

**RAMAIAH INSTITUTE OF TECHNOLOGY**

**MSR NAGAR , BENGALURU – 560054**

**DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING**

**(2023-24)**

**A Report on**

**LEETCODE PROBLEM ON**

**STACKS,RECURSION,QUEUES,LINKED LISTS AND BST**

**Submitted By:**

**SUHAS T G**

**(1MS22IS140)**

**Faculty In charge:**

**Mr.Shivanand**

**Assistant Professor(Dept of ISE)**

**RECURSION:**

**1.POWER OF TWO , Level : Easy**

Given an integer n, return true if it is a power of two. Otherwise, return false.

An integer n is a power of two, if there exists an integer x such that n == 2x.

Example 1:

Input: n = 1  
Output: true  
Explanation: 20 = 1

Example 2:

Input: n = 16  
Output: true  
Explanation: 24 = 16

Example 3:

Input: n = 3  
Output: false

Constraints:

-231 <= n <= 231 – 1

SOLUTION:

**bool isPowerOfTwo(int n) {**

**if( n == 1)**

**return true;**

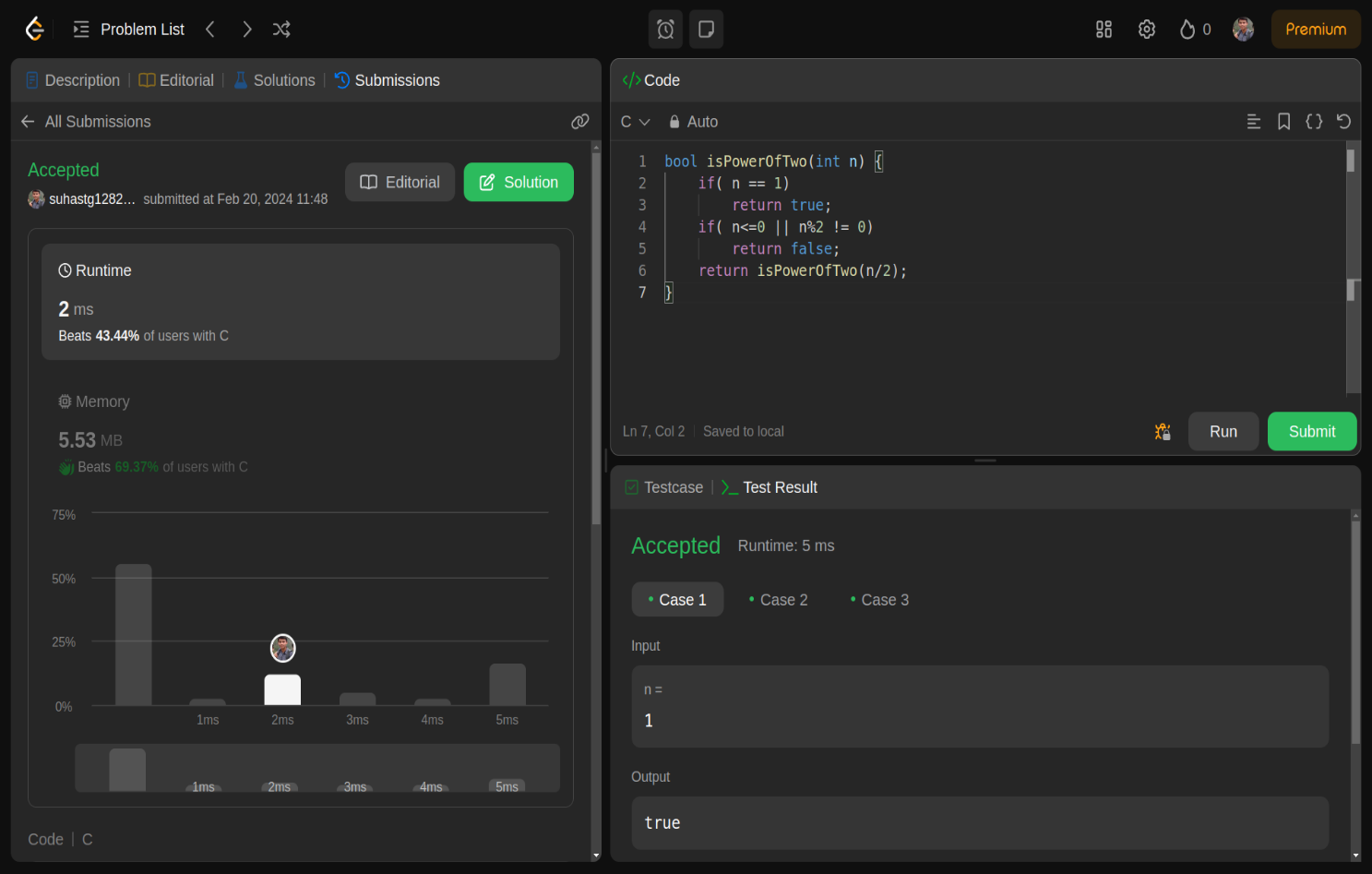
**if( n<=0 || n%2 != 0)**

**return false;**

**return isPowerOfTwo(n/2);**

**}**

OUTPUT:



**STACK:**

**2.IMPLEMENTING QUEUE USING STACK**

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).

Implement the MyQueue class:

void push(int x) Pushes element x to the back of the queue.

int pop() Removes the element from the front of the queue and returns it.

int peek() Returns the element at the front of the queue.

boolean empty() Returns true if the queue is empty, false otherwise.

SOLUTION :

**#include <stdbool.h>**

**#include <stdlib.h>**

**#define MAX\_SIZE 1000**

**typedef struct {**

**int stack1[MAX\_SIZE]; // Stack for enqueue operation**

**int top1; // Top of stack1**

**int stack2[MAX\_SIZE]; // Stack for dequeue operation**

**int top2; // Top of stack2**

**} MyQueue;**

**MyQueue\* myQueueCreate() {**

**MyQueue\* obj = (MyQueue\*)malloc(sizeof(MyQueue));**

**obj->top1 = -1;**

**obj->top2 = -1;**

**return obj;**

**}**

**void myQueuePush(MyQueue\* obj, int x) {**

**obj->stack1[++obj->top1] = x; // Push element to stack1**

**}**

**int myQueuePop(MyQueue\* obj) {**

**// If stack2 is empty, transfer elements from stack1 to stack2**

**if (obj->top2 == -1) {**

**while (obj->top1 != -1) {**

**obj->stack2[++obj->top2] = obj->stack1[obj->top1--];**

**}**

**}**

**// Pop element from stack2**

**return obj->stack2[obj->top2--];**

**}**

**int myQueuePeek(MyQueue\* obj) {**

**// If stack2 is empty, transfer elements from stack1 to stack2**

**if (obj->top2 == -1) {**

**while (obj->top1 != -1) {**

**obj->stack2[++obj->top2] = obj->stack1[obj->top1--];**

**}**

**}**

**// Return the top element of stack2 (front of the queue)**

**return obj->stack2[obj->top2];**

**}**

**bool myQueueEmpty(MyQueue\* obj) {**

**// If both stacks are empty, the queue is empty**

**return (obj->top1 == -1 && obj->top2 == -1);**

**}**

**void myQueueFree(MyQueue\* obj) {**

**free(obj);**

**}**

**/\*\***

**\* Your MyQueue struct will be instantiated and called as such:**

**\* MyQueue\* obj = myQueueCreate();**

**\* myQueuePush(obj, x);**

**\* int param\_2 = myQueuePop(obj);**

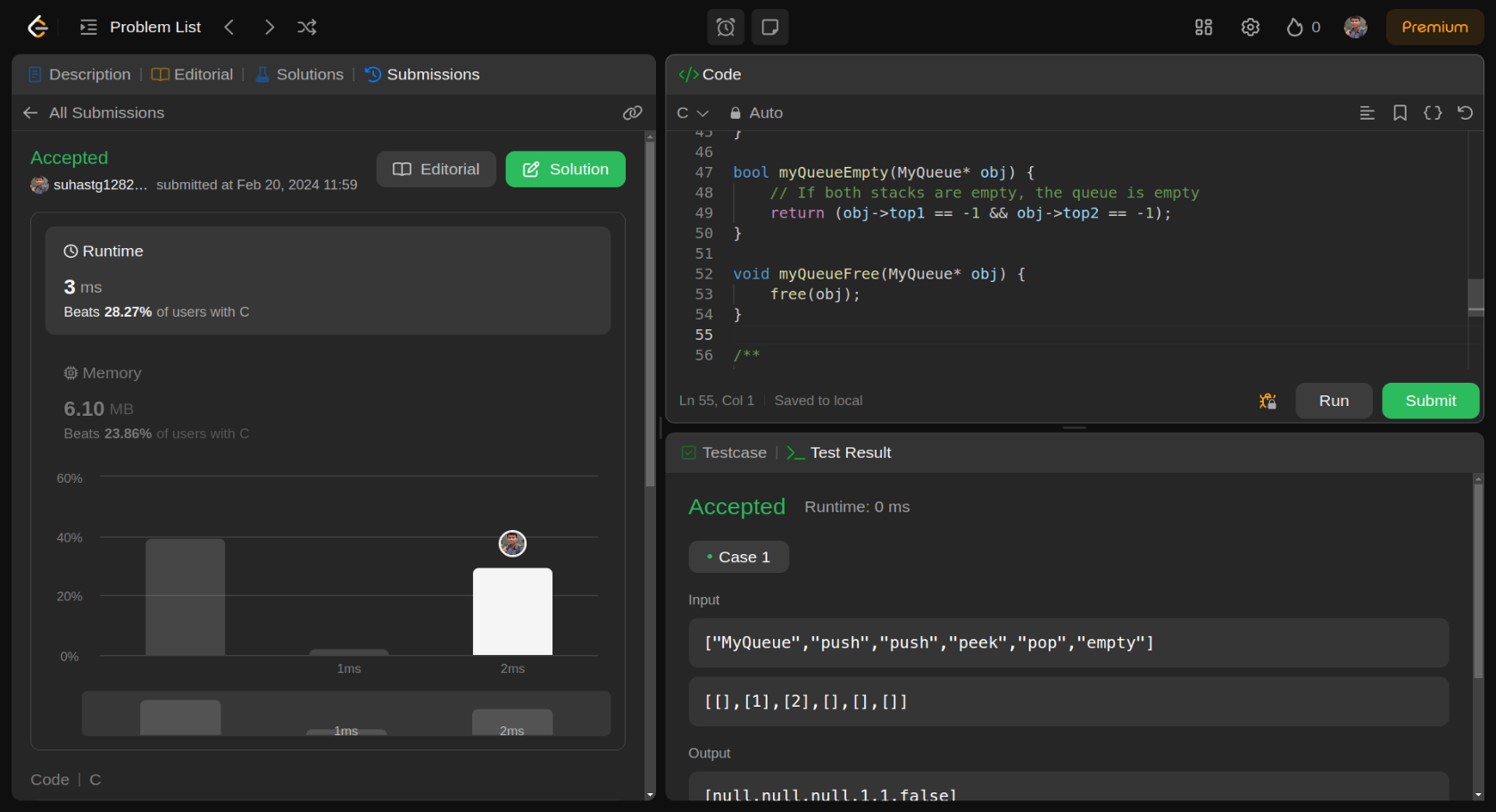
**\* int param\_3 = myQueuePeek(obj);**

**\* bool param\_4 = myQueueEmpty(obj);**

**\* myQueueFree(obj);**

**\*/**

OUTPUT:



**3.VALID PARENTHESES , Level – Medium**

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

Every close bracket has a corresponding open bracket of the same type.

Example 1:

Input: s = "()"  
Output: true

Example 2:

Input: s = "()[]{}"  
Output: true

Example 3:

Input: s = "(]"  
Output: false

SOLUTION :

**bool isValid(char\* s) {**

**char st[100];**

**int top = -1;**

**for(int i = 0; s[i] != '\0'; i++) {**

**if(s[i] == '(' || s[i] == '{' || s[i] == '[') {**

**st[++top] = s[i];**

**}**

**else if(s[i] == ')' || s[i] == '}' || s[i] == ']') {**

**if(top == -1)**

**return false;**

**else if(s[i] == ')' && st[top] != '(')**

**return false;**

**else if(s[i] == '}' && st[top] != '{')**

**return false;**

**else if(s[i] == ']' && st[top] != '[')**

**return false;**

**top--; // After successfully matching, decrement top**

**}**

**}**

**if(top == -1)**

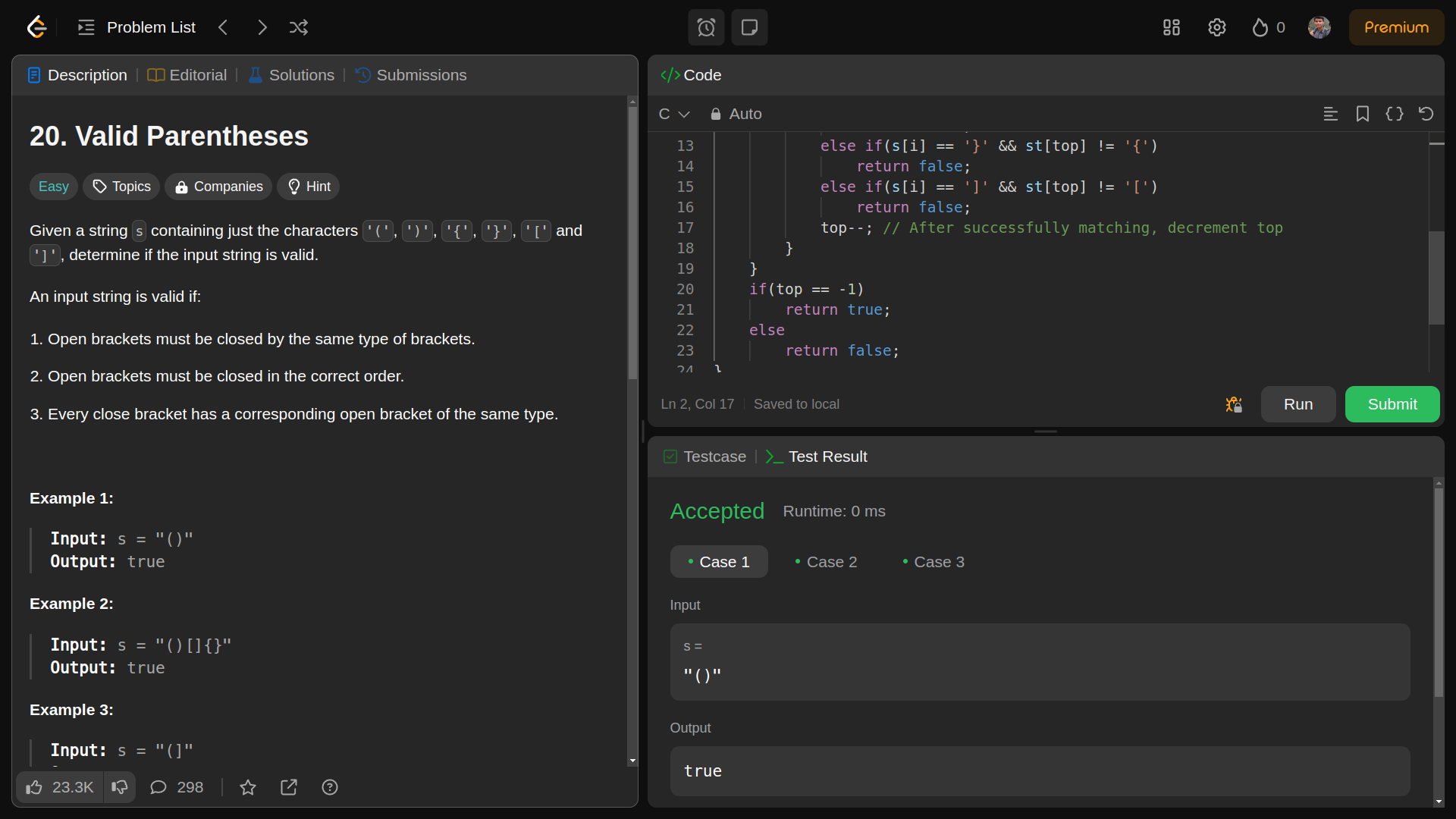
**return true;**

**else**

**return false;**

**}**

**OUTPUT:**



**QUEUE :**

**4. Number of Recent Calls ,Level : Easy**

You have a RecentCounter class which counts the number of recent requests within a certain time frame.

Implement the RecentCounter class:

RecentCounter() Initializes the counter with zero recent requests.

int ping(int t) Adds a new request at time t, where t represents some time in milliseconds, and returns the number of requests that has happened in the past 3000 milliseconds (including the new request). Specifically, return the number of requests that have happened in the inclusive range [t - 3000, t].

It is guaranteed that every call to ping uses a strictly larger value of t than the previous call.

Example 1:

Input  
["RecentCounter", "ping", "ping", "ping", "ping"]  
[[], [1], [100], [3001], [3002]]  
Output  
[null, 1, 2, 3, 3]

SOLUTION :

**typedef struct {**

**int \*requests; // Array to store requests**

**int front; // Front of the queue**

**int rear; // Rear of the queue**

**} RecentCounter;**

**RecentCounter\* recentCounterCreate() {**

**RecentCounter\* obj = (RecentCounter\*)malloc(sizeof(RecentCounter));**

**obj->requests = (int\*)malloc(10000 \* sizeof(int)); // Assuming maximum number of requests**

**obj->front = 0;**

**obj->rear = -1;**

**return obj;**

**}**

**int recentCounterPing(RecentCounter\* obj, int t) {**

**// Add the new request to the queue**

**obj->requests[++obj->rear] = t;**

**// Remove requests older than 3000 milliseconds**

**while (obj->requests[obj->front] < t - 3000)**

**obj->front++;**

**// Return the number of requests within the last 3000 milliseconds**

**return obj->rear - obj->front + 1;**

**}**

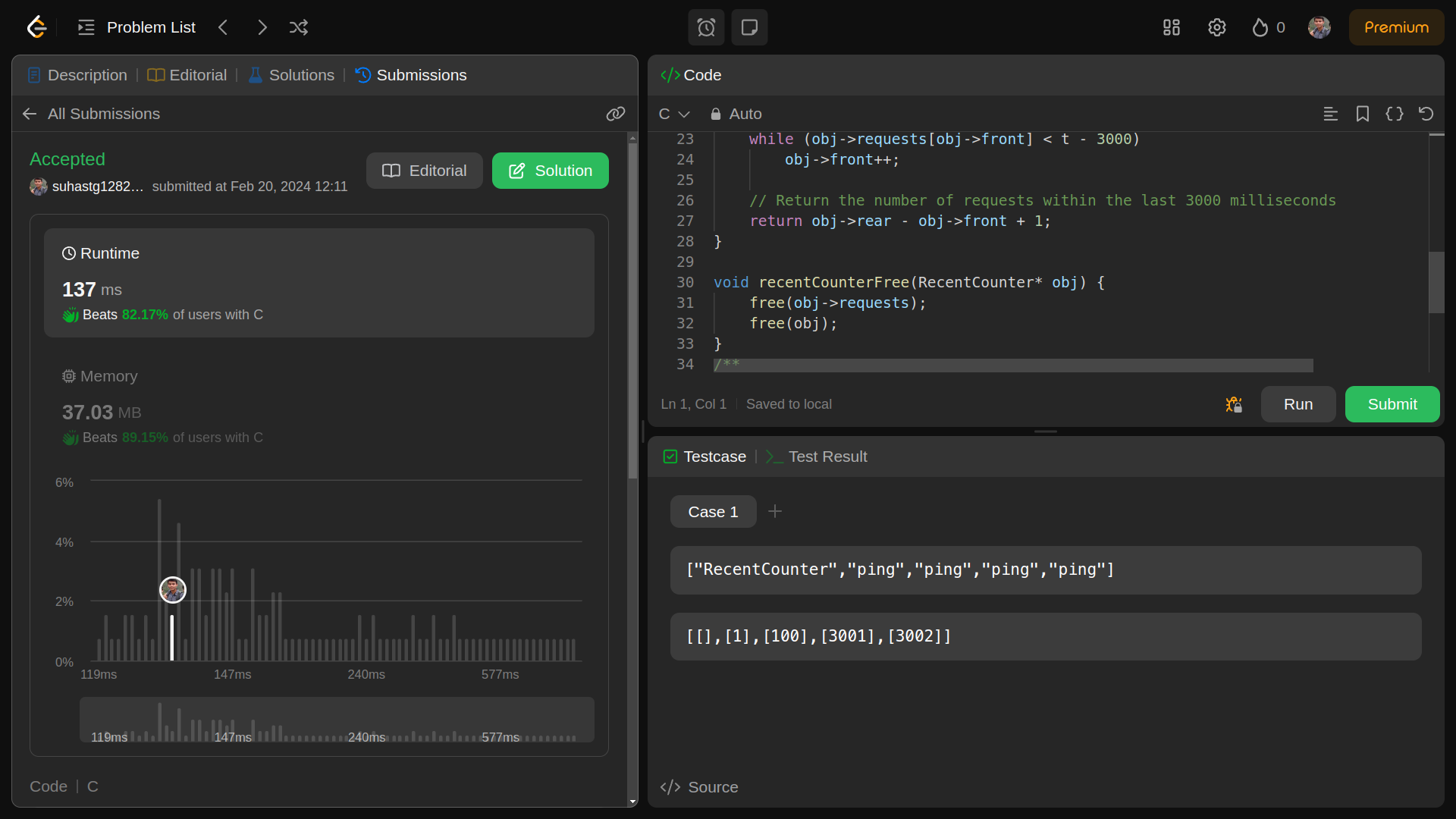
**void recentCounterFree(RecentCounter\* obj) {**

**free(obj->requests);**

**free(obj);**

**}**

**OUTPUT:**



**CIRCULAR QUEUE:**

**5.DESIGN A CIRCULAR QUEUE , Level : Medium**

Design your implementation of the circular queue. The circular queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle, and the last position is connected back to the first position to make a circle. It is also called "Ring Buffer".

One of the benefits of the circular queue is that we can make use of the spaces in front of the queue. In a normal queue, once the queue becomes full, we cannot insert the next element even if there is a space in front of the queue. But using the circular queue, we can use the space to store new values.

Implement the MyCircularQueue class:

MyCircularQueue(k) Initializes the object with the size of the queue to be k.

int Front() Gets the front item from the queue. If the queue is empty, return -1.

int Rear() Gets the last item from the queue. If the queue is empty, return -1.

boolean enQueue(int value) Inserts an element into the circular queue. Return true if the operation is successful.

boolean deQueue() Deletes an element from the circular queue. Return true if the operation is successful.

boolean isEmpty() Checks whether the circular queue is empty or not.

boolean isFull() Checks whether the circular queue is full or not.

You must solve the problem without using the built-in queue data structure in your programming language.

Example 1:

Input  
["MyCircularQueue", "enQueue", "enQueue", "enQueue", "enQueue", "Rear", "isFull", "deQueue", "enQueue", "Rear"]  
[[3], [1], [2], [3], [4], [], [], [], [4], []]  
Output  
[null, true, true, true, false, 3, true, true, true, 4]  
  
Explanation  
MyCircularQueue myCircularQueue = new MyCircularQueue(3);  
myCircularQueue.enQueue(1); // return True  
myCircularQueue.enQueue(2); // return True  
myCircularQueue.enQueue(3); // return True  
myCircularQueue.enQueue(4); // return False  
myCircularQueue.Rear(); // return 3  
myCircularQueue.isFull(); // return True  
myCircularQueue.deQueue(); // return True  
myCircularQueue.enQueue(4); // return True  
myCircularQueue.Rear(); // return 4

Constraints:

1 <= k <= 1000

0 <= value <= 1000

At most 3000 calls will be made to enQueue, deQueue, Front, Rear, isEmpty, and isFull.

**SOLUTION:**

**#include <stdbool.h>**

**#include <stdlib.h>**

**typedef struct {**

**int \*q;**

**int headIdx;**

**int cnt;**

**int sz;**

**} MyCircularQueue;**

**MyCircularQueue\* myCircularQueueCreate(int k) {**

**MyCircularQueue\* obj = (MyCircularQueue\*)malloc(sizeof(MyCircularQueue));**

**obj->q = (int\*)malloc(k \* sizeof(int));**

**obj->headIdx = 0;**

**obj->cnt = 0;**

**obj->sz = k;**

**return obj;**

**}**

**bool myCircularQueueEnQueue(MyCircularQueue\* obj, int value) {**

**if (myCircularQueueIsFull(obj))**

**return false;**

**obj->q[(obj->headIdx + obj->cnt) % obj->sz] = value;**

**obj->cnt += 1;**

**return true;**

**}**

**bool myCircularQueueDeQueue(MyCircularQueue\* obj) {**

**if (myCircularQueueIsEmpty(obj))**

**return false;**

**obj->headIdx = (obj->headIdx + 1) % obj->sz;**

**obj->cnt -= 1;**

**return true;**

**}**

**int myCircularQueueFront(MyCircularQueue\* obj) {**

**if (myCircularQueueIsEmpty(obj))**

**return -1;**

**return obj->q[obj->headIdx];**

**}**

**int myCircularQueueRear(MyCircularQueue\* obj) {**

**if (myCircularQueueIsEmpty(obj))**

**return -1;**

**return obj->q[(obj->headIdx + obj->cnt - 1) % obj->sz];**

**}**

**bool myCircularQueueIsEmpty(MyCircularQueue\* obj) {**

**return obj->cnt == 0;**

**}**

**bool myCircularQueueIsFull(MyCircularQueue\* obj) {**

**return obj->cnt == obj->sz;**

**}**

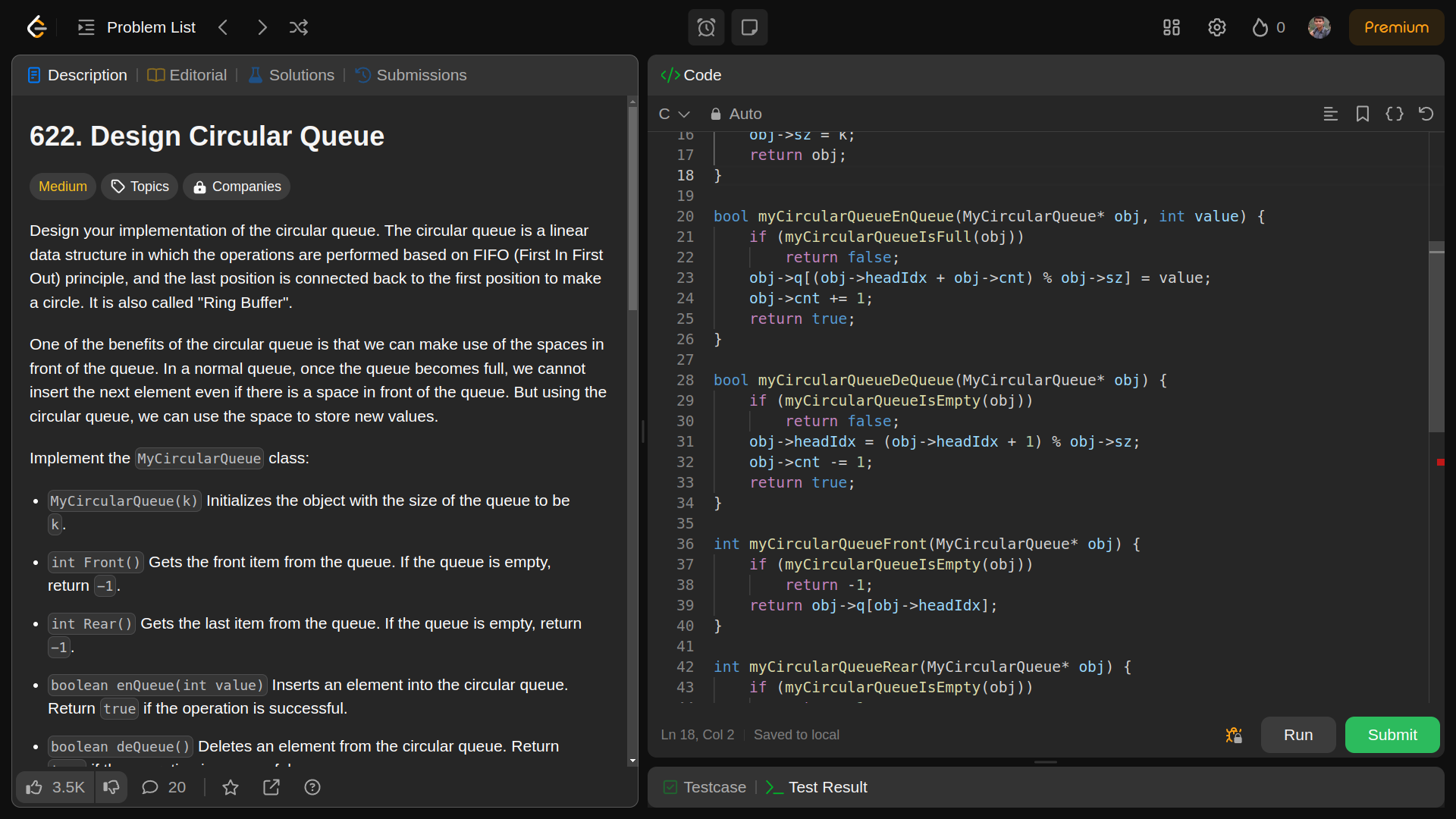
**void myCircularQueueFree(MyCircularQueue\* obj) {**

**free(obj->q);**

**free(obj);**

**}**

**OUTPUT:**



**LINKED LISTS :**

**6.REVERSE LINKED LIST , Level : Easy**

Given the head of a singly linked list, reverse the list, and return the reversed list.

Example 1:

Input: head = [1,2,3,4,5]  
Output: [5,4,3,2,1]

Example 2:

Input: head = [1,2]  
Output: [2,1]

Example 3:

Input: head = []  
Output: []

Constraints:

The number of nodes in the list is the range [0, 5000].

-5000 <= Node.val <= 5000

SOLUTION:

**/\*\***

**\* Definition for singly-linked list.**

**\* struct ListNode {**

**\* int val;**

**\* struct ListNode \*next;**

**\* };**

**\*/**

**struct ListNode\* reverseList(struct ListNode\* head) {**

**struct ListNode \*cur = head ,\*next = head , \*prev = NULL;**

**while(cur!=NULL)**

**{**

**next = cur->next;**

**cur->next = prev;**

**prev = cur;**

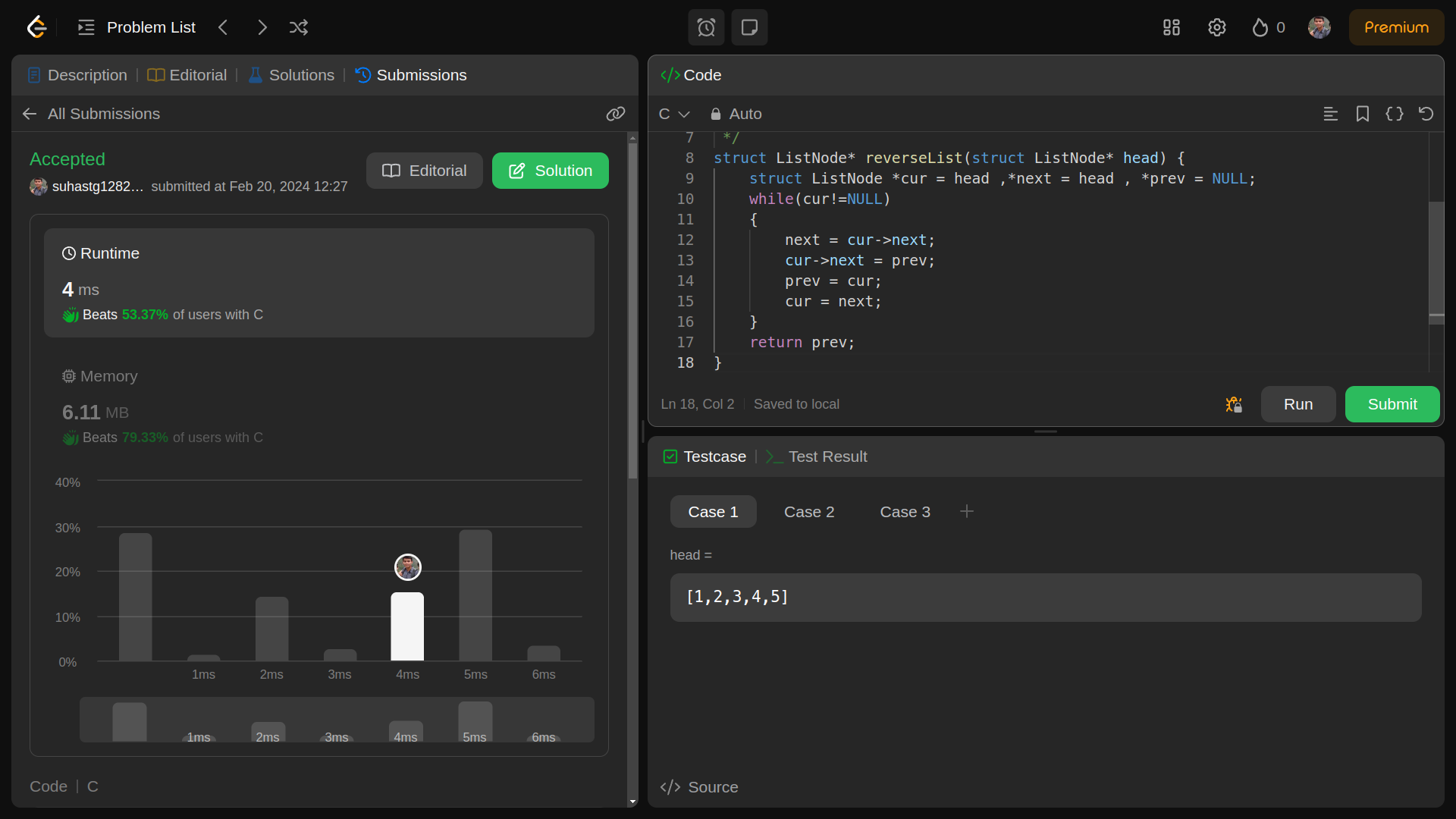
**cur = next;**

**}**

**return prev;**

**}**

**OUTPUT:**



**7 .ADD TWO NUMBERS , Level : MEDIUM**

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

**Example 1:**



**Input:** l1 = [2,4,3], l2 = [5,6,4]  
**Output:** [7,0,8]  
**Explanation:** 342 + 465 = 807.

**Example 2:**

**Input:** l1 = [0], l2 = [0]  
**Output:** [0]

**Example 3:**

**Input:** l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]  
**Output:** [8,9,9,9,0,0,0,1]

SOLUTION:

**#include <stdlib.h>**

**// Definition for singly-linked list.**

**struct ListNode {**

**int val;**

**struct ListNode \*next;**

**};**

**struct ListNode\* addTwoNumbers(struct ListNode\* l1, struct ListNode\* l2) {**

**struct ListNode\* dummy = (struct ListNode\*)malloc(sizeof(struct ListNode));**

**struct ListNode\* head = dummy;**

**struct ListNode\* tail = dummy;**

**int carry = 0;**

**while (l1 != NULL || l2 != NULL || carry != 0) {**

**int digit1 = (l1 != NULL) ? l1->val : 0;**

**int digit2 = (l2 != NULL) ? l2->val : 0;**

**int sum = digit1 + digit2 + carry;**

**int digit = sum % 10; // calculates unit of digit**

**carry = sum / 10;**

**struct ListNode\* res = (struct ListNode\*)malloc(sizeof(struct ListNode));**

**res->val = digit;**

**res->next = NULL;**

**tail->next = res;**

**tail = tail->next;**

**l1 = (l1 != NULL) ? l1->next : NULL;**

**l2 = (l2 != NULL) ? l2->next : NULL;**

**}**

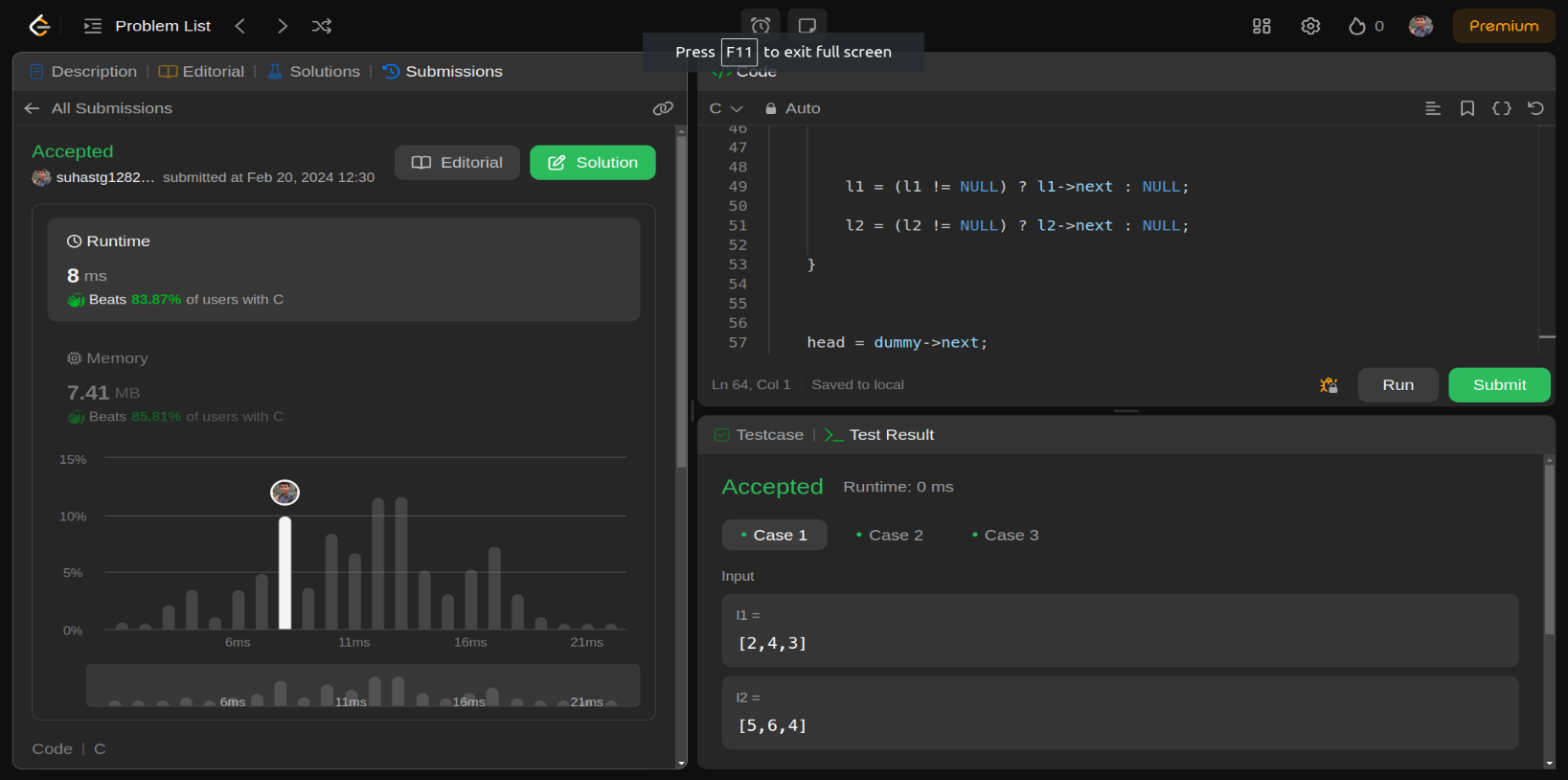
**head = dummy->next;**

**free(dummy);**

**return head;**

**}**

**OUTPUT:**



**TREES:**

**8. SAME TREE , Level:Easy**

Given the roots of two binary trees p and q, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

Example 1:

Input: p = [1,2,3], q = [1,2,3]  
Output: true

Example 2:

Input: p = [1,2], q = [1,null,2]  
Output: false

Example 3:

Input: p = [1,2,1], q = [1,1,2]  
Output: false

SOLUTION:

**/\*\***

**\* Definition for a binary tree node.**

**\* struct TreeNode {**

**\* int val;**

**\* struct TreeNode \*left;**

**\* struct TreeNode \*right;**

**\* };**

**\*/**

**bool isSameTree(struct TreeNode\* p, struct TreeNode\* q){**

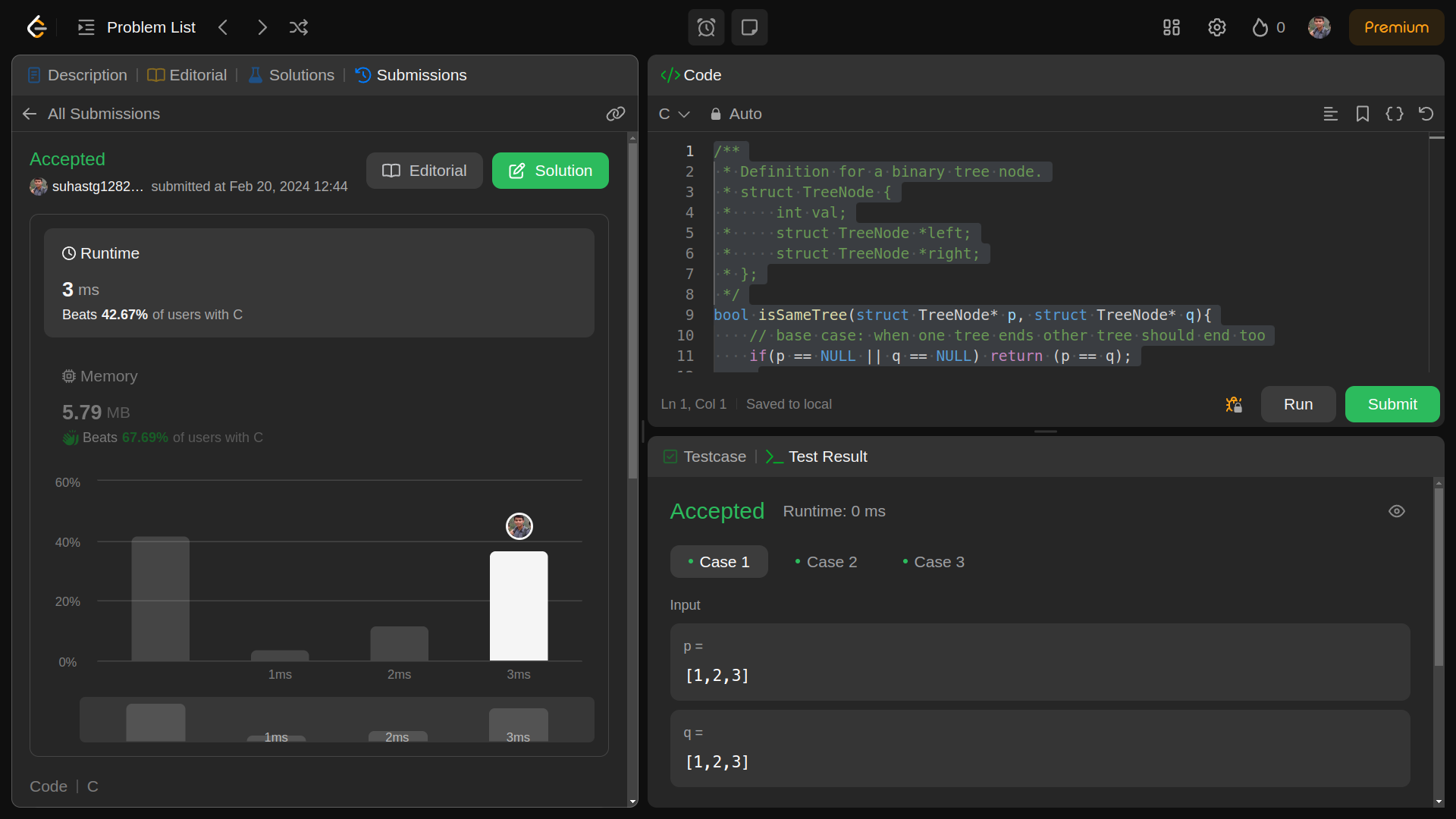
**// base case: when one tree ends other tree should end too**

**if(p == NULL || q == NULL) return (p == q);**

**// otherwise, the values should be equal at each node level.**

**return (p->val == q->val && isSameTree(p->left, q->left) && isSameTree(p->right, q->right));**

**}**

OUTPUT:  


**9.VALID BST ,Level:Medium**

Given the root of a binary tree, determine if it is a valid binary search tree (BST).

A valid BST is defined as follows:

The left

subtree

of a node contains only nodes with keys less than the node's key.

The right subtree of a node contains only nodes with keys greater than the node's key.

Both the left and right subtrees must also be binary search trees.

Example 1:

Input: root = [2,1,3]  
Output: true

Example 2:

Input: root = [5,1,4,null,null,3,6]  
Output: false  
Explanation: The root node's value is 5 but its right child's value is 4.

Constraints:

The number of nodes in the tree is in the range [1, 104].

-231 <= Node.val <= 231 - 1

SOLUTION:

**/\*\***

**\* Definition for a binary tree node.**

**\* struct TreeNode {**

**\* int val;**

**\* struct TreeNode \*left;**

**\* struct TreeNode \*right;**

**\* };**

**\*/**

**bool help(struct TreeNode \*root, long min, long max) {**

**if (root == NULL)**

**return true;**

**if (root->val <= min || root->val >= max)**

**return false;**

**if (!help(root->left, min, root->val) || !help(root->right, root->val, max))**

**return false;**

**return true;**

**}**

**bool isValidBST(struct TreeNode\* root) {**

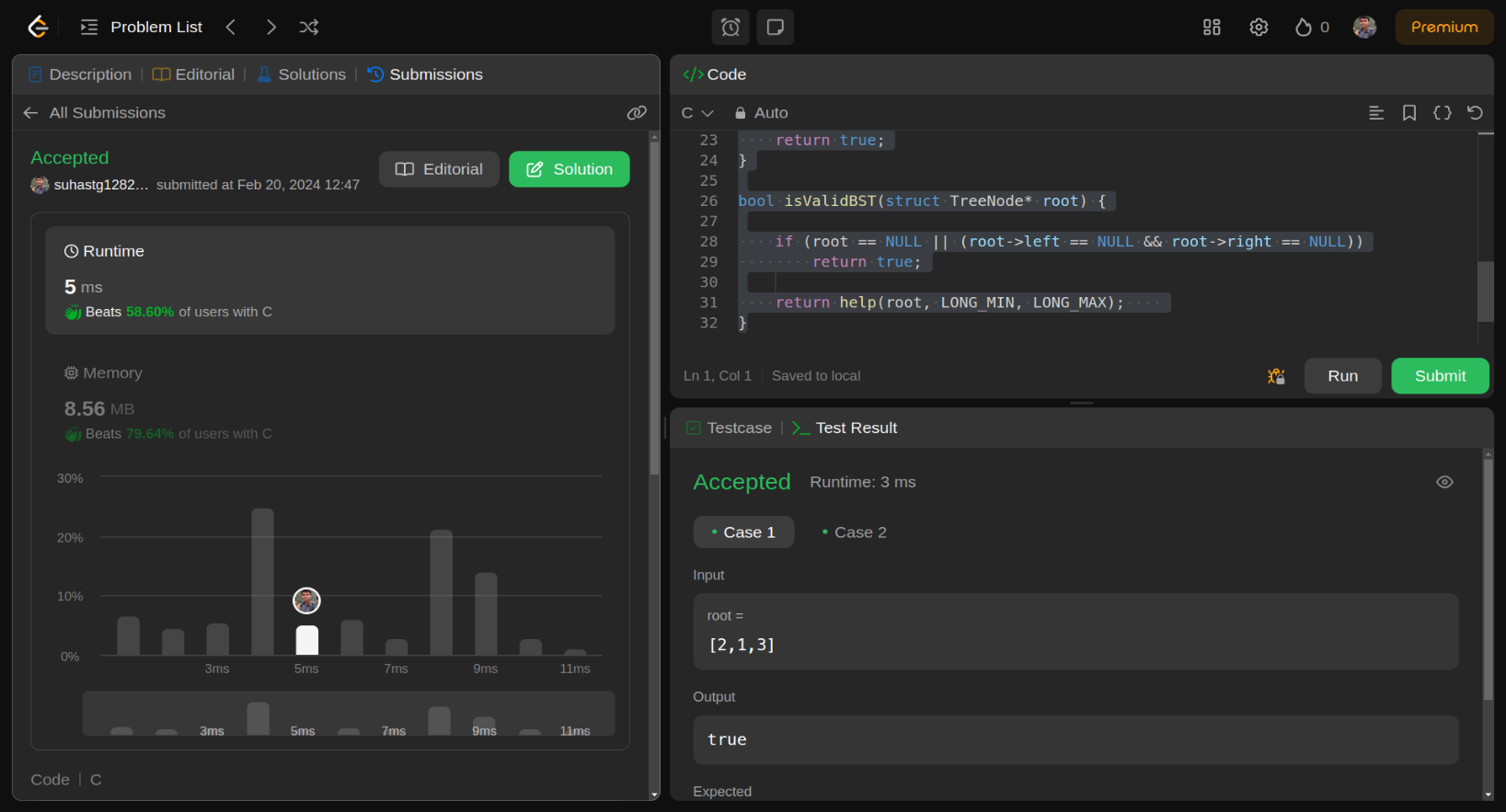
**if (root == NULL || (root->left == NULL && root->right == NULL))**

**return true;**

**return help(root, LONG\_MIN, LONG\_MAX);**

**}**

OUTPUT:



**10.DESIGN A HASHMAP , Level : Easy**

Design a HashMap without using any built-in hash table libraries.

Implement the MyHashMap class:

MyHashMap() initializes the object with an empty map.

void put(int key, int value) inserts a (key, value) pair into the HashMap. If the key already exists in the map, update the corresponding value.

int get(int key) returns the value to which the specified key is mapped, or -1 if this map contains no mapping for the key.

void remove(key) removes the key and its corresponding value if the map contains the mapping for the key.

Example 1:

Input  
["MyHashMap", "put", "put", "get", "get", "put", "get", "remove", "get"]  
[[], [1, 1], [2, 2], [1], [3], [2, 1], [2], [2], [2]]  
Output  
[null, null, null, 1, -1, null, 1, null, -1]  
  
Explanation  
MyHashMap myHashMap = new MyHashMap();  
myHashMap.put(1, 1); // The map is now [[1,1]]  
myHashMap.put(2, 2); // The map is now [[1,1], [2,2]]  
myHashMap.get(1); // return 1, The map is now [[1,1], [2,2]]  
myHashMap.get(3); // return -1 (i.e., not found), The map is now [[1,1], [2,2]]  
myHashMap.put(2, 1); // The map is now [[1,1], [2,1]] (i.e., update the existing value)  
myHashMap.get(2); // return 1, The map is now [[1,1], [2,1]]  
myHashMap.remove(2); // remove the mapping for 2, The map is now [[1,1]]  
myHashMap.get(2); // return -1 (i.e., not found), The map is now [[1,1]]

Constraints:

0 <= key, value <= 106

At most 104 calls will be made to put, get, and remove.

**SOLUTION:**

**#include <stdio.h>**

**#include <stdlib.h>**

**typedef struct {**

**int \*ans;**

**} MyHashMap;**

**MyHashMap\* myHashMapCreate() {**

**MyHashMap \*obj = (MyHashMap\*)malloc(sizeof(MyHashMap));**

**obj->ans = (int\*)malloc(1000001 \* sizeof(int));**

**for (int i = 0; i < 1000001; i++) {**

**obj->ans[i] = -1;**

**}**

**return obj;**

**}**

**void myHashMapPut(MyHashMap\* obj, int key, int value) {**

**obj->ans[key] = value;**

**}**

**int myHashMapGet(MyHashMap\* obj, int key) {**

**return obj->ans[key];**

**}**

**void myHashMapRemove(MyHashMap\* obj, int key) {**

**obj->ans[key] = -1;**

**}**

**void myHashMapFree(MyHashMap\* obj) {**

**free(obj->ans);**

**free(obj);**

**}**

**/\*\***

**\* Your MyHashMap struct will be instantiated and called as such:**

**\* MyHashMap\* obj = myHashMapCreate();**

**\* myHashMapPut(obj, key, value);**

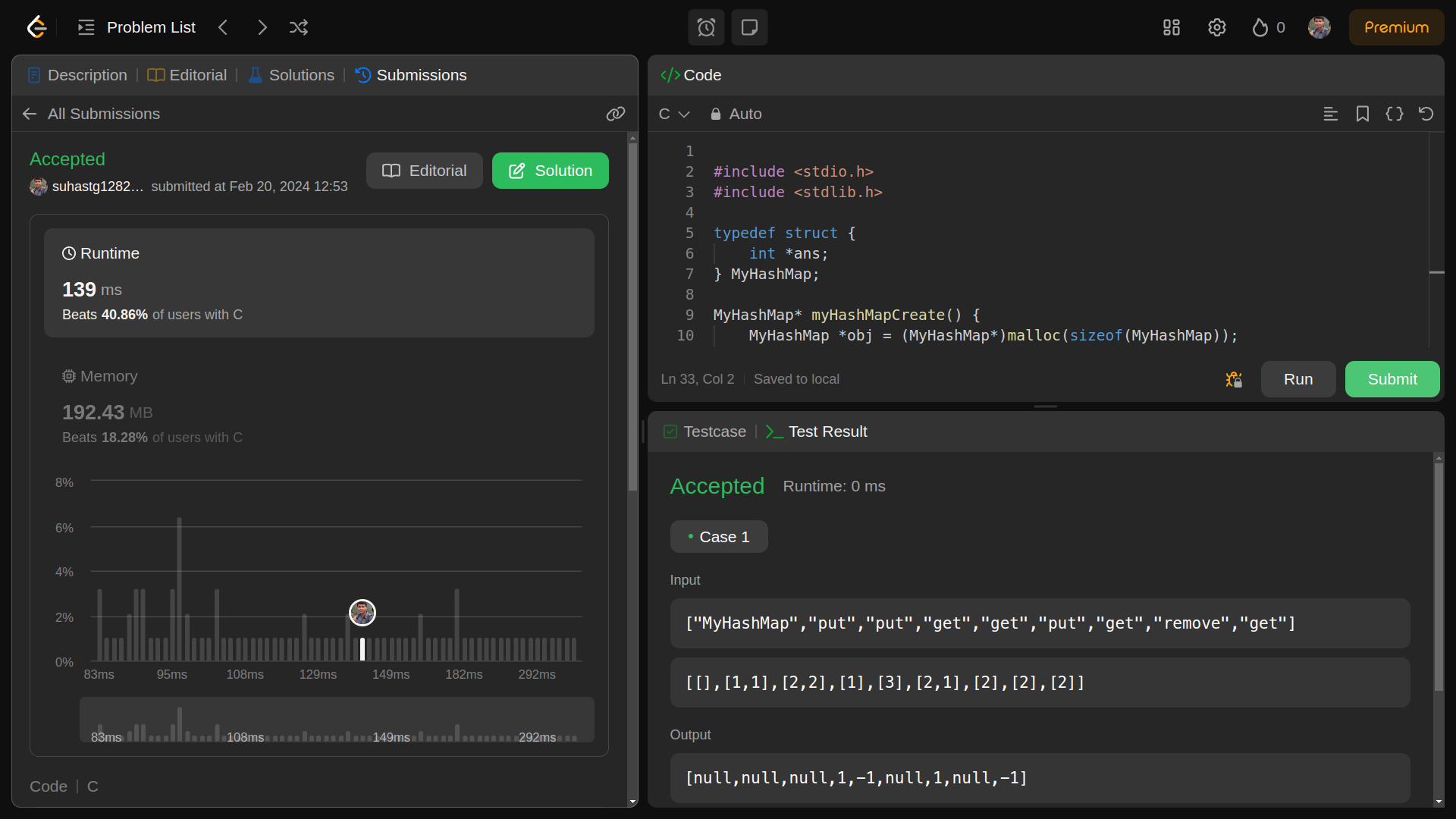
**\* int param\_2 = myHashMapGet(obj, key);**

**\* myHashMapRemove(obj, key);**

**\* myHashMapFree(obj);**

**\*/**

**OUTPUT:**



**REAL WORLD PROBLEM**

**Library-Management-System-using-Data-Structures**

Library management system is a simple console application using linked list in C programming language. User can perform basic library management operations like issuing books, returning the issued books and displaying records of the issued books with the user details.

Each book in the library has a unique identification number. The user issues the book by entering the book ID and the user details. Each user can issue only one book at a time. When the user returns the issued book, the book is available in the library for issuing again. The record of the issued book with user details can also be viewed.

CODE:

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <malloc.h>

#include <string.h>

struct book{

char name[30];

char author[30];

int id;

struct book \*next;

};

struct student{

char name[30];

char email[20];

char book[20];

char a[30];

int id;

struct student \*next;

};

struct book \*start\_lib=NULL;

struct student \*start=NULL;

struct book \*initialize\_lib(struct book \*);

struct student \*book\_issue(struct student \*);

struct student \*book\_return(struct student \*);

struct book \*diplay\_lib(struct book \*);

struct book \*delete\_book(int);

struct book \*add\_book(char [],char [],int);

void display(struct student \*);

void greetings();

void main\_menu();

int main(){

start\_lib=initialize\_lib(start\_lib);

greetings();

main\_menu();

return 0;

}

void greetings(){

printf("\n\n");

printf("\t\t\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t \* \*\n");

printf("\t\t\t \* \*\n");

printf("\t\t\t \* ---------------------------- \*\n");

printf("\t\t\t \* WELCOME TO STUDENT LIBRARY \*\n");

printf("\t\t\t \* ---------------------------- \*\n");

printf("\t\t\t \* \*\n");

printf("\t\t\t \* \*\n");

printf("\t\t\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n\n");

printf("\t\t\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t \* \*\n");

printf("\t\t\t \* ------------------------ \*\n");

printf("\t\t\t \* STUDENT LIBRARY \*\n");

printf("\t\t\t \* ------------------------ \*\n");

printf("\t\t\t \* \*\n");

printf("\t\t\t \* \*\n");

printf("\t\t\t \* Mumbai,Maharashtra,India \*\n");

printf("\t\t\t \* Email: studentlib@gmail.com \*\n");

printf("\t\t\t \* Contact:8800991010,8800992020 \*\n");

printf("\t\t\t \* \*\n");

printf("\t\t\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n\n\t\t\t Press any key to continue: ");

getch();

}

void main\_menu(){

int choice;

do{

printf("\n\n");

printf("\n\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n\t\t\t\t MAIN MENU: ");

printf("\n\t\t\t\t 1.ISSUE OF BOOKS ");

printf("\n\t\t\t\t 2.RETURN OF BOOKS ");

printf("\n\t\t\t\t 3.DISPLAY STUDENT DETAILS ");

printf("\n\t\t\t\t 4.EXIT\n ");

printf("\n\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n\t\t\t\t Enter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1:{

start=book\_issue(start);

break;

}

case 2:{

start=book\_return(start);

break;

}

case 3:{

display(start);

break;

}

case 4:{

exit(1);

}

default:{

printf("\n\t\t\t\t ...Invalid Option!...\n");

printf("\n\t\t\t\t Press any key to try again: ");

getch();

}

}

}while(choice!=4);

}

struct book \*initialize\_lib(struct book \*start){

struct book \*ptr,\*new\_book1,\*new\_book2,\*new\_book3,\*new\_book4,\*new\_book5;

new\_book1=(struct book \*)malloc(sizeof(struct book));

new\_book1->next=NULL;

start\_lib=new\_book1;

strcpy(new\_book1->name,"The Kite Runner");

strcpy(new\_book1->author,"Khaled Hosseini");

new\_book1->id=101;

ptr=new\_book1;

new\_book2=(struct book\*)malloc(sizeof(struct book));

new\_book2->next=NULL;

strcpy(new\_book2->name,"To Kill A Mockingbird");

strcpy(new\_book2->author,"Harper Lee");

new\_book2->id=102;

ptr->next=new\_book2;

ptr=new\_book2;

new\_book3=(struct book\*)malloc(sizeof(struct book));

new\_book3->next=NULL;

strcpy(new\_book3->name,"The Alchemist");

strcpy(new\_book3->author,"Paulo Coelho");

new\_book3->id=103;

ptr->next=new\_book3;

ptr=new\_book3;

new\_book4=(struct book\*)malloc(sizeof(struct book));

new\_book4->next=NULL;

strcpy(new\_book4->name,"Pride And Prejudice");

strcpy(new\_book4->author,"Jane Austen");

new\_book4->id=104;

ptr->next=new\_book4;

ptr=new\_book4;

new\_book5=(struct book\*)malloc(sizeof(struct book));

new\_book5->next=NULL;

strcpy(new\_book5->name,"A Tale Of Two Cities");

strcpy(new\_book5->author,"Charles Dickens");

new\_book5->id=105;

ptr->next=new\_book5;

return start\_lib;

}

struct student \*book\_issue(struct student \*start){

struct book \*ptr;

struct student \*ptr2,\*new\_student;

int i=1,id,flag=0;

if(start\_lib==NULL){

printf("\n\t\t\t\t No books left in the library to issue!\n\t\t\t\t Sorry for the inconvenience!\n");

}else{

system("cls");

ptr=start\_lib;

printf("\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Books Available: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

while(ptr!=NULL){

printf("\n\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n\t Book %d",i);

printf("\n\t Book Title: %s",ptr->name);

printf("\n\t Name of Author: %s",ptr->author);

printf("\n\t Book ID: %d",ptr->id);

printf("\n\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

ptr=ptr->next;

i++;

}

printf("\n\n\t Enter the Book ID: ");

scanf("%d",&id);

ptr=start\_lib;

while(ptr!=NULL){

if(ptr->id==id){

flag=1;

break;

}

ptr=ptr->next;

}

if(flag==1){

ptr=start\_lib;

while(ptr->id!=id){

ptr=ptr->next;

}

new\_student=(struct student \*)malloc(sizeof(struct student));

printf("\n\t Enter Student Details:\n ");

printf("\n\t Enter your Name: ");

scanf("%s",new\_student->name);

printf("\n\t Enter your Email: ");

scanf("%s",new\_student->email);

strcpy(new\_student->book,ptr->name);

strcpy(new\_student->a,ptr->author);

new\_student->id=ptr->id;

new\_student->next=NULL;

printf("\n\t Issue of Book ID %d done successfully!\n",new\_student->id);

printf("\n\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

if(start==NULL){

start=new\_student;

}else{

ptr2=start;

while(ptr2->next!=NULL){

ptr2=ptr2->next;

}

ptr2->next=new\_student;

}

start\_lib=delete\_book(new\_student->id);

printf("\n\n\t Press any key to go to the main menu: ");

getch();

system("cls");

}else{

printf("\n\t\t ...Invalid Option!...\n");

printf("\n\t\t Press any key to try again: ");

getch();

system("cls");

}

}

return start;

}

struct student \*book\_return(struct student \*start){

struct student \*ptr,\*preptr;

char bookname[30],authorname[30];

int flag=0,id,identity,c=0,d=1;

printf("\n\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Books Submission: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n\n\t Enter your Book ID: ");

scanf("%d",&identity);

ptr=start;

while(ptr!=NULL){

if(ptr->id==identity){

flag=1;

break;

}

ptr=ptr->next;

}

if(flag==1){

ptr=start;

while(ptr!=NULL){

c++;

ptr=ptr->next;

}

ptr=start;

while(ptr->id!=identity){

d++;

ptr=ptr->next;

}

ptr=start;

if( d==1 ){

printf("\n\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n\t Student Name: %s",start->name);

printf("\n\t Student Email: %s",start->email);

printf("\n\t Name of Book Issued: %s",start->book);

printf("\n\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n\n\t Return of Book ID %d done successfully!\n",identity);

printf("\n\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

strcpy(bookname,start->book);

strcpy(authorname,start->a);

id=start->id;

start=start->next;

free(ptr);

add\_book(bookname,authorname,id);

}else{

ptr=start;

while(ptr->id!=identity){

preptr=ptr;

ptr=ptr->next;

}

printf("\n\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n\t Student Name: %s",ptr->name);

printf("\n\t Student Email: %s",ptr->email);

printf("\n\t Name of Book Issued: %s",ptr->book);

printf("\n\t Book ID: %d",ptr->id);

printf("\n\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

strcpy(bookname,ptr->book);

strcpy(authorname,ptr->a);

id=ptr->id;

preptr->next=ptr->next;

free(ptr);

add\_book(bookname,authorname,id);

}

printf("\n\t Thank you! \n\t Do visit again! ");

printf("\n\n\t Press any key to go to the main menu: ");

getch();

system("cls");

}else{

printf("\n\tSorry the book doesn't exist! Please recheck the entered ID");

printf("\n\t\t\t\t Press any key to try again: ");

getch();

system("cls");

}

return start;

}

void display(struct student \*start){

struct student \*ptr;

ptr=start;

while(ptr!=NULL){

printf("\n\t\*\*\*\*\*\*\*\*\*\*\*\*\* Details of Students: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n\t\t Student Name: %s",ptr->name);

printf("\n\t\t Student Email: %s",ptr->email);

printf("\n\t\t Name of Book Issued: %s",ptr->book);

printf("\n\t\t Book ID: %d",ptr->id);

printf("\n\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

ptr=ptr->next;

}

printf("\n\n\t Press any key to go to the main menu: ");

getch();

system("cls");

}

struct book \*delete\_book(int id){

struct book \*ptr,\*preptr;

int c=0;

ptr=start\_lib;

while(ptr!=NULL){

c++;

ptr=ptr->next;

}

if(c==1){

ptr=start\_lib;

start\_lib=NULL;

free(ptr);

}else if(start\_lib->id==id){

ptr=start\_lib;

start\_lib=start\_lib->next;

free(ptr);

}else{

ptr=start\_lib;

while(ptr->id!=id){

preptr=ptr;

ptr=ptr->next;

}

preptr->next=ptr->next;

free(ptr);

}

return start\_lib;

}

struct book \*add\_book(char bookname[30],char authorname[30],int id){

struct book \*ptr,\*new\_book;

new\_book=(struct book \*)malloc(sizeof(struct book));

strcpy(new\_book->name,bookname);

strcpy(new\_book->author,authorname);

new\_book->id=id;

new\_book->next=NULL;

if(start\_lib==NULL){

start\_lib=new\_book;

}else{

ptr=start\_lib;

while(ptr->next!=NULL){

ptr=ptr->next;

}

ptr->next=new\_book;

}

return start\_lib;

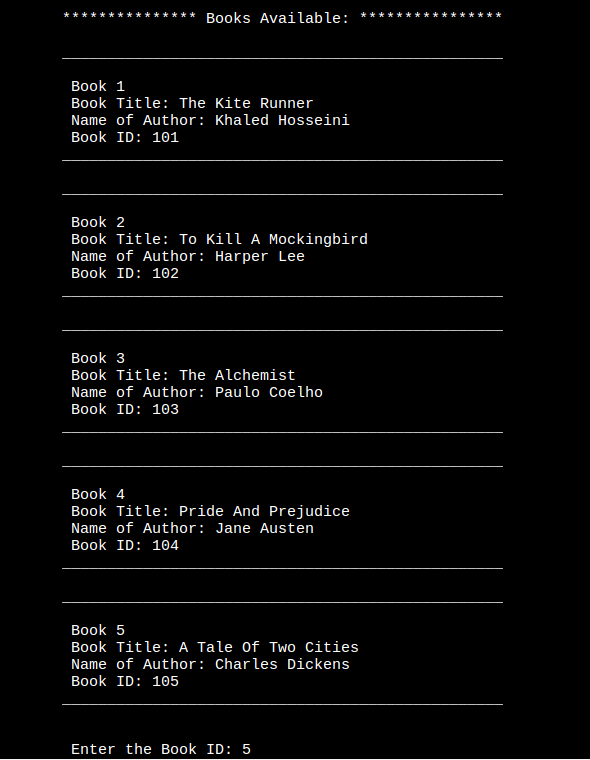
}

**OUTPUT:**

1.Welcome page:



2.Display of Book Available:



3.Entering Student Details and Issue

