Split Linked List in Halves CodeStudio

Implement a program to split a circular linked list into two halves.

Example:

1 -> 2 -> 9 -> 7 -> 5 -> 2 -> 4 -> 3 -> head (1)

After splitting the Linked List:

First Half: 1 -> 2 -> 9 -> 7 -> head (1)

Second Half: 5 -> 2 -> 4 -> 3 -> head (5)

Approach 1: Splits the circular linked list into two halves using length calculation

1. Traverse the circular linked list to calculate its length.
2. Find the middle of the circular linked list based on the length.
3. Update the next pointer of the last node of the left half to point to the head.
4. Update the next pointer of the last node of the right half to point to the head of the right half.
5. Return a pair with the head of the left half as the first element and the head of the right half as the second.
6. **Time Complexity:**
    - Traversing the circular linked list to calculate its length: O(n)
    - Finding the middle based on length: O(1)
    - Updating pointers: O(1).
    - **Overall, the time complexity is O(n) due to traversing the circular linked list once.**
7. **Space Complexity**:
    1. Pointers and variables used for traversal and calculations: O(1)
    2. **Overall, the space complexity is O(1) since the approach only uses a constant amount of additional space.**

Approach 2: Splits the circular linked list into two halves using optimized approach

1. Initialize two pointers, slowNode and fastNode, both initially pointing to the head of the circular linked list.

2. Move the slowNode one step and the fastNode two steps in each iteration until the fastNode reaches the end or the second-to-last node.
3. The slowNode will be at the midpoint, and the fastNode will be at the end or the second-to-last node.
4. Update the next pointer of the last node of the right half to point to the head.
5. Update the next pointer of the slowNode to point to the head of the right half.
6. Return a pair with the head of the left half as the first element and the head of the right half as the second.
7. **Time Complexity**:
    - Traversing the circular linked list with two pointers: O(n)
    - Updating pointers: O(1)
    - **Overall, the time complexity is O(n) due to traversing the circular linked list once.**
8. **Space Complexity**:
    - Pointers used for traversal and calculations: O(1)
    - **Overall, the space complexity is O(1) since the approach only uses a constant amount of additional space**.