*/\**

***DSU***

*有一棵 n 个结点的以 1 号结点为根的有根树。*

*每个结点都有一个颜色，颜色是以编号表示的， i 号结点的颜色编号为 ci*

*如果一种颜色在以 x 为根的子树内出现次数最多，称其在以 x 为根的子树中占主导地位。显然，同一子树中可能有多种颜色占主导地位。*

*你的任务是对于每一个 i∈[1,n]，求出以 i 为根的子树中，占主导地位的颜色的编号和。*

*\*/*

#include <bits/stdc++.h>

**using** **namespace** std;

**typedef** **long** **long** ll;

**const** **int** N = 1e5 + 50;

**int** c[N], n, ac[N], x, cnt[N];

ll ans[N];

**int** head[N], etop;

**struct** Edge

{

**int** v, nxt;

}e[N<<1];

**void** add(**int** u, **int** v)

{

e[++etop].v = v;

e[etop].nxt = head[u];

head[u] = etop;

}

**int** siz[N], son[N];

**void** dfs1(**int** u, **int** fa)

{

siz[u] = 1;

**for** (**int** v, i = head[u]; i; i = e[i].nxt)

{

v = e[i].v;

**if** (v == fa) **continue**;

dfs1(v, u);

siz[u] += siz[v];

**if** (siz[v] > siz[son[u]])

son[u] = v;

}

}

**void** update(**int** u, **int** val)

{

cnt[c[u]] += val;

**if** (val == -1) **return** ;

**if** (cnt[c[u]] > ac[x])

{

ac[x] = cnt[c[u]];

ans[x] = c[u];

}

**else** **if** (cnt[c[u]] == ac[x])

ans[x] += c[u];

}

**void** add(**int** u, **int** fa, **int** val)

{

update(u, val);

**for** (**int** v, i = head[u]; i; i = e[i].nxt)

{

v = e[i].v;

**if** (v == fa) **continue**;

add(v, u, val);

}

}

**void** dfs2(**int** u, **int** fa, **int** keep)

{

**for** (**int** v, i = head[u]; i; i = e[i].nxt)

{

v = e[i].v;

**if** (v == fa || v == son[u]) **continue**;

dfs2(v, u, 0);

}

**if** (son[u])

{

dfs2(son[u], u, 1);

ans[u] = ans[son[u]];

ac[u] = ac[son[u]];

}

x = u;

**for** (**int** v, i = head[u]; i; i = e[i].nxt)

{

v = e[i].v;

**if** (v == fa || v == son[u]) **continue**;

add(v, u, 1);

}

update(u, 1);

**if** (!keep)

add(u, fa, -1);

}

**int** main()

{

scanf("%d", &n);

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

scanf("%d", c+i);

**for** (**int** u, v, i = 1; i < n; i++)

{

scanf("%d%d", &u, &v);

add(u, v); add(v, u);

}

dfs1(1, 0);

dfs2(1, 0, 0);

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

printf("%lld ", ans[i]);

**return** 0;

}

*/\**

*2*

*1 2*

*1 2*

*\*/*

***从0开始kmp***

#include <bits/stdc++.h>

**using** **namespace** std;

**const** **int** N = 2e6 + 50;

**int** nxt[N];

**int** getnxt(string s)

{

nxt[0] = -1;

**int** i = 0, j = -1;

**int** len = s.size();

**while** (i < len)

{

**if** (j == -1 || s[i] == s[j])

{

++i; ++j;

nxt[i] = j;

}

**else** j = nxt[j];

}

**return** j;

}

string s, ans;

**int** n, len;

**int** main()

{

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

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

cin >> n >> s;

ans = s;

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

{

cin >> s;

len = min(ans.size(), s.size());

string tmp = s + "!@#$%^&\*" + ans.substr(ans.size() - len);

**int** res = getnxt(tmp);

ans += s.substr(res); *//0 - res-1 相同*

}

cout<<ans<<endl;

**return** 0;

}

**广义SAM+线段树合并**

#include <bits/stdc++.h>

**using** **namespace** std;

**const** **int** N = 4e5 + 50, D = 20;

*//同一个子串出现在不同位置算多个*

*//询问第k个串几次出现在[l, r]串里*

**namespace** Seg

{

**int** lc[N \* D], rc[N \* D], sum[N \* D], top;

**void** add(**int** &u, **int** l, **int** r, **int** pos, **int** val)

{

**if** (!u) u = ++top;

sum[u] += val;

**if** (l == r) **return** ;

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

**if** (pos <= mid) add(lc[u], l, mid, pos, val);

**else** add(rc[u], mid + 1, r, pos, val);

}

**int** merge(**int** x, **int** y)

{

**if** (!x || !y) **return** x + y;

**int** u = ++top;

sum[u] = sum[x] + sum[y];

lc[u] = merge(lc[x], lc[y]);

rc[u] = merge(rc[x], rc[y]);

**return** u;

}

**int** query(**int** u, **int** st, **int** ed, **int** l, **int** r)

{

**if** (!u) **return** 0;

**if** (st == l && ed == r) **return** sum[u];

**int** ret = 0, mid = l + r >> 1;

**if** (st <= mid) ret += query(lc[u], st, min(mid, ed), l, mid);

**if** (ed > mid) ret += query(rc[u], max(st, mid + 1), ed, mid + 1, r);

**return** ret;

}

}

**int** ch[N][26], mx[N], fa[N], last, top = 1, t[N], rk[N];

**int** add(**int** last, **int** c)

{

**if** (ch[last][c] && mx[last] + 1 == mx[ch[last][c]])

**return** ch[last][c];

**int** p = last, np = ++top;

mx[np] = mx[p] + 1;

**for** (; p && !ch[p][c]; p = fa[p]) ch[p][c] = np;

**if** (!p) fa[np] = 1;

**else**

{

**int** q = ch[p][c];

**if** (mx[q] == mx[p] + 1) fa[np] = q;

**else**

{

**int** nq = p == last ? np : ++top;

mx[nq] = mx[p] + 1;

memcpy(ch[nq], ch[q], **sizeof**(ch[q]));

**for** (; p && ch[p][c] == q; p = fa[p])

ch[p][c] = nq;

fa[np] = nq;

fa[nq] = fa[q];

fa[q] = nq;

}

}

**return** np;

}

**int** id[N], rt[N];

**char** s[N];

**void** work()

{

**for** (**int** i = 1; i <= top; i++) ++t[mx[i]];

**for** (**int** i = 1; i <= top; i++) t[i] += t[i - 1];

**for** (**int** i = top; i >= 1; i--) rk[t[mx[i]]--] = i;

**for** (**int** i = top; i >= 2; i--)

{

**int** u = rk[i];

rt[fa[u]] = Seg::merge(rt[fa[u]], rt[u]);

}

}

**int** main()

{

**int** n, q;

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

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

{

scanf("%s", s);

last = 1;

**for** (**int** len = strlen(s), j = 0; j < len; j++)

{

last = add(last, s[j] - 'a');

Seg::add(rt[last], 1, n, i, 1);

}

id[i] = last;

}

work();

**for** (**int** l, r, k; q--;)

{

scanf("%d%d%d", &l, &r, &k);

printf("%d**\n**", Seg::query(rt[id[k]], l, r, 1, n));

}

**return** 0;

}

*/\**

*5 1*

*a*

*ab*

*abab*

*ababab*

*b*

*1 5 1*

*\*/*

***珂朵莉树***

#include <bits/stdc++.h>

#define IT set<Node>::iterator

**using** **namespace** std;

**typedef** **long** **long** ll;

**const** ll P = 1e9 + 7;

**const** **int** N = 1e5 + 50;

**struct** Node

{

**int** l, r;

**mutable** ll v;

Node (**int** L, **int** R = -1, ll V=0):l(L), r(R), v(V) {}

**bool** **operator** <(**const** Node &t) **const**

{

**return** l < t.l;

}

};

set <Node> s;

IT split(**int** pos) *//将pos位置分为[l, pos - 1 ]和 [pos, r]*

{

IT it = s.lower\_bound(Node(pos));

**if** (it != s.end() && it->l == pos) **return** it;

--it;

**int** L = it->l, R= it->r;

ll V = it->v;

s.erase(it);

s.insert(Node(L, pos - 1, V));

**return** s.insert(Node(pos, R, V)).first;

}

**void** assign(**int** l, **int** r, ll val=0) *//区间推平*

{

IT itr = split(r + 1), itl = split(l);

s.erase(itl, itr); *//删除这个区间内所有元素*

s.insert(Node(l, r, val));

}

**void** add(**int** l, **int** r, ll val) *//区间加法*

{

IT itr = split(r + 1), itl = split(l);

**for** (; itl != itr; ++itl) itl->v += val;

}

ll rk(**int** l, **int** r, **int** k) *//区间第k小*

{

vector < pair<ll, **int**> > vec; *// 数值与出现次数*

IT itr = split(r + 1), itl = split(l);

**for** (; itl != itr; ++itl)

vec.push\_back(pair<ll, **int**>(itl->v, itl->r - itl->l + 1));

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

**for** (**auto** it = vec.begin(); it != vec.end(); ++it)

{

k -= it->second;

**if** (k <= 0) **return** it->first;

}

}

ll power(ll x, ll k, ll mod)

{

ll ret = 1;

**for** (; k; k >>= 1, x = x \* x % mod)

**if** (k & 1) ret = ret \* x % mod;

**return** ret;

}

ll sum(**int** l, **int** r, ll ex, ll mod)

{

IT itr = split(r + 1), itl = split(l);

ll ret = 0;

**for** (; itl != itr; ++itl)

{

ll siz = itl->r - itl->l + 1;

ret = (ret + siz \* power(itl->v % mod, ex, mod)) % mod;

}

**return** ret;

}

**int** n, m;

ll seed, vmax, a[N];

ll rnd()

{

ll ret = seed;

seed = (seed \* 7 + 13) % P;

**return** ret;

}

**int** main()

{

scanf("%d%d%lld%lld", &n, &m, &seed, &vmax);

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

{

a[i] = rnd() % vmax + 1;

s.insert(Node(i, i, a[i]));

}

**for** (**int** op, l, r; m--;)

{

ll x, y;

op = rnd() % 4 + 1;

l = rnd() % n + 1;

r = rnd() % n + 1;

**if** (l > r) swap(l, r);

**if** (op == 3) x = (rnd() % (r - l + 1)) + 1;

**else** x = rnd() % vmax + 1;

**if** (op == 4) y = rnd() % vmax + 1;

**if** (op == 1) add(l, r, x);

**if** (op == 2) assign(l, r, x);

**if** (op == 3) printf("%lld**\n**", rk(l, r, x));

**if** (op == 4) printf("%lld**\n**", sum(l, r, x, y));

}

**return** 0;

}

**AC自动机不路径压缩，保留last上一个出现的是串的结尾的节点**

**void** getfail()

{

queue <**int**> q;

**for** (**int** u, i = 0; i < 26; i++)

{

u = ch[0][i];

**if** (u)

{

q.push(u);

fail[u] = last[u] = 0;

}

}

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

{

**int** cur = q.front(); q.pop();

**for** (**int** u, v, i = 0; i < 26; i++)

**if** (ch[cur][i])

{

u = ch[cur][i];

q.push(u);

v = fail[cur];

**while** (v && !ch[v][i]) v = fail[v];

fail[u] = ch[v][i];

last[u] = val[fail[u]] ? fail[u] : last[fail[u]];

}

}

}