```cpp
/* Main */
#pragma GCC optimize(3,"Ofast","inline")
#include<bits/stdc++.h>
using namespace std;
#define lowbit(x) ((x)&(-(x)))
#define MP make_pair
#define fi first
#define se second
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count()); //mt
19937_64 for 64-bits
bool Finish_read;
template<class T>inline void read(T &x) {Finish_read = 0; x = 0; int f =
1; char ch = getchar(); while(!isdigit(ch)) {if(ch == '-')f = -1; if(ch =
= EOF)return; ch = getchar();} while(isdigit(ch))x = x * 10 + ch - '0', c
h = getchar(); x *= f; Finish_read = 1;}
typedef long long LL;
typedef unsigned long long ULL;
typedef pair<int,int> pii;
const double PI = acos(-1.0);
const double eps = 1e-6;
const int INF = 0x3f3f3f3f;
const int maxn = 1e5 + 11;
int N, M, T;

int main(){

    return 0;
}

/* Poly_Hash */

#pragma GCC optimize(3,"Ofast","inline")
#include<bits/stdc++.h>
//For a given string, construct all cyclic shifts, sort them lexicographi
cally,
//output the last column, and find the position of the original string in
 this list.
using namespace std;
typedef unsigned long long ULL;
const int maxn = 1e5 + 11;
int N, M, T;
char data[maxn]; string S;
void readin(){
    scanf("%s", data);
    S = string(data);
    N = S.size();
    S += S;
}
struct PolyHash {
    // -------- Static variables --------
    static const int mod = (int)1e9 + 123; // prime mod of polynomial has
hing
    static vector<int> pow1;        // powers of base modulo mod
    static vector<ULL> pow2;        // powers of base modulo 2^64
    static int base;                     // base (point of hashing)
    // --------- Static functons --------
    static inline int diff(int a, int b) {
        return (a -= b) < 0 ? a + mod : a;
    }
    // -------------- Variables of class -------------
    vector<int> pref1; // Hash on prefix modulo mod
    vector<ULL> pref2; // Hash on prefix modulo 2^64
    // Cunstructor from string:
    PolyHash(const string& s):pref1(s.size()+1u,0),pref2(s.size()+1u,0){
        assert(base < mod);
        const int n = s.size(); // Firstly calculated needed power of bas
e:
        while ((int)pow1.size() <= n) {
            pow1.push_back(1LL * pow1.back() * base % mod);
            pow2.push_back(pow2.back() * base);
        }
        for (int i = 0; i < n; ++i) { // Fill arrays with polynomial hash
es on prefix
            assert(base > s[i]);
            pref1[i+1] = (pref1[i] + 1LL * s[i] * pow1[i]) % mod;
            pref2[i+1] = pref2[i] + s[i] * pow2[i];
        }
    }
    // Polynomial hash of subsequence [pos, pos+len)
    // If mxPow != 0, value automatically multiply on base in needed powe
r. Finally base ^ mxPow
    inline pair<int, ULL> operator()(const int pos, const int len, const
int mxPow = 0) const {
        int hash1 = pref1[pos+len] - pref1[pos];
        ULL hash2 = pref2[pos+len] - pref2[pos];
        if (hash1 < 0) hash1 += mod;
        if (mxPow != 0) {
            hash1 = 1LL * hash1 * pow1[mxPow-(pos+len-1)] % mod;
            hash2 *= pow2[mxPow-(pos+len-1)];
        }
        return make_pair(hash1, hash2);
    }
};
int PolyHash::base(233);
vector<int> PolyHash::pow1{1};
vector<ULL> PolyHash::pow2{1};

void solve(){
    static int mxPow = 2*N;
    static PolyHash hs(S);
    vector<int> id(N);
    iota(id.begin(), id.end(), 0);
    stable_sort(id.begin(), id.end(), [&](const int &i, const int &j) {
            int left = 0, right = N + 1;
            while (left < right) {
                int mid = (left + right + 1) / 2;
                if (hs(i, mid, mxPow) == hs(j, mid, mxPow)) left = mid;
                else right = mid - 1;
            }
            return left <= N && S[i + left] < S[j + left];
         }
    );
    string ans; int pos = -1;
    for (int i = 0; i < id.size(); i++) {
        int j = id[i];
        if (j == 0 && pos == -1) pos = i;
        ans.push_back(S[j + N - 1]);
    }
    printf("%d\n%s\n", pos + 1, ans.c_str());
}
int main(){
    readin();
    solve();
    return 0;
}

/* Dynamic_Segment_Tree */

#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
typedef unsigned long long ULL;
const int maxn = 1e7 + 5e6;
int N, Q, M, tot, rt;
int lson[maxn], rson[maxn], sum[maxn], lazy[maxn];
int x1, x2, val;
void update(int L, int R, int& i) {
    if (!i) { i = ++tot; }
    if (x1 <= L && R <= x2) {sum[i] = val * (R - L + 1); lazy[i] = val; r
eturn;}
    int M = (L + R) / 2;
    if (lazy[i] != -1) {
        if (L != R){
            if (!lson[i]) { lson[i] = ++tot; }
            if (!rson[i]) { rson[i] = ++tot; }
            lazy[lson[i]] = lazy[rson[i]] = lazy[i];
            sum[lson[i]] = (M - L + 1) * lazy[i];
            sum[rson[i]] = (R - M) * lazy[i];
        }
        lazy[i] = -1;
    }
    if (x1 <= M) { update(L, M, lson[i]); }
    if (x2 > M) { update(M + 1, R, rson[i]); }
    sum[i] = sum[lson[i]] + sum[rson[i]];
}

int res;
void query(int L, int R, int& i) {
    if (x1 <= L && R <= x2) {res += sum[i]; return;}
    int M = (L + R) / 2;
    if (lazy[i] != -1) {
        if (!lson[i]) { lson[i] = ++tot; }
        if (!rson[i]) { rson[i] = ++tot; }
        lazy[lson[i]] = lazy[rson[i]] = lazy[i];
        sum[lson[i]] = (M - L + 1) * lazy[i];
        sum[rson[i]] = (R - M) * lazy[i];

        lazy[i] = -1;
    }
    if (x1 <= M) { query(L, M, lson[i]); }
    if (x2 > M) { query(M + 1, R, rson[i]); }
}

int main() {
    cin >> N >> Q;
    int l, r, k;
    memset(lazy, -1, sizeof(lazy));
    while (Q--) {
        scanf("%d%d%d", &l, &r, &k);
        x1 = l, x2 = r, val = 2 - k;
        update(1, N, rt);
        printf("%d\n", N - sum[1]);
    }
    return 0;
}

#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
typedef unsigned long long ULL;
const int maxn = 1e7 + 5e6;
int N, Q, M, tot, rt;
int lson[maxn], rson[maxn], sum[maxn], lazy[maxn];
int x1, x2, val;
void update(int L, int R, int& i) {
    if (!i) { i = ++tot; }
    if (x1 <= L && R <= x2) {sum[i] = val * (R - L + 1); lazy[i] = val; r
eturn;}
    int M = (L + R) / 2;
    if (lazy[i] != -1) {
        if (L != R){
            if (!lson[i]) { lson[i] = ++tot; }
            if (!rson[i]) { rson[i] = ++tot; }
            lazy[lson[i]] = lazy[rson[i]] = lazy[i];
            sum[lson[i]] = (M - L + 1) * lazy[i];
```

```
                sum[rson[i]] = (R - M) * lazy[i];
            }
            lazy[i] = -1;
        }
        if (x1 <= M) { update(L, M, lson[i]); }
        if (x2 > M) { update(M + 1, R, rson[i]); }
        sum[i] = sum[lson[i]] + sum[rson[i]];
    }

    int res;
    void query(int L, int R, int& i) {
        if (x1 <= L && R <= x2) {res += sum[i]; return;}
        int M = (L + R) / 2;
        if (lazy[i] != -1) {
            if (!lson[i]) { lson[i] = ++tot; }
            if (!rson[i]) { rson[i] = ++tot; }
            lazy[lson[i]] = lazy[rson[i]] = lazy[i];
            sum[lson[i]] = (M - L + 1) * lazy[i];
            sum[rson[i]] = (R - M) * lazy[i];

            lazy[i] = -1;
        }
        if (x1 <= M) { query(L, M, lson[i]); }
        if (x2 > M) { query(M + 1, R, rson[i]); }
    }

    int main() {
        cin >> N >> Q;
        int l, r, k;
        memset(lazy, -1, sizeof(lazy));
        while (Q--) {
            scanf("%d%d%d", &l, &r, &k);
            x1 = l, x2 = r, val = 2 - k;
            update(1, N, rt);
            printf("%d\n", N - sum[1]);
        }
        return 0;
    }
```