Pedro Augusto Ulisses Andrade Lucas Andrade

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

Contents

1 Utils

1.1 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()
```

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)
 [17,19]       | O(2^n * n^2)
 [18, 22]      | O(2^n * n)
 [24,26]       | O(2^n)
 ... 100       | O(n^4)
 ... 450       | O(n^3)
 ... 1500      | O(n^2.5)
 ... 2500      | O(n^2 * log n)
 ... 10^4      | O(n^2)
 ... 2*10^5    | O(n^1.5)
 ... 4.5*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 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
```

1.4 Files

```
#!/bin/bash

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

1.5 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;
}
```

1.6 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 vi            vector<int>
```

```cpp
#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;
}
```

2 Informações

2.1 Bitmask

```cpp
int n = 11, ans = 0, k = 3;

// Operacoes com bits
ans = n & k; // AND bit a bit
ans = n | k; // OR bit a bit
ans = n ^ k; // XOR bit a bit
ans = ~n;    // NOT bit a bit

// Operacoes com 2^k em O(1)
ans = n << k; // ans = n * 2^k
ans = n >> k; // ans = n / 2^k

int j;

// Ativa j-esimo bit (0-based)
ans |= (1<<j);

// Desativa j-esimo bit (0-based)
ans &= (1<<j);

// Inverte j-esimo bit (0-based)
ans ^= (1<<j);

// checar se j-esimo bit esta ativo (0-based)
ans = n & (1<<j);

// Pegar valor do bit menos significativo | Retorna o maior divisor
ans = n & -n;

// Ligar todos on n bits
ans = (1<<n) - 1;

// Contar quantos 1's tem no binario de n
ans = __builtin_popcount(n);

// Contar quantos 0's tem no final do binario de n
ans = __builtin_ctz(n);
```

2.2 Priority Queue

```cpp
// 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);
```

2.3 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; };

    // Sort parcial:
    partial_sort(v.begin(), v.begin() + n, v.end()); // sorta com n menos elementos
    partial_sort(v.rbegin(), v.rbegin() + n, s.rend()) // sorta com n maiores elementos

    // SORT VS SET
    * para um input com elementos distintos, sort é mais árpido que set
```

2.4 Set

```
set<int> st;

// Complexidade: O(log(n))
st.insert(x);
st.erase(x);
st.find(x);
st.erase(st.find(x));


// Complexidade: O(1)
st.size();
st.empty();

// Complexidade: O(n)
st.clear();
for (auto x : st) {}
```

	priority_queue		set		
op	call	compl	call	compl	melhor
insert	push	log(n)	insert	log(n)	pq
erase_menor	pop	log(n)	erase	log(n)	pq
get_menor	top	1	begin	1	set
get_maior	-	-	rbegin	1	set
erase_number	remove	n	erase	log(n)	set
find_number	-	-	find	log(n)	set
find_>=	-	-	lower	log(n)	set
find_<=	-	-	upper	log(n)	set
iterate	for	n	for	n	set

2.5 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);
```

2.6 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 Com Repeticao

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

3.2 @ Factorial

```
// Calcula o fatorial de um únmero n
// Complexidade: O(n)

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);
}
```

3.3 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;
}
```

3.4 @ 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
```

3.5 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);
}
```

3.6 Combinacao Simples

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

3.7 Arranjo Simples

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

3.8 Combinacao Com Repeticao

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

3.9 Permutacao Circular

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

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

4 Estruturas

4.1 Segmen Tree

```
// Segment Tree with Lazy Propagation
// Update Range: O(log(n))
// Querry Range: O(log(n))
// Memory: O(n)
// Build: O(n)

typedef vector<int> vi;

class SegmentTree {
    private:
        int n;
        vi A, st, lazy;
        int defaultVar; // min: INT_MIN | max: INT_MIN | sum: 0 | multiply: 1

        int l(int p) { return  p<<1; }
        int r(int p) { return (p<<1)+1; }

        int conquer(int a, int b) {
            if(a == defaultVar) return b;
            if(b == defaultVar) return a;
            return min(a, b);
        }

        void build(int p, int L, int R) {
            if (L == R) st[p] = A[L];
            else {
                int m = (L+R)/2;
                build(l(p), L  , m);
                build(r(p), m+1, R);
                st[p] = conquer(st[l(p)], st[r(p)]);
            }
        }

        void propagate(int p, int L, int R) {
            if (lazy[p] != defaultVar) {
                st[p] = lazy[p];
                if (L != R) lazy[l(p)] = lazy[r(p)] = lazy[p];
                else        A[L] = lazy[p];
                lazy[p] = defaultVar;
            }
        }

        int querry(int p, int L, int R, int i, int j) {
            propagate(p, L, R);
            if (i > j) return defaultVar;
            if ((L >= i) && (R <= j)) return st[p];
            int m = (L+R)/2;
            return conquer(querry(l(p), L  , m, i, min(m, j)),
                           querry(r(p), m+1, R, max(i, m+1), j));
        }

        void update(int p, int L, int R, int i, int j, int val) {
            propagate(p, L, R);
            if (i > j) return;
            if ((L >= i) && (R <= j)) {
                lazy[p] = val;
                propagate(p, L, R);
            }
            else {
                int m = (L+R)/2;
                update(l(p), L  , m, i          , min(m, j), val);
                update(r(p), m+1, R, max(i, m+1), j        , val);
                int lsubtree = (lazy[l(p)] != defaultVar) ? lazy[l(p)] : st[l(p)];
                int rsubtree = (lazy[r(p)] != defaultVar) ? lazy[r(p)] : st[r(p)];
                st[p] = conquer(lsubtree, rsubtree);
            }
        }

    public:
        SegmentTree(int sz, int defaultVal) : n(sz), A(n), st(4*n), lazy(4*n, defaultVal), defaultVar(defaultVal) {}

        // vetor referencia, valor default (min: INT_MIN | max: INT_MIN |
```

```
    sum: 0 | multiply: 1)
        SegmentTree(const vi &initialA, int defaultVal) : SegmentTree((int)initialA.size(), defaultVal) {
            A = initialA;
            build(1, 0, n-1);
        }

        // A[i..j] = val | 0 <= i <= j < n  | O(log(n))
        void update(int i, int j, int val) { update(1, 0, n-1, i, j, val); }

        // max(A[i..j]) | 0 <= i <= j < n  | O(log(n))
        int querry(int i, int j) { return querry(1, 0, n-1, i, j); }
};

void solve() {
    vi A = {18, 17, 13, 19, 15, 11, 20, 99};       // make n a power of 2
    int defaultVar = INT_MIN; // default value for max query
    SegmentTree st(A, defaultVar);
    int i = 1, j = 3;
    int ans = st.querry(i, j);
    int newVal = 77;
    st.update(i, j, newVal);
    ans = st.querry(i, j);
}
```

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 SparseTable {
    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 Bittree

```
/*        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(vector<int>& BITree, int index) {
    int sum = 0;
    index = index + 1;
    while (index>0) {
        sum += BITree[index];
        index -= index & (-index);
    }
    return sum;
}

void updateBIT(vector<int>& BITree, int n, int index, int val) {
    index = index + 1;

    while (index <= n) {
        BITree[index] += val;
        index += index & (-index);
    }
}

vector<int> constructBITree(vector<int>& arr, int n) {
    vector<int> BITree(n+1, 0);

    for (int i=0; i<n; i++)
        updateBIT(BITree, n, i, arr[i]);

    return BITree;
}

void solve() {
    vector<int> freq = {2, 1, 1, 3, 2, 3, 4, 5, 6, 7, 8, 9};
    int n = freq.size();
    vector<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);
}
```

4.4 Seg Tree

```
// 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 SegTree {

    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));
    }
};
```

4.5 Union Find

```
// Description: Union-Find (Disjoint Set Union)

typedef vector<int> vi;

struct UnionFind {
    vi p, rank, setSize;
    int numSets;
    UnionFind(int N) {
        p.assign(N, 0);
        for (int i = 0; i < N; ++i)
            p[i] = i;
        rank.assign(N, 0);
        setSize.assign(N, 1);
        numSets = N;
    }

    // Retorna o numero de sets disjuntos (separados)
    int numDisjointSets() { return numSets; }
    // Retorna o tamanho do set que contem o elemento i
    int sizeOfSet(int i) { return setSize[find(i)]; }

    int find(int i) { return (p[i] == i) ? i : (p[i] = find(p[i])); }
    bool same(int i, int j) { return find(i) == find(j); }
    void uni(int i, int j) {
        if (same(i, j))
            return;
        int x = find(i), y = find(j);
        if (rank[x] > rank[y])
            swap(x, y);
        p[x] = y;
        if (rank[x] == rank[y])
            ++rank[y];
        setSize[y] += setSize[x];
        --numSets;
    }
};

void solve() {
    int n; cin >> n;
    UnionFind UF(n);
    UF.uni(0, 1);
}
```

4.6 Fenwick Tree

```
#define LSOne(S) ((S) & -(S)) // the key operation

class FenwickTree { // index 0 is not used
    private:
        vi ft;

        void build(const vi &f) {
            int m = (int)f.size() - 1; // note f[0] is always 0
            ft.assign(m + 1, 0);
            for (int i = 1; i <= m; ++i) {
                ft[i] += f[i];
                if (i + LSOne(i) <= m)
                    ft[i + LSOne(i)] += ft[i];
            }
        }

public:
    // empty FT
    FenwickTree(int m) { ft.assign(m + 1, 0); }

    // FT based on f
    FenwickTree(const vi &f) { build(f); }

    // FT based on s, and m = max(s)
    FenwickTree(int m, const vi &s) {
        vi f(m + 1, 0);
```

```
        for (int i = 0; i < (int)s.size(); ++i)
            ++f[s[i]];
        build(f);
    }

    // RSQ(1, j)
    int rsq(int j)     {
        int sum = 0;
        for (; j; j -= LSOne(j))
            sum += ft[j];
        return sum;
    }

    // RSQ(i, j)
    int rsq(int i, int j) { return rsq(j) - rsq(i - 1); }

    // v[i] += v
    void update(int i, int v) {
        for (; i < (int)ft.size(); i += LSOne(i))
            ft[i] += v;
    }

    // n-th element >= k
    int select(int k) {
        int p = 1;
        while (p * 2 < (int)ft.size())
            p *= 2;
        int i = 0;
        while (p) {
            if (k > ft[i + p]) {
                k -= ft[i + p];
                i += p;
            }
            p /= 2;
        }
        return i + 1;
    }
};

// Range Update Point Query
class RUPQ {
    private:
        FenwickTree ft;
    public:

        // empty FT
        RUPQ(int m) : ft(FenwickTree(m)) {}

        // v[ui,...,uj] += v
        void range_update(int ui, int uj, int v) {
            ft.update(ui, v);
            ft.update(uj + 1, -v);
        }

        // rsq(i) = v[1] + v[2] + ... + v[i]
        int point_query(int i) { return ft.rsq(i); }
};


// Range Update Range Query
class RURQ {
    private:
        RUPQ rupq;
        FenwickTree purq;
    public:
        // empty structures
        RURQ(int m) : rupq(RUPQ(m)), purq(FenwickTree(m)) {}

        // v[ui,...,uj] += v
        void range_update(int ui, int uj, int v) {
            rupq.range_update(ui, uj, v);
            purq.update(ui, v * (ui - 1));
            purq.update(uj + 1, -v * uj);
        }

        // rsq(j) = v[1]*j - (v[1] + ... + v[j])
        int rsq(int j) {
            return rupq.point_query(j) * j -
                purq.rsq(j);
        }

        // rsq(i, j) = rsq(j) - rsq(i - 1)
        int rsq(int i, int j) { return rsq(j) - rsq(i - 1); }
};

int32_t main() {

    vi f = {0, 0, 1, 0, 1, 2, 3, 2, 1, 1, 0}; // index 0 is always 0
    FenwickTree ft(f);
    printf("%lli\n", ft.rsq(1, 6));  // 7 => ft[6]+ft[4] = 5+2 = 7
    printf("%lld\n", ft.select(7));    // index 6, rsq(1, 6) == 7, which
    is >= 7
    ft.update(5, 1);                  // update demo
    printf("%lli\n", ft.rsq(1, 10)); // now 12
    printf("=====\n");
    RUPQ rupq(10);
    RURQ rurq(10);
    rupq.range_update(2, 9, 7); // indices in [2, 3, .., 9] updated by +7
    rurq.range_update(2, 9, 7); // same as rupq above
    rupq.range_update(6, 7, 3); // indices 6&7 are further updated by +3
    (10)
    rurq.range_update(6, 7, 3); // same as rupq above
    // idx = 0 (unused) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |10
    // val = -          | 0 | 7 | 7 | 7 | 7 |10 |10 | 7 | 7 | 0
    for (int i = 1; i <= 10; i++)
        printf("%lld -> %lli\n", i, rupq.point_query(i));
    printf("RSQ(1, 10) = %lli\n", rurq.rsq(1, 10)); // 62
    printf("RSQ(6, 7) = %lli\n", rurq.rsq(6, 7));   // 20
    return 0;
}
```

5 Grafos

5.1 Encontrar Ciclo

```cpp
// Description: Encontrar ciclo em grafo nao direcionado
// Complexidade: O(n + m)

int n;
vector<vector<int>> adj;
vector<bool> vis;
vector<int> p;
int cycle_start, cycle_end;

bool dfs(int v, int par) {
    vis[v] = true;
    for (int u : adj[v]) {
        if(u == par) continue;
        if(vis[u]) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
        p[u] = v;
        if(dfs(u, p[u]))
            return true;
    }
    return false;
}

vector<int> find_cycle() {
    cycle_start = -1;

    for (int v = 0; v < n; v++)
        if (!vis[v] and dfs(v, p[v]))
            break;

    if (cycle_start == -1) return {};

    vector<int> cycle;
    cycle.push_back(cycle_start);
    for (int v = cycle_end; v != cycle_start; v = p[v])
        cycle.push_back(v);
    cycle.push_back(cycle_start);
    return cycle;
}

void solve() {
    int edg; cin >> n >> edg;
    adj.assign(n, vector<int>());
    vis.assign(n, false), p.assign(n, -1);
    while(edg--) {
        int a, b; cin >> a >> b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    vector<int> ans = find_cycle();
}
```

5.2 Topological Kahn

```cpp
// Description: Ordenamento topologico usando o algoritmo de Kahn.
// Complexidade: O(V+E)
vector<vector<int>> adj;

vector<int> topologicalSort(int V) {

    vector<int> indegree(V);
    for (int i = 0; i < V; i++) {
        for (auto it : adj[i]) {
            indegree[it]++;
        }
    }

    queue<int> q;
    for (int i = 0; i < V; i++) {
        if (indegree[i] == 0) {
            q.push(i);
        }
    }
    vector<int> result;
    while (!q.empty()) {

        int node = q.front(); q.pop();
        result.push_back(node);

        for (auto it : adj[node]) {
            indegree[it]--;
            if (indegree[it] == 0)
                q.push(it);
        }
    }

    if (result.size() != V) {
        cout << "Graph contains cycle!" << endl;
        return {};
    }

    return result;
}

void solve() {

    int n = 4; adj.resize(n);
    vector<pair<int, int>> edges = { { 0, 1 }, { 1, 2 }, { 3, 1 }, { 3, 2
    } };
    for (auto& [a,b] : edges) {
        adj[a].push_back(b);
    }

    vector<int> ans = topologicalSort(n);
}

int main() {
```

```
    solve();
}
```

5.3 Articulation

```
// Description: Encontra pontos de articulacao e pontes em um grafo nao direcionado
// Complexidade: O(V + E)

vector<vector<pii>> adj;
vi dfs_num, dfs_low, dfs_parent, articulation_vertex;
int dfsNumberCounter, dfsRoot, rootChildren;
vector<pii> bridgesAns;

void articulationPointAndBridgeUtil(int u) {

    dfs_low[u] = dfs_num[u] = dfsNumberCounter++;
    for (auto &[v, w] : adj[u]) {
        if (dfs_num[v] == -1) {
            dfs_parent[v] = u;
            if (u == dfsRoot) ++rootChildren;

            articulationPointAndBridgeUtil(v);

            if (dfs_low[v] >= dfs_num[u])
                articulation_vertex[u] = 1;
            if (dfs_low[v] > dfs_num[u])
                bridgesAns.push_back({u, v});
            dfs_low[u] = min(dfs_low[u], dfs_low[v]);
        }
        else if (v != dfs_parent[u])
            dfs_low[u] = min(dfs_low[u], dfs_num[v]);
    }
}

void articulationPointAndBridge(int n) {
    dfsNumberCounter = 0;
    f(u,0,n) {
        if (dfs_num[u] == -1) {
            dfsRoot = u; rootChildren = 0;
            articulationPointAndBridgeUtil(u);
            articulation_vertex[dfsRoot] = (rootChildren > 1);
        }
    }
}

void solve() {

    int n, ed; cin >> n >> ed;
    adj.assign(n, vector<pii>());

    f(i,0,ed) {
        int u, v, w; cin >> u >> v >> w;
        adj[u].emplace_back(v, w);
    }

    dfs_num.assign(n, -1); dfs_low.assign(n, 0);
    dfs_parent.assign(n, -1); articulation_vertex.assign(n, 0);

    articulationPointAndBridge(n);

    // Vertices: articulation_vertex[u] == 1
    // Bridges: bridgesAns
}
```

5.4 Bipartido

```
// Description: Determina se um grafo eh bipartido ou nao
// Complexidade: O(V+E)

vector<vi> AL;

bool bipartido(int n) {

    int s = 0;
    queue<int> q; q.push(s);

    vi color(n, INF); color[s] = 0;
    bool ans = true;
    while (!q.empty() && ans) {
        int u = q.front(); q.pop();

        for (auto &v : AL[u]) {
            if (color[v] == INF) {
                color[v] = 1 - color[u];
                q.push(v);
            }
            else if (color[v] == color[u]) {
                ans = false;
                break;
            }
        }
    }

    return ans;
}

void solve() {

    int n, edg; cin >> n >> edg;
    AL.resize(n, vi());

    while(edg--) {
        int a, b; cin >> a >> b;
        AL[a].push_back(b);
        AL[b].push_back(a);
    }

    cout << bipartido(n) << endl;
}
```

5.5 Bfs Nivelado

```cpp
// Description: Encontrar distancia entre S e outros pontos em que pontos estao agrupados (terminais)
// EXTRA: BFS diferenciado para armazenar distancias sem VIS

int n;
vi dist;
vector<vi> gruposDoItem, itensDoGrupo;

void bfs(int s) {

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

    while (!q.empty()) {
        auto [v, dis] = q.front(); q.pop();

        for(auto grupo : gruposDoItem[v]) {
            for(auto u : itensDoGrupo[grupo]) {
                if (dist[u] == 0) {
                    q.push({u, dis+1});
                    dist[u] = dis + 1;
                }
            }
        }
    }
}

void solve() {

    int n, ed; cin >> n >> ed;
    dist.clear(), itensDoGrupo.clear(), gruposDoItem.clear();
    itensDoGrupo.resize(n);

    f(i,0,ed) {

        int q; cin >> q;
        while(q--) {
            int v; cin >> v;
            gruposDoItem[v].push_back(i);
            itensDoGrupo[i].push_back(v);
        }
    }

    bfs(0);
}
```

5.6 Dfs

```cpp
vector<int> adj[MAXN], parent;
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 v = st.top(); st.pop();

        if (visited[v]) continue;
        visited[v] = true;

        for (int u : adj[v]) {
            if (!visited[u]) {
                parent[u] = v;
                st.push(u);
            }
        }
    }
}

// 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]) {
            parent[u] = v;
            dfs(u);
        }
    }
}

void solve() {
    int n; cin >> n;
    for (int i = 0; i < n; i++) {
        int u, v; cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    dfs(0);
}
```

5.7 Successor Graph

```cpp
// Encontra sucessor de um vertice dentro de um grafo direcionado
// Pre calcular: O(nlogn)
// Consulta: O(logn)

vector<vector<int>> adj;

int succ(int x, int u) {
    if(k == 1) return adj[x][0];
    return succ(succ(x, k/2), k/2);
}
```

5.8 Cycle Check

```cpp
// Descriptionn: Checa se um grafo direcionado possui ciclos e imprime os tipos de arestas.
```

```cpp
// Complexidade: O(V + E)

vector<vector<pii>> adj;
vi dfs_num, dfs_parent;

void cycleCheck(int u) {
    dfs_num[u] = -2;
    for (auto &[v, w] : adj[u]) {
        if (dfs_num[v] == -1) {
            dfs_parent[v] = u;
            cycleCheck(v);
        }
        else if (dfs_num[v] == -2) {
            if (v == dfs_parent[u])
                cout << " Bidirectional Edge (" << u << ", " << v << ")-("
     << v << ", " << u << ")\n";
            else
                cout << "Back Edge (" << u << ", " << v << ") (Cycle)\n";
        }
        else if (dfs_num[v] == -3)
            cout << "  Forward/Cross Edge (" << u << ", " << v << ")\n";
    }
    dfs_num[u] = -3;
}

void solve() {
    int n, ed; cin >> n >> ed;
    adj.assign(ed, vector<pii>());

    for (int i = 0; i < ed; ++i) {
        int u, v, w; cin >> u >> v >> w;
        adj[u].emplace_back(v, w);
    }

    cout << "Graph Edges Property Check\n";
    dfs_num.assign(ed, -1);
    dfs_parent.assign(ed, -1);
    for (int u = 0; u < n; ++u)
        if (dfs_num[u] == -1)
        cycleCheck(u);
}
```

5.9 Dijkstra

```cpp
// 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;

        for (auto [b, w] : adj[a]) {
            if (dist[a] + w < dist[b]) {
                dist[b] = dist[a] + w;
                parent[b] = a;
                q.push({-dist[b], b});
            }
        }
    }
}

//Complexidade: O(V)
vector<int> restorePath(int v) {
    vector<int> path;
    for (int u = v; u != -1; u = parent[u])
        path.push_back(u);
    reverse(path.begin(), path.end());
    return path;
}

void solve() {

    adj.resize(n); /*n = nodes*/
    f(i,0,n) {
        int a, b, w; cin >> a >> b >> w;
        adj[a].push_back({b, w});
        adj[b].push_back({a, w});
    }
    dijkstra(0);
}

// VARIANTES

/* Menor caminho de todos os vertices para um vertice s
    -> Inverter a direcao das arestas
    -> dijkstra(s)
*/

/* Multi-Sources Shortest Paths
    - Menor caminho de um conjunto de vertices para todos os outros
    -> dist[a] = 0, q.push(a) para todo source a
    -> dijkstra()
*/
```

5.10 Labirinto

```cpp
// Verifica se eh possivel sair de um labirinto
// Complexidade: O(4^(n*m))

vector<pair<int,int>> mov = {{1,0}, {0,1}, {-1,0}, {0,-1}};
vector<vector<int>> labirinto, sol;
vector<vector<bool>> visited;
int L, C;

bool valid(const int& x, const int& y) {
    return x >= 0 and x < L and y >= 0 and y < C and labirinto[x][y] != 0
    and !visited[x][y];
}

bool condicaoSaida(const int& x, const int& y) {
    return labirinto[x][y] == 2;
}

bool search(const int& x, const int& y) {

    if(!valid(x, y))
        return false;

    if(condicaoSaida(x,y)) {
        sol[x][y] = 2;
        return true;
    }

    sol[x][y] = 1;
    visited[x][y] = true;

    for(auto [dx, dy] : mov)
        if(search(x+dx, y+dy))
            return true;

    sol[x][y] = 0;
    return false;
}

int main() {

    labirinto = {
        {1, 0, 0, 0},
        {1, 1, 0, 0},
        {0, 1, 0, 0},
        {1, 1, 1, 2}
    };

    L = labirinto.size(), C = labirinto[0].size();
    sol.resize(L, vector<int>(C, 0));
    visited.resize(L, vector<bool>(C, false));

    cout << search(0, 0) << endl;
}
```

5.11 Bfs

```cpp
// 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;
vector<bool> vis;
vector<int> d, p;
vector<vector<int>> adj;

void bfs(int s) {

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

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

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

void solve() {
    cin >> n;
    adj.resize(n); d.resize(n, -1);
    vis.resize(n); p.resize(n, -1);

    for (int i = 0; i < n; i++) {
        int u, v; cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    bfs(0);
}

// OBS: Pode ser usado para encontrar o menor caminho entre dois vertices
    em um grafo sem pesos
```

5.12 Bfs Matriz

```cpp
// 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;
            }
        }
    }
}

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

5.13 Floyd Warshall

```
// Floyd-Warshall
//
// encontra o menor caminho entre todo
// par de vertices e detecta ciclo negativo
// returna 1 sse ha ciclo negativo
// d[i][i] deve ser 0
// para i != j, d[i][j] deve ser w se ha uma aresta
// (i, j) de peso w, INF caso contrario
//
// 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;
}

void solve() {
    cin >> n; int edg; cin >> edg;
    for (int i = 0; i < edg; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        d[u][v] = w;
    }
}
```

5.14 Pontos Articulacao

```
// Description: Encontra os pontos de çãarticulao de um grafo ãno
    direcionado
// Complexidade: O(V*(V+E))

int V;
vector<vi> adj;
vi ans;

void dfs(vector<bool>& vis, int i, int curr) {
    vis[curr] = 1;
    for (auto x : adj[curr]) {
        if (x != i) {
            if (!vis[x]) {
                dfs(vis, i, x);
            }
        }
    }
}

void AP() {

    f(i,1,V+1) {
        int components = 0;
        vector<bool> vis(V + 1, 0);
        f(j,1, V+1) {
            if (j != i) {
                if (!vis[j]) {
                    components++;
                    dfs(vis, i, j);
                }
            }
        }
        if (components > 1) {
            ans.push_back(i);
        }
    }
}

void solve() {

    V = n;
    adj.clear(), ans.clear();
    adj.resize(V+1);

    while(edg--) {
```

```
        int a, b; cin >> a >> b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    AP();

    // Vertices articulacao: ans
}
```

5.15 Graham Scan(elastico)

```
// çãFuno para calcular o produto vetorial de dois vetores
int cross_product(const pair<int, int>& o, const pair<int, int>& a, const pair<int, int>& b) {
    return (a.first - o.first) * (b.second - o.second) - (a.second - o.second) * (b.first - o.first);
}

// çãFuno para encontrar o ponto mais baixo (esquerda mais baixo)
pair<int, int> find_lowest_point(const vector<pair<int, int>>& points) {
    pair<int, int> lowest = points[0];
    for (const auto& point : points) {
        if (point.second < lowest.second || (point.second == lowest.second && point.first < lowest.first)) {
            lowest = point;
        }
    }
    return lowest;
}

// çãFuno para ordenar pontos por ângulo polar em çãrelao ao ponto mais baixo
bool compare(const pair<int, int>& a, const pair<int, int>& b, const pair<int, int>& lowest_point) {
    int cross = cross_product(lowest_point, a, b);
    if (cross != 0) {
        return cross > 0;
    }
    return (a.first != b.first) ? (a.first < b.first) : (a.second < b.second);
}

// çãFuno para encontrar o óenvoltrio convexo usando o algoritmo de Varredura de Graham
vector<pair<int, int>> convex_hull(vector<pair<int, int>>& points) {
    vector<pair<int, int>> convex_polygon;

    if (points.size() < 3) return convex_polygon;

    pair<int, int> lowest_point = find_lowest_point(points);
    sort(points.begin(), points.end(), [&lowest_point](const pair<int, int>& a, const pair<int, int>& b) {
        return compare(a, b, lowest_point);
    });

    convex_polygon.push_back(points[0]);
    convex_polygon.push_back(points[1]);

    for (int i = 2; i < points.size(); ++i) {
        while (convex_polygon.size() >= 2 && cross_product(convex_polygon[convex_polygon.size() - 2], convex_polygon.back(), points[i]) <= 0) {
            convex_polygon.pop_back();
        }
        convex_polygon.push_back(points[i]);
    }

    return convex_polygon;
}

void solve() {
    int n, turma = 0;


        vector<pair<int, int>> points(n);
        for (int i = 0; i < n; ++i) {
            cin >> points[i].first >> points[i].second; // x y
        }

        vector<pair<int, int>> convex_polygon = convex_hull(points);
        int num_vertices = convex_polygon.size();

        cout << num_vertices << endl; // qnt de vertices , se quiser os pontos so usar o vi convex_polygon

        cout << endl;
}
```

5.16 Kruskal

```
// DEscricao: Encontra a arvore geradora minima de um grafo
// Complexidade: O(E log V)

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];
}

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

    sort(edg.begin(), edg.end()); // Minimum Spanning Tree

    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};
}

void solve() {

    int n, ed;

    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);
}

// VARIANTES

// Maximum Spanning Tree: sort(edg.rbegin(), edg.rend());

/* 'Minimum' Spanning Subgraph:
    - Algumas arestas ja foram adicionadas (maior prioridade - Questao das rodovias)
    - Arestas que nao foram adicionadas (menor prioridade - ferrovias)
    -> kruskal(rodovias); kruskal(ferrovias);
*/

/* Minimum Spanning Forest:
    - Queremos uma floresta com k componentes
    -> kruskal(edg); if(mst.sizer() == k) break;
*/

/* MiniMax
    - Encontrar menor caminho entre dous vertices com maior quantidade de arestas
    -> kruskal(edg); dijsktra(mst);
*/

/* Second Best MST
    - Encontrar a segunda melhor arvore geradora minima
    -> kruskal(edg);
    -> flag mst[i] = 1;
    -> sort(cmp(edg.flag != -1)) => da prioridade para outras arestas
*/
```

5.17 Kosaraju

```
// Description: Encontra o numero de componentes fortemente conexas em um grafo direcionado
// Complexidade: O(V + E)

int dfsNumberCounter, numSCC;
vector<vii> adj, adj_t;
vi dfs_num, dfs_low, S, visited;
stack<int> St;

void kosarajuUtil(int u, int pass) {
    dfs_num[u] = 1;
    vii &neighbor = (pass == 1) ? adj[u] : adj_t[u];
    for (auto &[v, w] : neighbor)
        if (dfs_num[v] == -1)
        kosarajuUtil(v, pass);
    S.push_back(u);
}

bool kosaraju(int n) {

    S.clear();
    dfs_num.assign(n, -1);

    f(u,0,n) {
        if (dfs_num[u] == -1)
            kosarajuUtil(u, 1);
    }

    int numSCC = 0;
    dfs_num.assign(n, -1);
    f(i,n-1,-1) {
        if (dfs_num[S[i]] == -1)
            numSCC++, kosarajuUtil(S[i], 2);
    }

    return numSCC == 1;
}

void solve() {

    int n, ed; cin >> n >> ed;
    adj.assign(n, vii());
    adj_t.assign(n, vii());

    while (ed--) {
        int u, v, w; cin >> u >> v >> w;
        AL[u].emplace_back(v, 1);
        adj_t[v].emplace_back(u, 1);
    }

    // Printa se o grafo eh fortemente conexo
    cout << kosaraju(n) << endl;

    // Printa o numero de componentes fortemente conexas
    cout << numSCC << endl;

    // Printa os vertices de cada componente fortemente conexa
    f(i,0,n){
```

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

5.18 Euler Tree

```
// Descricao: Encontra a euler tree de um grafo
// Complexidade: O(n)
vector<vector<int>> adj(MAX);
vector<int> vis(MAX, 0);
vector<int> euTree(MAX);

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;
        }
    }
}

void solve() {

    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);
}
```

6 Matematica

6.1 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));

}
```

6.2 Mmc

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

6.3 Fatoracao

```
// 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;
}
```

6.4 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()];
}
```

6.5 Mmc Multiplo

```
// 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;
}
```

6.6 Contar Quanti Solucoes Eq 2 Variaveis

```
// Description: Dada uma equacao de 2 variaveis, calcula quantas
    combinacoes {x,y}
// inteiras que resolvem essa equacao
// Complexidade: O(sqrt(c))
// 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();
}
```

6.7 Mdc

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

6.8 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};
}
```

6.9 Mdc Multiplo

```
// 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;
}
```

6.10 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);
}
```

6.11 Factorial

```
unordered_map<int, int> memo;

// 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);
}
```

6.12 N Fibonacci

```
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);
    }
}
```

6.13 Divisores

```
// Descricao: Calcula 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;
}
```

6.14 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 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;
  }
}
```

6.16 Conversao De Bases

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

string decimalToBase(int n, int base = 10) {
    if (not n) return "0";
    stringstream ss;
    for (int 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)
int intForDigit(char digit) {
    int intDigit = digit - 48;
    if (intDigit > 9) return digit - 87;
    return intDigit;
}

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

6.17 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);
}
```

6.18 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;
}
```

6.19 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;
}
```

6.20 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);
}
```

6.21 Fast Exponentiation

```
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;
}
```

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);
}
```

7.2 Max 2d Range Sum

```
// Maximum Sum
// O(n^3) 1D DP + greedy (Kadane's) solution, 0.000s in UVa

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

#define f(i,s,e)     for(int i=s;i<e;i++)
#define MAX_n 110

int A[MAX_n][MAX_n];

int maxMatrixSum(vector<vector<int>> mat) {

    int n = mat.size();
    int m = mat[0].size();

    f(i,0,n) {
        f(j,0,m) {
            if (j > 0)
                mat[i][j] += mat[i][j - 1];
        }
    }

    int maxSum = INT_MIN;
    f(l,0,m)  {
        f(r,l,m) {
            vector<int> sum(n, 0);
            f(row,0,n) {
                sum[row] = mat[row][r] - (l > 0 ? mat[row][l - 1] : 0);
            }
            int maxSubRect = sum[0];
            f(i,1,n) {
                if (sum[i - 1] > 0)
                    sum[i] += sum[i - 1];
                maxSubRect = max(maxSubRect, sum[i]);
            }
            maxSum = max(maxSum, maxSubRect);
        }
    }
```

```
    return maxSum;
}
```

8 Strings

8.1 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;
}
```

8.2 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;
}
```

8.3 Chaves Colchetes Parenteses

```
// Description: Verifica se s tem uma êsequncia valida de {}, [] e ()
// Complexidade: O(n)
bool brackets(string s) {
    stack<char> st;

    for (char c : s) {
        if (c == '(' || c == '[' || c == '{') {
            st.push(c);
        } else {
            if (st.empty()) return false;
            if (c == ')' and st.top() != '(') return false;
            if (c == ']' and st.top() != '[') return false;
            if (c == '}' and st.top() != '{') return false;
            st.pop();
        }
    }

    return st.empty();
}
```

8.4 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);
}
```

8.5 Lower Upper

```
// 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);
```

8.6 Numeros E Char

```
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;
}
```

8.7 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;
}
```

8.8 Infixo Para Posfixo

```
// Description: Converte uma expressao matematica infixa para posfixa
// Complexidade: O(n)
string infixToPostfix(string s) {
    stack<char> st;
    string res;
    for (char c : s) {
        if (isdigit(c))
            res += c;
        else if (c == '(')
            st.push(c);
        else if (c == ')') {
            while (st.top() != '(') {
                res += st.top();
                st.pop();
            }
            st.pop();
        } else {
            while (!st.empty() and st.top() != '(' and
                   (c == '+' or c == '-' or (st.top() == '*' or st.top() == '/'))) {
                res += st.top();
                st.pop();
            }
            st.push(c);
        }
    }
    while (!st.empty()) {
        res += st.top();
        st.pop();
    }
    return res;
}
```

8.9 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;
}
```

8.10 Lexicograficamente Minima

```
// 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];
}
```

8.11 Calculadora Posfixo

```
// Description: Calculadora de expressoes posfixas
// Complexidade: O(n)
int posfixo(string s) {
    stack<int> st;
    for (char c : s) {
        if (isdigit(c)) {
            st.push(c - '0');
```

```
        } else {
            int b = st.top(); st.pop();
            int a = st.top(); st.pop();
            if (c == '+') st.push(a + b);
            if (c == '-') st.push(a - b);
            if (c == '*') st.push(a * b);
            if (c == '/') st.push(a / b);
        }
    }
    return st.top();
}
```

9 Vector

9.1 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;
}
```

9.2 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;
}
```

9.3 Subset Sum

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

```cpp
// Complexidade Temporal: O(sum * n)
// Complexidade Espacial: O(sum * n)

bool isSubsetSum(vi set, int n, int sum) {
    bool subset[n + 1][sum + 1];

    for (int i = 0; i <= n; i++)
        subset[i][0] = true;

    for (int i = 1; i <= sum; i++)
        subset[0][i] = false;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= sum; j++) {
            if (j < set[i - 1])
                subset[i][j] = subset[i - 1][j];
            if (j >= set[i - 1])
                subset[i][j]
                    = subset[i - 1][j]
                      || subset[i - 1][j - set[i - 1]];
        }
    }

    return subset[n][sum];
}
```

9.4 Contar Subarrays Somam K

```cpp
// 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 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;

}
```

9.6 K Maior Elemento

```cpp
// Description: Encontra o ké-simo maior elemento de um vetor
// Complexidade: O(n)

int Partition(vector<int>& A, int l, int r) {
    int p = A[l];
    int m = l;
    for (int k = l+1; k <= r; ++k) {
        if (A[k] < p) {
            ++m;
            swap(A[k], A[m]);
        }
    }
    swap(A[l], A[m]);
    return m;
}

int RandPartition(vector<int>& A, int l, int r) {
    int p = l + rand() % (r-l+1);
    swap(A[l], A[p]);
    return Partition(A, l, r);
}

int QuickSelect(vector<int>& A, int l, int r, int k) {
    if (l == r) return A[l];
    int q = RandPartition(A, l, r);
    if (q+1 == k)
        return A[q];
    else if (q+1 > k)
        return QuickSelect(A, l, q-1, k);
    else
        return QuickSelect(A, q+1, r, k);
```

```
}

void solve() {
    vector<int> A = { 2, 8, 7, 1, 5, 4, 6, 3 };
    int k = 1;
    cout << QuickSelect(A, 0, A.size()-1, k) << endl;
}
```

9.7 Remove Repetitive

```
// Remove repetitive elements from a vector
// Complexity: O(n)
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;
}

void solve() {
    vector<int> v = {1, 3, 2, 5, 4, 2, 3, 4, 5};
    vector<int> ans = removeRepetitive(v); // {1, 3, 2, 5, 4}
}
```

9.8 Troco

```
// Description: Retorna o menor únmero de moedas para formar um valor n
// Complexidade: O(n*m)
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;
}

void solve() {
    vector<int> coins = {1, 3, 4};
    vector<int> ans = troco(coins, 6); // {3,3}
}
```

9.9 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();
}
```

9.10 Maior Subsequência Crescente

```
// Retorna o tamanho da maior êsubsequncia crescente de v
// Complexidade: O(n log(n))
int maiorSubCrescSize(vector<int> &v) {

    vector<int> pilha;
    for (int i = 0; i < v.size(); i++) {
        auto it = lower_bound(pilha.begin(), pilha.end(), v[i]);
        if (it == pilha.end())
            pilha.push_back(v[i]);
        else
            *it = v[i];
    }

    return pilha.size();
}

// Retorna a maior êsubsequncia crescente de v
// Complexidade: O(n log(n))
vector<int> maiorSubCresc(vector<int> &v) {

    vector<int> pilha, resp;
    int pos[MAXN], pai[MAXN];
    for (int i = 0; i < v.size(); i++) {
        auto it = lower_bound(pilha.begin(), pilha.end(), v[i]);
        int p = it - pilha.begin();
        if (it == pilha.end())
            pilha.PB(v[i]);
        else
            *it = x;
        pos[p] = i;
```

```
        if (p == 0)
            pai[i] = -1; // seu pai áser -1
        else
            pai[i] = pos[p - 1];
    }

    int p = pos[pilha.size() - 1];
    while (p >= 0) {
        resp.PB(v[p]);
        p = pai[p];
    }
    reverse(resp.begin(), resp.end());

    return resp;
}

void solve() {
    vector<int> v = {1, 3, 2, 5, 4, 2, 3, 4, 5};
    cout << maiorSubCrescSize(v) << endl // 5
    /*************************************/
    vector<int> ans = maiorSubCresc(v); // {1,2,3,4,5}
}
```

9.11 Maior Subsequencia Comum

```
int s1[MAXN], s2[MAXN], tab[MAXN][MAXN];

// Description: Retorna o tamanho da maior êsubsequncia comum entre s1 e
    s2
// Complexidade: O(n*m)
int lcs(int a, int b){

    if(tab[a][b]>=0) return tab[a][b];
    if(a==0 or b==0) return tab[a][b]=0;
    if(s1[a]==s2[b]) return 1 + lcs(a-1, b-1);
    return tab[a][b] = max(lcs(a-1, b), lcs(a, b-1));
}

void solve() {

    s1 = {1, 3, 2, 5, 4, 2, 3, 4, 5};
    s2 = {1, 2, 3, 4, 5};
    int n = s1.size(), m = s2.size();
    memset(tab, -1, sizeof(tab));
    cout << lcs(n, m) << endl; // 5
}
```

9.12 Soma Maxima Sequencial

```
// Description: Soma maxima sequencial de um vetor
// Complexidade: O(n)
int max_sum(vector<int> s) {

    int ans = 0, maior = 0;

    for(int i = 0; i < s.size(); i++) {
        maior = max(0,maior+s[i]);
        ans = max(resp,maior);
    }

    return ans;
}

void solve() {
    vector<int> v = {1,-3,5,-1,2,-1};
    cout << max_sum(v) << endl; // 6 = {5,-1,2}
}
```

10 Outros

10.1 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;
}
```

10.2 Dp

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

const int MAX_gm = 30; // up to 20 garments at most and 20 models/garment
const int MAX_M = 210; // maximum budget is 200

int M, C, price[MAX_gm][MAX_gm]; // price[g (<= 20)][k (<= 20)]
int memo[MAX_gm][MAX_M];         // TOP-DOWN: dp table [g (< 20)][money
    (<= 200)]

int dp(int g, int money) {

    if (money < 0) return -1e9;
    if (g == C) return M - money;
    if (memo[g][money] != -1)
        return memo[g][money]; // avaliar linha g com dinheiro money (cada
     caso pensavel)
```

```
    int ans = -1;
    for (int k = 1; k <= price[g][0]; ++k)
        ans = max(ans, dp(g + 1, money - price[g][k]));
    return memo[g][money] = ans;
}

int main() {
    int TC;
    scanf("%d", &TC);
    while (TC--)
    {
        scanf("%d %d", &M, &C);
        for (int g = 0; g < C; ++g)
        {
            scanf("%d", &price[g][0]); // store k in price[g][0]
            for (int k = 1; k <= price[g][0]; ++k)
                scanf("%d", &price[g][k]);
        }
        memset(memo, -1, sizeof memo); // TOP-DOWN: init memo
        if (dp(0, M) < 0)
            printf("no solution\n"); // start the top-down DP
        else
            printf("%d\n", dp(0, M));
    }
    return 0;
}
```

10.3 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;
    }
}
```

10.4 Mochila

```
// Description: Problema da mochila 0-1: retorna o valor maximo que pode
    ser carregado
// Complexidade: O(n*capacidade)

const int MAX_QNT_OBJETOS = 60; // 50 + 10
const int MAX_PESO_OBJETO = 1010; // 1000 + 10

int n, memo[MAX_QNT_OBJETOS][MAX_PESO_OBJETO];
vi valor, peso;

int mochila(int id, int remW) {
    if ((id == n) || (remW == 0)) return 0;
    int &ans = memo[id][remW];
    if (ans != -1) return ans;
    if (peso[id] > remW) return ans = mochila(id+1, remW);
    return ans = max(mochila(id+1, remW), valor[id]+mochila(id+1, remW-
    peso[id]));
}

void solve() {

    memset(memo, -1, sizeof memo);

    int capacidadeMochila; cin >> capacidadeMochila;

    f(i,0,capacidadeMochila) { memo[0][i] = 0; } // testar com e sem essa
    linha

    cin >> n;

    valor.assign(n, 0);
    peso.assign(n, 0);

    f(i,0,n) {
      cin >> peso[i] >> valor[i];
    }

    cout << mochila(0, capacidadeMochila) << endl;

}
```

10.5 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);
}
```

10.6 Fibonacci

```
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);
}
```

10.7 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;
}
```

10.8 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;
}
```