Maratona

Container

```
var.clear(); // Remove todos elementos
```

```
bool vazio = var.empty(); // Vazio?
unsigned in tamanho = var.size(); // Tamanho

for (; it != var.end(); it++) //Loop
```

vector

```
#include <vector>
```

```
// Declaração
vector<TIPO> var;
vector<TIPO> var(QTD, VALOR);

var.push_back(VALOR);  // Adiciona elemento
var.pop_back(); // Remove elemento

vector<TIPO>::iterator it = var.begin(); //Iterador normal
vector<TIPO>::reverse_iterator rit = var.rbegin(); //Iterador reverso

var.insert(it, VALOR); // Insere na posição it
var.insert(it, QTD, VALOR); // Insere QTD VALORES na posição it
var.insert(it, ARRAY + 0, ARRAY + Tamanho); // Insere todos os valores de um array
.

TIPO primeiro = var.front(); // Primeiro elemento
TIPO ultimo = var.back(); // Ultimo elemento
```

deque << vector

```
#include <deque>
```

```
// Declaração
deque<TIPO> var;
var.push_front(VALOR); // Adiciona no inicio
var.pop_front(); // Remove do inicio
```

map

```
#include <map>
map<TIPO_KEY, TIPO_VALUE> var;

var.insert(make_pair(KEY, VALUE)); // Insere
var.erase(KEY); // Remove

TIPO_VALUE valor = var[KEY]; // Acessa mapa
map<int, int>::iterator key_value = var.find(KEY); // Busca elemento
bool existe = key_value != var.end(); // Existe?
```

multimap

```
#include <map>
multimap<TIPO_KEY, TIPO_VALUE> var;

var.insert(make_pair(KEY, VALUE)); // Insere
var.erase(KEY); // Remove todos os valores associados à KEY
multimap<TIPO_KEY, TIPO_VALUE>::iterator it = ...;
var.erase(it); // Remove uma entrada

multimap<int, int>::iterator key_value = var.find(KEY); // Busca alguma entrada
bool existe = key_value != var.end(); // Existe?

//Percorrimento das entradas de uma chave
for(multimap<TIPO_KEY, TIPO_VAL>::iterator it = var.lower_bound(KEY); it != var.u
pper_bound(KEY); it++){
    TIPO_KEY key = it->first;
    TIPO_VALUE value = it->second;
}
```

set

```
#include <set>
set<TIPO> var;

var.insert(ITEM); // Insere
var.erase(KEY); // Remove
bool existe = var.find(ITEM) != var.end(); // Contém?
```

queue

```
#include <queue>
queue<TIPO> var;

var.push(ITEM); // Insere
TIPO primeiro = var.front(); // Pega o primeiro sem remover
var.pop(); // Remove primeiro
```

priority queue

```
#include <queue>
priority_queue<TIPO> var; //Cria com comparador padrão Less<T>

class MyComparator{
    bool operator ()(const TIPO & v1, const TIPO & v2){...}
}

priority_queue<TIPO, std::vector<TIPO>, MyComparator> var; //Cria com comparador p
ersonalizado

var.push(ITEM); // Insere
TIPO primeiro = var.top(); // Pega o primeiro sem remover
var.pop(); // Remove primeiro
```

stream

```
istream& operator>>(istream &input, TIPO &p) {
  input >> p.VALOR1 >> p.VALOR2;
  return input;
}
ostream& operator<<(ostream &out, TIPO &p) {
  out << p.VALOR1 << " " << p.VALOR2;
  return out;
}


string a, b, c;
cin >> a;
cin.ignore(); // Discartar espaÃ§os
getline(cin, b);
cin >> c;
```

sstream

```
#include <sstream>
stringstream ss;
stringstream ss("text");
ss << "2"; // Input
int num;
ss >> num; // Output
string result = ss.str() // Tudo o que está no stream
```

iostream e iomanip

```
#include <iostream>
// input
char a, b, c;
istringstream ss("  123")'
ss >> skipws >> a >> b >> c;
cout << a << b << c; // '123'
iss.seekg(0);
ss >> noskipws >> a >> b >> c;
cout << a << b << c; // '  1'
// Float
cout.precision(5);
cout << f; // 3.14159
cout << fixed << n; // 2006.00000
cout << scientific << n; // 2.00600e+003
// Base
cout << uppercase << hex << n; //4D
cout << nouppercase << hex << n; // 4d
cout << showbase << hex << n; //0x4d
cout << dec << n;
cout << oct << n;
// Bool
cout << boolalpha << b; // true
cout << noboolalpha << b; // 1
// Justify
cout.width(6); cout << internal << n; // "-   77"
cout.width(6); cout << left << n;     // "-77   "
cout.width(6); cout << right << n;    // "   -77"

#include <iomanip>
// Float
cout << setprecision(5) << f; // 3.14159
// Base
cout << setbase(16) << n; // 4d
// Justify
cout << setw(6) << n; // "   -77"
// Fill
cout << setfill('x') << setw(6) << n; // "xxx-77"
```

algorithm

```
#include <algorithm>
```

```
// Básico
void imprime(int i) {cout << i << endl;}
bool menor(int i,int j) { return i<j; }
bool igual(int i,int j) { return i==j; }
bool par(int i){ return i%2==0; }
int dobrar(int i) { return 2*i; }
int aleatorio() { return rand()%100; }
vector<int> vet, vet2, ordenado;
// Considere <param> como opcional

// Percorre aplicando função
for_each(vet.begin(), vet.end(), imprime);

// Busca
vector<int>::iterator it;
it = find(vet.begin(), vet.end(), VALOR); // linear
it = find_if(vet.begin(), vet.end(), par); // linear com condição
it = binary_search(ordenado.begin(), ordenado.end(), VALOR, <menor>); // binária
it = lower_bound(ordenado.begin(), ordenado.end(), VALOR, <menor>); // ponteiro pa
ra o primeiro VALOR
it = upper_bound(ordenado.begin(), ordenado.end(), VALOR, <menor>); // ponteiro pa
ra o primeiro numero depois de VALOR
it = search(vet.begin(), vet.end(), vet2.begin(), vet2.end(), <igual>); // sublist
a
it = find_end(vet.begin(), vet.end(), vet2.begin(), vet2.end(), igual); // sublista
 no final
it = find_first_of(vet.begin(), vet.end(), vet2.begin(), vet2.end(), <igual>); //
um dos resultados
it = adjacent_find(vet.begin(), vet.end(), vet2.begin(), vet2.end(), <igual>); //
elementos repetidos
it = search_n(vet.begin(), vet.end(), QTD, VALOR); // QTD VALORES seguidos

pair<vector<int>::iterator,vector<int>::iterator> par;
par = mismatch(vet.begin(), vet.end(), vet2.begin()); // Primeira diferença
par = equal_range(ordenado.begin(), ordenado.end(), VALOR, <menor>); // Subconjunt
o de VALORES

// Contar
int c = count(vet.begin(), vet.end(), 10);
int c = count_if(vet.begin(), vet.end(), par);

// Igualgdade
bool igual = equal(vet.begin(), vet.end(), vet2.begin(), <igual>);

// Copiar
vet2.resize(vet.size());
copy(vet.begin(), vet.end(), vet2.begin());
copy_backward(vet.begin(), vet.end(), vet2.end()); // inverso
```

```
// Swap
swap(vet, vet2); // Swap variavel
swap_ranges(vet.begin() + 1, vet.end() - 1, vet2.begin()); // Swap parte do vetor
iter_swap(vet.begin(), vet2.begin() + 1); // Swap dentro de vetor.

// Transform
vet2.resize(vet.size());
transform(vet.begin(), vet.end(), vet2.begin(), dobrar); // mapeia dobrar para cada
 elemento de vet
transform(vet.begin(), vet.end(), vet2.begin(), result.begin(), soma); // mapeia so
ma para cada para <vet[i], vet2[i]>

// Substituir elementos
replace(vet.begin(), vet.end(), ORIGINAL, DESTINO);
replace_if(vet.begin(), vet.end(), par, DESTINO);
replace_copy(vet.begin(), vet.end(), vet2.begin(), ORIGINAL, DESTINO); // Resultado
 em vet2
replace_copy_if(vet.begin(), vet.end(), vet2.begin(), par, DESTINO); // Resultado e
m vet2

// Preencher
fill(vet.begin(), vet.begin() + 3, VALOR); // VALOR VALOR VALOR vet[4] vet[5] ...
fill_n(vet.begin(), 3, VALOR); // VALOR VALOR VALOR vet[4] vet[5] ...

// Gerar
struct c_unique {
  int current;
  c_unique() {current=0;}
  int operator()() {return ++current;}
} UniqueNumber;

generate(vet.begin(), vet.end(), UniqueNumber);
generate_n(vet.begin(), 3, aleatorio);

// Remover elementos
it = remove(vet.begin(), vet.end(), 20); // "20 10 20 15" -> "10 15 ? ?". Retorna n
ovo end
it = remove_if(ver.begin(), vet.end(), par);
remove_copy(vet.begin(), vet.end(), vet2.begin(), 20);
remove_copy_if(vet.begin(), vet.end(), vet2.begin(), par);

// Remover duplicatas consecutivas
it = unique(vet.begin(), vet.end(), <igual>); // "20 20 10 20" -> "20 10 20 ?". Re
torna novo end
unique_copy(vet.begin(), vet.end(), vet2.begin(), <igual>);

// Inverter
reverse(vet.begin(), vet.end());
reverse_copy(vet.begin(), vet.end(), vet2.end());

// Circular
```

```
rotate(vet.begin(), vet.begin() + 2, vet.end()); // "1 2 3 4" -> "3 4 1 2"
rotate_copy(vet.begin(), vet.begin() + 2, vet.end(), vet2.begin());

// Embaralhar
random_shuffle(vet.begin(), vet.end(), <aleatorio>);

// Separar elementos
it = partition(vet.begin(), vet.end(), par); // begin - it: pares; it - end: impare
s
it = stable_partition(vet.begin(), vet.end(), par); // mantem a ordem

// Ordenar
sort(vet.begin(), vet.end(), <menor>);
stable_sort(vet.begin(), vet.end(), <menor>); // mantem ordem de elementos semelha
ntes
partial_sort(vet.begin(), vet.begin() + 2, vet.end(), <menor>); // "4 3 2 1" -> "1
 2 4 3"
partial_sort_copy(vet.begin(), vet.end(), vet2.begin(), vet2.end(), <menor>);
nth_element(vet.begin(), vet.begin() + 5, vet.end(), <menor>); // vet[0..4] < vet[
5] < vet[6..]

// Junção
merge(ordenado.begin(), ordenado.end(), ordenado2.begin(), ordenado2.end(), vet.beg
in());
inplace_merge(ordenado.begin(), ordenado.begin()+5, ordenado.end());
bool inclui = includes(ordenado.begin(), ordenado.end(), ordenado2.begin(), ordena
do2.end());
it = set_union(ord.begin(), ord.end(), ord2.begin(), ord2.end(), vet.begin(), <men
or>); // Merge sem repetição
it = set_intersection(ord.begin(), ord.end(), ord2.begin(), ord2.end(), vet.begin()
, <menor>); // Interseção
it = set_difference(ord.begin(), ord.end(), ord2.begin(), ord2.end(), vet.begin(),
<menor>); // Diferença
it = set_symmetric_difference(ord.begin(), ord.end(), ord2.begin(), ord2.end(), vet
.begin(), <menor>); // (A - B) U (B - A)

// Heap
make_heap(vet.begin(), vet.end());
pop_head(vet.begin(), vet.end()); vet.pop_back();
vet.push_back(99); push_heap(vet.begin(), vet.end());
cout << "max: " << vet.front();
sort_heap(vet.begin(), vet.end());

// Min/max
v = min(VALOR, VALOR2);
v = max(VALOR, VALOR2);
it = min_element(vet.begin(), vet.end(), <menor>);
it = max_element(vet.begin(), vet.end(), <menor>);

// Comparação lexicográfica
int cmp = lexicographical_compare(vet.begin(), vet.end(), vet2.begin(), vet2.end()
, <menor>);
```

```
// Permutações, crescente
sort(vet.begin(), vet.end());
do {
    for_each(vet.begin(), vet.end(), imprime);
} while (next_permutation(vet.begin(), vet.end()));


// Permutações, decrescente
sort(vet.begin(), vet.end());
reverse(vet.begin(), vet.end());
do {
    for_each(vet.begin(), vet.end(), imprime);
} while (prev_permutation(vet.begin(), vet.end()));
```

functional

```
#include <functional>
```

```
// Aritmética / lógica
transform(vet.begin(), vet.end(), vet2.begin(), result.begin(), plus<TIPO>());
minus<TIPO>(); multiplies<TIPO>(); divides<TIPO>(); modulus<TIPO>(); negate<TIPO>
();
logical_and<bool>(); logical_or<bool>(); logical_not<bool>();
// Comparação
it = mismatch(vet.begin(), vet.end(), vet2.begin(), equal_to<int>());
not_equal_to<TIPO>(); greater<TIPO>(); less<TIPO>(); greater_equal<TIPO>(); less_e
qual<TIPO>();
// Negação
count_if(vet.begin(), vet.end(), not1(par));
it = mismatch(vet.begin(), vet.end(), vet2.begin(), not2(equal_to<int>()));
// Bind
count_if(vet.begin(), vet.end(), bind1st(equal_to<int>(), 10));
count_if(vet.begin(), vet.end(), bind2nd(less<int>(), 10));
// Converte ponteiro em função
transform(vet.begin(), vet.end(), vet2.begin(), ptr_fun(atoi));
// Usar método de objeto
transform(vet.begin(), vet.end(), vet2.begin(), mem_fun(&string::length)); //se va
lores forem ponteiros
transform(vet.begin(), vet.end(), vet2.begin(), mem_fun_ref(&string::length)); //s
e valores forem valores
```

utility

```
#include <utility>
```

```
swap(a, b);
pair<TIPO, TIPO> x = make_pair(a, b);
x.first; x.second;
```

numeric

```
#include <numeric>
// Acumular resultado - reduce
int v = accumulate(vet.begin(), vet.end(), INIT, <plus<int>()>);
// Subtrair valores adjacentes
adjacent_difference(vet.begin(), vet.end(), vet2.begin(), <minus<int>()>);
// Acumular produtos entre adjacentes
int v = inner_product(vet.begin(), vet.end(), vet2.begin(), INIT, <plus<int>(), m
ultiplies<int>()>)
// Soma parcial
partial_sum(vet.begin(), vet.end(), vet2.begin(), <plus<int>()>); // "1 2 3 4" ->
"1 3 6 10"
```

string

```
#include <string>
string s;
string s("abc"); // construtor
string::iterator b = s.begin(), e = s.end(); // iteradores
int tamanho = s.size(); tamanho = s.length(); // tamanh
bool vazia = s.empty(); // vazia?
cout << c[1] << c.at(1); // acessar
s += "outra"; s.append("outra");
s.push_back('c');
char * cstr = s.c_str(); // converte para c string
// Busca
size_t posicao = s.find("valor", <POSICAO>); // primeira ocorrencia
size_t posicao = s.rfind("valor", <POSICAO>); // ultima ocorrencia
size_t posicao = s.find_first_of("aeiou"); // algum dos chars
size_t posicao = s.find_last_of("aeiou"); // algum dos chars
size_t posicao = s.find_first_not_of("aeiou"); // algum que não é um dos chars
size_t posicao = s.find_last_not_of("aeiou"); // algum que não é um dos chars
bool achou = posicao != string::npos;

// Substring
string s2 = s.substr(INICIO, <TAMANHO>);

// Comparar
int cmp = compare(<INICIO, TAMANHO>, outra, <INICIO, TAMANHO>);

// Substituir
s.replace(INICIO, TAMANHO, outra);
s.replace(INICIO, TAMANHO, outra, INICIO, TAMANHO);
s.replace(INICIO, TAMANHO, outra, TAMANHO);
s.replace(INICIO, TAMANHO, QUANTIDADE, CHAR);

// Ler linha
getline(cin, s);
```

cmath

```
//Funções trigonométricas - em radianos
double angle = 3.14;
double v = sin(angle) + cos(angle) + tan(angle) +
     acos(1.0) + // [0, pi]
     asin(1.0) + // [-pi/2,+pi/2]
     atan(1.0);  // [-pi/2,+pi/2]
double x,y;
v = atan2(y, x);  // [-pi,+pi]

v = sinh(angle) + cosh(angle) + tanh(angle); // Funções hiperbólicas

double PI = 355.0 / 113.0; //Aproximação de Pi

double base = 3, expoent = 10;
v = sqrt(25);           // Raiz quadrada
v = pow(base, expoent); // 3^10
v = exp(expoent);       // e^10
v = log(1234);          // Logaritmo nepteriano
v = log10(1234);        // Log na base 10

int integer;
double frac = modf(13.4, &integer); //Separa as partes inteira e decimal.

v = log2(1024); // Log na base 2, C++11 apenas.

v = ceil(12.3);  // 13
v = floor(12.3); // 12
v = abs(-3.4);   // 3.4, Módulo

v = fmod(5.3, 2.0); // 1.3, resto fracionário da divisão inteira

// Constantes
v = NAN;
v = HUGE_VAL; // ~= infinito
v = INFINITY; // C++11
```

C++ Básico

```
// Classe - private por padrão
class Simple {
  int i_;
public:
  Simple(int i) : i_(i) {}; // Constructor
  ~Simple(); // Destructor
  Simple(const Simple &); // Copy Constructor
  Simple & operator=(const Simple &); // Assignment Operator
  void print_i(); // método
};
```

```
// Struct - public por padrão
struct Simple {
  Simple(int i) : i_(i) {}; // construtor
  void print_i(); // método
private:
  int i_;
};

void Simple::print_i() {
  cout << i_ << endl;
}

// Alocação dinâmica
int * a = new int [3];
delete [] a;

Simplt * s = new Simple;
delete s;

// Parametros:
void foo(int a = 2); // Valor padrão
void foo

// Try catch
try {
  //código
  throw 2;
} catch (RangeError &re) {
  // específica
} catch (int &i) {
  // Exceção 2
} catch (...) {
  // Qualquer exceção
}

vor foo(); // pode lançar qualquer exceção
vor foo() throw(); // não lança exceção
vor foo() throw(int); // só lança exceção int


// Operadores
TIPO operator()(TIPO param1, TIPO param2, ...) // function call
TIPO& operator++() // ++valor
TIPO operator++(int) // valor++
TIPO& operator+=(const TIPO& outro) // +=
TIPO operator+(TIPO primeiro, const TIPO& segundo) // primeiro + segundo
bool operator<(const TIPO& primeiro, const TIPO& segundo) // p < s; único neces
sário
bool operator==(const TIPO& primeiro, const TIPO& segundo) // p == s; único nec
essário
const TIPO& operator[](size_t idx) const // var[idx]
```

limits

```
#include <limits>
// Exemplos para int
numeric_limits<TIPO>::min(); // -2147483648
numeric_limits<TIPO>::max(); // 21477483647
numeric_limits<TIPO>::is_signed(); // 1
numeric_limits<TIPO>::digits(); // 31 bits
numeric_limits<TIPO>::has_infinity; // 0
```

complex

```
#include <complex>

complex<double> x(REAL, IMAG);
real(x); imag(x); // Pega parte real e imaginária
abs(x); arg(x) // x = abs(x)*e^i*arg(x) rad
norm(x); // Norma de x
conjugate(x); // Conjugado

x = polar(2, 0.5); x = 2*e^i*0.5
```

Tipos inteiros

```
int a;                  // 32 bits,  [-2.147.483.648, 2.147.483.647]
unsigned int a;          // 32 bits,  [0, 4.294.967.295]
long long int a;         // 64 bits,  [-9.223.372.036.854.775.808, 9.223.372.036
.854.775.807]
unsigned long long int a; // 64 bits,  [0, 18.446.744.073.709.551.615]
float a;                // 32 bits,  [1.2E-38, 3.4E+38]
double a;               // 64 bits,  [2.3E-308, 1.7E+308]
long double a;           // 128 bits, [3.4E-4932, 1.1E+4932]
```

Java BigInteger

```
import java.math.BigInteger;

// Constantes
BigInteger.ZERO;
BigInteger.ONE;
BigInteger.TEN;

// Construtores
int base = 10;
BigInteger.valueOf(123456789);
BigInteger.valueOf("123456789");
```

```
BigInteger.valueOf("123456789", base);
byte [] binaryRepr; // Representação binária em C2 com bit mais significante à esq
uerda
BigInteger.valueOf(binaryRepr);

// Operações
BigInteger a, b, c;
int expoent = 0;
c = a.add(b);
c = a.subtract(b);
c = a.multiply(b);
c = a.divide(b);
c = a.mod(b);
c = a.pow(expoent);
BigInteger[] arr = a.divideAndRemainder(b);
BigInteger divider = arr[0], remainder = arr[1];

boolean iguais = a.equals(b);
int comparacao = a.compareTo(b);

//Conversão
int val = a.intValue();
long val = a.longValue();
float val = a.floatValue();
// etc
String strVal = a.toString();
```

Java IO

```
import java.util.Scanner;
```

```
Scanner s = new Scanner(System.in);

// Leitura de input
int a = s.nextInt();
String str = s.next(); // Próxima palavra, retorna "ab" para input "ab cd"
str = s.nextLine();    // Lê linha
BigInteger bigI = s.nextBigInteger();

//Checagem de fim de input
boolean b;
b = s.hasNextLine();
b = s.hasNextInt();
b = s.hasNext();

// ...
s.close(); // recomendado
```

Djakstra (custo mínimo)

```
def djakstra(src, graph):
    costs = [INFINITY for v in graph] #array
    parent = [None for v in graph]    #array
    to_visit = [src]                  #fila
    visited = set([src])              #conjunto
    costs[src] = 0

    while to_visit:
        v = to_visit.pop(0)
        for adj in graph.adjs(v):
            newCost = costs[v] + graph[v][adj]
            if newCost < costs[adj]:
                costs[adj] = newCost
                parent[adj] = v

            if not adj in visited:
                visited.add(adj)
                to_visit.append(adj)

    return costs, parent
```

Bellman-Ford (custo mínimo, aresta negativa)

```
def bellman_ford(origin, nodes, edges):
    dist = [float('inf')] * nodes
    pred = [-1] * nodes
    dist[origin], pred[origin] = 0, -2

    for i in range(nodes - 1):
        for edge in edges:
            u, v, weight = edge
            if dist[u] + weight < dist[v]:
                dist[v] = dist[u] + weight
                pred[v] = u

    for edge in edges:
        u, v, weight = edge
        if dist[u] + weight < dist[v]:
            return 'cicle'

    return dist, pred
```

Floyd-Warshall (custo mínimo, todos nós para todos os nós)

```python
def floyd_warshall(nodes, edges):
    dist = [[float('inf')] * nodes for _ in range(nodes)] # N x N
    next = [[-1] * nodes for _ in range(nodes)]
    for u, v, w in edges:
        dist[u][v] = w
        next[u][v] = v

    for k in range(nodes):
        for i in range(nodes):
            for j in range(nodes):
                if dist[i][k] + dist[k][j] < dist[i][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]
                    next[i][j] = next[i][k]
    return dist, next
```

BFS

```cpp
bool bfs(T src, T dest){
    queue<T> fila;
    fila.push_back(src);
    set<T> visitados;
    map<T, T> parent;

    while(!fila.empty()){
        T cur = fila.front();

        if(cur == dest){
            return true;
        }

        fila.pop();
        visitados.insert(cur);
        foreach(T adj in cur.adjacencias){
            if(!visitados.contains(adj)){ // Pseudo-code
                fila.push_back(adj);
                parent.insert(make_pair(cur, adj));
                visitados.insert(adj);
            }
        }
    }
    return false;
}
```

Prim (árvore geradora mínima)

```
def findMinEdge(reached_vertices, graph):
    best_edge = None
    for u in reached_vertices:
        for v in adjs(u, graph):
            if not v in reached_vertices:
                candidate = (u,v)
                if not best_edge or cost(candidate) < cost(best_edge):
                    best_edge = candidate
    return best_edge


def prim(graph):
    reached_vertices = set([0]) # qualquer vértice como inicial
    selected_edges = []

    while len(reached_vertices) != len(graph): # enquanto não atingir todos os vér
tices
        minEdge = findMinEdge(reached_vertices, graph)
        selected_edges.append(minEdge)
        reached_vertices = reached_vertices.union(set(minEdge))

    return selected_edges
```

Ford-Fulkerson (Fluxo máximo)

```
int fordFulkerson(int grapv[V}[V], int s, int t) {
    int u, v;
    int rGraph[V][V]; // Residual graph
    for (u = 0; u < V; v++)
        for(v = 0; v < V; v++)
            rGraph[u][v] = graph[u][v];

    int parent[V]; // armazena caminho, BFS deve usar
    int max_flow = 0;

    // Enquanto tem caminho no grafo residual
    while(bfs(rGraph, s, t, parent)) {
        int path_flow = INT_MAX;
        for (v = t; v != s; v = parent[v]) {
            u = parent[v]
            path_flow = min(path_flow, rGraph[u][v]);
        }
        for (v = t; v != s; v = parent[v]) {
            u = parent[v];
```

```
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }

        max_flow += path_flow;
    }
    return max_flow;
}
```

Longest Common Subsequence

```
def lcs(a, b):
    X = [0]*(max(len(b), len(a)) + 1)
    Y = [0]*(max(len(b), len(a)) + 1)
    for i in range(len(a), -1, -1):
        for j in range(len(b), -1, -1):
            if i >= len(a) or j >= len(b):
                X[j] = 0
            elif a[i] == b[j]:
                X[j] = 1 + Y[j + 1]
            else:
                X[j] = max(Y[j], X[j + 1])
        Y = [x for x in X] # Copiar X
    return X[0]
```

Logest Increasing Subsequence

```
def lis(sequence):
    dp = [1] * len(sequence)
    prev = [1] * len(sequence)
    max_len, best_end = 1, 0

    for i in range(1, len(sequence)):
        dp[i] = 1
        prev[i] = -1

        for j in range(i - 1, -1, -1):
            if dp[j] + 1 > dp[i] and sequence[j] < sequence[i]:
                dp[i] = dp[j] + 1
                prev[i] = j
            if dp[i] > max_len:
                best_end = i
                max_len = dp[i]
    return max_len, best_end, prev
```

Mochila 01 (Prog Dinâmica)

```python
def mochila01(values, weights, W):
    item_count = len(values)
    table = [[0 for col in range(W+1)] for row in range(item_count+1)]

    # tabela (items+1) x (W+1)
    # 1a linha e 1a coluna = 0

    for capacity in range(1, W + 1):
        for item in range(1, item_count + 1):
            item_weight = weights[item-1]
            item_value = values[item-1]
            previous_value = table[item-1][capacity]

            if capacity >= item_weight:
                table[item][capacity] = max(table[item-1][capacity - item_weight]
+ item_value, previous_value)
            else:
                table[item][capacity] = previous_value

    return table[item_count][W]
```

Template de programação dinâmica

```python
def progDinamica(caso, lookup_table):
    if caso_basico:
        return calcula_caso_basico()
    elif lookup_table.contains(caso):
        return lookup_table[caso]
    else:
        resp_proximo_caso = progDinamica(caso+1, lookup_table)
        return lookup_table[caso] = combina(caso, resp_proximo_caso)
```

Modular exponentiation ((base ^ exp) % mod)

```c
int modular_pow(int base, int exp, int modulus) {
    int result = 1;
    base = base % modulus;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % modulus;
        }
        exp = exp >> 1;
        base = (base * base) % modulus;
    }
    return result;
}
```

Sieve of Eratosthenes (todos os primos menores do que N)

```
def sieve_of_eratosthenes(n):
    primes = []
    used = [0]*(n + 1)
    for i in range(2, n + 1):
        if not used[i]:
            primes.append(i)
            for j in range(i**2, n + 1, i):
                used[j] = 1
    return primes
```

N-ésimo primo:

```
#include <cmath>
int overestimate_prime(int n) {
    return (int) n*log(n) + n*log(log(n));
}
sieve_of_eratosthenes(overestimate_prime(n))[n - 1];
```

Contagem:

```
Permutações: (número de ordenações de n elementos em um conjunto de n elementos)
    P(n) = n!
k-Permutações: (número de ordenações de k elementos em um conjunto de n elementos)
    P(n,k) = n! / (n - k)!
Combinações: (número de subconjuntos de tamanho k formados a partir de um conjunto
com n elementos)
    C(n,k) = n! / (k! * (n - k)!)
    C(n,k) = C(n-1,k-1) + C(n-1,k)
    C(k,k) = 1
    C(n-k,0) = 1
Partições: (generalizam combinações para r conjuntos)
    Part = n! / (n1! * n2! * ... * nr!)

N-ésimo número de Catalan:
    Cn = C(2n,2) / (n + 1)
    ~ Quantas maneiras é possível construir fórmulas balanceadas a partir de n pare
s de parênteses
    ~ Quantas triangulações são possíveis em um polígono convexo

Números Eulerianos: (nútmero de permutações de tamanho n com exatamente k sequência
s ascendentes)
    E(n,k) = (n - k + 1) * E(n-1,k-1) + k * E(n-1,k)
    E(n-k,0) = 1
    E(k+1,k) = 1
```

Fibonacci

```
def fib_num(n):
    sq5 = sqrt(5)
    G = (1 + sq5)/2.0
    psi = 1 - G
    return (G**n - psi**n)/sq5
```