45EST – Algoritmos e Estruturas de Dados

Pilhas, filas e deques

Prof. Marcelo de Souza

UDESC Ibirama
Bacharelado em Engenharia de Software

marcelo.desouza@udesc.br
Versão compilada em 13 de agosto de 2020

Leitura obrigatória:

- Capítulo 6 de Goodrich et al. [2014] – Pilhas, filas e deques.

Leitura complementar:

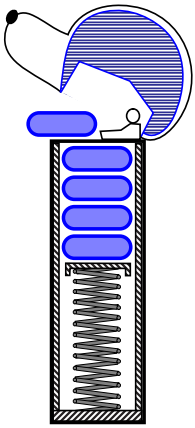
- Capítulo 6 de Preiss [2001] – Pilhas, filas e deques.

Pilhas

Ideia geral:

- Estratégia **LIFO**: *last-in-first-out*.
- Último elemento a ser inserido é o primeiro a ser removido.
- Simples, eficiente e muito utilizada.
- Operações:
  - `push`: insere elemento no topo da pilha.
  - `pop`: remove elemento do topo da pilha.
- Aplicações:
  - Histórico de acesso de um navegador.
  - Operações em um editor de texto (desfazer).

- Exemplo ilustrativo:



- Exemplo de funcionamento:
  - Operações de atualização: `push`, `pop`.
  - Operações auxiliares: `size`, `top`, `isEmpty`.

Method	Return Value	Stack Contents
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)

Interface Stack:

```java
public interface Stack<E> {
   int size();
   boolean isEmpty();
   void push(E e);
   E top();
   E pop();
}
```

Implementação baseada em vetor:

```java
public class ArrayStack<E> implements Stack<E> {
   public static final int CAPACITY = 1000;
   private E[] data;
   private int t = -1;

   public ArrayStack() { this(CAPACITY); }

   public ArrayStack(int capacity) {
      data = (E[]) new Object[capacity];
   }

   public int size() { return (t + 1); }

   public boolean isEmpty() { return (t == -1); }

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

   public E top() {
      if (isEmpty()) return null;
      return data[t];
   }

```

```
27      public E pop() {
28          if (isEmpty()) return null;
29          E answer = data[t];
30          data[t] = null;
31          t--;
32          return answer;
33      }
34  }
```

- Comentários:
  - Simples.
  - Eficiente.
  - Tamanho estático/fixo.
    - Desperdício de memória ou impossibilidade de inserção.
- Análise de eficiência:
  - `size`: $O(1)$.
  - `isEmpty`: $O(1)$.
  - `top`: $O(1)$.
  - `push`: $O(1)$.
  - `pop`: $O(1)$.

Exercícios:

1. Considere um sistema onde o usuário interage através de uma shell, fornecendo os comandos desejados na forma de texto. Implemente uma pilha baseada em vetor para armazenar os comandos informados pelo usuário. Ao digitar o comando '`sair`', o sistema deve desempilhar os comandos e mostrar ao usuário como log do sistema.

Implementação baseada em listas simplesmente encadeadas:

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

- Comentários:
  - Operações continuam com complexidade constante $O(1)$.

Exercícios:

1. Substitua a pilha utilizada no exercício anterior por uma pilha baseada em uma lista simplesmente encadeada.
2. Utilize pilhas para inverter a ordem dos elementos de um vetor de elementos (de qualquer tipo).

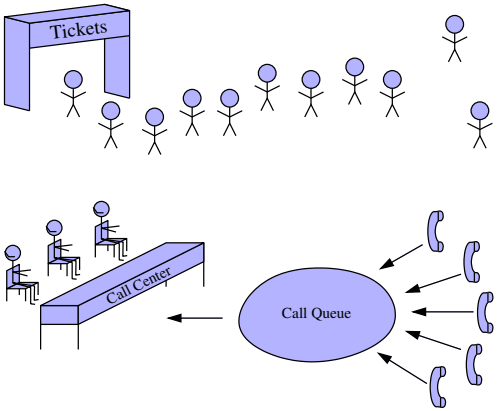
Filas

Ideia geral:

- Estratégia **FIFO**: *first-in-first-out*.
- Primeiro elemento a ser inserido é o primeiro a ser removido.
- Elemento entra no *final* da fila e sai o elemento da *frente* da fila.
- Operações:
  - `enqueue`: insere elemento no final da fila.

  - `dequeue`: remove elemento do início da fila.

- Aplicações:
  - Impressora de rede.
  - Servidor Web respondendo requisições.



- Exemplo de funcionamento:
  - Operações de atualização: `enqueue`, `dequeue`.
  - Operações auxiliares: `size`, `first`, `isEmpty`.

Method	**Return Value**	first $\leftarrow Q \leftarrow$ last
enqueue(5)	–	(5)
enqueue(3)	–	(5, 3)
size( )	2	(5, 3)
dequeue( )	5	(3)
isEmpty( )	false	(3)
dequeue( )	3	( )
isEmpty( )	true	( )
dequeue( )	null	( )
enqueue(7)	–	(7)
enqueue(9)	–	(7, 9)
first( )	7	(7, 9)
enqueue(4)	–	(7, 9, 4)

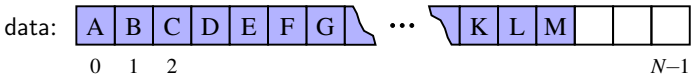
Interface Queue:

```
1  public interface Queue<E> {
2     int size();
3     boolean isEmpty();
4     void enqueue(E e);
5     E first();
6     E dequeue();
7  }
```

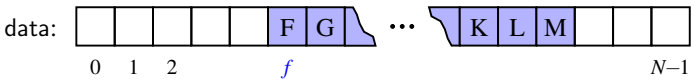
Detalhes de implementação:

- Inserção simples no 'final' do vetor.
- Remoção deve acontecer no início.
  1. Remover posição 0 e fazer shift. ← custoso: $O(n)$
  2. Utilizar uma variável como referência para a frente da fila. ← $O(1)$

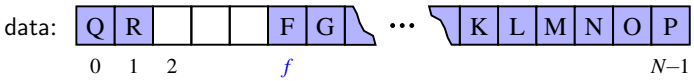
Ideia geral:



Remoção utilizando variável f:



Novo desafio: deve-se controlar espaços vagos no início do vetor.



Solução: implementar uma estratégia circular, cujo incremento na última posição leva à primeira.

Implementação baseada em vetores:

```
1  public class ArrayQueue<E> implements Queue<E> {
2     public static final int CAPACITY = 1000;
3     private E[] data;
4     private int f = 0;
5     private int sz = 0;
6
7     public ArrayQueue() {this(CAPACITY);}
8
9     public ArrayQueue(int capacity) {
10       data = (E[]) new Object[capacity];
11    }
12
13    public int size() { return sz; }
14    public boolean isEmpty() { return (sz == 0); }
15
16    public void enqueue(E e) {
17       if (sz == data.length)
18          throw new IllegalStateException("Queue is full");
19
20       int avail = f + sz;
21       if(avail >= data.length) avail -= data.length;
22
23       data[avail] = e;
24       sz++;
25    }
26
27    public E first() {
28       if (isEmpty()) return null;
29       return data[f];
30    }
31
32    public E dequeue() {
33       if (isEmpty()) return null;
```

```
34          E answer = data[f];
35          data[f] = null;
36
37          f = (f + 1);
38          if(f >= data.length) f = 0;
39
40          sz--;
41          return answer;
42      }
43  }
```

- Análise de eficiência:
  - `size`: $O(1)$.
  - `isEmpty`: $O(1)$.
  - `first`: $O(1)$.
  - `enqueue`: $O(1)$.
  - `dequeue`: $O(1)$.

Implementação baseada em listas simplesmente encadeadas:

```
1  public class LinkedQueue<E> implements Queue<E> {
2      private SinglyLinkedList<E> list = new SinglyLinkedList<>();
3      public LinkedQueue() { }
4      public int size() { return list.size(); }
5      public boolean isEmpty() { return list.isEmpty(); }
6      public void enqueue(E element) { list.addLast(element); }
7      public E first() { return list.first(); }
8      public E dequeue() { return list.removeFirst(); }
9  }
```

- Comentários:
  - Operações continuam com complexidade constante $O(1)$.

Exercícios:

1. Implemente uma fila baseada em vetor para armazenar os pacientes de um plantão de atendimento. Sempre que o paciente chega, um atendente faz seu cadastro, informando nome, idade e código do plano de saúde. O médico de plantão, que possui acesso ao mesmo sistema, efetua a chamada do próximo paciente.
2. Modifique a implementação do exercício anterior, utilizando uma fila baseada em uma lista simplesmente encadeada.

Deques

Ideia geral:

- Deque é o termo usado para *double-ended queue*.
- Fila com inserção e remoção dos dois lados (início e fim).
- Operações:
  - `addFirst`/`addLast`: insere elemento no início/final.
  - `removeFirst`/`removeLast`: remove o primeiro/último elemento.
- Exemplo de funcionamento:

Method	Return Value	D
addLast(5)	–	(5)
addFirst(3)	–	(3, 5)
addFirst(7)	–	(7, 3, 5)
first( )	7	(7, 3, 5)
removeLast( )	5	(7, 3)
size( )	2	(7, 3)
removeLast( )	3	(7)
removeFirst( )	7	( )
addFirst(6)	–	(6)
last( )	6	(6)
addFirst(8)	–	(8, 6)
isEmpty( )	false	(8, 6)
last( )	6	(8, 6)

Interface Deque:

```
1  public interface Deque<E> {
2     int size();
3     boolean isEmpty();
4     E first();
5     E last();
6     void addFirst(E e);
7     void addLast(E e);
8     E removeFirst();
9     E removeLast();
10 }
```

Implementação baseada em vetores:

```
1  public class ArrayDeque<E> implements Deque<E> {
2     public static final int CAPACITY = 1000;
3     private E[] data;
4     private int f = 0;
5     private int sz = 0;
6
7     public ArrayDeque() {this(CAPACITY);}
8
9     public ArrayDeque(int capacity) {
10       data = (E[]) new Object[capacity];
11    }
12
13    public int size() { return sz; }
14    public boolean isEmpty() { return (sz == 0); }
15
16    public E first() {
17       if (isEmpty()) return null;
18       return data[f];
19    }
20
21    public E last() {
22       if (isEmpty()) return null;
23       int index = f + sz - 1;
```

```
24          if(index >= data.length) index -= data.length;
25          return data[index];
26      }
27
28      public void addFirst(E e) {
29          if (sz == data.length)
30              throw new IllegalStateException("Queue is full");
31
32          int avail = f - 1;
33          if(avail < 0) avail = data.length - 1;
34
35          data[avail] = e;
36          f = avail;
37          sz++;
38      }
39
40      public void addLast(E e) {
41          if (sz == data.length)
42              throw new IllegalStateException("Queue is full");
43
44          int avail = f + sz;
45          if(avail >= data.length) avail -= data.length;
46
47          data[avail] = e;
48          sz++;
49      }
50
51      public E removeFirst() {
52          if (isEmpty()) return null;
53          E answer = data[f];
54          data[f] = null;
55
56          f = f + 1;
57          if(f >= data.length) f = 0;
58
59          sz--;
60          return answer;
61      }
```

```
62
63      public E removeLast() {
64          if (isEmpty()) return null;
65          int index = f + sz - 1;
66          if(index >= data.length) index -= data.length;
67
68          E answer = data[index];
69          data[index] = null;
70
71          sz--;
72          return answer;
73      }
74  }
```

- Comentários:
  - Todas as operações são realizadas em tempo constante $O(1)$.
  - Operador módulo controla a lista circular.

Implementação baseada em listas duplamente encadeadas:

```
1   public class LinkedDeque<E> implements Deque<E> {
2       private DoublyLinkedList<E> list = new DoublyLinkedList<>();
3       public int size() { return list.size(); }
4       public boolean isEmpty() { return list.isEmpty(); }
5       public E first() { return list.first(); }
6       public E last() { return list.last(); }
7       public void addFirst(E e) { list.addFirst(e); }
8       public void addLast(E e) { list.addLast(e); }
9       public E removeFirst() { return list.removeFirst(); }
10      public E removeLast() { return list.removeLast(); }
11  }
```

- Comentários:
  - Operações continuam com complexidade constante $O(1)$.

Exercícios:

1. Deseja-se armazenar um conjunto de livros para doação. Cada livro doado é inserido no catálogo, e cada pessoa interessada recebe um livro. Implemente uma estrutura de deque, onde cada livro recebido é armazenado em um lado aleatório da lista (início ou fim), e cada pessoa interessada escolhe um dos dois livros dos terminais da lista, o qual é retirado da mesma. Utilize uma implementação de deque baseada em vetores.
2. Modifique o exercício anterior e substitua a implementação do deque por uma baseada em listas duplamente encadeadas.

Atividades

1. Resolva os seguintes exercícios de Goodrich et al. [2014]:

R-6.1: Suponha que inicialmente uma pilha vazia `S` tenha realizado um total de 25 operações `push`, 12 operações `top` e 10 operações `pop`, 3 das quais retornaram `null`, indicando uma pilha vazia. Qual é o tamanho atual de `S`?

R-6.2: Sendo a pilha do exercício anterior uma instância da classe `ArrayStack`, qual seria o valor final da variável `t`?

R-6.3: Quais valores são retornados durante as seguintes operações, se executado inicialmente em uma pilha vazia? `push(5)`, `push(3)`, `pop()`, `push(2)`, `push(8)`, `pop()`, `pop()`, `push(9)`, `push(1)`, `pop()`, `push(7)`, `push(6)`, `pop()`, `pop()`, `push(4)`, `pop()`, `pop()`.

R-6.4: Implemente um método com a assinatura `transfer(S, T)` que transfere todos os elementos da pilha `S` para a pilha `T`, de modo que o elemento que iniciou no topo de `S` é o primeiro elemento a ser inserido em T, e o último elemento de S termina no topo de T.

R-6.5: Apresente um método recursivo que remove todos os elementos de uma pilha.

R-6.7: Suponha que uma fila vazia `Q` realizou um total de 32 operações de `enqueue`, 10 operações *first* e 15 operações `dequeue`, 5 das quais retornaram `null`, indicando uma fila vazia. Qual é o tamanho atual de `Q`?

R-6.8: Sendo a fila do exercício anterior uma instância da classe `ArrayQueue` com uma capacidade de 30 elementos nunca excedida, qual seria o valor final da variável `f`?

R-6.9: Quais são os valores retornados após as seguintes operações, se executadas em uma fila inicialmente vazia? `enqueue(5)`, `enqueue(3)`, `dequeue()`, `enqueue(2)`, `enqueue(8)`, `dequeue()`, `dequeue()`, `enqueue(9)`, `enqueue(1)`, `dequeue()`, `enqueue(7)`, `enqueue(6)`, `dequeue()`, `dequeue()`, `enqueue(4)`, `dequeue()`, `dequeue()`.

R-6.10: Faça um adaptador simples (classe) que implemente uma pilha usando uma instância de deque para armazenamento.

R-6.11: Faça um adaptador simples (classe) que implemente uma fila usando uma instância de deque para armazenamento.

R-6.12: Quais são os valores retornados após as seguintes operações, se executadas em um deque inicialmente vazio? `addFirst(3)`, `addLast(8)`, `addLast(9)`, `addFirst(1)`, `last()`, `isEmpty()`, `addFirst(2)`, `removeLast()`, `addLast(7)`, `first()`, `last()`, `addLast(4)`, `size()`, `removeFirst()`, `removeFirst()`.

R-6.13: Suponha que você tenha um deque `D` contendo os números (1, 2, 3, 4, 5, 6, 7, 8), nessa ordem. Supondo que além disso você tenha uma fila `Q` incialmente vazia. Apresente um pseudocódigo que utilize somente `D` e `Q` (mais nenhuma variável) e resulte em `D` armazenando os elementos na ordem (1, 2, 3, 5, 4, 6, 7, 8).

R-6.14: Repita o exercício anterior usando o deque `D` e uma pilha `S`, inicialmente vazia.

Referências

Goodrich, M. T., Tamassia, R., and Goldwasser, M. H. (2014). *Data structures and algorithms in Java*. John Wiley & Sons, 6th edition.

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