Practice Questions on Linear Data Structures (Week 6)

Tutorial Questions

1. Suppose that you are given the following sequence of letters.

   A R D G * O * C A R E L D O * * C F U L * * C L O C * K * * *

   a) If a letter means push and an asterisk means pop, give the sequence of letters returned by the pop operations when this sequence of operations is performed on an initially empty **stack**.
   b) If a letter means enqueue and an asterisk means dequeue, give the sequence of letters returned by the dequeue operations when this sequence of operations are performed on an initially empty **queue**.

2. Draw the queue **q** returned by the function below if the input is **n** = 6. Clearly indicate the head of the queue in your drawing.

```
def do_weird_stuff(n):
    q = Queue()
    for i in range(0, n):
        if i%2 == 1:
            x = q.dequeue()
            q.enqueue(x+i)
        else:
            q.enqueue(i)
    return q
```

3. Suppose that you can only push numbers from 1 to n onto a stack in increasing order (i.e. you can only push 3, after you have pushed 2. You can only push 2 only after you have pushed 1). At any point in time, you can call the pop operation and print the popped value.

   For example, the following operations:

```
s = Stack()
s.push(1)
print(s.pop())
s.push(2)
s.push(3)
print(s.pop())
s.push(4)
print(s.pop())
print(s.pop())
s.push(5)
print(s.pop())
```

   will print out the sequence 1 3 4 2 5.

   a) Give a list of operations (like above) that would result in the following sequence: 1 3 5 4 2
   b) Is there any sequence that cannot occur? If so, give an example and explain why.

4. Suppose you are given a stack **s** and a queue **q**.

   a) If initially **s** is empty and **q** is not empty, describe how to invert (reverse the ordering) the queue using the stack.
   e.g. **q** initially contains [(head)"a", "b", "c"(tail)]. After inverting **q**, **q** will contain: [(head)"c", "b", "a"(tail)]

   b) If initially **s** is not empty and **q** is empty, describe how to invert the stack using the queue.
   e.g. **s** initially contains [(top)"a", "b", "c"]. After inverting **s**, **s** will contain: [(top)"c", "b", "a"]

   If initially **s** is not empty and **q** is empty, describe how to delete an element in the stack using the queue. Other than the item being removed, the remaining content of the stack should remain in the same order after the removal.
   e.g. **s** initially contains [(top)"a", "b", "c"]. After deleting "b" from **s**, **s** will contain: [(top)"a", "c"]

5. The following is a recursive implementation of the factorial function. Convert it into a non-recursive implementation using a stack.

```
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n-1)
```

Extra Practice Questions

6. Draw the queue returned by the function below if the input **n** is 6. Clearly indicate the head of the queue in your drawing.

```
def do_strange_stuff(n):
    q = Queue()
    q.enqueue(0)
    for i in range(0, n):
        if i%2 == 0:
            x = q.dequeue()
            q.enqueue(i-x)
        else:
            q.enqueue(i)
    return q
```

7. Consider the following function:

```
01  def mystery(x):
02    s = Stack()
03    out = 0
04
05    while x > 0:
06      s.push(x % 2)
07      x = x // 2
08
09    while s.count() > 0:
10      out = out*10 + s.pop()
11    return out
```

   a) What will be returned by **mystery**(3)?
   b) What will be returned by **mystery**(5)?
   c) What does the **mystery** function do?

If you are able to answer (c),

d) **mystery**() above pushes **x%2** to the stack. Rewrite **mystery** so that **x//2** is pushed to the stack instead. What the function does remains the same.

8. Suppose that you can only enqueue numbers from 1 to n onto a queue in increasing order. At any point in time, you can call the dequeue operation and print the dequeued value.

   a) Give a list of queue operations that will produce the sequence: 1 2 3 4 5
   b) Is there any sequence that cannot occur? If so, give an example, and explain why.

9. You were given an example of how to write **is_palindrome(word)** using a queue and a stack in your slides. Now, design alternative implementations of **is_palindrome** using:

   a) two queues only
   b) one stack only

10. The following is a recursive implementation of the Dijkstra's algorithm to find the greatest common divisor of two numbers **a** and **b**. Convert it into a non-recursive implementation using stack.

```
def dijkstra(a, b):
  if a == b:
    return a
  if a > b:
    return dijkstra(a - b, b)
  else:
    return dijkstra(a, b - a)
```

~End