Lecture 6 - Linked Lists

1. Review

A **data structure** is a way to store data with a collection of supported operations that allow us to manipulate that data.

The collection of supported operations is called an **interface** (Abstract Data Type).

The difference between a data structure and its interface can be thought of as follows:

- An **interface** is a specification (what the data structure does)
- A **data structure** is a concrete implementation (how it's done!).

1.1. Sequence Interface

- Maintains a sequence of n items, e.g. `34,25,35,54` or `"sam", "joe", 1`
- Supports sequence operations

1.1.1. Sequence Operations

Note that this is not written as Python, we are just describing the type of things (operations) we should be able to do to the sequence

Name	Description
`create(X)`	create a sequence from items in `X`
`size()`	return the length of the sequence
`get(i)`	return the item at index `i`
`set(i,x)`	replace the item at index `i` with `x`
`insert(i,x)`	add `x` to position `i` (this will move all previous items at index `i`,`i+1`,... etc up 1)
`delete(i)`	delete the item at index `i` (this will move all previous items at index `i`,`i+1`,... etc down 1)
`insert_first(x)`	add `x` as the first item. Same as `insert(0,x)`
`delete_first()`	delete the first item. Same as `delete(0)`
`insert_last(x)`	add `x` as the last item. Same as `insert(size(),x)`
`delete_last()`	delete the last item. Same as `delete(size()-1)`

So far we have seen two types of sequence and analysed their worse-case performance.

Data Structure	`create(X)`	`get(i)` `set(i,x)`	`insert(i,x)` `delete(i)`	`insert_first(i,x)` `delete_first()`	`insert_last(i,x)` `delete_last()`
Array	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$

Data Structure	`create(X)`	`get(i)` `set(i,x)`	`insert(i,x)` `delete(i)`	`insert_first(i,x)` `delete_first()`	`insert_last(i,x)` `delete_last()`
Dynamic Array	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$**

Worse-case Complexity
**Amoritized Time

2. Linked Lists

https://visualgo.net/en/list

3. Python Code

```
# Linked list implementation in Python
# Programiz https://www.programiz.com/dsa/linked-list

class Node:
  """ Represents a single node"""
  def __init__(self, item):
      self.item = item
      self.next = None

class LinkedList:
  """ The whole linked list"""
  def __init__(self):
      self.head = None

if __name__ == '__main__':

  linked_list = LinkedList()

  # Assign item values
  linked_list.head = Node(1)
  second = Node(2)
  third = Node(3)

  # Connect nodes
  linked_list.head.next = second
  second.next = third

  # Print the linked list item
  current_node = linked_list.head                # set current node to head
  # find the last node
  while current_node.next != None:        # while the current node isn' the last
node
      print(current_node.item)
      current_node = current_node.next    # set current node to next node
  print(current_node.item)
```

```
# Linked list implementation in Python

class Node:
  # Creating a node
  def __init__(self, item):
      self.item = item
      self.next = None

class LinkedList:
  def __init__(self):
      self.head = None
      self._size = 0

  def create(self, X):                             # O(n)
      for item in reversed(X):
          self.insert_first(item)

  def insert_first(self, x):                       # O(1)
      new_node = Node(x)                           # create a new node with item x
      new_node.next = self.head                    # set the new node to point to the
current head
      self.head = new_node                         # replace the linked list head to be
new node
      self._size += 1                              # increase the size

if __name__ == '__main__':

  linked_list = LinkedList()

  # Assign item values
  linked_list.insert_first(1)
  linked_list.insert_first(2)
  linked_list.insert_first(3)

  # Print the linked list item
  current_node = linked_list.head                     # set current node to head
  # find the last node
  while current_node.next != None:        # while the current node isn' the last
node
      print(current_node.item)
      current_node = current_node.next    # set current node to next node
  print(current_node.item)
```

4. Review

4.1. Summary Table

5. Linked Lists

6. Idea

7. Operations

8. Analysis

9. Implementation in Python

10. Summary

Data Structure	`create(X)`	`get(i)` `set(i,x)`	`insert(i,x)` `delete(i)`	`insert_first(i,x)` `delete_first()`	`insert_last(i,x)` `delete_last()`	**Space**
Array	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Dynamic Array	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$**	$O(n)$
Singly Linked List	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$
Doubly Linked List	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$