Length of cycle in Linked List - GeeksforGeeks

Source: https://www.geeksforgeeks.org/find-length-of-loop-in-linked-list/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Length of cycle in Linked List Last Updated : 28 Aug, 2025 Given the head of a singly linked list, determine the length of the cycle (loop) if one exists. A cycle occurs when a node's next pointer points to a previously visited node in the list. If no cycle is present, return 0. Examples: Input: Output: 3 Explanation : There exists a loop in the linked list and the length of the loop is 3. Input: Output: 0 Explanation : There is no loop present in the Linked List. Try it on GfG Practice Table of Content [Naive Approach] Using Set - O(n) Time and O(n) Space [Expected Approach] Using Floyd’s Cycle Detection Algorithm - O(n) Time and O(1) Space [Naive Approach] Using Set - O(n) Time and O(n) Space The idea is to maintain a set to keep track of visited nodes so far. If the node is not present in set we will insert and move to another node, else we will maintain a counter from that node and will start traversing until we reach to it again by incrementing the counter variable every time. C++ #include <iostream> #include <unordered_set> using namespace std ; // Node structure class Node { public : int data ; Node * next ; Node (int x) { data = x ; next = nullptr ; } }; int lengthOfLoop (Node * head) { unordered_set < Node *> visited ; Node * current = head ; int count = 0 ; while (current != nullptr) { // if the node is already visited, // it means there is a loop if (visited . find (current) != visited . end ()) { Node * startOfLoop = current ; do { count ++ ; current = current -> next ; } while (current != startOfLoop); return count ; } // mark the current node as visited visited . insert (current); // move to the next node current = current -> next ; } return 0 ; } int main () { Node * head = new Node (25); head -> next = new Node (14); head -> next -> next = new Node (19); head -> next -> next -> next = new Node (33); head -> next -> next -> next -> next = new Node (10); head -> next -> next -> next -> next -> next = head -> next -> next ; cout << lengthOfLoop (head) << endl ; return 0 ; } Java import java.util.HashSet ; // Node structure class Node { int data ; Node next ; Node (int x) { data = x ; next = null ; } } class GfG { static int lengthOfLoop (Node head) { HashSet < Node > visited = new HashSet <> (); Node current = head ; int count = 0 ; while (current != null) { // if the node is already visited, // it means there is a loop if (visited . contains (current)) { Node startOfLoop = current ; do { count ++ ; current = current . next ; } while (current != startOfLoop); return count ; } // mark the current node as visited visited . add (current); // move to the next node current = current . next ; } return 0 ; } public static void main (String [] args) { Node head = new Node (25); head . next = new Node (14); head . next . next = new Node (19); head . next . next . next = new Node (33); head . next . next . next . next = new Node (10); head . next . next . next . next . next = head . next . next ; System . out . println (lengthOfLoop (head)); } } Python # Node structure class Node : def __init__ (self , x): self . data = x self . next = None def lengthOfLoop (head): visited = set () current = head count = 0 while current is not None : # if the node is already visited, # it means there is a loop if current in visited : startOfLoop = current while True : count += 1 current = current . next if current == startOfLoop : break return count # mark the current node as visited visited . add (current) # move to the next node current = current . next return 0 if __name__ == "__main__" : head = Node (25) head . next = Node (14) head . next . next = Node (19) head . next . next . next = Node (33) head . next . next . next . next = Node (10) head . next . next . next . next . next = head . next . next print (lengthOfLoop (head)) C# using System ; using System.Collections.Generic ; // Node structure class Node { public int data ; public Node next ; public Node (int x) { data = x ; next = null ; } } class GfG { static int lengthOfLoop (Node head) { HashSet < Node > visited = new HashSet < Node > (); Node current = head ; int count = 0 ; while (current != null) { // if the node is already visited, // it means there is a loop if (visited . Contains (current)) { Node startOfLoop = current ; do { count ++ ; current = current

. next ; } while (current != startOfLoop); return count ; } // mark the current node as visited visited . Add (current); // move to the next node current = current . next ; } return 0 ; } public static void Main (string [] args) { Node head = new Node (25); head . next = new Node (14); head . next . next = new Node (19); head . next . next . next = new Node (33); head . next . next . next . next = new Node (10); head . next . next . next . next . next = head . next . next ; Console . WriteLine (lengthOfLoop (head)); } } JavaScript // Node structure class Node { constructor (x) { this . data = x ; this . next = null ; } } function lengthOfLoop (head) { const visited = new Set (); let current = head ; let count = 0 ; while (current !== null) { // if the node is already visited, // it means there is a loop if (visited . has (current)) { const startOfLoop = current ; do { count ++ ; current = current . next ; } while (current !== startOfLoop); return count ; } // mark the current node as visited visited . add (current); // move to the next node current = current . next ; } return 0 ; } // Driver Code let head = new Node (25); head . next = new Node (14); head . next . next = new Node (19); head . next . next . next = new Node (33); head . next . next . next . next = new Node (10); head . next . next . next . next . next = head . next . next ; console . log (lengthOfLoop (head)); Output 3 [Expected Approach] Using Floyd’s Cycle Detection Algorithm - O(n) Time and O(1) Space The idea is to use Floyd’s Cycle detection algorithm for detecting the common point in the loop. Using fast and slow pointer Step by Step Approach: Use a fast and slow pointer pointing to head of linked list. move fast pointer to fast->next->next and slow pointer to slow->next. If a common meeting point exists between the slow and fast pointers, it confirms the presence of a loop. Once the loop is detected, we start counting the number of nodes in the loop by initializing a counter and traversing the loop starting from the meeting point. If no meeting point is found, it means there is no loop, so we return 0. C++ #include <iostream> using namespace std ; // Node structure class Node { public : int data ; Node * next ; Node (int x) { data = x ; next = nullptr ; } }; // Returns count of nodes present in loop. int countNodes (Node * node) { int res = 1 ; Node * curr = node ; while (curr -> next != node) { res ++ ; curr = curr -> next ; } return res ; } // Detects and Counts nodes in loop int lengthOfLoop (Node * head) { Node * slow = head , * fast = head ; while (slow != nullptr && fast != nullptr && fast -> next != nullptr) { slow = slow -> next ; fast = fast -> next -> next ; // If slow and fast meet at // some point then there is a loop if (slow == fast) return countNodes (slow); } return 0 ; } int main () { Node * head = new Node (25); head -> next = new Node (14); head -> next -> next = new Node (19); head -> next -> next -> next = new Node (33); head -> next -> next -> next -> next = new Node (10); head -> next -> next -> next -> next -> next = head -> next -> next ; cout << lengthOfLoop (head) << endl ; return 0 ; } Java // Node structure class Node { int data ; Node next ; Node (int x) { data = x ; next = null ; } } class GfG { // Returns count of nodes present in loop. static int countNodes (Node node) { int res = 1 ; Node curr = node ; while (curr . next != node) { res ++ ; curr = curr . next ; } return res ; } // Detects and Counts nodes in loop static int lengthOfLoop (Node head) { Node slow = head , fast = head ; while (slow != null && fast != null && fast . next != null) { slow = slow . next ; fast = fast . next . next ; // if slow and fast meet at // some point then there is a loop if (slow == fast) return countNodes (slow); } return 0 ; } public static void main (String [] args) { Node head = new Node (25); head . next = new Node (14); head . next . next = new Node (19); head . next . next . next = new Node (33); head . next . next . next . next = new Node (10); head . next . next . next . next . next = head . next . next ; System . out . println (lengthOfLoop (head)); } } Python # Node structure class Node : def __init__ (self , x): self . data = x self . next = None # Returns count of nodes present in loop. def countNodes (node): res = 1 curr = node while curr . next != node : res += 1 curr = curr . next return res # Detects and Counts nodes in loop def lengthOfLoop (head): slow = head fast = head while slow is not None and fast is not None \ and fast . next is not None : slow = slow . next fast = fast . next . next # if slow and fast meet at # some point then there is a loop if slow == fast : return countNodes (slow) return 0 if __name__ == "__main__" : head = Node (25) head . next = Node (14) head . next . next = Node (19) head . next . next . next = Node (33) head . next . next . next . next = Node (10) head . next . next . next . next . next = head . next . next print (lengthOfLoop (head)) C# using System ; // Node structure class Node { public int data ; public Node next ; public Node (int x) { data = x ; next = null ; } } class GfG { // Returns count of nodes present in loop. public static int countNodes (Node node) { int res = 1 ; Node curr = node ; while (curr . next != node) { res ++ ; curr = curr . next ; } return res ; } // Detects and Counts nodes in loop public static int lengthOfLoop (Node head) { Node slow = head , fast = head ; while (slow != null && fast != null && fast . next != null) { slow = slow . next ; fast = fast . next . next ; // if slow and fast meet at // some point then there is a loop if (slow == fast) return countNodes (slow); } return 0 ; } public static void Main (string [] args) { Node head = new Node (25); head . next = new Node (14); head . next . next = new Node (19); head . next . next . next = new Node (33); head . next . next . next . next = new Node (10); head . next . next . next . next . next =

head . next . next ; Console . WriteLine (lengthOfLoop (head)); } } JavaScript // Node structure class Node { constructor (x) { this . data = x ; this . next = null ; } } // Returns count of nodes present in loop. function countNodes (node) { let res = 1 ; let curr = node ; while (curr . next !== node) { res ++ ; curr = curr . next ; } return res ; } // Detects and Counts nodes in loop function lengthOfLoop (head) { let slow = head , fast = head ; while (slow !== null && fast !== null && fast . next !== null) { slow = slow . next ; fast = fast . next . next ; // if slow and fast meet at // some point then there is a loop if (slow === fast) return countNodes (slow); } return 0 ; } // Driver Code let head = new Node (25); head . next = new Node (14); head . next . next = new Node (19); head . next . next . next = new Node (33); head . next . next . next . next = new Node (10); head . next . next . next . next . next = head . next . next ; console . log (lengthOfLoop (head)); Output 3 Comment Article Tags: Article Tags: Linked List DSA Adobe Qualcomm