Guia Prático - PRO Maratona

Usando o terminal

Compilador

Para compilar o programa, usamos o comando:

`g++ -lm -pipe -O2 -Wall -std=c++20 -g <programa.cpp> -o Main`

onde, o `<programa.cpp>` refere-se ao nome do arquivo que queremos compilar, exemplo:

`g++ -lm -pipe -O2 -Wall -std=c++20 -g a.cpp -o Main`

para compilar o arquivo `a.cpp`

Executando o programa

Para executar o programa gerado pelo compilador, usamos o comando: `./Main`

Podemos usar um arquivo de texto para usar de entrada no programa.

Assim, o comando fica: `./Main < <entrada.txt>`

onde `<entrada.txt>` refere-se ao nome do arquivo que queremos usar de entrada.

Além disso, se quisermos jogar a saída da execução do progama para outro arquivo de texto, podemos usar o comando: `./Main < <entrada.txt> > <saida.txt>`

onde `<entrada.txt>` refere-se ao nome do arquivo que queremos usar de entrada e `<saida.txt>` refere-se ao nome do arquivo que queremos usar de saida do programa.

Verificando casos de teste

Para verificar se os casos de teste deram certo ou há alguma incongruência, usamos o comando: `diff -y <saida.txt> <resposta.txt>`

onde `<saida.txt>` refere-se ao nome do arquivo gerado pela execução do programa e `<resposta.txt>` refere-se ao nome do arquivo que contém a saída esperada do exercício.

Script para gerar arquivos de entrada e saída

Podemos automatizar a criação dos arquivos de entrada e saída que utilizaremos na maratona inteira. Assim pode-se criar os arquivos de script abaixo.

C++

Crie um arquivo `gerar_cpp.sh` com o conteúdo abaixo:

```
#!/bin/bash

# Letra 'a' na tabela ASCII
cod_letra=97
```

```
for i in $(seq 1 $1);
do
    # Escolhe a letra atual
    letra=$(printf "\\$(printf '%03o' "$cod_letra")")

    # Cria o arquivo cpp do exercicio atual
    cp template.cpp ${letra}.cpp

    for j in $(seq 1 $2);
    do
        # Cria os j's arquivos de entrada
        touch ${letra}${j}i.txt

        # Cria os j's arquivos de saída
        touch ${letra}${j}o.txt
    done
    ((cod_letra++))
done
```

Python

Crie um arquivo `gerar_py.sh` com o conteúdo abaixo:

```
#!/bin/bash

# Letra 'a' na tabela ASCII
cod_letra=97

for i in $(seq 1 $1);
do
    # Escolhe a letra atual
    letra=$(printf "\\$(printf '%03o' "$cod_letra")")

    # Cria o arquivo py do exercicio atual
    cp template.py ${letra}.py

    for j in $(seq 1 $2);
    do
        # Cria os j's arquivos de entrada
        touch ${letra}${j}i.txt

        # Cria os j's arquivos de saída
        touch ${letra}${j}o.txt
    done
```

```
    ((cod_letra++))
done
```

Script para testar código

Podemos automatizar os casos de teste disponíveis na maratona, verificando a execução do código com a saída esperada. Assim, pode-se criar os arquivos de script abaixo.

C++

Crie um arquivo `testar_cpp.sh` com o conteúdo abaixo:

```
#!/bin/bash

# Compilar o programa
g++ -lm -pipe -O2 -Wall -std=c++20 -g $1.cpp -o Main

for i in $(seq 1 $2);
do
    echo "Caso de Teste $i"

    # Executa o programa com a entrada do arquivo 'i.txt' e gera a saída no arquivo 'res.txt'
    ./Main < ${1}${i}i.txt  > res$i.txt

    # Procurar as diferenças entre a saida redirecionada do programa 'res.txt' com a saida esperada do exercicio 'o.txt'
    diff -y res$i.txt ${1}${i}o.txt
done
```

Python

Crie um arquivo `testar_py.sh` com o conteúdo abaixo:

```
for i in $(seq 1 $2);
do
    echo "Caso de Teste $i"

    # Executa o programa com a entrada do arquivo 'i.txt' e gera a saída no arquivo 'res.txt'
    python $1.py < ${1}${i}i.txt  > res$i.txt

    # Procurar as diferenças entre a saida redirecionada do programa 'res.txt' com a saida esperada do exercicio 'o.txt'
    diff -y res$i.txt ${1}${i}o.txt
done
```

> Observação
>
> Para que o computador deixe executar o script, é necessário dar **permissão de execução** para o arquivo. Podemos fazer isso no gerenciador de arquivo ou rodando o comando `chmod +x <arquivo.sh>` sendo `'<arquivo.sh>'` o nome do arquivo do script.
>
> Exemplo:
>
> - `chmod +x gerar_cpp.sh` e `chmod +x testar_cpp.sh` para `c++`;
> - `chmod +x gerar_py.sh` e `chmod +x testar_py.sh` para `python`;

> Dica
>
> Para executar os scripts, basta chamá-los no terminal utilizando o ponto e barra, `./`.
>
> Exemplos:
>
> - `./gerar_cpp.sh 5 3` para gerar 5 arquivos de exercício em `c++` com 3 casos de teste cada um;
> - `./testar_cpp.sh b 3` para compilar e testar o arquivo `b.cpp` e rodar 3 casos de teste do mesmo;
> - `./gerar_py.sh 5 3` para gerar 5 arquivos de exercício em `python` com 3 casos de teste cada um;
> - `./testar_py b 3` para interpretar e testar o arquivo `b.py` e rodar 3 casos de teste do mesmo;

Abreviações

É comum existir código repetido em muitos exercícios, por isso, toma-se espaço para montar algumas abreviações usuais.

> Dica
>
> O `cache` pode variar conforme o exercício, sendo necessário, inclusive, usar mais de um cache e/ou usar cache multivalorado.
>
> Exemplo:
>
> - `int primos[20];`: para números primos;
> - `ll fatoriais[50];`: para fatoriais grandes;
> - `map<string, int> alunos;`: para guardar nome e idade de alunos;

- `vi arvore;`: para usar uma árvore binária em ordem topológica, onde `arvore[i]` é o nó atual, `arvore[2*i + 1]` o nó esquerda e `arvore[2*i + 2]` o nó direita;

```cpp
// Biblioteca com a coletânea de bibliotecas padrão
#include<bits/stdc++.h>

// Desabilita sincronização entre cin/cout com scanf/printf para melhorar a velocidade
#define FASTIO ios_base::sync_with_stdio(false), cin.tie(0), cout.tie(0)

// Limpa a variavel 'vetor'
#define CLEAR(vetor) memset(vetor, 0, sizeof vetor)

// Printa o nome e valor de uma variavel 'var'
#define DEBUG(var) cout << #var << ": '" << var << "'\n"

// Abreviação de um laço de repetição simples
#define REP(i, a, b) for(int i=a; i<=b; i++)

// Abreviação para "\n"
#define ENDL "\n"

// Elementos comum em pair e map
#define F first
#define S second

// Funções comum em vector, list, set e pair
#define PB push_back
#define MP make_pair

// Convenção de tipos com inteiro
typedef long long ll;
typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

// Diz para o compilador que estamos usando a referência 'std'
// Assim, não precisamos escrever 'std::cin', 'std::cout', 'std::vector<int>', ...
using namespace std;

// Inteiro muito grande = infinito
const int POS_INF 1e9;
```

```
// Inteiro muito pequeno = -infinito
const int NEG_INF 1e-9;

// Inteiro muito grande para fazer aritmética modular
const int MOD = 1e9+7;

int cache[POS_INF];

// Configuração de cache
memset(cache, -1, sizeof cache); // Inicializa o cache com -1 em todas as posições
```

Simplificação de operações

Há operações que serão frequentemente utilizadas. Assim, abaixo está o código reduzido da opreção.

```
resposta = a ? b : c; // if (a) resposta = b; else resposta = c;

resposta += valor; // resposta = resposta + valor (funciona com outros operadores)

resposta = (int)((double)numero + 0.5); // arredondamento para inteiro mais próximo

resposta = min(resposta, valor); // computação de mínimo e máximo

indice = (indice + 1) % tamanho; // indice++; if (indice >= n) indice = 0;

indice = (indice + tamanho - 1) % tamanho; // indice--; if (indice < 0) indice = tamanho - 1;
```

Template

É comum utilizar-se de um arquivo base, contendo todas as abreviações necessárias, para todos os arquivos de exercícios. Este arquivo, portanto, é um template para a criação dos demais. Assim, com as abreviações supracitadas, temos os arquivos `template.cpp` e `template.py` abaixo:

> Observação
>
> Para exercícios **extremamente** rápidos de resolver, pode-se ignorar a criação do arquivo template e dos scripts para não perder tempo hábil.
>
> Após concluir a submissão dos exercícios rápidos, pode-se criar os arquivos de template e scritps para reduzir o tempo de testes nos códigos.

C++

```cpp
#include<bits/stdc++.h>

#define FASTIO ios_base::sync_with_stdio(false), cin.tie(0), cout.tie(0)
#define CLEAR(vetor) memset(vetor, 0, sizeof vetor)
#define DEBUG(var) cout << #var << ": '" << var << "'\n"
#define REP(i, a, b) for(int i=a; i<=b; i++)
#define ENDL "\n"

#define F first
#define S second
#define PB push_back
#define MP make_pair

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
typedef vector<pii> vii;

const int POS_INF = 1e9;
const int NEG_INF = 1e-9;
const int MOD = 1e9+7;

int cache[POS_INF];

int main() {
    FASTIO;



    return 0;
}
```

Python

```python
import math, collections, functools
from functools import cache

# Ler uma string, removendo os espaços
istr = lambda: input().strip()

# Ler um número, removendo os espaços da entradaa
```

```python
inum = lambda: int(input().strip())

# Ler um dicionario de números
imap = lambda: map(int,input().strip().split())

# Ler uma lista de números
ilist = lambda: list(map(int, input().strip().split()))

DEBUG = lambda x: print(f"'{x}'")

INF_POS = 1e9
INF_NEG = 1e-9
MOD = 1e9+7

# Dicionario para guardar informações no formato chave-valor
memo = {}

# main
```

Entrada e Saída

Há vários exemplos de casos de entrada e saída que deve-se atentar na resolução de problemas.

Número de casos de testes

```cpp
int casos; cin >> casos;
int a, b;

while(casos--) {
    cin >> a >> b;
    cout << (a+b) << ENDL;
}
```

Entrada	Saída
3	3
1 2	12
5 7	9
6 3	

Entrada encerra em 0

```
int a, b;

while(cin >> a >> b, (a || b)) {
    cout << (a+b) << ENDL;
}
```

Entrada	Saída
1 2	3
5 7	12
6 3	9
0 0	

Fim de arquivo

```
int caso=0;
int a, b;

while((cin >> a >> b) != EOF) {
    cout << "Caso " << caso++ << ": ";
    cout << (a+b) << ENDL;
}
```

Entrada	Saída
1 2	Caso 1: 3
5 7	Caso 2: 12
6 3	Caso 3: 9

Número variado de Entrada

```
int quantidade;
int soma, numero;

while((cin >> quantidade) != EOF) {
    soma = 0;

    while(quantidade--) {
        cin >> numero;
        soma += numero;
    }
```

```
    cout << soma << ENDL;
}
```

Entrada	Saída
1 1	1
2 3 4	7
3 8 1 1	10
4 7 2 9 3	21
5 1 1 1 1 1	5

Operações Binárias

p	q	!p	p & q	p | q	p ^ q
1	1	0	1	1	0
1	0	0	0	1	1
0	1	1	0	1	1
0	0	1	0	0	0

> Observação
>
> Operações binárias funcionam bit à bit, com “exceção” do operador `!` (not);
>
> Operações binárias são operadores básicos, ou seja, o mesmo código funciona em python, java ou c++.

> Lembrando
>
> Um `int` possui `4 bytes`, ou seja, `32 bits`
>
> Podemos classificar as operações como:
>
> - `!p` lê-se **não p**, ou seja, **o valor oposto de p**
> - `p & q` lê-se **p e q**, ou seja, **os valores verdadeiros de p e q, ao mesmo tempo?**
> - `p | q` lê-se **p ou inclusivo q**, ou seja, **pelomeno um dos valores é verdadeiro entre p e q?**
> - `p ^ q` lê-se **p ou exclusivo q**, ou seja, **o valor de p é diferente de q?**

```
int p = 10; // 1010 em binário
int q = 5; // 0101 em binário
```

```
!p; // !(1010) => 0000 em binário, ou seja, 0 em decimal
p & q; // 1010 & 0101 => 0000 em binário, ou seja, 0 em decimal
p | q; // 1010 | 0101 => 1111 em binário, ou seja, 15 em decimal
p ^ q; // 1010 ^ 0101 => 1111 em binário, ou seja, 15 em decimal
```

Bit Shift

Podemos precisar fazer a mudança dos `x` bits de posição, para esquerda ou direita, para isso usamos o operador *bit shift*. Essa mudança pode ser feita no formato:

- `numero << x`: para mudar `x` bits para a esquerda de `numero`;
- `numero >> x`: para mudar `x` bits para a direita de `numero`;

> Dica
>
> Fazer uma mudança de bits para a esquerda é, basicamente, multiplicar por 2

> Dica
>
> Fazer uma mudança de bits para a direita é, basicamente, dividir por 2

```
int p = 3; // 0011 em binário

p << 1; // 0011 mudando 1 bit para a esquerda = 0110 em binário, ou seja, 6 em decimal
p >> 1; // 0011 mudando 1 bit para a direita = 0001 em binário, ou seja, 1 em decimal
```

Estrutura de Dados

É importante o correto uso de estrutura de dados para usar de forma efetiva a memória disponível no exercício.

> Observação
>
> As estruturas abaixo são implementações em `c++`.
>
> No `python`, as estruturas abaixo podem ser utilizadas manipulando a `list` interna da linguagem, ou `dict` no caso da estrutura `map`.

C++

Vector

É uma estrutura de dados de tamanho variável, funcionando análogo a um vetor “sem limitação de memória” ou uma lista de valores.

```cpp
// Cria vetor de inteiros vazio
vector<int> vet1; // []

// Cria vetor de string com 5 valores iniciais sendo eles ""
vector<string> vet2 (5, ""); // ["", "", "", "", ""]

vet2[0] = "testemunha"; // ["testemunha", "", "", "", ""]
vet2[1] = "teste"; // ["testemunha", "teste", "", "", ""]
vet2[2] = "testão"; // ["testemunha", "teste", "testão", "", ""]
vet2[3] = "testinho"; // ["testemunha", "teste", "testão", "testinho", ""]

vet2.size(); // 5, que é o tamanho de 'vet2'
vet2.empty(); // false, pois 'vet2' contém elementos

// Adiciona elementos no final de 'vet1'
vet1.push_back(1); // [1]
vet1.push_back(2); // [1, 2]
vet1.push_back(3); // [1, 2, 3]
vet1.push_back(6); // [1, 2, 3, 6]
vet1.push_back(24); // [1, 2, 3, 6, 24]

// Remove o 3o elemento de 'vet2'
vet2.erase(vet2.begin() + 2); // ["testemunha", "teste", "testinho", ""]

// Remove o intervalo [0, 1] de 'vet2'
vet2.erase(vet2.begin(), vet2.begin() + 2); // ["testinho", ""]

// Ordena 'vet2' de forma crescente
sort(vet2.begin(), vet2.end()); // ["", "testinho"]

// Ordena 'vet1' de forma decrescente
sort(vet1.rbegin(), vet1.rend()); // [24, 6, 3, 2, 1]

// For: executa os comandos de 1 em 1 até chegar no tamanho de 'vet1'
for(int i=0; i<vet1.size(); i++) {
    cout << vet1[i] << " ";
}
cout << "\n";

// For com iterator: executa para cada elemento dentro de 'vet1', de forma
crescente
// it é declarado dinamicamente, c++ "descobre" seu tipo
'vector<int>::iterator'
```

```cpp
for(auto it = vet1.begin(); it != vet1.end(); it++) {
    cout << *it << " ";
}
cout << "\n";

// For com iterator: executa para cada elemento dentro de 'vet1', de forma decrescente
// it é declarado dinamicamente, c++ "descobre" seu tipo 'vector<int>::iterator'
for(auto it = vet1.rbegin(); it != vet1.rend(); it++) {
    cout << *it << " ";
}
cout << "\n";

// For-each: executa os comandos para cada elemento dentro de 'vet2'
// i é declarado dinamicamente, c++ "descobre" o seu tipo 'int'
for (auto i: vet2) {
    cout << i << " ";
}
cout << "\n";

// Limpa todo o conteúdo de 'vet1', esvaziando-o no processo
vet1.clear(); // []

// Remove cada elemento de 'vet2' até que fique vazio
while(!vet2.empty()) {
    vet2.pop_back(); // Remove o último elemento de 'vet2'
}
```

Set

O Conjunto é uma estrutura de dados que funciona como uma lista, mas não mantém valores iguais. Pode ser usada `set` ou `unordered_set` a depender da obrigatoriedade por ordenação dos dados.

```cpp
set<int> conjunto1 ({ 1, 2, 2, 1, 6, 5, 6, 3, 1, 1, 1 }); // {1, 2, 3, 5, 6}
unordered_set<int> conjunto2; // {}

conjunto1.size(); // 5, que é tamanho de 'conjunto1'
conjunto2.empty(); // true, pois 'conjunto2' não contém elementos

// Tenta adicionar valores em 'conjunto2'
conjunto2.insert(10); // {10}
conjunto2.insert(30); // {30, 10}
conjunto2.insert(20); // {20, 30, 10}
```

```cpp
conjunto2.insert(10); // {20, 30, 10}
conjunto2.insert(10); // {20, 30, 10}
conjunto2.insert(50); // {50, 20, 30, 10}
conjunto2.insert(40); // {40, 50, 20, 30, 10}
conjunto2.insert(60); // {60, 40, 50, 20, 30, 10}

// Tenta remover o valor 20 de 'conjunto2'
conjunto2.erase(20); // {60, 40, 50, 30, 10}

// Procura se existe o valor 2 em 'conjunto1'
conjunto1.count(2); // 1, pois este valor existe em 'conjunto1'

// Procura o iterator do valor 2 em 'conjunto1'
conjunto1.find(2); // set<int>::iterator apontando para 2

// Procura o iterator do maior valor menor ou igual à 3 em 'conjunto1'
conjunto1.lower_bound(3); // set<int>::iterator apontando para 3

// Procura o iterator do menor valor maior ou igual à 3 em 'conjunto1'
conjunto1.upper_bound(3); // set<int>::iterator apontando para 5

// For com iterator: executa para cada elemento dentro de 'conjunto1', de
forma crescente
// it é declarado dinamicamente, c++ "descobre" seu tipo
'set<int>::iterator'
for(auto it = conjunto1.begin(); it != conjunto1.end(); it++) {
    cout << *it << " ";
}
cout << "\n";

// For com iterator: executa para cada elemento dentro de 'conjunto1', de
forma decrescente
// it é declarado dinamicamente, c++ "descobre" seu tipo
'set<int>::iterator'
for(auto it = conjunto1.rbegin(); it != conjunto1.rend(); it++) {
    cout << *it << " ";
}
cout << "\n";

// For-each: executa os comandos para cada elemento dentro de 'conjunto1'
// i é declarado dinamicamente, c++ "descobre" o seu tipo 'int'
for (auto i: conjunto1) {
    cout << i << " ";
```

```
}
cout << "\n";

// Limpa todo o conteúdo de 'conjunto1', esvaziando-o no processo
conjunto1.clear(); // {}
```

Deque

O deque é uma estrutura de dados parecido com o vector, porém temos acesso direto às extremidades, inicio e fim.

```
deque<int> deque1; // []

deque<string> deque2 (2, "abc"); // ["abc", "abc"]

deque2.size(); // 2, que é o tamanho de 'deque2'
deque1.empty(); // true, pois 'deque1' não contém elementos

deque2[0] = ""; // ["", "abc"]
deque2[1] += "def"; // ["", "abcdef"]

// Pega o elemento no início do 'deque2'
deque2.front(); // ""

// Pega o elemento no final do 'deque2'
deque2.back(); // "abcdef"

// Adicionando elementos no final do 'deque1'
deque1.push_back(1); // [1]
deque1.push_back(2); // [1, 2]
deque1.push_back(3); // [1, 2, 3]

// Adicionando elementos no início do 'deque1'
deque1.push_front(4); // [4, 1, 2, 3]
deque1.push_front(5); // [5, 4, 1, 2, 3]
deque1.push_front(6); // [6, 5, 4, 1, 2, 3]

// Removendo elementos do final do 'deque1'
deque1.pop_back(); // [6, 5, 4, 1, 2]
deque1.pop_back(); // [6, 5, 4, 1]

// Removendo elementos do início do 'deque1'
deque1.pop_front(); // [5, 4, 1]
deque1.pop_front(); // [4, 1]
```

```
// For com iterator: executa para cada elemento dentro de 'deque1', de forma crescente
// it é declarado dinamicamente, c++ "descobre" seu tipo 'deque<int>::iterator'
for(auto it = deque1.begin(); it != deque1.end(); it++) {
    cout << *it << " ";
}
cout << "\n";

// For com iterator: executa para cada elemento dentro de 'deque1', de forma decrescente
// it é declarado dinamicamente, c++ "descobre" seu tipo 'deque<int>::iterator'
for(auto it = deque1.rbegin(); it != deque1.rend(); it++) {
    cout << *it << " ";
}
cout << "\n";

// For-each: executa os comandos para cada elemento dentro de 'deque1'
// i é declarado dinamicamente, c++ "descobre" o seu tipo 'int'
for (auto i: deque1) {
    cout << i << " ";
}
cout << "\n";

// Limpa todo o conteúdo de 'deque1', esvaziando-o no processo
deque1.clear(); // []

while(!deque2.empty()) {
    deque2.pop_front();
}
```

Queue

A fila, ou queue, é uma estrutura de dados no formato **FIFO (First In, First Out)**, ou seja, **o primeiro elemento que entra é o primeiro elemento que sai** enquanto que **o último elemento que entra é o último elemento que sai**.

```
queue<string> fila; // []

fila.size(); // 0, que é o tamanho de 'fila'
fila.empty(); // true, pois 'fila' não contém elementos
```

```
// Inserindo elementos NO FINAL de 'fila'
fila.push("Jorge"); // ["Jorge"]
fila.push("Amado"); // ["Jorge", "Amado"]
fila.push("Carlos"); // ["Jorge", "Amado", "Carlos"]
fila.push("Batista"); // ["Jorge", "Amado", "Carlos", "Batista"]
fila.push("Ricardo"); // ["Jorge", "Amado", "Carlos", "Batista", "Ricardo"]
fila.push("Eletro"); // ["Jorge", "Amado", "Carlos", "Batista", "Ricardo",
"Eletro"]

// Removendo elementos NO INÍCIO de 'fila'
fila.pop(); // ["Amado", "Carlos", "Batista", "Ricardo", "Eletro"]
fila.pop(); // ["Carlos", "Batista", "Ricardo", "Eletro"]

// Pega o elemento no início da 'fila'
fila.front(); // "Carlos"

// Pega o elemento no final da 'fila'
fila.back(); // "Eletro"

while(!fila.empty()) {
    fila.pop();
}
```

Stack

A pilha, ou stack, é uma estrutura de dados no formato **LIFO (Last In, First Out)**, ou seja, **o primeiro elemento que entra é o último elemento que sai** enquanto que **o último elemento que entra é o primeiro elemento que sai**.

```
stack<string> pilha;

pilha.empty(); // true, pois a 'pilha' não tem elementos
pilha.size(); // 0, pois a 'pilha' está vazia

// Inserindo elementos NO TOPO de 'pilha'
pilha.push("Jorge"); // ["Jorge"]
pilha.push("Amado"); // ["Amado", Jorge"]
pilha.push("Carlos"); // ["Carlos", "Amado", "Jorge"]
pilha.push("Batista"); // ["Batista", "Carlos", "Amado", "Jorge"]
pilha.push("Ricardo"); // ["Ricardo", "Batista", "Carlos", "Amado", "Jorge"]
pilha.push("Eletro"); // ["Eletro", "Ricardo", "Batista", "Carlos", "Amado",
"Jorge"]

// Removendo elementos NO TOPO de 'pilha'
pilha.pop(); // ["Ricardo", "Batista", "Carlos", "Amado", "Jorge"]
```

```
pilha.pop(); // ["Batista", "Carlos", "Amado", "Jorge"]

// Pega o elemento NO TOPO da 'pilha'
pilha.top(); // "Batista"

while(!pilha.empty()) {
    pilha.pop();
}
```

Map

O map é uma estrutura de dados que permite definir tanto o tipo da chave quanto o tipo do valor armazenado, semelhante ao dicionário do python.

> Observação
>
> Para a estrutura de dados `map`, seu `iterator` tem 2 atributos:
>
> - `first`: guarda a `chave`;
> - `second`: guarda o `valor`;

> Dica
>
> Quando usamos o método `find`, precisamos do operador `->` para acessar os atributos `first` e `second`.
>
> Se der problema, tente utilizar o operador `.` para acessar os atributos `first` e `second`.

```
map<string, int> dicionario; // {}

dicionario.empty(); // true, pois 'dicionario' não tem elementos
dicionario.size(); // 0, pois 'dicionario' está vazio

// Adicionando valores de 'dicionario'
dicionario["vermelhos"] = 5; // {"vermelhos": 5}
dicionario["azuis"] = 10; // {"vermelhos": 5, "azuis": 10}
dicionario["verdes"] = 0; // {"vermelhos": 5, "azuis": 10, "verdes": 0}

// Removendo valores de 'dicionario'
dicionario.erase("azuis"); // {"vermelhos": 5, "verdes": 0}

// Procurando um valor dentro de 'dicionario'
auto valor1 = dicionario.find("verdes"); // map<string,int>::iterator
apontando para dicionario["verdes"]
auto valor2 = dicionario.find("rosas"); // map<string,int>::iterator
```

```
apontando para dicionario.end(), pois não encontrou

// Imprimindo na tela no formato { 'chave': 'valor' }
cout << "{ '" << valor1->first << "': '" << valor1->second << "' }" << ENDL;

dicionario.count("verdes"); // 1, pois este valor está dentro de
'dicionario'

// For-each: executa os comandos para cada elemento dentro de 'dicionario'
// i é declarado dinamicamente, c++ "descobre" o seu tipo
'map<string,int>::iterator'
for(auto chave_valor: dicionario) {
    cout << "'" << chave_valor.first << "': '" << chave_valor.second << "'"
<< ENDL;
}
```

String

A string, ou cadeia de caracteres, é uma estrutura de dados que permite agrupar letras. Assim, podemos manipular palavras e frases de forma facilitada.

```
string palavra1 ("be a ba"); // "be a ba"
string palavra2 (5, 'a'); // "aaaaa"

palavra2 = "abc"; // "abc"

palavra1[0] = 'd'; // "de a ba"
palavra1[5] = 'd'; // "de a da"

// Procurando sub-palavras
palavra1.substr(3); // "a da"
palavra1.substr(2, 4); // " a d"

// Procurando o tamanho da palavra
palavra1.length(); // 7
palavra2.size(); // 3

palavra1.empty(); // false, pois 'palavra1' tem uma, ou mais, letra

// Adicionando letras à 'palavra2'
palavra2.push_back('d'); // "abcd"
palavra2.push_back('e'); // "abdce"
palavra2.push_back('f'); // "abcdef"
```

```cpp
// Ordenando por ordem alfabética
sort(palavra1.begin(), palavra1.end()); // "  aadde"
sort(palavra2.rbegin(), palavra2.rend()); // "fedcba"

// Operações de busca em palavras
palavra1.find_first_of(" "); // 0, índice de onde achou o primeiro caracter
' '
palavra1.find_last_of(" "); // 1, índice de onde achou o último caracter ' '

palavra2.find_first_not_of("aeiou"); // 0, índice de onde achou o primeiro
caractere que não pertence à "aeiou" (ou seja, primeira consoante achada)
palavra2.find_last_not_of("a"); // 4, índice de onde achou o último
caractere que não seja 'a'

// Removendo letras de 'palavra2'
palavra2.erase(3, 2); // "feda"

palavra1.clear(); // ""
```

Python

List

Uma lista é uma estrutura de dados que permite acessar os valores do inicio, fim e por índice, além de ter tamanho variável conforme for inserido elementos.

> Dica
>
> Para utilizar uma `stack` (pilha), basta utilizar somente os métodos:
>
> - `append(x)`: inserir valor no final (ou topo) - para `x` sendo o elemento desejado
> - `pop()`: remover valor no final (ou topo)

> Dica
>
> Para utilizar uma `queue` (fila), basta utilizas somente os métodos:
>
> - `insert(0, x)`: inserir o valor no inicio - para `x` sendo o elemento desejado
> - `pop()`: remover o valor do final

> Dica
>
> Para utilizar um `set` (conjunto), basta fazer a conversão da lista:
>
> Exemplo:

- `conjunto = set([1, 1, 1, 2, 3, 1, -1, 1, 0]) # {-1, 0, 1, 2, 3}`

```python
lista1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
lista2 = [] # []

# Adiciona elementos ao final de 'lista2'
lista2.append(10) # [10]
lista2.append(10) # [10, 10]
lista2.append(10) # [10, 10, 10]

# Adiciona elementos no índice desejado de 'lista1'
lista1.insert(0, 11) # [11, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
lista1.insert(5, 90) # [11, 2, 3, 4, 90, 5, 6, 7, 8, 9, 10]

# Adiciona elementos na posição escolhida em 'lista2'
lista2.insert(1, 5) # [10, 5, 10, 10]

# Tamanho de 'lista2'
len(lista2) # 4

# Conta quantas vezes um elemento repetiu em 'lista2'
lista2.count(10) # 3

# Remove o primeiro elemento escolhido de 'lista2'
lista2.remove(10) # [5, 10, 10]
lista2.remove(10) # [5, 10]

# Remove o último elemento de 'lista1'
lista1.pop() # [11, 1, 2, 3, 4, 90, 5, 6, 7, 8, 9]

# Inverte a ordem de 'lista1'
lista1.reverse() # [9, 8, 7, 6, 5, 90, 4, 3, 2, 1, 11]

# Ordena 'lista1' de forma crescente
lista1.sort() # [1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 90]

# Ordena 'lista1' de forma decrescente
lista1.sort(reverse=True) # [90, 11, 9, 8, 7, 6, 5, 4, 3, 2, 1]

# Limpa 'lista1', deixando vazio
lista1.clear() # []

# Verifica se elemento pertence a 'lista2'
```

```
5 in lista2 # True
6 in lista2 # False
```

Dict

O dicionário é uma estrutura de dados que permite definir tanto o tipo da chave quanto o tipo do valor armazenado, semelhante ao `map` do `c++`

```
dicionario = { "Joao": 10, "Jorge": 25 } # { "Joao": 10, "Jorge": 25 }

# Adicionando elementos em 'dicionario'
dicionario["Jose"] = 20 # { "Joao": 10, "Jorge": 25, "Jose": 20 }

# Tamanho de 'dicionario2'
len(dicionario) # 3

# Verificando se contém chave em 'dicionario2'
"Jose" in dicionario # True
"Jaco" in dicionario # False

# Removendo elemento de 'dicionario2'
del dicionario["Jorge"] # { "Joao": 10, "Jose": 20 }

# Percorrendo todos os elemtentos de 'dicionario2'
for chave, valor in dicionario.items():
    print(f"'{chave}': '{valor}'")
```

String

A string, ou cadeia de caracteres, é uma estrutura de dados que permite agrupar letras. Assim, podemos manipular palavras e frases de forma facilitada.

```
palavra = "   hello world     " # "   hello world     "

# Deixa as primeiras letras de cada palavra em maiúsculo
palavra.title() # "   Hello World     "

# Deixa 'palavra' inteira em maiúsculo
palavra.upper() # "   HELLO WORLD     "

# Deixa 'palavra' inteira em minúsculo
palavra.lower() # "   hello world     "

# Conta repetição de elemtento em 'palavra'
palavra.count("hello") # 1
```

```python
# Verifica o tamanho de 'palavra'
len(palavra) # 19

# Verifica se final de 'palavra' contém pelomenos 'lo' ou 'rld'
palavra.endswith(('lo', 'rld')) # True

# Procura por menor indice de elemento em 'palavra'
palavra.find(' wo') # 5

# Verifica se existe elemento em 'palavra'
"ell" in palavra # True

# Separa 'palavra' em lista, com o delimitador escolhido
palavra.split(" ") # ["", "", "", "hello", "world", "", "", "", "", ""]

# Une 'palavra' de uma lista para uma string, com o delimitador escolhido
"-".join(palavra.split(" ")) # ---hello-world-----

# Remove os espaços vazios do começo e fim de 'palavra'
palavra.strip() # hello world

# Troca letras dentro de 'palavra'
palavra.replace(" ", "0") # 000hello0world00000
```

Algoritmos

> Observação
>
> Quando utilizado `@cache` para python, fica implicito a declaração `from functools import cache`.
>
> Quando utilizado `cache[n]` para c++, fica implicito a declaração `int cache[MAX_N]` ou semelhante (cache pode ser `long long` em algum exercício).

Fatorial

> Dica
>
> Muito utilizado para decomposição de números.

C++

```cpp
ll fat(int n) {
    if(cache[n]) return cache[n];
```

```
    if (n == 1 || n == 0)
        cache[n] = 1;
    else
        cache[n] = n * fat(n-1);

    return cache[n];
}
```

Python

```
@cache
def fat(n):
    if (n == 1 or n == 0):
        return 1
    return n * fat(n-1)
```

Fibonacci

C++

```
ll fibo(int n) {
    if (cache[n]) return cache[n];

    if (n == 0 || n == 1)
        cache[n] = 1;
    else
        cache[n] = fibo(n-2) + fibo(n-1);

    return cache[n];
}
```

Python

```
@cache
def fibo(n):
    if (n == 0 or n == 1):
        return 1
    return fibo(n-2) + fibo(n-1)
```

Crivo de Erastótenes

> Observação
>
> Usado para pré-calcular (tabelar) números primos.

> Primeiro, marca-se todos os números do vetor como primos, depois exclui os múltiplos de cada primo, pois estes necessariamente são números compostos.

> Dica
>
> Para este algoritmo, é necessário inicializar o cache com todos os valores positivos.
>
> Este algoritmo tem a intenção de ser rodado somente 1 vez, populando todo o cache com a resolução se cada indice `i` é, ou não, um número primo.
>
> Por exemplo:
>
> - `memset(cache, 1, sizeof cache);`, para c++;
> - `cache = [1 for i in range(0, 200)]`, para python;

C++

```cpp
void crivo(int n) {
    cache[0] = cache[1] = 0;
    for(int i=2; i<=n; i++)
        if(cache[i])
            for(int j=(i+i); j<=n; j+=i)
                cache[j] = 0;
}

/* Exemplo de utilização
*
* memset(cache, 1 sizeof cache);
* crivo(300);
* REP(i, 0, 300) {
*       if (cache[i])
*               cout << i << " eh primo" << ENDL;
* }
*/
```

Python

```python
def crivo(n):
    cache[0], cache[1] = 0, 0
    for i in range(2, n+1):
        if (cache[i]):
            for j in range(i+i, n+1, i):
                cache[j] = 0

# Exemplo de utilização
```

```
# cache = [1 for i in range(0, 130)]
# crivo(120)
# for i in range(0, 121):
#       if cache[i]:
#               print(f"{i} eh primo")
```

Busca Binária

Faz uma busca, de algum valor e/ou critério, em tempo melhor que linear, pois sempre limita a sua próxima pesquisa em um fator de 2.

> Observação
>
> Para que a busca binária funcione, é **necessário estar com o vetor ordenado!**

> Dica
>
> Na verificação `if-else`, podemos substituir a condição do exemplo por algum retorno de função ou condição de interesse do exercício.
>
> Por exemplo: a condição de interesse ser `((1 << meio) >= n)` ou `(verifica(meio, n))` inerente ao exercício.

C++

```cpp
int busca_binaria(vi& vetor, int n) {
    int inicio = 0, fim = vetor.size();
    int meio = 0, resposta = -1;

    while(inicio <= fim) {
        meio = (inicio+fim) / 2;

        if (vetor[meio] > n) fim = meio-1;
        else if(vetor[meio] < n) inicio = meio+1;
        else {
            // Achou índice onde vetor[meio] == n
            resposta = meio;
            break;
        }
    }

    return resposta;
}
```

Python

```python
def busca_binaria(vetor, n):
    inicio = 0
    fim = len(vetor)
    resposta = -1

    while (inicio <= fim):
        meio = (inicio + fim) // 2

        if (vetor[meio] > n):
            fim = meio - 1
        elif (vetor[meio] < n):
            inicio = meio + 1
        else:
            #   Achou índice onde vetor[meio] == n
            resposta = meio
            break

    return resposta
```

Máximo Divisor Comum

C++

```cpp
int mdc(int a, int b) {
    return !b ? a : mdc(b, (a % b));
}
```

Python

```python
def mdc(a, b):
    if (b == 0):
        return a
    return mdc(b, (a % b))
```

Mínimo Múltiplo Comum

> Observação
>
> Para usar essa função, é necessário implementar a função `mdc` supracitada.

C++

```cpp
int mmc(int a, int b) {
    return a * (b / mdc(a, b));
}
```

Python

```python
def mmc(a, b):
    return a * (b // mdc(a, b))
```

Combinação

> Lembrando
>
> A fórmula da combinação é: $C(n, k) = \frac{n!}{k! \cdot (n - k)!}$

C++

```cpp
double combinacao(int n, int k) {
    if (k == n || !k) return 1;

    return combinacao(n-1, k-1) + combinacao(n-1, k);
}
```

Python

```python
def combinacao(n, k):
    if (k == n or k == 0):
        return 1
    return combinacao(n-1, k-1) + combinacao(n-1, k)
```

Catalão

> Dica
>
> O número de catalão pode ser utilizado, por exemplo, para:
>
> - Contar o número de árvores binárias distintas com `n` vértices;
> - Contar o número de expressões contendo `n` pares de parenteses corretamente correspondidos;
> - Contar o número de maneiras diferentes que os `n + 1` fatores podem ser colocados em parênteses;
> - Contar o número de maneiras que um poligono convexo de `n + 2` lados pode ser dividido em triângulos;
> - Contar o número de caminhos monotônicos entre os vértices de uma matriz $n \times n$, que não passa na diagonal;

> Lembrando
>
> A fórmula do número de catalão é: $Cat(n) = \frac{^{(2 \cdot n)}C_n}{(n + 1)} = \frac{2 \cdot n!}{n! \cdot n! \cdot (n + 1)}$

C++

```cpp
ll catalao(int n) {
    if (cache[n]) return cache[n];

    if (!n) { // n == 0
        cache[n] = 1;
    } else {
        cache[n] = ((2*n)*(2*n -1)) / ((n*(n+1))) * catalao(n - 1);
    }

    return cache[n];
}
```

Python

```python
@cache
def catalao(n):
    if (n == 0):
        return 1
    return ((2*n)*(2*n-1)) // (n*(n+1)) * catalao(n-1)
```