Clone a linked list with next and random pointer in O(1) space - GeeksforGeeks

Source: https://www.geeksforgeeks.org/clone-linked-list-next-random-pointer-o1-space/

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 Clone a linked list with next and random pointer in O(1) space Last Updated : 11 Feb, 2025 Given a linked list of size N where each node has two links: next pointer pointing to the next node and random pointer to any random node in the list. The task is to create a clone of this linked list in O(1) space, i.e., without any extra space. Examples: Input: Head of the below linked list Output: A new linked list identical to the original list. [Expected Approach] By Inserting Nodes In-place – O(3n) Time and O(1) Space The idea is to create duplicate of a node and instead of storing in a separate hash table, we can insert it in between the original node and the next node. Now, we will have new nodes at alternate positions. Now for a node X its duplicate will be X->next and the random pointer of the duplicate should point to X->random->next (as that is the duplicate of X->random). So, iterate over the entire linked list to update the random pointer of all the cloned nodes and then iterate again to separate the original linked list and the cloned linked list. Follow the steps mentioned below to implement the idea: Create the copy of node 1 and insert it between node 1 and node 2 in the original Linked List, create the copy of node 2 and insert it between 2 nd and 3 rd node and so on. Add the copy of N after the N th node Connect the clone node by updating the random pointers. Separate the cloned linked list from the original list by updating the next pointers. Below is the implementation if the above approach: C++ // C++ code to Clone a linked list with next and random // pointer by Inserting Nodes In-place #include <bits/stdc++.h> using namespace std ; struct Node { int data ; Node * next , * random ; Node (int x) { data = x ; next = random = NULL ; } }; Node * cloneLinkedList (Node * head) { if (head == NULL) { return NULL ; } // Create new nodes and insert them next to // the original nodes Node * curr = head ; while (curr != NULL) { Node * newNode = new Node (curr -> data); newNode -> next = curr -> next ; curr -> next = newNode ; curr = newNode -> next ; } // Set the random pointers of the new nodes curr = head ; while (curr != NULL) { if (curr -> random != NULL) curr -> next -> random = curr -> random -> next ; curr = curr -> next -> next ; } // Separate the new nodes from the original nodes curr = head ; Node * clonedHead = head -> next ; Node * clone = clonedHead ; while (clone -> next != NULL) { // Update the next nodes of original node // and cloned node curr -> next = curr -> next -> next ; clone -> next = clone -> next -> next ; // Move pointers of original as well as // cloned linked list to their next nodes curr = curr -> next ; clone = clone -> next ; } curr -> next = NULL ; clone -> next = NULL ; return clonedHead ; } // Function to print the linked list void printList (Node * head) { while (head != NULL) { cout << head -> data << "(" ; if (head -> random) cout << head -> random -> data << ")" ; else cout << "null" << ")" ; if (head -> next != NULL) cout << " -> " ; head = head -> next ; } cout << endl ; } int main () { // Creating a linked list with random pointer Node * head = new Node (1); head -> next = new Node (2); head -> next -> next = new Node (3); head -> next -> next -> next = new Node (4); head -> next -> next -> next -> next = new Node (5); head -> random = head -> next -> next ; head -> next -> random = head ; head -> next -> next -> random = head -> next -> next -> next -> next ; head -> next -> next -> next -> random = head -> next -> next ; head -> next -> next -> next -> next -> random = head -> next ; // Print the original list cout << "Original linked list: \n " ; printList (head); // Function call Node * clonedList = cloneLinkedList (head); cout << "Cloned linked list: \n " ; printList (clonedList); return 0 ; } Java // Java code to Clone a linked list with next and random // pointer by Inserting Nodes In-place class Node { int data ; Node next , random ; Node (int x) { data = x ; next = random = null ; } } class GfG { // Function to

clone the linked list static Node cloneLinkedList (Node head) { if (head == null) { return null ; } // Create new nodes and insert them next to the original nodes Node curr = head ; while (curr != null) { Node newNode = new Node (curr . data); newNode . next = curr . next ; curr . next = newNode ; curr = newNode . next ; } // Set the random pointers of the new nodes curr = head ; while (curr != null) { if (curr . random != null) { curr . next . random = curr . random . next ; } curr = curr . next . next ; } // Separate the new nodes from the original nodes curr = head ; Node clonedHead = head . next ; Node clone = clonedHead ; while (clone . next != null) { // Update the next nodes of original node // and cloned node curr . next = curr . next . next ; clone . next = clone . next . next ; // Move pointers of original and cloned // linked list to their next nodes curr = curr . next ; clone = clone . next ; } curr . next = null ; clone . next = null ; return clonedHead ; } // Function to print the linked list public static void printList (Node head) { while (head != null) { System . out . print (head . data + "("); if (head . random != null) { System . out . print (head . random . data); } else { System . out . print ("null"); } System . out . print (")"); if (head . next != null) { System . out . print (" -> "); } head = head . next ; } System . out . println (); } public static void main (String [] args) { // Creating a linked list with random pointer Node head = new Node (1); head . next = new Node (2); head . next . next = new Node (3); head . next . next . next = new Node (4); head . next . next . next . next = new Node (5); head . random = head . next . next ; head . next . random = head ; head . next . next . random = head . next . next . next . next ; head . next . next . next . random = head . next . next ; head . next . next . next . next . random = head . next ; // Print the original list System . out . println ("Original linked list:"); printList (head); // Function call Node clonedList = cloneLinkedList (head); System . out . println ("Cloned linked list:"); printList (clonedList); } } Python # Python code to Clone a linked list with next and random # pointer by Inserting Nodes In-place class Node : def __init__ (self , x): self . data = x self . next = None self . random = None # Function to clone the linked list def clone_linked_list (head): if head is None : return None # Create new nodes and insert them next to # the original nodes curr = head while curr is not None : new_node = Node (curr . data) new_node . next = curr . next curr . next = new_node curr = new_node . next # Set the random pointers of the new nodes curr = head while curr is not None : if curr . random is not None : curr . next . random = curr . random . next curr = curr . next . next # Separate the new nodes from the original nodes curr = head cloned_head = head . next clone = cloned_head while clone . next is not None : # Update the next nodes of original node # and cloned node curr . next = curr . next . next clone . next = clone . next . next # Move pointers of original as well as # cloned linked list to their next nodes curr = curr . next clone = clone . next curr . next = None clone . next = None return cloned_head # Function to print the linked list def print_list (head): while head is not None : print (f " { head . data } (" , end = "") if head . random : print (f " { head . random . data })" , end = "") else : print ("null)" , end = "") if head . next is not None : print (" -> " , end = "") head = head . next print () if __name__ == "__main__" : # Creating a linked list with random pointer head = Node (1) head . next = Node (2) head . next . next = Node (3) head . next . next . next = Node (4) head . next . next . next . next = Node (5) head . random = head . next . next head . next . random = head head . next . next . random = head . next . next . next . next head . next . next . next . random = head . next . next head . next . next . next . next . random = head . next # Print the original list print ("Original linked list:") print_list (head) # Function call cloned_list = clone_linked_list (head) print ("Cloned linked list:") print_list (cloned_list) C# // C# code to Clone a linked list with next and random // pointer by Inserting Nodes In-place using System ; using System.Collections.Generic ; public class Node { public int Data ; public Node next , Random ; public Node (int x) { Data = x ; next = Random = null ; } } class GfG { static Node CloneLinkedList (Node head) { if (head == null) return null ; // Create new nodes and insert them next to // the original nodes Node curr = head ; while (curr != null) { Node newNode = new Node (curr . Data); newNode . next = curr . next ; curr . next = newNode ; curr = newNode . next ; } // Set the random pointers of the new nodes curr = head ; while (curr != null) { if (curr . Random != null) curr . next . Random = curr . Random . next ; curr = curr . next . next ; } // Separate the new nodes from the original nodes curr = head ; Node clonedHead = head . next ; Node clone = clonedHead ; while (clone . next != null) { // Update the next nodes of original node // and cloned node curr . next = curr . next . next ; clone . next = clone . next . next ; // Move pointers of original as well as // cloned linked list to their next nodes curr = curr . next ; clone = clone . next ; } curr . next = null ; clone . next = null ; return clonedHead ; } // Function to print the linked list static void PrintList (Node head) { while (head != null) { Console . Write (head . Data + "("); if (head . Random != null) Console . Write (head . Random . Data + ")"); else Console . Write ("null)"); if (head . next != null) Console . Write (" -> "); head = head . next ; } Console . WriteLine (); } public static void Main () { // Creating a linked list with random pointer Node head = new Node (1); head . next = new Node (2); head . next . next = new Node (3); head .

next . next . next = new Node (4); head . next . next . next . next = new Node (5); head . Random = head . next . next ; head . next . Random = head ; head . next . next . Random = head . next . next . next . next ; head . next . next . next . Random = head . next . next ; head . next . next . next . next . Random = head . next ; // Print the original list Console . WriteLine ("Original linked list:"); PrintList (head); Node clonedList = CloneLinkedList (head); Console . WriteLine ("Cloned linked list:"); PrintList (clonedList); } } JavaScript // JavaScript code to Clone a linked list with next and random // pointer by Inserting Nodes In-place class Node { constructor (data) { this . data = data ; this . next = null ; this . random = null ; } } function cloneLinkedList (head) { if (head === null) { return null ; } // Create new nodes and insert them next to the // original nodes let curr = head ; while (curr !== null) { let newNode = new Node (curr . data); newNode . next = curr . next ; curr . next = newNode ; curr = newNode . next ; } // Set the random pointers of the new nodes curr = head ; while (curr !== null) { if (curr . random !== null) { curr . next . random = curr . random . next ; } curr = curr . next . next ; } // Separate the new nodes from the original nodes curr = head ; let clonedHead = head . next ; let clone = clonedHead ; while (clone . next !== null) { // Update the next nodes of original node and cloned node curr . next = curr . next . next ; clone . next = clone . next . next ; // Move pointers of original as well as cloned // linked list to their next nodes curr = curr . next ; clone = clone . next ; } curr . next = null ; clone . next = null ; return clonedHead ; } // Function to print the linked list function printList (head) { let result = "" ; while (head !== null) { result += head . data + "(" ; result += head . random ? head . random . data : "null" ; result += ")" ; if (head . next !== null) { result += " -> " ; } head = head . next ; } console . log (result); } // Creating a linked list with random pointer let head = new Node (1); head . next = new Node (2); head . next . next = new Node (3); head . next . next . next = new Node (4); head . next . next . next . next = new Node (5); head . random = head . next . next ; head . next . random = head ; head . next . next . random = head . next . next . next . next ; head . next . next . next . random = head . next . next ; head . next . next . next . next . random = head . next ; // Print the original list console . log ("Original linked list:"); printList (head); let clonedList = cloneLinkedList (head); console . log ("Cloned linked list:"); printList (clonedList); Output Original linked list: 1(3) -> 2(1) -> 3(5) -> 4(3) -> 5(2) Cloned linked list: 1(3) -> 2(1) -> 3(5) -> 4(3) -> 5(2) Time Complexity: O(3n), because we are traversing the linked list three times. Auxiliary Space: O(1), as we are storing all the cloned nodes in the original linked list itself, no extra space is required. Comment Article Tags: Article Tags: Linked List DSA