }
        if (f) return false;
    }
    return true;
}
```

4.3 blossom

```
vector<int> g[MAXN];int n,m,mt[MAXN],qh,qt,q[MAXN],ft[MAXN],
bs[MAXN];bool inq[MAXN],inb[MAXN],inp[MAXN];int lca(int root
,int x,int y){memset(inp,0,sizeof(inp));while(1){inp[x=bs[x]
]=true;if(x==root)break;x=ft[mt[x]];}while(1){if(inp[y=bs[y]
])return y;else y=ft[mt[y]];}}void mark(int z,int x){while(
bs[x]!=z){int y=mt[x];inb[bs[x]]=inb[bs[y]]=true;x=ft[y];if(
bs[x]!=z)ft[x]=y;}}void contr(int s,int x,int y){int z=lca(s
,x,y);memset(inb,0,sizeof(inb));mark(z,x);mark(z,y);if(bs[x]
```

```
!=z)ft[x]=y;if(bs[y]!=z)ft[y]=x;rep(x,n)if(inb[bs[x]]){bs[x]
=z;if(!inq[x])inq[q[++qt]=x]=true;}}int findp(int s){memset(
inq,0,sizeof(inq));memset(ft,-1,sizeof(ft));rep(i,n)bs[i]=i;
inq[q[qh=qt=0]=s]=true;while(qh<=qt){int x=q[qh++];for(int y
:g[x])if(bs[x]!=bs[y]&&mt[x]!=y){if(y==s||mt[y]>=0&&ft[mt[y]
]>=0)contr(s,x,y);else if(ft[y]<0){ft[y]=x;if(mt[y]<0)return
y;else if(!inq[mt[y]])inq[q[++qt]=mt[y]]=true;}}}return -1;}
int aug(int s,int t){int x=t,y,z;while(x>=0){y=ft[x];z=mt[y]
;mt[y]=x;mt[x]=y;x=z;}return t>=0;}int edmonds(){int r=0;
memset(mt,-1,sizeof(mt));rep(x,n)if(mt[x]<0)r+=aug(x,findp(x
));return r;}
```

4.4 chu-liu-minimum-spanning-arborescence

```
//O(n*m) minimum spanning tree in directed graph
//returns -1 if not possible
//included i-th edge if take[i]!=0
typedef int tw; tw INF=1ll<<30;
struct edge{int u,v,id;tw len;};
struct ChuLiu{
    int n; vector<edge> e;
    vector<int> inc,dec,take,pre,num,id,vis;
    vector<tw> inw;
    void add_edge(int x, int y, tw w){
        inc.pb(0); dec.pb(0); take.pb(0);
        e.pb({x,y,SZ(e),w});
    }
    ChuLiu(int n):n(n),pre(n),num(n),id(n),vis(n),inw(n){}
    tw doit(int root){
        auto e2=e;
        tw ans=0; int eg=SZ(e)-1,pos=SZ(e)-1;
        while(1){
            fore(i,0,n) inw[i]=INF,id[i]=vis[i]=-1;
            for(auto ed:e2) if(ed.len<inw[ed.v]){
                inw[ed.v]=ed.len; pre[ed.v]=ed.u;
                num[ed.v]=ed.id;
            }
            inw[root]=0;
            fore(i,0,n) if(inw[i]==INF) return -1;
            int tot=-1;
            fore(i,0,n){
                ans+=inw[i];
                if(i!=root)take[num[i]]++;
                int j=i;
                while(vis[j]!=i&&j!=root&&id[j]<0)vis[j]=i,j=
                    pre[j];
                if(j!=root&&id[j]<0){
                    id[j]=++tot;
                    for(int k=pre[j];k!=j;k=pre[k]) id[k]=tot;
                }
            }
            if(tot<0)break;
            fore(i,0,n) if(id[i]<0)id[i]=++tot;
            n=tot+1; int j=0;
            fore(i,0,SZ(e2)){
                int v=e2[i].v;
                e2[j].v=id[e2[i].v];
                e2[j].u=id[e2[i].u];
                if(e2[j].v!=e2[j].u){
                    e2[j].len=e2[i].len-inw[v];
                    inc.pb(e2[i].id);
                    dec.pb(num[v]);
                    take.pb(0);
                    e2[j++].id=++pos;
                }
            }
            e2.resize(j);
            root=id[root];
        }
        while(pos>eg){
            if(take[pos]>0) take[inc[pos]]++, take[dec[pos
                ]]--;
            pos--;
        }
        return ans;
    }
};
```

4.5 dinic

```
// time: O(E V^2)
//      O(E V^(2/3)) / O(E sqrt(E)) unit capacities
//      O(E sqrt(V))  (hopcroft-karp) unit networks
//unit network: c in {0,1} & forall v, indeg<=1 or outdeg<=1
//min-cut: nodes reachable from s in final residual graph
struct Dinic {
    struct Edge { int u, v; ll c, f = 0; };
    int N, s, t; vector<vector<int>> G;
    vector<Edge> E; vector<int> lvl, ptr;
    Dinic() {}
    Dinic(int N, int s, int t) : N(N), s(s), t(t), G(N) {}

    void add_edge(int u, int v, ll c) {
        G[u].push_back(E.size()); E.push_back({u, v, c});
        G[v].push_back(E.size()); E.push_back({v, u, 0});
    }

    ll push(int u, ll p) {
        if (u == t || p <= 0) return p;
        while (ptr[u] < G[u].size()) {
            int ei = G[u][ptr[u]++];
            Edge &e = E[ei];
            if (lvl[e.v] != lvl[u] + 1) continue;
            ll a = push(e.v, min(e.c - e.f, p));
            if (a <= 0) continue;
            e.f += a, E[ei ^ 1].f -= a; return a;
        }
        return 0;
    }

    ll maxflow() {
        ll f = 0;
        while (true) {
            lvl.assign(N, -1); queue<int> q;
            lvl[s] = 0; q.push(s);
            while (!q.empty()) {
                int u = q.front(); q.pop();
                for (int ei : G[u]) {
                    Edge &e = E[ei];
                    if (e.c-e.f<=0||lvl[e.v]!=-1) continue;
                    lvl[e.v] = lvl[u] + 1; q.push(e.v);
                }
            }
            if (lvl[t] == -1) break;
            ptr.assign(N,0);while(ll ff=push(s,INF))f += ff;
        }
        return f;
    }
};

/* Flujo con demandas (no necesariamente el maximo)
Agregar s' y t' nuevos source and sink
c'(s', v) = sum(d(u, v) for u in V) \forall arista (s', v)
c'(v, t') = sum(d(v, w) for w in V) \forall arista (v, t')
c'(u, v) = c(u, v) - d(u, v) \forall aristas antiguas
c'(t, s) = INF (el flujo por esta arista es el flujo real)*/
```

4.6 dominator-tree

```
//idom[i]=parent of i in dominator tree with root=rt, or -1
    if not exists
int n,rnk[MAXN],pre[MAXN],anc[MAXN],idom[MAXN],semi[MAXN],
    low[MAXN];
vector<int> g[MAXN],rev[MAXN],dom[MAXN],ord;
void dfspre(int pos){
    rnk[pos]=SZ(ord); ord.pb(pos);
    for(auto x:g[pos]){
```

```
        rev[x].pb(pos);
        if(rnk[x]==n) pre[x]=pos,dfspre(x);
    }
}
int eval(int v){
    if(anc[v]<n&&anc[anc[v]]<n){
        int x=eval(anc[v]);
        if(rnk[semi[low[v]]]>rnk[semi[x]]) low[v]=x;
        anc[v]=anc[anc[v]];
    }
    return low[v];
}
void dominators(int rt){
    fore(i,0,n){
        dom[i].clear(); rev[i].clear();
        rnk[i]=pre[i]=anc[i]=idom[i]=n;
        semi[i]=low[i]=i;
    }
    ord.clear(); dfspre(rt);
    for(int i=SZ(ord)-1;i;i--){
        int w=ord[i];
        for(int v:rev[w]){
            int u=eval(v);
            if(rnk[semi[w]]>rnk[semi[u]])semi[w]=semi[u];
        }
        dom[semi[w]].pb(w); anc[w]=pre[w];
        for(int v:dom[pre[w]]){
            int u=eval(v);
            idom[v]=(rnk[pre[w]]>rnk[semi[u]]?u:pre[w]);
        }
        dom[pre[w]].clear();
    }
    for(int w:ord) if(w!=rt&&idom[w]!=semi[w]) idom[w]=idom[
        idom[w]];
    fore(i,0,n) if(idom[i]==n)idom[i]=-1;
}
```

4.7 eulerian-path

```
// Directed version(uncomment commented code for undirected)
struct edge {
    int y; //   list<edge>::iterator rev;
    edge(int y):y(y){}
};
list<edge> g[MAXN];
void add_edge(int a, int b){
    g[a].push_front(edge(b));//auto ia=g[a].begin();
//   g[b].push_front(edge(a));auto ib=g[b].begin();
//   ia->rev=ib;ib->rev=ia;
}
vector<int> p;
void go(int x){
    while(g[x].size()){
        int y=g[x].front().y;//g[y].erase(g[x].front().rev);
        g[x].pop_front(); go(y);
    }
    p.push_back(x);
}
vector<int> get_path(int x){ // get a path that begins in x
// check that a path exists from x before calling get_path!
    p.clear();go(x);reverse(p.begin(),p.end());
    return p;
}
```

4.8 floyd-warshall

```
// calculate distances between every pair of nodes in O(V^3)
    time.
// works with negative edges, but not negative cycles.
void floyd(const vector<vector<pair<ll, int>>> &G, vector<
    vector<ll>> &D) {
    int N = G.size();
    D.assign(N, vector<ll>(N, INF));
    rep(u, N) D[u][u] = 0;
    rep(u, N) for (auto [w, v] : G[u]) D[u][v] = w;
    rep(k, N) rep(u, N) rep(v, N)
        D[u][v] = min(D[u][v], D[u][k] + D[k][v]);
}
```

4.9 heavy-light

```
struct Hld {
    vector<int> P, H, D, pos, top;

    Hld() {}
    void init(vector<vector<int>> &G) {
        int N = G.size();
        P.resize(N), H.resize(N), D.resize(N), pos.resize(N),
            top.resize(N);
        D[0] = -1, dfs(G, 0); int t = 0;
        rep(i, N) if (H[P[i]] != i) {
            int j = i;
            while (j != -1)
                { top[j] = i, pos[j] = t++; j = H[j]; }
        }
    }

    int dfs(vector<vector<int>> &G, int i) {
        int w = 1, mw = 0;
        D[i] = D[P[i]] + 1, H[i] = -1;
        for (int c : G[i]) {
            if (c == P[i]) continue;
            P[c] = i; int sw = dfs(G, c); w += sw;
            if (sw > mw) H[i] = c, mw = sw;
        }
        return w;
    }

    // visit the log N segments in the path from u to v
    template <class OP>
    void path(int u, int v, OP op) {
        while (top[u] != top[v]) {
            if (D[top[u]] > D[top[v]]) swap(u, v);
            op(pos[top[v]], pos[v] + 1); v = P[top[v]];
        }
        if (D[u] > D[v]) swap(u, v);
        op(pos[u], pos[v] + 1); // value on node
        // op(pos[u]+1, pos[v] + 1); // value on edge
    }

    // an alternative to `path` that considers order.
    // calls `op` with an `l <= r` inclusive-exclusive range,
        and a
    // boolean indicating if the query is forwards or
        backwards.
    template <class OP>
    void path(int u, int v, OP op) {
        int lu = u, lv = v;
        while (top[lu] != top[lv])
            if (D[top[lu]] > D[top[lv]]) lu = P[top[lu]];
            else lv = P[top[lv]];
        int lca = D[lu] > D[lv] ? lv : lu;

        while (top[u] != top[lca])
            op(pos[top[u]], pos[u] + 1, false), u = P[top[u
                ]];
        if (u != lca) op(pos[lca] + 1, pos[u] + 1, false);

        vector<int> stk;
        while (top[v] != top[lca])
            stk.push_back(v), v = P[top[v]];

        // op(pos[lca], pos[v] + 1, true); // value on node
        op(pos[lca] + 1, pos[v] + 1, true); // value on edge
        reverse(stk.begin(), stk.end());
        for (int w : stk) op(pos[top[w]], pos[w] + 1, true);
```

```
    }

    // commutative segment tree
    template <class T, class S>
    void update(S &seg, int i, T val) { seg.update(pos[i],
         val); }

    // commutative segment tree lazy
    template <class T, class S>
    void update(S &seg, int u, int v, T val) {
        path(u, v, [&](int l, int r) { seg.update(l, r, val);
             });
    }

    // commutative (lazy) segment tree
    template <class T, class S>
    T query(S &seg, int u, int v) {
        T ans = 0;
            // neutral element
        path(u, v, [&](int l, int r) { ans += seg.query(l, r)
             ; }); // query op
        return ans;
    }
};
```

4.10 hungarian

```
// find a maximum gain perfect matching in the given
     bipartite complete graph.
// input: gain matrix (G_{xy} = benefit of joining vertex x
     in set X with vertex
// y in set Y).
// output: maximum gain matching in members `xy[x]` and `yx[
     y]`.
// runtime: O(N^3)
struct Hungarian {
    int N, qi, root;
    vector<vector<ll>> gain;
    vector<int> xy, yx, p, q, slackx;
    vector<ll> lx, ly, slack;
    vector<bool> S, T;

    void add(int x, int px) {
        S[x] = true, p[x] = px;
        rep(y, N) if (lx[x] + ly[y] - gain[x][y] < slack[y])
             {
            slack[y] = lx[x] + ly[y] - gain[x][y], slackx[y]
                 = x;
        }
    }

    void augment(int x, int y) {
        while (x != -2) {
            yx[y] = x; swap(xy[x], y); x = p[x];
        }
    }

    void improve() {
        S.assign(N, false), T.assign(N, false), p.assign(N,
             -1);
        qi = 0, q.clear();
        rep(x, N) if (xy[x] == -1) {
            q.push_back(root = x), p[x] = -2, S[x] = true;
            break;
        }
        rep(y, N) slack[y] = lx[root] + ly[y] - gain[root][y
             ], slackx[y] = root;

        while (true) {
            while (qi < q.size()) {
                int x = q[qi++];
                rep(y, N) if (lx[x] + ly[y] == gain[x][y] && !
                     T[y]) {
                    if (yx[y] == -1) return augment(x, y);
                    T[y] = true, q.push_back(yx[y]), add(yx[y
                         ], x);
                }
            }

            ll d = INF;
            rep(y, N) if (!T[y]) d = min(d, slack[y]);
            rep(x, N) if (S[x]) lx[x] -= d;
            rep(y, N) if (T[y]) ly[y] += d;
            rep(y, N) if (!T[y]) slack[y] -= d;

            rep(y, N) if (!T[y] && slack[y] == 0) {
                if (yx[y] == -1) return augment(slackx[y], y);
                T[y] = true;
                if (!S[yx[y]]) q.push_back(yx[y]), add(yx[y],
                     slackx[y]);
            }
        }
    }

    Hungarian(vector<vector<ll>> g)
        : N(g.size()), gain(g), xy(N, -1), yx(N, -1), lx(N, -
             INF),
        ly(N), slack(N), slackx(N) {
        rep(x, N) rep(y, N) lx[x] = max(lx[x], ly[y]);
        rep(i, N) improve();
    }
};
```

4.11 kuhn

```
// get a maximum cardinality matching in a bipartite graph.
// input: adjacency lists.
// output: matching (in `mt` member).
// runtime: O(V E)
struct Kuhn {
    vector<vector<int>> G;
    int N, size;
    vector<bool> seen;
    vector<int> mt;

    bool visit(int i) {
        if (seen[i]) return false;
        seen[i] = true;
        for (int to : G[i])
            if (mt[to] == -1 || visit(mt[to])) {
                mt[to] = i;
                return true;
            }
        return false;
    }

    Kuhn(vector<vector<int>> adj) : G(adj), N(G.size()), mt(N
         , -1) {
        rep(i, N) {
            seen.assign(N, false);
            size += visit(i);
        }
    }
};
```

4.12 lca

```
// calculates the lowest common ancestor for any two nodes
     in O(log N) time,
// with O(N log N) preprocessing
struct Lca {
    int N, K, t = 0;
    vector<vector<int>> U;
    vector<int> L, R;

    Lca() {}
```

```
    Lca(vector<vector<int>> &G) : N(G.size()), L(N), R(N) {
        K = N <= 1 ? 0 : 32 - __builtin_clz(N - 1);
        U.resize(K + 1, vector<int>(N));
        visit(G, 0, 0);
        rep(k, K) rep(u, N) U[k + 1][u] = U[k][U[k][u]];
    }

    void visit(vector<vector<int>> &G, int u, int p) {
        L[u] = t++, U[0][u] = p;
        for (int v : G[u]) if (v != p) visit(G, v, u);
        R[u] = t++;
    }

    bool is_anc(int up, int dn) {
        return L[up] <= L[dn] && R[dn] <= R[up];
    }

    int find(int u, int v) {
        if (is_anc(u, v)) return u;
        if (is_anc(v, u)) return v;
        for (int k = K; k >= 0;)
            if (is_anc(U[k][u], v)) k--;
            else u = U[k][u];
        return U[0][u];
    }
};
```

4.13 maxflow-mincost

```
// time: O(F V E)         F is the maximum flow
//      O(V E + F E log V) if bellman-ford is replaced by
     johnson
struct Flow {
    struct Edge {
        int u, v;
        ll c, w, f = 0;
    };

    int N, s, t;
    vector<vector<int>> G;
    vector<Edge> E;
    vector<ll> d, b;
    vector<int> p;

    Flow() {}
    Flow(int N, int s, int t) : N(N), s(s), t(t), G(N) {}

    void add_edge(int u, int v, ll c, ll w) {
        G[u].push_back(E.size());
        E.push_back({u, v, c, w});
        G[v].push_back(E.size());
        E.push_back({v, u, 0, -w});
    }

    // naive distances with bellman-ford: O(V E)
    void calcdists() {
        p.assign(N, -1), d.assign(N, INF), d[s] = 0;
        rep(i, N - 1) rep(ei, E.size()) {
            Edge &e = E[ei];
            ll n = d[e.u] + e.w;
            if (d[e.u] < INF && e.c - e.f > 0 && n < d[e.v])
                d[e.v] = n, p[e.v] = ei;
        }
    }

    // johnsons potentials: O(E log V)
    void calcdists() {
        if (b.empty()) {
            b.assign(N, 0);
            // code below only necessary if there are
                negative costs
            rep(i, N - 1) rep(ei, E.size()) {
                Edge &e = E[ei];
                if (e.f < e.c) b[e.v] = min(b[e.v], b[e.u] + e
                    .w);
            }
        }
        p.assign(N, -1), d.assign(N, INF), d[s] = 0;
        priority_queue<pair<ll, int>> q;
        q.push({0, s});
        while (!q.empty()) {
            auto [w, u] = q.top();
            q.pop();
            if (d[u] < -w + b[u]) continue;
            for (int ei : G[u]) {
                auto e = E[ei];
                ll n = d[u] + e.w;
                if (e.f < e.c && n < d[e.v]) {
                    d[e.v] = n, p[e.v] = ei;
                    q.push({b[e.v] - n, e.v});
                }
            }
        }
        b = d;
    }

    ll solve() {
        b.clear();
        ll ff = 0;
        while (true) {
            calcdists();
            if (p[t] == -1) break;

            ll f = INF;
            for (int cur = t; p[cur] != -1; cur = E[p[cur]].u
                )
                f = min(f, E[p[cur]].c - E[p[cur]].f);
            for (int cur = t; p[cur] != -1; cur = E[p[cur]].u
                )
                E[p[cur]].f += f, E[p[cur] ^ 1].f -= f;
            ff += f;
        }
        return ff;
    }
};
```

4.14 parallel-dfs

```
struct Tree {
    int n,z[2];
    vector<vector<int>> g;
    vector<int> ex,ey,p,w,f,v[2];
    Tree(int n):g(n),w(n),f(n){}
    void add_edge(int x, int y){
        p.pb(g[x].size());g[x].pb(ex.size());
        ex.pb(x);ey.pb(y);
        p.pb(g[y].size());g[y].pb(ex.size());
        ex.pb(y);ey.pb(x);
    }
    bool go(int k){//returns 1 if it finds new node
        int& x=z[k];
        while(x>=0&&
            (w[x]==g[x].size()||w[x]==g[x].size()-1
            &&(g[x].back()^1)==f[x]))
            x=f[x]>=0?ex[f[x]]:-1;
        if(x<0)return false;
        if((g[x][w[x]]^1)==f[x])w[x]++;
        int e=g[x][w[x]],y=ey[e]; f[y]=e;
        w[x]++; w[y]=0; x=y; v[k].pb(x);
        return true;
    }
    vector<int> erase_edge(int e){
        e*=2;//erases eth edge, returns smaller comp
        int x=ex[e],y=ey[e]; p[g[x].back()]=p[e];
        g[x][p[e]]=g[x].back(); g[x].pop_back();
        p[g[y].back()]=p[e^1]; g[y][p[e^1]]=g[y].back();
        g[y].pop_back();
        f[x]=f[y]=-1; w[x]=w[y]=0; z[0]=x;z[1]=y;
```

```
        v[0]={x};v[1]={y};
        bool d0=true,d1=true;while(d0&&d1)d0=go(0),d1=go(1);
        return v[1-d1];
    }
};
```

4.15 push-relabel

```
#include "../common.h"

const ll INF = 1e18;

// maximum flow algorithm.
// to run, use `maxflow()`.
//
// time: O(V^2 sqrt(E)) <= O(V^3)
// memory: O(V^2)
struct PushRelabel {
    vector<vector<ll>> cap, flow;
    vector<ll> excess;
    vector<int> height;

    PushRelabel() {}
    void resize(int N) { cap.assign(N, vector<ll>(N)); }

    // push as much excess flow as possible from u to v.
    void push(int u, int v) {
        ll f = min(excess[u], cap[u][v] - flow[u][v]);
        flow[u][v] += f;
        flow[v][u] -= f;
        excess[v] += f;
        excess[u] -= f;
    }

    // relabel the height of a vertex so that excess flow may
        be pushed.
    void relabel(int u) {
        int d = INT32_MAX;
        rep(v, cap.size()) if (cap[u][v] - flow[u][v] > 0) d
            =
            min(d, height[v]);
        if (d < INF) height[u] = d + 1;
    }

    // get the maximum flow on the network specified by `cap`
        with source `s`
    // and sink `t`.
    // node-to-node flows are output to the `flow` member.
    ll maxflow(int s, int t) {
        int N = cap.size(), M;
        flow.assign(N, vector<ll>(N));
        height.assign(N, 0), height[s] = N;
        excess.assign(N, 0), excess[s] = INF;
        rep(i, N) if (i != s) push(s, i);

        vector<int> q;
        while (true) {
            // find the highest vertices with excess
            q.clear(), M = 0;
            rep(i, N) {
                if (excess[i] <= 0 || i == s || i == t)
                    continue;
                if (height[i] > M) q.clear(), M = height[i];
                if (height[i] >= M) q.push_back(i);
            }
            if (q.empty()) break;
            // process vertices
            for (int u : q) {
                bool relab = true;
                rep(v, N) {
                    if (excess[u] <= 0) break;
                    if (cap[u][v] - flow[u][v] > 0 && height[u
                        ] > height[v])
                        push(u, v), relab = false;
                }
                if (relab) {
                    relabel(u);
                    break;
                }
            }
        }

        ll f = 0; rep(i, N) f += flow[i][t]; return f;
    }
};
```

4.16 strongly-connected-components

```
/* time: O(V + E), memory: O(V)
after building:
    comp = map from vertex to component
           (components are toposorted, root first, leaf last)
    N = number of components
    G = condensation graph (component DAG)
byproducts:
    vgi = transposed graph
    order = reverse topological sort (leaf first, root last)
others:
    vn = number of vertices
    vg = original vertex graph        */
struct Scc {
    int vn, N;
    vector<int> order, comp;
    vector<vector<int>> vg, vgi, G;
    void toposort(int u) {
        if (comp[u]) return;
        comp[u] = -1;
        for (int v : vg[u]) toposort(v);
        order.push_back(u);
    }
    bool carve(int u) {
        if (comp[u] != -1) return false;
        comp[u] = N;
        for (int v : vgi[u]) {
            carve(v);
            if (comp[v] != N) G[comp[v]].push_back(N);
        }
        return true;
    }
    Scc() {}
    Scc(vector<vector<int>> &g)
     : vn(g.size()), vg(g), comp(vn), vgi(vn), G(vn), N(0) {
        rep(u, vn) toposort(u);
        rep(u, vn) for (int v : vg[u]) vgi[v].push_back(u);
        invrep(i, vn) N += carve(order[i]);
    }
};
```

4.17 two-sat

```
// calculate the solvability of a system of logical
    equations, where every equation is of the form `a or b
    `.
// `neg`: get negation of `u`
// `then`: `u` implies `v`
// `any`: `u` or `v`
// `set`: `u` is true
//
// after `solve` (O(V+E)) returns true, `sol` contains one
    possible solution.
// determining all solutions is O(V*E) hard (requires
    computing reachability in a DAG).
struct TwoSat {
    int N; vector<vector<int>> G;
    Scc scc; vector<bool> sol;
    TwoSat(int n) : N(n), G(2 * n), sol(n) {}
    TwoSat() {}
```

```
    int neg(int u) { return (u + N) % (2 * N); }
    void then(int u, int v) { G[u].push_back(v), G[neg(v)].
        push_back(neg(u)); }
    void any(int u, int v) { then(neg(u), v); }
    void set(int u) { G[neg(u)].push_back(u); }

    bool solve() {
        scc = Scc(G);
        rep(u, N) if (scc.comp[u] == scc.comp[neg(u)]) return
            false;
        rep(u, N) sol[u] = (scc.comp[u] > scc.comp[neg(u)]);
        return true;
    }
};
```

5 implementation

5.1 bit-tricks

```
y = x & (x-1) // Turn off rightmost 1bit
y = x & (-x)  // Isolate rightmost 1bit
y = x | (x-1) // Right propagate rightmost 1bit(fill in 1s)
y = x | (x+1) // Turn on rightmost 0bit
y = ~x & (x+1) // Isolate rightmost 0bit
// If x is of long type, use __builtin_popcountl(x)
// If x is of long long type, use __builtin_popcountll(x)
// 1. Counts the number of ones(set bits) in an integer.
__builtin_popcount(x)
// 2. Checks the Parity of a number. Returns true(1) if the
// number has odd number of set bits, else it returns
// false(0) for even number of set bits.
 __builtin_parity(x)
// 3. Counts the leading number of zeros of the integer.
__builtin_clz(x)
// 4. Counts the trailing number of zeros of the integer.
__builtin_ctz(x)
// 5. Returns 1 + the index of the least significant 1-bit.
__builtin_ffs(x) // If x == 0, returns 0.
// Iterate over non empty subsets of bitmask
for(int s=m;s;s=(s-1)&m) // Decreasing order
for (int s=0;s=s-m&m;) // Increasing order
```

5.2 common-template

```
#pragma GCC optimize("Ofast")
#pragma GCC target("bmi,bmi2,lzcnt,popcnt")
#pragma GCC target("avx,avx2,f16c,fma,sse3,ssse3,sse4.1,sse4
    .2")
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define repx(i, a, b) for (int i = a; i < b; i++)
#define rep(i, n) repx(i, 0, n)
#define invrepx(i, a, b) for (int i = b - 1; i >= a; i--)
#define invrep(i, n) invrepx(i, 0, n)
// Command to check time and memory usage:
//     /usr/bin/time -v ./tmp
// See "Maximum resident set size" for max memory used
// Commands for interactive checker:
//     mkfifo fifo
//     (./solution < fifo) | (./interactor > fifo)
// Does not work on the Windows file system, i.e., /mnt/c/
// The special fifo file must be used, otherwise the
// solution will not wait for input and will read EOF
```

5.3 dynamic-connectivity

```
struct DC {
    int n; Dsu D;
    vector<vector<pair<int, int>>> t;
    DC(int N) : n(N), D(N), t(2 * N) {}
    // add edge p to all times in interval [l, r]
    void upd(int l, int r, pair<int, int> p) {
        for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
            if (l & 1) t[l++].push_back(p);
            if (r & 1) t[--r].push_back(p);
        }
    }
    void process(int u = 1) { // process all queries
        for (auto &e : t[u]) D.unite(e.first, e.second);
        if (u >= n) {
            // do stuff with D at time u - n
        } else process(2 * u), process(2 * u + 1);
        for (auto &e : t[u]) D.rollback();
    }
};
```

5.4 hash-container

```
namespace{//add (#define tmpl template)(#define ty typename)
  tmpl<ty T> size_t mk_h(const T& v){return hash<T>()(v);}
  void h_cmb(size_t& h, const size_t& v)
  { h ^= v + 0x9e3779b9 + (h << 6) + (h >> 2); }
  tmpl<ty T> struct h_ct{size_t operator()(const T& v)const{
 size_t h=0;for(const auto& e:v){h_cmb(h,mk_h(e));}return h;
  }};
}namespace std{//support for pair<T,U>, vector<T> & map<T,U>
  tmpl<ty T, ty U> struct hash<pair<T, U>>{
    size_t operator()(const pair<T,U>& v) const
 {size_t h=mk_h(v.first);h_cmb(h, mk_h(v.second));return h;}
  };
tmpl<ty... T>struct hash<vector<T...>>:h_ct<vector<T...>>{};
tmpl<ty... T>struct hash<map<T...>>:h_ct<map<T...>>{}; }
```

5.5 mo

```
struct Query { int l, r, idx; };

// answer segment queries using only `add(i)`, `remove(i)`
    and `get()`
// functions.
//
// complexity: O((N + Q) * sqrt(N) * F)
//   N = length of the full segment
//   Q = amount of queries
//   F = complexity of the `add`, `remove` functions
template <class A, class R, class G, class T>
void mo(vector<Query> &queries, vector<T> &ans, A add, R
    remove, G get) {
    int Q = queries.size(), B = (int)sqrt(Q);
    sort(queries.begin(), queries.end(), [&](Query &a, Query
        &b) {
        return make_pair(a.l / B, a.r) < make_pair(b.l / B, b
            .r);
    });
    ans.resize(Q);

    int l = 0, r = 0;
    for (auto &q : queries) {
        while (r < q.r) add(r), r++;
        while (l > q.l) l--, add(l);
        while (r > q.r) r--, remove(r);
        while (l < q.l) remove(l), l++;
        ans[q.idx] = get();
    }
}
```

5.6 ordered-set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

typedef tree<int, null_type, less<int>, rb_tree_tag,
          tree_order_statistics_node_update> ordered_set;
// find_by_order(i) -> iterator to ith element
// order_of_key(k) -> position (int) of lower_bound of k
```

5.7 unordered-map

```
static mt19937 rng(chrono::steady_clock::now().
    time_since_epoch().count());
#define rnd(a,b) (uniform_int_distribution<ll>(a,b)(rng))

struct Hash {
    size_t operator()(const ll &x) const {
        const uint64_t RAND = chrono::steady_clock::now()
            .time_since_epoch().count();
        uint64_t z = x + RAND + 0x9e3779b97f4a7c15;
        z = (z ^ (z >> 30)) * 0xbf58476d1ce4e5b9;
        z = (z ^ (z >> 27)) * 0x94d049bb133111eb;
        return z ^ (z >> 31);
    }
};
template<class T,class U>using umap=unordered_map<T,U,Hash>;
template<class T> using uset = unordered_set<T, Hash>;
```

6 math

6.1 Linear Diophantine

```
ii extendedEuclid(ll a, ll b){
    ll x, y; //a*x + b*y = gcd(a,b)
    if (b == 0) return {1, 0};
    auto p = extendedEuclid(b, a%b);
    x = p.second;
    y = p.first - (a/b)*x;
    if(a*x + b*y == -__gcd(a,b)) x=-x, y=-y;
    return {x, y};
}
pair<ii, ii> diophantine(ll a, ll b, ll r){
    //a*x+b*y=r where r is multiple of gcd(a,b);
    ll d = __gcd(a, b);
    a/=d; b/=d; r/=d;
    auto p = extendedEuclid(a, b);
    p.first*=r; p.second*=r;
    assert(a*p.first + b*p.second == r);
    return {p, {-b, a}}; //solutions: p+t*ans.second
}
```

6.2 arithmetic

```
inline int floor_log2(int n)
{ return n <= 1 ? 0 : 31 - __builtin_clz(n); }
inline int ceil_log2(int n)
{ return n <= 1 ? 0 : 32 - __builtin_clz(n - 1); }
inline ll floordiv(ll a, ll b) {return a/b-((a^b)<0&&a%b);}
inline ll ceildiv(ll a, ll b) {return a/b+((a^b)>=0&&a%b);}
```

6.3 berlekamp-massey-linear-recurrence

```
vector<int> BM(vector<int> x) {
    vector<int> ls, cur;
    int lf, ld;
    rep(i, x.size()) {
        ll t = 0;
        rep(j, cur.size()) t = (t+x[i-j-1]*(ll)cur[j])%MOD;
        if ((t - x[i]) % MOD == 0) continue;
        if (!cur.size()) {
            cur.resize(i + 1); lf = i; ld = (t-x[i]) % MOD;
            continue;
        }
        ll k = -(x[i] - t) * bin_exp(ld, MOD - 2) % MOD;
        vector<int> c(i - lf - 1); c.push_back(k);
        rep(j, ls.size()) c.push_back(-ls[j] * k % MOD);
        if (c.size() < cur.size()) c.resize(cur.size());
        rep(j, cur.size()) c[j] = (c[j] + cur[j]) % MOD;
        if (i - lf + ls.size() >= cur.size())
            ls = cur, lf = i, ld = (t - x[i]) % MOD;
        cur = c;
    }
    rep(i, cur.size()) cur[i] = (cur[i] % MOD + MOD) % MOD;
    return cur;
}
// Linear Recurrence
ll MOD = 998244353;
ll LOG = 60;
struct LinearRec{
  typedef vector<int> vi;
  int n; vi terms, trans; vector<vi> bin;
  vi add(vi &a, vi &b){
    vi res(n*2+1);
    rep(i,n+1) rep(j,n+1)
      res[i+j]=(res[i+j]*1LL+(ll)a[i]*b[j])%MOD;
    for(int i=2*n; i>n; --i){
      rep(j,n)
        res[i-1-j]=(res[i-1-j]*1LL+(ll)res[i]*trans[j])%MOD;
      res[i]=0;
    }
    res.erase(res.begin()+n+1,res.end());
    return res;
  }
  LinearRec(vi &terms, vi &trans):terms(terms),trans(trans){
    n=trans.size();vi a(n+1);a[1]=1;
    bin.push_back(a);
    repx(i,1,LOG)bin.push_back(add(bin[i-1],bin[i-1]));
  }
  int calc(ll k){
    vi a(n+1);a[0]=1;
    rep(i,LOG)if((k>>i)&1)a=add(a,bin[i]);
    int ret=0;
    rep(i,n)ret=((ll)ret+(ll)a[i+1]*terms[i])%MOD;
    ret = ret%MOD + MOD;
    return ret%MOD;
  }
};
```

6.4 crt

```
pair<ll, ll> solve_crt(const vector<pair<ll, ll>> &eqs) {
    ll a0 = eqs[0].first, p0 = eqs[0].second;
    repx(i, 1, eqs.size()) {
        ll a1 = eqs[i].first, p1 = eqs[i].second;
        ll k1, k0;
        ll d = ext_gcd(p1, p0, k1, k0);
        a0 -= a1;
        if (a0 % d != 0) return {-1, -1};
        p0 = p0 / d * p1;
        a0 = a0 / d * k1 % p0 * p1 % p0 + a1;
        a0 = (a0 % p0 + p0) % p0;
    }
    return {a0, p0};
}
```

6.5 discrete-log

```
// discrete logarithm log_a(b).
// solve b ^ x = a (mod M) for the smallest x.
```

```
// returns -1 if no solution is found.
//
// time: O(sqrt(M))
ll dlog(ll a, ll b, ll M) {
    ll k = 1, s = 0;
    while (true) {
        ll g = __gcd(b, M);
        if (g <= 1) break;
        if (a == k) return s;
        if (a % g != 0) return -1;
        a /= g, M /= g, s += 1, k = b / g * k % M;
    }
    ll N = sqrt(M) + 1;

    umap<ll, ll> r;
    rep(q, N + 1) {
        r[a] = q;
        a = a * b % M;
    }

    ll bN = binexp(b, N, M), bNp = k;
    repx(p, 1, N + 1) {
        bNp = bNp * bN % M;
        if (r.count(bNp)) return N * p - r[bNp] + s;
    }
    return -1;
}
```

6.6 fast-hadamard-transform

```
ll c1[MAXN+9],c2[MAXN+9];//MAXN must be power of 2!
void fht(ll* p, int n, bool inv){
    for(int l=1;2*l<=n;l*=2)for(int i=0;i<n;i+=2*l)fore(j,0,l
      ){
        ll u=p[i+j],v=p[i+l+j];
        if(!inv)p[i+j]=u+v,p[i+l+j]=u-v; // XOR
        else p[i+j]=(u+v)/2,p[i+l+j]=(u-v)/2;
        //if(!inv)p[i+j]=v,p[i+l+j]=u+v; // AND
        //else p[i+j]=-u+v,p[i+l+j]=u;
        //if(!inv)p[i+j]=u+v,p[i+l+j]=u; // OR
        //else p[i+j]=v,p[i+l+j]=u-v;
    }
}
// like polynomial multiplication, but XORing exponents
// instead of adding them (also ANDing, ORing)
vector<ll> multiply(vector<ll>& p1, vector<ll>& p2){
    int n=1<<(32-__builtin_clz(max(SZ(p1),SZ(p2))-1));
    fore(i,0,n)c1[i]=0,c2[i]=0;
    fore(i,0,SZ(p1))c1[i]=p1[i];
    fore(i,0,SZ(p2))c2[i]=p2[i];
    fht(c1,n,false);fht(c2,n,false);
    fore(i,0,n)c1[i]*=c2[i];
    fht(c1,n,true);
    return vector<ll>(c1,c1+n);
}
```

6.7 fft

```
using cd = complex<double>;
const double PI = acos(-1);
// compute the DFT of a power-of-two-length sequence.
// if `inv` is true, computes the inverse DFT.
void fft(vector<cd> &a, bool inv) {
    int N = a.size(), k = 0, b;
    assert(N == 1 << __builtin_ctz(N));
    repx(i, 1, N) {
        for (b = N >> 1; k & b;) k ^= b, b >>= 1;
        if (i < (k ^= b)) swap(a[i], a[k]);
    }
    for (int l = 2; l <= N; l <<= 1) {
        double ang = 2 * PI / l * (inv ? -1 : 1);
        cd wl(cos(ang), sin(ang));
        for (int i = 0; i < N; i += l) {
            cd w = 1;
            rep(j, l / 2) {
                cd u = a[i + j], v = a[i + j + l / 2] * w;
                a[i + j] = u + v;
                a[i + j + l / 2] = u - v;
                w *= wl;
            }
        }
    }
    if (inv) rep(i, N) a[i] /= N;
}
const ll MOD = 998244353, ROOT = 15311432;
// const ll MOD = 2130706433, ROOT = 1791270792;
// const ll MOD = 922337203673733529711, ROOT =
    532077456549635698311;
void find_root_of_unity(ll M) {
    ll c = M - 1, k = 0;
    while (c % 2 == 0) c /= 2, k += 1;
    // find proper divisors of M - 1
    vector<ll> divs;
    for (ll d = 1; d < c; d++) {
        if (d * d > c) break;
        if (c % d == 0) rep(i, k + 1) divs.push_back(d << i);
    }
    rep(i, k) divs.push_back(c << i);
    // find any primitive root of M
    ll G = -1;
    repx(g, 2, M) {
        bool ok = true;
        for (ll d : divs) ok &= (binexp(g, d, M) != 1);
        if (ok) {
            G = g;
            break;
        }
    }
    assert(G != -1);
    ll w = binexp(G, c, M);
    cerr << "M = c * 2^k + 1" << endl;
    cerr << " M = " << M << endl;
    cerr << " c = " << c << endl;
    cerr << " k = " << k << endl;
    cerr << " w^(2^k) == 1" << endl;
    cerr << "   w = g^((M-1)/2^k) = g^c" << endl;
    cerr << "   g = " << G << endl;
    cerr << "   w = " << w << endl;
}
// compute the DFT of a power-of-two-length sequence, modulo
    a special prime
// number with an Nth root of unity, where N is the length
   of the sequence.
void ntt(vector<ll> &a, bool inv) {
    vector<ll> wn;
    for (ll p = ROOT; p != 1; p = p * p % MOD) wn.push_back(p
      );
    int N = a.size(), k = 0, b;
    assert(N == 1 << __builtin_ctz(N) && N <= 1 << wn.size())
      ;
    rep(i, N) a[i] = (a[i] % MOD + MOD) % MOD;
    repx(i, 1, N) {
        for (b = N >> 1; k & b;) k ^= b, b >>= 1;
        if (i < (k ^= b)) swap(a[i], a[k]);
    }
    for (int l = 2; l <= N; l <<= 1) {
        ll wl = wn[wn.size() - __builtin_ctz(l)];
        if (inv) wl = multinv(wl, MOD);
        for (int i = 0; i < N; i += l) {
            ll w = 1;
            repx(j, 0, l / 2) {
                ll u = a[i + j], v = a[i + j + l / 2] * w %
                  MOD;
                a[i + j] = (u + v) % MOD;
                a[i + j + l / 2] = (u - v + MOD) % MOD;
                w = w * wl % MOD;
            }
        }
```

```
    }
    ll q = multinv(N, MOD);
    if (inv) rep(i, N) a[i] = a[i] * q % MOD;
}
void convolve(vector<cd> &a, vector<cd> b, int n) {
    n = 1 << (32 - __builtin_clz(2 * n - 1));
    a.resize(n), b.resize(n);
    fft(a, false), fft(b, false);
    rep(i, n) a[i] *= b[i];
    fft(a, true);
}
```

6.8 gauss

```
const double EPS = 1e-9;
// solve a system of equations.
// complexity: O(min(N, M) * N * M)
// ‘a‘ is a list of rows
// the last value in each row is the result of the equation
// return values:
//  0 -> no solutions
//  1 -> unique solution, stored in ‘ans‘
// -1 -> infinitely many solutions, one of which is stored
    in ‘ans‘
// UNTESTED
int gauss(vector<vector<double>> a, vector<double> &ans) {
    int N = a.size(), M = a[0].size() - 1;
    vector<int> where(M, -1);
    for (int j = 0, i = 0; j < M && i < N; j++) {
        int sel = i;
        repx(k, i, N) if (abs(a[k][j]) > abs(a[sel][j])) sel
            = k;
        if (abs(a[sel][j]) < EPS) continue;
        repx(k, j, M + 1) swap(a[sel][k], a[i][k]);
        where[j] = i;
        rep(k, N) if (k != i) {
            double c = a[k][j] / a[i][j];
            repx(l, j, M + 1) a[k][l] -= a[i][l] * c;
        }
        i++;
    }
    ans.assign(M, 0);
    rep(i, M) if (where[i] != -1) ans[i] = a[where[i]][M] / a
        [where[i]][i];
    rep(i, N) {
        double sum = 0;
        rep(j, M) sum += ans[j] * a[i][j];
        if (abs(sum - a[i][M]) > EPS) return 0;
    }
    rep(i, M) if (where[i] == -1) return -1;
    return 1;
}
```

6.9 matrix

```
typedef vector<vector<double>> Mat;
Mat matmul(Mat l, Mat r) {
    int n = l.N, m = r.M, p = l.M; assert(l.M == r.N);
    Mat a(n, vector<double>(m)); // neutral
    rep(i, n) rep(j, m)
        rep(k, p) a[i][j] = a[i][j] + l[i][k] * r[k][j];
    return a;
}

double reduce(vector<vector<double>> &A) {
    int n = A.size(), m = A[0].size();
    int i = 0, j = 0; double r = 1.;
    while (i < n && j < m) {
        int l = i;
        repx(k, i+1, n) if(abs(A[k][j]) > abs(A[l][j])) l=k;
        if (abs(A[l][j]) < EPS) { j++; r = 0.; continue; }
        if (l != i) { r = -r; swap(A[i], A[l]); }
        r *= A[i][j];
        for (int k = m - 1; k >= j; k--) A[i][k] /= A[i][j];
        repx(k, 0, n) {
            if (k == i) continue;
            for(int l=m-1;l>=j;l--)A[k][l]-=A[k][j]*A[i][l];
        }
        i++, j++;
    }
    return r; // returns determinant
}
```

6.10 mobius

```
short mu[MAXN] = {0,1};
void mobius(){
    repx(i,1,MAXN)if(mu[i])for(int j=i+i;j<MAXN;j+=i)mu[j]-=
        mu[i];
}
```

6.11 multinv

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

// inverse exists if and only if a and M are coprime
// if M is prime: multinv(a, M) = (a**(M-2)) % M
ll multinv(ll a, ll M)
{ ll x, y; ext_gcd(a, M, x, y); return x; }

// all modular inverses from 1 to inv.size()-1
void multinv_all(vector<ll> &inv) {
    inv[1] = 1;
    repx(i, 2, inv.size())
        inv[i] = MOD - (MOD / i) * inv[MOD % i] % MOD;
}
```

6.12 polar-rho

```
ll mulmod(ll a, ll b, ll m) {
    ll r=a*b-(ll)((long double)a*b/m+.5)*m;
    return r<0?r+m:r;
}
bool is_prime_prob(ll n, int a){
    if(n==a)return true;
    ll s=0,d=n-1;
    while(d%2==0)s++,d/=2;
    ll x=expmod(a,d,n);
    if((x==1)||(x+1==n))return true;
    fore(_,0,s-1){
        x=mulmod(x,x,n);
        if(x==1)return false;
        if(x+1==n)return true;
    }
    return false;
}
bool rabin(ll n){ // true iff n is prime
    if(n==1)return false;
    int ar[]={2,3,5,7,11,13,17,19,23};
    fore(i,0,9)if(!is_prime_prob(n,ar[i]))return false;
    return true;
}
ll rho(ll n){
    if(!(n&1))return 2;
    ll x=2,y=2,d=1;
    ll c=rand()%n+1;
    while(d==1){
        x=(mulmod(x,x,n)+c)%n;
        fore(it,0,2) y=(mulmod(y,y,n)+c)%n;
```

```
        if(x>=y)d=__gcd(x-y,n);
        else d=__gcd(y-x,n);
    }
    return d==n?rho(n):d;
}
void fact(ll n, map<ll,int>& f){ //O (lg n)^3
    if(n==1)return;
    if(rabin(n)){f[n]++;return;}
    ll q=rho(n);fact(q,f);fact(n/q,f);
}
// optimized version: replace rho and fact with the
     following:
const int MAXP=1e6+1; // sieve size
int sv[MAXP]; // sieve
ll add(ll a, ll b, ll m){return (a+=b)<m?a:a-m;}
ll rho(ll n){
    static ll s[MAXP];
    while(1){
        ll x=rand()%n,y=x,c=rand()%n;
        ll *px=s,*py=s,v=0,p=1;
        while(1){
            *py++=y=add(mulmod(y,y,n),c,n);
            *py++=y=add(mulmod(y,y,n),c,n);
            if((x=*px++)==y)break;
            ll t=p; p=mulmod(p,abs(y-x),n);
            if(!p)return __gcd(t,n);
            if(++v==26){
                if((p=__gcd(p,n))>1&&p<n)return p;
                v=0;
            }
        }
        if(v&&(p=__gcd(p,n))>1&&p<n)return p;
    }
}
void init_sv(){ fore(i,2,MAXP)if(!sv[i])for(ll j=i;j<MAXP;j
     +=i)sv[j]=i; }
void fact(ll n,map<ll,int>&f){//call init_sv first!
    for(auto&& p:f)while(n%p.fst==0)p.snd++,n/=p.fst;
    if(n<MAXP)while(n>1)f[sv[n]]++,n/=sv[n];
    else if(rabin(n))f[n]++;
    else {ll q=rho(n);fact(q,f);fact(n/q,f);}
}
```

6.13 polynomials

```
typedef int tp; // type of polynomial
template<class T=tp>
struct poly { // poly<> : 1 variable, poly<poly<>>: 2
     variables, etc.
    vector<T> c;
    T& operator[](int k){return c[k];}
    poly(vector<T>& c):c(c){}
    poly(initializer_list<T> c):c(c){}
    poly(int k):c(k){}
    poly(){}
    poly operator+(poly<T> o);
    poly operator*(tp k);
    poly operator*(poly o);
    poly operator-(poly<T> o){return *this+(o*-1);}
    T operator()(tp v){
        T sum(0);
        for(int i=c.size()-1;i>=0;--i)sum=sum*v+c[i];
        return sum;
    }
};
// example: p(x,y)=2*x^2+3*x*y-y+4
// poly<poly<>> p={{4,-1},{0,3},{2}}
// printf("%d\n",p(2)(3)) // 27 (p(2,3))
set<tp> roots(poly<> p){ // only for integer polynomials
    set<tp> r;
    while(!p.c.empty()&&!p.c.back())p.c.pop_back();
    if(!p(0))r.insert(0);
    if(p.c.empty())return r;
    tp a0=0,an=abs(p[p.c.size()-1]);
    for(int k=0;!a0;a0=abs(p[k++]));
    vector<tp> ps,qs;
    fore(i,1,sqrt(a0)+1)if(a0%i==0)ps.pb(i),ps.pb(a0/i);
    fore(i,1,sqrt(an)+1)if(an%i==0)qs.pb(i),qs.pb(an/i);
    for(auto pt:ps)for(auto qt:qs)if(pt%qt==0){
        tp x=pt/qt;
        if(!p(x))r.insert(x);
        if(!p(-x))r.insert(-x);
    }
    return r;
}
pair<poly<>,tp> ruffini(poly<> p, tp r){ // returns pair (
     result,rem)
    int n=p.c.size()-1;
    vector<tp> b(n);
    b[n-1]=p[n];
    for(int k=n-2;k>=0;--k)b[k]=p[k+1]+r*b[k+1];
    return {poly<>(b),p[0]+r*b[0]};
}
// only for double polynomials
pair<poly<>,poly<> > polydiv(poly<> p, poly<> q){ // returns
     pair (result,rem)
    int n=p.c.size()-q.c.size()+1;
    vector<tp> b(n);
    for(int k=n-1;k>=0;--k){
        b[k]=p.c.back()/q.c.back();
        fore(i,0,q.c.size())p[i+k]-=b[k]*q[i];
        p.c.pop_back();
    }
    while(!p.c.empty()&&abs(p.c.back())<EPS)p.c.pop_back();
    return {poly<>(b),p};
}
// only for double polynomials
poly<> interpolate(vector<tp> x, vector<tp> y){
    poly<> q={1},S={0};
    for(tp a:x)q=poly<>({-a,1})*q;
    fore(i,0,x.size()){
        poly<> Li=ruffini(q,x[i]).fst;
        Li=Li*(1.0/Li(x[i])); // change for int polynomials
        S=S+Li*y[i];
    }
    return S;
}
```

6.14 primes

```
// counts the divisors of a positive integer in O(sqrt(n))
ll count_divisors(ll x) {
    ll divs = 1, i = 2;
    for (ll divs = 1, i = 2; x > 1; i++) {
        if (i * i > x) { divs *= 2; break; }
        for (ll d = divs; x % i == 0; x /= i) divs += d;
    }
    return divs;
}

// gets the prime factorization of a number in O(sqrt(n))
vector<pair<ll, int>> factorize(ll x) {
    vector<pair<ll, int>> f;
    for (ll k = 2; x > 1; k++) {
        if (k * k > x) { f.push_back({x, 1}); break; }
        int n = 0;
        while (x % k == 0) x /= k, n++;
        if (n > 0) f.push_back({k, n});
    }
    return f;
}

// iterate over all divisors of a number.
// divisor count upper bound: n^(1.07 / ln ln n)
template <class OP>
void divisors(ll x, OP op) {
    auto facts = factorize(x);
    vector<int> f(facts.size());
```

```
    while (true) {
        ll y = 1;
        rep(i, f.size()) rep(j, f[i]) y *= facts[i].first;
        op(y);

        int i;
        for (i = 0; i < f.size(); i++) {
            f[i] += 1;
            if (f[i] <= facts[i].second) break;
            f[i] = 0;
        }
        if (i == f.size()) break;
    }
}

// computes euler totative function phi(x), counting the
// amount of integers in [1, x] that are coprime with x.
// time: O(sqrt(x))
ll phi(ll x) {
    ll phi = 1, k = 2;
    for (; x > 1; k++) {
        if (k * k > x) { phi *= x - 1; break; }
        ll k1 = 1, k0 = 0;
        while (x % k == 0) x /= k, k0 = k1, k1 *= k;
        phi *= k1 - k0;
    }
    return phi;
}
```

6.15 simplex

```
/* 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: O(NM * \#pivots), where a pivot may be e.g. an edge
relaxation. O(2^n) in the general case.*/
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; vector<int> N, B; vvd D;
    LPSolver(const vvd &A,const vd &b,const vd &c) : m(b.size
        ()),n(c.size()),N(n+1),B(m),D(m+2,vd(n+2)){
        rep(i, m) rep(j, n) D[i][j] = A[i][j];
        rep(i, m) {
            B[i] = n + i; D[i][n] = -1; D[i][n + 1] = b[i];
        }
        rep(j, 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, m + 2) if (i != r && abs(D[i][s]) > eps) {
            T *b = D[i].data(), inv2 = b[s] * inv;
            repx(j, 0, n + 2) b[j] -= a[j] * inv2;
            b[s] = a[s] * inv2;
        }
        rep(j, n + 2) if (j != s) D[r][j] *= inv;
        rep(i, 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, n + 1) if (N[j] != -phase) ltj(D[x]);
            if (D[x][s] >= -eps) return true;
            int r = -1;
            rep(i, 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;
        repx(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, m) if (B[i] == -1) {
                int s = 0;
                repx(j, 1, n + 1) ltj(D[i]);
                pivot(i, s);
            }
        }
        bool ok = simplex(1);
        x = vd(n);
        rep(i, m) if (B[i] < n) x[B[i]] = D[i][n + 1];
        return ok ? D[m][n + 1] : inf;
    }
};
```

6.16 test-prime

```
// change to __int128 if checking numbers over 10^9
bool isprime(ll n) {
    if (n < 2 || n % 6 % 4 != 1) return n - 2 < 2;
    ll A[] = {2,325,9375,28178,450775,9780504,1795265022};
    ll s = __builtin_ctzll(n - 1), d = n >> s;
    for (int a : A) {
        ll p = binexp(a, d, n), i = s;
        while (p != 1 && p != n - 1 && a % n && i--) p = p *
            p % n;
        if (p != n - 1 && i != s) return 0;
    }
    return 1;
}
```

6.17 theorems

```
Burnside lemma
Tomemos imagenes x en X y operaciones (g: X -> X) en G.
Si #g es la cantidad de imagenes que son puntos fijos de g,
entonces la cantidad de objetos es `(sum_{g in G} #g) / |G|`
Es requisito que G tenga la operacion identidad, que toda
operacion tenga inversa y que todo par de operaciones tenga
su combinacion.

Rational root theorem
Las raices racionales de un polinomio de orden n con
coeficientes enteros A[i] son de la forma p / q, donde p y q
son coprimos, p es divisor de A[0] y q es divisor de A[n].
Notar que si A[0] = 0, cero es raiz, se puede dividir el
polinomio por x y aplica nuevamente el teorema.

Petersens theorem
```

```
Every cubic and bridgeless graph has a perfect matching.

Number of divisors for powers of 10
(0,1) (1,4) (2,12) (3,32) (4,64) (5,128) (6,240) (7,448)
(8,768) (9,1344) (10,2304) (11,4032) (12,6720) (13,10752)
(14,17280) (15,26880) (16,41472) (17,64512) (18,103680)

Kirchoff Theorem: Sea A la matriz de adyacencia del multi-
grafo (A[u][v] indica la cantidad de aristas entre u y v)
Sea D una matriz diagonal tal que D[v][v] es igual al grado
de v (considerando auto aristas y multi aristas). Sea
L = A - D. Todos los cofactores de L son iguales y equivalen
a la cantidad de Spanning Trees del grafo. Un cofactor (i,j)
de L es la multiplicacin de (-1)^{i + j} con el determinant
de la matriz al quitar la fila i y la columna j

Prufer Code: Dado un rbol con los nodos indexados: busca la
hoja de menor ndice, brrala y anota el ndice del nodo al
que estaba conectado. Repite el paso anterior n-2 veces. Lo
anterior muestra una biyeccin entre los arreglos de tamao
n-2 con elementos en [1, n] y los rboles de n nodos, por lo
que hay n^{n-2} spanning trees en un grafo completo.
Corolario: Si tenemos k componentes de tamaos s1,s2,...,sk
entonces podemos hacerlos conexos agregando k-1 aristas
entre nodos de s1*s2*...*sk*n^{k-2} formas

Combinatoria
Catalan: C_{n+1} = sum(C_i*C_{n-i} for i \in [0, n])
Catalan: C_n = \frac{1}{n+1}*\binom{2n, n}
Sea C_n^k las formas de poner n+k pares de parntesis, con
los primeros k parntesis abiertos (esto es, hay 2n + 2k
 carcteres ), se tiene que
C_n^k = (2n+k-1)*(2n+k)/(n*(n+k+1)) * C_{n-1}^k
Sea D_n el nmero de permutaciones sin puntos fijos, entoces
D_n = (n-1)*(D_{n-1} + D_{n-2}), D_0 = 1, D_1 = 0
```

6.18 tonelli-shanks

```
ll legendre(ll a, ll p) {
    if (a % p == 0) return 0; if (p == 2) return 1;
    return binexp(a, (p - 1) / 2, p);
}
// sqrt(n) mod p (p must be a prime)
// rnd(a, b) return a random number in [a, b]
ll tonelli_shanks(ll n, ll p) {
    if (n == 0) return 0;
    if (legendre(n, p) != 1) return -1; // no existe
    if (p == 2) return 1;
    ll s = __builtin_ctzll(p - 1);
    ll q = (p - 1LL) >> s, z = rnd(1, p - 1);
    if (s == 1) return binexp(n, (p + 1) / 4LL, p);
    while (legendre(z, p) != p - 1) z = rnd(1, p - 1);
    ll c = binexp(z, q, p), r = binexp(n, (q + 1) / 2, p);
    ll t = binexp(n, q, p), m = s;
    while (t != 1) {
        ll i = 1, ts = (t * t) % p;
        while (ts != 1) i++, ts = (ts * ts) % p;
        ll b = c;
        repx(_, 0, m - i - 1) b = (b * b) % p;
        r = r*b%p; c = b*b%p; t = t*c%p; m = i;
    }
    return r;
}
```

7 strings

7.1 Manacher

```
// odd[i]: length of longest palindrome centered at i
// even[i]: ...longest palindrome centered between i and i+1
void manacher(string &s,vector<int> &odd,vector<int> &even){
    string t = "$#";
    for(char c: s) t += c + string("#");
    t += "^";
    int n = t.size();
    vector<int> p(n);
    int l = 1, r = 1;
    repx(i, 1, n-1) {
        p[i] = max(0, min(r - i, p[l + (r - i)]));
        while(t[i - p[i]] == t[i + p[i]]) p[i]++;
        if(i + p[i] > r) l = i - p[i], r = i + p[i];
    }
    repx(i, 2, n-2) {
        if(i%2) even.push_back(p[i]-1);
        else odd.push_back(p[i]-1);
    }
}
```

7.2 aho-corasick

```
struct Vertex {
    int next[26], go[26];
    int p, link = -1, exit = -1, cnt = -1;
    vector<int> leaf;
    char pch;
    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        rep(i, 26) next[i] = -1, go[i] = -1;
    }
};
vector<Vertex> t(1);
void add(string &s, int id) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].leaf.push_back(id);
}
int go(int v, char ch);
int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0) t[v].link = 0;
        else t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}
int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1) t[v].go[c] = t[v].next[c];
        else t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}
int next_match(int v){ // Optional
    if(t[v].exit == -1){
        if(t[get_link(v)].leaf.size())t[v].exit=get_link(v);
        else t[v].exit = v==0 ? 0 : next_match(get_link(v));
    }
    return t[v].exit;
}
int cnt_matches(int v){ // Optional
    if(t[v].cnt == -1)
        t[v].cnt = v == 0 ? 0 : t[v].leaf.size() +
            cnt_matches(get_link(v));
    return t[v].cnt;
}
```

7.3 debruijn-sequence

```
// Given alphabet [0,k) constructs a cyclic string of length
// k^n that contains every length n string as substr.
vector<int> deBruijnSeq(int k, int n) { // Recursive FKM
    if (k == 1) return {0};
    vector<int> seq, aux(n+1);
    function<void(int,int)> gen = [&](int t, int p) {
        if (t > n) { // +lyndon word of len p
            if (n%p == 0) repx(i,1,p+1) seq.pb(aux[i]);
        } else {
            aux[t] = aux[t-p]; gen(t+1,p);
            while (++aux[t] < k) gen(t+1,t);
        }
    };
    gen(1,1); return seq;
}
```

7.4 hash

```
const int K = 2;
struct Hash{
    const ll MOD[K] = {999727999, 1070777777};
    const ll P = 1777771;
    vector<ll> h[K], p[K];
    Hash(string &s){
        int n = s.size();
        rep(k, K){
            h[k].resize(n+1, 0);
            p[k].resize(n+1, 1);
            repx(i, 1, n+1){
                h[k][i] = (h[k][i-1]*P + s[i-1]) % MOD[k];
                p[k][i] = (p[k][i-1]*P) % MOD[k];
            }
        }
    }
    vector<ll> get(int i, int j){
        vector<ll> r(K);
        rep(k, K){
            r[k] = (h[k][j] - h[k][i]*p[k][j-i]) % MOD[k];
            r[k] = (r[k] + MOD[k]) % MOD[k];
        } return r;
    }
};
```

7.5 palindromic-tree

```
struct Node {  // (*) = Optional
    int len;    // length of substring
    int to[26]; // insertion edge for all characters a-z
    int link;   // maximun palindromic suffix
    int i;      // (*) start index of current Node
    int cnt;    // (*) # of occurrences of this substring
    Node(int len, int link=0, int i=0, int cnt=1): len(len),
    link(link), i(i), cnt(cnt) {memset(to, 0, sizeof(to));}
}; struct EerTree {  // Palindromic Tree
    vector<Node> t; // tree (max size of tree is n+2)
    int last;      // current node
    EerTree(string &s) : last(0) {
        t.emplace_back(-1); t.emplace_back(0); // root 1 & 2
        rep(i, s.size()) add(i, s); // construct tree
        for(int i = t.size()-1; i > 1; i--)
            t[t[i].link].cnt += t[i].cnt;
    }
    void add(int i, string &s){      // vangrind warning:
        int p=last, c=s[i]-'a';      // i-t[p].len-1 = -1
        while(s[i-t[p].len-1] != s[i]) p = t[p].link;
        if(t[p].to[c]){ last = t[p].to[c]; t[last].cnt++; }
        else{
            int q = t[p].link;
            while(s[i-t[q].len-1] != s[i]) q = t[q].link;
            q = max(1, t[q].to[c]);
            last = t[p].to[c] = t.size();
            t.emplace_back(t[p].len + 2, q, i-t[p].len-1);
        }
    }
};
void main(){
    string s = "abcbab"; EerTree pt(s); // build EerTree
    repx(i, 2, pt.t.size()){// list all distinct palindromes
        repx(j,pt.t[i].i,pt.t[i].i+pt.t[i].len)cout << s[j];
        cout << " " << pt.t[i].cnt << endl;
    }
}
```

7.6 prefix-function

```
vector<int> prefix_function(string s) {
    int n = s.size();
    vector<int> pi(n);
    repx(i, 1, n) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
vector<vector<int>> aut;
void compute_automaton(string s) {
    s += '#';
    int n = s.size();
    vector<int> pi = prefix_function(s);
    aut.assign(n, vector<int>(26));
    rep(i, n) {
        rep(c, 26) {
            int j = i;
            while (j > 0 && 'a' + c != s[j])
                j = pi[j-1];
            if ('a' + c == s[j])
                j++;
            aut[i][c] = j;
        }
    }
}
// k = n - pi[n - 1]; if k divides n, then the string can be
// aprtitioned into blocks of length k otherwise there is no
// effective compression and the answer is n.
```

7.7 suffix-array

```
// build the suffix array
// suffixes are sorted, with each suffix represented by its
// starting position
vector<int> suffixarray(const string &s) {
    int N = s.size() + 1;//optional: include terminating NUL
    vector<int> p(N), p2(N), c(N), c2(N), cnt(256);
    rep(i, N) cnt[s[i]] += 1;
    repx(b, 1, 256) cnt[b] += cnt[b - 1];
    rep(i, N) p[--cnt[s[i]]] = i;
    repx(i, 1, N) c[p[i]] = c[p[i - 1]] + (s[p[i]] != s[p[i -
        1]]);
    for (int k = 1; k < N; k <<= 1) {
        int C = c[p[N - 1]] + 1;
        cnt.assign(C + 1, 0);
        for (int &pi : p) pi = (pi - k + N) % N;
        for (int cl : c) cnt[cl + 1] += 1;
        rep(i, C) cnt[i + 1] += cnt[i];
        rep(i, N) p2[cnt[c[p[i]]]++] = p[i];
        c2[p2[0]] = 0;
        repx(i, 1, N) c2[p2[i]] =
            c2[p2[i - 1]] + (c[p2[i]] != c[p2[i - 1]] ||
                            c[(p2[i] + k) % N] != c[(p2[i - 1]
                                + k) % N]);
```

```
        swap(c, c2), swap(p, p2);
    }
    p.erase(p.begin()); // optional: erase terminating NUL
    return p;
}
// build the lcp
// `lcp[i]` represents the length of the longest common
// prefix between suffix i and suffix i+1 in the suffix
//array `p`. the last element of `lcp` is zero by convention
vector<int> makelcp(const string &s, const vector<int> &p) {
    int N = p.size(), k = 0;
    vector<int> r(N), lcp(N);
    rep(i, N) r[p[i]] = i;
    rep(i, N) {
        if (r[i] + 1 >= N) { k = 0; continue; }
        int j = p[r[i] + 1];
        while (i + k < N && j + k < N && s[i + k] == s[j + k
            ]) k += 1;
        lcp[r[i]] = k;
        if (k) k -= 1;
    }
    return lcp;
}
// lexicographically compare the suffixes starting from `i`
// and `j`, considering only up to `K` characters.
// `r` is the inverse suffix array, mapping suffix offsets
// to indices. requires an LCP sparse table.
int lcp_cmp(vector<int> &r, Sparse<int> &lcp, int i, int j,
     int K) {
    if (i == j) return 0;
    int ii = r[i], jj = r[j];
    int l = lcp.query(min(ii, jj), max(ii, jj));
    if (l >= K) return 0;
    return ii < jj ? -1 : 1;
}
```

7.8 suffix-automaton

```
struct State {int len, link; map<char,int> next; };
State st[2*MAXN]; int sz, last;          // clear next!!
void sa_init(){ last=st[0].len=0; sz=1; st[0].link=-1; }
void sa_extend(char c){// total build O(n log alphabet_size)
    int k = sz++, p; st[k].len = st[last].len + 1;
    for(p=last; p!=-1 && !st[p].next.count(c); p=st[p].link)
        st[p].next[c] = k;
    if(p == -1) st[k].link = 0;
    else {
        int q = st[p].next[c];
        if(st[p].len + 1 == st[q].len) st[k].link = q;
        else {
            int w = sz++; st[w].len = st[p].len + 1;
            st[w].next=st[q].next; st[w].link=st[q].link;
            for(; p!=-1 && st[p].next[c]==q; p=st[p].link)
                st[p].next[c] = w;
            st[q].link=st[k].link = w;
        }
    }
    last = k;
} // # states <= 2n-1 && transitions <= 3n-4 (for n > 2)
// Follow link from `last` to 0, nodes on path are terminal
// # matches = # paths from state to a terminal node
// # substrings = # paths from 0 to any node
// # substrings = sum of (len - len(link)) for all nodes
```

7.9 z-function

```
// i-th element is equal to the greatest number of
// characters starting from the position i that coincide
// with the first characters of s
vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for(int i = 1; i < n; i++) {
        if(i < r) z[i] = min(r - i, z[i - l]);
        while(i + z[i] < n && s[z[i]] == s[i + z[i]])z[i]++;
        if(i + z[i] > r) {
            l = i;
            r = i + z[i];
        }
    }
    return z;
}
```