

Algoritmos e Estruturas de Dados I

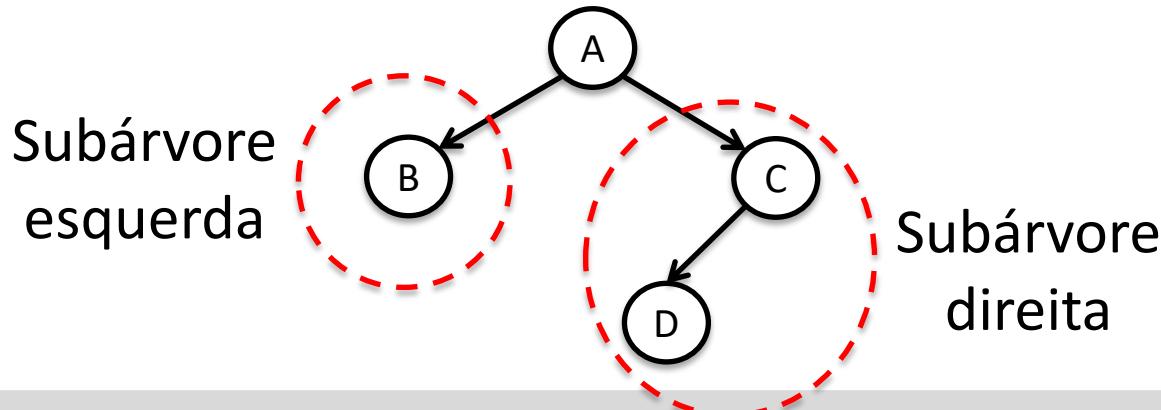
Aula 10: Árvores Binárias

Prof. Márcio Porto Basgalupp

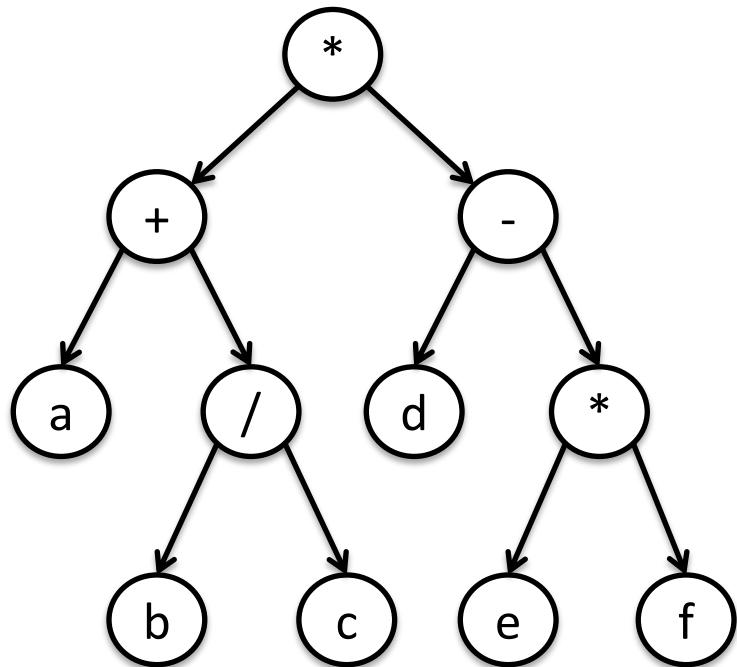
créditos: Prof. Jurandy G. Almeida Jr.

Universidade Federal de São Paulo
Departamento de Ciência e Tecnologia

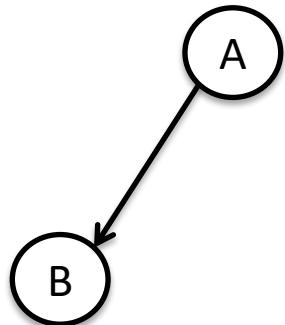
- **Árvores binárias** são árvores orientadas de grau 2
- Uma árvore binária é uma estrutura que é ou vazia ou possui 3 componentes:
 - Uma raiz
 - Uma subárvore esquerda
 - Uma subárvore direita
- As subárvore devem ser árvores binárias



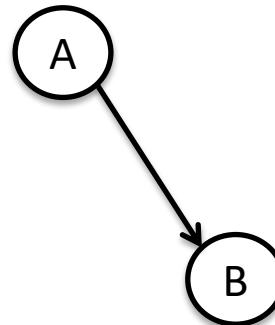
- Podemos, por exemplo, representar uma expressão aritmética (com operadores binários) por meio de uma AB, na qual cada operador é um nó da árvore e seus dois operandos representados como subárvore
- A árvore ao lado representa a expressão $(a+b/c)*(d-e*f)$



- As duas árvores binárias seguintes são distintas:
 - (i) a primeira tem subárvore direita vazia
 - (ii) a segunda tem subárvore esquerda vazia



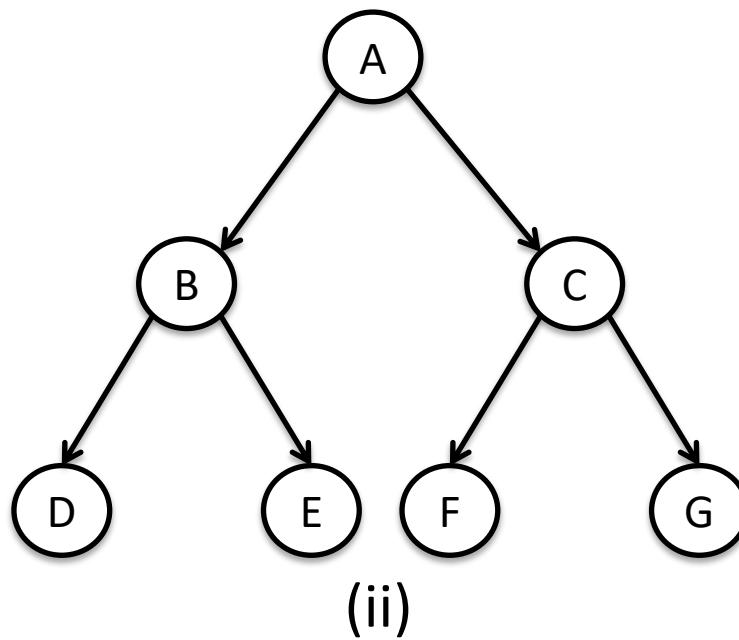
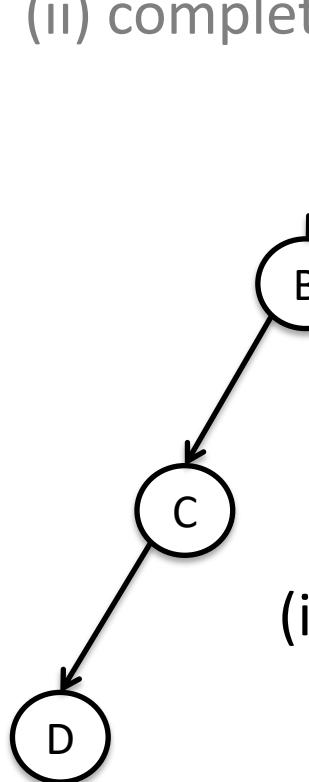
(i)



(ii)

- Exemplos de árvores binárias:

- (i) assimétrica à esquerda (degenerada)
- (ii) completa



- O número máximo de nós em uma árvore binária de altura **h** é dado por:

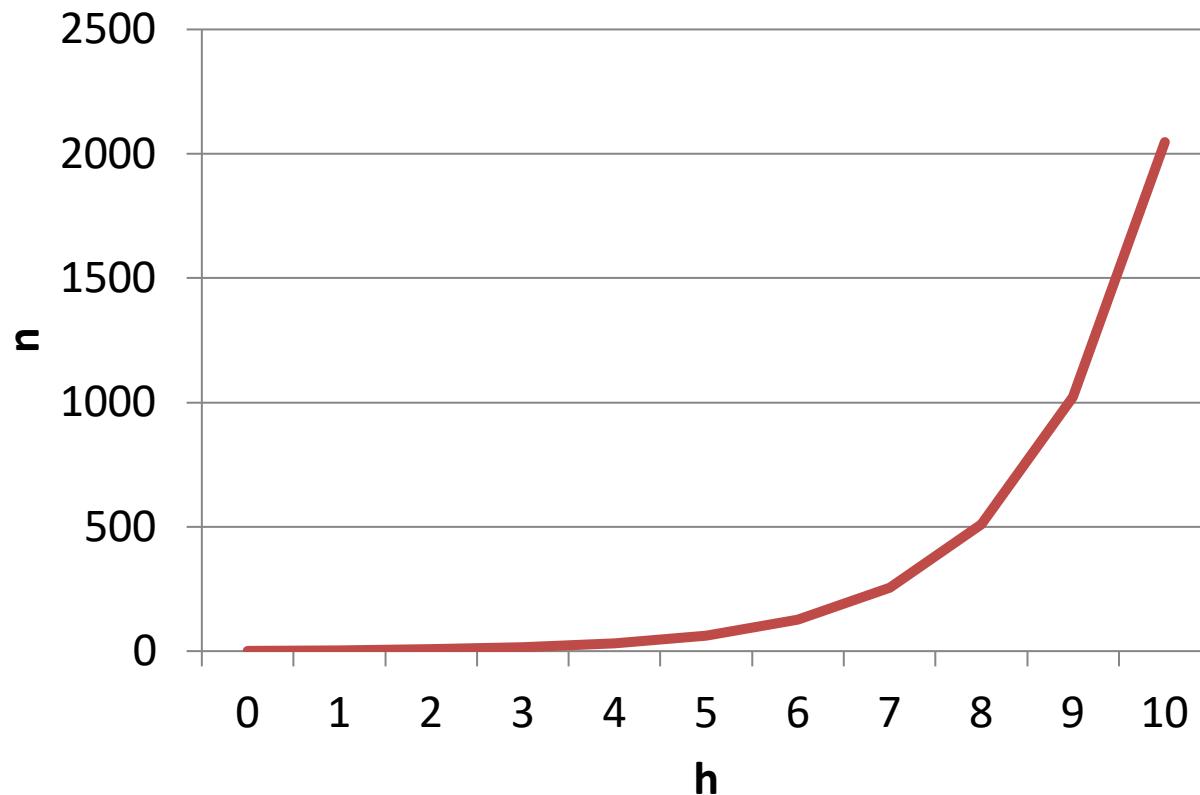
$$n = n(h, 2) = \sum_{i=0}^h 2^i = 2^{h+1} - 1$$

- Portanto, **n** elementos podem ser organizados em uma árvore binária de altura mínima $\approx \log_2 n$

$$h = \log_2(n + 1) - 1$$

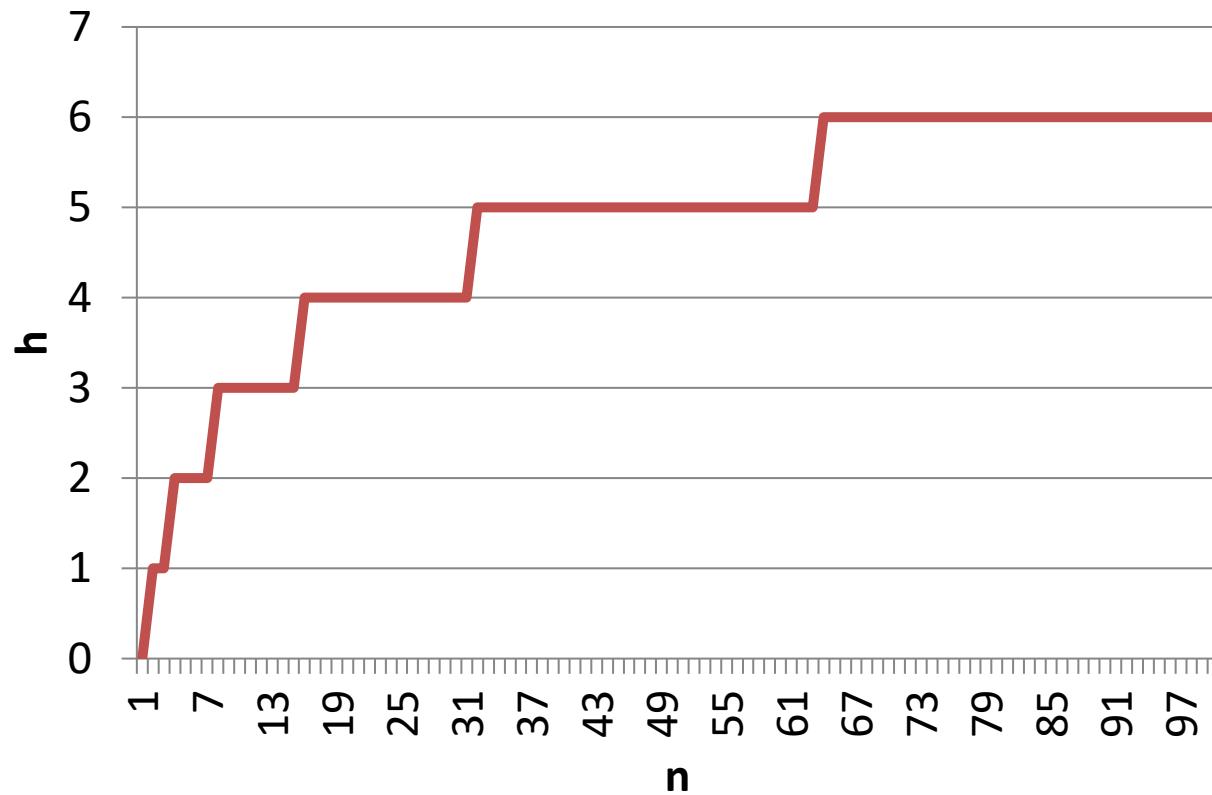
Árvore Binária de Altura Mínima

h	n
0	1
1	3
2	7
3	15
4	31
5	63
6	127
7	255
8	511
9	1023
10	2047



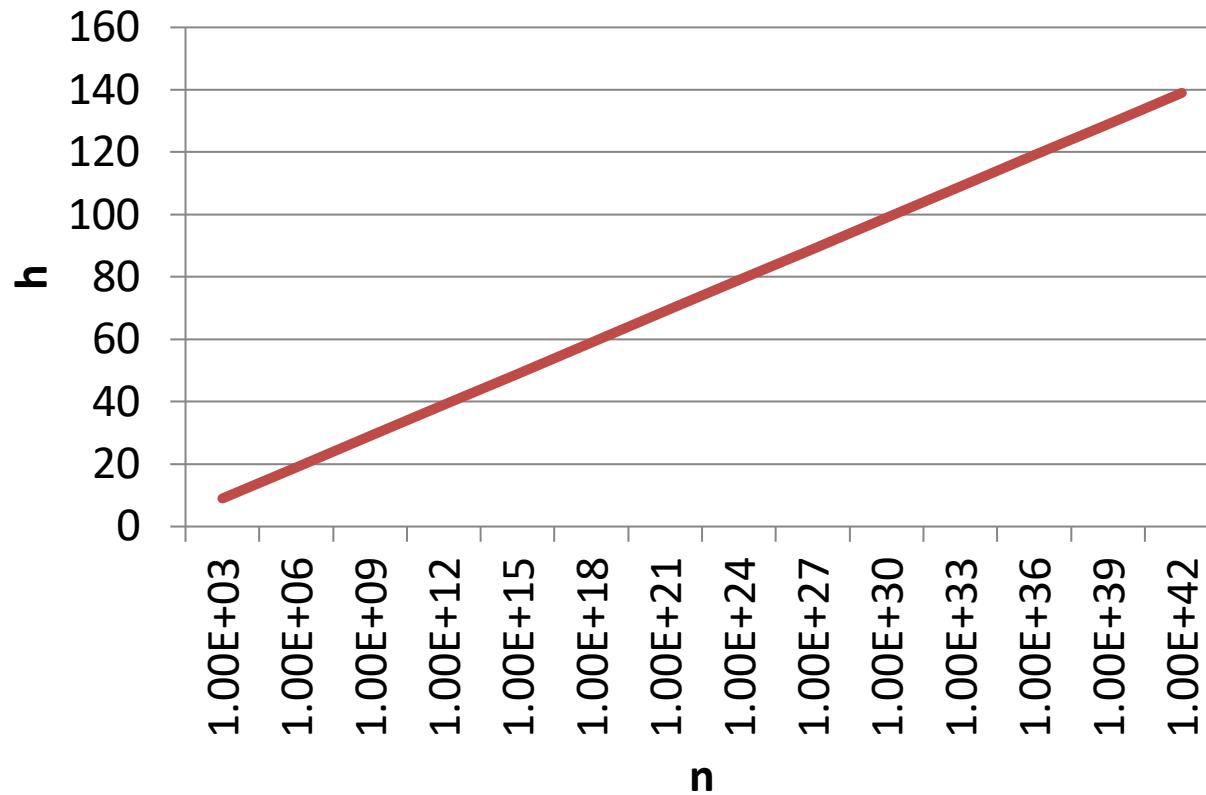
Árvore Binária de Altura Mínima

n	h
1	0
2	1
3	1
4	2
5	2
6	2
7	2
8	3
9	3
10	3
11	3
12	3
13	3
14	3
15	3
16	4



Árvore Binária de Altura Mínima

n	h
1.E+03	9
1.E+06	19
1.E+09	29
1.E+12	39
1.E+15	49
1.E+18	59
1.E+21	69
1.E+24	79
1.E+27	89
1.E+30	99
1.E+33	109
1.E+36	119
1.E+39	129
1.E+42	139



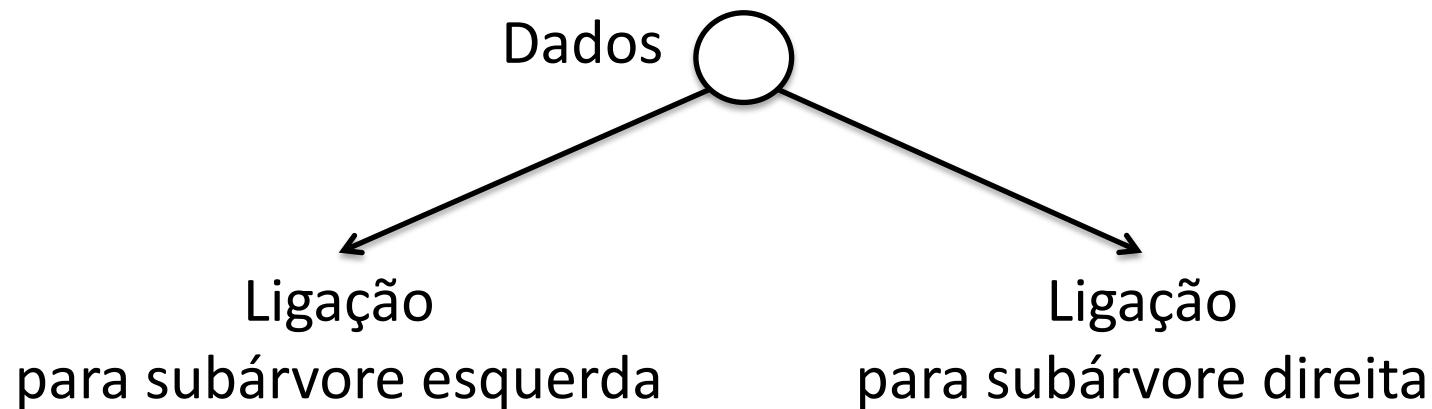
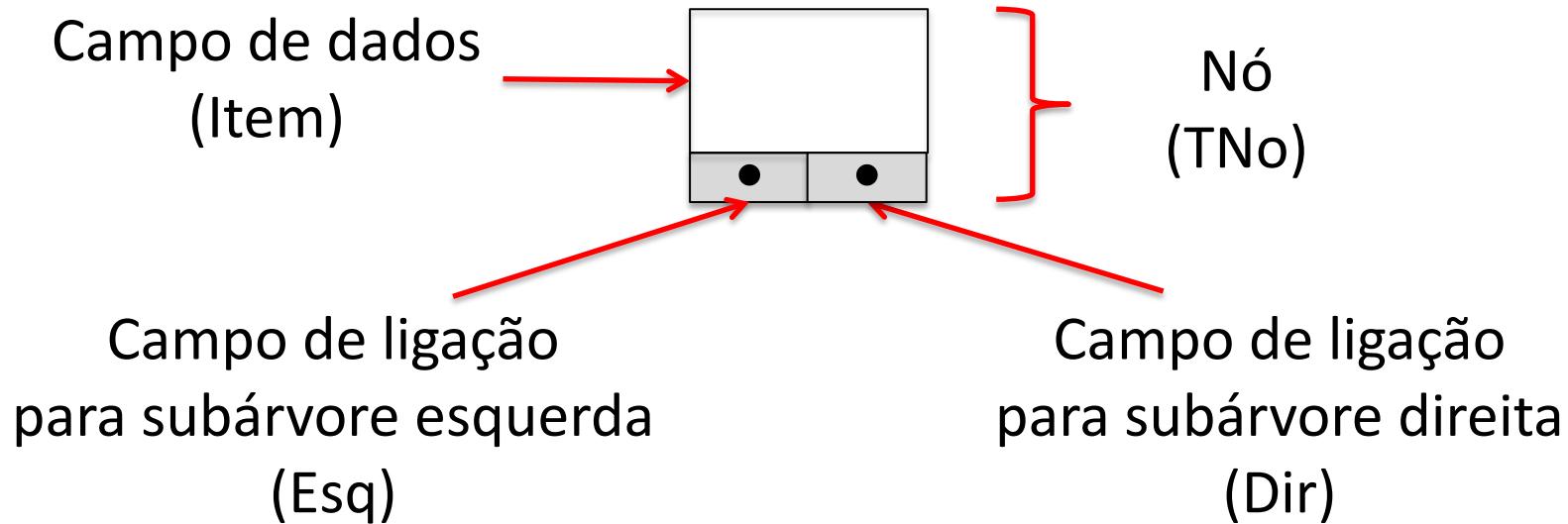
■ Conjunto de Operações

- **TArvBin_Inicia()**: Inicia uma árvore binária vazia
- **TArvBin_CriaNo(x, Esq, Dir)**: Retorna um apontador para um nó contendo o item *x* e as subárvore *Esq* e *Dir*
- **TArvBin_EhFolha(ArvBin)**: Retorna *true* se a árvore não possuir nenhum descendente; caso contrário, retorna *false*
- **TArvBin_SubarvoreEsquerda(ArvBin)**: Retorna a subárvore esquerda da raiz da árvore
- **TArvBin_SubarvoreDireita(ArvBin)**: Retorna a subárvore direita da raiz da árvore
- **TArvBin_Libera(ArvBin)**: Libera a memória ocupada pela árvore

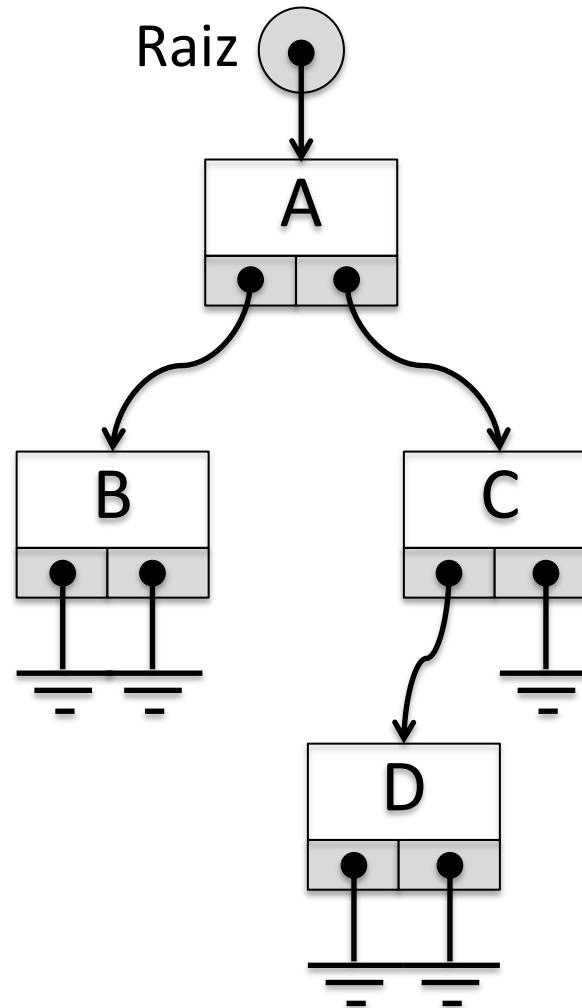
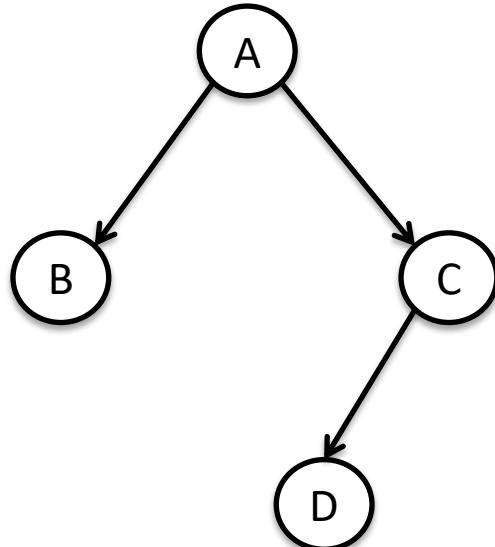
- Existem várias opções de estruturas de dados que podem ser usadas para representar árvores binárias
- Uma das representações mais utilizadas é:
 - Implementação por meio de **apontadores**

- É natural a implementação de árvores binárias por meio de apontadores
- Como toda árvore possui uma raiz, uma árvore vazia pode ser representada por um apontador aterrado (NULL em C/C++)
- Cada nó em uma árvore binária possui um campo de dados, um apontador para a subárvore esquerda e um apontador para a subárvore direita

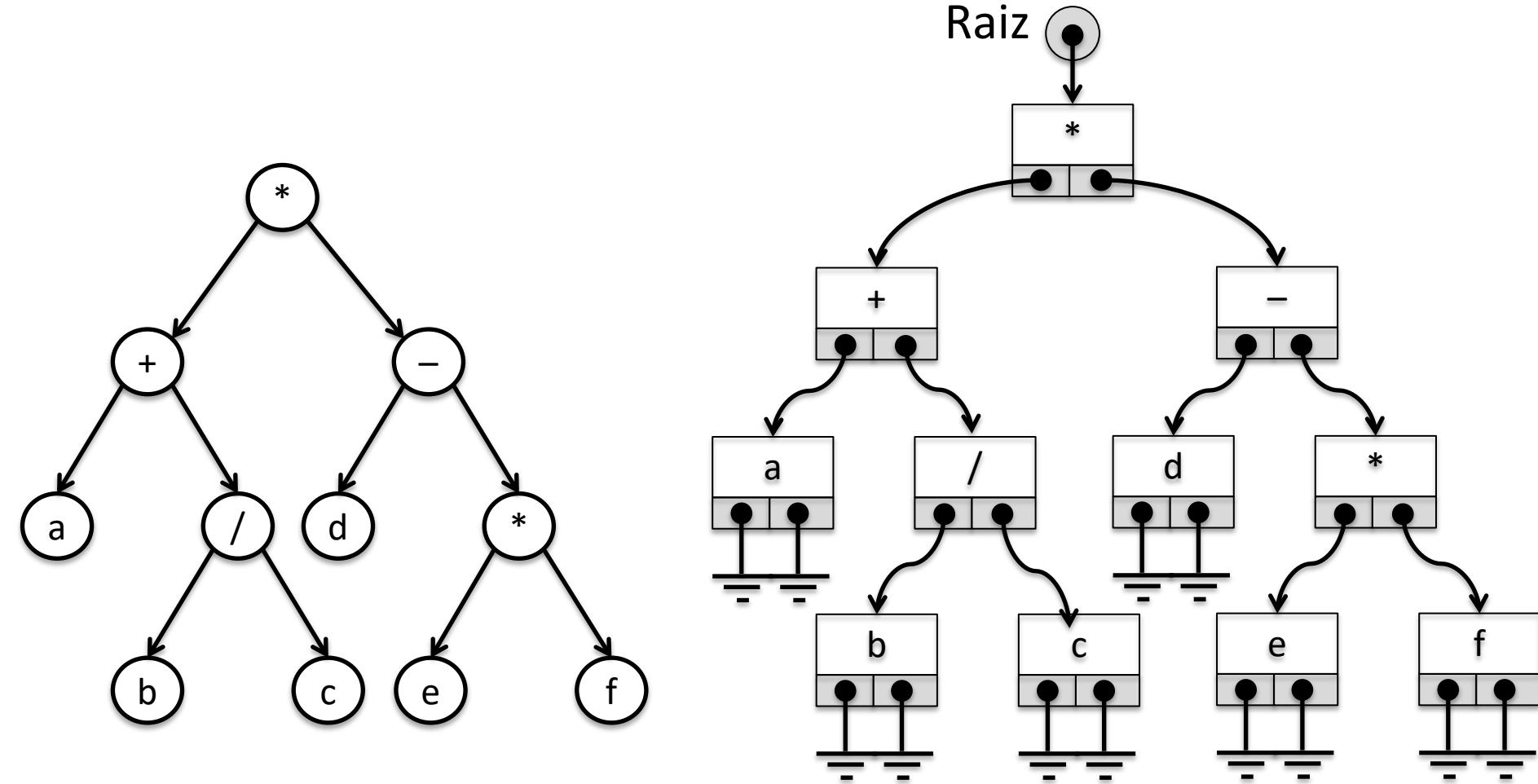
Implementação por Apontadores



Implementação por Apontadores



Implementação por Apontadores



Estrutura da Árvore Binária

```
typedef int TChave;

typedef struct {
    TChave Chave;
    /* outros componentes */
} TItem;

typedef struct SNo *TArvBin;

typedef struct SNo {
    TItem Item;
    TArvBin Esq, Dir;
} TNo;
```

Estrutura da Árvore Binária

```
/* procedimentos e funcoes do TAD */  
TArvBin TArvBin_Inicia();  
TArvBin TArvBin_CriaNo(TItem x, TArvBin Esq, TArvBin Dir);  
int TArvBin_EhFolha(TArvBin No);  
TArvBin TArvBin_SubarvoreEsquerda(TArvBin No);  
TArvBin TArvBin_SubarvoreDireita(TArvBin No);  
void TArvBin_Libera(TArvBin No);
```

Operações da Árvore Binária

```
TNo* TArvBin_Inicia(TItem x) { /* cria árvore com a raiz */  
    TNo* arv;  
    No = (TNo*) malloc(sizeof(TNo));  
    return No;  
}  
  
TNo* TArvBin_CriaNo(TItem x, TArvBin Esq, TArvBin Dir) {  
    TArvBin No;  
  
    No = (TArvBin) malloc(sizeof(TNo));  
    No->Item = x;  
    No->Esq = Esq;  
    No->Dir = Dir;  
  
    return No;  
}  
  
int TArvBin_EhFolha(TNo *No)  
{  
    return ((No == NULL) || ((No->Esq == NULL) && (No->Dir == NULL)));  
}
```

Operações da Árvore Binária

```
TArvBin TArvBin_SubarvoreEsquerda(TNo *No) {  
    return No->Esq;  
}  
  
TArvBin TArvBin_SubarvoreDireita(TNo *No) {  
    return No->Dir;  
}  
  
void TArvBin_Libera(TNo *No) {  
    if (No != NULL) {  
        TArvBin_Libera(No->Esq);  
        TArvBin_Libera(No->Dir);  
        free(No);  
    }  
}  
  
TNo* TArvBin_InsereEsq (TNo *Arvore, TItem x) { /* análogo para filho à direita */  
    TNo *No;  
  
    No = (TNo*) malloc(sizeof(TNo));  
    No->Item = x;  
    No->Esq = Esq; /* se filho à */  
    return No;  
}
```

Operações da Árvore Binária

```
TArvBin TArvBin_SubarvoreEsquerda(TNo *No) {
    return No->Esq;
}

TArvBin TArvBin_SubarvoreDireita(TNo *No) {
    return No->Dir;
}

void TArvBin_Libera(TNo *No) {
    if (No != NULL) {
        TArvBin_Libera(No->Esq);
        TArvBin_Libera(No->Dir);
        free(No);
    }
}

TNo* TArvBin_InsereEsq (TNo *No, TItem x) { /* análogo para filho à direita */
    No->Esq = (TNo*) malloc(sizeof(TNo));
    No->Esq->Item = x;
    return No;
}
```

Operações da Árvore Binária

```
TArvBin TArvBin_Inicia()
{
    return NULL;
}

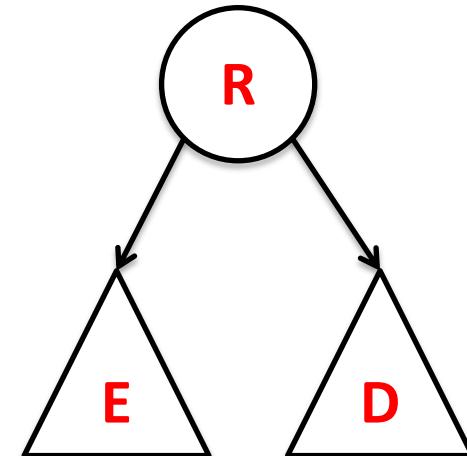
TArvBin TArvBin_CriaNo(TItem x, TArvBin Esq, TArvBin Dir)
{
    TArvBin No;

    No = (TArvBin) malloc(sizeof(TNo));
    No->Item = x;
    No->Esq = Esq;
    No->Dir = Dir;

    return No;
}

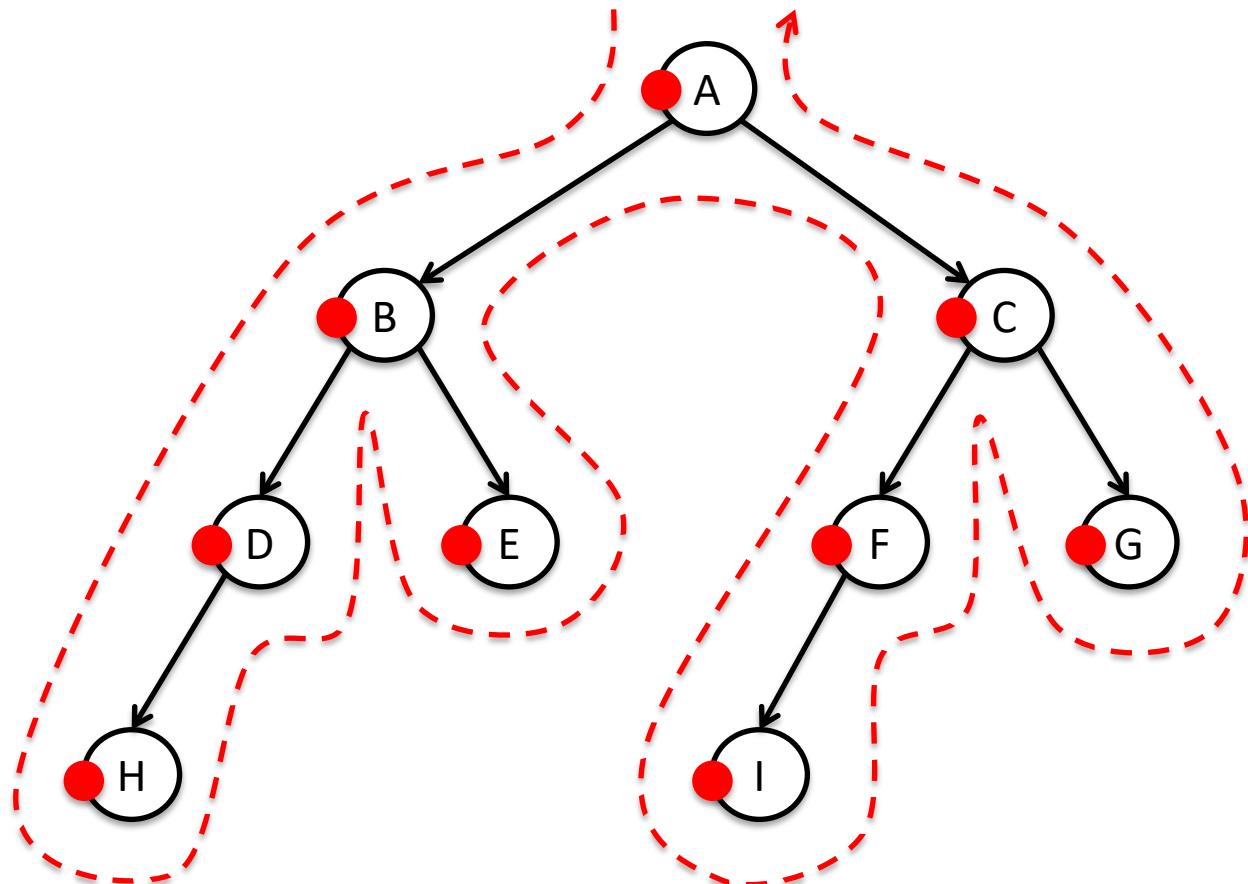
int TArvBin_EhFolha(TArvBin No)
{
    return ((No == NULL) || ((No->Esq == NULL) && (No->Dir == NULL)));
}
```

- Seja uma AB em que **R** denota sua raiz, **E** e **D** denotam as subárvores esquerda e direita, respectivamente
- Os nós de uma AB podem ser visitados de três formas (**varredura** da árvore):
 - Pré-ordem (*pre-order*): **R, E, D**
 - visitar a raiz antes das subárvores
 - Em-ordem (*in-order*): **E, R, D**
 - Pós-ordem (*post-order*): **E, D, R**
 - visitar a raiz após as subárvores



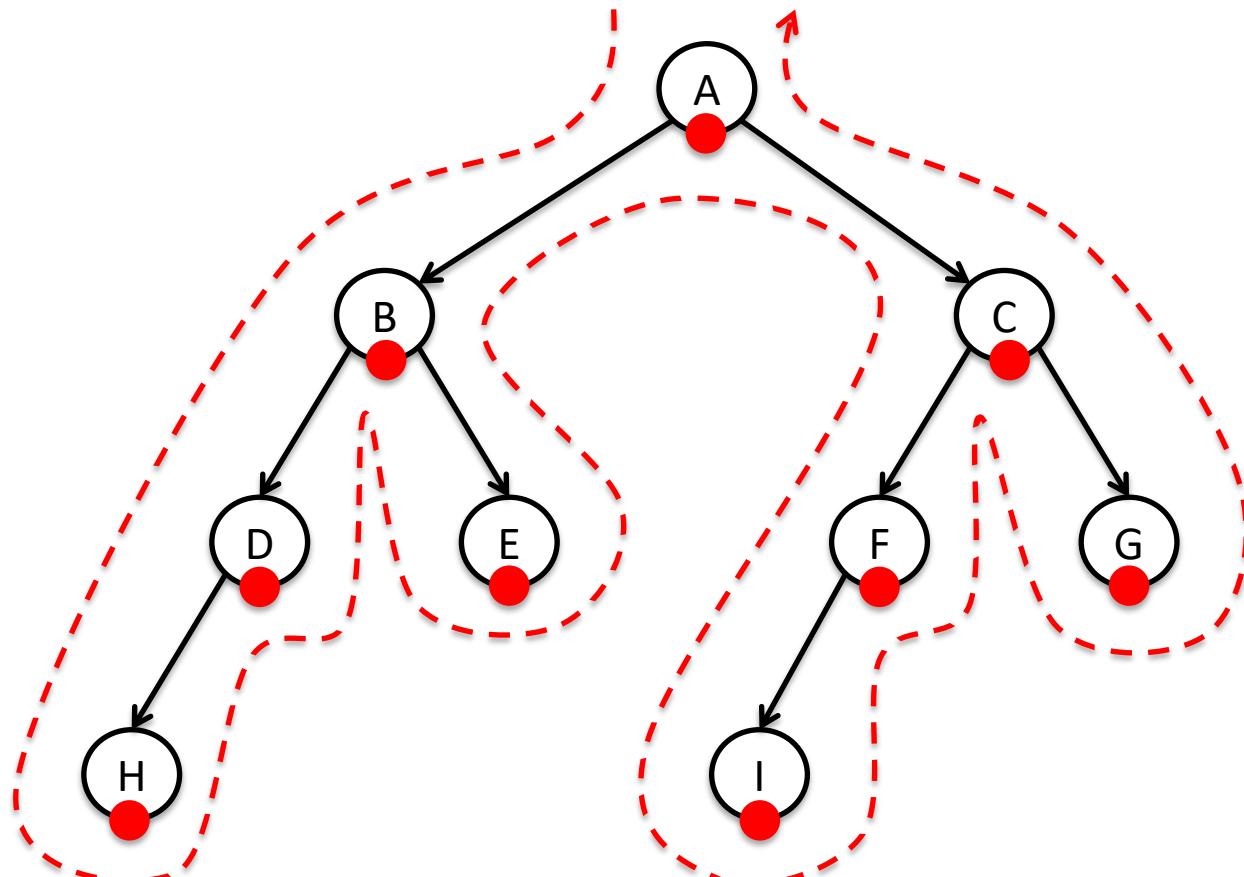
Percorso em Árvores Binárias

- Pré-ordem: A, B, D, H, E, C, F, I, G



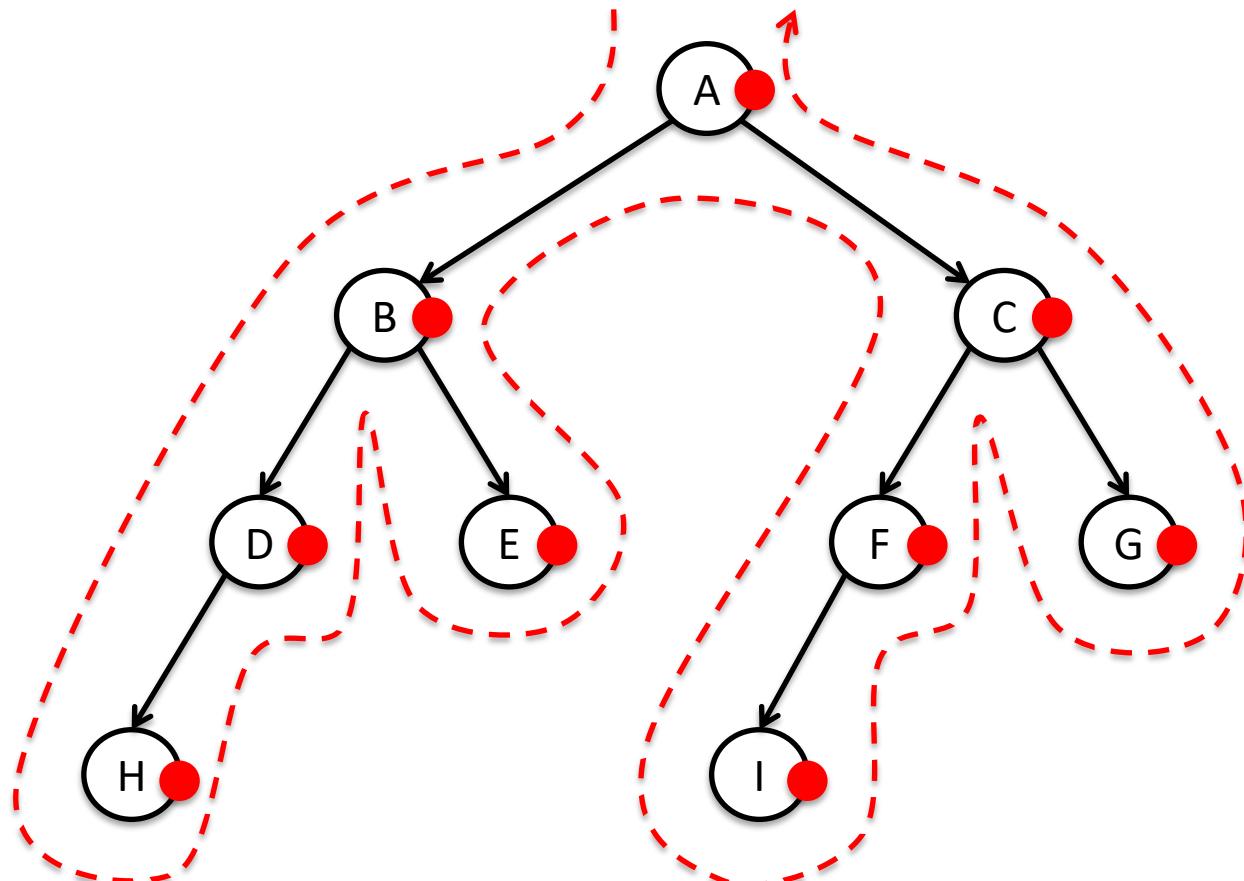
Percorso em Árvores Binárias

- Em-ordem: H, D, B, E, A, I, F, C, G



Percorso em Árvores Binárias

- Pós-ordem: H, D, E, B, I, F, G, C, A



Percorso em Árvores Binárias

- Pré-ordem

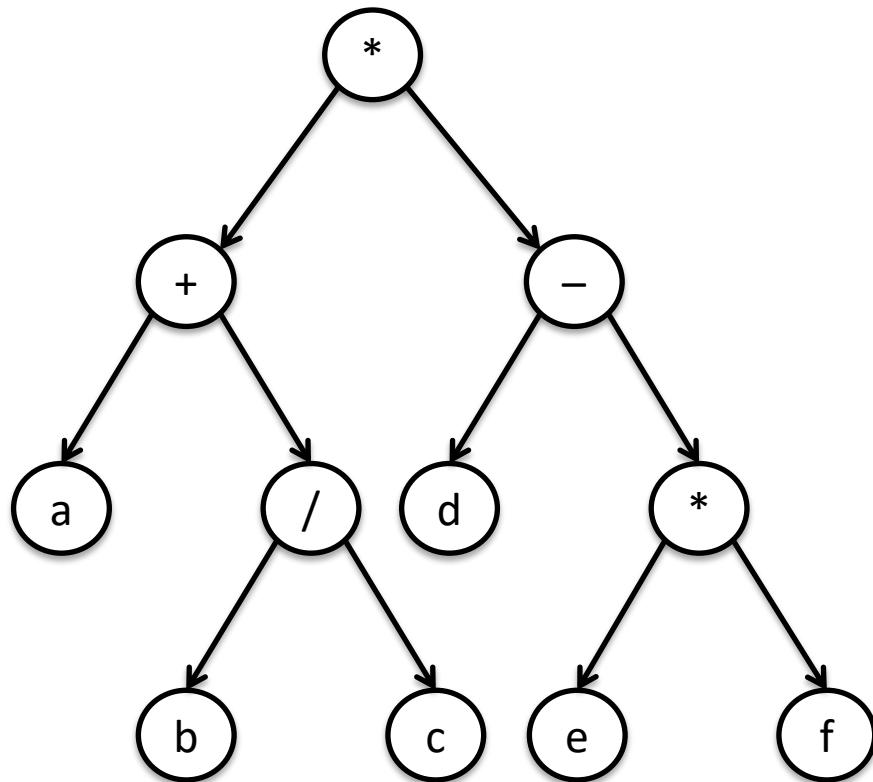
- $* + a / b c - d * e f$

- Em-ordem

- $a + b / c * d - e * f$

- Pós-ordem

- $a b c / + d e f * - *$



Para percorrer uma AB em uma dada ordem, assume-se que existe um procedimento (ou método) chamado

```
void processa(TItem x)
```

que efetua algum tipo de processamento com o item **x** passado como parâmetro, lembrando que **TItem** é o tipo de dado que é colocado na árvore

Percorso Pré-Ordem

```
void PreOrdem(TArvBin No)
{
    if (No != NULL) {
        processa (No->Item);
        PreOrdem(No->Esq);
        PreOrdem(No->Dir);
    }
}
```

Percorso Em-Ordem

```
void EmOrdem(TArvBin No)
{
    if (No != NULL) {
        EmOrdem(No->Esq);
        processa(No->Item);
        EmOrdem(No->Dir);
    }
}
```

Percorso Pós-Ordem

```
void PosOrdem(TArvBin No)
{
    if (No != NULL)  {
        PosOrdem(No->Esq) ;
        PosOrdem(No->Dir) ;
        processa(No->Item) ;
    }
}
```

Percorso Pós-Ordem

```
void PosOrdem(TArvBin No)
{
    if (No != NULL) {
        PosOrdem(No->Esq);
        PosOrdem(No->Dir);
        processa(No->Item);
    }
}
```



Em situações mais simples, *processa* pode ser substituído por um comando de escrita

Percorso Pós-Ordem

```
void PosOrdem(TArvBin No)
{
    if (No != NULL) {
        PosOrdem(No->Esq);
        PosOrdem(No->Dir);
        printf("%d\n", No->Item.Chave);
    }
}
```



Em situações mais simples, *processa* pode ser substituído por um comando de escrita

- Algoritmos de percurso em árvores (binárias ou não) visitam **todos** os nós da árvore, da raiz até as folhas
- Logo: para uma árvore com n nós, o número de nós que são processados pelo algoritmo de percurso é n

- Construa a árvore binária que representa os percursos abaixo:
 - Pré-Ordem:

A B D H E C F I G

- Em-Ordem:

H D B E A I F C G

- Pré-Ordem:

A B D H E C F I G

- Em-Ordem:

H D B E A I F C G

Árvores a partir de Percursos

- Pré-Ordem:

A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G

↑
esq

↑
dir



Raiz

Árvores a partir de Percursos

- Pré-Ordem:

A B D H E C F I G



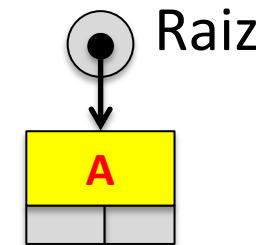
A red arrow points upwards from the word "nó" to the letter "A".

- Em-Ordem:

H D B E A I F C G



Two red arrows point upwards from the words "esq" and "dir" to the letters "D" and "G" respectively.



Árvores a partir de Percursos

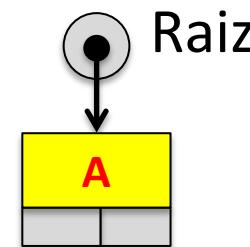
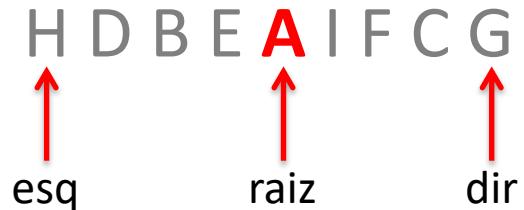
- Pré-Ordem:

A B D H E C F I G



- Em-Ordem:

H D B E A I F C G



Árvores a partir de Percursos

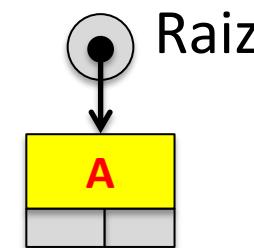
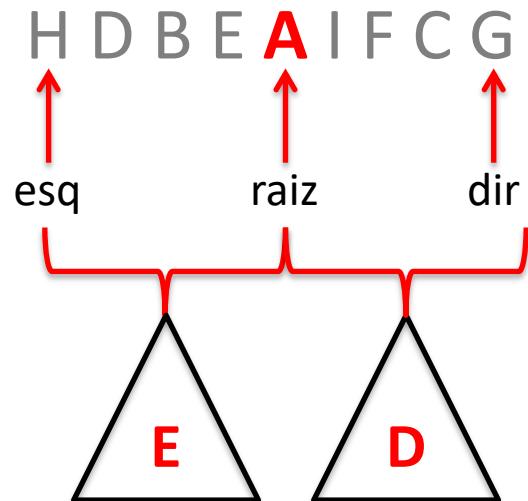
- Pré-Ordem:

A B D H E C F I G



A red arrow points from below to the letter A, with the word "nó" written vertically next to it.

- Em-Ordem:



- Pré-Ordem:

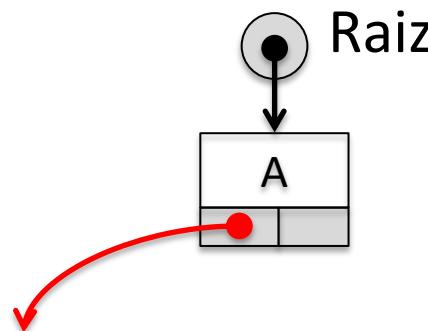
A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G

↑ esq ↑ dir



Árvores a partir de Percursos

- Pré-Ordem:

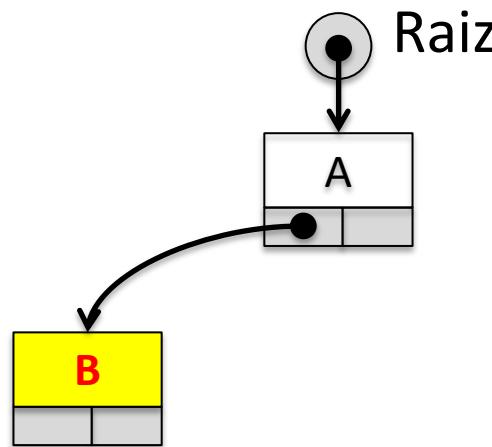
A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G

↑ esq ↑ raizdir



Árvores a partir de Percursos

- Pré-Ordem:

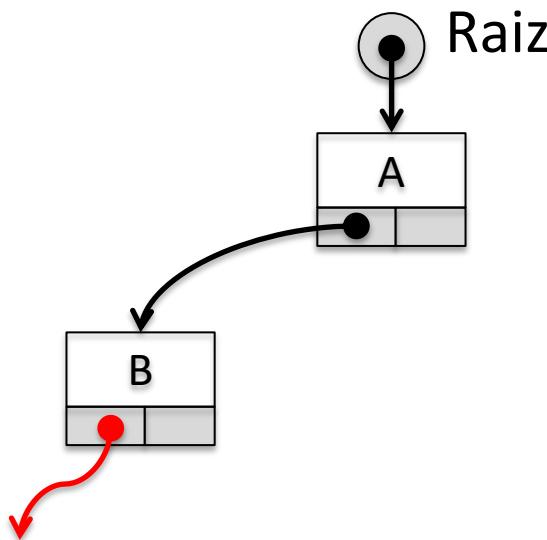
A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G

↑
esq dir



Árvores a partir de Percursos

- Pré-Ordem:

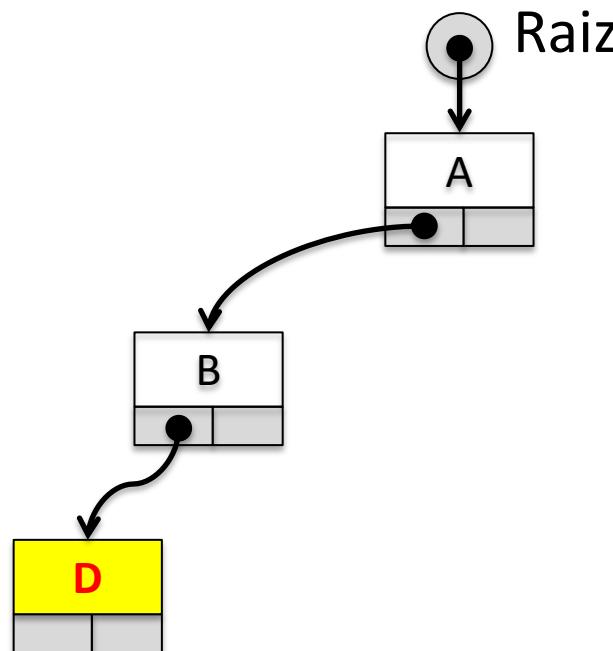
A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G

↑
esq dir = raiz



Árvores a partir de Percursos

- Pré-Ordem:

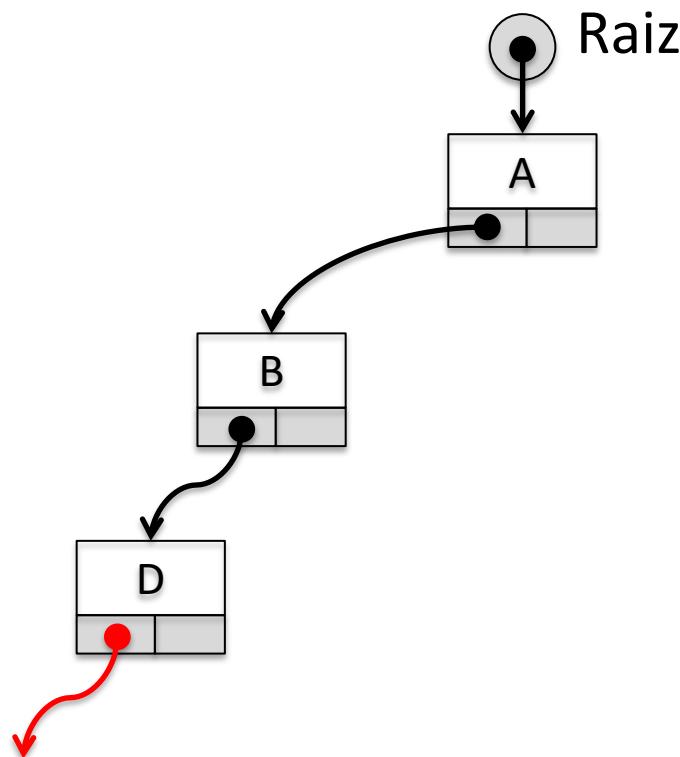
A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G

↑
esq = dir



Árvores a partir de Percursos

- Pré-Ordem:

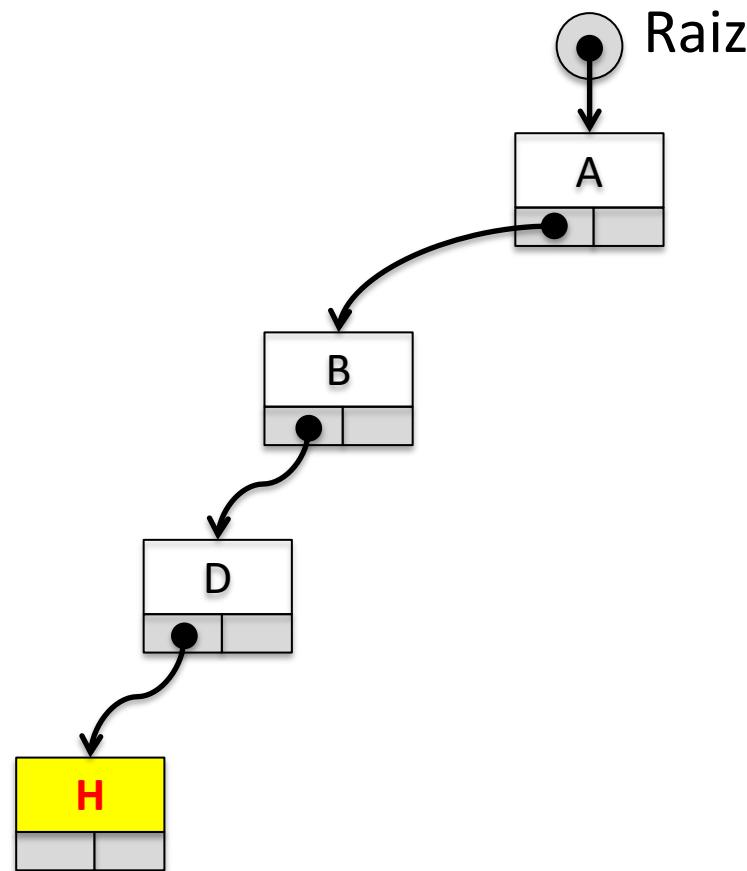
A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G

↑
esq = dir = raiz



Árvores a partir de Percursos

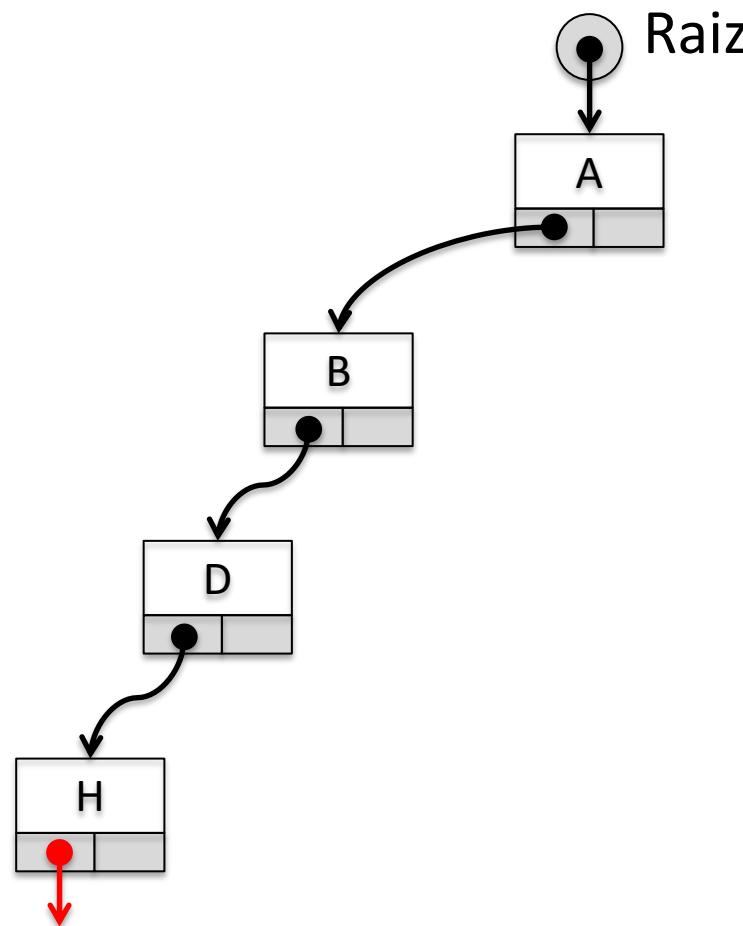
- Pré-Ordem:

A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G
↑↑
dir esq



Árvores a partir de Percursos

- Pré-Ordem:

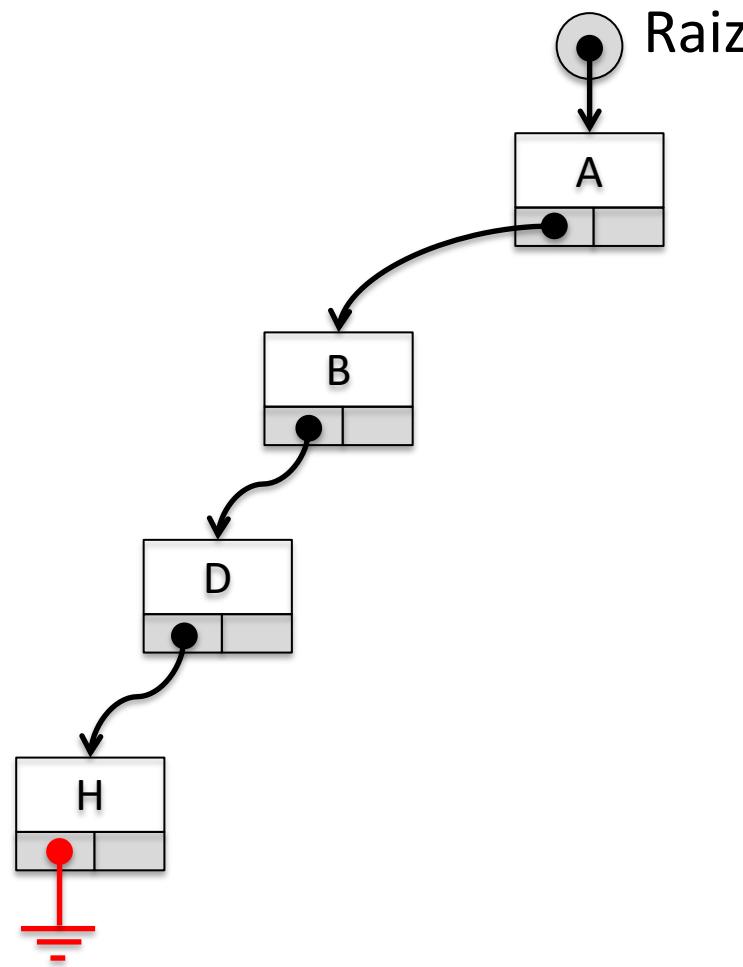
A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G

↑
↑
dir esq



Árvores a partir de Percursos

- Pré-Ordem:

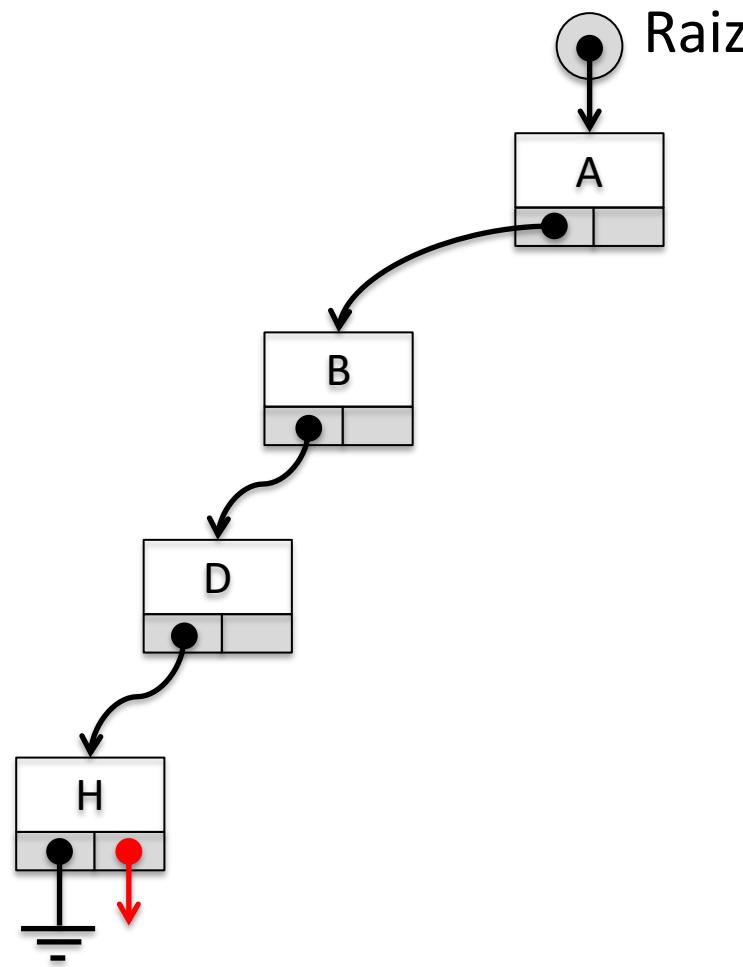
A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G

↑↑
dir esq



Árvores a partir de Percursos

- Pré-Ordem:

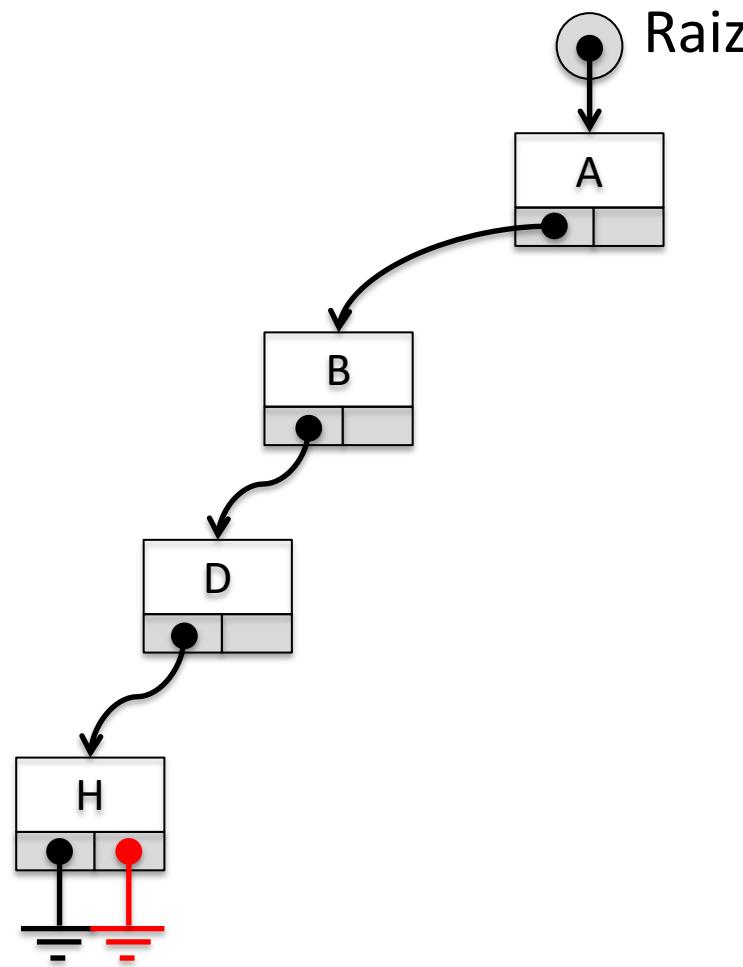
A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G

↑↑
dir esq



Árvores a partir de Percursos

- Pré-Ordem:

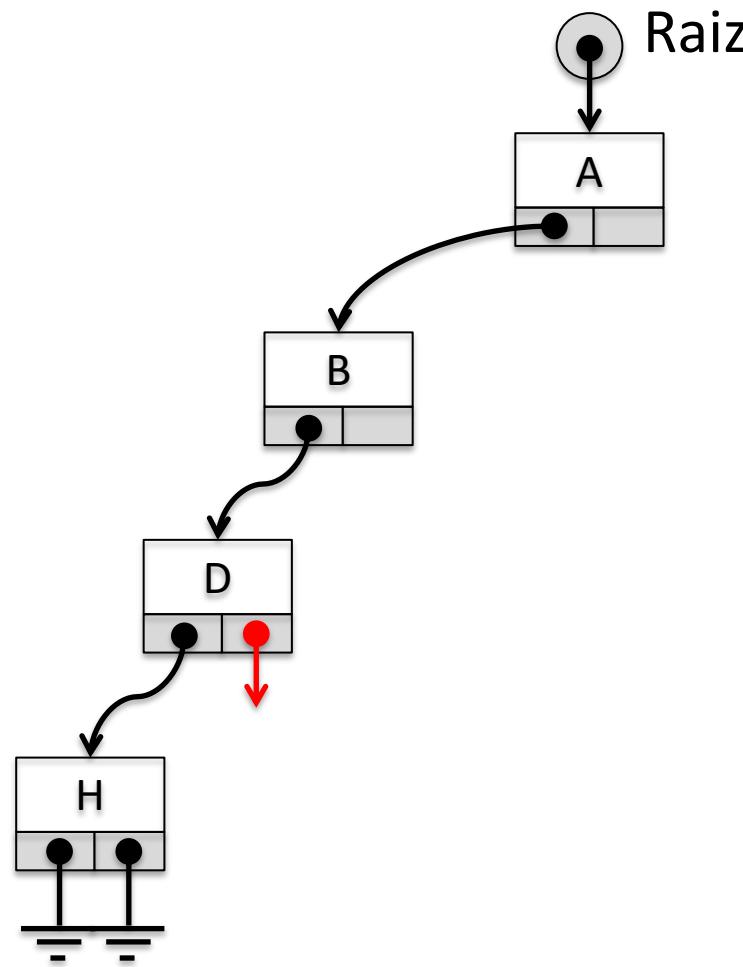
A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G

↑
↑
dir esq



Árvores a partir de Percursos

- Pré-Ordem:

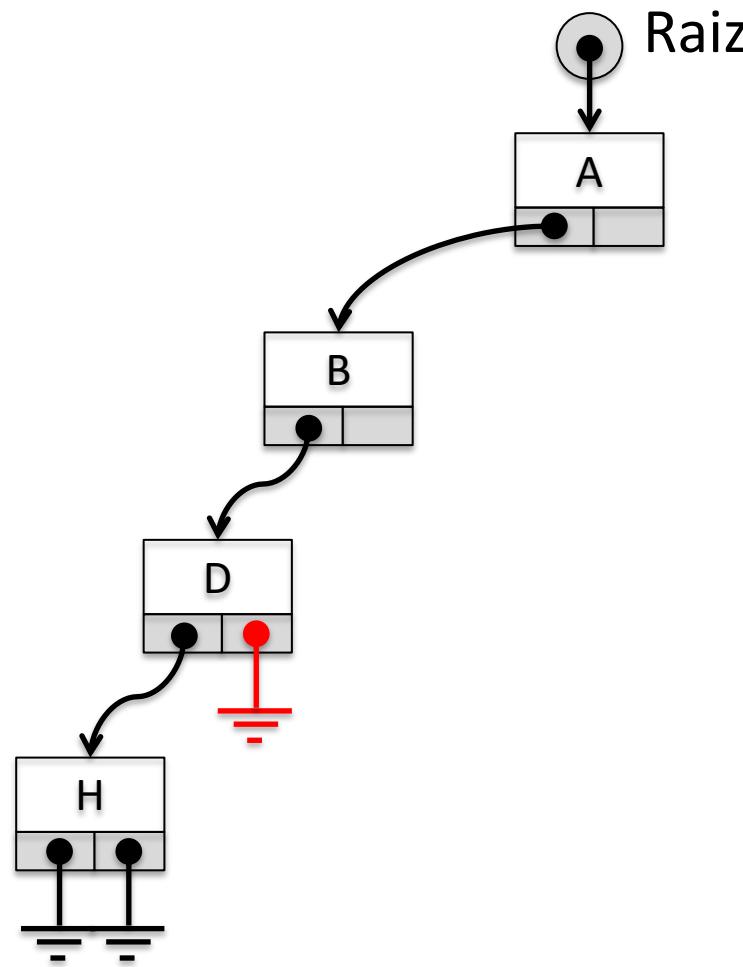
A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G

↑
↑
dir esq



Árvores a partir de Percursos

- Pré-Ordem:

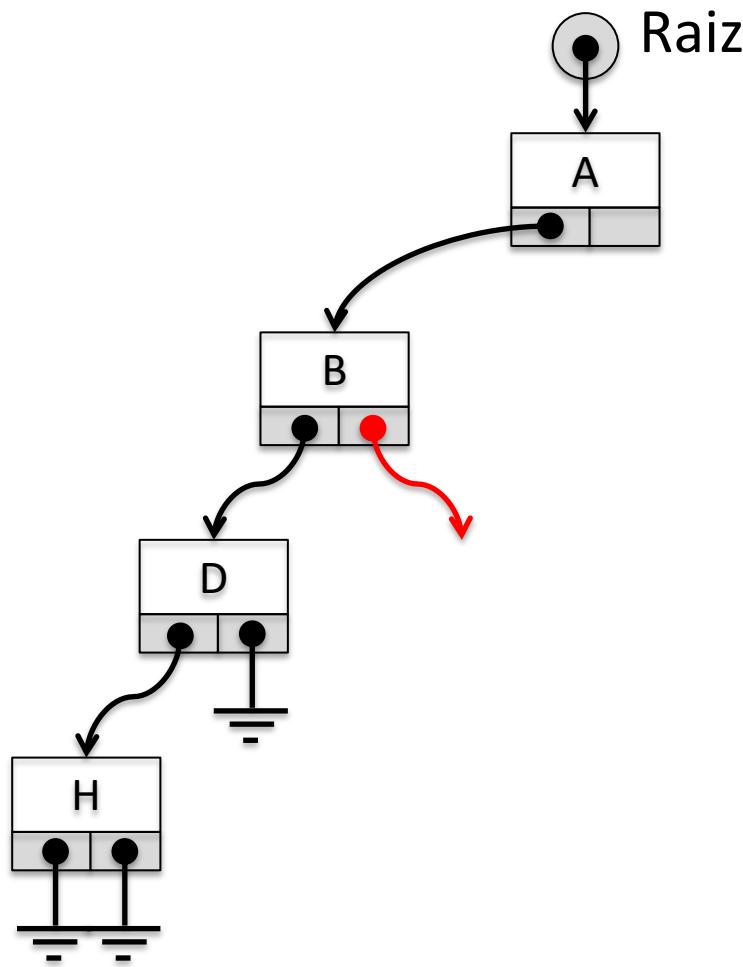
A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G

↑
esq = dir



Árvores a partir de Percursos

- Pré-Ordem:

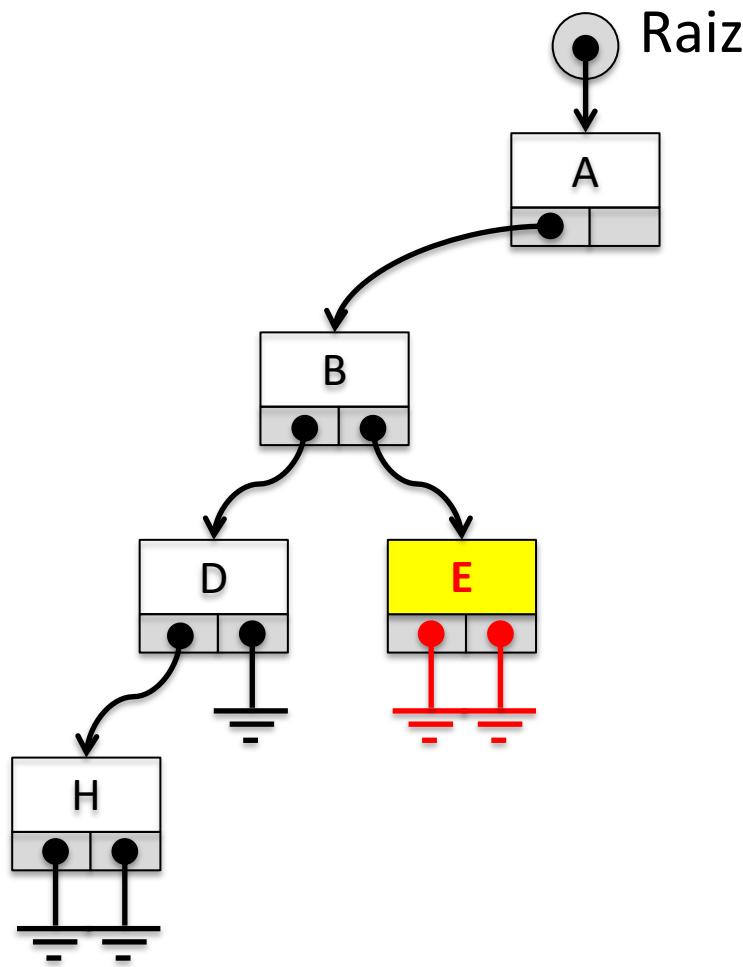
A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G

↑
esq = dir = raiz



Árvores a partir de Percursos

- Pré-Ordem:

A B D H E C F I G

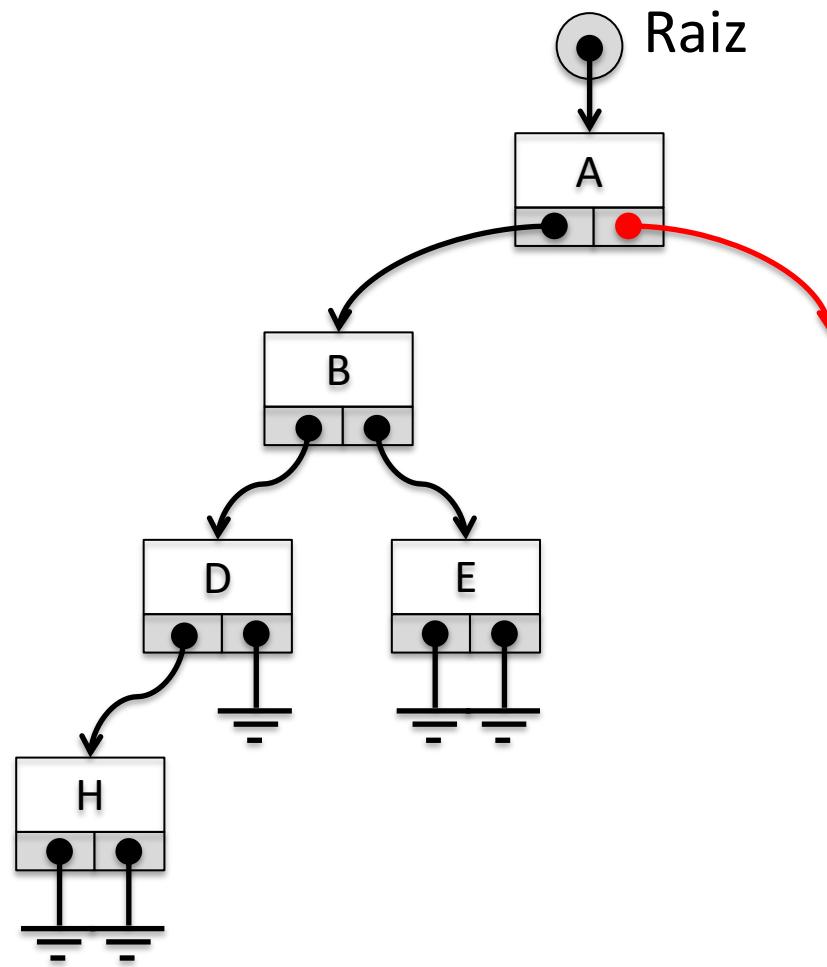
nó

- Em-Ordem:

H D B E A | F C G

esq

dir



Árvores a partir de Percursos

- Pré-Ordem:

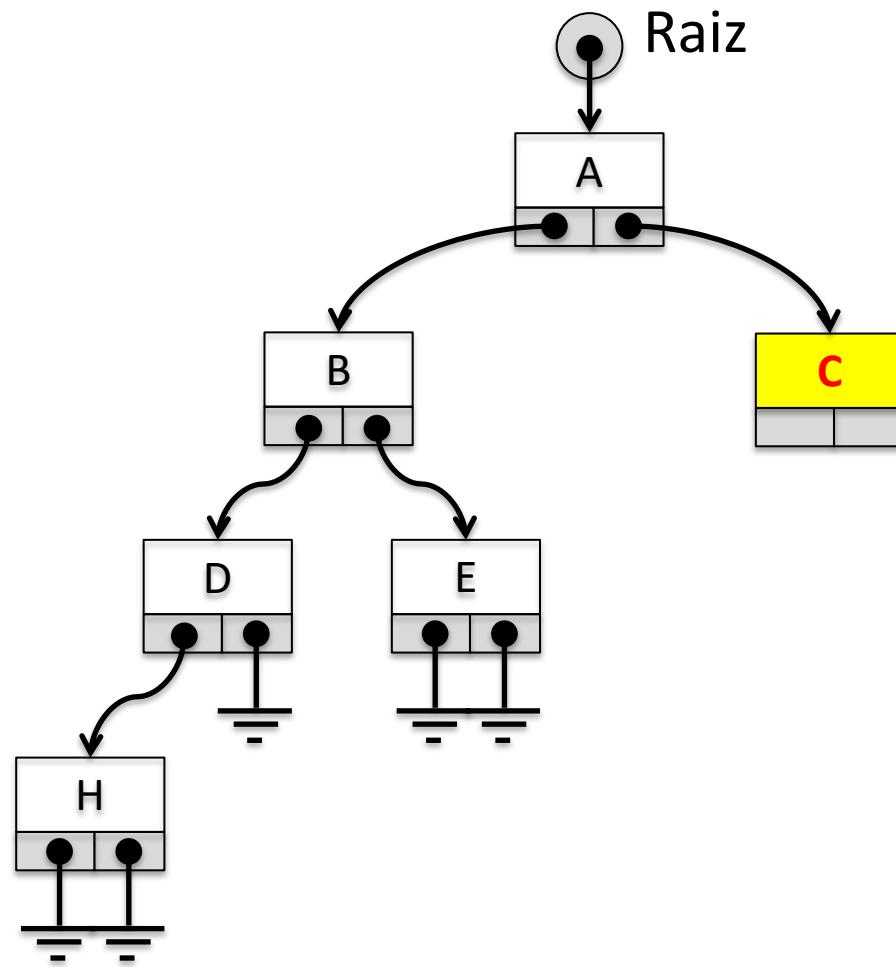
A B D H E C F I G

nó

- Em-Ordem:

H D B E A | F C G

esq raizdir



Árvores a partir de Percursos

- Pré-Ordem:

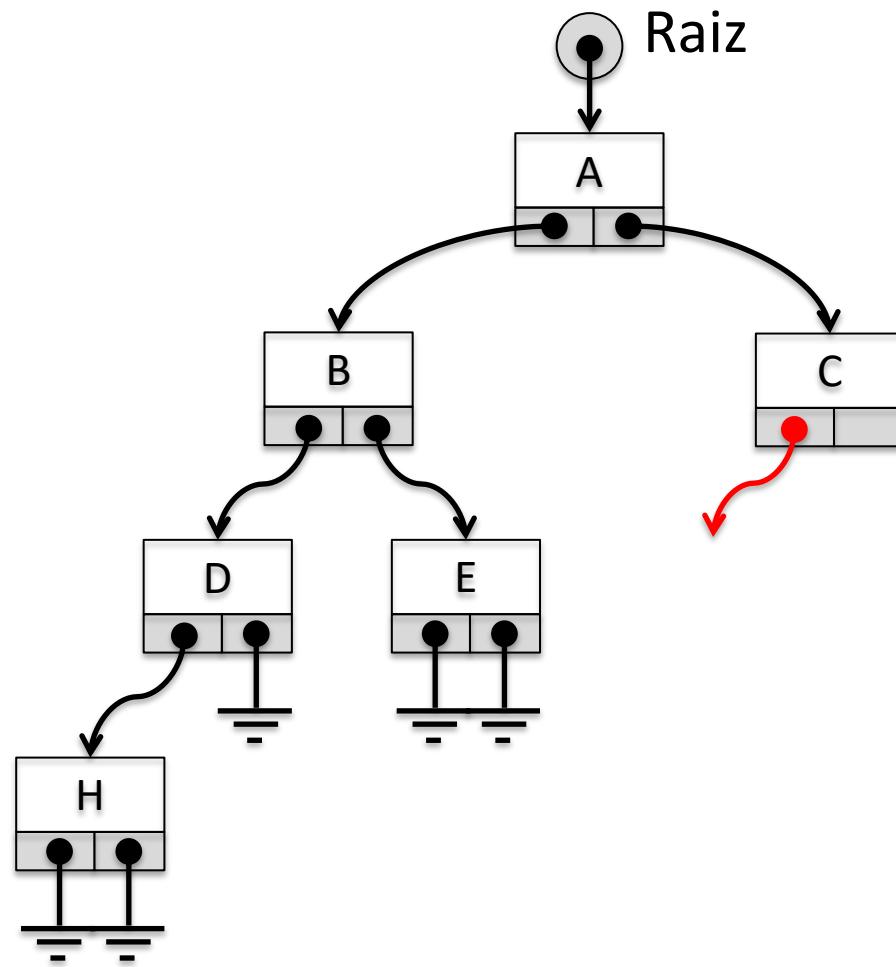
A B D H E C F I G

nó

- Em-Ordem:

H D B E A | F C G

esqdir



Árvores a partir de Percursos

- Pré-Ordem:

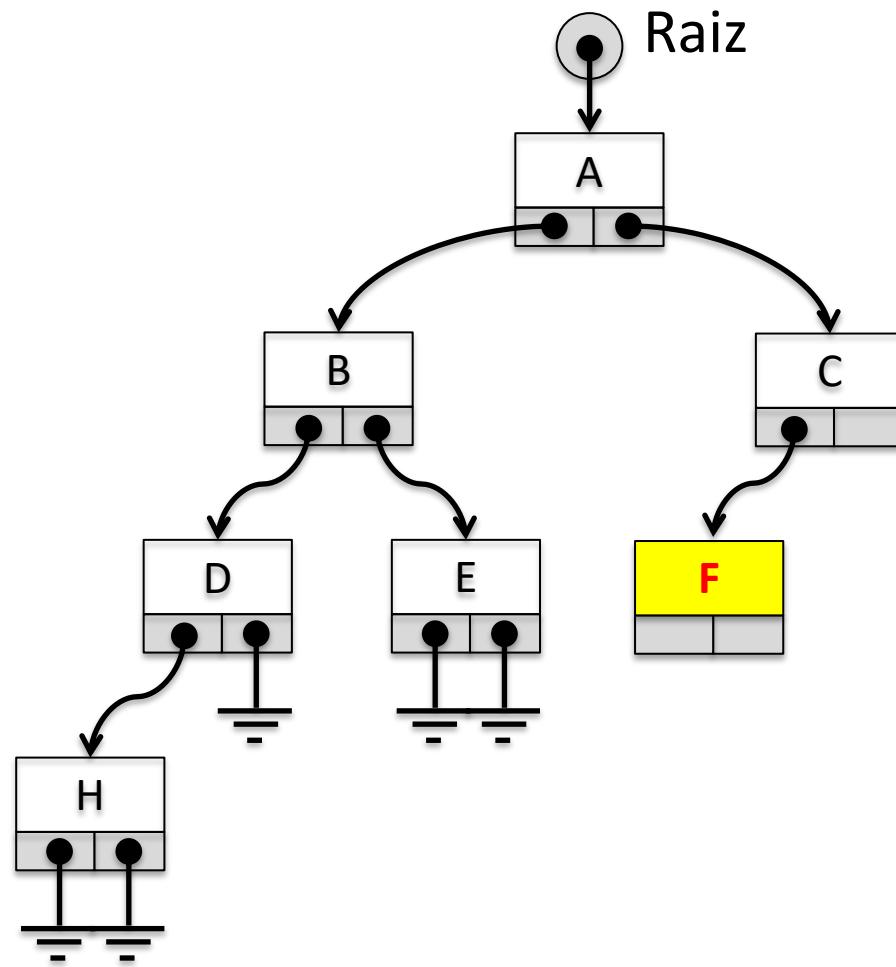
A B D H E C F | G

nó

- Em-Ordem:

H D B E A | F C G

esqdir = raiz



Árvores a partir de Percursos

- Pré-Ordem:

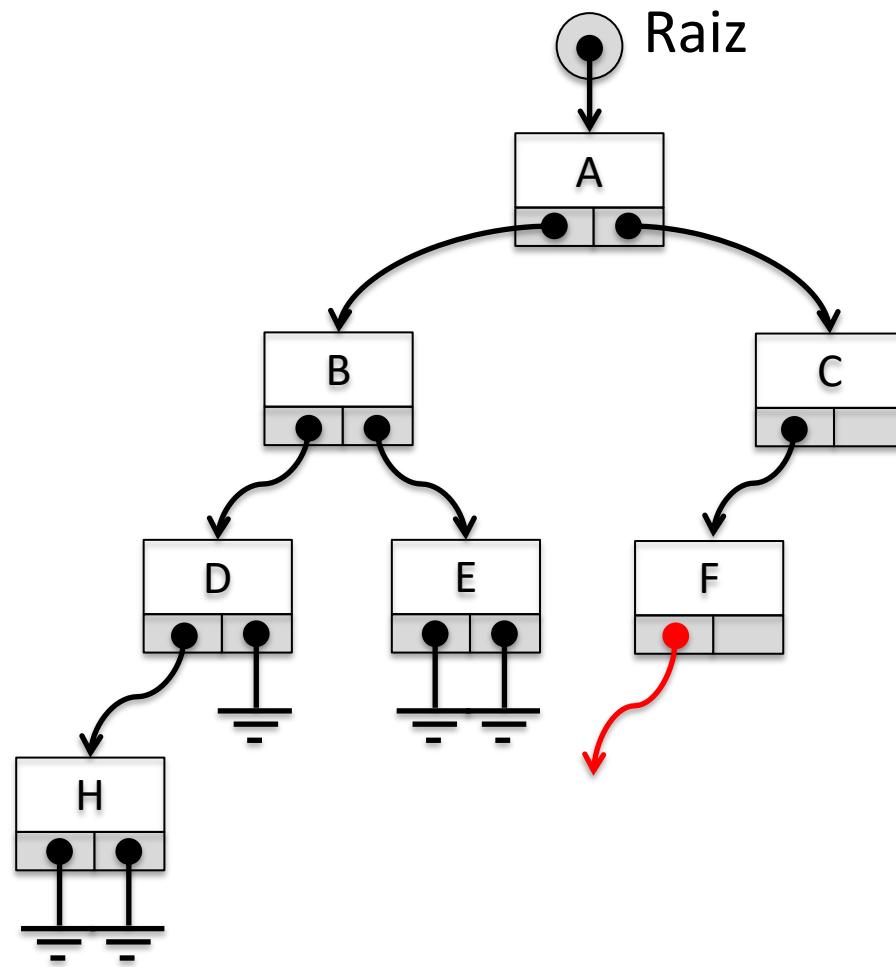
A B D H E C F | G

nó

- Em-Ordem:

H D B E A | F C G

esq = dir



Árvores a partir de Percursos

- Pré-Ordem:

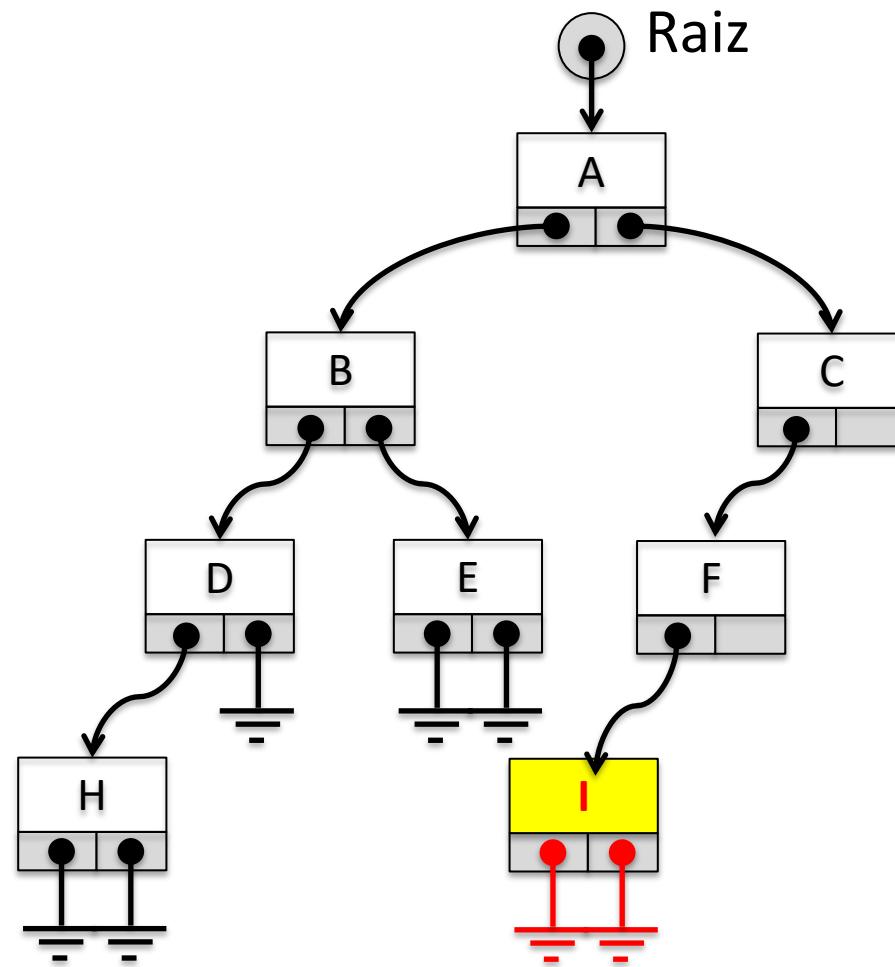
A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G

↑
esq = dir = raiz



Árvores a partir de Percursos

- Pré-Ordem:

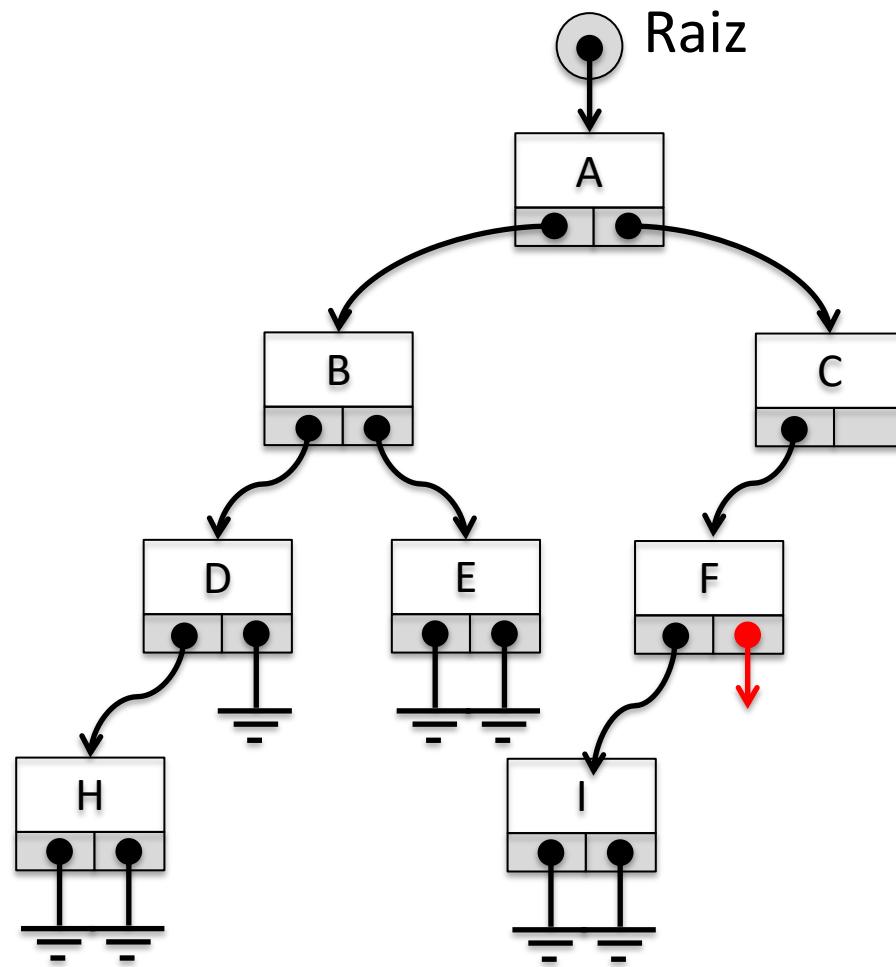
A B D H E C F I G

nó

- Em-Ordem:

H D B E A I F C G

diresq



Árvores a partir de Percursos

- Pré-Ordem:

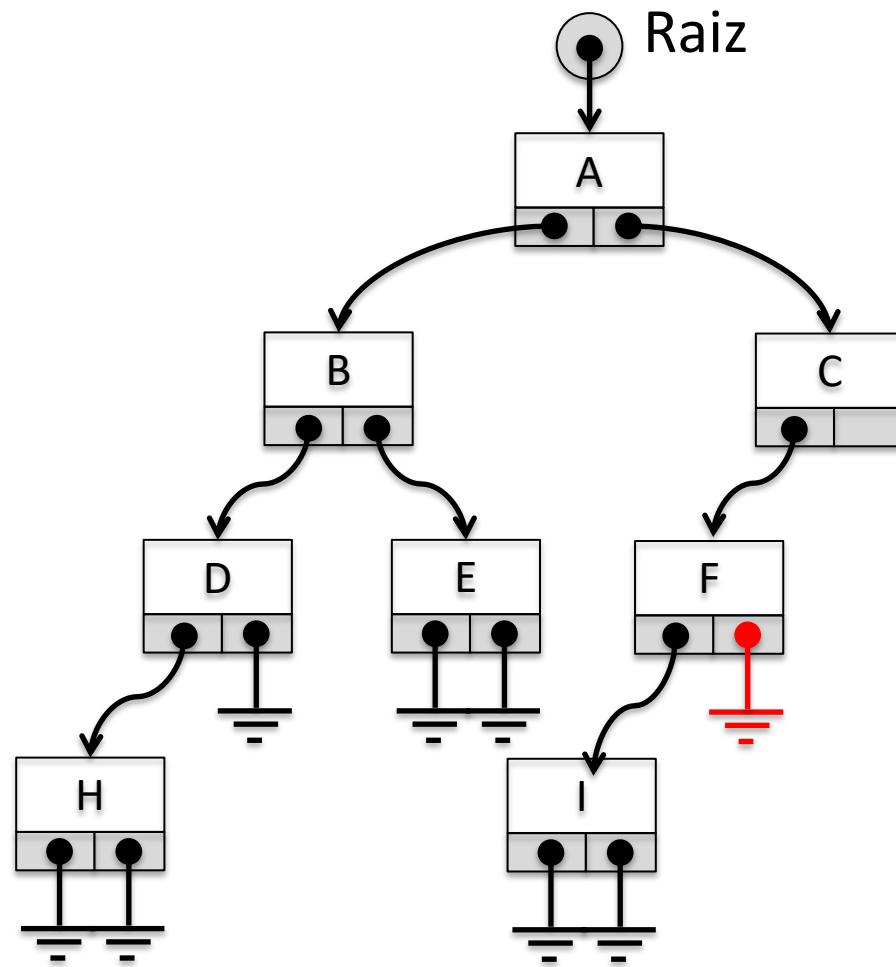
A B D H E C F I G

nó

- Em-Ordem:

H D B E A I F C G

diresq



Árvores a partir de Percursos

- Pré-Ordem:

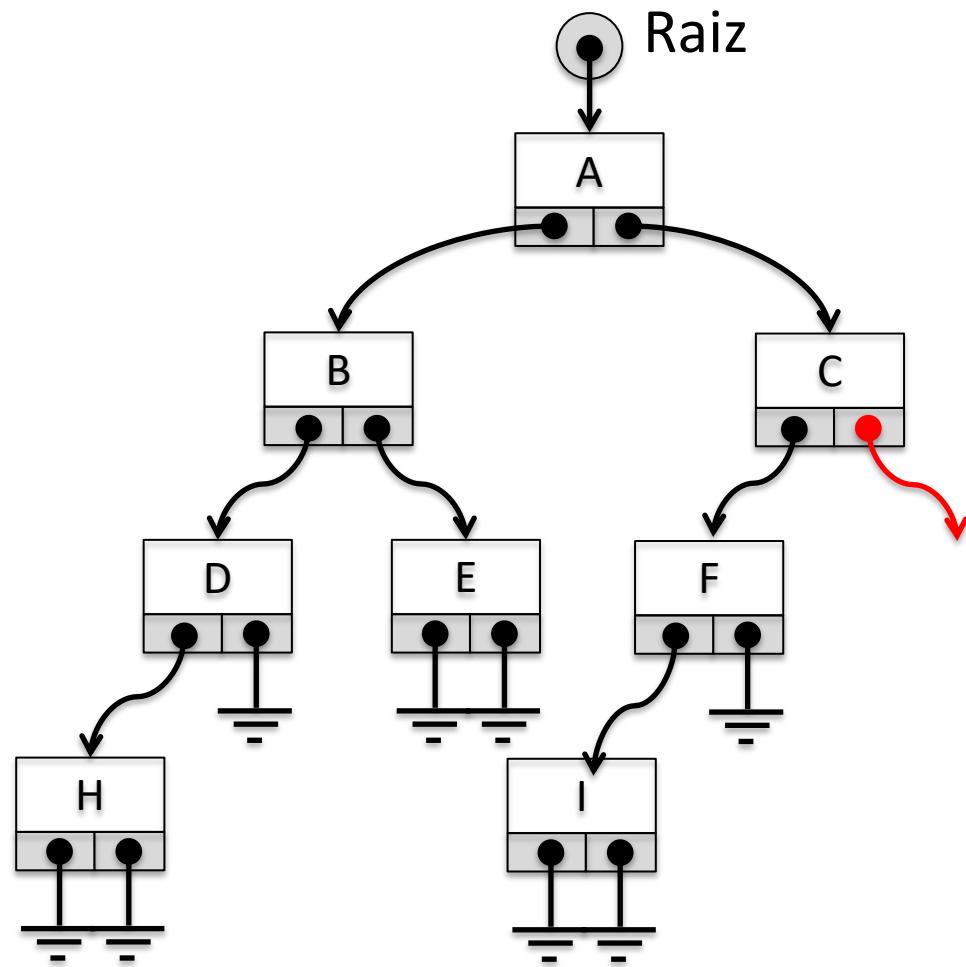
A B D H E C F I G

nó

- Em-Ordem:

H D B E A I F C G

esq = dir



Árvores a partir de Percursos

- Pré-Ordem:

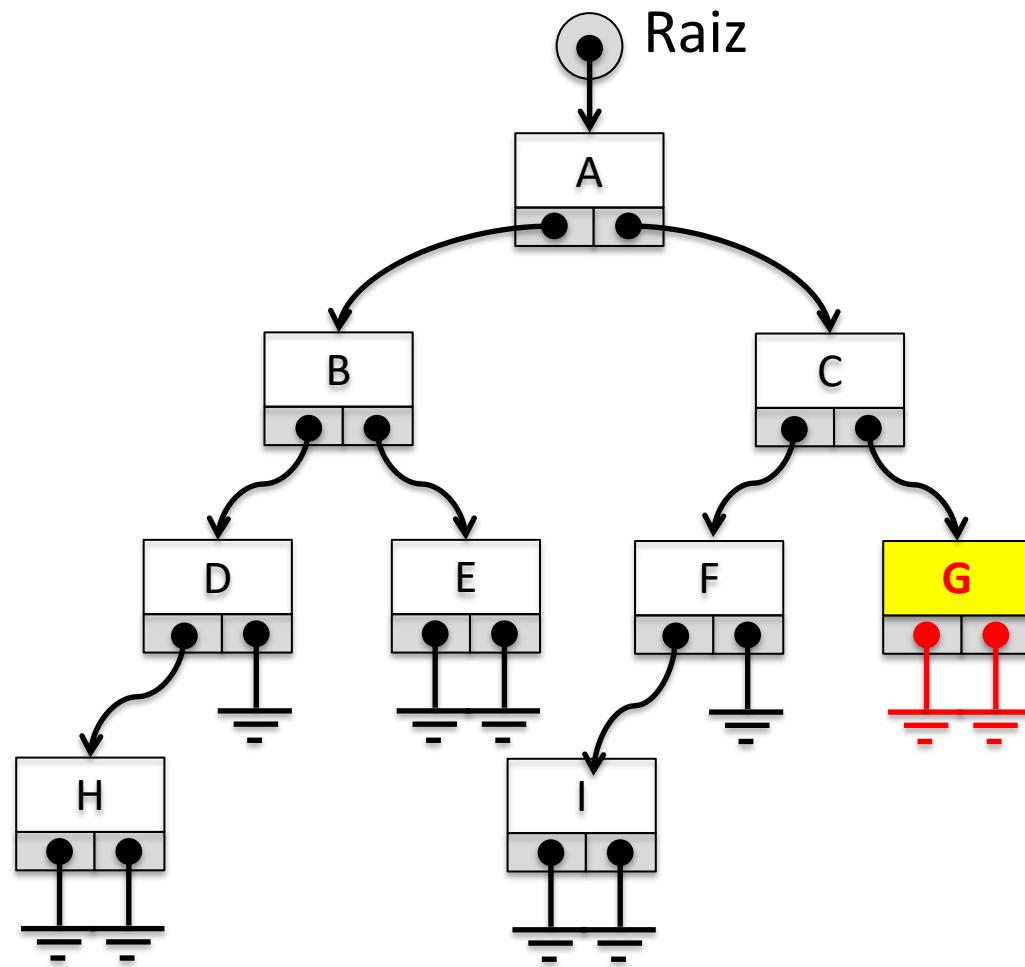
A B D H E C F I G

nó

- Em-Ordem:

H D B E A I F C G

raiz = esq = dir



Árvores a partir de Percursos

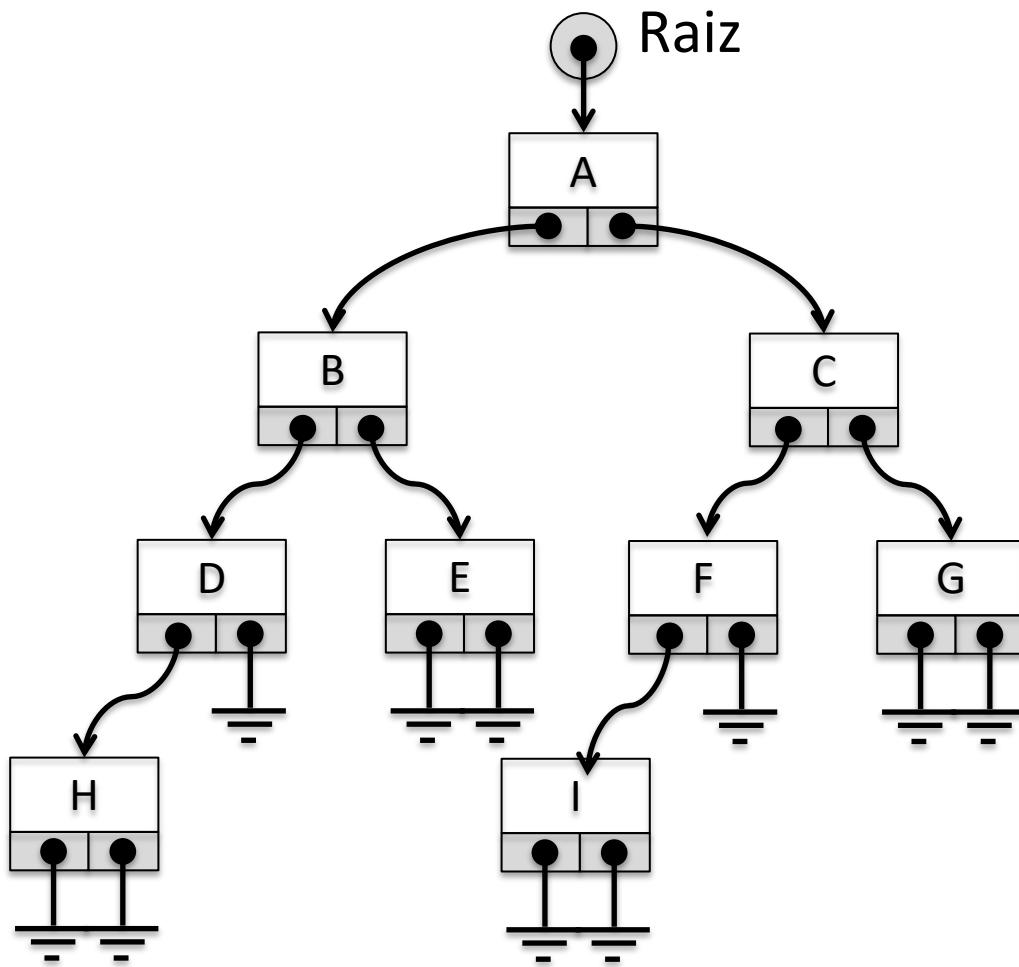
- Pré-Ordem:

A B D H E C F I G

↑
nó

- Em-Ordem:

H D B E A I F C G



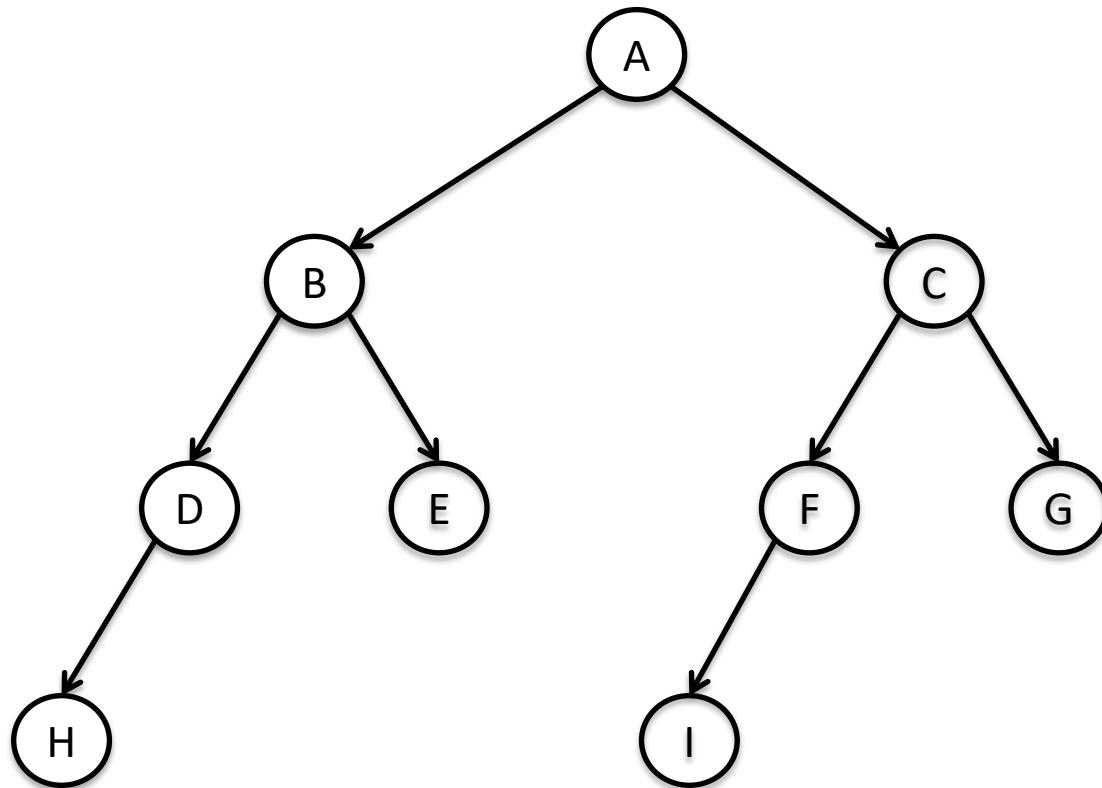
Árvores a partir de Percursos

- Pré-Ordem:

A B D H E C F I G

- Em-Ordem:

H D B E A I F C G



```
int Pesquisa(TItem *Em, int Esq, int Dir, TChave x)
{
    int Raiz;

    for (Raiz = Esq; Raiz <= Dir; Raiz++)
        if (Em[Raiz].Chave == x)
            return Raiz;

    return 0;
}

TArvBin Constroi(TItem *Pre, TItem *Em, int n)
{
    int No;
    No = 0;
    return ConstroiRecursivo(Pre, &No, Em, 0, n-1);
}
```

```
TArvBin ConstroiRecursivo(TItem *Pre, int *pNo,
                           TItem *Em, int Esq, int Dir)
{
    TArvBin pEsq, pDir;
    TItem Item;
    int Raiz;

    if (Esq <= Dir) {
        Item = Pre[(*pNo)++];
        Raiz = Pesquisa(Em, Esq, Dir, Item.Chave);
        pEsq = ConstroiRecursivo(Pre, pNo, Em, Esq, Raiz-1);
        pDir = ConstroiRecursivo(Pre, pNo, Em, Raiz+1, Dir);
        return TArvBin_CriaNo(Item, pEsq, pDir);
    }

    return TArvBin_Inicia();
}
```

- Imagine o processo de busca de informação em uma árvore (binária ou não)
 - Se as chaves não estão em uma ordem pré-estabelecida, toda a estrutura precisa ser percorrida para encontrar uma determinada chave (no pior caso), o que seria ineficiente
- Veremos na próxima aula uma forma de melhorar o tempo de busca, utilizando Árvores Binárias de Busca

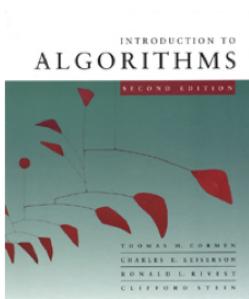
- Desenhe a árvore binária que representa os percursos abaixo:

- Pré-Ordem:

B M K A C J L Z N F W D Y E

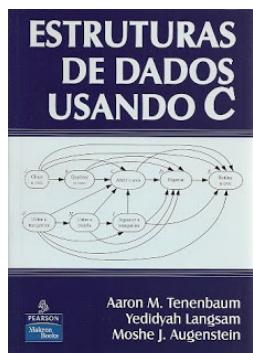
- Em-Ordem:

A C K M L J Z B F N D Y W E



CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to Algorithms**. 3^a Edição. MIT Press, 2009. **Seção 10.4**

ZIVIANI, N. **Projeto de Algoritmos com Implementações em Pascal e C**. 3^a Edição. Cengage Learning, 2010. **Seção 5.3**



TENENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. **Estruturas de Dados usando C**. Pearson Makron Books, 2008.
Capítulo 5