Team notebook

August 11, 2021

Contents

1 merged

```
/// persistent segtree
struct node{
    node *left, *right; int val;
    node(int a = 0, node *b = 0, node *c = 0):
        val(a),left(b), right(c) {}
    void build(int l, int r){
        if(l == r) return;
        left = new node(); right = new node();
        int mid = l + r >> 1;
        left -> build(l, mid); right ->
            build(mid+1, r);
    }
    node *update(int l, int r, int idx, int v){
        if(r < idx || l > idx) return this;
        if(l == r){
            node *ret = new node(val, left, right);
            ret -> val += v;
            return ret;
        }
        int mid = l + r >> 1;
        node *ret = new node(val);
        ret -> left = left -> update(l, mid, idx,
            v);
        ret -> right = right->update(mid+1, r,
            idx,v);
        ret -> val = ret->
            left->val+ret->right->val;
        return ret;
    }
    int query(int l, int r, int i, int j){
        if(r < i || l > j) return 0;
        if(i <= l && r <= j) return val;
        int mid = l + r >> 1;
        return left->query(l,mid,i,j) +
            right->query(mid+1,r,i,j);
    }
};
/// centroid decomp
int LCA(int u, int v){
    if(depth[u]<depth[v]) swap(u,v);
    for(int i=MAXLG-1; i>=0; i--)
        if( (1<<i) <= depth[u]-depth[v])
            u=parent[i][u];
    if(u==v) return v;
    for(int i=MAXLG-1; i>=0; i--)
        if(parent[i][u]!=parent[i][v])
            u=parent[i][u], v=parent[i][v];
    return parent[0][v];
}
void dfs1(int u, int p=0){
    parent[0][u]=p, depth[u]=depth[p]+1;
    for(int i=1; i<MAXLG; i++) parent[i][u] =
        parent[i-1][parent[i-1][u]];
    for(auto v: g[u])if(v!=p) dfs1(v, u);
}
void dfs2(int u, int p=0){
    child[u]=1;
    for(auto v: g[u]) if(v!=p and !vis[v])
        dfs2(v, u), child[u]+=child[v];
}
int getcenter(int u, int p, int n){
    for(auto v: g[u]) if(v!=p and !vis[v])
        if(child[v]>n) return getcenter(v, u, n);
    return u;
}
void decompos(int u, int p=0){
    dfs2(u);
    int c = getcenter(u, p, child[u]/2);
    vis[c] = true; /// work for c
    for(auto v : g[c]) if(!vis[v]) decompos(v, u);
}
/// range sum segtree
int tree[4*N], lazy[4*N], a[N];
void relax (int cn, int b, int e) {
  if (lazy[cn]) {
    tree[cn] += (e-b+1)*lazy[cn] ;
    if (b != e) {
      lazy[2*cn] += lazy[cn] ;
      lazy[2*cn + 1] += lazy[cn] ;
    }
    lazy[cn] = 0;
  }
}
void upd (int cn, int b, int e, int i, int j, int
    add) {
  relax(cn,b,e);
  if (b > j or e < i) return;
```

```
  int lc = 2*cn , rc = lc + 1 , mid = (b+e)/2;
  if (b >= i and e <= j) {
    lazy[cn] += add;
    relax(cn,b,e);
    return;
  }
  upd(lc,b,mid,i,j,add);
  upd(rc,mid+1,e,i,j,add);
  tree[cn] = tree[lc] + tree[rc];
}
int query (int cn, int b, int e, int i, int j) {
  relax(cn,b,e);
  if (b > j or e < i) return 0;
  if (b >= i and e <= j) return tree[cn];
  int lc = 2*cn, rc = lc + 1, mid = (b+e)/2;
  return query(lc,b,mid,i,j) +
      query(rc,mid+1,e,i,j);
}
/// HLD
void HLD(int u, int p=0){
    chainIdx[u] = chainCnt; flatIdx[u] = flatCnt;
    flat[flatCnt++] = u;
    int biggie = -1, mx = 0;
    for (int v : edg[u]) if(v!=p)
        if (mx < sbtr[v]) mx = sbtr[v], biggie = v;
    if (biggie==-1) return;
    HLD(biggie, u);
    for (int v : edg[u]) if(v!=p and v!=biggie)
        chainHead[++chainCnt]=v, HLD(v, u);
}
void upSegments(int l, int u, vector<PII>&vp){
    while (chainIdx[l] != chainIdx[u]) {
        int uhead = chainHead[chainIdx[u]];
        vp.push_back(pii(flatIdx[uhead],
            flatIdx[u]));
        u = par[0][uhead];
    }
    if (l!=u) vp.push_back(pii(flatIdx[l]+1,
        flatIdx[u]));
}
vector<PII>getChainSegments(int u, int v){
    int l = LCA(u, v); vector<PII>ret;
    ret.push_back(pii(flatIdx[l], flatIdx[l]));
    if (u==v) return ret;
    upSegments(l, u, ret), upSegments(l, v, ret);
    return ret;
}
in main :
/// ** fill sparse, subtr, lvl here
chainHead[0] = root;
HLD(root);
/// ost
#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> ost;
/// rng
mt19937 rng(chrono::system_clock ::
    now().time_since_epoch().count());
/// cht; this one returns maximum
const ll IS_QUERY = -(1LL << 62);
struct line {
  ll m, b;
  mutable function <const line* ()> succ;
  bool operator < (const line &rhs) const {
    if (rhs.b != IS_QUERY) return m < rhs.m;
    const line *s = succ(); if (!s) return 0;
    ll x = rhs.m; return b - s -> b < (s -> m -
        m) * x;
  }
};
struct HullDynamic : public multiset <line> {
  bool bad (iterator y) {
    auto z = next(y);
    if (y == begin()) { if (z == end()) return 0;
      return y -> m == z -> m && y -> b <= z -> b;
    }
    auto x = prev(y);
    if (z == end()) return y -> m == x -> m && y
        -> b <= x -> b;
    return 1.0*(x->b-y->b)*(z->m-y->m) >=
        1.0*(y->b-z->b)*(y->m-x->m);
  }
  void insert_line (ll m, ll b) {
    auto y = insert({m, b});
    y -> succ = [=] {return next(y) == end() ? 0
        : &*next(y);};
    if (bad(y)) {erase(y); return;}
    while (next(y) != end() && bad(next(y)))
        erase(next(y));
    while (y != begin() && bad(prev(y)))
        erase(prev(y));
  }
  ll eval (ll x) {
    auto l = *lower_bound((line) {x, IS_QUERY});
    return l.m * x + l.b;
  }
};
///Li Chao Tree
using ftype = long long;
using point = pair<ftype,ftype>; // lines are
    represented as y = mx+c = (m,c)
//We evaluate functions as dot(y,(x,1)) where dot
    is the vector scalar product
const int MAXN = 1000000; //size of segment tree
ftype dot(const point &p1,const point &p2) {
    return p1.first*p2.first +
        p1.second*p2.second;
}
ftype f(const point& func,ftype x) {
    return dot(func,{x,1});
}
vector<point> line(MAXN*4+1); // this is segtree
const ftype li_chao_tree_infinity = LONG_LONG_MAX;
void init() {
    line.assign(line.size(),
        {0,li_chao_tree_infinity});
}
void add_line(int at,int L,int R,point new_line) {
    int mid = (L+R)/2;
    bool l = f(new_line,L)<f(line[at],L);
    bool m = f(new_line,mid)<f(line[at],mid);
    if(m) swap(new_line,line[at]);
    if(L==R) return;
    if(l!=m) add_line(at*2,L,mid,new_line);
    else add_line(at*2+1,mid+1,R,new_line);
}
ftype get(int at,int L,int R,ftype x) {
    if(L==R) return f(line[at],x);
```

```
    int mid = (L+R)/2;
    if(x<=mid) return
        min(f(line[at],x),get(at*2,L,mid,x));
    else return
        min(f(line[at],x),get(at*2+1,mid+1,R,x));
}
/// euler circuit
multiset<int> gset[N]; vector<int> circuit;
void eularcircuit(int src) {
  stack<int> curr_path;
  circuit.clear();
  if(gset[src].empty())return;
  curr_path.push(src);
  int curr_v = src;
  while (!curr_path.empty()) {
    if (!gset[curr_v].empty()) {
      curr_path.push(curr_v);
      auto it=gset[curr_v].begin();
      int next_v = *it;
      gset[curr_v].erase(next_v);
      gset[next_v].erase(curr_v);
      curr_v = next_v;
    }
    else {
      circuit.push_back(curr_v);
      curr_v = curr_path.top();
      curr_path.pop();
    }
  }
  reverse(circuit.begin(),circuit.end());
}
/// cutnode and bridge
void dfs(int v, int p=0){
    vis[v]=true; int children=0;
    tin[v]=low[v]=timer++;
    for(int to : g[v]) if(to!=p){
        if(vis[to]) low[v]=min(low[v],tin[to]);
        else{
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if(low[to]>tin[v]) BIRIJ(v, to);
            if(low[to]>=tin[v]&&p) KAT(v);
            ++children;
        }
    }
    if(p==0 && children>1) KAT(v);
}
/// online bridge
//Complexity = O(nlogn+mlogn); returns number of
    bridges
vector<int>par,dsu_2ecc,dsu_cc,dsu_cc_size;
int bridges, lca_iteration;
vector<int> last_visit;
void init(int n){
  par.resize(n), dsu_2ecc.resize(n);
  dsu_cc.resize(n), dsu_cc_size.resize(n);
  lca_iteration = 0, last_visit.assign(n, 0);
  for (int i=0; i<n; ++i){
    dsu_2ecc[i] = i;
    dsu_cc[i] = i;
    dsu_cc_size[i] = 1;
    par[i] = -1;
  }
  bridges = 0;
}
int find_2ecc(int v){
  if (v == -1) return -1;
  return dsu_2ecc[v] == v ? v : dsu_2ecc[v] =
      find_2ecc(dsu_2ecc[v]);
}
int find_cc(int v){
  v = find_2ecc(v);
  return dsu_cc[v] == v ? v : dsu_cc[v] =
      find_cc(dsu_cc[v]);
}
void make_root(int v){
  v = find_2ecc(v);
  int root = v, child = -1;
  while (v != -1){
    int p = find_2ecc(par[v]);
    par[v] = child;
    dsu_cc[v] = root;
    child = v, v = p;
  }
  dsu_cc_size[root] = dsu_cc_size[child];
}
void merge_path (int a, int b){
  ++lca_iteration;
  vector<int> path_a, path_b;
  int lca = -1;
  while (lca == -1){
    if (a != -1){
      a = find_2ecc(a);
      path_a.push_back(a);
      if(last_visit[a]==lca_iteration) lca=a;
      last_visit[a] = lca_iteration;
      a = par[a];
    }
    if (b != -1){
      path_b.push_back(b);
      b = find_2ecc(b);
      if(last_visit[b]==lca_iteration) lca=b;
      last_visit[b] = lca_iteration;
      b = par[b];
    }
  }
  for (int v : path_a){
    dsu_2ecc[v] = lca;
    if (v == lca) break;
    --bridges;
  }
  for (int v : path_b){
    dsu_2ecc[v] = lca;
    if (v == lca) break;
    --bridges;
  }
}
void add_edge(int a, int b){
  a = find_2ecc(a), b = find_2ecc(b);
  if(a==b) return;
  int ca = find_cc(a);
  int cb = find_cc(b);
  if (ca != cb){
    ++bridges;
    if (dsu_cc_size[ca] > dsu_cc_size[cb])
      swap(a, b), swap(ca, cb);
    make_root(a);
    par[a] = dsu_cc[a] = b;
    dsu_cc_size[cb] += dsu_cc_size[a];
  }
  else merge_path(a, b);
}
```

```
/// fast io burunduk
inline int readChar(); template <class T = int>
    inline T readInt(); template <class T> inline
    void writeInt( T x, char end = 0 ); inline
    void writeChar( int x ); inline void
    writeWord( const char *s ); /*---*/ static
    const int buf_size = 4096; inline int
    getChar() { static char buf[buf_size]; static
    int len = 0, pos = 0; if(pos==len) pos=0,
    len=fread(buf,1,buf_size,stdin); if(pos==len)
    return -1; return buf[pos++]; } inline int
    readChar() { int c = getChar(); while (c <=
    32) c = getChar(); return c; } template
    <class T> inline T readInt() { int s=1,
    c=readChar(); T x=0; if (c == '-') s=-1,
    c=getChar(); while('0'<=c && c<= '9')
    x=x*10+c-'0', c=getChar(); return s == 1 ? x
    : -x; } static int write_pos = 0; static char
    write_buf[buf_size]; inline void writeChar(
    int x ) { if (write_pos == buf_size)
    fwrite(write_buf, 1, buf_size, stdout),
    write_pos = 0; write_buf[write_pos++] = x; }
    template <class T> inline void writeInt( T x,
    char end ) { if (x<0) writeChar('-'), x=-x;
    char s[24]; int n = 0; while (x || !n)
    s[n++]='0'+x%10, x/=10; while (n--)
    writeChar(s[n]); if (end) writeChar(end); }
    inline void writeWord( const char *s ) {
    while (*s) writeChar(*s++); } struct Flusher
    { ~Flusher() { if (write_pos)
    fwrite(write_buf, 1, write_pos, stdout),
    write_pos=0; } } flusher;
/// GEO hullll
const int INF = 2e9+100;
struct Point {
    ld x,y;
    Point() {}
    Point(ld x,ld y) : x(x), y(y) {}
    Point(const Point &p) : x(p.x), y(p.y) {}
    bool operator < (const Point &p) const {
        return make_pair(x,y) <
            make_pair(p.x,p.y);}
    bool operator > (const Point &p) const {
        return make_pair(x,y) >
            make_pair(p.x,p.y);}
    Point rot90(){ return Point(-y, x);}
    Point rottheta(ld ang){
        return Point( x*cos(ang)-y*sin(ang),
            x*sin(ang)+y*cos(ang));}
    bool operator == (const Point &p) const {
        return fabs(x-p.x) < eps && fabs(y-p.y) <
            eps; }
};
Point operator +(Point a,Point b) {
  return Point(a.x+b.x,a.y+b.y);}
Point operator -(Point a,Point b) {
  return Point(a.x-b.x,a.y-b.y);}
Point operator *(Point a,ld b) {
  return Point(a.x*b,a.y*b);}
Point operator /(Point a,ld b){
  return Point(a.x/b,a.y/b);}
ld operator *(Point a,Point b){ ///dot
    return a.x*b.x+a.y*b.y;}
ld operator ^(Point a,Point b){ ///cross
    return a.x*b.y-a.y*b.x;}
bool ccw(Point p,Point q,Point r) {
    return ((q-p)^(r-q)) > 0 ;}
inline int dcmp (ld x) { if (fabs(x)<eps) return
    0; else return x<0?-1:1;}
ld getLength(Point a) {return
    sqrtl(a.x*a.x+a.y*a.y); }
ld dist2(Point a, Point b){ Point c = a-b; return
    c.x*c.x+c.y*c.y; }
ld getAngle(Point v) { return atan2(v.y,v.x); }
struct ConvexHull {
  vector<Point> hull, lower, upper;
  int n;
  ll cross(Point p, Point q, Point r){ return
      (q-p)^(r-q);}
  Point LineLineIntersection(Point p1, Point p2,
      Point q1, Point q2) {
    ll a1 = cross(q1, q2, p1), a2 = -cross(q1,
        q2, p2);
    return (p1 * a2 + p2 * a1) / (a1 + a2);
  }
  void init(vector<Point> &poly){
    hull.clear() ; lower.clear(); upper.clear() ;
    sort(poly.begin(),poly.end()) ;
    for(int i=0 ; i<poly.size(); i++) {
      while(lower.size()>=2 and
          !ccw(lower[lower.size()-2],lower.back(),
          poly[i]))
        lower.pop_back();
      lower.push_back(poly[i]) ;
    }
    for(int i=(int)poly.size()-1 ; i>=0; i--){
      while(upper.size() >= 2 and
          !ccw(upper[upper.size()-2],upper.back(),
          poly[i]))
        upper.pop_back();
      upper.push_back(poly[i]) ;
    }
    hull = lower ;
    for(int i=1 ; i+1 < upper.size(); i++)
        hull.push_back(upper[i]) ;
    n = hull.size();
  }
  int sign(ll x) {
    if (x < 0) return -1 ;
    return x > 0 ;
  }
  int crossOp(Point p, Point q, Point r) {
    ll c = (q-p)^(r-q) ;
    if (c < 0) return -1 ;
    return (c > 0) ;
  }
/// test if Point p is inside or on hull, if
    Point p is on any side a,b is the index of
    two endpoint of the segment
bool contain(Point p,int&a,int&b){
  if(p.x < lower[0].x || p.x > lower.back().x)
      return 0;
  int id = lower_bound(lower.begin(),
      lower.end(),Point(p.x,-INF)) -
      lower.begin();
  if(lower[id].x == p.x){
    if(lower[id].y > p.y) return 0;
  } else {
    if(crossOp(lower[id-1], lower[id], p) < 0)
        return 0;
```

```
    if(crossOp(lower[id-1], lower[id], p) == 0){
        a = id - 1; b = id; return 1;}
  }
  id = lower_bound(upper.begin(),
      upper.end(),Point(p.x,INF),
      greater<Point>())-upper.begin();
  if(upper[id].x == p.x){
    if(upper[id].y < p.y) return 0;
  } else {
    if(crossOp(upper[id-1], upper[id],p) < 0)
        return 0;
    if(crossOp(upper[id-1], upper[id],p) == 0) {
        a = id - 1 + lower.size() - 1; b = id +
        lower.size() - 1; return 1;}
  }
  return 1;
}
int find(vector<Point>&vec, Point dir){
  int l = 0 , r = vec.size();
  while(l+5<r){
    int L =(l*2+r)/3, R =(l+r*2)/3;
    if(vec[L]*dir > vec[R]*dir) r=R;
    else l=L;
  }
  int ret = l;
  for(int k=l+1; k<r; k++)
      if(vec[k]*dir>vec[ret]*dir) ret=k;
  return ret;
}
/// if there are rays coming from infinite
    distance in dir direction, the furthest Point
    of the hull is returned
int findFarest(Point dir){
       if(sign(dir.y) > 0 || sign(dir.y) == 0 &&
           sign(dir.x) > 0)
              return ( (int)lower.size()-1 +
                  find(upper,dir)) % n;
   else return find(lower,dir);
}
Point get(int l,int r,Point p1,Point p2){
       int sl = crossOp(p1,p2,hull[l%n]);
       while(l+1<r){
              int m = (l+r)>>1;
              if(crossOp(p1,p2,hull[m%n]) == sl)
                  l=m;
              else r=m;
       }
       return LineLineIntersection(p1,p2,
           hull[l%n],hull[(l+1)%n]);
}
///Intersection between a line and a convex
    polygon. O(log(n)) touching the hull does not
    count as intersection
vector<Point> Line_Hull_Intersection(Point p1,
    Point p2){
       int X = findFarest((p2-p1).rot90());
       int Y = findFarest((p1-p2).rot90());
       if(X>Y) swap(X,Y);
       if(crossOp(p1,p2,hull[X]) *
           crossOp(p1,p2,hull[Y]) < 0)
              return {get(X,Y,p1,p2),
                  get(Y,X+n,p1,p2)};
        else return {};
}
void update_tangent(Point p, int id, int&a,int&b){
       if(crossOp(p,hull[a],hull[id]) > 0) a = id;
       if(crossOp(p,hull[b],hull[id]) < 0) b = id;
}
void binary_search(int l,int r,Point
    p,int&a,int&b){
       if(l==r) return;
       update_tangent(p,l%n,a,b);
       int sl = crossOp(p,hull[l%n],
           hull[(l+1)%n]);
       while(l+1<r){
              int m = l+r>>1;
              if(crossOp(p,hull[m%n],
                  hull[(m+1)%n]) == sl) l=m;
              else r=m;
       }
       update_tangent(p,r%n,a,b);
}
void get_tangent(Point p,int&a,int&b){
       if(contain(p,a,b)) return ;
       a = b = 0;
       int id = lower_bound(lower.begin(),
           lower.end(),p) - lower.begin();
       binary_search(0,id,p,a,b),
           binary_search(id,lower.size(), p,a,b);
       id = lower_bound(upper.begin(),
           upper.end(),p,greater<Point>()) -
           upper.begin();
       binary_search((int)lower.size()-1, (int)
           lower.size()-1 + id,p,a,b);
       binary_search((int) lower.size()-1 +
           id,(int) lower.size()-1 +
           upper.size(),p,a,b);
}
};
/// halfplane randomized
bool LinesParallel(Point a, Point b, Point c,
    Point d) {
  return fabs((b-a)^(c-d)) < eps;
}
Point ComputeLineIntersection(Point a, Point b,
    Point c, Point d) {
  b=b-a; d=c-d; c=c-a;
  return a + b*(c^d)/(b^d);
}
struct HalfPlane {
    Point A, B;
    bool onHalfPlane (Point p) {
        return ((A-B)^(B-p)) >= -eps;
    }
};
HalfPlane getHalfPlane(double a, double b, double
    c) {
    Point A, B, C; /// ax+by+c>=0
    if (abs(a) < eps) A=Point(0, -c/b),
        B=Point(1, -c/b);
    else A = Point(-c/a, 0),B = Point(-(b+c)/a,
        1);
    C = A + (B-A).rot90();
    if (a*C.x+b*C.y+c < 0) swap(A, B);
    return {A, B};
}
double INFl = 1e100;
bool halfPlaneIntersection(vector<HalfPlane>
    planes) {
    int n = planes.size();
```

```
    shuffle(planes.begin(), planes.end(),
        mt19937(time(NULL)));
    Point best(INFl, INFl);
    for (int i=0; i<n; i++) {
        HalfPlane &hp = planes[i];
        if (hp.onHalfPlane(best)) continue;
        Point dir = hp.B - hp.A;
        dir = dir/sqrtl(dir.x*dir.x+dir.y*dir.y);
        Point X = hp.A+dir*INFl, Y = hp.A-dir*INFl;
        for (int j=0; j<i; j++) {
            HalfPlane &cp = planes[j];
            if (LinesParallel(hp.A, hp.B, cp.A,
                cp.B)) {
                if (!cp.onHalfPlane(hp.A)) return
                    false;
            }
            else {
                Point O =
                    ComputeLineIntersection(hp.A,
                    hp.B, cp.A, cp.B);
                bool bX = cp.onHalfPlane(X);
                bool bY = cp.onHalfPlane(Y);
                if (bX && bY) continue;
                else if (bX) Y = O;
                else if (bY) X = O;
                else return false;
            }
        }
        if (X.x+X.y <Y.x+Y.y) best = Y;
        else best = X;
    }
    return true;
}
/// smallest covering circle
struct Circle {
    Point o; ld r;
    Circle () {}
    Circle (Point o, ld r = 0): o(o), r(r) {}
    Point point(ld rad) {
    return Point(o.x+cos(rad)*r, o.y+sin(rad)*r);
}
}; /*---*/
int signn(x) {return (((x)>eps)-((x)<(-eps))); }
bool eq(ld a, ld b) { return abs(b-a) <= eps;}
```

```
ld len(Point a){ return sqrtl(a.x*a.x+a.y*a.y);}
bool is_colinear(Point a, Point b, Point c) {
  Point u = b-a, v = c-a; ld w = u^v;
  return eq(fabs(w)+fabs(w),0);
}
bool on (Point a, Point b, Point x) {
   return eq(len(x-a) + len(x-b), len(a-b));
}
Point isect(Point a, Point b, Point c, Point d) {
  Point u = (b-a), v = (d-c), z = (c-a);
  ld vz = v^z, vu = v^u;
  ld s = (vz) / (vu) * signn(vz*vu);
  return a + u*s;
}
bool in_circle(const Point& v, const Circle& C) {
  return len(v - C.o) <= C.r + eps;
}
Circle better(Circle A, Circle B) {
  if (A.r < B.r) return A; return B;
}
Circle find_circle(Point a) { return Circle(a,0);}
Circle find_circle(Point a, Point b) { return
    Circle((a+b)/2,len(a-b)/2);}
Circle find_circle(Point a, Point b, Point c,
    bool force_on=false){
  Point u = (b-a), v = (c-a); ld norm = u ^ v;
  Point uperp = u*norm, vperp = v*norm;
  Point ab = (a+b)/2, ac = (a+c)/2;
  if (is_colinear(a,b,c)){
    if (on(a,b,c)) return { (a+b)/2, len(a-b) / 2
        };
    if (on(a,c,b)) return { (a+c)/2, len(a-c) / 2
        };
    if (on(c,b,a)) return { (c+b)/2, len(c-b) / 2
        };
  }
  Point ans = isect(ab, ab + uperp, ac, ac +
      vperp);
  Circle C = Circle( ans, (len(ans-a) +
      len(ans-b) + len(ans-c)) / 3.0 ) ;
  if (force_on) return C;
  Circle C_ab = find_circle(a,b);
  Circle C_bc = find_circle(b,c);
  Circle C_ac = find_circle(a,c);
```

```
  if (in_circle(c, C_ab)) C = better(C, C_ab);
  if (in_circle(a, C_bc)) C = better(C, C_bc);
  if (in_circle(b, C_ac)) C = better(C, C_ac);
  return C;
}
Circle find_circle(Point* P, int N, int K) {
  if (K >= 3) return
      find_circle(P[N-1],P[N-2],P[N-3],true);
  if (N == 1) return find_circle(P[0]);
  if (N == 2) return find_circle(P[0],P[1]);
  int i = rand()%(N-K);
  swap(P[i], P[N-1-K]); swap(P[N-1-K], P[N-1]);
  auto C = find_circle(P, N-1, K);
  swap(P[N-1-K], P[N-1]); swap(P[i], P[N-1-K]);
  if (in_circle(P[i],C)) return C;
  swap(P[i], P[N-1-K]); C = find_circle(P, N,
      K+1);
  swap(P[i], P[N-1-K]); return C;
}
/// circle start
ld cross(Point p, Point q, Point r){ return
    (q-p)^(r-q);}
Point LineLineIntersection(Point p1, Point p2,
    Point q1, Point q2) {
    ld a1 = cross(q1, q2, p1), a2 = -cross(q1,
        q2, p2);
    return (p1 * a2 + p2 * a1) / (a1 + a2);
}
ld getDistanceToLine (Point p, Point a, Point b)
    { return fabs(((b-a)^(p-a))/getLength(b-a)); }
ld getDistanceToSegment (Point p, Point a, Point
    b) {
    if (a == b) return getLength(p-a);
    Point v1 = b - a, v2 = p - a, v3 = p - b;
    if (dcmp((v1*v2)) < 0) return getLength(v2);
    else if (dcmp((v1*v3)) > 0) return
        getLength(v3);
    else return fabs((v1^v2) / getLength(v1));
}
Circle CircumscribedCircle(Point p1, Point p2,
    Point p3) {
    ld Bx = p2.x - p1.x, By = p2.y - p1.y;
    ld Cx = p3.x - p1.x, Cy = p3.y - p1.y;
    ld D = 2 * (Bx * Cy - By * Cx);
```

```
    ld cx = (Cy*(Bx*Bx+By*By) - By*(Cx*Cx+Cy*Cy))
        / D+p1.x;
    ld cy = (Bx*(Cx*Cx+Cy*Cy) - Cx*(Bx*Bx+By*By))
        / D+p1.y;
    Point p = Point(cx, cy); return Circle(p,
        getLength(p1 - p));
}
Circle InscribedCircle(Point p1, Point p2, Point
    p3) {
    ld a = getLength(p2 - p3);
    ld b = getLength(p3 - p1);
    ld c = getLength(p1 - p2);
    Point p = (p1*a + p2*b + p3*c) / (a+b+c);
    return Circle(p, getDistanceToLine(p, p1,
        p2));
}
int LineCircleInter(Point p, Point q, Circle O,
    vector<Point>&sol){
    Point v = q - p;
    ld a = v.x, b = p.x - O.o.x, c = v.y, d = p.y
        - O.o.y;
    ld e = a*a+c*c, f = 2*(a*b+c*d), g =
        b*b+d*d-O.r*O.r;
    ld delta = f*f - 4*e*g;
    if (dcmp(delta) < 0) return 0;
    double t1, t2;
    if (dcmp(delta) == 0) {
        t1 = t2 = -f / (2 * e);
        sol.push_back(p + v * t1);
        return 1;
    }
    t1 = (-f - sqrt(delta))/(2*e);
        sol.push_back(p + v*t1);
    t2 = (-f + sqrt(delta))/(2*e);
        sol.push_back(p + v*t2);
    return 2;
}
int getCircleCircleIntersection (Circle o1,
    Circle o2, vector<Point>& sol) {
    ld d = getLength(o1.o - o2.o);
    if (dcmp(d) == 0) {
        if (dcmp(o1.r - o2.r) == 0) return -1;
        return 0;}
    if (dcmp(o1.r + o2.r - d)<0) return 0;
    if (dcmp(fabs(o1.r-o2.r) - d)>0) return 0;
    ld a = getAngle(o2.o - o1.o);
    ld da = acos((o1.r*o1.r + d*d - o2.r*o2.r) /
        (2*o1.r*d));
    Point p1 = o1.point(a-da), p2 =
        o1.point(a+da);
    sol.push_back(p1);
    if (p1 == p2) return 1;
    sol.push_back(p2);
    return 2;
}
bool LinesParallel(Point a, Point b, Point c,
    Point d) {
  return fabs((b-a)^(c-d)) < eps;
}
bool LinesCollinear(Point a, Point b, Point c,
    Point d) {
  return LinesParallel(a, b, c, d) &&
  fabs((a-b)^(a-c)) < eps && fabs((c-d)^(c-a)) <
      eps;
}
///checks whether segment AB and segment CD
    intersects
bool SegmentsIntersect(Point a, Point b, Point c,
    Point d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < eps || dist2(a, d) < eps ||
        dist2(b, c) < eps || dist2(b, d) < eps)
            return true;
    if ((c-a)*(c-b) > 0 && (d-a)*(d-b) > 0 &&
        (c-b)*(d-b) > 0)
      return false;
    return true;
  }
  if (((d-a)^(b-a))*((c-a)^(b-a)) > 0) return
      false;
  if (((a-c)^(d-c))*((b-c)^(d-c)) > 0) return
      false;
  return true;
}
ld radToPositive(ld rad){
  if (dcmp(rad)< 0) rad= ceil(-rad/pi)* pi+rad;
  if (dcmp(rad-pi)>= 0) rad-= floor(rad/pi)*pi;
  return rad;
}
Point normalUnit(Point A){
  ld L = getLength(A); return Point(-A.y/L,
      A.x/L);
}
struct Line {
  Point p, v; ld ang; Line() {}
  ld a,b,c; // ax+by+c=0
  Line(Point p,Point v):p(p),v(v){
    ang=atan2(v.y,v.x);
    Point q = p+v;
    if( dcmp(q.x-p.x) == 0 ) {
      a = 1; b = 0; c = -p.x;
    }
    else{
      ld m = (q.y-p.y)/(q.x-p.x);
      a = m; b = -1, c = p.y - m*p.x;
    }
  }
  Line(ld a_,ld b_,ld c_){
    a = a_,b = b_,c = c_; v = Point(-b,a) ;
    if (dcmp(a) == 0) p = Point(0,-c/b);
    else p = Point(-c/a,0);
  }
  double val(Point q) { return a*q.x + b*q.y + c;
      }
  bool operator < (const Line & L) const {return
      ang<L.ang;}
  Point point(ld t) { return p+v*t;}
};
Line LineTranslation(Line l, Point v){
  l.p = l.p+v; return l;
}
/// sol has center
/// Circle Through A Point And Tangent To A Line
    With Radius
void ctpr(Point p, Line l, ld r,vector<Point>&
    sol) {
  Point e = normalUnit(l.v);
  Line l1= LineTranslation(l,e*r),
      l2=LineTranslation(l,Point(0,0)-e*r);
  LineCircleInter(l1.p,l1.v,Circle(p,r), sol);
  LineCircleInter(l2.p,l2.v,Circle(p,r), sol);
}
```

```
/// Circle Tangent To Two Lines With Radius
void cttlwr(Line l1,Line l2, ld r, vector<Point>&
    sol) {
  Point e1 = normalUnit(l1.v), e2 =
      normalUnit(l2.v);
  Line L1[2]={ LineTranslation(l1,e1*r),
      LineTranslation(l1,e1*(-r)) },
  L2[2]={ LineTranslation(l2,e2*r),
      LineTranslation(l2,Point(0,0)-e2*r) };
  for( int i = 0; i < 2; i++ )
    for( int j = 0; j < 2; j++ )
      sol.push_back(LineLineIntersection(L1[i].p,
          L1[i].v, L2[j].p, L2[j].v));
}
/// Circle Tangent To Two Disjoint Circles With
    Radius
void cttdwr(Circle c1, Circle c2, ld r,
    vector<Point>& sol) {
   c1.r+=r; c2.r+=r;
       getCircleCircleIntersection(c1,c2,sol);
}
///tangent from p to circle c,returns dir vec
    from p to c
int getTangents(Point p, Circle c, vector<Point>
    &sol){
  Point u= c.o-p; ld dist = getLength(u);
  if (dist<c.r) return 0;
  else if (dcmp(dist-c.r)==0){
    sol.push_back(u.rot90());return 1;
  }
  else{
    ld ang = asin(c.r / dist);
    sol.push_back(u.rottheta(-ang));
    sol.push_back(u.rottheta(ang));
    return 2;
  }
}
///common tangent of two circle A and B; return
    the point on circles the tangent touches.
    ai-bi is a common tangent
int getTangents(Circle A, Circle B, vector<Point>
    &a, vector<Point> &b) {
  int cnt = 0;
  if (A.r<B.r) { swap(A,B),swap(a,b); }
  ld d2 = (A.o.x-B.o.x) * (A.o.x-B.o.x)+
      (A.o.y-B.o.y) * (A.o.y-B.o.y);
  ld rdiff = A.r - B.r; ld rsum = A.r + B.r;
  if (d2 < rdiff*rdiff) return 0;
  ld base = atan2(B.o.y-A.o.y, B.o.x-A.o.x);
  if (d2 == 0 && A.r == B.r) return -1;
  if (dcmp(d2 - rdiff*rdiff)==0) {
    a.push_back(A.point(base));
        b.push_back(B.point(base));
    return 1;
  }
  ld ang = acos( (A.r-B.r) / sqrt(d2));
  a.push_back(A.point(base + ang));
  b.push_back(B.point(base + ang));
  a.push_back(A.point(base - ang));
  b.push_back(B.point(base - ang));
  if (dcmp(d2-rsum*rsum)) {
    a.push_back(A.point(base));
    b.push_back(B.point(base+pi));
  }
  else if (dcmp(d2-rsum*rsum)==1) {
    ld ang = acos((A.r+B.r) / sqrt(d2));
    a.push_back(A.point(base + ang));
    b.push_back(B.point(pi+ base+ ang));
    a.push_back(A.point(base- ang));
    b.push_back(B.point(pi+ base- ang));
  }
  return (int)a.size();
}
/// closest point pair
ll ClosestPair(vector<pii> pts) {
    int n = pts.size(); sort(pts.begin(),
        pts.end());
    set<pii> s;
    ll best_dist = 1e18; int j = 0;
    for (int i = 0; i < n; ++i) {
        int d = ceil(sqrt(best_dist));
        while (pts[i].first - pts[j].first >=
            best_dist)
            s.erase({pts[j].second,
                pts[j].first}), j++;
        auto it1 = s.lower_bound({pts[i].second -
            d, pts[i].first});
        auto it2 = s.upper_bound({pts[i].second +
            d, pts[i].first});
        for (auto it = it1; it != it2; ++it) {
            int dx = pts[i].first - it->second;
            int dy = pts[i].second - it->first;
            best_dist = min(best_dist, 1LL*dx *dx
                + 1LL*dy * dy);
        }
        s.insert({pts[i].second, pts[i].first});
    }
    return best_dist;
}
/// voronoi
// ax + by = c
struct line{
        double a, b, c; Point u, d;
        line(double a, double b,double c):a(a),
            b(b), c(c) { }
  line(Point u_, Point d_) {
    u = u_, d = d_; a = d.y, b = -d.x, c =
        -u.y*d.x + u.x*d.y;
  }
bool operator < (const line &l)const{
bool flag1 = make_pair(a, b) > make_pair(0.0,
    0.0);
bool flag2 = make_pair(l.a, l.b) > make_pair(0.0,
    0.0);
if(flag1 != flag2) return flag1 > flag2;
long double t = ccw(Point(0, 0), Point(a, b),
    Point(l.a, l.b));
return dcmp(t) == 0 ? c*hypot(l.a, l.b) < l.c *
    hypot(a, b):t>0;
}
Point slope() { return Point(a, b); }
};
Point cross(line a, line b){
        double det = a.a * b.b - b.a * a.b;
        return Point((a.c * b.b - a.b * b.c) / det,
            (a.a * b.c - a.c * b.a) / det);
}
bool bad(line a, line b, line c){
        if(ccw(Point(0, 0), a.slope(), b.slope())
            <= 0) return false;
        Point crs = cross(a, b);
```

```
        return crs.x * c.a + crs.y * c.b >= c.c;
}
// ax + by <= c;
bool hpi(vector<line> v, vector<Point> &solution){
        sort(v.begin(), v.end());
        deque<line> dq;
        for(auto &i : v) {
    if(!dq.empty() && !dcmp(ccw(Point(0, 0),
         dq.back().slope(), i.slope()))) continue;
    while(dq.size() >= 2 && bad(dq[dq.size()-2],
         dq.back(), i)) dq.pop_back();
                while(dq.size() >= 2 &&
                    bad(i,dq[0],dq[1]))
                    dq.pop_front();
                dq.push_back(i);
        }
        while(dq.size() > 2 &&
            bad(dq[dq.size()-2], dq.back(),
            dq[0])) dq.pop_back();
        while(dq.size() > 2 &&
            bad(dq.back(),dq[0],dq[1]))
            dq.pop_front();
        vector<Point> tmp;
        for(int i=0; i< dq.size(); i++){
                line cur = dq[i], nxt =
                    dq[(i+1)%dq.size()];
                if(ccw(Point(0, 0), cur.slope(),
                    nxt.slope()) <= eps) return
                    false;
                tmp.push_back(cross(cur, nxt));
        }
        solution = tmp;
        return true;
}
vector< vector<Point> > voron(vector<Point>& P){
    double R = 1e9; int n = P.size();
    vector<vector<Point> > voronoi_diagram;
    for(int i = 0; i < n; i++){
        vector<line> lines;
        lines.push_back(line(1,0,R));
            lines.push_back(line(-1,0,R));
        lines.push_back(line(0,1,R));
            lines.push_back(line(0,-1,R));
        for(int j = 0; j < n; j++){
            if(P[i] == P[j]) continue;
            Point u = (P[i]+P[j]) * 0.5;
            Point dir = P[j]-P[i]; Point dir_90 =
                dir.rot90();
            Point v = u + dir_90;
            double a = dir_90.y, b = -dir_90.x, c
                = -u.y*dir_90.x + u.x*dir_90.y;
            lines.push_back(line(a,b,c));
        }
        vector<Point> polygon; hpi(lines, polygon);
        voronoi_diagram.push_back(polygon);
    }
    return voronoi_diagram;
}
/// point rotation
typedef long long lint;
typedef pair<lint, lint> pi;
struct pnt{
    int x, y, idx;
    bool operator<(const pnt &p)const{
        return pi(x, y) < pi(p.x, p.y); }
}a[5005];
struct line{
    int dx, dy, i1, i2;
};
vector<line> v;
int n, rev[5005]; lint p, q;
lint ccw(pnt a, pnt b, pnt c){ /// returns 2*area
    int dx1 = b.x - a.x, dy1 = b.y - a.y;
    int dx2 = c.x - a.x, dy2 = c.y - a.y;
    return abs(1ll * dx1 * dy2 - 1ll * dy1 * dx2);
}
void solv(vector<pnt>a){
    sort(a.begin(), a.end());
    int n = a.size();
    for(int i=0; i < n; i++)
        a[i].idx = i, rev[i] = i;
    for(int i=0; i<n; i++)
        for(int j=i+1; j<n; j++)
            v.push_back({a[j].x - a[i].x, a[j].y -
                a[i].y, a[i].idx, a[j].idx});
    sort(v.begin(), v.end(), [&](const line &a,
        const line &b){
        lint cw = 1ll * a.dx * b.dy - 1ll * b.dx *
            a.dy;
        if(cw != 0) return cw > 0;
        return pi(a.i1, a.i2) < pi(b.i1, b.i2);
    });
    for(int i=0; i<v.size(); i++){
        int c1 = rev[v[i].i1], c2 = rev[v[i].i2];
        if(c1 > c2) swap(c1, c2);
        /// now a is sorted perpendicular to
            a[c1]-a[c2]
        //solve(c1, c2);
        swap(a[c1], a[c2]); swap(rev[v[i].i1],
            rev[v[i].i2]);
    }
}
/// link cut tree
struct Node{
    int label; ll valoo;
    Node *p, *pp, *l, *r; /* parent, pathparent,
        left, right*/
    Node() { p = pp = l = r = 0; }
};
void update(Node *x){
    x->valoo = arr[x->label]; /// keep value,
        size, anything
    if(x->l) x->valoo += x->l->valoo; if(x->r)
        x->valoo += x->r->valoo;}
void rotr(Node *x){
    Node *y, *z;
    y = x->p, z = y->p;
    if((y->l = x->r)) y->l->p = y;
    x->r = y, y->p = x;
    if((x->p = z))
        if(y == z->l) z->l = x;
        else z->r = x;
    x->pp = y->pp, y->pp = 0, update(y);
}
void rotl(Node *x){
    Node *y, *z;
    y = x->p, z = y->p;
    if((y->r = x->l)) y->r->p = y;
    x->l = y, y->p = x;
    if((x->p = z))
        if(y == z->l) z->l = x;
```

```
        else z->r = x;
    x->pp = y->pp, y->pp = 0, update(y);
}
void splay(Node *x){
    Node *y, *z;
    while(x->p){
        y = x->p;
        if(y->p == 0)
            if(x == y->l) rotr(x);
            else rotl(x);
        else{
            z = y->p;
            if(y == z->l)
                if(x == y->l) rotr(y), rotr(x);
                else rotl(x), rotr(x);
            else
                if(x == y->r) rotl(y), rotl(x);
                else rotr(x), rotl(x);
        }
    }
    update(x);
}
Node *access(Node *x){
    splay(x);
    if(x->r) x->r->pp = x, x->r->p = 0, x->r = 0,
        update(x);
    Node *last = x;
    while(x->pp){
        Node *y = x->pp, last = y, splay(y);
        if(y->r) y->r->pp = y, y->r->p = 0;
        y->r = x, x->p = y, x->pp = 0, update(y),
            splay(x);
    }
    return last;
}
Node *root(Node *x){
    access(x); while(x->l) x = x->l; splay(x);
        return x;}
void cut(Node *x){
    access(x), x->l->p = 0, x->l = 0, update(x);}
void link(Node *x, Node *y){
    access(x), access(y), x->l = y, y->p = x,
        update(x);}
Node *lca(Node *x, Node *y){
    access(x); return access(y);}
class LinkCut{
    Node *x;
    public:
    LinkCut(int n){
        x = new Node[n];
        for(int i = 0; i < n; i++) x[i].label = i,
            update(&x[i]);
    }
    virtual ~LinkCut(){delete[] x;}
    void link(int u, int v){::link(&x[u], &x[v]);}
    void cut(int u){::cut(&x[u]);}
    void aksess(int u){::access(&x[u]);}
    ll getvaloo(int u){return (&x[u])->valoo;}
    int root(int u){return ::root(&x[u])->label;}
    void mekrut(int z){
        Node *u = &x[z];
               access(u), splay(u);
               if(u->l) u->l->p = 0, u->l->pp = u,
                   u->l = 0, update(u);
    }
    int depth(int u){return ::depth(&x[u]);}
    int lca(int u, int v){return ::lca(&x[u],
        &x[v])->label;}
};
/// dominator tree
struct ChudirBhai{
    int n, T;
    VVI g, tree, rg, bucket;
    VI sdom, par, dom, dsu, label, arr, rev;
    ChudirBhai(int n):
        n(n),g(n+1),tree(n+1),rg(n+1),bucket(n+1),
        sdom(n+1), par(n+1), dom(n+1), dsu(n+1),
            label(n+1), arr(n+1), rev(n+1), T(0){
        for(int i = 1; i <= n; i++) sdom[i] =
            dom[i] = dsu[i] = label[i] = i;
    }
    void addEdge(int u, int v) {
        g[u].push_back(v); }
    void dfs0(int u){
        T++; arr[u] = T, rev[T] = u;
        label[T] = T, sdom[T] = T, dsu[T] = T;
        for(int i = 0; i < g[u].size(); i++){
            int w = g[u][i];
            if(!arr[w]) dfs0(w), par[arr[w]] =
                arr[u];
            rg[arr[w]].push_back(arr[u]);
        }
    }
    int Find(int u, int x = 0){
        if(u == dsu[u]) return x? -1: u;
        int v = Find(dsu[u], x+1);
        if(v < 0) return u;
        if(sdom[label[dsu[u]]] < sdom[label[u]])
            label[u] = label[dsu[u]];
        dsu[u] = v;
        return x? v: label[u];
    }
    void Union(int u, int v) { dsu[v] = u; }
    VVI buildAndGetTree(int s){
        dfs0(s);
        for(int i = n; i >= 1; i--){
            for(int j = 0; j < rg[i].size(); j++)
                sdom[i] = min(sdom[i],
                sdom[Find(rg[i][j])]);
            if(i > 1) bucket[sdom[i]].push_back(i);
            for(int j = 0; j < bucket[i].size();
                j++){
                int w = bucket[i][j], v = Find(w);
                if(sdom[v] == sdom[w]) dom[w] =
                    sdom[w];
                else dom[w] = v;
            }
            if(i > 1) Union(par[i], i);
        }
        for(int i = 2; i <= n; i++){
            if(dom[i] != sdom[i]) dom[i] =
                dom[dom[i]];
            tree[rev[i]].push_back(rev[dom[i]]);
            tree[rev[dom[i]]].push_back(rev[i]);
        }
        return tree;
    }
};
/// gomori hu tree
ll INF = (1ULL<<50); struct edge { /* maintain
    order */ int src, dst; ll capacity; int rev;
    ll residue; }; struct graph { int n;
```

```
    vector<vector<edge>> adj; graph(int n = 0) :
    n(n), adj(n) { } void add_edge(int src, int
    dst, ll capacity) { adj[src].push_back({src,
    dst, capacity, (int)adj[dst].size()});
    adj[dst].push_back({dst, src, capacity,
    (int)adj[src].size()-1}); } vector<int>
    level, iter; ll augment(int u, int t, ll cur)
    { if(u==t) return cur; for(int &i = iter[u];
    i<adj[u].size(); ++i){ edge &e = adj[u][i];
    if(e.residue>0 && level[u]<level[e.dst]) { ll
    f = augment(e.dst, t, min(cur, e.residue));
    if(f>0){ e.residue -= f,
    adj[e.dst][e.rev].residue += f; return f; } }
    } return 0; } int bfs(int s, int t) {
    level.assign(n, -1); level[s] = 0; queue<int>
    Q; Q.push(s); while (!Q.empty()){ int u =
    Q.front(); Q.pop(); if(u==t) break; for (auto
    &e: adj[u]) if (e.residue>0 &&
    level[e.dst]<0) Q.push(e.dst),
    level[e.dst]=level[u]+1; } return level[t]; }
    ll max_flow(int s, int t){ for (int u = 0; u
    < n; ++u) for (auto &e: adj[u]) e.residue =
    e.capacity; ll flow = 0, itera = 0;
    while(bfs(s, t)>=0){ iter.assign(n, 0);
    for(ll f; (f=augment(s, t, INF))>0;)flow +=
    f; } return flow; } vector<edge> tree;
    vector<int> parent; void gomory_hu(){
    tree.clear(), parent.clear(),
    parent.resize(n); for(int i=0;i<n;++i)
    parent[i]=0; for(int u = 1; u < n; ++u) {
    tree.push_back({u, parent[u], max_flow(u,
    parent[u])}); for(int v = u+1; v < n; ++v)
    if(level[v]>=0 && parent[v]==parent[u])
    parent[v]=u; } } };
/// prime counting func
namespace pcf {
    const int MAXN = 1000010, MAX_PRIMES =
        1000010;
    const int PHI_N = 100000, PHI_K = 100;
    unsigned int ar[(MAXN >> 6) + 5] = {0};
    int phi_dp[PHI_N][PHI_K], len=0,
        primes[MAX_PRIMES], counter[MAXN];
    bitset <MAXN> isComp;
    void Sieve(int N) {
        int  i, j, sq = sqrtl(N);
        isComp[1] = true;
        for (i = 4; i <= N; i += 2) isComp[i] =
            true;
        for (i = 3; i <= sq; i += 2) if
            (!isComp[i])
          for (j = i * i; j <= N; j += i + i)
              isComp[j] = 1;
        for (i = 1; i <= N; i++) {
          if (!isComp[i]) primes[len++] = i;
          counter[i] = len;
        }
    }
    void init() {
        Sieve(MAXN - 1); int k , n , res;
        for (n = 0; n < PHI_N; n++) phi_dp[n][0] =
            n;
        for (k = 1; k < PHI_K; k++) for (n = 0; n
            < PHI_N; n++)
            phi_dp[n][k] = phi_dp[n][k - 1] -
                phi_dp[n / primes[k - 1]][k - 1];
    }
    long long phi(long long n, int k) {
        if (n < PHI_N && k < PHI_K) return
            phi_dp[n][k];
        if (k == 1) return ((++n) >> 1);
        if (primes[k - 1] >= n) return 1;
        return phi(n, k - 1) - phi(n / primes[k -
            1], k - 1);
    }
    long long Lehmer(long long n) { ///
        n^(2/3).(logn)^(1/3)
        if (n<MAXN) return counter[n];
        long long w, res=0; int i,j,a,b,c,lim;
        b = sqrt(n), c = Lehmer(cbrt(n)), a =
            Lehmer(sqrt(b)), b = Lehmer(b);
        res = phi(n, a) + (((b + a - 2) * (b - a +
            1)) >> 1);
        for(i = a; i < b; i++) {
            w = n / primes[i],lim=Lehmer(sqrt(w)),
                res-=Lehmer(w);
            if(i <= c) for (j = i; j < lim; j++)
                res += j, res -= Lehmer(w /
                    primes[j]);
        }
        return res;
    }
}
/* simplex: Given m x n matrix A, m-vector b,
    n-vector c, finds n-vector x such that, A x
    <= b (component-wise) maximizing < x , c >,
    <> is dot product */
const DOUBLE EPS = 1e-9;
struct LPSolver{
int m, n; VI B, N; 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)){
  for (int i = 0; i < m; i++) for (int j = 0; j <
      n; j++) D[i][j] = A[i][j];
  for (int i = 0; i < m; i++)
    B[i] = n+i, D[i][n] = -1, D[i][n+1] = b[i];
  for (int j = 0; j < n; j++) N[j] = j, D[m][j] =
      -c[j];
  N[n] = -1, D[m+1][n] = 1;
}
void Pivot(int r, int s){
  for (int i = 0; i < m+2; i++) if (i != r)
    for (int j = 0; j < n+2; j++) if (j != s)
      D[i][j] -= D[r][j] * D[i][s] / D[r][s];
  for (int j = 0; j < n+2; j++) if (j != s)
      D[r][j] /= D[r][s];
  for (int i = 0; i < m+2; i++) if (i != r)
      D[i][s] /= -D[r][s];
  D[r][s] = 1.0 / D[r][s];
  swap(B[r], N[s]);
}
bool Simplex(int phase){
  int x = phase==1 ? m+1:m;
  while (true){
    int s = -1;
    for (int j = 0; j <= n; j++){
      if (phase == 2 && N[j] == -1) continue;
      if (s == -1 || D[x][j] < D[x][s] || D[x][j]
          == D[x][s] && N[j] < N[s]) s = j;}
    if (D[x][s] >= -EPS) return true;
    int r = -1;
    for (int i = 0; i < m; i++){
```

```
      if (D[i][s] <= 0) continue;
      if (r == -1 || D[i][n+1] / D[i][s] <
          D[r][n+1] / D[r][s] || D[i][n+1] /
          D[i][s] == D[r][n+1] / D[r][s] && B[i]
          < B[r]) r = i;}
    if (r == -1) return false;
    Pivot(r, s);
  }
}
DOUBLE Solve(VD &x){
  int r = 0;
  for (int i = 1; i < m; i++) if (D[i][n+1] <
      D[r][n+1]) r = i;
  if (D[r][n+1] <= -EPS){
    Pivot(r, n);
    if (!Simplex(1) || D[m+1][n+1] < -EPS) return
        -numeric_limits<DOUBLE>::infinity();
    for (int i = 0; i < m; i++) if (B[i] == -1){
      int s = -1;
      for (int j = 0; j <= n; j++)
        if (s == -1 || D[i][j] < D[i][s] ||
            D[i][j] == D[i][s] && N[j] < N[s]) s =
            j;
      Pivot(i, s);
    }
  }
  if (!Simplex(2)) return
      numeric_limits<DOUBLE>::infinity();
  x = VD(n);
  for (int i = 0; i < m; i++) if (B[i] < n)
      x[B[i]] = D[i][n+1];
  return D[m][n+1];
}
};
/// z algo
vector<int> z_function(string s) {
    int n = (int)s.length(); vector<int> z(n);
    for (int i=1, l=0, r=0; i<n; ++i) {
        if (i <= r) z[i] = min (r - i + 1, z[i -
            l]);
        while(i+z[i]<n && s[z[i]]==s[i+z[i]])
            ++z[i];
        if(i+z[i]-1 > r) l=i, r = i+z[i]-1;
    }
    return z;
}
/// manacher
vector<int> d1(n), d2(n);
for(int i=0, l=0, r=-1; i<n; i++){
    int k =(i>r) ? 1:min(d1[l+r-i], r-i+1);
    while(0<=i-k && i+k<n && s[i-k] == s[i+k])
        k++;
    d1[i] = k--;
    if(i+k>r) l=i-k, r=i+k;
}
for(int i=0, l=0, r=-1; i<n; i++){
    int k =(i>r) ? 0:min(d2[l+r-i+1], r-i+1);
    while(0<=i-k-1 && i+k<n && s[i-k-1] ==
        s[i+k]) k++;
    d2[i] = k--;
    if(i+k>r) l=i-k-1, r=i+k;
}
/// aho corasick
const int K = 26;
struct Vertex {
    int next[K]; bool leaf=false; int p = -1;
    char pch; int link = -1; int go[K];
    Vertex(int p=-1, char ch='$') : p(p), pch(ch)
        {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};
vector<Vertex> t(1);
void add_string(string const& s) {
    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 = true;
}
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];
}
/// suffix array
const int N = 2e4 + 5; const int ALPHA = 128, LOG
    = 20; struct SuffixArray { int
    sa[N],data[N],rnk[N],height[N],n; int
    wa[N],wb[N],wws[N],wv[N]; int lg[N],
    rmq[N][LOG], rev_sa[N]; int cmp(int *r,int
    a,int b,int l) { return (r[a]==r[b]) &&
    (r[a+l]==r[b+l]); } void DA(int *r,int
    *sa,int n,int m) { int i,j,p,*x=wa,*y=wb,*t;
    for(i=0; i<m; i++) wws[i]=0; for(i=0; i<n;
    i++) wws[x[i]=r[i]]++; for(i=1; i<m; i++)
    wws[i]+=wws[i-1]; for(i=n-1; i>=0; i--)
    sa[--wws[x[i]]]=i; for(j=1,p=1; p<n;
    j*=2,m=p) { for(p=0,i=n-j; i<n; i++)
    y[p++]=i; for(i=0; i<n; i++) if(sa[i]>=j)
    y[p++]= sa[i]-j; for(i=0; i<n; i++)
    wv[i]=x[y[i]]; for(i=0; i<m; i++) wws[i]=0;
    for(i=0; i<n; i++) wws[wv[i]]++; for(i=1;
    i<m; i++) wws[i]+= wws[i-1]; for(i=n-1; i>=0;
    i--) sa[--wws[wv[i]]]=y[i]; for(t=x,x=y,
    y=t,p=1, x[sa[0]]=0,i=1; i<n; i++) x[sa[i]]=
    cmp(y,sa[i-1], sa[i],j)? p-1:p++; } } void
    calheight(int *r,int *sa,int n) { int
    i,j,k=0; for(i=1; i<=n; i++) rnk[sa[i]]=i;
    for(i=0; i<n; height[rnk[i++]]=k)
    for(k?k--:0,j=sa[rnk[i]-1]; r[i+k]==r[j+k];
    k++); } void suffix_array (string &A) { n =
    A.size(); for(int i=0; i<max(n+5,ALPHA); i++)
    sa[i]= data[i]=rnk[i]=height[i] =wa[i]=wb[i]=
    wws[i]=wv[i]=0; for (int i = 0; i < n; i++)
```

```
    data[i] = A[i]; DA(data,sa,n+1,ALPHA);
    calheight(data,sa,n); for(int i = 0; i < n;
    i++) sa[i] = sa[i+1], height[i] =
    height[i+1], rev_sa[sa[i]] = i;
    range_lcp_init(); } void range_lcp_init() {
    for(int i = 0; i < n; i++) rmq[i][0] =
    height[i]; for(int j = 1; j < LOG; j++) {
    for(int i = 0; i < n; i++) { if (i+(1<<j)-1 <
    n) rmq[i][j] =
    min(rmq[i][j-1],rmq[i+(1<<(j-1))][j-1]); else
    break; } } lg[0] = lg[1] = 0; for(int i = 2;
    i <= n; i++) { lg[i] = lg[i/2] + 1; } } int
    query_lcp(int l, int r) { assert(l <= r);
    assert(l>=0 && l<n && r>=0 && r<n); if(l ==
    r) return n-sa[l]; l++; int k = lg[r-l+1];
    return min(rmq[l][k], rmq[r-(1<<k)+1][k]); }
    };
/// automaton
struct vertex { int link,len,cnt=0,d=0; int
    next[26]; vertex() { link = -1; len = 0;
    memset(next,-1,sizeof next); } }; vertex
    sa[N*2]; int last = 0,sz = 1; void
    add_char(char c) { c-='a'; int cur = sz++;
    sa[cur].len = sa[last].len+1; int u = last;
    while(u!=-1&&sa[u].next[c]==-1)sa[u].next[c]
    = cur, u = sa[u].link; if(u==-1) sa[cur].link
    = 0; else { int v = sa[u].next[c];
    if(sa[u].len+1 == sa[v].len) sa[cur].link =
    v; else { int nw = sz++; sa[nw].link =
    sa[v].link; sa[nw].len = sa[u].len+1;
    memcpy(sa[nw].next,sa[v].next,sizeof
    sa[v].next); while(u!=-1&&sa[u].next[c]==v)
    sa[u].next[c] = nw, u = sa[u].link;
    sa[cur].link = sa[v].link = nw; } } last =
    cur; } /*cnt is the number of instances of an
    equivalence class. init cnt with 1 except for
    clones and starting node d is the number of
    instances of an equivalence class being a
    prefix to count distinct replace all the cnt
    with 1*/ void pre() { vector<vector<int>>
    v(sz+1); for(int i=0; i<sz;
    i++)v[sa[i].len].push_back(i); for(int i=sz;
    i>=0; i--) for(auto x:v[i])
    if(x)sa[sa[x].link].cnt+=sa[x].cnt;
    sa[0].cnt=0; /*ignoring empty substring*/
    for(int i=sz; i>=0; i--) { for(auto x:v[i]) {
    sa[x].d=sa[x].cnt; for(auto
    u:sa[x].next)sa[x].d+=sa[u.se].d; } } }
///kmp
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]);
  }
}
}
/// Counting the number of occurrences of each
    prefix
vector<int> ans(n + 1);
for (int i = 0; i < n; i++)
  ans[pi[i]]++;
for (int i = n-1; i > 0; i--)
  ans[pi[i-1]] += ans[i];
for (int i = 0; i <= n; i++)
  ans[i]++;
/// simpson integration
const int N = 1000 * 1000; /// number of steps
    (already multiplied by 2)
double simpson_integration(double a, double b){
    double h = (b - a) / N;
    double s = f(a) + f(b); /// a = x_0 and b =
        x_2n
    for (int i = 1; i <= N - 1; ++i) {
        double x = a + h * i;
        s += f(x) * ((i & 1) ? 4 : 2);
    }
    s *= h / 3; return s;
}
/// mohsin wavelet
int a[N];
struct wavelet_tree{
    int lo, hi;
    wavelet_tree *l=0, *r=0;
    vector<int> b, c;
    wavelet_tree(int *from, int *to, int x, int
        y){
        lo = x, hi = y;
        if( from >= to) return;
        if( hi == lo ){
            b.reserve(to-from+1), b.pb(0);
            c.reserve(to-from+1), c.pb(0);
            for(auto it = from; it != to; it++){
                b.pb(b.back() + 1);
                c.pb(c.back()+*it);
            }
            return ;
        }
        int mid = (lo+hi)/2;
        auto f = [mid](int x){
            return x <= mid; };
        b.reserve(to-from+1), b.pb(0);
        c.reserve(to-from+1), c.pb(0);
        for(auto it = from; it != to; it++){
            b.pb(b.back() + f(*it));
            c.pb(c.back() + *it);
        }
        auto pivot = stable_partition(from, to, f);
        l = new wavelet_tree(from, pivot, lo, mid);
        r = new wavelet_tree(pivot, to, mid+1, hi);
    }
    void swapadjacent(int i){ /// i with i+1
        if(lo == hi) return ;
        b[i]= b[i-1] + b[i+1] - b[i];
        c[i] = c[i-1] + c[i+1] - c[i];
```

```
        if( b[i+1]-b[i] == b[i] - b[i-1]){
            if(b[i] -b[i-1]) return this->l->
                swapadjacent(b[i]);
            else return this->r->
                swapadjacent(i-b[i]); }
        else return ;
    }
    int kth(int l, int r, int k){
        if(l > r) return 0;
        if(lo == hi) return lo;
        int inLeft = b[r] - b[l-1];
        int lb = b[l-1], rb = b[r];
        if(k <= inLeft) return this->l->kth(lb+1,
            rb, k);
        return this->r->kth(l-lb, r-rb, k-inLeft);
    }
    int LTE(int l, int r, int k) {
        if(l > r or k < lo) return 0;
        if(hi <= k) return r - l + 1;
        int lb = b[l-1], rb = b[r];
        return this->l->LTE(lb+1, rb, k) +
            this->r->LTE(l-lb, r-rb, k);
    }
    int count(int l, int r, int k){
        if(l > r or k < lo or k > hi) return 0;
        if(lo == hi) return r - l + 1;
        int lb = b[l-1], rb = b[r], mid =
            (lo+hi)/2;
        if(k <= mid) return this->l->count(lb+1,
            rb, k);
        return this->r->count(l-lb, r-rb, k);
    }
    int sumk(int l, int r, int k) { /// sumof LTE
        k
        if(l > r or k < lo) return 0;
        if(hi <= k) return c[r] - c[l-1];
        int lb = b[l-1], rb = b[r];
        return this->l->sumk(lb+1, rb, k) +
            this->r->sumk(l-lb, r-rb, k);
    }
    ~wavelet_tree(){
    if(l) delete l; if(r) delete r; }
};
///call with wavelet_tree T(a+1, a+n+1, 1, MAX);
```

```
/// merge sort tree bild
merge(tree[2*cur].begin(), tree[2*cur].end(),
    tree[2*cur+1].begin(), tree[2*cur+1].end(),
    back_inserter(tree[cur]));
/// dynamic connectivity
/// {1, {}} add, /// {2, {}} remov, {3 } query
    num of compos; define F first S second
struct DynamicConnectivity { struct edge{ int
    a,b,l,r; }; vector<int> ret,tq,id,is;
    vector<vector<int> > g; int dfs(int x, int c)
    { id[x]=c; int r=is[x]; for (int nx:g[x]) if
    (!id[nx]) r|=dfs(nx, c); return r; } void
    go(int l, int r, int n, int out, vector<edge>
    es) { vector<edge> nes; for (int
    i=1;i<=n;i++) g[i].clear(), id[i]=0, is[i]=0;
    for (auto e:es) { if (e.l>r||e.r<l||e.a==e.b)
    continue; if (e.l<=l&&r<=e.r)
    g[e.a].push_back(e.b), g[e.b].push_back(e.a);
    else nes.push_back(e), is[e.a]=1, is[e.b]=1;
    } int i2=1; for (int i=1;i<=n;i++) { if
    (g[i].size()>0||is[i]){ if (!id[i]) { int
    a=dfs(i, i2); if (!a) out++; else i2++; } }
    else out++; } for (auto&e:nes) e.a=id[e.a],
    e.b=id[e.b]; if (l==r){ if(tq[l])
    ret[tq[l]-1]=out+i2-1;} else { int m=(l+r)/2;
    go(l, m, i2-1, out, nes), go(m+1, r, i2-1,
    out, nes); } } vector<int> solve(int n,
    vector<pair<int, pair<int, int> > > queries)
    { int qs=0; vector<edge> es; map<pair<int,
    int>, int> ae; tq.resize(queries.size()),
    id.resize(n+1), is.resize(n+1),
    g.resize(n+1); for (int
    i=0;i<(int)queries.size();i++) { auto
    q=queries[i]; if (q.S.F>q.S.S) swap(q.S.F,
    q.S.S); if (q.F==1) { if(ae[q.S]==0)
    ae[q.S]=i+1; } else if(q.F==2){ if(ae[q.S])
    es.push_back({q.S.F, q.S.S, ae[q.S]-1, i}),
    ae[q.S]=0; } else if (q.F==3) tq[i]=1+qs++; }
    for (auto e:ae) if (e.S) es.push_back({e.F.F,
    e.F.S, e.S-1, (int)queries.size()});
    ret.resize(qs); if ((int)queries.size()>0)
    go(0, (int)queries.size()-1, n, 0, es);
    return ret; } };
/// implicit segtree 0 indexed
```

```
struct Node {
    ll sum; Node *l, *r;
    Node() : sum(0), l(NULL), r(NULL) {}
};
void add(Node *v, int l, int r, int q_l, int q_r,
    ll val) {
    if (l > r || q_r < l || q_l > r) return;
    if (q_l <= l && r <= q_r) {
        v -> sum += val; return; }
    int mid = (l + r)/2;
    if (v->l == NULL) v -> l = new Node();
    if (v->r == NULL) v -> r = new Node();
    add(v->l, l, mid, q_l, q_r, val);
    add(v->r, mid + 1, r, q_l, q_r, val);
}
ll get(Node *v, int l, int r, int pos) {
    if (!v || l > r || pos < l || pos > r) return
        0;
    if (l == r) return v -> sum;
    int mid = (l + r)/2;
    return v -> sum + get(v -> l, l, mid, pos) +
        get(v -> r, mid + 1, r, pos);
}
/// sos dp
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^n lup chalano
for (int mask = 0; mask < (1<<n); mask++){
        F[mask] = A[0];
    for(int i=mask; i>0; i=(i-1) & mask) F[mask]
        += A[i];
}
/**Dinic Algorithm (V^2 * E)***/
struct Edge { int u, v; LL cap, flow; Edge() {}
    Edge(int u, int v, LL cap): u(u), v(v),
    cap(cap), flow(0) {} }; struct Dinic { int N;
    vector<Edge> edges; vector<vector<int>> adj;
    vector<int> d, pt; Dinic(int N): N(N),
    edges(0), adj(N), d(N), pt(N) {} void
    AddEdge(int u, int v, LL cap) { if (u != v) {
```

```
    edges.emplace_back(u, v, cap);
    adj[u].emplace_back(edges.size() - 1);
    edges.emplace_back(v, u, 0);
    adj[v].emplace_back(edges.size() - 1); } }
    bool BFS(int S, int T) { queue<int> q({S});
    fill(d.begin(), d.end(), N + 1); d[S] = 0;
    while(!q.empty()) { int u = q.front();
    q.pop(); if (u == T) break; for (int k:
    adj[u]) { Edge &e = edges[k]; if (e.flow <
    e.cap && d[e.v] > d[e.u] + 1) d[e.v] = d[e.u]
    + 1, q.emplace(e.v); } } return d[T] != N +
    1; } LL DFS(int u, int T, LL flow = -1) { if
    (u == T || flow == 0) return flow; for (int
    &i = pt[u]; i < adj[u].size(); ++i) { Edge &e
    = edges[adj[u][i]]; Edge &oe =
    edges[adj[u][i]^1]; if (d[e.v] == d[e.u] + 1)
    { LL amt = e.cap - e.flow; if (flow != -1 &&
    amt > flow) amt = flow; if (LL pushed =
    DFS(e.v, T, amt)) { e.flow += pushed; oe.flow
    -= pushed; return pushed; } } } return 0; }
    LL MaxFlow(int S, int T) { LL total = 0;
    while (BFS(S, T)) { fill(pt.begin(),
    pt.end(), 0); while (LL flow = DFS(S, T))
    total += flow; } return total; } };
/**mincost max flo expected V*E^2 **/
namespace mcmf{
const int MAX = 31000; const ll INF = 1LL << 60;
ll cap[MAX], flow[MAX], cost[MAX], dis[MAX];
int n, m, s, t, Q[MAX*10], adj[MAX], link[MAX],
    last[MAX], from[MAX], visited[MAX];
void init(int nodes, int source, int sink){
  m = 0, n = nodes, s = source, t = sink;
  for (int i = 0; i <= n; i++) last[i] = -1;
}
void addEdge(int u, int v, ll c, ll w){
  adj[m] = v, cap[m] = c, flow[m] = 0, cost[m] =
      +w,
  link[m] = last[u], last[u] = m++;
  adj[m] = u, cap[m] = 0, flow[m] = 0,
  cost[m] = -w, link[m] = last[v], last[v] = m++;
}
bool spfa() {
  int i, j, x, f = 0, l = 0;
  for (i = 0; i <= n; i++) visited[i] = 0, dis[i]
      = INF;
  dis[s] = 0, Q[l++] = s;
  while (f < l) {
    i = Q[f++];
    for (j = last[i]; j != -1; j = link[j]) {
      if (flow[j] < cap[j]) {
      x = adj[j];
        if (dis[x] > dis[i] + cost[j]) {
          dis[x] = dis[i] + cost[j], from[x] = j;
          if (!visited[x]) {
              visited[x] = 1;
              if (f && rand() & 7) Q[--f] = x;
              else Q[l++] = x;
          }
        }
      }
    }
    visited[i] = 0;
  }
  return (dis[t] != INF);
}
pair <ll, ll> solve() {
    int i, j; ll mincost = 0, maxflow = 0;
    while (spfa()) {
        ll aug = INF;
        for(i=t,j=from[i];i!=s;i=adj[j^1],j =
            from[i])
            aug = min(aug, cap[j] - flow[j]);
        for(i=t,j=from[i];i!=s;i=adj[j^1],j=from[i])
            flow[j] += aug, flow[j ^ 1] -= aug;
        maxflow += aug, mincost += aug * dis[t];
    }
    return make_pair(mincost, maxflow);
}
}
/// hopcroft karp
struct Hopcroft_Karp { /// // N = left node +
    right node
  const int
      NIL=0,INF=(1<<28),match[N],dist[N],n,m;
  vector <int> G[N] ;
  void init(int lft , int rgt) {
    n = lft , rgt = m ;
    for (int i = 0 ; i <= n+m+1 ; i++)
        G[i].clear() ;
  }
  void addEdge(int u , int v){ //u = left node
      from 1 to n
    G[u].push_back(v+n) ;//v = right node 1 to m
  }
  bool bfs(){
    queue <int> Q;
    for(int i = 1; i <= n ;i++) {
      if(match[i]==NIL) dist[i] = 0,Q.push(i);
      else dist[i] = INF;
    }
    dist[NIL] = INF;
    while(!Q.empty()) {
      int u = Q.front(); Q.pop();
      if(u!=NIL) {
        for(int i = 0; i < G[u].size(); i++) {
          int v = G[u][i];
          if(dist[match[v]]==INF) {
            dist[match[v]] = dist[u] + 1;
            Q.push(match[v]);
          }
        }
      }
    }
    return (dist[NIL]!=INF);
  }
  bool dfs(int u) {
    if(u!=NIL) {
      for(int i = 0; i < G[u].size() ; i++) {
        int v = G[u][i] ;
        if(dist[match[v]] == dist[u]+1) {
        if(dfs(match[v])) {
          match[v] = u;
          match[u] = v;
          return true;
        }
        }
      }
      dist[u] = INF;
      return false;
    }
    return true;
```

```
  }
  int hopcroft_karp() {
    memset( dist, 0, sizeof dist );
    memset( match, 0, sizeof match );
    int matching = 0 ;
    while(bfs())
      for(int i = 1 ; i <= n; i++)
        if(match[i]==NIL && dfs(i)) matching++ ;
    return matching;
  }
  void VertexCover(vector<int>&color){
  /*color[i]=1 -> i in min cover*/
      hopcroft_karp();
    vector< vector<int> > g(R+L+1) ; queue <int>
        Q;
    vector <int> vis(L+R+1,0) ;
    for(int u = 1 ; u <= L ; u++) {
      if (match[u]==0) Q.push(u) , vis[u] = 1;
      for(int i = 0 ; i < G[u].size(); i++) {
        int v = G[u][i] ;
        if (match[u] == v) g[v].push_back(u);
        else g[u].push_back(v);
      }
    }
    while(Q.size()) {
      int u = Q.front() ; Q.pop();
      for(int i = 0 ; i < g[u].size() ; i++) {
        int v = g[u][i] ;
        if (vis[v] == 0) vis[v] = 1 , Q.push(v);
      }
    }
    color.resize(R+L+1) ;
    for(int i = 1 ; i <= L ; i++) color[i] =
        (!vis[i]);
    for(int i = L+1 ; i <= L+R ; i++) color[i] =
        vis[i];
  }
/// call init() , then addEdge , then
    hopcroft_karp()
};
/**min Weighted Bipartite Matching (V^3)---Index
    from 1**/
/// Input:n*m sized cost matrix,where n <= m,
    a[i][j]; Output:m sized vector,ith worker
    will do ans[i]th work
#define MAX 55
#define INF 12345678
int a[MAX][MAX], n, m, x[MAX], y[MAX];
vector<int> hungarain(){
  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);
  for (int j=1; j<=m; ++j) ans[p[j]] = j;
  return ans;
}
/// demand flo
cap2(u -> v) = cap(u -> v) - lo(u -> v)
cap2(supsorc -> v) = sum of lo(u -> v)
cap2(u -> supsink) = sum of lo(u -> v)
cap2(sink -> sorc) = inf
/// blossom; 1-based Vertex index
const int MAXN = 2020 + 1;
struct GM {
  int vis[MAXN], par[MAXN], orig[MAXN],
      match[MAXN],
  aux[MAXN], t, N;
  vector<int> conn[MAXN]; queue<int> Q;
  void addEdge(int u, int v) {
    conn[u].push_back(v); conn[v].push_back(u); }
  void init(int n) {
    N = n; t = 0;
    for(int i=0; i<=n; ++i)
      conn[i].clear(), match[i] = aux[i] = par[i]
          = 0;
  }
  void augment(int u, int v) {
    int pv = v, nv;
    do { pv = par[v]; nv = match[pv]; match[v] =
        pv;
      match[pv] = v; v = nv;
    }
    while(u != pv);
  }
  int lca(int v, int w){
    ++t;
    while(true) {
      if(v) {
        if(aux[v] == t) return v;
        aux[v] = t; v = orig[par[match[v]]];
      }
      swap(v, w);
    }
  }
  void blossom(int v, int w, int a) {
    while(orig[v] != a) {
      par[v] = w; w = match[v];
      if(vis[w] == 1) Q.push(w), vis[w] = 0;
      orig[v] = orig[w] = a; v = par[w];
    }
  }
  bool bfs(int u){
    fill(vis+1, vis+1+N, -1);
    iota(orig + 1, orig + N + 1, 1); Q =
        queue<int> ();
    Q.push(u); vis[u] = 0;
    while(!Q.empty()) {
```

```
      int v = Q.front(); Q.pop();
      for(int x: conn[v]){
        if(vis[x] == -1) {
          par[x] = v; vis[x] = 1;
          if(!match[x]) return augment(u, x), true;
          Q.push(match[x]); vis[match[x]] = 0;
        }
        else if(vis[x] == 0 && orig[v] != orig[x]){
            int a = lca(orig[v], orig[x]);
            blossom(x, v, a); blossom(v, x, a);
        }
      }
    }
    return false;
  }
  int Match(){
    int ans = 0;
    vector<int> V(N-1); iota(V.begin(), V.end(),
        1);
    shuffle(V.begin(), V.end(), mt19937(0x94949));
    for(auto x: V) if(!match[x])
      for(auto y: conn[x]) if(!match[y]){
        match[x] = y, match[y] = x; ++ans; break;
      }
    for(int i=1; i<=N; ++i) if(!match[i] &&
        bfs(i)) ++ans;
    return ans;
  }
};
// global minimum cut, complexity: O(n^3);
    self-loop ignored.
ll n,m, g[maxN][maxN];//1-indexed,initialised
    with 0
bool vis[maxN];
ll dis[maxN],v[maxN];
ll stoer_wagner() {
  ll i, j, now,ans=inf;
  for(i=0; i<n; i++) v[i]=i+1;
  while(n>1) {
    for(now=0,i=1; i<n; i++)dis[v[i]]=0;
    for(i=1; i<n; i++) {
      swap(v[now],v[i-1]);
      for(now=j=i; j<n; j++) {
        dis[v[j]]+=g[v[i-1]][v[j]];
        if(dis[v[now]]<dis[v[j]])now=j;
      }
    }
    ans=min(ans,dis[v[now]]);
    for(j=0; j<n; j++)
      g[v[j]][v[now-1]]= g[v[now-1]][v[j]]+=
          g[v[j]][v[now]] ;
    v[now]=v[--n];
  }
  return ans ;
}
/// sparse table
void buildSparseforMIN(){
    for(int i=1;i<=n;i++) K[i][0]=A[i];
    for(int j=1; (1<<j)<=n ;j++)
        for(int i=1; (i+ (1<<j)-1) <=n; i++)
            K[i][j]=min(K[i][j-1],K[i +
                (1<<(j-1))][j - 1]);
}
int MIN(int i,int j){
    int k=log2(j-i+1);
    return min(K[i][k],K[j-(1<<k)+1][k]);
}
/// treap, 0 indexing
struct node {
    int prior, size;
    ll val, sum, lazy, rev;
    node *l, *r;
    node(int v = 0) {
        val = sum = v; lazy = 0; rev = 0;
        prior = rand();
        size = 1; l = r = NULL;
    }
} *root;
typedef node* pnode;
int sz(pnode t) { return t ? t -> size : 0;}
void upd_sz(pnode t) { if(t) t -> size = sz(t ->
    l) + 1 + sz(t -> r);}
void push(pnode t) {
    if(!t) return;
    if(t->lazy){
        t -> val += t -> lazy;
        t -> sum += t -> lazy * sz(t);
        if(t -> l) t -> l -> lazy += t -> lazy;
        if(t -> r) t -> r -> lazy += t -> lazy;
        t -> lazy = 0;
    }
    if(t->rev){
        swap(t->l,t->r);
        if(t -> l) t -> l -> rev ^= t -> rev;
        if(t -> r) t -> r -> rev ^= t -> rev;
        t -> rev = 0;
    }
}
void combine(pnode t) {
    //remember to reset datas of t first
    if(!t) return;
    push(t -> l);
    push(t -> r);
    t -> sum = t -> val; // Reset
    if(t -> l) t -> sum += t -> l -> sum;
    if(t -> r) t -> sum += t -> r -> sum;
}
void split(pnode t, pnode &l, pnode &r, int pos,
    int add = 0) {
    if(!t) return void(l = r = NULL); push(t);
    int curr = sz(t -> l) + add;
    if(curr <= pos) split(t -> r, t -> r, r, pos,
        curr + 1), l = t;
    else split(t -> l, l, t -> l, pos, add), r =
        t;
    upd_sz(t); combine(t);
}
void merge(pnode &t, pnode l, pnode r) {
    push(l), push(r);
    if(!l || !r) t = l ? l : r;
    else if(l -> prior > r -> prior) merge(l ->
        r, l -> r, r), t=l;
    else merge(r -> l, l, r -> l), t = r;
    upd_sz(t);
    combine(t);
}
ll query(pnode t, int l, int r) {
    pnode L, mid, R;
    split(t, L, mid, l - 1);
    split(mid, t, R, r - l);
    ll ans = t -> sum;
    merge(mid, L, t); merge(t, mid, R);
```

```
    return ans;
}
void update(pnode t, int l, int r, ll v) {
    pnode L, mid, R;
    split(t, L, mid, l - 1);
    split(mid, t, R, r - l);
    t -> lazy += v;
    merge(mid, L, t); merge(t, mid, R);
}
void insert(pnode &t, int pos, int v) {
    pnode L, R, tmp, y = new node(v);
    split(t, L, R, pos - 1);
    merge(tmp, L, y); merge(t, tmp, R);
}
void Del(pnode &t, int pos) {
    pnode L, R, mid;
    split(t, L, mid, pos - 1);
    split(mid, t, R, 0);
    pnode tmp = t;
    merge(t, L, R); free(tmp);
}
void cycshift(pnode &t, int l, int r) { ///left
    int val = query(t, l, l);
    Del(t, l);
    insert(t, r, val);
}
void Reverse(pnode t, int l, int r) {
    pnode L, mid, R;
    split(t, L, mid, l - 1);
    split(mid, t, R, r - l);
    t -> rev ^= 1;
    merge(mid, L, t); merge(t, mid, R);
}
/// 2 sat
struct TwoSAT{
int n,nn;
vector<int> G[2*N] , R[2*N], top;
int col[2*N],vis[2*N],ok[2*N]; /// col- sccNo, ok
    -> value assign
void init(int n){
    this->n=n, this->nn = n+n;
    for(int i= 0; i <= nn; i ++) G[i].clear(),
        R[i].clear(), col[i] = 0, vis[i] = 0,
        ok[i] = 0;
    top.clear();
}
int inv(int no) { return (no <= n) ? no + n: no -
    n; }
void add(int u,int v){
    G[u].push_back(v), R[v].push_back(u); }
void OR(int u,int v){
    add(inv(u), v), add(inv(v), u); }
void dfs(int u){
    vis[u] = 1;
    for(int v : G[u]) if(!vis[v]) dfs(v);
    top.push_back(u);
}
void dfs1(int u,int color){
    col[u] = color;
    if(u <= n) ok[u] = 1;
    else ok[u - n] = 0;
    for(int v : R[u]) if(!col[v])
        dfs1(v, color);
}
void FindScc(){
    for(int i=1; i<=nn; i++)
        if(!vis[i]) dfs(i);
    int color = 0;
    reverse(top.begin(), top.end());
    for(auto u: top) if(!col[u]) dfs1(u ,
        ++color);
}
bool solve(){
    FindScc();
    for(int i=1;i<=n;i++) if(col[i]==col[n + i])
        return false;
    /// ok[ node ] has state
    return true;
}};
//linear_time_mod_inverse in mod m
void init_inv() {
    inv[1] = 1;
    for(int i = 2; i < m; ++i)
    inv[i] = (m - (m/i) * inv[m%i] % m) % m;
}
//egcd : ax+by=gcd
ll egcd(ll a, ll b, ll& x, ll& y){
    if(!b) {y = 0, x = 1; return a;}
    ll g = egcd(b, a % b, y, x);
    y -= ((a / b) * x);
    return g;
}
// crt
pll GeneralCRT(pll a, pll b){
    if(a.second < b.second) swap(a, b);
    ll x, y; egcd(a.second, b.second, x, y);
    ll g = a.second * x + b.second * y;
    ll l = a.second / g * b.second;
    if((b.first - a.first) % g) return {-1, -1};
        // No Solution
    ll c = (b.first - a.first) % b.second;
    c = (c * x) % b.second;
    c = c / g * a.second; c += a.first;
    if(c < 0) c += l;
    return {c, l};
}
// Solve linear congruences equation:
// a[i] * x = b[i] MOD m[i] (mi don't need to be
    co-prime)
bool linearCongruences(const vector<ll> &a, const
    vector<ll> &b,
    const vector<ll> &m, ll &x, ll &M) {
    int n = a.size();
    x = 0; M = 1;
    for(int i=0;i<n;i++) {
        int a_ = a[i] * M, b_ = b[i] - a[i] * x,
            m_ = m[i];
        ll y, t, g = egcd(a_, m_, y, t);
        if (b_ % g) return false;
        b_ /= g; m_ /= g;
        x += M * (y * b_ % m_); M *= m_;
    }
    x = (x + M) % M;
    return true;
}
//Shank's; a^x=b in modulo mod; a,b and mod are
    co-prime
long long discrete_log(long long a,long long
    b,long long mod) {
    int n = (int)(sqrt(mod))+1;
    long long an = 1;
    for(int i=0;i<n;i++) an = (an*a)%mod;
```

```
    unordered_map<long long,int> values;
    long long curr=an;
    for(int p=1;p<=n;++p,curr=(1LL*curr*an)%mod)
        if(!values.count(curr)) values[curr]=p;
    curr = b;
    long long ans = discrete_log_infinity;
    for(int q=0;q<=n;q++,curr=(1LL*curr*a)%mod)
        if(values.count(curr)) ans =
            min(ans,1LL*values[curr]*n - q);
    return ans;
}
//a and b need not be coprime to mod
ll __discrete_log(ll a,ll b,ll mod) {
    ll g = __gcd(a,mod);
    if(g==1) return discrete_log(a,b,mod);
    if(b==1) return 0;
    if(b%g) return -1;
    b/=g; mod/=g; ll alpha=a/g;
    ll alpha_inverse =
        get<1>(extended_gcd(alpha,mod))%mod;
    if(alpha_inverse<0) alpha_inverse =
        (alpha_inverse+mod)%mod;
    b = (b*alpha_inverse)%mod;
    ll x = __discrete_log(a,b,mod);
    if(x==-1) return -1;
    x++; return x;
}
//deterministic miller-rabin
//Complexity : O(lgn)
bool deterministic_miller_rabin(u64 n) { //
    returns true if n is prime, else returns
    false.
    if (n < 2) return false;
    int r = 0; u64 d = n - 1;
    while ((d & 1) == 0)
        d >>= 1, r++;
    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23,
        29, 31, 37}) {
        if (n == a) return true;
        if (check_composite(n, a, d, r)) return
            false;
    }
    return true;
}
```

```
//pollard-rho-algorithm
//Complexity: n^(1/4)*(T(f(x)+lgn/m))
//Can't work on 0, 1, primes, power of primes
long long mult(long long a, long long b, long
    long mod) {
    return (__int128)a * b % mod;
}
long long f(long long x, long long c, long long
    mod) {
    return (mult(x, x, mod) + c) % mod;
}
long long pollard_rho(long long n, long long
    x0=2, long long c=1) {
    long long x = x0,g = 1, q = 1, xs, y;
    int m = 128; int l = 1;
    while (g == 1) {
        y = x;
        for (int i = 1; i < l; i++) x = f(x, c, n);
        int k = 0;
        while (k < l && g == 1) {
            xs = x;
            for (int i = 0; i < m && i < l - k;
                i++) {
                x = f(x, c, n); q = mult(q, abs(y -
                    x), n);
            }
            g = __gcd(q, n); k += m;
        }
        l *= 2;
    }
    if (g == n) {
        do {
            xs = f(xs, c, n); g = __gcd(abs(xs -
                y), n);
        } while (g == 1);
    }
    return g;
}
void factorize(u64 x,vector<u64> & v) {
    if(x==1) return;
    for(auto p:prime) {
        if((u64)p*p*p>x) break;
        while(x%p==0) v.push_back(p), x/=p;
    }
    if(x!=1) {
        if(deterministic_miller_rabin(x))
            v.push_back(x);
        else {
            u64 s = sqrtl(x);
            if(mult(s,s,x)==0) {
                v.push_back(s); v.push_back(s);
                return;
            }
            while(true) {
                u64 temp = pollard_rho(x,2 + rand()
                    % (x - 3),1);
                if(temp!=x) {
                    x/=temp;
                    if(x<temp) swap(x,temp);
                    v.push_back(temp),
                        v.push_back(x);
                    return;
                }
            }
        }
    }
}
//segmented_sieve
//set BLOCK_SIZE to sqrt(MAXN)
const int BLOCK_SIZE = 10;
vector<bool> is_prime(BLOCK_SIZE,true);
vector<int> prime;
void sieve() {
    is_prime[0] = is_prime[1] = false;
        is_prime[2] = true;
    prime.push_back(2);
    for(int i=4;i<BLOCK_SIZE;i+=2) is_prime[i] =
        false;
    for(int i=3;i<BLOCK_SIZE;i+=2) {
        if(is_prime[i]) {
            prime.push_back(i);
            for(long long
                j=1LL*i*i;j<BLOCK_SIZE;j+=2*i) {
                is_prime[j] = false;
            }
        }
    }
}
```

```
/*Segmented Sieve
    O(segment_size*lnln(segment_size)) */
void segmented_sieve(long long low, long long
    high,vector<long long> &lp_of_segment) {
    int sz = high-low+1; sieve();
    for(int i=0;i<sz;i++) lp_of_segment[i] =
        i+low;
    for(auto p: prime) {
        if(1LL*p*p>high) break;
        for(int i = (low+p-1)/p*p-low;i<sz;i+=p) {
            if(lp_of_segment[i]==i+low)
                lp_of_segment[i] = p;
        }
    }
}
/*Linear Sieve O(N); Time 10^7 = 0.279s, 10^8 =
    2.262s */
const int N = 10000001;
vector<int> prime;
int lp[N]; //holds the least prime that divides i
void linear_sieve() {
    lp[0] = 2; lp[1] = -1;
    for(int i=2;i<N;++i) {
        if(lp[i]==0) lp[i] = i, prime.push_back(i);
        for(int j=0;j<prime.size() &&
            prime[j]<=lp[i] &&
            1LL*prime[j]*i<N;++j)
            lp[i*prime[j]] = prime[j];
    }
}
///euler_phi_function O(sqrt(n))
int euler_totient_function(int n) {
    int phi = n; int temp = n;
    for(int i=2;1LL*i*i<=temp;i++) {
        bool flag = false;
        while(temp%i==0) temp/=i, flag = true;
        if(flag) phi/=i, phi*=(i-1);
    }
    if(temp!=1) phi/=temp, phi*=(temp-1);
    return phi;
}
/// mobius and stuff
void mobius(){
  memset(mu,-1,sizeof mu);
  mu[1]=1;
  for(int i=2;i<asz;i++) for(int
      j=2*i;j<asz;j+=i) mu[j]-=mu[i];
}
int number_of_coprime(vi &cnt, vi &divs){
  int a=0;
  for(auto x:divs)a+=cnt[x]*mu[x];
  return a;
}
vi dv[asz];
ll largest_coprime_product(vi &v,vi &cnt){
  ll ans=0; vi a;
  for(int i=(int)v.size()-1;i>=0;i--){
    int x=number_of_coprime(cnt,dv[v[i]]);
    while(x--){
      while(1){
        for(auto y:dv[a.back()])cnt[y]--;
        if(__gcd(a.back(),v[i])==1)break;
        a.pop_back();
      }
      ans=max(ans,a.back()*v[i]);
      a.pop_back();
    }
    a.push_back(v[i]);
    for(auto y:dv[v[i]])cnt[y]++;
  }
  while(!a.empty()){
    for(auto y:dv[a.back()])cnt[y]--;
    a.pop_back();
  }
  return ans;
}
///Computes floor sum [p/q]+[2p/q]+ ... +[np/q]
long long sum_of_floors(long long p,long long
    q,long long n){
  if(p == 0 || n == 0) return 0;
  if(n >= q) return p*(n/q)*(n+1) -
      (n/q)*((n/q)*p*q + p + q - 1)/2 +
      sum_of_floors(p,q,n%q);
  if(p >= q) return (p/q)*n*(n+1)/2 +
      sum_of_floors(p%q,q,n);
  return (n*p/q) * n - sum_of_floors(q,p,n*p/q);
}
/// polynomials
struct base { double x, y; base() { x = y = 0; }
    base(double x, double y): x(x), y(y) { } };
    inline base operator + (base a, base b) {
    return base(a.x + b.x, a.y + b.y); } inline
    base operator - (base a, base b) { return
    base(a.x - b.x, a.y - b.y); } inline base
    operator * (base a, base b) { return base(a.x
    * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
    inline base conj(base a) { return base(a.x,
    -a.y); } int lim = 1; vector<base> roots =
    {{0, 0}, {1, 0}}; vector<int> rev = {0, 1};
    const double PI = acosl(- 1.0); void
    ensure_base(int p) { if(p <= lim) return;
    rev.resize(1 << p); for(int i = 0; i < (1 <<
    p); i++) rev[i] = (rev[i >> 1] >> 1) + ((i &
    1) << (p - 1)); roots.resize(1 << p);
    while(lim < p) { double angle = 2 * PI / (1
    << (lim + 1)); for(int i = 1 << (lim - 1); i
    < (1 << lim); i++) { roots[i << 1] =
    roots[i]; double angle_i = angle * (2 * i + 1
    - (1 << lim)); roots[(i << 1) + 1] =
    base(cos(angle_i), sin(angle_i)); } lim++; }
    } void fft(vector<base> &a, int n = -1) {
    if(n == -1) n = a.size(); assert((n & (n -
    1)) == 0); int zeros = __builtin_ctz(n);
    ensure_base(zeros); int shift = lim - zeros;
    for(int i = 0; i < n; i++) if(i < (rev[i] >>
    shift)) swap(a[i], a[rev[i] >> shift]);
    for(int k = 1; k < n; k <<= 1) { for(int i =
    0; i < n; i += 2 * k) { for(int j = 0; j < k;
    j++) { base z = a[i + j + k] * roots[j + k];
    a[i + j + k] = a[i + j] - z; a[i + j] = a[i +
    j] + z; } } } } vector<int>
    multiply(vector<int> &a, vector<int> &b, int
    eq = 0) { int need = a.size() + b.size() - 1;
    int p = 0; while((1 << p) < need) p++;
    ensure_base(p); int sz = 1 << p; vector<base>
    A, B; if(sz > (int)A.size()) A.resize(sz);
    for(int i = 0; i < (int)a.size(); i++) { int
    x = (a[i] % mod + mod) % mod; A[i] = base(x &
    ((1 << 15) - 1), x >> 15); } fill(A.begin() +
    a.size(), A.begin() + sz, base{0, 0}); fft(A,
    sz); if(sz > (int)B.size()) B.resize(sz);
    if(eq) copy(A.begin(), A.begin() + sz,
```

```
B.begin()); else { for(int i = 0; i <
(int)b.size(); i++) { int x = (b[i] % mod +
mod) % mod; B[i] = base(x & ((1 << 15) - 1),
x >> 15); } fill(B.begin() + b.size(),
B.begin() + sz, base{0, 0}); fft(B, sz); }
double ratio = 0.25 / sz; base r2(0, - 1),
r3(ratio, 0), r4(0, - ratio), r5(0, 1);
for(int i = 0; i <= (sz >> 1); i++) { int j =
(sz - i) & (sz - 1); base a1 = (A[i] +
conj(A[j])), a2 = (A[i] - conj(A[j])) * r2;
base b1 = (B[i] + conj(B[j])) * r3, b2 =
(B[i] - conj(B[j])) * r4; if(i != j) { base
c1 = (A[j] + conj(A[i])), c2 = (A[j] -
conj(A[i])) * r2; base d1 = (B[j] +
conj(B[i])) * r3, d2 = (B[j] - conj(B[i])) *
r4; A[i] = c1 * d1 + c2 * d2 * r5; B[i] = c1
* d2 + c2 * d1; } A[j] = a1 * b1 + a2 * b2 *
r5; B[j] = a1 * b2 + a2 * b1; } fft(A, sz);
fft(B, sz); vector<int> res(need); for(int i
= 0; i < need; i++) { long long aa = A[i].x +
0.5; long long bb = B[i].x + 0.5; long long
cc = A[i].y + 0.5; res[i] = (aa + ((bb % mod)
<< 15) + ((cc % mod) << 30))%mod; } return
res; } template <int32_t MOD> struct modint {
int32_t value; modint() = default;
modint(int32_t value_) : value(value_) {}
inline modint<MOD> operator + (modint<MOD>
other) const { int32_t c = this->value +
other.value; return modint<MOD>(c >= MOD ? c
- MOD : c); } inline modint<MOD> operator -
(modint<MOD> other) const { int32_t c =
this->value - other.value; return
modint<MOD>(c < 0 ? c + MOD : c); } inline
modint<MOD> operator * (modint<MOD> other)
const { int32_t c = (int64_t)this->value *
other.value % MOD; return modint<MOD>(c < 0 ?
c + MOD : c); } inline modint<MOD> & operator
+= (modint<MOD> other) { this->value +=
other.value; if (this->value >= MOD)
this->value -= MOD; return *this; } inline
modint<MOD> & operator -= (modint<MOD> other)
{ this->value -= other.value; if (this->value
< 0) this->value += MOD; return *this; }
inline modint<MOD> & operator *= (modint<MOD>
other) { this->value = (int64_t)this->value *
other.value % MOD; if (this->value < 0)
this->value += MOD; return *this; } inline
modint<MOD> operator - () const { return
modint<MOD>(this->value ? MOD - this->value :
0); } modint<MOD> pow(uint64_t k) const {
modint<MOD> x = *this, y = 1; for (; k; k >>=
1) { if (k & 1) y *= x; x *= x; } return y; }
modint<MOD> inv() const { return pow(MOD -
2); } /* MOD must be a prime */ inline
modint<MOD> operator / (modint<MOD> other)
const { return *this * other.inv(); } inline
modint<MOD> operator /= (modint<MOD> other) {
return *this *= other.inv(); } inline bool
operator == (modint<MOD> other) const {
return value == other.value; } inline bool
operator != (modint<MOD> other) const {
return value != other.value; } inline bool
operator < (modint<MOD> other) const { return
value < other.value; } inline bool operator >
(modint<MOD> other) const { return value >
other.value; } }; template <int32_t MOD>
modint<MOD> operator * (int64_t value,
modint<MOD> n) { return modint<MOD>(value) *
n; } template <int32_t MOD> modint<MOD>
operator * (int32_t value, modint<MOD> n) {
return modint<MOD>(value % MOD) * n; }
template <int32_t MOD> ostream & operator <<
(ostream & out, modint<MOD> n) { return out
<< n.value; } using mint = modint<mod>;
struct poly { vector<mint> a; inline void
normalize() { while(a.size() && a.back() ==
0) a.pop_back(); } template<class...Args>
poly(Args...args): a(args...) { } poly(const
initializer_list<mint> &x): a(x.begin(),
x.end()) {} int size() const { return
(int)a.size(); } inline mint coef(const int
i)const { return (i < a.size()) ? a[i]:
mint(0); } mint operator[](const int i) {
return coef(i); } poly operator + (const poly
&x) const { int n = max(size(), x.size());
vector<mint> ans(n); for(int i = 0; i < n;
i++) ans[i] = coef(i) + x.coef(i); return
ans; } poly operator - (const poly &x) const
{ int n = max(size(), x.size()); vector<mint>
ans(n); for(int i = 0; i < n; i++) ans[i] =
coef(i) - x.coef(i); return ans; } poly
operator * (const poly& b) const {
vector<int> A, B; for(auto x: a)
A.push_back(x.value); for(auto x: b.a)
B.push_back(x.value); auto res = multiply(A,
B); vector<mint> ans; for(auto x: res)
ans.push_back(mint(x)); return ans; } poly
operator * (const mint& x) const { int n =
size(); vector<mint> ans(n); for(int i = 0; i
< n; i++) ans[i] = a[i] * x; return ans; }
poly operator / (const mint &x) const{ return
(*this) * x.inv(); } poly& operator += (const
poly &x) { return *this = (*this) + x; }
poly& operator -= (const poly &x) { return
*this = (*this) - x; } poly& operator *=
(const poly &x) { return *this = (*this) * x;
} poly& operator *= (const mint &x) { return
*this = (*this) * x; } poly& operator /=
(const mint &x) { return *this = (*this) / x;
} poly mod_xk(int k) const { return
{a.begin(), a.begin() + min(k, size())}; }
poly div_xk(int k) const { /* divide by x^k
*/ return vector<mint>(a.begin() + min(k,
(int)a.size()), a.end()); } poly
reverse_it(int n, bool rev = 0) const { /*
reverses and leaves only n terms */ poly
ans(*this); if(rev) { /* if rev = 1 then tail
goes to head */ ans.a.resize(max(n,
(int)ans.a.size())); } reverse(ans.a.begin(),
ans.a.end()); return ans.mod_xk(n); } poly
inverse(int n) const { /* 1 / p(x) % x^n */
assert(!a.empty()); assert(a[0] != 0); poly
ans{mint(1) / a[0]}; for(int i = 1; i < n; i
*= 2) { ans = (ans * mint(2) - ans * ans *
mod_xk(2 * i)).mod_xk(2 * i); } return
ans.mod_xk(n); } pair<poly, poly>
divmod_slow(const poly &b) const { /* when
divisor or quotient is small */ vector<mint>
A(a); vector<mint> ans; while(A.size() >=
b.a.size()) { ans.push_back(A.back() /
b.a.back()); if(ans.back() != mint(0)) {
for(size_t i = 0; i < b.a.size(); i++) {
```

```
    A[A.size() - i - 1] -= ans.back() *
    b.a[b.a.size() - i - 1]; } } A.pop_back(); }
    reverse(ans.begin(), ans.end()); return {ans,
    A}; } pair<poly, poly> divmod(const poly &b)
    const { /* returns quotient and remainder of
    a mod b */ if(size() < b.size()) return
    {poly{0}, *this}; int d = size() - b.size();
    if(min(d, b.size()) < 250) return
    divmod_slow(b); poly D = (reverse_it(d + 1) *
    b.reverse_it(d + 1).inverse(d + 1)).mod_xk(d
    + 1).reverse_it(d + 1, 1); return {D, *this -
    (D * b)}; } poly operator / (const poly &t)
    const {return divmod(t).first;} poly operator
    % (const poly &t) const {return
    divmod(t).second;} poly& operator /= (const
    poly &t) {return *this = divmod(t).first;}
    poly& operator %= (const poly &t) {return
    *this = divmod(t).second;} poly pow(int k,
    int n) const { /* p(x)^k mod x^n*/
    if(a.empty()) return *this; poly ans({1}), b
    = mod_xk(n); while(k) { if(k & 1) ans = (ans
    * b).mod_xk(n); b = (b * b).mod_xk(n); k >>=
    1; } return ans; } poly root(int n, int k =
    2) const { /*kth root of p(x) mod x^n*/
    if(a.empty()) return *this; assert(a[0] ==
    1); poly q({1}); for(int i = 1; i < n; i <<=
    1){ if(k == 2) q += mod_xk(2 * i) *
    q.inverse(2 * i); else q = q * mint(k - 1) +
    mod_xk(2 * i) * q.inverse(2 * i).pow(k - 1, 2
    * i); q = q.mod_xk(2 * i) / mint(k); } return
    q.mod_xk(n); } };
/// fft
using cd=complex<double>;
void fft(vector<cd> &a, bool invert) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if (i < j) swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert ? -1 :
            1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }
    if (invert) for (cd & x : a) x /= n;
}
vector<ll> multiply(vector<ll> &a,vector<ll> &b){
    vector<cd>
        fa(a.begin(),a.end()),fb(b.begin(),b.end());
    ll n=1;
    while(n<a.size()+b.size())n<<=1;
    fa.resize(n),fb.resize(n);
    fft(fa,false),fft(fb,false);
    for(int i=0;i<n;i++)fa[i]*=fb[i];
    fft(fa,true);
    vector<ll> res(n);
    for(int i=0;i<n;i++)
        res[i]=round(fa[i].real());
    return res;
}
//FWHT
#define bitwiseXOR 1
///#define bitwiseAND 2
///#define bitwiseOR 3
void FWHT(vector< LL >&p, bool inverse){
  int n = p.size();
  assert((n&(n-1))==0);
  for (int len = 1; 2*len <= n; len <<= 1) {
    for (int i = 0; i < n; i += len+len) {
      for (int j = 0; j < len; j++) {
        LL u = p[i+j], v = p[i+len+j];
        #ifdef bitwiseXOR
        p[i+j] = u+v, p[i+len+j] = u-v;
        #endif // bitwiseXOR
        #ifdef bitwiseAND
        if (!inverse) p[i+j] = v, p[i+len+j] = u+v;
        else p[i+j] = -u+v, p[i+len+j] = u;
        #endif // bitwiseAND
        #ifdef bitwiseOR
        if (!inverse) p[i+j] = u+v, p[i+len+j] = u;
        else p[i+j] = v, p[i+len+j] = u-v;
        #endif // bitwiseOR
      }
    }
  }
  #ifdef bitwiseXOR
  if (inverse) for (int i = 0; i < n; i++)
    assert(p[i]%n==0), p[i] /= n;
  #endif // bitwiseXOR
}
/// polynomial interpolation
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd res(n,0), temp(n,0);
  for(int k=0;k<n;k++) for(int i=k+1;i<n;i++)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  double last = 0; temp[0] = 1;
  for(int k=0;k<n;k++) for(int i=0;i<n;i++) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
}
/// binomial heap
#define pTree BinomialTree<T>*
template < class T >class BinomialTree{
public :
  T data ;
  int degree ;
  pTree parent, sibling, child;
  /// child points to largest degree child ,
      sibling points to smaller degree tree under
      same parent
  BinomialTree(T data = T(0)): data(data){
    degree = 0;
    parent = sibling = child = 0;
  }
  ~BinomialTree(){
```

```
    if(child) delete child;
    if(sibling) delete sibling;
  }
};
template<class T >pTree Union(pTree t1, pTree t2){
  assert(t1->degree == t2->degree);
  // make t1->data < t2->data
  if( t2->data < t1->data ) swap(t1, t2);
  t2->parent = t1 ;
  t2->sibling = t1->child;
  t1->child = t2; t1->degree ++;
  return t1;
}
template < class T > class BinomialHeap{
private :
  BinomialTree < T> * head;
  void AddTree(pTree &last_head, pTree new_head){
    new_head->parent = 0;
    if(last_head) last_head->sibling = new_head;
    else head=new_head ;
    last_head = new_head;
    new_head->sibling = 0;
  }
public:
  BinomialHeap():head(0) {}
  BinomialHeap(T data ){
    head = new BinomialTree<T>(data); }
  bool Empty(){
    return head == 0;}
  void Union(BinomialHeap<T> &h){
    pTree carry_head = 0;
    pTree head1 = head; /// head from this heap
    pTree head2 = h.head; /// head from h heap
    pTree last_head = 0;
    while(head1 or head2){
      int d1 = head1 ? head1->degree : 1000000 ;
      int d2 = head2 ? head2->degree : 1000001 ;
      int dmin = min(d1,d2);
      if(d1 > d2)
          swap(d1, d2 ), swap(head1, head2);
      assert(head1);
      /// now head1 contains minimum degree
      assert(dmin >= (carry_head ?
          carry_head->degree : -1 ));
      pTree nxt_head1 = head1;
      pTree nxt_head2 = head2;
      if(d1 == d2 ){
        assert(head1);
        assert(head2);
        nxt_head1 = head1 -> sibling;
        nxt_head2 = head2 -> sibling;
        pTree merged_head = ::Union(head1, head2);
        if(carry_head)
            AddTree(last_head,carry_head );
        carry_head = merged_head;
      }
      else{
        nxt_head1 = head1->sibling;
        if(carry_head){
          if(carry_head->degree < d1){
            AddTree(last_head, carry_head);
            AddTree(last_head , head1);
            carry_head = 0;
          }
          else{
            assert(carry_head->degree == d1);
            carry_head = ::Union(head1,
                carry_head);
          }
        }
        else AddTree(last_head, head1);
      }
      head1 = nxt_head1, head2 = nxt_head2;
    }
    if(carry_head) AddTree(last_head ,
        carry_head);
    h.head=0;
  }
  void Insert(T data){
    BinomialHeap h(data);
    Union(h);
  }
  pTree FindMin(){
    assert(head);
    pTree ret_head = head;
    pTree now = head->sibling;
    while(now){
      if(now->data < ret_head->data)
          ret_head = now;
        now = now->sibling;
      }
      return ret_head;
    }
    pTree ExtractMin(){
      pTree ret_head = FindMin();
      pTree last_head = 0 ; /// last head in root
          list where ret_head is sibling of
          last_head
      if(head != ret_head){
        last_head = head;
        while(last_head->sibling != ret_head)
            last_head = last_head->sibling;
      }
      /// detach ret_head from this heap
      if(last_head ) last_head->sibling =
          ret_head->sibling ;
      else head = ret_head->sibling;
      ret_head->parent = 0;
      ret_head->sibling = 0;
      /// now make a heap with child of ret_head
      if(ret_head->child){
        stack<pTree > heads;
        pTree now = ret_head->child;
        while(now){
          heads.push(now);
          now = now->sibling;
        }
        BinomialHeap h;
        last_head = 0;
        while(!heads.empty()){
          h.AddTree(last_head, heads.top());
          heads.pop();
        }
        Union(h);
      }
      return ret_head;
    }
    ~BinomialHeap(){
      if(head) delete head;
      head = 0;
    }
};
```

```
/// 2d bit with lazy
long long bit[4][mx][my];
void update( int x, int y, int val, int i ) {
    int y1;
    while( x<=mx ) {
        y1=y;
        while( y1<=my ) {
            bit[i][x][y1] += val;
            y1 += (y1&-y1);
        }
        x += (x&-x);
    }
}
long long query( int x, int y, int i) {
    long long ans=0; int y1;
    while( x>0 ) {
        y1 = y;
        while( y1>0 ) {
            ans += bit[i][x][y1];
            y1 -= (y1&-y1);
        }
        x -= (x&-x);
    }
    return ans;
}
// add value k from (x1,y1) to (x2,y2) inclusive
void add( int x1, int y1, int x2, int y2, int k ){
    update(x1,y1,k,0);
    update(x1,y2+1,-k,0);
    update(x2+1,y1,-k,0);
    update(x2+1,y2+1,k,0);
    update(x1,y1,k*(1-y1),1);
    update(x1,y2+1,k*y2,1);
    update(x2+1,y1,k*(y1-1),1);
    update(x2+1,y2+1,-y2*k,1);
    update(x1,y1,k*(1-x1),2);
    update(x1,y2+1,k*(x1-1),2);
    update(x2+1,y1,k*x2,2);
    update(x2+1,y2+1,-x2*k,2);
    update(x1,y1,(x1-1)*(y1-1)*k,3);
    update(x1,y2+1,-y2*(x1-1)*k,3);
    update(x2+1,y1,-x2*(y1-1)*k,3);
    update(x2+1,y2+1,x2*y2*k,3);
}
```

```
// get value from (x1,y1) to (x2,y2) inclusive
long long get( int x1, int y1, int x2, int y2 ){
  intl v1=query(x2,y2,0)*x2*y2+
      query(x2,y2,1)*x2+ query(x2,y2,2)*y2+
      query(x2,y2,3);
  intl v2=query(x2,y1-1,0)*x2*(y1-1)+
      query(x2,y1-1,1)*x2+
      query(x2,y1-1,2)*(y1-1)+ query(x2,y1-1,3);
  intl v3=query(x1-1,y2,0)*(x1-1)*y2+
      query(x1-1,y2,1)*(x1-1)+
      query(x1-1,y2,2)*y2+ query(x1-1,y2,3);
  intl v4=query(x1-1,y1-1,0)*(x1-1)*(y1-1)+
      query(x1-1,y1-1,1)*(x1-1)+
      query(x1-1,y1-1,2)*(y1-1)+
      query(x1-1,y1-1,3);
  intl ans=v1-v2-v3+v4;
  return ans;
}
/// gaussian
//returns -1 if impossible,1 if solution is
    unique, 0 otherwise
int gauss(vector<vector<double>>
    &mat,vector<double> &ans){
  int n=mat.size(),m=mat[0].size()-1;
  vector<int> w(m,-1); int ret=1;
  for(int i=0,j=0;j<n&&i<m;i++){
    int mx=j;
    for(int k=j+1;k<n;k++)
        if(mat[mx][i]<mat[k][i])mx=k;
    if(abs(mat[mx][i])<EPS){
      ret=0; continue;
    }
    for(int k=i;k<=m;k++)
        swap(mat[j][k],mat[mx][k]);
    for(int k=0;k<n;k++){
      if(k==j)continue;
      double d=mat[k][i]/mat[j][i];
      for(int l=i;l<=m;l++) mat[k][l]-=d*mat[j][l];
    }
    w[i]=j++;
  }
  ans.assign(m,0);
  for(int i=0;i<m;i++)if(w[i]!=-1) ans[i]=
      mat[w[i]][m]/mat[w[i]][i];
  for(int i=0;i<n;i++){
    double d=0;
    for(int j=0;j<m;j++)d+=ans[j]*mat[i][j];
    if(abs(d-mat[i][m])>EPS)return -1;
  }
  return ret;
}
/// matrix stuff
const ll MOD = 1e9 + 7;
const ll MOD2 = MOD * MOD * 3;
inline ll bigMod(ll a,ll b){
ll res=1;
while(b){
if(b&1) res=(res*a)%MOD;
a=(a*a)%MOD; b>>=1;
}
return res;
}
inline ll inv(ll n) {return bigMod(n,MOD-2);}
inline ll Mul(ll a,ll b) {return (a*b)%MOD;}
inline ll Div(ll a,ll b) {return Mul(a,inv(b));}
const int MAX = 100;
struct Matrix{
int row, col;
ll m[MAX][MAX];
Matrix() {memset(m,0,sizeof(m));}
void Set(int r,int c) {row = r; col = c;}
Matrix(int r,int c) {memset(m,0,sizeof(m));
    Set(r,c);}
void normalize(){
for(int i=1; i<=row; i++){
for(int j=1; j<=col; j++){
m[i][j] %= MOD;
if(m[i][j] < 0) m[i][j] += MOD;
}}};
Matrix Multiply(Matrix A,Matrix B){
Matrix ans(A.row,B.col);
for(int i=1;i<=A.row;i++){
for(int j=1;j<=B.col;j++){
ans.m[i][j]=0;
ll sm = 0;
for(int k=1;k<=A.col;k++){
sm+=(A.m[i][k]*B.m[k][j]);
if(sm >= MOD2) sm -= MOD2;
```

```
}
ans.m[i][j] = sm % MOD;
}}
return ans;
}
Matrix Power(Matrix mat,ll p){
Matrix res(mat.row , mat.col);
Matrix ans(mat.row , mat.col);
int n = ans.row;
for(int i=1;i<=n;i++){
for(int j=1;j<=n;j++){
    ans.m[i][j]=0;
    res.m[i][j]=mat.m[i][j];
}
ans.m[i][i]=1;
}
while(p){
if(p&1) ans=Multiply(ans,res);
res=Multiply(res,res); p=p/2;
}
return ans;
}
ll Det(Matrix mat){
assert(mat.row == mat.col);
int n = mat.row;
mat.normalize();
ll ret = 1;
for(int i = 1; i <= n; i++){
for(int j = i + 1; j <= n; j++){
while(mat.m[j][i]){
ll t = Div(mat.m[i][i], mat.m[j][i]);
for(int k = i; k <= n; ++k){
mat.m[i][k] -= Mul(mat.m[j][k] , t);
if(mat.m[i][k] < 0) mat.m[i][k] += MOD;
swap(mat.m[j][k], mat.m[i][k]);
}
ret = MOD - ret;
}}
if(mat.m[i][i] == 0) return 0;
ret = Mul(ret, mat.m[i][i]);
}
if(ret < 0) ret += MOD;
return ret;
}
ll Tmp[MAX<<1][MAX<<1];
Matrix Inverse(Matrix mat){
assert(mat.row == mat.col);
assert(Det(mat) != 0);
int n = mat.row;
mat.normalize();
for(int i=1;i<=n;i++){
for(int j=1;j<=n;j++) Tmp[i][j] = mat.m[i][j];
for(int j=n+1; j<=2*n; j++) Tmp[i][j] = 0;
Tmp[i][i+n] = 1;
}
for(int i=1; i<=n; i++){
assert(Tmp[i][i] != 0);
for(int j=1; j<=n; j++){
if(i == j) continue;
ll c = Div(Tmp[j][i], Tmp[i][i]);
for(int k=i; k<=2*n; k++){
    Tmp[j][k] = Tmp[j][k] - Mul(Tmp[i][k], c);
    if(Tmp[j][k] < 0) Tmp[j][k] += MOD;
}}}
Matrix Inv(n,n);
for(int i=1; i<=n; i++)
for(int j = 1; j <= n; j++)
Inv.m[i][j] = Div(Tmp[i][j+n], Tmp[i][i]);
return Inv;
}
```

```
/// mo with update
struct query{
  /* t = number of updates before this query*/
  int l, r, t, id;
  bool operator < (const query &x) const {
    if(l / B == x.l / B){
      if(r / B == x.r / B) return (t < x.t) ;
      return (r / B < x.r / B) ;
    }
    return l / B < x.l / B;
  }
};
struct upd{
  int pos, old, cur;
};
void update(int pos, int x) {
  if (curL<=pos && pos<=curL) {
    add(x);
    del(a[pos]);
  }
  a[pos] = x;
}
t = nu, curL = 1, curR = 0;
for (int i = 1; i <= nq; i++) {
  int L = Q[i].l, R = Q[i].r, T = Q[i].t;
  while(t < T) t++, update(U[t].pos, U[t].cur);
  while(t > T) update(U[t].pos, U[t].old), t--;
  while(curL > L) add(a[--curL]);
  while(curR < R) add(a[++curR]);
  while(curL < L) del(a[curL++]);
  while(curR > R) del(a[curR--]);
  ans[Q[i].id] = something;
}
```