**VON – BITS Goa ICPC Team**

**INDEX:**

1. PBDS

2. Priority Queue

3. Modular Arithmetic

4. Sparse Table

5. DSU

6. Kruskal

7. Segment Tree

8. Recursive Segment Tree

9. Fenwick Tree

10. Tree with LCA,binary lifting and Euler tour

11. Convex Hull Trick (CHT)

12. Geometry

13.Suffix Array

14. KMP

15. Z function

16. Ankush Sparse Table

17. Ankush Segment Tree

18. GCD

19. Kosaraju’s Algo (SCC)

20. Toposort

21. Dijkstra Algo

22. Bellman ford

23. Floyd Warshall

24. Mo’s algo

25. Trigonometric functions

26. Random primes

27. Random number generator

28. Linear Diophantine Equation Solver

29. Strongly Connected Components

30. 2SAT

31. Max Flow

32. Matrix Exponentiation

33. Miller Rabin

34. Pollard Rho

35. Lucas Theorem

**1. PBDS-**

#include <ext/pb\_ds/assoc\_container.hpp>

#include <ext/pb\_ds/tree\_policy.hpp>

using namespace \_\_gnu\_pbds;

template <typename T>

using ordered\_set = tree<T, null\_type, less<T>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;

//for multiset, change less to less\_equal

#define ook order\_of\_key // the number of items in a set that are strictly smaller than our item

#define fbo find\_by\_order // returns an iterator to the k-th

largest element (counting from zero)

// erase from pbds

// s.erase(s.find\_by\_order(s.order\_of\_key( x )));

**2. Priority Queue-**

template<class T> using pq = priority\_queue<T>;

template<class T> using pqg = priority\_queue<T, vector<T>, greater<T>>;

**3. Modular Arithmetic-**

int modAdd( int a , int b ) { return ( a%MOD + b%MOD ) % MOD; }

int modSubtract( int a , int b ) { return ( ( a%MOD - b%MOD )%MOD + MOD )%MOD; }

int modProduct( int a , int b ) { return ( a%MOD \* b%MOD ) % MOD; }

int powerMod( int a , int b ) {

if(b==0)

return 1;

int x = powerMod(a,b/2);

if(b%2==0)

return (x\*x) % MOD;

else

return ( ((x\*x) % MOD) \* a ) % MOD;

}

int modInverse(int a) { int x = powerMod( a , MOD-2 ); return x; }

int modDivide( int a , int b ) { return ( a%MOD \* modInverse(b)%MOD ) % MOD; }

**4. Sparse Table-**

template <typename T, class F = function<T(const T&, const T&)>>

class sparseTable {

public:

int n,K;

vector<vector<T>> st;

F comp;

sparseTable(const vector<T>& a, const F& f) : comp(f) {

n = static\_cast<int>(sz(a));

K = 32 - \_\_builtin\_clz(n);

st.resize(K);

st[0] = a;

for (int j = 1; j < K; j++) {

st[j].resize(n - (1 << j) + 1);

for (int i = 0; i <= n - (1 << j); i++) {

st[j][i] = comp(st[j - 1][i], st[j - 1][i + (1 << (j - 1))]);

}

}

}

T rq(int l, int r) const {

assert(0 <= l and l <= r and r <= n - 1);

int lg = 32 - \_\_builtin\_clz(r - l + 1) - 1;

return comp(st[lg][l], st[lg][r - (1 << lg) + 1]);

}

T rq2(int l,int r) const {

T ans = 0;

assert(0 <= l and l <= r and r <= n-1);

rloop(j,K+1,0){

if((1 << j) <= r-l+1){

ans = comp(ans,st[j][l]);

l += (1 << j);

}

}

return ans;

}

};

**5. DSU-**

struct DSU{

vector<int> e; int comps;

DSU(int n) { e = vector<int>(n,-1); comps = n;}

int get(int x) { return e[x] < 0 ? x : e[x] = get(e[x]);}

bool same(int a,int b) { return get(a) == get(b);}

int csize(int x) { return -e[get(x)];}

bool join(int x,int y){

x = get(x),y = get(y);

if(x == y) return false;

if(e[x] > e[y]) swap(x,y);

e[x] += e[y];

e[y] = x;

comps--;

return true;

}

};

**6. Kruskal-**

template<class T> T kruskal(int N,vector<pair<T,pi>> edges){

sort(all(edges));

T ans = 0; DSU D(N);

trv(e,edges) if(D.join(e.s.f,e.s.s)) ans += e.f;

return ans;

}

**7. Segment Tree :**

template<typename T,class F = function<T(const T&,const T&)>>

class segTree {

public :

int n; vector<T> t;

F comp;

T dv;

segTree(const vector<T>& a,const T val,const F& f) : comp(f){

dv = val; n = sz(a) + 1; t.assign(2\*n,dv);

loop(i,1,n) upd(i,a[i-1]);

}

void pull(int p) { t[p] = comp(t[2\*p],t[2\*p+1]); }

void upd(int p,T val){

for(t[p += n] = val,p /= 2;p;p /= 2) pull(p);}

T rq(int l,int r) {

T right = dv,left = dv;

for(l += n,r += n+1; l < r; l /= 2, r /= 2){

if(l & 1) left = comp(left,t[l++]);

if(r & 1) right = comp(t[--r],right);

}

return comp(left,right);

}

};

**8. Recursive Segment Tree :**

template<typename T,class F = function<T(const T&,const T&)>>

class segTree{

public:

int n; vector<T> t,a;

F comp;

T dv;

segTree(const vector<T>& b,const T val,const F& f) : comp(f){

dv = val,n = (int)b.size(); t.assign(4\*n,dv);

a = b;

build(1,0,n-1);

}

void build(int v,int l,int r){

if(l == r) t[v] = a[l];

else{

int m = (l + r)/2;

build(2\*v,l,m);

build(2\*v+1,m+1,r);

t[v] = comp(t[2\*v],t[2\*v+1]);

}

}

void update(int v,int l,int r,int p,T x){

if(l == r) t[v] = x;

else{

int m = (l + r)/2;

if(p <= m) update(2\*v,l,m,p,x);

else update(2\*v + 1,m+1,r,p,x);

t[v] = comp(t[2\*v],t[2\*v+1]);

}

}

T query(int v,int tl,int tr,int l,int r){

if(l > r) return 0;

if(l == tl and r == tr) return t[v];

int m = (tl + tr)/2;

return comp(query(2\*v,tl,m,l,min(r,m)),query(2\*v+1,m+1,tr,max(l,m+1),r));

}

// Functions :

void upd(int p,T x) { update(1,0,n-1,p,x); }

T rq(int l,int r) { return query(1,0,n-1,l,r); }

};

**9. Fenwick Tree :**

template<class T>

struct Fenwick{

int n;

vector<T> val;

Fenwick(int N) : n(N){ val.resize(N+1); }

void upd(int pos,T x){

for(; pos <= n; pos += pos&-pos) val[pos] += x;

}

T sum(int pos){

T res = 0;

for(; pos; pos -= pos&-pos) res += val[pos];

return res;

}

T query(int l,int r){ return sum(r) - sum(l-1); }

};

**10. Tree with LCA,binary lifting and Euler tour :**

class Tree {

public:

int n,L,t,root;

vector<vector<int>> up,adj;

vector<int> d,st,en,order;

void init(){

order.clear();

d = st = en = vector<int>(n);

L = ceil(log2(n));

up.resize(n,vector<int>(L+1));

t = 1;

dfs(root,root);

}

Tree(int N,vector<vector<int>> adj,int root = 0) : n(N),root(root),adj(adj){ init(); }

Tree(int N,vector<int> p) : n(N){

assert(N == (int)p.size());

adj.resize(n);

for(int i = 0; i < N; i++){

if(p[i] == -1) root = i;

else{

adj[i].push\_back(p[i]);

adj[p[i]].push\_back(i);

}

}

init();

}

void dfs(int s,int e){

order.push\_back(s);

st[s] = t++;

for(int i = 1; i <= L; i++)

up[s][i] = up[up[s][i-1]][i-1];

for(auto u : adj[s]){

if(u == e) continue;

d[u] = d[up[u][0] = s] + 1;

dfs(u,s);

}

en[s] = t-1;

}

bool anc(int a,int b){ return st[a] <= st[b] and en[a] >= en[b]; }

int jump(int x,int k){

for(int i = 0; i <= L; i++) if(k >> i & 1){

x = up[x][i];

}

return x;

}

int lca(int a,int b){

if(d[a] < d[b]) swap(a,b);

a = jump(a,d[a] - d[b]);

if(a == b) return a;

for(int i = L; i >= 0; i--){

int na = up[a][i],nb = up[b][i];

if(na != nb) a = na,b = nb;

}

return up[a][0];

}

int dist(int a,int b){ return d[a] + d[b] - 2\*d[lca(a,b)]; }

};

**11. DP Optimisation using CHT :**

/\*

Convex Hull Trick :

Add lines of the form : L(k,m) = kx + m.

Evaluate max at a point x across all lines.

Source : https://github.com/kth-competitive-programming/kactl/blob/main/content/data-structures/LineContainer.h

\*/

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 eval(ll x) {

assert(!empty());

auto l = \*lower\_bound(x);

return l.k \* x + l.m;

}

};

**12. Geometry :**

template <class T> int sgn(T x) { return (x > 0) - (x < 0); }

template<class T>

struct Point {

typedef Point P;

T x, y;

explicit Point(T x=0, T y=0) : x(x), y(y) {}

bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }

bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }

P operator+(P p) const { return P(x+p.x, y+p.y); }

P operator-(P p) const { return P(x-p.x, y-p.y); }

P operator\*(T d) const { return P(x\*d, y\*d); }

P operator/(T d) const { return P(x/d, y/d); }

T dot(P p) const { return x\*p.x + y\*p.y; }

T cross(P p) const { return x\*p.y - y\*p.x; }

T cross(P a, P b) const { return (a-\*this).cross(b-\*this); }

T dist2() const { return x\*x + y\*y; }

double dist() const { return sqrt((double)dist2()); }

// angle to x-axis in interval [-pi, pi]

double angle() const { return atan2(y, x); }

P unit() const { return \*this/dist(); } // makes dist()=1

P perp() const { return P(-y, x); } // rotates +90 degrees

P normal() const { return perp().unit(); }

// returns point rotated 'a' radians ccw around the origin

P rotate(double a) const {

return P(x\*cos(a)-y\*sin(a),x\*sin(a)+y\*cos(a)); }

friend ostream& operator<<(ostream& os, P p) {

return os << "(" << p.x << "," << p.y << ")"; }

};

template<class P> bool onSegment(P s, P e, P p) {

return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;

}

template<class P>

int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>

int sideOf(const P& s, const P& e, const P& p, double eps) {

auto a = (e-s).cross(p-s);

double l = (e-s).dist()\*eps;

return (a > l) - (a < -l);

}

typedef Point<ll> P;

vector<P> convexHull(vector<P> pts) {

if (sz(pts) <= 1) return pts;

sort(all(pts));

vector<P> h(sz(pts)+1);

int s = 0, t = 0;

for (int it = 2; it--; s = --t, reverse(all(pts)))

for (P p : pts) {

while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;

h[t++] = p;

}

return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};

}

bool inHull(const vector<P>& l, P p, bool strict = true) {

int a = 1, b = sz(l) - 1, r = !strict;

if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);

if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);

if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p)<= -r)

return false;

while (abs(a - b) > 1) {

int c = (a + b) / 2;

(sideOf(l[0], l[c], p) > 0 ? b : a) = c;

}

return sgn(l[a].cross(l[b], p)) < r;

}

**13. Suffix array :**

vector<int> suffArray(string s){

s += "$";

int n = s.size();

vector<int> p(n),c(n);

iota(p.begin(),p.end(),0);

sort(p.begin(),p.end(),[&](int i,int j){

return s[i] < s[j]; });

c[p[0]] = 0;

for(int i = 1; i < n; i++){

c[p[i]] = c[p[i-1]];

if(s[p[i]] != s[p[i-1]]) c[p[i]]++;

}

vector<int> np(n),nc(n);

for(int k = 0; (1 << k) < n; k++){

int sh = (1 << k);

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

np[i] = (p[i] - sh + n) % n;

}

vector<int> cnt(n);

for(int i = 0; i < n; i++) cnt[c[np[i]]]++;

for(int i = 1; i < n; i++) cnt[i] += cnt[i-1];

for(int i = n-1; i >= 0; i--){

p[--cnt[c[np[i]]]] = np[i];

}

nc[p[0]] = 0;

for(int i = 1; i < n; i++){

nc[p[i]] = nc[p[i-1]];

pair<int,int> cur = {c[p[i]],c[(p[i]+sh)%n]};

pair<int,int> nxt = {c[p[i-1]],c[(p[i-1]+sh)%n]};

if(cur != nxt) nc[p[i]]++;

}

swap(c,nc);

}

return p;

}

**14. KMP algorithm**

//will return all the indices where b is present is a in the form of vector of int

vector <int> kmp ( string a , string b )

{

int n = a.size() , m = b.size();

vector <int> pi(m); pi[0] = -1;

int j = 0;

for(int i=1 ; i<m ; i++){

if(b[i] != b[j])

j = 0;

if(b[i] == b[j]){

pi[i] = j;

j++;

}

else

pi[i] = -1;

}

vector <int> ans;

j = -1;

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

if(a[i] == b[j+1])

j++;

else{

while(j != -1){

if(a[i] == b[j+1])

break;

j = pi[j];

}

if(a[i] == b[j+1])

j++;

}

if(j == m-1){

ans.pb(i-m+1);

j = pi[j];

}

}

return ans;

}

**15. z function**

// z[i] -> length of longest substring starting at i which is also a prefix of s.

// z[0] -> 0

vector<int> z\_function(string &s)

{

int n = s.size();

vector<int> z(n , 0);

int l=0 , r=0;

for(int i=1 ; i<n ; i++){

if(i <= r) z[i] = min(z[i-l] , r-i+1);

while(i+z[i] < n and s[ z[i] ] == s[ i + z[i] ]) z[i]++;

if(i+z[i]-1 > r) l = i, r = i+z[i]-1;

}

return z;

}

**16. Ankush’s SPARSE TABLE**

template <typename T>

class SparseTable{

public:

vector <vector<T>> st;

//change function

T f(T a , T b){

return max(a , b);

}

void build(vector<T> &a , int n){

st.pb(a);

int y = 2 , counter=1;

while(y<=n){

vector <int> x;

for(int i=0 ; i<=n-y ; i++)

x.pb( f(st[counter-1][i] , st[counter-1][i+y/2] ) );

st.pb(x);

y \*= 2;

counter++;

}

}

// return query for subarray starting from i of length l

int get(int i , int l){

int logi = log2(l);

return f( st[logi][i] , st[logi][ i+l-pow(2,logi) ] );

}

// return query for subarray l to r

int getlr(int l , int r){

int len = r - l + 1;

return get(l , len);

}

};

**17. Segtree with lazy (Ankush’s)**

class SegTree

{

public:

vector<int> tree;

int n;

void init(int n){

this->n = n;

tree.assign(4\*n , 0);

}

void build(vector<int>&a , int x , int lx , int rx){

if(lx == rx){

tree[x] = a[lx];

return;

}

build(a , 2\*x , lx , (lx+rx)/2);

build(a , 2\*x+1 , (lx+rx)/2+1 , rx);

tree[x] = tree[2\*x] + tree[2\*x + 1];

}

void build(vector<int> &a){

build(a , 1 , 0 , n-1);

}

void set(int i , int v , int x , int lx , int rx){

if(i == lx and i == rx){

tree[x] = v;

return;

}

if(i<=(lx+rx)/2)

set(i , v , 2\*x , lx , (lx+rx)/2);

else

set(i , v , 2\*x+1 , (lx+rx)/2+1 , rx);

tree[x] = tree[2\*x] + tree[2\*x + 1];

}

void set(int i , int v){

set(i , v , 1 , 0 , n-1);

}

int get(int l , int r , int x , int lx , int rx){

if(r < lx or l > rx) return 0;

if(l <= lx and rx <= r) return tree[x];

int s1 = get(l , r , 2\*x , lx , (lx+rx)/2);

int s2 = get(l , r , 2\*x+1 , (lx+rx)/2+1 , rx);

return s1 + s2;

}

int get(int l , int r){

return get(l , r , 1 , 0 , n-1);

}

};

//segtree with lazy

class SegTree

{

public:

vector<int> tree , lazy;

int n;

void init(int n){

this->n = n;

tree.assign(4\*n , 0);

lazy.assign(4\*n , 0);

}

void lazyUpdate(int x , int lx , int rx){

if(lazy[x] == 0) return;

tree[x] += (rx-lx+1)\*lazy[x];

if(lx != rx){

lazy[2\*x] += lazy[x];

lazy[2\*x+1] += lazy[x];

}

lazy[x] = 0;

}

void build(vector<int>&a , int x , int lx , int rx){

if(lx == rx){

tree[x] = a[lx];

return;

}

build(a , 2\*x , lx , (lx+rx)/2);

build(a , 2\*x+1 , (lx+rx)/2+1 , rx);

tree[x] = tree[2\*x] + tree[2\*x + 1];

}

void build(vector<int> &a){

build(a , 1 , 0 , n-1);

}

int get(int l , int r , int x , int lx , int rx){

lazyUpdate(x , lx , rx);

if(r < lx or l > rx) return 0;

if(l <= lx and rx <= r) return tree[x];

int s1 = get(l , r , 2\*x , lx , (lx+rx)/2);

int s2 = get(l , r , 2\*x+1 , (lx+rx)/2+1 , rx);

return s1 + s2;

}

int get(int l , int r){

return get(l , r , 1 , 0 , n-1);

}

void update(int x , int lx , int rx , int l , int r , int v){

lazyUpdate(x , lx , rx);

if(l > rx or lx > r) return;

if(l <= lx and rx <= r){

lazy[x] += v;

lazyUpdate(x , lx , rx);

return;

}

update(2\*x , lx , (lx+rx)/2 , l , r , v);

update(2\*x+1 , (lx+rx)/2+1 , rx , l , r , v);

tree[x] = tree[2\*x] + tree[2\*x+1];

}

void update(int l , int r , int v){

update(1 , 0 , n-1 , l , r , v);

}

};

//multiple lazy queries

class SegTree

{

public:

vector<int> tree , lazy1 , lazy2;

int n;

void init(int n){

this->n = n;

tree.assign(4\*n , 0);

lazy1.assign(4\*n , 1);

lazy2.assign(4\*n , 0);

}

int modAdd(int a , int b){

long long int A = a , B = b;

long long int X = (A + B)%MOD;

int x = X;

return x;

}

int modProduct(int a , int b){

long long int A = a , B = b;

long long int X = (A \* B)%MOD;

int x = X;

return x;

}

void lazyUpdate(int x , int lx , int rx){

tree[x] = modAdd( modProduct(tree[x] , lazy1[x]) , modProduct(rx-lx+1 , lazy2[x]) );

if(lx != rx){

lazy1[2\*x] = modProduct(lazy1[2\*x] , lazy1[x]); lazy2[2\*x] = modProduct(lazy2[2\*x] , lazy1[x]);

lazy2[2\*x] = modAdd(lazy2[2\*x] , lazy2[x]);

lazy1[2\*x+1] = modProduct(lazy1[2\*x+1] , lazy1[x]); lazy2[2\*x+1] = modProduct(lazy2[2\*x+1] , lazy1[x]);

lazy2[2\*x+1] = modAdd(lazy2[2\*x+1] , lazy2[x]);

}

lazy1[x] = 1; lazy2[x] = 0;

}

void build(vector<int>&a , int x , int lx , int rx){

if(lx == rx){

tree[x] = a[lx];

return;

}

build(a , 2\*x , lx , (lx+rx)/2);

build(a , 2\*x+1 , (lx+rx)/2+1 , rx);

tree[x] = modAdd(tree[2\*x] , tree[2\*x + 1]);

}

void build(vector<int> &a){

build(a , 1 , 0 , n-1);

}

int get(int l , int r , int x , int lx , int rx){

lazyUpdate(x , lx , rx);

if(r < lx or l > rx) return 0;

if(l <= lx and rx <= r) return tree[x];

int s1 = get(l , r , 2\*x , lx , (lx+rx)/2);

int s2 = get(l , r , 2\*x+1 , (lx+rx)/2+1 , rx);

return modAdd(s1 , s2);

}

int get(int l , int r){

return get(l , r , 1 , 0 , n-1);

}

//multiply range l,r with v

void update1(int x , int lx , int rx , int l , int r , int v){

lazyUpdate(x , lx , rx);

if(l > rx or lx > r) return;

if(l <= lx and rx <= r){

lazy1[x] = modProduct(lazy1[x] , v); lazy2[x] = modProduct(lazy2[x] , v);

lazyUpdate(x , lx , rx);

return;

}

update1(2\*x , lx , (lx+rx)/2 , l , r , v);

update1(2\*x+1 , (lx+rx)/2+1 , rx , l , r , v);

tree[x] = modAdd(tree[2\*x] , tree[2\*x+1]);

}

void update1(int l , int r , int v){

if(l > r) return;

update1(1 , 0 , n-1 , l , r , v);

}

//add v to range l,r

void update2(int x , int lx , int rx , int l , int r , int v){

lazyUpdate(x , lx , rx);

if(l > rx or lx > r) return;

if(l <= lx and rx <= r){

lazy2[x] = modAdd(lazy2[x] , v);

lazyUpdate(x , lx , rx);

return;

}

update2(2\*x , lx , (lx+rx)/2 , l , r , v);

update2(2\*x+1 , (lx+rx)/2+1 , rx , l , r , v);

tree[x] = modAdd(tree[2\*x] , tree[2\*x+1]);

}

void update2(int l , int r , int v){

if(l > r) return;

update2(1 , 0 , n-1 , l , r , v);

}

// set range l,r with value v

void update3(int x , int lx , int rx , int l , int r , int v){

lazyUpdate(x , lx , rx);

if(l > rx or lx > r) return;

if(l <= lx and rx <= r){

lazy1[x] = modProduct(lazy1[x] , 0); lazy2[x] = modProduct(lazy2[x] , 0);

lazy2[x] = modAdd(lazy2[x] , v);

lazyUpdate(x , lx , rx);

return;

}

update3(2\*x , lx , (lx+rx)/2 , l , r , v);

update3(2\*x+1 , (lx+rx)/2+1 , rx , l , r , v);

tree[x] = modAdd(tree[2\*x] , tree[2\*x+1]);

}

void update3(int l , int r , int v){

if(l > r) return;

update3(1 , 0 , n-1 , l , r , v);

}

};

**18. Gcd**

int gcd (int a, int b) {

if (b == 0)

return a;

else

return gcd (b, a % b);

}

**19. Kosaraju (Strongly connected components)**

1. find toposort.

2. reverse the graph.

3. apply dfs starting from the top of stack

void Kosaraju(){

int n,m; inp(n,m);

vi adj[n],radj[n];

loop(i,0,m){

int u,v; inp(u,v); u--,v--;

adj[u].pb(v);

radj[v].pb(u);

}

vb vis(n);

vi order;

function<void(int)> dfs = [&](int s){

vis[s] = true;

for(auto u : adj[s]){

if(!vis[u]) dfs(u);

}

order.pb(s);

};

loop(i,0,n) if(!vis[i]) dfs(i);

reverse(all(order));

vi comp(n); // Component Index of each node

int comps = 0; // Total components

vis = vb(n);

function<void(int)> dfs1 = [&](int s){

comp[s] = comps;

vis[s] = true;

for(auto u : radj[s]){

if(!vis[u]) dfs1(u);

}

};

trv(u,order) if(!vis[u]){

dfs1(u);

comps++;

}

}

**20. Toposort**

vi topoSort(const vector<vi>& gr) {

vi indeg(sz(gr)), ret;

for (auto& li : gr) for (int x : li) indeg[x]++;

queue<int> q; // use priority queue for lexical largest ans

rep(i,0,sz(gr)) if (indeg[i] == 0) q.push(i);

while (!q.empty()) {

int i = q.front(); // top () for priority queue

ret.push\_back(i);

q.pop();

for (int x : gr[i])

if (--indeg[x] == 0) q.push(x);

}

return ret;

}

**21. Fast Dijkstra Algorithm**

// Implementation of Dijkstra's algorithm using adjacency lists

// and priority queue for efficiency.

//

// Running time: 0(|E| log (VI)

#include "template.h"

const int INF = 2000000000;

int main() {

int N, s, ti

scanf("%d %d%d", &N, &s, &t); vector<vector<pii> > edges (N); for (int i = 0; i < N; i++) { int M;

scanf("%d", &M);

for (int j = 0; j < M; j++) { int vertex, dist;

scanf("%d %d", &vertex, &dist);

edges [i].push\_back (make\_pair (dist, vertex)) ; // note order of arguments here

// use priority queue in which top element has the "smallest" priority

priority\_queue<pii, vector<pii>, greater<pii> > Q;

vector<int> dist (N, INF), dad (N, −1); Q.push (make\_pair (0, s));

dist [s] = 0;

while (!Q.empty()) {

pii p = Q.top();

Q.pop();

int here = p.second;

if (here == t) break;

if (dist [here] != p.first) continue;

for (vector<pii>::iterator it = edges [here]. begin(); it != edges [here].end(); it++) { if (dist [here] + it->first ‹ dist[it->

second]) {

dist [it->second];

dad [it->second]

=

=

dist [here] + it->first

here;

Q.push(make\_pair (dist [it->second], it-> second));

printf("%d\n", dist[t]);

if (dist [t] < INF)

for (int i = t; i != -1; i = dad[i]) printf("%d%c", i, (i

(i == s ? '\n' : ''));

return 0;

}

/\*

Sample input:

5 0 4

2 1 2 3 1

2 2 4 4 5

3 1 4 3 3 4 1

2 0 1 2 3

2 1 5 2 1

Expected:

5

4 2 3 0

\*/

**22. Bellman Ford Algorithm**

const ll inf = LLONG\_MAX;

struct Ed { int a, b, w, s() { return a < b ? a : -a; }};

struct Node { ll dist = inf; int prev = -1; };

void bellmanFord(vector<Node>& nodes, vector<Ed>& eds, int s) {

nodes[s].dist = 0;

sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });

int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled vert ices

rep(i,0,lim) for (Ed ed : eds) {

Node cur = nodes[ed.a], &dest = nodes[ed.b];

if (abs(cur.dist) == inf) continue;

ll d = cur.dist + ed.w;

if (d < dest.dist) {

dest.prev = ed.a;

dest.dist = (i < lim-1 ? d : -inf);

}

}

rep(i,0,lim) for (Ed e : eds) {

if (nodes[e.a].dist == -inf)

nodes[e.b].dist = -inf;

}

**23. Floyd Warshall Algo**

const ll inf = 1LL << 62;

void floydWarshall(vector<vector<ll>>& m) {

int n = sz(m);

rep(i,0,n) m[i][i] = min(m[i][i], 0LL);

rep(k,0,n) rep(i,0,n) rep(j,0,n)

if (m[i][k] != inf && m[k][j] != inf) {

auto newDist = max(m[i][k] + m[k][j], -inf);

m[i][j] = min(m[i][j], newDist);

}

rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)

if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;

}

**24. Mo’s Algo**

**﻿**

// Algorithm for sorting the quries in an order which

// minimizes the time required from O(n^2) to 0(( n+Q) sqrt (n))

// + QlogQ This is done by sorting the queries in // order of range on which they are performed // We store the queries and sort them using the compare

// function cmp. Also we need to make an add

function to

// calculate the value of range (1,r+1) from

value of range

// (1,r) and (1+1,r) from the value of (1,r), and a remove

// function to calculate the value of (1-1, r)

from the value

// of (1,r) and (1,r-1) from the value of (1,r) in constant time

// S is the max integer number which is less than sqrt (N);

int S = (int) (sqrt (N)); // Here see if you want

11

bool cmp (Query A, Query B)

}

if (A.1 S != B.1/S) return A.1 / S < B.1 /

S;

return A. r> B.r;

}

**24. Ansh Mo’s algo**

#include<bits/stdc++.h>

#include <ext/pb\_ds/assoc\_container.hpp>

#include <ext/pb\_ds/tree\_policy.hpp>

using namespace std;

using namespace \_\_gnu\_pbds;

#define ll long long

#define int ll

#define pb push\_back

#define mp make\_pair

#define MOD 1000000007

#define inf 1000000000000000000

#define MOD2 998244353

#define sumod(a,b) a=((a)%MOD+(b)%MOD)%MOD

#define PI 3.141592653589793238462643383279502884197169399375105820974944

#define print2d(n,m,a) for(int i=0;i<n;i++){for(int j=0;j<m;j++){cout<<a[i][j]<<" ";}cout<<endl;}

int ceilo(int n,int divo)

{

if(n%divo==0)

return n/divo;

else

return n/divo+1;

}

struct wow{

ll left,right,k,i,ans,val;

};

typedef pair<int,int>ii;

typedef tuple<int,int,int> iii;

typedef vector<int> vi;

typedef vector<ii> vii;

int n,q;

bool compare(wow x1,wow x2)

{

if(x1.left/550==x2.left/550)return x1.right<x2.right;

return x1.left/550<x2.left/550;

}

bool cmp(wow x1,wow x2)

{

return x1.i<x2.i;

}

signed main()

{

//mt19937 rng(chrono::steady\_clock::now().time\_since\_epoch().count());

std::mt19937 generator (123);

ios::sync\_with\_stdio(false);

cin.tie(0);

#ifndef ONLINE\_JUDGE

freopen("input.txt","r",stdin);

freopen("output.txt","w",stdout);

#endif

int test=1;

//cin>>test;

for(int testcase=1;testcase<=test;testcase++)

{

cin>>n>>q;

vector<int> a(n);

int maxa=0;

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

cin>>a[i];

maxa=max(maxa,a[i]);

}

vector<wow> query(q);

for(int i=0;i<q;i++){

int lol1,lol2,lol3;

cin>>lol1>>lol2>>lol3;

lol1--;lol2--;

query[i].left=lol1;

query[i].right=lol2;

query[i].k=lol3;

query[i].i=i;

query[i].ans=0;

query[i].val=(lol2-lol1+1)/lol3;

}

sort(query.begin(),query.end(),compare);

int freq[300001];

for(int i=0;i<=maxa;i++){

freq[i]=0;

}

int curl=0,curr=-1;

for(int i=0;i<q;i++){

while(query[i].left<curl){

curl--;

freq[a[curl]]++;

}

while(query[i].left>curl){

freq[a[curl]]--;

curl++;

}

while(query[i].right>curr){

curr++;

freq[a[curr]]++;

}

while(query[i].right<curr){

freq[a[curr]]--;

curr--;

}

int lmfao=LLONG\_MAX;

for(int j=1;j<=80;j++){

int index=generator()%(query[i].right-query[i].left+1)+query[i].left;

if(freq[a[index]]>query[i].val){

lmfao=min(lmfao,a[index]);

}

}

//check

if(lmfao!=LLONG\_MAX)query[i].ans=lmfao;

else query[i].ans=-1;

}

sort(query.begin(),query.end(),cmp);

for(int i=0;i<q;i++){

cout<<query[i].ans<<endl;

}

cerr << "Time elapsed: " << 1.0 \* clock() / CLOCKS\_PER\_SEC << " s.\n";

}

}

**25. Trignometric functions**

// C++ program to illustrate some of the

// above mentioned trigonometric functions

#include <iostream>

#include <math.h>

using namespace std;

int main()

{

double x = 2.3;

cout << "Sine value of x = 2.3: "

<< sin(x) << endl;

cout << "Cosine value of x = 2.3: "

<< cos(x) << endl;

cout << "Tangent value of x = 2.3: "

<< tan(x) << endl;

x = 1.0;

cout << "Arc Cosine value of x = 1.0: "

<< acos(x) << endl;

cout << "Arc Sine value of x = 1.0: "

<< asin(x) << endl;

cout << "Arc Tangent value of x = 1.0: "

<< atan(x) << endl;

x = 57.3; // in degrees

cout << "Hyperbolic Cosine of x=57.3: "

<< cosh(x) << endl;

cout << "Hyperbolic tangent of x=57.3: "

<< tanh(x) << endl;

return 0;

}

**26. random primes**

10000019

10000169

10000223

10000379

10000931

99999988898898889

99999999999899999

99999999999999997

Solutions of ax % m = b

**27. Random generator**

mt19937 rng(chrono::steady\_clock::now().time\_since\_epoch().count());

#define uid(a, b) uniform\_int\_distribution<int>(a, b)(rng)

**28. Linear Diophantine Equation Solver**

// General Solution : {x0 + k(b/g), y0 - k(a/g)}

template<class T>

T extended\_euclidean(T a,T b,T& x,T& y) {

if(!b) {

x = 1,y = 0;

return a;

}

T x1,y1;

T d = extended\_euclidean(b,a % b,x1,y1);

x = y1;

y = x1 - y1 \* (a/b);

return d;

}

template<class T>

bool base\_solution(T a,T b,T c,T& x,T& y,T& g) {

g = extended\_euclidean(abs(a),abs(b),x,y);

if(c % g) return false;

x \*= c/g;

y \*= c/g;

if(a < 0) x = -x;

if(b < 0) y = -y;

return true;

}

**29. Strongly Connected Components**

vector<vector<int>> adj,radj;

class SCC {

public:

int n,comps;

vector<int> order,vis,comp;

SCC(int n) : n(n) {

vis.resize(n);

comp = vis;

comps = 0;

makeSCC();

}

void dfs(int s) {

vis[s] = 1;

for(auto u : adj[s]) {

if(!vis[u]) dfs(u);

}

order.push\_back(s);

}

void dfs1(int s) {

comp[s] = comps;

vis[s] = 1;

for(auto u : radj[s]) {

if(!vis[u]) dfs1(u);

}

}

void makeSCC() {

for(int i = 0; i < n; i++) if(!vis[i]) dfs(i);

reverse(begin(order),end(order));

vis.assign(n,0);

for(auto u : order) if(!vis[u]) {

dfs1(u);

comps++;

}

}

};

**30. 2SAT**

class \_2SAT {

public:

int n;

vector<bool> assignment;

\_2SAT(int N) : n(2\*N) {

assignment.resize(N);

adj.assign(n,{});

radj.assign(n,{});

}

bool solve\_2SAT() {

SCC s(n);

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

if (s.comp[i] == s.comp[i + 1]) return false;

assignment[i/2] = s.comp[i] > s.comp[i + 1];

}

return true;

}

void add\_disjunction(int a, bool na, int b, bool nb){

a = 2\*a ^ na;

b = 2\*b ^ nb;

int neg\_a = a ^ 1;

int neg\_b = b ^ 1;

adj[neg\_a].push\_back(b);

radj[b].push\_back(neg\_a);

adj[neg\_b].push\_back(a);

radj[a].push\_back(neg\_b);

}

};

**31. Max Flow**

template<class F> struct Dinic {

struct Edge { int to,rev; F cap; };

int n;

vector<vector<Edge>> adj;

vector<int> lvl,ptr,prv;

Dinic(int N) : n(N),adj(n),lvl(n),ptr(n){}

pair<int,int> addEdge(int u,int v,F cap,F rcap = 0) {

assert(min(cap,rcap) >= 0);

adj[u].push\_back({v,sz(adj[v]),cap});

adj[v].push\_back({u,sz(adj[u])-1,rcap});

return {u,sz(adj[u])-1};

}

F edgeFlow(int u,int id) {

const Edge& e = adj[u][id];

return adj[e.to][e.rev].cap;

}

bool bfs(int s,int t) {

lvl = ptr = vector<int>(n);

lvl[s] = 1;

queue<int> q({s});

while(!q.empty()) {

auto u = q.front(); q.pop();

for(auto e : adj[u]) if(e.cap and !lvl[e.to]) {

q.push(e.to);

lvl[e.to] = lvl[u] + 1;

if(e.to == t) return true;

}

}

return false;

}

F dfs(int v,int t,F f) {

if(v == t or !f) return f;

for(int& i = ptr[v]; i < sz(adj[v]); i++) {

Edge& e = adj[v][i];

if(lvl[e.to] != lvl[v] + 1 or !e.cap) continue;

if(F b = dfs(e.to,t,min(f,e.cap))) {

e.cap -= b,adj[e.to][e.rev].cap += b;

return b;

}

}

return 0;

}

F maxFlow(int s,int t) {

F mf = 0;

while(bfs(s,t)) while(F b = dfs(s,t,numeric\_limits<F>::max())) mf += b;

return mf;

}

};

**32. Matrix Exponentiation**

#include <bits/stdc++.h>

using namespace std;

using ll = long long;

const ll MOD = 1e9 + 7;

using Matrix = array<array<ll, 2>, 2>;

Matrix mul(Matrix a, Matrix b) {

Matrix res = {{{0, 0}, {0, 0}}};

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

for (int j = 0; j < 2; j++) {

for (int k = 0; k < 2; k++) {

res[i][j] += a[i][k] \* b[k][j];

res[i][j] %= MOD;

}

}

}

return res;

}

int main() {

ll n;

cin >> n;

Matrix base = {{{1, 0}, {0, 1}}};

Matrix m = {{{1, 1}, {1, 0}}};

for (; n > 0; n /= 2, m = mul(m, m)) {

if (n & 1) base = mul(base, m);

}

cout << base[0][1];

}

**33. Miller Rabin**

Description: Deterministic Miller-Rabin primality test. Guaranteed to

work for numbers up to 7 · 1018; for larger numbers, use Python and extend A randomly.

Time: 7 times the complexity of a

b mod c.

bool isPrime(ull n) {

if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;

ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},

s = \_\_builtin\_ctzll(n-1), d = n >> s;

for (ull a : A) { // ^ count t ra i l in g zeroes

ull p = modpow(a%n, d, n), i = s;

while (p != 1 && p != n - 1 && a % n && i--)

p = modmul(p, p, n);

if (p != n-1 && i != s) return 0;

}

return 1;

**34. Pollard Rho**

Description: Pollard-rho randomized factorization algorithm. Returns

prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).

Time: O(n^0.25) less for numbers with small factors.

ull pollard(ull n) {

auto f = [n](ull x) { return modmul(x, x, n) + 1; };

ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;

while (t++ % 40 || \_\_gcd(prd, n) == 1) {

if (x == y) x = ++i, y = f(x);

if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;

x = f(x), y = f(f(y));

}

return \_\_gcd(prd, n);

}

vector<ull> factor(ull n) {

if (n == 1) return {};

if (isPrime(n)) return {n};

ull x = pollard(n);

auto l = factor(x), r = factor(n / x);

l.insert(l.end(), all(r));

return l;

**35. Lucas Theorem**

Let n, m be non-negative integers and p a prime. Write

n = nkpk + ... + n1p + n0 and m = mkpk + ... + m1p + m0. Then nCm ≡ πki=0 niCmi (mod p)

**36. Misc**

ios\_base::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);

#ifndef ONLINE\_JUDGE

freopen("input.txt", "r", stdin);

freopen("output.txt", "w", stdout);

#endif

**37. SegTree Onkar**

#include <bits/stdc++.h>

using namespace std;

#define int long long

const int INF = 1E18;

struct Node {

int pref, suff, sum, ans, mx, cnt;

};

Node ID = {0, 0, 0, 0, -INF, 0};

Node make(int x) {

Node r;

r.pref = max(x, 0ll);

r.suff = max(x, 0ll);

r.ans = max(x, 0ll);

r.sum = x;

r.mx = x;

r.cnt = 1;

return r;

}

struct SegmentTree {

vector<Node> st;

int n;

SegmentTree(int \_n) {

n = \_n;

st.resize(4\*n, ID);

}

Node combine(Node a, Node b) {

Node r;

r.pref = max(a.pref, a.sum + b.pref);

r.suff = max(a.suff + b.sum, b.suff);

r.sum = a.sum + b.sum;

r.ans = max(a.ans, b.ans);

r.ans = max(r.ans, a.suff + b.pref);

r.mx = max(a.mx, b.mx);

r.cnt = a.cnt + b.cnt;

return r;

}

void update(int node, int l, int r, int pos, int val, bool add) {

if(l == r) {

st[node] = make(val);

if(!add) st[node].cnt = 0;

} else {

int m = (l + r)/2;

if(pos <= m) update(node\*2, l, m, pos, val, add);

else update(node\*2+1, m+1, r, pos, val, add);

st[node] = combine(st[node\*2], st[node\*2+1]);

}

}

void update(int pos, int val, bool add) {

update(1, 0, n - 1, pos, val, add);

}

Node query(int node, int l, int r, int tl, int tr) {

if(l == tl && r == tr) {

return st[node];

}

int m = (l + r)/2;

Node a = ID;

Node b = ID;

if(tl <= m) a = query(node\*2, l, m, tl, min(tr, m));

if(tr > m) b = query(node\*2+1, m+1, r, max(tl,m+1), tr);

return combine(a, b);

}

int query(int tl, int tr) {

Node ans = query(1, 0, n - 1, tl, tr);

if(ans.ans == 0 && ans.cnt == tr - tl + 1) return min(ans.ans, ans.mx);

return ans.ans;

}

};

struct Query{

int l, r, x, ind;

};

int32\_t main() {

cin.tie(0)->sync\_with\_stdio(0);

int n, m;

cin >> n >> m;

SegmentTree st(m);

vector<Query> v(m);

for(int i = 0; i < m; i++) {

cin >> v[i].l >> v[i].r >> v[i].x;

--v[i].l; --v[i].r;

v[i].ind = i;

}

vector<vector<Query>> add(n), rem(n);

for(auto q : v) {

add[q.l].push\_back(q);

rem[q.r].push\_back(q);

}

int q;

cin >> q;

vector<Query> query(q);

for(int i = 0; i < q; i++) {

cin >> query[i].x >> query[i].l >> query[i].r;

--query[i].l; --query[i].r;

--query[i].x;

query[i].ind = i;

}

vector<vector<Query>> actual\_query(n);

for(auto i : query) {

actual\_query[i.x].push\_back(i);

}

vector<int> ans(q);

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

for(auto j : add[i]) {

st.update(j.ind, j.x, true);

}

for(auto j : actual\_query[i]) {

ans[j.ind] = st.query(j.l, j.r);

}

for(auto j : rem[i]) {

st.update(j.ind, 0, false);

}

}

for(auto i : ans) cout << i << '\n';

}

**38. Sieve**

vector<ll> sieve(int n) {

int\*arr = new int[n + 1]();

vector<ll> vect;

for (int i = 2; i <= n; i++)

if (arr[i] == 0) {

vect.push\_back(i);

for (int j = 2 \* i; j <= n; j += i)

arr[j] = 1;

}

return vect;

}

**39. NcR**

ll combination(ll n, ll r, ll m, ll \*fact, ll \*ifact) {

ll val1 = fact[n];

ll val2 = ifact[n - r];

ll val3 = ifact[r];

return (((val1 \* val2) % m) \* val3) % m;

}

40. **Max SubArrayXOR - Trie**

struct TrieNode{

int value;

TrieNode \*child[2];

TrieNode(){

this->value = 0;

this->child[0] = this->child[1] = NULL;

}

};

class Solution{

public:

void insert(TrieNode \*root,int pre\_xor){

TrieNode \*temp = root;

for(int i=31;i>=0;i--){

bool curr = pre\_xor & (1<<i);

if(temp->child[curr]==NULL){

temp->child[curr] = new TrieNode();

}

temp = temp->child[curr];

}

temp->value=pre\_xor;

}

int query(TrieNode \*root,int pre\_xor){

TrieNode \*temp = root;

for(int i=31;i>=0;i--){

bool curr = pre\_xor & (1<<i);

if(temp->child[1-curr] != NULL){

temp = temp->child[1-curr];

}else if(temp->child[curr] != NULL){

temp = temp->child[curr];

}

}

return pre\_xor^(temp->value);

}

int maxSubarrayXOR(int N, int arr[]){

TrieNode \*root = new TrieNode();

insert(root,0);

int result = INT\_MIN,pre\_xor=0;

for(int i=0;i<N;i++){

pre\_xor= pre\_xor^arr[i];

insert(root,pre\_xor);

result = max(result,query(root,pre\_xor));

}

return result;

}

};