MCS – 253P ADVANCED PROGRAMMING AND PROBLEM SOLVING

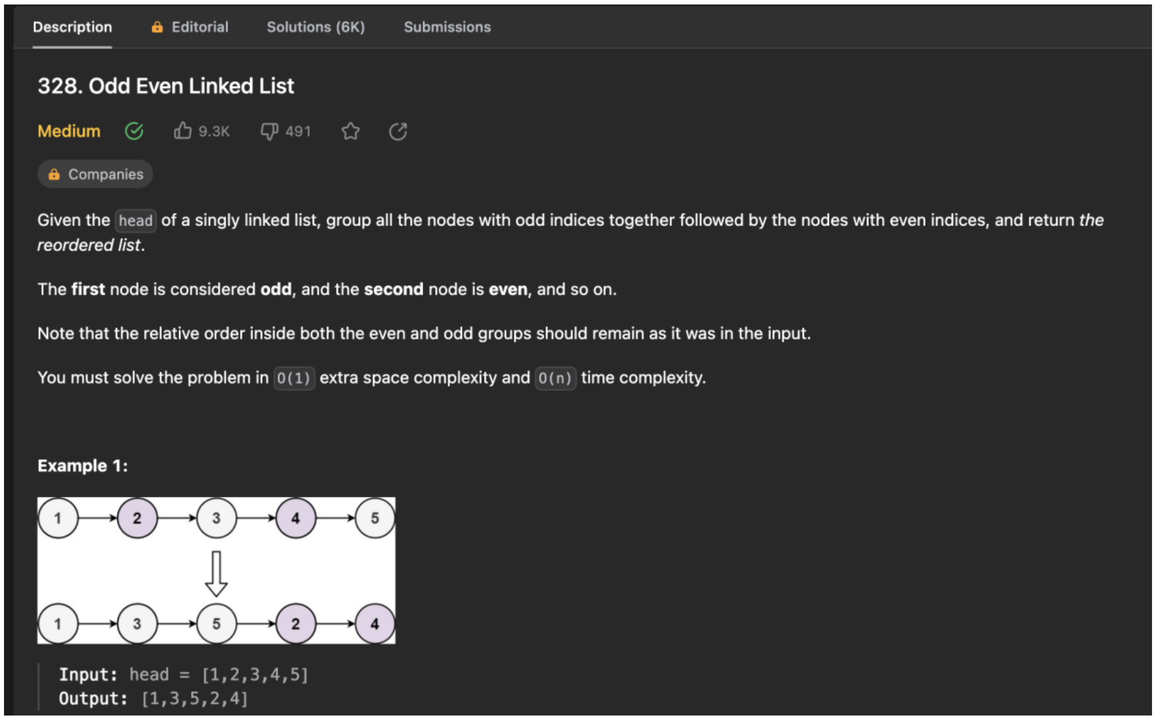
LAB 5 Write Up (Odd Even LinkedList)

Aswin Sampath
saswin@uci.edu

53844684

Date: 04/11/2023

Question:



328. Odd Even Linked List

Given the `head` of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*.

The **first** node is considered **odd**, and the **second** node is **even**, and so on.

Note that the relative order inside both the even and odd groups should remain as it was in the input.

You must solve the problem in `O(1)` extra space complexity and `O(n)` time complexity.

Example 1:

```
Input: head = [1,2,3,4,5]
Output: [1,3,5,2,4]
```

Understanding the Problem

We are given a singly linked list, and our goal is to reorder it such that all nodes with odd indices (1-based) appear together, followed by nodes with even indices, while maintaining the relative order within both groups. We need to solve this problem using O(1) extra space complexity and O(n) time complexity. To clarify the problem: The first node is considered an odd index (1st node). The second node is considered an even index (2nd node). The third node is an odd index (3rd node), and so on.

Identifying Edge Cases

- An empty linked list:
  When the list is empty, no reordering is required, and the same empty list should be returned.
- A single-node linked list: When the list has only one node, there are no other nodes to reorder, so the same list should be returned.
- A two-node linked list: When the list has exactly two nodes, no reordering is required. The order remains unchanged.

Effective Test Cases

- Input with odd and even indices mixed: [1, 2, 3, 4, 5]
  Expected Output: [1, 3, 5, 2, 4]
- Input with odd and even indices mixed: [2, 1, 3, 5, 6, 4, 7]
  Expected Output: [2, 3, 6, 7, 1, 5, 4]
- Empty linked list: []
  Expected Output: []
- Single-node linked list: [1]
  Expected Output: [1]
- Two-node linked list: [1, 2]
  Expected Output: [1, 2]

Algorithmic Solution

1. Check for edge cases (empty or single-node list). If the list is empty or has only one node, return the same list.
2. Initialize two pointers, oddHead and evenHead, to the first and second nodes, respectively. These will represent the heads of the odd and even groups.
3. Initialize two more pointers, odd and even, to keep track of the current nodes in each group.
4. Use a while loop to iterate through the list. Inside the loop:
5. Update odd->next to point to the next odd-indexed node (skip the even node).
6. Update odd to the next odd-indexed node.
7. Update even->next to point to the next even-indexed node (skip the odd node).
8. Update even to the next even-indexed node.
9. After the loop, connect the end of the odd group (odd->next) to the head of the even group (evenHead).
10. Return oddHead as the reordered list, where odd-indexed nodes come before even-indexed nodes.

Time and Space Complexity Analysis

The time complexity of this algorithm is O(n) because we traverse the entire list once, where n is the number of nodes in the linked list.

The space complexity is O(1) because we use a constant amount of extra space to store pointers and perform the reordering in-place without using additional data structures.