جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قسم الانظمة الطبية الذكية

Intelligent Medical Systems Department

Lap: (7)

Doubly Linked List (DLL)

Subject: Data Strectuer

Class: second

Lecturer: Prof.Dr. Mehdi Ebady Manaa

Programmer:- Fatima Hussein Jawad

Study Year: 2024-2025

Doubly Linked List (DLL)

A **Doubly Linked List** is a type of data structure consisting of nodes. Each node has three main components:

1. **Pointer to the Previous Node (Prev):** Stores the address of the node before the current node.
2. **Data (Data):** Stores the value/data in the node.
3. **Pointer to the Next Node (Next):** Stores the address of the node following the current node.

Difference Between Doubly Linked List and Singly Linked List:

- In a **Singly Linked List**, each node contains only a pointer to the next node.
- In a **Doubly Linked List**, each node has pointers to both the previous and the next nodes, allowing traversal in both directions.

Advantages of a Doubly Linked List:

1. **Bidirectional Traversal:** You can traverse the list forward and backward.
2. **Efficient Insertion/Deletion:** Nodes can be added or removed at any position without needing access to the previous node.
3. **Flexibility:** Reversing the list is straightforward.

Disadvantages:

1. Requires more memory than a **Singly Linked List** due to the extra pointer (Prev).
2. Operations can be slightly slower due to managing two pointers.

Structure of a Node:

In Python, a node in a **Doubly Linked List** can be represented as follows:

```
class Node:

    def __init__(self, data):

        self.data = data  # The value stored in the node

        self.prev = None  # Pointer to the previous node
```

```
        self.next = None  # Pointer to the next node
```

Examples of Basic Operations:

1. Insert Nodes at the Beginning:

```
dll = DoublyLinkedList()

dll.insert_at_beginning(10)

dll.insert_at_beginning(20)

dll.insert_at_beginning(30)

dll.display_forward()
```

 # Output: 30 <-> 20 <-> 10 <-> None

2. Insert Nodes at the End:

```
dll.insert_at_end(40)
dll.insert_at_end(50)
dll.display_forward()
```

```
# Output: 30 <-> 20 <-> 10 <-> 40 <-> 50 <-> None
```

3. Delete a Node by Value:

```
dll.delete_node(20)
dll.display_forward()
```

```
# Output: 30 <-> 10 <-> 40 <-> 50 <-> None
```

4. Display the List in Reverse:

```
dll.display_backward()
```

```
# Output: 50 <-> 40 <-> 10 <-> 30 <-> None
```

Doubly Linked List Class:

```
# تعريف فئة العقدة (Node)

class Node:

  def __init__(self, data):

    self.data = data       # البيانات المخزنة في العقدة

    self.next = None       # المؤشر إلى العقدة التالية

    self.prev = None       # المؤشر إلى العقدة السابقة

# تعريف فئة القائمة المرتبطة المزدوجة (DoublyLinkedList)

class DoublyLinkedList:

  def __init__(self):

    self.head = None       #( بداية القائمة (أول عقده

  # إدراج عقدة في بداية القائمة

  def insert_at_beginning(self, data):

    new_node = Node(data)       # إنشاء عقدة جديدة

    if self.head is None:       # إذا كانت القائمة فارغة

      self.head = new_node      # تعيين العقدة الجديدة كالرأس

    else:

      new_node.next = self.head          # تعيين المؤشر التالي للعقدة الجديدة

      self.head.prev = new_node          # تعيين المؤشر السابق للعقدة الحالية ليكون العقدة الجديدة

      self.head = new_node  # تعيين العقدة الجديدة كالرأس

  # إدراج عقدة في نهاية القائمة

  def insert_at_end(self, data):

    new_node = Node(data)  # إنشاء عقدة جديدة
```

```
        if self.head is None:  # إذا كانت القائمة فارغة
            self.head = new_node  # تعيين العقدة الجديدة كالرأس
        else:
            current = self.head
            while current.next:  # البحث عن العقدة الأخيرة
                current = current.next
            current.next = new_node  # تعيين المؤشر التالي للعقدة الأخيرة
        new_node.prev = current  # تعيين المؤشر السابق للعقدة الجديدة ليكون العقدة الأخيرة

    # حذف عقدة معينة حسب القيمة
    def delete_node(self, data):
        current = self.head
        while current:  # البحث عن العقدة التي تحتوي على البيانات المطلوبة
            if current.data == data:
                if current.prev:  # إذا لم تكن العقدة هي الرأس
                    current.prev.next = current.next
                else:  # إذا كانت العقدة هي الرأس
                    self.head = current.next
                if current.next:  # إذا لم تكن العقدة هي الذيل
                    current.next.prev = current.prev
                return
            current = current.next
        print("Node not found")

    # طباعة القائمة من البداية
```

```
    def display_forward(self):
        current = self.head
        while current:
            print(current.data, end=" <-> ")
            current = current.next
        print("None")
    # طباعة القائمة من النهاية
    def display_backward(self):
        current = self.head
        if not current:
            print("List is empty")
            return
        while current.next:                    # الانتقال إلى العقدة الأخيرة
            current = current.next
        while current:  # الانتقال بالعكس
            print(current.data, end=" <-> ")
            current = current.prev
        print("None")
# تجربة الكود
dll = DoublyLinkedList()
dll.insert_at_beginning(10)
dll.insert_at_beginning(20)
dll.insert_at_beginning(30)
```

```
dll.display_forward()        # Output: 30 <-> 20 <-> 10 <-> None

dll.insert_at_end(40)

dll.insert_at_end(50)

dll.display_forward()        # Output: 30 <-> 20 <-> 10 <-> 40 <-> 50 <-> None

dll.delete_node(20)

dll.display_forward()        # Output: 30 <-> 10 <-> 40 <-> 50 <-> None

dll.display_backward()       # Output: 50 <-> 40 <-> 10 <-> 30 <-> None
```

Explanation:

1. **`insert_at_beginning`:** Adds a new node at the start of the list.
2. **`insert_at_end`:** Adds a new node at the end of the list.
3. **`delete_node`:** Removes a node with a specific value from the list.
4. **`display_forward`:** Prints the list from the head to the tail.
5. **`display_backward`:** Prints the list from the tail to the head.