# Árvore Binária de Busca

## Árvore Binária de Busca (ABB)

Uma ABB é uma AB tal que para todo nó x:

- todos elementos chaves da subárvore esquerda de x são menores que a chave x
- todos elementos chaves da subárvore direita de x são maiores que a chave x
- as subárvores esquerda e direita são ABB

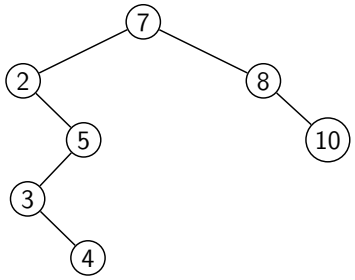
Um campo chave (key) é um dado de identificação única: RG, CPF, fragmento de DNA,...

## Mínimo e Máximo

- o elemento mínimo em uma ABB pode ser encontrado seguindo-se as subárvores esquerdas desde a raiz
- o elemento máximo em uma ABB pode ser encontrado seguindo-se as subárvores direitas desde a raiz

## Exemplo

7 8 2 5 8 3 5 10 4




## Busca em uma ABB

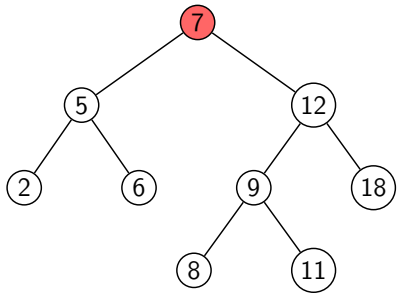
- Se x é a raiz da ABB então x foi encontrado, caso contrário
- Se x é menor que a raiz então procure x na sub-árvore esquerda, caso contrário
- Procure x na sub-árvore direita de x
- Se a ABB é vazia então a busca falha

## Árvore Binária de Busca

- também conhecidas como árvores de pesquisa ou árvores ordenadas
- tem utilidade de armazenar dados que são freqüentemente verificados (busca!)
- pode sofrer alterações (inserções e remoções de nós) após ter sido criada

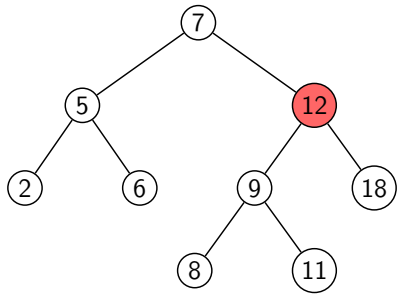
## Busca (x $= 9$)

- x $> 7$ – procurar na subárvore direita



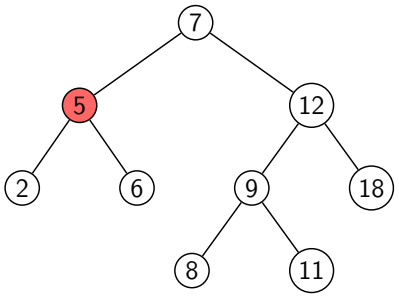
## Busca (x = 9)

- x < 12 – procurar na subárvore esquerda



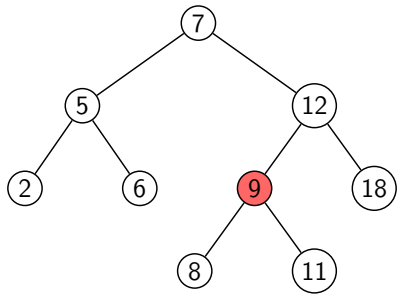

## Busca (x = 4)

- x < 5 – procurar na subárvore esquerda



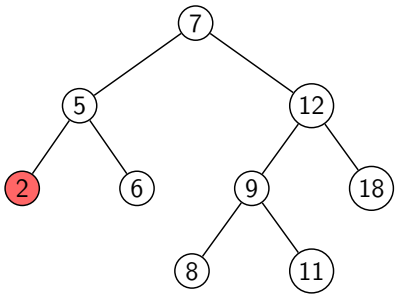

## Busca (x = 9)

- x = 9 – retornar o elemento encontrado!



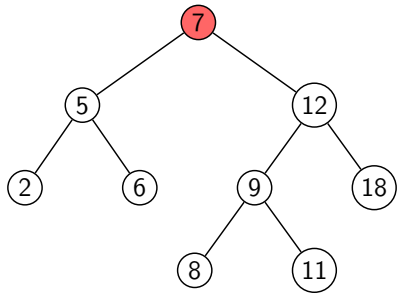

## Busca (x = 4)

- x > 2 – procurar na subárvore direita



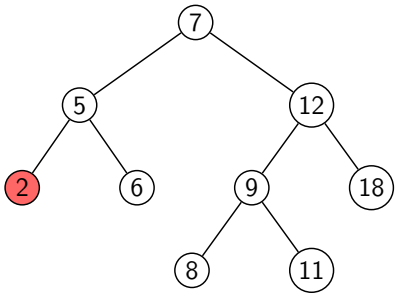

## Busca (x = 4)

- x < 7 – procurar na subárvore esquerda



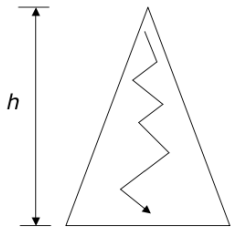

## Busca (x = 4)

- direita de 2 é vazia – falhou a busca



## Número de Comparações

- pior caso – ordem da altura da árvore
- para árvores perfeitamente balanceadas $h \approx \log_2 n$ $\Rightarrow$ $O(\log_2 n)$ comparações




## Inserção em ABB

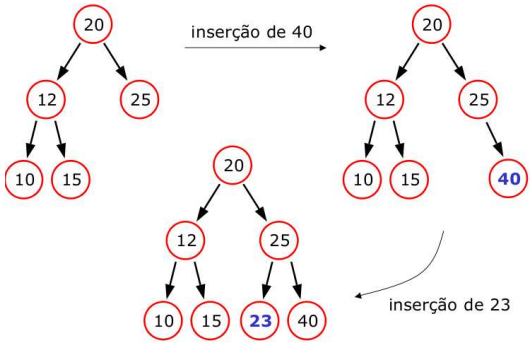
- se a árvore estiver vazia, adicione um novo nó contendo o elemento x
- se a raiz é maior que x então insira x na subárvore esquerda
- se a raiz é menor que x então insira x na subárvore direita

Vamos pensar na versão recursiva e iterativa!!

## Implementação da Busca (versão iterativa)

```
while (t != NULL && t->info != x)
    if(x < t->info)
      t = t->esq; // procurar subárvore esquerda
    else
      t = t->dir; // procurar subárvore direita
```

## Inserção - Exemplos




## Implementação da Busca (versão recursiva)

```
if(t == NULL)
  return 0; // x não encontrado

if(x < t->info)
  return busca(x,t->esq);
else
  if(x > t->info)
    return busca(x,t->dir);
  else     // x == t->info
    return 1; // encontrou x
```

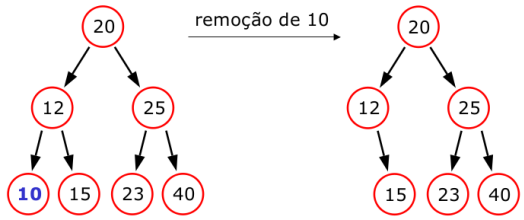
## Remoção em ABB

Possibilidades:

- o elemento é uma folha: remove-se simplesmente o elemento
- o elemento é um nó interno:
  - tem apenas um filho: esse filho é colocado no lugar do elemento a remover (seu pai)
  - tem dois filhos: coloca-se na posição do elemento a remover o menor elemento da subárvore direita (transferência de posição)

## Remoção de um nó folha

Remoção simples

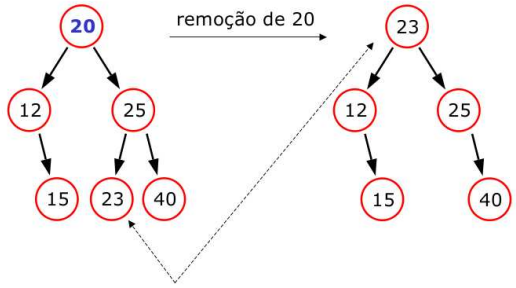



## Considerações Finais

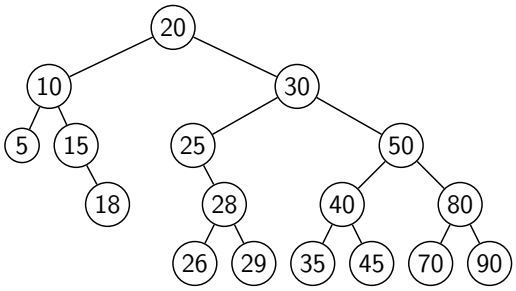
- Em uma ABB perfeitamente balanceada com $n$ nós e altura $h$, os algoritmos de busca e inserção tomam tempo $O(h) = O(\log_2 n)$
- Porém, uma árvore pode se degenerar em uma lista (pior caso – algoritmos levam $O(n)$)
- Em média $O(n/2)$
- A situação no pior caso conduz a um desempenho muito pobre

## Remoção de um nó com dois filhos

23 é o menor da subárvore direita do nó a remover



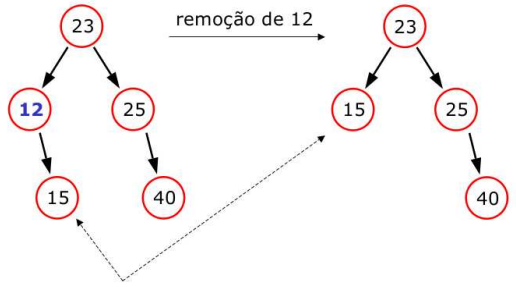

## Exercícios




Remover 10 50 70 20

## Remoção de um nó com um filho

O filho passa a ocupar o lugar do pai




## Exercício

Em uma árvore binária de busca, as seguintes chaves foram inseridas na seqüência: 50, 30, 80, 100, 10, 20, 98, 95, 12, 15, 25, 40, 150, 1, 7 e 3. Construa a árvore binária e descreva os percursos em-ordem, pós-ordem, pré-ordem e em largura.

## Exercícios

Uma árvore binária de busca foi percorrida em pré-ordem: 8, 7, 3, 2, 1, 5, 4, 6, 9, 11, 10, 15, 13, 12, 14, 19, 17, 16, 18, 20, 21

1. Construa a árvore binária de busca correspondente.
2. Percorra em pós-ordem
3. Percorra em em-ordem

## Exercício

Uma das aplicações interessantes de árvores binárias é a compactação de arquivos usando os códigos de Huffman. Os códigos de Huffman são códigos binários (atribuídos, por exemplo, a caracteres em um texto) de comprimentos variados que são determinados a partir da frequência de uso de um determinado caracter. A idéia central é associar números binários com menos bits aos caracteres mais usados num texto, possibilitando a sua compactação.

Estude e implemente o algoritmo de Huffman.

## Bibliografia


- José Augusto Baranauskas, notas de aula da disciplina de *Algoritmos e Estruturas de Dados I*, Departamento de Física e Matemática-USP.
- Material de apoio para a disciplina de *Algoritmos e Estruturas de Dados I*, Departamento de Informática - Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa