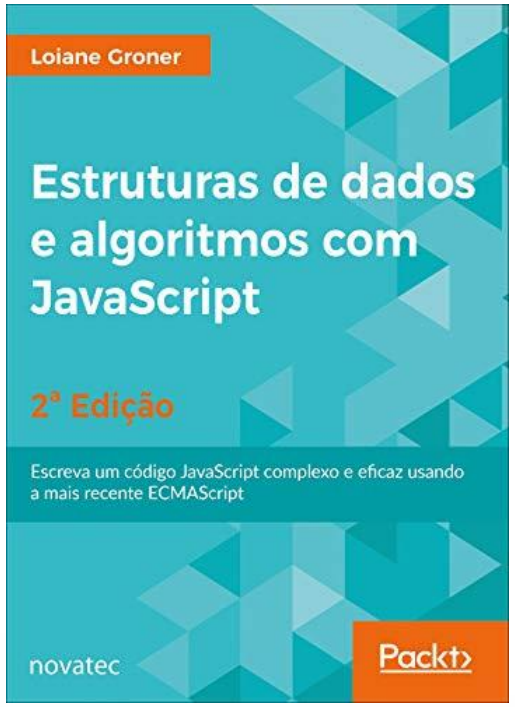
Filas (queues) e Deques

Referências Bibliográficas:

GRONER, Loiane. **Estruturas de dados e algoritmos com JavaScript**: Escreva um código JavaScript complexo e eficaz usando a mais recente ECMAScript. 2ª ed. São Paulo: Novatec, 2019.

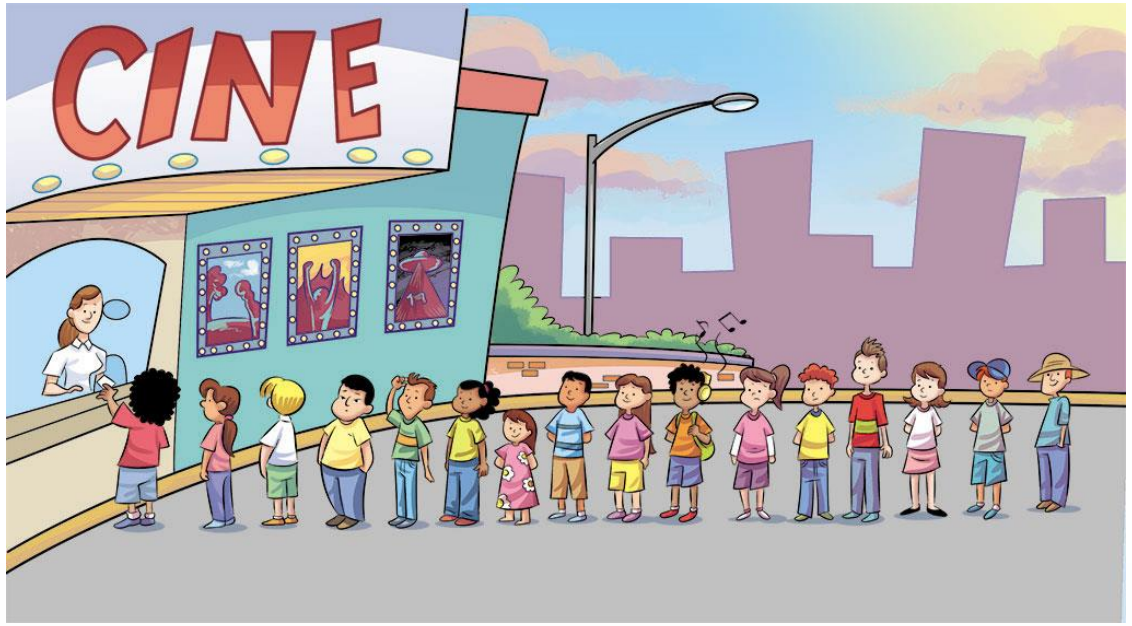


SIF005 - Estrutura de Dados

Prof. Dr. Anderson Sena – anderson.sena@iesb.edu.br

Os seguintes tópicos serão abordados:

- ✓ a estrutura de dados de fila (queue);
- ✓ a estrutura de dados de deque (fila de duas pontas);
- ✓ adição de elementos em uma fila e em um deque;
- ✓ remoção de elementos de uma fila e de um deque;
- ✓ simulação de filas circulares com o jogo de Batata Quente;
- ✓ verificação se uma frase é um palíndromo com um deque.

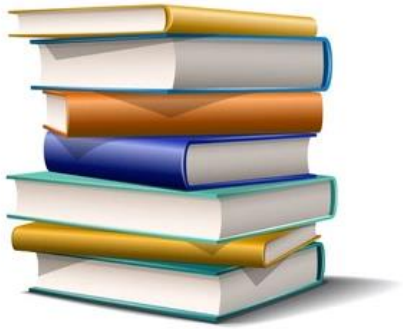


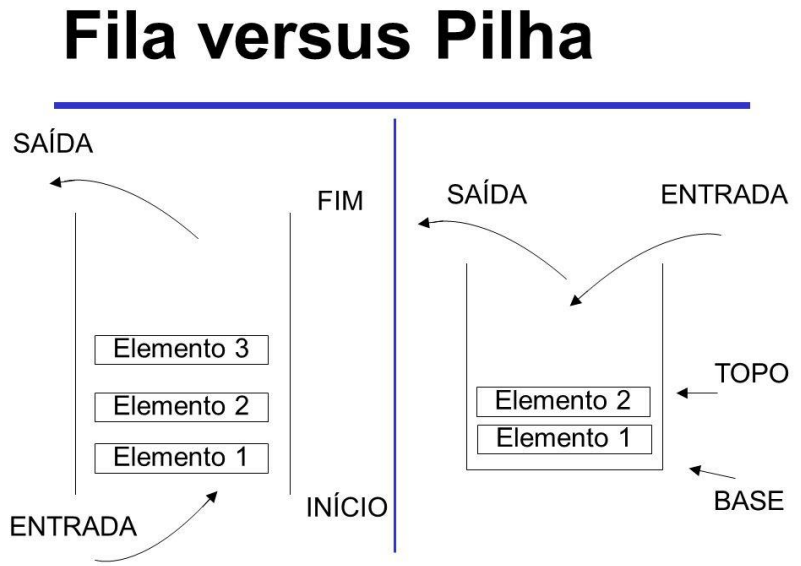
Relembrando conceito da Aula 01

- **Fila e pilha** são estruturas abstratas lineares. A escolha entre as duas está relacionada à ordem de entrada e saída dos elementos:

  ✓ **Fila** é para qualquer situação FIFO - first in, first out (primeiro que entra é o primeiro que sai).
  exemplos: playlist de músicas, pedidos de uma loja, documentos para impressão.

  ✓ **Pilha** é para situações de LIFO - last in, first out (último que entra é o primeiro que sai).
  exemplo: feed de notícias, função de fazer e refazer nos editores de texto.





Fonte: Repositório GitHub Fred Gomes <https://github.com/freddgomes/Estruturas-de-Dados-e-Algoritmos>, acesso em 31/01/2021

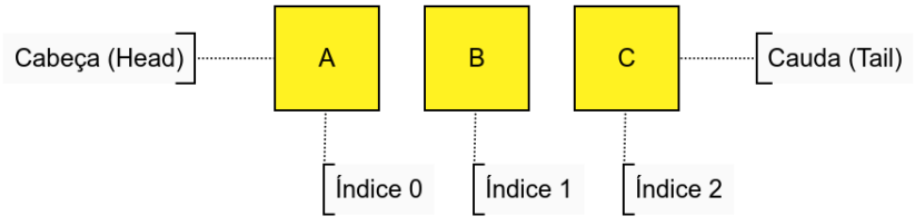
Estrutura de dados de fila

- Uma **fila** é uma coleção de itens baseada em **FIFO** (*First In First Out*, *isto é, o primeiro que entre é o primeiro que sai*), também conhecido como princípio do **first-come first-serve** (*o primeiro a chegar é o primeiro a ser servido*).

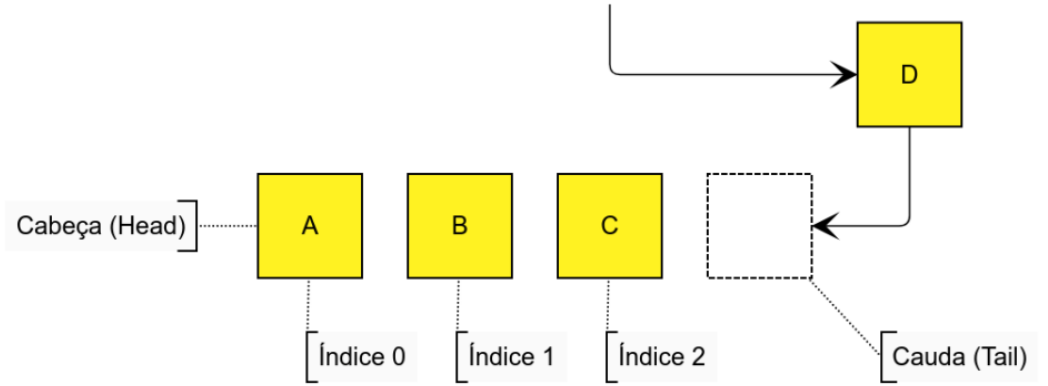
- A adição de novos elementos em uma fila é feita na **cauda** (*tail*) e a remoção, na **frente** (*head*).

- O elemento mais recente adicionado na fila deve esperar no final dela.

Uma fila com elementos A, B e C

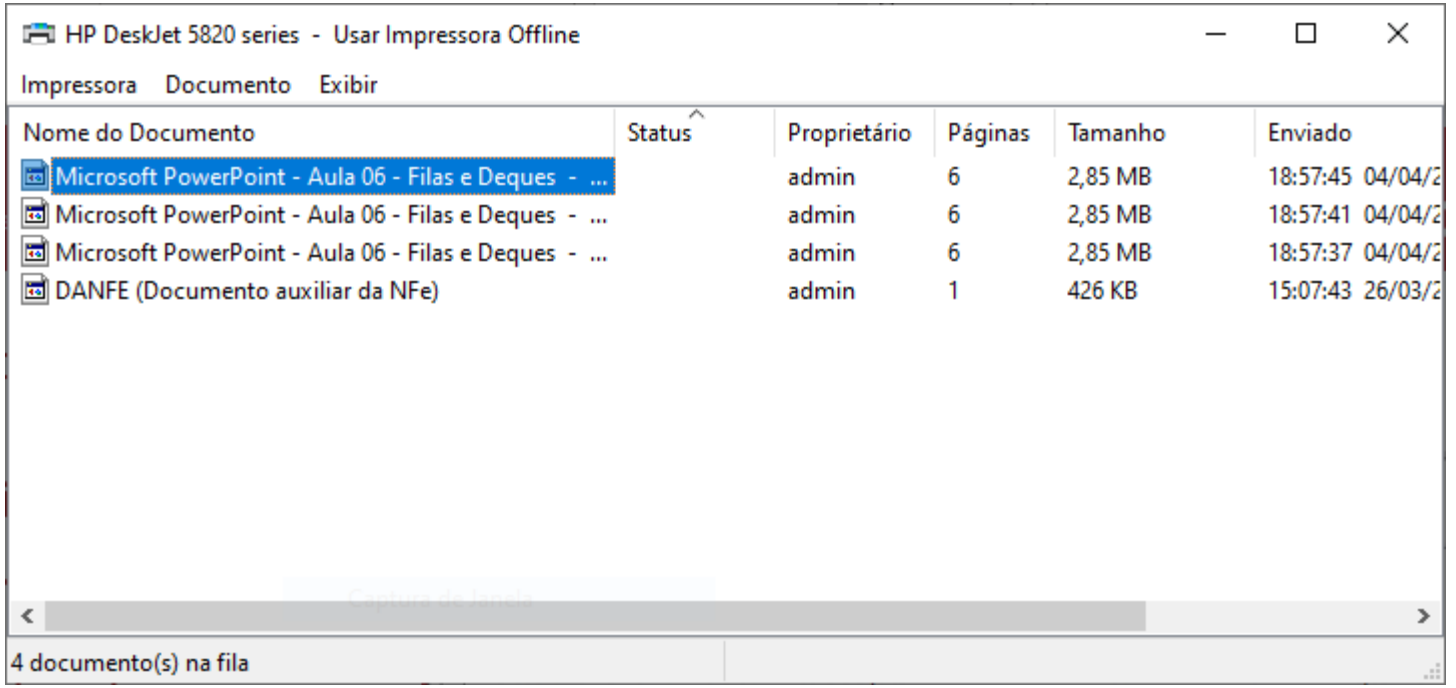


Adição do elemento D na mesma fila



Exemplo de fila

- ✓ Um exemplo muito conhecido em ciência da computação é a fila de impressão.
- ✓ Suponha que precisamos imprimir cinco documentos.
- ✓ Abrimos cada um dos documentos e clicamos no ícone para imprimir.
- ✓ Cada um será enviado para a fila da impressora.



- ✓ O primeiro documento para o qual solicitamos a impressão será impresso antes, e assim por diante, até que todos tenham sido impressos.

Criando a classe Queue

- ✓ Criaremos nossa própria classe Queue para representar uma fila.
- ✓ Usaremos, ao invés de array, um objeto **items = { }** para armazenar e acessar de forma mais eficiente nossos elementos.
- ✓ Você notará que as classes **Queue** e **Stack** (*pilha*) são muito parecidas: *somente os princípios de adição e remoção de elementos são diferentes.*

*Para controlar o tamanho da fila declaramos a propriedade **count**.*

*Como iremos remover elementos da **frente** (head) da **fila**, declaramos a propriedade **lowestCount** para manter o controle do primeiro elemento.*

```
1   // criaremos nossa própria classe para representar uma fila
2   class Queue {
3       constructor(){
4           // propriedade count para controlar o tamanho da fila
5           this.count = 0;
6           // como removeremos  da frente da fila, a propriedade
7           // lowestCount para manter o controle(índice) do primeiro elemento
8           this.lowestCount = 0;
9           // usaremos um objeto para armazenar elementos na fila
10          this.items = {};
11      }
12  }
13
```

Métodos para a classe Queue

Em seguida, devemos declarar os métodos disponíveis em uma fila:

- **enqueue(elemento)**: *esse método adiciona um novo elemento no final da fila.*
- ***dequeue( )***: *esse método remove o primeiro elemento da fila (o item que está na frente). Também devolve o elemento removido.*
- **peek( )**: *esse método devolve o primeiro elemento da fila como informação – é o primeiro item adicionado e o primeiro que será removido da fila. Funciona igualmente como o método* ***front****, como é conhecido em outras linguagens.*
- **isEmpty( )**: *esse método devolve true se a fila não contiver nenhum elemento, e false se o tamanho for maior que 0.*
- **size( )**: *esse método devolve o número de elementos contidos na fila. É semelhante à propriedade* ***length*** *do array.*
- **toString( )**: *para imprimir o conteúdo da fila.*

Métodos para a classe Queue

Em seguida, devemos declarar os métodos disponíveis em uma fila:

- **enqueue(elemento)**: *esse método adiciona um novo elemento no final da fila.*
- ***dequeue( )***: *esse método remove o primeiro elemento da fila (o item que está na frente). Também devolve o elemento removido.*
- **peek( )**: *esse método devolve o primeiro elemento da fila como informação – é o primeiro item adicionado e o primeiro que será removido da fila. Funciona igualmente como o método* ***front****, como é conhecido em outras linguagens.*
- **isEmpty( )**: *esse método devolve true se a fila não contiver nenhum elemento, e false se o tamanho for maior que 0.*
- **size( )**: *esse método devolve o número de elementos contidos na fila. É semelhante à propriedade* ***length*** *do array.*
- **toString( )**: *para imprimir o conteúdo da fila.*

```
1  // criaremos nossa própria classe para representar uma fila
2  class Queue {
3      constructor() {
4        // constrói o objeto com valores a fila
5        this.items = {};
6      }
7      enqueue(element) {
8        // incluir um elemento na fila
9      }
10     size() {
11       // retorna o tamanho da fila
12     }
13     isEmpty() {
14        // retorna true se a fila estiver vazia
15     }
16     dequeue() {
17       // remove o elemento da frente da fila
18     }
19     peek() {
20        // mostra o elemento da frente da fila
21     }
22     clear() {
23         // para limpar a fila
24     }
25     toString() {
26         //para imprimir a fila
27     }
28   }
```

Inserção de elementos na fila

- ✓ O primeiro método que implementaremos é o método **enqueue** (enfileirar).
- ✓ Esse método será responsável pela adição de novos elementos na fila.
- ✓ Importante: *só podemos adicionar novos itens no **final** da fila*.

O método **enqueue** tem a mesma implementação do método **push( )** da classe **Stack**.

Como a propriedade **items** será um **objeto** JavaScript, ela é uma coleção de pares chave e valor.

```
1   // criaremos nossa própria classe para representar uma fila
2   class Queue {
3       constructor(){
4           // propriedade count para controlar o tamanho da fila
5           this.count = 0;
6           // como removeremos  da frente da fila, a propriedade
7           // lowestCount para manter o controle do primeiro elemento
8           this.lowestCount = 0;
9           // usaremos um objeto para armazenar elementos na fila
10          this.items = {};
11      }
12
13      enqueue(element){
14          this.items[this.count] = element;
15          this.count++
16      }
17  }
18
```


Loiane Groner, 2019, pág. 130

Verificando se a Fila está vazia e seu tamanho

- Para calcular quantos elementos há na fila, basta calcular a diferença entre as chaves **count** e **lowestCount**.

✓ Suponha que a propriedade **count** tenha valor 2 e **lowestCount** seja igual a 0.

✓ Isso significa que temos dois elementos na fila.

✓ Em seguida, removemos um elemento dela.

✓ A propriedade **lowestCount** será atualizada com o valor 1 e **count** continuará com valor igual a 2.

✓ Agora a fila terá somente um elemento, e assim por diante.

✓ Para implementar o método isEmpty, é só verificar se o retorno do método **size( )** é igual a zero.

```
// basta retornar a diferença de count por lowestCount
size() {
  return this.count - this.lowestCount;
}
// devolverá true se a pilha estiver vazia.
isEmpty() {
  return this.size() === 0;
}
```


Loiane Groner, 2019, pág. 132

Remoção de elementos da fila

- Implementaremos o método **dequeue** (*desenfileirar*), responsável pela remoção de itens da fila.
- Como a fila utiliza o princípio **FIFO**, o **primeiro** item adicionado na fila será o item a ser **removido**.

✓ Inicialmente devemos verificar se a fila está vazia;

✓ Se a fila não estiver vazia, armazenaremos o valor da frente da fila para que possamos devolvê-lo depois que o elemento tiver sido removido.

✓ Também precisamos incrementar a propriedade **lowestCount** de **1**.

```
dequeue() {
  // verifica se a fila está vazia
  if (this.isEmpty()) {
    return undefined;
  }
  // armazenando o valor da frente da fila
  const result = this.items[this.lowestCount];
  // removendo o elemento da frente
  delete this.items[this.lowestCount];
  // será necessário incrementar a propriedade lowestCount
  this.lowestCount++;
  return result;
}
```

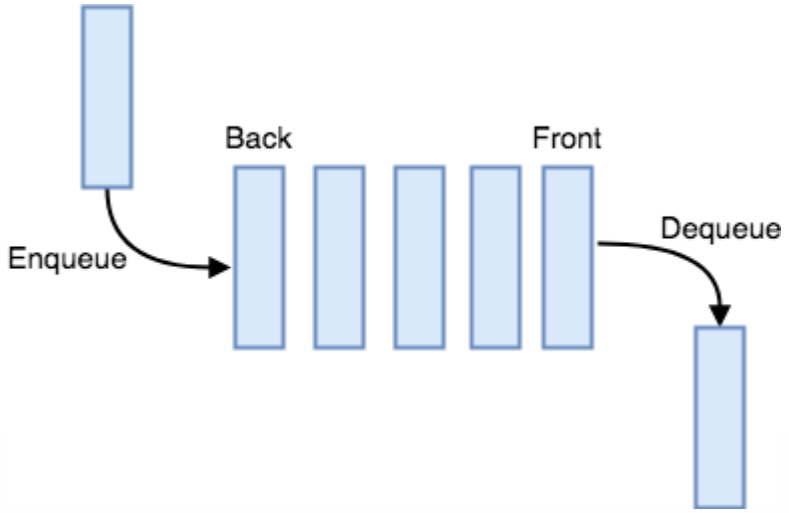

Loiane Groner, 2019, pág. 130

Dando uma espiada no elemento da frente da fila

- ✓ Se quisermos saber qual é o elemento que está na frente da nossa fila, podemos usar o método **peek**.

- ✓ Este método devolverá o item que está na frente da fila (usando **lowestCount** como chave para obter o valor do elemento):

```
// este método devolverá o item que está
// na frente da fila usando lowestCount
// como chave para obter o valor do elemento
peek() {
  if (this.isEmpty()) {
    return undefined;
  }
  return this.items[this.lowestCount];
}
```



Limpando a fila

Para limpar todos os elementos da fila:

- ✓ podemos chamar o método **dequeue** até que ele devolva **undefined**, ou
- ✓ podemos simplesmente reiniciar o valor das propriedades da classe **Queue** com os mesmos valores declarados em seu construtor

```
// para limpar a fila, basta reinicializar
// suas propriedades do método construtor
clear() {
    this.items = {};
    this.count = 0;
    this.lowestCount = 0;
}
```

Criando o método toString

- Nossa classe **Queue** está implementada, assim como fizemos na classe **Stack**.
- Mas também podemos acrescentar o método **toString( )** para apresentar todos os elementos da fila.
- Diferente da classe **Stack** (*pilha*), e como o primeiro índice da classe **Queue** pode não ser zero, começamos iterando a partir do índice **lowestCount**.

```
/**
 * Na classe Stack, começamos a iterar pelos valores dos
 * itens a partir do índice zero.
 * Como o primeiro índice da classe Queue pode não ser zero,
 * começamos iterando a partir do índice lowestCount.
 */
toString() {
  if (this.isEmpty()) {
    return '';
  }
  let objString = `${this.items[this.lowestCount]}`;
  for (let i = this.lowestCount + 1; i < this.count; i++) {
    objString = `${objString}, ${this.items[i]}`;
  }
  return objString;
}
```

As classes Queue e Stack são muito parecidas. A única diferença está nos métodos **dequeue** e **peek**, que se deve à distinção entre os princípios FIFO e LIFO.

Usando a classe Queue

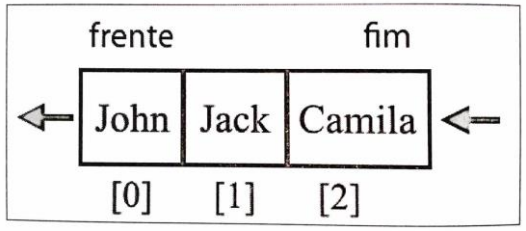


Figura 5.2

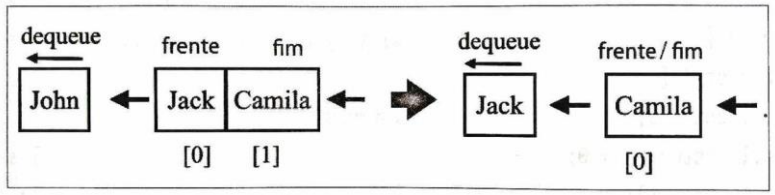


Figura 5.3

```
77  const fila = new Queue();
78  console.log(fila.isEmpty()); // exibe true
79  //adicionando duas pessoas na fila
80  fila.enqueue('Jhon');
81  fila.enqueue('Jack');
82  console.log('A fila possui inicialmente as pessoas: ' + fila.toString()); // Jhon, Jack
83  //vamos acrescentar outra pessoa na fila
84  fila.enqueue('Camila');
85  console.log('A fila agora possui as pessoas: ' + fila.toString()); // Jhon, Jack, Camila
86  console.log('O tamanho da fila é: ' + fila.size()); // exibe 3
87  console.log('Removendo a pessoa da frente da fila: ' + fila.dequeue()); // Jhon
88  console.log('A fila agora possui os elementos: ' + fila.toString()); // Jack, Camila
89  console.log('A pessoa da frente da fila agora é: ' + fila.peek()); // Jack
90  console.log('Retirando o próximo da fila: ' + fila.dequeue()); // Jack
91  console.log('A fila agora possui apenas: ' + fila.toString()); // Camila
92  console.log('A pessoa da frente da fila agora é: ' + fila.peek()); // Camila
93
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
true
A fila possui inicialmente as pessoas: Jhon, Jack
A fila agora possui as pessoas: Jhon, Jack, Camila
O tamanho da fila é: 3
Removendo a pessoa da frente da fila: Jhon
A fila agora possui os elementos: Jack, Camila
A pessoa da frente da fila agora é: Jack
Retirando o próximo da fila: Jack
A fila agora possui apenas: Camila
A pessoa da frente da fila agora é: Camila
```

Estrutura de dados de deque

- A estrutura de dados **deque**, também conhecida como **fila de duas pontas** (*double-ended queue*), é uma fila especial que nos permite inserir e remover elementos do final ou da frente da fila:

  - ✓ Um exemplo de um **deque** na vida real é a **fila típica** em cinemas, lanchonetes e assim por diante.
  - ✓ Por exemplo: *uma pessoa que acabou de comprar um ingresso poderia retornar para a frente da **fila** somente para pedir uma informação rápida*.
  - ✓ Se a pessoa que estiver no final da fila estiver com **pressa**, ela poderia também **sair da fila**.

Estrutura de dados de deque

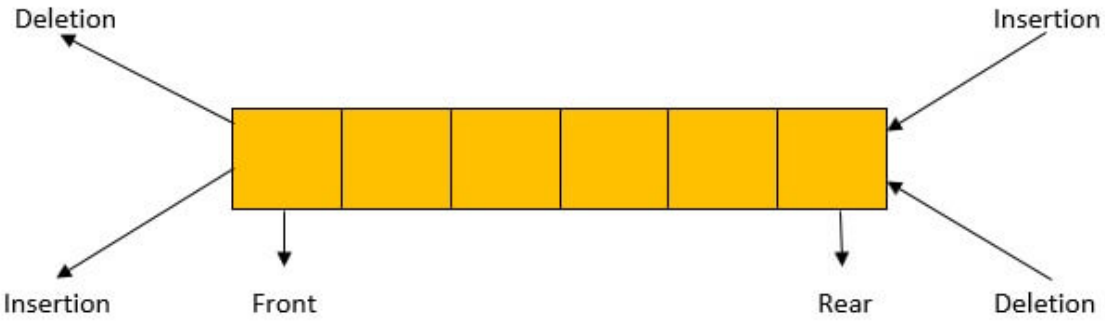
- Em ciência da computação, uma aplicação comum de um deque é na armazenagem de uma lista de operações para desfazer ações (undo ou ctrl+z).

  a. Sempre que um usuário executar uma operação no software, um **push( )** dessa operação será feito no **deque** (*exatamente como uma pilha*).

  b. Quando o usuário clicar no botão Undo (*Desfazer*), uma operação **pop( )** será efetuada no **deque**, o que significa que essa operação será removida do final.

  c. Depois de um número predefinido de operações, as operações mais antigas serão removidas da frente do **deque**.

  d. Como o **deque** implementa os princípios tanto de **FIFO** quanto de **LIFO**, podemos dizer também que o **deque** combina as estruturas de dados de **fila** e **pilha**.



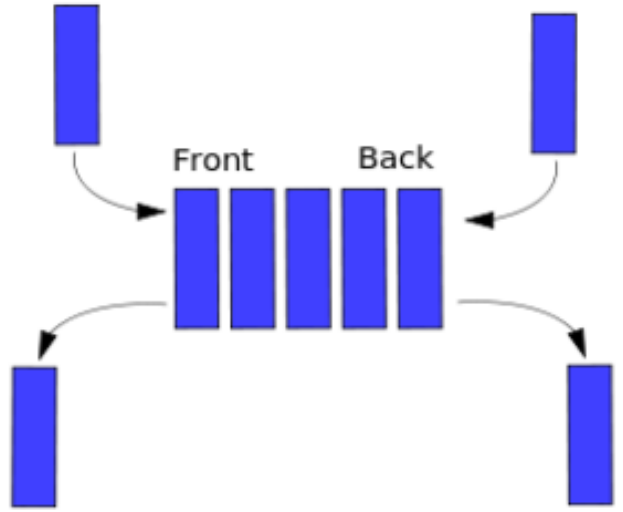
Loiane Groner, 2019, pág. 134

Criando a classe Deque

- Começaremos declarando a classe Deque e o seu construtor:

```
1   // declarando a classe Deque e seu construtor
2   class Deque {
3     construtor() {
4       // propriedade count para controlar o tamanho do deque
5       this.count = 0;
6       // lowestCount para manter o controle(índice) do primeiro elemento
7       this.lowestCount = 0;
8       // objeto items para armazenar elementos no deque
9       this.items = {};
10    }
11  }
12
```

- Como o deque é uma fila especial, percebemos que ele compartilha alguns trechos de código com o construtor, tem as mesmas propriedades internas e terá também os métodos: **isEmpty**, **clear**, **size** e **toString**.



Criando a classe Deque

Pelo fato de que o deque permitir inserir e remover elementos das duas extremidades, teremos também os métodos a seguir:

- **addFront(elemento)**: esse método adiciona um novo elemento na frente do deque.
- **addBack(elemento)**: esse método adiciona um novo elemento no fim do deque
  (*a mesma implementação do método **enqueue** da classe **Queue***).
- **removeFront( )**: esse método remove o primeiro elemento do deque
  (*a mesma implementação do método **dequeue** da classe **Queue***).
- **removeBack( )**: esse método remove o último elemento do deque
  (*a mesma implementação do método **pop** da classe **Stack***).
- **peekFront( )**: esse método devolve o primeiro elemento do deque
  (*a mesma implementação do método **peek** da classe **Queue***).
- **peekBack( )**: esse método devolve o último elemento do deque
  (*a mesma implementação do método **peek** da classe **Stack***).

Loiane Groner, 2019, pág. 136

Criando a classe Deque

Pelo fato de que o deque permitir inse
teremos também os métodos a seguir

- **addFront(elemento)**: esse método a
- **addBack(elemento)**: esse método ad
  (*a mesma implementação do métod*
- **removeFront( )**: esse método remov
  (*a mesma implementação do métod*
- **removeBack( )**: esse método remove
  (*a mesma implementação do métod*
- **peekFront( )**: esse método devolve o
  (*a mesma implementação do métod*
- **peekBack( )**: esse método devolve o
  (*a mesma implementação do métod*

```
1  // declarando a classe Deque e seu construtor
2  class Deque {
3      constructor() {
4        this.count = 0;
5        this.lowestCount = 0;
6        this.items = {};
7      }
8      addFront(element) { // adiciona um novo elemento na frente do deque
9      }
10     addBack(element) { //adiciona um novo elemento no fim do deque
11     }
12     removeFront() { // remove o primeiro elemento do deque
13     }
14     removeBack() { // remove o último elemento do deque
15     }
16     peekFront() { //devolve o primeiro elemento do deque
17     }
18     peekBack() { // devolve o último elemento do deque
19     }
20     size() { // para retornar o tamanho  do deque
21     }
22     isEmpty() { // verifica se o deque está vazio
23     }
24     toString() { // apresenta o conteúdo do deque
25     }
26 }
```

Adicionando elementos na frente do deque

Ao adicionar um elemento na frente do deque, há três cenários:

- ✓ O primeiro cenário é aquele em que o **deque** está vazio, e podemos chamar o método **addBack** para adicionar no final do deque, que nesse caso, também será a frente.
- ✓ O segundo cenário é aquele em que um elemento é removido da frente do deque, onde a propriedade **lowestCount** deve ser igual ou maior que 1, bastando decrementar e atribuir o elemento na posição desse objeto.
- ✓ O terceiro cenário é quando **lowestCount** for igual a zero, e devemos mover todos os elementos para a próxima posição, deixando a primeira posição livre.

```
// adiciona um novo elemento na frente do deque
addFront(element) {
  // primeiro cenário verifica se o deque está vazio
  if (this.isEmpty()) {
    // neste caso chamamos o método addBack (no final do deque)
    this.addBack(element);
  } else if (this.lowestCount > 0) {
    // o elemento é removido da frente do deque
    this.lowestCount--;
    this.items[this.lowestCount] = element;
  } else {
    // se lowestCount é igual a zero e para adicionar um novo elemento
    // na primeira posição, devemos mover para próxima posição e deixar
    // o primeiro index livre
    for (var i = this.count; i > 0; i--) {
      this.items[i] = this.items[i - 1];
    }
    this.count++;
    this.lowestCount = 0;
    this.items[0] = element;
  }
}
```

Adicionando e removendo elementos do deque

addBack(elemento): esse método adiciona um novo elemento no fim do deque (*a mesma implementação do método **enqueue** da classe **Queue***).

```
//adiciona um novo elemento no fim do deque
addBack(element) {
  this.items[this.count] = element;
  this.count++;
}
```

```
// remove o primeiro elemento do deque
removeFront() {
  // verifica se a fila está vazia
  if (this.isEmpty()) {
    return undefined;
  }
  // armazenando o valor da frente da fila
  const result = this.items[this.lowestCount];
  // removendo o elemento da frente
  delete this.items[this.lowestCount];
  // será necessário incrementar a propriedade lowestCount
  this.lowestCount++;
  return result;
}
```

removeFront( ): esse método remove o primeiro elemento do deque (*a mesma implementação do método **dequeue** da classe **Queue***).

removeBack( ): esse método remove o último elemento do deque (*a mesma implementação do método **pop** da classe **Stack***).

```
// remove o último elemento do deque
removeBack() {
  if (this.isEmpty()) {
    return undefined;
  }
  this.count--;
  const result = this.items[this.count];
  delete this.items[this.count];
  return result;
}
```

Métodos adicionais da classe Deque

peekFront( ): esse método devolve o primeiro elemento do deque (*a mesma implementação do método **peek** da classe **Queue***).

```
//devolve o primeiro elemento do deque
peekFront() {
  if (this.isEmpty()) {
    return undefined;
  }
  return this.items[this.lowestCount];
}
```

peekBack( ): esse método devolve o último elemento do deque (*a mesma implementação do método **peek** da classe **Stack***).

```
// devolve o último elemento do deque
peekBack() {
  return this.items[this.items.length - 1];
}
```

Métodos adicionais

```
// para retornar o tamanho  do deque
// basta retornar a diferença de count por lowestCount
size() {
  return this.count - this.lowestCount;
}
// verifica se o deque está vazio
isEmpty() {
  return this.size() === 0;
}
// apresenta o conteúdo do deque
toString() {
  if (this.isEmpty()) {
    return '';
  }
  let objString = `${this.items[this.lowestCount]}`;
  for (let i = this.lowestCount + 1; i < this.count; i++) {
    objString = `${objString}, ${this.items[i]}`;
  }
  return objString;
}
```

Usando a classe Deque

- ✓ Depois de instanciar a classe **Deque**, podemos chamar os seus métodos:

```
97   // usando a classe Deque
98   const deque = new Deque();
99   console.log(deque.isEmpty()); // exibe true
100  deque.addBack('João');
101  deque.addBack('Pedro');
102  console.log(deque.toString()); // João, Pedro
103  deque.addBack('Camila'); // João, Pedro, Camila
104  console.log(deque.size()); //exibe 3
105  console.log(deque.isEmpty()); // exibe false
106  deque.removeFront(); // remove João
107  console.log(deque.toString()); // Pedro, Camila
108  deque.removeBack(); // Camila decide sair
109  console.log(deque.toString()); // Pedro
110  deque.addFront('João'); // João retorna para pedir uma informação
111  console.log(deque.toString()); // João, Pedro
112
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\deque.js"
true
João, Pedro
3
false
Pedro, Camila
Pedro
João, Pedro
```

Fila circular – Batata Quente

- ✓ Como as filas são aplicadas com frequência em ciência da computação e em nossas vidas, há algumas versões modificadas em relação à fila padrão que implementamos até agora.
- ✓ Uma das versões modificadas é **a fila circular**.
- ✓ Um exemplo de fila circular é o jogo de Batata Quente (Hot Potato).
- ✓ Neste jogo, as crianças se organizam em círculo e passam a batata quente para o seu vizinho o mais rápido possível.
- ✓ Em determinado ponto do jogo, a batata quente para de ser passada pelo círculo e a criança que tiver a batata quente em mãos deverá sair do círculo.
- ✓ Esta ação será repetida até que reste apenas uma criança vencedora.
- ✓ Vamos criar uma função para simular o jogo:

```
function hotPotato(elementsList, num) {
  // usaremos a classe Queue implementada anteriormente
  const queue = new Queue();
  const elimitatedList = [];
  for (let i = 0; i < elementsList.length; i++) {
    //vamos obter uma lista de nomes e enfileirar
    queue.enqueue(elementsList[i]);
  }
  while (queue.size() > 1) {
    for (let i = 0; i < num; i++) {
      // removemos um item da fila e adicionamos no final
      // para simular a batata quente
      queue.enqueue(queue.dequeue());
    }
    // uma vez que o número foi alcançado,
    // a pessoa que tiver a batata será removida da fila
    elimitatedList.push(queue.dequeue());
  }
  return {
    eliminated: elimitatedList,
    // quando restar apenas uma pessoa, ela será a vencedora
    winner: queue.dequeue(),
  };
}
```

Simulando a função hotPotato

- ✓ Para implementar uma simulação desse jogo, usaremos a classe Queue que implementamos anteriormente.
- ✓ Vamos obter uma lista de nomes e enfileirar todos eles.
- ✓ Dado um número, devemos iterar pela fila.
- ✓ Removemos um item do início da fila e o adicionamos no final para simular a batata quente.
- ✓ Uma vez que o número for alcançado, a pessoa que tiver a batata quente será eliminada (removida da fila).
- ✓ Quando restar apenas uma pessoa, ela será declarada a vencedora.

```
95   const names = ['John', 'Jack', 'Camila', 'Ingrid', 'Carlos'];
96   const result = hotPotato(names, 7);
97   result.eliminated.forEach(name => {
98     console.log(`${name} foi eliminad@ do jogo da Batata Quente.`);
99   });
100  console.log(`O ganhador(a) foi: ${result.winner}`);
101
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
Camila foi eliminad@ do jogo da Batata Quente.
Jack foi eliminad@ do jogo da Batata Quente.
Carlos foi eliminad@ do jogo da Batata Quente.
Ingrid foi eliminad@ do jogo da Batata Quente.
O ganhador(a) foi: John
```

Simulando a função hotPotato

O diagrama (Figura 5.4) simula o resultado da função hotPotato.

Você pode alterar o número passado para a função **hotPotato** a fim de simular cenários diferentes.

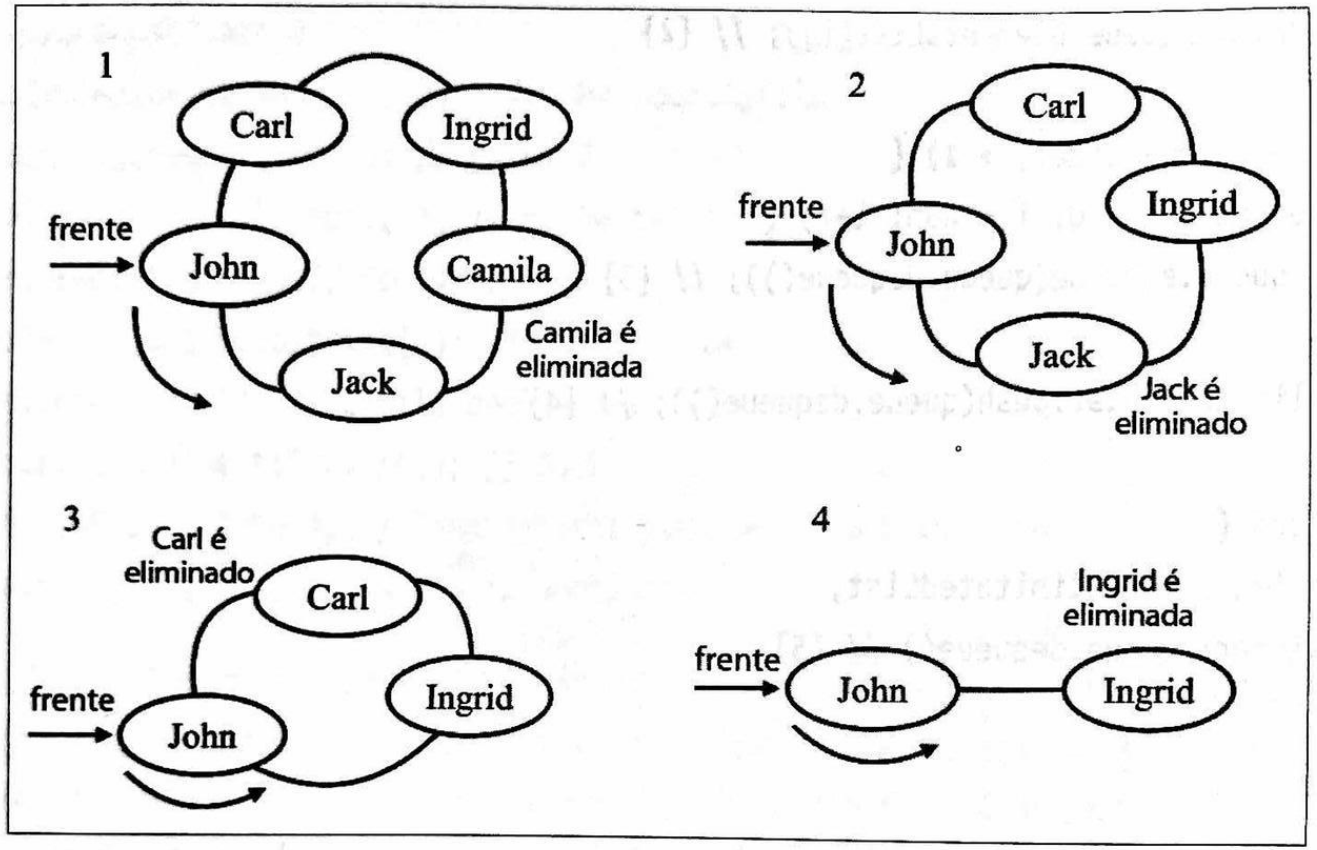


Figura 5.4

Loiane Groner, 2019, pág. 140

Verificador de palíndromo

- A seguir, apresentamos a definição e **PALÍNDROMO** de acordo com a Wikipedia:

Um palíndromo é uma palavra, frase, número ou outra sequência de caracteres que é lido igualmente de trás para frente ou de frente para trás, por exemplo, **madam** *ou* **racecar**.

- Há diferentes algoritmos que podem ser usados para verificar se uma frase ou uma string é um **palíndromo**.
- O modo mais fácil é **inverter** a string e compará-la com a string original;
- Se as duas strings forem iguais, teremos um palíndromo;
- Também podemos usar uma **pilha** para fazer isso;
- Mas a maneira mais fácil de resolver esse problema com uma estrutura de dados é usando um **deque**.

Fonte: <https://pt.wikipedia.org/wiki/Pal%C3%Adndromo>, acessado em 15abril2021

Verificador de palíndromo

- ✓ Primeiro devemos verificar se a **string** é válida;
- ✓ Depois instanciamos nossa classe **Deque** criada anteriormente;
- ✓ Agora convertemos todos os caracteres para minúsculo com a função **toLocaleLowerCase( )** e retiramos os espaços com a função **split( )** e **join( )**;
- ✓ Em seguida, inserimos cada um dos caracteres no **deque**;
- ✓ Agora, enquanto tiver caracteres, removemos o **primeiro** e o **último** caractere do deque;
- ✓ For fim, comparamos esses caracteres, pois para ser um **palíndromo**, os dois caracteres removidos do deque devem ser iguais.

```
// este algoritmo utiliza um deque para solucionar o palíndromo
function palindromeChecker(asString) {
  // verificando se a string é valida
  if ( asString === undefined ||
    asString === null ||
    (asString !== null && asString.length === 0)) {
    return false;
  }
  const deque = new Deque(); // usando a classe Deque já implementada
  // convertendo para minúsculo e retirando os espaços
  const lowerString = asString.toLocaleLowerCase().split(' ').join('');
  let isEqual = true;
  let firstChar, lastChar;
  for (let i = 0; i < lowerString.length; i++) {
    // inserindo cada um dos caracteres no deque
    deque.addBack(lowerString.charAt(i));
  }
  while (deque.size() > 1 && isEqual) {
    // enquanto tiver elementos, removemos do início e do fim do deque
    firstChar = deque.removeFront();
    lastChar = deque.removeBack();
    // para ser palíndromo os dois caracteres removidos devem ser iguais
    if (firstChar !== lastChar) {
      isEqual = false; // {8}
    }
  }
  return isEqual;
}
```

Verificador de palíndromo

```
// este algoritmo utiliza um deque para solucionar o palíndromo
function palindromeChecker(asString) {
  // verificando se a string é valida
  if ( asString === undefined ||
    asString === null ||
    (asString !== null && asString.length === 0)) {
    return false;
  }
  const deque = new Deque(); // usando a classe Deque já implementada
  // convertendo para minúsculo e retirando os espaços
  const lowerString = asString.toLocaleLowerCase().split(' ').join('');
  let isEqual = true;
  let firstChar, lastChar;
  for (let i = 0; i < lowerString.length; i++) {
    // inserindo cada um dos caracteres no deque
    deque.addBack(lowerString.charAt(i));
  }
  while (deque.size() > 1 && isEqual) {
    // enquanto tiver elementos, removemos do início e do fim do deque
    firstChar = deque.removeFront();
    lastChar = deque.removeBack();
    // para ser palíndromo os dois caracteres removidos devem ser iguais
    if (firstChar !== lastChar) {
      isEqual = false; // {8}
    }
  }
  return isEqual;
}
```

```
142  console.log('a', palindromeChecker('a'));
143  console.log('aa', palindromeChecker('aa'));
144  console.log('kayak', palindromeChecker('kayak'));
145  console.log('level', palindromeChecker('level'));
146  console.log('Was it a car or a cat I saw',
147    palindromeChecker('Was it a car or a cat I saw'));
148  console.log('Step on no pets',
149          palindromeChecker('Step on no pets'));
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\deque.js"
a true
aa true
kayak true
level true
Was it a car or a cat I saw true
Step on no pets true
```

Resumo

- ✓ Nesta aula, conhecemos a estrutura de dados **fila** (**queue**).
- ✓ Implementamos o nosso próprio algoritmo para representar uma **fila** e vimos como adicionar e remover elementos dela usando os métodos **enqueue** e **dequeue**, de acordo com o princípio de **FIFO** (First In First Out).
- ✓ Também conhecemos a estrutura de dados de **deque**, aprendemos a adicionar elementos na frente e no final do deque e a remover elementos da frente ou do final dessa estrutura.
- ✓ Além disso, discutimos como resolver dois problemas famosos usando as estruturas de dados de **fila** e de **deque**:
  - o jogo da **Batata Quente** (usando uma fila modificada: a **fila circular**) e
  - um verificador de **palíndromo** usando um **deque**.