Identical Linked Lists

Two Linked Lists are identical when they have same data and arrangement of data is also same. For example Linked lists a (1->2->3) and b(1->2->3) are identical. . Write a function to check if the given two linked lists are identical.

Method 1 (Iterative)

To identify if two lists are identical, we need to traverse both lists simultaneously, and while traversing we need to compare data.

C

```c
// An iterative C program to check if two linked lists are
// identical or not
#include<stdio.h>
#include<stdlib.h>

/* Structure for a linked list node */
struct node
{
    int data;
    struct node *next;
};

/* Returns true if linked lists a and b are identical,
   otherwise false */
bool areIdentical(struct node *a, struct node *b)
{
    while (a != NULL && b != NULL)
    {
        if (a->data != b->data)
            return false;

        /* If we reach here, then a and b are not NULL and
           their data is same, so move to next nodes in both
           lists */
        a = a->next;
        b = b->next;
    }

    // If linked lists are identical, then 'a' and 'b' must
    // be NULL at this point.
    return (a == NULL && b == NULL);
}
```

```
/* UTILITY FUNCTIONS TO TEST fun1() and fun2() */
/* Given a reference (pointer to pointer) to the head
  of a list and an int, push a new node on the front
  of the list. */
void push(struct node** head_ref, int new_data)
{
    /* allocate node */
    struct node* new_node =
        (struct node*) malloc(sizeof(struct node));

    /* put in the data  */
    new_node->data  = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}

/* Druver program to test above function */
int main()
{
    /* The constructed linked lists are :
     a: 3->2->1
     b: 3->2->1 */
    struct node *a = NULL;
    struct node *b = NULL;
    push(&a, 1);
    push(&a, 2);
    push(&a, 3);
    push(&b, 1);
    push(&b, 2);
    push(&b, 3);

    areIdentical(a, b)? printf("Identical"):
                        printf("Not identical");

    return 0;
}
```

Run on IDE

Java

```
// An iterative Java program to check if two linked lists
// are identical or not
class LinkedList
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d) { data = d; next = null; }
    }

    /* Returns true if linked lists a and b are identical,
       otherwise false */
    boolean areIdentical(LinkedList listb)
    {
        Node a = this.head, b = listb.head;
        while (a != null && b != null)
        {
            if (a.data != b.data)
                return false;

            /* If we reach here, then a and b are not null
               and their data is same, so move to next nodes
               in both lists */
            a = a.next;
            b = b.next;
        }

        // If linked lists are identical, then 'a' and 'b' must
        // be null at this point.
        return (a == null && b == null);
    }

    /* UTILITY FUNCTIONS TO TEST fun1() and fun2() */
    /*  Given a reference (pointer to pointer) to the head
        of a list and an int, push a new node on the front
        of the list. */

    void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                  Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
    }


    /* Drier program to test above functions */
    public static void main(String args[])
    {
        LinkedList llist1 = new LinkedList();
        LinkedList llist2 = new LinkedList();

        /* The constructed linked lists are :
           llist1: 3->2->1
           llist2: 3->2->1 */

        llist1.push(1);
        llist1.push(2);
        llist1.push(3);

        llist2.push(1);
        llist2.push(2);
        llist2.push(3);

        if (llist1.areIdentical(llist2) == true)
            System.out.println("Identical ");
        else
            System.out.println("Not identical ");

    }
} /* This code is contributed by Rajat Mishra */
```

Run on IDE

Output:

```
Identical
```

Method 2 (Recursive)

Recursive solution code is much cleaner than the iterative code. You probably wouldn't want to use the recursive version for production code however, because it will use stack space which is proportional to the length of the lists

C

```
// A recursive C function to check if two linked
// lists are identical or not
bool areIdentical(struct node *a, struct node *b)
{
    // If both lists are empty
    if (a == NULL && b == NULL)
       return true;

    // If both lists are not empty, then data of
    // current nodes must match, and same should
    // be recursively true for rest of the nodes.
    if (a != NULL && b != NULL)
       return (a->data == b->data) &&
              areIdentical(a->next, b->next);

    // If we reach here, then one of ths lists
    // is empty and other is not
    return false;
}
```

Run on IDE

Java

```
// A recursive Java method to check if two linked
// lists are identical or not
boolean areIdenticalRecur(Node a, Node b)
{
    // If both lists are empty
    if (a == null && b == null)
        return true;

    // If both lists are not empty, then data of
    // current nodes must match, and same should
    // be recursively true for rest of the nodes.
    if (a != null && b != null)
        return (a.data == b.data) &&
               areIdenticalRecur(a.next, b.next);

    // If we reach here, then one of ths lists
    // is empty and other is not
    return false;
}

/* Returns true if linked lists a and b are identical,
   otherwise false */
boolean areIdentical(LinkedList listb)
{
    return areIdenticalRecur(this.head, listb.head);
}
```

Run on IDE

Time Complexity: O(n) for both iterative and recursive versions. n is the length of the smaller list among a and b.

Please write comments if you find the above codes/algorithms incorrect, or find better ways to solve the same problem.

42 Comments Category: Linked Lists

Related Posts:

- Merge two sorted linked lists such that merged list is in reverse order
- Compare two strings represented as linked lists
- Rearrange a given linked list in-place.
- Sort a linked list that is sorted alternating ascending and descending orders?
- Select a Random Node from a Singly Linked List
- Merge Sort for Doubly Linked List
- Point to next higher value node in a linked list with an arbitrary pointer

- Swap nodes in a linked list without swapping data

(Login to Rate and Mark)

1.4 Average Difficulty : **1.4/5.0**
Based on **5** vote(s)

☐ Add to TODO List

☐ Mark as DONE

Like Share 9 people like this.

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.