Lesson:

Linked Lists

PW SKILLS

Pre-Requisites

- Classes
- Constructors

List of Concepts Involved

- Challenges of array
- What is a linked list
- Advantages of a linked list over an array
- Listnode
- Types of linked lists
- Implementation of a node in a singly linked list
- Display a singly linked list
- Insertion at kth position in a linked list
- Retrieving element at kth position in a singly linked list
- Deletion at kth position in a linked list

Topic 1: Challenges of array

- Linked list is a linear data structure. Many may think we have already learnt about arrays, so why do we need to learn about another linear data structure? The answer to this is simple, when we were learning about arrays, you might have noticed certain things-

1. The size of an array needs to be defined at the time of declaration and it cannot be updated. So, in cases where we don't know how much data we are going to store, we may end up doing one of the 2 things-
   a. Defining an array which is too long.For example, let's say we are trying to make an array in which we decide to store the names of all the students that are currently following the PW coding series. Initially we didn't know the exact number of students so we decided to give the array a size of 250k. However, when the entries were made, it was found out that there were only 200k students present i.e. we created an array with 50k additional memory blocks that remains unused. In other words, a lot of memory is wasted.
   b. Defining an array which is too short. To understand this let's take the same example again. This time let's say there was a sudden spike in the number of students following the series that led to additional 100k followers, giving a total of 300k. This time we realize that the size of the array given wasn't enough to store all the names, again which is a problem.
2. Addition or deletion of elements at any position in a dynamic array is an O(n) operation.
   a. arr = [1, 2, 3, 4, 5]
   We need to insert 6 into the array at index 2.
   Step 1: shift all the elements starting from the index 2 to the right by one place.
   arr = [1, 2, _, 3, 4, 5]
   Step 2: place the new element at index 2.
   arr = [1, 2, 6, 3, 4, 5]
   b. arr = [1, 2, 6, 3, 4, 5]
   We need to delete 3 from the array.
   Step 1: delete 3 from the array.
   arr = [1, 2, 6, _, 4, 5]
   Step 2: shift all the remaining elements one place to the left to fill up the gap.

arr = [1, 2, 6, 4, 5]

At this point many may say that how are we changing the size of the array when we know that it is fixed. The answer to this is that we are not. While implementing such an array, we define a large array and fill the unused indexes with bogus values. For example,

arr = [1, 2, 3, 4, 5] may actually be an array of size 8, arr = [1, 2, 3, 4, 5, -1, -1, -1] or more.

3. Array requires contiguous memory location. And for arrays as big as 200k, we would require such a big contiguous space in memory which might not always be available.

Topic 2: What is a linked list?

- A linked list is a linear data structure that stores data in the form of nodes that are connected to each other with the help of pointers.

- Unlike arrays, these nodes are not present at contiguous memory locations. It is quite possible that the first node of a linked list is located at a memory address of 100, second one is at 500 and the third one is at 200. In other words, there is no relation between the memory addresses at which each node will be stored. They are stored at random locations.
- Each node, along with data, has a pointer. This pointer points to the address of the next node in the sequence. That is, the pointer of the first node will point to the address of the second node. The pointer of the second node will point to the third node. And so on. This helps us traverse through the list.
- In a linked list, each node is defined independently and then added to the list. Consequently, we can add or remove nodes from a linked list as required, without interfering with the positions of the other nodes.

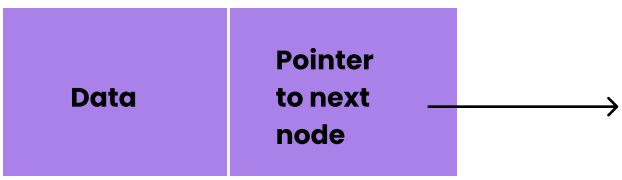
Topic 3: Advantages of a linked list over an array

In this section, let's see how a linked list helps us overcome the challenges that we faced in the case of arrays.

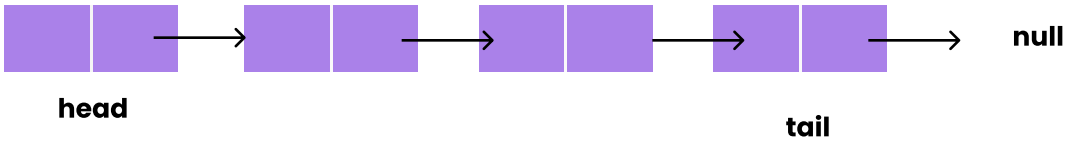
1. Dynamic size
   In a linked list, a node can be added or removed in constant time. In other words, the size of a linked list is not fixed.
2. Easy insertion and deletion of nodes
   In a linked list, each node is connected to another node with the help of pointers. So if we want to add or remove a node from a list, we just have to update the pointers of the nodes adjacent to the place where a node needs to be inserted or deleted from.
3. Efficient memory utilization
   Since in a linked list we can dynamically add and remove elements, we don't have to pre-allocate any memory. So there is no memory wastage.

Topic 4: ListNode

- A listnode or a node is a basic unit of a linked list.
- A node contains 2 parts - the data and the pointer to the successive node it is connected to.



- The first node of the list is referred to as the head node of the list and it helps us to access the elements of the list. Similarly we refer to the last node of the list as the tail node of the list. Being the last node of the list, it points to NULL. It just means that it points to nothing.



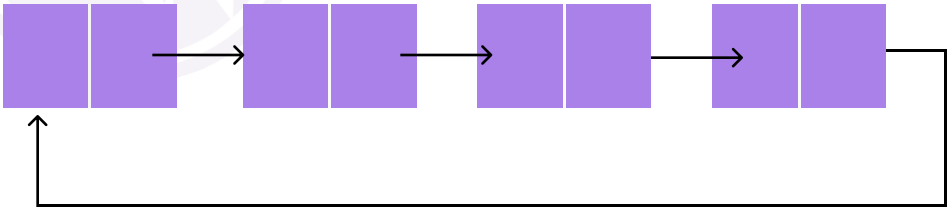
Topic 5: Types of linked lists

There are 3 types of linked lists-

1. Singly linked list:
   In general when we say linked list, we mean singly linked list. In this, each node is connected to its successive node, thus forming a chain-like structure.

2. Doubly linked list:
   It is the type of linked list where each node is connected to both its previous node and next node in the list.

3. Circular linked list:
   It is the type of linked list where in addition to normal connections, we connect the tail with the head of the list, thus forming a circle.



Topic 6: Implementation of a node in a singly linked list

- A node of a singly linked list consists of 2 parts- the data and the pointer to the succeeding node.

Code

- The stored data of a linked list can be of any data type and can have any number of data variables.

Examples

Topic 7: Display a Singly linked list

- In a singly linked list, each node is connected to its successor by a pointer. So if we want to go from a node to another the only way to do that will be through that pointer. This implies that unlike arrays, here we have no indexing of the nodes in the linked list.
- As a result we follow certain steps to traverse the linked list-
  1. Make a new node pointer for the traversal and make it point to the head of the linked list.
  2. At each node, process the data and update the value of the pointer using the next pointer of the current node such that we start pointing to the next node.

Code

Explanation

1. The *curr* node here represents the node we are currently processing.
2. Loop till *curr* node isn't equal to NULL i.e. we have not reached the end of the list. NULL node is the node to which the tail node of the linked list points to. It simply marks the end of the list.
3. At each iteration, process the data in the curr node and update the curr to move on to the next node in the sequence.

Note: The reason we create a new node for the traversal is that if we use the head node for the same, then once the traversal is over, we will lose the track of the head of the linked list. As a result we won't be able to access any of the elements of the list ever again.

Try this

Implement a method to find out the length of a Linked List (Iterative and Recursive)

1. Iterative approach
   a. Loop till head doesn't point to null.
   b. At each iteration, increment the count of nodes and move to the next node.

Code

2. Recursive approach
   a. Base case: the current node is null. In that case return 0.
   b. At each recursive iteration return the 1 + the length of the remaining list.

Code

Topic 8: Insertion at k^{th} position in a singly linked list

- There are 3 major ways to add a node to a linked list-

1. Add a node at the end
2. Add a node at the start
3. Add a node at an arbitrary position

Add a node at the end

Steps-

1. Make a new node to insert in the list and assign it to a variable to it.
2. Find the tail node and make the tail node point to this node.
3. Update the value of the tail node.
4. Return the new head of the list.

Code

Time complexity- O(n)

Note: In some cases you may be provided with the tail node. In those you don't have to traverse the entire list to reach the tail node. The time complexity for this type of insertion will be O(1).

Add a node at the start

- This is the simplest way of inserting an element in a linked list.

Steps-

1. Make a new node to insert in the list and assign it to a variable to it.
2. Make this new node point to the head of the linked list.
3. Return the new head of the list.

Code

Time complexity- O(1)

Add a node at an arbitrary position

- One of the disadvantages with linked lists, as we can already see, is that there is no way for us to directly jump to the respective node. We need to traverse the list for that.

Steps-

1. Make a new node to insert in the list and assign it to a variable.
2. Traverse the linked list till you reach the position after which the new node has to be inserted.
3. Make the new node point to the node after the prevNode, and then make the prevNode point to the new node.
4. Return the new head of the list.

Code

Time complexity- O(n)

Topic 9: Retrieving element at kth position in a singly linked list

1. Make a new node to traverse the list.
2. Traverse the list till you reach the required node.
3. Return the value of the required node

Code

Time complexity- O(n)

Topic 10: Deletion at kth index in a singly linked list

- **Steps-**

1. Traverse the linked list till you reach the position after which the node that has to be deleted is present.
2. Make a temporary node variable and point it to the node that has to be deleted.
3. Make the prevNode point to the node after the node to be deleted.
4. Return the head of the updated list.

CODE

Time complexity- O(n)

Upcoming Class Teasers

- Problems related to linked list