目录

[杂 1](#_Toc119172932)

[快读 1](#_Toc119172933)

[int128 1](#_Toc119172934)

[最长上升子序列 LIS 2](#_Toc119172935)

[miller\_rabin大质数判断 2](#_Toc119172936)

[康托展开 3](#_Toc119172937)

[组合数预处理 4](#_Toc119172938)

[树上背包 6](#_Toc119172939)

[枚举二进制下k位1的数字 7](#_Toc119172940)

[区间合并、区间求交 7](#_Toc119172941)

[SPFA 判负环 8](#_Toc119172942)

[数学 8](#_Toc119172943)

[Miller Rabin & Pollard Rho 8](#_Toc119172944)

[字符串 9](#_Toc119172945)

[可持久化kmp 9](#_Toc119172946)

[字符串哈希 9](#_Toc119172947)

[KMP 10](#_Toc119172948)

[马拉车 11](#_Toc119172949)

[PAM 12](#_Toc119172950)

[Z函数 扩展 KMP 13](#_Toc119172951)

[数据结构 15](#_Toc119172952)

[树状数组 15](#_Toc119172953)

[ST表 15](#_Toc119172954)

[Trie树 16](#_Toc119172955)

[并查集 16](#_Toc119172956)

[莫队 17](#_Toc119172957)

[莫队求 MEX 19](#_Toc119172958)

[欧拉序求 LCA 21](#_Toc119172959)

[染色法判二分图 21](#_Toc119172960)

[树链剖分 22](#_Toc119172961)

[树剖求 LCA 23](#_Toc119172962)

[虚树 24](#_Toc119172963)

[ODT 老司机树 26](#_Toc119172964)

[计算几何 27](#_Toc119172965)

[凸多边形判定 27](#_Toc119172966)

## 杂

### 快读

// #define int \_\_int128  
namespace io {  
 constexpr int BUFFER\_SIZE = 1 << 16;  
 char buffer[BUFFER\_SIZE], \*head, \*tail;  
  
 char get\_char() {  
 if (head == tail) {  
 int l = (int) fread(buffer, 1, BUFFER\_SIZE, stdin);  
 tail = (head = buffer) + l;  
 }  
 return \*head++;  
 }  
  
 long long read() {  
 long long x = 0, f = 1;  
 char c = get\_char();  
 for (; !isdigit(c); c = get\_char())  
 if (c == '-') f = -1;  
 for (; isdigit(c); c = get\_char()) x = x \* 10 + c - '0';  
 return x \* f;  
 }  
  
 void print(int x) {  
 static int sta[1997];  
 int top = 0;  
 do {  
 sta[top++] = x % 10, x /= 10;  
 } while (x);  
 while (top) putchar(sta[--top] + 48); // 48 是 '0'  
 }  
  
 void println(int x) { print(x), putchar('\n'); }  
} // namespace io  
using namespace io;

### int128

// #define int \_\_int128  
  
namespace io {  
 constexpr int BUFFER\_SIZE = 1 << 16;  
 char buffer[BUFFER\_SIZE], \*head, \*tail;  
  
 char get\_char() {  
 if (head == tail) {  
 int l = (int) fread(buffer, 1, BUFFER\_SIZE, stdin);  
 tail = (head = buffer) + l;  
 }  
 return \*head++;  
 }  
  
 \_\_int128 read() {  
 \_\_int128 x = 0, f = 1;  
 char c = get\_char();  
 for (; !isdigit(c); c = get\_char())  
 if (c == '-') f = -1;  
 for (; isdigit(c); c = get\_char()) x = x \* 10 + c - '0';  
 return x \* f;  
 }  
  
 void print(\_\_int128 x) {  
 if(x < 0) x = -x, putchar('-');  
 constexpr long long P = 1E18;  
 if(x < P) {  
 cout << ll(x) << " ";  
 } else {  
 cout << ll(x / P) << setw(18) << setfill('0') << ll(x % P) << " ";  
 }  
 }  
  
 void println(\_\_int128 x) { print(x), putchar('\n'); }  
  
 \_\_int128 sqrt(\_\_int128 x) {  
 \_\_int128 l = 0, r = (\_\_int128)10000000000000;   
 while(l < r) {  
 \_\_int128 mid = (l + r + 1) / 2;  
 if(mid \* mid > x) r = mid - 1;  
 else l = mid;  
 }  
 return l;  
 }  
}

### 最长上升子序列 LIS

int a[N], stk[N];  
int tt;  
  
int main() {  
 int n;  
 scanf("%d", &n);  
 for (int i = 0; i < n; i++) scanf("%d", &a[i]);  
 tt = 0;  
 // memset(stk, 0, sizeof stk);  
 int ans = -1e9 - 10;  
 for (int i = 0; i < n; i++) {  
 int p = lower\_bound(stk + 1, stk + 1 + tt, a[i]) - stk;  
 ans = max(ans, p);  
 stk[p] = a[i];  
 tt = max(tt, p);  
 }  
  
 printf("%d\n", ans);  
}

### miller\_rabin大质数判断

constexpr int mod = 998244353;  
ll mul(ll a, ll b, ll mod) { return a \* b % mod; }  
ll power(ll a, ll r, ll mod) {  
 ll res = 1;  
 for (; r; r >>= 1, a = mul(a, a, mod))  
 if (r & 1) res = mul(res, a, mod);  
 return res;  
}  
ll p[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};  
bool miller\_rabin(ll n) {  
 if (n == 1) return false;  
 if (n == 2) return true;  
 if (not(n & 1)) return false;  
 ll d = n - 1, r = 0;  
 for (; not(d & 1); d >>= 1) r += 1;  
 bool res = true;  
 for (int i = 0; i < 9 and p[i] < n and res; i += 1) {  
 ll x = power(p[i], d, n);  
 if (x == 1 or x == n - 1) continue;  
 for (int j = 1; j < r; j += 1) {  
 x = mul(x, x, n);  
 if (x == n - 1) break;  
 }  
 if (x != n - 1) res = false;  
 }  
 return res;  
};

### 康托展开

#include <bits/stdc++.h>  
using namespace std;  
using ll = long long;  
  
/\*  
康托展开用于求一个排列在所有 1 ~ n的排列间的字典序排名。  
其实康托展开的原理很简单。设有排列p= a102 ...an，那么对任意字典序比p小的排列，一  
定存在i，使得其前i-1(1<i<n)位与p对应位相同，第i位比p;小，后续位随意。于  
是对于任意i，满足条件的排列数就是从后n- i + 1位中选一个比ai小的数、并将剩下  
n-i个数任意排列的方案数,即为A\_i \* (n-i)!(Ai表示ai后面比a;小的数的个数)。  
遍历i即得总方案数>: \sum\_{i=1}^{n-1} Ai \* (n- i)!,再加1即为排名。  
  
\*/  
constexpr int MAXN = 1E5 + 10;  
  
namespace Nanfeng\_Ktzk\_brute {  
 ll fact[MAXN] = {1}, P[MAXN], A[MAXN]; // fact需要在外部初始化  
 ll cantor(int P[], int n) // 这里传入的P是1-index数组  
 {  
 ll ans = 1;  
 for (int i = 1; i <= n; i++)  
 for (int j = i + 1; j <= n; j++)  
 if (P[j] < P[i]) A[i]++;  
  
 for (int i = 1; i < n; i++) ans += A[i] \* fact[n - i];  
 return ans;  
 }  
}  
  
namespace Nanfeng\_Ktzk\_bit {  
 #define MOD 1000000007  
 ll fact[MAXN] = {1}, P[MAXN], A[MAXN], tree[MAXN];  
 ll lowbit(ll x) { return x & -x; }  
 ll query(ll x)  
 {  
 ll ans = 0;  
 for (int i = x; i >= 1; i -= lowbit(i))  
 ans += tree[i];  
 return ans;  
 }  
 void update(ll x, ll d)  
 {  
 for (int i = x; i < MAXN; i += lowbit(i))  
 tree[i] += d;  
 }  
 ll cantor(int P[], int n)  
 {  
 ll ans = 1;  
 for (int i = n; i >= 1; i--)  
 {  
 A[i] = query(P[i]);  
 update(P[i], 1);  
 }  
 for (int i = 1; i < n; i++)  
 ans = (ans + A[i] \* fact[n - i]) % MOD;  
 return ans;  
 }  
}

### 组合数预处理

const int MOD = 998244353;  
  
inline int mod(int x) {return x >= MOD ? x - MOD : x;}  
  
inline int ksm(int a, int b) {  
 int ret = 1; a = mod(a);  
 for(; b; b >>= 1, a = 1LL \* a \* a % MOD) if(b & 1) ret = 1LL \* ret \* a % MOD;  
 return ret;  
}  
  
template<int MOD>   
struct modint {  
 int x;  
 modint() {x = 0; }  
 modint(int y) {x = y;}  
 inline modint inv() const { return modint{ksm(x, MOD - 2)}; }  
 explicit inline operator int() { return x; }  
 friend inline modint operator + (const modint &a, const modint& b) { return modint(mod(a.x + b.x)); }  
 friend inline modint operator - (const modint &a, const modint& b) { return modint(mod(a.x - b.x + MOD)); }  
 friend inline modint operator \* (const modint &a, const modint& b) { return modint(1ll \* a.x \* b.x % MOD); }  
 friend inline modint operator / (const modint &a, const modint& b) { return modint(1ll \* a.x \* b.inv().x % MOD); }  
 friend inline modint operator + (const modint &a, const int& b) { return modint(mod(a.x + b)); }  
 friend inline modint operator - (const modint &a, const int& b) { return modint(mod(a.x - b + MOD)); }  
 friend inline modint operator \* (const modint &a, const int& b) { return modint(1ll \* a.x \* b % MOD); }  
 friend inline modint operator / (const modint &a, const int& b) { return modint(1ll \* a.x \* ksm(b, MOD - 2) % MOD); }   
 friend inline modint operator - (const modint &a) { return modint(mod(MOD - a.x)); }  
 friend inline modint& operator += (modint &a, const modint& b) { return a = a + b; }  
 friend inline modint& operator -= (modint &a, const modint& b) { return a = a - b; }  
 friend inline modint& operator \*= (modint &a, const modint& b) { return a = a \* b; }  
 friend inline modint& operator /= (modint &a, const modint& b) { return a = a / b; }  
 friend inline modint& operator += (modint &a, const int& b) { return a = a + b; }  
 friend inline modint& operator -= (modint &a, const int& b) { return a = a - b; }  
 friend inline modint& operator \*= (modint &a, const int& b) { return a = a \* b; }  
 friend inline modint& operator /= (modint &a, const int& b) { return a = a / b; }  
 friend auto &operator >> (istream &i, modint &a) {return i >> a.x; }  
 friend auto &operator << (ostream &o, const modint &z) { return o << z.x; }  
 inline bool operator == (const modint &b) { return x == b.x; }  
 inline bool operator == (const int &b) { return x == b; }  
 inline bool operator != (const modint &b) { return x != b.x; }  
 inline bool operator != (const int &b) { return x != b; }  
 inline bool operator < (const modint &a) { return x < a.x; }  
 inline bool operator < (const int &b) { return x < b; }  
 inline bool operator <= (const modint &a) { return x <= a.x; }  
 inline bool operator <= (const int &b) { return x <= b; }  
 inline bool operator > (const modint &a) { return x > a.x; }  
 inline bool operator > (const int &a) { return x > a; }  
 inline bool operator >= (const modint &a) { return x >= a.x; }  
 inline bool operator >= (const int &a) { return x >= a; }  
 operator int() const {  
 return x;  
 }  
 // inline void  
};  
  
typedef modint<MOD> mint;  
  
inline mint ksm(mint a, int b, mint ret = 1) {  
 for(; b; b >>= 1, a = a \* a ) if(b & 1) ret = ret \* a ;  
 return ret;  
}  
  
const int N = 2e5 + 10;  
  
mint fact[N + 1], infact[N + 1], inv[N + 1];  
  
void init() {  
 fact[0] = 1; for(int i = 1; i <= N; ++ i ) { fact[i] = fact[i - 1] \* i; }  
 infact[N] = ksm(fact[N], MOD - 2); for(int i = N - 1; i >= 0; -- i ) infact[i] = infact[i + 1] \* (i + 1);  
 inv[0] = inv[1] = 1; for(int i = 2; i <= N; ++ i) inv[i] = inv[MOD % i] \* (MOD - MOD / i);  
}  
  
mint C(int a, int b) {  
 if (a < b) return 0;  
 return fact[a] \* infact[b] \* infact[a - b];  
}

### 树上背包

int main() {  
 cin.tie(nullptr)->sync\_with\_stdio(false);  
 int n, m;  
 cin >> n >> m;  
 // n个物品，体积为m  
 vector<vector<int>> son(n + 2);  
  
 for (auto i = 0; i < n; i++) {  
 int x;  
 cin >> x; //买 i 必须先买 x  
 son[x].push\_back(i + 1);  
 }  
  
 vector<int> v(n + 1), w(n + 1);  
  
 for (int i = 1; i <= n; i++) {  
 cin >> v[i]; //体积  
 }  
  
 for (int i = 1; i <= n; i++) {  
 cin >> w[i]; //价值  
 }  
  
 // dfs序  
 vector<int> l(n + 2), r(n + 2), id(n + 2);  
 int tot = 0;  
  
 function<void(int)> dfs = [&](int u) {  
 l[u] = ++tot;  
 id[tot] = u;  
 for (auto &v : son[u]) {  
 dfs(v);  
 }  
 r[u] = tot;  
 };  
  
 dfs(0);  
  
 //背包  
 vector<vector<int>> f(tot + 3, vector<int>(m + 1));  
  
 for (int i = tot; i >= 1; i--) {  
 int u = id[i];  
 for (int j = 0; j <= m; j++) {  
 f[i][j] = f[r[u] + 1][j];  
 if (j >= v[u]) {  
 chkmax(f[i][j], f[i + 1][j - v[u]] + w[u]);  
 }  
 }  
 }  
  
 cout << f[1][m];  
  
 return 0;  
}

### 枚举二进制下k位1的数字

void show\_bit(int x) {  
 for (int i = 8; i >= 0; i--) putchar('0' + (x >> i & 1));  
 puts("");  
}  
  
int main() {  
 int n = 10, k = 4;  
 for (int cur = ((1 << n - k) - 1) ^ ((1 << n) - 1); ~cur;) {  
 show\_bit(cur);  
 if (int a = \_\_builtin\_ctz(~cur); a < k) {  
 int b = \_\_builtin\_ctz(cur ^= (1 << a) - 1);  
 cur ^= ((1 << b - a - 1) - 1) ^ ((1 << b + 1) - 1);  
 } else  
 cur = -1;  
 }  
}

### 区间合并、区间求交

void merge(vector<pair<int, int> > &segs) { //区间合并  
 if (segs.empty()) return;  
 vector<pair<int, int> > res;  
 sort(segs.begin(), segs.end());  
 int st = segs[0].first, ed = segs[0].second;  
 for (auto seg : segs) {  
 if (seg.first > ed) {  
 res.emplace\_back(st, ed);  
 st = seg.first, ed = seg.second;  
 } else  
 ed = max(ed, seg.second);  
 }  
 res.emplace\_back(st, ed);  
 segs = res;  
}  
  
vector<pair<int, int> > intersection(vector<pair<int, int> > a,  
 vector<pair<int, int> > b) {  
 //求区间交  
 vector<pair<int, int> > res;  
 int i = 0, j = 0;  
 while (i < a.size() && j < b.size()) {  
 int l = max(a[i].first, b[j].first);  
 int r = min(a[i].second, b[j].second);  
 if (l <= r) res.emplace\_back(l, r);  
 if (a[i].second < b[j].second)  
 i++;  
 else  
 j++;  
 }  
 return res;  
}

### SPFA 判负环

bool spfa() {  
 memset(st, 0, sizeof st);  
 memset(cnt, 0, sizeof cnt);  
 queue<int> q;  
 for (int i = 1; i <= n; i++) {  
 q.push(i);  
 st[i] = 1;  
 }  
 while (q.size()) {  
 auto t = q.front();  
 q.pop();  
 st[t] = 0;  
 for (int i = h[t]; ~i; i = ne[i]) {  
 int j = e[i];  
 if (dist[j] > dist[t] + w[i]) {  
 dist[j] = dist[t] + w[i];  
 cnt[j] = cnt[t] + 1;  
 if (cnt[j] >= n) return 1;  
 if (!st[j]) {  
 q.push(j);  
 st[j] = 1;  
 }  
 }  
 }  
 }  
 return 0;  
}

## 数学

### Miller Rabin & Pollard Rho

random\_device rd;  
mt19937\_64 gen(rd());  
uniform\_int\_distribution<ll> dis(0);  
  
ll qPow(ll b, ll p, ll mod){  
 ll ret = 1;  
 while(p){  
 if(p & 1) ret = (\_\_int128)ret \* b % mod;  
 b = (\_\_int128)b \* b % mod;  
 p >>= 1;  
 }  
 return ret;  
}  
bool MillerRabin(ll p){  
 if(p < 2) return false;  
 if(p < 4) return true;  
 ll d = p - 1;  
 int r = 0;  
 while((d & 1) == 0) ++r, d >>= 1;  
 for(ll k = 0; k < 10; ++k){  
 ll rv = dis(gen) % (p - 2) + 2;   
 ll x = qPow(rv, d, p);  
 if(x == 1 or x == p - 1) continue;  
 for(int i = 0; i < r - 1; ++i){  
 x = (\_\_int128) x \* x % p;  
 if(x == p - 1) break;  
 }  
 if(x != p - 1) return false;  
 }  
  
 return true;  
}  
ll PollardRho(ll n){ // 找到一个 n 的因子  
 // 可能不是质因子，也可能是它本身(可以while(return != n) )  
 ll c = rand() % (n - 1) + 1;  
 ll s = 0, t = 0;  
 for(ll goal = 1, val = 1; ; goal \*= 2, s = t, val = 1){  
 for(ll step = 1; step <= goal; ++step){  
 t = ((\_\_int128) t \* t + c) % n;  
 val = (\_\_int128)val \* abs(t - s) % n;  
 if(step % 127 == 0){  
 ll d = gcd(val, n);  
 if(d > 1) return d;  
 }  
 }  
 ll d = gcd(val, n);  
 if(d > 1) return d;  
 }  
}

## 字符串

### 可持久化kmp

int ne[N], nextval[N], last;  
char s[N], t[N];  
  
void add(char x) { //将x加进s的末尾，并更新ne数组的值,利用nextval加快匹配过程，导致每次匹配最多log(n)次  
 int j = last;  
 while (j && s[ne[j] + 1] != x) j = nextval[j]; //kmp过程  
 //把当前字符加进 s   
 s[++last] = x, j = ne[j] + 1; //last为字符串 s 的长度  
 if (last == 1) {  
 ne[1] = nextval[1] = 0;  
 } else if (s[j] == x) {  
 ne[last] = j;  
 if (s[ne[j] + 1] == s[j + 1]) {  
 nextval[last] = nextval[j];  
 } else {  
 nextval[last] = j;  
 }  
 } else {  
 ne[last] = nextval[last] = 0;  
 }  
}

### 字符串哈希

// mt19937\_64 rnd(chrono::steady\_clock::now().time\_since\_epoch().count());  
mt19937 rnd(random\_device{}());   
  
using ull = unsigned long long;  
using HH = pair<ull, ull>;  
  
constexpr HH mod = {1000000009, 1000000007};  
  
HH operator\*(const HH &a, const HH &b) {  
 return {(a.first \* b.first) % mod.first, (a.second \* b.second) % mod.second};  
}  
   
HH operator+(const HH &a, const HH &b) {  
 return {(a.first + b.first) % mod.first, (a.second + b.second) % mod.second};  
}  
   
HH operator-(const HH &a, const HH &b) {  
 return {(a.first - b.first + mod.first) % mod.first, (a.second - b.second + mod.second) % mod.second};  
}  
   
HH operator\*(const HH &a, const ull &b) {  
 return {(a.first \* b) % mod.first, (a.second \* b) % mod.second};  
}  
   
struct Hash {  
 string s; int n;  
 vector<HH> base, hsh1, hsh2; // 注意空字符串  
 Hash(string p): s(p), n(p.size()), base(n + 1), hsh1(n + 1), hsh2(n + 1) {  
 base[0] = hsh1[0] = hsh2[0] = {1, 1};  
 base[1] = {rnd() % mod.first, rnd() % mod.second};  
 for(int i = 1; i <= n; i ++ ) {  
 base[i] = base[i - 1] \* base[1];  
 hsh1[i] = hsh1[i - 1] \* base[1] + base[0] \* (s[i - 1] - 'a' + 1);  
 hsh2[i] = hsh2[i - 1] \* base[1] + base[0] \* (s[n - i] - 'a' + 1);  
 }  
 }  
  
 HH get1(int l, int r) { //返回 l 到 r 的哈希值  
 //注意 l > r, 默认 1 <= l <= r <= n  
 return hsh1[r] - (hsh1[l - 1] \* base[r - l + 1]);  
 }  
   
 HH get2(int l, int r) { //返回 r 到 l 的哈希值  
 //注意 l > r, 默认 1 <= l <= r <= n  
 l = n - l + 1, r = n - r + 1;  
 swap(l, r);  
 return hsh2[r] - (hsh2[l - 1] \* base[r - l + 1]);  
 }  
  
};

### KMP

#include <iostream>  
  
using namespace std;  
  
const int N = 100010, M = 1000010;  
  
int n, m;  
int ne[N];  
char s[M], p[N];  
// 1 index  
void solve1() {  
 cin >> n >> p + 1 >> m >> s + 1;  
  
 for (int i = 2, j = 0; i <= n; i++) {  
 while (j && p[i] != p[j + 1]) j = ne[j];  
 if (p[i] == p[j + 1]) j++;  
 ne[i] = j;  
 }  
  
 for (int i = 1, j = 0; i <= m; i++) {  
 while (j && s[i] != p[j + 1]) j = ne[j];  
 if (s[i] == p[j + 1]) j++;  
 if (j == n) {  
 printf("%d ", i - n);  
 j = ne[j];  
 }  
 }  
}  
// 0 index  
void solve2() {  
 cin >> m >> p >> n >> s;  
  
 ne[0] = -1;  
 for (int i = 1, j = -1; i < m; i++) {  
 while (j >= 0 && p[j + 1] != p[i]) j = ne[j];  
 if (p[j + 1] == p[i]) j++;  
 ne[i] = j;  
 }  
  
 for (int i = 0, j = -1; i < n; i++) {  
 while (j != -1 && s[i] != p[j + 1]) j = ne[j];  
 if (s[i] == p[j + 1]) j++;  
 if (j == m - 1) {  
 cout << i - j << ' ';  
 j = ne[j];  
 }  
 }  
}

### 马拉车

// https://www.acwing.com/problem/content/description/141/  
#include <bits/stdc++.h>  
using namespace std;  
  
constexpr int N = 2000010;  
  
int n, m, Case;  
char s[N], str[N];  
int p[N];  
  
int manacher() {  
 int rt = 0, mid = 0;  
 int res = 0;  
 for (int i = 1; i <= m; i++) {  
 p[i] = i < rt ? min(p[2 \* mid - i], rt - i) : 1;  
 while (str[i + p[i]] == str[i - p[i]]) ++p[i];  
 if (i + p[i] > rt) {  
 rt = i + p[i];  
 mid = i;  
 }  
 // p[i] - 1即为最长的回文子串长度  
 res = max(res, p[i] - 1);  
 }  
 return res;  
}  
  
int main() {  
 str[0] = '!', str[1] = '#';  
 //原串为 0 index ，对应在 str 中的下标为 (i + 1) \* 2  
 while (scanf("%s", s), s[0] != 'E') {  
 n = strlen(s);  
 for (int i = 0; i < n; i++) {  
 str[i \* 2 + 2] = s[i];  
 str[i \* 2 + 3] = '#';  
 }  
 m = n \* 2 + 1;  
 str[m + 1] = '@';  
 printf("Case %d: %d\n", ++Case, manacher());  
 }  
 return 0;  
}

### PAM

#include <bits/stdc++.h>  
  
using namespace std;  
  
struct PAM {  
 int n, tot, last, cur;  
 // int cnt[maxn], len[maxn], fail[maxn], son[maxn][27];  
 vector<int> cnt, len, fail;  
 vector<vector<int> > son;  
 string s, t;  
 //动态开点  
 int new\_node(int x) {  
 len[tot] = x;  
 cnt[tot] = 0;  
 // cout << tot << "\n";  
 return tot++;  
 }  
 int get\_fail(int x, int n) {  
 while (s[n - len[x] - 1] != s[n]) x = fail[x];  
 return x;  
 }  
 void init(string &ss, int nn) {  
 // scanf("%s", s + 1);  
 s = " " + ss; // t 用来哈希, s 用来 PAM  
 t = ss;  
 n = nn;  
 cnt.resize(n + 10);  
 len.resize(n + 10);  
 fail.resize(n + 10);  
 son.resize(n + 10);  
 for (int i = 0; i < n + 10; i++) son[i].resize(27);  
 tot = 0;  
 new\_node(0);  
 new\_node(-1);  
 fail[0] = 1;  
 last = 0;  
 }  
 void build() {  
 // Hash H(t); // 用来哈希  
 for (int i = 1; i <= n; i++) {  
 int x = s[i] - 'a';  
 cur = get\_fail(last, i);  
 if (!son[cur][x]) {  
 int nw = new\_node(len[cur] + 2);  
 fail[nw] = son[get\_fail(fail[cur], i)][x];  
 son[cur][x] = nw;  
 }  
 last = son[cur][x];  
 cnt[last]++;  
 // printf("%c %d %d\n", s[i], i, len[last];  
 //以 i 结尾，并且长度为 len[last] 的回文串  
 // 回文串 [i - len[last] + 1, i]  
 }  
 }  
  
 void gen() {  
 //由于每个cnt[last]都为1(or  
 //0)，如果要求每个回文串在整个字符串中出现的次数，需要求一下前缀和  
 for (int i = tot - 1; i >= 0; i--) {  
 cnt[fail[i]] += cnt[i];  
 }  
 // 此时 cnt[idx] 表示 idx 这个节点所代表的回文串的个数  
 // 如果还要知道每个 idx 所表示的字符串是什么，  
 // 可以使用 map[Hash.get(i - len[last] + 1, i)] = idx，通过遍历然后 idx =  
 // cnt[idx] 即可  
 }  
  
} pam;  
  
int main() {  
 ios::sync\_with\_stdio(false);  
 cin.tie(nullptr);  
 string s;  
 cin >> s;  
 pam.init(s, (int)s.size());  
 pam.build();  
  
 return 0;  
}

### Z函数 扩展 KMP

// https://www.luogu.com.cn/problem/P5410  
  
#include <bits/stdc++.h>  
using namespace std;  
using ll = long long;  
  
#ifdef LOCAL  
 #include <debugger>  
#else  
 #define debug(...) 42  
#endif  
  
template <typename T> void chkmax(T &x, T y) { x = max(x, y); }  
template <typename T> void chkmin(T &x, T y) { x = min(x, y); }  
  
constexpr int N = 2E7 + 10;  
  
int n, m;  
char s[N], p[N];  
int z[N], ext[N];  
// LCP 是最长公共前缀  
// z[i] 表示 模板串 p[i, strlen(p) - 1] 与 p 的 LCP  
// ext[i] 表示 匹配串 s[i, strlen(s) - 1] 与 p 的 LCP  
  
void getZ() {  
 z[0] = m; int now = 0;   
 while (now + 1 < m && p[now] == p[now + 1]) now ++;  
 z[1] = now;  
 for (int i = 2, j = 1; i < m; i ++ ) {  
 if (i + z[i - j] < j + z[j]) {  
 z[i] = z[i - j];  
 } else {  
 now = max(j + z[j] - i, 0);  
 while (now + i < m && p[now] == p[now + i]) ++ now;  
 z[i] = now;  
 j = i;  
 }  
 }  
}  
  
void getexkmp() {  
 int now = 0;  
 while (now < m && now < n && p[now] == s[now]) ++ now;  
 ext[0] = now;  
 for (int i = 1, j = 0; i < n; i ++ ) {  
 if (i + z[i - j] < j + ext[j]) {  
 ext[i] = z[i - j];  
 } else {  
 now = max(j + ext[j] - i, 0); // pretect i too large  
 while (now < m && now + i < n && p[now] == s[now + i]) ++ now;  
 ext[i] = now;  
 j = i;  
 }  
 }  
}  
  
  
  
void solve() {  
 scanf("%s%s", s, p);  
 n = strlen(s), m = strlen(p);  
 getZ(); getexkmp();  
 long long ans0 = 0, ans1 = 0;  
 for (int i = 0; i < m; i ++ ) {  
 ans0 ^= 1ll \* (i + 1) \* (z[i] + 1);  
 }  
 for (int i = 0; i < n; i ++ ) {  
 ans1 ^= 1ll \* (i + 1) \* (ext[i] + 1);  
 }  
 printf("%lld\n%lld\n", ans0, ans1);  
}  
int main() {  
 // freopen("1.in", "r", stdin);  
 solve();  
 return 0;  
}

## 数据结构

### 树状数组

template <typename T>  
class fenwick {  
 public:  
 vector<T> fenw;  
 int n;  
  
 fenwick(int \_n) : n(\_n) { fenw.resize(n); }  
  
 void modify(int x, T v) {  
 while (x < n) {  
 fenw[x] += v;  
 x |= (x + 1);  
 }  
 }  
  
 T get(int x) {  
 T v{};  
 while (x >= 0) {  
 v += fenw[x];  
 x = (x & (x + 1)) - 1;  
 }  
 return v;  
 }  
  
 T get(int l, int r) { return get(r) - get(l - 1); }  
};

### ST表

//使用的时候注意判断如果 多个最值 取左边下标还是右边下标  
//分别对应 >= > <= <  
template<typename T>   
class ST{  
 public:  
 ST(vector<T> a, int \_n) : a(a), n(\_n) { // cope with in [0,n-1]   
 lg.resize(n + 1); lg[1] = 0;   
 for (int i = 2; i <= n; i ++ ) lg[i] = lg[i >> 1] + 1;  
 int m = lg[n] + 1;  
 maxv.resize(m); minv.resize(m);  
   
 for (int i = 0; i < m; i ++ ) maxv[i].resize(n), minv[i].resize(n);   
 for (int i = 0; i < n; i ++ ) maxv[0][i] = minv[0][i] = a[i];  
  
 for (int i = 1; i < m; i ++ ) {  
 for (int j = 0; j <= n - (1 << i); j ++ ) {  
 maxv[i][j] = max(maxv[i - 1][j], maxv[i - 1][j + (1 << (i - 1))]);  
 minv[i][j] = min(minv[i - 1][j], minv[i - 1][j + (1 << (i - 1))]);  
 }  
 }  
  
 }  
 T getmax(int l,int r){  
 int k = lg[r - l + 1];  
 return max(maxv[k][l], maxv[k][r - (1 << k) + 1]);  
 }   
 T getmin(int l,int r){  
 int k = lg[r - l + 1];  
 return min(minv[k][l], minv[k][r - (1 << k) + 1]);  
 }   
private:  
 int n;  
 vector<T> a;  
 vector<int> lg;  
 vector<vector<T>> maxv, minv;  
 vector<vector<T>> maxvid, minvid;  
};

### Trie树

struct Trie {  
 int c[N][26], tot, v[N];  
 void init() {  
 while (tot) {  
 v[tot] = 0;  
 memset(c[tot], 0, sizeof(c[tot]));  
 --tot;  
 }  
 memset(c[0], 0, sizeof(c[0]));  
 v[0] = 0;  
 }  
 void insert(char s[], int len) {  
 int now = 0;  
 for (int i = 0; i < len; ++i) {  
 int to = s[i] - 'a';  
 if (!c[now][to])c[now][to] = ++tot;  
 now = c[now][to];  
 }  
 v[now] = 1;  
 }  
  
 bool query(char s[], int len) {  
 int now = 0;  
 for (int i = 0; i < len; i ++ ) {  
 int to = s[i] - 'a';  
 if (!c[now][to])return 0;  
 now = c[now][to];  
 }  
 return v[now];  
 }  
  
 // bool query(char s[], int len) {  
 // int now = 0;  
 // for (int i = len - 1; ~i; --i) {  
 // int to = s[i] - 'a';  
 // if (!c[now][to])return 0;  
 // now = c[now][to];  
 // }  
 // return v[now];  
 // }  
} A;

### 并查集

struct DSU {  
 std::vector<int> f, siz, rank;  
 int c;  
 DSU(int n) : c(n), f(n), rank(n, 0), siz(n, 1) { std::iota(f.begin(), f.end(), 0); }  
 int find(int x) {  
 while (x != f[x]) x = f[x] = f[f[x]];  
 return x;  
 }  
 bool same(int x, int y) { return find(x) == find(y); }  
 int merge(int x, int y) {  
 x = find(x);  
 y = find(y);  
 if (x == y) return -1;  
 else -- c;  
 if(rank[x] > rank[y]) swap(x, y);  
 siz[y] += siz[x];  
 f[x] = y;  
 if(rank[x] == rank[y]) rank[y] ++;  
 return y;  
 }  
 int size(int x) { return siz[find(x)]; }  
}; // 按秩合并 + 路径压缩  
  
struct DSU {  
 std::vector<int> f, siz;  
 DSU(int n) : f(n), siz(n, 1) { std::iota(f.begin(), f.end(), 0); }  
 int leader(int x) {  
 while (x != f[x]) x = f[x] = f[f[x]];  
 return x;  
 }  
 bool same(int x, int y) { return leader(x) == leader(y); }  
 bool merge(int x, int y) {  
 x = leader(x);  
 y = leader(y);  
 if (x == y) return false;  
 siz[x] += siz[y];  
 f[y] = x;  
 return true;  
 }  
 int size(int x) { return siz[leader(x)]; }  
}; // 路径压缩

### 莫队

#include <bits/stdc++.h>  
using namespace std;  
using LL = long long;  
// #define int \_\_int128  
namespace io {  
constexpr int BUFFER\_SIZE = 1 << 16;  
char buffer[BUFFER\_SIZE], \*head, \*tail;  
  
char get\_char() {  
 if (head == tail) {  
 int l = (int)fread(buffer, 1, BUFFER\_SIZE, stdin);  
 tail = (head = buffer) + l;  
 }  
 return \*head++;  
}  
  
int read() {  
 int x = 0, f = 1;  
 char c = get\_char();  
 for (; !isdigit(c); c = get\_char())  
 if (c == '-') f = -1;  
 for (; isdigit(c); c = get\_char()) x = x \* 10 + c - '0';  
 return x \* f;  
}  
  
void print(int x) {  
 static int sta[1997];  
 int top = 0;  
 do {  
 sta[top++] = x % 10, x /= 10;  
 } while (x);  
 while (top) putchar(sta[--top] + 48); // 48 是 '0'  
}  
  
void println(int x) { print(x), putchar('\n'); }  
} // namespace io  
using namespace io;  
  
const int N = 5e5 + 10, M = 510;  
int n, m, k, block;  
int lal = 1, lar = 0;  
int bl[N], cnt[N], a[N];  
LL ans[N], sum;  
int id[N], stk[N], idx;  
  
struct node {  
 int l, r, id;  
 bool operator<(const node &T) const {  
 return bl[l] ^ bl[T.l] ? bl[l] < bl[T.l]  
 : ((bl[l] & 1) ? r < T.r : r > T.r);  
 }  
} modui[N];  
  
void add(int x) {  
 cnt[x]++;  
 if (not(cnt[x] ^ 1)) {  
 stk[++idx] = x;  
 id[x] = idx;  
 } else if (not(cnt[x] ^ 2)) {  
 stk[id[x]] = stk[idx]; //将x清空，即将栈顶元素移动到 x 的位置  
 id[stk[idx]] = id[x]; //将栈顶元素的 id 置为 x 的 id  
 --idx;  
 }  
}  
  
void del(int x) {  
 cnt[x]--;  
 if (not(cnt[x] ^ 1)) {  
 stk[++idx] = x;  
 id[x] = idx;  
 } else if (!cnt[x]) {  
 stk[id[x]] = stk[idx];  
 id[stk[idx]] = id[x];  
 --idx;  
 }  
}  
  
int main() {  
 n = read();  
 block = sqrt(n);  
 for (int i = 1; i <= n; i++) {  
 a[i] = read();  
 bl[i] = (i - 1) / block;  
 }  
 m = read();  
 for (int i = 1; i <= m; i++) {  
 modui[i].l = read(), modui[i].r = read();  
 modui[i].id = i;  
 }  
  
 sort(modui + 1, modui + 1 + m);  
  
 for (int i = 1; i <= m; i++) {  
 int l = modui[i].l, r = modui[i].r;  
 while (lal > l) lal--, add(a[lal]);  
 while (lar < r) lar++, add(a[lar]);  
 while (lal < l) del(a[lal]), lal++;  
 while (lar > r) del(a[lar]), lar--;  
 ans[modui[i].id] = stk[idx];  
 }  
 for (int i = 1; i <= m; i++) {  
 println(ans[i]);  
 }  
  
 return 0;  
}

### 莫队求 MEX

#include <bits/stdc++.h>  
using namespace std;  
using LL = long long;  
// #define int \_\_int128  
namespace io {  
 constexpr int BUFFER\_SIZE = 1 << 16;  
 char buffer[BUFFER\_SIZE], \*head, \*tail;  
  
 char get\_char() {  
 if (head == tail) {  
 int l = (int) fread(buffer, 1, BUFFER\_SIZE, stdin);  
 tail = (head = buffer) + l;  
 }  
 return \*head++;  
 }  
  
 int read() {  
 int x = 0, f = 1;  
 char c = get\_char();  
 for (; !isdigit(c); c = get\_char())  
 if (c == '-') f = -1;  
 for (; isdigit(c); c = get\_char()) x = x \* 10 + c - '0';  
 return x \* f;  
 }  
  
 void print(int x) {  
 static int sta[1997];  
 int top = 0;  
 do {  
 sta[top++] = x % 10, x /= 10;  
 } while (x);  
 while (top) putchar(sta[--top] + 48); // 48 是 '0'  
 }  
  
 void println(int x) { print(x), putchar('\n'); }  
} // namespace io  
using namespace io;  
  
const int N = 2e5 + 10, M = 510;  
  
int n, m, block;  
int belong[N], cnt[N], a[N], ans[N], tot[N], bl[N], br[N];  
  
struct node {  
 int l, r, id;  
 bool operator < (const node &T) const {  
 return belong[l] ^ belong[T.l] ? belong[l] < belong[T.l] : ((belong[l] & 1) ? r < T.r : r > T.r);  
 }  
} modui[N];  
  
void add(int x) {  
 if (!cnt[x]) tot[belong[x]] ++;  
 cnt[x] ++;  
}  
  
void del(int x) {  
 if (cnt[x] == 1) tot[belong[x]] --;  
 cnt[x] --;   
}  
//值域分块 每次查询sqrt(n)  
int query() {  
 int now = 0;  
 while (tot[now] == br[now] - bl[now] + 1) ++ now;  
 for (int i = bl[now]; i <= br[now]; i ++ ) {  
 if (!cnt[i]) return i;  
 }  
 return -1;  
}  
  
int main() {  
   
 n = read(); m = read(); block = sqrt(n); // 0, ...  
 belong[0] = 0;  
 for(int i = 1; i <= n; i ++ ) { a[i] = min(read(), n + 1); belong[i] = i / block; }   
 belong[n + 1] = (n + 1) / block; belong[n + 2] = (n + 2) / block;  
 for(int i = 1; i <= m; i ++ ) {  
 modui[i].l = read(), modui[i].r = read(); modui[i].id = i;  
 }   
  
 for (int i = 0; i <= belong[n + 1]; i ++ ) {  
 bl[i] = i \* block;  
 br[i] = bl[i] + block - 1;  
 }  
   
 sort(modui + 1, modui + 1 + m);  
 int lal = modui[1].l, lar = modui[1].l - 1;  
 for(int i = 1; i <= m; i ++ ) {  
 int l = modui[i].l, r = modui[i].r;  
 while(lal > l) lal --, add(a[lal]);  
 while(lar < r) lar ++, add(a[lar]);  
 while(lal < l) del(a[lal]), lal ++;  
 while(lar > r) del(a[lar]), lar --;  
 ans[modui[i].id] = query();  
 }  
 for(int i = 1; i <= m; i ++ ) {  
 println(ans[i]);  
 }  
   
 return 0;  
}

### 欧拉序求 LCA

int root = 1; //改一下根节点  
 vector<vector<int> > son(n + 1);  
 for(int i = 1; i < n; i ++ ) {  
 int u, v; cin >> u >> v;  
 son[u].emplace\_back(v);  
 son[v].emplace\_back(u);  
 }  
  
 vector<int> depth(n + 1), sz(n + 1), parent(n + 1), dfn(n + 1), que((n << 1) + 10);  
 int ts = 0;   
  
 function<void(int, int, int)> dfs1 = [&] (int u, int fa, int dep) {  
 sz[u] = 1; depth[u] = dep; parent[u] = fa;  
 dfn[u] = ++ ts; que[ts] = u;  
 for(auto &v: son[u]) if(v != fa) {  
 dfs1(v, u, dep + 1);  
 sz[u] += sz[v];  
 que[++ ts] = u;  
 }  
 };  
   
 dfs1(root, 0, 1);  
  
 int m = \_\_lg((n << 1) + 10) + 1;  
 vector<vector<int> > f(m, vector<int>((n << 1) + 10));  
  
 for(int i = 1; i <= ts; i ++ ) f[0][i] = que[i];  
 for(int j = 1; j < m; j ++ ) {  
 for(int i = 1; i + (1 << j) <= ts; i ++ ) {  
 int f1 = f[j - 1][i], f2 = f[j - 1][i + (1 << (j - 1))];  
 f[j][i] = depth[f1] < depth[f2] ? f1 : f2;  
 }  
 }  
  
 auto lca = [&] (int u, int v) {  
 if(dfn[u] > dfn[v]) swap(u, v);  
 u = dfn[u], v = dfn[v];  
 int k = \_\_lg(v - u + 1);  
 int f1 = f[k][u], f2 = f[k][v - (1 << k) + 1];  
 return depth[f1] < depth[f2] ? f1 : f2;  
 };

### 染色法判二分图

#include <bits/stdc++.h>  
using namespace std;  
const int N = 1e5 + 5, M = N \* 2;  
int h[N], e[M], ne[M], idx;  
int n, m;  
int color[N];  
void add(int a, int b) { e[idx] = b, ne[idx] = h[a], h[a] = idx++; }  
bool dfs(int u, int c) {  
 color[u] = c;  
 for (int i = h[u]; i != -1; i = ne[i]) {  
 int j = e[i];  
 if (!color[j]) {  
 if (!dfs(j, 3 - c)) return false;  
 } else if (color[j] == c) {  
 return false;  
 }  
 }  
 return 1;  
}  
int main() {  
 scanf("%d%d", &n, &m);  
 memset(h, -1, sizeof h);  
 while (m--) {  
 int a, b; scanf("%d%d", &a, &b);   
 add(a, b), add(b, a);  
 }  
 bool flag = 1;  
 for (int i = 1; i <= n; i++)  
 if (!color[i]) if (!dfs(i, 1)) {  
 flag = 0;  
 break;  
 }  
 if (flag) puts("Yes");  
 else puts("No");  
 return 0;  
}

### 树链剖分

vector<vector<int> > g(n + 1);  
for(int i = 1, u, v; i < n; i ++ ) {  
 cin >> u >> v;  
 g[u].emplace\_back(v); g[v].emplace\_back(u);  
}  
  
vector<int> fa(n + 1), depth(n + 1), sz(n + 1), son(n + 1), id(n + 1), top(n + 1);   
int ts = 0;  
  
function<void(int, int)> dfs1 = [&] (int u, int father) {  
 fa[u] = father;  
 depth[u] = depth[father] + 1;  
 sz[u] = 1;  
 for(auto& v: g[u]) if(v != father) {  
 dfs1(v, u);  
 sz[u] += sz[v];  
 if(sz[son[u]] < sz[v]) {  
 son[u] = v;  
 }  
 }   
};  
  
dfs1(1, 0);  
  
function<void(int, int)> dfs2 = [&] (int u, int tp) {  
 id[u] = ++ ts, top[u] = tp;  
 if(!son[u]) return ;  
 dfs2(son[u], tp);  
 for(auto& v: g[u]) {  
 if(v == fa[u] || v == son[u]) continue;  
 dfs2(v, v);  
 }  
};   
  
dfs2(1, 1);  
  
auto query\_path = [&] (int u, int v, vector<pair<int, int> > &s) {   
 //查询 u->v 经过的所有链  
 while(top[u] != top[v]) {  
 if(depth[top[u]] < depth[top[v]]) swap(u, v);  
 s.emplace\_back(id[top[u]], id[u]);  
 u = fa[top[u]];  
 }  
 if(depth[u] < depth[v]) swap(u, v);  
 s.emplace\_back(id[v], id[u]);  
};

### 树剖求 LCA

struct LCA { // index is the range of [1, n]  
 int n;  
 vector<vector<int> > g;  
 vector<int> top, parent, son, depth, sz;  
  
 void dfs1(int u, int fa, int d) {  
 sz[u] = 1, parent[u] = fa, depth[u] = d;  
 for(int &v: g[u]) if(v != fa) {  
 dfs1(v, u, d + 1);  
 sz[u] += sz[v];  
 if(sz[son[u]] < sz[v]) son[u] = v;  
 }  
 }  
  
 void dfs2(int u, int id) {  
 top[u] = id;  
 if(!son[u]) return ;  
 dfs2(son[u], id);  
 for(int &v: g[u]) {  
 if(v == parent[u] || v == son[u]) {  
 continue;  
 }  
 dfs2(v, v);  
 }  
 }  
  
 int lca(int a, int b) {  
 while(top[a] != top[b]) {  
 if(depth[top[a]] > depth[top[b]]) {  
 a = parent[top[a]];  
 } else {  
 b = parent[top[b]];  
 }  
 }  
 if(depth[a] < depth[b]) {  
 return a;  
 } else {  
 return b;  
 }  
 }  
  
 LCA(int \_n, vector<vector<int> >& \_son) : n(n), g(\_son), top(\_n + 1), parent(\_n + 1), son(\_n + 1), depth(\_n + 1), sz(\_n + 1) {  
 dfs1(1, -1, 1);  
 dfs2(1, 1);  
 }  
   
}; // index is the range of [1, n]

### 虚树

constexpr int N = 2E5 + 10;  
vector<int> son[N];  
vector<int> g[N];  
int tp, a[N], use[N];  
bool ans, has\_root, f;  
int dfn[N], depth[N], sz[N], hson[N], top[N], parent[N];  
  
void dfs1(int u, int fa, int d) {  
 depth[u] = d;  
 sz[u] = 1;  
 parent[u] = fa;  
 for (int &v : son[u])  
 if (v != fa) {  
 dfs1(v, u, d + 1);  
 sz[u] += sz[v];  
 if (hson[u] == -1 || sz[hson[u]] < sz[v]) hson[u] = v;  
 }  
}  
  
void dfs2(int u, int id) {  
 top[u] = id;  
 dfn[u] = ++tp;  
 if (hson[u] == 0) return;  
 dfs2(hson[u], id);  
 for (int &v : son[u])  
 if (v != parent[u] && v != hson[u]) {  
 dfs2(v, v);  
 }  
}  
  
int lca(int u, int v) {  
 while (top[u] != top[v]) {  
 if (depth[top[u]] > depth[top[v]]) {  
 u = parent[top[u]];  
 } else {  
 v = parent[v];  
 }  
 }  
 return depth[u] < depth[v] ? u : v;  
}  
  
void dfs3(int u, int fa) {  
 int now = 0;  
 for (int &v : g[u]) {  
 if (v != fa) dfs3(v, u);  
 if (!f || v != 1) {  
 ++now;  
 }  
 }  
  
 if (!(f && u == 1) && now > 2) {  
 ans = true;  
 }  
 g[u].clear();  
}  
  
void solve() {  
 int n;  
 cin >> n;  
 for (int i = 1; i < n; i++) {  
 int u, v;  
 cin >> u >> v;  
 son[u].emplace\_back(v);  
 son[v].emplace\_back(u);  
 }  
  
 dfs1(1, 0, 1);  
 dfs2(1, 1);  
 int q;  
 cin >> q;  
  
 while (q--) {  
 int k;  
 cin >> k;  
 has\_root = false;  
  
 for (int i = 1; i <= k; i++) {  
 cin >> a[i];  
 if (a[i] == 1) has\_root = true;  
 }  
  
 sort(a + 1, a + 1 + k, [&](int x, int y) { return dfn[x] < dfn[y]; });  
  
 auto add = [&](int u, int v) {  
 g[u].emplace\_back(v);  
 g[v].emplace\_back(u);  
 };  
  
 vector<int> stk{1};  
  
 for (int i = 1; i <= k; i++) {  
 if (a[i] != 1) {  
 int p = lca(a[i], stk.back());  
 if (p != stk.back()) {  
 while (dfn[p] < dfn[stk[(int)stk.size() - 2]]) {  
 add(stk.back(), stk[(int)stk.size() - 2]);  
 stk.pop\_back();  
 }  
 add(p, stk.back()), stk.pop\_back();  
 if (dfn[p] > dfn[stk.back()]) stk.emplace\_back(p);  
 }  
 stk.emplace\_back(a[i]);  
 }  
 }  
 while (stk.size() > 1) {  
 if (stk.back() != 1) {  
 add(stk.back(), stk[(int)stk.size() - 2]);  
 stk.pop\_back();  
 }  
 }  
 f = (!has\_root && (int)g[1].size() == 1);  
 // 表示关键点里面没有根，并且根(一定存在)在虚树中只连了一个点  
 dfs3(1, 0);  
 if (ans) {  
 cout << "NO\n";  
 } else {  
 cout << "YES\n";  
 }  
  
 ans = false;  
 }  
}  
  
int main() {  
 ios::sync\_with\_stdio(false);  
 cin.tie(nullptr);  
 int T = 1; // cin >> T;  
 while (T--) solve();  
 return 0;  
}

### ODT 老司机树

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ll;  
  
#define rep(i, a, b) for (int i(a); i <= b; i++)  
#define dec(i, a, b) for (int i(a); i >= b; i--)  
  
#ifdef LOCAL  
#include <debugger>  
#else  
#define debug(...) 42  
#endif  
  
template <typename T>  
inline void chkmax(T &x, T y) {  
 x = max(x, y);  
}  
template <typename T>  
inline void chkmin(T &x, T y) {  
 x = min(x, y);  
}  
  
constexpr int mod = 998244353;  
constexpr int inv = 499122177;  
struct ODT {  
 ll l, r;  
 mutable ll v;  
 bool operator<(const ODT &T) const { return l < T.l; }  
};  
  
set<ODT> odt;  
  
using Iter = set<ODT>::iterator;  
  
Iter split(ll pos) {  
 auto it = odt.lower\_bound({pos, 0, 0});  
 if (it != odt.end() && it->l == pos) return it;  
 --it;  
 auto [l, r, v] = \*it;  
 odt.erase(it);  
 odt.insert({l, pos - 1, v});  
 return odt.insert({pos, r, v}).first;  
}  
  
ll assign(ll l, ll r, ll v) {  
 auto itr = split(r + 1), itl = split(l);  
  
 ll ret = 0;  
 for (auto it = itl; it != itr; it++) {  
 auto [x, y, v0] = \*it;  
 ret += (v - v0) % mod \* ((x + y) % mod) % mod \* ((y - x + 1) % mod) % mod \*  
 inv % mod;  
 }  
  
 odt.erase(itl, itr);  
 odt.insert({l, r, v});  
 return ret % mod;  
}  
  
void solve() {  
 ll n, q;  
 cin >> n >> q;  
 odt.insert({1, n, 0});  
 while (q--) {  
 ll l, r, d;  
 cin >> d >> l >> r;  
 cout << assign(l, r, d) << "\n";  
 }  
}  
  
int main() {  
 ios::sync\_with\_stdio(false);  
 cin.tie(nullptr);  
 solve();  
  
 return 0;  
}

## 计算几何

### 凸多边形判定

// 叉积: A(x1, y1) X B(x2, y2) = x1 y2 - x2 y1  
// 通过外积正负号，可以判定两个向量的相对位置，也就是夹角。  
// 夹角的性质，可以用来解决判别凸多边形，一个点在多边形内等问题。  
//例如：判别凸多边形：  
// 向量每个点相连接，假如是多边形 ABCD，则得到 AB，BC，CD，DA  
// 四个向量，分别判定  
// AB\* BC  
// BC\*CD  
// CD\*DA  
// DA\*AB  
// 四个叉乘同不同号，如果同号，证明是凸四边形。leetcode469  
  
class Solution {  
 public:  
 bool isConvex(std::vector<std::vector<int>>& points) {  
 int n = points.size();  
 int xa, ya, xb, yb, xc, yc;  
 int prevsign = 0;  
 for (int i = 0; i < n; i++) {  
 xa = points[i][0];  
 ya = points[i][1];  
 xb = points[(i + 1) % n][0];  
 yb = points[(i + 1) % n][1];  
 xc = points[(i + 2) % n][0];  
 yc = points[(i + 2) % n][1];  
 int cur = (xc - xb) \* (yb - ya) - (yc - yb) \* (xb - xa);  
 if (prevsign == 0) {  
 prevsign = cur;  
 } else {  
 if (prevsign < 0 && cur > 0) return false;  
 if (prevsign > 0 && cur < 0) return false;  
 }  
 }  
 return true;  
 }  
};