WE CREATE PROBLEMS

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;

#define ll long long
#define double long double
#define setbits(x) __builtin_popcountll(x)
#define pqb priority_queue<int> // maxheap
#define pqs
priority_queue<int,vector<int>,greater<int>
> // minheap
#define meminf(a) memset(a,0x7f,sizeof(a))
#define precise(x,y)
fixed<<setprecision(y)<<x
#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>
#define osetpii tree<pii, null_type,less<pii>,
rb_tree_tag,tree_order_statistics_node_upda
te>
mt19937
rng(chrono::steady_clock::now().time_since
_epoch().count());
// for long long use mt19937_64 and usage -
just do rng()

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

template<typename T, typename T1>T
amax(T &a, T1 b) {if (b > a)a = b; return a;}
template<typename T, typename T1>T
amin(T &a, T1 b) {if (b < a)a = b; return a;}
inline ll gcd(ll a, ll b) {return (b == 0) ? a :
gcd(b, a % b);}
```

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

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

—-------BIG PRIMES [9-12] DIGITS—----

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

—-----COMBINATORICS - C(N,R)------

```
vector<int>fac, inv, ifac;
void pre(int 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 (int 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);
 }
}
int ncr(int n, int 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<int> tree;
 void init(int _n) {
  n = _n;
  tree.resize(n + 5, 0);
 }
 void update(int idx, int inc) {
  for (int i = idx; i <= n; i += i & -i)
   tree[i] += inc;
 }
 int query(int idx) {
  int sum = 0;
  assert(idx<=n);
  for (int i = idx; i >= 1; i -= i & -i)
   sum += tree[i];
  return sum;
 }
};
```

—————————— SEGTREE LAZY ——————————

```
struct node {
```

```
 //req variable
 int val;
 //default value
 node()
 {
  val = inf; // neutral value for query function
 }
};

class segte {
public:
 int n{};
 vector<node> tree; // exact demanded
answer in query
 vector<int> lazy; // value to be propagated
in convenient form
 vector<int> a;
 vector<bool> prsnt;
 node neutral;

 void init(int N) {
  n = N;
  a.resize(n);
  tree.resize(4 * n + 1); // change if array
value's != neutral value of node
  //default values
  lazy.assign(4 * n + 1, 0);
  prsnt.assign(4 * n + 1, false);
 }
 void put(vector<int> &val) {
  a = val;
 }
 static void merge(node &curr, node &left,
node &right) { // query function
  curr.val = min(left.val, right.val);
 }
 void prop(int index, int ss, int se)
 {
  tree[index].val += lazy[index]; // update
function
  if (ss != se)
  {
   lazy[index << 1] += lazy[index];
   lazy[index << 1 | 1] += lazy[index];
   prsnt[index << 1] = prsnt[index << 1 | 1] =
true;
  }
  lazy[index] = 0;
  prsnt[index] = false;
 }

 void build(int index, int ss, int se) {
  if (ss == se) {
   tree[index].val = a[ss];
   return;
  }
  int mid = (ss + se) / 2;
  build(index << 1, ss, mid);
  build(index << 1 | 1, mid + 1, se);
  merge(tree[index], tree[index << 1],
tree[index << 1 | 1]);
 }
 void build() { // do for initial config of array
if req.
  build(1, 0, n - 1);
 }
 node query(int index, int ss, int se, int qs,
int qe)
 {
  if (prsnt[index])
   prop(index, ss, se);
  if (qs > se || qe < ss) return neutral;
  if (qs <= ss && qe >= se) {
   return tree[index];
  }
  int mid = (ss + se) / 2;
  node left = query(index << 1, ss, mid, qs,
qe);
  node right = query(index << 1 | 1, mid + 1,
se, qs, qe);
```

```
 node mer;
 merge(mer, left, right);
 return mer;
}
node query(int l, int r) {
 return query(1, 0, n - 1, l, r);
}

void update(int index, int ss, int se, int l, int
r, int inc) {
 if (prsnt[index])
  prop(index, ss, se);
 if (r < ss || l > se)
  return;
 if (ss >= l && se <= r) {
  prsnt[index] = true;
  lazy[index] = inc; // update function
  prop(index, ss, se);
  return;
 }
 int mid = (ss + se) / 2;
 update(index << 1, ss, mid, l, r, inc);
 update(index << 1 | 1, mid + 1, se, l, r, inc);
 merge(tree[index], tree[index << 1],
tree[index << 1 | 1]);
}

void update(int l, int r, int inc) {
 update(1, 0, n - 1, l, r, inc);
}

node get(int pos, int index, int ss, int se)
{
 if (prsnt[index])
  prop(index, ss, se);
 if (ss == se)
  return tree[index];
 node res;
 int mid = (ss + se) / 2;
 if (pos <= mid)
  res = get(pos, index << 1, ss, mid);
 else
  res = get(pos, index << 1 | 1, mid + 1, se);
 return res;
}

node get(int pos) {
 return get(pos, 1, 0, n - 1);
}
// leftmost index >= l with value >= x
int find_first_greater(int index, int ss, int se,
int l, int x)
{
 if (se < l || tree[index].mx < x)
  return -1;
 if (ss == se)
  return ss;
 int mid = (ss + se) / 2;
 int leftans = find_first_greater(2 * index,
ss, mid, l, x);
 if (leftans != -1)
  return leftans;
 else return find_first_greater(2 * index + 1,
mid + 1, se, l, x);
}
// k'th 1 from left
int find_kth(int k) {
 int index = 1;
 int ss = 0, se = n - 1;
 while (ss < se) {
  int mid = (ss + se) / 2;
  if (tree[2 * index].one < k) {
   k -= tree[2 * index].one;
   index = 2 * index + 1;
   ss = mid + 1;
  }
  else {
   index = 2 * index;
   se = mid;
  }
```

```
  }
  assert(ss == se);
  return se;
 }
};
```

———---------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 —---------------------

```
const int N=200001;
const int M=21;
int par[N][M];
int n;
vector<int>g[N];
int dep[N];
void dfs_par(int node,int p){
 par[node][0]=p;
 for(auto &c:g[node]){
  if(c!=p){
   dep[c]=dep[node]+1;
   dfs_par(c,node);
  }
 }
}
void fill(int root=0){
 dfs_par(root,root);
 for(int j=1;j<M;j++){
  for(int i=0;i<n;i++){
   par[i][j]=par[par[i][j-1]][j-1];
  }
 }
}
int get_kp(int node,int k){
 for(int j=M-1;j>=0;j--){
  if(k&(1ll<<j)){
   if(node!=-1)
    node=par[node][j];
  }
 }
 return node;
}
int lca(int x,int y){
```

```
 if(dep[x]<dep[y]) swap(x,y);
 x= get_kp(x,dep[x]-dep[y]);
 if(x==y) return x;
 for(int j=M-1;j>=0;j--){
  if(par[x][j]!=par[y][j]){
   x=par[x][j];
   y=par[y][j];
  }
 }
 return par[x][0];
}
void solve() {
 int q;
 cin>>n>>q;
 for(int i=1;i<n;i++){
  int val;
  cin>>val;
  --val;
  g[val].push_back(i);
  g[i].push_back(val);
 }
 fill(0);
 while(q--){
  int x,y;
  cin>>x>>y;
  --x;--y;
  cout<<lca(x,y)+1<<endl;
 }
}
```

—---------- MOD INVERSE —----------—

```
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 - (a / b) * y1;
 return d;
}
// 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 ----------

```
class node {
public:
 int left, right, cnt;
 node() {
  left = 0, right = 0, cnt = 0;
 }
};
class BT{
public:
 vector<node>t;
 int M;
 BT(int m=30):M(m) {
  t.emplace_back();
 }
 void insert(int num) {
  int idx = 0;
  t[idx].cnt++;
  for (int i = M; i >= 0; i--) {
   if (num & (1ll << i)) {
    if (!t[idx].right) {
     t[idx].right = sz(t);
     t.emplace_back();
    }
    idx = t[idx].right;
   } else {
    if (!t[idx].left) {
     t[idx].left = sz(t);
     t.emplace_back();
    }
    idx = t[idx].left;
   }
   t[idx].cnt++;
  }
 }
 bool isPresent(int num) {
  int idx = 0;
  for (int i = M; i >= 0; i--) {
   if (num & (1ll << i)) {
    if ((!t[idx].right) || (t[t[idx].right].cnt ==
0)) {
     return false;
    }
    idx = t[idx].right;
   } else {
    if ((!t[idx].left) || (t[t[idx].left].cnt == 0)) {
     return false;
    }
    idx = t[idx].left;
   }
  }
  return true;
 }
 bool remove(int num) {
  if (!isPresent(num)) return false;
  int idx = 0;
  t[idx].cnt--;
  for (int i = M; i >= 0; i--) {
   if (num & (1ll << i)) {
    idx = t[idx].right;
   } else {
    idx = t[idx].left;
   }
   t[idx].cnt--;
  }
  return true;
 }
 int min_xor(int num) {
  int ans = 0;
  int idx = 0;
  if (t[idx].cnt == 0)
   return -1;
  for (int i = M; i >= 0; i--) {
   if (!(num & (1ll << i))) {
    if (t[idx].left &&( t[t[idx].left].cnt > 0)) {
```

```
    idx = t[idx].left;
   } else {
    ans |= (1ll << i);
    idx = t[idx].right;
   }
  } else {
   if (t[idx].right && (t[t[idx].right].cnt > 0))
{
    idx = t[idx].right;
   } else {
    ans |= (1ll << i);
    idx = t[idx].left;
   }
  }
 }
 return ans;
}
};
```

—---------- 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——---------

```
int power(int x, int y, int p) {
 int res = 1;
 x = x % p;
 while (y > 0) {
  if (y & 1)
   res = (res * x) % p;
  y = y >> 1;
  x = (x * x) % p;
 }
 return res;
}
int modi(int a, int m) {
 return power(a, m - 2, m);
}
struct Hash {
 vector<int> pref;
 vector<int> pows;
 vector<int> i_pows;
```

```
 vector<int> pref_s;
 vector<int> pows_s;
 vector<int> i_pows_s;
 int p, m;
 int p_s, m_s;
 explicit Hash(string &s, int _p = 31, int _m
= mod, int __p = 53, int __m = mod2) :
p(_p), m(_m), p_s(__p),
                    m_s(__m) {
  int n = sz(s);
  pows.resize(n + 1, 0);
  pref.resize(n + 1, 0);
  i_pows.resize(n + 1, 0);
  pows_s.resize(n + 1, 0);
  pref_s.resize(n + 1, 0);
  i_pows_s.resize(n + 1, 0);
  pows[0] = 1;
  int p_inv = modi(p, m);
  i_pows[0] = 1;
  pows_s[0] = 1;
  int p_inv_s = modi(p_s, m_s);
  i_pows_s[0] = 1;
  for (int i = 1; i <= n; i++) {
   pref[i] = (pref[i - 1] + (s[i - 1] - 'a' + 1) *
pows[i - 1]) % m;
   pows[i] = (pows[i - 1] * p) % m;
   i_pows[i] = (i_pows[i - 1] * p_inv) % m;
  }
  for (int i = 1; i <= n; i++) {
   pref_s[i] = (pref_s[i - 1] + (s[i - 1] - 'a' + 1)
* pows_s[i - 1]) % m_s;
   pows_s[i] = (pows_s[i - 1] * p_s) % m_s;
   i_pows_s[i] = (i_pows_s[i - 1] * p_inv_s)
% m_s;
  }
 }
 pair<int, int> get(int l, int r) {
  ++l;
  ++r;
  return {((pref[r] - pref[l - 1] + m) *
i_pows[l - 1]) % m,
    ((pref_s[r] - pref_s[l - 1] + m_s) *
i_pows_s[l - 1]) % m_s};
 }
};
```

—---------- EXTENDED GCD —--------

```
// one of the solution for ax+by=gcd(a,b)
int egcd(int a,int b,int &x,int &y){
 if(b==0){
  x=1;
  y=0;
  return a;
 }
 int x1,y1;
 int d=egcd(b,a%b,x1,y1);
 x=y1;
 y=x1-(a/b)*y1;
 return d;
}
```

---------- 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) {
        TrieNode *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]);
  }
 }
}
```

—----------**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** —----------

```
void solve()
{
 string n;
 cin >> n;
 int d;
 cin >> d;
 int len = sz(n);
 vector<vector<int>> dp(d, vector<int>(2));
 // dp[sum][sm_already]
 // sum => sum modulo d till now(curr position)
 // sm_already => current prefix number has become smaller(than n) already?
 dp[0][0] = 1; // => 0 [ _ _ ]
 for (int pos = 0; pos < len; pos++) {
  vector<vector<int>> newdp(d, vector<int>(2));
  for (int sum = 0; sum < d; sum++) {
   for (int sm_already : {0, 1}) {
 for (int dig = 0; dig <= 9; dig++) {
  if (!sm_already && dig > n[pos] - '0')
   break;
  add(newdp[(sum + dig) % d][sm_already || (dig < n[pos] - '0')], dp[sum][sm_already]);
 }
   }
  }
  dp.swap(newdp);
 }
 cout << (dp[0][0] + dp[0][1] - 1 + mod) % mod << endl;
}
```

—---------- **DISCRETE LOG** —----------

```
// find smallest x such that a^x==b %m
// O(sqrt(m))
// doesnt matter if a , m are coprime or not
// beware of a=0
int solve(int a,int b,int m) {
 a %= m;
 b %= m;

 if(a==0) {
  if (b == 0) return 1;
  return -1;
 }

 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 * a / g) % m;
 }

 int n = sqrt(m) + 1;
 int an = 1;
 for (int i = 0; i < n; i++) {
  an *= a;
  an %= m;
 }
 unordered_map<int, int> vals;
 int cur = b;
 for (int j = 0; j <= n; j++) {
  vals[cur] = j;
  cur *= a;
  cur %= m;
 }
 cur = k;
 for (int j = 1; j <= n; j++) {
  cur *= an;
  cur %= m;
  if (vals.count(cur)) {
   return n * j - vals[cur] + add;
  }
 }
 return -1;
}
```

—---------- 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;
}
```

——————————TREAP LAZY——————————

```
// lower priority on top, all methods return
the new treap root
// To add new seg-tree supported properties,
edit recalc()
// To add lazyprop values, edit recalc() and
prop()
// If you only add by merging, skip add() and
recalc()
// If you don't need lazyprop, skip prop() and
rangeAdd()
struct Treap {
 int data;
 int priority;
 array<Treap*, 2> kids;
 int subtreeSize = 1, sum = 0, toProp = 0;
 bool reverseMe = 0;
 Treap(int data);
};
int size(Treap *me) {
 return me == NULL ? 0 : me->subtreeSize;
}
void recalc(Treap *me) {
 if (me == NULL) return;
 me->subtreeSize = 1;
 me->sum = me->data + me->toProp *
size(me);
 for (Treap* t : me->kids) if (t != NULL)
me->subtreeSize += t->subtreeSize;
 for (Treap* t : me->kids) if (t != NULL)
me->sum += t->sum + t->toProp * size(t);
}
void prop(Treap *me) {
 if (me == NULL) return;
 if (!me->reverseMe) return;
 // if (me->toProp == 0) return;
 for (Treap *t : me->kids) if (t != NULL)
t->reverseMe ^= 1;
 // for (Treap *t : me->kids) if (t != NULL)
t->toProp += me->toProp;
 swap(me->kids[0], me->kids[1]);
 // me->data += me->toProp;
 me->reverseMe = 0;
 // me->toProp = 0;
 recalc(me);
}
Treap* merge(Treap *l, Treap *r) {
 if (l == NULL) return r;
 if (r == NULL) return l;
```

```
 prop(l); prop(r);
 if (l->priority < r->priority) {
  l->kids[1] = merge(l->kids[1], r);
  recalc(l);
  return l;
 }
 else {
  r->kids[0] = merge(l, r->kids[0]);
  recalc(r);
  return r;
 }
}
array<Treap*, 2> split(Treap *me, int
nInLeft) {
 //returns lefthalf, rightHalf
 //nInLeft is size of left treap, aka index of
first thing in right treap
 if (me == NULL) return {NULL, NULL};
 prop(me);
 if (size(me->kids[0]) >= nInLeft) {
  array<Treap*, 2> leftRes =
split(me->kids[0], nInLeft);
  me->kids[0] = leftRes[1];
  recalc(me);
  return {leftRes[0], me};
 }
 else {
  nInLeft = nInLeft - size(me->kids[0]) - 1;
  array<Treap*, 2> rightRes =
split(me->kids[1], nInLeft);
  me->kids[1] = rightRes[0];
  recalc(me);
  return {me, rightRes[1]};
 }
 return {NULL, NULL};
}
Treap::Treap(int data) {
 kids = {NULL, NULL};
 this->data = data;
 this->priority = rng(); //
priority=random(decide for int vs ll)
 recalc(this);
}
// Treap* rangeAdd(Treap* t, int l, int r, int
toAdd) {
//  array<Treap*, 2> a = split(t, l), b =
split(a[1], r - l + 1);
//  b[0]->toProp += toAdd;
//  return merge(a[0], merge(b[0], b[1]));
// }
void solve()
{
 int n, q;
 cin >> n >> q;
 vector<int> v(n);
 forn(i, n)
 cin >> v[i];
 Treap* t = NULL;
 for (auto x : v)
  t = merge(t, new Treap(x));
 while (q--)
 {
  int type;
  cin >> type;
  int l, r;
  cin >> l >> r;
  l--; r--;
  auto [BeforeL, AfterL] = split(t, l);
  auto [LtoR, AfterR] = split(AfterL, r - l +
1);
  if (type == 1) // range reverse
   LtoR->reverseMe ^= 1;
  else cout << LtoR->sum << endl; // range
sum
  t = merge(BeforeL, merge(LtoR, AfterR));
 }
}
```

———--CONVEX HULL TRICK—------

```
// for recurrence of type: yi = mj*xi + cj
[j<i]
// for max yi, insert (mi,ci), max y = eval()
// for min yi, insert (-mi,-ci), min y = -eval()
const ll is_query = -(1LL<<62);
struct line {
    ll m, b;
    mutable function<const line*()> succ;
    bool operator<(const line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};

struct dynamic_hull : public multiset<line>
{ // will maintain upper hull for maximum
    const ll inf = LLONG_MAX;
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <=
z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m
&& y->b <= x->b;
        /* compare two lines by slope, make
sure denominator is not 0 */
        ll v1 = (x->b - y->b);
        if (y->m == x->m) v1 = x->b > y->b ?
inf : -inf;
        else v1 /= (y->m - x->m);
        ll v2 = (y->b - z->b);
        if (z->m == y->m) v2 = y->b > z->b ?
inf : -inf;
        else v2 /= (z->m - y->m);
        return v1 >= v2;
    }
    void insert_line(ll m, ll b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end()
? 0 : &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() &&
bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y)))
erase(prev(y));
    }
    ll eval(ll x) {
        auto l = *lower_bound((line) { x,
is_query });
        return l.m * x + l.b;
    }
};
```

—---------- 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——————————

```
// 1 base, sack on tree (basically small to
large merging)
const int N=100001;
vector<int>v[N];
int a[N];
int cnt[N];
vector<int>subtree[N];
int sz[N];
map<int,int>f;
int ans[N];
void sz_dfs(int node,int p=0) {
 sz[node] = 1;
 for (auto &c: v[node]) {
  if (c != p) {
   sz_dfs(c, node);
   sz[node] += sz[c];
  }
 }
}
void add(int node){
 f[cnt[a[node]]] -= a[node];
 if (f[cnt[a[node]]] == 0)
  f.erase(cnt[a[node]]);
 cnt[a[node]]++;
 f[cnt[a[node]]] += a[node];
}
void remove(int node){
 f[cnt[a[node]]] -= a[node];
 if (f[cnt[a[node]]] == 0)
  f.erase(cnt[a[node]]);
 cnt[a[node]]--;
 f[cnt[a[node]]] += a[node];
}
int get_ans() {
 return f.rbegin()->second;
}
void sack_dfs(int node,int p,bool keep) {
 int big = 0;
 for (auto &c: v[node]) {
  if (c != p && sz[big] < sz[c])
   big = c;
 }
 for (auto &c: v[node]) {
  if (c != p && c != big) {
   sack_dfs(c, node, false);
  }
 }
 if (big) {
  sack_dfs(big, node, true);
  swap(subtree[big], subtree[node]);
 }
 subtree[node].push_back(node);
 add(node);
 for (auto &c: v[node]) {
  if (c != p && c != big) {
   for (auto &d: subtree[c]) {
    add(d);
    subtree[node].push_back(d);
   }
  }
 }
```

```
 ans[node]=get_ans();
 if (!keep) {
  for (auto &c: subtree[node]) {
   remove(c);
  }
 }
}
void solve() {
 int n;
 cin>>n;
 for(int i=1;i<=n;i++){
  cin>>a[i];
 }
 for(int i=0;i<n-1;i++) {
  int x, y;
  cin >> x >> y;
  v[x].pb(y);
  v[y].pb(x);
 }
 sz_dfs(1);
 sack_dfs(1,0,false);
 for(int i=1;i<=n;i++){
  cout<<ans[i]<<" ";
 }
}
```

—-----SOLUTION AX + BY = C—---------

```
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;
}
```

--CHINESE REMAINDER THEOREM--

```
long long GCD(long long a, long long b) {
return (b == 0) ? a : GCD(b, a % b); }
inline long long LCM(long long a, long long
b) { return a / GCD(a, b) * b; }
inline long long normalize(long long x, long
long mod) { x %= mod; if (x < 0) x += mod;
return x; }
struct GCD_type { long long x, y, d; };
GCD_type ex_GCD(long long a, long long
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}
pii crt(vector<pii> &v) // x = v[0] % v[1]
{
 int n=sz(v);
 pii ans = v[0];
 for(int i=1;i<n;i++)
 {
  auto pom = ex_GCD(ans.second,
v[i].second);
  int x1 = pom.x;
  int 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;
}
```

—-------- GAUSS ELIMINATION —-----–

```
#include<bits/stdc++.h>
using namespace std;
const int N = 105, mod = 1e9 + 7;
int power(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;
}
int Gauss(vector<vector<int>> a,
vector<int> &ans){
  int n = a.size(), m = (int)a[0].size() - 1;
  vector <int> pos(m, -1);
  int free_var = 0;
  const long long MODSQ = (long long)mod
* mod;
  int det = 1, rank = 0;
  for (int col = 0, row = 0; col < m && row <
n; col++) {
 int mx = row;
 for (int k = row; k < n; k++) if (a[k][col] >
a[mx][col]) mx = k;
 if (a[mx][col] == 0) {det = 0; continue;}
 for (int j = col; j <= m; j++) swap(a[mx][j],
a[row][j]);
 if (row != mx) det = det == 0 ? 0 : mod -
det;
 det = 1LL * det * a[row][col] % mod;
 pos[col] = row;
 int inv = power(a[row][col], mod - 2);
 for (int i = 0; i < n && inv; i++){
   if (i != row && a[i][col]) {
  int x = ((long long)a[i][col] * inv) % mod;
  for (int j = col; j <= m && x; j++){
    if (a[row][j]) a[i][j] = (MODSQ + a[i][j] -
((long long)a[row][j] * x)) % mod;
  }
   }
 }
 row++; ++rank;
  }
  ans.assign(m, 0);
  for (int i = 0; i < m; i++){
 if (pos[i] == -1) free_var++;
 else ans[i] = ((long long)a[pos[i]][m] *
power(a[pos[i]][i], mod - 2)) % mod;
  }
  for (int i = 0; i < n; i++) {
 long long val = 0;
 for (int j = 0; j < m; j++) val = (val + ((long
long)ans[j] * a[i][j])) % mod;
 if (val != a[i][m]) return -1; //no solution
  }
  return free_var; //has solution
}
int32_t main() {
  int n, m; cin >> n >> m;
  vector<vector<int>> a(n, vector<int>(m +
1));
  for(int i = 0; i < n; i++) for(int j = 0; j <=
m; j++) cin >> a[i][j];
  vector<int> ans;
  int k = Gauss(a, ans);
  if(k == -1) cout << "no solution\n";
  else {
 for (auto x: ans) cout << x << '\n';
  }
  return 0;
}
```