Pedro Augusto Ulisses Andrade Lucas Andrade

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

Contents

1 Utils

1.1 Makefile

```
CXX = g++
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

q:
	cp temp.cpp $(f).cpp
	touch $(f).txt
	code $(f).txt
	code $(f).cpp
	clear
compile:
	g++ -g $(f).cpp $(CXXFLAGS) -o $(f)
exe:
	./$(f) < $(f).txt

runc: compile
runci: compile exe

clearexe:
	find . -maxdepth 1 -type f -executable -exec rm {} +
cleartxt:
	find . -type f -name "*.txt" -exec rm -f {} \;
clear: clearexe cleartxt
	clear
```

1.2 Limites

```
// LIMITES DE REPRESENTACAO DE DADOS

       tipo      | bits |          minimo .. maximo          | precisao decim.
-----------------+------+------------------------------------+----------------
char             |   8  |                0 .. 127            |        2
signed char      |   8  |             -128 .. 127            |        2
unsigned char    |   8  |                0 .. 255            |        2
short            |  16  |          -32.768 .. 32.767         |        4
unsigned short   |  16  |                0 .. 65.535         |        4
int              |  32  |       -2 x 10^9 .. 2 x 10^9        |        9
unsigned int     |  32  |                0 .. 4 x 10^9       |        9
int64_t          |  64  |     -9 x 10^18 .. 9 x 10^18        |       18
uint64_t         |  64  |                0 .. 18 x 10^18     |       19
float            |  32  |   1.2 x 10^-38 .. 3.4 x 10^38      |      6-9
double           |  64  |  2.2 x 10^-308 .. 1.8 x 10^308     |     15-17
long double      |  80  | 3.4 x 10^-4932 .. 1.1 x 10^4932    |     18-19
BigInt/Dec(java)      1 x 10^-2147483648 .. 1 x 10^2147483647 | 0

// LIMITES DE MEMORIA

1MB = 1,048,576 bool
1MB =   524,288 char
1MB =   262,144 int32_t
1MB =   131,072 int64_t
1MB =    65,536 float
1MB =    32,768 double
1MB =    16,384 long double
1MB =    16,384 BigInteger / BigDecimal

// ESTOURAR TEMPO

imput size    | complexidade para 1 s
--------------+---------------------
 [10,11]      | O(n!), O(n^6)
 [15,18]      | O(2^n * n^2)
 [18, 22]     | O(2^n * n)
 ... 100      | O(n^4)
 ... 400      | O(n^3)
 ... 2*10^3   | O(n^2 * log n)
 ... 5*10^4   | O(n^2)
 ... 10^5     | O(n^2 log n)
 ... 10^6     | O(n log n)
 ... 10^7     | O(n log log n)
 ... 10^8     | O(n), O(log n), O(1)


// FATORIAL

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

1.3 Mini Template Cpp

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

#define _ std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define all(a)          a.begin(), a.end()
#define int             long long int
#define double          long double
#define endl            "\n"
#define print_v(a)      for(auto x : a) cout << x << " "; cout << endl
#define f(i,s,e)        for(int i=s;i<e;i++)
#define rf(i,e,s)       for(int i=e-1;i>=s;i--)
#define dbg(x) cout << #x << " = " << x << endl;

void solve() {


}

int32_t main() { _

    int t = 1; // cin >> t;
    while (t--) {
        solve();
    }

    return 0;
}
```

1.4 Template Cpp

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

#define _ std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define all(a)        a.begin(), a.end()
#define int           long long int
#define double        long double
#define vi            vector<int>
#define pii           pair<int,int>
#define endl          "\n"
#define print_v(a)    for(auto x : a)cout<<x<<" ";cout<<endl
#define print_vp(a)   for(auto x : a)cout<<x.first<<" "<<x.second<< endl
#define f(i,s,e)      for(int i=s;i<e;i++)
#define rf(i,e,s)     for(int i=e-1;i>=s;i--)
#define CEIL(a, b)    ((a) + (b - 1))/b
#define TRUNC(x, n)   floor(x * pow(10, n))/pow(10, n)
#define ROUND(x, n)   round(x * pow(10, n))/pow(10, n)
#define dbg(x) cout << #x << " = " << x << " ";
#define dbgl(x) cout << #x << " = " << x << endl;

const int INF = 1e9;     // 2^31-1
const int LLINF = 4e18; // 2^63-1
const double EPS = 1e-9;
const int MAX = 1e6+10; // 10^6 + 10

void solve() {


}

int32_t main() { _

    clock_t z = clock();
    int t = 1; // cin >> t;
    while (t--) {
        solve();
    }
    cerr << fixed << "Run Time : " << ((double)(clock() - z) /
    CLOCKS_PER_SEC) << endl;
    return 0;
}
```

1.5 Files

```sh
#!/bin/bash

for c in {a..f}; do
    cp temp.cpp "$c.cpp"
    echo "$c" > "$c.txt"
    if [ "$c" = "$letter" ]; then
        break
    fi
done
```

1.6 Template Python

```python
import sys
import math
import bisect
from sys import stdin,stdout
from math import gcd,floor,sqrt,log
from collections import defaultdict as dd
from bisect import bisect_left as bl,bisect_right as br

sys.setrecursionlimit(100000000)

inp    =lambda: int(input())
strng  =lambda: input().strip()
jn     =lambda x,l: x.join(map(str,l))
strl   =lambda: list(input().strip())
mul    =lambda: map(int,input().strip().split())
mulf   =lambda: map(float,input().strip().split())
seq    =lambda: list(map(int,input().strip().split()))

ceil   =lambda x: int(x) if(x==int(x)) else int(x)+1
ceildiv=lambda x,d: x//d if(x%d==0) else x//d+1

flush  =lambda: stdout.flush()
stdstr =lambda: stdin.readline()
stdint =lambda: int(stdin.readline())
stdpr  =lambda x: stdout.write(str(x))

mod=1000000007

#main code

a = None
b = None
lista = None

def ident(*args):
    if len(args) == 1:
        return args[0]
    return args


def parsin(*, l=1, vpl=1, s=" "):
    if l == 1:
        if vpl == 1: return ident(input())
        else: return list(map(ident, input().split(s)))
    else:
        if vpl == 1: return [ident(input()) for _ in range(l)]
        else: return [list(map(ident, input().split(s))) for _ in range(l)
    ]


def solve():
    pass

# if __name__ == '__main__':
def main():
```

```
    st = clk()

    escolha = "in"
    #escolha = "num"

    match escolha:
        case "in":
            # êl infinitas linhas agrupadas de 2 em 2
            # pra infinitos valores em 1 linha pode armazenar em uma lista
            while True:
                global a, b
                try: a, b = input().split()
                except (EOFError): break #permite ler todas as linahs dentro do .txt
                except (ValueError): pass # consegue ler éat linhas em branco
                else:
                    a, b = int(a), int(b)
                solve()

        case "num":
            global lista
            # int l; cin >> l; while(l--){for(i=0; i<vpl; i++)}
            # retorna listas com inputs de cada linha
            # leia l linhas com vpl valores em cada uma delas
                # caseo seja mais de uma linha, retorna lista com listas de inputs
            lista = parsin(l=2, vpl=5)
            solve()

    sys.stderr.write(f"Run Time : {(clk() - st):.6f} seconds\n")

main()
```

2 Informações

2.1 Vector

```
// INICIALIZAR
vector<int> v (n); // n ócpias de 0
vector<int> v (n, v); // n ócpias de v

// PUSH_BACK
// Complexidade: O(1) amortizado (O(n) se realocar)
v.push_back(x);

// REMOVER
// Complexidade: O(n)
v.erase(v.begin() + i);

// INSERIR
// Complexidade: O(n)
v.insert(v.begin() + i, x);

// ORDENAR
// Complexidade: O(n log(n))
sort(v.begin(), v.end());
sort(all(v));

// BUSCA BINARIA
// Complexidade: O(log(n))
// Retorno: true se existe, false se ãno existe
binary_search(v.begin(), v.end(), x);

// FIND
// Complexidade: O(n)
// Retorno: iterador para o elemento, v.end() se ãno existe
find(v.begin(), v.end(), x);

// CONTAR
// Complexidade: O(n)
// Retorno: únmero de êocorrncias
count(v.begin(), v.end(), x);
```

2.2 Sort

```
vector<int> v;
    // Sort Crescente:
    sort(v.begin(), v.end());
    sort(all(v));

    // Sort Decrescente:
    sort(v.rbegin(), v.rend());
    sort(all(v), greater<int>());

    // Sort por uma çãfuno:
    auto cmp = [](int a, int b) { return a > b; }; // { 2, 3, 1 } -> { 3, 2, 1 }
    auto cmp = [](int a, int b) { return a < b; }; // { 2, 3, 1 } -> { 1, 2, 3 }
    sort(v.begin(), v.end(), cmp);
    sort(all(v), cmp);

    // Sort por uma çãfuno (çãcomparao de pares):
    auto cmp = [](pair<int, int> a, pair<int, int> b) { return a.second > b.second; };
```

2.3 Priority Queue

```
// HEAP CRESCENTE {5,4,3,2,1}
priority_queue<int> pq; // max heap
    // maior elemento:
    pq.top();

// HEAP DECRESCENTE {1,2,3,4,5}
priority_queue<int, vector<int>, greater<int>> pq; // min heap
    // menor elemento:
    pq.top();

// REMOVER ELEMENTO
// Complexidade: O(n)
// Retorno: true se existe, false se ãno existe
```

```
pq.remove(x);

// INSERIR ELEMENTO
// Complexidade: O(log(n))
pq.push(x);

// REMOVER TOP
// Complexidade: O(log(n))
pq.pop();

// TAMANHO
// Complexidade: O(1)
pq.size();

// VAZIO
// Complexidade: O(1)
pq.empty();

// LIMPAR
// Complexidade: O(n)
pq.clear();

// ITERAR
// Complexidade: O(n)
for (auto x : pq) {}

// çãOrdenao por çãfuno customizada passada por parametro ao criar a pq
// Complexidade: O(n log(n))
auto cmp = [](int a, int b) { return a > b; };
priority_queue<int, vector<int>, decltype(cmp)> pq(cmp);
```

2.4 String

```
// INICIALIZAR
string s; // string vazia
string s (n, c); // n ócpias de c
string s (s); // ócpia de s
string s (s, i, n); // ócpia de s[i..i+n-1]

// SUBSTRING
// Complexidade: O(n)
s.substr(i, n); // substring de s[i..i+n-1]
s.substr(i, j - i + 1); // substring de s[i..j]

// TAMANHO
// Complexidade: O(1)
s.size(); // tamanho da string
s.empty(); // true se vazia, false se ãno vazia

// MODIFICAR
// Complexidade: O(n)
s.push_back(c); // adiciona c no final
s.pop_back(); // remove o último
s += t; // concatena t no final
s.insert(i, t); // insere t a partir da çãposio i
s.erase(i, n); // remove n caracteres a partir da çãposio i
s.replace(i, n, t); // substitui n caracteres a partir da çãposio i por t
s.swap(t); // troca o úcontedo com t

// COMPARAR
// Complexidade: O(n)
s == t; // igualdade
s != t; // çdiferena
s < t; // menor que
s > t; // maior que
s <= t; // menor ou igual
s >= t; // maior ou igual

// BUSCA
// Complexidade: O(n)
s.find(t); // çãposio da primeira êocorrncia de t, ou string::npos se ãno existe
s.rfind(t); // çãposio da última êocorrncia de t, ou string::npos se ãno existe
s.find_first_of(t); // çãposio da primeira êocorrncia de um caractere de t, ou string::npos se ãno existe
s.find_last_of(t); // çãposio da última êocorrncia de um caractere de t, ou string::npos se ãno existe
s.find_first_not_of(t); // çãposio do primeiro caractere que ãno áest em t, ou string::npos se ãno existe
s.find_last_not_of(t); // çãposio do último caractere que ãno áest em t, ou string::npos se ãno existe

// SUBSTITUIR
// Complexidade: O(n)
s.replace(i, n, t); // substitui n caracteres a partir da çãposio i por t
s.replace(s.begin() + i, s.begin() + i + n, t.begin(), t.end()); // substitui n caracteres a partir da çãposio i por t
s.replace(s.begin() + i, s.begin() + i + n, t); // substitui n caracteres a partir da çãposio i por t
s.replace(s.begin() + i, s.begin() + i + n, n, c); // substitui n caracteres a partir da çãposio i por n ócpias de c
```

3 Combinatoria

3.1 Arranjo Simples

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

3.2 Combinacao Com Repeticao

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

3.3 Combinacao Simples

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

3.4 Permutacao Circular

```
// Permutacao objetos em posicao simetrica em um circulo

int permutacaoCircular(int n) {
    return fact(n - 1);
}
```

3.5 @ Tabela

```
// Sequencia de p elementos de um total de n

ORDEM \ REPETIC |           COM             |         SEM
----------------+---------------------------+--------------------
IMPORTA         | ARRANJO COM REPETICAO     | ARRANJO SIMPLES
NAO             | COMBINACAO COM REPETICAO  | COMBINACAO SIMPLES
```

3.6 Permutacao Com Repeticao

```
// Trocar elementos de lugar quando ha termos repetidos (ANAGRAMA)
int permutacaoComRepeticao(string s) {
    int n = s.size();
    int ans = fact(n);
    map<char, int> freq;
    for (char c : s) {
        freq[c]++;
    }
    for (auto [c, f] : freq) {
        ans /= fact(f);
    }
    return ans;
}
```

3.7 @ Factorial

```
int factdp[20];

int fact(int n) {
    if (n < 2) return 1;
    if (factdp[n] != 0) return factdp[n];
    return factdp[n] = n * fact(n - 1);
}
```

3.8 Permutacao Simples

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

int permutacaoSimples(int n) {
    return fact(n);
}
```

3.9 Arranjo Com Repeticao

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

4 Estruturas

4.1 Bittree

```
// C++ code to demonstrate operations of Binary Index Tree
#include <iostream>

using namespace std;

/*       n --> No. of elements present in input array.
    BITree[0..n] --> Array that represents Binary Indexed Tree.
    arr[0..n-1] --> Input array for which prefix sum is evaluated. */

// Returns sum of arr[0..index]. This function assumes
// that the array is preprocessed and partial sums of
// array elements are stored in BITree[].
int getSum(int BITree[], int index)
{
    int sum = 0; // Initialize result

    // index in BITree[] is 1 more than the index in arr[]
    index = index + 1;

    // Traverse ancestors of BITree[index]
    while (index>0)
    {
        // Add current element of BITree to sum
        sum += BITree[index];

        // Move index to parent node in getSum View
        index -= index & (-index);
    }
    return sum;
}

// Updates a node in Binary Index Tree (BITree) at given index
// in BITree. The given value 'val' is added to BITree[i] and
// all of its ancestors in tree.
void updateBIT(int BITree[], int n, int index, int val)
{
    // index in BITree[] is 1 more than the index in arr[]
    index = index + 1;

    // Traverse all ancestors and add 'val'
    while (index <= n)
    {
    // Add 'val' to current node of BI Tree
    BITree[index] += val;

    // Update index to that of parent in update View
    index += index & (-index);
    }
```

```
}

// Constructs and returns a Binary Indexed Tree for given
// array of size n.
int *constructBITree(int arr[], int n)
{
    // Create and initialize BITree[] as 0
    int *BITree = new int[n+1];
    for (int i=1; i<=n; i++)
        BITree[i] = 0;

    // Store the actual values in BITree[] using update()
    for (int i=0; i<n; i++)
        updateBIT(BITree, n, i, arr[i]);

    // Uncomment below lines to see contents of BITree[]
    //for (int i=1; i<=n; i++)
    //    cout << BITree[i] << " ";

    return BITree;
}


// Driver program to test above functions
int main()
{
    int freq[] = {2, 1, 1, 3, 2, 3, 4, 5, 6, 7, 8, 9};
    int n = sizeof(freq)/sizeof(freq[0]);
    int *BITree = constructBITree(freq, n);
    cout << "Sum of elements in arr[0..5] is "
        << getSum(BITree, 5);

    // Let use test the update operation
    freq[3] += 6;
    updateBIT(BITree, n, 3, 6); //Update BIT for above change in arr[]

    cout << "\nSum of elements in arr[0..5] after update is "
        << getSum(BITree, 5);

    return 0;
}
```

4.2 Sparse Table Disjunta

```
// Sparse Table Disjunta
//
// Resolve qualquer operacao associativa
// MAX2 = log(MAX)
//
// Complexidades:
// build - O(n log(n))
// query - O(1)

namespace sparse {
    int m[MAX2][2*MAX], n, v[2*MAX];
    int op(int a, int b) { return min(a, b); }
    void build(int n2, int* v2) {
        n = n2;
        for (int i = 0; i < n; i++) v[i] = v2[i];
        while (n&(n-1)) n++;
        for (int j = 0; (1<<j) < n; j++) {
            int len = 1<<j;
            for (int c = len; c < n; c += 2*len) {
                m[j][c] = v[c], m[j][c-1] = v[c-1];
                for (int i = c+1; i <  c+len; i++) m[j][i] = op(m[j][i-1],
     v[i]);
                for (int i = c-2; i >= c-len; i--) m[j][i] = op(v[i], m[j
    ][i+1]);
            }
        }
    }
    int query(int l, int r) {
        if (l == r) return v[l];
        int j = __builtin_clz(1) - __builtin_clz(l^r);
        return op(m[j][l], m[j][r]);
    }
}
```

4.3 Seg Tree

```
// SegTree
//
// Query: soma do range [a, b]
// Update: soma x em cada elemento do range [a, b]
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))
namespace seg {

    int seg[4*MAX];
    int n, *v;

    int op(int a, int b) { return a + b; }

    int build(int p=1, int l=0, int r=n-1) {
        if (l == r) return seg[p] = v[l];
        int m = (l+r)/2;
        return seg[p] = op(build(2*p, l, m), build(2*p+1, m+1, r));
    }

    void build(int n2, int* v2) {
        n = n2, v = v2;
        build();
    }

    int query(int a, int b, int p=1, int l=0, int r=n-1) {
        if (a <= l and r <= b) return seg[p];
        if (b < l or r < a) return 0;
        int m = (l+r)/2;
        return op(query(a, b, 2*p, l, m), query(a, b, 2*p+1, m+1, r));
    }

```

```
    int update(int a, int b, int x, int p=1, int l=0, int r=n-1) {
        if (a <= l and r <= b) return seg[p];
        if (b < l or r < a) return seg[p];
        int m = (l+r)/2;
        return seg[p] = op(update(a, b, x, 2*p, l, m), update(a, b, x, 2*p+1, m+1, r));
    }
};
```

5 Grafos

5.1 Bfs Matriz

```
// Description: BFS para uma matriz (n x m)
// Complexidade: O(n * m)

vector<vi> mat;
vector<vector<bool>> vis;
vector<pair<int,int>> mov = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};
int l, c;

bool valid(int x, int y) {
    return (0 <= x and x < l and 0 <= y and y < c and !vis[x][y] /*and mat[x][y]*/);
}

void bfs(int i, int j) {

    queue<pair<int,int>> q; q.push({i, j});

    while(!q.empty()) {

        auto [u, v] = q.front(); q.pop();
        vis[u][v] = true;

        for(auto [x, y]: mov) {
            if(valid(u+x, v+y)) {
                q.push({u+x,v+y});
                vis[u+x][v+y] = true;
            }
        }
    }
}

int main() {
    cin >> l >> c;
    mat.resize(l, vi(c));
    vis.resize(l, vector<bool>(c, false));
    /*preenche matriz*/
    bfs(0,0);
}
```

5.2 Bfs

```
// BFS com informacoes adicionais sobre a distancia e o pai de cada vertice
// Complexidade: O(V + E), onde V eh o numero de vertices e E o numero de areqas

int n; // n = numero de vertices
vector<bool> vis;
vector<int> d, p;
vector<vector<int>> adj; // liqa de adjacencia

void bfs(int s) {

    vis.resize(n, false);
    d.resize(n, -1);
    p.resize(n, -1);

    queue<int> q; q.push(s);
    vis[s] = true, d[s] = 0, p[s] = -1;

    while (!q.empty()) {
        int v = q.front(); q.pop(); visited[v] = true;

        for (int u : adj[v]) {
            if (!vis[u]) {
                vis[u] = true;
                q.push(u);
                d[u] = d[v] + 1;
                p[u] = v;
            }
        }
    }
}
```

5.3 Dijkstra

```
// Encontra o menor caminho de um évrtice s para todos os outros évrtices do grafo.
//Complexidade: O((V + E)logV)

int n;
vector<vector<pair<int, int>>> adj; // adj[a] = [{b, w}]
vector<int> dist, parent; /*dist[a] = dist(source -> a)*/
vector<bool> vis;

void dijkstra(int s) {

    dist.resize(n+1, LINF-10);
    vis.resize(n+1, false);
    parent.resize(n+1, -1);
    dist[s] = 0;

    priority_queue<pair<int, int>> q;
    q.push({0, s});

    while (!q.empty()) {
        int a = q.top().second; q.pop();

        if (vis[a]) continue;
        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 call() {
45
46      adj.resize(n); /*n = nodes*/
47
48      f(i,0,n) {
49          int a, b, w; cin >> a >> b >> w;
50          adj[a].push_back({b, w});
51          adj[b].push_back({a, w});
52      }
53
54      dijkstra(0);
55  }
```

5.4 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
       and !visited[x][y];
11 }
12
13 bool condicaoSaida(const int& x, const int& y) {
14     return labirinto[x][y] == 2;
15 }
16
17 bool search(const int& x, const int& y) {
18
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 }
```

5.5 Successor Graph

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

5.6 Floyd Warshall

```
1  // Floyd-Warshall
2  //
3  // encontra o menor caminho entre todo
4  // par de vertices e detecta ciclo negativo
5  // returna 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  //
```

```cpp
// O(n^3)

int n;
int d[MAX][MAX];

bool floyd_warshall() {
    for (int k = 0; k < n; k++)
    for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

    for (int i = 0; i < n; i++)
        if (d[i][i] < 0) return 1;

    return 0;
}
```

5.7 Kosaraju

```cpp
// Kosaraju
//
// Complexidade: O(n + m)

int n;
vector<int> g[MAX], gi[MAX]; // grafo invertido
int vis[MAX], comp[MAX]; // componente de cada évrtice
stack<int> S;

void dfs(int k) {
    vis[k] = 1;
    for (int i = 0; i < (int) g[k].size(); i++)
        if (!vis[g[k][i]]) dfs(g[k][i]);

    S.push(k);
}

// Descricao: Calcula as componentes fortemente conexas de um grafo
void scc(int k, int c) {
    vis[k] = 1;
    comp[k] = c; // componente de k eh c
    for (int i = 0; i < (int) gi[k].size(); i++)
        if (!vis[gi[k][i]]) scc(gi[k][i], c);
}

void kosaraju() {

    memset(vis, 0, sizeof(vis));
    for(int i=0; i<n; i++) if(!vis[i]) dfs(i);
    memset(vis, 0, sizeof(vis));

    while (S.size()) {
        int u = S.top(); S.pop();
        if (!vis[u]) scc(u, u);
    }
}
```

5.8 Euler Tree

```cpp
vector<vector<int>> adj(MAX);
vector<int> vis(MAX, 0);
vector<int> euTree(MAX);

// Eulerian path in tree
// Complexity: O(V)
void eulerTree(int u, int &index) {
    vis[u] = 1;
    euTree[index++] = u;
    for (auto it : adj[u]) {
        if (!vis[it]) {
            eulerTree(it, index);
            euTree[index++] = u;
        }
    }
}

int main() {

    f(i,0,n-1) {
        int a, b; cin >> a >> b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    int index = 0;
    eulerTree(1, index);
}
```

5.9 Kruskal

```cpp
// Kruskal
//
// Gera e retorna uma AGM e seu custo total a partir do vetor de arestas (edg) do grafo
//
// Complexidade: O(ElogE) onde E eh o numero de arestas
//
// O(m log(m) + m a(m)) = O(m log(m))

vector<int> id, sz;

int find(int a){ // O(a(N)) amortizado
    return id[a] = (id[a] == a ? a : find(id[a]));
}

void uni(int a, int b) { // O(a(N)) amortizado
    a = find(a), b = find(b);
    if(a == b) return;

    if(sz[a] > sz[b]) swap(a,b);
    id[a] = b, sz[b] += sz[a];
}

```

```cpp
pair<int, vector<tuple<int, int, int>>> kruskal(vector<tuple<int, int, int>>& edg) {

    sort(edg.begin(), edg.end());

    int cost = 0;
    vector<tuple<int, int, int>> mst; // opcional
    for (auto [w,x,y] : edg) if (find(x) != find(y)) {
        mst.emplace_back(w, x, y); // opcional
        cost += w;
        uni(x,y);
    }
    return {cost, mst};
}

int main() {

    int n/*nodes*/, ed/*edges*/;

    id.resize(n); iota(all(id), 0);
    sz.resize(n, -1);
    vector<tuple<int, int, int>> edg;

    f(i,0,ed) {
        int a, b, w; cin >> a >> b >> w;
        edg.push_back({w, a, b});
    }

    auto [cost, mst] = kruskal(edg);
}
```

5.10 Dfs

```cpp

vector<int> adj[MAXN];
int visited[MAXN];


// DFS com informacoes adicionais sobre o pai de cada vertice
// Complexidade: O(V + E), onde V eh o numero de vertices e E o numero de areqas
void dfs(int p) {
    memset(visited, 0, sizeof visited);
    stack<int> st;
    st.push(p);

    while (!st.empty()) {
        int curr = st.top();
        st.pop();
        if (visited[curr]==1)continue;
        visited[curr]=1;
        // process current node here

        for (auto i : adj[curr]) {
            st.push(i);
        }
    }
}

// DFS com informacoes adicionais sobre o pai de cada vertice
// Complexidade: O(V + E), onde V eh o numero de vertices e E o numero de areqas
void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
}
```

6 Matematica

6.1 N Fibonacci

```cpp
int dp[MAX];

int fibonacciDP(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    if (dp[n] != -1) return dp[n];
    return dp[n] = fibonacciDP(n-1) + fibonacciDP(n-2);
}

int nFibonacci(int minus, int times, int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    if (dp[n] != -1) return dp[n];
    int aux = 0;
    for(int i=0; i<times; i++) {
        aux += nFibonacci(minus, times, n-minus);
    }
}
```

6.2 Mdc Multiplo

```cpp
// Description: Calcula o MDC de um vetor de inteiros.
// Complexidade: O(nlogn) onde n eh o tamanho do vetor
int mdc_many(vector<int> arr) {
   int result = arr[0];

   for (int& num : arr) {
       result = mdc(num, result);

       if(result == 1) return 1;
   }
   return result;
}
```

6.3 Factorial

```cpp
unordered_map<int, int> memo;

```

```cpp
// Factorial
// Complexidade: O(n), onde n eh o numero a ser fatorado
int factorial(int n) {
    if (n == 0 || n == 1)  return 1;
    if (memo.find(n) != memo.end()) return memo[n];
    return memo[n] = n * factorial(n - 1);
}
```

6.4 Divisores

```cpp
// Calcudla os divisores de c, sem incluir c, sem ser fatorado
// Complexidade: O(sqrt(c))
set<int> calculaDivisores(int c) {
    int lim = sqrt(c);
    set<int> divisors;

    for(int i = 1; i <= lim; i++) {
        if (c % i == 0) {
            if(c/i != i)
                divisors.insert(c/i);
            divisors.insert(i);
        }
    }

    return divisors;
}
```

6.5 Operacoes Binarias

```cpp
// Checa se um únmero é ímpar
#define isOdd(a) (a & 1)

// Multiplica a por 2^b
#define mult2ToTheN(a, b) (a << b)

// Divide a por 2^b
#define div2ToTheN(a, b) (a >> b)

// 2^b
#define twoToTheN(b) (1 << b)

// Retorna -1 se n < 0, 0 se n = 0 e 1 se n > 0
#define sig(a) ((a >> 31) | -(a >> 31))

// Retorna o maior divisor (potencia de 2) de n
#define greatest2powDivisorOf(a) (a & -a)
```

6.6 Mmc Multiplo

```cpp
// Description: Calcula o mmc de um vetor de inteiros.
// Complexidade: O(nlogn) onde n eh o tamanho do vetor
int mmc_many(vector<int> arr)
{
    int result = arr[0];

    for (int &num : arr)
        result = (num * result / mdc(num, result));
    return result;
}
```

6.7 Fast Exponentiation

```cpp
const int mod = 1e9 + 7;

// Fast Exponentiation: retorna a^b % mod
// Quando usar: quando precisar calcular a^b % mod
int fexp(int a, int b)
{
    int ans = 1;
    while (b)
    {
        if (b & 1)
            ans = ans * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return ans;
}
```

6.8 Fatoracao

```cpp
// Fatora um únmero em seus fatores primos
// Complexidade: O(sqrt(n))
map<int, int> factorize(int n) {
    map<int, int> factorsOfN;
    int lowestPrimeFactorOfN = 2;

    while (n != 1) {
        lowestPrimeFactorOfN = lowestPrimeFactor(n, lowestPrimeFactorOfN);
        factorsOfN[lowestPrimeFactorOfN] = 1;
        n /= lowestPrimeFactorOfN;
        while (not (n % lowestPrimeFactorOfN)) {
            factorsOfN[lowestPrimeFactorOfN]++;
            n /= lowestPrimeFactorOfN;
        }
    }

    return factorsOfN;
}
```

6.9 Contar Quanti Solucoes Eq 2 Variaveis

```cpp
// Description: Dada uma equacao de 2 variaveis, calcudla quantas
    combinacoes {x,y} inteiras que resolvem essa equacao

// y = numerador / denominador
int numerador(int x) { return c - x; } // expressao do numerador
int denominador(int x) { return 2 * x + 1; } // expressao do denominador

int count2VariableIntegerEquationAnswers() {

    unordered_set<pair<int,int>, PairHash> ans; int lim = sqrt(c);
```

```
    for(int i=1; i<= lim; i++) {
        if (numerador(i) % denominador(i) == 0) {
            int x = i, y = numerador(i) / denominador(i);
            if(!ans.count({x,y}) and !ans.count({y,x}))
                ans.insert({x,y});
        }
    }

    return ans.size();
}
```

6.10 Conversao De Bases

```
// Converter um decimal (10) para base n [2, 8, 10, 16]
// Complexidade: O(log n)
char charForDigit(uint16_t digit) {
    if (digit > 9) return digit + 87;
    return digit + 48;
}

string decimalToBase(unsigned n, unsigned base = 10) {
    if (not n) return "0";
    stringstream ss;
    for (unsigned i = n; i > 0; i /= base) {
        ss << charForDigit(i % base);
    }
    string s = ss.str();
    reverse(s.begin(), s.end());
    return s;
}

// Converter um numero de base [2, 8, 10, 16] para decimal (10)
// Complexidade: O(n)
uint16_t intForDigit(char digit) {
    uint16_t intDigit = digit - 48;
    if (intDigit > 9) return digit - 87;
    return intDigit;
}

int baseToDecimal(const string& n, unsigned base = 10) {
    int result = 0;
    uint64_t basePow =1;
    for (auto it = n.rbegin(); it != n.rend(); ++it, basePow *= base)
        result += intForDigit(*it) * basePow;
    return result;
}
```

6.11 Sieve

```
// Crivo de óEratstenes para gerar primos éat um limite 'lim'
// Complexidade: O(n log log n), onde n é o limite
const int ms = 1e6 + 5;
bool notPrime[ms];   // notPrime[i] é verdadeiro se i ãno é um únmero primo
int primes[ms], qnt; // primes[] armazena os únmeros primos e qnt é a quantidade de primos encontrados

void sieve(int lim)
{
  primes[qnt++] = 1; // adiciona 1 como um únmero primo se ele for ávlido no problema
  for (int i = 2; i <= lim; i++)
  {
    if (notPrime[i])
      continue;                          // se i ãno é primo, pula
    primes[qnt++] = i;                   // i é primo, adiciona em primes []
    for (int j = i + i; j <= lim; j += i) // marca todos os úmltiplos de i como ãno primos
      notPrime[j] = true;
  }
}
```

6.12 Mdc

```
// Description: Calcula o mdc de dois numeros inteiros.
// Complexidade: O(logn) onde n eh o maior numero
unsigned mdc(unsigned a, unsigned b) {
    for (unsigned r = a % b; r; a = b, b = r, r = a % b);
    return b;
}
```

6.13 Fatorial Grande

```
static BigInteger[] dp = new BigInteger[1000000];

public static BigInteger factorialDP(BigInteger n) {
    dp[0] = BigInteger.ONE;
    for (int i = 1; i <= n.intValue(); i++) {
        dp[i] = dp[i - 1].multiply(BigInteger.valueOf(i));
    }
    return dp[n.intValue()];
}
```

6.14 Sieve Linear

```
// Sieve de Eratosthenes com linear sieve
// Encontra todos os únmeros primos no intervalo [2, N]
// Complexidade: O(N)

vector<int> sieve(const int N) {

    vector<int> lp(N + 1); // lp[i] = menor fator primo de i
    vector<int> pr;

    for (int i = 2; i <= N; ++i) {
        if (lp[i] == 0) {
            lp[i] = i;
            pr.push_back(i);
        }
        for (int j = 0; i * pr[j] <= N; ++j) {
            lp[i * pr[j]] = pr[j];
```

```
            if (pr[j] == lp[i])
                break;
        }
    }

    return pr;
}
```

6.15 Primo

```
// Descricao: Funcao que verifica se um numero n eh primo.
// Complexidade: O(sqrt(n))
int lowestPrimeFactor(int n, int startPrime = 2) {
    if (startPrime <= 3) {
        if (not (n & 1))
            return 2;
        if (not (n % 3))
            return 3;
        startPrime = 5;
    }

    for (int i = startPrime; i * i <= n; i += (i + 1) % 6 ? 4 : 2)
        if (not (n % i))
            return i;
    return n;
}

bool isPrime(int n) {
    return n > 1 and lowestPrimeFactor(n) == n;
}
```

6.16 Miller Rabin

```
// Teste de primalidade de Miller-Rabin
// Complexidade: O(k*log^3(n)), onde k eh o numero de testes e n eh o numero a ser testado
// Descicao: Testa se um numero eh primo com uma probabilidade de erro de 1/4^k

int mul(int a, int b, int m) {
    int ret = a*b - int((long double)1/m*a*b+0.5)*m;
    return ret < 0 ? ret+m : ret;
}

int pow(int x, int y, int m) {
    if (!y) return 1;
    int ans = pow(mul(x, x, m), y/2, m);
    return y%2 ? mul(x, ans, m) : ans;
}

bool prime(int n) {
    if (n < 2) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0) return 0;
    int r = __builtin_ctzint(n - 1), d = n >> r;

    // com esses primos, o teste funciona garantido para n <= 2^64
    // funciona para n <= 3*10^24 com os primos ate 41
    for (int a : {2, 325, 9375, 28178, 450775, 9780504, 795265022}) {
        int x = pow(a, d, n);
        if (x == 1 or x == n - 1 or a % n == 0) continue;

        for (int j = 0; j < r - 1; j++) {
            x = mul(x, x, n);
            if (x == n - 1) break;
        }
        if (x != n - 1) return 0;
    }
    return 1;
}
```

6.17 Decimal Para Fracao

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

6.18 Numeros Grandes

```
public static void BbigInteger() {

    BigInteger a = BigInteger.valueOf(1000000000);
                a = new BigInteger("10000000000");

    // çõOperaes com inteiros grandes
    BigInteger  arit = a.add(a);
                arit = a.subtract(a);
                arit = a.multiply(a);
                arit = a.divide(a);
                arit = a.mod(a);

    // çãComparao
    boolean bool = a.equals(a);
            bool = a.compareTo(a) > 0;
            bool = a.compareTo(a) < 0;
            bool = a.compareTo(a) >= 0;
            bool = a.compareTo(a) <= 0;

    // ãConverso para string
    String m = a.toString();

    // ãConverso para inteiro
    int     _int = a.intValue();
    long   _long = a.longValue();
    double _doub = a.doubleValue();

    // êPotncia
```

```
    BigInteger _pot = a.pow(10);
    BigInteger _sqr = a.sqrt();

}

public static void BigDecimal() {

    BigDecimal a = new BigDecimal("10000000000");
                a = new BigDecimal("10000000000.0000000000");
                a = BigDecimal.valueOf(1000000000, 10);


    // çõOperaes com reais grandes
    BigDecimal arit = a.add(a);
                arit = a.subtract(a);
                arit = a.multiply(a);
                arit = a.divide(a);
                arit = a.remainder(a);

    // çãComparao
    boolean bool = a.equals(a);
            bool = a.compareTo(a) > 0;
            bool = a.compareTo(a) < 0;
            bool = a.compareTo(a) >= 0;
            bool = a.compareTo(a) <= 0;

    // ãConverso para string
    String m = a.toString();

    // ãConverso para inteiro
    int     _int = a.intValue();
    long   _long = a.longValue();
    double _doub = a.doubleValue();

    // êPotncia
    BigDecimal _pot = a.pow(10);
}
```

6.19 Dois Primos Somam Num

```
// Description: Verifica se dois numeros primos somam um numero n.
// Complexity: O(sqrt(n))
bool twoNumsSumPrime(int n) {

    if(n % 2 == 0) return true;
    return isPrime(n-2);
}
```

6.20 Numeros Grandes

```
// Descricao: Implementacao de operacoes com numeros grandes
// Complexidade: O(n * m), n = tamanho do primeiro numero, m = tamanho do segundo numero

void normalize(vector<int>& num) {
    int carry = 0;
    for (int i = 0; i < num.size(); ++i) {
        num[i] += carry;
        carry = num[i] / 10;
        num[i] %= 10;
    }

    while (carry > 0) {
        num.push_back(carry % 10);
        carry /= 10;
    }
}


pair<int, vector<int>> bigSum(const pair<int, vector<int>>& a, const pair<int, vector<int>>& b) {
    if (a.first == b.first) {
        vector<int> result(max(a.second.size(), b.second.size()), 0);
        transform(a.second.begin(), a.second.end(), b.second.begin(), result.begin(), plus<int>());
        normalize(result);
        return {a.first, result};
    } else {
        vector<int> result(max(a.second.size(), b.second.size()), 0);
        transform(a.second.begin(), a.second.end(), b.second.begin(), result.begin(), minus<int>());
        normalize(result);
        return {a.first, result};
    }
}

pair<int, vector<int>> bigSub(const pair<int, vector<int>>& a, const pair<int, vector<int>>& b) {
    return bigSum(a, {-b.first, b.second});
}

pair<int, vector<int>> bigMult(const pair<int, vector<int>>& a, const pair<int, vector<int>>& b) {
    vector<int> result(a.second.size() + b.second.size(), 0);

    for (int i = 0; i < a.second.size(); ++i) {
        for (int j = 0; j < b.second.size(); ++j) {
            result[i + j] += a.second[i] * b.second[j];
        }
    }

    normalize(result);
    return {a.first * b.first, result};
}


void printNumber(const pair<int, vector<int>>& num) {
    if (num.first == -1) {
        cout << '-';
    }

    for (auto it = num.second.rbegin(); it != num.second.rend(); ++it) {
        cout << *it;
```

```
    }
    cout << endl;
}

int main() {

    pair<int, vector<int>> num1 = {1, {1, 2, 3}};  // Representing +321
    pair<int, vector<int>> num2 = {-1, {4, 5, 6}};  // Representing -654

    cout << "Sum: "; printNumber(bigSum(num1, num2));
    cout << "Difference: "; printNumber(bigSub(num1, num2));
    cout << "Product: "; printNumber(bigMult(num1, num2));

}
```

6.21 Mmc

```
// Description: Calcula o mmc de dois únmeros inteiros.
// Complexidade: O(logn) onde n eh o maior numero
unsigned mmc(unsigned a, unsigned b) {
    return a / mdc(a, b) * b;
}
```

6.22 Tabela Verdade

```
// Gerar tabela verdade de uma ãexpresso booleana
// Complexidade: O(2^n)

vector<vector<int>> tabelaVerdade;
int indexTabela = 0;

void backtracking(int posicao, vector<int>& conj_bool) {

    if(posicao == conj_bool.size()) { // Se chegou ao fim da BST
        for(size_t i=0; i<conj_bool.size(); i++) {
            tabelaVerdade[indexTabela].push_back(conj_bool[i]);
        }
        indexTabela++;

    } else {
        conj_bool[posicao] = 1;
        backtracking(posicao+1,conj_bool);
        conj_bool[posicao] = 0;
        backtracking(posicao+1,conj_bool);
    }
}

int main() {

    int n = 3;

    vector<int> linhaBool (n, false);
    tabelaVerdade.resize(pow(2,n));

    backtracking(0,linhaBool);
}
```

7 Matriz

7.1 Maior Retangulo Binario Em Matriz

```
// Description: Encontra o maior âretngulo ábinrio em uma matriz.
// Time: O(n*m)
// Space: O(n*m)
tuple<int, int, int> maximalRectangle(const vector<vector<int>>& mat) {
    int r = mat.size();
    if(r == 0) return make_tuple(0, 0, 0);
    int c = mat[0].size();

    vector<vector<int>> dp(r+1, vector<int>(c));

    int mx = 0;
    int area = 0, height = 0, length = 0;
    for(int i=1; i<r; ++i) {
        int leftBound = -1;
        stack<int> st;
        vector<int> left(c);

        for(int j=0; j<c; ++j) {
            if(mat[i][j] == 1) {
                mat[i][j] = 1+mat[i-1][j];
                while(!st.empty() and mat[i][st.top()] >= mat[i][j])
                    st.pop();

                int val = leftBound;
                if(!st.empty())
                    val = max(val, st.top());

                left[j] = val;
            } else {
                leftBound = j;
                left[j] = 0;
            }
            st.push(j);
        }
        while(!st.empty()) st.pop();

        int rightBound = c;
        for(int j=c-1; j>=0; j--) {
            if(mat[i][j] != 0) {

                while(!st.empty() and mat[i][st.top()] >= mat[i][j])
                    st.pop();

                int val = rightBound;
                if(!st.empty())
                    val = min(val, st.top());

                dp[i][j] = (mat[i][j]+1) * (((val-1)-(left[j]+1)+1)+1);
                if (dp[i][j] > mx) {
                    mx = dp[i][j];
                    area = mx;
                    height = mat[i][j];
```

```
                    length = (val-1)-(left[j]+1)+1;
                }
                st.push(j);
            } else {
                dp[i][j] = 0;
                rightBound = j;
            }
        }
    }

    return make_tuple(area, height, length);
}
```

8 Strings

8.1 Ocorrencias

```
// Description: çãFuno que retorna um vetor com as çõposies de todas as êocorrncias de uma substring em uma string.
// Complexidade: O(n * m) onde n é o tamanho da string e m é o tamanho da substring.
vector<int> ocorrencias(string str,string sub){
    vector<int> ret;
    int index = str.find(sub);
    while(index!=-1){
        ret.push_back(index);
        index = str.find(sub,index+1);
    }

    return ret;
}
```

8.2 Palindromo

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

8.3 Split Cria

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

8.4 Remove Acento

```
// Descricao: Funcao que remove acentos de uma string.
// Complexidade: O(n * m) onde n eh o tamanho da string e m eh o tamanho do alfabeto com acento.
string removeAcentro(string str) {

    string comAcento = "áéíóúâêôãõà";
    string semAcento = "aeiouaeoaoa";

    for(int i = 0; i < str.size(); i++){
        for(int j = 0; j < comAcento.size(); j++){
            if(str[i] == comAcento[j]){
                str[i] = semAcento[j];
                break;
            }
        }
    }

    return str;
}
```

8.5 Permutacao

```
// Funcao para gerar todas as permutacoes de uma string
// Complexidade: O(n!)

void permute(string& s, int l, int r) {
    if (l == r)
        permutacoes.push_back(s);
    else {
        for (int i = l; i <= r; i++) {
            swap(s[l], s[i]);
            permute(s, l+1, r);
            swap(s[l], s[i]);
        }
    }
}

int main() {

    string str = "ABC";
    int n = str.length();
    permute(str, 0, n-1);
}
```

8.6 Chaves Colchetes Parenteses

```
# Descricao: Funcao que verifica se uma string contendo chaves, colchetes e parenteses esta balanceada.
```

```python
def balanced(string) -> bool:
    stack = []

    for i in string:
        if i in '([{': stack.append(i)

        elif i in ')]}':
            if (not stack) or ((stack[-1],i) not in [('(',')'), ('[',']'), ('{','}')]):
                return False
            else:
                stack.pop()

    return not stack
```

8.7 Lower Iupper

```cpp
// Description: çãFuno que transforma uma string em lowercase.
// Complexidade: O(n) onde n é o tamanho da string.
string to_lower(string a) {
    for (int i=0;i<(int)a.size();++i)
       if (a[i]>='A' && a[i]<='Z')
          a[i]+='a'-'A';
    return a;
}

// para checar se é lowercase: islower(c);

// Description: çãFuno que transforma uma string em uppercase.
// Complexidade: O(n) onde n é o tamanho da string.
string to_upper(string a) {
    for (int i=0;i<(int)a.size();++i)
       if (a[i]>='a' && a[i]<='z')
          a[i]-='a'-'A';
    return a;
}

// para checar se e uppercase: isupper(c);
```

8.8 Lexicograficamente Minima

```cpp
// Descricao: Retorna a menor rotacao lexicografica de uma string.
// Complexidade: O(n * log(n)) onde n eh o tamanho da string
string minLexRotation(string str) {
    int n = str.length();

    string arr[n], concat = str + str;

    for (int i = 0; i < n; i++)
        arr[i] = concat.substr(i, n);

    sort(arr, arr+n);

    return arr[0];
}
```

8.9 Numeros E Char

```cpp
char num_to_char(int num) { // 0 -> '0'
    return num + '0';
}

int char_to_num(char c) { // '0' -> 0
    return c - '0';
}

char int_to_ascii(int num) { // 97 -> 'a'
    return num;
}

int ascii_to_int(char c) { // 'a' -> 97
    return c;
}
```

9 Vector

9.1 Remove Repetitive

```cpp
vector<int> removeRepetitive(const vector<int>& vec) {

    unordered_set<int> s;
    s.reserve(vec.size());

    vector<int> ans;

    for (int num : vec) {
        if (s.insert(num).second)
            v.push_back(num);
    }

    return ans;
}
```

9.2 Maior Triangulo Em Histograma

```cpp
// Calcula o maior âtringulo em um histograma
// Complexidade: O(n)
int maiorTrianguloEmHistograma(const vector<int>& histograma) {

    int n = histograma.size();
    vector<int> esquerda(n), direita(n);

    esquerda[0] = 1;
    f(i,1,n) {
        esquerda[i] = min(histograma[i], esquerda[i - 1] + 1);
    }

    direita[n - 1] = 1;
    rf(i,n-1,0) {
        direita[i] = min(histograma[i], direita[i + 1] + 1);
    }
```

```
    int ans = 0;
    f(i,0,n) {
        ans = max(ans, min(esquerda[i], direita[i]));
    }

    return ans;

}
```

9.3 Maior Retangulo Em Histograma

```
// Calcula area do maior retangulo em um histograma
// Complexidade: O(n)
int maxHistogramRect(const vector<int>& hist) {
    stack<int> s;
    int n = hist.size();

    int ans = 0, tp, area_with_top;

    int i = 0;
    while (i < n) {

        if (s.empty() || hist[s.top()] <= hist[i])
            s.push(i++);

        else {
            tp = s.top(); s.pop();

            area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);

            if (ans < area_with_top)
                ans = area_with_top;
        }
    }

    while (!s.empty()) {
        tp = s.top();  s.pop();
        area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);

        if (ans < area_with_top)
            ans = area_with_top;
    }

    return ans;
}

int main() {
    vector<int> hist = { 6, 2, 5, 4, 5, 1, 6 };
    cout << maxHistogramRect(hist) << endl;
}
```

9.4 Contar Subarrays Somam K

```
// Descricao: Conta quantos subarrays de um vetor tem soma igual a k
// Complexidade: O(n)
int contarSomaSubarray(vector<int>& v, int k) {
    unordered_map<int, int> prevSum; // map to store the previous sum

    int ret = 0, currentSum = 0;

    for(int& num : v) {
        currentSum += num;

        if (currentSum == k) ret++; /// Se a soma atual for igual a k, encontramos um subarray

        if (prevSum.find(currentSum - k) != prevSum.end()) // se subarray com soma (currentSum - k) existir, sabe que [0:n] eh um subarray com soma k
            ret += (prevSum[currentSum - k]);

        prevSum[currentSum]++;
    }

    return ret;
}
```

9.5 Troco

```
vector<int> troco(vector<int> coins, int n) {
    int first[n];
    value[0] = 0;
    for(int x=1; x<=n; x++) {
        value[x] = INF;
        for(auto c : coins) {
            if(x-c => 0 and value[x-c] + 1 < value[x]) {
                value[x] = value[x-c]+1;
                first[x] = c;
            }
        }
    }

    vector<int> ans;
    while(n>0) {
        ans.push_back(first[n]);
        n -= first[n];
    }
    return ans;
}
```

9.6 Elemento Mais Frequente

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

// Encontra o unico elemento mais frequente em um vetor
// Complexidade: O(n)
int maxFreq1(vector<int> v) {
    int res = 0;
    int count = 1;

```

```
    for(int i = 1; i < v.size(); i++) {

        if(v[i] == v[res])
            count++;
        else
            count--;

        if(count == 0) {
            res = i;
            count = 1;
        }
    }

    return v[res];
}

// Encontra os elemento mais frequente em um vetor
// Complexidade: O(n)
vector<int> maxFreqn(vector<int> v)
{
    unordered_map<int, int> hash;
    for (int i = 0; i < v.size(); i++)
        hash[v[i]]++;

    int max_count = 0, res = -1;
    for (auto i : hash) {
        if (max_count < i.second) {
            res = i.first;
            max_count = i.second;
        }
    }

    vector<int> ans;
    for (auto i : hash) {
        if (max_count == i.second) {
            ans.push_back(i.first);
        }
    }

    return ans;
}
```

9.7 Maior Sequencia Subsequente

```
// Maior sequencia subsequente
// {6, 2, 5, 1, 7, 4, 8, 3} => {2, 5, 7, 8}

int maiorCrescente(vector<int> v) {
    vector<int> lenght(v.size());
    for(int k=0; k<v.size(); k++) {
        lenght[k] = ;
        for(int i=0; i<k; i++) {
            if(v[i] < v[k]) {
                lenght[i] = max(lenght[k], lenght[i]+1)
            }
        }
    }
    return lenght.back();
}
```

10 Outros

10.1 Binario

```
// Descicao: conversao de decimal para binario
// Complexidade: O(logn) onde n eh o numero decimal
string decimal_to_binary(int dec) {
    string binary = "";
    while (dec > 0) {
        int bit = dec % 2;
        binary = to_string(bit) + binary;
        dec /= 2;
    }
    return binary;
}

// Descicao: conversao de binario para decimal
// Complexidade: O(logn) onde n eh o numero binario
int binary_to_decimal(string binary) {
    int dec = 0;
    int power = 0;
    for (int i = binary.length() - 1; i >= 0; i--) {
        int bit = binary[i] - '0';
        dec += bit * pow(2, power);
        power++;
    }
    return dec;
}
```

10.2 Horario

```
// Descricao: Funcoes para converter entre horas e segundos.
// Complexidade: O(1)
int cts(int h, int m, int s) {
    int total = (h * 3600) + (m * 60) + s;
    return total;
}

tuple<int, int, int> cth(int total_seconds) {
    int h = total_seconds / 3600;
    int m = (total_seconds % 3600) / 60;
    int s = total_seconds % 60;
    return make_tuple(h, m, s);
}
```

10.3 Intervalos

```
// Conta quantos intervalos nao tem overlap ([a,b] e [b,c] nao geram)

bool cmp(const pair<int,int>& p1, const pair<int,int>& p2) {
    if(p1.second != p2.second) return p1.second < p2.second;
    return p1.first < p2.first;
```

```
}

int countNonOverlappingIntervals(vector<pair<int,int>> intervals) {
    sort(all(intervals), cmp);
    int firstTermino = intervals[0].second;
    int ans = 1;
    f(i,1,intervals.size()) {
        if(intervals[i].first >= firstTermino) {
            ans++;
            firstTermino = intervals[i].second;
        }
    }

    return ans;
}
```

10.4 Max Subarray Sum

```
// Maximum Subarray Sum
// Descricao: Retorna a soma maxima de um subarray de um vetor.
// Complexidade: O(n) onde n eh o tamanho do vetor
int maxSubarraySum(vector<int> x) {
    int best = 0, sum = 0;
    for (int k = 0; k < n; k++) {
        sum = max(x[k],sum+x[k]);
        best = max(best,sum);
    }
    return best;
}
```

10.5 Binary Search

```
// Description: çãImplementao do algoritmo de busca ábinria.
// Complexidade: O(logn) onde n eh o tamanho do vetor
int BinarySearch(<vector>int arr, int x){
    int k = 0;
    int n = arr.size();

    for (int b = n/2; b >= 1; b /= 2) {
        while (k+b < n && arr[k+b] <= x) k += b;
    }
    if (arr[k] == x) {
        return k;
    }
}
```

10.6 Fibonacci

```
// Descricao: Funcao que retorna o n-esimo termo da sequencia de Fibonacci
    .
// Complexidade: O(2^n) onde n eh o termo desejado
int fib(int n){
    if(n <= 1){
        return n;
    }
    return fib(n - 1) + fib(n - 2);
}

vector<int> memo(MAX, -1);

// Descricao: Funcao que retorna o n-esimo termo da sequencia de Fibonacci
     utilizando programacao dinamica.
// Complexidade: O(n) onde n eh o termo desejado
int fibPD(int n) {
    if (n <= 1) return n;
    if (memo[n] != -1) return memo[n];
    return memo[n] = fibPD(n - 1) + fibPD(n - 2);
}
```