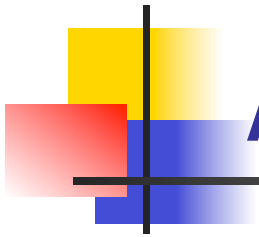
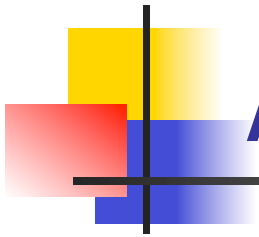


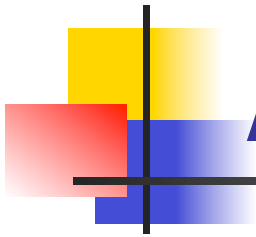
Árvore Binária de Busca



Árvore Binária de Busca

- construída de tal forma que, para cada nó:
  - nós com chaves menores estão na sub-árvore esquerda
  - nós com chaves maiores (ou iguais) estão na sub-árvore direita

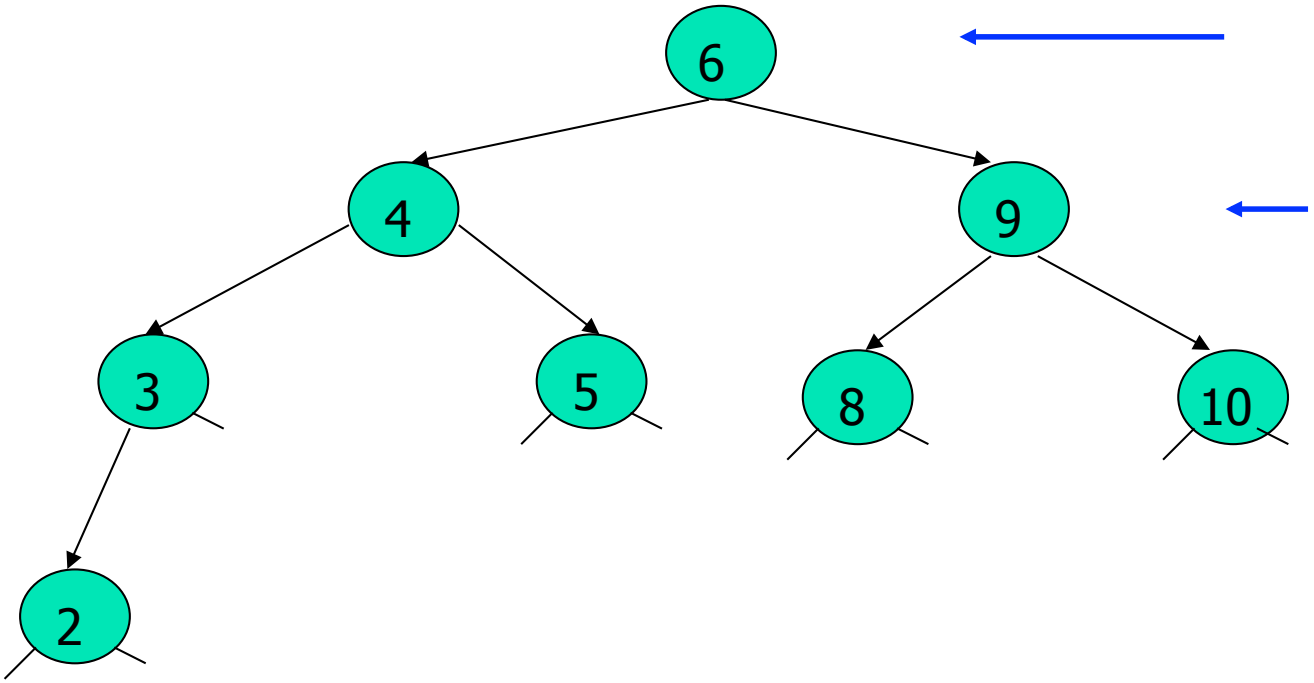
- a inserção dos nós da árvore deve satisfazer a essa propriedade



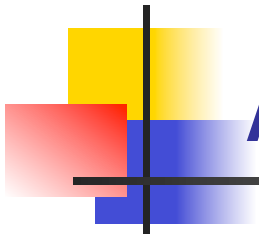
Árvore Binária de Busca

- para a busca de uma chave v na árvore binária de busca:
  - primeiro compare com a raiz
    - se menor, vá para a sub- árvore esquerda
    - se maior, para a sub-árvore direita
- aplique o método recursivamente

Árvore Binária de Busca



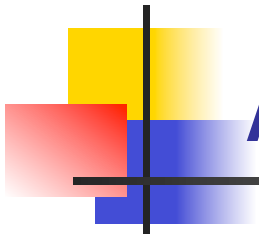
Buscando a chave 9



Árvore Binária de Busca

- a cada passo, garante-se que nenhuma outra parte da árvore contém a chave sendo buscada

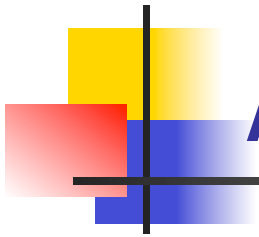
- o procedimento pára quando
  - o nó com v é encontrado
  - senão, chega-se a NULL



Árvore Binária de Busca

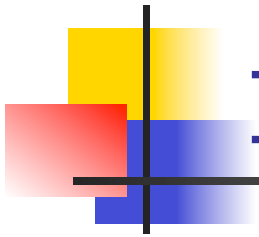
- chama a função de busca se a árvore é não vazia

```
busca_arvore_nao_recursivo (v, pt)
{
        ???

}
```



Árvore Binária de Busca

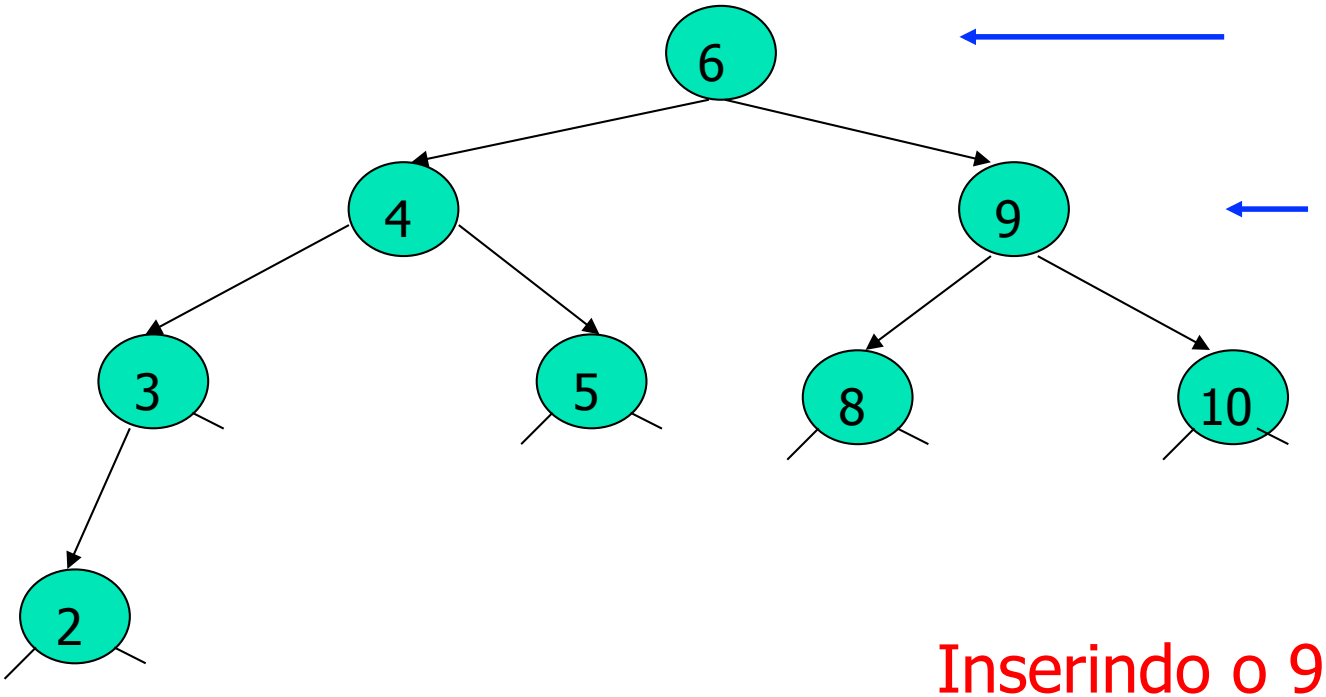
```
busca_arvore_nao_recursivo (v, pt)
{
   do {
            if (v < pt->info)
                     pt =  pt-> esq;
            else pt =  pt-> dir;
   }while (pt != NULL) && (v != pt->info);
   return(pt);
}
```



Inserindo em Árvore Binária de Busca

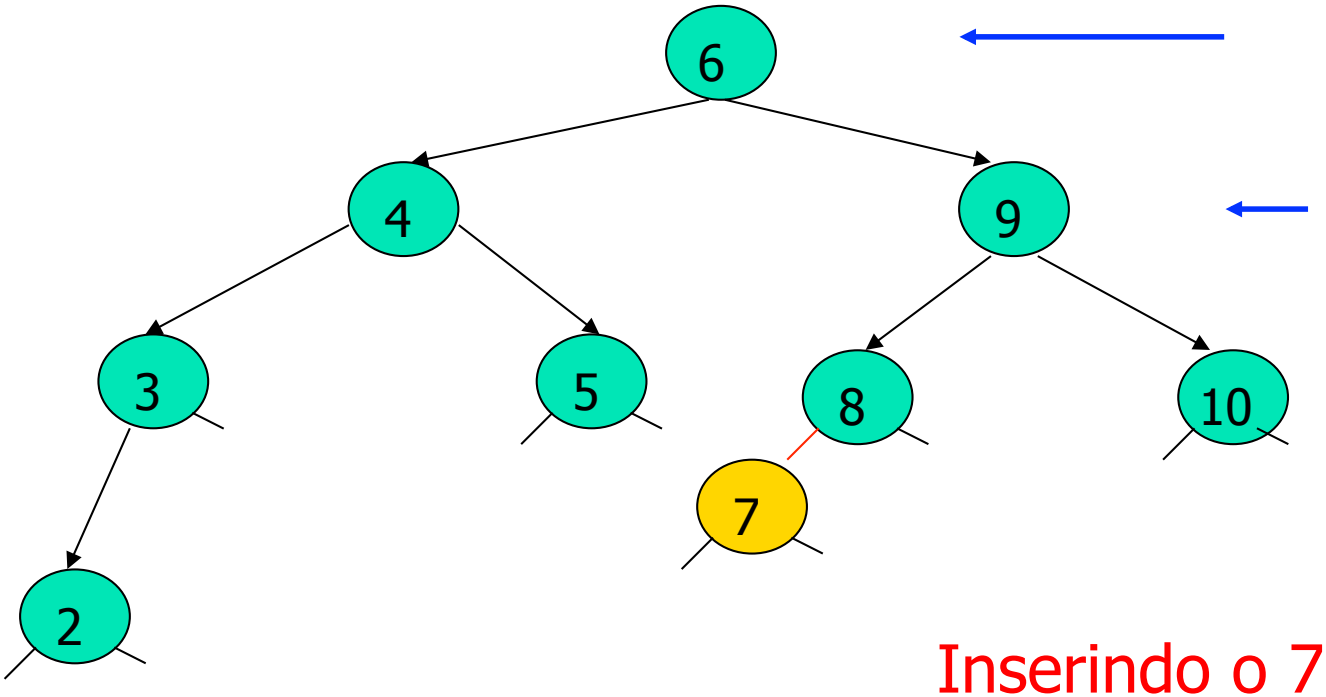
- Para inserir um nó na árvore:
  - fazer uma busca com insucesso
  - alocar um novo nó
  - é necessário saber por qual nó se chegou a NULL
    - será o pai do novo nó

Inserindo em Árvore Binária de Busca

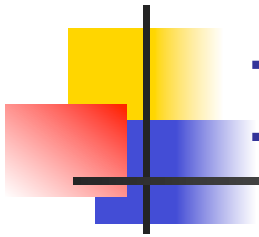


Inserindo o 9

Inserindo em Árvore Binária de Busca



Inserindo o 7



Inserção Árvore Binária de Busca

```
insere_árvore (int valor, tipo_nó * pt)
{




}
```

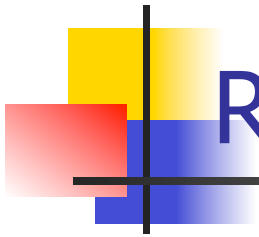
Inserção Árvore Binária de Busca

```
insere_árvore (int valor, tipo_nó * pt)
{ tipo_nó * pai;
   do{ pai = pt ;
         if (valor < pt->chave)  pt = pt ->esq ;
         else if ( valor > pt-> chave) pt = pt->dir;
    } while(pt != NULL) && (pt->chave != valor);
    if (pt == NULL){
          pt = aloca();
          pt ->chave = valor; pt->esq = NULL; pt->dir = NULL;
          if (v < pai->chave)  pai ->esq = pt ;
          else pai ->dir = pt ;
          return(pt);
      }
}
```

Inserção Árvore Binária de Busca

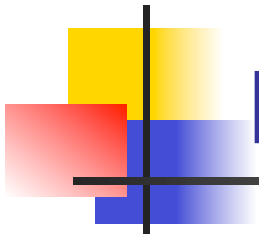
- a árvore está classificada se percorrida da forma correta (pre, pos ou em-ordem?)
  - as chaves aparecem em ordem se lidas da esquerda para a direita

- podemos ordenar uma sequência como se fosse uma série de inserções
  - o programa tem apenas os ponteiros para se preocupar
  - qual a diferença



Remoção em Árvore Binária de Busca

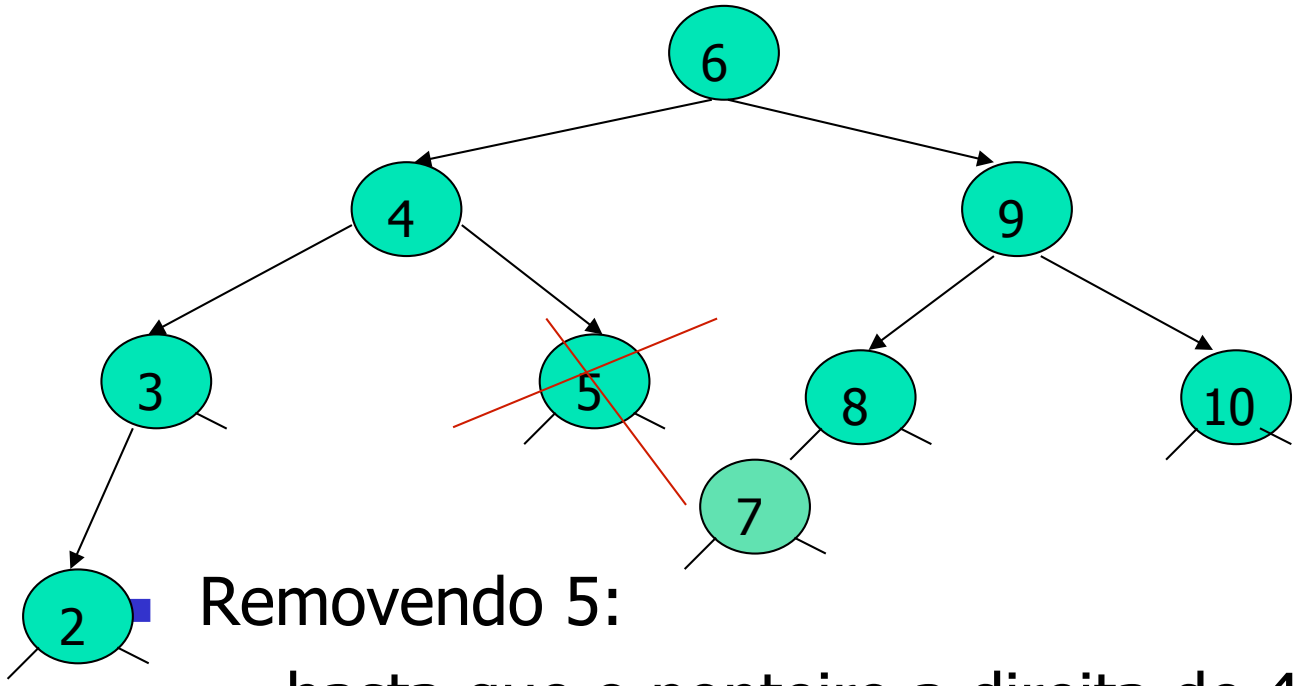
- até então, vimos que a implementação da operação de inserção é simples
- a remoção de um elemento já é mais complexa
  - remoção de um nó folha
    - os ponteiros esquerdo e direito do pai são setados para NULL
  - se possui apenas um filho
    - o ponteiro apropriado do pai passa a apontar para o filho
  - se o nó possui dois filhos
    - se um desses dois filhos não possui filhos, use esse nó para substituir o nó removido



Remoção em Árvore Binária de Busca

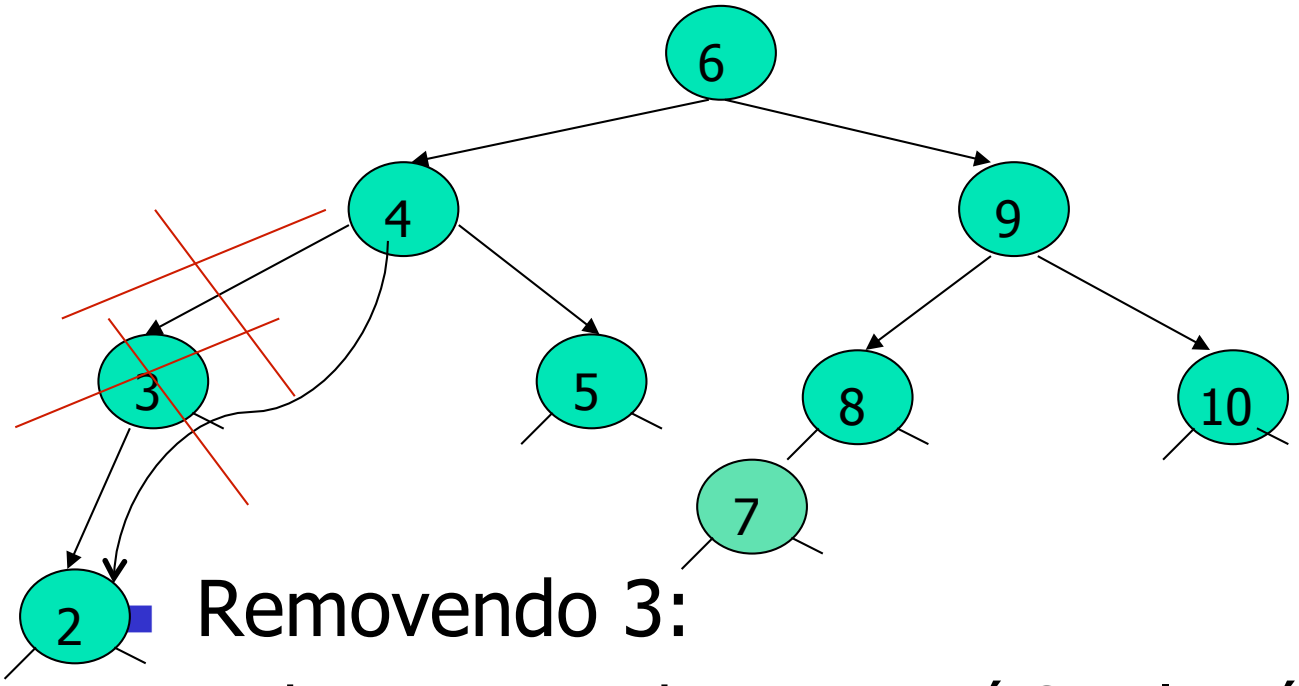
- senão: substituir o valor do nó a ser removido
  - substitua este com o elemento cuja chave é imediatamente maior (ou menor)
    - é sempre folha?
  - senão for folha – vá repetindo o procedimento
  - algoritmo?

Remoção em Árvore Binária de Busca



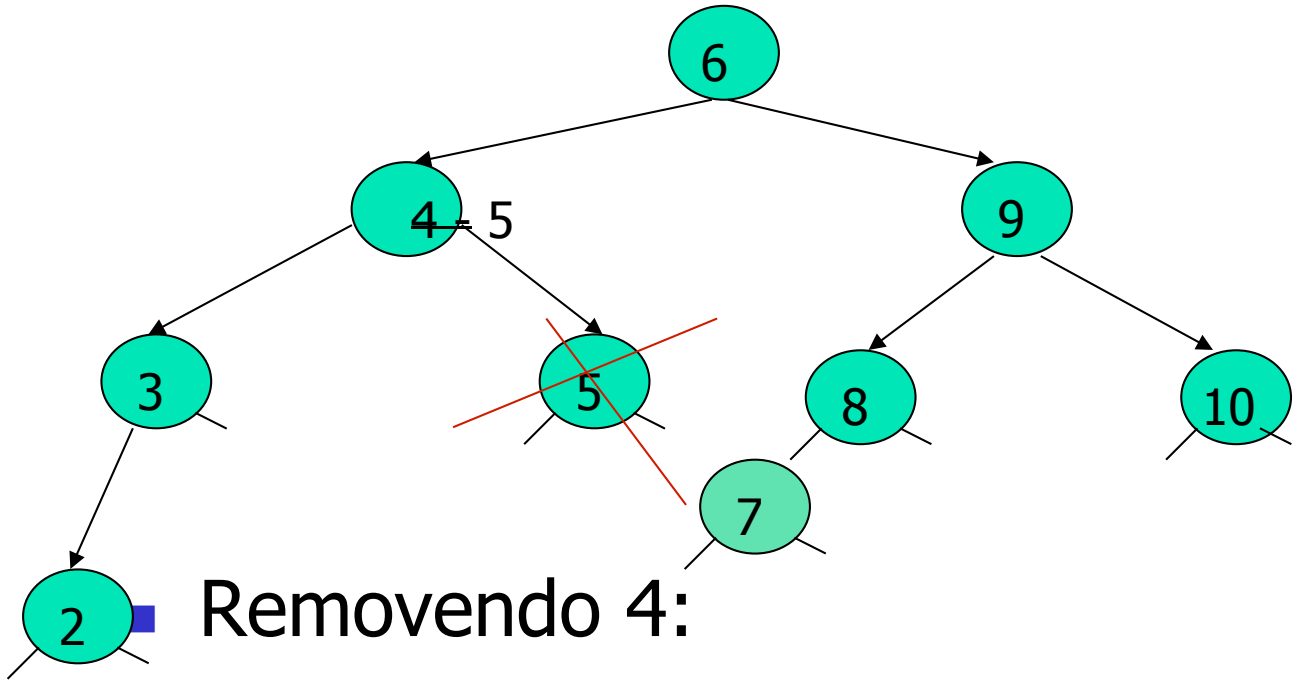
- Removendo 5:
  - basta que o ponteiro a direita de 4 aponte para NULL

Remoção em Árvore Binária de Busca



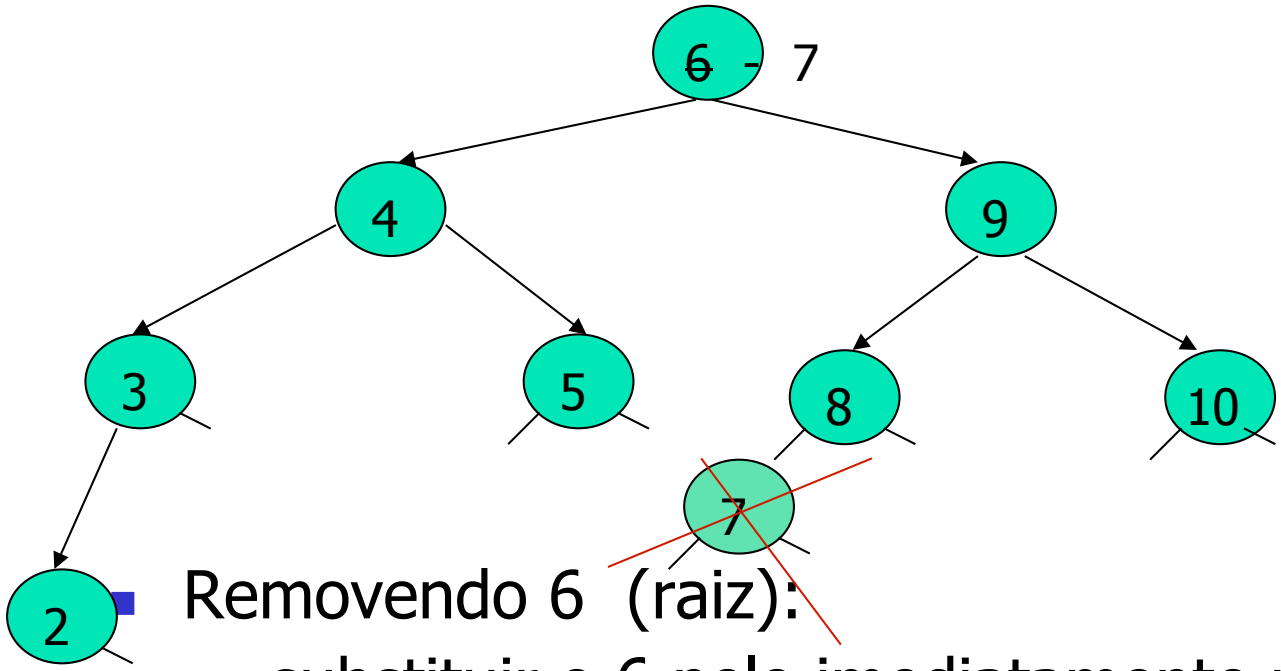
- Removendo 3:
  - basta que substituir o nó 3 pelo nó 2
  - continua valendo a regra de formação da árvore

Remoção em Árvore Binária de Busca



- Removendo 4:
  - basta que o substituir 4 pelo 5
  - continua valendo a regra de formação da árvore

Remoção em Árvore Binária de Busca



- Removendo 6 (raiz):
  - substituir o 6 pelo imediatamente maior: 7 – como determinar?
  - resulta na remoção do elemento de chave 7

Remoção em Árvore Binária de Busca

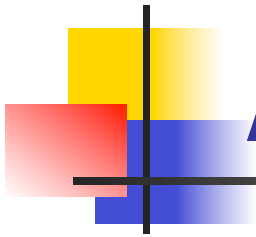
```
remove_ABB (int valor, tipo_nó * pt)
{ tipo_nó * pai;
    /* a busca  */
    do{ pai = pt ;
        if (valor < pt->chave)  pt = pt ->esq ;
        else if ( valor > pt-> chave) pt = pt->dir;
   } while(pt != NULL) && (pt->chave != valor);

    if (pt != NULL){  /* ok, encontrou o valor na ABB */
```

Remoção em Árvore Binária de Busca

```
if (pt != NULL){  /* ok, encontrou o valor na ABB */
      /* se é um nó interno com duas subárvores vazias */
      if (p-> esq != NULL) && (pt->dir != NULL){
          ptaux = pt;
          pai = pt; pt = pt->dir;
          while (pt->esq != NULL) { /* acha o imediatamente maior */
                    pai = pt;
                    pt = pt->esq;
          }
      /* troca o valor do nó a ser retirado pelo imediatamente maior */
          ptaux->valor = pt->valor;
      }
```

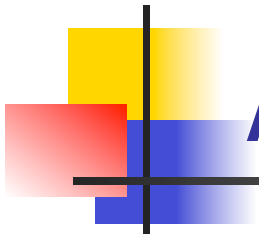
Remoção em Árvore Binária de Busca

```
if (pt != NULL){  /* ok, encontrou o valor na ABB */
        /* se é um nó interno com duas subárvores vazias */
        .......
        if (pt->esq != NULL) && (pt-> dir == NULL){
                    /* só tem o filho esq */
                    if (pai->esq ==pt) pai->esq = pt->esq;
                    else pai->dir = pt->esq;
        } else if (pt->esq == NULL) && (pt-> dir != NULL){
                    /* só tem o filho direito */
                    if (pai->esq ==pt) pai->esq = pt->dir;
                    else pai->dir = pt->dir;
                } else{/* então é folha */
                    if (pai->esq ==pt) pai->esq = NULL;
                    else pai->dir = NULL;
                }
        free (pt);
}
```



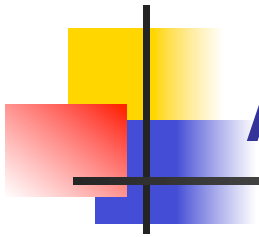
Árvore Binária de Busca

- A árvore obtida depende da seqüência de inserção de nós
- Para que a árvore binária de busca seja completa
  - completa tem altura mínima
  - o conjunto das chaves deve ser reordenado
  - árvore comum - *O* (n)
  - árvore completa - *O* (log n)



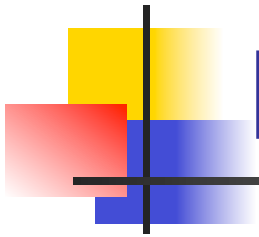
Árvore Binária de Busca

- uma árvore binária de busca completa
  - o conjunto das chaves deve ser re-ordenado
  - sejam s_0 e s_{n+1} duas chaves fictícias e já inseridas
  - a cada passo inserir em T uma nova chave que seja de índice médio entre i e j - duas chaves já inseridas



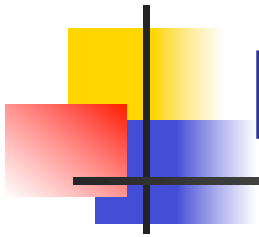
Árvore Binária de Busca

- árvore completa: ótima para a busca
  - quando a freqüência de acesso aos nós é igual
- normalmente estas freqüências são diferentes
- é interessante construir uma árvore binária que seja a melhor possível no que diz respeito à busca para freqüências conhecidas



Eficiência da Árvore de Busca

- Depende da ordem original dos dados
- Se o array original está ordenado (ascendente ou descendente), as árvores resultantes só tem filhos a direita ou a esquerda
  - a inserção do 1o. nó - 0 comparações
  - a inserção do 2o. nó - 2 comparações
  - a inserção do 3o. nó - 3 comparações
- 2 + 3 +....+n = n*(n+1)/2 -1
- Complexidade - $O(n^2)$ - para inserir n nós



Eficiência da Árvore de Busca

- Se a lista original estiver organizada, e se uma árvore completa (parecida com completa) for se formando:
  - complexidade da inserção = $O(n \log n)$