

# Árvores Binárias

## Estruturas de Dados I

Departamento de Computação

Universidade Federal de São Carlos (UFSCar)

# Sumário

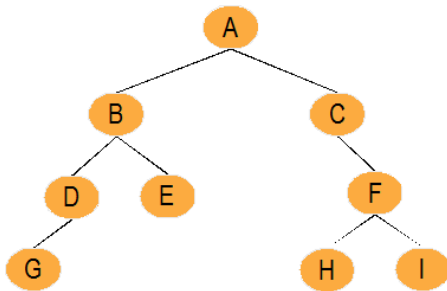
- 1 Conceitos Básicos
- 2 Implementação
- 3 Percurso em Árvore Binária
- 4 Outras Operações sobre Árvores Binárias

# Árvore Binárias

- Uma Árvore Binária (AB)  $T$  é um conjunto finito de elementos, denominados nós ou vértices, tal que
  - 1 Se  $T = \emptyset$ , 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

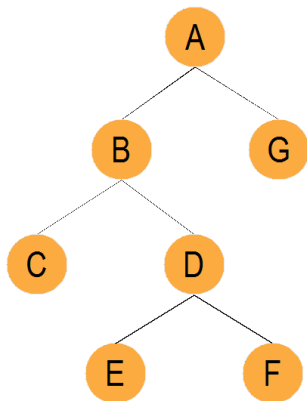
# Árvore Binárias

- 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 Binária

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

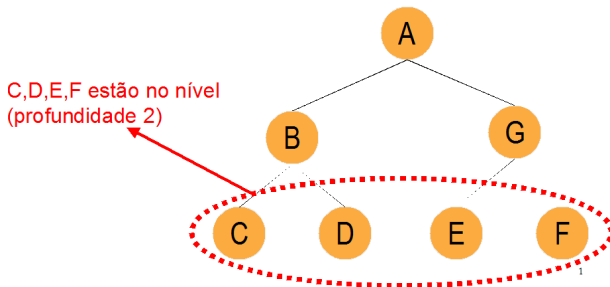


# Árvore Binária Completa

- **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)
  - $d = 1$ : 2 nós (total 3 nós)
  - $d = 2$ : 4 nós (total 7 nós)
  - ...
  - Profundidade  $d$ :  $2^d$  nós (total  $2^{d+1} - 1$  nós)

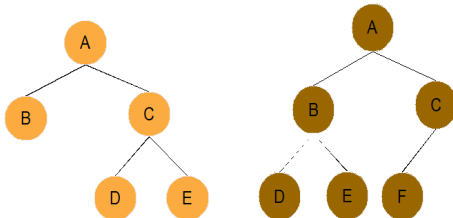


# Árvore Binária Completa Cheia

- Portanto, 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

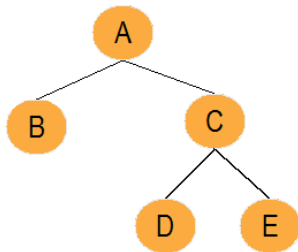
- **Árvore Binária Balanceada (ABB)**
  - Para cada nó, as alturas de suas duas sub-árvores diferem de, no máximo, 1



## Árvore Binária Perfeitamente Balanceada

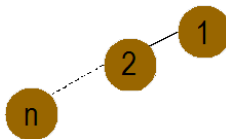
- **Árvore Binária Perfeitamente Balanceada**

- 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



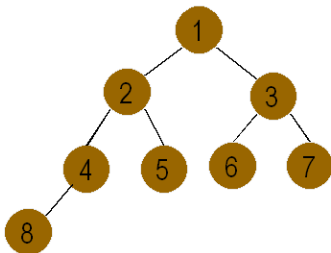
# Questões

- Qual a altura máxima de uma AB com  $n$  nós?
  - Resposta:  $n - 1$
  - Árvore degenerada  $\equiv$  Lista



# Questões

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



$n=1; \quad h=0$

$n=2,3; \quad h=1$

$n=4..7; \quad h=2$

$n=8..15; \quad h=3$

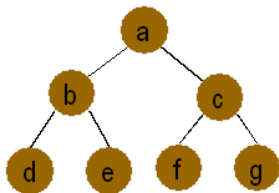
$$h_{min} = \lfloor \log_2 n \rfloor$$

# Sumário

- 1 Conceitos Básicos
- 2 Implementação**
- 3 Percurso em Árvore Binária
- 4 Outras Operações sobre Árvores Binárias

## 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)

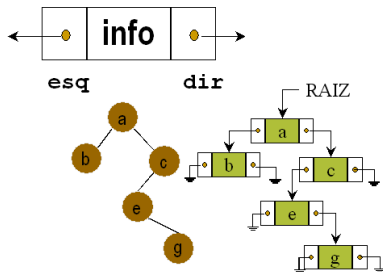
- Para um vetor indexado a partir da posição 0, se um nó está na posição  $i$ , seus filhos diretos estão nas posições
  - $2i + 1$  : filho da esquerda
  - $2i + 2$  : filho da direita
- **Vantagem:** espaço só p/ armazenar conteúdo; ligações implícitas
- **Desvantagem:** espaços vagos se árvore não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



# Implementação de AB (dinâmica)

- Para qualquer árvore, cada nó é do tipo

```
typedef struct {  
    int chave;  
    char valor;  
} INFO;  
  
typedef struct NO {  
    INFO info;  
    struct NO *fesq;  
    struct NO *fdir;  
} NO;  
  
typedef struct {  
    NO *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

```
1 void criar(ARVORE_BINARIA *arv) {
2     arv->raiz = NULL;
3 }
4
5 NO *criar_raiz(ARVORE_BINARIA *arv, INFO *info){
6     arv->raiz = (NO*)malloc(sizeof(NO));
7
8     if (arv->raiz != NULL) {
9         arv->raiz->fesq = NULL;
10        arv->raiz->fdire = NULL;
11        arv->raiz->info = *info;
12    }
13
14    return arv->raiz;
15 }
```

# Operações do TAD AB

```
1 NO *inserir_direita(NO *no, INFO *info) {
2     if (no != NULL) {
3         no->fdir = (NO*)malloc(sizeof(NO));
4         no->fdir->fesq = NULL;
5         no->fdir->fdir = NULL;
6         no->fdir->info = *info;
7         return no->fdir;
8     }
9     return NULL;
10 }
11
12 NO *inserir_esquerda(NO *no, INFO *info) {
13     if (no != NULL) {
14         no->fesq = (NO*)malloc(sizeof(NO));
15         no->fesq->fesq = NULL;
16         no->fesq->fdir = NULL;
17         no->fesq->info = *info;
18         return no->fesq;
19     }
20     return NULL;
21 }
```

# Sumário

- 1 Conceitos Básicos
- 2 Implementação
- 3 Percurso em Árvore Binária**
- 4 Outras Operações sobre Árvores Binárias

# 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

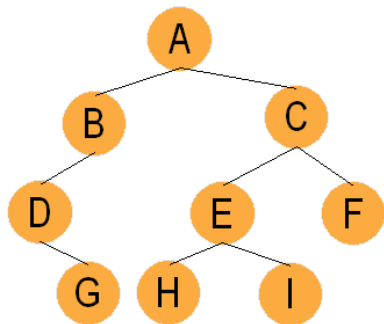
# AB - Percursos em Árvores

- 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 a subárvore a esquerda em em-ordem
    - visita a raiz
    - percorre a subárvore a direita em em-ordem
  - pós-ordem (Post-order)
    - percorre a 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

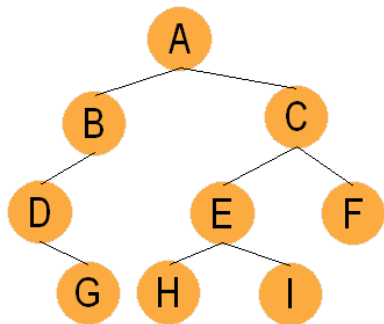
```
void preordem_aux(NO *raiz) {  
    if (raiz != NULL) {  
        printf("%c\n", raiz->info.valor);  
        preordem_aux(raiz->fesq);  
        preordem_aux(raiz->fdire);  
    }  
}  
  
void preordem(ARVORE_BINARIA *arv) {  
    preordem_aux(arv->raiz);  
}
```



• Resultado: ABDGCEHIF

# AB - Percurso Em-Ordem

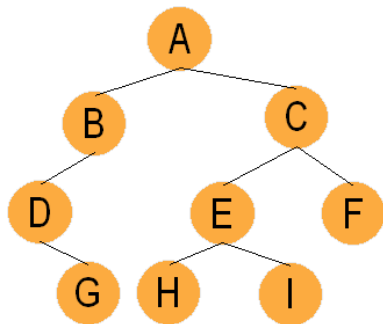
```
void emordem_aux(NO *raiz) {  
    if (raiz != NULL) {  
        emordem_aux(raiz->fesq);  
        printf("%c\n", raiz->info.valor);  
        emordem_aux(raiz->fdire);  
    }  
}  
  
void emordem(ARVORE_BINARIA *arv) {  
    emordem_aux(arv->raiz);  
}
```



• Resultado: DGBAHEICF

# AB - Percurso Pós-Ordem

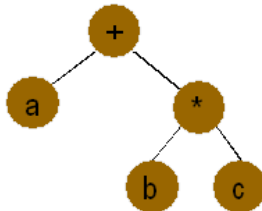
```
void posordem_aux(NO *raiz) {  
    if (raiz != NULL) {  
        posordem_aux(raiz->fesq);  
        posordem_aux(raiz->fdire);  
        printf("%c\n", raiz->info.valor);  
    }  
}  
  
void posordem(ARVORE_BINARIA *arv) {  
    posordem_aux(arv->raiz);  
}
```



• Resultado: GDBHIEFCA

# 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)

# Sumário

- 1 Conceitos Básicos
- 2 Implementação
- 3 Percurso em Árvore Binária
- 4 Outras Operações sobre Árvores Binárias

# Procedimento recursivo p/ destruir árvore, liberando o espaço alocado

```
1 void limpar_aux(NO *raiz) {  
2     if (raiz != NULL) {  
3         limpar_aux(raiz->fesq);  
4         limpar_aux(raiz->fdire);  
5         free(raiz);  
6     }  
7 }  
8  
9 void limpar(ARVORE_BINARIA *arv) {  
10     limpar_aux(arv->raiz);  
11     arv->raiz = NULL;  
12 }
```

# Função recursiva para calcular altura de uma árvore

```
1  int altura_aux(NO *raiz) {
2      if (raiz == NULL) {
3          return -1;
4      } else {
5          int altesq = altura_aux(raiz->fesq);
6          int altdir = altura_aux(raiz->fdir);
7          return ((altesq > altdir) ? altesq : altdir) + 1;
8      }
9  }
10
11 int altura(ARVORE_BINARIA *arv) {
12     return altura_aux(arv->raiz);
13 }
```



# 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

## Leitura Complementar

### 1. *Perfect binary tree*

<http://xlinux.nist.gov/dads//HTML/perfectBinaryTree.html>

### 2. *Binary tree*

<http://xlinux.nist.gov/dads//HTML/binarytree.html>

### 3. *Trees*

[http://www.gamedev.net/page/resources/\\_/technical/general-programming/...trees-part-1-r1374](http://www.gamedev.net/page/resources/_/technical/general-programming/...trees-part-1-r1374)

---

**Créditos:** aula baseada nos tópicos de aula e slides criados pelo prof. Fernando Vieira Paulovich, publicamente disponíveis em: [http://wiki.icmc.usp.br/index.php/Scs-202\(paulovich\)](http://wiki.icmc.usp.br/index.php/Scs-202(paulovich))