**Problem Name:** Intersection of Two Linked List

**Topics:** Linked list, Hash Table, Two Pointers

**Companies:** Facebook, Microsoft, Amazon, Apple, LinkedIn, Nvidia, tiktok.

**Level:** Easy

**Language:** C++

**Problem Statement**: Given the heads of two singly linked-lists headA and headB, return the node at which the two lists intersect. If the two linked lists have no intersection at all, return null.

**Input Format:**

First line of the input contain integer n (size of list)

Second line contain n space separated integer list values.

Third line contain integer value m (size of second list)

Fourth line contain m space separated integer list values.

Last line contain integer pos (position of intersection)

Ex:

3

2 4 3

3

5 6 4

2

**Output Format:** Print true if returned pointer is equal to intersection pointer of node otherwise false.

**Constraints:**

* The number of nodes of listA is in the m.
* The number of nodes of listB is in the n.
* 1 <= m, n <= 3 \* 104
* 1 <= Node.val <= 105
* intersectVal is 0 if listA and listB do not intersect.
* intersectVal == listA[skipA] == listB[skipB] if listA and listB intersect.

**Examples:**

**Input:** intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5]

**Output:** Intersected at '8'

**Explanation:** The intersected node's value is 8 (note that this must not be 0 if the two lists intersect).

From the head of A, it reads as [4,1,8,4,5]. From the head of B, it reads as [5,6,1,8,4,5]. There are 2 nodes before the intersected node in A; There are 3 nodes before the intersected node in B.

**Brute force Solution:**

**Explanation:** Traverse entire LL-1 for each node in LL-2 and keep checking for the intersection point.

**Code:**

#include <bits/stdc++.h>

using namespace std;

class ListNode

{

    public:

        int val;

        ListNode\* next;

        ListNode(int a){

            val = a;

            next = NULL;

        }

};

void insertNode(ListNode\* &head,int val) {

    ListNode\* newNode = new ListNode(val);

    if(head == NULL) {

        head = newNode;

        return;

    }

    ListNode\* temp = head;

    while(temp->next != NULL)

     temp = temp->next;

    temp->next = newNode;

    return;

}

void printList(ListNode \*node)

{

    while (node!=NULL)

    {

        cout<<node->val<<" ";

        node = node->next;

    }

}

ListNode\* make\_intersection(ListNode\* head, ListNode\* tail, int k){

    ListNode\* curr = head;

    for(int i=1; i<k; i++){

        curr = curr->next;

    }

    ListNode\* tail\_pos = tail;

    while(tail\_pos->next != NULL)

        tail\_pos = tail\_pos->next;

    tail\_pos->next = curr;

    return curr;

}

ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB) {

    ListNode \*tempA=headA;

    ListNode \*tempB=headB;

    if((tempA==NULL)||(tempB==NULL))

        return NULL;

    while(tempB!=NULL)

    {

        while(tempA!=NULL)

        {

            if(tempA==tempB)

                return tempA;

            tempA=tempA->next;

        }

        tempA=headA;

        tempB=tempB->next;

    }

    return NULL;

}

int main() {

    ListNode\* a = NULL;

    ListNode\* b = NULL;

    ListNode\* res = NULL;

    ListNode\* pos = NULL;

    int n, m, temp,k;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cin>>m;

    while(m--){

        cin>>temp;

        insertNode(b, temp);

    }

    cin>>k;

    if(k>0)

        pos = make\_intersection(a,b, k);

    res = getIntersectionNode(a, b);

    if(res != NULL && res==pos)

        cout<<true<<"\n";

    else

        cout<<false<<"\n";

    return 0;

}

**Time Complexity**: O(N\*M)

**Space Complexity:** O(1)

**Optimized Solution:**

**Explanation:** Traverse LL-1 and LL-2 together, the one that reaches the end first, put it on the head of the other LL. (for example, LL-1 reaches null, then make ptr to point at the head of LL-2) Keep doing this until the address doessn't become same. i.e, until we doesnt find the intersection point. We're doing this to cover up the difference b/w LL-1 and LL-2. So, what if intersection point isnt there?? Will this method work? Yes, It will. Because if the intersection point isnt there, they both will store same address of NULL and that will be returned.

**Code:**

#include <bits/stdc++.h>

using namespace std;

class ListNode

{

    public:

        int val;

        ListNode\* next;

        ListNode(int a){

            val = a;

            next = NULL;

        }

};

void insertNode(ListNode\* &head,int val) {

    ListNode\* newNode = new ListNode(val);

    if(head == NULL) {

        head = newNode;

        return;

    }

    ListNode\* temp = head;

    while(temp->next != NULL)

     temp = temp->next;

    temp->next = newNode;

    return;

}

void printList(ListNode \*node)

{

    while (node!=NULL)

    {

        cout<<node->val<<" ";

        node = node->next;

    }

}

ListNode\* make\_intersection(ListNode\* head, ListNode\* tail, int k){

    ListNode\* curr = head;

    for(int i=1; i<k; i++){

        curr = curr->next;

    }

    ListNode\* tail\_pos = tail;

    while(tail\_pos->next != NULL)

        tail\_pos = tail\_pos->next;

    tail\_pos->next = curr;

    return curr;

}

ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB) {

    ListNode \*a = headA, \*b = headB;

    while (a != b) {

        a = !a ? headB : a->next;

        b = !b ? headA : b->next;

    }

    return a;

}

int main() {

    ListNode\* a = NULL;

    ListNode\* b = NULL;

    ListNode\* res = NULL;

    ListNode\* pos = NULL;

    int n, m, temp,k;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cin>>m;

    while(m--){

        cin>>temp;

        insertNode(b, temp);

    }

    cin>>k;

    if(k>0)

        pos = make\_intersection(a,b, k);

    res = getIntersectionNode(a, b);

    if(res != NULL && res==pos)

        cout<<true<<"\n";

    else

        cout<<false<<"\n";

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(1)