Union and Intersection of two Linked Lists

Given two Linked Lists, create union and intersection lists that contain union and intersection of the elements present in the given lists. Order of elments in output lists doesn't matter.

Example:

```
Input:
   List1: 10->15->4->20
   lsit2:  8->4->2->10
Output:
   Intersection List: 4->10
   Union List: 2->8->20->4->15->10
```

Method 1 (Simple)

Following are simple algorithms to get union and intersection lists respectively.

Intersection (list1, list2)

Initialize result list as NULL. Traverse list1 and look for its each element in list2, if the element is present in list2, then add the element to result.

Union *(list1, list2):*

Initialize result list as NULL. Traverse list1 and add all of its elements to the result.

Traverse list2. If an element of list2 is already present in result then do not insert it to result, otherwise insert.

This method assumes that there are no duplicates in the given lists.

Thanks to Shekhu for suggesting this method. Following are C and Java implementations of this method.

C/C++

```
// C/C++ program to find union and intersection of two unsorted
// linked lists
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct node
{
```

```
    int data;
    struct node* next;
};

/* A utility function to insert a node at the beginning of
   a linked list*/
void push(struct node** head_ref, int new_data);

/* A utility function to check if given data is present in a list */
bool isPresent(struct node *head, int data);

/* Function to get union of two linked lists head1 and head2 */
struct node *getUnion(struct node *head1, struct node *head2)
{
    struct node *result = NULL;
    struct node *t1 = head1, *t2 = head2;

    // Insert all elements of list1 to the result list
    while (t1 != NULL)
    {
        push(&result, t1->data);
        t1 = t1->next;
    }

    // Insert those elements of list2 which are not
    // present in result list
    while (t2 != NULL)
    {
        if (!isPresent(result, t2->data))
            push(&result, t2->data);
        t2 = t2->next;
    }

    return result;
}

/* Function to get intersection of two linked lists
  head1 and head2 */
struct node *getIntersection(struct node *head1,
                             struct node *head2)
{
    struct node *result = NULL;
    struct node *t1 = head1;

    // Traverse list1 and search each element of it in
    // list2. If the element is present in list 2, then
    // insert the element to result
    while (t1 != NULL)
    {
        if (isPresent(head2, t1->data))
            push (&result, t1->data);
        t1 = t1->next;
    }

    return result;
}

/* A utility function to insert a node at the begining of a linked 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;
}

/* A utility function to print a linked list*/
void printList (struct node *node)
{
    while (node != NULL)
    {
        printf ("%d ", node->data);
        node = node->next;
    }
}

/* A utility function that returns true if data is
   present in linked list else return false */
bool isPresent (struct node *head, int data)
{
    struct node *t = head;
    while (t != NULL)
    {
        if (t->data == data)
            return 1;
        t = t->next;
    }
    return 0;
}

/* Drier program to test above function*/
int main()
{
    /* Start with the empty list */
    struct node* head1 = NULL;
    struct node* head2 = NULL;
    struct node* intersecn = NULL;
    struct node* unin = NULL;

    /*create a linked lits 10->15->5->20 */
    push (&head1, 20);
    push (&head1, 4);
    push (&head1, 15);
    push (&head1, 10);

    /*create a linked lits 8->4->2->10 */
    push (&head2, 10);
    push (&head2, 2);
    push (&head2, 4);
    push (&head2, 8);

    intersecn = getIntersection (head1, head2);
    unin = getUnion (head1, head2);

    printf ("\n First list is \n");
    printList (head1);

    printf ("\n Second list is \n");
    printList (head2);

    printf ("\n Intersection list is \n");
    printList (intersecn);
```

```
    printf ("\n Union list is \n");
    printList (unin);

    return 0;
}
```

Run on IDE

Java

```
// Java program to find union and intersection of two unsorted
// linked lists
class LinkedList
{
    Node head;  // head of list

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

    /* Function to get Union of 2 Linked Lists */
    void getUnion(Node head1, Node head2)
    {
        Node t1 = head1, t2 = head2;

        //insert all elements of list1 in the result
        while (t1 != null)
        {
            push(t1.data);
            t1 = t1.next;
        }

        // insert those elements of list2 that are not present
        while (t2 != null)
        {
            if (!isPresent(head, t2.data))
                push(t2.data);
            t2 = t2.next;
        }
    }

    void getIntersection(Node head1, Node head2)
    {
        Node result = null;
        Node t1 = head1;

        // Traverse list1 and search each element of it in list2.
        // If the element is present in list 2, then insert the
        // element to result
        while (t1 != null)
        {
            if (isPresent(head2, t1.data))
                push(t1.data);
```

```
            t1 = t1.next;
        }
    }

    /* Utility function to print list */
    void printList()
    {
        Node temp = head;
        while(temp != null)
        {
            System.out.print(temp.data+" ");
            temp = temp.next;
        }
        System.out.println();
    }


    /*  Inserts a node at start of linked 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;
    }


    /* A utilty function that returns true if data is present
       in linked list  else return false */
    boolean isPresent (Node head, int data)
    {
        Node t = head;
        while (t != null)
        {
            if (t.data == data)
                return true;
            t = t.next;
        }
        return false;
    }


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

        /*create a linked lits 10->15->5->20 */
        llist1.push(20);
        llist1.push(4);
        llist1.push(15);
        llist1.push(10);

        /*create a linked lits 8->4->2->10 */
        llist2.push(10);
        llist2.push(2);
        llist2.push(4);
```

```
        llist2.push(8);

        intersecn.getIntersection(llist1.head, llist2.head);
        unin.getUnion(llist1.head, llist2.head);

        System.out.println("First List is");
        llist1.printList();

        System.out.println("Second List is");
        llist2.printList();

        System.out.println("Intersection List is");
        intersecn.printList();

        System.out.println("Union List is");
        unin.printList();
    }
} /* This code is contributed by Rajat Mishra */
```

Run on IDE

Output:

```
 First list is
10 15 4 20
 Second list is
8 4 2 10
 Intersection list is
4 10
 Union list is
2 8 20 4 15 10
```

Time Complexity: O(mn) for both union and intersection operations. Here m is the number of elements in first list and n is the number of elements in second list.

Method 2 (Use Merge Sort)

In this method, algorithms for Union and Intersection are very similar. First we sort the given lists, then we traverse the sorted lists to get union and intersection.

Following are the steps to be followed to get union and intersection lists.

1) Sort the first Linked List using merge sort. This step takes O(mLogm) time. Refer this post for details of this step.

2) Sort the second Linked List using merge sort. This step takes O(nLogn) time. Refer this post for details of this step.

3) Linearly scan both sorted lists to get the union and intersection. This step takes O(m + n) time. This step can be implemented using the same algorithm as sorted arrays algorithm discussed here.

Time complexity of this method is O(mLogm + nLogn) which is better than method 1's time complexity.

Method 3 (Use Hashing)

Union (list1, list2)
Initialize the result list as NULL and create an empty hash table. Traverse both lists one by one, for each element being visited, look the element in hash table. If the element is not present, then insert the element to result list. If the element is present, then ignore it.

Intersection (list1, list2)
Initialize the result list as NULL and create an empty hash table. Traverse list1. For each element being visited in list1, insert the element in hash table. Traverse list2, for each element being visited in list2, look the element in hash table. If the element is present, then insert the element to result list. If the element is not present, then ignore it.

Both of the above methods assume that there are no duplicates.

Time complexity of this method depends on the hashing technique used and the distribution of elements in input lists. In practical, this approach may turn out to be better than above 2 methods.

Source: http://geeksforgeeks.org/forum/topic/union-intersection-of-unsorted-lists

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

56 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)

2.5 Average Difficulty : **2.5/5.0**
Based on **4** vote(s)

Add to TODO List
Mark as DONE

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

56 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

RAJAT • 4 months ago
Method -3 Hashing implementation

http://ideone.com/GxqPhd

• Reply • Share ›

RODNEY JEROME • 4 months ago
It causes conflicting type error.
There is no need to give the function prototype! as the function definition is also above the main method!!
BTW its was relle helpful. Keep up the good work!!1

• Reply • Share ›

StoneGates • 5 months ago
In method 1, the complexity of union is larger than O(mn), right? Because the temporal result is increasing.

• Reply • Share ›

Hemil Kadakia • 5 months ago
For method 3 instead of walking the list twice to create separate intersection and union list why not use the foll. method :
Step 1: Create an empty hash table. Traverse list1 and if not already present mark the element in the hash table. If the element was not already present also add it in the union

list.
Step 2: Now traverse list2. Check for each element in the hash table. If entry exists in hash table, insert that element in the intersection list.

The above algorithm does not work for duplicates in list2 but in list1.

• Reply • Share ›

Karan Kapoor • 5 months ago

What about duplicates in the two linked lists ?
do we take both of them ?

• Reply • Share ›

Balaji • 5 months ago

//using merge sort and removing duplicates

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node *next;

};

struct node *newnode(struct node *head,int data);

struct node *sortedmerge(struct node *a,struct node *b);

void mergesort(struct node **head_ref);

see more

• Reply • Share ›

SlickHackz • 6 months ago

Method 2 (Merge Sort) - Iteration
Inputs are unsorted. They are sorted using MergeSort.
http://code.geeksforgeeks.org/...

• Reply • Share ›

SlickHackz • 6 months ago

Method 2 (Merge Sort) - Recursion
Inputs are unsorted. They are sorted using MergeSort.

http://code.geeksforgeeks.org/...

• Reply • Share ›

Deepan Negi • 7 months ago

(Correction)
Method 3 (Use Hashing)
Union (list1, list2)
Initialize the result list as NULL and create an empty hash table. Traverse both lists one by one, for each element being visited, look the element in hash table. If the element is not present, then insert the element to result list AND THE HASH TABLE. If the element is present, then ignore it.

• Reply • Share ›

Bewkoof_coder • 7 months ago

why its in order 2 -> 8 -> 20 ->4 etc
not like 8 -> 4 -> 2 -> 10 ->25 ->20 ???

• Reply • Share ›

Deepan Negi ➚ Bewkoof_coder • 7 months ago

Read the problem statement..

it says: "Order of elments (should be elements) in output lists doesn't matter"

• Reply • Share ›

Mohammad Nadeem Bhati ➚ Bewkoof_coder • 7 months ago

same que Does order matters here?

• Reply • Share ›

Ashish Maheshwari • 9 months ago

C++ implementation of method 1 with union complexity as (m+n) and intersection complexity as (mn):
https://github.com/maheshwari-...

• Reply • Share ›

Rekha Hindwar • 10 months ago

we can improve the complexity... by traversing both list simultaneously it will take o(m) or o(n) whichever is bigger instead of o(m+n).

• Reply • Share ›

devakar verma ➚ Rekha Hindwar • 9 months ago

You can't do so

• Reply • Share ›

Klaus ➚ Rekha Hindwar • 10 months ago

The two lists are not sorted.

• Reply • Share ›

Binary • a year ago

In this programme union will fail if there are repeated elements in first list

• Reply • Share ›

Paritosh Agrawal ➚ Binary • 9 months ago

even if we want to do so, we can use concept of visited

• Reply • Share ›

Paritosh Agrawal ➚ Binary • 9 months ago

while we are talking about union it is understood that we are supposed to do union of 2 or more sets..
and set can not contain repeated elements..
so here consider 2 linked lists as 2 sets..

1 • Reply • Share ›

lovey • a year ago

union and intersection of two list when we have duplicates then use visited array concept i.e. Hashing

ideone link of the code:http://ideone.com/4guGQ8

• Reply • Share ›

温发琥 • a year ago

here's a simple example of method 2, a, b are the head nodes of 2 sorted linked list

```
void inter(Node* a, Node* b, struct Linkedlist* res)

{

if(a!=NULL&&b!=NULL)

{

if(a->data == b->data)

{

nodePushBack(res, a->data);

inter(a->next, b->next, res);

}
```

```
else if(a->data < b->data)
```

see more

1 • Reply • Share ›

random_dude • a year ago

Third method can be easily modified to work with duplicates as well. Instead of storing whether the element is present in the hash, store the number of occurrences of each element...

2 • Reply • Share ›

xxxxx • a year ago

<script>alert(1)</script>

• Reply • Share ›

Manraj Singh ➚ xxxxx • a year ago

Fail.

• Reply • Share ›

super • a year ago

please tell me how hashing work

• Reply • Share ›

Jun • a year ago

can anybody please share implementation of method3

• Reply • Share ›

Abhinav Bhardwaj ➚ Jun • a year ago

This is Java implementation of Intersection of two Linked List using Hashing

```
public Node findIntersection(Node head1, Node head2) {
HashSet<integer> myhash = new HashSet<integer>();

Node firstCurrent = head1;
Node secondCurrent = head2;
Node IntersectionHead = null;

Node prev = null;
while (firstCurrent != null) {
myhash.add(firstCurrent.data);
firstCurrent = firstCurrent.next;
}
while (secondCurrent != null) {
if (myhash.contains(secondCurrent.data)) {
```

```
Node newNode = new Node(secondCurrent.data);
if (prev == null)
IntersectionHead = newNode;
```

see more

• Reply • Share ›

pradeep kumar • 2 years ago

why do we use these type of assignment in many functions:-

struct node *t1 = head1, *t2 = head2;

And we start use t1 and t2.....plz tell me.....

• Reply • Share ›

GOPI GOPINATH ➚ pradeep kumar • 2 years ago

We are assigning the heads of the lists to temporary pointers in order to traverse the list, yet preserving the heads of the list .

• Reply • Share ›

ashish jaiswal • 2 years ago

```
#include<stdio.h>

#include<stdlib.h>

struct node;

void push(struct node**, int);

void print(struct node*);

struct node* getunion(struct node*, struct node*);

struct node* getintersect(struct node*, struct node*);

typedef struct node

{

int data;

struct node*next;
```

see more

• Reply • Share ›

ashish jaiswal ➚ ashish jaiswal • a year ago

sorry complexity is more

sorry complexity is more..

• Reply • Share ›

Himanshu Dagar • 2 years ago

In the third method of Using Hash Table what will be its complexity??
Will it be O(m+n) or O(mn)??

If searching for an element in hash table ,it will take O(n) time then surely it will be done in O(mn) .So is it true??

• Reply • Share ›

Avata

This comment was deleted.

ryan Guest • a year ago

it will be O(max(m,n))

• Reply • Share ›

Nishanth • 2 years ago

what will be the size of hash array? that is , the integer can be -ve too right?? then how can we hash it??

• Reply • Share ›

Sumit Poddar • 3 years ago

I think there is one more method which will take time complexity of O(n) where n is the number of elements in the list which has maximum number of elements. Below is the code in Java.

```
public class SplitList {

   public static void main(String[] args) {
       Node n1 = new Node(20, null);
       Node n2 = new Node(4, n1);
       Node n3 = new Node(15, n2);
       Node n = new Node(10, n3);

       Node m1 = new Node(10, null);
       Node m2 = new Node(2, m1);
       Node m3 = new Node(4, m2);
       Node m = new Node(8, m3);

       split(n, m);
```

see more

• Reply • Share ›

Arun • 3 years ago

1 more method to find the intersection:

1. Sort the 1 list with n elements -> O(nlogn).
2. Use binary search for finding each element of list 2(m) in the sorted list1 -> O(mlogn).
Hence we get the results in overall O((m+n)logn) which is better than the Method 2 suggest above using merge sort.

• Reply • Share ›

Aryaa Arun • a year ago

binary search is not possible with linked list as it needs random access

1 • Reply • Share ›

eragon Arun • 2 years ago

And how do you do a logn binary search on a singly linked list?

1 • Reply • Share ›

gaurav Arun • 3 years ago

I think we can get union also by this method with little modification

Initialize union with first list
initialize intersection with null

while performing binary search -->
1.element is there in first list-->add to intersection list
2.element not present-->add to union list

correct if i am wrong..

• Reply • Share ›

Gupta gaurav • 2 years ago

I think it is also asymptotically better than above merge sort method... coz we will select the list to sort which has less number of nodes, by checking countnodes1 and countnodes2... so time complexity will become O((m+n)logn), where n<m, which="" is="" better="" than="" o(mlogm+nlogn).="" please="" correct="" me="" if="" i="" am="" wrong..="">

• Reply • Share ›

linux.kindle • 4 years ago

how abt. we maintain a BST for implementing intersection of LLs? We can then traverse the tree to get the output.

• Reply • Share ›

Rediff • 4 years ago

```
/* Paste your code here (You may delete these lines if not writing code) */
```

public class UnionIntersection{

public static Node unionOfTwoLists (Node a, Node b){

HashMap commons = new HashMap();

Node union = null;

while(a!=null){
if(union == null){
union = new Node(a.data);
commons.put(a.data, null);
}

if(!commons.containsKey(a.data) && union != null)
{
commons.put(a.data, null);

see more

• Reply • Share ›

GeeksforGeeks • 4 years ago

@Shekhu: The method 1 assumes that there are no duplicates in the input lists. You can modify the intersection function to following to handle duplicates.

```
/* Function to get intersection of two linked lists head1 and head2 */
struct node *getIntersection (struct node *head1, struct node *head2)
{
    struct node *result = NULL;
    struct node *t1 = head1;

    while (t1 != NULL)
    {
        // Note the second condition
        if (isPresent(head2, t1->data) && !isPresent(result, t1->data))
            push (&result, t1->data);
        t1 = t1->next;
    }

    return result;
}
```

• Reply • Share ›

Shekhu • 4 years ago

The first Method given above would not work when the first list is having duplicate values.

e.g.
List1={1->4->5->1->4}
List2={1->4->9->78}

Intersection result would be {1->4->1->4}
whereas it should have been {1->4} only.

• Reply • Share ›

Ritesh Shekhu • 4 years ago

Dude ,when they are talking about union and intersection assume that the list given will follow the property of a set.

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

Nitin Gupta • 4 years ago

Well i have another solution for this....
First traverse First linked list and build a binary tree (say it BIF) (for duplicate element , put it on the left of that element)
complexity O(n logn)

then traverse second list (complexity O(n)) and and take each element of Second list and match to element in BIF tree...if it found then put it in a Intersection Linked and if not found then connect that element in BIF tree. Complexity (n logn)
On Result you have a Intersection Linked List and A BIF Tree which is Your Union List ...Build a Linked List using BIF Tree..... (Note when you traverse whole BIF tree then delete it)
Total complexity O(n log n)

This technique will also handle duplicate element . Using this technique you can reduce complexity to O(n).

```
/* Paste your code here (You may delete these lines if not writing code) */
```

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

coder Nitin Gupta • 4 years ago

there is one problem if input is like below
List 1 :- 4 6 8 9
List 2 :- 5 2 2 10

Now as you said if element is not found then add to the BT of the 1st List so 1st 2 of the List 2 will be added to the Binary Tree so your BT will have 4 6 8 9 2 now for the next 2 of the List 2 element will be found in the BT and that will be added to the intersection (this is wrong) because intersection of both list is {} , not {2}.
so better unmatched elements should be added to BT only after list 2 elements are exhausted

• Reply • Share ›

anonymous Nitin Gupta • 4 years ago

what should be intersection & union list when,
list1 : 1->2->3
list2 : 2->2->2

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

anonymous Nitin Gupta • 4 years ago

very nice solution. :)

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

Load more comments