TEAMNOTE ICPC

**-----------DP-Digit**

***\*Chú ý:*** Dùng thêm mảng ‘ko’ nếu có tính đến thứ tự số 0 đằng trước. F[i][ko][ ][..]

int dp(int i, int gh, int ko, int last, int cnt) {

if (i < 0) {

if (cnt <= k) return 1;

return 0;

}

if (gh == 0 && f[i][ko][last][cnt] != -1) return f[i][ko][last][cnt];

int ghm = (gh ? a[i] : 9), kq = 0;

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

kq += dp(i - 1, gh && j == ghm, ko | (j > 0), j, cnt + (ko && abs(j - last) <= d));

if (gh == 0) f[i][ko][last][cnt] = kq;

return kq;

}

**------------HASHING**

p[0] = 1;

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

cin >> s[i], h[i] = (h[i - 1] \* 31 + s[i] - 'a' + 1) % MOD, p[i] = (p[i - 1]\*31)% MOD;

**bool** check(int len) {

mp.clear();

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

x = (h[i + len - 1] - h[i - 1] \* p[len] + MOD \* MOD) % MOD;

++mp[x];

if (mp[x] >= k) return true;

}

return false;

}

**------------TRIE**

**void** **trie**(string s) {

**int** u **=** 0, best **=** 0;

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

**int** k **=** s[i] **-** 'a';

**if** (**!**t[u][k]) {

t[u][k] **=** **++**cnt;

}

u **=** t[u][k];

}

}

**----------------LCA**

**void** **dfs**(**int** u) {

**for** (**auto** v**:** ke[u]) **if** (v **!=** up[u][0]) {

h[v] **=** h[u] **+** 1;

up[v][0] **=** u;

**for** (**int** i **=** 1; i **<** 19; **++**i) up[v][i] **=** up[up[v][i **-** 1]][i **-** 1];

dfs(v);

}

}

**int** **lca**(**int** u, **int** v) {

**if** (h[u] **!=** h[v]) {

**if** (h[u] **<** h[v]) swap(u, v);

**int** k **=** h[u] **-** h[v];

**for** (**int** i **=** 0; i **<** 19; **++**i)

**if** (k **&** (1 **<<** i)) u **=** up[u][i];

}

**if** (u **==** v) **return** u;

**for** (**int** i **=** 18; i **>=** 0; **--**i)

**if** (up[u][i] **!=** up[v][i]) u **=** up[u][i], v **=** up[v][i];

**return** up[u][0];

}

**---------------HLD**

**void** **dfs**(**int** u) {

sz[u] **=** 1;

**for** (**auto** v**:** ke[u]) **if** (v **!=** par[u]) {

par[v] **=** u;

dfs(v);

sz[u] **+=** sz[v];

}

}

**void** **HLD**(**int** u) {

**if** (**!**headChain[nchain]) headChain[nchain] **=** u;

chainIn[u] **=** nchain;

num[u] **=** **++**cnt;

**int** best **=** 0;

**for** (**auto** v**:** ke[u]) **if** (v **!=** par[u]) {

**if** (best **==** 0 **||** sz[best] **<** sz[v]) best **=** v;

}

**if** (best) HLD(best);

**for** (**auto** v**:** ke[u])

**if** (v **!=** par[u] **&&** v **!=** best) **++**nchain, HLD(v);

}

**node** ansQuery(int u, int v) {

**node** ans = {inf, inf, inf, inf};

while (1) {

if (chainIn[u] == chainIn[v]) {

ans = combine(get(1, 1, n, num[v], num[u]), ans);

break;

}

ans = combine(get(1, 1, n, num[chainHead[chainIn[u]]], num[u]), ans);

u = up[chainHead[chainIn[u]]][0];

}

return ans;

}

**--------------DIJKSTRA STRUCT**

**struct** **node**{

ll kc,s,f;

};

vector**<**node**>** ke[N];

**struct** **cmp**{

**bool** **operator**() (node a, node b) {

**return** a.kc **>** b.kc;

}

};

**void** **dijkstra**(){

priority\_queue**<**node, vector**<**node**>**, cmp**>** q;

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

**for** (**int** j**=**0;j**<=**k;**++**j) d[i][j]**=**inf;

q.push({0,1,0});

**while** (**!**q.empty()){

node tmp**=**q.top();q.pop();

ll u**=**tmp.s, w**=**tmp.kc, val**=**tmp.f;

**if** (w **>** d[u][val]) **continue**;

**for** (**auto** v **:** ke[u]) {

**if** (val**+**v.f**<=**k **&&** w**+**v.kc**<**d[v.s][val**+**v.f]) {

d[v.s][val**+**v.f]**=**w**+**v.kc;

q.push({d[v.s][val**+**v.f],v.s,val**+**v.f});

}

}

}

res**=**inf;

**for** (**int** i**=**0;i**<=**k;**++**i) res**=**min(res,d[n][i]);

**if** (res**==**inf) cout**<<**-1**<<**endl; **else** cout**<<**res**<<**endl;

}

**------------DSU : SMALL TO LARGE (Gộp Set)**

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

cin **>>** x;

par[i] **=** i;

sz[i] **=** 1;

color[i][x] **=** 1;

}

**int** **find\_set**(**int** u) {

**if** (u **!=** par[u]) par[u] **=** find\_set(par[u]);

**return** par[u];

}

**void** **union\_set**(**int** a, **int** b) {

a **=** find\_set(a); b **=** find\_set(b);

**if** (a **!=** b) {

**if** (sz[a] **<** sz[b]) swap(a, b);

**if** (color[a].size() **<** color[b].size()) swap(color[a], color[b]);

par[b] **=** a;

sz[a] **+=** sz[b];

**for** (**auto** it**:** color[b]) color[a][it.first] **+=** it.second;

color[b].clear();

}

**-------------SEGMENT TREE (LAZY UPDATE)**

**\* Lưu ý**: Down() cả update() và get(). Down() xong gán lại Lazy[k] = 0;

**struct** **node**{

**int** A, L, B, val;

} t[N];

**void** down(int id) {

int t = nodes[id].lazy;

nodes[id\*2].lazy += t;

nodes[id\*2].val += t;

nodes[id\*2+1].lazy += t;

nodes[id\*2+1].val += t;

nodes[id].lazy = 0;

}

**void** **update**(**int** k, **int** l, **int** r, **int** u, **int** v, **int** L, **int** a, **int** b) {

**if** (r **<** u **||** l **>** v) **return**;

**if** (l **>=** u **&&** r **<=** v) {

t[k].A **=** (t[k].A **+** a) **%** MOD;

t[k].B **=** (t[k].B **+** b) **%** MOD;

t[k].L **=** (t[k].L **+** L **\*** a) **%** MOD;

t[k].val **=** (t[k].val **+** (((((((l **+** r) **\*** (r **-** l **+** 1) **/** 2) **%** MOD) **;**

**return**;

}

**int** mid **=** (l **+** r) **>>** 1;

down(k, l, r, mid);

update(k **<<** 1, l, mid, u, v, L, a, b);

update(k **<<** 1 **|** 1, mid **+** 1, r, u, v, L, a, b);

t[k].val **=** (t[k **<<** 1].val **+** t[k **<<** 1 **|** 1].val) **%** MOD;

}

**int** **get**(**int** k, **int** l, **int** r, **int** u, **int** v) {

**if** (r **<** u **||** l **>** v) **return** 0;

**if** (l **>=** u **&&** r **<=** v) **return** t[k].val;

**int** mid **=** (l **+** r) **>>** 1;

down(k, l, r, mid);

**return** (get(k **<<** 1, l, mid, u, v) **+** get(k **<<** 1 **|** 1, mid **+** 1, r, u, v)) **%** MOD;

}

**-----------WALK ON TREE**

**int query**(int root, int l, int r, int k) {

if (st[root] > k) return -1; //nếu cả đoạn [l, r] đều lớn hơn k thì không thỏa mãn

if (l == r) return l; //khi đoạn có 1 phần tử thì đó là kết quả

int mid = (l + r) / 2;

(Nếu có Lazy update thì Down() ở đây)

if (st[root \* 2] <= k) //nếu min cây con trái không vượt quá k

return query(root \* 2, l, mid, k);

//ngược lại thì kết quả nằm ở bên cây con phải

return query(root \* 2 + 1, mid + 1, r, k);

}

**int query**(int root, int l, int r, int lowerbound, int k) {

if (st[root] > k) return -1; //nếu cả đoạn [l, r] đều lớn hơn k thì không thỏa mãn

if (r < lowerbound) return -1; //ta chỉ xét những vị trí không nhỏ hơn lowerbound

if (l == r) return l; //khi đoạn có 1 phần tử thì đó là kết quả

int mid = (l + r) / 2;

int res = -1;

if (st[root \* 2] <= k) //nếu min cây con trái không vượt quá k

res = query(root \* 2, l, mid, lowerbound, k);

//nếu cây con trái không tìm được kết quả <=> min nằm ngoài lowerbound

//thì ta sẽ tìm kết quả ở cây con phải

if (res == -1)

res = query(root \* 2 + 1, mid + 1, r, lowerbound, k);

return res;

}

**----------MO’S ALGORITHM + HILBERT ORDER (ĐPT O(nLogn) all Queris)**

#include <bits/stdc++.h>

using namespace std;

#define endl '\n'

#define int long long

#define ii pair<int,int>

const int N = 1e6+5, M = 1e3+1, inf = 1e18, MOD = 1e9+7;

int test, n, m, x, y, res, k, cnt1, sum, block;

int a[N], ans[N], cnt[N], maxn = 1 << 19;

**long long hilbert\_order**(int x, int y) {

long long d = 0;

for (int s = 1 << 19; s; s >>= 1) {

bool rx = x & s, ry = y & s;

d = d << 2 | rx \* 3 ^ static\_cast<int>(ry);

if (!ry) {

if (rx) {

x = maxn - x;

y = maxn - y;

}

swap(x, y);

}

}

return d;

}

**struct node**{

int l, r, id, ord;

} b[N];

**bool cmp**(node a, node b) {

return a.ord < b.ord;

}

**signed main()** {

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

cin >> n >> m;

for (int i = 1; i <= n; ++i) cin >> a[i];

for (int i = 1; i <= m; ++i) cin >> b[i].l >> b[i].r, b[i].id = i, **b[i].ord = hilbert\_order(b[i].l, b[i].r);**

block = **sqrt(n);**

sort(b + 1, b + m + 1, cmp);

int L = 0, R = 0;

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

while (L < b[i].l) {

res += (-2 \* cnt[a[L]] + 1) \* a[L];

--cnt[a[L]];

++L;

}

while (L > b[i].l) {

--L;

res += (2 \* cnt[a[L]] + 1) \* a[L];

++cnt[a[L]];

}

while (R < b[i].r) {

++R;

res += (2 \* cnt[a[R]] + 1) \* a[R];

++cnt[a[R]];

}

while (R > b[i].r) {

res += (-2 \* cnt[a[R]] + 1) \* a[R];

--cnt[a[R]];

--R;

}

ans[b[i].id] = res;

}

for (int i = 1; i <= m; ++i) cout << ans[i] << endl;

}