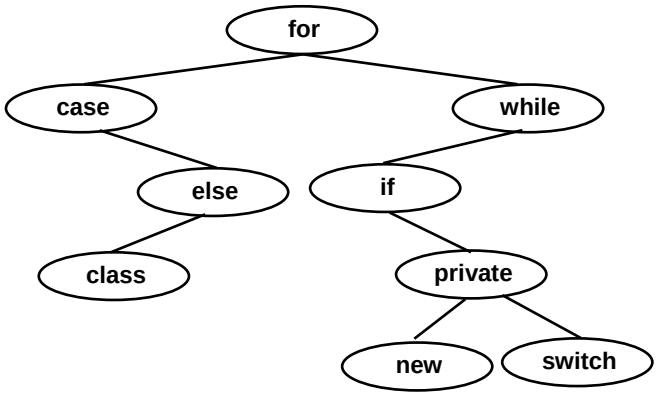
Árvores Binárias de Busca

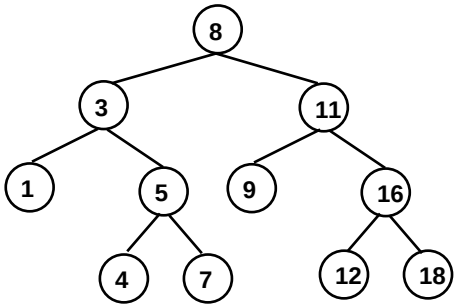
Uma Árvore Binária de Busca T (ABB) ou Árvore Binária de Pesquisa é tal que ou T = 0 e a árvore é dita vazia ou seu nó raiz contém uma chave e:

1. Todas as chaves da sub-árvore esquerda são menores que a chave da raiz.
2. Todas as chaves da sub-árvore direita são maiores que a chave raiz.
3. As sub-árvores direita e esquerda são também ***Árvores Binárias de Busca***.

Tabelas de Palavras reservadas de um compilador Java:



Em algoritmo de busca a medida de eficiência é dada pelo número de comparações necessárias para se localizar uma chave, ou descobrir que ela não existe.



Numa lista linear com ***n*** chaves, temos que, no pior caso fará ***n*** comparações. O número de comparações cresce linearmente em função do número de chaves.

Um percurso *em-ordem* nessa árvore resulta na seqüência de valores em ordem crescente.

Uma árvore de busca criada a partir de um conjunto de valores não é única: o resultado depende da seqüência de inserção dos dados.
A grande utilidade da árvore binária de busca é armazenar dados contra os quais outros dados são freqüentemente verificados (***busca!***)
Uma árvore de busca binária é dinâmica e pode sofrer alterações (inserções e remoções de nós) após ter sido criada

Busca Binária

Pesquisa realizada se informação está armazenada de forma ordenada e em seqüência (em um vetor).

Qual a eficiência do Algoritmo de busca binária?

- Para ***n = 8*** posições, o pior caso é quando precisamos chegar à posição 1 ou 8.
- A cada divisão, desprezamos toda a porção maior ou menor que a chave (posição do meio a cada iteração).

Com busca binária obtemos a melhor eficiência possível em um vetor, mas ficamos limitados à representação seqüencial (deslocamentos, previsão de memória, etc.).
Podemos utilizar a ***Árvore Binária de Busca*** e obteremos o mesmo desempenho anterior - desde que a altura seja mínima. É a característica de altura mínima que garante que estamos tomando a chave do meio da porção pesquisada!
Com ***Árvore Binária de Busca Perfeitamente Balanceada*** temos altura mínima. O número de comparações será igual a:

$$h_{min} = \text{menor inteiro maior ou igual a } log_2(n+1)$$

Para garantir o desempenho ótimo temos que garantir que a árvore seja balanceada durante a construção e que o balanceamento seja mantido em inserções e eliminações!

Operações em uma Árvore Binária de Busca

As operações básicas em uma Árvore Binária de Busca são:

- inserção
- remoção
- busca

Inserção

Passos do algoritmo de inserção:

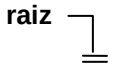
- Procure um "local" para inserir o novo nó, começando a procura a partir do nó-raiz;
- Para cada nó-raiz de uma sub-árvore, compare; se o novo nó possui um valor menor do que o valor nó raiz (vai para sub-árvore esquerda), ou se o valor é maior que o valor no nó-raiz (vai para sub-árvore direita);
- Se uma referência (filho esquerdo/direito de um nó raiz) nula é atingida, coloque o novo nó como sendo filho do nó-raiz.

Exemplo

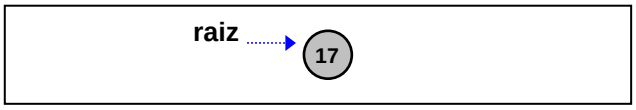
Para entender o algoritmo considere a inserção do conjunto de números, na seqüência:

17, 99, 13, 1, 3, 100, 400

1. No início a ABB está vazia!

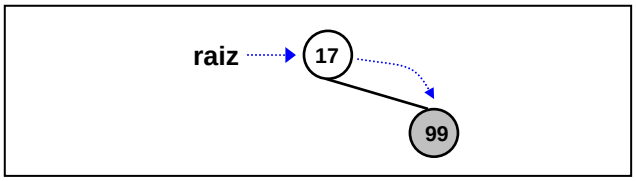


2. O número ***17*** será inserido tornando-se o nó raiz;

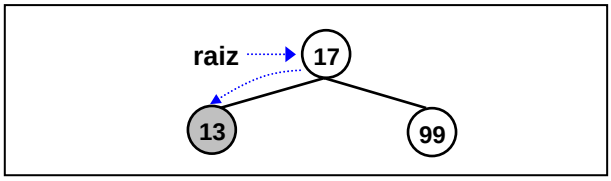


Erro!

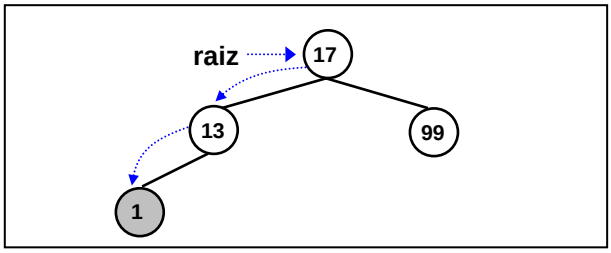
3. A inserção do ***99*** inicia-se na raiz. Compara-se ***99*** com ***17***. Como ***99 > 17***, ***99*** deve ser colocado na sub-árvore direita do nó contendo ***17*** (sub-árvore direita, inicialmente, nula);



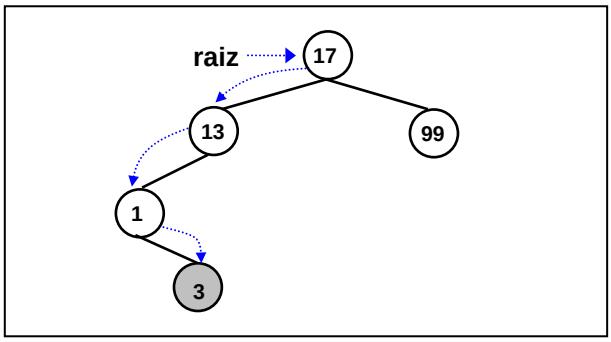
4. A inserção do ***13*** inicia-se na raiz. Compara-se ***13*** com ***17***. Como ***13 < 17***, ***13*** deve ser colocado na sub-árvore esquerda do nó contendo ***17***. Já que o nó ***17*** não possui descendente esquerdo, ***13*** é inserido na árvore nessa posição.



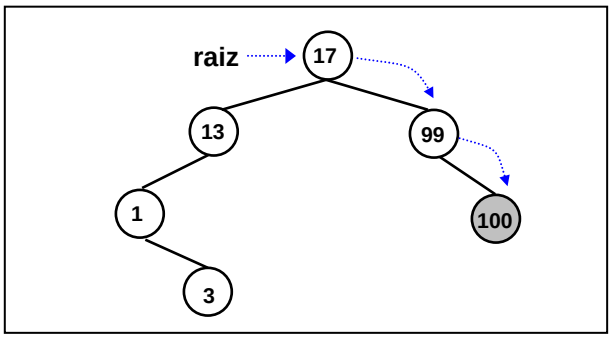
5. Para inserir o valor ***1***, repete-se o procedimento. ***1<17***, então será inserido na sub-árvore esquerda. Chegando nela, encontra-se o nó ***13***, ***1<13,*** então ele será inserido na sub-árvore esquerda de ***13***.



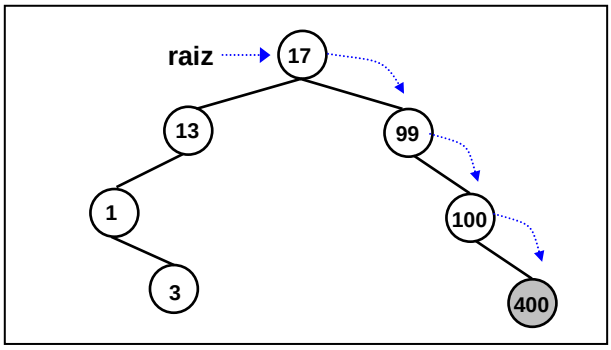
6. Para inserir o valor ***3***, repete-se o procedimento. ***3<17***, então será inserido na sub-árvore esquerda. Chegando nela, encontra-se o nó ***13***, ***3<13***. Chegando à sub-árvore esquerda encontra-se o nó ***1***, ***3>1,*** então ele será inserido na sub-árvore esquerda de ***13***.



7. Repete-se o procedimento para inserir o elemento ***100***:
100 > 17 (vai para a direita) e
100 > 99 (vai para a direita);



8. Repete-se o procedimento para inserir o elemento ***400***:
400 > 17 (vai para a direita)
400 > 99 (vai para a direita)
400 > 100 (vai para a direita)

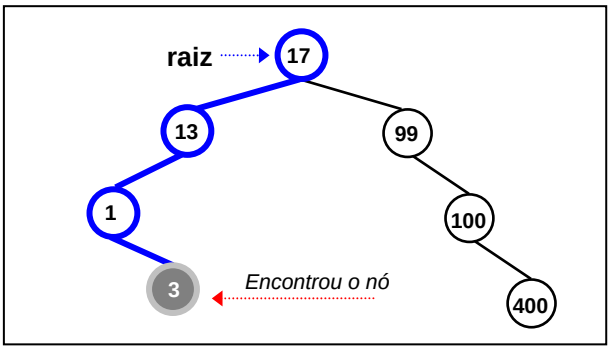


Busca

Passos do algoritmo de busca

1. Comece a busca a partir do nó-raiz;
2. Para cada nó-raiz de uma sub-árvore compare:
   Se o valor procurado **é menor** que o valor no nó-raiz (continua pela ***sub-árvore esquerda***), ou se o valor **é maior** que o valor no nó-raiz (***sub-árvore direita***);
   Caso o nó contendo o valor pesquisado seja **encontrado**, retorne **o nó**; **caso contrário** retorne **nulo**.

Por exemplo, para encontrar a chave ***3***, o caminho de busca é representado a seguir:



Exercícios

1. Implemente o método de inserção para a Árvore de Busca Binária, que armazene valores inteiros.

2. Implemente o método que realiza a busca de um elemento dentro de uma Árvore de Busca Binária.

Obs: *Para os métodos pedidos, faça a implementação no modo iterativo e recursivo!*

Remoção

Para a remoção de um nó em uma árvore binária, devem ser considerados três casos:

Caso 1: o nó é folha
O nó pode ser retirado sem problema;

Caso 2: o nó possui uma sub-árvore (esq./dir.)
O nó-raiz da sub-árvore (esq./dir.) "ocupa" o lugar do nó retirado;

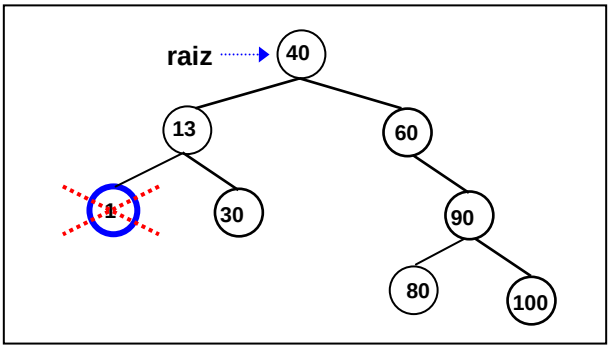
Caso 3: o nó possui duas sub-árvores
O nó contendo o menor valor da sub-árvore direita pode "ocupar" o lugar; ou o maior valor da sub-árvore esquerda pode "ocupar" o lugar

Exemplos

Caso 1: Remoção do nó ***1***

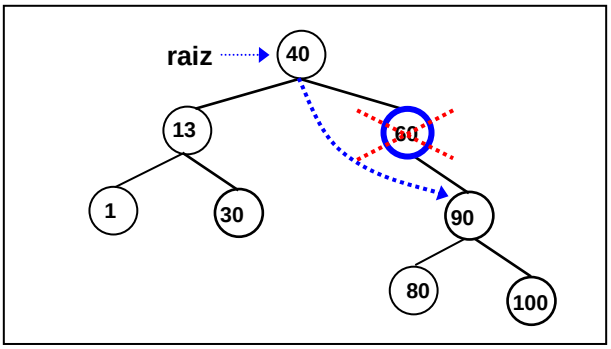
Ele pode ser removido sem problema, pois não requer ajustes posteriores.

Os nós ***30, 80*** e ***100*** também podem ser removidos sem problemas!



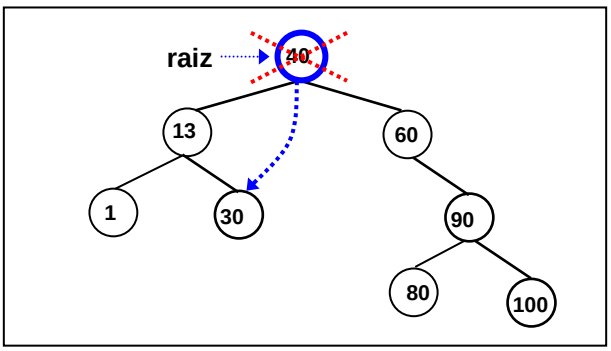
Caso 2: Remoção do nó ***60***

Como ele possui apenas a sub-árvore direita, o nó contendo o valor ***90*** pode "ocupar" o lugar do nó removido.

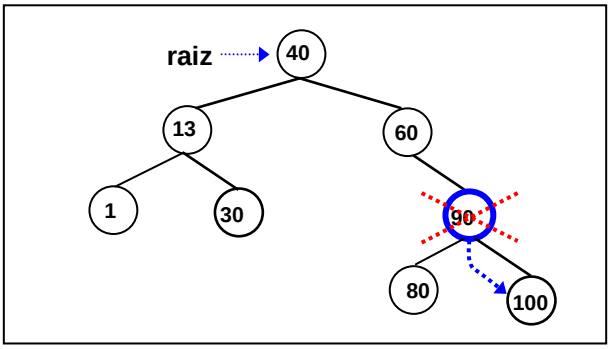


Caso 3: Remoção do nó ***40***

Neste caso, existem ***2 opções***:
- O nó com valor ***30*** pode "ocupar" o lugar do nó-raiz, ou
- O nó com valor ***60*** pode "ocupar" o lugar do nó-raiz..



Este caso também se aplica ao nó ***90***:
- O nó com valor ***80*** pode "ocupar" o lugar do nó-raiz, ou
- O nó com valor ***100*** pode "ocupar" o lugar do nó-raiz.



Importante: Uma vez definida a regra de escolha do nó substituto, ela deve ser a mesma para todas as operações de remoção!

Custo da busca em ABB

Pior caso: número de passos é determinado pela altura da árvore.

A altura da Árvore de Busca Binária depende da seqüência de inserção das chaves.
Considere, por exemplo, o que acontece se uma seqüência ordenada de chaves é inserida.
Seria possível gerar uma árvore balanceada com essa mesma seqüência, se ela fosse conhecida *a priori*.
A busca pode ser considerada eficiente se a árvore estiver razoavelmente balanceada.