Deletion and Reverse in Circular Linked List

solved by	Senan
Platform	GeeksForGeeks
difficulty	Medium
tags	Linked List
language	C++
solved on	@04/10/2024
link	https://www.geeksforgeeks.org/problems/deletion-and-reverse-in-linked-list/1
Completion	✓

Intuition

In this problem, we are working with a circular linked list. The goal is to perform two operations:

1. Reverse the circular linked list.
2. Delete a node by a given key from the circular linked list.

Reversing:

Reversing a circular linked list is a bit tricky compared to a normal linked list due to the circular nature of the list. We need to ensure that after reversing, the circular structure is maintained. Essentially, the first node's `next` will point to the last node, and the list will remain connected in a circular fashion.

Deletion:

For deletion, we need to traverse the list, find the node that contains the key, and then remove it while ensuring that the circular connection is maintained.

Approach

Reversing the Circular Linked List:

1. **Initial Check**: If the list is empty, we return `NULL`.
2. **Single Node Case**: If there's only one node, it already forms a valid circular list, so no action is needed.
3. **Traverse and Reverse**: We maintain pointers to reverse the `next` links while iterating through the list. Since the list is circular, we stop when we return to the `head`.

Deleting a Node by Key:

1. **Initial Check**: If the list is empty, we return `NULL`.
2. **Single Node Case**: If there is only one node, and it matches the key, we delete the node and return `NULL` since the list becomes empty.
3. **Find and Delete**: We traverse the list, find the node with the given key, and delete it by adjusting the `next` pointer of the previous node to the next node. We also need to maintain the circular nature of the list after deletion.

Complexity

Time Complexity:

- **Reversing**: Traversing the entire circular linked list requires `O(n)` time, where `n` is the number of nodes.
- **Deleting**: In the worst case, we traverse the entire list to find the node with the given key, so the time complexity is also `O(n)`.

Space Complexity:

- Both functions operate in constant space, as they only use a few extra pointers for traversal and manipulation. Hence, the space complexity is `O(1)`.

Code

```
class Solution {
  public:
    // Function to reverse a circular linked list
    Node* reverse(Node* head) {
        if(head == NULL) return NULL;

        // Initialize pointers to traverse and reverse the list
        Node *curr = head->next;
        Node *prev = head;

        // Loop through the list until we circle back to the head
        while(curr != head) {
            Node* next = curr->next;
            curr->next = prev;
            prev = curr;
            curr = next;
        }

        // Adjust the head's next pointer to point to the new tail
        head->next = prev;
        return prev;
    }

    // Function to delete a node from the circular linked list by key
    Node* deleteNode(Node* head, int key) {
        if(head == NULL) return NULL;

        // If there's only one node and it matches the key
        if(head == head->next && head->data == key) {
            delete head;
            return NULL;
        }

        Node *curr = head;

        do {
            if(curr->data == key) {
                // Swap data with the next node and delete the next node
                swap(curr->data, curr->next->data);
                Node* temp = curr->next;
                curr->next = temp->next;
                delete temp;
                break;
            }
            curr = curr->next;
        } while(curr != head);

        return head;
    }
};
```