1. **Finding middle element in a linked list**

Given a singly linked list of **N** nodes. The task is to find the middle of the linked list. For example, if given linked list is 1->2->3->4->5 then the output should be 3.  
If there are even nodes, then there would be two middle nodes, we need to print the second middle element. For example, if given linked list is 1->2->3->4->5->6 then the output should be 4.

**Example 1:**

**Input:**

LinkedList: 1->2->3->4->5

**Output:** 3

**Example 2:**

**Input:**

LinkedList: 2->4->6->7->5->1

**Output:** 7

**Your Task:**  
The task is to complete the function **getMiddle**() which takes a head reference as the only argument and should return the data at the middle node of the linked list.

**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(1).

**Constraints:**  
1 <= N <= 5000

Code:

#include <bits/stdc++.h>

using namespace std;

struct Node

{

int data;

struct Node\* next;

Node(int x){

data = x;

next = NULL;

}

};

void printList(Node\* node)

{

while (node != NULL) {

cout << node->data <<" ";

node = node->next;

}

cout<<"\n";

}

/\* Function to get the middle of the linked list\*/

int getMiddle(Node \*head);

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int data;

cin>>data;

struct Node \*head = new Node(data);

struct Node \*tail = head;

for (int i = 0; i < n-1; ++i)

{

cin>>data;

tail->next = new Node(data);

tail = tail->next;

}

cout<<getMiddle(head)<<endl;

}

return 0;

}

// } Driver Code Ends

/\* Link list Node

struct Node {

int data;

Node\* next;

Node(int x){

data = x;

next = NULL;

}

}; \*/

/\* Should return data of middle node. If linked list is empty, then -1\*/

int getMiddle(Node \*head)

{

int cnt=0;

Node\*curr;

for(curr=head;curr!=NULL;curr=curr->next)cnt++;

curr=head;

for(int i=0;i<cnt/2;i++)

curr=curr->next;

return curr->data;

// Your code here

}

2. **Reverse a linked list**

Given a linked list of **N**nodes. The task is to reverse this list.

**Example 1:**

**Input:**

LinkedList: 1->2->3->4->5->6

**Output:** 6 5 4 3 2 1

**Explanation:** After reversing the list,

elements are 6->5->4->3->2->1.

**Example 2:**

**Input:**

LinkedList: 2->7->8->9->10

**Output:** 10 9 8 7 2

**Explanation:** After reversing the list,

elements are 10->9->8->7->2.

**Your Task:**  
The task is to complete the function **reverseList**() with head reference as the only argument and should return new head after reversing the list.

**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(1).

**Constraints:**  
1 <= N <= 104

Code:

//Initial Template for C++

// C program to find n'th Node in linked list

#include <stdio.h>

#include <stdlib.h>

#include<iostream>

using namespace std;

/\* Link list Node \*/

struct Node {

int data;

struct Node \*next;

Node(int x)

{

data = x;

next = NULL;

}

};

/\* Function to get the middle of the linked list\*/

struct Node \*reverseList(struct Node \*head);

void printList(struct Node \*head)

{

struct Node \*temp = head;

while (temp != NULL)

{

printf("%d ", temp->data);

temp = temp->next;

}

}

/\* Driver program to test above function\*/

int main()

{

int T,n,l,firstdata;

cin>>T;

while(T--)

{

struct Node \*head = NULL, \*tail = NULL;

cin>>n;

cin>>firstdata;

head = new Node(firstdata);

tail = head;

for (int i=1; i<n; i++)

{

cin>>l;

tail->next = new Node(l);

tail = tail->next;

}

head = reverseList(head);

printList(head);

cout << endl;

}

return 0;

}

// } Driver Code Ends

/\* Linked List Node structure:

struct Node

{

int data;

struct Node \*next;

}

\*/

// Should reverse list and return new head.

struct Node\* reverseList(struct Node \*head)

{

struct Node\*temp=head;

struct Node\*prev=NULL;

struct Node\*ahead=temp->next;

while(temp!=NULL){

ahead=temp->next;

temp->next=prev;

prev=temp;

temp=ahead;

}

return prev;// code here

// return head of reversed list

}

3. **Rotate a Linked List**

Given a singly linked list of size **N**. The task is to rotate the linked list counter-clockwise by **k** nodes, where k is a given positive integer smaller than or equal to length of the linked list.

**Example 1:**

**Input:**

N = 8

value[] = {1,2,3,4,5,6,7,8}

k = 4

**Output:** 5 6 7 8 1 2 3 4

**Explanation:**

**Example 2:**

**Input:**

N = 5

value[] = {2,4,7,8,9}

k = 3

**Output:** 8 9 2 4 7

**Explanation:**

**Your Task:**  
The task is to complete the function **rotate**() which takes a **head reference** as the **first argument and** **k** as the **second argument**. The **printing**is done **automatically**by the **driver code**.

**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(1).

**Constraints:**  
1 <= N <= 103  
1 <= k <= 103

Code:

#include <bits/stdc++.h>

using namespace std;

struct Node {

int data;

struct Node \*next;

Node(int x) {

data = x;

next = NULL;

}

};

Node \*rotate(struct Node \*head, int k);

void printList(Node \*n)

{

while (n != NULL)

{

cout<< n->data << " ";

n = n->next;

}

cout<< endl;

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n, val, k;

cin>>n;

cin>> val;

Node \*head = new Node(val);

Node \*tail = head;

for(int i=0; i<n-1; i++)

{

cin>> val;

tail->next = new Node(val);

tail = tail->next;

}

cin>> k;

head = rotate(head,k);

printList(head);

}

return 1;

}

// } Driver Code Ends

/\*

struct Node {

int data;

struct Node \*next;

Node(int x) {

data = x;

next = NULL;

}

};

\*/

// This function should rotate list counter-clockwise

// by k and return new head (if changed)

Node\* rotate(Node\* head, int k)

{

Node\* temp=head;

//make linked list circular

while(temp->next!=NULL)

temp=temp->next;

temp->next=head;

//move temp at (k)th node

temp=head;

for(int i=0;i<k-1;i++)

temp=temp->next;

//make k+1 th node as new head

head=temp->next;

//make kth node as last node of new linked list

temp->next=NULL;

return head;

// Your code here

}

4. **Reverse a Linked List in groups of given size.**

Given a linked list of size **N**. The task is to reverse every **k** nodes (where k is an input to the function) in the linked list.

**Example 1:**

**Input:**

LinkedList: 1->2->2->4->5->6->7->8

K = 4

**Output:** 4 2 2 1 8 7 6 5

**Example 2:**

**Input:**

LinkedList: 1->2->3->4->5

K = 3

**Output:** 3 2 1 5 4

**Your Task:**  
The task is to complete the function **reverse**() which should reverse the linked list in group of size **k**and return the head of the modified linked list.

Code:

#include<bits/stdc++.h>

using namespace std;

/\* Link list node \*/

struct node \*reverse (struct node \*head, int k);

struct node

{

int data;

struct node\* next;

node(int x){

data = x;

next = NULL;

}

};

/\* UTILITY FUNCTIONS \*/

/\* Function to push a node \*/

/\* Function to print linked list \*/

void printList(struct node \*node)

{

while (node != NULL)

{

printf("%d ", node->data);

node = node->next;

}

printf("\n");

}

/\* Drier program to test above function\*/

int main(void)

{

/\* Start with the empty list \*/

int t;

cin>>t;

while(t--)

{

struct node\* head = NULL;

struct node\* temp = NULL;

int n;

cin >> n;

for(int i = 0;i<n;i++){

int value;

cin >> value;

if(i == 0)

{

head = new node(value);

temp = head;

}

else

{

temp->next = new node(value);

temp = temp->next;

}

}

int k;

cin>>k;

head = reverse(head, k);

printList(head);

}

return(0);

}

// } Driver Code Ends

/\*

Reverse a linked list

The input list will have at least one element

Return the node which points to the head of the new LinkedList

Node is defined as

struct node

{

int data;

struct node\* next;

node(int x){

data = x;

next = NULL;

}

}\*head;

\*/

struct node \*reverse (struct node \*head, int k)

{

// Complete this method

node\* curr=head,\*next=NULL,\*prev=NULL;

int count=0;

while(curr!=NULL && count<k)

{

next=curr->next;

curr->next=prev;

prev=curr;

curr=next;

count++;

}

if(next!=NULL)

{

node\* resthead=reverse(next,k);

head->next=resthead;

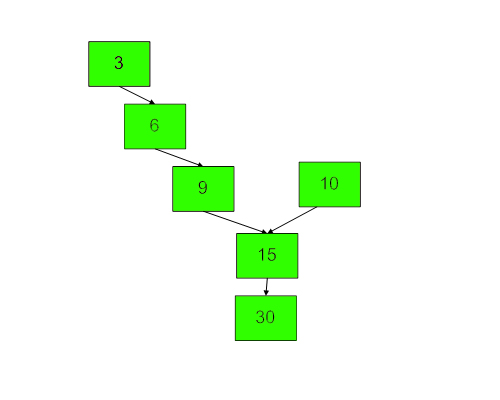
}

return prev;

}

5. **Intersection Point in Y Shapped Linked Lists**

Given two singly linked lists of size **N** and **M,**write a program to get the point where two linked lists intersect each other.

  
Above diagram shows an example with two linked list having 15 as intersection point.

**Example 1:**

**Input:**

LinkList1 = {10,20,5,10}

LinkList2 = {30,40,50,5,10}

**Output:** 5

**Explanation:**The point of intersection of

two linked list is 5, means both of them

get linked (intersects) with each other

at node whose value is 5.

**Your Task:**  
The task is to complete the function **intersetPoint**() which finds the point of intersection of two linked list. The function should return data value of a node where two linked lists merge. If linked list do not merge at any point, then it should return **-1**.

**Challenge**: Try to solve the problem without using any extra space.

**Expected Time Complexity:** O(N+M)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
1 <= N <= 100  
1 <= value <= 1000

Code:

#include<iostream>

#include<stdio.h>

using namespace std;

/\* Link list Node \*/

struct Node

{

int data;

struct Node \*next;

Node(int x)

{

data = x;

next = NULL;

}

};

int intersectPoint(struct Node\* head1, struct Node\* head2);

Node\* inputList(int size)

{

if(size==0) return NULL;

int val;

cin>> val;

Node \*head = new Node(val);

Node \*tail = head;

for(int i=0; i<size-1; i++)

{

cin>>val;

tail->next = new Node(val);

tail = tail->next;

}

return head;

}

/\* Driver program to test above function\*/

int main()

{

int T,n1,n2,n3;

cin>>T;

while(T--)

{

cin>>n1>>n2>>n3;

Node\* head1 = inputList(n1);

Node\* head2 = inputList(n2);

Node\* common = inputList(n3);

Node\* temp = head1;

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

temp = temp->next;

if(temp!=NULL) temp->next = common;

temp = head2;

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

temp = temp->next;

if(temp!=NULL) temp->next = common;

cout << intersectPoint(head1, head2) << endl;

}

return 0;

}

// } Driver Code Ends

/\* Linked List Node

struct Node {

int data;

struct Node \*next;

Node(int x) {

data = x;

next = NULL;

}

}; \*/

/\* Should return data of intersection point of two linked

lists head1 and head2.

If there is no intersecting point, then return -1. \*/

//find difference in number of nodes of both the linked list

int intersectPoint(Node\* head1, Node\* head2){

Node \*temp = head1;

while(temp){

temp->data = -1 \* (temp->data +1);

temp = temp->next;

}

temp = head2;

int ans = -1,p=0;

while(temp){

if(temp->data < 0){

p=1;

ans = -1\*temp->data;

break;

}

temp = temp->next;

}

if(p == 1)

return (ans-1);

return ans;

}

6. **Detect Loop in linked list**

Given a linked list of **N** nodes. The task is to check if the the linked list has a loop. Linked list can contain self loop.

**Example 1:**

**Input:**

N = 3

value[] = {1,3,4}

x = 2

**Output:** True

**Explanation:** In above test case N = 3.

The linked list with nodes N = 3 is

given. Then value of x=2 is given which

means last node is connected with xth

node of linked list. Therefore, there

exists a loop.

**Example 2:**

**Input:**

N = 4

value[] = {1,8,3,4}

x = 0

**Output:** False

**Explanation:** For N = 4 ,x = 0 means

then lastNode->next = NULL, then

the Linked list does not contains

any loop.

**Your Task:**  
The task is to complete the function **detectloop**() which contains reference to the head as only argument. This function should return 1 if linked list contains loop, else return 0.

**Expected Time Complexity:** O(N)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
1 <= N <= 104  
1 <= Data on Node <= 103

Code:

// driver code

#include <bits/stdc++.h>

using namespace std;

struct Node

{

int data;

Node\* next;

Node(int val)

{

data = val;

next = NULL;

}

};

void loopHere(Node\* head, Node\* tail, int position)

{

if(position==0) return;

Node\* walk = head;

for(int i=1; i<position; i++)

walk = walk->next;

tail->next = walk;

}

bool detectLoop(Node\* head);

int main()

{

int t;

cin>>t;

while(t--)

{

int n, num;

cin>>n;

Node \*head, \*tail;

cin>> num;

head = tail = new Node(num);

for(int i=0 ; i<n-1 ; i++)

{

cin>> num;

tail->next = new Node(num);

tail = tail->next;

}

int pos;

cin>> pos;

loopHere(head,tail,pos);

if( detectLoop(head) )

cout<< "True\n";

else

cout<< "False\n";

}

return 0;

}

// } Driver Code Ends

/\*

struct Node

{

int data;

struct Node \*next;

Node(int x) {

data = x;

next = NULL;

}

\*/

bool detectLoop(Node\* head)

{

//make 2 pointers - Floyd's cycle finding Algo

Node\* slow=head;

Node\* fast=head;

while(slow && fast && fast->next)

{

slow=slow->next;

fast=fast->next->next;

//if two pointers meet then cycle is detected

if(slow==fast)

return true;

}

return false;

// your code here

}

7. **Remove loop in Linked List**

You are given a linked list of **N** nodes. The task is to remove the loop from the linked list, if present.

**Note**: **C**is the position of the node to which the last node is connected. If it is 0 then no loop.

**Example 1:**

**Input:**

N = 3

value[] = {1,3,4}

C = 2

**Output:** 1

**Explanation:** In the first test case

N = 3.The linked list with nodes

N = 3 is given. Here, x = 2 which

means last node is connected with xth

node of linked list. Therefore, there

exists a loop.

**Example 2:**

**Input:**

N = 4

value[] = {1,8,3,4}

C = 0

**Output:** 1

**Explanation:** N = 4 and x = 0, which

means lastNode->next = NULL, thus

the Linked list does not contains

any loop.

**Your Task:**  
Your task is to complete the function **removeLoop**(). The only argument of the function is head pointer of the linked list. Simply remove the loop in the list (if present) without disconnecting any nodes from the list. The driver code will print **1**if your code is correct.

**Expected time complexity :** O(n)

**Expected auxiliary space :**O(1)

**Constraints:**  
1 <= N <= 104

Code:

// driver code

#include <iostream>

using namespace std;

struct Node

{

int data;

Node\* next;

Node(int val)

{

data = val;

next = NULL;

}

};

void loopHere(Node\* head, Node\* tail, int position)

{

if(position==0) return;

Node\* walk = head;

for(int i=1; i<position; i++)

walk = walk->next;

tail->next = walk;

}

bool isLoop(Node\* head)

{

if(!head) return false;

Node\* fast = head->next;

Node\* slow = head;

while( fast != slow)

{

if( !fast || !fast->next ) return false;

fast=fast->next->next;

slow=slow->next;

}

return true;

}

int length(Node\* head)

{

int ret = 0;

while(head)

{

ret++;

head = head->next;

}

return ret;

}

void removeLoop(Node\* head);

int main()

{

int t;

cin>>t;

while(t--)

{

int n, num;

cin>>n;

Node \*head, \*tail;

cin>> num;

head = tail = new Node(num);

for(int i=0 ; i<n-1 ; i++)

{

cin>> num;

tail->next = new Node(num);

tail = tail->next;

}

int pos;

cin>> pos;

loopHere(head,tail,pos);

removeLoop(head);

if( isLoop(head) || length(head)!=n )

cout<<"0\n";

else

cout<<"1\n";

}

return 0;

}

// } Driver Code Ends

/\*

structure of linked list node:

struct Node

{

int data;

Node\* next;

Node(int val)

{

data = val;

next = NULL;

}

};

\*/

void removeLoop(Node\* head)

{

if(!head) return;

Node\* fast = head->next;

Node\* slow = head;

while( fast != slow )

{

if( !fast || !fast->next ) return;

fast=fast->next->next;

slow=slow->next;

}

int size = 1;

fast = fast->next;

while( fast != slow )

{

size++;

fast = fast->next;

}

slow = head;

fast = head;

for(int i=0; i<size-1; i++)

fast = fast->next;

while( fast->next != slow )

{

fast = fast->next;

slow = slow->next;

}

fast->next = NULL;

}

8. **Nth node from end of linked list**

Given a linked list consisting of **L** nodes and given a number **N**. The task is to find the **N**th node from the end of the linked list.

**Example 1:**

**Input:**

N = 2

LinkedList: 1->2->3->4->5->6->7->8->9

**Output:** 8

**Explanation:** In the first example, there

are 9 nodes in linked list and we need

to find 2nd node from end. 2nd node

from end os 8.

**Example 2:**

**Input:**

N = 5

LinkedList: 10->5->100->5

**Output:** -1

**Explanation:** In the second example, there

are 4 nodes in the linked list and we

need to find 5th from the end. Since 'n'

is more than the number of nodes in the

linked list, the output is -1.

**Your Task:**  
The task is to complete the function **getNthFromLast**() which takes two **arguments**: **reference**to **head and N** and you need to**return Nth**from the end or -1 in case node doesn't exist..

**Note:**  
Try to solve in single traversal.

**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(1).

**Constraints:**  
1 <= L <= 103  
1 <= N <= 103

Code:

// C program to find n'th Node in linked list

#include <stdio.h>

#include <stdlib.h>

#include<iostream>

using namespace std;

/\* Link list Node \*/

struct Node {

int data;

struct Node \*next;

Node(int x) {

data = x;

next = NULL;

}

};

/\* Function to get the nth node from the last of a linked list\*/

int getNthFromLast(struct Node\* head, int n);

/\* Driver program to test above function\*/

int main()

{

int T,i,n,l,k;

cin>>T;

while(T--){

struct Node \*head = NULL, \*tail = NULL;

cin>>n>>k;

int firstdata;

cin>>firstdata;

head = new Node(firstdata);

tail = head;

for(i=1;i<n;i++)

{

cin>>l;

tail->next = new Node(l);

tail = tail->next;

}

cout<<getNthFromLast(head, k)<<endl;

}

return 0;

}// } Driver Code Ends

/\* struct Node {

int data;

struct Node \*next;

Node(int x) {

data = x;

next = NULL;

}

};

\*/

/\* Should return data of n'th node from the end of linked list.

\* head: head of the linked list

\* n: nth node from end to find

\*/

int getNthFromLast(Node \*head, int n)

{

Node\* f=head;

//traversing n-1 nodes first

for(int i=0;i<n-1;i++)

{

f=f->next;

if(f==NULL)

return -1;

}

//then traversing m-n nodes

Node\* ans=head;

while(f->next!=NULL)

{

f=f->next;

ans=ans->next;

}

//ans is at M-N th node from front that is Nth Node back

return ans->data;

// Your code here

}

9. **Flattening a Linked List**

Given a Linked List of size N, where every node represents a linked list and contains two pointers of its type:  
(i) a**next**pointer to the next node,  
(ii) a**bottom** pointer to a linked list where this node is head.

**Note:** The **flattened**list will be printed using the **bottom**pointer **instead**of **next**pointer.

**Example 1:**

**Input:**

5 -> 10 -> 19 -> 28

| | | |

7 20 22 35

| | |

8 50 40

| |

30 45

**Output:**  5-> 7-> 8- > 10 -> 19-> 20->

22-> 28-> 30-> 35-> 40-> 45-> 50.

**Note:** | represents the bottom pointer.

**Your Task:**  
You need to complete the function **flatten()** that takes **head**of the list as **parameter**and **returns**the **root**of **flattened**list. The printing is done by the **driver code**.

**Note**: Try to solve the problem without using any extra space.

**Expected Time Complexity:** O(N\*M)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
0 <= N <= 50  
1 <=**Mi**<= 20  
1 <= Element of linked list <= 103

Code:

#include <bits/stdc++.h>

struct Node{

int data;

struct Node \* next;

struct Node \* bottom;

Node(int x){

data = x;

next = NULL;

bottom = NULL;

}

};

using namespace std;

void printList(Node \*Node)

{

while (Node != NULL)

{

printf("%d ", Node->data);

Node = Node->bottom;

}

}

Node\* flatten (Node\* root);

int main(void) {

int t;

cin>>t;

while(t--){

int n,m,flag=1,flag1=1;

struct Node \* temp=NULL;

struct Node \* head=NULL;

struct Node \* pre=NULL;

struct Node \* tempB=NULL;

struct Node \* preB=NULL;

cin>>n;

int work[n];

for(int i=0;i<n;i++)

cin>>work[i];

for(int i=0;i<n;i++){

m=work[i];

--m;

int data;

scanf("%d",&data);

temp = new Node(data);

temp->next = NULL;

temp->bottom = NULL;

if(flag){

head = temp;

pre = temp;

flag = 0;

flag1 = 1;

}

else{

pre->next = temp;

pre = temp;

flag1 = 1;

}

for(int j=0;j<m;j++){

int temp\_data;

scanf("%d",&temp\_data);

tempB = new Node(temp\_data);

if(flag1){

temp->bottom=tempB;

preB=tempB;

flag1=0;

}

else{

preB->bottom=tempB;

preB=tempB;

}

}

}

Node \*fun = head;

Node \*fun2=head;

Node\* root = flatten(head);

printList(root);

cout<<endl;

}

return 0;

}

// } Driver Code Ends

/\* Node structure used in the program

struct Node{

int data;

struct Node \* next;

struct Node \* bottom;

Node(int x){

data = x;

next = NULL;

bottom = NULL;

}

};

\*/

Node \*merge (Node \*h1,Node \*h2)

{

if(h1==NULL) return(h2);

if(h2==NULL) return(h1);

if(h1->data<h2->data) {

h1->bottom=merge(h1->bottom,h2);

return(h1);

}

else{

h2->bottom=merge(h1,h2->bottom);

return(h2);

}

}

Node \*flatten(Node \*root)

{

Node \*a =root;

Node \*b=root->next;

while(b!=NULL)

{

Node \*c=b->next;

a=merge(a,b);

b=c;

}

return(a);

}

10. **Merge two sorted linked lists**

Given two sorted linked lists consisting of **N and M** nodes respectively. The task is to merge both of the list (in-place) and return head of the merged list.  
**Note:** It is strongly recommended to do merging in-place using **O(1)** extra space.

**Example 1:**

**Input:**

N = 4, M = 3

valueN[] = {5,10,15,40}

valueM[] = {2,3,20}

**Output:** 2 3 5 10 15 20 40

**Explanation:** After merging the two linked

lists, we have merged list as 2, 3, 5,]

10, 15, 20, 40.

**Example 2:**

**Input:**

N = 2, M = 2

valueN[] = {1,1}

valueM[] = {2,4}

**Output:**1 1 2 4

**Explanation:** After merging the given two

linked list , we have 1, 1, 2, 4 as

output.

**Your Task:**  
The task is to complete the function **sortedMerge**() which takes references to the heads of two linked lists as the arguments and returns the head of merged linked list.

**Expected Time Complexity** : O(n+m)  
**Expected Auxilliary Space** : O(1)

**Constraints:**  
1 <= N, M <= 104  
1 <= Node's data <= 105

Code:

#include<iostream>

using namespace std;

/\* Link list Node \*/

struct Node

{

int data;

struct Node \*next;

Node(int x)

{

data = x;

next = NULL;

}

};

Node\* sortedMerge(struct Node\* a, struct Node\* b);

/\* Function to print Nodes in a given linked list \*/

void printList(struct Node \*n)

{

while (n!=NULL)

{

cout << n->data << " ";

n = n->next;

}

cout << endl;

}

/\* Driver program to test above function\*/

int main()

{

int t;

cin>>t;

while(t--)

{

int n,m;

cin>>n>>m;

int data;

cin>>data;

struct Node \*head1 = new Node(data);

struct Node \*tail1 = head1;

for (int i = 1; i < n; ++i)

{

cin>>data;

tail1->next = new Node(data);

tail1 = tail1->next;

}

cin>>data;

struct Node \*head2 = new Node(data);

struct Node \*tail2 = head2;

for(int i=1; i<m; i++)

{

cin>>data;

tail2->next = new Node(data);

tail2 = tail2->next;

}

Node \*head = sortedMerge(head1, head2);

printList(head);

}

return 0;

}

// } Driver Code Ends

/\* Link list Node

struct Node {

int data;

struct Node \*next;

Node(int x) {

data = x;

next = NULL;

}

};

\*/

Node\* sortedMerge(Node\* head\_A, Node\* head\_B)

{

if(head\_A==NULL)

return head\_B;

if(head\_B==NULL)

return head\_A;

Node\* temp=NULL;

if(head\_A->data<head\_B->data)

{

temp=head\_A;

temp->next=sortedMerge(head\_A->next,head\_B);

}

else

{

temp=head\_B;

temp->next=sortedMerge(head\_A,head\_B->next);

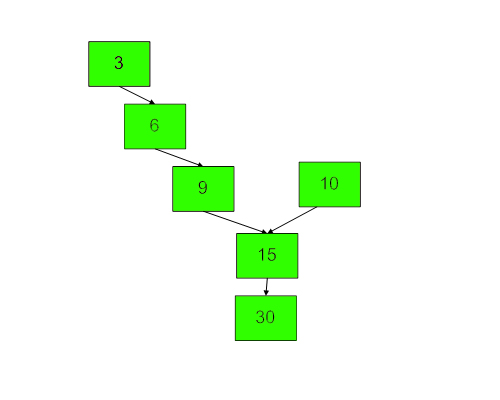
}

return temp;

}

11. **Intersection Point in Y Shapped Linked Lists**

Given two singly linked lists of size **N** and **M,**write a program to get the point where two linked lists intersect each other.

  
Above diagram shows an example with two linked list having 15 as intersection point.

**Example 1:**

**Input:**

LinkList1 = {10,20,5,10}

LinkList2 = {30,40,50,5,10}

**Output:** 5

**Explanation:**The point of intersection of

two linked list is 5, means both of them

get linked (intersects) with each other

at node whose value is 5.

**Your Task:**  
The task is to complete the function **intersetPoint**() which finds the point of intersection of two linked list. The function should return data value of a node where two linked lists merge. If linked list do not merge at any point, then it should return **-1**.

**Challenge**: Try to solve the problem without using any extra space.

**Expected Time Complexity:** O(N+M)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
1 <= N <= 100  
1 <= value <= 1000

Code:

#include<iostream>

#include<stdio.h>

using namespace std;

/\* Link list Node \*/

struct Node

{

int data;

struct Node \*next;

Node(int x)

{

data = x;

next = NULL;

}

};

int intersectPoint(struct Node\* head1, struct Node\* head2);

Node\* inputList(int size)

{

if(size==0) return NULL;

int val;

cin>> val;

Node \*head = new Node(val);

Node \*tail = head;

for(int i=0; i<size-1; i++)

{

cin>>val;

tail->next = new Node(val);

tail = tail->next;

}

return head;

}

/\* Driver program to test above function\*/

int main()

{

int T,n1,n2,n3;

cin>>T;

while(T--)

{

cin>>n1>>n2>>n3;

Node\* head1 = inputList(n1);

Node\* head2 = inputList(n2);

Node\* common = inputList(n3);

Node\* temp = head1;

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

temp = temp->next;

if(temp!=NULL) temp->next = common;

temp = head2;

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

temp = temp->next;

if(temp!=NULL) temp->next = common;

cout << intersectPoint(head1, head2) << endl;

}

return 0;

}

// } Driver Code Ends

/\* Linked List Node

struct Node {

int data;

struct Node \*next;

Node(int x) {

data = x;

next = NULL;

}

}; \*/

/\* Should return data of intersection point of two linked

lists head1 and head2.

If there is no intersecting point, then return -1. \*/

//find difference in number of nodes of both the linked list

int intersectPoint(Node\* head1, Node\* head2){

Node \*temp = head1;

while(temp){

temp->data = -1 \* (temp->data +1);

temp = temp->next;

}

temp = head2;

int ans = -1,p=0;

while(temp){

if(temp->data < 0){

p=1;

ans = -1\*temp->data;

break;

}

temp = temp->next;

}

if(p == 1)

return (ans-1);

return ans;

}

12. **Pairwise swap elements of a linked list**

Given a singly linked list of size **N**. The task is to swap elements in the linked list pairwise.  
For example, if the input list is 1 2 3 4, the resulting list after swaps will be 2 1 4 3.

**Example 1:**

**Input:**

LinkedList: 1->2->2->4->5->6->7->8

**Output:** 2 1 4 2 6 5 8 7

**Explanation:** After swapping each pair

considering (1,2), (2, 4), (5, 6).. so

on as pairs, we get 2, 1, 4, 2, 6, 5,

8, 7 as a new linked list.

**Your Task:**  
The task is to complete the function **pairWiseSwap**() which takes the head node as the only argument and returns the modified head.

**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(1).

**Constraints:**  
1 <= N <= 103

Code:

#include <bits/stdc++.h>

using namespace std;

struct Node

{

int data;

struct Node\* next;

Node(int x){

data = x;

next = NULL;

}

};

Node\* pairWiseSwap(Node\* head);

void printList(Node\* node)

{

while (node != NULL) {

cout << node->data <<" ";

node = node->next;

}

cout<<"\n";

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int data;

cin>>data;

struct Node \*head = new Node(data);

struct Node \*tail = head;

for (int i = 0; i<n-1; ++i)

{

cin>>data;

tail->next = new Node(data);

tail = tail->next;

}

head = pairWiseSwap(head);

printList(head);

}

return 0;

}

// } Driver Code Ends

/\*

Pairwise swap a linked list

The input list will have at least one element

node is defined as

struct node

{

int data;

struct node\* next;

node(int x){

data = x;

next = NULL;

}

}\*head;

\*/

Node\* pairWiseSwap(struct Node\* head) {

Node\* curr=head;

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

{

int x=curr->data;

curr->data=curr->next->data;

curr->next->data=x;

curr=curr->next->next;

}

return head;

}

13. **Add two numbers represented by linked lists**

Given two numbers represented by two linked lists of size **N** and **M**. The task is to return a sum list. The sum list is a linked list representation of the addition of two input numbers.

**Example 1:**

**Input:**

N = 2

valueN[] = {4,5}

M = 3

valueM[] = {3,4,5}

**Output:** 3 9 0

**Explanation:** For the given two linked

list (4 5) and (3 4 5), after adding

the two linked list resultant linked

list will be (3 9 0).

**Example 2:**

**Input:**

N = 2

valueN[] = {6,3}

M = 1

valueM[] = {7}

**Output:** 7 0

**Explanation:** For the given two linked

list (6 3) and (7), after adding the

two linked list resultant linked list

will be (7 0).

**Your Task:**  
The task is to complete the function **addTwoLists**() which has node reference of both the linked lists and returns the head of the new list.

**Expected Time Complexity:**O(N) + O(M)  
**Expected Auxiliary Space:**O(N) + O(M)

**Constraints:**  
1 <= N, M <= 5000

Code:

// driver

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

/\* Linked list Node \*/

struct Node {

int data;

struct Node\* next;

Node(int x) {

data = x;

next = NULL;

}

};

struct Node\* buildList(int size)

{

int val;

cin>> val;

Node\* head = new Node(val);

Node\* tail = head;

for(int i=0; i<size-1; i++)

{

cin>> val;

tail->next = new Node(val);

tail = tail->next;

}

return head;

}

void printList(Node\* n)

{

while(n)

{

cout<< n->data << " ";

n = n->next;

}

cout<< endl;

}

struct Node\* addTwoLists(struct Node\* first, struct Node\* second);

int main()

{

int t;

cin>>t;

while(t--)

{

int n, m;

cin>>n;

Node\* first = buildList(n);

cin>>m;

Node\* second = buildList(m);

Node\* res = addTwoLists(first,second);

printList(res);

}

return 0;

}

// } Driver Code Ends

/\* node for linked list:

struct Node {

int data;

struct Node\* next;

Node(int x) {

data = x;

next = NULL;

}

};

\*/

struct Node\* addTwoLists(struct Node\* first, struct Node\* second)

{

//reverse both LL

Node\* prev=NULL;

Node\* curr=first;

Node\* forw=curr->next;

while(curr)

{

forw=curr->next;

curr->next=prev;

prev=curr;

curr=forw;

}

Node\* head1=prev;

//reversing second LL

prev=NULL;

curr=second;

forw=curr->next;

while(curr)

{

forw=curr->next;

curr->next=prev;

prev=curr;

curr=forw;

}

Node\* head2=prev;

//adding both node by node

int sum=(head1->data+head2->data)%10;

int carry=(head1->data+head2->data)/10;

Node\* head=new Node(sum);

Node\* tail=head;

head1=head1->next;

head2=head2->next;

while(head1 && head2)

{

sum=(head1->data+head2->data+carry)%10;

Node\* temp=new Node(sum);

tail->next=temp;

tail=tail->next;

carry=(head1->data+head2->data+carry)/10;

head1=head1->next;

head2=head2->next;

}

if(head1) //if LL1 is not empty as yet

{

while(head1)

{

Node\* temp=new Node((head1->data+carry)%10);

carry=(head1->data+carry)/10;

tail->next=temp;

tail=tail->next;

head1=head1->next;

}

}

if(head2) //if LL2 is not empty as yet

{

while(head2)

{

Node\* temp=new Node((head2->data+carry)%10);

carry=(head2->data+carry)/10;

tail->next=temp;

tail=tail->next;

head2=head2->next;

}

}

//border case if carry still has something

if(carry)

{

Node\* temp=new Node(carry);

tail->next=temp;

tail=tail->next;

}

//now reverse so to get correct order

prev=NULL;

curr=head;

forw=curr->next;

while(curr)

{

forw=curr->next;

curr->next=prev;

prev=curr;

curr=forw;

}

return prev;

}

14. **Check if Linked List is Palindrome**

Given a singly linked list of size **N** of integers. The task is to check if the given linked list is palindrome or not.

**Example 1:**

**Input:**

N = 3

value[] = {1,2,1}

**Output:** 1

**Explanation:** The given linked list is

1 2 1 , which is a palindrome and

Hence, the output is 1.

**Example 2:**

**Input:**

N = 4

value[] = {1,2,3,4}

**Output:** 0

**Explanation:** The given linked list

is 1 2 3 4 , which is not a palindrome

and Hence, the output is 0.

**Your Task:**  
The task is to complete the function **isPalindrome**() which takes head as reference as the only parameter and returns true or false if linked list is palindrome or not respectively.

**Expected Time Complexity**: O(N)  
**Expected Auxialliary Space Usage**: O(1)  (ie, you should not use the recursive stack space as well)

**Constraints:**  
1 <= N <= 50

Code:

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

#include <stack>

using namespace std;

/\* Link list Node \*/

struct Node {

int data;

struct Node \*next;

Node(int x) {

data = x;

next = NULL;

}

};

bool isPalindrome(Node \*head);

/\* Driver program to test above function\*/

int main()

{

int T,i,n,l,firstdata;

cin>>T;

while(T--)

{

struct Node \*head = NULL, \*tail = NULL;

cin>>n;

// taking first data of LL

cin>>firstdata;

head = new Node(firstdata);

tail = head;

// taking remaining data of LL

for(i=1;i<n;i++)

{

cin>>l;

tail->next = new Node(l);

tail = tail->next;

}

cout<<isPalindrome(head)<<endl;

}

return 0;

}

// } Driver Code Ends

/\*

struct Node {

int data;

struct Node \*next;

Node(int x) {

data = x;

next = NULL;

}

};

\*/

/\*You are required to complete this method \*/

bool isPalindrome(Node \*head)

{

Node\* slow=head;

Node\* fast=head;

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

{

fast=fast->next->next;

slow=slow->next;

}

//now we will reverse second half of LL after slow pointer

Node\* prev=NULL;

Node\* curr=slow;

Node\* forw=curr->next;

while(curr!=NULL)

{

forw=curr->next;

curr->next=prev;

prev=curr;

curr=forw;

}

//now prev is new head of reversed LL

//now compare both LL

fast=head;

slow=prev;

while(slow!=NULL)

{

//compare for palindrome

if(fast->data!=slow->data)

return false;

fast=fast->next;

slow=slow->next;

}

return true;

//Your code here

}

15. **Implement Queue using Linked List**

Implement a Queue using Linked List.   
A Query **Q** is of 2 Types  
**(i)** 1 x   (a query of this type means  pushing **'x'** into the queue)  
**(ii)** 2     (a query of this type means to pop an element from the queue and print the poped element)

**Example 1:**

**Input:**

Q = 5

Queries = 1 2 1 3 2 1 4 2

**Output:** 2 3

**Explanation:** n the first testcase

1 2 the queue will be {2}

1 3 the queue will be {2 3}

2   poped element will be 2 the

  queue will be {3}

1 4 the queue will be {3 4}

2   poped element will be 3.

**Example 2:**

**Input:**

Q = 4

Queries = 1 2 2 2 1 3

**Output:** 2 -1

**Explanation:** In the second testcase

1 2 the queue will be {2}

2   poped element will be {2} then

  the queue will be empty.

2   the queue is empty and hence -1

1 3 the queue will be {3}.

**Your Task:**  
Complete the function **push()** which takes an integer as input parameter and **pop()** which will remove and return an element(-1 if queue is empty).

**Expected Time Complexity:**O(1).  
**Expected Auxiliary Space:**O(1).

**Constraints:**  
1 <=Q <= 100  
1 <= x <= 100

Code:

#include<bits/stdc++.h>

using namespace std;

struct QueueNode

{

int data;

QueueNode \*next;

QueueNode(int a)

{

data = a;

next = NULL;

}

};

struct MyQueue {

QueueNode \*front;

QueueNode \*rear;

void push(int);

int pop();

MyQueue() {front = rear = NULL;}

};

int main()

{

int T;

cin>>T;

while(T--)

{

MyQueue \*sq = new MyQueue();

int Q;

cin>>Q;

while(Q--){

int QueryType=0;

cin>>QueryType;

if(QueryType==1)

{

int a;

cin>>a;

sq->push(a);

}else if(QueryType==2){

cout<<sq->pop()<<" ";

}

}

cout<<endl;

}

}

// } Driver Code Ends

/\* Structure of a node in Queue

struct QueueNode

{

int data;

QueueNode \*next;

QueueNode(int a)

{

data = a;

next = NULL;

}

};

And structure of MyQueue

struct MyQueue {

QueueNode \*front;

QueueNode \*rear;

void push(int);

int pop();

MyQueue() {front = rear = NULL;}

}; \*/

/\* The method push to push element into the queue\*/

void MyQueue:: push(int x)

{

// Your Code

QueueNode \*temp= new QueueNode(x);

if(front==NULL)

{

front=rear=temp;

return;

}

else

{

rear->next=temp;

rear=temp;

}

}

/\*The method pop which return the element

poped out of the queue\*/

int MyQueue :: pop()

{

// Your Code

if(front==NULL)

return -1;

else

{

QueueNode \*temp=front;

int t=front->data;

front=front->next;

if(front==NULL)

rear=NULL;

delete temp;

return t;

}

}

16. **Implement Stack using Linked List**

Let's give it a try! You have a linked list and you have to implement the functionalities push and pop of stack using this given linked list. Your task is to use the class as shown in the comments in the code editor and complete the functions push() and pop() to implement a stack.

**Example 1**:

**Input**:

push(2)

push(3)

pop()

push(4)

pop()

**Output**: 3, 4

**Explanation**:

push(2) the stack will be {2}

push(3) the stack will be {2 3}

pop() poped element will be 3,

  the stack will be {2}

push(4) the stack will be {2 4}

pop() poped element will be 4

**Example 2:**

**Input**:

pop()

push(4)

push(5)

pop()

**Output**: -1, 5

**Your Task:**You are required to complete two methods **push() and pop().**The push() method takes one argument, an integer **'x'** to be pushed into the stack and **pop()** which returns an integer present at the top and popped out from the stack. If the stack is empty then return **-1** from the pop() method.  
  
**Expected Time Complexity:**O(1) for both **push()**and **pop()**.  
**Expected Auxiliary Space:**O(1) for both **push()**and **pop()**.  
  
**Constraints:**  
1 <= Q <= 100  
1 <= x <= 100

Code:

#include <bits/stdc++.h>

using namespace std;

struct StackNode {

int data;

StackNode \*next;

StackNode(int a) {

data = a;

next = NULL;

}

};

class MyStack {

private:

StackNode \*top;

public:

void push(int);

int pop();

MyStack() { top = NULL; }

};

int main() {

int T;

cin >> T;

while (T--) {

MyStack \*sq = new MyStack();

int Q;

cin >> Q;

while (Q--) {

int QueryType = 0;

cin >> QueryType;

if (QueryType == 1) {

int a;

cin >> a;

sq->push(a);

} else if (QueryType == 2) {

cout << sq->pop() << " ";

}

}

cout << endl;

}

}

// } Driver Code Ends

/\*

The structure of the node of the stack is

struct StackNode

{

int data;

StackNode \*next;

StackNode(int a)

{

data = a;

next = NULL;

}

};

// And this is structure of MyStack

class MyStack {

private:

StackNode \*top;

public :

void push(int);

int pop();

MyStack()

{

top = NULL;

}

};

/\* The method push to push element

into the stack \*/

void MyStack ::push(int x) {

StackNode\* ins=new StackNode(x);

if(top==NULL) {top=ins; return;}

StackNode \*p=top;

while(p->next!=NULL) p=p->next;

p->next=ins;

}

/\* The method pop which return the element

poped out of the stack\*/

int MyStack ::pop() {

if(top==NULL) return -1;

if(top->next==NULL){int h=top->data; top=NULL; return h;}

StackNode \*p=top;

while(p->next->next!=NULL) p=p->next;

int h=p->next->data;

p->next=NULL;

return h;

}

17. **Given a linked list of 0s, 1s and 2s, sort it.**

Given a linked list of **N** nodes where nodes can contain values **0s**, **1s,** and **2s**only. The task is to segregate **0s**, **1s,** and **2s** linked list such that all zeros segregate to head side, 2s at the end of the linked list, and 1s in the mid of 0s and 2s.

**Example 1:**

**Input:**

N = 8

value[] = {1,2,2,1,2,0,2,2}

**Output:** 0 1 1 2 2 2 2 2

**Explanation:** All the 0s are segregated

to the left end of the linked list,

2s to the right end of the list, and

1s in between.

**Example 2:**

**Input:**

N = 4

value[] = {2,2,0,1}

**Output:** 0 1 2 2

**Explanation:** After arranging all the

0s,1s and 2s in the given format,

the output will be 0 1 2 2.

**Your Task:**  
The task is to complete the function **segregate**() which segregates the nodes in the linked list as asked in the problem statement and returns the head of the modified linked list. The **printing**is done **automatically**by the **driver code**.

**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(1).

**Constraints:**  
1 <= N <= 103

Code:

#include <bits/stdc++.h>

using namespace std;

/\* Link list Node \*/

struct Node {

int data;

struct Node \*next;

Node(int x) {

data = x;

next = NULL;

}

};

struct Node \*start = NULL;

struct Node\* segregate(struct Node \*head);

// Function to sort a linked list of 0s, 1s and 2s

void printList(struct Node \*Node) {

while (Node != NULL) {

printf("%d ", Node->data);

Node = Node->next;

}

printf("\n");

}

/\* Drier program to test above function\*/

void insert(int n1) {

int n, value, i;

// scanf("%d",&n);

n = n1;

struct Node \*temp;

for (i = 0; i < n; i++) {

scanf("%d", &value);

if (i == 0) {

start = new Node(value);

temp = start;

continue;

} else {

temp->next = new Node(value);

temp = temp->next;

temp->next = NULL;

}

}

}

int main() {

int n;

int t;

scanf("%d", &t);

while (t--) {

scanf("%d", &n);

insert(n);

struct Node \*newHead = segregate(start);

printList(newHead);

}

return 0;

}// } Driver Code Ends

/\*

Sort the list of 0's,1's and 2's

The input list will have at least one element

Node is defined as

struct Node {

int data;

struct Node \*next;

Node(int x) {

data = x;

next = NULL;

}

};

\*/

// This function is to segregate the elememtns in the linked list

// This will do the required arrangement by changing the links

Node\* segregate(Node \*head) {

int cnt0=0;

int cnt1=0;

int cnt2=0;

Node\* temp=head;

while(temp)

{

if(temp->data==0)

cnt0++;

else if(temp->data==1)

cnt1++;

else

cnt2++;

temp=temp->next;

}

temp=head;

for(int i=0;i<cnt0;i++)

{

temp->data=0;

temp=temp->next;

}

for(int i=0;i<cnt1;i++)

{

temp->data=1;

temp=temp->next;

}

for(int i=0;i<cnt2;i++)

{

temp->data=2;

temp=temp->next;

}

return head;

// Add code here

}

18. **Delete without head pointer**

You are given a pointer/ reference to the node which is to be deleted from the linked list of **N**nodes. The task is to delete the node. Pointer/ reference to head node is not given.   
**Note:** No head reference is given to you.

**Example 1:**

**Input:**

N = 2

value[] = {1,2}

node = 1

**Output:** 2

**Explanation:** After deleting 1 from the

linked list, we have remaining nodes

as 2.

**Example 2:**

**Input:**

N = 4

value[] = {10,20,4,30}

node = 20

**Output:** 10 4 30

**Explanation:** After deleting 20 from

the linked list, we have remaining

nodes as 10, 4 and 30.

**Your Task:**  
You only need to complete the **function deleteNode**that takes **reference**to the node that needs to be **deleted**. The **printing**is done **automatically**by the**driver code**.

**Expected Time Complexity** : O(n)  
**Expected Auxilliary Space** : O(n)

**Constraints:**  
1 <= N <= 103

Code:

#include<bits/stdc++.h>

using namespace std;

/\* Link list node \*/

struct Node {

int data;

struct Node \*next;

Node(int x) {

data = x;

next = NULL;

}

}\*head;

Node \*findNode(Node\* head, int search\_for)

{

Node\* current = head;

while (current != NULL)

{

if (current->data == search\_for)

break;

current = current->next;

}

return current;

}

void insert()

{

int n,i,value;

Node \*temp;

scanf("%d",&n);

for(i=0; i<n; i++)

{

scanf("%d",&value);

if(i==0)

{

head=new Node(value);

temp=head;

continue;

}

else

{

temp->next= new Node(value);

temp=temp->next;

temp->next=NULL;

}

}

}

/\* Function to print linked list \*/

void printList(Node \*node)

{

while (node != NULL)

{

printf("%d ", node->data);

node = node->next;

}

cout << endl;

}

void deleteNode(Node \*node\_ptr);

/\* Drier program to test above function\*/

int main(void)

{

/\* Start with the empty list \*/

int t,k,n,value;

scanf("%d",&t);

while(t--)

{

insert();

scanf("%d",&k);

Node \*del = findNode(head, k);

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

{

deleteNode(del);

}

printList(head);

}

return(0);

}

// } Driver Code Ends

/\*

struct Node {

int data;

struct Node \*next;

Node(int x) {

data = x;

next = NULL;

}

}\*head;

\*/

// This function should delete node from linked list. The function

// may assume that node exists in linked list and is not last node

// node: reference to the node which is to be deleted

void deleteNode(Node \*node)

{Node\*prev;

while(node->next!=NULL){

node->data=node->next->data;

prev=node;

node=node->next;}

prev->next=NULL;

}// Your code here