Lesson:

Doubly linked list

PW SKILLS

Concepts involved

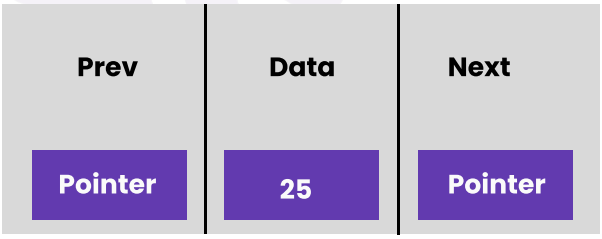
- Pointers
- Singly linked list

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 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

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.



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

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

Code

- 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 pointer for the traversal and make it point 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 pointer being used for traversal using the next pointer of the current node.

Code

Time complexity- O(n)

- **Backward direction**

Steps-

1. Make a new node pointer for the traversal and make it point 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 pointer being used for traversal using the prev pointer of the current

Code

Time complexity- O(n)

Topic 6: Insertion at k^{th} 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.
3. Return the new head of the list.

Code

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

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 prev_node and let's call the node after it next_node.
2. Make the prev_node and the new_node point to the new_node and the next_node respectively using their next pointers.
3. Make the new_node and the next_node point to the prev_node and the new_node respectively using their prev pointers.

Code

Time complexity- O(n)

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

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

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 pointer to point to the current head of the linked list.
2. Make the head pointer 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. Free the memory space that was taken by the previous head.
5. Return the head of the updated list.

Code

Time complexity- O(1)

Delete a node from the end

- Steps-

1. Make a temporary node pointer to point to the current tail of the linked list.

1. Make the tail pointer point to the second last element of the list.
2. Make the new tail point to NULL using its next pointer.
3. Free the space taken up by the deleted node.
4. Return the head of the updated list.

Code

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 pointer 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 prev_node point to the next_node using its next pointer and make next_node point to prev_node using its prev pointer.
5. Free the space taken up by the deleted node.
6. Return the head of the updated list.

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.

Code

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, curr_node.

Steps:

1. Make 2 new pointers to represent the nodes before and after the node to be deleted.
   Node *prev_node = curr_node->prev, *next_node = curr_node->next;
2. If the prev_node is not NULL, make it point to the next_node using its next pointer.prev_node->next = next_node;
3. If the next_node is not NULL, make it point to the prev_node using its prev pointer.next_node->prev = prev_node;

Code

Topic 10: Problems based on doubly linked list

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 the nodes of the linked list.
Example 1
Input
4
1 2 3 4
Output
4→3→2→1

Example 2
Input
4
3 5 20 10
Output
10→20→5→3

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 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/Nxm8pQwn

Problem 2: Given the head of a doubly linked list, find if it's a palindrome or not. If it is a palindrome, return 1. Else return 0.
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
No

Example 2
Input
4
1 2 2 1
Output
Yes

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/USETu0Vt

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

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

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

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/FASBLq1W

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: https://pastebin.com/6tUx4z3z

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

CODE: https://pastebin.com/k50rry53

Upcoming Class Teasers:

- Circular linked list and stl.