SCC0202 – Algoritmos e Estruturas de Dados I

Árvores Binárias

Prof.: Dr. Rudinei Goularte
(rudinei@icmc.usp.br)
Instituto de Ciências Matemáticas e de Computação - ICMC
Sala 4-229

Conteúdo

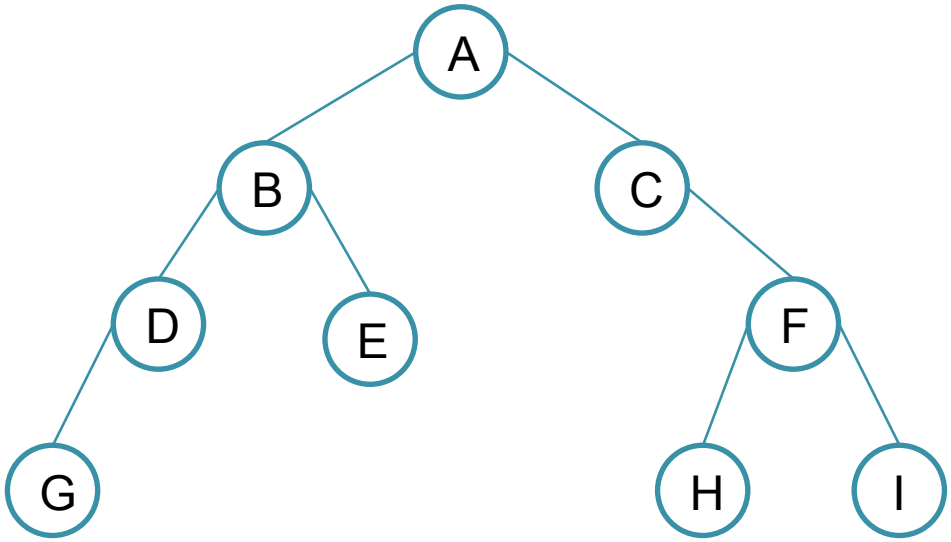
- Conceitos Básicos
- Implementação
- Percurso em Árvore Binária
- Outras Operações sobre Árvores Binárias

Árvore Binária

- Uma Árvore Binária (AB) *T* é um conjunto finito de elementos, denominados nós ou vértices, tal que
  1. Se *T* = Ø, a árvore é dita vazia, ou
  2. T contém um nó especial *r*, chamado raiz de *T*, e os demais nós podem ser subdivididos em dois sub-conjuntos distintos T_E e T_D, os quais também são árvores binárias (possivelmente vazias)
     - T_E e T_D são denominados sub-árvore esquerda e sub-árvore direita de *T*, respectivamente

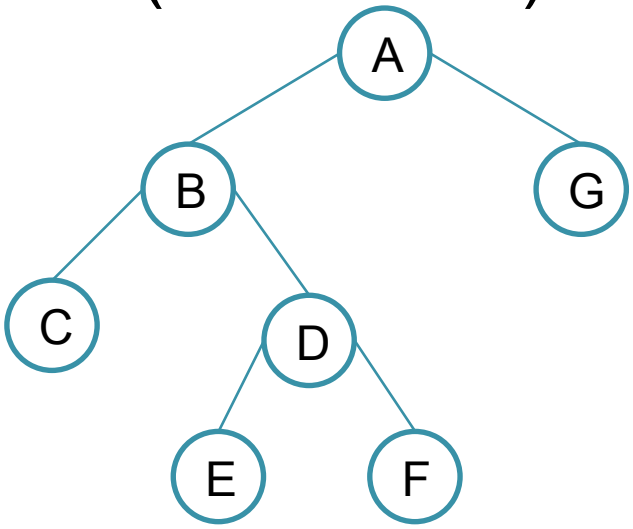
Árvore Binária

- A raiz da sub-árvore esquerda (direita) de um nó *v*, se existir, é denominada filho esquerdo (direito) de *v*
  - Pela natureza da árvore binária, o filho esquerdo pode existir sem o direito, e vice-versa



Árvore Estritamente Binária

- Uma Árvore Estritamente Binária (ou Árvore Própria) tem nós com 0 (nenhum) ou 2 (dois) filhos
- Nós interiores (não folhas) sempre têm 2 filhos

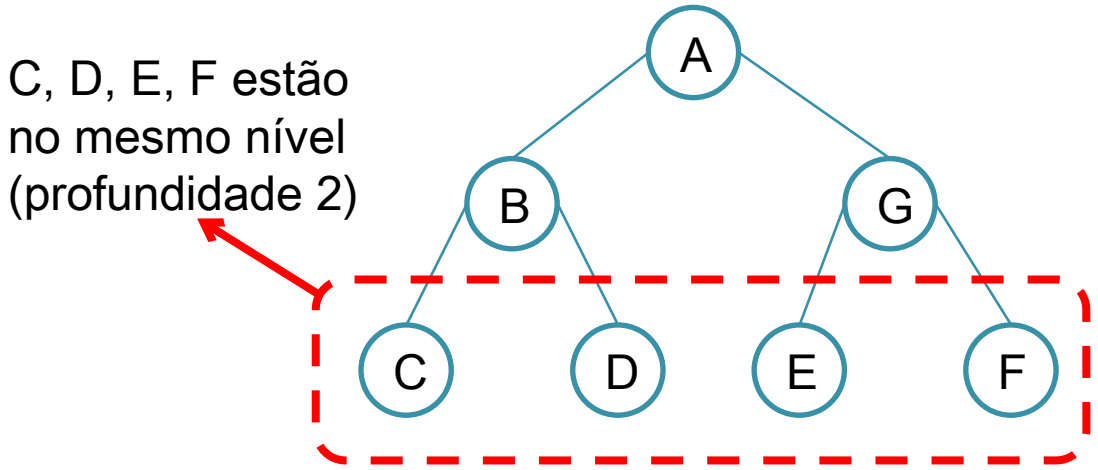


Árvore Binária Completa

- Árvore Binária Completa (ABC)
  - Se a **profundidade da árvore** é *d*, então cada nó folha está no nível *d* - 1 ou no nível *d*
  - O nível *d* - 1 está totalmente preenchido
  - Os nós folha no nível *d* estão todos mais à esquerda possível

Árvore Binária Completa Cheia

- Árvore Binária Completa Cheia (ABCC)
  - É uma Árvore Estritamente Binária
  - Todos os seus nós-folha estão no mesmo nível



Árvore Binária Completa Cheia

- Qual é o número total de nós de uma ABCC de profundidade *d*?

Árvore Binária Completa Cheia

- Dada uma ABCC e sua profundidade *d*, pode-se calcular o número total de nós na árvore
  - *d* = 0: 1 nó (total 1 nó)
  - *d* = 1 : 2 nós (total 3 nós)
  - *d* = 2 : 4 nós (total 7 nós)
  - ...
  - Profundidade *d* : 2^d nós (total 2^{d+1} - 1 nós)

Árvore Binária Completa Cheia

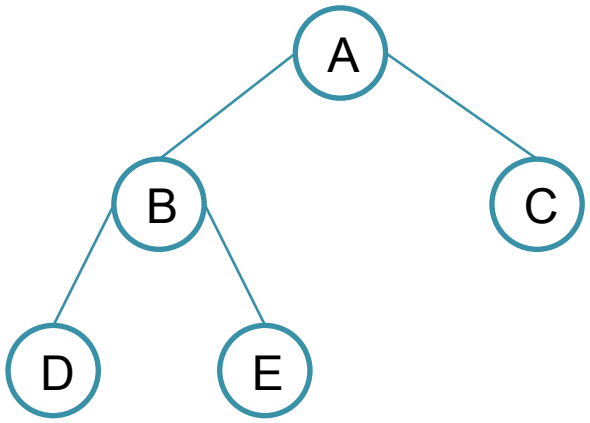
- Portanto, se o número de nós, n, para uma árvore binária completa cheia de profundidade d é
  - $n = 2^{d+1} - 1$
- Então, *n* nós podem ser distribuídos em uma árvore binária completa cheia de profundidade ...

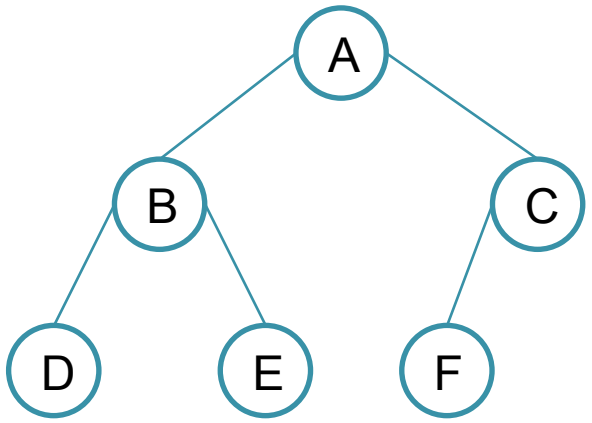
Árvore Binária Completa Cheia

- Portanto, o número de nós, n, para uma árvore binária completa cheia de profundidade d é
  - $n = 2^{d+1} - 1$
- Então *n* nós podem ser distribuídos em uma árvore binária completa cheia de profundidade:
  - $n = 2^{d+1} - 1$
  - $\log_2(n + 1) = \log_2(2^{d+1})$
  - $d = \log_2(n + 1) - 1$

Árvore Binária Balanceada

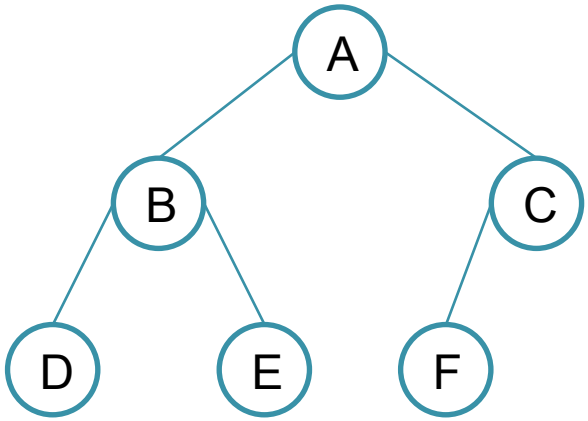
- Árvore Binária Balanceada
  - Para cada nó, as alturas de suas duas sub-árvores diferem de, no máximo, 1





Árvore Binária Perfeitamente Balanceada

- Árvore Binária Perfeitamente Balanceada
  - Para cada nó, o número de nós de suas sub-árvores esquerda e direita difere em, no máximo, 1
  - Toda Árvore Binária Perfeitamente Balanceada é Balanceada, mas o inverso não é necessariamente verdade



Questões

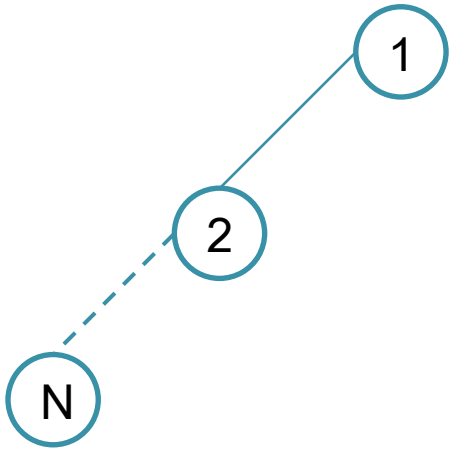
- Toda árvore balanceada é completa?
- E o inverso?
- Toda árvore perfeitamente balanceada é completa?
- E o inverso?

Questões

- Qual a altura máxima de uma AB com *n* nós?

Questões

- Qual a altura máxima de uma AB com *n* nós?
  - Resposta: *n* - 1
  - Árvore degenerada ≡ Lista

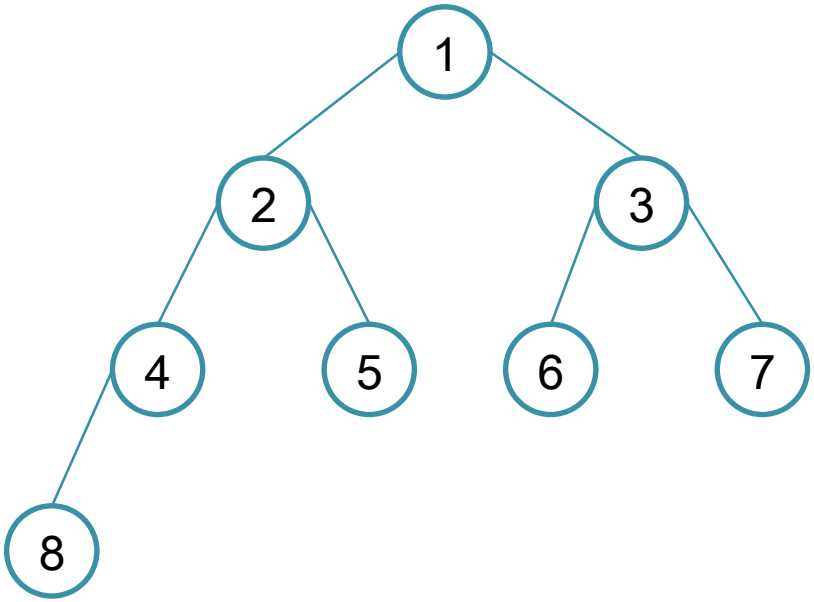


Questões

- Qual a altura mínima de uma AB com *n* nós?

Questões

- Qual a altura mínima de uma AB com *n* nós?
  - Resposta: a mesma de uma AB Perfeitamente Balanceada com *n* nós

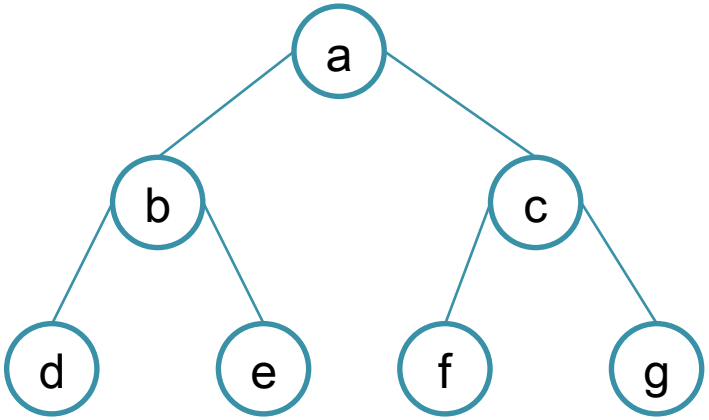


N = 1 H = 0
N = 2, 3 H = 1
N = 4 ... 7 H = 2
N = 8 ... 15 H = 3

$$H_{\min} = \lfloor \log_2 n \rfloor$$

Implementação de ABCC (organização sequencial)

- Armazenar os nós, por nível, em um array



0	1	2	3	4	5	6		n
a	b	c	d	e	f	g		...

Implementação de ABC (organização sequencial)

- Para um vetor indexado a partir da posição 0, se um nó está na posição i, seus filhos diretos estão nas posições
  - $2_i + 1$: filho da esquerda
  - $2_i + 2$: filho da direita
- Vantagem: espaço só p/ armazenar conteúdo; ligações implícitas
- Desvantagem: espaços vagos se árvore não é completa por níveis, ou se sofrer eliminação

Implementação de AB (Encadeada)

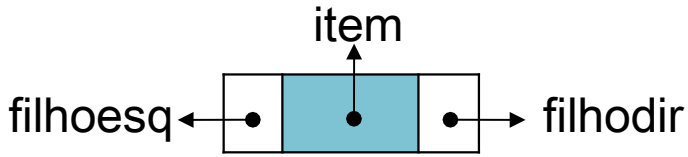
- Para qualquer árvore, cada nó é do tipo

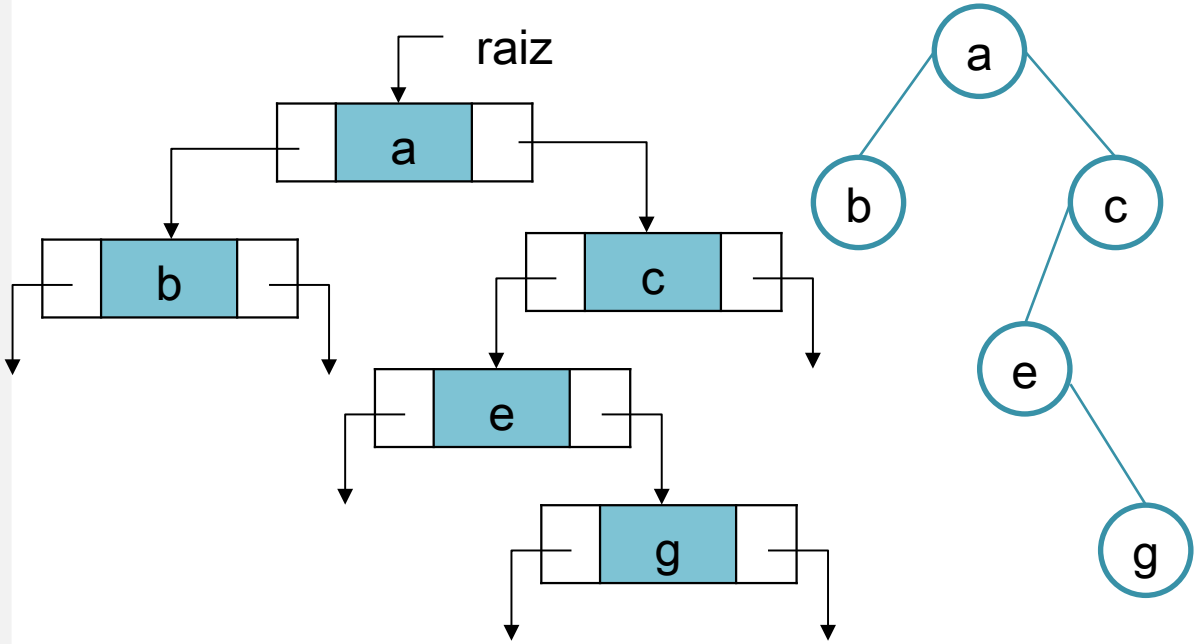
```
//.h
#include “item.h”
typedef struct arv_bin AB;

//.c
typedef struct No NO;
struct No {
   ITEM *item;
   NO *esq;
   NO *dir;
};

struct arv_bin {
   NO *raiz;
   int profundidade;
   ...
};

//main.c
AB *T;
T= ab_criar();
```





Operações do TAD AB I

- Criar árvore
  - Pré-condição: nenhuma
  - Pós-condição: inicia a estrutura de dados

Operações do TAD AB II

- Inserir um nó filho
  - Pré-condição: nó pai não nulo.
  - Pós-condição: dado um nó pai, cria seu nó filho e o insere à direita ou esquerda do pai. Retorna VERDADEIRO se o pode ser criado, FALSO caso contrário.

Operações do TAD AB (Criar)

```
1 AB *ab_criar(void) {
2    AB *r = (AB *) malloc(sizeof(AB));
3    if (r != NULL) {
4       r->raiz = NULL;
5       r->profundidade = -1;
6    }
7    return (r);
8  }
9
```

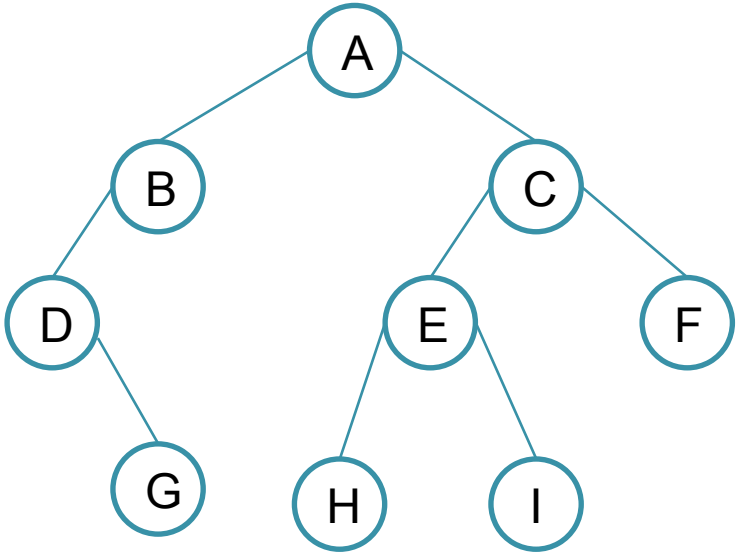
AB - Percursos

- Percorrer uma AB “visitando” cada nó uma única vez
  - “Visitar” um nó pode ser
    - Mostrar o seu valor
    - Modificar o valor do nó
    - ...
- Um percurso gera uma sequência linear de nós, e podemos então falar de nó predecessor ou sucessor de um nó, segundo um dado percurso
- Não existe um percurso único para árvores (binárias ou não): diferentes percursos podem ser realizados, dependendo da aplicação

AB - Percursos em Árvores

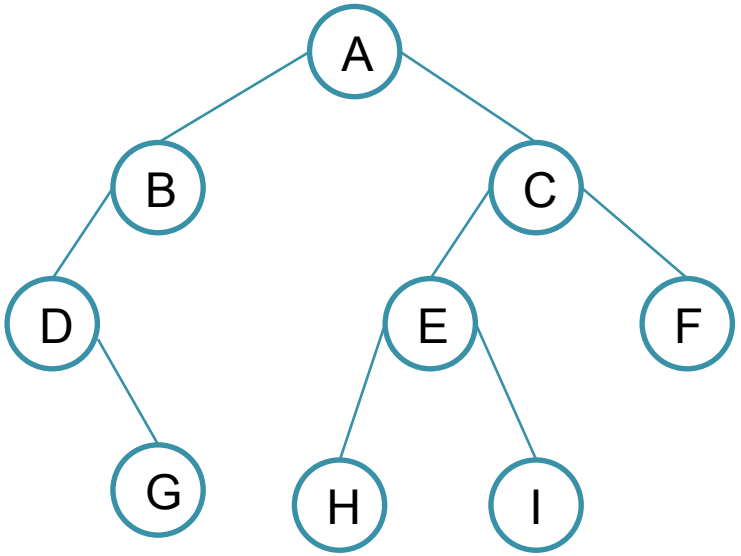
- 3 percursos básicos para AB's:
  - pré-ordem (Pre-order)
    - visita a raiz
    - percorre a subárvore a esquerda em pré-ordem
    - percorre a subárvore a direita em pré-ordem
  - em-ordem (In-order)
    - percorre e subárvore a esquerda em-ordem
    - visita a raiz
    - percorre a subárvore a direita em-ordem
  - pós-ordem (Post-order)
    - percorre e subárvore a esquerda em pós-ordem
    - percorre a subárvore a direita em pós-ordem
    - visita a raiz
- A diferença entre eles está, basicamente, na ordem em que os nós são "visitados"

AB - Percurso Pré-Ordem



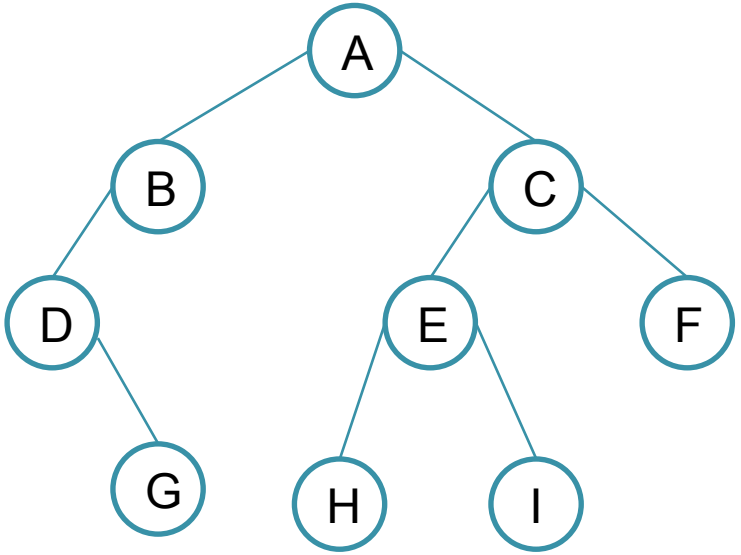
AB - Percurso Pré-Ordem

```
void ab_preordem(NO *raiz) {
  if (raiz != NULL) {
    item_imprimir(raiz->item);
    ab_preordem(raiz->esq);
    ab_preordem(raiz->dir);
  }
}
```



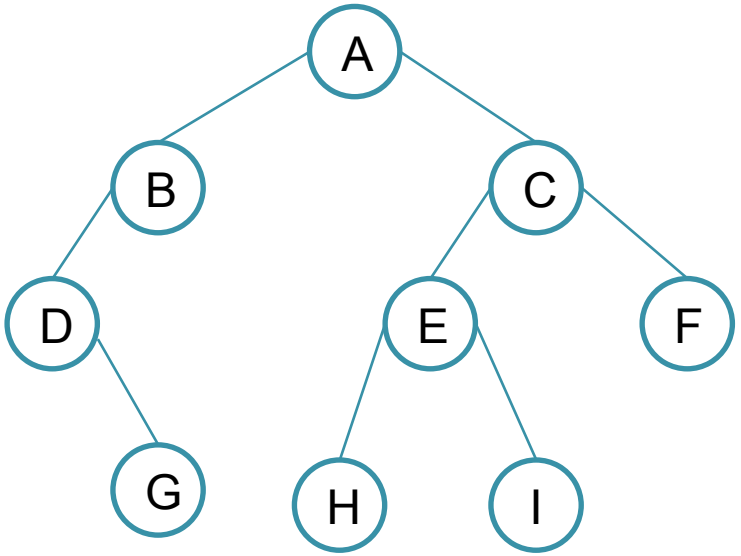
- Resultado: ABDGCEHIF

AB - Percurso Em-Ordem



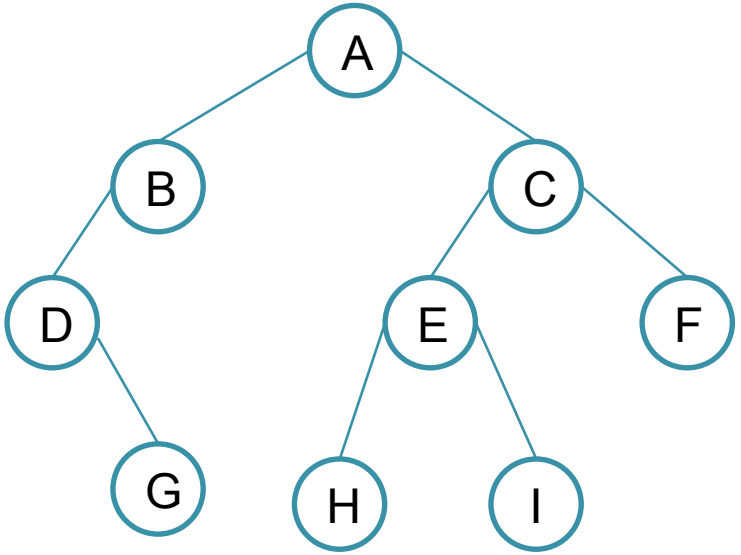
AB - Percurso Em-Ordem

```
void ab_emordem(NO *raiz) {
  if (raiz != NULL) {
    ab_emordem(raiz->esq);
    item_imprimir(raiz->item);
    ab_emordem(raiz->dir);
  }
}
```



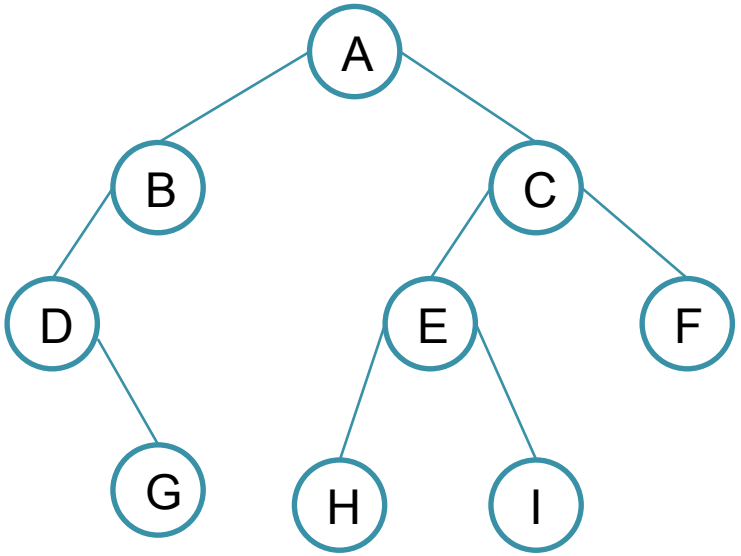
- Resultado: DGBAHEICF

AB - Percurso Pós-Ordem



AB - Percurso Pós-Ordem

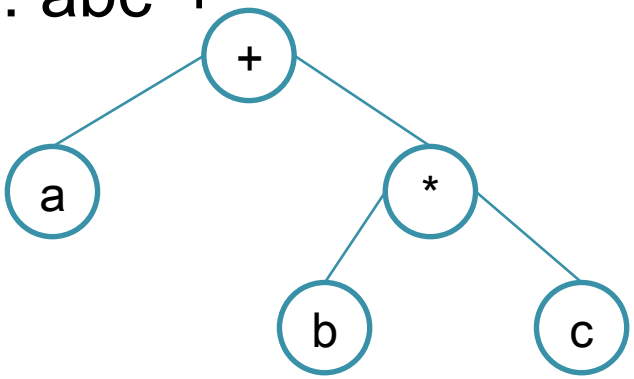
```
void ab_posordem(NO *raiz) {
  if (raiz != NULL) {
     ab_posordem(raiz->esq);
     ab_posordem(raiz->dir);
     item_imprimir(raiz->item);
  }
}
```



- Resultado: GDBHIEFCA

AB - Percursos

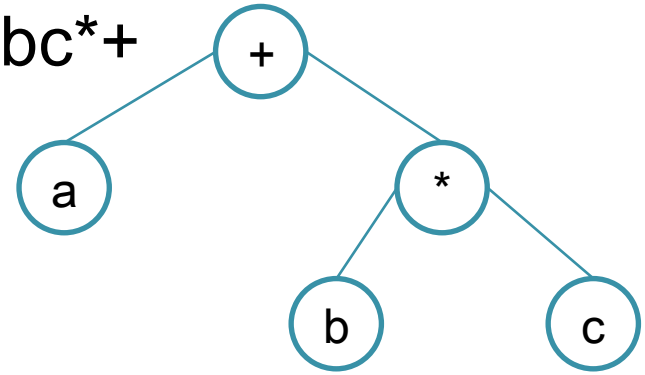
- Percurso para expressões aritméticas
  - Em-ordem: a+(b*c)
  - Pré-ordem: +a*bc
  - Pós-ordem: abc*+



  - Qual percurso utilizar?
  - Como calcular?

AB - Percursos

- Percurso para expressões aritméticas
  - Pré-ordem: +a*bc
  - Em-ordem: a+(b*c)
  - Pós-ordem: abc*+



- Algoritmos para cálculo podem usar pilhas.

Operações do TAD AB (Inserção)

Operações do TAD AB (Inserção)

```
1 /* .h: #define FILHO_ESQ 0 #define FILHO_DIR 1 */
3 NO *ab_inserir_no(NO *raiz, NO *no, int lado, int chave) {
4   if (raiz != NULL) {
5     raiz->esq = ab_inserir_no(raiz->esq, no, lado, chave);
6     raiz->dir = ab_inserir_no(raiz->dir, no, lado, chave);
7     if (chave == item_get_chave(raiz->item)){
8         if (lado == FILHO_ESQ)
9            raiz→esq = no;
10         else if (lado == FILHO_DIR)
11            raiz→dir = no;
12    }
13  }
14  return(raiz);
15 }
16
17 bool ab_inserir(AB *T, ITEM *item, int lado, int chave){
18   if (T->raiz == NULL)
19     return((T->raiz = ab_cria_no(item)) != NULL);
20   else {
21      NO *novo_no = ab_cria_no(item);
22      if(novo_no != NULL){
23         T→raiz = ab_inserir_no(T→raiz, novo_no, lado, chave);
24          return(true);
25       }
26       return(false);
27   }
28 }
```

Exercícios

- Toda árvore binária completa cheia é uma árvore binária completa?
- Toda árvore estritamente binária é uma árvore binária completa?
- Escreva um procedimento recursivo que calcula a profundidade de uma AB
- Escreva um procedimento recursivo que apaga uma árvore

Função recursiva para calcular profundidade de uma árvore

```
1  int ab_profundidade(NO *no) {
2    if (no == NULL) return -1;
3    int e = ab_profundidade(no->esq);
4    int d = ab_profundidade(no->dir);
5    return ((e > d) ? e : d) + 1;
6  }
7
```

Procedimento recursivo p/ destruir árvore, liberando o espaço alocado

```
1  void apagar_arvore(NO **raiz) {
2    if (*raiz != NULL) {
3      apagar_arvore(&(*raiz)->esq);
4      apagar_arvore(&(*raiz)->dir);
5      item_apagar(&(*raiz)->item));
6      free(*raiz);
7      *raiz = NULL;
8    }
9  }
10
11 void ab_apagar_arvore(AB **T) {
12   apagar_arvore(&(*T)->raiz);
13   free(*T);
14   *T = NULL;
15 }
```

Exercícios

- Considerando uma árvore que armazene inteiros
  - Implemente um método que retorne a quantidade de elementos em uma árvore
  - Implemente um método que retorne o maior elemento de uma árvore
  - Implemente um método que retorne o menor elemento de uma árvore
  - Implemente um método que retorne a soma de todos elementos de uma árvore