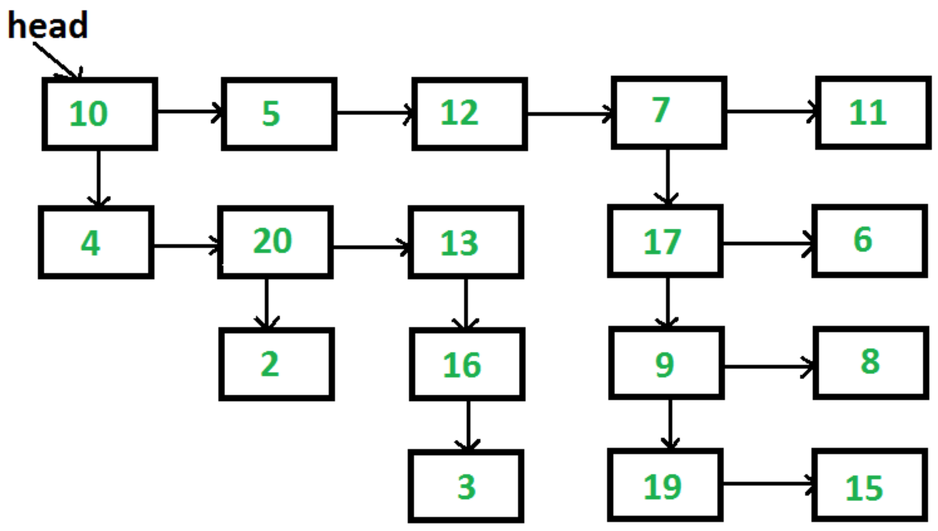
Flatten a multilevel linked list

Given a linked list where in addition to the next pointer, each node has a child pointer, which may or may not point to a separate list. These child lists may have one or more children of their own, and so on, to produce a multilevel data structure, as shown in below figure.You are given the head of the first level of the list. Flatten the list so that all the nodes appear in a single-level linked list. You need to flatten the list in way that all nodes at first level should come first, then nodes of second level, and so on.

Each node is a C struct with the following definition.

```
struct list
{
    int data;
    struct list *next;
    struct list *child;
};
```

Run on IDE



The above list should be converted to 10->5->12->7->11->4->20->13->17->6->2->16->9->8->3->19->15

The problem clearly say that we need to flatten level by level. The idea of solution is, we start from first level, process all nodes one by one, if a node has a child, then we append the child at the end of list, otherwise we don't do anything. After the first level is processed, all next level nodes will be appended after first level. Same process is followed for the appended nodes.

```
1) Take "cur" pointer, which will point to head of the fist level of the list
2) Take "tail" pointer, which will point to end of the first level of the list
3) Repeat the below procedure while "curr" is not NULL.
    I) if current node has a child then
        a) append this new child list to the "tail"
                tail->next = cur->child
        b) find the last node of new child list and update "tail"
                tmp = cur->child;
                while (tmp->next != NULL)
                        tmp = tmp->next;
                tail = tmp;
    II) move to the next node. i.e. cur = cur->next
```

Following is C implementation of the above algorithm.

```
// Program to flatten list with next and child pointers
#include <stdio.h>
#include <stdlib.h>

// Macro to find number of elements in array
#define SIZE(arr) (sizeof(arr)/sizeof(arr[0]))

// A linked list node has data, next pointer and child pointer
struct node
{
    int data;
    struct node *next;
    struct node *child;
};

// A utility function to create a linked list with n nodes. The data
// of nodes is taken from arr[].  All child pointers are set as NULL
struct node *createList(int *arr, int n)
{
    struct node *head = NULL;
    struct node *p;

    int i;
    for (i = 0; i < n; ++i) {
        if (head == NULL)
            head = p = (struct node *)malloc(sizeof(*p));
        else {
            p->next = (struct node *)malloc(sizeof(*p));
            p = p->next;
        }
        p->data = arr[i];
        p->next = p->child = NULL;
    }
    return head;
}

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

// This function creates the input list.  The created list is same
// as shown in the above figure
struct node *createList(void)
{
    int arr1[] = {10, 5, 12, 7, 11};
    int arr2[] = {4, 20, 13};
    int arr3[] = {17, 6};
    int arr4[] = {9, 8};
    int arr5[] = {19, 15};
    int arr6[] = {2};
    int arr7[] = {16};
    int arr8[] = {3};

    /* create 8 linked lists */
    struct node *head1 = createList(arr1, SIZE(arr1));
    struct node *head2 = createList(arr2, SIZE(arr2));
    struct node *head3 = createList(arr3, SIZE(arr3));
    struct node *head4 = createList(arr4, SIZE(arr4));
    struct node *head5 = createList(arr5, SIZE(arr5));
    struct node *head6 = createList(arr6, SIZE(arr6));
    struct node *head7 = createList(arr7, SIZE(arr7));
    struct node *head8 = createList(arr8, SIZE(arr8));


    /* modify child pointers to create the list shown above */
    head1->child = head2;
    head1->next->next->next->child = head3;
    head3->child = head4;
    head4->child = head5;
    head2->next->child = head6;
    head2->next->next->child = head7;
    head7->child = head8;


    /* Return head pointer of first linked list.  Note that all nodes are
       reachable from head1 */
    return head1;
}

/* The main function that flattens a multilevel linked list */
void flattenList(struct node *head)
{
    /*Base case*/
    if (head == NULL)
       return;

    struct node *tmp;

    /* Find tail node of first level linked list */
    struct node *tail = head;
    while (tail->next != NULL)
        tail = tail->next;

    // One by one traverse through all nodes of first level
    // linked list till we reach the tail node
    struct node *cur = head;
    while (cur != tail)
    {
        // If current node has a child
        if (cur->child)
        {
            // then append the child at the end of current list
            tail->next = cur->child;

            // and update the tail to new last node
            tmp = cur->child;
            while (tmp->next)
                tmp = tmp->next;
            tail = tmp;
        }

        // Change current node
        cur = cur->next;
    }
}

// A driver program to test above functions
int main(void)
{
    struct node *head = NULL;
    head = createList();
    flattenList(head);
    printList(head);
    return 0;
}
```

Run on IDE

Output:

```
10 5 12 7 11 4 20 13 17 6 2 16 9 8 3 19 15
```

Time Complexity: Since every node is visited at most twice, the time complexity is O(n) where n is the number of nodes in given linked list.

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

95 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)

0 Average Difficulty : **0/5.0**
No votes yet.

☐ Add to TODO List
☐ Mark as DONE

Like Share 57 people like this.

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

95 Comments GeeksforGeeks 1 Login

Recommend 3 Share Sort by Newest

Join the discussion…

GOPI RAM BISHNOI • a month ago

using queue :)

```
void flattenList(struct node *head,struct queue *s)

{

if(head==NULL)

{

head=delet(s);

if(head==NULL)

return;

}
printf("%d ",head->data);

if(head->child)

insert(head->child,s);

flattenList(head->next,s);

}
```

• Reply • Share ›

Payal Priyadarshini • 2 months ago

Queue can be used to flatten the linked list. BFS.

• Reply • Share ›

neeraj kumar Payal Priyadarshini • 2 months ago

Exactly!

Keep saving child nodes in a queue, and dequeue them to insert as soon as next is found null.

• Reply • Share ›

ss • 3 months ago

y are we not doing cur->child=NULL, once we have appended it at last

• Reply • Share ›

Drogba • 3 months ago

simple O(N) solution

```
void flattenList(struct node *head)

{

/*Base case*/

if (head == NULL)

return;

struct node *tmp;

struct node *current = head;

while (current != NULL)

{
```

tmp = current;

see more

• Reply • Share ›

Drogba • 3 months ago

void flattenList(struct node *head)

{

/*Base case*/

if (head == NULL)

return;

struct node *tmp;

struct node *current = head;

while (current != NULL)

{

tmp = current;

while(tmp->next)

tmp = tmp->next;

tmp->next = current->child;

current = current->next;

}

}

• Reply • Share ›

grokbrokkoli • 5 months ago

In the above code, after "tail = tmp" you need to add "cur->child = NULL". Otherwise, you retain the child pointer after moving it to the end and it becomes very messy. It's not seen because the print function only prints the next, not the child.

2 • Reply • Share ›

Karan Kapoor • 5 months ago

this is modified BFS :)

• Reply • Share ›

sahil narula • 6 months ago

Simple solution using queue in O(N):

/* Author: Sahil Narula

Date: 11 July 2015 (Saturday)

Time: 16:56

Description: Guillermo

*/

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node* next;

see more

• Reply • Share ›

gsd • 6 months ago

isn't the time complexity n^2, i also need to update tail pointers at end of each level inside the while loop. the outer loop runs for O(n). inside while loop also tail update takes O(n) max.? am i wrong?

• Reply • Share ›

Lokesh • 6 months ago

Looks like there is a bug. In the end we need to check for the child of tail node also i.e child of node 15 in the above case.

2 • Reply • Share ›

prashant patel Lokesh • 5 months ago

i also think so, i have implemented with the condition (current!=null).

• Reply • Share ›

Karan Tankshali Lokesh • 6 months ago

Yes I also think so, the condition should be while(current!=null)

• Reply • Share ›

Return_0 • 6 months ago

Simple iterative code...no queue required O(1) space...O(n) time..

void flattenList(node *head)

{

node *f1,*f2;

f1=f2=head;

while(f2!=NULL)

{

while(f1->next!=NULL)

f1=f1->next;

while(f2!=NULL && f2->child==NULL)

f2=f2->next;

see more

1 • Reply • Share ›

Hitesh Saini • 7 months ago

One more Method
We can use queue.
whenever we found a child node then we will insert it in queue
and pop the queue when the one list reaches its end.

in this method we will traverse each node only once so the time complexity is O(n).

Here is my function STL queue.

```
void flattenList(struct node *head)
{
if(head==NULL)
return;
else
{
struct node *p=head;
while(p!=NULL)
{
printf("%d ",p->data);
```

see more

2 • Reply • Share ›

Sarthak Garg → Hitesh Saini • 6 months ago

Good one! Exactly the solution I was thinking about.

1 • Reply • Share ›

Shashank Shekhar → Sarthak Garg • 6 months ago

Yes using queue seems to be a better solution

• Reply • Share ›

Hitesh Saini → Sarthak Garg • 6 months ago

thank you..:)

• Reply • Share ›

Yash Jain → Hitesh Saini • 6 months ago

Yes nice one!!

• Reply • Share ›

numid • 7 months ago

one solution can be to maintain a queue,

1.Traverse the linked list horizontally and if any node has child then put it in the queue.
2.When the list can no longer be traversed, then pop an element from queue
3.Repeat the steps 1 and 2 till queue becomes empty and list reaches the end.

2 • Reply • Share ›

Mr. Lazy • 7 months ago

Iterative Solution... TC: O(N) SC: O(N)
http://ideone.com/sbBu46

• Reply • Share ›

Rachit Nagdev • 7 months ago

Using queue (of pointers), (drawback: O(n) Extra Space):

```
void flattenList(struct node *head)
{
    struct node* tail = NULL;
    struct node* temp = NULL;
    q.push(head);
    while(!q.empty())
    {
        temp = q.front();
        q.pop();
        while(temp != NULL)
        {
                if(temp->child != NULL)
                {
                        q.push(temp->child);
                        temp->child = NULL;
                }
                if(tail != NULL) tail->next = temp;
                tail = temp;
                temp = temp->next;
        }
    }
}
```

• Reply • Share ›

Rachit Nagdev • 7 months ago

simple and compact:

```
void flattenList(struct node *head)
{
    struct node* tail = head;
    while(head != NULL)
    {
        while(tail->next != NULL) tail = tail->next;
        if(head->child != NULL)
        {
                tail->next = head->child;
                head->child = NULL;
        }
        head = head->next;
    }
}
```

• Reply • Share ›

soumyajit bhattacharyay • 7 months ago

simple bfs traversal will solve the problem!!

here is the c code:-

http://ideone.com/RT3DFz

• Reply • Share ›

Tarun • 7 months ago

Very Very small code for the Flatten function.

Same running time and no new nodes created.

```
struct node* flattenList(struct node *head)

{

struct node *i, *j, *k;
i=head;
j=head;
k=head;

while(k!=NULL)
{

while(j->next!=NULL)

j=j->next;

j->next=k->child;

k=k->next;

}

return i;
}
```

1 • Reply • Share ›

sup dig • 9 months ago

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
struct node *child;
};

void printlist(struct node *root)
{
while(root!=NULL)
{
printf("%d ",root->data);
root=root->next;
}
printf("\n");
}
```

see more

• Reply • Share ›

thevagabond85 • a year ago

Here's my QUEUE based implementation (tested on given example)

```
void flattenList(struct node *head)
{
/*Base case*/
if (head == NULL)
return;

queue<node*> Q;
while(head->next != NULL || head->child != NULL || !Q.empty()){
if(head->child){
Q.push(head->child);
}

if(head->next == NULL && !Q.empty()){
head->next = Q.front();
Q.pop();
}

if(head!=NULL)
head=head->next;
}
}
```

• Reply • Share ›

katamaran • a year ago

In the above diagram, if node 20 was the child of node 5, will the algorithm work, when we reach tail = 13 and cur = 5?

• Reply • Share ›

Jasprit • a year ago

Make use of the Queue !!!

```
void flatten(struct node *head)
{
if (head == NULL)
return;

std::queue<node *=""> Q;
```

```
while(head)
{
Q.push(head);
head = head->next;
}

while(! Q.empty() )
{
struct node * pop = Q.front();
Q.pop();
```

see more

• Reply • Share ›

anon • a year ago

```
public static ListNode flatten(ListNode head) {
ListNode newHead = new ListNode(0);
ListNode p = newHead;
Queue<listnode> queue = new LinkedList<listnode>();
while (head != null || !queue.isEmpty()) {
if (head != null) {
if (head.child != null) {
queue.offer(head.child);
head.child = null;
}
p.next = head;
head = head.next;
p = p.next;
} else {
head = queue.poll();
}
}
return newHead.next;
}
```

• Reply • Share ›

Kerem Sahin • a year ago

Recursive implementation:

http://ideone.com/Yzfn6K

• Reply • Share ›

f2qbook • a year ago

we can solve this problem using queue .

1 p=start of top most level

1.p=start of top most level.
2while(p!=NULL){ if(p->child!=NULL){ push_back(q,p->child);} cout<<p->data;
p=p->next;
if(p==NULL&& q.empty!=NULL){p=q.front();q.pop();}
}

this method is working finely.
space=O(number0f_sub_levels);
time=O(n); where n is total number of nodes in whole levels.

4 • Reply • Share ›

lovey f2qbook • a year ago

bt this method require space complexity i.e. a queue

• Reply • Share ›

Rajat Shrivastava • a year ago

Java implementation of above algorithm

http://ideone.com/7UPPlV

• Reply • Share ›

Vishal • a year ago

```
void flattenList(Node *head)
{
/*Base case*/
if (head == NULL)
return;

std::queue<node*> NodeQ;
///NodeQ.push(head);
Node *tmp = head;
while(tmp)
{
if(!tmp->next)
{
tmp->next = NodeQ.front();
NodeQ.pop();
}

if(tmp->child)
NodeQ.push(tmp->child);

tmp = tmp->next;
}
}
```

^ | ˅ • Reply • Share ›

kamran siddique • a year ago

on time success!!!

http://ideone.com/YaJ1Zg

^ | ˅ • Reply • Share ›

Avinash Abhi • a year ago

A correction is needed while (cur != tail)

should be changed to

while ((cur != tail) || (cur->child != NULL))

^ | ˅ • Reply • Share ›

flop • a year ago

while(cur !=NULL)
it is given as while(cur!=tail)

2 ^ | ˅ • Reply • Share ›

np ➦ flop • a year ago

i think it should be while(cur !=NULL) anybody please explain

^ | ˅ • Reply • Share ›

geekyprateek ➦ np • a year ago

It is as similar as while(cur !=NULL), just stopped one step before, we dont need to go till we found null, we can stop at the last node which is pointed by tail, as tail->next=NULL;

1 ^ | ˅ • Reply • Share ›

Guest • a year ago

there is no function which tell size ,neither defined in program.hhhow this code run??

^ | ˅ • Reply • Share ›

Pankaj Pal • 2 years ago

beautiful !!

^ | ˅ • Reply • Share ›

logic_bomber • 2 years ago

Nice Algorithm, :) But I think the code can be much simplified as follows ,

```
/*
if (cur->child)
{
```

```
// then append the child at the end of current list
tail->next = cur->child;

// and update the tail to new last node
tmp = cur->child;
while (tmp->next)
tmp = tm/p->next;
tail = tmp;
}

// Change current node
cur = cur->next;*/
as follows

tail->next = cur->child;
while(tail->next)
tail=tail->next;
cur=cur->next;
```

• Reply • Share ›

KeshaShah • 2 years ago

Isn't queue a good idea ? traverse the first row and keep inserting in the queue those elements whose down pointer is not null. And repeat this till list is not empty

4 • Reply • Share ›

codetilldeath → KeshaShah • 2 years ago

Can you please explain how it is a "good" idea? Using a queue requires O(n) space while the given solution does it with O(1) space and same time complexity.

• Reply • Share ›

Nomad → codetilldeath • a year ago

The above approach modifies the original linked list. If we are not allowed to do that, I think using a Q would be a better approach.

2 • Reply • Share ›

typing.. → Nomad • a year ago

the question is about modifng the LL, i.e. flatten the list. It is not traversing the list only.. so this is best approach..

1 • Reply • Share ›

Rudra → KeshaShah • 2 years ago

great idea man!

• Reply • Share ›

Just-a-Beginner Rudra • a year ago

Hahahaha!!!

• Reply • Share ›

Guest Rudra • 2 years ago

Its woman here :P

• Reply • Share ›

Load more comments

Subscribe Add Disqus to your site Add Disqus Add Privacy