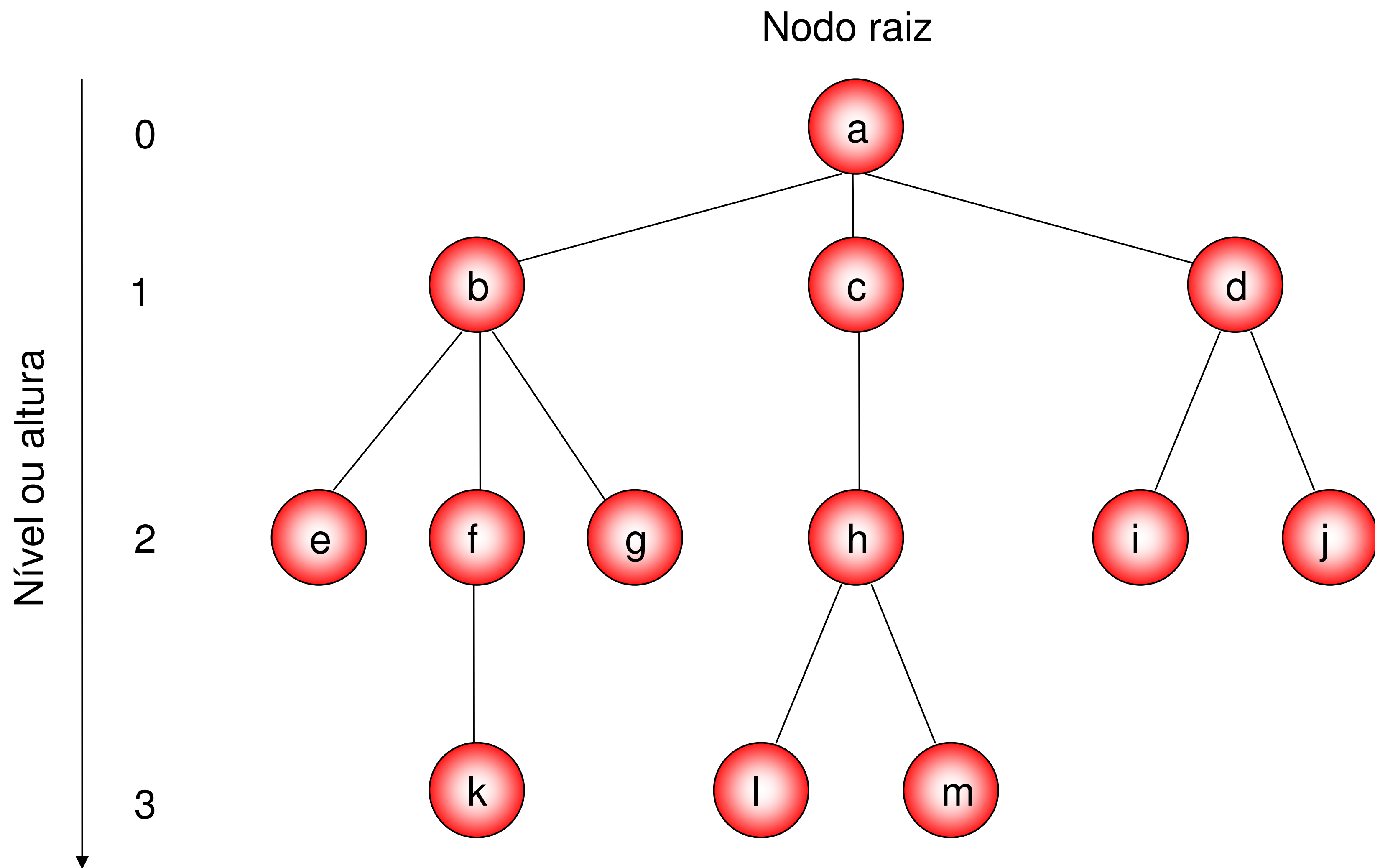
Aula 11 – Estrutura de Dados: Árvores

Prof. Tiago A. E. Ferreira

Árvore - Definição

- Uma árvore é ma coleção de $n \geq 0$ nodos:
  - Se $n = 0$, a árvore é dita nula
  - Se $n > 0$, a árvore tem as características:
    - O nodo inicial é chamado de **raiz (*root*)**
    - **Os demais nodos são particionados em T_1, T_2, ..., T_k estruturas disjuntas de árvores**
    - **As estruturas T_1, T_2, ..., T_k denominam-se sub-árvores**

Definição Gráfica de uma Árvore



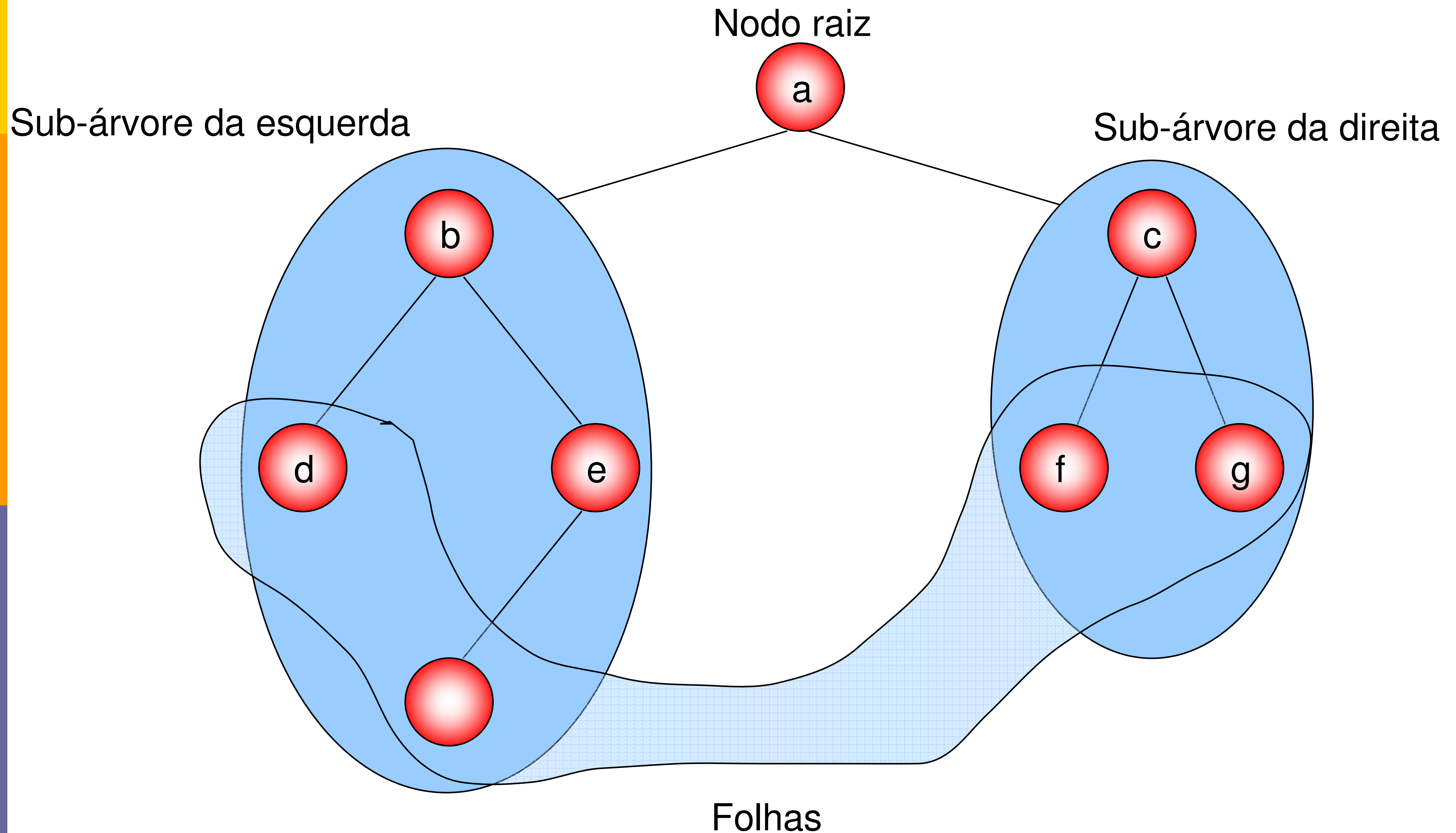
Observações

- O número de sub-árvores de um nodo denomina-se de **grau ou fator de expansão**
  - **Exemplo:** na figura passada o nodo ***a*** tem grau = 3, o nodo ***c*** tem grau = 1 e o nodo ***d*** tem grau = 1;
- **O grau de uma árvore é o maior grau existente em seus nodos**
  - **Exemplo:** a árvore passada tem grau = 3
- **Um nodo que tem grau = 0 denomina-se de terminal ou folha**
  - **Exemplo:** são folhas da árvore passada: *e*, *k*, *g*, *l*, *m*, *i* e *j*

Árvore Binária

- Uma árvore binária é um conjunto finito de elementos que está **vazio ou é particionada em três subconjuntos disjuntos**
  - **Primeiro subconjunto:** raiz ou root
  - **Segundo subconjunto:** sub-árvore esquerda
  - **Terceiro subconjunto:** sub-árvore direita
- Desta forma, uma árvore binária tem **grau = 2.**

Representação de uma Árvore Binária



Mais Observações...

- Se uma árvore binária contiver **m nós no nível l, então no nível l+1 conterá no máximo 2m nós**
  - Assim, no **nível l** uma árvore poderá conter no máximo $\mathbf{2^l}$
  - Logo, uma árvore binária completa de profundidade d conterá no máximo um total de nós (tn),

$$tn = 2^0 + 2^1 + 2^2 + \cdots + 2^d = \sum_{j=0}^{d} 2^j$$

Mais Observações...

- Por indução,

$$2^0 = 2^{0+1} - 1 = 1$$

$$2^0 + 2^1 = 2^{1+1} - 1 = 3$$

$$2^0 + 2^1 + 2^2 = 2^{2+1} - 1 = 7$$

$$\vdots$$

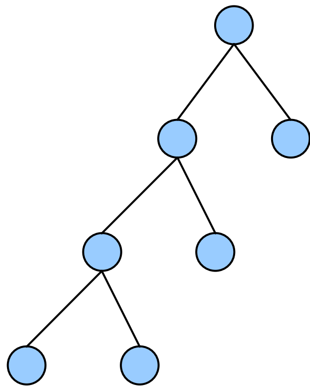
$$\sum_{j=0}^{d} 2^j = 2^{d+1} - 1$$

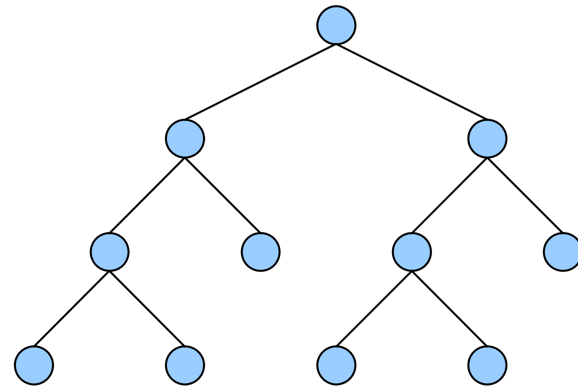
  - Assim, também é possível determinar a profundidade da árvore binária dado a quantidade de nós

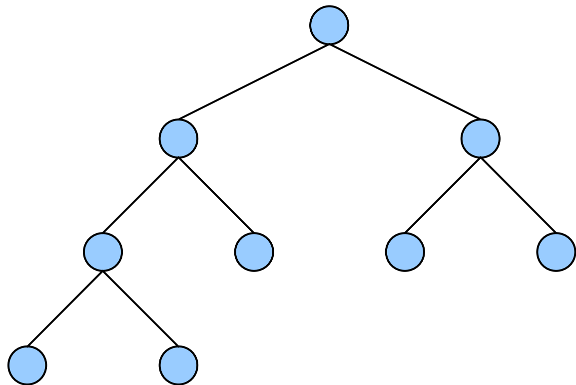
Árvore Completa e Quase-Completa

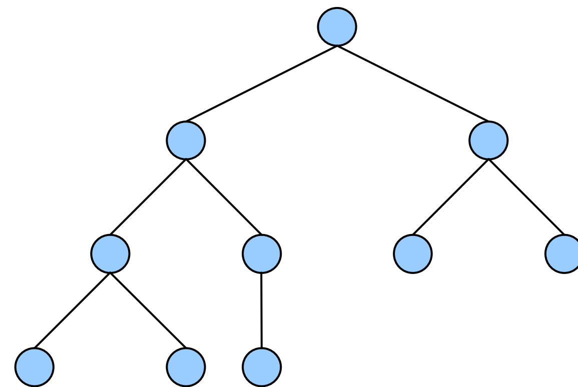
- Uma árvore binária é completa quando todos os pais têm dois filhos (esquerdo e direito)
- Uma árvore binária é quase completa quando:
  - Cada folha da árvore estiver no nível **d ou no nível d-1**
  - Para cada nó nd na árvore com um descendente direito no nível *d*, todos os descendentes esquerdos de *nd* que forem folhas estiverem também no nível *d.*

Quais das Árvores abaixo são Quase-Completas?



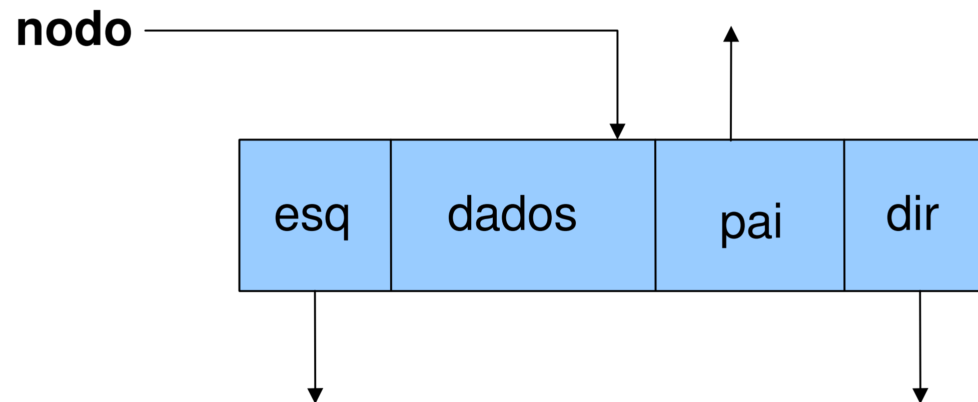






Representação de um Nodo

- Um nodo para uma árvore binária deve conter:
  - Um campo DADOS
  - Um ponteiro para o Nodo Filho ESQUERDO
  - Um ponteiro para o Nodo Filho DIREITO
  - Um ponteiro para o Nodo PAI



Operações Básicas

- Existem várias operações possíveis com árvores binárias, porém as mais comuns são:
  - **info(p):** retorna o conteúdo do nó
  - **esquerdo(p)** ou **left(p)**: retorna o filho esquerdo de p ou NULL caso não exista filho esquerdo
  - **direito(p)** ou **right(p)**: retorna o filho direito de p ou NULL caso não exista filho direito
  - **pai(p)** ou **father(p)**: retorna o pai de **p ou NULL caso não exista pai**
  - **irmao(p) ou brother(p):** retorna o irmão de p ou NULL caso não exista irmão
  - **ehEsquerdo(p) ou isleft(p):** retorna TRUE se p é filho esquerdo do seu pai e FALSE caso contrário
  - **ehDireito(p) ou isright(p):** retorna TRUE se p é filho direito do seu pai e FALSE caso contrário

Operações Básicas

- As funções para retorno de filhos esquerdo e direito, pai e conteúdo são triviais:
  - def getInfo(p): **return** self.dado
  - def getLeft(p): **return** self.esq
  - def getRight(p): **return** self.dir
  - def getFather(p): **return** self.pai
    - Onde p é um ponteiro para um nodo da árvore!
- Mas, e as demais funções, como implementá-las?

Operações Básicas

- Utilizando as funções get*Left(p), getRight(p) e getFather(p) é possível escrever as demais funções:*
  - ***Função isleft(p): é filho esquerdo?***

```
...
q = getFather(p)
if  q== none:
   return false; #quando p aponta para raiz
if getLeft(q)==p:
   return true
return false
```

Pilha - Definição

- ***Função isright(p): é filho direito?***

```
...
q = getFather(p)
if q == null:
   return false #quando p aponta para raiz
If getRight(q)==p:
   return true
return false
```

Operações Básicas

- ***Função brother(p): retorna irmão de p***

```
...
if getFather(p) == null:
   return false #quando p aponta para raiz
if isleft(p):
   return getRight(getFather(p))
return getLeft(getFather(p))
```

Aplicações

- Uma árvore binária é útil quando há a necessidade de tomada de decisão bidirecinal (binária) em cada ponto de um processo
  - Exemplo:
    - Imagine que se deseje encontrar todas as repetições em uma lista de números. O número de comparações realizadas nesta tarefa pode ser bastante reduzida com a utilização de uma árvore. Como?

Resolvendo problema proposto...

- Ao ler o primeiro número da lista, crie uma árvore binária:
  - Este número é o nó raiz
- Ao ler o segundo número da lista, compare com a árvore,
  - Se for igual ao nó atual, temos uma repetição;
  - Se for maior que o nó atual, vá para o filho direito e repita a comparação;
  - Se for menor que o nó atual, vá para o filho da esquerda e repita a comparação

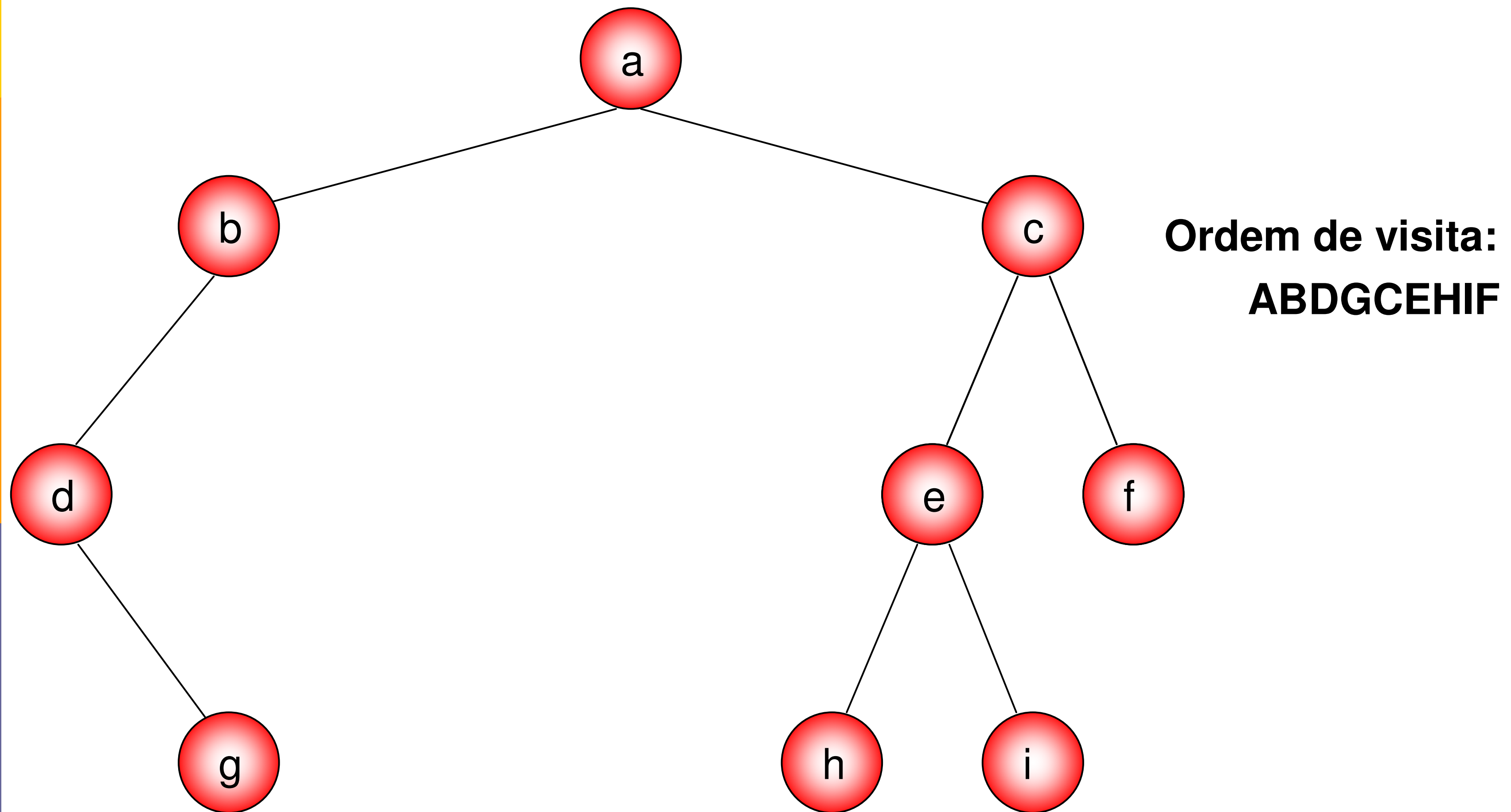
Percorrendo uma Árvore

- Operação de **percorrer uma árvore:**
  - **É o ato de caminhar sobre a árvore enumerando cada um dos seus nós uma vez**
    - **É dito visitar um nó a medida que ele é enumerado**
    - **Não existe uma ordem natural para se visitar os nós de uma árvore! É possível citar três métodos:**
      - **Pré-ordem ou profundidade**
      - **Em ordem ou ordem simétrica**
      - **Pós-ordem**

Pré-Ordem ou Percurso em Profundidade

1. Visitamos a raiz;
2. Visitamos a sub-árvore **esquerda em ordem prévia;**
3. **Percorremos a sub-árvore direita em ordem prévia;**

Exemplo: Pré-Ordem



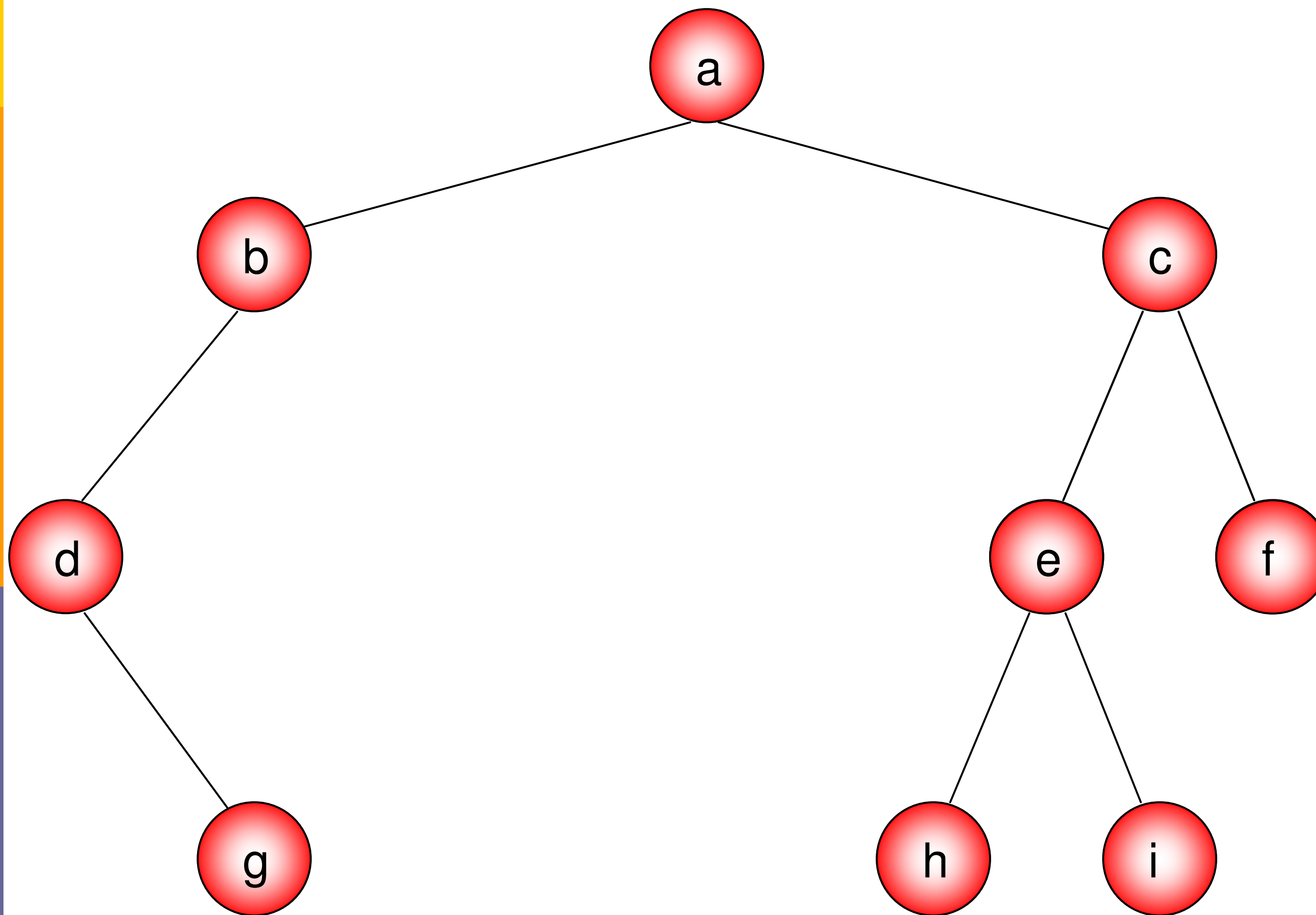
Ordem de visita:

ABDGCEHIF

Em Ordem ou Ordem Simétrica

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

Exemplo: Em Ordem



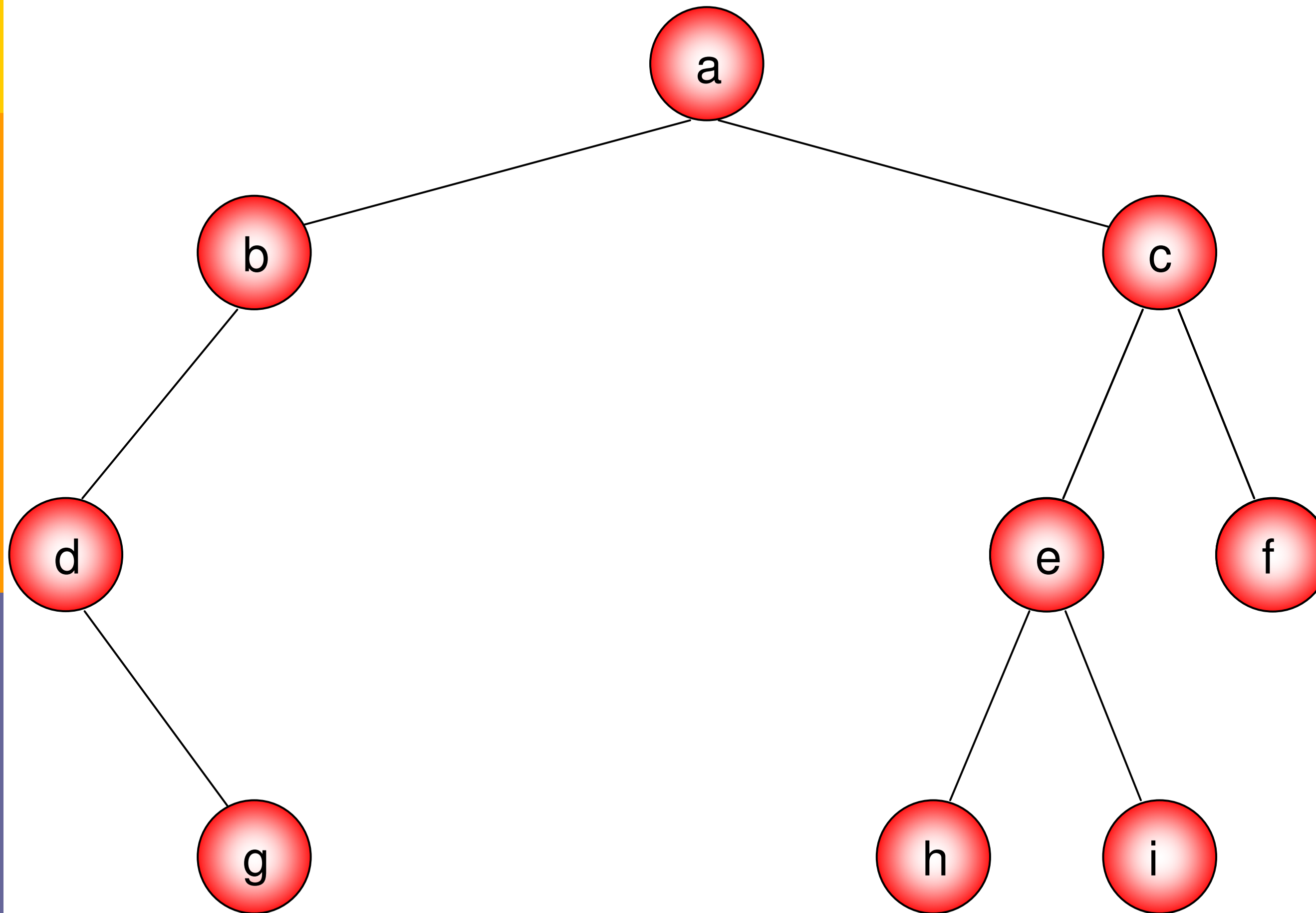
Ordem de visita:

DGBAHEICF

Pós-Ordem

1. Percorre-se a sub-árvore esquerda em ordem posterior;
2. Percorre-se a sub-árvore direita em ordem posterior;
3. Visita-se a raiz.

Exemplo: Pós-Ordem



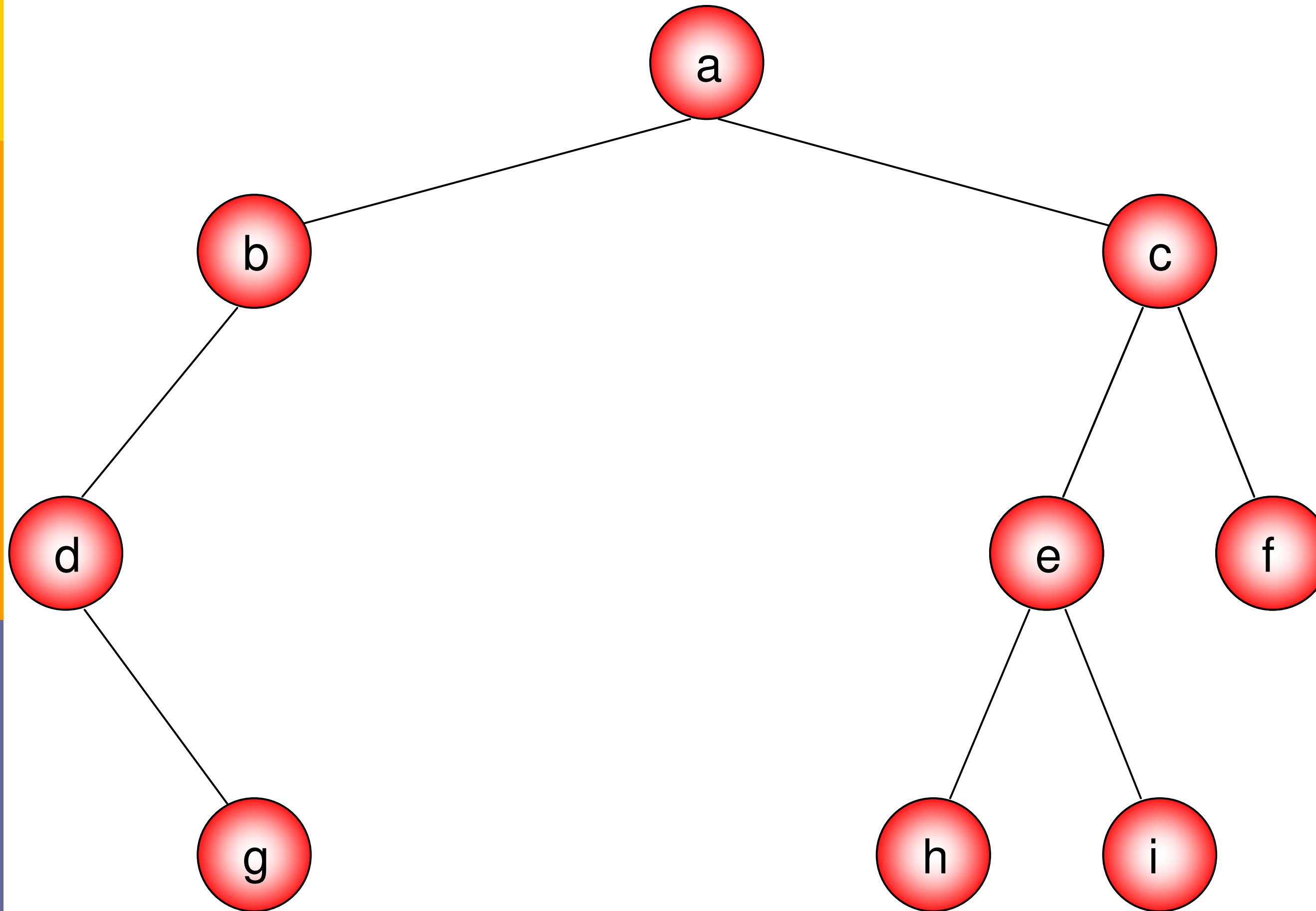
Ordem de visita:

GDBHIEFCA

Largura

- Existe ainda um outro método para se percorrer uma árvore chamado **Em Largura:**
  1. **Visita-se a raiz;**
  2. **Para todos os demais níveis, visita-se todo os nós do nível da esquerda para a direita**

Exemplo em Largura



Ordem de visita:

ABCDEFGHI

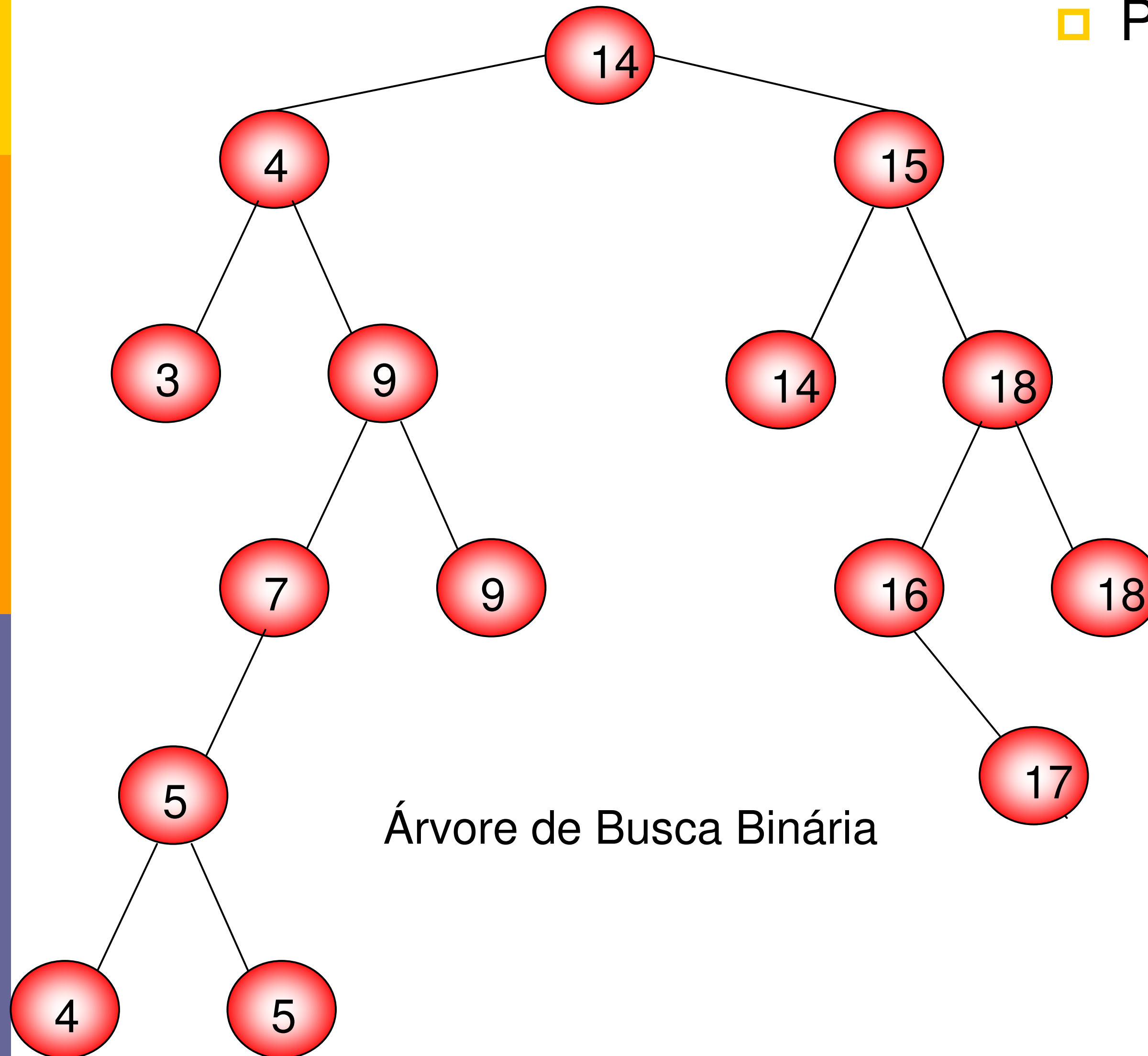
Exemplos de Aplicações

- Alguns algoritmos de árvore binárias primeiro criam a árvore e depois percorrem-na.
- Imagine que uma lista de números em uma arquivo
  - Quer-se exibir os números em ordem crescente

Exemplo de Aplicações

- Imagine a lista:
  - 14, 15, 4, 9, 7, 18, 3, 5, 16, 4, 20, 17, 9, 14, 5
  - Vamos construir uma árvore binária de tal forma que,
    - Se o número for menor que o nó atual, este vai para a ramificação esquerda;
    - Se o número é maior ou igual vai para a ramificação direita

Exemplo de Aplicações



Árvore de Busca Binária

- Propriedade:
  - Todos os elementos da sub-árvore esqueda de um nó **n são menores que o conteúdo de n.**
  - **Todos os elementos da sub-árvore direita de um nó n são maiores ou igauis ao conteúdo de n.**

Exercícios Práticos

- Exercício 1: Implementar uma classe árvore binária

- Exercício 2: Implemente a aplicação proposta para verificação de números repetidos em uma lista.