Detect Cycle in Linked List - GeeksforGeeks

Source: https://www.geeksforgeeks.org/detect-loop-in-a-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 Detect Cycle in Linked List Last Updated : 28 Aug, 2025 Given the head of a singly linked list, determine whether the list contains a cycle. A cycle exists if, while traversing the list through next pointers, you encounter a node that has already been visited instead of eventually reaching nullptr. Examples: Input : head: 1 -> 3 -> 4 -> 3 Output : true Explanation: The last node of the linked list does not point to NULL; instead, it points to an earlier node in the list, creating a cycle. Input: head: 1 -> 8 -> 3 -> 4 -> NULL Output : false Explanation: The last node of the linked list points to NULL, indicating the end of the list. Try it on GfG Practice Table of Content [Naive Approach] Using HashSet - O(n) Time and O(n) Space [Expected Approach] Using Floyd's Cycle-Finding Algorithm [Naive Approach] Using HashSet - O(n) Time and O(n) Space The idea is to insert the nodes in the Hashset while traversing and whenever a node is encountered that is already present in the hashset (which indicates there's a cycle (loop) in the list) then return true. If the node is NULL, represents the end of Linked List, return false as there is no loop. C++ #include <iostream> #include <unordered_set> using namespace std ; class Node { public : int data ; Node * next ; Node (int x) { this -> data = x ; this -> next = nullptr ; } }; bool detectLoop (Node * head) { unordered_set < Node *> st ; while (head != nullptr) { // if this node is already present // in hashmap it means there is a cycle if (st . find (head) != st . end ()) return true ; // if we are seeing the node for // the first time, insert it in hash st . insert (head); head = head -> next ; } return false ; } int main () { Node * head = new Node (1); head -> next = new Node (3); head -> next -> next = new Node (4); head -> next -> next -> next = head -> next ; if (detectLoop (head)) cout << "true" ; else cout << "false" ; return 0 ; } Java import java.util.HashSet ; class Node { int data ; Node next ; Node (int x) { this . data = x ; this . next = null ; } } class GfG { static boolean detectLoop (Node head) { HashSet < Node > st = new HashSet <> (); while (head != null) { // if this node is already present // in hashmap it means there is a cycle if (st . contains (head)) return true ; // if we are seeing the node for // the first time, insert it in hash st . add (head); head = head . next ; } return false ; } public static void main (String [] args) { Node head = new Node (1); head . next = new Node (3); head . next . next = new Node (4); head . next . next . next = head . next ; if (detectLoop (head)) System . out . println ("true"); else System . out . println ("false"); } } Python class Node : def __init__ (self , x): self . data = x self . next = None def detectLoop (head): st = set () while head is not None : # if this node is already present # in hashmap it means there is a cycle if head in st : return True # if we are seeing the node for # the first time, insert it in hash st . add (head) head = head . next return False if __name__ == "__main__" : head = Node (1) head . next = Node (3) head . next . next = Node (4) head . next . next . next = head . next if detectLoop (head): print ("true") else : print ("false") C# using System ; using System.Collections.Generic ; class Node { public int data ; public Node next ; public Node (int x) { this . data = x ; this . next = null ; } } class GfG { static bool detectLoop (Node head) { HashSet < Node > st = new HashSet < Node > (); while (head != null) { // if this node is already present // in hashmap it means there is a cycle if (st . Contains (head)) return true ; // if we are seeing the node for // the first time, insert it in hash st . Add (head); head = head . next ; } return false ; } static void Main () { Node head = new Node (1); head . next = new Node (3); head . next . next = new Node (4); head . next . next . next = head . next ; if (detectLoop (head)) Console . WriteLine ("true"); else Console . WriteLine ("false"); } } JavaScript class Node { constructor (x) { this . data = x ; this . next = null ; } } function detectLoop (head) { const st = new Set (); while (head !== null) { // if this node is already present // in hashmap it means there is a cycle if (st . has (head)) return true ; // if we are seeing the node for // the

first time, insert it in hash st . add (head); head = head . next ; } return false ; } // Driver Code let head = new Node (1); head . next = new Node (3); head . next . next = new Node (4); head . next . next . next = head . next ; if (detectLoop (head)) console . log ("true"); else console . log ("false"); Output true [Expected Approach] Using Floyd's Cycle-Finding Algorithm This idea is to use Floyd's Cycle-Finding Algorithm to find a loop in a linked list. It uses two pointers slow and fast, fast pointer move two steps ahead and slow will move one step ahead at a time. Algorithm: Traverse linked list using two pointers. Move one pointer(slow) by one step ahead and another pointer(fast) by two steps ahead. If these pointers meet at the same node then there is a loop. If pointers do not meet then the linked list doesn't have a loop. Below is the illustration of above algorithm: Why meeting is guaranteed if a cycle exists ? Let: m = number of nodes before the cycle starts and c = length of the cycle (number of nodes in the loop) When both pointers enter the cycle: At that point, the difference in steps between fast and slow increases by 1 each turn (because fast moves 1 step more than slow each iteration). This difference is taken modulo c because positions wrap around inside the cycle. Mathematically: If d is the difference in positions (mod c), then each iteration: $d \equiv d + 1 \pmod{c}$ Since 1 and c are coprime, adding 1 repeatedly cycles through all residues 0, 1, 2, …, c-1. Thus, eventually $d \equiv 0 \rightarrow$ meaning both pointers are at the same node (meeting point). For more details about the working & proof of this algorithm, Please refer to this article, How does Floyd’s Algorithm works . C++ #include <iostream> using namespace std ; class Node { public : int data ; Node * next ; Node (int x) { this -> data = x ; this -> next = nullptr ; } }; bool detectLoop (Node * head) { // Fast and slow pointers // initially points to the head Node * slow = head , * fast = head ; // Loop that runs while fast and slow pointer are not // nullptr and not equal while (slow && fast && fast -> next) { slow = slow -> next ; fast = fast -> next -> next ; // If fast and slow pointer points to the same node, // then the cycle is detected if (slow == fast) { return true ; } } return false ; } int main () { Node * head = new Node (1); head -> next = new Node (3); head -> next -> next = new Node (4); head -> next -> next -> next = head -> next ; if (detectLoop (head)) cout << "true" ; else cout << "false" ; return 0 ; } C #include <stdbool.h> #include <stdio.h> #include <stdlib.h> struct Node { int data ; struct Node * next ; }; struct Node * createNode (int new_data) { struct Node * new_node = (struct Node *) malloc (sizeof (struct Node)); new_node -> data = new_data ; new_node -> next = NULL ; return new_node ; } int detectLoop (struct Node * head) { // Fast and slow pointers initially points to the head struct Node * slow = head , * fast = head ; // Loop that runs while fast and slow pointer are not // nullptr and not equal while (slow && fast && fast -> next) { slow = slow -> next ; fast = fast -> next -> next ; // If fast and slow pointer points to the same node, // then the cycle is detected if (slow == fast) { return true ; } } return false ; } int main () { struct Node * head = createNode (1); head -> next = createNode (3); head -> next -> next = createNode (4); head -> next -> next -> next = head -> next ; if (detectLoop (head)) printf ("true"); else printf ("false"); return 0 ; } Java class Node { int data ; Node next ; public Node (int x) { this . data = x ; this . next = null ; } } class GfG { static boolean detectLoop (Node head) { // Fast and slow pointers initially points to the head Node slow = head , fast = head ; // Loop that runs while fast and slow pointer are not // null and not equal while (slow != null && fast != null && fast . next != null) { slow = slow . next ; fast = fast . next . next ; // If fast and slow pointer points to the same node, // then the cycle is detected if (slow == fast) { return true ; } } return false ; } public static void main (String [] args) { Node head = new Node (1); head . next = new Node (3); head . next . next = new Node (4); head . next . next . next = head . next ; if (detectLoop (head)) System . out . println ("true"); else System . out . println ("false"); } } Python class Node : def __init__ (self , x): self . data = x self . next = None def detectLoop (head): # fast and slow pointers initially points to the head slow = head fast = head # loop that runs while fast and slow pointer are not # None and not equal while slow and fast and fast . next : slow = slow . next fast = fast . next . next # if fast and slow pointer points to the same node, # then the cycle is detected if slow == fast : return True return False if __name__ == "__main__" : head = Node (1) head . next = Node (3) head . next . next = Node (4) head . next . next . next = head . next if detectLoop (head): print ("true") else : print ("false") C# using System ; class Node { public int data ; public Node next ; public Node (int x) { this . data = x ; this . next = null ; } } class GfG { static bool detectLoop (Node head) { // fast and slow pointers initially points to the head Node slow = head , fast = head ; // loop that runs while fast and slow pointer are not // null and not equal while (slow != null && fast != null && fast . next != null) { slow = slow . next ; fast = fast . next . next ; // if fast and slow pointer points to the same node, // then the cycle is detected if (slow == fast) { return true ; } } return false ; } static void Main () { Node head = new Node (1); head . next = new Node (3); head . next . next = new Node (4); head . next . next . next = head . next ; if (detectLoop (head)) Console . WriteLine ("true"); else Console . WriteLine ("false"); } } JavaScript class Node { constructor (x) { this . data = x ; this . next = null ; } } function detectLoop (

head) { // fast and slow pointers // initially points to the head let slow = head , fast = head ; // Loop that runs while fast and slow pointer are not // null and not equal while (slow && fast && fast . next) { slow = slow . next ; fast = fast . next . next ; // If fast and slow pointer points to the same node, // then the cycle is detected if (slow === fast) { return true ; } } return false ; } // Driver Code let head = new Node (1); head . next = new Node (3); head . next . next = new Node (4); head . next . next . next = head . next ; if (detectLoop (head)) console . log ("true"); else console . log ("false"); Output true Time Complexity: O(n) because in the worst case, both pointers traverse at most n nodes before meeting or terminating, where n is the total number of nodes in the linked list. Auxiliary Space: O(1) Related Articles: Remove loop in Linked List Find Starting node of a loop in Linked List Comment Article Tags: Article Tags: Linked List DSA Linked Lists loop Amazon Samsung Accolite MAQ Software Tortoise-Hare-Approach + 5 More