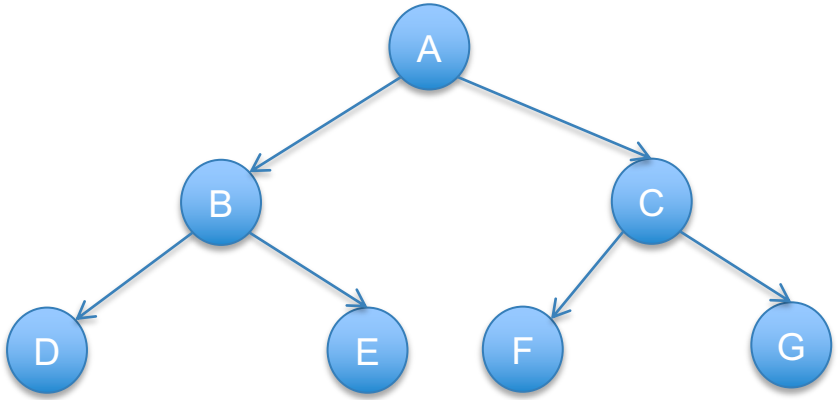
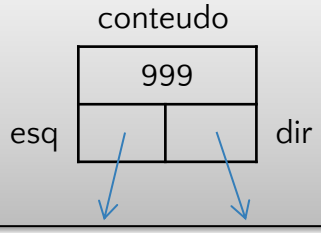
Universidade Federal de Goiás
Instituto de Informática
Prof. Ronaldo Martins da Costa

Árvores Binárias

- Uma árvore é chamada binária quando todo “nó não folha” possuir subárvore à esquerda e subárvore à direita




```
typedef struct reg {
      int conteudo;
      no *esq;
      no *dir;
   } no;
```



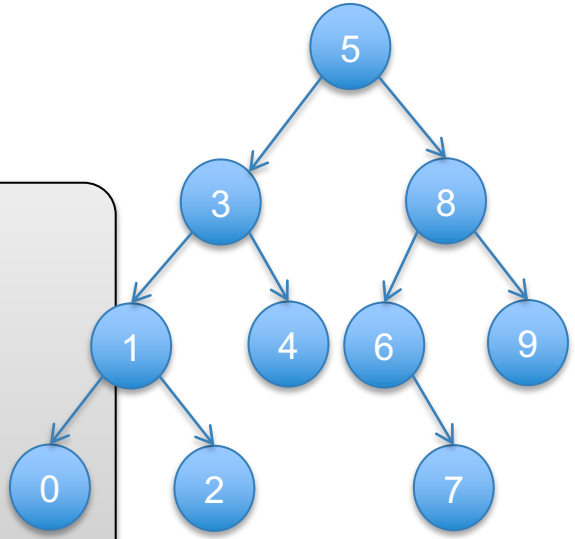
Árvores Binárias

- O campo conteudo contém a informação armazenada do nó
- Os dois outros campos servem apenas para dar estrutura à árvore
  - O campo esq e dir de cada nó contém NULL ou o endereço de outro nó

Árvores Binárias

- Operações em Árvores – Criar o nó Raiz

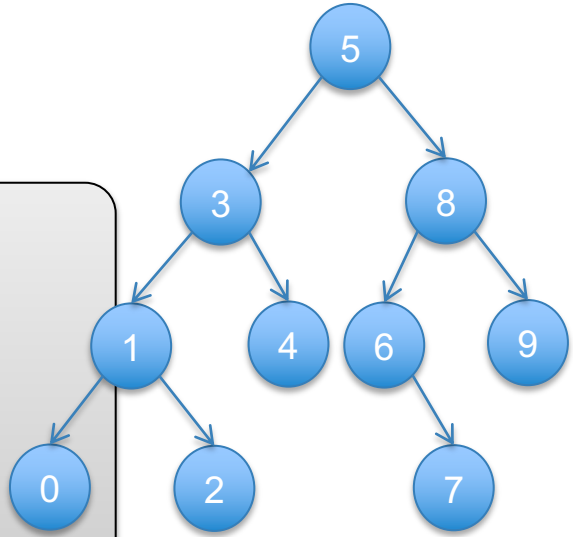
```
struct No {
    int numero;
    struct No *esquerda;
    struct No *direita;
};
typedef struct No No;

void criarArvore(No **pRaiz) {
    *pRaiz = NULL;
}

int main() {
    No *raiz;
    criarArvore(&raiz);
}
```



Árvores Binárias

- Operações em Árvores – Inserir nós

```
void insercao(No **pRaiz, int numero2){
    if (*pRaiz == NULL){
        *pRaiz = (No *)malloc(sizeof(No));
        (*pRaiz)->esquerda = NULL;
        (*pRaiz)->direita = NULL;
        (*pRaiz)->numero = numero2;
    } else {
        if (numero2 < ((*pRaiz)->numero)) {
            insercao(&((*pRaiz)->esquerda), numero2);
        } else {
            insercao(&((*pRaiz)->direita), numero2);
        }
    }
}
```

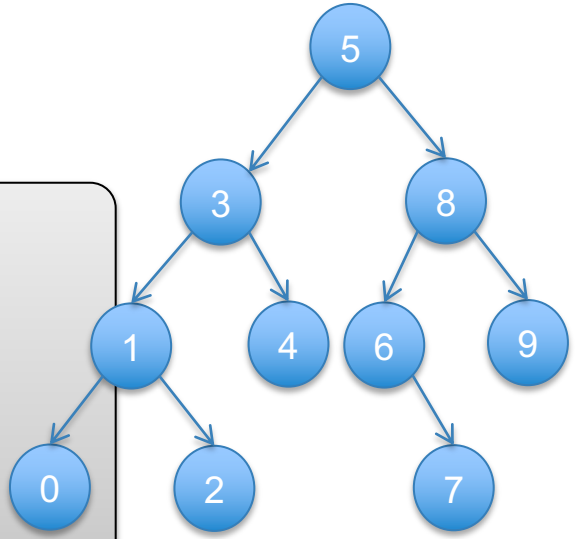


- Caso a árvore esteja disposta na ordem binária, a função mantém a ordem inserindo os novos elementos nas corretas posições

Árvores Binárias

- Operações em Árvores – Inserir nós

```
int main() {
    insercao(&raiz, 5);
    insercao(&raiz, 3);
    insercao(&raiz, 8);
    insercao(&raiz, 1);
    insercao(&raiz, 4);
    insercao(&raiz, 6);
    insercao(&raiz, 9);
    insercao(&raiz, 0);
    insercao(&raiz, 2);
    insercao(&raiz, 7);
}
```

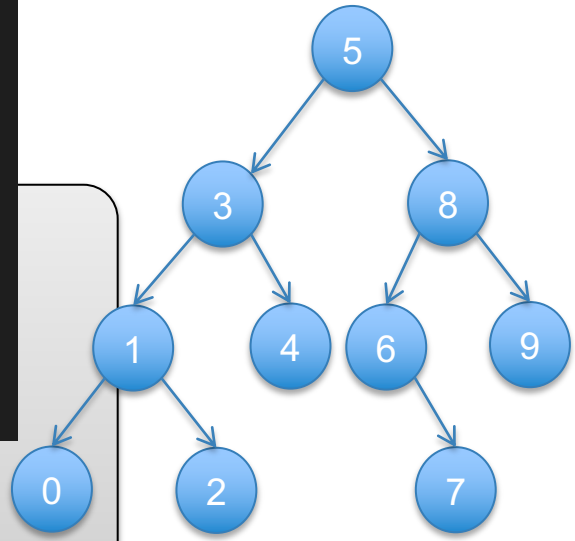


Árvores Binárias

- Operações em Árvores – Exibindo os nós

```
int main() {
printf("Nivel 0\n");
    printf("Raiz: %d\n", raiz->numero);
    printf("Nivel 1\n");
    printf(" %d\n", raiz->esquerda->numero);
    printf(" %d\n", raiz->direita->numero);
    printf("Nivel 2\n");
    printf(" %d\n", raiz->esquerda->esquerda->numero);
    printf(" %d\n", raiz->esquerda->direita->numero);
    printf(" %d\n", raiz->direita->esquerda->numero);
    printf(" %d\n", raiz->direita->direita->numero);
    printf("Nivel 3\n");
    printf(" %d\n", raiz->esquerda->esquerda->esquerda->numero);
    printf(" %d\n", raiz->esquerda->esquerda->direita->numero);
    printf(" %d\n", raiz->direita->esquerda->direita->numero);
}
```

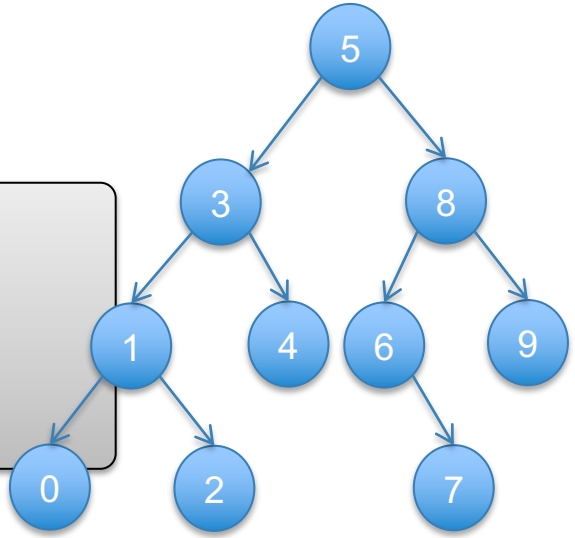
```
Nivel 0
Raiz: 5
Nivel 1
 3
 8
Nivel 2
 1
 4
 6
 9
Nivel 3
 0
 2
 7
```



Árvores Binárias

- Operações em Árvores – Percurso Em-Ordem

```
void exibirEmOrdem(No *pRaiz) {
    if (pRaiz != NULL) {
        exibirEmOrdem(pRaiz->esquerda);
        printf("\n%i", pRaiz->numero);
        exibirEmOrdem(pRaiz->direita);
    }
}
```



- Visita subárvore esquerda em em-ordem
- Visita a Raiz
- Visita subárvore direita em pós-ordem

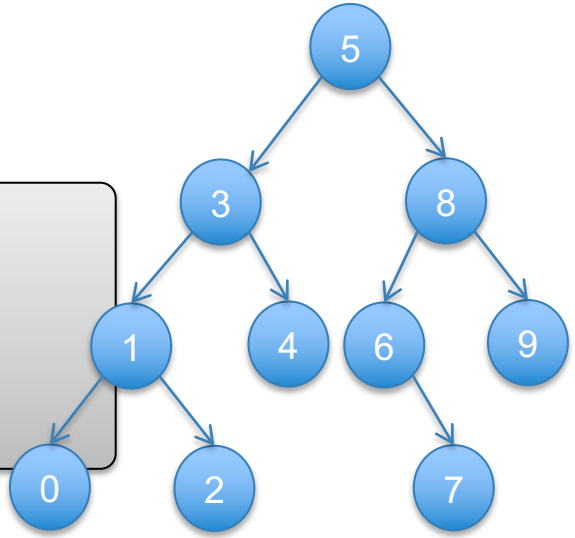
O resultado da função seria:

✓ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Árvores Binárias

- Operações em Árvores – Percurso Pré-Ordem

```
void exibirPreOrdem(No *pRaiz){
    if (pRaiz != NULL) {
        printf("\n%i", pRaiz->numero);
        exibirPreOrdem(pRaiz->esquerda);
        exibirPreOrdem(pRaiz->direita);
    }
}
```



- Visita a Raiz
- Visita subárvore esquerda em pré-ordem
- Visita subárvore direita em pré-ordem

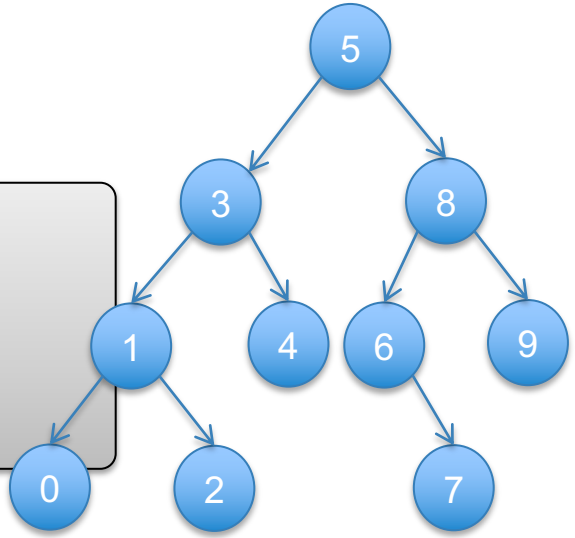
O resultado da função seria:

✓ 5, 3, 1, 0, 2, 4, 8, 6, 7, 9

Árvores Binárias

- Operações em Árvores – Percurso Pós-Ordem

```
void exibirPosOrdem(No *pRaiz) {
    if (pRaiz != NULL) {
        exibirPosOrdem(pRaiz->esquerda);
        exibirPosOrdem(pRaiz->direita);
        printf("\n%i", pRaiz->numero);
    }
}
```



- Visita subárvore esquerda em pós-ordem
- Visita subárvore direita em pós-ordem
- Visita a Raiz

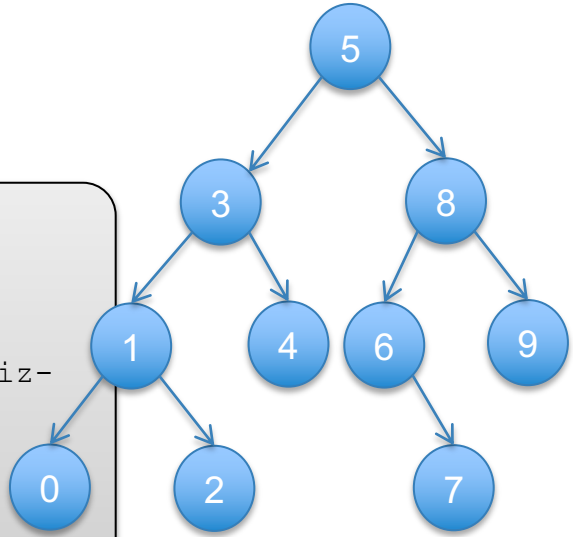
O resultado da função seria:

✓ 0, 2, 1, 4, 3, 7, 6, 9, 8, 5

Árvores Binárias

- Operações em Árvores – Contar os Nós e as Folhas

```
int contarNos(No *pRaiz) {
    if (pRaiz == NULL)
        return 0;
    else
        return 1 + contarNos(pRaiz->esquerda) + contarNos(pRaiz-
>direita);
}
int contarFolhas(No *pRaiz) {
    if (pRaiz == NULL)
        return 0;
    if (pRaiz->esquerda == NULL && pRaiz->direita == NULL)
        return 1;
    return contarFolhas(pRaiz->esquerda) + contarFolhas(pRaiz-
>direita);
}
```



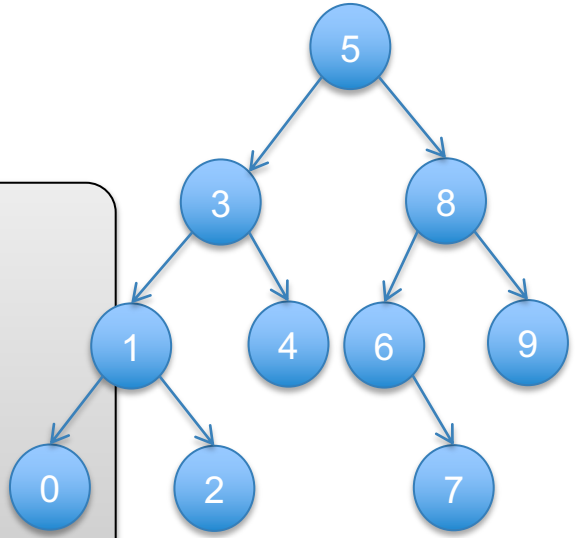
O resultado das funções seria:

✓ Nós: 10 e Folhas: 5

Árvores Binárias

- Operações em Árvores – Altura da Árvore

```
int maior(int a, int b) {
    if (a > b)
        return a;
    else
        return b;
} //maior
int altura(No *pRaiz) {
    if ((pRaiz == NULL) || (pRaiz->esquerda == NULL && pRaiz-
>direita == NULL))
        return 0;
    else
        return 1 + maior(altura(pRaiz->esquerda), altura(pRaiz-
>direita));
}
```



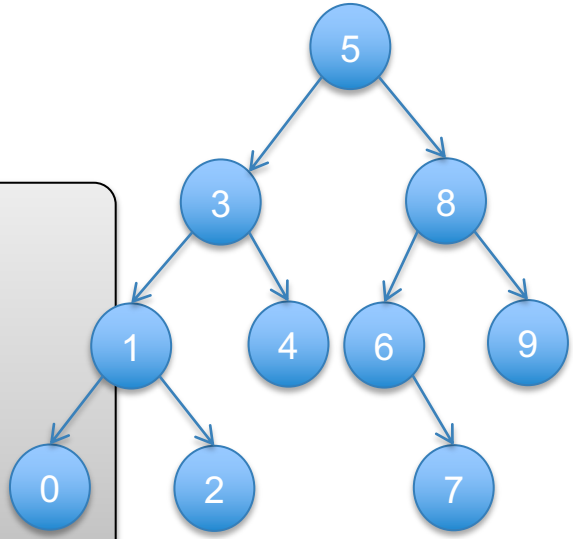
O resultado das funções seria:

✓ 3

Árvores Binárias

- Operações em Árvores – Busca

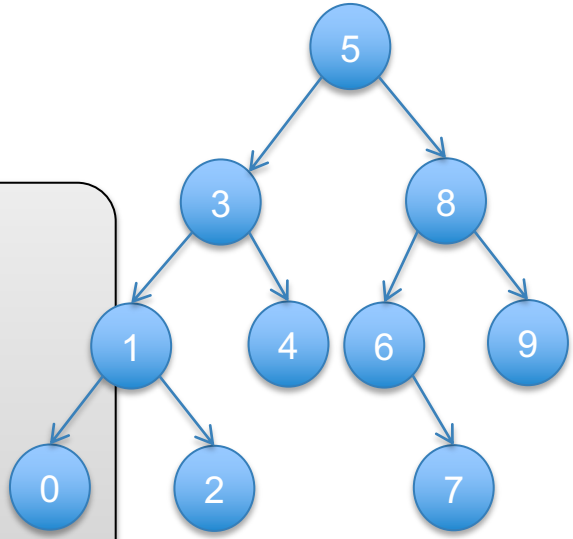
```
No *busca(No *praiz, int valor) {
    if (praiz == NULL)
        return NULL;
    if (valor == praiz->numero)
        return praiz;
    if (valor < praiz->numero)
        return busca(praiz->esquerda, valor);
    else
        return busca(praiz->direita, valor);
}
```



Árvores Binárias

- Operações em Árvores – Remoção

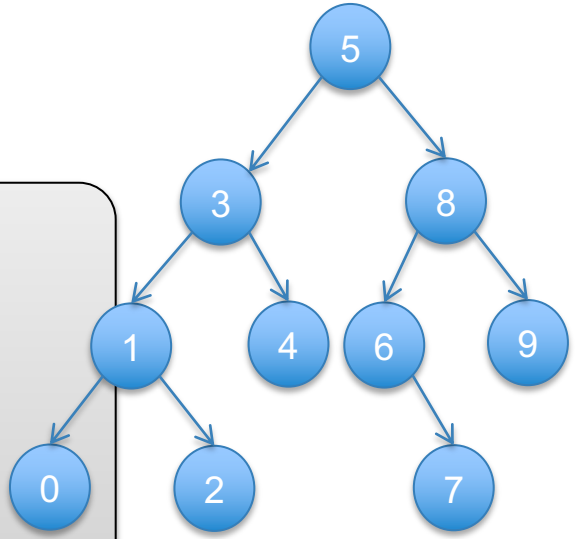
```
No *remover(No *r, int v) {
    if (r == NULL)
        return NULL;
    else if (r->numero > v)
        r->esquerda = remover(r->esquerda, v);
    else if (r->numero < v)
        r->direita = remover(r->direita, v);
    else {
        if (r->esquerda == NULL && r->direita == NULL) {
            free(r);
            r = NULL;
        }
        else if (r->esquerda == NULL) {
            No *t = r;
            r = r->direita;
            free(t);
        }
/* Continua... */
```



Árvores Binárias

- Operações em Árvores – Remoção

```
/* Continuação... */
        else if (r->direita == NULL) {
            No *t = r;
            r = r->esquerda;
            free(t);
        } else {
            No *f = r->esquerda;
            while (f->direita != NULL) {
                f = f->direita;
            }
            r->numero = f->numero; /* troca as informações */
            f->numero = v;
            r->esquerda = remover(r->esquerda, v);
        }
    }
    return r;
}
```



Universidade Federal de Goiás
Instituto de Informática
Prof. Ronaldo Martins da Costa