Team notebook

Universidad Mayor de San Simón - Club de Frontón 2880: Oliver Pozo, Rodrigo Salguero, Mateo Valdivia

April 5, 2024

Contents

1 DataStructures

1.1 lazy segment tree

```
int ar[tam], t[4 * tam], l[4 * tam];
void push(int b, int e, int node) {
  if(l[node]) {
    t[node] += l[node];
    if(b < e)
      l[node * 2 + 1] += l[node], l[node * 2 + 2] += l[node];
    l[node] = 0; } }
void update(int b, int e, int node, int i, int j, int val) {
  if(b > e) return; push(b, e, node);
  if(e < i || b > j) return;
  if(b >= i && e <= j) {
    l[node] += val; push(b, e, node); return; }
  index; update(b, mid, l, i, j, val);
  update(mid + 1, e, r, i, j, val);
  t[node] = max(t[l], t[r]); }
```

1.2 link-cut

```
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;}
```

1.3 merge sort tree

```
vi t[4*tam]; int ar[tam];
void build(int node, int b, int e) {
  if (b == e) { t[node].pb(ar[b]); return; }
  build(2*node, b, (b+e)/2);
  build(2*node+1, (b+e)/2+1, e);
  merge(t[2*node].begin(), t[2*node].end(),
    t[2*node+1].begin(), t[2*node+1].end(),
    back_inserter(t[node])); }
int query(int node, int b, int e, int i, int j, int x, int y) {
  if (j < b || i > e) return 0;
  if (i <= b && j >= e)
  return upper_bound(t[node].begin(), t[node].end(), y) -
      lower_bound(t[node].begin(), t[node].end(), x);
  return query(2*node, b, (b+e)/2, i, j, x, y) +
     query(2*node+1, (b+e)/2+1, e, i, j, x, y); }
```

1.4 segment-tree-iterativo

```
int t[2 * tam];
void build() { // build the tree
  for (int i = n - 1; i > 0; --i) t[i] = t[i<<1] + t[i<<1|1]; }
void modify(int p, int value) { // set value at position p
  for (t[p += n] = value; p > 1; p >>= 1) t[p>>1] = t[p] + t[p^1]; }
int query(int l, int r) { // sum on interval [l, r)
  int res = 0;
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
  if (l&1) res += t[l++];
  if (r&1) res += t[--r]; }
  return res; }
```

1.5 Sparse Table

```
int ar[tam], table[logTam][tam];
void inispar() {
  fore(i, 0, n) table[0][i] = ar[i];
  for(int k = 0; (1 << k) < n; k++)
    for(int i = 0; i + (1 << k) < n; i++)
      table[k + 1][i] = min(table[k][i], table[k][i + (1 << k)]); }
int query(int b, int e) {
  int lev = 31 - __builtin_clz(e - b + 1);
  return min(table[lev][b], table[lev][e - (1 << lev) + 1]); }
```

1.6 treap

```
struct item {
  int key, pri, siz;
  item *l, *r; item() {}
  item(int key) : key(key), siz(1), pri(rand()), l(0), r(0) {} };
typedef item* pitem;
int sz(pitem t) {
  return (t?t->siz:0); }
void up_sz(pitem t) {
  if(t) t->siz = sz(t->l) + 1 + sz(t->r); }
void split(pitem t, pitem &l, pitem &r, int val) {
  if(!t) r = l = NULL;
  else if(t->key < val) split(t->r, t->r, r, val), l = t;
  else split(t->l, l, t->l, val), r = t; up_sz(t); }
void merge(pitem &t, pitem l, pitem r) {
```

```
  if(!l || !r) t=(l?l:r);
  else if(l->pri >= r->pri) merge(l->r, l->r, r), t = l;
  else merge(r->l, l, r->l),t=r; up_sz(t); }
```

2 DP

2.1 ConvexHull

```
// kx+m, and query maximum values at points x.
#pragma once
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; } };
struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) { x->p = inf; return false; }
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p; }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y)); }
  ll query(ll x) {
    assert(!empty()); auto l = *lower_bound(x);
    return l.k * x + l.m; } };
```

2.2 divide-and-conquer

```
//DP[i][j]=min(DP[i-1][k]+C[k][j]); K[i][j]<=K[i][j+1]
ll lastDP[tam], DP[tam]; int C[tam][tam];
void DC(int b, int e, int KL, int KR) {
  int mid = (b + e) / 2;
  pair<ll, int> best = mp(-1, KL);
  for (int k = KL; k < min(mid, KR+1); k++)
  best = max( best, mp(lastDP[k] + C[k+1][mid], k) );
  DP[mid] = best.first; int K = best.second;
  if (b <= mid-1) DC(b, mid-1, KL, K);
  if (mid+1 <= e) DC(mid+1, e, K, KR); }
```

2.3 knuth

```
/*Basado en: Minimo costo de cortar una barra en K lugares pre-definidos
Costo de corte: largo de la barra que se esta cortando
A[i] -> posicion del i-esimo corte, A[0] = 0, A[n-1] = Largo total
DP[i][j] = min( DP[i][k] + DP[k][j] ) + C[i][j],
para i <= k <= j donde C[i][j] es el largo de la barra A[i] <-> A[j]
K[i][j-1] <= K[i][j] <= K[i+1][j]*/
fore(sz, 0, n) {
  for (int i = 0; i + sz < n; i++) {
  int j = i+sz; // CASOS BASE
  if (sz <= 1) { // Barra inexistente o con cero cortes en medio
    DP[i][j] = 0; continue; }
  if (sz == 2) { // Barra con un solo corte posible en medio
    K[i][j] = i+1; DP[i][j] = C[i][j]; continue; }
  int KL = K[i][j-1]; int KR = K[i+1][j]; DP[i][j] = INF;
  for (int k = KL; k <= KR; k++) {
    int newVal = DP[i][k] + DP[k][j] + C[i][j];
    if (newVal < DP[i][j]) {
    K[i][j] = k; DP[i][j] = newVal; } } } }
```

2.4 lis

```
for (int i = 0; i < n; ++i) {
  // increasing: lower_bound; non-decreasing: upper_bound
  int j = lower_bound(dp, dp + lis, v[i]) - dp;
  dp[j] = min(dp[j], v[i]); lis = max(lis, j + 1); }
```

2.5 matrix-fast-pow

```
// kth term of linear recurrence
// of size m a_i = sum(a_(i-j)*p_j)
```

```
// f(x) = x^m - sum(x^(m-j)*p_j)
// g(x^k) = g(x^k mod f)
typedef vector<vector<ll> > Matrix;
Matrix ones(int n) {
  Matrix r(n,vector<ll>(n)); fore(i,0,n)r[i][i]=1; return r; }
Matrix operator*(Matrix &a, Matrix &b) {
  int n=SZ(a),m=SZ(b[0]),z=SZ(a[0]); Matrix r(n,vector<ll>(m));
  fore(i,0,n)fore(j,0,m)fore(k,0,z)
  r[i][j]+=a[i][k]*b[k][j],r[i][j]%=mod; return r; }
Matrix be(Matrix b, ll e) {
  Matrix r=ones(SZ(b));
  while(e){if(e&1LL)r=r*b;b=b*b;e/=2;} return r; }
```

2.6 SOS

```
//iterative version
for(int mask = 0; mask < (1<<N); ++mask){
  dp[mask][-1] = A[mask]; //handle base case separately (leaf states)
  for(int i = 0;i < N; ++i){
  if(mask & (1<<i)) dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
  else dp[mask][i] = dp[mask][i-1]; }
  F[mask] = dp[mask][N-1]; }
//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i) F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
  if(mask & (1<<i)) F[mask] += F[mask^(1<<i)]; }
```

3 Flows

3.1 Algoritmo hungaro de asignacion

```
int n, m; int a[n+1][m+1]; // matriz de costos, 1-index
vector<int> u(n+1), v(m+1), p(m+1), way(m+1);
for (int i = 1; i <= n; ++i) {
  p[0] = i; int j0 = 0;
  vector<int> minv(m + 1, INF); vector<char> used(m + 1, false);
  do {
    used[j0] = true; int i0 = p[j0], delta = INF, j1;
    for (int j = 1; j <= m; ++j)
      if ( !used[j] ) {
        int cur = a[i0][j] - u[i0] - v[j];
        if ( cur < minv[j] ) minv[j] = cur, way[j] = j0;
        if ( minv[j] < delta ) delta = minv[j], j1 = j; }
    for (int j = 0; j <= m; ++j)
      if ( used[j] ) u[ p[j] ] += delta, v[j] -= delta;
      else minv[j] -= delta;
    j0 = j1;
  } while ( p[j0] != 0 );
  do {
    int j1 = way[j0]; p[j0] = p[j1]; j0 = j1;
  } while ( j0 ); }
vector<int> ans(n+1); // Para cada fila, en que columna esta el resultado.
for (int j = 1; j <= m; ++j) ans[ p[j] ] = j; int cost = -v[0];
```

3.2 max flow

```
struct Dinitz{
  const int INF = 1e9 + 7; Dinitz(){}
  Dinitz(int n, int s, int t) {init(n, s, t);}
  void init(int n, int s, int t) {
    S = s, T = t; nodes = n;
    G.clear(), G.resize(n); Q.resize(n); }
  struct flowEdge { int to, rev, f, cap; };
  vector<vector<flowEdge> > G;
  void addEdge(int st, int en, int cap) {
    flowEdge A = {en, (int)G[en].size(), 0, cap};
    flowEdge B = {st, (int)G[st].size(), 0, 0};
    G[st].pb(A); G[en].pb(B); }
  int nodes, S, T; vi work, lvl, Q;
  bool bfs() {
    int qt = 0; Q[qt++] = S;
    lvl.assign(nodes, -1); lvl[S] = 0;
    for (int qh = 0; qh < qt; qh++) {
      int v = Q[qh];
      for (flowEdge &e : G[v]) {
        int u = e.to;
        if (e.cap <= e.f || lvl[u] != -1) continue;
        lvl[u] = lvl[v] + 1; Q[qt++] = u; } }
    return lvl[T] != -1; }
  int dfs(int v, int f) {
    if (v == T || f == 0) return f;
    for (int &i = work[v]; i < G[v].size(); i++) {
      flowEdge &e = G[v][i]; int u = e.to;
```

```
      if (e.cap <= e.f || lvl[u] != lvl[v] + 1) continue;
      int df = dfs(u, min(f, e.cap - e.f));
      if (df) {
        e.f += df; G[u][e.rev].f -= df; return df; } }
    return 0; }
  int maxFlow() {
    int flow = 0;
    while (bfs()) {
      work.assign(nodes, 0);
      while (true) {
        int df = dfs(S, INF); if (df == 0) break;
        flow += df; } }
    return flow; }
};
```

3.3 min cost flow

```
// O(min(E^2 V ^2,   EVFLOW  ))
struct CheapDinitz{
  const int INF = 1e9 + 7; CheapDinitz() {}
  CheapDinitz(int n, int s, int t) {init(n, s, t);}
  int nodes, S, T; vi dist, pot, curFlow, prevNode, prevEdge, Q, inQue;
  struct flowEdge{ int to, rev, flow, cap, cost; };
  vector<vector<flowEdge>> G;
  void init(int n, int s, int t) {
    nodes = n, S = s, T = t;
    curFlow.assign(n, 0), prevNode.assign(n, 0), prevEdge.assign(n, 0);
    Q.assign(n, 0), inQue.assign(n, 0); G.clear(); G.resize(n); }
  void addEdge(int s, int t, int cap, int cost) {
    flowEdge a = {t, (int)G[t].size(), 0, cap, cost};
    flowEdge b = {s, (int)G[s].size(), 0, 0, -cost};
    G[s].pb(a); G[t].pb(b); }
  void bellmanFord() {
    pot.assign(nodes, INF); pot[S] = 0; int qt = 0; Q[qt++] = S;
    for (int qh = 0; (qh - qt) % nodes != 0; qh++) {
      int u = Q[qh % nodes]; inQue[u] = 0;
      for (int i = 0; i < (int)G[u].size(); i++) {
        flowEdge &e = G[u][i];
        if (e.cap <= e.flow) continue;
        int v = e.to; int newDist = pot[u] + e.cost;
        if (pot[v] > newDist) {
          pot[v] = newDist;
          if (!inQue[v]) { Q[qt++ % nodes] = v; inQue[v] = 1; }
        } } } }
  ii MinCostFlow() {
    bellmanFord(); int flow = 0; int flowCost = 0;
    while (true) {
      set<ii> s; s.insert({0, S}); dist.assign(nodes, INF);
      dist[S] = 0; curFlow[S] = INF;
      while (s.size() > 0) {
        int u = s.begin() -> s; int actDist = s.begin() -> f;
        s.erase(s.begin());
        if (actDist > dist[u]) continue;
        for (int i = 0; i < (int)G[u].size(); i++) {
          flowEdge &e = G[u][i]; int v = e.to;
          if (e.cap <= e.flow) continue;
          int newDist = actDist + e.cost + pot[u] - pot[v];
          if (newDist < dist[v]) {
            dist[v] = newDist; s.insert({newDist, v});
            prevNode[v] = u; prevEdge[v] = i;
            curFlow[v] = min(curFlow[u], e.cap - e.flow);
          } } }
      if (dist[T] == INF) break;
      for (int i = 0; i < nodes; i++)
        pot[i] += dist[i];
      int df = curFlow[T]; flow += df;
      for (int v = T; v != S; v = prevNode[v]) {
        flowEdge &e = G[prevNode[v]][prevEdge[v]];
        e.flow += df; G[v][e.rev].flow -= df;
        flowCost += df * e.cost; } }
    return {flow, flowCost}; }
};
```

3.4 simplex

```
vector<int> X,Y; vector<vector<double> > A;
vector<double> b,c; double z; int n, m;
void pivot(int x,int y){
  swap(X[y],Y[x]); b[x]/=A[x][y];
  fore(i,0,m)if(i!=y)A[x][i]/=A[x][y]; A[x][y]=1/A[x][y];
  fore(i,0,n)if(i!=x&&abs(A[i][y])>EPS){
  b[i]-=A[i][y]*b[x];
  fore(j,0,m)if(j!=y)A[i][j]-=A[i][y]*A[x][j];
  A[i][y]=-A[i][y]*A[x][y]; }
  z+=c[y]*b[x]; fore(i,0,m)if(i!=y)c[i]-=c[y]*A[x][i];
  c[y]=-c[y]*A[x][y]; }
```

```
pair<double,vector<double> > simplex( // maximize c^T x s.t. Ax<=b, x>=0
  vector<vector<double> > _A, vector<double> _b, vector<double> _c){
  // returns pair (maximum value, solution vector)
  A=_A;b=_b;c=_c; n=b.size();m=c.size();z=0.;
  X=vector<int>(m);Y=vector<int>(n);
  fore(i,0,m)X[i]=i; fore(i,0,n)Y[i]=i+m;
  while(1){
  int x=-1,y=-1; double mn=-EPS;
  fore(i,0,n)if(b[i]<mn)mn=b[i],x=i; if(x<0)break;
  fore(i,0,m)if(A[x][i]<-EPS){y=i;break;}
  assert(y>=0); // no solution to Ax<=b
  pivot(x,y); }
  while(1){
  double mx=EPS; int x=-1,y=-1;
  fore(i,0,m)if(c[i]>mx)mx=c[i],y=i; if(y<0)break; double mn=1e200;
  fore(i,0,n)if(A[i][y]>EPS&&b[i]/A[i][y]<mn)mn=b[i]/A[i][y],x=i;
  assert(x>=0); // c^T x is unbounded
  pivot(x,y); }
  vector<double> r(m); fore(i,0,n)if(Y[i]<m)r[Y[i]]=b[i];
  return mp(z,r); }
```

4 Geometry

4.1 centroid

```
// calcula el centro de masa de un poligono antihorario
point cen(vector<point> p) {
  double x = 0, y = 0, area = 0, ax; int n = p.size()-1;
  fore(i, 0, n) {
    ax = (p[i] ^ p[i+1]) / 2; area += ax;
    x += ax * (p[i].x + p[i+1].x) / 3;
    y += ax * (p[i].y + p[i+1].y) / 3; }
  return point(x / area, y / area); }
```

4.2 chull

```
// devuelve horario
vector<point> hull(vector<point> p) {
  int n = p.size(); vector<point> h; sort(all(p));
  fore(i, 0, n) {
  while(h.size() >= 2 && p[i].left(h[sz(h) - 2], h.back())) h.pop_back();
  h.push_back(p[i]); } h.pop_back(); int k = h.size();
  for(int i = n-1; i > -1; i--) {
  while(h.size() >= k + 2 && p[i].left(h[sz(h) - 2], h.back()))
      h.pop_back();
  h.pb(p[i]); }
  h.pop_back(); return h; }
```

4.3 circle2ptsrad

```
bool circle2PtsRad(point a, point b, double r, point &c) {//dados 2
    puntos y un radio
  double det = (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y);
  det = r * r / det - 0.25; if(det < 0.0) return false;
  det = sqrt(det); c.x = (a.x + b.x) * 0.5 + (b.y-a.y) * det;
  c.y = (a.y + b.y) * 0.5 + (a.x-b.x) * det; return true; }
```

4.4 halfplane

```
const double DINF=1e100;
struct pt {
  double x,y;
  pt(double x, double y):x(x),y(y){} pt(){}
  double norm2(){return *this**this;}
  double norm(){return sqrt(norm2());}
  bool operator==(pt p){return abs(x-p.x)<=EPS&&abs(y-p.y)<=EPS;}
  pt operator+(pt p){return pt(x+p.x,y+p.y);}
  pt operator-(pt p){return pt(x-p.x,y-p.y);}
  pt operator*(double t){return pt(x*t,y*t);}
  pt operator/(double t){return pt(x/t,y/t);}
  double operator*(pt p){return x*p.x+y*p.y;}
  double angle(pt p){ // redefine acos for values out of range
  return acos(*this*p/(norm()*p.norm()));}
  pt unit(){return *this/norm();}
  double operator%(pt p){return x*p.y-y*p.x;}
  bool operator<(pt p)const{ // for convex hull
  return x<p.x-EPS||(abs(x-p.x)<=EPS&&y<p.y-EPS);}
  bool left(pt p, pt q){ // is it to the left of directed line pq?
  return (q-p)%(*this-p)>EPS;}
  pt rot(pt r){return pt(*this%r,*this*r);}
  pt rot(double a){return rot(pt(sin(a),cos(a)));} };
```

```
pt ccw90(1,0); pt cw90(-1,0); int sgn2(double x){return x<0?-1:1;}
struct ln {
  pt p,pq; ln(pt p, pt q):p(p),pq(q-p){} ln(){}
  bool has(pt r){return dist(r)<=EPS;}
  bool seghas(pt r){return has(r)&&(r-p)*(r-(p+pq))<=EPS;}
  bool operator/(ln l){return abs(pq.unit()%l.pq.unit())<=EPS;}
  bool operator==(ln l){return *this/l&&has(l.p);}
  pt operator^(ln l){ // intersection
  if(*this/l)return pt(DINF,DINF); pt r=l.p+l.pq*((p-l.p)%pq/(l.pq%pq));
  return r; }
  double angle(ln l){return pq.angle(l.pq);}
  int side(pt r){return has(r)?0:sgn2(pq%(r-p));}
  pt proj(pt r){return p+pq*((r-p)*pq/pq.norm2());}
  pt ref(pt r){return proj(r)*2-r;}
  double dist(pt r){return (r-proj(r)).norm();}
  ln rot(auto a){return ln(p,p+pq.rot(a));} };
ln bisector(ln l, ln m){ // angle bisector
  pt p=l^m; return ln(p,p+l.pq.unit()+m.pq.unit()); }
ln bisector(pt p, pt q){ return ln((p+q)*.5,p).rot(ccw90); }
// polygon intersecting left side of halfplanes
struct halfplane:public ln{
  double angle; halfplane(){}
  halfplane(pt a,pt b){p=a; pq=b-a; angle=atan2(pq.y,pq.x);}
  bool operator<(halfplane b)const{return angle<b.angle;}
  bool out(pt q){return pq%(q-p)<-EPS;} };
vector<pt> intersect(vector<halfplane> b){
  vector<pt>bx={{DINF,DINF},{-DINF,DINF},{-DINF,-DINF},{DINF,-DINF}};
  fore(i,0,4) b.pb(halfplane(bx[i],bx[(i+1)%4]));
  sort(all(b)); int n=sz(b),q=1,h=0;
  vector<halfplane> c(sz(b)+10);
  fore(i,0,n){
  while(q<h&&b[i].out(c[h]^c[h-1])) h--;
  while(q<h&&b[i].out(c[q]^c[q+1])) q++; c[++h]=b[i];
  if(q<h&&abs(c[h].pq%c[h-1].pq)<EPS){
    if(c[h].pq*c[h-1].pq<=0) return {}; h--;
    if(b[i].out(c[h].p)) c[h]=b[i]; } }
  while(q<h-1&&c[q].out(c[h]^c[h-1]))h--;
  while(q<h-1&&c[h].out(c[q]^c[q+1]))q++;
  if(h-q<=1)return {}; c[h+1]=c[q]; vector<pt> s;
  fore(i,q,h+1) s.pb(c[i]^c[i+1]); return s; }
struct pol {
  int n;vector<pt> p; pol(){}
  pol(vector<pt> _p){p=_p;n=p.size();}
  double area(){
    double r=0.; fore(i,0,n)r+=p[i]%p[(i+1)%n];
    return abs(r)/2; // negative if CW, positive if CCW
  } };
```

4.5 line

```
struct line {
  double a, b, c;
  line(point p, point q) {
  a = p.y - q.y; b = q.x - p.x; c = -a * p.x - b * p.y; };
  void setOrigin(point p) { c += a * p.x + b * p.y; } //trasladar linea
      como si p fuera el origen
};
double det(double a, double b, double c, double d) {
  return a * d - b * c; }
point intersec(line a, line b) { //primero estar seguro si no son
    paralelas
  double d = -det(a.a, a.b, b.a, b.b);
  return point(det(a.c, a.b, b.c, b.b) / d, det(a.a, a.c, b.a, b.c) / d);
      }
```

4.6 minkowski

```
typedef vector<point> poly;
void norm(poly &pol) {
  int pos = 0;
  fore(i, 0, pol.size()) { if(pol[i] < pol[pos]) pos = i; }
  rotate(pol.begin(), pol.begin() + pos, pol.end()); }
poly minkos(poly &a, poly &b) {
  norm(a); norm(b);
  int posa = 0, posb = 0, ta = a.size(), tb = b.size();
  poly res; ll cro;
  while(posa < ta || posb < tb) {
  res.pb(a[(posa) % ta] + b[(posb) % tb]);
  cro = (a[(posa + 1) % ta] - a[posa % ta]) ^ (b[(posb + 1) % tb] -
      b[posb % tb]);
  if(cro == 0) posa++, posb++;
  else if(cro < 0) posb++;
  else posa++; }
  return res; }
```

4.7 point-in-poly

```
// logaritmico counterclockwise
bool inpol(poly &pol, point p) {
  int n = pol.size();
  if(((pol[1]-pol[0])^(p-pol[0]))<0||((pol[n - 1]-pol[0])^(p-pol[0]))>0)
    return 0;
  int lo = 1, hi = n - 2, mid, res;
  while(lo <= hi) {
    mid = (lo + hi) / 2;
    if(((pol[mid] - pol[0])^(p - pol[0])) >= 0) res = mid, lo=mid+1;
    else hi = mid - 1; }
  return ((pol[res + 1] - pol[res]) ^ (p - pol[res])) >= 0;
}
```

5 Graph

5.1 2sat

```
namespace sat2{
  set<int> G[tam], Ginv[tam];
  int N, mark[tam], mark_comp[tam], valor[tam];
  int neg(const int& x) { return (x>=N)? x - N : x + N;}
  void add_(const int& x,const int& y) {G[x].insert(y);Ginv[y].insert(x);}
  void addor(const int x,const int y) {add_(neg(x),y);add_(neg(y),x);}
  void dfs0(int u, vector<int>& orden) { mark[u] = 1;
  for(auto& v: G[u]) {
    if (!mark[v]) dfs0(v,orden);
  }  orden.push_back(u);
  }
  void dfs1(int u, const int& cmp) { mark_comp[u] = cmp;
  for(auto& v: Ginv[u]) {
    if (!mark_comp[v]) dfs1(v,cmp);
  }
  }
  bool check() { bool impos = false;
  for(int i = 0; i < N; i++) {
    impos |= (mark_comp[i] == mark_comp[neg(i)]);
     valor[i] = (mark_comp[i] > mark_comp[neg(i)]) ;}
  return !impos;
  }
}
```

5.2 articulation-bridges-biconnected

```
namespace art_bic {
  vector<int> g[tam];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[tam],B[tam],T;
  int nbc; // number of biconnected components
  int art[tam]; // 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]--;
  }
}
```

5.3 bellman-ford

```
int n; vector<ii> g[tam]; // u->[(v,cost)]
ll dist[tam];
void bford(int src){ // O(nm)
  fill(dist,dist+n,INF);dist[src]=0;
  fore(_,0,n)fore(x,0,n)if(dist[x]!=INF)for(auto t:g[x]){
  dist[t.fst]=min(dist[t.fst],dist[x]+t.snd);
  }
  fore(x,0,n)if(dist[x]!=INF)for(auto t:g[x]){
  if(dist[t.fst]>dist[x]+t.snd){
    // neg cycle: all nodes reachable from t.fst have -INF distance
    // to reconstruct neg cycle: save "prev" of each node, go up from
        t.fst until repeating a node. this node and all nodes between the
        two occurences form a neg cycle
  }  }  }
```

5.4 centroid

```
namespace cent_{
  vector<int> g[tam];int n;
  bool tk[tam];
  int fat[tam]; // father in centroid decomposition
  int szt[tam]; // size of subtree
  int calcsz(int x, int f){
  szt[x]=1;
  for(auto y:g[x])if(y!=f&&!tk[y])szt[x]+=calcsz(y,x);
  return szt[x];
  }
  void cdfs(int x=0, int f=-1, int sz=-1){ // O(nlogn)
  if(sz<0)sz=calcsz(x,-1);
  for(auto y:g[x])if(!tk[y]&&szt[y]*2>=sz){
    szt[x]=0;cdfs(y,f,sz);return;
  }
  tk[x]=true;fat[x]=f; // next is ops
  for(auto y:g[x])if(!tk[y])cdfs(y,x);
  }
  void centroid(){memset(tk,false,sizeof(tk));cdfs();}
}
```

5.5 dynamic-connectivity

```
namespace dyn_con {
struct UnionFind {
  int n,comp;
  vector<int> uf,si,c;
  UnionFind(int n=0):n(n),comp(n),uf(n),si(n,1){
  fore(i,0,n)uf[i]=i;}
  int find(int x){return x==uf[x]?x:find(uf[x]);}
  bool join(int x, int y){
  if((x=find(x))==(y=find(y)))return false;
  if(si[x]<si[y])swap(x,y);
  si[x]+=si[y];uf[y]=x;comp--;c.pb(y);
  return true;
  }
  int snap(){return c.size();}
  void rollback(int snap){
  while(c.size()>snap){
    int x=c.back();c.pop_back();
    si[uf[x]]-=si[x];uf[x]=x;comp++;
  }
  }  };
enum {ADD,DEL,QUERY};
struct Query {int type,x,y;};
struct DynCon {
  vector<Query> q;
  UnionFind dsu;
  vector<int> mt;
  map<pair<int,int>,int> last;
  DynCon(int n):dsu(n){}
  void add(int x, int y){
  if(x>y)swap(x,y);
  q.pb((Query){ADD,x,y});mt.pb(-1);last[{x,y}]=q.size()-1;
  }
  void remove(int x, int y){
  if(x>y)swap(x,y);
  q.pb((Query){DEL,x,y});
  int pr=last[{x,y}];mt[pr]=q.size()-1;mt.pb(pr);
  }
  void query(){q.pb((Query){QUERY,-1,-1});mt.pb(-1);}
  void process(){ // answers all queries in order
  if(!q.size())return;
  fore(i,0,q.size())if(q[i].type==ADD&&mt[i]<0)mt[i]=q.size();
  go(0,q.size());
  }
```

```
  void go(int s, int e){
  if(s+1==e){
    if(q[s].type==QUERY) // answer query using DSU
    printf("%d\n",dsu.comp); // can ask current state UnionFind
    return;
  }
  int k=dsu.snap(),m=(s+e)/2;
  for(int i=e-1;i>=m;--i)if(mt[i]>=0&&mt[i]<s)dsu.join(q[i].x,q[i].y);
  go(s,m);dsu.rollback(k);
  for(int i=m-1;i>=s;--i)if(mt[i]>=e)dsu.join(q[i].x,q[i].y);
  go(m,e);dsu.rollback(k);
  }  };
}
```

5.6 edmonds-blossom

```
namespace ed_bls{ // undirected G
  vector<int> g[tam];
  int n,m,mt[tam],qh,qt,q[tam],ft[tam],bs[tam];
  bool inq[tam],inb[tam],inp[tam];
  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;
  fore(x,0,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));
  fore(i,0,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(){ // O(n^2 m)
  int r=0;
  memset(mt,-1,sizeof(mt));
  fore(x,0,n)if(mt[x]<0)r+=aug(x,findp(x));
  return r;
  }
}
```

5.7 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 to get_path!
  p.clear();go(x);reverse(p.begin(),p.end());
  return p;
}
```

5.8 HLD LCA con heavy paths

```
/*
Nota:
- el segment tree no esta implementado, solo HLD

Uso:
- poner grafo en G
- correr init(N)
- para cada nodo v -> segBase[pos[v]] = val[v] (segBase es un nuevo
    arreglo)
- inicializar segment tree para [0, N-1]
- implementar la funcion segQuery(int l, int r)
*/
vector<vi> G;
vi parent, depth, heavy, head, pos;
int curPos;

int dfs(int v) {
  int size = 1;
  int maxChildrenSize = 0;
  for (int c : G[v]) {
    if (c == parent[v]) continue;
    parent[c] = v;
    depth[c] = depth[v] + 1;
    int cSize = dfs(c);
    size += cSize;
    if (cSize > maxChildrenSize) {
      maxChildrenSize = cSize;
      heavy[v] = c;
    }
  }
  return size;
}

void decompose(int v, int h) {
  head[v] = h;
  pos[v] = curPos++;
  if (heavy[v] != -1) {
    decompose(heavy[v], h);
  }
  for (int c : G[v]) {
    if (c != parent[v] && c != heavy[v]) {
      decompose(c, c);
    }
  }
}

void init(int nodes) {
  parent.assign(nodes, -1);
  depth.resize(nodes);
  heavy.assign(nodes, -1);
  head.resize(nodes);
  pos.resize(nodes);
  curPos = 0;
  // Raiz 0
  dfs(0);
  decompose(0, 0);
}

int segQuery(int l, int r);
```

```
int hldQuery(int a, int b) {
  int res = 0;
  for (; head[a] != head[b]; b = parent[head[b]]) {
    if (depth[head[a]] > depth[head[b]]) {
      swap(a, b);
    }
    int pathMax = segQuery(pos[head[b]], pos[b]);
    res = max(pathMax, res);
  }
  if (depth[a] > depth[b]) swap(a, b);
  int lastPath = segQuery(pos[a], pos[b]);
  res = max(res, lastPath);
  return res;
}
```

5.9 hld

```
vector<int> parent, depth, heavy, head, pos;
int cur_pos;
int dfs(int v, vector<vector<int>> const& adj) {
  int size = 1;
  int max_c_size = 0;
  for (int c : adj[v]) {
  if (c != parent[v]) {
    parent[c] = v, depth[c] = depth[v] + 1;
    int c_size = dfs(c, adj);
    size += c_size;
    if (c_size > max_c_size)
    max_c_size = c_size, heavy[v] = c;
  }
  }
  return size;
}
void decompose(int v, int h, vector<vector<int>> const& adj) {
  head[v] = h, pos[v] = cur_pos++;
  if (heavy[v] != -1)
  decompose(heavy[v], h, adj);
  for (int c : adj[v]) {
  if (c != parent[v] && c != heavy[v])
    decompose(c, c, adj);
  }
}
void init(vector<vector<int>> const& adj) {
  int n = adj.size();
  parent = vector<int>(n);
  depth = vector<int>(n);
  heavy = vector<int>(n, -1);
  head = vector<int>(n);
  pos = vector<int>(n);
  cur_pos = 0;

  dfs(0, adj);
  decompose(0, 0, adj);
  // init segtree with base[pos[i]]=val[i]
}
int query(int a, int b) { // for max
  int res = 0;
  for (; head[a] != head[b]; b = parent[head[b]]) {
  if (depth[head[a]] > depth[head[b]])
    swap(a, b);
  int cur_heavy_path_max = segQuery(pos[head[b]], pos[b]);
  res = max(res, cur_heavy_path_max);
  }
  if (depth[a] > depth[b])
  swap(a, b);
  int last_heavy_path_max = segQuery(pos[a], pos[b]);
  res = max(res, last_heavy_path_max);
  return res;
}
```

6 Math

6.1 berlekamp

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

```
vector<ll> berlekampMassey(vector<ll> s) {
  int n = sz(s), L = 0, m = 0;
  vector<ll> C(n), B(n), T;
  C[0] = B[0] = 1;
  ll b = 1;
  fore(i,0,n) { ++m;
  ll d = s[i] % mod;
  fore(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
  if (!d) continue;
  T = C; ll coef = d * modpow(b, mod-2) % mod;
  fore(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
  if (2 * L > i) continue;
  L = i + 1 - L; B = T; b = d; m = 0;
  }
  C.resize(L + 1); C.erase(C.begin());
  for (ll& x : C) x = (mod - x) % mod;
  return C;
}
```

6.2 catalan

```
// Catalan, parentesis balanceados, arboles binarios, triangulacion
    poligono convexto de n + 2 lados, caminos en grilla sin atravesar
    diagonal
// Cat[n] = C(2n, n) / (n + 1)
// C(n, k) es el coeficiente binomial
Cat[0] = 1;
Cat[n+1] = Cat[n] * 2 * (2 * n + 1) / (n + 2);
Cat[n] = Cat[n-1] * 2 * (2 * n - 1) / (n + 1);
```

6.3 chinese-remainder

```
constexpr long long safe_mod(long long x, long long m) {
  x %= m;
  if (x < 0) x += m;
  return x;
}
constexpr std::pair<long long, long long> inv_gcd(long long a, long long
    b) {
  a = safe_mod(a, b);
  if (a == 0) return {b, 0};
  long long s = b, t = a;
  long long m0 = 0, m1 = 1;
  while (t) {
  long long u = s / t;
  s -= t * u;
  m0 -= m1 * u;
  auto tmp = s;
  s = t;
  t = tmp;
  tmp = m0;
  m0 = m1;
  m1 = tmp;
  }
  if (m0 < 0) m0 += b / s;
  return {s, m0};
}
std::pair<long long, long long> crt(const std::vector<long long>& r,
        const std::vector<long long>& m) {
  assert(r.size() == m.size());
  int n = int(r.size());
  long long r0 = 0, m0 = 1;
  for (int i = 0; i < n; i++) {
  assert(1 <= m[i]);
  long long r1 = safe_mod(r[i], m[i]), m1 = m[i];
  if (m0 < m1) {
    std::swap(r0, r1);
    std::swap(m0, m1);
  }
  if (m0 % m1 == 0) {
    if (r0 % m1 != r1) return {0, 0};
    continue;
  }
  long long g, im;
  std::tie(g, im) = inv_gcd(m0, m1);
  long long u1 = (m1 / g);
  if ((r1 - r0) % g) return {0, 0};
  long long x = (r1 - r0) / g % u1 * im % u1;
  r0 += x * m0;
  m0 *= u1;
  if (r0 < 0) r0 += m0;
  }
  return {r0, m0};
}
cin>>a>>b>>c>>d;
extendedEuclid(b, d);
```

```
mul = b / g * d;
b /= g;
d /= g;
cout<<(mulmod(x, mulmod(b, c, mul), mul) + mulmod(y, mulmod(d, a, mul),
    mul)) % mul<<' '<<mul<<'\n';
```

6.4 exteded-euclid

```
int x, y, d;
void extendedEuclid(int a, int b)//ecuacion diofantica ax + by = d
{
  if(b==0) {x=1; y=0; d=a; return;}
  extendedEuclid(b,a%b);
  int x1=y;
  y = x-(a/b)*y;
  x=x1;
}
```

6.5 fast-gcd

```
int gcd(int a, int b) {
  if (!a || !b)
  return a | b;
  unsigned shift = __builtin_ctz(a | b);
  a >>= __builtin_ctz(a);
  do {
  b >>= __builtin_ctz(b);
  if (a > b)
    swap(a, b);
  b -= a;
  } while (b);
  return a << shift;
}
```

6.6 fft-operations

```
// MAXN must be power of 2 !!
// MOD-1 needs to be a multiple of MAXN !!
// big mod and primitive root for NTT:
typedef int tf;
typedef vector<tf> poly;
const tf MOD=998244353,RT=3,MAXN=1<<16;
tf addmod(tf a, tf b){tf r=a+b;if(r>=MOD)r-=MOD;return r;}
tf submod(tf a, tf b){tf r=a-b;if(r<0)r+=MOD;return r;}
tf mulmod(ll a, ll b){return a*b%MOD;}
tf pm(ll a, ll b){
  ll r=1;
  while(b){
  if(b&1) r=mulmod(r,a); b>>=1;
  a=mulmod(a,a);
  }
  return r;
}
tf inv(tf a){return pm(a,MOD-2);}
// FFT
/*struct CD {
  double r,i;
  CD(double r=0, double i=0):r(r),i(i){}
  double real()const{return r;}
  void operator/=(const int c){r/=c, i/=c;}
};
CD operator*(const CD& a, const CD& b){
  return CD(a.r*b.r-a.i*b.i,a.r*b.i+a.i*b.r);}
CD operator+(const CD& a, const CD& b){return CD(a.r+b.r,a.i+b.i);}
CD operator-(const CD& a, const CD& b){return CD(a.r-b.r,a.i-b.i);}
const double pi=acos(-1.0);*/
// NTT
struct CD {
  tf x;
  CD(tf x):x(x){}
  CD(){}
};
CD operator*(const CD& a, const CD& b){return CD(mulmod(a.x,b.x));}
CD operator+(const CD& a, const CD& b){return CD(addmod(a.x,b.x));}
CD operator-(const CD& a, const CD& b){return CD(submod(a.x,b.x));}
vector<tf> rts(MAXN+9,-1);
CD root(int n, bool inv){
  tf r=rts[n]<0?rts[n]=pm(RT,(MOD-1)/n):rts[n];
  return CD(inv?pm(r,MOD-2):r);
}
CD cp1[MAXN+9],cp2[MAXN+9];
int R[MAXN+9];
void dft(CD* a, int n, bool inv){
  fore(i,0,n)if(R[i]<i)swap(a[R[i]],a[i]);
```

```
  for(int m=2;m<=n;m*=2){
  //double z=2*pi/m*(inv?-1:1); // FFT
  //CD wi=CD(cos(z),sin(z)); // FFT
  CD wi=root(m,inv); // NTT
  for(int j=0;j<n;j+=m){
    CD w(1);
    for(int k=j,k2=j+m/2;k2<j+m;k++,k2++){
    CD u=a[k];CD v=a[k2]*w;a[k]=u+v;a[k2]=u-v;w=w*wi;
    }
  }
  }
  //if(inv)fore(i,0,n)a[i]/=n; // FFT
  if(inv){ // NTT
  CD z(pm(n,MOD-2)); // pm: modular exponentiation
  fore(i,0,n)a[i]=a[i]*z;
  }
}
poly multiply(poly& p1, poly& p2){
  int n=p1.size()+p2.size()+1;
  int m=1,cnt=0;
  while(m<=n)m+=m,cnt++;
  fore(i,0,m){R[i]=0;fore(j,0,cnt)R[i]=(R[i]<<1)|((i>>j)&1);}
  fore(i,0,m)cp1[i]=0,cp2[i]=0;
  fore(i,0,p1.size())cp1[i]=p1[i];
  fore(i,0,p2.size())cp2[i]=p2[i];
  dft(cp1,m,false);dft(cp2,m,false);
  fore(i,0,m)cp1[i]=cp1[i]*cp2[i];
  dft(cp1,m,true);
  poly res;
  n-=2;
  //fore(i,0,n)res.pb((tf)floor(cp1[i].real()+0.5)); // FFT
  fore(i,0,n)res.pb(cp1[i].x); // NTT
  return res;
}
//Polynomial division: O(n*log(n))
//Multi-point polynomial evaluation: O(n*log^2(n))
//Polynomial interpolation: O(n*log^2(n))
//Works with NTT. For FFT, just replace addmod,submod,mulmod,inv
poly add(poly &a, poly &b){
  int n=SZ(a),m=SZ(b);
  poly ans(max(n,m));
  fore(i,0,max(n,m)){
  if(i<n) ans[i]=addmod(ans[i],a[i]);
  if(i<m) ans[i]=addmod(ans[i],b[i]);
  }
  while(SZ(ans)>1&&!ans.back())ans.pop_back();
  return ans;
}
poly invert(poly &b, int d){
 poly c = {inv(b[0])};
 while(SZ(c)<=d){
   int j=2*SZ(c);
  auto bb=b; bb.resize(j);
  poly cb=multiply(c,bb);
  fore(i,0,SZ(cb)) cb[i]=submod(0,cb[i]);
  cb[0]=addmod(cb[0],2);
  c=multiply(c,cb);
  c.resize(j);
 }
 c.resize(d+1);
 return c;
}
pair<poly,poly> divslow(poly &a, poly &b){
  poly q,r=a;
  while(SZ(r)>=SZ(b)){
  q.pb(mulmod(r.back(),inv(b.back())));
  if(q.back()) fore(i,0,SZ(b)){
    r[SZ(r)-i-1]=submod(r[SZ(r)-i-1],mulmod(q.back(),b[SZ(b)-i-1]));
  }
  r.pop_back();
  }
  reverse(ALL(q));
  return {q,r};
}
pair<poly,poly> divide(poly &a, poly &b){ //returns {quotient,remainder}
  int m=SZ(a),n=SZ(b),MAGIC=750;
  if(m<n) return {{0},a};
  if(min(m-n,n)<MAGIC)return divslow(a,b);
  poly ap=a; reverse(ALL(ap));
  poly bp=b; reverse(ALL(bp));
  bp=invert(bp,m-n);
  poly q=multiply(ap,bp);
  q.resize(SZ(q)+m-n-SZ(q)+1,0);
  reverse(ALL(q));
  poly bq=multiply(b,q);
  fore(i,0,SZ(bq)) bq[i]=submod(0,bq[i]);
  poly r=add(a,bq);
  return {q,r};
}
vector<poly> tree;
```

```
void filltree(vector<tf> &x){
  int k=SZ(x);
  tree.resize(2*k);
  fore(i,k,2*k) tree[i]={submod(0,x[i-k]),1};
  for(int i=k-1;i;i--) tree[i]=multiply(tree[2*i],tree[2*i+1]);
}
vector<tf> evaluate(poly &a, vector<tf> &x){
  filltree(x);
  int k=SZ(x);
  vector<poly> ans(2*k);
  ans[1]=divide(a,tree[1]).snd;
  fore(i,2,2*k) ans[i]=divide(ans[i>>1],tree[i]).snd;
  vector<tf> r; fore(i,0,k) r.pb(ans[i+k][0]);
  return r;
}
poly derivate(poly &p){
  poly ans(SZ(p)-1);
  fore(i,1,SZ(p)) ans[i-1]=mulmod(p[i],i);
  return ans;
}
poly interpolate(vector<tf> &x, vector<tf> &y){
  filltree(x);
  poly p=derivate(tree[1]);
  int k=SZ(y);
  vector<tf> d=evaluate(p,x);
  vector<poly> intree(2*k);
  fore(i,k,2*k) intree[i]={mulmod(y[i-k],inv(d[i-k]))};
  for(int i=k-1;i;i--){
  poly p1=multiply(tree[2*i],intree[2*i+1]);
  poly p2=multiply(tree[2*i+1],intree[2*i]);
  intree[i]=add(p1,p2);
  }
  return intree[1];
}
int main(){FIN;
  int m,k; cin>>m>>k;
  int top=max(k,m)+2;
  vector<int> x,y;
  int ac=0;
  fore(i,0,top){
  ac=addmod(ac,pm(i,k));
  x.pb(i); y.pb(ac);
  }
  poly p=interpolate(x,y);
  vector<int> xs;
  fore(i,0,m){
  ll x; cin>>x; x%=MOD;
  xs.pb(x);
  }
  while(SZ(xs)!=top) xs.pb(0);
  vector<int> ans=evaluate(p,xs);
  fore(i,0,m)cout<<ans[i]<<" ";cout<<"\n";
}
```

6.7 fht

```
ll c1[tam+9],c2[tam+9]; // tam 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);
}
void fht(vector<ll>& p, bool inv) {
  fore(i, 0, sz(p)) c1[i] = p[i];
  fht(c1, sz(p), inv);
  fore(i, 0, sz(p)) p[i] = c1[i];
}
```

6.8 formulas

```
//Stirling number of the second kind is the number of ways to partition a set of n objects into k non-empty subsets.
S(n,k)= ks (n-1,k)+S(n-1,k-1),where S(0,0)=1,s(n,0)=s(0,n)=0
S(n,2)=2^(n-1)-1
S(n,k) k !=number of ways to color n nodes using colors from 1 to k such that each color is used at least once.
An r-associated Stirling number of the second kind is the number of ways to partition a set of n objects into k subsets,
with each subset containing at least r elements.It is denoted by Sr(n,k) and obeys the recurrence relation.
Sr(n+1,k)=kSr(n,k)+C(n,r-1)Sr(n-r+1,k-1)
//The Stirling numbers of the first kind count permutations according to their number of cycles (counting fixed points as cycles of length one).
S(n,k) counts the number of permutations of n elements with k disjoint cycles.
S(n,k) = (n-1)S(n-1,k)+S(n-1,k-1),where,s(0,0) =1,S(n,0)=s(0,n)=0
Sum(k,0,n) S(n,k) = n!
The unsigned Stirling numbers may also be defined algebraically, as the coefficient of the rising factorial:
    x^(~n)=x(x+1)(x+n1)=Sum(k,0,n)s(n,k)x^k
//Bell number count the number of partition of a set
Bn+1 = Sum(k,0,n){C(n,k)*Bk}
Bn = S Sum(k,0,n)Sr(n,k), where Sr is Stirling number of 2kind
//Formally, for a sequence of numbers {ai}, we define the ordinary generating function (OGF) of a to be A(x)=Sum(i,0,inf)aix^i.
1/(1 x ) = 1+ x + x^2+...= Sum(n,0,inf)x^n
 ln (1 x )=x + x^2/2 + x^3/3+...=Sum(n,0,inf)x^n/n
e^x=1+x + x^2/2! + x^3/3!+...=Sum(n,0,inf)x^n/n!
(1 x )^ k =C( k1 ,0)x^0+C(k,1)x^1+C(k+1,2)x^2+...=Sum(n,0,inf)C(n+k-1,n)x^n
For OGF, C(x)=A(x)^k generates the sequence
    cn=Sum(i1...ik,i1+i2+...+ik=n)(ai1*ai2...*aik)
For EGF, C(x)=A(x)^k generates the sequence
cn=Sum(i1...ik,i1+i2+...+ik=n)(ai1*ai2...*aik)*n!/(i1!*...ik!)
Suppose want to generate the sequence cn=a0+a1+...+an. Then, we can take C(x)=1/(1x)*A(x).
```

6.9 gauss

```
// resuelve Ax = b, dada la matriz a de n * (m + 1), n ecuaciones y m variables, siendo la ultima columna el vector b
// The function returns the number of solutions of the system (0,1,or ). if there's at least a solution, it's in ans
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity or a big number
int gauss (vector < vector<double> > a, vector<double> & ans) {
  int n = (int) a.size();
  int m = (int) a[0].size() - 1;
  vector<int> where (m, -1);
  for (int col=0, row=0; col<m && row<n; ++col) {
  int sel = row;
  for (int i=row; i<n; ++i)
    if (abs (a[i][col]) > abs (a[sel][col]))
    sel = i;
  if (abs (a[sel][col]) < EPS)
    continue;
  for (int i=col; i<=m; ++i)
    swap (a[sel][i], a[row][i]);
  where[col] = row;
  for (int i=0; i<n; ++i)
    if (i != row) {
    double c = a[i][col] / a[row][col];
    for (int j=col; j<=m; ++j)
      a[i][j] -= a[row][j] * c;
    }
  ++row;
  }
  ans.assign (m, 0);
  for (int i=0; i<m; ++i)
  if (where[i] != -1)
    ans[i] = a[where[i]][m] / a[where[i]][i];
  for (int i=0; i<n; ++i) {
  double sum = 0;
  for (int j=0; j<m; ++j)
    sum += ans[j] * a[i][j];
  if (abs (sum - a[i][m]) > EPS)
    return 0;
  }
  for (int i=0; i<m; ++i)
  if (where[i] == -1)
    return INF;
  return 1;
}
```

6.10 interpol-o(n)

```
// evaluar un "polinomio interpolado" en o(nlogMOD)
// debe cumplir xi+1 -  xi  = xj +1  - xj  for all i,  j  < n
// recibe vector de ys tal que f(i) = y[i]
ll eval(vll ys, ll x) {
  int n = ys.size();
  if(x < n) return ys[x];
  ll res = 0, up = 1, dow = 1;
  fore(i, 1, n)
  dow = dow * (MOD - i) % MOD,
  up = up * (x - i) % MOD;
  fore(i, 1, n) {
  up = up * (x - i + 1) % MOD * pot(x - i, MOD - 2) % MOD;
  dow = dow * i % MOD * pot(MOD - (n - i), MOD - 2) % MOD;
  res = (res + ys[i] * up % MOD * pot(dow, MOD - 2) % MOD) % MOD;
  }
  return res;
}
```

6.11 karatsuba

```
typedef ll tp;
// #define add(n,s,d,k) fore(i,0,n)(d)[i]+=(s)[i]*k
#define add(n,s,d,k) fore(i,0,n)(d)[i]+=(s)[i]*k%MOD, (d)[i] = ((d)[i] %
    MOD + MOD) % MOD;
tp* ini(int n){tp *r=new tp[n];fill(r,r+n,0);return r;}
void karatsura(int n, tp* p, tp* q, tp* r){
  if(n<=0)return;
  // if(n<35)fore(i,0,n)fore(j,0,n)r[i+j]+=p[i]*q[j];
  if(n<35)fore(i,0,n)fore(j,0,n)r[i+j]+=p[i]*q[j] % MOD, r[i + j] %= MOD;
  else {
  int nac=n/2,nbd=n-n/2;
  tp *a=p,*b=p+nac,*c=q,*d=q+nac;
  tp *ab=ini(nbd+1),*cd=ini(nbd+1),*ac=ini(nac*2),*bd=ini(nbd*2);
  add(nac,a,ab,1);add(nbd,b,ab,1);
  add(nac,c,cd,1);add(nbd,d,cd,1);
  karatsura(nac,a,c,ac);karatsura(nbd,b,d,bd);
  add(nac*2,ac,r+nac,-1);add(nbd*2,bd,r+nac,-1);
  add(nac*2,ac,r,1);add(nbd*2,bd,r+nac*2,1);
  karatsura(nbd+1,ab,cd,r+nac);
  free(ab);free(cd);free(ac);free(bd);
  }
}
vector<tp> multiply(vector<tp> p0, vector<tp> p1){
  int n=max(p0.size(),p1.size());
  tp *p=ini(n),*q=ini(n),*r=ini(2*n);
  fore(i,0,p0.size())p[i]=p0[i];
  fore(i,0,p1.size())q[i]=p1[i];
  karatsura(n,p,q,r);
  vector<tp> rr(r,r+p0.size()+p1.size()-1);
  free(p);free(q);free(r);
  return rr;
}
```

6.12 LinearRecurrence

```
/*
Description: Generates the kth term of an n-order linear recurrence
S[i] =    S[i    j     1]tr[j], given S[0 . . .  n     1] and tr[0 . . . n
      1]. Faster
than matrix multiplication. Useful together with BerlekampMassey.
Usage: linearRec({0, 1}, {1, 1}, k) // kth Fibonacci number
Time: O (n^2 log k)*/
typedef vector<ll> Poly;
#define sz(x) (int)(x).size()
  ll linearRec(Poly S, Poly tr, ll k) {
  int n = sz(tr);
  auto combine = [&](Poly a, Poly b) {
  Poly res(n * 2 + 1);
  fore(i,0,n+1) fore(j,0,n+1)
  res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
  for (int i = 2 * n; i > n; --i) fore(j,0,n)
  res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
  res.resize(n + 1);
  return res;
  };
  Poly pol(n + 1), e(pol);
  pol[0] = e[1] = 1;
  for (++k; k; k /= 2) {
  if (k % 2) pol = combine(pol, e);
  e = combine(e, e);
  }
  ll res = 0;
  fore(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
  return res;
```

```
}
```

6.13 matrix-determinant

```
const double EPS=1e-4;
double reduce(vector<vector<double> >& x){ // returns determinant
  int n=x.size(),m=x[0].size();
  int i=0,j=0;double r=1.;
  while(i<n&&j<m){
  int l=i;
  fore(k,i+1,n)if(abs(x[k][j])>abs(x[l][j]))l=k;
  if(abs(x[l][j])<EPS){j++;r=0.;continue;}
  if(l!=i){r=-r;swap(x[i],x[l]);}
  r*=x[i][j];
  for(int k=m-1;k>=j;k--)x[i][k]/=x[i][j];
  fore(k,0,n){
    if(k==i)continue;
    for(int l=m-1;l>=j;l--)x[k][l]-=x[k][j]*x[i][l];
  }
  i++;j++;
  }
  return r;
}
```

6.14 matrix-fast-pow

```
typedef vector<vector<ll> > Matrix;
Matrix ones(int n) {
  Matrix r(n,vector<ll>(n));
  fore(i,0,n)r[i][i]=1;
  return r;
}
Matrix operator*(Matrix &a, Matrix &b) {
  int n=SZ(a),m=SZ(b[0]),z=SZ(a[0]);
  Matrix r(n,vector<ll>(m));
  fore(i,0,n)fore(j,0,m)fore(k,0,z)
  r[i][j]+=a[i][k]*b[k][j],r[i][j]%=mod;
  return r;
}
Matrix be(Matrix b, ll e) {
  Matrix r=ones(SZ(b));
  while(e){if(e&1LL)r=r*b;b=b*b;e/=2;}
  return r;
}
```

6.15 moebius

```
//f(n)=sum(d|n,g(d))=>g(n)=sum(d|n,f(d)*mu(n/d))
//f(n)=sum(i->inf,g(i*n)*mu(i));f(n)=#f(a)->n;g(n)=#f(a)->xn
int mu[tam], is_prime [tam];
fore(i, 0, tam) mu[i]=is_prime[i]=1;
fore(i, 2, tam) if(is_prime[i]) {
  forg(j, i, tam, i) {
  if(j > i) is_prime[j] = 0;
  if(j / i % i == 0) mu[j]=0;
  mu[j] = -mu[j];
  }
}
```

6.16 pollard-rho-miller-rabil

```
ll gcd(ll a, ll b){return a?gcd(b%a,a):b;}
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;
}
ll expmod(ll b, ll e, ll m){
  if(!e)return 1;
  ll q=expmod(b,e/2,m);q=mulmod(q,q,m);
  return e&1?mulmod(b,q,m):q;
}
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;
}
// 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.f==0){
    p.s++; n/=p.f;
  }
  }
  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);}
}
```

6.17 simplex

```
vector<int> X,Y;
vector<vector<double> > A;
vector<double> b,c;
double z;
int n,m;
void pivot(int x,int y){
  swap(X[y],Y[x]);
  b[x]/=A[x][y];
  fore(i,0,m)if(i!=y)A[x][i]/=A[x][y];
  A[x][y]=1/A[x][y];
  fore(i,0,n)if(i!=x&&abs(A[i][y])>EPS){
  b[i]-=A[i][y]*b[x];
  fore(j,0,m)if(j!=y)A[i][j]-=A[i][y]*A[x][j];
  A[i][y]=-A[i][y]*A[x][y];
  }
  z+=c[y]*b[x];
  fore(i,0,m)if(i!=y)c[i]-=c[y]*A[x][i];
  c[y]=-c[y]*A[x][y];
}
pair<double,vector<double> > simplex( // maximize c^T x s.t. Ax<=b, x>=0
  vector<vector<double> > _A, vector<double> _b, vector<double> _c){
  // returns pair (maximum value, solution vector)
  A=_A;b=_b;c=_c;
  n=b.size();m=c.size();z=0.;
  X=vector<int>(m);Y=vector<int>(n);
  fore(i,0,m)X[i]=i;
  fore(i,0,n)Y[i]=i+m;
  while(1){
  int x=-1,y=-1;
  double mn=-EPS;
  fore(i,0,n)if(b[i]<mn)mn=b[i],x=i;
  if(x<0)break;
  fore(i,0,m)if(A[x][i]<-EPS){y=i;break;}
  assert(y>=0); // no solution to Ax<=b
  pivot(x,y);
  }
  while(1){
  double mx=EPS;
  int x=-1,y=-1;
  fore(i,0,m)if(c[i]>mx)mx=c[i],y=i;
  if(y<0)break;
  double mn=1e200;
  fore(i,0,n)if(A[i][y]>EPS&&b[i]/A[i][y]<mn)mn=b[i]/A[i][y],x=i;
```

```
  assert(x>=0); // c^T x is unbounded
  pivot(x,y);
  }
  vector<double> r(m);
  fore(i,0,n)if(Y[i]<m)r[Y[i]]=b[i];
  return mp(z,r);
}
```

6.18 stirling y bell

```
// stir[n][k] cantidad de formas de paritcionar un conjunto de n
    elementos en k conjuntos
// bell[n] cantidad de formas de particionar un conjunto
ll stir[tam][tam];
ll bell[tam];
void stirBell() {
  fore(i, 1, tam) {
  stir[i][1] = 1;
  fore(j, 2, 1010)
  stir[i][j] = (j * stir[i - 1][j] % MOD + stir[i - 1][j - 1]) % MOD;
  }
  fore(i, 1, tam)
  fore(j, 1, i + 1)
  bell[i] = (bell[i] + stir[i][j]) % MOD;
}
```

7 Shortcuts

7.1 dsu

```
int cnt[maxn];
void dfs(int v, int p, bool keep){
  int mx = -1, bigChild = -1;
  for(auto u : g[v])
   if(u != p && sz[u] > mx)
    mx = sz[u], bigChild = u;
  for(auto u : g[v])
  if(u != p && u != bigChild)
    dfs(u, v, 0); // run a dfs on small childs and clear them from cnt
  if(bigChild != -1)
  dfs(bigChild, v, 1); // bigChild marked as big and not cleared from cnt
  for(auto u : g[v])
  if(u != p && u != bigChild)
  for(int p = st[u]; p < ft[u]; p++)
  cnt[ col[ ver[p] ] ]++;
  cnt[ col[v] ]++;
  //now cnt[c] is the number of vertices in subtree of vertex v that has
      color c. You can answer the queries easily.
  if(keep == 0)
  for(int p = st[v]; p < ft[v]; p++)
  cnt[ col[ ver[p] ] ]--;
}
```

7.2 mo

```
void remove(idx);
void add(idx);
int get_answer();
int block_size;
struct Query {
  int l, r, idx;
  bool operator<(Query other) const
  {
  if (l / block_size != other.l / block_size)
    return mp(l, r) < mp(other.l, other.r);
  return ((l / block_size) & 1) ? (r < other.r) : (r > other.r);
  }
};
vector<int> mo_s_algorithm(vector<Query> queries) {
  vector<int> answers(queries.size());
  sort(queries.begin(), queries.end());
  int cur_l = 0;
  int cur_r = -1;
  for (Query q : queries) {
  while (cur_l > q.l) {
    cur_l--;
    add(cur_l);
  }
  while (cur_r < q.r) {
    cur_r++;
    add(cur_r);
  }
  while (cur_l < q.l) {
```

```
    remove(cur_l);
    cur_l++;
  }
  while (cur_r > q.r) {
    remove(cur_r);
    cur_r--;
  }
  answers[q.idx] = get_answer();
  }
  return answers;
}
```

7.3 shortcuts

```
// Better random mt19937_64 para 64 bits
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
cout << rng() << endl;
shuffle(permutation.begin(), permutation.end(), rng);
// while TLE
double t = clock(), TLE = 3;
while((clock() - t) / CLOCKS_PER_SEC < TLE);
// 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 kth largest 0 indexed, order_of_key finds how many are
    less than
// Faster map gp_hash_table<int,int,my_hash> m;
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
struct my_hash {
  const uint64_t RANDOM =
      chrono::steady_clock::now().time_since_epoch().count();
  static uint64_t splitmix64(uint64_t x) {
  x += 0x9e3779b97f4a7c15;
  x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
  x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
  return x ^ (x >> 31);
  }
  size_t operator()(uint64_t x) const {
  return splitmix64(x + RANDOM);
  }
};
```

8 Strings

8.1 aho-corasick

```
struct vertex {
  int go[26], pch, par, link = -1, super = -1, leaf = 0;
  vertex(): link(0), super(0) { mem(go, -1); }
  vertex(int ch, int from): pch(ch), par(from) { mem(go, -1); }
};
vector<vertex> t(1);
void add(string &s, int pos) {
  int node = 0;
  for(char ch : s) {
  ch -= 'a';
  if(t[node].go[ch] == -1)
    t[node].go[ch] = t.size(); t.emplace_back(ch, node);
  node = t[node].go[ch];
  }
  t[node].leaf = 1;
}
int go(int node, char c);
int suff(int node) {
  if(t[node].link == -1)
  t[node].link = t[node].par == 0 ? 0 : go(suff(t[node].par),
      t[node].pch);
  return t[node].link;
}
int go(int node, char ch) {
  if(t[node].go[ch] == -1)
  t[node].go[ch] = node == 0 ? 0 : go(suff(node), ch);
  return t[node].go[ch];
}
int super(int v) {
  if(t[v].super == -1)
  t[v].super = t[suff(v)].leaf ? suff(v) : super(suff(v));
  return t[v].super;
}
```

8.2 hsh-128

```
#define bint __int128
struct Hash {
  bint MOD=212345678987654321LL,P=1777771,PI=106955741089659571LL;
  vector<bint> h,pi;
  Hash(string& s){
  assert((P*PI)%MOD==1);
  h.resize(s.size()+1);pi.resize(s.size()+1);
  h[0]=0;pi[0]=1;
  bint p=1;
  fore(i,1,s.size()+1){
    h[i]=(h[i-1]+p*s[i-1])%MOD;
    pi[i]=(pi[i-1]*PI)%MOD;
    p=(p*P)%MOD;
  }
  }
  ll get(int s, int e){
  return (((h[e]-h[s]+MOD)%MOD)*pi[s])%MOD;
  }
};
```

8.3 manacher

```
int d1[MAXN];//d1[i] = max odd palindrome centered on i
int d2[MAXN];//d2[i] = max even palindrome centered on i
//s  aabbaacaabbaa
//d1 1111117111111
//d2 0103010010301
void manacher(string& s){
  int l=0,r=-1,n=s.size();
  fore(i,0,n){
  int k=i>r?1:min(d1[l+r-i],r-i);
  while(i+k<n&&i-k>=0&&s[i+k]==s[i-k])k++;
  d1[i]=k--;
  if(i+k>r)l=i-k,r=i+k;
  }
  l=0;r=-1;
  fore(i,0,n){
  int k=i>r?0:min(d2[l+r-i+1],r-i+1);k++;
  while(i+k<=n&&i-k>=0&&s[i+k-1]==s[i-k])k++;
  d2[i]=--k;
  if(i+k-1>r)l=i-k,r=i+k-1;
  }
}
```

8.4 preffix-function

```
vector<int> prefix_function(string &s) {
  int n = (int)s.length();
  vector<int> pi(n);
  for (int i = 1; i < n; i++) {
  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;
}
void compute_automaton(string &s, vector<vector<int>>& aut) {
  s += '#'; int n = s.size();
  vector<int> pi = prefix_function(s);
  aut.assign(n, vector<int>(26));
  for (int i = 0; i < n; i++) {
  for (int c = 0; c < 26; c++) {
    if (i > 0 && 'a' + c != s[i])
    aut[i][c] = aut[pi[i-1]][c];
    else
    aut[i][c] = i + ('a' + c == s[i]);
  }
  }
}
```

8.5 suffix-array

```
vector<vector<int>> table;
vector<int> suffixa(string &s){
  int n = s.size(), cc, ax;
  vector<int> sa(n), sa1(n), col(n), col1(n), head(n);
  fore(i, 0, n) sa[i] = i;
  auto cmp = [&](int a, int b){ return s[a] < s[b]; };
  stable_sort(sa.begin(), sa.end(), cmp);
```

```
  head[0] = col[sa[0]] = cc = 0;
  fore(i, 1, n){
  if(s[sa[i]] != s[sa[i-1]])
    cc++, head[cc] = i;
  col[sa[i]] = cc;
  }
  table.pb(col);
  for(int k = 1; k < n; k *= 2){
  fore(i, 0, n){
    ax = (sa[i] - k + n) % n;
    sa1[head[col[ax]]++] = ax;
  }
  swap(sa, sa1);
  col1[sa[0]] = head[0] = cc = 0;
  fore(i, 1, n){
    if(col[sa[i]] != col[sa[i - 1]] || col[(sa[i] + k) % n] != col[(sa[i
        - 1] + k) % n])
    cc++, head[cc] = i;
    col1[sa[i]] = cc;
  }
  swap(col, col1); table.pb(col);
  if(col[sa[n - 1]] == n - 1) break;
  }
  return sa;
}
pair<int, int> query(int b, int e){
  int lev = 31 - __builtin_clz(e - b + 1);
  return mp(table[lev][b], table[lev][e - (1 << lev) + 1]);
}
bool comp(int b1, int e1, int b2, int e2){
  int siz = min(e1 - b1, e2 - b2);
  ii le = query(b1, b1 + siz), ri = query(b2, b2 + siz);
  if(le == ri)
  return e1 - b1 < e2 - b2;
  return le < ri;
}
vector<int> lcp(string &s, vector<int> &sa){
  int n = s.size(), k, z = 0;
  vector<int> sa1(n), lcp(n);
  fore(i, 0, n) sa1[sa[i]] = i;
  fore(i, 0, n){
  k = sa1[i];
  if(k < n - 1)
    while(s[i + z] == s[sa[k+1] + z])
    z++;
  lcp[k] = z; z = max(z-1, 0);
  }
  return lcp;
}
```

8.6 suffix-automata

```
struct state {int len,link;map<char,int> next;}; //clear next!!
state st[100005];
int sz,last;
void sa_init(){
  last=st[0].len=0;sz=1;
  st[0].link=-1;
}
void sa_extend(char c){
  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;
}
// input: abcbcbc
// i,link,len,next
// 0 -1 0 (a,1) (b,5) (c,7)
// 1 0 1 (b,2)
// 2 5 2 (c,3)
// 3 7 3 (b,4)
// 4 9 4 (c,6)
// 5 0 1 (c,7)
// 6 11 5 (b,8)
// 7 0 2 (b,9)
// 8 9 6 (c,10)
```

```
//  9 5 3 (c,11)
//  10 11 7
```

```
//  11 7 4 (b,8)
```