

Code coverage	2
1. Queue coverage	2
2. Splay tree coverage	4

Code coverage

https://youtube.com/playlist?list=PLAwxTw4SYaPkWVHeC_8aSlbSxE_NXI76g&si=lhs0mKJKm5pW3D1R

1. Queue coverage

- In the test function we call checkRep function after every single test that we do, just as a further test. Any of the test that we do that call the internals of the queue class should probably be in checkRep, because we don't necessarily want our test code to have to worry about the internals of it, we want the internal testing to be taken care of by the class itself.

TASK: Achieve full statement coverage on the Queue class.

```
import array

class Queue:
    def __init__(self, size_max):
        assert size_max > 0
        self.max = size_max
        self.head = 0
        self.tail = 0
        self.size = 0
        self.data = array.array('i', range(size_max))

    def empty(self):
        return self.size == 0

    def full(self):
        return self.size == self.max

    def enqueue(self, x):
        if self.size == self.max:
            return False
        self.data[self.tail] = x
        self.size += 1
        self.tail += 1
        if self.tail == self.max:
            self.tail = 0
        return True

    def dequeue(self):
        if self.size == 0:
            return None
        x = self.data[self.head]
        self.size -= 1
        self.head += 1
        if self.head == self.max:
            self.head = 0
        return x
```

```

def checkRep(self):
    assert self.tail >= 0
    assert self.tail < self.max
    assert self.head >= 0
    assert self.head < self.max
    if self.tail > self.head:
        assert (self.tail-self.head) == self.size
    if self.tail < self.head:
        assert (self.head-self.tail) == (self.max-self.size)
    if self.head == self.tail:
        assert (self.size==0) or (self.size==self.max)

def test():
    q = Queue(2)
    assert q
    q.checkRep()

    empty = q.empty()
    assert empty
    q.checkRep()

    full = q.full()
    assert not full
    q.checkRep()

    result = q.dequeue()
    assert result == None
    q.checkRep()

    result = q.enqueue(10)
    assert result == True
    q.checkRep()

    result = q.enqueue(20)
    assert result == True
    q.checkRep()

    empty = q.empty()
    assert not empty
    q.checkRep()

    full = q.full()
    assert full
    q.checkRep()

    result = q.enqueue(30)
    assert result == False
    q.checkRep()

    result = q.dequeue()
    assert result == 10
    q.checkRep()

    result = q.dequeue()
    assert result == 20
    q.checkRep()

test()

```

- It doesn't really take a huge amount of code to get full coverage of the queue class. In fact, what is done here is probably a bit much. It doesn't necessarily tell us a whole lot about whether the queue class is, in fact, correct in any meaningful way.

2. Splay tree coverage

- In the previous implementation there was a bug. Remove and splay functions are updated.

```
# TASK:
# We didn't achieve full statement coverage before, but we will now.

class Node:
    def __init__(self, key):
        self.key = key
        self.left = self.right = None

    def equals(self, node):
        return self.key == node.key

class SplayTree:
    def __init__(self):
        self.root = None
        self.header = Node(None) #For splay()

    def insert(self, key):
        if (self.root == None):
            self.root = Node(key)
            return

        self.splay(key)
        if self.root.key == key:
            # If the key is already there in the tree, don't do anything.
            return

        n = Node(key)
        if key < self.root.key:
            n.left = self.root.left
            n.right = self.root
            self.root.left = None
        else:
            n.right = self.root.right
            n.left = self.root
            self.root.right = None
        self.root = n
```

```

def remove(self, key):
    self.splay(key)
    # if key != self.root.key:
    #     raise 'key not found in tree'
    if self.root is None or key != self.root.key: # update
        return

    # Now delete the root.
    if self.root.left == None:
        self.root = self.root.right
    else:
        x = self.root.right
        self.root = self.root.left
        self.splay(key)
        self.root.right = x

def findMin(self):
    if self.root == None:
        return None
    x = self.root
    while x.left != None:
        x = x.left
    self.splay(x.key)
    return x.key

def findMax(self):
    if self.root == None:
        return None
    x = self.root
    while (x.right != None):
        x = x.right
    self.splay(x.key)
    return x.key

def find(self, key):
    if self.root == None:
        return None
    self.splay(key)
    if self.root.key != key:
        return None
    return self.root.key

def isEmpty(self):
    return self.root == None

```

```

def splay(self, key):
    l = r = self.header
    t = self.root
    if t is None: # update
        return # update
    self.header.left = self.header.right = None
    while True:
        if key < t.key:
            if t.left == None:
                break
            if key < t.left.key:
                y = t.left
                t.left = y.right
                y.right = t
                t = y
                if t.left == None:
                    break
            r.left = t
            r = t
            t = t.left
        elif key > t.key:
            if t.right == None:
                break
            if key > t.right.key:
                y = t.right
                t.right = y.left
                y.left = t
                t = y
                if t.right == None:
                    break
            l.right = t
            l = t
            t = t.right
        else:
            break
    l.right = t.left
    r.left = t.right
    t.left = self.header.right
    t.right = self.header.left
    self.root = t

def test():
    s = SplayTree()
    current_min = None
    current_max = None

    empty = s.isEmpty()
    assert empty == True
    _min = s.findMin()
    assert _min == None
    _max = s.findMax()
    assert _max == None

    found = s.find(10)
    assert found == None

```

```

s.insert(100)
current_min = 100
current_max = 100

for i in range(10,20):
    empty = s.isEmpty()
    assert empty == False

    s.insert(i)
    s.insert(i)

    if not current_min or i < current_min:
        current_min = i
    if not current_max or i > current_max:
        current_max = i

    found = s.find(i)
    assert found == i

    _min = s.findMin()
    assert _min == current_min

    _max = s.findMax()
    assert _max == current_max

for i in range(10,20):
    empty = s.isEmpty()
    assert empty == False

    s.remove(i)
    s.remove(i)

    found = s.find(i)
    assert found == None

s.insert(373)
s.remove(373)

test()

```

Notă¹

¹ © Copyright 2022 Lect. dr. Sorina-Nicoleta PREDUT
Toate drepturile rezervate.