Alternating split of a given Singly Linked List

Write a function AlternatingSplit() that takes one list and divides up its nodes to make two smaller lists 'a' and 'b'. The sublists should be made from alternating elements in the original list. So if the original list is 0->1->0->1->0->1 then one sublist should be 0->0->0 and the other should be 1->1->1.

Method 1(Simple)
The simplest approach iterates over the source list and pull nodes off the source and alternately put them at the front (or beginning) of 'a' and b'. The only strange part is that the nodes will be in the reverse order that they occurred in the source list. Method 2 inserts the node at the end by keeping track of last node in sublists.

```
/*Program to alternatively split a linked list into two halves */
#include<stdio.h>
#include<stdlib.h>
#include<assert.h>

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

/* pull off the front node of the source and put it in dest */
void MoveNode(struct node** destRef, struct node** sourceRef) ;

/* Given the source list, split its nodes into two shorter lists.
  If we number the elements 0, 1, 2, ... then all the even elements
  should go in the first list, and all the odd elements in the second.
  The elements in the new lists may be in any order. */
void AlternatingSplit(struct node* source, struct node** aRef,
                              struct node** bRef)
{
  /* split the nodes of source to these 'a' and 'b' lists */
  struct node* a = NULL;
  struct node* b = NULL;

  struct node* current = source;
  while (current != NULL)
  {
    MoveNode(&a, &current); /* Move a node to list 'a' */
    if (current != NULL)
    {
       MoveNode(&b, &current); /* Move a node to list 'b' */
    }
  }
  *aRef = a;
  *bRef = b;
}

/* Take the node from the front of the source, and move it to the front of th
   It is an error to call this with the source list empty.

   Before calling MoveNode():
   source == {1, 2, 3}
   dest == {1, 2, 3}

   Affter calling MoveNode():
   source == {2, 3}
   dest == {1, 1, 2, 3}
*/
void MoveNode(struct node** destRef, struct node** sourceRef)
{
  /* the front source node  */
```

```
  struct node* newNode = *sourceRef;
  assert(newNode != NULL);

  /* Advance the source pointer */
  *sourceRef = newNode->next;

  /* Link the old dest off the new node */
  newNode->next = *destRef;

  /* Move dest to point to the new node */
  *destRef = newNode;
}

/* UTILITY FUNCTIONS */
/* 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);

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

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

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

  /* Let us create a sorted linked list to test the functions
   Created linked list will be 0->1->2->3->4->5 */
  push(&head, 5);
  push(&head, 4);
```

```
  push(&head, 3);
  push(&head, 2);
  push(&head, 1);
  push(&head, 0);

  printf("\n Original linked List:  ");
  printList(head);

  /* Remove duplicates from linked list */
  AlternatingSplit(head, &a, &b);

  printf("\n Resultant Linked List 'a' ");
  printList(a);

  printf("\n Resultant Linked List 'b' ");
  printList(b);

  getchar();
  return 0;
}
```

Time Complexity: O(n) where n is number of node in the given linked list.

Method 2(Using Dummy Nodes)
Here is an alternative approach which builds the sub-lists in the same order as the source list. The code uses a temporary dummy header nodes for the ‘a’ and ‘b’ lists as they are being built. Each sublist has a “tail” pointer which points to its current last node — that way new nodes can be appended to the end of each list easily. The dummy nodes give the tail pointers something to point to initially. The dummy nodes are efficient in this case because they are temporary and allocated in the stack. Alternately, local “reference pointers” (which always points to the last pointer in the list instead of to the last node) could be used to avoid Dummy nodes.

```
void AlternatingSplit(struct node* source, struct node** aRef,
                      struct node** bRef)
{
  struct node aDummy;
  struct node* aTail = &aDummy; /* points to the last node in 'a' */
  struct node bDummy;
  struct node* bTail = &bDummy; /* points to the last node in 'b' */
  struct node* current = source;
  aDummy.next = NULL;
  bDummy.next = NULL;
  while (current != NULL)
  {
    MoveNode(&(aTail->next), &current); /* add at 'a' tail */
    aTail = aTail->next; /* advance the 'a' tail */
    if (current != NULL)
    {
      MoveNode(&(bTail->next), &current);
      bTail = bTail->next;
```

```
    }
  }
  *aRef = aDummy.next;
  *bRef = bDummy.next;
}
```

Time Complexity: O(n) where n is number of node in the given linked list.

Source: http://cslibrary.stanford.edu/105/LinkedListProblems.pdf

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

Related Topics:

- Clone a linked list with next and random pointer | Set 2
- Given a linked list of line segments, remove middle points
- Construct a Maximum Sum Linked List out of two Sorted Linked Lists having some Common nodes
- Given a linked list, reverse alternate nodes and append at the end
- Pairwise swap elements of a given linked list by changing links
- Self Organizing List | Set 1 (Introduction)
- Merge a linked list into another linked list at alternate positions
- QuickSort on Singly Linked List

Writing code in comment? Please use **ideone.com** and share the link here.

Swasti Gupta · 20 days ago

```
void splitAlternateNodes(list **head, list **head1, list **head2)

{

list *current = *head;

if(*head == NULL || (*head)->next == NULL)

{

*head1 = *head;
```

```
*head2 = NULL;

return ;

}

*head1= current; *head2 = current->next;

list *temp1 = *head1, *temp2 = *head2;
```

see more

• Reply • Share ›

Aryan Parker • 23 days ago

A very simple method - here's what I have done to solve the problem, I have taken two global pointers start2, start3 for the new lists.
split func splits the link list
Please comment if you find any bug in my code.

```
void split(struct node*start1){
struct node*temp;
temp=start1;
while(true){
if(start2==NULL)start2=beg(start2,temp->data);
else start2=end(start2,temp->data);
if(temp->next)
{
temp=temp->next;
if(start3==NULL)start3=beg(start3,temp->data);
else start3=end(start3,temp->data);

}
else break;

if(temp->next)temp=temp->next;else break;
}

}
```

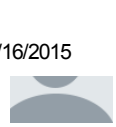
• Reply • Share ›

Aryan Parker ➚ Aryan Parker • 23 days ago

here's the link to full working code-
https://ideone.com/Q97rxx

• Reply • Share ›

Utkarsh Mishra • a month ago



To split into k equal intervals :

```
void split(node *head,int k,node** p[100]) //k is number of interval and
{ //*p[i]is head of respective lists
int i,j;
node *loop[100];
for(i=0;i<k;i++) {="" loop[i]="head;" head="head-">link;
}
node* a[100];
node* temp;
for(i=0;i<1;i++)
{
temp=(node*)malloc(sizeof(node));
temp->data=loop[i]->data;
temp->link=NULL;
*p[i]=temp;
a[i]=*p[i];
}
for(i=0;i<k;i++) {="" while(loop[i]!="NULL)" {="" temp="(node*)malloc(sizeof(node));" temp-
```

see more

• Reply • Share ›

Utkarsh Mishra • a month ago

```
void split(node *loop,node **head1,node **head2)
{
int i=0;
node* temp1=(node*)malloc(sizeof(node));temp1->link=NULL;
node* temp2=(node*)malloc(sizeof(node));temp2->link=NULL;
node *l1=temp1;
node *l2=temp2;
node *temp;
while(loop!=NULL)
{
temp=(node*)malloc(sizeof(node));
temp->data=loop->data;
temp->link=NULL;
if(i%2==0)
{
l1->link=temp;
l1=temp;
}
```

see more

• Reply • Share ›

uday • a month ago

Use two pointers to traverse the list . first pointer move by one count and the second pointer move by two count and swap the data until second pointer next or next to next is NULL

here is the code :

```
AlternateSplit(struct node* head)
{
struct node *ptr1 = head;
struct node *ptr2 = head;

while((NULL != ptr2->next) && (NULL != ptr2->next->next))
{
ptr1 = ptr1->next;
ptr2 = ptr2->next->next;
swap(&ptr1->data, &ptr2->data);
}

/*at the end ptr->next will be the starting of second list and head is anyway starting of first list */
return ptr1->next;
}
```

• Reply • Share ›

thevagabond85 • a month ago

In method 1 , there's no need of those temporary nodes a and b, instead we can go like this :

```
void AlternatingSplit(struct node* source, struct node** aRef, struct node** bRef){

Node current = source;
while(current != NULL){
MoveNode(aRef, & current);
if(current != NULL){
MoveNode(bRef, & current);
}
}
}
```

• Reply • Share ›

wolf • 2 months ago

MoveNode(&(bTail->next), ¤t)

what does &(bTail->next) is passing and how does it works?? plz help

• Reply • Share ›

Stack ➚ wolf • a month ago

Say nodes are A->B->C

Initially tail points to dummy so when we pass &(tail->next) we pass the address of the next part of the dummy in the first case. In the mov_node function new_node->next=*dest_ref (*dest_ref=NULL since dummy.next=NULL and therefore tail->next=NULL) so new_node->next=NULL and *dest_ref will point to the new_node. Now the list is dummy->A(tail points at A) and the original list is B->C. Now when we pass tail->next we actually pass the address of A->next and B is added to A and B now becomes the tail.

• Reply • Share ›

Ashish P Magar • 2 months ago

Do we need to retain original list?
If not, what about following code:

```
headA = head;
headB= head->next;
currentA = headA;
currentB = headB;
while(currentA != NULL || currentB !=NULL)
{
currerntA->next = currentB->next;
currentA = currentA->next;
if (currentA==NULL) break; //corner case, for even number of elements
currentB->next = currentA->next;
currentB = currentB->next;
}
```

//this code will change the links only and data will be at same memory locations and we will get two sublists of alternate elements in the list

---

see more

• Reply • Share ›

ANKIT SINGH • 2 months ago

Can anyone see and tell me if there is any problem with the given code.
http://ideone.com/HxJ1nr
This code is not using any dummy variable and is giving the splitted linked list in the required order.

• Reply • Share ›

Antariksh Srivastava • 4 months ago

I have taken two pointers which will run simultaneously.

One addtional space is there (*sec) to hold the starting address of the second list. (used while printing the list)

```
void deleteAlternate(struct node *root){

struct node *temp1=root;
struct node *temp2,*sec;

if(temp1->link == NULL) return;

sec=temp1->link;
temp2=sec;

while(temp2->link!=NULL) {
temp1->link=temp2->link;
temp1=temp2->link;

if(temp1->link!=NULL){
```

see more

• Reply • Share ›

pleasereply • 5 months ago

why doesn't this work

```
void alter(struct node *u1)

{

struct node *u2=u1;

struct node *u3=u1->next;

while(u2->next->next!=NULL && u3->next!=NULL)

{

u3->next=u3->next->next;

u2->next=u2->next->next;

u2=u2->next;

u3=u3->next;

}
```

• Reply • Share ›

Naval ➚ pleasereply • 3 months ago

suppose according to your code suppose in link-list there is 3 node is present then u1 and u2 pointing at 1st node of your list and u3 is pointing at 2nd node

in second iteration u2->next=u2->next->next so u2 is pointing at 3rd position node and u3->next=u3->next->u3->next; so u3 becomes null and if condition is cheacked then u3 is NULL but u3->next is pointing illegal position (now u3 is dangling pointer) and also u2 is at last position(3rd node) of yur list so u2->next->next is also dangling pointer so your condition is never becomes false and result is unpredictable(unexpected behaviour) behaviour will be .show

• Reply • Share ›

usualraj • 5 months ago

The function below returns the head of the second split list. Could you review this..

```
struct node* alternateSplit(struct node* firstHead) {
if(firstHead == NULL || firstHead->next == NULL)
return NULL;

struct node* secondHead = firstHead->next;
struct node* cur1 = firstHead;
struct node* cur2 = secondHead;

while(cur1->next != NULL && cur2->next != NULL) {
cur1->next = cur1->next->next;
cur2->next = cur2->next->next;

cur1 = cur1->next;
if(cur2->next != NULL)
cur2 = cur2->next;
}
cur1->next = NULL;
cur2->next = NULL;
return secondHead;
}
```

1 • Reply • Share ›

tintin • 6 months ago

Implementation in Java

This is same as what devakar verma did

```
public static void split(LinkedList list, LinkedList firstList, LinkedList secondList) {

Node firstCurrent = list.getHead();

Node secondCurrent = list.getHead().next;

while (firstCurrent != null && firstCurrent.next != null) {
```

```
firstList.insert(firstCurrent.data);

firstCurrent = firstCurrent.next.next;

}

if(firstCurrent != null) {

firstList.insert(firstCurrent.data);
```

see more

• Reply • Share ›

devakar verma • 7 months ago

```
//it would be much simpler than others.

void split_alter(struct node *head,struct node **a,struct node **b)

{

struct node *temp1=head;

struct node *temp2=head->next;

while(temp1!=NULL && temp1->next!=NULL)

{

insert_end(a,temp1->data); //insert at the end of 1st sublist

temp1=temp1->next->next;

}

if(temp1)
```

see more

• Reply • Share ›

parit13 • 7 months ago

```
/*using c++ stl ,but have some bug in my code , need suggestion */

#include <iostream>
#include <list>
#include <algorithm>
#include <stdlib.h>
using namespace std;

int main()
```

```
int main()
{
list<int> mylist,a,b;
list<int>::iterator it;
mylist.push(1);
mylist.push(2);
mylist.push(3);
mylist.push(4);
mylist.push(5);
for(it=mylist.begin();it!=mylist.end();it+2)
```

see more

• Reply • Share ›

typing.. • 7 months ago

we can simply consider it as two list. one with starting head and other with starting head->next and then proceed as in last post of deleting alternate nodes of a linked list... Here is code ::

```
void AlternatingSplit(struct node* head, struct node** a,

struct node** b)

{

if(head!=NULL && head->next==NULL)

*a=head;

else if(head && head->next)

{

struct node *h1,*h2;

*a=head;
```

see more

• Reply • Share ›

Guest • 7 months ago

another method: spliting using Recursive Function

```
#include<stdio.h>
#include<stdlib.h>

struct nodeLinkedList {
int nValue;
struct nodeLinkedList *path;
```

```
struct nodeLinkedList *path;
};

void alternateSplit (struct nodeLinkedList **node, struct nodeLinkedList **list1, struct
nodeLinkedList **list2, int place);

int main() {

int newEntry = 1, response, j, nodeCount, nodeValue;

struct nodeLinkedList *node = NULL, *startNode, *previousNode, *sublist1 = NULL, *sublist2 =
NULL;;

while (newEntry == 1) {
```

see more

• Reply • Share ›

Sravya • 8 months ago

```
struct node *altSplit(struct node *head)
{
if(head==NULL || head->next==NULL)
{
return;
}
else
{
struct node *head1 = head->next;
struct node *first = head;
struct node *second = head1;
int flag = 0;
while(flag!=1)
{
if(second->next!=NULL)
{
first->next = second->next;
first = first->next;
```

see more

• Reply • Share ›

Sravya ➦ Sravya • 8 months ago

sorry.. der is a mistake.. in the first else, after flag = 1; there will be a break statement to avoid remaining code to execute

• Reply • Share ›

Kim Jong-il ➦ Sravya • 7 months ago

there is also a mistake in logic, If there is head is not NULL and head->next is NULL then you can not simply return, It means that their is at least 1 node available in the list so that should be assigned to the first list.

• Reply • Share ›

Abhi • 8 months ago

```
#include<stdio.h> /*I think time complexity here is O(n/2)*/

#include<stdlib.h>

typedef struct Node

{

int data;

struct Node* next;

}Node;

void InsertNode(Node** head,int new_data)

{

Node* new_node=(Node*)malloc(sizeof(Node));

new_node->data=new_data;
```

see more

• Reply • Share ›

Heracles • 9 months ago

How about this?

```
public void alternatingSplit(LinkedList s, LinkedList a, LinkedList b)
{
Node oldfirst = s.first;
int count = 0;
while(oldfirst!=null)
{
count++;
if(count%2!=0)
{
Node x = new Node();
x.item = oldfirst.item;
x.next = a.first;
a.first = x;
```

```
oldfirst = oldfirst.next;
}
else
{
Node y = new Node();
y.item = oldfirst.item;
y.next = b.first;
b.first = y;
oldfirst = oldfirst.next;
}
}
}
```

• Reply • Share ›

Nishtha Rai • 10 months ago

Question is to split the list into two parts. We are supposed to modify the original list by modifying its next pointers. Do we have to use double pointers for this. Here's a code that splits the list using only two pointers in a recursive manner.

Call from main : alternate_split(head1, head2);
where head1 = Original head
and head2 = head->next

```
void alternate_split(node *head1, node *head2)
{
if (!head1 || !head2) {
return;
}

if (head1->next) {
head1->next = head1->next->next;
}
if (head2->next) {
head2->next = head2->next->next;
}

alternate_split(head1->next, head2->next);
}
```

15 • Reply • Share ›

Guest ➚ Nishtha Rai • 2 months ago

Why shouldn't double pointers be used here?

• Reply • Share ›

<HoldOnLife!#> ➚ Nishtha Rai • 8 months ago

a=head
b=head->next according to given question right?

• Reply • Share ›

Guest Nishtha Rai • 8 months ago

how are u assigning null pointer at the end of both linked list. there is need to assign null pointer at end. so
if(head1->next ==NULL)
head1->next=NULL;
if(head2->next==NULL)
head2->next=NULL;
should be there.

good solution. :)

• Reply • Share ›

Shadeslayer Nishtha Rai • 8 months ago

Simple and elegant. Works like a charm :)

@GeeksforGeeks this could be a good addition to the post.

1 • Reply • Share ›

Khirod Kant Naik Nishtha Rai • 9 months ago

Can be shortened further

```
void alternateSplit(struct node *head1, struct node *head2)
{
    if (head1 && head2) {
        head1->next = head2->next;
        head2->next = (head2->next)?(head2->next)->next:NULL;
        alternateSplit(head1->next, head2->next);
    }
}
```

Nice method though !!

1 • Reply • Share ›

ANA Nishtha Rai • 9 months ago

if (head1->next) {
head1->next = head1->next->next;
}

instead of this we can directly write
head1->next = head2->next

• Reply • Share ›

Karshit Jaiswal ➚ Nishtha Rai • 10 months ago

@GeeksforGeeks as per the problem statement, I think this approach is much better and simple.

1 ▲ | ▼ • Reply • Share ›

Guest ➚ Nishtha Rai • 10 months ago

How are u preserving the heads for the two fresh link lists created in main.
Either the function should return two pointer (somehow using pair<node*,node*> etc) or the implementation will lose one of the linked list in the process of splitting.

▲ | ▼ • Reply • Share ›

popeye ➚ Guest • 10 months ago

She has stored head->next in another node* variable head2 before passing it to this function. That way both heads can be preserved.

▲ | ▼ • Reply • Share ›

Karshit Jaiswal ➚ popeye • 9 months ago

exactly correct..!!

▲ | ▼ • Reply • Share ›

thatsme ➚ Karshit Jaiswal • 9 months ago

sale tu yahan kya kar rha hai?

▲ | ▼ • Reply • Share ›

Karshit Jaiswal ➚ thatsme • 9 months ago

@jadoo
yaar mjhe kuch samjh ni aa ra tha so comment me accha solution dhoond ra tha.. :D

▲ | ▼ • Reply • Share ›

Ateet Kakkar • a year ago

//In main(), alt_split(head, &head1, &head2, 0) will be passed where head1 and head2 are initialized to NULL. Complexity is O(n).

```
void alt_split(struct node *curr, struct node **pfirst1, struct node **pfirst2, int x){
if(!curr)
return;
alt_split(curr->next, pfirst1, pfirst2, 1-x);
if(x){
curr->next = *pfirst2;
*pfirst2 = curr;
}
```

```
else{
curr->next = *pfirst1;
*pfirst1 = curr;
}
}
```

• Reply • Share ›

Mohan Rajoria • a year ago

An alternate way is the same as deleting alternative nodes. We can add deleted nodes to new list, by this way we'll get solution for both the problem 1.Delete alternative nodes.
2. Split alternative nodes into two list.

1 • Reply • Share ›

Ankur Teotia • a year ago

an easier way to do this would be to traverse the list and have a counter integer initialized to zero which would increment whenever a node would get traversed.

now if the counter is even then put the node in one list and if it is odd , put it in another list.

the complete code -> http://ideone.com/CEN0ss

here's the code snippet of the modifield alternating split function.

struct node* b1 = NULL;

struct node* a1= NULL;

void AlternatingSplit(struct node* source, struct node** aRef,

struct node** bRef)

{

/* split the nodes of source to these 'a' and 'b' lists */

see more

• Reply • Share ›

popeye ➦ Ankur Teotia • 10 months ago

You are making new nodes for the lists. The existing nodes must be moved into the 2 lists

• Reply • Share ›

Vishal Kumar Tiwari • a year ago

My below post will give the list as required but having the reference of the original list...
So below program is just to print the expected output...

no new memory allocations...

• Reply • Share ›

Vishal Kumar Tiwari • a year ago

```
void AlternatingSplit(Node *head, Node **evenlist,Node **oddlist)
{
Node *temp = head;
Node *pevenlist = *evenlist;
Node *poddlist = *oddlist;
int i =0;
while(temp)
{
if(i%2 == 0)
{
if(pevenlist){
pevenlist->next= temp;
pevenlist = pevenlist->next;}
else{

pevenlist = temp;
*evenlist = pevenlist;
}
```

see more

• Reply • Share ›

Himanshu Dagar • a year ago

whole source code from dummy variable concept is here(at below link)

http://ideone.com/Xw7Wh1

• Reply • Share ›

bhavesh • a year ago

```
#include<stdio.h>

#include<stdlib.h>

#include<assert.h>

/* Link list node */

struct node

{

int data;
```

```
struct node* next;

};

/* pull off the front node of the source and put it in dest */

void MoveNode(struct node** destRef, struct node** sourceRef) ;
```

see more

• Reply • Share ›

Shikha Gupta • a year ago

http://ideone.com/JvD1UP
can this be simplified further

• Reply • Share ›

Vikash876 • a year ago

```
node * list_split_alternate(node *head)
{
node *sav, *retnode = head->next;
while(head->next!=NULL)
{
sav = head->next;
head->next = head->next->next;
head = sav;
}
sav->next = NULL;
return retnode;
}
```

• Reply • Share ›

ankit • 2 years ago

```
#include <stdio.h>
#include <stdlib.h>

struct treeNode
{
    int val;
    struct treeNode *next;

};
struct treeNode *root=NULL;

struct treeNode *a=NULL;
```

```
struct treeNode *b=NULL;

struct  treeNode*  getNode()
{
    return (struct treeNode*)malloc(sizeof(struct treeNode));
}
```

see more

• Reply • Share ›

GP • 2 years ago

[sourcecode language="C"]

/* simple approach*/

void alternatingSplit(struct node** head_ref,struct node** head1,struct node** head2)

{

if(*head_ref==NULL||(*head_ref)->next==NULL)

return;

struct node* current=*head_ref;

struct node* nNext,*next;

*head1=*head_ref;

*head2=(*head_ref)->next;

while(current!=NULL&&nNext!=NULL)

see more

• Reply • Share ›

Load more comments

-
-
- Interview Experiences
- Advanced Data Structures
- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Mathematical Algorithms
- Recursion
- Geometric Algorithms
-

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

- # Recent Comments

  - Baba

    Traverse the tree in inorder fashion and insert...

    Populate Inorder Successor for all nodes · 54 minutes ago

  - creeping_death

    Ruby solution, slightly different, easier to...

    Longest Even Length Substring such that Sum of First and Second Half is same · 1 hour ago

  - darkprotocol

    Anyone round-1 4th question?

    Amazon Interview | Set 121 (On-Campus for SDE-1) · 1 hour ago

  - S.Nkm

    Would a simple answer involving the Unix epoch...

    Find number of days between two given dates · 1 hour ago

  - Shomina

    Found Real Job INterview Questions Here...

    Flipkart Interview Experience | Set 20 (For SDE-II) · 2 hours ago

  - Shimona

    Found Real Job INterview Questions here...

    Myntra Interview Experience | Set 4 (For Senior Software Engineer) · 2 hours ago