Fundação CECIERJ
Tecnologia em Sistemas de Computação
Programação com Interfaces Gráficas
AD1 1^o semestre de 2019
Gabarito

Professores: Mario Benevides e Paulo Roma

Resumo

Árvores Balanceadas-Ponderadas Amortizadas

1 Atividade

A seguir serão apresentados algumas das implementações relativas ao que deve ser entregue nessa primeira etapa. Não existe apenas uma forma de fazer a implementação, os trechos de códigos apresentados devem servir de inspiração. **Na AD2 ainda deve ser implementado o que faltou na AD1.**

```
1  def rebalance(self, bstNode):
2      def make_tree(nodes, begin, end, parent):
3          if begin > end:
4              return None
5          elif
6              middle = math.ceil(begin + (end - begin) / 2.0)
7              node = nodes[middle]
8              node.parent = parent
9              node.left = make_tree(nodes, begin, middle - 1, node)
10             node.right= make_tree(nodes, middle + 1, end, node)
11             return node
12
13     if bstNode == None:
14         return
15     node_parent = bstNode.parent
16
17     node_list = []
18     self.__inOrder(bstNode, node_list)
19
20     subtree_root = make_tree(node_list, 0,
21                  node_list.__len__(), bstNode.parent)
22
23     if subtree_root.parent == None:
```

```
        self.__root = subtree_root
    elif node_parent.left == bstNode:
        node_parent.left = subtree_root
    else:
        node_parent.right = subtree_root
    self.count_node(self.__root)
```

Remoção de nó.

```
def remove(self, obj):
    node = self.findEntry(obj)
    if node == None:
        return False
    parent = node.parent

    self.unlinkNode(n)
    self.count_node(self.__root)
 # update counters

    if self.self_balancing:
        unbalanced_node = self.find_unbalanced(parent)
        if unbalanced_node == not None:
            self.rebalance(unbalanced_node)

    return True
```

Operação de diferença:

```
def set_diff(itr1, itr2):
    p1 = peekable(itr1)
    p2 = peekable(itr2)
    result = type(itr1)()

    while p1.hasNext():
        i1 = p1.peek()
        i2 = p2.peek()
        flag = False

        while p2.hasNext():
            i2 = p2.peek()
            if i2 == i1:
                flag = True
            else:
                next(p2)
        if flag == False:
            result.append(i1)
    return result

```

Operação de união:

```
def set_union(itr1, itr2):
    p1 = peekable(itr1)
    p2 = peekable(itr2)
    result = type(itr1)()
    while p1.hasNext() or p2.hasNext():
        i1 = p1.peek()
        i2 = p2.peek()
        try:
            if i1 < i2:
                result.append(i1)
                next(p1)
            elif i2 < i1:
                result.append(i2)
                next(p2)
            else:
                result.append(i1)
                next(p1)
                next(p2)
        except TypeError:
            if p1.isLast() and not p2.isLast():
                result.append(i2)
                next(p2)
            elif p2.isLast() and not p1.isLast():
                result.append(i1)
                next(p1)
    return result
```

As demais operações ainda devem ser implementadas a tempo na AD2.