



Cervejeiros do Mendes

Caue Trindade, Guga Tonnera & Vini Araujo

Contents

1 Datastructures	2	3.7 Modulo	8
1.1 Dsueclass	2	3.8 Primefactorization	8
1.2 Segtree	2	4 Strings	8
1.3 Sparsetable	3	4.1 Hash(tiagodfs)	8
2 Graphs	3	4.2 Liscomseg	9
2.1 Djikstra	3	4.3 Trie	10
2.2 Binarylifting	3	4.4 Zfunction(tiagodfs)	11
2.3 Diameter	4	5 Templates	11
2.4 Eulertour	4	5.1 Base	11
2.5 Kosaraju	5		
2.6 Kruskal	5		
2.7 Lca	6		
3 Maths	6		
3.1 Catalan	6		
3.2 Chineseremainder	7		
3.3 Diofantineequation	7		
3.4 Eulertotient	7		
3.5 Gcdbasics	7		
3.6 Gcdextended	8		



1 Datastructures

1.1 Dsuclass

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 class DSU
6 {
7     vector<int> parent;
8     vector<int> card;
9
10 public:
11     DSU(int n): parent(n+1), card(n+1,1)
12     {
13         for(int i = 1; i <= n; i++)
14             parent[i] = i;
15     }
16
17     /* O(log n) */
18     int find_set(int x)
19     {
20         if(x == parent[x])
21             return x;
22
23         return parent[x] = find_set(parent[x]);
24     }
25
26     bool same_set(int a, int b)
27     {
28         return find_set(a) == find_set(b);
29     }
30
31     /* O(log n) */
32     void join_sets(int a, int b)
33     {
34         a = find_set(a);
35         b = find_set(b);
36
37         if(card[a] < card[b])
38             swap(a,b);
39
40         card[a] += card[b];
41         parent[b] = a;
42     }
43 };

```

1.2 Segtree

```

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
9
10 //passar vetor indexado em 1
11
12 class SegTree {
13     vector<ll> st;
14     vector<ll> v;
15     int size;
16     int elem_neutro = 0; // changes based on the
17                             segtree function
18
19 public:
20     SegTree(vector<ll> arr, int size): st(4*size,0),
21         v(size+1,0) {

```

```

21         this->size = size;
22
23         for(int i = 1; i<size+1; i++){
24             v[i] = arr[i];
25         }
26     }
27
28     ll f(ll a, ll b){ //type and return of function
29         are variable, depending on the segtree
30         return a+b;
31     }
32
33     void build(int l, int r, int nodo){
34         if(l ==r){
35             st[nodo] = v[l];
36             return;
37         }
38         int m = (l+r)/2;
39
40         build(l,m,nodo*2); //desceu p esquerda
41         build(m+1,r,nodo*2+1); //desceu p direita
42
43         st[nodo] = f(st[nodo*2], st[nodo*2+1]);
44     }
45
46     void update_range(int i, int l, int r, ll x, int
47         nodo) {
48         if(l == r){
49             st[nodo] = x;
50             return;
51         }
52
53         int m = (l+r)/2;
54
55         if(i <=m){
56             update_range(i, l, m, x, nodo*2);
57         }
58         else{
59             update_range(i,m+1,r,x,nodo*2+1);
60         }
61
62         st[nodo] = f(st[nodo*2],st[nodo*2+1]);
63     };
64
65     void update(int i, ll x){
66         int l =1;
67         int r = size;
68         int nodo = 1;
69         return update_range(i,l,r,x,nodo);
70     }
71
72     ll query_range(int ql, int qr, int l, int r, int
73         nodo){
74         if(l >qr or r < ql){
75             return elem_neutro;
76         }
77
78         if( l >= ql and r <= qr){
79             return st[nodo];
80         }
81
82         int m = (l+r)/2;
83         ll suml = query_range(ql,qr,l,m,nodo*2);
84         ll sumr = query_range(ql,qr,m+1,r,nodo*2+1);
85         return f(suml,sumr);
86     }
87
88     ll query(int ql, int qr){
89         int l = 1;
90         int r = size;
91         int nodo = 1;

```



```

91         return query_range(ql,qr,l,r,nodo);
92     }
93 };
94
95 int main() {
96     ios_base::sync_with_stdio(0);
97     cin.tie(0);
98
99     return 0;
100 }

```

1.3 Sparsetable

```

1  #include <bits/stdc++.h>
2  #define ll long long
3  #define vi vector<int>
4  #define vll vector<long long>
5  using namespace std;
6
7
8  // Complexity: The pre processing is O(nlog(n)).
9  // Most queries such as sum are O(log(n));
10 // So in that case, it is pretty much always better
11 // to use a segTree.
12 // However, when the queries are min or max, the
13 // complexity of each query becomes O(1);
14
15 class SparseTable
16 {
17     int logn;
18     vector<vll> tb;
19     vi logs;
20
21 public:
22     // Constructor takes only 'n' and 'arr' as
23     // arguments
24     SparseTable(int n, vll arr) : logs(n + 1), tb(
25         log2(n) + 1, vll(n)) {
26         // Build the 'logs' array
27         logs[1] = 0;
28         for (int i = 2; i <= n; i++) {
29             logs[i] = logs[i / 2] + 1;
30         }
31
32         // Set logn to the maximum power needed for '
33         // n'
34         this->logn = logs[n];
35
36         // Initialize the first row of 'tb' with 'arr
37         // values
38         for (int i = 0; i < n; i++) {
39             tb[0][i] = arr[i];
40         }
41
42         // Build the sparse table
43         for (int k = 1; k <= logn; k++) {
44             for (int i = 0; i + (1 << k) <= n; i++) {
45                 tb[k][i] = min(tb[k - 1][i], tb[k -
46                     1][i + (1 << (k - 1))]);
47             }
48         }
49
50         // Range minimum query on the range [l, r]
51         ll query(int l, int r) {
52             //if not indexed in 1 remove this.
53             l--;
54             r--;
55             int len = r - l + 1;
56             int k = logs[len];
57             return min(tb[k][l], tb[k][r - (1 << k) + 1])
58         }
59     };
60 }

```

```

52 };
53
54 int main() {
55     ios_base::sync_with_stdio(0);
56     cin.tie(0);
57
58     int n, q;
59     cin >> n >> q;
60     vll arr(n);
61
62     // Initializing array;
63     for (int i = 0; i < n; i++) {
64         cin >> arr[i];
65     }
66
67     // Initialize the Sparse Table with the array
68     SparseTable sp(n, arr);
69
70     // Perform range minimum queries
71     for (int i = 0; i < q; i++) {
72         int l, r;
73         cin >> l >> r;
74         cout << sp.query(l, r) << endl;
75     }
76
77     return 0;
78 }

```

2 Graphs

2.1 Dijkstra

```

1  #include<bits/stdc++.h>
2
3  using namespace std;
4
5
6  #define pii pair<int, int>
7  #define ll long long
8
9
10 int N;
11 const int oo = 1e6+7; // depende
12
13 vector<vector<pii>> g(N);
14 vector<bool> used(N);
15 vector<ll> d(N, oo);
16 priority_queue< pii, vector<pii>, greater<pii> > q;
17
18 void dijkstra(int k) {
19     d[k] = 0;
20     q.push({0, k});
21
22     while (!q.empty()) {
23         auto [w, u] = q.top();
24         q.pop();
25         if (used[u]) continue;
26         used[u] = true;
27
28         for (auto [v, w]: g[u]) {
29             if (d[v] > d[u] + w) {
30                 d[v] = d[u] + w;
31                 q.push({d[v], v});
32             }
33         }
34     }
35 }

```

2.2 Binarylifting

```

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

```



```

2
3 using namespace std;
4
5 #define vi vector<int>
6
7
8
9 class TreeAncestor {
10     int LOG;
11     vector<vi> up; // [n][log] -> o antecessor de n
12     // em 2^log
13     vi depth;
14 public:
15     TreeAncestor(int n, vi& pai){
16         LOG = 0;
17         while((1 << LOG) <= n){
18             LOG++;
19             // formula de calcular o log
20         }
21         up = vector<vi>(n, vi(LOG));
22         depth = vi(n);
23
24         // pai[i]<i:
25         pai[0] = 0;
26         for(int v = 0; v < n; v++){
27             up[v][0] = pai[v];
28             if(v != 0) depth[v] = depth[pai[v]] + 1;
29             for(int j = 1; j < LOG; j++){
30                 up[v][j] = up[up[v][j-1]][j-1];
31             }
32         }
33     }
34
35     int getAncestralK (int nodo, int k){
36
37         if(depth[nodo] < k){
38             return -1; // impossivel
39         }
40
41         // 1 << j = 2^j
42
43         for(int j = LOG-1; j >= 0; j--){
44             if(k >= (1 << j)){
45                 nodo = up[nodo][j];
46                 k -= (1 << j);
47             }
48         }
49
50         return nodo;
51     }
52 };
53
54

```

2.3 Diameter

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int N = 1e4; // tamanho da Árvore
6
7 vector<vector<int>> g(N); // Árvore por lista de
8 // adjacências;
9
10 void dfs(int x, int p, int l, int& longestPath, int&
11 // nodo){
12     for(auto v : g[x]){
13         if(v == p) continue;
14         dfs(v, x, l+1, longestPath, nodo);
15     }
16     if(l > longestPath){
17

```

```

15         longestPath = l;
16         nodo = x;
17     }
18 }
19
20 // x = primeiro nó da Árvore
21 pair<int, pair<int, int>> findDiameter(int x){
22     // g = tree por lista de adjacência
23     int nodo = x;
24     int longestPath = 0;
25
26     dfs(x, -1, 1, longestPath, nodo);
27
28     longestPath = 0;
29     x = nodo;
30
31     dfs(x, -1, 1, longestPath, nodo);
32
33     return make_pair(longestPath, make_pair(x, nodo));
34 }

```

2.4 Eulertour

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 /*
6     Utilidades para o Euler Tour:
7     o menor subarray entre dois elementos da array de
8     EulerTour é o caminho entre esses dois
9     vértices
10     com isso, é possível descobrir várias coisas,
11     como:
12     qual o lca range (o depth mínimo);
13     qual a soma de distâncias do caminho entre
14     dois vértices
15     qual a maior distância do caminho entre dois
16     vértices
17     dentre outros
18 */
19
20 const int N = 1e5;
21
22 vector<vector<int>> g(N+1);
23 vector<int> depth(N+1); // definir depth[raiz] = 1;
24 vector<int> eulerTour; // vai ter 2*N-1 elementos;
25 vector<int> first(N+1, -1); // primeira vez que um nó
26 // aparece no euler tour
27 vector<vector<int>> appears(N+1); // posições em que
28 // um nó aparece no eulertour
29
30 void dfs(int u){
31
32     if(first[u] == -1){
33         first[u] = eulerTour.size();
34     }
35
36     appears[u].push_back(eulerTour.size());
37     eulerTour.push_back(u);
38
39     for(auto v : g[u]){
40         depth[v] = depth[u] + 1;
41         dfs(v);
42         appears[u].push_back(eulerTour.size());
43         eulerTour.push_back(u);
44     }
45 }
46
47 void print(){
48     cout << eulerTour.size() << endl;
49     for(auto x : eulerTour) cout << x << " ";
50     cout << endl;

```



```

44     for(auto x : eulerTour) cout << depth[x] << " ";
45     cout << endl;
46     for(int x = 1; x < 9; x++) cout << first[x] << " ";
47     ;
48 }
49
50 int main() {
51     g[1].push_back(2);
52     g[2].push_back(3);
53     g[2].push_back(4);
54     g[4].push_back(5);
55     g[1].push_back(6);
56     g[6].push_back(7);
57     g[6].push_back(8);
58
59     depth[1] = 1;
60     dfs(1);
61
62     print();
63
64     return 0;
65 }

```

2.5 Kosaraju

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  /*
6   Algoritmo de Kosaraju:
7   contexto : grafo direcionado;
8   objetivo: encontrar componentes fortemente
9   conectados* no grafo
10
11   *fortemente conectado: Ãr possÃvel chegar em
12   todos os nÃs saindo de qualquer nÃ;
13   *componente fortemente conectado (SCC): maior
14   quantidade possÃvel de vÃrtices fortemente
15   conectados no grafo
16
17   Um grafo G direcionado pode ser representado por
18   um grafo acÃclico direcionado S onde cada nÃ de
19   S Ã um SCC de G;
20   *S == G, se G Ã acÃclico
21
22   ExecuÃÃo:
23   encontra a pÃs-ordem por DFS;
24   cria um grafo Ginv igual a G mas com a direÃÃo
25   das arestas trocada.
26   Obs, um componente fortemente conectado em G
27   tambÃm Ã em Ginv;
28   Performa DFS em Ginv para encontrar os SCC
29   diferentes.
30 */
31
32 const int N = 1e5+1;
33 vector<vector<int>> g(N), ginv(N); //grafo original,
34     inverso e resultado do kosaraju;
35 vector<int> vis(N,0);
36 vector<int> pai(N); //nÃs que vai representar o
37     componente conexo (noÃÃo de dsu);
38 vector<int> scc; //lista de representantes de
39     componentes scc;
40 stack<int> posord;
41
42 void dfs(int u){
43     vis[u]=1;
44     for(auto v: g[u]){
45         if(!vis[v]) dfs(v);
46     }
47     posord.push(u);
48 }
49
50 }

```

```

36 void invdfs(int u){
37     vis[u] = 1;
38     for(auto v: ginv[u]){
39
40         if(!vis[v]){
41             pai[v] = pai[u];
42             invdfs(v);
43         }
44     }
45 }
46
47 void kosaraju(int n){
48     for(int i = 1; i <= n; i++) vis[i] = 0;
49
50     for(int i = 1; i <= n; i++) if(!vis[i]) dfs(i);
51
52     for(int i = 1; i <= n; i++) vis[i] = 0;
53
54     while(!posord.empty()){
55         int u = posord.top(); posord.pop();
56         if(vis[u]) continue;
57
58         pai[u] = u;
59         scc.push_back(u);
60         invdfs(u);
61     }
62 }
63 }

```

2.6 Kruskal

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  /*
6   Kruskal:
7   percorre todas as arestas em ordem,
8   se a ordem for crescente retorna a minimum spanning
9   tree;
10  se a ordem for decrescente retorna a maximum spanning
11  tree;
12  se os dois vÃrtices que cada aresta liga jÃ estÃo
13  conectados, pula essa aresta;
14  se nÃo, conecta os dois componentes e adiciona essa
15  aresta;
16
17  a prova de que funciona Ã meio trivial, e a
18  implementaÃÃo Ã bem tranquila com dsu,
19  sÃ to adicionando ai pra jÃ ter uma que retorna
20  mastigadinho com a Ãrvore montada por adj;
21 */
22
23 class DSU
24 {
25     vector<int> parent;
26     vector<int> card;
27
28 public:
29     DSU(int n): parent(n+1), card(n+1,1)
30     {
31         for(int i = 1; i <= n; i++)
32             parent[i] = i;
33     }
34
35     /* O(log n) */
36     int find_set(int x)
37     {
38         if(x == parent[x])
39             return x;
40
41         return parent[x] = find_set(parent[x]);
42     }
43 }

```



```

37
38 bool same_set(int a, int b)
39 {
40     return find_set(a) == find_set(b);
41 }
42
43 /* O(log n) */
44 void join_sets(int a, int b)
45 {
46     a = find_set(a);
47     b = find_set(b);
48
49     if(card[a] < card[b])
50         swap(a,b);
51
52     card[a] += card[b];
53     parent[b] = a;
54 }
55 };
56
57 // n = quantidade de vértices;
58 // retorna tree com tree[x] = {weight, vertice};
59
60 vector<vector<pair<int,int>>> Kruskal(vector<pair<int,int>
    ,pair<int,int>>>& arestas, int n){
61     DSU d(n);
62     vector<vector<pair<int,int>>> tree(n+1);
63
64     for(auto [w,p] : arestas){
65         if(d.same_set(p.first, p.second)) continue;
66         d.join_sets(p.first, p.second);
67         tree[p.first].push_back({w,p.second});
68         tree[p.second].push_back({w,p.first});
69     }
70
71     return tree;
72 }

```

2.7 Lca

```

1 //LCA com binary lifting;
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 #define vi vector<int>
7
8 int MAX_N = 1e6;
9 int LOG = 20;
10
11 vector<vi> children(MAX_N,vi(MAX_N));
12 vector<vi> up(MAX_N, vi(LOG));
13 vi depth(MAX_N);
14
15 //definir up[a]=a antes;
16 //construtor se passarmos o a como raiz;
17 void dfs(int a){
18     for(int b: children[a]){
19         up[b][0] = a;
20         depth[b] = depth[a] + 1;
21         for(int j = 1; j < LOG; j++){
22             up[b][j] = up[ up[b][j-1] ][j-1];
23         }
24         dfs(b);
25     }
26 }
27
28 int jump(int a, int k){
29     for(int j = LOG-1; j >= 0; j--){
30         if(k >= (1<<j)){
31             a = up[a][j];
32             k -= (1<<j);
33         }
34     }
35     return a;
36 }

```

```

34     }
35     return a;
36 }
37
38 int getAncestralK (int nodo, int k){
39
40     if(depth[nodo]<k){
41         return -1; //impossível
42     }
43
44     // 1 << j = 2^j
45
46     for(int j = LOG-1; j>=0; j--){
47         if(k >= (1<<j)){
48             nodo = up[nodo][j];
49             k -= (1<<j);
50         }
51     }
52
53     return nodo;
54 }
55
56 int find_lca(int a, int b){
57
58     if(depth[a]< depth[b]){
59         swap(a,b); // a sempre o mais profundo;
60     }
61     int k = depth[a] - depth[b];
62     a = jump(a,k);
63
64     if(a == b) return a;
65
66     for(int j = LOG-1; j>=0; j--){
67         if(up[a][j]!= up[b][j]){
68             a = up[a][j];
69             b = up[b][j];
70         }
71     }
72     return up[a][0];
73 }
74
75 int dist(int a, int b){
76     return depth[a] + depth[b] - 2*depth[find_lca(a,b)];
77 }

```

3 Maths

3.1 Catalan

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5
6 #define ll long long
7 #define vll vector<long long>
8
9
10 vll fac;
11
12 /* C(n) = Combinacao(2n) * 1/(n+1)
13         ( n)
14
15 C(n) = n! de sequencias a com n elementos +1 e n
16         elementos -1,
17 tal que p todo k, 1<=k <=2n; a1 + a2 +...+ak >=0;
18 Aplicações:
19 sequencias de parenteses
20 qtd de arvores binarias
21 etc depois escrevo o resto
22 */

```



```

22
23 ll Catalan(ll n){
24     return Comb(2*n,n,mod)*expMod(n+1,mod-2,mod);
25 }

```

3.2 Chineseremainder

```

1  /*
2  com todos m primos entre si, a equação :
3
4  x= a1 mod m1
5  x = a2 mod m2
6  x = a3 mod m3
7  x = an mod mn
8
9  Já resolvida por x0 = a1*X1*inv_mod_m1(X1) + ... + an
    *Xn*inv_mod_mn(Xn);
10 sendo Xk = (m1*m2*m3*...*mk*...*mn)/mk
11 ou seja, Xk já um número divisível por todos m
    menos por mk, e por isso
12 a expressão resolve. Mais mastigado que isso já com
    o leitor kkkkkkkk
13
14 x_geral = x0 + k(m1*m2*m3*...*mn),    k inteiro
15
16 */

```

3.3 Diofantineequation

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  /* LINEAR DIOFANTINE EQUATIONS
6  ax + by + c with a,b and c given integers solving for
    x and y integers.
7
8  by .\gcdExtended.cpp, we can find ax' + by' = g with
    g = gcd(a,b);
9  therefore, if c is divisible by g, there are
    solutions, else there are none.
10 ( a = t.g , b = r.g, c = t.g + r.g = (t+r).g)
11 from that, follows:
12 ax'.(c/g) + by'.(c/g) = c
13 a solution then is:
14 ax0 + by0 = c;
15 x0 = x'.(c/g)
16 y0 = y'.(c/g)
17
18 to generalize it, take the equation:
19 a(x0 + b/g) + b(y0 - a/g) = c;
20 ax0 + by0 + ab/g - ab/g = c; it's still true since g
    = gcd(a,b) then:
21 ax+ by = c;
22 x = x0 + k.(b/g);
23 y = y0 + k.(y/g);
24
25 */

```

3.4 Eulertotient

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  /*Função totiente de Euler phi(n);
5  retorna a quantidade de números coprimos no
    intervalo fechado [1,n];
6
7  propriedades:
8  1-> para p primo: phi(p) = p-1;
9  2-> para p primo e k >=1: existem (p^k) * 1/p;
    números em [1,p^k] divisíveis por p; -> phi(p^k)
    =p^k - p^(k-1);

```

```

10  3-> para a e b coprimos, phi(a*b) = phi(a) * phi(
    b); (discorre do teorema do resto chinês);
11
12  Portanto, sendo p_1^k_1 * p_2^k_2*...*p_i^k_i = n
    a fatoração em primos de um número n,
13  phi(n) = phi(p_1^k_1)*...*phi(p_i^k_i) [Prop 3]
14  phi(n) = (p_1^k_1 - p_1^(k_1-1))* ... * (p_i^k_i
    - p_i^(k_i-1)) [Prop 2]
15  phi(n) = p_1^k_1(1 - 1/p_1)*...*p_i^k_i(1 - 1/p_i
    )
16  phi(n) = p_1^k_1*p_2^k_2*...*p_i^k_i*(1 - 1/p_1)*(1 -
    1/p_2)*(1 - 1/p_i);
17
18  */
19
20 //0(sqrt(n))
21 int phi(int n) {
22     int result = n;
23     for (int i = 2; i * i <= n; i++) {
24         if (n % i == 0) {
25             while (n % i == 0)
26                 n /= i;
27             result -= result / i;
28         }
29     }
30
31     if (n > 1) // n é primo;
32         result -= result / n;
33     return result;
34 }
35
36 // Faz o que a função phi faz (n*(1 - 1/p_i) = n - n
    /p_i) para cada fator primo p_i de n;
37 //O(nlog log n)
38 vector<int> phi_sieve(int n){
39     vector<int> phi(n+1);
40     for(int i = 0; i<=n; i++){
41         phi[i]=i;
42     }
43     for(int i =2; i<=n; i++){
44         if(phi[i]==i){ // i é primo
45             for(int j =1; j*i <=n; j++){
46                 phi[j*i] -= phi[j*i]/i; //n - n/primo
47             }
48         }
49     }
50     return phi;
51 }
52
53
54 // fonte : https://cp-algorithms.com/algebra/phi-
    function.html

```

3.5 Gcdbasics

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  // for c++17 and beyond, gcd and lcm are built-in
    functions, and
6  // they're very probably just better to use than this
    bellow, but still:
7
8  /* SIMPLE EUCLIDIAN GCD (MDC POR ALGORITMO DE
    EUCLIDES)
9
10 g = gcd(a,b) -> g|a ^ g|b
11 g|a ^ g|b -> g|(a-bx); tal que b > (a-bx) >= 0;
    (a-bx) == a%b;
12 g|b ^ g|(a%b) -> g|(b-(a%b))
13
14 gcd(x, 0) == x;

```



```

15
16 */
17 int euclidian_gcd (int a, int b){
18     while(b > 0){
19         a %=b;
20         swap(a,b);
21     }
22     return a;
23 }
24
25 /* LCM (MMC)
26 gcd is double counted, so remove it;
27 */
28
29 int lcm (int a, int b){
30     return (a/ euclidian_gcd(a,b)) *b;
31 }

```

3.6 Gcdextended

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 /* EXTENDED EUCLIDIAN ALGORITHM
6 returns gcd(a,b) as well as x and y such that a.x + b
   .y = gcd(a,b);
7 the iterative version is faster, but way harder to
   comprehend.
8 */
9
10 // honestly, blackbox.
11 int gcd(int a, int b, int& x, int& y) {
12     x = 1, y = 0;
13     int x1 = 0, y1 = 1, a1 = a, b1 = b;
14     while (b1) {
15         int q = a1 / b1;
16         tie(x, x1) = make_tuple(x1, x - q * x1);
17         tie(y, y1) = make_tuple(y1, y - q * y1);
18         tie(a1, b1) = make_tuple(b1, a1 - q * b1);
19     }
20     return a1;
21 }
22
23 /* RECURSIVE
24 when gcd(a,b) = g is found, there is:
25 1.g + 0.0 = g;
26
27 now, assume there is x1, y1 such b.x1 + (a/b).y1 = g;
28 and from there, lets try to "up a step", find x and y
   such a.x + b.y = g;
29 as stated in .\gcdBasics.cpp, a/b = a - (a/b)*b; !
   disclaimer!: a/b is the floor of a/b, as default
   in cpp;
30 so, b.x1 + (a - (a/b)*b)*y1 = g;
31 b.x1 + a.y1 - (a/b)*b*y1 =g;
32 a.y1 + b(x1 - (a/b)*y1) =g;
33 therefore, as intended:
34 x = y1;
35 y = x1 - (a/b)*y1;
36
37 */
38 int recursive_extended_gcd(int a, int b, int& x, int&
   y){
39     if(b == 0){
40         x = 1;
41         y =0;
42         return a; //gcd is found;
43     }
44     int x1, y1;
45     int g = recursive_extended_gcd(a, a/b, x1, y1);
46
47     int x = y1;

```

```

48     int y = x1 - y1*(a/b);
49
50     return g;
51 }

```

3.7 Modulo

```

1 // Anotações importantes sobre módulo e
   expressões modulares;
2 // a = b (mod m); <-> a + (k*m) = b + (t*m), k e t
   sendo inteiros;
3 // -x (mod m) = - x + m (mod m);
4 // (a/b) (mod m) != (a (mod m) / b (mod m)) mod m
5 // a/b (mod m) = (a (mod m)) * b_inverso (mod m)
6 // gcd(m,b) = 1, b_inverso = b^(phi(m)-1) [ver
   EulerTotient.cpp];
7 // se m Ã primo, phi(m) = m-1, b_inverso = b^(m
   -2);

```

3.8 Primefactorization

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 vector<pair<int, int>> primeFactorization(int n) {
6     vector<pair<int, int>> factors;
7
8
9     // Divide out 2 first
10    int count = 0;
11    while (n % 2 == 0) {
12        n /= 2;
13        count++;
14    }
15    if (count > 0) factors.emplace_back(2, count);
16
17    // Now check odd numbers
18    for (int i = 3; i*i <= n; i += 2) {
19        count = 0;
20        while (n % i == 0) {
21            n /= i;
22            count++;
23        }
24        if (count > 0) factors.emplace_back(i, count)
25    }
26
27    // If n is still > 1, it's a prime factor
28    if (n > 1) factors.emplace_back(n, 1);
29
30    return factors;
31 }

```

4 Strings

4.1 Hash(tiagodfs)

```

1 // String Hash template
2 // constructor(s) - O(|s|)
3 // query(l, r) - returns the hash of the range [l,r]
   from left to right - O(1)
4 // query_inv(l, r) from right to left - O(1)
5
6 #include <bits/stdc++.h>
7 #define ll long long
8
9 using namespace std;
10
11 #define MOD 1000000009
12
13

```




```

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
21     (n) {
22         for (int i=0;i<n;i++) p[i] = (i ? P*p[i-1]:1)
23         % MOD;
24         for (int i=0;i<n;i++)
25             h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
26         for (int i=n-1;i>=0;i--)
27             hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P)
28             % MOD;
29     }
30     int query(int l, int r) {
31         ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD :
32         0));
33         return hash < 0 ? hash + MOD : hash;
34     }
35     int query_inv(int l, int r) {
36         ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
37         +1] % MOD : 0));
38         return hash < 0 ? hash + MOD : hash;
39     }
40 };
41
42 int main(){
43     string s = "abcde";
44     string t = "edcba";
45
46     Hash h1(s);
47     Hash h2(t);
48
49     cout << h1.query(0,h1.n-1) << " " << h2.query_inv
50     (0, h2.n-1) << endl;
51     cout << h1.query_inv(0, h1.n-1) << " " << h2.query
52     (0, h2.n-1) ;
53     return 0;
54 }

```

4.2 Liscomseg

```

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
15 -1...0),v[i]>v[j] );
16 ou seja, dos elementos de valor menor do que o
17 comparado, qual tem a maior lis (possivelmente 0)
18 ;
19 isso pode ser feito usando uma seg de max, da
20 seguinte forma:
21 manter uma seg de range dos valores* (0...maior a[i])
22 , e percorrer o vetor analisado
23 da esquerda pra direita definindo o lis pra cada como
24 na linha 14, e dando update na seg
25
26 blog extramente Ãžtil:
27 https://codeforces.com/blog/entry/101210
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89

```



```

90     }
91
92     if( l >= ql and r <= qr){
93         return st[nodo];
94     }
95
96     int m = (l+r)/2;
97     ll suml = query_range(ql,qr,l,m,nodo*2);
98     ll sumr = query_range(ql,qr,m+1,r,nodo*2+1);
99     return f(suml,sumr);
100 }
101
102 ll query(int ql, int qr){
103     int l = 1;
104     int r = size;
105     int nodo = 1;
106     return query_range(ql,qr,l,r,nodo);
107 }
108 };
109
110 /* Errichto falou que ãl a versãõ mais rápida pra
111 normalizar;
112 de fato, não usa set, nem map e nem binary, parece
113 mais rápido mesmo;
114 Obs.: pra refazer a normalizaãõ, basta manter um
115 vector sorted e daí o
116 o vetor normalizado vira um vetor "ponteiro" pra
117 esse sorted.
118 */
119
120 void normalize( vi& v){
121     int n = v.size();
122     vector<pii> pairs(n);
123
124     for(int i=0; i<n; i++){
125         pairs[i] = {v[i],i };
126     }
127
128     sort(pairs.begin(),pairs.end());
129     int nxt = 0;
130     for(int i =0; i<n; i++){
131         if(i>0 && pairs[i-1].first != pairs[i].first)
132             nxt++;
133         v[pairs[i].second] = nxt;
134     }
135 }
136
137 int find_lis(const vi& v){
138     int n = v.size();
139     vi empty(n,0);
140     SegTree seg(empty,n);
141     seg.build(1,n,1);
142
143     int lis = 0;
144     for(int i =0; i<n; i++){
145         if(v[i]>0){
146             lis = seg.query(1,v[i]-1);
147         }else{
148             lis = 0;
149         }
150         seg.update(v[i],lis+1);
151     }
152     return seg.query(1,n);
153 }
154
155 int main() {
156     ios_base::sync_with_stdio(0);
157     cin.tie(0);

```

```

158     return 0;
159 }

```

4.3 Trie

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define ll long long
6 #define QTD_CARACTERES 26 //alfabeto minuscúlo
7 #define LETRA_BASE 'a' //primeira letra pra fazer
8 indexaããõ
9
10 struct Nodo
11 {
12     Nodo* filhos[QTD_CARACTERES];
13     bool fimDePalavra;
14     ll ocorrencias = 0;
15
16     Nodo() {
17         fimDePalavra = false;
18         ocorrencias = 0;
19         for(int i = 0; i<QTD_CARACTERES; i++){
20             filhos[i] = NULL;
21         }
22     }
23 };
24
25 struct Trie
26 {
27     Nodo* raiz;
28     char letraBase = LETRA_BASE;
29
30     public:
31
32     Trie() {
33         raiz = new Nodo();
34     }
35
36     void insert(string s){
37         Nodo* cur = raiz;
38         for(char c : s){
39             if(cur->filhos[c- letraBase] == NULL){
40                 Nodo* novoNodo = new Nodo();
41                 cur->filhos[c-letraBase] = novoNodo;
42             }
43             cur->ocorrencias++;
44             cur = cur->filhos[c-letraBase];
45         }
46         cur->ocorrencias++;
47         cur->fimDePalavra = true;
48     }
49
50     bool searchWord(string s){
51         Nodo* cur = raiz;
52
53         for(char c : s){
54             if(cur->filhos[c-letraBase]== NULL){
55                 return false;
56             }
57             cur = cur->filhos[c-letraBase];
58         }
59         return cur->fimDePalavra;
60     }
61
62     ll countPrefix(string s){
63         Nodo* cur = raiz;

```



```

68         for(char c: s){
69             if(cur->filhos[c-lettraBase]==NULL){
70                 return false;
71             }
72         }
73         cur = cur->filhos[c-lettraBase];
74     }
75     return cur->ocorrencias;
76 }
77 };
78 };
79
80 int main() {
81
82     string s = "doguinho";
83     string p = "dogao";
84     string q = "dogg";
85     string m = "doguimio";
86
87     Trie trie;
88     trie.insert(s);
89     cout << trie.countPrefix("dog") << endl;
90     cout << trie.searchWord("doguinho") << endl;
91     trie.insert(p);
92     trie.insert(q);
93
94     cout << trie.countPrefix("dog") << endl;
95     cout << trie.searchWord("dog") << endl;
96     cout << trie.countPrefix("doga") << endl;
97     cout << trie.searchWord("dogao") << endl;
98     cout << trie.countPrefix("oi") << endl;
99     cout << trie.searchWord("sim");
100
101
102
103
104
105
106
107 }
108 }

```

4.4 Zfunction(tiagodfs)

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5
6  vector<int> Z(string s) {
7      int n = s.size();
8      vector<int> z(n);
9      int l = 0, r = 0;
10     for (int i = 1; i < n; i++) {
11         z[i] = max(0, min(z[i - 1], r - i + 1));
12         while (i + z[i] < n and s[z[i]] == s[i + z[i]
13     ]]) {
14             l = i; r = i + z[i]; z[i]++;
15         }
16     }
17     return z;
18 }

```

```

17 }
18
19 int main(){
20     ios_base::sync_with_stdio(0);
21     cin.tie(0);
22
23     string s;
24
25     cin >> s;
26     vector<int> v = Z(s);
27
28     for(int i : v){
29         cout << i << " ";
30     }
31
32
33     return 0;
34 }

```

5 Templates

5.1 Base

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  #define vi vector<int>
6  #define ll long long
7  #define pii pair<int,int>
8  #define pll pair<long long, long long>
9  #define vll vector<long long>
10 #define endl "\n"
11
12
13 void solve() {
14
15
16
17 }
18
19
20 bool const testcases = true;
21 int main() {
22     ios::sync_with_stdio(0);
23     cin.tie(0);
24
25
26     int t = 1; if(testcases){ cin >> t;}
27
28     while(t--){
29         solve();
30     }
31
32
33     return 0;
34 }
35 }

```