**Suffix Array**

typedef pair<int, int> ii;

typedef vector<int> vi;

int Log[MAX+1];

int st[MAX][21 + 1];

**void computeLog()**

{

Log[1] = 0;

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

Log[i] = Log[i/2] + 1;

}

char special[] = {'@','$','#', '&', '\*','^', '?','\0'};

class SuffixArray

{

private:

vi RA;

**void countingSort(int k)**

{

int maxi = max(300, n);

vi c(maxi, 0);

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

++c[i+k < n ? RA[i+k] : 0];

for (int i = 0, sum = 0; i < maxi; ++i)

{

int t = c[i];

c[i] = sum;

sum += t;

}

vi tempSA(n);

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

tempSA[c[SA[i]+k < n ? RA[SA[i]+k] : 0]++] = SA[i];

swap(SA, tempSA);

}

**void constructSA()**

{

SA.resize(n);

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

RA.resize(n);

for (int i = 0; i < n; ++i) RA[i] = T[i];

for (int k = 1; k < n; k <<= 1)

{

countingSort(k);

countingSort(0);

vi tempRA(n);

int r = 0;

tempRA[SA[0]] = r;

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

tempRA[SA[i]] = ((RA[SA[i]] == RA[SA[i-1]]) && (RA[SA[i]+k] == RA[SA[i-1]+k])) ? r : ++r;

swap(RA, tempRA);

if (RA[SA[n-1]] == n-1) break;

}

}

**void computeLCP()**

{

vi Phi(n);

vi PLCP(n);

PLCP.resize(n);

Phi[SA[0]] = -1;

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

Phi[SA[i]] = SA[i-1];

// remember prev suffix

// compute PLCP in O(n)

for (int i = 0, L = 0; i < n; ++i)

{

if (Phi[i] == -1)

{

// special case

PLCP[i] = 0;

continue;

}

while ((i+L < n) && (Phi[i]+L < n) && (T[i+L] == T[Phi[i]+L]))

++L; // L incr max n times

PLCP[i] = L;

L = max(L-1, 0); // L dec max n times

}

LCP.resize(n);

for (int i = 0; i < n; ++i) // compute LCP in O(n)

LCP[i] = PLCP[SA[i]]; // restore PLCP

}

**void initRMQ()**

{

// jhamela hoile eitare clear kore neya jaite pare.

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

{

st[i][0] = LCP[i];

}

int K = log2(n) + 1;

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

{

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

{

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

}

}

}

public: // bar bar input nile const ta change kora lagte pare may be

const char\* T; // the input string

int n; // the length of T specail character included

vi SA; // Suffix Array

vi LCP; // of adj sorted suffixes

**void init**(const char\* initialT, int \_n)

{

T = initialT;

n = \_n;

constructSA(); // O(n log n)

computeLCP();

initRMQ(); // O(n)

} // where P is found

int askRMQ(int L, int R)

{

// amar mone hoi L + 1 kore query korte hobe

// jemon 1 theke 3 er modde lcp lagle 2 theke 3 query korte hobe

int j = Log[R - L + 1];

int minimum = min(st[L][j], st[R - (1 << j) + 1][j]);

return minimum;

}

};

const int MAX\_N = 450010; // can go up to 450K chars

char T[MAX\_N];

char P[MAX\_N];

**int main()**

{

// suffix array te prothome 0 index a thakbe special character

// then thakbe jeita choto seita...

// LCP array te pasha pashi duita SA[i] SA[i + 1] er LCP thakbe.

computeLog();

scanf("%s", &T); // read T

int n = (int)strlen(T); // count n

T[n++] = '$';

// add terminating symbol

SuffixArray S;

S.init(T, n); // construct SA+LCP

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

{

printf("%d ", S.SA[i]);

}

printf("\n");

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

{

printf("%d ", S.LCP[i]);

}

printf("\n");

// 1 theke 3 er modde lcp koto?

cout << S.askRMQ(2, 3) << endl;

return 0;

}

**ii stringMatching**(const char \*P) // in O(m log n)

{

int m = (int)strlen(P); // usually, m < n

int lo = 0, hi = n-1; // range = [0..n-1]

while (lo < hi) // find lower bound

{

int mid = (lo+hi) / 2; // this is round down

int res = strncmp(T+SA[mid], P, m); // P in suffix SA[mid]?

(res >= 0) ? hi = mid : lo = mid+1; // notice the >= sign

}

if (strncmp(T+SA[lo], P, m) != 0) return {-1, -1}; // if not found

ii ans;

ans.first = lo;

hi = n-1; // range = [lo..n-1]

while (lo < hi) // now find upper bound

{

int mid = (lo+hi) / 2;

int res = strncmp(T+SA[mid], P, m);

(res > 0) ? hi = mid : lo = mid+1; // notice the > sign

}

if (strncmp(T+SA[hi], P, m) != 0) --hi; // special case

ans.second = hi;

return ans; // returns (lb, ub)

} // where P is found

**ii LRS()** // (LRS length, index)

{

int idx = 0, maxLCP = -1;

for (int i = 1; i < n; ++i) // O(n), start from i = 1

if (LCP[i] > maxLCP)

maxLCP = LCP[i], idx = i;

return {maxLCP, idx};

}

**ii LCS(int split\_idx)** // (LCS length, index)

{

int idx = 0, maxLCP = -1;

for (int i = 1; i < n; ++i) // O(n), start from i = 1

{

// if suffix SA[i] and suffix SA[i-1] came from the same string, skip

if ((SA[i] < split\_idx) == (SA[i-1] < split\_idx)) continue;

if (LCP[i] > maxLCP)

maxLCP = LCP[i], idx = i;

}

return {maxLCP, idx};

}

**int upper\_boundL(int lcp, int pos)**

{

//ei function r kaj hocce uporer dike kotodur gele lcp beshi ba soman

// lcp thake.

// lo 0 kore dilam ar hi = pos korlam. mid obdhi lcp thik thakbe

int lo = 0, hi = pos, res, mid, val = pos; // range = [0..n-1]

while (lo <= hi) // find lower bound

{

mid = (lo+hi) / 2; // this is round down

res = askRMQ(mid, pos); // P in suffix SA[mid]?

if(res >= lcp)

{

val = mid - 1;

hi = mid - 1;

}

else lo = mid + 1; // notice the >= sign

}

return val;

}

**int upper\_boundR(int lcp, int pos)**

{

int lo = pos + 1, hi = n - 1, res, val = pos; // range = [0..n-1]

while (lo <= hi) // find lower bound

{

int mid = (lo+hi) / 2; // this is round down

res = askRMQ(pos + 1, mid); // P in suffix SA[mid]?

if(res >= lcp)

{

val = mid;

lo = mid + 1;

}

else hi = mid - 1; // notice the >= sign

}

return val;

}

**Hashing + Palindromic Tree:**

const int N = 2e5 + 4;

const ll P = 233;

const int A = 26;

const int MOD = 1e9 + 123;

ll h[N], pwr[N], inv[N];

vector<ll> prefix\_sum;

map<ll, ll> M;

int n, q; ll l, r;

char s[200005];

int tree[N][26], idx;

int len[N], link[N], t;

//------------ Hash code Starts

ll bigMod (ll a, ll e) {

if (e == -1) e = MOD - 2;

ll ret = 1;

while (e) {

if (e & 1) ret = ret \* a % MOD;

a = a \* a % MOD, e >>= 1;

}

return ret;

}

void init\_hash(){

pwr[0] = inv[0] = 1;

ll INV\_P = bigMod(P, -1);

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

pwr[i] = pwr[i - 1] \* P % MOD;

inv[i] = inv[i - 1] \* INV\_P % MOD;

}

}

void build\_hash(){

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

h[i] = (h[i - 1] + (s[i] - 'a' + 1) \* pwr[i]) % MOD;

}

}

ll get\_hash(int l, int r){

ll one = (h[r] - h[l - 1]) \* inv[l] % MOD;

if (one < 0) one += MOD;

// h[i] one based index hobe,

return one;

}

// -------------------Hash code ends

struct struct\_pair{

int idx, len;

struct\_pair(){}

struct\_pair(int a, int b){

idx = a; len = b;

}

bool operator<(struct\_pair other)const{

int mx = min(len, other.len); mx--;

int lo = 0, hi = mx, mid;

while(lo <= hi){

mid = (lo + hi) / 2;

if(get\_hash(idx, idx + mid) == get\_hash(other.idx,other.idx + mid))lo = mid + 1;

else hi = mid - 1;

}

if(lo == mx + 1)return len < other.len; // emon case aabaa aabaabaa

return s[idx + lo] < s[other.idx + lo];

}

};

vector<struct\_pair> vsp;

//------------Palindromic Tree Starts

void extend(int p) {

while(s[p - len[t] - 1] != s[p]) t = link[t];

int x = link[t], c = s[p] - 'a';

while(s[p - len[x] - 1] != s[p]) x = link[x];

if(!tree[t][c]) {

tree[t][c] = ++idx;

len[idx] = len[t] + 2;

link[idx] = len[idx] == 1 ? 2 : tree[x][c];

vsp.push\_back(struct\_pair(p - len[idx] + 1, len[idx]));

} t = tree[t][c];

}

//------------Palindromic Tree Ends.

int main(){

scanf("%d", &n);

scanf("%s", s + 1);

len[1] = -1; link[1] = 1;

len[2] = 0; link[2] = 1;

idx = t = 2;

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

init\_hash();

build\_hash();

}

**Prefix FUnction**:

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;

}

**BIT**:

struct FT

{

int n;

FT(int \_n)

{

n = \_n;

} void update(int x, ll delta)

{

for(; x <= n; x += x&-x) BIT[x] += delta;

} ll query(int x, ll sum = 0LL)

{

for(; x > 0; x -= x&-x) sum += BIT[x];

return sum;

}

};

**MANACHAR:**

struct Manachar{

int n;

vector<int> P;

Manachar(int \_n) {

n = \_n;

P.resize(2 \* n + 10, 0);

} string convert(string s){

string nwS = "@";

for (int i = 0; i < (int)s.size(); i++)

{

nwS += "#" + s.substr(i, 1);

}

nwS += "#$";

return nwS;

} void build(string s)

{

int \_n = s.size();

int l = 0, r = -1;

for(int i = 1; i < (int)s.size() - 1; i++)

{

int k;

if(i > r)

{

k = 0;

}

else

{

int j = r - i + l;

k = min (P[j], r - i);

}

while(i + k < \_n and i - k >= 0 and s[i+k] == s[i-k])k++;

if(s[i-k] != s[i+k])k--;

P[i] = k;

if(i + k > r)

{

r = i + k;

l = i - k;

}

}

} ll calculate(string s, string original)

{

int maxpal = 0;

int idx = 0;

for(int i = 1; i < s.size() - 1; i++)

{

cout << i <<" "<< s[i]<<" "<< P[i] << endl;

if(P[i] > maxpal)

{

maxpal = P[i];

idx = i - P[i] ;

idx /= 2;

}

}

cout<<maxpal<<endl;

cout<<idx<<endl;

cout<<original.substr(idx,maxpal)<<endl;

} void solve(string s)

{

string original = s;

s = convert(s);

build(s);

calculate(s, original);

}

};

int main()

{

int testCase;

scanf("%d", &testCase);

while (testCase--)

{

scanf("%s", ch);

string s = ch;

int n = s.size();

Manachar mc(n);

mc.solve(s);

}

}

**TRIE:**

struct node

{

bool endmark;

node\* next[26 + 1];

node()

{

endmark = false;

for (int i = 0; i < 26; i++)next[i] = NULL;

}

} \* root;

struct Trie

{

string s;

Trie(string \_s)

{

s = \_s;

} void insert(string str, int len)

{

node\* curr = root;

for (int i = 0; i < len; i++)

{

int id = str[i] - 'a';

if (curr->next[id] == NULL)curr->next[id] = new node();

curr = curr->next[id];

}

curr->endmark = true;

} bool search(string str, int len)

{

node\* curr = root;

for (int i = 0; i < len; i++)

{

int id = str[i] - 'a';

if (curr->next[id] == NULL)return false;

curr = curr->next[id];

}

return curr->endmark;

} void del(node\* cur)

{

for (int i = 0; i < 26; i++)if (cur->next[i])del(cur->next[i]);

delete (cur);

} void solve()

{

root = new node();

insert(s, s.size());

cout << search("abcd", s.size()) << endl;

del(root);

}

};

**SACK:**

int cnt[MAX];

int col[MAX];

int sz[MAX];

bool big[MAX];

vector<int> g[MAX];

struct SACK

{

void init(int n)

{

mem(big, false);

mem(cnt, 0);

} void getsz(int v, int p)

{

sz[v] = 1;

for(auto u : g[v])

{

if(u != p)

{

getsz(u, v);

sz[v] += sz[u];

}

}

} void add(int v, int p, int x)

{

cnt[col[v]] += x;

for(auto u: g[v])

{

if(u != p && !big[u])

{

add(u, v, x);

}

}

} void dfs(int v, int p, bool keep)

{

int mx = -1, bigChild = -1;

for(auto u : g[v])

{

if(u != p && sz[u] > mx)

{

mx = sz[u], bigChild = u;

}

}

for(auto u : g[v])

{

if(u != p && u != bigChild)

{

dfs(u, v, 0);

}

}

if(bigChild != -1)

{

dfs(bigChild, v, 1), big[bigChild] = 1;

}

add(v, p, 1);//now cnt[c] is the number of vertices in subtree of vertex v that has color c. You can answer the queries easily.

if(bigChild != -1)

{

big[bigChild] = 0;

}

if(keep == 0)

{

add(v, p, -1);

}

} void solve()

{

getsz(0, -1);

}

};

int main()

{

int n;

scanf("%d", &n);

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

{

int a, b;

scanf("%d %d", &a, &b);

a--;

b--;

g[a].pb(b);

g[b].pb(a);

}

SACK sack;

sack.solve();

}

**MO’s:**

// dorkar hoile k = sqrt((n \* n)/ q) kora jaite pare.

const int blocksize = 350;

struct query

{

int l, r, idx;

query(int a, int b, int c)

{

l = a;

r = b;

idx = c;

} bool operator <(query other) const

{

int l1 = l / blocksize;

int l2 = other.l / blocksize;

if(l1 != l2) return l < other.l;

return l1 % 2? r > other.r : r < other.r;

}

};

struct MO

{

vector<query> queries;

vector<int> ans;

int n, l, r;

MO(int \_n)

{

n = \_n;

l = 0;

r = -1;

queries.clear();

ans.resize(n, 0);

} void mo()

{

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

{

while(l > queries[i].l)

{

cout << l << " 1"<< endl;

add(--l);

}

while(r < queries[i].r)

{

cout << r << " 2"<< endl;

add(++r);

}

while(l < queries[i].l)

{

cout << l << " 3" << endl;

remove(l++);

}

while(r > queries[i].r)

{

cout << r <<" 4"<< endl;

remove(r--);

}

ans[queries[i].id] = sum;

}

} void add(int x) sum += a[x];

} void remove(int x)

{

sum -= a[x];

}

void solve()

{

int a, b;

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

{

scanf("%d %d", &a, &b);

queries.pb(query(a,b,i));

}

sort(queries.begin(), queries.end());

mo();

for (auto it: queries)cout << it.l<<" " << it.r <<" " << it.idx << endl;

}

};

int main ()

{

int n;

cin >> n;

MO mos(n);

mos.solve();

}

{

**MO’s Update**:

MO’s Update:

int l, r, t, blocksize;

struct query

{

int l, r, t, id;

bool operator <(query other) const

{

int l1, r1, l2, r2;

l1 = l / blocksize, l2 = other.l / blocksize,r1 = r / blocksize, r2 = other.r / blocksize;

if(l1 != l2) return l1 < l2;

if(r1 != r2) return r1 < r2;

//return l1 % 2? r > other.r : r < other.r;

return t < other.t;

// return l1 % 2 ? t > other.t: t < other.t;

}

} Q[MAX];

struct update

{

int x, pre, now;

} U[MAX];

struct MO\_UP

{

vector<int> input, Copy;

vector<int> last, ans;

int n, q, distinct;

MO\_UP(int \_n, int \_q)

{

l = 0, r = -1, t = 0;

n = \_n;

q = \_q;

distinct = 0;

blocksize = (int)pow(n, 2.0/ 3.0);

input.clear();

Copy.clear();

ans.clear();

last.clear();

last.resize(n);

} void mo\_up()

{

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

{

while(t < Q[i].t)

{

t++, apply(U[t].x, U[t].now);

} // Forward Update

while(t > Q[i].t)

{

apply(U[t].x, U[t].pre), t--; // Reverse Update

}

while(l > Q[i].l)

{

add(--l);

}

while(r < Q[i].r)

{

add(++r);

}

while(l < Q[i].l)

{

remove(l);

l++;

}

while(r > Q[i].r)

{

remove(r);

r--;

}

ans[Q[i].id] = distinct;

}

} void apply(int x, int y)

{

if(l <= x && x <= r) // l, r is the l, r from MO's algo

remove(x);

input[x] = y;

add(x);

}

else input[x] = y;

} void add(int x)

{

a = input[x];

occ[a]++;

sum += (1LL \* a \* (angry[occ[a]]));//cout << sum << endl;

}

void remove(int x)

{

a = input[x];

sum -= (1LL \* a \* (angry[occ[a]]));

//cout << sum <<" b"<< endl;

occ[a]--;

}

void Resize()

{

Copy = input;

sort(Copy.begin(), Copy.end());

Copy.resize(unique(Copy.begin(), Copy.end())- Copy.begin());

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

{

int idx = lower\_bound(Copy.begin(), Copy.end(), input[i]) - Copy.begin();

input[i] = idx;

}

}

void solve()

{

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

{

scanf("%d", &input[i]);

last[i] = input[i];

}

scanf("%d", &q);

int type, left, right, x, time = 0, idx = 0, y;

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

{

scanf("%d", &type);

if (type == 1)

{

scanf("%d %d", &left, &right);

left --;

right --; // 0 based

idx++;

Q[idx - 1].l = left;

Q[idx - 1].r = right;

Q[idx - 1].t = time;

Q[idx - 1].id = idx - 1;

}

else

{

time++;

scanf("%d %d", &x, &y);

x--; //0 based

U[time].x = x;

U[time].pre = last[x];

U[time].now = y;

last[x] = y;

}

}

q = idx;

ans.resize(q);

mo\_up();

}

};

int main()

{

int n, m;

scanf("%d %d", &n, &m);

MO\_UP mos(n, m);

mos.solve();

return 0;

}

{

**SCC**:

vector<int> adj[MAX], adj\_rev[MAX];

vector<bool> used;

vector<int> order, component;

vector<int> roots;

vector<int> root\_nodes;

vector<vector<int>> adj\_scc(MAX); // Eikhane Max er jaigai n hobe

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() {

int n, m;

// ... read n ...

cin >> n >> m;

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

int a, b; cin >> a >> b;

// ... read next directed edge (a,b) ...

adj[a].push\_back(b);

adj\_rev[b].push\_back(a);

}

used.assign(n, false);

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

if (!used[i])

dfs1(i);

used.assign(n, false);

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

roots.assign(n, 0);

for (auto v : order){

if (!used[v]) {

dfs2 (v);

int root = component.front();

for (auto u : component) roots[u] = root;

root\_nodes.push\_back(root);

// ... processing next component ...

for(auto com: component)cout << com <<" "; cout << endl;

component.clear();

}

}

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

for (auto u : adj[v]) {

int root\_v = roots[v], root\_u = roots[u];

if (root\_u != root\_v)

adj\_scc[root\_v].push\_back(root\_u);

}

}

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

for(auto u: adj\_scc[v]){

cout << v <<" "<< u << endl;

}

}

}

**SPF + Seive:**

vector<ll>prime;//bool mark[MAX1];

ll N=1e7;

ll spf[10000004];

void seive ()

{

for(int i = 1; i <= N; i++)spf[i] = i;

int limit = sqrt(N\*1.0)+2;

spf[1] = 1;

spf[2] = 2;

for (int i=4; i<=N; i += 2)spf[i]=2;

prime.push\_back(2);//i\*i <=N kora jave

for (int i=3; i<=N; i+=2)

{

if(spf[i]==i)

{

prime.push\_back(i);

if(i<=limit)

{

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

{

if(spf[j]==j)spf[j] = i;

}

}

}

}

}

while (x != 1)

{

ret.push\_back(spf[x]);

x = x / spf[x];

}

return ret

**1 to N total bit set per position:**

ll Find\_Xor(ll n){ll power = 2LL;ll sum = 0LL;if((n / 2LL) % 2LL) sum += 1LL;ll val = 0LL;while (power <= n) {ll pair\_ase = n / power;val = 1LL \* (pair\_ase / 2LL) \* power;val += (pair\_ase & 1LL) ? (n % power) : 0LL;if(val % 2) sum += power;power <<= 1LL;}return sum;}

**RNG + PBDS**:

mt19937\_64 rng(chrono::steady\_clock::now().time\_since\_epoch().count());

template<class T> using oset=tree<T, null\_type, less\_equal<T>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;

// order\_of\_key returns the 0 based position of a value

// find\_by\_order returns a pointer pointing at the kth element // also 0 indexed

**Phi Div:**

void phi\_1\_to\_n(int n) {vector<int> phi(n + 1);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];}

#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* *int*

#define pb push\_back

#define mp make\_pair

#define all(*x*) x.begin(),x.end()

#define Max 10000000000000000

#define min\_heap priority\_queue <ll, vector<ll>, greater<ll> >

*template* <*typename* T>

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

**Number Theory:**

#Sum of devisors SOD(n) = V((pf n+1 -1)/(pf-1))  
#Sum of devisors SOD(n m ) = V((pf n\*m+1 -1)/(pf-1))  
#Number of devisor NOD(n)=V(a[i]+1) Where n=pf a[i]

#In a circle of n seats one can fill the circles in ways ∑(K GCD(i,n) /n) where has K different types of people

#Extended GCD

If GCD(a,b)=g then there has two value x,y wherw ax+by=g

ll egcd(ll *a*,ll *b*,ll &*x*,ll &*y*){

if(*a*==0){

*x*=0,*y*=1;

return *b*;

}

ll x1,y1,d = egcd(*b*%*a*,*a*,x1,y1);

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

return d;

}

#Finding all totient function values in a range (1, n)

for(ll i=0;i<range;i++) phi[i]=i;

for(ll i=2;i<range;i++){

if(!mark[i]){

for(ll j=i;j<range;j+=i){

mark[j]=1;

phi[j]-=(phi[j]/i);

}

}

}

#Sum of all co-primes of N is phi(N)\*N/2

#Totient function phi(N)=N\*V(1-1/pf)

#Prime factorization in log(n)

Sieve where ll sc[j]=highest prime k as j mod k=0

pf.pb(mp(sc[n],1));

n/=sc[n];

while(n!=1){

if(pf[pf.size()-1].first==sc[n]){

pf[pf.size()-1].second++;

n/=sc[n];

}

else{

pf.pb(mp(sc[n],1));

n/=sc[n];

}

}

#SUM of fibonacci numbers = fib(n+2)-1

#In an equation,the max output for the value of unknown variable can be determined by defferentiation.

#Combinatorics

1D: nCr= n-1Cr+n/(n-r)

2D: nCr= n-1Cr-1+ n-1Cr

#modinverse(nm )=modinverse(bigmod(n,m))

#The number of different k sets made by n elements (Stirling number)

1: k\* n-1Ck+ n-1Ck-1

2: (n-1)\* n-1Ck+ n-1Ck-1

#de-arrangement(N)= (N-1){de-arrangement(N-1) + de-arrangement(N-2)}

#The number of pairs can be formed by n and a smaller number than n with gcd g is phi[n/g]

**Dynamic Programming**

**#Digit DP**

ll dp[11][2][100];

string x;

ll solve(ll *pos*,ll *chk*,ll *sum*){

if(*sum*<0) return 0;

if(!*sum*) return 1;

if(*pos*==x.size()) return 0;

if(dp[*pos*][*chk*][*sum*]!=-1) return 0;

dp[*pos*][*chk*][*sum*]=0;

for(*char* i='0';i<='9';i++){

if(!*chk* && i>x[*pos*]) break;

ll new\_chk=*chk*?1:(i!=x[*pos*]);

dp[*pos*][*chk*][*sum*]+=solve(*pos*+1,new\_chk,*sum*-(ll)(i-'0'));

}

return dp[*pos*][*chk*][*sum*];

}

**#LIS O(nlogn)**

vector<ll> v;

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

ll id=lower\_bound(all(v),a[i])-v.begin();

if(id<v.size()) v[id]=a[i];

else v.pb(a[i]);

}

#Longest Palindromic subsequence

for(ll i=1;i<=s.size();i++){

for(ll j=1,k=i;k<=s.size();j++,k++){

if(j==k) dp[j][k]=1;

else{

if(s[j-1]==s[k-1]){

if(i==2) dp[j][k]=2;

else dp[j][k]=dp[j+1][k-1]+2;

}

else dp[j][k]=max(dp[j+1][k],dp[j][k-1]);

}

}

}

**#MCM**

ll a[107];

ll dp[107][107],path[107][107];

ll solve(ll *i*,ll *j*){

if(*i*==*j*) return 0;

if(dp[*i*][*j*]!=Max) return dp[*i*][*j*];

for(ll k=*i*;k<*j*;k++){

ll x=solve(*i*,k)+solve(k+1,*j*)+(a[*i*]\*a[k+1]\*a[*j*+1]);

if(x<dp[*i*][*j*]){

dp[*i*][*j*]=x;

path[*i*][*j*]=k;

}

}

return dp[*i*][*j*];

}

#define fast ios\_base::sync\_with\_stdio(false);cin.tie(0);cout.tie(0);

ll dx[]= {1,-1,0,0,1,-1,-1,1};

ll dy[]= {0,0,1,-1,1,1,-1,-1};

ll knx[]= {2,2,1,-1,-2,-2,1,-1};

ll kny[]= {1,-1,2,2,1,-1,-2,-2};

priority\_queue< pair<ll,ll>, vector<pair<ll,ll>>,greater<pair<ll,ll> > >pq;

//floyed warshal

if (floyed\_weight[i][j] > (floyed\_weight[i][k] + floyed\_weight[k][j])

&& (floyed\_weight[k][j] != INF

&& floyed\_weight[i][k] != INF)) {

floyed\_weight[i][j] = floyed\_weight[i][k] + floyed\_weight[k][j];

}

/// **Articulation Point**

vector<ll>edj[10010];

ll start[10010];

ll en[10010],cnt;

bitset<10010>color,ap;

void dfs\_ap(ll node,ll parent){

ll child=0; start[node]=cnt; en[node]=cnt++;

ll siz=edj[node].size();

color[node]=true;

for(ll i=0; i<siz; i++) {

ll tm=edj[node][i];

if(color[tm]==false) {

child++; color[tm]=true;

dfs\_ap(tm,node);

en[node]=min(en[tm],en[node]);

if (parent == -1 && child > 1) {

ap[node] = true;

}

if(parent != -1&& start[node]<=en[tm]) {

ap[node]=true; }

}

else if(tm!=parent){

en[node]=min(start[tm],en[node]);

}

}

}

/// **bridge**

vector<ll>edj[100010];vector<pair<ll,ll>>V;

bitset<100010>color;

ll start[100010],en[100010];ll cnt;

void bridge(ll node,ll parent){

color[node]=true;

start[node]=cnt;

en[node]=cnt++;

for(ll i:edj[node]) {

if(color[i]==false) {

color[i]=true;bridge(i,node);

start[node]=min(start[node],start[i]);

if(en[node]<start[i]) {

ll a=node,b=i;

if(a>b)swap(a,b);

V.push\_back({a,b});

}

}

else {

if(parent!=i)

start[node]=min(start[node],en[i]);

} }

}

**/\* Black Box DMST\*/**

struct Edge

{

int x, y, w;

};

int dmst(int N, vector<Edge> E, int root)

{

const int oo = 1e9;

vector<int> cost(N), Back(N), label(N), bio(N);

int ret = 0;

for (;;) {

for(int i=0; i<N; i++) cost[i] = oo;

for (auto e : E) {

if (e.x == e.y) continue;

if (e.w < cost[e.y]) cost[e.y] = e.w, Back[e.y] = e.x;

}

cost[root] = 0;

for(int i=0; i<N; i++) if (cost[i] == oo) return -1;

for(int i=0; i<N; i++) ret += cost[i];

int K = 0;

for(int i=0; i<N; i++) label[i] = -1;

for(int i=0; i<N; i++) bio[i] = -1;

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

int x = i;

for (; x != root && bio[x] == -1; x = Back[x]) bio[x] = i;

if (x != root && bio[x] == i) {

for (; label[x] == -1; x = Back[x]) label[x] = K;

++K;

}

}

if (K == 0) break;

for(int i=0; i<N; i++) if (label[i] == -1) label[i] = K++;

for (auto &e : E) {

int xx = label[e.x];

int yy = label[e.y];

if (xx != yy) e.w -= cost[e.y];

e.x = xx;

e.y = yy;

}

root = label[root];

N = K;

}

return ret;

}

/// Merge Sort Tree

merge(seg[left].begin(), seg[left].end(), seg[right].begin(), seg[right].end(),back\_inserter(seg[n]));

///**LCA**

#define N 200100

vector<ll>edj[N];

bool color[N];ll label[N];ll sparse[N][20];ll parent[N];

void dfs(ll node){

color[node]=false;

ll siz=edj[node].size();

for(ll i=0; i<siz; i++) {

ll tm=edj[node][i];

if(color[tm]==true) {

color[tm]=false; label[tm]=label[node]+1;

parent[tm]=node;

sparse[tm][0]=node; dfs(tm);

} }

}

void pre\_process(ll n){

for(ll j=1; (1<<j)<=n; j++) {

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

sparse[i][j]=sparse[sparse[i][j-1]][j-1];

}

}

}

ll query(ll u,ll v){

if(label[u]<label[v])swap(u,v);

ll lg=0;

for(lg=0; (1<<lg)<=label[u]; lg++);

lg--;

if(label[u]!=label[v]) {

for(ll i=lg; i>=0; i--) {

if(label[u]-(1<<i)>=label[v]) {

u=sparse[u][i];

}

}

}

if(u==v) { return u; }

for(ll i=lg; i>=0; i--) {

if(sparse[u][i]!=-1&&sparse[u][i]!=sparse[v][i]) {

u=sparse[u][i];

v=sparse[v][i];

}

}

return parent[u]; }

ll nth\_parent(ll u,ll nth){

ll lg=0;

for(lg=0; (1<<lg)<=label[u]; lg++);

lg--;

ll nth\_parent\_label=nth;

if(label[u]!=nth\_parent\_label) {

for(ll i=lg; i>=0; i--) {

if(label[u]-(1<<i)>=nth\_parent\_label) {

u=sparse[u][i];

}

}

}

return u;

}

ll lc=query(u,v);

ll ans=(dis[u]-dis[lc])+(dis[v]-dis[lc]);

/// Trie

int trie[100005][53];int idx;

int value(char ch){

if(isupper(ch)) return ch-'A';

return ch-'a'+26;

}

void insert(string str,ll len){

int cur=0,i;

for(i=0; i<len; i++) {

int j=value(str[i]);

if(trie[cur][j]==-1) {

trie[cur][j]=++idx;

memset(trie[idx],-1,sizeof(trie[idx]));

trie[idx][52]=0;

}

cur= trie[cur][j];

}

trie[cur][52]+=1;

}

int search(string str,ll len){

int cur=0,i;

for(i=0; i<len; i++) {

int j=value(str[i]);

if(trie[cur][j]==-1) {

return 0; }

cur= trie[cur][j];

}

return trie[cur][52];

}

///**Edmonds-Karp algorithm**

int n;

vector<vector<int>> capacity;

vector<vector<int>> adj;

int bfs(int s, int t, vector<int>& parent) {

fill(parent.begin(), parent.end(), -1);

parent[s] = -2;

queue<pair<int, int>> q;

q.push({s, INF});

while (!q.empty()) {

int cur = q.front().first;

int flow = q.front().second;

q.pop();

for (int next : adj[cur]) {

if (parent[next] == -1 && capacity[cur][next]) {

parent[next] = cur;

int new\_flow = min(flow, capacity[cur][next]);

if (next == t)

return new\_flow;

q.push({next, new\_flow});

}

}

}

return 0;

}

int maxflow(int s, int t) {

int flow = 0;

vector<int> parent(n); int new\_flow;

while (new\_flow = bfs(s, t, parent)) {

flow += new\_flow;int cur = t;

while (cur != s) {

int prev = parent[cur];

capacity[prev][cur] -= new\_flow;

capacity[cur][prev] += new\_flow;

cur = prev; } }

return flow;

}

// C++ program to find

**// maximum XOR subset**

#include<bits/stdc++.h>

using namespace std;

// Number of bits to

// represent int

#define INT\_BITS 32

// Function to return

// maximum XOR subset

// in set[]

int maxSubarrayXOR(int set[], int n)

{

// Initialize index of

// chosen elements

int index = 0;

// Traverse through all

// bits of integer

// starting from the most

// significant bit (MSB)

for (int i = INT\_BITS-1; i >= 0; i--)

{

// Initialize index of

// maximum element and

// the maximum element

int maxInd = index;

int maxEle = INT\_MIN;

for (int j = index; j < n; j++)

{

// If i'th bit of set[j]

// is set and set[j] is

// greater than max so far.

if ( (set[j] & (1 << i)) != 0

&& set[j] > maxEle )

maxEle = set[j], maxInd = j;

}

// If there was no

// element with i'th

// bit set, move to

// smaller i

if (maxEle == INT\_MIN)

continue;

// Put maximum element

// with i'th bit set

// at index 'index'

swap(set[index], set[maxInd]);

// Update maxInd and

// increment index

maxInd = index;

// Do XOR of set[maxIndex]

// with all numbers having

// i'th bit as set.

for (int j=0; j<n; j++)

{

// XOR set[maxInd] those

// numbers which have the

// i'th bit set

if (j != maxInd &&

(set[j] & (1 << i)) != 0)

set[j] = set[j] ^ set[maxInd];

}

// Increment index of

// chosen elements

index++;

}

// Final result is

// XOR of all elements

int res = 0;

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

res ^= set[i];

return res;

}

// Driver program

int main()

{

int set[] = {9, 8, 5};

int n = sizeof(set) / sizeof(set[0]);

cout << "Max subset XOR is ";

cout << maxSubarrayXOR(set, n);

return 0;

}

-------------------Faizul --------------

**Dijkstra**

struct node{

int At,Cost;

node(int a,int b){

At=a; Cost=b;

}

bool operator<(node other) const{

return Cost>other.Cost;

}

};

**Prim’s**

while(!PQ.empty()) {

int U1=PQ.top().U; int C1=PQ.top().Cost; PQ.pop();

if(Mark[U1]) continue;

Mark[U1]=true; ans+=C1;

for(int i=0;i<G[U1].size();i++) {

pair<int,int> V=G[U1][i];

if(Mark[V.first]) continue;

if(V.second<Dist[V.first]) {

PQ.push(edge(V.first,V.second));

Dist[V.first]=V.second;

}

}

}

**Linear Sieve**

#define MAX 100000001

vector<int> Primes;

int least[MAX];

void linearsieve(){

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

if(!least[i]){

least[i]=i;

Primes.pb(i);

}

for(int x:Primes){

if(x>least[i] | i\*x>=MAX) break;

least[i\*x]=x;

}

}

}

**Phi(divisor property)**

int phi[maxn];

void totient(){

phi[1]=1;

for(int i=2;i<maxn;i++) phi[i]=i-1;

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

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

phi[j]-=phi[i];

}

}

}

**Mulmod(1e15)**

ll mulmod(ll a,ll b,ll m){

ll res=0;

while(a>0){

if(a&1){

res+=b;

if(res>=m) res-=m;

}

a>>=1;

b<<=1;

if(b>=m) b-=m;

}

return res;

}

**Pbds**

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

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

using namespace \_\_gnu\_pbds;

template<class T> using oset=tree<T, null\_type, less<T>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;

// order\_of\_key returns the 0 based position of a value

// find\_by\_order returns a pointer pointing at the kth element

**Aho Corasick(Improved fail)**

char text[1000005],pat[505];

int fail[maxn],trans[maxn],res[maxn],query[505];

// trans - ekta full pattern match korche etake point kore.

// res - cur node e ze pattern end hoiche seta koto bar occur korche

// query - ze pattern ze node e end hoiche tar number ta store kore rakhbo

struct node{

int child[26],endpoint;

node(){

endpoint=0;

for(int i=0;i<26;i++){

child[i]=-1;

}

}

} root;

vector<node> tri;

int idx;

void init(){

tri.clear();

for(int i=0;i<=500;i++){

query[i]=0;

}

for(int i=0;i<=1e6;i++){

fail[i]=trans[i]=res[i]=0;

}

tri.pb(root);

idx=0;

}

void Add(char \*s,int len,int cur){

int now=0;

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

int id=s[i]-'a';

if(tri[now].child[id]==-1){

tri[now].child[id]=++idx;

node next=node();

tri.pb(next);

}

now=tri[now].child[id];

}

tri[now].endpoint++;

query[cur]=now;

}

void bfs(){

queue<int> q;

int now=0;

for(int i=0;i<26;i++){

int cur=tri[now].child[i];

if(cur!=-1){

q.push(cur);

if(tri[cur].endpoint) trans[cur]=cur;

}

else{

tri[now].child[i]=0;

}

}

while(!q.empty()){

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

for(int i=0;i<26;i++){

int v=tri[u].child[i];

if(v!=-1){

q.push(v);

fail[v]=tri[fail[u]].child[i];

if(fail[v]==-1) fail[v]=0;

if(tri[v].endpoint) trans[v]=v;

else trans[v]=trans[fail[v]];

}

else{

tri[u].child[i]=tri[fail[u]].child[i];

}

}

}

}

void match(){

int now=0,len=strlen(text);

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

int id=text[i]-'a';

now=tri[now].child[id];

if(now==-1) now=0;

int temp=trans[now];

while(temp>0){

res[temp]++;

temp=trans[fail[temp]];

}

}

}

**KMP(optimized)**

int fail[maxn][27];

struct KMP{

string text,pat;

int n,m;

KMP(string t,string p){

text=t; pat=p;

n=t.length(); m=p.length();

}

vector<int> prefix\_function(){

vector<int> lps(m);

lps[0]=0;

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

int j=lps[i-1];

while(j>0 && pat[i]!=pat[j])

j = lps[j-1];

if (pat[i]==pat[j]) ++j;

lps[i]=j;

}

return lps;

}

void lps\_process(){

vector<int> lps=prefix\_function();

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

int k=lps[i-1];

for(int j=0;j<26;j++){

if(pat[i]-'a'==j){

fail[i][j]=i;

}

else{

fail[i][j]=fail[lps[i-1]][j];

}

}

}

}

vector<int> kmp(){

vector<int> lps=prefix\_function();

lps\_process();

vector<int> idx;

int j=0;

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

j=fail[j][text[i]-'a'];

if(text[i]==pat[j]) j++;

if(j==m){

idx.pb(i-m+1);

j=lps[j-1];

}

}

return idx;

}

};

**Z**

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

int n=s.length();

vector<int> z(n);

for (int i=1,l=0,r=0;i<n;i++){

if(i<=r){

z[i]=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;

}

**Hashing**

ll p[maxn],Hash[maxn],rHash[maxn],base=10;

void build(string s,int n){//1based

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

p[i]=Hash[i]=rHash[i]=0;

}

p[0]=1;

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

p[i]=(p[i-1]\*base)%MOD;

}

string t=s;

reverse(t.begin(),t.end());

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

Hash[i]=(Hash[i-1]\*base+(ll)(s[i-1]-'a'+1))%MOD; //hash

rHash[i]=(rHash[i-1]\*base+(ll)(t[i-1]-'a'+1))%MOD; ///reverse hash

}

}

ll gethash(int a,int b){//1based

return (Hash[b] - (Hash[a - 1] \* p[b - a + 1]) % MOD + MOD) % MOD;

}

ll getrhash(int a,int b,int n){

int x=n-b+1; int y=n-a+1;

return (rHash[y] - (rHash[x - 1] \* p[y - x + 1]) % MOD + MOD) % MOD;

}

**Modular Inverse O(1)**

ll inv[maxn],fact[maxn];

void pre(){

fact[0]=inv[0]=1;

for(ll i=1;i<maxn;i++){

fact[i]=(fact[i-1]\*i)%MOD;

}

inv[maxn-1]=bigmod(fact[maxn-1],MOD-2,MOD);

for(ll i=maxn-2;i>=1;i--){

inv[i]=(inv[i+1]\*(i+1))%MOD;

}

}

ll ncr(ll n,ll r){

ll x=fact[n];

ll y=(inv[r]\*inv[n-r])%MOD;

return (x\*y)%MOD;

}

**Binary Search in pairs**

struct compare{

bool operator()(const pair<int, int>& value,const int& key){

return (value.first < key);

}

bool operator()(const int& key,const pair<int, int>& value){

return (key < value.first);

}

};

binary\_search(v.begin(), v.end(),3, compare());

**Double value Compare**

bool LessThan(double a,double b){

return (b-a)>((fabs(a)<fabs(b)?fabs(b):fabs(a))\*EPS);

}

bool GreaterThan(double a,double b)

{

return (a-b)>((fabs(a)<fabs(b)?fabs(b):fabs(a))\*EPS);

}

bool Equal(double a,double b){

return fabs(a-b)<EPS;

}

**Random value**

mt19937\_64 rng(chrono::steady\_clock::now().time\_since\_epoch().count());

ll aa=rng();

-------------tanbir-------------

**Martix Expo -->**

#include<bits/stdc++.h>

using namespace std;

const int mod = 998244353;

struct Mat {

int n, m;

vector<vector<int>> a;

Mat() { }

Mat(int \_n, int \_m){

n = \_n;

m = \_m;

a.assign(n, vector<int>(m, 0));

}

Mat(vector< vector<int> > v){

n = v.size();

m = n ? v[0].size() : 0;

a = v;

}

inline void make\_unit() {

assert(n == m);

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

for (int j = 0; j < n; j++) a[i][j] = i == j;

}

}

inline Mat operator + (const Mat &b) {

assert(n == b.n && m == b.m);

Mat ans = Mat(n, m);

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

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

ans.a[i][j] = (a[i][j] + b.a[i][j]) % mod;

}

}

return ans;

}

inline Mat operator - (const Mat &b) {

assert(n == b.n && m == b.m);

Mat ans = Mat(n, m);

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

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

ans.a[i][j] = (a[i][j] - b.a[i][j] + mod) % mod;

}

}

return ans;

}

inline Mat operator \* (const Mat &b) {

assert(m == b.n);

Mat ans = Mat(n, b.m);

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

for(int j = 0; j < b.m; j++) {

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

ans.a[i][j] = (ans.a[i][j] + 1LL \* a[i][k] \* b.a[k][j] % mod) % mod;

}

}

}

return ans;

}

inline Mat pow(long long k) {

assert(n == m);

Mat ans(n, n), t = a;

ans.make\_unit();

while (k) {

if (k & 1) ans = ans \* t;

t = t \* t;

k >>= 1;

}

return ans;

}

inline Mat& operator += (const Mat& b) { return \*this = (\*this) + b; }

inline Mat& operator -= (const Mat& b) { return \*this = (\*this) - b; }

inline Mat& operator \*= (const Mat& b) { return \*this = (\*this) \* b; }

inline bool operator == (const Mat& b) { return a == b.a; }

inline bool operator != (const Mat& b) { return a != b.a; }

};

int32\_t main() {

ios\_base::sync\_with\_stdio(0);

cin.tie(0);

int n, m, k;

cin >> n >> m >> k;

Mat a(n, m);

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

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

cin >> a.a[i][j];

}

}

Mat b(m, k);

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

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

cin >> b.a[i][j];

}

}

Mat ans = a \* b;

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

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

cout << ans.a[i][j] << ' ';

}

cout << '\n';

}

return 0;

}

**Goussian Elemination**

#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;

}

----------------------------------

#include<bits/stdc++.h>

using namespace std;

const int N = 3e5 + 9;

const double eps = 1e-9;

int Gauss(vector<vector<double>> a, vector<double> &ans) {

int n = (int)a.size(), m = (int)a[0].size() - 1;

vector<int> pos(m, -1);

double det = 1; int rank = 0;

for(int col = 0, row = 0; col < m && row < n; ++col) {

int mx = row;

for(int i = row; i < n; i++) if(fabs(a[i][col]) > fabs(a[mx][col])) mx = i;

if(fabs(a[mx][col]) < eps) {det = 0; continue;}

for(int i = col; i <= m; i++) swap(a[row][i], a[mx][i]);

if (row != mx) det = -det;

det \*= a[row][col];

pos[col] = row;

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

if(i != row && fabs(a[i][col]) > eps) {

double c = a[i][col] / a[row][col];

for(int j = col; j <= m; j++) a[i][j] -= a[row][j] \* c;

}

}

++row; ++rank;

}

ans.assign(m, 0);

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

if(pos[i] != -1) ans[i] = a[pos[i]][m] / a[pos[i]][i];

}

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

double sum = 0;

for(int j = 0; j < m; j++) sum += ans[j] \* a[i][j];

if(fabs(sum - a[i][m]) > eps) return -1; //no solution

}

for(int i = 0; i < m; i++) if(pos[i] == -1) return 2; //infinte solutions

return 1; //unique solution

}

int main() {

int n, m; cin >> n >> m;

vector< vector<double> > v(n);

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

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

double x; cin >> x; v[i].push\_back(x);

}

}

vector<double> ans;

int k = Gauss(v, ans);

if(k) for(int i = 0; i < n; i++) cout << fixed << setprecision(5) << ans[i] << ' ';

else cout << "no solution\n";

return 0;

}

**Convex Hull-->Graham's Scan algorithm O(NlogN)**

#include<iostream>

#include<stack>

#include<algorithm>

#include<vector>

using namespace std;

struct point { //define points for 2d plane

int x, y;

};

point p0; //used to another two points

point secondTop(stack<point> &stk) {

point tempPoint = stk.top();

stk.pop();

point res = stk.top(); //get the second top element

stk.push(tempPoint); //push previous top again

return res;

}

int squaredDist(point p1, point p2) {

return ((p1.x-p2.x)\*(p1.x-p2.x) + (p1.y-p2.y)\*(p1.y-p2.y));

}

int direction(point a, point b, point c) {

int val = (b.y-a.y)\*(c.x-b.x)-(b.x-a.x)\*(c.y-b.y);

if (val == 0)

return 0; //colinear

else if(val < 0)

return 2; //anti-clockwise direction

return 1; //clockwise direction

}

int comp(const void \*point1, const void\*point2) {

point \*p1 = (point\*)point1;

point \*p2 = (point\*)point2;

int dir = direction(p0, \*p1, \*p2);

if(dir == 0)

return (squaredDist(p0, \*p2) >= squaredDist(p0, \*p1))?-1 : 1;

return (dir==2)? -1 : 1;

}

vector<point> findConvexHull(point points[], int n) {

vector<point> convexHullPoints;

int minY = points[0].y, min = 0;

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

int y = points[i].y;

//find bottom most or left most point

if((y < minY) || (minY == y) && points[i].x < points[min].x) {

minY = points[i].y;

min = i;

}

}

swap(points[0], points[min]); //swap min point to 0th location

p0 = points[0];

qsort(&points[1], n-1, sizeof(point), comp); //sort points from 1 place to end

int arrSize = 1; //used to locate items in modified array

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

//when the angle of ith and (i+1)th elements are same, remove points

while(i < n-1 && direction(p0, points[i], points[i+1]) == 0)

i++;

points[arrSize] = points[i];

arrSize++;

}

if(arrSize < 3)

return convexHullPoints; //there must be at least 3 points, return empty list.

//create a stack and add first three points in the stack

stack<point> stk;

stk.push(points[0]); stk.push(points[1]); stk.push(points[2]);

for(int i = 3; i<arrSize; i++) { //for remaining vertices

while(direction(secondTop(stk), stk.top(), points[i]) != 2)

stk.pop(); //when top, second top and ith point are not making left turn, remove point

stk.push(points[i]);

}

while(!stk.empty()) {

convexHullPoints.push\_back(stk.top()); //add points from stack

stk.pop();

}

}

int main() {

point points[] = {{-7,8},{-4,6},{2,6},{6,4},{8,6},{7,-2},{4,-6},{8,-7},{0,0},

{3,-2},{6,-10},{0,-6},{-9,-5},{-8,-2},{-8,0},{-10,3},{-2,2},{-10,4}};

int n = 18;

vector<point> result;

result = findConvexHull(points, n);

cout << "Boundary points of convex hull are: "<<endl;

vector<point>::iterator it;

for(it = result.begin(); it!=result.end(); it++)

cout << "(" << it->x << ", " <<it->y <<") ";

}

**Num of Divisor of N -->O(sqrt(N))**

//Need seive to precompute prime

ll NOD(ll n){

ll i,j;

ll root=sqrt(n);

ll ret = 1;

for(i=0; i<prime.size() and prime[i]<=root; i++){

if(n%prime[i]==0){

ll cnt=1;

while(n%prime[i]==0){

cnt++;

n/=prime[i];

}

ret\*=cnt;

root=sqrt(n);

}

}

if(n>1) ret\*=2;

return ret;

}

**Numbers of Divisor from 1 to N (10^6)**

#define mx 1000000 // --> O( N\*log(N) )

ll divisor[mx+2];

void Num\_of\_Divisor(){

ll i,j;

for(i=1; i<=mx; i++){

for(j=i; j<=mx; j+=i){

divisor[j]++;

}

}

}

**Euler Totient (Phi) -->O(sqrt(N))**

// Need seive to precompute prime

ll Ephi(ll n){

ll i,j;

ll root = sqrt(n);

ll ret = n;

for(i=0; i<prime.size() and prime[i]<=root; i++){

if(n%prime[i]==0){

while(n%prime[i]==0){

n/=prime[i];

}

root=sqrt(n);

ret-=(ret/prime[i]);

}

}

if(n>1) ret-=(ret/n);

return ret;

}

**Euler Totient (Phi) Pre-compute --> O(NlogN)**

#define mx 1000003

ll phi[mx];

void PreEphi(){

ll i,j;

for(i=0; i<mx; i++) phi[i]=i;

for(i=2; i<mx; i++){

if(phi[i]==i){

for(j=i; j<mx; j+=i){

phi[j]-=(phi[j]/i);

}

}

}

}

**Sparse Table for min value**

int main(){

ll n;

cin>>n;

ll a[n];

for(ll i=0;i<n;i++) cin>>a[i];

ll sparse[n][(ll)log2(n)+5];

memset(sparse,-1,sizeof(sparse));

for(ll i=1;i<=n;i\*=2){

for(ll j=0;j<n;j++){

if(i==1) sparse[j][(ll)log2(i)]=a[j];

else if(j+i<=n)

sparse[j][(ll)log2(i)]=min(sparse[j][(ll)log2(i)-1],sparse[j+i/2][(ll)log2(i)-1]);

}

}

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

for(ll j=0;j<=(ll)log2(n)+1;j++){

if(sparse[i][j]!=-1) cout<<sparse[i][j]<<" ";

}cout<<endl;

}

ll q;

cin>>q;

while(q--){

ll l,r;

cin>>l>>r;

ll d=log2(r-l+1);

cout<<min(sparse[l][d],sparse[r-(ll)pow(2,d)+1][d])<<endl;

}

return 0;

}

**Bitwise Seive**

ll Check(ll n,ll i){return (n&(1LL<<i));}

ll Set(ll n,ll i){return (n|(1LL<<i));}

ll status[(ll)range/64+5];

Vector primes;

void sieve(){

for(ll i=3;i<=sqrt(range);i+=2){

if(Check(status[i/64],i%64)==0){

for(ll j=i\*i;j<range;j+=2\*i){

status[j/64]=Set(status[j/64],j%64);

} } }primes.pb(2);

for(ll i=3;i<range;i+=2)

if(Check(status[i/64],i%64)==0)

primes.pb(i);

}

**SOS\_DP**

//iterative version

for(int mask = 0; mask < (1<<N); ++mask){

dp[mask][-1] = A[mask]; //handle base case separately (leaf states)

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

if(mask & (1<<i))

dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];

else

dp[mask][i] = dp[mask][i-1];

}

F[mask] = dp[mask][N-1];

}

//memory optimized, super easy to code.

for(int i = 0; i<(1<<N); ++i)

F[i] = A[i];

for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){

if(mask & (1<<i))

F[mask] += F[mask^(1<<i)];

}

The above algorithm runs in O(N 2N) time.

**#SegmentTree**

void upd(ll node, ll left, ll right, ll i, ll j, ll val){

if(i>right or j<left) return;

if(left>=i and right<=j){

block[node]=(right-left+1)\*val;

prop[node]=val;

return;

}

ll l=2\*node, r=2\*node+1;

ll mid=(left+right)/2;

upd(l,left, mid, i,j,val);

upd(r,mid+1,right,i,j,val);

block[node]=block[l]+block[r];

return;

}

ll query(ll node, ll left, ll right, ll i, ll j){

if(right<i or left>j) return 0;

if(left>=i and right<=j){

return block[node];

}

ll l=2\*node, r=2\*node+1;

ll mid=(left+right)/2;

if(prop[node]!=-1){

prop[l]=prop[node];

prop[r]=prop[node];

block[l]=(mid-left+1)\*prop[node];

block[r]=(right-mid)\*block[node];

prop[node]=-1;

}

return query(l,left, mid, i,j)+query(r,mid+1,right,i,j);

}

**---------MATH---------**

**## Prime: 29996224275833, 998244353, 1000004119,** 1000003651, 1000003271, 1000001311, 1234572167, 1234572223, 5432183299, 5432182541, 5432179111,

54321793501, 54321794117, 54321792307, 10000001113

543217899787, 543217894717, 543217897961, 100000004917

**##** It gives the largest power of a prime p that divides factorial n. Express Ep(n!).

Ep(n!) = floor(n/pow(p,1)) + floor(n/pow(p,2)) +.......+ floor(n/pow(p,i)); [ where, pow(p,i)<=n ]

or, Ep(n!) = (n-Sp(n))/(p-1); [ Here, Sp(n)=the sum of the digits in the p-based expansion of n ]

# ll sum = accumulate(arr,arr+m,0LL);

# n^1 + n^2 +........+n^r = n\*(n^r - 1) / (n-1)

# nCr + nCr-1 = n+1Cr

**##** dearrangement recursive d(n)=(n-1)\*(d(n-1)+d(n-2))

base case d(1)=0, d(2)=1

another solution---> n! - nc1\*(n-1)! + nc2\*(n-2)! - ...... + (-1)^k \* nck \* (n-k)! + ......

**##** Josephous Problem ---- O(N)

josephus(n) = (josephus(n - 1) + k-1) % n + 1

josephus(1) = 1

1.