



Pedro Augusto Ulisses Andrade Lucas Andrade

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

Contents

1	Utils	2			
1.1	Template Python	2	3.8	Combinacao Com Repeticao	7
1.2	Limites	2	3.9	Permutacao Circular	7
1.3	Makefile	3	4	Estruturas	7
1.4	Files	3	4.1	Segmen Tree	7
1.5	Mini Template Cpp	3	4.2	Sparse Table Disjunta	8
1.6	Template Cpp	3	4.3	Bittree	8
2	Informações	4	4.4	Seg Tree	8
2.1	Bitmask	4	4.5	Union Find	9
2.2	Priority Queue	4	4.6	Fenwick Tree	9
2.3	Sort	5	5	Grafos	11
2.4	Set	5	5.1	Encontrar Ciclo	11
2.5	Vector	5	5.2	Topological Kahn	11
2.6	String	6	5.3	Articulation	12
3	Combinatoria	6	5.4	Bipartido	12
3.1	Arranjo Com Repeticao	6	5.5	Bfs Nivelado	12
3.2	@ Factorial	6	5.6	Dfs	13
3.3	Permutacao Com Repeticao	6	5.7	Successor Graph	13
3.4	@ Tabela	6	5.8	Cycle Check	13
3.5	Permutacao Simples	6	5.9	Dijkstra	14
3.6	Combinacao Simples	7	5.10	Labirinto	14
3.7	Arranjo Simples	7	5.11	Bfs	15
			5.12	Bfs Matriz	15
			5.13	Floyd Warshall	16
			5.14	Pontos Articulacao	16

5.15	Graham Scan(elastico)	17	9	Vector	27
5.16	Kruskal	17	9.1	Maior Retangulo Em Histograma	27
5.17	Kosaraju	18	9.2	Elemento Mais Frequente	27
5.18	Euler Tree	19	9.3	Subset Sum	27
6	Matematica	19	9.4	Contar Subarrays Somam K	28
6.1	Numeros Grandes	19	9.5	Maior Triangulo Em Histograma	28
6.2	Mmc	20	9.6	K Maior Elemento	28
6.3	Fatoracao	20	9.7	Remove Repetitive	29
6.4	Fatorial Grande	20	9.8	Troco	29
6.5	Mmc Multiplo	20	9.9	Maior Sequencia Subsequente	29
6.6	Contar Quanti Solucoes Eq 2 Variaveis	20	9.10	Maior Subsequência Crescente	29
6.7	Mdc	20	9.11	Maior Subsequencia Comum	30
6.8	Decimal Para Fracao	20	9.12	Soma Maxima Sequencial	30
6.9	Mdc Multiplo	20	10	Outros	30
6.10	Tabela Verdade	21	10.1	Intervalos	30
6.11	Factorial	21	10.2	Dp	30
6.12	N Fibonacci	21	10.3	Binary Search	31
6.13	Divisores	21	10.4	Mochila	31
6.14	Sieve Linear	21	10.5	Horario	31
6.15	Sieve	22	10.6	Fibonacci	31
6.16	Conversao De Bases	22	10.7	Binario	32
6.17	Numeros Grandes	22	10.8	Max Subarray Sum	32
6.18	Miller Rabin	23			
6.19	Primo	23			
6.20	Dois Primos Somam Num	23			
6.21	Fast Exponentiation	23			
7	Matriz	24			
7.1	Maior Retangulo Binario Em Matriz	24			
7.2	Max 2d Range Sum	24			
8	Strings	25			
8.1	Palindromo	25			
8.2	Ocorrencias	25			
8.3	Chaves Colchetes Parenteses	25			
8.4	Permutacao	25			
8.5	Lower Upper	25			
8.6	Numeros E Char	25			
8.7	Split Cria	26			
8.8	Infixo Para Posfixo	26			
8.9	Remove Acento	26			
8.10	Lexicograficamente Minima	26			
8.11	Calculadora Posfixo	26			

1 Utils

1.1 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             # ãl 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 éat 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
82     de inputs
83         lista = parsin(l=2, vpl=5)
84         solve()
85
86     sys.stderr.write(f"Run Time : {(clk() - st):.6f} seconds\n")
87
88 main()
```

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

```

```

30 // ESTOURAR TEMPO

```

```

32 input size      | complexidade para 1 s
33 -----+-----
34 [10,11]          | 0(n!), 0(n^6)
35 [17,19]          | 0(2^n * n^2)
36 [18, 22]         | 0(2^n * n)
37 [24,26]          | 0(2^n)
38 ... 100          | 0(n^4)
39 ... 450          | 0(n^3)
40 ... 1500         | 0(n^2.5)
41 ... 2500         | 0(n^2 * log n)
42 ... 10^4         | 0(n^2)
43 ... 2*10^5       | 0(n^1.5)
44 ... 4.5*10^6     | 0(n log n)
45 ... 10^7         | 0(n log log n)
46 ... 10^8         | 0(n), 0(log n), 0(1)

```

```

49 // FATORIAL

```

```

51 12! = 479.001.600 [limite do (u)int]
52 20! = 2.432.902.008.176.640.000 [limite do (u)int64_t]

```

1.3 Makefile

```

1 CXX = g++
2 CPXXFLAGS = -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 q:
5     cp temp.cpp $(f).cpp
6     touch $(f).txt
7     code $(f).txt
8     code $(f).cpp
9     clear
10 compile:
11     g++ -g $(f).cpp $(CXXFLAGS) -o $(f)
12 exe:
13     ./$(f) < $(f).txt
14
15 runc: compile
16 runci: compile exe
17
18 clearexe:

```

```

19 find . -maxdepth 1 -type f -executable -exec rm {} +
20 cleartxt:
21     find . -type f -name "*.txt" -exec rm -f {} \;
22 clear: clearexe cleartxt
23 clear

```

1.4 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

```

1.5 Mini Template Cpp

```

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

```

1.6 Template Cpp

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define _ std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
5 #define all(a) a.begin(), a.end()
6 #define int long long int
7 #define double long double
8 #define vi vector<int>

```

```

9 #define pii          pair<int,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.first<<" "<<x.second<< 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
16 #define TRUNC(x, n)  floor(x * pow(10, n))/pow(10, n)
17 #define ROUND(x, n)  round(x * pow(10, n))/pow(10, n)
18 #define dbg(x) cout << #x << " = " << x << " ";
19 #define dbg1(x) cout << #x << " = " << x << endl;
20
21 const int INF = 1e9;      // 2^31-1
22 const int LLINF = 4e18;  // 2^63-1
23 const double EPS = 1e-9;
24 const int MAX = 1e6+10;  // 10^6 + 10
25
26 void solve() {
27
28 }
29
30
31 int32_t main() { _
32
33     clock_t z = clock();
34     int t = 1; // cin >> t;
35     while (t--) {
36         solve();
37     }
38     cerr << fixed << "Run Time : " << ((double)(clock() - z) /
39     CLOCKS_PER_SEC) << endl;
40     return 0;
41 }

```

2 Informações

2.1 Bitmask

```

1 int n = 11, ans = 0, k = 3;
2
3 // Operacoes com bits
4 ans = n & k; // AND bit a bit
5 ans = n | k; // OR bit a bit
6 ans = n ^ k; // XOR bit a bit
7 ans = ~n;    // NOT bit a bit
8
9 // Operacoes com 2^k em O(1)
10 ans = n << k; // ans = n * 2^k
11 ans = n >> k; // ans = n / 2^k
12
13 int j;
14
15 // Ativa j-esimo bit (0-based)
16 ans |= (1<<j);
17
18 // Desativa j-esimo bit (0-based)

```

```

19 ans &= (1<<j);
20
21 // Inverte j-esimo bit (0-based)
22 ans ^= (1<<j);
23
24 // checar se j-esimo bit esta ativo (0-based)
25 ans = n & (1<<j);
26
27 // Pegar valor do bit menos significativo | Retorna o maior divisor
28 ans = n & -n;
29
30 // Ligar todos on n bits
31 ans = (1<<n) - 1;
32
33 // Contar quantos 1's tem no binario de n
34 ans = __builtin_popcount(n);
35
36 // Contar quantos 0's tem no final do binario de n
37 ans = __builtin_ctz(n);

```

2.2 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);

```

2.3 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,
2, 1 }
12 auto cmp = [](int a, int b) { return a < b; }; // { 2, 3, 1 } -> { 1,
2, 3 }
13 sort(v.begin(), v.end(), cmp);
14 sort(all(v), cmp);
15
16 // Sort por uma çãfuno (çãcomparao de pares):
17 auto cmp = [](pair<int, int> a, pair<int, int> b) { return a.second >
b.second; };
18
19 // Sort parcial:
20 partial_sort(v.begin(), v.begin() + n, v.end()); // sorta com n menos
elementos
21 partial_sort(v.rbegin(), v.rbegin() + n, s.rend()) // sorta com n
maiores elementos
22
23 // SORT VS SET
24 * para um input com elementos distintos, sort é mais árpido que set

```

2.4 Set

```

1 set<int> st;
2
3 // Complexidade: O(log(n))
4 st.insert(x);
5 st.erase(x);
6 st.find(x);
7 st.erase(st.find(x));
8
9
10 // Complexidade: O(1)
11 st.size();
12 st.empty();
13

```

```

14 // Complexidade: O(n)
15 st.clear();
16 for (auto x : st) {}
17
18 | priority_queue | set |
19 -----+-----+-----+-----+-----+-----+
20 op | call | compl | call | compl | melhor
21 -----+-----+-----+-----+-----+-----+
22 insert | push | log(n) | insert | log(n) | pq
23 erase_menor | pop | log(n) | erase | log(n) | pq
24 get_menor | top | 1 | begin | 1 | set
25 get_maior | - | - | rbegin | 1 | set
26 erase_number | remove | n | erase | log(n) | set
27 find_number | - | - | find | log(n) | set
28 find_>= | - | - | lower | log(n) | set
29 find_<= | - | - | upper | log(n) | set
30 iterate | for | n | for | n | set
31 -----+-----+-----+-----+-----+-----+

```

2.5 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: úmnero de âocorrncias
35 count(v.begin(), v.end(), x);

```

2.6 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 Com Repeticao

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

3.2 @ Factorial

```
1 // Calcula o fatorial de um únmero n
2 // Complexidade: O(n)
3
4 int factdp[20];
5
6 int fact(int n) {
7     if (n < 2) return 1;
8     if (factdp[n] != 0) return factdp[n];
9     return factdp[n] = n * fact(n - 1);
10 }
```

3.3 Permutacao Com Repeticao

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

3.4 @ 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
```

3.5 Permutacao Simples

```
1 // Agrupamentos distintos entre si pela ordem (FILA)
2 // Diferença do arranjo: usa todos os elementos para o calculo
3 // SEM repeticao
```

```
4
5 int permutacaoSimples(int n) {
6     return fact(n);
7 }
```

3.6 Combinacao Simples

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

3.7 Arranjo Simples

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

3.8 Combinacao Com Repeticao

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

3.9 Permutacao Circular

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

4 Estruturas

4.1 Segmen Tree

```
1 // Segment Tree with Lazy Propagation
2 // Update Range: O(log(n))
3 // Query Range: O(log(n))
4 // Memory: O(n)
5 // Build: O(n)
6
7 typedef vector<int> vi;
8
9 class SegmentTree {
10     private:
11         int n;
12         vi A, st, lazy;
13         int defaultVar; // min: INT_MIN | max: INT_MAX | sum: 0 | multiply
14         : 1
15
16         int l(int p) { return p<<1; }
17         int r(int p) { return (p<<1)+1; }
18
19         int conquer(int a, int b) {
```

```
19         if(a == defaultVar) return b;
20         if(b == defaultVar) return a;
21         return min(a, b);
22     }
23
24     void build(int p, int L, int R) {
25         if (L == R) st[p] = A[L];
26         else {
27             int m = (L+R)/2;
28             build(l(p), L, m);
29             build(r(p), m+1, R);
30             st[p] = conquer(st[l(p)], st[r(p)]);
31         }
32     }
33
34     void propagate(int p, int L, int R) {
35         if (lazy[p] != defaultVar) {
36             st[p] = lazy[p];
37             if (L != R) lazy[l(p)] = lazy[r(p)] = lazy[p];
38             else A[L] = lazy[p];
39             lazy[p] = defaultVar;
40         }
41     }
42
43     int query(int p, int L, int R, int i, int j) {
44         propagate(p, L, R);
45         if (i > j) return defaultVar;
46         if ((L >= i) && (R <= j)) return st[p];
47         int m = (L+R)/2;
48         return conquer(query(l(p), L, m, i, min(m, j)),
49                         query(r(p), m+1, R, max(i, m+1), j));
50     }
51
52     void update(int p, int L, int R, int i, int j, int val) {
53         propagate(p, L, R);
54         if (i > j) return;
55         if ((L >= i) && (R <= j)) {
56             lazy[p] = val;
57             propagate(p, L, R);
58         }
59         else {
60             int m = (L+R)/2;
61             update(l(p), L, m, i, min(m, j), val);
62             update(r(p), m+1, R, max(i, m+1), j, val);
63             int lsubtree = (lazy[l(p)] != defaultVar) ? lazy[l(p)] :
64             st[l(p)];
65             int rsubtree = (lazy[r(p)] != defaultVar) ? lazy[r(p)] :
66             st[r(p)];
67             st[p] = conquer(lsubtree, rsubtree);
68         }
69     }
70
71     public:
72         SegmentTree(int sz, int defaultVal) : n(sz), A(n), st(4*n), lazy
73         (4*n, defaultVal), defaultVar(defaultVal) {}
74
75         // vetor referencia, valor default (min: INT_MIN | max: INT_MAX |
```



```

73     sum: 0 | multiply: 1)
74     SegmentTree(const vi &initialA, int defaultVal) : SegmentTree((int)
75     )initialA.size(), defaultVal) {
76         A = initialA;
77         build(1, 0, n-1);
78
79         // A[i..j] = val | 0 <= i <= j < n | O(log(n))
80         void update(int i, int j, int val) { update(1, 0, n-1, i, j, val); }
81
82         // max(A[i..j]) | 0 <= i <= j < n | O(log(n))
83         int query(int i, int j) { return query(1, 0, n-1, i, j); }
84
85 void solve() {
86     vi A = {18, 17, 13, 19, 15, 11, 20, 99}; // make n a power of 2
87     int defaultVar = INT_MIN; // default value for max query
88     SegmentTree st(A, defaultVar);
89     int i = 1, j = 3;
90     int ans = st.query(i, j);
91     int newVal = 77;
92     st.update(i, j, newVal);
93     ans = st.query(i, j);
94 }

```

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 SparseTable {
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     int query(int l, int r) {
28         if (l == r) return v[l];

```

```

29         int j = __builtin_clz(1) - __builtin_clz(1^r);
30         return op(m[j][l], m[j][r]);
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
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

4.3 Bittree

```

1 /*      n --> No. of elements present in input array.
2     BITree[0..n] --> Array that represents Binary Indexed Tree.
3     arr[0..n-1] --> Input array for which prefix sum is evaluated. */
4
5 // Returns sum of arr[0..index]. This function assumes
6 // that the array is preprocessed and partial sums of
7 // array elements are stored in BITree[].
8 int getSum(vector<int>& BITree, int index) {
9     int sum = 0;
10    index = index + 1;
11    while (index>0) {
12        sum += BITree[index];
13        index -= index & (-index);
14    }
15    return sum;
16 }
17
18 void updateBIT(vector<int>& BITree, int n, int index, int val) {
19     index = index + 1;
20
21     while (index <= n) {
22         BITree[index] += val;
23         index += index & (-index);
24     }
25 }
26
27 vector<int> constructBITree(vector<int>& arr, int n) {
28     vector<int> BITree(n+1, 0);
29
30     for (int i=0; i<n; i++)
31         updateBIT(BITree, n, i, arr[i]);
32
33     return BITree;
34 }
35
36 void solve() {
37     vector<int> freq = {2, 1, 1, 3, 2, 3, 4, 5, 6, 7, 8, 9};
38     int n = freq.size();
39     vector<int> BITree = constructBITree(freq, n);
40     cout << "Sum of elements in arr[0..5] is" << getSum(BITree, 5);
41     // Let use test the update operation
42     freq[3] += 6;
43     updateBIT(BITree, n, 3, 6); //Update BIT for above change in arr[]
44
45     cout << "\nSum of elements in arr[0..5] after update is "
46         << getSum(BITree, 5);
47 }

```

4.4 Seg Tree

```

1 // Query: soma do range [a, b]
2 // Update: soma x em cada elemento do range [a, b]
3 //
4 // Complexidades:
5 // build - O(n)
6 // query - O(log(n))
7 // update - O(log(n))
8 namespace SegTree {
9
10     int seg[4*MAX];
11     int n, *v;
12
13     int op(int a, int b) { return a + b; }
14
15     int build(int p=1, int l=0, int r=n-1) {
16         if (l == r) return seg[p] = v[l];
17         int m = (l+r)/2;
18         return seg[p] = op(build(2*p, l, m), build(2*p+1, m+1, r));
19     }
20
21     void build(int n2, int* v2) {
22         n = n2, v = v2;
23         build();
24     }
25
26     int query(int a, int b, int p=1, int l=0, int r=n-1) {
27         if (a <= l and r <= b) return seg[p];
28         if (b < l or r < a) return 0;
29         int m = (l+r)/2;
30         return op(query(a, b, 2*p, l, m), query(a, b, 2*p+1, m+1, r));
31     }
32
33     int update(int a, int b, int x, int p=1, int l=0, int r=n-1) {
34         if (a <= l and r <= b) return seg[p];
35         if (b < l or r < a) return seg[p];
36         int m = (l+r)/2;
37         return seg[p] = op(update(a, b, x, 2*p, l, m), update(a, b, x, 2*p
38 +1, m+1, r));
39     }

```

4.5 Union Find

```

1 // Description: Union-Find (Disjoint Set Union)
2
3 typedef vector<int> vi;
4
5 struct UnionFind {
6     vi p, rank, setSize;
7     int numSets;
8     UnionFind(int N) {
9         p.assign(N, 0);
10         for (int i = 0; i < N; ++i)
11             p[i] = i;
12         rank.assign(N, 0);
13         setSize.assign(N, 1);
14         numSets = N;

```

```

15     }
16
17     // Retorna o numero de sets disjuntos (separados)
18     int numDisjointSets() { return numSets; }
19     // Retorna o tamanho do set que contem o elemento i
20     int sizeOfSet(int i) { return setSize[find(i)]; }
21
22     int find(int i) { return (p[i] == i) ? i : (p[i] = find(p[i])); }
23     bool same(int i, int j) { return find(i) == find(j); }
24     void uni(int i, int j) {
25         if (same(i, j))
26             return;
27         int x = find(i), y = find(j);
28         if (rank[x] > rank[y])
29             swap(x, y);
30         p[x] = y;
31         if (rank[x] == rank[y])
32             ++rank[y];
33         setSize[y] += setSize[x];
34         --numSets;
35     }
36 };
37
38 void solve() {
39     int n; cin >> n;
40     UnionFind UF(n);
41     UF.uni(0, 1);
42 }

```

4.6 Fenwick Tree

```

1 #define LSOne(S) ((S) & -(S)) // the key operation
2
3 class FenwickTree { // index 0 is not used
4     private:
5         vi ft;
6
7         void build(const vi &f) {
8             int m = (int)f.size() - 1; // note f[0] is always 0
9             ft.assign(m + 1, 0);
10             for (int i = 1; i <= m; ++i) {
11                 ft[i] += f[i];
12                 if (i + LSOne(i) <= m)
13                     ft[i + LSOne(i)] += ft[i];
14             }
15         }
16
17     public:
18         // empty FT
19         FenwickTree(int m) { ft.assign(m + 1, 0); }
20
21         // FT based on f
22         FenwickTree(const vi &f) { build(f); }
23
24         // FT based on s, and m = max(s)
25         FenwickTree(int m, const vi &s) {
26             vi f(m + 1, 0);

```

```

27     for (int i = 0; i < (int)s.size(); ++i)
28         ++f[s[i]];
29     build(f);
30 }
31
32 // RSQ(1, j)
33 int rsq(int j) {
34     int sum = 0;
35     for (; j; j -= LSOne(j))
36         sum += ft[j];
37     return sum;
38 }
39
40 // RSQ(i, j)
41 int rsq(int i, int j) { return rsq(j) - rsq(i - 1); }
42
43 // v[i] += v
44 void update(int i, int v) {
45     for (; i < (int)ft.size(); i += LSOne(i))
46         ft[i] += v;
47 }
48
49 // n-th element >= k
50 int select(int k) {
51     int p = 1;
52     while (p * 2 < (int)ft.size())
53         p *= 2;
54     int i = 0;
55     while (p) {
56         if (k > ft[i + p]) {
57             k -= ft[i + p];
58             i += p;
59         }
60         p /= 2;
61     }
62     return i + 1;
63 }
64 };
65
66 // Range Update Point Query
67 class RUPQ {
68     private:
69         FenwickTree ft;
70     public:
71
72         // empty FT
73         RUPQ(int m) : ft(FenwickTree(m)) {}
74
75         // v[ui,...,uj] += v
76         void range_update(int ui, int uj, int v) {
77             ft.update(ui, v);
78             ft.update(uj + 1, -v);
79         }
80
81         // rsq(i) = v[1] + v[2] + ... + v[i]
82         int point_query(int i) { return ft.rsq(i); }
83 };
84
85 // Range Update Range Query
86 class RURQ {
87     private:
88         RUPQ rupq;
89         FenwickTree purq;
90     public:
91         // empty structures
92         RURQ(int m) : rupq(RUPQ(m)), purq(FenwickTree(m)) {}
93
94         // v[ui,...,uj] += v
95         void range_update(int ui, int uj, int v) {
96             rupq.range_update(ui, uj, v);
97             purq.update(ui, v * (ui - 1));
98             purq.update(uj + 1, -v * uj);
99         }
100
101         // rsq(j) = v[1]*j - (v[1] + ... + v[j])
102         int rsq(int j) {
103             return rupq.point_query(j) * j -
104                 purq.rsq(j);
105         }
106
107         // rsq(i, j) = rsq(j) - rsq(i - 1)
108         int rsq(int i, int j) { return rsq(j) - rsq(i - 1); }
109 };
110
111 int32_t main() {
112
113     vi f = {0, 0, 1, 0, 1, 2, 3, 2, 1, 1, 0}; // index 0 is always 0
114     FenwickTree ft(f);
115     printf("%lli\n", ft.rsq(1, 6)); // 7 => ft[6]+ft[4] = 5+2 = 7
116     printf("%lld\n", ft.select(7)); // index 6, rsq(1, 6) == 7, which
117                                     // is >= 7
118     ft.update(5, 1); // update demo
119     printf("%lli\n", ft.rsq(1, 10)); // now 12
120     printf("====\n");
121     RUPQ rupq(10);
122     RURQ rurq(10);
123     rupq.range_update(2, 9, 7); // indices in [2, 3, ..., 9] updated by +7
124     rurq.range_update(2, 9, 7); // same as rupq above
125     rupq.range_update(6, 7, 3); // indices 6&7 are further updated by +3
126     rurq.range_update(6, 7, 3); // same as rupq above
127     // idx = 0 (unused) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
128     // val = -         | 0 | 7 | 7 | 7 | 7 | 10 | 10 | 7 | 7 | 0
129     for (int i = 1; i <= 10; i++)
130         printf("%lld -> %lli\n", i, rupq.point_query(i));
131     printf("RSQ(1, 10) = %lli\n", rurq.rsq(1, 10)); // 62
132     printf("RSQ(6, 7) = %lli\n", rurq.rsq(6, 7)); // 20
133     return 0;
134 }

```

5 Grafos

5.1 Encontrar Ciclo

```
1 // Description: Encontrar ciclo em grafo nao direcionado
2 // Complexidade: O(n + m)
3
4 int n;
5 vector<vector<int>> adj;
6 vector<bool> vis;
7 vector<int> p;
8 int cycle_start, cycle_end;
9
10 bool dfs(int v, int par) {
11     vis[v] = true;
12     for (int u : adj[v]) {
13         if(u == par) continue;
14         if(vis[u]) {
15             cycle_end = v;
16             cycle_start = u;
17             return true;
18         }
19         p[u] = v;
20         if(dfs(u, p[u]))
21             return true;
22     }
23     return false;
24 }
25
26 vector<int> find_cycle() {
27     cycle_start = -1;
28
29     for (int v = 0; v < n; v++)
30         if (!vis[v] and dfs(v, p[v]))
31             break;
32
33     if (cycle_start == -1) return {};
34
35     vector<int> cycle;
36     cycle.push_back(cycle_start);
37     for (int v = cycle_end; v != cycle_start; v = p[v])
38         cycle.push_back(v);
39     cycle.push_back(cycle_start);
40     return cycle;
41 }
42
43 void solve() {
44     int edg; cin >> n >> edg;
45     adj.assign(n, vector<int>());
46     vis.assign(n, false), p.assign(n, -1);
47     while(edg--) {
48         int a, b; cin >> a >> b;
49         adj[a].push_back(b);
50         adj[b].push_back(a);
51     }
52     vector<int> ans = find_cycle();
```

53 }

5.2 Topological Kahn

```
1 // Description: Ordenamento topologico usando o algoritmo de Kahn.
2 // Complexidade: O(V+E)
3 vector<vector<int>> adj;
4
5 vector<int> topologicalSort(int V) {
6
7     vector<int> indegree(V);
8     for (int i = 0; i < V; i++) {
9         for (auto it : adj[i]) {
10             indegree[it]++;
11         }
12     }
13
14     queue<int> q;
15     for (int i = 0; i < V; i++) {
16         if (indegree[i] == 0) {
17             q.push(i);
18         }
19     }
20     vector<int> result;
21     while (!q.empty()) {
22
23         int node = q.front(); q.pop();
24         result.push_back(node);
25
26         for (auto it : adj[node]) {
27             indegree[it]--;
28             if (indegree[it] == 0)
29                 q.push(it);
30         }
31     }
32
33     if (result.size() != V) {
34         cout << "Graph contains cycle!" << endl;
35         return {};
36     }
37
38     return result;
39 }
40
41 void solve() {
42
43     int n = 4; adj.resize(n);
44     vector<pair<int, int>> edges = { { 0, 1 }, { 1, 2 }, { 3, 1 }, { 3, 2 } };
45     for (auto& [a,b] : edges) {
46         adj[a].push_back(b);
47     }
48
49     vector<int> ans = topologicalSort(n);
50 }
51
52 int main() {
```

```

53     solve();
54 }

```

5.3 Articulation

```

1 // Description: Encontra pontos de articulacao e pontes em um grafo nao
  direcionado
2 // Complexidade: O(V + E)
3
4 vector<vector<pii>> adj;
5 vi dfs_num, dfs_low, dfs_parent, articulation_vertex;
6 int dfsNumberCounter, dfsRoot, rootChildren;
7 vector<pii> bridgesAns;
8
9 void articulationPointAndBridgeUtil(int u) {
10
11     dfs_low[u] = dfs_num[u] = dfsNumberCounter++;
12     for (auto &[v, w] : adj[u]) {
13         if (dfs_num[v] == -1) {
14             dfs_parent[v] = u;
15             if (u == dfsRoot) ++rootChildren;
16
17             articulationPointAndBridgeUtil(v);
18
19             if (dfs_low[v] >= dfs_num[u])
20                 articulation_vertex[u] = 1;
21             if (dfs_low[v] > dfs_num[u])
22                 bridgesAns.push_back({u, v});
23             dfs_low[u] = min(dfs_low[u], dfs_low[v]);
24         }
25         else if (v != dfs_parent[u])
26             dfs_low[u] = min(dfs_low[u], dfs_low[v]);
27     }
28 }
29
30 void articulationPointAndBridge(int n) {
31     dfsNumberCounter = 0;
32     f(u,0,n) {
33         if (dfs_num[u] == -1) {
34             dfsRoot = u; rootChildren = 0;
35             articulationPointAndBridgeUtil(u);
36             articulation_vertex[dfsRoot] = (rootChildren > 1);
37         }
38     }
39 }
40
41 void solve() {
42
43     int n, ed; cin >> n >> ed;
44     adj.assign(n, vector<pii>());
45
46     f(i,0,ed) {
47         int u, v, w; cin >> u >> v >> w;
48         adj[u].emplace_back(v, w);
49     }
50
51     dfs_num.assign(n, -1); dfs_low.assign(n, 0);

```

```

52     dfs_parent.assign(n, -1); articulation_vertex.assign(n, 0);
53
54     articulationPointAndBridge(n);
55
56     // Vertices: articulation_vertex[u] == 1
57     // Bridges: bridgesAns
58 }

```

5.4 Bipartido

```

1 // Description: Determina se um grafo eh bipartido ou nao
2 // Complexidade: O(V+E)
3
4 vector<vi> AL;
5
6 bool bipartido(int n) {
7
8     int s = 0;
9     queue<int> q; q.push(s);
10
11     vi color(n, INF); color[s] = 0;
12     bool ans = true;
13     while (!q.empty() && ans) {
14         int u = q.front(); q.pop();
15
16         for (auto &v : AL[u]) {
17             if (color[v] == INF) {
18                 color[v] = 1 - color[u];
19                 q.push(v);
20             }
21             else if (color[v] == color[u]) {
22                 ans = false;
23                 break;
24             }
25         }
26     }
27
28     return ans;
29 }
30
31 void solve() {
32
33     int n, edg; cin >> n >> edg;
34     AL.resize(n, vi());
35
36     while(edg--) {
37         int a, b; cin >> a >> b;
38         AL[a].push_back(b);
39         AL[b].push_back(a);
40     }
41
42     cout << bipartido(n) << endl;
43 }

```

5.5 Bfs Nivelado

```

1 // Description: Encontrar distancia entre S e outros pontos em que pontos
    estao agrupados (terminais)
2 // EXTRA: BFS diferenciado para armazenar distancias sem VIS
3
4 int n;
5 vi dist;
6 vector<vi> gruposDoItem, itensDoGrupo;
7
8 void bfs(int s) {
9
10     queue<pair<int, int>> q; q.push({s, 0});
11
12     while (!q.empty()) {
13         auto [v, dis] = q.front(); q.pop();
14
15         for(auto grupo : gruposDoItem[v]) {
16             for(auto u : itensDoGrupo[grupo]) {
17                 if (dist[u] == 0) {
18                     q.push({u, dis+1});
19                     dist[u] = dis + 1;
20                 }
21             }
22         }
23     }
24 }
25
26 void solve() {
27
28     int n, ed; cin >> n >> ed;
29     dist.clear(), itensDoGrupo.clear(), gruposDoItem.clear();
30     itensDoGrupo.resize(n);
31
32     f(i,0,ed) {
33
34         int q; cin >> q;
35         while(q--) {
36             int v; cin >> v;
37             gruposDoItem[v].push_back(i);
38             itensDoGrupo[i].push_back(v);
39         }
40     }
41
42     bfs(0);
43 }

```

5.6 Dfs

```

1 vector<int> adj[MAXN], parent;
2 int visited[MAXN];
3
4 // DFS com informacoes adicionais sobre o pai de cada vertice
5 // Complexidade:  $O(V + E)$ , onde V eh o numero de vertices e E o numero de
    areqas
6 void dfs(int p) {
7     memset(visited, 0, sizeof visited);
8     stack<int> st;
9     st.push(p);

```

```

11 while (!st.empty()) {
12     int v = st.top(); st.pop();
13
14     if (visited[v]) continue;
15     visited[v] = true;
16
17     for (int u : adj[v]) {
18         if (!visited[u]) {
19             parent[u] = v;
20             st.push(u);
21         }
22     }
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
    areqas
28 void dfs(int v) {
29     visited[v] = true;
30     for (int u : adj[v]) {
31         if (!visited[u]) {
32             parent[u] = v;
33             dfs(u);
34         }
35     }
36 }
37
38 void solve() {
39     int n; cin >> n;
40     for (int i = 0; i < n; i++) {
41         int u, v; cin >> u >> v;
42         adj[u].push_back(v);
43         adj[v].push_back(u);
44     }
45     dfs(0);
46 }

```

5.7 Successor Graph

```

1 // Encontra sucessor de um vertice dentro de um grafo direcionado
2 // Pre calcular:  $O(n \log n)$ 
3 // Consulta:  $O(\log n)$ 
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 }

```

5.8 Cycle Check

```

1 // Descriptionnn: Checa se um grafo direcionado possui ciclos e imprime os
    tipos de arestas.

```

```

2 // Complexidade: O(V + E)
3
4 vector<vector<pii>> adj;
5 vi dfs_num, dfs_parent;
6
7 void cycleCheck(int u) {
8     dfs_num[u] = -2;
9     for (auto &[v, w] : adj[u]) {
10         if (dfs_num[v] == -1) {
11             dfs_parent[v] = u;
12             cycleCheck(v);
13         }
14         else if (dfs_num[v] == -2) {
15             if (v == dfs_parent[u])
16                 cout << " Bidirectional Edge (" << u << ", " << v << ") - ("
17                 << v << ", " << u << ") \n";
18             else
19                 cout << "Back Edge (" << u << ", " << v << ") (Cycle) \n";
20         }
21         else if (dfs_num[v] == -3)
22             cout << " Forward/Cross Edge (" << u << ", " << v << ") \n";
23     }
24     dfs_num[u] = -3;
25 }
26
27 void solve() {
28     int n, ed; cin >> n >> ed;
29     adj.assign(ed, vector<pii>());
30
31     for (int i = 0; i < ed; ++i) {
32         int u, v, w; cin >> u >> v >> w;
33         adj[u].emplace_back(v, w);
34     }
35
36     cout << "Graph Edges Property Check \n";
37     dfs_num.assign(ed, -1);
38     dfs_parent.assign(ed, -1);
39     for (int u = 0; u < n; ++u)
40         if (dfs_num[u] == -1)
41             cycleCheck(u);
42 }

```

5.9 Dijkstra

```

1 // Encontra o menor caminho de um vértice s para todos os outros vértices
  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
35 //Complexidade: O(V)
36 vector<int> restorePath(int v) {
37     vector<int> path;
38     for (int u = v; u != -1; u = parent[u])
39         path.push_back(u);
40     reverse(path.begin(), path.end());
41     return path;
42 }
43
44 void solve() {
45
46     adj.resize(n); /*n = nodes*/
47     f(i,0,n) {
48         int a, b, w; cin >> a >> b >> w;
49         adj[a].push_back({b, w});
50         adj[b].push_back({a, w});
51     }
52     dijkstra(0);
53 }
54
55 // VARIANTES
56
57 /* Menor caminho de todos os vertices para um vertice s
58 -> Inverter a direcao das arestas
59 -> dijkstra(s)
60 */
61
62 /* Multi-Sources Shortest Paths
63 - Menor caminho de um conjunto de vertices para todos os outros
64 -> dist[a] = 0, q.push(a) para todo source a
65 -> dijkstra()
66 */

```

5.10 Labirinto

```

1 // Verifica se eh possivel sair de um labirinto
2 // Complexidade: O(4^(n*m))
3
4 vector<pair<int,int>> mov = {{1,0}, {0,1}, {-1,0}, {0,-1}};
5 vector<vector<int>> labirinto, sol;
6 vector<vector<bool>> visited;
7 int L, C;
8
9 bool valid(const int& x, const int& y) {
10     return x >= 0 and x < L and y >= 0 and y < C and labirinto[x][y] != 0
11     and !visited[x][y];
12 }
13
14 bool condicaoSaida(const int& x, const int& y) {
15     return labirinto[x][y] == 2;
16 }
17
18 bool search(const int& x, const int& y) {
19     if(!valid(x, y))
20         return false;
21
22     if(condicaoSaida(x,y)) {
23         sol[x][y] = 2;
24         return true;
25     }
26
27     sol[x][y] = 1;
28     visited[x][y] = true;
29
30     for(auto [dx, dy] : mov)
31         if(search(x+dx, y+dy))
32             return true;
33
34     sol[x][y] = 0;
35     return false;
36 }
37
38 int main() {
39
40     labirinto = {
41         {1, 0, 0, 0},
42         {1, 1, 0, 0},
43         {0, 1, 0, 0},
44         {1, 1, 1, 2}
45     };
46
47     L = labirinto.size(), C = labirinto[0].size();
48     sol.resize(L, vector<int>(C, 0));
49     visited.resize(L, vector<bool>(C, false));
50
51     cout << search(0, 0) << endl;
52 }

```

5.11 Bfs

```

1 // BFS com informacoes adicionais sobre a distancia e o pai de cada

```

```

2 // Complexidade: O(V + E), onde V eh o numero de vertices e E o numero de
3 // areqas
4
5 int n;
6 vector<bool> vis;
7 vector<int> d, p;
8 vector<vector<int>> adj;
9
10 void bfs(int s) {
11     queue<int> q; q.push(s);
12     vis[s] = true, d[s] = 0, p[s] = -1;
13
14     while (!q.empty()) {
15         int v = q.front(); q.pop();
16         vis[v] = true;
17
18         for (int u : adj[v]) {
19             if (!vis[u]) {
20                 vis[u] = true;
21                 q.push(u);
22                 // d[u] = d[v] + 1;
23                 // p[u] = v;
24             }
25         }
26     }
27 }
28
29 void solve() {
30     cin >> n;
31     adj.resize(n); d.resize(n, -1);
32     vis.resize(n); p.resize(n, -1);
33
34     for (int i = 0; i < n; i++) {
35         int u, v; cin >> u >> v;
36         adj[u].push_back(v);
37         adj[v].push_back(u);
38     }
39
40     bfs(0);
41 }
42
43 // OBS: Pode ser usado para encontrar o menor caminho entre dois vertices
44 // em um grafo sem pesos

```

5.12 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 int l, c;
8
9 bool valid(int x, int y) {

```



```

10     return (0 <= x and x < l and 0 <= y and y < c and !vis[x][y] /*and mat26
    [x][y]*/);
11 }
12
13 void bfs(int i, int j) {
14
15     queue<pair<int,int>> q; q.push({i, j});
16
17     while(!q.empty()) {
18
19         auto [u, v] = q.front(); q.pop();
20         vis[u][v] = true;
21
22         for(auto [x, y]: mov) {
23             if(valid(u+x, v+y)) {
24                 q.push({u+x,v+y});
25                 vis[u+x][v+y] = true;
26             }
27         }
28     }
29 }
30
31 void solve() {
32     cin >> l >> c;
33     mat.resize(l, vi(c));
34     vis.resize(l, vector<bool>(c, false));
35     /*preenche matriz*/
36     bfs(0,0);
37 }

```

5.13 Floyd Warshall

```

1 // Floyd-Warshall
2 //
3 // encontra o menor caminho entre todo
4 // par de vertices e detecta ciclo negativo
5 // retorna 1 sse ha ciclo negativo
6 // d[i][i] deve ser 0
7 // para i != j, d[i][j] deve ser w se ha uma aresta
8 // (i, j) de peso w, INF caso contrario
9 //
10 // O(n^3)
11
12 int n;
13 int d[MAX][MAX];
14
15 bool floyd_warshall() {
16     for (int k = 0; k < n; k++)
17         for (int i = 0; i < n; i++)
18             for (int j = 0; j < n; j++)
19                 d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
20
21     for (int i = 0; i < n; i++)
22         if (d[i][i] < 0) return 1;
23
24     return 0;
25 }

```

```

27 void solve() {
28     cin >> n; int edg; cin >> edg;
29     for (int i = 0; i < edg; i++) {
30         int u, v, w;
31         cin >> u >> v >> w;
32         d[u][v] = w;
33     }
34 }

```

5.14 Pontos Articulacao

```

1 // Description: Encontra os pontos de articulo de um grafo ão
    direcionado
2 // Complexidade: O(V*(V+E))
3
4 int V;
5 vector<vi> adj;
6 vi ans;
7
8 void dfs(vector<bool>& vis, int i, int curr) {
9     vis[curr] = 1;
10    for (auto x : adj[curr]) {
11        if (x != i) {
12            if (!vis[x]) {
13                dfs(vis, i, x);
14            }
15        }
16    }
17 }
18
19 void AP() {
20
21     f(i,1,V+1) {
22         int components = 0;
23         vector<bool> vis(V + 1, 0);
24         f(j,1, V+1) {
25             if (j != i) {
26                 if (!vis[j]) {
27                     components++;
28                     dfs(vis, i, j);
29                 }
30             }
31         }
32         if (components > 1) {
33             ans.push_back(i);
34         }
35     }
36 }
37
38 void solve() {
39
40     V = n;
41     adj.clear(), ans.clear();
42     adj.resize(V+1);
43
44     while(edg--) {

```

```

45     int a, b; cin >> a >> b;
46     adj[a].push_back(b);
47     adj[b].push_back(a);
48 }
49
50 AP();
51
52 // Vertices articulacao: ans
53 }

```

5.15 Graham Scan(elastico)

```

1 // çãFunco para calcular o produto vetorial de dois vetores
2 int cross_product(const pair<int, int>& o, const pair<int, int>& a, const
3     pair<int, int>& b) {
4     return (a.first - o.first) * (b.second - o.second) - (a.second - o.
5         second) * (b.first - o.first);
6 }
7
8 // çãFunco para encontrar o ponto mais baixo (esquerda mais baixo)
9 pair<int, int> find_lowest_point(const vector<pair<int, int>>& points) {
10     pair<int, int> lowest = points[0];
11     for (const auto& point : points) {
12         if (point.second < lowest.second || (point.second == lowest.second
13             && point.first < lowest.first)) {
14             lowest = point;
15         }
16     }
17     return lowest;
18 }
19
20 // çãFunco para ordenar pontos por ângulo polar em çãrelao ao ponto mais
21     baixo
22 bool compare(const pair<int, int>& a, const pair<int, int>& b, const pair<
23     int, int>& lowest_point) {
24     int cross = cross_product(lowest_point, a, b);
25     if (cross != 0) {
26         return cross > 0;
27     }
28     return (a.first != b.first) ? (a.first < b.first) : (a.second < b.
29         second);
30 }
31
32 // çãFunco para encontrar o óenvoltrio convexo usando o algoritmo de
33     Varredura de Graham
34 vector<pair<int, int>> convex_hull(vector<pair<int, int>>& points) {
35     vector<pair<int, int>> convex_polygon;
36
37     if (points.size() < 3) return convex_polygon;
38
39     pair<int, int> lowest_point = find_lowest_point(points);
40     sort(points.begin(), points.end(), [&lowest_point](const pair<int, int
41         >& a, const pair<int, int>& b) {
42         return compare(a, b, lowest_point);
43     });
44 }

```

```

37     convex_polygon.push_back(points[0]);
38     convex_polygon.push_back(points[1]);
39
40     for (int i = 2; i < points.size(); ++i) {
41         while (convex_polygon.size() >= 2 && cross_product(convex_polygon[
42             convex_polygon.size() - 2], convex_polygon.back(), points[i]) <= 0) {
43             convex_polygon.pop_back();
44         }
45         convex_polygon.push_back(points[i]);
46     }
47
48     return convex_polygon;
49 }
50
51 void solve() {
52     int n, turma = 0;
53
54     vector<pair<int, int>> points(n);
55     for (int i = 0; i < n; ++i) {
56         cin >> points[i].first >> points[i].second; // x y
57     }
58
59     vector<pair<int, int>> convex_polygon = convex_hull(points);
60     int num_vertices = convex_polygon.size();
61
62     cout << num_vertices << endl; // qnt de vertices , se quiser os
63     pontos so usar o vi convex_polygon
64
65     cout << endl;
66 }

```

5.16 Kruskal

```

1 // DEscriçao: Encontra a arvore geradora minima de um grafo
2 // Complexidade: O(E log V)
3
4 vector<int> id, sz;
5
6 int find(int a){ // O(a(N)) amortizado
7     return id[a] = (id[a] == a ? a : find(id[a]));
8 }
9
10 void uni(int a, int b) { // O(a(N)) amortizado
11     a = find(a), b = find(b);
12     if(a == b) return;
13
14     if(sz[a] > sz[b]) swap(a,b);
15     id[a] = b, sz[b] += sz[a];
16 }
17
18 pair<int, vector<tuple<int, int, int>>> kruskal(vector<tuple<int, int, int
19     >>& edg) {
20
21     sort(edg.begin(), edg.end()); // Minimum Spanning Tree
22
23     int cost = 0;

```

```

23 vector<tuple<int, int, int>> mst; // opcional
24 for (auto [w,x,y] : edg) if (find(x) != find(y)) {
25     mst.emplace_back(w, x, y); // opcional
26     cost += w;
27     uni(x,y);
28 }
29 return {cost, mst};
30 }
31
32 void solve() {
33
34     int n, ed;
35
36     id.resize(n); iota(all(id), 0);
37     sz.resize(n, -1);
38     vector<tuple<int, int, int>> edg;
39
40     f(i,0,ed) {
41         int a, b, w; cin >> a >> b >> w;
42         edg.push_back({w, a, b});
43     }
44
45     auto [cost, mst] = kruskal(edg);
46 }
47
48 // VARIANTES
49
50 // Maximum Spanning Tree: sort(edg.rbegin(), edg.rend());
51
52 /* 'Minimum' Spanning Subgraph:
53     - Algumas arestas ja foram adicionadas (maior prioridade - Questao das
54       rodovias)
55     - Arestas que nao foram adicionadas (menor prioridade - ferrovias)
56     -> kruskal(rodovias); kruskal(ferrovias);
57 */
58
59 /* Minimum Spanning Forest:
60     - Queremos uma floresta com k componentes
61     -> kruskal(edg); if(mst.sizer() == k) break;
62 */
63
64 /* MiniMax
65     - Encontrar menor caminho entre dous vertices com maior quantidade de
66       arestas
67     -> kruskal(edg); dijsktra(mst);
68 */
69
70 /* Second Best MST
71     - Encontrar a segunda melhor arvore geradora minima
72     -> kruskal(edg);
73     -> flag mst[i] = 1;
74     -> sort(cmp(edg.flag != -1)) => da prioridade para outras arestas
75 */

```

5.17 Kosaraju

```

1 // Description: Encontra o numero de componentes fortemente conexas em um

```

```

        grafo direcionado
2 // Complexidade: O(V + E)
3
4 int dfsNumberCounter, numSCC;
5 vector<vii> adj, adj_t;
6 vi dfs_num, dfs_low, S, visited;
7 stack<int> St;
8
9 void kosarajuUtil(int u, int pass) {
10     dfs_num[u] = 1;
11     vii &neighbor = (pass == 1) ? adj[u] : adj_t[u];
12     for (auto &[v, w] : neighbor)
13         if (dfs_num[v] == -1)
14             kosarajuUtil(v, pass);
15     S.push_back(u);
16 }
17
18 bool kosaraju(int n) {
19
20     S.clear();
21     dfs_num.assign(n, -1);
22
23     f(u,0,n) {
24         if (dfs_num[u] == -1)
25             kosarajuUtil(u, 1);
26     }
27
28     int numSCC = 0;
29     dfs_num.assign(n, -1);
30     f(i,n-1,-1) {
31         if (dfs_num[S[i]] == -1)
32             numSCC++, kosarajuUtil(S[i], 2);
33     }
34
35     return numSCC == 1;
36 }
37
38 void solve() {
39
40     int n, ed; cin >> n >> ed;
41     adj.assign(n, vii());
42     adj_t.assign(n, vii());
43
44     while (ed--) {
45         int u, v, w; cin >> u >> v >> w;
46         AL[u].emplace_back(v, 1);
47         adj_t[v].emplace_back(u, 1);
48     }
49
50     // Printa se o grafo eh fortemente conexo
51     cout << kosaraju(n) << endl;
52
53     // Printa o numero de componentes fortemente conexas
54     cout << numSCC << endl;
55
56     // Printa os vertices de cada componente fortemente conexa
57     f(i,0,n){

```

```

58     if (dfs_num[i] == -1) cout << i << ": " << "Nao visitado" << endl;
59     else cout << i << ": " << dfs_num[i] << endl;
60 }
61 }

```

5.18 Euler Tree

```

1 // Descricao: Encontra a euler tree de um grafo
2 // Complexidade: O(n)
3 vector<vector<int>> adj(MAX);
4 vector<int> vis(MAX, 0);
5 vector<int> euTree(MAX);
6
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 void solve() {
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; eulerTree(1, index);
27 }

```

6 Matematica

6.1 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    }

```

```

17
18
19 pair<int, vector<int>> bigSum(const pair<int, vector<int>>& a, const pair<
   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
71 }

```

6.2 Mmc

```

1 // Description: Calcula o mmc de dois números inteiros.
2 // Complexidade: O(logn) onde n eh o maior numero
3 int mmc(int a, int b) {
4     return a / mdc(a, b) * b;
5 }

```

6.3 Fatoracao

```

1 // Fatora um número em seus fatores primos
2 // Complexidade: O(sqrt(n))
3 map<int, int> factorize(int n) {
4     map<int, int> factorsOfN;
5     int lowestPrimeFactorOfN = 2;
6
7     while (n != 1) {
8         lowestPrimeFactorOfN = lowestPrimeFactor(n, lowestPrimeFactorOfN);
9         factorsOfN[lowestPrimeFactorOfN] = 1;
10        n /= lowestPrimeFactorOfN;
11        while (not (n % lowestPrimeFactorOfN)) {
12            factorsOfN[lowestPrimeFactorOfN]++;
13            n /= lowestPrimeFactorOfN;
14        }
15    }
16
17    return factorsOfN;
18 }

```

6.4 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 }

```

6.5 Mmc Multiplo

```

1 // Description: Calcula o mmc de um vetor de inteiros.
2 // Complexidade: O(nlogn) onde n eh o tamanho do vetor
3 int mmc_many(vector<int> arr)
4 {
5     int result = arr[0];
6
7     for (int &num : arr)

```

```

8         result = (num * result / mdc(num, result));
9     return result;
10 }

```

6.6 Contar Quanti Solucoes Eq 2 Variaveis

```

1 // Description: Dada uma equacao de 2 variaveis, calcula quantas
2 // combinacoes {x,y}
3 // inteiras que resolvem essa equacao
4 // Complexidade: O(sqrt(c))
5 // y = numerador / denominador
6 int numerador(int x) { return c - x; } // expressao do numerador
7 int denominador(int x) { return 2 * x + 1; } // expressao do denominador
8
9 int count2VariableIntegerEquationAnswers() {
10
11     unordered_set<pair<int,int>, PairHash> ans; int lim = sqrt(c);
12     for(int i=1; i<= lim; i++) {
13         if (numerador(i) % denominador(i) == 0) {
14             int x = i, y = numerador(i) / denominador(i);
15             if(!ans.count({x,y}) and !ans.count({y,x}))
16                 ans.insert({x,y});
17         }
18     }
19     return ans.size();
20 }

```

6.7 Mdc

```

1 // Description: Calcula o mdc de dois numeros inteiros.
2 // Complexidade: O(logn) onde n eh o maior numero
3 int mdc(int a, int b) {
4     for (int r = a % b; r; a = b, b = r, r = a % b);
5     return b;
6 }

```

6.8 Decimal Para Fracao

```

1 // Converte um decimal para fracao irredutivel
2 // Complexidade: O(log n)
3 pair<int, int> toFraction(double n, unsigned p) {
4     const int tenP = pow(10, p);
5     const int t = (int) (n * tenP);
6     const int rMdc = mdc(t, tenP);
7     return {t / rMdc, tenP / rMdc};
8 }

```

6.9 Mdc Multiplo

```

1 // Description: Calcula o MDC de um vetor de inteiros.
2 // Complexidade: O(nlogn) 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 }

```

6.10 Tabela Verdade

```

1 // Gerar tabela verdade de uma expressão booleana
2 // Complexidade:  $O(2^n)$ 
3
4 vector<vector<int>> tabelaVerdade;
5 int indexTabela = 0;
6
7 void backtracking(int posicao, vector<int>& conj_bool) {
8
9     if(posicao == conj_bool.size()) { // Se chegou ao fim da BST
10         for(size_t i=0; i<conj_bool.size(); i++) {
11             tabelaVerdade[indexTabela].push_back(conj_bool[i]);
12         }
13         indexTabela++;
14
15     } else {
16         conj_bool[posicao] = 1;
17         backtracking(posicao+1, conj_bool);
18         conj_bool[posicao] = 0;
19         backtracking(posicao+1, conj_bool);
20     }
21 }
22
23 int main() {
24
25     int n = 3;
26
27     vector<int> linhaBool (n, false);
28     tabelaVerdade.resize(pow(2,n));
29
30     backtracking(0, linhaBool);
31 }

```

6.11 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 }

```

6.12 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 }

```

6.13 Divisores

```

1 // Descricao: Calcula os divisores de c, sem incluir c, sem ser fatorado
2 // Complexidade:  $O(\sqrt{c})$ 
3 set<int> calculaDivisores(int c) {
4     int lim = sqrt(c);
5     set<int> divisors;
6
7     for(int i = 1; i <= lim; i++) {
8         if (c % i == 0) {
9             if(c/i != i)
10                 divisors.insert(c/i);
11             divisors.insert(i);
12         }
13     }
14
15     return divisors;
16 }

```

6.14 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 }

```

6.15 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 }

```

6.16 Conversão De Bases

```

1 // Converter um decimal (10) para base n [2, 8, 10, 16]
2 // Complexidade: O(log n)
3 char charForDigit(int digit) {
4     if (digit > 9) return digit + 87;
5     return digit + 48;
6 }
7
8 string decimalToBase(int n, int base = 10) {
9     if (not n) return "0";
10    stringstream ss;
11    for (int i = n; i > 0; i /= base) {
12        ss << charForDigit(i % base);
13    }
14    string s = ss.str();
15    reverse(s.begin(), s.end());
16    return s;
17 }
18
19 // Converter um número de base [2, 8, 10, 16] para decimal (10)
20 // Complexidade: O(n)

```

```

21 int intForDigit(char digit) {
22     int intDigit = digit - 48;
23     if (intDigit > 9) return digit - 87;
24     return intDigit;
25 }
26
27 int baseToDecimal(const string& n, int base = 10) {
28     int result = 0;
29     int basePow = 1;
30     for (auto it = n.rbegin(); it != n.rend(); ++it, basePow *= base)
31         result += intForDigit(*it) * basePow;
32     return result;
33 }

```

6.17 Números Grandes

```

1 public static void BigInteger() {
2
3     BigInteger a = BigInteger.valueOf(1000000000);
4     a = new BigInteger("1000000000");
5
6     // Operações 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    // Comparação
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    // Conversão para string
21    String m = a.toString();
22
23    // Conversão para inteiro
24    int _int = a.intValue();
25    long _long = a.longValue();
26    double _doub = a.doubleValue();
27
28    // Potência
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
41    // Operações com reais grandes

```

```

42     BigDecimal arit = a.add(a);
43         arit = a.subtract(a);
44         arit = a.multiply(a);
45         arit = a.divide(a);
46         arit = a.remainder(a);
47
48     // Comparao
49     boolean bool = a.equals(a);
50         bool = a.compareTo(a) > 0;
51         bool = a.compareTo(a) < 0;
52         bool = a.compareTo(a) >= 0;
53         bool = a.compareTo(a) <= 0;
54
55     // Converso para string
56     String m = a.toString();
57
58     // Converso para inteiro
59     int _int = a.intValue();
60     long _long = a.longValue();
61     double _doub = a.doubleValue();
62
63     // Potncia
64     BigDecimal _pot = a.pow(10);
65 }

```

6.18 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 }

```

6.19 Primo

```

1 // Descricao: Funcao que verifica se um numero n eh primo.
2 // Complexidade: O(sqrt(n))
3 int lowestPrimeFactor(int n, int startPrime = 2) {
4     if (startPrime <= 3) {
5         if (not (n & 1))
6             return 2;
7         if (not (n % 3))
8             return 3;
9         startPrime = 5;
10    }
11
12    for (int i = startPrime; i * i <= n; i += (i + 1) % 6 ? 4 : 2)
13        if (not (n % i))
14            return i;
15    return n;
16 }
17
18 bool isPrime(int n) {
19     return n > 1 and lowestPrimeFactor(n) == n;
20 }

```

6.20 Dois Primos Somam Num

```

1 // Description: Verifica se dois numeros primos somam um numero n.
2 // Complexity: O(sqrt(n))
3 bool twoNumsSumPrime(int n) {
4
5     if(n % 2 == 0) return true;
6     return isPrime(n-2);
7 }

```

6.21 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 {
7     int ans = 1;
8     while (b)
9     {
10         if (b & 1)
11             ans = ans * a % mod;
12         a = a * a % mod;
13         b >>= 1;
14     }

```



```

15     return ans;
16 }

```

7 Matriz

7.1 Maior Retangulo Binario Em Matriz

```

1 // Description: Encontra o maior âretnngulo âbinrio em uma matriz.
2 // Time: O(n*m)
3 // Space: O(n*m)
4 tuple<int, int, int> maximalRectangle(const vector<vector<int>>& mat) {
5     int r = mat.size();
6     if(r == 0) return make_tuple(0, 0, 0);
7     int c = mat[0].size();
8
9     vector<vector<int>> dp(r+1, vector<int>(c));
10
11     int mx = 0;
12     int area = 0, height = 0, length = 0;
13     for(int i=1; i<r; ++i) {
14         int leftBound = -1;
15         stack<int> st;
16         vector<int> left(c);
17
18         for(int j=0; j<c; ++j) {
19             if(mat[i][j] == 1) {
20                 mat[i][j] = 1+mat[i-1][j];
21                 while(!st.empty() and mat[i][st.top()] >= mat[i][j])
22                     st.pop();
23
24                 int val = leftBound;
25                 if(!st.empty())
26                     val = max(val, st.top());
27
28                 left[j] = val;
29             } else {
30                 leftBound = j;
31                 left[j] = 0;
32             }
33             st.push(j);
34         }
35         while(!st.empty()) st.pop();
36
37         int rightBound = c;
38         for(int j=c-1; j>=0; j--) {
39             if(mat[i][j] != 0) {
40
41                 while(!st.empty() and mat[i][st.top()] >= mat[i][j])
42                     st.pop();
43
44                 int val = rightBound;
45                 if(!st.empty())
46                     val = min(val, st.top());
47
48                 dp[i][j] = (mat[i][j]+1) * (((val-1)-(left[j]+1)+1)+1);
49                 if (dp[i][j] > mx) {

```

```

50                     mx = dp[i][j];
51                     area = mx;
52                     height = mat[i][j];
53                     length = (val-1)-(left[j]+1)+1;
54                 }
55                 st.push(j);
56             } else {
57                 dp[i][j] = 0;
58                 rightBound = j;
59             }
60         }
61     }
62
63     return make_tuple(area, height, length);
64 }

```

7.2 Max 2d Range Sum

```

1 // Maximum Sum
2 // O(n^3) 1D DP + greedy (Kadane's) solution, 0.000s in UVa
3
4 #include <bits/stdc++.h>
5 using namespace std;
6
7 #define f(i,s,e) for(int i=s;i<e;i++)
8 #define MAX_n 110
9
10 int A[MAX_n][MAX_n];
11
12 int maxMatrixSum(vector<vector<int>> mat) {
13
14     int n = mat.size();
15     int m = mat[0].size();
16
17     f(i,0,n) {
18         f(j,0,m) {
19             if (j > 0)
20                 mat[i][j] += mat[i][j - 1];
21         }
22     }
23
24     int maxSum = INT_MIN;
25     f(l,0,m) {
26         f(r,l,m) {
27             vector<int> sum(n, 0);
28             f(row,0,n) {
29                 sum[row] = mat[row][r] - (l > 0 ? mat[row][l - 1] : 0);
30             }
31             int maxSubRect = sum[0];
32             f(i,1,n) {
33                 if (sum[i - 1] > 0)
34                     sum[i] += sum[i - 1];
35                 maxSubRect = max(maxSubRect, sum[i]);
36             }
37             maxSum = max(maxSum, maxSubRect);
38         }
39     }

```

```

40     return maxSum;
41 }
42 }

```

8 Strings

8.1 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 }

```

8.2 Ocorrencias

```

1 // Description: çãFunco que retorna um vetor com as çõposies de todas as
2 // çocorrncias de uma substring em uma string.
3 // Complexidade: O(n * m) onde n é o tamanho da string e m é o tamanho da
4 // substring.
5 vector<int> ocorrencias(string str, string sub){
6     vector<int> ret;
7     int index = str.find(sub);
8     while(index!=-1){
9         ret.push_back(index);
10        index = str.find(sub, index+1);
11    }
12    return ret;
13 }

```

8.3 Chaves Colchetes Parenteses

```

1 // Description: Verifica se s tem uma êsequencia valida de {}, [] e ()
2 // Complexidade: O(n)
3 bool brackets(string s) {
4     stack<char> st;
5
6     for (char c : s) {
7         if (c == '(' || c == '[' || c == '{') {
8             st.push(c);
9         } else {
10            if (st.empty()) return false;
11            if (c == ')' and st.top() != '(') return false;
12            if (c == ']' and st.top() != '[') return false;
13            if (c == '}' and st.top() != '{') return false;
14            st.pop();
15        }
16    }
17 }

```

```

18     return st.empty();
19 }

```

8.4 Permutacao

```

1 // Funcao para gerar todas as permutacoes de uma string
2 // Complexidade: O(n!)
3
4 void permute(string& s, int l, int r) {
5     if (l == r)
6         permutacoes.push_back(s);
7     else {
8         for (int i = l; i <= r; i++) {
9             swap(s[l], s[i]);
10            permute(s, l+1, r);
11            swap(s[l], s[i]);
12        }
13    }
14 }
15
16 int main() {
17
18     string str = "ABC";
19     int n = str.length();
20     permute(str, 0, n-1);
21 }

```

8.5 Lower Upper

```

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 é uppercase: isupper(c);

```

8.6 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 }

```

8.7 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 }

```

8.8 Infixo Para Posfixo

```

1 // Description: Converte uma expressao matematica infixa para posfixa
2 // Complexidade: O(n)
3 string infixToPostfix(string s) {
4     stack<char> st;
5     string res;
6     for (char c : s) {
7         if (isdigit(c))
8             res += c;
9         else if (c == '(')
10            st.push(c);
11        else if (c == ')') {
12            while (st.top() != '(') {
13                res += st.top();
14                st.pop();
15            }
16            st.pop();
17        } else {
18            while (!st.empty() and st.top() != '(' and
19                (c == '+' or c == '-' or (st.top() == '*' or st.top()
20                == '/')))) {
21                res += st.top();
22                st.pop();
23            }
24            st.push(c);
25        }
26    }
27    return res + st.top();
28 }

```

```

25    }
26    while (!st.empty()) {
27        res += st.top();
28        st.pop();
29    }
30    return res;
31 }

```

8.9 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 }

```

8.10 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 }

```

8.11 Calculadora Posfixo

```

1 // Description: Calculadora de expressoes posfixas
2 // Complexidade: O(n)
3 int posfixo(string s) {
4     stack<int> st;
5     for (char c : s) {
6         if (isdigit(c)) {
7             st.push(c - '0');
8         }
9     }
10    return st.top();
11 }

```

```

8     } else {
9         int b = st.top(); st.pop();
10        int a = st.top(); st.pop();
11        if (c == '+') st.push(a + b);
12        if (c == '-') st.push(a - b);
13        if (c == '*') st.push(a * b);
14        if (c == '/') st.push(a / b);
15    }
16    }
17    return st.top();
18 }

```

9 Vector

9.1 Maior Retangulo Em Histograma

```

1 // Calcula area do maior retangulo em um histograma
2 // Complexidade: O(n)
3 int maxHistogramRect(const vector<int>& hist) {
4     stack<int> s;
5     int n = hist.size();
6
7     int ans = 0, tp, area_with_top;
8
9     int i = 0;
10    while (i < n) {
11
12        if (s.empty() || hist[s.top()] <= hist[i])
13            s.push(i++);
14
15        else {
16            tp = s.top(); s.pop();
17
18            area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);
19
20            if (ans < area_with_top)
21                ans = area_with_top;
22        }
23    }
24
25    while (!s.empty()) {
26        tp = s.top(); s.pop();
27        area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);
28
29        if (ans < area_with_top)
30            ans = area_with_top;
31    }
32
33    return ans;
34 }
35
36 int main() {
37     vector<int> hist = { 6, 2, 5, 4, 5, 1, 6 };
38     cout << maxHistogramRect(hist) << endl;
39 }

```

9.2 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 }

```

9.3 Subset Sum

```

1 // Description: Verifica se algum subset dentro do array soma igual a sum

```

```

2 // Complexidade Temporal: O(sum * n)
3 // Complexidade Espacial: O(sum * n)
4
5 bool isSubsetSum(vi set, int n, int sum) {
6     bool subset[n + 1][sum + 1];
7
8     for (int i = 0; i <= n; i++)
9         subset[i][0] = true;
10
11     for (int i = 1; i <= sum; i++)
12         subset[0][i] = false;
13
14     for (int i = 1; i <= n; i++) {
15         for (int j = 1; j <= sum; j++) {
16             if (j < set[i - 1])
17                 subset[i][j] = subset[i - 1][j];
18             if (j >= set[i - 1])
19                 subset[i][j]
20                     = subset[i - 1][j]
21                     || subset[i - 1][j - set[i - 1]];
22         }
23     }
24
25     return subset[n][sum];
26 }

```

9.4 Contar Subarrays Somam 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,
12         encontramos um subarray
13
14         if (prevSum.find(currentSum - k) != prevSum.end()) // se subarray
15         com soma (currentSum - k) existir, sabe que [0:n] eh um subarray com
16         soma k
17             ret += (prevSum[currentSum - k]);
18
19         prevSum[currentSum]++;
20     }
21
22     return ret;
23 }

```

9.5 Maior Triangulo Em Histograma

```

1 // Calcula o maior âtringulo em um histograma
2 // Complexidade: O(n)

```

```

3 int maiorTrianguloEmHistograma(const vector<int>& histograma) {
4
5     int n = histograma.size();
6     vector<int> esquerda(n), direita(n);
7
8     esquerda[0] = 1;
9     f(i,1,n) {
10         esquerda[i] = min(histograma[i], esquerda[i - 1] + 1);
11     }
12
13     direita[n - 1] = 1;
14     rf(i,n-1,0) {
15         direita[i] = min(histograma[i], direita[i + 1] + 1);
16     }
17
18     int ans = 0;
19     f(i,0,n) {
20         ans = max(ans, min(esquerda[i], direita[i]));
21     }
22
23     return ans;
24 }
25 }

```

9.6 K Maior Elemento

```

1 // Description: Encontra o k-ésimo maior elemento de um vetor
2 // Complexidade: O(n)
3
4 int Partition(vector<int>& A, int l, int r) {
5     int p = A[l];
6     int m = l;
7     for (int k = l+1; k <= r; ++k) {
8         if (A[k] < p) {
9             ++m;
10             swap(A[k], A[m]);
11         }
12     }
13     swap(A[l], A[m]);
14     return m;
15 }
16
17 int RandPartition(vector<int>& A, int l, int r) {
18     int p = l + rand() % (r-l+1);
19     swap(A[l], A[p]);
20     return Partition(A, l, r);
21 }
22
23 int QuickSelect(vector<int>& A, int l, int r, int k) {
24     if (l == r) return A[l];
25     int q = RandPartition(A, l, r);
26     if (q+1 == k)
27         return A[q];
28     else if (q+1 > k)
29         return QuickSelect(A, l, q-1, k);
30     else
31         return QuickSelect(A, q+1, r, k);

```

```

32 }
33
34 void solve() {
35     vector<int> A = { 2, 8, 7, 1, 5, 4, 6, 3 };
36     int k = 1;
37     cout << QuickSelect(A, 0, A.size()-1, k) << endl;
38 }

```

9.7 Remove Repetitive

```

1 // Remove repetitive elements from a vector
2 // Complexity: O(n)
3 vector<int> removeRepetitive(const vector<int>& vec) {
4
5     unordered_set<int> s;
6     s.reserve(vec.size());
7
8     vector<int> ans;
9
10    for (int num : vec) {
11        if (s.insert(num).second)
12            v.push_back(num);
13    }
14
15    return ans;
16 }
17
18 void solve() {
19     vector<int> v = {1, 3, 2, 5, 4, 2, 3, 4, 5};
20     vector<int> ans = removeRepetitive(v); // {1, 3, 2, 5, 4}
21 }

```

9.8 Troco

```

1 // Description: Retorna o menor número de moedas para formar um valor n
2 // Complexidade: O(n*m)
3 vector<int> troco(vector<int> coins, int n) {
4     int first[n];
5     value[0] = 0;
6     for(int x=1; x<=n; x++) {
7         value[x] = INF;
8         for(auto c : coins) {
9             if(x-c >= 0 and value[x-c] + 1 < value[x]) {
10                 value[x] = value[x-c]+1;
11                 first[x] = c;
12             }
13         }
14     }
15
16     vector<int> ans;
17     while(n>0) {
18         ans.push_back(first[n]);
19         n -= first[n];
20     }
21     return ans;
22 }

```

```

23
24 void solve() {
25     vector<int> coins = {1, 3, 4};
26     vector<int> ans = troco(coins, 6); // {3,3}
27 }

```

9.9 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 }

```

9.10 Maior Subsequência Crescente

```

1 // Retorna o tamanho da maior subsequencia crescente de v
2 // Complexidade: O(n log(n))
3 int maiorSubCrescSize(vector<int> &v) {
4
5     vector<int> pilha;
6     for (int i = 0; i < v.size(); i++) {
7         auto it = lower_bound(pilha.begin(), pilha.end(), v[i]);
8         if (it == pilha.end())
9             pilha.push_back(v[i]);
10        else
11            *it = v[i];
12    }
13
14    return pilha.size();
15 }
16
17 // Retorna a maior subsequencia crescente de v
18 // Complexidade: O(n log(n))
19 vector<int> maiorSubCresc(vector<int> &v) {
20
21     vector<int> pilha, resp;
22     int pos[MAXN], pai[MAXN];
23     for (int i = 0; i < v.size(); i++) {
24         auto it = lower_bound(pilha.begin(), pilha.end(), v[i]);
25         int p = it - pilha.begin();
26         if (it == pilha.end())
27             pilha.push_back(v[i]);
28         else
29             *it = v[i];
30         pos[p] = i;

```

```

31     if (p == 0)
32         pai[i] = -1; // seu pai áser -1
33     else
34         pai[i] = pos[p - 1];
35 }
36
37 int p = pos[pilha.size() - 1];
38 while (p >= 0) {
39     resp.PB(v[p]);
40     p = pai[p];
41 }
42 reverse(resp.begin(), resp.end());
43
44 return resp;
45 }
46
47 void solve() {
48     vector<int> v = {1, 3, 2, 5, 4, 2, 3, 4, 5};
49     cout << maiorSubCrescSize(v) << endl // 5
50     /*****
51     vector<int> ans = maiorSubCresc(v); // {1,2,3,4,5}
52 }

```

9.11 Maior Subsequencia Comum

```

1 int s1[MAXN], s2[MAXN], tab[MAXN][MAXN];
2
3 // Description: Retorna o tamanho da maior subsequencia comum entre s1 e
4 // Complexidade: O(n*m)
5 int lcs(int a, int b){
6
7     if(tab[a][b]>=0) return tab[a][b];
8     if(a==0 or b==0) return tab[a][b]=0;
9     if(s1[a]==s2[b]) return 1 + lcs(a-1, b-1);
10    return tab[a][b] = max(lcs(a-1, b), lcs(a, b-1));
11 }
12
13 void solve() {
14
15     s1 = {1, 3, 2, 5, 4, 2, 3, 4, 5};
16     s2 = {1, 2, 3, 4, 5};
17     int n = s1.size(), m = s2.size();
18     memset(tab, -1, sizeof(tab));
19     cout << lcs(n, m) << endl; // 5
20 }

```

9.12 Soma Maxima Sequencial

```

1 // Description: Soma maxima sequencial de um vetor
2 // Complexidade: O(n)
3 int max_sum(vector<int> s) {
4
5     int ans = 0, maior = 0;
6
7     for(int i = 0; i < s.size(); i++) {

```

```

8         maior = max(0, maior+s[i]);
9         ans = max(resp, maior);
10    }
11
12    return ans;
13 }
14
15 void solve() {
16     vector<int> v = {1,-3,5,-1,2,-1};
17     cout << max_sum(v) << endl; // 6 = {5,-1,2}
18 }

```

10 Outros

10.1 Intervalos

```

1 // Conta quantos intervalos nao tem overlap ([a,b] e [b,c] nao geram)
2
3 bool cmp(const pair<int,int>& p1, const pair<int,int>& p2) {
4     if(p1.second != p2.second) return p1.second < p2.second;
5     return p1.first < p2.first;
6 }
7
8 int countNonOverlappingIntervals(vector<pair<int,int>> intervals) {
9     sort(all(intervals), cmp);
10    int firstTermino = intervals[0].second;
11    int ans = 1;
12    f(i,1,intervals.size()) {
13        if(intervals[i].first >= firstTermino) {
14            ans++;
15            firstTermino = intervals[i].second;
16        }
17    }
18
19    return ans;
20 }

```

10.2 Dp

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int MAX_gm = 30; // up to 20 garments at most and 20 models/garment
5 const int MAX_M = 210; // maximum budget is 200
6
7 int M, C, price[MAX_gm][MAX_gm]; // price[g (<= 20)][k (<= 20)]
8 int memo[MAX_gm][MAX_M]; // TOP-DOWN: dp table [g (< 20)][money
9                             (<= 200)]
10
11 int dp(int g, int money) {
12
13     if (money < 0) return -1e9;
14     if (g == C) return M - money;
15     if (memo[g][money] != -1)
16         return memo[g][money]; // avaliar linha g com dinheiro money (cada
17                                 caso pensavel)

```

```

16     int ans = -1;
17     for (int k = 1; k <= price[g][0]; ++k)
18         ans = max(ans, dp(g + 1, money - price[g][k]));
19     return memo[g][money] = ans;
20 }
21
22 int main() {
23     int TC;
24     scanf("%d", &TC);
25     while (TC--)
26     {
27         scanf("%d %d", &M, &C);
28         for (int g = 0; g < C; ++g)
29         {
30             scanf("%d", &price[g][0]); // store k in price[g][0]
31             for (int k = 1; k <= price[g][0]; ++k)
32                 scanf("%d", &price[g][k]);
33         }
34         memset(memo, -1, sizeof memo); // TOP-DOWN: init memo
35         if (dp(0, M) < 0)
36             printf("no solution\n"); // start the top-down DP
37         else
38             printf("%d\n", dp(0, M));
39     }
40     return 0;
41 }

```

10.3 Binary Search

```

1 // Description:  Implementação do algoritmo de busca binária.
2 // Complexidade:  $O(\log n)$  onde  $n$  é o tamanho do vetor
3 int BinarySearch(const 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 }

```

10.4 Mochila

```

1 // Description: Problema da mochila 0-1: retorna o valor máximo que pode
   ser carregado
2 // Complexidade:  $O(n \cdot \text{capacidade})$ 
3
4 const int MAX_QNT_OBJETOS = 60; // 50 + 10
5 const int MAX_PESO_OBJETO = 1010; // 1000 + 10
6
7 int n, memo[MAX_QNT_OBJETOS][MAX_PESO_OBJETO];
8 vi valor, peso;
9
10 int mochila(int id, int remW) {

```

```

11     if ((id == n) || (remW == 0)) return 0;
12     int &ans = memo[id][remW];
13     if (ans != -1) return ans;
14     if (peso[id] > remW) return ans = mochila(id+1, remW);
15     return ans = max(mochila(id+1, remW), valor[id]+mochila(id+1, remW-
   peso[id]));
16 }
17
18 void solve() {
19
20     memset(memo, -1, sizeof memo);
21
22     int capacidadeMochila; cin >> capacidadeMochila;
23
24     f(i,0, capacidadeMochila) { memo[0][i] = 0; } // testar com e sem essa
   linha
25
26     cin >> n;
27
28     valor.assign(n, 0);
29     peso.assign(n, 0);
30
31     f(i,0,n) {
32         cin >> peso[i] >> valor[i];
33     }
34
35     cout << mochila(0, capacidadeMochila) << endl;
36
37 }

```

10.5 Horário

```

1 // Descrição: Funções 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 }

```

10.6 Fibonacci

```

1 vector<int> memo(MAX, -1);
2
3 // Descrição: Função que retorna o  $n$ -ésimo termo da sequência de Fibonacci
   utilizando programação dinâmica.
4 // Complexidade:  $O(n)$  onde  $n$  é o termo desejado
5 int fibPD(int n) {
6     if (n <= 1) return n;
7     if (memo[n] != -1) return memo[n];

```



```

8     return memo[n] = fibPD(n - 1) + fibPD(n - 2);
9 }

```

10.7 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 }

```

10.8 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 }

```