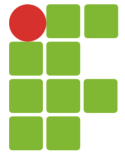
Pilhas

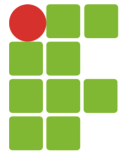
Introdução à Programação Competitiva

Prof. Daniel Saad Nogueira Nunes

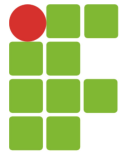
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário



Sumário



Introdução

- A Pilha é um **tipo abstrato de dados** que possui a propriedade LIFO (last-in-first-out).
- Isto é, os últimos elementos que são inseridos devem ser os primeiros a serem retirados.
- **Idealmente** as operações de inserção, remoção e acesso ao topo da pilha, devem ser realizadas em tempo $\Theta(1)$.

Introdução

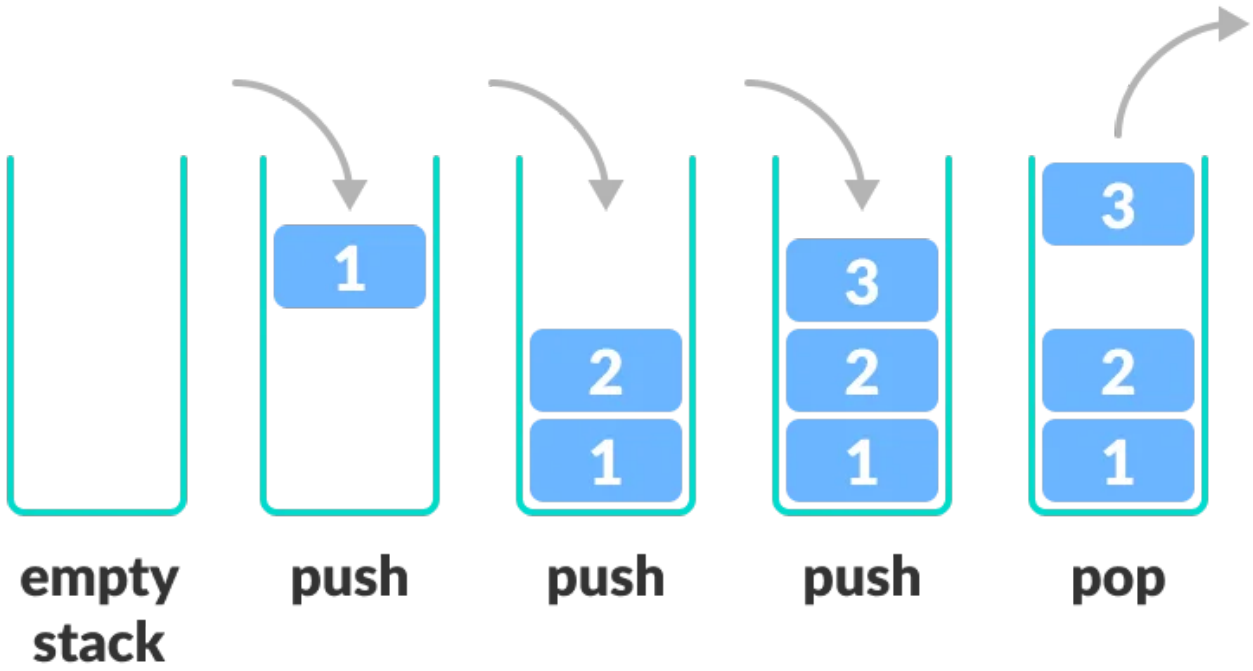
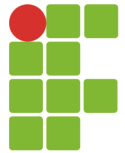
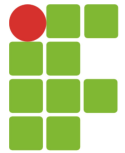


Figura: Fonte: `https://www.programiz.com/dsa/stack`



Implementação do TAD

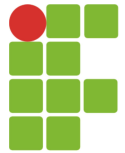
- Pilhas podem ser implementadas de diversas formas.
- Duas das formas são através de:
  - Listas autorreferenciadas.
  - Vetores dinâmicos



Implementação do TAD

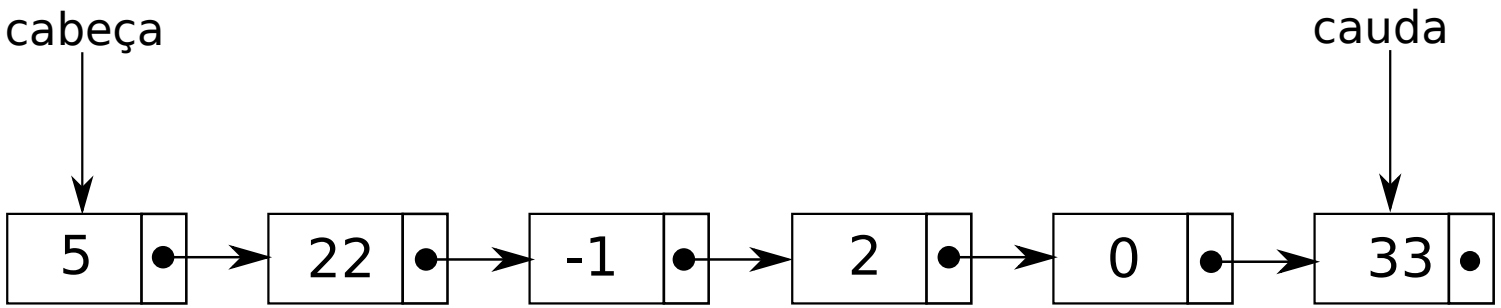
Listas autorreferenciadas

- Para suportar a operação de inserção, basta inserir na cabeça da lista.
- Para suportar a operação de remoção, basta remover na cauda da lista.
- O acesso ao topo equivale ao acesso à cabeça da lista.



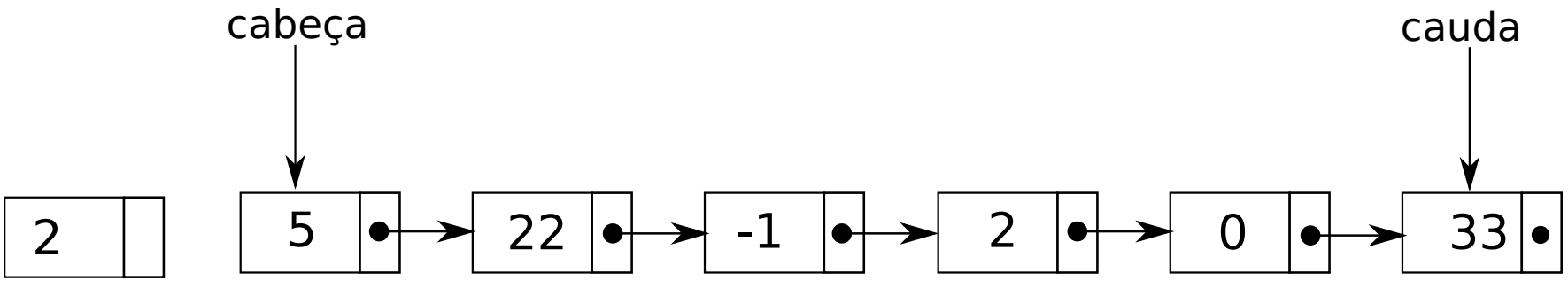
Implementação do TAD: listas autorreferenciadas

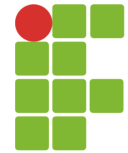
Inserção



Implementação do TAD: listas autorreferenciadas

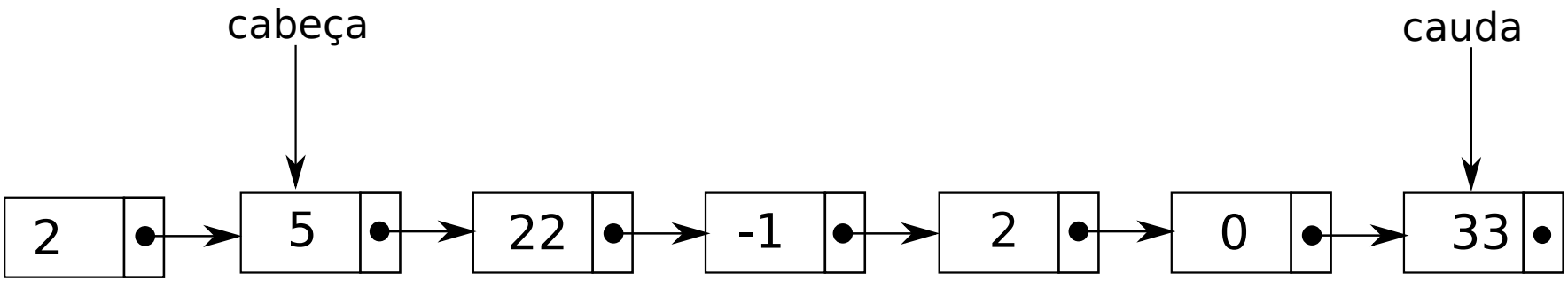
Inserção

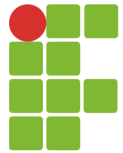




Implementação do TAD: listas autorreferenciadas

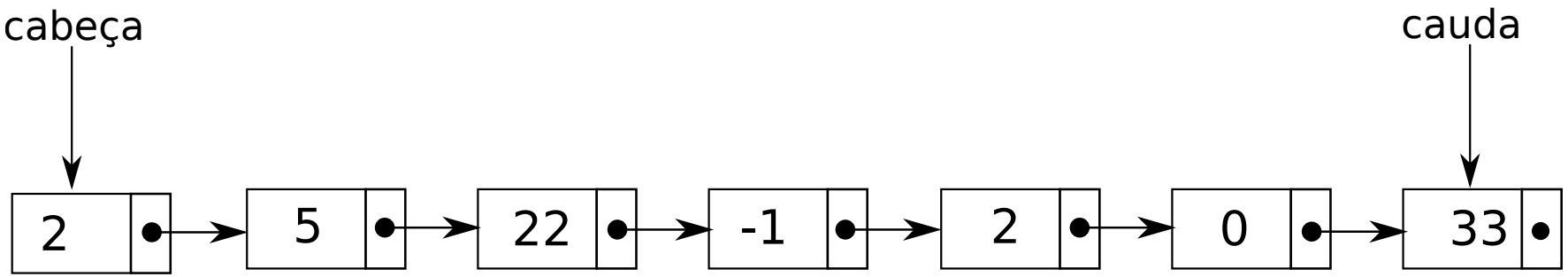
Inserção

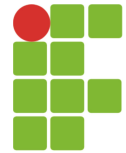




Implementação do TAD: listas autorreferenciadas

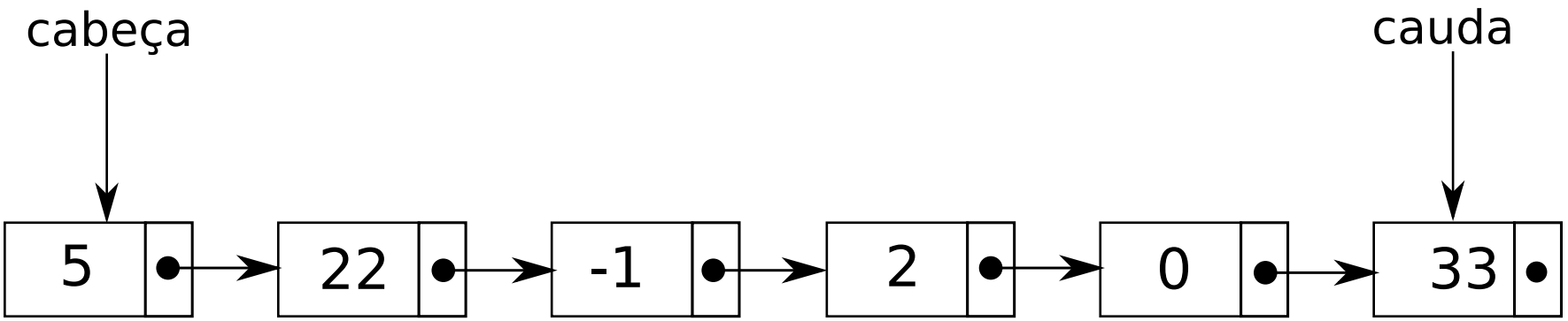
Inserção

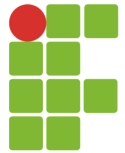




Implementação do TAD: listas autorreferenciadas

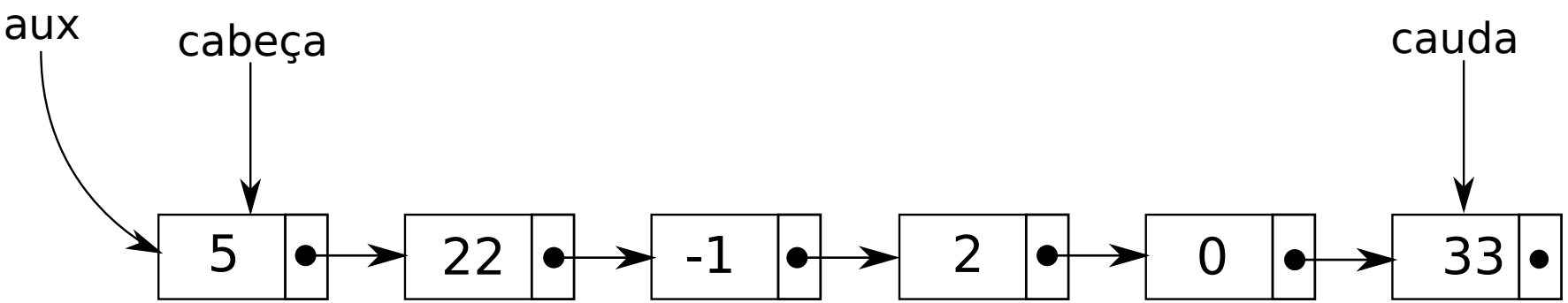
Remoção

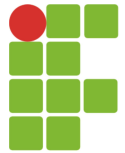




Implementação do TAD: listas autorreferenciadas

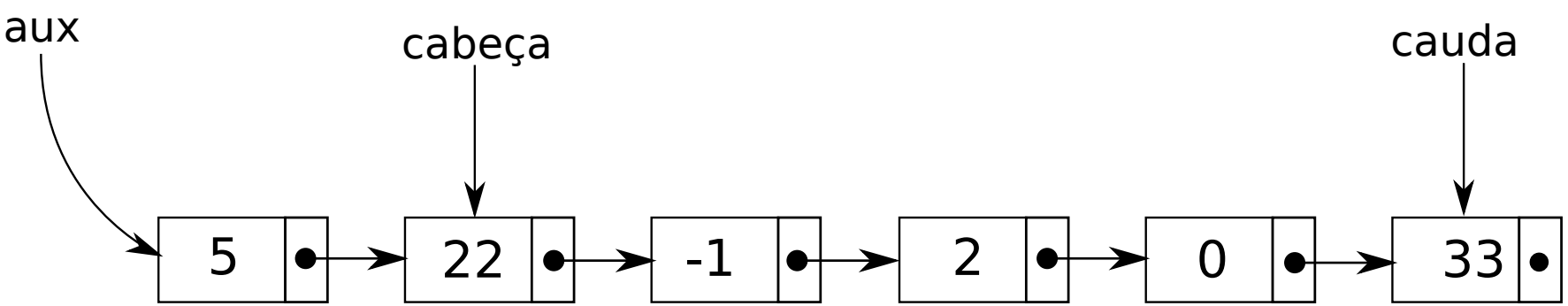
Remoção

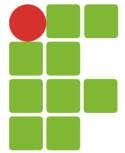




Implementação do TAD: listas autorreferenciadas

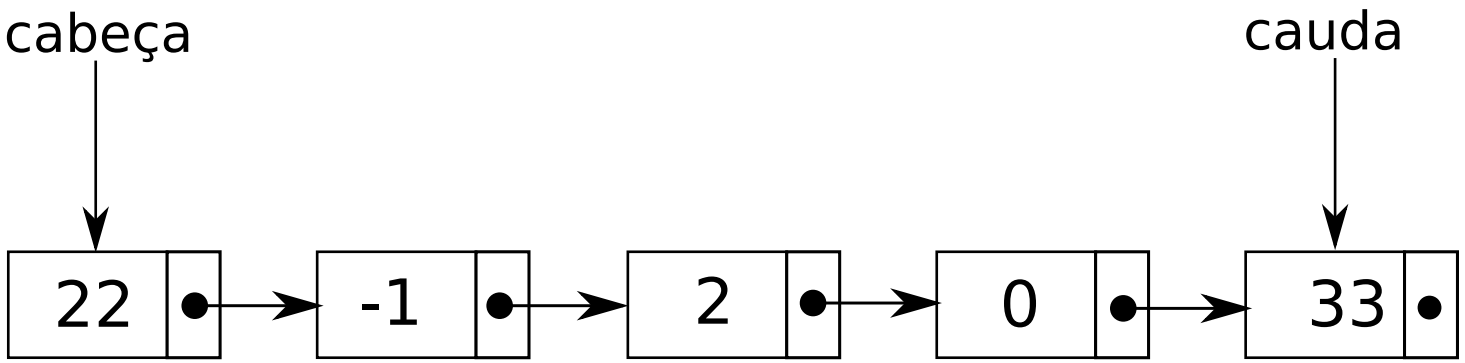
Remoção

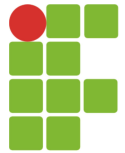




Implementação do TAD: listas autorreferenciadas

Remoção

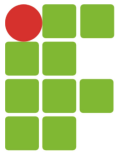




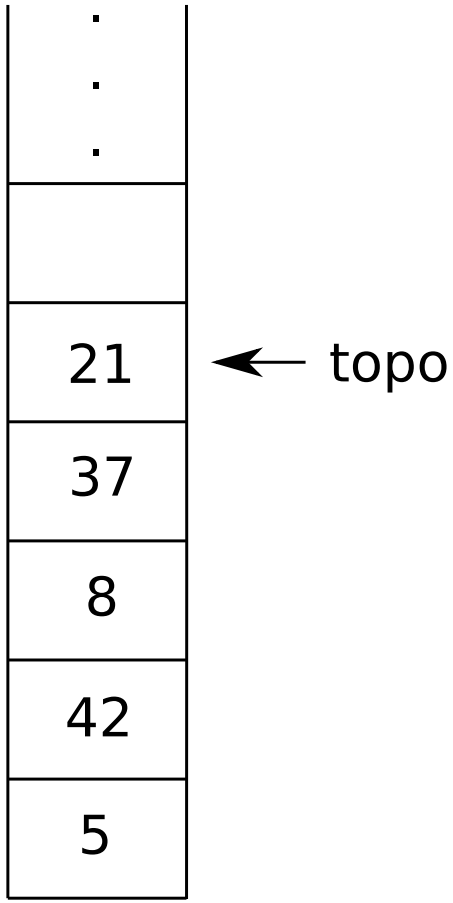
Implementação do TAD

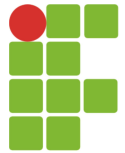
Vetores dinâmicos

- É possível manter uma variável indicando o topo do vetor.
- Sempre que um elemento é inserido, incrementa-se esta variável e insere-se na posição indicada por ela.
- Quando um elemento é removido, basta decrementar esta variável.
- Caso a pilha fique cheia, ou vazia demais, basta redimensionar o vetor.

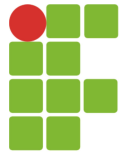


Implementação do TAD: vetores dinâmicos



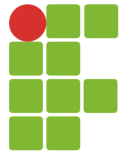


Sumário

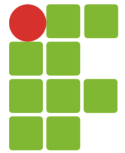


Pilhas: C++

- A STL do C++ fornece o tipo lista totalmente parametrizável.
- Por padrão é implementada através de um `std::deque`, mas outros *containers* como `std::vector` e `std::list` podem ser utilizados.
- Inserção, remoção e acesso ao topo em tempo $\Theta(1)$!
- Cabeçalho: `<stack>`.

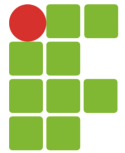


Sumário

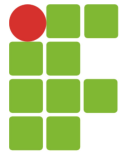


Declaração

- Para declarar uma pilha, utilizamos: `std::stack<T> nome_variavel;`, em que `<T>` corresponde ao tipo desejado.
- Exemplos:
  - `std::stack<int> pilha_int;`
  - `std::stack<pessoa> pilha_pessoa;`
  - `std::stack<std::pair<int,string>> pilha_par;`
  - `std::stack<vector<int>> pilha_vector;`

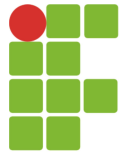


Sumário

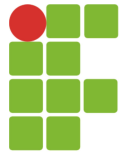


Inserção

- Para inserir no topo da pilha, utilizamos o método `push`.
- O `emplace` também pode ser utilizado no caso de uma inserção *in-place* (`C++11`).
- Tempo $\Theta(1)$.

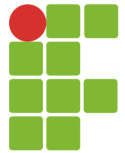


Sumário

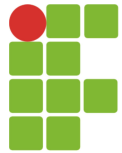


Remoção

- Para remover o item do topo da pilha, utilizamos o `pop`.
- Tempo $\Theta(1)$.

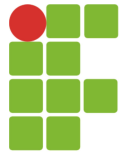


Sumário

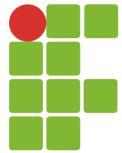


Acesso

- O acesso ao topo da pilha é realizado através do método `top`.
- Tempo $\Theta(1)$.

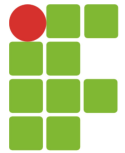


Sumário

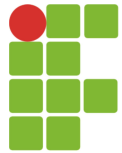


Limpeza

- O método `clear` pode ser utilizado para deletar todos os elementos de uma pilha.

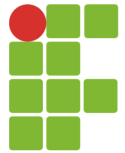


Sumário

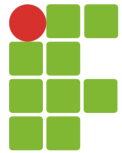


Métodos auxiliares

- `bool empty() const;` : retorna verdadeiro se a pilha está vazia.
- `size_t size() const;` : retorna o tamanho da pilha.



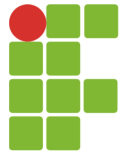
Sumário



Exemplo

Valores menores mais próximos

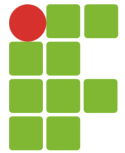
Seja uma sequência de n inteiros. Determine, para cada elemento desta sequência, a posição do elemento mais próximo à esquerda que seja menor do que o primeiro.



Exemplo

Entrada

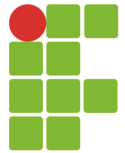
A primeira linha da entrada possui um inteiro n, o tamanho da sequência. A segunda linha possui os inteiros $x_1, x_2, \ldots, x_n$.



Exemplo

Saída

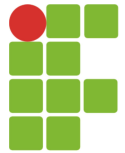
Para cada inteiro da sequência, imprima a posição do elemento mais próximo a esquerda que seja menor que o primeiro. Se não existir tal posição, imprima 0.



Exemplo

Restrições

- $1 \leq n \leq 2 \cdot 10^5$
- $1 \leq x_i \leq 10^9$



Exemplo

Exemplo de entrada/saída

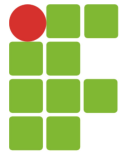
- Entrada:

```
8

2 5 1 4 8 3 2 5
```

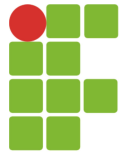
- Saída:

```
0 1 0 3 4 3 3 7
```



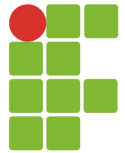
Exemplo

- Para resolver este problema, podemos utilizar uma pilha que guarda o índice dos elementos.
- Para cada elemento x_i da sequência, enquanto o índice no topo da pilha corresponder a um elemento maior ou igual a x_i, retiramos o elemento do topo da pilha.
- Se a pilha ficou vazia, imprimimos 0, caso contrário, imprimimos o índice que está no topo adicionado de 1.
- Inserimos a posição i na pilha.
- Complexidade: $\Theta(n)$.



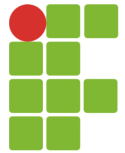
Solução

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int n;
6 vector<int> v;
```



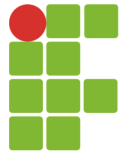
Solução

```
8  void read_input() {
9      cin >> n;
10     v.resize(n);
11     for (auto &x : v)
12         cin >> x;
13 }
```



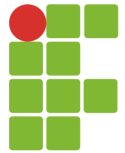
Solução

```
15 void solve() {
16     stack<int> stck;
17     for (size_t i = 0; i < v.size(); i++) {
18         while (!stck.empty() && v[stck.top()] >= v[i]) {
19             stck.pop();
20         }
21         auto pos = stck.empty() ? 0 : stck.top() + 1;
22         stck.push(i);
23         cout << pos;
24         cout << (i < v.size() - 1 ? ' ' : '\n');
25     }
26 }
```

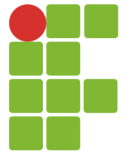


Solução

```
28 int main() {
29     ios::sync_with_stdio(false);
30     read_input();
31     solve();
32     return 0;
33 }
```



Sumário



Referências

cppreference, *cppreference.com*, `https://en.cppreference.com/`, Acessado em 11/2022.