Week 2 - Linked Lists

Brenden West - 2023

Contents

Learning Outcomes

- Linked Lists

Reading & Videos

- *LaFore - Ch. 5*
- https://www.geeksforgeeks.org/data-structures/linked-list/ (review)

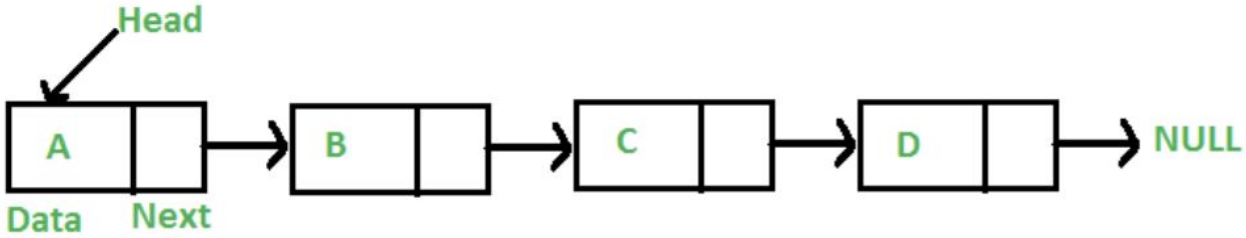
Linked List

Linked Lists are a common alternative to arrays for structuring a data collection and are independent of any specific programming language.

Each item in a linked list is an object with data and a **pointer** to the next item in the list. Lists are **traversed** by following these pointers from the **head** item.

Linked lists have some key differences from arrays:

- Items are not stored in contiguous memory locations
- The list can be defined without knowing the list size
- List items do not have an **index position** and can only be accessed by traversing the list
- List items can be added or removed more easily



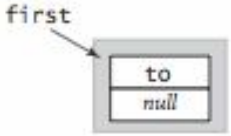
Constructing a Linked List

To build a linked list, we

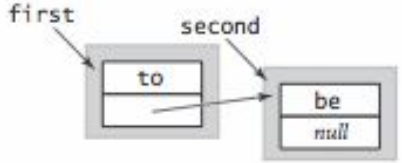
- Define a Node object
- Create a `Node` for each item,
- Set the item field to the desired value
- Set the **`next`** field to the next node in the list

```
public class Main {
  private static class Node {
        private String item;
        private Node next;
  }

  public static void main(String[]
args) {

      // list operations
  }
}
```

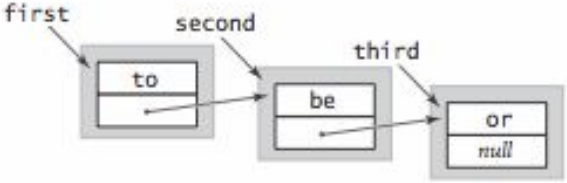
```
Node first  = new Node();
first.item  = "to";
```




```
Node second = new Node();
second.item = "be";
first.next  = second;
```




```
Node third  = new Node();
third.item  = "or";
second.next = third;
```



Traversing a Linked List

Linked lists are traversed from the head node until reaching the desired node or the end of the list.

Traversing can use a **loop** or a **recursive** method.

```
    Node first = new Node();
    first.item = "to";

    Node second = new Node();
    second.item = "be";
    first.next = second;

    Node current = first;
    while (current != null) {

System.out.println(current.item);
        current = current.next;
    }
```

Inserting items into a Linked List

Inserting a list item involves creating a new item and setting values for this and related list items.

The process differs slightly for these scenarios:

- Insert at start - time complexity = O(1)
- Insert at end - time complexity = O(n)
- Insert in the middle

```
Node newItem = new Node();

// insert item at start
newItem.next = first;
first = newItem;

// insert item at end
Node current = first;
while (current != null) {
  current = current.next;
}
current.next = newItem
```

Inserting an item after a specific node

Inserting an item in the middle of a linked list requires that you identify the node it should come after.

Steps are:

- Find the **target** node to insert after
- Keep a reference to target's **next** node
- Reset the target's next to the new node
- Reset the new node's next to the old next node

```
Node newItem = new Node();

// insert after target node
String target = "target value";
Node current = first;
while (current != null) {
  if (current.item == target) {
        Node oldNext = current.next;
        current.next = newItem;
        newItem.next = oldNext;
        break;
  }
  current = current.next;         }
```

Deleting from a Linked List

As with insertion, the deletion process differs slightly for these scenarios:

- Delete at start - time complexity = O(1)
- Delete at end - time complexity = O(n)
- Delete in the middle

```
// delete item at start
Node temp = first;
first = temp.next;
temp = null; // free memory

// delete item at end
Node current = first;
Node previous = null;
while (current.next != null) {
  previous = current
  current = current.next;
}
previous.next = null;
```

Deleting from a Linked List

Deleting an item in the middle of a linked list requires that you identify the target node and its predecessor.

```
// delete the target node
String target = "inserted";
Node current = first;
Node previous = null;
while (current != null) {
  if (current.item == target) {
    if (previous == null) { // target
is head
      Node temp = current;
      first = temp.next;
      temp = null; // free memory
      break;
    } else {
      previous.next = current.next;
      current = null;
      break;
     }
    }
    previous = current;
    current = current.next;
}
```