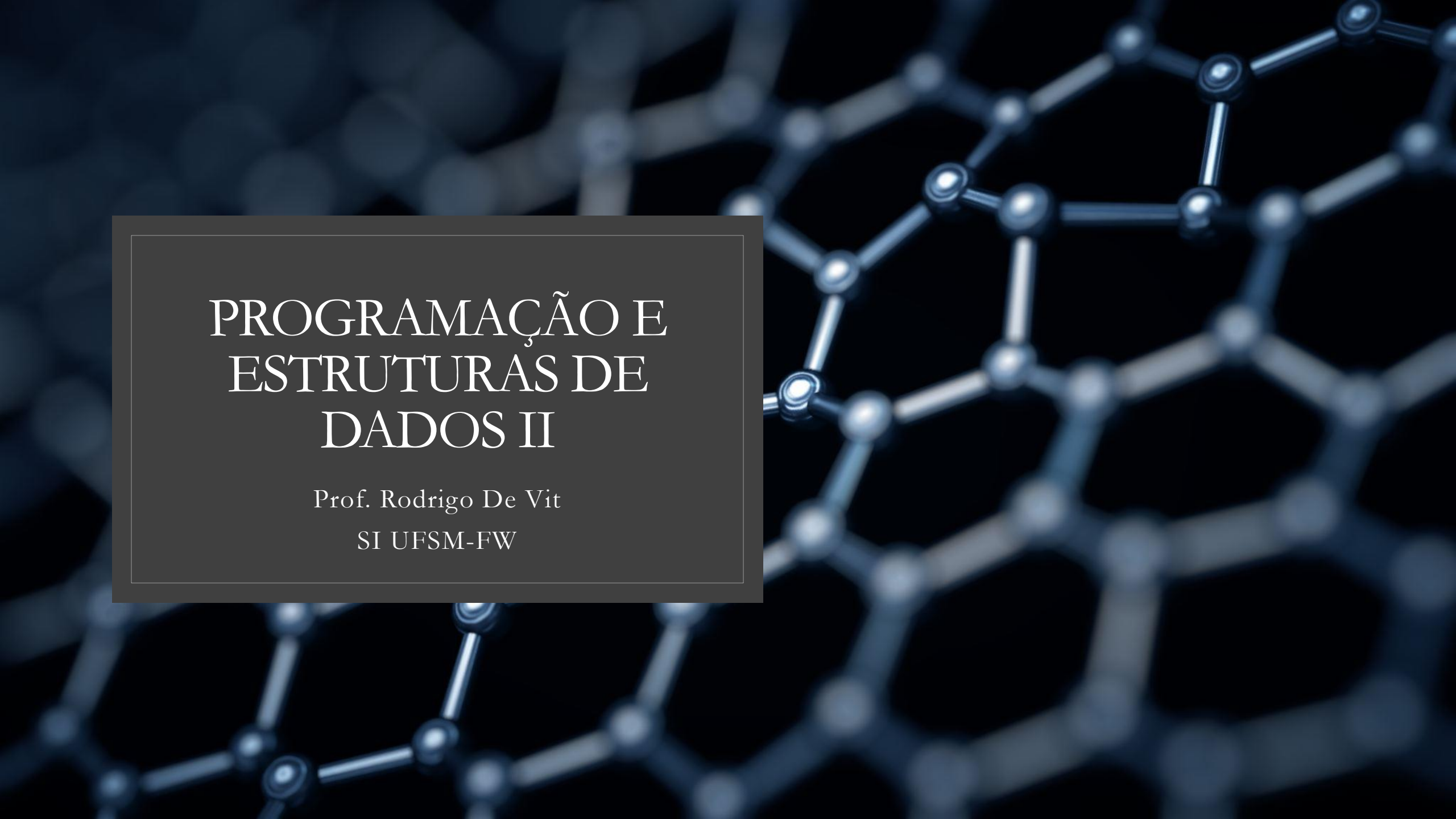


PROGRAMAÇÃO E ESTRUTURAS DE DADOS II

Prof. Rodrigo De Vit

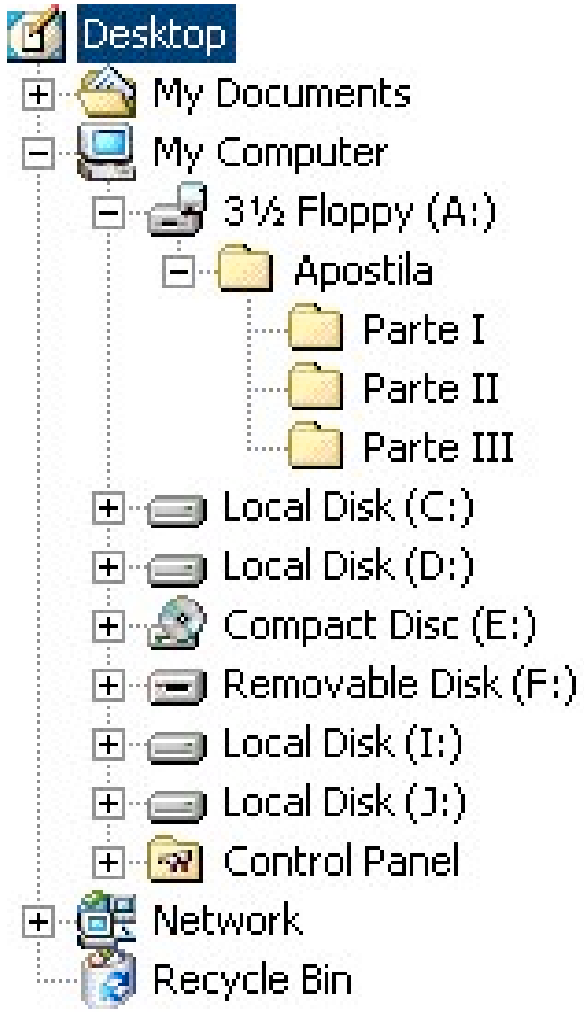
SI UFSM-FW

Revisão: P1

- Gabarito P1.

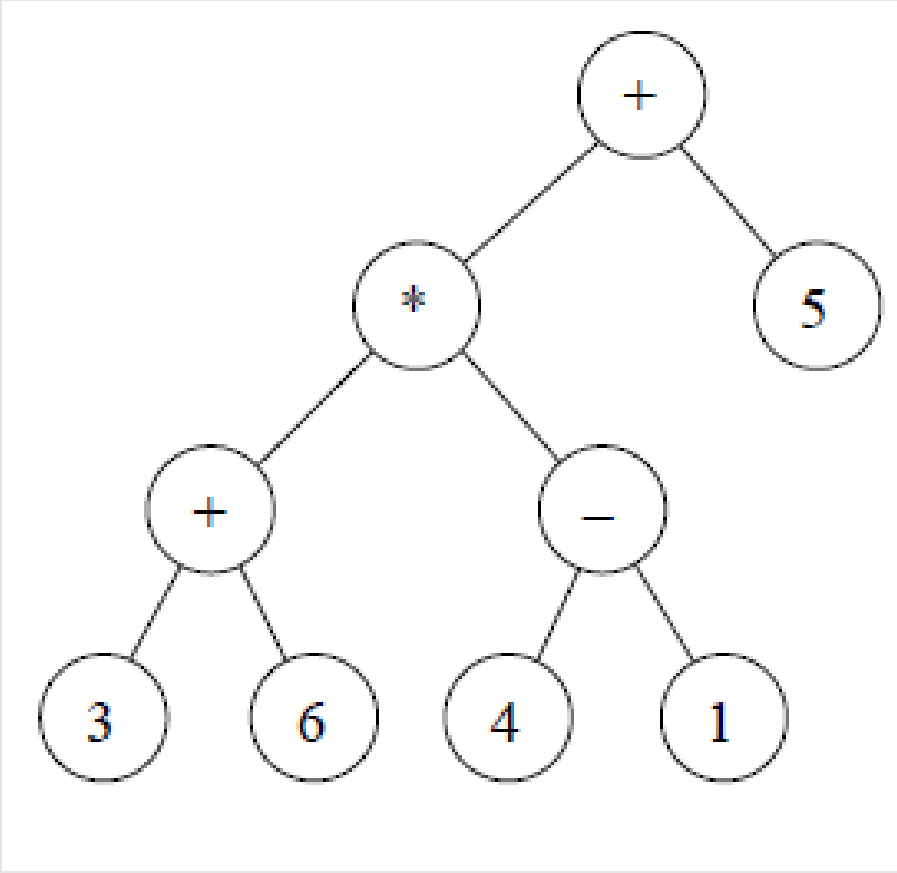
Árvores

- Nos capítulos anteriores examinamos as estruturas de dados que podem ser chamadas de unidimensionais ou lineares, como vetores e listas. A importância dessas estruturas é inegável, mas elas não são adequadas para representarmos dados que devem ser dispostos de maneira hierárquica. Por exemplo, os arquivos (documentos) que criamos num computador são armazenados dentro de uma estrutura hierárquica de diretórios (pastas).
- A forma mais natural para definirmos uma estrutura de árvore é usando recursividade. Uma árvore é composta por um conjunto de nós. Existe um nó r, denominado *raiz*, que contém zero ou mais sub-árvores, cujas raízes são ligadas diretamente a r. Esses nós raízes das sub-árvores são ditos *filhos* do nó *pai*, r.
- O número de filhos permitido por nó e as informações armazenadas em cada nó diferenciam os diversos tipos de árvores existentes.

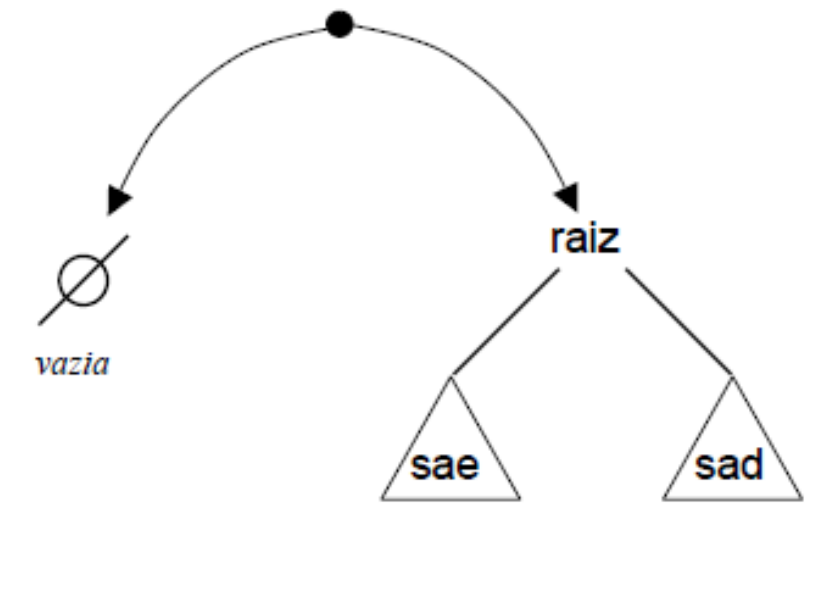


Árvores binárias

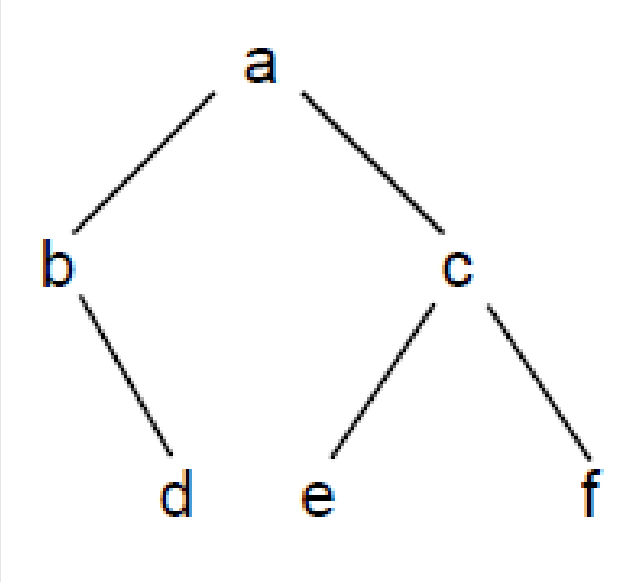
- Um exemplo de utilização de árvores binárias está na avaliação de expressões. Como trabalhamos com operadores que esperam um ou dois operandos, os nós da árvore para representar uma expressão têm no máximo dois filhos. Nessa árvore, os nós folhas representam operandos e os nós internos operadores. Uma árvore que representa, por exemplo a expressão (3+6)*(4-1)+5 é ilustrada ao lado.
- Numa árvore binária, cada nó tem zero, um ou dois filhos. De maneira recursiva, podemos definir uma árvore binária como sendo:
  - uma árvore vazia; ou
  - um nó raiz tendo duas sub-árvores, identificadas como a sub-árvore da direita (*sad*) e a sub-árvore da esquerda (*sae*).



Árvores binárias



- A Figura 13.4 ilustra a definição de árvore binária. Essa definição recursiva será usada na construção de algoritmos, e na verificação (informal) da correção e do desempenho dos mesmos.



- A Figura 13.5 ilustra uma estrutura de árvore binária. Os nós *a*, *b*, *c*, *d*, *e*, *f* formam uma árvore binária da seguinte maneira: a árvore é composta do nó *a*, da subárvore à esquerda formada por *b* e *d*, e da sub-árvore à direita formada por *c*, *e* e *f*. O nó *a* representa a raiz da árvore e os nós *b* e *c* as raízes das sub-árvores. Finalmente, os nós *d*, *e* e *f* são folhas da árvore.

Árvores binárias

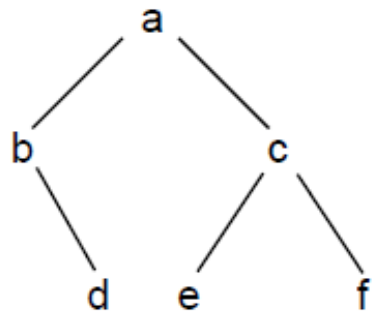


Figura 13.5: Exemplo de árvore binária

Para descrever árvores binárias, podemos usar a seguinte notação textual: a árvore vazia é representada por <>, e árvores não vazias por <*raiz sae sad*>. Com essa notação, a árvore da Figura 13.5 é representada por: <a<b<><d<><>>><c<e<><>><f<><>>>>.

Pela definição, uma sub-árvore de uma árvore binária é sempre especificada como sendo a *sae* ou a *sad* de uma árvore maior, e qualquer das duas sub-árvores pode ser vazia. Assim, as duas árvores da Figura 13.6 são distintas.

Figura 13.6: Duas árvores binárias distintas.

Isto também pode ser visto pelas representações textuais das duas árvores, que são, respectivamente: <a <b<><>> <> > e <a <> <b<><>> >.

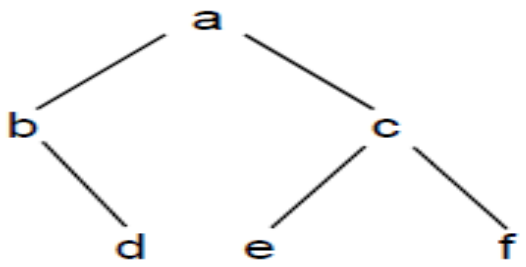


Figura 13.5: Exemplo de árvore binária

Figura 13.6: Duas árvores binárias distintas.

Árvores binárias

- Uma propriedade fundamental de todas as árvores é que só existe um caminho da raiz para qualquer nó. Com isto, podemos definir a *altura* de uma árvore como sendo o comprimento do caminho mais longo da raiz até uma das folhas. Por exemplo, a altura da árvore da Figura 13.5 é 2, e a altura das árvores da Figura 13.6 é 1. Assim, a altura de uma árvore com um único nó raiz é zero e, por conseguinte, dizemos que a altura de uma árvore vazia é negativa e vale -1.

Representação em C

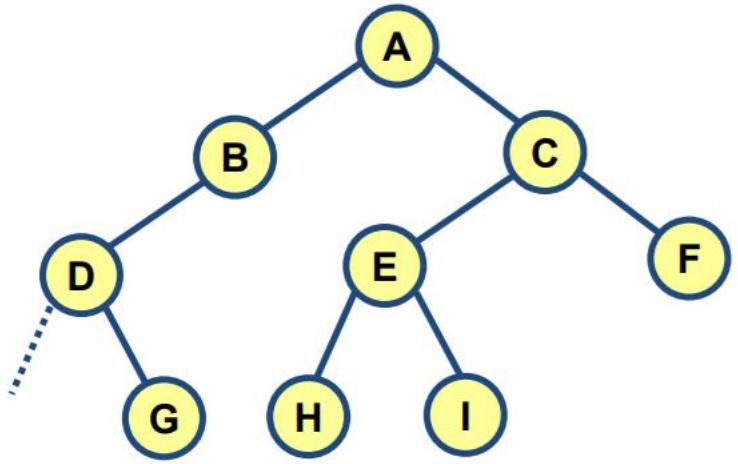
- Árvore binária em C.
- Enxerto e poda em árvores. (vide `main()`)

Ordens de percurso em árvores binárias

- A programação da operação imprime, vista anteriormente, seguiu a ordem empregada na definição de árvore binária para decidir a ordem em que as três ações seriam executadas. Entretanto, dependendo da aplicação em vista, esta ordem poderia não ser a preferível, podendo ser utilizada uma ordem diferente desta, por exemplo:
  - `imprime(a->esq); /* mostra sae */`
  - `imprime(a->dir); /* mostra sad */`
  - `printf("%c ", a->info); /* mostra raiz */`
- Muitas operações em árvores binárias envolvem o percurso de todas as sub-árvores, executando alguma ação de tratamento em cada nó, de forma que é comum percorrer uma árvore em uma das seguintes ordens:
  - *pré-ordem*: trata *raiz*, percorre *sae*, percorre *sad*;
  - *ordem simétrica*: percorre *sae*, trata *raiz*, percorre *sad*;
  - *pós-ordem*: percorre *sae*, percorre *sad*, trata *raiz*.

Ordens de percurso em árvores binárias

- **Exemplos**



Pré-ordem: ABDGCEHIF
Ordem: DGBAHEICF
Pós-ordem: GDBHIEFCA

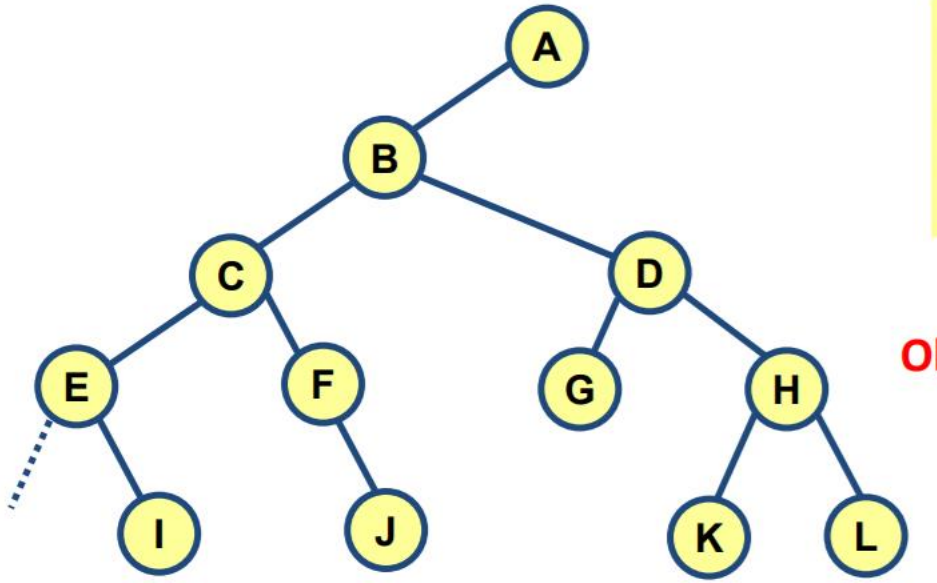
Observação

No percurso em ordem, "D" é o limite da subárvore esquerda, por isso é o ponto de partida. Na ausência de um filho esquerdo, a sequência de leitura manda ler a raiz.

- *pré-ordem*: trata *raiz*, percorre *sae*, percorre *sad*;
- *ordem simétrica*: percorre *sae*, trata *raiz*, percorre *sad*;
- *pós-ordem*: percorre *sae*, percorre *sad*, trata *raiz*.

Ordens de percurso em árvores binárias

- **Exemplos**



Pré-ordem: ABCEIFJDGHKL
Ordem: EICFJBGDKHLA
Pós-ordem: IEJFCGKLHDBA

Observação

No percurso em ordem, "E" é o limite da subárvore esquerda, por isso é o ponto de partida. Na ausência de um filho esquerdo, a sequência de leitura manda ler a raiz.

- *pré-ordem*: trata *raiz*, percorre *sae*, percorre *sad*;
- *ordem simétrica*: percorre *sae*, trata *raiz*, percorre *sad*;
- *pós-ordem*: percorre *sae*, percorre *sad*, trata *raiz*.