



Pedro Augusto Ulisses Andrade Lucas Andrade

Katia Volte Para Mim! Cantarei Boate Azul Para Você (>o<)

Contents

1	Utils	2			
1.1	Makefile	2	5.2	Factorial	7
1.2	Limites	2	5.3	Mmc Multiplo	7
1.3	Mini Template Cpp	2	5.4	Fast Exponentiation	7
1.4	Template Cpp	3	5.5	Sieve	7
1.5	Files	3	5.6	Fact Grande	8
1.6	Template Python	3	5.7	Mdc	8
2	Informações	4	5.8	Primo	8
2.1	Vector	4	5.9	Miller Rabin	8
2.2	Sort	4	5.10	Fatorial Grande	9
2.3	Priority Queue	4	5.11	Sieve Linear	9
3	Estruturas	5	5.12	Numeros Grandes	9
3.1	Sparse Table Disjunta	5	5.13	Mmc	10
3.2	Seg Tree	5	6	Strings	10
4	Grafos	6	6.1	Ocorrencias	10
4.1	Bfs	6	6.2	Upper Case	10
4.2	Dijkstra	6	6.3	Palindromo	10
4.3	Kruskal	6	6.4	Split Cria	10
4.4	Dfs	7	6.5	Remove Acento	10
5	Matematica	7	6.6	Chaves Colchetes Parenteses	11
5.1	Mdc Multiplo	7	6.7	Lower Case	11
			6.8	Lexicograficamente Minima	11
			6.9	Split	11
			7	Vector	11
			7.1	Teste	11

7.2	Remove Repetitive	11
7.3	Elemento Mais Frequente	12
8	Outros	12
8.1	Binario	12
8.2	Horario	12
8.3	Max Subarray Sum	12
8.4	Binary Search	13
8.5	Fibonacci	13

1 Utils

1.1 Makefile

```
1 CXX = g++
2 CXXFLAGS = -fsanitize=address,undefined -fno-omit-frame-pointer -g -Wall -Wshadow -std=c++17 -Wno-unused-result -Wno-sign-compare -Wno-char-subscripts #-fuse-ld=gold
3
4 clear:
5     find . -maxdepth 1 -type f -executable -exec rm {} +
6
7 runc:
8     g++ -g $(f).cpp $(CPPFLAGS) -o $(f)
9     ./$(f)
10
11 runci:
12     g++ -g $(f).cpp $(CPPFLAGS) -o $(f)
13     ./$(f) < $(f).txt
14
15 runp:
16     python3 $(f).py
17
18 runpt:
19     python3 $(f).py < $(f).txt
```

1.2 Limites

```
1 // LIMITES DE REPRESENTAÇÃO DE DADOS
2
3      tipo      | bits |      mínimo .. máximo      |  preciso decimal
4 -----+-----+-----+-----+-----
5 char          |  8   |      0 .. 127              |      2
6 signed char    |  8   |     -128 .. 127           |      2
7 unsigned char  |  8   |      0 .. 255             |      2
8 short         | 16   |    -32.768 .. 32.767      |      4
9 unsigned short | 16   |      0 .. 65.535          |      4
10 int           | 32   | -2 x 10**9 .. 2 x 10**9    |      9
11 unsigned int   | 32   |      0 .. 4 x 10**9        |      9
12 int64_t        | 64   | -9 x 10**18 .. 9 x 10**18 |     18
13 uint64_t       | 64   |      0 .. 18 x 10**18     |     19
14
15 // LIMITES DE MEMÓRIA
16
17 1MB = 1,048,576 bool
18 1MB = 524,288 char
19 1MB = 262,144 int32_t
20 1MB = 131,072 int64_t
21 1MB = 65,536 float
22 1MB = 32,768 double
23 1MB = 16,384 long double
24
25 // ESTIMAR TEMPO
26
27 1s = 10^8 operações
```

```
29 // FATORIAL
30
31 0! = 1
32 1! = 1
33 2! = 2
34 3! = 6
35 4! = 24
36 5! = 120
37 6! = 720
38 7! = 5.040
39 8! = 40.320
40 9! = 362.880
41 10! = 3.628.800
42 11! = 39.916.800
43 12! = 479.001.600 [limite do (unsigned) int]
44 13! = 6.227.020.800
45 14! = 87.178.291.200
46 15! = 1.307.674.368.000
47 16! = 20.922.789.888.000
48 17! = 355.687.428.096.000
49 18! = 6.402.373.705.728.000
50 19! = 121.645.100.408.832.000
51 20! = 2.432.902.008.176.640.000 [limite do (u)int64_t]
```

1.3 Mini Template Cpp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define _ ios_base::sync_with_stdio(0); cin.tie(0);
5
6 #define all(a) a.begin(), a.end()
7 #define int long long int
8 #define double long double
9 #define vi vector<int>
10 #define endl "\n"
11 #define print_v(a) for(auto x : a) cout << x << " "; cout << endl
12 #define f(i,s,e) for(int i=s;i<e;i++)
13 #define rf(i,e,s) for(int i=e-1;i>=s;i--)
14
15 #define dbg(x) cout << #x << " = " << x << endl;
16
17 void solve() {
18
19 }
20
21 int32_t main() { _
22
23     int t = 1; // cin >> t;
24     while (t--)
25         //while(cin >> a >> b)
26             solve();
27
28     return 0;
29 }
```

1.4 Template Cpp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define _ ios_base::sync_with_stdio(0); cin.tie(0);
5
6 #define all(a) a.begin(), a.end()
7 #define int long long int
8 #define double long double
9 #define vi vector<int>
10 #define endl "\n"
11 #define print_v(a) for(auto x : a) cout << x << " "; cout << endl
12 #define print_vp(a) for(auto x : a) cout << x.F << " " << x.S << endl
13 #define print2(a,x,y) for(int i = x; i < y; i++) cout<< a[i]<< " "; cout
    << endl
14 #define f(i,s,e) for(int i=s;i<e;i++)
15 #define rf(i,e,s) for(int i=e-1;i>=s;i--)
16 #define CEIL(a, b) ((a) + (b - 1))/b
17 #define TRUNC(x) floor(x * 100) / 100
18
19 #define dbg(x) cout << #x << " = " << x << " ";
20 #define dbg1(x) cout << #x << " = " << x << endl;
21 #define bug(...) __f (#__VA_ARGS__, __VA_ARGS__)
22
23 const int INF = 0x7f3f3f3f;
24 const int LINF = 0x3f3f3f3f3f3f3f3f;
25 const double PI = acos(-1);
26 const int MAX = 1e6+10; // 10^6 + 10
27
28 template <typename Arg1> void __f (const char* name, Arg1&& arg1) { cout
    << name << " : " << arg1 << endl; }
29 template <typename Arg1, typename... Args> void __f (const char* names,
    Arg1&& arg1, Args&&... args) {
30     const char* comma = strchr (names + 1, ',');
31     cout.write (names, comma - names) << " : " << arg1 << " | "; __f (
        comma + 1, args...);
32 }
33
34 void solve() {
35
36 }
37
38 int32_t main() { _
39
40     clock_t z = clock();
41
42     int t = 1; // cin >> t;
43     while (t--)
44         //while(cin >> a >> b)
45             solve();
46
47     cerr << fixed << "Run Time : " << ((double)(clock() - z) /
        CLOCKS_PER_SEC) << endl;
48     return 0;
49 }
```

1.5 Files

```
1 #!/bin/bash
2
3 for c in {a..f}; do
4     cp temp.cpp "$c.cpp"
5     echo "$c" > "$c.txt"
6     if [ "$c" = "$letter" ]; then
7         break
8     fi
9 done
```

1.6 Template Python

```
1 import sys
2 import math
3 import bisect
4 from sys import stdin,stdout
5 from math import gcd,floor,sqrt,log
6 from collections import defaultdict as dd
7 from bisect import bisect_left as bl,bisect_right as br
8
9 sys.setrecursionlimit(100000000)
10
11 inp =lambda: int(input())
12 strng =lambda: input().strip()
13 jn =lambda x,l: x.join(map(str,l))
14 str1 =lambda: list(input().strip())
15 mul =lambda: map(int,input().strip().split())
16 mulf =lambda: map(float,input().strip().split())
17 seq =lambda: list(map(int,input().strip().split()))
18
19 ceil =lambda x: int(x) if(x==int(x)) else int(x)+1
20 ceildiv=lambda x,d: x//d if(x%d==0) else x//d+1
21
22 flush =lambda: stdout.flush()
23 stdstr =lambda: stdin.readline()
24 stdint =lambda: int(stdin.readline())
25 stdpr =lambda x: stdout.write(str(x))
26
27 mod=1000000007
28
29 #main code
30
31 a = None
32 b = None
33 lista = None
34
35 def ident(*args):
36     if len(args) == 1:
37         return args[0]
38     return args
39
40
41 def parsin(*, l=1, vpl=1, s=" "):
42     if l == 1:
```

```

43     if vpl == 1: return ident(input())
44     else: return list(map(ident, input().split(s)))
45 else:
46     if vpl == 1: return [ident(input()) for _ in range(1)]
47     else: return [list(map(ident, input().split(s))) for _ in range(1)]
48 ]
49
50 def solve():
51     pass
52
53 # if __name__ == '__main__':
54 def main():
55     st = clk()
56
57     escolha = "in"
58     #escolha = "num"
59
60     match escolha:
61         case "in":
62             # Ël infinitas linhas agrupadas de 2 em 2
63             # pra infinitos valores em 1 linha pode armazenar em uma lista
64             while True:
65                 global a, b
66                 try: a, b = input().split()
67                 except (EOFError): break #permite ler todas as linhahs
68             dentro do .txt
69             except (ValueError): pass # consegue ler êat linhas em
70             branco
71             else:
72                 a, b = int(a), int(b)
73                 solve()
74
75         case "num":
76             global lista
77             # int l; cin >> l; while(l--){for(i=0; i<vpl; i++)}
78             # retorna listas com inputs de cada linha
79             # leia l linhas com vpl valores em cada uma delas
80             # caseo seja mais de uma linha, retorna lista com listas
81             de inputs
82             lista = parsin(l=2, vpl=5)
83             solve()
84
85     sys.stderr.write(f"Run Time : {(clk() - st):.6f} seconds\n")
86
87 main()

```

2 Informações

2.1 Vector

```

1 // INICIALIZAR
2 vector<int> v (n); // n ôcpias de 0
3 vector<int> v (n, v); // n ôcpias de v
4
5 // PUSH_BACK

```

```

6 // Complexidade: O(1) amortizado (O(n) se realocar)
7 v.push_back(x);
8
9 // REMOVE
10 // Complexidade: O(n)
11 v.erase(v.begin() + i);
12
13 // INSERIR
14 // Complexidade: O(n)
15 v.insert(v.begin() + i, x);
16
17 // ORDENAR
18 // Complexidade: O(n log(n))
19 sort(v.begin(), v.end());
20 sort(all(v));
21
22 // BUSCA BINARIA
23 // Complexidade: O(log(n))
24 // Retorno: true se existe, false se ão existe
25 binary_search(v.begin(), v.end(), x);
26
27 // FIND
28 // Complexidade: O(n)
29 // Retorno: iterador para o elemento, v.end() se ão existe
30 find(v.begin(), v.end(), x);
31
32 // CONTAR
33 // Complexidade: O(n)
34 // Retorno: únmero de êocorrncias
35 count(v.begin(), v.end(), x);

```

2.2 Sort

```

1 vector<int> v;
2 // Sort Crescente:
3 sort(v.begin(), v.end());
4 sort(all(v));
5
6 // Sort Decrescente:
7 sort(v.rbegin(), v.rend());
8 sort(all(v), greater<int>());
9
10 // Sort por uma çãfuno:
11 auto cmp = [](int a, int b) { return a > b; }; // { 2, 3, 1 } -> { 3,
12 2, 1 }
13 auto cmp = [](int a, int b) { return a < b; }; // { 2, 3, 1 } -> { 1,
14 2, 3 }
15 sort(v.begin(), v.end(), cmp);
16 sort(all(v), cmp);
17
18 // Sort por uma çãfuno (çãcomparao de pares):
19 auto cmp = [](pair<int, int> a, pair<int, int> b) { return a.second >
20 b.second; };

```

2.3 Priority Queue

```

1 // HEAP CRESCENTE {5,4,3,2,1}
2 priority_queue<int> pq; // max heap
3 // maior elemento:
4 pq.top();
5
6 // HEAP DECRESCENTE {1,2,3,4,5}
7 priority_queue<int, vector<int>, greater<int>> pq; // min heap
8 // menor elemento:
9 pq.top();
10
11 // REMOVER ELEMENTO
12 // Complexidade: O(n)
13 // Retorno: true se existe, false se não existe
14 pq.remove(x);
15
16 // INSERIR ELEMENTO
17 // Complexidade: O(log(n))
18 pq.push(x);
19
20 // REMOVER TOP
21 // Complexidade: O(log(n))
22 pq.pop();
23
24 // TAMANHO
25 // Complexidade: O(1)
26 pq.size();
27
28 // VAZIO
29 // Complexidade: O(1)
30 pq.empty();
31
32 // LIMPAR
33 // Complexidade: O(n)
34 pq.clear();
35
36 // ITERAR
37 // Complexidade: O(n)
38 for (auto x : pq) {}
39
40 // çãOrdenao por çãfuno customizada passada por parametro ao criar a pq
41 // Complexidade: O(n log(n))
42 auto cmp = [](int a, int b) { return a > b; };
43 priority_queue<int, vector<int>, decltype(cmp)> pq(cmp);

```

3 Estruturas

3.1 Sparse Table Disjunta

```

1 // Sparse Table Disjunta
2 //
3 // Resolve qualquer operacao associativa
4 // MAX2 = log(MAX)
5 //
6 // Complexidades:
7 // build - O(n log(n))
8 // query - O(1)

```

```

9
10 namespace sparse {
11     int m[MAX2][2*MAX], n, v[2*MAX];
12     int op(int a, int b) { return min(a, b); }
13     void build(int n2, int* v2) {
14         n = n2;
15         for (int i = 0; i < n; i++) v[i] = v2[i];
16         while (n&(n-1)) n++;
17         for (int j = 0; (1<<j) < n; j++) {
18             int len = 1<<j;
19             for (int c = len; c < n; c += 2*len) {
20                 m[j][c] = v[c], m[j][c-1] = v[c-1];
21                 for (int i = c+1; i < c+len; i++) m[j][i] = op(m[j][i-1],
22                                     v[i]);
23                 for (int i = c-2; i >= c-len; i--) m[j][i] = op(v[i], m[j]
24                                     ][i+1]);
25             }
26         }
27     }
28     int query(int l, int r) {
29         if (l == r) return v[l];
30         int j = __builtin_clz(1) - __builtin_clz(l^r);
31         return op(m[j][l], m[j][r]);
32     }
33 }

```

3.2 Seg Tree

```

1 // SegTree
2 //
3 // Query: soma do range [a, b]
4 // Update: soma x em cada elemento do range [a, b]
5 //
6 // Complexidades:
7 // build - O(n)
8 // query - O(log(n))
9 // update - O(log(n))
10 namespace seg {
11
12     int seg[4*MAX];
13     int n, *v;
14
15     int op(int a, int b) { return a + b; }
16
17     int build(int p=1, int l=0, int r=n-1) {
18         if (l == r) return seg[p] = v[l];
19         int m = (l+r)/2;
20         return seg[p] = op(build(2*p, l, m), build(2*p+1, m+1, r));
21     }
22
23     void build(int n2, int* v2) {
24         n = n2, v = v2;
25         build();
26     }
27
28     int query(int a, int b, int p=1, int l=0, int r=n-1) {
29         if (a <= l and r <= b) return seg[p];
30     }
31 }

```

```

30     if (b < l or r < a) return 0;
31     int m = (l+r)/2;
32     return op(query(a, b, 2*p, l, m), query(a, b, 2*p+1, m+1, r));
33 }
34
35 int update(int a, int b, int x, int p=1, int l=0, int r=n-1) {
36     if (a <= l and r <= b) return seg[p];
37     if (b < l or r < a) return seg[p];
38     int m = (l+r)/2;
39     return seg[p] = op(update(a, b, x, 2*p, l, m), update(a, b, x, 2*p
40 +1, m+1, r));
41 }
42
43 };

```

4 Grafos

4.1 Bfs

```

1 // BFS com informacoes adicionais sobre a distancia e o pai de cada
   vertice
2 // Complexidade: O(V + E), onde V eh o numero de vertices e E o numero de
   areqas
3 vector<vector<int>> adj; // liqa de adjacencia
4 int n, s; // n = numero de vertices, s = vertice inicial
5
6 vector<bool> used(n);
7 vector<int> d(n), p(n);
8
9 void bfs(int s) {
10     queue<int> q;
11     q.push(s);
12     used[s] = true;
13     d[s] = 0;
14     p[s] = -1;
15
16     while (!q.empty()) {
17         int v = q.front();
18         q.pop();
19         for (int u : adj[v]) {
20             if (!used[u]) {
21                 used[u] = true;
22                 q.push(u);
23                 d[u] = d[v] + 1;
24                 p[u] = v;
25             }
26         }
27     }
28 }
29
30 //pra uma bfs que n guarda o backtracking:
31 void bfs(int p) {
32     memset(visited, 0, sizeof visited);
33     queue<int> q;
34     q.push(p);
35
36     while (!q.empty()) {

```

```

37     int curr = q.top();
38     q.pop();
39     if (visited[curr]==1) continue;
40     visited[curr]=1;
41     // process current node here
42
43     for (auto i : adj[curr]) {
44         q.push(i);
45     }
46 }
47
48 }

```

4.2 Dijkstra

```

1 vector<vector<pair<int, int>>> adj; // adj[a] = [{b, w}]
2 int n;
3
4 vector<int> dist(n, LLINF);
5 vector<int> parent(n, -1);
6 vector<bool> used(n);
7
8 //Complexidade: O((V + E)logV)
9 void dijkstra(int s) {
10
11     dist[s] = 0;
12
13     priority_queue<pair<int, int>> q;
14     q.push({0, s});
15
16     while (!q.empty()) {
17         int a = q.top().second; q.pop();
18
19         if (used[a]) continue;
20         used[a] = true;
21
22         for (auto [b, w] : adj[a]) {
23             if (dist[a] + w < dist[b]) {
24                 dist[b] = dist[a] + w;
25                 parent[b] = a;
26                 q.push({-dist[b], b});
27             }
28         }
29     }
30 }
31
32 //Complexidade: O(V)
33 vector<int> restorePath(int v) {
34     vector<int> path;
35     for (int u = v; u != -1; u = parent[u])
36         path.push_back(u);
37     reverse(path.begin(), path.end());
38     return path;
39 }

```

4.3 Kruskal

```

1 // Kruskal
2 //
3 // Gera e retorna uma AGM e seu custo total a partir do vetor de arestas (
4 // do grafo
5 //
6 //  $O(m \log(m) + m a(m))$ 
7
8 vector<tuple<int, int, int>> edg; // {peso,x,y}
9 vector<int> id, sz;
10
11 int find(int p){ //  $O(a(N))$  amortizado
12     return id[p] = (id[p] == p ? p : find(id[p]));
13 }
14
15 void uni(int p, int q) { //  $O(a(N))$  amortizado
16     p = find(p), q = find(q);
17     if(p == q) return;
18     if(sz[p] > sz[q]) swap(p,q);
19     id[p] = q, sz[q] += sz[p];
20 }
21
22 pair<int, vector<tuple<int, int, int>>> kruskal() {
23
24     sort(edg.begin(), edg.end());
25
26     int cost = 0;
27     vector<tuple<int, int, int>> mst; // opcional
28     for (auto [w,x,y] : edg) if (find(x) != find(y)) {
29         mst.emplace_back(w, x, y); // opcional
30         cost += w;
31         uni(x,y);
32     }
33     return {cost, mst};
34 }

```

4.4 Dfs

```

1
2 vector<int> adj[MAXN];
3 int visited[MAXN];
4
5 void dfs(int p) {
6     memset(visited, 0, sizeof visited);
7     stack<int> st;
8     st.push(p);
9
10    while (!st.empty()) {
11        int curr = st.top();
12        st.pop();
13        if (visited[curr]==1) continue;
14        visited[curr]=1;
15        // process current node here
16
17        for (auto i : adj[curr]) {
18            st.push(i);
19        }

```

```

20    }
21 }

```

5 Matematica

5.1 Mdc Multiplo

```

1 int mdc_many(vector<int> arr) {
2     int result = arr[0];
3     for (size_t i = 1; i < arr.size(); i++) {
4         result = mdc(arr[i], result);
5
6         if(result == 1)
7             return 1;
8     }
9     return result;
10 }

```

5.2 Factorial

```

1 unordered_map<int, int> memo;
2
3 int factorial(int n) {
4     if (n == 0 || n == 1) return 1;
5     if (memo.find(n) != memo.end()) return memo[n];
6     return memo[n] = n * factorial(n - 1);
7 }

```

5.3 Mmc Multiplo

```

1 int mmc(vector<int> arr) {
2     int result = arr[0];
3     for(size_t i = 1; i < arr.size(); i++)
4         result = (arr[i] * result / mmc_util(arr[i], result ));
5     return ans;
6 }

```

5.4 Fast Exponentiation

```

1 const int mod = 1e9+7;
2 int fexp(int a, int b)
3 {
4     int ans = 1;
5     while (b)
6     {
7         if (b & 1)
8             ans = ans * a % mod;
9         a = a * a % mod;
10        b >>= 1;
11    }
12    return ans;
13 }

```

5.5 Sieve


```

1 // Crivo de Eratstenes para gerar primos até um limite 'lim'
2 // Complexidade: O(n log log n), onde n é o limite
3 const int ms = 1e6 + 5;
4 bool notPrime[ms]; // notPrime[i] é verdadeiro se i não é um número
    primo
5 int primes[ms], qnt; // primes[] armazena os números primos e qnt é a
    quantidade de primos encontrados
6
7 void sieve(int lim)
8 {
9     primes[qnt++] = 1; // adiciona 1 como um número primo se ele for válido
    no problema
10    for (int i = 2; i <= lim; i++)
11    {
12        if (notPrime[i])
13            continue; // se i não é primo, pula
14        primes[qnt++] = i; // i é primo, adiciona em primes
15        for (int j = i + i; j <= lim; j += i) // marca todos os múltiplos de i
            notPrime[j] = true;
16    }
17 }
18 }

```

5.6 Fact Grande

```

1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 vector<int> multiply(const vector<int>& num, int multiplier) {
7     vector<int> result;
8     int carry = 0;
9
10    for (size_t i = 0; i < num.size(); ++i) {
11        int product = num[i] * multiplier + carry;
12        result.push_back(product % 10);
13        carry = product / 10;
14    }
15
16    while (carry) {
17        result.push_back(carry % 10);
18        carry /= 10;
19    }
20
21    return result;
22 }
23
24 vector<int> factorial(int n) {
25     vector<int> result;
26     result.push_back(1);
27
28     for (int i = 2; i <= n; ++i) {
29         result = multiply(result, i);
30     }
31 }

```

```

32     reverse(result.begin(), result.end());
33     return result;
34 }

```

5.7 Mdc

```

1 int mdc(int x, int y) {
2     return y ? mdc(y, x % y) : abs(x);
3 }

```

5.8 Primo

```

1 bool prime(int a) {
2     if (a == 1)
3         return 0;
4     for (int i = 2; i <= round(sqrt(a)); ++i)
5         if (a % i == 0)
6             return 0;
7     return 1;
8 }

```

5.9 Miller Rabin

```

1 // Miller-Rabin
2 //
3 // Testa se n eh primo, n <= 3 * 10^18
4 //
5 // O(log(n)), considerando multiplicacao
6 // e exponenciacao constantes
7
8 int mul(int a, int b, int m) {
9     int ret = a*b - int((long double)1/m*a*b+0.5)*m;
10    return ret < 0 ? ret+m : ret;
11 }
12
13 int pow(int x, int y, int m) {
14     if (!y) return 1;
15     int ans = pow(mul(x, x, m), y/2, m);
16     return y%2 ? mul(x, ans, m) : ans;
17 }
18
19 bool prime(int n) {
20     if (n < 2) return 0;
21     if (n <= 3) return 1;
22     if (n % 2 == 0) return 0;
23     int r = __builtin_ctzint(n - 1), d = n >> r;
24
25     // com esses primos, o teste funciona garantido para n <= 2^64
26     // funciona para n <= 3*10^24 com os primos ate 41
27     for (int a : {2, 325, 9375, 28178, 450775, 9780504, 795265022}) {
28         int x = pow(a, d, n);
29         if (x == 1 or x == n - 1 or a % n == 0) continue;
30
31         for (int j = 0; j < r - 1; j++) {
32             x = mul(x, x, n);
33             if (x == n - 1) break;
34         }

```

```

35     if (x != n - 1) return 0;
36 }
37 return 1;
38 }

```

5.10 Fatorial Grande

```

1 void multiply(vector<int>& num, int x) {
2     int carry = 0;
3     for (int i = 0; i < num.size(); i++) {
4         int prod = num[i] * x + carry;
5         num[i] = prod % 10;
6         carry = prod / 10;
7     }
8     while (carry != 0) {
9         num.push_back(carry % 10);
10        carry /= 10;
11    }
12 }
13
14 vector<int> factorial(int n) {
15     vector<int> result;
16     result.push_back(1);
17     for (int i = 2; i <= n; i++) {
18         multiply(result, i);
19     }
20     return result;
21 }

```

5.11 Sieve Linear

```

1 // Sieve de Eratosthenes com linear sieve
2 // Encontra todos os números primos no intervalo [2, N]
3 // Complexidade: O(N)
4
5 const int N = 10000000;
6 vector<int> lp(N + 1); // lp[i] = menor fator primo de i
7 vector<int> pr;        // vetor de primos
8
9 for (int i = 2; i <= N; ++i)
10 {
11     if (lp[i] == 0)
12     {
13         lp[i] = i;
14         pr.push_back(i);
15     }
16     for (int j = 0; i * pr[j] <= N; ++j)
17     {
18         lp[i * pr[j]] = pr[j];
19         if (pr[j] == lp[i])
20         {
21             break;
22         }
23     }
24 }

```

5.12 Numeros Grandes

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4
5 using namespace std;
6
7 void normalize(vector<int>& num) {
8     int carry = 0;
9     for (int i = 0; i < num.size(); ++i) {
10        num[i] += carry;
11        carry = num[i] / 10;
12        num[i] %= 10;
13    }
14
15    while (carry > 0) {
16        num.push_back(carry % 10);
17        carry /= 10;
18    }
19 }
20
21 pair<int, vector<int>> makePair(int sign, const vector<int>& magnitude) {
22     return {sign, magnitude};
23 }
24
25 pair<int, vector<int>> bigSum(const pair<int, vector<int>>& a, const pair<
26     int, vector<int>>& b) {
27     if (a.first == b.first) {
28         vector<int> result(max(a.second.size(), b.second.size()), 0);
29         transform(a.second.begin(), a.second.end(), b.second.begin(),
30             result.begin(), plus<int>());
31         normalize(result);
32         return makePair(a.first, result);
33     } else {
34         // If signs are different, perform subtraction
35         vector<int> result(max(a.second.size(), b.second.size()), 0);
36         transform(a.second.begin(), a.second.end(), b.second.begin(),
37             result.begin(), minus<int>());
38         normalize(result);
39         return makePair(a.first, result);
40     }
41 }
42
43 pair<int, vector<int>> bigSub(const pair<int, vector<int>>& a, const pair<
44     int, vector<int>>& b) {
45     return bigSum(a, makePair(-b.first, b.second));
46 }
47
48 pair<int, vector<int>> bigMult(const pair<int, vector<int>>& a, const pair<
49     int, vector<int>>& b) {
50     vector<int> result(a.second.size() + b.second.size(), 0);
51
52     for (int i = 0; i < a.second.size(); ++i) {
53         for (int j = 0; j < b.second.size(); ++j) {
54             result[i + j] += a.second[i] * b.second[j];
55         }
56     }
57 }

```

```

51 }
52
53 normalize(result);
54 return makePair(a.first * b.first, result);
55 }
56
57
58
59 void printNumber(const pair<int, vector<int>>& num) {
60     if (num.first == -1) {
61         cout << '-';
62     }
63
64     for (auto it = num.second.rbegin(); it != num.second.rend(); ++it) {
65         cout << *it;
66     }
67     cout << endl;
68 }
69
70 int main() {
71     // Example usage
72     pair<int, vector<int>> num1 = makePair(1, {1, 2, 3}); // Representing
73     +321
74     pair<int, vector<int>> num2 = makePair(-1, {4, 5, 6}); //
75     Representing -654
76
77     pair<int, vector<int>> sum = bigSum(num1, num2);
78     pair<int, vector<int>> difference = bigSub(num1, num2);
79     pair<int, vector<int>> product = bigMult(num1, num2);
80
81     cout << "Sum: ";
82     printNumber(sum);
83
84     cout << "Difference: ";
85     printNumber(difference);
86
87     cout << "Product: ";
88     printNumber(product);
89
90     return 0;
91 }

```

5.13 Mmc

```

1 int mmc(int x, int y) {
2     return (x && y ? (return abs(x) / mdc(x, y) * abs(y)) : abs(x | y));
3 }

```

6 Strings

6.1 Ocorrencias

```

1 /**
2  * @brief str.find() aprimorado
3  * @param str string to be analysed

```

```

4  * @param sub substring to be searched
5  * @return vector<int> com indices de todas as @ocorrencias de uma
6  * substring em uma string
7  */
8
9 vector<int> ocorrencias(string str, string sub){
10
11     vector<int> ret;
12     int index = str.find(sub);
13     while(index != -1){
14         ret.push_back(index);
15         index = str.find(sub, index+1);
16     }
17
18     return ret;
19 }

```

6.2 Upper Case

```

1 string to_upper(string a) {
2     for (int i=0; i<(int)a.size(); ++i)
3         if (a[i]>='a' && a[i]<='z')
4             a[i]-='a'-'A';
5     return a;
6 }
7
8 // para checar se e uppercase: isupper(c);

```

6.3 Palindromo

```

1 bool isPalindrome(string str) {
2     for (int i = 0; i < str.length() / 2; i++) {
3         if (str[i] != str[str.length() - i - 1]) {
4             return false;
5         }
6     }
7     return true;
8 }

```

6.4 Split Cria

```

1 vector<string> split(string s, string del = " ") {
2     vector<string> retorno;
3     int start, end = -1*del.size();
4     do {
5         start = end + del.size();
6         end = s.find(del, start);
7         retorno.push_back(s.substr(start, end - start));
8     } while (end != -1);
9     return retorno;
10 }

```

6.5 Remove Acento

```

1 string removeAcento(string str) {
2

```

```

3     string comAcento = "áéíóúâêôãõä";
4     string semAcento = "aeiouaeoaoa";
5
6     for(int i = 0; i < str.size(); i++){
7         for(int j = 0; j < comAcento.size(); j++){
8             if(str[i] == comAcento[j]){
9                 str[i] = semAcento[j];
10                break;
11            }
12        }
13    }
14
15    return str;
16 }

```

6.6 Chaves Colchetes Parenteses

```

1 def balanced(string) -> bool:
2     stack = []
3
4     for i in string:
5         if i in '([{' : stack.append(i)
6
7         elif i in ')]}' :
8             if (not stack) or ((stack[-1],i) not in [(',','), ([','), ('{','}')]):
9                 return False
10            else:
11                stack.pop()
12
13    return not stack

```

6.7 Lower Case

```

1 string to_lower(string a) {
2     for (int i=0;i<(int)a.size();++i)
3         if (a[i]>='A' && a[i]<='Z')
4             a[i]+='a'-'A';
5     return a;
6 }
7
8 // para checar se é lowercase: islower(c);

```

6.8 Lexicograficamente Minima

```

1 // A simple C++ program to find lexicographically minimum rotation of a
   given string
2 #include <iostream>
3 #include <algorithm>
4 using namespace std;
5
6 // This functionr return lexicographically minimum rotation of str
7 string minLexRotation(string str)
8 {
9     // Find length of given string
10    int n = str.length();

```

```

11
12    // Create an array of strings to store all rotations
13    string arr[n];
14
15    // Create a concatenation of string with itself
16    string concat = str + str;
17
18    // One by one store all rotations of str in array.
19    // A rotation is obtained by getting a substring of concat
20    for (int i = 0; i < n; i++)
21        arr[i] = concat.substr(i, n);
22
23    // Sort all rotations
24    sort(arr, arr+n);
25
26    // Return the first rotation from the sorted array
27    return arr[0];
28 }
29
30 // Driver program to test above function
31 int main()
32 {
33     cout << minLexRotation("GEEKSFORGEEKS") << endl;
34     cout << minLexRotation("GEEKSQUIZ") << endl;
35     cout << minLexRotation("BCABDADAB") << endl;
36 }

```

6.9 Split

```

1 //split a string with a delimiter
2 //eg.: split("ã0l, tudo bem?", " ") -> ["ã0l,", "tudo", "bem?"]
3
4 vector<string> split(string in, string delimiter){
5     vector<string> numbers;
6     string token = "";
7     int pos;
8     while(true){
9         pos = in.find(delimiter);
10        if(pos == -1) break;
11        token = in.substr(0, pos);
12        numbers.push_back(token);
13        in = in.erase(0, pos + delimiter.length());
14    }
15    numbers.push_back(in);
16    return numbers;
17 }

```

7 Vector

7.1 Teste

7.2 Remove Repetitive

```

1 vector<int> removeRepetitive(const vector<int>& vec) {
2
3     unordered_set<int> s;
4     s.reserve(vec.size());
5
6     vector<int> ans;
7
8     for (int num : vec) {
9         if (s.insert(num).second)
10             v.push_back(num);
11     }
12
13     return ans;
14 }

```

7.3 Elemento Mais Frequente

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 // Encontra o unico elemento mais frequente em um vetor
5 // Complexidade: O(n)
6 int maxFreq1(vector<int> v) {
7     int res = 0;
8     int count = 1;
9
10    for(int i = 1; i < v.size(); i++) {
11
12        if(v[i] == v[res])
13            count++;
14        else
15            count--;
16
17        if(count == 0) {
18            res = i;
19            count = 1;
20        }
21    }
22
23    return v[res];
24 }
25
26 // Encontra os elemento mais frequente em um vetor
27 // Complexidade: O(n)
28 vector<int> maxFreqn(vector<int> v)
29 {
30     unordered_map<int, int> hash;
31     for (int i = 0; i < v.size(); i++)
32         hash[v[i]]++;
33
34     int max_count = 0, res = -1;
35     for (auto i : hash) {
36         if (max_count < i.second) {
37             res = i.first;
38             max_count = i.second;
39         }
40     }

```

```

41
42     vector<int> ans;
43     for (auto i : hash) {
44         if (max_count == i.second) {
45             ans.push_back(i.first);
46         }
47     }
48
49     return ans;
50 }

```

8 Outros

8.1 Binario

```

1 string decimal_to_binary(int dec) {
2     string binary = "";
3     while (dec > 0) {
4         int bit = dec % 2;
5         binary = to_string(bit) + binary;
6         dec /= 2;
7     }
8     return binary;
9 }
10
11 int binary_to_decimal(string binary) {
12     int dec = 0;
13     int power = 0;
14     for (int i = binary.length() - 1; i >= 0; i--) {
15         int bit = binary[i] - '0';
16         dec += bit * pow(2, power);
17         power++;
18     }
19     return dec;
20 }

```

8.2 Horario

```

1 int cts(int h, int m, int s) {
2     int total = (h * 3600) + (m * 60) + s;
3     return total;
4 }
5
6 tuple<int, int, int> cth(int total_seconds) {
7     int h = total_seconds / 3600;
8     int m = (total_seconds % 3600) / 60;
9     int s = total_seconds % 60;
10    return make_tuple(h, m, s);
11 }

```

8.3 Max Subarray Sum

```

1 int maxSubarraySum(vector<int> x){
2
3     int best = 0, sum = 0;

```

```

4     for (int k = 0; k < n; k++) {
5         sum = max(x[k], sum+x[k]);
6         best = max(best, sum);
7     }
8     return best;
9 }

```

8.4 Binary Search

```

1 int BinarySearch(<vector>int arr, int x){
2     int k = 0;
3     int n = arr.size();
4
5     for (int b = n/2; b >= 1; b /= 2) {
6         while (k+b < n && arr[k+b] <= x) k += b;

```

```

7     }
8     if (arr[k] == x) {
9         return k;
10    }
11 }

```

8.5 Fibonacci

```

1 int fib(int n){
2     if(n <= 1){
3         return n;
4     }
5     return fib(n - 1) + fib(n - 2);
6 }

```