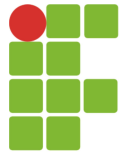
# Deques

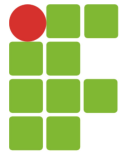
## Introdução à Programação Competitiva


Prof. Daniel Saad Nogueira Nunes

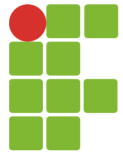
IFB – Instituto Federal de Brasília,
Campus Taguatinga



# Sumário

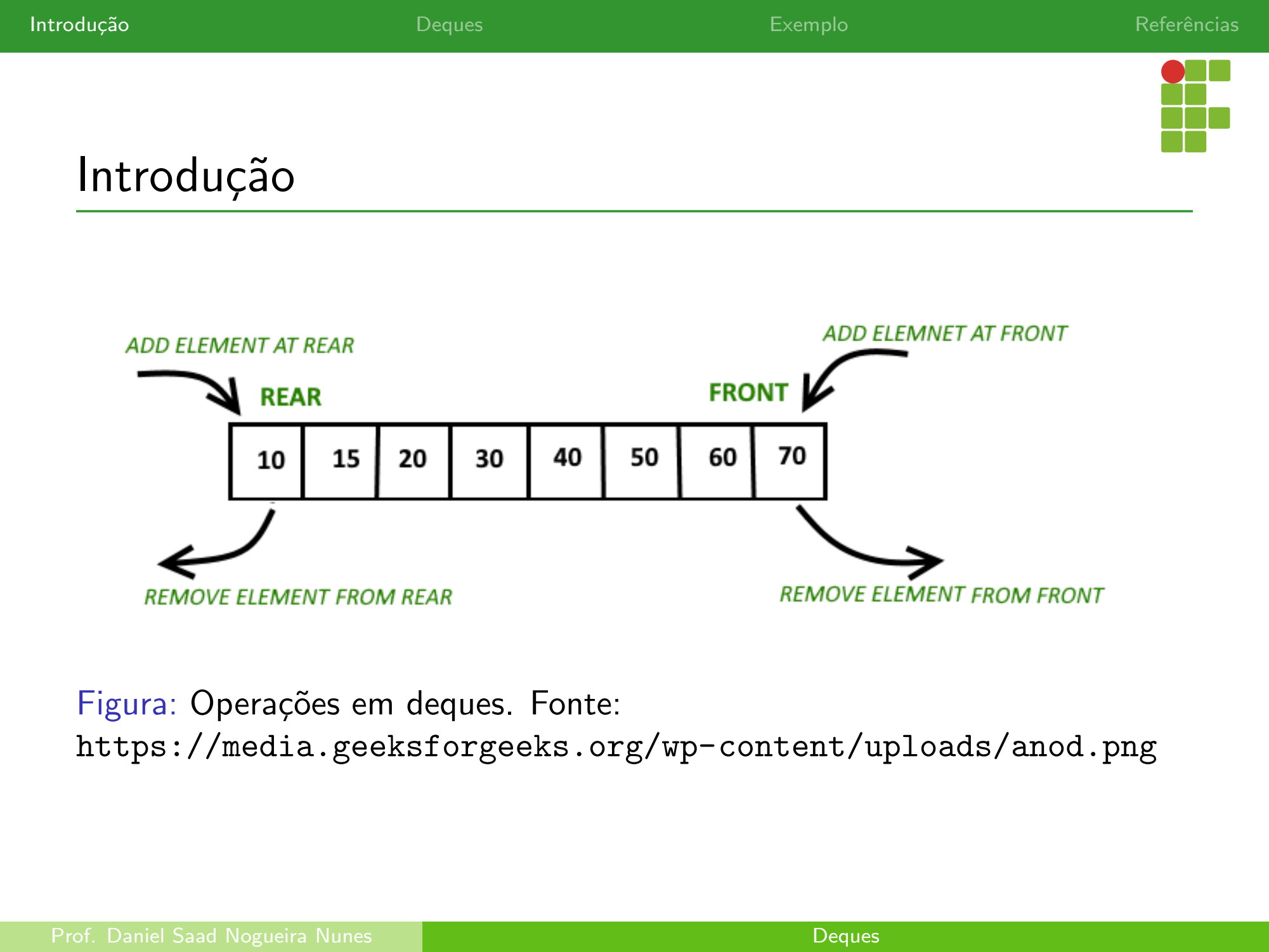


# Sumário



# Introdução

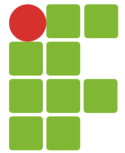
- Um Deque (Double-ended-queue) é um **tipo abstrato de dados** que possibilita operações eficientes em suas duas extremidades.
- **Idealmente** as operações de inserção e remoção na frente do deque devem levar tempo $\Theta(1)$.
- Podem simular uma fila ou pilha facilmente.



# Introdução

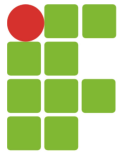



Figura: Operações em deques. Fonte: `https://media.geeksforgeeks.org/wp-content/uploads/anod.png`

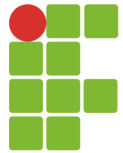


# Implementação do TAD

- Deques podem ser implementados através de uma lista duplamente encadeada, já que ela suporta operações eficientes nas suas extremidades.
- O C++ implementa os deques de uma outra maneira, possibilitando acesso em tempo $\Theta(1)$ a qualquer elemento:
  *As opposed to std::vector, the elements of a deque are not stored contiguously: typical implementations use a sequence of individually allocated fixed-size arrays, with additional bookkeeping, which means indexed access to deque must perform two pointer dereferences, compared to vector's indexed access which performs only one.*

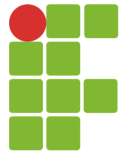


# Sumário

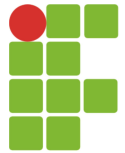


# Deques: C++

- A STL do C++ fornece o tipo deque totalmente parametrizável.
- Inserção, remoção e acesso às extremidades leva tempo $\Theta(1)$!
- Acesso a qualquer elemento também leva tempo $\Theta(1)$!
- Também é possível inserir ou remover elementos do meio do deque, como em uma lista.
- Cabeçalho: `<deque>`.

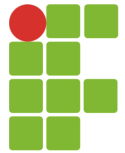


# Sumário

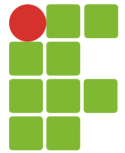


# Declaração

- Para declarar uma pilha, utilizamos: `std::deque<T> nome_variavel;`, em que `<T>` corresponde ao tipo desejado.
- Exemplos:
  - `std::deque<int> deque_int;`
  - `std::deque<pessoa> deque_pessoa;`
  - `std::deque<std::pair<int,string>> deque_par;`
  - `std::deque<vector<int>> deque_vetor;`

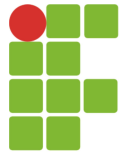


# Sumário



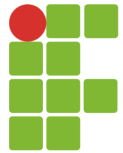
# Inserção

- Para inserir na frente do deque, utilizamos o método `push_front`.
- O `emplace_front` também pode ser utilizado no caso de uma inserção *in-place* (`C++11`).
- `push_back` e `emplace_back` funcionam de maneira análoga, mas na parte de trás do deque.
- Tempo $\Theta(1)$.



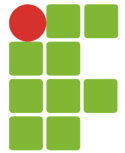
# Inserção

- Com posse do iterador para a posição em que se deseja inserir é possível adicionar elementos no meio do deque, como fazíamos nas listas.
- Para isso, usamos o método `insert`.
- Tempo: linear no número de elementos inseridos se tivermos o iterador.



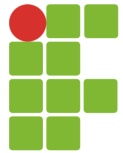
# Inserção

```
1  #include <deque>
2  #include <iostream>
3  #include <iterator>
4
5  void print(int id, const std::deque<int> &container) {
6      std::cout << id << ". ";
7      for (const int x : container)
8          std::cout << x << ' ';
9      std::cout << '\n';
10 }
11
```



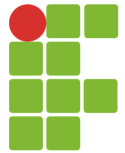
# Inserção

```
12  int main() {
13      std::deque<int> c1(3, 100);
14      print(1, c1);
15
16      auto it = c1.begin();
17      it = c1.insert(it, 200);
18      print(2, c1);
19
20      c1.insert(it, 2, 300);
21      print(3, c1);
22
23      // reset `it` to the begin:
24      it = c1.begin();
25
```

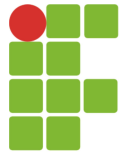


# Inserção

```
26      std::deque<int> c2(2, 400);
27      c1.insert(std::next(it, 2), c2.begin(), c2.end());
28      print(4, c1);
29
30      int arr[] = {501, 502, 503};
31      c1.insert(c1.begin(), arr, arr + std::size(arr));
32      print(5, c1);
33
34      c1.insert(c1.end(), {601, 602, 603});
35      print(6, c1);
36
37      return 0;
38  }
```

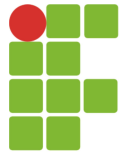


# Sumário



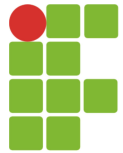
# Remoção

- Para remover da frente do deque, utilizamos o método `pop_front`.
- Para remover de trás do deque utilizamos o método `pop_back`.
- Tempo $\Theta(1)$.



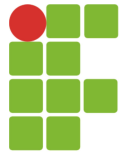
# Remoção

- Com posse do iterador para a posição em que se deseja remover é possível remover elementos do meio do deque, como fazíamos nas listas.
- Para isso, usamos o método `erase`.
- O erase retorna o iterador para o elemento que sucede o(s) removido(s) ou ( `end()` ).
- Tempo: linear no número de elementos removidos se tivermos o iterador.



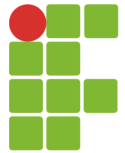
# Remoção

```
1  #include <deque>
2  #include <iostream>
3
4  void print_container(const std::deque<int> &c) {
5      for (int i : c)
6          std::cout << i << " ";
7      std::cout << '\n';
8  }
9
```

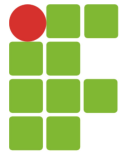


# Remoção

```
10  int main() {
11      std::deque<int> c{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
12      print_container(c);
13
14      c.erase(c.begin());
15      print_container(c);
16
17      c.erase(c.begin() + 2, c.begin() + 5);
18      print_container(c);
19
20      // Erase all even numbers
21      for (std::deque<int>::iterator it = c.begin(); it != c.end();) {
22          if (*it % 2 == 0)
23              it = c.erase(it);
24          else
25              ++it;
26      }
27      print_container(c);
28      return 0;
29  }
```

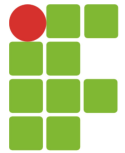


# Sumário

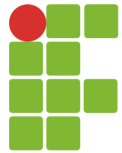


# Acesso

- O acesso ao elemento da frente do deque é realizado através do método `front` enquanto o acesso ao último é realizado através do método `back`.
- Podemos usar o operador `[]` para acessar qualquer elemento do deque. Exemplo: `D[i]` nos dá o $i$-ésimo elemento do deque $D$.
- Tempo $\Theta(1)$.

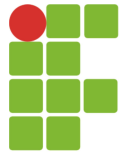


# Sumário

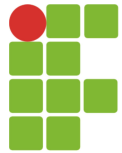


# Limpeza

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

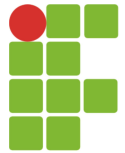


# Sumário

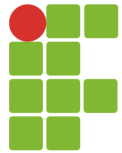


# Métodos auxiliares

- `bool empty() const;` : retorna verdadeiro se o deque está vazio.
- `size_t size() const;` : retorna o tamanho do deque.



# Sumário

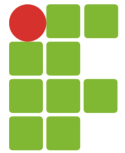


# Exemplo

## Teque (Kattis)

Implemente uma estrutura de dados *teque* (triple-ended-queue) $t$ que suporte as seguintes operações:

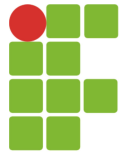
- `push_front(T,x)` : insere um inteiro $x$ no início de um teque $T$.
- `push_back(T,x)` : insere um inteiro $x$ no final do teque $T$.
- `push_middle(T,x)` : insere um inteiro $x$ no meio de um teque $T$. Isto é, se o tamanho do deque é $k$, o elemento deve ser inserido na posição $\lfloor (k+1)/2 \rfloor$ (índices baseados em $0$).
- `get(T,i)` : retorna o $i$-ésimo elemento do teque $T$ (índices baseados em $0$).



# Exemplo

## Entrada

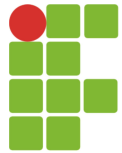
Cada linha da entrada possui um comando e um inteiro.



# Exemplo

## Saída

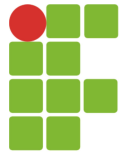
Para cada comando `get`: imprima o elemento correspondente do teque $T$.



# Exemplo

## Restrições

- $1 \leq n \leq 10^6$, em que $n$ é o número de comandos.



# Exemplo

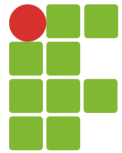
## Exemplo de entrada/saída

- Entrada:

```
9
push_back 9
push_front 3
push_middle 5
get 0
get 1
get 2
push_middle 1
get 1
get 2
```

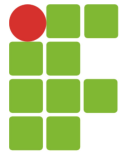
- Saída:

```
3
5
9
5
1
```



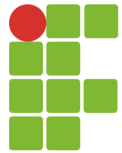
# Exemplo

- Para resolver este problema, podemos utilizar dois deques, o da esquerda e o da direita, para suportar a operação de `push_middle` mais facilmente.
- Se o comando for `push_front` inserimos no início do deque da esquerda.
- Se o comando for `push_back`, inserimos no final do deque da direita.
- Se o comando for `push_middle`, inserimos no começo do deque da esquerda.
- Se o comando for `get`, examinamos o índice e calculamos a posição em do elemento no deque correto com base no tamanho dos dois deques.
- **Importante**: deixar o tamanho do segundo deque menor ou igual ao do primeiro deque para a operação `push_middle`.



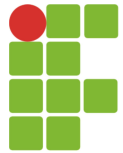
# Solução

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  int n;
6  vector<pair<string, int>> commands;
7
8  void read_input() {
9      cin >> n;
10     commands.resize(n);
11     for (auto &[s, x] : commands) {
12         cin >> s >> x;
13     }
14 }
15
```



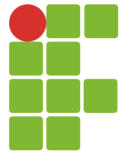
# Solução

```
16  void solve() {
17      deque<int> dl, dr;
18      for (const auto &[s, x] : commands) {
19          if (s == "push_front") {
20              dl.push_front(x);
21              if (dl.size() > dr.size() + 1) {
22                  dr.push_front(dl.back());
23                  dl.pop_back();
24              }
25          } else if (s == "push_back") {
26              dr.push_back(x);
27              if (dr.size() > dl.size()) {
28                  dl.push_back(dr.front());
29                  dr.pop_front();
30              }
31          } else if (s == "push_middle") {
32              dr.push_front(x);
33              if (dr.size() > dl.size()) {
34                  dl.push_back(dr.front());
35                  dr.pop_front();
36              }
37
38          } else if (s == "get") {
39              cout << (x < dl.size() ? dl[x] : dr[x - dl.size()]) << '\n';
40          }
41      }
42  }
43
```

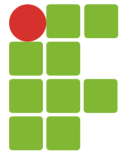


# Solução

```
44  int main() {
45      ios::sync_with_stdio(false);
46      read_input();
47      solve();
48      return 0;
49  }
```



# Sumário



# Referências

cppreference, *cppreference.com*, `https://en.cppreference.com/`, Acessado em 12/2022.