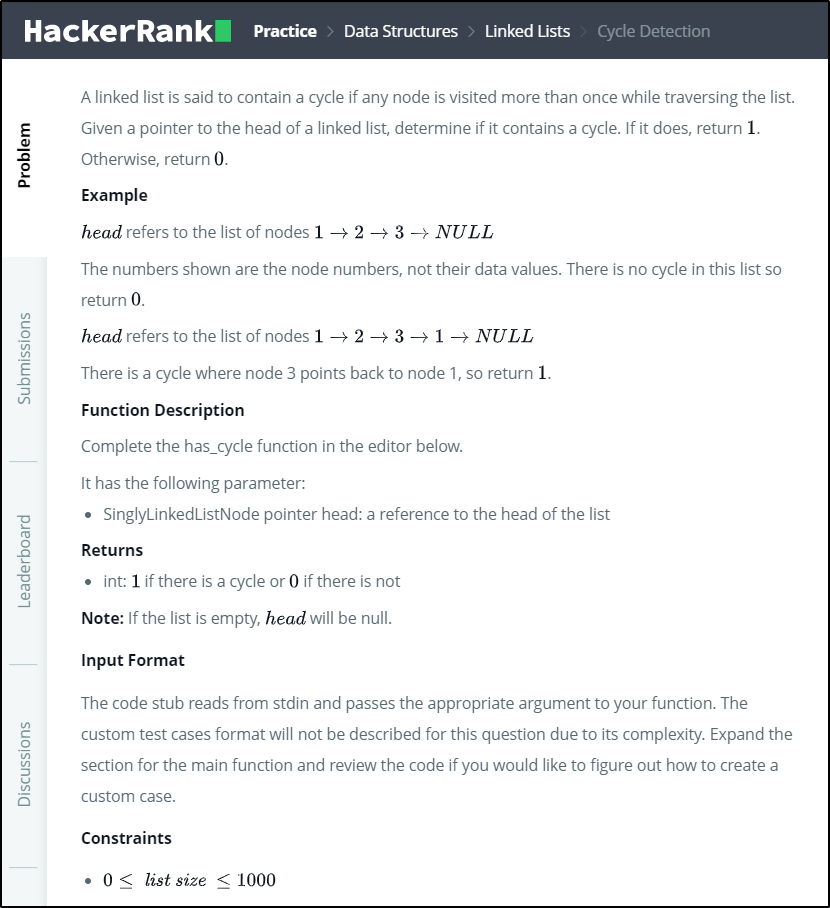
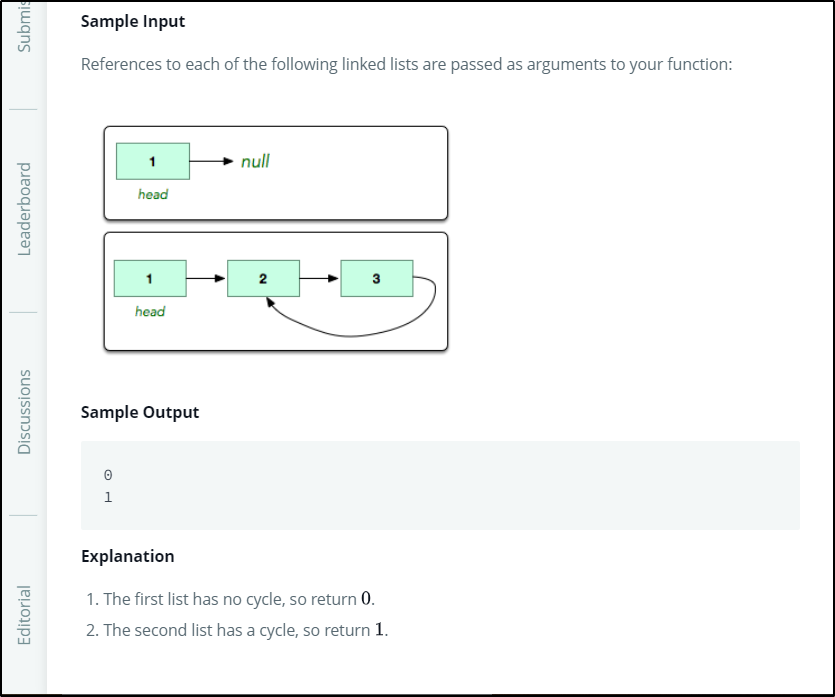
**PROBLEM STATEMENT**





**PROGRAM USED TO SOLVE THE PROBLEM STATEMENT**

#include <assert.h>

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char\* readline();

typedef struct SinglyLinkedListNode SinglyLinkedListNode;

typedef struct SinglyLinkedList SinglyLinkedList;

struct SinglyLinkedListNode {

    int data;

    SinglyLinkedListNode\* next;

};

struct SinglyLinkedList {

    SinglyLinkedListNode\* head;

    SinglyLinkedListNode\* tail;

};

SinglyLinkedListNode\* create\_singly\_linked\_list\_node(int node\_data) {

    SinglyLinkedListNode\* node = malloc(sizeof(SinglyLinkedListNode));

    node->data = node\_data;

    node->next = NULL;

    return node;

}

void insert\_node\_into\_singly\_linked\_list(SinglyLinkedList\*\* singly\_linked\_list, int node\_data) {

    SinglyLinkedListNode\* node = create\_singly\_linked\_list\_node(node\_data);

    if (!(\*singly\_linked\_list)->head) {

        (\*singly\_linked\_list)->head = node;

    } else {

        (\*singly\_linked\_list)->tail->next = node;

    }

    (\*singly\_linked\_list)->tail = node;

}

void print\_singly\_linked\_list(SinglyLinkedListNode\* node, char\* sep, FILE\* fptr) {

    while (node) {

        fprintf(fptr, "%d", node->data);

        node = node->next;

        if (node) {

            fprintf(fptr, "%s", sep);

        }

    }

}

void free\_singly\_linked\_list(SinglyLinkedListNode\* node) {

    while (node) {

        SinglyLinkedListNode\* temp = node;

        node = node->next;

        free(temp);

    }

}

// Complete the has\_cycle function below.

/\*

 \* For your reference:

 \*

 \* SinglyLinkedListNode {

 \*     int data;

 \*     SinglyLinkedListNode\* next;

 \* };

 \*

 \*/

bool has\_cycle(SinglyLinkedListNode\* head) {

    int count = 0;

    SinglyLinkedListNode \*temp = head;

    while (temp!=NULL) {

        temp = temp->next;

        ++count;

        if (count>1000) {

            return true;

        }

    }

    return false;

}

int main()

{

    FILE\* fptr = fopen(getenv("OUTPUT\_PATH"), "w");

    char\* tests\_endptr;

    char\* tests\_str = readline();

    int tests = strtol(tests\_str, &tests\_endptr, 10);

    if (tests\_endptr == tests\_str || \*tests\_endptr != '\0') { exit(EXIT\_FAILURE); }

    for (int tests\_itr = 0; tests\_itr < tests; tests\_itr++) {

        char\* index\_endptr;

        char\* index\_str = readline();

        int index = strtol(index\_str, &index\_endptr, 10);

        if (index\_endptr == index\_str || \*index\_endptr != '\0') { exit(EXIT\_FAILURE); }

        SinglyLinkedList\* llist = malloc(sizeof(SinglyLinkedList));

        llist->head = NULL;

        llist->tail = NULL;

        char\* llist\_count\_endptr;

        char\* llist\_count\_str = readline();

        int llist\_count = strtol(llist\_count\_str, &llist\_count\_endptr, 10);

        if (llist\_count\_endptr == llist\_count\_str || \*llist\_count\_endptr != '\0') { exit(EXIT\_FAILURE); }

        for (int i = 0; i < llist\_count; i++) {

            char\* llist\_item\_endptr;

            char\* llist\_item\_str = readline();

            int llist\_item = strtol(llist\_item\_str, &llist\_item\_endptr, 10);

            if (llist\_item\_endptr == llist\_item\_str || \*llist\_item\_endptr != '\0') { exit(EXIT\_FAILURE); }

            insert\_node\_into\_singly\_linked\_list(&llist, llist\_item);

        }

        SinglyLinkedListNode\* extra = create\_singly\_linked\_list\_node(-1);

        SinglyLinkedListNode\* temp = llist->head;

        for (int i = 0; i < llist\_count; i++) {

            if (i == index) {

                extra = temp;

            }

            if (i != llist\_count-1) {

                temp = temp->next;

            }

        }

        temp->next = extra;

        bool result = has\_cycle(llist->head);

        fprintf(fptr, "%d\n", result);

    }

    fclose(fptr);

    return 0;

}

char\* readline() {

    size\_t alloc\_length = 1024;

    size\_t data\_length = 0;

    char\* data = malloc(alloc\_length);

    while (true) {

        char\* cursor = data + data\_length;

        char\* line = fgets(cursor, alloc\_length - data\_length, stdin);

        if (!line) { break; }

        data\_length += strlen(cursor);

        if (data\_length < alloc\_length - 1 || data[data\_length - 1] == '\n') { break; }

        size\_t new\_length = alloc\_length << 1;

        data = realloc(data, new\_length);

        if (!data) { break; }

        alloc\_length = new\_length;

    }

    if (data[data\_length - 1] == '\n') {

        data[data\_length - 1] = '\0';

    }

    data = realloc(data, data\_length);

    return data;

}

**TEST CASES**

