CC BY

Receita de linguiça: tira a tripa do porco e põe o porco na tripa.

Árvore de Busca Binária

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

Struct

Geralmente usamos uma struct similar a seguinte:

```
struct nodo{
   int chave;
   struct nodo* pai;
   struct nodo* fe;
   struct nodo* fd;
};
```

Struct

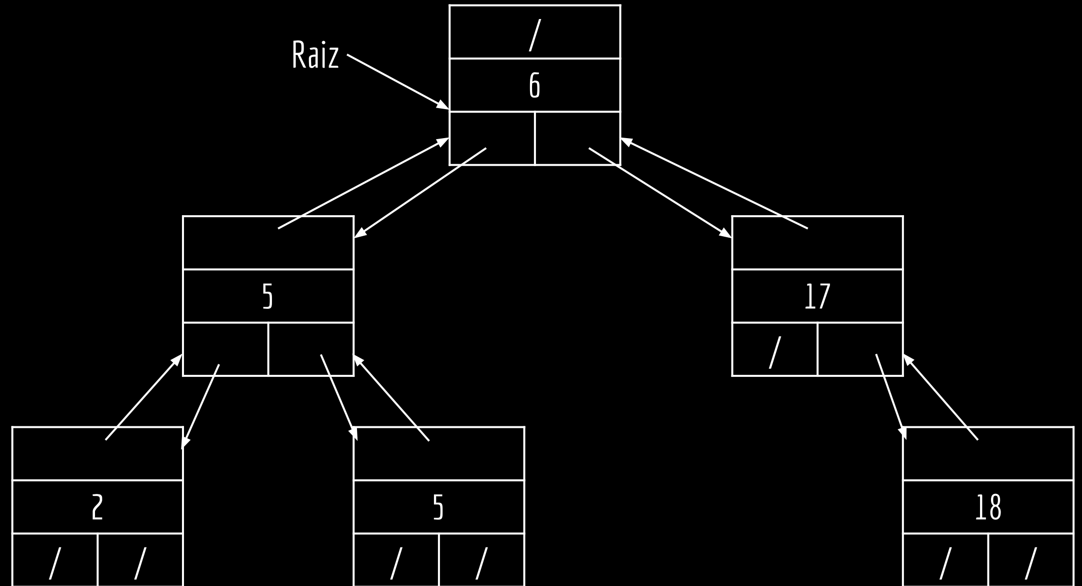
Geralmente usamos uma struct similar a seguinte:

```
struct nodo{
    int chave;
    struct nodo* pai;
    struct nodo* fe;
    struct nodo* fd;
};
```

Adicionamos um ponteiro para o pai do nodo.
Veremos adiante como isso pode facilitar nossa vida.

Struct

```
struct nodo{
   int chave;
   struct nodo* pai;
   struct nodo* fe;
   struct nodo* fd;
};
```



Propriedade

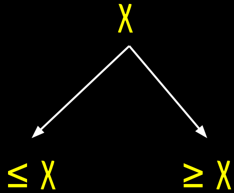
Propriedade da árvore de busca binária:

Seja x um nodo qualquer na árvore de busca binária. Se y é um nodo que está na subárvore esquerda de x, então y.chave ≤ x.chave. Se y é um nodo que está na subárvore direita de x, então y.chave ≥ x.chave.

Propriedade

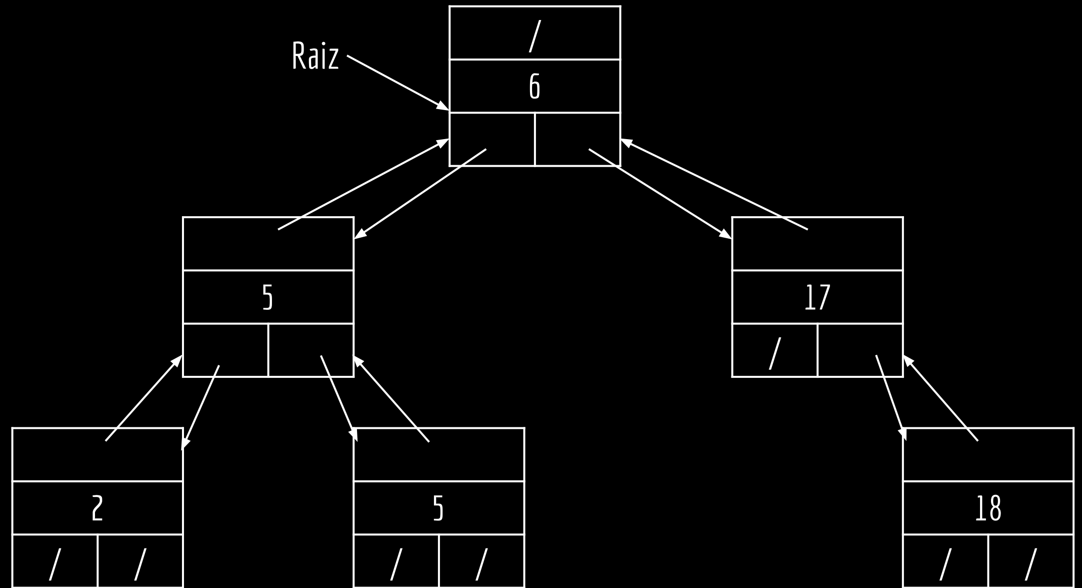
Propriedade da árvore de busca binária:

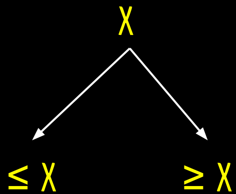
Seja x um nodo qualquer na árvore de busca binária. Se y é um nodo que está na subárvore esquerda de x, então y.chave ≤ x.chave. Se y é um nodo que está na subárvore direita de x, então y.chave ≥ x.chave.



Pergunta

A árvore ao lado é uma árvore de busca binária?

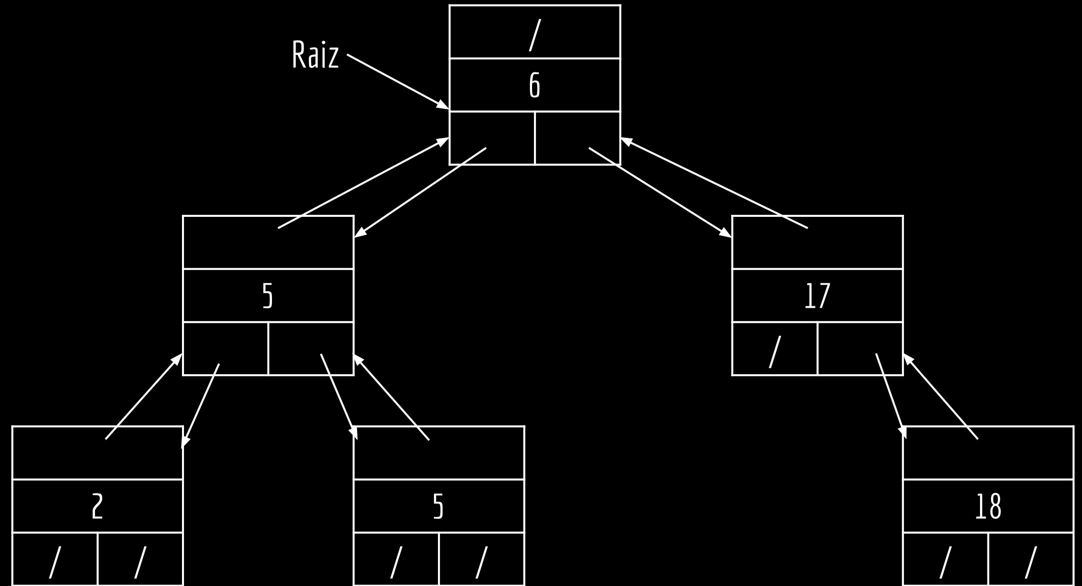


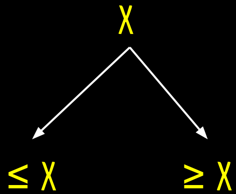


Pergunta

A árvore ao lado é uma árvore de busca binária?

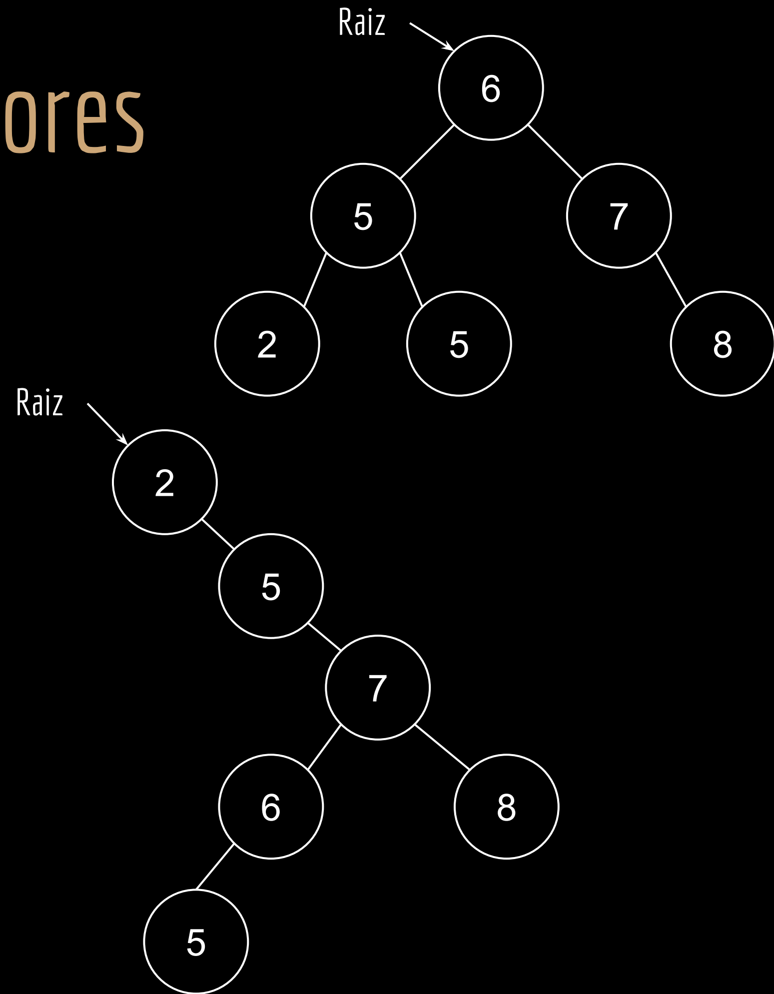
Sim.





Mesmas chaves, diferentes árvores

Quais das árvores a seguir são árvores de busca binária.



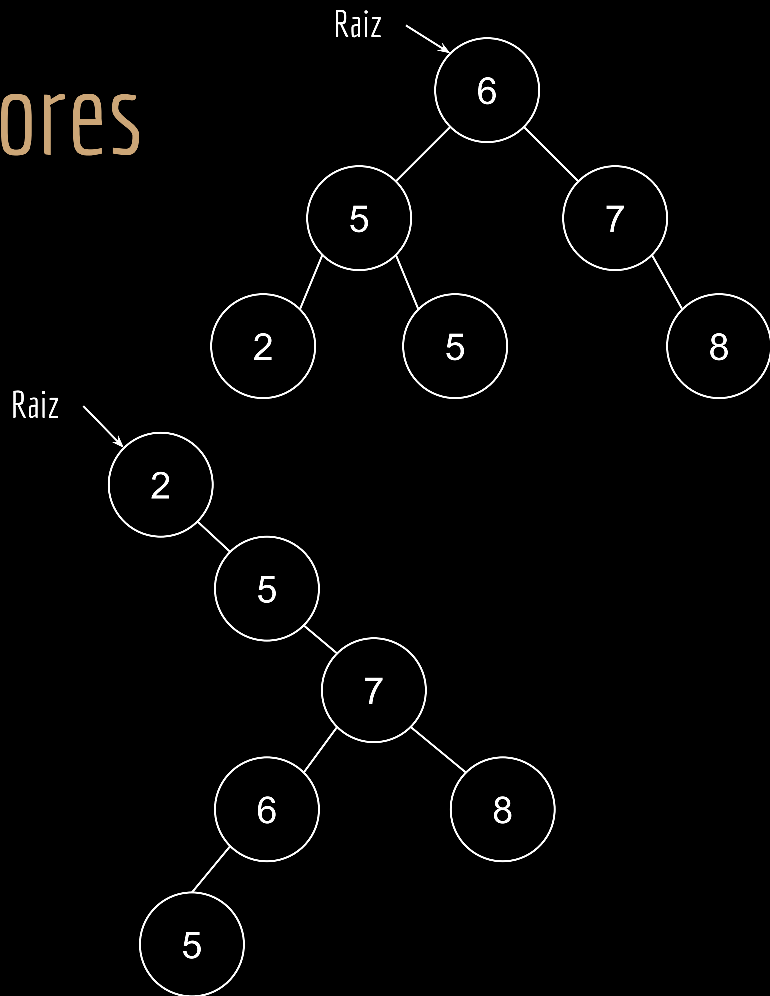
Mesmas chaves, diferentes árvores

Quais das árvores a seguir são árvores de busca binária.

Ambas são árvores de busca binária.

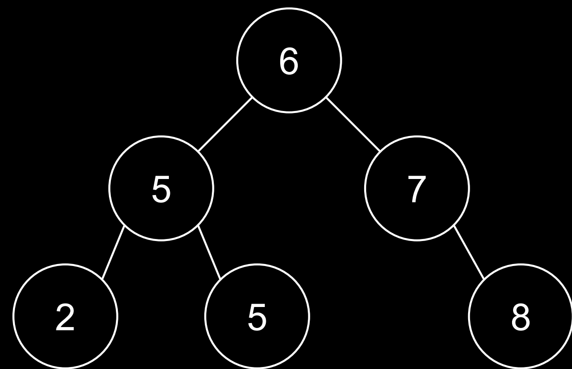
E ambas as árvores possuem as mesmas chaves!

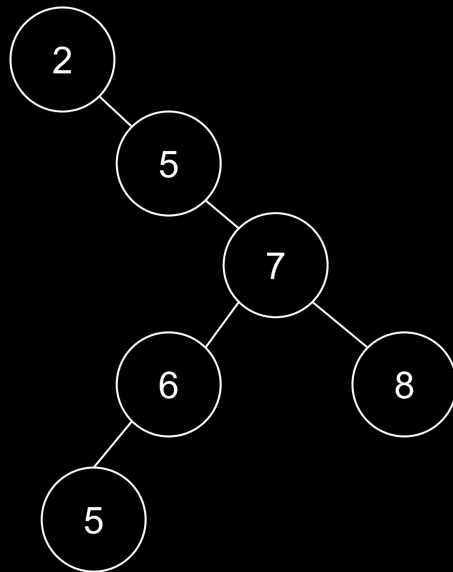
Podem existir múltiplas árvores de busca binária para um mesmo conjunto de chaves.



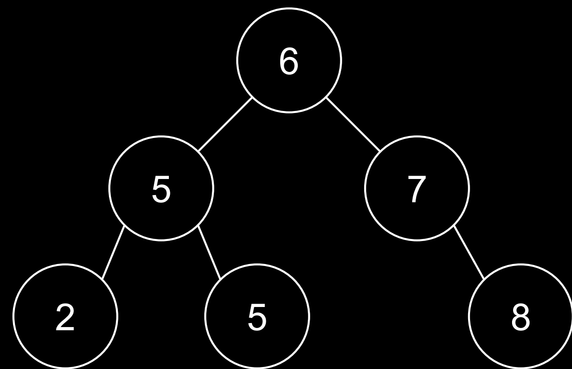
Faça você mesmo

Para o mesmo conjunto de chaves das árvores ao lado, crie uma terceira árvore de busca binária diferente.





Questão de ordem



A propriedade da árvore de busca binária força uma ordem entre os itens.

Podemos facilmente, por exemplo, imprimir os itens da árvore em ordem.

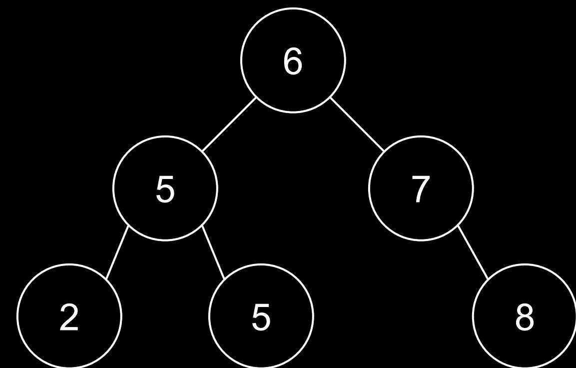
Como?

Questão de ordem

A propriedade da árvore de busca binária força uma ordem entre os itens.

Podemos facilmente, por exemplo, imprimir os itens da árvore em ordem.

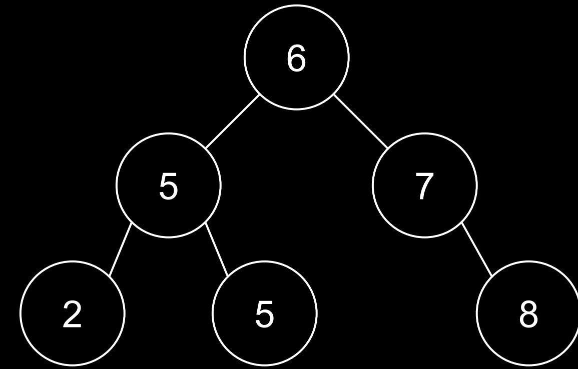
Como?



- Aplicar em-ordem na subárvore esquerda.
- Visitar a raiz.
- Aplicar em-ordem na subárvore direita.

Questão de ordem

Imprimir os itens da árvore em ordem usando o algoritmo em-ordem.



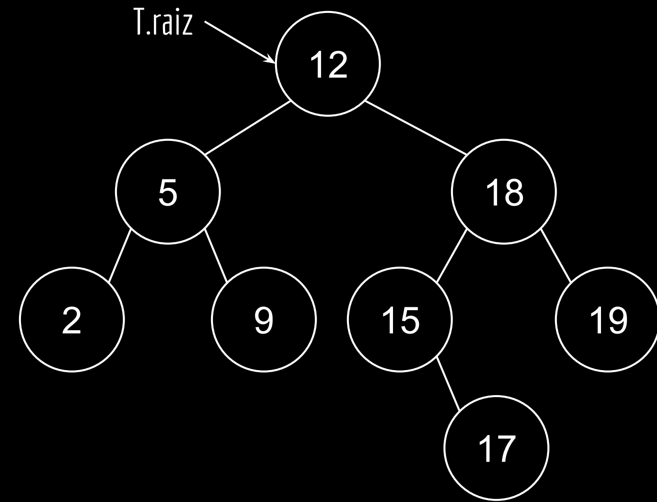

```
função emOrdem(r)
entrada: nodo raiz da subárvore de busca binária r.
saída: a árvore é impressa em ordem.
se r não é NULO
     emOrdem(r.fe)
     imprima r.chave
     emOrdem(r.fd)
```

Inserção

```
função inserir(T,chave)
entrada: árvore de busca binária T, e a chave a ser inserida.
saída: um novo nodo com a chave é inserido na posição correta na árvore.

novo.chave = chave
novo.fe = NULO
novo.fd = NULO

atual = T.raiz
pai = NULO
enquanto atual não é NULO
     pai = atual
     se novo.chave < atual.chave
          atual = atual.fe
     senão
          atual = atual.fd
novo.pai = pai
se pai é NULO
     T.raiz = novo //a árvore estava vazia
senão
     se novo.chave < pai.chave
          pai.fe = novo
     senão
          pai.fd = novo
```

Teste de mesa

```
função inserir(T,chave)

novo.chave = chave
novo.fe = NULO
novo.fd = NULO

atual = T.raiz
pai = NULO
enquanto atual não é NULO
      pai = atual
      se novo.chave < atual.chave
            atual = atual.fe
      senão
            atual = atual.fd
novo.pai = pai
se pai é NULO
      T.raiz = novo //a árvore estava vazia
senão
      se novo.chave < pai.chave
            pai.fe = novo
      senão
            pai.fd = novo
```



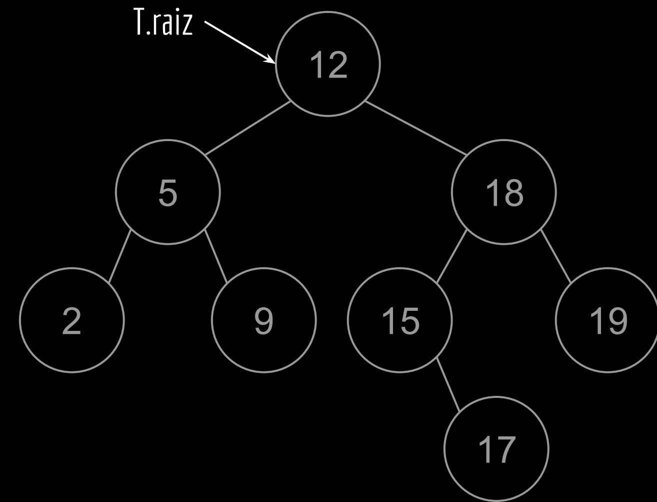

```
função inserir(T,13)
```

atual pai novo

Teste de mesa

```
função inserir(T,chave)

novo.chave = chave
novo.fe = NULO
novo.fd = NULO

atual = T.raiz
pai = NULO
enquanto atual não é NULO
      pai = atual
      se novo.chave < atual.chave
            atual = atual.fe
      senão
            atual = atual.fd
novo.pai = pai
se pai é NULO
      T.raiz = novo //a árvore estava vazia
senão
      se novo.chave < pai.chave
            pai.fe = novo
      senão
            pai.fd = novo
```

13

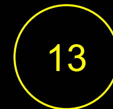


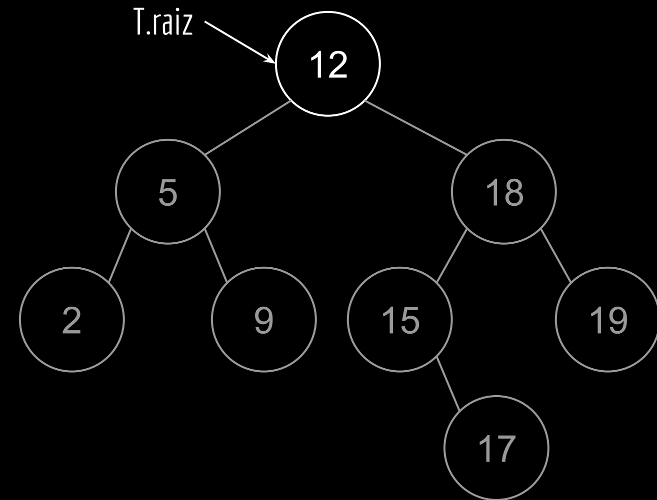
função **inserir**(T,13)

atual	pai	novo
		13

Teste de mesa

```
função inserir(T,chave)

novo.chave = chave
novo.fe = NULO
novo.fd = NULO

atual = T.raiz
pai = NULO
enquanto atual não é NULO
      pai = atual
      se novo.chave < atual.chave
            atual = atual.fe
      senão
            atual = atual.fd
novo.pai = pai
se pai é NULO
      T.raiz = novo //a árvore estava vazia
senão
      se novo.chave < pai.chave
            pai.fe = novo
      senão
            pai.fd = novo
```



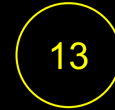


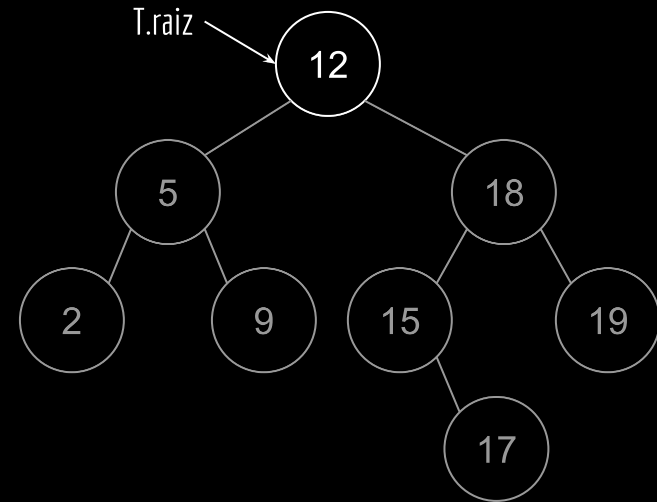
função **inserir**(T,13)

atual	pai	novo
12		13

Teste de mesa

```
função inserir(T,chave)

novo.chave = chave
novo.fe = NULO
novo.fd = NULO

atual = T.raiz
pai = NULO
enquanto atual não é NULO
      pai = atual
      se novo.chave < atual.chave
            atual = atual.fe
      senão
            atual = atual.fd
novo.pai = pai
se pai é NULO
      T.raiz = novo //a árvore estava vazia
senão
      se novo.chave < pai.chave
            pai.fe = novo
      senão
            pai.fd = novo
```



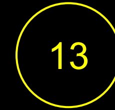


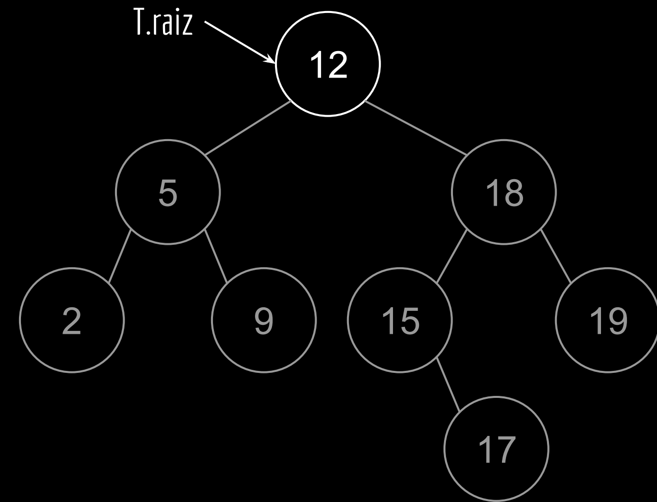
função **inserir**(T,13)

atual	pai	novo
12	NULO	13

Teste de mesa

```
função inserir(T,chave)

novo.chave = chave
novo.fe = NULO
novo.fd = NULO

atual = T.raiz
pai = NULO
enquanto atual não é NULO
      pai = atual
      se novo.chave < atual.chave
            atual = atual.fe
      senão
            atual = atual.fd
novo.pai = pai
se pai é NULO
      T.raiz = novo //a árvore estava vazia
senão
      se novo.chave < pai.chave
            pai.fe = novo
      senão
            pai.fd = novo
```



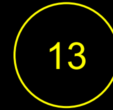


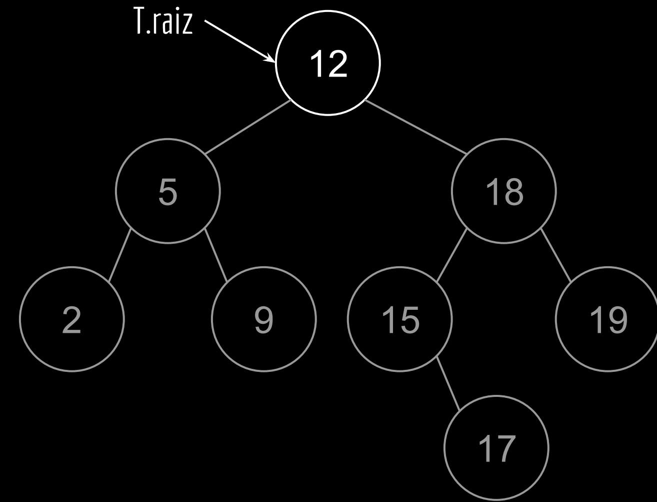
função **inserir**(T,13)

atual	pai	novo
12	NULO	13

Teste de mesa

```
função inserir(T,chave)

novo.chave = chave
novo.fe = NULO
novo.fd = NULO

atual = T.raiz
pai = NULO
enquanto atual não é NULO
      pai = atual
      se novo.chave < atual.chave
            atual = atual.fe
      senão
            atual = atual.fd
novo.pai = pai
se pai é NULO
      T.raiz = novo //a árvore estava vazia
senão
      se novo.chave < pai.chave
            pai.fe = novo
      senão
            pai.fd = novo
```



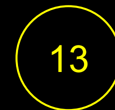


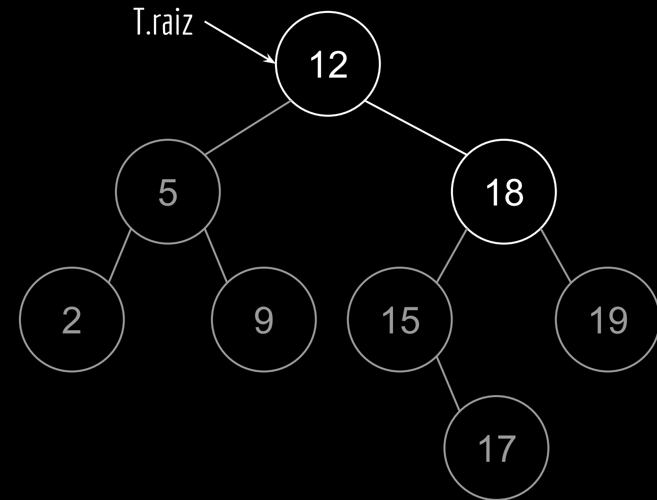
função **inserir**(T,13)

atual	pai	novo
12	12	13

Teste de mesa

```
função inserir(T,chave)

novo.chave = chave
novo.fe = NULO
novo.fd = NULO

atual = T.raiz
pai = NULO
enquanto atual não é NULO
      pai = atual
      se novo.chave < atual.chave
            atual = atual.fe
      senão
            atual = atual.fd
novo.pai = pai
se pai é NULO
      T.raiz = novo //a árvore estava vazia
senão
      se novo.chave < pai.chave
            pai.fe = novo
      senão
            pai.fd = novo
```



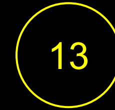


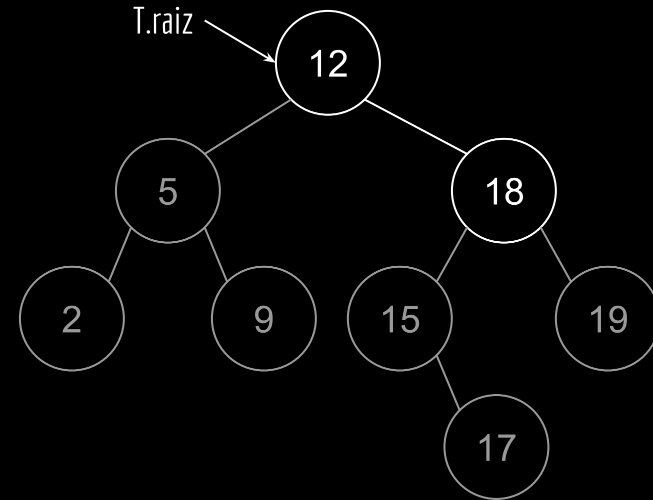
função **inserir**(T,13)

atual	pai	novo
18	12	13

Teste de mesa

```
função inserir(T,chave)

novo.chave = chave
novo.fe = NULO
novo.fd = NULO

atual = T.raiz
pai = NULO
enquanto atual não é NULO
      pai = atual
      se novo.chave < atual.chave
            atual = atual.fe
      senão
            atual = atual.fd
novo.pai = pai
se pai é NULO
      T.raiz = novo //a árvore estava vazia
senão
      se novo.chave < pai.chave
            pai.fe = novo
      senão
            pai.fd = novo
```



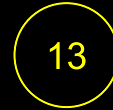


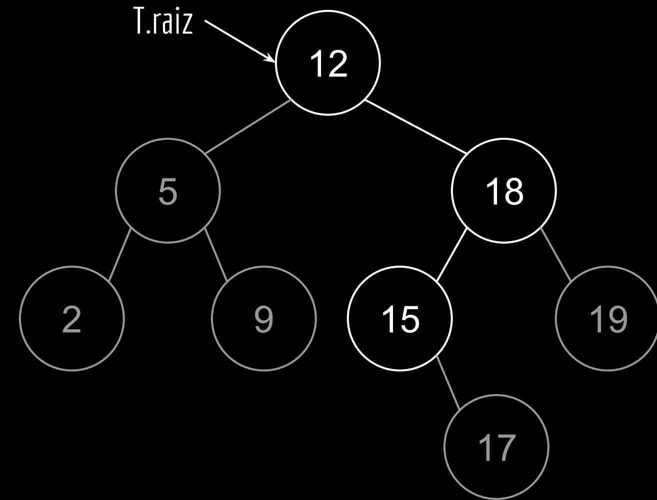
`função inserir(T,13)`

atual	pai	novo
18	12	13
	18	

Teste de mesa

```
função inserir(T,chave)

novo.chave = chave
novo.fe = NULO
novo.fd = NULO

atual = T.raiz
pai = NULO
enquanto atual não é NULO
      pai = atual
      se novo.chave < atual.chave
            atual = atual.fe
      senão
            atual = atual.fd
novo.pai = pai
se pai é NULO
      T.raiz = novo //a árvore estava vazia
senão
      se novo.chave < pai.chave
            pai.fe = novo
      senão
            pai.fd = novo
```



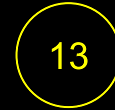


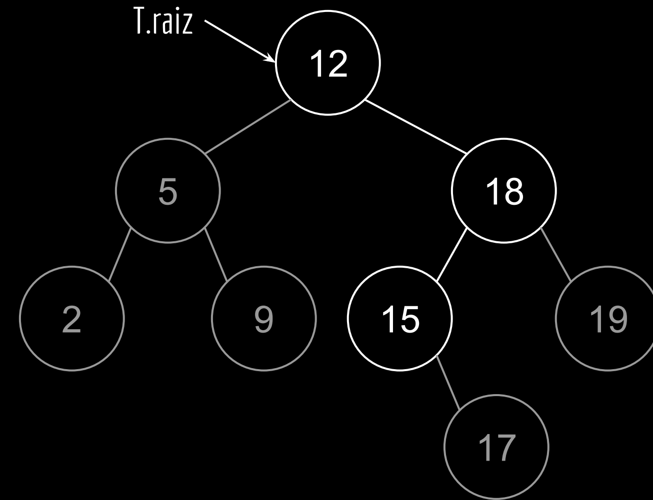

```
função inserir(T,13)
```

atual	pai	novo
18	12	13
15	18	

Teste de mesa

```
função inserir(T,chave)

novo.chave = chave
novo.fe = NULO
novo.fd = NULO

atual = T.raiz
pai = NULO
enquanto atual não é NULO
      pai = atual
      se novo.chave < atual.chave
            atual = atual.fe
      senão
            atual = atual.fd
novo.pai = pai
se pai é NULO
      T.raiz = novo //a árvore estava vazia
senão
      se novo.chave < pai.chave
            pai.fe = novo
      senão
            pai.fd = novo
```



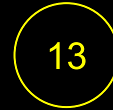


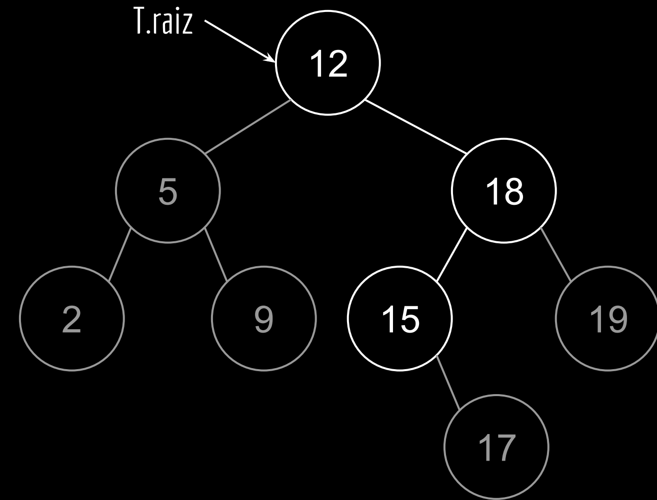
função **inserir**(T,13)

atual	pai	novo
18	12	13
15	18	
	15	

Teste de mesa

```
função inserir(T,chave)

novo.chave = chave
novo.fe = NULO
novo.fd = NULO

atual = T.raiz
pai = NULO
enquanto atual não é NULO
      pai = atual
      se novo.chave < atual.chave
            atual = atual.fe
      senão
            atual = atual.fd
novo.pai = pai
se pai é NULO
      T.raiz = novo //a árvore estava vazia
senão
      se novo.chave < pai.chave
            pai.fe = novo
      senão
            pai.fd = novo
```



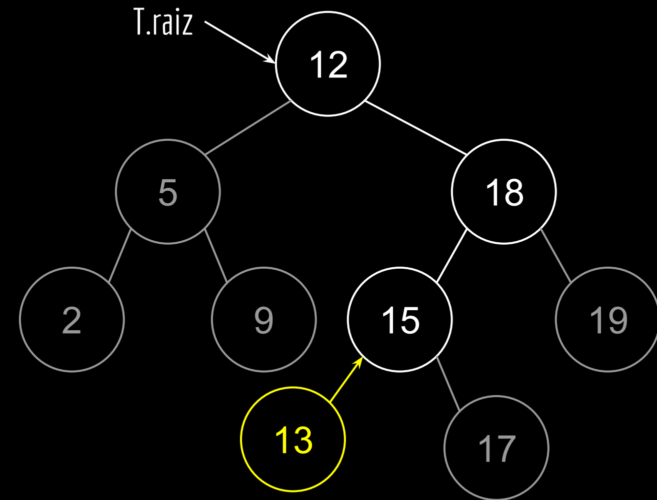


função **inserir**(T,13)

atual	pai	novo
18	12	13
15	18	
NULO	15	

Teste de mesa

```
função inserir(T,chave)

novo.chave = chave
novo.fe = NULO
novo.fd = NULO

atual = T.raiz
pai = NULO
enquanto atual não é NULO
      pai = atual
      se novo.chave < atual.chave
            atual = atual.fe
      senão
            atual = atual.fd
novo.pai = pai
se pai é NULO
      T.raiz = novo //a árvore estava vazia
senão
      se novo.chave < pai.chave
            pai.fe = novo
      senão
            pai.fd = novo
```

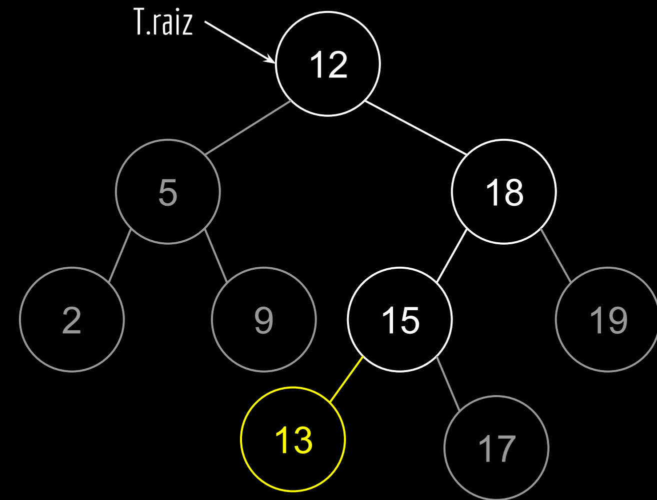


função **inserir**(T,13)

atual	pai	novo
18	12	13
15	18	
NULO	15	

Teste de mesa

```
função inserir(T,chave)

novo.chave = chave
novo.fe = NULO
novo.fd = NULO

atual = T.raiz
pai = NULO
enquanto atual não é NULO
      pai = atual
      se novo.chave < atual.chave
            atual = atual.fe
      senão
            atual = atual.fd
novo.pai = pai
se pai é NULO
      T.raiz = novo //a árvore estava vazia
senão
      se novo.chave < pai.chave
            pai.fe = novo
      senão
            pai.fd = novo
```



função **inserir**(T,13)

atual	pai	novo
18	12	13
15	18	
NULO	15	

Inserção

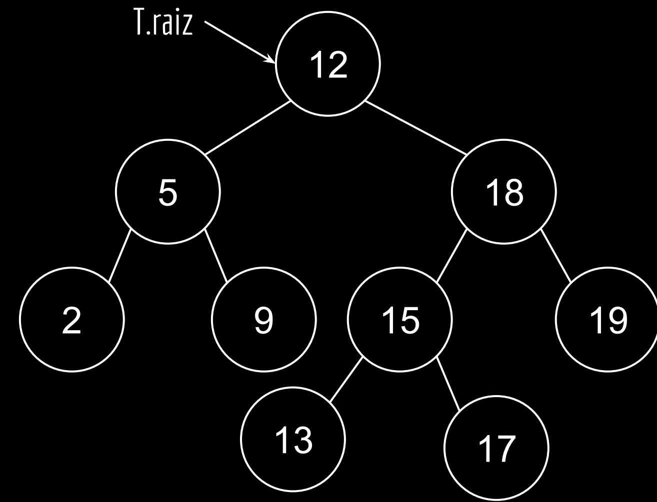
```
função inserir(T,chave)
entrada: árvore de busca binária T, e a chave a ser inserida.
saída: um novo nodo com a chave é inserido na posição correta na árvore.

novo.chave = chave
novo.fe = NULO
novo.fd = NULO

atual = T.raiz
pai = NULO
enquanto atual não é NULO
     pai = atual
     se novo.chave < atual.chave
          atual = atual.fe
     senão
          atual = atual.fd
novo.pai = pai
se pai é NULO
     T.raiz = novo //a árvore estava vazia
senão
     se novo.chave < pai.chave
          pai.fe = novo
     senão
          pai.fd = novo
```

Obs.: existem versões recursivas do algoritmo. Pesquise na literatura.

Buscar

Como buscar por uma chave na uma árvore de busca binária?

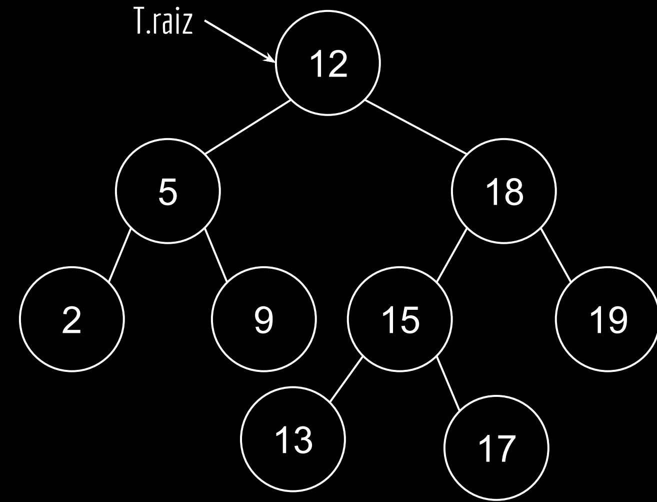
Por exemplo, buscar se a chave 14 existe na árvore?



Buscar

```
função buscar(r,chave)
entrada: nodo raiz da subárvore r, e a chave a ser buscada.
saída: o ponteiro para um nodo que contém a chave, ou NULO caso o nodo
não exista.

se r é NULO ou r.chave == chave
     retorne r

se chave < r.chave
     retorne buscar(r.fe,chave)
retorne buscar(r.fd,chave)
```

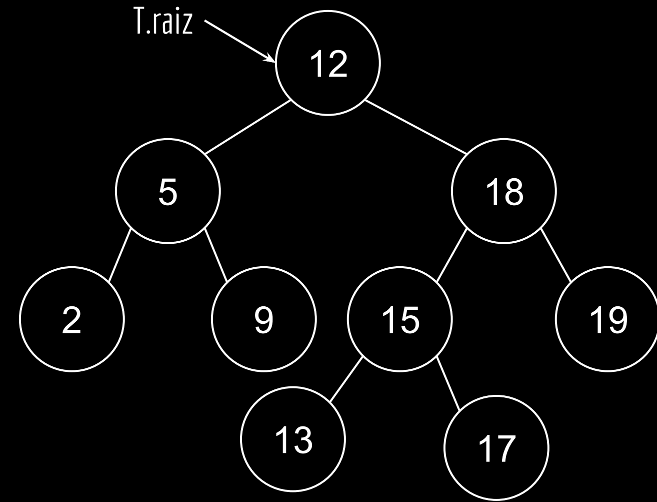
Teste de mesa



buscar(T.raiz ,14)

```
função buscar(r,chave)

se r é NULO ou r.chave == chave
     retorne r

se chave < r.chave
     retorne buscar(r.fe,chave)
retorne buscar(r.fd,chave)
```

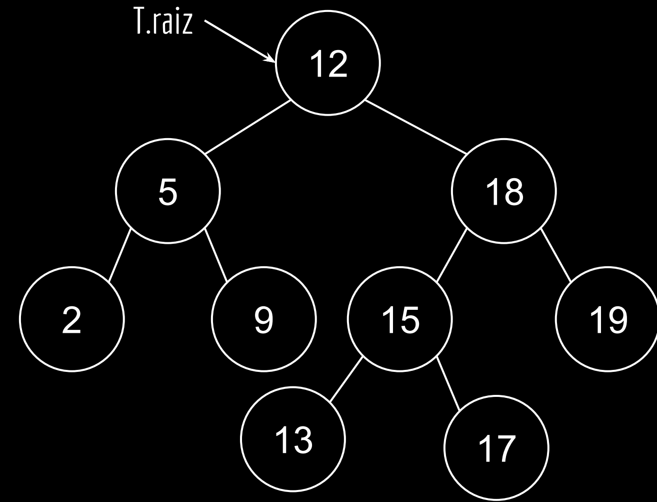
Teste de mesa



buscar(T.raiz ,14)
r
12

```
função buscar(r,chave)

se r é NULO ou r.chave == chave
     retorne r

se chave < r.chave
     retorne buscar(r.fe,chave)
retorne buscar(r.fd,chave)
```

Teste de mesa



buscar(T.raiz ,14)
r
12

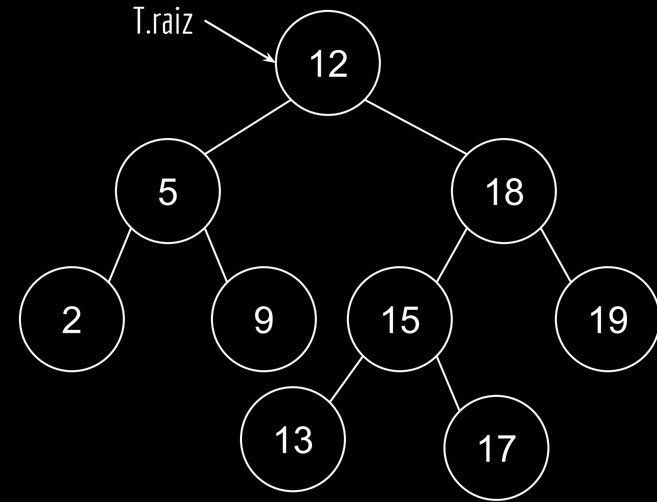
```
função buscar(r,chave)

se r é NULO ou r.chave == chave
     retorne r

se chave < r.chave
     retorne buscar(r.fe,chave)
retorne buscar(r.fd,chave)
```

Teste de mesa

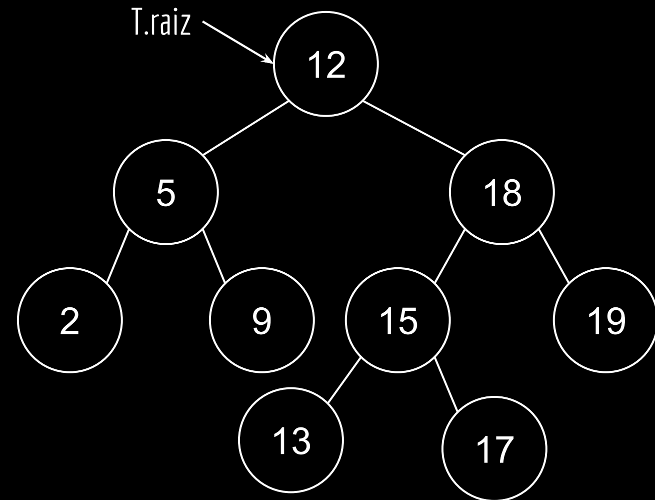
buscar(T.raiz ,14)
r
12

buscar(r.fd,14)
r
18

```
função buscar(r,chave)

se r é NULO ou r.chave == chave
     retorne r

se chave < r.chave
     retorne buscar(r.fe,chave)
retorne buscar(r.fd,chave)
```



Teste de mesa

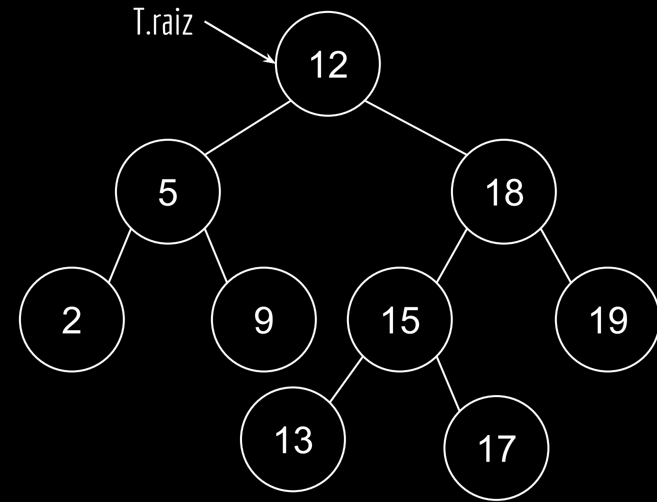


buscar(T.raiz ,14)
r
12

buscar(r.fd,14)
r
18

```
função buscar(r,chave)

se r é NULO ou r.chave == chave
     retorne r

se chave < r.chave
     retorne buscar(r.fe,chave)
retorne buscar(r.fd,chave)
```

Teste de mesa



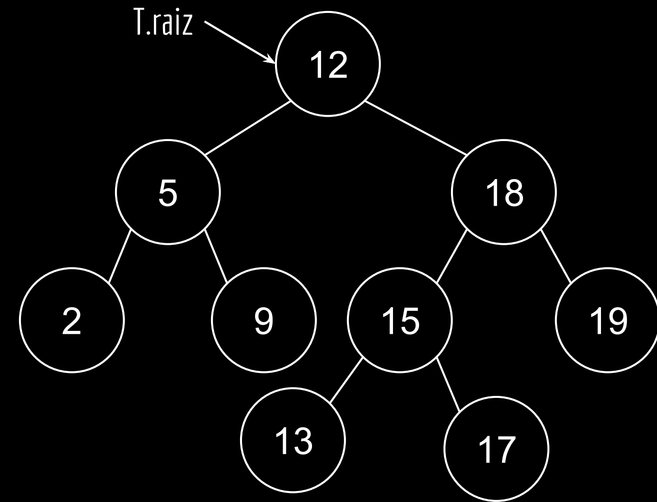
buscar(T.raiz ,14)
r
12

buscar(r.fd,14)
r
18

buscar(r.fe,14)
r
15

```
função buscar(r,chave)

se r é NULO ou r.chave == chave
     retorne r

se chave < r.chave
     retorne buscar(r.fe,chave)
retorne buscar(r.fd,chave)
```

Teste de mesa



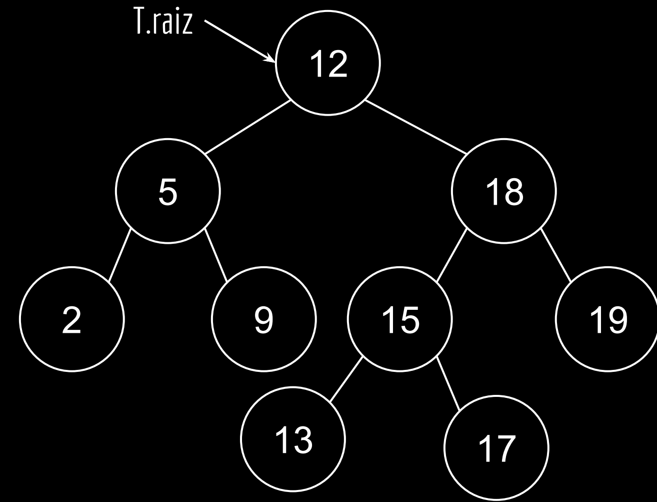
buscar(T.raiz ,14)
r
12

buscar(r.fd,14)
r
18

buscar(r.fe,14)
r
15

```
função buscar(r,chave)

se r é NULO ou r.chave == chave
     retorne r

se chave < r.chave
     retorne buscar(r.fe,chave)
retorne buscar(r.fd,chave)
```

Teste de mesa



buscar(T.raiz ,14)
r
12

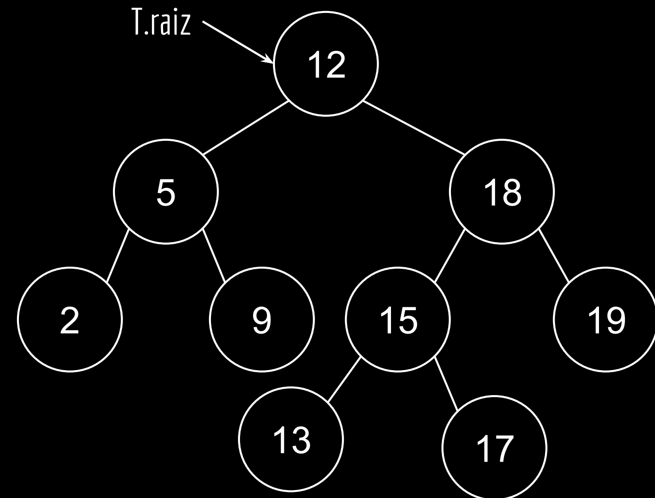
buscar(r.fd,14)
r
18

buscar(r.fe,14)
r
15

buscar(r.fe,14)
r
13

```
função buscar(r,chave)

se r é NULO ou r.chave == chave
     retorne r

se chave < r.chave
     retorne buscar(r.fe,chave)
retorne buscar(r.fd,chave)
```

Teste de mesa



buscar(T.raiz ,14)
r
12

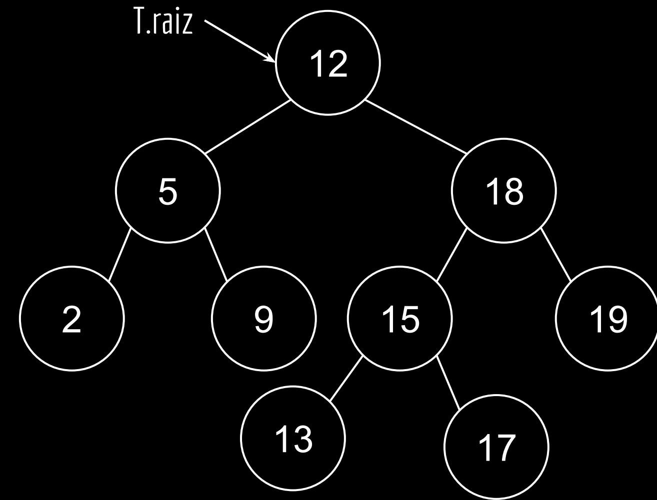
buscar(r.fd,14)
r
18

buscar(r.fe,14)
r
15

buscar(r.fe,14)
r
13

```
função buscar(r,chave)

se r é NULO ou r.chave == chave
     retorne r

se chave < r.chave
     retorne buscar(r.fe,chave)
retorne buscar(r.fd,chave)
```

Teste de mesa



buscar(T.raiz ,14)
r
12

buscar(r.fd,14)
r
18

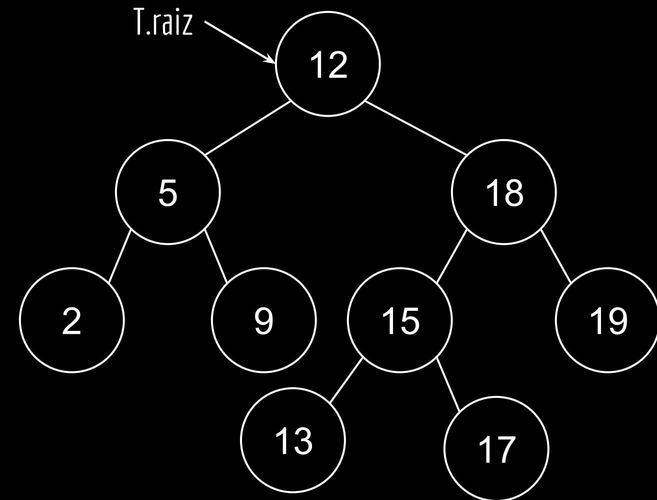
buscar(r.fe,14)
r
15

buscar(r.fe,14)
r
13

buscar(r.fe,14)
r
NULO

```
função buscar(r,chave)

se r é NULO ou r.chave == chave
    retorne r

se chave < r.chave
    retorne buscar(r.fe,chave)
retorne buscar(r.fd,chave)
```

Teste de mesa



resposta: NULO

buscar(T.raiz ,14)
r
12

buscar(r.fd,14)
r
18

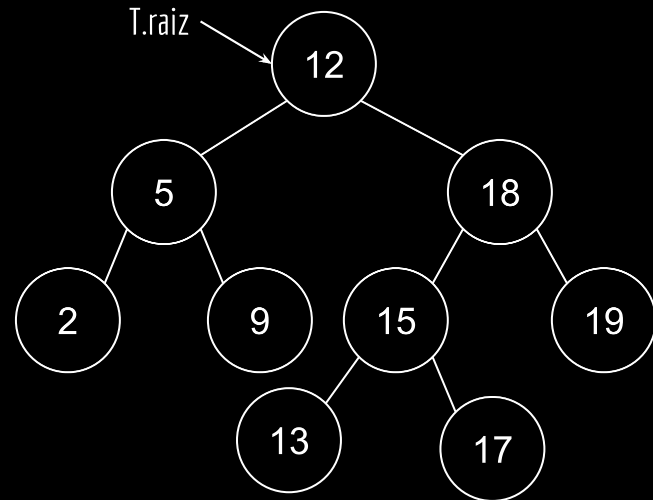
buscar(r.fe,14)
r
15

buscar(r.fe,14)
r
13

buscar(r.fe,14)
r
NULO

```
função buscar(r,chave)

se r é NULO ou r.chave == chave
     retorne r

se chave < r.chave
     retorne buscar(r.fe,chave)
retorne buscar(r.fd,chave)
```

Faça você mesmo



buscar(T.raiz ,2)

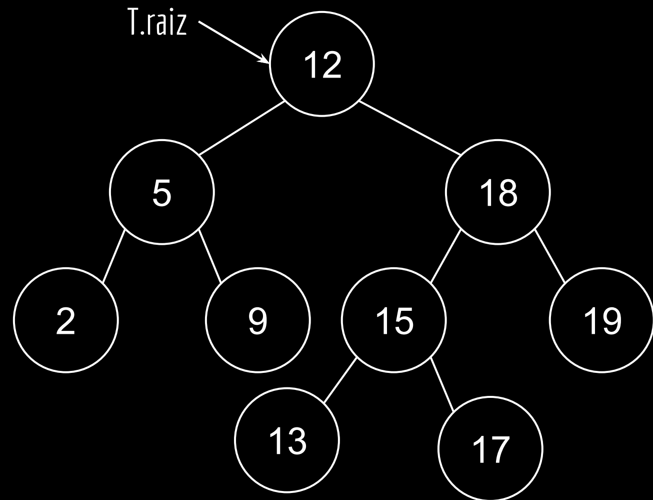
```
função buscar(r,chave)

se r é NULO ou r.chave == chave
     retorne r

se chave < r.chave
     retorne buscar(r.fe,chave)
retorne buscar(r.fd,chave)
```

Faça você mesmo

Note que a recursão no algoritmo é de calda.

Podemos transformar trivialmente em um loop.

Veja uma versão iterativa do algoritmo em Cormen et. al. 2012.




```
função buscar(r,chave)

se r é NULO ou r.chave == chave
     retorne r

se chave < r.chave
     retorne buscar(r.fe,chave)
retorne buscar(r.fd,chave)
```

Custo

Tanto a busca quanto a inserção (veremos adiante que remoção também):

1. Salta para o próximo nível da árvore a cada chamada recursiva (iteração do loop na inserção).
2. O custo do algoritmo depende da altura da árvore

Logo, as operações em uma árvore de busca binária são O(h), onde h é a altura.

Isso é bom? Ruim? Para uma árvore contendo n chaves, qual o custo?

```
função buscar(r,chave)

se r é NULO ou r.chave = chave
     retorne r

se chave < r.chave
     retorne buscar(r.fe,chave)
retorne buscar(r.fd,chave)
```

Custo

Em uma árvore completa, $h = \lfloor \log_2 n \rfloor$.

Logo, se a árvore está completa, ou quase , o custo é:

$O(\log_2 n)$, onde n é o número de nodos na árvore.

A árvore está **balanceada**.

Mas qual o pior caso para a altura?

Custo

Em uma árvore completa, $h = \lfloor \log_2 n \rfloor$.

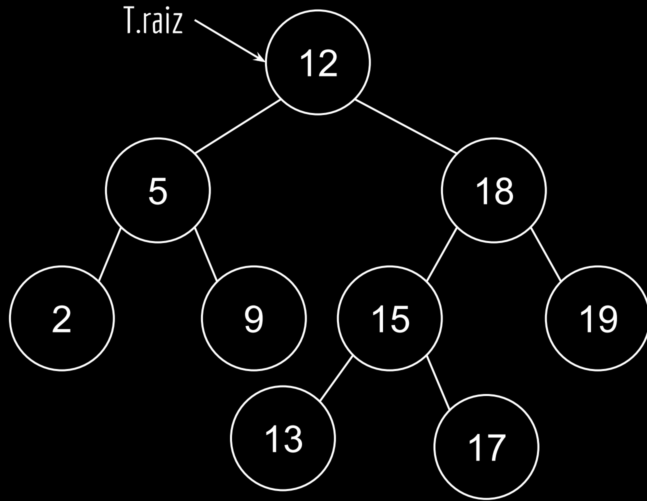
Logo, se a árvore está completa, ou quase , o custo é:

- $O(\log_2 n)$, onde n é o número de nodos na árvore.
- A árvore está **balanceada**.

Mas qual o pior caso para a altura?

- Se todos (ou a maioria) dos nodos se concentrarem “de um lado da árvore”, a altura será aproximadamente igual ao número de nodos.
- No pior caso, o curso das operações é $O(n)$.

Exemplo



T.raiz
2
5
9
12
13
15
17
18
19

Mínimo e Máximo

Para encontrar o menor valor, siga os filhos esquerdos até encontrar NULO.

Para encontrar o maior valor, siga os filhos direitos até encontrar NULO.

```
função minimo(r)

enquanto r.fe diferente de NULO
     r <- r.fe
retorne r
```

```
função maximo(r)

enquanto r.fd diferente de NULO
     r <- r.fd
retorne r
```

Mais algoritmos

Em Cormen et. al você encontrará os algoritmos para computar o sucessor e o antecessor do nodo.

Pesquise esses algoritmos e os implemente em C.

- Vai usar o atributo pai dos nodos.

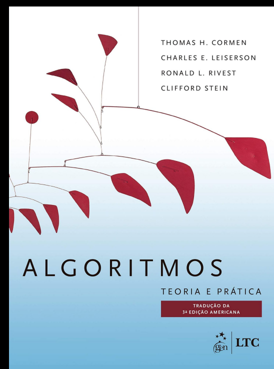
Exercícios

1. Suponha uma árvore de busca binária vazia. Faça o teste de mesa para inserir os seguintes itens (nesta ordem).
    a. Inserir 12
    b. Inserir 18
    c. Inserir 19
    d. Inserir 15
2. Implemente os algoritmos em C.
3. (Cormen et al.) Você está buscando pela chave 363 em uma árvore de busca binária que contém valores entre 1 e 1000. Qual das sequências a seguir não pode ser uma sequência de nodos examinada durante a busca?
    a. 2, 252, 401, 398, 330, 344, 397, 363.
    b. 924, 220, 911, 244, 898, 258, 362, 363.
    c. 925, 202, 911, 240, 912, 245, 363.
    d. 2, 399, 387, 219, 266, 382, 381, 278, 363.
    e. 935, 278, 347, 621, 299, 392, 358, 363.

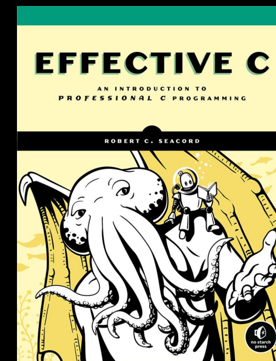
Referências

R. Sedgewick, K. Wayne. Algorithms Part I. 4a ed. 2011

T. Cormen, C. Leiserson, R. Rivest, C. Stein. Algoritmos: Teoria e Prática. 3a ed. 2012.



Seacord, R. C. Effective C: An ntroduction to Professional C Programming. 2020.



Licença

Esta obra está licenciada com uma Licença [Creative Commons Atribuição 4.0 Internacional.]

cc BY