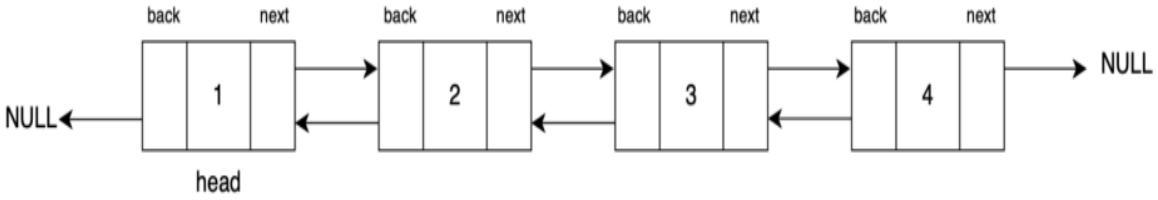
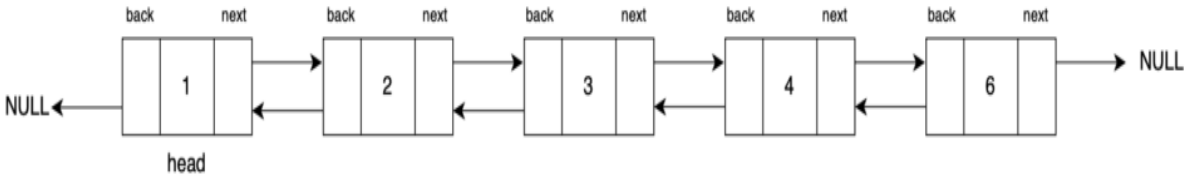
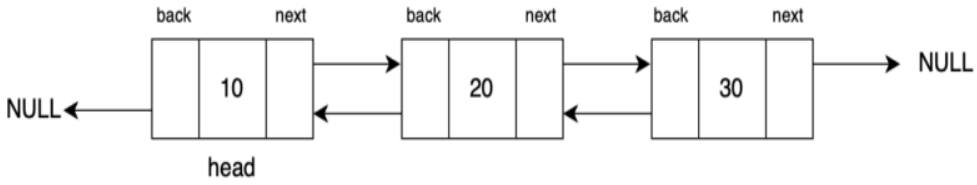
Insert at end of Doubly Linked List

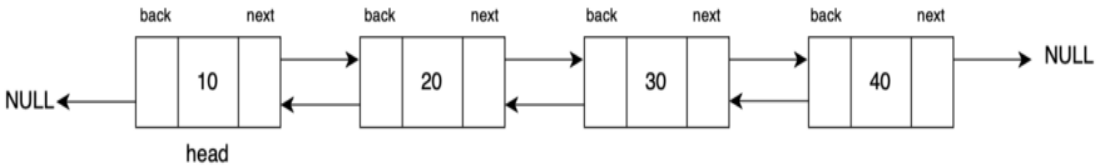
Problem Statement: Given a doubly linked list, and a value **‘k’**, insert a node having value **‘k’** at the end of the doubly linked list.

Examples









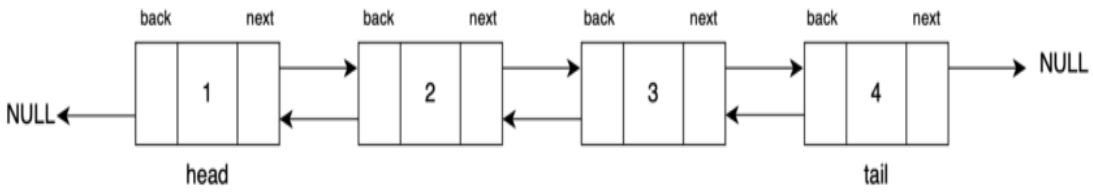
Solution

Approach:

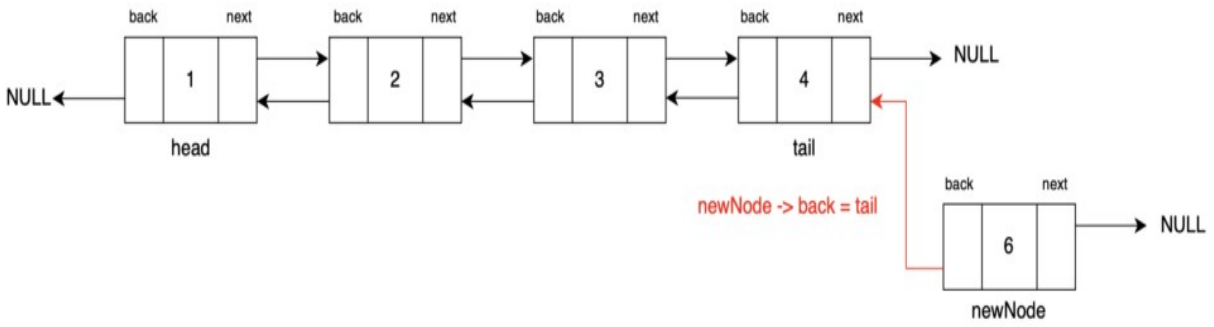
To insert before a given node, begin by identifying its **previous node**. This step is assured since the provided node is never the head. Create a **new node** with the specified value and set its **back** and **next** pointers to

the **previous node** and the **given node**, respectively. To seamlessly integrate the new node into the doubly linked list, set the previous node's **next** pointer and the given node's **back** pointer to the **new node**.

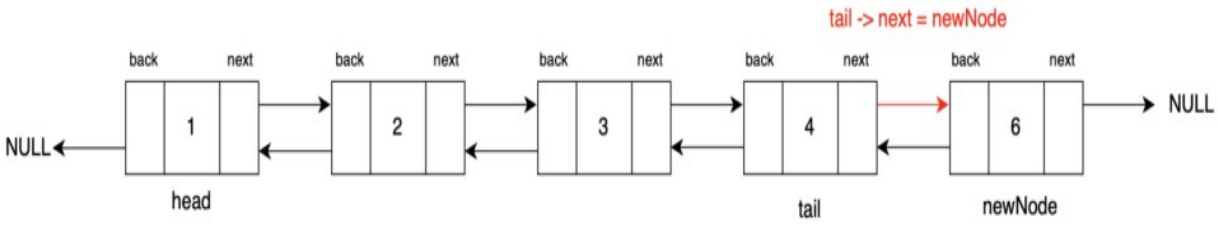
Step 1: Traverse through the list, and reach the tail of the DLL. Let's use a node **tail** traverse from the head.



Step 2: Create a **new node** with its **data as k** and back **pointer pointing to tail** and next pointer pointing to null as the new tail will point to null**.**



Step 3: Update the **next pointer** of the current **tail node** to point to the newly created node which will be our new tail post this. Then, **return the head** of the updated doubly linked list.



Code:

```
#include <iostream>
#include <bits/stdc++.h>

using namespace std;

// Define a Node class for doubly linked list
```

```
class Node {
public:
    // Data stored in the node
    int data;
    // Pointer to the next node in the list (forward direction)
    Node* next;
    // Pointer to the previous node in the list (backward direction)
    Node* back;

    // Constructor for a Node with both data, a reference to the next node,
and a reference to the previous node
    Node(int data1, Node* next1, Node* back1) {
        data = data1;
        next = next1;
        back = back1;
    }

    // Constructor for a Node with data, and no references to the next and
previous nodes (end of the list)
    Node(int data1) {
        data = data1;
        next = nullptr;
        back = nullptr;
    }
};

// Function to convert an array to a doubly linked list
Node* convertArr2DLL(vector<int> arr) {
    // Create the head node with the first element of the array
    Node* head = new Node(arr[0]);
    // Initialize 'prev' to the head node
    Node* prev = head;

    for (int i = 1; i < arr.size(); i++) {
        // Create a new node with data from the array and set its 'back'
pointer to the previous node
        Node* temp = new Node(arr[i], nullptr, prev);
        // Update the 'next' pointer of the previous node to point to the
new node
        prev->next = temp;
         // Move 'prev' to the newly created node for the next iteration
        prev = temp;
    }
    // Return the head of the doubly linked list
    return head;
}

// Function to print the elements of the doubly linked list
void print(Node* head) {
    while (head != nullptr) {
        // Print the data in the tail node
        cout << head->data << " ";
         // Move to the next node
        head = head->next;
    }
}
```

```
// Function to insert a new node with value 'k' at the end of the doubly
linked list
Node* insertAtTail(Node* head, int k) {
    // Create a new node with data 'k'
    Node* newNode = new Node(k);

    // If the doubly linked list is empty, set 'head' to the new node
    if (head == nullptr) {
        return newNode;
    }

    // Traverse to the end of the doubly linked list
    Node* tail = head;
    while (tail->next != nullptr) {
        tail = tail->next;
    }

    // Connect the new node to the last node in the list
    tail->next = newNode;
    newNode->back = tail;

    return head;
}



int main() {
    vector<int> arr = {12, 5, 8, 7, 4};
    Node* head = convertArr2DLL(arr);

cout << "Doubly Linked List Initially: " << endl;
    print(head);


    cout << endl << "Doubly Linked List After Inserting at the tail with
value 10: " << endl;
     // Insert a node with value 10 at the end
    head = insertAtTail(head, 10);
    print(head);

    return 0;
}
```

Output:

Doubly Linked List Initially:
12 5 8 7 4
Doubly Linked List After Inserting at the tail with value 10:
12 5 8 7 4 10

Time Complexity: O(N) The time complexity of this insertion operation is O(N) because we have to **traverse** the entire list to reach its tail. The complexity would be O(1) if we were given the tail node directly.

Space Complexity: O(1) The space complexity is also O(1) because we are **notusing** any **extradatastructures** to do the operations apart from creating a single new node.