Comprehensive Guide to Loops in Singly Linked Lists

GSSR

August 22, 2024

Contents

1 Introduction

This document covers various types and variations of problems related to loops in singly linked lists. Each section focuses on a specific category, providing a thorough understanding of different challenges you might encounter.

2 Detection and Identification

1. **Detect a Loop**
   - Use Floyd's Cycle-Finding Algorithm (Tortoise and Hare).
   - Use a HashSet to track visited nodes.
2. **Identify the Starting Node of the Loop**
   - Reset one pointer to the head and move both pointers one step at a time until they meet at the loop's start.
3. **Check if a Given Node is Inside a Loop**
   - After detecting the loop, check if a given node is part of the loop by iterating through the loop nodes.

3 Counting and Measuring

1. **Count the Number of Nodes in the Loop**
   - Count nodes by iterating until you reach the starting node again.
2. **Find the Length of the Linked List Including the Loop**
   - Traverse the list until you reach the start of the loop and add the count of nodes in the loop.
3. **Find the Length of the Non-Looped Portion of the List**
   - Traverse the list until the loop starts, count the nodes, and stop.

4 Modification and Removal

1. **Remove the Loop**
   - Find the last node in the loop and set its next pointer to `null`.
2. **Break the Loop at a Given Node**
   - Identify the node after which you want to break the loop and set its next pointer to `null`.
3. **Break the Loop After a Certain Number of Iterations**
   - Traverse the loop a specified number of times, then break the loop by setting the next pointer to `null`.

5 Merging and Intersection

1. **Check if Two Linked Lists Have a Common Loop**
   - Detect loops in both lists and check if they share any nodes in the loop.
2. **Find the Intersection Point of Two Linked Lists with Loops**
   - Detect loops in both lists and find where they first intersect.
3. **Merge Two Linked Lists with Loops**

- Merge two lists, each containing a loop, ensuring the merged list has only one loop.

4. **Check if Two Lists Merge Before Forming a Loop**
   - Check if two lists merge into one before forming a loop by traversing both lists before detecting any loops.

6 Advanced Operations

1. **Reverse the Nodes Inside a Loop**
   - Reverse the sequence of nodes inside the loop after detecting it.
2. **Rotate Nodes Inside the Loop by K Positions**
   - Rotate the nodes inside the loop by `k` positions after detecting the loop.
3. **Check if a Loop is Palindromic**
   - Determine if the sequence of nodes inside the loop forms a palindrome.
4. **Duplicate a Loop**
   - Create a separate loop that is a copy of the detected loop and attach it to another node in the list.

7 Traversal and Search

1. **Find the Kth Node in the Loop**
   - Find the node at the kth position within the loop after detecting it.
2. **Find the Middle Node of the Loop**
   - Detect the loop and find the middle node by advancing one pointer two steps at a time and the other one step.
3. **Find the Last Node in the Loop**
   - Identify the node whose next pointer leads back to the start of the loop.
4. **Check if a List is Circular (Entirely a Loop)**
   - Check if the linked list has a loop that includes all the nodes, i.e., it forms a perfect circle.

8 Edge Cases and Special Scenarios

1. **Single Node Loop**
   - Detect and handle cases where a single node points to itself, forming a loop.
2. **Two Node Loop**
   - Detect and handle loops formed by two nodes pointing to each other.
3. **Large Loop Near the End of the List**
   - Handle cases where the loop forms after traversing most of the list, with a small loop at the end.
4. **Multiple Loops (Complex Lists)**
   - Handle theoretical cases where modifications create multiple loops.

9 Comparisons and Variations

1. **Compare Two Loops for Equality**
   - Determine if two detected loops in two different lists are identical in terms of nodes and structure.
2. **Check if a Loop is a Subset of Another Loop**
   - Determine if all nodes in one loop are part of another loop.

10 Conversion and Reversal

1. **Convert a Loop to a Linear List**
   - Break the loop and convert it into a linear linked list.
2. **Convert a Linear List to a Circular List (Form a Loop)**
   - Attach the last node of a linear list back to a previous node, creating a loop.
3. **Reverse a List with a Loop**
   - Reverse the entire list, ensuring the loop remains intact after reversal.

11 Memory and Performance

1. **Optimize Loop Detection for Space**
   - Implement loop detection algorithms that use minimal space, such as Floyd's Cycle-Finding Algorithm.
2. **Optimize Loop Removal for Time**
   - Remove the loop in the least number of operations.

12 Conclusion

This document provides a comprehensive overview of the various problems and solutions related to loops in singly linked lists