

Лекция 21

Двоичные деревья

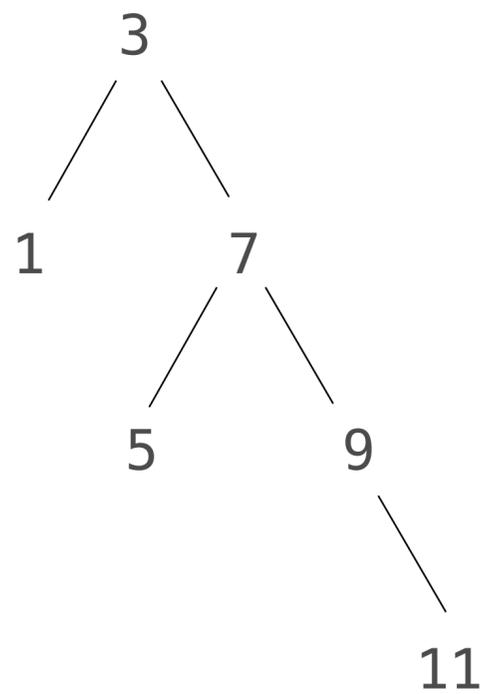
Класс Binary Tree

Класс Binary Tree

Двоичное дерево – это дерево, у которого есть левая и правая ветвь.

Класс Binary Tree

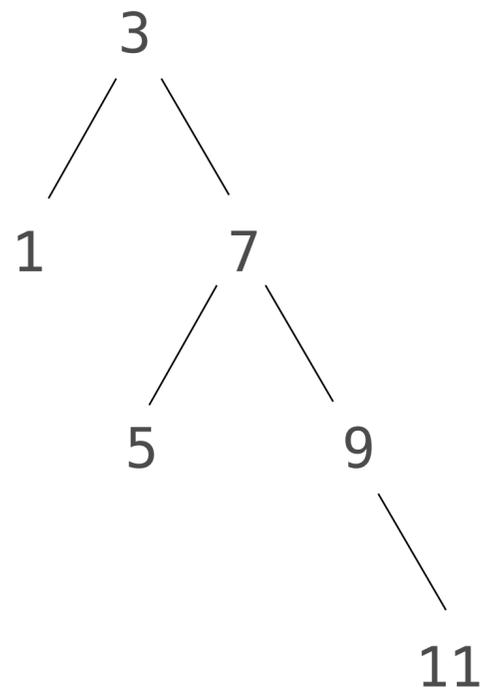
Двоичное дерево – это дерево, у которого есть левая и правая ветвь.



Класс Binary Tree

Двоичное дерево – это дерево, у которого есть левая и правая ветвь.

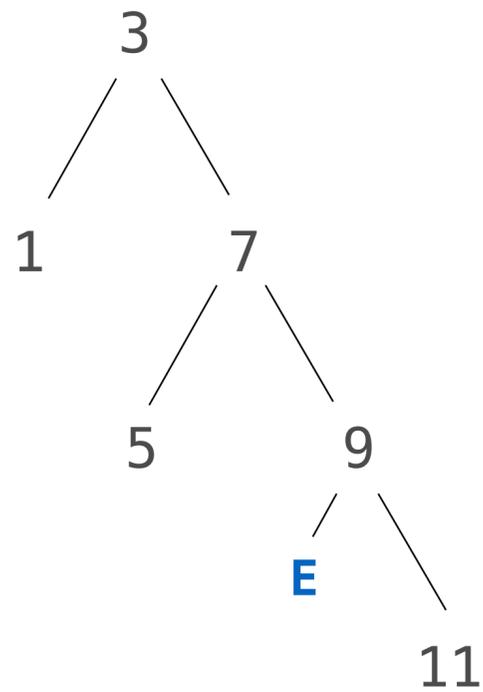
Идея: Заполнять отсутствующие ветви пустыми деревьями.



Класс Binary Tree

Двоичное дерево – это дерево, у которого есть левая и правая ветвь.

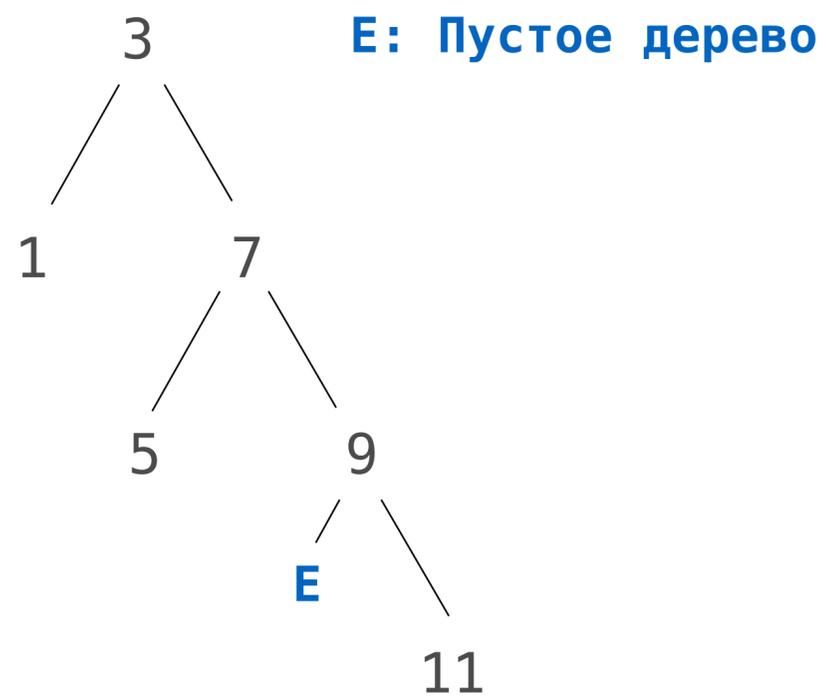
Идея: Заполнять отсутствующие ветви пустыми деревьями.



Класс Binary Tree

Двоичное дерево – это дерево, у которого есть левая и правая ветвь.

Идея: Заполнять отсутствующие ветви пустыми деревьями.

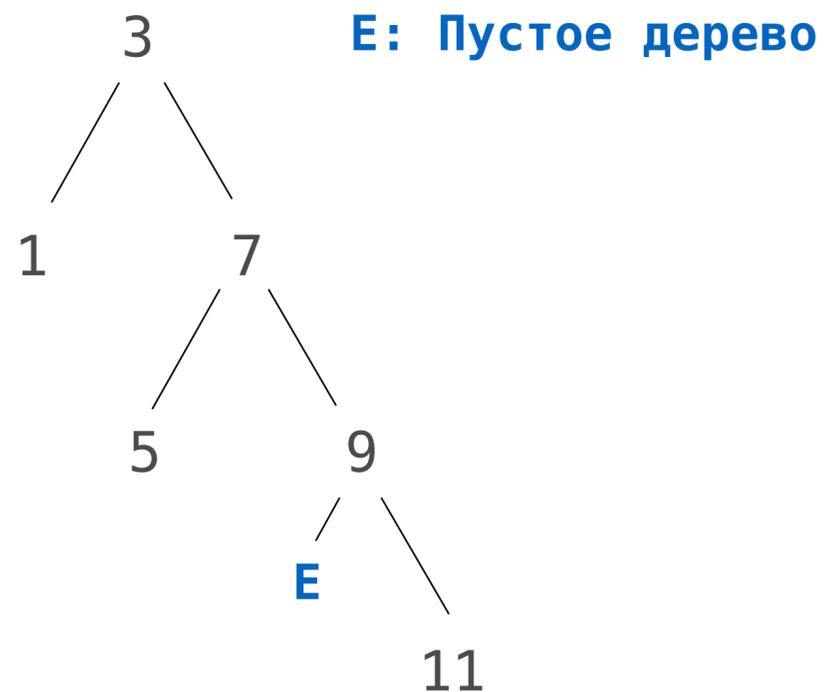


Класс Binary Tree

Двоичное дерево – это дерево, у которого есть левая и правая ветвь.

Идея: Заполнять отсутствующие ветви пустыми деревьями.

Идея 2: Экземпляр BinaryTree всегда будет иметь в точности две ветви.

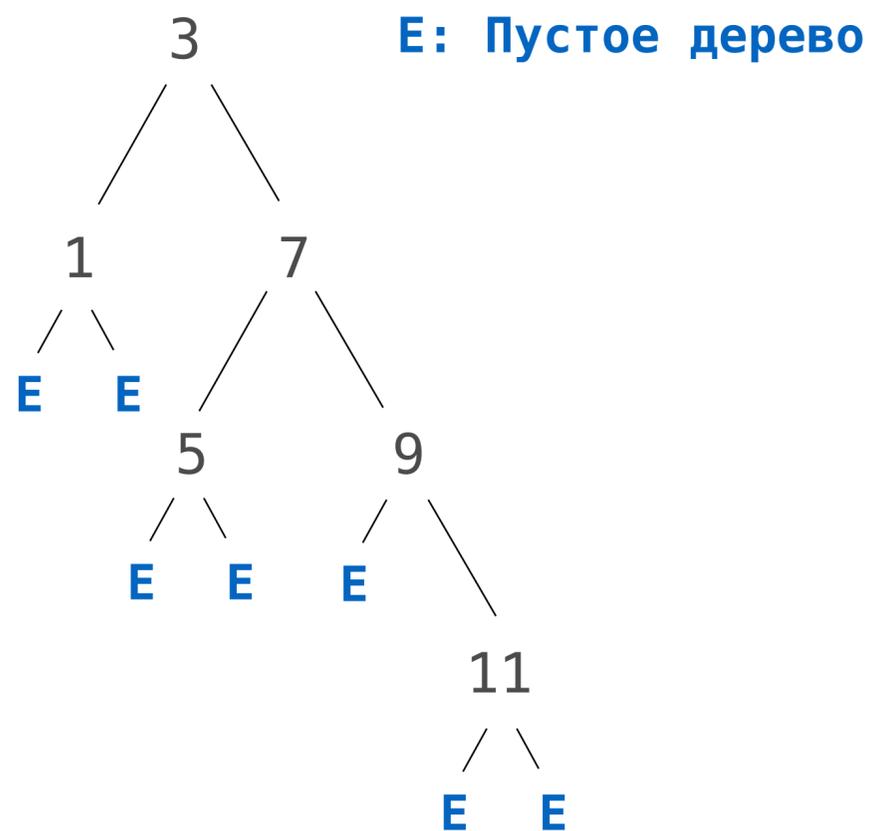


Класс Binary Tree

Двоичное дерево – это дерево, у которого есть левая и правая ветвь.

Идея: Заполнять отсутствующие ветви пустыми деревьями.

Идея 2: Экземпляр BinaryTree всегда будет иметь в точности две ветви.

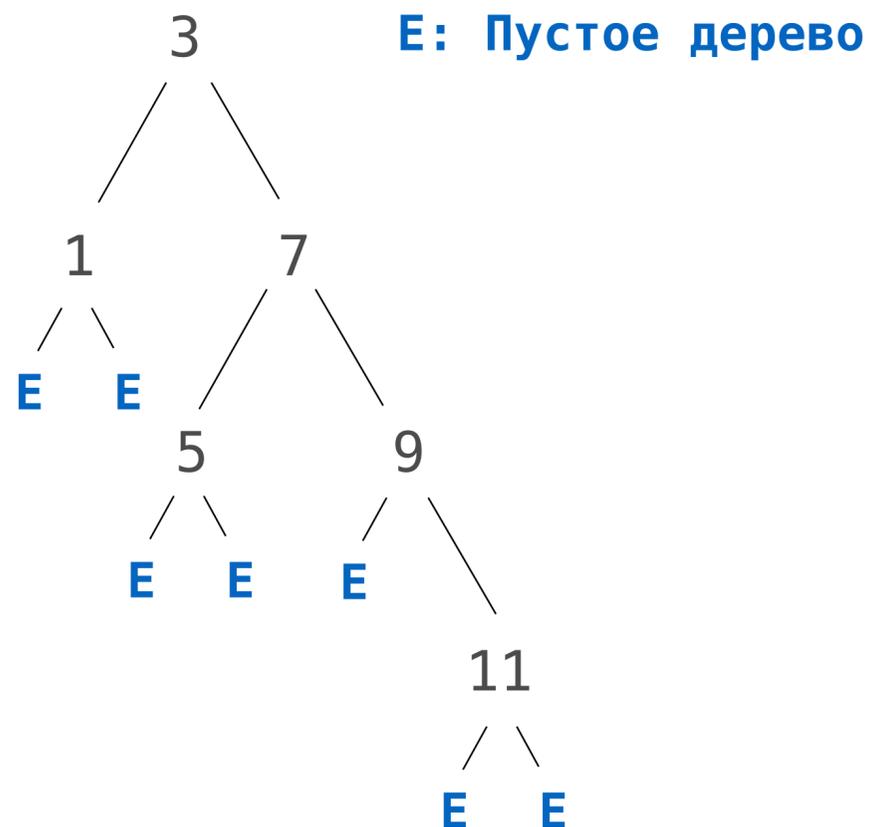


Класс Binary Tree

Двоичное дерево – это дерево, у которого есть левая и правая ветвь.

Идея: Заполнять отсутствующие ветви пустыми деревьями.

Идея 2: Экземпляр BinaryTree всегда будет иметь в точности две ветви.



```
class BinaryTree(Tree):
    empty = Tree(None)

    def __init__(self, label, left=empty, right=empty):
        Tree.__init__(self, label, [left, right])

    @property
    def left(self):
        return self.branches[0]

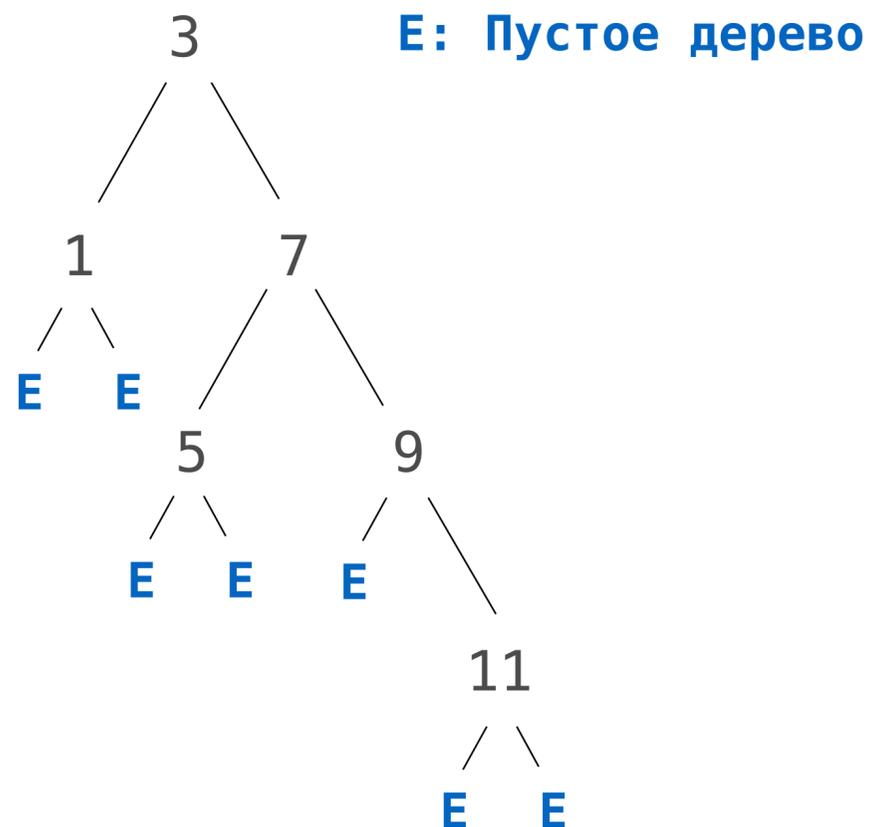
    @property
    def right(self):
        return self.branches[1]
```

Класс Binary Tree

Двоичное дерево – это дерево, у которого есть левая и правая ветвь.

Идея: Заполнять отсутствующие ветви пустыми деревьями.

Идея 2: Экземпляр BinaryTree всегда будет иметь в точности две ветви.



```
class BinaryTree(Tree):  
    empty = Tree(None)
```

```
def __init__(self, label, left=empty, right=empty):  
    Tree.__init__(self, label, [left, right])
```

```
@property  
def left(self):  
    return self.branches[0]
```

```
@property  
def right(self):  
    return self.branches[1]
```

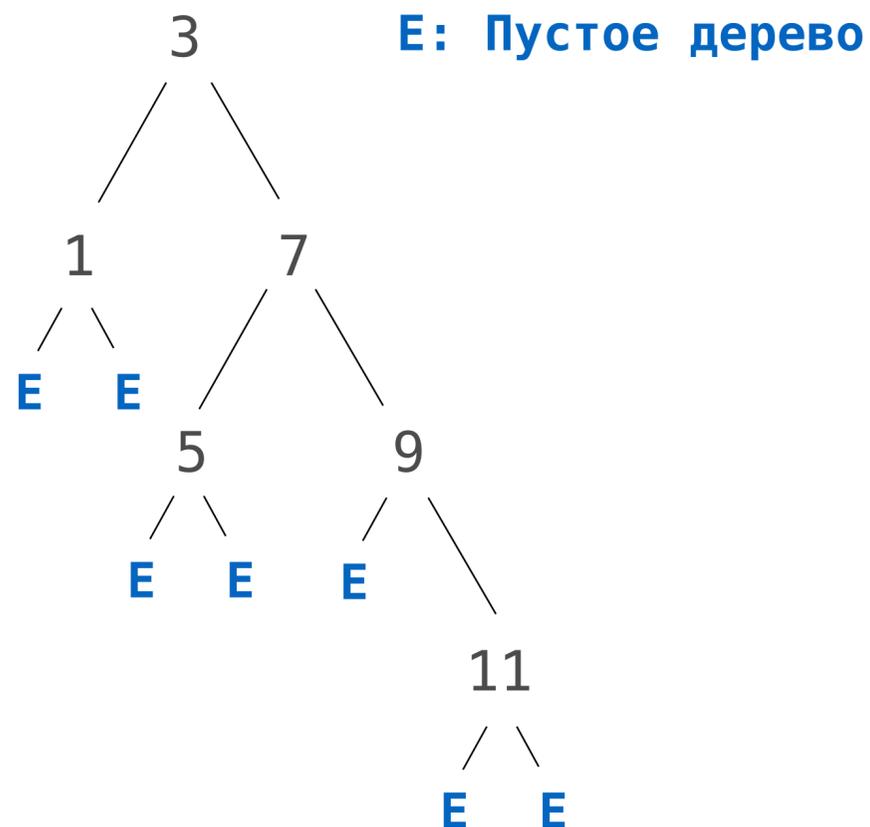
```
Bin = BinaryTree
```

Класс Binary Tree

Двоичное дерево – это дерево, у которого есть левая и правая ветвь.

Идея: Заполнять отсутствующие ветви пустыми деревьями.

Идея 2: Экземпляр BinaryTree всегда будет иметь в точности две ветви.



```
class BinaryTree(Tree):
    empty = Tree(None)
```

```
def __init__(self, label, left=empty, right=empty):
    Tree.__init__(self, label, [left, right])
```

```
@property
def left(self):
    return self.branches[0]
```

```
@property
def right(self):
    return self.branches[1]
```

```
Bin = BinaryTree
t = Bin(3, Bin(1),
        Bin(7, Bin(5),
            Bin(9, Bin.empty,
                Bin(11))))
```

Двоичные деревья поиска

Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]

Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]



Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]

Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]

Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



False

Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



False

4 в [1, 2, 4, 8, 16, 32]

Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



False

4 в [1, 2, 4, 8, 16, 32]



Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



False

4 в [1, 2, 4, 8, 16, 32]



[1, 2, 4, 8, 16, 32]

Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



False

4 в [1, 2, 4, 8, 16, 32]



[1, 2, 4, 8, 16, 32]



Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



False

4 в [1, 2, 4, 8, 16, 32]



[1, 2, 4, 8, 16, 32]



[1, 2, 4, 8, 16, 32]

Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



False

4 в [1, 2, 4, 8, 16, 32]



[1, 2, 4, 8, 16, 32]



[1, 2, 4, 8, 16, 32]



Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



False

4 в [1, 2, 4, 8, 16, 32]



[1, 2, 4, 8, 16, 32]



[1, 2, 4, 8, 16, 32]



True

Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



False

4 в [1, 2, 4, 8, 16, 32]



[1, 2, 4, 8, 16, 32]



[1, 2, 4, 8, 16, 32]



True

Для упорядоченного списка длины n , каков будет временной порядок роста?

Двоичный поиск

Стратегия поиска значения в упорядоченном списке: проверка среднего значения и выбор первой или второй половины для дальнейшего поиска

20 в [1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



[1, 2, 4, 8, 16, 32, 64]



False

4 в [1, 2, 4, 8, 16, 32]



[1, 2, 4, 8, 16, 32]



[1, 2, 4, 8, 16, 32]



True

Для упорядоченного списка длины n , каков будет временной порядок роста? $\Theta(\log n)$

Двоичные деревья поиска

Двоичные деревья поиска

Двоичное дерево поиска – двоичное дерево, в котором значение узла:

Двоичные деревья поиска

Двоичное дерево поиска – двоичное дерево, в котором значение узла:

— Больше всех узловых значений левой ветви

Двоичные деревья поиска

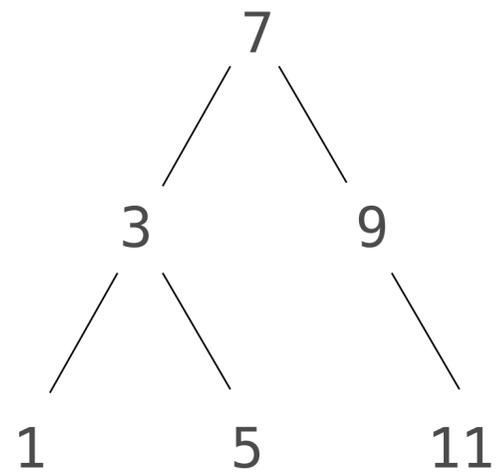
Двоичное дерево поиска – двоичное дерево, в котором значение узла:

- Больше всех узловых значений левой ветви
- Меньше всех значений правой ветви

Двоичные деревья поиска

Двоичное дерево поиска – двоичное дерево, в котором значение узла:

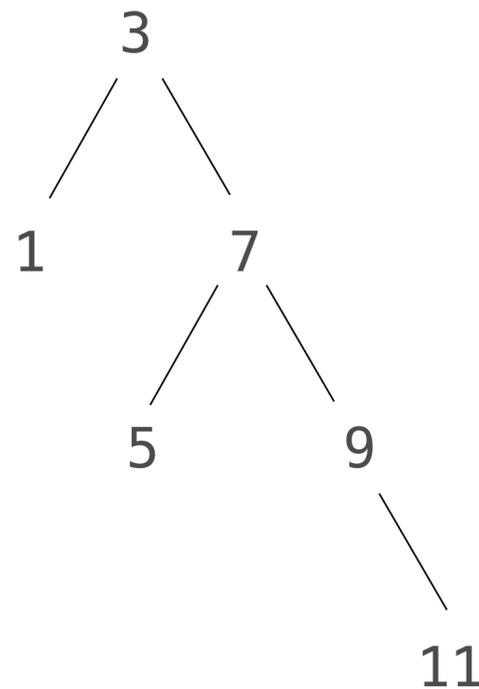
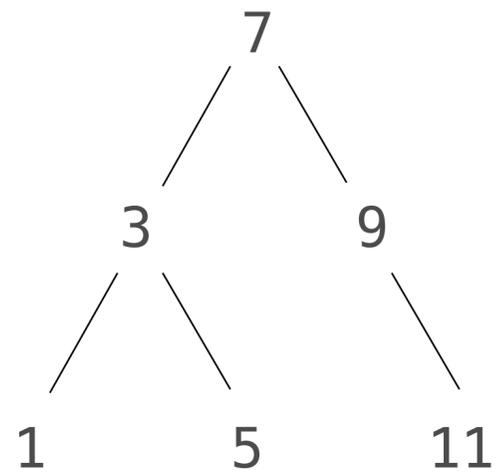
- Больше всех узловых значений левой ветви
- Менее всех значений правой ветви



Двоичные деревья поиска

Двоичное дерево поиска – двоичное дерево, в котором значение узла:

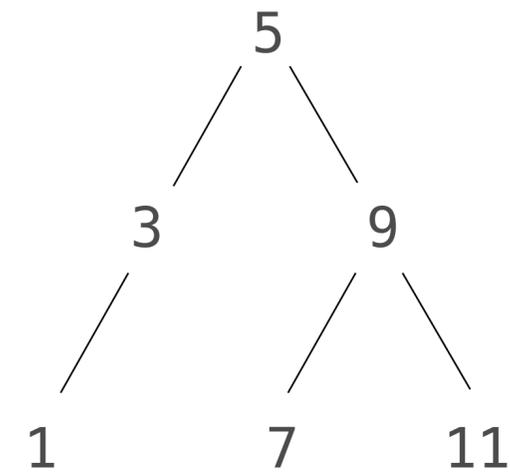
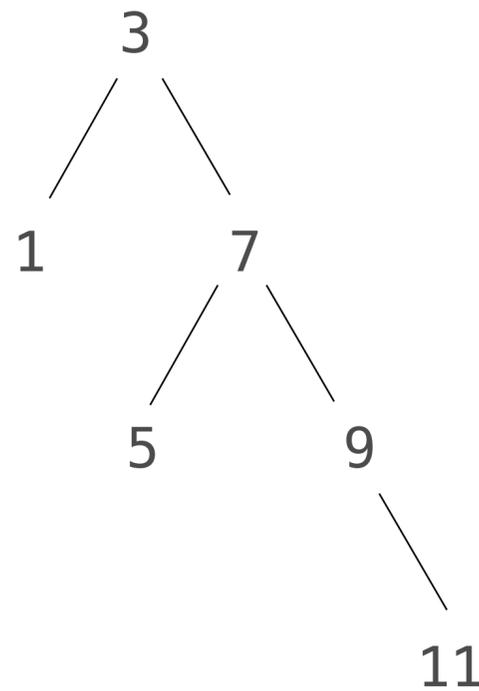
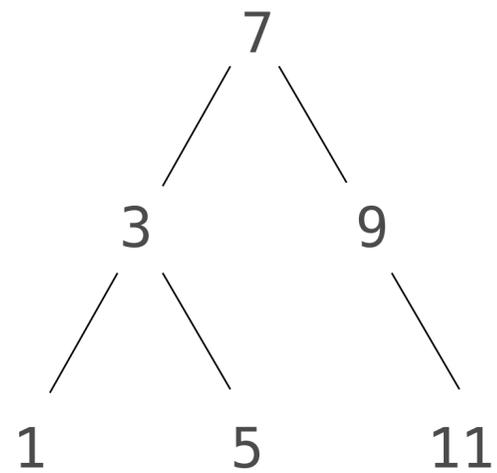
- Больше всех узловых значений левой ветви
- Менее всех значений правой ветви



Двоичные деревья поиска

Двоичное дерево поиска – двоичное дерево, в котором значение узла:

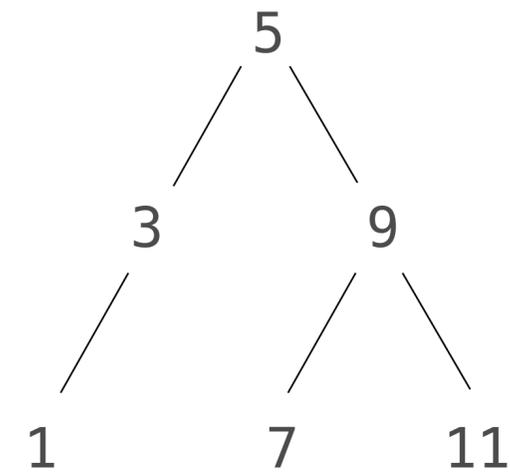
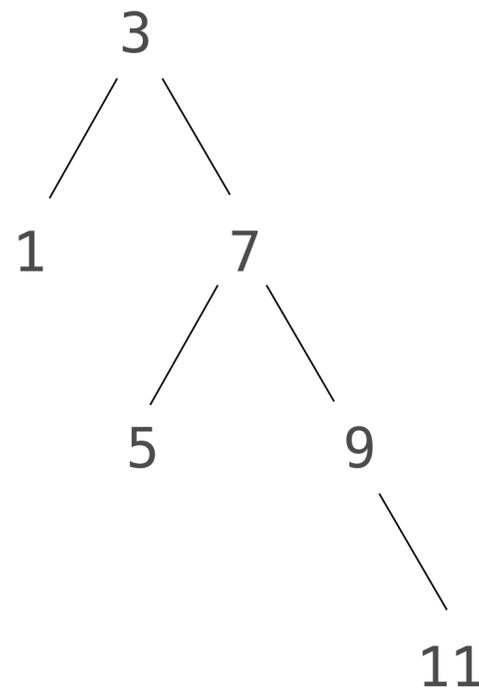
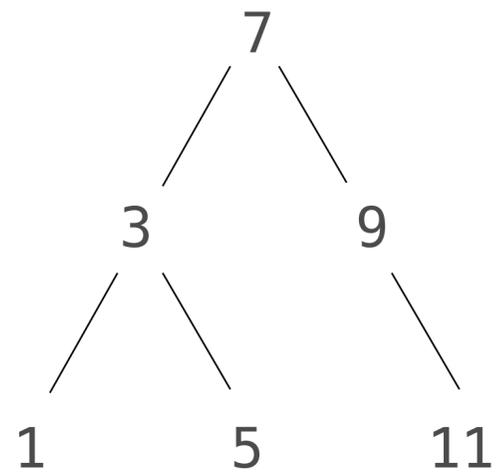
- Больше всех узловых значений левой ветви
- Менее всех значений правой ветви



Двоичные деревья поиска

Двоичное дерево поиска – двоичное дерево, в котором значение узла:

- Больше всех узловых значений левой ветви
- Менее всех значений правой ветви



(Пример)

Вопрос

Каков наибольший
элемент в BST?

Вопрос

Каков наибольший
элемент в BST?

```
def largest(t):  
    if _____:  
        return _____  
    else:  
        return _____
```

Вопрос

Каков наибольший элемент в BST?

```
def largest(t):  
    if t.right is BTree.empty :  
        return _____  
    else:  
        return _____
```

Вопрос

Каков наибольший элемент в BST?

```
def largest(t):  
    if t.right is BTree.empty :  
        return t.label  
    else:  
        return largest(t.right)
```

Вопрос

Каков наибольший элемент в BST?

```
def largest(t):  
    if t.right is BTree.empty :  
        return t.label  
    else:  
        return largest(t.right)
```

Вопрос

Каков наибольший элемент в BST?

Какой элемент в BST второй по величине?

```
def largest(t):  
    if t.right is BTree.empty :  
        return t.label  
    else:  
        return largest(t.right)
```

Вопрос

Каков наибольший элемент в BST?

```
def largest(t):  
    if t.right is BTree.empty :  
        return t.label  
    else:  
        return largest(t.right)
```

Какой элемент в BST второй по величине?

```
def second(t):  
    if t.is_leaf():  
        return None  
    elif _____ :  
        return _____  
    elif _____ :  
        return _____  
    return _____
```

Вопрос

Каков наибольший элемент в BST?

```
def largest(t):  
    if t.right is BTree.empty :  
        return t.label  
    else:  
        return largest(t.right)
```

Какой элемент в BST второй по величине?

```
def second(t):  
    if t.is_leaf():  
        return None  
    elif t.right is BTree.empty :  
        return _____  
    elif _____ :  
        return _____  
    return _____
```

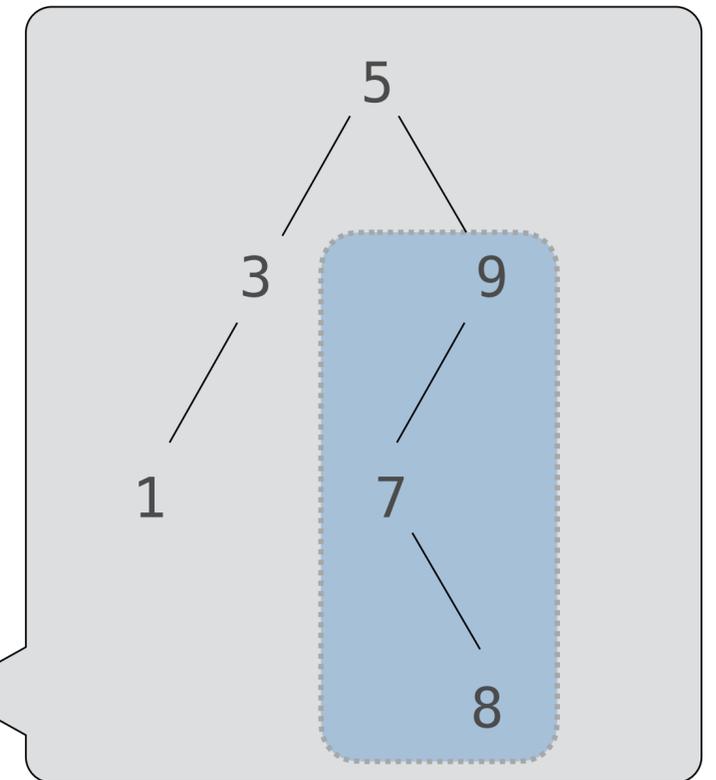
Вопрос

Каков наибольший элемент в BST?

```
def largest(t):  
    if t.right is BTree.empty :  
        return t.label  
    else:  
        return largest(t.right)
```

Какой элемент в BST второй по величине?

```
def second(t):  
    if t.is_leaf():  
        return None  
    elif t.right is BTree.empty :  
        return _____  
    elif _____ :  
        return _____  
    return _____
```



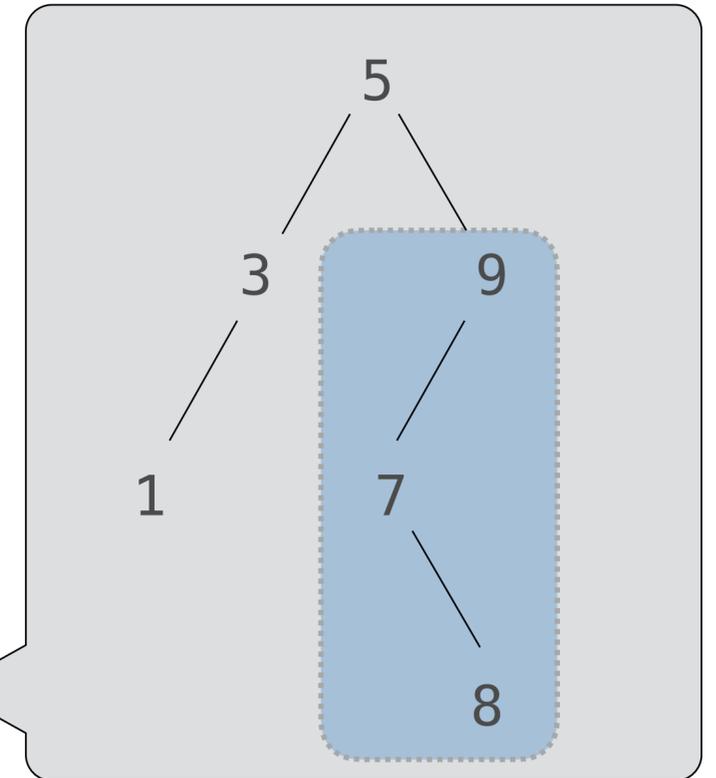
Вопрос

Каков наибольший элемент в BST?

```
def largest(t):  
    if t.right is BTree.empty :  
        return t.label  
    else:  
        return largest(t.right)
```

Какой элемент в BST второй по величине?

```
def second(t):  
    if t.is_leaf():  
        return None  
    elif t.right is BTree.empty :  
        return largest(t.left)  
    elif _____ :  
        return _____  
    return _____
```



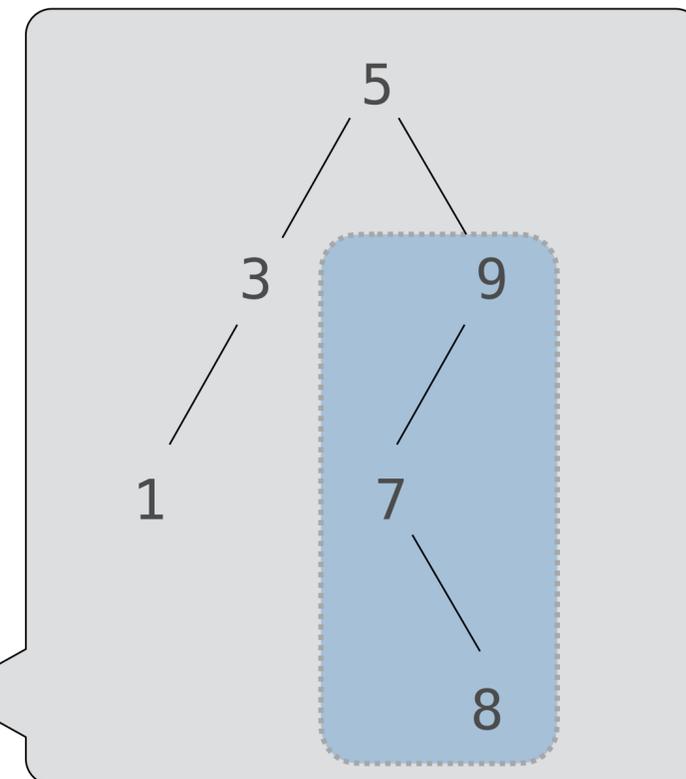
Вопрос

Каков наибольший элемент в BST?

```
def largest(t):  
    if t.right is BTree.empty :  
        return t.label  
    else:  
        return largest(t.right)
```

Какой элемент в BST второй по величине?

```
def second(t):  
    if t.is_leaf():  
        return None  
    elif t.right is BTree.empty :  
        return largest(t.left)  
    elif t.right.is_leaf() :  
        return _____  
    return _____
```



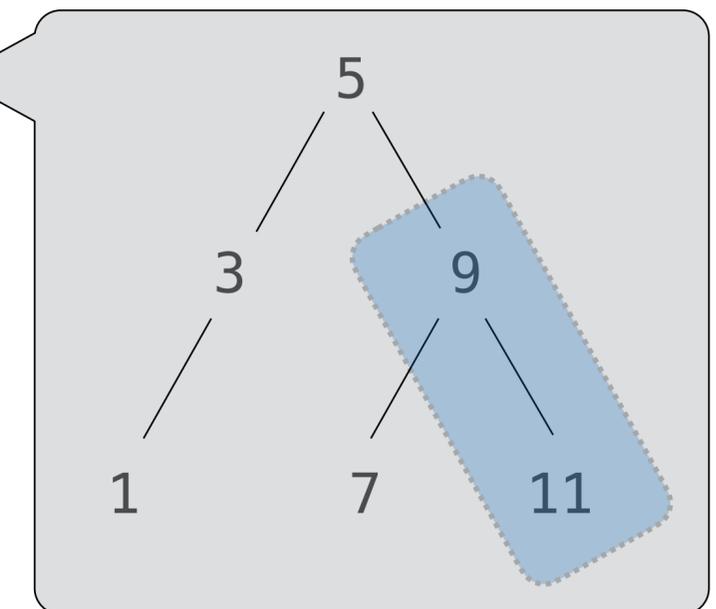
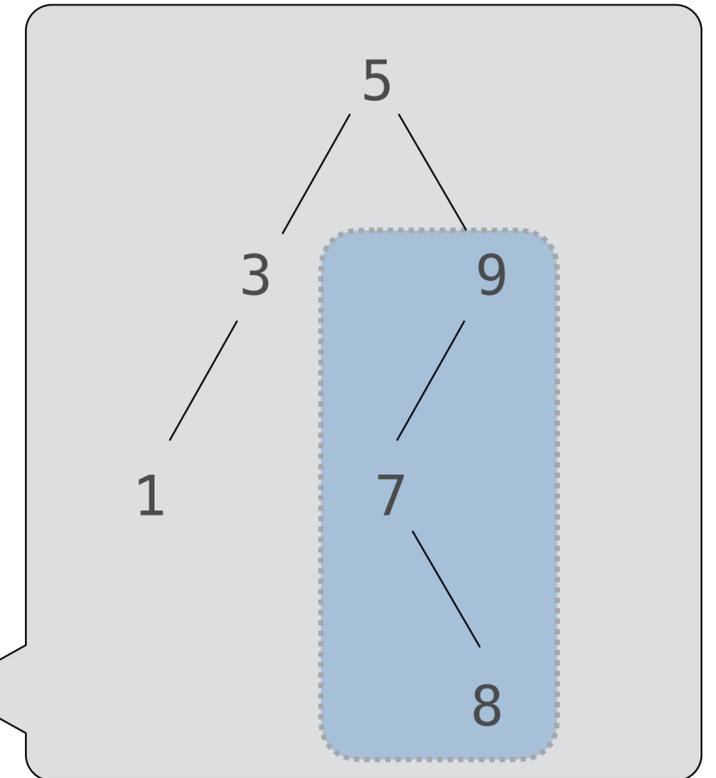
Вопрос

Каков наибольший элемент в BST?

```
def largest(t):  
    if t.right is BTree.empty :  
        return t.label  
    else:  
        return largest(t.right)
```

Какой элемент в BST второй по величине?

```
def second(t):  
    if t.is_leaf():  
        return None  
    elif t.right is BTree.empty :  
        return largest(t.left)  
    elif t.right.is_leaf() :  
        return t.label  
    return _____
```



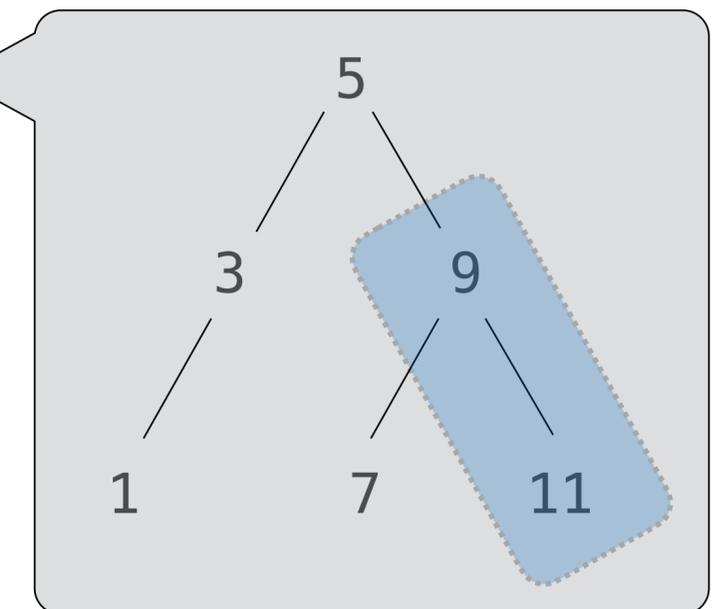
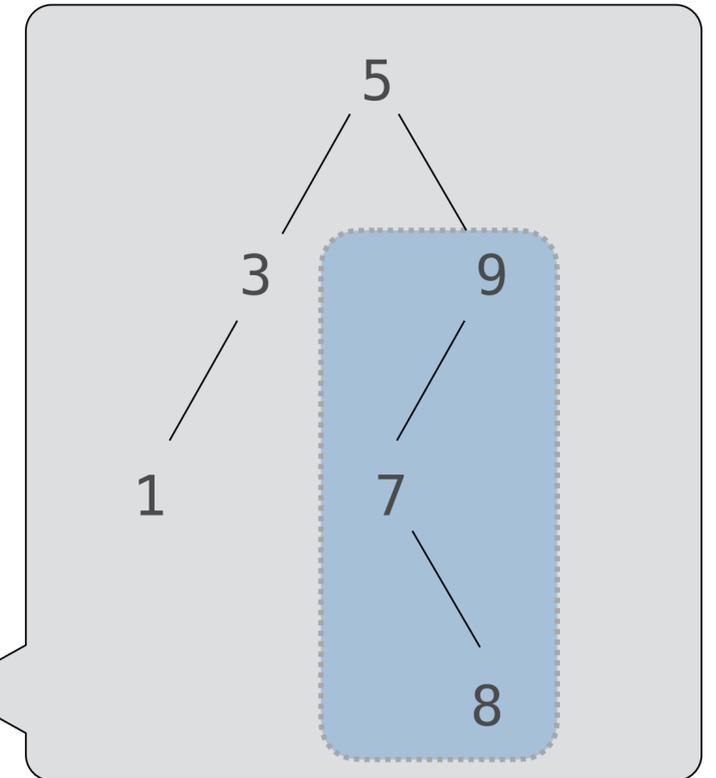
Вопрос

Каков наибольший элемент в BST?

```
def largest(t):  
    if t.right is BTree.empty :  
        return t.label  
    else:  
        return largest(t.right)
```

Какой элемент в BST второй по величине?

```
def second(t):  
    if t.is_leaf():  
        return None  
    elif t.right is BTree.empty :  
        return largest(t.left)  
    elif t.right.is_leaf() :  
        return t.label  
    return second(t.right)
```



Множества на двоичных деревьях поиска

Поиск в двоичном дереве поиска

Поиск в двоичном дереве поиска

`contains` обходит дерево

Поиск в двоичном дереве поиска

`contains` обходит дерево

- Если искомое значение v не узловое значение, то оно должно быть либо в правой, либо в левой ветви.

Поиск в двоичном дереве поиска

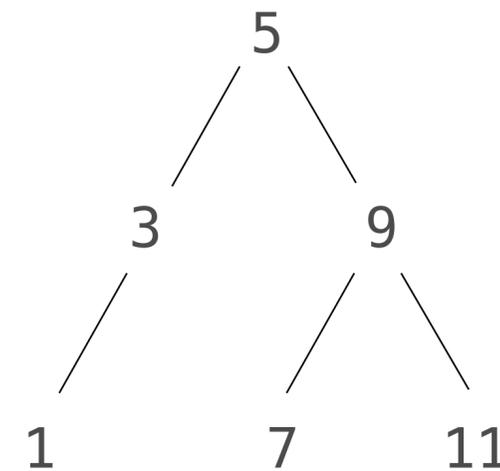
`contains` обходит дерево

- Если искомое значение v не узловое значение, то оно должно быть либо в правой, либо в левой ветви.
- Выбор одну ветви приводит к сокращению пространства поиска на размер второй ветви каждый рекурсивный вызов.

Поиск в двоичном дереве поиска

`contains` обходит дерево

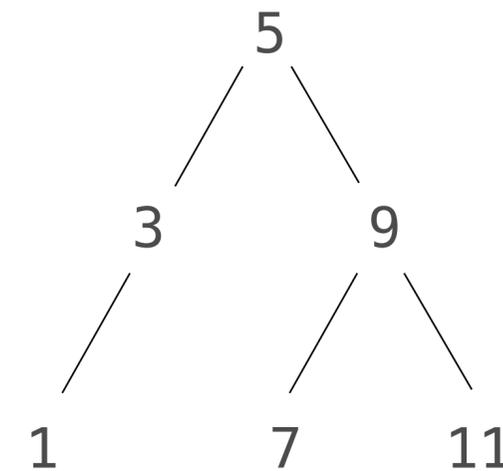
- Если искомое значение v не узловое значение, то оно должно быть либо в правой, либо в левой ветви.
- Выбор одну ветви приводит к сокращению пространства поиска на размер второй ветви каждый рекурсивный вызов.



Поиск в двоичном дереве поиска

`contains` обходит дерево

- Если искомое значение v не узловое значение, то оно должно быть либо в правой, либо в левой ветви.
- Выбор одну ветви приводит к сокращению пространства поиска на размер второй ветви каждый рекурсивный вызов.



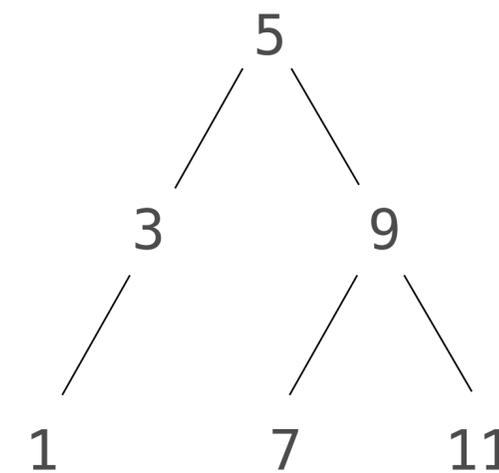
9

Поиск в двоичном дереве поиска

`contains` обходит дерево

- Если искомое значение v не узловое значение, то оно должно быть либо в правой, либо в левой ветви.
- Выбор одну ветви приводит к сокращению пространства поиска на размер второй ветви каждый рекурсивный вызов.

```
def contains(s, v):
```



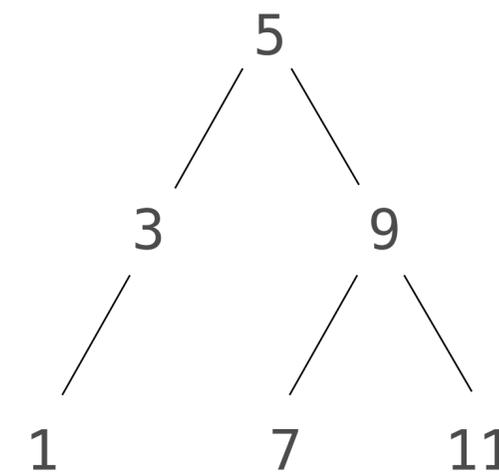
9

Поиск в двоичном дереве поиска

`contains` обходит дерево

- Если искомое значение v не узловое значение, то оно должно быть либо в правой, либо в левой ветви.
- Выбор одну ветви приводит к сокращению пространства поиска на размер второй ветви каждый рекурсивный вызов.

```
def contains(s, v):  
    if s is BTree.empty:  
        return False
```



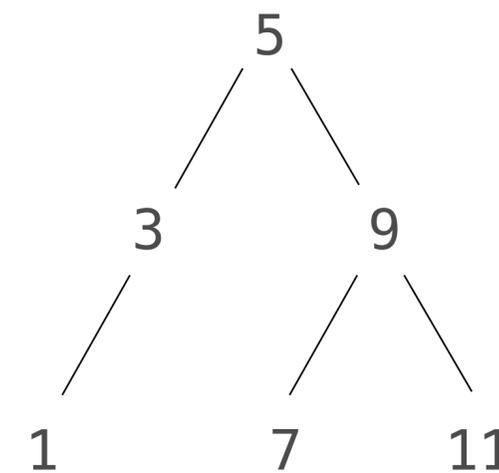
9

Поиск в двоичном дереве поиска

`contains` обходит дерево

- Если искомое значение v не узловое значение, то оно должно быть либо в правой, либо в левой ветви.
- Выбор одну ветви приводит к сокращению пространства поиска на размер второй ветви каждый рекурсивный вызов.

```
def contains(s, v):  
    if s is BTree.empty:  
        return False  
    elif s.label == v:  
        return True
```



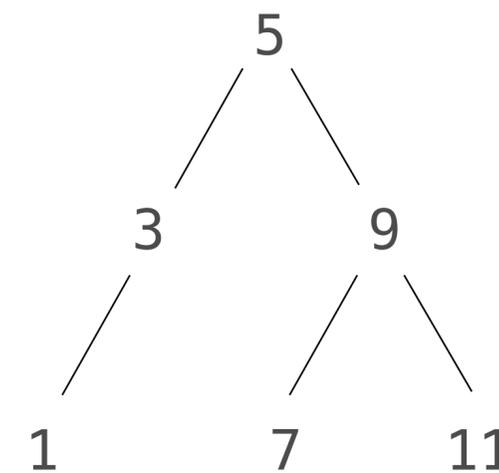
9

Поиск в двоичном дереве поиска

`contains` обходит дерево

- Если искомое значение v не узловое значение, то оно должно быть либо в правой, либо в левой ветви.
- Выбор одну ветви приводит к сокращению пространства поиска на размер второй ветви каждый рекурсивный вызов.

```
def contains(s, v):  
    if s is BTree.empty:  
        return False  
    elif s.label == v:  
        return True  
    elif s.label < v:  
        return set_contains(s.right, v)
```



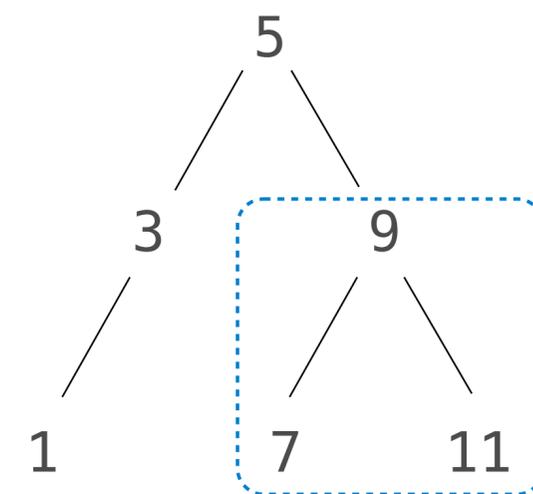
9

Поиск в двоичном дереве поиска

`contains` обходит дерево

- Если искомое значение v не узловое значение, то оно должно быть либо в правой, либо в левой ветви.
- Выбор одну ветви приводит к сокращению пространства поиска на размер второй ветви каждый рекурсивный вызов.

```
def contains(s, v):  
    if s is BTree.empty:  
        return False  
    elif s.label == v:  
        return True  
    elif s.label < v:  
        return set_contains(s.right, v)
```



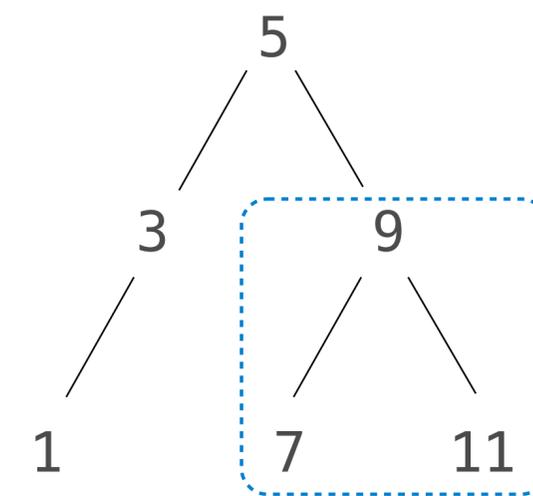
Если 9 входит в множество, надо искать в этой ветви

Поиск в двоичном дереве поиска

`contains` обходит дерево

- Если искомое значение v не узловое значение, то оно должно быть либо в правой, либо в левой ветви.
- Выбор одну ветви приводит к сокращению пространства поиска на размер второй ветви каждый рекурсивный вызов.

```
def contains(s, v):  
    if s is BTree.empty:  
        return False  
    elif s.label == v:  
        return True  
    elif s.label < v:  
        return set_contains(s.right, v)  
    elif s.label > v:  
        return set_contains(s.left, v)
```



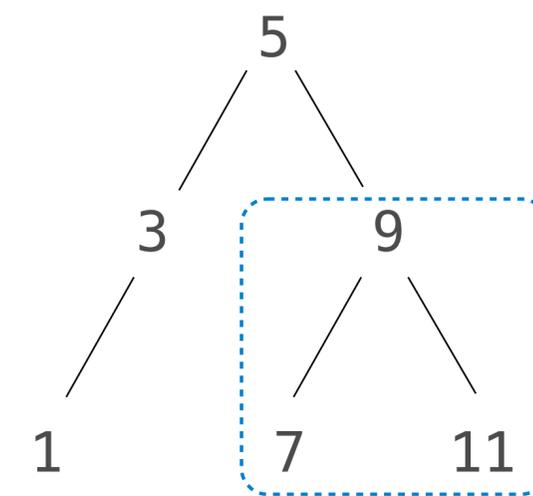
Если 9 входит в множество, надо искать в этой ветви

Поиск в двоичном дереве поиска

`contains` обходит дерево

- Если искомое значение v не узловое значение, то оно должно быть либо в правой, либо в левой ветви.
- Выбор одну ветви приводит к сокращению пространства поиска на размер второй ветви каждый рекурсивный вызов.

```
def contains(s, v):  
    if s is BTree.empty:  
        return False  
    elif s.label == v:  
        return True  
    elif s.label < v:  
        return set_contains(s.right, v)  
    elif s.label > v:  
        return set_contains(s.left, v)
```



Если 9 входит в множество, надо искать в этой ветви

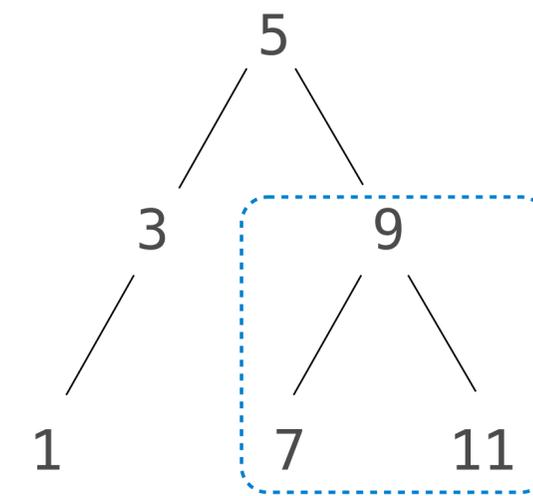
Порядок роста?

Поиск в двоичном дереве поиска

`contains` обходит дерево

- Если искомое значение v не узловое значение, то оно должно быть либо в правой, либо в левой ветви.
- Выбор одну ветви приводит к сокращению пространства поиска на размер второй ветви каждый рекурсивный вызов.

```
def contains(s, v):  
    if s is BTree.empty:  
        return False  
    elif s.label == v:  
        return True  
    elif s.label < v:  
        return set_contains(s.right, v)  
    elif s.label > v:  
        return set_contains(s.left, v)
```



Если 9 входит в множество, надо искать в этой ветви

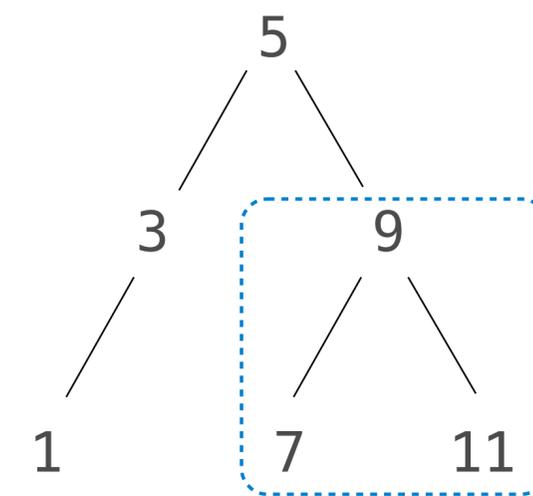
Порядок роста? $\Theta(h)$ в среднем

Поиск в двоичном дереве поиска

`contains` обходит дерево

- Если искомое значение v не узловое значение, то оно должно быть либо в правой, либо в левой ветви.
- Выбор одну ветви приводит к сокращению пространства поиска на размер второй ветви каждый рекурсивный вызов.

```
def contains(s, v):  
    if s is BTree.empty:  
        return False  
    elif s.label == v:  
        return True  
    elif s.label < v:  
        return set_contains(s.right, v)  
    elif s.label > v:  
        return set_contains(s.left, v)
```



Если 9 входит в множество, надо искать в этой ветви

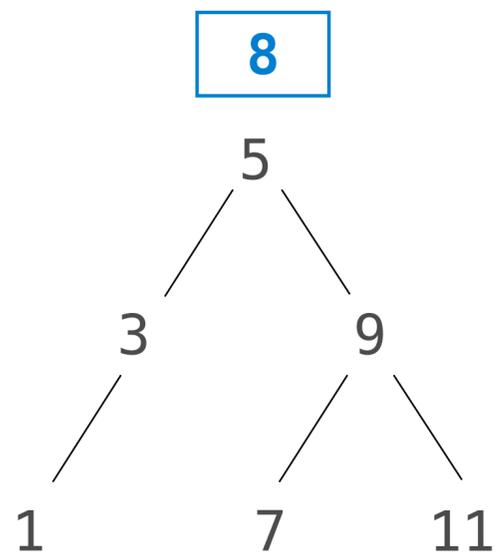
Порядок роста?

$\Theta(h)$ в среднем

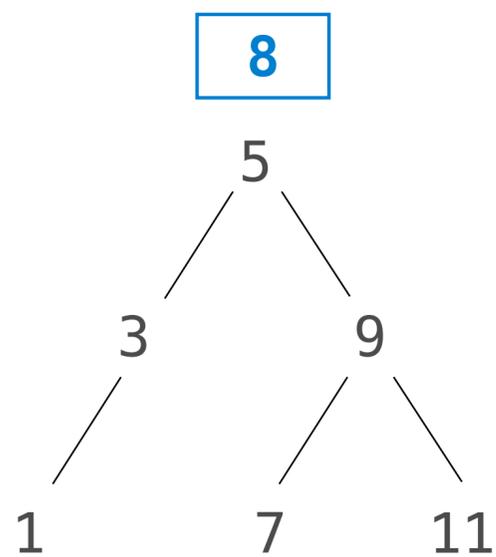
$\Theta(\log n)$ в среднем, в сбалансированном дереве

Добавление элемента в множество (BST)

Добавление элемента в множество (BST)

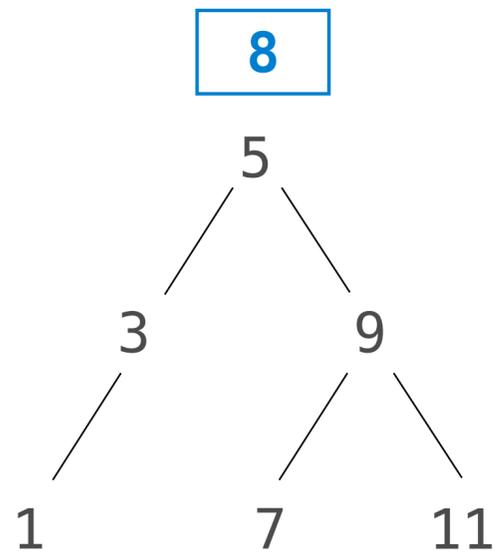


Добавление элемента в множество (BST)



Right!

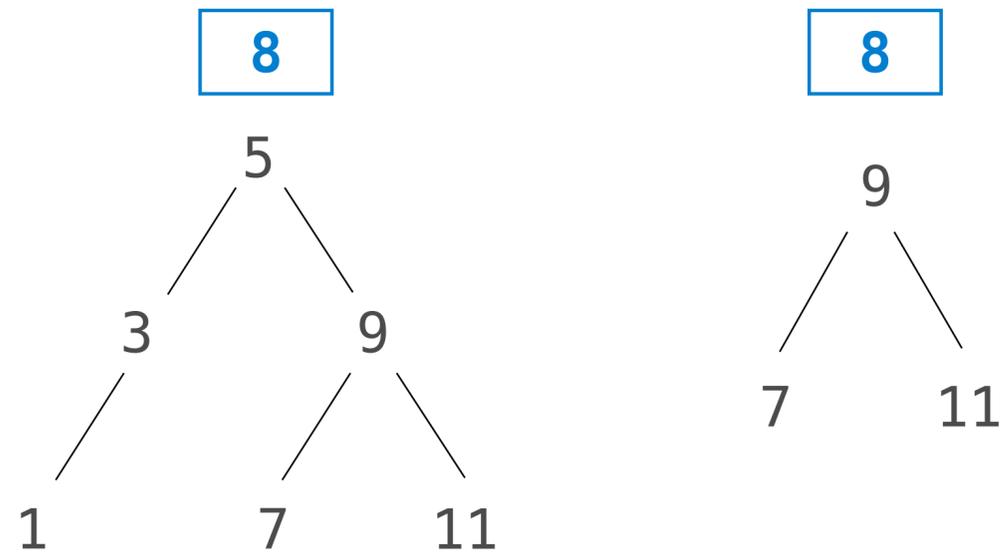
Добавление элемента в множество (BST)



Right!



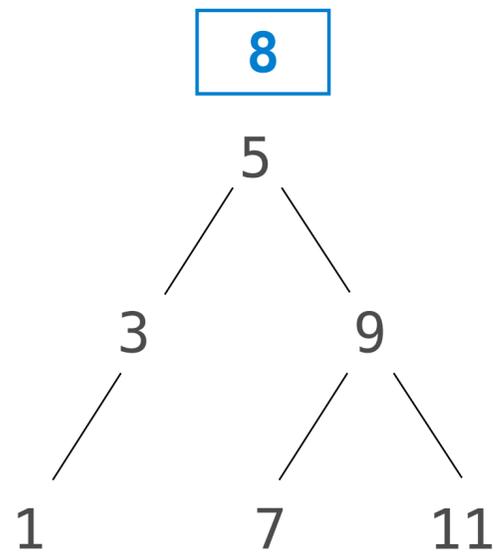
Добавление элемента в множество (BST)



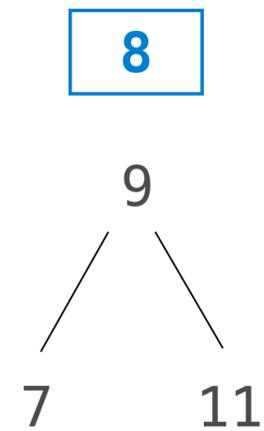
Right!



Добавление элемента в множество (BST)



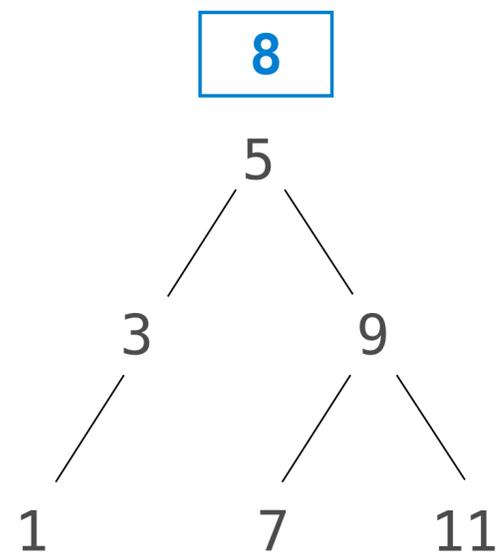
Right!



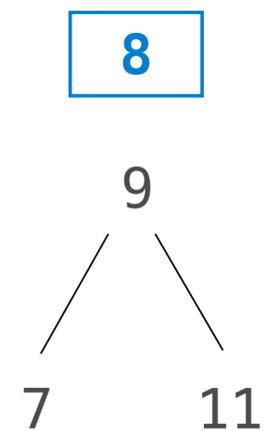
Left!



Добавление элемента в множество (BST)



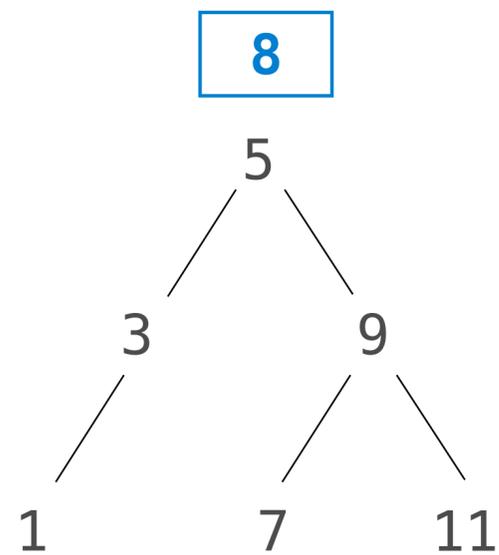
Right!



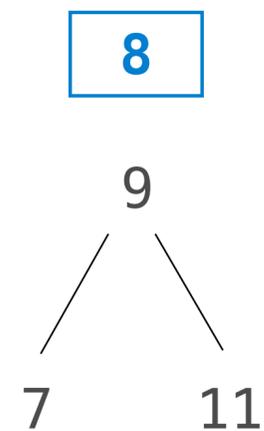
Left!



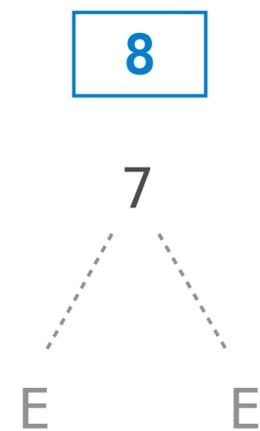
Добавление элемента в множество (BST)



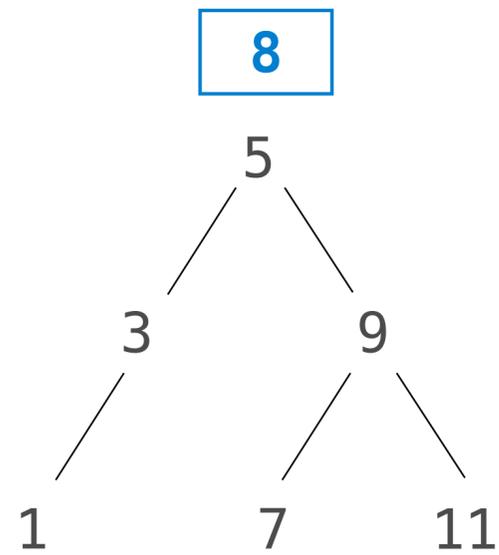
Right!



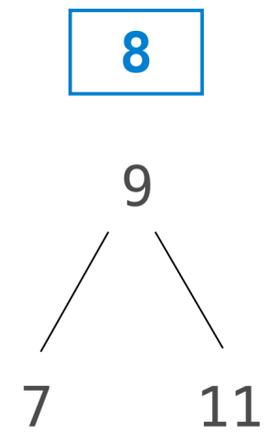
Left!



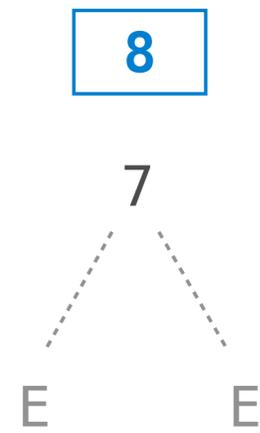
Добавление элемента в множество (BST)



Right!



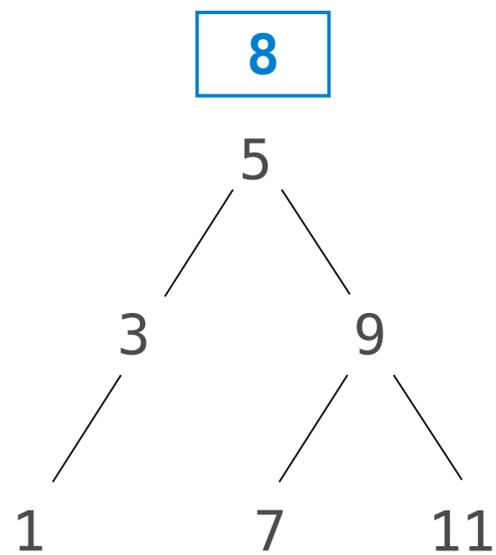
Left!



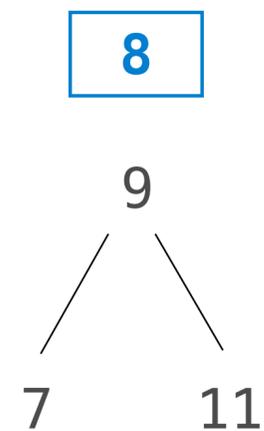
Right!



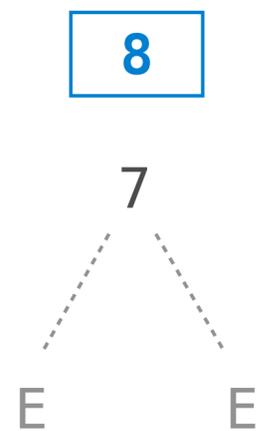
Добавление элемента в множество (BST)



Right!



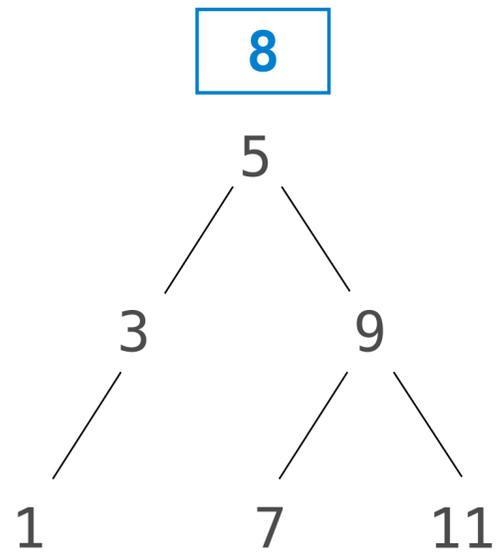
Left!



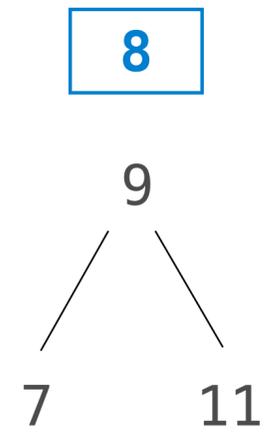
Right!



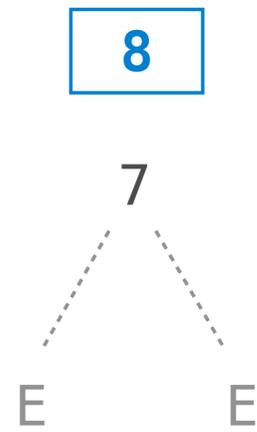
Добавление элемента в множество (BST)



Right!



Left!



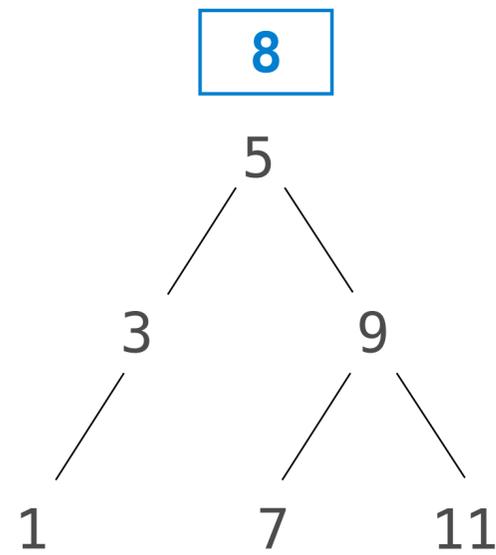
Right!



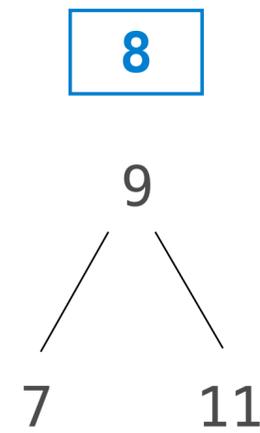
Stop!



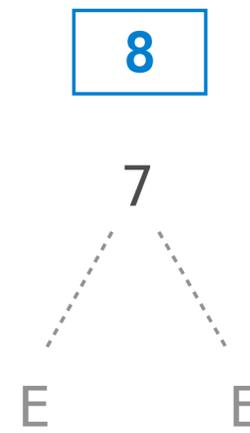
Добавление элемента в множество (BST)



Right!



Left!



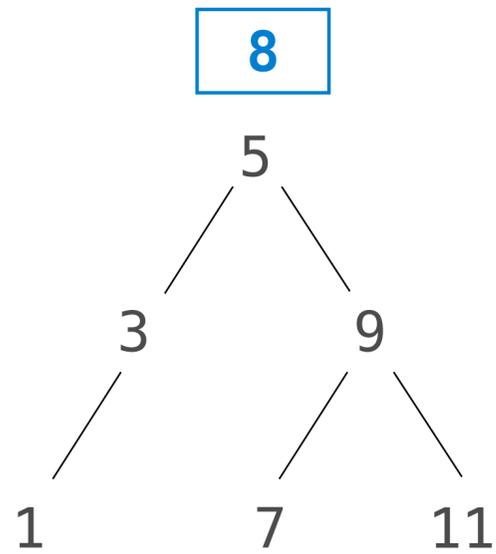
Right!



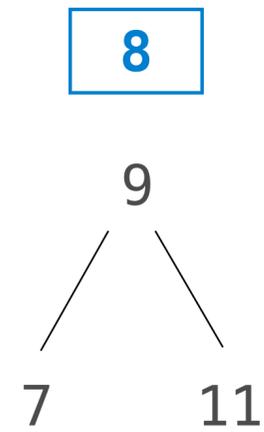
Stop!



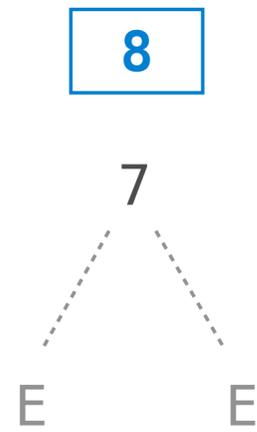
Добавление элемента в множество (BST)



Right!



Left!



Right!

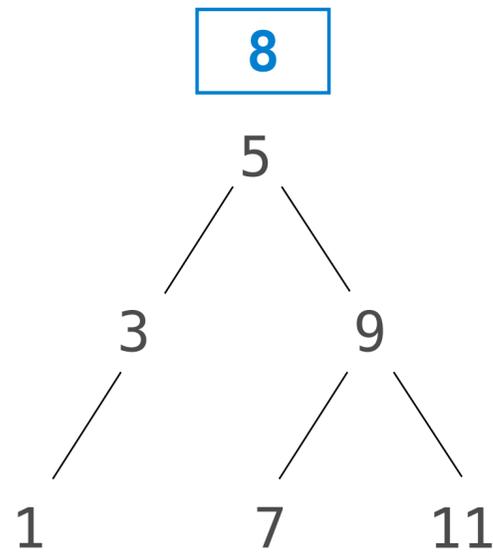


Stop!

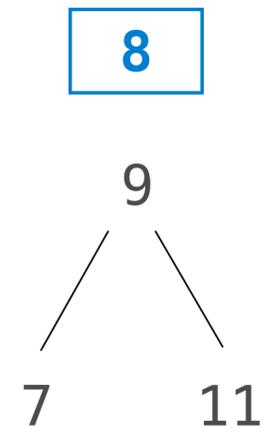


8

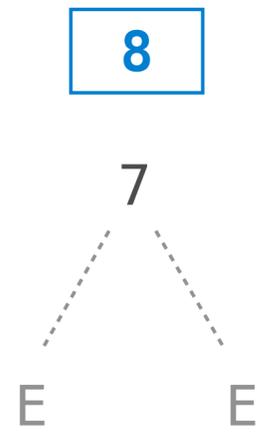
Добавление элемента в множество (BST)



Right!



Left!



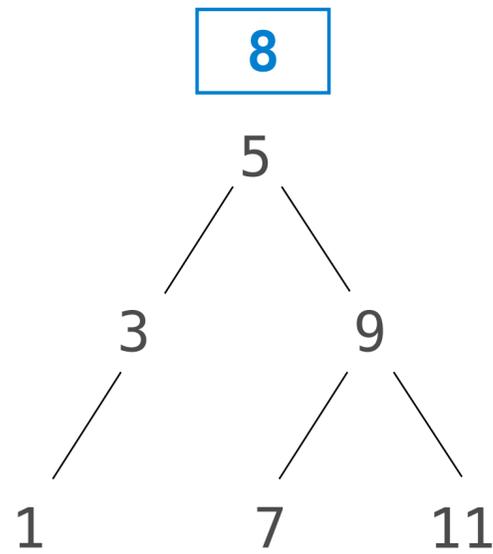
Right!



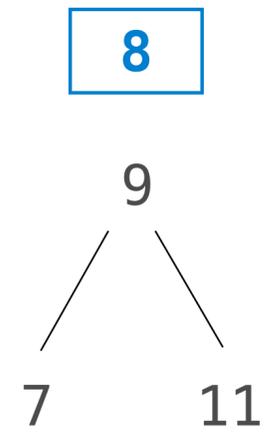
Stop!



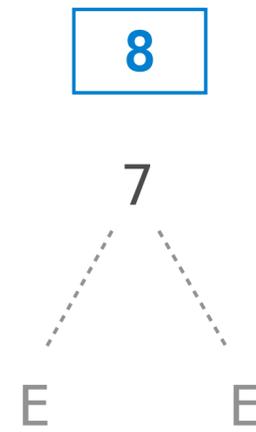
Добавление элемента в множество (BST)



Right!



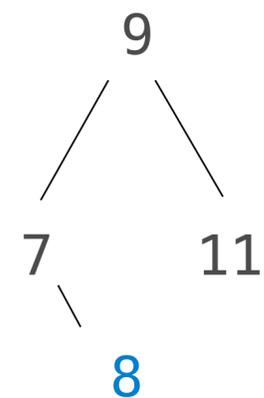
Left!



Right!

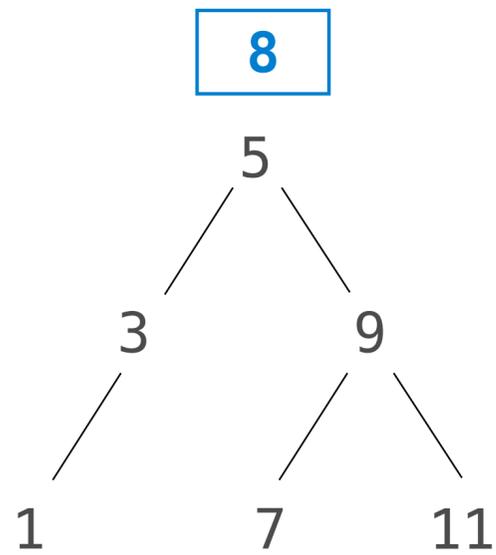


Stop!

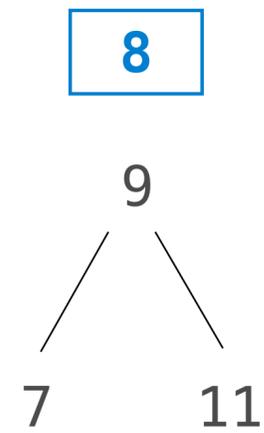


8

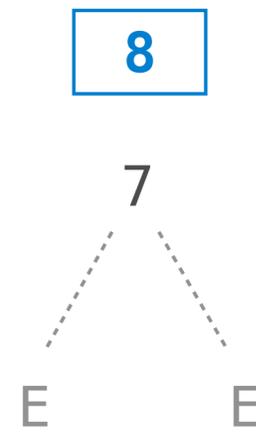
Добавление элемента в множество (BST)



Right!



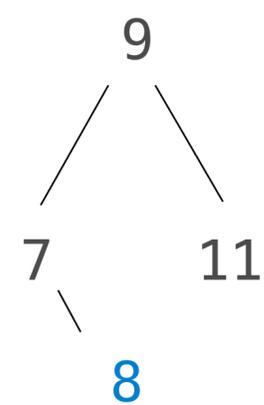
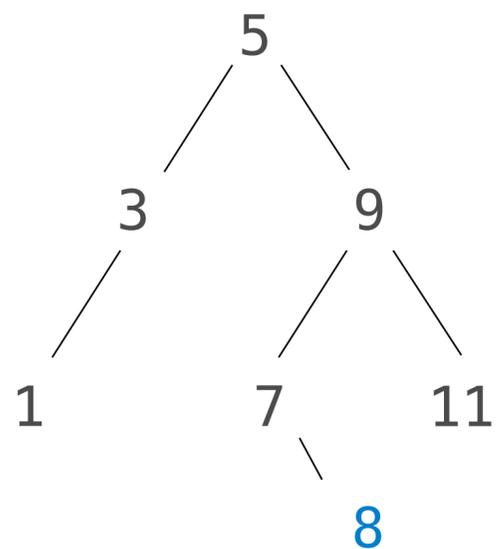
Left!



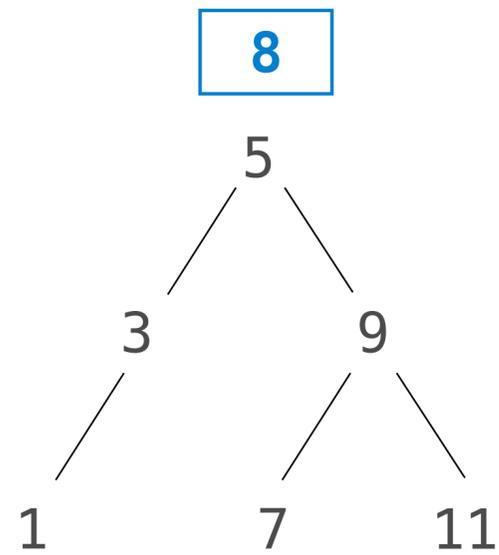
Right!



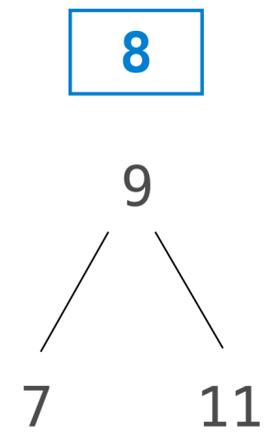
Stop!



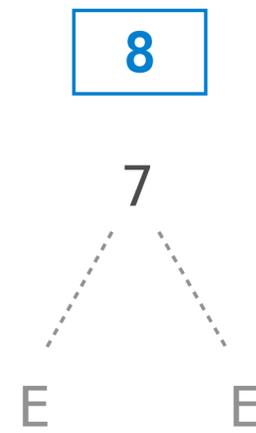
Добавление элемента в множество (BST)



Right!



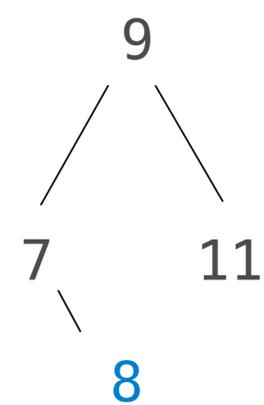
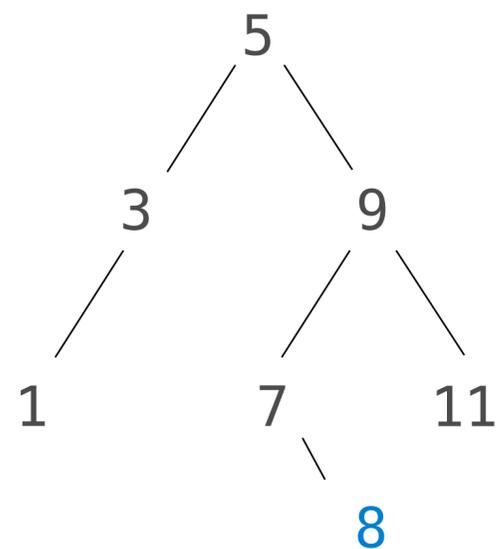
Left!



Right!



Stop!



(Demo)



8