Doubly Linked List in Python

Last Updated : 03 Jun, 2024

Doubly Linked List is a type of linked list in which each node contains a data element and two links pointing to the next and previous node in the sequence. This allows for more efficient operations such as traversals, insertions, and deletions because it can be done in both directions.

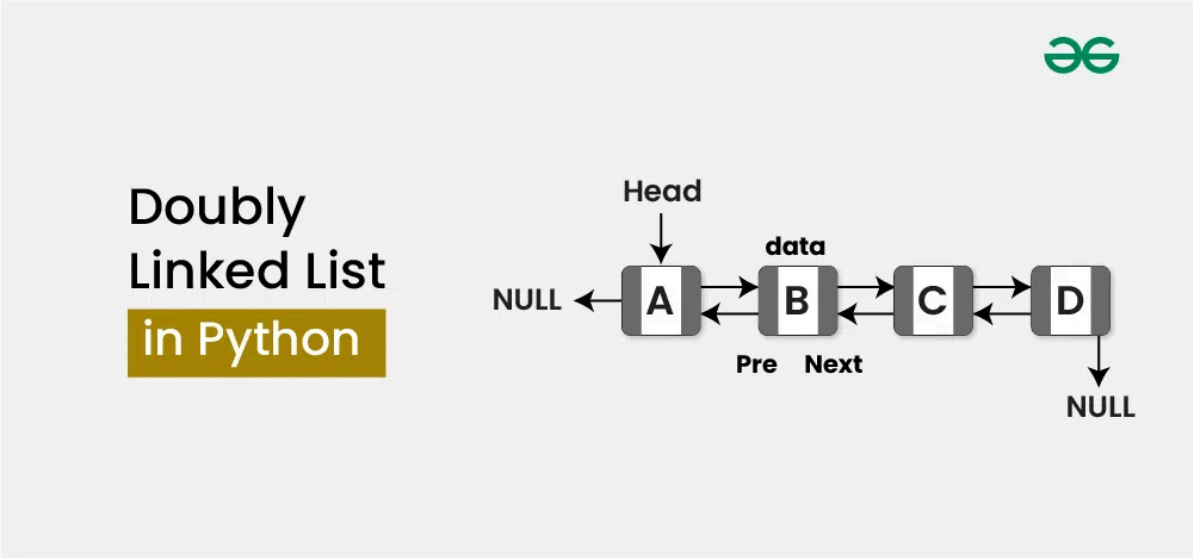

Table of Content

What is a Doubly Linked List?

Doubly Linked List (DLL) is a special type of linked list in which each node contains a pointer to the previous node as well as the next node of the linked

Representation of Doubly Linked List in Python:

Here is the representation of the doubly linked list in python:

Python

```python
# Node of a doubly linked list
class Node:
    def __init__(self, next=None, prev=None, data=None):
        # reference to next node in DLL
        self.next = next
        # reference to previous node in DLL
        self.prev = prev
        self.data = data
```

Traversal of Doubly Linked List in Python:

To traverse a doubly linked list in Python, you can simply start from the head of the list and iterate through each node, printing its data.

Below is the implementation of the above idea:

Python

```python
# Python Program for traversal of a doubly linked list
class Node:
    def __init__(self, data):
        # Initialize a new node with data, previous, and next pointers
        self.data = data
        self.next = None
        self.prev = None


def traverse(head):
    # Traverse the doubly linked list and print its elements
    current = head
    while current:
      # Print current node's data
        print(current.data, end=" <-> ")
```

```
        print("None")


def insert_at_beginning(head, data):
    # Insert a new node at the beginning of the doubly linked
    new_node = Node(data)
    new_node.next = head
    if head:
        head.prev = new_node
    return new_node


# Driver Code
head = None
head = insert_at_beginning(head, 4)
head = insert_at_beginning(head, 3)
head = insert_at_beginning(head, 2)
head = insert_at_beginning(head, 1)

# To traverse and print the nodes:
traverse(head)
```

Output

```
1 <-> 2 <-> 3 <-> 4 <-> None
```

Insertion of Doubly Linked List in Python:

Inserting a new node in a doubly linked list is very similar to inserting new node in linked list. There is a little extra work required to maintain the link of the previous node. A node can be inserted in a Doubly Linked List in five ways:

- At the front of the DLL.
- After a given node.
- Before a given node.
- At the end of the DLL.

1. Insertion at the Beginning:

To insert a node at the beginning of a doubly linked list in Python, you need to follow these steps:

- Create a new node with the given data.
- Set the "**next**" pointer of the new node to point to the current head (if any).
- Set the "**previous**" pointer of the new node to None (as it will become the new head).
- If the list is not empty, update the "**previous**" pointer of the current head to point to the new node.
- Update the head of the list to point to the new node.

Below is the implementation of the above idea:

Python

```python
# Python Program for a doubly linked list at the beginning of node
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None

# Function to insert a node at the beginning of a doubly linke list
def insert_at_beginning(head, data):
    new_node = Node(data)
    new_node.next = head
    if head:
        head.prev = new_node
    return new_node

# Function to display the elements of the doubly linked list
def display(head):
    current = head
    while current:
        print(current.data, end=" <-> ")
        current = current.next
    print("None")
```

```
# Driver Code
head = None
head = insert_at_beginning(head, 3)
head = insert_at_beginning(head, 2)
head = insert_at_beginning(head, 1)

print("Doubly Linked List after insertion at the beginning:")
display(head)
```

Output

```
Doubly Linked List after insertion at the beginning:
1 <-> 2 <-> 3 <-> None
```

2. Insertion after a given node:

To insert a node after a given node in a doubly linked list in Python, you can follow these steps:

- Create a new node with the given data.
- Set the "**next**" pointer of the new node to point to the next node of the given node.
- Set the "**previous**" pointer of the new node to point to the given node.
- If the next node of the given node is not None, update the "**previous**" pointer of that node to point to the new node.
- Update the "**next**" pointer of the given node to point to the new node.

Below is the implementation of the above idea:

Python

```
# Python Program for Insertion after a given node
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
```

```python
# Function to insert a node after a given node in a doubly lin
list
def insert_after_node(node, data):
    if node is None:
        print("Error: The given node is None")
        return

    new_node = Node(data)
    new_node.prev = node
    new_node.next = node.next

    if node.next:
        node.next.prev = new_node

    node.next = new_node

# Function to display the elements of the doubly linked list
def display(head):
    current = head
    while current:
        print(current.data, end=" <-> ")
        current = current.next
    print("None")


# Driver Code
head = Node(1)
node2 = Node(2)
node3 = Node(3)

head.next = node2
node2.prev = head
node2.next = node3
node3.prev = node2

print("Doubly Linked List before insertion:")
display(head)

insert_after_node(node2, 4)

print("Doubly Linked List after insertion:")
```

Output

```
Doubly Linked List before insertion:
1 <-> 2 <-> 3 <-> None
Doubly Linked List after insertion:
1 <-> 2 <-> 4 <-> 3 <-> None
```

3. Insertion before a given node:

To insert a node before a given node in a doubly linked list in Python, you can follow these steps:

- Create a new node with the given data.
- Set the "**next**" pointer of the new node to point to the given node.
- Set the "**previous**" pointer of the new node to point to the previous node of the given node.
- If the previous node of the given node is not None, update the "**next**" pointer of that node to point to the new node.
- Update the "**previous**" pointer of the given node to point to the new node.

Below is the implementation of the above idea:

Python

```
# Python Program for Insertion before a given node
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None

# Function to insert a node before a given node in a doubly li
list
def insert_before_node(node, data):
    if node is None:
        print("Error: The given node is None")
```

```
    new_node.prev = node.prev
    new_node.next = node

    if node.prev:
        node.prev.next = new_node

    node.prev = new_node

# Function to display the elements of the doubly linked list
def display(head):
    current = head
    while current:
        print(current.data, end=" <-> ")
        current = current.next
    print("None")

# Driver Code
head = Node(1)
node2 = Node(2)
node3 = Node(3)

head.next = node2
node2.prev = head
node2.next = node3
node3.prev = node2

print("Doubly Linked List before insertion:")
display(head)

insert_before_node(node2, 4)

print("Doubly Linked List after insertion:")
display(head)
```

Output

```
Doubly Linked List before insertion:
1 <-> 2 <-> 3 <-> None
Doubly Linked List after insertion:
```

4. Insertion at the end:

To insert a node at the end of a doubly linked list in Python, you need to follow these steps:

- Create a new node with the given data.
- If the list is empty (head is None), make the new node the head of the list.
- Otherwise, traverse the list to find the last node.
- Set the "**next**" pointer of the last node to point to the new node.
- Set the "**previous**" pointer of the new node to point to the last node.
- Optionally, update the head of the list to point to the new node if it's the first node in the list.

Below is the implementation of the above idea:

Python

```python
# Python Program for Insertion at the end
class Node:
    def __init__(self, data):
        # Initialize a new node with data, previous, and next pointers
        self.data = data
        self.next = None
        self.prev = None


def insert_at_end(head, data):
    # Insert a new node at the end of the doubly linked list
    new_node = Node(data)
    if head is None:
        return new_node

    current = head
    while current.next:
        current = current.next

    current.next = new_node
```

```
24
25 def display(head):
26     # Display the doubly linked list elements
27     current = head
28     while current:
29       # Print current node's data
30         print(current.data, end=" <-> ")
31         # Move to the next node
32         current = current.next
33     print("None")
34
35
36 # Driver Code
37 head = None
38 head = insert_at_end(head, 1)
39 head = insert_at_end(head, 2)
40 head = insert_at_end(head, 3)
41
42 print("Doubly Linked List after insertion at the end:")
43 display(head)
```

Output

```
Doubly Linked List after insertion at the end:
1 <-> 2 <-> 3 <-> None
```

Deletion of Doubly Linked List in Python:

Deletion of a node in Doubly Linked List generally involves modifying the next and the previous pointers of nodes. Deletion can be done in 3 ways:

- At the beginning of DLL
- At the end of DLL
- At a given position in DLL

1. Deletion at the beginning:

To delete a node from the beginning of a doubly linked list in Python, you need

- Check if the list is empty (**head is None**). If it is empty, there is nothing to delete.
- If the list has only one node, set the head to None to delete the node.
- Otherwise, update the head to point to the next node.
- Set the "**previous**" pointer of the new head to None.
- Optionally, free the memory allocated to the deleted node.

Below is the implementation of the above idea:

Python

```python
# Python Program for the deletion at the beginning
class Node:
    def __init__(self, data):
        # Initialize a new node with data, previous, and next
        self.data = data
        self.next = None
        self.prev = None

def delete_at_beginning(head):
    # Delete the first node from the beginning of the doubly l
list
    if head is None:
        print("Doubly linked list is empty")
        return None

    if head.next is None:
        return None

    new_head = head.next
    new_head.prev = None
    del head
    return new_head

def traverse(head):
    # Traverse the doubly linked list and print its elements
    current = head
    while current:
        # Print current node's data
```

```
        current = current.next
    print("None")

def insert_at_beginning(head, data):
    # Insert a new node at the beginning of the doubly linked
    new_node = Node(data)
    new_node.next = head
    if head:
        head.prev = new_node
    return new_node

# Driver Code
head = None
head = insert_at_beginning(head, 4)
head = insert_at_beginning(head, 3)
head = insert_at_beginning(head, 2)
head = insert_at_beginning(head, 1)

# Delete the first node (node with data 1) from the beginning:
head = delete_at_beginning(head)

# Traverse and print the nodes after deletion:
traverse(head)
```

Output

```
2 <-> 3 <-> 4 <-> None
```

2. Deletion at a given position:

To delete a node at a given position in a doubly linked list in Python, you need to follow these steps:

- Check if the list is empty (head is None). If it is empty, there is nothing to delete.
- If the position is less than **0**, print an error message as it's an invalid position.
- If the position is **0**, delete the node at the beginning of the list.

- Update the "**next**" pointer of the previous node to skip the node to be deleted.
- Update the "**previous**" pointer of the next node to point to the previous node of the node to be deleted.
- Optionally, free the memory allocated to the deleted node.

Below is the implementation of the above idea:

Python

```python
# Python Program for Deletion of a given node
class Node:
    def __init__(self, data):
        # Initialize a new node with data, previous, and next pointers
        self.data = data
        self.next = None
        self.prev = None


def delete_at_position(head, position):
    # Delete the node at a given position from the doubly link
    if head is None:
        print("Doubly linked list is empty")
        return None

    if position < 0:
        print("Invalid position")
        return head

    if position == 0:
        if head.next:
            head.next.prev = None
        return head.next

    current = head
    count = 0
    while current and count < position:
        current = current.next
```

```
    if current is None:
        print("Position out of range")
        return head

    if current.next:
        current.next.prev = current.prev
    if current.prev:
        current.prev.next = current.next

    del current
    return head


def traverse(head):
    # Traverse the doubly linked list and print its elements
    current = head
    while current:
      # Print current node's data
        print(current.data, end=" <-> ")
        # Move to the next node
        current = current.next
    print("None")


def insert_at_beginning(head, data):
    # Insert a new node at the beginning of the doubly linked
    new_node = Node(data)
    new_node.next = head
    if head:
        head.prev = new_node
    return new_node


# Driver Code
head = None
head = insert_at_beginning(head, 4)
head = insert_at_beginning(head, 3)
head = insert_at_beginning(head, 2)
head = insert_at_beginning(head, 1)

# Delete the node at position 2 (node with data 3):
```

```
# Traverse and print the nodes after deletion:
traverse(head)
```

Output

```
1 <-> 2 <-> 4 <-> None
```

3. Deletion at the end:

To delete a node at the end of a doubly linked list in Python, you need to follow these steps:

- Check if the list is empty (**head is None**). If it is empty, there is nothing to delete.
- If the list has only one node, set the head to None to delete the node.
- Traverse the list to find the last node.
- Set the "**next**" pointer of the second-to-last node to None.
- Optionally, free the memory allocated to the deleted node.

Below is the implementation of the above idea:

Python

```python
# Python Program Deletion at the end
class Node:
    def __init__(self, data):
        # Initialize a new node with data, previous, and next pointers
        self.data = data
        self.next = None
        self.prev = None

def delete_at_end(head):
    # Delete the last node from the end of the doubly linked l
    if head is None:
        print("Doubly linked list is empty")
```

```
        return None

    current = head
    while current.next.next:
        current = current.next

    del_node = current.next
    current.next = None
    del del_node
    return head

def traverse(head):
    # Traverse the doubly linked list and print its elements
    current = head
    while current:
      # Print current node's data
        print(current.data, end=" <-> ")
        # Move to the next node
        current = current.next
    print("None")

def insert_at_beginning(head, data):
    # Insert a new node at the beginning of the doubly linked
    new_node = Node(data)
    new_node.next = head
    if head:
        head.prev = new_node
    return new_node

# Driver Code
head = None
head = insert_at_beginning(head, 4)
head = insert_at_beginning(head, 3)
head = insert_at_beginning(head, 2)
head = insert_at_beginning(head, 1)

# Delete the last node (node with data 4) from the end:
head = delete_at_end(head)

# Traverse and print the nodes after deletion:
traverse(head)
```

Output

```
1 <-> 2 <-> 3 <-> None
```

Related Practice Problem on Doubly Linked List:

Article	Link
Design a data structure for LRU Cache	Link
Reverse a Doubly Linked List	Link
Delete a node in a Doubly Linked List	Link
Insertion in a Doubly Linked List	Link
Insert value in sorted way in a sorted doubly linked list	Link
Find pairs with given sum in doubly linked list	Link
Merge Sort for Doubly Linked List	Link
QuickSort on Doubly Linked List	Link
XOR Linked List – A Memory Efficient Doubly Linked List | Set 2	Link

Comment More info

Next Article

Circular Linked List in Python

Similar Reads

Doubly Linked List in Python

Doubly Linked List is a type of linked list in which each node contains a data element and two links pointing to the next and previous node in the sequence. This allows for more efficient operations such as traversals,...

13 min read

Circular Linked List in Python

A Circular Linked List is a variation of the standard linked list. In a standard linked list, the last element points to null, indicating the end of the list. However, in a circular linked list, the last element points back to the first...

13 min read

Doubly Linked List in C

A doubly linked list is a type of linked list in which each node contains 3 parts, a data part and two addresses, one points to the previous node and one for the next node. It differs from the singly linked list as it has an...

14 min read

Doubly Linked List in C++

A Doubly Linked List (DLL) is a two-way list in which each node has two pointers, the next and previous that have reference to both the next node and previous node respectively. Unlike a singly linked list where each...

13 min read

How to copy linked list in Python?

Linked Lists: Linked Lists are a type of 'Abstract Data Types' in Python. These data structures are linear in nature. Linked Lists are different from Lists as Linked Lists are stored in non-contiguous (non-continuous)...

13 min read

Python Linked List

In this article, we will learn about the implementation of a linked list in Python. To implement the linked list in Python, we will use classes in Python. Now, we know that a linked list consists of nodes and nodes have tw...

13 min read

Traversal in Doubly Linked List

15+ min read

Python | Stack using Doubly Linked List

A stack is a collection of objects that are inserted and removed using Last in First out Principle(LIFO). User can insert elements into the stack, and can only access or remove the recently inserted object on top of the stack....

4 min read

Python | Queue using Doubly Linked List

A Queue is a collection of objects that are inserted and removed using First in First out Principle(FIFO). Insertion is done at the back(Rear) of the Queue and elements are accessed and deleted from first(Front)...

3 min read

Python Library for Linked List

Linked list is a simple data structure in programming, which obviously is used to store data and retrieve it accordingly. To make it easier to imagine, it is more like a dynamic array in which data elements are linked via...

5 min read

Doubly Linked List Tutorial

A doubly linked list is a more complex data structure than a singly linked list, but it offers several advantages. The main advantage of a doubly linked list is that it allows for efficient traversal of the list in both directions....

15+ min read

Singly Linked List in Python

A Singly Linked List is a type of data structure that is made up of nodes that are created using self-referential structures. Each node contains a data element and a reference (link) to the next node in the sequence. This...

10 min read

Doubly Linked List meaning in DSA

A doubly linked list is a special type of linked list in which each node contains a pointer to the previous node as well as the next node in the structure. Characteristics of the Doubly Linked List: The characteristics of a...

3 min read

Memory efficient doubly linked list

We need to implement a doubly linked list with the use of a single pointer in each node. For that we are given a stream of data of size n for the linked list, your task is to make the function insert() and getList(). The insert...

9 min read

Why use a Doubly Linked List?

Program to find size of Doubly Linked List

Given a doubly linked list, The task is to find the number of nodes in the given doubly linked list. Example: Input : 1<->2<->3<->4output : 4 Input : 1<->2output : 2 By Traversing the Doubly linked list – O(n) Time and...

5 min read

Print Doubly Linked list in Reverse Order

Given a doubly-linked list of positive integers. The task is to print the given doubly linked list data in reverse order. Examples: Input: List = 1 <=> 2 <=> 3 <=> 4 <=> 5 Output: 5 4 3 2 1 Input: 10 <=> 20 <=> 30 <=> 40...

7 min read

Deletion in a Doubly Linked List

Deleting a node in a doubly linked list is very similar to deleting a node in a singly linked list. However, there is a little extra work required to maintain the links of both the previous and next nodes. In this article, we will...

15+ min read

Insertion in a Doubly Linked List

Inserting a new node in a doubly linked list is very similar to inserting new node in linked list. There is a little extra work required to maintain the link of the previous node. In this article, we will learn about different way...

15+ min read

Pretty print Linked List in Python

Creating custom data types can be tricky, especially when you want to use it like any other data type. Linked List can be thought of as an example of a custom data type. In other languages, if you want to print the linke...

3 min read

Article Tags : DSA Linked List doubly linked list Python-DSA

Practice Tags : Linked List

Sector 137, Noida, Gautam Buddh
Nagar, Uttar Pradesh, 201305