Exam Simulation Solution

Scientific Programming Algolab

Wednesday 21 Dec 2016

1) Insertion sort solution

In [2]:

```python
import unittest

def insertion_sort(A):
    """ Sorts in-place list A with insertion sort.  """

    for i in range(1, len(A)):
        temp = A[i]
        j = i
        while j > 0 and A[j-1] > temp:
            A[j] = A[j-1]
            j -= 1
        A[j] = temp


class InsertionSortTest(unittest.TestCase):

    def test_zero_elements(self):
        v = []
        insertion_sort(v)
        self.assertEqual(v,[])

    def test_return_none(self):
        self.assertEquals(None, insertion_sort([2]))

    def test_one_element(self):
        v = ["a"]
        insertion_sort(v)
        self.assertEqual(v,["a"])

    def test_three_elements(self):
        v = [1,3,2]
        insertion_sort(v)
        self.assertEqual(v,[1,2,3])

    def test_two_elements(self):
        v = [2,1]
        insertion_sort(v)
        self.assertEqual(v,[1,2])

    def test_piccinno_list(self):
        v = [23, 34, 55, 32, 7777, 98, 3, 2, 1]
        insertion_sort(v)
        vcopy = v[:]
        vcopy.sort()
        self.assertEqual(v, vcopy)
```

2) Rev and copy Solution

```python
import unittest

class Node:
    """ A Node of an UnorderedList. Holds data provided by the user. """

    def __init__(self,initdata):
        self._data = initdata
        self._next = None

    def get_data(self):
        return self._data

    def get_next(self):
        return self._next

    def set_data(self,newdata):
        self._data = newdata

    def set_next(self,newnext):
        self._next = newnext


class UnorderedList:
    """
        This is a stripped down version of the UnorderedList seen in the lab

    """

    def __init__(self):
        self._head = None

    def to_python(self):
        """ Returns this UnorderedList as a regular Python list. This method
            is very handy for testing.
        """
        python_list = []
        current = self._head

        while (current != None):
            python_list.append(current.get_data())
            current = current.get_next()
        return python_list

    def __str__(self):
        current = self._head
        strings = []

        while (current != None):
            strings.append(str(current.get_data()))
            current = current.get_next()

        return "UnorderedList: " + ",".join(strings)


    def add(self,item):
        """ Adds item at the beginning of the list """

        new_head = Node(item)
        new_head.set_next(self._head)
```

```python
        self._head = new_head

    def remove(self, item):
        """ Removes first occurrence of item from the list

            If item is not found, raises an Exception.
        """
        current = self._head
        prev = None

        while (current != None):
            if (current.get_data() == item):
                if prev == None:  # we need to remove the head
                    self._head = current.get_next()
                else:
                    prev.set_next(current.get_next())
                    current = current.get_next()
                return  # Found, exits the function
            else:
                prev = current
                current = current.get_next()

        raise Exception("Tried to remove a non existing item! Item was: " + str(item
))


    def rev(self):
        """ Returns a *new* UnorderedList, which is the reversed version of this one
.

            Function must run in O(n), and try to make this function as fast as poss
ible,

            without using python lists or extra fields.

            Usage example:

            >>> lst = UnorderedList()
            >>> lst.add('c')
            >>> lst.add('b')
            >>> lst.add('a')
            >>> print lst
                UnorderedList: 'a','b','c'
            >>> print lst.rev()
                UnorderedList: 'c','b','a'
            >>> print lst
                UnorderedList: 'a','b','c'
        """

        ret = UnorderedList()

        current = self._head

        while current != None:
            ret.add(current.get_data())
            current = current.get_next()

        return ret

    def copy(self):
        """ Return a *copy* of this UnorderedList in O(n)
            NOTE: since we are making a copy, the output of this function
            won't contain any Node instance from the original list. Still, new Node
            instances will point to the same data items of the original list
```

```python
        Example (for more examples look at the tests):

        >>> orig = new UnorderedList()
        >>> orig.add('c')
        >>> orig.add('a')
        >>> print orig
            UnorderedList: 'a','c'
        >>> cp = orig.copy()
        >>> print cp
            UnorderedList: 'a','c'
        >>> cp.remove('c')
        >>> print cp
            UnorderedList: 'a'
        >>> print orig
            UnorderedList: 'a','c'
        """

        # this could be faster and occupy less memory, but it's still O(n)  ;-)
        return self.rev().rev()


class UnorderedListTest(unittest.TestCase):
    """ Test cases for UnorderedList

    """

    def test_init(self):
        ul = UnorderedList()

    def test_str(self):
        ul = UnorderedList()
        self.assertTrue('UnorderedList' in str(ul))
        ul.add('z')
        self.assertTrue('z' in str(ul))
        ul.add('w')
        self.assertTrue('z' in str(ul))
        self.assertTrue('w' in str(ul))



    def test_add(self):
        """ Remember 'add' adds stuff at the beginning of the list ! """

        ul = UnorderedList()
        self.assertEquals(ul.to_python(), [])
        ul.add('b')
        self.assertEquals(ul.to_python(), ['b'])
        ul.add('a')
        self.assertEquals(ul.to_python(), ['a', 'b'])

    def test_remove_empty_list(self):
        ul = UnorderedList()
        with self.assertRaises(Exception):
            ul.remove('a')

    def test_remove_one_element(self):
        ul = UnorderedList()
        ul.add('a')
        with self.assertRaises(Exception):
            ul.remove('b')
        ul.remove('a')
        self.assertEquals(ul.to_python(), [])

    def test_remove_two_element(self):
        ul = UnorderedList()
        ul.add('b')
```

```python
        ul.add('b')
        ul.add('a')
        with self.assertRaises(Exception):
            ul.remove('c')
        ul.remove('b')
        self.assertEquals(ul.to_python(), ['a'])
        ul.remove('a')
        self.assertEquals(ul.to_python(), [])


    def test_remove_first_occurrence(self):
        ul = UnorderedList()
        ul.add('b')
        ul.add('b')
        with self.assertRaises(Exception):
            ul.remove('c')
        ul.remove('b')
        self.assertEquals(ul.to_python(), ['b'])
        ul.remove('b')
        self.assertEquals(ul.to_python(), [])


    def test_rev_empty(self):
        ul = UnorderedList()

        self.assertEquals([], ul.to_python())
        self.assertEquals([], ul.rev().to_python())

    def test_rev_one(self):
        ul = UnorderedList()
        ul.add('a')
        self.assertEquals(['a'], ul.to_python())
        self.assertEquals(['a'], ul.rev().to_python())


    def test_rev_three(self):
        ul = UnorderedList()
        ul.add('c')
        ul.add('b')
        ul.add('a')
        self.assertEquals(['a','b','c'], ul.to_python())
        self.assertEquals(['c', 'b', 'a'], ul.rev().to_python())

    def test_copy_empty(self):
        orig = UnorderedList()

        cp = orig.copy()
        self.assertEquals([], cp.to_python())

        orig.add('a')
        self.assertEquals(['a'], orig.to_python())
        self.assertEquals([], cp.to_python())


    def test_copy_one(self):
        orig = UnorderedList()
        orig.add('b')

        cp = orig.copy()
        self.assertEquals(['b'], cp.to_python())

        orig.add('a')
        self.assertEquals(['a', 'b'], orig.to_python())
        self.assertEquals(['b'], cp.to_python())
        orig.remove('b')
```

```
        self.assertEquals(['a'], orig.to_python())
        self.assertEquals(['b'], cp.to_python())


    def test_copy_two(self):
        orig = UnorderedList()
        orig.add('c')
        orig.add('b')

        cp = orig.copy()
        self.assertEquals(['b', 'c'], cp.to_python())

        orig.add('a')
        self.assertEquals(['a', 'b', 'c'], orig.to_python())
        self.assertEquals(['b','c'], cp.to_python())

        orig.remove('c')
        self.assertEquals(['a', 'b'], orig.to_python())
        self.assertEquals(['b','c'], cp.to_python())
```

In [4]:

```
algolab.run(UnorderedListTest)
```

```
.............
----------------------------------------------------------------------
Ran 13 tests in 0.025s

OK
```

In [5]: