

Árvores

Conceitos, implementação e árvores binárias

Prof. Marcelo de Souza

`marcelo.desouza@udesc.br`



Algoritmos e Estruturas de Dados
Bacharelado em Engenharia de Software
Universidade do Estado de Santa Catarina



Leitura obrigatória:

- Capítulo 5 de [Edelweiss and Galante \(2009\)](#) – Árvores.

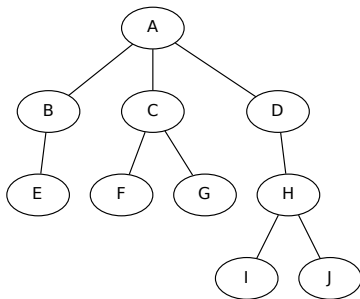
Leitura complementar:

- Capítulo 8 de [Goodrich et al. \(2014\)](#) – Árvores.
- Capítulo 14 de [Pereira \(2008\)](#) – Árvores.



Árvore

- Estrutura de dados que modela **dependência** ou **hierarquia**.
 - Elementos não são organizados de maneira linear.
- Representada graficamente por um conjunto de nodos interligados.





- **Hierarquias de especialização** (classes/sub-classes):
 - Tipos de veículos:
 - ▷ Aquático (barco, canoa).
 - ▷ Aéreo (avião, helicóptero).
 - ▷ Terrestre (carro, moto).



- **Hierarquias de especialização** (classes/sub-classes):
 - Tipos de veículos:
 - ▷ Aquático (barco, canoa).
 - ▷ Aéreo (avião, helicóptero).
 - ▷ Terrestre (carro, moto).
- **Hierarquias de composição** (todo/parte):
 - Um carro é composto por chassi, motor e rodas.



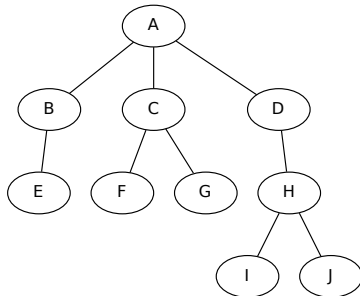
- **Hierarquias de especialização** (classes/sub-classes):
 - Tipos de veículos:
 - ▷ Aquático (barco, canoa).
 - ▷ Aéreo (avião, helicóptero).
 - ▷ Terrestre (carro, moto).
- **Hierarquias de composição** (todo/parte):
 - Um carro é composto por chassi, motor e rodas.
- **Hierarquias de subordinação ou dependência**:
 - Uma empresa organiza seus cargos e setores hierarquicamente:
 - ▷ Presidência comanda diretores.
 - ▷ Diretores comandam gerentes.
 - ▷ Gerentes comandam operadores.



- **Hierarquias de especialização** (classes/sub-classes):
 - Tipos de veículos:
 - ▷ Aquático (barco, canoa).
 - ▷ Aéreo (avião, helicóptero).
 - ▷ Terrestre (carro, moto).
- **Hierarquias de composição** (todo/parte):
 - Um carro é composto por chassi, motor e rodas.
- **Hierarquias de subordinação ou dependência**:
 - Uma empresa organiza seus cargos e setores hierarquicamente:
 - ▷ Presidência comanda diretores.
 - ▷ Diretores comandam gerentes.
 - ▷ Gerentes comandam operadores.

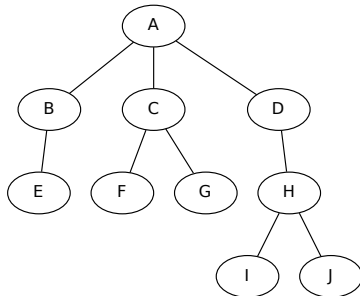
Faça a representação gráfica destes exemplos.

- **Raiz:** é o primeiro nodo da árvore, ao qual todos os demais são subordinados. O acesso à árvore sempre inicia por ele.
 - *A* é a raiz da árvore.

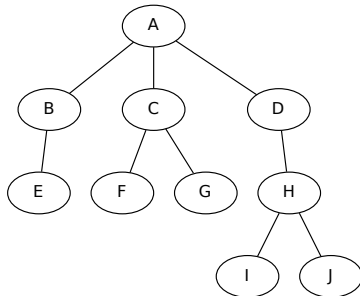




- **Nodos descendentes:** aqueles que estão abaixo de um determinado nodo mais acima, apresentando alguma relação de dependência.
 - Nodos mais acima são chamados ascendentes.
 - Descendentes diretos são chamados de *filhos*.
 - Um ascendente direto é chamado de *pai*.
 - F e G são descendentes/filhos de C (pai) e são ditos *irmãos*.
 - F e G também são descendentes de A .

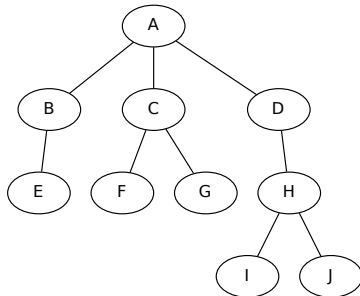


- **Sub-árvore:** conjunto de nodos descendentes de um mesmo nodo.
 - C , F e G formam uma sub-árvore de A .
 - H , I e J formam uma sub-árvore de D .



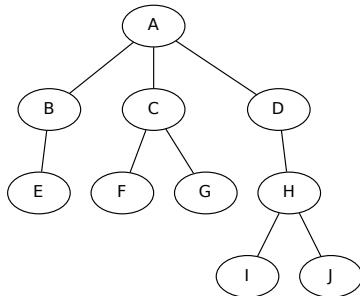


- **Grau do nodo:** número de sub-árvores que o nodo possui. Ou seja, o número de filhos de um nodo.
 - O nodo *A* possui grau 3.
 - O nodo *C* possui grau 2.
 - O nodo *E* possui grau 0.



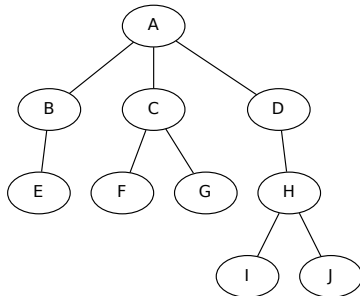


- **Grau da árvore:** o maior valor entre os graus dos seus nodos.
 - A árvore possui grau 3.



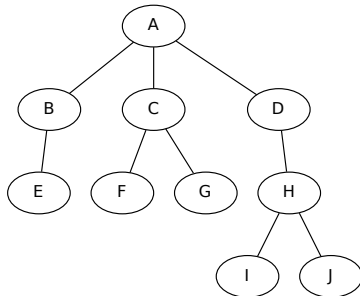


- **Folha:** nodos de grau 0, ou seja, que não possuem filhos.
 - Também chamados de nodos terminais ou externos.
 - As folhas da árvore são *E*, *F*, *G*, *I* e *J*.



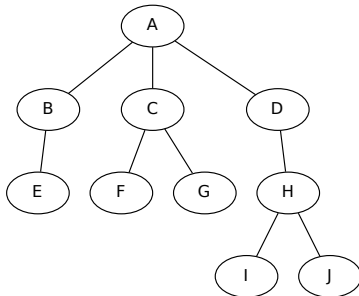


- **Nodo de derivação:** nodos de grau maior que 0, ou seja, que apresentam filhos (sub-árvores).
 - Também chamados de nodos internos.
 - Os nodos de derivação da árvore são *A*, *B*, *C*, *D* e *H*.



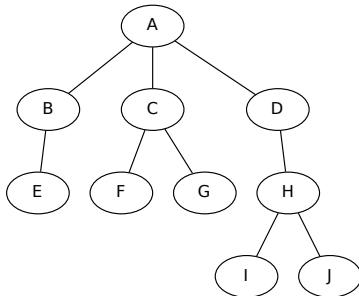


- **Nível do nodo:** número de ligações entre o nodo e a raiz da árvore. Corresponde à profundidade do nodo na árvore. A raiz possui nível 1, seus filhos possuem nível 2, etc.
 - Também chamado de profundidade do nodo.
 - Os nodos *B*, *C* e *D* possuem nível 2.
 - O nodo *I* possui nível 4.

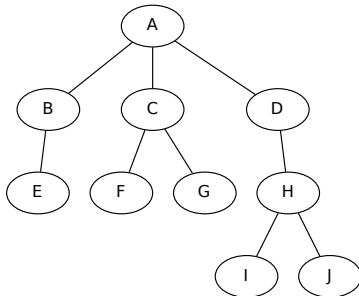




- **Caminho:** consiste em uma sequência de nodos consecutivos distintos entre dois nodos.
 - $A - D - H - J$ é um caminho na árvore.
 - O *comprimento de um caminho* é o número de ligações que ele possui (número de saltos).

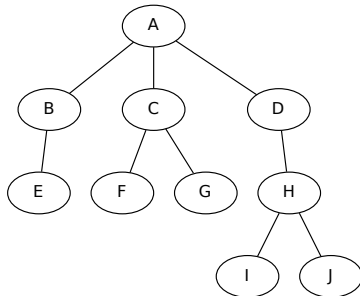


- **Altura:** a *altura de um nodo* é o número de nodos do maior caminho deste nodo até um dos seus descendentes-folha. A *altura da árvore* é igual ao maior nível dos seus nodos.
 - Todos os nodos-folha possuem altura 1.
 - A altura da árvore é 4, o que indica que existe pelo menos um nodo com distância 3 da raiz.

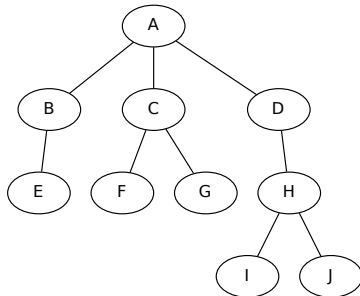




- **Árvore ordenada:** é uma árvore onde a ordem das suas sub-árvores é relevante para o seu contexto.
 - Se a árvore for ordenada, trocar a posição das sub-árvores com raiz B e C resulta em uma árvore diferente.



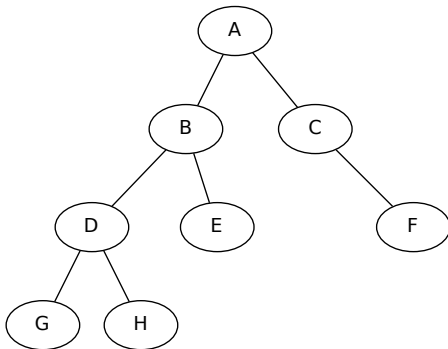
- **Árvore binária:** uma árvore cujos nodos possuem no máximo grau 2. Ou seja, um nodo possui 0, 1 ou 2 filhos.
 - Uma árvore n -ária possui grau no máximo n .
 - A árvore não é binária.



Árvores binárias



- Seus nodos possuem no máximo dois filhos.
- Cada nodo possui:
 - Filho da esquerda e filho da direita.
 - Sub-árvore da esquerda e uma sub-árvore da direita.

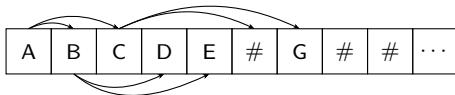
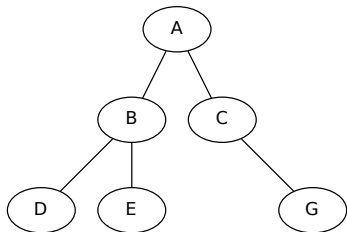


Implementação

Contiguidade – vetores



- Vetor armazena todos os nodos da árvore em uma ordem predefinida.
- Primeiro a raiz, depois seus filhos, depois os filhos dos seus filhos, etc.

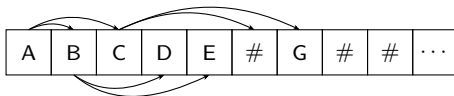
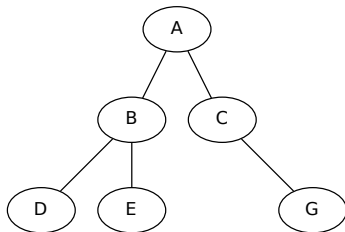


Implementação

Contiguidade – vetores



- Vetor armazena todos os nodos da árvore em uma ordem predefinida.
- Primeiro a raiz, depois seus filhos, depois os filhos dos seus filhos, etc.



Detalhes de implementação

Seja um elemento i da árvore:

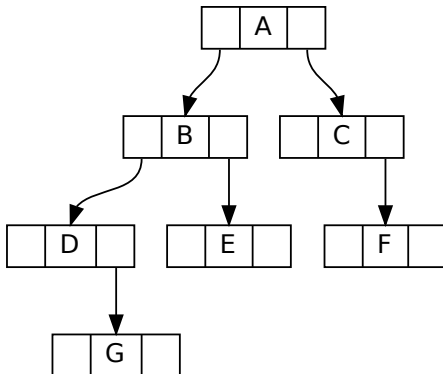
- $\text{leftChild}(i) = 2i + 1$
- $\text{rightChild}(i) = 2i + 2$
- $\text{parent}(i) = \lfloor (i - 1) / 2 \rfloor$

Implementação



Encadeamento

- Cada nodo possui um apontamento para seus filhos (direita e esquerda) e para seu pai, além de armazenar o elemento correspondente.
- A árvore é representada através de uma referência para sua raiz.



Exercício



Implementação com encadeamento

1. Implemente uma árvore binária usando encadeamento. A estrutura de dados deve fornecer as seguintes operações:
 - tamanho
 - vazio
 - inserir um elemento
 - setar um elemento
 - remover um elemento
 - irmãos de um nodo
 - profundidade de um nodo
 - altura de um nodo
 - verificar se um nodo é interno, externo e raiz

Dicas: crie uma classe `Node` que conterá o elemento, filhos e pai; o elemento é definido pela classe `Element`; a estrutura de nodos é visível externamente.

Exercício



Implementação com encadeamento

1. Implemente uma árvore binária usando encadeamento. A estrutura de dados deve fornecer as seguintes operações:
 - tamanho
 - vazio
 - inserir um elemento
 - setar um elemento
 - remover um elemento
 - irmãos de um nodo
 - profundidade de um nodo
 - altura de um nodo
 - verificar se um nodo é interno, externo e raiz

Dicas: crie uma classe `Node` que conterá o elemento, filhos e pai; o elemento é definido pela classe `Element`; a estrutura de nodos é visível externamente.

Veja a implementação nos códigos-fonte da disciplina.



- Um percurso (também chamado de travessia ou caminhamento) de uma árvore consiste em acessar (ou visitar) todos os seus nodos.
- Principais métodos de travessia de árvores:
 - Pré-ordem (profundidade);
 - Pós-ordem;
 - In-ordem;
 - Largura.
- Esses métodos são aplicáveis a qualquer árvore.
- Os algoritmos percorrem toda a árvore – complexidade $O(n)$.



- Dada uma árvore T , o percurso de pré-ordem:
 1. visita a raiz de T .
 2. as sub-árvores dos filhos são visitadas recursivamente.
- Pode ser adotada alguma estratégia para explorar os filhos seguindo uma ordem desejada.

Algorithm: preorder(Node<E> n)

Visit node n

foreach *child* c in *children*(n) **do**

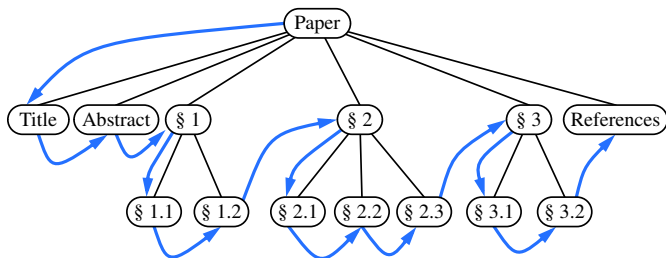
 | preorder(c)

Percurso em pré-ordem



Estratégia de profundidade:

- Acessa os filhos até chegar no final da árvore.
- Volta e continua a busca pelos demais filhos.





- Dada uma árvore T , o percurso de pós-ordem:
 1. visita recursivamente os filhos da raiz T .
 2. visita a raiz de T .
- Pode ser vista como o oposto da pré-ordem.

Algorithm: postorder(Node<E> n)

foreach *child* c **in** $children(n)$ **do**

 | postorder(c)

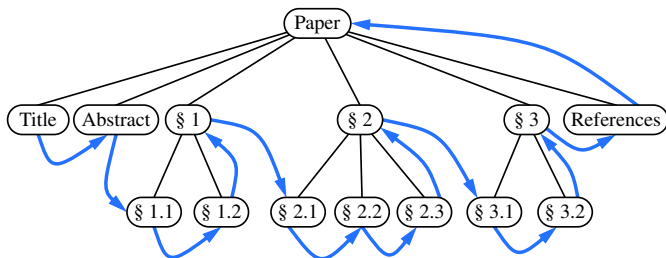
Visit node n

Percurso em pós-ordem



Estratégia das folhas até a raiz:

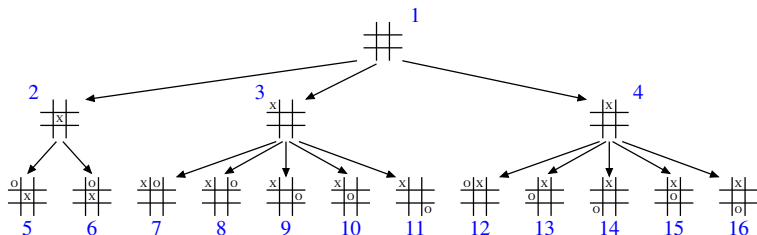
- Visitação começa pelas folhas.
- Último elemento visitado é a raiz da árvore.



Percurso em largura



- Percorre os nodos por níveis.
 1. explora todos os nodos de uma profundidade.
 2. depois passa para a próxima.
- Também chamado *breadth-first traversal*.
- Exemplo: encontrar a solução de jogos.





- Pode-se usar uma fila para implementar a estratégia (*FIFO*).
- Procedimento não é recursivo, pois não se aplica igualmente a sub-árvores.

Algorithm: `breadthfirst()`

Initialize queue Q to contain root

while Q is not empty **do**

$n \leftarrow Q.dequeue()$

 Visit node n

for each child c in $children(n)$ **do**

$Q.enqueue(n)$

Exercício



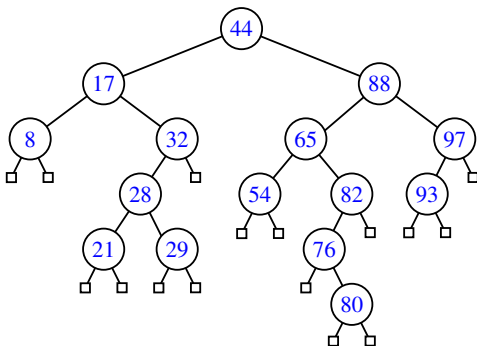
Implementação dos métodos de percurso

1. Implemente os métodos de percurso de pré-ordem, pós-ordem e em largura na sua árvore binária usando encadeamento. Utilize esses métodos para implementar algumas tarefas:
 - imprimir todos os elementos da árvore.
 - fazer um somatório dos elementos numéricos armazenados na árvore.
 - buscar um elemento na árvore.

Árvores binárias de busca



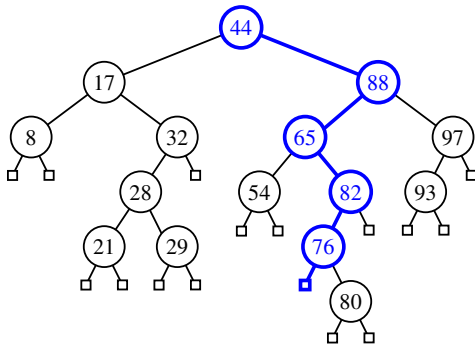
- Trata-se de uma **árvore binária ordenada**.
- Isso permite a busca eficiente de elementos.
- Propriedades:
 - Nodo n armazena um elemento $e(n)$.
 - Elementos na sub-árvore **esquerda** de n são **menores** que $e(n)$.
 - Elementos na sub-árvore **direita** de n são **maiores** que $e(n)$.



Árvores binárias de busca



- Buscar um elemento em uma árvore binária de busca é similar a fazer uma busca binária (pois a árvore está ordenada).
 1. Se o elemento buscado for o da raiz, retorna sucesso.
 2. Se for menor, repete a busca na sub-árvore da esquerda.
 3. Se for maior, repete a busca na sub-árvore da direita.
 4. Se chegar a alguma folha sem encontrar, elemento não existe.
- A cada iteração, metade da árvore é descartada – complexidade $O(\log n)$.

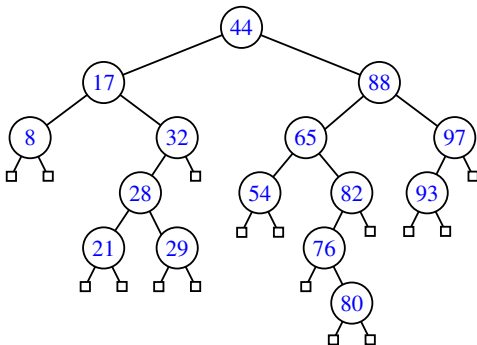


Árvores binárias de busca



Inserção de elementos

- A inserção deve respeitar a ordem dos elementos.
 1. Busca a posição correta de inserção.
 2. Insere o novo elemento.
- **Exemplo:**
 - o elemento 50 seria inserido como filho à esquerda do elemento 54.
 - o elemento 15 seria inserido como filho à direita do elemento 8.

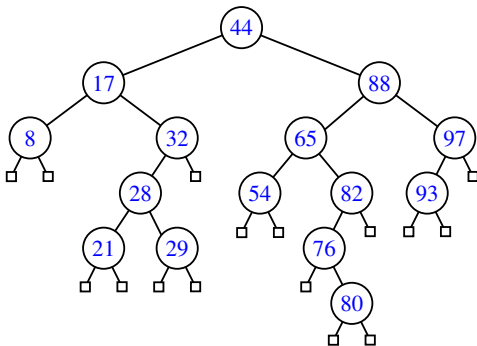


Árvores binárias de busca



Remoção de elementos

- A remoção deve manter a ordem dos elementos.
 1. Remove elemento.
 2. Se nodo removido é uma folha, termina.
 3. Se nodo removido possui um filho, filho toma sua posição.
 4. Se nodo removido possui dois filhos, rearranja.

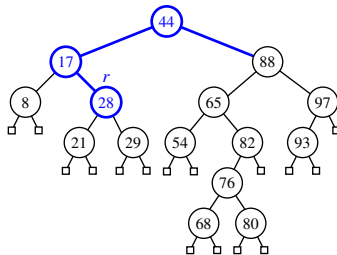
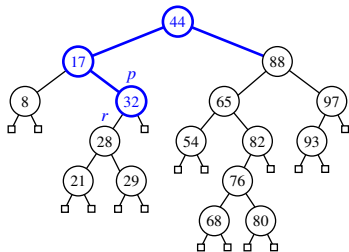


Árvores binárias de busca



Remoção de elementos

- Caso 1 – promover filho:
 1. p é o elemento a remover.
 2. seu único filho r toma seu lugar.



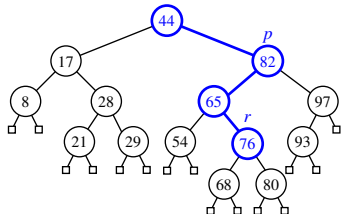
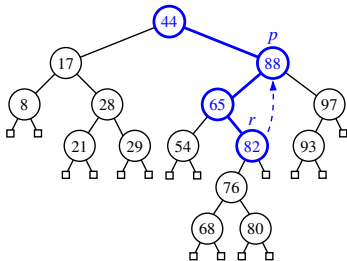
Árvores binárias de busca



Remoção de elementos

- Caso 2 – rearranjar:

1. p é o elemento a remover.
2. r é o maior elemento estritamente menor que p .
3. r toma o lugar de p .
4. r não terá filho à direita, portanto seu filho à esquerda toma seu lugar.



Árvores binárias de busca



Percurso em ordem

- Aplicável a árvores binárias **ordenadas**.
- Dada uma árvore T , o percurso em ordem:
 1. visita recursivamente a sub-árvore à esquerda da raiz T .
 2. visita a raiz de T .
 3. visita recursivamente a sub-árvore à direita da raiz T .

Algorithm: `inorder(Node<E> n)`

`inorder(left(n))`

Visit node n

`inorder(right(n))`

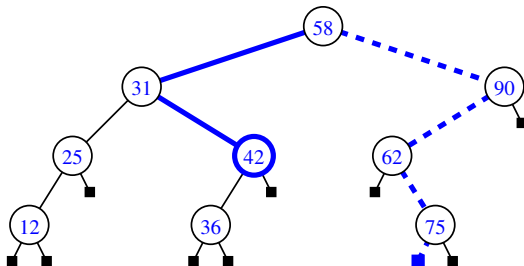
Árvores binárias de busca



Percurso em ordem

- Um percurso em ordem da árvore abaixo imprimirá:

12 – 25 – 31 – 36 – 42 – 58 – 62 – 75 – 90



Árvores binárias de busca



Complexidade das operações

- No pior caso, uma busca vai da raiz até uma das folhas.
- Sendo h a altura da árvore, a operação possui $O(h)$.
- Logo, inserção, substituição, remoção e busca em uma árvore binária de busca executam em $O(h)$.
- Melhor caso: $h = O(\log n)$.
- Pior caso: $h = O(n)$.
- Caso médio: $h = O(\log n)$.
- Variações da ABB garantem complexidade $O(\log n)$ no pior caso.
 - Exemplo: arvores balanceadas.

Árvores binárias de busca



Algumas aplicações

- Mapas:
 - Armazenam elementos $\langle \text{chave}, \text{valor} \rangle$ de forma ordenada.
 - Eficiente para as operações.
 - Veja a implementação em [Goodrich et al. \(2014\)](#).
- Filas de prioridade:
 - Armazenam elementos $\langle \text{chave}, \text{valor} \rangle$ onde a chave é a prioridade.
 - Eficiente para as operações.
 - São chamadas de *heaps*.
 - Veja a implementação em [Goodrich et al. \(2014\)](#).

Exercício



Implementação com encadeamento

1. Implemente uma árvore binária de busca usando encadeamento. Implemente todas as operações referentes a árvores e os algoritmos de percurso estudados.
 - Utilize a estrutura criada para armazenar elementos na estrutura $\langle \text{chave}, \text{valor} \rangle$. A chave pode ser um campo numérico, e o valor armazena objetos de uma classe qualquer.



Edelweiss, N. and Galante, R. (2009). *Estruturas de Dados: Volume 18*. Bookman Editora.

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

Pereira, S. d. L. (2008). *Estruturas de dados fundamentais: Conceitos e aplicações*.