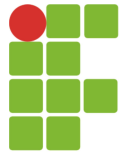
Deques

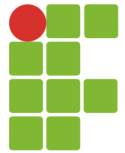
Estruturas de Dados e Algoritmos

Prof. Daniel Saad Nogueira Nunes

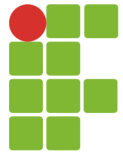
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

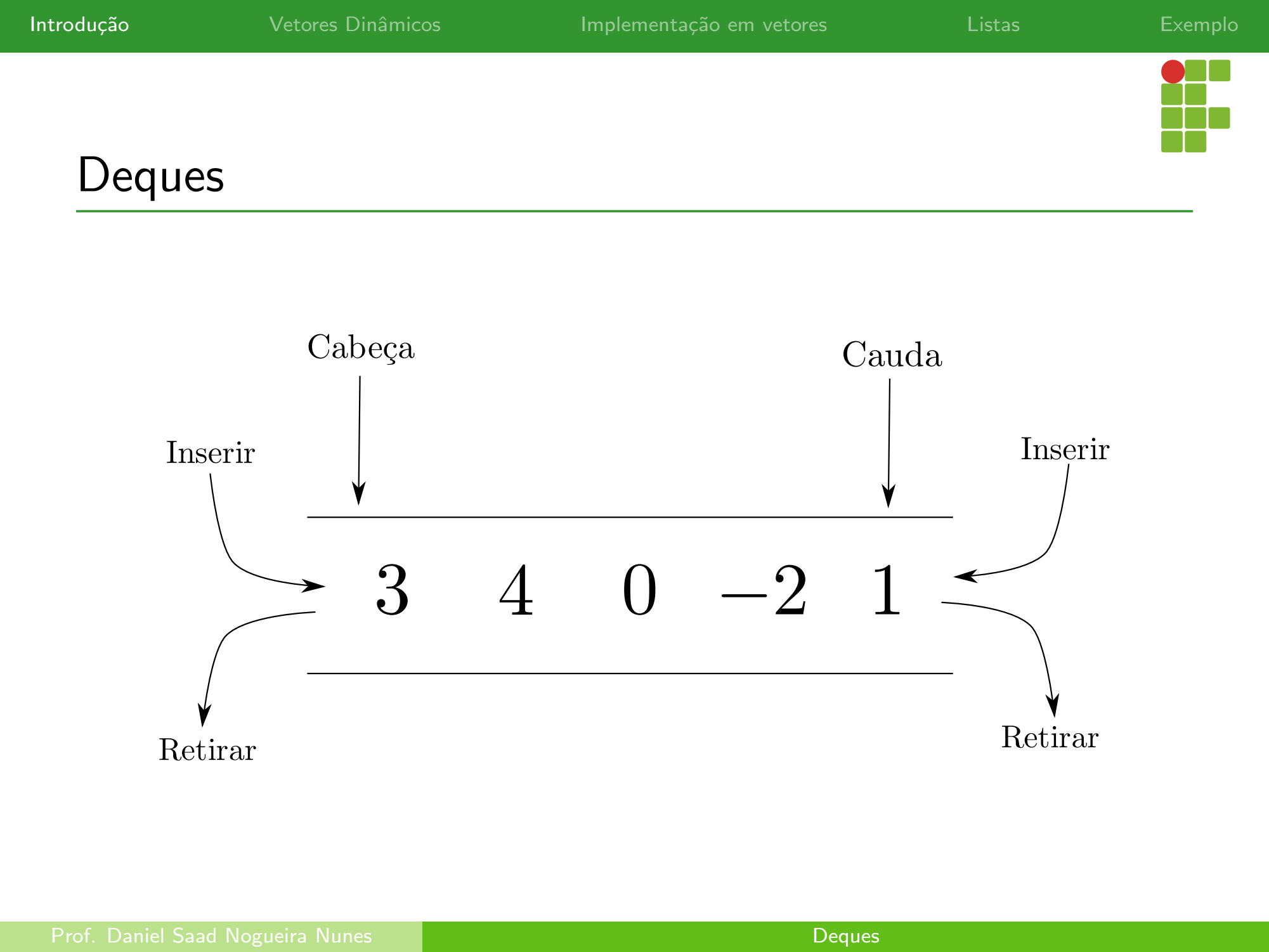


Sumário

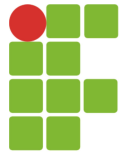


Deques

- Deque: Double-Ended-Queue.
- Deques são TADs que generalizam filas.
- Em deques, elementos podem ser adicionados tanto no início quando no fim da fila.
- Pode-se retirar elementos nas duas extremidades também.

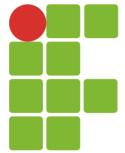


Deques



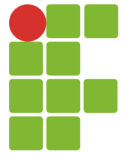
Operações em Filas

- Algumas das operações suportadas por um deque devem ser:
  - Inserção no início.
  - Remoção no início.
  - Inserção no fim.
  - Remoção no fim.
  - Acessar o elemento do início;
  - Acessar o elemento do fim;
  - Verificar se o deque está vazio;
  - Verificar o tamanho do deque;

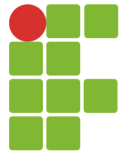


Representação de deques

- Assim como filas, deques podem ser representadas de várias maneiras, duas delas são por meio de:
  1. Vetores;
  2. Estruturas auto-referenciadas;

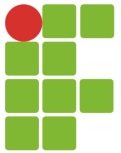


Sumário

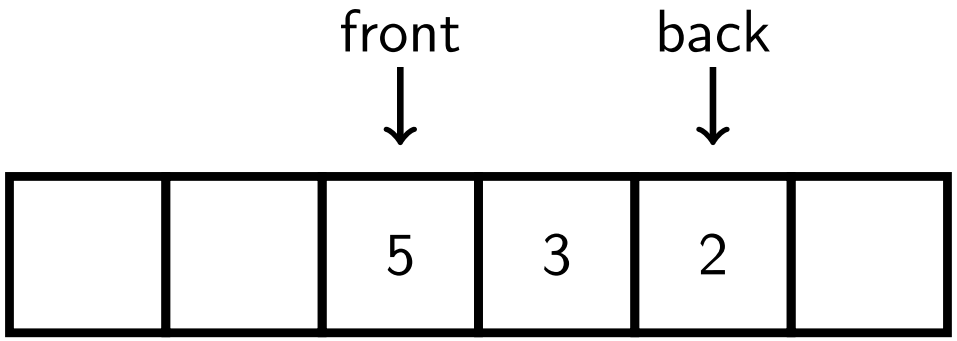


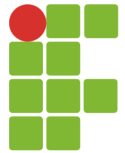
Representação de deques em vetores

- Para representar deques em vetores utilizamos a mesma estrutura utilizada nas filas, que é composta por um vetor e dois índices, que apontam,respectivamente, para a posição da frente (início) da fila e a posição de trás (fim) da fila.

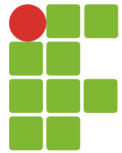


Representação de deques em Vetores



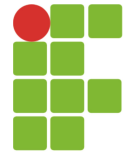


Sumário

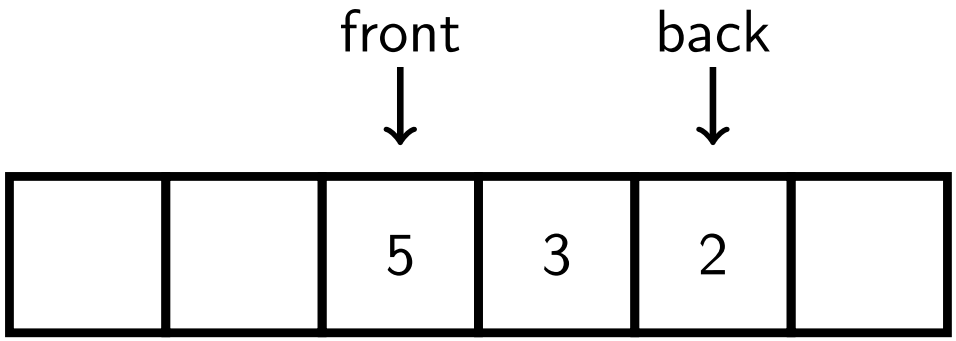


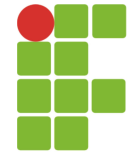
Inserção no final em vetores

- Para inserir um elemento ao final de um deque utilizamos o mesmo procedimento realizado na fila.
- incrementamos o índice do fim e colocamos o novo elemento nessa posição.

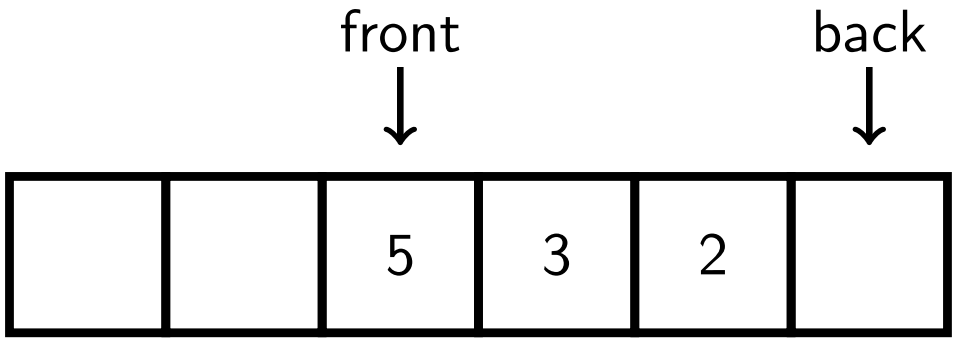


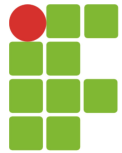
Inserçao no final em vetores



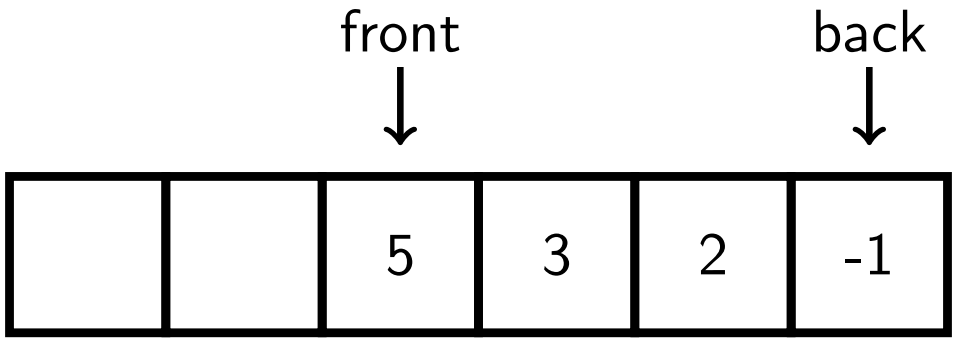


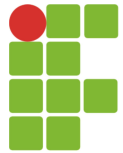
Inserçao no final em vetores





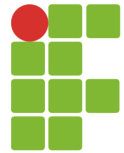
Inserçao no final em vetores



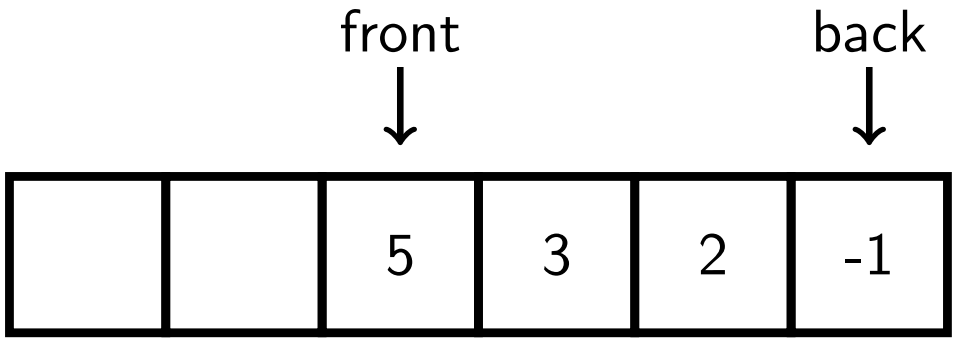


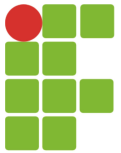
Inserção no final em vetores

- Quando o índice do fim estiver no final do vetor, podemos simplesmente "dar a volta" pra reutilizar o espaço!

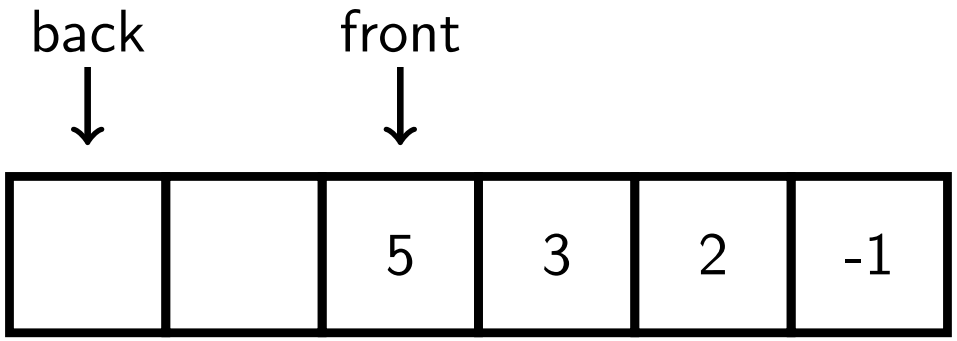


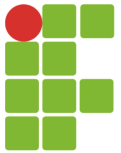
Enfileiramento em vetores



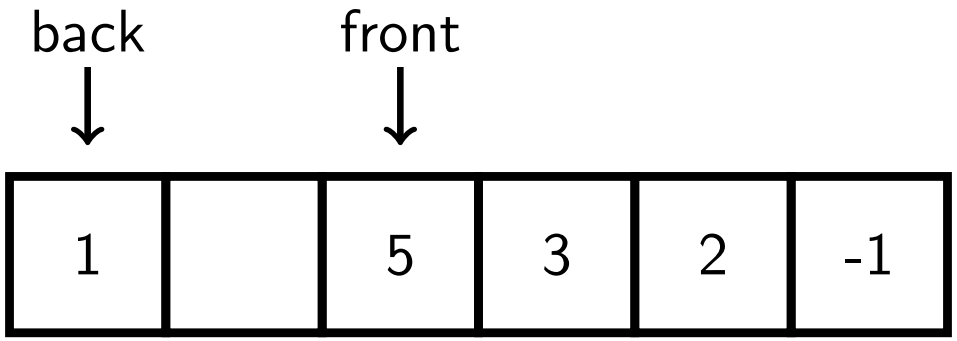


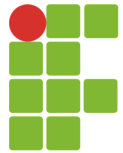
Enfileiramento em vetores





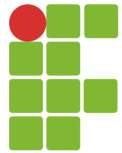
Enfileiramento em vetores





Inserção no final em vetores

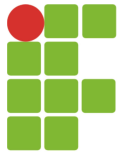
- Caso o vetor venha a ficar totalmente preenchido, duplicamos a sua capacidade.
- Contudo, existem dois casos que devemos observar:
  1. Índice do início menor que o do fim.
  2. Índice do início menor que o do fim.



Inserção no final em vetores

Índice do início menor que o do fim

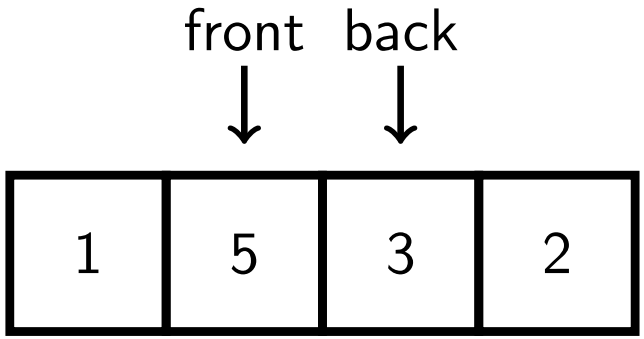
- Só precisamos duplicar a capacidade do vetor.

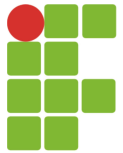


Inserção no final em vetores

Índice do início menor que o do fim

- Só precisamos duplicar a capacidade do vetor.

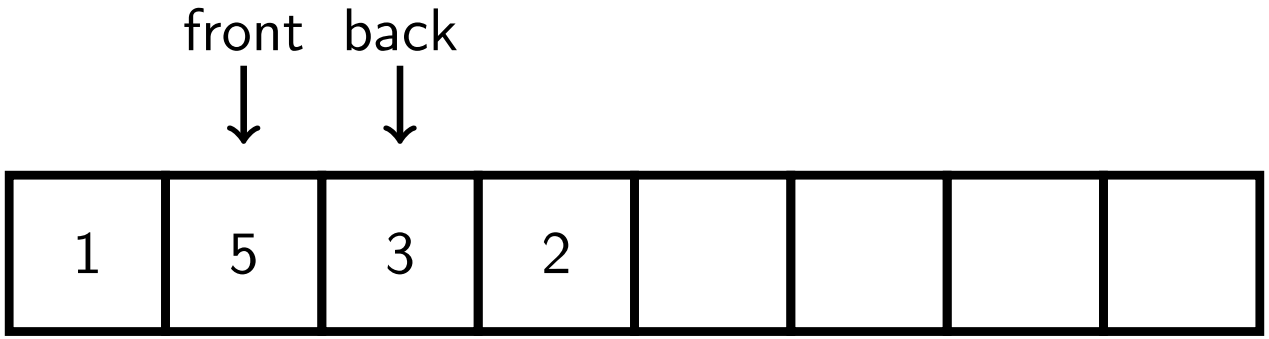


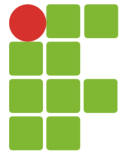


Inserção no final em vetores

Índice do início menor que o do fim

- Só precisamos duplicar a capacidade do vetor.

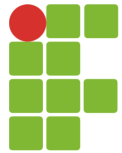




Inserção no final em vetores

Índice do início maior que o do fim

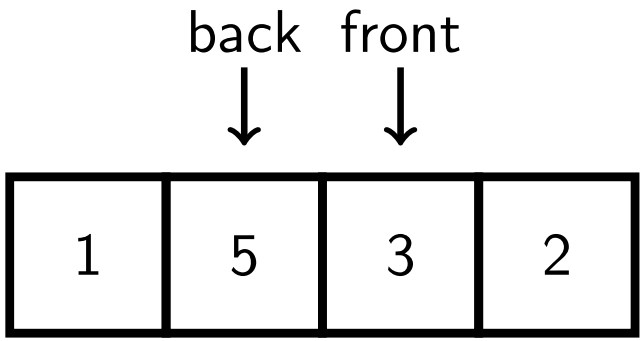
- Para deixar o deque em um estado consistente, devemos copiar todos os elementos que se encontram do início para frente para o final do vetor realocado.

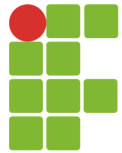


Inserção no final em vetores

Índice do início maior que o do fim

- Para deixar o deque em um estado consistente, devemos copiar todos os elementos que se encontram do início para frente para o final do vetor realocado.

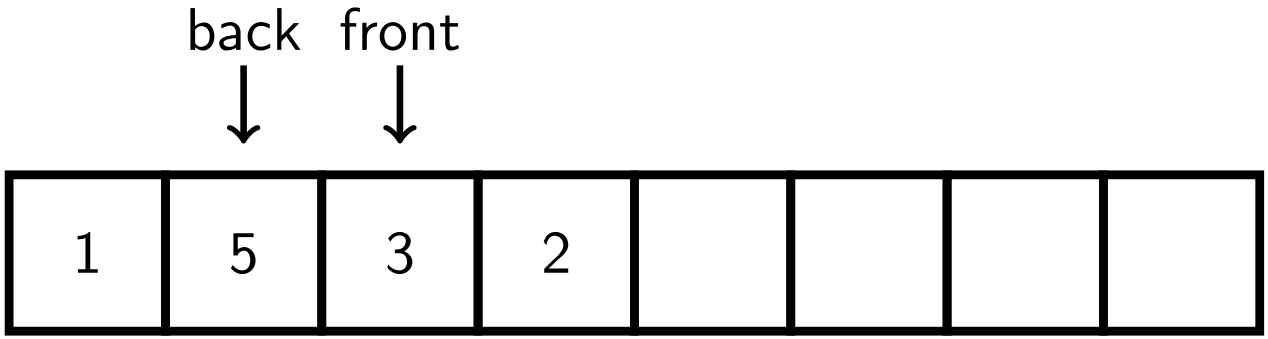


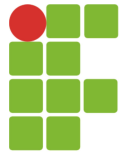


Inserção no final em vetores

Índice do início maior que o do fim

- Para deixar o deque em um estado consistente, devemos copiar todos os elementos que se encontram do início para frente para o final do vetor realocado.

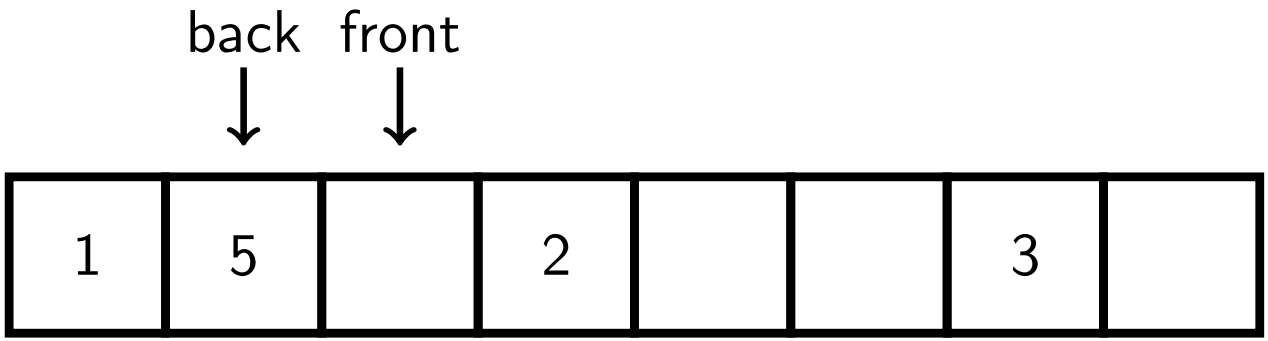


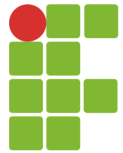


Inserção no final em vetores

Índice do início maior que o do fim

- Para deixar o deque em um estado consistente, devemos copiar todos os elementos que se encontram do início para frente para o final do vetor realocado.

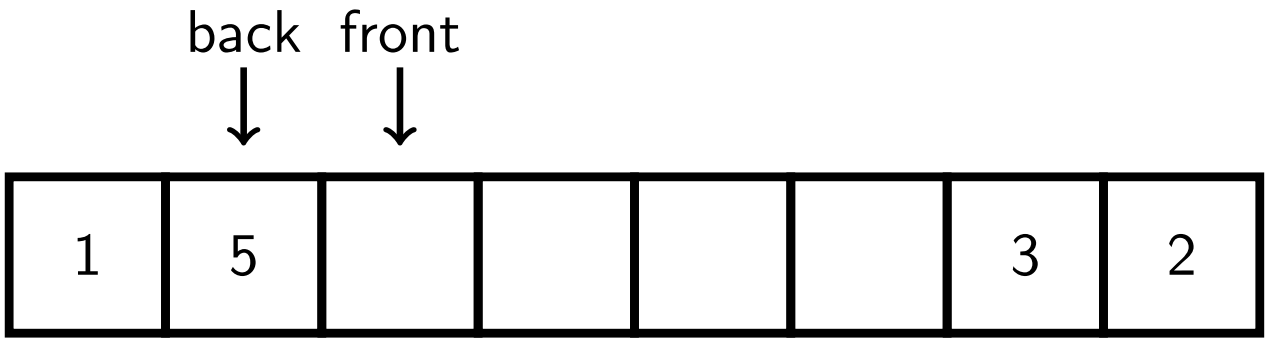


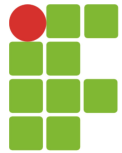


Inserção no final em vetores

Índice do início maior que o do fim

- Para deixar o deque em um estado consistente, devemos copiar todos os elementos que se encontram do início para frente para o final do vetor realocado.

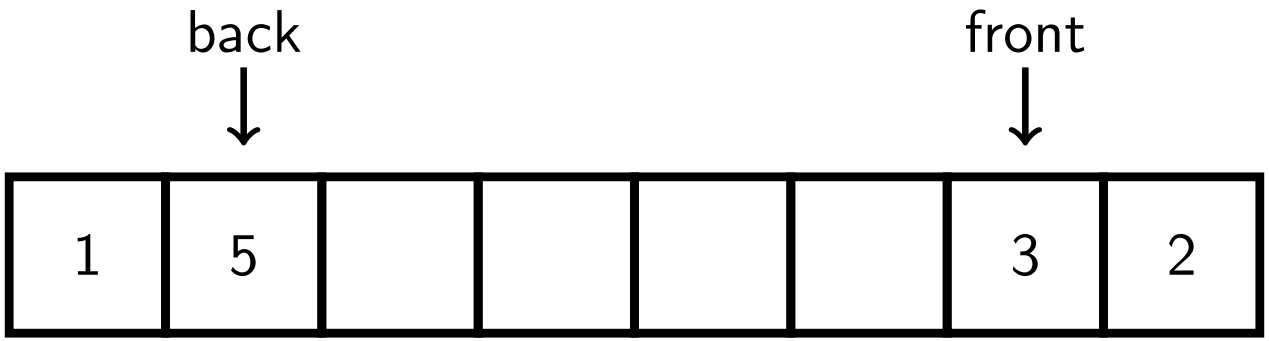


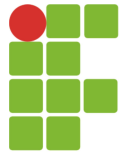


Inserção no final em vetores

Índice do início maior que o do fim

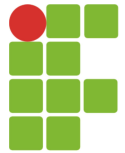
- Para deixar o deque em um estado consistente, devemos copiar todos os elementos que se encontram do início para frente para o final do vetor realocado.



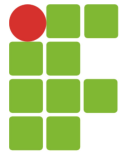


Inserção no final em vetores

- Copiar os elementos ao duplicar o vetor não é uma operação tão custosa ao longo do tempo, pois só fazemos isso cada vez que o vetor está cheio. Isso só ocorre após $\frac{n}{2}$ inserções, logo, o custo é amortizado.

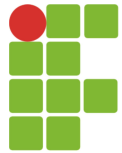


Sumário

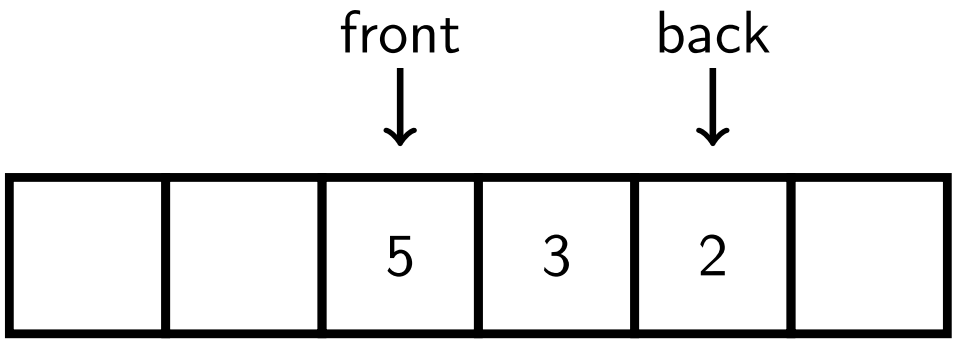


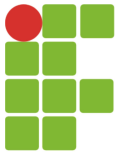
Inserção no inídio

- A inserção no início é extremamente parecida com a inserção ao final.
- Até nos casos em que o vetor fica cheio.
- A única diferença é que o índice do início é diminuido e o elemento é inserido.
- Caso o índice fosse zero antes da inserção, ele passa a ser igual a capacidade do vetor menos um.

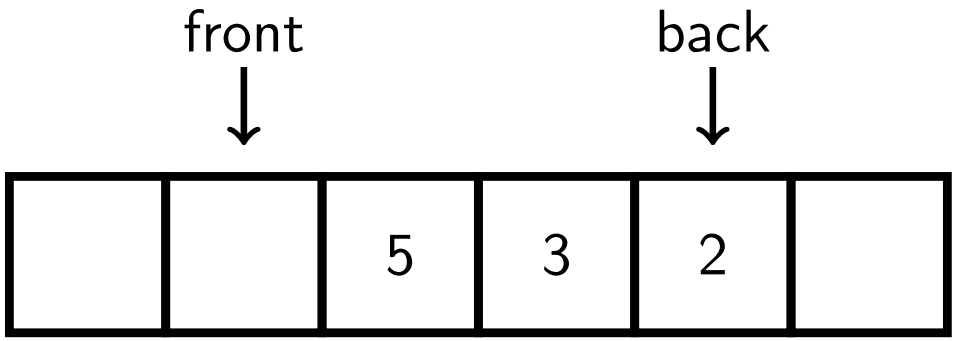


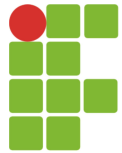
Inserção no início



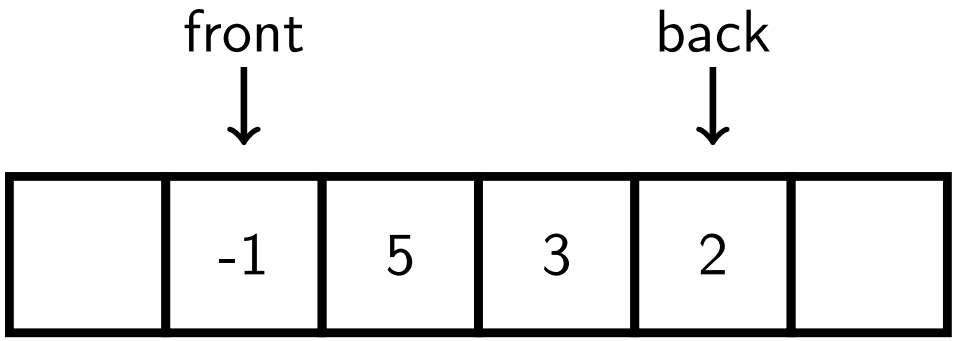


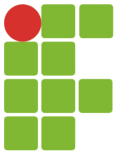
Inserção no início



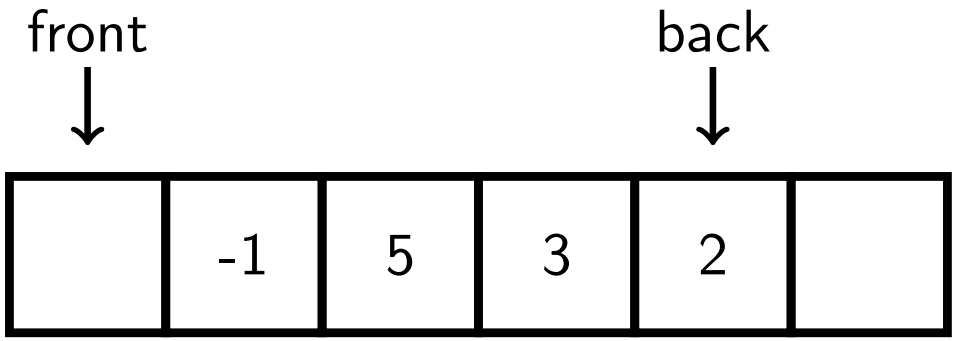


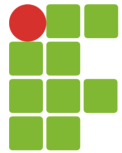
Inserção no início



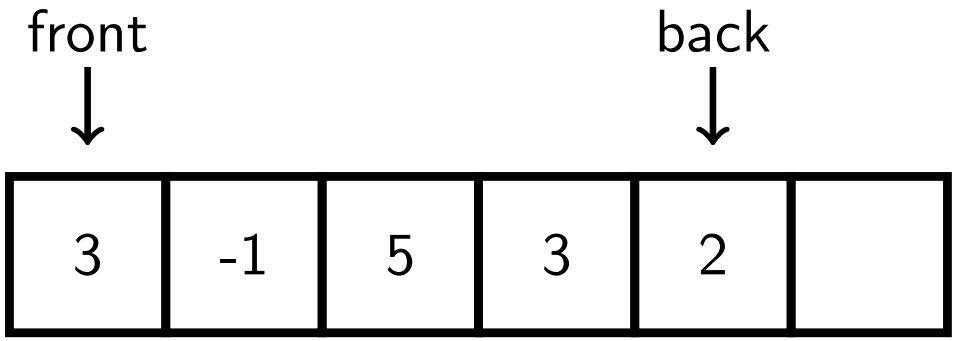


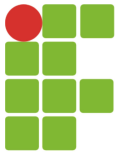
Inserção no início



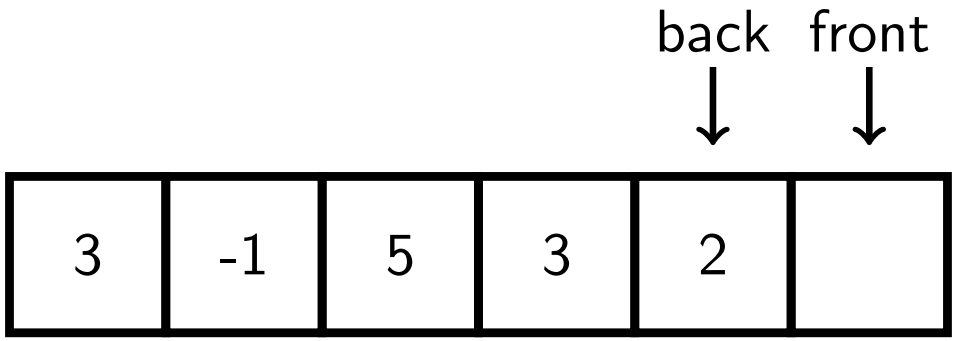


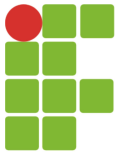
Inserção no início



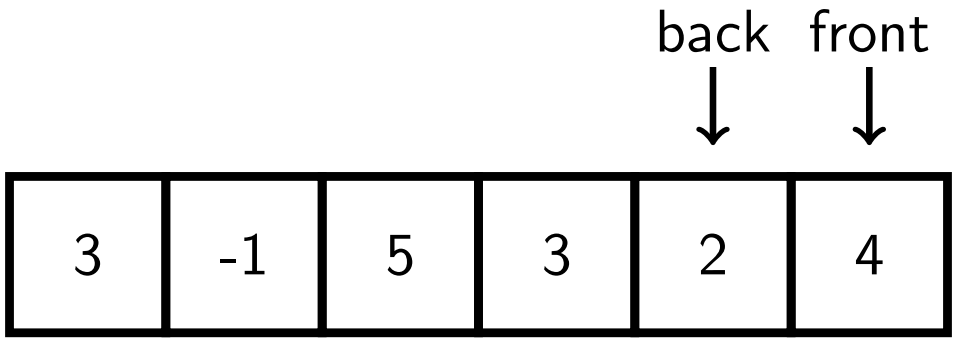


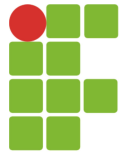
Inserção no início



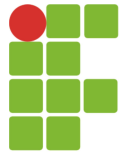


Inserção no início



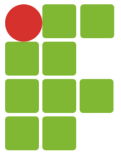


Sumário

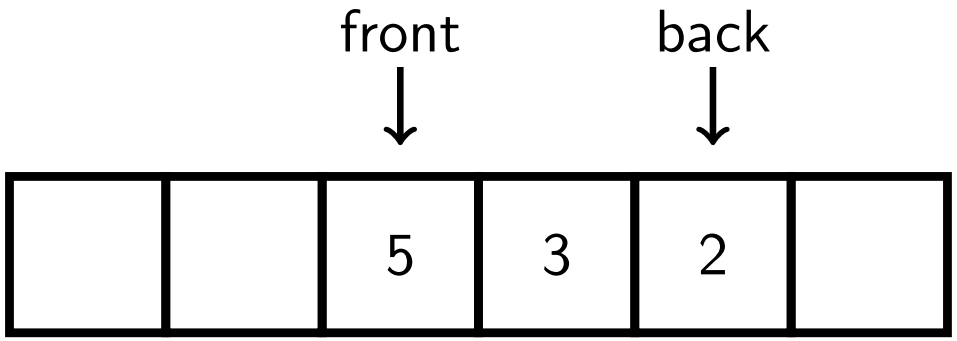


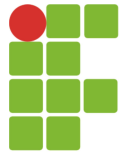
Remoção no início em vetores

- Remover do início consiste em incrementar o índice do início.

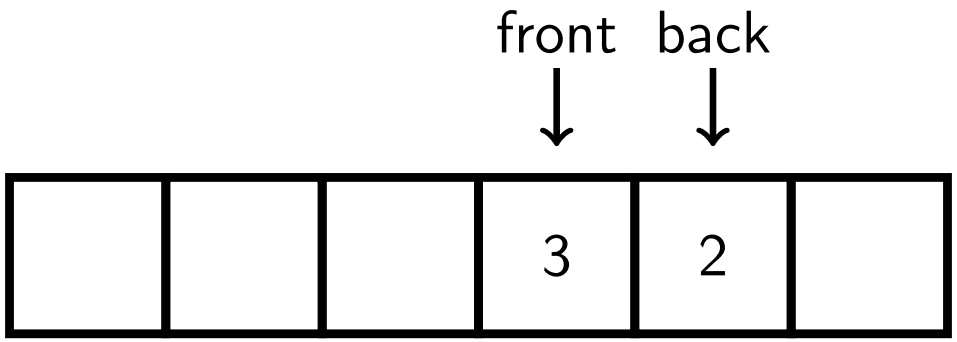


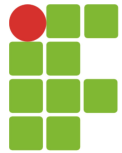
Remoção no início em vetores





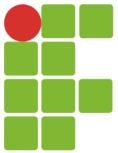
Remoção no início em vetores



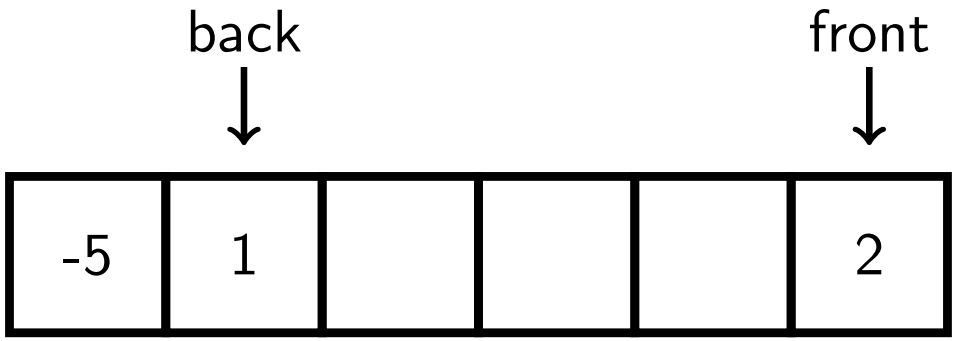


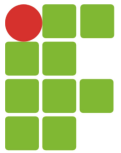
Remoção no início

- Utilizamos a mesma estratégia do enfileiramento: quando o índice do início estiver no final do vetor, damos a volta.

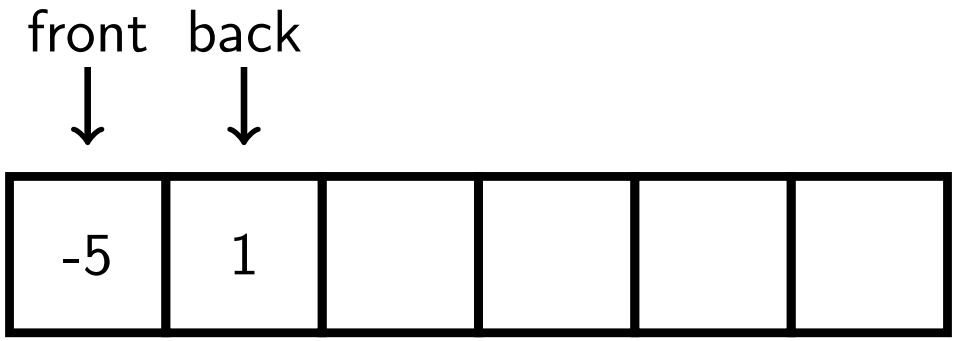


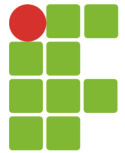
Remoção no início





Remoção no início

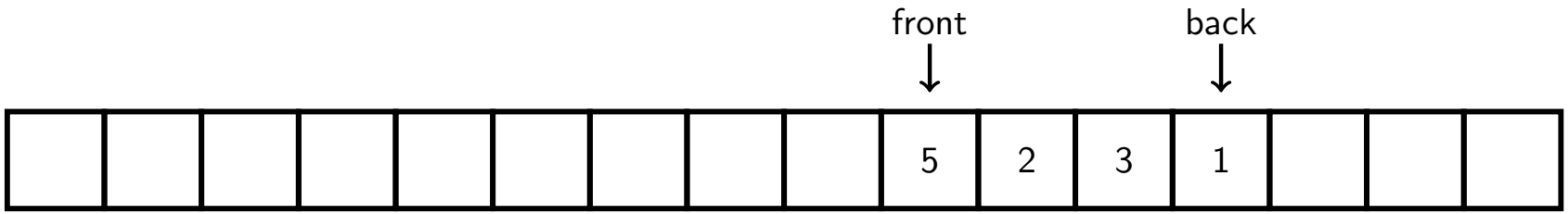


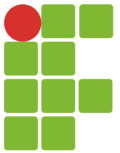


Remoção no início

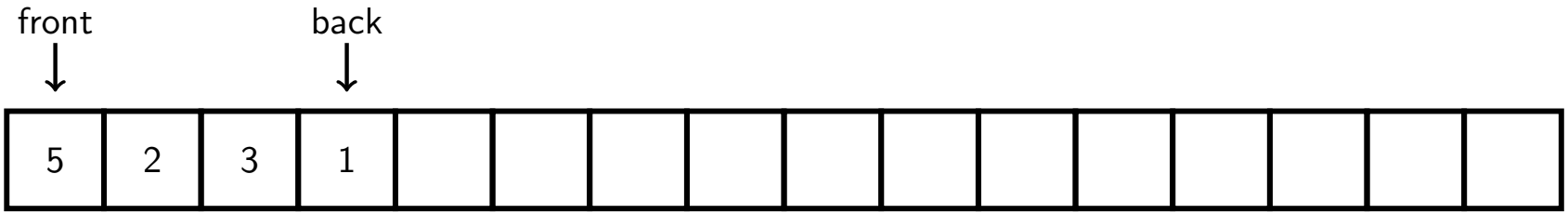
- Quando a ocupação do vetor está em $\frac{1}{4}$ de sua capacidade, devemos reduzir a sua capacidade pela metade.
- Todos os elementos da fila são movidos para o início do vetor realocado, independente da posição dos índices de início e fim.

Remoção no início

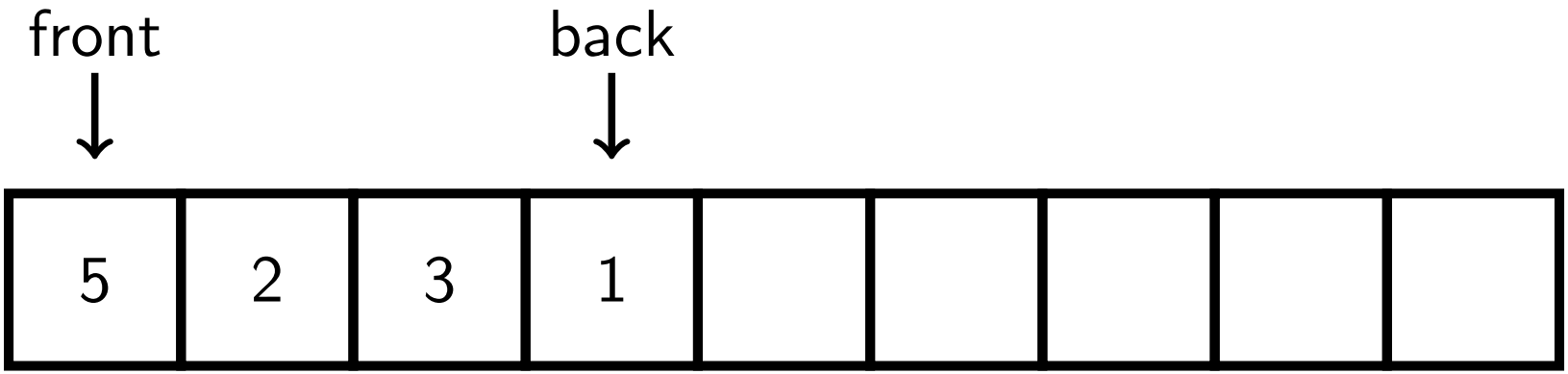


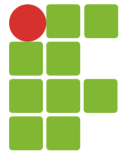


Remoção no início

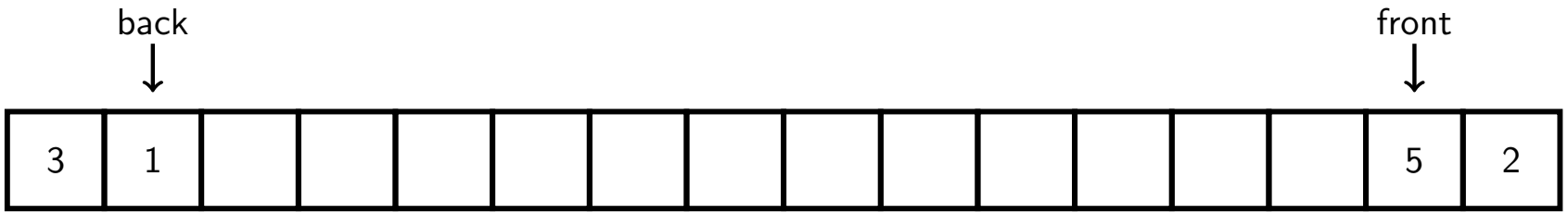


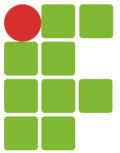
Remoção no início



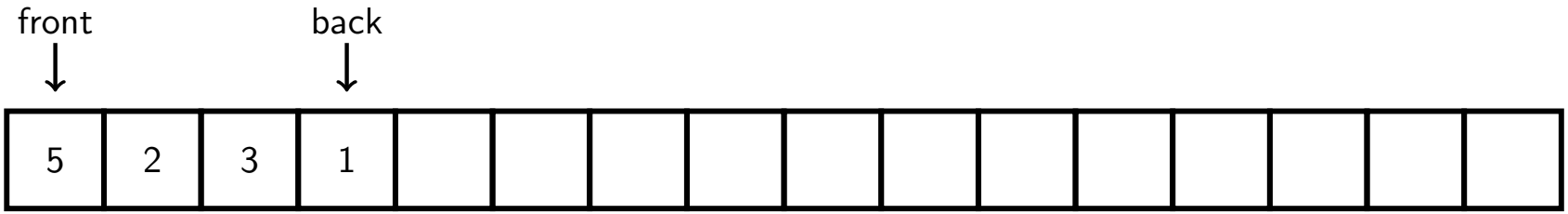


Remoção no início

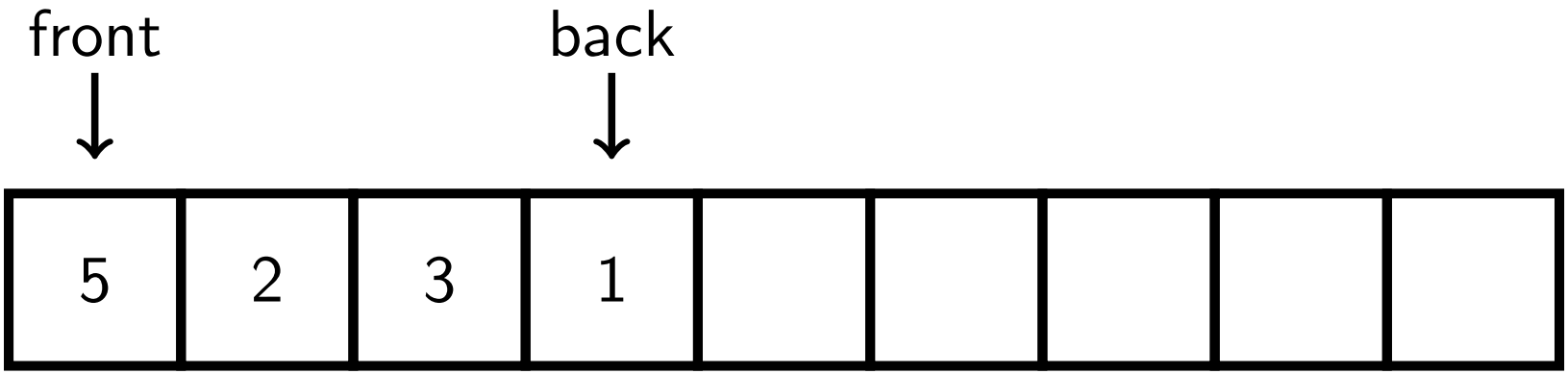


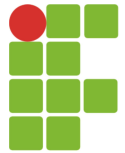


Remoção no início



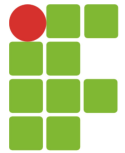
Remoção no início



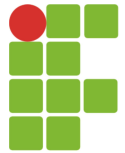


Remoção no início

- Copiar os elementos ao reduzir o vetor pela metade não é uma operação tão custosa ao longo do tempo, pois só fazemos isso quando o vetor está a $\frac{1}{4}$ de sua capacidade. Dessa forma, o custo é amortizado.

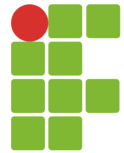


Sumário

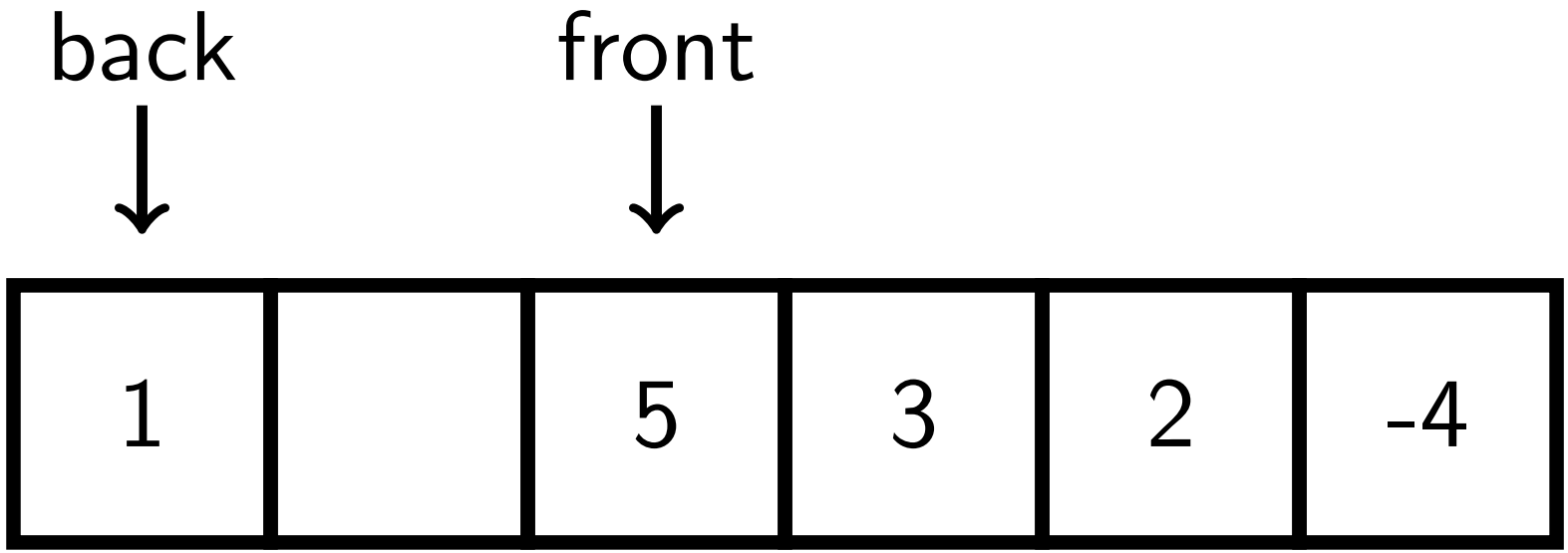


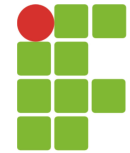
Remoção no fim

- Extremamente similar à remoção no início.
- Até quando é necessário diminuir a capacidade do vetor.
- A diferença é que precisamos incrementar o índice de fim, em vez de decrementar o índice de início.
- Caso o índice de fim chegue ao final do vetor, ele volta para o início.

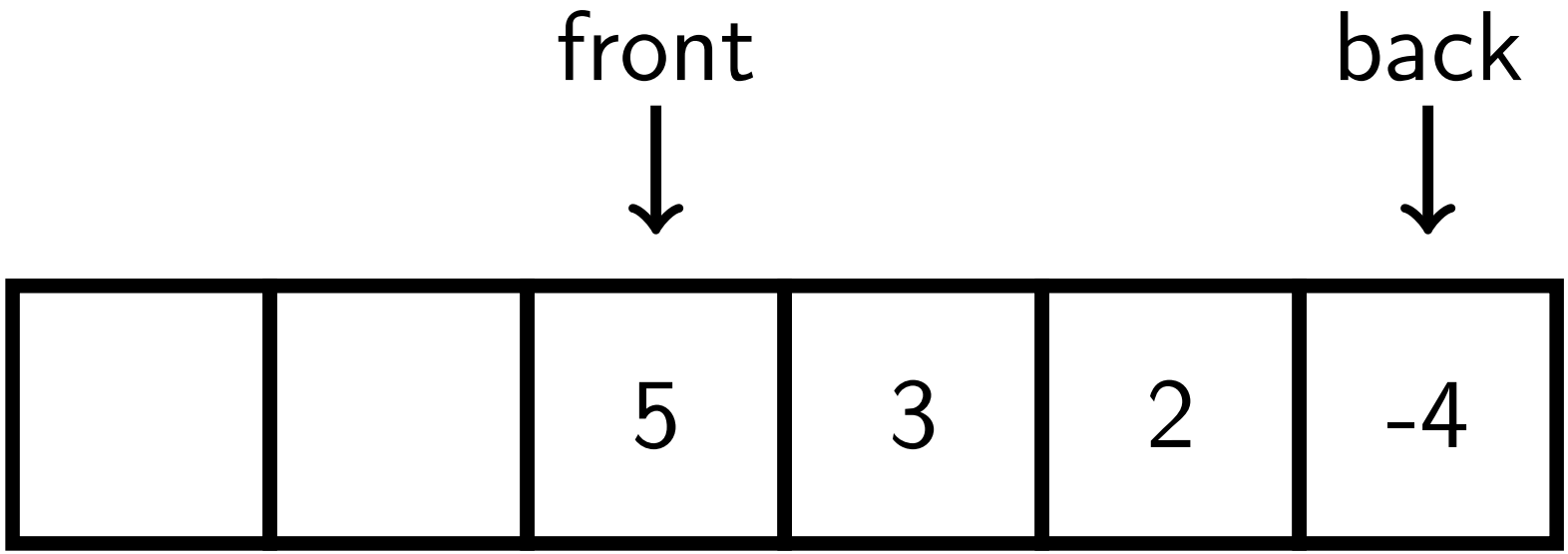


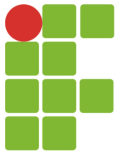
Remoção no início



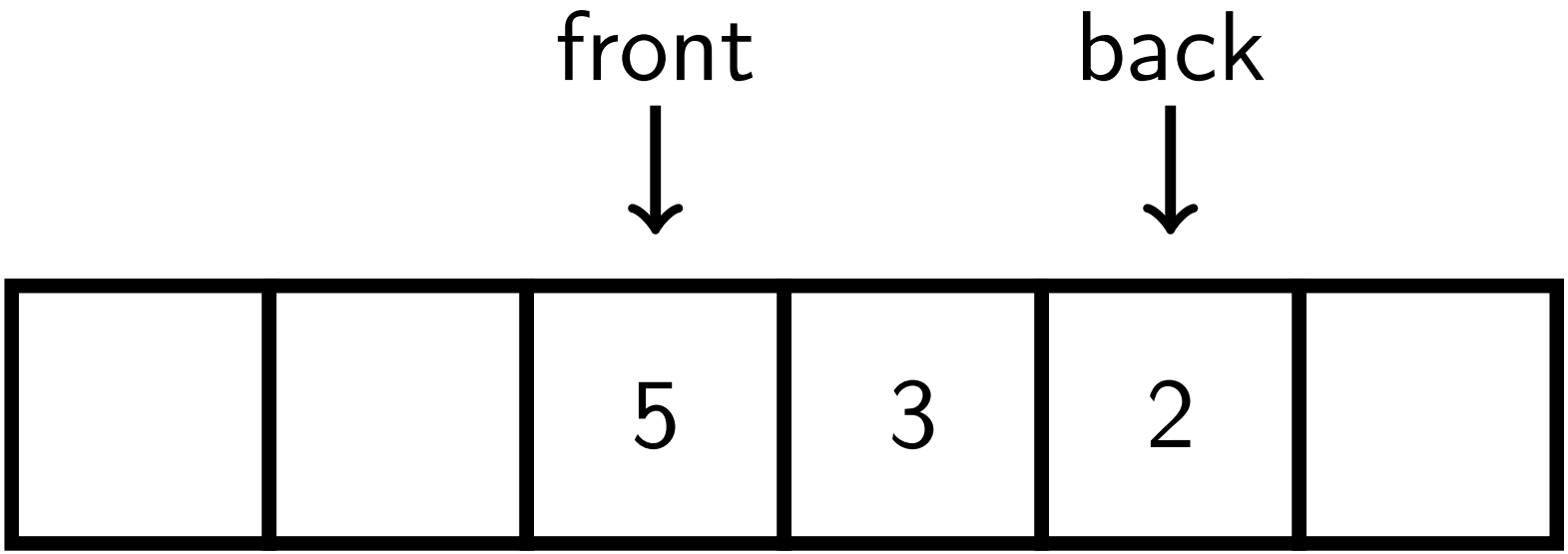


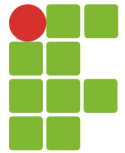
Remoção no início



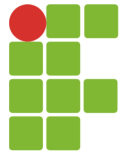


Remoção no início

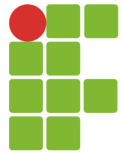




Sumário

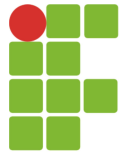


Sumário

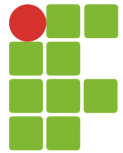


Definição

```
7  typedef struct deque_t {
8      size_t front;
9      size_t back;
10     size_t size;
11     size_t capacity;
12     int *deque;
13 } deque_t;
```

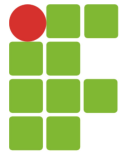


Sumário

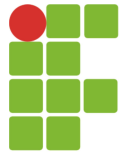


Inicialização

```
41 void deque_initialize(deque_t **d) {
42     (*d) = mallocx(sizeof(deque_t));
43     (*d)->front = 0;
44     (*d)->back = 3;
45     (*d)->size = 0;
46     (*d)->capacity = 4;
47     (*d)->deque = mallocx(sizeof(int) * 4);
48 }
```

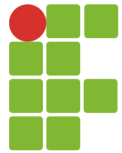


Sumário



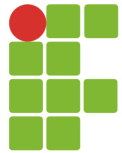
Funções auxiliares

```
106 size_t deque_size(deque_t *d) {
107     return d->size;
108 }
```



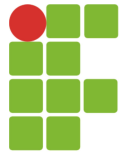
Funções auxiliares

```
110 bool deque_empty(deque_t *d) {
111     return deque_size(d) == 0;
112 }
```



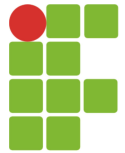
Funções auxiliares

```
8  static void deque_expand(deque_t *d) {
9      size_t old_capacity = d->capacity;
10     d->capacity *= 2;
11     d->deque = reallocx(d->deque, sizeof(int) * d->capacity);
12     if (d->front > d->back) {
13         for (size_t i = d->front; i < old_capacity; i++) {
14             d->deque[i + old_capacity] = d->deque[i];
15         }
16         d->front = d->front + old_capacity;
17     }
18 }
```

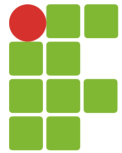


Funções auxiliares

```
20 static void deque_shrink(deque_t *d) {
21     size_t new_capacity = d->capacity / 2;
22     if (d->front <= d->back) {
23         for (size_t i = d->front, j = 0; i <= d->back; i++, j++) {
24             d->deque[j] = d->deque[i];
25         }
26     } else {
27         size_t front_len = d->capacity - d->front;
28         for (int i = d->back; i >= 0; i--) {
29             d->deque[i + front_len] = d->deque[i];
30         }
31         for (size_t i = d->front, j = 0; i < d->capacity; i++, j++) {
32             d->deque[j] = d->deque[i];
33         }
34     }
35     d->front = 0;
36     d->back = d->size - 1;
37     d->capacity = new_capacity;
38     d->deque = reallocx(d->deque, d->capacity * sizeof(int));
39 }
```

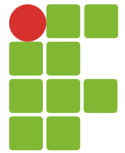


Sumário

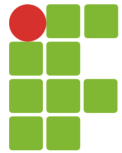


Inserção ao final

```
56 void deque_push_back(deque_t *d, int data) {
57     if (d->size == d->capacity) {
58         deque_expand(d);
59     }
60     d->back++;
61     if (d->back == d->capacity)
62         d->back = 0;
63     d->deque[d->back] = data;
64     d->size++;
65 }
```

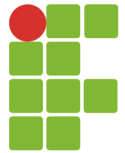


Sumário

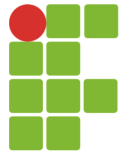


Inserção no início

```
67  void deque_push_front(deque_t *d, int data) {
68      if (d->size == d->capacity) {
69          deque_expand(d);
70      }
71      d->front = d->front == 0 ? d->capacity - 1 : d->front - 1;
72      d->deque[d->front] = data;
73      d->size++;
74  }
```

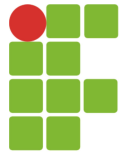


Sumário

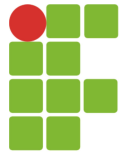


Remoção no início

```
87  void deque_pop_back(deque_t *d) {
88      if (d->size == d->capacity / 4 && d->capacity > 4) {
89          deque_shrink(d);
90      }
91      assert(d->size > 0);
92      d->back = d->back == 0 ? d->capacity - 1 : d->back - 1;
93      d->size--;
94  }
```

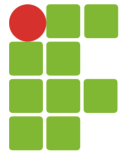


Sumário

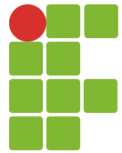


Remoção no início

```
76  void deque_pop_front(deque_t *d) {
77      assert(d->size > 0);
78      if (d->size == d->capacity / 4 && d->capacity > 4) {
79          deque_shrink(d);
80      }
81      d->front++;
82      d->size--;
83      if (d->front == d->capacity)
84          d->front = 0;
85  }
```

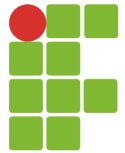


Sumário

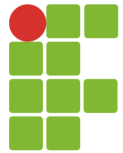


Acessar o início

```
96  int deque_front(deque_t *d) {
97      assert(d->front < d->capacity);
98      return d->deque[d->front];
99  }
```

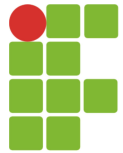


Sumário

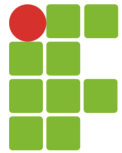


Acessar o final

```
101 int deque_back(deque_t *d) {
102     assert(d->front < d->capacity);
103     return d->deque[d->back];
104 }
```

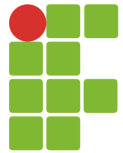


Sumário

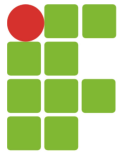


Limpeza

```
50  void deque_delete(deque_t **d) {
51      free((*d)->deque);
52      free(*d);
53      *d = NULL;
54  }
```



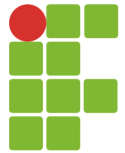
Sumário



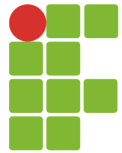
Deques: análise

Complexidade das Operações

Operação	Complexidade
Inserir no final	$\Theta(1)$ amortizado
Inserir no início	$\Theta(1)$ amortizado
Remover no final	$\Theta(1)$ amortizado
Remover no início	$\Theta(1)$ amortizado
Acessar o início	$\Theta(1)$
Acessar o fim	$\Theta(1)$

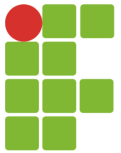


Sumário

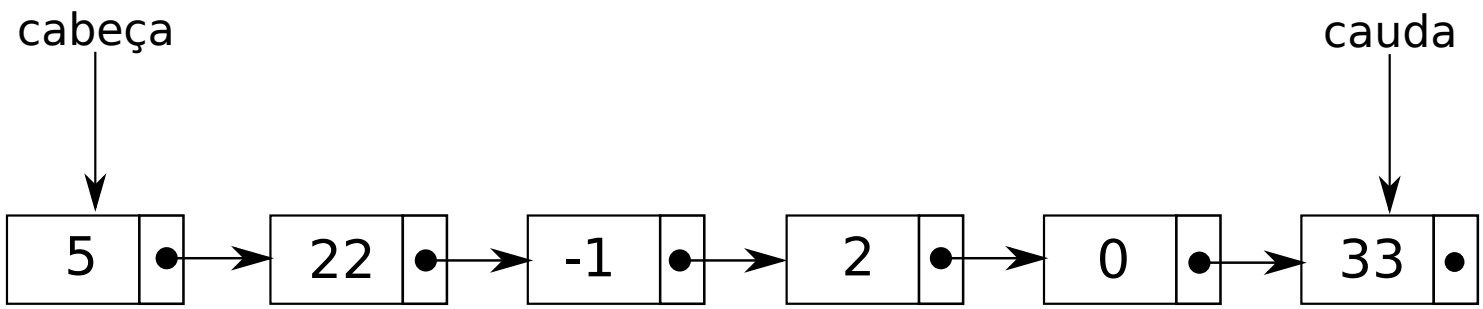


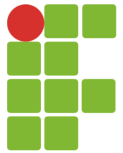
Implementação de deques em listas

- Também podemos utilizar uma lista para implementar um deque.
- Será que uma lista encadeada simples funciona?



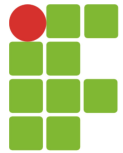
Implementação de Deques





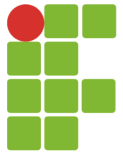
Listas Encadeadas: Análise

Operação	Complexidade
Inserção na cabeça	$\Theta(1)$
Inserção na cauda	$\Theta(1)$
Remoção da cabeça	$\Theta(1)$
Remoção da cauda	$\Theta(n)$
Acesso à cabeça	$\Theta(1)$
Acesso à cauda	$\Theta(1)$



Implementação de deques em listas

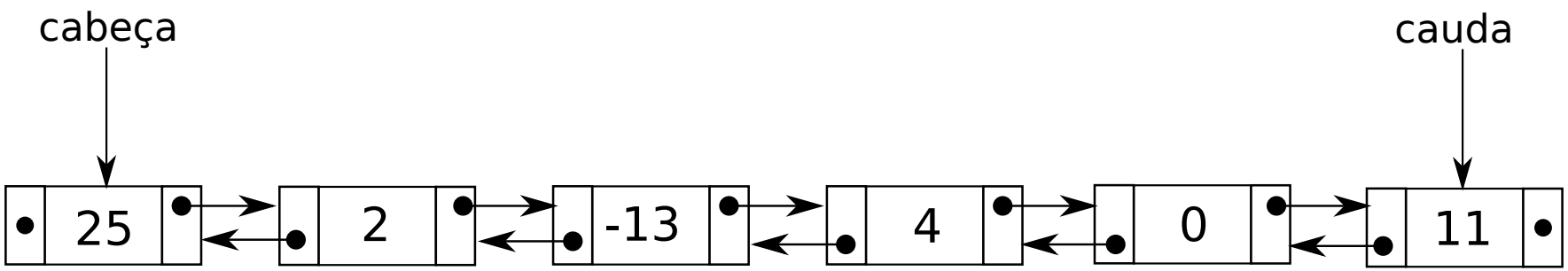
- Listas encadeadas simples não permitem remoção da cauda em tempo constante.
- Temos que recorrer às listas duplamente encadeadas!

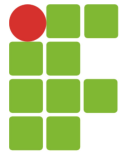


Listas duplamente encadeadas: Análise

Operação	Complexidade
Inserção na cabeça	$\Theta(1)$
Inserção na cauda	$\Theta(1)$
Remoção da cabeça	$\Theta(1)$
Remoção da cauda	$\Theta(1)$
Acesso à cabeça	$\Theta(1)$
Acesso à cauda	$\Theta(1)$

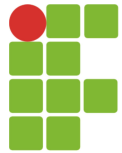
Implementação de Deques





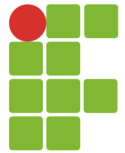
Implementação de Deques

- Listas duplamente encadeadas se mostram uma escolha certa para implementação de Deques!

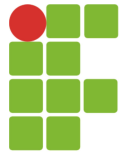


Implementação de deques em listas

- Inserir ao final: inserção na cauda.
- Inserir no início: inserção na cabeça.
- Remover do final: remoção da cauda.
- Remover do início: remoção da cabeça.
- Acesso no início: acesso à cabeça.
- Acesso no final: acesso à cauda.
- Verificar o tamanho do deque: verificar o tamanho da lista.
- Verificar se o deque está vazio: verificar se a lista é vazia.

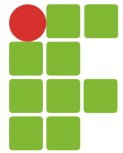


Sumário



Exemplo de utilização da biblioteca

```
1  #include "alloc.h"
2  #include "deque.h"
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void) {
7      int i;
8      deque_t *d;
9      deque_initialize(&d);
10     for (i = 0; i < 16; i++) {
11         if (i % 2 == 0) {
12             printf("Inserindo %d na frente do deque\n", i);
13             deque_push_front(d, i);
14         } else {
15             printf("Inserindo %d atras do deque\n", i);
16             deque_push_back(d, i);
17         }
18     }
```



Exemplo de utilização da biblioteca

```
19      i =0;
20      while (!deque_empty(d)) {
21          if (i % 2 == 1) {
22              printf("Retirando %d da frente do deque\n", deque_front(d));
23              deque_pop_front(d);
24          } else {
25              printf("Retirando %d de tras do deque\n", deque_back(d));
26              deque_pop_back(d);
27          }
28          i++;
29      }
30      deque_delete(&d);
31      return 0;
32  }
```