Linked Lists

Robert Horvick
SOFTWARE ENGINEER

@bubbafat www.roberthorvick.com

Overview

What are linked lists?

- Nodes
- Node chains

Singly linked lists

Doubly linked lists

Sorted linked lists

Demo: Updating the contact manager

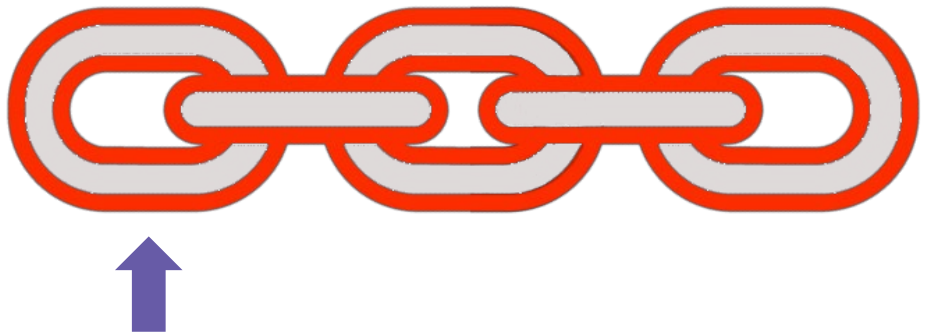
Linked List

A container where data is stored in nodes consisting of a single value item and a reference to the next node.

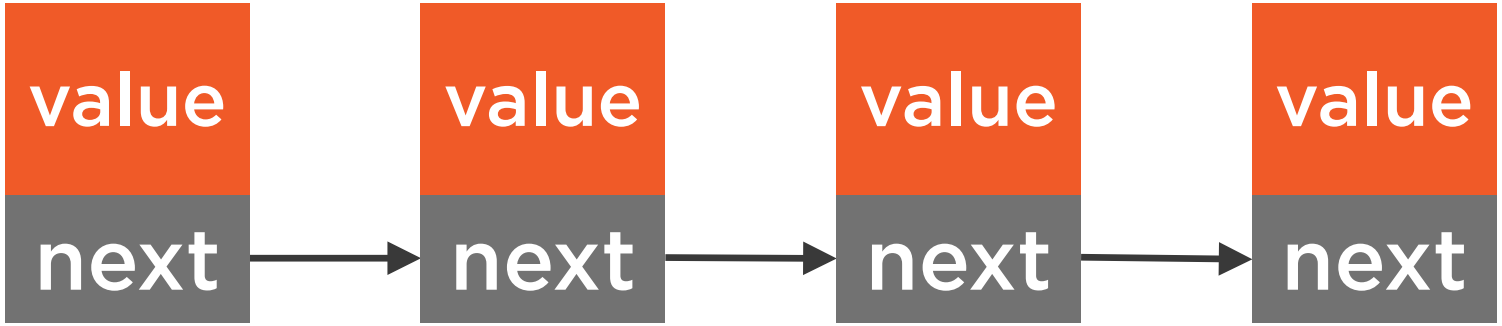
Similar to a chain

Start at the first link

Follow the chain to the last link



The Node



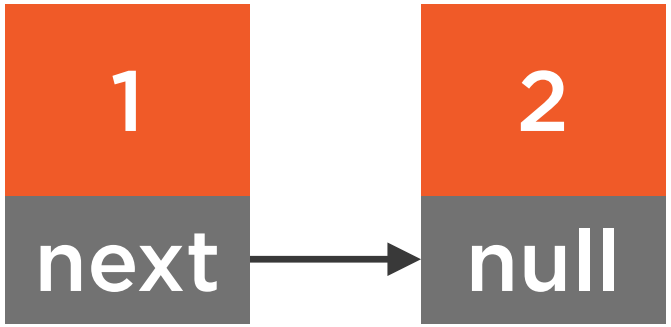
```
class Node
{
    public Node(int value)
    {
        this.Value = value;
        this.Next = null;
    }

    public Node Next;
    public int Value;
}
```

```
Node head = new Node(1);
```

Connecting Nodes into a List

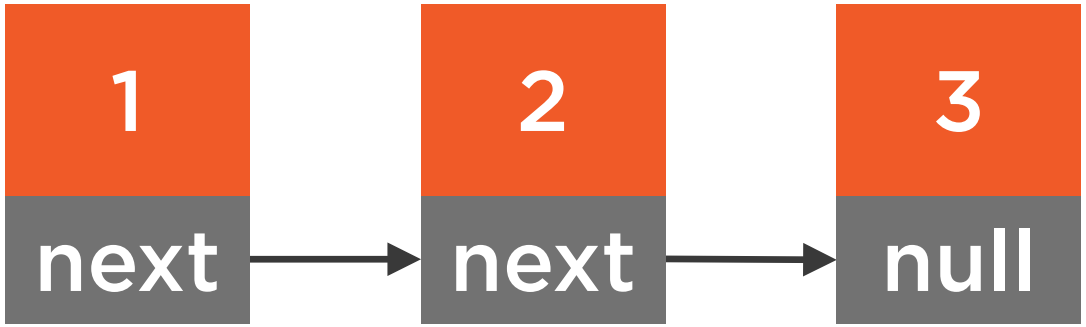
```
Node head = new Node(1);

head.Next = new Node(2);
```

Connecting Nodes into a List



```
Node head = new Node(1);

head.Next = new Node(2);

head.Next.Next = new Node(3);
```

Connecting Nodes into a List



Singly Linked List

A linked list that provides forward iteration from the start to the end of the list.

```
class LinkedListNode<TNode> {

    public LinkedListNode(TNode value, LinkedListNode<TNode> next = null) {
        this.Value = value;
        this.Next = next;
    }

    public LinkedListNode<TNode> Next;
    public TNode Value;
}
```

Singly Linked List Node

A generic class containing the data and reference to the next node

Doubly Linked List

A linked list that provides forward iteration from the start to the end of the list, and reverse iteration, from end to start.

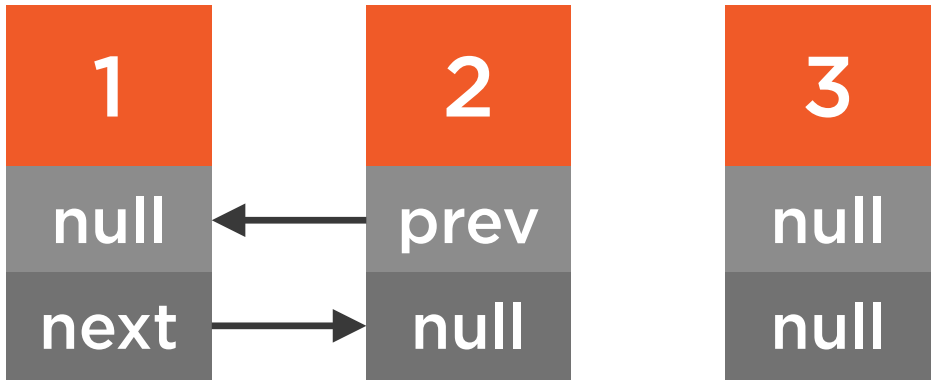
```
class Node
{
    public Node(int value)
    {
        this.Value = value;
        this.Previous = null;
        this.Next = null;
    }

    public Node Previous;
    public Node Next;
    public int Value;
}
```

```
Node node1 = new Node(1);

Node node2 = new Node(2);

Node node2 = new Node(3);
```

Connecting Doubly Linked Nodes Into a List

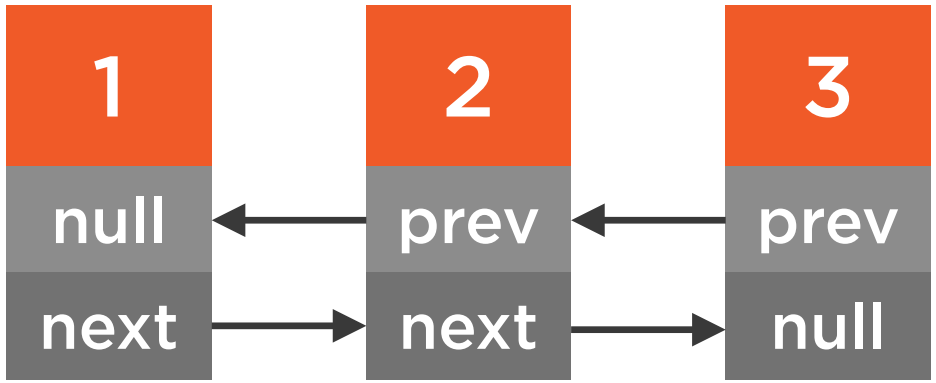
```
node1.Next = node2;

node2.Previous = node1;
```

Connecting Doubly Linked Nodes Into a List



```
node2.Next = node3;

node3.Previous = node2;
```

Connecting Doubly Linked Nodes Into a List



Doubly Linked List Node

```
class DoublyLinkedListNode<TNode>
{
    public DoublyLinkedListNode(TNode value,
                                Node<TNode> prev = null,
                                Node<TNode> next = null) {
        this.Value = value;
        this.Previous = prev;
        this.Next = next;
    }

    public Node<TNode> Previous;
    public Node<TNode> Next;
    public TNode Value;
}
```

Construction

```
public class DoublyLinkedList<T> : IEnumerable<T>
{
   Node<T> head = null;
   Node<T> tail = null;

   public int Count
   {
      get;
      private set;
   }
}
```

Doubly Linked List Class

A generic class containing references to the first (head) and last (tail) nodes

Adding Items

Adding Items

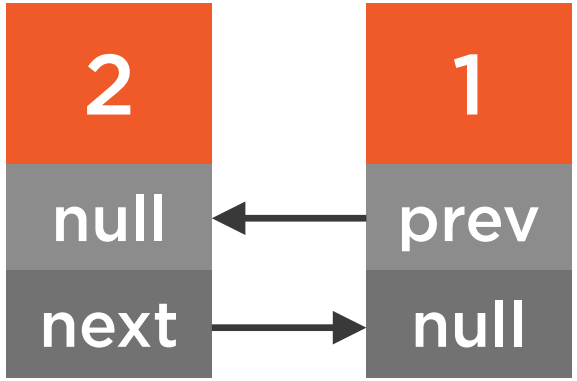
Function	Behavior	Complexity
AddHead	Adds a value to the beginning of the list	O(1)
AddTail	Adds a value at the end of the linked list	O(1)

```
DoublyLinkedList<int> ints = new DoublyLinkedList<int>();

for (int i = 1; i <= 5; i++)

{

    ints.AddHead(i);

}
```

Adding Values Using AddHead

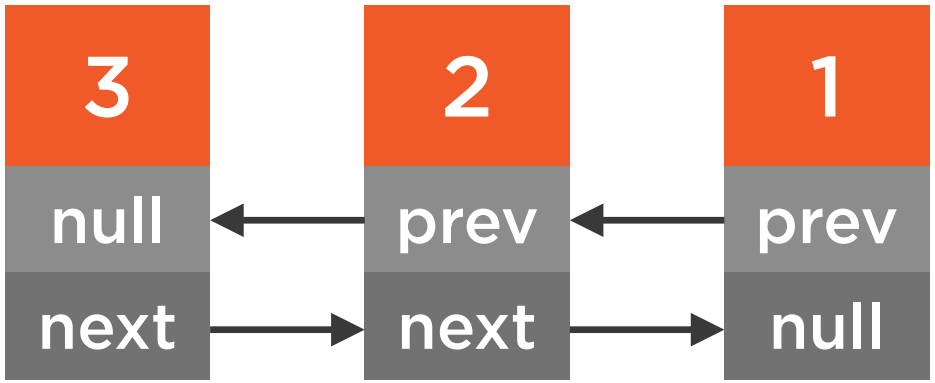
```
DoublyLinkedList<int> ints = new DoublyLinkedList<int>();

for (int i = 1; i <= 5; i++)

{

    ints.AddHead(i);

}
```

Adding Values Using AddHead



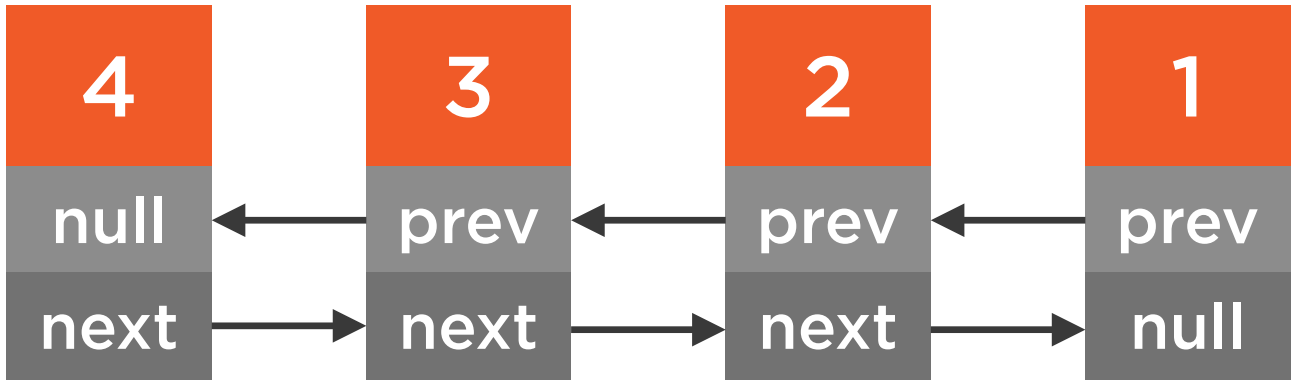
```
DoublyLinkedList<int> ints = new DoublyLinkedList<int>();

for (int i = 1; i <= 5; i++)

{

    ints.AddHead(i);

}
```

Adding Values Using AddHead



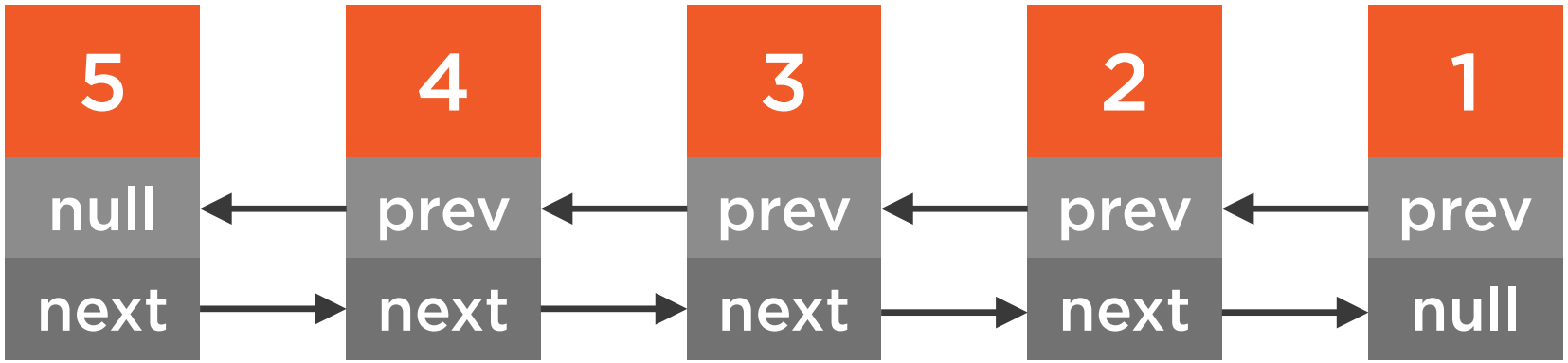
```
DoublyLinkedList<int> ints = new DoublyLinkedList<int>();

for (int i = 1; i <= 5; i++)

{

    ints.AddHead(i);

}
```

Adding Values Using AddHead



```
DoublyLinkedList<int> ints = new DoublyLinkedList<int>();

for (int i = 1; i <= 5; i++)

{

   ints.AddHead(i);

}
```

Adding Values Using AddHead



```
public void AddHead(T value)
{
   DoublyLinkedListNode<T> adding = new DoublyLinkedListNode
<T>(value, null, head);

   if(head != null)
   {
      head.Previous = adding;
   }

   head = adding;

   if (tail == null)
   {
      tail = head;
   }

   Count++;
}
```

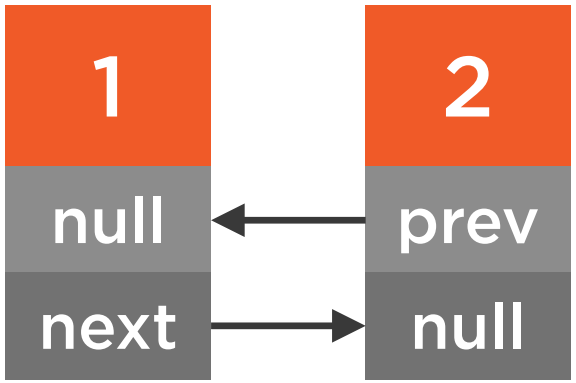
- Adding a value to the head (start) of the list
- Allocate the new node
- If there was an existing head node
- Update its previous pointer to the new node
- Set the head pointer to the new node
- If the list was empty (tail is null)
- Then the head and tail are the same
- Increment the Count value (items in the list)

```
DoublyLinkedList<int> ints = new DoublyLinkedList<int>();

for (int i = 1; i <= 5; i++)

{

    ints.AddTail(i);

}
```

Adding Values Using AddTail

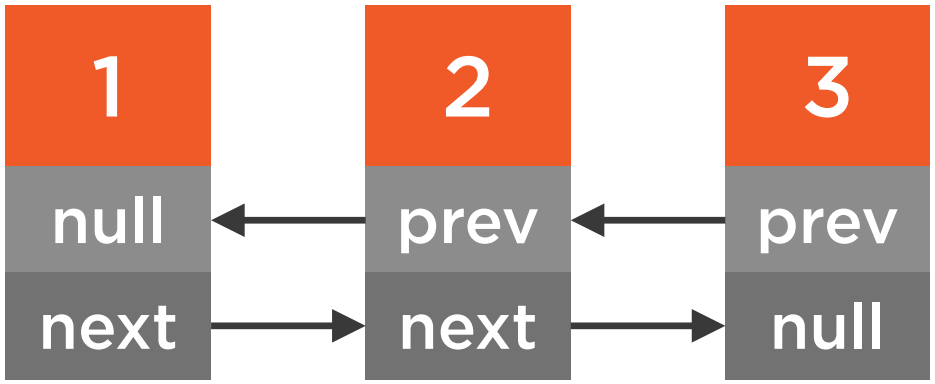
```
DoublyLinkedList<int> ints = new DoublyLinkedList<int>();

for (int i = 1; i <= 5; i++)

{

    ints.AddTail(i);

}
```

Adding Values Using AddTail



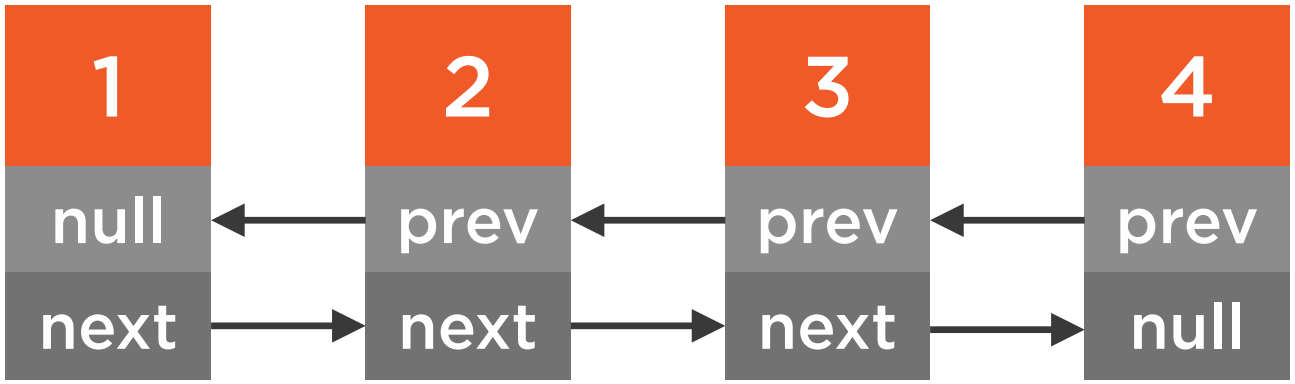
```
DoublyLinkedList<int> ints = new DoublyLinkedList<int>();

for (int i = 1; i <= 5; i++)

{

    ints.AddTail(i);

}
```

Adding Values Using AddTail



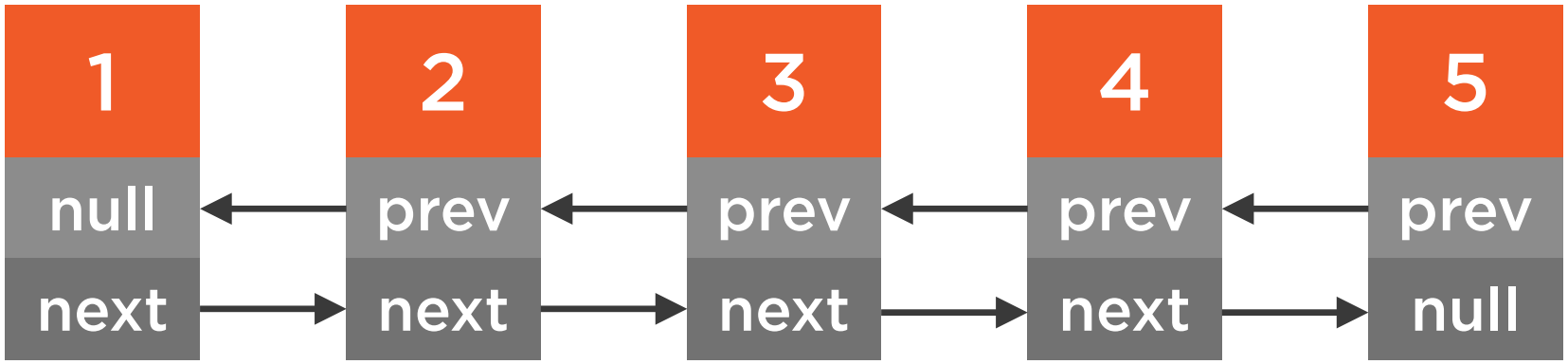
```
DoublyLinkedList<int> ints = new DoublyLinkedList<int>();

for (int i = 1; i <= 5; i++)

{

    ints.AddTail(i);

}
```

Adding Values Using AddTail



```
DoublyLinkedList<int> ints = new DoublyLinkedList<int>();

for (int i = 1; i <= 5; i++)

{

    ints.AddTail(i);

}
```

Adding Values Using AddTail



```
public void AddTail(T value)
{
   if (tail == null)
   {
      AddHead(value);
   }
   else
   {
      DoublyLinkedListNode<T> adding = new
DoublyLinkedListNode<T>(value, tail);

      tail.Next = adding;

      tail = adding;

      Count++;
   }
}
```

- Adding a value to the tail (end) of the list
- If the list is empty defer to AddHead
- Allocate the node being added
- Point the current tail's next to the new node
- Set the list tail to the new node
- Increment the Count value (items in the list)

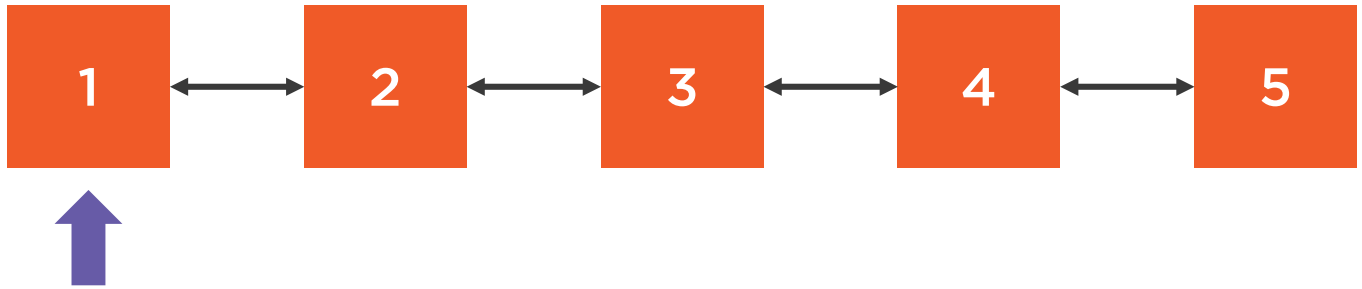
Finding Items

Finding Items

Function	Behavior	Complexity
Find	Finds the first node whose value equals the provided argument	O(n)
Contains	Returns true if the specified value exists in the list, false otherwise	O(n)

```
private LinkedListNode<T> Find(T value)
{
   LinkedListNode<T> current = head;

   while(current != null) {
      if(current.Value.Equals(value))
         return current;

      current = current.Next;
   }

   return null;
}
```

Finding Values in the List



```
public bool Contains(T value)
{
    return Find(value) != null;
}
```

Determine if the List Contains a Value

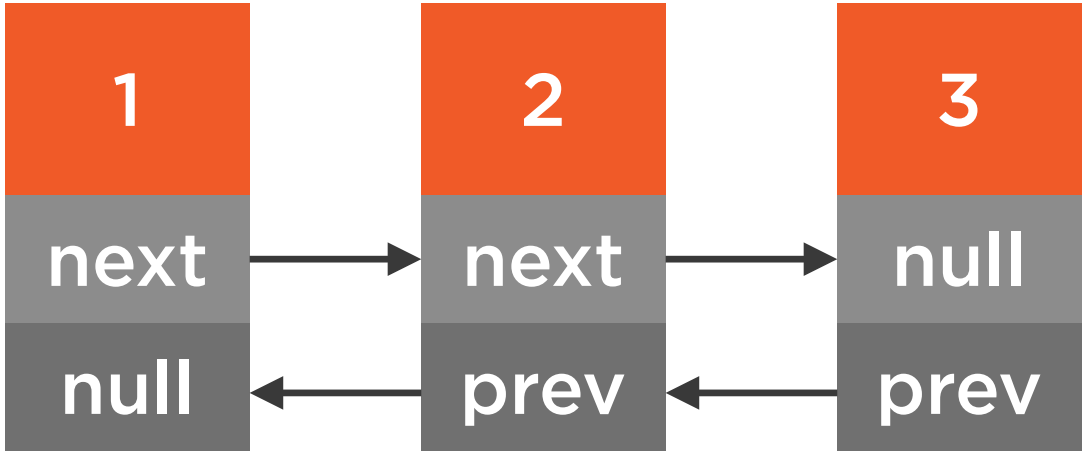
Return true if the value is found, false otherwise

Removing Items

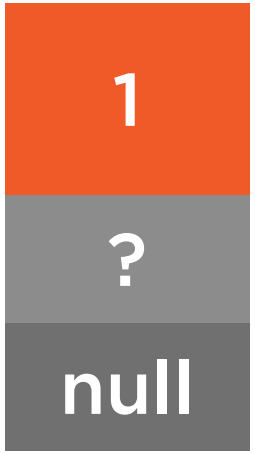
Removing Items

Function	Behavior	Complexity
Remove	Removes the first node whose value is equal to the argument	O(n)

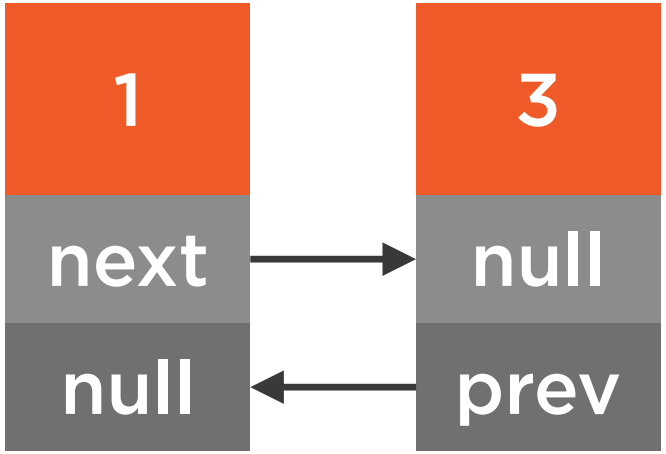
Removing a Node



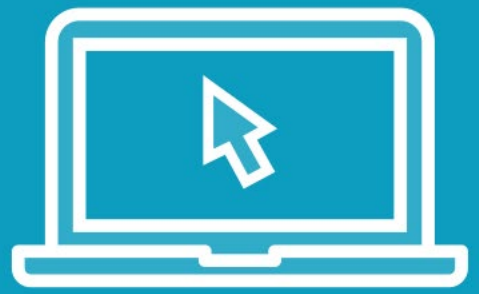
Removing a Node



Removing a Node



Demo



Remove algorithm

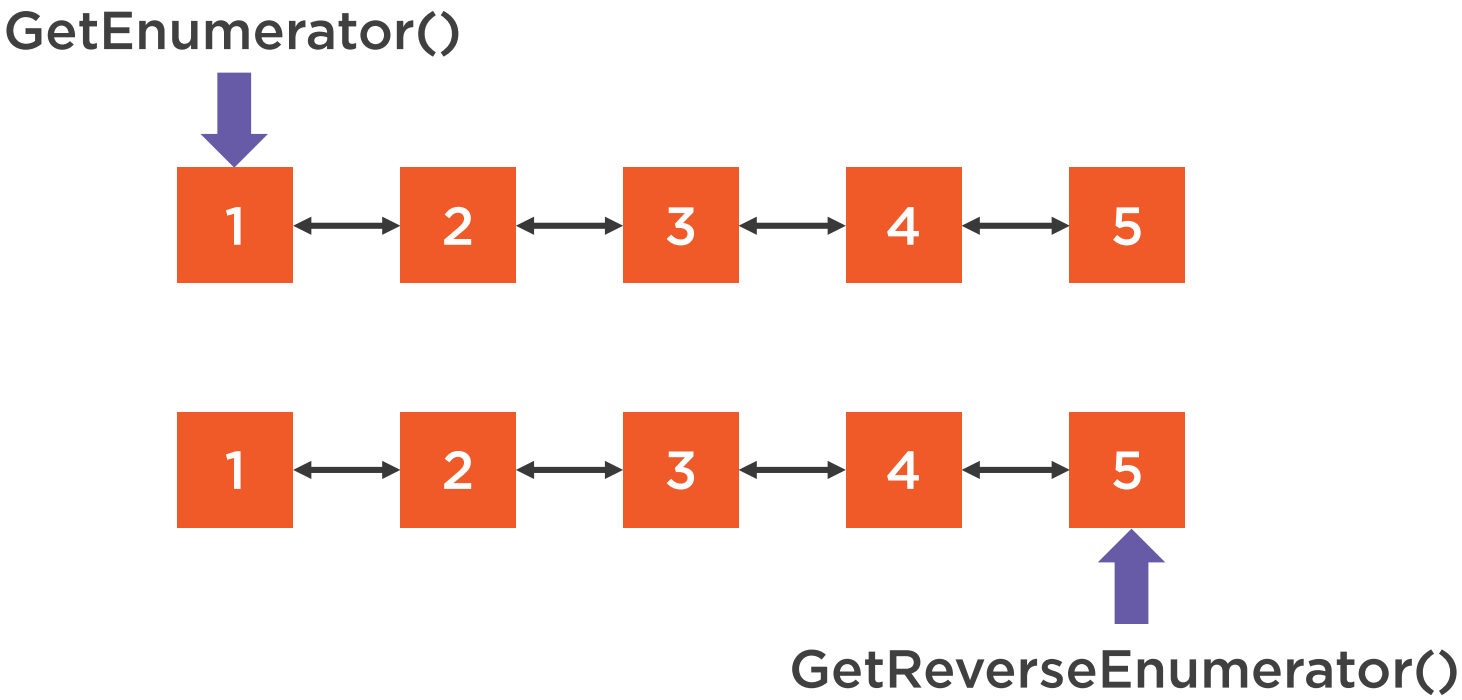
- Find the node to delete
- Remove the found node

Three Cases

- Empty list
- Single node
- Multiple nodes

Enumeration

Enumeration



```
LinkedList<int> ints = LinkedList<int>();

ints.AddTail(1);

ints.AddTail(2);

ints.AddTail(3);


foreach(int value in ints)

{

   Console.WriteLine(value);

}
```

- Create a list of integers
- Build the list with the values 1->2->3
- Enumerate from the head to the tail
- Print out the list values (1, 2, 3)

```
LinkedList<int> ints = LinkedList<int>();

ints.AddTail(1);

ints.AddTail(2);

ints.AddTail(3);


foreach(int value in

      ints.GetReverseEnumerator())

{

   Console.WriteLine(value);

}
```

- Create a list of integers
- Build the list with the values 1->2->3
- Enumerate from the tail to the head
- Print out the list values (3, 2, 1)

Sorted List

A doubly linked list where the values are inserted and stored in sort-order.

Adding Sorted Items

Function	Behavior	Complexity
Add	Adds the specified item to the linked list in the sort order of the item type.	O(n)

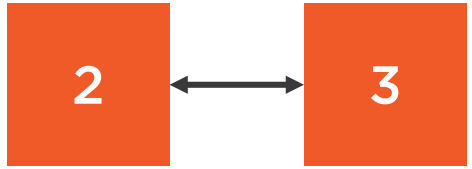
```
SortedList<int> sorted = new SortedList<int>();

sorted.Add(3);

sorted.Add(2);

sorted.Add(5);

sorted.Add(4);

sorted.Add(1);
```

Inserting Values into a Sorted List

3

```
SortedList<int> sorted = new SortedList<int>();

sorted.Add(3);

sorted.Add(2);

sorted.Add(5);

sorted.Add(4);

sorted.Add(1);
```

Inserting Values into a Sorted List

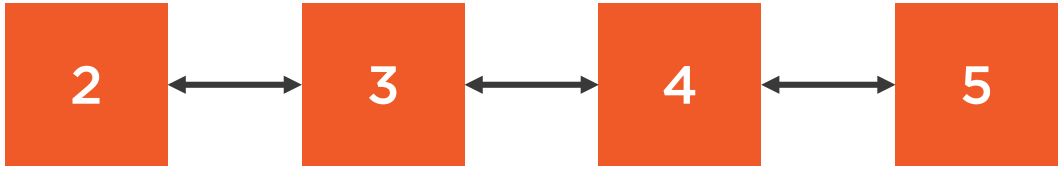


```
SortedList<int> sorted = new SortedList<int>();

sorted.Add(3);

sorted.Add(2);

sorted.Add(5);

sorted.Add(4);

sorted.Add(1);
```

Inserting Values into a Sorted List

```
SortedList<int> sorted = new SortedList<int>();

sorted.Add(3);

sorted.Add(2);

sorted.Add(5);

sorted.Add(4);

sorted.Add(1);
```

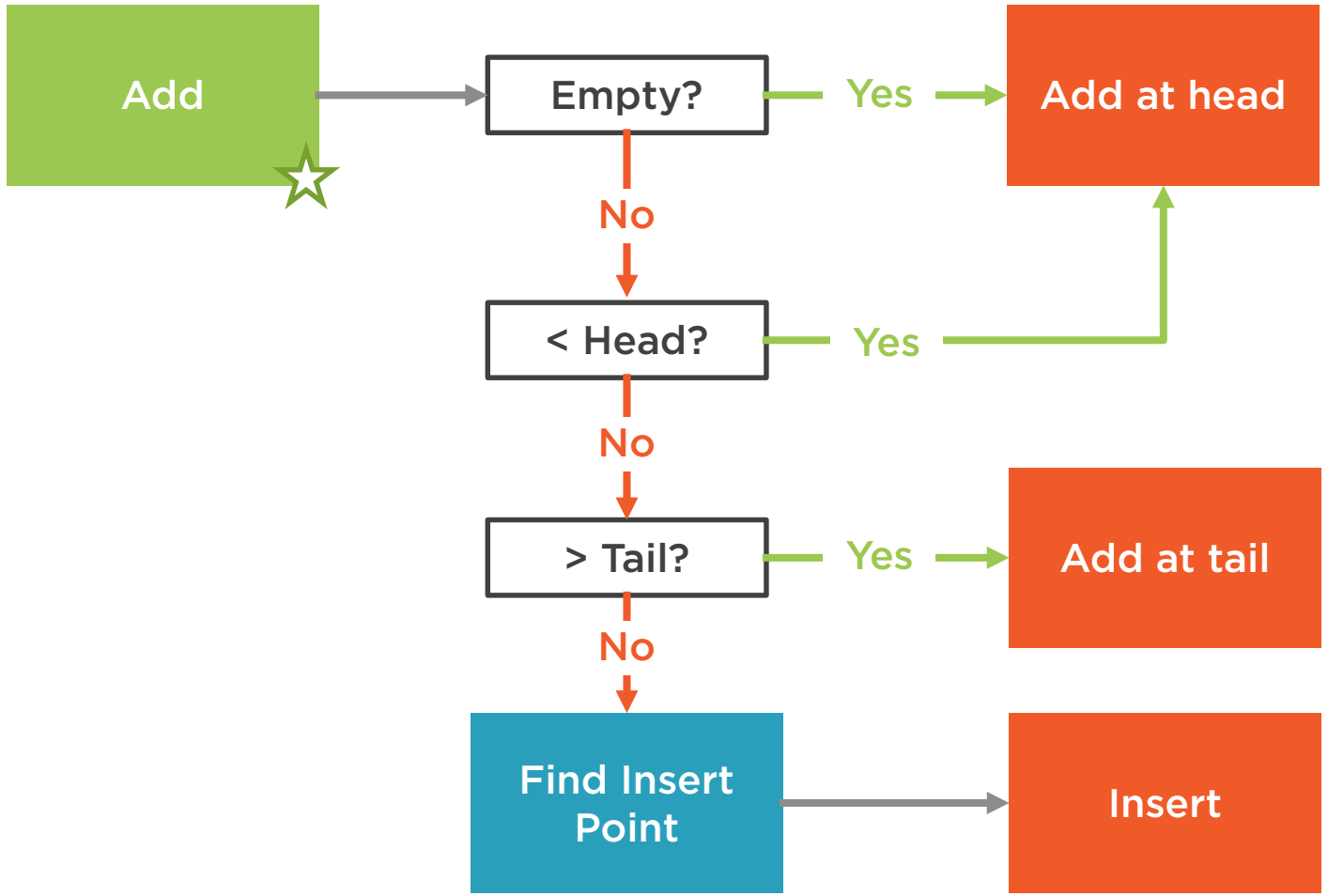
Inserting Values into a Sorted List



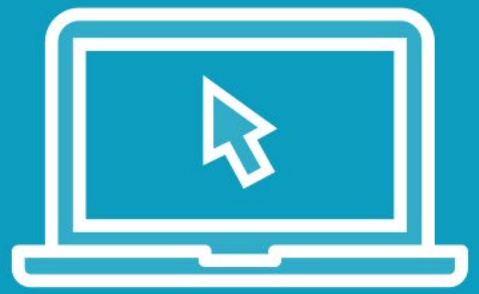
```
SortedList<int> sorted = new SortedList<int>();

sorted.Add(3);

sorted.Add(2);

sorted.Add(5);

sorted.Add(4);

sorted.Add(1);
```

Inserting Values into a Sorted List

Adding Sorted Items Algorithm



Demo



Replace the Contact Manager storage array with a sorted list

- Removes size limit
- Contacts are implicitly sorted
- Code is simplified by not managing the array storage

Lists are nodes chains together

Operations

- Adding and removing
- Enumeration
- Searching

Demo: Contact Manager

- Removed size limitation
- Simplified code