Cervejeiros do Mendes

Caue Trindade, Guga Tonnera & Vini Araujo

Contents

1 Datastructures

1.1 Dsuclass

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

using namespace std;

class DSU
{
    vector<int> parent;
    vector<int> card;

public:
    DSU(int n): parent(n+1), card(n+1,1)
    {
        for(int i = 1; i <= n; i++)
            parent[i] = i;
    }

    /* O(log n) */
    int find_set(int x)
    {
        if(x == parent[x])
            return x;

        return parent[x] = find_set(parent[x]);
    }

    bool same_set(int a, int b)
    {
        return find_set(a) == find_set(b);
    }

    /* O(log n) */
    void join_sets(int a, int b)
    {
        a = find_set(a);
        b = find_set(b);

        if(card[a] < card[b])
            swap(a,b);

        card[a] += card[b];
        parent[b] = a;
    }
};
```

1.2 Segtree

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

using namespace std;

#define ll long long
#define vi vector<int>
#define pii pair<int,int>


//passar vetor indexado em 1

class SegTree {
    vector<ll> st;
    vector<ll> v;
    int size;
    int elem_neutro = 0;// changes based on the segtree function

public:

    SegTree(vector<ll> arr, int size): st(4*size,0), v(size+1,0) {
        this->size = size;

        for(int i = 1; i<size+1; i++){
            v[i] = arr[i];
        }
    }

    ll f(ll a, ll b){ //type and return of function are variable, depending on the segtree
        return a+b;
    }

    void build(int l, int r, int nodo){
        if(l ==r){
            st[nodo] = v[l];
            return;
        }
        int m = (l+r)/2;


        build(l,m,nodo*2); //desceu p esquerda
        build(m+1,r,nodo*2+1); //desceu p direita


        st[nodo] = f(st[nodo*2], st[nodo*2+1]);
    }

    void update_range(int i, int l, int r, ll x, int nodo) {
        if(l == r){
            st[nodo]  = x;
            return;
        }

        int m = (l+r)/2;

        if(i <=m){
            update_range(i, l, m, x, nodo*2);
        }
        else{
            update_range(i,m+1,r,x,nodo*2+1);
        }

        st[nodo] = f(st[nodo*2],st[nodo*2+1]);
    };

    void update(int i, ll x){
        int l =1;
        int r = size;
        int nodo = 1;
        return update_range(i,l,r,x,nodo);
    }

    ll query_range(int ql, int qr, int l, int r, int nodo){
        if(l >qr or r < ql){
            return elem_neutro;
        }

        if( l >= ql and r <= qr){
            return st[nodo];
        }

        int m = (l+r)/2;
        ll suml = query_range(ql,qr,l,m,nodo*2);
        ll sumr = query_range(ql,qr,m+1,r,nodo*2+1);
        return f(suml,sumr);
    }

    ll query(int ql, int qr){
        int l = 1;
        int r = size;
        int nodo =1;
        return query_range(ql,qr,l,r,nodo);
    }
};

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    return 0;
    }
```

1.3 Sparsetable

```cpp
#include <bits/stdc++.h>
#define ll long long
#define vi vector<int>
#define vll vector<long long>
using namespace std;


// Complexity: The pre proccessing is O(nlog(n)). Most queries such as sum are O(log(n));
// So in that case, it is pretty much always better to use a segTree.
// However, when the queries are min or max, the complexity of each query becomes O(1);

class SparseTable
{
    int logn;
    vector<vll> tb;
    vi logs;

public:
    // Constructor takes only ‘n‘ and ‘arr‘ as arguments
    SparseTable(int n, vll arr) : logs(n + 1), tb(log2(n) + 1, vll(n)) {
        // Build the ‘logs‘ array
        logs[1] = 0;
        for (int i = 2; i <= n; i++) {
            logs[i] = logs[i / 2] + 1;
        }

        // Set logn to the maximum power needed for ‘n‘
        this->logn = logs[n];

        // Initialize the first row of ‘tb‘ with ‘arr‘ values
        for (int i = 0; i < n; i++) {
            tb[0][i] = arr[i];
        }

        // Build the sparse table
        for (int k = 1; k <= logn; k++) {
            for (int i = 0; i + (1 << k) <= n; i++) {
                tb[k][i] = min(tb[k - 1][i], tb[k - 1][i + (1 << (k - 1))]);
            }
        }
    }

    // Range minimum query on the range [l, r]
    ll query(int l, int r) {
        //if not indexed in 1 remove this.
        l--;
        r--;
        int len = r - l + 1;
        int k = logs[len];
        return min(tb[k][l], tb[k][r - (1 << k) + 1]);
    }
};

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int n, q;
    cin >> n >> q;
    vll arr(n);

    // Initializing array;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    // Initialize the Sparse Table with the array
    SparseTable sp(n, arr);

    // Perform range minimum queries
    for (int i = 0; i < q; i++) {
        int l, r;
        cin >> l >> r;
        cout << sp.query(l, r) << endl;
    }

    return 0;
}
```

2 Graphs

2.1 Djikstra

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

using namespace std;


#define pii pair<int, int>
#define ll long long


int N;
const int oo = 1e6+7; // depende

vector<vector<pii>> g(N);
vector<bool> used(N);
vector<ll> d(N, oo);
priority_queue< pii, vector<pii>, greater<pii> > q;

void dijkstra(int k) {
    d[k] = 0;
    q.push({0, k});

    while (!q.empty()) {
        auto [w, u] = q.top();
        q.pop();
        if (used[u]) continue;
        used[u] = true;

        for (auto [v, w]: g[u]) {
            if (d[v] > d[u] + w) {
                d[v] = d[u] + w;
                q.push({d[v], v});
            }
        }
    }
}
```

2.2 Binarylifting

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

```cpp

using namespace std;

#define vi vector<int>



class TreeAncestor {
    int LOG;
    vector<vi> up;  //[n][log] -> o antecessor de n em 2^log
    vi depth;

public:
    TreeAncestor(int n, vi& pai){
        LOG =0;
        while((1 << LOG)<= n){
            LOG++;
            //formula de cÃąlcular o log
        }
        up = vector<vi>(n,vi(LOG));
        depth = vi(n);

        // pai[i]<i:
        pai[0] =0;
        for(int v =0; v<n; v++){
            up[v][0]=pai[v];
            if(v !=0) depth[v] =depth[pai[v]]+1;
            for(int j = 1; j<LOG; j++){
                up[v][j] = up[ up[v][j-1]][j-1];
            }
        }

    }


    int getAncestralK (int nodo, int k){

        if(depth[nodo]<k){
            return -1; //impossÃŋvel
        }

        // 1 << j = 2^j

        for(int j = LOG-1; j>=0; j--){
            if(k >= (1 << j)){
                nodo = up[nodo][j];
                k -= (1 << j);
            }
        }

        return nodo;
    }
};
```

2.3 Diameter

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

using namespace std;

const int N = 1e4; //tamanho da Ãąrvore

vector<vector<int>> g(N); // Ãąrvore por lista de adjacÃłncias;

void dfs(int x, int p, int l, int& longestPath, int& nodo){
    for(auto v : g[x]){
        if(v==p) continue;
        dfs(v,x,l+1,longestPath,nodo);
    }
    if(l>longestPath){
        longestPath = l;
        nodo = x;
    }
}

//x = primeiro nÃş da Ãąrvore
pair<int,pair<int,int>> findDiameter(int x){
    //g = tree por lista de adjacÃłncia
    int nodo = x;
    int longestPath =0;

    dfs(x,-1,1,longestPath,nodo);

    longestPath =0;
    x = nodo;

    dfs(x, -1,1, longestPath, nodo);

    return make_pair(longestPath, make_pair(x,nodo));
}
```

2.4 Eulertour

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

using namespace std;

/*
    Utilidades para o Euler Tour:
    o menor subarray entre dois elementos da array de EulerTour Ãľ o caminho entre esses dois vÃľrtices
    com isso, Ãľ possÃŋvel descobrir vÃąrias coisas, como:
        qual o lca range (o depth mÃŋnimo);
        qual a soma de distÃćncias do caminho entre dois vÃľrtices
        qual a maior distÃćncia do caminho entre dois vÃľrtices
        dentre outros
*/

const int N = 1e5;

vector<vector<int>> g(N+1);
vector<int> depth(N+1); // definir depth[raiz] =1;
vector<int> eulerTour; // vai ter 2*N-1 elementos;
vector<int> first(N+1,-1); // primeira vez que um nÃş aparece no euler tour
vector<vector<int>> appears(N+1); //posiÃğÃţes em que um nÃş aparece no eulertour

void dfs(int u){

    if(first[u]==-1){
        first[u]=eulerTour.size();
    }

    appears[u].push_back(eulerTour.size());
    eulerTour.push_back(u);

    for(auto v : g[u]){
        depth[v] = depth[u]+1;
        dfs(v);
        appears[u].push_back(eulerTour.size());
        eulerTour.push_back(u);
    }
}

void print(){
    cout << eulerTour.size() << endl;
    for(auto x : eulerTour) cout << x << " ";
    cout << endl;
    for(auto x : eulerTour) cout << depth[x] << " ";
    cout << endl;
    for(int x = 1; x <9; x++) cout << first[x] << " ";

}

int main() {
    g[1].push_back(2);
    g[2].push_back(3);
    g[2].push_back(4);
    g[4].push_back(5);
    g[1].push_back(6);
    g[6].push_back(7);
    g[6].push_back(8);

    depth[1]= 1;
    dfs(1);

    print();

    return 0;
}
```

2.5 Kosaraju

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

using namespace std;

/*
    Algoritmo de Kosaraju:
    contexto : grafo direcionado;
    objetivo: encontrar componentes fortemente conectados* no grafo
        *fortemente conectado: Ãľ possÃŋvel chegar em todos os nÃş saindo de qualquer nÃş;
        *componente fortemente conectado (SCC): maior quantidade possÃŋvel de vÃľrtices fortemente conectados no grafo

    Um grafo G direcionado pode ser representado por um grafo acÃŋclico direcionado S onde cada nÃş de S Ãľ um SCC de G;
        *S == G, se G Ãľ acÃŋclico

    ExecuÃğÃčo:
    encontra a pÃşs-ordem por DFS;
    cria um grafo Ginv igual a G mas com a direÃğÃčo das arestas trocada.
        Obs, um componente fortemente conectado em G tambÃľm Ãľ em Ginv;
    Performa DFS em Ginv para encontrar os SCC diferentes.
*/
const int N = 1e5+1;
vector<vector<int>> g(N), ginv(N); //grafo original, inverso e resultado do kosaraju;
vector<int> vis(N,0);
vector<int> pai(N); //nÃş que vai representar o componente conexo (noÃğÃčo de dsu);
vector<int> scc; //lista de representantes de componentes scc;
stack<int> posord;

void dfs(int u){
    vis[u]=1;
    for(auto v: g[u]){
        if(!vis[v]) dfs(v);
    }
    posord.push(u);
}

void invdfs(int u){
    vis[u] =1;
    for(auto v: ginv[u]){

        if(!vis[v]){
            pai[v] = pai[u];
            invdfs(v);
        }
    }
}

void kosaraju(int n){
    for(int i = 1; i<=n ; i++) vis[i]=0;

    for(int i =1; i<=n; i++) if(!vis[i]) dfs(i);

    for(int i =1; i<=n; i++) vis[i]=0;

    while(!posord.empty()){
        int u = posord.top(); posord.pop();
        if(vis[u]) continue;

        pai[u] = u;
        scc.push_back(u);
        invdfs(u);
    }

}
```

2.6 Kruskal

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

using namespace std;

/*
Kruskal:
percorre todas as arestas em ordem,
se a ordem for crescente retorna a minimum spanning tree;
se a ordem for decrescente retorna a maximum spanning tree;
se os dois vÃľrtices que cada aresta liga jÃą estÃčo conectados, pula essa aresta;
se nÃčo, conecta os dois componentes e adiciona essa aresta;

a prova de que funciona Ãľ meio trivial, e a implementaÃğÃčo Ãľ bem tranquila com dsu,
sÃş to adicionando ai pra jÃą ter uma que retorna mastigadinho com a Ãąrvore montada por adj;
*/

class DSU
{
    vector<int> parent;
    vector<int> card;

public:
    DSU(int n): parent(n+1), card(n+1,1)
    {
        for(int i = 1; i <= n; i++)
            parent[i] = i;
    }

    /* O(log n) */
    int find_set(int x)
    {
        if(x == parent[x])
            return x;

        return parent[x] = find_set(parent[x]);
    }

    bool same_set(int a, int b)
    {
        return find_set(a) == find_set(b);
    }

    /* O(log n) */
    void join_sets(int a, int b)
    {
        a = find_set(a);
        b = find_set(b);

        if(card[a] < card[b])
            swap(a,b);

        card[a] += card[b];
        parent[b] = a;
    }
};

// n = quantidade de vÃľrtices;
// retorna tree com tree[x] = {weigth, vertice};

vector<vector<pair<int,int>>> Kruskal(vector<pair<int,pair<int,int>>>& arestas, int n){
    DSU d(n);
    vector<vector<pair<int,int>>> tree(n+1);

    for(auto [w,p] : arestas){
        if(d.same_set(p.first, p.second)) continue;
        d.join_sets(p.first, p.second);
        tree[p.first].push_back({w,p.second});
        tree[p.second].push_back({w,p.first});
    }

    return tree;
}
```

2.7 Lca

```
//LCA com binary lifting;
#include <bits/stdc++.h>

using namespace std;

#define vi vector<int>

int MAX_N = 1e6;
int LOG = 20;

vector<vi> children(MAX_N,vi(MAX_N));
vector<vi> up(MAX_N, vi(LOG));
vi depth(MAX_N);

//definir up[a]=a antes;
//construtor se passarmos o a como raiz;
void dfs(int a){
    for(int b: children[a]){
        up[b][0] =a;
        depth[b] =depth[a] +1;
        for(int j =1; j<LOG; j++){
            up[b][j]= up[ up[b][j-1] ][j-1];
        }
        dfs(b);
    }
}

int jump(int a, int k){
    for(int j = LOG-1; j>=0; j--){
        if(k >= (1<<j)){
            a = up[a][j];
            k -= (1 << j);
        }
    }
    return a;
}

 int getAncestralK (int nodo, int k){

        if(depth[nodo]<k){
            return -1; //impossÃŋvel
        }

        // 1 << j = 2^j

        for(int j = LOG-1; j>=0; j--){
            if(k >= (1 << j)){
                nodo = up[nodo][j];
                k -= (1 << j);
            }
        }

        return nodo;
    }

int find_lca(int a, int b){

    if(depth[a]< depth[b]){
        swap(a,b); // a sempre o mais profundo;
    }
    int k = depth[a] - depth[b];
    a = jump(a,k);

    if(a == b) return a;

    for(int j = LOG-1; j>=0; j--){
        if(up[a][j]!= up[b][j]){
            a = up[a][j];
            b = up[b][j];
        }
    }
    return up[a][0];
}

int dist(int a, int b){
    return depth[a] + depth[b] - 2*depth[find_lca(a,b)];
}
```

3 Maths

3.1 Catalan

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

using namespace std;


#define ll long long
#define vll vector<long long>


vll fac;

/* C(n) = Combinacao(2n) * 1/(n+1)
                    ( n)

C(n) = nÂž de sequencias a com n elementos +1 e n elementos -1,
tal que p todo k, 1<=k <=2n; a1 + a2 +...+ak >=0;
AplicaÃğÃţes:
sequencias de parenteses
qtd de arvores binarias
etc depois escrevo o resto
*/
```

```

ll Catalan(ll n){
    return Comb(2*n,n,mod)*expMod(n+1,mod-2,mod);
}
```

3.2 Chineseremainder

```
/*
com todos m primos entre si, a equaÃ§Ãčo :

x= a1 mod m1
x = a2 mod m2
x = a3 mod m3
x = an mod mn

Ãľ resolvida por x0 = a1*X1*inv_mod_m1(X1) + ... + an*Xn*inv_mod_mn(Xn);
sendo Xk = (m1*m2*m3*...*mk*...*mn)/mk
ou seja, Xk Ãľ um nÃžmero divisÃŋvel por todos m menos por mk, e por isso
a expressÃčo resolve. Mais mastigado que isso Ãľ com o leitor kkkkkkkk

x_geral = x0 + k(m1*m2*m3*...*mn),    k inteiro

*/
```

3.3 Diofantineequation

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

using namespace std;

/* LINEAR DIOFANTINE EQUATIONS
ax + by + c with a,b and c given integers solving for x and y integers.

by .\gcdExtended.cpp, we can find ax' + by' = g with g = gcd(a,b);
    therefore, if c is divisible by g, there are solutions, else there are none.
    ( a = t.g , b = r.g, c = t.g + r.g = (t+r).g)
from that, follows:
ax'.(c/g) + by'.(c/g) = c
a solution then is:
ax0 + by0 = c;
x0 = x'.(c/g)
y0 = y'.(c/g)

to generalize it, take the equation:
a(x0 + b/g) + b(y0 - a/g) =c;
ax0 + by0 + ab/g - ab/g = c; it's still true since g = gcd(a,b) then:
ax+ by = c;
x = x0 + k.(b/g);
y = y0 + k.(y/g);

*/
```

3.4 Eulertotient

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

/*FunÃ§Ãčo totiente de Euler phi(n);
retorna a quantidade de nÃžmeros coprimos no intervalo fechado [1,n];

propriedades:
    1-> para p primo: phi(p) = p-1;
    2-> para p primo e k >=1: existem (p^k) * 1/p; nÃžmeros em [1,p^k] divisÃŋveis por p; -> phi(p^k)=p^k - p^k-1;
    3-> para a e b coprimos, phi(a*b) = phi(a) * phi(b); (discorre do teorema do resto chinÃłs);

    Portanto, sendo p_1^k_1 * p_2^k_2*...*p_i^k_i = n a fatoraÃ§Ãčo em primos de um nÃžmero n,
    phi(n) = phi(p_1^k_1)*...*phi(p_i^k_i) [Prop 3]
    phi(n) = (p_1^k_1 - p_1^(k_1 -1))* ... * (p_i^k_i - p_i^(k_i -1)) [Prop 2]
    phi(n) = p_1^k_1(1 - 1/p_1)*...*p_i^k_i(1 - 1/p_i)
    phi(n) = p_1^k1*p2_k2*...*p_i^ki*(1- 1/p_1)*(1- 1/p_2)*(1- 1/p_i);

*/

//O(sqrt(n))
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }

    if (n > 1) // n Ãľ primo;
        result -= result / n;
    return result;
}

// Faz o que a funÃ§Ãčo phi faz (n*(1- 1/p_i) = n - n/p_i) para cada fator primo p_i de n;
//O(nlog log n)
vector<int> phi_sieve(int n){
    vector<int> phi(n+1);
    for(int i = 0; i<=n; i++){
        phi[i]=i;
    }
    for(int i =2; i<=n; i++){
        if(phi[i]==i){ // i Ãľ primo
            for(int j =1; j*i <=n; j++){
                phi[j*i] -= phi[j*i]/i; //n - n/primo
            }
        }
    }
    return phi;
}


// fonte : https://cp-algorithms.com/algebra/phi-function.html
```

3.5 Gcdbasics

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

using namespace std;

// for c++17 and beyond, gcd and lcm are built-in functions, and
// they're very probably just better to use then this bellow, but still:

/* SIMPLE EUCLIDIAN GCD (MDC POR ALGORITMO DE EUCLIDES)

g = gcd(a,b) -> g|a ^ g|b
g|a ^ g|b -> g|(a-bx);  tal que  b > (a-bx) >= 0; (a-bx) == a%b;
g|b ^ g|(a%b) -> g|(b-(a%b))

gcd(x, 0) == x;
```

```cpp

*/
int euclidian_gcd (int a, int b){
    while(b > 0){
        a %=b;
        swap(a,b);
    }
    return a;
}

/* LCM (MMC)
gcd is double counted, so remove it;
*/

int lcm (int a, int b){
    return (a/ euclidian_gcd(a,b)) *b;
}
```

3.6 Gcdextended

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

using namespace std;

/* EXTENDED EUCLIDIAN ALGORITHM
returns gcd(a,b) as well as x and y such that a.x + b.y = gcd(a,b);
the iterative version is faster, but way harder to comprehend.
*/

// honestly, blackbox.
int gcd(int a, int b, int& x, int& y) {
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
    return a1;
}

/* RECURSIVE
when gcd(a,b) = g is found, there is:
1.g + 0.0 = g;

now, assume there is x1, y1 such b.x1 + (a%b).y1 = g;
and from there, lets try to "up a step", find x and y such a.x + b.y = g;
as stated in .\gcdBasics.cpp, a%b = a - (a/b)*b; ! disclaimer!: a/b is the floor of a/b, as default in cpp;
so, b.x1 + (a - (a/b)*b)*y1 = g;
b.x1 + a.y1 - (a/b)*b*y1 =g;
a.y1 + b(x1 - (a/b)*y1) =g;
therefore, as intended:
x = y1;
y = x1 - (a/b)*y1;

*/
int recursive_extended_gcd(int a, int b, int& x, int& y){
    if(b == 0){
        x = 1;
        y =0;
        return a; //gcd is found;
    }
    int x1, y1;
    int g = recursive_extended_gcd(a, a%b, x1, y1);

    int x = y1;
    int y = x1 - y1*(a/b);

    return g;
}
```

3.7 Modulo

```cpp
// AnotaÃğÃţes importantes sobre mÃşdulo e expressÃţes modulares;
// a = b (mod m); <-> a + (k*m) = b + (t*m), k e t sendo inteiros;
// -x (mod m) = - x + m (mod m);
// (a/b) (mod m) != (a (mod m) / b (mod m)) mod m
//      a/b (mod m) = (a (mod m)) * b_inverso (mod m)
//      gcd(m,b) = 1, b_inverso = b^(phi(m)-1)  [ver EulerTotient.cpp];
//      se m ÃĽ primo, phi(m) = m-1, b_inverso = b^(m-2);
```

3.8 Primefactorization

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

using namespace std;


vector<pair<int, int>> primeFactorization(int n) {
    vector<pair<int, int>> factors;

    // Divide out 2 first
    int count = 0;
    while (n % 2 == 0) {
        n /= 2;
        count++;
    }
    if (count > 0) factors.emplace_back(2, count);

    // Now check odd numbers
    for (int i = 3; i*i <= n; i += 2) {
        count = 0;
        while (n % i == 0) {
            n /= i;
            count++;
        }
        if (count > 0) factors.emplace_back(i, count);
    }

    // If n is still > 1, it's a prime factor
    if (n > 1) factors.emplace_back(n, 1);

    return factors;
}
```

4 Strings

4.1 Hash(tiagodfs)

```cpp
// String Hash template
// constructor(s) - O(|s|)
// query(l, r) - returns the hash of the range [l,r] from left to right - O(1)
// query_inv(l, r) from right to left - O(1)

#include <bits/stdc++.h>
#define ll long long

using namespace std;

#define MOD 1000000009


```

```cpp
14 
15 struct Hash {
16     const ll P = 31;
17     int n; string s;
18     vector<ll> h, hi, p;
19     Hash() {}
20     Hash(string s): s(s), n(s.size()), h(n), hi(n), p(n) {
21         for (int i=0;i<n;i++) p[i] = (i ? P*p[i-1]:1) % MOD;
22         for (int i=0;i<n;i++)
23             h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
24         for (int i=n-1;i>=0;i--)
25             hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P) % MOD;
26     }
27     int query(int l, int r) {
28         ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD : 0));
29         return hash < 0 ? hash + MOD : hash;
30     }
31     int query_inv(int l, int r) {
32         ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l+1] % MOD : 0));
33         return hash < 0 ? hash + MOD : hash;
34     }
35 };
36 
37 int main(){
38 
39     string s = "abcde";
40     string t = "edcba";
41 
42     Hash h1(s);
43     Hash h2(t);
44 
45     cout << h1.query(0,h1.n-1) << " " << h2.query_inv(0, h2.n-1) << endl;
46     cout << h1.query_inv(0, h1.n-1) << " "<< h2.query(0, h2.n-1) ;
47     return 0;
48 }
```

4.2 Liscomseg

```cpp
1 #include <bits/stdc++.h>
2 
3 using namespace std;
4 
5 #define ll long long
6 #define vi vector<int>
7 #define pii pair<int,int>
8 #define vll vector<long long>
9 
10 const int oo = 1e9+3;
11 
12 
13 /*LIS com seg:
14 em lis com dp, temos que dp[i] = max(dp[j], para j (i-1...0),v[i]>v[j] );
15 ou seja, dos elementos de valor menor do que o comparado, qual tem a maior lis (possivelmente 0);
16 isso pode ser feito usando uma seg de max, da seguinte forma:
17 manter uma seg de range dos valores* (0...maior a[i]), e percorrer o vetor analisado
18 da esquerda pra direita definindo o lis pra cada como na linha 14, e dando update na seg
19 
20 blog extramente Ãžtil:
21 https://codeforces.com/blog/entry/101210
22 
23 *OBS.: Quase sempre vale a pena normalizar/comprimir o vetor original
24 */
25 
26 
27 class SegTree { //construir a seg vazia primeiro;
28     vector<ll> st;
29     vector<ll> v;
30     int size;
31     int elem_neutro = -oo;// changes based on the segtree function
32 
33 public:
34 
35     SegTree(vector<int> arr, int size): st(4*size,0), v(size+1,0) {
36         this->size = size;
37 
38         for(int i = 1; i<size+1; i++){
39             v[i] = arr[i];
40         }
41     }
42 
43     ll f(ll a, ll b){ //type and return of function are variable, depending on the segtree
44         return max(a,b);
45     }
46 
47     void build(int l, int r, int nodo){
48         if(l ==r){
49             st[nodo] = v[l];
50             return;
51         }
52         int m = (l+r)/2;
53 
54 
55         build(l,m,nodo*2); //desceu p esquerda
56         build(m+1,r,nodo*2+1); //desceu p direita
57 
58 
59         st[nodo] = f(st[nodo*2], st[nodo*2+1]);
60     }
61 
62     void update_range(int i, int l, int r, ll x, int nodo) {
63         if(l == r){
64             st[nodo]  = x;
65             return;
66         }
67 
68         int m = (l+r)/2;
69 
70         if(i <=m){
71             update_range(i, l, m, x, nodo*2);
72         }
73         else{
74             update_range(i,m+1,r,x,nodo*2+1);
75         }
76 
77         st[nodo] = f(st[nodo*2],st[nodo*2+1]);
78     };
79 
80     void update(int i, ll x){
81         int l =1;
82         int r = size;
83         int nodo = 1;
84         return update_range(i,l,r,x,nodo);
85     }
86 
87     ll query_range(int ql, int qr, int l, int r, int nodo){
88         if(l >qr or r < ql){
89             return elem_neutro;
```

```
        }

        if( l >= ql and r <= qr){
            return st[nodo];
        }

        int m = (l+r)/2;
        ll suml = query_range(ql,qr,l,m,nodo*2);
        ll sumr = query_range(ql,qr,m+1,r,nodo*2+1);
        return f(suml,sumr);
    }

    ll query(int ql, int qr){
        int l = 1;
        int r = size;
        int nodo =1;
        return query_range(ql,qr,l,r,nodo);
    }
};

/* Errichto falou que Ãľ a versÃčo mais rÃąpida pra normalizar;
de fato, nÃčo usa set, nem map e nem binary, parece mais rÃąpido mesmo;
Obs.: pra refazer a normalizaÃğÃčo, basta manter um vector sorted e dai o
o vetor normalizado vira um vetor "ponteiro" pra esse sorted.
*/

void normalize( vi& v){
    int n = v.size();
    vector<pii> pairs(n);

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

    sort(pairs.begin(),pairs.end());
    int nxt = 0;
    for(int i =0; i<n; i++){
        if(i>0 && pairs[i-1].first != pairs[i].first) nxt++;
        v[pairs[i].second] = nxt;
    }
}

int find_lis(const vi& v){
    int n = v.size();
    vi empty(n,0);
    SegTree seg(empty,n);
    seg.build(1,n,1);

    int lis = 0;

    for(int i =0; i<n; i++){
        if(v[i]>0){
            lis = seg.query(1,v[i]-1);
        }else{
            lis =0;
        }
        seg.update(v[i],lis+1);
    }

    return seg.query(1,n);
}


int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);


    return 0;
}
```

4.3 Trie

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

using namespace std;

#define ll long long
#define QTD_CARACTERES 26 //alfabeto minusculo
#define LETRA_BASE 'a' //primeira letra pra fazer indexaÃğÃčo

struct Nodo
{
    Nodo* filhos[QTD_CARACTERES];
    bool fimDePalavra;
    ll ocorrencias =0;


    Nodo() {
        fimDePalavra = false;
        ocorrencias = 0;
        for(int i = 0; i<QTD_CARACTERES; i++){
            filhos[i] = NULL;
        }
    }
};

struct Trie
{
    Nodo* raiz;
    char letraBase = LETRA_BASE;


    public:

    Trie() {
        raiz = new Nodo();
    }

    void insert(string s){
        Nodo* cur = raiz;
        for(char c : s){
            if(cur->filhos[c- letraBase] == NULL){
                Nodo* novoNodo = new Nodo();

                cur->filhos[c-letraBase] = novoNodo;
            }
            cur->ocorrencias++;
            cur = cur->filhos[c-letraBase];
        }
        cur->ocorrencias++;
        cur->fimDePalavra = true;
    }

    bool searchWord(string s){
        Nodo* cur = raiz;

        for(char c : s){
            if(cur->filhos[c-letraBase]== NULL){
                return false;
            }

            cur = cur->filhos[c-letraBase];
        }

        return cur->fimDePalavra;
    }

    ll countPrefix(string s){
        Nodo* cur = raiz;
```

```cpp

        for(char c: s){
            if(cur->filhos[c-letraBase]==NULL){
                return false;
            }

            cur = cur->filhos[c-letraBase];
        }
        return cur->ocorrencias;
    }
};

int main() {

    string s = "doguinho";
    string p = "dogao";
    string q = "dogg";
    string m = "doguimio";

    Trie trie;
    trie.insert(s);
    cout << trie.countPrefix("dog") << endl;
    cout << trie.searchWord("doguinho") << endl;
    trie.insert(p);
    trie.insert(q);

    cout << trie.countPrefix("dog") << endl;
    cout << trie.searchWord("dog") << endl;
    cout << trie.countPrefix("doga") << endl;
    cout << trie.searchWord("dogao") << endl;
    cout << trie.countPrefix("oi") << endl;
    cout << trie.searchWord("sim");









}
```

4.4 Zfunction(tiagodfs)

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

using namespace std;


vector<int> Z(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        z[i] = max(0, min(z[i - l], r - i + 1));
        while (i + z[i] < n and s[z[i]] == s[i + z[i]]) {
            l = i; r = i + z[i]; z[i]++;
        }
    }
    return z;
}

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    string s;

    cin >> s;
    vector<int> v = Z(s);

    for(int i : v){
        cout << i << " ";
    }


    return 0;
}
```

5 Templates

5.1 Base

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

using namespace std;

#define vi vector<int>
#define ll long long
#define pii pair<int,int>
#define pll pair<long long, long long>
#define vll vector<long long>
#define endl "\n"


void solve() {



}


bool const testcases = true;
int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);


    int t =1; if(testcases){ cin >> t;}

    while(t--){
        solve();
    }



    return 0;
}
```