```
1
2   # HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
3   # Path   : uebung06/al/aufgabe02
4   # Version: Mon Mar 24 18:14:58 CET 2025
5
6   from uebung06.al.aufgabe02.binary_search_tree import BinarySearchTree
7   from uebung06.al.aufgabe02.binary_search_tree_adv import BinarySearchTreeADV
8   from uebung06.al.aufgabe02.tree_printer import TreePrinterAcc
9   from uebung06.al.aufgabe02.tree_printer import TreePrinter
10
11  if __name__ == '__main__':
12
13    bst = BinarySearchTree()
14    #bst =  BinarySearchTreeADV("Binary-Search-Tree")
15
16    treePrinterAcc = TreePrinterAcc(bst)
17    treePrinter = TreePrinter(treePrinterAcc)
18    print("Height: " + str(bst.height()))
19    bst.insert(5, "Fuenf")
20    treePrinter.print()
21    print("Height: " + str(bst.height()))
22    bst.insert(3, "Drei")
23    treePrinter.print()
24    print("Height: " + str(bst.height()))
25    bst.insert(6, "Sechs")
26    treePrinter.print()
27    print("Height: " + str(bst.height()))
28    bst.insert(1, "Eins")
29    treePrinter.print()
30    print("Height: " + str(bst.height()))
31    bst.insert(2, "Zwei:1")
32    treePrinter.print()
33    print("Height: " + str(bst.height()))
34    bst.insert(4, "Vier:1")
35    treePrinter.print()
36    print("MaxDepth: " + str(bst.height()))
37    print("insert(4, \"Vier:2\")")
38    bst.insert(4, "Vier:2")
39    treePrinter.print()
40    print("Height: " + str(bst.height()))
41    print("insert(2, \"Zwei:2\")")
42    bst.insert(2, "Zwei:2")
43    treePrinter.print()
44    print("Height: " + str(bst.height()))
45
46    # Some Tests:
47    print("find(3): " + str(bst.find(3)))
48    print("find(2): " + str(bst.find(2)))
49    print("find(7): " + str(bst.find(7)))
50
51
```



```
51
52  """ Session-Log:
53
54  Height: -1
55  === Tree: ========================
56  (5,Fuenf)
57  === Tree. ========================
58  Height: 0
59  === Tree: ========================
60       (5,Fuenf)
61  (3,Drei)/
62  === Tree. ========================
63  Height: 1
64  === Tree: ========================
65       (5,Fuenf)
66  (3,Drei)/ \(6,Sechs)
67  === Tree. ========================
68  Height: 1
69  === Tree: ========================
70            (5,Fuenf)
71       (3,Drei)/ \(6,Sechs)
72  (1,Eins)/
73  === Tree. ========================
74  Height: 2
75  === Tree: ========================
76                (5,Fuenf)
77            (3,Drei)/ \(6,Sechs)
78  (1,Eins)/
79       \(2,Zwei:1)
80  === Tree. ========================
81  Height: 3
82  === Tree: ========================
83                         (5,Fuenf)
84              (3,Drei)/       \(6,Sechs)
85  (1,Eins)/       \(4,Vier:1)
86       \(2,Zwei:1)
87  === Tree. ========================
88  MaxDepth: 3
89  insert(4, "Vier:2")
90  === Tree: ========================
91                         (5,Fuenf)
92              (3,Drei)/       \(6,Sechs)
93  (1,Eins)/       \(4,Vier:2)
94       \(2,Zwei:1)
95  === Tree. ========================
96  Height: 3
97  insert(2, "Zwei:2")
98  === Tree: ========================
99                         (5,Fuenf)
100             (3,Drei)/       \(6,Sechs)
101 (1,Eins)/       \(4,Vier:2)
102      \(2,Zwei:2)
103 === Tree. ========================
104 Height: 3
105 find(3): Drei
106 find(2): Zwei:2
107 find(7): None
108
109 """
```

```python

# HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
# Path   : uebung06/al/aufgabe02
# Version: Mon Mar 24 18:14:58 CET 2025


class BinarySearchTree:
  """
  A Binary-Search-Tree with internal nodes which store key/values and
  external nodes as 'Leave-Marker' (without key/values).
  """

  class _Node:

    def __init__(self, bst):
      self._bst = bst  # the Binary-Search-Tree
      self._key = None
      self._value = None
      self._left = None
      self._right = None

    def get_key(self):
      return self._key

    def set_value(self, value):
      self._value = value

    def get_value(self):
      return self._value

    def get_left(self):
      return self._left

    def get_right(self):
      return self._right

    def is_external(self):
      return self._left == None and self._right == None

    def convert_to_internal_node(self, key, value):
      self._key = key
      self._value = value
      self._left = self._bst._new_node()
      self._right = self._bst._new_node()

    # End of class _Node


  def __init__(self):
    self._root = self._new_node()

  def _new_node(self):
    """
    Factory-Method: Creates a new node.

    Returns a new created node.
    """
    return BinarySearchTree._Node(self)

  def height(self):
    """
    Calculates the height of this tree.

    Returns the height. For an empty tree: -1
    """
    return self._height(self._root)


```



```python


  def _height(self, node):
    """
    Calculates recursively the height of the subtree below node.

    node: The root of the subtree.
    Returns the height of this subtree. For an empty tree: -1
    """

    # TODO: Implement here...

    return -2

  def find(self, key):
    """
    Searches for key in the tree.

    key: The key to search for.
    Returns the associated value or None if key is not found.
    """

    # TODO: Implement here...

    return None

  def _search(self, key, node):
    """
    Searches recursively for key in the subtree with node as root.

    key: The key to search for.
    node: The root of the subtree.
    Returns: If key found the corresponding internal node,
             else the corresponding external node.
    """

    # TODO: Implement here...

    return None

  def insert(self, key, value):
    """
    Inserts a key and its associated value into the tree in a way, that a
    inorder-traverse will return the elements in sorted order.
    """
    if self._root.is_external():
      self._root.convert_to_internal_node(key, value)
    else:
      self._insert_but_wrong(key, value, self._root)  # TODO


  def _insert_but_wrong(self, key, value, node):
    if node.get_left().is_external():
      node.get_left().convert_to_internal_node(key, value)
      return
    if node.get_right().is_external():
      node.get_right().convert_to_internal_node(key, value)
      return
    self._insert_but_wrong(key, value, node.get_right())


```