CS152: Data Structure

Week 5 Knowledge Check Solutions

Write the full name of all collaborators inside this box

Discuss these qustions and write your answers in the space provided below it.

1. Assume that the position of an item to be removed from a singly linked structure has been located. State the run-time complexity for completing the removal operation from that point.

Ans:

- The run-time complexity for removing an item from a singly linked structure after that item has been located is O(1).

2. Can a binary search be performed on items that are in sorted order within a singly linked structure? If not, why not?

Ans:

- It is not practical to perform a binary search for an item on a singly linked structure. The runtime cost of locating the midpoint would be linear, so the running time of such a search would be worse than that of a sequential search.

3. Suggest a good reason that Python list uses an array rather than a linked structure to hold its items

Ans:

- A Python list uses an array rather than a linked structure to hold its items because the running times of the access and replacement operations are O(1), and the memory usage is better when more than half of the array's positions are occupied.

4. What advantage does a circular linked structure with a dummy header node give the programmer?

Ans:

- A circular linked structure with a dummy header node enables the programmer to avoid special cases in the code for inserting or removing items at the head and tail of the structure.

5. Describe one benefit and one cost of a doubly linked structure, as compared to a singly linked structure.

Ans:

- One benefit of a doubly linked structure is that the operation to move to a previous item runs in constant time. One cost is that an extra memory cell is required for the second link in each node.

6. Suppose you start with an empty circular linked list that uses a dummy header node (as in the code below). You then call the `insert` method three times with the values 10, 20, and 30 (in that order):

```python
# Recall the same circular linked structure implementation shown in the slides
class Node:
    def __init__(self, data=None, next=None):
        self.data = data
        self.next = next

class CircularLinkedList:
    def __init__(self):
        # Create a dummy header node that points to itself
        self.header = Node()
        self.header.next = self.header

    def insert(self, data):
        # Insert new node at the end (before header)
        new_node = Node(data, self.header)
        probe = self.header
        while probe.next != self.header:
            probe = probe.next
        probe.next = new_node
# Using the circular linked structure
clist = CircularLinkedList()
clist.insert(10)
clist.insert(20)
clist.insert(30)
```

- Draw the resulting circular linked structure, clearly labeling:
  - The dummy header node,
  - All data nodes (with their values),
  - All next pointers.
  - Show the direction of the links and indicate which node is the header.

ANS

- Here's how the circular linked structure would look after inserting 10, 20, and 30 (in that order) using the provided insert method:
- Let's label the nodes:

```
H = header (dummy node, data=None)
N1 = node with data=10
N2 = node with data=20
N3 = node with data=30
The structure and pointers after all insertions:
```

```
+---------+      +---------+      +---------+      +---------+
| header  | ---> |   10    | ---> |   20    | ---> |   30    |
+---------+      +---------+      +---------+      +---------+
    ^                                                   |
    |                                                   |
    +---------------------------------------------------+
```