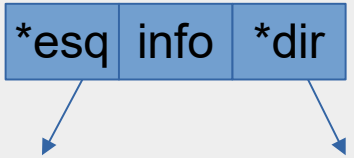
GSI011 – Estrutura de Dados 2

Profª: Christiane Regina Soares Brasil

GSI011 – Estrutura de Dados 2 – Aula 11

- Implementação de Árvore Binária:
  - Lista Encadeada (alocação dinâmica).

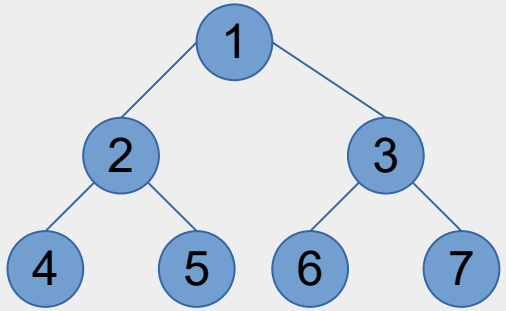
```
//ArvoreBinaria.h

typedef struct No* ArvBin;
…

//ArvoreBinaria.c
#include “ArvoreBinaria.h”
struct No{
int info;
struct No*esq;
struct No*dir;
};
```




```
//na main
ArvBin* raiz;
```

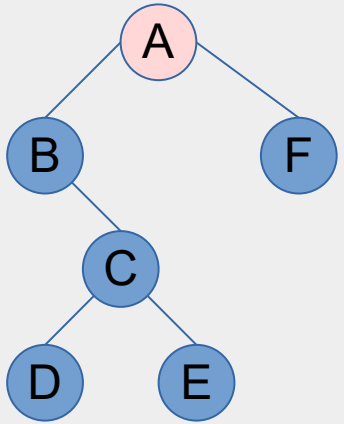
- Podemos percorrer a Árvore Binária de três modos:
  - Percurso pré-ordem
  - Percurso in-ordem
  - Percurso pós-ordem

- Percurso Pré-ordem:
  - Raiz
  - Filho esquerdo
  - Filho direito



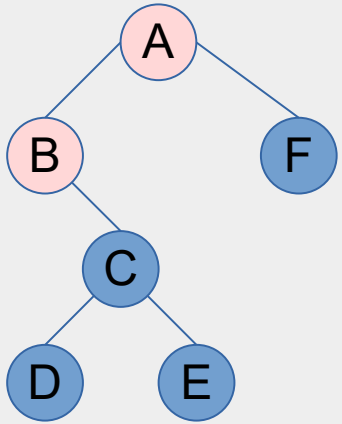
Pré-ordem: 1, 2, 4, 5, 3, 6, 7

- Percurso Pré-ordem:
  - Raiz
  - Filho esquerdo
  - Filho direito



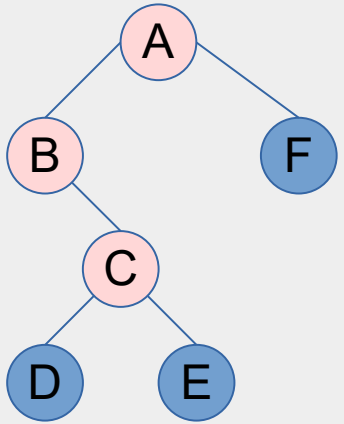
Pré-ordem: A

- Percurso Pré-ordem:
  - Raiz
  - Filho esquerdo
  - Filho direito



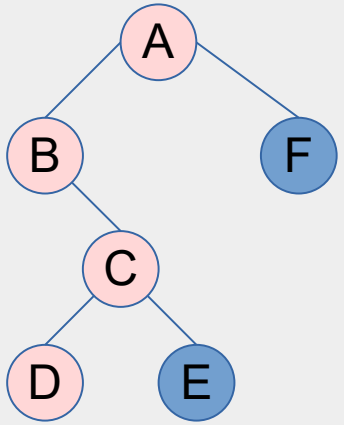
Pré-ordem: A, B

- Percurso Pré-ordem:
  - Raiz
  - Filho esquerdo
  - Filho direito



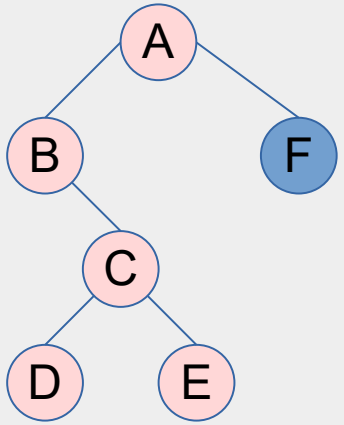
Pré-ordem: A, B, C

- Percurso Pré-ordem:
  - Raiz
  - Filho esquerdo
  - Filho direito



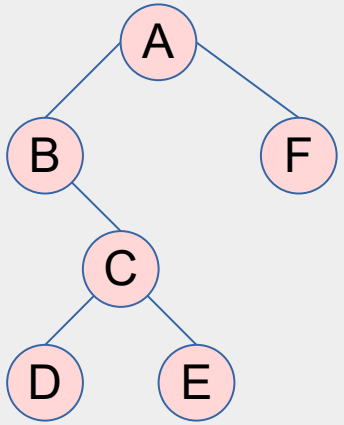
Pré-ordem: A, B, C, D

- Percurso Pré-ordem:
  - Raiz
  - Filho esquerdo
  - Filho direito



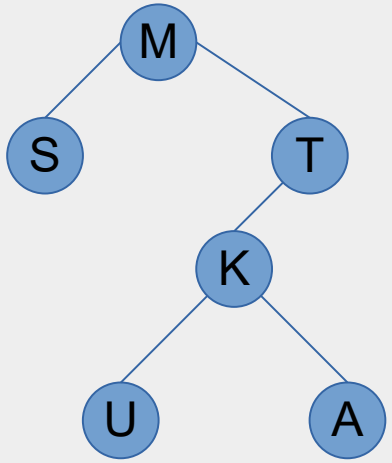
Pré-ordem: A, B, C, D, E

- Percurso Pré-ordem:
  - Raiz
  - Filho esquerdo
  - Filho direito



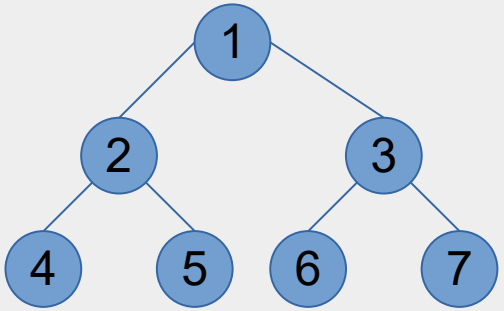
Pré-ordem: A, B, C, D, E, F

- Percurso Pré-ordem:
  - Raiz
  - Filho esquerdo
  - Filho direito



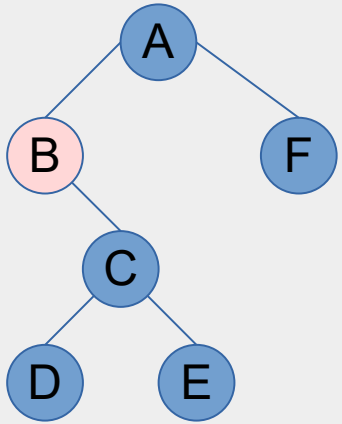
Pré-ordem: M, S, T, K, U, A

- Percurso In-ordem:
  - Filho esquerdo
  - Raiz
  - Filho direito



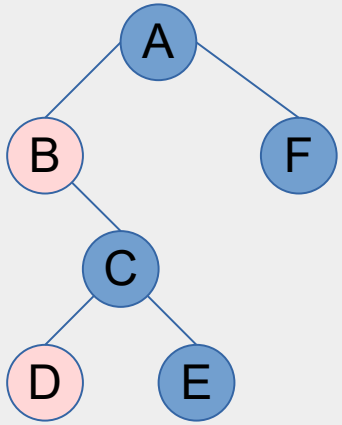
In-ordem: 4, 2, 5, 1, 6, 3, 7

- Percurso In-ordem:
  - Filho esquerdo
  - Raiz
  - Filho direito



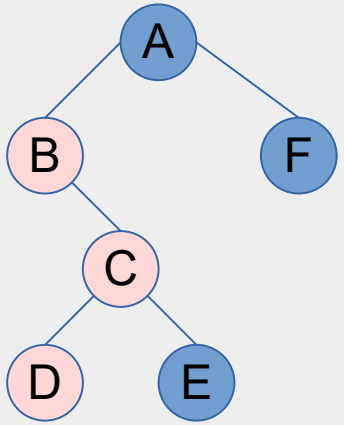
Pré-ordem: B

- Percurso In-ordem:
  - Filho esquerdo
  - Raiz
  - Filho direito



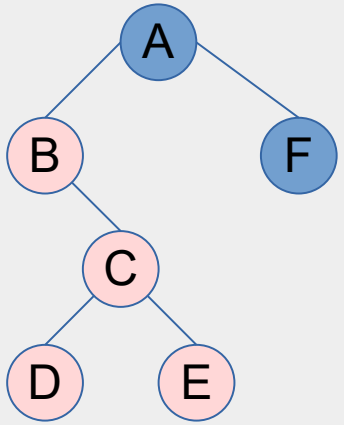
In-ordem: B, D

- Percurso In-ordem:
  - Filho esquerdo
  - Raiz
  - Filho direito



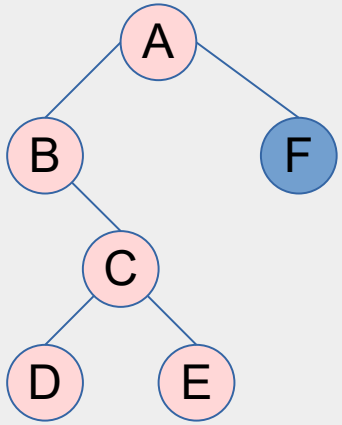
In-ordem: B, D, C

- Percurso In-ordem:
  - Filho esquerdo
  - Raiz
  - Filho direito



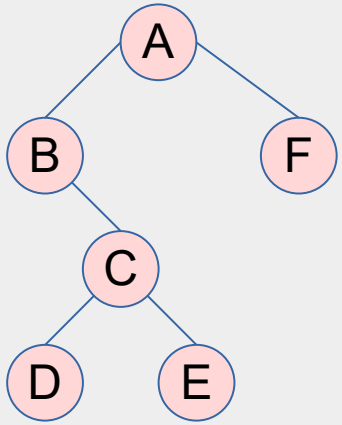
In-ordem: B, D, C, E

- Percurso In-ordem:
  - Filho esquerdo
  - Raiz
  - Filho direito



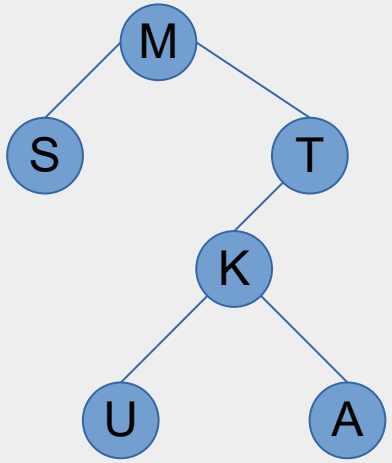
In-ordem: B, D, C, E, A

- Percurso In-ordem:
  - Filho esquerdo
  - Raiz
  - Filho direito



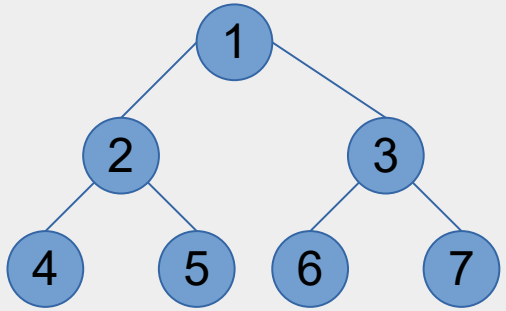
In-ordem: B, D, C, E, A, F

- Percurso In-ordem:
  - Filho esquerdo
  - Raiz
  - Filho direito



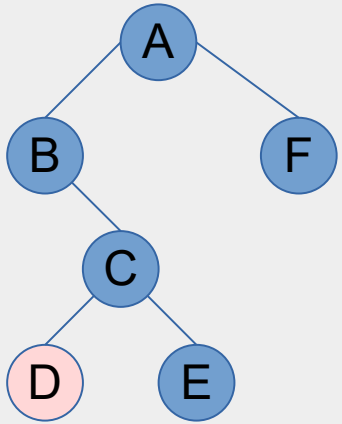
In-ordem: S, M, U, K, A, T

- Percurso Pós-ordem:
  - Filho esquerdo
  - Filho direito
  - Raiz



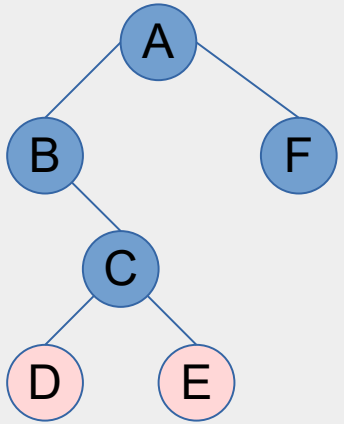
Pós-ordem: 4, 5, 2, 6, 7, 3, 1

- Percurso Pós-ordem:
  - Filho esquerdo
  - Filho direito
  - Raiz



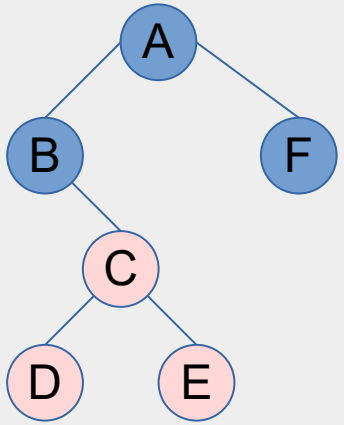
Pós-ordem: D

- Percurso Pós-ordem:
  - Filho esquerdo
  - Filho direito
  - Raiz



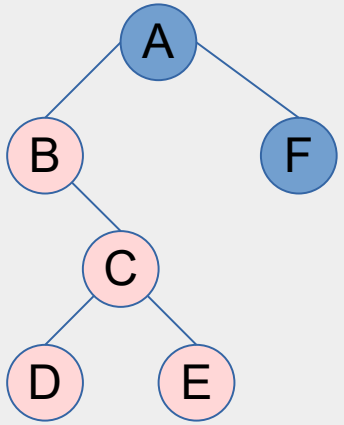
Pós-ordem: D, E

- Percurso Pós-ordem:
  - Filho esquerdo
  - Filho direito
  - Raiz



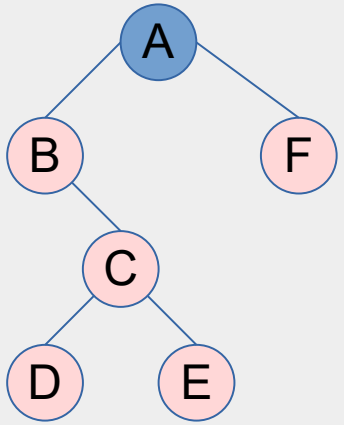
Pós-ordem: D, E, C

- Percurso Pós-ordem:
  - Filho esquerdo
  - Filho direito
  - Raiz



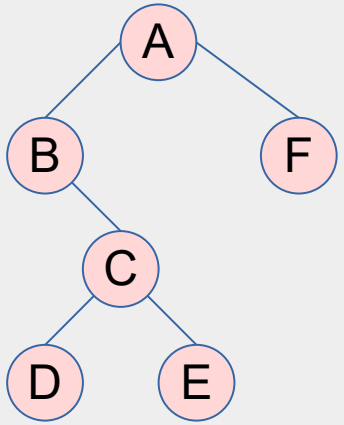
Pós-ordem: D, E, C, B

- Percurso Pós-ordem:
  - Filho esquerdo
  - Filho direito
  - Raiz



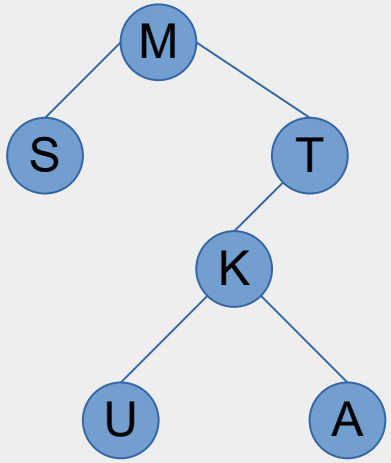
Pós-ordem: D, E, C, B, F

- Percurso Pós-ordem:
  - Filho esquerdo
  - Filho direito
  - Raiz



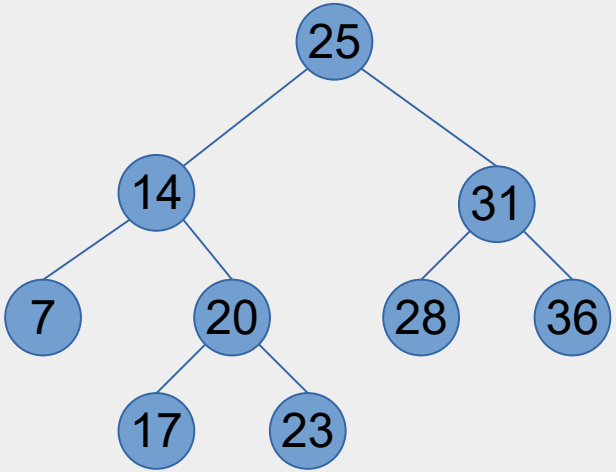
Pós-ordem: D, E, C, B, F, A

- Percurso Pós-ordem:
  - Filho esquerdo
  - Filho direito
  - Raiz



Pós-ordem: S, U, A, K, T, M

- Árvore de Busca Binária (ABB)
  - Para cada nó pai, todos os valores da subárvore esquerda são menores que o pai, e da direita são maiores que o pai. Essa regra é obrigatória para todos os nós.



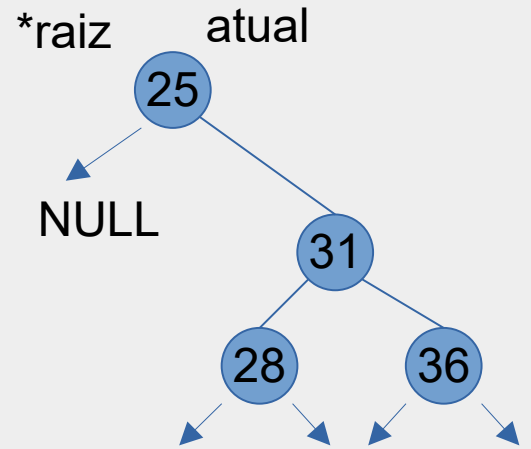
Complexidade:
Ordem logarítmica

IMPORTANTE: ABB não admite elementos repetidos.

- Implementação de busca na ABB:

```
//ArvoreBinaria.c
//Buscar um elemento em ABB
int buscaABB(ArvBin* raiz, int v)
{
    if( raiz == NULL) return 0;//não foi alocada
    struct No* atual = *raiz;
    while(atual!=NULL)
    {
        if(v==atual→ info) return 1;\\encontrou
        if(v<atual→ info) atual = atual→ esq;
        else                    atual = atual→ dir;
    }
    return 0;\\não encontrou
}
```

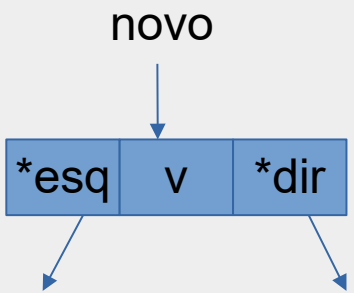
Exemplo: Simule com v=12 e depois v=50.



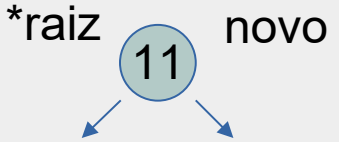
- Implementação de inserção na ABB:

```
//Inserir um elemento no local correto na ABB
int insereABB(ArvBin* raiz, int v)
{
    if( raiz == NULL) return 0;//não alocou
    struct No* novo = (struct No*) malloc (sizeof(struct No));
    if(novo == NULL) return 0;//não alocou
    novo→ info = v;
    novo→dir = novo→esq = NULL;
    if(*raiz == NULL)//árvore vazia
      * raiz = novo;
```

Continua..



11, 12, 23, 54

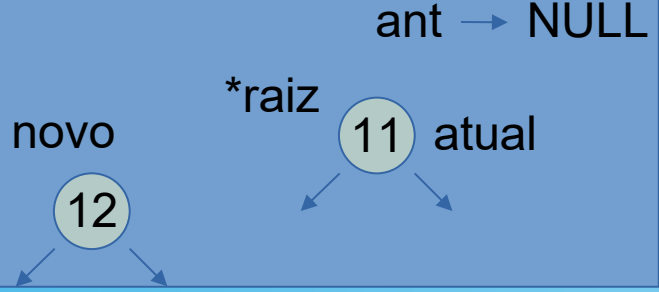


- Implementação de inserção na ABB:

```
else{//procurar a posição correta
    struct No* atual = *raiz, *ant = NULL;
    while( atual!= NULL)
    {
       ant = atual;
       if(v == atual→info )//já existe
       {
            free(novo);
            return 0;
       }
       if(v < atual→ info) atual = atual→ esq;
       else                atual = atual→dir;
    }//fim while
```

```
if(v<ant→info)//inserindo pela esquerda
      ant→esq = novo;
 else
     ant→dir = novo;
}//fim else
return 1;//inserido

}//fim função
```

11, 12, 23, 54

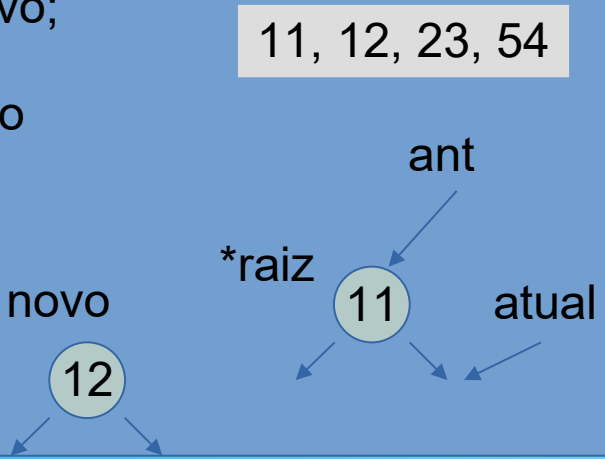


GSI011 – Estrutura de Dados 2 – Aula 11

- Implementação de inserção na ABB:

```
else{//procurar a posição correta
    struct No* atual = *raiz, *ant = NULL;
    while( atual!= NULL)
    {
       ant = atual;
       if(v == atual→info )//já existe
       {
            free(novo);
            return 0;
       }
       if(v < atual→ info) atual = atual→ esq;
       else                atual = atual→dir;
    }//fim while
```

```
if(v<ant→info)//inserindo pela esquerda
      ant→esq = novo;
 else
     ant→dir = novo;
}//fim else
return 1;//inserido

}//fim função
```

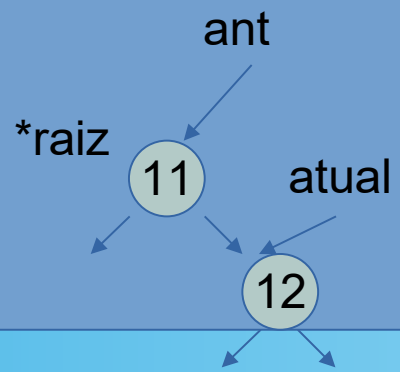


- Implementação de inserção na ABB:

```
else{//procurar a posição correta
    struct No* atual = *raiz, *ant = NULL;
    while( atual!= NULL)
    {
       ant = atual;
       if(v == atual→info )//já existe
       {
            free(novo);
            return 0;
       }
       if(v < atual→ info) atual = atual→ esq;
       else                atual = atual→dir;
    }//fim while
```

```
if(v<ant→info)//inserindo pela esquerda
      ant→esq = novo;
 else
     ant→dir = novo;
}//fim else
return 1;//inserido

}//fim função
```

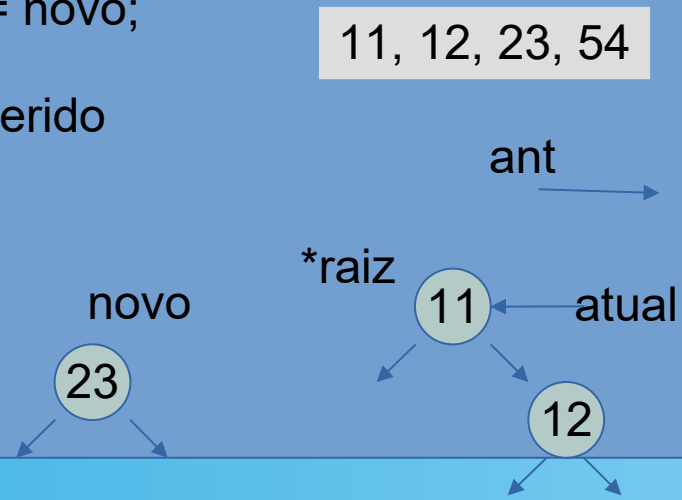
11, 12, 23, 54



- Implementação de inserção na ABB:

```
else{//procurar a posição correta
    struct No* atual = *raiz, *ant = NULL;
    while( atual!= NULL)
    {
       ant = atual;
       if(v == atual→info )//já existe
       {
            free(novo);
            return 0;
       }
       if(v < atual→ info) atual = atual→ esq;
       else                atual = atual→dir;
    }//fim while
```

```
if(v<ant→info)//inserindo pela esquerda
      ant→esq = novo;
 else
     ant→dir = novo;
}//fim else
return 1;//inserido

}//fim função
```

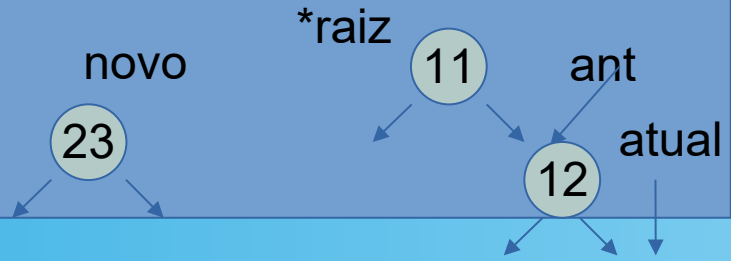


- Implementação de inserção na ABB:

```
else{//procurar a posição correta
    struct No* atual = *raiz, *ant = NULL;
    while( atual!= NULL)
    {
       ant = atual;
       if(v == atual→info )//já existe
       {
            free(novo);
            return 0;
       }
       if(v < atual→ info) atual = atual→ esq;
       else                atual = atual→dir;
    }//fim while
```

```
if(v<ant→info)//inserindo pela esquerda
      ant→esq = novo;
 else
     ant→dir = novo;
}//fim else
return 1;//inserido

}//fim função
```

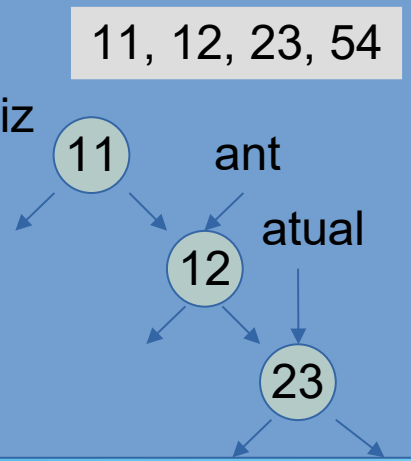
11, 12, 23, 54



- Implementação de inserção na ABB:

```
else{//procurar a posição correta
    struct No* atual = *raiz, *ant = NULL;
    while( atual!= NULL)
    {
       ant = atual;
       if(v == atual→info )//já existe
       {
            free(novo);
            return 0;
       }
       if(v < atual→ info) atual = atual→ esq;
       else                atual = atual→dir;
    }//fim while
```

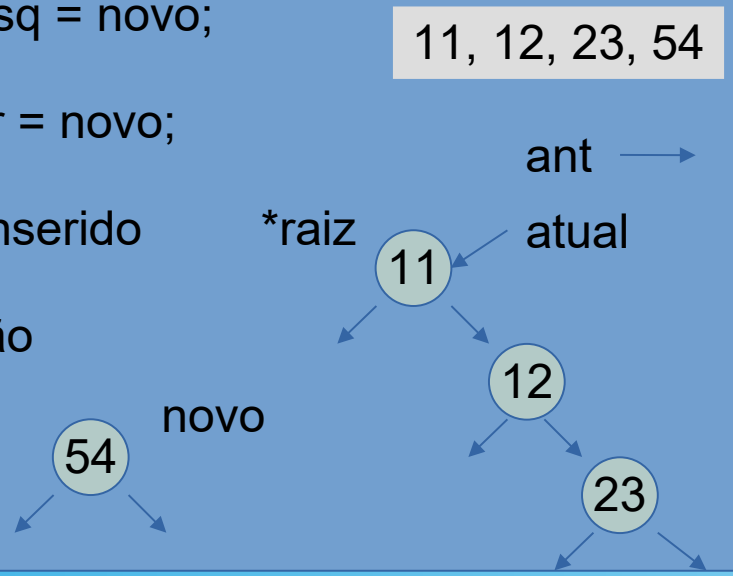
```
if(v<ant→info)//inserindo pela esquerda
      ant→esq = novo;
 else
     ant→dir = novo;
}//fim else
return 1;//inserido

}//fim função
```



- Implementação de inserção na ABB:

```
else{//procurar a posição correta
    struct No* atual = *raiz, *ant = NULL;
    while( atual!= NULL)
    {
       ant = atual;
       if(v == atual→info )//já existe
       {
            free(novo);
            return 0;
       }
       if(v < atual→ info) atual = atual→ esq;
       else                atual = atual→dir;
    }//fim while
```

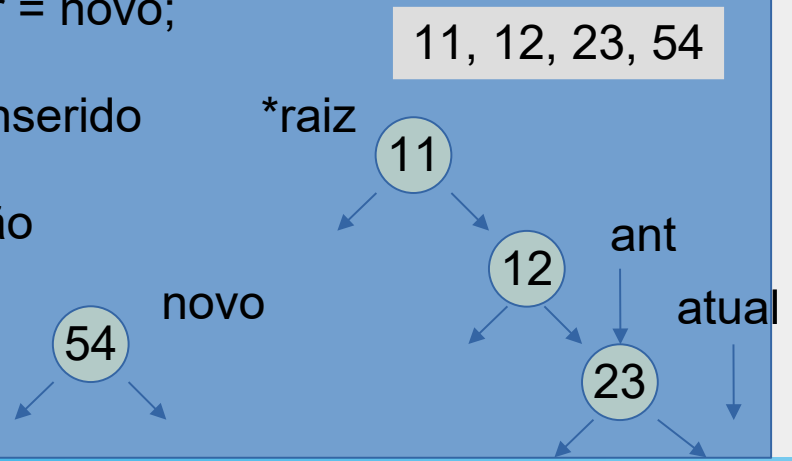
```
if(v<ant→info)//inserindo pela esquerda
      ant→esq = novo;
 else
     ant→dir = novo;
}//fim else
return 1;//inserido

}//fim função
```



- Implementação de inserção na ABB:

```
else{//procurar a posição correta
    struct No* atual = *raiz, *ant = NULL;
    while( atual!= NULL)
    {
       ant = atual;
       if(v == atual→info )//já existe
       {
            free(novo);
            return 0;
       }
       if(v < atual→ info) atual = atual→ esq;
       else                atual = atual→dir;
    }//fim while
```

```
if(v<ant→info)//inserindo pela esquerda
      ant→esq = novo;
 else
     ant→dir = novo;
}//fim else
return 1;//inserido

}//fim função
```

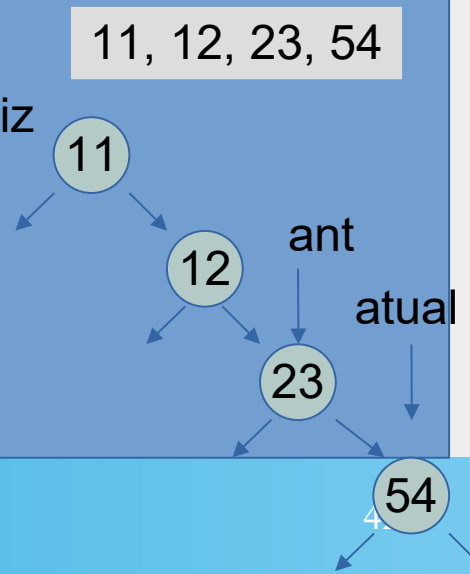


GSI011 – Estrutura de Dados 2 – Aula 11

- Implementação de inserção na ABB:

```
else{//procurar a posição correta
    struct No* atual = *raiz, *ant = NULL;
    while( atual!= NULL)
    {
       ant = atual;
       if(v == atual→info )//já existe
       {
            free(novo);
            return 0;
       }
       if(v < atual→ info) atual = atual→ esq;
       else                atual = atual→dir;
    }//fim while
```

```
if(v<ant→info)//inserindo pela esquerda
      ant→esq = novo;
 else
     ant→dir = novo;
}//fim else
return 1;//inserido

}//fim função
```



GSI011 – Estrutura de Dados 2 – Aula 11

- Implementação de remoção na ABB:

```
int removeABB(ArvBin* raiz, int v)
{
  if(raiz == NULL) return 0;//não foi alocada
   struct No* ant = NULL, *atual = *raiz;
   while(atual!= NULL)
   {
      if(v == atual→ info)//achei o elemento
      {
           if(atual == *raiz)  *raiz = removeAtual(atual);
            else
               if(ant→ esq == atual) ant→ esq = removeAtual(atual);
               else                          ant→ dir =  removeAtual(atual);

           return 1;//elemento removido
       }//fim if
```

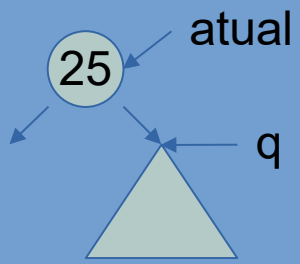
```
//continua procurando
ant = atual;
if(v < atual→ info)
  atual = atual→ esq;
eles
  atual = atual→ esq;
}//fim while
 return 0;//não foi removido
}
```

GSI011 – Estrutura de Dados 2 – Aula 11

- Implementação de remoção na ABB:

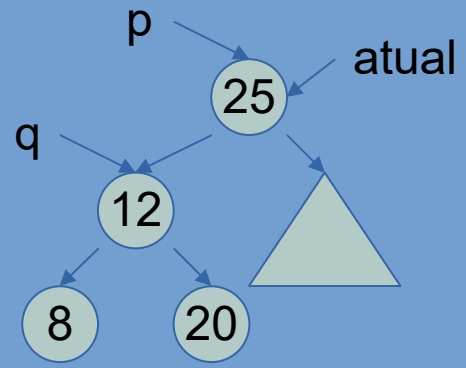
```
int removeAtual(struct No* atual)
{
   struct No* p, *q;
  //não existe subarv. esq.
   if(atual→ esq == NULL)
   {
      q = atual → dir;
      free(atual);
      return q;
  }
```

//Caso 1:




```
//procura o filho mais à direita da
//subárvore da esquerda
p = atual;
q = atual→esq;
while(q→ dir!=NULL)
{
   p = q;
   q = q→ dir;
}
```

//Caso 2:

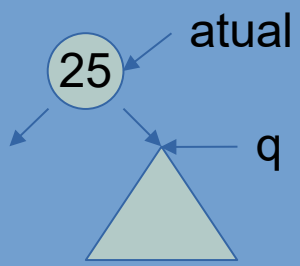


Continua..

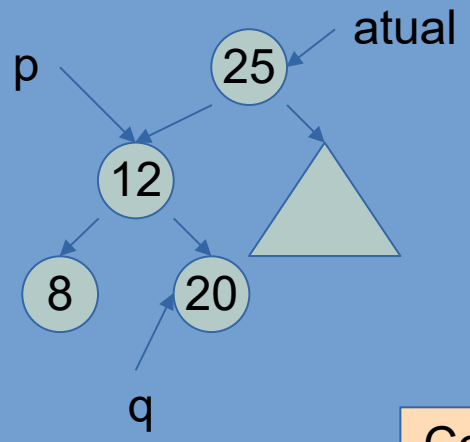
GSI011 – Estrutura de Dados 2 – Aula 11

- Implementação de remoção na ABB:

```
int removeAtual(struct No* atual)
{
   struct No* p, *q;
   if(atual→ esq == NULL)
   {                                        //Caso 1:
      q = atual → dir;
      free(atual);
      return q;
  }
```




```
//procura o filho mais à direita (o maior) da
//subárvore da esquerda
p = atual;
q = atual→esq;
while(q→ dir!=NULL)                         //Caso 2:
{
   p = q;
   q = q→ dir;
}
```



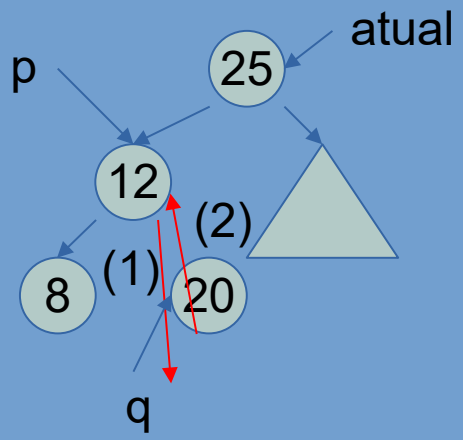
Continua..

GSI011 – Estrutura de Dados 2 – Aula 11

- Implementação de remoção na ABB:

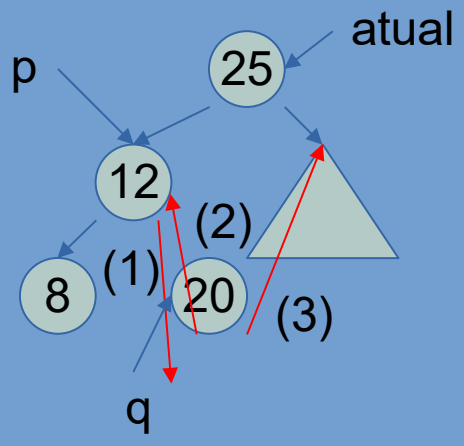
```
//tem filho mais à direita
if(p != atual)
  {
     p→ dir = q→ esq;//(1)
     q→ esq = atual→ esq;//(2)
  }
```

//Caso 2:




```
q → dir = atual→ dir;//(3)
free(atual);
return q;

}//fim função
```

//Caso 3:



GSI011 – Estrutura de Dados 2 – Aula 11

- Implementação de remoção na ABB:

```
//tem filho mais à direita
if(p != atual)
   {
      p→ dir = q→ esq;//(1)
      q→ esq = atual→ esq;//(2)
  }
```

```
q → dir = atual→ dir;//(3)
free(atual);
return q;

}//fim função
```

//Caso 3: