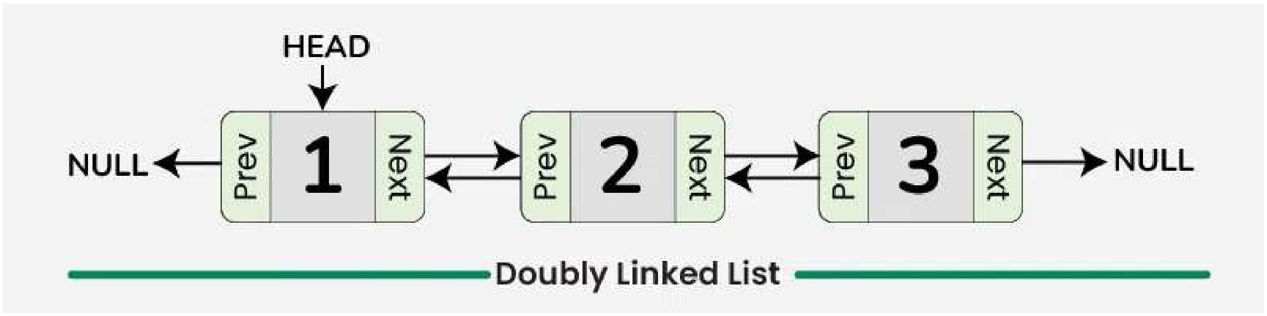
Search...

Doubly Linked List Tutorial

Last Updated : 19 Feb, 2025

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. This is because each node in the list contains a pointer to the previous node and a pointer to the next node. This allows for quick and easy insertion and deletion of nodes from the list, as well as efficient traversal of the list in both directions.

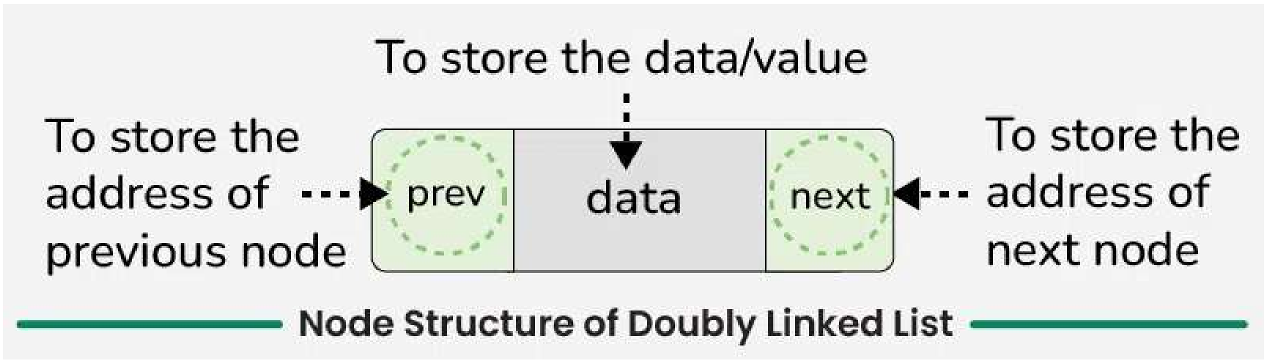


Doubly Linked List

Representation of Doubly Linked List in Data Structure

In a data structure, a doubly linked list is represented using nodes that have three fields:

1. Data
2. A pointer to the next node (**next**)
3. A pointer to the previous node (**prev**)


To store the data/value
To store the address of previous node
prev
data
next
To store the address of next node
Node Structure of Doubly Linked List

Node Structure of Doubly Linked List

Node Definition

Here is how a node in a Doubly Linked List is typically represented:

C++ C Java Python C# JavaScript

```
class Node
{
    // To store the value or data
    public int Data;

    // Pointer to the next node
    public Node Next;

    // Pointer to the previous node
    public Node Prev;

    // Constructor
    public Node(int d)
    {
        Data = d;
        Prev = Next = null;
    }
}
```

Each node in a **Doubly Linked List** contains the **data** it holds, a pointer to the **next** node in the list, and a pointer to the **previous** node in the list. By linking these nodes together through the **next** and **prev** pointers, we can traverse the list in both directions (forward and backward), which is a key feature of a Doubly Linked List.

Table of Content

1. Traversal in Doubly Linked List

Traversal in a **Doubly Linked List** involves visiting each node, processing its data, and moving to the next or previous node using the forward (`next`) and backward (`prev`) pointers.

Step-by-Step Approach for Traversal:

1. **Start from the head** of the list.
2. **Traverse forward**:
   - Visit the current node and process its data (e.g., print it).
   - Move to the next node using `current = current->next`.
   - Repeat the process until the end of the list (`current == NULL`).

3. **Optionally, traverse backward**:
   - Start from the tail (last node).
   - Visit the current node and process its data.
   - Move to the previous node using `current = current->prev`.
   - Repeat the process until the beginning of the list (`current == NULL`).

Traversal is useful for displaying or processing all nodes in a doubly linked list.

To read more about Traversal Operation Refer, Traversal in Doubly Linked List

2. Finding Length of Doubly Linked List

A Doubly Linked List (DLL) is a type of linked list where each node has two pointers:

1. One pointing to the next node in the sequence.
2. One pointing to the previous node in the sequence.

To find the length of a doubly linked list, we need to traverse the list while counting the nodes.

Step-by-Step Approach for finding length:

1. **Initialize a counter**: Start with a counter variable (`count = 0`).
2. **Set a pointer to the head node**: Use a pointer (`current`) and initialize it to the head of the linked list.

3. **Traverse the list**:
   - While the pointer (`current`) is not `NULL`, increment the `count` by `1`.
   - Move to the next node (`current = current.next`).
4. **Stop at the end of the list**: When the pointer reaches `NULL`, stop the loop.
5. **Return the count**: The final value of `count` gives the length of the doubly linked list.

To read more about Finding Length of DLL Refer, Program to find size of Doubly Linked List

3. Insertion in a Doubly Linked List

Insertion in a **Doubly Linked List (DLL)** involves adding a new node at a specific position while maintaining the connections between nodes. Since each node contains a pointer to both the previous and next node, insertion requires adjusting these pointers carefully.

There are three primary types of insertion in a DLL:

1. Insertion at the Beginning

1. Create a new node with the given data.
2. Set the `next` pointer of the new node to the current head.
3. If the list is not empty, update the `prev` pointer of the current head to point to the new node.
4. Update the head of the list to the new node.

Read more about Insertion at the Beginning Refer, Insert a Node at Front/Beginning of Doubly Linked List

2. Insertion at the End

1. Create a new node with the given data.
2. If the list is empty, set the new node as the head.
3. Traverse the list until the last node is found.
4. Set the `next` pointer of the last node to the new node.
5. Set the `prev` pointer of the new node to the last node.

Read more about Insertion at the End Refer, Insert a Node at the end of Doubly Linked List

3. Insertion at a Specific Position

1. Create a new node with the given data.
2. If inserting at the beginning, follow the steps for insertion at the start.
3. Traverse the list to find the node after which insertion is needed.
4. Set the `next` pointer of the new node to the next node of the current position.
5. Set the `prev` pointer of the new node to the current node.
6. Update the `prev` pointer of the next node to point to the new node (if it exists).
7. Update the `next` pointer of the previous node to point to the new node.

Read more about Insertion at a specific position Refer, Insert a Node at a specific position in Doubly Linked List

4. Deletion in a Doubly Linked List

Deletion in a **Doubly Linked List (DLL)** involves removing a node while maintaining the integrity of the list. Since each node contains pointers to both its previous and next nodes, deletion requires careful pointer adjustments to ensure no broken links occur.

Types of Deletion in a Doubly Linked List

1. Deletion at the Beginning

1. Check if the list is empty; if it is, return as there is nothing to delete.
2. Store the current head node in a temporary variable.
3. Move the head pointer to the next node.
4. If the new head exists, update its `prev` pointer to `NULL`.
5. Delete the old head node to free memory.

Read more about Deletion at the Beginning Refer, Deletion at beginning (Removal of first node) in a Doubly Linked List

2. Deletion at the End

1. Check if the list is empty; if it is, return.
2. Traverse the list to find the last node.
3. Store the last node in a temporary variable.
4. Update the `next` pointer of the second-last node to `NULL`, making it the new tail.
5. Delete the last node to free memory.

Read more about Deletion at the End Refer, Deletion at end (Removal of last node) in a Doubly Linked List

3. Deletion at a Specific Position

1. Check if the list is empty; if it is, return.
2. Traverse the list to find the node to be deleted.
3. Store the node to be deleted in a temporary variable.
4. Update the `next` pointer of the previous node to point to the next node.
5. Update the `prev` pointer of the next node to point to the previous node (if it exists).
6. Delete the target node to free memory.

Read more about Deletion at a specific position Refer, Delete a Doubly Linked List node at a given position

Advantages of Doubly Linked List

- **Efficient traversal in both directions:** Doubly linked lists allow for efficient traversal of the list in both directions, making it suitable for applications where frequent insertions and deletions are required.
- **Easy insertion and deletion of nodes:** The presence of pointers to both the previous and next nodes makes it easy to insert or delete nodes from the list, without having to traverse the entire list.
- **Can be used to implement a stack or queue:** Doubly linked lists can be used to implement both stacks and queues, which are common data

Disadvantages of Doubly Linked List

- **More complex than singly linked lists:** Doubly linked lists are more complex than singly linked lists, as they require additional pointers for each node.
- **More memory overhead:** Doubly linked lists require more memory overhead than singly linked lists, as each node stores two pointers instead of one.

Applications of Doubly Linked List

- Implementation of undo and redo functionality in text editors.
- Cache implementation where quick insertion and deletion of elements are required.
- Browser history management to navigate back and forth between visited pages.
- Music player applications to manage playlists and navigate through songs efficiently.
- Implementing data structures like Deque (double-ended queue) for efficient insertion and deletion at both ends.

Practice Questions on Doubly Linked List
MCQs on Linked List

Comment More info Advertise with us

Next Article

Difference between Singly linked list and Doubly linked list

Similar Reads

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

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....

9 min read

Difference between Singly linked list and Doubly linked list

Introduction to Singly linked list : A singly linked list is a set of nodes where each node has two fields 'data' and 'link'. The 'data' field stores actual piece of information and 'link' field is used to point to next node....

2 min read

Applications, Advantages and Disadvantages of Doubly Linked List

Doubly linked list is a type of linked list in which nodes contains information and two pointers i.e. left pointer and right pointer. The left pointer in the doubly linked list points to the previous node and the right pointer...

4 min read

Operations on Doubly Linked

Doubly Linked List in Different Languages

How to Create a Doubly Linked List in C?

A doubly linked list is a type of linked list in which each node contains a pointer to both the next node and the previous node. This allows traversal in both forward and backward directions. Each node in a doubly linked li...

4 min read

Introduction to Doubly Linked Lists in Java

Doubly linked list is a data structure that has reference to both the previous and next nodes in the list. It provides simplicity to traverse, insert and delete the nodes in both directions in a list. In a doubly linked list,...

11 min read

Doubly Linked List in Python

[mtouchquiz 2151]

1 min read

Implementation of Doubly Linked List in JavaScript

This article will demonstrate the Implementation of Doubly Linked List In JavaScript. A 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...

4 min read

JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Python Tkinter
Python Web Scraping
OpenCV Tutorial
Python Interview Question
Django

Computer Science
Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths
Software Development
Software Testing

DevOps
Git
Linux
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design
High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

Inteview Preparation
Competitive Programming
Top DS or Algo for CP
Company-Wise Recruitment Process
Company-Wise Preparation
Aptitude Preparation
Puzzles

School Subjects
Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

GeeksforGeeks Videos
DSA
Python
Java
C++
Web Development
Data Science
CS Subjects