



Pedro Augusto Ulisses Andrade Lucas Andrade

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

Contents

1	Utils	2	4	Estruturas	6
1.1	Makefile	2	4.1	Bittree	6
1.2	Limites	2	4.2	Sparse Table Disjunta	7
1.3	Mini Template Cpp	2	4.3	Seg Tree	7
1.4	Template Cpp	3	5	Grafos	8
1.5	Files	3	5.1	Bfs Matriz	8
1.6	Template Python	3	5.2	Bfs	8
2	Informações	4	5.3	Dijkstra	8
2.1	Vector	4	5.4	Successor Graph	9
2.2	Sort	4	5.5	Kosaraju	9
2.3	Priority Queue	4	5.6	Euler Tree	9
2.4	String	5	5.7	Kruskal	9
3	Combinatoria	5	5.8	Dfs	10
3.1	Arranjo Simples	5	6	Matematica	10
3.2	Combinacao Com Repeticao	5	6.1	N Fibonacci	10
3.3	Combinacao Simples	5	6.2	Mdc Multiplo	11
3.4	Permutacao Circular	6	6.3	Factorial	11
3.5	@ Tabela	6	6.4	Mmc Multiplo	11
3.6	Permutacao Com Repeticao	6	6.5	Fast Exponentiation	11
3.7	@ Factorial	6	6.6	Sieve	11
3.8	Permutacao Simples	6	6.7	Mdc	11
3.9	Arranjo Com Repeticao	6	6.8	Fatorial Grande	11
			6.9	Sieve Linear	11
			6.10	Primo	12

6.11	Miller Rabin	12
6.12	Numeros Grandes	12
6.13	Numeros Grandes	13
6.14	Mmc	13
7	Strings	14
7.1	Ocorrencias	14
7.2	Palindromo	14
7.3	Split Cria	14
7.4	Remove Acento	14
7.5	Chaves Colchetes Parenteses	14
7.6	Lower Iupper	14
7.7	Lexicograficamente Minima	15
7.8	Numeros E Char	15
8	Vector	15
8.1	Remove Repetitive	15
8.2	Troco	15
8.3	Contar Soma Subarray Igual K	15
8.4	Elemento Mais Frequente	15
8.5	Maior Sequencia Subsequente	16
9	Outros	16
9.1	Binario	16
9.2	Horario	16
9.3	Max Subarray Sum	17
9.4	Binary Search	17
9.5	Fibonacci	17

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 compile:
5     g++ -g $(f).cpp $(CXXFLAGS) -o $(f)
6 exec:
7     ./$(f)
8
9 runc: compile exec
10 runci: compile
11     ./$(f) < $(f).txt
12
13 clearexe:
14     find . -maxdepth 1 -type f -executable -exec rm {} +
15 cleartxt:
16     find . -type f -name "*.txt" -exec rm -f {} \;
17 clear: clearexe cleartxt
18
19 runp:
20     python3 $(f).py
21 runpt:
22     python3 $(f).py < $(f).txt
```

1.2 Limites

```
1 // LIMITES DE REPRESENTACAO DE DADOS
2
3      tipo      | bits |      minimo .. maximo      | precisao decim.
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 float          | 32   |     1.2 x 10^-38 .. 3.4 x 10^38 | 6-9
15 double         | 64   |     2.2 x 10^-308 .. 1.8 x 10^308 | 15-17
16 long double    | 80   |     3.4 x 10^-4932 .. 1.1 x 10^4932 | 18-19
17 BigInt/Dec(java) 1 x 10^-2147483648 .. 1 x 10^2147483647 | 0
18
19 // LIMITES DE MEMORIA
20
21 1MB = 1,048,576 bool
22 1MB = 524,288 char
23 1MB = 262,144 int32_t
24 1MB = 131,072 int64_t
25 1MB = 65,536 float
```

```
26 1MB = 32,768 double
27 1MB = 16,384 long double
28 1MB = 16,384 BigInteger / BigDecimal
29
30 // ESTOURAR TEMPO
31
32 input size      | complexidade para 1 s
33 -----+-----
34 [10,11]         | O(n!), O(n^6)
35 [15,18]         | O(2^n * n^2)
36 [18, 22]        | O(2^n * n)
37 ... 100         | O(n^4)
38 ... 400         | O(n^3)
39 ... 2*10^3       | O(n^2 * log n)
40 ... 5*10^4       | O(n^2)
41 ... 10^5         | O(n^2 log n)
42 ... 10^6         | O(n log n)
43 ... 10^7         | O(n log log n)
44 ... 10^8         | O(n), O(log n), O(1)
45
46
47 // FATORIAL
48
49 12! = 479.001.600 [limite do (u)int]
50 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 endl "\n"
10 #define print_v(a) for(auto x : a) cout << x << " "; cout << endl
11 #define f(i,s,e) for(int i=s;i<e;i++)
12 #define rf(i,e,s) for(int i=e-1;i>=s;i--)
13 #define dbg(x) cout << #x << " = " << x << endl;
14
15 void solve() {
16
17 }
18
19 int32_t main() { _
20
21     int t = 1; // cin >> t;
22     while (t--) {
23         solve();
24     }
25
26     return 0;
27 }
```

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 f(i,s,e)     for(int i=s;i<e;i++)
14 #define rf(i,e,s)    for(int i=e-1;i>=s;i--)
15 #define CEIL(a, b)   ((a) + (b - 1))/b // Arredonda para cima
16 #define TRUNC(x, n)  floor(x * pow(10, n))/pow(10, n) // Trunca n casas
17 #define ROUND(x, n)  round(x * pow(10, n))/pow(10, n) // Arredonda n c
18
19 #define dbg(x) cout << #x << " = " << x << " ";
20 #define dbg1(x) cout << #x << " = " << x << endl;
21
22 const int INF = 0x7f3f3f3f; // 0x7f com 3 3f's (10^9)
23 const int LINf = 0x3f3f3f3f3f3f3f3f; // 0x com 7 3f's (10^18)
24 const int MAX = 1e6+10; // 10^6 + 10
25
26 void solve() {
27
28 }
29
30 int32_t main() { _
31
32     clock_t z = clock();
33
34     int t = 1; // cin >> t;
35     while (t--) {
36         solve();
37     }
38
39     cerr << fixed << "Run Time : " << ((double)(clock() - z) /
40     CLOCKS_PER_SEC) << endl;
41     return 0;
42 }
```

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 strl = 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(l)]
47         else: return [list(map(ident, input().split(s))) for _ in range(l)]
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         # êl 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 linhas
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 ócpias de 0
3 vector<int> v (n, v); // n ócpias 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: úmero de âocorrncias
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 ão 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);

```

2.4 String

```

1 // INICIALIZAR
2 string s; // string vazia
3 string s (n, c); // n ócpias de c
4 string s (s); // ócpia de s
5 string s (s, i, n); // ócpia de s[i..i+n-1]
6
7 // SUBSTRING
8 // Complexidade: O(n)
9 s.substr(i, n); // substring de s[i..i+n-1]
10 s.substr(i, j - i + 1); // substring de s[i..j]
11
12 // TAMANHO
13 // Complexidade: O(1)
14 s.size(); // tamanho da string
15 s.empty(); // true se vazia, false se ão vazia
16
17 // MODIFICAR
18 // Complexidade: O(n)
19 s.push_back(c); // adiciona c no final
20 s.pop_back(); // remove o último
21 s += t; // concatena t no final
22 s.insert(i, t); // insere t a partir da çãposio i
23 s.erase(i, n); // remove n caracteres a partir da çãposio i
24 s.replace(i, n, t); // substitui n caracteres a partir da çãposio i por t

```

```

25 s.swap(t); // troca o úcontedo com t
26
27 // COMPARAR
28 // Complexidade: O(n)
29 s == t; // igualdade
30 s != t; // çdiferena
31 s < t; // menor que
32 s > t; // maior que
33 s <= t; // menor ou igual
34 s >= t; // maior ou igual
35
36 // BUSCA
37 // Complexidade: O(n)
38 s.find(t); // çãposio da primeira êocorrncia de t, ou string::npos se ão
    existe
39 s.rfind(t); // çãposio da última êocorrncia de t, ou string::npos se ão
    existe
40 s.find_first_of(t); // çãposio da primeira êocorrncia de um caractere de t
    , ou string::npos se ão existe
41 s.find_last_of(t); // çãposio da última êocorrncia de um caractere de t,
    ou string::npos se ão existe
42 s.find_first_not_of(t); // çãposio do primeiro caractere que ão áest em t
    , ou string::npos se ão existe
43 s.find_last_not_of(t); // çãposio do último caractere que ão áest em t, ou
    string::npos se ão existe
44
45 // SUBSTITUIR
46 // Complexidade: O(n)
47 s.replace(i, n, t); // substitui n caracteres a partir da çãposio i por t
48 s.replace(s.begin() + i, s.begin() + i + n, t.begin(), t.end()); //
    substitui n caracteres a partir da çãposio i por t
49 s.replace(s.begin() + i, s.begin() + i + n, t); // substitui n caracteres
    a partir da çãposio i por t
50 s.replace(s.begin() + i, s.begin() + i + n, n, c); // substitui n
    caracteres a partir da çãposio i por n ócpias de c

```

3 Combinatoria

3.1 Arranjo Simples

```

1 int arranjoSimples(int p, int n) {
2     return fact(n) / fact(n - p);
3 }

```

3.2 Combinacao Com Repeticao

```

1 int combinacaoComRepeticao(int p, int n) {
2     return fact(n + p - 1) / (fact(p) * fact(n - 1));
3 }

```

3.3 Combinacao Simples

```

1 int combinacaoSimples(int p, int n) {
2     return fact(n) / (fact(p) * fact(n - p));
3 }

```

3.4 Permutacao Circular

```
1 // Permutacao objetos em posicao simetrica em um circulo
2
3 int permutacaoCircular(int n) {
4     return fact(n - 1);
5 }
```

3.5 @ Tabela

```
1 // Sequencia de p elementos de um total de n
2
3 ORDEM \ REPETIC |          COM          |          SEM
4 -----+-----+-----+-----+-----+-----+-----+-----+-----+
5 IMPORTA      | ARRANJO COM REPETICAO | ARRANJO SIMPLES
6 NAO          | COMBINACAO COM REPETICAO | COMBINACAO SIMPLES
```

3.6 Permutacao Com Repeticao

```
1 // Trocar elementos de lugar quando ha termos repetidos (ANAGRAMA)
2
3 int permutacaoComRepeticao(string s) {
4     int n = s.size();
5     int ans = fact(n);
6     map<char, int> freq;
7     for (char c : s) {
8         freq[c]++;
9     }
10    for (auto [c, f] : freq) {
11        ans /= fact(f);
12    }
13    return ans;
14 }
```

3.7 @ Factorial

```
1 int factdp[20];
2
3 int fact(int n) {
4     if (n < 2) return 1;
5     if (factdp[n] != 0) return factdp[n];
6     return factdp[n] = n * fact(n - 1);
7 }
```

3.8 Permutacao Simples

```
1 // Agrupamentos distintos entre si pela ordem (FILA)
2 // Diferenca do arranjo: usa todos os elementos para o calculo
3 // SEM repeticao
4
5 int permutacaoSimples(int n) {
6     return fact(n);
7 }
```

3.9 Arranjo Com Repeticao

```
1 int arranjoComRepeticao(int p, int n) {
2     return pow(n, p);
3 }
```

4 Estruturas

4.1 Bittree

```
1 // C++ code to demonstrate operations of Binary Index Tree
2 #include <iostream>
3
4 using namespace std;
5
6 /*      n --> No. of elements present in input array.
7     BITree[0..n] --> Array that represents Binary Indexed Tree.
8     arr[0..n-1] --> Input array for which prefix sum is evaluated. */
9
10 // Returns sum of arr[0..index]. This function assumes
11 // that the array is preprocessed and partial sums of
12 // array elements are stored in BITree[].
13 int getSum(int BITree[], int index)
14 {
15     int sum = 0; // Initialize result
16
17     // index in BITree[] is 1 more than the index in arr[]
18     index = index + 1;
19
20     // Traverse ancestors of BITree[index]
21     while (index > 0)
22     {
23         // Add current element of BITree to sum
24         sum += BITree[index];
25
26         // Move index to parent node in getSum View
27         index -= index & (-index);
28     }
29     return sum;
30 }
31
32 // Updates a node in Binary Index Tree (BITree) at given index
33 // in BITree. The given value 'val' is added to BITree[i] and
34 // all of its ancestors in tree.
35 void updateBIT(int BITree[], int n, int index, int val)
36 {
37     // index in BITree[] is 1 more than the index in arr[]
38     index = index + 1;
39
40     // Traverse all ancestors and add 'val'
41     while (index <= n)
42     {
43         // Add 'val' to current node of BI Tree
44         BITree[index] += val;
45
46         // Update index to that of parent in update View
47         index += index & (-index);
48     }
```

```

49 }
50
51 // Constructs and returns a Binary Indexed Tree for given
52 // array of size n.
53 int *constructBITree(int arr[], int n)
54 {
55     // Create and initialize BITree[] as 0
56     int *BITree = new int[n+1];
57     for (int i=1; i<=n; i++)
58         BITree[i] = 0;
59
60     // Store the actual values in BITree[] using update()
61     for (int i=0; i<n; i++)
62         updateBIT(BITree, n, i, arr[i]);
63
64     // Uncomment below lines to see contents of BITree[]
65     //for (int i=1; i<=n; i++)
66     //    cout << BITree[i] << " ";
67
68     return BITree;
69 }
70
71
72 // Driver program to test above functions
73 int main()
74 {
75     int freq[] = {2, 1, 1, 3, 2, 3, 4, 5, 6, 7, 8, 9};
76     int n = sizeof(freq)/sizeof(freq[0]);
77     int *BITree = constructBITree(freq, n);
78     cout << "Sum of elements in arr[0..5] is "
79          << getSum(BITree, 5);
80
81     // Let use test the update operation
82     freq[3] += 6;
83     updateBIT(BITree, n, 3, 6); //Update BIT for above change in arr[]
84
85     cout << "\nSum of elements in arr[0..5] after update is "
86          << getSum(BITree, 5);
87
88     return 0;
89 }

```

4.2 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(1^r);
31         return op(m[j][l], m[j][r]);
32     }
33 }

```

4.3 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         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 }

```

5 Grafos

5.1 Bfs Matriz

```

1 // Description: BFS para uma matriz (n x m)
2 // Complexidade: O(n * m)
3
4 vector<vi> mat;
5 vector<vector<bool>> vis;
6 vector<pair<int,int>> mov = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};
7
8 bool valid(int x, int y) {
9     return (0 <= x and x < l and 0 <= y and y < c and !vis[x][y] /*and mat
10 [x][y]*/);
11 }
12
13 void bfs(int i, int j, int linhas, int columnas) {
14
15     mat.resize(l, vi(c));
16     vis.resize(l, vector<bool>(c, false));
17
18     queue<pair<int,int>> q; q.push({i, j});
19
20     while(!q.empty()) {
21
22         auto [u, v] = q.front(); q.pop();
23         vis[u][v] = true;
24
25         for(auto [x, y]: mov) {
26             if(valid(u+x, v+y)) {
27                 q.push({u+x,v+y});
28                 vis[u+x][v+y] = true;
29             }
30         }
31     }

```

5.2 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
4 int n; // n = numero de vertices
5 vector<bool> vis;

```

```

6 vector<int> d, p;
7 vector<vector<int>> adj; // liqa de adjacencia
8
9 void bfs(int s) {
10
11     vis.resize(n, false);
12     d.resize(n, -1);
13     p.resize(n, -1);
14
15     queue<int> q; q.push(s);
16     vis[s] = true, d[s] = 0, p[s] = -1;
17
18     while (!q.empty()) {
19         int v = q.front(); q.pop(); visited[v] = true;
20
21         for (int u : adj[v]) {
22             if (!vis[u]) {
23                 vis[u] = true;
24                 q.push(u);
25                 d[u] = d[v] + 1;
26                 p[u] = v;
27             }
28         }
29     }
30 }

```

5.3 Dijkstra

```

1 // Encontra o menor caminho de um évrtice s para todos os outros évrtices
   do grafo.
2 //Complexidade: O((V + E)logV)
3
4 int n;
5 vector<vector<pair<int, int>>> adj; // adj[a] = [{b, w}]
6 vector<int> dist, parent; /*dist[a] = dist(source -> a)*/
7 vector<bool> vis;
8
9 void dijkstra(int s) {
10
11     dist.resize(n+1, LINF-10);
12     vis.resize(n+1, false);
13     parent.resize(n+1, -1);
14     dist[s] = 0;
15
16     priority_queue<pair<int, int>> q;
17     q.push({0, s});
18
19     while (!q.empty()) {
20         int a = q.top().second; q.pop();
21
22         if (vis[a]) continue;
23         vis[a] = true;
24
25         for (auto [b, w] : adj[a]) {
26             if (dist[a] + w < dist[b]) {
27                 dist[b] = dist[a] + w;
28                 parent[b] = a;

```

```

29         q.push({-dist[b], b});
30     }
31 }
32 }
33 }
34 //Complexidade: O(V)
35 vector<int> restorePath(int v) {
36     vector<int> path;
37     for (int u = v; u != -1; u = parent[u])
38         path.push_back(u);
39     reverse(path.begin(), path.end());
40     return path;
41 }
42 }
43
44 void call() {
45
46     adj.resize(n); /*n = nodes*/
47
48     f(i,0,n) {
49         int a, b, w; cin >> a >> b >> w;
50         adj[a].push_back({b, w});
51         adj[b].push_back({a, w});
52     }
53
54     dijkstra(0);
55 }

```

5.4 Successor Graph

```

1 // Encontra sucessor de um vertice dentro de um grafo direcionado
2 // Pre calcular: O(nlogn)
3 // Consulta: O(logn)
4
5 vector<vector<int>> adj;
6
7 int succ(int x, int u) {
8     if(k == 1) return adj[x][0];
9     return succ(succ(x, k/2), k/2);
10 }

```

5.5 Kosaraju

```

1 // Kosaraju
2 //
3 // Complexidade: O(n + m)
4
5 int n;
6 vector<int> g[MAX], gi[MAX]; // grafo invertido
7 int vis[MAX], comp[MAX]; // componente de cada vértice
8 stack<int> S;
9
10 void dfs(int k) {
11     vis[k] = 1;
12     for (int i = 0; i < (int) g[k].size(); i++)
13         if (!vis[g[k][i]]) dfs(g[k][i]);

```

```

14
15     S.push(k);
16 }
17
18 // Descricao: Calcula as componentes fortemente conexas de um grafo
19 void scc(int k, int c) {
20     vis[k] = 1;
21     comp[k] = c; // componente de k eh c
22     for (int i = 0; i < (int) gi[k].size(); i++)
23         if (!vis[gi[k][i]]) scc(gi[k][i], c);
24 }
25
26 void kosaraju() {
27
28     memset(vis, 0, sizeof(vis));
29     for(int i=0; i<n; i++) if(!vis[i]) dfs(i);
30     memset(vis, 0, sizeof(vis));
31
32     while (S.size()) {
33         int u = S.top(); S.pop();
34         if (!vis[u]) scc(u, u);
35     }
36 }

```

5.6 Euler Tree

```

1 vector<vector<int>> adj(MAX);
2 vector<int> vis(MAX, 0);
3 vector<int> euTree(MAX);
4
5 // Eulerian path in tree
6 // Complexity: O(V)
7 void eulerTree(int u, int &index) {
8     vis[u] = 1;
9     euTree[index++] = u;
10    for (auto it : adj[u]) {
11        if (!vis[it]) {
12            eulerTree(it, index);
13            euTree[index++] = u;
14        }
15    }
16 }
17
18 int main() {
19
20     f(i,0,n-1) {
21         int a, b; cin >> a >> b;
22         adj[a].push_back(b);
23         adj[b].push_back(a);
24     }
25
26     int index = 0;
27     eulerTree(1, index);
28 }

```

5.7 Kruskal

```

1 // Kruskal
2 //
3 // Gera e retorna uma AGM e seu custo total a partir do vetor de arestas (
4 //   edg) do grafo
5 //
6 // Complexidade:  $O(E \log E)$  onde E eh o numero de arestas
7 //
8 //  $O(m \log(m) + m a(m)) = O(m \log(m))$ 
9
10 vector<int> id, sz;
11
12 int find(int a){ //  $O(a(N))$  amortizado
13     return id[a] = (id[a] == a ? a : find(id[a]));
14 }
15
16 void uni(int a, int b) { //  $O(a(N))$  amortizado
17     a = find(a), b = find(b);
18     if(a == b) return;
19
20     if(sz[a] > sz[b]) swap(a,b);
21     id[a] = b, sz[b] += sz[a];
22 }
23
24 pair<int, vector<tuple<int, int, int>>> kruskal(vector<tuple<int, int, int>
25     >>& edg) {
26
27     sort(edg.begin(), edg.end());
28
29     int cost = 0;
30     vector<tuple<int, int, int>> mst; // opcional
31     for (auto [w,x,y] : edg) if (find(x) != find(y)) {
32         mst.emplace_back(w, x, y); // opcional
33         cost += w;
34         uni(x,y);
35     }
36     return {cost, mst};
37 }
38
39 int main() {
40
41     int n/*nodes*/, ed/*edges*/;
42
43     id.resize(n); iota(all(id), 0);
44     sz.resize(n, -1);
45     vector<tuple<int, int, int>> edg;
46
47     f(i,0,ed) {
48         int a, b, w; cin >> a >> b >> w;
49         edg.push_back({w, a, b});
50     }
51
52     auto [cost, mst] = kruskal(edg);
53 }

```

5.8 Dfs

```

1 vector<int> adj[MAXN];
2 int visited[MAXN];
3
4
5
6 // DFS com informacoes adicionais sobre o pai de cada vertice
7 // Complexidade:  $O(V + E)$ , onde V eh o numero de vertices e E o numero de
8 // areqas
9 void dfs(int p) {
10     memset(visited, 0, sizeof visited);
11     stack<int> st;
12     st.push(p);
13
14     while (!st.empty()) {
15         int curr = st.top();
16         st.pop();
17         if (visited[curr]==1) continue;
18         visited[curr]=1;
19         // process current node here
20
21         for (auto i : adj[curr]) {
22             st.push(i);
23         }
24     }
25
26 // DFS com informacoes adicionais sobre o pai de cada vertice
27 // Complexidade:  $O(V + E)$ , onde V eh o numero de vertices e E o numero de
28 // areqas
29 void dfs(int v) {
30     visited[v] = true;
31     for (int u : adj[v]) {
32         if (!visited[u])
33             dfs(u);
34     }
35 }

```

6 Matematica

6.1 N Fibonacci

```

1 int dp[MAX];
2
3 int fibonacciDP(int n) {
4     if (n == 0) return 0;
5     if (n == 1) return 1;
6     if (dp[n] != -1) return dp[n];
7     return dp[n] = fibonacciDP(n-1) + fibonacciDP(n-2);
8 }
9
10 int nFibonacci(int minus, int times, int n) {
11     if (n == 0) return 0;
12     if (n == 1) return 1;
13     if (dp[n] != -1) return dp[n];
14     int aux = 0;
15     for(int i=0; i<times; i++) {
16         aux += nFibonacci(minus, times, n-minus);
17     }
18 }

```

```

17     }
18 }

```

6.2 Mdc Multiplo

```

1 // Description: Calcula o MDC de um vetor de inteiros.
2 // Complexidade:  $O(n \log n)$  onde n eh o tamanho do vetor
3 int mdc_many(vector<int> arr) {
4     int result = arr[0];
5
6     for (int& num : arr) {
7         result = mdc(num, result);
8
9         if(result == 1) return 1;
10    }
11    return result;
12 }

```

6.3 Factorial

```

1 unordered_map<int, int> memo;
2
3 // Factorial
4 // Complexidade:  $O(n)$ , onde n eh o numero a ser fatorado
5 int factorial(int n) {
6     if (n == 0 || n == 1) return 1;
7     if (memo.find(n) != memo.end()) return memo[n];
8     return memo[n] = n * factorial(n - 1);
9 }

```

6.4 Mmc Multiplo

```

1 // Description: Calcula o mmc de um vetor de inteiros.
2 // Complexidade:  $O(n \log n)$  onde n eh o tamanho do vetor
3 int mmc_many(vector<int> arr) {
4     int result = arr[0];
5
6     for(int& num : arr)
7         result = (num * result / mmc(num, result ));
8     return ans;
9 }

```

6.5 Fast Exponentiation

```

1 const int mod = 1e9+7;
2
3 // Fast Exponentiation: retorna  $a^b \% mod$ 
4 // Quando usar: quando precisar calcular  $a^b \% mod$ 
5 int fexp(int a, int b) {
6     int ans = 1;
7     while (b) {
8         if (b & 1)
9             ans = ans * a % mod;
10        a = a * a % mod;
11        b >>= 1;

```

```

12    }
13    return ans;
14 }

```

6.6 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
16            notPrime[j] = true; // como não primos
17    }
18 }

```

6.7 Mdc

```

1 // Description: Calcula o mdc de dois numeros inteiros.
2 // Complexidade:  $O(\log n)$  onde n eh o maior numero
3 int mdc(int x, int y) {
4     return y ? mdc(y, x % y) : abs(x);
5 }

```

6.8 Fatorial Grande

```

1 static BigInteger[] dp = new BigInteger[1000000];
2
3 public static BigInteger factorialDP(BigInteger n) {
4     dp[0] = BigInteger.ONE;
5     for (int i = 1; i <= n.intValue(); i++) {
6         dp[i] = dp[i - 1].multiply(BigInteger.valueOf(i));
7     }
8     return dp[n.intValue()];
9 }

```

6.9 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 vector<int> sieve(const int N) {
6
7     vector<int> lp(N + 1); // lp[i] = menor fator primo de i
8     vector<int> pr;
9
10    for (int i = 2; i <= N; ++i) {
11        if (lp[i] == 0) {
12            lp[i] = i;
13            pr.push_back(i);
14        }
15        for (int j = 0; i * pr[j] <= N; ++j) {
16            lp[i * pr[j]] = pr[j];
17            if (pr[j] == lp[i])
18                break;
19        }
20    }
21
22    return pr;
23 }

```

6.10 Primo

```

1 // Descricao: Funcao que verifica se um numero n eh primo.
2 // Complexidade: O(sqrt(n))
3 bool prime(int a) {
4     if (a == 1)
5         return false;
6     int lim = round(sqrt(a));
7     for (int i = 2; i <= lim; ++i)
8         if (a % i == 0)
9             return false;
10    return true;
11 }

```

6.11 Miller Rabin

```

1 // Teste de primalidade de Miller-Rabin
2 // Complexidade: O(k*log^3(n)), onde k eh o numero de testes e n eh o
   numero a ser testado
3 // Descricao: Testa se um numero eh primo com uma probabilidade de erro de
   1/4^k
4
5 int mul(int a, int b, int m) {
6     int ret = a*b - int((long double)1/m*a*b+0.5)*m;
7     return ret < 0 ? ret+m : ret;
8 }
9
10 int pow(int x, int y, int m) {
11     if (!y) return 1;
12     int ans = pow(mul(x, x, m), y/2, m);
13     return y%2 ? mul(x, ans, m) : ans;
14 }
15
16 bool prime(int n) {
17     if (n < 2) return 0;

```

```

18     if (n <= 3) return 1;
19     if (n % 2 == 0) return 0;
20     int r = __builtin_ctzint(n - 1), d = n >> r;
21
22     // com esses primos, o teste funciona garantido para n <= 2^64
23     // funciona para n <= 3*10^24 com os primos ate 41
24     for (int a : {2, 325, 9375, 28178, 450775, 9780504, 795265022}) {
25         int x = pow(a, d, n);
26         if (x == 1 or x == n - 1 or a % n == 0) continue;
27
28         for (int j = 0; j < r - 1; j++) {
29             x = mul(x, x, n);
30             if (x == n - 1) break;
31         }
32         if (x != n - 1) return 0;
33     }
34     return 1;
35 }

```

6.12 Numeros Grandes

```

1 public static void BbigInteger() {
2
3     BigInteger a = BigInteger.valueOf(1000000000);
4     a = new BigInteger("1000000000");
5
6     // çõperaes com inteiros grandes
7     BigInteger arit = a.add(a);
8     arit = a.subtract(a);
9     arit = a.multiply(a);
10    arit = a.divide(a);
11    arit = a.mod(a);
12
13    // çãComparao
14    boolean bool = a.equals(a);
15    bool = a.compareTo(a) > 0;
16    bool = a.compareTo(a) < 0;
17    bool = a.compareTo(a) >= 0;
18    bool = a.compareTo(a) <= 0;
19
20    // ãConverso para string
21    String m = a.toString();
22
23    // ãConverso para inteiro
24    int _int = a.intValue();
25    long _long = a.longValue();
26    double _doub = a.doubleValue();
27
28    // @Potncia
29    BigInteger _pot = a.pow(10);
30    BigInteger _sqr = a.sqrt();
31
32 }
33
34 public static void BigDecimal() {
35
36     BigDecimal a = new BigDecimal("1000000000");

```

```

37     a = new BigDecimal("1000000000.000000000");
38     a = BigDecimal.valueOf(1000000000, 10);
39
40 // çõOperaes com reais grandes
41 BigDecimal arit = a.add(a);
42     arit = a.subtract(a);
43     arit = a.multiply(a);
44     arit = a.divide(a);
45     arit = a.remainder(a);
46
47 // çãComparao
48 boolean bool = a.equals(a);
49     bool = a.compareTo(a) > 0;
50     bool = a.compareTo(a) < 0;
51     bool = a.compareTo(a) >= 0;
52     bool = a.compareTo(a) <= 0;
53
54 // ãConverso para string
55 String m = a.toString();
56
57 // ãConverso para inteiro
58 int _int = a.intValue();
59 long _long = a.longValue();
60 double _doub = a.doubleValue();
61
62 // @Potncia
63 BigDecimal _pot = a.pow(10);
64 }

```

6.13 Numeros Grandes

```

1 // Descricao: Implementacao de operacoes com numeros grandes
2 // Complexidade: O(n * m), n = tamanho do primeiro numero, m = tamanho do
   segundo numero
3
4 void normalize(vector<int>& num) {
5     int carry = 0;
6     for (int i = 0; i < num.size(); ++i) {
7         num[i] += carry;
8         carry = num[i] / 10;
9         num[i] %= 10;
10    }
11
12    while (carry > 0) {
13        num.push_back(carry % 10);
14        carry /= 10;
15    }
16 }
17
18
19 pair<int, vector<int>> bigSum(const pair<int, vector<int>>& a, const pair<71
   int, vector<int>>& b) {
20     if (a.first == b.first) {
21         vector<int> result(max(a.second.size(), b.second.size()), 0);
22         transform(a.second.begin(), a.second.end(), b.second.begin(),
           result.begin(), plus<int>());

```

```

23         normalize(result);
24         return {a.first, result};
25     } else {
26         vector<int> result(max(a.second.size(), b.second.size()), 0);
27         transform(a.second.begin(), a.second.end(), b.second.begin(),
           result.begin(), minus<int>());
28         normalize(result);
29         return {a.first, result};
30     }
31 }
32
33 pair<int, vector<int>> bigSub(const pair<int, vector<int>>& a, const pair<
   int, vector<int>>& b) {
34     return bigSum(a, {-b.first, b.second});
35 }
36
37 pair<int, vector<int>> bigMult(const pair<int, vector<int>>& a, const pair<
   <int, vector<int>>& b) {
38     vector<int> result(a.second.size() + b.second.size(), 0);
39
40     for (int i = 0; i < a.second.size(); ++i) {
41         for (int j = 0; j < b.second.size(); ++j) {
42             result[i + j] += a.second[i] * b.second[j];
43         }
44     }
45
46     normalize(result);
47     return {a.first * b.first, result};
48 }
49
50
51 void printNumber(const pair<int, vector<int>>& num) {
52     if (num.first == -1) {
53         cout << '-';
54     }
55
56     for (auto it = num.second.rbegin(); it != num.second.rend(); ++it) {
57         cout << *it;
58     }
59     cout << endl;
60 }
61
62 int main() {
63
64     pair<int, vector<int>> num1 = {1, {1, 2, 3}}; // Representing +321
65     pair<int, vector<int>> num2 = {-1, {4, 5, 6}}; // Representing -654
66
67     cout << "Sum: "; printNumber(bigSum(num1, num2));
68     cout << "Difference: "; printNumber(bigSub(num1, num2));
69     cout << "Product: "; printNumber(bigMult(num1, num2));
70 }

```

6.14 Mmc

```

1 // Description: Calcula o mmc de dois números inteiros.
2 // Complexidade: O(logn) onde n é o maior número

```

```

3 int mmc(int x, int y) {
4     return (x && y ? (return abs(x) / mdc(x, y) * abs(y)) : abs(x | y));
5 }

```

7 Strings

7.1 Ocorrencias

```

1 // Description: çãFunco que retorna um vetor com as çõposies de todas as
   êocorrncias de uma substring em uma string.
2 // Complexidade: O(n * m) onde n é o tamanho da string e m é o tamanho da
   substring.
3 vector<int> ocorrencias(string str,string sub){
4     vector<int> ret;
5     int index = str.find(sub);
6     while(index!=-1){
7         ret.push_back(index);
8         index = str.find(sub,index+1);
9     }
10
11     return ret;
12 }

```

7.2 Palindromo

```

1 // Descricao: Funcao que verifica se uma string eh um palindromo.
2 // Complexidade: O(n) onde n eh o tamanho da string.
3 bool isPalindrome(string str) {
4     for (int i = 0; i < str.length() / 2; i++) {
5         if (str[i] != str[str.length() - i - 1]) {
6             return false;
7         }
8     }
9     return true;
10 }

```

7.3 Split Cria

```

1 // Descricao: Funcao que divide uma string em um vetor de strings.
2 // Complexidade: O(n * m) onde n eh o tamanho da string e m eh o tamanho
   do delimitador.
3 vector<string> split(string s, string del = " ") {
4     vector<string> retorno;
5     int start, end = -1*del.size();
6     do {
7         start = end + del.size();
8         end = s.find(del, start);
9         retorno.push_back(s.substr(start, end - start));
10    } while (end != -1);
11    return retorno;
12 }

```

7.4 Remove Acento

```

1 // Descricao: Funcao que remove acentos de uma string.
2 // Complexidade: O(n * m) onde n eh o tamanho da string e m eh o tamanho
   do alfabeto com acento.
3 string removeAcento(string str) {
4
5     string comAcento = "áéíóúâêôãõà";
6     string semAcento = "aeiouaeoaoa";
7
8     for(int i = 0; i < str.size(); i++){
9         for(int j = 0; j < comAcento.size(); j++){
10             if(str[i] == comAcento[j]){
11                 str[i] = semAcento[j];
12                 break;
13             }
14         }
15     }
16
17     return str;
18 }

```

7.5 Chaves Colchetes Parenteses

```

1 # Descricao: Funcao que verifica se uma string contendo chaves, colchetes
   e parenteses esta balanceada.
2 def balanced(string) -> bool:
3     stack = []
4
5     for i in string:
6         if i in '([{': stack.append(i)
7
8         elif i in ')]}':
9             if (not stack) or ((stack[-1],i) not in [(('(',')'), (('[',']')),
10                (('{','}')'))):
11                 return False
12             else:
13                 stack.pop()
14
15     return not stack

```

7.6 Lower Iupper

```

1 // Description: çãFunco que transforma uma string em lowercase.
2 // Complexidade: O(n) onde n é o tamanho da string.
3 string to_lower(string a) {
4     for (int i=0;i<(int)a.size();++i)
5         if (a[i]>='A' && a[i]<='Z')
6             a[i]+='a'-'A';
7     return a;
8 }
9
10 // para checar se é lowercase: islower(c);
11
12 // Description: çãFunco que transforma uma string em uppercase.
13 // Complexidade: O(n) onde n é o tamanho da string.
14 string to_upper(string a) {
15     for (int i=0;i<(int)a.size();++i)

```

```

16         if (a[i]>='a' && a[i]<='z')
17             a[i]-='a'-'A';
18     return a;
19 }
20
21 // para checar se e uppercase: isupper(c);

```

7.7 Lexicograficamente Minima

```

1 // Descricao: Retorna a menor rotacao lexicografica de uma string.
2 // Complexidade: O(n * log(n)) onde n eh o tamanho da string
3 string minLexRotation(string str) {
4     int n = str.length();
5
6     string arr[n], concat = str + str;
7
8     for (int i = 0; i < n; i++)
9         arr[i] = concat.substr(i, n);
10
11     sort(arr, arr+n);
12
13     return arr[0];
14 }

```

7.8 Numeros E Char

```

1 char num_to_char(int num) { // 0 -> '0'
2     return num + '0';
3 }
4
5 int char_to_num(char c) { // '0' -> 0
6     return c - '0';
7 }
8
9 char int_to_ascii(int num) { // 97 -> 'a'
10    return num;
11 }
12
13 int ascii_to_int(char c) { // 'a' -> 97
14    return c;
15 }

```

8 Vector

8.1 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 }

```

8.2 Troco

```

1 vector<int> troco(vector<int> coins, int n) {
2     int first[n];
3     value[0] = 0;
4     for(int x=1; x<=n; x++) {
5         value[x] = INF;
6         for(auto c : coins) {
7             if(x-c >= 0 and value[x-c] + 1 < value[x]) {
8                 value[x] = value[x-c]+1;
9                 first[x] = c;
10            }
11        }
12    }
13
14    vector<int> ans;
15    while(n>0) {
16        ans.push_back(first[n]);
17        n -= first[n];
18    }
19    return ans;
20 }

```

8.3 Contar Soma Subarray Igual K

```

1 // Descricao: Conta quantos subarrays de um vetor tem soma igual a k
2 // Complexidade: O(n)
3 int contarSomaSubarray(vector<int>& v, int k) {
4     unordered_map<int, int> prevSum; // map to store the previous sum
5
6     int ret = 0, currentSum = 0;
7
8     for(int& num : v) {
9         currentSum += num;
10
11         if (currentSum == k) ret++; /// Se a soma atual for igual a k,
            encontramos um subarray
12
13         if (prevSum.find(currentSum - k) != prevSum.end()) // se subarray
            com soma (currentSum - k) existir, sabe que [0:n] eh um subarray com
            soma k
14             ret += (prevSum[currentSum - k]);
15
16         prevSum[currentSum]++;
17     }
18
19     return ret;
20 }

```

8.4 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 }

```

8.5 Maior Sequencia Subsequente

```

1 // Maior sequencia subsequente
2 // {6, 2, 5, 1, 7, 4, 8, 3} => {2, 5, 7, 8}
3
4 int maiorCrescente(vector<int> v) {

```

```

5     vector<int> lenght(v.size());
6     for(int k=0; k<v.size(); k++) {
7         lenght[k] = ;
8         for(int i=0; i<k; i++) {
9             if(v[i] < v[k]) {
10                 lenght[i] = max(lenght[k], lenght[i]+1)
11             }
12         }
13     }
14     return lenght.back();
15 }

```

9 Outros

9.1 Binario

```

1 // Descricao: conversao de decimal para binario
2 // Complexidade: O(logn) onde n eh o numero decimal
3 string decimal_to_binary(int dec) {
4     string binary = "";
5     while (dec > 0) {
6         int bit = dec % 2;
7         binary = to_string(bit) + binary;
8         dec /= 2;
9     }
10    return binary;
11 }
12
13 // Descricao: conversao de binario para decimal
14 // Complexidade: O(logn) onde n eh o numero binario
15 int binary_to_decimal(string binary) {
16     int dec = 0;
17     int power = 0;
18     for (int i = binary.length() - 1; i >= 0; i--) {
19         int bit = binary[i] - '0';
20         dec += bit * pow(2, power);
21         power++;
22     }
23     return dec;
24 }

```

9.2 Horario

```

1 // Descricao: Funcoes para converter entre horas e segundos.
2 // Complexidade: O(1)
3 int cts(int h, int m, int s) {
4     int total = (h * 3600) + (m * 60) + s;
5     return total;
6 }
7
8 tuple<int, int, int> cth(int total_seconds) {
9     int h = total_seconds / 3600;
10    int m = (total_seconds % 3600) / 60;
11    int s = total_seconds % 60;
12    return make_tuple(h, m, s);
13 }

```

9.3 Max Subarray Sum

```
1 // Maximum Subarray Sum
2 // Descricao: Retorna a soma maxima de um subarray de um vetor.
3 // Complexidade:  $O(n)$  onde n eh o tamanho do vetor
4 int maxSubarraySum(vector<int> x) {
5     int best = 0, sum = 0;
6     for (int k = 0; k < n; k++) {
7         sum = max(x[k], sum+x[k]);
8         best = max(best, sum);
9     }
10    return best;
11 }
```

9.4 Binary Search

```
1 // Description: çImplementao do algoritmo de busca ábinria.
2 // Complexidade:  $O(\log n)$  onde n eh o tamanho do vetor
3 int BinarySearch(<vector>int arr, int x){
4     int k = 0;
5     int n = arr.size();
6
7     for (int b = n/2; b >= 1; b /= 2) {
8         while (k+b < n && arr[k+b] <= x) k += b;
9     }
10    if (arr[k] == x) {
```

```
11        return k;
12    }
13 }
```

9.5 Fibonacci

```
1 // Descricao: Funcao que retorna o n-esimo termo da sequencia de Fibonacci
2 // Complexidade:  $O(2^n)$  onde n eh o termo desejado
3 int fib(int n){
4     if(n <= 1){
5         return n;
6     }
7     return fib(n - 1) + fib(n - 2);
8 }
9
10 vector<int> memo(MAX, -1);
11
12 // Descricao: Funcao que retorna o n-esimo termo da sequencia de Fibonacci
13 // Complexidade:  $O(n)$  onde n eh o termo desejado
14 int fibPD(int n) {
15     if (n <= 1) return n;
16     if (memo[n] != -1) return memo[n];
17     return memo[n] = fibPD(n - 1) + fibPD(n - 2);
18 }
```