SALEM MAINA

SCT212-0167/2022

QUIZ 2 DATA STRUCTURES AND ALGORITHMS ASSIGNMENT

1. To solve this problem, we can use Floyd's Tortoise and Hare (Cycle Detection) algorithm. The algorithm works by having two pointers, one moving at twice the speed of the other. If there is a cycle in the linked list, the two pointers will eventually meet at some point within the cycle.

Here's how we can find the node where the cycle begins:

1. Initialize two pointers, slow and fast, both starting at the head of the linked list.

2. Move slow one step at a time and fast two steps at a time.

3. If there is a cycle, the two pointers will meet at some point within the cycle.

4. Reset one of the pointers to the head of the linked list and move both pointers one step at a time until they meet again. The node where they meet is the start of the cycle.

Code in C

*#include <stdio.h>*

*struct ListNode {*

*int val;*

*struct ListNode \*next;*

*};*

*bool hasCycle(struct ListNode \*head) {*

*if (head == NULL || head->next == NULL) {*

*return false;*

*}*

*struct ListNode \*slow = head;*

*struct ListNode \*fast = head;*

*while (fast != NULL && fast->next != NULL) {*

*slow = slow->next;*

*fast = fast->next->next;*

*if (slow == fast) {*

*return true;*

*}*

*}*

*return false;*

*}*

*struct ListNode \*detectCycle(struct ListNode \*head) {*

*if (head == NULL || head->next == NULL) {*

*return NULL;*

*}*

*struct ListNode \*slow = head;*

*struct ListNode \*fast = head;*

*bool hasCycle = false;*

*while (fast != NULL && fast->next != NULL) {*

*slow = slow->next;*

*fast = fast->next->next;*

*if (slow == fast) {*

*hasCycle = true;*

*break;*

*}*

*}*

*if (!hasCycle) {*

*return NULL;*

*}*

*slow = head;*

*while (slow != fast) {*

*slow = slow->next;*

*fast = fast->next;*

*}*

*return slow;*

*}*

After detecting the cycle using Floyd's algorithm, we reset one pointer to the head and move both pointers one step at a time until they meet again. The node where they meet is the start of the cycle.

2.

To solve this problem, we can use Floyd's Tortoise and Hare (Cycle Detection) algorithm. The algorithm works by having two pointers, one moving at twice the speed of the other. If there is a cycle in the linked list, the two pointers will eventually meet at some point within the cycle.

Here's how we can find the node where the cycle begins:

1. Initialize two pointers, slow and fast, both starting at the head of the linked list.

2. Move slow one step at a time and fast two steps at a time.

3. If there is a cycle, the two pointers will meet at some point within the cycle.

4. Reset one of the pointers to the head of the linked list and move both pointers one step at a time until they meet again. The node where they meet is the start of the cycle.

Code in C

*#include <stdbool.h>*

*struct ListNode {*

*int val;*

*struct ListNode \*next;*

*};*

*struct ListNode \*detectCycle(struct ListNode \*head) {*

*if (head == NULL || head->next == NULL) {*

*return NULL;*

*}*

*struct ListNode \*slow = head;*

*struct ListNode \*fast = head;*

*bool hasCycle = false;*

*while (fast != NULL && fast->next != NULL) {*

*slow = slow->next;*

*fast = fast->next->next;*

*if (slow == fast) {*

*hasCycle = true;*

*break;*

*}*

*}*

*if (!hasCycle) {*

*return NULL;*

*}*

*slow = head;*

*while (slow != fast) {*

*slow = slow->next;*

*fast = fast->next;*

*}*

*return slow;*

*}*

In this code:

- The detectCycle function uses Floyd's Tortoise and Hare algorithm to detect if there is a cycle in the linked list. It returns the node where the cycle begins if a cycle is found, and NULL otherwise.

- The function initializes two pointers, slow and fast, both starting at the head of the linked list. It moves slow one step at a time and fast two steps at a time. If there is a cycle, the two pointers will eventually meet at some point within the cycle.

- Once a cycle is detected, one of the pointers is reset to the head of the linked list, and both pointers are moved one step at a time until they meet again. The node where they meet is returned as the start of the cycle.

- If no cycle is found, the function returns NULL.

3.

To write a function that reverses a linked list, you can follow these steps in Python

python

*class Node:*

*def \_\_init\_\_(self, value):*

*self.value = value*

*self.next = None*

*def reverse\_linked\_list(head):*

*prev = None*

*current = head*

*while current is not None:*

*next\_node = current.next*

*current.next = prev*

*prev = current*

*current = next\_node*

*return prev*

*# Example usage*

*def print\_linked\_list(head):*

*current = head*

*while current is not None:*

*print(current.value, end=" ")*

*current = current.next*

*print()*

*# Create a sample linked list*

*node1 = Node(1)*

*node2 = Node(2)*

*node3 = Node(3)*

*node1.next = node2*

*node2.next = node3*

*print("Original linked list:")*

*print\_linked\_list(node1)*

*reversed\_head = reverse\_linked\_list(node1)*

*print("Reversed linked list:")*

*print\_linked\_list(reversed\_head*)

In this code

1. We define a Node class to represent nodes in the linked list.

2. We define the reverse\_linked\_list function that takes the head of the linked list as input and returns the head of the reversed list.

3. We iterate through the linked list, reversing the pointers of each node to point to the previous node.

4. Finally, we return the new head of the reversed linked list.