Algoritmos II

Árvores: percursos e propriedades

Prof. Rodrigo Minetto

rminetto@dainf.ct.utfpr.edu.br
Universidade Tecnológica Federal do Paraná
2 Semestre 2012

Sumário

Árvores - Percurso

Um **percurso** corresponde a uma **visita** sistemática a cada um dos nós da árvore. Muitas operações em árvores binárias envolvem o percurso de todas as subárvores, com a execução de alguma ação de tratamento de cada nó.

Árvores - Percurso

Por exemplo, se cada nó da árvore possui um campo que armazena o salário, então podemos querer visitar cada nó para fazer um reajuste salarial. A visita seria atualizar o campo salarial. Não podemos esquecer de nenhum nó, nem queremos visitar um nó mais do que uma vez. Nesse caso a ordem da visita não é importante. Mas em algumas outras aplicações, queremos visitar os nós em certa ordem desejada.

Árvores - Percurso

Visitar um nó significa trabalhar com a informação do nó (por exemplo: imprimir, modificar, etc). Contudo, durante um percurso podemos passar várias vezes por alguns dos nós sem visitá-los.

Árvores - Percurso

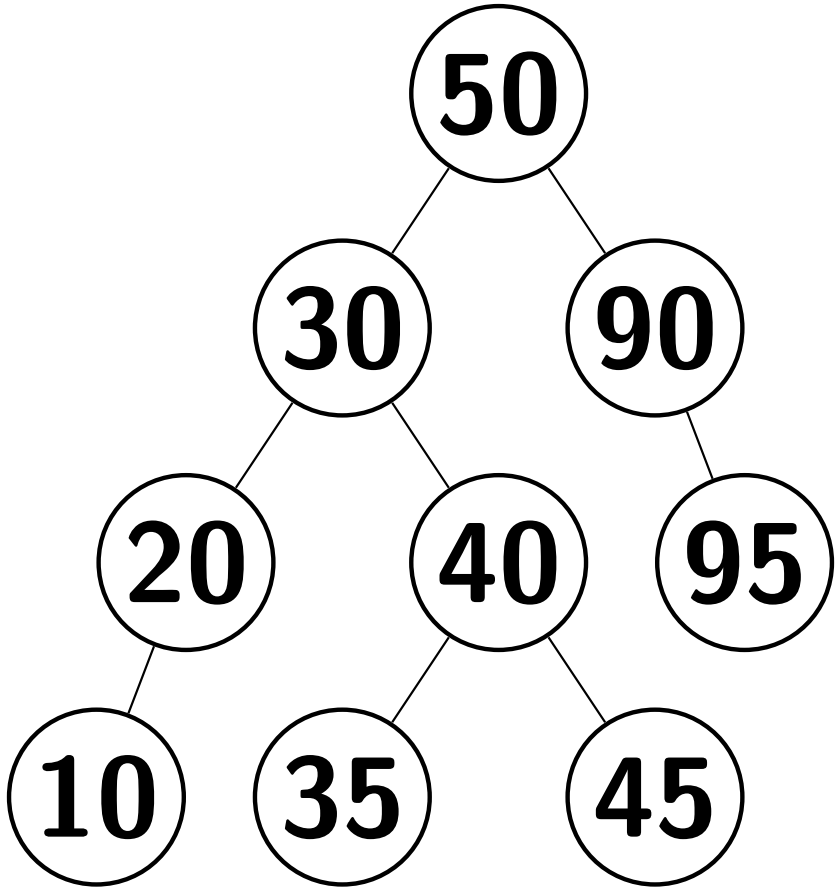
É comum percorrer uma árvore em uma das seguintes ordens:

- Pré-ordem (**R, E, D**):
  - Visitar a **raiz**;
  - Percorrer a sua **subárvore esquerda** em pré-ordem;
  - Percorrer a sua **subárvore direita** em pré-ordem;

Introdução - Percurso em árvores

Pré-ordem:

Exemplo: **50,30,20,10,40,35,45,90,95.**



Árvores - Percurso

O percurso em pré-ordem segue os nós até chegar aos mais **"profundos"**, em **"ramos"** de subárvores da esquerda para a direita. É conhecida pelo nome de percurso em **profundidade**.

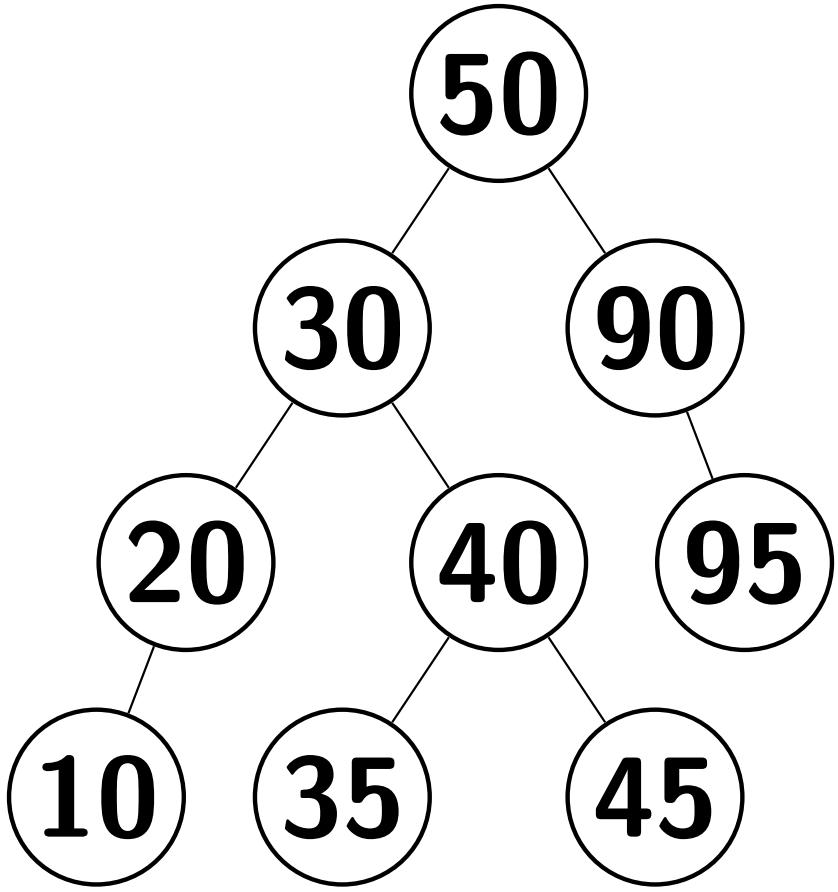
Árvores - Percurso

- Percurso in-ordem (**E, R, D**):
  - Percorrer a sua **subárvore esquerda** em in-ordem;
  - Visitar a **raiz**;
  - Percorrer a sua **subárvore direita** em in-ordem;

Introdução - Percurso em árvores

In-ordem:

Exemplo: **10,20,30,35,40,45,50,90,95.**



Árvores - Percurso

O percurso em in-ordem visita a raiz entre as ações de percorrer as duas subárvores. Esse percurso também é conhecido pelo nome de **ordem simétrica**.

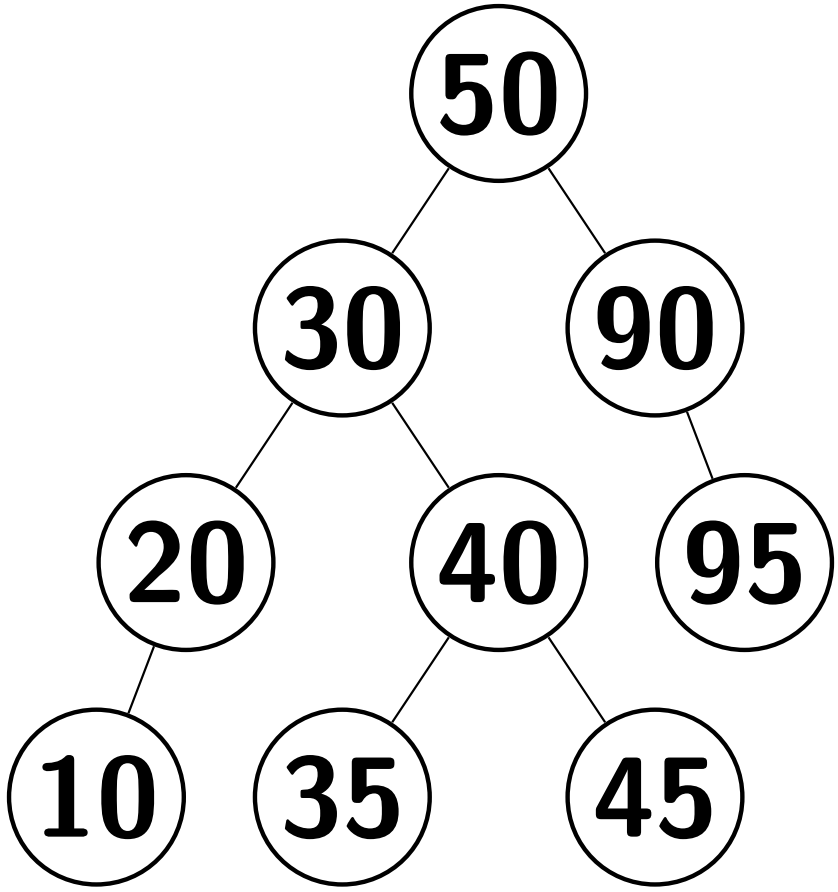
Árvores - Percurso

- Percurso pós-ordem (**E, D, R**):
  - Percorrer a sua **subárvore esquerda** em pós-ordem;
  - Percorrer a sua **subárvore direita** em pós-ordem;
  - Visitar a **raiz**;

Introdução - Percurso em árvores

Pós-ordem:

Exemplo: **10,20,35,45,40,30,95,90,50.**



Sumário

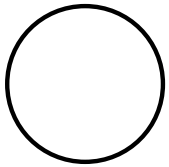
Árvores - Altura

Uma propriedade fundamental de todas as árvores é que só existe um caminho da raiz para qualquer nó. Com isso, podemos definir a **altura** de uma árvore como sendo o **comprimento do caminho mais longo da raiz até uma das folhas**.

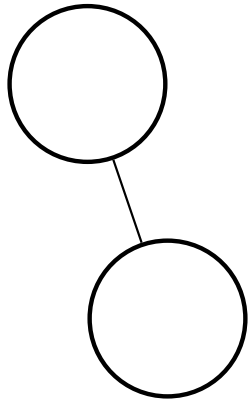
Árvores - Altura

Assim, a altura de uma árvore com um único nó raiz é zero e, por consequência, dizemos que a altura de uma árvore vazia é negativa e vale -1. Exemplos:

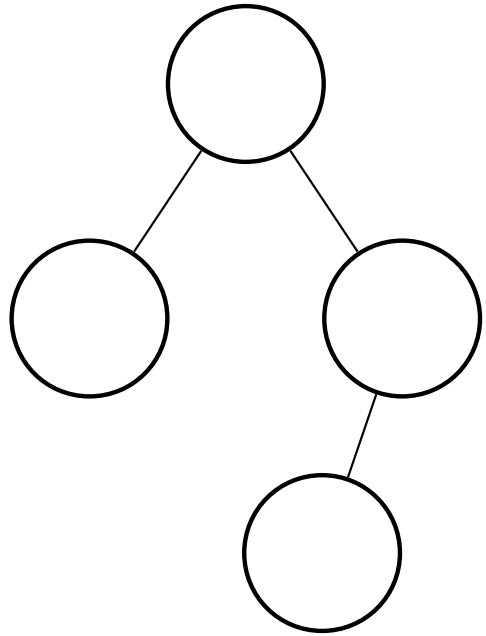
Altura $= 0$



Altura $= 1$

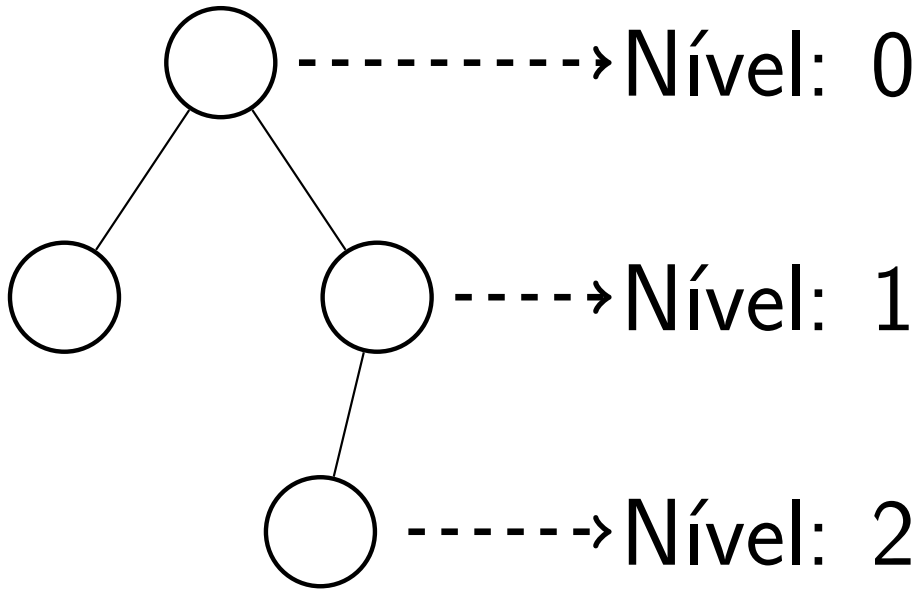


Altura $= 2$



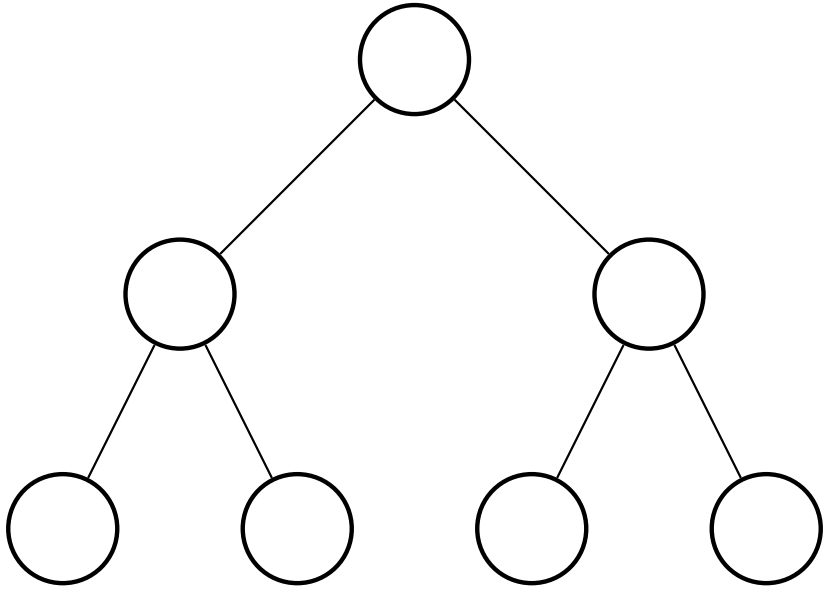
Árvores - Altura

Também podemos numerar os **níveis** em que os nós aparecem na árvore. A raiz está no nível 0, seus filhos diretos no nível 1, e assim por diante. O último nível da árvore é o nível h, sendo h a altura da árvore.



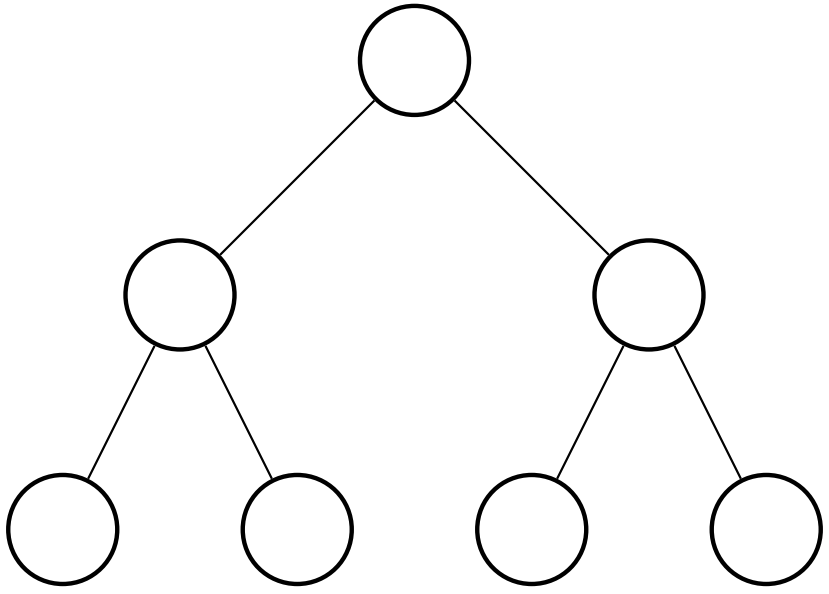
Árvores - Altura

Uma árvore binária é dita **cheia (ou completa)** se todos os seus nós internos têm duas subárvores associadas e todos os nós folhas estão no último nível. Exemplo:



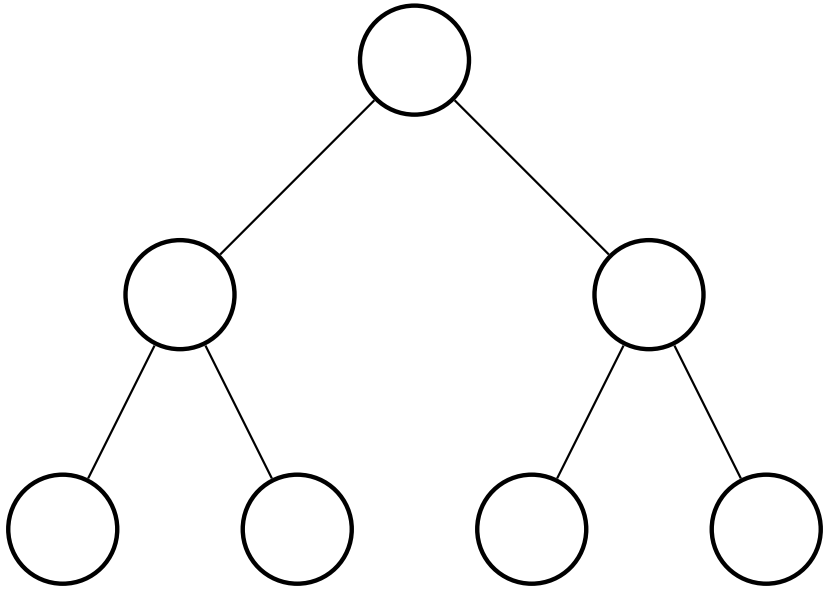
Árvores - Altura

Podemos notar que nesse tipo de árvore temos **um** nó no nível **0**, **dois** nós no nível **1**, **quatro** nós no nível **2**, **oito** nós no nível **3**, e assim por diante. Isto é, no nível n, temos 2^n nós.



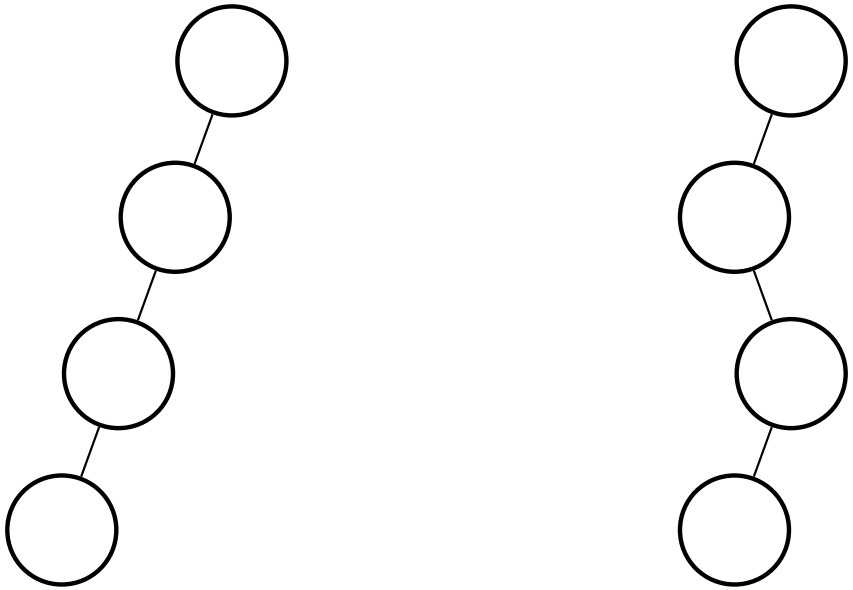
Árvores - Altura

É possível mostrar que uma árvore cheia de altura h tem um número de nós dado por $2^{h+1} - 1$.



Árvores - Altura

Uma árvore é dita **degenerada** se todos os seus nós internos têm uma única subárvore associada. Em uma árvore degenerada, temos um único nó em cada nível. Assim, uma árvore degenerada de altura h tem $h + 1$ nós.



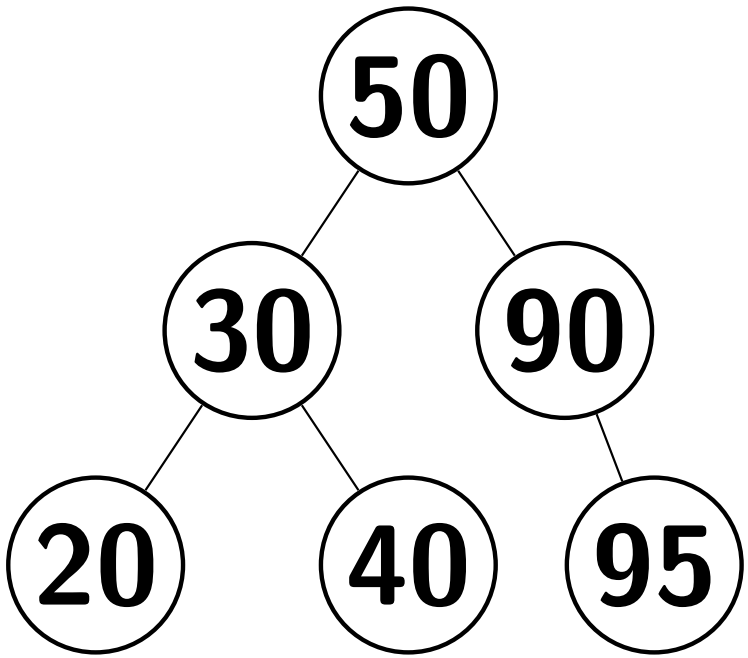
Árvores - Altura

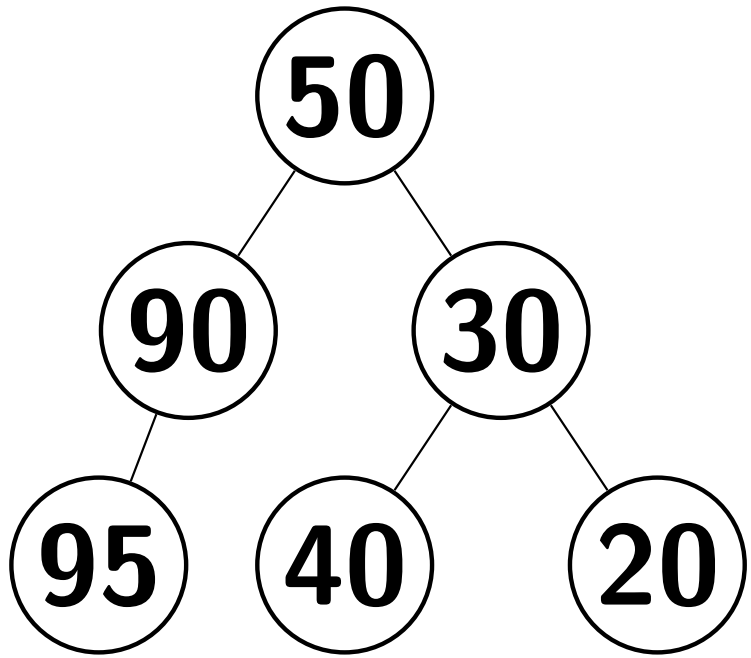
A altura de uma árvore é uma medida importante na avaliação da eficiência com que visitamos os nós da árvore. Uma árvore binária com n nós tem uma altura mínima proporcional a $\log n$ (caso da árvore cheia) e uma altura máxima proporcional a n (caso da árvore degenerada). **A altura indica o esforço computacional** *necessário para alcançar qualquer nó da árvore.*

Sumário

Introdução - Percurso em árvores

Duas árvores são **espelho-similares** se elas são vazias ou se elas não são vazias e suas subárvores esquerda de cada uma são **espelho-similares** as subárvores direita da outra.





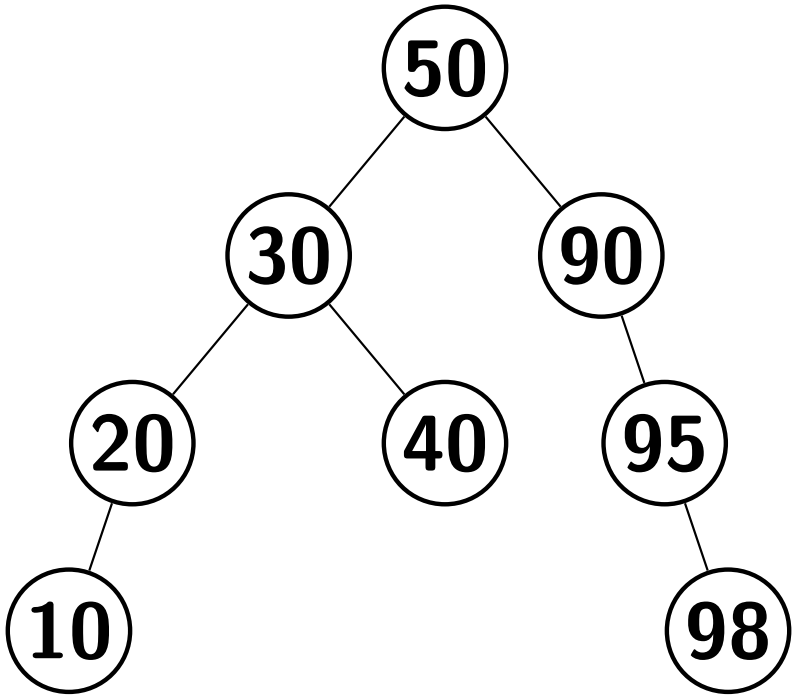
Sumário

Exercícios

Exercício 1) Escreva uma função que recebe uma árvore binária com entrada e a imprime nos seguintes tipos de percursos:

- pré-ordem
- in-ordem
- pós-ordem

Exercício 2) Suponha a árvore binária abaixo. (a) ache o grau de cada um dos nós. (b) determine os nós folhas. (c) complete a árvore com quantos nós forem necessários, para transformá-la em uma árvore binária cheia.



Exercício 3) Escreva uma função que conte o número de nós de uma árvore binária. Utilize o seguinte protótipo para a sua função:

```
int conta_nos (Arvore *a);
```

Exercício 4) Escreva uma função que imprime e retorna o maior valor inteiro em uma árvore binária composta por valores inteiros. Utilize o seguinte protótipo para a sua função:

```
int max_arvore (Arvore *a);
```

Exercício 5) Escreva uma função que calcula a altura de uma árvore binária. Utilize o seguinte protótipo para a sua função:

```
int calcula_altura_arvore (Arvore *a);
```

Exercício 6) Escreva uma função que conta o número de nós folhas em uma árvore binária. Utilize o seguinte protótipo para a sua função:

```
int conta_nos_folha (Arvore *a);
```

Exercício 7) Escreva uma função que verifica se uma dada árvore binária é cheia (ou completa). Utilize o seguinte protótipo para a sua função (retorna 1 se cheia, 0 caso contrário):

```
int verifica_arvore_cheia (Arvore *a);
```

Exercício 8) Escreva uma função que faz o espelho de uma árvore binária. Utilize o seguinte protótipo para a sua função:

```
Arvore espelha_arvore (Arvore *a);
```