**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**RACHIT SINHA (1BM23CS256)**

**in partial fulfillment for the award of the degree of**

**BACHELOR OF ENGINEERING**

**in**

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**September 2024-January 2025**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

**(Affiliated To Visvesvaraya Technological University, Belgaum)**

**Department of Computer Science and Engineering**

****

This is to certify that the Lab work entitled **“DATA STRUCTURES”** carried out by **RACHIT SINHA** **(1BM23CS256)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)** work prescribed for the said degree.

**Prof. Lakshmi Neelima M** **Dr. Kavitha Sooda**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl.**  **No.** | | **Experiment Title** | | **Page No.** |
| 1 | | Working of Stack | | 4 |
| 2 | | Infix to Postfix | | 7 |
| 3 | | Working of Linear Queue & Circular Queue | | 10 |
| 4 | | Single Linked List – Insertion, Deletion, Display | | 18 |
| 5 | | Single Linked List – Sort, Reverse and Concatenate | | 24 |
| 6 | | Stack & Queue Using Linked List | | 30 |
| 7 | | Double Linked List - Insertion, Deletion, Display | | 35 |
| 8 | | Binary Search Tree | | 40 |
| 9 | | BFS DFS traversal | | 43 |
| 10 | | Hashing Linear Probing | | 47 |
| **LeetCode Programs** | | | | |
| 1 | Move Zeroes | | 50 | |
| 2 | Majority Element | | 51 | |
| 3 | Game of 2 Stacks | | 52 | |
| 4 | Palindrome | | 53 | |
| 5 | Path Sum | | 54 | |

**Course outcomes:**

|  |  |
| --- | --- |
| CO1 | Apply the concept of linear and nonlinear data structures. |
| CO2 | Analyze data structure operations for a given problem |
| CO3 | Design and develop solutions using the operations of linear and nonlinear data structure for a given specification. |
| CO4 | Conduct practical experiments for demonstrating the operations of different data structures. |

**Lab program 1:**

**Write a program to simulate the working of stack using an array with the following:**

**a) Push**

**b) Pop**

**c) Display**

**The program should print appropriate messages for stack overflow, stack underflow.**

#include <stdio.h>

#include<stdlib.h>

#define STACK\_SIZE 5

void push(int st[],int \*top)

{

int item;

if(\*top==STACK\_SIZE-1)

printf("Stack overflow\n");

else

{

printf("\nEnter an item :");

scanf("%d",&item);

(\*top)++;

st[\*top]=item;

}

}

void pop(int st[],int \*top)

{

if(\*top==-1)

printf("Stack underflow\n");

else

{

printf("\n%d item was deleted",st[(\*top)--]);

}

}

void display(int st[],int \*top)

{

int i;

if(\*top==-1)

printf("Stack is empty\n");

for(i=0;i<=\*top;i++)

printf("%d\t",st[i]);

}

void main()

{

int st[10],top=-1, c,val\_del;

while(1)

{

printf("\n1. Push\n2. Pop\n3. Display\n");

printf("\nEnter your choice :");

scanf("%d",&c);

switch(c)

{

case 1: push(st,&top);

break;

case 2: pop(st,&top);

break;

case 3: display(st,&top);

break;

default: printf("\nInvalid choice!!!");

exit(0);

}

}

}

**Output:**

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A computer screen with white text

Description automatically generated

**Lab program 2:**

**Write a Program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), \* (multiply) and / (divide)**

#include <stdio.h>

#define MAX 100

char stack[MAX];

int top = -1;

void push(char c) {

if (top < MAX - 1) {

stack[++top] = c;

}

}

char pop() {

return (top >= 0) ? stack[top--] : '\0';

}

char peek() {

return (top >= 0) ? stack[top] : '\0';

}

int precedence(char c) {

if (c == '+' || c == '-') return 1;

if (c == '\*' || c == '/') return 2;

return 0;

}

int isOperator(char c) {

return c == '+' || c == '-' || c == '\*' || c == '/';

}

int isOperand(char c){

return(c>='0'&&c<='9')||(c>='A'&&c<='Z')||(c>='a'&&c<='z');

}

void infixToPostfix(char\* infix, char\* postfix) {

int j = 0; // Postfix index

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

char c = infix[i];

if (isOperand(c)) { // Operand

postfix[j++] = c;

} else if (c == '(') {

push(c);

} else if (c == ')') {

while (peek() != '(') {

postfix[j++] = pop();

}

pop(); // Remove '(' from stack

} else if (isOperator(c)) {

while (top != -1 && precedence(peek()) >= precedence(c)) {

postfix[j++] = pop();

}

push(c);

}

}

while (top != -1) {

postfix[j++] = pop();

}

postfix[j] = '\0'; // Null-terminate the postfix expression

}

int main() {

char infix[MAX], postfix[MAX];

printf("Enter a valid parenthesized infix expression: ");

fgets(infix, sizeof(infix), stdin);

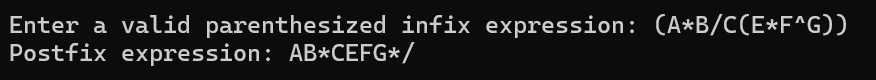
infixToPostfix(infix, postfix);

printf("Postfix expression: %s\n", postfix);

return 0;

}

**OUTPUT:**

****

**Lab Program 3:**

1. **WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert,Delete, Display. The program should print appropriate messages for queue empty and queue overflow conditions.**

#include<stdio.h>

#define MAX 5

void enqueue(int queue[], int \*front, int \*rear, int value)

{

if(\*rear==MAX-1)

printf("Queue is full. Element cannot be inserted.\n");

else

{

if(\*front==-1)

\*front=0;

(\*rear)++;

queue[\*rear]=value;

printf("The element has been inserted.\n");

}

}

int dequeue(int queue[], int \*front, int \*rear)

{

if(\*front==-1||\*front>\*rear)

{

printf("Queue is empty. cannot delete element from the queue.\n");

return -1;

}

int deletedvalue=queue[\*front];

(\*front)++;

return deletedvalue;

if(\*front>\*rear)

{

\*front=\*rear=-1;

}

}

void display(int queue[], int front, int rear)

{

if(front==-1||front>rear)

{

printf("Queue is empty. Cannot display elements");

}

else

{

for(int i=front; i<=rear; i++)

{

printf("%d", queue[i]);

}

printf("\n");

}

}

int main()

{

int choice, value;

int queue[MAX];

int front=-1;

int rear=-1;

do

{

printf("Queue Menu:\n");

printf("1. Insert\n");

printf("2.Delete\n");

printf("Display\n");

printf("4.Exit\n");

printf("Enter your choice\n");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Enter the element to be inserted:");

scanf("%d", &value);

enqueue(queue, &front, &rear, value);

break;

case 2:

value=dequeue(queue, &front, &rear);

if(value!=-1)

{

printf("The deleted element is %d\n", value);

}

break;

case 3:

display(queue, front, rear);

break;

case 4:

printf("Exited the program\n");

break;

default:

printf("Enter a valid choice\n");

break;

}

}while(choice!=4);

}

**OUTPUT:**

**A screenshot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions**

#include <stdio.h>

#define SIZE 3

int queue[SIZE];

int front = -1, rear = -1;

int is\_full() {

return (front == (rear + 1) % SIZE);

}

int is\_empty() {

return (front == -1);

}

void insert(int n) {

if (is\_full()) {

printf("Queue Overflow\n");

return;

}

if (is\_empty())

front = 0;

rear = 0;

else

rear = (rear + 1) % SIZE;

queue[rear] = n;

printf("Element %d inserted.\n", n);

}

void delete() {

if (is\_empty()) {

printf("Queue Underflow.\n");

return;

}

printf("Element %d deleted\n", queue[front]);

if (front == rear){

front = -1;

rear = -1;

}

else

front = (front + 1) % SIZE;

}

void display() {

if (is\_empty()) {

printf("Queue is empty\n");

return;

}

printf("Queue elements: ");

int i = front;

while (1) {

printf("%d ", queue[i]);

if (i == rear)

break;

i = (i + 1) % SIZE;

}

printf("\n");

}

int main() {

int choice, value;

printf("\nCircular Queue Operations:\n");

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Display\n");

printf("4. Exit\n");

while (1) {

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to insert: ");

scanf("%d", &value);

insert(value);

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

printf("Exiting...\n");

return 0;

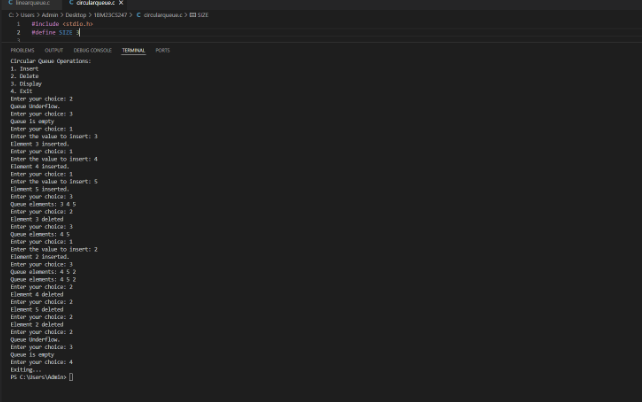
default:

printf("Invalid choice! Please try again.\n");

}

}

}



**Lab Program 4**

**WAP to Implement Singly Linked List with following operations**

**a) Create LinkedList.**

**b) Insertion of a node at first position, at any position and at end of list.**

**c) Deletion of first element, specified element and last element in the list.**

**Display the contents of the linked list**.

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* create\_node(int data) {

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

new\_node->data = data;

new\_node->next = NULL;

return new\_node;

}

void insert\_at\_beginning(struct Node\*\* head, int data) {

struct Node\* new\_node = create\_node(data);

new\_node->next = \*head;

\*head = new\_node;

}

void insert\_at\_end(struct Node\*\* head, int data) {

struct Node\* new\_node = create\_node(data);

if (\*head == NULL) {

\*head = new\_node;

return;

}

struct Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = new\_node;

}

void insert\_at\_position(struct Node\*\* head, int data, int position) {

if (position < 0) return;

if (position == 0) {

insert\_at\_beginning(head, data);

return;

}

struct Node\* new\_node = create\_node(data);

struct Node\* temp = \*head;

for (int i = 0; i < position - 1; i++) {

if (temp == NULL) return;

temp = temp->next;

}

new\_node->next = temp->next;

temp->next = new\_node;

}

void delete\_at\_beginning(struct Node\*\* head) {

if (\*head != NULL) {

struct Node\* temp = \*head;

\*head = (\*head)->next;

free(temp);

}

}

void delete\_at\_end(struct Node\*\* head) {

if (\*head == NULL) return;

if ((\*head)->next == NULL) {

free(\*head);

\*head = NULL;

return;

}

struct Node\* temp = \*head;

while (temp->next && temp->next->next != NULL) {

temp = temp->next;

}

free(temp->next);

temp->next = NULL;

}

void delete\_at\_key(struct Node\*\* head, int key) {

if (\*head == NULL) return;

if ((\*head)->data == key) {

struct Node\* temp = \*head;

\*head = (\*head)->next;

free(temp);

return;

}

struct Node\* temp = \*head;

while (temp->next != NULL && temp->next->data != key) {

temp = temp->next;

}

if (temp->next == NULL) return;

struct Node\* to\_delete = temp->next;

temp->next = temp->next->next;

free(to\_delete);

}

void delete\_before\_key(struct Node\*\* head, int key) {

if (\*head == NULL || (\*head)->next == NULL) return;

if ((\*head)->next->data == key) {

struct Node\* temp = \*head;

\*head = (\*head)->next;

free(temp);

return;

}

struct Node\* temp = \*head;

while (temp->next != NULL && temp->next->next != NULL) {

if (temp->next->next->data == key) {

struct Node\* to\_delete = temp;

temp = temp->next;

free(to\_delete);

return;

}

temp = temp->next;

}

}

void delete\_after\_key(struct Node\*\* head, int key) {

struct Node\* temp = \*head;

while (temp != NULL && temp->data != key) {

temp = temp->next;

}

if (temp != NULL && temp->next != NULL) {

struct Node\* to\_delete = temp->next;

temp->next = temp->next->next;

free(to\_delete);

}

}

void display(struct Node\* head) {

struct Node\* temp = head;

while (temp != NULL) {

printf("%d -> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

int data, key;

printf("Choice :\n1.insert\_at\_beginning\n2.insert\_at\_end\n3.insert\_at\_position\n4.delete\_at\_beginning\n5.delete\_at\_end\n6.delete\_at\_key\n7.delete\_before\_key\n8.delete\_after\_key\n9.display\n10.exit\n");

int c;

while (1) {

printf("Enter choice: ");

scanf("%d", &c);

switch (c) {

case 1:

printf("Enter the data: ");

scanf("%d", &data);

insert\_at\_beginning(&head, data);

break;

case 2:

printf("Enter the data: ");

scanf("%d", &data);

insert\_at\_end(&head, data);

break;

case 3:

printf("Enter the data and position: ");

scanf("%d%d", &data, &key);

insert\_at\_position(&head, data, key);

break;

case 4:

delete\_at\_beginning(&head);

break;

case 5:

delete\_at\_end(&head);

break;

case 6:

printf("Enter the key to delete: ");

scanf("%d", &key);

delete\_at\_key(&head, key);

break;

case 7:

printf("Enter the key to delete before: ");

scanf("%d", &key);

delete\_before\_key(&head, key);

break;

case 8:

printf("Enter the key to delete after: ");

scanf("%d", &key);

delete\_after\_key(&head, key);

break;

case 9:

display(head);

break;

case 10:

exit(0);

default:

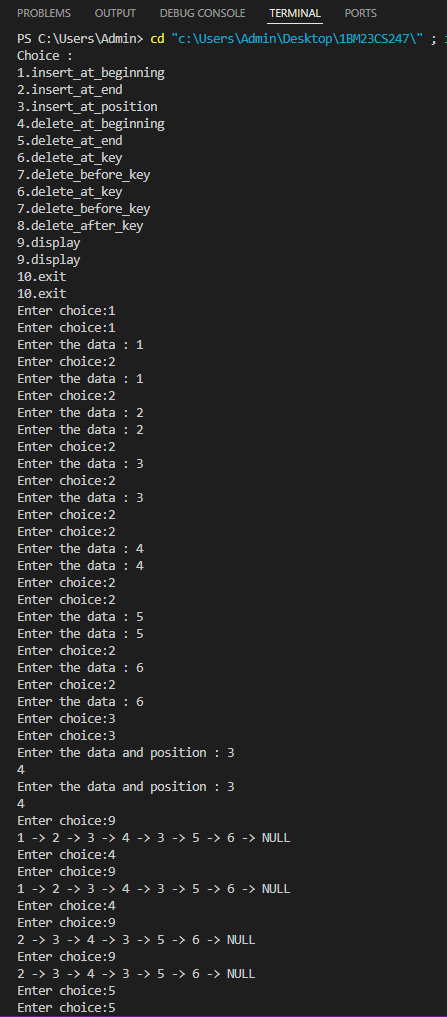
printf("Invalid choice...\n");

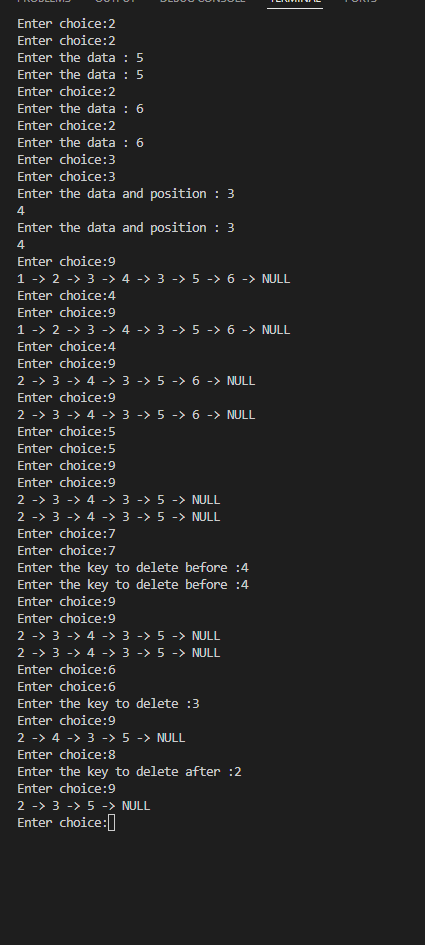
}

}

return 0;

}





**LAB PROGRAM 5:**

**WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*CreateNode(int data)

{

struct node \*newnode = (struct node \*)malloc(sizeof(struct node));

newnode->data = data;

newnode->next = NULL;

return newnode;

}

void insertatfirst(struct node \*\*head, int data)

{

struct node \*newnode = CreateNode(data);

newnode->next = \*head;

\*head = newnode;

}

struct node \*Concate(struct node \*head1,struct node \*head2){

struct node \*temp = head1;

while(temp->next!=NULL){

temp = temp->next;

}

temp->next = head2;

return head1;

};

struct node \*Sort(struct node \*head){

struct node \*temp,\*current;

current = head;

while(current!=NULL){

temp = head;

while(temp->next!=NULL){

if(temp->data > temp->next->data){

int t = temp->data;

temp->data = temp->next->data;

temp->next->data = t;

}

temp = temp->next;

}

current = current->next;

}

return head;

}

struct node \*Reverse(struct node \*head){

struct node \*temp,\*prev,\*next;

prev = NULL;

temp = head;

while(temp!=NULL){

next = temp->next;

temp->next = prev;

prev = temp;

temp = next;

}

head = prev;

return head;

}

void display(struct node \*\*head)

{

if (\*head == NULL)

{

printf("Empty");

}

else

{

struct node \*temp = \*head;

while (temp != NULL)

{

printf("%d->", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

}

int main() {

struct node \*head1 = NULL;

struct node \*head2 = NULL;

int choice, value;

printf("Enter 5 elements for Linked List 1:\n");

for (int i = 0; i < 5; i++) {

printf("Enter element %d: ", i + 1);

scanf("%d", &value);

insertatfirst(&head1, value);

}

printf("Enter 5 elements for Linked List 2:\n");

for (int i = 0; i < 5; i++) {

printf("Enter element %d: ", i + 1);

scanf("%d", &value);

insertatfirst(&head2, value);

}

printf("Initial Linked Lists:\n");

display(&head1);

display(&head2);

do {

printf("\nChoose an operation:\n");

printf("1. Sort\n");

printf("2. Reverse\n");

printf("3. Concatenate\n");

printf("4. Display Linked Lists\n");

printf("0. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

head1 = Sort(head1);

head2 = Sort(head2);

printf("Sorted Linked Lists:\n");

display(&head1);

display(&head2);

break;

case 2:

head1 = Reverse(head1);

head2 = Reverse(head2);

printf("reversed Linked Lists:\n");

display(&head1);

display(&head2);

break;

case 3:

head1 = Concate(head1, head2);

printf("concatenated Linked Lists:\n");

display(&head1);

display(&head2);

break;

case 4:

printf("Current Linked Lists:\n");

display(&head1);

break;

case 0:

printf("Exiting program.\n");

break;

default:

printf("Invalid choice. Please try again.\n");

}

} while (choice != 0);

return 0;

}

**OUTPUT:**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**LAB PROGRAM 6:**

**WAP to Implement Single Link List to simulate Stack & Queue Operations.**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*next;

};

struct node \*front = NULL, \*rear = NULL;

struct node \*head = NULL;

void enqueue(int val) {

struct node \*newNode = malloc(sizeof(struct node));

newNode->data = val;

newNode->next = NULL;

if (front == NULL && rear == NULL) {

front = rear = newNode;

} else {

rear->next = newNode;

rear = newNode;

}

}

void dequeue() {

struct node \*temp;

if (front == NULL) {

printf("Queue is Empty. Unable to perform dequeue.\n");

} else {

temp = front;

front = front->next;

if (front == NULL) {

rear = NULL;

}

printf("Dequeued element = %d\n", temp->data);

free(temp);

}

}

void printQueue() {

struct node \*temp = front;

printf("Queue: ");

while (temp) {

printf("%d->", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

void push(int val) {

struct node \*newNode = malloc(sizeof(struct node));

newNode->data = val;

newNode->next = head;

head = newNode;

}

void pop() {

struct node \*temp;

if (head == NULL) {

printf("Stack is Empty.\n");

} else {

printf("Popped element = %d\n", head->data);

temp = head;

head = head->next;

free(temp);

}

}

void printStack() {

struct node \*temp = head;

printf("Stack: ");

while (temp != NULL) {

printf("%d->", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

int main() {

int choice, value;

do {

printf("\nSelect an operation:\n");

printf("1. Enqueue \n");

printf("2. Dequeue \n");

printf("3. Push \n");

printf("4. Pop \n");

printf("5. Queue\n");

printf("6. Stack\n");

printf("0. Exit\n");

printf("Enter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to enqueue: ");

scanf("%d", &value);

enqueue(value);

break;

case 2:

dequeue();

break;

case 3:

printf("Enter value to push: ");

scanf("%d", &value);

push(value);

break;

case 4:

pop();

break;

case 5:

printQueue();

break;

case 6:

printStack();

break;

case 0:

printf("Exiting program.\n");

break;

default:

printf("Invalid choice. Please try again.\n");

}

} while (choice != 0);

return 0;

}

**OUTPUT:**

**A screenshot of a computer program

Description automatically generated A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**LAB PROGRAM 7:**

**WAP to Implement doubly link list with primitive operations  
  
a) Create a doubly linked list.  
b) Insert a new node to the left of the node.  
c) Delete the node based on a specific value  
Display the contents of the list**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

struct Node\* prev;

};

void insertFront(struct Node\*\* head, int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = (\*head);

newNode->prev = NULL;

if ((\*head) != NULL)

(\*head)->prev = newNode;

(\*head) = newNode;

}

void deleteAtPosition(struct Node\*\* head, int position)

{

if (\*head == NULL) {

printf("The list is already empty.\n");

return;

}

struct Node\* temp = \*head;

if (position == 1) {

if (\*head == NULL) {

printf("The list is already empty.\n");

return;

}

struct Node\* temp = \*head;

\*head = (\*head)->next;

if (\*head != NULL) {

(\*head)->prev = NULL;

}

free(temp);

return;

}

for (int i = 1; temp != NULL && i < position; i++) {

temp = temp->next;

}

if (temp == NULL) {

printf("Position is greater than the number of "

"nodes.\n");

return;

}

if (temp->next != NULL) {

temp->next->prev = temp->prev;

}

if (temp->prev != NULL) {

temp->prev->next = temp->next;

}

free(temp);

}

void displayList(struct Node\* node) {

struct Node\* last;

while (node != NULL) {

printf("%d->", node->data);

last = node;

node = node->next;

}

if (node == NULL)

printf("NULL\n");

}

int main() {

struct Node\* head = NULL; // Initialize the doubly linked list as empty

int choice, data, position;

while (1) {

printf("\n--- Doubly Linked List Operations ---\n");

printf("1. Insert at Front\n");

printf("2. Delete at Position\n");

printf("3. Display List\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert: ");

scanf("%d", &data);

insertFront(&head, data);

printf("Node inserted at the front.\n");

break;

case 2:

22 printf("Enter the position to delete: ");

scanf("%d", &position);

deleteAtPosition(&head, position);

break;

case 3:

printf("The list is: ");

displayList(head);

break;

case 4:

printf("Exiting the program.\n");

// Free all remaining nodes in the list before exiting

while (head != NULL) {

struct Node\* temp = head;

head = head->next;

free(temp);

}

return 0;

default:

printf("Invalid choice. Please try again.\n");

}

}

return 0;

}

**OUTPUT:**

**A screenshot of a computer program

Description automatically generated**

**A black screen with white text

Description automatically generated**

**Lab Program 8**

**Write a program**

**a) To construct a binary Search tree.**

**b) To traverse the tree using all the methods i.e., inorder, preorder and post order**

**c) To display the elements in the tree.**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;

}

struct Node\* insert(struct Node\* root, int data) {

if (root == NULL) {

return createNode(data);

}

if (data < root->data) {

root->left = insert(root->left, data);

} else if (data > root->data) {

root->right = insert(root->right, data);

}

return root;

}

void inorderTraversal(struct Node\* root) {

if (root == NULL) {

return;

}

inorderTraversal(root->left);

printf("%d ", root->data);

inorderTraversal(root->right);

}

void preorderTraversal(struct Node\* root) {

if (root == NULL) {

return;

}

printf("%d ", root->data);

preorderTraversal(root->left);

preorderTraversal(root->right);

}

void postorderTraversal(struct Node\* root) {

if (root == NULL) {

return;

}

postorderTraversal(root->left);

postorderTraversal(root->right);

printf("%d ", root->data);

}

int main() {

struct Node\* root = NULL;

int choice, data;

while (1) {

printf("\nBinary Search Tree Operations:\n");

printf("1. Insert a node\n");

printf("2. In-order traversal\n");

printf("3. Pre-order traversal\n");

printf("4. Post-order traversal\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to insert: ");

scanf("%d", &data);

root = insert(root, data);

printf("Node %d inserted.\n", data);

break;

case 2:

printf("In-order traversal: ");

inorderTraversal(root);

printf("\n");

break;

case 3:

printf("Pre-order traversal: ");

preorderTraversal(root);

printf("\n");

break;

case 4:

printf("Post-order traversal: ");

postorderTraversal(root);

printf("\n");

break;

case 5:

printf("Exiting...\n");

exit(0);

default:

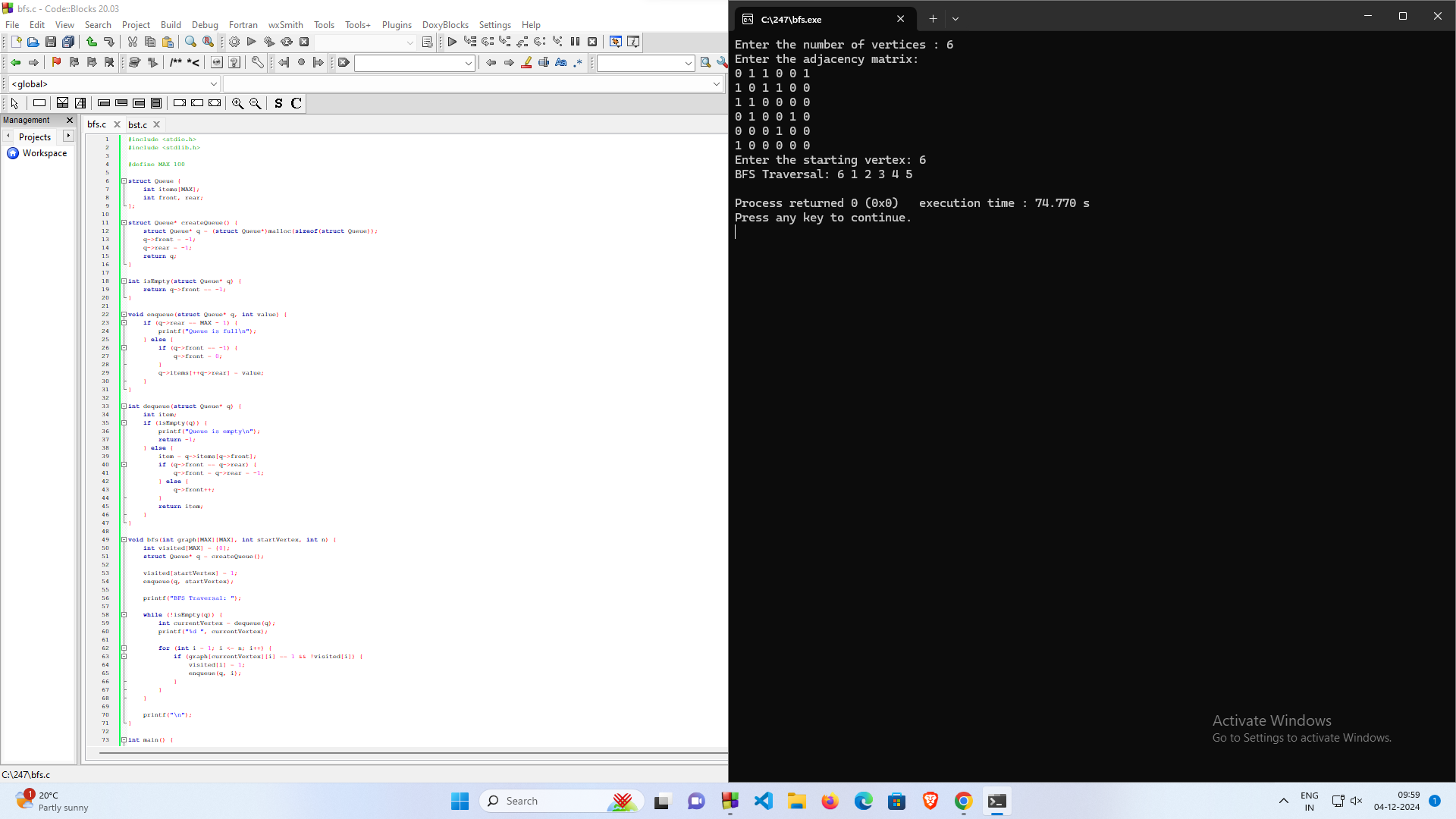
printf("Invalid choice, please try again.\n");

}

}

return 0;

}



**Lab Program 9**

**a) Write a program to traverse a graph using BFS method**.

#include <stdio.h>

#define MAX 5

void bfs(int adj[][MAX], int visited[], int start) {

int queue[MAX], rear = -1, front = -1, i;

for (int k = 0; k < MAX; k++)

visited[k] = 0;

queue[++rear] = start;

++front;

visited[start] = 1;

while (rear >= front) {

start = queue[front++];

printf("%d -> ", start);

for (i = 0; i < MAX; i++) {

if (adj[start][i] && visited[i] == 0) {

queue[++rear] = i;

visited[i] = 1;

}

}

}

}

int main() {

int visited[MAX] = {0};

int adj[MAX][MAX], i, j;

printf("Enter the adjacency matrix of the graph (%d x %d):\n", MAX, MAX);

for (i = 0; i < MAX; i++)

for (j = 0; j < MAX; j++)

scanf("%d", &adj[i][j]);

printf("BFS Traversal starting from node 0:\n");

bfs(adj, visited, 0);

return 0;

}

**OUTPUT:**

**A black screen with white text

Description automatically generated**

**b) Write a program to check whether given graph is connected or not using DFS method.**

#include <stdio.h>

#define MAX\_NODES 100

int adjacencyMatrix[MAX\_NODES][MAX\_NODES];

int visited[MAX\_NODES];

int nodes;

void DFS(int vertex) {

visited[vertex] = 1;

printf("%d ", vertex); // Print visited node

for (int i = 0; i < nodes; i++) {

if (adjacencyMatrix[vertex][i] == 1 && !visited[i]) {

DFS(i);

}

}

}

int isConnected() {

for (int i = 0; i < nodes; i++) {

visited[i] = 0;

}

DFS(0);

for (int i = 0; i < nodes; i++) {

if (!visited[i]) {

return 0;

}

}

return 1;

}

int main() {

printf("Enter the number of nodes: ");

scanf("%d", &nodes);

printf("Enter the adjacency matrix:\n");

for (int i = 0; i < nodes; i++) {

for (int j = 0; j < nodes; j++) {

scanf("%d", &adjacencyMatrix[i][j]);

}

}

if (isConnected()) {

printf("\nThe graph is connected.\n");

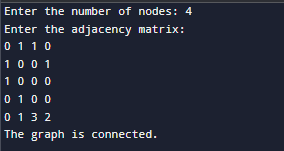
} else {

printf("\nThe graph is not connected.\n");

}

return 0;

}



**Lab Program 10**

**Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function H: K -> L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

int hashTable[MAX\_SIZE];

int m;

void initializeHashTable() {

for (int i = 0; i < m; i++) {

hashTable[i] = -1;

}

}

int hashFunction(int key) {

return key % m;

}

void insertKey(int key) {

int address = hashFunction(key);

int originalAddress = address;

while (hashTable[address] != -1)

{

printf("Collision detected at address %d for key %d. Probing...\n", address, key);

address = (address + 1) % m;

if (address == originalAddress) {

printf("Hash table is full! Cannot insert key %d.\n", key);

return;

}

}

hashTable[address] = key;

printf("Key %d inserted at address %d.\n", key, address);

}

void displayHashTable() {

printf("\nHash Table:\n");

for (int i = 0; i < m; i++) {

if (hashTable[i] == -1)

printf("Address %d: EMPTY\n", i);

else

printf("Address %d: %d\n", i, hashTable[i]);

}

}

int main() {

int n;

int key;

printf("Enter the number of memory locations in the hash table (m): ");

scanf("%d", &m);

if (m > MAX\_SIZE) {

printf("Error: m exceeds the maximum size %d.\n", MAX\_SIZE);

return 1;

}

initializeHashTable();

printf("Enter the number of employee keys (n): ");

scanf("%d", &n);

printf("Enter %d employee keys (4-digit integers):\n", n);

for (int i = 0; i < n; i++) {

scanf("%d", &key);

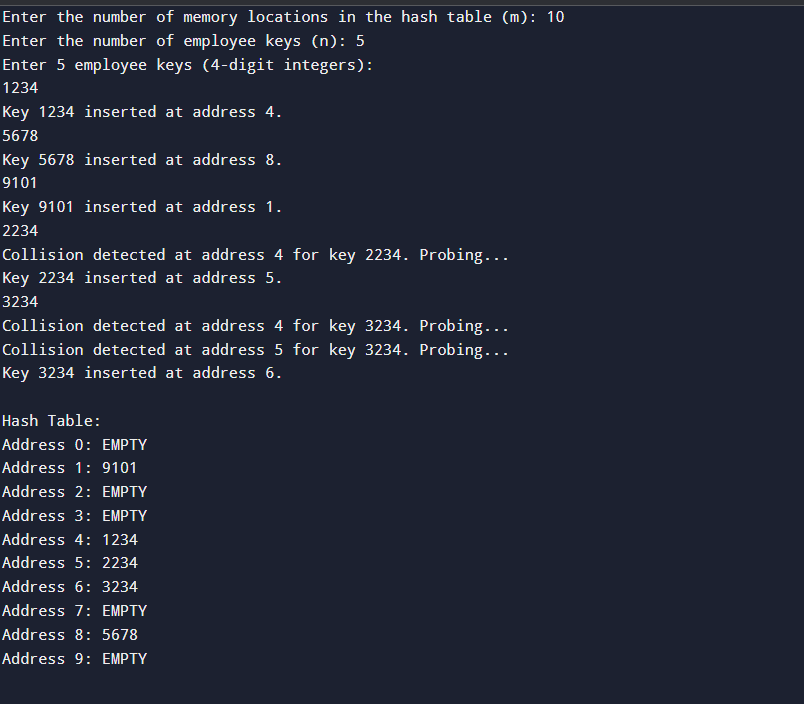
insertKey(key);

}

displayHashTable();

return 0;

}



**Leet Code Programs:**

**1. Move Zeroes**

void moveZeroes(int\* nums, int numsSize) {

int index = 0;

for (int i = 0; i < numsSize; i++) {

if (nums[i] != 0) {

nums[index] = nums[i];

index++;

}

}

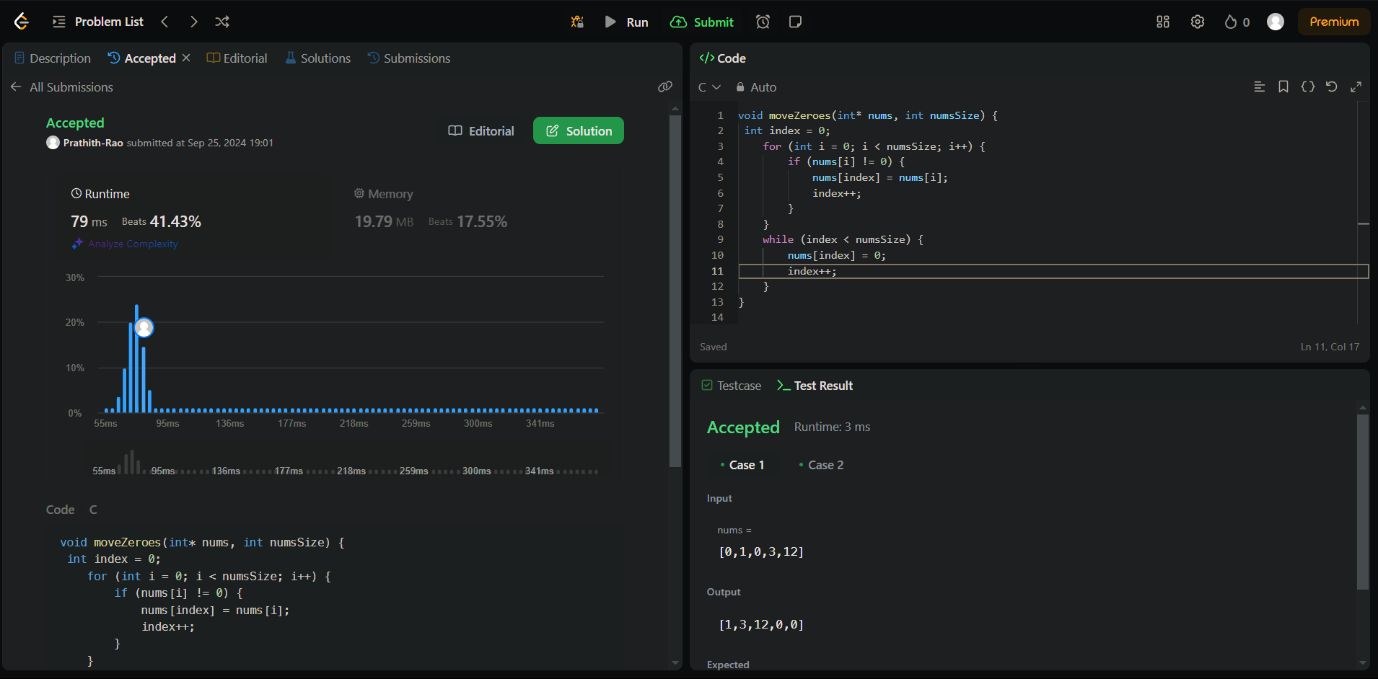
while (index < numsSize) {

nums[index] = 0;

index++;

}

}



**2. Majority Element**

int majorityElement(int\* nums, int numsSize) {

int ele=0;

int c=0;

for(int i=0;i<numsSize;i++)

{

if(c==0)

{

ele=nums[i];

c++;

}

else if(nums[i]==ele)

c++;

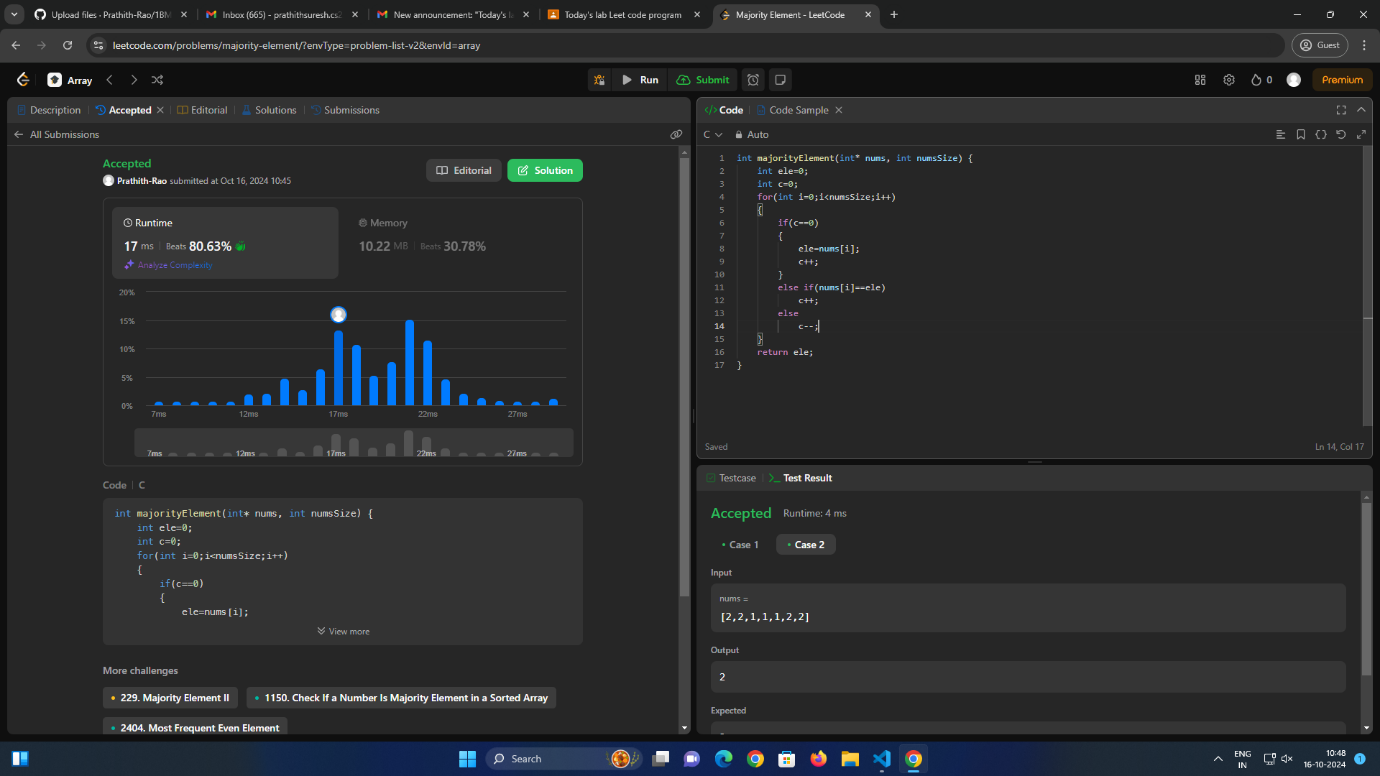
else

c--;

}

return ele;

}



**3. Game of Two Stacks (Hackerrank)**

int twoStacks(int maxSum, int a\_count, int\* a, int b\_count, int\* b) {

int sum\_a = 0, score\_a = 0;

int sum\_b = 0, score\_b = 0;

int max\_score = 0;

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

if (sum\_a + a[i] > maxSum) break;

sum\_a += a[i];

score\_a++;

}

max\_score = score\_a;

for (int j = 0; j < b\_count; j++) {

sum\_b += b[j];

score\_b++;

while (sum\_a + sum\_b > maxSum && score\_a > 0) {

sum\_a -= a[score\_a - 1];

score\_a--;

}

if (sum\_a + sum\_b <= maxSum) {

int current\_score = score\_a + score\_b;

if (current\_score > max\_score) {

max\_score = current\_score;

}

} else {

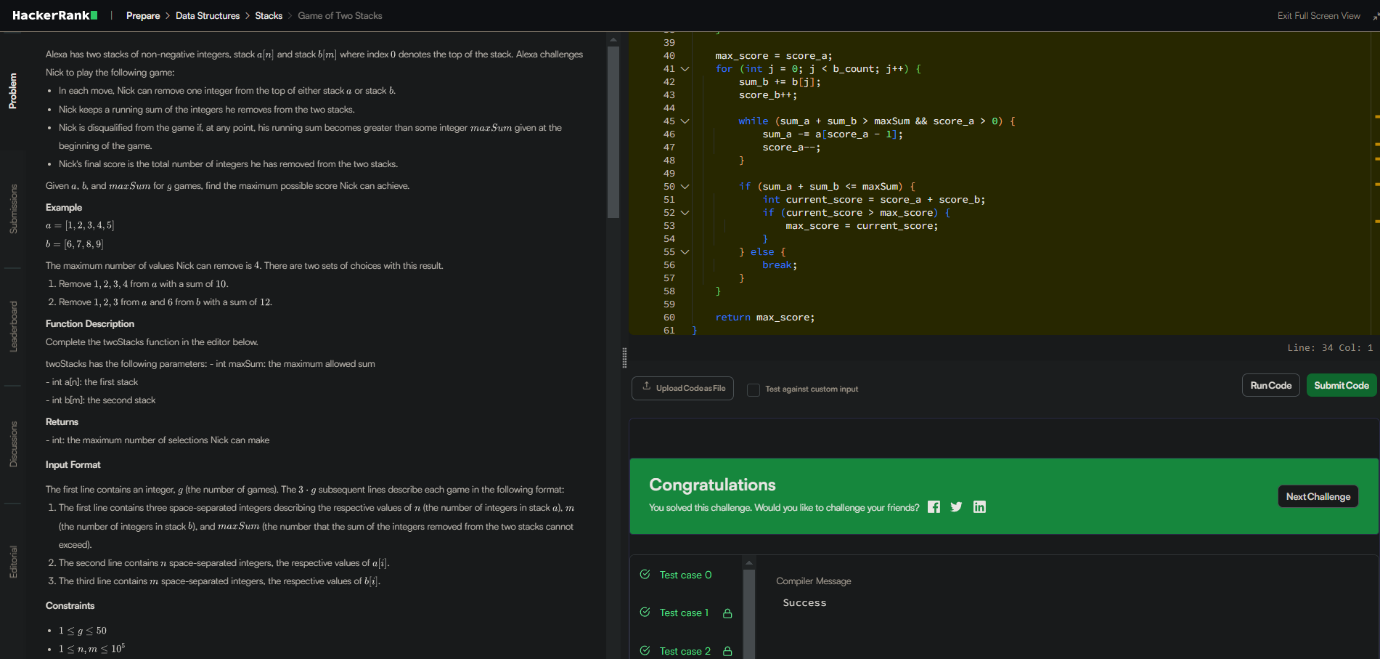
break;

}

}

return max\_score;

}



**4. Palindrome**

bool isPalindrome(struct ListNode\* head) {

struct ListNode\* temp=head;

if(temp->next==NULL) return true;

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

while (fast->next && fast->next->next) {

slow = slow->next;

fast = fast->next->next;

}

temp=slow;

struct ListNode\* prev = NULL;

struct ListNode\* curr = temp;

struct ListNode\* next;

while (curr) {

next = curr->next;

curr->next = prev;

prev = curr;

curr = next;

}

temp=prev;

struct ListNode\* n1=head;

struct ListNode\* n2=temp;

for(;n1&&n2;n1=n1->next,n2=n2->next)

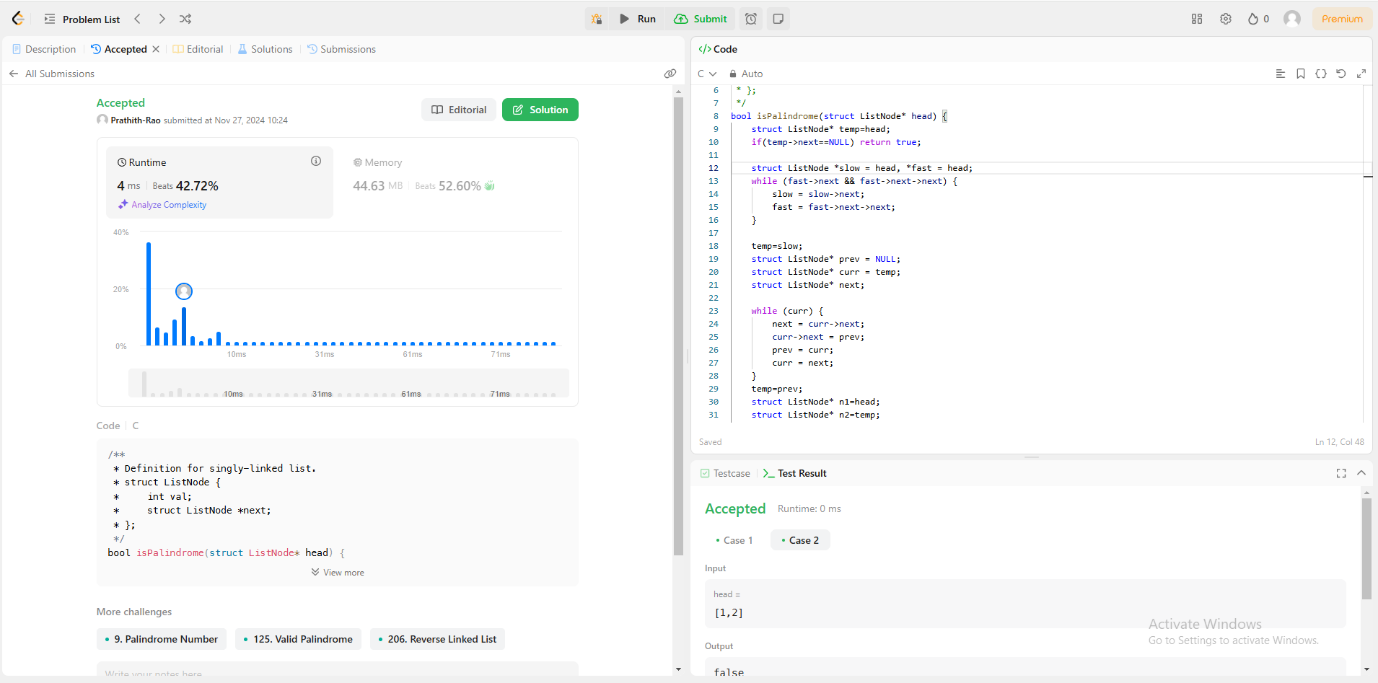
{

if(n1->val!=n2->val)return false;

}

return true;

}



**5. Path Sum**

struct TreeNode {

int val;

struct TreeNode \*left;

struct TreeNode \*right;

};

bool hasPathSum(struct TreeNode\* root, int targetSum) {

if(root==NULL)

return false;

if(root->val==targetSum && root->left==NULL && root->right==NULL)

return true;

else{

return(hasPathSum(root->left,targetSum-root->val) || hasPathSum(root->right,targetSum-root->val));

}

}

