Team notebook

Pontificia Universidad Catolica del Peru - Treevial

November 8, 2019

Contents

1 Data Structures

1.1 2D Segment Tree

```
// Iterative, 2d, commutative segment tree.
// Read into t[n,2*n)*[m,2*m), call build().
typedef int T;
const int MAX = 1e3;
const T OpId = 0; // Identity element
// Associative, conmutative operation
T Op(T val1, T val2){
   return val1 + val2;
}
T t[2*MAX][2*MAX];
int n, m;

void build(){
  for( int i = n-1 ; i > 0 ; i-- )
             for ( int j = 2*m ; j >= m; j-- )
                    t[i][j] = Op(t[i<<1][j],
                        t[i<<1|1][j]);
       for ( int i = 2*n ; i > 0 ; i-- )
             for ( int j = m-1 ; j > 0 ; j-- )
                    t[i][j] = Op(t[i][j<<1],
                        t[i][j<<1|1])
}

void mody( int x, int y ){
       int q = y;
       while (q >>= 1)
             t[x][q] = Op(t[x][q<<1],
                 t[x][q<<1|1]);
}

void modify( int x , int y , T val ){
       t[x+=n][y+=m] = val;
       mody(x,y);
       while (x >>= 1) {
             t[x][y] = Op(t[x<<1][y],
                 t[x<<1|1][y]);
             mody(x, y);
       }
}
```

```
T gety(int x, int yl, int yr){
       T ans = OpId;
       for ( yl += m, yr += m ; yl < yr ; yl >>=
           1, yr >>= 1){
              if (yl&1) ans = Op(ans, t[x][yl++]);
              if (yr&1) ans = Op(ans, t[x][--yr]);
       }
       return ans;
}

T get(int xl, int xr, int yl, int yr){ //
    [xl,xr)*[yl,yr)
       T ans = OpId;
       for ( xl += n, xr += n ; xl < xr ; xl >>=
           1, xr >>= 1){
              if (xl&1) ans = Op(ans, gety(xl++,
                  yl, yr));
              if (xr&1) ans = Op(ans, gety(--xr,
                  yl, yr));
       }
       return ans;
}
```

1.2 Block Cut Tree

```
/* TAP Police Query */

#include <sys/resource.h>
#define N 1010000
#define LGMAX 25

/* Start Block Cut Tree */

int P[N][LGMAX];
vector<ii> st;
int n,m,gid,art[N],num[N],low[N],par[N],idBct[N];
vector<int> adj[N];
vector<int> bct[N]; // bct = block cut tree
vector< pair<ii,int> > bridges;
vector< vector<ii> > bcc;
```

```
inline ii edge(int &a, int &b){ return
    ii(min(a,b),max(a,b)); }

void Tarjan(int u, bool root = false){
   num[u] = low[u] = gid++;
   int child = 0;
   for(int v : adj[u]) if(v != par[u]){
      if(num[v] == -1){
         child++;
         par[v] = u;
         st.push_back(make_pair(u,v));
         Tarjan(v);
         low[u] = min(low[u],low[v]);
         if(low[v] > num[u])
             bridges.push_back(make_pair(edge(u,v),0));
         if((root and child > 1) or (!root and
             low[v] >= num[u])){
            // punto de articulacion
            art[u] = 1;
            vector<ii> tmp;
            while(st.back() != make_pair(u,v))
               tmp.push_back(st.back()),
                   st.pop_back();
            tmp.push_back(st.back()),
                st.pop_back();
            bcc.push_back(tmp);
         }
      }else if(num[v] < num[u]){
         low[u] = min(low[u],num[v]);
         st.push_back(make_pair(u,v));
      }
   }
}

void BuildBlockCut(){
   int cnt = 0;
   for(vector<ii> &vec : bcc){
      int aux = cnt++;
      if(vec.size() == 1){

         (*lower_bound(bridges.begin(),bridges.end(),
         make_pair(edge(vec[0].first,vec[0].second),-1))).second
             = aux;
      }
      set<int> s;
      for(ii &p : vec){
         int a = p.first, b = p.second;
         if(art[a]){
            if(idBct[a] == -1){
               s.insert(a);
               idBct[a] = cnt++;
               bct[aux].push_back(idBct[a]);
               bct[idBct[a]].push_back(aux);
            }else if(!s.count(a)){
               s.insert(a);
               bct[aux].push_back(idBct[a]);
               bct[idBct[a]].push_back(aux);
            }
         }else idBct[a] = aux;
         if(art[b]){
            if(idBct[b] == -1){
               s.insert(b);
               idBct[b] = cnt++;
               bct[aux].push_back(idBct[b]);
               bct[idBct[b]].push_back(aux);
            }else if(!s.count(b)){
               s.insert(b);
               bct[aux].push_back(idBct[b]);
               bct[idBct[b]].push_back(aux);
            }
         }else idBct[b] = aux;
      }
   }
}

void Go(){
   // Tarjan
   gid = 0;
   for(int i = 0; i < n; ++i)
      art[i] = 0, par[i] = num[i] = low[i] = -1;

   for(int i = 0; i < n; ++i) if(num[i] == -1){
      Tarjan(i,true);
      if(st.size() > 0) bcc.push_back(st);
      st.clear();
   }

   sort(bridges.begin(),bridges.end());
   // construir block-cut tree
   memset(idBct,-1,sizeof(idBct));
   BuildBlockCut();
}

/* End Block Cut Tree */

int tin[N],tout[N],dep[N];

bool ancestor(int a, int b){
   return tin[b] >= tin[a] and tout[b] <= tout[a];
}

int __lca(int a, int b){
   if(dep[a] > dep[b]) swap(a,b);
   for(int j = LGMAX-1; j >= 0; --j)
      if(dep[b] - (1<<j) >= dep[a])
         b = P[b][j];
   if(a == b) return a;
   for(int j = LGMAX-1; j >= 0; --j)
      if(P[a][j] != P[b][j])
         a = P[a][j], b = P[b][j];
   return P[a][0];
}

void dfs(int u, int p = -1, int d = 0){
   tin[u] = gid++;
   P[u][0] = p;
   dep[u] = d;
   for(int j = 1; j < LGMAX; ++j)
      if(P[u][j-1] != -1)
         P[u][j] = P[P[u][j-1]][j-1];
   for(int v : bct[u]) if(v != p)
      dfs(v,u,d+1);
   tout[u] = gid++;
}

bool isBridge(int a, int b){
   vector< pair<ii,int> >::iterator it =
       lower_bound(bridges.begin(),
   bridges.end(),make_pair(edge(a,b),-1));
   if(it == bridges.end()) return false;
   return (*it).first == edge(a,b);
}
```

```
bool isLeaf(int a, int b){
   return adj[a].size() == 1 or adj[b].size() ==
       1;
}

bool onPath(int a, int c, int b){
   return ancestor(a,c) and ancestor(c,b);
}

int getBridgeId(int a, int b){
   return (*lower_bound(bridges.begin(),
   bridges.end(),make_pair(edge(a,b),-1))).second;
}

int main(){
   int q,t,a,b,c,d;
   scanf("%d %d",&n,&m);
   for(int i = 0; i < m; ++i){
      scanf("%d %d",&a,&b);
      // indexar en cero
      a--, b--;
      adj[a].push_back(b);
      adj[b].push_back(a);
   }

   Go();

   gid = 0;
   dfs(0);
   /* Responder queries */
   scanf("%d",&q);
   while(q--){
      scanf("%d",&t);
      if(t == 1){
         scanf("%d %d %d %d",&a,&b,&c,&d);
         a--; b--; c--; d--;
         if(isBridge(c,d)){
            if(isLeaf(c,d)){
               if( (adj[c].size() == 1) and (a == c
                   or b == c) )
                  printf("ne\n");
               else if( (adj[d].size() == 1) and (a
                   == d or b == d) )
                  printf("ne\n");
               else printf("da\n");
            }else{
               int u = idBct[a], v = idBct[b];
               int w = getBridgeId(c,d), lca =
                   __lca(u,v);
               if(onPath(lca,w,u) or
                   onPath(lca,w,v)) printf("ne\n");
               else printf("da\n");
            }
         }else printf("da\n");
      }else{
         scanf("%d %d %d",&a,&b,&c);
         a--; b--; c--;
         if(art[c]){
            int u = idBct[a], v = idBct[b], w =
                idBct[c], lca = __lca(u,v);
            if(onPath(lca,w,u) or onPath(lca,w,v))
                printf("ne\n");
            else printf("da\n");
         }else printf("da\n");
      }
   }

   return 0;
}
```

1.3 Dynamic Hash

```
/* Dynamic Hash
*/

ll len1[N], len2[N];

struct Hash{
   vi dp1, dp2;
   string s;
   ll ans1, ans2, n;
   inline void build(ll id){
      if(!id) ans1=0, ans2=0;
      n=sz(s);
      FER(i, id, n) dp1.pb(0), dp2.pb(0);
      FER(i,id,sz(s)){
         if(s[i]>='a' && s[i]<='z'){
            ans1=(ans1*bas)%mod1;
            ans1+=(s[i]-'a'+1);
            ans2=(ans2*bas)%mod2;
            ans2+=(s[i]-'a'+1);
            if(ans1>=mod1) ans1-=mod1;
            if(ans2>=mod2) ans2-=mod2;
            dp1[i]=ans1, dp2[i]=ans2;
         }
         else if(s[i]>='A' && s[i]<='Z'){
            ans1=(ans1*bas)%mod1;
            ans1+=(s[i]-'A'+27);
            ans2=(ans2*bas)%mod2;
            ans2+=(s[i]-'A'+27);
            if(ans1>=mod1) ans1-=mod1;
            if(ans2>=mod2) ans2-=mod2;
            dp1[i]=ans1, dp2[i]=ans2;
         }
         else{
            ans1=(ans1*bas)%mod1;
            ans1+=(s[i]-'0'+55);
            ans2=(ans2*bas)%mod2;
            ans2+=(s[i]-'0'+5);
            if(ans1>=mod1) ans1-=mod1;
            if(ans2>=mod2) ans2-=mod2;
            dp1[i]=ans1, dp2[i]=ans2;
         }
      }
   }
   inline ii Query(ll l, ll r){
      ll ans1=dp1[r-1], ans2=dp2[r-1];
      if(l){
         ll add1=(dp1[l-1]*len1[r-l])%mod1;
         ll add2=(dp2[l-1]*len2[r-l])%mod2;
         ans1-=add1, ans2-=add2;
         if(ans1<0) ans1+=mod1;
         if(ans2<0) ans2+=mod2;
         return {ans1, ans2};
      }
      return {ans1, ans2};
   }
};
```

```
int main(){
   fastio;
   ll t1=1, t2=1;
   FER(i,0,N) {
      len1[i]=t1;
      len2[i]=t2;
      t1=(t1*bas)%mod1;
      t2=(t2*bas)%mod2;
   }
   return 0;
}
```

1.4 Dynamic Segment Tree Beats

```
/* Dynamic Segment Tree Beats */

struct T{
   ll fmx, fmn, smx, smn;
   T(){}
   T(ll fmx, ll fmn, ll smx, ll smn):
      fmx(fmx), fmn(fmn), smx(smx), smn(smn){}

   inline T Op(T a, T b){
      if(a.fmx < b.fmx){
         a.smx=a.fmx;
         a.fmx=b.fmx;
      }
      if(a.fmx == b.fmx){
         a.smx=max(a.smx, b.smx);
      }
      else a.smx=max(a.smx, b.fmx);
      if(a.fmn > b.fmn){
         a.smn=a.fmn;
         a.fmn=b.fmn;
      }
      if(a.fmn==b.fmn){
         a.smn=min(a.smn, b.smn);
      }
      else a.smn=min(a.smn, b.fmn);
      return a;
   }
   inline void Umax(ll x){
      if(fmx==fmn) fmx=x;
      if(smx==fmn) smx=x;
      fmn=x;
   }
   inline void Umin(ll x){
      if(fmn==fmx) fmn=x;
      if(smn==fmx) smn=x;
      fmx=x;
   }
};

struct Dinamic_Segment_Tree_Beats{
   ll n, nodes, root;
   T lazy[N];
   ll L[N], R[N];

   inline void updpro(T val, ll id, ll l, ll r){
      if(val.fmx<lazy[id].fmx){
         lazy[id].Umin(val.fmx);
      }
      if(val.fmn>lazy[id].fmn){
         lazy[id].Umax(val.fmn);
      }
   }

   inline void newLeaf(T val, ll id, ll l, ll r){
      lazy[id]=val;
   }

   inline void proh(ll id, ll l, ll r){
      ll mid=(l+r)>>1;
      if(l+1==r) return ;
      if(!L[id]){
         L[id]=nodes++;
         newLeaf(lazy[id], L[id], l, mid);
      }
      else updpro(lazy[id], L[id], l, mid);
      if(!R[id]){
         R[id]=nodes++;
         newLeaf(lazy[id], R[id], mid, r);
      }
      else updpro(lazy[id], R[id], mid, r);
   }

   inline void UpdMax(ll x, ll y, ll val, ll &id,
       ll l, ll r){
      if(!id){
         id=nodes;
         L[id]=R[id]=0;
         nodes++;
      }
      proh(id, l, r);
      if(x>=r || y<=l || lazy[id].fmn>=val) return;
      if(x<=l && r<=y && lazy[id].smn>val &&
          lazy[id].fmn<val) {
         lazy[id].Umax(val);
         return;
      }
      ll mid=(l+r)>>1;
      if(x<mid) UpdMax(x, y, val, L[id], l, mid);
      if(y>mid) UpdMax(x, y, val, R[id], mid, r);
      lazy[id]=lazy[id].Op(lazy[L[id]],
          lazy[R[id]]);
   }

   inline ll Sol(ll x, ll y, ll id, ll l, ll r){
      proh(id, l, r);
      if(x>=r || y<=l) return -INF;
      if(x<=l && r<=y) return lazy[id].fmx;
      ll mid=(l+r)>>1, left, right;
      left=Sol(x, y, L[id], l, mid);
      right=Sol(x, y, R[id], mid, r);
      return max(left, right);

   }

   inline void build() {
      nodes=1, root=0;
      FER(i, 1, 2){
         lazy[i].fmx=lazy[i].smx=-INF;
         lazy[i].fmn=lazy[i].smn=INF;
         lazy[i].fmx=lazy[i].fmn=-1;
      }
   }

   inline void updMax(ll x, ll y, ll val) {
       UpdMax(x, y, val, root, 1, n);}
```

```
   inline ll que(ll x, ll y) { return Sol(x, y,
       root, 1, n);}
}st;

int main(){
   fastio;
   ll n, q; cin>>n>>q;
   st.n=n+1;
   st.build();
   st.updMax(1, st.n, 0);
   ll l, r, x, t;
   FER(i, 0, q){
      cin>>t;
      if(t==1){
         cin>>l>>r>>x;
         st.updMax(l, r+1, x);
      }
      else {
         cin>>l>>r;
         x=st.que(l, r+1);
         cout<<x<<endl;
      }
   }
   return 0;
}
```

1.5 Heavy Light Decomposition

```
// works over a segment tree st

struct HLDES{
    ll n, gid;
    ll tsz[1<<17], rt[1<<17], id[1<<17],
        idb[1<<17], d[1<<17], p[1<<17],
        val[1<<17];
    vii adj[1<<17];
    vi v[1<<17];
    inline void dfs(ll u, ll pp, ll depth, ll w){
        tsz[u]=1, d[u]=depth, p[u]=pp, val[u]=w;
        for(auto xd: adj[u]) if(xd.ff!=pp)
            dfs(xd.ff, u, depth+1, xd.ss),
                tsz[u]+=tsz[xd.ff];
    }
    inline void go(ll u, ll root){
        rt[u]=root, id[u]=gid++;
        ll w=-1, bc=0;
        for(auto xd: adj[u]) if(xd.ff!=p[u] and
            tsz[xd.ff]>w) w=tsz[xd.ff], bc=xd.ff;
        if(bc) go(bc, root);
        for(auto xd: adj[u]) if(xd.ff!=p[u] and
            xd.ff!=bc) go(xd.ff, xd.ff);
    }

    inline void goback(){
        FER(i, 0, n) v[rt[i]].pb(i);
        FER(i, 0, n) if(sz(v[i])) for(auto xd:
            v[i]){
            idb[xd]=id[i]+sz(v[i])-(d[xd]-d[i])-1;
        }
    }
    inline void upd(ll a, ll b){
        ll l, r;
        while(rt[a]!=rt[b]){
            if(d[rt[a]]>d[rt[b]]) swap(a, b);
            l=id[rt[b]], r=id[b]+1;
            st.upd(l, r, 1);
            b=p[rt[b]];
        }
        if(d[a]>d[b]) swap(a, b);
        l=id[a]+1, r=id[b]+1;
        st.upd(l, r, 1);
    }

    inline void build(){
        st.n=n, gid=0;
        dfs(0, -1, 0, 0);
        go(0, 0);
        goback();
        FER(i, 0, n) st.ar[id[i]]=val[i];
        st.build();
    }
}hld;
```

1.6 LCA Tree

```
vi adj[1<<17];
ll p[1<<17][18], be[1<<17], en[1<<17], pos;

inline void dfs(ll u, ll pp){
    p[u][0]=pp, be[u]=pos++;
    FER(i, 1, 18) if(p[u][i-1]!=-1)
        p[u][i]=p[p[u][i-1]][i-1];
    for(auto xd: adj[u]) if(xd!=pp) dfs(xd, u);
    en[u]=pos++;
}

inline bool test(ll a, ll b){
    if(a==-1) return true;
    return be[a]<=be[b] && en[a]>=en[b];
}

inline ll lca(ll a, ll b){
    if(test(a, b)) return a;
    if(test(b, a)) return b;
    IFR(i, 17, 0) if(!test(p[a][i], b)) a=p[a][i];
    return p[a][0];
}

inline void build() {fill(p, -1), dfs(0, -1);}
```

1.7 Lazy Segment Tree

```
struct ST{
       ll n, lazy[1<<19], ar[1<<19];
       ii tree[1<<19];

       inline ii Op(ii &val1, ii &val2) { return
           val1.ff>val2.ff? val1: val2;}
       inline void updpro(ll laz, ll id, ll l, ll
           r) {
           if(laz) {
                  tree[id].ff+=laz;
                  lazy[id]+=laz;
           }
```

```
}
inline void proh(ll id, ll l, ll r){
       ll mid=(l+r)>>1;
       updpro(lazy[id], id<<1, l, mid);
       updpro(lazy[id], id<<1|1, mid, r);
       lazy[id]=0;
}
inline void Upd(ll x, ll y, ll val, ll id,
    ll l, ll r) {
       if(x>=r or y<=l) return ;
       if(x<=l and r<=y) {
              updpro(val, id, l, r);
              return;
       }
       proh(id, l, r) ;
       ll mid=(l+r)>>1;
       Upd(x, y, val, id<<1, l, mid);
       Upd(x, y, val, id<<1|1, mid, r);
       tree[id]=Op(tree[id<<1],
           tree[id<<1|1]);
}
inline ii Que(ll x, ll y, ll id, ll l, ll
    r) {
       if(x>=r or y<=l) return ii{-INF,
           -1};
       if(x<=l and r<=y) return tree[id];
       proh(id, l, r);
       ll mid=(l+r)>>1;
       ii left, right;
       left=Que(x, y, id<<1, l, mid);
       right=Que(x, y, id<<1|1, mid, r);
       return Op(left, right);
}
inline void Build(ll id, ll l, ll r) {
       if(l+1==r) {
              tree[id]=ii{ar[l], l};
              return ;
       }
       ll mid=(l+r)>>1;
       Build(id<<1, l, mid),
           Build(id<<1|1, mid, r) ;
       tree[id]=Op(tree[id<<1],
           tree[id<<1|1]);
}
       inline void build() { fill(lazy, 0),
           Build(1, 0, n);}
       inline void upd(ll x, ll y, ll val) {
           Upd(x, y, val, 1, 0, n);}
       inline ii que(ll x, ll y) { return Que(x,
           y, 1, 0, n);}
}st;
```

1.8 Link Cut Tree

```
//Link cut tree

const int N = 1e5 + 2;

struct Node {
    Node *left, *right, *parent;
    bool revert;
    Node() : left(0), right(0), parent(0),
        revert(false) {}
    bool isRoot() {
        return parent == NULL ||
            (parent->left != this && parent->right
                != this);
    }
    void push() {
        if (revert) {
            revert = false;
            Node *t = left;
            left = right;
            right = t;
            if (left != NULL) left->revert =
                !left->revert;
            if (right != NULL) right->revert =
                !right->revert;
        }
    }
};

struct LinkCutTree{
    Node nos[N];

    LinkCutTree(){
        FER(i,0,N) nos[i] = Node();
    }

    void connect(Node *ch, Node *p, bool
        isLeftChild) {
        if (ch != NULL) ch->parent = p;
        if (isLeftChild) p->left = ch;
        else p->right = ch;
    }

    void rotate(Node *x){
        Node* p = x->parent;
        Node* g = p->parent;
        bool isRoot = p->isRoot();
        bool leftChild = x == p->left;

        connect(leftChild ? x->right : x->left, p,
            leftChild);
        connect(p, x, !leftChild);
        if (!isRoot) connect(x, g, p == g->left);
        else x->parent = g;
    }

    void splay(Node *x){
        while (!x->isRoot()) {
            Node *p = x->parent;
            Node *g = p->parent;
            if (!p->isRoot()) g->push();
            p->push();
            x->push();
            if (!p->isRoot()) {
                rotate((x == p->left) == (p ==
                    g->left) ? p : x);
            }
            rotate(x);
        }
        x->push();
    }

    Node *expose(Node *x) {
        Node *last = NULL, *y;
        for (y = x; y != NULL; y = y->parent) {
            splay(y);
            y->left = last;
```

```
            last = y;
        }
        splay(x);
        return last;
    }

    void makeRoot(Node *x) {
        expose(x);
        x->revert = !x->revert;
    }

    bool connected(Node *x, Node *y) {
        if (x == y) return true;
        expose(x);
        expose(y);
        return x->parent != NULL;
    }

    bool link(Node *x, Node *y) {
        if (connected(x, y)) return false;
        makeRoot(x);
        x->parent = y;
        return true;
    }

    bool cut(Node *x, Node *y) {
        makeRoot(x);
        expose(y);
        if (y->right != x || x->left != NULL ||
            x->right != NULL)
            return false;
        y->right->parent = NULL;
        y->right = NULL;
        return true;
    }
};
```

1.9 Persistent Lazy Segment Tree Beats

```
struct T{
    ll fmx, fmn, smx, smn, cmx, cmn;
    ll sum, lval, add;
    T(){}
    T(ll fmx, ll fmn, ll smx, ll smn, ll cmx, ll
        cmn, ll sum):
        fmx(fmx), fmn(fmn), smx(smx), smn(smn),
            cmx(cmx), cmn(cmn), sum(sum){}
    inline void clear(){
        lval=INF, add=0;
    }
    inline T Op(T a, T b){
        if(a.fmx<b.fmx){
            a.cmx=0;
            a.smx=a.fmx;
            a.fmx=b.fmx;
        }
        if(a.fmx==b.fmx){
            a.cmx+=b.cmx;
            a.smx=max(a.smx, b.smx);
        }
        else a.smx=max(a.smx, b.fmx);
        if(a.fmn>b.fmn){
            a.cmn=0;
            a.smn=a.fmn;
            a.fmn=b.fmn;
        }
        if(a.fmn==b.fmn){
            a.cmn+=b.cmn;
            a.smn=min(a.smn, b.smn);
        }
        else a.smn=min(a.smn, b.fmn);
        a.sum+=b.sum;
        a.clear();
        return a;
    }
    inline void UMax(ll x){
        sum+=(x-fmn)*cmn;
        if(fmx==fmn) fmx=x;
        if(smx==fmn) smx=x;
        fmn=x;
        if(lval!=INF && lval<x) lval=x;
    }
    inline void Umin(ll x){
        sum+=(x-fmx)*cmx;
        if(fmn==fmx) fmn=x;
        if(smn==fmx) smn=x;
        fmx=x;
        if(lval!=INF && lval>x) lval=x;
    }


};

struct PLSTB{
    ll n;
    vi L, R, ar;
    vector<T> lazy;
    vector<bool> IsLazy;
    inline void updpro(T val , ll id, ll l, ll r){
        if(val.fmx<lazy[id].fmx){
            lazy[id].Umin(val.fmx);
        }
        if(val.fmn>lazy[id].fmn){
            lazy[id].UMax(val.fmn);
        }
    }

    inline void updadd(ll val, ll id, ll l, ll r){
        lazy[id].fmx+=val, lazy[id].fmn+=val;
        if(lazy[id].smx!=-INF) lazy[id].smx+=val;
        if(lazy[id].smn!=INF) lazy[id].smn+=val;
        lazy[id].sum+=val*(r-l);
        if(lazy[id].lval!=INF) lazy[id].lval+=val;
        else lazy[id].add+=val;
    }

    inline void updput(ll val ,ll id, ll l, ll r){
        lazy[id].fmx=val, lazy[id].fmn=val;
        lazy[id].smx=-INF, lazy[id].smn=INF;
        lazy[id].cmx=r-l;
        lazy[id].cmn=r-l;
        lazy[id].sum=val*(r-l);
        lazy[id].lval=val;
        lazy[id].add=0LL;
    }

    inline ll newLazyCre(ll id, ll l, ll r){
        L.pb(L[id]), R.pb(R[id]);
```

```
    lazy.pb(lazy[id]), IsLazy.pb(true);
    return sz(lazy)-1;
}
inline ll newParent(ll l, ll r){
    L.pb(l), R.pb(r), IsLazy.pb(false);
    lazy.pb(lazy[0].Op(lazy[l], lazy[r]));
    return sz(lazy)-1;
}

inline ll newLeaf(ll val){
    ll p=sz(lazy);
    lazy.pb(T{val, val, -INF, INF, 1, 1, val});
    IsLazy.pb(false), L.pb(0), R.pb(0);
    lazy[p].clear();
    return p;
}

inline void proh(ll id, ll l, ll r){
    if(IsLazy[id]){
        if(l>=r) return;
        ll mid=(l+r)>>1;
        L[id]=newLazyCre(L[id], l, mid);
        R[id]=newLazyCre(R[id], mid, r);
        if(lazy[id].lval!=INF){
            updput(lazy[id].lval, L[id], l,
                mid);
            updput(lazy[id].lval, R[id], mid,
                r);
            lazy[id].lval=INF;
            IsLazy[id]=false;
            return;
        }
        if(lazy[id].add){
            updadd(lazy[id].add, L[id], l, mid);
            updadd(lazy[id].add, R[id], mid, r);
            lazy[id].add=0LL;
        }
        updpro(lazy[id], L[id], l, mid);
        updpro(lazy[id], R[id], mid, r);
        IsLazy[id]=false;
    }
}
```

```
inline ll UpdMin(ll x, ll y, ll val, ll id,
    ll l, ll r){
    if(x>=r or y<=l or lazy[id].fmx<=val)
        return id;
    if(x<=l and r<=y and lazy[id].smx<val and
        lazy[id].fmx>val){
        ll idx=newLazyCre(id, l, r);
        lazy[idx].Umin(val);
        return idx;
    }
    proh(id, l, r);
    ll mid=(l+r)>>1, left, right;
    left=UpdMin(x, y, val, L[id], l, mid);
    right=UpdMin(x, y, val, R[id], mid, r);
    return newParent(left, right);
}

inline ll UpdMax(ll x, ll y, ll val, ll id,
    ll l, ll r){
    if(x>=r or y<=l or lazy[id].fmn>=val)
        return id;
    if(x<=l and r<=y and lazy[id].smn > val
        and lazy[id].fmn < val){
        ll idx=newLazyCre(id, l, r);
        lazy[idx].UMax(val);
        return idx;
    }
    proh(id, l, r);
    ll mid=(l+r)>>1, left, right;
    left=UpdMax(x, y, val, L[id], l, mid);
    right=UpdMax(x, y, val, R[id], mid, r);
    return newParent(left, right);
}

inline ll UpdQue(ll x, ll y, ll val, ll id,
    ll l, ll r){
    if(x>=r or y<=l) return id;
    if(x<=l and r<=y){
        ll idx=newLazyCre(id, l, r);
        updadd(val, idx, l, r);
        return idx;
    }
    proh(id, l, r);
    ll mid=(l+r)>>1, left, right;
    left=UpdQue(x, y, val, L[id], l, mid);
    right=UpdQue(x, y, val, R[id], mid, r);
    return newParent(left, right);
}

inline ll UpdPut(ll x, ll y, ll val, ll id,
    ll l, ll r){
    if(x>=r or y<=l) return id;
    if(x<=l and r<=y) {
        ll idx=newLazyCre(id, l, r);
        updput(val, idx, l, r);
        return idx;
    }
    proh(id, l, r);
    ll mid=(l+r)>>1, left, right;
    left=UpdPut(x, y, val, L[id], l, mid);
    right=UpdPut(x, y, val, R[id], mid, r);
    return newParent(left, right);
}

inline ll Query(ll x, ll y, ll id, ll l, ll
    r){
    if(x>=r or y<=l) return 0LL;
    if(x<=l and r<=y) return lazy[id].sum;
    proh(id, l, r);
    ll mid=(l+r)>>1, left, right;
    left=Query(x, y, L[id], l, mid);
    right=Query(x, y, R[id], mid, r);
    return left+right;
}

inline ll Build(ll l, ll r){
    ll mid=(l+r)>>1;
    if(l+1==r) return newLeaf(ar[l]);
    ll left=Build(l, mid), right=Build(mid, r);
    return newParent(left, right);
}

inline ll que(ll x, ll y, ll root) { return
    Query(x, y, root, 0, n);}
inline ll updmax(ll x, ll y, ll val, ll root)
    { return UpdMax(x, y, val, root, 0, n);}
inline ll updmin(ll x, ll y, ll val, ll root)
    { return UpdMin(x, y, val, root, 0, n);}
```

```
    inline ll updA(ll x, ll y, ll val, ll root) {
        return UpdQue(x, y, val, root, 0, n);}
    inline ll updP(ll x, ll y, ll val, ll root) {
        return UpdPut(x, y, val, root, 0, n);}
    inline ll build() { n=sz(ar); return Build(0,
        n);}

}st;


int main(){
    fastio;
    ll n, q; cin>>n>>q;
    FER(i,0,n) {
        ll x; cin>>x;
        st.ar.pb(x);
    }
    ll root=st.build();
    ll l, r, x, t;
    FER(i,0,q){
        cin>>t>>l>>r; l--;
        if(t==1){
            cin>>x;
            root=st.updmin(l, r, x, root);
        }
        else if(t==2){
            cin>>x;
            root=st.updmax(l, r, x, root);
        }
        else if(t==3){
            cin>>x;
            root=st.updA(l, r, x, root);
        }
        else if(t==4){
            cin>>x;
            root=st.updP(l, r, x, root);
        }
        else{
            x=st.que(l, r, root);
            cout<<x<<endl;
        }
    }
    return 0;
}
```

1.10 Persistent Lazy Segment Tree

```
// Persistent Lazy Segment Tree for a hash value
// All implementation are based on The clasical
    problem

struct T{
    ll a, b, t, s;
    T(){}
    T(ll a, ll b, ll t, ll s): a(a), b(b), t(t),
        s(s){}
    inline void clear(){
        a=b=t=s=0;
    }
};

struct PLST{
    ll n;
    vi L, R, lazy, ar;
    vector<bool> IsLazy;
    vector<T> tree;

    inline T Op(T val1, T val2){
        T ty;
        ty.t=val1.t+val2.t;
        ty.s=val1.s+val2.s;
        ty.a=(val1.a*dp1[val2.t])%mod1;
        ty.a+=val2.a;
        ty.b=(val1.b*dp2[val2.t])%mod2;
        ty.b+=val2.b;
        if(ty.a>=mod1) ty.a-=mod1;
        if(ty.b>=mod2) ty.b-=mod2;
        return ty;
    }

    inline ll newLazy(ll id, ll add, ll l, ll r){
        L.pb(L[id]), R.pb(R[id]), lazy.pb(add),
            IsLazy.pb(true);
      ll p=sz(lazy)-1;
        if(add==INF){
        tree.pb(tree[id]);
        lazy[p]=lazy[id];
        IsLazy[p]=false;
           return p;
        }
        T cur;
        cur.t=r-l, cur.s=(r-l)*add;
        if(add==0) cur.a=ln1[r-l], cur.b=ln2[r-l];
        if(add==1) cur.a=len1[r-l],
            cur.b=len2[r-l];
        tree.pb(cur);
        return p;
    }

    inline ll newParent(ll l, ll r){
        L.pb(l), R.pb(r), tree.pb(Op(tree[l],
            tree[r]));
        lazy.pb(INF), IsLazy.pb(false);
        return sz(tree)-1;
    }

    inline ll newLeaf(T val){
        L.pb(0), R.pb(0), tree.pb(val);
        lazy.pb(INF), IsLazy.pb(false);
         return sz(tree)-1;
    }

    inline void proh(ll id, ll l, ll r){
        if(IsLazy[id]){
            if(l>=r) return;
            ll mid=(l+r)>>1;
            L[id]=newLazy(L[id], lazy[id], l, mid);
            R[id]=newLazy(R[id], lazy[id], mid, r);
            IsLazy[id]=false;
            lazy[id]=INF;
            return;
        }
    }

    inline ll updR(ll x, ll y, ll add, ll id, ll
        l, ll r){
        if(x>=r || y<=l) return id;
        if(x<=l && r<=y) return newLazy(id, add,
            l, r);
```

```
    proh(id, l, r);
    ll mid=(l+r)>>1;
    ll left=updR(x, y, add, L[id], l, mid),
        right=updR(x, y, add, R[id], mid, r);
    return newParent(left, right);
}

inline T Query(ll x, ll y, ll id, ll l, ll r){
    if(x>=r || y<=l) return T(0LL, 0LL, 0LL,
        0LL);
    if(x<=l && r<=y) return tree[id];
    proh(id, l, r);
    ll mid=(l+r)>>1;
    T left=Query(x, y, L[id], l, mid);
    T right=Query(x, y, R[id], mid, r);
    return Op(left, right);
}

inline ll Build(ll l, ll r){
    ll mid=(l+r)>>1;
    if(l+1==r) return newLeaf(T(ar[l], ar[l],
        1, 0));
    ll left=Build(l, mid), right=Build(mid, r);
    return newParent(left, right);
}

inline ll que(ll l, ll r, ll timer) {
    T cur=Query(l, r, timer, 0, n);
    return cur.s;
}

inline ll find(ll a, ll p, ll l, ll r){
    proh(a, l, r);
    if(l>p) return -1;
    if(l+1==r) return tree[a].s==tree[a].t?
        -1: l;
    ll mid=(l+r)>>1;
    ll right=find(R[a], p, mid, r);
    if(right!=-1) return right;
    return find(L[a], p, l, mid);
}

inline ll upd(ll x, ll y, ll add, ll timer) {
    return updR(x, y, add, timer, 0, n);}
```

```
    inline ll build() { return Build(0, n);}
}st;
```

1.11 Persistent Segment Tree

```
struct PST{
    ll n;
    vi L, R, tree, ar;
    inline ll Op(ll &val1, ll &val2) { return
        val1+val2;}
    inline ll newParent(ll l, ll r){
        L.pb(l), R.pb(r), tree.pb(Op(tree[l],
            tree[r]));
        return sz(tree)-1;
    }

    inline ll newLeaf(ll val){
        L.pb(0), R.pb(0), tree.pb(val);
        return sz(tree)-1;
    }

    inline ll updR(ll p, ll add, ll id, ll l, ll
        r){
        if(l+1==r) return newLeaf(Op(tree[id],
            add));
        ll mid=(l+r)>>1;
        if(p<mid) return newParent(updR(p, add,
            L[id], l, mid), R[id]);
        return newParent(L[id], updR(p, add,
            R[id], mid, r));
    }

    inline ll Query(ll x, ll y, ll id, ll l, ll
        r){
        if(x>=r or y<=l) return 0LL;
        if(x<=l and r<=y) return tree[id];
        ll mid=(l+r)>>1, left, right;
        left=Query(x, y, L[id], l, mid);
        right=Query(x, y, R[id], mid, r);
        return Op(left, right);
    }
```

```
    inline ll Build(ll l, ll r){
        ll mid=(l+r)>>1;
        if(r-l<2) return newLeaf(ar[l]);
        ll left=Build(l, mid), right=Build(mid, r);
        return newParent(left, right);
    }

    inline ll que(ll l, ll r, ll timer) { return
        Query(l, r, timer, 0, n);}
    inline ll upd(ll p, ll add, ll timer) {
        return updR(p, add, timer, 0, n);}
    inline ll build() { n=sz(ar); return Build(0,
        n);}
}st;
```

1.12 Persistent Trie

```
/* Persistent Trie */
struct Ptrie{
   ll nod, m[1<<23][2], ind[1<<24];
   inline ll AddTrie(ll last, ll i, ll val, ll
       pos){
      ll rt=nod++;
      if(i<0) return rt;
      ll r=(val>>i)&1;
      m[rt][r^1]=m[last][r^1];
      m[rt][r]=AddTrie(m[last][r], i-1, val, pos);
      ind[m[rt][r]]=pos;
      return rt;
   }

   inline ll QueryMax(ll id, ll val, ll pos){
      ll ans=0;
      IFR(i,19,0){
         ll r=(val>>i)&1;
         if(m[id][r^1]) {
            ll curnod=m[id][r^1];
            ll index=ind[curnod];
            if(index>=pos) id=m[id][r^1],
                ans+=((r^1)<<i);
            else id=m[id][r], ans+=(r<<i);
         }
```

```
        else {
          id=m[id][r], ans+=(r<<i);
        }
      }
      return ans ^ val;
   }
}st;

int main(){
   st.nod=1
   st.AddTrie(), st.QueryMax();
}
```

1.13 SQRTREE

```
ll mo;
struct T{
  ll l, r, cur;
  T(){}
  T(ll l, ll r, ll cur): l(l), r(r), cur(cur){}
};
struct SQRTREE{
  ll n, lg, indexsz;
  vector<T> v;
  vi clz, layers, onlayer;
  vector<vector<T>> pref, suf, between;
  inline T Op(T &val1, T &val2){
    T ty;
    ty.l=val1.l;
    ty.r=val2.r;
    ty.cur=((val1.cur*val2.cur)%mo+mo)%mo;
    return ty;
  }
  inline ll Get(ll n){
    ll ans=0;
    while((1<<ans)<n) ans++;
    return ans;
  }
  inline void preprocess(vector<T> &a){
    n=sz(a), lg=Get(n), v=a;
    clz.resize(1<<lg), onlayer.resize(lg+1);
    FER(i, 0, sz(clz)) clz[i]=(i==0)? 0 :
        clz[i>>1]+1;
    ll tlg=lg;
    while(tlg>1){
      onlayer[tlg]=sz(layers);
      layers.pb(tlg);
      tlg=(tlg+1)>>1;
    }
    IFR(i, lg-1, 0) onlayer[i]=max(onlayer[i],
        onlayer[i+1]);
    ll mid=max(0LL, sz(layers)-1);
    ll t=(lg+1)>>1, m=1<<t;
    indexsz=(n+m-1)>>t;
    v.resize(n+indexsz);
    pref.assign(sz(layers), vector<T>(n+indexsz));
    suf.assign(sz(layers), vector<T>(n+indexsz));
    between.assign(mid, vector<T>((1<<lg)+m));
  }
  inline void BuildBlock(ll id, ll l, ll r){
    FER(i, l, r) pref[id][i]=(i==l)? v[i]:
        Op(pref[id][i-1], v[i]);
    IFR(i, r-1, l) suf[id][i]=(i==r-1)? v[i]:
        Op(v[i], suf[id][i+1]);
  }
  inline void BuildBetween(ll id, ll l, ll r, ll
      idx){
    ll t=(layers[id]+1)>>1, m=layers[id]>>1;
    ll ta1=1<<t, len=(r-l+ta1-1)>>t;
    FER(i, 0, len){
      T ans;
      FER(j, i, len){
        T add=suf[id][l+(j<<t)];
        ans=(i==j)? add: Op(ans, add);
        between[id-1][idx+l+(i<<m)+j]=ans;
      }
    }
  }

  inline void BuildZero(){
    ll m=(lg+1)>>1;
    FER(i, 0, indexsz) v[n+i]=suf[0][i<<m];
    build(1, n, n+indexsz, (1<<lg)-n);
  }
  inline void updZero(ll id){
    ll m=(lg+1)>>1;
    v[n+id]=suf[0][id<<m];
    update(1, n, n+indexsz, (1<<lg)-n, n+id);
  }

  inline void build(ll id, ll l, ll r, ll idx){
    if(id>=sz(layers)) return;
    ll m=1<<((layers[id]+1)>>1);
    for(ll left=l; left<r; left+=m){
      ll right=min(left+m, r);
      BuildBlock(id, left, right);
      build(id+1, left, right, idx);
    }
    id? BuildBetween(id, l, r, idx) : BuildZero();
  }
  inline void update(ll id, ll l, ll r, ll idx,
      ll p){
    if(id>=sz(layers)) return ;
    ll m=(layers[id]+1)>>1, t=1<<m;
    ll bid=(p-l)>>m;
    ll left=l+(bid<<m), right=min(l+t, r);
    BuildBlock(id, left, right);
    id? BuildBetween(id, l, r, idx): updZero(bid);
    update(id+1, left, right, idx, p);
  }
  inline T query(ll l, ll r, ll idx, ll b){
    if(l==r) return v[l];
    if(l+1==r) return Op(v[l], v[r]);
    ll id=onlayer[clz[(l-b) ^ (r-b)]];
    ll t=(layers[id]+1)>>1, m=layers[id]>>1;
    ll in=(((l-b)>>layers[id])<<layers[id])+b;
    ll left=((l-in)>>t)+1, right=((r-in)>>t)-1;
    T ans=suf[id][l];
    if(left<=right){
      ll idd=idx+in+(left<<m)+right;
      T add=(id==0)? query(n+left, n+right,
          (1<<lg)-n, n):between[id-1][idd];
      ans=Op(ans, add);
    }
    ans=Op(ans, pref[id][r]);
    return ans;
  }
  inline T que(ll l, ll r){
    return query(l, r, 0, 0);
```

```
  }
  inline void upd(ll p, T val){
    v[p]=val;
    update(0, 0, n, 0, p);
  }
  inline void build(){
    build(0, 0, n, 0);
  }
};
ll queries[N];
int main(){
  fastio;
  ll t; cin>>t;
  FER(ikl, 0, t){
    SQRTREE sqrtree;
    ll n, q; cin>>n>>mo>>q;
    vector<T> a;
    FER(i, 0, n){
      T x;
      cin>>x.cur;
      x.l=i, x.r=i;
      a.pb(x);
    }
    sqrtree.preprocess(a);
    sqrtree.build();
    ll q1=(q>>6)+2;
    ll l, r;
    FER(i,0,q1) cin>>queries[i];
    ll ja=0;
    FER(i, 0, q){
      (i%(1<<6))? (l=(l+ja)%n, r=(r+ja)%n):
          (l=(queries[i>>6]+ja)%n,
          r=(queries[(i>>6)+1]+ja)%n);
      if(l>r) swap(l, r);
      ja=(sqrtree.que(l, r).cur+1)%mo;
    }
    cout<<ja<<"\n";
  }
  return 0;
}
```

1.14 Segment Tree

```
//Segment Tree

struct SegmentTree{
  int n, t[2 * N];
  int OpId = 0;
  SegmentTree() {}
  int Op(int &u, int &v){ return u + v; }
  void build(){ IFR(i, n - 1, 1) t[i]=Op(t[i <<
      1], t[i << 1 | 1]); }
  void modify(int p, int val){ for(t[p+=n] = val
      ; p >>= 1;) t[p] = Op(t[p << 1], t[p << 1 |
      1]); }
  int query(int l, int r){
   int ansl, ansr;
   ansl = ansr = OpId;
   for(l += n, r += n; l < r; l >>= 1, r >>= 1){
      if(l&1) ansl = Op(ansl, t[l++]);
      if(r&1) ansr = Op(t[--r], ansr);
   }
   return Op(ansl, ansr);
  }
};
```

1.15 Splay Tree

```
//-----------------------------------------------
// Lazy splay tree propagation
// make -> n, preprocess() y build()
// Be careful about the lazy push values
// there isn't need to use an Op function (this
    may cause TL)

struct Node{
   Node *child[2], *p;
   bool t;
   ll val, ta, sum, add, l, r, mid;
   inline void clear(){
      ta=-INF;
   }
};

struct SplayTree{
   Node *nil, *root;
   ll ar[1<<18], n;
   inline void preprocess(){
      nil=new Node();
      nil->child[0] = nil->child[1] = nil->p=nil;
      nil->val=nil->ta=nil->sum=0;
      nil->t=false;
      nil->l=-INF;
      nil->r=-INF;
      nil->mid=-INF;
      nil->add=0;
      root=nil;
   }
   inline void upd(Node *&x){
      x->ta=x->child[0]->ta + x->child[1]->ta+1;
      x->sum=x->child[0]->sum+x->child[1]->sum+x->val;
      ll l=max(x->child[0]->l,
          x->child[0]->sum+x->val);
      ll r=max(x->val, x->child[0]->r+x->val);
      ll mid=max(l, max(r, max(x->val,
          max(x->child[0]->mid,
          x->child[0]->r+x->val))));
      ll l1=max(l,
          x->child[0]->sum+x->val+x->child[1]->l);
      ll r1=max(x->child[1]->r,
          x->child[1]->sum+r);
      ll mid1=max(l1, max(r1,
          max(x->child[1]->mid, max(mid,
          r+x->child[1]->l))));
      x->l=l1, x->r=r1, x->mid=mid1;
   }
   inline void updpro(Node *&x, ll val){
      x->add+=val;
      x->val+=val;
      x->sum+=x->ta*val;
      x->l+=x->ta*val;
      x->r+=x->ta*val;
      x->mid+=x->ta*val;
   }
   inline void push(Node *&x){
      if(x==nil) return;
```

```
    if(x->add!=0){
      if(x->child[0]!=nil) updpro(x->child[0],
          x->add);
      if(x->child[1]!=nil) updpro(x->child[1],
          x->add);
      x->add=0;
    }
    if(x->t){
      swap(x->child[0], x->child[1]);
      x->child[0]->t=!x->child[0]->t;
      x->child[1]->t=!x->child[1]->t;
      x->t=false;
    }
  }
  inline void set(Node *x, Node *y, ll d){
    x->child[d]=y;
    y->p=x;
  }
  inline ll get(Node *x, Node *y){
    return x->child[0]==y? 0 : 1;
  }
  inline ll rot(Node *&x, ll d){
    Node *y=x->child[d], *z=x->p;
    set(x, y->child[d^1], d);
    set(y, x, d^1);
    set(z, y, get(z, x));
    upd(x), upd(y);
  }
  inline void splay(Node *&x){
    push(x);
    while(x->p != nil){
      Node *y=x->p, *z=y->p;
      ll dy=get(y, x), dz=get(z, y);
      if(z==nil) rot(y, dy);
      else if(dy==dz) rot(z, dz), rot(y, dy);
      else rot(y, dy), rot(z, dz);
    }
    upd(x);
  }
  inline Node * getnode(Node *x, ll pos){
    while(push(x), x->child[0]->ta!=pos){
      pos<x->child[0]->ta? x=x->child[0]:
          (pos-=x->child[0]->ta+1,
          x=x->child[1]);
    }
    return splay(x), x;
  }
  inline void split(Node *x, ll l, Node * &t1,
      Node * &t2){
    if(l==0) t1=nil, t2=x;
    else{
      t1=getnode(x, l-1);
      t2=t1->child[1];
      t1->child[1]=t2->p =nil;
      upd(t1);
    }
  }
  inline Node * unir(Node *x, Node *y){
    if(x==nil) return y;
    x=getnode(x, x->ta-1);
    set(x, y, 1);
    upd(x);
    return x;
  }
  inline void AddRange(ll l, ll r, ll add){
    Node *t1, * t2, * t3;
    split(root, r, t1, t3);
    split(t1, l, t1, t2);
    t2->add+=add;
    t2->val+=add;
    t2->sum+=t2->ta * add;
    root=unir(t1, unir(t2, t3));
  }
  inline void modify(ll pos, ll val){
    root=getnode(root, pos);
    root->val=val;
    upd(root);
  }
  inline void AddVal(ll pos, ll val){
    Node *t1, *t2;
    Node *cur=new Node();
    cur->val=val, cur->p=nil, cur->t=false;
    cur->ta=1, cur->sum=val;
    cur->child[0]=cur->child[1]=nil;
    ll r=pos;
    if(r==root->ta || r==0){
      if(r==0) root=unir(cur, root);
      else root=unir(root, cur);
      return;
    }
    split(root, r, t1, t2);
    root=unir(unir(t1, cur), t2);
  }
  inline void DelVal(ll pos){
    Node *t1, *t2, *t3;
    ll l=pos, r=pos+1;
    if(l==0 || r==root->ta){
      if(l==0){
        split(root, r, t1, t2);
        root=t2;
      }
      else{
        split(root, l, t1, t2);
        root=t1;
      }
      return;
    }
    split(root, r, t1, t3);
    split(t1, l, t1, t2);
    root=unir(t1, t3);
    delete t2;
    return;
  }
  inline void Reverse(ll l, ll r){
    Node * t1, * t2, * t3;
    split(root, r, t1, t3);
    split(t1, l, t1, t2);
    t2->t=!t2->t;
    root=unir(unir(t1, t2), t3);
  }
  inline void move(ll l, ll r, ll l1, ll r1){
    Node *t1, *t2, *t3, *t4, *t5;
    split(root, r1, t4, t5);
    split(t4, l1, t3, t4);
    split(t3, r, t2, t3);
    split(t2, l, t1, t2);
    root=unir(unir(t1,unir(t4, t3)), unir(t2,
        t5));
  }
  inline ll que(ll l, ll r){
    Node *t1, * t2, *t3;
    split(root, r, t1, t3);
```

```
    split(t1, l, t1, t2);
    ll ans=t2->mid;
    root=unir(unir(t1, t2), t3);
    return ans;
  }
  inline ll Permute(ll l, ll r){
    ll x=que(r-1, r);
    AddVal(l, x);
    DelVal(r);
  }
  inline Node * build1(ll l, ll r){
    if(l==r) return nil;
    ll mid=(l+r)>>1;
    Node *x=new Node();
    x->val=ar[mid];
    x->sum=ar[mid];
    x->mid=ar[mid];
    x->l=ar[mid];
    x->r=ar[mid];
    x->p=nil;
    x->t=false;
    x->add=0;
    set(x, build1(l, mid), 0);
    set(x, build1(mid+1, r), 1);
    upd(x);
    return x;
  }
  inline void build(){
    Node *root1=build1(0, n);
    root=root1;
  }
}st;


int main(){
  ll n; scanf("%lld", &n);
  st.n=n;
  FER(i,0,n) scanf("%lld", &st.ar[i]);
  st.preprocess();
  st.build();
  ll q, l, r; scanf("%lld", &q);
  char s;
  FER(i,0,q){
    scanf(" %c", &s);
    if(s=='I'){
      scanf("%lld%lld", &l, &r); l--;
      st.AddVal(l, r);
    }
    else if(s=='D'){
      scanf("%lld", &l); l--;
      st.DelVal(l);
    }
    else if(s=='R'){
      scanf("%lld%lld", &l, &r); l--;
      st.modify(l, r);
    }
    else{
      scanf("%lld%lld", &l, &r); l--;
      ll froz=st.que(l, r);
      printf("%lld\n", froz);
    }
  }
  return 0;
}
```

1.16 ksetTree

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
using namespace __gnu_pbds;

tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> X;
// El primer parametro int es el tipo de dato del
    set
// El segundo parametro es null_type para set, u
    otro tipo para map. Por ejemplo
    tree<string,int,...> es un map<string,int>
// El tercer parmetro es la funcin de
    comparacin, por defecto usar less<T>
// El cuarto y el quinto se ponen como estn
// soportan las operaciones *x.find_by_order(k);
    x.order_of_key(key);
typedef tree<int, null_type, less<int>,
    rb_tree_tag,
    tree_order_statistics_node_update> kset;

/*Implementacion usada para el subregional de
    Moscow 2008*/
template <class T> struct ord{
      bool operator() (const ii &x, const ii& y)
          const{
            if(x.fst != y.fst)
                  return x.fst > y.fst;
            return x.snd < y.snd;
      }
};

typedef tree< ii,string, ord<ii>,rb_tree_tag,
    tree_order_statistics_node_update> kset;

int main(){

      kset x;



      return 0;
}
```

2 Flujos

2.1 Bipartite Matching

```
/* Bipartite matching BFS-DFS
   Autor: elManco
*/

const int INF = 1e9;
int n, m, nil, match[N], dist[N];
vector<int> adj[N];

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) continue;
      for(int v : adj[u]){
         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 v : adj[u]){
         if(dist[match[v]] != dist[u] + 1)
             continue;
         if(dfs(match[v])){
            match[v] = u;
            match[u] = v;
            return true;
         }
      }
      dist[u] = INF;
      return false;
   }
   return true;
}

int bipMatching(){
   nil = n + m + 1;
   for(int i = 1; i <= n+m; ++i)
      match[i] = nil;
   int matching = 0;
   while(bfs()){
      for(int i = 1; i <= n; ++i)
         if(match[i] == nil and dfs(i))
            ++matching;
   }
   return matching;
}
```

2.2 Dinic Max Flow

```
/*  Dinic Max Flow
   Autor: Matias Hunicken
   Intrucciones de uso:
      Min cut: nodes with dist>=0 vs nodes with
          dist<0
      Matching MVC: left nodes with dist<0 + right
          nodes with dist>0
*/

struct Dinic{
   int nodes,src,dst;
   vector<int> dist,q,work;
   struct edge {int to,rev;ll f,cap;};
   vector<vector<edge>> g;
   Dinic(int
       x):nodes(x),g(x),dist(x),q(x),work(x){}
   void add_edge(int s, int t, ll cap){
      g[s].pb((edge){t,SZ(g[t]),0,cap});
      g[t].pb((edge){s,SZ(g[s])-1,0,0});
   }
   bool dinic_bfs(){
      fill(ALL(dist),-1);dist[src]=0;
      int qt=0;q[qt++]=src;
      for(int qh=0;qh<qt;qh++){
         int u=q[qh];
         FER(i,0,SZ(g[u])){
            edge &e=g[u][i];int v=g[u][i].to;
            if(dist[v]<0&&e.f<e.cap)
               dist[v]=dist[u]+1,q[qt++]=v;
         }
      }
      return dist[dst]>=0;
   }
   ll dinic_dfs(int u, ll f){
      if(u==dst)return f;
      for(int &i=work[u];i<SZ(g[u]);i++){
         edge &e=g[u][i];
         if(e.cap<=e.f)continue;
         int v=e.to;
         if(dist[v]==dist[u]+1){
            ll df=dinic_dfs(v,min(f,e.cap-e.f));
            if(df>0){e.f+=df;g[v][e.rev].f-=df;return
                df;}
         }
      }
      return 0;
   }
   ll max_flow(int _src, int _dst){
      src=_src;dst=_dst;
      ll result=0;
      while(dinic_bfs()){
         fill(ALL(work),0);
         while(ll
             delta=dinic_dfs(src,INF))result+=delta;
      }
      return result;
   }
};
```

2.3 Edmonds Blossom

```
/* Edmonds Blossom - General matching para grafos
    generales
   Autor: Phibrain
*/

struct EdmondsBlossom{
   ll n;
   vi match, vis, tmp;
   set<ii> used;
   inline void join(ll a, ll b) {
      match[a]=b, match[b]=a;
   }
   inline bool dfs(ll k, vector<vi> &com, vi
       &blossom){
      vis[k] = 0;
```

```
    FER(i, 0, n) if(com[k][i]){
      if(vis[i] == -1){
        vis[i] = 1;
        if(match[i] == -1 || dfs(match[i],
            com, blossom)) { join(k, i);
            return true;}
      }
      if(vis[i] == 0 || sz(blossom)){
        blossom.pb(i), blossom.pb(k);
        if(k == blossom[0]) { match[k] = -1;
            return true;}
        return false;
      }
    }
    return false;
}
inline bool augment(vector<vi> &com){
    FER(m, 0, n) if(match[m] == -1){
      vi blossom;
      vis = vi(n, -1);
      if(!dfs(m, com, blossom)) continue;
      if(sz(blossom) == 0) return true;

      ll base = blossom[0], s = sz(blossom);
      vector<vi> newcom = com;

        FER(i, 1, s-1) FER(j, 0, n)
            newcom[base][j] = newcom[j][base]
            |= com[blossom[i]][j];
      FER(i, 1, s-1) FER(j, 0, n)
          newcom[blossom[i]][j] =
          newcom[j][blossom[i]] = 0;
      newcom[base][base] = 0;

        if(!augment(newcom)) return false;

        ll k = match[base];
        if(k != -1) FER(i, 0, s)
            if(com[blossom[i]][k]){
        join(blossom[i], k);
        if(i&1) for(ll j = i + 1; j < s; j +=
            2) join(blossom[j], blossom[j +
            1]);
            else for(ll j = 0; j < i; j += 2)
                join(blossom[j], blossom[j + 1]);
            break;
          }
          return true;
        }
        return false;
    }
    inline ll edmonds(vector<vi> & com){
      ll res=0;
      match=vi(n, -1);
      while(augment(com)) res++;
      return res;
    }
    inline void preprocess(vector<vi> &com){
      tmp.clear();
      tmp.assign(n, 0);
      FER(i,0,n) com.pb(tmp);
      used.clear();
      match.clear();
      vis.clear();
    }
    inline void add(ll a, ll b, vector<vi> &com){
      if(a>b) swap(a, b);
      if(used.count({a, b})) return;
      used.insert({a, b});
      com[a][b]=com[b][a]=1;
    }
};
```

2.4 Gomory Hu Tree

```
// Gomory Hu Tree 0 index base
// graph is stored in vector<pi> gph

#include <bits/stdc++.h>
using namespace std;
typedef long long lint;
typedef pair<int, int> pi;

const int MAXN = 205;
struct maxflow{
    struct edg{int pos, cap, rev;};
    vector<edg> gph[MAXN];

    void clear(){
        for(int i=0; i<MAXN; i++){
            gph[i].clear();
        }
    }

    void add_edge(int s, int e, int x){
        gph[s].push_back({e, x,
            (int)gph[e].size()});
        gph[e].push_back({s, 0,
            (int)gph[s].size()-1});
    }

    int dis[MAXN], pnt[MAXN];

    bool bfs(int src, int sink){
        memset(dis, 0, sizeof(dis));
        memset(pnt, 0, sizeof(pnt));
        queue<int> que;
        que.push(src);
        dis[src] = 1;
        while(!que.empty()){
            int x = que.front();
            que.pop();
            for(int i=0; i<gph[x].size(); i++){
                edg e = gph[x][i];
                if(e.cap > 0 && !dis[e.pos]){
                    dis[e.pos] = dis[x] + 1;
                    que.push(e.pos);
                }
            }
        }
        return dis[sink] > 0;
    }

    int dfs(int x, int sink, int f){
        if(x == sink) return f;
        for(; pnt[x] < gph[x].size(); pnt[x]++){
            edg e = gph[x][pnt[x]];
            if(e.cap > 0 && dis[e.pos] == dis[x] +
                1){
```

```
                int w = dfs(e.pos, sink, min(f,
                    e.cap));
                if(w){
                    gph[x][pnt[x]].cap -= w;
                    gph[e.pos][e.rev].cap += w;
                    return w;
                }
            }
        }
        return 0;
    }

    lint match(int src, int sink){
        lint ret = 0;
        while(bfs(src, sink)){
            int r;
            while((r = dfs(src, sink, 2e9))) ret
                += r;
        }
        return ret;
    }
};

struct gomory_hu{
    struct edg{ int s, e, x; };
    vector<edg> edgs;
    maxflow mf;

    void clear(){
        edgs.clear();
    }

    void add_edge(int s, int e, int x){
        edgs.push_back({s, e, x});
    }

    bool vis[MAXN];

    void dfs(int x){
        if(vis[x]) return;
        vis[x] = 1;
        for(auto &i : mf.gph[x]){
            if(i.cap > 0) dfs(i.pos);
        }
    }

    vector<pi> solve(int n){
        vector<pi> ret(n);
        for(int i=1; i<n; i++){
            for(auto &j : edgs){
                mf.add_edge(j.s, j.e, j.x);
                mf.add_edge(j.e, j.s, j.x);
            }
            ret[i].first = mf.match(i,
                ret[i].second);
            memset(vis, 0, sizeof(vis));
            dfs(i);
            for(int j=i+1; j<n; j++){
                if(ret[j].second == ret[i].second
                    && vis[j]){
                    ret[j].second = i;
                }
            }
            mf.clear();
        }
        return ret;
    }
}gh;

vector<pi> gph[205];

int main(){
    int n, m;
    cin >> n >> m;
    for(int i=0; i<m; i++){
        int s, e, x;
        scanf("%d %d %d",&s,&e,&x);
        s--, e--;
        gh.add_edge(s, e, x);
    }
    auto w = gh.solve(n);
    for(int i=1; i<n; i++){
        gph[w[i].second].push_back(pi(i,
            w[i].first));
        gph[i].push_back(pi(w[i].second,
            w[i].first));
    }
    for(int i=0; i<n; i++){
        cout<<i<<endl;
        for(auto xd: gph[i]) cout<<xd.first<<"
            "<<xd.second<<endl;
    }
    return 0;
}
```

2.5 Lower Bound Upper Bound Flow

```
/* Max flow with upper and lower bounds

   add(a, b, c, d): crea un nodo de "a" hacia "b"
       con lower value de c
   y upper value de "d". Todos los nodos son 1
       base

   Implementacion basada en verificar que el
       frujo es feasible y para encontrar
   el max flow usa binary search

   La construccion del grafo esta dada por
       construct(sink, source, #de nodos)
   El grafo esta guardado en graph[][] y adj1[]

   Para reconstruir los caminos usar el weighted
       eulerian path

   Usar Push Relabel flow!
*/
#define N1 101

ll graph[N1][N1];
vector<tri> adj[N1];
vii adj1[N1];

inline void add(ll a, ll b, ll d, ll c){
   adj[a].pb(tri{b, {d, c}});
}

inline ll solve(ll x, ll sn, ll sr, ll n,
    PushRelabel &pr){
   ll nod=0, ands=0, neg=0;
```

```
   ll sale[N1], entra[N1], d[N1];
   fill(sale, 0), fill(entra, 0);
   FER(i,0,n+1) adj1[i].clear();
   adj[nod].clear();
   adj[nod].pb(tri{sn, {x, INF}});
   FER(i,0,n+1){
      if(sz(adj[i])) for(auto xd: adj[i]){
         sale[i]+=xd.tm2;
         entra[xd.tm1]+=xd.tm2;
      }
   }
   ll t1=n+1, t2=n+2;
   FER(i, 0, n+1) d[i]=sale[i]-entra[i];
   FER(i, 0, n+1) (d[i]>0)? ands+=d[i] :
       neg+=-d[i];
   FER(i, 0, n+1) {
      if(d[i]>0) pr.AddEdge(i, t2, d[i]);
      if(d[i]<0) pr.AddEdge(t1, i, -d[i]);
   }
   FER(i, 0, n+1){
      if(sz(adj[i])) for(auto xd: adj[i]){
         ll w=xd.tm3-xd.tm2;
         pr.AddEdge(i, xd.tm1, w);
      }
   }
   pr.AddEdge(sr, nod, INF);
   ll froz=pr.GetMaxFlow(t1, t2);
   if(froz==ands && neg==ands) return 1;
   return 0;
}

inline void go(ll x, ll sn, ll sr, ll n){
   PushRelabel pr(n+3);
   ll pe=solve(x, sn, sr, n, pr);
   map<ii, ll> m;
   for(auto xd: pr.G){
      for(auto gg: xd){
         if(gg.flow>0) m[{gg.from,
             gg.to}]+=gg.flow;
      }
   }
   fill(graph, 0);
   FER(i, 1, n+1){
     if(sz(adj[i]))for(auto xd: adj[i]){
```

```
         ll w=xd.tm2+m[{i, xd.tm1}];
         if(w==0) continue;
         graph[i][xd.tm1]=w;
         adj1[i].pb(ii{xd.tm1, w});
       }
   }
}

inline ll construct(ll sn, ll sr, ll n){
   ll ini=0, fin=INF-1;
   FER(i,0,100){
      if(ini+1==fin) break;
      ll mid=(ini+fin)>>1;
      PushRelabel pr(n+3);
      if(solve(mid, sn, sr, n, pr)) ini=mid;
      else fin=mid;
   }
   PushRelabel pr(n+3);
   ll pe=solve(ini, sn, sr, n, pr);
   if(pe && ini) {
      go(ini, sn, sr, n);
      return 1;
   }
   return 0;
}
```

2.6 Max Flow Extensions

```
// In Max Flow, to add a "demand" function d(u,v)
    which forces
// edge (u,v) to have a flow of at least d(u,v),
    let s, t be
// the source and sink respectively.

// This problem is reduced to a normal max flow
    in graph
// V' = V u {s', t'}
// with new capacities c', source s' and sink t':
// c'(s', v) = sum{ d(u,v), u in V }
// c'(v, t') = sum{ d(v,w), w in V }
// c'(u, v) = c(u,v) - d(u,v)
// c'(t, s) = INF
// There exists a solution in problem V if and
    only if the
// problem in graph V' has a saturating max flow
// (with |f| = sum{ d(u, v) / u,v in V }).
// If a max flow / min flow solution is needed,
    consider adding
// a node to control the total flow, controlling
    its
// capacity / demand in a binary search.
```

2.7 Min Cost Circulation

```
// Min Cost Circulation

struct Edge {
  int cap, flow;
  ll cost;
  Edge(int cap = 0, int flow = 0, ll cost = 0) :
   cap(cap), flow(flow), cost(cost) {}
};

struct MinCostCirculation{
   int s,t,n;
   ll maxFlow, minCost;
   vector<vector<Edge> > G; //nonexistent edge =
       Edge(0,0,0);
   vector<int> vis,parent;
   vector<ll> dist;
   MinCostCirculation (int _n = 0){
      n = _n;
      G.resize(n+1); dist.resize(n+1);
          vis.resize(n+1); parent.resize(n+1);
          //OK;
      FER(i,0,n+1) G[i].resize(n+1);
      minCost = 0;
   }
   void AddEdge(int from, int to, int cap, ll
       cost){
      G[from][to] = Edge(cap, 0, cost);
      G[to][from] = Edge(0, 0,-cost);
   }
```

```
void updateEdge(int from, int to, int ncap){
   G[from][to].cap = ncap;
}
void Augment(int u){
   FER(i,0,n+1) vis[i] = 0;
  while(!vis[u]){
     vis[u] = 1;
    u = parent[u];
  }
  int aux = u;
  ll addedCost = 0;
  ll delta = INF;
  do{
     int v = parent[u];
    addedCost += G[v][u].cost;
     delta = min(delta,(ll) G[v][u].cap -
         G[v][u].flow);
    u = v;
  }while(u != aux);
  do{
     int v = parent[u];
    G[v][u].flow += delta;
    G[u][v].flow -= delta;
    u = v;
   }while(u != aux);
  minCost += addedCost*delta;
}

bool negCycle(){
   /*Vertex n is a dummy vertex => |V| = n+1*/
   FER(i,0,n) dist[i] = INF;
   dist[n] = 0;
   parent[n] = -1;
   FER(k,0,n) FER(i,0,n+1) FER(j,0,n+1){
      if(G[i][j].flow == G[i][j].cap) continue;
      if(dist[i] + G[i][j].cost < dist[j]){
         dist[j] = dist[i] + G[i][j].cost;
         parent[j] = i;
      }
   }
   FER(i,0,n+1) FER(j,0,n+1){
      if(G[i][j].flow == G[i][j].cap) continue;
      if(dist[i] + G[i][j].cost < dist[j]){
         parent[j] = i;
            Augment(j);
            return true;
         }
      }
      return false;
   }

   ll GetMinCostCirculation(){
      while(negCycle());
      return minCost;
   }
};
```

2.8 Min Cost Max Flow

```
const int MAXN = 5010;

const ll INF = 1e15;
struct edge { int dest;ll origcap, cap; ll cost;
    int rev; };

struct MinCostMaxFlow {

   vector<edge> adj[MAXN];
   ll dis[MAXN], cost;
   int source, target, iter;
   ll cap;
   edge* pre[MAXN];
   int queued[MAXN];
   MinCostMaxFlow (){}
   void AddEdge(int from, int to, ll cap, ll
       cost) {
      adj[from].push_back(edge {to, cap, cap,
          cost, (int)adj[to].size()});
      adj[to].push_back(edge {from,0, 0, -cost,
          (int)adj[from].size() - 1});
   }

   bool spfa() {
      FER(i,0,MAXN) queued[i] = 0;
      fill(dis, dis + MAXN, INF);
      queue<int> q;
      pre[source] = pre[target] = 0;
      dis[source] = 0;
      q.emplace(source);
      queued[source] = 1;
      while (!q.empty()) {
         int x = q.front();

         ll d = dis[x];
         q.pop();
         queued[x] = 0;
         for (auto& e : adj[x]) {
            int y = e.dest;
            ll w = d + e.cost;
            if (e.cap < 1 || dis[y] <= w) continue;
            dis[y] = w;
            pre[y] = &e;
            if(!queued[y]){
               q.push(y);
               queued[y] = 1;
            }
         }
      }
      edge* e = pre[target];

      if (!e) return 0;
      while (e) {
         edge& rev = adj[e->dest][e->rev];
         e->cap -= cap;
         rev.cap += cap;
         cost += cap * e->cost;
         e = pre[rev.dest];
      }
      return 1;
   }

   pair<ll,ll> GetMaxFlow(int S, int T) {
      cap = 1, source = S, target = T, cost = 0;
      while(spfa()) {}
      ll totflow = 0;
      for(auto e: adj[source]){
         totflow += (e.origcap - e.cap);
      }
      return make_pair(totflow, cost);
   }
```

```
};
```

2.9 Push Relabel Max Flow

```
/* Push Relabel Max Flow
   Autor: itu
   Instrucciones de uso:
      O(|V|^3), corre bien para n ~ 5000
      Todos los edges con flow > 0 tienen flujo
          corriendo
*/

struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int flow, int
      index) :
   from(from), to(to), cap(cap), flow(flow),
       index(index) {}
};

struct PushRelabel {
  int N;
  vector<vector<Edge> > G;
  vector<ll> excess;
  vector<int> dist, active, count;
  queue<int> Q;

  PushRelabel(int N) :
   N(N), G(N), excess(N), dist(N), active(N),
       count(2*N) {}

  void AddEdge(int from, int to, int cap) {
   G[from].push_back(Edge(from, to, cap, 0,
       G[to].size()));
   if (from == to) G[from].back().index++;
   G[to].push_back(Edge(to, from, 0, 0,
       G[from].size() - 1));
  }

   void Enqueue(int v) {
      if (!active[v] && excess[v] > 0) {
        active[v] = true; Q.push(v);
      }
   }

  void Push(Edge &e) {
   int amt = int(min(excess[e.from], ll(e.cap -
       e.flow)));
   if (dist[e.from] <= dist[e.to] || amt == 0)
       return;
   e.flow += amt;
   G[e.to][e.index].flow -= amt;
   excess[e.to] += amt;
   excess[e.from] -= amt;
   Enqueue(e.to);
  }

  void Gap(int k) {
   for (int v = 0; v < N; v++) {
     if (dist[v] < k) continue;
     count[dist[v]]--;
     dist[v] = max(dist[v], N+1);
     count[dist[v]]++;
     Enqueue(v);
   }
  }

  void Relabel(int v) {
   count[dist[v]]--;
   dist[v] = 2*N;
   for (int i = 0; i < G[v].size(); i++)
     if (G[v][i].cap - G[v][i].flow > 0)
   dist[v] = min(dist[v], dist[G[v][i].to] + 1);
   count[dist[v]]++;
   Enqueue(v);
  }

   void Discharge(int v) {
   for (int i = 0; excess[v] > 0 && i <
       G[v].size(); i++)
     Push(G[v][i]);
   if (excess[v] > 0) {
        if (count[dist[v]] == 1) Gap(dist[v]);
        else Relabel(v);
     }
   }

  ll GetMaxFlow(int s, int t) {
   count[0] = N-1;
   count[N] = 1;
   dist[s] = N;
   active[s] = active[t] = true;
   for (int i = 0; i < G[s].size(); i++) {
     excess[s] += G[s][i].cap;
     Push(G[s][i]);
   }

   while (!Q.empty()) {
     int v = Q.front();
     Q.pop();
     active[v] = false;
     Discharge(v);
   }

   ll totflow = 0;
   for (int i = 0; i < G[s].size(); i++) totflow
       += G[s][i].flow;
   return totflow;
  }
};
```

3 Geometria

3.1 Circle Tangent

```
//Asumo que las circunferencias son diferentes
//Centro p y radio r1, centro q y radio r2
//Al menos un radio tiene que ser mayor que 0
//maneja el caso en que un circulo es punto
vector< Line > ComputeTgtInt(PT p, double r1, PT
    q, double r2){
   vector< Line > out;
   out.clear();
   if(equal(p, q)) return out;
   if(r1 > r2){
      swap(r1, r2);
      swap(p, q);
```

```
  }

  double distcent = sqrt(dist2(p, q));
  if(distcent + r1 - EPS < r2){
    return out;
  }

  vector< PT > ccinter = CircleCircleInter(p, q,
      r1, r2);
  Line lin;
  if(sz(ccinter) == 2) return out;
  else{
    if(sz(ccinter) == 1){
      PT tmp = ccinter[0];
      lin.fst = tmp;
      lin.snd = tmp + RotateCCW90(p - tmp);
      out.pb(lin);
    }
    else{
      PT O1;
      O1 = (p * r2 + q * r1) / (r1 + r2);
      double distc = sqrt(dist2(q, O1));
      double Alfa = asin(r2 / distc);
      PT tmp1 = (p - O1) * cos(Alfa);
      PT tmp2 = (q - O1) * cos(Alfa);
      lin.fst = O1 + RotateCCW(tmp1, Alfa);
      lin.snd = O1 + RotateCCW(tmp2, Alfa);
      out.pb(lin);
      lin.fst = O1 + RotateCCW(tmp1, 2 * PI -
          Alfa);
      lin.snd = O1 + RotateCCW(tmp2, 2 * PI -
          Alfa);
      out.pb(lin);
    }
    return out;
  }
}

//Asumo que las circunferencias son diferentes
//Centro p y radio r1, centro q y radio r2
//Siempre devuelve 0 o 2 tangentes
//En el caso de que solo haya 1, devuelve 2
    rectas que son
//las mismas salvo por EPS
vector< Line > ComputeTgtExt(PT p, double r1, PT
    q, double r2){
  vector< Line > out;
  out.clear();
  if(equal(p, q)) return out;
  if(r1 > r2){
    swap(r1, r2);
    swap(p, q);
  }

  double distcent = sqrt(dist2(p, q));
  if(distcent + r1 + EPS < r2){
    return out;
  }

  Line lin;
  PT tmp1, tmp2;
  if(abs(r1 - r2) < EPS){
    tmp1 = q - p;
    tmp1 = tmp1 * (r1 / distcent);
    tmp2 = p - q;
    tmp2 = tmp2 * (r2 / distcent);
    lin.fst = p + RotateCCW90(tmp1);
    lin.snd = q + RotateCW90(tmp2);
    out.pb(lin);
    lin.fst = p + RotateCW90(tmp1);
    lin.snd = q + RotateCCW90(tmp2);
    out.pb(lin);
  }
  else{
    double sinalfa = (r2 - r1) / distcent;
    sinalfa = max(sinalfa, 0.0);
    double Alfa = asin(sinalfa);
    PT O1;
    O1 = p + (p - q) * r1 / (r2 - r1);
    tmp1 = (p - O1) * cos(Alfa);
    tmp2 = (q - O1) * cos(Alfa);
    lin.fst = O1 + RotateCCW(tmp1, Alfa);
    lin.snd = O1 + RotateCCW(tmp2, Alfa);
    out.pb(lin);
    lin.fst = O1 + RotateCCW(tmp1, 2 * PI -
        Alfa);
    lin.snd = O1 + RotateCCW(tmp2, 2 * PI -
        Alfa);
    out.pb(lin);
  }
  return out;
}
```

3.2 Convex Hull - Doubles

```
//  INPUT:  a vector of input points, unordered.
//  OUTPUT: a vector of points in the convex
    hull,
//      counterclockwise, starting with bottom
    left

#define REMOVE_REDUNDANT

typedef double T;
const T EPS = 1e-7;
struct PT {
  T x, y;
  PT() {}
  PT(T x, T y) : x(x), y(y) {}
  bool operator<(const PT &rhs) const {
    return mp(y,x) < mp(rhs.y,rhs.x);
  }
  bool operator==(const PT &rhs) const {
    return mp(y,x) == mp(rhs.y,rhs.x);
  }
};

T cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
T area2(PT a, PT b, PT c) {
  return cross(a,b) + cross(b,c) + cross(c,a);
}

#ifdef REMOVE_REDUNDANT
bool between(const PT &a, const PT &b, const PT
    &c) {
  return (fabs(area2(a,b,c)) < EPS &&
          (a.x-b.x)*(c.x-b.x) <= 0 &&
          (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif
```

```
void ConvexHull(vector<PT> &pts) {
   sort(pts.begin(), pts.end());
   pts.erase(unique(pts.begin(), pts.end()),
       pts.end());
   vector<PT> up, dn;
   for (int i = 0; i < pts.size(); i++) {
      while (up.size() > 1 &&
         area2(up[up.size()-2], up.back(), pts[i])
             >= 0)
           up.pop_back();
      while (dn.size() > 1 &&
         area2(dn[dn.size()-2], dn.back(), pts[i])
             <= 0)
           dn.pop_back();
      up.push_back(pts[i]);
      dn.push_back(pts[i]);
   }
   pts = dn;
   for (int i = (int) up.size() - 2; i >= 1; i--)
      pts.push_back(up[i]);

#ifdef REMOVE_REDUNDANT
   if (pts.size() <= 2) return;
   dn.clear();
   dn.push_back(pts[0]);
   dn.push_back(pts[1]);
   for (int i = 2; i < pts.size(); i++) {
      if (between(dn[dn.size()-2],
          dn[dn.size()-1], pts[i]))
         dn.pop_back();
      dn.push_back(pts[i]);
   }
   if (dn.size() >= 3 && between(dn.back(),
       dn[0], dn[1])) {
      dn[0] = dn.back();
      dn.pop_back();
   }
  pts = dn;
#endif
}
```

3.3 Convex Hull - Enteros

```
ll dist(ii a, ii b){
   a.first -= b.first; a.second -= b.second;
   return ll(a.first)*a.first +
       ll(a.second)*a.second;
}

ll dot(ii o, ii a, ii b){
   return ll(a.first-o.first)*(b.first-o.first)
        +
            ll(a.second-o.second)*(b.second-o.second);
}

ll cross(ii o, ii a, ii b){
   return ll(a.first-o.first)*(b.second-o.second)
        - ll(a.second-o.second)*(b.first-o.first);
}

vector<ii> convexHull(vector<ii> &P){
   sort(P.begin(),P.end());
   P.erase(unique(P.begin(),P.end()),P.end());
   int n = sz(P);
   if(n == 1) return P;

   vector<ii> bot = {P[0]};
   FER(i,1,n){
      while(sz(bot) > 1 and
          cross(bot[sz(bot)-2],bot.back(),P[i])
          <= 0)
         bot.pop_back();
      bot.pb(P[i]);
   }
   bot.pop_back();

   vector<ii> up = {P[n-1]};
   for(int i = n-1; i >= 0; --i){
      while(sz(up) > 1 and
          cross(up[sz(up)-2],up.back(),P[i]) <= 0)
         up.pop_back();
      up.pb(P[i]);
   }
   up.pop_back();
   bot.insert(bot.end(),up.begin(),up.end());
   return bot;
}
```

3.4 Delaunay Triangulation

```
// Slow but simple Delaunay triangulation. Does
    not handle
// degenerate cases (from O'Rourke, Computational
    Geometry in C)
//
// Running time: O(n^4)
//
// INPUT:  x[] = x-coordinates
//         y[] = y-coordinates
//
// OUTPUT:  triples = a vector containing m
    triples of indices
//              corresponding to triangle vertices

#include<vector>
using namespace std;

typedef double T;

struct triple {
   int i, j, k;
   triple() {}
   triple(int i, int j, int k) : i(i), j(j), k(k)
       {}
};

vector<triple> delaunayTriangulation(vector<T>&
    x, vector<T>& y) {
   int n = x.size();
   vector<T> z(n);
   vector<triple> ret;

   for (int i = 0; i < n; i++)
      z[i] = x[i] * x[i] + y[i] * y[i];

   for (int i = 0; i < n-2; i++) {
```

```
      for (int j = i+1; j < n; j++) {
      for (int k = i+1; k < n; k++) {
         if (j == k) continue;
         double xn = (y[j]-y[i])*(z[k]-z[i]) -
             (y[k]-y[i])*(z[j]-z[i]);
         double yn = (x[k]-x[i])*(z[j]-z[i]) -
             (x[j]-x[i])*(z[k]-z[i]);
         double zn = (x[j]-x[i])*(y[k]-y[i]) -
             (x[k]-x[i])*(y[j]-y[i]);
         bool flag = zn < 0;
         for (int m = 0; flag && m < n; m++)
         flag = flag && ((x[m]-x[i])*xn +
               (y[m]-y[i])*yn +
               (z[m]-z[i])*zn <= 0);
         if (flag) ret.push_back(triple(i, j, k));
      }
      }
   }
   return ret;
}

int main(){
   T xs[]={0, 0, 1, 0.9};
   T ys[]={0, 1, 0, 0.9};
   vector<T> x(&xs[0], &xs[4]), y(&ys[0], &ys[4]);
   vector<triple> tri = delaunayTriangulation(x,
       y);

   //expected: 0 1 3
   //          0 3 2

   int i;
   for(i = 0; i < tri.size(); i++)
      printf("%d %d %d\n", tri[i].i, tri[i].j,
          tri[i].k);
   return 0;
}
```

3.5 Geometry Routines

```
// C++ routines for computational geometry

const ld INF = 1e100;
const ld EPS = 1e-12;
const ld PI = acos(-1);

struct PT {
  ld x, y;
  PT() {}
  PT(ld x, ld y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y) {}
  PT operator + (const PT &p) const { return
      PT(x+p.x, y+p.y); }
  PT operator - (const PT &p) const { return
      PT(x-p.x, y-p.y); }
  PT operator * (ld c)  const { return PT(x*c, y*c
      ); }
  PT operator / (ld c)  const { return PT(x/c, y/c
      ); }
};

ld dot(PT p, PT q)  { return p.x*q.x+p.y*q.y; }
ld norm(PT p) { return sqrt(dot(p,p)); }
ld dist2(PT p, PT q) { return dot(p-q,p-q); }
ld cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }

ld angle(PT p){
   ld res = acos(p.x / norm(p));
   if (p.y > 0) return res;
   else return 2*PI - res;
}

// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p) { return PT(-p.y,p.x); }
PT RotateCW90(PT p) { return PT(p.y,-p.x); }
PT RotateCCW(PT p, ld t) {
  return PT(p.x*cos(t)-p.y*sin(t),
      p.x*sin(t)+p.y*cos(t));
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a
    and b
PT ProjectPointSegment(PT a, PT b, PT c) {
  ld r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}

// distance from c to segment between a and b
ld DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b,
      c)));
}

// distance between point (x,y,z) and plane
    ax+by+cz=d
ld DistancePointPlane(ld x, ld y, ld z,
          ld a, ld b, ld c, ld d)
{
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// Whethes lines (a,b), (c,d) are
    parallel/collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}
bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d)
     && fabs(cross(a-b, a-c)) < EPS
     && fabs(cross(c-d, c-a)) < EPS;
}

// determine if line segment from a to b
    intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT 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 (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 &&
        dot(c-b, d-b) > 0) return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0)
      return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0)
      return false;
  return true;
}

// compute intersection of line passing through a
    and b
// with line passing through c and d, assuming
    that unique
// intersection exists; for segment intersection,
    check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT
    d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b,
      b+RotateCW90(a-b), c, c+RotateCW90(a-c));
}

// determine if point is in a possibly non-convex
    polygon
// (by William Randolph Franklin); returns 1 for
    strictly
// interior points, 0 for strictly exterior
    points, and 0 or 1
// for the remaining points. Note that it is
    possible to
// convert this into an *exact* test using
    integer arithmetic
// by taking care of the division appropriately
    (making sure
// to deal with signs properly) and then by
    writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
   int j = (i+1)%p.size();
   if (((p[i].y <= q.y && q.y < p[j].y) ||
     (p[j].y <= q.y && q.y < p[i].y)) &&
     q.x < p[i].x + (p[j].x - p[i].x) *(q.y -
         p[i].y) /
      (p[j].y - p[i].y))
     c = !c;
  }
  return c;
}

// determine if point is on the boundary of a
    polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
  for (int i = 0; i < p.size(); i++){
   if (dist2(ProjectPointSegment(p[i],
       p[(i+1)%p.size()], q), q) < EPS)
     return true;
  }

  return false;
}

// compute intersection of line through points a
    and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT
    c, ld r) {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  ld A = dot(b, b);
  ld B = dot(a, b);
  ld C = dot(a, a) - r*r;
  ld D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
   ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}

// compute intersection of circle centered at a
    with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleInter(PT a, PT b, ld r, ld
    R) {
  vector<PT> ret(0);
  ld d = sqrt(dist2(a, b));
  if (d > r+R + EPS || d+min(r, R) + EPS < max(r,
      R)) return ret;
  ld x = (d*d-R*R+r*r)/(2*d);
  ld y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
   ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}

// Area or centroid of a (possibly nonconvex)
    polygon,
// assuming the coordinates are listed in a
    clockwise or
// counterclockwise order. Note that the centroid
    is often
// known as the "center of gravity" or "center of
    mass".
ld ComputeSignedArea(const vector<PT> &p) {
  ld area = 0;
  for(int i = 0; i < p.size(); i++) {
   int j = (i+1) % p.size();
   area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}

ld ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}
```

```
PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  ld scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
   int j = (i+1) % p.size();
   c = c + (p[i]+p[j])*(p[i].x*p[j].y -
       p[j].x*p[i].y);
  }
  return c / scale;
}

// Whether or not a given (CW or CCW) polygon is
    simple
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
   for (int k = i+1; k < p.size(); k++) {
     int j = (i+1) % p.size();
     int l = (k+1) % p.size();
     if (i == l || j == k) continue;
     if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
      return false;
   }
  }
  return true;
}

// Computes the circumcenter of a Triangle PQR
PT circumcenter(PT p, PT q, PT r) {
   PT a = p-r, b = q-r;
   PT c = PT(dot(a, (p + r)) / 2, dot(b, (q + r))
       / 2);
   return PT(dot(c, RotateCW90(PT(a.y, b.y))),
      dot(PT(a.x, b.x), RotateCW90(c))) / dot(a,
          RotateCW90(b));
}
```

3.6 Minkowski Sum

```
//Calcula suma de Minkowski en O(n + m)
//A y B deben estar en sentido antihorario
inline bool compare(PT a, PT b){
   // mas abajo, mas a la izquierda
   if(a.y < b.y) return 1;
   if(a.y == b.y) return a.x < b.x;
   return 0;
}

vector<PT> minkow_sum(const vector<PT>& a, const
    vector<PT>& b){
   vector< PT > out;
   out.clear();
   int lena = int(a.size());
   int lenb = int(b.size());
   int i = 0, j = 0;
   for(int q = 0; q < lena; ++q) if(compare(a[q],
       a[i])) i = q;
   for(int q = 0; q < lenb; ++q) if(compare(b[q],
       b[j])) j = q;
   ll pr;
   int nxti, nxtj;
   do{
      out.pb(a[i] + b[j]);
      nxti = (i + 1) % lena;
      nxtj = (j + 1) % lenb;
      pr = cross(a[nxti] - a[i], b[nxtj] - b[j]);
      if(pr > 0) i = nxti;
      else if(pr < 0) j = nxtj;
      else i = nxti, j = nxtj; // paralelas, subo
          en ambas
   }while((a[i] + b[j]) != out[0]);
   return out;
}
```

3.7 Polygon

```
int sgn(double x){return x<-EPS?-1:x>EPS;}
struct pol {
   int n; vector<PT> p;
   pol(){}
   pol(vector<PT> _p){p=_p;n=p.size();}
   double area(){
      double r=0.;
      FER(i,0,n) r+=p[i]%p[(i+1)%n];
      return abs(r)/2; // negative if CW, positive
          if CCW
   }
   PT centroid(){ // (barycenter)
      PT r(0,0);double t=0;
      FER(i,0,n){
         r=r+(p[i]+p[(i+1)%n])*(p[i]%p[(i+1)%n]);
         t+=p[i]%p[(i+1)%n];
      }
      return r/t/3;
   }
   bool has(PT q){ // O(n)
      FER(i,0,n)if(ln(p[i],p[(i+1)%n]).seghas(q))return
          true;
      int cnt=0;
      FER(i,0,n){
         int j=(i+1)%n;
         int k=sgn((q-p[j])%(p[i]-p[j]));
         int u=sgn(p[i].y-q.y),v=sgn(p[j].y-q.y);
         if(k>0&&u<0&&v>=0)cnt++;
         if(k<0&&v<0&&u>=0)cnt--;
      }
      return cnt!=0;
   }
   void normalize(){ // (call before haslog,
       remove collinear first)
      if(p[2].left(p[0],p[1]))
         reverse(p.begin(),p.end());
      int pi=min_element(p.begin(),
      p.end())-p.begin();
      vector<PT> s(n);
      FER(i,0,n)s[i]=p[(pi+i)%n];
      p.swap(s);
   }
   bool haslog(PT q){ // O(log(n)) only CONVEX.
       Call normalize first
      if(q.left(p[0],p[1])||q.left(p.back(),p[0]))return
          false;
      int a=1,b=p.size()-1; // returns true if
          point on boundary
      while(b-a>1){      // (change sign of EPS in
          left
         int c=(a+b)/2;    //  to return false in
             such case)
```

```
      if(!q.left(p[0],p[c]))a=c;
      else b=c;
   }
   return !q.left(p[a],p[a+1]);
}
PT farthest(PT v){ // O(log(n)) only CONVEX
   if(n<10){
      int k=0;
      FER(i,1,n)if(v*(p[i]-p[k])>EPS)k=i;
      return p[k];
   }
   if(n==SZ(p))p.pb(p[0]);
   PT a=p[1]-p[0];
   int s=0,e=n,ua=v*a>EPS;
   if(!ua&&v*(p[n-1]-p[0])<=EPS)return p[0];
   while(1){
      int m=(s+e)/2;PT c=p[m+1]-p[m];
      int uc=v*c>EPS;
      if(!uc&&v*(p[m-1]-p[m])<=EPS)return p[m];
      if(ua&&(!uc||v*(p[s]-p[m])>EPS))e=m;
      else if(ua||uc||v*(p[s]-p[m])>=-EPS)
         s=m,a=c,ua=uc;
      else e=m;
      assert(e>s+1);
   }
}
pol cut(ln l){ // cut CONVEX polygon by line l
   vector<PT> q; // returns part at left of l.pq
   FER(i,0,n){
      int d0=sgn(l.pq%(p[i]-l.p));
      int d1=sgn(l.pq%(p[(i+1)%n]-l.p));
      if(d0>=0)q.pb(p[i]);
      ln m(p[i],p[(i+1)%n]);
      if(d0*d1<0&&!(l/m))q.pb(l^m);
   }
   return pol(q);
}
double intercircle(circle c){ // area of
    intersection with circle
   double r=0.;
   FER(i,0,n){
      int j=(i+1)%n;double
          w=c.intertriangle(p[i],p[j]);
      if((p[j]-c.o)%(p[i]-c.o)>0)r+=w;
      else r-=w;
   }
   return abs(r);
}
double callipers(){ // square distance of most
    distant points
   double r=0;  // prereq: convex, ccw, NO
       COLLINEAR POINTS
   for(int i=0,j=n<2?0:1;i<j;++i){
      for(;;j=(j+1)%n){
         r=max(r,(p[i]-p[j]).norm2());
         if((p[(i+1)%n]-p[i])%
            (p[(j+1)%n]-p[j])<=EPS)break;
      }
   }
   return r;
}
};
// Dynamic convex hull trick
vector<pol> w;
void add(PT q){ // add(q), O(log^2(n))
   vector<PT> p={q};
   while(!w.empty()&&SZ(w.back().p)<2*SZ(p)){
      for(PT v:w.back().p)p.pb(v);
      w.pop_back();
   }
   w.pb(pol(chull(p)));
}
ll query(PT v){ // max(q*v:q in w), O(log^2(n))
   ll r=-INF;
   for(auto& p:w)r=max(r,p.farthest(v)*v);
   return r;
}
```

3.8 Rotating Callipers

```
/* Maximum Distance of a pair of points
in a Range Using Rotatin Calipers and Segment Tree
Note that to build the convex hull in a range
always there at least 2 points less for that
we can build the convexhull, for queries
we just construct what we need */

struct PT{
   ll x, y;
   PT(){}
   PT(ll x, ll y): x(x), y(y){}
};

PT points[1<<18];
ll tt[1<<18];


ll GetDist(PT &a, PT &b){
   return sqr(a.x-b.x)+sqr(a.y-b.y);
}

inline ll GetOri(ll id1, ll id2, ll id3){
   ll r=(points[id2].y-points[id1].y)*
      (points[id3].x-points[id1].x);
   r-=(points[id2].x-points[id1].x)*
      (points[id3].y-points[id1].y);
   return (r==0)? 0: ((r>0)? 1: -1);
}

inline bool compare(ll id1, ll id2){
   if(points[id1].x<points[id2].x) return true;
   if(points[id1].x==points[id2].x and
   points[id1].y<points[id2].y) return true;
   return false;
}



struct ConvexHull{
   vi uh, lh;
   ll count=0;
   inline void set(ll i){
      uh.resize(1, i);
      lh.resize(1, i);
      count=1;
   }
};
// Dinamic Segment Tree

struct ST{
```

```
vector<ConvexHull> t;
ll n;

inline ll Join(vi &v1, vi &v2, ll *tt){
   ll i=0, j=0, k=0, n1=sz(v1), n2=sz(v2);
   while(i<n1 && j<n2){
      if(compare(v1[i], v2[j])) tt[k++]=v1[i++];
      else tt[k++]=v2[j++];
   }
   while(i<n1) tt[k++]=v1[i++];
   while(j<n2) tt[k++]=v2[j++];
   return k;
}

inline ConvexHull Op(ConvexHull &t1,
    ConvexHull &t2){
   if(t1.count==0) return t2;
   if(t2.count==0) return t1;
   ConvexHull ty;
   ll count=Join(t1.uh, t2.uh, tt), l=0, r=0;
   FER(i,0,count){
      while(l>1 and GetOri(tt[l-2], tt[l-1],
          tt[i])<=0) --l;
      tt[l++]=tt[i];
   }
   ty.uh.insert(ty.uh.end(), tt, tt+l);
   count=Join(t1.lh, t2.lh, tt);
   FER(i,0,count){
      while(r>1 and GetOri(tt[r-2], tt[r-1],
          tt[i])>=0) --r;
      tt[r++]=tt[i];
   }
   ty.lh.insert(ty.lh.end(), tt, tt+r);
   ty.count=sz(ty.uh)+sz(ty.lh)-2;
   if(ty.count==0) ty.count=1;
   return ty;
}

ST(ll n): n(n), t(2*n){
   FER(i,0,n) t[i+n].set(i);
   IFR(i, n-1, 1) t[i]=Op(t[i<<1], t[i<<1|1]);
}

inline ConvexHull Query(ll l, ll r){
   ConvexHull ans;
   for(l+=n, r+=n; l<r; l>>=1, r>>=1){
      if(l&1) ans=Op(ans, t[l++]);
      if(r&1) ans=Op(t[--r], ans);
   }
   return ans;
}

inline ll que(ll l, ll r){
   if(r-l<=1) return GetDist(points[l],
       points[r]);
   ConvexHull cur=Query(l, r+1);
   ll lr=sz(cur.uh), rr=sz(cur.lh), i=0, j=rr-1;
   ll ans=0, curdist, val;
   while(i<lr-1 or j>0){
      PT &left=points[cur.uh[i]];
      PT &right=points[cur.lh[j]];
      curdist=GetDist(left, right);
      // first brute force
      if(curdist>ans) ans=curdist;
      if(i==lr-1) --j;
      else if(j==0) i++;
      else{
         // rotating calipers
         PT& t1=points[cur.uh[i+1]];
         PT& t2=points[cur.lh[j-1]];
         val=(t1.y-left.y)*(right.x-t2.x);
         val-=(t1.x-left.x)*(right.y-t2.y);
         if(val>0) i++;
         else --j;
      }
   }
   return ans;
}
};


int main(){
   fastio;
   ll n; cin>>n;
   FER(i,0,n) cin>>points[i].x>>points[i].y;
   ST st(n);
   ll q, l, r, froz; cin>>q;
   FER(i,0,q){
      cin>>l>>r; l--, r--;
      froz=st.que(l, r);
      cout<<froz<<endl;
   }
   return 0;
}
```

3.9 Voronoi Diagram

```
//Voronoi diagrams: O(n^2 logn)
//Convex hull: O(nlogn)
// Requires PT, ComputeLineIntersection, dot,
    cross
#define MAXN 1024
double INF = 1e10;
double EPS = 1e-7;

typedef struct {
  int id;
  double x;
  double y;
  double ang;
} chp;

int n;
double x[MAXN], y[MAXN]; // Input points
chp inv[2*MAXN]; // Points after inversion for
    Hull
int vors;
int vor[MAXN]; // Set of points in convex hull;
          //starts at lefmost; last same as
              first!!
PT ans[MAXN][2];

int chpcmp(const void *aa, const void *bb) {
  double a = ((chp *)aa)->ang;
  double b = ((chp *)bb)->ang;
  if (a<b) return -1;
  else if (a>b) return 1;
  else return 0;
}
```

```
int orient(chp *a, chp *b, chp *c) {
  double s = a->x*(b->y-c->y) + b->x*(c->y-a->y) +
      c->x*(a->y-b->y);
  if (s>0) return 1;
  else if (s<0) return -1;
  else if (a->ang==b->ang && a->ang==c->ang)
      return -1;
  else return 0;
}

int convexHull(int n, chp *pt, int *ans) {
  int i, j, st, anses=0;

  qsort(pt, n, sizeof(chp), chpcmp);
  for (i=0; i<n; i++) pt[n+i] = pt[i];
  st = 0;
  for (i=1; i<n; i++) {
      if (pt[i].x<pt[st].x ||
         (pt[i].x==pt[st].x && pt[i].y<pt[st].y))
             st = i;
  }
  ans[anses++] = st;
  for (i=st+1; i<=st+n; i++) {
   for (j=anses-1; j; j--) {
     if (orient(pt+ans[j-1], pt+ans[j], pt+i)>=0)
         break;
     // Should change the above to strictly
         greater,
     // if you don't want redundan points
     // If you want them, you might put an epsilon
         in orient
   }
   ans[j+1] = i;
   anses = j+2;
  }
  for (i=0; i<anses; i++) ans[i] = pt[ans[i]].id;
  return anses;
}

int main(void) {
  int i, j, jj;
  double tmp;

  scanf("%d", &n);
  for (i=0; i<n; i++) scanf("%lf %lf", &x[i],
      &y[i]);
  for (i=0; i<n; i++) {
   x[n] = 2*(-INF)-x[i]; y[n] = y[i];
   x[n+1] = x[i]; y[n+1] = 2*INF-y[i];
   x[n+2] = 2*INF-x[i]; y[n+2] = y[i];
   x[n+3] = x[i]; y[n+3] = 2*(-INF)-y[i];
   for (j=0; j<n+4; j++) if (j!=i) {
     jj = j - (j>i);
     inv[jj].id = j;
     tmp = (x[j]-x[i])*(x[j]-x[i]) +
         (y[j]-y[i])*(y[j]-y[i]);
     inv[jj].x = (x[j]-x[i])/tmp;
     inv[jj].y = (y[j]-y[i])/tmp;
     inv[jj].ang = atan2(inv[jj].y, inv[jj].x);
   }
   vors = convexHull(n+3, inv, vor);
   // Build bisectors
   for (j=0; j<vors; j++) {
     ans[j][0].x = (x[i]+x[vor[j]])/2;
     ans[j][0].y = (y[i]+y[vor[j]])/2;
     ans[j][1].x = ans[j][0].x - (y[vor[j]]-y[i]);
     ans[j][1].y = ans[j][0].y + (x[vor[j]]-x[i]);
   }
   printf("Around (%lf, %lf)\n", x[i], y[i]);
   // List all intersections of the bisectors
   for (j=1; j<vors; j++) {
     PT vv;
     vv = ComputeLineIntersection(ans[j-1][0],
         ans[j-1][1],
              ans[j][0], ans[j][1]);
     printf("%lf, %lf\n", vv.x, vv.y);
   }
   printf("\n");
  }
  return 0;
}
```

4 Grafos

4.1 2-SAT

```
/* 2-Satisfiability
   Autor: Phibrain
   Instrucciones de uso:
      assig[i]: 0 pushear, 1: no pushear
      nodes para el nodo n >> true: 2*n, false:
          2*n + 1
*/

struct TWOSAT{
   vi adj[N], adjt[N];
   ll n;
   ll comp[N], vis[N], assig[N];
   vi eulerian;
   inline void dfs1(ll u){
      vis[u]=1;
      for(auto xd: adj[u]) if(vis[xd]==0) dfs1(xd);
      eulerian.pb(u);
   }
   inline void dfs2(ll u, ll rt){
      comp[u]=rt;
      for(auto xd: adjt[u]) if(comp[xd]==-1)
          dfs2(xd, rt);
   }
   inline ll solve(){
      fill(vis, 0), fill(comp, -1), fill(assig, 0);
      ll pos=0;
      FER(i,0,n) for(auto xd: adj[i])
          adjt[xd].pb(i);
      FER(i,0,n) if(vis[i]==0) dfs1(i);
      reverse(all(eulerian));
      for(auto xd: eulerian) if(comp[xd]==-1)
          dfs2(xd, pos++);
      FER(i,0,n/2) assig[i]=1;
      for(ll i=0; i<n; i+=2){
         if(comp[i]==comp[i+1]) return 0;
         assig[i/2]=(comp[i]>comp[i+1]);
      }
      return 1;
   }
};
```

4.2 BiconnectedComponents

```
vector<ii> st;
int tc,n,m,par[N];
int cnt,num[N],low[N];
vector<int> adj[N];
vector< vector<ii> > bcc;

void tarjan(int u, bool root = 0){
   num[u] = low[u] = cnt++;
   int child = 0;
   for(int v : adj[u]) if(v != par[u]){
      if(num[v] == -1){
         child++, par[v] = u;
         st.push_back(ii(u,v));
         tarjan(v);
         low[u] = min(low[u],low[v]);
         if((root && child > 1) || (!root &&
             num[u] <= low[v])){
            vector<ii> tmp;
            while(st.back() != ii(u,v))
                tmp.pb(st.back()), st.pop_back();
            tmp.pb(st.back()), st.pop_back();
            bcc.pb(tmp);
         }
      }else if(num[v] < num[u]){
         low[u] = min(low[u],num[v]);
         st.pb(ii(u,v));
      }
   }
}

int main(){

   int TC,a,b;
   cin >> TC;
   for(tc = 1; tc <= TC; ++tc){
      cin >> n >> m;
      bcc.clear();
      for(int i = 0; i <= n; ++i)
         adj[i].clear();
      for(int i = 0; i < m; ++i){
         cin >> a >> b;
         adj[a].pb(b);
         adj[b].pb(a);
      }
      cnt = 0;
      for(int i = 0; i <= n; ++i)
         par[i] = num[i] = low[i] = -1;

      for(int i = 1; i <= n; ++i) if(num[i] == -1){
         tarjan(i,1);
         if(sz(st)) bcc.pb(st);
         st.clear();
      }
   }

   return 0;
}
```

4.3 Bridges and Articulation Points

```
/* Bridges and Articulation Points
   Autor: Froz
   Instrucciones de uso:
      fill(id, -1) antes de usarlo
      En bridge se guardaran los puentes
      art[i] == True si el nodo i es articulacion
*/

const int N = 1e5 + 2;

int low[N], id[N], parent[N];
vector<int> adj[N];
int curr_id =0;
int root, rootchild;

bool art[N];
vector<ii> bridges;

void dfs(int u) {
   low[u] = id[u] = curr_id++;
   for(int &v: adj[u]){
      if (id[v] == -1){
         parent[v] = u;
         if (u == root) rootchild++;
         dfs(v);
         if (low[v] >= id[u]) art[u] = true;
         if (low[v] > id[u]){
            bridges.pb({u, v});
         }
         low[u] = min(low[u], low[v]);
      }
      else if (v != parent[u]) low[u] =
          min(low[u], id[v]);
   }
}

//inside int main()
fill(id, -1);
FER(i, 0, n){
   if (id[i] == -1) {
      root = i; rootchild = 0; dfs(i);
      art[root] = (rootchild > 1);
   }
}
```

4.4 Chu Liu

```
// O(n * m) MST for directed graphs

typedef ll tw;const tw INF=1LL<<60;
struct edge {int src,dst;tw w;};
struct ChuLiu {
   int n,r;tw cost;bool found;
   vector<int> no,pr,mark;
   vector<vector<int> > comp,nx;
   vector<tw> mcost;
   vector<vector<edge> > h;
   ChuLiu(int n):n(n),h(n){}
   void add_edge(int x, int y, tw
       w){h[y].pb((edge){x,y,w});}
   void visit(int v, int s){
      if(mark[v]){
         vector<int> temp=no;found=true;
```

```
      do {
         cost+=mcost[v];v=pr[v];
         if(v!=s)while(comp[v].size()>0){
            no[comp[v].back()]=s;
            comp[s].pb(comp[v].back());
            comp[v].pop_back();
         }
      }while(v!=s);
      for(int j:comp[s])if(j!=r)for(edge&
          e:h[j])
         if(no[e.src]!=s)e.w-=mcost[temp[j]];
   }
   mark[v]=true;
   for(int
       i:nx[v])if(no[i]!=no[v]&&pr[no[i]]==v)
      if(!mark[no[i]]||i==s)
         visit(i,s);
}
tw doit(int _r){ // r: root (O(nm))
   r=_r;
   no.resize(n);comp.clear();comp.resize(n);
   FER(x,0,n)comp[x].pb(no[x]=x);
   for(cost=0;;){
      pr.clear();pr.resize(n,-1);
      mcost=vector<tw>(n,INF);
      FER(j,0,n)if(j!=r)for(edge e:h[j])
         if(no[e.src]!=no[j]&&e.w<mcost[no[j]])
            mcost[no[j]]=e.w,pr[no[j]]=no[e.src];
      nx.clear();nx.resize(n);
      FER(x,0,n)if(pr[x]>=0)nx[pr[x]].pb(x);
      bool stop=true;
      mark.clear();mark.resize(n);
      FER(x,0,n)if(x!=r&&!mark[x]&&
         !comp[x].empty()){
         found=false;visit(x,x);
         if(found)stop=false;
      }
      if(stop){
         FER(x,0,n)if(pr[x]>=0)cost+=mcost[x];
         return cost;
      }
   }
}
};
```

4.5 Erdos Gallai

```
/*  Erdos Gallai Theorem
   Autor: Froz
   Instrucciones de uso:
      Dado un vector de N nodos con grado d[i],
          establecer si
      es posible tener un grafo con esa
          construccion.
*/

bool erdosgallai(vector<int> d) {
   sort(all(d), greater<int>());
   vector<ll> pd(sz(d));
   int n = sz(d), p = n-1;
   FER(i, 0, n) pd[i] = d[i] + (i > 0 ? pd[i-1] :
       0);
   FER(k, 1, n + 1){
      while(p >= 0 && d[p] < k) p--;
      ll sum;
      if (p >= k-1) sum = (p-k+1)*1ll*k + pd[n-1]
          - pd[p];
      else sum = pd[n-1] - pd[k-1];
      if (pd[k-1] > k*(k-1LL) + sum) return false;
   }
   return (pd[n-1] % 2 == 0);
}
```

4.6 Eulerian Path

```
// Finds Eulerian Path (visits every edge exactly
    once)
// CYCLE exists iff all edges even degree, all
    edges in
// same connected component.
// PATH exists iff cycle exists and once edge
    removed
// [ Hamiltonian (all vertices) is NP complete ]
struct Edge;
typedef list<Edge>::iterator iter;
struct Edge
{
   int next_vertex;
   iter reverse_edge;
   Edge(int next_vertex)
       :next_vertex(next_vertex) { }
};
const int max_vertices = ;
int num_vertices;
list<Edge> adj[max_vertices];  // adjacency list
vector<int> path;

void find_path(int v)
{
   while(adj[v].size() > 0)
   {
      int vn = adj[v].front().next_vertex;
      adj[vn].erase(adj[v].front().reverse_edge);
      adj[v].pop_front();
      find_path(vn);
   }
   path.push_back(v);
}
void add_edge(int a, int b)
{
   adj[a].push_front(Edge(b));
   iter ita = adj[a].begin();
   adj[b].push_front(Edge(a));
   iter itb = adj[b].begin();
   ita->reverse_edge = itb;
   itb->reverse_edge = ita;
}
```

4.7 Graph Utils

```
// Lexicographically smallest toposort

vector<int> g[MAXN];int n;
vector<int> tsort(){
```

```
   vector<int> r;priority_queue<int> q;
   vector<int> d(2*n,0);
   FER(i,0,n)FER(j,0,g[i].size())d[g[i][j]]++;
   FER(i,0,n)if(!d[i])q.push(-i);
   while(!q.empty()){
      int x=-q.top();q.pop();r.pb(x);
      FER(i,0,g[x].size()){
         d[g[x][i]]--;
         if(!d[g[x][i]])q.push(-g[x][i]);
      }
   }
   return r; // if not DAG it will have less than
       n elements
}

//Floyd - Warshall and Bellman Ford

// g[i][j]: weight of edge (i, j) or INF if
    there's no edge
// g[i][i]=0
ll g[MAXN][MAXN];int n;
void floyd(){ // O(n^3) . Replaces g with min
    distances
   FER(k,0,n)FER(i,0,n)if(g[i][k]<INF)
      FER(j,0,n)if(g[k][j]<INF)
         g[i][j]=min(g[i][j],g[i][k]+g[k][j]);
}
bool inNegCycle(int v){return g[v][v]<0;}
bool hasNegCycle(int a, int b){ // true iff
    there's neg cycle in between
   FER(i,0,n)if(g[a][i]<INF&&
      g[i][b]<INF&&g[i][i]<0)return true;
   return false;
}
```

4.8 Kruskal

```
/* Kruskal para Minimum Spanning Tree
   Autor: Phibrain
*/

struct Kruskal{
  ll id[N], n, ans;
  vector<pair<ll,ii>> v;
  vii adj[N];
  inline void build(){
   FER(i,0,n) id[i]=i;
  }
  inline ll find(ll x){
   while(id[x]!=x) id[x]=id[id[x]], x=id[x];
   return x;
  }
  inline void unir(ll x, ll y){
   ll p=find(x), q=find(y);
   id[p]=id[q];
  }
  inline void clear(){
   fill(id,0);
   ans=0;
  }
  inline void go(ll l, ll r){
   FER(i,l,r){
    ll w=v[i].tm1, x=v[i].tm2, y=v[i].tm3;
    if(find(x)!=find(y)) ans+=w,
        unir(x,y),adj[x].pb({y,w}),
        adj[y].pb({x,w});
   }
  }
};
```

4.9 Maximal Cliques

```
// Bron-Kerbosch algorithm for finding all the
// maximal cliques of a graph in O(3^(n/3))
// 3 ^ 13 = 1.6e6

// Call them using clique(0, (1LL << n) - 1, 0)
// n vertexs
ll adj[65];
// This algorithm finds all the maximal cliques
    containing an edge
// The cliques are found explicitly (the vertex
    of the cliques)
void clique(ll r, ll p, ll x) {
   if (p == 0 && x == 0) {
      /* r is a maximal clique */
      /* Every 1 in r is a vertex of the clique
      Then, __builtin_popcountll(r) is the size of
          the clique*/
      return;
      }
   int pivot = -1;
   int menor = INF;
   for (int i = 0; i < n; i++) {
      if ( ((1LL << i) & p) || ((1LL << i) & x) ) {
         int x = __builtin_popcountll(p &
             (~(adj[i])));
         if (x < menor) {
            pivot = i;
            menor = x;
         }
      }
   }
   for (int i = 0; i < n; i++) {
      if ((1LL << i) & p) {
         if (pivot != -1 && adj[pivot] & (1LL <<
             i)) continue;
         clique(r | (1LL << i), p & adj[i], x &
             adj[i]);
         p = p ^ (1LL << i);
         x = x | (1LL << i);
      }
   }
}

// This one has the same idea, but is faster
// However, it only finds the size of the cliques
void clique2(int r, ll p, ll x){
   if(p == 0 && x == 0){
      // r is the size of the clique
   }
   if(p == 0) return;
   int u = __builtin__ctzll(p | x);
   ll c = p & ~ adj[u];
   while(c){
      int v = __builint_ctzll(c); //Number of
          trailing zeros
      clique(r + 1, p & adj[v], x & adj[v]);
```

```
      p ^= (1LL << v);
      x |= (1LL << v);
      c ^= (1LL << v);
   }
}
```

4.10 Planar Dual Graph

```
vector<int> g[MAXN];int n; // input graph (must
    be connected)
vector<int> gd[MAXN];int nd; // output graph
vector<int> nodes[MAXN]; // nodes delimiting
    region (in CW order)
map<pair<int,int>,int> ps,es;
void get_dual(vector<PT> p){ // p: points
    corresponding to nodes
   ps.clear();es.clear();
   FER(x,0,n){
      Cmp pc(p[x]); // (radial order of points)
      auto comp=[&](int a, int b){return
          pc(p[a],p[b]);};
      sort(g[x].begin(),g[x].end(),comp);
      FER(i,0,g[x].size())ps[mp(x,g[x][i])]=i;
   }
   nd=0;
   FER(xx,0,n)for(auto
       yy:g[xx])if(!es.count(mp(xx,yy))){
      int
          x=xx,y=yy;gd[nd].clear();nodes[nd].clear();
      while(!es.count(mp(x,y))){
         es[mp(x,y)]=nd;nodes[nd].pb(y);
         int z=g[y][(ps[mp(y,x)]+1)%g[y].size()];
         x=y;y=z;
      }
      nd++;
   }
   for(auto p:es){
      pair<int,int> q=mp(p.fst.snd,p.fst.fst);
      assert(es.count(q));
      if(es[q]!=p.snd)gd[p.snd].pb(es[q]);
   }
   FER(i,0,nd){
      sort(gd[i].begin(),gd[i].end());
      gd[i].erase(unique(gd[i].begin(),
      gd[i].end()),gd[i].end());
   }
}
```

4.11 Tarjan Strongly Connected Components

```
/* Tarjan's SCC
   Autor: Halim
   Instrucciones de uso:
      Inicializar d[N] en -1!
      No olvidarse de limpiar variables
      Es como un dfs, lanzar un for y correr si
          nodo actual
      no esta visitado.
*/

vector<int> g[N];
int d[N], low[N], scc[N];
bool stacked[N];
stack<int> s;
int ticks, current_scc;

void tarjan(int u){
  d[u] = low[u] = ticks++;
  s.push(u);
  stacked[u] = true;
  const vector<int> &out = g[u];
  for (int k=0, m=out.size(); k<m; ++k){
   const int &v = out[k];
   if (d[v] == -1){
     tarjan(v);
     low[u] = min(low[u], low[v]);
   }else if (stacked[v]){
     low[u] = min(low[u], low[v]);
   }
  }
  if (d[u] == low[u]){
   int v;
   do{
     v = s.top();
     s.pop();
     stacked[v] = false;
     scc[v] = current_scc;
   }while (u != v);
   current_scc++;
  }
}
```

4.12 TreeIsomorphism

```
vector< pair< vector<int>,int > >
    Radixsort(vector< vector<int> > &vec){
       int lmax = 0, m = 0;
       for(vector<int> &v : vec){
              lmax = max(lmax,sz(v));
              for(int &x : v) m = max(m,x+1);
       }
       deque<int> dq;
       vector< queue<int> > q(m);
       vector< vector<int> > hold(lmax);
       for(int i = 0; i < sz(vec); ++i){
              hold[sz(vec[i])-1].pb(i);
       }
       for(int i = lmax-1; i >= 0; --i){
              for(int x : hold[i])
                  dq.push_front(x);
              while(sz(dq)){
                     int x = dq.front();
                     dq.pop_front();
                     q[vec[x][i]].push(x);
              }
              for(int j = 0; j < m; ++j){
                     while(sz(q[j])){
                            dq.push_back(q[j].front());
                            q[j].pop();
                     }
              }
       }
```

```
        vector< pair< vector<int>,int > >
            ans(sz(vec));
        for(int i = 0; i < sz(vec); ++i){
                int x = dq.front(); dq.pop_front();
                ans[i] = mp(vec[x],x);
        }

        return ans;
}

int vist[N];
int lev1,lev2,id1[N],id2[N],pai1[N],pai2[N];
vector<int> noleaf1[N],noleaf2[N];
vector<int> leaf1[N],leaf2[N],tup1[N],tup2[N];

void dfs1(Tree &tree, int u, int p, int dep = 0){
        pai1[u] = p;
        lev1 = max(lev1,dep);
        vist[u] = 1;
        int tam = 0;
        for(int v : tree[u]) if(!vist[v]){
                dfs1(tree,v,u,dep+1);
                tam++;
        }
        if(tam >= 1) noleaf1[dep].pb(u);
        else leaf1[dep].pb(u), id1[u] = 0;
}

void dfs2(Tree &tree, int u, int p, int dep = 0){
        pai2[u] = p;
        lev2 = max(lev2,dep);
        vist[u] = 1;
        int tam = 0;
        for(int v : tree[u]) if(!vist[v]){
                dfs2(tree,v,u,dep+1);
                tam++;
        }
        if(tam >= 1) noleaf2[dep].pb(u);
        else leaf2[dep].pb(u), id2[u] = 0;
}

bool isomorphic(Tree &t1, int r1, Tree &t2, int
    r2, int ban){
        for(int i = 0; i < N; ++i){
                vist[i] = 0;
                noleaf1[i].clear(),
                    noleaf2[i].clear();
                leaf1[i].clear(), leaf2[i].clear();
                tup1[i].clear(), tup2[i].clear();
        }
        vist[ban] = 1;
        lev1 = lev2 = 0;
        dfs1(t1,r1,r1);
        FER(i,0,N) vist[i] = 0; vist[ban] = 1;
        dfs2(t2,r2,r2);
        if(lev1 != lev2) return false;
        vector<int> l1,l2;
        for(int x : leaf1[lev1]) l1.pb(x);
        for(int x : leaf2[lev2]) l2.pb(x);
        if(sz(l1) != sz(l2)) return false;

        for(int i = lev1-1; i >= 0; --i){
                vector< vector<int> > v1,v2;
                for(int u : l1){
                        tup1[pai1[u]].pb(id1[u]);
                }
                for(int u : l2){
                        tup2[pai2[u]].pb(id2[u]);
                }
                for(int u : noleaf1[i])
                        v1.pb(tup1[u]);
                for(int u : noleaf2[i])
                        v2.pb(tup2[u]);
                if(sz(v1) != sz(v2)) return false;

                vector< pair< vector<int>,int > >
                    res1, res2;
                res1 = Radixsort(v1);
                res2 = Radixsort(v2);
                int cnt = 1;
                l1.clear(); l2.clear();

                for(int x : leaf1[i]) l1.pb(x);
                for(int x : leaf2[i]) l2.pb(x);
                for(int j = 0; j < sz(res1); ++j){
                        if(res1[j].first !=
                            res2[j].first) return
                            false;
                        if(j > 0 and res1[j].first
                            != res1[j-1].first)
                            cnt++;
                        l1.pb(noleaf1[i][res1[j].second]);
                        l2.pb(noleaf2[i][res2[j].second]);
                        id1[noleaf1[i][res1[j].second]]
                            = cnt;
                        id2[noleaf2[i][res2[j].second]]
                            = cnt;
                }
        }
        return true;
}
```

4.13 Weighted Eulerian Path

```
/*
   Dado un grafo dirigido con pesos y 2 nodos
       "a", "b"
   encuentra todos los caminos de "a" hacia "b"

   la idea clave es vacear todos los ciclos
       recursivamente
   mientras se hace el dfs de "a" hacia "b"
   ... El grafo esta guardado en graph[][] y
       adj1[]

*/


ll graph[N1][N1];
vii adj1[N1];

vi ands, cycles[N1], path[N1], pru[N1], cam[N1];
vector<tri> tnt;
ll vis[N1], mark[N1], visC[N1], enc[N1];

inline void GoBack(ll u, ll v, ll &flag, vi &vec){
   if(flag) return;
   enc[u]++;
   vec.pb(u);
   for(auto xd: adj1[u]){
```

```
      if(flag) return;
      if(!graph[u][xd.ff]) continue;
      if(xd.ff==v){
         flag++;
         if(flag==1) vec.pb(v);
         return;
      }
      if(!enc[xd.ff]){
         GoBack(xd.ff, v, flag, vec);
         if(!flag) vec.pop_back();
      }
   }
}


inline void fillout(ll u){
   vi cur;
   ll ans, flag;
   while(!visC[u]){
      flag=0;
      cur.clear();
      fill(enc, 0);
      GoBack(u, u, flag, cur);
      if(flag){
         ans=INF;
         FER(i, 1, sz(cur)) ans=min(ans,
             graph[cur[i-1]][cur[i]]);
         FER(i, 1, sz(cur))
             graph[cur[i-1]][cur[i]]-=ans;
         FER(j, 0, ans) FER(i, 0, sz(cur)-1)
             cycles[u].pb(cur[i]);
      }
      else{
         visC[u]++;
         break;
      }
   }
}


inline void dfs(ll u, ll v){
   if(mark[u]) return;
   mark[u]++;
   if(!visC[u]) fillout(u);
   for(auto xd: cycles[u]) {
      if(!mark[xd]) dfs(xd, v);
      path[v].pb(xd);
   }
}


inline void findallpaths(ll u, ll v, ll &flag){
   if(flag) return;
   vis[u]++;
   if(!mark[u]){
      dfs(u, u);
      path[u].pb(u);
      for(auto xd: path[u]) ands.pb(xd);
   }
   else ands.pb(u);
   for(auto xd: adj1[u]){
      if(flag) return;
      if(graph[u][xd.ff]==0) continue;
      if(xd.ff==v){
         flag++;
         if(flag==1) ands.pb(v), graph[u][xd.ff]--;
         return;
      }
      if(!vis[xd.ff]){
         graph[u][xd.ff]--;
         findallpaths(xd.ff, v, flag);
      }
   }
}

int main(){
   fastio;
   ll cnt=0;
   FER(i, 1, n+1) cnt+=graph[a][i];
   FER(i, 1, n+1) cnt-=graph[i][a];
   cout<<cnt<<endl;
   fill(mark, 0);
   FER(k, 0, cnt){
      FER(i, 1, n+1) vis[i]=0;
      ands.clear();
      t=0;
      findallpaths(a, b, t);
      for(auto xd: ands) cout<<xd<<" "; cout<<endl;
   }
   return 0;
}
```

5 Math

5.1 Baby Step Giant Step

```
// Given a, b and m, fin a value "x" that
// a^x=(b)mod(m)
// Complexity in sqrt(m)

inline ll GetExpo(ll a, ll b, ll c){
    ll n=sqrt(c)+1;
    ll ans=1, cur;
    map<ll, ll> m;
    FER(i,1,n+1) ans=(ans*a)%c;
    cur=ans;
    FER(i,1,n+1){
        if(!m[cur]) m[cur]=i;
        cur=(cur*ans)%c;
    }
    cur=b;
    FER(i,0,n){
        if(m[cur]){
               return m[cur]*n-i;
        }
        cur=(cur*a)%c;
    }
}

int main(){
    fastio;
    ll t, ans; cin>>t;
    while(t--){
        ll a, b, m; cin>>a>>b>>m;
        if(a==b%m) {cout<<1<<endl; continue;}
        if(b==1 && m!=1) {cout<<0<<endl; continue;}
        ans=GetExpo(a, b, m);
        cout<<ans<<endl;
    }
```

```
    return 0;
}
```

5.2 Chinese Remainder Theorem

```
/* Chinese remainder theorem.
    Find z such that z % x[i] = a[i] for all i.
*/
ll crt(vector<ll> &a, vector<ll> &x) {
  ll z = 0;
  ll n = 1;
  for (int i = 0; i < x.size(); ++i)
   n *= x[i];

  for (int i = 0; i < a.size(); ++i) {
   ll tmp = (a[i] * (n / x[i])) % n;
   tmp = (tmp * mod_inv(n / x[i], x[i])) % n;
   z = (z + tmp) % n;
  }

  return (z + n) % n;
}
```

5.3 Cribas

```
// Criba en O(n)
// p[i] indica el valor del primo i-esimo
// A[i] indica que el menor factor primo de i
//        es el primo A[i] - esimo
#define MAXN 100000

int A[MAXN + 1], p[MAXN + 1], pc = 0;

void sieve()
{
   for(int i=2; i<=MAXN; i++){
      if(!A[i]) p[A[i] = ++pc] = i;
      for(int j=1; j<=A[i] && i*p[j]<=MAXN; j++)
         A[i*p[j]] = j;
   }
}

//Criba para phi
int phi[MAX];

void CribaEuler(){
   FER(i,0, MAX) primo[i] = 1, phi[i] = 1;
   primo[0] = primo[1] = false;
   FER(i,2,MAX){
      if(primo[i]){
         phi[i] = i - 1;
         for(int j = i+i; j < MAX; j += i){
            primo[j] = false;
            int pot = 1, aux = j/i;
            while( aux % i == 0 ){
               aux /= i, pot *= i;
            }
            phi[j] *= (i-1)*pot ;
         }
      }
   }
}
```

5.4 Euler Totient

```
//Euler Totient
//Finds all k <= n where gcd(k,n) == 1

int phi(int n){
   int res = n;
   for(int p = 2; p*p <= n; p++){
      if(n % p == 0){
         while(n % p == 0)
            n/=p;
         res -= res/p;
      }
   }
   if (n > 1) res -= res/n;
   return res;
}
```

5.5 FFT

```
//FFT

#define pi acos(-1)
struct cpx{
   double a, b;
   cpx() : a(0.0), b(0.0){}
   cpx(double a, double b) : a(a), b(b){}
   const cpx operator+(const cpx &c) const{return
       cpx(a+c.a, b+c.b);}
   const cpx operator-(const cpx &c) const{return
       cpx(a-c.a, b-c.b);}
   const cpx operator*(const cpx &c) const{return
       cpx(a*c.a-b*c.b, a*c.b+b*c.a);}
};
struct FFT{
   vector<cpx> data, roots;
   vi rev;
   ll s, n;
   inline void init(ll pw){
      s=pw, n=(1<<s);
      rev=vi(n), data=vector<cpx> (n),
          roots=vector<cpx> (n+1);
      FER(i,0,n) FER(j,0,s) if((i & (1<<j))!=0)
          rev[i]+=(1<<(s-j-1));
      roots[0]=cpx(1, 0);
      cpx mult=cpx(cos(2*pi/n), sin(2*pi/n));
      FER(i,1,n+1) roots[i]=roots[i-1]*mult;
   }
   inline void transform(bool inverse=false){
      vector<cpx> temp(n);
      FER(i,0,n) temp[i]=data[rev[i]];
      FER(i,0,n) data[i]=temp[i];
      FER(i,1,s+1){
         ll m=(1<<i), md2=m/2;
         ll start=0, incr=(1<<(s-i));
         cpx t;
         if(inverse) start=n, incr*=-1;
         for(ll k=0; k<n; k+=m){
            ll idx=start;
            FER(j,k,md2+k){
               t=roots[idx]*data[j+md2];
```

```
            idx+=incr;
            data[j+md2]=data[j]-t;
            data[j]=data[j]+t;
          }
        }
      }
      if(inverse) FER(i,0,n) data[i].a/=n,
          data[i].b/=n;
   }
};
inline void build(vi &a, FFT &f, ll y, ll n){
   f.init(y);
   FER(i, 0, sz(a)) f.data[i]=cpx(a[i], 0);
   FER(i,sz(a), n) f.data[i]=cpx(0, 0);
   f.transform();
}
inline vi GetConvolution(vi &a, vi &b){
   ll t1=sz(a), t2=sz(b), ta=t1+t2-1, x=0, n;
   while((1<<x)<ta) x++;
   n=1<<x;
   FFT f1, f2;
   build(a, f1, x, n), build(b, f2, x, n);
   FER(i,0,n) f1.data[i]=f1.data[i]*f2.data[i];
   f1.transform(true);
   vi froz(ta);
   FER(i,0,ta) froz[i]=(ll) (f1.data[i].a+0.5);
   return froz;
}
int main(){
   fastio;
   ll t; cin>>t;
   while(t--){
      ll n; cin>>n;
      vi v1, v2;
      FER(i,0,n+1){
         ll x; cin>>x;
         v1.pb(x);
      }
      FER(i,0,n+1) {
         ll x; cin>>x;
         v2.pb(x);
      }
      vi c=GetConvolution(v1, v2);
      for(auto xd: c) cout<<xd<<" "; cout<<endl;
   }
   return 0;
}
```

5.6 FWHTAnd

```
// FWHT And

inline ll bp(ll a, ll n){
   if(n==0) return 1;
   if(n&1) return (a*bp(a, n-1))%mod;
   ll x=bp(a, n>>1);
   return (sqr(x)%mod+mod)%mod;
}

// Fast Walsh-Hadamard Transform for And
    multiplication
// take note about the matrix associated
// |-1 +1|
// |+1 +0|
// The inverse is not the same as the matrix, so
    it's necesary
// to have a bool parameter in the convolution

struct FWHTAnd{
   ll n;
   inline void GetInverse(vi &v, ll flag){
      n=1;
      while(n<sz(v)) n<<=1;
      v.resize(n);
      for(ll len=1; (len<<1)<=n; len<<=1){
         for(ll i=0; i<n; i+=(len<<1)){
            FER(j,0,len){
               ll a=v[i+j], b=v[i+j+len];
               if(flag){
                  v[i+j]=b;
                  v[i+j+len]=a+b;
                  v[i+len+j]%=mod;
               }
               else{
                  v[i+j]=mod-a+b;
                  v[i+j+len]=a;
                  v[i+j]%=mod;
               }
            }
         }
      }

   }
}fwht;
int main(){
   fastio;
   ll n, top=0, ans=0; cin>>n;
   vi v1, v2;
   FER(i,0,n){
      ll x; cin>>x;
      top=max(top, x);
      v1.pb(x);
   }
   top++;
   v2.resize(top);
   v2[0]=1;
   for(auto xd: v1) v2[xd]=1;
   fwht.GetInverse(v2, 1);
   FER(i,0,fwht.n) v2[i]=bp(v2[i], n);
   fwht.GetInverse(v2, 0);
   FER(i,0,fwht.n) if(v2[i]) ans++;
   cout<<ans<<endl;
   return 0;
}
```

5.7 FWHTXor

```
// FWHT XOR

inline ll bp(ll a, ll n){
   if(n==0) return 1;
   if(n&1) return (a*bp(a, n-1))%mod;
   ll x=bp(a, n>>1);
   return (sqr(x)%mod+mod)%mod;
}

// Fast Walsh-Hadamard Transform for xor
    multiplication
```

```
// take note about the matrix associated
// |+1 +1|
// |+1 -1|
// The inverse is the same as the matrix, so it's
    not necesary
// to have a bool parameter in the convolution

struct FWHTXor{
   ll n;
   inline void GetInverse(vi &v){
      n=1;
      while(n<sz(v)) n<<=1;
      v.resize(n);
      for(ll len=1; (len<<1)<=n; len<<=1){
         for(ll i=0; i<n; i+=(len<<1)){
            FER(j,0,len){
               ll a=v[i+j], b=v[i+j+len];
               v[i+j]=(a+b)%mod;
               v[i+j+len]=(mod+a-b)%mod;
            }
         }
      }
   }
}fwht;
int main(){
   fastio;
   ll n, k; cin>>n>>k;
   vi v;
   FER(i,0,k+1) v.pb(1);
   fwht.GetInverse(v);
   FER(i,0,fwht.n) v[i]=bp(v[i], n);
   fwht.GetInverse(v);
   ll inv=bp(fwht.n, mod-2), ans=0;
   FER(i,1,fwht.n){
      ans+=(v[i]*inv)%mod;
      ans%=mod;
   }
   cout<<ans<<endl;
   return 0;
}
```

5.8 Gauss Jordan

```
// O(n^3) Gauss-Jordan elimination with full
    pivoting.
// Takes a[][] nxn, b[][] nxm
// Returns b = a^{-1}b in b, a^{-1} in a,
    determinant
const double EPS = 1e-10;
typedef double T;
typedef vector<vector<double>> vvt;

T GaussJordan(vvt &a, vvt &b) {
  const int n = a.size();
  const int m = b[0].size();
  vi irow(n), icol(n), ipiv(n);
  T det = 1;

   for (int i = 0; i < n; i++) {
      int pj = -1, pk = -1;
      for (int j = 0; j < n; j++) if (!ipiv[j])
         for (int k = 0; k < n; k++) if (!ipiv[k])
            if (pj == -1 || fabs(a[j][k]) >
                fabs(a[pj][pk])) {
               pj = j; pk = k;
            }
      if (fabs(a[pj][pk]) < EPS) {
         cerr << "Matrix is singular." << endl;
            exit(0);
      }
      ipiv[pk]++;
      swap(a[pj], a[pk]);
      swap(b[pj], b[pk]);
      if (pj != pk) det *= -1;
      irow[i] = pj;
      icol[i] = pk;

      T c = 1.0 / a[pk][pk];
      det *= a[pk][pk];
      a[pk][pk] = 1.0;
      for (int p = 0; p < n; p++) a[pk][p] *= c;
      for (int p = 0; p < m; p++) b[pk][p] *= c;
      for (int p = 0; p < n; p++) if (p != pk) {
         c = a[p][pk];
         a[p][pk] = 0;
         for (int q = 0; q < n; q++) a[p][q] -=
             a[pk][q] * c;
         for (int q = 0; q < m; q++) b[p][q] -=
             b[pk][q] * c;
      }
   }

   for (int p = n-1; p >= 0; p--) if (irow[p] !=
       icol[p]) {
      for (int k = 0; k < n; k++)
         swap(a[k][irow[p]], a[k][icol[p]]);
   }
   return det;
}
```

5.9 Inverso Modular

```
/** Inverso Modular **/
#define MAX 100
#define MOD 1000000009

ll inverso[MAX];

void inv(){
   inverso[1] = 1;
   FER(i,2,MAX) inverso[i] = ( (MOD-MOD/i) *
       inverso[MOD%i] ) % MOD;
}
```

5.10 Matrix Determinant

```
/* Matrix Determinant */

ll matrix[1<<8][1<<8];

inline ll FindDeterminant(ll n, ll m){
   FER(i,0,n) FER(j,0,n) {
      matrix[i][j]%=m;
      matrix[i][j]+=m;
```

```
      if(matrix[i][j]>=m) matrix[i][j]-=m;
   }
   ll res=1;
   FER(i,0,n){
      if(!matrix[i][i]){
         bool flag=false;
         FER(j,i+1,n){
            if(matrix[j][i]){
               flag=true;
               FER(k,i,n) swap(matrix[i][k],
                   matrix[j][k]);
               res=-res;
               break;
            }
         }
         if(!flag) return 0;
      }
      FER(j,i+1,n){
         while(matrix[j][i]){
            ll t=matrix[i][i]/matrix[j][i];
            FER(k,i,n){
               matrix[i][k]=(matrix[i][k]-
                  t*matrix[j][k])%m;
               swap(matrix[i][k], matrix[j][k]);
            }
            res=-res;
         }
      }
      res=(res*matrix[i][i])%m;
   }
   return (res+m)%m;
}
```

5.11 Matrix Exponentiation

```
void sum(ll &a, ll b){
   a += b;
   if(a >= MOD) a -= MOD;
}

void Init(ll a[N][N]){
   FER(i,0,N) FER(j,0,N) a[i][j] = (i == j);
}

void Cpy(ll a[N][N], ll b[N][N]){
   FER(i,0,N) FER(j,0,N) b[i][j] = a[i][j];
}

void Mult(ll a[N][N], ll b[N][N], ll c[N][N]){
   FER(i,0,N) FER(j,0,N) c[i][j] = 0;
   FER(i,0,N) FER(j,0,N) FER(k,0,N)
      sum(c[i][j],(a[i][k]*b[k][j])%MOD);
}

void BinPow(ll a[N][N], ll p, ll ans[N][N]){
   ll aux[N][N]; Init(ans);
   while(p){
      if(p&1LL) Mult(ans,a,aux), Cpy(aux,ans);
      p >>= 1; Mult(a,a,aux); Cpy(aux,a);
   }
}
```

5.12 Miller Rabin

```
//Miller-Rabin primality test

ll pow(ll a, ll b, ll c){
   ll ans = 1;
   while(b){
      if(b&1) ans = (1LL*ans*a)%c;
      a = (1LL*a*a)%c;
      b >>=1;
   }
   return ans;
}

bool miller(ll p, ll it = 10){
   if(p<2) return 0;
   if(p!=2 && (p&1) == 0) return 0;
   ll s=p-1;
   while((s&1) == 0) s>>=1;
   while(it--){
      ll a = rand()%(p-1)+1, temp = s;
      ll mod = pow(a,temp,p);
      while(temp!= p-1 && mod!=1 && mod!=p-1){
         mod = (1LL*mod*mod)%p;
         temp<<=1;
      }
      if(mod!=p-1 && (temp&1) == 0) return 0;
   }
   return 1;
}
```

5.13 Mobius Function

```
// Mobius function

vi mo, lp, primes;
ll ar[1<<20], cnt[1<<20];

inline void ContructMobius(ll n){
   lp.resize(n+1, -1), mo.resize(n+1, -1);
   mo[1]=1;
   FER(i, 2, n+1){
      if(lp[i]==-1) lp[i]=i, primes.pb(i);
      ll j=0;
      for(; j<sz(primes) && primes[j]*i<=n &&
          primes[j]< lp[i]; j++){
         lp[primes[j]*i]=primes[j],
             mo[primes[j]*i]=-mo[i];
      }
      if(primes[j]*i<=n) mo[primes[j]*i]=0;
   }
}

int main(){
   fastio;
   ContructMobius(1e6);
   ll n; cin>>n;
   FER(i,0,n) cin>>ar[i], cnt[ar[i]]++;
   ll ans=0;
   FER(i, 1, 1e6+1){
      for(ll j=i<<1; j<=1e6; j+=i) cnt[i]+=cnt[j];
   }
```

```
   FER(i, 1, 1e6+1)
       ans+=mo[i]*(sqr(cnt[i])-cnt[i])*(cnt[i]-2)/6;
   cout<<ans<<endl;
   return 0;
}
```

5.14 NTTDFT

```
//NTT

struct NTT{
   ll root_1, root, root_pw;
   inline ll bp(ll a, ll b){
      if(b==0) return 1;
      if(b&1) return (a*(bp(a, b-1)))%mod;
      ll x=bp(a, b>>1)%mod;
      x=(x+mod)%mod;
      return ((x*x)%mod+mod)%mod;;
   }
   inline ll inverse(ll a){
      return bp(a, mod-2);
   }
   inline void build(){
      // Si:c*pow(2, k)+1=mod
      // Entonces root cumple:
      // pow(root, 1<<(pow(2, k))=1mod
      // Mejor complejidad=Primary root test.
      root_pw=1<<23;
      FER(i,2,1e5+1) if(bp(i, root_pw)==1){
          root=i; break;}
      root_1=inverse(root);
   }
   inline void fft(vi &a , bool inv){
      ll n=sz(a);
      for(ll i=1, j=0; i<n; i++){
         ll bt=n>>1;
         for(; j&bt; bt>>=1) j^=bt;
         j^=bt;
         if(i<j) swap(a[i], a[j]);
      }
      for(ll len=2; len<=n; len<<=1){
         ll wlen=inv? root_1: root;
         for(ll i=len; i<root_pw; i<<=1)
             wlen=(sqr(wlen)%mod+mod)%mod;
         for(ll i=0; i<n; i+=len){
            ll w=1;
            for(ll j=0; j<len/2; j++){
               ll u=a[i+j], v=(a[i+j+len/2]*w)%mod;
               a[i+j]=u+v<mod? u+v: u+v-mod;
               a[i+j+len/2]=u-v>=0? u-v: u-v+mod;
               w=((w*wlen)%mod+mod)%mod;
            }
         }
      }
      if(inv){
         ll n_1=inverse(sz(a));
         for(auto &xd: a)
             xd=((xd*n_1)%mod+mod)%mod;
      }
   }
   inline vi GetConvolution(vi a, vi b){
      ll n=1;
      while(n<sz(a)+sz(b)) n<<=1;
      a.resize(n), b.resize(n);
      fft(a, false), fft(b, false);
      FER(i,0,n) a[i]=((a[i]*b[i])%mod+mod)%mod;
      fft(a, true);
      return a;
   }
   inline vi BinPolyExp(vi &a, ll b){
      if(b==1) return a;
      ll n=1;
      while(n<sz(a)*b) n<<=1;
      a.resize(n);
      fft(a, false);
      for(auto &xd: a) xd=bp(xd, b);
      fft(a, true);
      return a;
   }
}ntt;


int main(){
   fastio;
   NTT ntt;
   ntt.build();
   ll n, k; cin>>n>>k;
   vi v(10);
   FER(i,0,k){
      ll x; cin>>x;
      v[x]=1;
   }
   ll ans=0;
   vi froz=ntt.BinPolyExp(v, n/2);
   for(auto xd: froz) ans=(ans+sqr(xd))%mod;
   ans=(ans+mod)%mod;
   cout<<ans<<endl;
   return 0;
}
```

5.15 Number Theory

```
// return a % b (positive value)
int mod(int a, int b) {
  return ((a%b)+b)%b;
}

// returns d = gcd(a,b); finds x,y such that d =
    ax + by
int extended_euclid(int a, int b, int &x, int &y)
    {
  int xx = y = 0;
  int yy = x = 1;
  while (b) {
   int q = a/b;
   int t = b; b = a%b; a = t;
   t = xx; xx = x-q*xx; x = t;
   t = yy; yy = y-q*yy; y = t;
  }
  return a;
}

// finds all solutions to ax = b (mod n)
vi modular_linear_equation_solver(int a, int b,
    int n) {
  int x, y;
  vi solutions;
  int d = extended_euclid(a, n, x, y);
```

```
  if (!(b%d)) {
   x = mod (x*(b/d), n);
   for (int i = 0; i < d; i++)
     solutions.push_back(mod(x + i*(n/d), n));
  }
  return solutions;
}

// computes b such that ab = 1 (mod n), returns
    -1 on failure
int mod_inverse(int a, int n) {
  int x, y;
  int d = extended_euclid(a, n, x, y);
  if (d > 1) return -1;
  return mod(x,n);
}

// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i. Note that the
    solution is
// unique modulo M = lcm_i (x[i]). Return (z,M).
    On
// failure, M = -1. Note that we do not require
    the a[i]'s
// to be relatively prime.
// Note that it's NOT necessary that x[i] > 0
// But if they are postive, it makes MODULO > 0
    (convenient)
ii chinese_remainder(int x, int a, int y, int b) {
  int s, t;
  x = abs(x) , y = abs(y); // x and y NEED to be
      positive here
  int d = extended_euclid(x, y, s, t);
  if (a%d != b%d) return make_pair(0, -1);
  return make_pair(mod(s*b*x+t*a*y,x*y)/d, x*y/d);
}
ii chinese_remainder(const vi &x, const vi &a) {
  ii ret = make_pair(a[0], abs(x[0]));
  for (int i = 1; i < x.size(); i++) {
   ret = chinese_remainder(ret.second, ret.first,
       x[i], a[i]);
   if (ret.second == -1) break;
  }
  return ret;
}
```

```
// Criba en O(n)
// p[i] indica el valor del primo i-esimo
// A[i] indica que el menor factor primo de i
//        es el primo A[i] - esimo
#define MAXN 100000
int A[MAXN + 1], p[MAXN + 1], pc = 0;
void sieve()
{
   for(int i=2; i<=MAXN; i++){
      if(!A[i]) p[A[i] = ++pc] = i;
      for(int j=1; j<=A[i] && i*p[j]<=MAXN; j++)
         A[i*p[j]] = j;
   }
}

// Lucas Teorem
// Cuando n y m son grandes y se pide
    comb(n,m)%MOD, donde
// MOD es un numero primo, se puede usar el
    Teorema de Lucas.
ll comb( ll n, ll k ){
   ll ans = 1;
   while( n > 0 ){
      ll ni = n%MOD,ki = k%MOD;
      n /= MOD; k /= MOD;
      if( ni - ki < 0 )return 0;
      ll temp = (FP[ki]*FP[ni-ki])%MOD;
      temp = (temp*F[ni])%MOD;
      ans = (ans*temp)%MOD;
   }
   return ans;
}

// Criba en O(n) real

ll vis[100000005];
inline bool check(int prime[], int x){
   return prime[x/64] & (1<<((x>>1)&31));
}

inline void mark(int prime[], int x){
   prime[x/64]|=(1<<((x>>1)&31));
}
```

```
}
inline void bit(ll n){
   int prime[n/64];
   fill(prime, 0);
   for(ll i=3; sqr(i)<=n; i+=2){
      if(!check(prime, i)){
         for(ll j=sqr(i), k=i<<1; j<n; j+=k)
             mark(prime, j);
      }
   }
   printf("2\n");
   ll cnt=1;
   for(ll i=3; i<n; i+=2) if(!check(prime, i)) {
      cnt++;
      if(cnt%100==1) printf("%lld\n", i);
   }
}
int main(){
   bit(1e8);
   return 0;
}
```

5.16 PiFunction

```
/* Pi Function*/

ll vis[10000005], cant[10000005],
    phi[100005][105];
vi primos;

inline ll FindPhi(ll x, ll p){
   if(!p) return x;
   if(primos[p-1]>=x) return 1;
   if(x<=1e5 && p<=1e2) return phi[x][p];
   return FindPhi(x, p-1)-FindPhi(x/primos[p-1],
       p-1);
}

inline ll PiFuncion(ll n){
   if(n<1e7) return cant[n];
   ll raizC2=sqrt(n), raizC3=pow(n, 1.0/3.0);
   ll m=PiFuncion(raizC3);
```

```
   ll ans=FindPhi(n, m)+m-1;
   for(ll i=m; primos[i]<=raizC2; i++){
      ans-=PiFuncion(n/primos[i]);
      ans+=PiFuncion(primos[i]);
      ans--;
   }
   return ans;
}
int main(){
   fastio;
   FER(i,2,1e7){
      cant[i]+=cant[i-1];
      if(vis[i]) continue;
      cant[i]++;
      primos.pb(i);
      for(ll j=i*i; j<1e7; j+=i) vis[j]++;
   }
   FER(i,0,101){
      FER(j,0,1e5+1) phi[j][i]=i?
          phi[j][i-1]-phi[j/primos[i-1]][i-1]: j;
   }
   ll n, ans=0; cin>>n;
   for(auto xd: primos){
      ll x=xd, y=n/xd;
      if(sqr(x)>n) break;
      ans+=PiFuncion(y)-PiFuncion(x);
   }
   for(auto xd: primos) {
      if(sqr(xd)*xd<=n) ans++;
      else break;
   }
   cout<<ans<<endl;
   return 0;
}
```

5.17 Pisano

```
// Find pisano of Fib[i]%x
// i.e. Spoj pisano

inline ll bm(ll a, ll b, ll m){
       if(a==0) return 0LL;
       if(a&1) {
              ll x=bm(a-1, b, m);
              x+=b;
              if(x>=m) x-=m;
              return x;
       }
       ll x=bm(a>>1, b, m);
       x<<=1;
       if(x>=m) x-=m;
       return x;
}


struct matrix{
       ll ar[2][2];
       inline void clear(){
              fill(ar, 0);
       }
};

matrix ind;

inline matrix mul(matrix &a, matrix b, ll m){
       matrix ty;
       FER(i,0,2) FER(j,0,2){
              ll ans=0;
              FER(k,0,2){
                     ans+=bm(a.ar[i][k],
                         b.ar[k][j], m);
                     if(ans>=m) ans-=m;
              }
              ty.ar[i][j]=ans;
       }
       return ty;
}


inline matrix bpmatrix(matrix &a, ll n, ll m){
       if(n==1) return a;
       if(n&1) return mul(a, bpmatrix(a, n-1, m),
           m);
       matrix x=bpmatrix(a, n>>1, m);
       return mul(x, x, m);
}


inline ll fib(ll n, ll m){
       matrix cur=bpmatrix(ind, n, m);
       return cur.ar[0][1];
}

// here we need the pollard rho algorithm
// asuming that we have that here


map<ll, ll> mapis;

inline bool testpisano(ll n, ll m){
       if(fib(1+n, m)==1 && fib(2+n, m)==1)
           return true;
       return false;
}

inline ll pisanoprime(ll n, ll m){
       ll resp=n, ans, cur;
       vii v;
       make(resp);
       construct(v);
       for(auto xd: v){
              ans=1LL, cur=-1;
              FER(i, 0, xd.ss+1){
                     if(testpisano(resp/ans, m))
                         cur=ans;
                     ans*=xd.ff;
              }
              resp/=cur;
       }
       return resp;
}


inline ll pisano(ll n){
       if(mapis.count(n)) return mapis[n];
       ll p=n%5, m=n-1;
       if(p==1 || p==4) return
           mapis[n]=pisanoprime(m, n);
       m+=2, m<<=1;
       return mapis[n]=pisanoprime(m, n);
```

```
}

inline ll binpow(ll a, ll n){
	if(!n) return 1LL;
	if(n&1) return a*binpow(a, n-1);
	ll x=binpow(a, n>>1);
	return sqr(x);
}

inline ll findpisano(ll n){
	vii v;
	make(n);
	construct(v);
	ll ans, cur, resp=1LL;
	for(auto xd: v){
		ans=binpow(xd.ff, xd.ss-1);
		cur=pisano(xd.ff)*ans;
		ans=__gcd(resp, cur);
		resp/=ans;
		resp*=cur;
	}
	return resp;
}



int main(){
	fastio;
	build();
	// because of pollard rho template
	ind.ar[0][0]=1, ind.ar[0][1]=1;
	ind.ar[1][0]=1, ind.ar[1][1]=0;
	mapis[2]=3, mapis[3]=8, mapis[5]=20;
	ll t; cin>>t;
	while(t--){
		ll x; cin>>x;
		cout<<findpisano(x)<<endl;
	}
	return 0;
}
```

5.18 Pollard Rho + Divisors

```
/* Pollard Rho + Recursive Divisors
   Autor: Phibrain
*/

const int maxp = 1e6 + 1, maxv = 25, maxc =
    (int)1e4 + 1;
int ptot, pr[maxp], d[maxp], cnt;
ll p[maxc];
inline ll mod_add(ll x, ll y, ll p) {
  return (x += y) < p ? x : x - p;
}
inline ll mod_mul(ll x, ll y, ll p) {
  ll ret = x * y - (ll)((long double)x * y / p +
      0.5) * p;
  return ret < 0 ? ret + p : ret;
}
inline ll mod_pow(ll x, ll k, ll p) {
  ll ret = 1 % p;
  for( ; k > 0; k >>= 1, x = mod_mul(x, x, p))
   (k & 1) && (ret = mod_mul(ret, x, p));
  return ret;
}
bool miller_rabin(ll n) {
  if(n == 2) return 1;
  if(n < 2 || !(n & 1))
   return 0;
  ll s = 0, r = n - 1;
  for( ; !(r & 1); r >>= 1, ++s);
  for(int i = 0; pr[i] < n && pr[i] < maxv; ++i) {
   ll cur = mod_pow(pr[i], r, n), nxt;
   for(int j = 0; j < s; ++j) {
     nxt = mod_mul(cur, cur, n);
     if(nxt == 1 && cur != 1 && cur != n - 1)
         return 0;
     cur = nxt;
   }
   if(cur != 1) return 0;
  }
  return 1;
}
inline ll gcd(ll a, ll b) {
  int ret = 0;
  while(a) {
   for( ; !(a & 1) && !(b & 1); ++ret, a >>= 1, b
       >>= 1);
   for( ; !(a & 1); a >>= 1);
   for( ; !(b & 1); b >>= 1);
   if(a < b)
     swap(a, b);
   a -= b;
  }
  return b << ret;
}
inline ll pollard_rho(ll n) {
  static ll seq[maxp];
  while(1) {
   ll x = rand() % n, y = x, c = rand() % n;
   ll *px = seq, *py = seq, tim = 0, prd = 1;
   while(1) {
     *py++ = y = mod_add(mod_mul(y, y, n), c, n);
     *py++ = y = mod_add(mod_mul(y, y, n), c, n);
     if((x = *px++) == y) break;
     ll tmp = prd;
     prd = mod_mul(prd, abs(y - x), n);
     if(!prd) return gcd(tmp, n);
     if((++tim) == maxv) {
      if((prd = gcd(prd, n)) > 1 && prd < n)
          return prd;
      tim = 0;
     }
   }
   if(tim && (prd = gcd(prd, n)) > 1 && prd < n)
       return prd;
  }
}
inline void decompose(ll n) {
  for(int i = 0; i < cnt; ++i)
   if(n % p[i] == 0) {
     p[cnt++] = p[i];
     n /= p[i];
   }
  if(n < maxp) {
   for( ; n > 1; p[cnt++] = d[n], n /= d[n]);
  } else if(miller_rabin(n)) {
   p[cnt++] = n;
```

```
  } else {
   ll fact = pollard_rho(n);
   decompose(fact), decompose(n / fact);
  }
}
inline void build(){
  FER(i, 2, maxp) {
   if(!d[i]) pr[ptot++] = d[i] = i;
   for(int j = 0, k; (k = i * pr[j]) < maxp; ++j)
       {
     d[k] = pr[j];
     if(d[i] == pr[j]) break;
   }
  }
}
inline void construct(vii &v){
  sort(p, p+cnt);
  for(int i = 0; i < cnt; ) {
   ll ctr=0;
   for(ll pp = p[i]; i < cnt && p[i] == pp; ++i,
       ++ctr);
   v.pb({p[i-1], ctr});
  }
  cnt=0;
}
inline void make(ll n){
  decompose(n);
}
vi froz[N], v[N];
ll ja;
inline void go(ll n, ll t, vi &tt){
   if(n>=t){
      for(auto xd: tt) froz[ja].pb(xd);
      ja++;
      tt.pop_back();
      return;
   }
   FER(i,0,sz(v[n])){
      tt.pb(v[n][i]);
      go(n+1, t, tt);
   }
   if(sz(tt)) tt.pop_back();
}
```

```
inline void solve(ll t){
   vi tt;
   go(0, t, tt);
}

inline void kotaro(ll n, vi &v){
   FER(i, 0, n){
      ll t = 1;
      for(auto xd: froz[i]) t *= xd;
      v.pb(t);
   }
}

int main(){
   build();
   ll n; cin>>n;
   make(n);
   vii v1;
   construct(v1);
   cout<<"La factorizacion es: "<<endl;
   for(auto xd: v1){
      cout<<xd.ff<<" "<<xd.ss<<endl;
   }
   cout<<"Los divisores son: "<<endl;
   ja=0;
   FER(i,0,sz(v1)){
      ll t=1;
      ll ta=v1[i].ss+1, num=v1[i].ff;
      FER(j,0,ta) {
         v[i].pb(t);
         t*=v1[i].ff;
      }
   }
   solve(sz(v1));
   vi tnt;
   kotaro(ja, tnt);
   sort(all(tnt));
   for(auto xd: tnt) cout<<xd<<" "; cout<<endl;
   return 0;
}
```

5.19 Primality Test (pq prime)

```
// Primality test for [0,3*10^18] (Lucas97)
const ll MOD = 1e9 + 7;
const int MN = 1e6 + 2;
bool primos[MN * 5]; // 1 if is prime, 0 otherwise
int cnt = 0; // cnt denotes the number of primes
int prime[MN * 5];

ll mulmod(ll a, ll b, ll c) {
  ll x = 0, y = a % c;
  while (b) {
   if (b & 1) x = (x + y) % c;
   y = (y << 1) % c;
   b >>= 1;
  }
  return x % c;
}

ll fastPow(ll x, ll n, ll mod) {
  ll ret = 1;
  while (n) {
   if (n & 1) ret = mulmod(ret, x, mod);
   x = mulmod(x, x, mod);
   n >>= 1;
  }
  return ret;
}

bool isPrime(ll n) {
   // Verificar primero los casos pequeos (hasta
       1e6)
  if(n <= 5 * MN - 10){
   if(!primos[n]) return 1;
   else return 0;
  }
  // Fin de verificacion
  ll d = n - 1;
  int s = 0;
  while (d % 2 == 0) {
   s++;
   d >>= 1;
  }
```

```
  // These values will work for any number
      smaller than 3*10^18
  //(Only 9 are necessary)
  int a[11] = { 2, 3, 5, 7, 11, 13, 17, 19, 23,
      29, 31 };
  for(int i = 0; i < 11; i++) {
   bool comp = fastPow(a[i], d, n) != 1;
   if(comp) for(int j = 0; j < s; j++) {
     ll fp = fastPow(a[i], (1LL << (ll)j)*d, n);
     if (fp == n - 1) {
      comp = false;
      break;
     }
   }
   if(comp) return false;
  }
  return true;
}
```

5.20 Primitive Root

```
// If a primitive root exists, there are
    phi(phi(n)) of them */

//Primitive root modulo p(prime p>=2)
//Complexity: O(plogp)
int generator(int p){
     int phi = p - 1, np = phi ;
     vector<int>fact;
     for(int i = 2; i*i<=np ; ++i){
        if(np%i == 0){
           fact.pb(i);
           while(np%i == 0) np/=i;
        }
     }
     if(np > 1) fact.pb(np);
     for(int ret = 1; ret < p ; ++ret){
        bool ok = 1;
        for(int i = 0; i<fact.size(); ++i){
           if( binpow(ret,phi/fact[i],p) == 1LL){
           ok = 0; break;
           }
        }
        if(ok) return ret;
     }
     return -1;
}

//Probabilistic method , Complexity: O(loglogp *
    p)
ll prob_generator(ll p){
   ll g , order;
      while(1){
         g = rand() % (p - 1) + 1;
         ll G = 1;
         for(order = 1 ;; ++order){
            G = (G * g) % p;
            if(G == 1) break;
         }
              if(order == p-1) break;
      }
   return g;
}
```

5.21 Row Reduced Form

```
// O(n^3) RREF via Gauss-Jordan with partial
    pivoting.
// Takes a[][] nxm matrix, convers to rref,
    returns rank
const double EPS = 1e-10;
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;
int rref(VVT &a) {
  int n = a.size();
  int m = a[0].size();
  int r = 0;
  for (int c = 0; c < m && r < n; c++) {
   int j = r;
   for (int i = r+1; i < n; i++)
     if (fabs(a[i][c]) > fabs(a[j][c])) j = i;
   if (fabs(a[j][c]) < EPS) continue;
   swap(a[j], a[r]);
   T s = 1.0 / a[r][c];
   for (int j = 0; j < m; j++) a[r][j] *= s;
   for (int i = 0; i < n; i++) if (i != r) {
     T t = a[i][c];
     for (int j = 0; j < m; j++) a[i][j] -= t *
         a[r][j];
   }
   r++;
  }
  return r;
}
```

5.22 Simplex Method

```
/* Simplex Method
   Autor: el Vasito
*/

vector<int> X,Y;
vector<vector<double> > A;
vector<double> b,c;
double z;
int n,m;
void pivot(int x,int y){
   swap(X[y],Y[x]);
   b[x]/=A[x][y];
   FER(i,0,m)if(i!=y)A[x][i]/=A[x][y];
   A[x][y]=1/A[x][y];
   FER(i,0,n)if(i!=x&&abs(A[i][y])>EPS){
      b[i]-=A[i][y]*b[x];
      FER(j,0,m)if(j!=y)A[i][j]-=A[i][y]*A[x][j];
      A[i][y]=-A[i][y]*A[x][y];
   }
   z+=c[y]*b[x];
   FER(i,0,m)if(i!=y)c[i]-=c[y]*A[x][i];
   c[y]=-c[y]*A[x][y];
}
pair<double,vector<double> > simplex( // maximize
    c^T x s.t. Ax<=b, x>=0
     vector<vector<double> > _A, vector<double>
         _b, vector<double> _c){
```

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

5.23 Simpson Integral

```
// Numerical Integration with Simpson's Rule
// Error is f^4(x) * (b-a)^5 / 2880

int n = 500; // n tiene que ser par
ld f( ld x ){ return sqrt( 1 + 1.0/x ) / x; }
int main(){
   int a , b ;
   while( cin >> a >> b ){
      ld dt = ( b - a )/(ld)n;
      ld I = 0 , pos = a;
      for( int i = 0 ; i < n ; i +=2 )
         I += dt * ( f( pos ) + 4*f( pos + dt ) +
            f( pos = pos + dt + dt ));
      printf( "%.2f\n" , double( I/3.0 ) );
   }
}
```

5.24 Stirling Numbers Second Kind

```
// ................ Stirling Numbers Second Kind
    ..................

// Building second kind of stirling numbers
// based on stir(n, k)=k*stir(n-1, k)+stir(n-1,
    k-1)
// stir(i, 0)=stir(0, i)=0, stir(0, 0)=1
// pow(x, p)= sum[k={1,...,p}]{stir(p,
    k)*k!*comb(x, k)
// sum[x={1, ..., n}]{pow(x, p}=
// sum[k={1,...,p}]{stir(p, k)*k!*comb(n+1, k+1)


ll stir[1<<10][1<<10], dp[1<<10], dpi[1<<10];

inline ll bp(ll a, ll n, ll m){
       if(!n) return 1LL%m;
       if(n&1) return (a*bp(a, n-1, m))%m;
       ll x=bp(a, n>>1, m);
       return (x*x)%m;
}
inline void Build(ll n, ll k, ll m){
       FER(i,0,n) stir[i][0]=0;
       FER(i,0,n) stir[0][i]=0;
       stir[0][0]=1;
       FER(i,1,n+1){
              FER(j,1,k+1){
                     stir[i][j]=(j*stir[i-1][j]+stir[i-1][j-1])%m;
              }
       }
       dp[0]=dp[1]=1;
       dpi[1]=1;
       FER(i,2,n) {
              dp[i]=(dp[i-1]*i)%m;
              dpi[i]=(dpi[i-1]*bp(i, m-2, m))%m;
       }
}

inline ll FindSum(ll n, ll k, ll m){
       ll ans=0, cur=1, coe=(n), comb=n+1;
       if(!k) return n;
       FER(i,1,k+1){
              cur=(dp[i]*stir[k][i])%m;
              comb=(comb*coe)%m;
              cur=(comb*cur)%m;
              cur=(cur*dpi[i+1])%m;
              ans+=cur;
              if(ans>=m) ans-=m;
              coe--;
       }
       return ans;
}

int main(){
       Build(350, 350, mod);
       ll t, n, k, froz, ands, tnt; scanf("%lld",
           &t);
       while(t--){
              scanf("%lld%lld", &n, &k);
              froz=FindSum(n, k, mod);
              ands=FindSum(n, k+1, mod);
              tnt=(froz*(n+1)-ands+mod)%mod;
              if(tnt>=mod) tnt-=mod;
              if(tnt<0) tnt+=mod;
              printf("%lld\n", tnt);
       }
       return 0;
}
```

5.25 Teorema de Lucas

```
ll comb[105][105];
```

```
//Devuleve la comb(n,k) % m para n,k grandes y m
    pequeno

ll lucas( ll n , ll k , ll m ){
  //Se puede precalcular la combinatoria afuera
  FER(i,0,52) FER(j,0,52){
    if( j == 0 ) comb[i][0] = 1;
    else if( j > i ) comb[i][j] = 0;
    else comb[i][j] = ( comb[i-1][j] +
        comb[i-1][j-1] ) % m;
  }

  ll ans = 1, x, y;

  while( n ){
    x = n % m, y = k % m;
    ans = ( ans * comb[x][y] ) % m;
    n /= m, k /= m;
  }
  return ans;
}
```

6 Misc

6.1 Centroid Decomposition Phibrain

```
//Centroid Decomposition Phibrain

struct CD{
  ll gid, n;
  vi graph[N], adj[N];
  ll tsz[N], p[N], id[N], rt[N];
  ll tsz1[N], p1[N], d1[N];
  inline void add(ll a, ll b){
    graph[a].pb(b);
    graph[b].pb(a);
  }
  inline ll dfs(ll u, ll pp){
    tsz[u]=1;
    for(auto xd: graph[u]) if(xd!=pp &&
        p[xd]==-1) tsz[u]+=dfs(xd, u);
    return tsz[u];
  }
  inline void descompose(ll u, ll pp, ll sb, ll
      prev){
    for(auto xd: graph[u]) if(xd!=pp &&
        p[xd]==-1 && (tsz[xd]*2) > sb)
        {descompose(xd, u, sb, prev); return;}
    p[u]=prev; ll pe;
    for(auto xd: graph[u]) if(p[xd]==-1)
        pe=dfs(xd, -1), descompose(xd, u,
        tsz[xd], u);
  }
  inline ll dfs1(ll u, ll pp, ll depth){
    p1[u]=pp, d1[u]=depth, tsz1[u]=1;
    for(auto xd: adj[u]) if(xd!=pp)
        tsz1[u]+=dfs1(xd, u, depth+1);
    return tsz1[u];
  }
  inline void make(ll u){
    ll pe=dfs1(u, -1, 0);
  }
  inline void go(ll u, ll root){
    id[u]=gid++, rt[u]=root;
    ll w=-1, bc=-1;
    for(auto xd: adj[u]) if(xd!=p1[u] &&
        tsz1[xd]>w) w=tsz1[xd], bc=xd;
    if(bc!=-1) go(bc, root);
    for(auto xd: adj[u]) if(xd!=p1[u] && xd!=bc)
        go(xd, xd);
  }
  inline ll lca(ll a, ll b){
    while(rt[a]!=rt[b]) d1[rt[a]]>d1[rt[b]]?
        a=p1[rt[a]]: b=p1[rt[b]];
    return d1[a]>d1[b]? b: a;
  }
  inline void build(ll u){
    FER(i,0,N) p[i]=-1;
    ll ja=dfs(u, -1);
    descompose(u, -1, tsz[u], -2);
    ll nod;
    FER(i,0,n) if(p[i]==-2) {nod=i; break;}
    FER(i,0,n) if(i!=nod) adj[i].pb(p[i]),
        adj[p[i]].pb(i);
    gid=0;
    make(nod);
    go(nod, nod);
  }
}cd;

//cd.add(a, b); cd.build(0); rooted in 0
```

6.2 Centroid Decomposition itu

```
//Centroid Decomposition itu

const int MAXCD = 1e5 + 5;
struct CentroidD{
  //MAXCD: tamanho maximo del grafo
  vector< int > graph[MAXCD];
  int sub[MAXCD];//size de subtree en CD tree
  int cpar[MAXCD];//padre en el centroid tree
  //la raiz tiene padre -2

  void add_edge(int a, int b){
    graph[a].pb(b);
    graph[b].pb(a);
  }

  void dfs(int cur, int parent){
    sub[cur] = 1;
    for(int i = 0; i < sz(graph[cur]); ++i){
      int to = graph[cur][i];
      if(to != parent && cpar[to] == -1){
        dfs(to, cur);
        sub[cur] += sub[to];
      }
    }
  }

  void decompose(int cur, int parent, int sb,
      int prevc){
    for(int i = 0; i < sz(graph[cur]); ++i){
      int to = graph[cur][i];
      if(to != parent && cpar[to] == -1 && (2 *
          sub[to] > sb)){
        decompose(to, cur, sb, prevc);
```

```
            return;
         }
      }
      cpar[cur] = prevc;
      for(int i = 0; i < sz(graph[cur]); ++i){
         int to = graph[cur][i];
         if(cpar[to] == -1){
            dfs(to, - 1);
            decompose(to, cur, sub[to], cur);
         }
      }
   }

   void init(int start){
      for(int i = 0; i < MAXCD; ++i) cpar[i] = -1;
      dfs(start, - 1);
      decompose(start, -1, sub[start], -2);
   }
};
```

6.3 Closest Pair

```
//Closest Pair Algorithm with Sweep
//Complexity: O(nlogn)

#define MAX_N 100000
#define px second
#define py first
typedef pair<long long, long long> point;

int N;
point P[MAX_N];
set<point> box;

bool compare_x(point a, point b){ return
    a.px<b.px; }

inline double dist(point a, point b){
       return
           sqrt((a.px-b.px)*(a.px-b.px)+(a.py-b.py)*(a.py-b.py));
}
double closest_pair(){
       if(N<=1) return -1;
       sort(P,P+N,compare_x);
       double ret = dist(P[0],P[1]);
       box.insert(P[0]);
       set<point> :: iterator it;
       for(int i = 1,left = 0;i<N;++i){
              while(left<i &&
                  P[i].px-P[left].px>ret)
                  box.erase(P[left++]);
              for(it =
                  box.lower_bound(make_pair(P[i].py-ret,P[i].px-ret));
              it!=box.end() &&
                  P[i].py+ret>=(*it).py;++it)
              ret = min(ret, dist(P[i],*it));
              box.insert(P[i]);
       }
       return ret;
}
```

6.4 Convex Hull Trick

```
 // Simple Hull
struct HullSimple { // Upper envelope for Maximum.
   // Special case: strictly increasing slope in
       insertions,
   // increasing value in queries.
   deque<pair<ll, ll> > dq;
   ld cross(pair<ll, ll> l1, pair<ll, ll> l2){
      return (ld)(l2.snd - l1.snd) / (ld)(l1.fst -
          l2.fst);
   }
   void insert_line(ll m, ll b){
      pair<ll,ll> line = mp(m,b);
      while (sz(dq) > 1 && cross(line,
          dq[sz(dq)-1]) <=
           cross(dq[sz(dq)-1], dq[sz(dq)-2]))
               dq.pop_back();
      dq.pb(mp(m,b));
   }
   ll eval(pair<ll, ll> line, ll x){
      return line.fst * x + line.snd;
   }

   ll eval(ll x){
      while (sz(dq) > 1 && eval(dq[0], x) <
          eval(dq[1],x))
         dq.pop_front();
      return eval(dq[0],x);
   }
};

// Dynamic Hull
// Compile with g++ -std=c++11 file.cpp -o file
typedef long double ld;
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;
   }
};
// Upper envelope for Maximum
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 (x->b - y->b)*(z->m - y->m) >=
              (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;
  }
};
```

6.5 Dates

```
// Months: [1,12], Days: [1,31], Years: [0, 9999]
string dayOfWeek[7] = {"Mon", "Tue", "Wed",
    "Thu", "Fri", "Sat", "Sun"};
// During a leap year, february is 29 and NOT 28
int daysPerMonth[12] = {31, 28, 31, 30, 31, 30,
    31, 31, 30, 31, 30, 31 }
// Decides whether a year is leap in Gregorian
    calendar
int isLeap(int y){
  if (y%4) return 0;
  if (y%100) return 1;
  if (y%400) return 0;
  return 1;
}
// converts Gregorian date to integer (Julian day
    number)
int dateToInt (int m, int d, int y){
 return
  1461 * (y + 4800 + (m - 14) / 12) / 4 +
  367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
  3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
  d - 32075;
}
// converts integer (Julian day number) to
    Gregorian date: month/day/year
void intToDate (int jd, int &m, int &d, int &y){
 int x, n, i, j;
 x = jd + 68569;
 n = 4 * x / 146097;
 x -= (146097 * n + 3) / 4;
 i = (4000 * (x + 1)) / 1461001;
 x -= 1461 * i / 4 - 31;
 j = 80 * x / 2447;
 d = x - 2447 * j / 80;
 x = j / 11;
 m = j + 2 - 12 * x;
 y = 100 * (n - 49) + i + x;
}
```

6.6 Divide and Conquer Trick

```
//Divide and Conquer Trick
//Solves dp[i][j] = min k < j {dp[i - 1][k] +
    C[k][j]}
//Think about the base cases and set them on main
//The most important part is to set the method
    for getval
//Hint: Should be some dp or segment tree

void go(ll i, ll l, ll r, ll optl, ll optr){
      if(l >= r) return;
   ll m = (l + r) >> 1;
   ll opt = -1;
   dp[i][m] = -1;
   FER(k, optl, min(optr, m) + 1){
       ll c = dp[i-1][k] + getval(k + 1, m + 1);
       if(c > dp[i][m]){
           dp[i][m] = c; opt = k;
       }
   }
   go(i, l, m, optl, opt);
   go(i, m + 1, r, opt, optr);
}

int main(){
   FER(i, 0, n) dp[0][i] = //base cases!;
       FER(i, 1, m) go(i, 0, n , 0, n);
       cout << dp[m - 1][n - 1] << endl;
       return 0;
}
```

6.7 Fractions

```
struct Frac{
      ll a,b;
      Frac(){}
      Frac(ll _a, ll _b) : a(_a), b(_b){
            if(b < 0) a = -a, b = -b;
      }
      void toZero(){ a = 0; }
      long double getReal(){ return (long
          double)a / b; }
      Frac operator * (ll x){ return
          Frac(a*x,b); }
      Frac operator + (Frac other){
            return Frac(a*other.b + b *
                other.a,b * other.b);
      }
      bool positive(){ return a > 0;}
      bool negative(){ return a < 0;}
      bool operator == (Frac &other){
            return a * other.b == b * other.a;
      }
      bool operator < (const Frac &other) const{
            return a * other.b < b * other.a;
      }
      bool operator > (const Frac &other) const{
            return a * other.b > b * other.a;
      }
      bool operator <= (const Frac &other) const{
            return a * other.b <= b * other.a;
      }
      bool operator >= (const Frac &other) const{
            return a * other.b >= b * other.a;
      }
};
```

6.8 IO formatting

```
// Fast number input
#define getcx getchar_unlocked
inline void read(int &number) {
   number = 0;
   int ch = getcx();
   while (ch < '0' || ch > '9')
      ch = getcx();
   while(ch >= '0' && ch <= '9'){
      number = (number << 3) + (number << 1) + ch
          - '0';
      ch = getcx();
   }
}

int main(){
   cout << setprecision(5) << fixed; // Amount of
       decimal
   cout.setf(ios::showpoint); // Show decimal
       point
   cout.setf(ios::showpos); // Show + in positiv
   cout << hex << dec << endl; // Hexadecimal /
       Normal
}
```

6.9 Longest Increasing Subsequence

```
// Simple O( nlogn ) Longest Increasing
    Subsequence
// Answer is stored in array b[N]

int LIS( vi &a ){
   int b[N];
   int sz = 0;
   FER(i,0,a.size()){
      int j = lower_bound( b , b + sz , a[ i ] ) -
          b;
      // (lower) a < b < c
      // (upper) a <= b <= c
      b[ j ] = a[ i ];
      if( j == sz ) sz++;
   }
   return sz;
}
```

6.10 MinRotation

```
// finds the point to rotate to make s has the
    smallests values first
int min_rotation(vl s) {
       int a=0, N=sz(s);
       s.insert(s.end(),all(s));
       FOR(b,N) FOR(i,N) {
              if (a+i == b || s[a+i] < s[b+i]) {b
                  += max(0, i-1); break;}
              if (s[a+i] > s[b+i]) { a = b;
                  break; }
       }
       return a;
}

// rotates v
vl rot_min(vl v) {
    int x = min_rotation(v);
    rotate(v.begin(),v.begin()+x,v.end());
    return v;
}
```

6.11 Mo algorithm

```
map<ll , ll> m;
ll z;
inline ll OnlineCom(ll x){
   return m.count(x)? m[x]: m[x]=++z;
}

struct Que{
   ll l, r, t, id, lq, rq;
   Que(){}
   Que(ll l, ll r, ll t, ll id, ll lq, ll rq) :
       l(l), r(r), t(t), id(id), lq(lq), rq(rq){}
}Q[1<<18];

struct Upd{
   ll p, last, nxt;
   Upd(){}
   Upd(ll p, ll last, ll nxt) : p(p), last(last),
       nxt(nxt){}
}U[1<<18];

struct MO{
   ll n, curR, curL, q, t;
   ll ans[1<<18], ar[1<<18], cnt[1<<18],
       fre[1<<18];
   MO(){}
   inline void Add(ll val){
      fre[cnt[val]]--;
      fre[++cnt[val]]++;
   }
   inline void Rem(ll val){
      fre[cnt[val]]--;
      fre[--cnt[val]]++;
   }
   inline void Retime(ll i, ll cur){
      if(i>=curL && i<curR){
         Add(cur);
         Rem(ar[i]);
      }
      ar[i]=cur;
   }
   inline void Query(ll l, ll r, ll timer, ll id){
      while(t<timer) Retime(U[t].p, U[t].nxt), t++;
      while(t>timer) t--, Retime(U[t].p,
          U[t].last);
      while(curL<l) Rem(ar[curL++]);
      while(curL>l) Add(ar[--curL]);
      while(curR>r) Rem(ar[--curR]);
      while(curR<r) Add(ar[curR++]);
      ll mex=1;
      while(fre[mex++]);
      ans[id]=mex-1;
   }
   inline void build(){
      sort(Q, Q+q, [&](Que &a, Que &b){
         if(a.lq!=b.lq) return a.lq<b.lq;
         if(a.rq!=b.rq) return a.rq<b.rq;
```

```
        return a.t<b.t;
    }
    );
    curL=0, curR=0;
    FER(i,0,q) Query(Q[i].l, Q[i].r, Q[i].t,
        Q[i].id);
  }
}mo;

int main(){
  fastio;
  ll n, q, add; cin>>n>>q;
  add=(ll) pow(n, 2.0/3.0);
  mo.n=n;
  FER(i,0,n) {
    cin>>mo.ar[i];
    mo.ar[i]=OnlineCom(mo.ar[i]);
  }
  ll type, l, r, t1=0, t2=0;
  FER(i,0,q){
    cin>>type>>l>>r; l--;
    if(type==1){
      Q[t1]={l, r, t2, t1, l/add, (r-1)/add};
      t1++;
    }
    else{
      U[t2].p=l, U[t2].nxt=OnlineCom(r);
      t2++;
    }
  }
  mo.q=t1, mo.t=t2;
  FER(i,0,t2){
    U[i].last=mo.ar[U[i].p];
    mo.ar[U[i].p]=U[i].nxt;
  }
  mo.build();
  FER(i,0,t1) cout<<mo.ans[i]<<endl;
  return 0;
}
```

6.12 Parallel Binary Search

```
// Paralel Binary Search
// Key idea is to mantain a node with the range
    and depth
// then use a queue mantaining a ni current nivel
    and nt current index
// in a grid, given heights of a the mountain,
    find the smallest height that
// are in a path from 2 given coordenates.
// i.e. Mountainers cf gym 102021 M

ll val[1<<19];
ll nn, mm;

struct node{
    ll l, r, d;
    vi v;
    node(){}
    node(ll l, ll r, ll d): l(l), r(r), d(d){}
    inline void clear(){
        l=r=d=0;
        v.clear();
    }
};
struct Kruskal{
    ll n, id[1<<19];
    vector<tri> v;
    inline void build(){
        FER(i,0,n) id[i]=i;
    }
    inline ll find(ll x){
        while(id[x]!=x) id[x]=id[id[x]],
            x=id[x];
        return x;
    }
    inline void unir(ll a, ll b){
        ll p=find(a), q=find(b);
        id[p]=id[q];
    }
    inline ll go(ll lim, ll idx){
        FER(i,idx,sz(v)){
            ll w=v[i].tm1, a=v[i].tm2,
                b=v[i].tm3;
            if(w>=lim){
                return i;
            }
            ll node=a*mm+b;
            FER(t1, -1, 2) FER(t2, -1,
                2){
                if(abs(t1)+abs(t2)!=1)
                    continue;
                ll x=a+t1, y=b+t2;
                if(x>=0 && x<nn &&
                    y>=0 && y<mm){
                    ll cur=x*mm+y;
                    if(val[cur]<lim){
                        unir(cur,
                            node);
                    }
                }
            }
        }
        return sz(v);
    }
}kr;

ii queries[1<<19];
ll ans[1<<19];

int main(){
    fastio;
    ll n, m, k; cin>>n>>m>>k;
    nn=n, mm=m;
    kr.n=n*m+1;
    FER(i,0,n){
        FER(j,0,m) {
            ll w; cin>>w;
            ll node=i*m+j;
            val[node]=w;
            kr.v.pb({w, {i, j}});
        }
    }
    kr.build();
    sort(all(kr.v));
    ll ini=0;
    FER(i,0,k){
        ll x1, y1; cin>>x1>>y1; x1--, y1--;
        ll node1=x1*m+y1;
```

```
        ll x2, y2; cin>>x2>>y2; x2--, y2--;
        ll node2=x2*m+y2;
        if(node1==node2){
                ans[i]=val[node1];
                continue;
        }
        queries[i]={node1, node2};
}
node cur, cur1, cur2;
cur.l=0, cur.r=1e9, cur.d=0;
FER(i,0,k) if(!ans[i]) cur.v.pb(i);
queue<node> q;
q.push(cur); cur.clear();
ll ni=-1, nt=0;
while(sz(q)){
        cur=q.front(); q.pop();
        if(cur.d!=ni) {
                kr.build();
                ni=cur.d, nt=0;
        }
        ll mid=(cur.l+cur.r)>>1;
        nt=kr.go(mid, nt);
        if(cur.l+1==cur.r){
                nt=kr.go(cur.r, nt);
                for(auto xd: cur.v){
                        ll a=queries[xd].ff,
                            b=queries[xd].ss;
                        ll p1=kr.find(a),
                            p2=kr.find(b);
                        (p1==p2)?
                            ans[xd]=mid:
                            ans[xd]=-1;
                }
                continue;
        }
        for(auto xd: cur.v){
                ll a=queries[xd].ff,
                    b=queries[xd].ss;
                ll p1=kr.find(a),
                    p2=kr.find(b);
                (p1==p2)? cur1.v.pb(xd):
                    cur2.v.pb(xd);
        }
                cur1.l=cur.l, cur1.r=mid,
                    cur1.d=cur.d+1;
                cur2.l=mid, cur2.r=cur.r,
                    cur2.d=cur.d+1;
                if(sz(cur1.v)) q.push(cur1);
                if(sz(cur2.v)) q.push(cur2);
                nt=kr.go(cur.r, nt);
                cur.clear(), cur1.clear(),
                    cur2.clear();
        }
        FER(i,0,k){
                assert(ans[i]>=0);
                cout<<ans[i]<<endl;
        }
        return 0;
}
```

6.13 Sack Guni

```
//Sack Guni DS
//Solves Lomsat Gelral:
//Sum of all dominating colors in the subtree of
    vertex v
//Color c is dominating in the subtree of vertex v
//if there are no other colors that appear in the
    subtree
//of vertex v more times than color c

ll tsz[1<<17], d[1<<17], big[1<<17], col[1<<17];
ll sum[1<<17], cnt[1<<17], fre[1<<17], val,
    ans[1<<17];
vi adj[1<<17];

struct ST{
        ll n, t[1<<19];
        inline ll Op(ll &val1, ll &val2){
                return val1+val2;
        }
        inline void Upd(ll p, ll val, ll id, ll l,
            ll r){
                if(l>p || r<=p) return;
                if(l+1==r && l==p){
                        t[id]+=val;
                        return;
                }
                ll mid=(l+r)>>1;
                Upd(p, val, id<<1, l, mid);
                Upd(p, val, id<<1|1, mid, r);
                t[id]=Op(t[id<<1], t[id<<1|1]);
        }

        inline ll find(ll id, ll l, ll r){
                if(l+1==r && t[id]!=0) return t[id];
                ll valor=t[id<<1|1], mid=(l+r)>>1;
                if(!valor) return find(id<<1, l,
                    mid);
                return find(id<<1|1, mid, r);
        }

        inline void build() { fill(t, 0);}
        inline void upd(ll p, ll val) { Upd(p,
            val, 1, 0, n);}
        inline ll que() { return find(1, 0, n);}
}st;

inline void add(ll v, ll p, ll x){
        ll valor=fre[col[v]];
        st.upd(valor, -col[v]);
        fre[col[v]]+=x;
        valor=fre[col[v]];
        st.upd(valor, col[v]);
        for(auto xd: adj[v]) if(xd!=p && !big[xd])
            add(xd,v,x);
}


inline void dfs(ll u, ll pp, ll depth){
        d[u]=depth, tsz[u]=1;
        for(auto xd: adj[u]) if(xd!=pp) dfs(xd, u,
            depth+1), tsz[u]+=tsz[xd];
}

inline void go(ll u, ll p, ll x){
        ll w=-1, bc=-1;
        for(auto xd: adj[u]) if(xd!=p &&
            tsz[xd]>w) w=tsz[xd], bc=xd;
```

```
        for(auto xd: adj[u]) if(xd!=bc && xd!=p)
            go(xd, u, 0);
        if(bc!=-1) go(bc, u, 1), big[bc]=1;
        add(u, p, 1);
        ans[u]=st.que();
        if(bc!=-1) big[bc]=0;
        if(x==0) add(u, p, -1);
}


int main(){
        fastio;
        ll n; cin>>n;
        st.n=n;
        FER(i, 1, n+1) cin>>col[i];
        FER(i, 0, n-1){
                ll a, b; cin>>a>>b; //a--, b--;
                adj[a].pb(b);
                adj[b].pb(a);
        }
        st.build();
        dfs(1, -1, 0);
        go(1, -1, 1);
        FER(i, 1, n+1) cout<<ans[i]<<" ";
        cout<<endl;
        return 0;
}
```

6.14 Stable Marriage

```
// Stable Marriage Problem
#define N 505
int n;
int pref_men[N][N], pref_women[N][N], inv[N][N];
int cont[N], husband[N], wife[N];
void stable_marriage(){
   FER(i,0,n) FER(j,0,n) inv[i][pref_women[i][j]]
       = j;
   clr(cont, 0);
   clr(husband, -1);
   int m, w, dumped;
   FER(i,0,n){
      m = i;
      while(m >= 0){
         while(1){
            w = pref_men[m][cont[m]];
            ++cont[m];
            if(husband[w] < 0 || inv[w][m] <
                inv[w][husband[w]])
               break;
         }
         dumped = husband[w];
         husband[w] = m;
         wife[m]= w;
         m = dumped;
      }
   }
}
```

6.15 Unordered Map

```
unordered_map<int,int> mp;
mp.reserve(1024); // power of 2 is better
mp.max_load_factor(0.25); // 0.75 used in java
```

7 Strings

7.1 Aho Corasick - Iterative

```
//Iterative Aho - Corasick

#define N 1050
#define MOD 1000003

inline char f(char ch){
   if(ch >= 'a' and ch <= 'z') return ch - 'a';
   if(ch >= 'A' and ch <= 'Z') return ch - 'A' +
       26;
   return ch - '0' + 52;
}

struct AhoCorasick{
   int cnt;
   char pch[N];
   int t[N][27], gt[N][27],link[N],slink[N],p[N];
   int leaf[N];

   AhoCorasick(){
      cnt = 1;
      p[0] = 0;
      fill(t,-1);
      fill(gt,-1);
      fill(link,-1);
      fill(slink,-1);
      fill(leaf,0);
   }

   void add_string(string &s, int id){
      int u = 0;
      for(char ch : s){
         int c = f(ch);
         if(t[u][c] == -1){
            t[u][c] = cnt++;
            p[cnt-1] = u;
         }
         u = t[u][c];
         pch[u] = ch;
      }
      leaf[u] = 1;
   }

   int get_link(int v){
      if(link[v] == -1){
         if(!v or !p[v]) link[v] = 0;
         else link[v] = go(get_link(p[v]),pch[v]);
      }
      return link[v];
   }

   int go(int v, char ch){
      int c = f(ch);
      if(gt[v][c] == -1){
         if(t[v][c] != -1) gt[v][c] = t[v][c];
         else gt[v][c] = (v == 0) ? 0 :
             go(get_link(v),ch);
```

```
    }
    return gt[v][c];
  }

  int get_superlink(int u){
    if(slink[u] == -1){
      int lk = get_link(u);
      if(lk == 0) slink[u] = 0;
      else if(leaf[lk]) slink[u] = lk;
      else slink[u] = get_superlink(lk);
    }
    return slink[u];
  }
};
```

7.2 Aho Corasick - Phibrain

```
struct T{
  ll nxt[1<<6], fail, cnt;
  inline void clear(){
    fill(nxt, 0);
    fail=cnt=0;
  }
};

inline ll Find(char &s){
  if(s>='a' and s<='z') return s-'a';
  if(s>='A' and s<='Z') return s-'A'+26;
  if(s>='0' and s<='9') return s-'0'+52;
  return 62;
}

struct AhoCorasick{
  vector<T> t;
  T nil;
  ll id, nod;
  vi End, Q, ans, cnt;
  vector<string> set;
  inline void init(){
    nod=0;
    t.clear();
    End.clear();
    Q.clear();
    ans.clear();
    cnt.clear();
    nil.clear();
    set.clear();
    t.pb(nil);
    End.pb(-1);
    Q.pb(-1);
    cnt.pb(0);
  }

  inline void Add(string &st){
    ll cur=0, idx;
    for(auto xd: st){
      idx=Find(xd);
      if(!t[cur].nxt[idx])
          t[cur].nxt[idx]=++nod, t.pb(nil),
          cnt.pb(0);
      cur=t[cur].nxt[idx];
    }
    End.pb(cur);
    cnt[cur]++;
    set.pb(st);
  }
  inline void Build(){
    id=1;
    FER(i,0,1<<6) if(t[0].nxt[i])
        Q.pb(t[0].nxt[i]);
    while(id<=sz(Q)-1){
      ll u=Q[id++];
      cnt[u]+=cnt[t[u].fail];
      FER(i,0,1<<6){
        ll v=t[u].nxt[i];
        if(!v){
          t[u].nxt[i]=t[t[u].fail].nxt[i];
          continue;
        }
        Q.pb(v);
        t[v].fail=t[t[u].fail].nxt[i];
      }
    }
  }

  inline void AddText(string &st){
    ll cur=0, idx;
    for(auto xd: st){
      idx=Find(xd);
      cur=t[cur].nxt[idx];
      t[cur].cnt++;
    }
  }

  inline void PushAll(){
    ans.resize(sz(End));
    IFR(i, sz(Q)-1, 1)
        t[t[Q[i]].fail].cnt+=t[Q[i]].cnt;
    FER(i, 1, sz(ans)) ans[i]=t[End[i]].cnt;
  }

  inline ll Get(string &st){
    ll cur=0, idx, ans=0, n;
    for(auto xd: st){
      idx=Find(xd);
      cur=t[cur].nxt[idx];
      ans+=cnt[cur];
    }
    return ans;
  }
}ac[20][2];


inline void Upd(ll n, ll idx, string &s){
  ll id=-1;
  FER(i, 0, 20){
    ll val=(n>>i)&1;
    if(!(val)) {id=i; break;}
  }
  ac[id][idx].init();
  FER(i, 0, id){
    for(auto xd: ac[i][idx].set)
        ac[id][idx].Add(xd);
    ac[i][idx].init();
  }
  ac[id][idx].Add(s);
  ac[id][idx].Build();
}
```

```
inline void Query(string &s){
   ll ans=0;
   FER(i, 0, 20){
      if(sz(ac[i][0].set)) ans+=ac[i][0].Get(s);
      if(sz(ac[i][1].set)) ans-=ac[i][1].Get(s);
   }
   cout<<ans<<endl;
   cout.flush();
}
int main(){
   fastio;
   ll q, t, tot1=0, tot2=0; cin>>q;
   string s;
   while(q--){
      cin>>t>>s;
      if(t==1){
         Upd(tot1, 0, s);
         tot1++;
      }
      else if(t==2){
         Upd(tot2, 1, s);
         tot2++;
      }
      else{
         Query(s);
      }
   }
   return 0;
}
```

7.3 KMP

```
// KMP + KMP Automata

void kmp(string &cad){
   p[0] = 0;
   for(int i = 1; i < m; ++i){
      p[i] = p[i-1];
      while(p[i] > 0 and cad[p[i]] != cad[i])
         p[i] = p[p[i]-1];
      if(cad[p[i]] == cad[i]) p[i]++;
   }
}

int go[N][300];

vector<int> kmp(string &s){
   int n = sz(s);
   vector<int> p(n);
   p[0] = 0;
   for(int i = 1; i < n; ++i){
      p[i] = p[i-1];
      while(p[i] and s[p[i]] != s[i]) p[i] =
          p[p[i]-1];
      if(s[p[i]] == s[i]) p[i]++;
   }
   return p;
}

void process(string &s){
   int n = sz(s);
   vector<int> vec = kmp(s);
   fill(go,0);
   for(int i = 0; i < n; ++i) go[i][s[i]] = i+1;
   for(int i = 1; i <= n; ++i)
      for(char j = 'a'; j <= 'z'; ++j){
         if(go[i][j]) continue;
         go[i][j] = go[vec[i-1]][j];
      }
}

void search(string &pat, string &text){
   int cnt = 0;
   vector<int> vec = kmp(pat);
   for(char c : text){
      while(cnt and c != pat[cnt]) cnt =
          vec[cnt-1];
      if(c == pat[cnt]) cnt++;
      if(cnt == sz(pat)){
         // pattern found
         cnt = vec[cnt-1];
      }
   }
}
```

7.4 Manacher

```
// O(n) Manacher's algorithm for finding all
    palindromes
int n;
char s[200200];
char aux[100100];
int p[200200];

int main(){
   scanf("%s%n", aux, &n);
   s[0] = '^';
   s[1] = '#';
   FER(i,0,n){
      s[2*i+2] = aux[i];
      s[2*i+3] = '#';
   }
   s[2*n+2] = '\0';
   int c = 0, r = 0;
   FER(i,0,2*n+2){
      if (i > r) p[i] = 0;
      else p[i] = min(r-i, p[2*c-i]);
      while (s[i+p[i]+1] == s[i-p[i]-1]) p[i]++;
      if (i + p[i] > r){
         c = i;
         r = i + p[i];
      }
   }

   printf("%s\n", s);
   FER(i,0,2*n+2) {
      printf("%d", p[i]);
   }
   printf("\n");
   return 0;
}
```

7.5 Palindromic Tree

```
// adamant's palindromic tree online
    O(n*log(|E|)) construction
```

```
// Tutorial: http://adilet.org/blog/25-09-14/
// Add/Delete operation can be supported in
    O(logn) by doing
// check(link[v]), v = slink[v] in get_link
// (periodicity -> same initial char)
const int maxn = 5e5, sigma = 26, INF = 1e9;
int s[maxn], len[maxn], link[maxn],
    to[maxn][sigma];
int n, last, sz;
// All these optional (palindromic factoring)
int d[maxn], slink[maxn], dpe[maxn], dpo[maxn];
int anse[maxn], anso[maxn], prve[maxn],
    prvo[maxn];

void init(){ // Call with n=0
  s[n++] = -1;
  link[0] = 1;
  len[1] = -1;
  sz = 2;
  anse[0] = 0;
  anso[0] = INF;
}

int get_link(int v){
  while(s[n - len[v] - 2] != s[n - 1]) v =
      link[v];
  return v;
}

ii getmin(int v, int* ans, int* dp, int* prv){
  dp[v] = ans[n - (len[slink[v]] + d[v]) - 1];
  int best = n - (len[slink[v]] + d[v]) - 1;
  if (d[v] == d[link[v]]){
    if (dp[v] > dp[link[v]]){
      dp[v] = dp[link[v]];
      best = prv[n-1-d[v]];
    }
  }
  return mp(dp[v] + 1, best);
}

void add_letter(int c){
  s[n++] = c;
  last = get_link(last);
  if(!to[last][c]) {
    len [sz] = len[last] + 2;
    link[sz] = to[get_link(link[last])][c];
    d[sz] = len[sz] - len[link[sz]];
    if (d[sz] == d[link[sz]]) slink[sz] =
        slink[link[sz]];
    else slink[sz] = link[sz];
    to[last][c] = sz++;
  }
  last = to[last][c];

  anse[n-1] = INF;
  for (int v = last; len[v] > 0; v = slink[v]){
    ii acte = getmin(v, anso, dpe, prve);
    if (act.fst < anse[n-1]){
      anse[n-1] = act.fst;
      prve[n-1] = act.snd;
    }
  }

  anso[n-1] = INF;
  for (int v = last; len[v] > 0; v = slink[v]){
    ii act = getmin(v, anse, dpo, prvo);
    if (act.fst <= anso[n-1]){
      anso[n-1] = act.fst;
      prvo[n-1] = act.snd;
    }
  }
}
```

7.6 Suffix Array

```
//Suffix array, fast enough for 1e5
// p[i]: order of the suffix starting at i
// r: Suffix array itself (indexes of ordered
    starts of suffixes)
// h: kasai array

struct SA{
 ll n,t;
 ll p[N], r[N], h[N];
 string s;
 ll rmq[M][N], flog2[N];
 inline void fix_index(ll b, ll e){
  ll lastpk, pk, d;
  lastpk=p[r[b]+t];
  d=b;
  FER(i,b,e){
   if(((pk=p[r[i]+t]) !=lastpk) && (b>lastpk ||
       pk>=e)){
    lastpk = pk;
    d=i;
   }
   p[r[i]]=d;
  }
 }
 inline void suf_arr(){
  s[n++]='$';
  ll bc[256];
  FER(i,0,256) bc[i]=0;
  FER(i,0,n) bc[(ll)s[i]]++;
  FER(i,1,256) bc[i]+=bc[i-1];
  IFR(i,n-1,0) r[--bc[(ll)s[i]]]=i;
  IFR(i,n-1,0) p[i]=bc[(ll)s[i]];
  for(t=1;t<n;t<<=1){
   for(ll i=0,j=1;i<n;i=j++){
    while(j<n && p[r[j]]==p[r[i]]) ++j;
    if(j-i>1){
     sort(r+i,r+j, [&](const ll &i, const ll
         &j){return p[i+t]<p[j+t];});
     fix_index(i,j);
    }
   }
  }
 }
 inline void initlcp(){
  ll tam=0,j;
  FER(i,0,n-1){
   j=r[p[i]-1];
   while(s[i+tam]==s[j+tam]) ++tam;
   h[p[i]-1]=tam;
   if(tam>0) --tam;
  }
 }
 inline void makelcp(){
  initlcp();
```

```
   FER(i,0,n-1) rmq[0][i]=h[i];
   ll lg=0,pw=1;
   do{
     FER(i,pw,2*pw) flog2[i]=lg;
     lg++;pw*=2;
     FER(i,0,n-1){
      if(i+pw/2 < n-1)
          rmq[lg][i]=min(rmq[lg-1][i],
          rmq[lg-1][i+pw/2]);
      else rmq[lg][i]=rmq[lg-1][i];
     }
   } while(pw<n);
  }
  inline ll lcp(ll i, ll j){
   if(i==j) return n-r[i]-1;
   ll lg=flog2[j-i], pw=(1<<lg);
   return min(rmq[lg][i], rmq[lg][j-pw]);
  }
  inline void build(){
   fill(p,0);
   fill(r,0);
   fill(h,0);
   fill(rmq,0);
   fill(flog2,0);
   n=sz(s);
   suf_arr();
   makelcp();
  }
}sa;
```

7.7 Suffix Automaton

```
// O(n) Online suffix automaton construction
// len[u]: Max length of a string accepted by u
// link[u]: Suffix link of u
// Link edges give the suffix tree of reverse(s)
// Terminal nodes can be obtained by
//    traversing last's links

const int MAX = 1000000;
int len[MAX*2];
int link[MAX*2];
map<char,int> adj[MAX*2];
int sz, last;

// To reuse, clear adj[]
void sa_init() {
   sz = last = 0;
   len[0] = 0;
   link[0] = -1;
   sz++;
}

void sa_extend (char c) {
   int cur = sz++;
   len[cur] = len[last] + 1;
   int p;
   for (p=last; p!=-1 && !adj[p].count(c); p =
       link[p])
      adj[p][c] = cur;
   if (p == -1)
      link[cur] = 0;
   else {
      int q = adj[p][c];
      if (len[p] + 1 == len[q])
         link[cur] = q;
      else {
         int clone = sz++;
         len[clone] = len[p] + 1;
         adj[clone] = adj[q];
         link[clone] = link[q];
         for (; p != -1 && adj[p][c] == q; p =
             link[p])
            adj[p][c] = clone;
         link[q] = link[cur] = clone;
      }
   }
   last = cur;
}
```

7.8 Z-Algorithm

```
//Zfun(i) devuelve la longitud del maximo prefijo
    que empieza en i
vi Zfun(string s){
   vi Z(s.sz,0);
   int l = 0, r = 0;
   FER(i,1,sz(s)){
      if ( i<=r ) Z[i] = min(Z[i-l], r-i+1);
      while ( i+Z[i]<s.sz and s[i+Z[i]]==s[Z[i]] )
          Z[i]++;
      if ( i+Z[i]-1>r ) l = i, r = i+Z[i]-1;
   }
   return Z;
}
```

8 Templates

8.1 Makefile

```
CXX = g++
CXXFLAGS = -std=c++14 -Wall -Wextra
    -Wno-sign-compare -O2 -g

all: %
%: %.cpp
   $(CXX) $(CXXFLAGS) -o $@ $@.cpp
```

8.2 Random Generator

```
//Mersenne Twister Random Generator
//Note: If you want 64-bit random numbers,
//    just declare it as mt19937_64.

mt19937 rng(chrono::steady_clock::now().
        time_since_epoch().count());
```

8.3 Script

```
# Gracias men de la uni
```

```
echo "Compiling gen.cpp"
g++ gen.cpp -std=c++14 -o gen
echo "Compiling ac.cpp"
g++ ac.cpp -std=c++14 -o ac
echo "Compiling main.cpp"
g++ main.cpp -std=c++14 -o main

((count = 0))
((testCases = 1000))
for((i = 1; ; i++)); do
   echo "Running on test ${i}"
   ./gen $i > test
   ./ac < test > out1
   ./main < test > out2
   diff -w out1 out2 || break
   ((count++))
done

if [ $count -eq $testCases ]
then
   echo "Accepted"
else
   ((count++))
   echo "Wrong Answer on test ${count}"
   more test
   echo "Answer : "
   more out1
   echo "Output : "
   more out2
fi
```

8.4 Stack Size

```
#include <sys/resource.h>

int main (int argc, char **argv){
   const rlim_t kStackSize = 64L * 1024L * 1024L;
       // min stack size = 64 Mb
   struct rlimit rl;
   int result;
   result = getrlimit(RLIMIT_STACK, &rl);
   if (result == 0)
   {
      if (rl.rlim_cur < kStackSize)
      {
         rl.rlim_cur = kStackSize;
         result = setrlimit(RLIMIT_STACK, &rl);
         if (result != 0)
         {
            fprintf(stderr, "setrlimit returned
                result = %d\n", result);
         }
      }
   }

   // ...

   return 0;
}
```

8.5 Template

```
#pragma GCC optimize ("Ofast,unroll-loops")
#pragma GCC target ("sse,sse2,sse3,ssse3,sse4")
#pragma GCC target
    ("popcnt,abm,mmx,avx,tune=native")

#include <bits/stdc++.h>
using namespace std;
#define sz(x) int(x.size())
#define pb push_back
#define FER(i,a,b) for(ll i = ll(a); i < ll(b);
    ++i)
#define IFR(i,a,b) for(ll i = ll(a); i >= ll(b);
    --i)
#define all(v) (v).begin(),(v).end()
#define mp make_pair
#define ff first
#define ss second
#define tm1 first
#define tm2 second.first
#define tm3 second.second
#define fill(x,v) memset(x,v,sizeof(x))
#define fastio
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define sqr(x) (x)*(x)
#define bas 987625403
#define N 100010
typedef long double ld;
typedef long long ll;
typedef pair<ll,ll> ii;
typedef pair<ll,ii> tri;
typedef vector<ll> vi;
typedef vector<ii> vii;
#define trace(...) fff(#__VA_ARGS__,__VA_ARGS__)

template<typename t> void fff(const char *x, t&&
    val1){
   cout << x << " : " << val1 << endl;
}

template<typename t1, typename... t2> void
    fff(const char *x, t1&& val1, t2&&... val2){
   const char *xd = strchr(x+1,',');
   cout.write(x,xd-x)<< " : " << val1 << " | ";
   fff(xd+1,val2...);
}

int main(){
   fastio;
   return 0;
}
```

8.6 Vim Configuration (vimrc)

```
set number
set autoindent
set ts=4
set mouse=a
```