Universidade de Pernambuco

Monitoria de Estruturas de Dados

2015.1

Monitoria Estrutura de Dados 2015.1

Hartur Barreto Brito
Escola Politécnica de Pernambuco

Abril, 2015

Sumário

1 Introdução

Estruturas de dados consistem em formas de organização dos dados para que possam ser utilizados de maneira mais eficiente. A forma de organizar os dados pode contribuir de diversas maneiras, e cada estrutura tem características diferentes. Exemplos de contribuíções são: Flexibilidade das estruturas, rapidez no processamento da informação, facilidade de manipulação, entre outros.
Algumas dessas estruturas serão introduzidas nesse documento, são elas: **lista**, **fila**, **pilha** e **árvore**.

2 Nó

Antes de descrever as estruturas de dados, é necessário conhecer o elemento que é utilizado no desenvolvimento delas, o **nó**.
O conteúdo do nó irá depender da estrutura de dados que será utilizada. Geralmente ele é composto por um atributo, que será o conteúdo do nó, e um ou mais ponteiros, que irão apontar para o(s) próximo(s) nó(s).
As formas de organização dos dados do nó vão depender de como as operações são realizadas sobre as estruturas.
Alguns exemplos de Nós podem ser observados nas figuras 1 e 2.

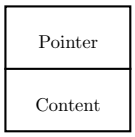

Figura 1: Nó com um ponteiro.

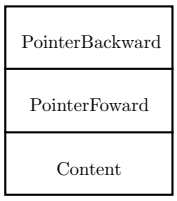

Figura 2: Nó com dois ponteiros.

3 Lista

Uma das vantagens na utilização da estrutura **lista** é a flexibilidade na manipulação dos dados por ela não precisar ser armazenada sequencialmente na memória, além de permitir que sejam adicionados ou removidos elementos (nós) em qualquer parte da estrutura, contanto que mantenha o princípio da organização do nó.
Alguns exemplos de operações realizadas sobre uma lista podem ser vizualizadas nas figuras 3 e 4

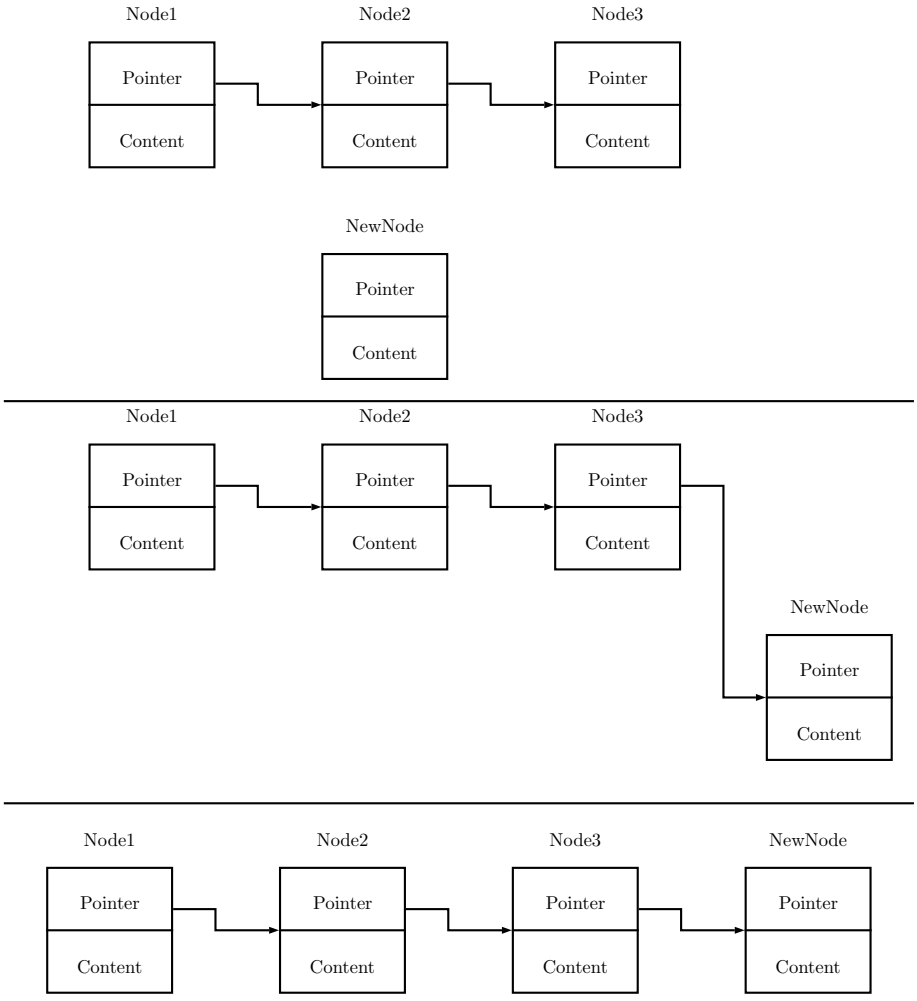

Figura 3: Adicionar nó no fim da lista (Append).

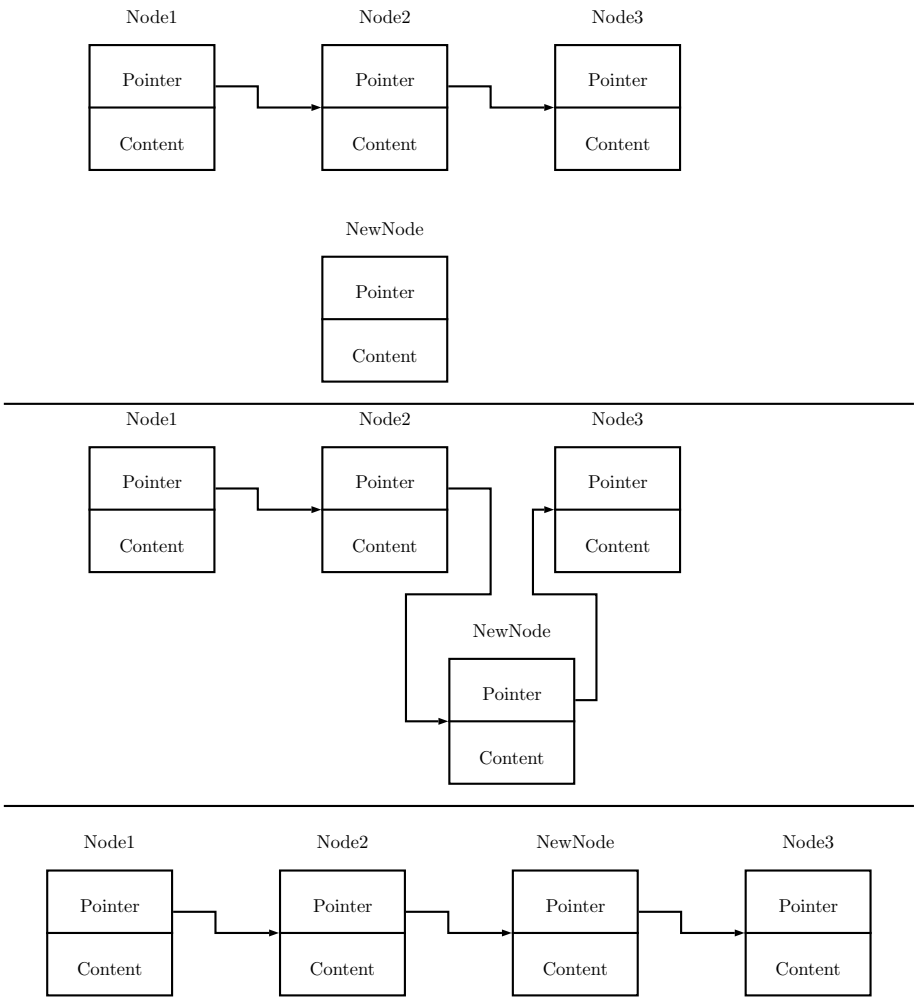

Figura 4: Adicionar nó no meio da lista.

Existem várias variações da estrutura lista, algumas delas são: **lista encadeada** (3.1), **lista circular** (3.2) e **lista duplamente ligada** (3.3).

3.1 Lista Encadeada, Ligada ou Linear

É composta por nós que apontam para o próximo elemento da lista e o último elemento apontará para nulo. A estrutura de uma lista duplamente ligada pode ser vizualizada na figura 5.

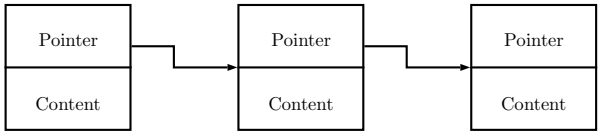

Figura 5: Lista Encadeada, Ligada ou Linear.

3.2 Lista Circular

É composta por nós que apontam para o próximo elemento da lista e o último elemento apontará para o primeiro elemento. A estrutura de uma lista duplamente ligada pode ser vizualizada na figura 6.

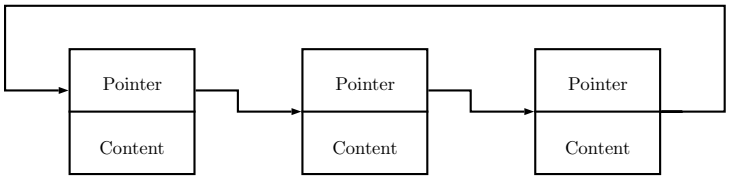

Figura 6: Lista Circular.

3.3 Lista Duplamente Ligada

É composta por nós que apontam tanto para o próximo elemento da lista quanto para o elemento anterior.
O último elemento é ligado com o primeiro e o primeiro com o último.
A estrutura de uma lista duplamente ligada pode ser vizualizada na figura 7.

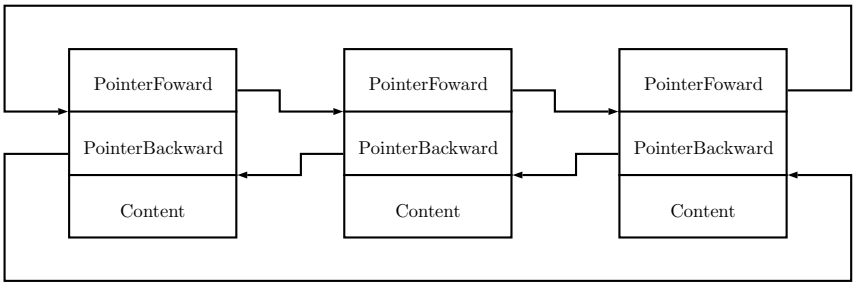

Figura 7: Lista Duplamente Ligada.

4 Fila

A estrutura **fila** é baseada na estrutura de uma lista. Entretanto ela utiliza o princípio FIFO (*First In First Out*), isto é, o primeiro elemento que chega é o primeiro a sair.
As operações realizadas nas filas são: *enqueue* (enfileirar) e *dequeue* (desenfileirar).

4.1 Enqueue

Esta operação é utilizada para adicionar novos elementos no fim da fila. Ela pode ser representada pelo seguinte pseudocódigo:

```
1                %The pointer of the last element of the queue must be set to the new ...
                     element.
2                last.pointer = newNode
3                %The pointer of the new element of the queue must point to null.
4                newNode.pointer = null
5                %The last element of the queue must be set to the new element.
6                last = newNode
```

Esta operação também pode ser visualizada na figura 8.

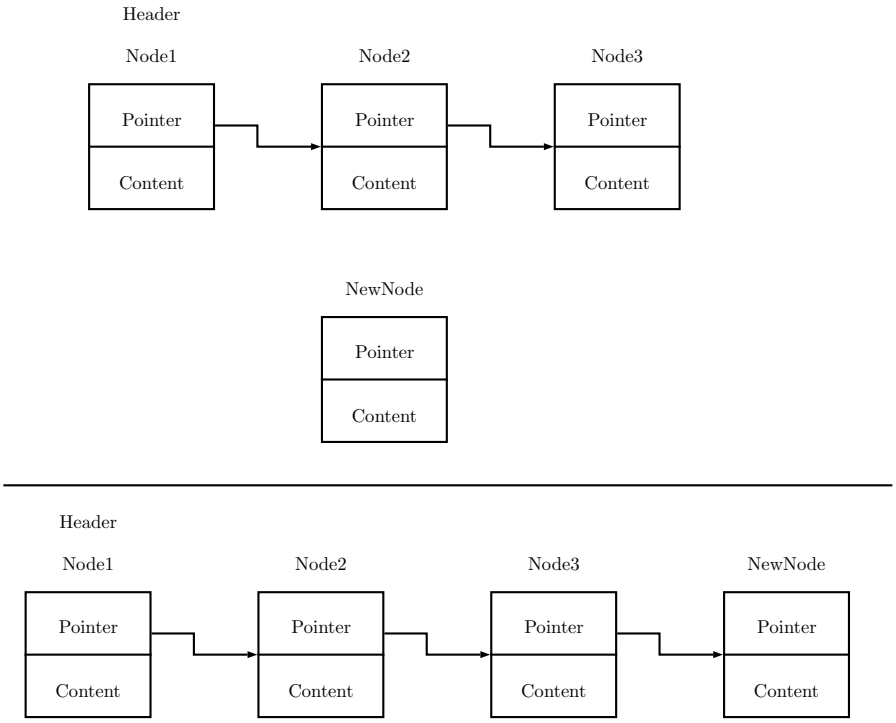

Figura 8: Operação Enqueue.

4.2 Dequeue

Esta operação é utilizada para retirar o elemento que está no início da fila. Ela pode ser representada pelo seguinte pseudocódigo:

```
1                %The queue's begin pointer must be set to the second element of the ...
                     queue.
2                head = firstNode.pointer
3                %The queue's first element must be returned.
4                return firstNode
```

Esta operação também pode ser visualizada na figura 9.

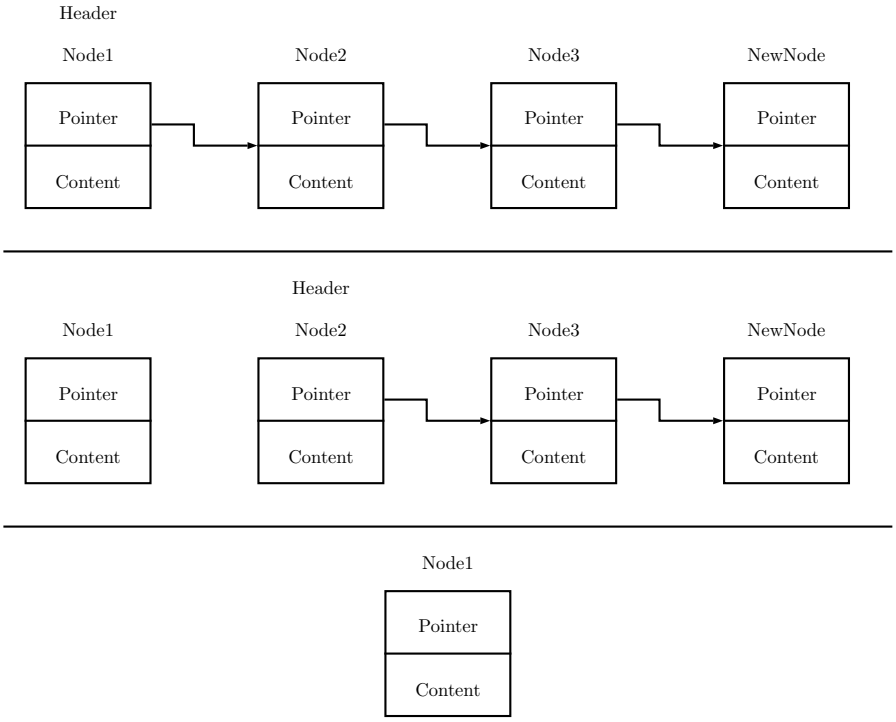

Figura 9: Operação Dequeue.

5 Pilha

A estrutura **pilha** é baseada na estrutura de uma lista. Entretanto ela utiliza o princípio FILA (*First In Last Out*), isto é, o primeiro elemento que chega é o último a sair.
As operações realizadas nas pilhas são: *push*, *pop*, *peak* e *isEmpty*.

5.1 Operaçõs

5.1.1 Push

Essa operação é utilizada para adicionar novos elementos na pilha. Ela pode ser representada pelo seguinte pseudocódigo:

```
1                    %The pointer of the new element must point to the top of the stack.
2                    newNode.pointer = top
3                    %The top's pointer must point to the new element
4                    top.pointer = newNode
```

Esta operação também pode ser visualizada na figura 10.

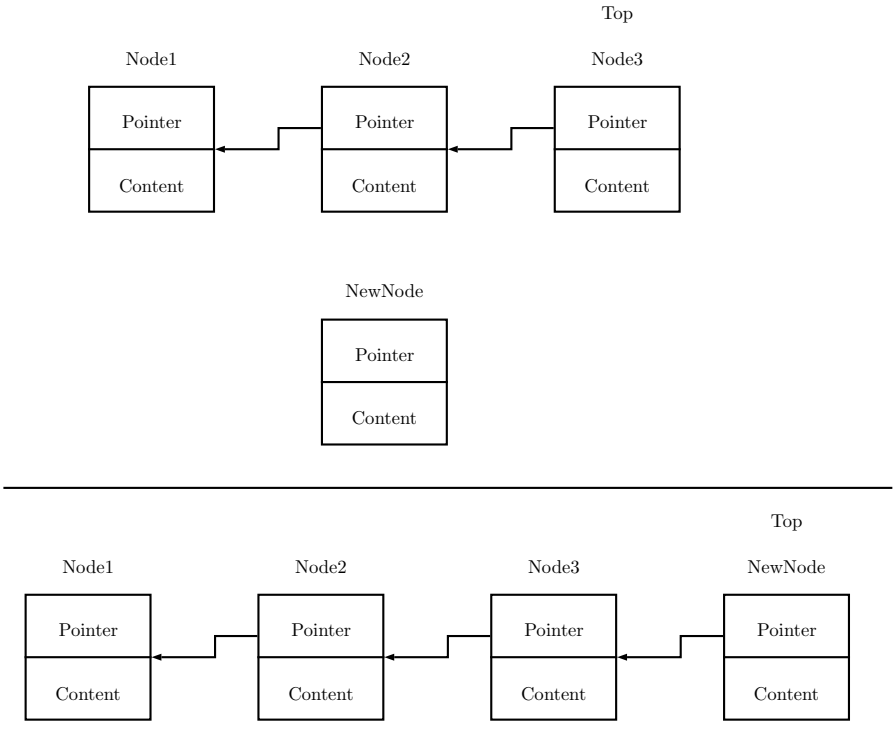

Figura 10: Operação Push.

5.1.2 Pop

Essa operação é utilizada para remover elementos da pilha. Ela pode ser representada pelo seguinte pseudocódigo:

```
%An auxiliar element must store the value of the top
aux = top
%The top's pointer must point to the second element in the stack
top.pointer = top.next
%The auxiliar element must be returned
return aux
```

Esta operação também pode ser visualizada na figura 11.

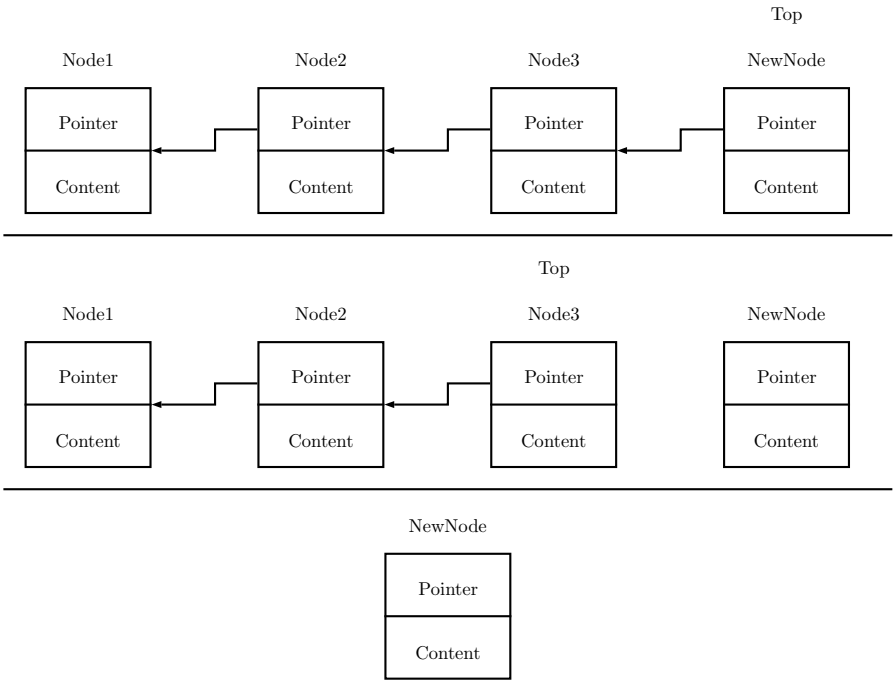

Figura 11: Operação Pop.

5.1.3 Peek

Essa operação é utilizada para verificar qual elemento está no topo da pilha. Ela pode ser representada pelo seguinte pseudocódigo:

```
%The top of the stack must be returned
return top
```

5.1.4 isEmpty

Essa operação é utilizada para verificar se a pilha está vazia. Ela pode ser representada pelo seguinte pseudocódigo:

```
%If the stack is empty, true must be returned and false otherwise.
return top == null
```

5.2 Exemplos de utilização

5.2.1 Expressões pós-fixas

Expressões pós-fixas é uma forma de representar expressões (Ex: matemáticas ou linguagens de programação) de uma forma diferente, na qual as operações são seguidas dos operandos.
Exemplo:
$(a + b) * (c - d)$
Pode ser representada como:
$a\ b + c\ d - *$

5.2.2 Expressões pré-fixas

Expressões pré-fixas segue a mesma ideia das expressões pós-fixas, entretanto, as operações são escritas antes dos operandos.
Exemplo: $(a + b) * (c - d)$
Pode ser representada como:
$* + a\ b - c\ d$

6 Árvore

A estrutura **árvore** tem esse nome por seu formato que, após estruturada, se assemelha ao de uma árvore.
Para entender uma árvore, precisamos saber identificar os seus elementos. São eles:

- Nó sem pai: **raiz**;
- Nós com mesmo pai: **irmão**;
- Nó sem filhos: **folha**.

Esses elementos podem ser melhor analisados na figura 12.

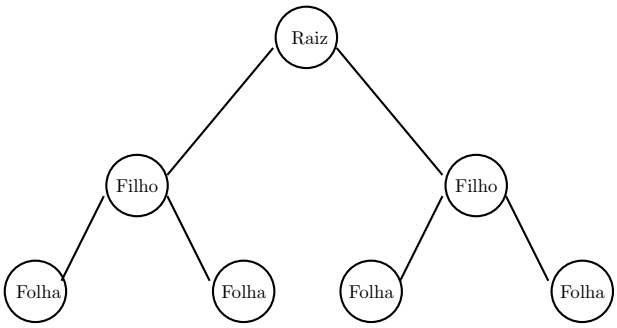

Figura 12: Elementos da Árvore.

Outros conceitos também importantes para o entendimento do funcionamento das árvores são:

- Profundidade do nó:
  Distância do nó até a raiz. O nó raiz sempre terá profundidade 0. Tomando como exemplo a figura 12, os nós *folha* estão a uma profundidade de 2;
- Nível da árvore:
  Conjunto de nós com a mesma profundidade. No exemplo da figua 12, os nós filho estão no nível 1 e os nós folha no nível 2;
- Altura da árvore:
  Maior profundidade de um nó. A árvore da figura 12,por exemplo, tem 2 de altura;
- Tamanho do caminho:
  número de arestas seguidas no caminho;
- Caminho de tamanho 0:
  nó para ele mesmo.

6.1 Percorrendo os Nós

Antes de seguir para os tipos de árvores, serão explanadas algumas formas de percorrer os nós das árvores.

6.1.1 Pré-ordem

O passo-a-passo para percorrer os nós de uma árvore em pré-ordem são:

1. Visita raiz;
2. Percorre sub-árvore esquerda em pré-ordem;

3. Percorre sub-árvore direita em pré-ordem.

E esse passo-a-passo pode ser descrito pelo seguinte pseudocódigo:

```
1                   preOrder(Tree t){
2                       t.seed
3                       preOrder(t.seed.left)
4                       preOrder(t.seed.right)
5                   }
```

Na figura 13, os números dentro dos nós representam a ordem que eles seram visitados.

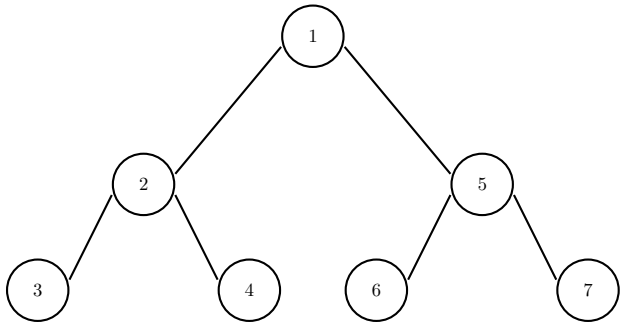

Figura 13: Visitar nós em Pré-Ordem.

6.1.2 Simétrica

O passo-a-passo para percorrer os nós de uma árvore em ordem simétrica são:

1. Percorre sub-árvore esquerda em ordem simétrica;
2. Visita raiz;
3. Percorre sub-árvore direita em ordem simétrica.

E esse passo-a-passo pode ser descrito pelo seguinte pseudocódigo:

```
1                   inOrder(Tree t){
2                       inOrder(t.seed.left)
3                       t.seed
4                       inOrder(t.seed.right)
5                   }
```

Na figura 14, os números dentro dos nós representam a ordem que eles seram visitados.

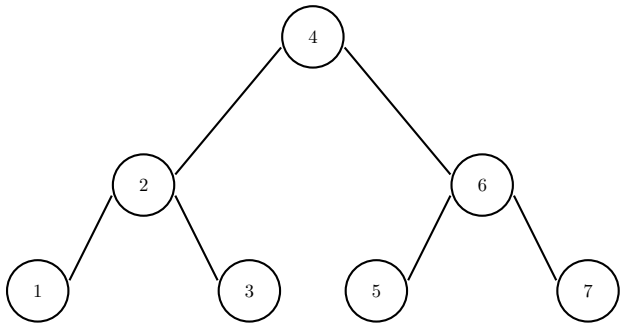

Figura 14: Visitar nós em Ordem.

6.1.3 Pós-ordem

O passo-a-passo para percorrer os nós de uma árvore em ordem pós-ordem são:

1. Percorre sub-árvore esquerda em pós-ordem;
2. Percorre sub-árvore direita em pós-ordem;
3. Visita raiz.

E esse passo-a-passo pode ser descrito pelo seguinte pseudocódigo:

```
1                   postOrder(Tree t){
2                       postOrder(t.seed.left)
3                       postOrder(t.seed.right)
4                       t.seed
5                   }
```

Na figura 15, os números dentro dos nós representam a ordem que eles seram visitados.

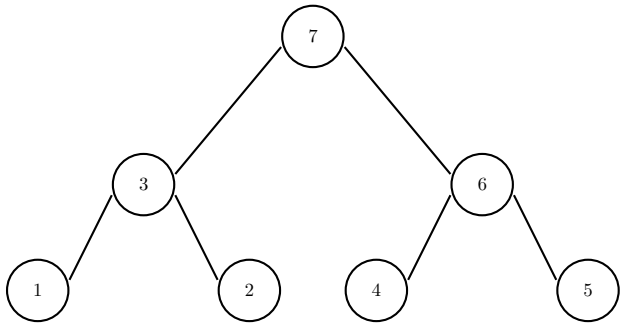

Figura 15: Visitar nós em Pós-Ordem.

6.2 Tipos de Árvores

Existem várias variações da estrutura árvore, algumas delas são: **Árvore Binária** (6.2.1) e **Árvore AVL** (6.2.2).

6.2.1 Árvores Binárias

As árvores binárias são caracterizadas por cada elemento ter no máximo 2 filhos, que são referenciados como "filho da direita"e "filo da esquerda".
As árvores binárias também podem ser classificadas como **Árvore Binária Ordenada**, na qual ao adicionar um novo elemento, ele sempre ficará à esquerda dos elementos de maior valor, ou à direita dos elementos de menor valor. A árvore representada na figura 16 a ordem de adição dos nós foi: 200, 123, 150, 100, 250, 300 e 220
As árvores binárias podem ser consideradas **Árvores Binárias Completas** se somente se todas as

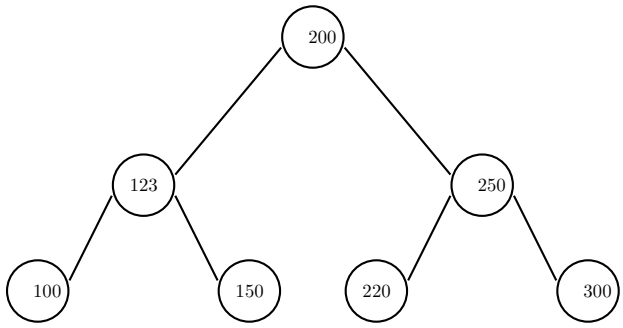

Figura 16: Árvore Binária Ordenada.

folhas de uma árvore estritamente binária de profundidade d estiverem no nível d, isto é, todas as folhas estarem no mesmo nível. A árvore representada na figura 12 é um exemplo de Árvore Binária Completa.

6.2.2 Árvores AVL

As árvores AVL são caracterizadas por serem balanceadas, isto é, a diferença de altura entre a sub-árvore do lado esquerdo e a do lado direito tem que ser menor ou igual à 1.
A altura de uma árvore AVL com n nós é $\log_2(n)$.
Para balancear uma árvore e torná-la uma AVL, existem algumas operações que podem ser realizadas. São elas:

- Rotação à direita:
  A figura 17 demonstra como é realizada a operação de rotação à direita.
- Rotação à esquerda:
  A figura 18 demonstra como é realizada a operação de rotação à esquerda.

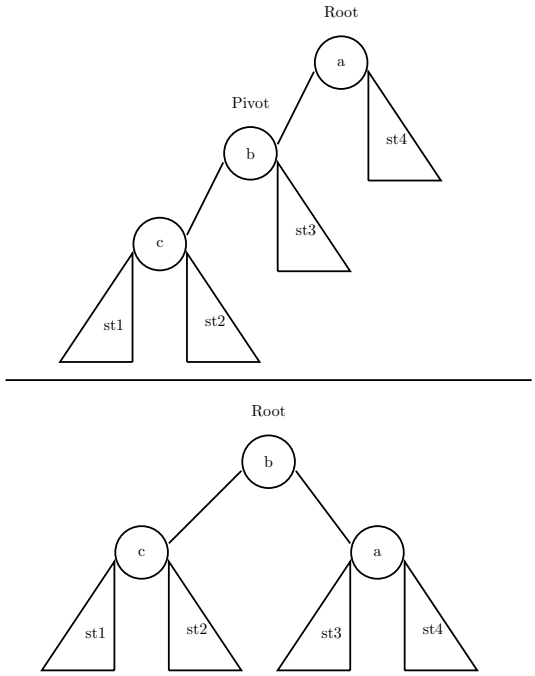

Figura 17: Rotação à direita.

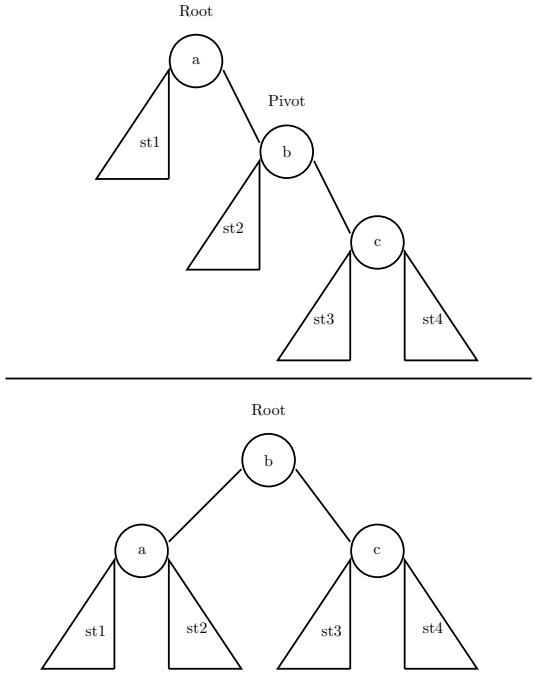

Figura 18: Rotação à esquerda.

6.3 Exemplos de Utilização

6.3.1 Árvores de Expressões

As **árvores de expressões** são utilizadas para representar expressões utilizando a estrutura de dados árvore.

Os modos de visitar os nós vai depender do formato utilizado na representação da expressão.

- Visita de nós Simetricamente:
  Utilizando a visita de nós simétrica (em ordem), a expressão será impressa normalmente com parênteses.
- Visita de nós em Pós-ordem:
  Utilizando a visita de nós pós-fixa (pós-ordem), a expressão será impressa na notação pós-fixa.
- Visita de nós em Pré-ordem:
  Utilizando a visita de nós pré-fixa (pré-ordem), a expressão será impressa na notação pré-fixa.

Um exemplo de Árvore de Expressão pode ser observado na figura 19

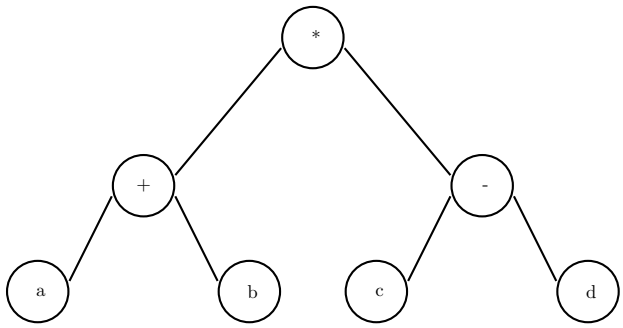

Figura 19: Expression Tree.

7 Questões

1. Considerando que os métodos de *push*, *pop* e *isEmpty* de uma pilha já foram implementados, implemente o método *peek* dessa pilha.

2. Verifique se uma String é palíndromo, retornando *true* caso seja e *false* caso não, usando a estrutura pilha.

3. Inverta uma String usando a estrutura fila.