VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

SUMIT KUMAR

CHAUDHARY

(1BM22CS296)

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
Dec 2023- March 2024

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 SUMIT KUMAR CHAUDHARY **(1BM22CS296)**, 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 2023-
24. 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	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

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 {
        (*top)++;
        st[*top] = item;
    }
}

int pop(int st[], int *top) {
    int deletedItem;
    if (*top == -1) {
        printf("Stack underflow\n");
        return -1;
    } else {
        deletedItem = st[*top];
        (*top)--;
        return deletedItem;
    }
}

void display(int st[], int *top) {
    int i;
    if (*top == -1) {
        printf("Stack is empty\n");
    } else {
        printf("Stack elements: ");
        for (i = 0; i <= *top; i++) {
            printf("%d\t", st[i]);
        }
        printf("\n");
    }
}

int main() {
    int st[STACK_SIZE];
    int top = -1, c, item;
```

```
    while (1) {
        printf("\n1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &c);

        switch (c) {
            case 1:
                printf("Enter an item: ");
                scanf("%d", &item);
                push(st, &top, item);
                break;
            case 2:
                item = pop(st, &top);
                if (item != -1)
                    printf("%d item was deleted\n", item);
                break;
            case 3:
                display(st, &top);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice!!!\n");
        }
    }

    return 0;
}
```

Output:

```
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter an item: 34
Enter your choice: 1
Enter an item: 56
Enter your choice: 1
Enter an item: 23
Enter your choice: 3
Stack elements: 34  56  23
Enter your choice: 2
23 item was deleted
Enter your choice: 3
Stack elements: 34  56
Enter your choice: 2
56 item was deleted
Enter your choice: 3
Stack elements: 34
Enter your choice: 4
```

LAB PROGRAM-2

WAP 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>
#include <stdlib.h>
#include <string.h>

#define MAX_SIZE 100

typedef struct {
    char items[MAX_SIZE];
    int top;
} Stack;

void initialize(Stack *s) {
    s->top = -1;
}

int isEmpty(Stack *s) {
    return s->top == -1;
}

int isFull(Stack *s) {
    return s->top == MAX_SIZE - 1;
}

void push(Stack *s, char c) {
    if (!isFull(s)) {
        s->items[++(s->top)] = c;
    }
}

char pop(Stack *s) {
    if (!isEmpty(s)) {
        return s->items[(s->top)--];
    }
    return '\0';
}

char peek(Stack *s) {
    if (!isEmpty(s)) {
        return s->items[s->top];
    }
    return '\0';
}
```

```
int isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/');
}
int precedence(char op) {
    if (op == '+' || op == '-')
        return 1;
    if (op == '*' || op == '/')
        return 2;
    return 0;
}

void infixToPostfix(char *infix, char *postfix) {
    Stack operatorStack;
    initialize(&operatorStack);
    int i, j = 0;
    for (i = 0; infix[i]; i++) {
        char token = infix[i];
        if (token == '(') {
            push(&operatorStack, token);
        } else if (token == ')') {
            while (!isEmpty(&operatorStack) && peek(&operatorStack) != '(') {
                postfix[j++] = pop(&operatorStack);
            }
            pop(&operatorStack); // Discard '('
        } else if (isOperator(token)) {
            while (!isEmpty(&operatorStack) && precedence(peek(&operatorStack)) >=
precedence(token)) {
                postfix[j++] = pop(&operatorStack);
            }
            push(&operatorStack, token);
        } else { // Operand
            postfix[j++] = token;
        }
    }
    while (!isEmpty(&operatorStack)) {
        postfix[j++] = pop(&operatorStack);
    }
    postfix[j] = '\0';
}

int main() {
    char infix[MAX_SIZE], postfix[MAX_SIZE];

    printf("Enter a valid parenthesized infix arithmetic expression: ");
    fgets(infix, sizeof(infix), stdin);
    infix[strcspn(infix, "\n")] = '\0'; // Remove trailing newline

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

    return 0;  }
```

```
Enter a valid parenthesized infix arithmetic expression: ((A
    +B)*C/d)
Postfix expression: AB+C*d/
```

LAB PROGRAM-3a)

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>
#include <stdlib.h>

#define MAX_SIZE 5

int queue[MAX_SIZE];
int front = -1, rear = -1;

void insert(int value) {
   if (rear == MAX_SIZE - 1) {
      printf("Queue overflow\n");
      return;
   }
   if (front == -1)
      front = 0;
   rear++;
   queue[rear] = value;
}

int delete() {
   if (front == -1 || front > rear) {
      printf("Queue underflow\n");
      return -1; // Return some default value to indicate failure
   }
   int deletedItem = queue[front];
   front++;
   return deletedItem;
}

void display() {
   if (front == -1 || front > rear) {
```

```
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    for (int i = front; i <= rear; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        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:
                printf("Deleted item: %d\n", delete());
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 34
Enter your choice: 1
Enter the value to insert: 56
Enter your choice: 1
Enter the value to insert: 12
Enter your choice: 3
Queue elements: 34 56 12
Enter your choice: 2
Deleted item: 34
Enter your choice: 3
Queue elements: 56 12
Enter your choice: 2
Deleted item: 56
Enter your choice: 3
Queue elements: 12
Enter your choice: 2
Deleted item: 12
Enter your choice: 4
```

LAB PROGRAM-3b)

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>
#include <stdlib.h>

#define MAX_SIZE 5

int queue[MAX_SIZE];
int front = -1, rear = -1;

void insert(int value) {
   if ((front == 0 && rear == MAX_SIZE - 1) || (rear == (front - 1) % (MAX_SIZE - 1))) {
      printf("Queue overflow\n");
      return;
   } else if (front == -1) {
      front = rear = 0;
      queue[rear] = value;
   } else if (rear == MAX_SIZE - 1 && front != 0) {
      rear = 0;
      queue[rear] = value;
   } else {
      rear++;
      queue[rear] = value;
   }
}

int delete() {
   if (front == -1) {
      printf("Queue underflow\n");
      return -1; // Return some default value to indicate failure
   }
   int deletedItem = queue[front];
   if (front == rear) {
      front = rear = -1;
   } else if (front == MAX_SIZE - 1) {
      front = 0;
   } else {
      front++;
```

```
    }
    return deletedItem;
}

void display() {
    if (front == -1) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    if (rear >= front) {
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
    } else {
        for (int i = front; i < MAX_SIZE; i++) {
            printf("%d ", queue[i]);
        }
        for (int i = 0; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
    }
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        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:
                printf("Deleted item: %d\n", delete());
                break;
```

```
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

OUTPUT:

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 23
Enter your choice: 1
Enter the value to insert: 45
Enter your choice: 1
Enter the value to insert: 12
Enter your choice: 3
Queue elements: 23 45 12
Enter your choice: 2
Deleted item: 23
Enter your choice: 3
Queue elements: 45 12
Enter your choice: 2
Deleted item: 45
Enter your choice: 3
Queue elements: 12
Enter your choice: 2
Deleted item: 12
Enter your choice: 4
```

LAB PROGRAM-4

WAP to Implement Singly Linked List with following operations:
a) Create a linked list.
b) Insertion of a node at first position, at any position and at end of list. Display the contents of the linked list.

```
#include <stdio.h>
#include <stdlib.h>

// Node structure
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to display the linked list
void display(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}
```

```
// Function to insert a node at the beginning of the list
struct Node* insertAtBeginning(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = head;
    return newNode;
}

// Function to insert a node at any position in the list
void insertAtPosition(struct Node* head, int data, int position) {
    if (position < 1) {
        printf("Invalid position\n");
        return;
    }
    struct Node* newNode = createNode(data);
    struct Node* current = head;
    for (int i = 1; i < position - 1 && current != NULL; i++) {
        current = current->next;
    }
    if (current == NULL) {
        printf("Position out of range\n");
        return;
    }
    newNode->next = current->next;
    current->next = newNode;
}

// Function to insert a node at the end of the list
void insertAtEnd(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        head = newNode;
        return;
    }
    struct Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
}

int main() {
```

```
    struct Node* head = NULL;
    int choice, data, position;

    while (1) {
        printf("\n1. Insert at beginning\n2. Insert at position\n3. Insert at end\n4. Display\n5.
Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the data to insert: ");
                scanf("%d", &data);
                head = insertAtBeginning(head, data);
                break;
            case 2:
                printf("Enter the data to insert: ");
                scanf("%d", &data);
                printf("Enter the position to insert: ");
                scanf("%d", &position);
                insertAtPosition(head, data, position);
                break;
            case 3:
                printf("Enter the data to insert: ");
                scanf("%d", &data);
                insertAtEnd(head, data);
                break;
            case 4:
                display(head);
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

OUTPUT:

```
1. Insert at beginning
2. Insert at position
3. Insert at end
4. Display
5. Exit
Enter your choice: 1
Enter the data to insert: 12
Enter your choice: 4
12 -> NULL
Enter your choice: 1
Enter the data to insert: 8
Enter your choice: 4
8 -> 12 -> NULL
Enter your choice: 3
Enter the data to insert: 45
Enter your choice: 4
8 -> 12 -> 45 -> NULL
Enter your choice: 2
Enter the data to insert: 18
Enter the position to insert: 2
Enter your choice: 4
8 -> 18 -> 12 -> 45 -> NULL
Enter your choice: 5
```

LAB PROGRAM-5

WAP to Implement Singly Linked List with following operations :
a) Create a linked list.
b) Deletion of first element, specified element and last element in the list.
c) Display the contents of the linked list.

```
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node* next;
};

struct node *create_list() {
    struct node *head = NULL;
    struct node *temp, *new_node;
    int num;
    char ch;
    do {
        printf("Enter data: ");
        scanf("%d", &num);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node->data = num;
        new_node->next = NULL;
        if (head == NULL) {
            head = new_node;
            temp = head;
        } else {
            temp->next = new_node;
            temp = new_node;
        }
        printf("Do you want to add another node? (y/n): ");
        scanf(" %c", &ch);
    } while(ch == 'y' || ch == 'Y');
    return head;
}

struct node *delete_beg(struct node *head){
    struct node *temp;
```

```
    temp=head;
    head=head->next;
    free(temp);
    return head;
}

struct node *delete_end(struct node *head){
    struct node *temp, *preptr;
    preptr=temp=head;
    while(temp->next!=NULL){
        preptr=temp;
        temp=temp->next;
    }
    if(temp==head){
        head=NULL;
    }
    else{
        preptr->next=NULL;
    }
    free(temp);
    return head;
}

struct node *deleteafter_pos(struct node *head){
    struct node *preptr,*temp;
    int value;
    printf("Enter the value after which you want to delete: ");
    scanf("%d",&value);
    temp=head;
    preptr=temp;
    while(preptr->data!=value){
        preptr=temp;
        temp=temp->next;
    }
    preptr->next=temp->next;
    free(temp);
    return head;
}

struct node *deletenode(struct node *head){
    struct node *preptr,*temp;
    int value;
```

```
    printf("Enter the value you want to delete: ");
    scanf("%d",&value);
    temp=head;
    if(temp->data==value){
        head=delete_beg(head);
        return head;
    }
    else{
        while(temp->data!=value){
            preptr=temp;
            temp=temp->next;
        }
        preptr->next=temp->next;
        free(temp);
        return head;
    }
}

void display(struct node *head){
    if(head == NULL){
        printf("List is empty.\n");
        return;
    }
    struct node *temp = head;
    while(temp != NULL){
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    struct node *head = NULL;
    int choice;

    printf("Create Linked List:\n");
    head = create_list();

    do {
        printf("\n1. Delete First\n");
        printf("2. Delete Last\n");
        printf("3. Delete After Position\n");
```

```
        printf("4. Delete Node\n");
        printf("5. Display\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                head = delete_beg(head);
                printf("Node deleted from beginning.\n");
                break;
            case 2:
                head = delete_end(head);
                printf("Node deleted from end.\n");
                break;
            case 3:
                head = deleteafter_pos(head);
                printf("Node deleted after given position.\n");
                break;
            case 4:
                head = deletenode(head);
                printf("Node deleted.\n");
                break;
            case 5:
                printf("Linked List: ");
                display(head);
                break;
            case 0:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
                break;
        }
    } while (choice != 0);

    return 0;
}
```

OUTPUT:

```
Create Linked List:
Enter data: 12
Do you want to add another node? (y/n): y
Enter data: 23
Do you want to add another node? (y/n): y
Enter data: 45
Do you want to add another node? (y/n): y
Enter data: 67
Do you want to add another node? (y/n): y
Enter data: 78
Do you want to add another node? (y/n): n

1. Delete First
2. Delete Last
3. Delete After Position
4. Delete Node
5. Display
0. Exit
Enter your choice: 1
Node deleted from beginning.
Enter your choice: 2
Node deleted from end.
Enter your choice: 5
Linked List: 23 45 67
Enter your choice: 3
Enter the value after which you want to delete: 23
Node deleted after given position.
```

LAB PROGRAM-6a):

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;
};

typedef struct Node Node;

Node* createNode(int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   if (newNode == NULL) {
      printf("Memory allocation failed!\n");
      exit(1);
   }
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

void append(Node** head, int data) {
   Node* newNode = createNode(data);
   if (*head == NULL) {
      *head = newNode;
      return;
   }
   Node* current = *head;
   while (current->next != NULL) {
      current = current->next;
   }
   current->next = newNode;
}

void display(Node* head) {
```

```
    while (head != NULL) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

void sort(Node** head) {
    Node *current, *index;
    int temp;

    if (*head == NULL) {
        return;
    }

    for (current = *head; current->next != NULL; current = current->next) {
        for (index = current->next; index != NULL; index = index->next) {
            if (current->data > index->data) {
                temp = current->data;
                current->data = index->data;
                index->data = temp;
            }
        }
    }
}

void reverse(Node** head) {
    Node *prev, *current, *next;
    prev = NULL;
    current = *head;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}

void concatenate(Node** head1, Node* head2) {
    if (*head1 == NULL) {
        *head1 = head2;
```

```
        return;
    }
    Node* current = *head1;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = head2;
}

Node* createLinkedList() {
    Node* head = NULL;
    int n, data;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("Enter the elements: ");
    for (int i = 0; i < n; ++i) {
        scanf("%d", &data);
        append(&head, data);
    }
    return head;
}

int main() {
    Node* list1 = NULL;
    Node* list2 = NULL;

    printf("Creating list 1:\n");
    list1 = createLinkedList();

    printf("Creating list 2:\n");
    list2 = createLinkedList();

    int choice;
    do {
        printf("\nChoose operation:\n");
        printf("1. Sort the linked list\n");
        printf("2. Reverse the linked list\n");
        printf("3. Concatenate two linked lists\n");
        printf("4. Display the linked list\n");
        printf("5. Exit\n");

        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                sort(&list1);
                break;
            case 2:
                reverse(&list1);
                break;
            case 3:
                concatenate(&list1, list2);
                break;
            case 4:
                display(list1);
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    } while (choice != 5);

    return 0;
}
```

OUTPUT:

```
Creating list 1:
Enter the number of elements: 3
Enter the elements: 12
34
45
Creating list 2:
Enter the number of elements: 3
Enter the elements: 56
78
89
1. Sort the linked list
2. Reverse the linked list
3. Concatenate two linked lists
4. Display the linked list
5. Exit

Choose operation:
1

Choose operation:
4
12 -> 34 -> 45 -> NULL

Choose operation:
2

Choose operation:
4
45 -> 34 -> 12 -> NULL
```

```
Choose operation:
3

Choose operation:
4
45 -> 34 -> 12 -> 56 -> 78 -> 89 -> NULL

Choose operation:
5
Exiting...
```

LAB PROGRAM-6b):

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

```
#include <stdio.h>
#include <stdlib.h>

// Node structure
struct Node {
    int data;
    struct Node* next;
};

typedef struct Node Node;

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    newNode->data = data;
```

```
    newNode->next = NULL;
    return newNode;
}

// Stack operations: push and pop
void push(Node** top, int data) {
    Node* newNode = createNode(data);
    newNode->next = *top;
    *top = newNode;
}

int pop(Node** top) {
    if (*top == NULL) {
        printf("Stack underflow!\n");
        exit(1);
    }
    Node* temp = *top;
    int poppedData = temp->data;
    *top = (*top)->next;
    free(temp);
    return poppedData;
}

// Queue operations: enqueue and dequeue
void enqueue(Node** rear, int data) {
    Node* newNode = createNode(data);
    if (*rear == NULL) {
        *rear = newNode;
    } else {
        (*rear)->next = newNode;
        *rear = newNode;
    }
}

int dequeue(Node** front) {
    if (*front == NULL) {
        printf("Queue underflow!\n");
        exit(1);
    }
    Node* temp = *front;
    int dequeuedData = temp->data;
    *front = (*front)->next;
```

```c
    free(temp);
    return dequeuedData;
}

// Display the elements of the stack or queue
void display(Node* head) {
    if (head == NULL) {
        printf("Empty\n");
        return;
    }
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

int main() {
    Node* stackTop = NULL;  // Initialize stack top
    Node* queueFront = NULL; // Initialize queue front
    Node* queueRear = NULL; // Initialize queue rear

    int choice, data;
    do {
        printf("\nChoose operation:\n");
        printf("1. Push (Stack)\n");
        printf("2. Pop (Stack)\n");
        printf("3. Enqueue (Queue)\n");
        printf("4. Dequeue (Queue)\n");
        printf("5. Display (Stack)\n");
        printf("6. Display (Queue)\n");
        printf("7. Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to push: ");
                scanf("%d", &data);
                push(&stackTop, data);
                break;
            case 2:
                if (stackTop == NULL) {
```

```
            printf("Stack is empty.\n");
        } else {
            printf("Popped element: %d\n", pop(&stackTop));
        }
        break;
    case 3:
        printf("Enter data to enqueue: ");
        scanf("%d", &data);
        enqueue(&queueRear, data);
        if (queueFront == NULL) {
            queueFront = queueRear;
        }
        break;
    case 4:
        if (queueFront == NULL) {
            printf("Queue is empty.\n");
        } else {
            printf("Dequeued element: %d\n", dequeue(&queueFront));
        }
        break;
    case 5:
        printf("Stack elements: ");
        display(stackTop);
        break;
    case 6:
        printf("Queue elements: ");
        display(queueFront);
        break;
    case 7:
        printf("Exiting...\n");
        break;
    default:
        printf("Invalid choice. Please enter a valid option.\n");
    }
  } while (choice != 7);

  return 0;
}
```

OUTPUT:

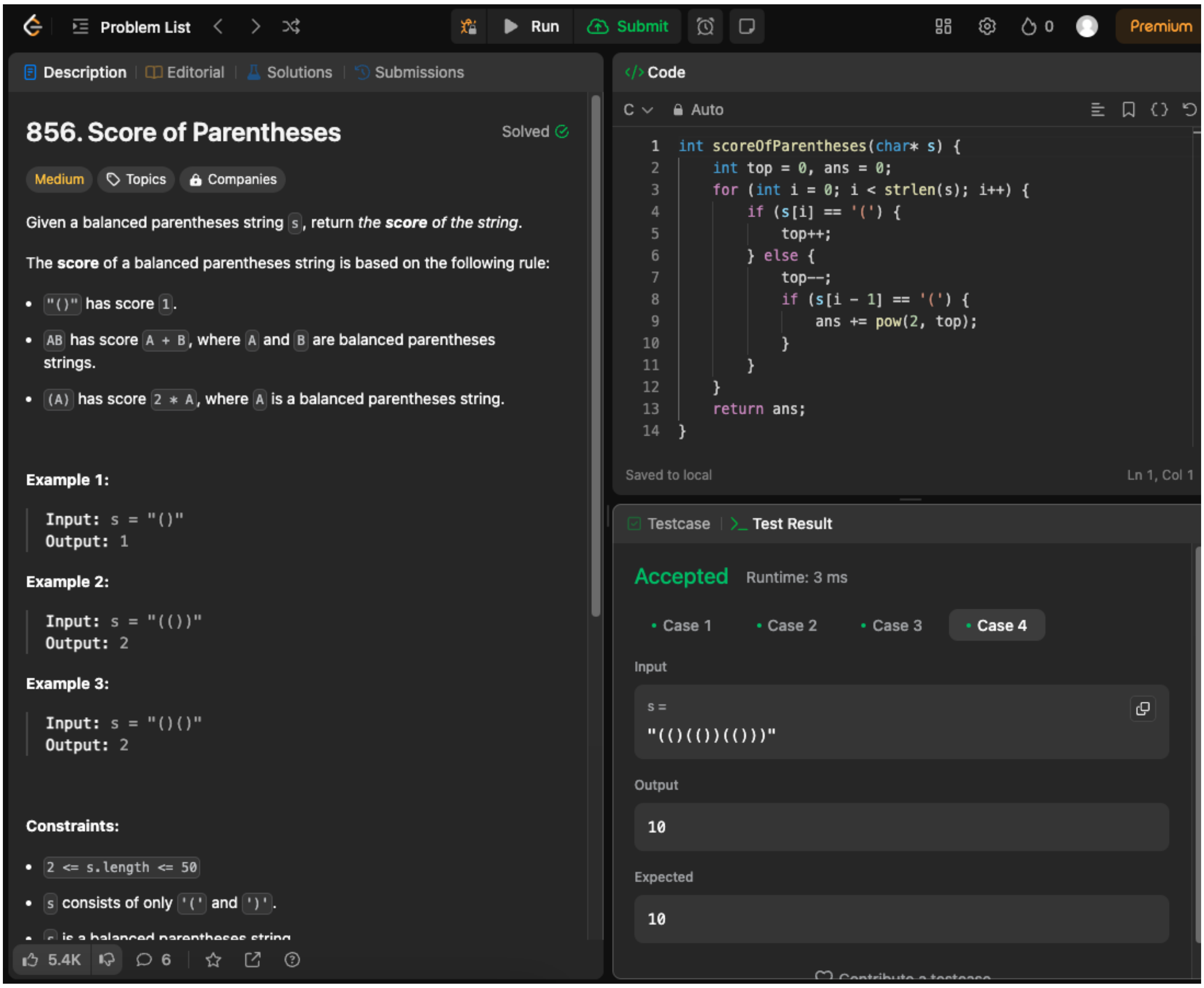
```
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display (Stack)
6. Display (Queue)
7. Exit

Choose operation:
1
Enter data to push: 12

Choose operation:
1
Enter data to push: 23

Choose operation:
5
Stack elements: 23 12

Choose operation:
2
Popped element: 23

Choose operation:
5
Stack elements: 12

Choose operation:
```

```
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display (Stack)
6. Display (Queue)
7. Exit

Choose operation:
3
Enter data to enqueue: 23

Choose operation:
3
Enter data to enqueue: 45

Choose operation:
6
Queue elements: 23 45

Choose operation:
4
Dequeued element: 23

Choose operation:
6
Queue elements: 45

Choose operation:
7
Exiting...
```

LEETCODE PROBLEM-1:

Score of Parentheses:

```
int scoreOfParentheses(char* s) {
    int count=0;
    int ans=0;
    int size=sizeof(s)/sizeof(s[0]);
    for(int i=0;s[i]!='\0';i++)
    {
        if(s[i]=='(')
        {
            count++;
        }
        else{
            count--;
            if(s[i-1]=='(')
            {
                ans+=pow(2,count);
            }
        }

    }
     return ans;
}
```



<u>LAB PROGRAM-7:</u>

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

d) Display the contents of the list

```
#include <stdio.h>
#include <stdlib.h>

// Node structure
struct Node {
    int data;
    struct Node* prev;
```

```
    struct Node* next;
};

typedef struct Node Node;


Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a new node to the left of a specified node
void insertLeft(Node** head, int value, int newValue) {
    Node* newNode = createNode(newValue);
    Node* current = *head;
    while (current != NULL && current->data != value) {
        current = current->next;
    }
    if (current == NULL) {
        printf("Node with value %d not found. New node not inserted.\n", value);
        free(newNode);
        return;
    }
    newNode->next = current;
    newNode->prev = current->prev;
    if (current->prev != NULL) {
        current->prev->next = newNode;
    }
    current->prev = newNode;
    if (current == *head) {
        *head = newNode;
    }
}

// Function to delete a node based on a specific value
```

```
void deleteNode(Node** head, int value) {
    Node* current = *head;
    while (current != NULL && current->data != value) {
        current = current->next;
    }
    if (current == NULL) {
        printf("Node with value %d not found.\n", value);
        return;
    }
    if (current->prev != NULL) {
        current->prev->next = current->next;
    }
    if (current->next != NULL) {
        current->next->prev = current->prev;
    }
    if (current == *head) {
        *head = current->next;
    }
    free(current);
}

// Function to display the contents of the list
void display(Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    while (head != NULL) {
        printf("%d <-> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

int main() {
    Node* head = createNode(1);
    head->next = createNode(2);
    head->next->prev = head;
    head->next->next = createNode(3);
    head->next->next->prev = head->next;

    int choice, value, newValue;
```

```
    do {
        printf("\nChoose operation:\n");
        printf("1. Insert a new node to the left of a specified node\n");
        printf("2. Delete a node based on a specific value\n");
        printf("3. Display the contents of the list\n");
        printf("4. Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value of the existing node: ");
                scanf("%d", &value);
                printf("Enter the value of the new node: ");
                scanf("%d", &newValue);
                insertLeft(&head, value, newValue);
                break;
            case 2:
                printf("Enter the value of the node to delete: ");
                scanf("%d", &value);
                deleteNode(&head, value);
                break;
            case 3:
                printf("List contents:\n");
                display(head);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    } while (choice != 4);

    return 0;
}
```

OUTPUT:

```
enter operation
1.create
2.display
3.insert at left
4.delete at position
5.-1 to end
enter operation
1
Enter the no. of elements:
2
Enter the 1 element :
23
Enter the 2 element :
45
enter operation
2
23
45
enter operation
3
enter the node
2
enter data
56
```

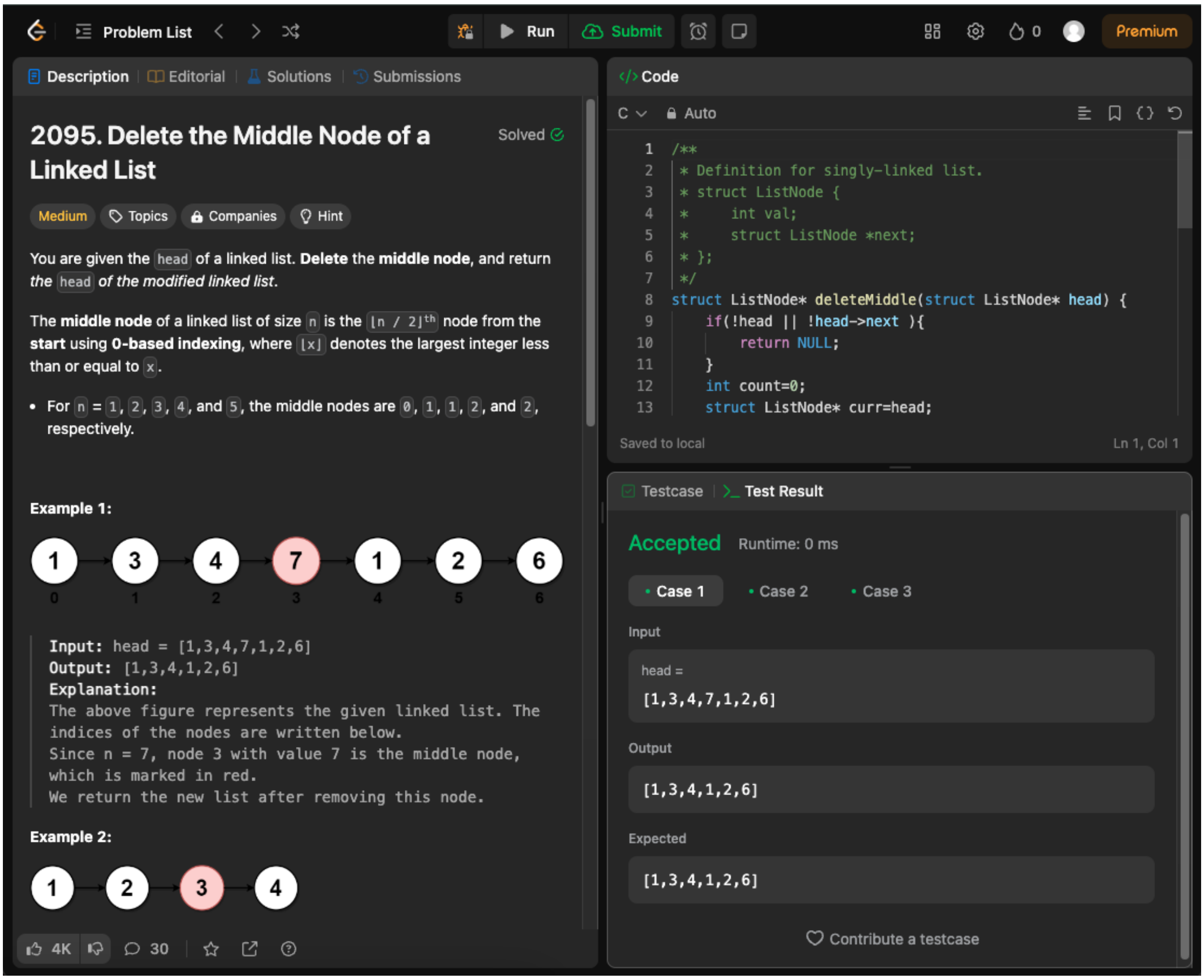
```
enter operation
2
23
56
45
enter operation
4
enter position
2
enter operation
2
23
45
enter operation
```

LEETCODE PROBLEM-2:

Delete the Middle Node of the Linked List

```
struct node {
 int val;
  struct node *next;
};

struct node* deleteMiddle(struct node* head) {
   if(head==NULL) return NULL;
   struct node* newnode=(struct node*)malloc(sizeof(struct node));
   newnode->val=0;
   newnode->next=head;
   struct node* preptr=newnode;
   struct node* ptr=head;
   while(ptr!=NULL && ptr->next!=NULL){
      preptr=preptr->next;
      ptr=ptr->next->next;
```

```
    }
    struct node *temp=preptr->next;
    preptr->next=preptr->next->next;
    free(temp);
    struct node* newHead=newnode->next;
    free(newnode);
    return newHead;
    return head;
}
```



LAB PROGRAM-8:

Write a program

a) To construct a binary Search tree.

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

c) To display the elements in the tree.

```
#include <stdio.h>
#include <stdlib.h>

// Structure for a node of the binary search tree
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Function to insert a new node into the binary search tree
struct Node* insert(struct Node* root, int value) {
    if (root == NULL) {
        return createNode(value);
    }

    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }

    return root;
}

// Function to traverse the binary search tree using inorder traversal
void inorderTraversal(struct Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

// Function to traverse the binary search tree using postorder traversal
void postorderTraversal(struct Node* root) {
    if (root != NULL) {
```

```
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}

// Function to traverse the binary search tree using preorder traversal
void preorderTraversal(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

// Function to display the elements in the binary search tree
void display(struct Node* root) {
    printf("Elements in the tree: ");
    inorderTraversal(root);
    printf("\n");
}

int main() {
    struct Node* root = NULL;
    int choice, value;

    do {
        printf("\nBinary Search Tree Operations:\n");
        printf("1. Insert\n");
        printf("2. Inorder Traversal\n");
        printf("3. Postorder Traversal\n");
        printf("4. Preorder Traversal\n");
        printf("5. Display\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
            case 2:
                printf("Inorder Traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;
            case 3:
                printf("Postorder Traversal: ");
                postorderTraversal(root);
                printf("\n");
                break;
            case 4:
                printf("Preorder Traversal: ");
                preorderTraversal(root);
```

```
            printf("\n");
            break;
        case 5:
            display(root);
            break;
        case 6:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice! Please enter a valid option.\n");
      }
   } while (choice != 6);

   return 0;
}
```

OUTPUT:

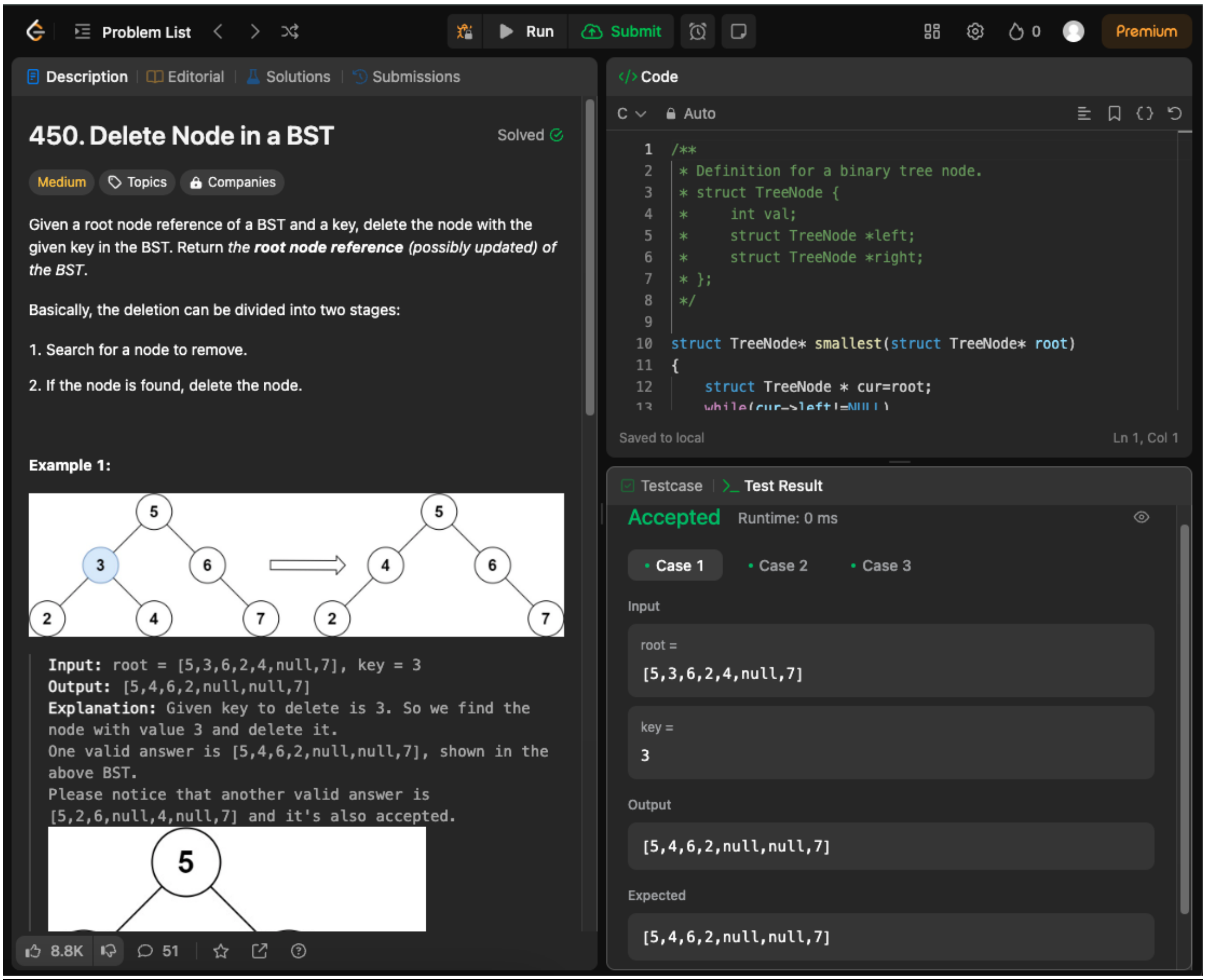
```
1.Create
2.InOrder display
3.PostOrder Display
4.PreOrder Display
5.EXIT
ENTER YOUR CHOICE : 1
ENTER THE DATA : 21
DO YOU WANT TO ENTER MORE (Y/N) : y
ENTER THE DATA : 45
DO YOU WANT TO ENTER MORE (Y/N) : y
ENTER THE DATA : 12
DO YOU WANT TO ENTER MORE (Y/N) : n
ENTER YOUR CHOICE : 2
ELEMENTS OF TREE ARE (INORDER) : 12 21 45
ENTER YOUR CHOICE : 3
ELEMENTS OF TREE ARE (PREORDER) : 21 12 45
ENTER YOUR CHOICE : 4
ELEMENTS OF TREE ARE (POSTORDER) : 12 45 21
ENTER YOUR CHOICE : 5
```

LEETCODE PROBLEM-4:

Delete a node in BST

```
struct TreeNode* smallest(struct TreeNode* root)
{
    struct TreeNode * cur=root;
    while(cur->left!=NULL)
    {
        cur=cur->left;
    }
    return cur;
}

struct TreeNode* deleteNode(struct TreeNode* root, int key)
{

            // base case
    if(root==NULL)
    {
        return root;
    }
    if(key < root->val)
    {
        root->left = deleteNode(root->left,key);
    }
    else if(key > root->val)
    {
        root->right=deleteNode(root->right,key);
    }
 else
    {

        if (root->left == NULL)
        {
            struct TreeNode *temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL)
        {
            struct TreeNode *temp = root->left;
```

```
            free(root);
            return temp;
        }
        struct TreeNode *temp = smallest(root->right);
        root->val = temp->val;
        root->right = deleteNode(root->right, root->val);
    }

    return root;
}
```

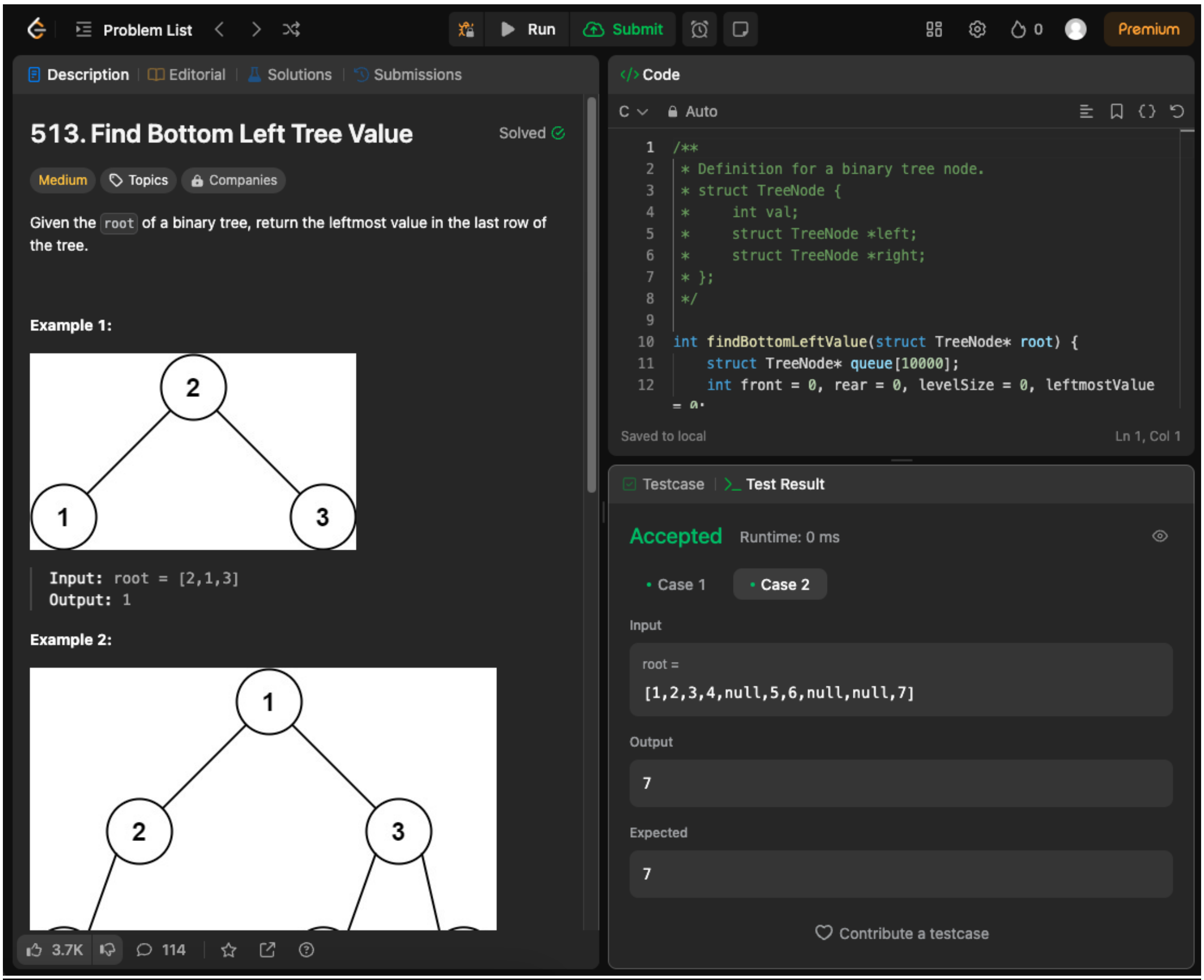


LEETCODE PROBLEM-5:

Find bottom left tree value

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */

#define MAX_SIZE 1000

typedef struct {
    struct TreeNode* data[MAX_SIZE];
    int front;
    int rear;
} RingBuffer;

void append(RingBuffer* buffer, struct TreeNode* node) {
    if (buffer->front == -1) {
        buffer->front = 0;
        buffer->rear = 0;
    }
    else if (buffer->rear == MAX_SIZE - 1)  buffer->rear = 0;
    else buffer->rear++;

    buffer->data[buffer->rear] = node;
}

struct TreeNode* popleft(RingBuffer* buffer) {
    struct TreeNode* node = buffer->data[buffer->front];
    if (buffer->front == buffer->rear) {
        buffer->front = -1;
        buffer->rear = -1;
    }
    else if (buffer->front == MAX_SIZE - 1)  buffer->front = 0;
    else     buffer->front++;

    return node;
}
```

```c
int findBottomLeftValue(struct TreeNode* root) {
    if (root == NULL)  return -1;
    RingBuffer buffer = {
        .front = -1,
        .rear = -1
    };
    append(&buffer, root);

    struct TreeNode* node;
    while( buffer.front != -1 ){
        node = popleft(&buffer);
        if (node->right) append(&buffer, node->right);
        if (node->left)  append(&buffer, node->left);
    }

    return node->val;
}
```



LAB PROGRAM-9a):

Write a program to traverse a graph using BFS method.

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_NODES 100

// Define a structure for a node in the graph
struct Node {
    int data;
    struct Node* next;
};

// Define a structure for the graph
struct Graph {
    int numNodes;
    struct Node* adjLists[MAX_NODES];
    int visited[MAX_NODES];
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to create a graph with n nodes
struct Graph* createGraph(int n) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numNodes = n;
    for (int i = 0; i < n; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }
    return graph;
}

// Function to add an edge to the graph
void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to dest
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;
```

```
    // Add edge from dest to src
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// Function to perform Breadth First Search
void BFS(struct Graph* graph, int startNode) {
    // Create a queue for BFS
    int queue[MAX_NODES];
    int front = 0, rear = 0;

    // Mark the current node as visited and enqueue it
    graph->visited[startNode] = 1;
    queue[rear++] = startNode;

    while (front < rear) {
        // Dequeue a vertex from queue and print it
        int current = queue[front++];
        printf("%d ", current);

        // Get all adjacent vertices of the dequeued vertex current
        // If an adjacent has not been visited, then mark it visited and enqueue it
        struct Node* temp = graph->adjLists[current];
        while (temp) {
            int adjNode = temp->data;
            if (!graph->visited[adjNode]) {
                graph->visited[adjNode] = 1;
                queue[rear++] = adjNode;
            }
            temp = temp->next;
        }
    }
}

int main() {
    // Get the number of nodes from the user
    int numNodes;
    printf("Enter the number of nodes: ");
    scanf("%d", &numNodes);

    // Create a graph with the specified number of nodes
    struct Graph* graph = createGraph(numNodes);

    // Get the number of edges from the user
    int numEdges;
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    // Add edges
    for (int i = 0; i < numEdges; i++) {
```

```c
        int src, dest;
        printf("Enter edge %d (source destination): ", i + 1);
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }

    // Print BFS traversal
    int startNode;
    printf("Enter the starting node for BFS traversal: ");
    scanf("%d", &startNode);
    printf("BFS traversal starting from node %d: ", startNode);
    BFS(graph, startNode);

    return 0;
}
```

OUTPUT:

```
Enter the number of nodes: 5
Enter the number of edges: 4
Enter edge 1 (source destination): 0
1
Enter edge 2 (source destination): 0
2
Enter edge 3 (source destination): 2
3
Enter edge 4 (source destination): 1
4
Enter the starting node for BFS traversal: 0
BFS traversal starting from node 0: 0 2 1 3 4
```

LAB PROGRAM-9b):

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

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_NODES 100

// Define a structure for a node in the graph
struct Node {
    int data;
    struct Node* next;
};

// Define a structure for the graph
struct Graph {
    int numNodes;
    struct Node* adjLists[MAX_NODES];
    int visited[MAX_NODES];
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to create a graph with n nodes
struct Graph* createGraph(int n) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numNodes = n;
    for (int i = 0; i < n; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }
    return graph;
}

// Function to add an edge to the graph
void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to dest
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Add edge from dest to src
    newNode = createNode(src);
```

```
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// Function to perform Depth First Search
void DFS(struct Graph* graph, int startNode) {
    // Mark the current node as visited
    graph->visited[startNode] = 1;
    printf("%d ", startNode);

    // Get all adjacent vertices of the current node
    struct Node* temp = graph->adjLists[startNode];
    while (temp) {
        int adjNode = temp->data;
        if (!graph->visited[adjNode]) {
            DFS(graph, adjNode);
        }
        temp = temp->next;
    }
}

int main() {
    // Get the number of nodes from the user
    int numNodes;
    printf("Enter the number of nodes: ");
    scanf("%d", &numNodes);

    // Create a graph with the specified number of nodes
    struct Graph* graph = createGraph(numNodes);

    // Get the number of edges from the user
    int numEdges;
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    // Add edges
    for (int i = 0; i < numEdges; i++) {
        int src, dest;
        printf("Enter edge %d (source destination): ", i + 1);
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }

    // Print DFS traversal
    int startNode;
    printf("Enter the starting node for DFS traversal: ");
    scanf("%d", &startNode);
    printf("DFS traversal starting from node %d: ", startNode);
    DFS(graph, startNode);

    return 0;
```

```
}
```

OUTPUT:

```
Enter the number of nodes: 5
Enter the number of edges: 4
Enter edge 1 (source destination): 0
1
Enter edge 2 (source destination): 0
2
Enter edge 3 (source destination): 1
3
Enter edge 4 (source destination): 1
4
Enter the starting node for DFS traversal: 0
DFS traversal starting from node 0: 0 2 1 4 3
```

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 SIZE 10 // Size of the hash table

// Employee structure
struct Employee {
```

```
    int key; // 4-digit key
    int data; // Data associated with the employee
};

typedef struct Employee Employee;

// Hash table structure
struct HashTable {
    Employee* table[SIZE];
};

typedef struct HashTable HashTable;

// Function to initialize the hash table
void initializeHashTable(HashTable* ht) {
    for (int i = 0; i < SIZE; i++) {
        ht->table[i] = NULL;
    }
}

// Hash function: H(K) = K mod m (remainder method)
int hashFunction(int key) {
    return key % SIZE;
}

// Function to insert a record into the hash table
void insert(HashTable* ht, int key, int data) {
    int index = hashFunction(key);
    while (ht->table[index] != NULL) {
        index = (index + 1) % SIZE; // Linear probing
    }
    Employee* newEmployee = (Employee*)malloc(sizeof(Employee));
    newEmployee->key = key;
    newEmployee->data = data;
    ht->table[index] = newEmployee;
}

// Function to display the contents of the hash table
void displayHashTable(HashTable* ht) {
    printf("Hash Table Contents:\n");
    printf("Index\tKey\tData\n");
    for (int i = 0; i < SIZE; i++) {
```

```
        if (ht->table[i] != NULL) {
            printf("%d\t%d\t%d\n", i, ht->table[i]->key, ht->table[i]->data);
        } else {
            printf("%d\tEmpty\n", i);
        }
    }
}

int main() {
    HashTable ht;
    initializeHashTable(&ht);

    int key, data;
    char choice;
    do {
        printf("Enter employee key (4-digit): ");
        scanf("%d", &key);
        printf("Enter employee data: ");
        scanf("%d", &data);
        insert(&ht, key, data);

        printf("Do you want to enter another employee record? (y/n): ");
        scanf(" %c", &choice);
    } while (choice == 'y' || choice == 'Y');

    // Displaying the contents of the hash table
    displayHashTable(&ht);

    return 0;
}
```

OUTPUT:

```
Enter employee key (4-digit): 1234
Enter employee data: 1001
Do you want to enter another employee record? (y/n): y
Enter employee key (4-digit): 1235
Enter employee data: 1002
Do you want to enter another employee record? (y/n): n
Hash Table Contents:
Index   Key Data
0   Empty
1   Empty
2   Empty
3   Empty
4   1234    1001
5   1235    1002
6   Empty
7   Empty
8   Empty
9   Empty
```