**Problem Name:** Linked list Cycles II

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

**Companies:** Amazon, Microsoft, Visa, Nvidia, Apple, Facebook, Google, Goldman Sachs, Oracle

**Level:** Medium

**Language:** C++

**Problem Statement**: Given the head of a linked list, return *the node where the cycle begins. If there is no cycle, return*null.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to (**0-indexed**). It is -1 if there is no cycle. **Note that** pos **is not passed as a parameter**.

**Do not modify** the linked list.

**Input Format:**

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

Second line contain n space separated integer list values.

Last line contain integer value pos representing index of connected node.

Ex:

5

1 2 3 4 5

1

**Output Format:** Print true if value of node at given position matches with detected cycle node

otherwise false.

**Constraints:**

* The number of the nodes in the list is in the range [0, 104].
* -105 <= Node.val <= 105
* pos is -1 or a **valid index** in the linked-list.

**Examples:**

**Input:** head = [3,2,0,-4], pos = 1

**Output:** tail connects to node index 1

**Explanation:** There is a cycle in the linked list, where tail connects to the second node.

**Solution:**

**Explanation:**

This problem is an application of [Floyd's cycle detection algorithm](https://en.wikipedia.org/wiki/Cycle_detection#Floyd's_Tortoise_and_Hare): the core idea is that in case there is a cycle, hare and turtle will meet at a point that is multiple of the length of the cycle itself, then you can get its starting point by "resetting" the turtle, so that their distance is now double it and finally get the start of the cycle having them proceed at normal speed

**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;

}

ListNode \*detectCycle(ListNode \*head) {

    ListNode \*slow = head, \*fast = head;

    while (fast && fast->next) {

        slow = slow->next;

        fast = fast->next->next;

        if (slow == fast)

            break;

    }

    if (!(fast && fast->next))

        return NULL;

    while (head != slow) {

        head = head->next;

        slow = slow->next;

    }

    return head;

}

ListNode\* make\_loop(ListNode\* head, int k){

    ListNode\* curr = head;

    for(int i=1; i<k; i++){

        curr = curr->next;

    }

    ListNode\* kth\_pos = curr;

    while(curr->next != NULL)

        curr = curr->next;

    curr->next = kth\_pos;

    //a->next->next->next->next = a;

    return kth\_pos;

}

int main()

{

    ListNode\* a = NULL;

    ListNode\* res =NULL;

    ListNode\* posn = NULL;

    int n, temp, pos;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cin>>pos;

    if(pos>=0)

        posn= make\_loop(a,pos);

    res = detectCycle(a);

    if(posn)

        cout<<true;

    else

        cout<<false;

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(1)

**Another approach:**

**Explanation:** Solution by using hash map

**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;

}

ListNode \*detectCycle(ListNode \*head) {

    unordered\_map<ListNode\*,bool> mp;

    while(head)

    {

        if(mp.count(head))

            return head;

        else

        {

            mp[head]=true;

            head=head->next;

        }

    }

    return NULL;

}

ListNode\* make\_loop(ListNode\* head, int k){

    ListNode\* curr = head;

    for(int i=1; i<k; i++){

        curr = curr->next;

    }

    ListNode\* kth\_pos = curr;

    while(curr->next != NULL)

        curr = curr->next;

    curr->next = kth\_pos;

    //a->next->next->next->next = a;

    return kth\_pos;

}

int main()

{

    ListNode\* a = NULL;

    ListNode\* res =NULL;

    ListNode\* posn = NULL;

    int n, temp, pos;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cin>>pos;

    if(pos>=0)

        posn= make\_loop(a,pos);

    res = detectCycle(a);

    if(posn)

        cout<<true;

    else

        cout<<false;

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(N)