

Pilhas, Filas e Deques

Conceitos e implementação

Prof. Marcelo de Souza

45EST – Algoritmos e Estruturas de Dados
Universidade do Estado de Santa Catarina





Leitura principal:

- ▶ Capítulo 6 de [Goodrich et al. \(2014\)](#)¹ – Pilhas, filas e dequeues.

Leitura complementar:

- ▶ Capítulo 6 de [Preiss \(2001\)](#)² – Pilhas, filas e dequeues.

¹Michael T Goodrich et al. (2014). *Data structures and algorithms in Java*. 6ª ed. John Wiley & Sons.

²Bruno R Preiss (2001). *Estruturas de dados e algoritmos: padrões de projetos orientados a objetos com Java*. Campus.



Estruturas sequenciais

Ou seja, TADs sequenciais

Vamos construir tipos abstratos de dados (TADs) sequenciais que usam as estruturas de dados fundamentais (arranjos e listas encadeadas) para a implementação de estruturas mais abstratas e com comportamentos específicos.



Estruturas sequenciais

Ou seja, TADs sequenciais

Vamos construir tipos abstratos de dados (TADs) sequenciais que usam as estruturas de dados fundamentais (arranjos e listas encadeadas) para a implementação de estruturas mais abstratas e com comportamentos específicos.

Em particular, implementaremos:

- ▶ **Pilhas** (*stacks*);
- ▶ **Filas** (*queues*);
- ▶ **Deque** (*double ended queues*).



Estruturas sequenciais

Ou seja, TADs sequenciais

Vamos construir tipos abstratos de dados (TADs) sequenciais que usam as estruturas de dados fundamentais (arranjos e listas encadeadas) para a implementação de estruturas mais abstratas e com comportamentos específicos.

Em particular, implementaremos:

- ▶ **Pilhas** (*stacks*);
- ▶ **Filas** (*queues*);
- ▶ **Deque** (*double ended queues*).

Essas estruturas se diferenciam pela **política de inserção e remoção dos dados** adotada. Ou seja, somente serão permitidas as operações de interesse.

Pilhas

Ideia geral

Uma **pilha** implementa a política **LIFO** – *last in, first out*. Ou seja, o último elemento adicionado à estrutura é o primeiro a ser removido.



Pilhas

Ideia geral

Uma **pilha** implementa a política **LIFO** – *last in, first out*. Ou seja, o último elemento adicionado à estrutura é o primeiro a ser removido.

A pilha simula uma “pilha de objetos”, logo ela tem um **topo**.



Pilhas

Ideia geral

Uma **pilha** implementa a política **LIFO** – *last in, first out*. Ou seja, o último elemento adicionado à estrutura é o primeiro a ser removido.

A pilha simula uma “pilha de objetos”, logo ela tem um **topo**.

Operações:

- ▶ **push**: insere um elemento no topo da pilha.
- ▶ **pop**: remove (e retorna) o elemento do topo da pilha.
- ▶ **top**: retorna o elemento do topo da pilha (sem remover).



Pilhas

Ideia geral

Uma **pilha** implementa a política **LIFO** – *last in, first out*. Ou seja, o último elemento adicionado à estrutura é o primeiro a ser removido.

A pilha simula uma “pilha de objetos”, logo ela tem um **topo**.

Operações:

- ▶ **push**: insere um elemento no topo da pilha.
- ▶ **pop**: remove (e retorna) o elemento do topo da pilha.
- ▶ **top**: retorna o elemento do topo da pilha (sem remover).

Aplicações:

- ▶ Histórico de acesso de um navegador (voltar, avançar).
- ▶ Operações em um editor de texto (desfazer, refazer).

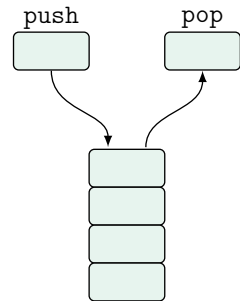


Pilhas

Exemplo de funcionamento

Seja uma pilha implementada usando uma lista (sequencial), onde o topo é o final da lista.

Método	Retorno	Conteúdo da pilha
push(5)	—	(5)
push(3)	—	(5, 3)
size()	2	(5, 3)
pop()	3	(5)
isEmpty()	false	(5)
pop()	5	()
isEmpty()	true	()
pop()	null	()
push(7)	—	(7)
push(9)	—	(7, 9)
top()	9	(7, 9)
push(4)	—	(7, 9, 4)
size()	3	(7, 9, 4)
pop()	4	(7, 9)
push(6)	—	(7, 9, 6)
push(8)	—	(7, 9, 6, 8)
pop()	8	(7, 9, 6)



(representação gráfica)

Interface Stack:

```
1  public interface Stack<E> {  
2      int size();  
3      boolean isEmpty();  
4      void push(E e);  
5      E top();  
6      E pop();  
7  }
```

Usamos uma interface para definir nosso TAD pilha. A interface só define a API (métodos), a qual pode ser implementada de diversas formas (e usando diferentes estruturas).

Implementação usando arranjos

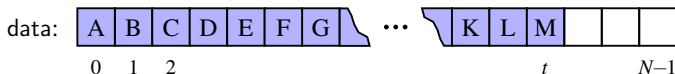
```
1  public class ArrayStack<E> implements Stack<E> {  
2      public static final int CAPACITY = 1000;  
3      private E[] data;  
4      private int t = -1;  
5  
6      public ArrayStack() { this(CAPACITY); }  
7  
8      public ArrayStack(int capacity) {  
9          data = (E[]) new Object[capacity];  
10     }  
11  
12     public int size() { return (t + 1); }  
13     public boolean isEmpty() { return (t == -1); }  
14  
15     // ...  
16 }
```

Pilhas

Implementação usando arranjos

```
1 public class ArrayStack<E> implements Stack<E> {
2     public static final int CAPACITY = 1000;
3     private E[] data;
4     private int t = -1;
5
6     public ArrayStack() { this(CAPACITY); }
7
8     public ArrayStack(int capacity) {
9         data = (E[]) new Object[capacity];
10    }
11
12    public int size() { return (t + 1); }
13    public boolean isEmpty() { return (t == -1); }
14
15    // ...
16 }
```

- ▶ ArrayStack implementa a interface Stack.
- ▶ A pilha possui uma capacidade fixa, dada pela constante CAPACITY ou definida pelo usuário no construtor parametrizado.
- ▶ Os dados são armazenados no *array* data.
- ▶ O atributo t indica o topo da pilha, armazenando o índice do elemento do topo.
- ▶ O tamanho da pilha é dado por $t + 1$ (conforme método size).
- ▶ A lista é vazia quando não há elementos, ou seja, o topo t é -1 (método isEmpty).



Método push:

```
1 public void push(E e) {  
2     if (size() == data.length) throw new IllegalStateException("Stack is full");  
3     data[++t] = e;  
4 }
```

Método push:

```
1 public void push(E e) {  
2     if (size() == data.length) throw new IllegalStateException("Stack is full");  
3     data[++t] = e;  
4 }
```

Método top:

```
1 public E top() {  
2     if (isEmpty()) return null;  
3     return data[t];  
4 }
```

Método push:

```
1 public void push(E e) {  
2     if (size() == data.length) throw new IllegalStateException("Stack is full");  
3     data[++t] = e;  
4 }
```

Método top:

```
1 public E top() {  
2     if (isEmpty()) return null;  
3     return data[t];  
4 }
```

Método pop:

```
1 public E pop() {  
2     if (isEmpty()) return null;  
3     E answer = data[t];  
4     data[t] = null;  
5     t--;  
6     return answer;  
7 }
```




Método toString:

```
1 public String toString() {  
2     StringBuilder sb = new StringBuilder("(");  
3     for (int j = t; j >= 0; j--) {  
4         sb.append(data[j]);  
5         if (j > 0) sb.append(", ");  
6     }  
7     sb.append(")");  
8     return sb.toString();  
9 }
```

- O método apresenta os elementos iniciando pelo topo da pilha (t).

Implementação usando encadeamento

```
1 public class LinkedStack<E> implements Stack<E> {  
2     private SinglyLinkedList<E> list = new SinglyLinkedList<>();  
3     public int size() { return list.size(); }  
4     public boolean isEmpty() { return list.isEmpty(); }  
5     public void push(E element) { list.addFirst(element); }  
6     public E top() { return list.first(); }  
7     public E pop() { return list.removeFirst(); }  
8     public String toString() { return list.toString(); }  
9 }
```

```
1 public class LinkedStack<E> implements Stack<E> {  
2     private SinglyLinkedList<E> list = new SinglyLinkedList<>();  
3     public int size() { return list.size(); }  
4     public boolean isEmpty() { return list.isEmpty(); }  
5     public void push(E element) { list.addFirst(element); }  
6     public E top() { return list.first(); }  
7     public E pop() { return list.removeFirst(); }  
8     public String toString() { return list.toString(); }  
9 }
```

- ▶ Essa implementação usa uma lista simplesmente encadeada.
- ▶ Não há limite de capacidade.
- ▶ Como as operações no início de uma lista simplesmente encadeada são mais eficientes, o topo da pilha é o início da lista.
- ▶ A implementação é simples, pois basta invocar os métodos da lista encadeada que executam as operações desejadas (`size`, `isEmpty`, `addFirst`, ...).



Operação	ArrayStack	LinkedStack
size	$O(1)$	$O(1)$
isEmpty	$O(1)$	$O(1)$
top	$O(1)$	$O(1)$
push	$O(1)$	$O(1)$
pop	$O(1)$	$O(1)$

