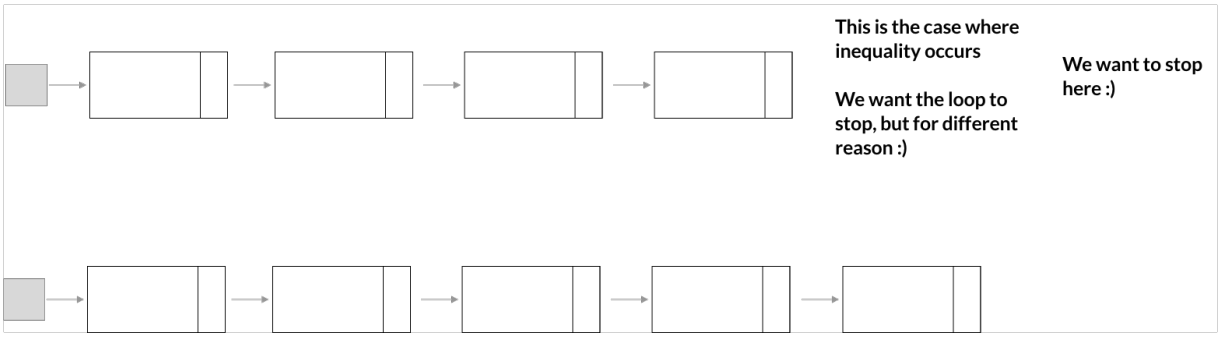
CSC148 Worksheet 13 Solution

Hyungmo Gu

April 24, 2020

Question 1

a. The following diagram tells us the stopping condition occurs when both *curr1* and *curr2* is *None*.



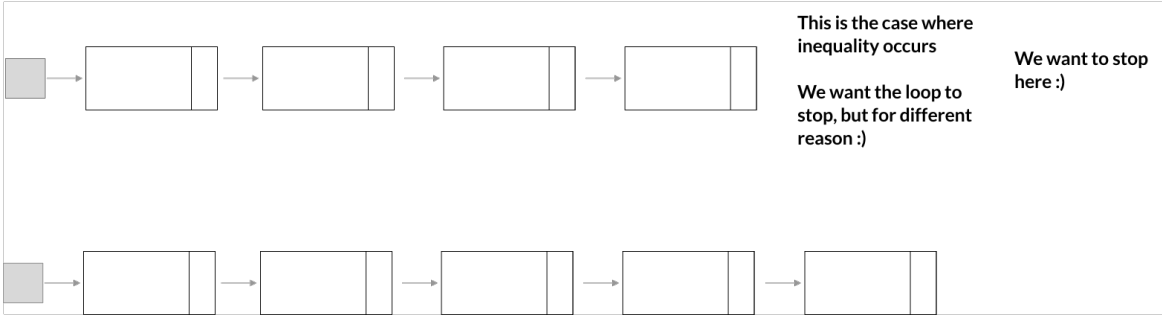
Using this fact, the python expression involving *curr1* and *curr2* that expresses the stopping condition is

```
    (curr1 is not None) and (curr2 is not None)

```

Correct Solution:

The following diagram tells us the stopping condition occurs when both *curr1* and *curr2* is *None*.



Using this fact, the python expression involving *curr1* and *curr2* that expresses the stopping condition is

```
    (curr1 is None) and (curr2 is None) # <- Correct Solution

```

b. Python expression for the while loop condition is

```
    while (curr1 is not None) and (curr2 is not None):
        ...

```

c. The code for traversing two list is

```
    while (curr1 is not None) and (curr2 is not None):
        if curr1 is None or curr2 is None:
            return False

        if curr1.item != curr2.item:
            return False

        curr1 = curr1.next
        curr2 = curr2.next

```

d. After the loop ends, we know all items in curr1 and curr2 are identical.

e. Because we know on successful loop termination, all items in curr1 and curr2 are the same, we can use this information to conclude the two linked lists have the same length.

f. The code that should go after the end of while loop is

```
    return True

```

Correct Solution:

The code that should go after the end of while loop is

```
        return curr1 is None and curr2 is None # <- Correct solution

```

```
def __eq__(self, other: LinkedList) -> bool:
    """Return whether this list and the other list are equal.

    >>> lst1 = LinkedList([1, 2, 3])
    >>> lst2 = LinkedList([])
    >>> lst1.__eq__(lst2)
    False
    >>> lst2.append(1)
    >>> lst2.append(2)
    >>> lst2.append(3)
    >>> lst1.__eq__(lst2)
    True
    """
    curr1 = self._first
    curr2 = other._first

    while (curr1 is not None) and (curr2 is not None):
        if curr1 is None or curr2 is None:
            return False

        if curr1.item != curr2.item:
            return False

        curr1 = curr1.next
        curr2 = curr2.next

    return curr1 is None and curr2 is None
```

Listing 1: worksheet_13_q2_solution.py

Question 2

a. Initially, *curr* and i are as follows

```
    curr = self._first
    i = 0

```

b. The stopping condition for the while loop is

```
    curr is not None

```

Using this fact, we can conclude that the while loop condition is

```
    while curr is not None:
        ...

```

Correct Solution:

The stopping condition for the while loop is

```
    (curr is None) or (i > index) # <- Correct solution

```

Using this fact, we can conclude that the while loop condition is

```
    while (curr is not None) and (i <= index): # <- Correct Solution
        ...

```

c. The code for the loop body is

```
    # 2. If index - 1 != current_index, then continue to next node
    if index - 1 != current_index:
        curr = curr.next
        current_index += 1
        continue

    # 3. If curr.next is none, then let it terminate naturally
    if curr.next is None:
        curr = curr.next
        current_index += 1
        continue

    # 4. If index - 1 == current_index, then return item of curr.next
    return curr.next.item

```

Correct Solution:

The code for the loop body is

```
    # 1. If index == 0, then return curr.item (edge case)
    if index == 0:
        return curr.item

    # 2. If index - 1 != i, then continue to next node
    if index - 1 != i:
        curr = curr.next
        i += 1
        continue

    # 3. If curr.next is none, then let it terminate naturally
    if curr.next is None:
        curr = curr.next
        i += 1
        continue

    # 4. If index - 1 == i, then return item of curr.next
    return curr.next.item

```

d. After the loop ends, we know *curr* is None and *index == len(self)*.

Using this fact, we can write that the post-loop code is

```
    raise IndexError

```

Correct Solution:

After the loop ends, we know *curr* is None or $i > index$.

Since both mean the index is out of bound, we can write that the post-loop code is

```
    raise IndexError

```

```
    def __getitem__(self, index: int) -> Any:
        """Return the item at position <index> in this list.
        Raise an IndexError if the <index> is out of bounds.
        Precondition: index >= 0.

        >>> lst = LinkedList([1, 2, 3])
        >>> print(lst[0])
```

```
        1
        >>> print(lst[1])
        2
        >>> print(lst[2])
        3
        >>> print(lst[3])
        Traceback (most recent call last):
            ...
        IndexError
        """

        curr = self._first
        i = 0

        while (curr is not None) and (i <= index):
            # 1. If index == 0, then return curr.item (edge case)
            if index == 0:
                return curr.item

            # 2. If index - 1 != i, then continue to next node
            if index - 1 != i:
                curr = curr.next
                i += 1
                continue

            # 3. If curr.next is none, then let it terminate naturally
            if curr.next is None:
                curr = curr.next
                i += 1
                continue

            # 4. If index - 1 == i, then return item of curr.next
            return curr.next.item

        raise IndexError
```

Listing 2: worksheet_13_q2_solution.py