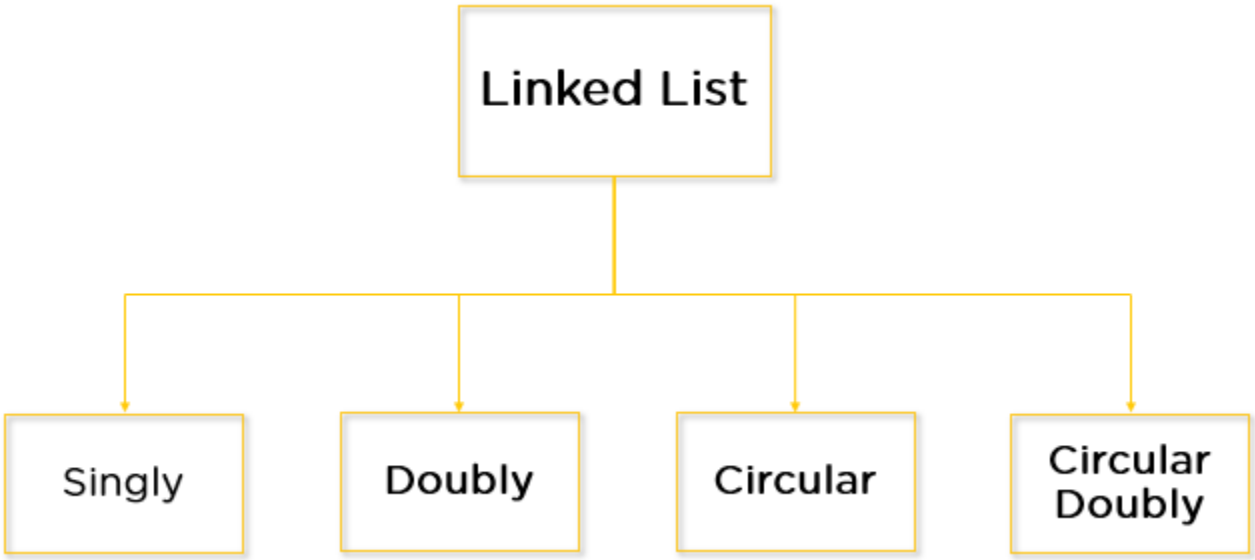
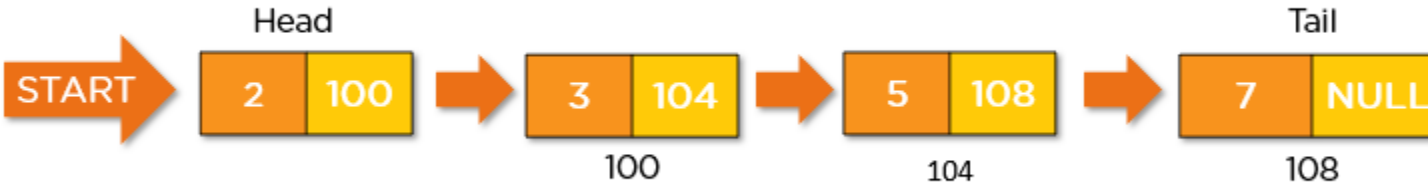
What Are the Types of Linked Lists?



There are four key types of linked lists:

- Singly linked lists
- Doubly linked lists
- Circular linked lists
- Circular doubly linked lists

What is a Singly Linked List?



A singly linked list is a unidirectional linked list. So, you can only traverse it in one direction, i.e., from head node to tail node.

Structure of Singly Linked List

There are many applications for singly linked lists. One common application is to store a list of items that need to be processed in order. For example, a singly linked list can be used to store a list of tasks that need to be completed, with the head node representing the first task to be completed and the tail node representing the last task to be completed.

Singly linked lists are also often used in algorithms that need to process a list of items in reverse order. For example, the popular sorting algorithm quicksort uses a singly linked list to store the list of items that need to be sorted. By processing the list in reverse order, quicksort can sort the list more efficiently.

Creation and Traversal of Singly Linked List

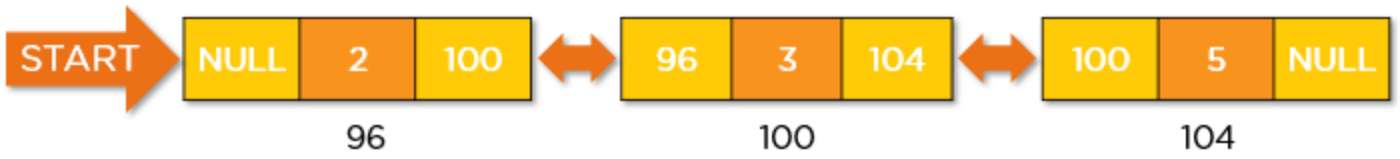
A linked list is a data structure that stores a sequence of elements. Each element in the list is called a node, and each node has a reference to the next node in the list. The first node in the list is called the head, and the last node in the list is called the tail.

To create a singly linked list, we first need to create a node class. Each node will have two data members: an integer value and a reference to the next node in the list. Next, we need to create a LinkedList class. This class will have two data members: a head node and a tail node. The head node will store the first element in the list, and the tail node will store the last element in the list.

To add an element to the list, we need to create a new node and set the next reference of the previous tail node to point to the new node. Then, we can set the new node as the new tail of the list.

To remove an element from the list, we need to find the node that contains the value that we want to remove. We can do this by traversing the list until we find the node with the matching value. Once we find the node, we need to set the next reference of the previous node to point to the next node. To search for an element in the list, we need to traverse the list until we find the node with the matching value. To traverse the list, we can start at the head node and follow the next references until we reach the tail node.

What is a Doubly Linked List?



A doubly linked list is a bi-directional linked list. So, you can traverse it in both directions. Unlike singly linked lists, its nodes contain one extra pointer called the previous pointer. This pointer points to the previous node.

Structure of Doubly Linked List

A doubly linked list of singly linked lists is a data structure that consists of a set of singly linked lists (SLLs), each of which is doubly linked. It is used to store data in a way that allows for fast insertion and deletion of elements.

Each SLL is made up of two parts: a head and a tail. The head of each SLL contains a pointer to the first element in the list, and the tail contains a pointer to the last element.

It is advantageous over other data structures because it allows for quick insertion and deletion of elements. Additionally, it is easy to implement and can be used in a variety of applications.

Creation and Traversal of Doubly Linked List

A doubly linked list is a type of data structure that allows for the storage of data in a linear fashion, much like a singly linked list. However, unlike a singly linked list, a doubly linked list allows for both forward and backward traversal of the data stored within it. This makes it an ideal data structure for applications that require the ability to move both forward and backward through a list of data.

To create a doubly linked list, we first need to create a Node class that will be used to store our data. This Node class will have two attributes: data and next. The data attribute will be used to store the actual data that we want to store in our list, and the next attribute will be used to store a reference to the next node in the list.

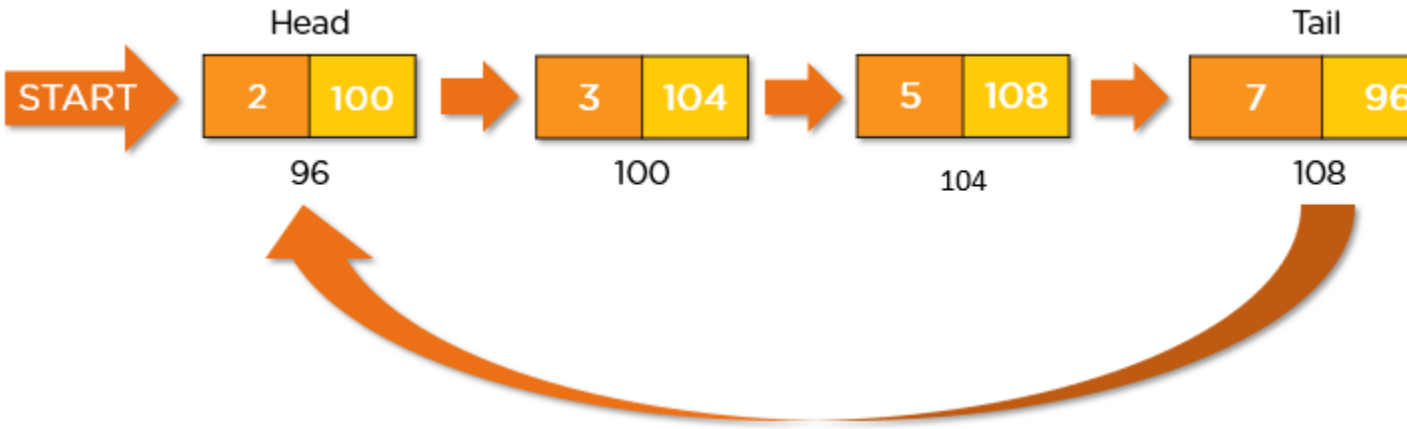
Once we have our Node class, we can create our doubly linked list. To do this, we need to create a class that will represent our list. This class will have two attributes: head and tail. The head attribute will be used to store a reference to the first node in our list, and the tail attribute will be used to store a reference to the last node in our list.

After the list class is created, we can begin adding data to it. To add data to our list, we need to create a new node and set its data attribute to the data that we want to add. Then, we need to set the next attribute of the new node to point to the head node of our list. Finally, we need to set the head node of our list to point to the new node.

Now that we know how to add data to our list, we can write a function to traverse our list. To do this, we need to create a variable that will keep track of the current node that we are traversing. We can set this variable to the head node of our list to start. Then, we need to create a while loop that will continue until the current node is None, which indicates that we have reached the end of our list.

Within our while loop, we need to print out the data that is stored in the current node. Then, we need to set the current node to the next node in our list before continuing to the next iteration of the loop.

What is a Circular Linked List?



A circular Linked list is a unidirectional linked list. So, you can traverse it in only one direction. But this type of linked list has its last node pointing to the head node. So while traversing, you need to be careful and stop traversing when you revisit the head node.

Structure of Circular Linked List

A circular linked list is a type of data structure that uses linked list technology to store data in a linear, sequential fashion. However, unlike a traditional linked list, a circular linked list has no beginning or end – it is essentially a ring of nodes. This makes circular linked lists ideal for

applications where data needs to be processed in a continuous loop, such as in real-time applications or simulations.

Circular linked lists are typically implemented using a singly linked list data structure. This means that each node in the list is connected to the next node via a pointer. The last node in the list is then connected back to the first node, creating the ring-like structure.

Accessing data in a circular linked list is similar to accessing data in a traditional linked list. However, because there is no defined beginning or end to the list, it can be difficult to know where to start traversing the list. As a result, many implementations of circular linked lists use a "head" pointer that points to the first node in the list.

Circular linked lists have a number of advantages over traditional linked lists. First, because there is no defined beginning or end to the list, data can be added and removed from the list at any time. This makes circular linked lists ideal for applications where data needs to be constantly added or removed, such as in a real-time application.

Second, because data is stored in a ring-like structure, it can be accessed in a continuous loop. This makes circular linked lists ideal for applications where data needs to be processed in a continuous loop, such as in a real-time application or simulation.

Third, because there is no defined beginning or end to the list, circular linked lists are typically more efficient than traditional linked lists when it comes to memory usage. This is because traditional linked lists often require additional memory for pointers that point to the beginning and end of the list. Circular linked lists, on the other hand, only require a single pointer to be stored in memory – the head pointer.

Finally, circular linked lists are often easier to implement than traditional linked lists. This is because traditional linked lists often require the use of additional data structures, such as stacks and queues, to keep track of the list's beginning and end. Circular linked lists, on the other hand, only require a singly linked list data structure.

Creation and Traversal of Circular Linked List

A circular linked list is a type of data structure that can store a collection of items. It is related to both the singly linked list and the doubly linked list. Unlike a singly linked list, which has a NULL pointer at the end of the list, a circular linked list has a pointer that points back to the first node in the list. This makes it possible to traverse the entire list without having to keep track of the end of the list.

There are two types of circular linked lists: singly linked and doubly linked. In a singly linked circular linked list, each node has a pointer that points to the next node in the list. The last node

in the list points back to the first node. In a doubly linked circular linked list, each node has pointers that point to both the next node and the previous node.

Circular linked lists have many applications. They can be used to implement queues, stacks, or deques. They can also be used for applications that require circular buffers or circular arrays.

There are two ways to create a circular linked list:

1. Create a singly linked list and make the last node point to the first node.
2. Create a doubly linked list and make the last node point to the first node and the first node point to the last node.

To create a singly linked circular linked list, we first need to create a singly linked list. We can do this by creating a Node class and a LinkedList class. The Node class will represent each node in the list, and the LinkedList class will represent the list itself.

To create a doubly linked circular linked list, we first need to create a doubly linked list. We can do this by creating a Node class and a LinkedList class. The Node class will represent each node in the list, and the LinkedList class will represent the list itself.

Once we have created our Node and LinkedList classes, we can create a circular linked list by creating a few nodes and linking them together. To do this, we first need to create a few nodes. Then, we need to link the nodes together to form a circular linked list. We can also link the nodes together in a doubly linked list.

There are two ways to traverse a circular linked list:

1. Traverse the list until you reach the head node again.
2. Keep track of the nodes you have visited.

To traverse the list until you reach the head node again, you can use a while loop. If you want to keep track of the nodes you have visited, you can use a list or set.

What is a Circular Doubly Linked List?

A circular doubly linked list is a mixture of a doubly linked list and a circular linked list. Like the doubly linked list, it has an extra pointer called the previous pointer, and similar to the circular linked list, its last node points at the head node. This type of linked list is the bi-directional list. So, you can traverse it in both directions.

Structure of Doubly Circular Linked List

A doubly circular linked list (DCL) is a variation of the standard doubly linked list. In a DCL, the first and last nodes are linked together, forming a circle. This allows for quick and easy traversal of the entire list without the need for special case handling at the beginning or end of the list. DCLs are often used in applications where data needs to be processed in a sequential fashion, such as in a video or audio player. The circular nature of the list allows for quick and easy movement from one node to the next without the need for special case handling. DCLs are also sometimes used in applications where data needs to be accessed randomly, such as in a database. In this case, the circular nature of the list allows for quick and easy movement to any node in the list without the need for special case handling.

Creation and Traversal of Doubly Circular Linked List

To create a doubly circular linked list, we first need to create a node structure that will hold our data and the pointers. We can then create a head pointer and initialize it to null. Next, we need to create a function to insert new nodes into the list. This function should take as input the data that we want to insert and the head pointer. It should then create a new node with this data and insert it into the list. Finally, we need to write a function to traverse the list and print out the data contained in each node. This function should take as input the head pointer. It should then loop through each node in the list, printing out the data contained in that node.

Next, you will explore some of the applications of these linked lists.

What Are the Applications of Different Types of Linked Lists?

- A linked list is used to implement stack and queues
- A linked list is used to represent sparse matrices
- You can implement an image viewer using a circular linked list
- You can use the linked list concept to navigate through web pages
- You can use a circular doubly linked list to implement a Fibonacci heap