# Laporan
## Struktur Data
## **Red Black Tree**


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. | Red-BlackTree.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 RBTree: |
| 35 | def __init__(self): |
| 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 | |
| 42 | def insert(self, key, value): |
| 43 | new_node = Node(key, value) |

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

```
90          if y.getRight() != self.nil:
91              y.getRight().setParent(node)
92
93          y.setParent(node.getParent())
94          if node.getParent() == None:
95              self.root = y
96          elif node == node.getParent().getRight():
97              node.getParent().setRight(y)
98          else:
99              node.getParent().setLeft(y)
100         y.setRight(node)
101         node.setParent(y)
102
103     def fix_insert(self, new_node):
104         while self.root != new_node and True == new_node.getParent().getRed():
105             if new_node.getParent() == new_node.getParent().getParent().getLeft():
106                 if new_node.getParent().getParent().getRight().getRed():
107                     new_node.getParent().getParent().getRight().setRed(False)
108                     new_node.getParent().getParent().setRed(True)
109                     new_node.getParent().setRed(False)
110                     new_node = new_node.getParent().getParent()
111                 else:
112                     if new_node == new_node.getParent().getRight():
113                         self.rotate_left( new_node.getParent() )
114                     new_node.getParent().setRed(False)
115                     new_node.getParent().getParent().setRed(True)
116                     self.rotate_right( new_node.getParent().getParent() )
117             else:
118                 if new_node.getParent().getParent().getLeft().getRed():
119                     new_node.getParent().getParent().getLeft().setRed(False)
120                     new_node.getParent().getParent().setRed(True)
121                     new_node.getParent().setRed(False)
122                     new_node = new_node.getParent().getParent()
123                 else:
124                     if new_node == new_node.getParent().getLeft():
125                         self.rotate_right( new_node.getParent() )
126                     new_node.getParent().setRed(False)
127                     new_node.getParent().getParent().setRed(True)
128                     self.rotate_left( new_node.getParent().getParent() )
129         self.root.setRed(False)
130
131     def minKeyNode(self, node):
132         current = node
133
134         while(current.getLeft() is not self.nil):
135             current = current.getLeft()
```

```
136
137         return current
138
139     def transplant(self, deletedNode, replacer):
140         if deletedNode.getParent() == self.nil:
141             self.root = replacer
142         elif deletedNode == deletedNode.getParent().getLeft():
143             deletedNode.getParent().setLeft(replacer)
144         else:
145             deletedNode.getParent().setRight(replacer)
146         replacer.setParent(deletedNode.getParent())
147
148
149     def delete_fixup(self, node):
150         while node != self.root and node.getRed() == False:
151             if node == node.getParent().getLeft():
152                 siblings = node.getParent().getRight()
153                 if siblings.getRed() == True:
154                     siblings.setRed(False)
155                     node.getParent().setRed(True)
156                     self.rotate_left(node.getParent())
157                     siblings = node.getParent().getRight()
158
159                 if siblings.getLeft().getRed() == False and siblings.getRight().getRed() == False:
160                     siblings.setRed(True)
161                     node = node.getParent()
162
163                 else:
164                     if siblings.getRight().getRed() == False:
165                         siblings.getLeft().setRed(False)
166                         siblings.setRed(True)
167                         self.rotate_right(siblings)
168                         siblings = node.getParent().getRight()
169
170                     siblings.setRed(node.getParent().getRed())
171                     node.getParent().setRed(False)
172                     siblings.getRight().setRed(False)
173                     self.rotate_left(node.getParent())
174                     node = self.root
175
176             else:
177                 siblings = node.getParent().getLeft()
178                 if siblings.getRed() == True:
179                     siblings.setRed(False)
180                     node.getParent().setRed(True)
181                     self.rotate_right(node.getParent())
```

```
182                 siblings = node.getParent().getLeft()
183
184             if siblings.getRight().getRed() == False and siblings.getLeft().getRed() == False:
185                 siblings.setRed(True)
186                 node = node.getParent()
187
188             else:
189                 if siblings.getLeft().getRed() == False:
190                     siblings.getRight().setRed(False)
191                     siblings.setRed(True)
192                     self.rotate_left(siblings)
193                     siblings = node.getParent().getLeft()
194
195                 siblings.setRed(node.getParent().getRed())
196                 node.getParent().setRed(False)
197                 siblings.getLeft().setRed(False)
198                 self.rotate_right(node.getParent())
199                 node = self.root
200
201     node.setRed(False)
202
203 def delete(self, key):
204     if self.search(key):
205         deletedNode = self.search(key)
206     else:
207         print(f"Tidak bisa menghapus, key:{key} tidak ada")
208         return
209     x = None
210     replacer_orignal_color = deletedNode.getRed()
211     if deletedNode.getLeft() == self.nil:
212         x = deletedNode.getRight()
213         self.transplant(deletedNode, deletedNode.getRight())
214
215     elif deletedNode.getRight() == self.nil:
216         x = deletedNode.getLeft()
217         self.transplant(deletedNode, deletedNode.getLeft())
218
219     else:
220         replacer = self.minKeyNode(deletedNode.getRight())
221         replacer_orignal_color = replacer.getRed()
222         x = replacer.getRight()
223         if replacer.getParent()== deletedNode:
224             x.setParent(deletedNode)
225
226         else:
227             self.transplant(replacer, replacer.getRight())
```

```
228                 replacer.setRight(deletedNode.getRight())
229                 replacer.getRight().setParent(replacer)
230 
231             self.transplant(deletedNode, replacer)
232             replacer.setLeft(deletedNode.getLeft())
233             replacer.getLeft().setParent(replacer)
234             replacer.setRed(deletedNode.getRed())
235 
236         if replacer_orignal_color == False:
237             self.delete_fixup(x)
238 
239     def exist(self, key):
240         if self.root == None:
241             print(f"Tree Kosong, key:{key} Tidak Ada")
242         elif self.search(key):
243             print(f"key:{key} Ada")
244             return True
245         else:
246             print(f"key:{key} Tidak Ada")
247             return False
248 
249     def edit(self, key, value):
250         if self.exist(key):
251             temp = self.search(key).getValue()
252             self.search(key).setValue(value)
253             print(f"Red Black Tree dengan key:{key}, valuenya telah diubah dari {temp} menjadi
    {self.search(key).getValue()}")
254         else:
255             print("Key tidak ditemukan, tidak bisa mengupdate value")
256 
257 
258     def search_helper(self,node, key):
259         if key < node.getKey():
260             if node.getLeft() is None:
261                 return False
262             return self.search_helper(node.getLeft(),key)
263         elif key > node.getKey():
264             if node.getRight() is None:
265                 return False
266             return self.search_helper(node.getRight(),key)
267         else:
268             return node
269 
270     def search(self, key):
271         return self.search_helper(self.root,key)
272 
```

```
273     def height(self,node):
274         return 1 + max(self.height(node.getLeft()), self.height(node.getRight())) if node else -1
275 
276     def PrintTree(self):
277         nlevels = self.height(self.root)
278         width =  pow(2,nlevels+1)
279 
280         q=[(self.root,0,width,'c')]
281         levels=[]
282 
283         while(q):
284             node,level,x,align= q.pop(0)
285             if node:
286                 if len(levels)<=level:
287                     levels.append([])
288 
289                 levels[level].append([node,level,x,align])
290                 seg= width//(pow(2,level+1))
291                 q.append((node.getLeft(),level+1,x-seg,'l'))
292                 q.append((node.getRight(),level+1,x+seg,'r'))
293 
294         for i,l in enumerate(levels):
295             pre=0
296             preline=0
297             linestr=''
298             pstr=''
299             seg= width//(pow(2,i+1))
300             for n in l:
301                 valstr= str(n[0].getKey())
302                 if n[3]=='r':
303                     linestr+=' '*(n[2]-preline-1-seg-seg//2)+ '¯'*(seg +seg//2)+'\\'
304                     preline = n[2]
305                 if n[3]=='l':
306                     linestr+=' '*(n[2]-preline-1)+'/' + '¯'*(seg+seg//2)
307                     preline = n[2] + seg + seg//2
308                 valstrC = "\033[0;31m"+ valstr +"\033[0m" if n[0].getRed() == True else valstr
309                 pstr+=' '*(n[2]-pre-len(valstr))+ valstrC
310                 pre = n[2]
311             print(linestr)
312             print(pstr)
313 
314 r = RBTree()
315 r.insert(1, "Satu")
316 print("insert(1):")
317 r.PrintTree()
318 r.insert(2, "Dua")
```

```
print("insert(2):")
r.PrintTree()
r.insert(3, "Tiga")
print("insert(3):")
r.PrintTree()
r.insert(4, "Empat")
print("insert(4):")
r.PrintTree()
r.insert(5, "Lima")
print("insert(5):")
r.PrintTree()
r.insert(6, "Enam")
print("insert(6):")
r.PrintTree()
r.insert(7, "Tujuh")
print("insert(7):")
r.PrintTree()
r.insert(8, "Delapan")
print("insert(8):")
r.PrintTree()
print("delete(5):")
print("Before:")
r.PrintTree()
print()
print("After:")
r.delete(5)
r.PrintTree()
print()
print("\033[0;31m"+"exist(10):"+"\033[0m")
r.exist(10)
print()
print("\033[0;31m"+"edit(3,'Three'):"+"\033[0m")
r.edit(3,'Three')
print()
```

## #Hasil Run

### Red-BlackTree.py