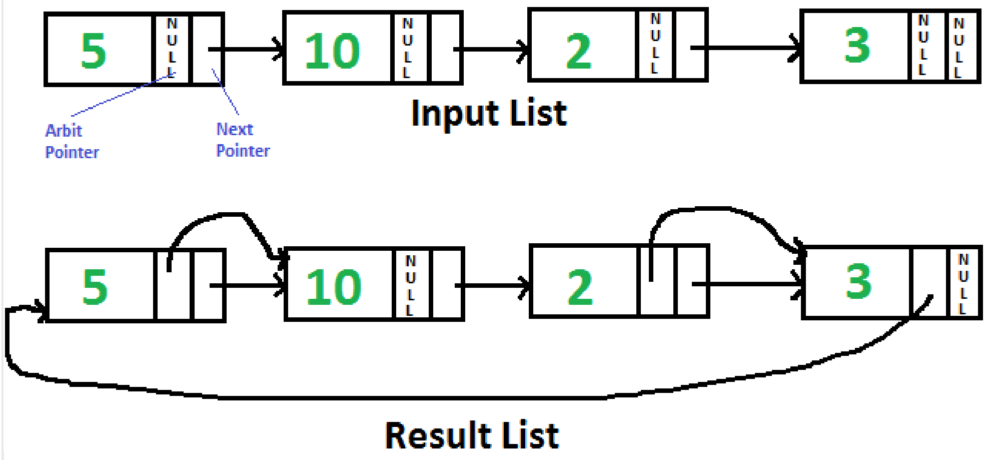
Point to next higher value node in a linked list with an arbitrary pointer

Given singly linked list with every node having an additional “arbitrary” pointer that currently points to NULL. Need to make the “arbitrary” pointer point to the next higher value node.



We strongly recommend to minimize your browser and try this yourself first

A **Simple Solution** is to traverse all nodes one by one, for every node, find the node which has next greater value of current node and change the next pointer. Time Complexity of this solution is O(n^2.

An **Efficient Solution** works in O(nLogn) time. The idea is to use Merge Sort for linked list.
1) Traverse input list and copy next pointer to arbit pointer for every node.
2) Do Merge Sort for the linked list formed by arbit pointers.

Below is C implementation of above idea. All of the merger sort functions are taken from here. The taken functions are modified here so that they work on arbit pointers instead of next pointers.

```c
// C program to populate arbit pointers to next higher value
// using merge sort
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct node
{
    int data;
    struct node* next, *arbit;
};

/* function prototypes */
struct node* SortedMerge(struct node* a, struct node* b);
void FrontBackSplit(struct node* source,
                    struct node** frontRef, struct node** backRef);

/* sorts the linked list formed by arbit pointers
  (does not change next pointer or data) */
void MergeSort(struct node** headRef)
{
    struct node* head = *headRef;
    struct node* a, *b;

    /* Base case -- length 0 or 1 */
    if ((head == NULL) || (head->arbit == NULL))
        return;

    /* Split head into 'a' and 'b' sublists */
    FrontBackSplit(head, &a, &b);

    /* Recursively sort the sublists */
    MergeSort(&a);
    MergeSort(&b);

    /* answer = merge the two sorted lists together */
    *headRef = SortedMerge(a, b);
}

/* See http://geeksforgeeks.org/?p=3622 for details of this
   function */
struct node* SortedMerge(struct node* a, struct node* b)
{
    struct node* result = NULL;

    /* Base cases */
    if (a == NULL)
        return (b);
    else if (b==NULL)
        return (a);

    /* Pick either a or b, and recur */
    if (a->data <= b->data)
    {
        result = a;
        result->arbit = SortedMerge(a->arbit, b);
    }
    else
    {
        result = b;
        result->arbit = SortedMerge(a, b->arbit);
    }

    return (result);
}

/* Split the nodes of the given list into front and back halves,
   and return the two lists using the reference parameters.
   If the length is odd, the extra node should go in the front list.
   Uses the fast/slow pointer strategy.  */
void FrontBackSplit(struct node* source,
                    struct node** frontRef, struct node** backRef)
{
    struct node* fast, *slow;

    if (source==NULL || source->arbit==NULL)
    {
        /* length < 2 cases */
        *frontRef = source;
        *backRef = NULL;
        return;
    }

    slow = source,  fast = source->arbit;

    /* Advance 'fast' two nodes, and advance 'slow' one node */
    while (fast != NULL)
    {
        fast = fast->arbit;
        if (fast != NULL)
        {
            slow = slow->arbit;
            fast = fast->arbit;
        }
    }

    /* 'slow' is before the midpoint in the list, so split it in two
      at that point. */
    *frontRef = source;
    *backRef = slow->arbit;
    slow->arbit = NULL;
}

/* Function to insert a node at the beginging of the 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);

    new_node->arbit = NULL;

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

// Utility function to print result linked list
void printListafter(struct node *node, struct node *anode)
{
    printf("Traversal using Next Pointer\n");
    while (node!=NULL)
    {
        printf("%d, ", node->data);
        node = node->next;
    }

    printf("\nTraversal using Arbit Pointer\n");
    while (anode!=NULL)
    {
        printf("%d, ", anode->data);
        anode = anode->arbit;
    }
```

```
}

// This function populates arbit pointer in every node to the
// next higher value. And returns pointer to the node with
// minimum value
struct node* populateArbit(struct node *head)
{
    // Copy next pointers to arbit pointers
    struct node *temp = head;
    while (temp != NULL)
    {
        temp->arbit = temp->next;
        temp = temp->next;
    }

    // Do merge sort for arbitrary pointers
    MergeSort(&head);

    // Return head of arbitrary pointer linked list
    return head;
}

/* Drier program to test above functions*/
int main()
{
    /* Start with the empty list */
    struct node* head = NULL;

    /* Let us create the list shown above */
    push(&head, 3);
    push(&head, 2);
    push(&head, 10);
    push(&head, 5);

    /* Sort the above created Linked List */
    struct node *ahead = populateArbit(head);

    printf("\nResult Linked List is: \n");
    printListafter(head, ahead);

    getchar();
    return 0;
}
```

Run on IDE

Output:

```
Result Linked List is:
Traversal using Next Pointer
5, 10, 2, 3,
Traversal using Arbit Pointer
2, 3, 5, 10,
```

This article is contributed by **Saurabh Bansal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

30 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
- Swap nodes in a linked list without swapping data
- Generic Linked List in C

(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

Like Share 34 people like this.

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

30 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

Pawan Samarwal • 3 months ago

```
#include <iostream>

using namespace std;

struct node {

int data;

node *next;

node *arb;

};

void create(node *&t,int a)

{

if(t==NULL)
```

see more

• Reply • Share ›

Hitendra Singh Rathore • 4 months ago

but the problem is that what if there are duplicate values in list

• Reply • Share ›

sasha • 6 months ago

I dont understqnd the point in sorting the nodes.
Plz explain.

• Reply • Share ›

vergil sasha • 6 months ago

We have to make each node's arbit point to smallest node greater than it. In other words, on traversing the list using arbit pointers we should get a sorted list. Hence, sorting is used.

• Reply • Share ›

BeautifulCode • 6 months ago

Inside populateArbit() method

MergeSort(&head); // head is a local parameter how come passing address of this make sense ?

• Reply • Share ›

Sachit Sharma • 6 months ago

Same approach as http://www.geeksforgeeks.org/n...
in O(n) time and O(n) Auxillary space

• Reply • Share ›

vergil Sachit Sharma • 6 months ago

We don't have to find the greater on the right side of the node!
Instead we need to find the smallest element greater than it in the list (can be before it or even after it).

• Reply • Share ›

Nikhil Khurana vergil • 5 months ago

this approach can be used it would take just one more traversal , for remaining elemnts in stack .

• Reply • Share ›

Bewkoof_coder • 6 months ago

better solution O(n) time complexity using stack...
traverse first..O(n)
then again traverse..O(n)..
overalla O(n)+O(n)===O(n)

o/p of my code is=

5-->nextGreater-->10

10-->nextGreater-->NULL

2-->nextGreater-->3

3-->nextGreater-->5....

code:

void nextGreater(struct node *start){

struct node *temp=start;

see more

• Reply • Share ›

Hitesh Saini • 7 months ago

what happens when our linked list has duplicate nodes..??
method 1 will work but method two in which we use merge sort will not give the correct result..??

Correct me if i am wrong

3 ^ | v • Reply • Share ›

Naveen Kumar Badgujar ➦ Hitesh Saini • 6 months ago

yes, method 2 will give wrong result if there are duplicates in the given LL.

1 ^ | v • Reply • Share ›

Abhimanyu Mittal • 7 months ago

Can we use hash table for the above question. Moreover it will be solved in O(n) time?

^ | v • Reply • Share ›

Hitesh Saini ➦ Abhimanyu Mittal • 7 months ago

how ..?
what will be your hash function.?

^ | v • Reply • Share ›

Abhimanyu Mittal ➦ Hitesh Saini • 6 months ago

The normal hash function storing the keys for the elements. This hash table will be able to perform the operations in O(n) time.

^ | v • Reply • Share ›

Hitesh Saini ➦ Abhimanyu Mittal • 6 months ago

How..??
Explain Your Complete Algorithm.

^ | v • Reply • Share ›

Bhoop Singh • 7 months ago

This can be solved in O(n) time using a STACK with O(n) space complexity.

Steps
(1) Just traverse the linked list ,for every new node say N1 while traversing compare that to the top most element in stack if N1 node value is higher keep popping nodes from stack till the condition meets and set the N1 as next higher node for all such nodes.
(2) Add the node N1 on top of the stack.

^ | v • Reply • Share ›

Eknoor ➦ Bhoop Singh • 5 months ago

This solution will only give the first greater element on the right side. We want the smallest greatest number in the entire linked list and not just from the right side

^ | ˅ • Reply • Share ›

Hitesh Saini ➦ Bhoop Singh • 7 months ago
this will not work ..
take an example 5->11->3->6->8->10
then your stack will be
10
8
6
3
11
5

when i traverse and comparing element with top of the stack it will produce wrong answer

e.g.
for 5 it will give answer 10 while the correct answer is 6.

Correct me if i am wrong..

^ | ˅ • Reply • Share ›

Sachit Sharma ➦ Hitesh Saini • 6 months ago
I guess @bhoop is referring to a approach similar to this question http://www.geeksforgeeks.org/n...

And I think its working here.

^ | ˅ • Reply • Share ›

shubham garg • 7 months ago
A better approach can be first we can reverse the linked list and then first node will be max for current element and we traverse and assign arbitary pointer to current max and traverse in forward direction and if we encounter a node of greater value then it becomes new max and when we reach end we can reverse it. eg-
1->3->5->4->2
reversing it gives us
2->4->5->3->1
current max =NULL
so 2 arbitary pointer points to NULL
now max becomes 2
4 arbitray pointer points to 2
4>2
so new max becomes 4
5 arbitary pointer points to 4
and new max becomes 5

3 arbitary pointer points to 5
max remains same
1 arbitary pointer points to 5
now we again reverse the list
so total complexity will be
O(n) + O(n) + O(n) = O(n)
1 • Reply • Share ›

sandeep Agrawal shubham garg • 7 months ago
i think above sol will not work for some outputs like 1-5-7-8-9
• Reply • Share ›

Goku • 7 months ago
Clearner Code in C++ : http://ideone.com/AFkXNt
• Reply • Share ›

sage • 7 months ago
Isn't this the Next Greater Element problem? Use a stack<node*> for O(n) time.
• Reply • Share ›

varun • 7 months ago
Traversal using arbit pointer always forms sorted linked list.Is it right?
• Reply • Share ›

Udit • 7 months ago
@geeksforgeeks
It will be better if we use recursion.
we traverse the list recursively. When the last node is reached we set arbit to null and max to the value of the last node and store the node corresponding to max
we keep updating value of max as the nodes pop and if the curreent value is less than max,we make its arbit to point to the max node
• Reply • Share ›

GOPI GOPINATH • 9 months ago
change the title of the post to
Point to next higher value node on any side in a linked list with an arbitrary pointer of the list.
• Reply • Share ›

Rk Srinivas • 9 months ago
can't we use stack and do it in 0(n).
traverse from head of list to end of list
while(stack not empty and cur_elem > stack top)

stack top arbit ptr = cur element
pop stack top
store cur_elem it in stack

• Reply • Share ›

ltc Rk Srinivas • 9 months ago

No it won't work. Question is to link arbitrary pointer to next higher value node. In your solution it will link to any higher value node which comes first.

Example:
5 -> 10 -> 6 -> NULL

Arbitrary pointer of 5 should point to 6 not 10

1 • Reply • Share ›

Rk Srinivas ltc • 9 months ago

Thanks, I misunderstood the question

• Reply • Share ›

Mudit Rk Srinivas • 9 months ago

+1 ...was about to comment same. Clean, simple and elegant way.

• Reply • Share ›