Notebook - Maratona de Programação

DSUm balão da cor sim cor não

Contents

1 Algoritmos

1.1 Busca Binaria

```cpp
#include <bits/stdc++.h>
using namespace std;

bool check(int valor, int x) {
    return valor <= x;
}

int bb(int a, int b, int x){
    int l = a;
    int r = b;
    while (l < r) {
        int mid = (l + r) / 2;
        if (check(mid, x)) r = mid;
        else l = mid + 1;
    }
    return l;
}

bool check(int valor) {
    return valor <= 10;
}

int bb_menor(int a, int b){
    int l = a;
    int r = b;
    while (l < r) {
        int mid = (l + r) / 2;
        if (check(mid)) r = mid;
        else l = mid + 1;
    }

    return l;
}


int bb_maior(int a, int b){
    int l = a;
    int r = b;
    while (l < r) {
        int mid = (l + r) / 2;
        if (!check(mid)) r = mid;
        else l = mid + 1;
    }
}
```

1.2 Busca Binaria Double

```cpp
//
// Complexidade : O(NlogN)

#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
const int MAX = 1e5+1;
const double EPS = 0.0000001;

vector<int> v(100001);
int n;
ll check(double x){
    ll sum = 0;
    for(int i=0; i<n; i++){
        sum += (v[i]/x);
    }
    return sum;
}

int main(){
    int k;
    cin>>n>>k;

    for(int i=0; i<n; i++)cin>>v[i];

    double l=0.0000000, r=10000000.0000000;
    double mid;
    while(r-l>EPS){
        mid = (double)((l + r)/2);
        if (check(mid)>=k){
            l=mid;
        }
        else{
            r = mid;
        }
    }
    cout<<fixed<<setprecision(7)<<mid<<endl;

    return 0;
}
```

1.3 Busca Binaria Resposta

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define loop(i,a,n) for(int i=a; i < n; i++)

ll upperbound(ll maior, ll k, vector<ll> tabuas){
    ll mid = 0, l = 0, r = maior, count = 300;
    ll aux;
    while((l < r) && (count--)){
        aux = 0;
        mid = (l+r)/2;
        loop(i,0,tabuas.size()){
            if(mid > 0){aux += (tabuas[i]/mid);}
        }
        if(aux >= k){l = mid;}
        else{r = mid;}
    }

    ll aux_2 = 0;
    loop(i,0,tabuas.size()){
        aux_2 += (tabuas[i]/(mid+1));
    }
    if(aux_2 >= k){return mid+1;}

    if(aux < k){
        int aux_2 = 0;
        loop(i,0,tabuas.size()){
            if(mid - 1 > 0){aux_2 += (tabuas[i]/(mid
-1));}
        }
        if(aux_2 >= k){return mid-1;}
    }

    return mid;
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie( NULL);
    cout.tie(NULL);
    int n; cin >> n;
    ll k; cin >> k;
    vector<ll> tabuas(n);
    ll maior = 0;
    loop(i,0,n){
        cin >> tabuas[i];
        maior = max(maior,tabuas[i]);
    }
    cout << upperbound(maior,k,tabuas);
}
```

1.4 Busca Ternaria

```cpp
// Uma busca em uma curva, avaliando dois pontos diferentes
// Complexidade: O(Nlog3N)

double check(vector<int> v, vector<int> t, double x){
    double ans = 0;
    for(int i=0; i<v.size(); i++){
        ans = max(ans, (double)(abs(v[i]-x) + t[i]));
    }
    return ans;
}

int32_t main(){ sws;

    int t; cin>>t;
    while(t--){
        int n; cin>>n;
        vector<int> v(n);
        vector<int> t(n);
        input(v);
        input(t);

        double ans = 0.0;
        double l=0.0, r=1e9;
        while(r-l >= EPS){

            double mid1 = (double) l + (r - l) / 3;
            double mid2 = (double) r - (r - l) / 3;

            double x1 = check(v, t, mid1);
            double x2 = check(v, t, mid2);

            if(x1 < x2){
                r = mid2;
            }else{
                l = mid1;
                ans = l;
            }
        }
        cout<<fixed<<setprecision(7);
        cout<<ans<<endl;
    }
    return 0;
}
```

1.5 Delta

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int n, q;
    cin >> n >> q;
    vector<int> v(n,0);
    vector<int> delta(n+2, 0);

    while(q--){
        int l, r, x;
        cin >> l >> r >> x;
        delta[l] += x;
        delta[r+1] -= x;
    }

    int atual = 0;
    for(int i=0; i < n; i++){
        atual += delta[i];
        v[i] = atual;
    }

    for(int i=0; i< n; i++){
        cout << v[i] << " ";
    }
    cout << endl;

    return 0;
}
```

1.6 Fast Exponentiaton

```cpp
// recursivo
int fast_exp(int base, int e, int m){
    if(!e) return 1;
    int ans = fast_exp(base * base % m, e/2, m);
    if(e % 2) return base * ans % m;
    else return ans;
}
//iterativo
int fast_exp(int base, int e, int m) { // iterativo
    int ret = 1;
    while (e) {
        if (e & 1) ret = (ret * base) % m;
        e >>= 1;
        base = (base * base) % m;
    }
    return ret;
}
```

1.7 Psum

```cpp
#include <bits/stdc++.h>
using namespace std;

#define input(x) for (auto &it : x) cin >> it
typedef long long ll;
vector<ll> psum(1e5);

int solve(int l, int r){
    if(l==0) return psum[r];
    else return psum[r] - psum[l-1];
}

int main(){

    int n, q;
    cin>>n>>q;

    vector<int> v(n);
    input(v);
    for(int i=0; i<n; i++){
        if(i==0) psum[i] = v[i];
        else psum[i] = psum[i-1] + v[i];
    }
    while(q--){
        int l, r;
        cin>>l>>r;

        cout<<(solve(l,r))<<endl;
    }

    return 0;
}
```

1.8 Psum2d

```cpp
int psum[MAX][MAX];

int32_t main(){ sws;
    int t; cin>>t;
    while(t--){
        memset(psum, 0, sizeof(psum));
        int n, q; cin>>n>>q;

```

```
        for(int i=0; i<n; i++){
            int x, y;
            cin>>x>>y;

            psum[x][y] += x*y;
        }

        for(int i=1; i<MAX; i++)
            for(int j=1; j<MAX; j++)
                psum[i][j] += psum[i-1][j];

        for(int i=1; i<MAX; i++){
            for(int j=1; j<MAX; j++){
                psum[i][j] += psum[i][j-1];
            }
        }

        for(int i=0; i<q; i++){
            int x1, y1, x2, y2;
            cin>>x1>>y1>>x2>>y2;
            x2--; y2--;

            int soma = psum[x1][y1] + psum[x2][y2] - psum[x2][y1] - psum[x1][y2];
            cout<<soma<<endl;
        }
    }
    return 0;
}
```

2 DP

2.1 Dp

```
// DP - Dynamic Programming

#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
const int MAX = 110;

int n;
int tab[MAX];
vector<int> v;

ll dp(int i){
    if(i>=n) return 0;
    if(tab[i] != -1) return tab[i];

    int pega = v[i] + dp(i+2);
    int npega = dp(i+1);

    tab[i] = max(pega, npega);
    return tab[i];
}

int main(){
    memset(tab, -1, sizeof(tab));
    cin>>n;

    v.assign(n, 0);

    cout<<dp(0)<<endl;

    return 0;
}
```

2.2 Knapsack

```
#include <bits/stdc++.h>
using namespace std;

#define int long long
#define ll long long
#define sws ios::sync_with_stdio(false);cin.tie( NULL );cout.tie(NULL);
#define pb(x) push_back(x);
#define pii pair<int,int>
const int N = 1e3+5;

int n, t;
int tab[N][N];
bool pegou[N][N];
vector<pair<int,int>> v;

vector<int> resposta;

int dp(int idx, int dias){
    if(idx >= n) return 0;
    if(tab[idx][dias] != -1) return tab[idx][dias];

    int pega=0;
    if(dias+v[idx].first <= t){
        pega = dp(idx+1, dias+v[idx].first)+v[idx].second;
    }

    int npega = dp(idx+1, dias);

    if(pega>npega) pegou[idx][dias] = true;

    return tab[idx][dias] = max(pega, npega);
}

int32_t main(){
    memset(tab, -1, sizeof(tab));
    cin>>n>>t;
    for(int i=0; i<n; i++){
        int ti, di;
        cin>>ti>>di;

        v.push_back({ti, di});
    }
    dp(0, 0);
    int i = 0, j =0;
    vector<int> ans;
    // retornar os valores
    while(i < n){
        if(pegou[i][j]){
            j += v[i].first;
            ans.push_back(i+1);
        }
        i++;
    }
    cout<<ans.size()<<endl;
    for(int i=0; i<ans.size(); i++){
        cout<<ans[i]<<" ";
    }

}
```

2.3 Lis

```
// Longest increase sequence
// O(nlogn)
multiset<int> S;
for(int i=0;i<n;i++){
    auto it = S.upper_bound(vet[i]); // upper - longest strictly increase sequence
    if(it != S.end())
        S.erase(it);
    S.insert(vet[i]);
}
```

```
// size of the lis
int ans = S.size();

//////////////////////////// see that later
// https://codeforces.com/blog/entry/13225?#comment-180208

vi LIS(const vi &elements){
    auto compare = [&](int x, int y) {
        return elements[x] < elements[y];
    };
    set< int, decltype(compare) > S(compare);

    vi previous( elements.size(), -1 );
    for(int i=0; i<int( elements.size() ); ++i){
        auto it = S.insert(i).first;
        if(it != S.begin())
            previous[i] = *prev(it);
        if(*it == i and next(it) != S.end())
            S.erase(next(it));
    }

    vi answer;
    answer.push_back( *S.rbegin() );
    while ( previous[answer.back()] != -1 )
        answer.push_back( previous[answer.back()] );
    reverse( answer.begin(), answer.end() );
    return answer;
}
```

2.4 Mochila Iterativa

```
#include <bits/stdc++.h>
using namespace std;

const int maxn = 110, maxp = 1e5+10;
const long long inf = 0x3f3f3f3f3f3f3f3f; // ~= 10^18

int v[maxn], p[maxn];
long long dp[maxn][maxp];

int main() {
    int n, C; scanf("%d %d", &n, &C);
    for(int i = 1; i <= n; i++)
        scanf("%d %d", &p[i], &v[i]);

    long long ans = 0;
    // inicializando o vetor
    for(int i = 1; i <= n; i++)
        for(int P = p[i]; P <= C; P++)
            dp[i][P] = -inf;
    // definindo o caso base
    dp[0][0] = 0;

    for(int i = 1; i <= n; i++) {
        for(int P = 0; P <= C; P++) {
            dp[i][P] = dp[i-1][P];
            if(P >= p[i])
                dp[i][P] = max(dp[i][P], dp[i-1][P-p[i]] + v[i]);
            ans = max(ans, dp[i][P]);
        }
    }

    printf("%lld\n", ans);
}
```

2.5 Mochila Recursiva

```
#include <bits/stdc++.h>
using namespace std;

const int maxn = 110, maxp = 1e5+10;

int v[maxn], p[maxn], n;
long long dp[maxn][maxp];
bool vis[maxn][maxp];

long long solve(int i, int P) {
    if(i == n+1) return 0; // caso base, nao ha mais itens para se considerar
    if(vis[i][P]) return dp[i][P];
    vis[i][P] = 1;

    // primeira possibilidade, nao adicionar o elemento
    dp[i][P] = solve(i+1, P);

    // segunda possibilidade, adicionar o elemento.
    // Lembrar de tirar o maximo com o valor ja calculado da primeira possibilidade
    if(P >= p[i])
        dp[i][P] = max(dp[i][P], solve(i+1, P - p[i]) + v[i]);

    return dp[i][P];
}

int main() {
    int C; scanf("%d %d", &n, &C);
    for(int i = 1; i <= n; i++)
        scanf("%d %d", &p[i], &v[i]);
    printf("%lld\n", solve(1, C));
}
```

3 ED

3.1 Bitwise

```
// Bitwise Operations

#include <bits/stdc++.h>
using namespace std;


// Verificar se o bit esta ligado
bool isSet(int bitPosition, int number) {
    bool ret = ((number & (1 << bitPosition)) != 0);
    return ret;
}

// Ligar o bit
bool setBit(int bitPosition, int number) {
    return (number | (1 << bitPosition) );
}

// Gerando todos os subconjuntos de um conjunto em binario
void possibleSubsets(char S[], int N) {
    for(int i = 0;i < (1 << N); ++i) {  // i = [0, 2^N - 1]
        for(int j = 0;j < N;++j)
            if(i & (1 << j))  // se o j-esimo bit de i esta setado, printamos S[j]
                cout << S[j] << " ";
        cout << endl;
    }
}
```

3.2 Dsu

```
#include <bits/stdc++.h>
using namespace std;
```

```

// Complexidade
// build : O(N)
// find : O(logN)
class DSU{
    vector<int> parent, sz;
    public:
    void make(int v){
        parent[v] = v;
        sz[v] = 1;
    }

    int find(int v){
        if (v == parent[v]) return v;
        return parent[v] = find(parent[v]);
    }

    void union_(int a, int b){
        a = find(a);
        b = find(b);

        if(sz[b]>sz[a]) swap(a,b);
        if (a != b){
            sz[a] += sz[b];
            parent[b] = a;
        }
    }

    bool same(int a, int b){
        a = find(a), b = find(b);
        return a == b;
    }

    DSU(int n): parent(n+1), sz(n+1){
        for(int i=1; i<=n; i++) make(i);
    }
};


int main(){
    DSU dsu(10);
    return 0;
}
```

3.3 Lazy Seg

```
//Seg Tree. Considering I = 1, L = 0 and R = N-1; I is the first index in st.
class SegTree{
    private:
        ll st[4*MAX], lazy[4*MAX];

        ll merge(ll a, ll b){
            return min(a,b);
        }

        void push(int i, long long x = 0){
            st[i] += (lazy[i]+x);
            if(2*i < 4*MAX) lazy[2*i] += (lazy[i]+x);
            if(2*i+1 < 4*MAX) lazy[2*i+1] += (lazy[i]+x);
            lazy[i] = 0;
        }

    public:
        void build(int i = 1, int l = 0, int r = n-1){
            if(l == r){
                st[i] = a[l]; //leaf node.
                lazy[i] = 0;
            }
            else{
                int mid = (r+l)/2;
                lazy[i] = 0;
                build(2*i, l, mid);
                build(2*i + 1, mid+1, r);
                st[i] = merge(st[2*i], st[2*i + 1]); //parent node.
            }
            return;
        }

        ll query(int l, int r, int i = 1, int auxl = 0, int auxr = n-1){
            if(l <= auxl && r >= auxr){ //total overlap.
                if(lazy[i]){
                    push(i);
                }
                return st[i];
            }
            else if(auxr < l || auxl > r){ //no overlap.
                return LLINF;
            }
            else{ //partial overlap
                int auxmid = (auxr+auxl)/2;
                push(i);
                return merge(query(l, r, 2*i, auxl, auxmid), query(l, r, 2*i + 1, auxmid+1, auxr));
            }
        }

        void update(int l, int r, ll x, int i = 1, int auxl = 0, int auxr = n-1){
            if(l <= auxl && auxr <= r){ //total overlap.
                push(i,x);
            }
            else if(auxr < l || auxl > r){ //no overlap.
                return;
            }
            else{ //partial overlap
                int auxmid = (auxr+auxl)/2;
                update(l, r, x, 2*i, auxl, auxmid);
                update(l, r, x, 2*i + 1, auxmid+1, auxr);
                st[i] = merge(st[2*i],st[2*i+1]);
            }
        }
};

int main(){
    int q; cin >> n >> q;
    SegTree seg;
    for(int i = 0; i < n; i++){
        cin >> a[i];
    }
    seg.build();
    for(int i = 0; i < q; i++){
        int op; cin >> op;
        if(op == 1){
            int l, r, x; cin >> l >> r >> x;
            seg.update(l-1,r-1,x);
        }
        else{
            int k; cin >> k;
            cout << seg.query(k-1,k-1) << "\n";
        }
    }
}
```

3.4 Merge Sort

```
#include <bits/stdc++.h>
```

```cpp
using namespace std;

#define INF 1000000000

void merge_sort(vector<int> &v){
    if(v.size()==1)return;

    vector<int> v1, v2;

    for(int i=0; i<v.size()/2; i++) v1.push_back(v[i]);
    for(int i=v.size()/2; i<v.size(); i++) v2.push_back(v[i]);

    merge_sort(v1);
    merge_sort(v2);

    v1.push_back(INF);
    v2.push_back(INF);

    int ini1=0, ini2=0;

    for(int i=0; i<v.size(); i++){
        if(v1[ini1]<v2[ini2]){
            v[i] = v1[ini1];
            ini1++;
        }else{
            v[i] = v2[ini2];
            ini2++;
        }
    }
    return;
}
```

3.5 Ordered Set

```cpp
// disable define int long long
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <class T>
    using ord_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

// k-th maior elemento - O(logN) - idx em 0
s.find_by_order(k)

// qtd elementos < k - O(logN)
s.order_of_key(k)

ord_set<int> s;
```

3.6 Segtree 1

```cpp
#include <bits/stdc++.h>
using namespace std;

class SegTree{
    vector<int> seg;
    int size;
    int elem_neutro = 0;

    int merge(int a, int b){
        return a^b;
    }
    void update(int idx, int val, int stl, int str, int no){
        if(stl == idx and str==idx){
            seg[no] = val;
            return;
        }
        if(stl>idx or str<idx) return;

        int mid = (stl+str)/2;
        update(idx, val, stl, mid, 2*no);
        update(idx, val, mid+1, str, 2*no+1);

        seg[no] = merge(seg[2*no], seg[2*no+1]);
    }

    int query(int l, int r, int stl, int str, int no){
        if(str<l or stl>r) return elem_neutro;
        if(stl>=l and str<=r) return seg[no];

        int mid = (stl+str)/2;
        int x = query(l, r, stl, mid, 2*no);
        int y = query(l, r, mid+1, str, 2*no+1);
        return merge(x, y);
    }
    public:
        SegTree(int n): seg(4*n, 0){size=n;}
        int query(int l, int r){return query(l, r, 0, size-1, 1);}
        void update(int idx, int val){update(idx, val, 0, size-1, 1);}
        void out(){for(int i=0; i<size; i++){cout<<query(i, i)<<" ";cout<<endl;}}
};

int32_t main(){
    int n, q;
    cin>>n>>q;
    SegTree seg(n);
    for(int i=0; i<n; i++){
        int x; cin>>x;
        seg.update(i,x);
    };
    for(int i=0; i<q; i++){
        int a, b;
        cin>>a>>b;

        cout<<seg.query(a-1, b-1)<<endl;
    }

    return 0;
}
```

3.7 Segtree 2

```cpp
#include <bits/stdc++.h>
using namespace std;
//SEG-TREES are used when we want to apply queries in intervals(segmentes) of a vector, such as getting the min value, getting the max value or
//getting the sum of this segment, and also doing updates to these segments in a efficient O complexity.

//It takes O(n) to build a segment tree.
//It takes O(log n) to answer a query and to update a segment.
//Also, an important thing to notice is that we dont actually implement a tree, we do it in the form of an array.
//Queries are done in the same way despite of the seg tree type, therefore what actually changes is how we build the tree considering what we want
//such as: max, min, sum...

const int INF = 0x3f3f3f3f;
const int MAX = 200005;
int n;
int a[MAX];
```

```

// Min Seg Tree. Considering L = 1 and R = N;
class MinSegTree{
    private:
        int st[4*MAX];
    public:
        void build(int i, int l, int r){
            if(l == r){
                st[i] = a[l]; //leaf node.
            }
            else{
                int mid = (r+l)/2;
                build(2*i, l, mid);
                build(2*i + 1, mid+1, r);
                st[i] = min(st[2*i], st[2*i + 1]); // parent node.
            }
            return;
        }

        int getmin(int i, int auxl, int auxr, int l, int r){
            if(l <= auxl && r >= auxr){ //total overlap.
                return st[i];
            }
            else if(auxr < l || auxl > r){ //no overlap.
                return INF;
            }
            else{ //partial overlap
                int auxmid = (auxr+auxl)/2;
                return min( getmin(2*i, auxl, auxmid, l, r), getmin(2*i+1, auxmid+1, auxr, l, r));
            }
        }

        void update(int i, int v, int x, int l, int r){
            if(l == r){
                st[i] = x;
            }
            else{
                int mid = (r+l)/2;
                if(v <= mid){
                    update(2*i, v, x, l, mid);
                }
                else{
                    update(2*i+1, v, x, mid+1, r);
                }
                st[i] = min(st[2*i],st[2*i + 1]);
            }
            return;
        }
};

int main(){
    int n, q; cin >> n >> q;
    MinSegTree seg;
    for(int i = 1; i < n+1; i++){
        cin >> a[i];
    }
    seg.build(1,1,n);
    for(int i = 0; i < q; i++){
        int op; cin >> op;
        if(op == 1){
            int v, x; cin >> v >> x;
            seg.update(1,v,x,1,n);
        }
        else{
            int l, r; cin >> l >> r;
            cout << seg.getmin(1,1,n,l,r) << "\n";
        }
    }
}
```

3.8 Segtree Lazy Propagation

```
const int MAX = 2e5+20;
vector<int> lazy(4*MAX, 0);
int tree[4*MAX], vet[MAX];
int n;

int merge(int a, int b){
    return a + b; //seg de soma
}

void build(int l=0, int r=n-1, int no=1){
    if(l==r){
        tree[no] = vet[l];
        lazy[no] = 0;
        return;
    }
    int mid = (l+r)/2;
    build(l, mid, 2*no);
    build(mid+1, r, 2*no+1);

    tree[no] = merge(tree[2*no], tree[2*no+1]);
}

void prop(int l, int r, int no){
    if(lazy[no]!=0){
        tree[no] += (r-l+1)*lazy[no]; //update de soma
        if(l!=r){
            lazy[2*no] += lazy[no]; //update de soma
            lazy[2*no+1] += lazy[no]; //update de soma
        }
        lazy[no] = 0;
    }
}

void update(int A, int B, int x, int l=0, int r=n-1, int no=1){
    prop(l, r, no);
    // caso 1
    if(B<l or r<A) return;
    // caso 2
    if(A<=l and r<=B){
        lazy[no] += x; //update de soma
        prop(l, r, no);
        return;
    }
    // caso 3
    int mid = (l+r)/2;

    update(A, B, x, l, mid, 2*no);
    update(A, B, x, mid+1, r, 2*no+1);

    tree[no] = merge(tree[2*no], tree[2*no+1]);
}

int query(int A, int B, int l=0, int r=n-1, int no=1){
    prop(l, r, no);
    // caso 1
    if(B<l or r<A) return 0;
    // caso 2
    if(A<=l and r<=B) return tree[no];
    // caso 3
    int mid = (l+r)/2;

    return merge(query(A, B, l, mid, 2*no),
                 query(A, B, mid+1, r, 2*no+1));
}
```

```
int32_t main(){
    int q;
    cin>>n>>q;
    for(int i=0;i<n;i++) cin >> vet[i];
    build();
    while(q--){
        int opt; cin>>opt;
        if(opt == 1){
            int l, r, u;
            cin>>l>>r>>u;
            l--; r--;
            update(l, r, u);
        }else{
            int k; cin>>k;
            k--;
            cout<< query(k, k)<<endl;
        }
    }
    return 0;
}
```

4 Geometria

4.1 Geometria

```
const long double EPS = 1e-9;
typedef long double ld;

// point p(x, y);
struct point {
    ld x, y;
    int id;
    point(ld x=0, ld y=0): x(x), y(y){}

    point operator+(const point &o) const{ return {x+o.x, y+o.y}; }
    point operator-(const point &o) const{ return {x-o.x, y-o.y}; }
    point operator*(ld t) const{ return {x*t, y*t}; }
    point operator/(ld t) const{ return {x/t, y/t}; }
    ld operator*(const point &o) const{ return x * o.x + y * o.y; }
    ld operator^(const point &o) const{ return x * o.y - y * o.x; }
};

// line l(point(x1, y1), point(x2, y2));
struct line{
    point a, b;
    line(){}
    line(point a, point b) : a(a), b(b){}
};

// ponto e em relacao a linha l
// counterclockwise
int ccw(line l, point e){
    // -1=dir; 0=colinear; 1=esq;
    point a = l.b-l.a, b=e-l.a;
    ld tmp = a ^ b;
    return (tmp > EPS) - (tmp < -EPS);
}

// se o ponto ta em cima da linha
bool isinseg(point p, line l){
    point a = l.a-p, b = l.b-p;
    return ccw(l, p) == 0 and (a * b) <= 0;
}

// se o seg de r intersecta o seg de s
bool interseg(line r, line s) {
    if (isinseg(r.a, s) or isinseg(r.b, s)
        or isinseg(s.a, r) or isinseg(s.b, r)) return true;

    return (ccw(r, s.a)>0) != (ccw(r, s.b)>0) and
        (ccw(s, r.a)>0) != (ccw(s, r.b)>0);
}

// area do poligono
ld area_polygon(vector<point> vp){
    ld area = 0;
    for(int i=1; i<vp.size()-1; i++){
        area += (vp[0]-vp[i]) ^ (vp[0]-vp[i+1]);
    }
    return (abs(area)/2);
}

// localizacao do ponto no poligono
int point_polygon(vector<point> vp, point p){
    // -1=outside; 0=boundary; 1=inside;
    int sz = vp.size();
    int inter = 0;
    for(int i=0; i<sz; i++){
        int j = (i+1)%sz;
        line l(vp[i], vp[j]);
        if(isinseg(p, l)) return 0;

        if(vp[i].x <= p.x and p.x < vp[j].x and ccw(l, p) == 1) inter++;
        else if(vp[j].x <= p.x and p.x < vp[i].x and ccw(l, p) == -1) inter++;
    }

    if(inter%2==0) return -1;
    else return 1;
}
```

5 Grafos

5.1 Bellman Ford

```
/*
Algoritmo de busca de caminho minimo em um digrafo (grafo orientado ou dirigido) ponderado, ou seja, cujas arestas tem peso, inclusive negativo.
Acha ciclo negativo
O(V*E)
*/

int d[MAX];
int parent[MAX];
vector<pair<int,int>> adj[MAX];

int32_t main(){ sws;
    int n, m;
    cin>>n>>m;
    for(int i=1; i<=n; i++){
        d[i] = LLINF;
    }
    for(int i=0; i<m; i++){
        int a, b, c;
        cin>>a>>b>>c;
        adj[a].pb({b,c});
    }
    d[1] = 0;

    int src_cycle = -1;
    for(int j=1; j<=n and src_cycle; j++){
        src_cycle = 0;
        for(int u=1; u <= n; u++){
            for(auto [v, w]: adj[u]){
                if(d[u] + w < d[v]){
```

```
                    d[v] = d[u] + w;
                    parent[v] = u;
                    src_cycle = v;
                }
            }
        }
    }
    // there is no negative cycle
    if(!src_cycle){cout<<"NO"<<endl;}
    else {
        // there is negative cycle
        cout<<"YES"<<endl;
        vector<int> v;
        int a = src_cycle;
        for(int i = 0; i < n; i++)
            src_cycle = parent[src_cycle];

        int atual=src_cycle;
        while(true){
            v.pb(atual);
            if(atual == src_cycle && v.size()>1)break
    ;
            atual = parent[atual];
        }
        reverse(all(v));
        print(v, (int)v.size());
    }

    return 0;
}
```

5.2 Bfs

```
#include <bits/stdc++.h>[]
using namespace std;

//------------------------
#define MAXN 50050

int n, m;
bool visited[MAXN];
vector<int> lista[MAXN];
//------------------------

void bfs(int x){

    queue<int> q;
    q.push(x);
    while(!q.empty()){
        int v = q.front();
        q.pop();
        visited[v] = true;
        for(auto i : lista[v]){
            if(!visited[i]){
                q.push(i);
            }
        }
    }
}
```

5.3 Binary Lifting

```
vector<int> adj[MAX];
const int LOG = 30;
int up[MAX][LOG], parent[MAX];

void process(int n){
    for(int v=1; v<=n; v++){
        up[v][0]= parent[v];
        for(int i=1; i<LOG; i++){
            up[v][i] = up[ up[v][i-1] ][i-1];
        }
    }
}

int jump(int n, int k){
    for(int i=0; i<LOG; i++){
        if(k & (1 << i)){
            n = up[n][i];
        }
    }
    if(n == 0) return -1;
    return n;
}

int32_t main(){

    int n, q; cin>>n>>q;

    parent[1] = 0;
    for(int i=1; i<=n-1; i++){
        int x;
        cin>>x;
        parent[i+1] = x;

        adj[i+1].pb(x);
        adj[x].pb(i+1);
    }
    process(n);
    for(int i=0; i<q; i++){
        int a, b;
        cin>>a>>b;

        cout<<(jump(a,b))<<endl;
    }
}
```

5.4 Bridges

```
#include <bits/stdc++.h>
using namespace std;

#define endl "\n"
#define sws std::ios::sync_with_stdio(false); cin.tie
    (NULL); cout.tie(NULL);
#define pb push_back
const int MAX = 1e5+5;

vector<int> adj[MAX];
int timer=0;
int low[MAX], tin[MAX];
bool bridge=false;
bool visited[MAX];

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v]){}
                //IS_BRIDGE(v, to);
        }
    }
}

int32_t main(){ sws;
    int n, m;
    cin>>n>>m;

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

```
        int a, b;
        cin>>a>>b;

        adj[a].pb(b);
        adj[b].pb(a);
    }
    for(int i=1; i<=n; i++){
        if(!visited[i]) dfs(i);
    }
    if(bridge)cout<<"YES"<<endl;
    else cout<<"NO"<<endl;

    return 0;
}
```

5.5 Dfs

```
#include <iostream>
#include <vector>
#include <stack>

using namespace std;

//------------------------
#define MAXN 50050

int n, m;
bool visited[MAXN];
vector<int> lista[MAXN];
//------------------------

void dfs(int x){
    visited[x] = true;
    for(auto i : lista[x]){
        if(!visited[x]){
            dfs(i);
        }
    }
}

void dfsStack(int x){
    stack<int> s;
    s.push(x);
    while(!s.empty()){
        int v = s.top();
        s.pop();
        visited[v] = true;
        for(auto i : lista[v]){
            if(!visited[i]){
                s.push(i);
            }
        }
    }
}
```

5.6 Dfs Tree

```
const int MAX = 1e5;
int desce[MAX], sobe[MAX], vis[MAX], h[MAX];
int backedges[MAX], pai[MAX];

// backedges[u] = backedges que comecam embaixo de ( ou =) u e sobem pra cima de u; backedges[u] == 0 => u eh ponte
void dfs(int u, int p) {
    if(vis[u]) return;
    pai[u] = p;
    h[u] = h[p]+1;
    vis[u] = 1;

    for(auto v : g[u]) {
        if(p == v or vis[v]) continue;
        dfs(v, u);
        backedges[u] += backedges[v];
    }
    for(auto v : g[u]) {
        if(h[v] > h[u]+1)
            desce[u]++;
        else if(h[v] < h[u]-1)
            sobe[u]++;
    }
    backedges[u] += sobe[u] - desce[u];
}
```

5.7 Diametro Arvore Bfs

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef vector<int> vi;
typedef pair<int,int> pii;
const int MAX = 1e5+10;
const ll MOD = 1e9+5;

vector<int> adj[MAX];

pair<int, int> bfs(int s, int N){

    vi dist(N + 1, MAX); dist[s] = 0;
    queue<int> q; q.push(s);
    int last = s;

    while(!q.empty()){
        auto u = q.front();q.pop();
        last = u;

        for(auto v: adj[u]){
            if(dist[v]==MAX){
                dist[v]=dist[u]+1;
                q.push(v);
            }
        }
    }

    return {last, dist[last]};
}

int diameter(int N){
    auto [v, _] = bfs(1, N);
    auto [w, D] = bfs(v, N);

    return D;
}
```

5.8 Diametro Arvore Dfs

```
// DIAMETRO ARVORE - DFS

#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef vector<int> vi;
typedef pair<int,int> pii;
const int MAX = 1e5+10;
const ll MOD = 1e9+5;
const ll LLINF = 0x3f3f3f3f3f3f3f3f;

int to_leaf[MAX];
int max_length[MAX];
int dist[MAX];

vector<int> adj(MAX);
```

```cpp
/*
void dfs(int u, int p, vector<int> &dist){
    for(auto [v, w] : adj[u]){
        if(v!=p){
            dist[v] = dist[u] + w;
            dfs(v, u, dist);
        }
    }
}

int solve(int n){
    vector<int> dist(n+1, 0);

    dfs(0, -1, dist);

    auto v = (int)(max_element(dist.begin(), dist.end()) - dist.begin());

    dist[v] = 0;
    dfs(v, -1, dist);

    return *max_element(dist.begin(), dist.end());
}*/

void dfs(int u, int p){
    vi ds;

    for(auto v: adj[u]){
        if(v==p)continue;

        dfs(v, u);
        ds.pb(to_leaf[v]);
    }
    sort(ds.begin(), ds.end());

    to_leaf[u] = ds.empty() ? 0 : ds.back() + 1;

    auto N = ds.size();

    switch(N){
        case 0:
            max_length[u]=0;
            break;
        case 1:
            max_length[u] = ds.back() + 1;
            break;
        default:
            max_length[u] = ds[N-1] + ds[N-2] + 2;
    }
}

int diameter(int root, int N){
    dfs(root, 0);

    int d=0;

    for(int u=1; u<=N; u++){
        d= max(d, max_length[u]);
    }
}
```

5.9 Dijkstra

```cpp
// Acha o menor caminho de um ponto inicial para todos os outros
// Complexidade: O(NlogN)

#include <bits/stdc++.h>
using namespace std;
#define ll long long
typedef pair<int,int> pii;

const int N = 100005;
const ll oo = 1e18;

ll d[N]; // vetor onde guardamos as distancias

int n; // numeros de vertices
vector<pair<int, ll>> adj[N];

void dijkstra(int start){
    for(int u = 1; u <= n; u++)
        d[u] = oo;

    priority_queue<pii,
    vector<pii>,
    greater<pii> > pq;

    d[start] = 0;
    pq.push({d[start], start});

    ll dt, w;
    int u, v;
    while(!pq.empty()){
        auto [dt, u] = pq.top(); pq.pop();
        if(dt > d[u]) continue;
        for(auto [v, w] : adj[u]){
            if(d[v] > d[u] + w){
                d[v] = d[u] + w;
                pq.push({d[v], v});
            }
        }
    }
}

int main(){

    // le o input, qnt de vertices, arestas
    // e vertice inicial(start)
    int start = 0; // inicial
    dijkstra(start);

    for(int u = 1; u <= n; u++){
        printf("Distancia de %d para %d: %lld\n", start, u, d[u]);
    }

}
```

5.10 Floyd Warshall

```cpp
/*
Algoritmo de caminho mais curto com todos os pares.
Complexidade: O(³N)
*/

#include <bits/stdc++.h>
using namespace std;

const int oo = 100000000; // infinito

int main(){

    int n, m; cin>>n>>m;

    vector<vector<int>> dist(n+1, vector<int> (n+1));

    for(int i=0; i<n+1; i++){
        for(int j=0; j<n+1; j++){
            dist[i][j] = oo;
        }
    }

    for(int i=0; i<n +1; i++){
        dist[i][i]=0;
    }
```

```
26
27      for(int i=0; i<m; i++){
28          int comeca, termina, custo;
29          cin>>comeca>>termina>>custo;
30
31              // grafo direcionado
32          dist[comeca][termina] = custo;
33      }
34
35      for(int k=1; k<=n; k++){ // intermediario
36          for(int i=1; i<=n; i++){
37              for(int j=1; j<=n; j++){
38                  //(i,k,j) = ir de i pra j passando
    por k;
39
40                  // relaxar distancia de i pra j
41                  dist[i][j] = min(dist[i][j], dist[i][
    k] + dist[k][j]);
42              }
43          }
44      }
45          return 0;
46  }
```

5.11 Kosaraju

```
1  // Acha componentes fortemente conexas
2  // ou seja, que tem caminho entre todos os pares de
       vertices
3  // O(n+m)
4  bool visited[MAX];
5  int n;
6  vector<int> adj[MAX];
7  vector<int> adj2[MAX];// grafo inverso
8  stack<int> st;
9  int conex[MAX]; // conexo de cada vertice
10 void dfs(int u){
11      visited[u] = true;
12      for(auto v : adj[u]){
13          if(!visited[v]) dfs(v);
14      }
15      st.push(u);
16 }
17 // strongly connected component
18 void scc(int u, int c){
19      // cout<<u<<" ";
20      visited[u] = true;
21      conex[u] = c;
22      for(auto v : adj2[u]){
23          if(!visited[v]) scc(v, c);
24      }
25 }
26
27 void kosaraju(){
28      for(int i=1; i<=n; i++)visited[i]=false;
29      for(int i=1; i<=n; i++) if(!visited[i]) dfs(i);
30
31      for(int i=1; i<=n; i++)visited[i]=false;
32      while(!st.empty()){
33          int u = st.top(); st.pop();
34          if(!visited[u]) scc(u, u);
35      }
36 }
37 int main(){
38      int m;
39      cin>>n>>m;
40
41      for(int i=0; i<m; i++){
42          int a, b;
43          cin>>a>>b;
44          adj[a].pb(b);
45          adj2[b].pb(a);
46      }
47
48      kosaraju();
49
50      return 0;
51 }
```

5.12 Kruskall

```
1  /*
2  Busca uma arvore geradora minima para um grafo conexo
       com pesos.
3  */
4
5  #include <iostream>
6  #include <algorithm>
7
8  using namespace std;
9
10 struct t_aresta{
11      int dis;
12      int x, y;
13 };
14
15 bool comp(t_aresta a, t_aresta b){ return a.dis < b.
       dis; }
16
17 //--------------------
18 #define MAXN 50500
19 #define MAXM 200200
20
21 int n, m; // numero de vertices e arestas
22 t_aresta aresta[MAXM];
23
24 // para o union find
25 int pai[MAXN];
26 int peso[MAXN];
27
28 // a arvore
29 t_aresta mst[MAXM];
30 //--------------------
31
32 // funcoes do union find
33 int find(int x){
34      if(pai[x] == x) return x;
35      return pai[x] = find(pai[x]);
36 }
37
38 void join(int a, int b){
39
40      a = find(a);
41      b = find(b);
42
43      if(peso[a] < peso[b]) pai[a] = b;
44      else if(peso[b] < peso[a]) pai[b] = a;
45      else{
46          pai[a] = b;
47          peso[b]++;
48      }
49
50 }
51
52
53 int main(){
54
55      // ler a entrada
56      cin >> n >> m;
57
58      for(int i = 1;i <= m;i++)
59          cin >> aresta[i].x >> aresta[i].y >> aresta[i
       ].dis;
60
61
62      // inicializar os pais para o union-find
```

```
    for(int i = 1;i <= n;i++) pai[i] = i;

    // ordenar as arestas
    sort(aresta+1, aresta+m+1, comp);

    int size = 0;
    for(int i = 1;i <= m;i++){

        if( find(aresta[i].x) != find(aresta[i].y) ){ // se estiverem em componentes distintas
            join(aresta[i].x, aresta[i].y);

            mst[++size] = aresta[i];
        }

    }

    // imprimir a MST
    for(int i = 1;i < n;i++) cout << mst[i].x << " " << mst[i].y << " " << mst[i].dis << "\n";
    return 0;
}
```

5.13 Lca

```
/*
Lowest Common ancestor (LCA) - dado uma Arvore cuja raiz eh um vertice arbitrario e dois vertices u,v que a pertencem, diga qual eh o no mais baixo(relativo a raiz) que eh ancestral de u,v.
*/
// Complexidades:
// build - O(n log(n))
// lca - O(log(n))

#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int SIZE = 2e5+5;
const int LOG = 30; // log2(SIZE)+1;
int depth[SIZE];
//ll weight[SIZE];
int pai[SIZE];
vector<pair<int,int>> adj[SIZE];
int up[SIZE][LOG];

//
void dfs(int u, int p) {
    for(auto edge : adj[u]) {
        int v, w;
        tie(v, w) = edge;
        if(v != p){
            up[v][0] = u;
            //weight[v] = weight[u] + w;
            depth[v] = depth[u] + 1;
            for(int i=1; i<LOG; i++){
                up[v][i] = up[ up[v][i-1] ][i-1];
            }
            dfs(v, u);
        }
    }
}

int lca(int a, int b) {
    if(depth[a] < depth[b]) swap(a,b);
    int k = depth[a] - depth[b];
    for(int i=0; i<LOG; i++){
        if(k & (1 << i)){
            a = up[a][i];
        }
    }
    if(a == b) return a;
    for (int i = LOG-1; i >= 0; i--) {
        if(up[a][i] != up[b][i]) {
            a = up[a][i];
            b = up[b][i];
        }
    }
    return up[a][0];
}

ll dist(int u, int v){
    return depth[u] + depth[v] - 2*depth[lca(u,v)];
    // return weight[u] + weight[v] -2*weight[lca(u,v)];
}

int main() {
    int n; cin>>n;

    for(int i=0; i<n-1; i++){
        int x, y, z;
        cin>>x>>y>>z;
        adj[x].push_back({y, z});
        adj[y].push_back({x, z});
    }
    // raiz
    dfs(1, 0);

    int q; cin>>q;
    while(q--){
        int a, b, c;
        cin>>a>>b>>c;
        long long x = dist(a, b) + dist(b, c);
        cout<<x<<endl;
    }
}
```

5.14 Topo Sort

```
// topological sort
// retorna uma ordenacao topologica
// caso for um dag, se nao, retorna vazio se tiver ciclo
// O(n+m)
// indexado em 1 os vertices

int n;
int visited[MAX];
vector<int> adj[MAX];
int pos=0;
vector<int> ord;
bool has_cycle=false;

void dfs(int v){
    visited[v] = 1;
    for(auto u : adj[v]){
        if(visited[u] == 1) has_cycle=true;
        else if(!visited[u]) dfs(u);
    }
    ord[pos--] = v;
    visited[v] = 2;
}

vector<int> topo_sort(int n){
    ord.assign(n, 0);
    has_cycle = false;
    pos = n-1;
    for(int i=1; i<=n; i++){
        if(!visited[i]) dfs(i);
    }

    if(has_cycle) return {};
    else return ord;
}

```

```cpp
int main(){
    int m;
    cin>>n>>m;

    for(int i=0; i<m; i++){
        int a, b;
        cin>>a>>b;
        adj[a].pb(b);
    }

    vector<int> ans = topo_sort(n);

    return 0;
}
```

6 Math

6.1 Combinatoria

```cpp
// quantidade de combinacoes possiveis sem repeticao de 2 numeros
int comb(int k){
    if(k==1 or k==0)return 0;
    return (k*(k-1))/2;
}
```

6.2 Divisibilidade

```cpp
// 2 -> se eh par
// 3 -> se a soma dos algarismos eh divisivel por 3
// 4 -> se os dois ultimos algarismos eh divisivel por 4
// 5 -> se o ultima algarismo eh 0 ou 5
// 6 -> se eh par e a soma dos algarismos eh divisivel por 3
// 7 -> se o dobro do ultimo algarismo subtraido do numero sem o ultimo algarismo eh divisivel por 7
// 8 -> se os 3 ultimos algarismos eh divisivel por 8
// 9 -> se a soma dos algarimos eh divisivel por 9
// 10 -> se o ultimo algarimo eh 0
```

6.3 Divisores

```cpp
#include <bits/stdc++.h>
using namespace std;

vector<long long> get_divisors(long long n){
    vector<long long> divs;
    for(long long i = 1; i*i <=n; i++){
        if(n%i == 0){
            divs.push_back(i);
            long long j = n/i;
            if(j != i)
                divs.push_back(j);
        }
    }
    return divs;
}
```

6.4 Fatora

```cpp
map<int,int> fatora(int n){
    map<int,int> fact;
    for(int i = 2; i*i <= n; i++){
        while(n%i == 0){
            fact[i]++;
            n /= i;
        }
    }
    if(n > 1)
        fact[n]++;
    return fact;
}
```

6.5 Mdc

```cpp
// Greatest common divisor / MDC

long long gcd(long long a, long long b){
    return b ? gcd(b, a % b) : a;
}

// or just use __gcd(a,b)
```

6.6 Mmc

```cpp
// Least Common Multiple - MMC
#include <bits/stdc++.h>
using namespace std;

long long lcm(long long a, long long b){
    return (a/__gcd(a,b)*b);
}
```

6.7 Pa

```cpp
// Termo Geral
// An = A1 + (n-1)*d

// Soma
// Sn = (n/2)(2*A1+(n-1)*d)

// óSomatrio de 1 a K
int pa(int k){
    return (k*(k+1))/2;
}
```

6.8 Pg

```cpp
// Termo Geral
// An = A1 * r^(n-1)

// Soma
// (A(r^(n)-1))/(r-1)
```

6.9 Pollard-rho

```cpp
// O(sqrt(N) * logN)

ll a[MAX];

ll mul(ll a, ll b, ll m){
    ll ret = a*b - (ll)((long double)1/m*a*b+0.5)*m;
    return ret < 0 ? ret+m : ret;
}

ll pow(ll x, ll y, ll m) {
    if (!y) return 1;
    ll ans = pow(mul(x, x, m), y/2, m);
    return y%2 ? mul(x, ans, m) : ans;
}

bool prime(ll n) {
    if (n < 2) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0) return 0;

    ll r = __builtin_ctzll(n - 1), d = n >> r;
    for (int a : {2, 325, 9375, 28178, 450775, 9780504, 795265022}) {
        ll x = pow(a, d, n);
```

```
        if (x == 1 or x == n - 1 or a % n == 0)
    continue;

        for (int j = 0; j < r - 1; j++) {
            x = mul(x, x, n);
            if (x == n - 1) break;
        }
        if (x != n - 1) return 0;
    }
    return 1;
}

ll rho(ll n) {
    if (n == 1 or prime(n)) return n;
    auto f = [n](ll x) {return mul(x, x, n) + 1;};

    ll x = 0, y = 0, t = 30, prd = 2, x0 = 1, q;
    while (t % 40 != 0 or __gcd(prd, n) == 1) {
        if (x==y) x = ++x0, y = f(x);
        q = mul(prd, abs(x-y), n);
        if (q != 0) prd = q;
        x = f(x), y = f(f(y)), t++;
    }
    return __gcd(prd, n);
}

vector<ll> fact(ll n) {
    if (n == 1) return {};
    if (prime(n)) return {n};
    ll d = rho(n);
    vector<ll> l = fact(d), r = fact(n / d);
    l.insert(l.end(), r.begin(), r.end());
    return l;
}


int main(){
    set<ll> primes;
    int M, N, K; cin >> M >> N >> K;
    loop(i,0,N){
        cin >> a[i];
        vector<ll> aprimes = fact(a[i]);
        for(auto prime : aprimes){
            primes.insert(prime);
        }
    }
    int m, n, d;
    loop(i,0,K) cin >> m >> n >> d;
    for(auto prime : primes){
        cout << prime << " ";
    }
}
```

6.10 Primos

```
// PRIMALIDADE

#include <bits/stdc++.h>
using namespace std;

const int MAX = 1e5+7;

void crivo(){
    vector<int> crivo(MAX, 1);
    for(int i=2; i*i<=MAX; i++){
        if(crivo[i]==1){
            for(int j=i+i; j<MAX; j+=i){
                crivo[j]=0;
            }
        }
    }
}

bool is_prime(int num){
    for(int i = 2; i*i<= num; i++) {
       if(num % i == 0) {
          return false;
       }
    }
    return true;
}
```

7 Strings

7.1 Suffix Array

```
#include <bits/stdc++.h>
using namespace std;

#define ll long long
#define sws ios::sync_with_stdio(false);cin.tie( NULL );cout.tie(NULL);
#define print(x) for (auto &it : x) cout<<it<<' ';<<cout<<endl;
#define loop(i,a,n) for(int i=a; i < n; i++)
#define pb(x) push_back(x);
#define vi vector<int>
#define mp(x,y) make_pair(x,y)
#define pii pair<int,int>
#define pqi priority_queue<int, vector<int>, greater<int>>
const ll MOD = 1e9+7;
const int INF = 0x3f3f3f3f;
const ll LLINF = 0x3f3f3f3f3f3f3f3f;

vector<int> suffix_array(string s) {
    s += "$";
    int n = s.size(), N = max(n, 260);
    vector<int> sa(n), ra(n);
    for (int i = 0; i < n; i++) sa[i] = i, ra[i] = s[i];

    for (int k = 0; k < n; k ? k *= 2 : k++) {
        vector<int> nsa(sa), nra(n), cnt(N);

        for (int i = 0; i < n; i++) nsa[i] = (nsa[i]-k+n)%n, cnt[ra[i]]++;
        for (int i = 1; i < N; i++) cnt[i] += cnt[i-1];
        for (int i = n-1; i+1; i--) sa[--cnt[ra[nsa[i]]]] = nsa[i];

        for (int i = 1, r = 0; i < n; i++) nra[sa[i]] = r += ra[sa[i]] !=
            ra[sa[i-1]] or ra[(sa[i]+k)%n] != ra[(sa[i-1]+k)%n];
        ra = nra;
        if (ra[sa[n-1]] == n-1) break;
    }
    return vector<int>(sa.begin()+1, sa.end());
}

vector<int> kasai(string s, vector<int> sa) {
    int n = s.size(), k = 0;
    vector<int> ra(n), lcp(n);
    for (int i = 0; i < n; i++) ra[sa[i]] = i;

    for (int i = 0; i < n; i++, k -= !!k) {
        if (ra[i] == n-1) { k = 0; continue; }
        int j = sa[ra[i]+1];
        while (i+k < n and j+k < n and s[i+k] == s[j+k]) k++;
        lcp[ra[i]] = k;
    }
    return lcp;
```

```
}


int32_t main(){
    sws;
    string s;
    cin>>s;

    vector<int> suf = suffix_array(s);
    vector<int> lcp = kasai(s, suf);

    ll ans = 0;
    for(int i=0; i<s.size(); i++){
        if(islower(s[suf[i]])){
            int sz = s.size()-suf[i];
            ans += (sz - lcp[i]);
        }
    }
    cout<<ans<<endl;
}
```

7.2 Trie

```
// Constroi e procura por uma string em uma arvore
// Trie t;
// Trie t(qtd_char, c_min, max_size)
// qtd_char = qntd maxima de caracteres
// c_min = menor caractere
// max_size = tamanho maximo de strings

// Complexidade - O(N*|s|*qtd_char)

#include <bits/stdc++.h>
using namespace std;

#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
const int MAX = 2005;

class Trie{
    int node = 1;
    char c_min;
    int qtd_char, max_size;
    vector<vector<int>> trie;
    vector<int> pref, end;

    public:
    void add(string s){
        int cur = 1;
        for(auto c: s){
            if(!trie[cur][c-c_min]){
                trie[cur][c-c_min] = ++node;
            }
            cur = trie[cur][c-c_min];
            pref[cur]++;
        }
        end[cur]++;
    }

    void erase(string s){
        int cur = 1;
        for(auto c: s){
            cur = trie[cur][c-c_min];
            pref[cur]--;
        }
        end[cur]--;
    }

    int find(string s){
        int cur = 1;
        for(auto c: s){
            if(!trie[cur][c-c_min]) return 0;
            cur = trie[cur][c-c_min];
        }
        return cur;
    }

    int count_pref(string s){
        return pref[find(s)];
    }

    Trie(int qtd_char_=26, char c_min_ = 'a', int max_size_=MAX):
    c_min(c_min_), qtd_char(qtd_char_), max_size(max_size_){
        trie.resize(max_size, vector<int>(qtd_char));
        pref.resize(max_size);
        end.resize(max_size);
    }
};

int32_t main(){ sws;
    Trie t;
    t.add("abcd");
    t.add("ad");
    t.erase("ad");
    cout<<t.count_pref("a")<<endl;

    return 0;
}
```

8 Template

8.1 Template

```
#include <bits/stdc++.h>
using namespace std;
//alias comp='g++ -std=c++17 -g -O2 -Wall -Wconversion -Wshadow -fsanitize=address,undefined -fno-sanitize-recover -ggdb -o out'

#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define int long long
#define endl "\n"
#define loop(i,a,n) for(int i=a; i < n; i++)
#define input(x) for (auto &it : x) cin >> it
#define pb push_back
#define all(x) x.begin(), x.end()
#define ff first
#define ss second
#define mp make_pair
#define TETO(a, b) ((a) + (b-1))/(b)
#define dbg(msg, x) cout << msg << " = " << x << endl
#define print(x,y) loop(i,0,y){cout << x[i] << " ";}
    cout << "\n";

typedef long long ll;
typedef long double ld;
typedef vector<int> vi;
typedef pair<int,int> pii;
typedef priority_queue<int, vector<int>, greater<int>> pqi;

const ll MOD = 1e9+7;
const int MAX = 1e4+5;
const ll LLINF = 0x3f3f3f3f3f3f3f3f;
const double PI = acos(-1);


int32_t main(){ sws;


    return 0;
}
```

9 zExtra

9.1 Getline

```cpp
#include <bits/stdc++.h>
using namespace std;
// Sempre usar cin.ignore() entre um cin e um getline
int main() {

    string s1; cin>>s1;
    cin.ignore();
    while (true) {
        string s; getline(cin, s);
        if (s == "PARO") break;
        cout<<"A"<<endl;
    }
    string s2; cin>>s2;
    cin.ignore();
    while (true) {
        string s3; getline(cin, s3);
        if (s3 == "PARO") break;
        cout<<"A"<<endl;
    }
}
```