**WE_HAVE_AK**

ICPC NOTEBOOK

---------- **GENERAL** ----------

```
#pragma GCC optimize("Ofast")
#pragma GCC
target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,
mmx,avx,avx2,fma")
#pragma GCC optimize("unroll-loops")

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

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

typedef long long ll;
typedef long double ld;
typedef vector<int> vi;
typedef vector<ld> vld;
typedef vector<pair<ll , ll>> vpll;
typedef vector<pair<ld , ld>> vplld;
typedef pair<int,int> pii;
typedef vector<pair<int,int>> vpii;
typedef vector<ll> vl;
typedef pair<ll,ll> pll;
typedef priority_queue<ll> pq;

#define ff first
#define ss second
#define pb push_back
#define mp make_pair
#define pi 3.1415926536
#define all(x)          (x).begin(), (x).end()
#define precise(x,y)
fixed<<setprecision(y)<<x
#define NO    {cout<<"NO"<<endl; return;}
#define YES {cout<<"YES"<<endl; return;}
#define NEG1 {cout<<"-1"<<endl; return;}
#define setbits(x)  __builtin_popcountll(x)
#define pqb         priority_queue<int>
#define pqs
priority_queue<int,vector<int>,greater<int>
>
#define piipqs
priority_queue<pii,vector<pii>,greater<pii>
>
#define piipqb      priority_queue<pii>
#define pqbl        priority_queue<ll>
#define pqsl
priority_queue<ll,vector<ll>,greater<ll>>
#define pllpqs
priority_queue<pll,vector<pll>,greater<pll>
>
#define pllpqb      priority_queue<pll>
#define FIO ios_base::sync_with_stdio(0);
cin.tie(0)
#define oset tree<int, null_type,less<int>,
rb_tree_tag,tree_order_statistics_node_upda
te>

const ll inf = 2e18;
const double epsilon = 1e-7;

int main() {
  FIO;
  #ifndef ONLINE_JUDGE
    freopen("input.txt", "r" , stdin);
    freopen("output.txt", "w" , stdout);
  #endif
  solve();
  return 0;
}
```

---------- **POWER** ----------

```
ll mypow(ll x, ll y) {
 x%=mod;
 ll res = 1;
 while (y) {
  if (y & 1) res = (res * x) % mod;
  x = (x * x) % mod;
  y >>= 1;
 }
 return res;
}
ll imod(ll a, ll m)
{ return mypow(a, m - 2);}
```

**——-----BIG PRIMES [9-12] DIGITS—----**

```
ll p9 = 999999937;
ll p10 = 9999999967;
ll p11 = 99999999977;
ll p12 = 999999999989;
```

**—----COMBINATORICS - C(N,R)------**

```
vector<ll>fac, inv, ifac;
void pre(ll lim) {
 fac.push_back(1); fac.push_back(1);
 ifac.push_back(1); ifac.push_back(1);
 inv.push_back(0); inv.push_back(1);
 for (ll i = 2; i <= lim; i++) {
  fac.push_back(fac.back()*i % mod);
  inv.push_back(mod - (inv[mod % i] * (mod
/ i) % mod));
  ifac.push_back(ifac.back()*inv.back() %
mod);
 }
}
ll ncr(ll n, ll r) {
 if (n < r || n < 0)
  return 0;
 if (r == 0)
  return 1;
 return (((fac[n] * ifac[r]) % mod) * ifac[n -
r]) % mod;
}
```

**—---------- DSU —----------**

```
class dsu {
public:
  vector<int> parent;
  vector<int> size;
  vector<int> rank;

  explicit dsu(int a) {
    parent.resize(a);
    size.resize(a);
    rank.resize(a);
    for (int i = 0; i < a; i++) {
      parent[i] = i;
      size[i] = 1;
      rank[i]=0;
    }
  }

  int par(int i) {
    if (i == parent[i])
      return i;
    return parent[i] = par(parent[i]);
  }

  bool unite(int a, int b) {
    a = par(a);
    b = par(b);
    if (a != b) {
      if (rank[a] > rank[b])
        swap(a, b);
      parent[a] = b;
      size[b] += size[a];
      size[a] = 0;
      if (rank[a] == rank[b])
        rank[b]++;
```

```
    return true;
   }
   return false;
  }
};
```

**—----------FENWICK TREE —----------**

```
class fente { //1-base
public:
 int n;
 vector<ll> tree;
 void init(int _n) {
  n = _n;
  tree.resize(n + 5, 0);
 }
 void update(int idx, ll inc) {
  for (int i = idx; i <= n; i += i & -i)
   tree[i] += inc;
 }
 ll query(int idx) {
  if(idx<0) return 0;
  ll sum = 0;
  assert(idx<=n);
  for (int i = idx; i >= 1; i -= i & -i)
   sum += tree[i];
  return sum;
 }
};
```

**---------- SEGTREE NORMAL ---------**

```
struct node {
 ll val;
};

struct segtree {
 int size;
 vector<struct node> values;
 // default here
 struct node empty_node = {0};

 void init(int n) {
  size = 1;
  while(size < n) size*=2;
  values.assign(2*size , empty_node);
 }

 struct node merge(struct node p1 , struct
node p2) {
  struct node merge_node;
  // operation here
  merge_node.val = p1.val + p2.val;
  return merge_node;
 }

 void set(int i , ll v , int x , int  lx , int rx) {
  if(rx  - lx == 1) {
   values[x].val = v;
   return;
  }
  int m = (lx + rx)/2;
  if( i < m) {
   set(i , v , 2*x + 1 , lx , m);
  }
  else {
   set(i , v , 2*x + 2 , m , rx);
  }
  values[x] =
merge(values[2*x+1] , values[2*x+2]);
 }

 void set(int i , ll v) {
  set(i , v, 0 , 0 , size);
 }

 struct node range_calc (int l , int r , int x ,
int lx , int rx) {

  if(lx >= r || l >= rx) return empty_node;
  if(l <= lx && r >= rx) return values[x];
  int m = (lx + rx)/2;
  struct node lf =
        range_calc(l , r , 2*x+1 , lx , m);
```

```
    struct node rt =
            range_calc(l , r , 2*x+2 , m , rx);
    return merge(lf , rt);
  }
  ll range_calc(int l , int r) {
    return range_calc(l , r , 0 , 0 , size).val;
  }

int find_first(int l , int r , ll v , int x , int lx ,
int rx) {
    if(lx >= r || l >= rx) return -1;
    int m = (lx + rx)/2;
    if(rx - lx == 1) {
      if(values[x].val >= v) return lx;
      return -1;
    }
    if(l <= lx && r >= rx) {
     if(values[2*x+1].val >= v) {
     return find_first(l , r , v ,  2*x+1 , lx , m);
      }
   else {
     return find_first(l , r , v ,  2*x+2 , m , rx);
     }
  }
   int ans1  =
           find_first(l , r , v ,  2*x+1 , lx , m);
    if(ans1 != -1) return ans1;
    return find_first(l , r , v , 2*x+2 , m , rx);
  }

  int find_first(int l , int r , ll v) {
    return find_first(l , r ,  v , 0 , 0 , size);
    // first guy >= v in range (l,r)
  }
};
```

**—--------- SEGTREE LAZY —----------**

```
struct node {
  ll add;      // lazy addition in range example
  ll maxi;   // find max
};

struct segtree {
  int size;
  vector<struct node> oper;
  struct node defval = {0ll ,0ll};

  void init(int n) {
    size = 1;
    while(size < n) size*=2;
    oper.assign(2*size , defval);
  }

struct node merge(struct node p1 , struct
node p2) {
     struct node merge_node = defval;
     // operation here
     merge_node.maxi =
                    max(p1.maxi , p2.maxi);
     return merge_node;
}

struct node apply_oper(struct node x , struct
node y)
{
    struct node child = x;
     // y is parent , res is child
    child.add += y.add;
    return child;
}

void propagation(int x , int lx , int rx) {
    if(rx - lx == 1) {
      // relax all lazy operations last leaf
       oper[x].maxi += oper[x].add;
       oper[x].add = 0;
```

```
      return;
    }
    else {
     oper[2*x+1] =
            apply_oper(oper[2*x+1] , oper[x]);
     oper[2*x+2] =
            apply_oper(oper[2*x+2] , oper[x]);
     // relax all lazy operations inter node
     oper[x].maxi += oper[x].add;
     oper[x].add = 0;

   }
  }

  void set(int i , ll v, int x , int lx , int rx)
  {
   propagation(x , lx , rx);
   if(rx - lx == 1) {
    oper[x].maxi = v;
    return;
   }
   int m = (lx + rx)/2;

   if(i < m)
    set(i, v ,2*x+1 , lx , m);
   else
    set(i, v ,2*x+2 , m , rx);
  }

  void set(int i , ll v) {
    set( i , v , 0 , 0 , size);
  }

void range_update
(int l , int r ,ll v, int x , int lx , int rx)
{
   propagation(x , lx , rx);
   if(lx >= r || l >= rx) return ;
   if(l <= lx && r >= rx) {  // included
    oper[x].add += v;
    propagation(x , lx , rx);
    return;
   }

   int m = (lx + rx)/2;
   range_update(l , r , v , 2*x+1 , lx , m);
   range_update(l , r , v , 2*x+2 , m , rx);
   propagation(2*x+1 , lx , m);
   propagation(2*x+2 , m , rx);
   oper[x] =
       merge(oper[2*x+1] , oper[2*x+2]);
}
void range_update(int l , int r , ll v) {
range_update(l , r  , v, 0 , 0 , size);
}

struct node range_calc
(int l , int r , int x , int lx , int rx) {
   propagation(x , lx , rx);
   if(lx >= r || l >= rx) return {0 , (ll)-inf};
   if(l <= lx && r >= rx) {
    return oper[x];   // included in range
   }
   int m = (lx + rx)/2;
   propagation(2*x+1 , lx , m);
   propagation(2*x+2 , m , rx);

   oper[x] =
        merge(oper[2*x+1] , oper[2*x+2]);
   return
     merge(range_calc(l , r , 2*x+1 , lx , m),
range_calc(l , r , 2*x+2 , m , rx));
}

 ll range_calc(int l , int r) {
   return range_calc(l , r , 0 , 0 , size).maxi;
 }
};
```

**———---------SPARSE TABLE———----------**

```
const int N=200000;
const int M = 21;
int tab[N+1][M+1],L[N+1],a[N];

struct st{
  int n;
  st(int _n){
    n=_n;
    for(int i=2;i<=n;i++) L[i]=L[i/2]+1;
  }
  int f(int x,int y){
    return max(x,y);
  }
  void build(){
    for(int i=0;i<n;i++) tab[i][0]=a[i];
    for(int j=1;j<=M;j++){
      for(int i=0;i<n;i++){
        if(i+(1<<j)-1<n)

tab[i][j]=f(tab[i][j-1],tab[i+(1<<(j-1))][j-1]);
      }
    }
  }
  int qry(int l,int r){
    int len=r-l+1;
    int idx=l;
    int tot=0; // initialize neutral
    for(int j=M;j>=0;j--){
      if(len&(1ll<<j)){
        tot=f(tot,tab[idx][j]);
        idx+=(1<<j);
      }
    }
    return tot;
  }
  int qry_i(int l,int r){
    int lg=L[r-l+1];
    return f(tab[l][lg],tab[r-(1<<lg)+1][lg]);
  }
};
```

**———---------LCA ———---------**

```
int n, l;
vector<vector<int>> adj;
int timer;
vector<int> tin, tout;
vector<vector<int>> up;
void dfs(int v, int p)
{
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }
    tout[v] = ++timer;
}
bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u] >=
tout[v];
}
int lca(int u, int v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}
void preprocess(int root) {
    tin.resize(n);
    tout.resize(n);
```

```
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}
```

**——————Extended euclidean:——————**

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
```

**——————Linear Diophantine——————**

```
// extended euclidean algo here

bool find_any_solution(int a, int b, int c, int
&x0, int &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }
    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
```

**—————————— MOD INVERSE ——————————**

```
// extended euclidean algo here
// gcd(a,m) should be 1
// solution of ax+my=1 as (ax)%m=(1)%m
bool modi(int a,int m,int &ai) {
 int x, y;
 int g = gcd(a, m, x, y);
 if (g != 1) {
  return false;
 } else {
  ai = (x % m + m) % m;
 }
 return true;
}
```

**—————————— BITSET ——————————**

```
set() - Set the bit value given index to 1.
reset() - Set the bit value given index to 0.
flip() - Flip the bit value at the given index.
count() - Count the number of set bits.
test() - Returns boolean value at given index.
any() - Checks if any bit is set.
none() - Checks if none bit is set.
all() - Check if all bit is set.
size() - Returns the size of the bitset.
to_string() - Converts bitset to string.
to_ulong() - Converts bitset to ulong.
to_ullong() - Converts bitset to ull.
```

**——————— LIS LENGTH- N*log(N) ——————————**

```
vector<int> last(N + 1);
int inf = 1e8;
for (int i = 1; i <= N; i++)
    last[i] = inf;
last[0] = -inf;
for (int i = 1; i <= N; i++) {
    int idx = lower_bound(last.begin(),
last.end(), H[i - 1])-last.begin();
    last[idx] = H[i-1];
}
int lis_len = 0;
for (int i = 1; i <= N; i++)
    if (last[i] != inf)
```

```
        lis_len = i;

 ---------- BINARY TRIE ----------
const int BLen = 31;

struct BTNode {
    BTNode* child[2];
    BTNode *par;
    int leafCnt, prefixCnt;

    BTNode() {
        leafCnt = prefixCnt = 0;
        par = NULL;
        child[0] = child[1] = NULL;
    }
};

struct BinaryTrie{
    BTNode *dummy;

    BinaryTrie(){dummy = new BTNode();}

    void add_num(int x) {
        BTNode *itr = dummy;
        for(int i= BLen - 1; i>=0; i--){
            int idx = (x >> i)&1;
            itr->prefixCnt++;
            if(itr->child[idx] == NULL) {
                BTNode *newBTNode =
                                    new BTNode();
                newBTNode->par = itr;
                itr->child[idx] = newBTNode;
            }
            itr = itr->child[idx];
        }
+
        itr->prefixCnt++;itr->leafCnt++;
        dummy->prefixCnt--;
    }

// Find the closest number to x
// (whose xor with x is min.)

ll closestNum(ll x) {

    ll res = 0;
    BTNode *itr = dummy;
    for(int i= BLen - 1; i>=0; i--){

        int idx = (x >> i)&1;
        if(itr->child[idx] == NULL){
                res |= (1 << i);
                itr = itr->child[1 - idx];}
        else itr = itr->child[idx];
    }
    return res;
}

void delete_string(ll x) {
    BTNode *itr = dummy;
    vector<BTNode*> delBTNodes;
    for(int i= BLen - 1; i>=0; i--){
        int idx = (x >> i)&1;
        itr->prefixCnt--;itr = itr->child[idx];
        if(itr->prefixCnt == 1){
            itr->par->child[idx] = NULL;
            delBTNodes.push_back(itr);
        }
    }
    itr->prefixCnt--;itr->leafCnt--;
    for(auto &nn:delBTNodes)    delete nn;
    dummy->prefixCnt++;
}

// How many strings have this as prefix
ll prefixCnt(ll x) {
    BTNode *itr = dummy;
    for(int i= BLen - 1; i>=0; i--){
        int idx = (x >> i)&1;
        if(itr->child[idx] == NULL) return 0;
```

```
            itr = itr->child[idx];
        }
        return itr->prefixCnt;
    }
};
```

—---------- **PAIR_HASH**----------

```
struct pair_hash
{
    template <class T1, class T2>
    size_t operator() (const pair<T1, T2>
&pair) const {
        return std::hash<T1>()(pair.first) ^
std::hash<T2>()(pair.second);
    }
};
void solve()
{
    unordered_map<pii, int, pair_hash> cnt;
    cnt[ {1, 2}]++;
    cout<<cnt[{1,2}];
}
```

**—------ BASIS VECTOR —----------**

```
const int M = 31;
int basis[M];
int cnt;
void init() {
 memset(basis, 0, sizeof basis);
 cnt = 0;
}
bool insertVector(int val) {
 for (int j = M - 1; j >= 0; j--) {
  if (!(val & (1 << j))) continue;
  if (basis[j] == 0) {
   basis[j] = val;
   cnt++;
   return true;
  }
  val ^= basis[j];
 }
 return false;
}
int max_ele() {
 int ans = 0;
 for (int j = M - 1; j >= 0; j--) {
  if (ans & (1 << j)) continue;
  ans ^= basis[j];
 }
 return ans;
}
int kth_ele(int k){
 int ans=0;
 int rem=cnt;
 for(int j=M-1;j>=0;j--) {
  if (!basis[j]) continue;
  rem--;
  if (ans & (1 << j)) {
   if ((1 << rem) >= k) {
    ans ^= basis[j];
   }else{
    k-=(1<<rem);
   }
  } else {
   if ((1 << rem) < k) {
    ans ^= basis[j];
    k -= (1 << rem);
   }
  }
 }
 return ans;
}
bool is_in_space(int x) {
 for (int j = M - 1; j >= 0; j--) {
  if (!(x & (1 << j))) continue;
  if (!basis[j]) return false;
  x ^= basis[j];
 }
 return true;
}
```

**—----------MATRIX EXPO —----------**

```
class mat {
public:
 vector<vector<int>> m;
 int n;
 explicit mat(int _n) {
  n = _n;
  m.resize(n, vector<int>(n, 0));
 }
 mat operator*(mat const &b) {
  mat ans(n);
  for (int i = 0; i < n; i++) {
   for (int j = 0; j < n; j++) {
    for (int k = 0; k < n; k++) {
     (ans.m[i][j] += m[i][k] * b.m[k][j]) %=
mod;
    }
   }
  }
  return ans;
 }
 vector<int> operator*(vector<int>&b)
const {
  vector<int> ans(n, 0);
  for (int i = 0; i < n; i++) {
   for (int j = 0; j < n; j++) {
    (ans[i] += m[i][j] * b[j])%=mod;
   }
  }
  return ans;
 }
 mat power(int p) {
  mat ans(n);
  mat curr = *this;
  for (int i = 0; i < n; i++) {
   ans.m[i][i] = 1;
  }
  while (p) {
   if (p & 1) {
    ans = ans * curr;
   }
   p /= 2;
   curr = curr * curr;
  }
  return ans;
 }
};
```

**—----------STRING HASHING—---------**

```
const int base = 29;
const int mod1 = 1e9+7;
const int mod2 = 1e9+9;
string text,pattern;
#define N 1000000
pii prefix[1+N];
int pw1[1+N];
int pw2[1+N];
void _build(){
        //text
   pw1[0] = 1;
   pw2[0] = 1;
   for(int i = 1;i <= text.size();++i){
        pw1[i] = mul(pw1[i-1],base,mod1);
        pw2[i] = mul(pw2[i-1],base,mod2);
   }

    prefix[0].ff = 0;
    prefix[0].ss = 0;
    for(int i = 1;i <= text.size();++i)   {
         prefix[i].ff = prefix[i-1].ff;
add_self(prefix[i].ff,mul(text[i-1]-'a',pw1[i-1
],mod1),mod1);
        prefix[i].ss = prefix[i-1].ss;
add_self(prefix[i].ss,mul(text[i-1]-'a',pw2[i-
1],mod2),mod2);
        }
}
pii _hash(){
        //patern
```

```
        int ans1 = 0,ans2 = 0;
        for(int i = 0;i < pattern.size();++i){
add_self(ans1,mul(pattern[i]-'a',pw1[i],mod1
),mod1);
add_self(ans2,mul(pattern[i]-'a',pw2[i],mod2
),mod2);
        }
        return {ans1,ans2};
}
pii _hash(int l,int r){
        //text
        int ans1 =
mul(sub(prefix[r].ff,prefix[l-1].ff,mod1),mo
dInverse(pw1[l-1],mod1),mod1);
        int ans2 =
mul(sub(prefix[r].ss,prefix[l-1].ss,mod2),mo
dInverse(pw2[l-1],mod2),mod2);
        return {ans1,ans2};
}
```

---------- **NTT - FFT** ----------

```
int mod = 998244353;
int G = 3;
using cd = complex < double > ;
const double PI = acos(-1);

// m = 9223372036737335297, g = 3
// m = 18446744069414584321, g = 7

ll power(ll x, ll n , ll mod)
{
   if(n==0) return 1;
   if(x%mod==0) return 0;
   n = n%(mod-1);
   ll pow = 1;
   while (n)
   {
      if (n & 1)
         pow = (pow*x)%mod;
      n = n >> 1;
      x = (x*x)%mod;
   }
   return pow;
}

ll imod(ll n , ll mod)
{
   if(n == 1) return 1;
   return (mod - (ll)(mod/n) * imod(mod%n
, mod) % mod) % mod;
}


void fft(vector < cd > & a, bool invert) {
  ll n = a.size();
  for (ll i = 1, j = 0; i < n; i++) {
    ll bit = n >> 1;
    for (; j & bit; bit >>= 1)
      j ^= bit;
    j ^= bit;

    if (i < j)
      swap(a[i], a[j]);
  }
  for (ll len = 2; len <= n; len <<= 1) {
    double ang = 2 * PI / len * (invert ? -1 :
1);
    cd wlen(cos(ang), sin(ang));
    for (ll i = 0; i < n; i += len) {
      cd w(1);
      for (ll j = 0; j < len / 2; j++) {
        cd u = a[i + j], v = a[i + j + len / 2] * w;
        a[i + j] = u + v;
        a[i + j + len / 2] = u - v;
        w *= wlen;
      }
    }
  }
  if (invert) {
    for (cd & x: a)
      x /= n;
```

```
  }
}
void NTT(vector < ll > & a, ll len, ll opt,
vector < ll > & rev) {
  for (ll i = 0; i < len; i++)
    if (i < rev[i])
      swap(a[i], a[rev[i]]);
  for (ll i = 1; i < len; i <<= 1) {
    ll wn = power(G, (opt * ((mod - 1) / (i <<
1)) + mod - 1) % (mod - 1), mod);
    ll step = i << 1;
    for (ll j = 0; j < len; j += step) {
      ll w = 1;
      for (ll k = 0; k < i; k++, w = (1ll * w *
wn) % mod) {
        ll x = a[j + k];
        ll y = 1ll * w * a[j + k + i] % mod;
        a[j + k] = (x + y) % mod;
        a[j + k + i] = (x - y + mod) % mod;
      }
    }
  }
  if (opt == -1) {
    ll r = imod(len, mod);
    for (ll i = 0; i < len; i++)
      a[i] = 1ll * a[i] * r % mod;
  }
}
vector < ll > multiply(vector < ll > & a,
vector < ll > & b) {
  ll n = a.size() - 1, m = b.size() - 1;
  ll tot = m + n;
  ll l = 0, len = 1;
  while (len <= tot) {
    len <<= 1;
    l++;
  }
  a.resize(len), b.resize(len);
  vector < ll > rev(len, 0), res(len, 0);
  for (ll i = 0; i < len; i++)
    rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (l
- 1));
  NTT(a, len, 1, rev); NTT(b, len, 1, rev);
  for (ll i = 0; i < len; i++)
    res[i] = (ll)(a[i] * b[i]) % mod;
  NTT(res, len, -1, rev);
  res.resize(tot + 1);
  return res;
}
```

**——————————Z FUNCTION ——————————**

```
vector<int> z_function(string s) {
 int n = s.length();
 vector<int> z(n, 0);
 int l = 0, r = 0;
 for (int i = 1; i < n; i++) {
  z[i] = max((int) 0, min(r - i + 1, z[i - l]));
  while (i + z[i] < n && s[z[i]] == s[i + z[i]])
{
   z[i]++;
  }
  if (i + z[i] - 1 > r) {
   l = i;
   r = i + z[i] - 1;
  }
 }
 return z;
}
```

**—————————— MERGE SORT TREE ———————**

```
#define N 100005
vector<ll>Tree[5*N];
void build_tree (vector<ll> &a, int cur,int
l,int r)  {
  if(l==r)  {
    Tree[cur].push_back(a[l]);
    return ;
  }
  int mid = l+(r-l)/2;
  build_tree(a,2*cur+1 , l , mid );
  build_tree(a,2*cur+2 , mid+1 ,r);
```

```
  merge(Tree[2*cur+1].begin(),
Tree[2*cur+1].end(),Tree[2*cur+2].begin(),
Tree[2*cur+2].end(),back_inserter(Tree[cur]
)); //Merging the two sorted arrays
}
ll query(int cur,int l,int r,int x,int y,int k) {
  if(r<x||l>y)
    return 0;
  if(x<=l && r<=y) {
    return
lower_bound(Tree[cur].begin(),Tree[cur].en
d(),k)-Tree[cur].begin();
  }
  int mid=l+(r-l)/2;
  return
query(2*cur+1,l,mid,x,y,k)+query(2*cur+2,
mid+1,r,x,y,k);
}
```

**——————————TOPO SORT——————————**

```
queue<int> q;
vector<int> indegree(n, 0);
for (int i = 0; i < n; i++) for (auto it : adj[i])
indegree[it]++;
for (int i = 0; i < n; i++) if (indegree[i] == 0)
q.push(i);
vector<int> topo;
while (!q.empty()) {
   int node = q.front();
   q.pop();
   topo.push_back(node);
   for (auto it : adj[node]) {
      indegree[it]--;
      if (indegree[it] == 0) {
         q.push(it);
      }
   }
}
```

**———————— CHAR TRIE ————————**

```
class TrieNode {
public:
   TrieNode *next[26];
   bool is_word;
   int cnt;
   TrieNode(bool b = false) {
      memset(next, 0, sizeof(next));
      is_word = b;
      cnt=0;
   }
};

class Trie {
public:
   TrieNode *root;
   Trie() { root = new TrieNode();}
   void insert(string s) {
         Node *p = root;
      for(int i = 0; i < s.size(); ++ i) {
         if(p -> next[s[i] - 'a'] == NULL)
            p -> next[s[i] - 'a'] = new
TrieNode();
         p = p -> next[s[i] - 'a'];
         p->cnt++;
      }
      p -> is_word = true;
   }
   bool search(string key) {
      TrieNode *p = find(key);
      return p != NULL && p -> is_word;
   }
   int startsWith(string prefix) {
      TrieNode* prefNode=find(prefix);
      return prefNode==NULL ? 0LL :
prefNode->cnt;
   }
   int get_sum(string key) {
      TrieNode *p = root;
      int sum=0;
```

```
        for(int i = 0; i < key.size(); ++ i) {
            p = p -> next[key[i] - 'a'];
            sum += p->is_word;
        }
        return sum;
    }
    TrieNode* find(string key) {
        TrieNode *p = root;
        for(int i = 0; i < key.size() && p !=
NULL; ++ i)
            p = p -> next[key[i] - 'a'];
        return p;
    }
};
```

**—---------- KMP —----------—-**

```
// pi[i] => longest length j s.t. s[0,1..,j-1] =
s[i-j+1,..,i]
// to search pat in s, calc pi array for string
t=pat+"#"+s => pat ends at i in t if
pi[i]=pat.size()
vector<int> prefix_function(string s) {
 int n = (int)s.length();
 vector<int> pi(n);
 for (int i = 1; i < n; i++) {
  int j = pi[i-1];
  while (j > 0 && s[i] != s[j])
   j = pi[j-1];
  if (s[i] == s[j])
   j++;
  pi[i] = j;
 }
 return pi;
}
// aut[i][j] => (new pi value on appending
character j after string position with pi value
i)
void compute_automaton(string s,
vector<vector<int>>& aut) {
 s += '#';
 int n = s.size();
 vector<int> pi = prefix_function(s);
 aut.assign(n, vector<int>(26));
 for (int i = 0; i < n; i++) {
  for (int c = 0; c < 26; c++) {
   if (i > 0 && 'a' + c != s[i])
    aut[i][c] = aut[pi[i-1]][c];
   else
    aut[i][c] = i + ('a' + c == s[i]);
  }
 }
}

// search fn for automaton
int search(string& arr,string& pattern) {

    vector<vector<int>> aut;
    compute_automaton(pattern,aut);
    int cnt = 0;
    int m = (int)pattern.size();
    int n = (int)arr.size();
    pattern += "#";

    int j = 0,i = 0;
    for(int i = 0 ; i < n ; i ++) {
        j = aut[j][arr[i] - 'a'];
        if(j == m) cnt++;
    }
    return cnt;
}
```

**—----------LINEAR SIEVE----------**

```
const int N = 10000000;
vector<int> lp(N+1);
vector<int> pr;
for (int i=2; i <= N; ++i) {
    if (lp[i] == 0) lp[i] = i, pr.push_back(i);
    for (int j=0; j < (int)pr.size() && pr[j] <=
lp[i] && i*pr[j] <= N; ++j) {
        lp[i * pr[j]] = pr[j];
    }
```

```
}
```

**——————————DIGIT DP ——————————**

```
string a , b;
// numbers >= a , <= b
// init: TTF  p1 -> parameter
// c1,c2 constraint checks

int n = 20;
ll dp[20][180][2][2][2];

ll digit_dp
(int index , ll p1 , bool c1 , bool c2  , bool st)
{
   ll ans = 0;
   if(index == -1) {
          // basecase
   }

   if(dp[index][p1][c1][c2][st] != -1)
           return dp[index][p1][c1][c2][st];

   for(int d = 0 ; d <= 9 ; d ++) {
      bool cc1 = c1&((a[n-index-1]-'0')>d);
      bool cc2 = c2&((b[n-index-1]-'0')<d);

   if(!cc1 && !cc2) {
     bool cc1 = c1&((a[n-index-1]-'0') >= d);
     bool cc2 = c2&((b[n-index-1]-'0') <= d);

     ll new_p1 = p1 + d;

     ans += digit_dp
(index-1 , new_p1 , cc1 , cc2  , st|(d!=0));
     if(ans >= mod) ans -= mod;
    }
  }
  return dp[index][p1][c1][c2][st] = ans;
}
```

**——---------- FENWICK TREE 2D ——-------**

```
class fente2d {
public:
 int n, m;
 vector<vector<int>> tree;
 void init(int _n, int _m) {
  n = _n;
  m = _m;
  tree.resize(n + 5, vector<int>(m + 5, 0));
 }
 void update(int x, int y, int inc) {
  for (int i = x; i <= n; i += i & -i) {
   for (int j = y; j <= m; j += j & -j) {
    tree[i][j] += inc;
   }
  }
 }
 int query_pref(int x, int y) {
  int tot = 0;
  for (int i = x; i >= 1; i -= i & -i) {
   for (int j = y; j >= 1; j -= j & -j) {
    tot += tree[i][j];
   }
  }
  return tot;
 }
 int query(int x1, int y1, int x2, int y2) {
  int tot = query_pref(x2,y2);
  tot -= query_pref(x2,y1-1);
  tot -= query_pref(x1-1,y2);
  tot += query_pref(x1-1,y1-1);
  return tot;
 }
};
```

**——————————— SUFFIX ARRAY ————————**

```
struct sufar{
 string s;
 vector<int>lcp,order,rank;
 int n;
 sufar(string _s){
```

```
 s=_s+"$";
 n=s.length();
}
void build(){
 order.resize(n);
 rank.resize(n);
 {
  vector<pair<int,int>>temp;
  for(int i=0;i<n;i++){
   temp.push_back({s[i]-'a',i});
  }
  sort(temp.begin(),temp.end());
  for(int i =0;i<n;i++){
   order[i]=temp[i].second;
  }
  rank[order[0]]=0;
  for(int i=1;i<n;i++){

rank[order[i]]=rank[order[i-1]]+(temp[i].firs
t!=temp[i-1].first);
  }
 }
 int k=0;
 vector<int>order_t(n,0),rank_t(n,0);
 while((1<<k)<n){
  for(int i =0;i<n;i++){
   (order[i]-=(1<<k)-n)%=n;
  }
  vector<int>cnt(n,0),pos(n,0);
  for(auto &c:rank)
   cnt[c]++;
  for(int i=1;i<n;i++)
   pos[i]=pos[i-1]+cnt[i-1];
  for(int i=0;i<n;i++)
   order_t[pos[rank[order[i]]]++]=order[i];
  order=order_t;
  for(int i=1;i<n;i++){

pair<int,int>old_val={rank[order[i-1]],rank[
(order[i-1]+(1<<k))%n]};

pair<int,int>new_val={rank[order[i]],rank[(
order[i]+(1<<k))%n]};

rank_t[order[i]]=rank_t[order[i-1]]+(old_val
!=new_val);
  }
  rank=rank_t;
  k++;
 }
}
void build_lcp(){
 lcp.resize(n,0);
 int k=0;
 for(int i=0;i<n-1;i++){
  int pos=rank[i];
  int j=order[pos-1];
  while(s[i+k]==s[j+k]) k++;
  lcp[pos]=k;
  k=max(k-1,(int)0);
 }
}
int occ_of_string(string &p){
 int m=p.length();
 int l=-1,r=n;
 while(l+1<r){
  int mid=(l+r)/2;
  if(s.substr(order[mid],m)<p){
   l=mid;
  }else{
   r=mid;
  }
 }
 int left=r;
 l=-1,r=n;
 while(l+1<r){
  int mid=(l+r)/2;
  if(s.substr(order[mid],m)<=p){
   l=mid;
  }else{
```

```
   r=mid;
  }
 }
 int right=l;
 int val=0;
 if(left<=right){
  val=right-left+1;
 }
 return val;
}
int different_substr(){
 build_lcp();
 int tot=(n*(n-1))/2;
 for(int i=1;i<n;i++) tot-=lcp[i];
 return tot;
}
};
```

**—------- PHI PRECALCULATE —-----—**

```
int phi(int n){
 int res=n;
 for(int i=2;i*i<=n;i++){
  if(n%i==0){
   while(n%i==0){
    n/=i;
   }
   res-=res/i;
  }
 }
 if(n>1) res-=res/n;
 return res;
}
//using divisor sum property
//eg: divisors of 10-> 1,2,5,10 ->
phi[1]+phi[2]+phi[5]+phi[10]=10
void etf(int n) {
 phi[0] = 0;
 phi[1] = 1;
 for (int i = 2; i <= n; i++)
  phi[i] = i - 1;
 for (int i = 2; i <= n; i++)
  for (int j = 2 * i; j <= n; j += i)
   phi[j] -= phi[i];
}
```

—----------**SCC**----------

```
vector<vector<int>> adj, adj_rev; // adj -
edges ; adj_rev - reverse edges of original
graph
vector<bool> used;
vector<int> order, component;
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);
}
int main() {
    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);
            // ... process component ...
            component.clear();
        }
}
```

**———-PRIMALITY TESTS———-**

```
using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}
bool check_composite(u64 n, u64 a, u64 d,
int s) {
    u64 x = binpower(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) { // returns true if n
is prime, else returns false.
    if (n < 2)
        return false;
    int r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }
    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23,
29, 31, 37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

**———————- SEGTREE 2D ———————-**

```
class segte2d{
public:
 int n,m;
 vector<vector<int>>tr;
 vector<vector<int>>a;
 segte2d(int _n,int
_m,vector<vector<int>>&_a){
  a=_a;
  n=_n;
  m=_m;
  tr.resize(4*n+1,vector<int>(4*m+1,0));
 }
 int f(int x,int y) {
  return x + y;
 }
 void build_y(int vx,int vy,int sx,int ex,int
sy,int ey){
  if(sy==ey) {
   if (sx == ex) tr[vx][vy] = a[sx][sy];
   else tr[vx][vy] = f(tr[2 * vx + 1][vy] , tr[2
* vx + 2][vy]);
   return;
  }
  int my=(sy+ey)/2;
  build_y(vx,2*vy+1,sx,ex,sy,my);
  build_y(vx,2*vy+2,sx,ex,my+1,ey);

tr[vx][vy]=f(tr[vx][2*vy+1],tr[vx][2*vy+2])
;
 }
 void build_x(int vx,int sx,int ex){
  if(sx!=ex){
```

```
   int mx=(sx+ex)/2;
   build_x(2*vx+1,sx,mx);
   build_x(2*vx+2,mx+1,ex);
  }
  build_y(vx,0,sx,ex,0,m-1);
 }
 void build(){
  build_x(0,0,n-1);
 }
 void update_y(int vx,int vy,int sx,int ex,int
sy,int ey,int x,int y,int val){
  if(sy==ey){
   if(sx==ex){
    tr[vx][vy]=f(tr[vx][vy],val);
   }else{

tr[vx][vy]=f(tr[2*vx+1][vy],tr[2*vx+2][vy])
;
   }
   return;
  }
  int mid=(sy+ey)/2;
  if(y<=mid)
   update_y(vx,2*vy+1,sx,ex,sy,mid,x,y,val);
  else

update_y(vx,2*vy+2,sx,ex,mid+1,ey,x,y,val)
;
tr[vx][vy]=f(tr[vx][2*vy+1],tr[vx][2*vy+2])
;
 }
 void update_x(int vx,int sx,int ex,int x,int
y,int val){
  if(sx!=ex){
   int mid=(sx+ex)/2;
   if(x<=mid){
    update_x(2*vx+1,sx,mid,x,y,val);
   }else{
    update_x(2*vx+2,mid+1,ex,x,y,val);
   }
  }
  update_y(vx,0,sx,ex,0,m-1,x,y,val);
 }
 void update(int x,int y,int val){
  update_x(0,0,n-1,x,y,val);
 }
 int query_y(int vx,int vy,int sy,int ey,int
qsy,int qey){
  if(qey<sy || qsy>ey){
   return 0;
  }
  if(sy>=qsy && ey<=qey){
   return tr[vx][vy];
  }
  int my=(sy+ey)/2;
  int A=query_y(vx,2*vy+1,sy,my,qsy,qey);
  int
B=query_y(vx,2*vy+2,my+1,ey,qsy,qey);
  return f(A,B);
 }
 int query_x(int vx,int qsx,int qex,int qsy,int
qey,int sx,int ex){
  if(qsx>ex || qex<sx)
   return 0;
  if(sx>=qsx && ex<=qex){
   return query_y(vx,0,0,m-1,qsy,qey);
  }
  int mx=(sx+ex)/2;
  int
A=query_x(2*vx+1,qsx,qex,qsy,qey,sx,mx)
;
  int
B=query_x(2*vx+2,qsx,qex,qsy,qey,mx+1,e
x);
  return f(A,B);
 }
 int query(int sx,int ex,int sy,int ey){
  return query_x(0,sx,ex,sy,ey,0,n-1);
 }
};
```

**——————————MO'S ALGO——————————**

```
const int block_size = 200;
const int maxn = 3e4 + 10;
const int maxai = 1e6 + 10;
int ans;
int v[maxn];
int freq[maxai];
void remove(int x) {
 freq[x]--;
 if (!freq[x])
  ans--;
}
void add(int x) {
 freq[x]++;
 if (freq[x] == 1)
  ans++;
}
int get_answer() {
 return ans;
}
// bool cmp(pair<int, int> p, pair<int, int> q)
{
//  if (p.first / BLOCK_SIZE != q.first /
BLOCK_SIZE)
//   return p < q;
//  return (p.first / BLOCK_SIZE & 1) ?
(p.second < q.second) : (p.second >
q.second);
// }
struct Query {
 int l, r, idx;
 bool operator<(Query other) const {
  return make_pair(l / block_size, r) <
make_pair(other.l / block_size, other.r);
 }
};
vector<int> mo_s_algorithm(vector<Query>
queries) {
 vector<int> answers(queries.size());
 sort(queries.begin(), queries.end());
 // TODO: initialize data structure
 int cur_l = 0;
 int cur_r = -1;
 // invariant: data structure will always
reflect the range [cur_l, cur_r]
 for (Query q : queries) {
  while (cur_l > q.l) {
   cur_l--;
   add(v[cur_l]);
  }
  while (cur_r < q.r) {
   cur_r++;
   add(v[cur_r]);
  }
  while (cur_l < q.l) {
   remove(v[cur_l]);
   cur_l++;
  }
  while (cur_r > q.r) {
   remove(v[cur_r]);
   cur_r--;
  }
  answers[q.idx] = get_answer();
 }
 return answers;
}
void solve() {
 int n;
 cin >> n;
 forn(i, n) cin >> v[i];
 int q;
 cin >> q;
 vector<Query> ask(q);
 forn(i, q) {
  ask[i].idx = i;
  cin >> ask[i].l >> ask[i].r;
  ask[i].l--;
  ask[i].r--;
 }
```

```
 vector<int> ansvec = mo_s_algorithm(ask);
 forn(i, q) cout << ansvec[i] << endl;
}
```

**——————————SACK ON TREE——————————**

```
// sack on tree (small to large merging)

#define maxsz 200005
vi adjlst[maxsz];
ll ans[maxsz];
ll a[maxsz];
// dist guys in subtree
struct info {
  set<ll> store;
  info(ll val) {
    // init
    store.insert(val);
  }
};

info *ptr[maxsz];

info* merge(info *a , info *b) {
  if(a->store.size() < b->store.size()) {
    swap(a , b);
  }
  for(auto k:b->store) {
    // process
    a->store.insert(k);
  }
  return a;
}

void dfs(int u , int p) {
  for(int v : adjlst[u]) {
    if(v!=p) {
      dfs(v , u);
      ptr[u] = merge(ptr[u] , ptr[v]);
    }
  }
  ans[u] = (ptr[u]->store).size();
}
```

**--CHINESE REMAINDER THEOREM--**

```
ll GCD(ll a, ll b) { return (b == 0) ? a :
GCD(b, a % b); }
inline ll LCM(ll a, ll b) { return a / GCD(a,
b) * b; }
inline ll normalize(ll x, ll mod) { x %= mod;
if (x < 0) x += mod; return x; }
struct GCD_type { ll x, y, d; };
GCD_type ex_GCD(ll a, ll b)
{
 if (b == 0) return {1, 0, a};
 GCD_type pom = ex_GCD(b, a % b);
 return {pom.y, pom.x - a / b * pom.y,
pom.d};
}
// returns {x,lcm} where x:[0,lcm) satisfying
all equations else {-1,-1}
pll crt(vector<pll> &v) // x = v[0] % v[1]
{
 ll n=v.size();
 pll ans = v[0];
 for(ll i=1;i<n;i++)
 {
  auto pom = ex_GCD(ans.second,
v[i].second);
  ll x1 = pom.x;
  ll d = pom.d;
  if((v[i].first - ans.first) % d != 0)
   return {-1,-1};
  ans.first = normalize(ans.first + x1 *
(v[i].first - ans.first) / d % (v[i].second / d) *
ans.second, ans.second * v[i].second / d);
  ans.second = LCM(ans.second,
v[i].second); // you can save time by
replacing above lcm * n[i] /d by lcm = lcm *
n[i] / d
```

```
 }
 return ans;
}
```

**——————— Smallest prime sieve ——————————**

```
int sieve[((int)1e7)+1];

void fill_sieve(int n){
   for(int i = 1;i <= n;i++)
     sieve[i] = i;

   for(int i = 2;i <= n;i++){
     if(sieve[i] == i){
       for(int mul = 2;mul*i <= n;mul++){
         sieve[i*mul] = min(sieve[i*mul],i);
       }
     }
   }
}
```

**——————————— Floyd-Warshall ———————————**

```
#define N
int d[1+N][1+N];

void floyd(int n){
  for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i)    {
       for (int j = 0; j < n; ++j)   {
           if (d[i][k] < inf && d[k][j] < inf)
d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
                }
             }
       }
}
```

**——————————— String hashing ———————————**

```
const int base = 29;
const int mod1 = 1e9+7;
const int mod2 = 1e9+9;

string text,pattern;

#define N 1000000
pii prefix[1+N];
int pw1[1+N];
int pw2[1+N];
void _build(){
          //text
   pw1[0] = 1;
   pw2[0] = 1;

   for(int i = 1;i <= text.size();++i){
         pw1[i] = mul(pw1[i-1],base,mod1);
         pw2[i] = mul(pw2[i-1],base,mod2);
   }

    prefix[0].ff = 0;
    prefix[0].ss = 0;

    for(int i = 1;i <= text.size();++i)   {
          prefix[i].ff = prefix[i-1].ff;

add_self(prefix[i].ff,mul(text[i-1]-'a',pw1[i-1
],mod1),mod1);

          prefix[i].ss = prefix[i-1].ss;

add_self(prefix[i].ss,mul(text[i-1]-'a',pw2[i-
1],mod2),mod2);
          }
}

pii _hash(){
          //patern
          int ans1 = 0,ans2 = 0;
          for(int i = 0;i < pattern.size();++i){

add_self(ans1,mul(pattern[i]-'a',pw1[i],mod1
),mod1);

add_self(ans2,mul(pattern[i]-'a',pw2[i],mod2
),mod2);
```

```
        }
        return {ans1,ans2};
}




pii _hash(int l,int r){  //text
    int ans1 =
mul(sub(prefix[r].ff,prefix[l-1].ff,mod1),mo
dInverse(pw1[l-1],mod1),mod1);
    int ans2 =
mul(sub(prefix[r].ss,prefix[l-1].ss,mod2),mo
dInverse(pw2[l-1],mod2),mod2);
        return {ans1,ans2};
}
```

**—-—— BRIDGE ARTICULATION —------**

```
// articlaution point template
#define maxsz 200005

vi visited(maxsz);
vi adjlst[maxsz];
vi low(maxsz);
vi disc(maxsz);
vi ap(maxsz);

int Counter = 1;

int dfs(int u , int p) {
  low[u] = disc[u] = Counter++;
  int child = 0;
  for(int v : adjlst[u]) {
    if(v!=p) {
      if(!disc[v]) {
        child++;
        dfs(v , u);
        if(disc[u] <= low[v]) {
          // change to < for bridges
          // if < satisfies then (u,v) is bridge
          ap[u] = 1;
        }
        low[u] = min(low[u] , low[v]);
      }
      else {
        low[u] = min(low[u] , disc[v]);
      }
    }
  }
  return child;
}

void solve(){
  // adjlst input
  for(int i = 1 ; i <= n ; i ++) {
    if(!disc[i]) {
      Counter = 1;
      ap[i] = (dfs(i , i) > 1);
    }
  }

// ap[i] = 1 (Arti point)
// ap[i] = 0 (Not AP)
}
```

**—------- BELLMAN FORD —----------**

```
struct Edge{
        int a,b,cost;
};
cin >> n >> m;
vector<Edge> edges;
for(int i = 0;i < m;++i){
    int a,b,c;cin >> a >> b >> c;
     --a;
     --b;
     edges.pb(Edge{a,b,c});
```

```
}
vl d(n, inf);
d[0] = 0;
vector<int> p(n, -1);
int x;
for (int i = 0; i < m; ++i) {
  x = -1;
  for (Edge e : edges)
  /*comment the if condition below for
finding     cycles even in disconnected
components*/
  if (d[e.a] < inf)
  if (d[e.b] > d[e.a] + e.cost) {
     d[e.b] = max(-inf, d[e.a] + e.cost);
     p[e.b] = e.a;
     x = e.b;
    }
}
if (x == -1)
cout << "NO";
else {
       cout << "YES\n";
int y = x;
for (int i = 0; i < n; ++i)
     y = p[y];
vector<int> path;
for (int cur = y;; cur = p[cur]) {
   path.push_back(cur);
if (cur == y && path.size() > 1)
   break;
}
reverse(path.begin(), path.end());
for (int u : path)
   cout << u+1 << ' ';
}
```

**—---------- DIJKSTRA—----------—-**

```
cin >> n >> m;
alist.resize(n);
for(int i = 0;i < m;++i){
   int a,b,c;cin >> a >> b >> c;
    --a; --b;
   alist[a].pb({b,c});
}
vl dist(n,inf);
pllpqs q;
q.push({0, 0});
dist[0] = 0;
while(!q.empty()){
  auto curr = q.top();
  q.pop();
   if(curr.ff > dist[curr.ss]){
          continue;
   }
   for(auto next : alist[curr.ss]){
         if(dist[next.ff] > curr.ff + next.ss){
           dist[next.ff] = curr.ff + next.ss;
           q.push({dist[next.ff],next.ff});
         }
      }
}
for(int i = 0;i < n;++i)
     cout << dist[i] << " ";
cout << "\n";
```

**—---------- DINIC FLOW —----------**

```
#define MAXLIM 1005

ll maxflow;
ll adjflow[MAXLIM][MAXLIM];
int parent[MAXLIM];
int visited[MAXLIM];
int n , m;

void addedge(int u , int v , int c) {
```

```
  adjflow[u][v] += c;
}
int start = 0 , stop = 0;

bool reachable() {
  for(int i = start ; i <= stop ; i ++)
                                      visited[i] = 0;
  for(int i = start ; i <= stop ; i ++)
                                     parent[i] = -1;
  queue<int> q;
  q.push(start);
  visited[start] = 1;

  while(q.size()) {
    int u = q.front();
    q.pop();
    for(int j = start ; j <= stop ; j ++) {
      if(adjflow[u][j] && !visited[j]) {
        visited[j] = 1;
        q.push(j);
        parent[j] = u;
      }
    }
  }
  return visited[stop];
}

void dinic() {

 maxflow = 0;
 while(reachable()) {

   ll minfreeflow = INT_MAX;
   for(int v = stop ; v != start; v = parent[v]){
       int u = parent[v];
       minfreeflow =  min(minfreeflow ,
adjflow[u][v]);
     }
     maxflow += minfreeflow;

   for(int v = stop ; v != start; v = parent[v]){
       int u = parent[v];
       adjflow[u][v] -=  minfreeflow;
       adjflow[v][u] +=  minfreeflow;
    }
  }
}
void initialize() {
  start = 0;
  stop = n-1;
  for(int i = start ; i <= stop ; i ++) {
    for(int j = start ; j <= stop; j ++) {
      adjflow[i][j] = 0;
    }
    parent[i] = -1;
  }
}
```