```
1
2   # HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
3   # Path   : uebung03/ml/aufgabe02
4   # Version: Sun Oct  1 20:02:09 CEST 2023
5
6   from uebung03.ml.aufgabe02.vector_tree import VectorTree
7   from uebung03.ml.aufgabe02.vector_tree import NoSuchNodeException
8
9   if __name__ == '__main__':
10
11    vt = VectorTree()
12
13    vt.print_vector("Empty tree:")
14
15    if vt.size() != 0:
16      raise Exception("Bad size: " + vt.size() + " != 0")
17
18    if vt.root() != None:
19        raise Exception("vt.root() != None")
20
21    a = 'A'
22    vt.set_root(a)
23    vt.print_vector("Setting root with 'A':")
24    if vt.size() != 1:
25      raise Exception("Bad size: " + vt.size() + " != 1")
26    if not vt.is_root(a):
27      raise Exception("not vt.root(a)")
28    if vt.root() != a:
29      raise Exception("vt.root() != a: " + vt.root())
30    if not vt.is_external(a):
31      raise Exception("not vt.is_external(a)")
32    if vt.parent(a) != None:
33      raise Exception("vt.parent(a) != None")
34
35
36    d = 'D'
37    vt.set_right_child(vt.root(), d)
38    vt.print_vector("Setting right child of 'A' with 'D':")
39    if  vt.size() != 2:
40      raise Exception("Bad size: " + vt.size() + " != 2")
41    if not vt.right_child(vt.root()) == d:
42      raise Exception("not vt.right_child(vt.root()) == d : " + vt.right_child(vt.root()
    ))
43    if not vt.is_external(d):
44      raise Exception("not vt.is_external(d)")
45    if not vt.is_internal(vt.root()):
46      raise Exception("!vt.is_internal(vt.root()")
47    if not vt.parent(d) == a:
48      raise Exception("not vt.parent(d) == a")
49
50
51    b = 'B'
52    vt.set_left_child(vt.root(), b)
53    vt.print_vector("Setting left child of 'A' with 'B':")
54    if vt.size() != 3:
55      raise Exception("Bad size: " + vt.size() + " != 3")
56
57    f = 'F'
58    vt.set_right_child(b, f)
59    vt.print_vector("Setting right child of 'B' with 'F':")
60
61    h = 'H'
62    vt.set_right_child(f, h)
63    vt.print_vector("Setting right child of 'F' with 'H':")
64
65
```



```
65
66    g = 'G'
67    vt.set_left_child(f, g)
68    vt.print_vector("Setting left child of 'F' with 'G':")
69    if vt.size() != 6:
70      raise Exception("Bad size: " + vt.size() + " != 6")
71    if not vt.is_internal(f):
72      raise Exception("not vt.is_internal(f)")
73    if not vt.is_external(h):
74      raise Exception("not vt.is_external(h)")
75    if not vt.right_child(vt.right_child(vt.left_child(vt.root()))) == h:
76      raise Exception("not vt.right_child(vt.right_child(vt.left_child(vt.root()))) == h
  ")
77
78    vt.remove_left_child(b)
79    if vt.size() != 6:
80      raise Exception("Bad size: " + vt.size() + " != 6")
81
82    vt.remove_right_child(b)
83    vt.print_vector("Removing right child of 'B':")
84    if vt.size() != 3:
85      raise Exception("Bad size: " + vt.size() + " != 3")
86    if not vt.is_external(b):
87      raise Exception("not vt.is_external(b)")
88
89    vt.set_right_child(d, 'J')
90    vt.print_vector("Setting right child of 'D' with 'J':")
91
92    vt.set_right_child(a, 'X')
93    vt.print_vector("Setting right child of root 'A' with 'X':")
94    if vt.size() != 3:
95      raise Exception("Bad size: " + vt.size() + " != 3")
96
97    vt.set_root('Y')
98    vt.print_vector("Setting root with 'Y':")
99    if vt.size() != 1:
100     raise Exception("Bad size: " + vt.size() + " != 1")
101
102   print("\nTesting if root is external: ", end = "")
103   if not vt.is_external(vt.root()):
104     raise Exception("not vt.is_external(vt.root())")
105   print("o.k.")
106
107   print("\nAsking for node which does not exist: ", end = "")
108   rightChild = vt.right_child('Y')
109   if rightChild != None:
110     raise Exception("rightChild != None")
111   print("o.k.")
112
113   print("\nUsing node which does not exist: ", end = "")
114   noSuchNodeException = None
115   try:
116     vt.set_right_child('A', 'B')
117   except (NoSuchNodeException) as e:
118     noSuchNodeException = e
119   if noSuchNodeException == None:
120     raise Exception("NoSuchNodeException missing!")
121   print("o.k.")
122
123
```

```
123
124 """ Session-Log::
125
126 Empty tree:
127 [None, None]
128
129 Setting root with 'A':
130 [None, 'A']
131
132 Setting right child of 'A' with 'D':
133 [None, 'A', None, 'D']
134
135 Setting left child of 'A' with 'B':
136 [None, 'A', 'B', 'D']
137
138 Setting right child of 'B' with 'F':
139 [None, 'A', 'B', 'D', None, 'F', None, None]
140
141 Setting right child of 'F' with 'H':
142 [None, 'A', 'B', 'D', None, 'F', None, None, None, None, None, 'H', None, None, None,
    None]
143
144 Setting left child of 'F' with 'G':
145 [None, 'A', 'B', 'D', None, 'F', None, None, None, None, 'G', 'H', None, None, None, N
    one]
146
147 Removing right child of 'B':
148 [None, 'A', 'B', 'D', None, None, None, None, None, None, None, None, None, None, None
    , None]
149
150 Setting right child of 'D' with 'J':
151 [None, 'A', 'B', 'D', None, None, None, 'J', None, None, None, None, None, None, None,
     None]
152
153 Setting right child of root 'A' with 'X':
154 [None, 'A', 'B', 'X', None, None, None, None, None, None, None, None, None, None, None
    , None]
155
156 Setting root with 'Y':
157 [None, 'Y', None, None, None, None, None, None, None, None, None, None, None, None, No
    ne, None]
158
159 Testing if root is external: o.k.
160
161 Asking for node which does not exist: o.k.
162
163 Using node which does not exist: o.k.
164
165 """
```



```
1
2  # HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
3  # Path   : uebung03/ml/aufgabe02
4  # Version: Sun Oct  1 20:02:09 CEST 2023
5
6  import enum
7  from uebung03.ml.aufgabe02.no_such_node_exception import NoSuchNodeException
8
9
10 ROOT_INDEX = 1
11
12 class VectorTree:
13
14   class _child_side(enum.Enum):
15     LEFT = enum.auto()
16     RIGHT = enum.auto()
17
18   def __init__(self):
19     self._binary_tree = []
20     self._binary_tree.append(None)
21     self._binary_tree.append(None)
22     self._size = 0
23
24   def root(self):
25     return self._binary_tree[ROOT_INDEX]
26
27   def set_root(self, root):
28     if self.root() != None:
29       self._remove(ROOT_INDEX)
30     self._binary_tree[ROOT_INDEX] = root
31     self._size = 1
32
33   def parent(self, child):
34     if self.is_root(child):
35       return None  # this object is the root... thus no parent...
36     return self._binary_tree[self._position(child) // 2]
37
38   def left_child(self, parent):
39     return self._get_child(parent, VectorTree._child_side.LEFT)
40
41   def right_child(self, parent):
42     return self._get_child(parent, VectorTree._child_side.RIGHT)
43
44   def _get_child(self, parent, child_side):
45     child_pos = self._child_pos_by_value(parent, child_side)
46     if not self._has_node_at_position(child_pos):
47       return None
48     return self._binary_tree[child_pos]
49
50   def _child_pos_by_value(self, parent, child_side):
51     return self._child_pos_by_index(self._position(parent), child_side)
52
53
54   def _child_pos_by_index(self, parent_pos, child_side):
55     return parent_pos * 2 + (0 if child_side == VectorTree._child_side.LEFT else 1)
56
57   def is_internal(self, node):
58     return not self.is_external(node)
59
60   def is_external(self, node):
61     left_child_pos = self._child_pos_by_value(node, VectorTree._child_side.LEFT)
62     rigth_child_pos = left_child_pos + 1
63     result = None
64     if self._has_node_at_position(left_child_pos) or self._has_node_at_position(rigth_
   child_pos):
65       return False
66     else:
67       return True
68     return result
69
70
```

```
70
71    def _has_node_at_position(self, pos):
72      if pos > (len(self._binary_tree) - 1):
73        return False
74      return self._binary_tree[pos] != None
75
76    def is_root(self, node):
77      return node == self._binary_tree[ROOT_INDEX]
78
79    def set_right_child(self, parent, child):
80      self._set_child(parent, child, VectorTree._child_side.RIGHT)
81
82    def set_left_child(self, parent, child):
83      self._set_child(parent, child, VectorTree._child_side.LEFT)
84
85    def _set_child(self, parent, child, child_side):
86      self._remove_child(parent, child_side)
87      child_pos = self._child_pos_by_value(parent, child_side)
88      self._assure_size(child_pos)
89      if self._binary_tree[child_pos] == None:
90        self._size += 1
91      self._binary_tree[child_pos] = child
92
93    def remove_right_child(self, parent):
94      self._remove_child(parent, VectorTree._child_side.RIGHT)
95
96    def remove_left_child(self, parent):
97      self._remove_child(parent, VectorTree._child_side.LEFT)
98
99    def _remove_child(self, parent, child_side):
100     child_pos = self._child_pos_by_value(parent, child_side)
101     if self._has_node_at_position(child_pos):
102       self._remove(child_pos)
103
104   def _remove(self, node_pos):
105     """ Precondition: Node exists. """
106     for child_side in VectorTree._child_side:
107       child_pos = -1
108       child_pos = self._child_pos_by_index(node_pos, child_side)
109       if self._has_node_at_position(child_pos):
110         self._remove(child_pos)
111     self._binary_tree[node_pos] = None
112     self._size -= 1
113
114   def size(self):
115     return self._size
116
117   def _position(self, node):
118     try:
119       pos = self._binary_tree.index(node)
120     except ValueError:
121       raise NoSuchNodeException("No node:" + node)
122     return pos
123
124   def _assure_size(self, pos):
125     current_length = len(self._binary_tree)
126     if pos >= current_length:
127       new_length = 2 * current_length
128       i = current_length
129       while i < new_length:
130         self._binary_tree.append(None)
131         i += 1
132
133   def print_vector(self, message):
134     print("\n" + message)
135     print(self._binary_tree)
```



```
1
2   # HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
3   # Path   : uebung03/ml/aufgabe02
4   # Version: Sun Oct  1 20:02:09 CEST 2023
5
6   class NoSuchNodeException(Exception):
7
8       def __init__(self, err):
9         super().__init__(err)
10
11
```