Laporan
Struktur Data
HashMap

Dosen Pengampu:

Muchammad Chandra Cahyo Utomo, M.Kom. 199205202019031013

Disusun Oleh :

Guntur Wisnu Saputra	11211042
Muhammad Insan Kamil	11211058
Muhammad Ricky Zakaria	11211062
Ramadhan Djibran Sanjaya	11211070
Rangga Hermawan	11211071
Rendy Pernanda	11211074

21 November 2022

Source Code

No.	HashMap.py

```
1  class Node:
2      def __init__(self, key, value):
3          self.__parent = None
4          self.__left = None
5          self.__right = None
6          self.__key = key
7          self.__isRed = False
8          self.__value = value
9      def setRed(self, boolean):
10         self.__isRed = boolean
11     def setParent(self, parent):
12         self.__parent = parent
13     def setLeft(self,left):
14         self.__left = left
15     def setRight(self,right):
16         self.__right = right
17     def setKey(self, key):
18         self.__key = key
19     def setValue(self, value):
20         self.__value = value
21     def getRed(self):
22         return self.__isRed
23     def getParent(self):
24         return self.__parent
25     def getLeft(self):
26         return self.__left
27     def getRight(self):
28         return self.__right
29     def getKey(self):
30         return self.__key
31     def getValue(self):
32         return self.__value
33
34 class HashMap:
35     def __init__(self,sensitive):
36         self.nil = Node(0,"nil")
37         self.nil.setRed(False)
38         self.nil.setLeft(None)
39         self.nil.setRight(None)
40         self.root = self.nil
41         self.sensitive = sensitive
42
43     def add(self, key, value):
```

```
44          key = key if self.sensitive == True else key.lower()
45          if self.sensitive == False and type(value) == str:
46              value = value.lower()
47          new_node = Node(key, value)
48          new_node.setParent(None)
49          new_node.setLeft(self.nil)
50          new_node.setRight(self.nil)
51          new_node.setRed(True)
52
53          parent = None
54          current = self.root
55          while current != self.nil:
56              parent = current
57              if new_node.getKey() < current.getKey():
58                  current = current.getLeft()
59              elif new_node.getKey() > current.getKey():
60                  current = current.getRight()
61              else:
62                  return
63
64          new_node.setParent(parent)
65          if parent == None:
66              self.root = new_node
67          elif new_node.getKey() < parent.getKey():
68              parent.setLeft(new_node)
69          else:
70              parent.setRight(new_node)
71
72          self.fix_add(new_node)
73
74      def rotate_left(self, node):
75          y = node.getRight()
76          node.setRight(y.getLeft())
77          if y.getLeft() != self.nil:
78              y.getLeft().setParent(node)
79
80          y.setParent(node.getParent())
81          if node.getParent() == None:
82              self.root = y
83          elif node == node.getParent().getLeft():
84              node.getParent().setLeft(y)
85          else:
86              node.getParent().setRight(y)
87          y.setLeft(node)
88          node.setParent(y)
89
```

```
90      def rotate_right(self, node):
91          y = node.getLeft()
92          node.setLeft(y.getRight())
93          if y.getRight() != self.nil:
94              y.getRight().setParent(node)
95
96          y.setParent(node.getParent())
97          if node.getParent() == None:
98              self.root = y
99          elif node == node.getParent().getRight():
100             node.getParent().setRight(y)
101         else:
102             node.getParent().setLeft(y)
103         y.setRight(node)
104         node.setParent(y)
105
106     def fix_add(self, new_node):
107         while self.root != new_node and True == new_node.getParent().getRed():
108             if new_node.getParent() == new_node.getParent().getParent().getLeft():
109                 if new_node.getParent().getParent().getRight().getRed():
110                     new_node.getParent().getParent().getRight().setRed(False)
111                     new_node.getParent().getParent().setRed(True)
112                     new_node.getParent().setRed(False)
113                     new_node = new_node.getParent().getParent()
114                 else:
115                     if new_node == new_node.getParent().getRight():
116                         self.rotate_left( new_node.getParent() )
117                     new_node.getParent().setRed(False)
118                     new_node.getParent().getParent().setRed(True)
119                     self.rotate_right( new_node.getParent().getParent() )
120             else:
121                 if new_node.getParent().getParent().getLeft().getRed():
122                     new_node.getParent().getParent().getLeft().setRed(False)
123                     new_node.getParent().getParent().setRed(True)
124                     new_node.getParent().setRed(False)
125                     new_node = new_node.getParent().getParent()
126                 else:
127                     if new_node == new_node.getParent().getLeft():
128                         self.rotate_right( new_node.getParent() )
129                     new_node.getParent().setRed(False)
130                     new_node.getParent().getParent().setRed(True)
131                     self.rotate_left( new_node.getParent().getParent() )
132         self.root.setRed(False)
133
134     def maxKeyNode_printHelper(self):
135         current = self.root
```

```
136
137         while(current.getRight() is not self.nil):
138             current = current.getRight()
139
140         return current
141
142
143     def minKeyNode_printHelper(self):
144         current = self.root
145
146         while(current.getLeft() is not self.nil):
147             current = current.getLeft()
148
149         return current
150
151     def minKeyNode(self, node):
152         current = node
153
154         while(current.getLeft() is not self.nil):
155             current = current.getLeft()
156
157         return current
158
159     def transplant(self, deletedNode, replacer):
160         if deletedNode.getParent() == self.nil:
161             self.root = replacer
162         elif deletedNode == deletedNode.getParent().getLeft():
163             deletedNode.getParent().setLeft(replacer)
164         else:
165             deletedNode.getParent().setRight(replacer)
166         replacer.setParent(deletedNode.getParent())
167
168
169     def delete_fixup(self, node):
170         while node != self.root and node.getRed() == False:
171             if node == node.getParent().getLeft():
172                 siblings = node.getParent().getRight()
173                 if siblings.getRed() == True:
174                     siblings.setRed(False)
175                     node.getParent().setRed(True)
176                     self.rotate_left(node.getParent())
177                     siblings = node.getParent().getRight()
178
179                 if siblings.getLeft().getRed() == False and siblings.getRight().getRed() == False:
180                     siblings.setRed(True)
181                     node = node.getParent()
```

```
182
183             else:
184                 if siblings.getRight().getRed() == False:
185                     siblings.getLeft().setRed(False)
186                     siblings.setRed(True)
187                     self.rotate_right(siblings)
188                     siblings = node.getParent().getRight()
189
190                 siblings.setRed(node.getParent().getRed())
191                 node.getParent().setRed(False)
192                 siblings.getRight().setRed(False)
193                 self.rotate_left(node.getParent())
194                 node = self.root
195
196         else:
197             siblings = node.getParent().getLeft()
198             if siblings.getRed() == True:
199                 siblings.setRed(False)
200                 node.getParent().setRed(True)
201                 self.rotate_right(node.getParent())
202                 siblings = node.getParent().getLeft()
203
204             if siblings.getRight().getRed() == False and siblings.getLeft().getRed() == False:
205                 siblings.setRed(True)
206                 node = node.getParent()
207
208             else:
209                 if siblings.getLeft().getRed() == False:
210                     siblings.getRight().setRed(False)
211                     siblings.setRed(True)
212                     self.rotate_left(siblings)
213                     siblings = node.getParent().getLeft()
214
215                 siblings.setRed(node.getParent().getRed())
216                 node.getParent().setRed(False)
217                 siblings.getLeft().setRed(False)
218                 self.rotate_right(node.getParent())
219                 node = self.root
220
221     node.setRed(False)
222
223 def delete(self, key):
224     key = key if self.sensitive == True else key.lower()
225     if self.search(self.root,key):
226         deletedNode = self.search(self.root,key)
227     else:
```

```
228             print(f"Tidak bisa menghapus, key:'{key}' tidak ada")
229             return
230         x = None
231         replacer_orignal_color = deletedNode.getRed()
232         if deletedNode.getLeft() == self.nil:
233             x = deletedNode.getRight()
234             self.transplant(deletedNode, deletedNode.getRight())
235 
236         elif deletedNode.getRight() == self.nil:
237             x = deletedNode.getLeft()
238             self.transplant(deletedNode, deletedNode.getLeft())
239 
240         else:
241             replacer = self.minKeyNode(deletedNode.getRight())
242             replacer_orignal_color = replacer.getRed()
243             x = replacer.getRight()
244             if replacer.getParent()== deletedNode:
245                 x.setParent(deletedNode)
246 
247             else:
248                 self.transplant(replacer, replacer.getRight())
249                 replacer.setRight(deletedNode.getRight())
250                 replacer.getRight().setParent(replacer)
251 
252             self.transplant(deletedNode, replacer)
253             replacer.setLeft(deletedNode.getLeft())
254             replacer.getLeft().setParent(replacer)
255             replacer.setRed(deletedNode.getRed())
256 
257         if replacer_orignal_color == False:
258             self.delete_fixup(x)
259 
260     def edit(self, key, value):
261         key = key if self.sensitive == True else key.lower()
262         if self.sensitive == False and type(value) == str:
263             value = value.lower()
264         self.search(self.root,key).setValue(value)
265 
266     def get(self, key):
267         key = key if self.sensitive == True else key.lower()
268         if self.search(self.root,key):
269             return self.search(self.root,key).getValue()
270         else:
271             return f"Key:'{key}' tidak ada"
272 
273     def search(self,node, key):
```

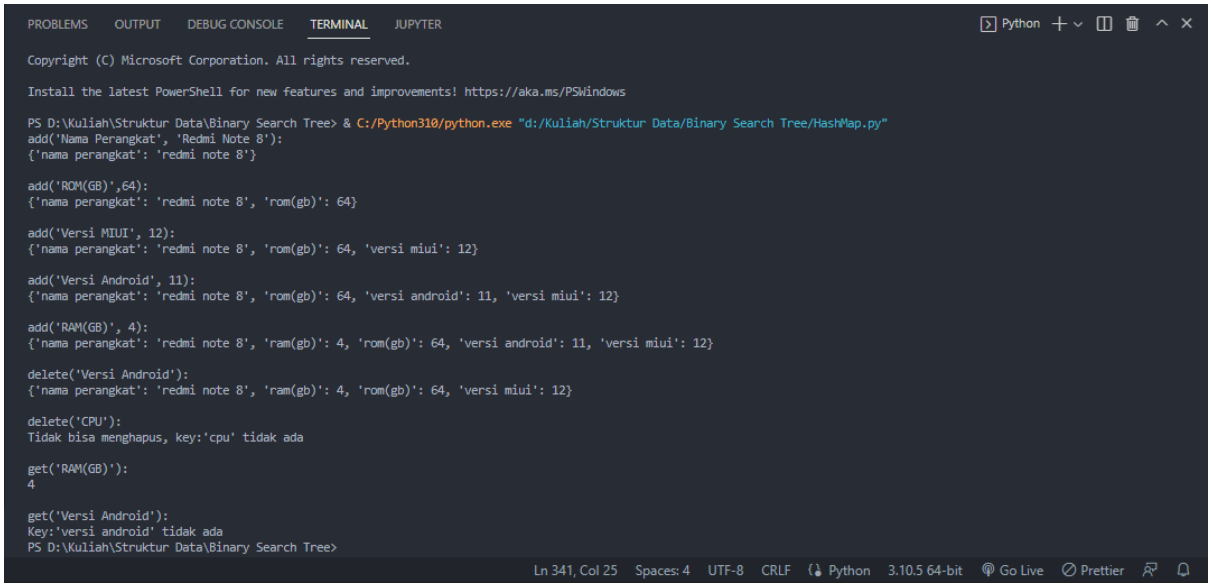
```
274             key = key if self.sensitive == True else key.lower()
275             if key < node.getKey():
276                 if node.getLeft() is self.nil:
277                     return False
278                 return self.search(node.getLeft(),key)
279             elif key > node.getKey():
280                 if node.getRight() is self.nil:
281                     return False
282                 return self.search(node.getRight(),key)
283             else:
284                 return node
285
286     def printHashMap(self):
287         return self.printHelper(self.root)
288
289     def printHelper(self,node):
290         if node:
291             self.printHelper(node.getLeft())
292             if type(node.getKey()) == str and type(node.getValue()) == str:
293                 if node == self.minKeyNode_printHelper() and node.getRight() == self.nil and
    node.getLeft() == self.nil and node == self.root:
294                     txt3 = "{}'{}': '{}'".format("{",node.getKey(),node.getValue())
295                     print(txt3, end="}\n")
296                 elif node == self.minKeyNode_printHelper():
297                     txt3 = "{}'{}': '{}'".format("{",node.getKey(),node.getValue())
298                     print(txt3, end=", ")
299                 elif node == self.maxKeyNode_printHelper():
300                     print(f"'{node.getKey()}': '{node.getValue()}'", end="}\n")
301                 else:
302                     print(f"'{node.getKey()}': '{node.getValue()}'", end=", ")
303             elif type(node.getKey()) == str and type(node.getValue()) != str:
304                 if node == self.minKeyNode_printHelper():
305                     txt3 = "{}'{}': {}".format("{",node.getKey(),node.getValue())
306                     print(txt3, end=", ")
307                 elif node == self.maxKeyNode_printHelper():
308                     print(f"'{node.getKey()}': {node.getValue()}", end="}\n")
309                 else:
310                     print(f"'{node.getKey()}': {node.getValue()}", end=", ")
311             self.printHelper(node.getRight())
312
313 Device = HashMap(False)
314 print("add('Nama Perangkat', 'Redmi Note 8'):")
315 Device.add("Nama Perangkat", "Redmi Note 8")
316 Device.printHashMap()
317 print()
318 print("add('ROM(GB)',64):")
```

```
319 Device.add("ROM(GB)",64)
320 Device.printHashMap()
321 print()
322 print("add('Versi MIUI', 12):")
323 Device.add("Versi MIUI", 12)
324 Device.printHashMap()
325 print()
326 print("add('Versi Android', 11):")
327 Device.add("Versi Android", 11)
328 Device.printHashMap()
329 print()
330 print("add('RAM(GB)', 4):")
331 Device.add("RAM(GB)", 4)
332 Device.printHashMap()
333 print()
334 print("delete('Versi Android'):")
335 Device.delete("Versi Android")
336 Device.printHashMap()
337 print()
338 print("delete('CPU'):")
339 Device.delete("CPU")
340 print()
341 print("get('RAM(GB)'):")
342 print(Device.get("RAM(GB)"))
343 print()
344 print("get('Versi Android'):")
345 print(Device.get("Versi Android"))
```

#Hasil Run

HashMap.py

Case-insensitive



Case-sensitive