Tipos Especiais de Listas

Introdução a Árvores Binárias

Sumário

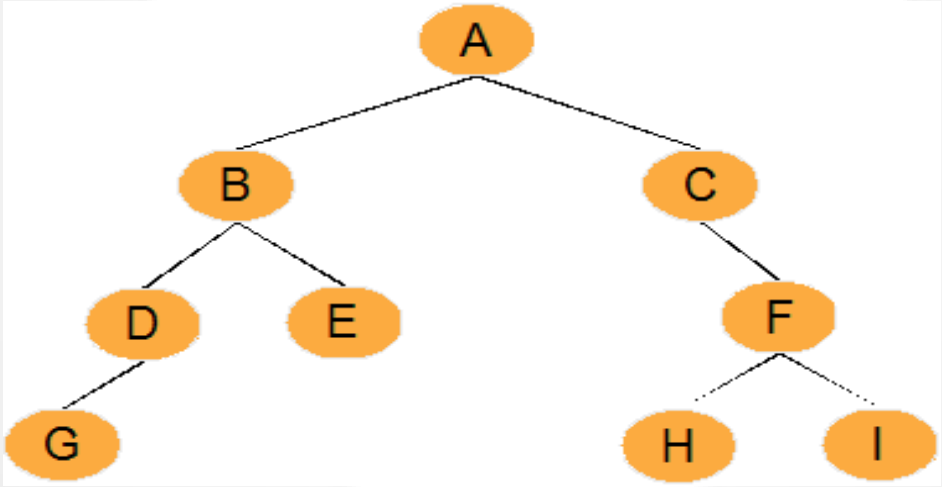
- Conceitos básicos
- Implementação
- Percurso em Árvores Binária
- Outras operações sobre árvores binárias
- Exercícios

Árvores Binárias

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

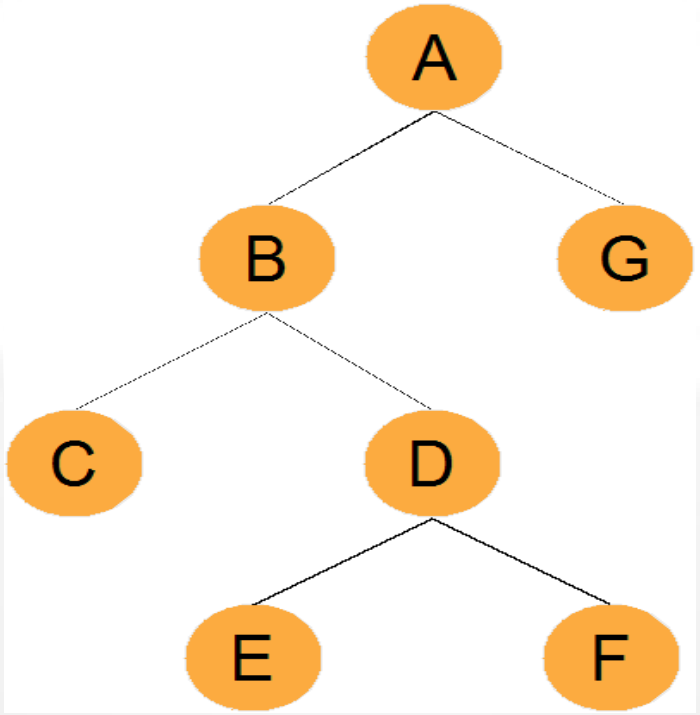
Árvore Binárias

- 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 Binárias

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

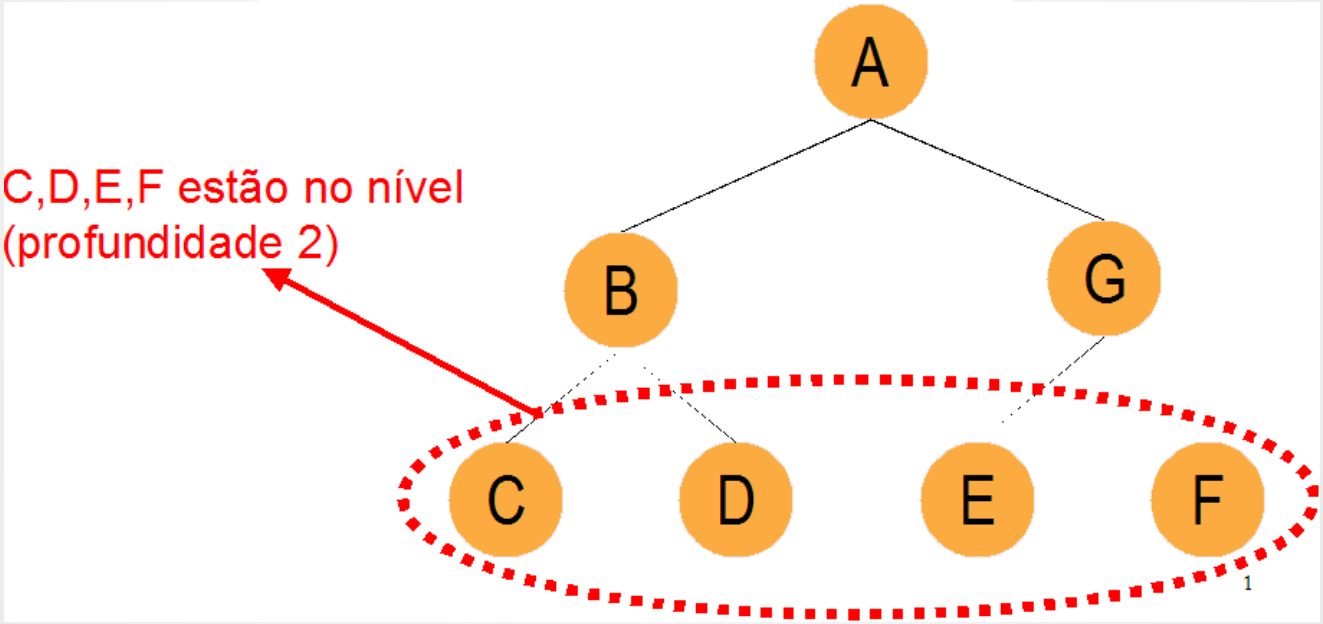


Árvore Binária Completa

- 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

- Dada uma ABCC e sua profundidade d, pode-se calcular o número total de nós na árvore
  - d = 0: 1 nó (total 1 nó)

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 nó (total 3 nó)

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 nó (total 3 nó)
  - d = 2: 4 nó (total 7 nó)
  - ….

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 nó (total 3 nó)
  - d = 2: 4 nó (total 7 nó)
  - ….
  - Profundidade d : 2^d nós (total 2^(d+1) -1 nós)

Árvore Binária Completa Cheia

- Uma ABCC tem $n_d = 2^d$ nós no nível d, e no total tem $S_n = \sum_{k=0}^{d} n_k = 2^{d+1} - 1$ nós

Prova

$$
\begin{aligned}
S_n &= a_0q^0 + a_0q^1 + a_0q^2 + \ldots + a_0q^d \\
q \cdot S_n &= a_0q^1 + a_0q^2 + a_0q^3 + \ldots + a_0q^{d+1} \\
q \cdot S_n - S_n &= a_0q^{d+1} - a_0 \\
(q-1)S_n &= a_0(q^{d+1} - 1) \\
S_n &= \frac{a_0(q^{d+1}-1)}{(q-1)}
\end{aligned}
$$

- Com $a_0 = 1$ e $q = 2$, temos $S_n = 2^{d+1} - 1$

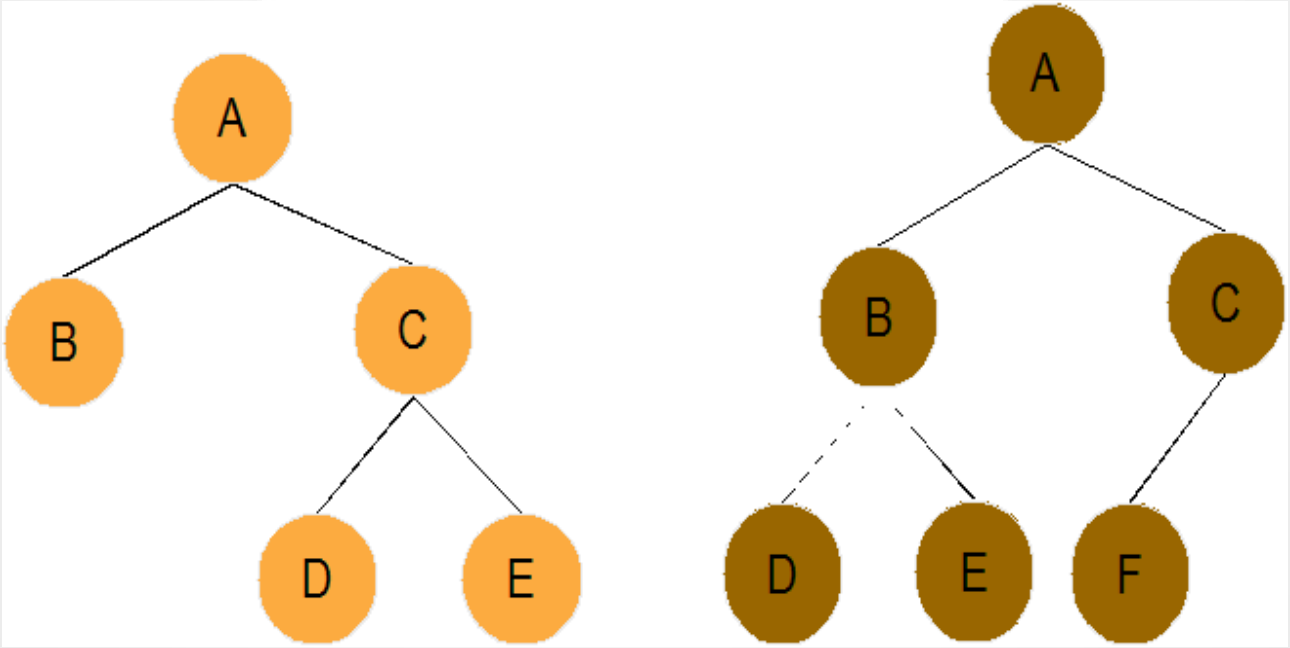
Árvore Binária Completa Cheia

- 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
  - log2(n + 1) = log2(2^d+1)
  - d = log2(n + 1) - 1

Árvore Binária Balanceada

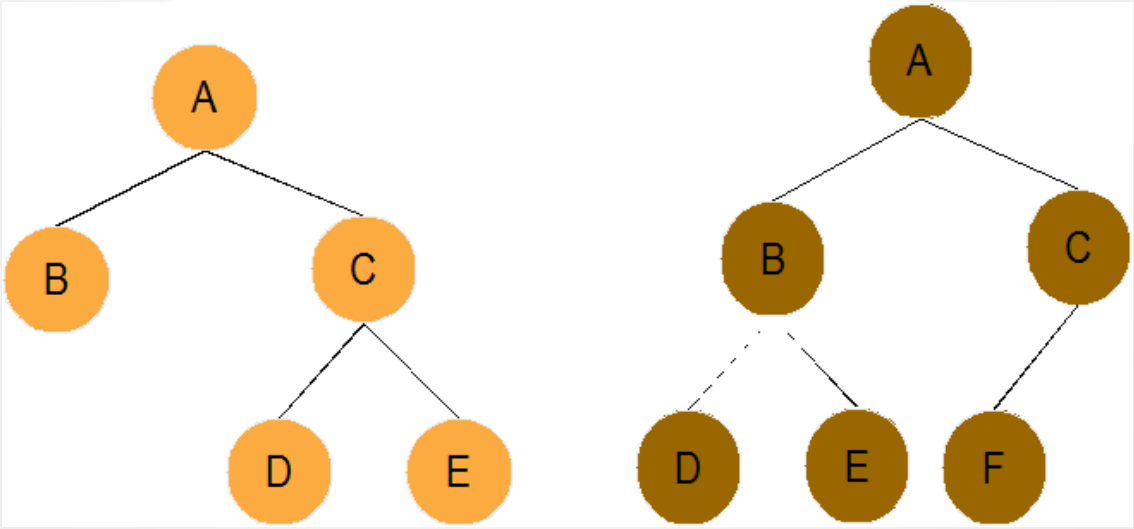
- Árvore Binária Balanceada (ABB)
  - Para cada nó, as alturas de suas duas sub-árvores diferem de, no máximo, 1



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

Qual?

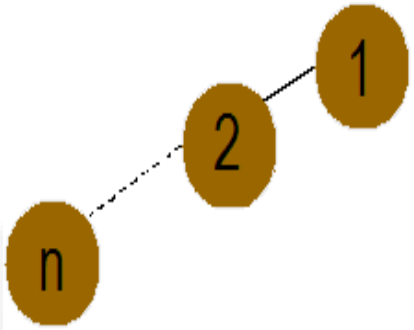


Questões

- 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

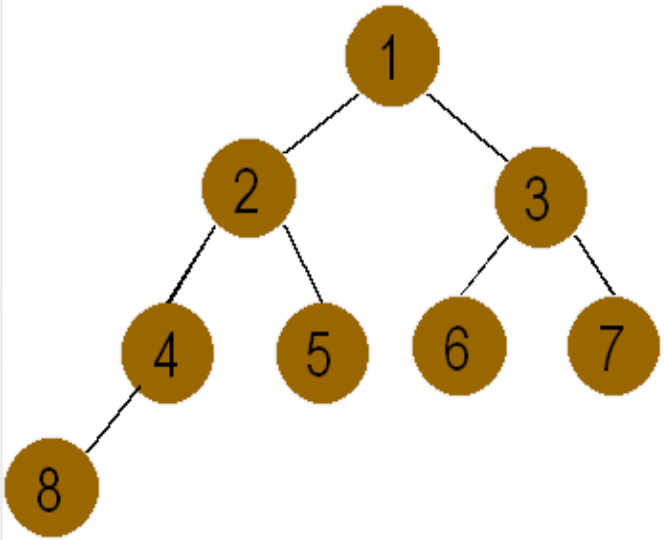


Questões

- Qual a altura miníma de uma AB com n nós?

Questões

- Qual a altura miníma 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

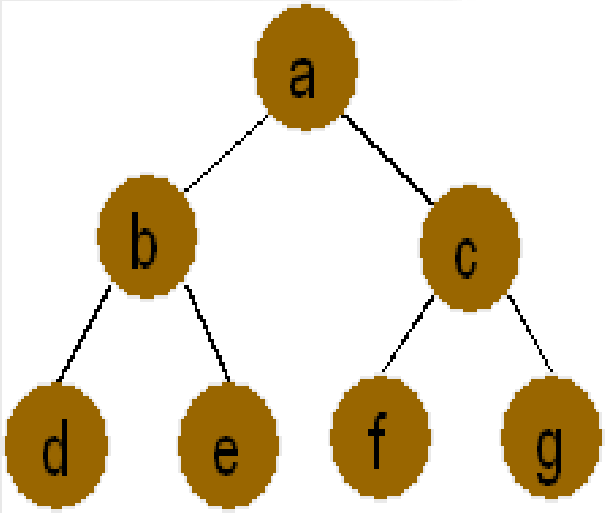
n=8..15; h=3

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

Implementação

Implementação de ABC (alocação estática, seqüencial)

- Armazenar os nós, por nível, em um array



1	2	3	4	5	6	7		...
a	b	c	d	e	f	g		...

Implementação de ABC (alocação estática, seqüencial)

- Para um vetor indexado a partir da posição 0, se um nó está na posição i, seus filhos diretos estão nas posições
  - 2i + 1 : filho da esquerda
  - 2i + 2 : filho da direita

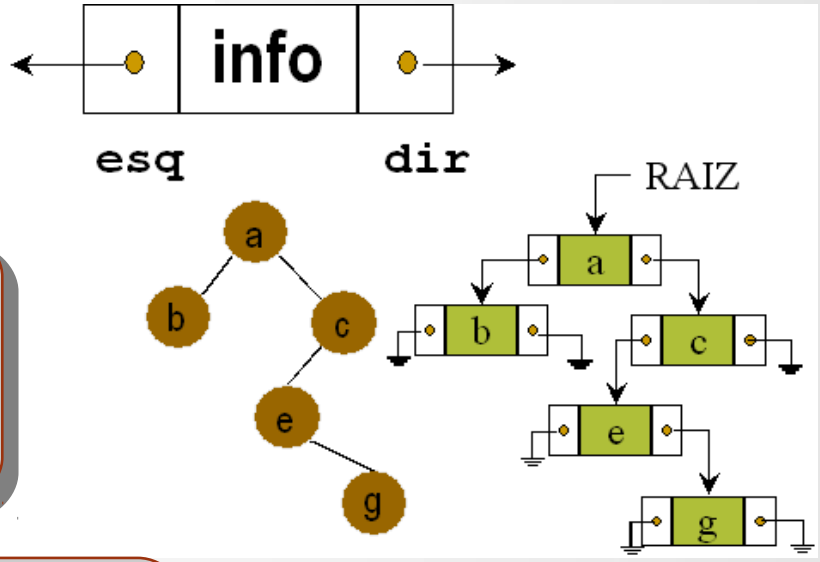
- Vantagem: espaço só p/ armazenar conteúdo; ligações implícitas

- Desvantagem: espaços vagos se árvore não é completa por níveis, ou se sofrer eliminação

Implementação de AB (dinâmica)

- Para qualquer árvore, cada nó é do tipo

```
typedef struct {
   int lin;
   char chave;
} INFO;
```




```
typedef struct NO {
   INFO info;
   struct NO *fesq;
   struct NO *fdir;
} tNO;
```

```
typedef struct NO {
   tNO *raiz;
} Arvore_Binaria;
```

Operações do TAD AB I

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

- Criar raiz
  - Pré-condição: nenhuma
  - Pós-condição: cria o nó raiz da árvore e armazena um valor. Retorna true se conseguiu criar, false caso contrário

Operações do TAD AB II

- Inserir o nó a direita de um nó
  - Pré-condição: nó não nulo.
  - Pós-condição: dado um nó, cria seu filho a direita e
  - armazena um valor. Retorna esse filho, se o mesmo pode ser criado, NULL caso contrário

- Inserir o nó a esquerda de um nó
  - Pré-condição: nó não nulo
  - Pós-condição: dado um nó, cria seu filho a esquerda e armazena um valor. Retorna esse filho, se o mesmo pode ser criado, NULL caso contrário

Operações do TAD AB

```
void criar(ARVORE_BINARIA *arv) {
 arv->raiz = NULL;
}
```

```
NO *criar_raiz(ARVORE_BINARIA *arv, INFO *info){
   arv->raiz = (NO*)malloc(sizeof(NO));
   if (arv->raiz != NULL) {
      arv->raiz->fesq = NULL;
      arv->raiz->fdir = NULL;
      arv->raiz->info = *info;
   }
   return arv->raiz;
}
```

Operações do TAD AB

```
NO *inserir_direita(NO *no, INFO *info) {
  if (no != NULL) {
    no->fdir = (NO*)malloc(sizeof(NO));
    no->fdir->fesq = NULL;
    no->fdir->fdir = NULL;
    no->fdir->info = *info;
   return no->fdir;
  }
 return NULL;
 }
```

Operações do TAD AB

```
NO *inserir_esquerda(NO *no, INFO *info) {
   if (no != NULL) {
    no->fesq = (NO*)malloc(sizeof(NO));
    no->fesq->fesq = NULL;
    no->fesq->fdir = NULL;
    no->fesq->info = *info;
    return no->fesq;
   }
 return NULL;
}
```

Percurso em Árvore Binária

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

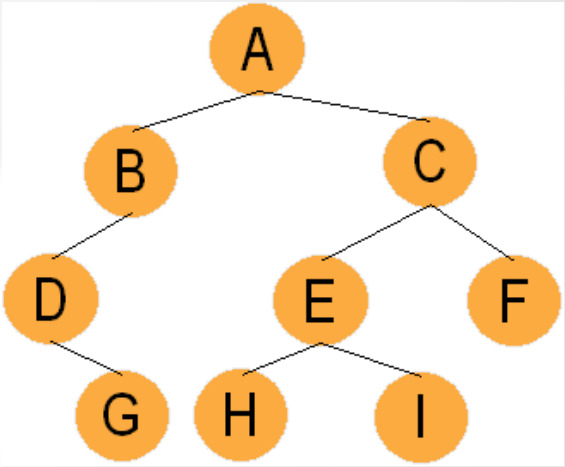
- Não existe um percurso único para árvores (binárias ou não): diferentes percursos podem ser realizados, dependendo da aplicação

AB - Percursos

- 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 em-ordem
    - visita a raiz
    - percorre a subárvore a direita em 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

```
void preordem_aux(NO *raiz) {
 if (raiz != NULL) {
  printf("%c\n", raiz->info.valor);
  preordem_aux(raiz->fesq);
  preordem_aux(raiz->fdir);
 }
}
```

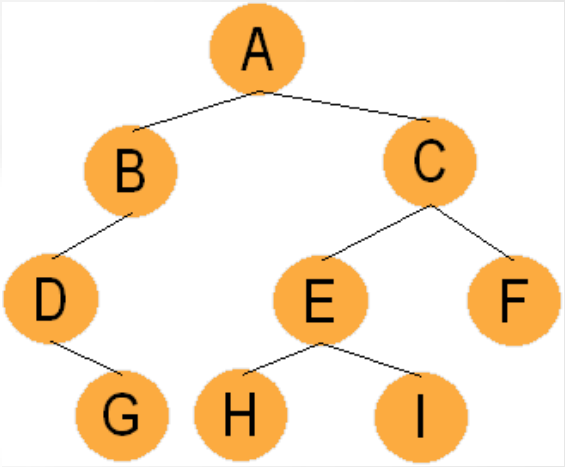


Resultado: ABDGCEHIF

```
void preordem(ARVORE_BINARIA *arv)
{
  preordem_aux(arv->raiz);
}
```

AB - Percurso Em-Ordem

```
void emordem_aux(NO *raiz) {
 if (raiz != NULL) {
  emordem_aux(raiz->fesq);
  printf("%c\n", raiz->info.valor);
  emordem_aux(raiz->fdir);
 }
}
```

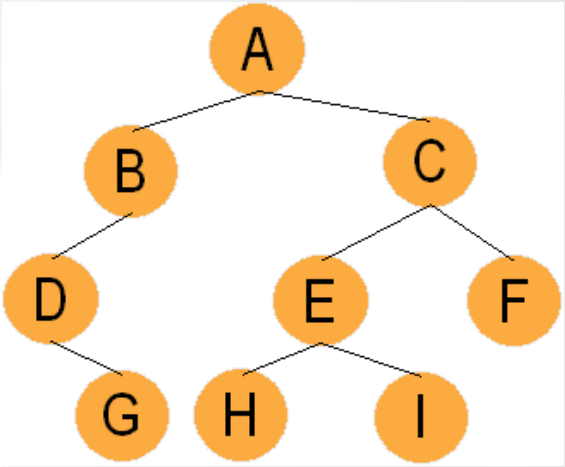


Resultado: DGBAHEICF

```
void emordem(ARVORE_BINARIA *arv) {
emordem_aux(arv->raiz);
}
```

AB - Percurso Pós-Ordem

```
void posordem_aux(NO *raiz) {
 if (raiz != NULL) {
  posordem_aux(raiz->fesq);
  posordem_aux(raiz->fdir);
  printf("%c\n", raiz->info.valor);
 }
}
```

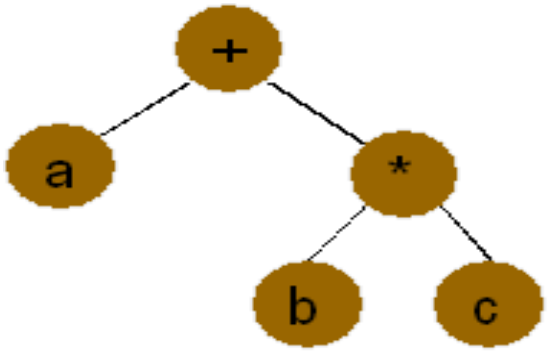


Resultado: GDBHIEFCA

```
void posordem(ARVORE_BINARIA *arv)
{
posordem_aux(arv->raiz);
}
```

AB - Percursos

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



- Em algoritmos iterativos utiliza-se uma pilha ou um campo a mais em cada nó para guardar o nó anterior (pai)

Exercícios

- Uma árvore binária completa cheia é uma árvore binária completa?
- Uma árvore estritamente binária é uma árvore binária Completa?
- Escreva um procedimento recursivo que calcula a altura de uma AB
- Escreva um procedimento recursivo que apaga uma árvore (executa free() em todos os nós)

Outras Operações sobre Árvores Binárias

Procedimento recursivo p/ destruir árvore, liberando o espaço alocado

```
void limpar_aux(NO *raiz) {
 if (raiz != NULL) {
   limpar_aux(raiz->fesq);
   limpar_aux(raiz->fdir);
 free(raiz);
 }
}
```

```
void limpar(ARVORE_BINARIA *arv) {
 limpar_aux(arv->raiz);
 arv->raiz = NULL;
}
```

Função recursiva para calcular altura de uma árvore

```
int altura_aux(NO *raiz) {
    if (raiz == NULL) {
     return -1;
    } else {
    int altesq = altura_aux(raiz->fesq);
    int altdir = altura_aux(raiz->fdir);
    return ((altesq > altdir) ? altesq : altdir) + 1;
 }
}
```

```
int altura(ARVORE_BINARIA *arv) {
  return altura_aux(arv->raiz);
}
```

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