



Algoritmos e Estruturas de Dados I

Prof^a Priscilla Abreu
priscilla.abreu@ime.uerj.br
2022.1

Algoritmos e Estruturas de Dados I



Roteiro da aula

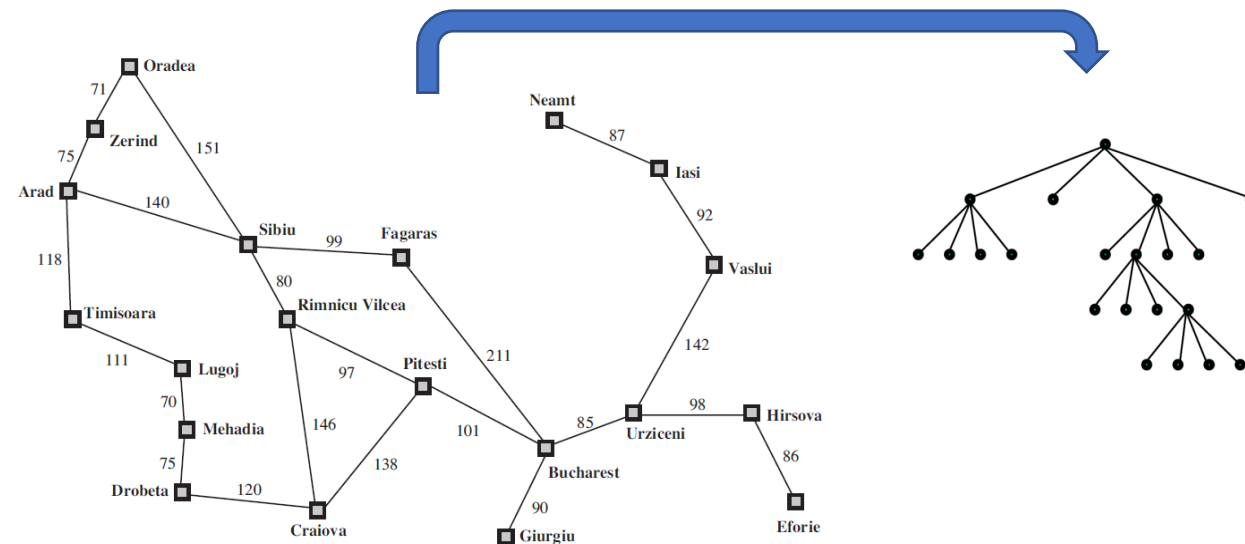
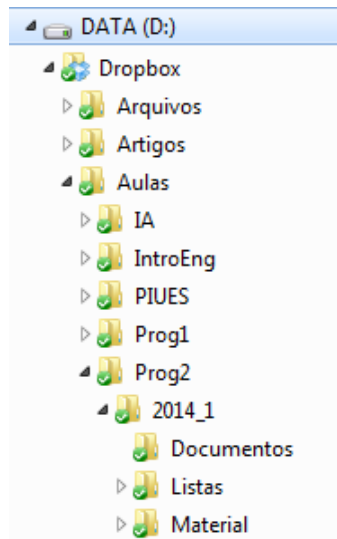
- Árvores
 - Implementação

Algoritmos e Estruturas de Dados I



ÁRVORE

Um estrutura de dados do tipo árvore permite que dados sejam organizados de forma hierárquica.

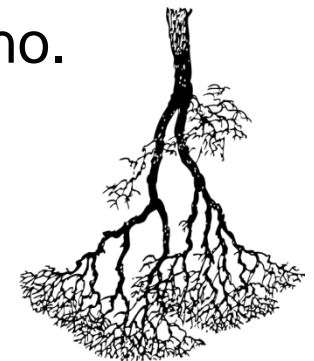
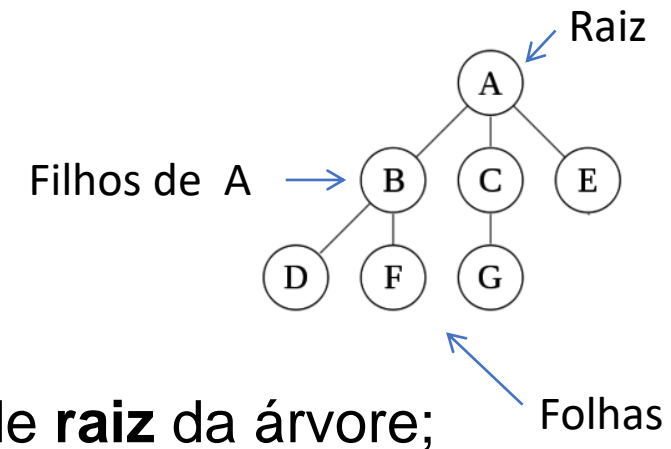


Algoritmos e Estruturas de Dados I



Árvores – conceitos

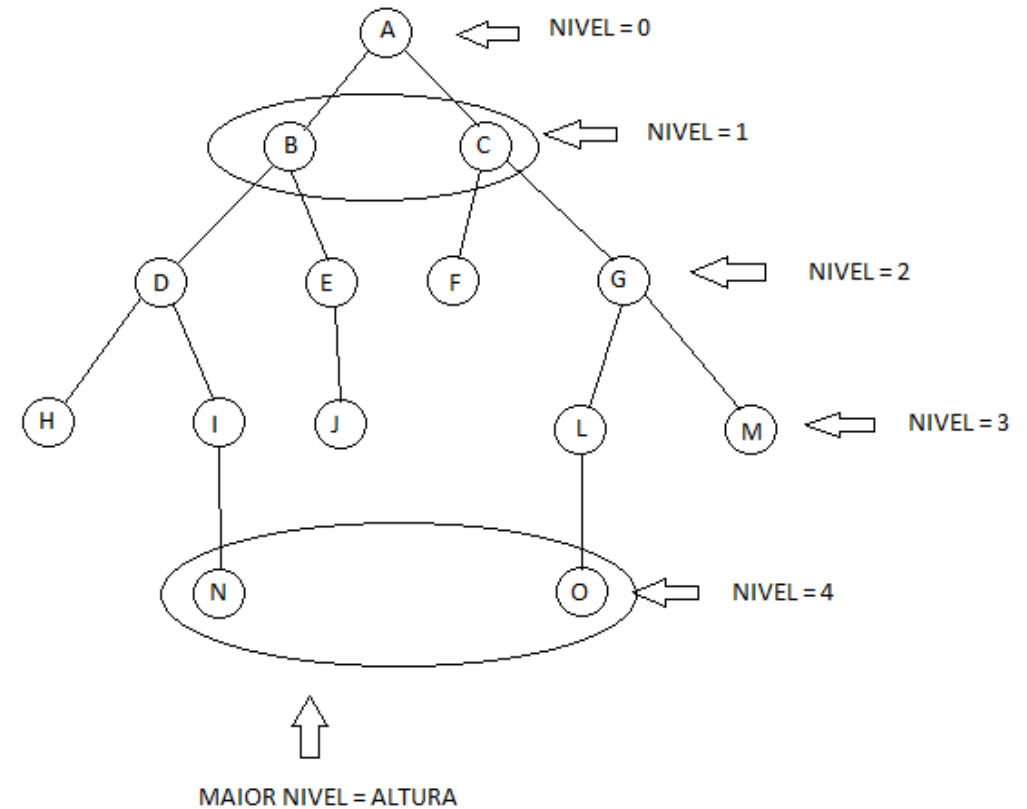
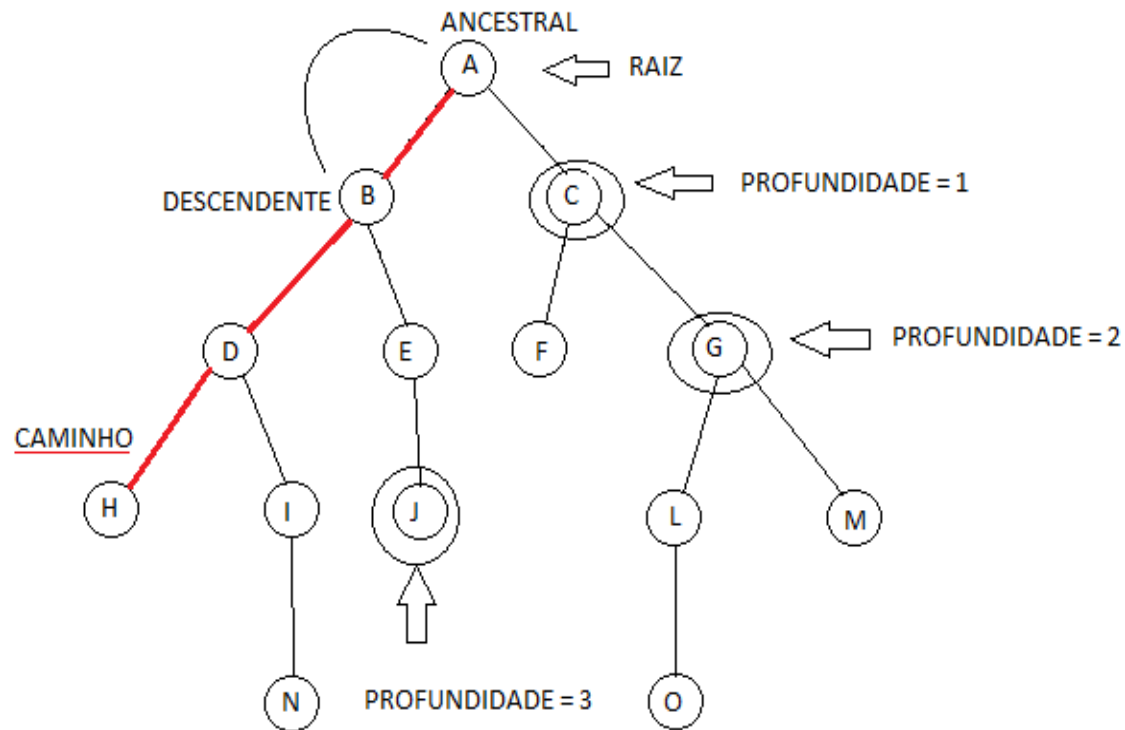
- Cada elemento de uma árvore é denominado **nó**;
- Toda árvore tem um elemento inicial que chamamos de **raiz** da árvore;
- Cada elemento da árvore pode ou não possuir nós abaixo dele hierarquicamente, denominados **filhos**.
- Os nós que não possuem filhos são denominados **folha** ou nó externo.
- Grau de um nó: número de filhos que ele possui.
- Grau da árvore: definido pelo nó de maior grau da árvore.



Algoritmos e Estruturas de Dados I



Árvores – conceitos

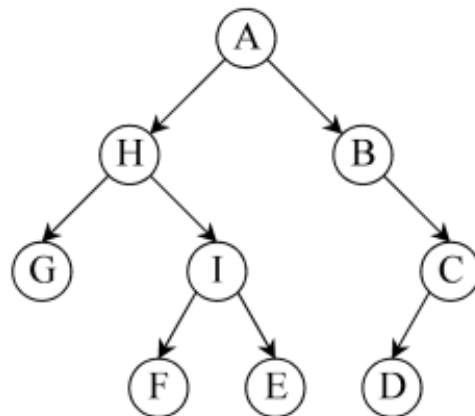


Algoritmos e Estruturas de Dados I



Árvores Binárias

Estrutura de dados que é constituída por um conjunto finito de nós, em que cada nó pode ter no máximo **dois** filhos, ou sub-árvores: a sub-árvore da **direita** (sad) e a sub-árvore da **esquerda** (sae).

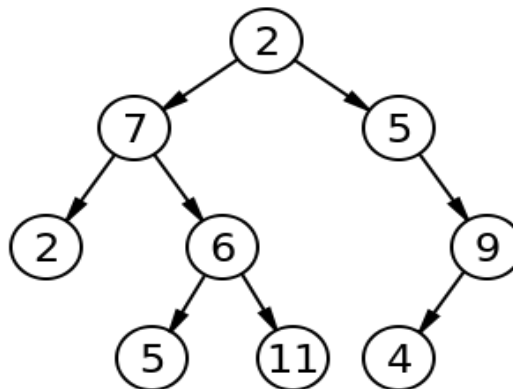


Árvore Binária – Percurso

- PRÉ ORDEM

No percurso em pré-ordem, primeiramente a **raiz** é visitada; depois, a sub-árvore **esquerda**; e finalmente, a sub-árvore **direita**.

No exemplo, o percurso seria feito na seguinte ordem: 2, 7, 2, 6, 5, 11, 5, 9 e 4.

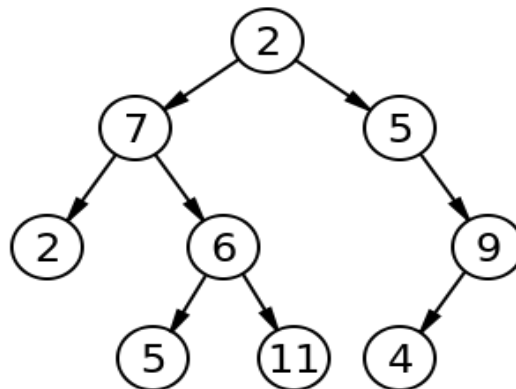


Árvore Binária – Percurso

- EM ORDEM (SIMÉTRICO)

No percurso simétrico (em ordem), primeiro é visitada a sub-árvore **esquerda**; logo após, a **raiz**; por final, a sub-árvore **direita**.

No exemplo, o percurso seria feito na seguinte ordem: 2, 7, 5, 6, 11, 2, 5, 4 e 9.

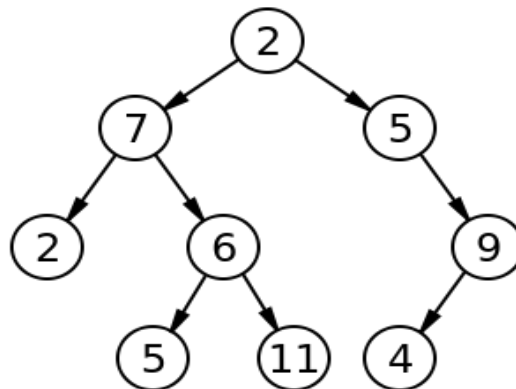


Árvore Binária – Percurso

- PÓS ORDEM

O percurso em pós-ordem inicia-se visitando a sub-árvore **esquerda**; em seguida, a sub-árvore **direita**; encerrando, a **raiz** é visitada.

No exemplo, o percurso seria feito na seguinte ordem: 2, 5, 11, 6, 7, 4, 9, 5 e 2.

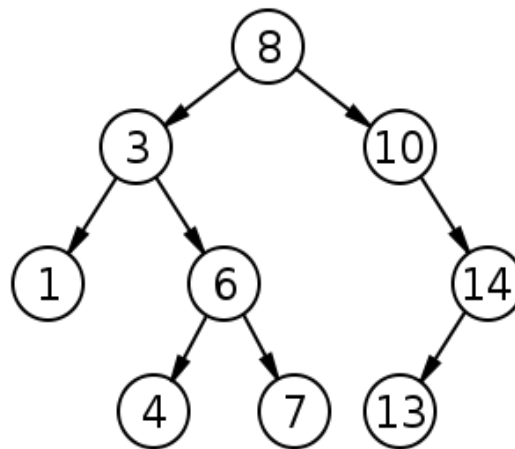


Algoritmos e Estruturas de Dados I



Árvore Binária de Busca

Árvore binária baseada em nós, onde todos os nós da subárvore esquerda possuem um valor numérico inferior ao nó raiz e todos os nós da subárvore direita possuem um valor superior ao nó raiz.





Como podemos definir uma *struct* para representar uma árvore binária?

Algoritmos e Estruturas de Dados I



```
#include <stdio.h>
#include <stdlib.h>
typedef struct no{
    int info;
    struct no *esq, *dir;
}no;
no *raiz;
```

Algoritmos e Estruturas de Dados I



```
no* inicializaArv(){  
    return NULL;  
}
```

Algoritmos e Estruturas de Dados I



```
no* criaNo(int valor){
    no* aux = (no*) malloc(sizeof(no));
    if(aux!=NULL){
        aux->info = valor;
        aux->esq = NULL;
        aux->dir = NULL;
    }
    return aux;
}
```





Como inserir um elemento na árvore?

Algoritmos e Estruturas de Dados I



- Árvore está vazia

raiz 

aux

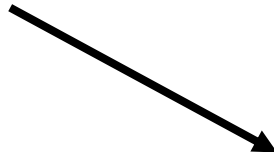
/	8	/
---	---	---

Algoritmos e Estruturas de Dados I



- Árvore está vazia

raiz



```
no *aux = criaNo(8);  
if (raiz == NULL) {  
    raiz = aux;  
}
```

Algoritmos e Estruturas de Dados I

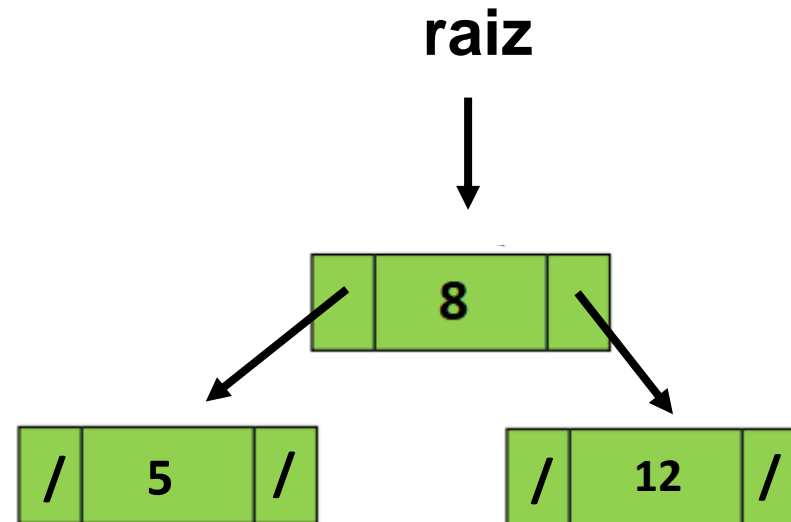


- Árvore não está vazia
 - Será necessário percorrer a árvore em busca de um nó sem o número máximo de filhos preenchidos (pela esquerda ou direita)

Algoritmos e Estruturas de Dados I



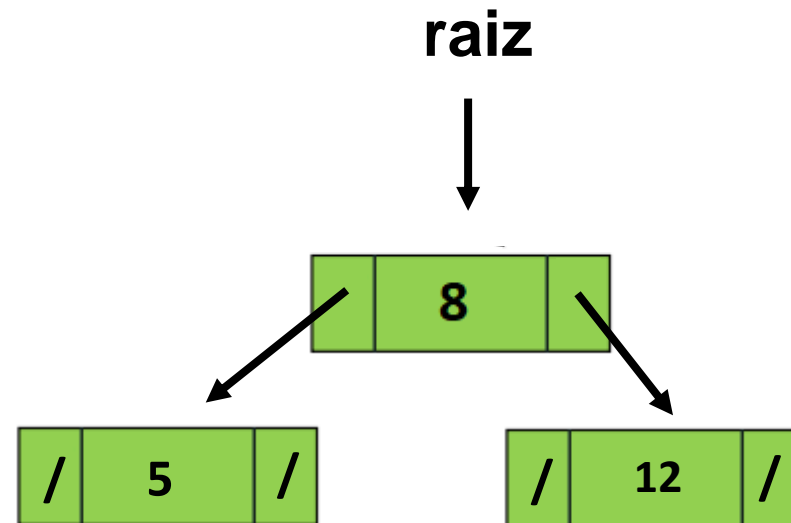
- Árvore não está vazia



Algoritmos e Estruturas de Dados I



- Árvore não está vazia

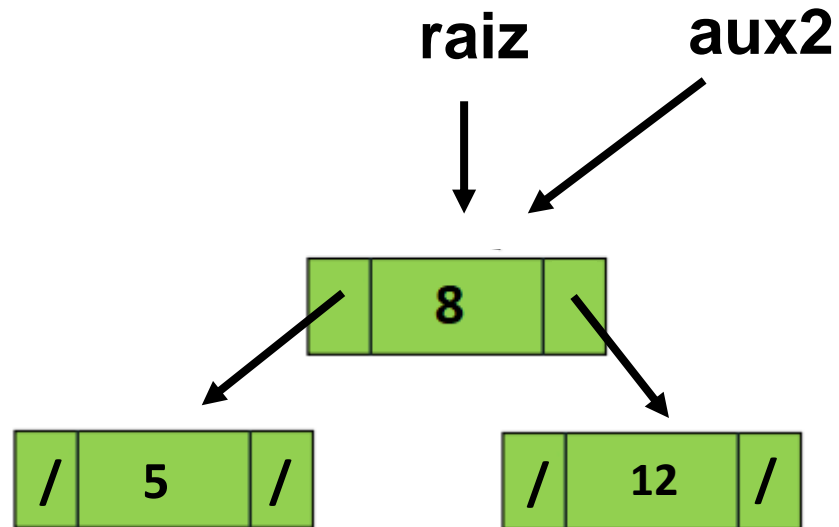


Precisaremos de uma variável auxiliar para encontrar o local a inserir o novo nó!

Algoritmos e Estruturas de Dados I



- Árvore não está vazia

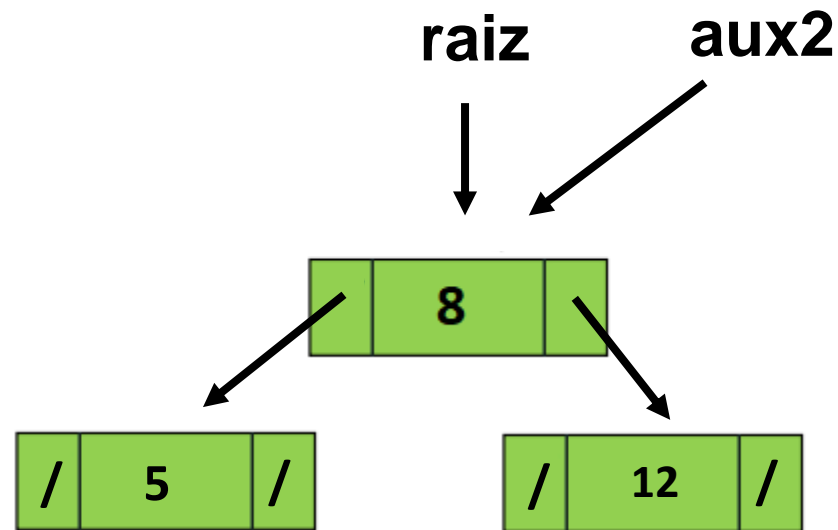


```
no *aux = criaNo(8);  
if (raiz == NULL) {  
    raiz = aux;  
}  
else{  
    no *aux2 = raiz;  
    ...  
}
```

Algoritmos e Estruturas de Dados I



- Árvore não está vazia

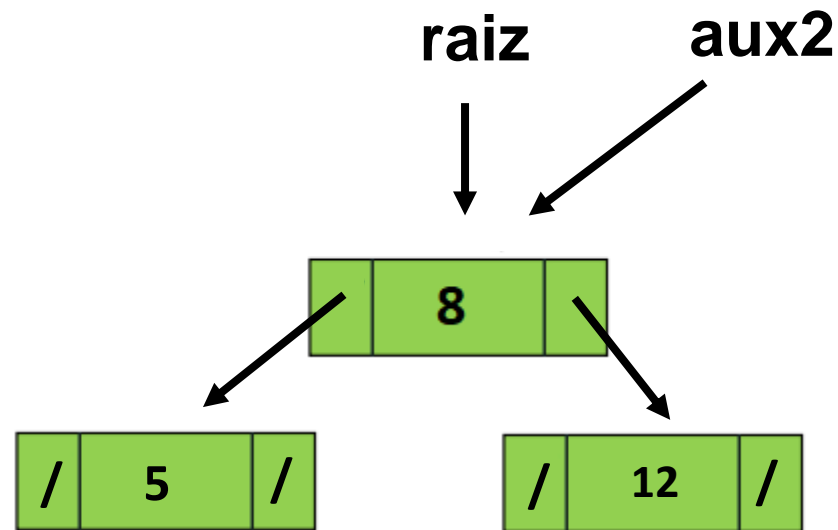


aux2 tem o campo
do filho à esquerda
vazio???

Algoritmos e Estruturas de Dados I



- Árvore não está vazia

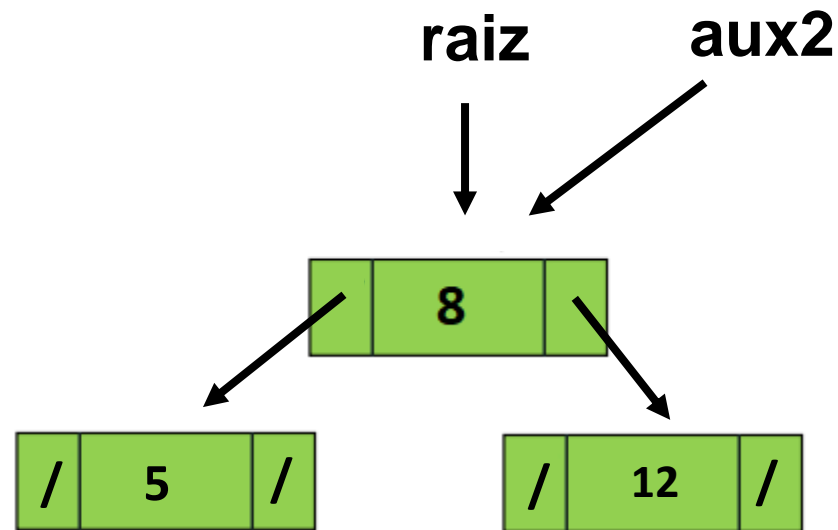


aux2 tem o campo
do filho à direita
vazio???

Algoritmos e Estruturas de Dados I



- Árvore não está vazia



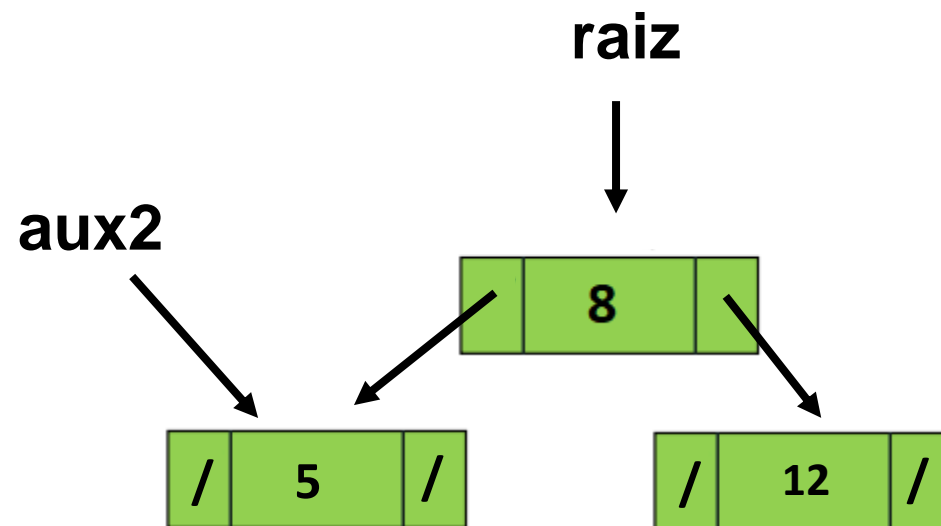
Vamos passar
aux2 para um
dos filhos

`aux2 = aux2 -> esq;`

Algoritmos e Estruturas de Dados I



- Árvore não está vazia

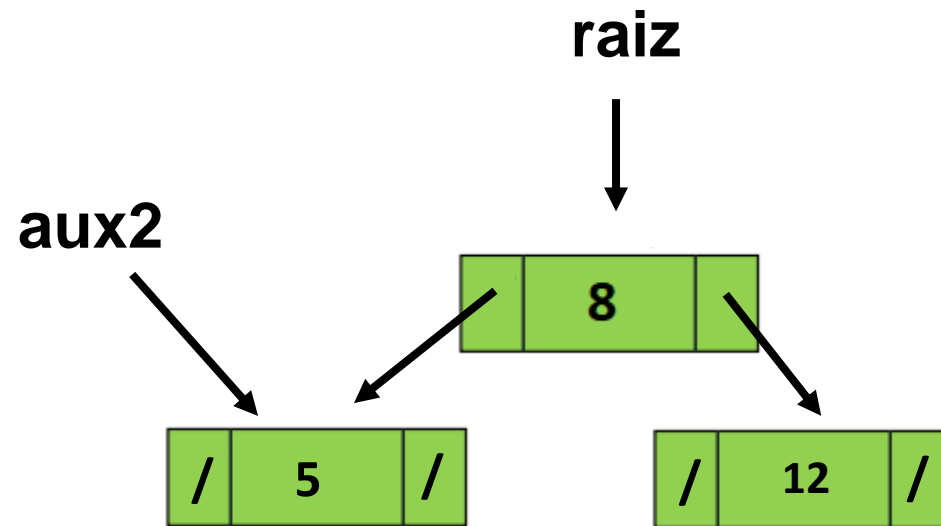


aux2 tem o campo
do filho à esquerda
vazio???

Algoritmos e Estruturas de Dados I



- Árvore não está vazia



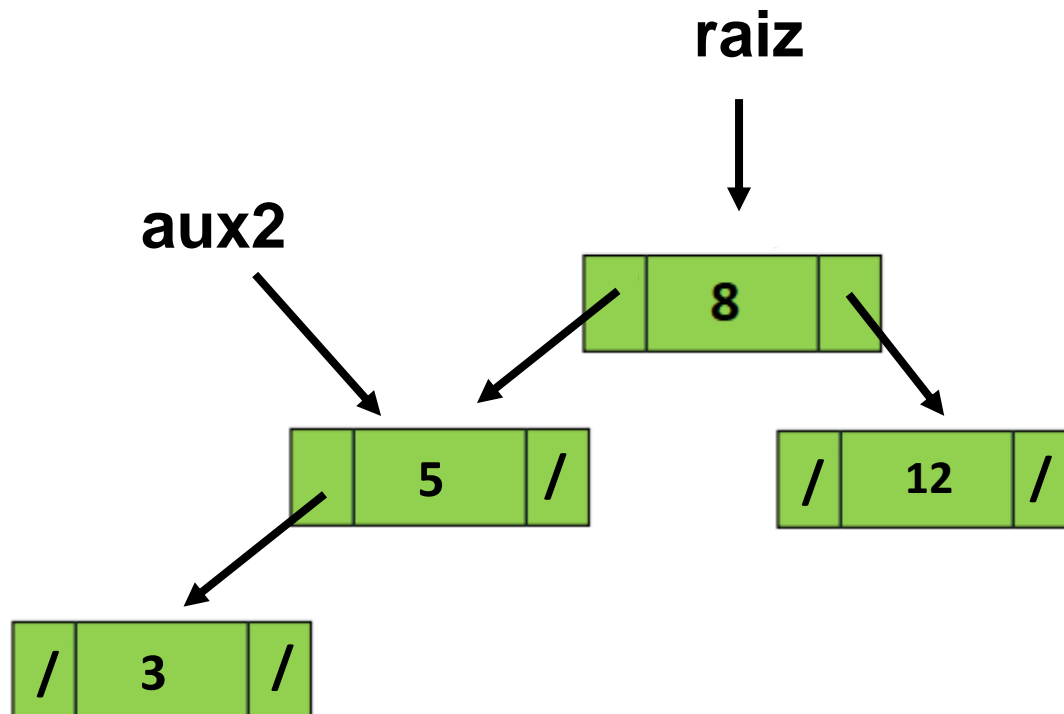
aux2 tem o campo
do filho à esquerda
vazio???

Podemos inserir
nessa posição!

Algoritmos e Estruturas de Dados I



- Árvore não está vazia



Algoritmos e Estruturas de Dados I



```
void insereNo(int valor){
    no *aux = criaNo(valor);
    if (arv==NULL){
        raiz = aux;
    }
    else{
        no *aux2 = raiz;
        while(aux2->esq!=NULL && aux2->dir!=NULL){
            aux2 = aux2->esq;
        }
        if(aux2->esq==NULL)
            aux2->esq = aux;
        else
            aux2->dir = aux;
    }
}
```

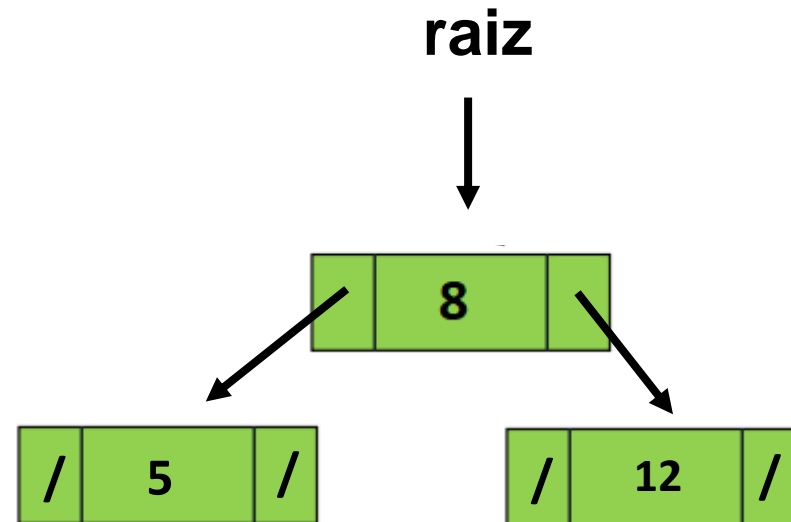


Como exibir os elementos de uma árvore binária?

Algoritmos e Estruturas de Dados I



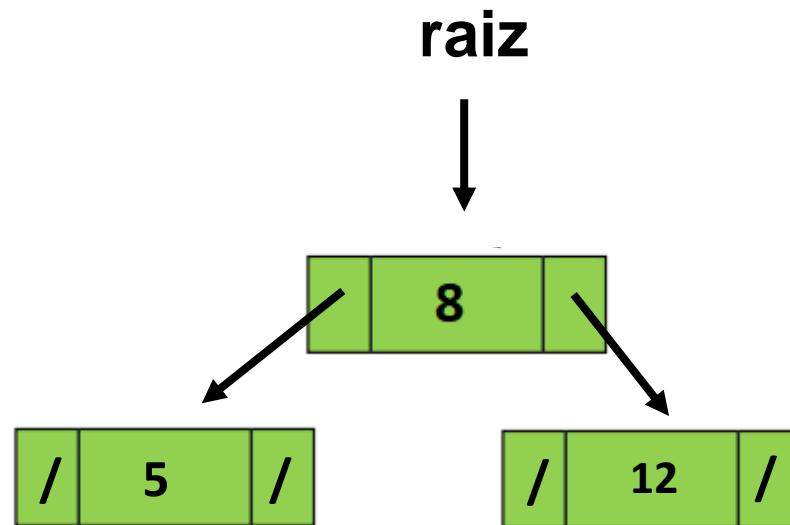
- Percorrer



Algoritmos e Estruturas de Dados I



- Percorrer

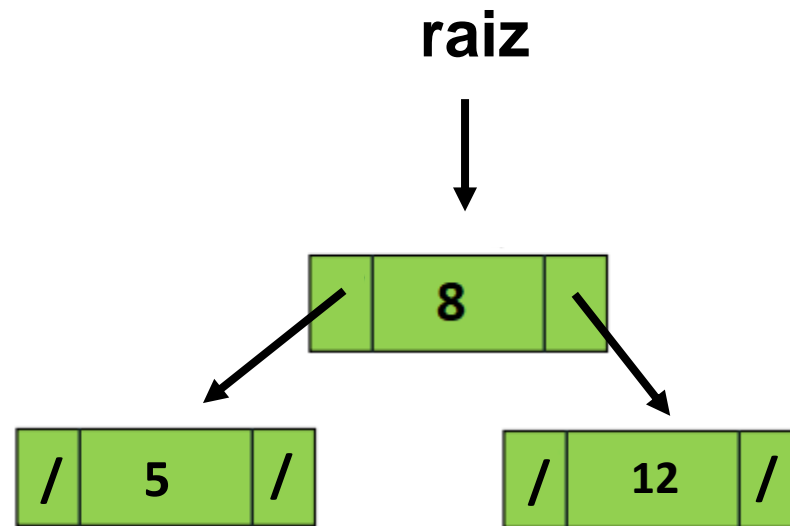


Utilizaremos uma função recursiva para percorrer a árvore!

Algoritmos e Estruturas de Dados I



- Percorrer

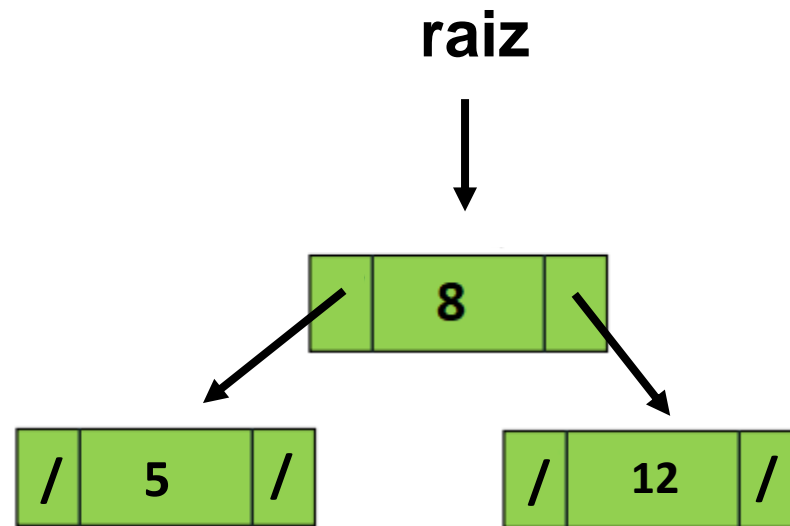


A raiz é diferente de NULL?

Algoritmos e Estruturas de Dados I



- Percorrer



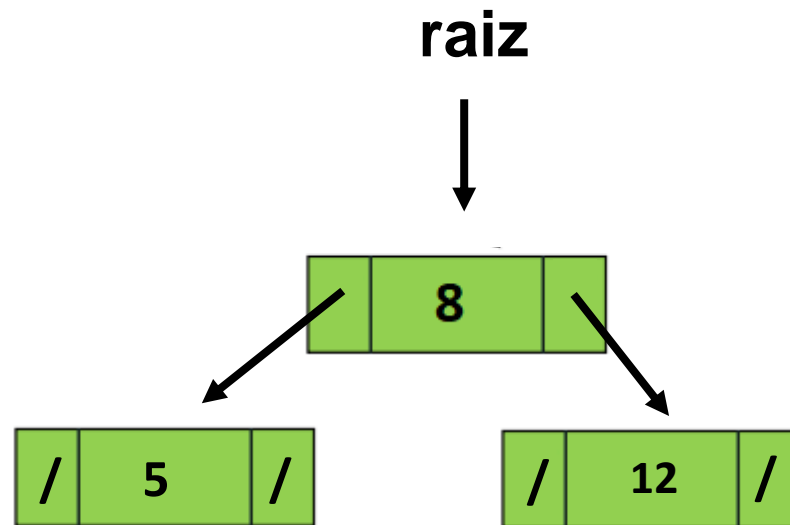
A raiz é diferente de NULL?

Posso exibir seu campo info!

Algoritmos e Estruturas de Dados I



- Percorrer

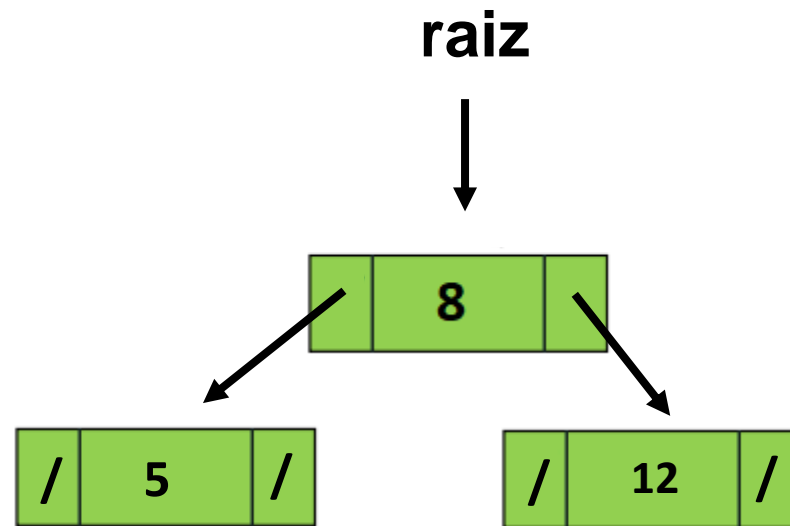


```
void imprime (no *raiz){  
    if (raiz!=NULL){  
        printf("%d ",raiz->info);  
        ...  
    }  
}
```

Algoritmos e Estruturas de Dados I



- Árvore não está vazia

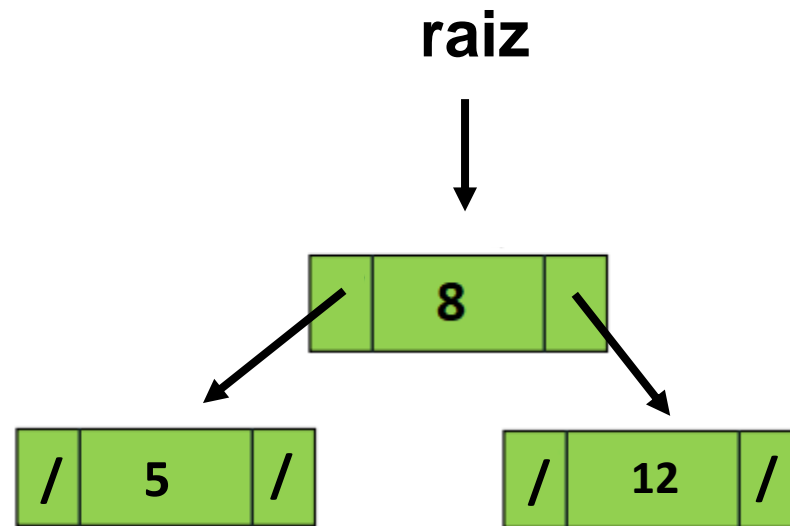


Repetimos o processo para a subárvore esquerda e depois para a subárvore direita.

Algoritmos e Estruturas de Dados I



- Árvore não está vazia



```
void imprime (no *raiz){  
    if (raiz!=NULL){  
        printf("%d ",raiz->info);  
        imprime(raiz->esq);  
        imprime(raiz->dir);  
    }  
}
```



Função principal

Algoritmos e Estruturas de Dados I



```
int main(){
    raiz = inicializaArv();
    int i,num;
    for(i=0;i<6;i++){
        printf("Informe um valor: ");
        scanf("%d",&num);
        insere(num);
    }
    imprime(arv);
}
```

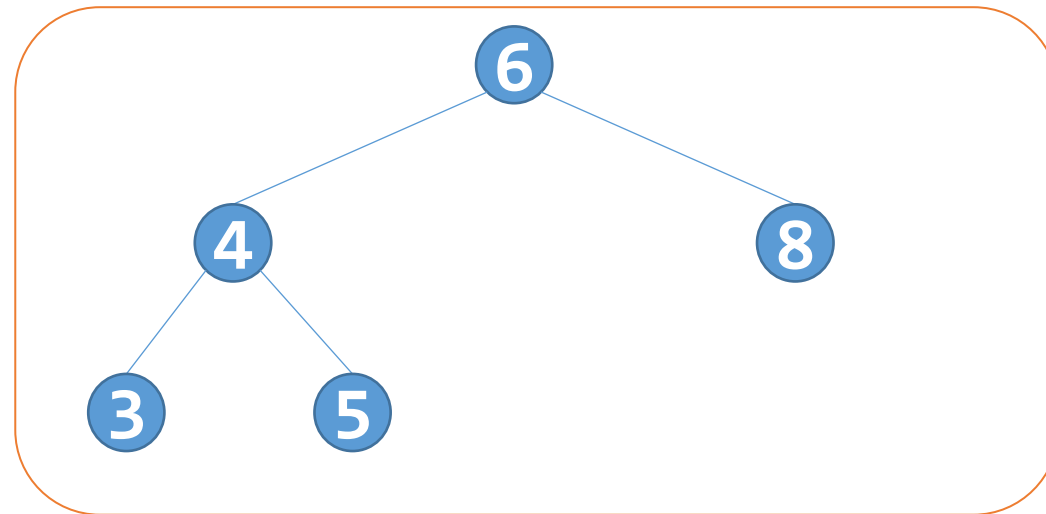


Percursos

Algoritmos e Estruturas de Dados I



- **Árvore**

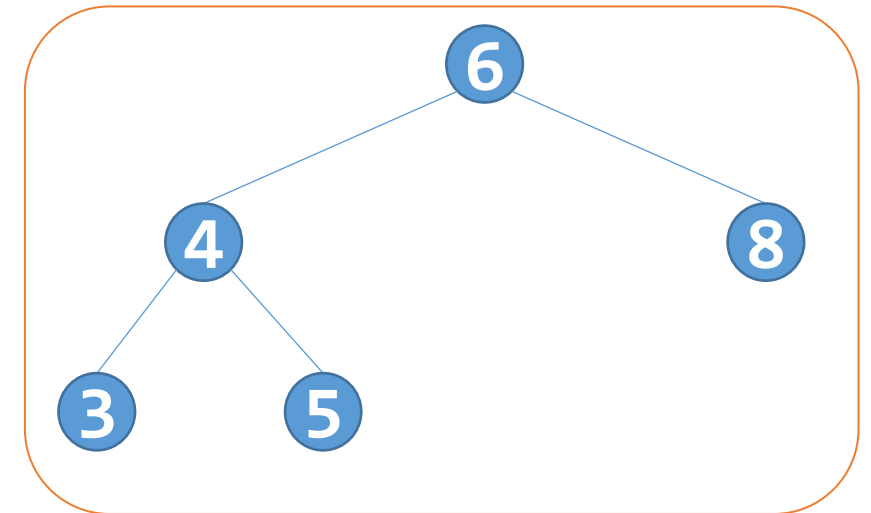


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

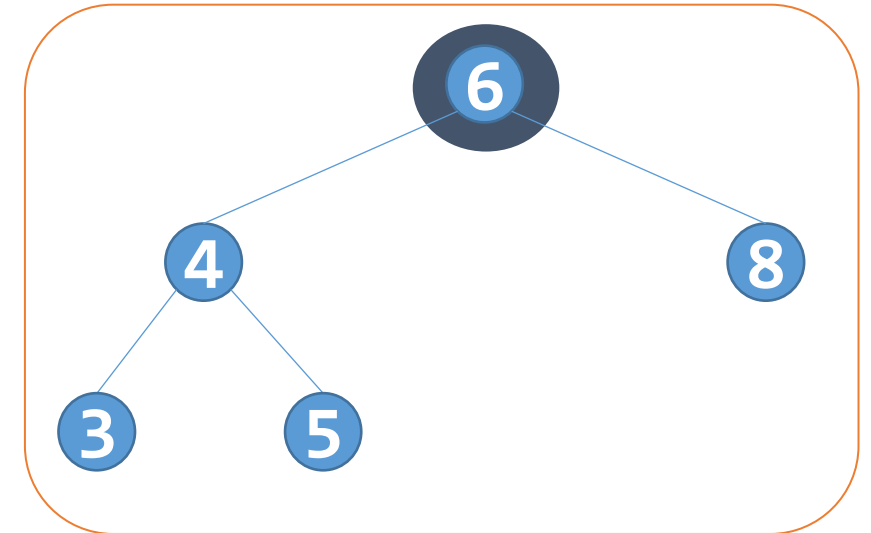


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        ➡ printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

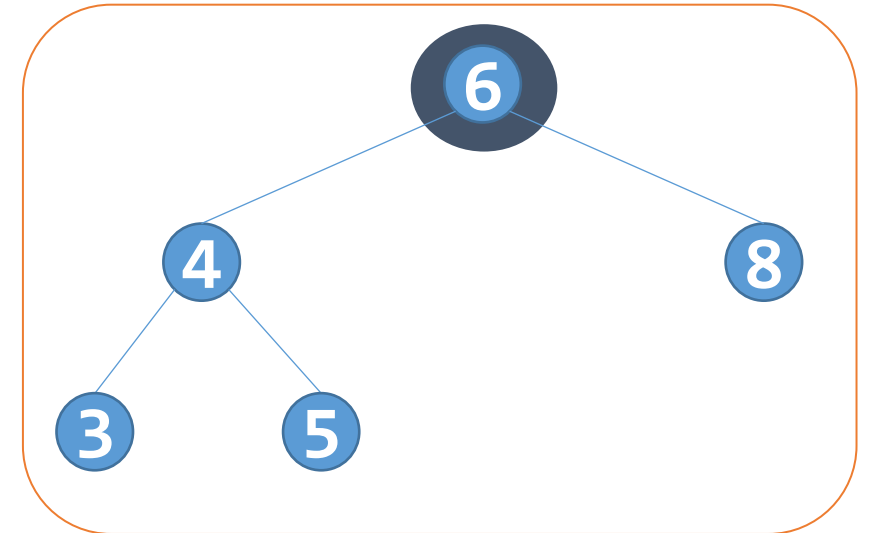


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        ➡ preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

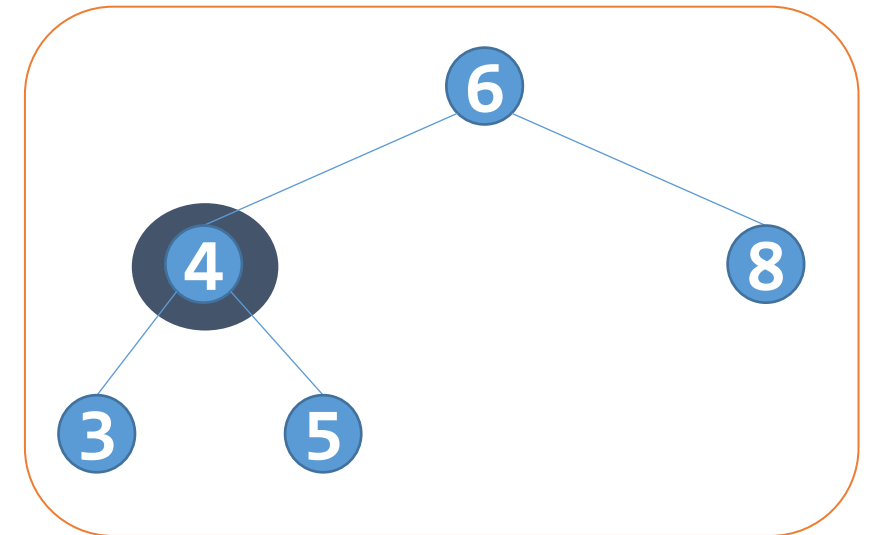


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    ➡ if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

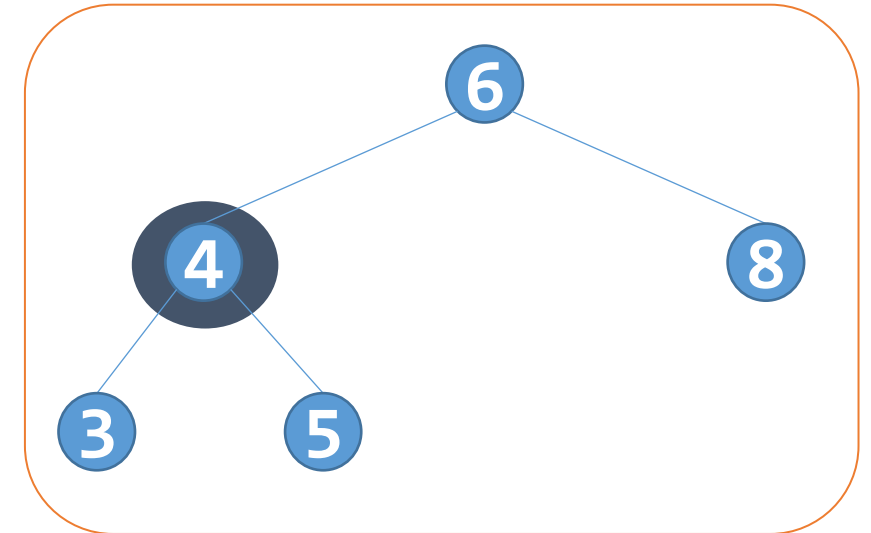


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        ➡ printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

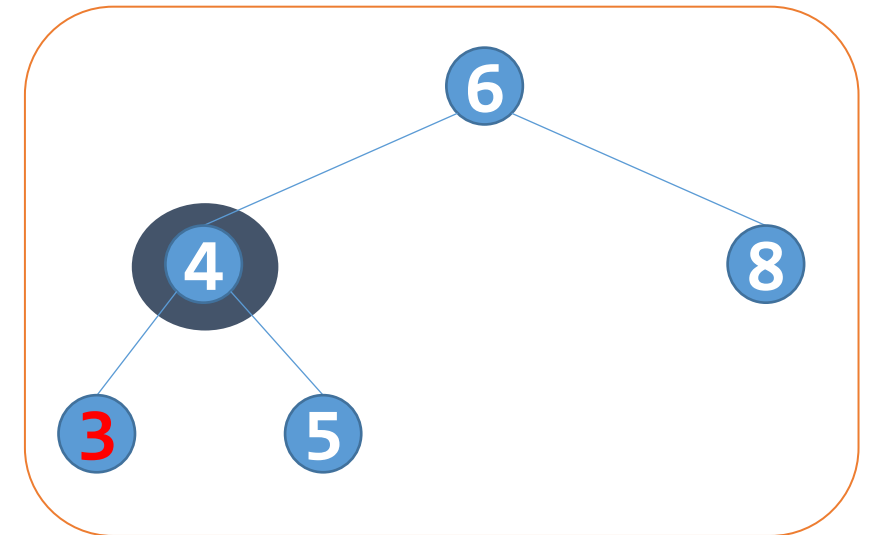


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        ➡ preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

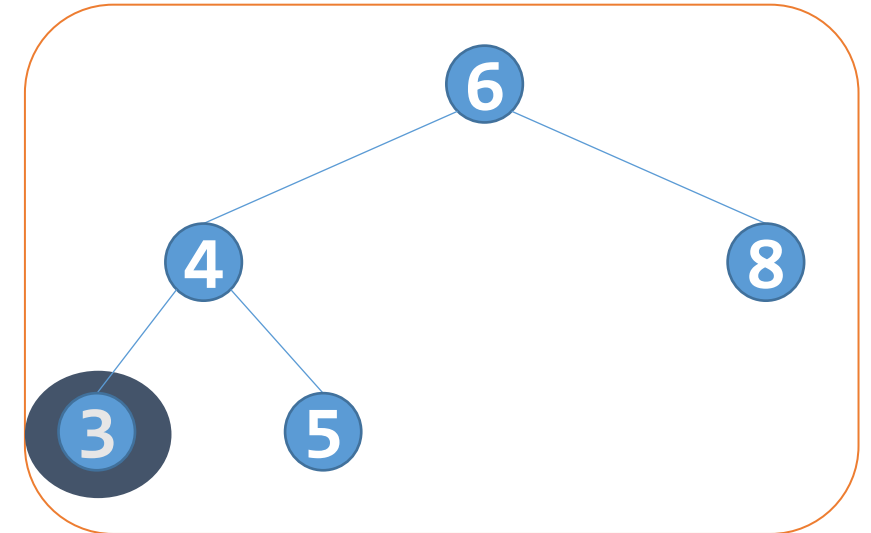


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    ➡ if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

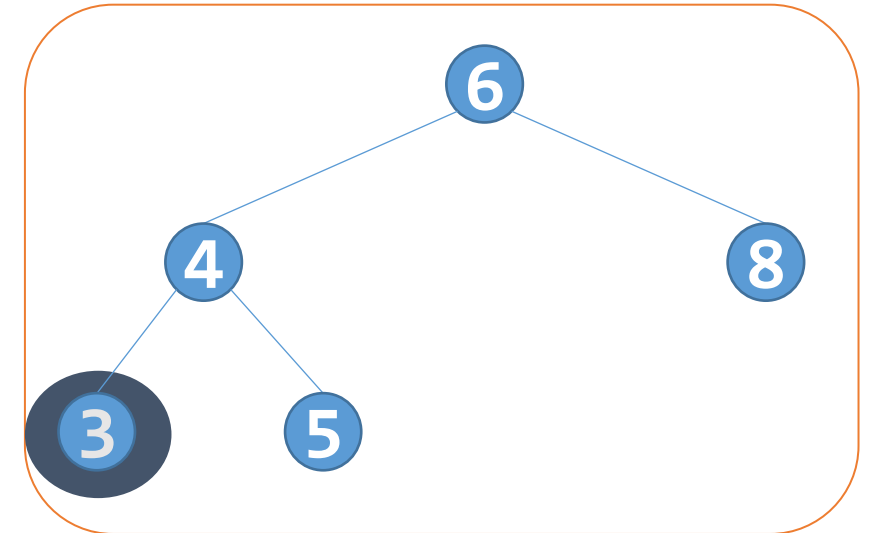


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        ➡ printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

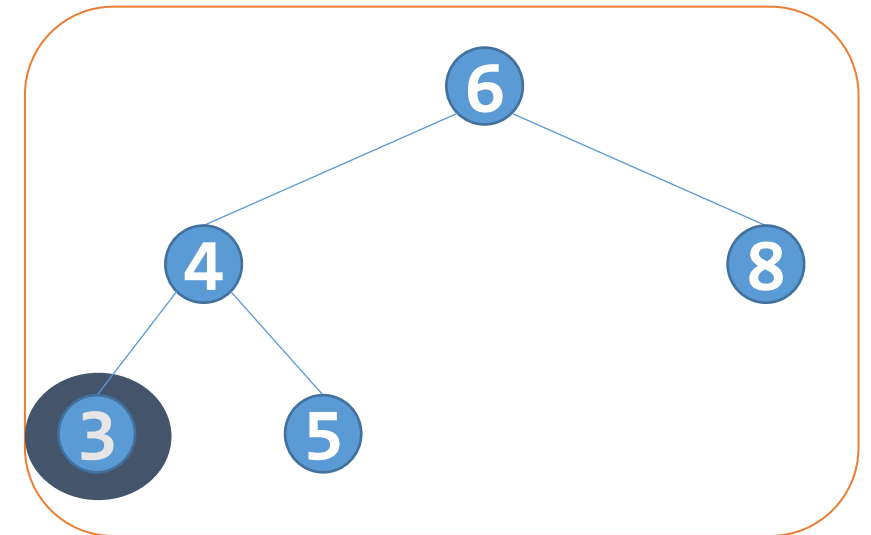


Algoritmos e Estruturas de Dados I



- **Pré-ordem**


```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        ➡ preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

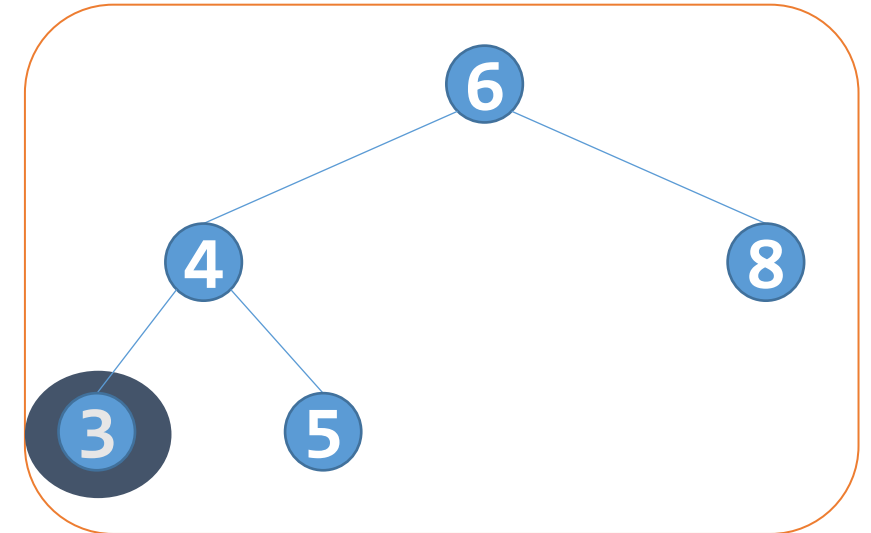


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        ➡ preOrdem(raiz->esq);   
        preOrdem(raiz->dir);  
    }  
}
```

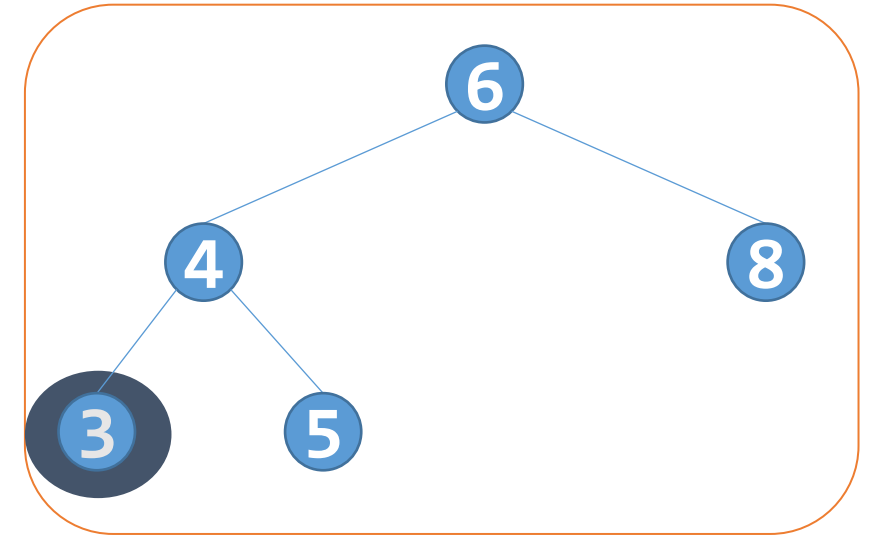


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        ➡ preOrdem(raiz->dir);  
    }  
}
```

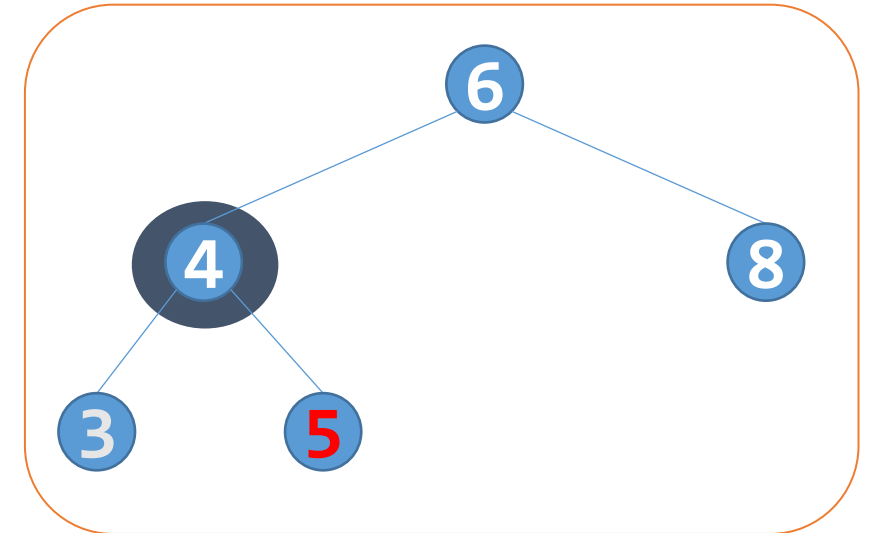


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        ➡ preOrdem(raiz->dir);  
    }  
}
```

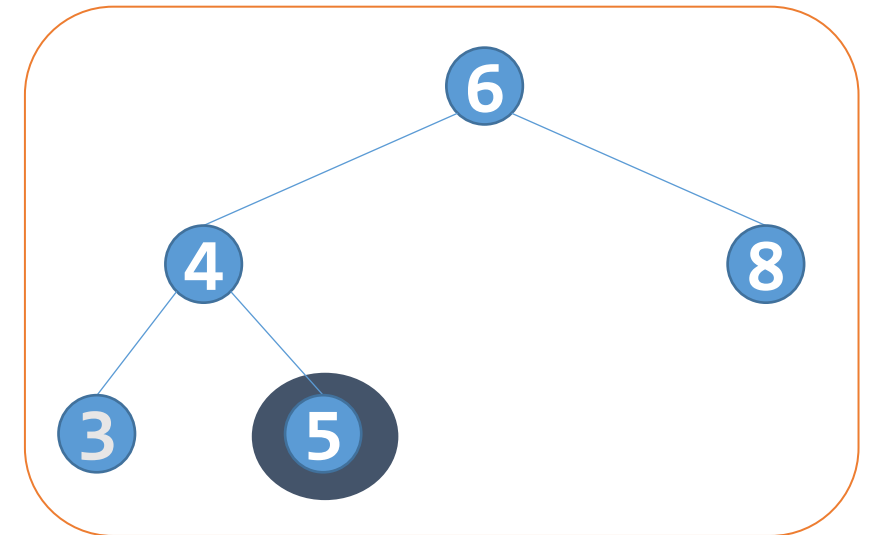


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    → if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

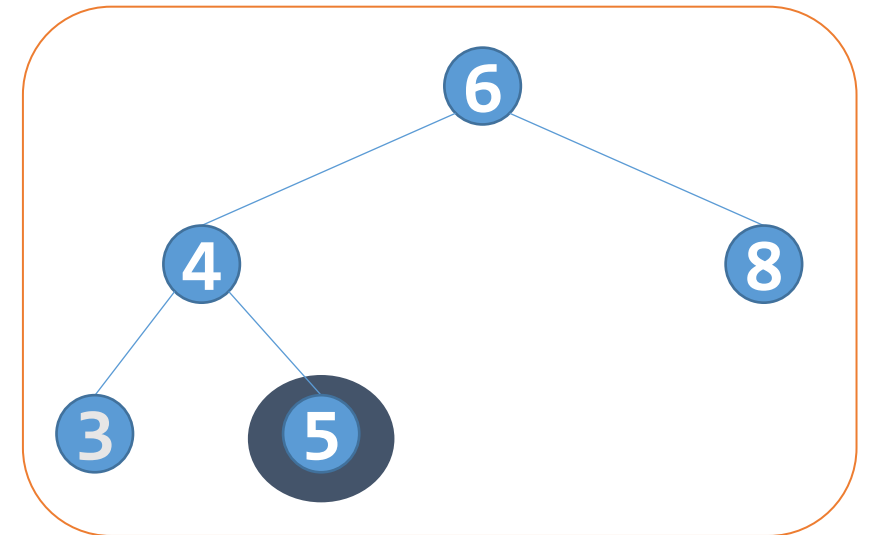


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        ➡ printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

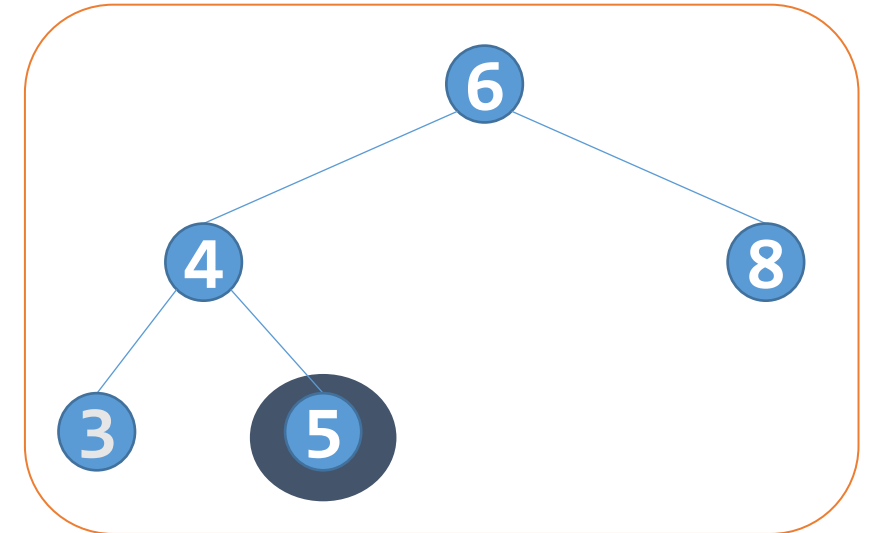


Algoritmos e Estruturas de Dados I



- **Pré-ordem**


```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        ➡ preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

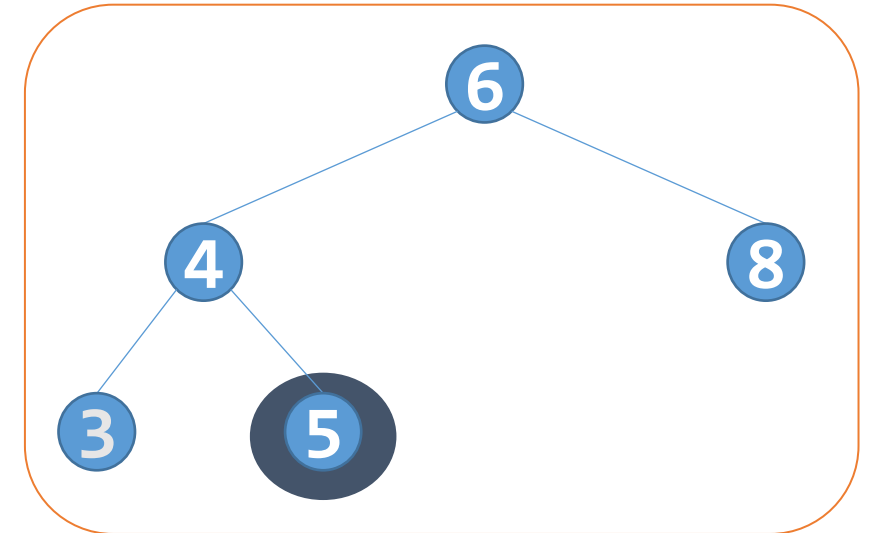


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        ➡ preOrdem(raiz->esq);   
        preOrdem(raiz->dir);  
    }  
}
```

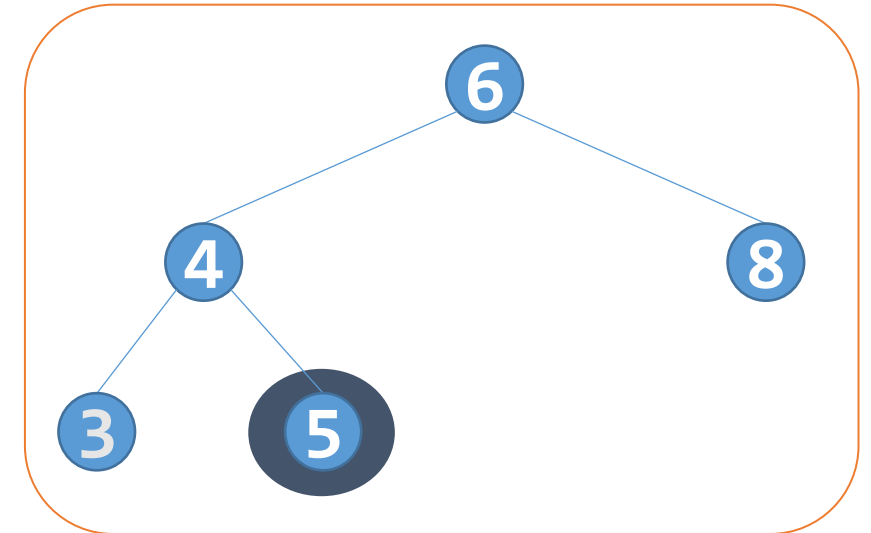


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        ➡ preOrdem(raiz->dir);  
    }  
}
```

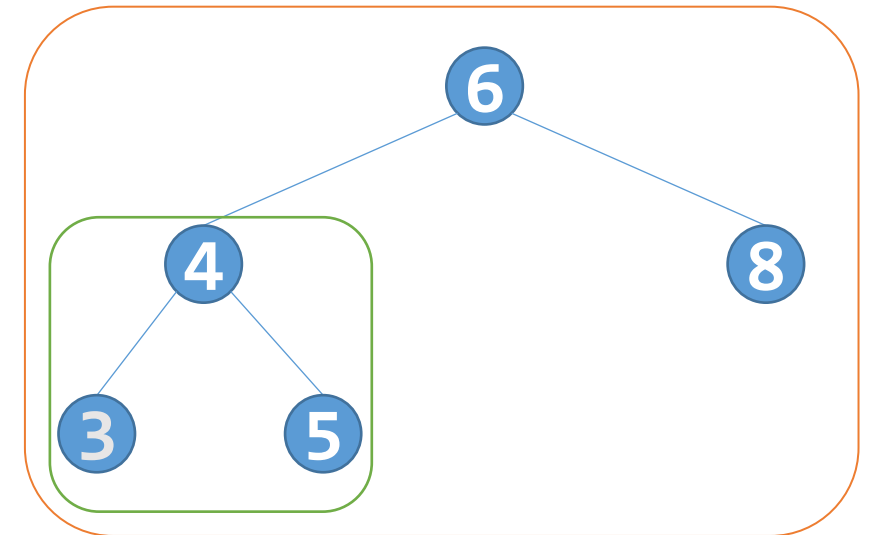


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

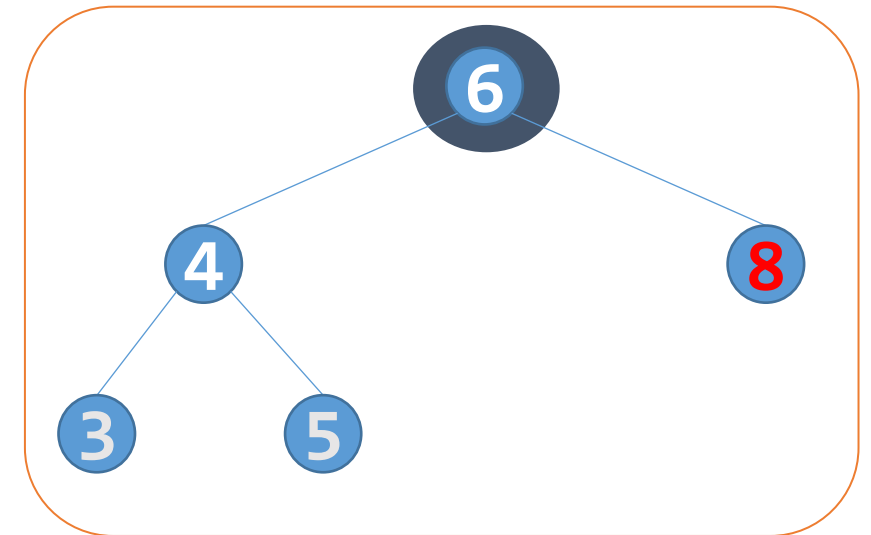


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        ➔ preOrdem(raiz->dir);  
    }  
}
```

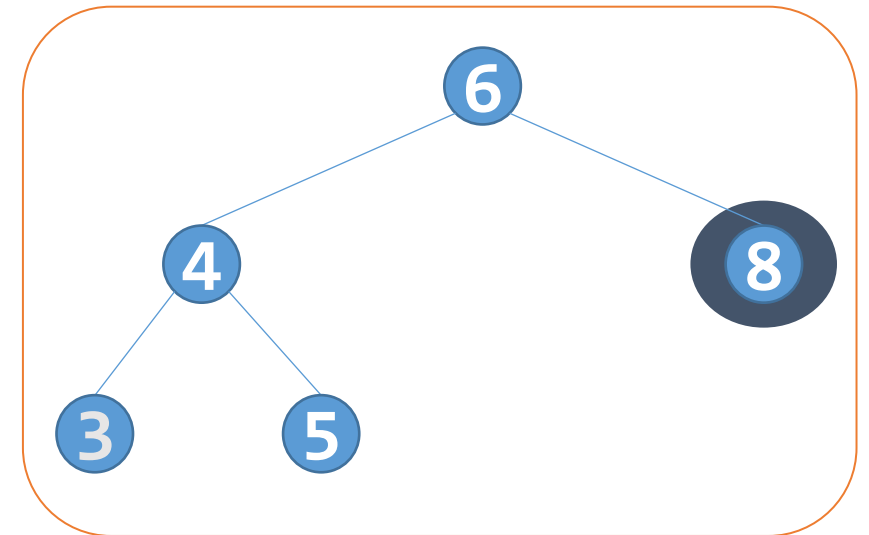


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    ➡ if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

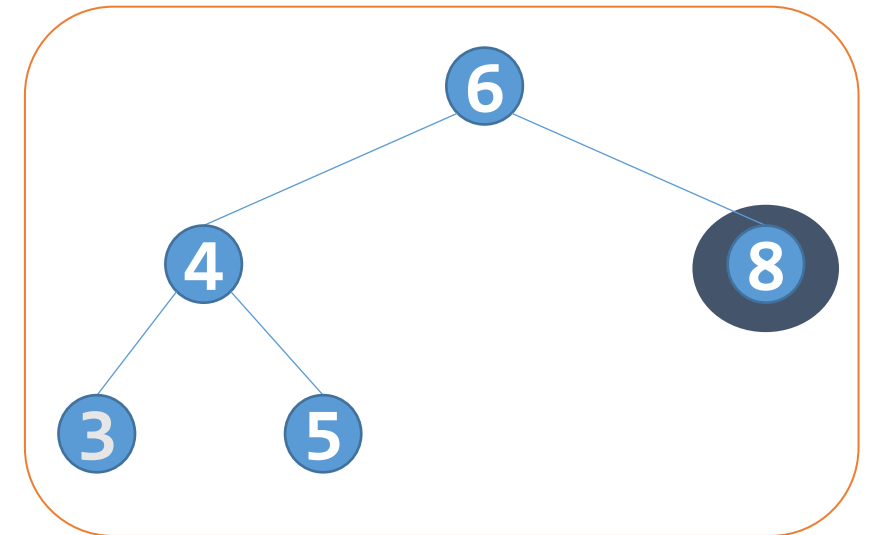


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        ➡ printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

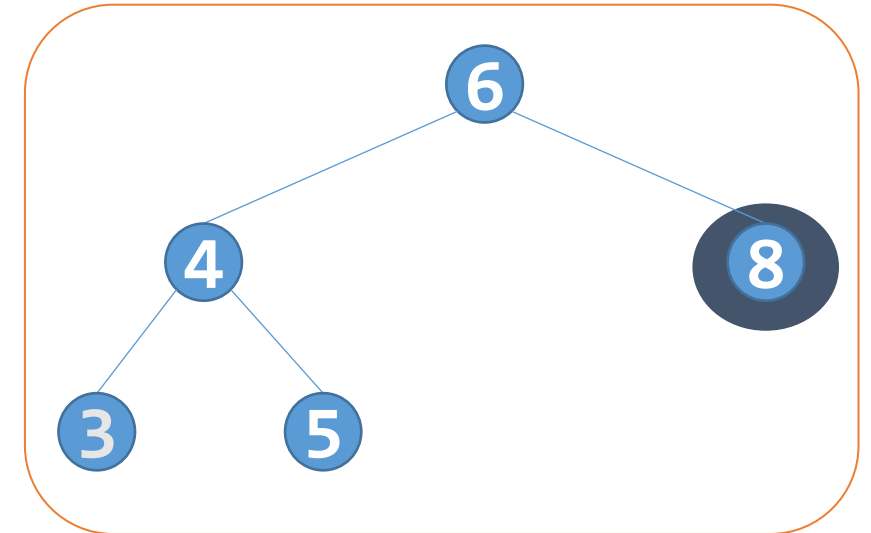


Algoritmos e Estruturas de Dados I



- **Pré-ordem**


```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        ➡ preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

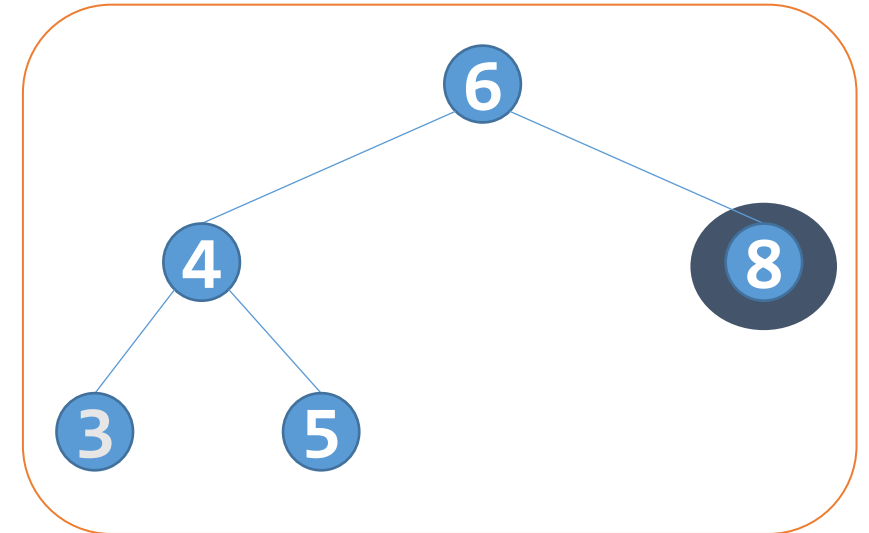


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        ➡ preOrdem(raiz->esq);   
        preOrdem(raiz->dir);  
    }  
}
```

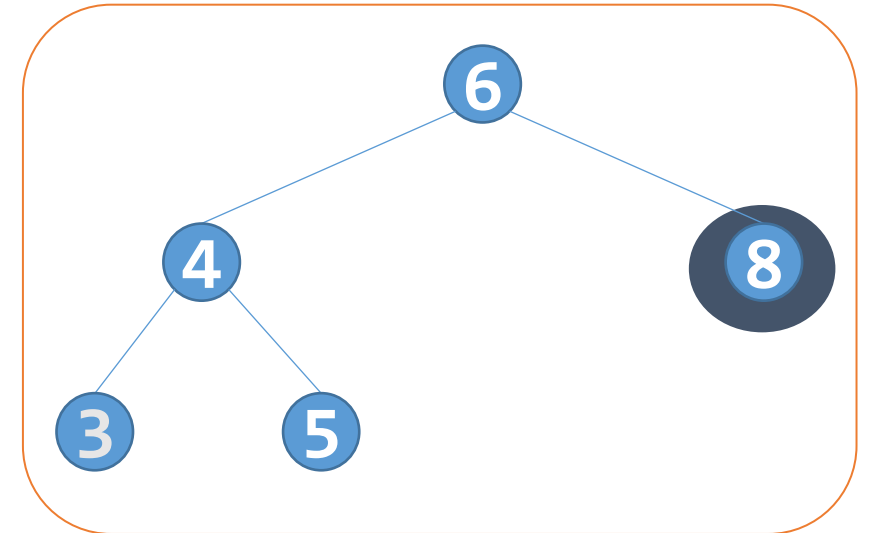


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        ➡ preOrdem(raiz->dir);  
    }  
}
```

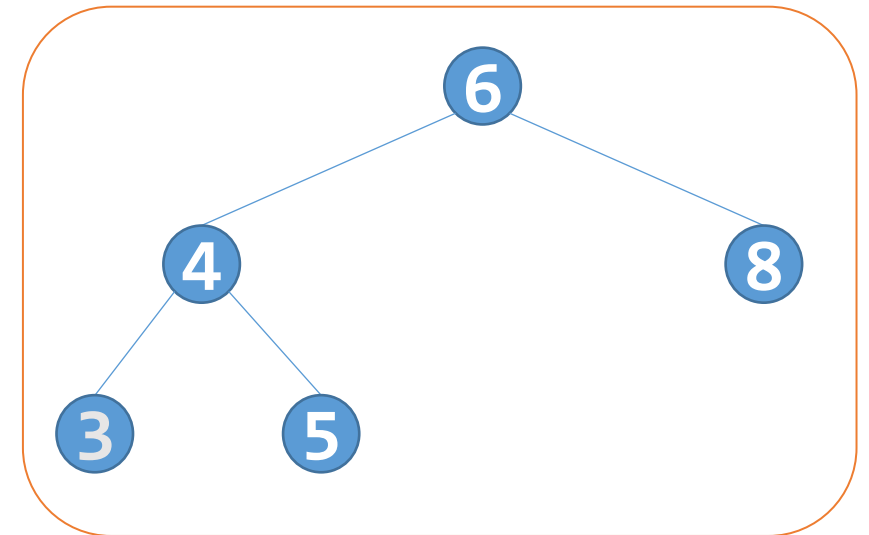


Algoritmos e Estruturas de Dados I



- **Pré-ordem**

```
void preOrdem(no *raiz){  
    if(raiz!=NULL){  
        printf("%d ",raiz->info);  
        preOrdem(raiz->esq);  
        preOrdem(raiz->dir);  
    }  
}
```

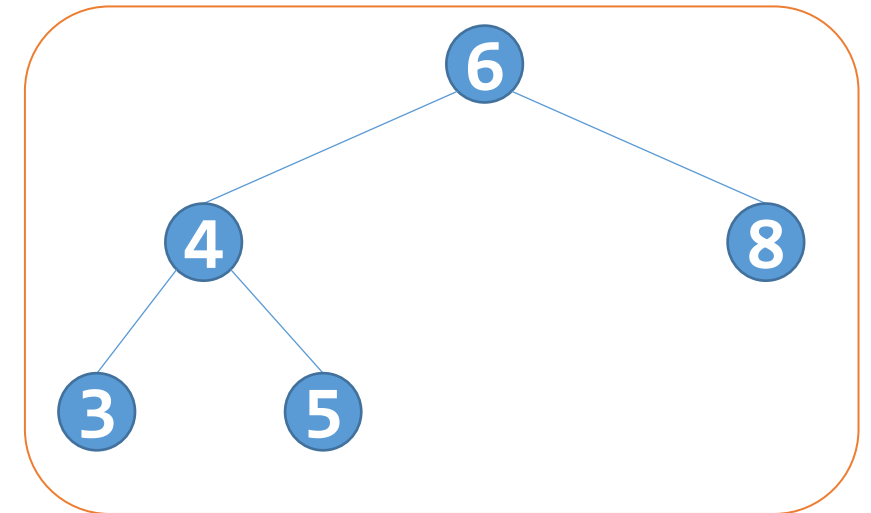


Algoritmos e Estruturas de Dados I



- **Em ordem (Simétrico)**

```
void EmOrdem(no *raiz){  
    if(raiz!=NULL){  
        EmOrdem(raiz->esq);  
        printf("%d ",raiz->info);  
        EmOrdem(raiz->dir);  
    }  
}
```

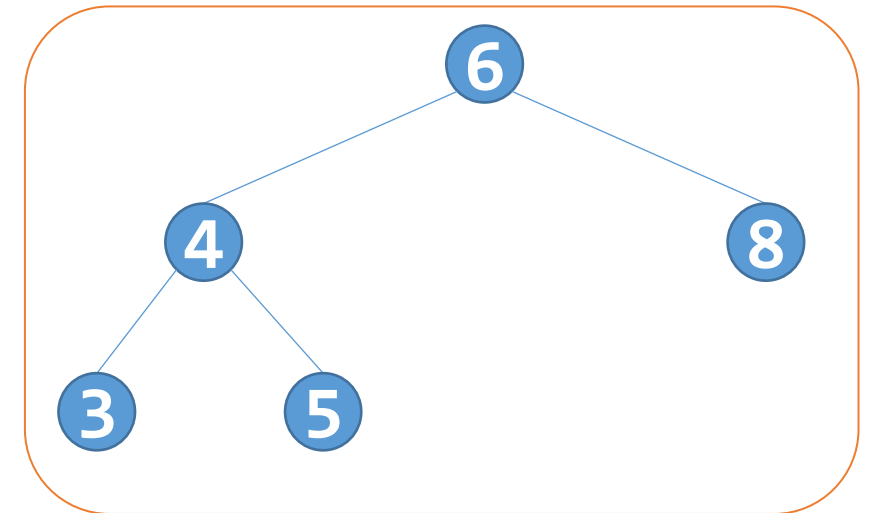


Algoritmos e Estruturas de Dados I



- **Em ordem (Simétrico)**

```
void PosOrdem(no *raiz){  
    if(raiz!=NULL){  
        PosOrdem(raiz->esq);  
        PosOrdem(raiz->dir);  
        printf("%d ",raiz->info);  
    }  
}
```



Algoritmos e Estruturas de Dados I



Exercício

Complemente a implementação dessa aula, apresentando um menu ao usuário que possibilite que ele insira elementos na árvore, busque por um valor e exiba todos os elementos, podendo escolher que tipo de percurso deseja executar.



DÚVIDAS???



FIM