



# SIN211 Algoritmos e Estruturas de Dados

---

Prof. João Batista Ribeiro

joao42ibatista@gmail.com



---

Universidade Federal de Viçosa

---

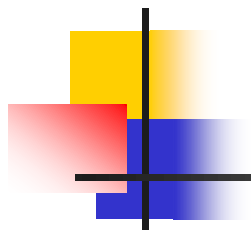
Slides baseados no material da Prof.<sup>a</sup> Rachel Reis



# Aula de Hoje

---

- Árvore de Decisão
- Árvore Binária de Pesquisa
- Árvore n-ária



# Árvore de Decisão



# Situação Problema

---

- Em alguns jogos de adivinhação a pessoa que detém a resposta escolhe um tema (ex: animal, cidade etc) que é apresentado aos participantes do jogo. Em seguida essa pessoa só poderá responder sim ou não para perguntas formuladas pelos participantes. Ganha o jogo o participante que encontrar a resposta mais rápido.

Se você tivesse que representar esse jogo usando um tipo abstrato de dados, que estrutura você indicaria? Dê um exemplo.



# Árvore - Aplicação

---

- Uma aplicação importante de árvores é na tomada de decisões.

## → Árvore de Decisão :

- *Definição*: é um instrumento de apoio à tomada de decisão que consiste numa representação gráfica das alternativas disponíveis geradas a partir de uma decisão inicial.
- *Vantagem*: possibilita que um problema complexo seja decomposto em diversos subproblemas mais simples.



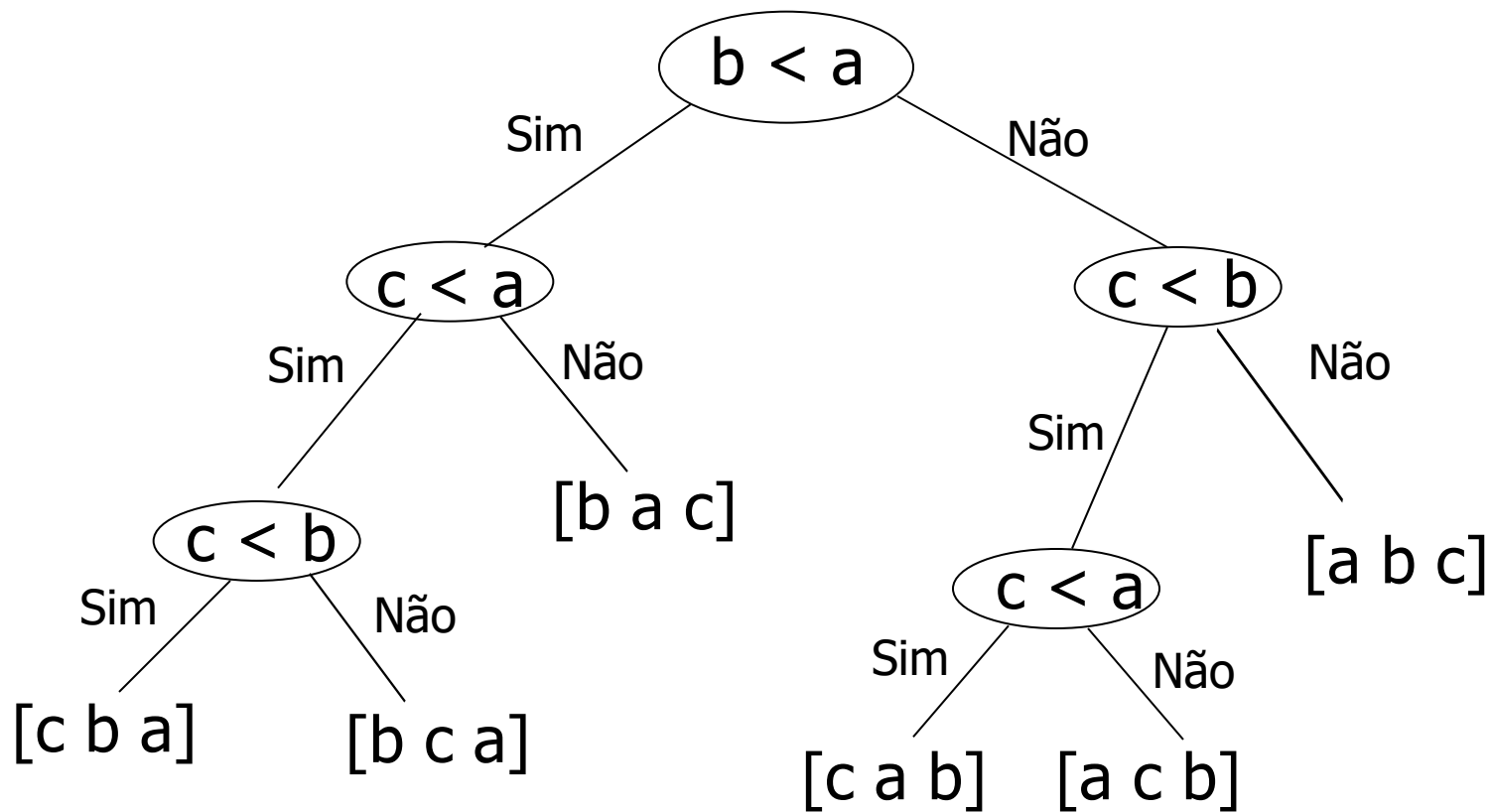
# Árvore - Aplicação

---

- A representação gráfica da árvore de decisão utiliza linhas para identificar a decisão (por ex. "sim" ou "não") e nós para identificar as questões sobre as quais se deve decidir.
- Cada um dos ramos formado por linhas e nós termina numa espécie de folha que identifica a consequência mais provável da sequência de decisões tomadas.

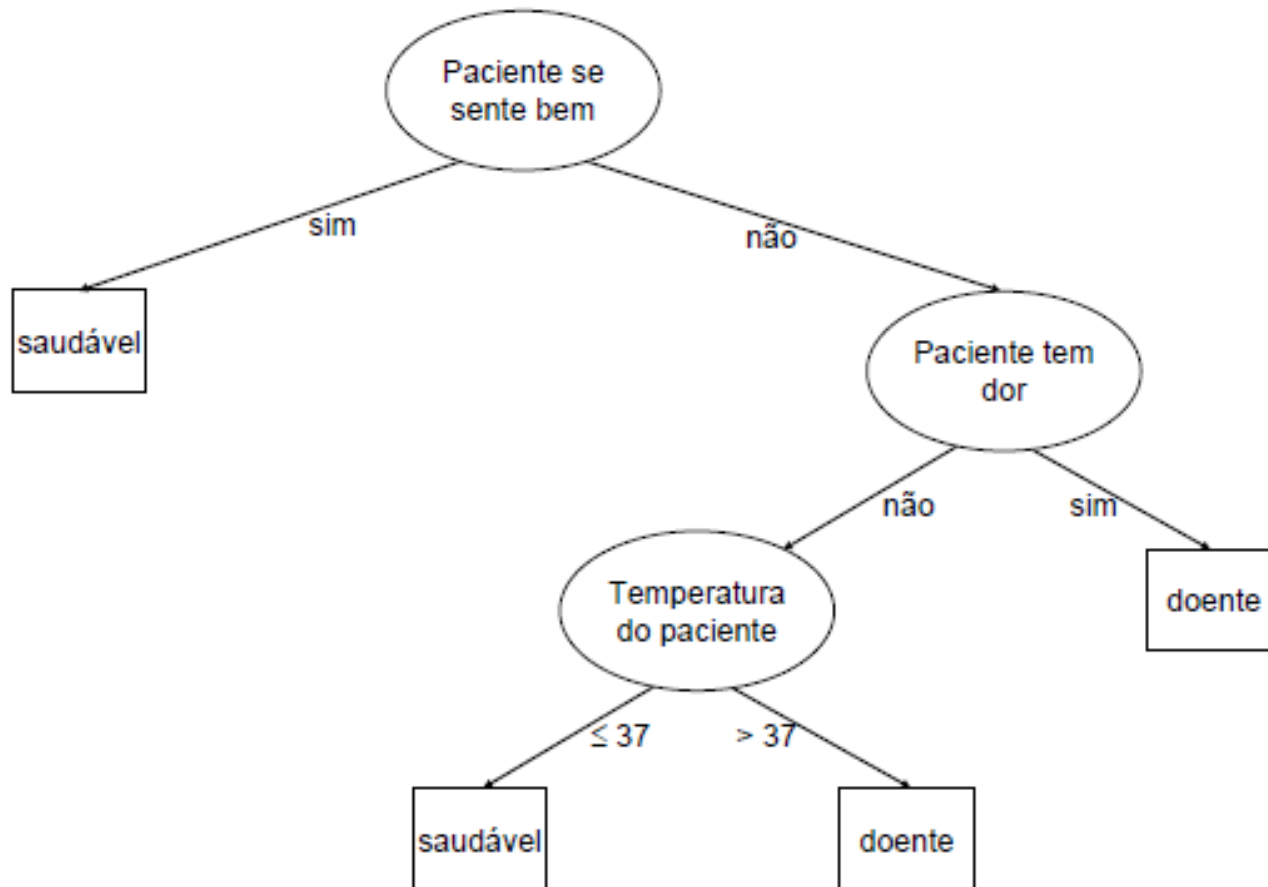
# Árvore - Aplicação

- Exemplo 1: Ordenação



# Árvore - Aplicação

- Exemplo 2:







---

# Árvore Binária de Pesquisa



# Definição

---

- Uma Árvore Binária de Pesquisa possui as mesmas propriedades de uma Árvore Binária, acrescida das seguintes propriedades:
  - Os nós pertencentes à subárvore esquerda possuem valores menores do que o valor associado ao nó raiz
  - Os nós pertencentes à subárvore direita possuem valores maiores do que o valor associado ao nó raiz
  - Um percurso *em-ordem* nessa árvore resulta na sequência de valores em ordem crescente



# Definição

---

- Se invertermos as propriedades descritas na definição anterior, de maneira que a subárvore esquerda de um nó contivesse valores maiores e a subárvore direita valores menores, o percurso *em-ordem* resultaria nos valores em ordem decrescente
- Uma árvore de busca criada a partir de um conjunto de valores não é única: o resultado depende da sequência de inserção dos dados



# Definição

---

- Uma árvore binária de pesquisa é dinâmica e pode sofrer alterações (inserções e remoções de nós) após ter sido criada
- Operações em árvores binárias de pesquisa:
  - Definição da estrutura de dados árvore
  - Inicializar
  - Inserir
  - Pesquisar
  - Remover



# Árvore Binária de Pesquisa

---

```
typedef struct sNo {  
    int info;  
    struct sNo*  esq;  
    struct sNo*  dir;  
}NO;
```

```
typedef struct sArvBinPesq {  
    NO* ptRaiz;  
}ArvBinPesq;
```



# Operação – inicializar

---

```
void inicializar(NO** raiz) {  
    *raiz = NULL;  
}
```



# Operação – inserir

---

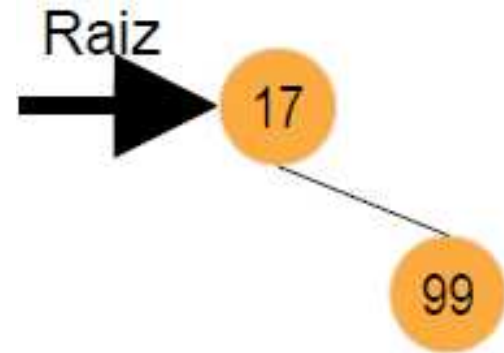
- Para entender o algoritmo considere a inserção do conjunto de números, na sequência:

$\{17, 99, 13, 1, 3, 100, 400\}$

- No início a árvore binária de pesquisa está vazia!

# Operação – inserir

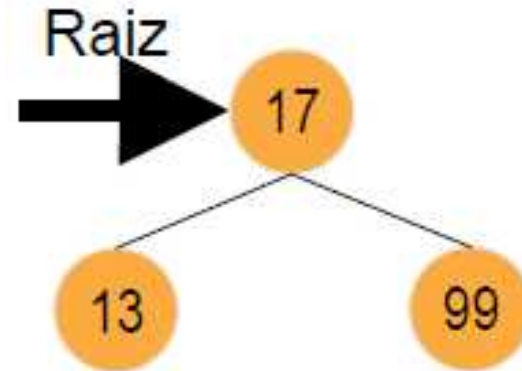
- O número 17 será inserido tornando-se o nó raiz
- A inserção do 99 inicia-se na raiz. Compara-se 99 com 17.
- Como  $99 > 17$ , ele deve ser colocado na subárvore direita do nó contendo 17



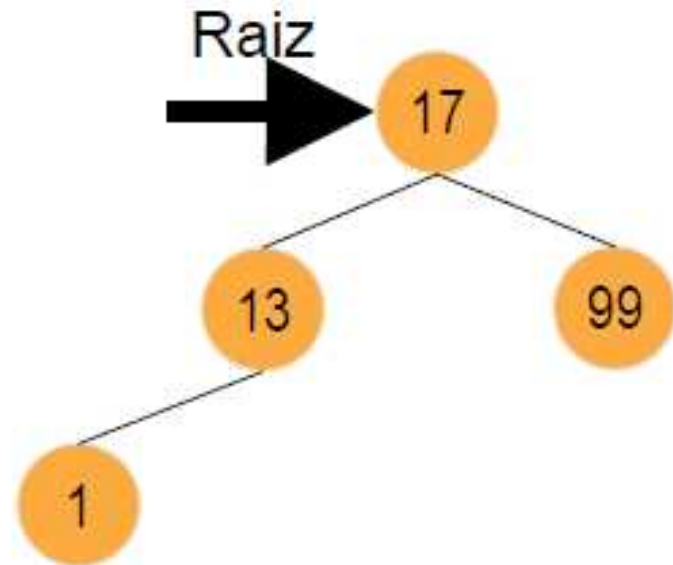


# Operação – inserir

- A inserção do 13 inicia-se na raiz
- Compara-se 13 com 17. Como  $13 < 17$ , ele deve ser colocado na subárvore esquerda do nó contendo 17
- Já que o nó 17 não possui descendente esquerdo, 13 é inserido na árvore nessa posição



# Operação – inserir

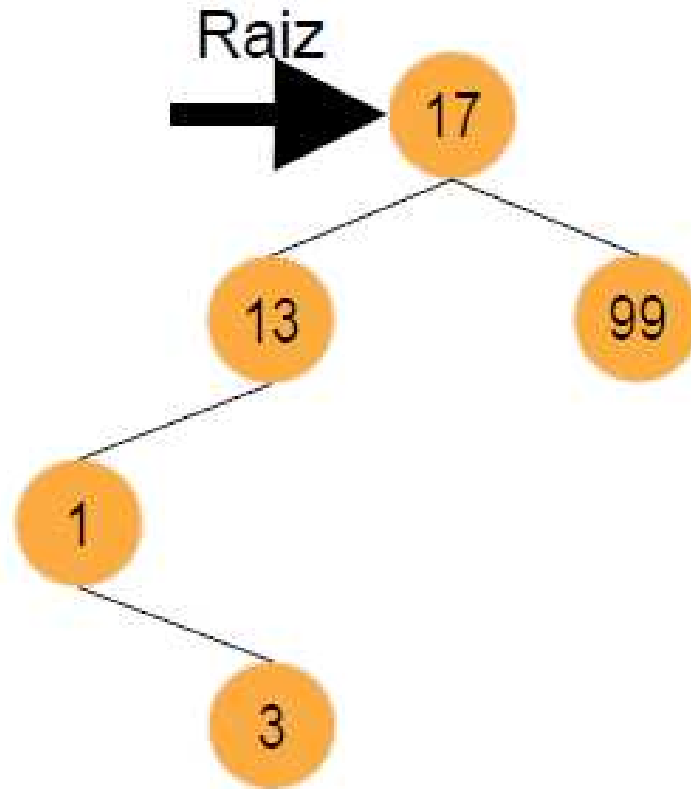


- Repete-se o procedimento para inserir o valor 1
- $1 < 17$ , então será inserido na subárvore esquerda
- Chegando nela, encontra-se o nó 13,  $1 < 13$  então ele será inserido na subárvore esquerda de 13

# Operação – inserir

- Repete-se o procedimento para inserir o elemento 3:

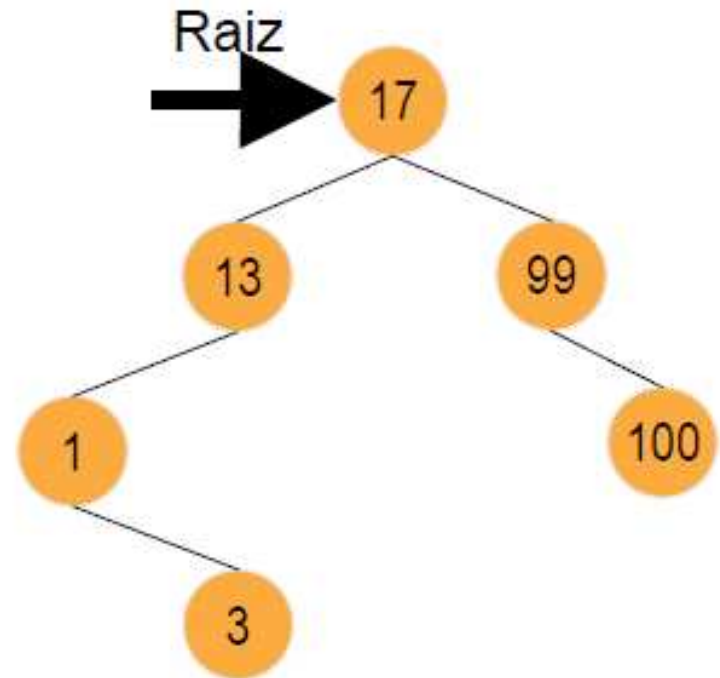
- $3 < 17$
- $3 < 13$
- $3 > 1$



# Operação – inserir

- Repete-se o procedimento para inserir o elemento 100:

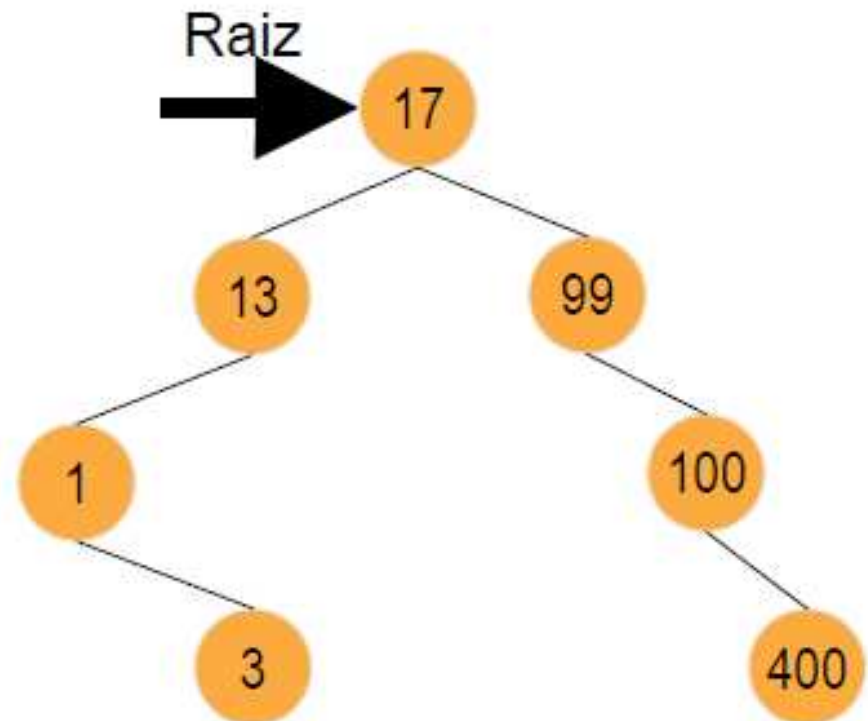
- $100 > 17$
- $100 > 99$



# Operação – inserir

- Repete-se o procedimento para inserir o elemento 400:

- ❑  $400 > 17$
- ❑  $400 > 99$
- ❑  $400 > 100$



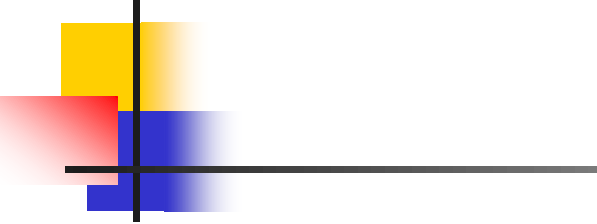


# Operação – inserir (implementação)

```
void inserir(NO** raiz, int x){
    NO* aux = *raiz;
    NO* aux2 = NULL;
    NO* novo;
    novo = (NO*) malloc(sizeof(NO));
    if (novo == NULL){
        printf("\nErro. Memoria nao alocada");
        return;
    }

    novo->info = x;
    novo->esq = NULL;
    novo->dir = NULL;
    ...
}
```

# Operação – inserir (cont.)



```
while (aux != NULL) {  
    aux2 = aux;  
    if (x < aux->info)  
        aux = aux->esq;  
    else  
        aux = aux->dir;  
}  
if (aux2 == NULL)  
    *raiz = novo;  
else {  
    if (x < aux2->info)  
        (aux2)->esq = novo;  
    else  
        (aux2)->dir = novo;  
}  
}
```



# Operação – inserir

---

Exercício: Desenvolva uma função recursiva para a operação de inserção

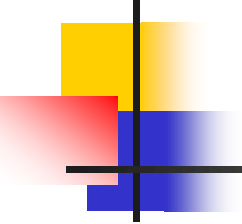




# Operação – pesquisar

---

- 1) Comece a pesquisar a partir do nó raiz;
- 2) Para cada nó raiz de uma subárvore compare:
  - se o valor procurado é menor que o valor no nó raiz  
(continue pela subárvore esquerda)
  - se o valor é maior que o valor no nó raiz  
(continue pela subárvore direita);
- 3) Caso o nó contendo o valor pesquisado seja encontrado, retorne um ponteiro para o nó; caso contrário, retorne um ponteiro nulo.



# Operação – pesquisar (implementação)

```
NO* pesquisar(NO* raiz, int k) {  
    NO* aux;  
    aux = raiz;  
    if (aux == NULL)  
        return NULL;  
    else if (k < aux->info)  
        return pesquisar(aux->esq, k);  
    else if (k > aux->info)  
        return pesquisar(aux->dir, k);  
    else  
        return aux;  
}
```



# Operação – remover

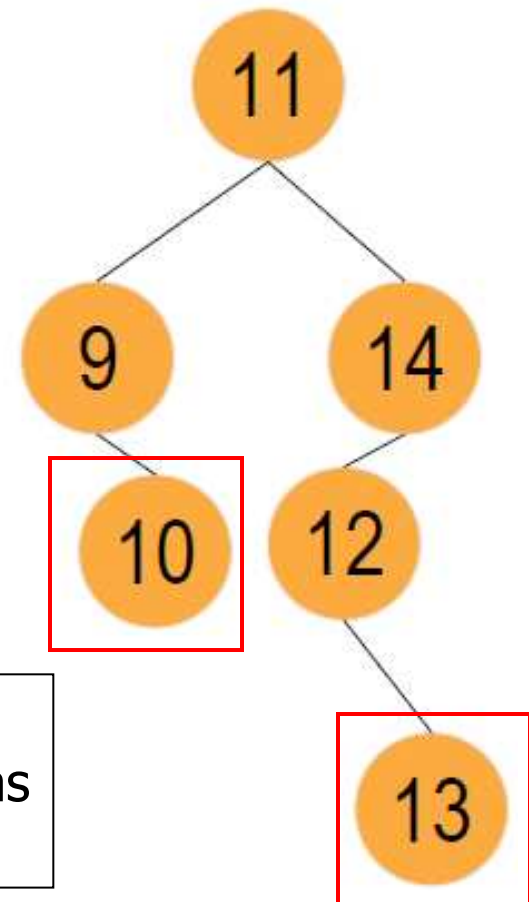
---

- Casos a serem considerados :
  - Caso 1: o nó é folha
    - O nó pode ser removido sem problema
  - Caso 2: o nó possui uma subárvore (esq ou dir)
    - O nó raiz da subárvore (esq ou dir.) “ocupa” o lugar do nó removido
  - Caso 3: o nó possui duas subárvores (esq e dir)
    - O nó contendo o menor valor da subárvore direita pode “ocupar” o lugar; ou o maior valor da subárvore esquerda pode “ocupar”

# Operação – remove

- Remover – Caso 1

Os nós com os valores 10 e 13 podem ser removidos sem necessidade de reajuste.

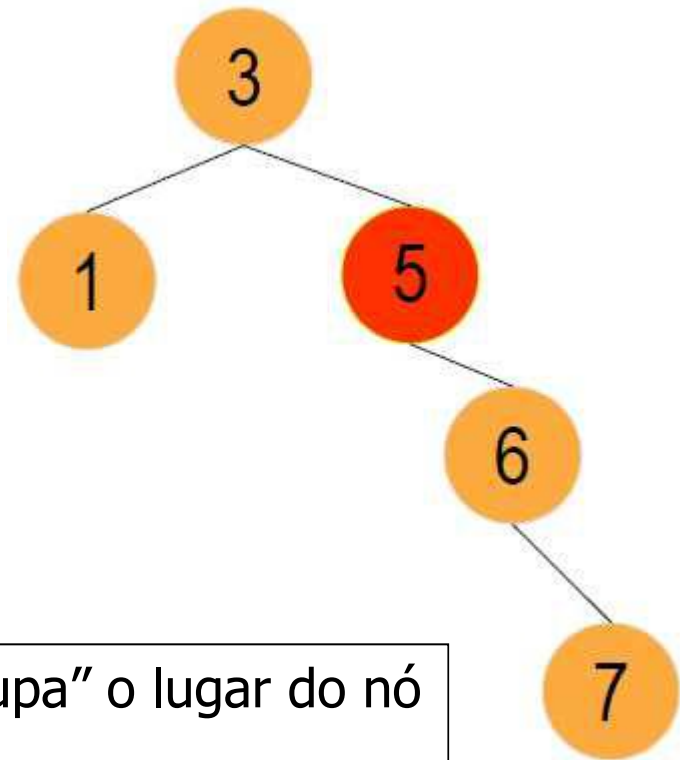


“Quando o nó a ser removido é folha, o mesmo pode ser retirado da árvore sem outras alterações”

# Operação – remove

- Remover – Caso 2

Como o elemento com valor 5 possui uma subárvore direita, o nó contendo o valor 6 irá “ocupar” o lugar do nó removido



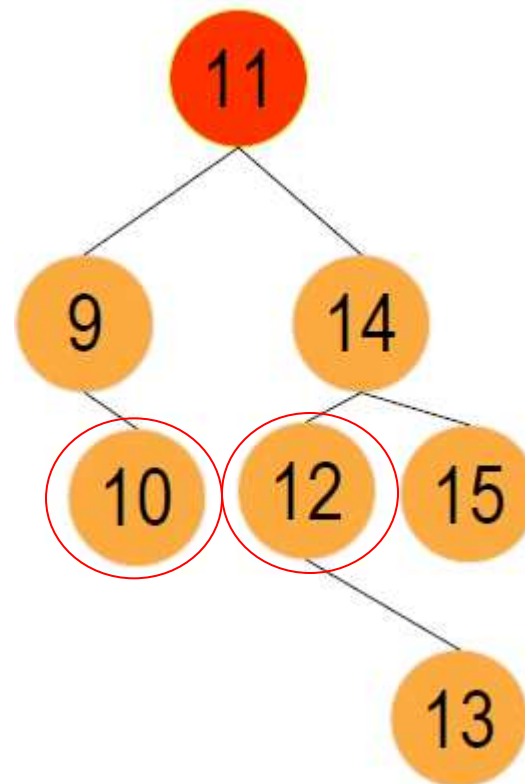
“O nó raiz da subárvore (esq. ou dir.) “ocupa” o lugar do nó removido”

# Operação – remove

- Remove – Caso 3

Neste caso, existem 2 opções:

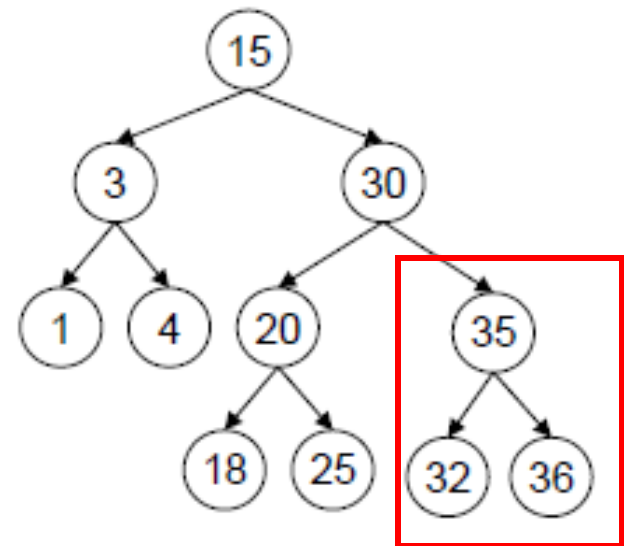
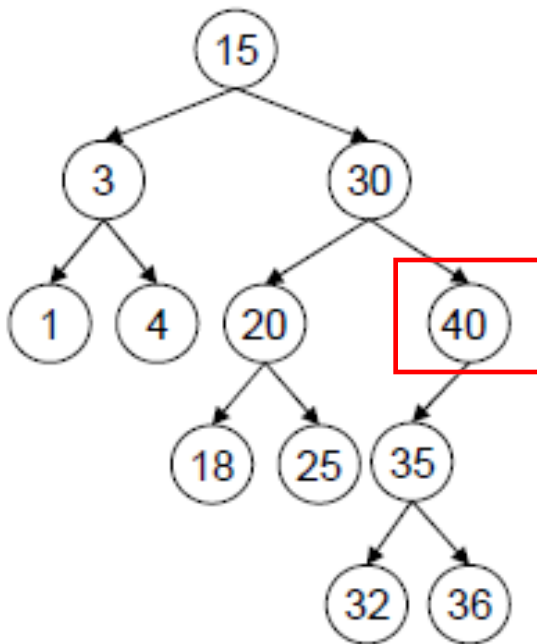
- 1) O nó com chave 10 pode “ocupar” o lugar do nó raiz,
- 2) O nó com chave 12 pode “ocupar” o lugar do nó raiz



“O nó contendo o menor valor da subárvore direita pode “ocupar” o lugar; ou o maior valor da subárvore esquerda pode “ocupar” “

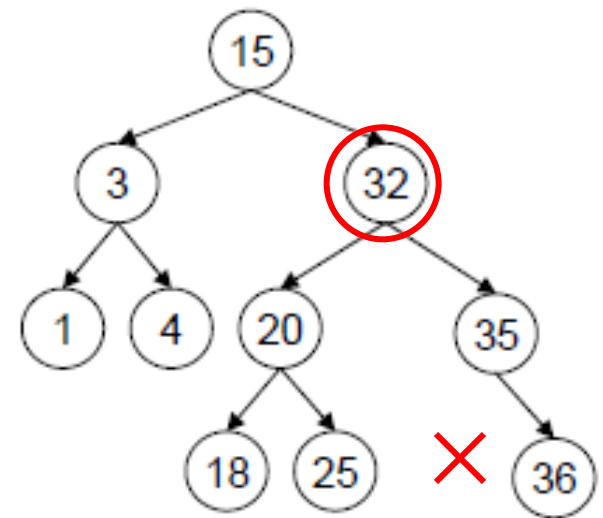
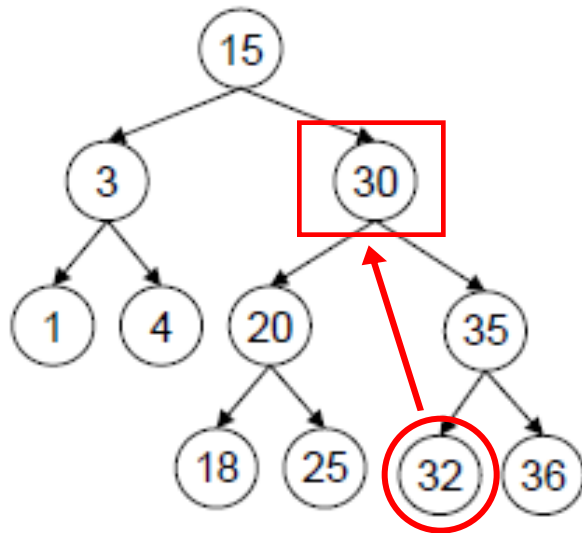
# Operação – remove

Remover o elemento 40



# Operação – remove

Remover o elemento 30

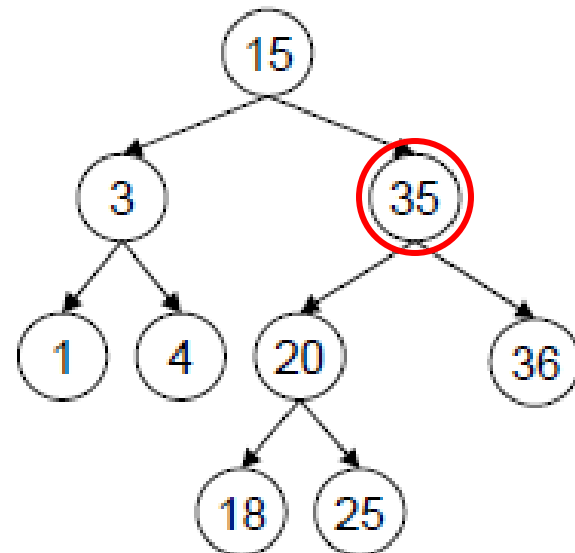
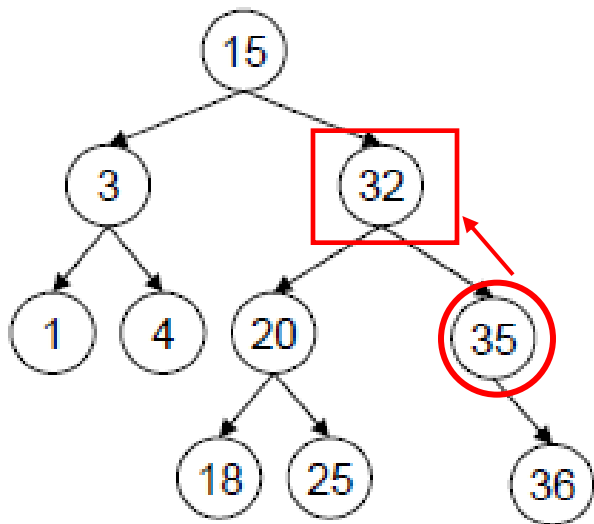


Qual seria a outra opção?



# Operação – remove

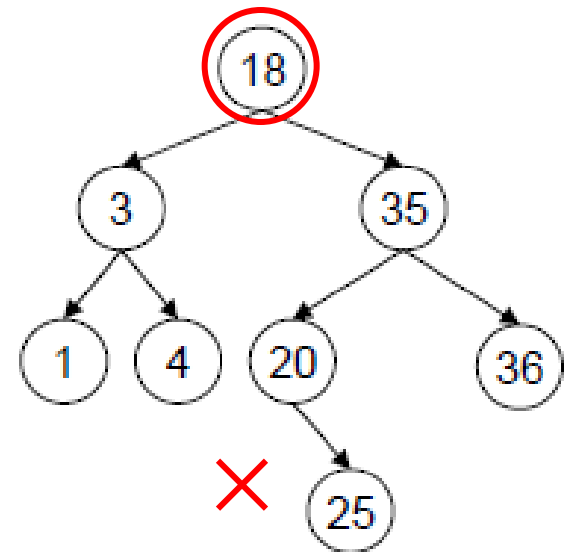
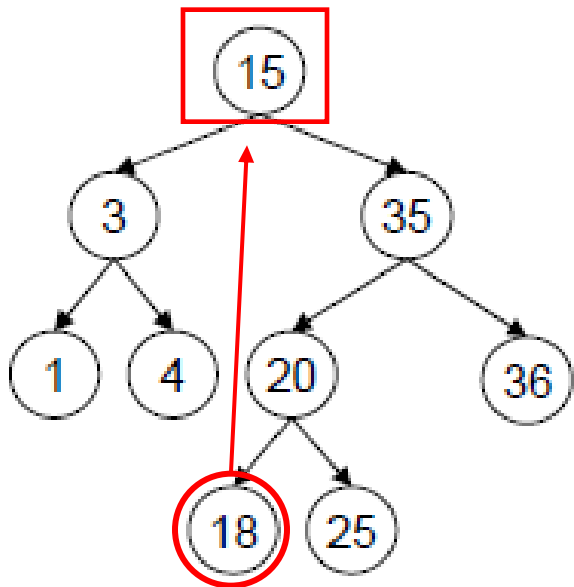
Remover o elemento 32



Qual seria a outra opção?

# Operação – remove

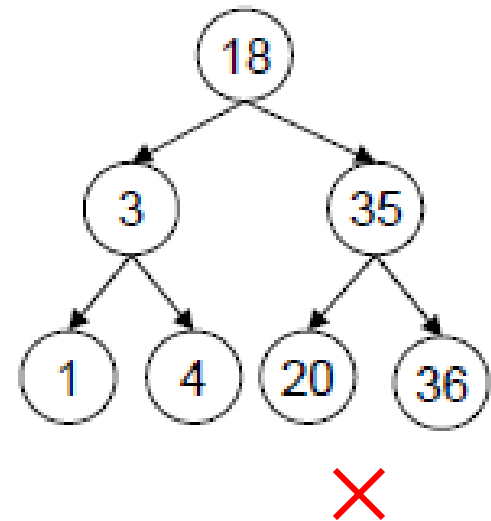
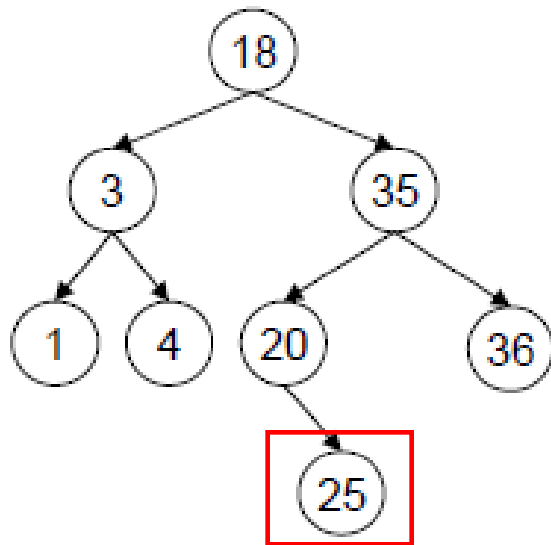
Remover o elemento 15



Qual seria a outra opção?

# Operação – remover

Remover o elemento 25





# Operação – remove

---

→ Revisão: quando o elemento **x** possui as duas subárvores não vazias, há duas estratégias possíveis para substituir o valor de **x**, preservando as propriedades de árvore binária de pesquisa:

❑ Encontrar o elemento de **menor** valor **y** na subárvore **direita** de **x** e transferi-lo para o nó ocupado por **x**

❑ Encontrar o elemento de **maior** valor **y** na subárvore **esquerda** de **x** e transferi-lo para o nó ocupado por **x**



# Operação – remover (implementação)

```
NO* remover(NO *raiz, int k){  
    if (raiz == NULL)  
        return NULL;  
    else if (raiz->info > k)  
        raiz->esq = remover(raiz->esq, k);  
    else if (raiz->info < k)  
        raiz->dir = remover(raiz->dir, k);  
    ...  
}
```

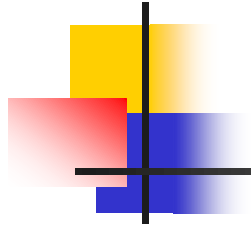
# Operação – remover (cont. 2)

## (implementação)

```
...  
else { /* achou o elemento */  
    if (raiz->esq == NULL && raiz->dir == NULL) { /* elemento sem filhos */  
        free (raiz);  
        raiz = NULL;  
    } else if (raiz->esq == NULL) { /* só tem filho à direita */  
        NO* temp = raiz;  
        raiz = raiz->dir;  
        free(temp);  
    } else if (raiz->dir == NULL) { /* só tem filho à esquerda */  
        NO* temp = raiz;  
        raiz = raiz->esq;  
        free(temp);  
    }  
...  
}
```

# Operação – remover (cont. 3) (implementação)

```
...  
    else { /* tem os dois filhos */  
        NO* Pai = raiz;  
        NO* F = raiz->esq;  
        while (F->dir != NULL) {  
            Pai = F;  
            F = F->dir;  
        }  
        raiz->info = F->info; /* troca as informações */  
        F->info = k;  
        raiz->esq = remove(raiz->esq, k);  
    }  
}  
return raiz;  
}
```



# Árvore n-ária





# Implementação de uma Árvore n-ária

---

- Como seria a implementação de uma árvore n-ária?
  - Nesse tipo de árvore cada nó pode conter um número diferente de subárvores.
- ✓ Solução 1: criar ponteiros de acordo com o grau da árvore, ou seja, se a árvore tiver grau 5, cinco ponteiros serão criados.
  - Desvantagem: para os nós com grau menor que 5 acontecerá de alguns ponteiros nunca serem usados.
- ✓ Solução 2: transformar a árvore n-ária em árvore binária

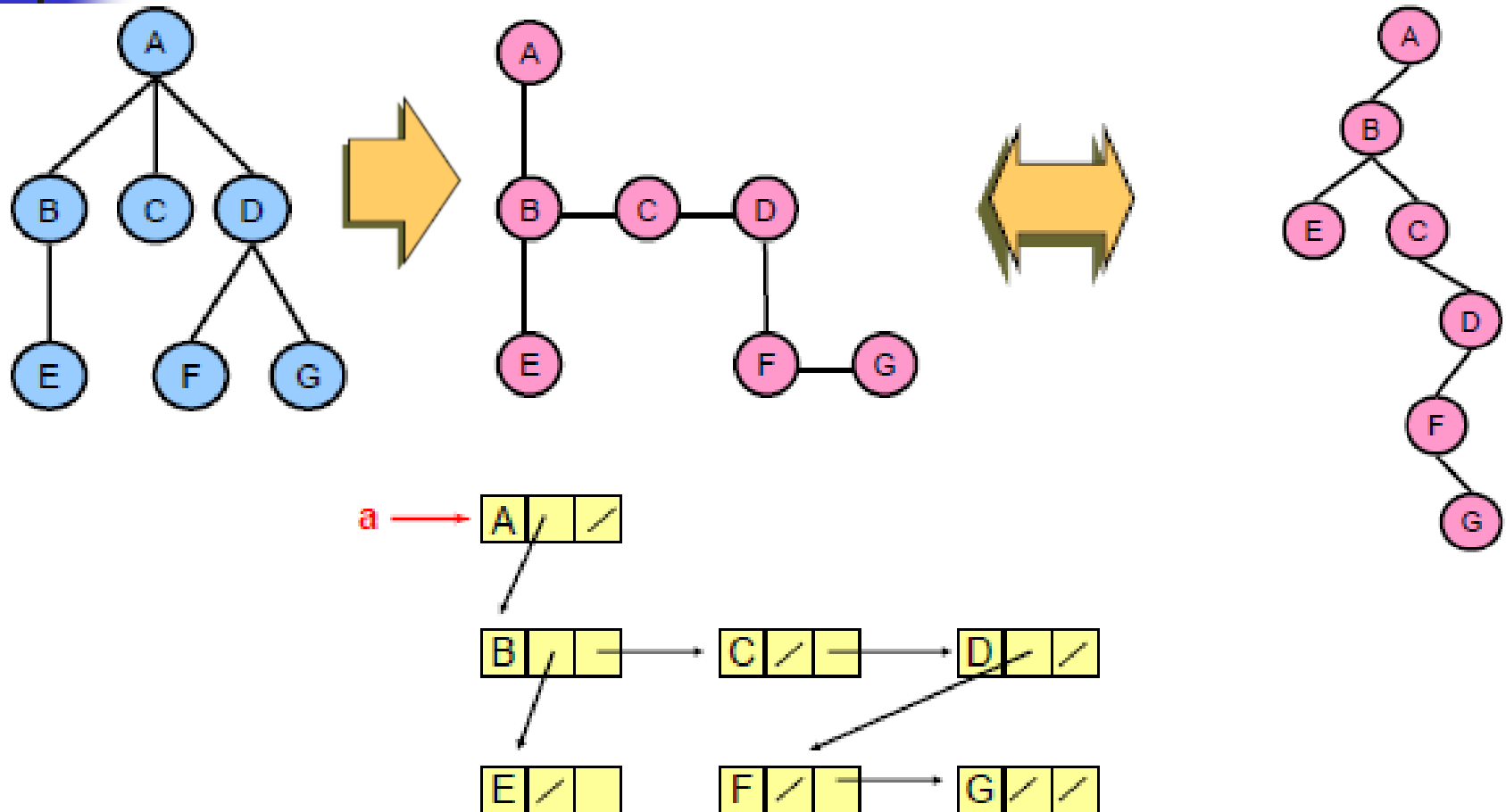


# Conversão de Árvore n-ária em Árvore binária

---

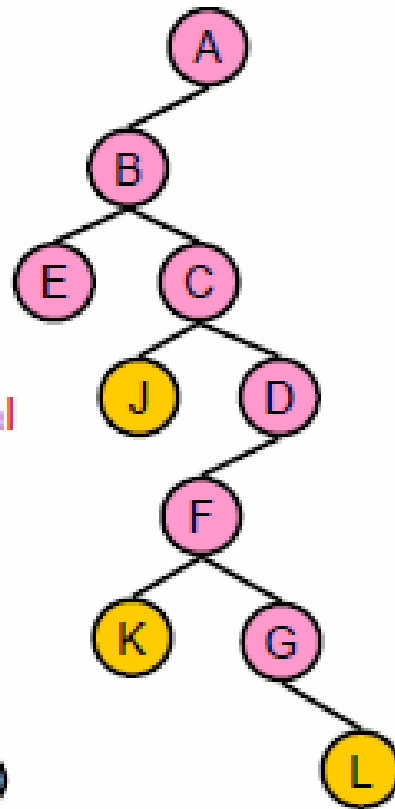
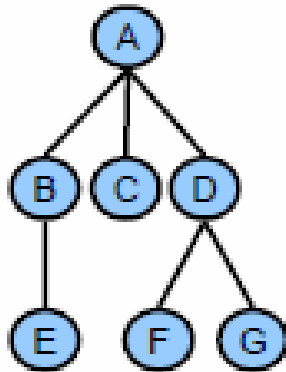
- Passos:
  - 1) O primeiro filho de um nó passa a ser seu filho à esquerda na árvore binária
  - 2) Os demais filhos de um nó passam a ser filhos à direita do seu irmão imediato à esquerda
  - 3) Executar o mesmo processo para cada nó da árvore
    - Filho à esquerda = primeiro filho
    - Filho à direita = irmão seguinte

# Conversão de Árvore n-ária em Árvore binária

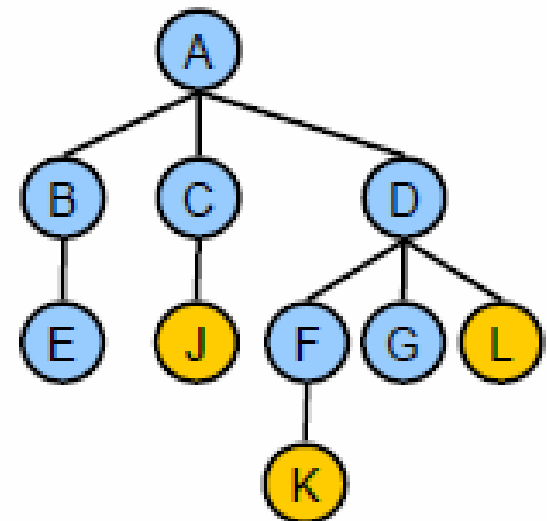


# Reconstituição Árvore n-ária

Árvore original



Árvore modificada



# Transformação de Floresta em Árvore Binária

- Para converter uma floresta em árvore binária, basta considerar as raízes das árvores como nós irmãos e aplicar a conversão anterior

