**Problem Name:** Linked list Cycles

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

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

**Level:** Easy

**Language:** C++

**Problem Statement**: Given head, the head of a linked list, determine if the linked list has a cycle in it. 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. Note that pos is not passed as a parameter.

Return true*if there is a cycle in the linked list*. Otherwise, return false

**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:** Boolean value 1 0r 0 indicating cycle is present in list or not.

**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:** true

**Explanation:** There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

**Solution:**

**Explanation:**  We use two pointers to traverse the list: The first one is moving one node at the time and the second two nodes at the time. If there is a cycle, sooner or later pointers will meet and we return true. If the fast pointer reached the end of the list, that means there is no cycle and we can return false.

*For reference:*[*Floyd's Cycle Detection Algorithm*](https://en.wikipedia.org/wiki/Cycle_detection#Floyd's_tortoise_and_hare)

**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;

}

bool hasCycle(ListNode \*head) {

    if(head == NULL)

        return false;

    ListNode \*fast = head;

    ListNode \*slow = head;

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

    {

        fast = fast->next->next;

        slow = slow->next;

        if(fast == slow)

            return true;

    }

    return false;

}

void 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;

}

int main()

{

    ListNode\* a = NULL;

    int n, temp, pos;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cin>>pos;

    if(pos>=0)

        make\_loop(a,pos);

    cout<<hasCycle(a);

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(1)

**Optimized Solution:**

**Explanation:** Solution using Hashmap

**Code:**

//Hashmap method

#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;

}

bool hasCycle(ListNode \*head) {

    if(head==NULL)

        return false;

    unordered\_map<ListNode\*,int> map;

    while(head!=NULL){

        if(map.count(head)>0)

            return true;

        else

            map[head]=1;

        head=head->next;

    }

    return false;

}

void 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;

}

int main()

{

    ListNode\* a = NULL;

    int n, temp, pos;

    cin>>n;

    while(n--){

        cin>>temp;

        insertNode(a, temp);

    }

    cin>>pos;

    if(pos>=0)

        make\_loop(a,pos);

    cout<<hasCycle(a);

    return 0;

}

**Time Complexity**: O(1)

**Space Complexity:** O(1)