Algoritmos e Estruturas de Dados I

Árvores Binárias de Busca

Prof. Tiago Eugenio de Melo
tmelo@uea.edu.br

www.tiagodemelo.info

Observações

- O conteúdo dessa aula é parcialmente proveniente do Capítulo 14 do livro “*Data Structure and Algorithms Using Python*”.
- As palavras com a fonte `Courier` indicam uma palavra-reservada da linguagem de programação.

Árvores Binárias de Busca

Introdução

Introdução

- Busca (pesquisa) de dados em texto é uma operação bastante comum e, por isso, bastante estudada.

Introdução

- Busca (pesquisa) de dados em texto é uma operação bastante comum e, por isso, bastante estudada.
- Uma pesquisa linear de um array ou uma lista de Python é bastante lenta.

Introdução

- Busca (pesquisa) de dados em texto é uma operação bastante comum e, por isso, bastante estudada.
- Uma pesquisa linear de um array ou uma lista de Python é bastante lenta.
  - Essa pesquisa pode ser melhorada com uma busca binária.

Introdução

- Busca (pesquisa) de dados em texto é uma operação bastante comum e, por isso, bastante estudada.
- Uma pesquisa linear de um array ou uma lista de Python é bastante lenta.
  - Essa pesquisa pode ser melhorada com uma busca binária.
  - Porém, arrays e listas de Python ainda têm dificuldade de realizar as operação de inserção e remoção de chaves.

Introdução

- Busca (pesquisa) de dados em texto é uma operação bastante comum e, por isso, bastante estudada.
- Uma pesquisa linear de um array ou uma lista de Python é bastante lenta.
  - Essa pesquisa pode ser melhorada com uma busca binária.
  - Porém, arrays e listas de Python ainda têm dificuldade de realizar as operação de inserção e remoção de chaves.
  - Exemplo: a remoção de uma chave obriga a reorganização da lista, pois a busca binária somente funciona em listas (arrays) ordenados.

Introdução

- Busca (pesquisa) de dados em texto é uma operação bastante comum e, por isso, bastante estudada.
- Uma pesquisa linear de um array ou uma lista de Python é bastante lenta.
  - Essa pesquisa pode ser melhorada com uma busca binária.
  - Porém, arrays e listas de Python ainda têm dificuldade de realizar as operação de inserção e remoção de chaves.
  - Exemplo: a remoção de uma chave obriga a reorganização da lista, pois a busca binária somente funciona em listas (arrays) ordenados.
- O objetivo primário de uma árvore de busca é fornecer uma eficiente operação de busca para rapidamente localizar um específico item na árvore.

Árvore Binária de Busca (ABB)

Árvore Binária de Busca (ABB)

- Uma árvore binária de busca (ABB) – binary search tree (BST) – é uma árvore binária em que cada nó contém um campo chave e a árvore é estruturada de tal maneira que para cada nó interior V:

Árvore Binária de Busca (ABB)

- Uma árvore binária de busca (ABB) – binary search tree (BST) – é uma árvore binária em que cada nó contém um campo chave e a árvore é estruturada de tal maneira que para cada nó interior V:
  - Todas as chaves menores que a chave do nó V devem ser armazenadas à esquerda de V.

Árvore Binária de Busca (ABB)

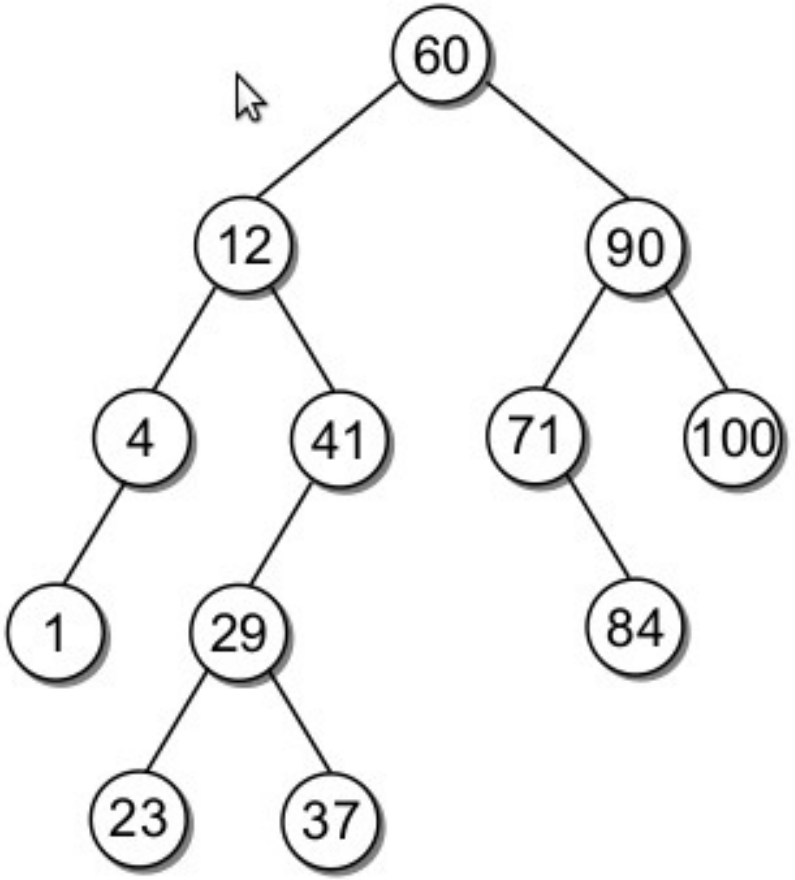
- Uma árvore binária de busca (ABB) – binary search tree (BST) – é uma árvore binária em que cada nó contém um campo chave e a árvore é estruturada de tal maneira que para cada nó interior V:
  - Todas as chaves menores que a chave do nó V devem ser armazenadas à esquerda de V.
  - Todas as chaves maiores que a chave do nó V devem ser armazenadas à direita de V.

Árvore Binária de Busca (ABB)

- Exemplo de ABB:

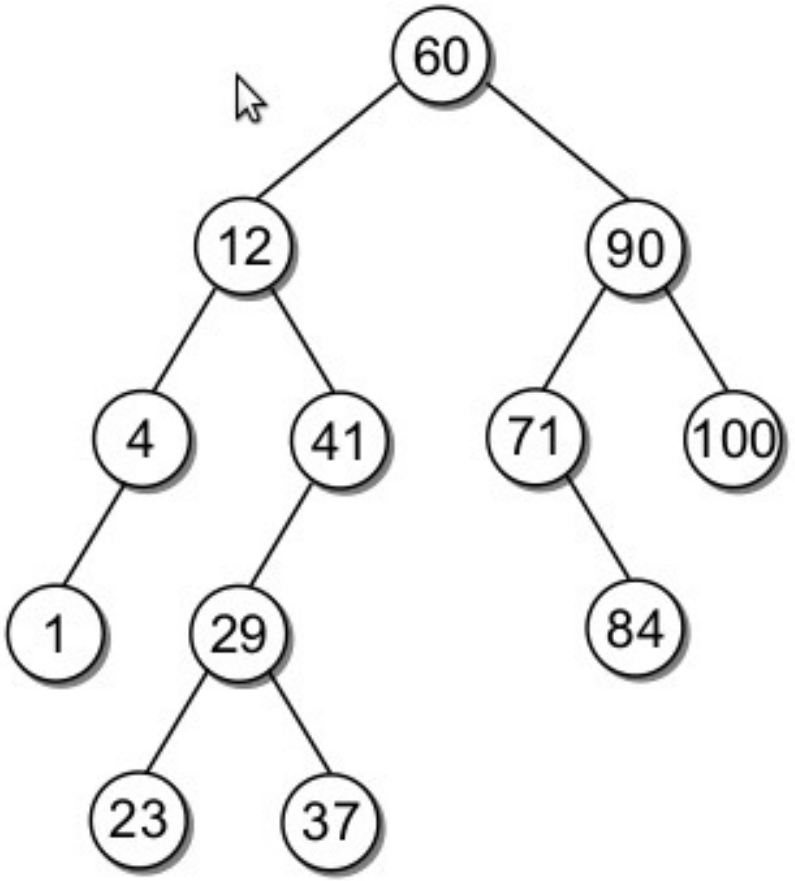
Árvore Binária de Busca (ABB)

- Exemplo de ABB:



Árvore Binária de Busca (ABB)

- Exemplo de ABB:



1 → 4 → 12 → 23 → 29 → 37 → 41 → 60 → 71 → 84 → 90 → 100

Árvore Binária de Busca (ABB)

Árvore Binária de Busca (ABB)

- Observações:

Árvore Binária de Busca (ABB)

- Observações:
  - A nossa definição de ABB impede o armazenamento de chaves duplicadas.

Árvore Binária de Busca (ABB)

- Observações:
  - A nossa definição de ABB impede o armazenamento de chaves duplicadas.
    - É uma suposição realista para vários problemas.

Árvore Binária de Busca (ABB)

- Observações:
  - A nossa definição de ABB impede o armazenamento de chaves duplicadas.
    - É uma suposição realista para vários problemas.
    - Isto torna a implementação mais simples.

Árvore Binária de Busca (ABB)

- Observações:
  - A nossa definição de ABB impede o armazenamento de chaves duplicadas.
    - É uma suposição realista para vários problemas.
    - Isto torna a implementação mais simples.
    - Essa restrição poderia ser alterada para permitir o armazenamento de chaves duplicadas.

Árvore Binária de Busca (ABB)

- Observações:
  - A nossa definição de ABB impede o armazenamento de chaves duplicadas.
    - É uma suposição realista para vários problemas.
    - Isto torna a implementação mais simples.
    - Essa restrição poderia ser alterada para permitir o armazenamento de chaves duplicadas.
    - Por questões didáticas, nós estamos apresentando apenas uma versão que considera chaves únicas.

TAD de ABB

TAD de ABB

```
1  class BSTMap :
2     # Creates an empty map instance.
3    def __init__( self ):
4      self._root = None
5      self._size = 0
6
7
8
9
10
11
12
13
```

TAD de ABB

```
1  class BSTMap :
2     # Creates an empty map instance.
3    def __init__( self ):
4      self._root = None
5      self._size = 0
6
7     # Returns the number of entries in the map.
8    def __len__( self ):
9      return self._size
10
11
12
13
```

TAD de ABB

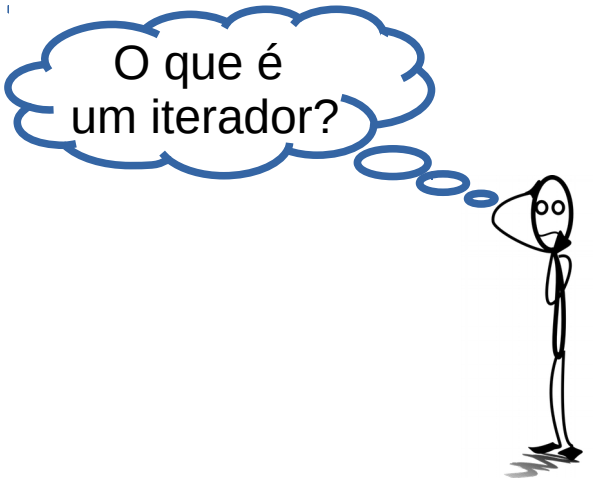
```
1  class BSTMap :
2     # Creates an empty map instance.
3    def __init__( self ):
4      self._root = None
5      self._size = 0
6
7     # Returns the number of entries in the map.
8    def __len__( self ):
9      return self._size
10
11    # Returns an iterator for traversing the keys in the map.
12   def __iter__( self ):
13     return _BSTMapIterator( self._root )
```

TAD de ABB

```
1  class BSTMap :
2     # Creates an empty map instance.
3    def __init__( self ):
4      self._root = None
5      self._size = 0
6
7     # Returns the number of entries in the map.
8    def __len__( self ):
9      return self._size
10
11    # Returns an iterator for traversing the keys in the map.
12   def __iter__( self ):
13     return _BSTMapIterator( self._root )
```

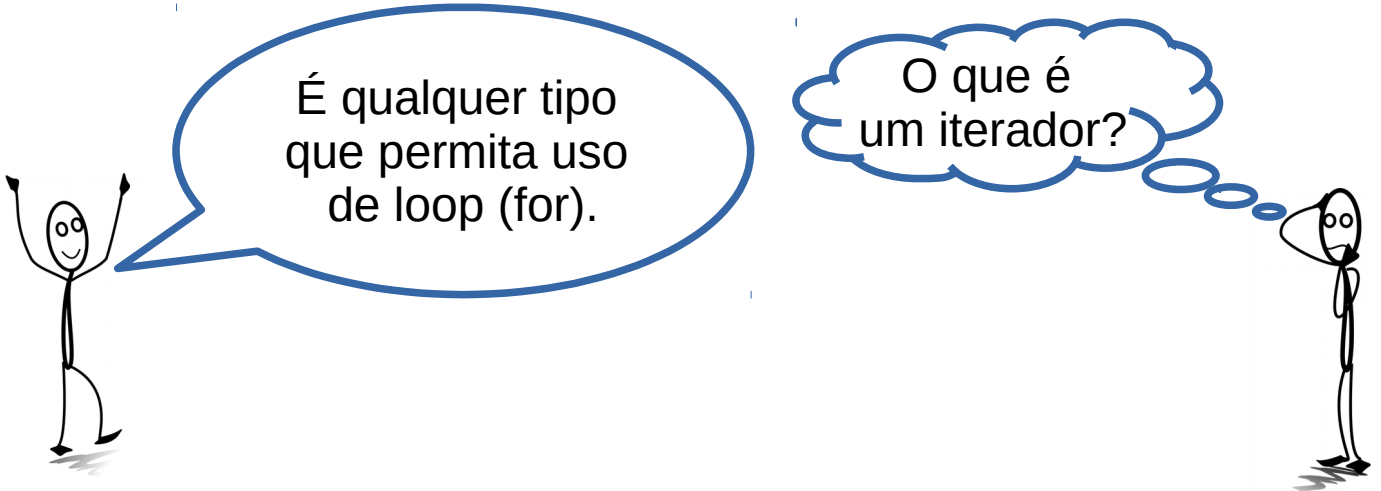
TAD de ABB

```
1  class BSTMap :
2     # Creates an empty map instance.
3    def __init__( self ):
4      self._root = None
5      self._size = 0
6
7     # Returns the number of entries in the map.
8    def __len__( self ):
9      return self._size
10
11    # Returns an iterator for traversing the keys in the map.
12   def __iter__( self ):
13     return _BSTMapIterator( self._root )
```



TAD de ABB

```
1  class BSTMap :
2     # Creates an empty map instance.
3    def __init__( self ):
4      self._root = None
5      self._size = 0
6
7     # Returns the number of entries in the map.
8    def __len__( self ):
9      return self._size
10
11    # Returns an iterator for traversing the keys in the map.
12   def __iter__( self ):
13     return _BSTMapIterator( self._root )
```



TAD de ABB

TAD de ABB

```
15  # Storage class for the binary search tree nodes of the map.
16  class _BSTMapNode :
17    def __init__( self, key, value ):
18      self.key = key
19      self.value = value
20      self.left = None
21      self.right = None
```

Operação de Busca

Operação de Busca

- Dada uma árvore binária, alguém sempre vai querer buscar na árvore para determinar se ela contém uma determinada chave (ou localizar um elemento específico).

Operação de Busca

- Dada uma árvore binária, alguém sempre vai querer buscar na árvore para determinar se ela contém uma determinada chave (ou localizar um elemento específico).
- Lembre-se:

Operação de Busca

- Dada uma árvore binária, alguém sempre vai querer buscar na árvore para determinar se ela contém uma determinada chave (ou localizar um elemento específico).
- Lembre-se:
  - Existe um único caminho da raiz até algum dos nós da árvore.

Operação de Busca

- Dada uma árvore binária, alguém sempre vai querer buscar na árvore para determinar se ela contém uma determinada chave (ou localizar um elemento específico).
- Lembre-se:
  - Existe um único caminho da raiz até algum dos nós da árvore.
- A questão é:

Operação de Busca

- Dada uma árvore binária, alguém sempre vai querer buscar na árvore para determinar se ela contém uma determinada chave (ou localizar um elemento específico).
- Lembre-se:
  - Existe um único caminho da raiz até algum dos nós da árvore.
- A questão é:
  - Como nós sabemos qual caminho seguir?

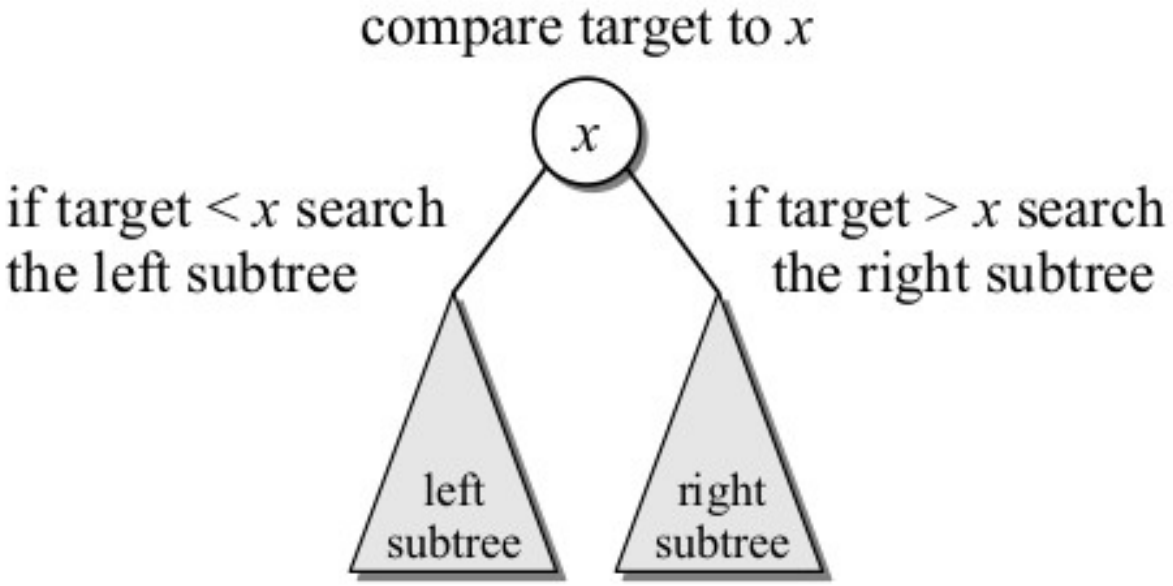
Operação de Busca

Operação de Busca

- O início da busca sempre se iniciará pela raiz.

Operação de Busca

- O início da busca <u>sempre</u> se iniciará pela raiz.



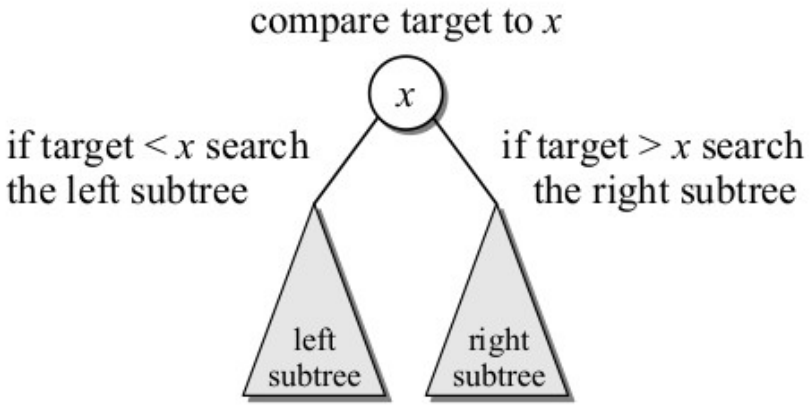
Operação de Busca

Operação de Busca

- O elemento procurado é comparado com o nó da raiz.

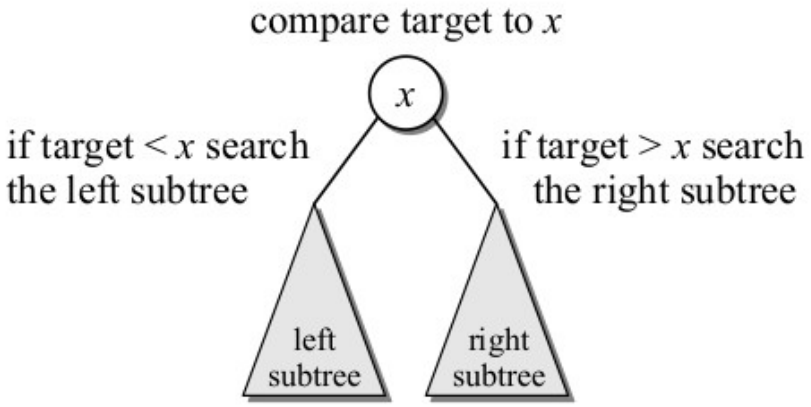
Operação de Busca

- O elemento procurado é comparado com o nó da raiz.



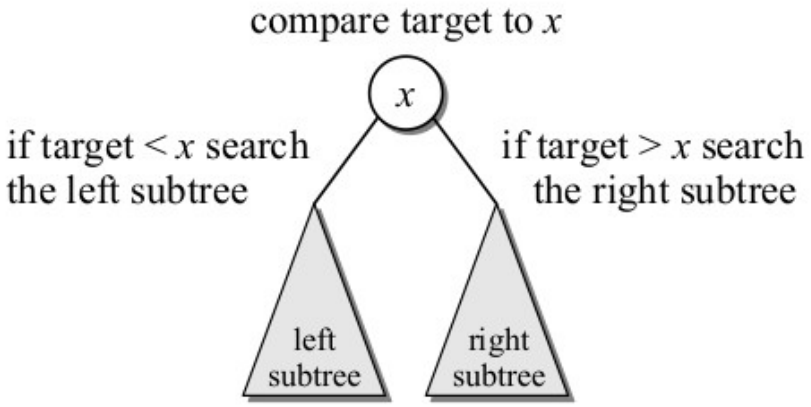
Operação de Busca

- O elemento procurado é comparado com o nó da raiz.
  - Se ele for igual, então encontramos.



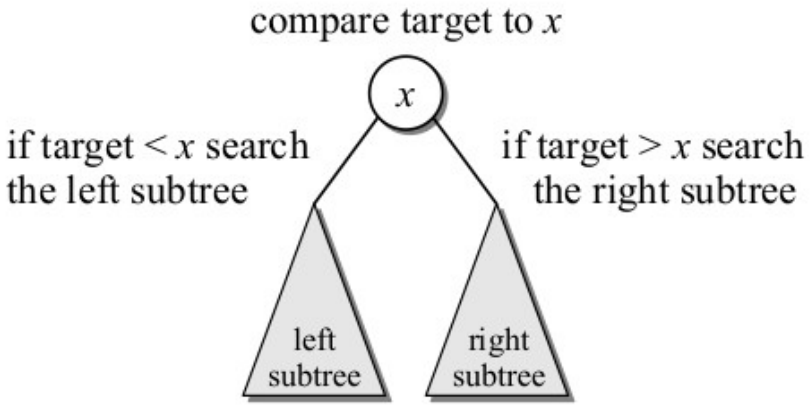
Operação de Busca

- O elemento procurado é comparado com o nó da raiz.
  - Se ele for igual, então encontramos.
  - Se ele for maior, então estará à direita da raiz.



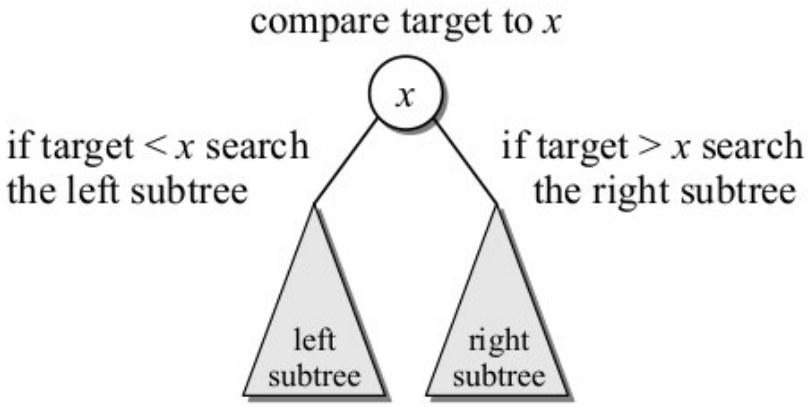
Operação de Busca

- O elemento procurado é comparado com o nó da raiz.
  - Se ele for igual, então encontramos.
  - Se ele for maior, então estará à direita da raiz.
  - Se ele for menor, então estará à esquerda da raiz.



Operação de Busca

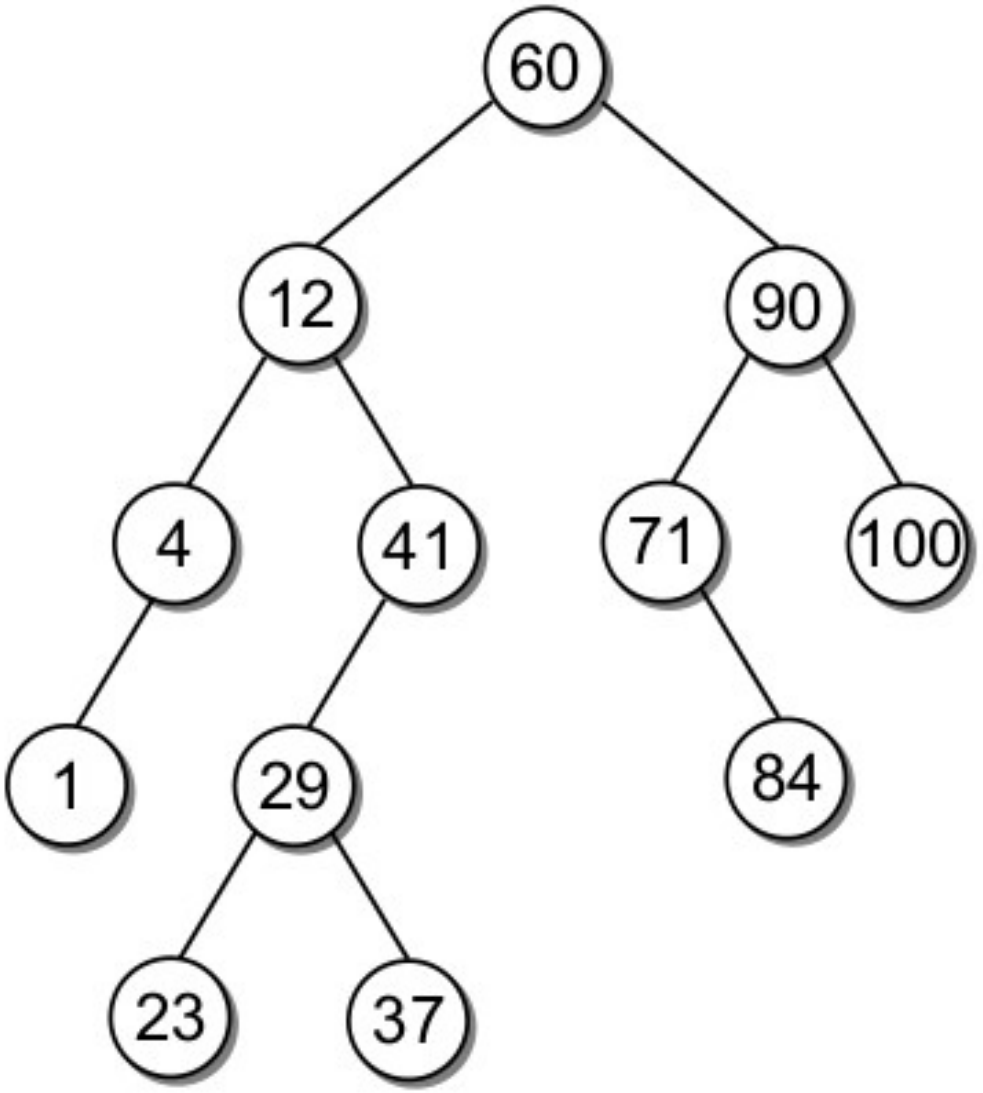
- O elemento procurado é comparado com o nó da raiz.
  - Se ele for igual, então encontramos.
  - Se ele for maior, então estará à direita da raiz.
  - Se ele for menor, então estará à esquerda da raiz.
  - Processo é repetido até localizarmos o alvo ou encontrarmos um nó filho nulo.



Operação de Busca

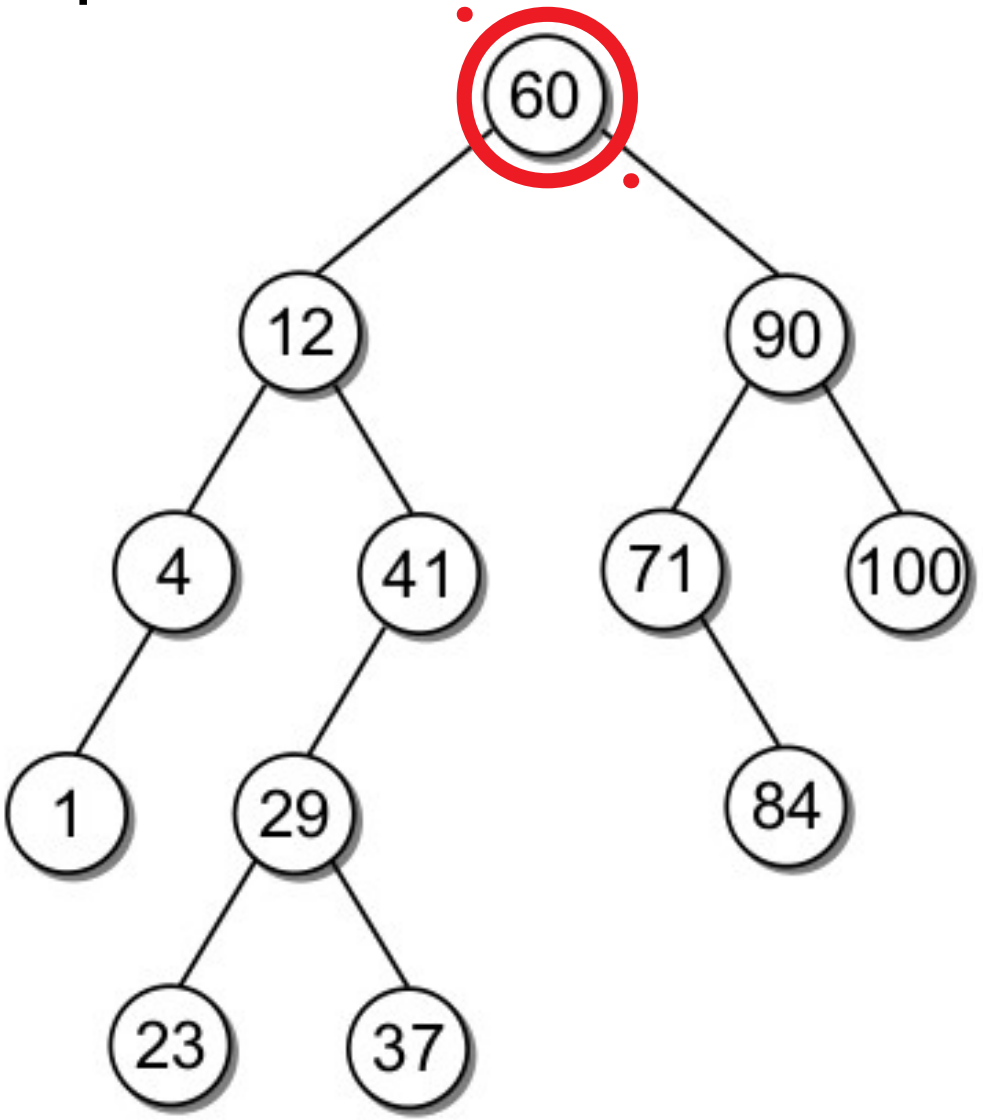
Operação de Busca

- Suponha que vamos buscar o elemento 29:



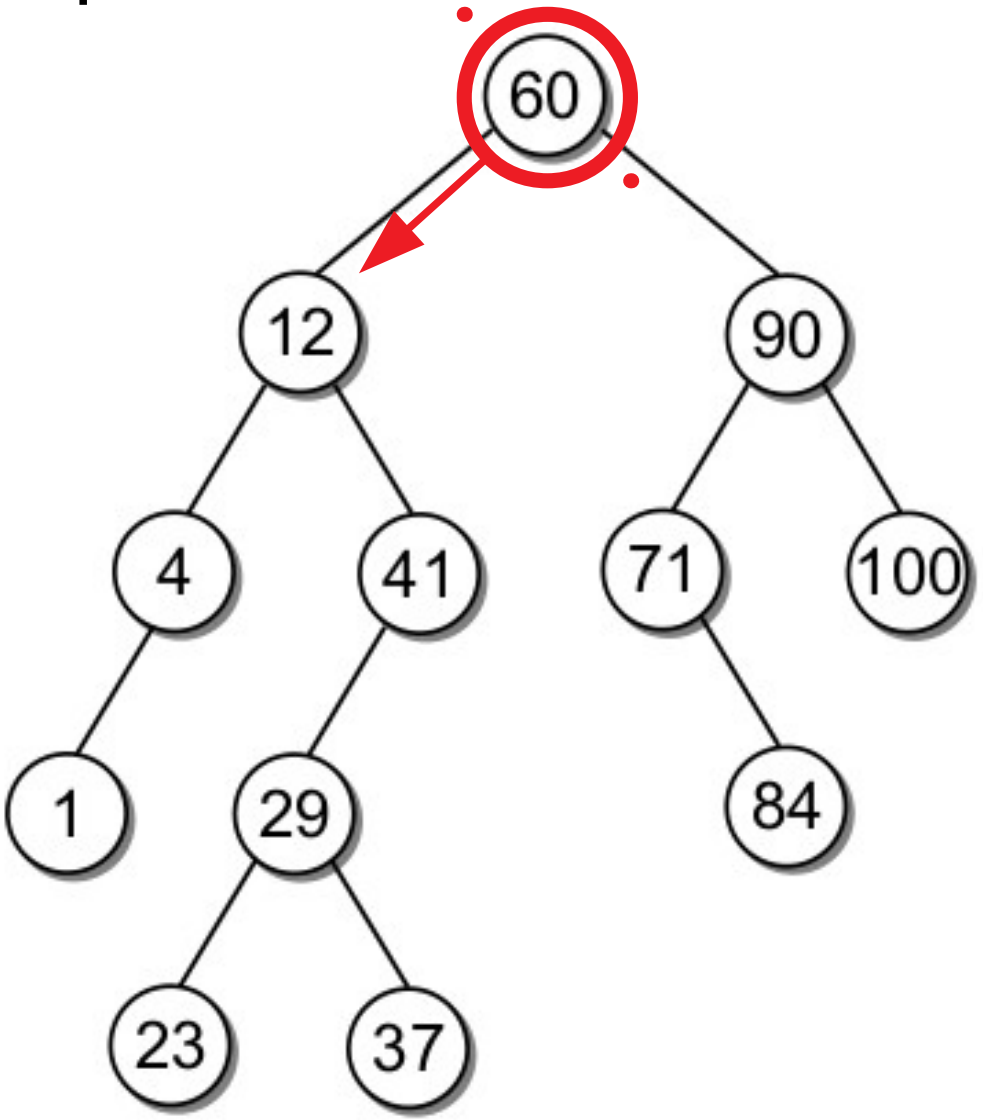
Operação de Busca

- Suponha que vamos buscar o elemento 29:



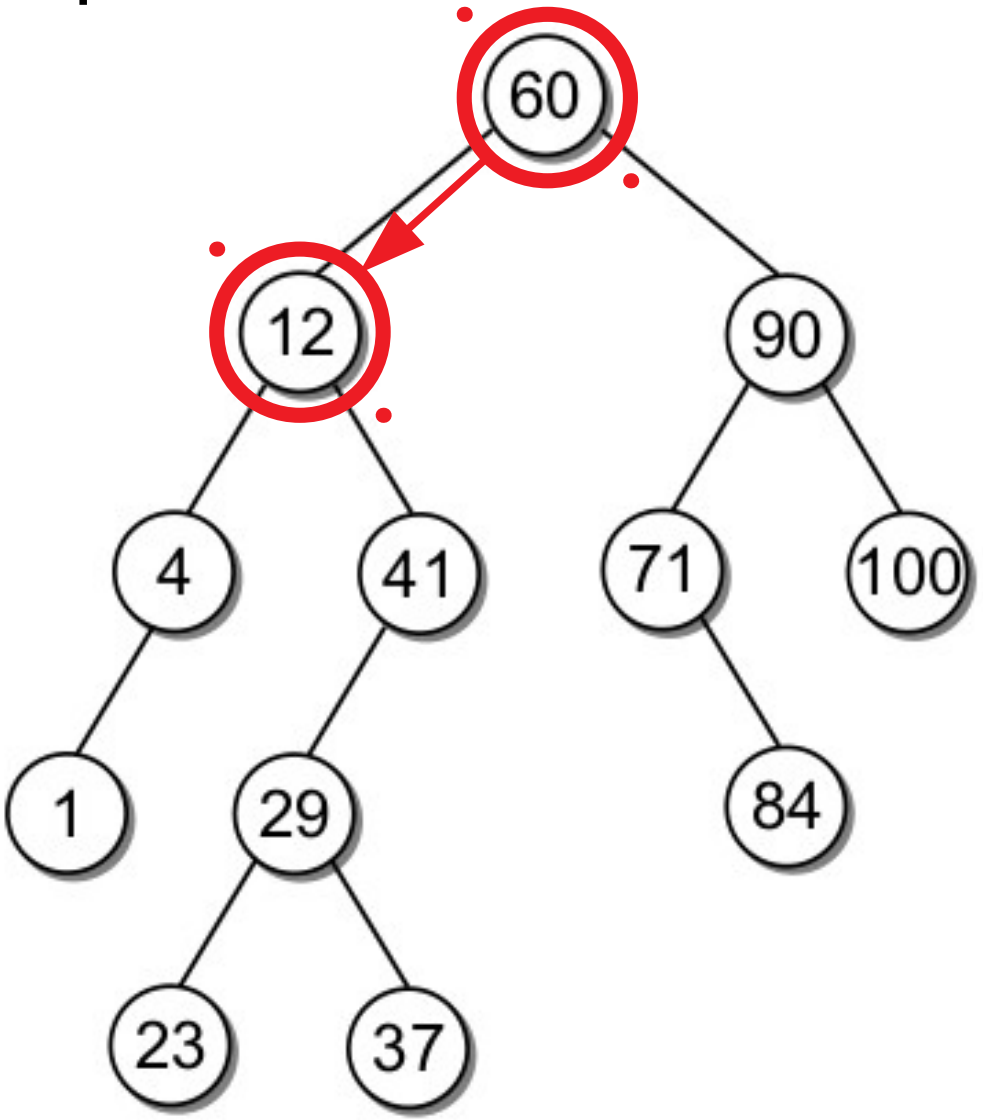
Operação de Busca

- Suponha que vamos buscar o elemento 29:



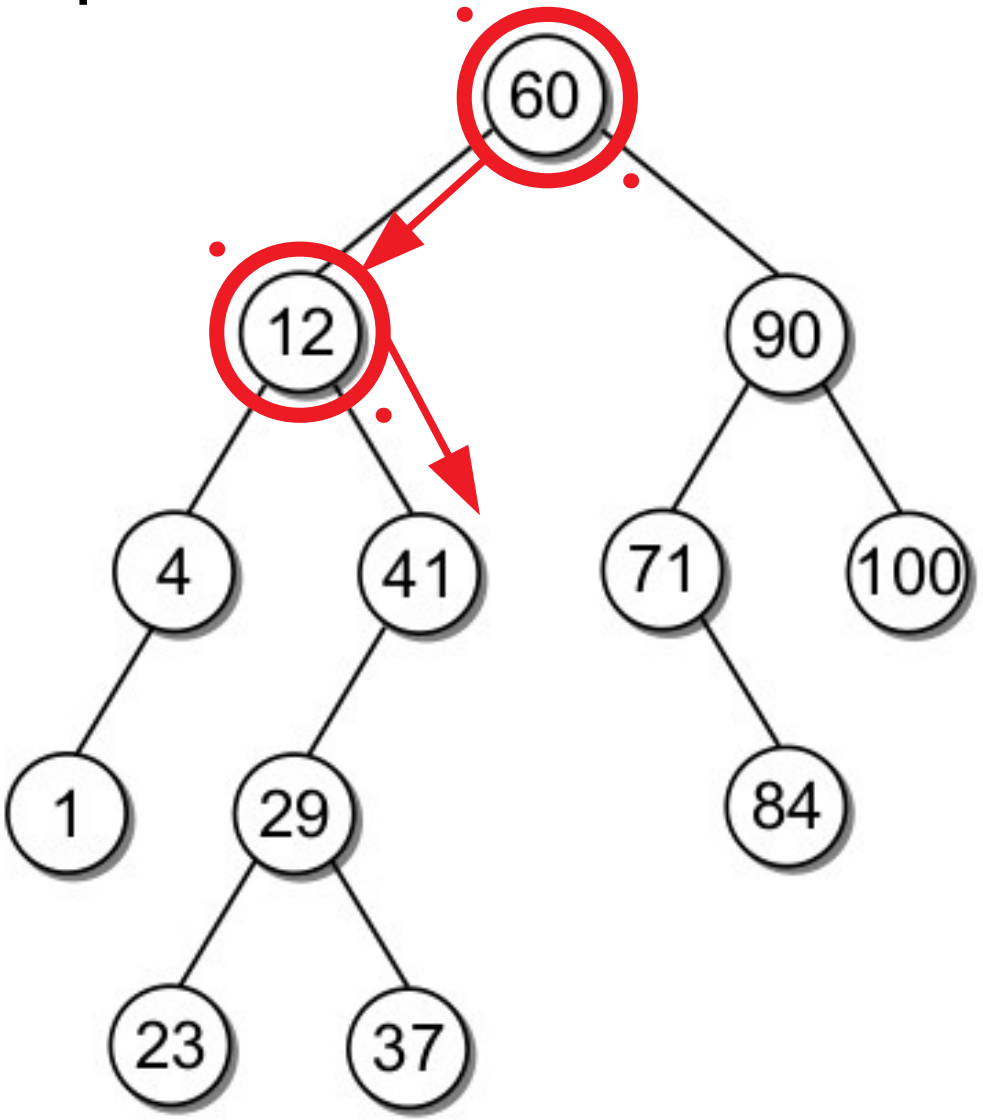
Operação de Busca

- Suponha que vamos buscar o elemento 29:



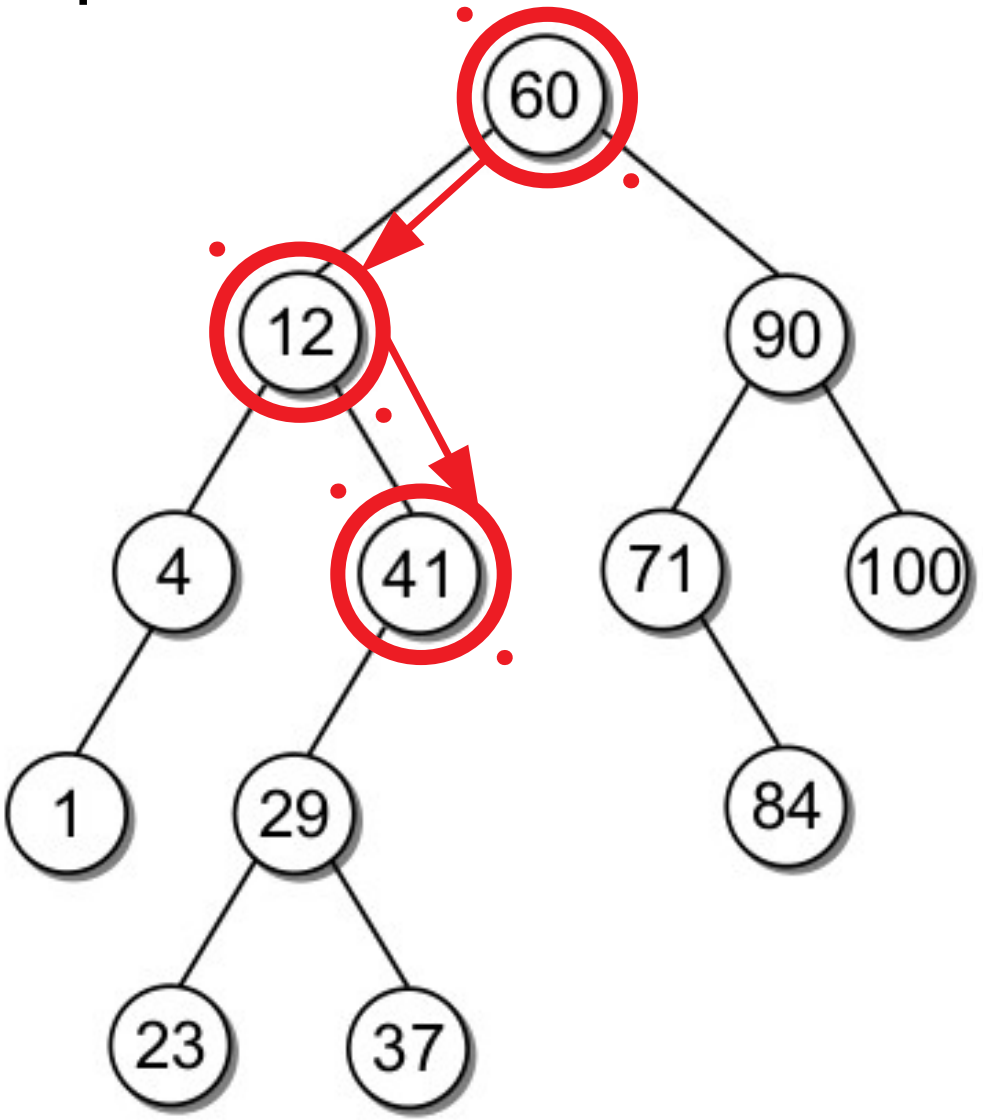
Operação de Busca

- Suponha que vamos buscar o elemento 29:



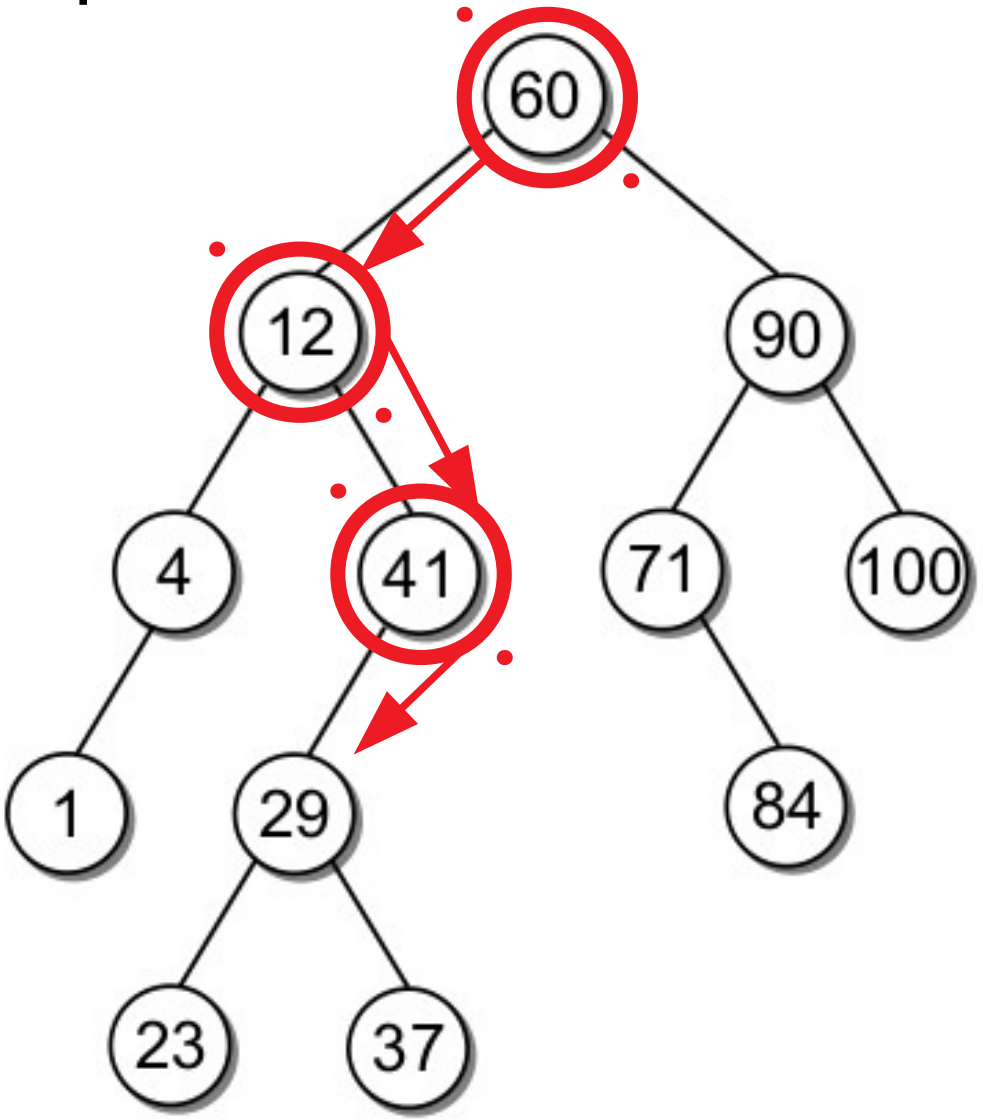
Operação de Busca

- Suponha que vamos buscar o elemento 29:



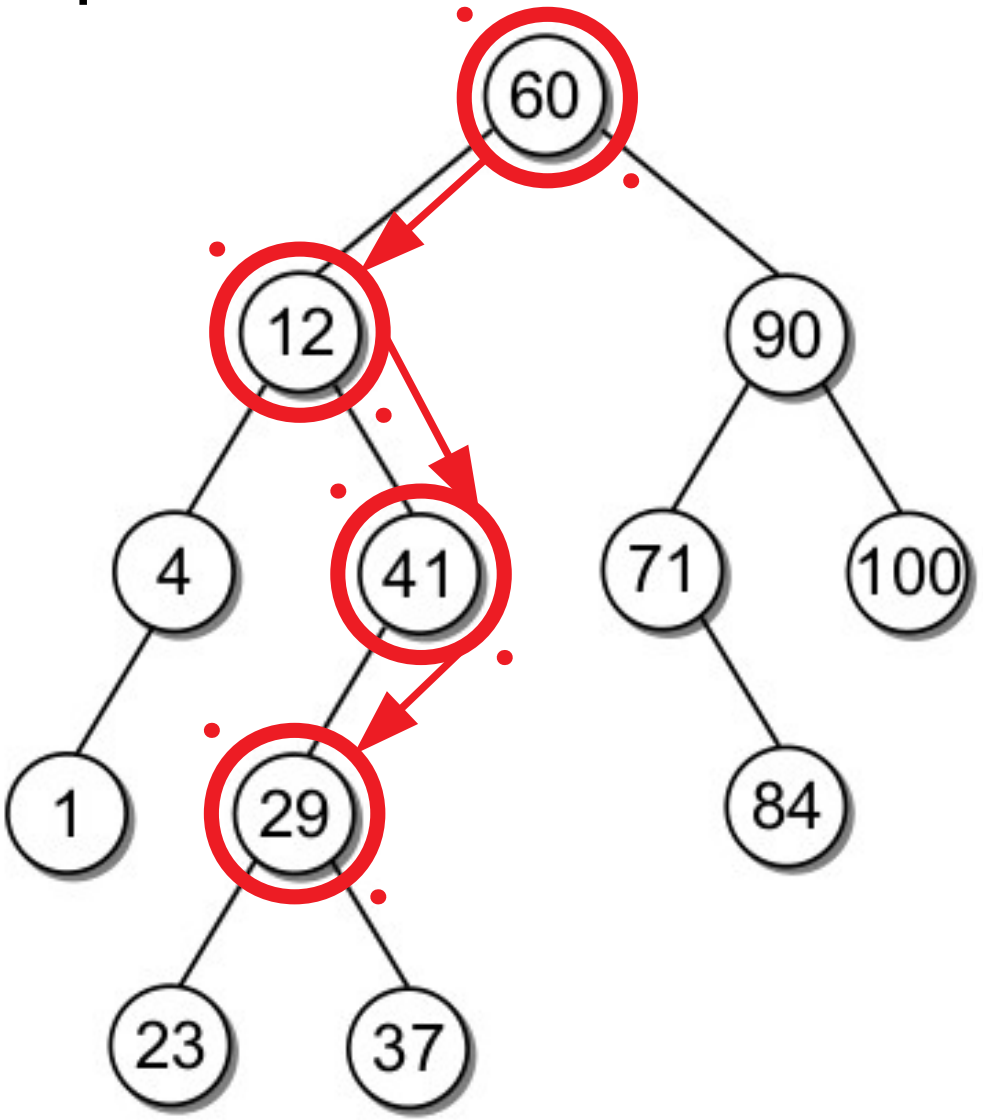
Operação de Busca

- Suponha que vamos buscar o elemento 29:



Operação de Busca

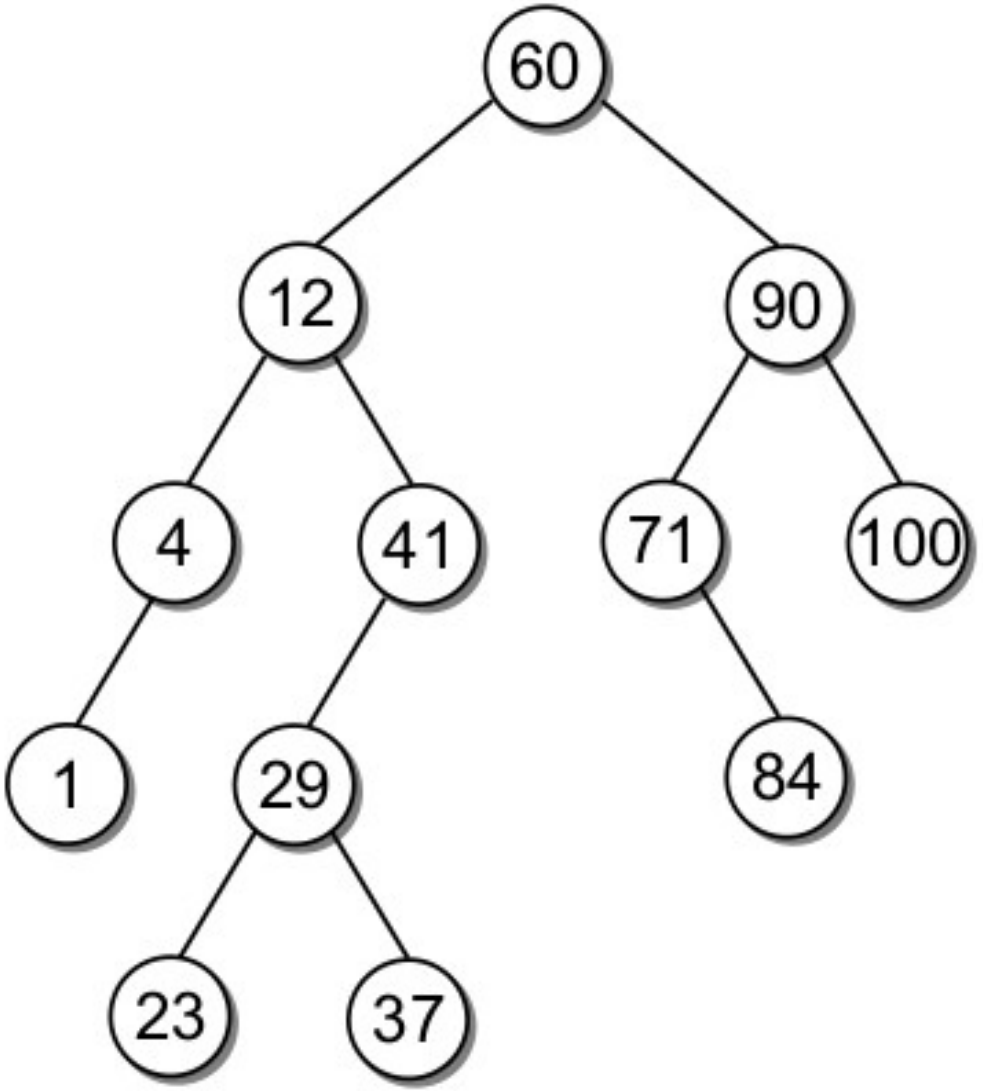
- Suponha que vamos buscar o elemento 29:



Operação de Busca

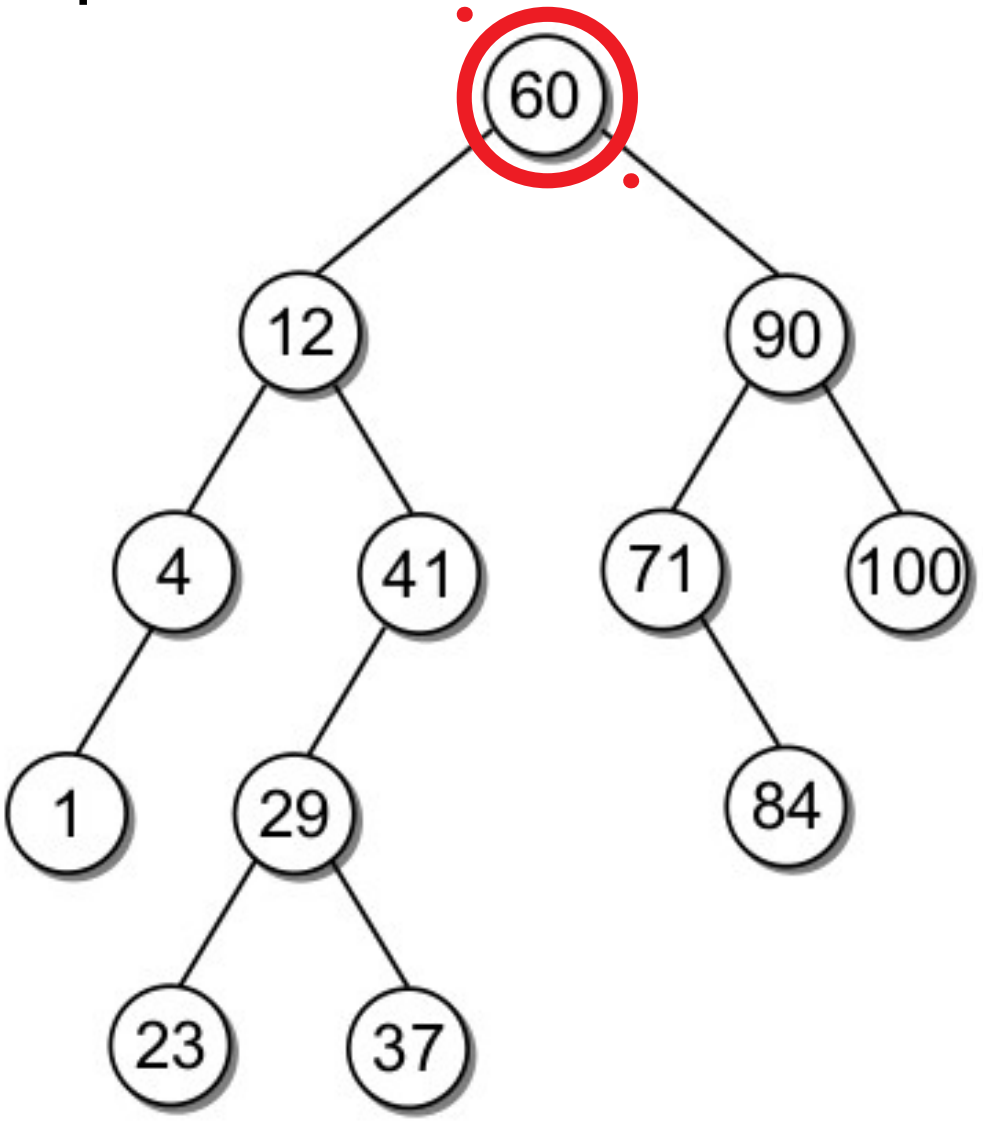
Operação de Busca

- Suponha que vamos buscar o elemento 68:



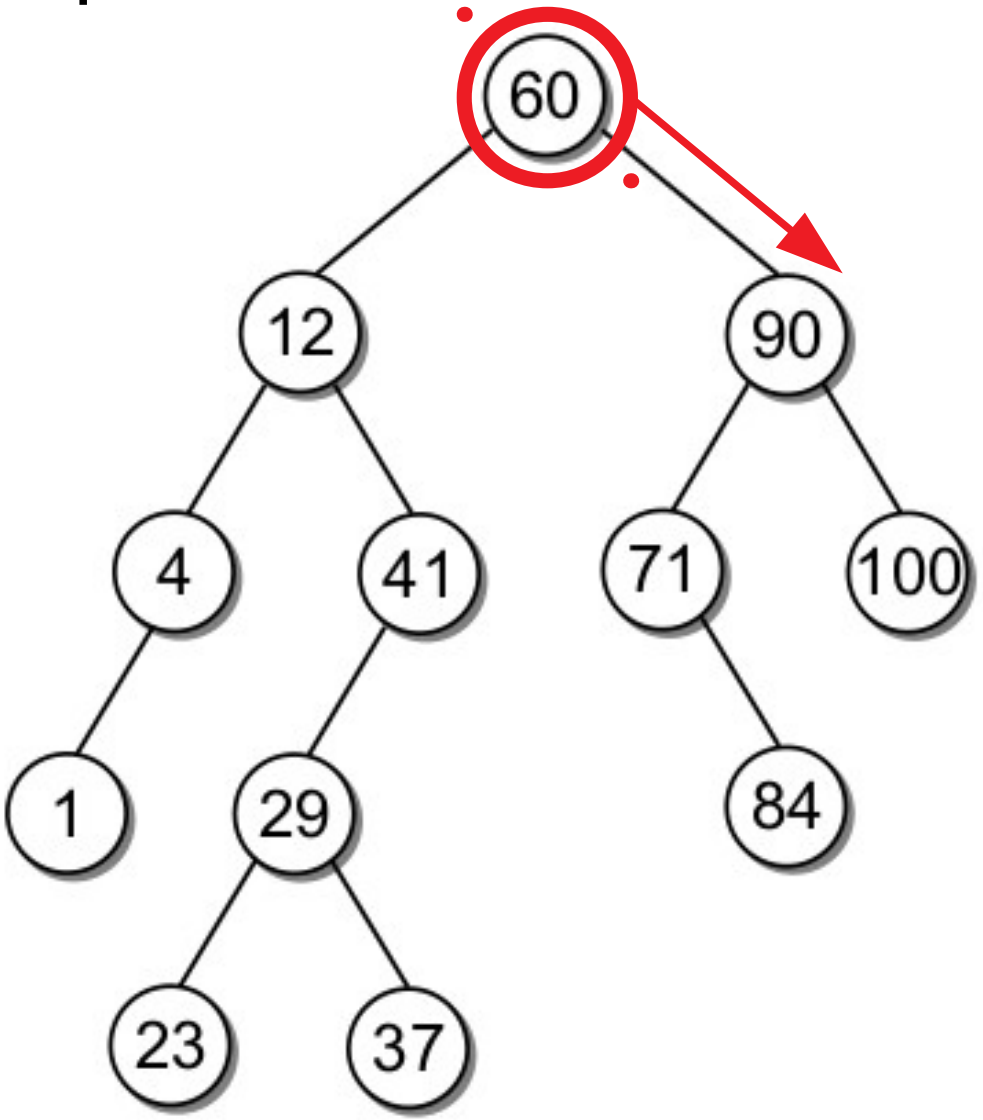
Operação de Busca

- Suponha que vamos buscar o elemento 68:



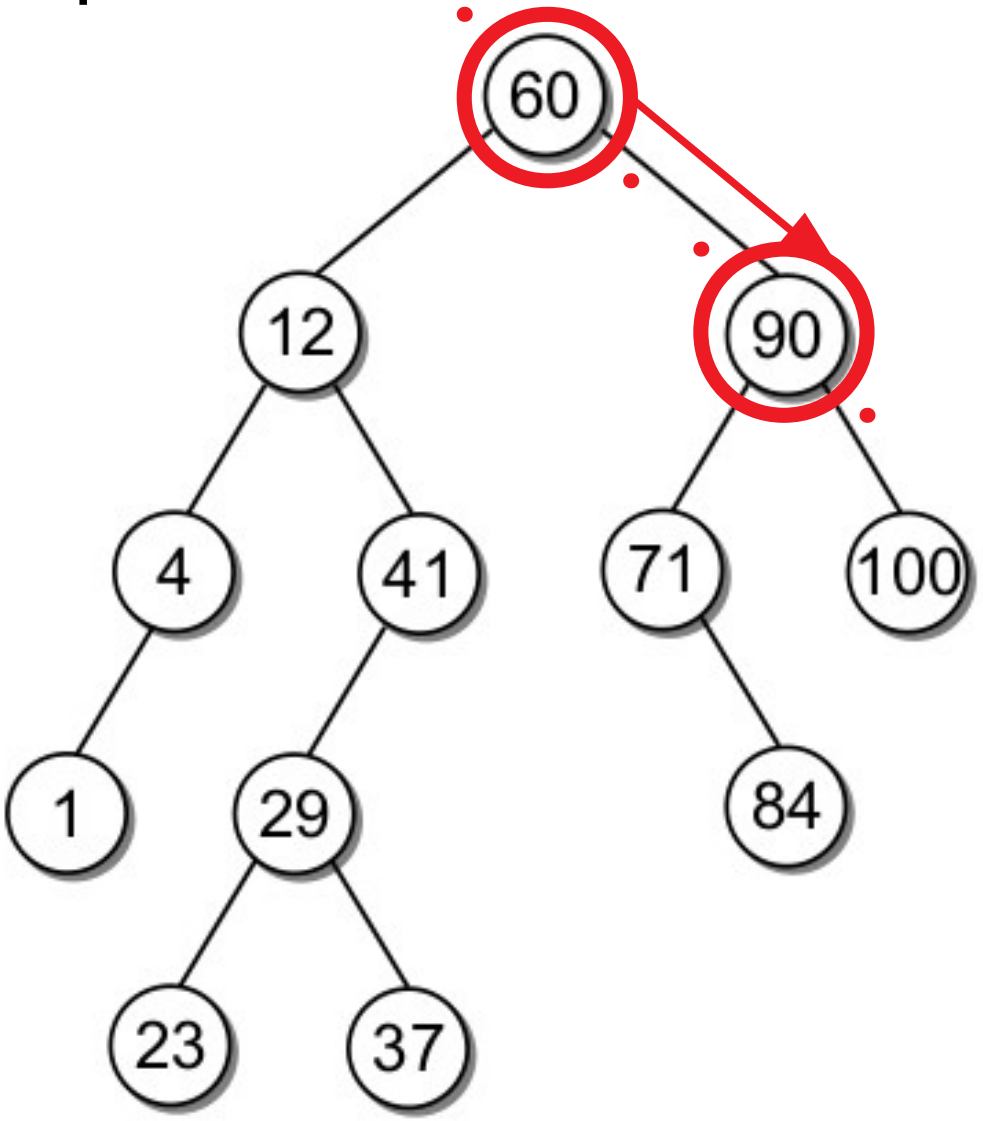
Operação de Busca

- Suponha que vamos buscar o elemento 68:



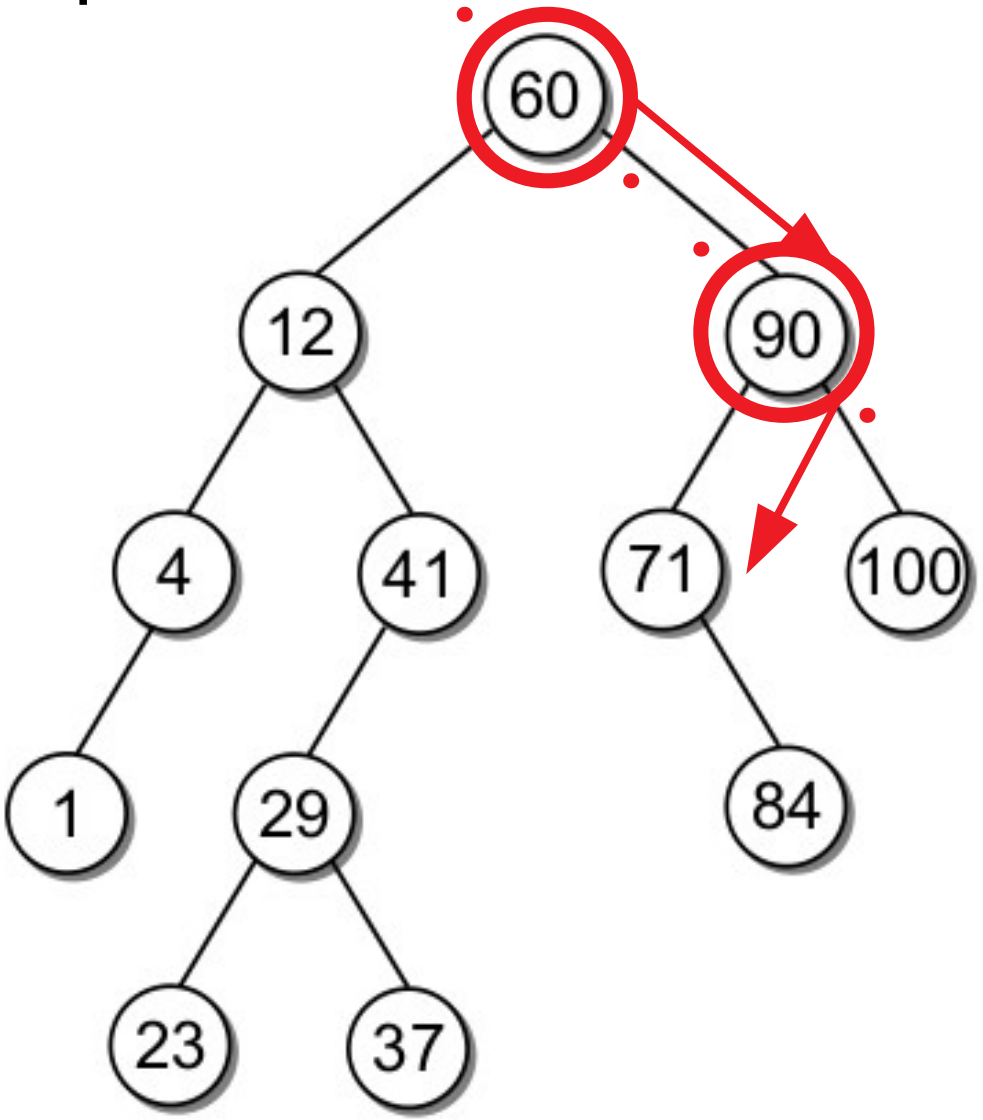
Operação de Busca

- Suponha que vamos buscar o elemento 68:



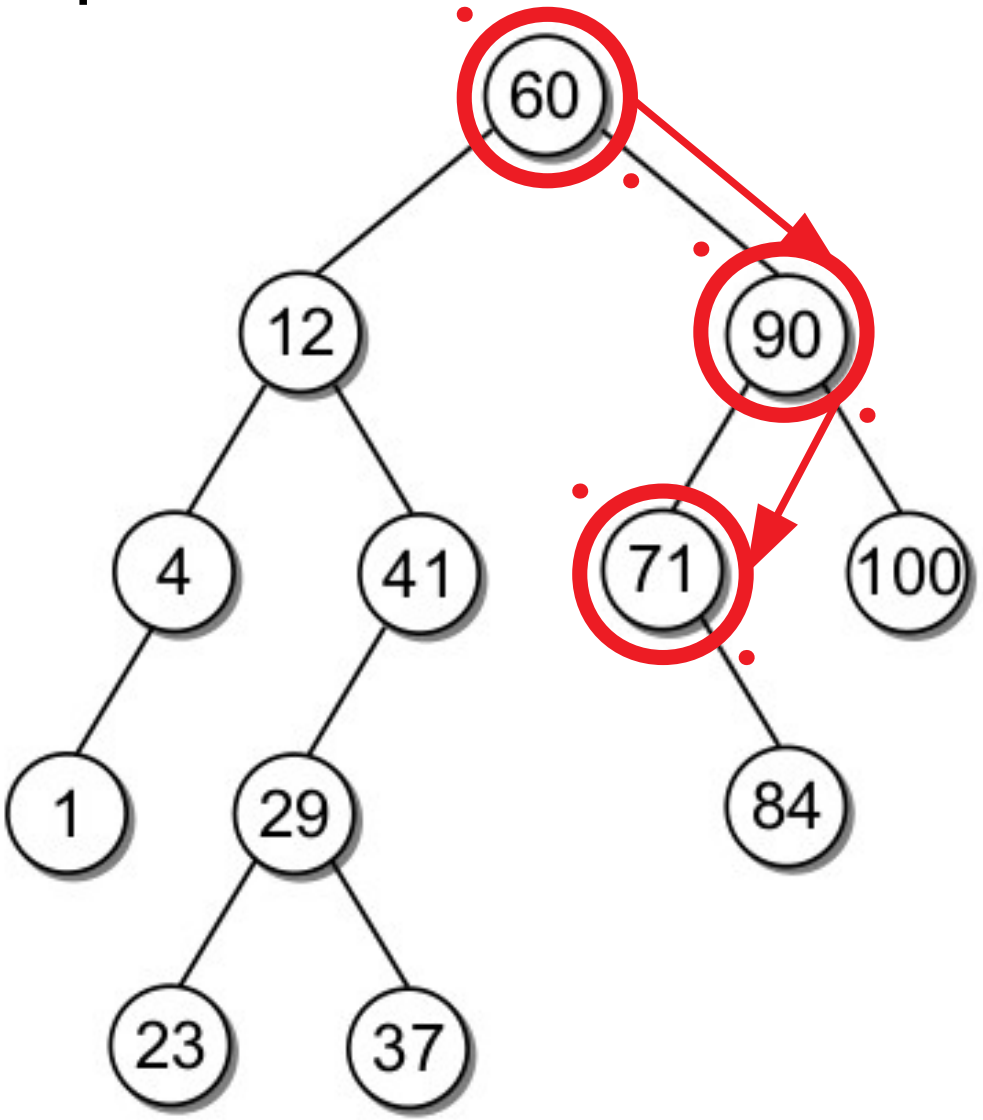
Operação de Busca

- Suponha que vamos buscar o elemento 68:



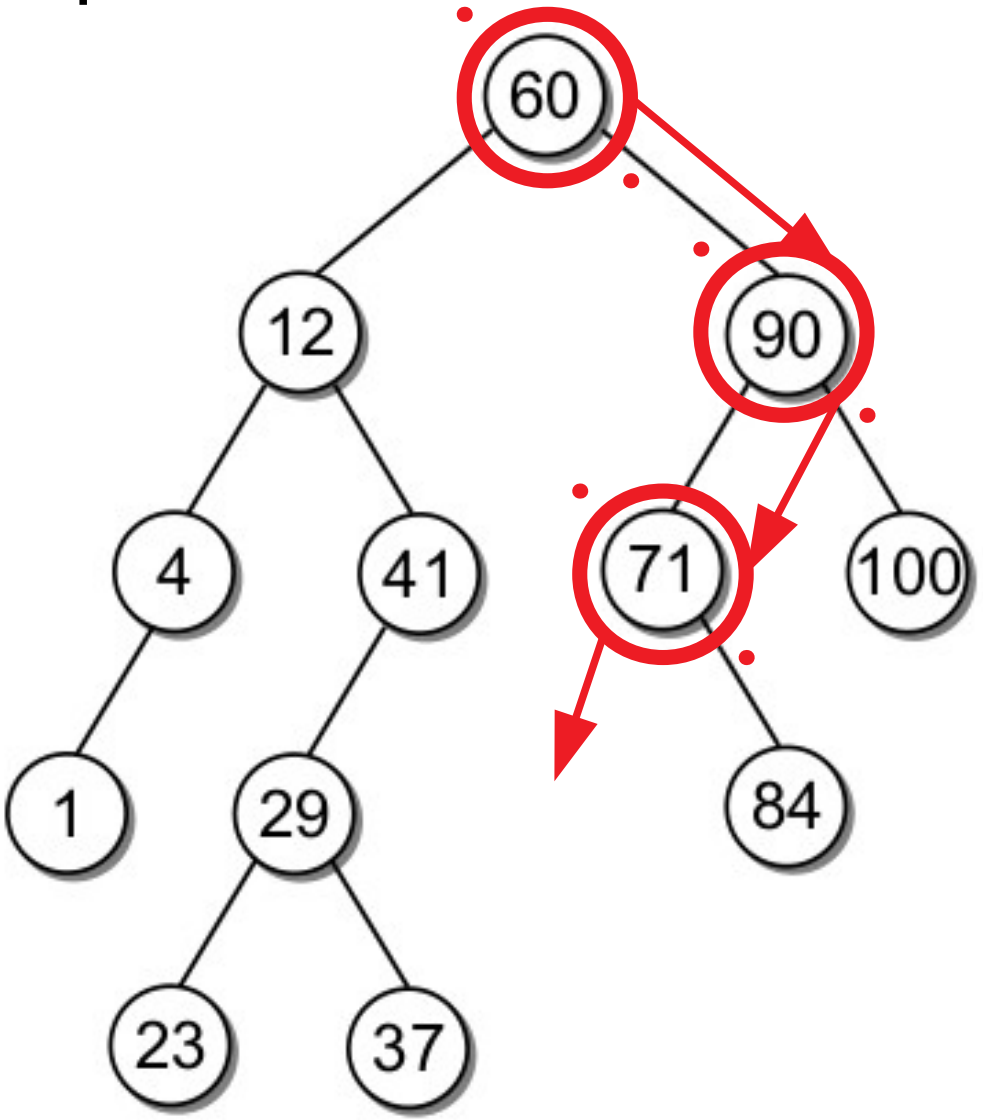
Operação de Busca

- Suponha que vamos buscar o elemento 68:



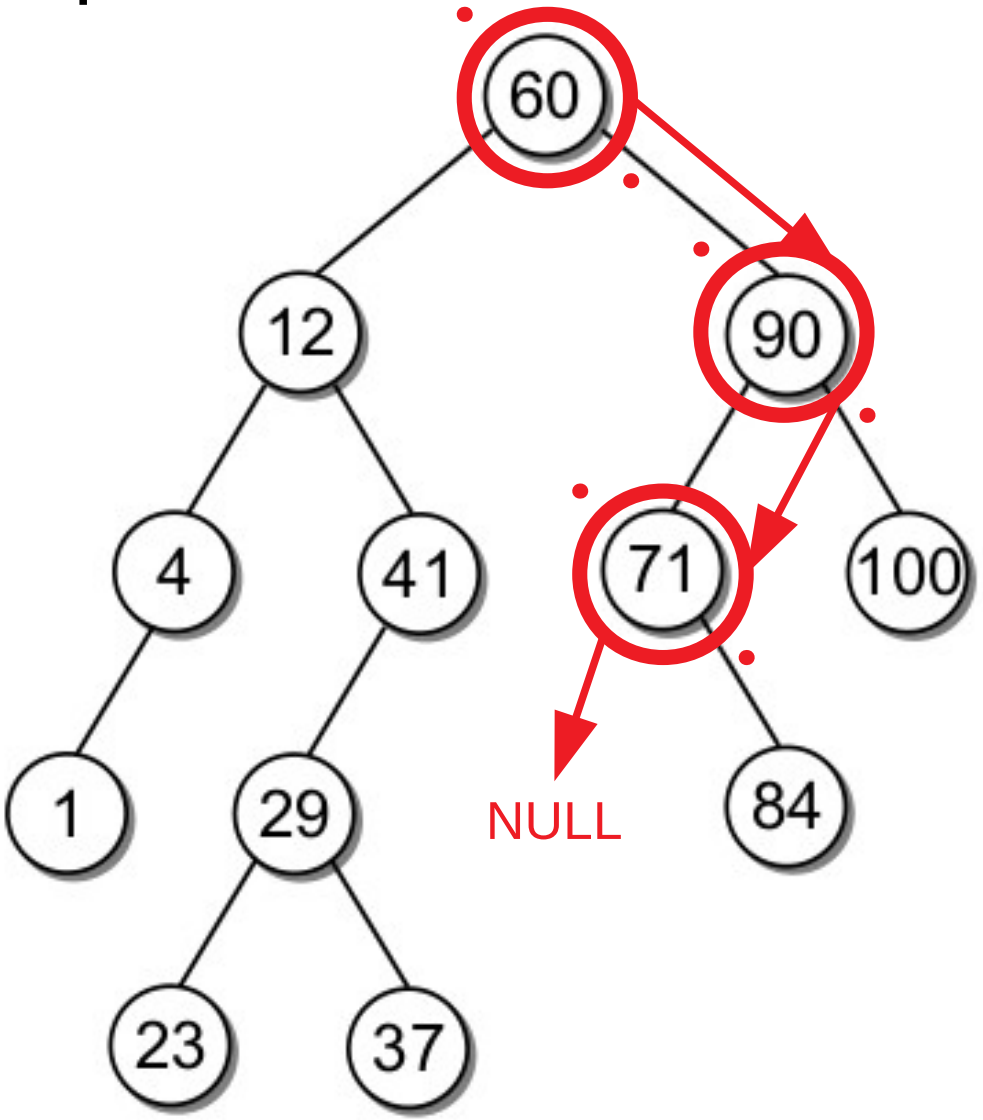
Operação de Busca

- Suponha que vamos buscar o elemento 68:



Operação de Busca

- Suponha que vamos buscar o elemento 68:



Operação de Busca

Operação de Busca

- As operações de busca na árvore binária podem ser implementadas iterativamente ou com recursão.

Operação de Busca

- As operações de busca na árvore binária podem ser implementadas iterativamente ou com recursão.
- Vamos ver um exemplo de implementação com recursão.

Operação de Busca

- As operações de busca na árvore binária podem ser implementadas iterativamente ou com recursão.
- Vamos ver um exemplo de implementação com recursão.
- O modo iterativo ficará como exercício!!!

Operação de Busca

Operação de Busca

- A função tem dois casos base:

Operação de Busca

- A função tem dois casos base:
  - O elemento alvo está no nó atual da busca ou

Operação de Busca

- A função tem dois casos base:
  - O elemento alvo está no nó atual da busca ou
  - Um nó filho NULL é encontrado.

Operação de Busca

- A função tem dois casos base:
  - O elemento alvo está no nó atual da busca ou
  - Um nó filho NULL é encontrado.
- Portanto, a função retornará:

Operação de Busca

- A função tem dois casos base:
  - O elemento alvo está no nó atual da busca ou
  - Um nó filho NULL é encontrado.
- Portanto, a função retornará:
  - A referência ao nó que contém o elemento procurado ou

Operação de Busca

- A função tem dois casos base:
  - O elemento alvo está no nó atual da busca ou
  - Um nó filho NULL é encontrado.
- Portanto, a função retornará:
  - A referência ao nó que contém o elemento procurado ou
  - `None`.

Operação de Busca

Operação de Busca

```
1  class BSTMap :
2  # ...
3     # Determines if the map contains the given key.
4    def __contains__( self, key ):
5      return self._bstSearch( self._root, key ) is not None
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

Operação de Busca

```
1  class BSTMap :
2  # ...
3     # Determines if the map contains the given key.
4    def __contains__( self, key ):
5      return self._bstSearch( self._root, key ) is not None
6
7     # Returns the value associated with the key.
8    def valueOf( self, key ):
9      node = self._bstSearch( self._root, key )
10     assert node is not None, "Invalid map key."
11     return node.value
12
13
14
15
16
17
18
19
20
21
22
```

Operação de Busca

```
1  class BSTMap :
2  # ...
3     # Determines if the map contains the given key.
4    def __contains__( self, key ):
5      return self._bstSearch( self._root, key ) is not None
6
7     # Returns the value associated with the key.
8    def valueOf( self, key ):
9      node = self._bstSearch( self._root, key )
10     assert node is not None, "Invalid map key."
11     return node.value
12
13    # Helper method that recursively searches the tree for a target key.
14   def _bstSearch( self, subtree, target ):
15     if subtree is None :          # base case
16       return None
17     elif target < subtree.key :  # target is left of the subtree root.
18       return self._bstSearch( subtree.left )
19     elif target > subtree.key :  # target is right of the subtree root.
20       return self._bstSearch( subtree.right )
21     else :                        # base case
22       return subtree
```

Busca por valores Max e Min

Busca por valores Max e Min

- Uma outra operação de busca comumente executada em uma árvore binária de busca é encontrar os maiores e menores valores (chaves).

Busca por valores Max e Min

- Uma outra operação de busca comumente executada em uma árvore binária de busca é encontrar os maiores e menores valores (chaves).
- De acordo com a definição de ABB, nós sabemos que o valor mínimo está na raiz ou em algum nó a sua esquerda.

Busca por valores Max e Min

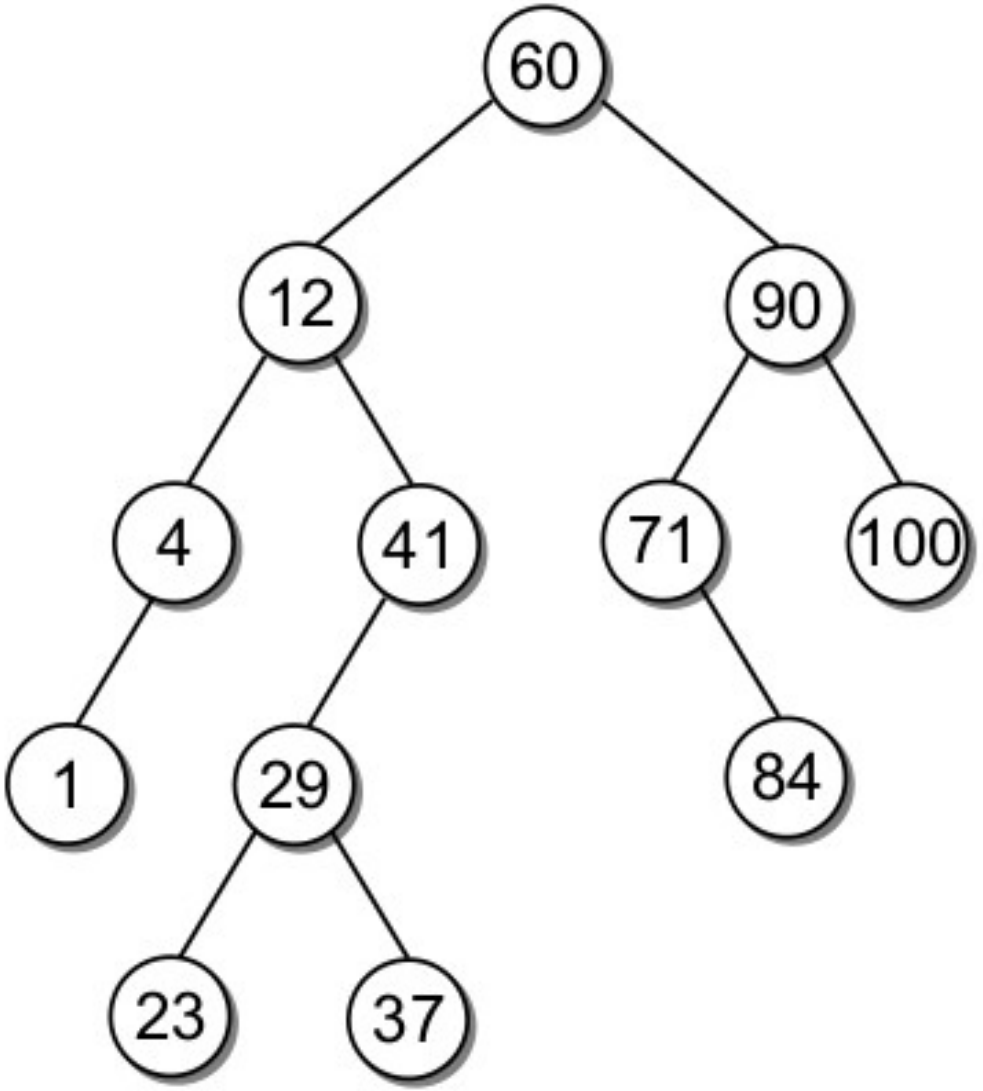
- Uma outra operação de busca comumente executada em uma árvore binária de busca é encontrar os maiores e menores valores (chaves).
- De acordo com a definição de ABB, nós sabemos que o valor mínimo está na raiz ou em algum nó a sua esquerda.
- Existe alguma maneira de fazer a busca pelo menor elemento sem ter a necessidade de fazer a comparação individual?

Busca por valores Max e Min

- Como achar o menor valor na ABB abaixo?

Busca por valores Max e Min

- Como achar o menor valor na ABB abaixo?



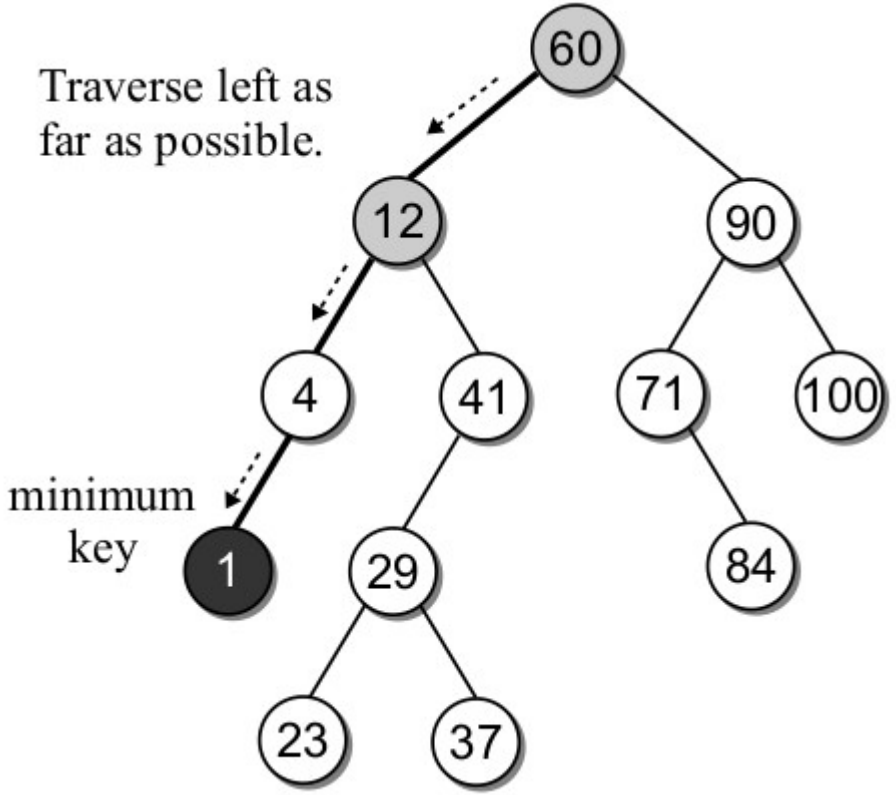
Busca por valores Max e Min

Busca por valores Max e Min

- A ideia é que se o nó raiz contém um elemento a sua esquerda, então esse nó não pode ser o menor.

Busca por valores Max e Min

- A ideia é que se o nó raiz contém um elemento a sua esquerda, então esse nó não pode ser o menor.



Busca por valores Max e Min

- A mesma ideia da busca pelo menor valor se aplica à busca do maior valor.

Busca por valores Max e Min

- Código:

```
1  class BSTMap :
2  # ...
3     # Helper method for finding the node containing the minimum key.
4  def _bstMinumum( self, subtree ):
5      if subtree is None :
6        return None
7      elif subtree.left is None :
8        return subtree
9      else :
10       return self._bstMinimum( subtree.left )
```

Inserção

Inserção

- Na construção de uma ABB, as chaves (elementos) são adicionados um por vez.

Inserção

- Na construção de uma ABB, as chaves (elementos) são adicionados um por vez.
- Suponha que vamos inserir a seguinte relação de elementos [60, 25, 100, 35, 17, 80].

Inserção

Inserção

Inserir 60

Inserção

Inserir 60

Inserção

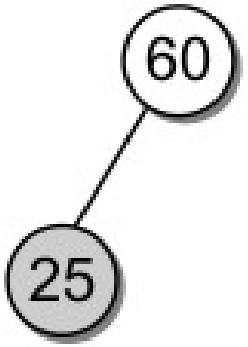
Inserir 60

Inserir 25

Inserção

Inserir 60

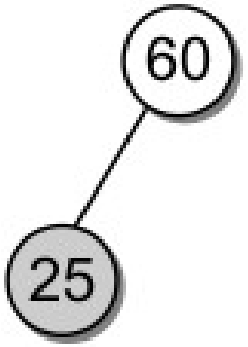
Inserir 25



Inserção

Inserir 60

Inserir 25

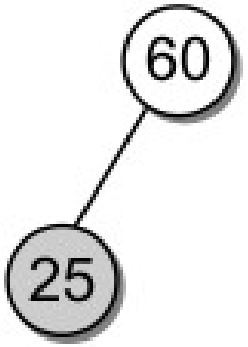


Inserir 100

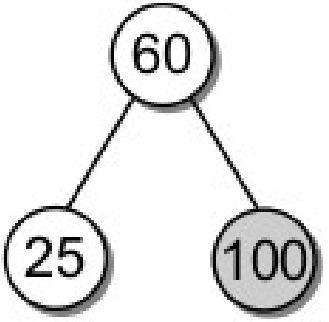
Inserção

Inserir 60

Inserir 25



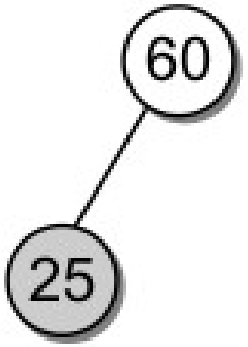
Inserir 100



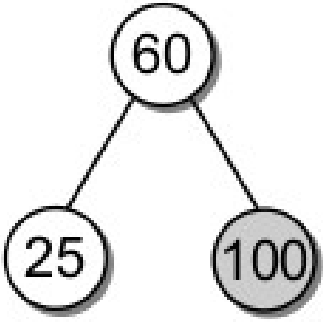
Inserção

Inserir 60

Inserir 25



Inserir 100

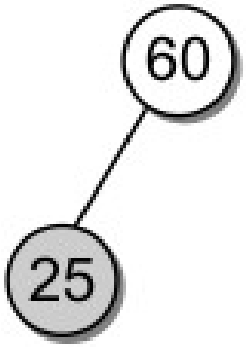


Inserir 35

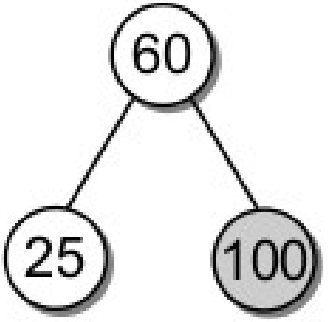
Inserção

Inserir 60

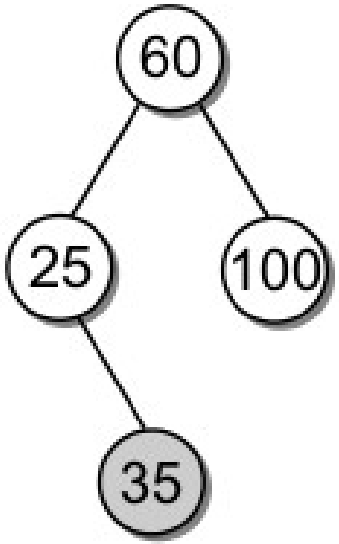
Inserir 25



Inserir 100



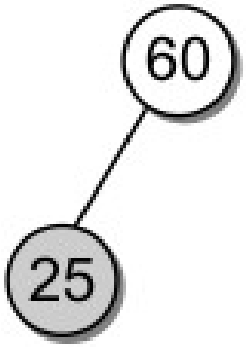
Inserir 35



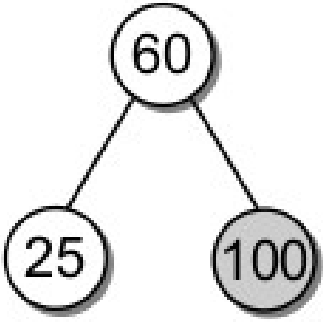
Inserção

Inserir 60

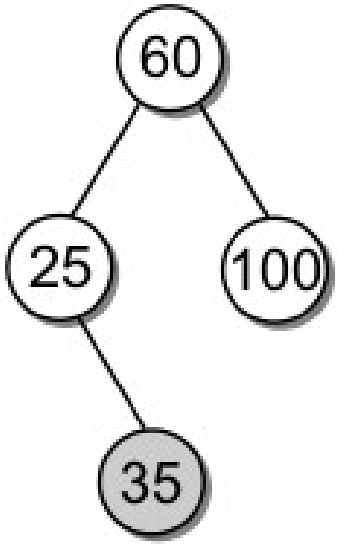
Inserir 25



Inserir 100



Inserir 35

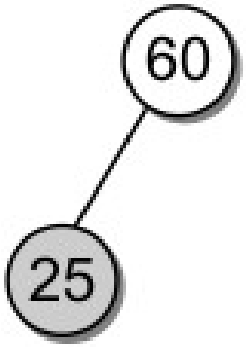


Inserir 17

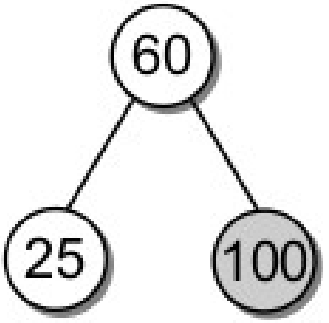
Inserção

Inserir 60

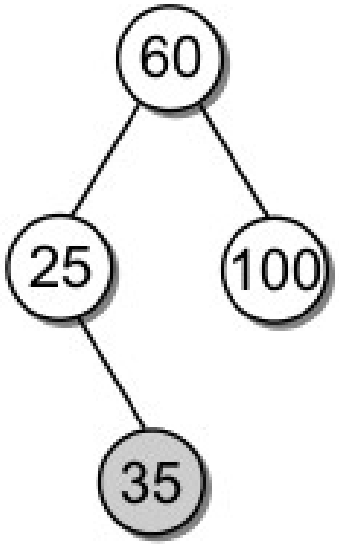
Inserir 25



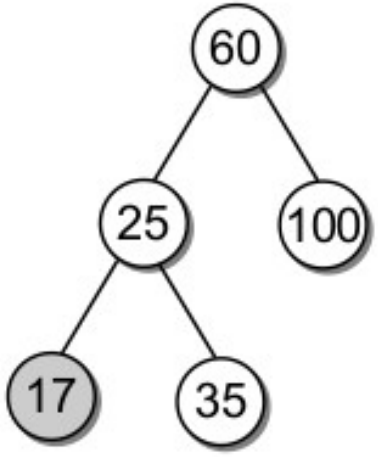
Inserir 100



Inserir 35



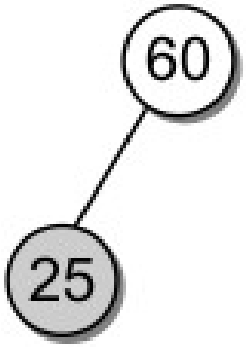
Inserir 17



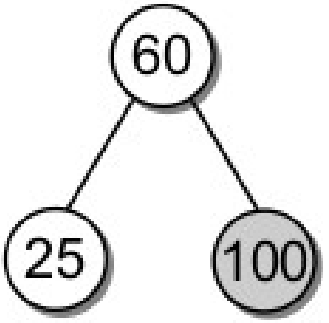
Inserção

Inserir 60

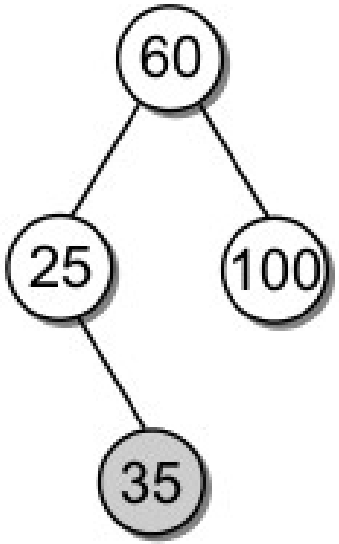
Inserir 25



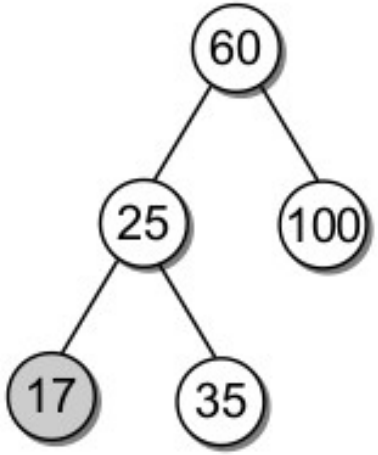
Inserir 100



Inserir 35



Inserir 17

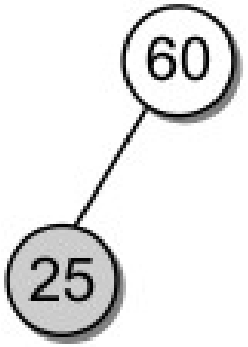


Inserir 80

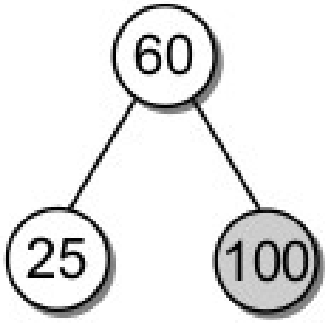
Inserção

Inserir 60

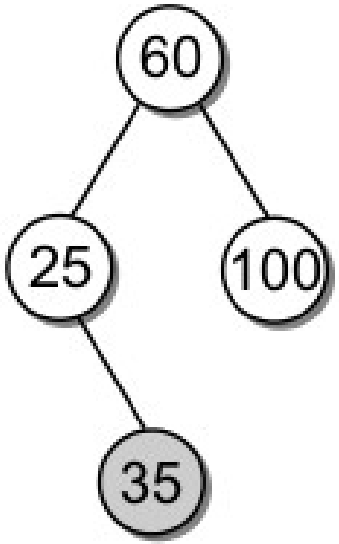
Inserir 25



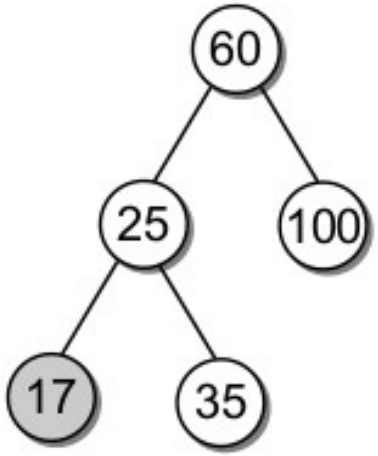
Inserir 100



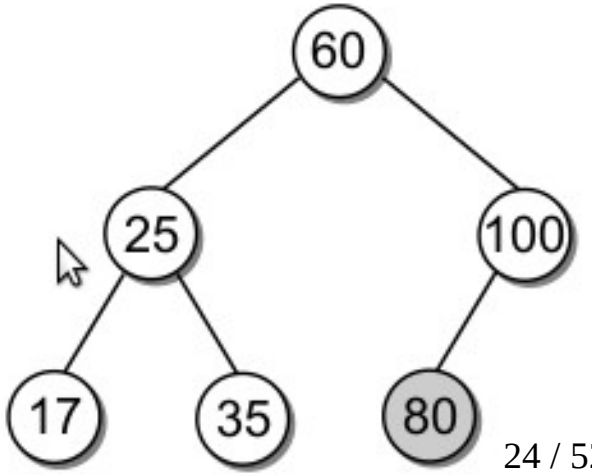
Inserir 35



Inserir 17



Inserir 80



Inserção

Inserção

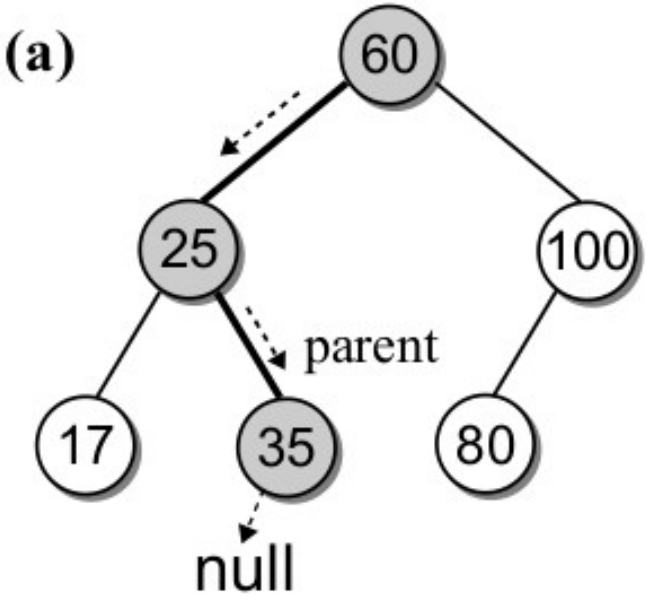
- Suponha que vamos inserir a chave 30 na árvore anterior.

Inserção

- Suponha que vamos inserir a chave 30 na árvore anterior.
- O que aconteceria se nós usarmos o função `_bstSearch( )` e pesquisarmos pelo nó 30?

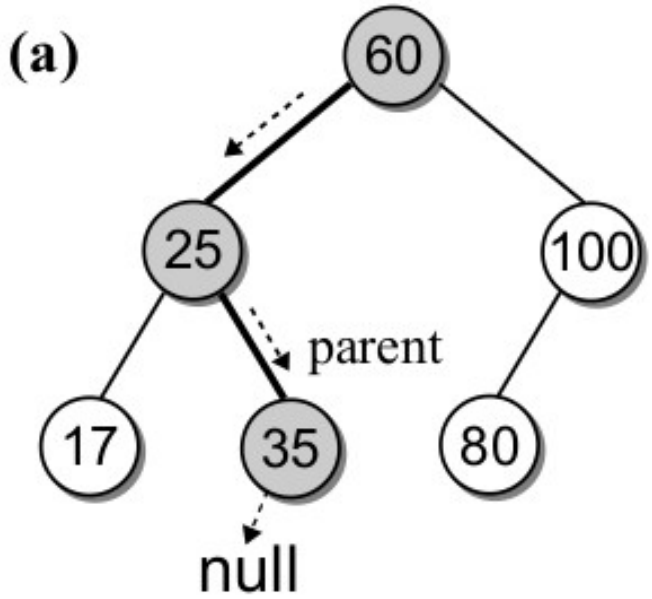
Inserção

- Suponha que vamos inserir a chave 30 na árvore anterior.
- O que aconteceria se nós usarmos o função `_bstSearch( )` e pesquisarmos pelo nó 30?



Inserção

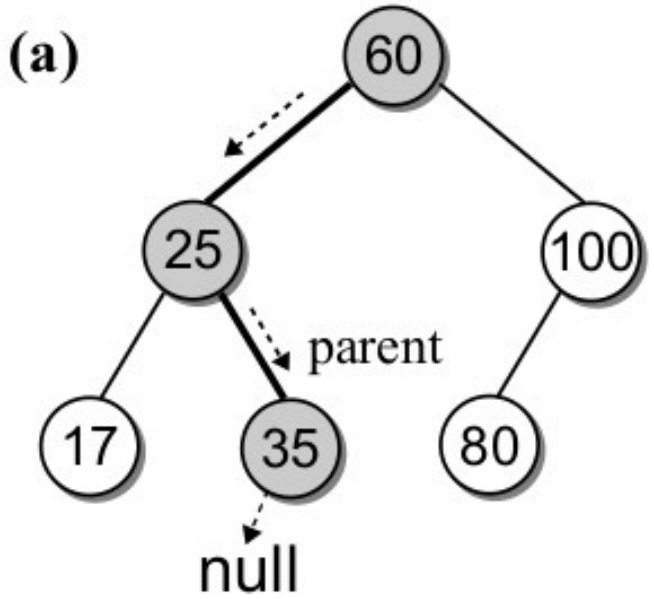
- Suponha que vamos inserir a chave 30 na árvore anterior.
- O que aconteceria se nós usarmos o função `_bstSearch( )` e pesquisarmos pelo nó 30?



A busca não localizará o nó 30.

Inserção

- Suponha que vamos inserir a chave 30 na árvore anterior.
- O que aconteceria se nós usarmos o função `_bstSearch( )` e pesquisarmos pelo nó 30?



A busca não localizará o nó 30.

Porém, este será o lugar correto onde a nova chave deverá ser inserida.

Inserção

- Código para inserir um nó na ABB:

```
 # Adds a new entry to the map or replaces the value of an existing key.
def add( self, key, value ):
   # Find the node containing the key, if it exists.
  node = self._bstSearch( key )
   # If the key is already in the tree, update its value.
  if node is not None :
    node.value = value
    return False
   # Otherwise, add a new entry.
  else :
    self._root = self._bstInsert( self._root, key, value )
    self._size += 1
    return True
```

Inserção

- Código para inserir um nó na ABB:

```
 # Helper method that inserts a new item, recursively.
def _bstInsert( self, subtree, key, value ):
  if subtree is None :
    subtree = _BSTMapNode( key, value )
  elif key < subtree.key :
    subtree.left = self._bstInsert( subtree.left, key, value )
  elif key > subtree.key :
    subtree.right = self._bstInsert( subtree.right, key, value )
  return subtree
```

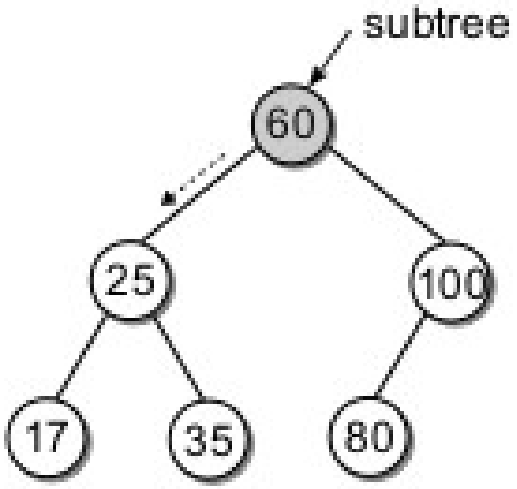
Inserção

Inserção

- O método deve pesquisar a localização do novo nó.

Inserção

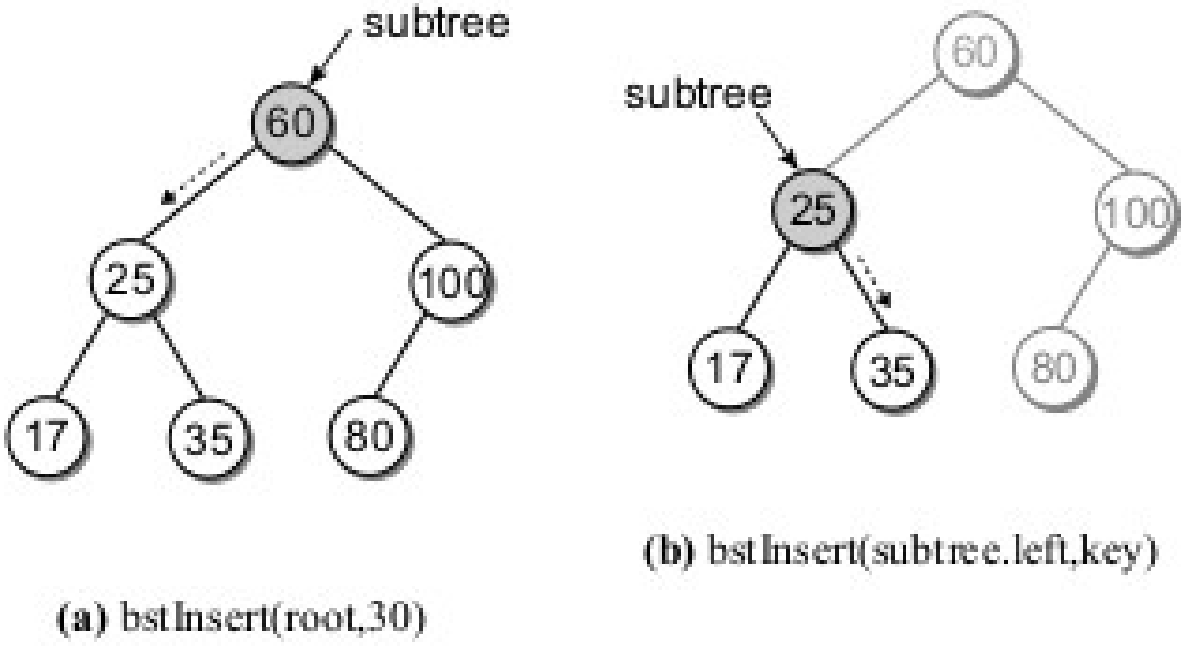
- O método deve pesquisar a localização do novo nó.



(a) bstInsert(root,30)

Inserção

- O método deve pesquisar a localização do novo nó.

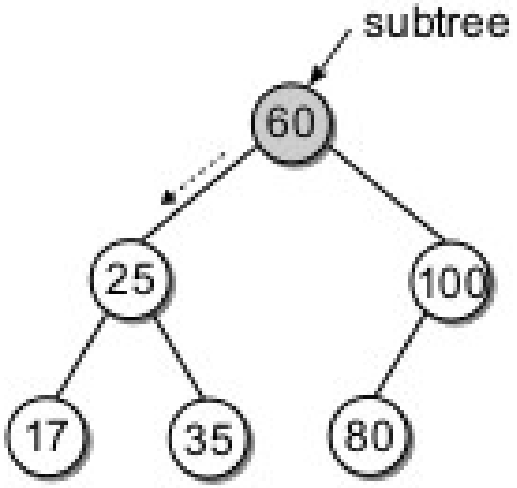


(a) bstInsert(root,30)

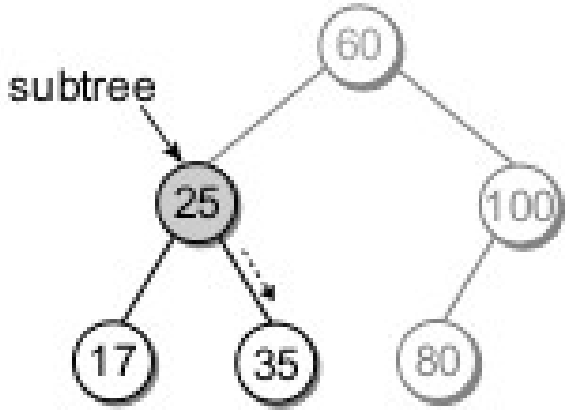
(b) bstInsert(subtree.left,key)

Inserção

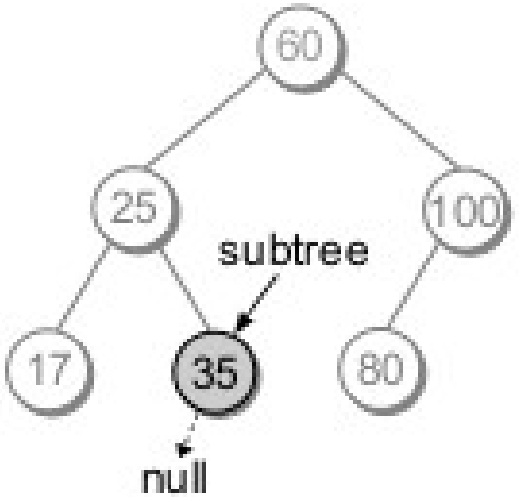
- O método deve pesquisar a localização do novo nó.



(a) bstInsert(root,30)



(b) bstInsert(subtree.left,key)



(c) bstInsert(subtree.right,key)

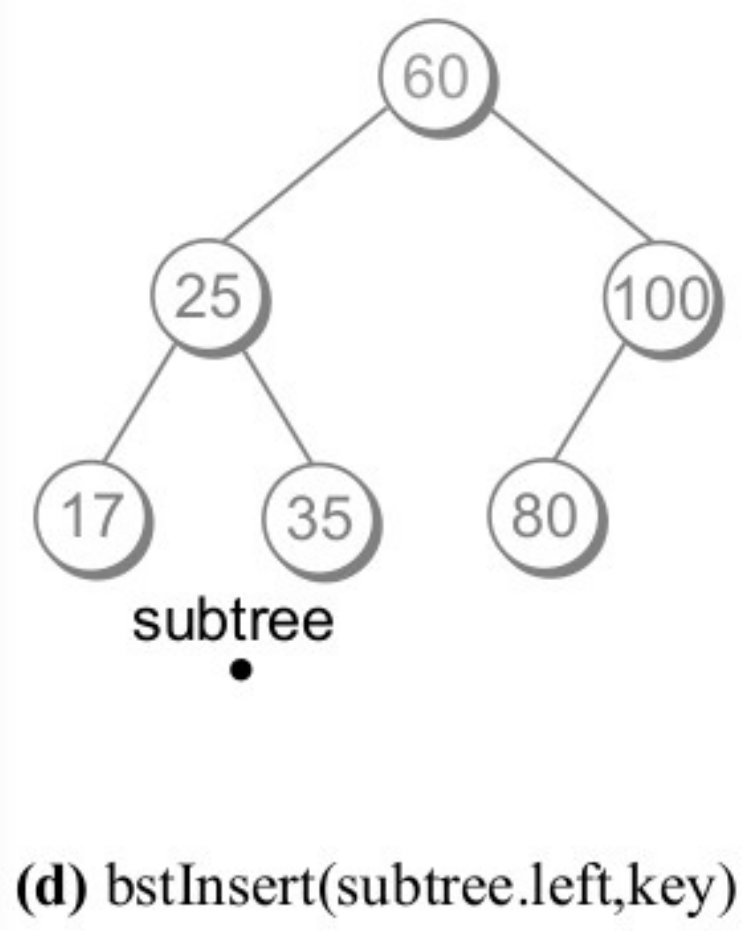
Inserção

Inserção

- O caso base é alcançado quando uma subárvore vazia é encontrada.

Inserção

- O caso base é alcançado quando uma subárvore vazia é encontrada.



(d) bstInsert(subtree.left,key)

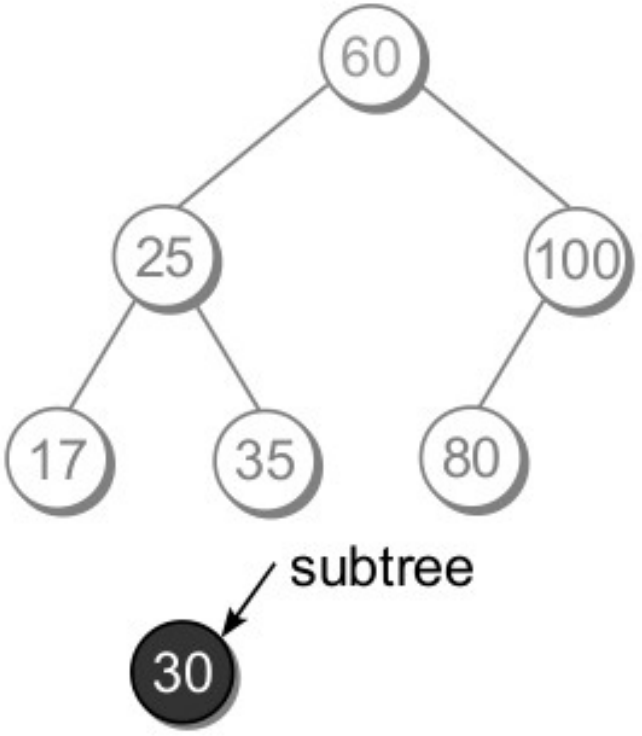
Inserção

Inserção

- Neste momento, um novo nó da árvore é criado e seus valores são atualizados.

Inserção

- Neste momento, um novo nó da árvore é criado e seus valores são atualizados.



(e) subtree = TreeNode(key)

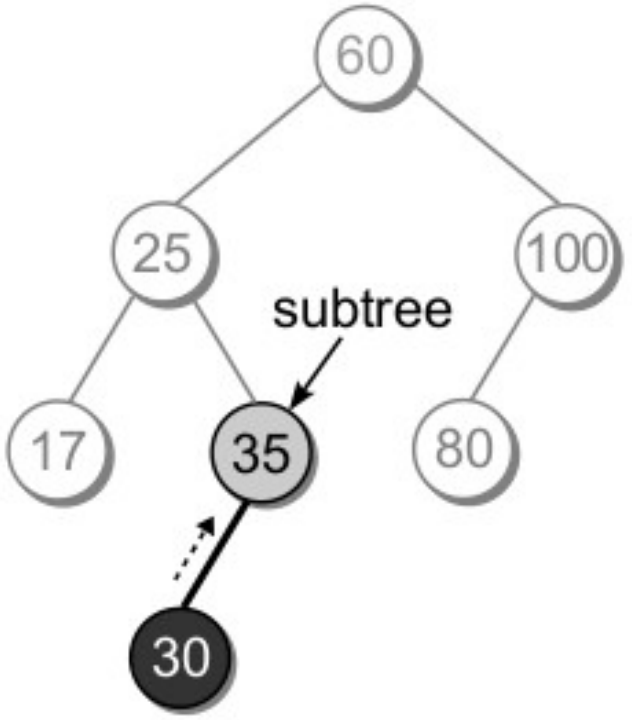
Inserção

Inserção

- A recursão continua...

Inserção

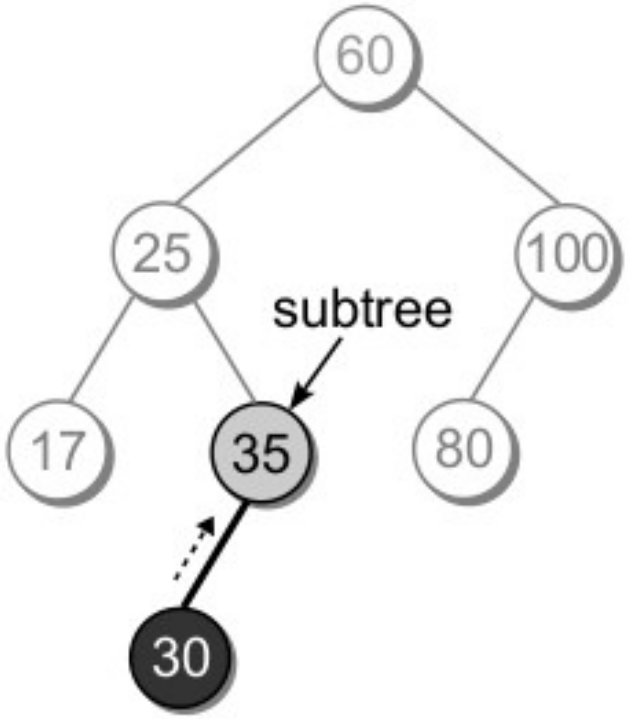
- A recursão continua...



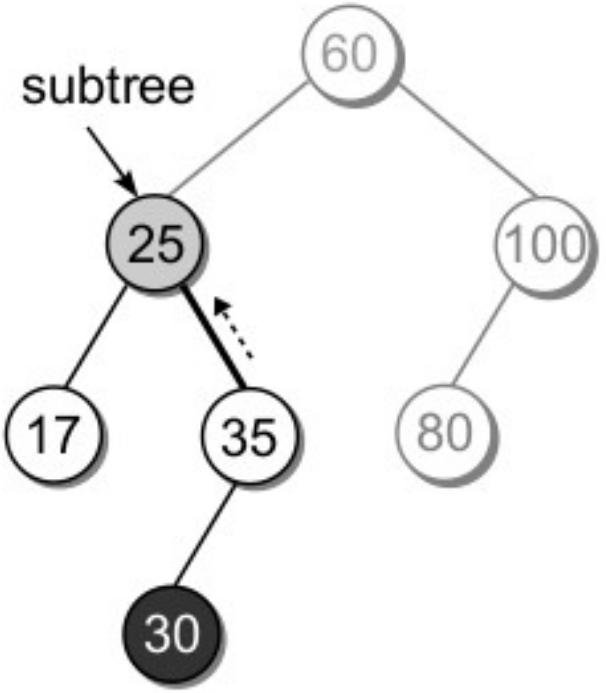
(f) subtree.left = bstInsert(...)

Inserção

- A recursão continua...



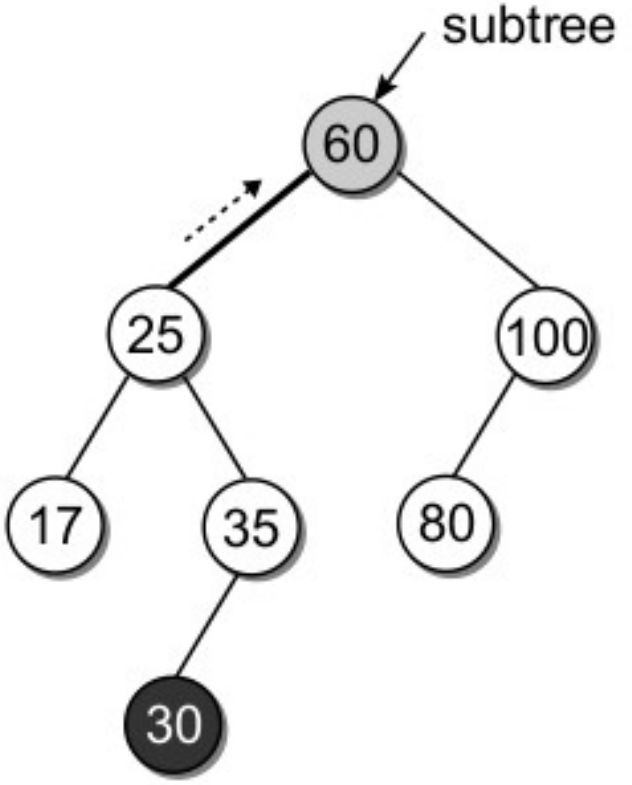
(f) subtree.left = bstInsert(...)



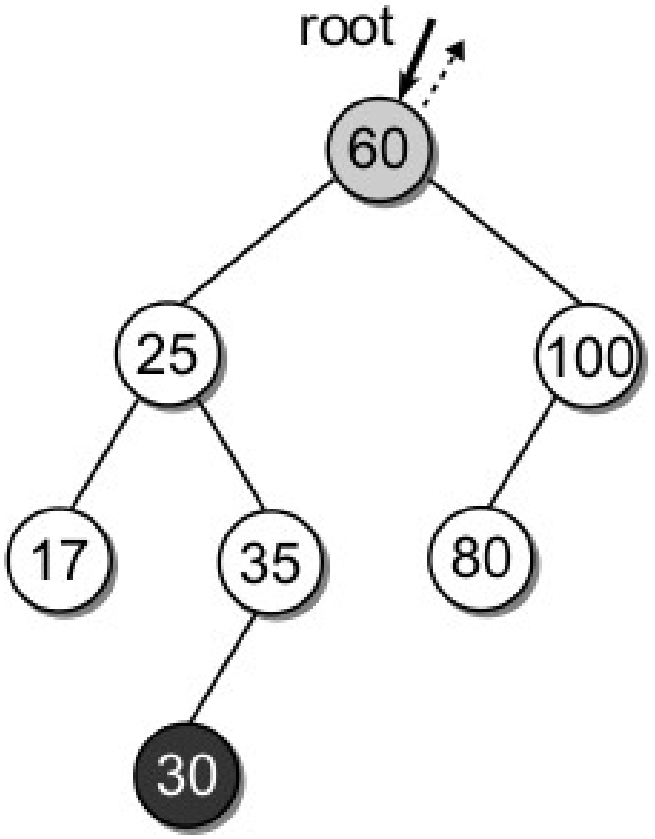
(g) subtree.right = bstInsert(...)

Inserção

- Até alcançar o nó raiz.



(h) subtree.left = bstInsert(...)



(i) root = bstInsert(...)

Remoção

Remoção

- A operação de remoção é um pouco mais complicada do que as operações de busca ou inserção.

Remoção

- A operação de remoção é um pouco mais complicada do que as operações de busca ou inserção.
- Operações envolvidas na remoção:

Remoção

- A operação de remoção é um pouco mais complicada do que as operações de busca ou inserção.
- Operações envolvidas na remoção:
  - Localizar o elemento.

Remoção

- A operação de remoção é um pouco mais complicada do que as operações de busca ou inserção.
- Operações envolvidas na remoção:
  - Localizar o elemento.
  - Remover o nó da árvore.

Remoção

- A operação de remoção é um pouco mais complicada do que as operações de busca ou inserção.
- Operações envolvidas na remoção:
  - Localizar o elemento.
  - Remover o nó da árvore.
  - Os nós restantes devem preservar a propriedade de ABB.

Remoção

- Existem três casos a serem considerados antes de remover um nó:
  - O nó é uma folha.
  - O nó tem um único filho.
  - O nó tem dois filhos.

Remoção

Remoção

- Remoção de nó folha

Remoção

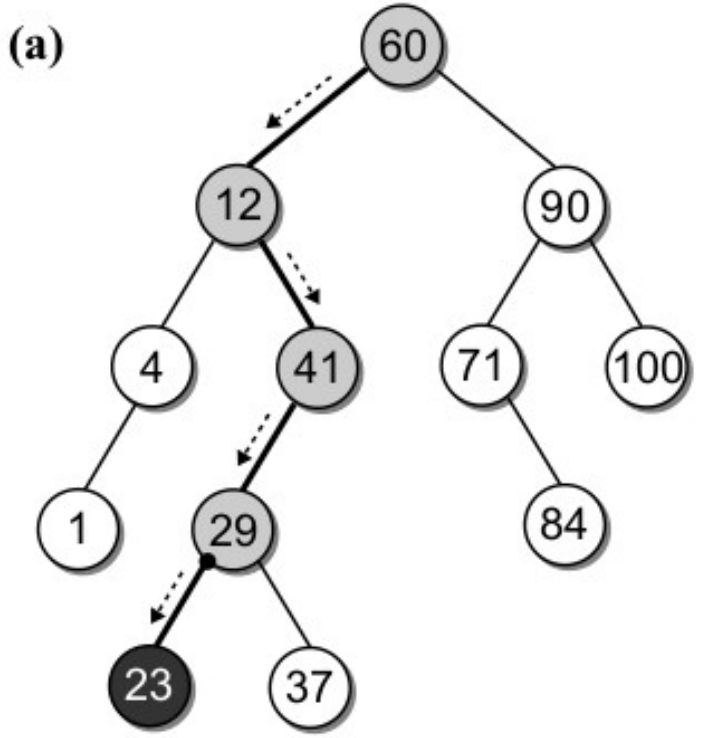
- Remoção de nó folha
  - É o caso mais simples.

Remoção

- Remoção de nó folha
  - É o caso mais simples.
  - Depois de encontrar o nó, basta removê-lo da ABB.

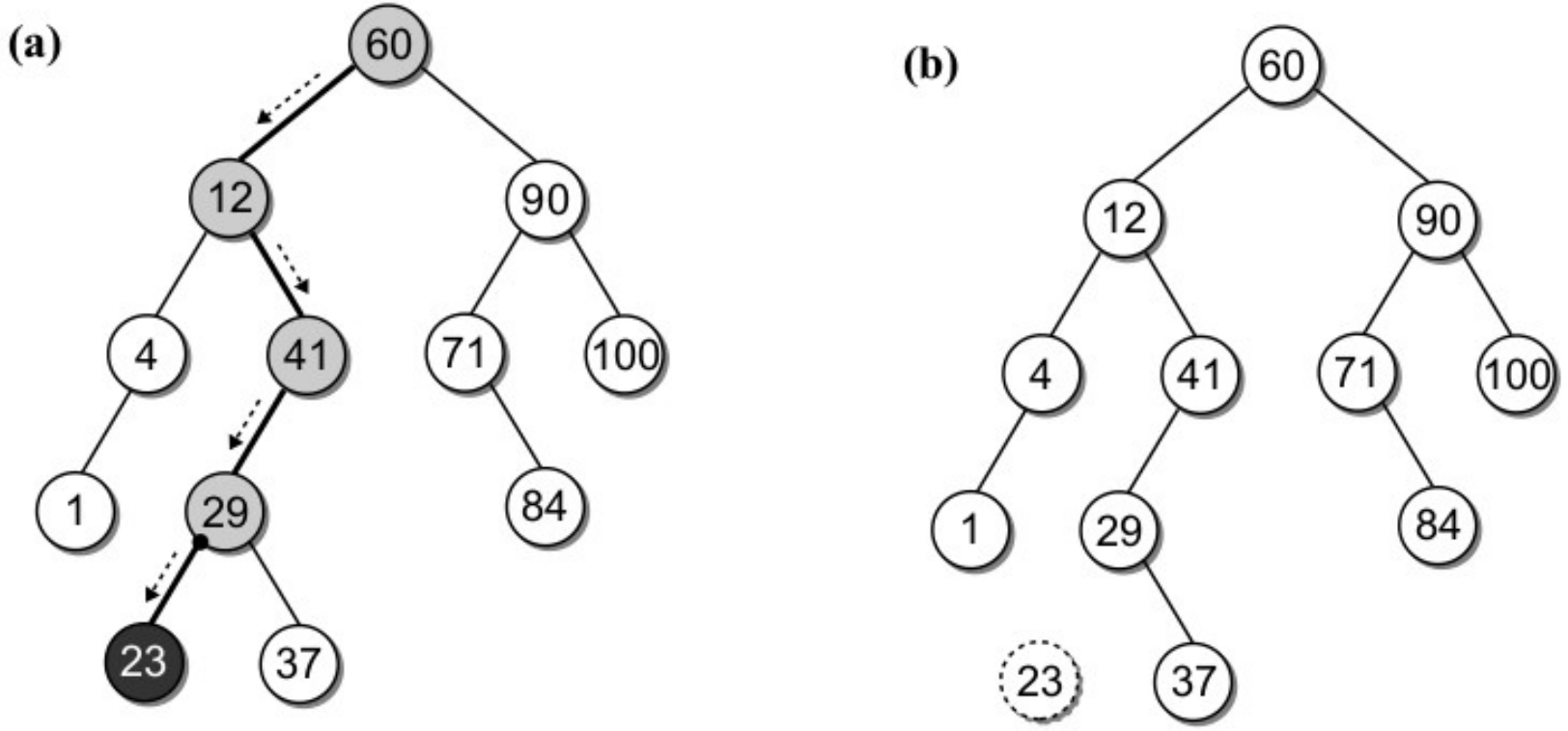
Remoção

- Remoção de nó folha
  - É o caso mais simples.
  - Depois de encontrar o nó, basta removê-lo da ABB.



Remoção

- Remoção de nó folha
  - É o caso mais simples.
  - Depois de encontrar o nó, basta removê-lo da ABB.



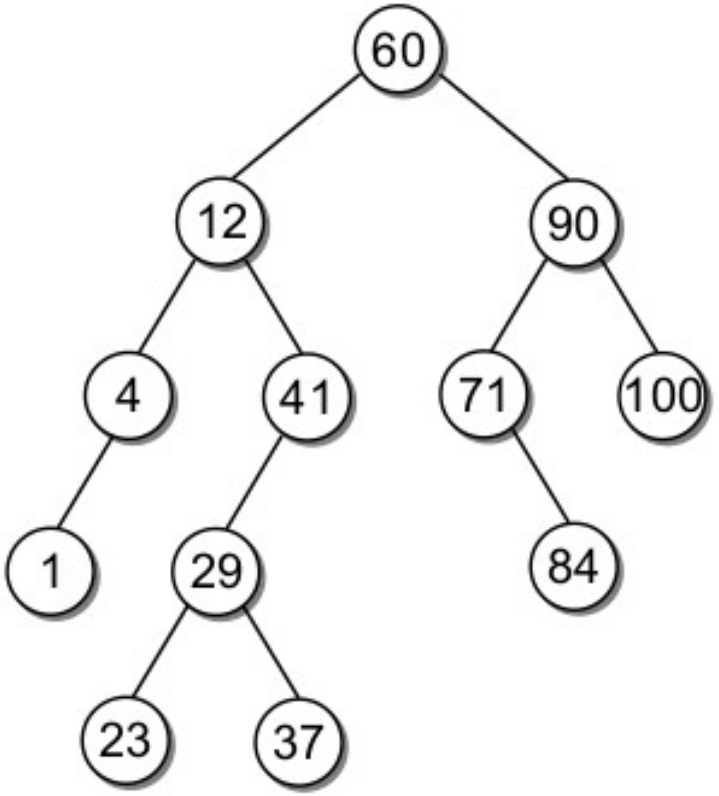
Remoção

Remoção

- Remover nó interior com 1 filho

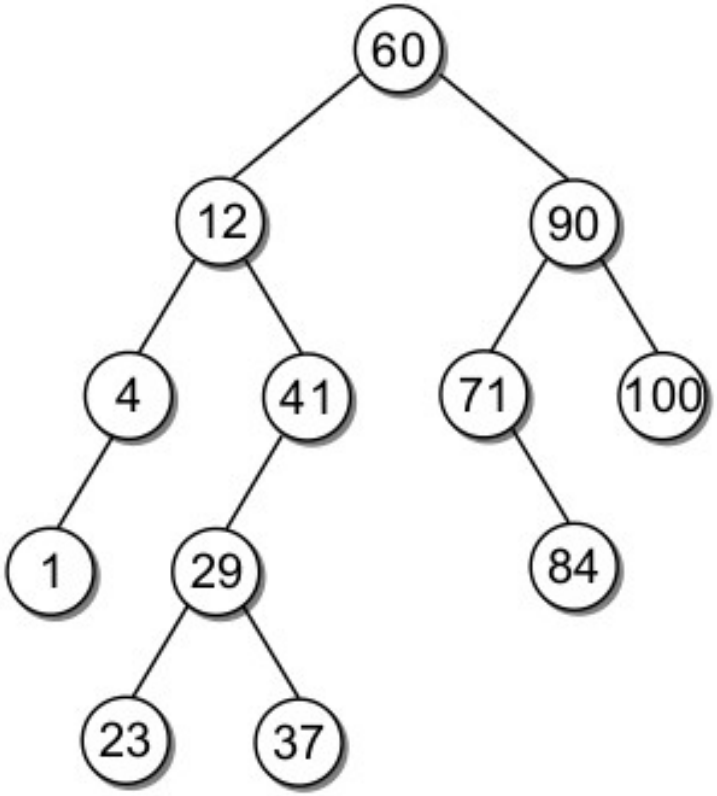
Remoção

- Remover nó interior com 1 filho



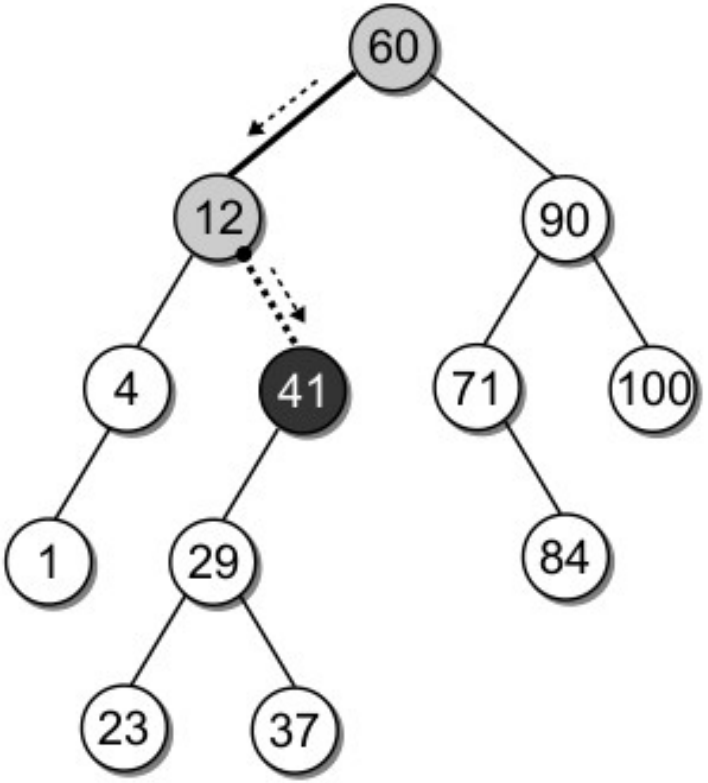
Remoção

- Remover nó interior com 1 filho
  - Como seria remover o nó 41?



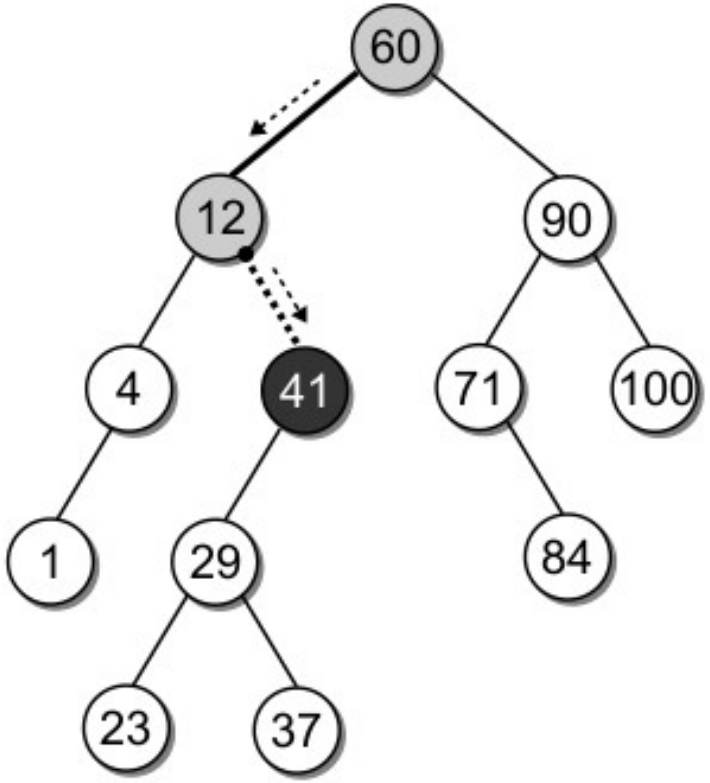
Remoção

Remoção



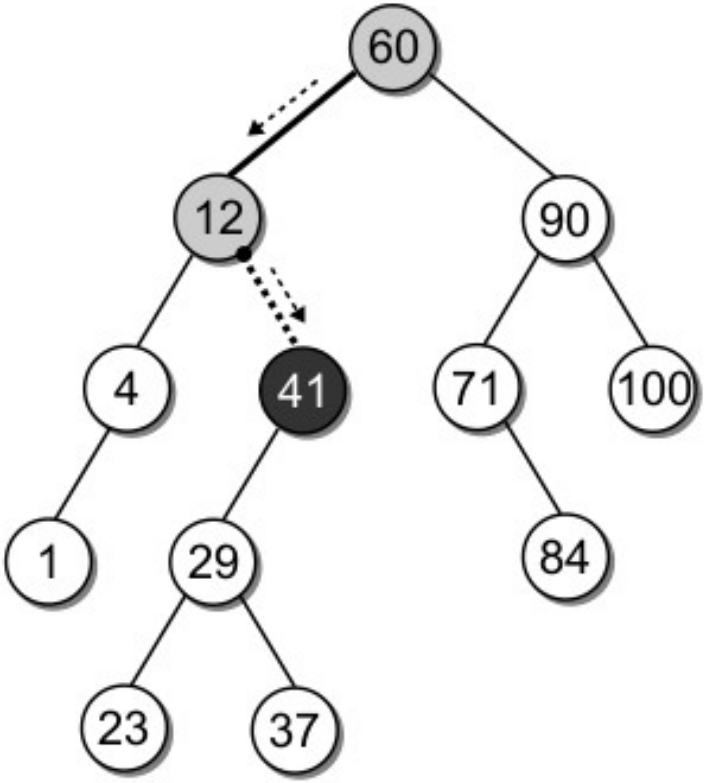
Remoção

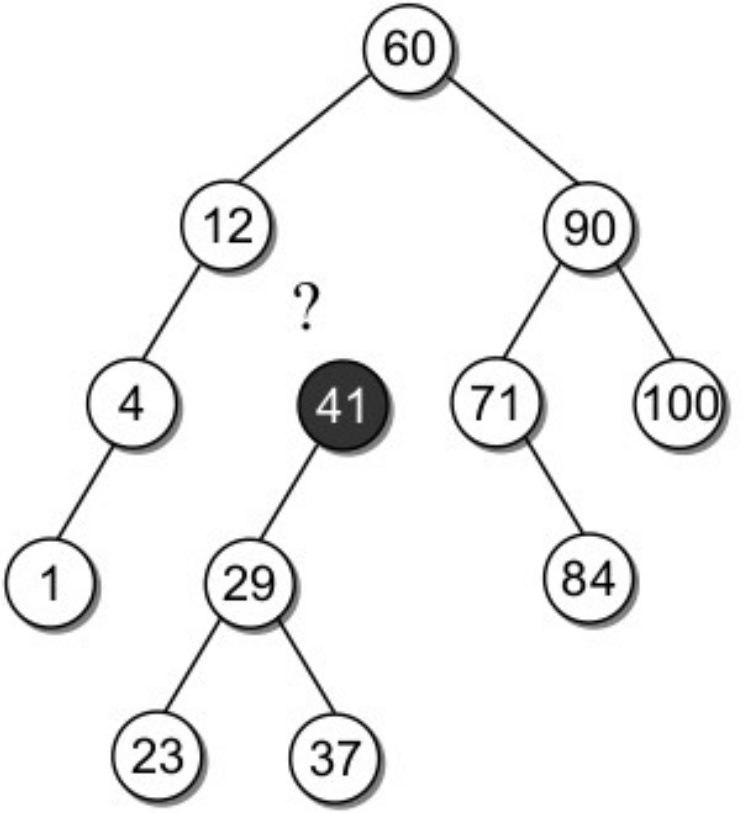
- Bastaria que o nó anterior (12) apontasse para NULL?



Remoção

- Bastaria que o nó anterior (12) apontasse para NULL?





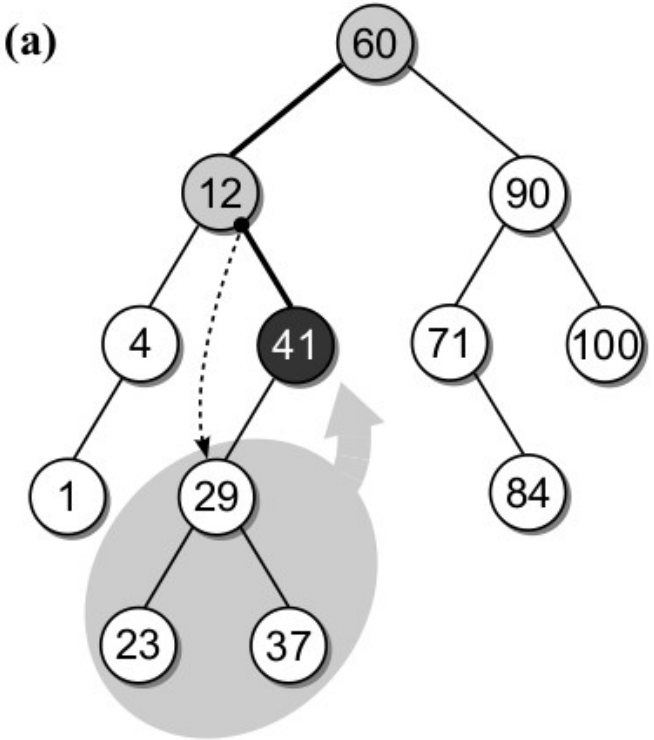
Remoção

Remoção

- Como o nó 41 tem apenas 1 único filho, então todos os seus descendentes têm chaves maiores do que o nó 12.

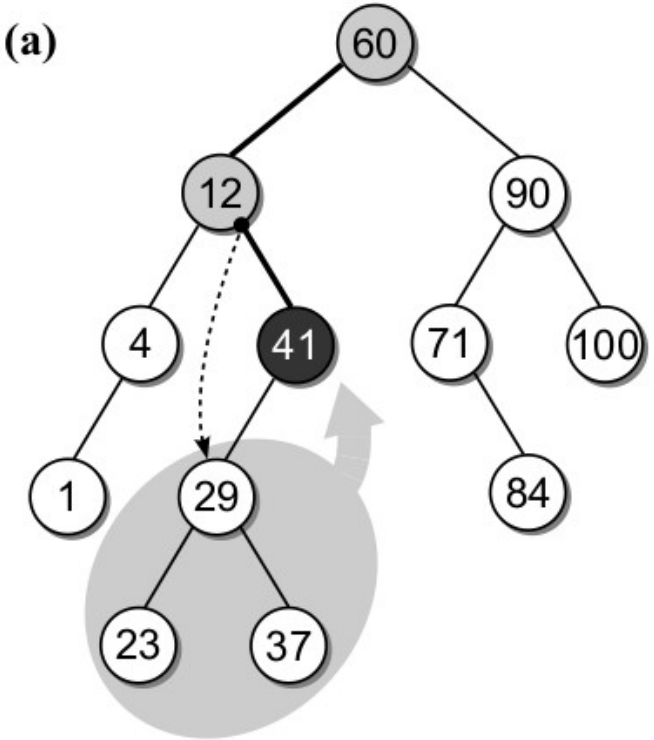
Remoção

- Como o nó 41 tem apenas 1 único filho, então todos os seus descendentes têm chaves maiores do que o nó 12.



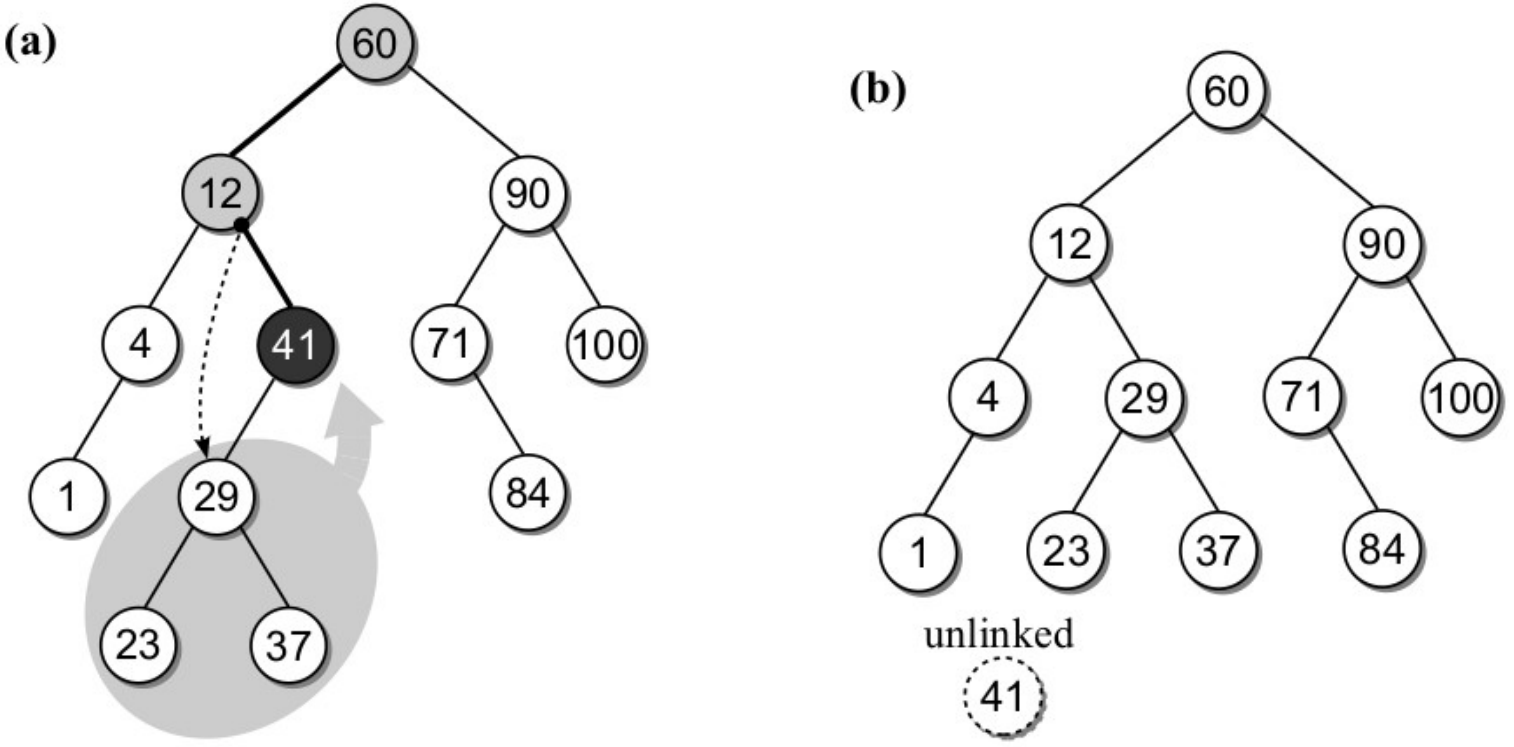
Remoção

- Como o nó 41 tem apenas 1 único filho, então todos os seus descendentes têm chaves maiores do que o nó 12.
- Assim, basta ligar o nó 12 ao filho do nó 41.



Remoção

- Como o nó 41 tem apenas 1 único filho, então todos os seus descendentes têm chaves maiores do que o nó 12.
- Assim, basta ligar o nó 12 ao filho do nó 41.



Remoção

Remoção

- Remover nó interior com 2 filhos

Remoção

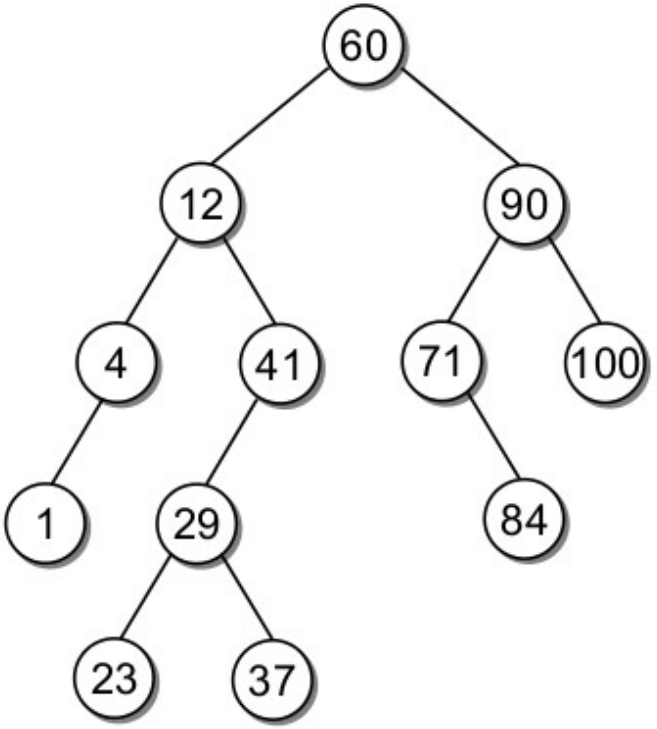
- Remover nó interior com 2 filhos
  - Essa é a sua situação mais difícil.

Remoção

- Remover nó interior com 2 filhos
  - Essa é a sua situação mais difícil.
  - O que aconteceria na árvore para remover o nó 12?

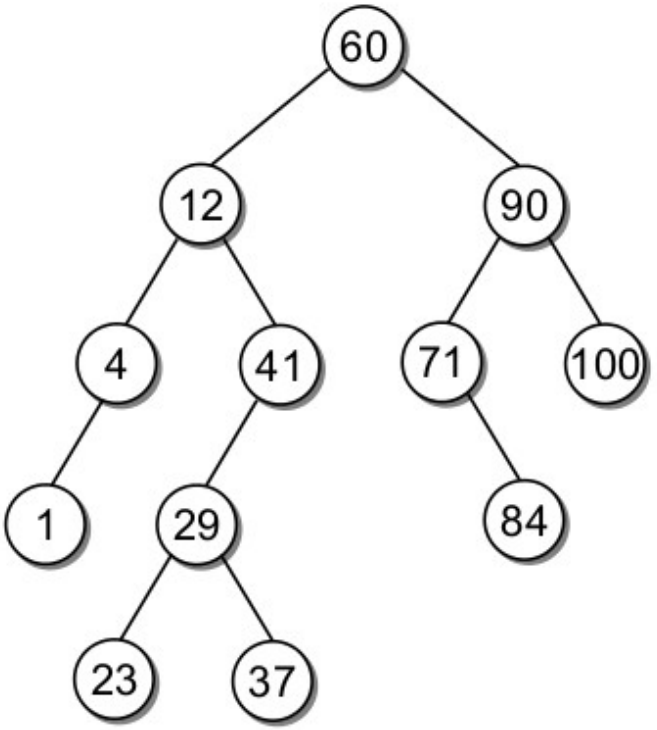
Remoção

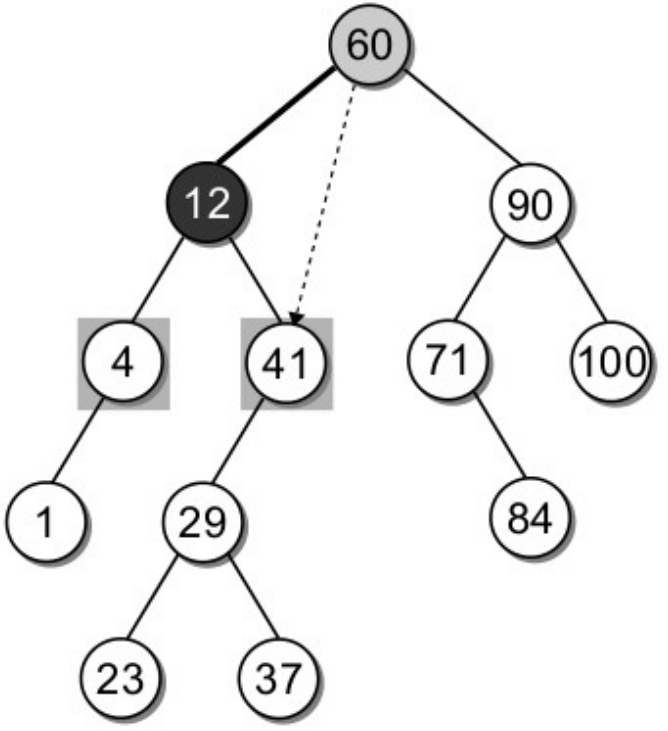
- Remover nó interior com 2 filhos
  - Essa é a sua situação mais difícil.
  - O que aconteceria na árvore para remover o nó 12?



Remoção

- Remover nó interior com 2 filhos
  - Essa é a sua situação mais difícil.
  - O que aconteceria na árvore para remover o nó 12?





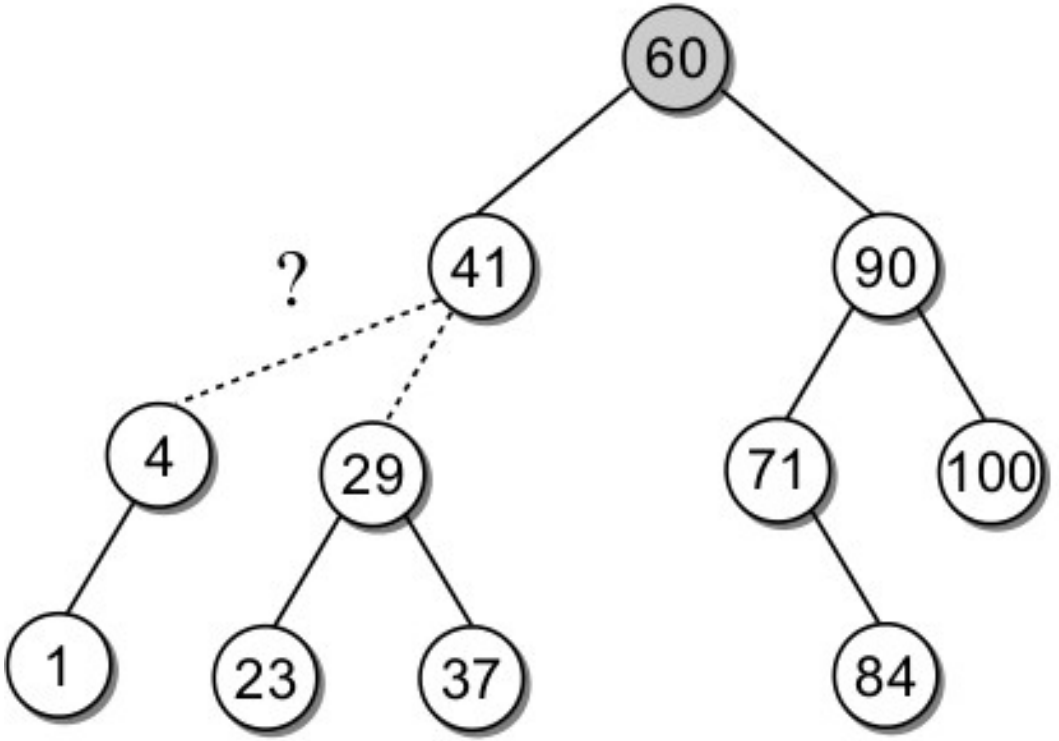
Remoção

Remoção

- O resultado da remoção anterior seria a árvore abaixo.

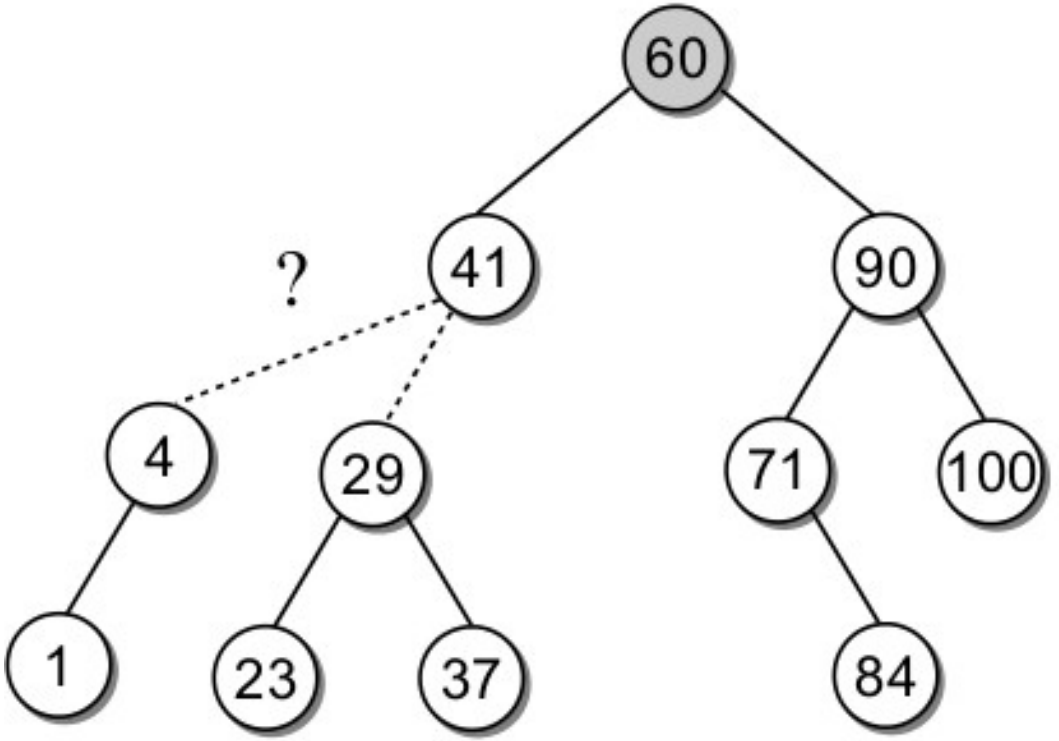
Remoção

- O resultado da remoção anterior seria a árvore abaixo.



Remoção

- O resultado da remoção anterior seria a árvore abaixo.
- O nó 41 teria dois filhos à esquerda.



Remoção

Remoção

- Remover o nó interior com dois filhos:

Remoção

- Remover o nó interior com dois filhos:
  - Encontrar o sucessor lógico, S, do nó que será removido N.

Remoção

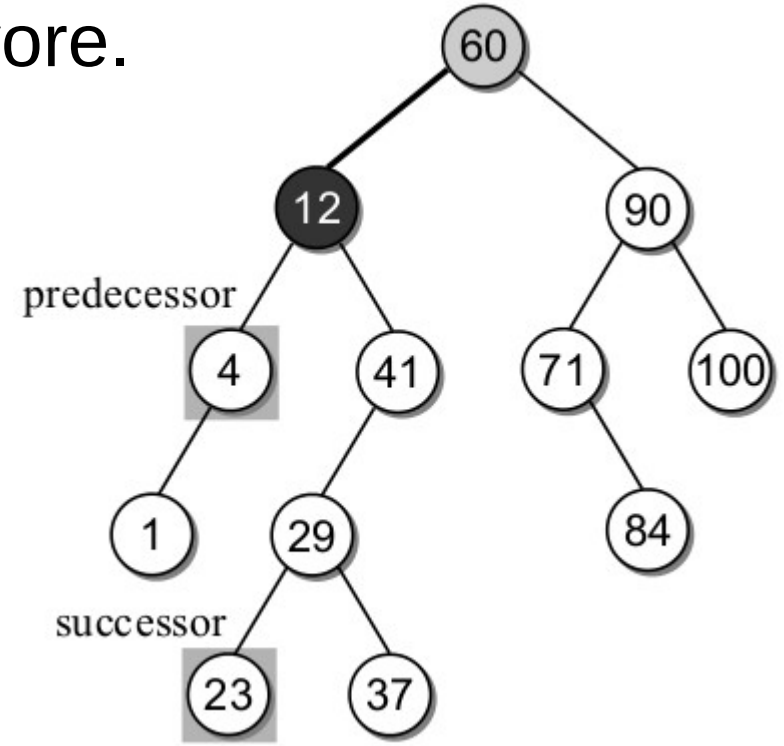
- Remover o nó interior com dois filhos:
  - Encontrar o sucessor lógico, S, do nó que será removido N.
  - Copiar a chave do nó S para N.

Remoção

- Remover o nó interior com dois filhos:
  - Encontrar o sucessor lógico, S, do nó que será removido N.
  - Copiar a chave do nó S para N.
  - Remover o nó S da árvore.

Remoção

- Remover o nó interior com dois filhos:
  - Encontrar o sucessor lógico, S, do nó que será removido N.
  - Copiar a chave do nó S para N.
  - Remover o nó S da árvore.



Remoção

Remoção

- A dúvida que surge é:

Remoção

- A dúvida que surge é:
  - Como encontrar o sucessor de um nó e onde ele estará localizado na árvore?

Remoção

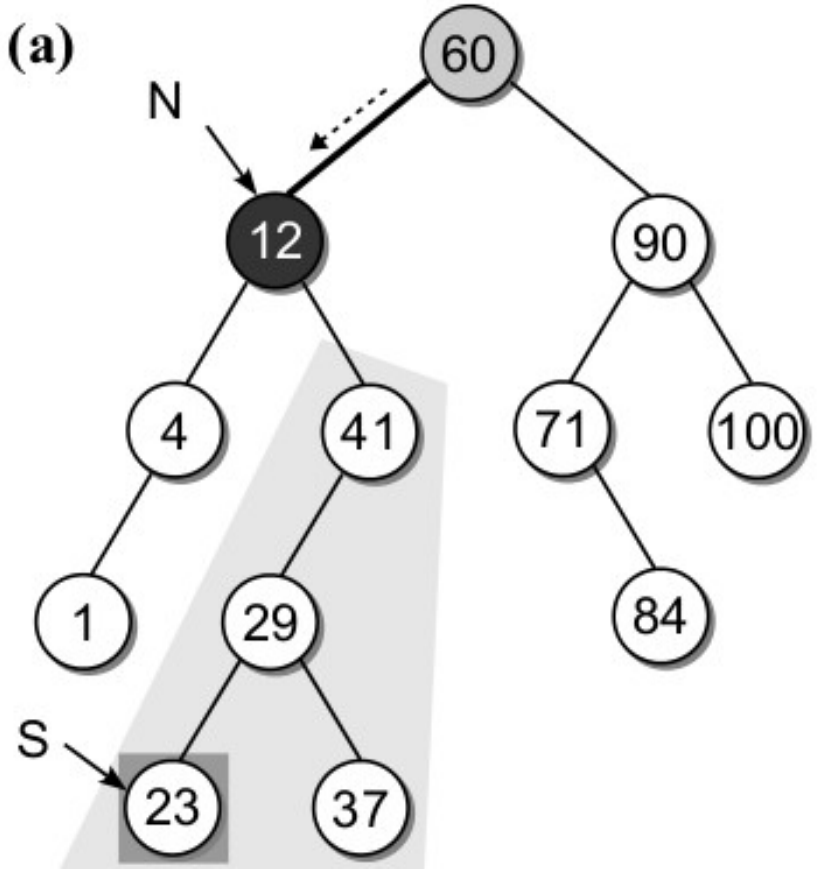
- A dúvida que surge é:
  - Como encontrar o sucessor de um nó e onde ele estará localizado na árvore?
  - O sucessor é o menor elemento dos elementos que sejam maiores que o nó que será removido.

Remoção

- A dúvida que surge é:
  - Como encontrar o sucessor de um nó e onde ele estará localizado na árvore?
  - O sucessor é o menor elemento dos elementos que sejam maiores que o nó que será removido.
  - De acordo com as propriedades de uma ABB, o sucessor de um nó é o seu pai ou algum nó que esteja a sua direita.

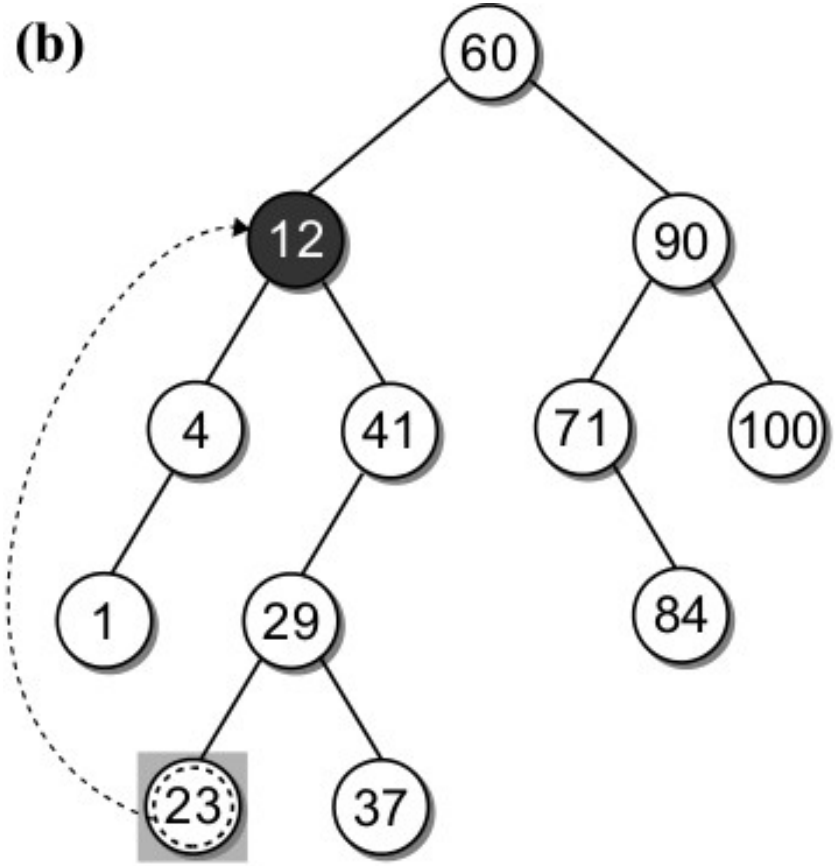
Remoção

- Passo 1
  - Achar o nó N e o seu sucessor S.



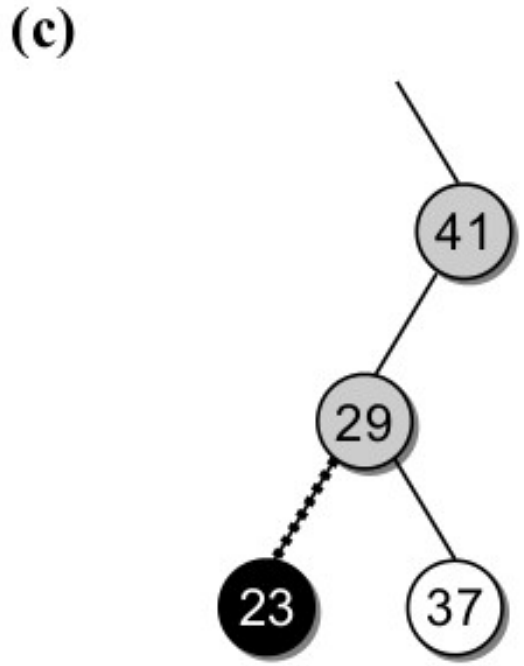
Remoção

- Passo 2
  - Copiar o nó S para N.



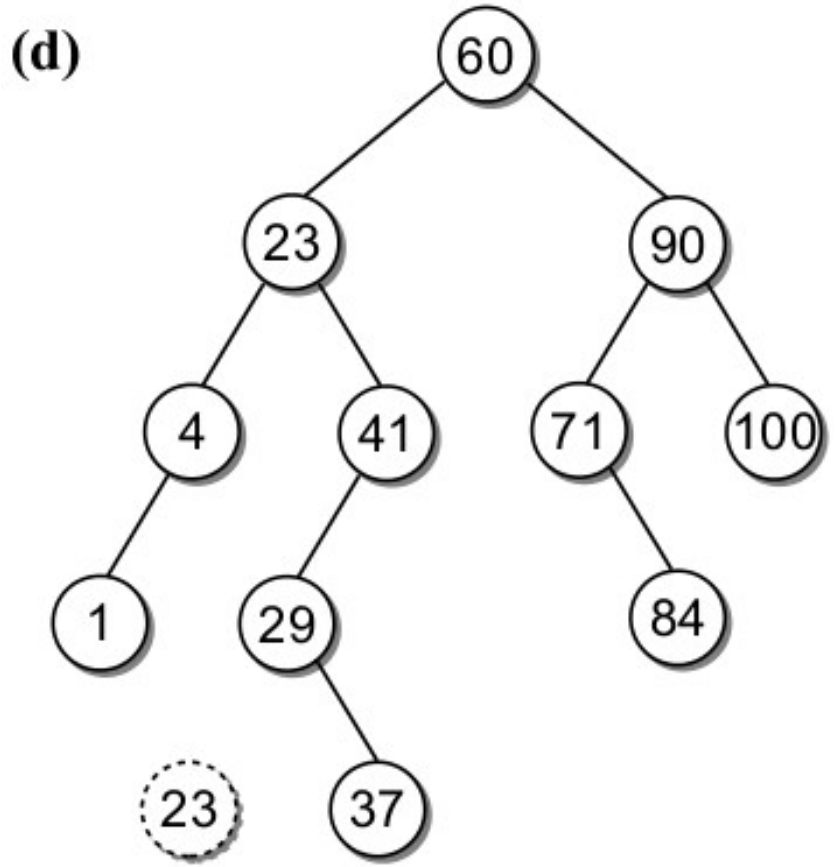
Remoção

- Passo 3
  - Remova o sucessor S que estava à direita.



Remoção

- Passo 4
  - Remova o sucessor S que estava à direita.

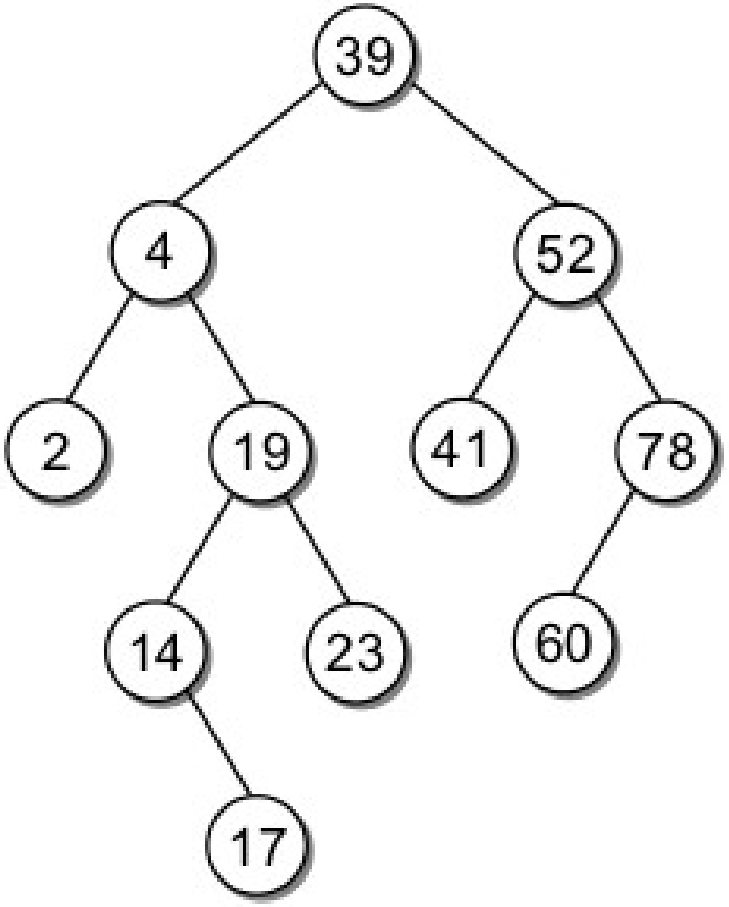


Eficiência das ABB

Operation	Worst Case
`_bstSearch(root, k)`	$O(n)$
`_bstMinimum(root)`	$O(n)$
`_bstInsert(root, k)`	$O(n)$
`_bstDelete(root, k)`	$O(n)$
traversal	$O(n)$

Exercícios

- Considere a árvore binária de busca abaixo e mostre o resultado após deletar cada uma das seguintes chaves: 14, 52 e 39.



Exercícios

- Discuta a avaliação entre uma implementação de uma lista ordenada e a implementação de uma árvore binária de busca.
- Considere que as chaves 50, 30, 70, 20, 40, 60, 80, 15, 25, 35, 45 e 36 são inseridas, nesta ordem, numa árvore de busca inicialmente vazia. Desenha a árvore resultante.
- O que aconteceria se o nó 30 fosse removido da árvore construída no exercício anterior? Desenhe essa árvore resultante.

Exercícios

- Escreva uma função em Python para verificar se uma árvore é uma ABB.
- Escreva uma função em Python para verificar se duas ABBs são iguais.

Exercícios

[ENADE 2021 – Ciência da Computação] O uso da estrutura de dados tipo Árvore Binária de Busca é uma técnica fundamental de programação. Uma árvore binária é um conjunto finito de elementos que está vazio ou é particionado em três subconjuntos, a saber: 1) raiz da árvore - elemento inicial (único), 2) subárvore da esquerda - se vista isoladamente compõe outra árvore e 3) subárvore da direita - se vista isoladamente compõe outra árvore. A árvore pode não ter qualquer elemento (árvore vazia). A definição de árvore é recursiva e, devido a isso, muitas operações sobre árvores binárias utilizam recursão. Sendo “A” a raiz de uma árvore binária e “B” a raiz de sua subárvore esquerda ou direita, é dito que “A” é pai de “B” e que “B” é filho de “A”. Um elemento sem filhos é chamado de folha. A altura da árvore é o número de elementos encontrados no caminho descendente mais longo que liga a sua raiz até uma folha.

Uma Árvore de Busca Binária é uma árvore binária especializada, na qual a informação que o elemento filho esquerdo possui é numericamente menor que a informação do elemento pai. De forma análoga, a informação que o elemento filho direito possui é numericamente maior ou igual à informação do elemento pai. O objetivo de organizar dados em Árvores Binárias de Busca é facilitar a tarefa de encontrar um determinado elemento. O percurso completo de uma árvore binária consiste em visitar todos os elementos desta árvore, segundo algum critério, a fim de processá-los. Três formas são bem conhecidas para a realização deste percurso: 1) pré-ordem, 2) em-ordem e 3) pós-ordem. A figura a seguir mostra um exemplo de árvore binária.

Exercícios

Considerando o texto e a figura apresentados e que a seguinte lista de elementos numéricos: (27, 34, 40, 18, 23, 5, 25, 36, 10, 7, -2) seja totalmente transferida para uma estrutura de Árvore Binária de Busca, inicialmente vazia, elemento a elemento, da esquerda para a direita, assinale a alternativa correta.

A) A árvore resultante terá 5 níveis de altura, com 6 elementos à esquerda da raiz principal (inicial) e 4 elementos à direita.

B) O percurso da árvore em Pré-ordem irá processar os elementos na seguinte ordem (do primeiro ao último): -2, 7, 10, 5, 25, 23, 18, 36, 40, 34, 27.

C) O percurso da árvore em Em-ordem irá processar os elementos na seguinte ordem (do primeiro ao último): -2, 5, 7, 10, 18, 23, 25, 27, 34, 36, 40.

D) O percurso da árvore em Pós-ordem irá processar os elementos na seguinte ordem (do primeiro ao último): 27, 18, 5, -2, 10, 7, 23, 25, 34, 40, 36.

E) O número máximo de elementos que essa árvore poderá ter com 10 níveis será de 1 024 elementos.

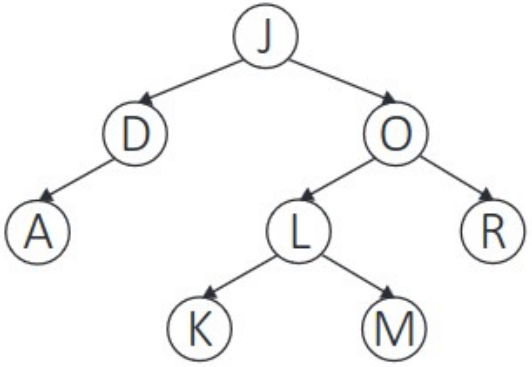


Figura – Exemplo de Árvore Binária