

Aula 13: Estruturas de dados básicas, parte 1

David Déharbe

Programa de Pós-graduação em Sistemas e Computação

Universidade Federal do Rio Grande do Norte

Centro de Ciências Exatas e da Terra

Departamento de Informática e Matemática Aplicada

Download me from <http://DavidDeharbe.github.io>.



Plano da aula

Introdução

Pilhas

Filas

Filas de duas entradas



PPgSC

Motivação

- ▶ Diferentes aplicações computacionais utilizam matrizes e álgebra linear para modelar e calcular
 - ▶ previsão do tempo
 - ▶ engenho de busca
 - ▶ jogos
 - ▶ análise estática de programas
 - ▶ etc.
- ▶ Como representar uma matriz?

Motivação

- ▶ Diferentes aplicações computacionais utilizam matrizes e álgebra linear para modelar e calcular
 - ▶ previsão do tempo
 - ▶ engenho de busca
 - ▶ jogos
 - ▶ análise estática de programas
 - ▶ etc.
- ▶ Como representar uma matriz?
 - ▶ arranjo de arranjos
 - ▶ lista encadeada das entradas não nulas de cada linha e de cada coluna.
- ▶ Qual a melhor forma?



Motivação

- ▶ Diferentes aplicações computacionais utilizam matrizes e álgebra linear para modelar e calcular
 - ▶ previsão do tempo
 - ▶ engenho de busca
 - ▶ jogos
 - ▶ análise estática de programas
 - ▶ etc.
- ▶ Como representar uma matriz?
 - ▶ arranjo de arranjos
 - ▶ lista encadeada das entradas não nulas de cada linha e de cada coluna.
- ▶ Qual a melhor forma? Assume matrizes de dimensão n , sendo que a probabilidade de uma entrada ser 0 é p . Calcule:
 - ▶ Quantidade de espaço utilizado
 - ▶ Custo da multiplicação de duas matrizes

T_n e T_p são o tamanho da representação de um número e de um ponteiro.



Caracterização

Ref: Cormen et al. Capítulo 11.

Como implementar uma coleção homogênea de dados?

- ▶ **coleção** conjunto de dados armazenados
- ▶ Evolução dinâmica do conjunto armazenado
- ▶ **homogênea** todos os dados têm um mesmo tipo
- ▶ Todos os dados possuem uma **chave**
- ▶ Relação de ordem entre chaves

$$\forall x, y \cdot x < y \vee x > y \vee x = y$$

- ▶ Estrutura entre os dados
- ▶ Operações disponíveis e seu custo



Tipos e operações básicas

tipos

- ▶ chave
- ▶ dado
- ▶ coleção
- ▶ posição na coleção (iterador)

operações



PPgSC

Tipos e operações básicas

- | | |
|-----------|---|
| tipos | ▶ chave
▶ dado
▶ coleção
▶ posição na coleção (iterador) |
| operações | ▶ busca chave em coleção, retorna posição
▶ da primeira ocorrência, ou
▶ da primeira ocorrência a partir da posição dada,
ou
▶ da próxima ocorrência desde a última busca. |



Tipos e operações básicas

- | | |
|-----------|--|
| tipos | ▶ chave
▶ dado
▶ coleção
▶ posição na coleção (iterador) |
| operações | ▶ insere chave em coleção,
▶ depois da posição dada, ou
▶ antes da posição dada, ou
▶ na posição correspondente ao invariante. |



Tipos e operações básicas

- tipos
 - ▶ chave
 - ▶ dado
 - ▶ coleção
 - ▶ posição na coleção (iterador)
- operações
 - ▶ **remove** elemento da coleção
 - ▶ na posição dada, ou
 - ▶ com a chave dada, ou
 - ▶ o elemento correspondente ao invariante.



Tipos e operações básicas

- tipos
- ▶ chave
 - ▶ dado
 - ▶ coleção
 - ▶ posição na coleção (iterador)

- operações
- ▶ **menor** elemento da coleção.
 - ▶ **maior** elemento da coleção.



Tipos e operações básicas

- | | |
|-----------|--|
| tipos | ▶ chave
▶ dado
▶ coleção
▶ posição na coleção (iterador) |
| operações | ▶ posição inicial na coleção.
▶ posição final na coleção.
▶ posição seguinte à posição dada na coleção.
▶ posição anterior à posição dada na coleção.
▶ n-ésima posição na coleção. |



Pilhas



Pilhas



34
12
41
37

remover

12
41
37

inserir 25

25
12
41
37

inserir 9

9
25
12
41
37

Interface

Pilhas

- ▶ Tipos
 - ▶ chave, dado, coleção
- ▶ Operações
 - ▶ Inserção (*push*)
 - ▶ falha em pilha cheia
 - ▶ Remoção (*pop*)
 - ▶ elemento mais recentemente inserido (*LIFO*)
 - ▶ falha em pilha vazia
 - ▶ Acesso (*top*)
 - ▶ elemento mais recentemente inserido
 - ▶ falha em pilha vazia
 - ▶ Consultas ao estado
 - ▶ vazia
 - ▶ cheia



Implementação

Pilhas

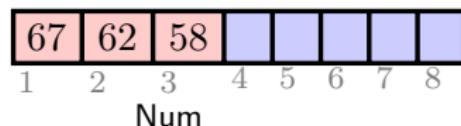
P.Data // arranjo com os elementos na pilha
P.Num // número de elementos na pilha
P.Size // número máximo de elementos na pilha
 $0 \leq P.Num \leq P.Size$

As posições de um arranjo são numeradas a partir de 1.



Simulação

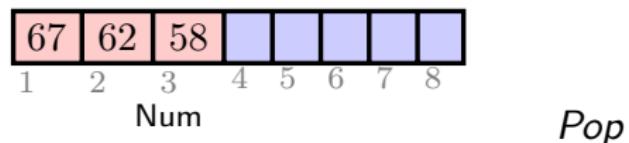
Pilhas



PPgSC

Simulação

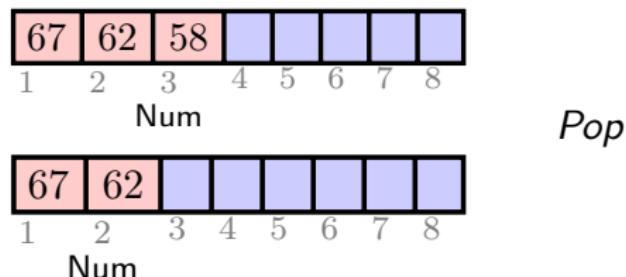
Pilhas



PPgSC

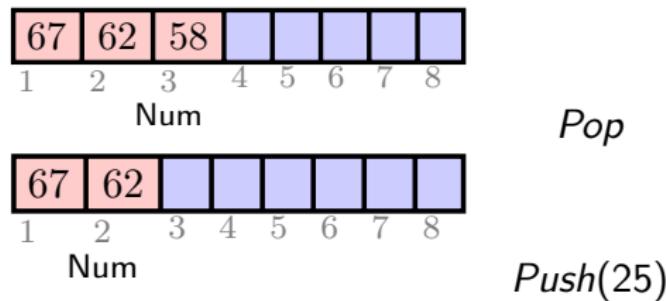
Simulação

Pilhas



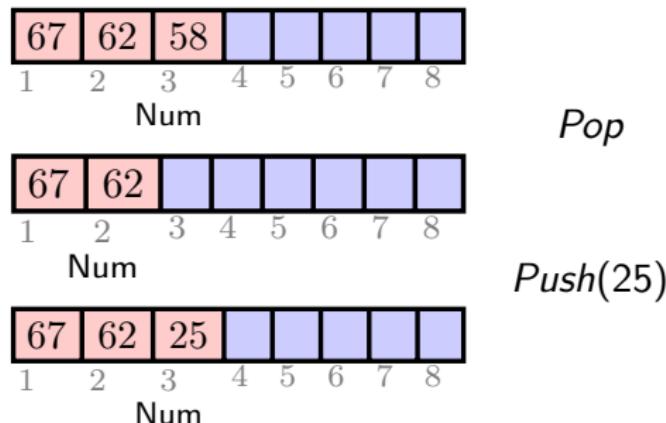
Simulação

Pilhas



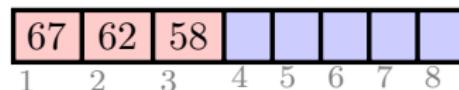
Simulação

Pilhas

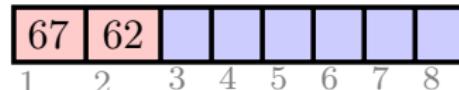


Simulação

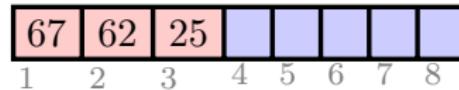
Pilhas



Pop



Push(25)

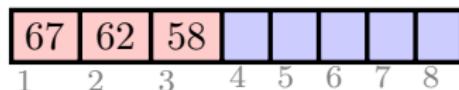


Push(9)

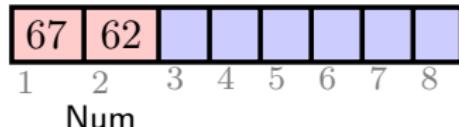


Simulação

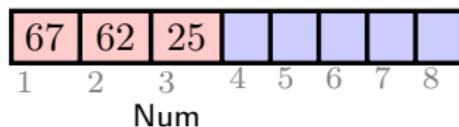
Pilhas



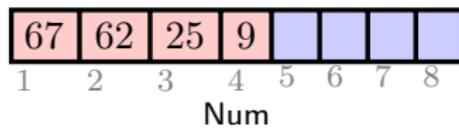
Pop



Push(25)



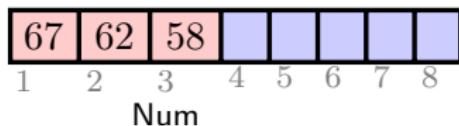
Push(9)



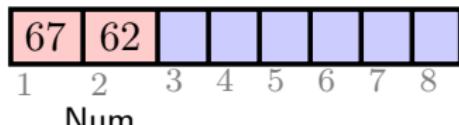
Pop

Simulação

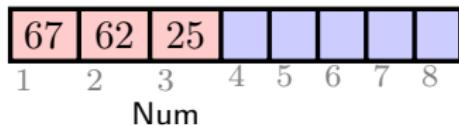
Pilhas



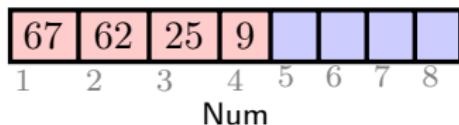
Pop



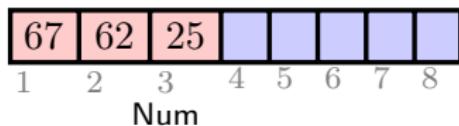
Push(25)



Push(9)



Pop



Implementação

Pilhas

$\text{INIT}(P)$

1 $P.\text{Num} = 0$



PPgSC

Implementação

Pilhas

$\text{INIT}(P)$

1 $P.\text{Num} = 0$

$\text{EMPTY}(P)$

1 **return** $P.\text{Num} == 0$



Implementação

Pilhas

$\text{INIT}(P)$

1 $P.\text{Num} = 0$

$\text{EMPTY}(P)$

1 **return** $P.\text{Num} == 0$

$\text{FULL}(P)$

1 **return** $P.\text{Num} == P.\text{Size}$



Implementação

Pilhas

```
TOP( $P$ )
1   if  $P.Num > 0$ 
2       return  $P.Data[P.Num]$ 

INIT( $P$ )
1    $P.Num = 0$ 

EMPTY( $P$ )
1   return  $P.Num == 0$ 

FULL( $P$ )
1   return  $P.Num == P.Size$ 
```



Implementação

Pilhas

	TOP(P)
INIT(P)	1 if $P.Num > 0$
1 $P.Num = 0$	2 return $P.Data[P.Num]$
	PUSH(P, v)
EMPTY(P)	1 if $P.Num < P.Size$
1 return $P.Num == 0$	2 $P.Num = P.Num + 1$
	3 $P.Data[P.Num] = v$
FULL(P)	
1 return $P.Num == P.Size$	



Implementação

Pilhas

	TOP(P)
INIT(P)	1 if $P.Num > 0$ 2 return $P.Data[P.Num]$
1 $P.Num = 0$	
	PUSH(P, v)
EMPTY(P)	1 if $P.Num < P.Size$ 2 $P.Num = P.Num + 1$ 3 $P.Data[P.Num] = v$
FULL(P)	
1 return $P.Num == P.Size$	POP(P)
	1 if $P.Num > 0$ 2 $P.Num = P.Num - 1$



Opcionais

Pilhas

- ▶ Tipos
 - ▶ iterador
- ▶ Operações
 - ▶ acesso à primeira posição da pilha;
 - ▶ acesso à última posição da pilha;
 - ▶ tamanho da pilha.
- ▶ Implementação
 - ▶ Redimensionamento dinâmico da capacidade



Filas



Filas



$\langle 34, 12, 41, 37 \rangle$

remover

Filas



$\langle 34, 12, 41, 37 \rangle$
remover $\langle 12, 41, 37 \rangle$
inserir 25

Filas



$\langle 34, 12, 41, 37 \rangle$

remover $\langle 12, 41, 37 \rangle$

inserir 25 $\langle 12, 41, 37, 25 \rangle$

inserir 9

Filas



$\langle 34, 12, 41, 37 \rangle$

remover $\langle 12, 41, 37 \rangle$

inserir 25 $\langle 12, 41, 37, 25 \rangle$

inserir 9 $\langle 12, 41, 37, 25, 9 \rangle$

remover

Filas



$\langle 34, 12, 41, 37 \rangle$

remover $\langle 12, 41, 37 \rangle$

inserir 25 $\langle 12, 41, 37, 25 \rangle$

inserir 9 $\langle 12, 41, 37, 25, 9 \rangle$

remover $\langle 41, 37, 25, 9 \rangle$

- ▶ Tipos
 - ▶ chave, dado, coleção
- ▶ Operações
 - ▶ Inserção (*enqueue*)
 - ▶ falha em fila cheia
 - ▶ Remoção (*dequeue*)
 - ▶ elemento com mais tempo na fila (*FIFO*)
 - ▶ falha em fila vazia
 - ▶ Acesso (*head*)
 - ▶ elemento com mais tempo na fila
 - ▶ falha em pilha vazia
 - ▶ Consultas ao estado
 - ▶ vazia
 - ▶ cheia



Implementação

Filas

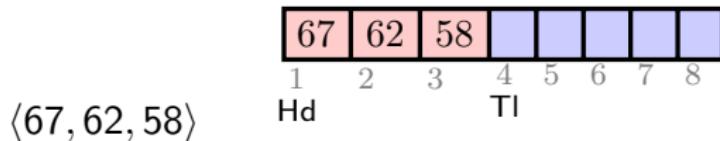
```
Q.Data      // arranjo com os elementos na fila  
Q.Hd        // posição do primeiro elemento  
Q.Tl        // posição do próximo elemento a entrar  
P.Size      // número máximo de elementos na fila  
 $1 \leq Q.Tl \leq Q.Size$   
 $1 \leq Q.Hd \leq Q.Size$   
 $(Q.Tl - Q.Hd) \bmod Q.Size = n$   
// n sendo o número de elementos atualmente na fila.
```

- ▶ Posições ocupadas: $Q.Hd, Q.Hd + 1, \dots, Q.Tl - 1$.
- ▶ No máximo: a capacidade é $Q.Size - 1$.



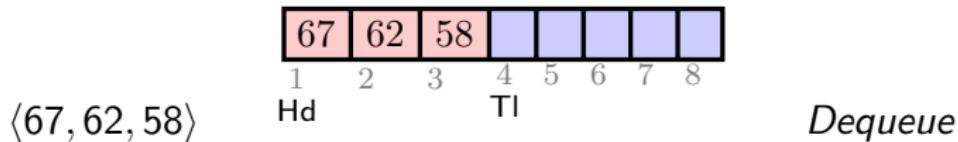
Simulação

Filas



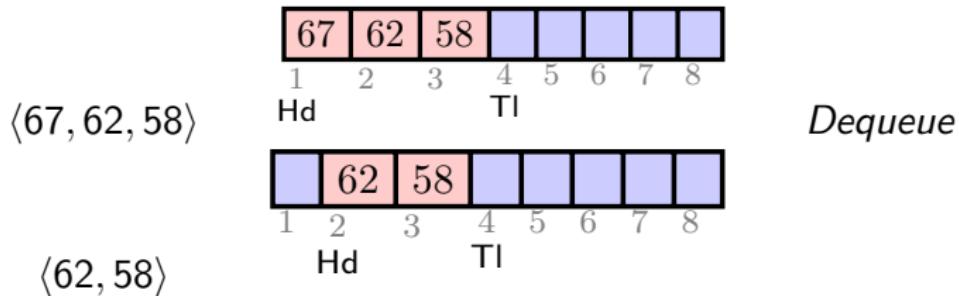
Simulação

Filas



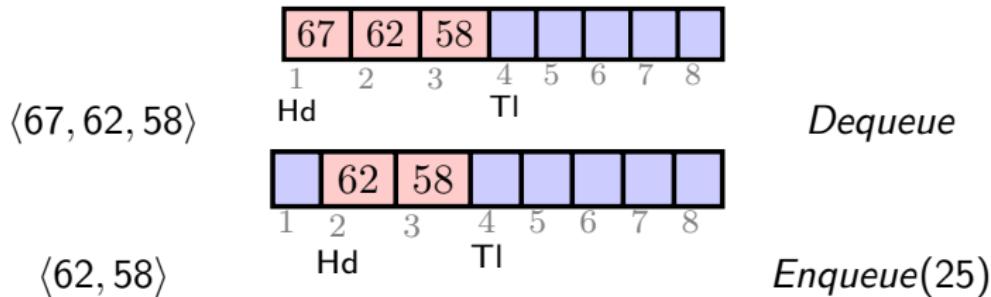
Simulação

Filas



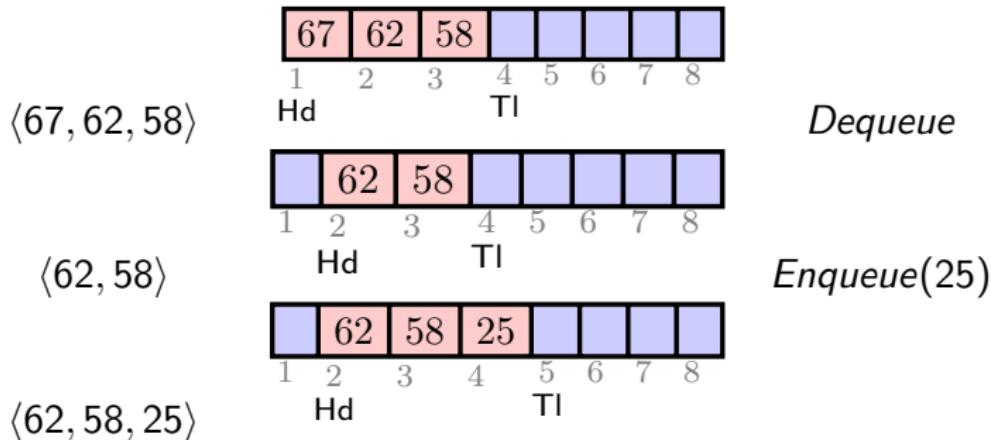
Simulação

Filas



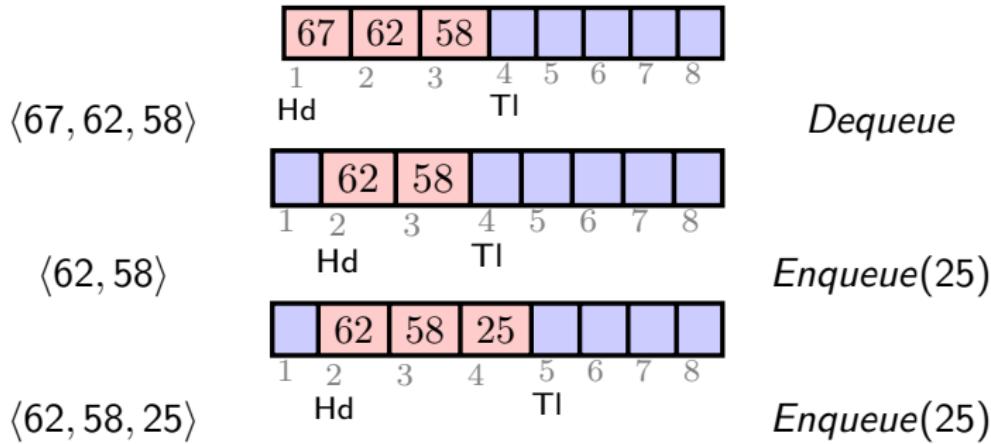
Simulação

Filas



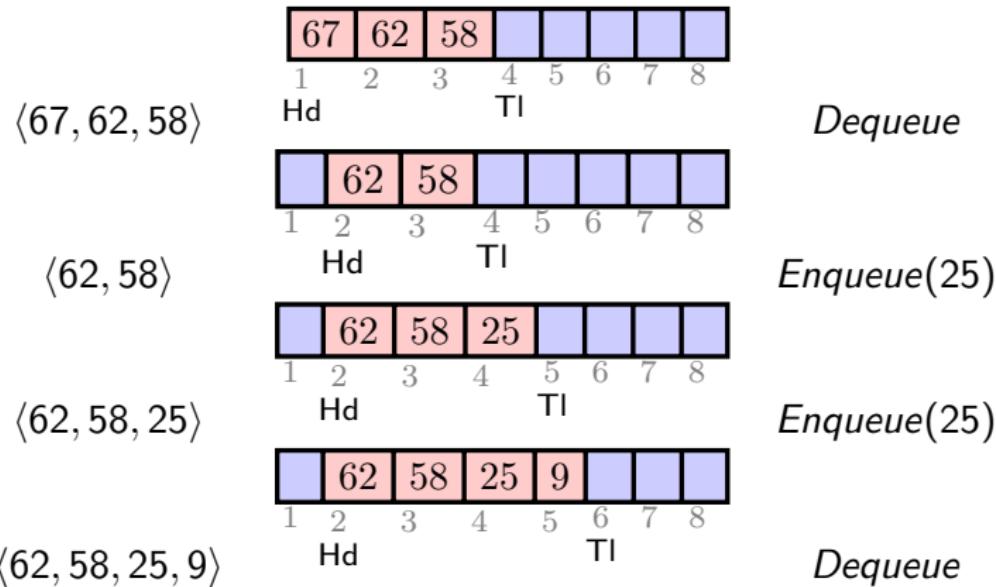
Simulação

Filas



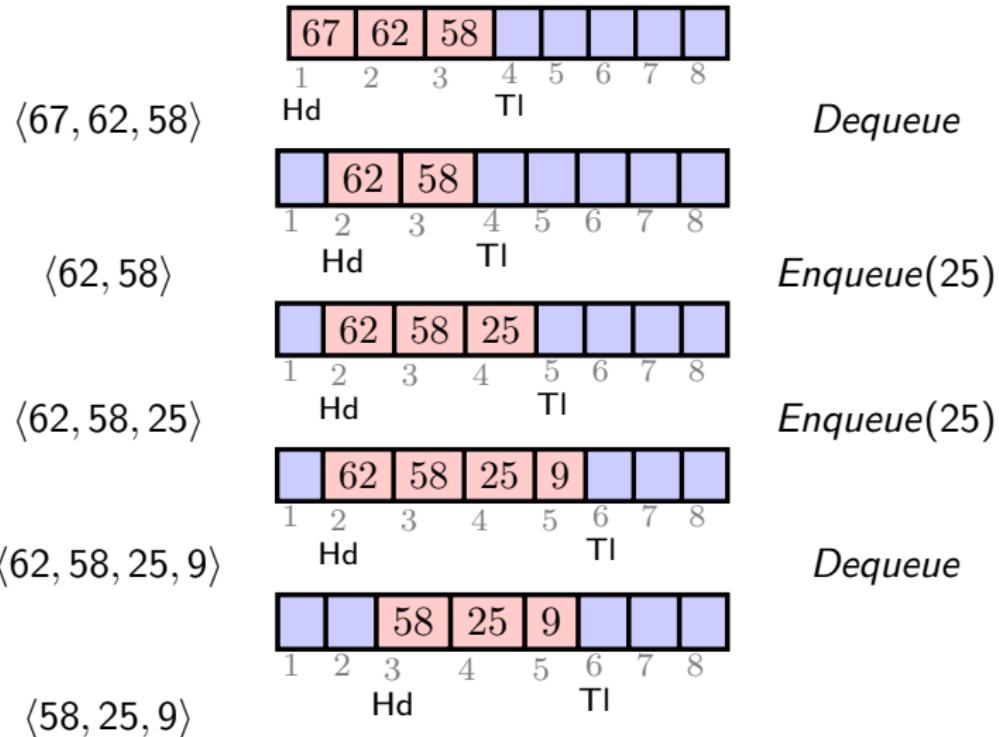
Simulação

Filas



Simulação

Filas



Implementação

Filas

$\text{INIT}(Q)$

- 1 $Q.Hd = 1$
- 2 $Q.Tl = 1$

$\text{EMPTY}(Q)$

- 1 **return** $Q.Hd == Q.Tl$

$\text{FULL}(Q)$

- 1 **return** $Q.Hd == Q.Tl + 1$ or
- 2 $(Q.Hd == 1 \text{ and } Q.Hd == Q.Size)$



Implementação

Filas

$\text{HEAD}(Q)$

- 1 **if** not $\text{EMPTY}(Q)$
- 2 **return** $Q.\text{Data}[Q.\text{Hd}]$

$\text{ENQUEUE}(Q, v)$

- 1 **if** not $\text{FULL}(Q)$
- 2 $Q.\text{Data}[Q.\text{Tl}] = v$
- 3 $Q.\text{Tl} = 1 + (Q.\text{Tl} \bmod Q.\text{Size})$

$\text{DEQUEUE}(Q)$

- 1 **if** not $\text{EMPTY}(Q)$
- 2 $Q.\text{Hd} = 1 + (Q.\text{Hd} \bmod Q.\text{Size})$



Opcionais

Fila

- ▶ Tipos
 - ▶ iterador
- ▶ Operações
 - ▶ acesso à primeira posição da fila;
 - ▶ acesso à última posição da fila;
 - ▶ tamanho da fila.
- ▶ Implementação
 - ▶ Redimensionamento dinâmico da capacidade



Filas de duas entradas

- ▶ *Double-ended queue (deque);*
- ▶ inserção e remoção nas duas pontas da fila;
- ▶ acesso aos elementos nas pontas da fila;
- ▶ pode ser usada para representar fila e pilha.



Deque

- ▶ Tipos
 - ▶ chave, dado, coleção
- ▶ Operações
 - ▶ Inserção na cabeça (*push_front*) e na cauda (*push_back*)
 - ▶ falha em fila cheia
 - ▶ Remoção na cabeça (*pop_front*) e na cauda (*pop_back*)
 - ▶ falha em fila vazia
 - ▶ Acesso na cabeça (*front*) e na cauda (*back*)
 - ▶ falha em pilha vazia
 - ▶ Consultas ao estado
 - ▶ vazia
 - ▶ cheia



PPgSC

Implementação

Deque

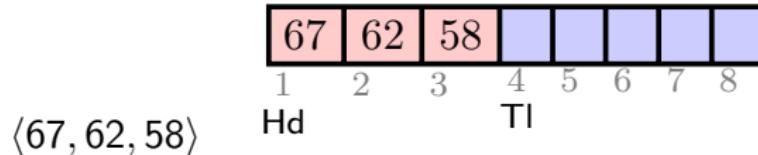
```
Q.Data      // arranjo com os elementos na pilha  
Q.Hd        // posição do primeiro elemento  
Q.Tl        // posição do próximo elemento a entrar  
P.Size      // número máximo de elementos na pilha  
 $1 \leq Q.Tl \leq Q.Size$   
 $1 \leq Q.Hd \leq Q.Size$   
 $(Q.Tl - Q.Hd) \bmod Q.Size = n$   
// n sendo o número de elementos atualmente na fila.
```

- ▶ Posições ocupadas: $Q.Hd, Q.Hd + 1, \dots, Q.Tl - 1$.
- ▶ No máximo: a capacidade é $Q.Size - 1$.



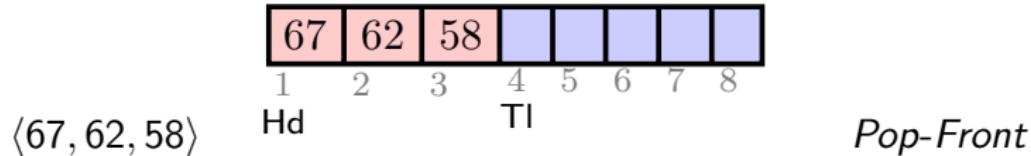
Simulação

Deques



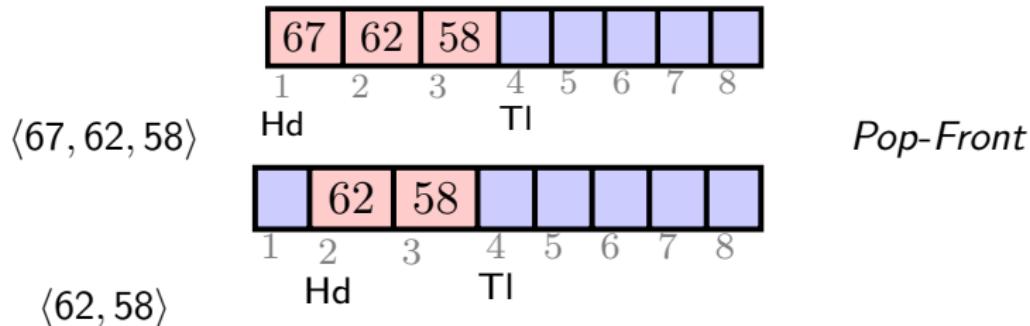
Simulação

Deques



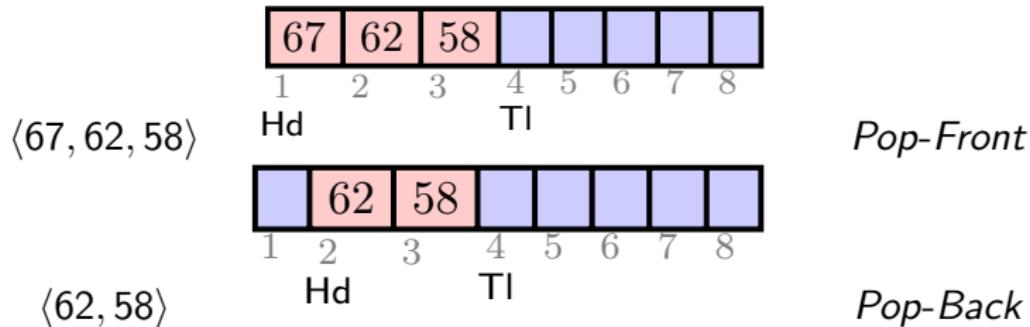
Simulação

Deques



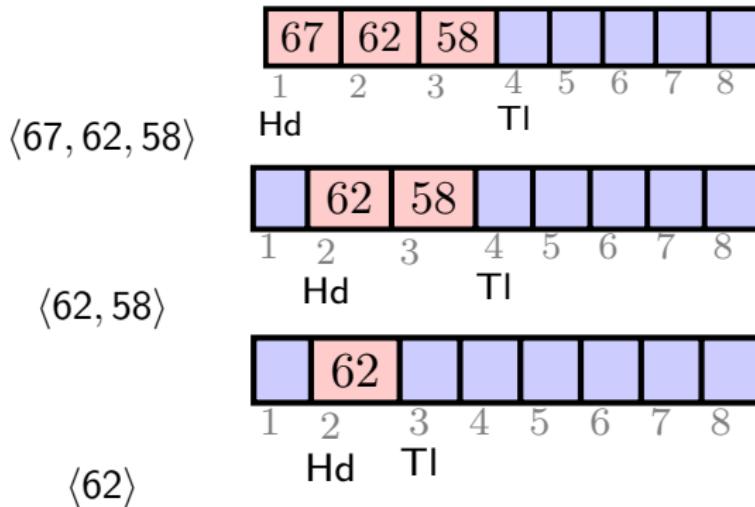
Simulação

Deques



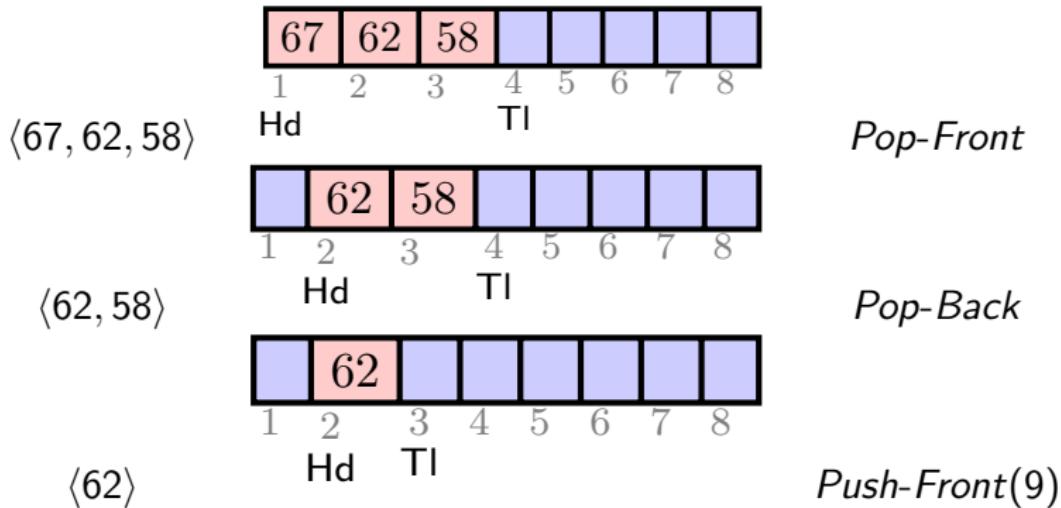
Simulação

Deques



Simulação

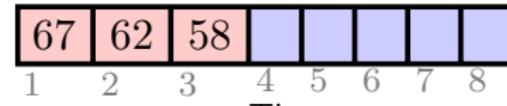
Deques



Simulação

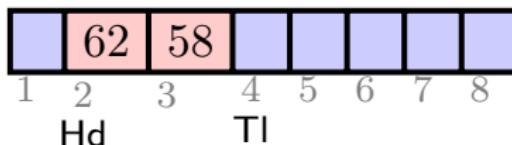
Deques

$\langle 67, 62, 58 \rangle$



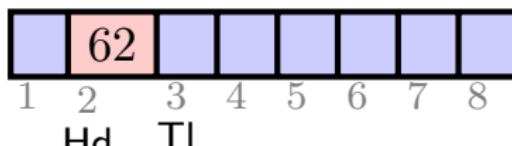
Pop-Front

$\langle 62, 58 \rangle$



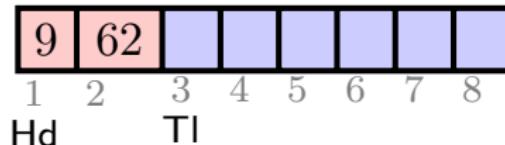
Pop-Back

$\langle 62 \rangle$



Push-Front(9)

$\langle 9, 62 \rangle$

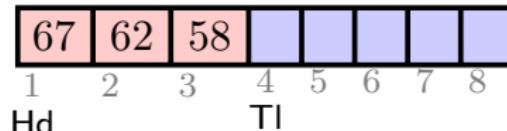


Push-Back(25)

Simulação

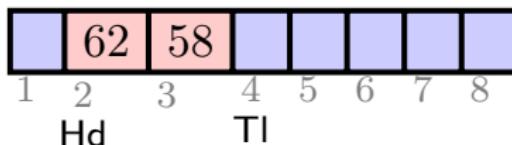
Deques

$\langle 67, 62, 58 \rangle$



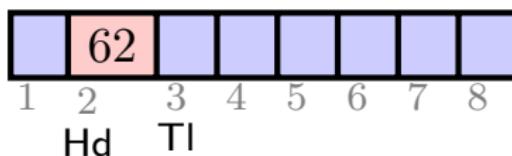
Pop-Front

$\langle 62, 58 \rangle$



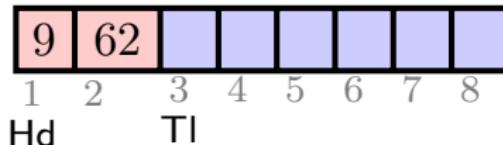
Pop-Back

$\langle 62 \rangle$



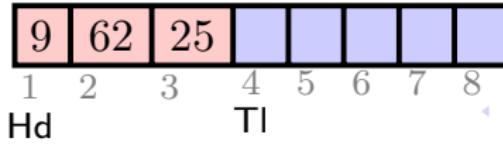
Push-Front(9)

$\langle 9, 62 \rangle$



Push-Back(25)

$\langle 9, 62, 25 \rangle$



Implementação

Deque

$\text{INIT}(Q)$

- 1 $Q.Hd = 1$
- 2 $Q.Tl = 1$

$\text{EMPTY}(Q)$

- 1 **return** $Q.Hd == Q.Tl$

$\text{FULL}(Q)$

- 1 **return** $Q.Hd == Q.Tl + 1$ or
- 2 $(Q.Hd == 1 \text{ and } Q.Hd == Q.Size)$



Implementação

Deque

FRONT(Q)

- 1 **if** not EMPTY(Q)
- 2 **return** $Q.\text{Data}[Q.\text{Hd}]$

PUSH-BACK(Q, v)

- 1 **if** not FULL(Q)
- 2 $Q.\text{Data}[Q.\text{Tl}] = v$
- 3 $Q.\text{Tl} = 1 + (Q.\text{Tl} \bmod Q.\text{Size})$

POP-FRONT(Q)

- 1 **if** not EMPTY(Q)
- 2 $Q.\text{Hd} = 1 + (Q.\text{Hd} \bmod Q.\text{Size})$



Implementação

Deque

BACK(Q)

```
1 if not EMPTY( $Q$ )
2     return  $Q.Data[Q.Tl]$ 
```

PUSH-FRONT(Q, v)

```
1 if not FULL( $Q$ )
2      $Q.Data[Q.Hd] = v$ 
3     if  $Q.Hd == 1$ 
4          $Q.Hd = Q.Size$ 
5     else  $Q.Hd = Q.Hd - 1$ 
```

POP-BACK(Q)

```
1 if not EMPTY( $Q$ )
2     if  $Q.Tl == 1$ 
3          $Q.Tl = Q.Size$ 
4     else  $Q.Tl = Q.Hd - 1$ 
```

Exercícios

- ▶ É possível implementar as funcionalidades de fila utilizando pilha(s)? Se sim, como? Se não, por que?
- ▶ É possível implementar as funcionalidades de pilha utilizando fila(s)? Se sim, como? Se não, por que?
- ▶ É possível implementar as funcionalidades de deque utilizando pilha(s) e fila(s)? Se sim, como? Se não, por que?

