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. A node can be inserted in a Doubly Linked List in four ways:

- At the front of the DLL.
- In between two nodes
  - After a given node.
  - Before a given node.
- At the end of the DLL.

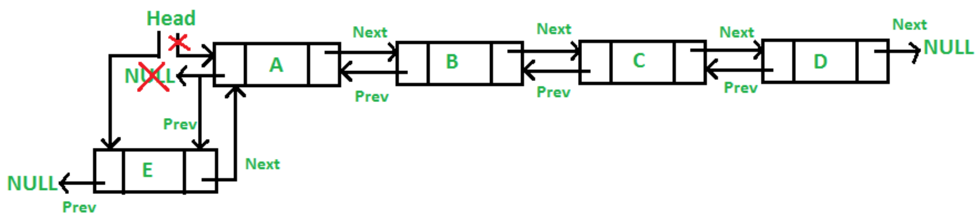
Add a node at the front in a Doubly Linked List:

The new node is always added before the head of the given Linked List. The task can be performed by using the following 5 steps:

1. Firstly, allocate a new node (say **new_node**).
2. Now put the required data in the new node.
3. Make the next of **new_node** point to the current head of the doubly linked list.
4. Make the previous of the current head point to **new_node**.
5. Lastly, point head to **new_node**.

Illustration:

See the below illustration where **E** is being inserted at the beginning of the doubly linked list.



```
public void push(int new_data)
{
    // 1. allocate node
    // 2. put in the data */
    Node new_Node = new Node(new_data);

    // 3. Make next of new node as head and previous as NULL
    new_Node.next = head;
    new_Node.prev = null;

    // 4. change prev of head node to new node
    if (head != null)
        head.prev = new_Node;

    // 5. move the head to point to the new node
    head = new_Node;
}
```

Time Complexity: O(1)
Auxiliary Space: O(1)

Add a node in between two nodes:

It is further classified into the following two parts:

Add a node after a given node in a Doubly Linked List:

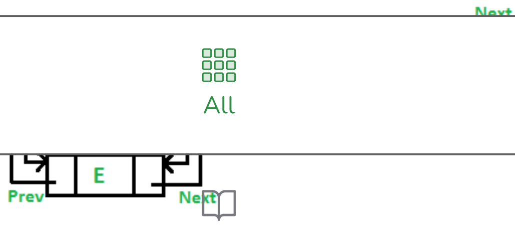
We are given a pointer to a node as **prev_node**, and the new node is inserted after the given node. This can be done using the following 6 steps:

1. Firstly create a new node (say **new_node**).
2. Now insert the data in the new node.
3. Point the next of **new_node** to the next of **prev_node**.
4. Point the next of **prev_node** to **new_node**.
5. Point the previous of **new_node** to **prev_node**.
6. Change the pointer of the new node's previous pointer to **new_node**.

Illustration:

Dash

All



Articles

Videos

Below is the implementation of the 7 steps to insert a node after a given node in the linked list:

C++ Java

Problems

Quiz

Contest

```
 // Given a node as prev_node, insert a new node
// after the given node
public void InsertAfter(Node prev_Node, int new_data)
{
    // Check if the given prev_node is NULL
    if (prev_Node == null) {
        System.out.println(
            "The given previous node cannot be NULL ");
        return;
    }

    // 1. allocate node
    // 2. put in the data
    Node new_node = new Node(new_data);

    // 3. Make next of new node as next of prev_node
    new_node.next = prev_Node.next;

    // 4. Make the next of prev_node as new_node
    prev_Node.next = new_node;

    // 5. Make prev_node as previous of new_node
    new_node.prev = prev_Node;

    // 6. Change previous of new_node's next node
    if (new_node.next != null)
        new_node.next.prev = new_node;
```

Menu

Track Progress

32 of 132 Complete. (25%)

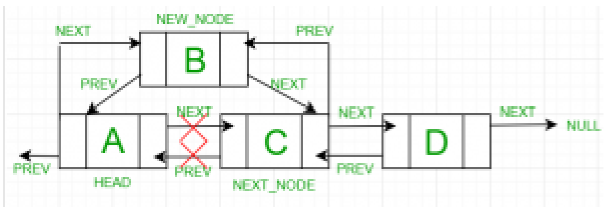
Time Complexity: O(1)

Let the pointer to this given node be **next_node**. This can be done using the following 6 steps.

1. Allocate memory for the new node, let it be called **new_node**.
2. Put the data in **new_node**.
3. Set the previous pointer of this **new_node** as the previous node of the **next_node**.
4. Set the previous pointer of the **next_node** as the **new_node**.
5. Set the next pointer of this **new_node** as the **next_node**.
6. Now set the previous pointer of **new_node**.
   - If the previous node of the new_node is not NULL, then set the next pointer of this previous node as **new_node**.
   - Else, if the prev of new_node is NULL, it will be the new head node.

Illustration:

See the below illustration where '**B**' is being inserted before '**C**'.



Below is the implementation of the above approach.

C++ Java

```
 // Given a node as prev_node, insert a new node
// after the given node
public void InsertBefore(Node next_Node, int new_data)
{
    // Check if the given next_node is NULL
    if (next_Node == null) {
        System.out.println(
            "The given next node cannot be NULL ");
        return;
    }
    Node new_node = new Node(new_data);
```

```
    // 4. Make the prev of next_node as new_node
    next_Node.prev = new_node;

    // 5. Make next_node as next of new_node
    new_node.next = next_Node;

    // 6. Change next of new_node's previous node
    if (new_node.prev != null)
        new_node.prev.next = new_node;
    else
        head = new_node;
}
```

Time Complexity: O(1)
Auxiliary Space: O(1)

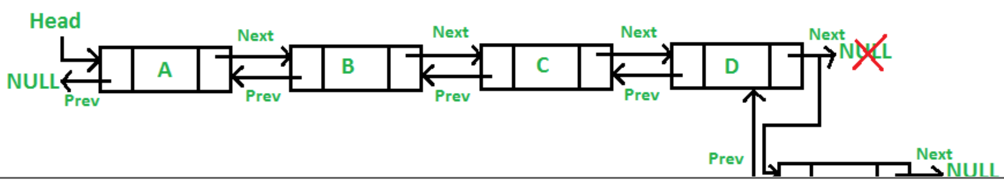
Add a node at the end in a Doubly Linked List:

The new node is always added after the last node of the given Linked List. This can be done using the following 7 steps:

1. Create a new node (say **new_node**).
2. Put the value in the new node.
3. Make the next pointer of **new_node** as null.
4. If the list is empty, make **new_node** as the head.
5. Otherwise, travel to the end of the linked list.
6. Now make the next pointer of last node point to **new_node**.
7. Change the previous pointer of **new_node** to the last node of the list.

Illustration:

See the below illustration where '**D**' is inserted at the end of the linked list.



Below is the implementation of the 7 steps to insert a node at the end of the linked

```
 // Add a node at the end of the list
void append(int new_data)
{
    // 1. allocate node
    // 2. put in the data
    Node new_node = new Node(new_data);

    Node last = head; /* used in step 5*/

    // 3. This new node is going to be the last node, so
    // make next of it as NULL
    new_node.next = null;

    // 4. If the Linked List is empty, then make the new
    // node as head

        head = new_node;
        return;
    }

    // 5. Else traverse till the last node
    while (last.next != null)
        last = last.next;

    // 6. Change the next of last node
    last.next = new_node;

    // 7. Make last node as previous of new node
    new_node.prev = last;
}
```

Time Complexity: O(n)
Auxiliary Space: O(1)

Track Progress
32 of 132 Complete. (25%)

Dash

All

Articles

Videos

Problems

Quiz

Contest

Menu

Track Progress

32 of **132** Complete. (25%)