Algorithms and Data Structure 7

Yiping Deng

April 6, 2018

Problem 1

a)

The following is the concrete implementation in Python. Python is by-default generic, since it is not statically typed language.
Note: The python code includes a small unit test. You can simply run

```
$ python -m doctest -v stack.py
```

```
class StackNode:
    """
    A basic linked list implementation
    """
    def __init__(self, data = None, nxt = None):
        self.data = data
        self.next = nxt

class Stack:
    """
    Implementation of Stack According to the code in the assignment
    >>> s = Stack(2)
    >>> s.isEmpty()
    True
    >>> s.push(5)
    >>> s.isEmpty()
    False
    >>> s.current_size
    1
    >>> s.push(4)
    >>> s.current_size
    2
    >>> s.pop()
    4
    >>> s.pop()
    5
    >>> s.isEmpty()
    True
    """
    def __init__(self, size = None):
        if size is None:
            self.size = -1
        elif size <= 0:
            raise Exception('Invalid size')
        else:
            self.size = size

        self.current_size = -1
```

```
        self.top = None

    def push(self, data):
        # check the size
        if self.size != -1 and self.current_size == self.size:
            raise Exception('Stack Overflow Exception')

        if self.top is None:
            self.top = StackNode(data) #create a node
            self.current_size = 1 #set the proper size
        else:
            self.top = StackNode(data, self.top) #simply append the linked list
            self.current_size = self.current_size + 1 # increment

    def pop(self):
        if self.top is None:
            raise Exception('Stack Underflow Exception')

        val = self.top.data
        self.top = self.top.next # move to next one
        self.current_size = self.current_size - 1
        return val

    def isEmpty(self):
        return self.top is None
```

b)

We can implement a queue via two stacks. The first stack store the input, or freshly enqueued elements. The second stack store the output, or the elements that is ready is dequeue. When we try to dequeue, and if the second stack is empty, dump the first stack into the second one(poping from stack 1 and push into stack 2). The implementation in Python is also included here:

Note: The python code includes a small unit test. You can simply run

```
$ python -m doctest -v queue.py
```

```
from stack import Stack

class Queue:
    """
    A simple Queue using two stack
    >>> q = Queue()
    >>> q.enqueue(2)
    >>> q.enqueue(3)
    >>> q.enqueue(4)
    >>> q.dequeue()
    2
    >>> q.enqueue(5)
    >>> q.dequeue()
    3
    >>> q.dequeue()
    4
    >>> q.dequeue()
    5
    """
    def __init__(self):
        # initalize two Stack
        self.s1 = Stack()
        self.s2 = Stack()
```

```
    def enqueue(self, data):
        # just put it into stack 2
        self.s1.push(data)

    def prepare_dequeue(self):
        # dump stack 1 into stack 2
        if self.s2.isEmpty():
            while not self.s1.isEmpty():
                self.s2.push(self.s1.pop())

    def dequeue(self):
        self.prepare_dequeue() #prepare
        if self.s2.isEmpty():
            raise Exception('Queue Underflow Exception')
        return self.s2.pop()
```

Problem 2

a)