**Template :-**

ios\_base::sync\_with\_stdio(false), cin.tie(nullptr), cout.tie(nullptr);

#include <bits/stdc++.h>

#define ll long long

#define vi vector<ll>

#define vii vector<vi>

#define vc vector<char>

#define vcc vector<vc>

#define mi map<int,int>

#define mc map<char,int>

#define sortx(X) sort(X.begin(),X.end());

#define all(X) X.begin(),X.end();

#define ln '\n'

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

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

**Function :-**memset(BIT, 0, sizeof(BIT)); //intlize arr with 0

int converted = stoi(dec, nullptr, 16);

// compare function -> min is top

struct com {

bool operator()(int &l, int &r){

return l > r;

}

};

priority\_queue<int, vi, com> L;

//get random index of array

#include <random>

mt19937 gen(123456789);

uniform\_int\_distribution<ll> dist(0, points.size() - 1);

ll idx1 = dist(gen);

//rotate matrix

void rotat(vii& X) {

vii tmp(X[0].size(), vi(X.size()));

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

for (int j = 0; j < X[i].size(); j++)

tmp[X[i].size() - j - 1][i] = X[i][j];

X = tmp;

}

if (str.find(str1) != string::npos)

transform(str.begin(), str.end(), str.begin(), ::toupper);

// get all sequances of array

string a = "abcd";

do {

cout << a;

} while (next\_permutation(a.begin(), a.end()));

**Bitwise :-**

//first digit of number

X & -X

//pow of two just have 1 only

n & (n - 1) && (n != 0) -- > 0

//get the velue of but in number

(x >> indx) & 1

bitset<2000007> Z;

// all point start with one

Z.set();

// get first digit with 1 after b

a = Z.\_Find\_next(b);

int n = 10;

bitset<32> s(n);

00000000000000000000000000001010

s.to\_ullong() -- >> to decimal value

s.set(3); // Set bit at position 3 to 1

s.reset(7); // Set bit at position 7 to 0

bool bit3 = s[3]; // Check bit at position 3

bool getBit(ll num, int idx) {

return ((num >> idx) & 1);

}

ll setBit(ll num, int idx, bool val) {

return val ? (num | (1LL << idx)) : (num & ~(1LL << idx));

}

ll flipBit(ll num, int idx) {

return (num ^ (1LL << idx));

}

ll leastBit(ll num) {

return (num & -num);

}

//num%mod, mod is a power of 2

ll Mod(ll num, ll mod) {

return (num & mod - 1);

}

bool isPowerOfTwo(ll num) {

return (num & num - 1) == 0;

}

int turnOnLastZero(int S) {

return S | S + 1;

}

int turnOffLastBit(int S) {

return S & S - 1;

}

int turnOnLastConsecutiveZeroes(int S) {

return S | S - 1;

}

int turnOffLastConsecutiveBits(int S) {

return S & S + 1;

}

vector<int> genAllSubmask(int mask) {

vector<int> v;

for (int subMask = mask;; subMask = (subMask - 1) & mask) {

v.push\_back(subMask);

if (subMask == 0)

break;

}

return v;

}

**math :-**

Pi 3.141592653589793238462643383279502884

// seive prime factorization

vi prime(1e7 + 10, 0);

for (int i = 2; i \* i < 1e7; ++i) {

if (prime[i]) continue;

for (int j = i; j \* j <= 1e7; j+=i)

prime[j] = i;

}

// mode\_inverse

ll exp\_mod(ll a, ll n) {

ll res = 1;

for (; n > 0; n /= 2) {

if (n % 2) res = res \* a % mode;

a = a \* a % mode;

}

return res;

}

ll inv\_mod(ll a) {

return exp\_mod(a, mode - 2);

}

//gcd(a+b\*c,b) = gcd(a,b)

//ax+by=gcd(a,b)

ll gcdExtended(ll a, ll b, ll\* x, ll\* y)

{

if (a == 0){

\*x = 0;

\*y = 1;

return b;

}

ll x1, y1;

ll gcd = gcdExtended(b % a, a, &x1, &y1);

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

\*y = x1;

return gcd;

}

ll gcd(ll a, ll b){

if (b == 0)

return a;

return gcd(b, a % b);

}

ll lcm(ll a, ll b){

return (a / gcd(a, b)) \* b;

}

nCr = fac(n)/fec(r)\*fac(n-r)

nCr(n,r) = nCr(n - 1, r - 1) + nCr(n - 1, r)

nPr = fac(n)/fac(n-r)

double log\_a\_to\_base\_b(double a, double b) {

return log2(a) / log2(b);

}

(a + b) % c = ((a % c) + (b % c)) % c

(a \* b) % c = ((a % c) \* (b % c)) % c

(a - b) % c = ((a % c) - (b % c) + c) % c

(a / b) % c = ((a % c) \* ((b ^ -1) % c)) % c

ll modfpow(ll x, ll y, ll z){

ll result = 1;

while (y > 0){

if (y % 2 == 1)result = ((result % z) \* (x % z)) % z;

x = ((x % z) \* (x % z)) % z;

y = y / 2;

}

return result;

}

long long sumOdd(long long num) {

long long res = (num + 1) / 2;

long long finalRes = res \* res;

return finalRes;

}

long long sumEven(long long num) {

long long res = (num \* (num + 1));

return res;

//summ of numpers divisible by m from 1 to n

ll first\_term = m;

ll last\_term = (n / m) \* m;

ll num\_terms = (last\_term - first\_term) / m + 1;

ll sum = (num\_terms \* (first\_term + last\_term)) / 2;

void matrix\_multiply(int A[][MAX\_SIZE], int B[][MAX\_SIZE], int C[][MAX\_SIZE], int m, int n, int p) {

// Multiply matrices A and B, storing the result in matrix C

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

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

int sum = 0;

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

sum += A[i][k] \* B[k][j];

}

C[i][j] = sum;

}

}

}

}

phi(a\* b) = phi(a) \* phi(b)

time = sqrt(n)

ll phi(ll n)

{

ld result = n;

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

if (n % i == 0){

while (n % i == 0) n /= i;

result \*= (1.0 - (1.0 / float(i)));

}

}

if (n > 1)result \*= (1.0 - (1.0 / float(n)));

ll jj = result;

return jj;

}

ll catalan(int n){

return (nCr(2 \* n, n) \* Inv(n + 1)) % MOD;

}

**Binary-search-tree :-**

//creat node

struct node

{

int key;

struct node\* left, \* right;

};

struct node\* newNode(int item)

{

node\* temp = new node;

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

//insert to node node

node\* insert(struct node\* node, int key)

{

if (node == NULL)

return newNode(key);

if (key < node->key) node->left = insert(node->left, key);

else node->right = insert(node->right, key);

return node;

}

//delete node

struct node\* minValueNode(struct node\* node)

{

struct node\* current = node;

while (current && current->left != NULL)

current = current->left;

return current;

}

struct node\* deleteNode(struct node\* root, int key)

{

if (root == NULL) return root;

if (key < root->key) root->left = deleteNode(root->left, key);

else if (key > root->key) root->right = deleteNode(root->right, key);

else {

if (root->left == NULL) {

struct node\* temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL) {

struct node\* temp = root->left;

free(root);

return temp;

}

struct node\* temp = minValueNode(root->right);

root->key = temp->key;

root->right = deleteNode(root->right, temp->key);

}

return root;

}

// get element of BST in ordered way

void inorder(struct node\* root)

{

if (root != NULL)

{

inorder(root->left);

cout << root->key << " ";

inorder(root->right);

}

}

**Fenwick-tree :-**

ll rangequery(const vi& Z, ll l, ll r) {

ll summ = 0;

for (int i = r; i > 0; i -= i & -i)

summ += Z[i];

for (int i = l - 1; i > 0; i -= i & -i)

summ -= Z[i];

return summ;

}

void updatepoint(vi& Z, ll n, ll at) {

ll summ = 0;

//sum = rangequery(Z, at, at)

for (int i = at; i < Z.size(); i += i & -i)

Z[i] += n - summ;

}

// To fenwik of 2d array we nested loop it to get summ of squir  
for (int x; x > 0; x -= x & -x)

for (int yy = y; yy > 0; yy -= yy & -yy)

sum += BIT[x][yy];

// I can use it to make sorted dynamic array   
// to insert update(x,+1) delete update(x,-1)

for ( ; i < BIT.size(); i+= i & -i)

BIT[i]++;

// To get the value of index O(log(n)^2) we use lower pound

while (l < r) {

at = (l + r) / 2;

if (k <= sum(at, BIT)) r = at;

else l = at + 1;

}

// to get all element greater than k in range l , r -> take quary in array and sort in  
// I can make fenwick of XoR

// I can binary search for prefix value on it

**Segment-tree :-**

// it like fenwick but can do more operation like min or max in range

// we first make array of size 2\*n -> left and right are 2\*i+1 – 2\*i+2

int getMid(int s, int e) { return s + (e - s) / 2; }

int getSumUtil(int\* st, int ss, int se, int qs, int qe, int si){

if (qs <= ss && qe >= se) return st[si];

if (se < qs || ss > qe) return 0;

int mid = getMid(ss, se);

return getSumUtil(st, ss, mid, qs, qe, 2 \* si + 1) +

getSumUtil(st, mid + 1, se, qs, qe, 2 \* si + 2);

}

void updateValueUtil(int\* st, int ss, int se, int i, int diff, int si){

if (i < ss || i > se)

return;

st[si] = st[si] + diff;

if (se != ss){

int mid = getMid(ss, se);

updateValueUtil(st, ss, mid, i, diff, 2 \* si + 1);

updateValueUtil(st, mid + 1, se, i, diff, 2 \* si + 2);

}

}

int constructSTUtil(int arr[], int ss, int se, int\* st, int si)

{

if (ss == se){

st[si] = arr[ss];

return arr[ss];

}

int mid = getMid(ss, se);

st[si] = constructSTUtil(arr, ss, mid, st, si \* 2 + 1) +

constructSTUtil(arr, mid + 1, se, st, si \* 2 + 2);

return st[si];

}

int\* constructST(int arr[], int n)

{

int x = (int)(ceil(log2(n)));

int max\_size = 2 \* (int)pow(2, x) - 1;

int\* st = new int[max\_size];

constructSTUtil(arr, 0, n - 1, st, 0);

return st;

}

// lazy propagation -> add value to range and update it only of needed  
// we use another array call lazy to store values

//to get summ or update we first check if there was lazy value

// put this if at the top of get sum and update

if (lazy[si] != 0){

tree [si] += (se - ss + 1) \* lazy[si];

if (ss != se)

{

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

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

}

lazy[si] = 0;

}

// my function

vi Z(1e6 \* 3, 0);

ll n;

int getrange(int s, int e, int l, int r, int k) {

if (s > r || e < l) return 0;

if (s >= l && e <= r) return Z[k];

int at = (s + e) / 2;

return getrange(s, at, l, r, k \* 2 + 1) + getrange(at + 1, e, l, r, k \* 2 + 2);

}

void updates(int s, int e, int k, int index, int val) {

if (s == e) {

Z[k] += val;

return;

}

int at = (s + e) / 2;

if (index >= s && index <= at) updates(s, at, k \* 2 + 1, index, val);

else updates(at + 1, e, k \* 2 + 2, index, val);

Z[k] += val;

}

void update(int i, ll k) {

updates(0, n - 1, 0, i, k);

}

ll summ(int i) {

return getrange(0, n - 1, 0, i, 0);

}

ll tree[1000000] = { 0 };

ll lazy[1000000] = { 0 };

void updateRangeUtil(ll si, ll ss, ll se, ll us,

ll ue, ll diff)

{

if (lazy[si] != 0) {

tree[si] = max(lazy[si], tree[si]);

if (ss != se)

{

lazy[si \* 2 + 1] = max(lazy[si], lazy[si \* 2 + 1]);

lazy[si \* 2 + 2] = max(lazy[si], lazy[si \* 2 + 2]);

}

lazy[si] = 0;

}

if (ss > se || ss > ue || se < us)

return;

if (ss >= us && se <= ue)

{

tree[si] = max(tree[si], diff);

if (ss != se)

{

lazy[si \* 2 + 1] = max(diff, lazy[si \* 2 + 1]);

lazy[si \* 2 + 2] = max(diff, lazy[si \* 2 + 2]);

}

return;

}

ll mid = (ss + se) / 2;

updateRangeUtil(si \* 2 + 1, ss, mid, us, ue, diff);

updateRangeUtil(si \* 2 + 2, mid + 1, se, us, ue, diff);

tree[si] = max(tree[si \* 2 + 1], tree[si \* 2 + 2]);

}

void updateRange(ll n, ll us, ll ue, ll diff)

{

updateRangeUtil(0, 0, n - 1, us, ue, diff);

}

ll getSumUtil(ll ss, ll se, ll qs, ll qe, ll si)

{

if (lazy[si] != 0) {

tree[si] = max(lazy[si], tree[si]);

if (ss != se)

{

lazy[si \* 2 + 1] = max(lazy[si], lazy[si \* 2 + 1]);

lazy[si \* 2 + 2] = max(lazy[si], lazy[si \* 2 + 2]);

}

lazy[si] = 0;

}

if (ss > se || ss > qe || se < qs)

return 0;

if (ss >= qs && se <= qe)

return tree[si];

ll mid = (ss + se) / 2;

return max(getSumUtil(ss, mid, qs, qe, 2 \* si + 1),

getSumUtil(mid + 1, se, qs, qe, 2 \* si + 2));

}

ll getSum(ll n, ll qs, ll qe)

{

return getSumUtil(0, n - 1, qs, qe, 0);

}

**Trie-tree :-**

// to start new node node \* X = new node();

struct node {

node\* next[2] = { NULL };

int val = 0;

};

node\* root, \* AT\_TRIE, \* TRIE = new node[32 \* 100100];

void add(node\*& X, int n, int k) {

if (!X) \*(X = AT\_TRIE++) = node();

if (!n) X->val += k;

else add(X->next[n & 1], n >> 1, k);

}

int get(node\* X, int n) {

if (!X) return 0;

if (!n) return X->val;

return get(X->next[n & 1], n >> 1);

}

struct node {

node\* next[26] = { NULL };

int val = INT32\_MAX;

};

node\* root;

int st;

void add(node\*& X, string& s, int at) {

if (!X)X = new node();

X->val = min(X->val, st);

if (at == st)return;

add(X->next[s[at] - 'a'], s, at + 1);

}

int get(node\* X, string& s, int at) {

if (!X) return -1;

if (at == st) return X->val;

return get(X->next[s[at] - 'a'], s, at + 1);

}

**Graph ------------------🡪>>>>>>>>>**

// Tree to know every child and it’s parents

struct node

{

int val, pra;

vector<int> c;

};

vii Y;

vector<node> Z;

void build(int k) {

for (int i = 0; i < Y[k].size(); i++) {

if (Y[k][i] == Z[k].pra)continue;

Z[k].c.push\_back(Y[k][i]);

Z[Y[k][i]].pra = k;

build(Y[k][i]);

}

}

**BFS :-**

// to get level of the node or distance

vii Y;

vector<int> lvl(10100, -1);

int nodeLevel(int start)

{

queue<int> Q;

Q.push(start);

lvl[start] = 0;

while (!Q.empty()) {

int re = Q.front();

Q.pop();

for (int i = 0; i < Y[re].size(); i++) {

if (lvl[Y[re][i]] != -1) continue;

lvl[Y[re][i]] = lvl[re] + 1;

Q.push(Y[re][i]);

}

}

}

**Bijaction :-**

vector<vector<pair<ll, ll>>> Y;

vi shortestPath(int src)

{

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

vi dist(Y.size(), INT64\_MAX);

pq.push({ 0,src });

dist[src] = 0;

while (!pq.empty()) {

int u = pq.top().second;

pq.pop();

for (int i = 0; i < Y[u].size(); i++)

{

int v = Y[u][i].first;

int weight = Y[u][i].second;

if (dist[v] > dist[u] + weight) {

dist[v] = dist[u] + weight;

pq.push(make\_pair(dist[v], v));

}

}

}

return dist;

}

**DP ------------------🡪>>>>>>>>>**

ll Longest\_Increasing\_SubSeq(vi X) {

vi Z;

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

ll re = upper\_bound(Z.begin(), Z.end(), X[i]) - Z.begin();

if (re == Z.size())Z.push\_back(X[i]);

else Z[re] = X[i];

}

return Z.size();

}

**string ------------------🡪>>>>>>>>>**

**suffix :-**

// it take longer time with longlong

void radix\_Sort(vi& X,vi &Z) {

ll n = X.size(), re = 0;

vi Y(max(n, 255ll)), K(n);

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

Y[Z[X[i]]]++;

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

re += Y[i],Y[i] = re - Y[i];

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

K[Y[Z[X[i]]]] = X[i],Y[Z[X[i]]]++;

X = K;

}

void suff(vi& X, vi& Z, ll k) {

ll n = X.size();

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

X[i] = (X[i] - k + n) % n;

radix\_Sort(X, Z);

vi K(n);

K[X[0]] = 0;

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

if (Z[X[i]] != Z[X[i - 1]] || Z[(X[i] + k) % n] != Z[(X[i - 1] + k) % n])

K[X[i]] = K[X[i - 1]] + 1;

else K[X[i]] = K[X[i - 1]];

if (k \* 2 < Z.size()) suff(X, K, k \* 2);

}

vi suffix(string s) {

s += '$';

ll n = s.size();

vi Z(n), X(n);

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

Z[i] = s[i], X[i] = i;

radix\_Sort(X, Z);

Z[X[0]] = 0;

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

if (s[X[i]] == s[X[i-1]])Z[X[i]] = Z[X[i-1]];

else Z[X[i]] = Z[X[i - 1]] + 1;

suff(X, Z, 1);

return X;

}

// suffix faster time;

vector<int> suffix\_array(const string& s, int lim = 128)

{

int n = s.size() + 1, cl = lim;

vector<int> sa(n), c(n), cn(n), pn(n), cnt(max(n, lim));

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

for (int i = 0; i < n - 1; i++) c[i] = s[i];

for (int k = 0; k <= n; k = k ? 2 \* k : 1)

{

if (k) for (int i = 0; i < n; i++) pn[i] = (sa[i] + n - k) % n;

for (int i = 0; i < cl; i++) cnt[i] = 0;

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

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

for (int i = n - 1; i >= 0; i--) sa[--cnt[c[pn[i]]]] = pn[i];

cn[sa[0]] = 0;

cl = 1;

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

if (c[sa[i]] != c[sa[i - 1]] || c[(sa[i] + k) % n] != c[(sa[i - 1] + k) % n])

cl++;

cn[sa[i]] = cl - 1;

}

if (cl == n) break;

swap(c, cn);

}

sa.erase(sa.begin());;

return sa;

}

ll LowerBound(const vi& X, const string& s, const string& t) {

ll l = 0, r = s.size(), at;

while (l < r)

{

at = (l + r) / 2;

if (s.substr(X[at], t.size()) >= t)

r = at;

else l = at + 1;

}

return l;

}

//longest common prefixes for suffix array

vi lcp\_suffix(const vi& X, const string& s) {

ll n = s.size(), k = 0;

vi Z(n), Y(n);

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

Z[X[i]] = i;

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

{

for (; k < n - i - 1; k++)

if (s[i + k] != s[X[Z[i] - 1] + k])

break;

Y[Z[i]] = k;

k = max(k - 1, 0);

}

return Y;

}

// get number of diffrent subarray by (total numbers of subarr - summ of lcp array)

// to get longest lcp betwen 2 string

void solve() {

ll n, k = 0, re, summ = 0,mx = 0;

string s , t;

cin >> s >> t;

n = s.size() + 1;

s += "$" + t + "#";

auto X = suffix\_array(s), Y = lcp\_suffix(X, s);

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

if ((X[i] >= n && X[i - 1] < n) || (X[i] < n && X[i - 1] >= n))

if (Y[i] > mx)

mx = max(mx, Y[i]),re = X[i];

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

cout << s[i + re];

}

**Other :-**

// monomatic stack

ll n; cin >> n;

vi v(n); cin(v);

vector<ll>m(n, -1);

stack<pair<ll, ll>>st;

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

if (st.empty())

{

st.emplace(v[i], i);

}

else if (v[i] > st.top().first)

{

while (!st.empty() && v[i] > st.top().first)

{

m[st.top().second] = i;

st.pop();

}

st.emplace(v[i], i);

}

else

{

st.emplace(v[i], i);

}

}

ll q; cin >> q;

while (q--)

{

ll x; cin >> x;

if (m[x - 1] == -1)cout << "-1\n";

else cout << m[x - 1] + 1 << "\n";

}

ll ncr(ll n, ll r) {

if (r > n) {

return 0;

}

ll res = 1;

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

res \*= (n - i);

res /= (i + 1);

}

return res;

}

ll nCr(ll n, ll r) {

if (r > n) return 0;

if (r == 0 || n == r) return 1;

ld res = 0;

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

res += log(n - i) - log(i + 1);

}

return (ll)round(exp(res));

}