#include <bits/stdc++.h>

using namespace std;

#define ll long long

#define l1(i, n) for (ll i = 1; i <= n; i++)

#define l0(i, n) for (ll i = 0; i < n; i++)

#define pb push\_back

#define sorted(x) sort(x.begin(), x.end())

#define reversed(x) reverse(x.begin(), x.end())

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

#define ms(a, b) memset(a, b, sizeof(a));

#define cases(tc) cout<<"Case #"<<tc<<": "

#define nl cout<<"\n";

#define pi acos(-1)

#define mod 1000000007

#define inf 999999999999999999

#define maxn 100001

#define xx first

#define yy second

int main()

{

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

cin.tie(0), cout.tie(0);

ll t;

cin>>t;

while(t--){

ll n;

cin>>n;

}

return 0;

}

**DEBUG:**

#define Gene template< class

#define Rics printer& operator,

Gene c > struct rge {c b, e;};

Gene c > rge<c> range(c i, c j) { return {i, j};}

struct printer {

~printer() {cerr << endl;}

Gene c > Rics(c x) { cerr << boolalpha << x; return \*this;}

Rics(string x) {cerr << x; return \*this;}

Gene c, class d > Rics(pair<c, d> x) { return \*this, "(", x.first, ", ", x.second, ")";}

Gene ... d, Gene ... > class c > Rics(c<d...> x) { return \*this, range(begin(x), end(x));}

Gene c > Rics(rge<c> x) {\*this, "["; for (auto it = x.b; it != x.e; ++it)\*this, (it == x.b ? "" : ", "), \*it; return \*this, "]";

}

};

#define debug() cerr<<"LINE "<<\_\_LINE\_\_<<" >> ", printer()

#define dbg(x) "[",#x,": ",(x),"] "

**UNIFORM RANDOM**:

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

int my\_rand(int l, int r) {

return uniform\_int\_distribution<int>(l, r) (rng);

}

**PBDS:**

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

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

using namespace \_\_gnu\_pbds;

#define ordered\_set tree<int, null\_type,less<int>, rb\_tree\_tag,tree\_order\_statistics\_node\_update>

//declare ordered\_set setname

//setname.find\_by\_order(k): It returns to an iterator to the kth element (counting from zero) in the set in O(logn)

//setname.order\_of\_key(k) : It returns to the number of items that are strictly smaller than our item k in O(logn) time

**POWER:**

ll power(ll a, ll b){

if(b==0) return 1;

ll temp=power(a, b/2);

if(b & 1) return a\*temp\*temp;

else return temp\*temp; }

**MODULAR EXPONENTATION:**

ll pmod(ll a, ll b, ll mod){

if(b==0) return 1;

ll temp=pmod(a, b/2, mod);

if(b & 1) return (((a\*temp)%mod)\*temp)%mod;

else return (temp\*temp)%mod; }

**SIEVE:**

vector <ll> primes;

ll chk[maxn];

void sieve(){

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

if(!chk[i]){

for(ll j=i\*i; j<maxn; j+=i) if(!chk[j]) chk[j]=i;

}

}

for(ll i=2; i<maxn; i++) if(!chk[i]) chk[i]=i;

for(ll i=2; i<maxn; i++) if(chk[i]==i) primes.pb(i);

}

**PRIME FACTORIZE:**

//SIEVE

vector <ll> fac;  
void factors(ll a){

fac.clear();

ll val=a;

for(ll i=0; i<primes.size() && primes[i]\*primes[i]<=a; i++){

if(!(val%primes[i])){

fac.pb(primes[i]);

while(!(val%primes[i])){

val/=primes[i];

}

}

}

if(val!=1) fac.pb(val); }

**BFS:**

vector <ll> adj[maxn];

ll dis[maxn];

void bfs(ll n, ll st){

l1(i, n) dis[i]=-1;

queue <ll> q;

q.push(st);

dis[st]=0;

while(!q.empty()){

ll a=q.front();

q.pop();

for(ll i=0; i<adj[a].size(); i++){

if(dis[adj[a][i]]==-1){

dis[adj[a][i]]=dis[a]+1;

q.push(adj[a][i]);

} } } }

**DFS:**

vector <ll> adj[maxn];

bool visited[maxn];

void dfs(ll a){

if(!visited[a]){

visited[a]=true;

for(ll i=0; i<adj[a].size(); i++){

dfs(adj[a][i]);

}

}

return; }

**DJIKSTRA:**

vector <pair<ll, ll>> adj[maxn];

vector <ll> dist(maxn, inf);

vector <bool> visited(maxn);

void dijkstra(ll s){

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

pq.push({0, s});

dist[s]=0;

while(!pq.empty()){

ll u=pq.top().second;

pq.pop();

if(visited[u]) continue;

visited[u]=true;

for(ll i=0; i<adj[u].size(); i++){e

ll v=adj[u][i].first;

ll w=adj[u][i].second;

if(dis[v]>dis[u]+w){

dis[v]=dis[u]+w;

pq.push({-dis[v], v});

} } } }

**DSU:**

ll cnt;

ll par[maxn];

ll rnk[maxn];

ll sz[maxn];

void make\_set(ll a){

par[a]=a;

rnk[a]=1;

sz[a]=1; }

ll find(ll a){

if(a==par[a]) return a;

return par[a]=find(par[a]); }

void merge(ll a, ll b){

ll p1=find(a);

ll p2=find(b);

if(p1==p2){

return;

}

if(p1>p2) swap(p1, p2);

par[p1]=p2;

sz[p2]+=sz[p1];

if(rnk[p1]==rnk[p2]) rnk[p2]++;

cnt--; }

bool same(ll a, ll b){

return(par[a]==par[b]); }

ll count(){

return cnt; }

ll get\_size(ll a){

return sz[par[a]];}

**MSTKRUSKAL**

//dsu

int main()

{

ll t;

t=1;

while(t--){

cnt=0;

ll n, m;

cin>>n>>m;

vector<vector <ll>> v(m, vector <ll> (3));

ll ans=0;

l0(i, n) make\_set(i);

l0(i, m){

ll a, b, c;

cin>>a>>b>>c;

v[i]={c, a, b};

}

sorted(v);

l0(i, m){

if(find(v[i][1])!=find(v[i][2])){

merge(v[i][1], v[i][2]);

ans+=v[i][0];

}

}

cout<<ans;

nl

}

return 0; }

**EULERTOUR:**

vector <ll> adj[maxn];

ll dt[2\*maxn];

ll st[maxn];

ll en[maxn];

ll cnt=0;

void dfs(ll a, ll p){

dt[++cnt]=a;

st[a]=cnt;

for(ll i=0; i<adj[a].size(); i++){

if(adj[a][i]!=p) dfs(adj[a][i], a);

}

dt[++cnt]=-a;

en[a]=cnt;

return;}

**LCA:**

vector <ll> adj[maxn];

ll par[maxn][19];

ll lev[maxn];

void dfs(ll a, ll p){

par[a][0]=p;

lev[a]=lev[p]+1;

for(ll i=1; i<=18; i++){

par[a][i]=par[par[a][i-1]][i-1];

}

for(ll i=0; i<adj[a].size(); i++){

if(adj[a][i]!=p) dfs(adj[a][i], a);

}

return; }

ll lca(ll u, ll v) {

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

for(ll k=18; k>=0; k--) if (lev[par[u][k]] >= lev[v]) u = par[u][k];

if (u == v) return u;

for(ll k=18; k>=0; k--) if (par[u][k] != par[v][k]) u = par[u][k], v = par[v][k];

return par[u][0]; }

**CENTROID DECOMPOSITION**

vector <ll> adj[maxn];

ll del[maxn], sz[maxn], par[maxn];

ll cursz;

void dfs(ll a, ll p){

sz[a]=1;

for(ll i=0; i<adj[a].size(); i++){

ll nd=adj[a][i];

if(nd!=p && !del[nd]){

dfs(adj[a][i], a);

sz[a]+=sz[adj[a][i]];

} } }

ll findcen(ll a, ll p){

for(ll i=0; i<adj[a].size(); i++){

ll nd=adj[a][i];

if(nd!=p && !del[nd] && sz[nd]>cursz/2){

return findcen(nd, a);

}

}

return a; }

void decomp(ll a, ll p){

dfs(a, -1);

cursz=sz[a];

ll cen=findcen(a, -1);

if(p==-1) p=cen;

par[cen]=p, del[cen]=1;

for(ll i=0; i<adj[cen].size(); i++){

ll nd=adj[cen][i];

if(!del[nd]) decomp(nd, cen);

} }

**SEGMENT TREE**:

ll seg[4\*maxn];

ll lazy[4\*maxn];

ll dt[maxn];

void build(ll st, ll en, ll nd){

if(st==en){

seg[nd]=dt[st];

return;

}

ll mid=(st+en)/2;

build(st, mid, 2\*nd);

build(mid+1, en, 2\*nd+1);

seg[nd]=seg[2\*nd]+seg[2\*nd+1]; }

void update(ll st, ll en, ll nd, ll l, ll r, ll val){

if(lazy[nd]!=0){

ll temp=lazy[nd];

lazy[nd]=0;

seg[nd]+=temp\*(en-st+1);

if(st!=en){

lazy[2\*nd]+=temp;

lazy[2\*nd+1]+=temp;

}

}

if(st>r || en<l){

return;

}

if(st>=l && en<=r){

seg[nd]+=val\*(en-st+1);

if(st!=en){

lazy[2\*nd]+=val;

lazy[2\*nd+1]+=val;

}

return; }

ll mid=(st+en)/2;

update(st, mid, 2\*nd, l, r, val);

update(mid+1, en, 2\*nd+1, l, r, val);

seg[nd]=seg[2\*nd]+seg[2\*nd+1]; }

ll query(ll st, ll en, ll nd, ll l, ll r){

if(lazy[nd]!=0){

ll temp=lazy[nd];

lazy[nd]=0;

seg[nd]+=temp\*(en-st+1);

if(st!=en){

lazy[2\*nd]+=temp;

lazy[2\*nd+1]+=temp;

}

}

if(st>r || en<l){

return 0;

}

if(st>=l && en<=r){

return seg[nd];

}

ll mid=(st+en)/2;

return query(st, mid, 2\*nd, l, r) + query(mid+1, en, 2\*nd+1, l, r); }

**PERSISTANT SEGMENT TREE:**

ll dt[maxn];

ll seg[120\*maxn];

ll lazy[120\*maxn];

ll lef[120\*maxn];

ll rig[120\*maxn];

ll ver[maxn];

ll nf=1;

void build(ll st, ll en, ll nd){

if(st==en){

seg[nd]=dt[st];

return;

}

lef[nd]=++nf;

rig[nd]=++nf;

ll mid=(st+en)/2;

build(st, mid, lef[nd]);

build(mid+1, en, rig[nd]);

seg[nd]=seg[lef[nd]]+seg[rig[nd]]; }

ll propogate(ll st, ll en, ll nd, ll val){

ll nnd=++nf;

lef[nnd]=lef[nd];

rig[nnd]=rig[nd];

lazy[nnd]=lazy[nd];

lazy[nnd]+=val;

seg[nnd]=seg[nd]+(en-st+1)\*val;

return nnd; }

ll update(ll st, ll en, ll nd, ll l, ll r, ll val){

if(lazy[nd]!=0){

ll temp=lazy[nd];

lazy[nd]=0;

if(st!=en){

ll mid=(st+en)/2;

lef[nd]=propogate(st, mid, lef[nd], temp);

rig[nd]=propogate(mid+1, en, rig[nd], temp);

}

}

if(st>r || en<l){

return nd;

}

ll nnd=++nf;

if(st>=l && en<=r){

seg[nnd]=seg[nd]+(en-st+1)\*val;

lazy[nnd]=val;

lef[nnd]=lef[nd];

rig[nnd]=rig[nd];

return nnd;

}

ll mid=(st+en)/2;

lef[nnd]=update(st, mid, lef[nd], l, r, val);

rig[nnd]=update(mid+1, en, rig[nd], l, r, val);

seg[nnd]=seg[lef[nnd]]+seg[rig[nnd]];

return nnd; }

ll query(ll st, ll en, ll nd, ll l, ll r){

if(lazy[nd]!=0){

ll temp=lazy[nd];

lazy[nd]=0;

if(st!=en){

ll mid=(st+en)/2;

lef[nd]=propogate(st, mid, lef[nd], temp);

rig[nd]=propogate(mid+1, en, rig[nd], temp);

}

}

if(st>r || en<l){

return 0;

}

if(st>=l && en<=r){

return seg[nd];

}

ll mid=(st+en)/2;

return query(st, mid, lef[nd], l, r) + query(mid+1, en, rig[nd], l, r); }

**PERSISTANT SEGMENT TREE LAZY:**

seg[120\*maxn];

ll lazy[120\*maxn];

ll lef[120\*maxn];

ll rig[120\*maxn];

ll ver[maxn];

ll nf=1;

// Take build() from above

ll propogate(ll st, ll en, ll nd, ll val){

ll nnd=++nf;

lef[nnd]=lef[nd];

rig[nnd]=rig[nd];

lazy[nnd]=lazy[nd];

lazy[nnd]+=val;

seg[nnd]=seg[nd]+(en-st+1)\*val;

return nnd; }

ll update(ll st, ll en, ll nd, ll l, ll r, ll val){

if(lazy[nd]!=0){

ll temp=lazy[nd];

lazy[nd]=0;

if(st!=en){

ll mid=(st+en)/2;

lef[nd]=propogate(st, mid, lef[nd], temp);

rig[nd]=propogate(mid+1, en, rig[nd], temp);

}

}

if(st>r || en<l){

return nd;

}

ll nnd=++nf;

if(st>=l && en<=r){

seg[nnd]=seg[nd]+(en-st+1)\*val;

lazy[nnd]=val;

lef[nnd]=lef[nd];

rig[nnd]=rig[nd];

return nnd;

}

ll mid=(st+en)/2;

lef[nnd]=update(st, mid, lef[nd], l, r, val);

rig[nnd]=update(mid+1, en, rig[nd], l, r, val);

seg[nnd]=seg[lef[nnd]]+seg[rig[nnd]];

return nnd; }

ll query(ll st, ll en, ll nd, ll l, ll r){

if(lazy[nd]!=0){

ll temp=lazy[nd];

lazy[nd]=0;

if(st!=en){

ll mid=(st+en)/2;

lef[nd]=propogate(st, mid, lef[nd], temp);

rig[nd]=propogate(mid+1, en, rig[nd], temp);

}

}

if(st>r || en<l){

return 0;

}

if(st>=l && en<=r){

return seg[nd];

}

ll mid=(st+en)/2;

return query(st, mid, lef[nd], l, r) + query(mid+1, en, rig[nd], l, r); }

**MO'S ALGORITHM:**

ll sq;

bool compare(pair<ll, ll> p1, pair<ll, ll> p2){

if(p1==p2) return false;

if(p1.first/sq!=p2.first/sq) return p1.first/sq<p2.first/sq;

return p1.second<p2.second;

}

int main()

{

ll t;

t=1;

while(t--){

ll n, q;

cin>>n>>q;

vector <ll> x(n+1);

l1(i, n) cin>>x[i];

vector <pair<ll, ll>> v;

l0(i, q){

ll a, b;

cin>>a>>b;

v.pb({a, b});

}

sq=sqrt(n);

sort(v.begin(), v.end(), compare);

vector <ll> freq(n+1, 0);

ll l=0, r=-1;

ll ans=0;

l0(i, q){

if(l<v[i].first){

while(l!=v[i].first){

freq[x[l]]--;

if(!freq[x[l]]) ans--;

l++;

}

}

else if(l>v[i].first){

while(l!=v[i].first){

freq[x[l-1]]++;

if(freq[x[l-1]]==1) ans++;

l--;

}

}

if(r<v[i].second){

while(r!=v[i].second){

freq[x[r+1]]++;

if(freq[x[r+1]]==1) ans++;

r++;

}

}

else if(r>v[i].second){

while(r!=v[i].second){

freq[x[r]]--;

if(!freq[x[r]]) ans--;

r--;

}

}

cout<<ans;

nl

}

}

return 0; }

**SPARSE TABLE:**

ll st[maxn][32];

void buildSparseTable(ll dt[], ll n){

for (ll i=0; i<n; i++) st[i][0]=dt[i];

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

for(ll i=0; (i+(1<<j)-1)<n; i++) st[i][j]=min(st[i][j-1], st[i+(1<<(j-1))][j-1]);

} }

ll query(ll l, ll r)

{

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

if (st[l][j]<=st[r-(1<<j)+1][j]) return st[l][j];

else return st[r-(1<<j)+1][j]; }

**DOUBLE HASH:**

ll base1 = 1e9+21, base2 = 1e9+181, mod=2000000063;

string s;

ll pw1[maxn], pw2[maxn], len;

void pw\_calc() {

pw1[0] = pw2[0] = 1;

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

pw1[i] = (pw1[i-1] \* base1) % mod;

pw2[i] = (pw2[i-1] \* base2) % mod;

} }

struct hash {

ll h1[maxn], h2[maxn];

void init() {

h1[0] = h2[0] = 0;

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

h1[i] = (h1[i-1] \* base1 + s[i]) % mod;

h2[i] = (h2[i-1] \* base2 + s[i]) % mod;

}

}

inline ll hashval(int l, int r) {

ll hsh1 = (h1[r] - h1[l-1] \* pw1[r-l+1]) % mod;

if(hsh1 < 0) hsh1 += mod;

ll hsh2 = (h2[r] - h2[l-1] \* pw2[r-l+1]) % mod;

if(hsh2 < 0) hsh2 += mod;

return (hsh1 << 32) | hsh2;

}

inline ll hashone(int l, int r) {

ll hsh1 = (h1[r] - h1[l-1] \* pw1[r-l+1]) % mod;

if(hsh1 < 0) hsh1 += mod;

return hsh1;

}

inline ll hashtwo(int l, int r) {

ll hsh2 = (h2[r] - h2[l-1] \* pw2[r-l+1]) % mod;

if(hsh2 < 0) hsh2 += mod;

return hsh2;

} } h;

**KMP:**

ll f[maxn];

vector <ll> ans;

void failure(string &y){

f[0]=0;

ll i=1;

ll len=0;

ll m=y.size();

while(i<m){

if(y[len]==y[i]){

len++;

f[i]=len;

i++;

}

else{

if(len!=0){

len=f[len-1];

}

else{

f[i]=0;

i++;

} } } }

void kmp(string &x, string &y){

failure(y);

ll n=x.size();

ll m=y.size();

ll i=0;

ll j=0;

while(i<n){

if(x[i]==y[j]){

i++;

j++;

if(j==m){

ans.pb(i-j);

j=f[j-1];

}

}

else{

if(j!=0){

j=f[j-1];

}

else{

i++;

} } } }

**TRIE:**

struct node{

bool end;

node\* next[26];

node(){

end=false;

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

next[i]=NULL;

}

}

}\*root;

void insert(string str, ll len){

node\* curr=root;

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

ll ch=str[i]-'a';

if(curr->next[ch]==NULL){

curr->next[ch]=new node();

}

curr=curr->next[ch];

}

curr->end=true; }

bool search(string str, ll len){

node\* curr=root;

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

ll ch=str[i]-'a';

if(curr->next[ch]==NULL){

return false;

}

curr=curr->next[ch];

}

return curr->end; }

void del(node\* curr){

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

if(curr->next[i]) del(curr->next[i]);

}

delete(curr); }

**SUFFIX ARRAY:**

string s;

ll n;

ll ra[maxn], tempra[maxn];

ll sa[maxn], tempsa[maxn];

ll lcp[maxn], plcp[maxn];

ll phi[maxn];

ll cnt[maxn];

void countingsort(ll k){

memset(cnt, 0, sizeof cnt);

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

if(i+k<n) cnt[ra[i+k]]++;

else cnt[0]++;

}

ll mx=max(n, 300ll);

ll sum=0;

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

ll temp=cnt[i];

cnt[i]=sum;

sum+=temp;

}

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

if(sa[i]+k<n) tempsa[cnt[ra[sa[i]+k]]++]=sa[i];

else tempsa[cnt[0]++]=sa[i];

}

for(ll i=0; i<n; i++) sa[i]=tempsa[i]; }

void suffixarray(){

for(ll i=0; i<n; i++) ra[i]=s[i];

for(ll i=0; i<n; i++) sa[i]=i;

for(ll k=1; k<n; k<<=1){

countingsort(k);

countingsort(0);

ll r=0;

tempra[sa[0]]=r;

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

if(ra[sa[i]]==ra[sa[i-1]] && ra[sa[i]+k]==ra[sa[i-1]+k]) tempra[sa[i]]=r;

else tempra[sa[i]]=++r;

}

for(ll i=0; i<n; i++) ra[i]=tempra[i];

if(ra[sa[n-1]]==n-1) break;

} }

void lcprefix(){

phi[sa[0]]=-1;

for(ll i=1; i<n; i++) phi[sa[i]]=sa[i-1];

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

if(phi[i]==-1){

plcp[i]=0;

continue;

}

while(s[i+len]==s[phi[i]+len]) len++;

plcp[i]=len;

len=max(len-1, 0ll);

}

for(ll i=0; i<n; i++) lcp[i]=plcp[sa[i]]; }

int main()

{

ll t;

t=1;

while(t--){

cin>>s;

s+='$';

n=s.size();

suffixarray();

lcprefix();

}

return 0; }

**LI CIAO TREE**

/\* \* Can be used to solve Convex Hull Trick problems

\* Adding line (y = mx+c) : O(logn)

\* Query : O(logn)

\* To find minimum, use f1(x) < f2(x), min in update & query.

\* To find maximum, use f1(x) > f2(x), max in update & query.

\* The line of tree[node] in range [lo, hi] represents

that this line gives the best result for point [x, y] range, where x <= mid <= y. Here, mid = (lo+hi) / 2. \*/

const ll sz = 1e5 + 10;

ll pnt[sz]; // 1 based indexing

// This array stores the points (in ascending order) needed to be queried.

struct Line{

ll m, c; // y = m\*x + c

inline ll f(ll x) {return m\*x + c;}

} tree[4\*sz];

bool exist[4\*sz];

// This will track the nodes updated.

// Thus query'll be optimized by not traversing

// the non-updated nodes.

void add(ll lo, ll hi, Line line, ll node){

// To initialize the tree, add this line:

// if(!exist[node]) tree[node] = {m, c};

// replace m, c with your desired values.

exist[node] = 1;

if(lo == hi){

if(line.f(pnt[lo]) < tree[node].f(pnt[lo]))

tree[node] = line;

return;}

ll mid = lo+hi >> 1;

bool left = line.f(pnt[lo]) < tree[node].f(pnt[lo]);

bool m = line.f(pnt[mid]) < tree[node].f(pnt[mid]);

if(m) swap(tree[node], line);

// if m == true, new line gives the best answer in point [x,y] range,

// where x <= mid <= y.

// left != m means line intersection between new line and

// tree[node]'s line is occurring in the left side of mid point

if(left != m) add(lo, mid, line, node<<1);

else add(mid+1, hi, line, node<<1|1);}

ll query(ll lo, ll hi, ll idx, ll node){

if(lo == hi)

return tree[node].f(pnt[idx]);

ll mid = lo+hi >> 1, ret = tree[node].f(pnt[idx]);

// We are not traversing the non-updated nodes. Thus query is optimized.

if(idx <= mid && exist[node<<1]) ret = min(ret, query(lo, mid, idx, node<<1));

else if(idx > mid && exist[node<<1|1]) ret = min(ret, query(mid+1, hi, idx, node<<1|1));

return ret;}

/\* \* To add a line: Call add(1, n, {m, c}, 1)

\* To query for a point x:

Let, x is in i index of pnt array.

Call query(1, n, i, 1)

\* [N.B: Query points are in the pnt array from

index 1 to index n] \*/