COMPSCI 105 S1 2017
Principles of Computer Science

18 Linked List(2)

Agenda & Readings

- Agenda
  - The UnorderedList ADT version 2
  - The OrderedList ADT
  - Linked List Iterators
- Reference:
  - Textbook:
    - Problem Solving with Algorithms and Data Structures
      - Chapter 3 – Lists
      - Chapter 3 – The OrderedList Abstract Data Type

UnorderedList Version2

- We can add a count variable to count the number of items in the list

```
class UnorderedListV2:
    def __init__(self):
        self.head = None
        self.count = 0
```

```
def add(self, item):
    new_node = Node(item)
    ...
    self.count += 1
```

```
def remove(self, item):
    current = self.head
    ...
    self.count -= 1
```


UnorderedList Version2

- We can add a count variable to count the number of items in the list

```
def size(self):
    return self.count

def is_empty(self):
    return self.count == 0
```

Time complexity: O(1)

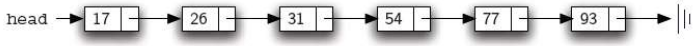
Time Complexity

- Summary

	Python List		UnorderedList
if len(my_plist)== 0: …	***O(1)***	**is_empty**	***O(1)***
len	***O(1)***	**size**	***O(1)*** *with count variable* ***O(n)*** *without count variable*
append (end of the python List) insert (i, item)	***O(1)*** ***O(n)***	**add**	***O(1)*** *(beginning of the linked list)*
remove del	***O(n)*** ***O(n)***	**remove**	***O(n)***
in	***O(n)***	**search**	***O(n)***

The Ordered List

- Unordered Vs Ordered
  - A list is a collection of items where each item holds a relative position with respect to the others
    - It must maintain a reference to the first node (head)
    - It is commonly known as a linked list
  - Unordered Vs Ordered
    - Unordered meaning that the items are not stored in a sorted fashion
    - The structure of an ordered list is a collection of items where each item holds a relative position that is based upon some underlying characteristic of the item
  - Ordered linked-list example:



Operations

- List()
  - Creates a new list that is empty
  - It needs no parameters and returns an empty list
- add(item)
  - Adds a new item to the list
  - It needs the item and returns nothing
  - **Assume** the item **is not already** in the list
- remove(item)
  - Removes the item from the list
  - It needs the item and modifies the list
  - **Assume** the item is **present** in the list

No checking is done in the implementation

Operations

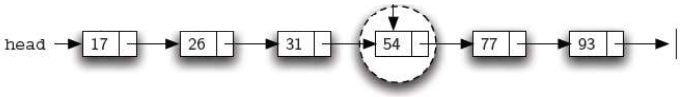
- search(item)
  - Searches for the item in the list
  - It needs the item and **returns** a boolean value
- is_empty()
  - Tests to see whether the list is empty
  - It needs no parameters and **returns** a boolean value
- size()
  - Returns the number of items in the list
  - It needs no parameters and **returns** an integer

Implementation - Add

- Determine the point of insertion
  - Starting point:
    - current = self.head
    - previous = None
    - stop = False

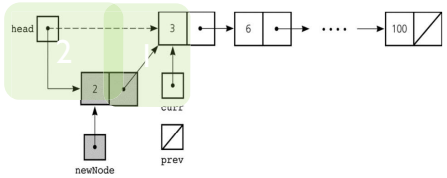
```
while current != None and not stop:
    if current.get_data() > item:
        stop = True
    else:
        previous = current
    current = current.get_next()
```

```
my_orderedlist.add(49)
```



Implementation - Add

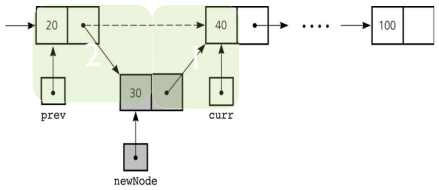
- Insert at the beginning of a linked list




```
1 new_node.set_next(self.head)
2 self.head = new_node
```


Implementation - Add

- Insert at the middle of a linked list
  - Change the next reference of the new node to refer to the current node of the list
  - Modify the next reference of the previous node to refer to the new node

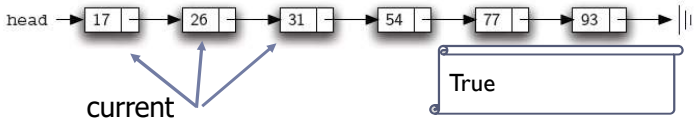



```
1 new_node.set_next(current)
2 previous.set_next(new_node)
```

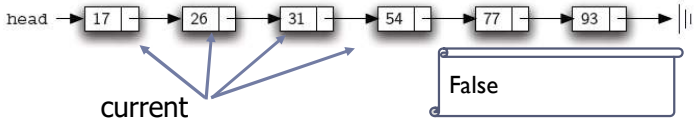

Implementation - Search

- Searches for the item in the list
  - Returns a Boolean
  - Examples:

```
print (my_linked_list.search(31))
```



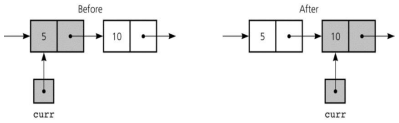

```
print (my_linked_list.search(39))
```



Implementation - Search

- To search an item in a linked list:
  - Set a pointer to be the same address as head, process the data in the node, (search) move the pointer to the next node, and so on
  - Loop stops either
    - Found the item
    - The next pointer is None
    - The value in the node is greater than the item that we are searching

```
current = self.head
while current != None:
    if current.get_data() == item:
        return True
    elif current.get_data() > item:
        return False
    else:
        current = current.get_next()
return False
```



Time Complexity

- Summary

	UnorderedList	OrderedList
is_empty	***O(1)***	***O(1)***
size	***O(1)*** *with count variable*	***O(1)*** *with count variable*
add	***O(1)***	***O(n)***
remove	***O(n)***	***O(n)***
search	***O(n)***	***O(n)***

Exercise 1

- What is the output of the following program? (You should be able to draw the linked list after every line of code)

```
my_list = OrderedList()
my_list.add(30)
my_list.add(10)
my_list.add(80)
my_list.remove(30)
my_list.add(56)
print(my_list.size())
```


Iterators

- Traversals are very common operations, especially on containers
- Python's for loop allows programmer to traverse items in strings, lists, tuples, and dictionaries:

```
for <eachItem> in <collection>:
    <do something with eachItem>
```

  - Lists

```
for item in [1, 2, 3, 4]:
    print(item)
```

  - Tuples

```
for item in (1, 2, 3):
    print(item)
```

  - Dictionaries

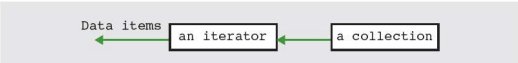
```
for key in {'a': 1, 'b': 2, 'c': 3}:
    print(key)
```

  - Strings

```
for char in 'hello':
    print(char)
```

Iterators

- Python compiler translates for loop to code that uses a special type of object called an iterator



- An iterator guarantees that each element is visited exactly once
  - It is useful to be able to traverse an UnorderedList or an OrderedList, i.e., visit each element exactly once
- To explicitly create an iterator, use the built-in iter function:

```
i = iter(num_list)
print(next(i)) # fetch first value
print(next(i))
```

```
1
2
```


Iterators

- You can create your own iterators if you write a function to generate the next item
- You need to add:
  - Constructor
  - The __iter__() method, which must return the iterator object
  - The __next__() method, which returns the next element from a sequence.
- For example:

```
my_iterator = NumberIterator(11, 20)
for num in my_iterator:
    print(num, end=" ")
```

```
11 12 13 14 15 16 17 18 19 20
```


Iterators

- The NumberIterator Class

```
class NumberIterator:
    def __init__(self, low, high):
        self.current = low
        self.high = high
    def __iter__(self):
        return self
    def __next__(self):
        if self.current > self.high:
            raise StopIteration
        else:
            self.current += 1
            return self.current - 1
```

Raise this error to tell consumer to stop

Linked List Traversals

- Now, we would like to traverse an UnorderedList or an OrderedList using a for-loop, i.e., visit each element exactly once

```
for num in my_linked_list:
    print(num, end=" ")
```

- However, we will get the following error:

```
for num in my_linked_list:
TypeError: 'UnorderedList' object is not iterable
```

- Solution:
  - Create an iterator class for the linked list
  - Add the __iter()__ method to return an instance of the LinkedListIterator class

The LinkedListIterator

- The UnorderedList class:

```
from Node import Node
from LinkedListIterator import LinkedListIterator
class UnorderedList:
 ...
    def __iter__(self):
        return LinkedListIterator(self.head)
```

```
class LinkedListIterator:
    def __init__(self, head):
        self.current = head
    def __next__( self ):
        if self.current != None:
            item = self.current.get_data()
            self.current = self.current.get_next()
            return item
        else:
            raise StopIteration
```


The LinkedListIterator

- Example:

```
def TestListIterator():
    my_linked_list = UnorderedList()
    num_list = [24, 65, 12]
    for num in num_list:
        my_linked_list.add(num)

    for num in my_linked_list:
        print(num, end=" ")
```

```
12 65 24
```


Exercise 2

- What is the output of the following program?

```
name_list = ["Gill", "Tom", "Eduardo", "Raffaele", "Serena", "Bella"]
my_unorderedlist = OrderedList()
for name in name_list:
    my_unorderedlist.add(name)
for name in my_unorderedlist:
    print(name, end=" ")
```


Applications - Sum

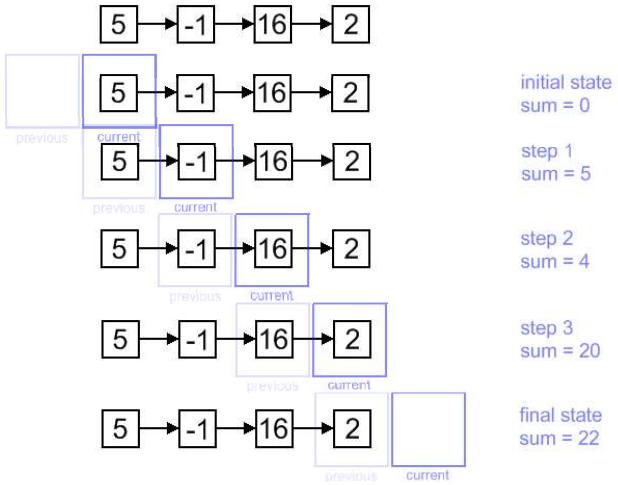
- A linked list containing some numbers:

```
my_list = UnorderedList()
my_list.add(1)
my_list.add(3)
my_list.add(5)
```

- How do we calculate the sum of the values in the above linked list?
  - Case 1: Adding numbers using a for loop
  - Case 2: Adding numbers using a for loop and iterators
  - Case 3: Add a getSum() method within the linked list class itself

Applications - Sum

- Calculate the sum of values



Applications - Sum

- Case 1: Using a for loop (without iterators) => ERRORS

```
my_linked_list = OrderedList()
num_list = [1, 3, 5]
for num in num_list:
    my_linked_list.add(num)   #adding numbers to the linked list
    total = 0
for num in my_linked_list:
    total += num
```

TypeError: 'OrderedList' object is not iterable

Applications - Sum

- Case 1: Using a for loop and iterators => OK

```
my_linked_list = OrderedList()
num_list = [1, 3, 5]
for num in num_list:
    my_linked_list.add(num)   #adding numbers to the linked list
    total = 0
for num in my_linked_list:
    total += num
```

  - The LinkedListIterator class contains the following methods:
    - __iter__(): this returns the actual iterator object
    - __next__(): this method is called on the iterator object, and it returns the next item in the collection

```
class LinkedListIterator:
 ...
    def __next__( self ):
     ...
    return item
```


Applications - Sum

- Case 3: Add a getSum() method inside the class itself
  - Set total = 0
  - Initialize current to the head of the list
  - Check that current is not NULL
    - If it is, there is nothing more to do (Stop)
  - Take the value from current and add it to the total
  - Set current equal to the next node in the list
  - Go to step 3

Applications - Sum

- Case 3: Add a getSum() method inside the class itself

```
class UnorderedList:
  ...
   def getSum(self):
      total = 0
      current = self.head
      while current != None:
         total = total + current.get_data()
         current = current.get_next()
      return total
```


Summary

- Different implementations have different complexity
- The linked-list can be in order