Lesson:

Doubly and Circular linked list

PW SKILLS

Pre-Requisites

- Singly linked list

List of Concepts Involved

- What is a doubly linked list?
- Advantages of a doubly linked list over a singly linked list
- Disadvantages of a doubly linked list over a singly linked list
- Implementation of a node in a doubly linked list
- Traversal in a doubly linked list
- Insertion at kth position in a doubly list
- Updation at kth position in a doubly list
- Deletion at kth position in a doubly list
- Deletion in a doubly linked list when the pointer to the node to be deleted is given
- Problems based on doubly linked list
- What is a circular linked list?
- Advantages of a circular linked list over a singly linked list
- Types of circular linked lists
- Implementation of a node in a circular linked list
- Traversal in a circular linked list
- Insertion at kth position in a circular linked list
- Updation at kth position in a circular linked list
- Deletion at kth position in a circular linked list

Topic 1: What is a doubly linked list?

A doubly linked list is a type of linked list in which we have an additional pointer that points to the previous node of the list. This helps in bidirectional traversal of the list.

Node

Topic 2: Advantages of a doubly linked list over a singly linked list

1. Bidirectional traversal: The presence of an additional pointer that points to the previous node, allows us to traverse in both forward and backward directions.
2. Easier and more efficient in insertion and deletion of nodes: Because of bidirectional traversal the insertion and deletion of nodes becomes easy in doubly linked lists.

Topic 3: Disadvantages of a doubly linked list over a singly linked list

1. In a doubly-linked list, each node has an extra pointer which requires extra space.
2. Doubly linked list operations require more pointers to be handled.

Topic 4: Implementation of a node in a doubly linked list

- A node of a doubly linked list consists of 3 parts- the data, the pointer to the succeeding node and the pointer to the preceding node.

CODE LINK: https://pastebin.com/3tKxwf89

- Just like in a singly linked list, we can store multiple variables in data of various data types. For example,

CODE LINK: https://pastebin.com/zLtDM1BA

- In case of a doubly linked list, there are 2 pointers that point to NULL-

1. The prev pointer of the head node. It represents that there is no node before the head node.
2. The next pointer of the tail node. It represents that there is no pointer after the tail node.

Topic 5: Traversal in a doubly linked list

- In a doubly linked list there are 2 types of traversals- forward and backward. We use the next pointer of a node to move in forward direction and the prev pointer to move in backward direction.
- **Forward direction**

Steps-

1. Make a new node variable for the traversal and assign it to the head of the linked list.
2. Loop till you encounter the end of the list or the required node.A. process the data.B. update the value of the variable being used for traversal using the next pointer of the current node.

CODE LINK: https://pastebin.com/fzYPbJJ6

Time complexity- O(n)

- **Backward direction**

Steps-

1. Make a new node variable for the traversal and assign it to the tail of the linked list.
2. Loop till you encounter the start of the list or the required node.A. process the data.B. update the value of the variable being used for traversal using the prev pointer of the current node.

CODE LINK: https://pastebin.com/fzYPbJJ6

Time complexity- O(n)

- **Backward direction**

Steps-

1. Make a new node variable for the traversal and assign it to the tail of the linked list.
2. Loop till you encounter the start of the list or the required node.A. process the data.B. update the value of the variable being used for traversal using the prev pointer of the current node.

CODE LINK: https://pastebin.com/iWrDXCME

Time complexity- O(n)

Topic 6: Insertion at kth position in a doubly linked list

Add a node at the start

- Steps-

1. If the head is NULL, then replace it with a new node with the given value and return.
2. Make the new node point to the current head of the list using its next pointer and make the current head of the list point to it using its prev pointer.

CODE LINK: https://pastebin.com/JqVEMZfy

Time complexity- O(1)

Add a node at the end

- Steps-

1. If the head is NULL, then replace it with a new node with the given value and return.
2. Traverse the list to find the tail of the list.
3. Make the new node point to the tail of the list using its prev pointer and make the tail of the list point to it using its next pointer.
4. Return the head of the updated list.

CODE LINK: https://pastebin.com/75H1KYH6

Time complexity- O(n)

Add a node at an arbitrary position

- Steps-

1. Traverse the linked list till you reach the position after which the new node has to be inserted. Let's call it prevNode and let's call the node after it nextNode.
2. Make the prevNode and the newNode point to the newNode and the nextNode respectively using their next pointers.//image
3. Make the newNode and the nextNode point to the prevNode and the newNode respectively using their prev pointers.//image

CODE LINK: https://pastebin.com/Dxz29pCv

Time complexity- O(n)

Topic 7: Updation at kth position in a doubly linked list

Steps-

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

Code link: https://pastebin.com/W1gCr1g2

Time complexity- O(n)

Topic 8: Deletion at kth position in a doubly linked list

Delete a node from the start

- Steps-

1. Make a temporary node variable point to the current head of the linked list.
2. Make the head variable point to the second element in the linked list.
3. Make the prev pointer of the new head point to NULL using its prev pointer.
4. Return the head of the updated list.

CODE LINK: https://pastebin.com/ybrFPBjB

Time complexity- O(1)

Delete a node from the end

- Steps-

1. Make a temporary node variable to point to the current tail of the linked list.
2. Make the tail pointer point to the second last element of the list.
3. Make the new tail point to NULL using its next pointer.
4. Return the head of the updated list.

CODE LINK: https://pastebin.com/ptN9iR3K

Time complexity- O(n)

Delete a node from an arbitrary position

- 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 to point to the node that has to be deleted.
3. Make a new pointer to represent the node after the node that has to be deleted.
4. Make the prevNode point to the nextNode using its next pointer and make nextNode point to prevNode using its prev pointer.
5. Return the head of the updated list.

CODE LINK: https://pastebin.com/w8t6X9Ln

Time complexity- O(n)

Assumption: The given position is always valid i.e. given position is between 0 to n - 1 using 0-based indexing, where n is the length of the linked list.

Topic 9: Deletion in a doubly linked list when the pointer to the node to be deleted is given

For simplicity, let's call the node to be deleted, currNode.
Steps:

1. Make 2 new variables to represent the nodes before and after the node to be deleted.
2. If the prevNode is not NULL, make it point to the nextNode using its next pointer.
3. If the nextNode is not NULL, make it point to the prevNode using its prev pointer.

CODE LINK: https://pastebin.com/63QqcGyH

Problem 1: Given the head of a doubly linked list, reverse it.
The first line of input contains the value of n, number of nodes.
The second line of input contains n integers representing the values stored in the nodes of the linked list.
Example 1
Input
4
1 2 3 4

Output
4→3→2→1→null

Example 2
Input
4
3 5 20 10

Output
10→20→5→3→null

Solution:
Steps:

1. Traverse the list from the head.
2. For each node swap the values stored in next and prev pointers.
3. To move to the next node, update the value of the current node with the help of its prev pointer as after swapping the address of the next node is stored in the prev pointer.

CODE LINK: https://pastebin.com/7UVzbM2g

Problem 2: Given the head of a doubly linked list, find if it's a palindrome or not. If it is a palindrome, return true. Else return false.
The first line of input contains the value of n, number of nodes.

The second line of input contains n integers representing the values stored in the nodes of the linked list.

Example 1

Input

4
1 2 3 4

Output

false

Example 2

Input

4
1 2 2 1

Output

true

Solution:

Steps:

1. Find the tail of the linked list.
2. Traverse the list in forward direction using the head pointer and in reverse direction using the tail pointer simultaneously till they pass each other.
3. At each iteration, check if the head and the tail have the same data or not. If not then the list isn't a palindrome i.e. return false.
4. If by the end of iteration we find no pair of nodes with different data, then the list is a palindrome i.e. return true.

CODE LINK: https://pastebin.com/i3eiHWg5

Problem 3: Given the head of a doubly linked list, delete the nodes whose neighbors have the same value.
Traverse the list from right to left.
The first line of input contains the value of n, number of nodes.
The second line of input contains n integers representing the values stored in the nodes of the linked list.

Example 1

Input

5
2 1 1 2 1

Output

2→1→1→null

Explanation

Iteration-1: 1 has only one neighbor 2. So it will remain unaffected. linked list = 2→1→1→2→1→null
Iteration-2: 2 has two 1's as its neighbors. So it will be deleted. linked list = 2→1→1→1→null
Iteration-3: 1 has two 1's as its neighbors. So it will be deleted. linked list = 2→1→1→null
Iteration-4: 1 has 1 and 2 as its neighbors. So it will remain unaffected. linked list = 2→1→1→null
Iteration-5: 2 has only one neighbor. So it will remain unaffected. linked list = 2→1→1→null

Example 2

Input

4
2 1 2 1

Output

2→1→1→null

Explanation

Iteration-1: 1 has only one neighbor 2. So it will remain unaffected. linked list = 2→1→2→1→null
Iteration-2: 2 has two 1's as its neighbors. So it will be deleted. linked list = 2→1→1→null
Iteration-3: 1 has a one and a two as its neighbors. So it will remain unaffected. linked list = 2→1→1→null
Iteration 4: 2 has only one neighbor. So it will remain unaffected. linked list = 2→1→1→null

Example 3

Input

5
2 1 2 1 1

Output

2→1→1→1→null

Explanation

Iteration-1: 1 has only one neighbor 2. So it will remain unaffected. linked list = 2→1→2→1→1→null
Iteration-2: 1 has one and two as its neighbors. So it will remain unaffected. linked list = 2→1→2→1→1→null
Iteration-3: 2 has 2 ones as its neighbors. So it will be deleted. linked list = 2→1→1→1→null
Iteration 4: 1 has one and two as its neighbors. So it will remain unaffected. linked list = 2→1→1→1→null
Iteration 5: 2 has only one neighbor. So it will remain unaffected. linked list = 2→1→1→1→null

Solution:

Steps:

1. Find the tail of the linked list.
2. Starting from the second last node, traverse the list in reverse order. At each node, check if its neighbors have the same value or not.
3. If the neighbors have the same value, delete the current node.

CODE LINK: https://pastebin.com/wpxUdUxJ

Problem 4: A critical point in a linked list is defined as either a local maxima or a local minima. Given a linked list tail, return an array of length 2 containing [minDistance, maxDistance] where minDistance is the minimum distance between any two distinct critical points and maxDistance is the maximum distance between any two distinct critical points. If there are fewer than two critical points, return [-1, -1].
Note that a node can only be a local maxima/minima if there exists both a previous node and a next node.
The first line of input contains the value of n, number of nodes.
The second line of input contains n integers representing the values stored in the nodes of the linked list.
Example 1

Input

4
1 2 3 4

Output
-1 -1
Example 2
Input
6
2 1 2 5 2 3

Output
1 3

Solution:
Steps:

1. Note that the maxDistance will be the distance between the 2 critical nodes at the extremes and the minDistance will be the minimum of all the distances between any 2 adjacent critical nodes.
2. Make 2 variables to store the first and the last critical point encountered.
3. Make another variable to keep count of the number of nodes encountered so far. This will act as indexing for the nodes.
4. Initialize all the variables as -1.
5. Traverse the list in reverse order starting from the second last node as a critical node won't be at the end points.
6. If the current node is a critical node-
   a. If it is the first critical node, update the critical node and the last node variables.
   b. Else update the values of minDistance (minimum of its current value and the distance between the current node and the last critical node) and maxDistance (the distance between the current node and the first critical node).
7. Return the minDistance and the maxDistance.

CODE LINK: https://pastebin.com/GvuhS186

Problem 5: Given the head of a doubly linked list. The values of the linked list are sorted in non-decreasing order. Find if there exists a pair of distinct nodes such that the sum of their values is x. Return the pair in the form of a vector [l, r], where l and r are the values stored in the 2 nodes pointed by the pointers. If there are multiple such pairs, return any of them. If there is no such pair return [-1, -1].
The first line of input contains the value of n, number of nodes.
The second line of input contains n integers representing the values stored in the nodes of the linked list.
The third line contains the value of x.
Example 1
Input
4
1 2 3 4
10

Output
-1 -1
Example 2
Input
6

1 2 2 4 6 10
8

Output
2 6

Solution:
Steps:

1. Use 2 Node variables. Assign one to the start of the linked list and the other to the end of the linked list.
2. Loop till the first variable points to a node before the second variable.
   a. If the sum of the values in the 2 nodes, pointed by the 2 variables, is equal to x, return the pair.
   b. If the sum is less than x, shift the first variable to the right by one position.
   c. Else shift the second variable to the left by one position.
3. If no pair is found, return [-1, -1].

CODE LINK: https://pastebin.com/kJL0uEcn

Topic 10: What is a circular linked list?

A circular linked list is a normal linked list with its tail node being connected to the head node, thus forming a circle. There is no NULL pointer at the end.

Topic 11: Advantages of a circular linked list over a singly linked list

1. We can start our traversal from any node of the list. We'll stop when we encounter a visited node for the second time.
2. It is useful in cases where we have to traverse a list multiple times in a circular fashion.

Topic 12: Types of circular linked lists

1. Circular singly linked listIt is a modification of a singly linked list, where the tail node is connected to the head node of the linked list.
2. Circular doubly linked listIt is a modification of a doubly linked list, where the tail node is connected to the head node and vice versa. In this the tail node's next pointer points to the head node and the head node's prev pointer points to the tail node.

Topic 13: Implementation of a node in a circular linked list

- Since the only difference between a circular linked list and a normal linked list is that the tail is connected to the head, the implementation of the node remains the same.

Topic 14: Traversal in a circular linked list

- The forward traversal is the same for both circular singly linked list and circular doubly linked list.
- Steps-

1. Make a new node variable for the traversal and make it point to the head of the linked list.
2. Loop till you don't encounter the head of the list again. A. process the dataB. update the value of the variable being used for traversal.

CODE LINK: https://pastebin.com/sGr4XkMY

Time complexity- O(n)

- In the case of circular doubly linked lists, we have backward traversal too. The steps are the same as in forward traversal with the only change in the node value updation. In the case of backward traversal, we use the prev pointer of the node instead of the next pointer.

CODE LINK: https://pastebin.com/i465Vigy

- **Time complexity- O(n)**
- Since all the nodes in a circular list are symmetrical, it doesn't matter much if you start your traversal from the head or the tail unless specified otherwise.

Topic 15: Insertion at kth position in a circular linked list

- Due to the symmetric nature of a circle, the process to insert a node at any point is going to be the same.

Circular singly linked list

- Steps-

1. Make a new node to insert in the list and assign it to a variable.
2. Check if the list is empty. If yes, then make the newNode the head of the list and make it point to itself to make a circle.
3. Traverse the linked list till you reach the position after which the new node has to be inserted.
4. Make the newNode point to the node after the prevNode.
5. Make the prevNode point to the newNode.
6. If the position of insertion is 0, then update the head of the list.

Time complexity- O(n)Assuming that the input position is always valid.

CODE LINK: https://pastebin.com/tKnSdtqd

Circular doubly linked list

- Steps-

1. Make a new node to insert in the list and assign it to a variable.
2. Check if the list is empty. If yes, then make the newNode the head of the list and make it point to itself to make a circle using both its pointers.
3. Traverse the linked list till you reach the position after which the new node has to be inserted.
4. Make a new variable to represent the next node after the prevNode

1. Make the prevNode and newNode point to the newNode and the nextNode respectively using their next pointers.
2. Make the newNode and the nextNode point to the prevNode and the newNode respectively using their prev pointers.
3. If the position of insertion is 0, then update the head of the list.

Time complexity- O(n)

CODE LINK: https://pastebin.com/3QQacyeh

Topic 16: Updation at kth position in a circular linked list

- The way of updation is the same in both circular singly linked list and circular doubly linked list.
- Steps-

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

CODE LINK: https://pastebin.com/EuqPQxkn

Time complexity- O(n)

Topic 17: Deletion at kth position in a singly linked list

- Same as insertion, the deletion of a node is symmetric for all nodes in the list.

Circular 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.

1. Make a temporary node variable and assign it the node that has to be deleted.
2. Make the prevNode point to the node after the node to be deleted.

1. If the position of deletion is 0, update the head pointer.
2. Update the size of the list.

Time complexity- O(n)

CODE LINK: https://pastebin.com/NApFV8Eq

Circular doubly linked list

- Steps-

1. Traverse the linked list till you reach the position after which the node that has to be deleted is present.

1. Make a temporary node variable and assign it the node that has to be deleted.
2. Make a new variable representing the node after the node to be deleted.

1. Make a new variable representing the node after the node to be deleted.
2. Make the prevNode point to the nextNode using its next pointer and make the nextNode point to the prevNode using its prev pointer.
3. If the position of deletion is 0, update the head pointer.
4. Update the size of the list.

Time complexity- O(n)

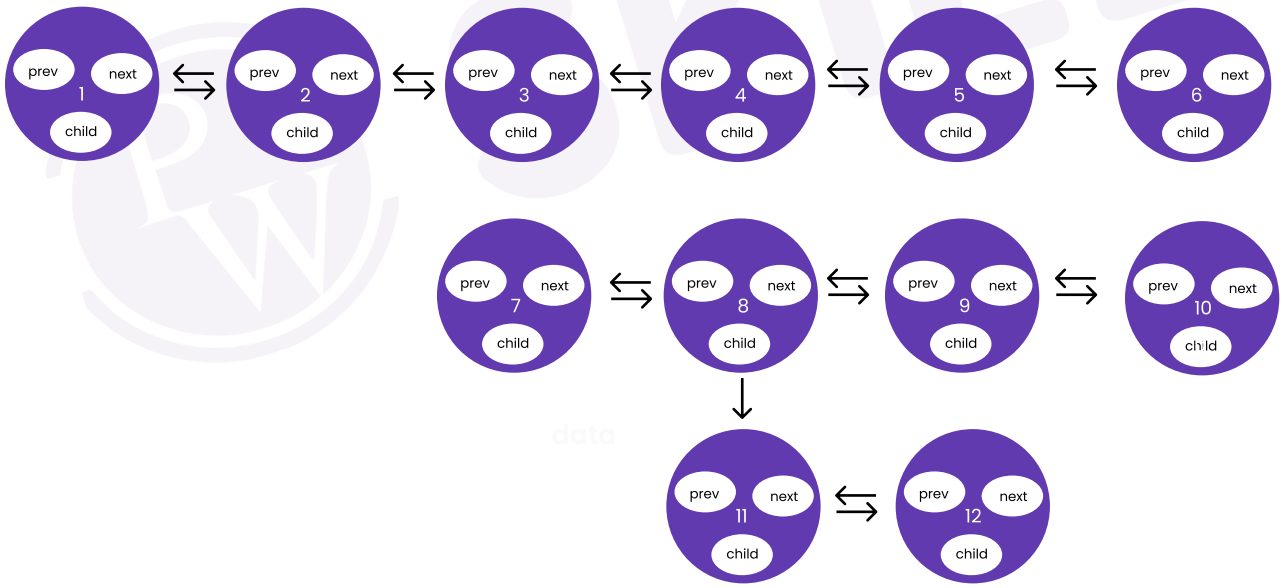
CODE LINK: https://pastebin.com/9AFWuugQ

Problem 1: Leetcode 430. Flatten a Multilevel Doubly Linked List
You are given a doubly linked list, which contains nodes that have a next pointer, a previous pointer, and an additional child pointer. This **child pointer** may or may not point to a separate doubly linked list, also containing these special nodes. These child lists may have one or more children of their own, and so on, to produce a **multilevel data structure** as shown in the example below.

Given the **head** of the first level of the list, **flatten** the list so that all the nodes appear in a single-level, doubly linked list. Let **curr** be a node with a child list. The nodes in the child list should appear **after curr** and **before curr.next** in the flattened list.

Return *the head of the flattened list. The nodes in the list must have all of their child pointers set to null.*

Example 1:



Input: head = [1,2,3,4,5,6,null,null,null,7,8,9,10,null,null,11,12]
Output: [1,2,3,7,8,11,12,9,10,4,5,6]

Explanation: The multilevel linked list in the input is shown.
After flattening the multilevel linked list it becomes:

Solution:
Approach:

1. Start form the head , move one step each time to the next node
2. When meet with a node with child, say node p, follow its child chain to the end and connect the tail node with p.next, by doing this we merged the child chain back to the main thread
3. Return to p and proceed until find the next node with the child.
4. Repeat until reach null

Code Link: https://pastebin.com/1KFGnHM1

Upcoming Class Teasers

- Stacks