Contents

1 Aho Corasic

```
///aho corasick///
namespace AC{
  const int AlphabetSize=26;
  struct Vertex{
    int next[AlphabetSize];
    int go[AlphabetSize];
    int leaf=-1;
    int p=-1;
    char pch;
    int elink=-1;//link to the maximum proper suffix of
        current string
    int vlink=-1;//link to the maximum proper suffix of
        current string which is also a string in the
        given set
    Vertex(int p=-1,char pch='$'):p(p), pch(pch){
      memset(next,-1,sizeof(next));
      memset(go,-1,sizeof(go));}
    void reset(){
      memset(next,-1,sizeof(next));
      memset(go,-1,sizeof(go));
      leaf=p=elink=vlink=-1;}};
  const int MaxLength=1e6;
  Vertex trie[MaxLength];
  int curSize=1;
  //returns identifier of the string currently at the
      end vertex of s
  //id should be unique identifier of the string
  int addString(const string& s,int id){
    int v=0;
    for(char ch:s){
      int c=ch-'a';
      if(trie[v].next[c]==-1){
        trie[curSize].p=v;
        trie[curSize].pch=ch;
        trie[v].next[c]= curSize++;}
      v=trie[v].next[c];}
    if(trie[v].leaf==-1)
      trie[v].leaf=id;
    return trie[v].leaf;}
  int addString(char* s,int id){
    int v=0;
    while(*s!='\0'){
      int c=*s-'a';
      if(trie[v].next[c]==-1){
        trie[curSize].p=v;
        trie[curSize].pch=*s;
        trie[v].next[c]= curSize++;}
```

```
    v=trie[v].next[c];
    ++s;}
  if(trie[v].leaf==-1)
    trie[v].leaf=id;
  return trie[v].leaf;}
//should be called after adding all string to the trie
void constructAutomation(void){
  trie[0].elink=0;
  //assume that empty string always exist in the set
  trie[0].vlink=0;
  queue<int> Q;
  for(int i=0;i<AlphabetSize;++i){
    trie[0].go[i]= (trie[0].next[i]==-1?0:
        trie[0].next[i]);
    if(trie[0].next[i]==-1){
      trie[0].go[i]=0;}
    else{
      trie[0].go[i]=trie[0].next[i];
      Q.push(trie[0].next[i]);}}
  while(!Q.empty()){
    int v=Q.front();
    Q.pop();
    int p=trie[v].p;
    char pch=trie[v].pch;
    trie[v].elink= trie[trie[p].elink] .go[pch-'a'];
    if(trie[v].elink==v)
      trie[v].elink=0;
    if(trie[trie[v].elink].leaf!=-1)
      trie[v].vlink=trie[v].elink;
    else
      trie[v].vlink= trie[trie[v].elink].vlink;
    for(int i=0;i<AlphabetSize;++i){
      if(trie[v].next[i]==-1){
        trie[v].go[i]= trie[trie[v].elink].go[i];
      }
      else{
        trie[v].go[i]=trie[v].next[i];
        Q.push(trie[v].next[i]);}}}}
//implementation of function that older code used
int go(int v,char ch){
  return trie[v].go[ch-'a'];}
//implementation of function that older code used
int getLink(int v){
  return trie[v].elink;}
int nMatch[MaxLength];
const int MaxNumberOfString=1e6;
int ans[MaxNumberOfString];
//O(n+m^2) where m is the number of distinct length
//of string in the set added to the structure
void dfs(int v){
  if(trie[v].leaf!=-1){
    int vv=v;
    while(vv>0){
      ans[trie[vv].leaf]+=nMatch[v];
      vv=trie[vv].vlink;}}
  for(int i=0;i<AlphabetSize;++i){
    if(trie[v].next[i]==-1)
      continue;
    dfs(trie[v].next[i]);}}
//O(n)
void reverseBFS(int v){
  stack<int> st;
  queue<int> q;
  q.push(v);
  while(!q.empty()){
    v=q.front();
    q.pop();
    if(trie[v].leaf!=-1)
      st.push(v);
    for(int i=0;i<AlphabetSize;++i){
      if(trie[v].next[i]!=-1){
        q.push(trie[v].next[i]);}}}
  while(!st.empty()){
    v=st.top();
    st.pop();
    ans[trie[v].leaf]+=nMatch[v];
    nMatch[trie[v].vlink]+=nMatch[v];}}
//compute number of occurrence of each of the string
    added to the trie
//store the result to ans[]
//call constructAutomation() before calling this
    function
//important: careful about multiple occurrence of same
    string in the trie
//trie can only hold id for 1 string
void computeAllMatch(const string& s){
  int v=0;
  for(char ch:s){
    v=trie[v].go[ch-'a'];
    if(trie[v].leaf!=-1){
      ++nMatch[v];}
    else
      ++nMatch[trie[v].vlink];}
  //dfs(0);
  reverseBFS(0);}
void reset(){
  for(int i=0;i<curSize;++i){
    trie[i].reset();
    nMatch[i]=ans[i]=0;}
  curSize=1;}}
```

2 Articulation Point

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

vector<int>cuts;
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                cuts.pb(v+1);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        cuts.pb(v+1);
}
void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}

find_cutpoints();
sort(cuts.begin(), cuts.end());
cuts.erase(unique(cuts.begin(), cuts.end()), cuts.end());
cout<<cuts.size()<<endl;
for(auto u:cuts)
cout<<u<<" ";
```

3 BIT 1D

```
//Tested : LightOJ 1164 - Horrible Queries

#include<bits/stdc++.h>
using namespace std;
#define ll long long int
const int MAXN = 1000005;

ll BIT[2][MAXN];
void update(int cs, int indx, ll val){
    while(indx < MAXN){
        BIT[cs][indx] += val;
        indx += (indx & -indx);
    }
}

ll sum(int cs, int indx){
    ll ans = 0;
    while(indx != 0) {
        ans += BIT[cs][indx];
        indx -= (indx & -indx);
    }
    return ans;
}

void updateRange(int l, int r, ll val){
    update(0,l,val);     update(0,r+1,-val);
    update(1,l,val*(l-1)); update(1,r+1,-val*r);
}

ll sumRange(int indx) {return sum(0,indx)*indx -
    sum(1,indx);}
ll QueryRange(int l, int r) {return
    sumRange(r)-sumRange(l-1);}

const int LOGN = 20;
int LowerBound(int cs, ll v){
    ll sum = 0;
    int indx = 0;
    for(int i = LOGN; i >= 0; i--){
        int nPos = indx + (1<<i);
        if(nPos < MAXN && sum + BIT[cs][nPos] < v){
            sum += BIT[cs][nPos];
            indx = nPos;
        }
    }
    //pos = maximal x such that Sum(x) < v
    return indx + 1; //+1 for LowerBound
}
```

4 BIT 2D

```
#define ll long long int
const int MAXN = 1005;
struct BIT2D{
    ll tree[4][MAXN][MAXN];
    BIT2D() {memset(tree,0,sizeof(tree));}
    //Add v to submatrix [i,j] to [inf,inf]
    void update(int p, int q, ll v){
        if((p <= 0) || (q <= 0) || (p >= MAXN) || (q >=
            MAXN)) return;
```

```
        int i = p, c = p - 1, d = q - 1;
        while(i < MAXN){
            int j = q;
            while(j < MAXN){
                tree[0][i][j] += v;      tree[1][i][j] +=
                    (v * d);
                tree[2][i][j] += (v * c); tree[3][i][j]
                    += (v * c * d);
                j += (j & -j);
            }
            i += (i & -i);
        }
    }
    //returns Sum over submatrix [1,1] to [p,q]
    ll query(int p, int q){
        int i, j;
        ll x, y, z, c, d, res;
        i = p, x = y = z = 0;
        while(i != 0){
            j = q, c = d = 0;
            while(j != 0){
                c += tree[0][i][j];      d +=
                    tree[1][i][j];
                y += tree[2][i][j];      z +=
                    tree[3][i][j];
                j ^= (j & (-j));
            }
            i ^= (i & -i);
            x += (c * q - d);
        }
        res = (x * p) - (y * q) + z;
        return res;
    }
    //Add v to submatrix [i,j] to [k,l]
    void update(int i, int j, int k, int l, ll v){
        update(i, j, v);                 update(k + 1, j,
            -v);
        update(k + 1, l + 1, v);         update(i, l + 1,
            -v);
    }
    //returns Sum over submatrix [i,j] to [k,l]
    ll query(int i, int j, int k, int l){
        if (i > k || j > l) return 0;
        ll res = query(k, l) - query(i - 1, l) + query(i
            - 1, j - 1) - query(k, j - 1);
        return res;
    }
};

BIT2D bit;
cout<<bit.query(a,b,c,d);
bit.update(a,b,c,d,v);
```

5 Berlekamp Massey

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

6 Bridge

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

vector<pii> bridges;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v]){
                //IS_BRIDGE(v, to);
                bridges.pb({v+1,to+1});
               // cout<<"bridge "<<v<<" "<<to<<endl;
            }
        }
    }
}
void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}


find_bridges();
cout<<bridges.size()<<endl;
for(auto [u,v]:bridges)
cout<<u<<" "<<v<<endl;
```

7 CRT

```
#define rng_23 mt19937 rng(chrono:: steady_clock::
    now().time_since_epoch().count())
const ll siz=1000000;
ll moduli[siz],rem[siz];
pii egcd(ll a,ll b){
  ll r0=max(a,b),r1=min(a,b),
  s0=1,s1=0,t0=0,t1=1,q,tmp;
  while(r1){
    q=r0/r1;
    r0-=r1*q,s0-=s1*q,t0-=t1*q;
    tmp=r1,r1=r0, r0=tmp;
    tmp=s1,s1=s0,s0=tmp;
    tmp=t1,t1=t0,t0=tmp;}
  if(a<b) swap(s0,t0);
  return {s0,t0};
  //r0 is GCD}
ll crt(ll n){
  ll n1, a1, m1, n2, a2, d, x; //x=n1*m1+a1=n2*m2+a2
  n1=moduli[0],a1=rem[0];
  for(ll i=1;i<n;i++){
    n2=moduli[i], a2=rem[i];
    d=a2-a1; //n1*m1-n2*m2=a2-a1=d
    m1=(egcd(n1, n2).fi * d/__gcd(n1, n2))%n2;
    x=n1*m1+a1;
    n1=n1/__gcd(n1, n2)*n2;
    a1=x%n1;
    if(a1<0)
      a1+=n1;}
  return a1;}
```

8 Centroid$_D$ecomposition

```
vector<int> tree[sz],ctree[sz];
int sub[sz],vis[sz],cparent[sz];
void dfs(int a,int p){
  sub[a]=1;
  for(auto v: tree[a]){
    if(v!=p && vis[v]==0){
      dfs(v,a);
      sub[a]+=sub[v];}}}
int findCentroid(int a,int p,int num){
  bool sig=true;
  while(sig){
    sig=false;
    for(auto v: tree[a]){
      if(v!=p && vis[v]==0 && 2*sub[v]>num){
        p=a;
        a=v;
        sig=true;
        break;}}}
return a;}
int Decompose(int a,int p,int num){
  dfs(a,p);
  int centroid= findCentroid(a,p,num);
  vis[centroid]=1;
  //printf("%d\n",centroid);
  for(auto v: tree[centroid]){
    if(vis[v]==0){
    if(sub[v]>sub[centroid])
      ctree[centroid].pb( Decompose(v,centroid,
          num-sub[centroid]));
    else
      ctree[centroid].pb( Decompose(v, centroid,sub[v]));
    cparent[ ctree[centroid].back()]= centroid;}}
  return centroid;}
```

9 Combinatorics Mashroor

```
ll invfac[1000009], fact[1000009], mod=1000000007;

void factorials(){
    invfac[0]=1;
    invfac[1] = 1;
    fact[0]=1;
    fact[1] = 1;
    for(ll i=2; i<1000009; i++) invfac[i] = (mod -
        (mod/i)*invfac[mod%i]%mod)%mod;
    for(ll i=1; i<1000009; i++){
        fact[i]=i*fact[i-1];
        invfac[i]*=invfac[i-1];
        fact[i]%=mod;
        invfac[i]%=mod;
    }
}

ll ncr(ll n, ll r){
```

```
    if(n<0 || r<0 || n<r) return 0;
    ll k=fact[n];
    k*=invfac[r]; k%=mod;
    k*=invfac[n-r]; k%=mod;
    return k;
}

ll catalan(ll r, ll c){
    if(r>c) return -1;
    if(r==0 && c==0) return 1;
    return es(ncr(r+c,c),ncr(r+c,c+1));
}
```

10 Condensation Graph

```
const int N = 500005;

vector<int> adj[N], adj_rev[N],adj_scc[N];
vector<bool> used;
vector<int> order, condensedOrder, component, root_nodes;
int sz[N];
int res[N];
int roots[N];

void dfs1(int v) {
    used[v] = true;
    for (auto u : adj[v])
        if (!used[u])
            dfs1(u);
    order.push_back(v);
}
void dfs2(int v) {
    used[v] = true;
    component.push_back(v);
    for (auto u : adj_rev[v])
        if (!used[u])
            dfs2(u);
}
void solve(){

    int n,m;
    cin>>n>>m;
    used.assign(n,false);
    int ara[n];
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++) cin>>ara[j];
        for(int j=1;j<n;j++){
            int a = ara[j-1];
            int b = ara[j];
            a--;
            b--;
            adj[a].push_back(b);
            adj_rev[b].push_back(a);
        }
    }
    for (int i = 0; i < n; i++)
        if (!used[i])
            dfs1(i);
    used.assign(n, false);
    reverse(order.begin(), order.end());

    for (auto v : order){
        if (!used[v]) {
            dfs2 (v);
            int root = component.front();
            for (auto u : component) roots[u] = root;
            root_nodes.push_back(root);
            sz[root] = component.size();
            component.clear();
        }
    }

    for (int v = 0; v < n; v++){
        for (auto u : adj[v]) {
            int root_v = roots[v],
                root_u = roots[u];

            if (root_u != root_v)
                adj_scc[root_v].push_back(root_u);
        }
    }

    // topsort of the condensed graph
    for(auto ord:order)
        if( roots[ord] == ord )
            condensedOrder.push_back(ord);
    reverse(condensedOrder.begin(),condensedOrder.end());
    // deleting multiple edges
    for(auto root:root_nodes){
        if( adj_scc[root].size()>=2 )
            sort(adj_scc[root].begin(),adj_scc[root].end());
        vector<int> nodes;
        if( adj_scc[root].size() ) nodes.push_back(
            adj_scc[root][0] );
        for(int i=1;i<adj_scc[root].size();i++){
            if( adj_scc[root][i]!=adj_scc[root][i-1] ){
                nodes.push_back(adj_scc[root][i]);
            }
        }
        adj_scc[root] = nodes;
    }

    for(auto ord:condensedOrder) {
        res[ord] = sz[ord]-1;
        for(int i=0;i<adj_scc[ord].size();i++){
            res[ord]+=res[ adj_scc[ord][i] ]+1;
        }
    }

    for(int i=0;i<n;i++){
        cout<<res[ roots[i] ]<<" ";
    }
    cout<<endl;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int tc=1;
    //cin>>tc;

    for(int i=0;i<tc;i++){
        solve();
    }
}
```

11 Convex Hull Trick

```
/*****************************************/
Fully Generalized implementation for Monotone
    slope,Arbitrary query
runtime insert() O(1) query() O(logn)
/*******************/
class MonotoneCHT{
  deque<Line> Q;
  int type;
  void insertBack(Line nl){
    //handle parallel line insertion,there cannot be
        more than one parallel line to new line
        currently inside Q;
    if(!Q.empty()&&Q.back().parallel(nl)){
      if(type<2){
        if(Q.back().c>nl.c)
          Q.pop_back();
        else return;}
      else{
        if(Q.back().c<nl.c)
          Q.pop_back();
        else return; }}
    while(Q.size()>1 && Q.back().intersect(nl) <
        Q[Q.size()-2].intersect(nl)) Q.pop_back();
    Q.push_back(nl);}
  void insertFront(Line nl){
    //handle parallel line insertion,there cannot be
    //more than one parallel line to new line currently
        inside Q;
    if(!Q.empty()&&Q[0].parallel(nl)){
      if(type<2){
        if(Q[0].c>nl.c)
          Q.pop_front();
        else return;}
      else{
        if(Q[0].c<nl.c)
          Q.pop_front();
        else return;}}
    while(Q.size()>1 && Q[0].intersect(nl)>
        Q[1].intersect(nl))
      Q.pop_front();
    Q.push_front(nl);}
  pii bSearch(ll x){
    if(Q.size()==1|| Q[0].intersect( Q[1]).first>=x)
      return {0,0};
    int l=1,r=(int)Q.size()-1;
    while(r>l+1){
      int mid=(l+r)/2;
      if(Q[mid].intersect( Q[mid-1]).first<x)
        l=mid;
      else r=mid;}
    return {l,r};}
public:
  //slope increasing or decreasing(not query point,query
      point is arbitrary),querying for maximum or minimum
  MonotoneCHT(bool increasing,bool maximum){
    type=increasing;
    if(maximum)
      type|=2;}
  void insert(Line nl){
    if(type==3||type==0)
      insertBack(nl);
    else
      insertFront(nl);}
//if monotone query satisfied //not tested,although it
    should be ok
  ll fastQuery(ll x){
    #ifdef INCREASING_QUERY
    while(Q.size()>1&&Q[0].intersect(Q[1]).first<x)
      Q.pop_front();
    return Q[0](x);
    #else
    while(Q.size()>1&& Q.back().intersect(
        Q[(int)Q.size()-2]).first>x)
      Q.pop_back();
    return Q.back()(x);
    #endif
  }
  ll query(ll x){
    pii indx=bSearch(x);
    if(type<2)
      return min(Q[indx.first](x), Q[indx.second](x));
    return max(Q[indx.first](x), Q[indx.second](x));}
  void clear(){
```

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

12 DEBUG TEMPLATE

```
void err(istream_iterator<string> it) {cout<<endl;}
template<typename T, typename... Args>
void err(istream_iterator<string> it, T a, Args... args){
 cout << *it << " = " << a << " ";err(++it, args...);
}
template<class T1, class T2>
ostream &operator <<(ostream &os, pair<T1,T2>&p) {
 os<<"{"<<p.first<<", "<<p.second<<"} ";
 return os;
}
#define debug(args...) { string _s = #args;
    replace(_s.begin(), _s.end(), ',', ' ');
    stringstream _ss(_s); istream_iterator<string>
    _it(_ss); err(_it, args); }
```

13 DSU with Rollback

```
struct dsu_save {
    int v, rnkv, u, rnku;

    dsu_save() {}

    dsu_save(int _v, int _rnkv, int _u, int _rnku)
        : v(_v), rnkv(_rnkv), u(_u), rnku(_rnku) {}
};

struct dsu_with_rollbacks {
    vector<int> p, rnk;
    int comps;
    stack<dsu_save> op;

    dsu_with_rollbacks() {}

    dsu_with_rollbacks(int n) {
        p.resize(n);
        rnk.resize(n);
        for (int i = 0; i < n; i++) {
            p[i] = i;
            rnk[i] = 0;
        }
        comps = n;
    }

    int find_set(int v) {
        return (v == p[v]) ? v : find_set(p[v]);
    }

    bool unite(int v, int u) {
        v = find_set(v);
        u = find_set(u);
        if (v == u)
            return false;
        comps--;
        if (rnk[v] > rnk[u])
            swap(v, u);
        op.push(dsu_save(v, rnk[v], u, rnk[u]));
        p[v] = u;
        if (rnk[u] == rnk[v])
            rnk[u]++;
        return true;
    }

    void rollback() {
        if (op.empty())
            return;
        dsu_save x = op.top();
        op.pop();
        comps++;
        p[x.v] = x.v;
        rnk[x.v] = x.rnkv;
        p[x.u] = x.u;
        rnk[x.u] = x.rnku;
    }
};

struct query {
    int v, u;
    bool united;
    query(int _v, int _u) : v(_v), u(_u) {
    }
};

struct QueryTree {
    vector<vector<query>> t;
    dsu_with_rollbacks dsu;
    int T;

    QueryTree() {}

    QueryTree(int _T, int n) : T(_T) {
        dsu = dsu_with_rollbacks(n);
        t.resize(4 * T + 4);
    }

    void add_to_tree(int v, int l, int r, int ul, int
         ur, query& q) {
        if (ul > ur)
            return;
        if (l == ul && r == ur) {
            t[v].push_back(q);
            return;
        }
        int mid = (l + r) / 2;
        add_to_tree(2 * v, l, mid, ul, min(ur, mid), q);
        add_to_tree(2 * v + 1, mid + 1, r, max(ul, mid +
             1), ur, q);
    }

    void add_query(query q, int l, int r) {
        add_to_tree(1, 0, T - 1, l, r, q);
    }

    void dfs(int v, int l, int r, vector<int>& ans) {
        for (query& q : t[v]) {
            q.united = dsu.unite(q.v, q.u);
        }
        if (l == r)
            ans[l] = dsu.comps;
        else {
            int mid = (l + r) / 2;
            dfs(2 * v, l, mid, ans);
            dfs(2 * v + 1, mid + 1, r, ans);
        }
        for (query q : t[v]) {
            if (q.united)
                dsu.rollback();
        }
    }

    vector<int> solve() {
        vector<int> ans(T);
        dfs(1, 0, T - 1, ans);
        return ans;
    }
};

int main(){
    freopen("connect.in","r",stdin);
    freopen("connect.out","w",stdout);
    int n,q,a,b;
    scanf("%d %d",&n,&q);
    QueryTree qt(q+1,n);
    bool isaquery[q];
    map< pii,int> e;
    for(int i=1;i<=q;i++){
        char c;
        cin>>c;
        isaquery[i] = false;
        if( c == '+' ){

            scanf("%d %d",&a,&b);
            a--;b--;
            if( a>b ) swap(a,b);
            e[mp(a,b)] = i; // marking the start of an
                 edge
        } else if( c == '-' ){
            // here the node got destroyed right? so it
                 was alive for e[{a,b}] to i-1
            scanf("%d %d",&a,&b);
            a--;b--;
            if( a>b ) swap(a,b);
            qt.add_query( query(a,b),e[mp(a,b)],i-1 );
            e.erase(mp(a,b)); // deleting the occurance
                 of this
        }else{
            isaquery[i] = true;
        }
    }
    // so now we have handled all the created and
         deleted nodes.
    // so all that lefts are nodes who remained alive
         for all the time
    for( auto it = e.begin();it!=e.end();++it ){
        qt.add_query( query( it->F.F,it->F.S ),it->S,q
             ); // remember we stored e[{a,b}] =
             position. so starts at pos and ends at q
             (always was)
    }
    vector<int> res = qt.solve();
        for(int i=1;i<=q;i++){
        if(isaquery[i])
        {
            printf("%d\n",res[i]);
        }
    }
```

```
    fclose(stdout);
    return 0;
}
```

14 Dijkstra

```
int dist[10001],vis[10001];
vector<pii> graph[10001];
void dijkstra(int src,int n){
  priority_queue<pii, vector<pii>, greater<pii>>
      cur_dist;
  for(int i=1;i<=n;i++)
    dist[i]=inf;
  for(int i=1;i<=n;i++)
    cur_dist.push( make_pair( dist[i], i));
  dist[src]=0;
  cur_dist.push(make_pair(0, src));
  while(!cur_dist.empty()){
     int centr= cur_dist.top().second;
     cur_dist.pop();
     if(vis[centre])
      continue;
    for(auto itr: graph[centre]){
      if(dist[centre]+ itr.second< dist[itr.first]){
        dist[itr.first]= dist[centre]+ itr.second;
        cur_dist.push( make_pair(dist[itr.first],
            itr.first));}}
    vis[centre]=1;}}
//floyd warshall
ll g[500][500];
void floyedWarshal(int n){
  //g[i][i] should be zero
  for(int k=0;k<n;k++){
  for(int i=0;i<n;i++){
  for(int j=0;j<n;j++){
    if(g[i][k]<LLONG_MAX&&g[k][j]<LLONG_MAX)
      g[i][j]=min(g[i][j], g[i][k]+ g[k][j]);}}}}
//bellmen-ford
int dist[10001];
struct edge{
  int a,b,cost;
};
vector<edge> graph;
void bell_ford(int src,int n){
   for(int i=1;i<=n;i++)
     dist[i]=inf;
   dist[src]=0;
   for(int i=1;i<n;i++){
     for(int j=0;j< graph.size();j++)
       dist[graph[j].b]= min(dist[graph[j].b],
           dist[graph[j].a]+ graph[j].cost);}}
```

15 Dinic

```
struct FlowEdge {
  int v, u;
  long long cap, flow = 0;
  FlowEdge(int v, int u, long long cap) : v(v), u(u),
      cap(cap) {}};
class Dinic {
  const long long flow_inf = 1e18;
  vector<FlowEdge> edges;
  vector<vector<int>> adj;
  int n, m = 0;
  int s, t;
  int *level, *ptr;
  queue<int> q;
  // 0 based index
  Dinic(int n, int s, int t) : n(n), s(s), t(t) {
    adj.resize(n);
    level=new int[n];
    ptr=new int[n];}
void add_edge(int v, int u, long long cap) {
    edges.emplace_back(v, u, cap);
    edges.emplace_back(u, v, 0);
    adj[v].push_back(m);
    adj[u].push_back(m + 1);
    m += 2;}
bool bfs() { //builds level graph
    while (!q.empty()) {
      int v = q.front();
      q.pop();
      for (int id : adj[v]) {
        if (edges[id].cap - edges[id].flow < 1)
          continue;
        if (level[edges[id].u] != -1)
          continue;
        level[edges[id].u] = level[v] + 1;
        q.push(edges[id].u);}}
    return level[t] != -1;}
long long dfs(int v, long long pushed) { //pushes flow
     through level graph
    if (pushed == 0)
      return 0;
    if (v == t)
      return pushed;
    for (int& cid = ptr[v]; cid < (int)adj[v].size();
        cid++) {
      int id = adj[v][cid];
      int u = edges[id].u;
      if (level[v] + 1 != level[u] || edges[id].cap -
          edges[id].flow < 1)
        continue;
      long long tr = dfs(u, min(pushed, edges[id].cap -
          edges[id].flow));
      if (tr == 0)
        continue;
      edges[id].flow += tr;
      edges[id ^ 1].flow -= tr;
      return tr;}
    return 0;}
long long flow() {
    long long f = 0;
    while (true) {
      memset(level, -1, sizeof level);
      level[s] = 0;
      q.push(s);
      if (!bfs()) //sink not reachable means end
        break;
      memset(ptr, 0, sizeof ptr);
      while (long long pushed = dfs(s, flow_inf)) {
        f += pushed;}}
    return f;}};
```

16 Discrete Log

```
// Returns minimum x for which a ^ x % m = b % m.
    O(sqrt(m) )
int solve(int a, int b, int m){
// if (a == 0)
//  return b == 0 ? 1 : -1;
 a %= m, b %= m;int k = 1, add = 0, g;
 while ((g = __gcd(a, m)) > 1) {
  if (b == k)return add;
  if (b % g) return -1;
  b /= g, m /= g, ++add; k = (k * 1ll * a / g) % m;
 }
 int n = sqrt(m) + 1; int an = 1;
 for (int i = 0; i < n; ++i) an = (an * 1ll * a) % m;
 unordered_map<int, int> vals;
 for (int q = 0, cur = b; q <= n; ++q) {
  vals[cur] = q; cur = (cur * 1ll * a) % m;
 }
 for (int p = 1, cur = k; p <= n; ++p) {
  cur = (cur * 1ll * an) % m;
  if (vals.count(cur)) { int ans = n * p - vals[cur] +
      add;
   return ans;
  }
 }
 return -1;
}
```

17 Enumerating All Submask

```
bitmask m,iterate through all of its submasks,
for (int s=m; ; s=(s-1)&m){
.........
if (s==0) break;
}
Iterating through all masks with their submasks
for (int m=0; m<(1<<n); ++m)
for (int s=m; s; s=(s-1)&m){
 ... s and m ...
}
```

18 Euler Tour

```
vector<multiset<int> >adj; vector<int>ans;
void euler_circuit(int src){
 stack<int>st; st.push(src);
 while(!st.empty()){
  int v = st.top();
  if(adj[v].size()==0){
   ans.push_back(v); st.pop();
  }
  else{
   int f = *adj[v].begin();
   adj[v].erase(adj[v].begin());
       adj[f].erase(adj[f].find(v));
   st.push(f);
  }
 }
}
```

19 FFT

```
/*Iterative Implementation of FFT and FFTanymod.
    Complexity: O(N log N)
1. Whenever possible remove leading zeros.
2. Custom Complex class may slightly improve performance.
3. Use pairfft to do two ffts of real vectors at once,
    slightly less accurate
than doing two ffts, but faster by about 30%.
4. FFT accuracy depends on answer. x <= 5e14 (double), x
    <= 1e18(long double)
   where x = max(ans[i]) for FFT, and x = N*mod for
       anymod*/
struct CD {
  double x, y;
  CD(double x=0, double y=0) :x(x), y(y) {}
  CD operator+(const CD& o) { return {x+o.x, y+o.y};}
  CD operator-(const CD& o) { return {x-o.x, y-o.y};}
  CD operator*(const CD& o) { return {x*o.x-y*o.y,
      x*o.y+o.x*y};}
  void operator /= (double d) { x/=d; y/=d;}
  double real() {return x;}
  double imag() {return y;}};
CD conj(const CD &c) {return CD(c.x, -c.y);}
```

```
const double PI = acos(-1.0L);
namespace FFT {
  int N;
  vector<int> perm;
  vector<CD> wp[2];
  void precalculate(int n) {
    assert((n & (n-1)) == 0);
    N = n;
    perm = vector<int> (N, 0);
    for (int k=1; k<N; k<<=1) {
      for (int i=0; i<k; i++) {
        perm[i] <<= 1;
        perm[i+k] = 1 + perm[i];}}
   wp[0] = wp[1] = vector<CD>(N);
    for (int i=0; i<N; i++) {
      wp[0][i] = CD( cos(2*PI*i/N), sin(2*PI*i/N) );
      wp[1][i] = CD( cos(2*PI*i/N), -sin(2*PI*i/N) );}}
   void fft(vector<CD> &v, bool invert = false) {
    if (v.size() != perm.size()) precalculate(v.size());
   for (int i=0; i<N; i++)
      if (i < perm[i])
        swap(v[i], v[perm[i]]);
   for (int len = 2; len <= N; len *= 2) {
      for (int i=0, d = N/len; i<N; i+=len) {
        for (int j=0, idx=0; j<len/2; j++, idx += d) {
          CD x = v[i+j];
          CD y = wp[invert][idx]*v[i+j+len/2];
          v[i+j] = x+y;
          v[i+j+len/2] = x-y;}}}
   if (invert) {
      for (int i=0; i<N; i++) v[i]/=N;}}
   void pairfft(vector<CD> &a, vector<CD> &b, bool
       invert = false) {
    int N = a.size();
    vector<CD> p(N);
    for (int i=0; i<N; i++) p[i] = a[i] + b[i] * CD(0,
        1);
    fft(p, invert);
    p.push_back(p[0]);

    for (int i=0; i<N; i++) {
      if (invert) {
        a[i] = CD(p[i].real(), 0);
        b[i] = CD(p[i].imag(), 0);}
      else {
        a[i] = (p[i]+conj(p[N-i]))*CD(0.5, 0);
        b[i] = (p[i]-conj(p[N-i]))*CD(0, -0.5);}}}

  vector<ll> multiply(const vector<ll> &a, const
      vector<ll> &b) {
    int n = 1;
    while (n < a.size()+ b.size()) n<<=1;
    vector<CD> fa(a.begin(), a.end()), fb(b.begin(),
        b.end());
    fa.resize(n); fb.resize(n);
//    fft(fa); fft(fb);
    pairfft(fa, fb);
    for (int i=0; i<n; i++) fa[i] = fa[i] * fb[i];
    fft(fa, true);
    vector<ll> ans(n);
    for (int i=0; i<n; i++) ans[i] = round(fa[i].real());
    return ans;}
const int M = 1e9+7, B = sqrt(M)+1;
  vector<ll> anyMod(const vector<ll> &a, const
      vector<ll> &b) {
    int n = 1;
    while (n < a.size()+ b.size()) n<<=1;
    vector<CD> al(n), ar(n), bl(n), br(n);
    for (int i=0; i<a.size(); i++) al[i] = a[i]%M/B,
        ar[i] = a[i]%M%B;
    for (int i=0; i<b.size(); i++) bl[i] = b[i]%M/B,
        br[i] = b[i]%M%B;
    pairfft(al, ar); pairfft(bl, br);
//    fft(al); fft(ar); fft(bl); fft(br);
    for (int i=0; i<n; i++) {
      CD l1 = (al[i] * bl[i]), lr = (al[i] * br[i]);
      CD rl = (ar[i] * bl[i]), rr = (ar[i] * br[i]);
      al[i] = l1; ar[i] = lr;
      bl[i] = rl; br[i] = rr;}
   pairfft(al, ar, true); pairfft(bl, br, true);
//    fft(al, true); fft(ar, true); fft(bl, true);
    fft(br, true);
   vector<ll> ans(n);
    for (int i=0; i<n; i++) {
      ll right = round(br[i].real()), left =
          round(al[i].real());;
      ll mid = round(round(bl[i].real()) +
          round(ar[i].real()));
      ans[i] = ((left%M)*B*B + (mid%M)*B + right)%M;}
    return ans;}
}

vector<LL> a(n), b(m);
vector<LL> ans = FFT::anyMod(a, b);
ans.resize(n+m-1);
```

20 FWHT

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

const int N = 3e5 + 9, mod = 1e9 + 7;

int POW(long long n, long long k) {
    int ans = 1 % mod; n %= mod; if (n < 0) n += mod;
    while (k) {
        if (k & 1) ans = (long long) ans * n % mod;
        n = (long long) n * n % mod;
        k >>= 1;
    }
    return ans;
}
const int inv2 = (mod + 1) >> 1;
#define M (1 << 20)
#define OR 0
#define AND 1
#define XOR 2
struct FWHT{
    int P1[M], P2[M];
    void wt(int *a, int n, int flag = XOR) {
        if (n == 0) return;
        int m = n / 2;
        wt(a, m, flag); wt(a + m, m, flag);
        for (int i = 0; i < m; i++){
            int x = a[i], y = a[i + m];
            if (flag == OR) a[i] = x, a[i + m] = (x + y)
                % mod;
            if (flag == AND) a[i] = (x + y) % mod, a[i +
                m] = y;
            if (flag == XOR) a[i] = (x + y) % mod, a[i +
                m] = (x - y + mod) % mod;
        }
    }
    void iwt(int* a, int n, int flag = XOR) {
        if (n == 0) return;
        int m = n / 2;
        iwt(a, m, flag); iwt(a + m, m, flag);
        for (int i = 0; i < m; i++){
            int x = a[i], y = a[i + m];
            if (flag == OR) a[i] = x, a[i + m] = (y - x +
                mod) % mod;
            if (flag == AND) a[i] = (x - y + mod) % mod,
                a[i + m] = y;
            if (flag == XOR) a[i] = 1LL * (x + y) * inv2
                % mod, a[i + m] = 1LL * (x - y + mod) *
                inv2 % mod; // replace inv2 by >>1 if not
                required
        }
    }
    // Finds the pairwise XOR / AND / OR of two vector A
        and B
    vector<int> multiply(int n, vector<int> A,
        vector<int> B, int flag = XOR) {
        assert(__builtin_popcount(n) == 1);
        A.resize(n); B.resize(n);
        for (int i = 0; i < n; i++) P1[i] = A[i];
        for (int i = 0; i < n; i++) P2[i] = B[i];
        wt(P1, n, flag); wt(P2, n, flag);
        for (int i = 0; i < n; i++) P1[i] = 1LL * P1[i]
            * P2[i] % mod;
        iwt(P1, n, flag);
        return vector<int> (P1, P1 + n);
    }
    // Finds the presence of a value in all possible
        subset XOR / AND / OR of A
    vector<int> pow(int n, vector<int> A, long long k,
        int flag = XOR) {
        assert(__builtin_popcount(n) == 1);
        A.resize(n);
        for (int i = 0; i < n; i++) P1[i] = A[i];
        wt(P1, n, flag);
        for(int i = 0; i < n; i++) P1[i] = POW(P1[i], k);
        iwt(P1, n, flag);
        return vector<int> (P1, P1 + n);
    }
}t;
int32_t main() {
    int n; cin >> n;
    vector<int> a(M, 0);
    for(int i = 0; i < n; i++) {
        int k; cin >> k; a[k]++;
    }
    // Finds the pairwise XOR / AND / OR of two vector A
        and B
    vector<int> v2 = t.multiply(M,a,a,XOR);
    // Finds the presence of a value in all possible
        subset XOR / AND / OR of A
    vector<int> v = t.pow(M, a, n, AND);
    int ans = 1;
    for(int i = 1; i < M; i++) ans += v[i] > 0;
    cout << ans << '\n';;
    return 0;
}
```

21 FastIO

```
ios_base::sync_with_stdio(false);
cin.tie(NULL); cout.tie(NULL);
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
```

22 Formulas

```
Catalan number C(n) = (2n)! / ((n + 1)! * n!)
Catalan Triangle C(n,k) { (n+k)! * (n-k-1) } / { k! *
    (n+1)! }
Stirling2 S(n,k) = k*S(n-1,k)+S(n-1,k-1)
S(0,0) = 1 ; S(n,0) = S(0,n) = 0
```

23 Geo-2D

```
const double pi = 4 * atan(1);const double eps = 1e-6;
inline int dcmp (double x) { if (fabs(x) < eps) return
    0; else return x < 0 ? -1 : 1; }
double fix_acute(double th) {return th<-pi ? (th+2*pi):
    th>pi ? (th-2*pi) : th;}
inline double getDistance (double x, double y) { return
    sqrt(x * x + y * y); }
inline double torad(double deg) { return deg / 180 * pi;
    }
struct Pt {
 double x, y;
 Pt (double x = 0, double y = 0): x(x), y(y) {}
 void read () { scanf("%lf%lf", &x, &y); }
 void write () { printf("%lf %lf", x, y); }
 bool operator == (const Pt& u) const { return dcmp(x -
     u.x) == 0 && dcmp(y - u.y) == 0; }
 bool operator != (const Pt& u) const { return !(*this
     == u); }
 bool operator < (const Pt& u) const { return dcmp(x -
     u.x) < 0 || (dcmp(x-u.x)==0 && dcmp(y-u.y) < 0); }
 bool operator > (const Pt& u) const { return u < *this;
     }
 bool operator <= (const Pt& u) const { return *this < u
     || *this == u; }
 bool operator >= (const Pt& u) const { return *this > u
     || *this == u; }
 Pt operator + (const Pt& u) { return Pt(x + u.x, y +
     u.y); }
 Pt operator - (const Pt& u) { return Pt(x - u.x, y -
     u.y); }
 Pt operator * (const double u) { return Pt(x * u, y *
     u); }
 Pt operator / (const double u) { return Pt(x / u, y /
     u); }
 double operator * (const Pt& u) { return x*u.y - y*u.x;
     }
};
typedef Pt Vector;
typedef vector<Pt> Polygon;
struct Line {
 double a, b, c;
 Line (double a = 0, double b = 0, double c = 0): a(a),
     b(b), c(c) {}
};
struct Segment{
 Pt a;Pt b;
 Segment(){}
 Segment(Pt aa,Pt bb) {a=aa,b=bb;}
};
struct DirLine {
 Pt p;Vector v;
 double ang;
 DirLine () {}
 DirLine (Pt p, Vector v): p(p), v(v) { ang = atan2(v.y,
     v.x); }
 bool operator < (const DirLine& u) const { return ang <
     u.ang; }
};
namespace Punctual {
 double getDistance (Pt a, Pt b) { double x=a.x-b.x,
     y=a.y-b.y; return sqrt(x*x + y*y); }
};
namespace Vectorial {
 double getDot (Vector a, Vector b) { return a.x * b.x +
     a.y * b.y; }
 double getCross (Vector a, Vector b) { return a.x * b.y
     - a.y * b.x; }
 double getLength (Vector a) { return sqrt(getDot(a,
     a)); }
 double getPLength (Vector a) { return getDot(a, a); }
 double getAngle (Vector u) { return atan2(u.y, u.x); }
 double getSignedAngle (Vector a, Vector b) {return
     getAngle(b)-getAngle(a);}
 Vector rotate (Vector a, double rad) { return
     Vector(a.x*cos(rad)-a.y*sin(rad),
     a.x*sin(rad)+a.y*cos(rad)); }
 Vector ccw(Vector a, double co, double si) {return
     Vector(a.x*co-a.y*si, a.y*co+a.x*si);}
 Vector cw (Vector a, double co, double si) {return
     Vector(a.x*co+a.y*si, a.y*co-a.x*si);}
 Vector scale(Vector a, double s = 1.0) {return a /
     getLength(a) * s;}
 Vector getNormal (Vector a) { double l = getLength(a);
     return Vector(-a.y/l, a.x/l); }
};
namespace ComplexVector {
 typedef complex<double> Pt;
 typedef Pt Vector;
 double getDot(Vector a, Vector b) { return
     real(conj(a)*b); }
 double getCross(Vector a, Vector b) { return
     imag(conj(a)*b); }
 Vector rotate(Vector a, double rad) { return
     a*exp(Pt(0, rad)); }
};
namespace Linear {
 using namespace Vectorial;
 Line getLine (double x1, double y1, double x2, double
     y2) { return Line(y2-y1, x1-x2, y1*x2-x1*y2); }
 Line getLine (double a, double b, Pt u) { return
     Line(a, -b, u.y * b - u.x * a); }
 bool getIntersection (Line p, Line q, Pt& o) {
  if (fabs(p.a * q.b - q.a * p.b) < eps)
   return false;
  o.x = (q.c * p.b - p.c * q.b) / (p.a * q.b - q.a *
      p.b);
  o.y = (q.c * p.a - p.c * q.a) / (p.b * q.a - q.b *
      p.a);
  return true;
 }
 bool getIntersection (Pt p, Vector v, Pt q, Vector w,
     Pt& o) {
  if (dcmp(getCross(v, w)) == 0) return false;
  Vector u = p - q;
  double k = getCross(w, u) / getCross(v, w);
  o = p + v * k;
  return true;
 }
 double getDistanceToLine (Pt p, Pt a, Pt b) { return
     fabs(getCross(b-a, p-a) / getLength(b-a)); }
 double getDistanceToSegment (Pt p, Pt a, Pt b) {
  if (a == b) return getLength(p-a);
  Vector v1 = b - a, v2 = p - a, v3 = p - b;
  if (dcmp(getDot(v1, v2)) < 0) return getLength(v2);
  else if (dcmp(getDot(v1, v3)) > 0) return
      getLength(v3);
  else return fabs(getCross(v1, v2) / getLength(v1));
 }
 double getDistanceSegToSeg (Pt a,Pt b,Pt c,Pt d){
  double Ans=INT_MAX;
  Ans=min(Ans,getDistanceToSegment(a,c,d));
  Ans=min(Ans,getDistanceToSegment(b,c,d));
  Ans=min(Ans,getDistanceToSegment(c,a,b));
  Ans=min(Ans,getDistanceToSegment(d,a,b));
  return Ans;
 }
 Pt getPtToLine (Pt p, Pt a, Pt b) { Vector v = b-a;
     return a+v*(getDot(v, p-a) / getDot(v,v)); }
 bool onSegment (Pt p, Pt a, Pt b) { return
     dcmp(getCross(a-p, b-p)) == 0 && dcmp(getDot(a-p,
     b-p)) <= 0; }
 bool haveIntersection (Pt a1, Pt a2, Pt b1, Pt b2) {
  if(onSegment(a1,b1,b2)) return true;
  if(onSegment(a2,b1,b2)) return true;
  if(onSegment(b1,a1,a2)) return true;
  if(onSegment(b2,a1,a2)) return true; //Case of touch
  double c1=getCross(a2-a1, b1-a1), c2=getCross(a2-a1,
      b2-a1), c3=getCross(b2-b1, a1-b1),
      c4=getCross(b2-b1,a2-b1);
  return dcmp(c1)*dcmp(c2) < 0 && dcmp(c3)*dcmp(c4) < 0;
 }
 bool onLeft(DirLine l, Pt p) { return dcmp(l.v *
     (p-l.p)) >= 0; }
}
namespace Triangular {
 using namespace Vectorial;
 double getAngle (double a, double b, double c) { return
     acos((a*a+b*b-c*c) / (2*a*b)); }
 double getArea (double a, double b, double c) { double
     s =(a+b+c)/2; return sqrt(s*(s-a)*(s-b)*(s-c)); }
 double getArea (double a, double h) { return a * h / 2;
     }
 double getArea (Pt a, Pt b, Pt c) { return
     fabs(getCross(b - a, c - a)) / 2; }
 double getDirArea (Pt a, Pt b, Pt c) { return
     getCross(b - a, c - a) / 2;}
 //ma/mb/mc = length of median from side a/b/c
 double getArea_(double ma,double mb,double mc) {double
     s=(ma+mb+mc)/2; return 4/3.0 *
     sqrt(s*(s-ma)*(s-mb)*(s-mc));}
 //ha/hb/hc = length of perpendicular from side a/b/c
 double get_Area(double ha,double hb,double hc){
  double H=(1/ha+1/hb+1/hc)/2; double _A_ = 4 * sqrt(H *
      (H-1/ha)*(H-1/hb)*(H-1/hc)); return 1.0/_A_;
 }
 bool PtInTriangle(Pt a, Pt b, Pt c, Pt p){
  double s1 = getArea(a,b,c);
  double s2 = getArea(p,b,c) + getArea(p,a,b) +
      getArea(p,c,a);
  return dcmp(s1 - s2) == 0;
 }
};
namespace Polygonal {
 using namespace Vectorial;
 using namespace Linear;
 using namespace Triangular;
 double getSignedArea (Pt* p, int n) {
  double ret = 0;
  for (int i = 0; i < n-1; i++)
   ret += (p[i]-p[0]) * (p[i+1]-p[0]);
  return ret/2;
 }
 int getConvexHull (Pt* p, int n, Pt* ch) {
  sort(p, p + n);
  // preparing lower hull
  int m = 0;
  for (int i = 0; i < n; i++){
   while (m > 1 && dcmp(getCross(ch[m-1]-ch[m-2],
       p[i]-ch[m-1])) <= 0) m--;
   ch[m++] = p[i];
  }
  // preparing upper hull
  int k = m;
  for (int i = n-2; i >= 0; i--){
```

```
  while (m > k && dcmp(getCross(ch[m-1]-ch[m-2],
      p[i]-ch[m-2])) <= 0) m--;
  ch[m++] = p[i];
 }
 if (n > 1) m--;
 return m;
}
int isPtInPolygon (Pt o, Pt* p, int n) {
 int wn = 0;
 for (int i = 0; i < n; i++) {
  int j = (i + 1) % n;
  if (onSegment(o, p[i], p[j]) || o == p[i]) return 0;
  int k = dcmp(getCross(p[j] - p[i], o-p[i]));
  int d1 = dcmp(p[i].y - o.y);
  int d2 = dcmp(p[j].y - o.y);
  if (k > 0 && d1 <= 0 && d2 > 0) wn++;
  if (k < 0 && d2 <= 0 && d1 > 0) wn--;
 }
 return wn ? -1 : 1;
}
void rotatingCalipers(Pt *p, int n, vector<Segment>&
    sol) {
 sol.clear();
 int j = 1; p[n] = p[0];
 for (int i = 0; i < n; i++) {
  while (getCross(p[j+1]-p[i+1], p[i]-p[i+1]) >
      getCross(p[j]-p[i+1], p[i]-p[i+1]))
   j = (j+1) % n;
  sol.push_back(Segment(p[i],p[j]));
  sol.push_back(Segment(p[i + 1],p[j + 1]));
 }
}
void rotatingCalipersGetRectangle (Pt *p, int n,
    double& area, double& perimeter) {
 p[n] = p[0];
 int l = 1, r = 1, j = 1;
 area = perimeter = 1e20;
 for (int i = 0; i < n; i++) {
  Vector v = (p[i+1]-p[i]) / getLength(p[i+1]-p[i]);
  while (dcmp(getDot(v, p[r%n]-p[i]) - getDot(v,
      p[(r+1)%n]-p[i])) < 0) r++;
  while (j < r || dcmp(getCross(v, p[j%n]-p[i]) -
      getCross(v,p[(j+1)%n]-p[i])) < 0) j++;
  while (l < j || dcmp(getDot(v, p[l%n]-p[i]) -
      getDot(v, p[(l+1)%n]-p[i])) > 0) l++;
  double w = getDot(v, p[r%n]-p[i])-getDot(v,
      p[l%n]-p[i]);
  double h = getDistanceToLine (p[j%n], p[i], p[i+1]);
  area = min(area, w * h);
  perimeter = min(perimeter, 2 * w + 2 * h);
 }
}
Polygon cutPolygon (Polygon u, Pt a, Pt b) {
 Polygon ret;
 int n = u.size();
 for (int i = 0; i < n; i++) {
  Pt c = u[i], d = u[(i+1)%n];
  if (dcmp((b-a)*(c-a)) >= 0) ret.push_back(c);
  if (dcmp((b-a)*(d-c)) != 0) {
   Pt t;
   getIntersection(a, b-a, c, d-c, t);
   if (onSegment(t, c, d))
    ret.push_back(t);
  }
 }
 return ret;
}
int halfPlaneIntersection(DirLine* li, int n, Pt* poly)
    {
 sort(li, li + n);
 int first, last;
 Pt* p = new Pt[n];
 DirLine* q = new DirLine[n];
 q[first=last=0] = li[0];
 for (int i = 1; i < n; i++) {
  while (first < last && !onLeft(li[i], p[last-1]))
      last--;
  while (first < last && !onLeft(li[i], p[first]))
      first++;
  q[++last] = li[i];
  if (dcmp(q[last].v * q[last-1].v) == 0) {
   last--;
   if (onLeft(q[last], li[i].p)) q[last] = li[i];
  }
  if (first < last)
   getIntersection(q[last-1].p, q[last-1].v, q[last].p,
       q[last].v, p[last-1]);
 }
 while (first < last && !onLeft(q[first], p[last-1]))
     last--;
 if (last - first <= 1) { delete [] p; delete [] q;
     return 0; }
 getIntersection(q[last].p, q[last].v, q[first].p,
     q[first].v, p[last]);
 int m = 0;
 for (int i = first; i <= last; i++) poly[m++] = p[i];
 delete [] p; delete [] q;
 return m;
}
Polygon simplify (const Polygon& poly) {
 Polygon ret;
 int n = poly.size();
 for (int i = 0; i < n; i++) {
  Pt a = poly[i];
  Pt b = poly[(i+1)%n];
  Pt c = poly[(i+2)%n];
  if (dcmp((b-a)*(c-b)) != 0 && (ret.size() == 0 || b
      != ret[ret.size()-1]))
   ret.push_back(b);
 }
 return ret;
}
Pt ComputeCentroid( Pt* p,int n){
 Pt c(0,0);
 double scale = 6.0 * getSignedArea(p,n);
 for (int i = 0; i < n; i++){
  int j = (i+1) % n;
  c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
 }
 return c / scale;
}
// pt must be in ccw order with no three collinear Pts
// returns inside = 1, on = 0, outside = -1
int PtInConvexPolygon(Pt* pt, int n, Pt p){
 assert(n >= 3);
 int lo = 1 , hi = n - 1 ;
 while(hi - lo > 1){
  int mid = (lo + hi) / 2;
  if(getCross(pt[mid] - pt[0], p - pt[0]) > 0) lo = mid;
  else hi = mid;
 }
 bool in = PtInTriangle(pt[0], pt[lo], pt[hi], p);
 if(!in) return -1;
 if(getCross(pt[lo] - pt[lo-1], p - pt[lo-1]) == 0)
     return 0;
 if(getCross(pt[hi] - pt[lo], p - pt[lo]) == 0) return
     0;
 if(getCross(pt[hi] - pt[(hi+1)%n], p - pt[(hi+1)%n])
     == 0) return 0;
 return 1;
}
// Calculate [ACW, CW] tangent pair from an external Pt
#define CW          -1
#define ACW          1
int direction(Pt st, Pt ed, Pt q) {return
    dcmp(getCross(ed - st, q - ed));}
bool isGood(Pt u, Pt v, Pt Q, int dir) {return
    direction(Q, u, v) != -dir;}
Pt better(Pt u, Pt v, Pt Q, int dir) {return
    direction(Q, u, v) == dir ? u : v;}
Pt tangents(Pt* pt, Pt Q, int dir, int lo, int hi){
 while(hi - lo > 1){
  int mid = (lo + hi)/2;
  bool pvs = isGood(pt[mid], pt[mid - 1], Q, dir);
  bool nxt = isGood(pt[mid], pt[mid + 1], Q, dir);
  if(pvs && nxt) return pt[mid];
  if(!(pvs || nxt)){
   Pt p1 = tangents(pt, Q, dir, mid+1, hi);
   Pt p2 = tangents(pt, Q, dir, lo, mid - 1);
   return better(p1, p2, Q, dir);
  }
  if(!pvs){
   if(direction(Q, pt[mid], pt[lo]) == dir) hi = mid -
       1;
   else if(better(pt[lo], pt[hi], Q, dir) == pt[lo]) hi
       = mid - 1;
   else lo = mid + 1;
  }
  if(!nxt){
   if(direction(Q, pt[mid], pt[lo]) == dir) lo = mid +
       1;
   else if(better(pt[lo], pt[hi], Q, dir) == pt[lo]) hi
       = mid - 1;
   else lo = mid + 1;
  }
 }
 Pt ret = pt[lo];
 for(int i = lo + 1; i <= hi; i++) ret = better(ret,
     pt[i], Q, dir);
 return ret;
}
// [ACW, CW] Tangent
pair<Pt, Pt> get_tangents(Pt* pt, int n, Pt Q){
 Pt acw_tan = tangents(pt, Q, ACW, 0, n - 1);
 Pt cw_tan = tangents(pt, Q, CW, 0, n - 1);
 return make_pair(acw_tan, cw_tan);
}
};
struct Circle {
 Pt o;double r;
 Circle () {}
 Circle (Pt o, double r = 0): o(o), r(r) {}
 void read () { o.read(), scanf("%lf", &r); }
 Pt pt(double rad) { return Pt(o.x + cos(rad)*r, o.y +
     sin(rad)*r); }
 double getArea (double rad) { return rad * r * r / 2; }
 //area of the circular sector cut by a chord with
     central angle alpha
 double sector(double alpha) {return r * r * 0.5 *
     (alpha - sin(alpha));}
};

//Polar Sort
inline bool up (Pt p) {
  return p.y > 0 or (p.y == 0 and p.x >= 0);
}
sort(v.begin(), v.end(), [] (Pt a, Pt b) {
```

```
  return up(a) == up(b) ? a.x * b.y > a.y * b.x : up(a)
      < up(b);
});
```

24 Geo-3D

```
const double pi = 4 * atan(1);
const double eps = 1e-10;
inline int dcmp (double x) { if (fabs(x) < eps) return
    0; else return x < 0 ? -1 : 1; }
inline double torad(double deg) { return deg / 180 * pi;
    }
struct Point{
 double x, y;
 Point (double x = 0, double y = 0): x(x), y(y) {}
 Point operator + (const Point& u) { return Point(x +
     u.x, y + u.y); }
 Point operator - (const Point& u) { return Point(x -
     u.x, y - u.y); }
 Point operator * (const double u) { return Point(x * u,
     y * u); }
 Point operator / (const double u) { return Point(x / u,
     y / u); }
 double operator * (const Point& u) { return x*u.y -
     y*u.x; }
};
struct Pt3D{
 double x, y, z;
 Pt3D() {}
 void read () {cin>>x>>y>>z;}
 void write () {cout<<x<<" --- "<<y<<" --- "<<z<<"\n";}
 Pt3D(double x, double y, double z) : x(x), y(y), z(z) {}
 Pt3D(const Pt3D &p) : x(p.x), y(p.y), z(p.z) {}
 Pt3D operator +(Pt3D b) {return Pt3D(x+b.x,y+b.y,
     z+b.z);}
 Pt3D operator -(Pt3D b) {return Pt3D(x-b.x,y-b.y,
     z-b.z);}
 Pt3D operator *(double b) {return Pt3D(x*b,y*b, z*b);}
 Pt3D operator /(double b) {return Pt3D(x/b,y/b, z/b);}
 bool operator <(Pt3D b) {return
     make_pair(make_pair(x,y),z) <
     make_pair(make_pair(b.x,b.y),b.z);}
 bool operator ==(Pt3D b) {return dcmp(x-b.x)==0 &&
     dcmp(y-b.y) == 0 && dcmp(z-b.z) == 0;}
};
typedef Pt3D Vector3D;
typedef vector<Point> Polygon;
typedef vector<Pt3D> Polyhedron;
namespace Vectorial{
 double getDot (Vector3D a, Vector3D b) {return
     a.x*b.x+a.y*b.y+a.z*b.z;}
 Vector3D getCross(Vector3D a, Vector3D b) {return
     Pt3D(a.y*b.z-a.z*b.y, a.z*b.x-a.x*b.z,
     a.x*b.y-a.y*b.x);}
 double getLength (Vector3D a) {return sqrt(getDot(a,
     a)); }
 double getPLength (Vector3D a) {return getDot(a, a); }
 Vector3D unitVector(Vector3D v) {return v/getLength(v);}
 double getUnsignedAngle(Vector3D u,Vector3D v){
  double cosTheta =
      getDot(u,v)/getLength(u)/getLength(v);
  cosTheta = max(-1.0,min(1.0,cosTheta));
  return acos(cosTheta);
 }
 Vector3D rotate(Vector3D v,Vector3D a,double rad){
  a = unitVector(a);
  return v * cos(rad) + a * (1 - cos(rad)) * getDot(a,v)
      + getCross(a,v) * sin(rad);
 }
}
struct Line3D{
 Vector3D v; Pt3D o;
 Line3D() {};
 Line3D(Vector3D v,Pt3D o):v(v),o(o){}
 Pt3D getPoint(double t) {return o + v*t;}
};
namespace Linear{
 using namespace Vectorial;
 double getDistSq(Line3D l, Pt3D p) {return
     getPLength(getCross(l.v,p-l.o))/getPLength(l.v);}
 double getDistLinePoint(Line3D l, Pt3D p) {return
     sqrt(getDistSq(l,p));}
 bool cmp(Line3D l,Pt3D p, Pt3D q) {return getDot(l.v,p)
     < getDot(l.v,q);}
 Pt3D projection(Line3D l,Pt3D p) {return l.o + l.v *
     getDot(l.v,p-l.o)/getPLength(l.v);}
 Pt3D reflection(Line3D l,Pt3D p) {return
     projection(l,p)+projection(l,p)-p;}
 double getAngle(Line3D l,Line3D m) {return
     getUnsignedAngle(l.v,m.v);}
 bool isParallel(Line3D p,Line3D q) {return
     dcmp(getPLength(getCross(p.v,q.v))) == 0;}
 bool isPerpendicular(Line3D p,Line3D q) {return
     dcmp(getDot(p.v,q.v)) == 0;}
 double getDist(Line3D l, Line3D m){
  Vector3D n = getCross(l.v, m.v);
  if(getPLength(n) == 0) return getDistLinePoint(l,m.o);
  else return fabs(getDot(m.o-l.o , n)) / getLength(n);
 }
 Pt3D getClosestPointOnLine1(Line3D l,Line3D m){
  Vector3D n = getCross(l.v, m.v);
  Vector3D n2 = getCross(m.v, n);
  return l.o + l.v * getDot(m.o-l.o, n2) / getDot(l.v,
      n2);
 }
}
struct Plane{
 Vector3D n; //normal n
 double d; //getDot(n,p) = d for any point p on the plane
 Plane() {}
 Plane(Vector3D n, double d) : n(n), d(d) {}
 Plane(Vector3D n, Pt3D p) : n(n), d(Vectorial ::
     getDot(n,p)) {}
 Plane(const Plane &p) : n(p.n), d(p.d) {}
};
namespace Planar{
 using namespace Vectorial;
 Plane getPlane(Pt3D a,Pt3D b,Pt3D c) {return
     Plane(getCross(b-a,c-a),a);}
 Plane translate(Plane p,Vector3D t) {return Plane(p.n,
     p.d+getDot(p.n,t));}
 Plane shiftUp(Plane p,double dist) {return Plane(p.n,
     p.d+dist*getLength(p.n));}
 Plane shiftDown(Plane p,double dist) {return Plane(p.n,
     p.d-dist*getLength(p.n));}
 double getSide(Plane p,Pt3D a) {return
     getDot(p.n,a)-p.d;}
 double getDistance(Plane p,Pt3D a) {return
     fabs(getSide(p,a))/getLength(p.n);}
 Pt3D projection(Plane p,Pt3D a) {return
     a-p.n*getSide(p,a)/getPLength(p.n);}
 Pt3D reflection(Plane p,Pt3D a) {return
     a-p.n*getSide(p,a)/getPLength(p.n)*2;}
 bool intersect(Plane p, Line3D l, Pt3D& a){
  if(dcmp(getDot(p.n,l.v)) == 0) return false;
  a = l.o - l.v * getSide(p,l.o) / getDot(p.n,l.v);
  return true;
 }
 bool intersect(Plane p,Plane q,Line3D& l){
  l.v = getCross(p.n,q.n);
  if(dcmp(getPLength(l.v)) == 0) return false;
  l.o = getCross(q.n*p.d - p.n*q.d , l.v) /
      getPLength(l.v);
  return true;
 }
 double getAngle(Plane p,Plane q) {return
     getUnsignedAngle(p.n,q.n);}
 bool isParallel(Plane p,Plane q) {return
     dcmp(getPLength(getCross(p.n,q.n))) == 0;}
 bool isPerpendicular(Plane p,Plane q) {return
     dcmp(getDot(p.n,q.n)) == 0;}
 bool getAngle(Plane p,Line3D l) {return pi/2.0 -
     getUnsignedAngle(p.n,l.v);}
 bool isParallel(Plane p,Line3D l) {return
     dcmp(getDot(p.n,l.v)) == 0;}
 bool isPerpendicular(Plane p,Line3D l) {return
     dcmp(getPLength(getCross(p.n,l.v))) == 0;}
 Line3D perpThrough(Plane p,Pt3D a) {return
     Line3D(p.n,a);}
 Plane perpThrough(Line3D l,Pt3D a) {return
     Plane(l.v,a);}
 //Modify p.n if necessary with respect to the reference
     point
 Vector3D rotateCCW90(Plane p,Vector3D d) {return
     getCross(p.n,d);}
 Vector3D rotateCW90(Plane p,Vector3D d) {return
     getCross(d,p.n);}
 pair<Pt3D, Pt3D> TwoPointsOnPlane(Plane p){
  Vector3D N = p.n; double D = p.d;
  assert(dcmp(N.x) != 0 || dcmp(N.y) != 0 || dcmp(N.z)
      != 0);
  if(dcmp(N.x) == 0 && dcmp(N.y) == 0) return
      {Pt3D(1,0,D/N.z), Pt3D(0,1,D/N.z)};
  if(dcmp(N.y) == 0 && dcmp(N.z) == 0) return
      {Pt3D(D/N.x,1,0), Pt3D(D/N.x,0,1)};
  if(dcmp(N.z) == 0 && dcmp(N.x) == 0) return
      {Pt3D(1,D/N.y,0), Pt3D(0,D/N.y,1)};
  if(dcmp(N.x) == 0) return {Pt3D(1,D/N.y,0),
      Pt3D(0,0,D/N.z)};
  if(dcmp(N.y) == 0) return {Pt3D(0,1,D/N.z),
      Pt3D(D/N.x,0,0)};
  if(dcmp(N.z) == 0) return {Pt3D(D/N.x,0,1),
      Pt3D(0,D/N.y,0)};
  if(dcmp(D)!=0) return {Pt3D(D/N.x,0,0),
      Pt3D(0,D/N.y,0)};
  return {Pt3D(N.y,-N.x,0), Pt3D(-N.y,N.x,0)};
 }
 Point From3Dto2D(Plane p, Pt3D a){
  assert( dcmp(getSide(p,a)) == 0 );
  auto Pair = TwoPointsOnPlane(p);
  Pt3D A = Pair.first;
  Pt3D B = Pair.second;
  Vector3D Z = p.n;   Z = Z / getLength(Z);
  Vector3D X = B - A;    X = X / getLength(X);
  Vector3D Y = getCross(Z,X);
  Vector3D v = a - A;
  assert( dcmp(getDot(v,Z)) == 0);
  return Point(getDot(v,X),getDot(v,Y));
 }
 Pt3D From2Dto3D(Plane p, Point a){
  auto Pair = TwoPointsOnPlane(p);
  Pt3D A = Pair.first;
  Pt3D B = Pair.second;
  Vector3D Z = p.n;   Z = Z / getLength(Z);
```

```
  Vector3D X = B - A;    X = X / getLength(X);
  Vector3D Y = getCross(Z,X);
  return A + X * a.x + Y * a.y;
 }
}
struct Sphere{
 Pt3D c;
 double r;
 Sphere() {}
 Sphere(Pt3D c, double r) : c(c), r(r) {}
 //Spherical cap with polar angle theta
 double Height(double alpha) {return r*(1-cos(alpha));}
 double BaseRadius(double alpha) {return r*sin(alpha);}
 double Volume(double alpha) {double h = Height(alpha);
     return pi*h*h*(3*r-h)/3.0;}
 double SurfaceArea(double alpha) {double h =
     Height(alpha); return 2*pi*r*h;}
};
namespace Spherical{
 using namespace Vectorial;
 using namespace Planar;
 using namespace Linear;
 Sphere CircumscribedSphere(Pt3D a,Pt3D b,Pt3D c,Pt3D d){
  assert( dcmp(getSide(getPlane(a,b,c), d)) != 0);
  Plane U = Plane(a-b, (a+b)/2);
  Plane V = Plane(b-c, (b+c)/2);
  Plane W = Plane(c-d, (c+d)/2);
  Line3D l1,l2;
  bool ret1 = intersect(U,V,l1);
  bool ret2 = intersect(V,W,l2);
  assert(ret1 == true && ret2 == true);
  assert( dcmp(getDist(l1,l2)) == 0);
  Pt3D C = getClosestPointOnLine1(l1,l2);
  return Sphere(C, getLength(C-a));
 }
 pair<double,double> SphereSphereIntersection(Sphere
     s1,Sphere s2){
  double d = getLength(s1.c-s2.c);
  if(dcmp(d - s1.r -s2.r) >= 0) return {0,0};
  double R1 = max(s1.r,s2.r); double R2 = min(s1.r,s2.r);
  double y = R1 + R2 - d;
  double x = (R1*R1 - R2*R2 + d*d) / (2*d);
  double h1 = R1 - x;
  double h2 = y - h1;
  double Volume = pi*h1*h1*(3*R1-h1)/3.0 +
      pi*h2*h2*(3*R2-h2)/3.0;
  double SurfaceArea = 2*pi*R1*h1 + 2*pi*R2*h2;
  return make_pair(SurfaceArea,Volume);
 }
 Pt3D getPointOnSurface(double r,double Lat,double Lon){
  Lat = torad(Lat); //North-South
  Lon = torad(Lon); //East-West
  return Pt3D(r*cos(Lat)*cos(Lon), r*cos(Lat)*sin(Lon),
      r*sin(Lat));
 }
 int intersect(Sphere s,Line3D l, vector<Pt3D>& ret){
  double h2 = s.r*s.r - getDistSq(l,s.c);
  if(dcmp(h2)<0) return 0;
  Pt3D p = projection(l,s.c);
  if(dcmp(h2) == 0) {ret.push_back(p); return 1;}
  Vector3D h = l.v * sqrt(h2) / getLength(l.v);
  ret.push_back(p-h); ret.push_back(p+h); return 2;
 }
 double GreatCircleDistance(Sphere s,Pt3D a,Pt3D b){
  return s.r * getUnsignedAngle(a-s.c, b-s.c);
 }
}
namespace Poly{
 using namespace Vectorial;
```

```
 Sphere SmallestEnclosingSphere(Polyhedron p){
  int n = p.size();
  Pt3D C(0,0,0);
  for(int i=0; i<n; i++) C = C + p[i];
  C = C / n;
  double P = 0.1;
  int pos = 0;
  int Accuracy = 70000;
  for (int i = 0; i < Accuracy; i++) {
   pos = 0;
   for (int j = 1; j < n; j++){
    if(getPLength(C - p[j]) > getPLength(C - p[pos]))
        pos = j;
   }
   C = C + (p[pos] - C)*P;
   P *= 0.998;
  }
  return Sphere(C, getPLength(C - p[pos]));
 }
}
```

25 Geometry — Mashroor

```
typedef double db;
#define pi acos(-1.0)
#define mp make_pair
#define DB_MAX 1000000000.00

class point{
public:
    db x;
    db y;
    point(db x, db y){
        this->x=x; this->y=y;
    }
    point(){
        this->x=0; this->y=0;
    }
    db dist(point a){
        return (this->x-a.x)* (this->x-a.x)+
            (this->y-a.y)* (this->y-a.y);
    }
    db dist(){
        return (this->x)*(this->x)+(this->y)*(this->y);
    }
    db angle(){
        db d=sqrt(this->dist());
        db t=asin(this->y/d);
        t=abs(t);
        if(x>=0 && y>=0) return t;
        if(x<0 && y>=0) return pi-t;
        if(x>=0 && y<0) return pi+pi-t;
        return pi+t;
    }
    db angle(point p){
        point h(p.x-this->x, p.y-this->y);
        return h.angle();
    }
    friend ostream& operator<<(ostream& os, const point&
        dt);

    point operator +(const point& b) const { return
        point{x+b.x, y+b.y}; }
    point operator -(const point& b) const { return
        point{x-b.x, y-b.y}; }
    ll operator *(const point& b) const { return (ll) x
        * b.y - (ll) y * b.x; }
    void operator +=(const point& b) { x += b.x; y +=
        b.y; }
    void operator -=(const point& b) { x -= b.x; y -=
        b.y; }
    void operator *=(const int k) { x *= k; y *= k; }

    ll cross(const point& b, const point& c) const {
        return (b - *this) * (c - *this);
    }
    bool bet(point a, point b){
        db dx=(a.x-x)*(x-b.x);
        db dy=(a.y-y)*(y-b.y);
        return (dx<0 && dy<0)? false : true;
    }
};

ostream& operator<<(ostream& os, const point& dt)
{
    os << "("<< dt.x<< ","<< dt.y<< ")";
    return os;
}

class line{
public:
        db a;
        db b;
        db c;
        line(db a, db b, db c){
                this->a=a;
                this->b=b;
                this->c=c;
        }
        line(){
                this->a=0;
                this->b=0;
                this->c=0;
        }
        line(point P, point Q){
                a = Q.y - P.y;
                b = P.x - Q.x;
                c = -(a * (P.x) + b * (P.y));
        }
        line(db x1, db y1, db r1, db x2, db y2, db r2){
                a=2*x1-2*x2;
                b=2*y1-2*y2;
                c=x2*x2-x1*x1+y2*y2-y1*y1+r1*r1-r2*r2;
        }
        db side(point g){
                return (g.x*a+g.y*b+c);
        }
        db dis(point g){
                return abs((a*g.x+b*g.y+c)/sqrt(a*a+b*b));
        }
        bool equal(line L){
                db f;
                if(L.a!=0) f=this->a/L.a;
                else if(L.b!=0) f=this->b/L.b;
                else if(L.c!=0) f=this->c/L.c;
                return ((this->a/L.a==f || (this->a==L.a
                    && this->a==0))&&
                       (this->b/L.b==f || (this->b==L.b
                           && this->b==0))&&
                       (this->c/L.c==f || (this->c==L.c
                           && this->c==0)))? true : false;
        }

        bool parallel(line L){
                db f;
                if(L.a!=0) f=this->a/L.a;
                else if(L.b!=0) f=this->b/L.b;
                return ((this->a/L.a==f || (this->a==L.a
                    && this->a==0))&&
                (this->b/L.b==f || (this->b==L.b &&
                    this->b==0)))? true : false;
```

```
        }
        point intersect(line L){
                db a1=this->a;
                db b1=this->b;
                db c1=this->c;
                db a2=L.a;
                db b2=L.b;
                db c2=L.c;
                db det=-a1*b2+a2*b1;

                if (det == 0){
                        return point(DB_MAX, DB_MAX);
                }
                else{
                        db x = (b2*c1 - b1*c2)/det;
                        db y = (a1*c2 - a2*c1)/det;
                        return point(x, y);
                }
        }
        db getx(db Y){
        return -(b*Y+c)/a;
        }
        db gety(db X){
        return -(a*X+c)/b;
        }
    friend ostream& operator<<(ostream& os, const line&
         dt);
    bool operator==(const line& dt){
        point o(0,0);
        line L1(this->a,this->b,this->c);
        line L2(dt.a,dt.b,dt.c);
        return (L1.a*L2.b==L1.b*L2.a &&
            L1.dis(o)==L2.dis(o))? true : false;
    }
};

ostream& operator<<(ostream& os, const line& dt)
{
    os << "("<< dt.a<< ","<< dt.b<< ","<< dt.c<< ")";
    return os;
}

class circle{
public:
    point c;
    db r;
    circle(db x, db y, db r){
        c.x=x;
        c.y=y;
        this->r=r;
    }
    circle(){
        c.x=0;
        c.y=0;
        this->r=0;
    }
    db area(){
        return pi*r*r;
    }
    db intersection(circle C2){
        circle C1=*this;
        db l = sqrt(C1.c.dist(C2.c)), ans;
        if(l>=C1.r+C2.r) ans = 0;
        else if(l<=abs(C1.r-C2.r)) ans =
            pi*min(C1.r,C2.r)*min(C1.r,C2.r);

        else{

            db t1 =
                acos((C1.r*C1.r+l*l-C2.r*C2.r)/(2*l*C1.r));
            db t2 =
                acos((C2.r*C2.r+l*l-C1.r*C1.r)/(2*l*C2.r));

            ans =C1.r*C1.r*t1 - C1.r*C1.r*cos(t1)*sin(t1)
                + C2.r*t2*C2.r -
                C2.r*C2.r*cos(t2)*sin(t2);

        }
        return ans;
    }
};

class polygon{
public:
    ll size;
    point* P;
    polygon(ll n){
        size=n;
        P= new point[n];
    }
    polygon(point* a, point* b){
        ll n=b-a;
        size=n;
        P= new point[n];
        for(ll i=0; i<n; i++){
            P[i].x=a[i].x;
            P[i].y=a[i].y;
        }
    }
    polygon(vector< point > a){
        ll n=a.size();
        size=n;
        P= new point[n];
        for(ll i=0; i<n; i++){
            P[i].x=a[i].x;
            P[i].y=a[i].y;
        }
    }
    db area(){
        db up=0;
        db dn=0;
        ll n=size;
        for(ll i=0; i<n; i++){
            up+=P[i].x*P[(i+1)%n].y;
            dn+=P[i].y*P[(i+1)%n].x;
        }
        return up-dn;
    }
    point centroid(){
        if(size==1) return P[0];
        if(size==2){
            point p((P[0].x+P[1].x)/2,(P[0].y+P[1].y)/2);
            return p;
        }
        db A=area();
        point c(0,0);
        ll n=size;
        for(ll i=0; i<n; i++){
            ll j=(i+1)%n;
            c.x+=((P[i].x+P[j].x)*
                (P[i].x*P[j].y-P[j].x*P[i].y));
            c.y+=((P[i].y+P[j].y)*
                (P[i].x*P[j].y-P[j].x*P[i].y));
        }
        c.x/=(3*A);
        c.y/=(3*A);
        return c;
    }
};

point atx(point a, point b, db d){
    if(a.x==d) return a;
    if(b.x==d) return b;
    line L(a,b);
    point c(d,L.gety(d));
    return c;
}

bool between(db a, db b, db c){
    db d=(a-b)*(b-c);
    return (d<0)? false : true;
}

point pointrat(point a, db ad, point b, db bd){
    point c((ad*a.x+bd*b.x)/
        (ad+bd),(ad*a.y+bd*b.y)/(ad+bd));
    return c;
}

bool onseg(point a, point b, point c){
    line L1(a,b), L2(a,c);
    //cout<< (a.x-b.x)*(c.x-b.x)<< " uuuu "<<
        (a.y-b.y)*(c.y-b.y)<< endl;
    return (L1==L2 && (a.x-b.x)*(c.x-b.x)<=0 &&
        (a.y-b.y)*(c.y-b.y)<=0)? true : false;
}


//----------------------Convex Hull
point start;

db area(point a, point b, point c){
    return ((a.x*b.y+b.x*c.y+c.x*a.y)-
        (a.y*b.x+b.y*c.x+c.y*a.x));
}

bool compPoint(point a, point b){
    db r=area(start,a,b);
    if(r==0){
        return start.dist(a)<=start.dist(b);
    }
    return (r<0)? true : false;
}

bool compLast(point a, point b){
    return start.dist(a)>=start.dist(b);
}

polygon convexhull(vector<point> hull){
    ll n=hull.size();
        bool* ishull=new bool[n];
    for(ll i=0; i<n ; i++) ishull[i]=false;

    point corner(DB_MAX,DB_MAX);
    int X, Y, z;
    for(int i=0; i<n; i++){
        if(hull[i].x<corner.x){
            corner.x=hull[i].x;
            corner.y=hull[i].y;
            z=i;
        }
        else if(hull[i].x==corner.x &&
             hull[i].y<corner.y){
            corner.x=hull[i].x;
            corner.y=hull[i].y;
            z=i;
        }
    }
    start=hull[z];
    swap(hull[0],hull[z]);

   sort(hull.begin()+1,hull.end(),compPoint);

    ll d=n-2;
    while(d>0 && area(start,hull[d],hull[d+1])==0) d--;
    sort(hull.begin()+d+1,hull.end(),compLast);

    vector< ll > ans;
```

```
    ans.push_back(0);
    ans.push_back(1);
    ans.push_back(2);
    int v=ans.size();
    for(ll i=3; i<n; i++){
        ll a=v-2;
        ll b=v-1;
        if(area(hull[ans[a]],hull[ans[b]],hull[i])<=0){
            if(ans.size()==v) ans.push_back(i);
            else ans[v]=i;
            v++;
        }
        else{
            v--;
            i--;
        }
    }
    polygon pg(v);
    for(ll i=0; i<v; i++) pg.P[i]=hull[ans[i]];
    return pg;
}

typedef struct{
    ll x1;
    ll y1;
    ll x2;
    ll y2;
} rect;

ll area(rect R){
    if(R.x1>R.x2 || R.y1>R.y2) return 0;
    return (R.x2-R.x1)*(R.y2-R.y1);
}

ll intersect(rect P, rect Q){
    rect R;
    R.x1=max(P.x1,Q.x1);
    R.y1=max(P.y1,Q.y1);
    R.x2=min(P.x2,Q.x2);
    R.y2=min(P.y2,Q.y2);
    return area(R);
}

rect sect(rect P, rect Q){
    rect R;
    R.x1=max(P.x1,Q.x1);
    R.y1=max(P.y1,Q.y1);
    R.x2=min(P.x2,Q.x2);
    R.y2=min(P.y2,Q.y2);
    return R;
}

//-----------------------Geometry
```

26 HLD

```
namespace HLD{
  ///lazy propagation
  template<class T>
  class SegTree{
  public:
    T* sgt;
    ///combine must clear out any unpropagated value
    T (*combine)(T,T);
    T (*propagate)(T to,T from,int);
    int n;
    SegTree(int sz,T(*combine)(T,T),
        T(*propagate)(T,T,int),T* data=NULL){
      this->combine=combine;
      this->propagate=propagate;
      n=sz;
      sgt=new T[4*sz];
      if(data!=NULL)
        build(1,0,n-1,data);}
    void build(int v,int vl, int vr,T* data){
      if(vl==vr){
        sgt[v]=data[vl];
        return;}
      int mid=(vl+vr)/2;
      build(2*v,vl,mid,data);
      build(2*v+1,mid+1,vr,data);
      sgt[v]=combine(sgt[2*v], sgt[2*v+1]);}
    void update(int v,int vl,int vr,int l,int r,T unp){
      if(vl==l&&vr==r){
        sgt[v]=propagate( sgt[v],unp,r-l+1);
        return;}
      int mid=(vl+vr)/2;
      sgt[2*v]=propagate( sgt[2*v],sgt[v],mid-vl+1);
      sgt[2*v+1]=propagate( sgt[2*v+1],sgt[v],vr-mid);
      if(r<=mid)
        update(2*v,vl,mid,l, r,unp);
      else if(l>mid)
        update(2*v+1,mid+1,vr, l,r,unp);
      else{
        update(2*v,vl,mid,l, mid,unp);
        update(2*v+1,mid+1,vr, mid+1,r,unp);}
      sgt[v]=combine(sgt[2*v], sgt[2*v+1]);}
    void update(int l,int r,T unp){
      update(1,0,n-1,l,r,unp);}
    T query(int v,int vl,int vr,int l,int r){
      if(vl==l&&vr==r){
        return sgt[v];}
      int mid=(vl+vr)/2;
      sgt[2*v]=propagate( sgt[2*v],sgt[v],mid-vl+1);
      sgt[2*v+1]=propagate( sgt[2*v+1],sgt[v],vr-mid);
      sgt[v]=combine(sgt[2*v], sgt[2*v+1]);
      if(r<=mid)
        return query(2*v,vl, mid,l,r);
      else if(l>mid)
        return query(2*v+1, mid+1,vr,l,r);
      else
        return combine( query(2*v,vl,mid,l,mid),
            query(2*v+1,mid+1,vr,mid+1,r));}
    T query(int l,int r){
      return query(1,0,n-1,l,r);}
    ~SegTree(){
      if(n!=0&&sgt!=NULL)
        delete[] sgt;}};
  struct Node{
    int mn=INT_MAX,unp=INT_MAX;
    Node(){}
    Node(int mn,int unp): mn(mn),unp(unp){}};
  inline Node combine(Node a,Node b){
    return {min(a.mn,b.mn),INT_MAX};}
  inline Node propagate(Node to,Node from,int len){
    if(from.unp==INT_MAX)
      return to;
    to.mn=min(to.mn,from.unp);
    to.unp=min(to.unp,from.unp);
    return to;}
  #define MAX_SIZE 100001
  vector<vector<int> >G;
  vector<int> parent,depth, heavy,head,pos;
  vector<int> node_val;
  vector<int> node_val_order;
  SegTree<Node> sgt( MAX_SIZE,combine,propagate);
  int cur_pos;
  int dfs(int node){
    int sz=1;
    int max_c_size=0;
    for(auto c:G[node]){
      if(c!=parent[node]){
        parent[c]=node;
        depth[c]=depth[node]+1;
        int c_size=dfs(c);
        sz+=c_size;
        if(c_size>max_c_size){
          max_c_size=c_size;
          heavy[node]=c; }}}
    return sz;}
  void decompose(int node,int h){
    pos[node]=cur_pos++;
    head[node]=h;
    if(heavy[node]!=-1)
      decompose(heavy[node],h);
    for(int c:G[node]){
      if(c!=parent[node]&& c!=heavy[node])
        decompose(c,c);}
    return;}
  //for query on path the node_val of a node is the cost
      of the edge to parent
  //exclude=true for query on path,it excludes the value
      stored in lca(a,b)
  int query(int a,int b,int n,bool exclude=false){//n
      number of node in the tree
    Node res;//not really generalized,for min max update
        accordingly
    while(head[a]!=head[b]){
      if(depth[head[a]]> depth[head[b]])swap(a,b);
      res=combine(res, sgt.query(pos[head[b]],pos[b]));
      b=parent[head[b]];}
    if(depth[a]>depth[b])swap(a,b);
    res=combine(res,exclude? sgt.query(pos[a]+1,pos[b])
        :sgt.query(pos[a],pos[b]));
    return res.mn;}
//  void update(int node,int val)
//  {
//    sgt.update(pos[node],val);
//  }
  void update(int a,int b,int val){
    while(head[a]!=head[b]){
      if(depth[head[a]]> depth[head[b]])
        swap(a,b);
      //res=combine(res,sgt.query(pos[head[b]],pos[b]));
      sgt.update(pos[head[b]], pos[b],{0,val});
      b=parent[head[b]];
    }
    if(depth[a]>depth[b])swap(a,b);
    //res=combine(res,exclude? sgt.query(pos[a]+1,
        pos[b]): sgt.query(pos[a],pos[b]));
    sgt.update(pos[a],pos[b],{0,val});
    //return res.mn;}
  void init(int n){
    parent.resize(n);
    depth.resize(n);
    heavy.assign(n,-1);
    head.resize(n);
    pos.resize(n);
    parent[0]=0;//might change later
    dfs(0);
    cur_pos=0;
    decompose(0,0);
    node_val_order.resize(n);
    for(int i=0;i<n;++i){
      node_val_order[pos[i]]= node_val[i];}
    sgt.n=n;
    //sgt.build(node_val_order, 1,0,n-1); }}
```

27 Hungarian

```
#define MAX 666
#define MAXIMIZE +1
#define MINIMIZE -1
#define inf (~0U >> 1)
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) -
    (a) + 1)) + (a))
namespace wm{ /// hash = 581023
  bool visited[MAX];
  int U[MAX], V[MAX], P[MAX], way[MAX], minv[MAX],
      match[MAX], ar[MAX][MAX];
  /// n = number of row and m = number of columns in 1
      based, flag = MAXIMIZE or MINIMIZE
  /// match[i] contains the column to which row i is
      matched
  int hungarian(int n, int m, int mat[3][3], int flag){
    clr(U), clr(V), clr(P), clr(ar), clr(way);
    for (int i = 1; i <= n; i++){
      for (int j = 1; j <= m; j++){
        ar[i][j] = mat[i][j];
        if (flag == MAXIMIZE) ar[i][j] = -ar[i][j];}}
    if (n > m) m = n;
    int i, j, a, b, c, d, r, w;
    for (i = 1; i <= n; i++){
      P[0] = i, b = 0;
      for (j = 0; j <= m; j++) minv[j] = inf, visited[j]
          = false;
      do{
        visited[b] = true;
        a = P[b], d = 0, w = inf;
        for (j = 1; j <= m; j++){
          if (!visited[j]){
            r = ar[a][j] - U[a] - V[j];
            if (r < minv[j]) minv[j] = r, way[j] = b;
            if (minv[j] < w) w = minv[j], d = j;}}
        for (j = 0; j <= m; j++){
          if (visited[j]) U[P[j]] += w, V[j] -= w;
          else minv[j] -= w;}
        b = d;
      } while (P[b] != 0);
      do{
        d = way[b];
        P[b] = P[d], b = d;
      } while (b != 0);}
    for (j = 1; j <= m; j++) match[P[j]] = j;
    return (flag == MINIMIZE) ? -V[0] : V[0];}}
```

28 Josephus Problem

```
ll Josephus(ll n, ll k, ll m){
    m = n - m;
    if (k <= 1) return n - m;
    ll i = m;
    while (i < n){
        ll r = (i - m + k - 2) / (k - 1);
        if ((i + r) > n) r = n - i;
        else if (!r) r = 1;
        i += r;
        m = (m + (r * k)) % i;
    }
    return m + 1;
}
```

29 KMP

```
int failure[1000001];
void build_failure(string &str){
  int i,j,k;
  int cur;
  memset(failure,0,sizeof failure);
  failure[0]=0;
  failure[1]=0;
  for(i=2;i<=str.length();i++){
    cur=i-1;
    while(cur!=0){
       if(str[failure[cur]] == str[i-1]){
        failure[i]= failure[cur]+1;
        break;}
       cur=failure[cur];}}}
int kmp(string &text,string &pat){
  int i,j,k;
  int cur=0;
  int ocur=0;
  for(i=0;i<text.length();i++){
    if(cur==pat.length()){
      ocur++;}
    while(cur and text[i]!= pat[cur])
        cur= failure[cur];
    if(text[i]==pat[cur])
      cur++;}
  if(cur==pat.length())
    ocur++;
  return ocur;}
void z_function(const string& str, vector<int>& z){
  int sz=str.size();
  z[0]=0;
  int i,l=0,r=0,j,k;
  for(i=1;i<sz;i++){
    z[i]=0;
    if(i<=r){
      z[i]=min(r-i+1, z[i-l]);}
    while(i+z[i]<sz&&str[z[i]] == str[i+z[i]])z[i]++;
    if(i+z[i]-1>r){
      l=i;
      r=i+z[i]-1;}}
  return;}
```

30 Knuth Optimization

```
ll dp[1002][1002];
int indx[1002][1002];
int d[1002];
int cd[1002];
//assumes d and cd (cumulative sum) are calculated and 1
    indexed
void knuthArraySplitting(int n){
for(int i=1;i<=n;++i){
  dp[i][i]=0;
  indx[i][i]=i;}
for(int l=2;l<=n;++l){
  int i=1,j=l;
  while(j<=n){
    dp[i][j]=LLONG_MAX;
    for(int k= indx[i][j-1];k<= indx[i+1][j];++k){
      if(dp[i][k]+ dp[k+1][j]+ cd[j]- cd[i-1]<dp[i][j]){
        dp[i][j]=dp[i][k]+ dp[k+1][j]+ cd[j]-cd[i-1];
        indx[i][j]=k;}}
      ++i;++j;}}}
```

31 LCA

```
const int MAXN=100001, LOG=20;
vector<vector<int>> g;
int n;
int bparent[MAXN][LOG], depth[MAXN],vis[MAXN];
void dfs(int a){
  vis[a]=1;
  for(auto v: g[a]){
    if(!vis[v]){
      bparent[v][0]=a;
      depth[v]=1+depth[a];
      dfs(v);}}}
void build_ancestor(){
  dfs(1);
  for(int i=1;(1<<i)<n;i++)
    for(int j=1;j<=n;j++)
      bparent[j][i]=bparent[bparent[j][i-1]][i-1];}
int pth_ancestor(int a,int p){
  for(int i=0;(1<<i)<=p;i++)
    if((1<<i)&p)
      a=bparent[a][i];
  return a;
}
int lca(int u,int v){
  if(depth[v]>depth[u])
    v=pth_ancestor(v,depth[v]-depth[u]);
  if(depth[u]>depth[v])
    u=pth_ancestor(u,depth[u]-depth[v]);
  if(u==v)
    return u;
  //printf("%d %d\n",u,v);
  for(int i=log2(n-1);i>=0;i--){
    if(bparent[u][i]!=bparent[v][i]){
      u=bparent[u][i];
      v=bparent[v][i];}}
  return bparent[u][0];}
```

32 LIS

```
/*****************************************/
///out[i] contain length of longest increasing sequence
///ending at index i(0 based)
/*****************************************/
void LIS(vector<int> & a,vector<int>& out){
  int n=a.size();
  int inf=1e9+1;
  vector<int> dp(n+1,INT_MAX);
  dp[0]=-INT_MAX;
  int i;
  for(i=0;i<n;i++){
    int j=upper_bound(dp.begin(), dp.end(),
        a[i])-dp.begin();
    int k=lower_bound(dp.begin(), dp.end(),
        a[i])-dp.begin();
    out[i]=k;
    if(dp[j-1]<a[i]&&a[i]<dp[j]){
      dp[j]=a[i];}}}
```

33 Linear Diophantine Equation

```
template <class T>pair<T,T> extended_euclid(T n,T
    m){//returns <u,v> such that nu+mv=(n,m)
  T rn_1,rn,sn_1,sn,tn_1,tn,tr,ts,tt,q;
  rn_1=n;sn_1=1;tn_1=0;
  rn=m;sn=0;tn=1;
  while(1){
    tr=rn_1%rn;
    q=(rn_1-tr)/rn;
    ts=sn_1-(q*sn);
    tt=tn_1-(q*tn);
    if(tr==0){
      return mp(sn,tn);
      //return (sn+m)%m;}
    sn_1=sn;sn=ts;
    tn_1=tn;tn=tt;
    rn_1=rn;rn=tr;}}
```

```
///asuming the line equation in form ax+by=c
///assuming at least one of a or b is not zero,check it
    before passing to this function
ll shiftx(ll x,ll refx,ll q){
  if(x==refx) return x;
  if(x<refx){
    ll d=refx-x;
    if(d%q){ d=d-d%q+q;}
    return x+d;}
  else{ll d=x-refx;
    if(d%q)
      d-=d%q;
    return x-d;}}
ll solveDiophantine(ll a,ll b,ll c,ll x1,ll x2,ll y1,ll
    y2){
  if(a==0){
    if(abs(c)%abs(b))return 0;
    ll y=c/b;
    if(y>=y1&&y<=y2)
      return (x2-x1+1);
    return 0;}
  if(b==0){
    if(abs(c)%abs(a)) return 0;
    ll x=c/a;
    if(x>=x1&&x<=x2)
      return (y2-y1+1);
    return 0;}
  ll g=__gcd(abs(a),abs(b));
  if(abs(c)%g){return 0;}
  pair<ll,ll> sol= extended_euclid(abs(a),abs(b));
  if(a<0){a=-a; b=-b; c=-c;}
  if(b<0){sol.second=-sol.second;}
  sol.first*=(c/g);
  sol.second*=(c/g);
  ll x1p,x2p;
  //slope is positive
  if(b<0){
    x1p=floor(1.0*(c-b*y1)/a);
    if((c-b*y1)%a!=0)x1p++;
    x2p=floor(1.0*(c-b*y2)/a);}
  //slope is negative
  else{
    x2p=floor(1.0*(c-b*y1)/a);
    x1p=floor(1.0*(c-b*y2)/a);
    if((c-b*y2)%a!=0)
      x1p++;}
  ll x11,x22;
  x11=max(x1,x1p);
  x22=min(x2,x2p);
  if(x22<x11)return 0;
  ll lx,rx;
  lx=shiftx(sol.first,x11,abs(b/g));
  rx=shiftx(sol.first,x22,abs(b/g));
  if(rx!=x22){rx-=abs(b/g);}
  return max((rx-lx)/(abs(b/g))+ 1, 0ll); }
void solvecases(int cse){
  ll a,b,c,x1,x2,y1,y2;
  cin>>a>>b>>c>>x1>>x2>>y1>>y2;
  ll sol;
  if(a==0&&b==0){
    if(c==0)
      sol=(x2-x1+1)*(y2-y1+1);
    else
      sol=0;}
  else{
    sol=solveDiophantine(a,b,-c, x1,x2,y1,y2);}
  cout<<"Case "<<cse<<": "<<sol<<'\n';}
```

34 MOs Algorithm

```
// 1 indexed on MO array
const int N = 300007;
int BLOCK = 500;
int cnt[N];
int res = 0;
void add(int n){
    if( cnt[n] == 0 ) res++;
    cnt[n]++;
}
void rem(int n){
    cnt[n]--;
    if( cnt[n] == 0 ) res--;
}
int main(){
    FASTIO;
    int n = 10;
    //cin>>n;
    int MO[] = {0,1,5,2,4,1,6,7,2,2,2}; // 1 indexed
    //for(int i=1;i<=n;i++) cin>>ara[i]; // 1 indexed
    int q;
    cin>>q;
    pair<pii,pii> queries[q]; // start,end,index, ans
    for(int i=0;i<q;i++){
        int a,b;
        cin>>a>>b;
        queries[i] = {{a,b},{i,0}};
    }
    sort(queries,queries+q,[](pair<pii,pii>
         A,pair<pii,pii> B){
         return A.F.F/BLOCK == B.F.F/BLOCK ?
             A.F.S<B.F.S: A.F.F<B.F.F;
         });
    int l_it =1,r_it = 0;
    for( int i=0;i<q;i++ ){
        int L = queries[i].F.F;
        int R = queries[i].F.S;
        while( r_it<R ) add( MO[++r_it] );
        while( l_it>L ) add( MO[ --l_it ] );
        while( r_it>R ) rem( MO[r_it--] );
        while( l_it<L ) rem( MO[l_it++] );
        queries[i].S.S = res;
       // cout<<res<<endl;
    }
    sort( queries,queries+q,[](pair<pii,pii>
         A,pair<pii,pii> B){
         return A.S.F<B.S.F;
         });
    for(int i=0;i<q;i++){
        cout<<queries[i].S.S<<endl;
    }
}
```

35 Manacher

```
int pal[100001];
string str_mod(string str){
  string mstr;
  mstr.push_back('#');
  for(int i=0;i < str.length(); i++){
    mstr.push_back(str[i]);
    mstr.push_back('#');}
  return mstr;}
void manacher(string str){
  int l,r;
  int lc=0,rc=0;
  int lt,rt;
  int mir=0;
  string mstr=str_mod(str);
  pal[0]=0;
  for(r=1;r<mstr.length(); r++){
    l=2*mir-r;
    if(l>=0 && pal[l]< (rc-r))
      pal[r]= pal[l];
    else {
      mir=r;
      pal[mir]=max(0,rc-mir);
      for(rt=max(r+1,rc+1),lt=2*mir-rt;lt>=0 &&
          rt<mstr.size() &&
          mstr[lt]==mstr[rt];rt++,lt=2*mir-rt){
        pal[mir]++;}
      rc=rt-1;
      lc=lt+1;}}}
```

36 Matrix Multiplication

```
ll** matMul(ll n1, ll n2, ll n3, ll** L1, ll** L2, ll
    mod){
       ll** M = new ll*[n1];
       for(ll i=0; i<n1; i++) M[i]=new ll[n3];

       for(ll i=0; i<n1; i++){
       for(ll j=0; j<n3; j++){
              M[i][j]=0;
              for(ll k=0; k<n2; k++){
                     M[i][j]+= (L1[i][k]*L2[k][j])%mod;
                     M[i][j]%=mod;
              }
       }
       }

       return M;
}

ll** matExpo(ll m, ll n, ll** L, ll mod){
       ll** M = new ll*[n];
       for(ll i=0; i<n; i++){
              M[i]=new ll[n];
              for(ll j=0; j<n; j++){
                     M[i][j]=0;
              }
              M[i][i]=1;
       }

       if(m==0) return M;

       M=matExpo(m/2,n,L,mod);
       M=matMul(n,n,n,M,M,mod);
       if(m%2==1) M=matMul(n,n,n,M,L,mod);
       return M;
}

ll fibonacci(ll n, ll mod){
       if(n<2) return n;
       n--;
       ll** init = new ll*[2];
       for(ll i=0; i<2; i++) init[i]=new ll[1];
       init[0][0]=1;
       init[1][0]=0;

       ll** fibo = new ll*[2];
       for(ll i=0; i<2; i++) fibo[i]=new ll[2];
       fibo[0][0]=1;
       fibo[0][1]=1;
       fibo[1][0]=1;
       fibo[1][1]=0;

       fibo = matExpo(n,2,fibo,mod);
       init = matMul(2,2,1,fibo,init,mod);
       //cout<< init[0][0]<< " "<< init[1][0]<< endl;
       return init[0][0];
}
```

37 MaxFlowMinCost

```
namespace mcmf {
  typedef long long F; typedef long long C;
  const F infF = 1e18; const C infC = 1e18;
  const int N = 5005;
  typedef pair<C, F> PCF;
  struct Edge {int frm, to; C cost; F cap, flow;};
  int n, s, t;
  vector<Edge> edges;
  vector<int> adj[N];
  C pi[N], dis[N];
  F fl[N];
  int prv[N], vis[N];
  void init(int nodes, int source, int sink) {
    n = nodes, s = source, t = sink;
    for (int i=0; i<n; i++) pi[i] = 0, adj[i].clear();
    edges.clear();}
  void addEdge(int u, int v, F cap,C cost) {
    edges.push_back({u, v, cost, cap, 0});
    edges.push_back({v, u, -cost, 0, 0});
    adj[u].push_back(edges.size()-2);
    adj[v].push_back(edges.size()-1);}
  bool SPFA() {
    for (int i=0; i<n; i++) {
      dis[i] = infC; fl[i] = 0;
      vis[i] = 0; prv[i] = -1;}
    queue<int> q;
    q.push(s);
    dis[s] = 0; fl[s] = infF; vis[s] = 1;
    while (!q.empty()) {
      int u = q.front(); q.pop();
      vis[u] = 0;
      for (int eid : adj[u]) {
        Edge &e = edges[eid];
        if (e.cap == e.flow) continue;

        if (dis[u] + e.cost < dis[e.to]) {
          dis[e.to] = dis[u] + e.cost;
          fl[e.to] = min(fl[u], e.cap - e.flow);
          prv[e.to] = eid^1;
          if (!vis[e.to]) q.push(e.to);}}}
    return fl[t] > 0;}
  PCF solveSPFA() {
    C cost = 0; F flow = 0;
    while (SPFA()) {
      C pathcost = dis[t];
      cost += pathcost*fl[t]; flow += fl[t];
      for (int u=t, e=prv[u]; e!=-1; u=edges[e].to,
          e=prv[u]) {
        edges[e].flow -= fl[t];
        edges[e^1].flow += fl[t];}}
    return {cost, flow};}
  void normalize() {
    SPFA();
    for (int i=0; i<n; i++) pi[i] = dis[i];}
  bool Dijkstra() {
    for (int i=0; i<n; i++) {
      dis[i] = infC; fl[i] = 0;
      vis[i] = 0; prv[i] = -1;}
    priority_queue<pair<C, int>> pq;
    pq.emplace(0, s);
    dis[s] = 0; fl[s] = infF;
    while (!pq.empty()) {
      int u = pq.top().second; pq.pop();
      if (vis[u]) continue;
      vis[u] = 1;
      for (int eid : adj[u]) {
        Edge &e = edges[eid];
        if (vis[e.to] || e.cap == e.flow) continue;
        C nw = dis[u] + e.cost - pi[e.to] + pi[u];
        if (nw < dis[e.to]) {
          dis[e.to] = nw;
          fl[e.to] = min(fl[u], e.cap - e.flow);
          prv[e.to] = eid^1;
          pq.emplace(-dis[e.to], e.to);}}}
    return fl[t] > 0;}
  PCF solveDijkstra() {
    normalize();
    C cost = 0; F flow = 0;
    while (Dijkstra()) {
      for (int i=0; i<n; i++)
        if (fl[i])  pi[i] += dis[i];
      C pathcost = pi[t]-pi[s];
      cost += pathcost*fl[t]; flow += fl[t];

      for (int u=t, e=prv[u]; e!=-1; u=edges[e].to,
          e=prv[u]) {
        edges[e].flow -= fl[t];
        edges[e^1].flow += fl[t];}}
    return {cost, flow};}}
```

38 Mobius

```
int mo[N];
void mobius(){
    memset(mo, -1, sizeof mo);
    mu[1] = 1;
    for(int i = 2; i < N; i++)
    for(int j = (i << 1) ; j < N; j += i)
    mo[j] -= mo[i];
}
```

39 NT Mashroor

```
inverse modulo;
r[1] = 1;
    for(ll i=2; i<1000009; ++i) r[i] = (mod  (mod/i) *
        r[mod%i] % mod) % mod;

ll mod=1000000007, px=29;
ll binpow(ll a, ll b){
    if(b==0) return 1;
    ll k=binpow(a,b/2);
    k*=k; k%=mod;
    if(b%2==1) k*=a;
    return k%mod;
}
ll modinv(ll a){
    return binpow(a,mod-2);
}
ll moddiv(ll a, ll b){
    return (a*modinv(b))%mod;
}
ll hashconcat(char c, ll n, ll h, bool before){
    ll k=(c-'a');
    if(before){
        k*=binpow(px,n-1);
        k%=mod;
    }
    else{
        h*=px;
        h%=mod;
    }
    return (h+k)%mod;
}
ll hashdel(char c, ll n, ll h, bool before){
    ll k=(c-'a');
    if(before){
        k*=binpow(px,n-1);
        k%=mod;
        h=(h-k+mod)%mod;
    }
    else{
        h-=k;
        h=moddiv(h,px);
    }
    return h;
}
ll hashstr(string s){
    ll h=0;
    ll n=s.size();
    for(ll i=0; i<n; i++){
        h=hashcat(s[i],i+1,h,false);
    }
    return h;
}

bool isp[1000009]={false};
ll fct[1000009];

void soe(){
    for(ll i=2; i<1000009; i++){
        if(!isp[i]){
            fct[i]=i;
            for(ll j=i*i; j<1000009; j+=i){
                if(!isp[j]) fct[j]=i;
                isp[j]=true;
            }
        }
    }
    return;
}

vector<ll> divpath(ll n){
    vector<ll> path;
    while(n>1){
        path.push_back(fct[n]);
        n/=fct[n];
    }
    if(path.size()>1) sort(path.begin(),path.end());
    return path;
}

vector< pll > eulerform(ll n){
    vector<ll> path=divpath(n);
    vector< pll > yes;

    if(path.size()==0) return yes;
    ll j=1;
    for(ll i=1; i<path.size(); i++){
        if(path[i]!=path[i-1]){
            yes.push_back(mp(path[i-1],j));
            j=1;
        }
        else j++;
    }
    yes.push_back(mp(path[path.size()-1],j));
    return yes;
}

ll noofdiv(ll n){
    ll ans=1;
    vector<ll> path=divpath(n);
    if(path.size()==0) return 1;
    ll j=1;
    for(ll i=1; i<path.size(); i++){
        if(path[i]!=path[i-1]){
            ans*=(j+1);
            j=1;
        }
```

```
        else j++;
    }
    ans*=(j+1);
    return ans;
}

ll ei(ll a){
    return binpow(a%mod,mod-2);
}
ll ed(ll a, ll b){
    return (a%mod *ei(b))%mod;
}
ll em(ll a, ll b){
    return (a%mod * b%mod)%mod;
}
ll ep(ll a, ll b){
    return (a+b)%mod;
}
ll es(ll a, ll b){
    return (a+mod-b)%mod;
}

vector<ll> pfs(ll n){
    vector<ll> p;
    for(ll i=2; i*i<=n; i++){
        if(n%i==0){
            p.push_back(i);
            while(n%i==0) n/=i;
        }
    }
    if(n>1) p.push_back(n);
    return p;
}

ll truephi(ll n, vector<ll> p){
    ll m=p.size();
    ll h=1<<m;
    ll ans=n;
    //double a1=log(n), a2;
    for(ll i=0; i<h; i++){
        ll o=0, d=1, g=1;
        //a2=0;
        for(ll j=0; j<m; j++){
            if((i&g)!=0){
                o++;
                //a2+=log(p[j]);
                d*=p[j];
            }
            g+=g;
        }
        (o%2==1)? ans+=(n/d) : ans-=(n/d);
    }
    return ans;
}


totient values of all numbers in almost O(n):

ll phi[N];

void findphi(){
    for(ll i=0; i<N; i++) phi[i]=i;
    for(ll i=2; i<N; i++){
        if(phi[i]==i){
            for(ll j=1; j*i<N; j++){
                phi[j*i] = (phi[j*i]/i)*(i-1);
            }
        }
    }
    phi[0]=0;
    phi[1]=0;
    for(ll i=1; i<N; i++){
        //if(i<=20) cout<< i<< " "<< phi[i]<< endl;
        phi[i]=phi[i]*phi[i] + phi[i-1];
    }
    return ;
}
```

40 NTT

```
/**
Iterative Implementation of Number Theoretic Transform
Complexity: O(N log N)

7340033 = 7 * 2^20, G = 3
645922817 = 77 * 2^23, G = 3
897581057 = 107 * 2^23, G = 3
998244353 = 119 * 2^23, G = 3
Author: anachor
**/
namespace NTT {
  vector<int> perm, wp[2];
  const int mod = 998244353, G = 3; ///G is the
      primitive root of M
  int root, inv, N, invN;
  int power(int a, int p) {
    int ans = 1;
    while (p) {
      if (p & 1) ans = (1LL*ans*a)%mod;
      a = (1LL*a*a)%mod;
      p >>= 1;}
    return ans;}
void precalculate(int n) {
    assert( (n&(n-1)) == 0 && (mod-1)%n==0);
    N = n;
    invN = power(N, mod-2);
    perm = wp[0] = wp[1] = vector<int>(N);

    perm[0] = 0;
    for (int k=1; k<N; k<<=1)
      for (int i=0; i<k; i++) {
        perm[i] <<= 1;
        perm[i+k] = 1 + perm[i];}
  root = power(G, (mod-1)/N);
    inv = power(root, mod-2);
    wp[0][0]=wp[1][0]=1;
    for (int i=1; i<N; i++) {
      wp[0][i] = (wp[0][i-1]*1LL*root)%mod;
      wp[1][i] = (wp[1][i-1]*1LL*inv)%mod;}}
void fft(vector<int> &v, bool invert = false) {
    if (v.size() != perm.size()) precalculate(v.size());
    for (int i=0; i<N; i++)
      if (i < perm[i])
        swap(v[i], v[perm[i]]);
  for (int len = 2; len <= N; len *= 2) {
      for (int i=0, d = N/len; i<N; i+=len) {
        for (int j=0, idx=0; j<len/2; j++, idx += d) {
          int x = v[i+j];
          int y = (wp[invert][idx]* 1LL*v[i+j+len/2])%mod;
          v[i+j] = (x+y>=mod ? x+y-mod : x+y);
          v[i+j+len/2] = (x-y>=0 ? x-y : x-y+mod);}}}
    if (invert) {
      for (int &x : v) x = (x*1LL*invN)%mod;}}
  vector<int> multiply(vector<int> a, vector<int> b) {
    int n = 1;
    while (n < a.size()+ b.size()) n<<=1;
    a.resize(n);
    b.resize(n);
    fft(a);
    fft(b);
    for (int i=0; i<n; i++) a[i] = (a[i] * 1LL *
        b[i])%mod;
    fft(a, true);
    return a;}};
///Solves https://old.yosupo.jp/problem/convolution_mod
const int M = 998244353;
int main() {
  ios::sync_with_stdio(0);
  cin.tie(0);
  int n, m;
  cin>>n>>m;
  vector<int> a(n), b(m);
  for (int i=0; i<n; i++) cin>>a[i];
  for (int i=0; i<m; i++) cin>>b[i];
  vector<int> c = NTT::multiply(a, b);
  c.resize(n+m-1);
  for (int x: c) cout<<x<<" ";}
```

41 OOP String Hash

```
mt19937 rng(chrono ::steady_clock ::now().
    time_since_epoch(). count());
const ll MAXN=1000001,hp=31+rng()%20, mod=1000004023;
ll power[MAXN],ipower[MAXN];
void ipower_build();
class Hush{
public:
  ll *hf;string str;
  Hush(string &str): str(str)
  {hf=new ll[str.length()];hf[0]=str[0]-'a'+1;
  for(ll i=1;i<str.length();i++)
    hf[i]=(hf[i-1]+ (str[i]-'a'+1)* power[i])%mod;}
  ll getHash(){return hf[str.length()-1];}
  ll getHash(ll i1,ll i2){
    if(i1==0)return hf[i2];
    return ((hf[i2]-hf[i1-1]+mod)* ipower[i1])%mod;
  }};
unsigned seed = chrono ::system_clock ::now().
    time_since_epoch(). count();
linear_congruential_engine<uint_fast32_t, 1002517UL,
    1001593UL, 2147483647UL> lcg(seed);
int primes_for_mod[]={1000019353, 1000001087,
    1000020353, 1000003267, 1000000439, 1000018001,
    1000019569, 1000020701, 1000016929, 1000007521,
1000007773, 1000013323, 1000018379, 1000017203,
    1000006211, 1000004693, 1000013011, 1000020829,
    1000011277, 1000007147};
int primes_for_base[]={1831, 1061, 5927, 6689, 7529,
    9719, 3917, 271, 6029, 6091, 9719, 2819, 4877,
    9679, 6373, 6101, 1039, 4591, 5531};
```

42 OST builtin

```
typedef tree < int, null_type, less < int >,
    rb_tree_tag, tree_order_statistics_node_update >
    o_set;
typedef tree < pair < int, int >, null_type, less < pair
    < int, int > >, rb_tree_tag,
    tree_order_statistics_node_update > o_setp;
 o_set s;

/**
the tree supports the same operations as the set
but also there are two new features
find_by_order() : The first returns an iterator to the
    k-th largest element (counting from zero)
order_of_key() : The number of items in a set that are
    strictly smaller than our item.
*/
/**
To store duplicate values:
Store as {key,index}
To search search all by {key,Max value
```

```
or we can also search in a range [0,index] i guess using
    {key,index}
*/
s.insert(1);
cout<<*s.find_by_order(1)<<endl; // 2
cout<<(end(s)==s.find_by_order(6))<<endl;
cout<<s.order_of_key(-5)<<endl;
cout<<s.size()<<endl;
s.erase(2);
```

43 Palindromic Tree

```
/*-> diff(v) = len(v) - len(link(v))
-> series link will lead from the vertex v to the vertex
    u corresponding
  to the maximum suffix palindrome of v which satisfies
      diff(v) != diff(u)
-> path within series links to the root contains only
    O(log n) vertices
-> cnt contains the number of palindromic suffixes of
    the node*/
struct PalindromicTree {
 struct node {
  int nxt[26], len, st, en, link, diff, slink, cnt, oc;
 };
 string s;vector<node> t;
 int sz, last;
 PalindromicTree() {}
 PalindromicTree(string _s) {
  s = _s;
  int n = s.size();
  t.clear();
  t.resize(n + 9);sz = 2, last = 2;
  t[1].len = -1, t[1].link = 1;
  t[2].len = 0, t[2].link = 1;
  t[1].diff = t[2].diff = 0;
  t[1].slink = 1;t[2].slink = 2;
 }
 int extend(int pos) { // returns 1 if it creates a new
     palindrome
  int cur = last, curlen = 0;
  int ch = s[pos] - 'a';
  while (1) {curlen = t[cur].len;
   if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] ==
       s[pos]) break;
   cur = t[cur].link;
  }
  if (t[cur].nxt[ch]) {last = t[cur].nxt[ch];
   t[last].oc++;
   return 0;
  }
  sz++;last = sz;
  t[sz].oc = 1;t[sz].len = t[cur].len + 2;
  t[cur].nxt[ch] = sz;t[sz].en = pos;
  t[sz].st = pos - t[sz].len + 1;
  if (t[sz].len == 1) {
   t[sz].link = 2;t[sz].cnt = 1;
   t[sz].diff = 1;t[sz].slink = 2;
   return 1;
  }
  while (1) {
   cur = t[cur].link;curlen = t[cur].len;
   if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] ==
       s[pos]) {
    t[sz].link = t[cur].nxt[ch];
    break;
   }
  }
  t[sz].cnt = 1 + t[t[sz].link].cnt;
  t[sz].diff = t[sz].len - t[t[sz].link].len;
  if (t[sz].diff == t[t[sz].link].diff) t[sz].slink =
      t[t[sz].link].slink;
  else t[sz].slink = t[sz].link;
  return 1;
 }
 void calc_occurrences() {
  for (int i = sz; i >= 3; i--) t[t[i].link].oc +=
      t[i].oc;
 }
 vector<array<int, 2>> minimum_partition() { //(even,
     odd), 1 indexed
  int n = s.size();
  vector<array<int, 2>> ans(n + 1, {0, 0}), series_ans(n
      + 5, {0, 0});
  ans[0][1] = series_ans[2][1] = 1e9;
  for (int i = 1; i <= n; i++) {
   extend(i - 1);
   for (int k = 0; k < 2; k++) {
    ans[i][k] = 1e9;
    for (int v = last; t[v].len > 0; v = t[v].slink) {
     series_ans[v][!k] = ans[i - (t[t[v].slink].len +
         t[v].diff)][!k];
     if (t[v].diff == t[t[v].link].diff)
         series_ans[v][!k] = min(series_ans[v][!k],
         series_ans[t[v].link][!k]);
     ans[i][k] = min(ans[i][k], series_ans[v][!k] + 1);
    }
   }
  }
  return ans;
 }
} t;
int32_t main() {
 string s;cin >> s;
 PalindromicTree t(s);
 for (int i = 0; i < s.size(); i++) t.extend(i);
 t.calc_occurrences();
 long long ans = 0;
 for (int i = 3; i <= t.sz; i++) ans += t.t[i].oc;
 cout << ans << '\n';
 //auto ans = t.minimum_partition();
 // for (int i = 1; i <= s.size(); i++) {
 //   cout << (ans[i][1] == 1e9 ? -1 : ans[i][1]) << ' ';
 //   cout << (ans[i][0] == 1e9 ? -2 : ans[i][0]) <<
     '\n';
 // }
}
```

44 Pollard Rho

```
typedef long long LL;
typedef unsigned long long ULL;
namespace Rho {
  ULL mult(ULL a, ULL b, ULL mod) {
    LL ret = a * b - mod * (ULL)(1.0L / mod * a * b);
    return ret + mod * (ret < 0) - mod * (ret >= (LL)
        mod);}
  ULL power(ULL x, ULL p, ULL mod){
    ULL s=1, m=x;
    while(p) {
      if(p&1) s = mult(s, m, mod);
      p>>=1;
      m = mult(m, m, mod);}
    return s;}
  vector<LL> bases = {2, 325, 9375, 28178, 450775,
      9780504, 1795265022};
  bool isprime(LL n) {
    if (n<2) return 0;
    if (n%2==0)  return n==2;
    ULL s = __builtin_ctzll(n-1), d = n>>s;
    for (ULL x: bases) {
      ULL p = power(x%n, d, n), t = s;
      while (p!=1 && p!=n-1 && x%n && t--) p = mult(p,
          p, n);
      if (p!=n-1 && t != s) return 0;}
    return 1;}
  ///Returns a proper divisor if n is composite, n
      otherwise
  ///Possible Optimization: use binary gcd for ~10%
      speedup
  mt19937_64 rng(chrono ::system_clock ::now().
      time_since_epoch() .count());
  ULL FindFactor(ULL n) {
    if (n == 1 || isprime(n)) return n;
    ULL c = 1, x = 0, y = 0, t = 0, prod = 2, x0 = 1, q;
    auto f = [&](ULL X) { return mult(X, X, n) + c;};
    while (t++ % 128 or gcd(prod, n) == 1) {
      if (x == y) c = rng()%(n-1)+1, x = x0, y = f(x);
      if ((q = mult(prod, max(x, y) - min(x, y), n)))
          prod = q;
      x = f(x), y = f(f(y));}
    return gcd(prod, n);}
  ///Returns all prime factors
  vector<ULL> factorize(ULL x) {
    if (x == 1)  return {};
    ULL a = FindFactor(x), b = x/a;
    if (a == x) return {a};
    vector<ULL> L = factorize(a), R = factorize(b);
    L.insert(L.end(), R.begin(), R.end());
    return L;}}
int main() {
  long long x;
  vector<ULL> ans = Rho::factorize(x);}
```

45 Primality Test

```
//able to compute a*b%mod a,b<2^63
ll moduloMultiplication(ll a,ll b,ll mod){
  ll res = 0;a %= mod;
  while (b){
    if(b&1)res=(res+a)%mod;
    a=(2*a)%mod;b>>=1;}
  return res;}
ll modular_exp(ll a,ll b,ll mod){
  ll ans=1;
  while(b>0){
    if(b&1)ans=moduloMultiplication(ans,a,mod);
    b=b>>1;
    a=moduloMultiplication(a,a,mod);}
  return ans;}
bool farmatsTest(ll n,int it=10){
  if(n<4)return n==2||n==3;
  for(int i=0;i<it;i++){
    ll a=2+rand()%(n-3);
    if(modular_exp(a,n-1,n)!=1)
      return false;}
  return true;}
/**********miller rabin********************/
using u64 = uint64_t;
using u128=__uint128_t;
u64 modular_exp(u64 a,u64 b,u64 mod){
  ll ans=1;
  while(b>0){
    if(b&1)ans=(__uint128_t)ans*a%mod;
    b=b>>1;
    a=(__uint128_t)a*a%mod;}
  return ans;}
bool check_composite(u64 n,u64 a,u64 d,int s){
  u64 x=modular_exp(a,d,n);
```

```
  if(x==1||x==n-1)
    return false;
  for(int r=1;r<s;r++){
    x=(u128)x*x%n;
    if(x==n-1)
      return false;}
  return true;}
bool millerRabin(u64 n){
  if(n<2)return false;
  u64 d=n-1;
  int r=0;
  while(!(d&1)){
    d=d>>1;
    r++;}
  for(int a:{2,3,5,7,11,13,17,19,23,29,31,37})
  {if(n==a)
     return true;
    if(check_composite(n,a,d,r))
      return false;}
  return true;}
```

46 Segment Tree Lazy

```
#include<bits/stdc++.h>
using namespace std;
#define INF 2047483647
#define INFL 9223372036854775807
#define ll long long
#define pii pair<ll,ll>
#define F first
#define S second
#define mp make_pair
#define pb push_back
#define ull unsigned long long
//#define M 1000000007
#define M 998244353
#define FASTIO
    ios_base::sync_with_stdio(false);cin.tie(NULL);
    cout.tie(NULL);
#define take(x) scanf("%d",&x)
#define DE(x) printf("\ndebug %d\n",x);
#define vout(x) for(int i=0;i<x.size();i++) printf("%d
    ",x[i]);
#define pie acos(-1)
#define MOD 998244353

struct Node{
    ll set;
    ll increase;
};

const int N = 2000007;
ll ara[N],segtree[4*N];
Node lazy[4*N];
// sum Segtree with lazy updates
// will find maximum of value in a range ,
void build(int pos,int l,int r){
    if( l == r ){
        segtree[pos] = ara[l];
        return;
    }
    int mid = (l+r)/2;
    build(pos*2,l,mid);
    build(pos*2+1,mid+1,r);
    segtree[pos] = segtree[pos*2]+segtree[pos*2+1] ;
}

void lazyUpdate(int pos,int l,int r){
    if( lazy[pos].set ){
        segtree[pos]=lazy[pos].set*(r-l+1);
        if( l!=r ){
            lazy[pos*2]={lazy[pos].set,0};
            lazy[pos*2+1]={lazy[pos].set,0};
        }
        lazy[pos].set = 0;
    }
    if( lazy[pos].increase ){
        segtree[pos]+=lazy[pos].increase*(r-l+1);
        if( l!=r ){
            lazy[pos*2].increase+=lazy[pos].increase;
            lazy[pos*2+1].increase+=lazy[pos].increase;
        }
        lazy[pos].increase = 0;
    }
}

void update(int pos,int l,int r,int L,int R,int val){
    lazyUpdate(pos,l,r);
    if( l>r ) return;
    if( l> R or r<L ) return;
    if( l>=L and r<=R ){
        lazy[pos].increase+=val;
        lazyUpdate(pos,l,r);
        return;
    }
    int mid = (l+r)/2;
    update(pos*2,l,mid,L,R,val);
    update(pos*2+1,mid+1,r,L,R,val);
    segtree[pos] = segtree[pos*2+1]+ segtree[pos*2] ;
}
void updateSet(int pos,int l,int r,int L,int R,int val){
    lazyUpdate(pos,l,r);
    if( l>r ) return;
    if( l> R or r<L ) return;
    if( l>=L and r<=R ){
        lazy[pos].set=val;
        lazyUpdate(pos,l,r);
        return;
    }
    int mid = (l+r)/2;
    updateSet(pos*2,l,mid,L,R,val);
    updateSet(pos*2+1,mid+1,r,L,R,val);
    segtree[pos] = segtree[pos*2+1]+ segtree[pos*2] ;
}

ll query(int pos,int l, int r,int L,int R){
    lazyUpdate(pos,l,r);
    if( l>r ) return 0;
    if( l> R or r<L ) return 0;
    if( l>=L and r<=R ) return segtree[pos];
    int mid = (l+r)/2;
    ll val1 = query(pos*2,l,mid,L,R);
    ll val2 = query(pos*2+1,mid+1,r,L,R);
    return val1+val2;
}

int main(){
    FASTIO;
    int n,q;
    cin>>n>>q;

    for(int i=1;i<=n;i++){
        cin>>ara[i];
    }
    build(1,1,n);
    while(q--){
        int t;
        cin>>t;
        if( t == 1 ){
            int a,b,x;
            cin>>a>>b>>x;
            update(1,1,n,a,b,x);
        }else if( t == 2){
            int a,b,x;
            cin>>a>>b>>x;
            updateSet(1,1,n,a,b,x);
        }else{
            int a,b;
            cin>>a>>b;
            cout<<query(1,1,n,a,b)<<endl;
        }
    }
}
```

47 Segment Tree Persistent

```
struct Node{
    ll sum;
    Node* lft;
    Node* rht;
    Node(){
        lft=rht=NULL;
        sum=0;
    }
    Node(ll s,Node* llft,Node* rrht){
        sum=s;
        lft=llft;
        rht=rrht;
    }
};
int value[200005];
void build(Node* v,int vl,int vr){
    if(vl==vr){
        v->sum=value[vl];
        return;
    }
    int mid=(vl+vr)/2;
    v->lft=new Node();
    v->rht=new Node();
    build(v->lft,vl,mid);
    build(v->rht,mid+1,vr);
    v->sum=v->lft->sum+v->rht->sum;
}

ll query(Node* v,int vl,int vr,int l,int r){
    if(vl==l&&vr==r){
        return v->sum;
    }
    int mid=(vl+vr)/2;
    if(r<=mid){
        return query(v->lft,vl,mid,l,r);
    } else if(l>mid){
        return query(v->rht,mid+1,vr,l,r);
    } else {
        return query(v->lft,vl,mid,l,mid) +
            query(v->rht,mid+1,vr,mid+1,r);
    }
}
void update(Node* vold,Node* vnew,int vl,int vr,int
    pos,int el) {
    if(vl==vr){
        vnew->sum=el;
        return;
    }
    int mid=(vl+vr)/2;
    if(pos<=mid) {
        vnew->rht=vold->rht;
        vnew->lft=new Node();
        update(vold->lft,vnew->lft,vl,mid,pos,el);
    }else{
        vnew->lft=vold->lft;
        vnew->rht=new Node();
        update(vold->rht,vnew->rht,mid+1,vr,pos,el);
    }
```

```
    vnew->sum=vnew->lft->sum+vnew->rht->sum;
}

void solve(){
    vector<Node*>roots;
    int n,q;
    cin>>n>>q;
    for(int i=1;i<=n;i++){
        cin>>value[i];
    }
    Node* root = new Node();
    build(root,1,n);
    roots.push_back(root);
    while(q--){
        int type;
        cin>>type;
        if( type == 1 ){
            int k,a,x;
            cin>>k>>a>>x;
            k--;
            Node* newRoot = new Node(); //< important to
                create a new node to be assigned root
            update(roots[k],newRoot,1,n,a,x);
            roots[k] = newRoot;
        }else if( type == 2 ){
            int k,a,b;
            cin>>k>>a>>b;
            k--;
            cout<<query(roots[k],1,n,a,b)<<endl;;
        }else{
            int k;
            cin>>k;
            k--;
            roots.push_back(roots[k]);
        }
    }


}
int main(){
    FASTIO;
    int tc=1;
    //cin>>tc;
    for(int t=1;t<=tc;t++){
        //cout<<"Case "<<t<<":"<<endl;
        solve();
    }
}

// https://cses.fi/problemset/task/1737/
```

48 Segmented Sieve

```
vector<char> segmentedSieve(long long L, long long R) {
 // generate all primes up to sqrt(R)
 long long lim = sqrt(R);
 vector<char> mark(lim + 1, false);
 vector<long long> primes;
 for (long long i = 2; i <= lim; ++i) {
  if (!mark[i]) {
   primes.emplace_back(i);
   for (long long j = i * i; j <= lim; j += i)
    mark[j] = true;
  }
 }

 vector<char> isPrime(R - L + 1, true);
 for (long long i : primes)
  for (long long j = max(i * i, (L + i - 1) / i * i); j
      <= R; j += i)
   isPrime[j - L] = false;
 if (L == 1)
  isPrime[0] = false;
 return isPrime;
}
```

49 Set with Custom Comparator

```
struct Edge {
int a, b, w;
};

struct cmp {
bool operator() (const Edge &x, const Edge &y) const {
    return x.w < y.w; }
};
```

50 Stable Marriage

```
//order[i][j]=indexOfMan i in j-th
    women'sListOfPreference
//prefer[i]=listOfWomen inOrderOf decreasingPreference
int n;
int pre[N][N],order[N][N],nxt[N];
queue<int>q;
int future_wife[N],future_husband[N];
void engage(int man , int woman){
 int m1 = future_husband[woman];
 if(m1==0) {
  future_wife[man] = woman;future_husband[woman] = man;
 }
 else{
  future_wife[man] = woman;future_husband[woman] = man;
  future_wife[m1] =0;q.push(m1);
 }
}

void TEST_CASES(int cas){
 while(!q.empty())q.pop();
 cin>>n;
 for(int i=1;i<=n;i++) {
  for(int j=1;j<=n;j++) {
   cin>>pre[i][j]; pre[i][j]-= n;
  }
  nxt[i] = 1;future_wife[i] = 0;q.push(i);
 }
 for(int i=1;i<=n;i++) {
  for(int j=1;j<=n;j++) {
   int x; cin>>x;
   order[i][x] = j;
  }
  future_husband[i] = 0;
 }
 while(!q.empty()) {
  int man = q.front(); q.pop();
  int woman = pre[man][nxt[man]++];
  if(future_husband[woman]==0) {
   engage(man , woman);
  }
  else if(order[woman][man] <
      order[woman][future_husband[woman]]) {
   engage(man , woman);
  }
  else{ q.push(man); }
 }
 for(int i=1;i<=n;i++) {
  cout<<" ("<<i<<" "<<future_wife[i]+n<<")";
 }
}
```

51 Stress Test

```
#!/bin/sh
echo "Enter the name of first File : "
read file
echo "Enter the name of second File : "
read file2
g++ -o test_gen test_gen.cpp
g++ -o $file $file.cpp
g++ -o $file2 $file2.cpp
while true
do
   ./test_gen
   ./$file <input.txt> out1.txt
   ./$file2 <input.txt> out2.txt
   if cmp -s "out1.txt" "out2.txt"; then
 echo "Test Case OK"
   else
 echo "ERROR ENCOUNTERED"
 break
   fi
done
```

52 Suffix Array

```
vector<pll> radixsort(vector< pll > v){
    ll n=v.size(), e=1e6;

    {
        vector<ll> cnt(n);
        for(ll i=0; i<n; i++){
            cnt[v[i].x%e]++;
        }
        vector< pll > a_new(n);
        vector<ll> pos(n);
        pos[0]=0;
        for(ll i=1; i<n; i++){
            pos[i]=pos[i-1]+cnt[i-1];
        }
        for(ll i=0; i<n; i++){
            ll k=v[i].x%e;
            a_new[pos[k]]=v[i];
            pos[k]++;
        }
        v = a_new;
        //for(ll i=0; i<n; i++){
            //cout<< v[i].x<< "_"<< v[i].y<< endl;
        //}
    }

    {
        vector<ll> cnt(n);
        for(ll i=0; i<n; i++){
            cnt[v[i].x/e]++;
        }
        vector< pll > a_new(n);
        vector<ll> pos(n);
        pos[0]=0;
        for(ll i=1; i<n; i++){
            pos[i]=pos[i-1]+cnt[i-1];
        }
        for(ll i=0; i<n; i++){
            ll k=v[i].x/e;
            a_new[pos[k]]=v[i];
            pos[k]++;
        }
        v = a_new;
        //for(ll i=0; i<n; i++){
            //cout<< v[i].x<< "_"<< v[i].y<< endl;
        //}
    }
```

```
    //cout<< "okay\n";
    return v;

}

vector<ll> sufarr(string s, ll h){
    ll n=s.size();
    if(h==1){
        ll b[128]={0};
        for(ll i=0; i<n; i++){
            b[s[i]]++;
        }
        ll k=0;
        for(ll i=0; i<128; i++){
            if(b[i]>0){
                b[i]=k;
                k++;
            }
        }


        vector<ll> a(n);
        for(ll i=0; i<n; i++){
            a[i]=b[s[i]];
            //cout<< a[i]<< "_"<< "oo\n";
        }

        return a;
    }
    vector< pll > v(n);
    vector<ll> a=sufarr(s,h/2);
    ll e=1e6;
    for(ll i=0; i<n; i++){
        v[i]= mp(a[i]*e+a[(i+h/2)%n],i);
    }
    //sort(v.begin(),v.end());
    v = radixsort(v);
    for(ll i=0; i<n; i++){
        a[v[i].y]=i;
    }
    for(ll i=1; i<n; i++){
        if(v[i].x==v[i-1].x) a[v[i].y]=a[v[i-1].y];
    }
    return a;
}

vector<ll> suffixarray(string s){
    string q=s;
    q=q+'$';
    ll n=s.size();
    ll h=1;
    while(h<n) h+=h;
    return sufarr(q,h);
}

int main()
{
    string s; cin>> s;
    vector<ll> a = suffixarray(s);
    vector< pll > v;

    for(ll i=0; i<s.size()+1; i++){
        //cout<< a[i]<< " "<< s.substr(i,s.size()-i)<<
            endl;
        v.push_back(mp(a[i],i));
    }
    sort(v.begin(),v.end());
    for(ll i=0; i<s.size()+1; i++){
        ll k=v[i].y;
        cout<< k<< " ";//<< s.substr(k,s.size()-k)<<
            endl;
    }
    cout<< endl;
}
```

53 Treap

```
mt19937 rng(chrono:: steady_clock ::now().
    time_since_epoch(). count());
typedef struct item * pitem;
struct item {
    int prior, value, cnt;
    LL sum;
    bool rev;
    item(int value):prior(rng()), value(value) {
        cnt = 0;
        rev = 0;
        sum = value;
        l = r = nullptr;}
    pitem l, r;};
namespace Treap {
    int cnt (pitem it) {
        return it != nullptr? it->cnt : 0;}
    LL sum(pitem it) {
        return it != nullptr? it->sum : 0;}
    void upd_cnt (pitem it) {
        if (it!=nullptr) {
            it->cnt = cnt(it->l) + cnt(it->r) + 1;
            it->sum = sum(it->l) + sum(it->r) +
                it->value; }}
    void push (pitem it) {
        if (it != nullptr && it->rev == true) {
            it->rev = false;
            swap (it->l, it->r);
            if (it->l) it->l->rev ^= true;
            if (it->r) it->r->rev ^= true; }}
    void merge (pitem & t, pitem l, pitem r) {
        push (l);
        push (r);
        if (l==nullptr || r==nullptr)
            t = (l!=nullptr) ? l : r;
        else if (l->prior > r->prior)
            merge (l->r, l->r, r), t = l;
        else
            merge (r->l, l, r->l), t = r;
        upd_cnt (t);}
    void split (pitem t, pitem & l, pitem & r, int key,
         int add = 0) {
        if (t==nullptr) {
            l = r = nullptr;
            return;}
        push(t);
        int cur_key = add + cnt(t->l);
        if (key <= cur_key)
            split (t->l, l, t->l, key, add), r = t;
        else
            split (t->r, t->r, r, key, add + 1 +
                cnt(t->l)), l = t;
        upd_cnt (t);}
    void reverse (pitem &t, int l, int r) {
        pitem t1, t2, t3;
        split (t, t1, t2, l); //< split t into t1 and t2
            at position l, thus we have [..,l] and
            [l+1,..]
        split (t2, t2, t3, r-l+1); //< split t2 into t2
            and t3
        // so we have now 3 tree t1,t2 and t3
        assert(t2 != NULL);
        t2->rev ^= true;
        merge (t, t1, t2); // merge t1 and t2 into t
        // so now we have t and t3
        merge (t, t, t3); // merge t and t3 into t
    }
    void cut (pitem &t, int l, int r) {
        pitem L, mid, R;
        split(t, L, mid, l );
        split(mid, mid, R, r - l+1);
        merge(t, L, R);
        merge(t, t, mid);}
    LL query (pitem &t, int l, int r) {
        pitem t1, t2, t3;
        split (t, t1, t2, l);
        split (t2, t2, t3, r-l+1);
        LL ans = t2->sum;
        merge (t, t1, t2);
        merge (t, t, t3);
        return ans;}
    void insert (pitem & t, int key, int value) {
        pitem x = new item(value);
        pitem L, R;
        split(t, L, R, key);
        merge(L, L, x);
        merge(t, L, R);
        upd_cnt(t);}
    int erase (pitem & t, int key) {
        assert(cnt(t) > key);
        pitem L, MID, R;
        split(t, L, MID, key);
        split(MID, MID, R, 1);
        merge(t, L, R);
        upd_cnt(t);
        int rt = MID->value;
        delete MID;
        return rt;}
    void output (pitem t, string &v) {
        if (t==nullptr) return;
        push (t);
        output (t->l, v);
        v.push_back(t->value);
        output (t->r, v);}
    void output2 (pitem t) {
        if (t==nullptr) return;
        push (t);
//        cout << "(";
        output2 (t->l);
        cout << (t->value) << " ";
        output2 (t->r);
//        cout << ")";
    }}
int main(){
    pitem root = nullptr;
    int n,q;
    cin>>n>>q;
    string s;
    cin>>s;
    for(int i=0;i<n;i++){
        Treap::insert(root,i,s[i]);}
    while(q--){
        int l,r;
        cin>>l>>r;
        l--;r--;
        Treap::cut(root,l,r);}
    string ans;
    Treap::output(root,ans);
    cout<<ans;}
```

54 Voronoi

```
const Tf INF = 1e10;
vector<Polygon> voronoi(vector<PT> site, Tf bsq) {
  int n = site.size();
```

```
  vector<Polygon> region(n);
  PT A(-bsq, -bsq), B(bsq, -bsq),
  C(bsq, bsq), D(-bsq, bsq);
  for(int i = 0; i < n; ++i) {
    vector<DirLine> li(n - 1);
    for(int j = 0, k = 0; j < n; ++j) {
      if(i == j) continue;
      li[k++] = DirLine((site[i] + site[j]) / 2,
      rotate90(site[j] - site[i]));
    }
    li.emplace_back(A,B-A); li.emplace_back(B,C-B);
    li.emplace_back(C,D-C); li.emplace_back(D,A-D);
    region[i] = halfPlaneIntersection(li);
  }
  return region;
}
```

55 Wavelet Tree

```
const int MAXN = (int)3e5 + 9;
const int MAXV = (int)1e9 + 9; //maximum value of any
    element in array

//array values can be negative too, use appropriate
    minimum and maximum value
// tree is 1 indexed
struct wavelet_tree {
  int lo, hi;
  wavelet_tree *l, *r;
  int *b, *c, bsz, csz; // c holds the prefix sum of
      elements

  wavelet_tree() {
    lo = 1;
    hi = 0;
    bsz = 0;
    csz = 0, l = NULL;
    r = NULL;
  }

  void init(int *from, int *to, int x, int y) {
    lo = x, hi = y;
    if(from >= to) return;
    int mid = (lo + hi) >> 1;
    auto f = [mid](int x) {
      return x <= mid;
    };
    b = (int*)malloc((to - from + 2) * sizeof(int));
    bsz = 0;
    b[bsz++] = 0;
    c = (int*)malloc((to - from + 2) * sizeof(int));
    csz = 0;
    c[csz++] = 0;
    for(auto it = from; it != to; it++) {
      b[bsz] = (b[bsz - 1] + f(*it));
      c[csz] = (c[csz - 1] + (*it));
      bsz++;
      csz++;
    }
    if(hi == lo) return;
    auto pivot = stable_partition(from, to, f);
    l = new wavelet_tree();
    l->init(from, pivot, lo, mid);
    r = new wavelet_tree();
    r->init(pivot, to, mid + 1, hi);
  }
  //kth smallest element in [l, r]
  //for array [1,2,1,3,5] 2nd smallest is 1 and 3rd
      smallest is 2
  int kth(int l, int r, int k) {
    if(l > r) return 0;
    if(lo == hi) return lo;
    int inLeft = b[r] - b[l - 1], lb = b[l - 1], rb =
        b[r];
    if(k <= inLeft) return this->l->kth(lb + 1, rb, k);
    return this->r->kth(l - lb, r - rb, k - inLeft);
  }
  //count of numbers in [l, r] Less than or equal to k
  int LTE(int l, int r, int k) {
    if(l > r || k < lo) return 0;
    if(hi <= k) return r - l + 1;
    int lb = b[l - 1], rb = b[r];
    return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l
        - lb, r - rb, k);
  }
  //count of numbers in [l, r] equal to k
  int count(int l, int r, int k) {
    if(l > r || k < lo || k > hi) return 0;
    if(lo == hi) return r - l + 1;
    int lb = b[l - 1], rb = b[r];
    int mid = (lo + hi) >> 1;
    if(k <= mid) return this->l->count(lb + 1, rb, k);
    return this->r->count(l - lb, r - rb, k);
  }
  //sum of numbers in [l ,r] less than or equal to k
  int sum(int l, int r, int k) {
    if(l > r or k < lo) return 0;
    if(hi <= k) return c[r] - c[l - 1];
    int lb = b[l - 1], rb = b[r];
    return this->l->sum(lb + 1, rb, k) + this->r->sum(l
        - lb, r - rb, k);
  }
  ~wavelet_tree() {
    delete l;
    delete r;
  }
};
wavelet_tree t;
int a[MAXN];
int main() {
  int i, j, k, n, m, q, l, r;
  cin >> n;
  for(i = 1; i <= n; i++) cin >> a[i];
  t.init(a + 1, a + n + 1, -MAXV, MAXV);
  //beware! after the init() operation array a[] will
      not be same
  //kth smallest
  cout << t.kth(1, r, k) << endl;
  //less than or equal to K
  cout << t.LTE(1, r, k) << endl;
  //count occurence of K in [l, r]
  cout << t.count(1, r, k) << endl;
  //sum of elements less than or equal to K in [l, r]
  cout << t.sum(1, r, k) << endl;
  return 0;
}
```

56 Xor Basis Set

```
struct Basis{
const int LOG_A = 25;
int basis[LOG_A] = {0};
void insert(int x){
    for(int i=LOG_A-1;i>=0;i--){
        if( !basis[i] ){
            basis[i] = x;
            return;
        }
        x^=basis[i];
    }
}
int getMaxXor(){
    int mxXorr = 0;
    for(int i=LOG_A-1;i>=0;i--){
        if( !basis[i] ) continue;
        if( mxXorr & (1<<i) ) continue;
        mxXorr ^= basis[i];
    }
    return mxXorr;
}
bool isPossibleXor(int x){
    for(int i=LOG_A;i>=0;i--){
        if( ( x & (1<<i) ) == 0 ) continue;
        if( !basis[i] ) return false;
        x ^= basis[i];
    }
    return true;
}
};
```