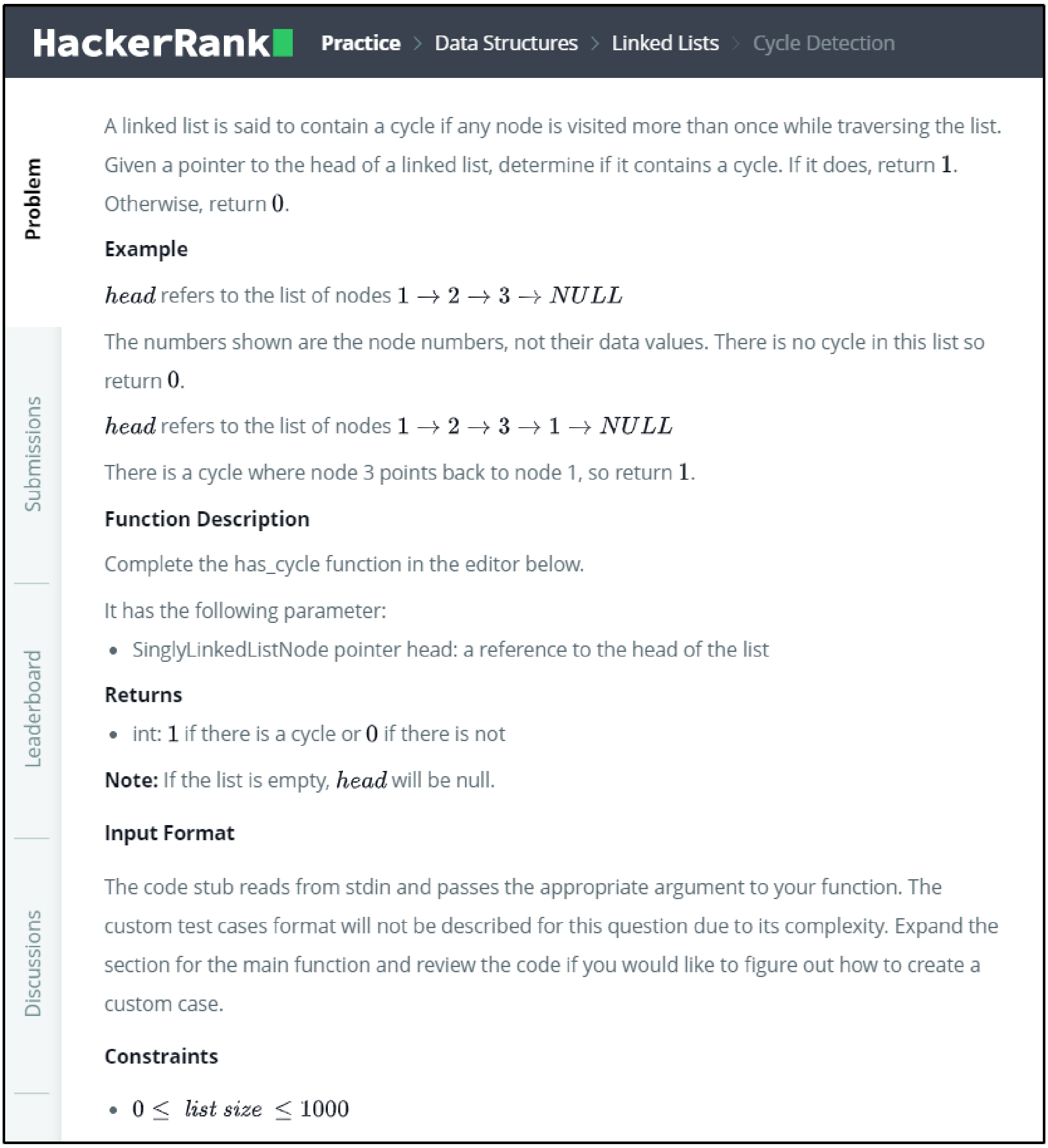
PROBLEM STATEMENT



A linked list is said to contain a cycle if any node is visited more than once while traversing the list. Given a pointer to the head of a linked list, determine if it contains a cycle. If it does, return 1. Otherwise, return 0.

Example

$head$ refers to the list of nodes $1 \rightarrow 2 \rightarrow 3 \rightarrow NULL$

The numbers shown are the node numbers, not their data values. There is no cycle in this list so return 0.

$head$ refers to the list of nodes $1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow NULL$

There is a cycle where node 3 points back to node 1, so return 1.

Function Description

Complete the has_cycle function in the editor below.

It has the following parameter:

- SinglyLinkedListNode pointer head: a reference to the head of the list

Returns

- int: 1 if there is a cycle or 0 if there is not

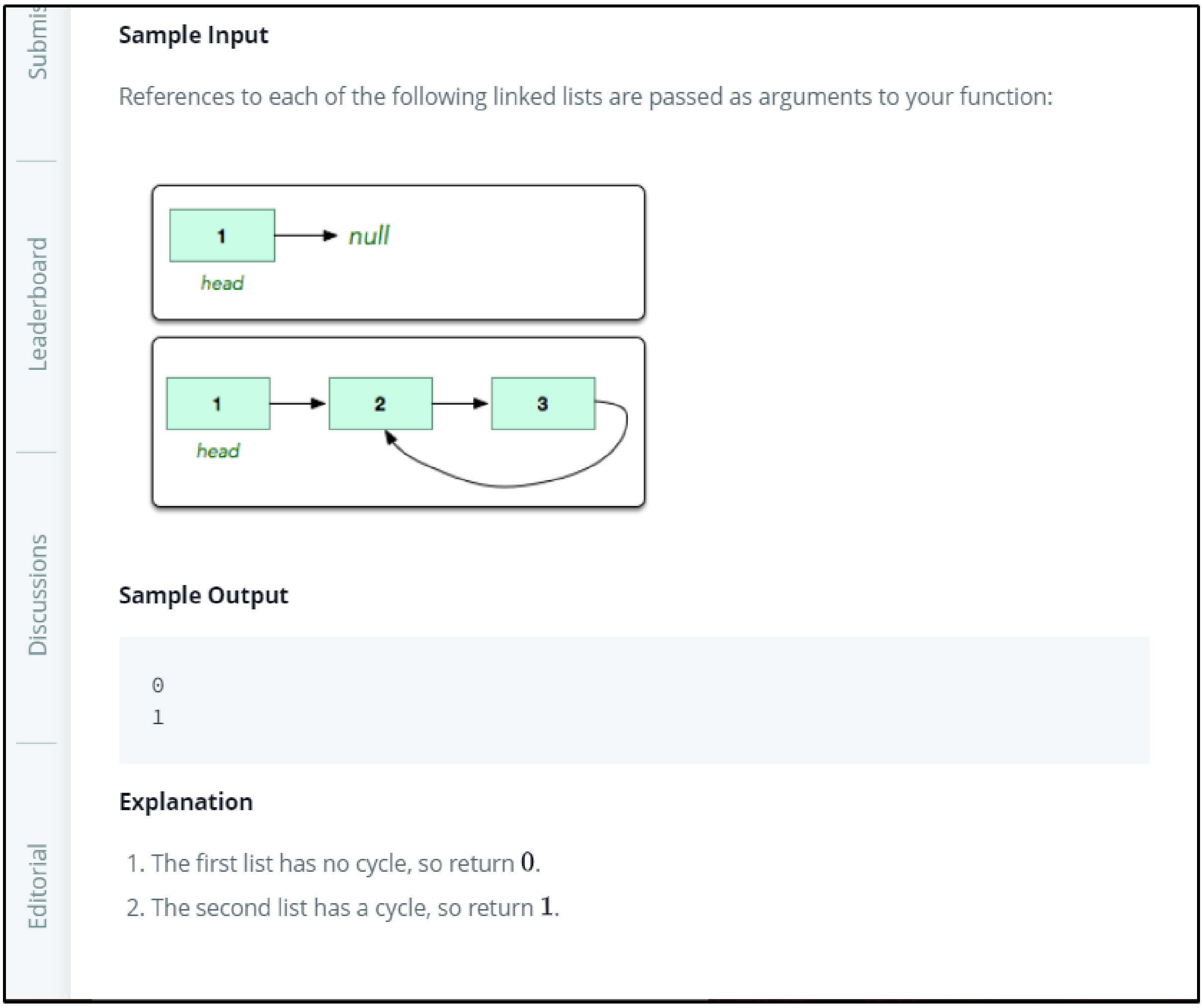
Note: If the list is empty, $head$ will be null.

Input Format

The code stub reads from stdin and passes the appropriate argument to your function. The custom test cases format will not be described for this question due to its complexity. Expand the section for the main function and review the code if you would like to figure out how to create a custom case.

Constraints

- $0 \leq \ list\ size\ \leq 1000$



Sample Input

References to each of the following linked lists are passed as arguments to your function:

Sample Output

```
0
1
```

Explanation

1. The first list has no cycle, so return 0.
2. The second list has a cycle, so return 1.

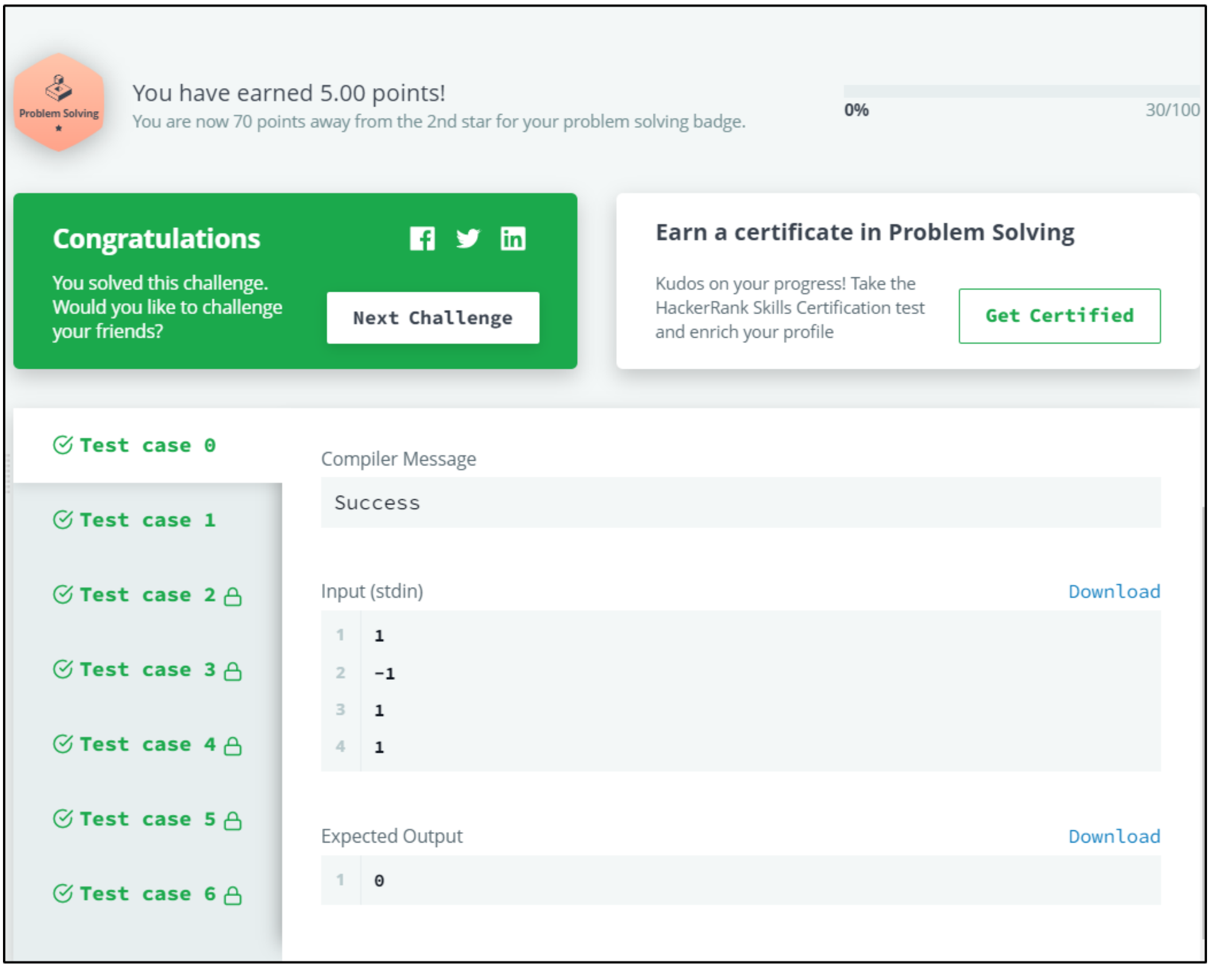
PROGRAM USED TO SOLVE THE PROBLEM STATEMENT

```c
#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(SinglyLink
edList));
        llist->head = NULL;
        llist->tail = NULL;

        char* llist_count_endptr;
        char* llist_count_str = readline();
        int llist_count = strtol(llist_count_str, &llist_c
ount_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;
}
```

```c
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



You have earned 5.00 points!

You are now 70 points away from the 2nd star for your problem solving badge.

0% 30/100

Congratulations

You solved this challenge. Would you like to challenge your friends?

Earn a certificate in Problem Solving

Kudos on your progress! Take the HackerRank Skills Certification test and enrich your profile

✓ Test case 0

✓ Test case 1

✓ Test case 2 🔒

✓ Test case 3 🔒

✓ Test case 4 🔒

✓ Test case 5 🔒

✓ Test case 6 🔒

Compiler Message

```
Success
```

Input (stdin) Download

```
1
-1
1
1
```

Expected Output Download

```
0
```