Sri Eshwar College of Engineering

Kinathukadavu. Coimbatore-641202

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

U23CS451-DATA STRUCTURES

LABORATORY

BONAFIDE CERTIFICATE

Certified that this is the Bonafide record of work done by,

Name: Mr. / Ms. .....

Register No: ….....

of I year B.E/B.Tech .....

in the U23CS451–DATA STRUCUTERS LABORATORY during the II Semester of the Academic year 2024 – 2025(Even semester)

Signature of Faculty In-Charge **Head of the Department**

Submitted for the practical examinations held on .....

INTERNAL EXAMINER **EXTERNAL EXAMINER**

LIST OF EXPERIMENTS

S. No.	Date	Name of the Experiments	Page No.	Marks (75)	Faculty Signature
1.		Multimedia Library Management System - Linked List Operations			
2a.		Implementation of Reverse Operation in Linked List			
2b.		Implementation of Loop Detection in Circularly Linked List			
2c.		Implementation of Segregation of Even and Odd Nodes in Linked List			
3a.		Implementation of Stack using Array			
3b.		Implementation of Stack using Linked List			
3c.		Implementation of Queue using Array			
3d.		Implementation of Queue using Linked List			
4a.		Implementation of Postfix Evaluation			
4b.		Implementation of Balancing Parentheses			
5.		Implementation of Binary Tree			
6.		Implementation of Binary Search Tree			
7a.		Segment Tree and Range of Minimum Query			
7b.		Maximum Depth of Binary Tree			
8.		Graph Traversal			
9.		Implementation of Topological Sort			
10.		Food Delivery Route Optimization using Dijkstra's Algorithm - Graph Operations			
11a.		Optimized Network Infrastructure: Minimum Spanning Tree (MST) using Prims's Algorithm			
11b.		Optimized Network Infrastructure: Minimum Spanning Tree (MST) using Kruskal's Algorithm			
12.		Student Grade Management System			
13a.		Product Inventory Management System - Search Operations			
13b.		Product Inventory Management System - Search Operations			
14		Demonstration of Applications of Hashing			

Marks in Words:

Average :

SIGNATURE OF THE FACULTY

Exp No. 1	MULTIMEDIA LIBRARY MANAGEMENT SYSTEM - LINKED LIST OPERATIONS
Date:	

AIM:

To organize and manage multimedia content - image viewer using linked list operations suitable for each content type.

ALGORITHM:

1. Define structures for image nodes.
2. Create linked lists for each content type.
3. Create separate head pointers for each content type
4. Implement insertion and deletion operations based on content type.
5. Ensure null checks and type-based field access.
6. Traverse and display multimedia data.
7. Free allocated memory.

PROGRAM:

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

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

//Node creation for storing a image file.
struct node* createNode(struct node **head, int val){
  struct node *newNode = (struct node*)malloc(sizeof(struct node));
  newNode->Image_No=val;
  newNode->next=NULL;
  return newNode;
}

// Insertion at the End of the Linked List
void append(struct node **head, int val){
  struct node *newNode = createNode(head, val);
  if(*head==NULL)
    *head = newNode;
  else{
    struct node *temp = * head;
    while(temp->next!=NULL){
      temp = temp->next;
    }
    temp->next = newNode;
  }
}
```

```
//Insertion at the Begining
void insertAtBegin(struct node **head, int val){
  struct node *newNode = createNode(head, val);
  newNode->next = *head;
  *head = newNode;
}

//Insertion at the Middle
void insertAtMid(struct node** head, int pos, int val){
  struct node *newNode = createNode(head, val);
  int count =1;
  struct node *temp = *head;
  while(count<pos-1){
    temp = temp->next;
    count++;
  }
  newNode->next = temp->next;
  temp->next = newNode;
}

//Display the image numbers in the list
void display(struct node *head){
  struct node *temp = head;
  while(temp!=NULL){
    printf("Image No.%d --> ", temp->Image_No);
    temp = temp->next;
  }
}

//Deletion operation
void delAtBegin(struct node **head){
  struct node *temp = *head;
  *head = temp->next;
  free(temp);
}

int main(void) {
  struct node *head=NULL;
  int num, val;
  printf("\n Enter the no. of images to be inserted\n");
  scanf("%d",&num);
  for(int i=0;i<num;i++){
      scanf("%d",&val);
      append(&head, val);
  }
  printf("\nList of image numbers in the list\n");
  display(head);

  insertAtBegin(&head,5);
  printf("\n\nAfter insertion at beginning\n");
  display(head);
```

```
    printf("\n\nAfter insertion at some position\n");
    int pos;
    printf("Enter the position to insert: ");
    scanf("%d", &pos);

    insertAtMid(&head, pos, 25);
    display(head);

    delAtBegin(&head);
    printf("\n\nAfter deletion at beginning\n");
    display(head);

    return 0;
}
```

OUTPUT:

```
 Enter the no. of images to be inserted
4
10 20 30 40

List of image numbers in the list
Image No.10 --> Image No.20 --> Image No.30 --> Image No.40 -->

After insertion at beginning
Image No.5 --> Image No.10 --> Image No.20 --> Image No.30 --> Image No.40 -->

After insertion at some position
Enter the position to insert: 4
Image No.5 --> Image No.10 --> Image No.20 --> Image No.25 --> Image No.30 --> Image No.40 -->

After deletion at beginning
Image No.10 --> Image No.20 --> Image No.25 --> Image No.30 --> Image No.40 -->

...Program finished with exit code 0
Press ENTER to exit console.
```

TEST CASES:

1. **Input:** 3 images (10, 20, 30)
    - **Expected Output:** 10 -> 20 -> 30 -> NULL
2. **Input:** Insert 5 at the beginning
    - **Expected Output:** 5 -> 10 -> 20 -> 30 -> NULL
3. **Input:** Insert 25 at position 2
    - **Expected Output:** 5 -> 25 -> 10 -> 20 -> 30 -> NULL
4. **Input:** Delete the first node
    - **Expected Output:** 25 -> 10 -> 20 -> 30 -> NULL

RESULT:

Thus the C program to organize and manage multimedia content - image viewer using linked list was completed successfully.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 2 a	IMPLEMENTATION OF REVERSE OPERATION IN LINKED LIST
Date:	

AIM:

To implement a function to reverse a singly linked list using iterative approach.

ALGORITHM:

- Initialize three pointers: prev (NULL), curr (head), and next (NULL).
- Traverse the linked list.
- Store the next node: next = curr->next
- Reverse the link: curr->next = prev
- Move pointers forward: prev = curr and curr = next
- Set head to prev (new head of reversed list).

PROGRAM:

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

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

struct node* createNode(int val) {
   struct node* newNode = (struct node*)malloc(sizeof(struct node));
   newNode->data = val;
   newNode->next = NULL;
   return newNode;
}

void append(struct node** head, int val) {
   struct node* newNode = createNode(val);
   if (*head == NULL)
      *head = newNode;
   else {
      struct node* temp = *head;
      while (temp->next != NULL)
         temp = temp->next;
      temp->next = newNode;
   }
}

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

```
}

void reverseLinkedList(struct node** head) {
    struct node *prev = NULL, *curr = *head, *next = NULL;
    while (curr != NULL) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    *head = prev;
}

int main() {
    struct node* head = NULL;
    append(&head, 10);
    append(&head, 20);
    append(&head, 30);
    append(&head, 40);

    printf("Original List:\n");
    display(head);

    reverseLinkedList(&head);

    printf("\nReversed List:\n");
    display(head);

    return 0;
}
```

OUTPUT:

```
Original List:
10 -> 20 -> 30 -> 40 -> NULL

Reversed List:
40 -> 30 -> 20 -> 10 -> NULL


...Program finished with exit code 0
Press ENTER to exit console.
```

TEST CASE:

1. **Reverse a Multi-Node List**
   Input: 10 -> 20 -> 30 -> 40 -> 50 -> NULL
   Operation: Reverse the linked list
   Expected Output: 50 -> 40 -> 30 -> 20 -> 10 -> NULL

RESULT:

Thus the C program to implement a function to reverse a singly linked list using iterative approach.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 2 b	IMPLEMENTATION OF LOOP DETECTION IN CIRCULARLY LINKED LIST
Date:	

AIM:

To detect a loop in a circularly linked list using Floyd's Cycle-Finding Algorithm (Tortoise and Hare approach).

ALGORITHM:

1. Initialize two pointers: slow and fast pointing to the head.
2. Traverse the list using step 3 and 4
3. Move slow by one step.
4. Move fast by two steps.
5. If slow and fast meet, a loop is detected.
6. If fast or fast->next is NULL, no loop exists.

PROGRAM:

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

struct node {
    int data;
    struct node* next;
};
struct node* createNode(int val) {
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = val;
    newNode->next = NULL;
    return newNode;
}
void append(struct node** head, int val) {
    struct node* newNode = createNode(val);
    if (*head == NULL)
        *head = newNode;
    else {
        struct node* temp = *head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
    }
}

void createLoop(struct node* head, int pos) {
    struct node *temp = head, *loopNode = NULL;
    int counter = 1;
    while (temp->next != NULL) {
        if (counter == pos)
            loopNode = temp;
        temp = temp->next;
```

```
        counter++;
      }
      temp->next = loopNode;
   }

   int detectLoop(struct node* head) {
      struct node *slow = head, *fast = head;
      while (fast && fast->next) {
         slow = slow->next;
         fast = fast->next->next;
         if (slow == fast) {
            return 1; // Loop detected
         }
      }
      return 0; // No loop
   }
   int main() {
      struct node* head = NULL;
      append(&head, 10);
      append(&head, 20);
      append(&head, 30);
      append(&head, 40);
      append(&head, 50);
      createLoop(head, 3); // Creates a loop from node 50 to node 30
      if (detectLoop(head))
         printf("Loop Detected in the List\n");
      else
         printf("No Loop Detected\n");
      return 0;
}
```

OUTPUT:

```
Loop Detected in the List


...Program finished with exit code 0
Press ENTER to exit console.
```

TEST CASES:

1. Input: 10 -> 20 -> 30 -> 40 -> 50 -> Loop back to 30
   Expected Output: Loop Detected in the List

2. Input: 10 -> 20 -> 30 -> 40 -> 50 -> NULL (No Loop)
   Expected Output: No Loop Detected

RESULT:

Thus the C program to detect a loop in a circularly linked list using Floyd's Cycle-Finding Algorithm was completed successfully.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 2 c	IMPLEMENTATION OF SEGREGATION OF EVEN AND ODD NODES IN LINKED LIST
Date:	

AIM:

To rearrange a linked list by placing all even nodes before odd nodes while maintaining their relative order.

ALGORITHM:

1. Initialize pointers for even and odd lists: evenStart, evenEnd, oddStart, oddEnd.
2. Traverse the list and segregate nodes based on even or odd values.
3. Link the even list to the odd list.
4. Ensure the end of the odd list points to NULL.

PROGRAM:

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

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

struct node* createNode(int val) {
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = val;
    newNode->next = NULL;
    return newNode;
}

void append(struct node** head, int val) {
    struct node* newNode = createNode(val);
    if (*head == NULL)
        *head = newNode;
    else {
        struct node* temp = *head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
    }
}

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

```
void segregateEvenOdd(struct node** head) {
    struct node *evenStart = NULL, *evenEnd = NULL;
    struct node *oddStart = NULL, *oddEnd = NULL;
    struct node *current = *head;

    while (current != NULL) {
        int val = current->data;
        if (val % 2 == 0) {
            if (evenStart == NULL) {
                evenStart = evenEnd = current;
            } else {
                evenEnd->next = current;
                evenEnd = evenEnd->next;
            }
        } else {
            if (oddStart == NULL) {
                oddStart = oddEnd = current;
            } else {
                oddEnd->next = current;
                oddEnd = oddEnd->next;
            }
        }
        current = current->next;
    }

    if (evenStart == NULL || oddStart == NULL)
        return;

    evenEnd->next = oddStart;
    oddEnd->next = NULL;
    *head = evenStart;
}

int main() {
    struct node* head = NULL;
    append(&head, 17);
    append(&head, 15);
    append(&head, 8);
    append(&head, 12);
    append(&head, 10);
    append(&head, 5);
    append(&head, 4);

    printf("Original List:\n");
    display(head);

    segregateEvenOdd(&head);
    printf("\nAfter Segregation:\n");
    display(head);

    return 0;
}
```

OUTPUT:

```
Original List:
17 -> 15 -> 8 -> 12 -> 10 -> 5 -> 4 -> NULL

After Segregation:
8 -> 12 -> 10 -> 4 -> 17 -> 15 -> 5 -> NULL


...Program finished with exit code 0
Press ENTER to exit console.
```

TEST CASES:

1. Input: 2 -> 4 -> 6 -> 8 -> 10 -> NULL (All Even)
   Expected Output: 2 -> 4 -> 6 -> 8 -> 10 -> NULL
2. Input: Single Node (5 -> NULL)
   Expected Output: 5 -> NULL

RESULT:

Thus the C program to rearrange a linked list by placing all even nodes before odd nodes while maintaining their relative order was completed Successfully.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 3 a	IMPLEMENTATION OF STACK USING ARRAY
Date:	

AIM:

To write a program to create a Stack and manipulate it using one dimensional Array

ALGORITHM:

1. Start
2. Declare aone-dimensional array variable
3. Using Switch case statement select one appropriate operation
   a. PUSH:
      i. Read an element
      ii. Store it at the top of the stack
      iii. Increment the TOS
   b. POP
      i. Read the element from the top of the stack
      ii. Display it
      iii. Decrement the TOS
   c. PEEK
      i. Read the element from the top of the stack
      ii. Display it
4. Stop

PROGRAM:

```
// C program to create a stack with given capacity
#include <stdio.h>
#include <stdlib.h>

struct Stack {
    int top, cap;
    int *a;
};

struct Stack* createStack(int cap) {
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->cap = cap;
    stack->top = -1;
    stack->a = (int*)malloc(cap * sizeof(int));
    return stack;
}

void deleteStack(struct Stack* stack) {
    free(stack->a);
    free(stack);
```

```
}

int isFull(struct Stack* stack) {
   return stack->top >= stack->cap - 1;
}

int isEmpty(struct Stack* stack) {
   return stack->top < 0;
}

int push(struct Stack* stack, int x) {
   if (isFull(stack)) {
      printf("Stack Overflow\n");
      return 0;
   }
   stack->a[++stack->top] = x;
   return 1;
}

int pop(struct Stack* stack) {
   if (isEmpty(stack)) {
      printf("Stack Underflow\n");
      return 0;
   }
   return stack->a[stack->top--];
}

int peek(struct Stack* stack) {
   if (isEmpty(stack)) {
      printf("Stack is Empty\n");
      return 0;
   }
   return stack->a[stack->top];
}

int main() {
   struct Stack* s = createStack(5);
   push(s, 10);
   push(s, 20);
   push(s, 30);
   printf("%d popped from stack\n", pop(s));

   printf("Top element is: %d\n", peek(s));

   printf("Elements present in stack: ");
   while (!isEmpty(s)) {
      printf("%d ", peek(s));
      pop(s);
   }

   deleteStack(s);
   return 0;
}
```

OUTPUT:

30 popped from stack
Top element is: 20
Elements present in stack: 20 10

TEST CASES:

1. **Input:**
   createStack(3);
   push(1);
   push(2);
   push(3);
   push(4); // Should trigger overflow
   Expected Output:
   Stack Overflow
2. **Input:**
   createStack(2);
   push(100);
   push(200);
   pop(); // returns 200
   pop(); // returns 100
   pop(); // underflow
   Expected Output:
   200 popped from stack
   100 popped from stack
   Stack Underflow

RESULT:

Thus, the implementation of stack using array is completed and executed successfully.

EVALUATION:

PARAMETERS	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 3 b	IMPLEMENTATION OF STACK USING LINKED LIST
Date:	

AIM:

To write a program to create a Stack and manipulate it using Linked List

ALGORITHM:

1. Start
2. Read the value of ch
3. Allocate the malloc size to a variable
4. Get the item and store in data then link to Null value
5. Link the temp to top. Then check the top is Null then display as " Stack is Empty"
6. If not as Null then print the top value
7. Print data as temp becomes Null
8. Stop

PROGRAM:

```
// C program to implement a stack using singly linked list
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>

// Struct representing a node in the linked list
typedef struct Node {
   int data;
   struct Node* next;
} Node;
Node* createNode(int new_data) {
   Node* new_node = (Node*)malloc(sizeof(Node));
   new_node->data = new_data;
   new_node->next = NULL;
   return new_node;
}

// Struct to implement stack using a singly linked list
typedef struct Stack {
   Node* head;
} Stack;

// Constructor to initialize the stack
void initializeStack(Stack* stack) { stack->head = NULL; }

// Function to check if the stack is empty
int isEmpty(Stack* stack) {

   // If head is NULL, the stack is empty
   return stack->head == NULL;
}
```

```
// Function to push an element onto the stack
void push(Stack* stack, int new_data) {

    // Create a new node with given data
    Node* new_node = createNode(new_data);

    // Check if memory allocation for the new node failed
    if (!new_node) {
        printf("\nStack Overflow");
        return;
    }

    // Link the new node to the current top node
    new_node->next = stack->head;

    // Update the top to the new node
    stack->head = new_node;
}

// Function to remove the top element from the stack
void pop(Stack* stack) {

    // Check for stack underflow
    if (isEmpty(stack)) {
        printf("\nStack Underflow\n");
        return;
    }
    else {

        // Assign the current top to a temporary variable
        Node* temp = stack->head;

        // Update the top to the next node
        stack->head = stack->head->next;

        // Deallocate the memory of the old top node
        free(temp);
    }
}

// Function to return the top element of the stack
int peek(Stack* stack) {

    // If stack is not empty, return the top element
    if (!isEmpty(stack))
        return stack->head->data;
    else {
        printf("\nStack is empty");
        return INT_MIN;
    }
}
```

```
// Driver program to test the stack implementation
int main() {

    // Creating a stack
    Stack stack;
    initializeStack(&stack);

    // Push elements onto the stack
    push(&stack, 11);
    push(&stack, 22);
    push(&stack, 33);
    push(&stack, 44);

    // Print top element of the stack
    printf("Top element is %d\n", peek(&stack));


        // removing two elemements from the top
        printf("Removing two elements...\n");
    pop(&stack);
    pop(&stack);

    // Print top element of the stack
    printf("Top element is %d\n", peek(&stack));

    return 0;
}
```

OUTPUT:

```
Top element is 44
Removing two elements...
Top element is 22
```

TEST CASES:

1. Input:
   ```
   push(11);
   push(22);
   push(33);
   push(44);
   peek();
   ```
   Expected Output:
   ```
   Top element is 44
   ```
2. Input:
   ```
   pop(); // removes 44
   pop(); // removes 33
   pop(); // removes 22
   pop(); // removes 11
   pop(); // stack underflow
   ```
   Expected output:
   ```
   Stack Underflow
   ```

RESULT:

The given program is implemented, executed, tested and verified successfully

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 3 c	IMPLEMENTATION OF QUEUE USING ARRAY
Date:	

AIM:

To write a program to create a Queue using one dimensional Array

ALGORITHM:

1. Start

2. Read ch and n value

3. check (ch!=4) and if (rear =n) then read value

4. Assign value to q(rear) and increment the rear, else Display as queue if full

5. if (front=rear) then print front value of queue and increment front Else Display as queue is empty

6. if (rear=0) then assign front to I and write q[i]. Else Display as queue is empty

7. Go to step 3

8. stop

PROGRAM:

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

struct Queue {
   int *arr;
   int front;
   int rear;
   int capacity;
};

struct Queue* createQueue(int capacity) {
   struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
   queue->capacity = capacity;
   queue->front = 0;
   queue->rear = -1;
   queue->arr = (int*)malloc(capacity * sizeof(int));
   return queue;
}

int isEmpty(struct Queue* queue) {
   return queue->front > queue->rear;
}

void enqueue(struct Queue* queue, int x) {
   if (queue->rear < queue->capacity - 1) {
      queue->arr[++queue->rear] = x;
   }
}
```

```
void dequeue(struct Queue* queue) {
    if (!isEmpty(queue)) {
        queue->front++;
    }
}

int getFront(struct Queue* queue) {
    return isEmpty(queue) ? -1 : queue->arr[queue->front];
}

void display(struct Queue* queue) {
    for (int i = queue->front; i <= queue->rear; i++) {
        printf("%d ", queue->arr[i]);
    }
    printf("\n");
}

int main() {
    struct Queue* q = createQueue(100);
    enqueue(q, 1);
    enqueue(q, 2);
    enqueue(q, 3);
    printf("%d\n", getFront(q));
    dequeue(q);
    enqueue(q, 4);
    display(q);
    return 0;
}
```

OUTPUT:

1
2 3 4

TEST CASES:

1) Input:
   enqueue(q, 1);
   enqueue(q, 2);
   enqueue(q, 3);
   getFront(q);
   Expected Output:
   1
2) Input:
   enqueue(q, 4); // queue is now 2, 3, 4
   display(q);
   Expected Output:
   2 3 4

RESULT:

Thus the implementation of queue using array is completed and executed successfully.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 3 d	IMPLEMENTATION OF QUEUE USING LINKED LIST
Date:	

AIM:

To write a program to create a Queue using Linked list

ALGORITHM:

1. Start
2. Read the value of ch
3. Get the variable temp and set malloc for size
4. Get item and store in data and link as Null
5. Get link of rear to temp. If front as Null and print Queue as Empty
6. Else print front element
7. print the data until the temp get Null
8. Stop

PROGRAM:

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

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

// Queue structure definition
struct Queue {
   struct Node* front;
   struct Node* rear;
};

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

// Function to initialize the queue
struct Queue* createQueue() {
   struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
   q->front = q->rear = NULL;
   return q;
}
```

```
// Function to check if the queue is empty
int isEmpty(struct Queue* q) {
    return q->front == NULL;
}

// Function to add an element to the queue
void enqueue(struct Queue* q, int new_data) {
    struct Node* new_node = newNode(new_data);
    if (isEmpty(q)) {
        q->front = q->rear = new_node;
        printQueue(q);
        return;
    }
    q->rear->next = new_node;
    q->rear = new_node;
    printQueue(q);
}

// Function to remove an element from the queue
void dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        return;
    }
    struct Node* temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) q->rear = NULL;
    free(temp);
    printQueue(q);
}

// Function to print the current state of the queue
void printQueue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return;
    }
    struct Node* temp = q->front;
    printf("Current Queue: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Driver code to test the queue implementation
int main() {
    struct Queue* q = createQueue();

    // Enqueue elements into the queue
    enqueue(q, 10);
    enqueue(q, 20);
```

```
    // Dequeue elements from the queue
    dequeue(q);
    dequeue(q);

    // Enqueue more elements into the queue
    enqueue(q, 30);
    enqueue(q, 40);
    enqueue(q, 50);

    // Dequeue an element from the queue (this should print 30)
    dequeue(q);

    return 0;
}
```

OUTPUT:

```
Current Queue: 10
Current Queue: 10 20
Current Queue: 20
Queue is empty
Current Queue: 30
Current Queue: 30 40
Current Queue: 30 40 50
Current Queue: 40 50
```

TEST CASES:

1. Input:
   enqueue(q, 10);
   enqueue(q, 20);
   Expected Output:
   Current Queue: 10
   Current Queue: 10 20
2. Input:
   dequeue(q); // Removes 10
   dequeue(q); // Removes 20
   Exptected Output:
   Current Queue: 20
   Queue is empty

RESULT:

Thus the program is executed and the required output is obtained

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 4 a	IMPLEMENTATION OF POSTFIX EVALUATION
Date:	

AIM:

To write a program to implement postfix evaluation

ALGORITHM:

1. Start
2. Read postfix expression Left to Right until) encountered
3. If operand is encountered, push it onto Stack
4. If operator is encountered, Pop two elements
   i) A -> Top element
   ii) B-> Next to Top element
   iii) Evaluate B operator A
5. push B operator A onto Stack
6. Set result = pop
7. END

PROGRAM:

```
#include<stdio.h> int stack[20];
int top = -1;
void push(int x){
	stack[++top] = x;
}
int pop(){
	return stack[top--];
}
int main(){
	char exp[20];
	char *e;
	int n1,n2,n3,num;
	printf("Enter the expression :: ");
	scanf("%s",exp);
	e = exp;
	while(*e != '\0'){
		if(isdigit(*e)){
			num = *e - 48;
			push(num);
		}
		else{
			n1 = pop();
			n2 = pop();
			switch(*e){
			case '+':
				n3 = n1 + n2;
				break;
			case '-':
				n3 = n2 - n1;
```

```
                        break;
                case '*':
                        n3 = n1 * n2;
                        break;
                case '/':
                        n3 = n2 / n1;
                        break;
                }
                push(n3);
            }
            e++;
        }
        printf("\nThe result of expression %s = %d\n\n",exp,pop());
        return 0;
}
```

OUTPUT:

Enter the expression: 245+*
The result of expression 245+* = 18

TEST CASES:

1) Input: 23*54*+
   Expected Output:
   The result of expression 23*54*+ = 26
2) Input: 42/3+
   Expected output:
   The result of expression 42/3+ = 5

RESULT:

Thus, the program is executed and the required output is obtained.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 4 b	IMPLEMENTATION OF BALANCING PARANTHESES
Date:	

AIM:

To write a C program that checks whether a given string of parentheses/brackets is balanced using a stack.

ALGORITHM:

1. Initialize a stack (array + top = -1).
2. Iterate through each character in the string:
   i) If character is an opening bracket ((, {, [): Push it onto the stack.
   ii) If character is a closing bracket (), },]): Check if the stack is not empty AND the top of the stack matches the corresponding opening bracket:
      - If matched, pop the top of the stack.
      - If not matched or stack is empty, return Not Balanced.
3. After iterating through the string:
   i) If the stack is empty (top == -1), return Balanced.
   ii) If the stack still contains items, return Not Balanced.

PROGRAM:

```
#include <stdio.h>
int top = -1;
char stack[10];
int isEmpty(){
   if(top == -1)
      return 1;
   else
      return 0;
}
int isBalanced(char str[]){

   for(int i=0;i<4;i++){
      if(str[i] == '(' || str[i] == '{' || str[i] == '['){
         stack[++top] = str[i];
      }
      else{
         if(!isEmpty() &&
            ((stack[top] == '(' && str[i] == ')') ||
             (stack[top] == '{' && str[i] == '}') ||
             (stack[top] == '[' && str[i] == ']')))
             top--;
         else{
            return 0;
         }
      }
   }
   return 1;
}
int main()
{
```

```
    char str[10]="{[]}";
    if(isBalanced(str)){
        printf("Balanced");
    }
    else{
        printf("Not Balanced");
    }
    return 0;
}
```

OUTPUT:

{[]}
Balanced

TEST CASE:

1) Input String: {[}
   Expected Output: Not Balanced
2) Input String: ()[{}]
   Expected Output: Balanced

RESULT:

Thus the implementation of parentheses balance check using a stack is executed successfully.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 5	IMPLEMENTATION OF BINARY TREE
Date:	

AIM:

To develop a movie recommendation system using a binary tree, enabling efficient storage, search, and retrieval of movies based on users' past watching history and preferences.

ALGORITHM:

1. Define Structures for Movie Nodes.
2. Create Binary Tree of Movies.
3. Traverse and Display All Movies (In-Order Traversal).
4. Implement Search Operation Based on Movie ID.
5. Recommend Movies Based on Genre.

PROGRAM:

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

#define MAX_TITLE 100
#define MAX_GENRE 30

// Define a node for a movie
typedef struct Movie
{
    int id;
    char title[MAX_TITLE];
    char genre[MAX_GENRE];
    struct Movie *left;
    struct Movie *right;
} Movie;

// Function prototypes
Movie* createMovieNode();
Movie* searchMovie(Movie* root, int key);
void inorder(Movie* root);
void recommendMovies(Movie* root);

// Main Function
int main()
{
    Movie* root = NULL;

    printf("Create your movie binary tree (enter -1 for movie ID to stop):\n");
    root = createMovieNode();

    printf("\nAll Movies (In-Order Traversal):\n");
    inorder(root);
```

```
    printf("\nRecommended Movies (based on genre preferences):\n");
    recommendMovies(root);

    int searchId;
    printf("\nEnter Movie ID to search: ");
    scanf("%d", &searchId);

    Movie* found = searchMovie(root, searchId);
    if (found)
    {
        printf("Found Movie: %s [%s]\n", found->title, found->genre);
    }
    else
    {
        printf("Movie with ID %d not found.\n", searchId);
    }
    return 0;
}

// Function to create a node
Movie* createMovieNode()
{
    int id;
    printf("Enter Movie ID (-1 to return NULL): ");
    scanf("%d", &id);
    if (id == -1)
        return NULL;

    Movie* node = (Movie*)malloc(sizeof(Movie));
    node->id = id;

    printf("Enter Movie Title: ");
    getchar();                                          // clear newline from buffer
    fgets(node->title, MAX_TITLE, stdin);
    node->title[strcspn(node->title, "\n")] = 0;        // remove newline

    printf("Enter Genre: ");
    fgets(node->genre, MAX_GENRE, stdin);
    node->genre[strcspn(node->genre, "\n")] = 0;

    printf("Enter left child of \"%s\" (ID %d):\n", node->title, node->id);
    node->left = createMovieNode();

    printf("Enter right child of \"%s\" (ID %d):\n", node->title, node->id);
    node->right = createMovieNode();

    return node;
}

// In-order traversal
void inorder(Movie* root)
{
```

```
    if (root == NULL)
        return;
    inorder(root->left);
    printf("ID: %d | Title: %s | Genre: %s\n", root->id, root->title, root->genre);
    inorder(root->right);
}

// Search by Movie ID
Movie* searchMovie(Movie* root, int key)
{
    if (root == NULL)
        return NULL;
    if (root->id == key)
        return root;

    Movie* leftSearch = searchMovie(root->left, key);
    if (leftSearch != NULL)
        return leftSearch;

    return searchMovie(root->right, key);
}

// Recommendation based on genre
void recommendMovies(Movie* root)
{
    if (root == NULL)
        return;

    recommendMovies(root->left);

    // Simulate preference: Recommend if genre is Action or Comedy
    if (strcmp(root->genre, "Action") == 0 || strcmp(root->genre, "Comedy") == 0)
    {
        printf("Recommended: %s [%s]\n", root->title, root->genre);
    }
    recommendMovies(root->right);
}
```

OUTPUT:

```
Create your movie binary tree (enter -1 for movie ID to stop):
Enter Movie ID (-1 to return NULL): 1
Enter Movie Title: Avengers
Enter Genre: Action
Enter left child of "Avengers" (ID 1):
Enter Movie ID (-1 to return NULL): 2
Enter Movie Title: Home Alone
Enter Genre: Comedy
Enter left child of "Home Alone" (ID 2):
Enter Movie ID (-1 to return NULL): -1
Enter right child of "Home Alone" (ID 2):
Enter Movie ID (-1 to return NULL): -1
```

Enter right child of "Avengers" (ID 1):
Enter Movie ID (-1 to return NULL): 3
Enter Movie Title: The Note Book
Enter Genre: Romance
Enter left child of "The Note Book" (ID 3):
Enter Movie ID (-1 to return NULL): -1
Enter right child of "The Note Book" (ID 3):
Enter Movie ID (-1 to return NULL): -1

All Movies (In-Order Traversal):
ID: 2 | Title: Home Alone | Genre: Comedy
ID: 1 | Title: Avengers | Genre: Action
ID: 3 | Title: The Note Book | Genre: Romance

Recommended Movies (based on genre preferences):
Recommended: Home Alone [Comedy]
Recommended: Avengers [Action]

Enter Movie ID to search: 2
Found Movie: Home Alone [Comedy]

TEST CASES:

Test Case 1: Basic Tree with Mixed Genres
Movie ID: 1
Title: Avengers
Genre: Action

Left child of 1:
Movie ID: 2
Title: Home Alone
Genre: Comedy

Left child of 2: -1
Right child of 2: -1

Right child of 1:
Movie ID: 3
Title: Titanic
Genre: Romance

Left child of 3: -1
Right child of 3: -1

Search ID: 2

All Movies (In-Order Traversal):
ID: 2 | Title: Home Alone | Genre: Comedy
ID: 1 | Title: Avengers | Genre: Action
ID: 3 | Title: Titanic | Genre: Romance

Recommended Movies:

Recommended: Home Alone [Comedy]
Recommended: Avengers [Action]

Found Movie: Home Alone [Comedy]

Test Case 2: Tree with No Preferred Genre
Movie ID: 10
Title: The Godfather
Genre: Crime

Left child of 10:
Movie ID: 20
Title: A Beautiful Mind
Genre: Drama

Left/Right of 20: -1

Right child of 10:
Movie ID: 30
Title: Schindler's List
Genre: History

Left/Right of 30: -1

Search ID: 30

All Movies (In-Order Traversal):
ID: 20 | Title: A Beautiful Mind | Genre: Drama
ID: 10 | Title: The Godfather | Genre: Crime
ID: 30 | Title: Schindler's List | Genre: History

Recommended Movies:

Found Movie: Schindler's List [History]

Test Case 3: Single Node Tree
Movie ID: 99
Title: Rush Hour
Genre: Comedy

Left/Right child: -1

Search ID: 99

All Movies (In-Order Traversal):
ID: 99 | Title: Rush Hour | Genre: Comedy

Recommended Movies:
Recommended: Rush Hour [Comedy]

Found Movie: Rush Hour [Comedy]

RESULT:

Thus, the C program to develop a movie recommendation system using a binary tree was completed successfully, enabling efficient storage, search, and retrieval of movies based on users' past watching history and preferences.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 6	IMPLEMENTATION OF BINARY SEARCH TREE
Date:	

AIM:

To manage the online bookstore inventory using a Binary Search Tree (BST), enabling efficient searching, insertion, and deletion of books to provide a seamless user experience.

ALGORITHM:

1) Define Structure for Book Node.
2) Create a New Book Node.
3) Insert Book into Binary Search Tree.
4) Search for a Book by ID.
5) Display All Books in Inventory (Sorted by ID).

PROGRAM:

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

typedef struct Book
{
   int id;
   char title[100];
   char author[100];
   float price;
   struct Book* left;
   struct Book* right;
} Book;

// Create a new book node
Book* createBook(int id, const char* title, const char* author, float price)
{
   Book* newBook = (Book*)malloc(sizeof(Book));
   newBook->id = id;
   strcpy(newBook->title, title);
   strcpy(newBook->author, author);
   newBook->price = price;
   newBook->left = newBook->right = NULL;
   return newBook;
}

// Insert into BST
Book* insertBook(Book* root, Book* newBook)
{
```

```
    if (root == NULL)
        return newBook;
    if (newBook->id < root->id)
        root->left = insertBook(root->left, newBook);
    else if (newBook->id > root->id)
        root->right = insertBook(root->right, newBook);
    else
        printf("Book ID %d already exists. Skipping duplicate.\n", newBook->id);
    return root;
}

// Search for a book by ID
Book* searchBook(Book* root, int id)
{
    if (root == NULL || root->id == id)
        return root;
    if (id < root->id)
        return searchBook(root->left, id);
    return searchBook(root->right, id);
}

// In-order traversal (sorted by Book ID)
void displayInventory(Book* root)
{
    if (root == NULL) return;
    displayInventory(root->left);
    printf("ID: %d | Title: %s | Author: %s | Price:$%.2f\n",root->id,root->title,root->author,root->price);
    displayInventory(root->right);
}

// Menu-driven interaction
int main()
{
    Book* inventory = NULL;
    int choice, id;
    char title[100], author[100];
    float price;

    while (1)
    {
        printf("\n====== Online Bookstore ======\n");
        printf("1. Add Book\n");
        printf("2. Search Book by ID\n");
        printf("3. Display Inventory\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            printf("Enter Book ID: ");
            scanf("%d", &id);
```

```
            printf("Enter Title: ");
            getchar();                                          // Clear newline
            fgets(title, 100, stdin);
            title[strcspn(title, "\n")] = 0;
            printf("Enter Author: ");
            fgets(author, 100, stdin);
            author[strcspn(author, "\n")] = 0;
            printf("Enter Price: ");
            scanf("%f", &price);
            inventory = insertBook(inventory, createBook(id, title, author, price));
            break;

        case 2:
            printf("Enter Book ID to search: ");
            scanf("%d", &id);
            Book* result = searchBook(inventory, id);
            if (result)
            {
                printf("Book Found:\n");
                printf("Title: %s\nAuthor: %s\nPrice: $%.2f\n", result->title, result->author, result->price);
            }
            else
            {
                printf("Book with ID %d not found.\n", id);
            }
            break;

        case 3:
            printf("\nInventory:\n");
            displayInventory(inventory);
            break;

        case 4:
            printf("Exiting...\n");
            return 0;

        default:
            printf("Invalid choice.\n");
        }
    }
}
```

OUTPUT:

```
====== Online Bookstore ======
1. Add Book
2. Search Book by ID
3. Display Inventory
4. Exit
Enter your choice: 1
Enter Book ID: 101
Enter Title: The Alchemist
```

Enter Author: Paulo Coelho
Enter Price: 9.99

====== Online Bookstore ======
1. Add Book
2. Search Book by ID
3. Display Inventory
4. Exit
Enter your choice: 1
Enter Book ID: 203
Enter Title: Clean Code
Enter Author: Robert Martin
Enter Price: 29.99

====== Online Bookstore ======
1. Add Book
2. Search Book by ID
3. Display Inventory
4. Exit
Enter your choice: 3

Inventory:
ID: 101 | Title: The Alchemist | Author: Paulo Coelho | Price: $9.99
ID: 203 | Title: Clean Code | Author: Robert Martin | Price: $29.99

====== Online Bookstore ======
1. Add Book
2. Search Book by ID
3. Display Inventory
4. Exit
Enter your choice: 2
Enter Book ID to search: 203
Book Found:
Title: Clean Code
Author: Robert Martin
Price: $29.99

====== Online Bookstore ======
1. Add Book
2. Search Book by ID
3. Display Inventory
4. Exit
Enter your choice: 4
Exiting...

TEST CASES:

Test Case 1: Add and Search for Multiple Books
Add Book:
ID: 100
Title: The Hobbit
Author: J.R.R. Tolkien

Price: 12.99

Add Book:
ID: 50
Title: 1984
Author: George Orwell
Price: 8.99
Add Book:
ID: 150
Title: To Kill a Mockingbird
Author: Harper Lee
Price: 10.49

Search Book by ID: 50
Book Found:
Title: 1984
Author: George Orwell
Price: $8.99

Test Case 2: Display Full Inventory (In-Order Traversal)
Insert:
ID: 300, Title: Dune
ID: 200, Title: Brave New World
ID: 400, Title: The Catcher in the Rye
Inventory:
ID: 200 | Title: Brave New World | Author: ...
ID: 300 | Title: Dune | Author: ...
ID: 400 | Title: The Catcher in the Rye | Author: ...

Test Case 3: Search for Non-Existent Book
Add Book:
ID: 10, Title: Sapiens
Search Book by ID: 99
Book with ID 99 not found.

RESULT:

Thus, the C program to manage the online bookstore inventory using a Binary Search Tree (BST) was completed successfully, enabling efficient searching, insertion, and deletion of books to provide a seamless user experience.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 7a	SEGMENT TREE AND RANGE OF MINIMUM QUERY
Date:	

AIM:

To construct a segment tree and perform efficient range minimum queries, showcasing the practical use of advanced tree data structures for optimized interval queries.

ALGORITHM:

1. Initialize Input Array.
2. Calculate Segment Tree Size.
3. Allocate Memory for Segment Tree.
4. Build the Segment Tree.
5. Perform Range Minimum Queries.
6. Print Results.

PROGRAM:

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

// Utility to find minimum
int min(int x, int y)
{
    return (x < y) ? x : y;
}

// Function to build the segment tree
void buildSegmentTree(int arr[], int segTree[], int low, int high, int pos)
{
    if (low == high)
    {
        segTree[pos] = arr[low];
        return;
    }

    int mid = (low + high) / 2;
    buildSegmentTree(arr, segTree, low, mid, 2 * pos + 1);
    buildSegmentTree(arr, segTree, mid + 1, high, 2 * pos + 2);

    segTree[pos] = min(segTree[2 * pos + 1], segTree[2 * pos + 2]);
}

// Function to query the minimum in a range
int rangeMinQuery(int segTree[], int qlow, int qhigh, int low, int high, int pos)
```

```
{
    // Total overlap
    if (qlow <= low && qhigh >= high)
        return segTree[pos];

    // No overlap
    if (qlow > high || qhigh < low)
        return INT_MAX;

    // Partial overlap
    int mid = (low + high) / 2;
    return min(rangeMinQuery(segTree, qlow, qhigh, low, mid, 2 * pos + 1),
        rangeMinQuery(segTree, qlow, qhigh, mid + 1, high, 2 * pos + 2));
}

int main()
{
    int arr[] = {4, 2, 1, 5, 3, 8, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Height of segment tree
    int height = (int)(ceil(log2(n)));
    int maxSize = 2 * (int)pow(2, height) - 1;

    int *segTree = (int *)malloc(sizeof(int) * maxSize);

    buildSegmentTree(arr, segTree, 0, n - 1, 0);

    int qlow, qhigh;

    printf("Array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    // Sample queries
    printf("\n\nRange Minimum Queries:\n");

    qlow = 1;
    qhigh = 4;
    printf("Min from index %d to %d: %d\n",qlow,qhigh,rangeMinQuery(segTree,qlow,qhigh,0,n-1,0));

    qlow = 0;
    qhigh = 7;
    printf("Min from index %d to %d: %d\n",qlow,qhigh,rangeMinQuery(segTree,qlow,qhigh,0,n-1,0));

    qlow = 3;
    qhigh = 5;
    printf("Min from index %d to %d: %d\n",qlow,qhigh,rangeMinQuery(segTree,qlow,qhigh,0,n-1,0));

    free(segTree);
    return 0;
}
```

OUTPUT:

Array: 4 2 1 5 3 8 6 7

Range Minimum Queries:
Min from index 1 to 4: 1
Min from index 0 to 7: 1
Min from index 3 to 5: 3

TEST CASES:

Test Case 1: Small Array with Distinct Elements
arr[] = {5, 2, 6, 3, 1, 7}
Query Range: 1 to 4
Minimum from index 1 to 4: 1

Test Case 2: Full Array Query
arr[] = {10, 20, 15, 30, 5, 25}
Query Range: 0 to 5
Minimum from index 0 to 5: 5

Test Case 3: Query on Single Element
arr[] = {4, 8, 6, 1, 9, 2}
Query Range: 3 to 3
Minimum from index 3 to 3: 1

RESULT:

Thus, the C program to construct a segment tree and perform efficient range minimum queries was completed successfully, showcasing the practical use of advanced tree data structures for optimized interval queries.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 7b	MAXIMUM DEPTH OF BINARY TREE
Date:	

AIM:

To determine the maximum depth of a binary tree.

ALGORITHM:

1. Define Node Structure.
2. Construct the Binary Tree.
3. Compute Depth of the Tree.
4. Print Maximum Depth.

PROGRAM:

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

// Node structure
struct Node
{
   int data;
   struct Node* left;
   struct Node* right;
};

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

// Calculate max depth of binary tree
int maxDepth(struct Node* root)
{
   if (root == NULL)
      return 0;

   int leftDepth = maxDepth(root->left);
   int rightDepth = maxDepth(root->right);

   return (leftDepth > rightDepth ? leftDepth : rightDepth) + 1;
}
```

```
int main()
{
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);

    int depth = maxDepth(root);
    printf("Maximum Depth of the Binary Tree: %d\n", depth);

    return 0;
}
```

OUTPUT:

Maximum Depth of the Binary Tree: 3

TEST CASES:

Test Case 1: Balanced Tree
Input Tree:

```
   1
  / \
 2   3
/ \
4   5
```

Expected Output: 3

Test Case 2: Right-skewed Tree
Input Tree:

```
1
 \
  2
   \
    3
     \
      4
```

Expected Output: 4

Test Case 3: Single Node Tree
Input Tree:
1

Expected Output: 1

RESULT:

Thus, the C program to determine the maximum depth of a binary tree was completed successfully.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 8	GRAPH TRAVERSAL
Date:	

AIM:

To model the food delivery network using graph traversal algorithm, enabling the understanding of user connections, tracking information flow, and identifying influential users within the network.

ALGORITHM:

1. Define Graph Structures.
2. Create Graph.
3. Add Undirected Edges.
4. Do Breadth-First Search (BFS) Traversal.
5. Find the Most Influential User.

PROGRAM:

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

#define MAX_USERS 100

// Node for adjacency list
typedef struct AdjNode
{
    int user;
    struct AdjNode* next;
} AdjNode;

// Graph structure
typedef struct Graph
{
    int numUsers;
    AdjNode* adjList[MAX_USERS];
} Graph;

// Create graph with numUsers nodes
Graph* createGraph(int numUsers)
{
    Graph* graph = (Graph*)malloc(sizeof(Graph));
    graph->numUsers = numUsers;
    for (int i = 0; i < numUsers; i++)
        graph->adjList[i] = NULL;
    return graph;
}
```

```
// Add edge (undirected)
void addEdge(Graph* graph, int src, int dest)
{
    AdjNode* newNode = (AdjNode*)malloc(sizeof(AdjNode));
    newNode->user = dest;
    newNode->next = graph->adjList[src];
    graph->adjList[src] = newNode;

    // Since undirected, add edge in both directions
    newNode = (AdjNode*)malloc(sizeof(AdjNode));
    newNode->user = src;
    newNode->next = graph->adjList[dest];
    graph->adjList[dest] = newNode;
}

// BFS to find reach of user (number of users reachable)
int bfs(Graph* graph, int startUser)
{
    int visited[MAX_USERS] = {0};
    int queue[MAX_USERS];
    int front = 0, rear = 0;

    visited[startUser] = 1;
    queue[rear++] = startUser;

    int count = 1;                                          // count self

    while (front < rear)
    {
        int curr = queue[front++];
        AdjNode* temp = graph->adjList[curr];
        while (temp != NULL)
        {
            int adjUser = temp->user;
            if (!visited[adjUser])
            {
                visited[adjUser] = 1;
                queue[rear++] = adjUser;
                count++;
            }
            temp = temp->next;
        }
    }
    return count;
}

// Find most influential user by max reach
int findInfluentialUser(Graph* graph)
{
    int maxReach = 0;
    int influentialUser = -1;
    for (int i = 0; i < graph->numUsers; i++)
    {
```

```
            int reach = bfs(graph, i);
            //printf("User %d can reach %d users\n", i, reach);
            if (reach > maxReach)
            {
                maxReach = reach;
                influentialUser = i;
            }
        }
        return influentialUser;
    }

    int main()
    {
        int numUsers = 6;
        Graph* network = createGraph(numUsers);

        // Add connections (edges) between users
        addEdge(network, 0, 1);
        addEdge(network, 0, 2);
        addEdge(network, 1, 3);
        addEdge(network, 2, 4);
        addEdge(network, 4, 5);

        int influentialUser = findInfluentialUser(network);
        printf("Most influential user is User %d\n", influentialUser);

        return 0;
    }
```

OUTPUT:

Most influential user is User 0

TEST CASES:

Test Case 1: Simple Chain Network
User 0 — User 1 — User 2 — User 3
Edges:

- (0,1)
- (1,2)
- (2,3)

Most influential user is User 0

Test Case 2: Star Network

```
          User 0
      /      |      \
```

User 1 User 2 User 3
Edges:

- (0,1)

- (0,2)
- (0,3)

Most influential user is User 0

Test Case 3: Disconnected Graph
Component 1: User 0 — User 1
Component 2: User 2 — User 3 — User 4
Edges:

- (0,1)
- (2,3)
- (3,4)

Most influential user is User 2

RESULT:

Thus, the C program to model the food delivery network using graph traversal algorithms was completed successfully, enabling the understanding of user connections, tracking information flow, and identifying influential users within the network.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 9	IMPLEMENTATION OF TOPOLOGICAL SORT
Date:	

AIM:

To streamline travel planning and organize travel itineraries using graph algorithms - Topological Sort, Connectivity Check, and Ticket Itinerary reconstruction.

ALGORITHM:

1. Setup structures.
2. Construct graph.
3. Do Topological Sort.
4. Check graph connectivity.
5. Reconstruct ticket itinerary.

PROGRAM:

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

#define MAX_CITIES 100
#define MAX_TICKETS 1000

// Graph structures
typedef struct AdjNode
{
   int cityIndex;
   struct AdjNode* next;
} AdjNode;

typedef struct Graph
{
   int numCities;
   char* cityNames[MAX_CITIES];
   AdjNode* adjList[MAX_CITIES];
} Graph;

// Ticket structure
typedef struct Ticket
{
   int src;
   int dest;
```

```
} Ticket;

int edgeCount[MAX_CITIES][MAX_CITIES]; // For itinerary reconstruction

// Create graph with n cities
Graph* createGraph(int numCities)
{
   Graph* graph = (Graph*) malloc(sizeof(Graph));
   graph->numCities = numCities;
   for (int i = 0; i < numCities; i++)
      graph->adjList[i] = NULL;
   return graph;
}

// Add edge directed graph
void addEdge(Graph* graph, int src, int dest)
{
   AdjNode* newNode = (AdjNode*) malloc(sizeof(AdjNode));
   newNode->cityIndex = dest;
   newNode->next = graph->adjList[src];
   graph->adjList[src] = newNode;
}

// Topological Sort (Kahn's Algorithm)
void topologicalSort(Graph* graph)
{
   int indegree[MAX_CITIES] = {0};
   int numCities = graph->numCities;

   for (int i = 0; i < numCities; i++)
   {
      AdjNode* temp = graph->adjList[i];
      while (temp)
      {
         indegree[temp->cityIndex]++;
         temp = temp->next;
      }
   }

   int queue[MAX_CITIES], front = 0, rear = 0;
   for (int i = 0; i < numCities; i++)
   {
      if (indegree[i] == 0)
         queue[rear++] = i;
   }

   int count = 0;
   printf("Travel order (Topological Sort): ");
   while (front < rear)
   {
      int u = queue[front++];
      printf("%s ", graph->cityNames[u]);
      count++;
```

```
        AdjNode* temp = graph->adjList[u];
        while (temp)
        {
            indegree[temp->cityIndex]--;
            if (indegree[temp->cityIndex] == 0)
                queue[rear++] = temp->cityIndex;
            temp = temp->next;
        }
    }

    if (count != numCities)
        printf("\nCycle detected! Trip plan invalid.\n");
    else
        printf("\n");
}

// DFS for connectivity
void dfs(Graph* graph, int start, int visited[])
{
    visited[start] = 1;
    AdjNode* temp = graph->adjList[start];
    while (temp)
    {
        if (!visited[temp->cityIndex])
            dfs(graph, temp->cityIndex, visited);
        temp = temp->next;
    }
}

int isConnected(Graph* graph, int start)
{
    int visited[MAX_CITIES] = {0};
    dfs(graph, start, visited);

    for (int i = 0; i < graph->numCities; i++)
    {
        if (!visited[i])
            return 0;
    }
    return 1;
}

// Build graph from tickets and fill edgeCount
void buildGraphFromTickets(Graph* graph, Ticket tickets[], int n)
{
    for (int i = 0; i < graph->numCities; i++)
        for (int j = 0; j < graph->numCities; j++)
            edgeCount[i][j] = 0;

    for (int i = 0; i < n; i++)
    {
        addEdge(graph, tickets[i].src, tickets[i].dest);
```

```
        edgeCount[tickets[i].src][tickets[i].dest]++;
    }
}

// Backtracking to find valid itinerary
bool visitTickets(Graph* graph, int currentCity, int totalTickets, int usedTickets, int itinerary[], int pos)
{
    if (usedTickets == totalTickets)
    {
        itinerary[pos] = currentCity;
        return true;
    }

    itinerary[pos] = currentCity;

    for (AdjNode* temp = graph->adjList[currentCity]; temp != NULL; temp = temp->next)
    {
        int nextCity = temp->cityIndex;
        if (edgeCount[currentCity][nextCity] > 0)
        {
            edgeCount[currentCity][nextCity]--;
            if (visitTickets(graph, nextCity, totalTickets, usedTickets + 1, itinerary, pos + 1))
                return true;
            edgeCount[currentCity][nextCity]++; // backtrack
        }
    }
    return false;
}

int main()
{
    // Cities
    char* cities[] = {"A", "B", "C", "D"};
    int numCities = 4;

    Graph* graph = createGraph(numCities);
    for (int i = 0; i < numCities; i++)
        graph->cityNames[i] = cities[i];

    // --- Topological Sort Example ---
    addEdge(graph, 0, 1);                                   // A -> B
    addEdge(graph, 1, 2);                                   // B -> C
    addEdge(graph, 0, 3);                                   // A -> D

    printf("\n=== Topological Sort ===\n");
    topologicalSort(graph);

    // --- Connectivity Check ---
    printf("\n=== Connectivity Check ===\n");
    if (isConnected(graph, 0))
        printf("All cities reachable from %s\n", cities[0]);
    else
        printf("Not all cities reachable from %s\n", cities[0]);
```

```
    // --- Ticket Itinerary Reconstruction ---
    printf("\n=== Ticket Itinerary Reconstruction ===\n");

    // Tickets: A->B, B->C, C->D
    Ticket tickets[] =
    {
        {0, 1},
        {1, 2},
        {2, 3}
    };
    int totalTickets = sizeof(tickets) / sizeof(tickets[0]);

    // Reset graph for tickets
    graph = createGraph(numCities);
    for (int i = 0; i < numCities; i++)
        graph->cityNames[i] = cities[i];

    buildGraphFromTickets(graph, tickets, totalTickets);

    int itinerary[MAX_TICKETS];
    if (visitTickets(graph, 0, totalTickets, 0, itinerary, 0))
    {
        printf("Itinerary: ");
        for (int i = 0; i <= totalTickets; i++)
        {
            printf("%s", cities[itinerary[i]]);
            if (i != totalTickets)
                printf(" -> ");
        }
        printf("\n");
    }
    else
    {
        printf("No valid itinerary found\n");
    }

    return 0;
}
```

OUTPUT:

```
=== Topological Sort ===
Travel order (Topological Sort): A B D C

=== Connectivity Check ===
Not all cities reachable from A

=== Ticket Itinerary Reconstruction ===
Itinerary: A -> B -> C -> D
```

TEST CASES:

Test Case 1: Simple Linear Dependency (No Cycle)

- Cities: A, B, C, D
- Edges for Topological Sort:
  A -> B, B -> C, C -> D
- Tickets:
  A -> B, B -> C, C -> D
- Start city for connectivity: A

Travel order (Topological Sort): A B C D
All cities reachable from A
Itinerary: A -> B -> C -> D

Test Case 2: Graph with Cycle (Invalid Trip)

- Cities: A, B, C
- Edges for Topological Sort:
  A -> B, B -> C, C -> A (cycle)
- Tickets:
  A -> B, B -> C, C -> A
- Start city for connectivity: A

Travel order (Topological Sort):
Cycle detected! Trip plan invalid.
All cities reachable from A
Itinerary: A -> B -> C -> A

Test Case 3: Disconnected Graph

- Cities: A, B, C, D
- Edges for Topological Sort:
  A -> B, C -> D
- Tickets:
  A -> B, C -> D
- Start city for connectivity: A

Travel order (Topological Sort): A B C D
Not all cities reachable from A
Itinerary: No valid itinerary found

RESULT:

Thus, the C program to streamline travel planning and organize travel itineraries using graph algorithms - Topological Sort, Connectivity Check, and Ticket Itinerary reconstruction—was completed successfully.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 10	FOOD DELIVERY ROUTE OPTIMIZATION USING DIJKSTRA'S ALGORITHM - GRAPH OPERATIONS
Date:	

AIM:

To design a food delivery application for a bustling city using Dijkstra's algorithm to ensure the shortest and fastest delivery routes for thousands of daily orders.

ALGORITHM:

1. Represent the city map as a graph with delivery zones as nodes and roads as weighted edges.
2. Initialize distances from the source (restaurant) to all other nodes as INFINITY, except for the source itself (0).
3. Use a priority queue to select the node with the minimum distance.
4. For the current node, update the distances to its adjacent nodes if a shorter path is found.
5. Repeat the process until all nodes are visited.
6. The final distance values represent the shortest delivery times from the source to each location.

PROGRAM:

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

#define INF 1000000000
#define MAX 100

typedef struct {
    int vertex;
    int weight;
} Edge;

typedef struct {
    Edge edges[MAX];
    int size;
} AdjList;

typedef struct {
    int vertex;
    int dist;
} Node;

typedef struct {
    Node heap[MAX];
    int size;
} MinHeap;

AdjList graph[MAX];
int dist[MAX];

// Min Heap utility functions
void swap(Node* a, Node* b) {
    Node temp = *a;
    *a = *b;
    *b = temp;
```

```
}

void heapifyUp(MinHeap* h, int i) {
    while (i != 0 && h->heap[(i - 1) / 2].dist > h->heap[i].dist) {
        swap(&h->heap[i], &h->heap[(i - 1) / 2]);
        i = (i - 1) / 2;
    }
}

void heapifyDown(MinHeap* h, int i) {
    int smallest = i;
    int left = 2*i + 1;
    int right = 2*i + 2;

    if (left < h->size && h->heap[left].dist < h->heap[smallest].dist)
        smallest = left;
    if (right < h->size && h->heap[right].dist < h->heap[smallest].dist)
        smallest = right;

    if (smallest != i) {
        swap(&h->heap[i], &h->heap[smallest]);
        heapifyDown(h, smallest);
    }
}

void insertHeap(MinHeap* h, int vertex, int dist) {
    h->heap[h->size].vertex = vertex;
    h->heap[h->size].dist = dist;
    heapifyUp(h, h->size);
    h->size++;
}

Node extractMin(MinHeap* h) {
    Node root = h->heap[0];
    h->heap[0] = h->heap[h->size - 1];
    h->size--;
    heapifyDown(h, 0);
    return root;
}

int isEmpty(MinHeap* h) {
    return h->size == 0;
}

// Dijkstra's algorithm
void dijkstra(int src, int n) {
    for (int i = 0; i < n; i++)
        dist[i] = INF;

    dist[src] = 0;
    MinHeap h;
    h.size = 0;
    insertHeap(&h, src, 0);
```

```
    while (!isEmpty(&h)) {
        Node node = extractMin(&h);
        int u = node.vertex;

        for (int i = 0; i < graph[u].size; i++) {
            int v = graph[u].edges[i].vertex;
            int weight = graph[u].edges[i].weight;

            if (dist[v] > dist[u] + weight) {
                dist[v] = dist[u] + weight;
                insertHeap(&h, v, dist[v]);
            }
        }
    }
}

int main() {
    int n, e;
    printf("Enter number of delivery locations and roads: ");
    scanf("%d %d", &n, &e);

    // Initialize adjacency lists
    for (int i = 0; i < n; i++)
        graph[i].size = 0;

    printf("Enter roads (u v w):\n");
    for (int i = 0; i < e; i++) {
        int u, v, w;
        scanf("%d %d %d", &u, &v, &w);
        graph[u].edges[graph[u].size++] = (Edge){v, w};
        graph[v].edges[graph[v].size++] = (Edge){u, w};  // If undirected
    }

    int src;
    printf("Enter the source (restaurant location): ");
    scanf("%d", &src);

    dijkstra(src, n);

    printf("Shortest delivery times from source:\n");
    for (int i = 0; i < n; i++) {
        printf("To %d : %d mins\n", i, dist[i]);
    }

    return 0;
}
```

OUTPUT:

```
Enter number of delivery locations and roads: 5 6
Enter roads (u v w):
0 1 10
0 2 3
1 2 1
1 3 2
2 3 8
2 4 2
Enter the source (restaurant location): 0
Shortest delivery times from source:
To 0 : 0 mins
To 1 : 4 mins
To 2 : 3 mins
To 3 : 6 mins
To 4 : 5 mins
```

TEST CASES:

1) **Input: Basic City Graph with 4 Locations**
   Number of delivery locations: 4
   Roads (u v w):
   0 1 5
   1 2 10
   1 2 3
   1 3 9
   Source (restaurant): 0
   Expected Output:
   To 0 : 0 mins
   To 1 : 5 mins
   To 2 : 8 mins
   To 3 : 14 mins

2) **Input: Include a Shorter Alternate Path**
   Number of delivery locations: 5
   Roads (u v w):
   0 1 4
   0 2 1
   2 1 2
   1 3 5
   2 3 8
   3 4 3
   Source (restaurant): 0
   Expected Output:
   To 0 : 0 mins
   To 1 : 3 mins
   To 2 : 1 min
   To 3 : 8 mins
   To 4 : 11 mins

3) **Input: Disconnected Node**
Number of delivery locations: 4
Roads (u v w):
0 1 2
1 2 4
// Node 3 has no connection
Source (restaurant): 0
Expected Output:
To 0 : 0 mins
To 1 : 2 mins
To 2 : 6 mins
To 3 : 1000000000 mins (INF)

RESULT:

Thus the C program to design a food delivery application for a bustling city using Dijkstra's algorithm is executed successfully.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 11 a	OPTIMIZED NETWORK INFRASTRUCTURE: MINIMUM SPANNING TREE (MST) USING PRIMS'S ALGORITHM
Date:	

AIM:

To design an optimized network infrastructure connecting various departments within an organization using the minimum amount of cabling to reduce costs and enhance network efficiency by implementing Prim's algorithms for finding the Minimum Spanning Tree (MST).

ALGORITHM:

1. Initialize key values for all vertices as infinite and mstSet[] as false.
2. Set key value of the first vertex to 0 so that it is picked first.
3. Repeat until MST contains all vertices:
   a. Pick the vertex u not in mstSet[] with minimum key value.
   b. Include u in mstSet[].
   c. For each adjacent vertex v of u, if weight(u, v) < key[v] and v not in mstSet[], update key[v] = weight(u, v) and set parent[v] = u.

PROGRAM:

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

#define V 5  // Number of vertices

int minKey(int key[], int mstSet[]) {
    int min = INT_MAX, min_index = -1;
    for (int v = 0; v < V; v++)
        if (!mstSet[v] && key[v] < min)
            min = key[v], min_index = v;
    return min_index;
}

void printMST(int parent[], int graph[V][V]) {
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);
}

void primMST(int graph[V][V]) {
    int parent[V];
    int key[V];
    int mstSet[V];

    for (int i = 0; i < V; i++) {
        key[i] = INT_MAX;
        mstSet[i] = 0;
```

```
    }

    key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet);
        mstSet[u] = 1;

        for (int v = 0; v < V; v++) {
            if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }

    printMST(parent, graph);
}

int main() {
    int graph[V][V] = {
        {0, 2, 0, 6, 0},
        {2, 0, 3, 8, 5},
        {0, 3, 0, 0, 7},
        {6, 8, 0, 0, 9},
        {0, 5, 7, 9, 0},
    };

    printf("Minimum Spanning Tree using Prim's Algorithm:\n");
    primMST(graph);

    return 0;
}
```

OUTPUT:

```
Minimum Spanning Tree using Prim's Algorithm:
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```

TEST CASES:

1) V=5, Graph as in program
2) Check connectivity and output correctness matches manual MST calculation

RESULT:

The MST for the network infrastructure was computed successfully using Prim's algorithm, ensuring minimal cabling cost and efficient connectivity.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 11b	OPTIMIZED NETWORK INFRASTRUCTURE: MINIMUM SPANNING TREE (MST) USING KRUSKAL'S ALGORITHM
Date:	

AIM:

To design an optimized network infrastructure connecting various departments within an organization using the minimum amount of cabling to reduce costs and enhance network efficiency by implementing Kruskal's algorithm for finding the Minimum Spanning Tree (MST).

ALGORITHM:

1. Create a list of all edges and sort them in non-decreasing order of their weight.
2. Initialize MST as empty. Initialize parent[] and rank[] for union-find.
3. For each edge (u, v) in sorted order:
   a. Find sets (roots) of u and v using find().
   b. If roots are different, include this edge in MST and union the sets.
4. Stop when MST has (V-1) edges.

PROGRAM:

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

#define MAX_EDGES 100

struct Edge {
   int src, dest, weight;
};

int compare(const void* a, const void* b) {
   struct Edge* e1 = (struct Edge*)a;
   struct Edge* e2 = (struct Edge*)b;
   return e1->weight - e2->weight;
}

int find(int parent[], int i) {
   if (parent[i] != i)
      parent[i] = find(parent, parent[i]);
   return parent[i];
}

void unionSet(int parent[], int rank[], int x, int y) {
   int xroot = find(parent, x);
   int yroot = find(parent, y);

   if (rank[xroot] < rank[yroot])
      parent[xroot] = yroot;
   else if (rank[xroot] > rank[yroot])
      parent[yroot] = xroot;
```

```
    else {
        parent[yroot] = xroot;
        rank[xroot]++;
    }
}

void kruskalMST(struct Edge edges[], int E, int V) {
    qsort(edges, E, sizeof(struct Edge), compare);

    struct Edge result[MAX_EDGES];
    int parent[V], rank[V];
    int i, e = 0;

    for (i = 0; i < V; i++) {
        parent[i] = i;
        rank[i] = 0;
    }

    for (i = 0; i < E && e < V - 1; i++) {
        struct Edge next = edges[i];
        int x = find(parent, next.src);
        int y = find(parent, next.dest);

        if (x != y) {
            result[e++] = next;
            unionSet(parent, rank, x, y);
        }
    }

    printf("Edge \tWeight\n");
    for (i = 0; i < e; i++) {
        printf("%d - %d \t%d\n", result[i].src, result[i].dest, result[i].weight);
    }
}

int main() {
    int V = 5;
    struct Edge edges[] = {
        {0, 1, 2}, {0, 3, 6}, {1, 2, 3}, {1, 3, 8}, {1, 4, 5}, {2, 4, 7}, {3, 4, 9}
    };
    int E = sizeof(edges) / sizeof(edges[0]);

    printf("Minimum Spanning Tree using Kruskal's Algorithm:\n");
    kruskalMST(edges, E, V);

    return 0;
```

}

OUTPUT:

Minimum Spanning Tree using Kruskal's Algorithm:
Edge Weight
0 - 1 2
1 - 2 3
0 - 3 6
1 - 4 5

TEST CASES:

1) V=5, Graph as in program
2) Check connectivity and output correctness matches manual MST calculation

RESULT:

The MST for the network infrastructure was computed successfully using Kruskal's algorithm, ensuring minimal cabling cost and efficient connectivity.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 12	STUDENT GRADE MANAGEMENT SYSTEM
Date:	

AIM:

To organize and manage student grades across different courses and efficiently display them using suitable sorting algorithms for different operations.

ALGORITHM:

1. Define a structure to store student name, ID, course, and marks.
2. Insert student records dynamically.
3. Use:
   - Insertion Sort for quick overview
   - Bubble Sort during entry
   - Quick Sort for course-wise ranking
   - Merge Sort for full report
4. Display sorted results as needed.

PROGRAM:

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

#define MAX 100

typedef struct {
    char name[50], course[50];
    int id;
    float grade;
} Student;

// Display students
void display(Student s[], int n) {
    for (int i = 0; i < n; i++) {
        printf("ID: %d | Name: %s | Course: %s | Grade: %.2f\n",
               s[i].id, s[i].name, s[i].course, s[i].grade);
    }
    printf("-----------------------------\n");
}

// Bubble Sort by ID
void bubbleSort(Student s[], int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (s[j].id > s[j + 1].id) {
                Student temp = s[j];
```

```
            s[j] = s[j + 1];
            s[j + 1] = temp;
        }
    }
}

// Insertion Sort by Grade (Ascending)
void insertionSort(Student s[], int n) {
    for (int i = 1; i < n; i++) {
        Student key = s[i];
        int j = i - 1;
        while (j >= 0 && s[j].grade > key.grade) {
            s[j + 1] = s[j];
            j--;
        }
        s[j + 1] = key;
    }
}

// Quick Sort by Grade (Descending)
int partition(Student s[], int low, int high) {
    float pivot = s[high].grade;
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (s[j].grade > pivot) {
            i++;
            Student temp = s[i];
            s[i] = s[j];
            s[j] = temp;
        }
    }
    Student temp = s[i + 1];
    s[i + 1] = s[high];
    s[high] = temp;
    return i + 1;
}

void quickSort(Student s[], int low, int high) {
    if (low < high) {
        int pi = partition(s, low, high);
        quickSort(s, low, pi - 1);
        quickSort(s, pi + 1, high);
    }
}

// Merge Sort by Grade (Ascending)
void merge(Student s[], int l, int m, int r) {
```

```
    int n1 = m - l + 1, n2 = r - m;
    Student L[MAX], R[MAX];

    for (int i = 0; i < n1; i++)
        L[i] = s[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = s[m + 1 + j];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i].grade <= R[j].grade)
            s[k++] = L[i++];
        else
            s[k++] = R[j++];
    }
    while (i < n1) s[k++] = L[i++];
    while (j < n2) s[k++] = R[j++];
}

void mergeSort(Student s[], int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        mergeSort(s, l, m);
        mergeSort(s, m + 1, r);
        merge(s, l, m, r);
    }
}

int main() {
    Student students[] = {
        {"Alice", "Math", 101, 87.5},
        {"Bob", "Math", 102, 92.0},
        {"Charlie", "Math", 103, 78.5},
        {"David", "Math", 104, 85.0}
    };
    int n = 4;

    printf("Original:\n");
    display(students, n);

    Student overview[MAX], markEntry[MAX], ranking[MAX], report[MAX];
    memcpy(overview, students, sizeof(students));
    memcpy(markEntry, students, sizeof(students));
    memcpy(ranking, students, sizeof(students));
    memcpy(report, students, sizeof(students));
```

```
    insertionSort(overview, n);
    printf("Quick Overview (Insertion Sort by Grade Ascending):\n");
    display(overview, n);

    bubbleSort(markEntry, n);
    printf("Mark Entry (Bubble Sort by ID):\n");
    display(markEntry, n);

    quickSort(ranking, 0, n - 1);
    printf("Course Ranking (Quick Sort by Grade Descending):\n");
    display(ranking, n);
    mergeSort(report, 0, n - 1);
    printf("Overall Grade Report (Merge Sort by Grade Ascending):\n");
    display(report, n);
    return 0;
}
```

OUTPUT:

```
Original:
ID: 101 | Name: Alice | Course: Math | Grade: 87.50
ID: 102 | Name: Bob | Course: Math | Grade: 92.00
ID: 103 | Name: Charlie | Course: Math | Grade: 78.50
ID: 104 | Name: David | Course: Math | Grade: 85.00
-----------------------------
Quick Overview (Insertion Sort by Grade Ascending):
ID: 103 | Name: Charlie | Course: Math | Grade: 78.50
ID: 104 | Name: David | Course: Math | Grade: 85.00
ID: 101 | Name: Alice | Course: Math | Grade: 87.50
ID: 102 | Name: Bob | Course: Math | Grade: 92.00
-----------------------------
Mark Entry (Bubble Sort by ID):
ID: 101 | Name: Alice | Course: Math | Grade: 87.50
ID: 102 | Name: Bob | Course: Math | Grade: 92.00
ID: 103 | Name: Charlie | Course: Math | Grade: 78.50
ID: 104 | Name: David | Course: Math | Grade: 85.00
-----------------------------
Course Ranking (Quick Sort by Grade Descending):
ID: 102 | Name: Bob | Course: Math | Grade: 92.00
ID: 101 | Name: Alice | Course: Math | Grade: 87.50
ID: 104 | Name: David | Course: Math | Grade: 85.00
ID: 103 | Name: Charlie | Course: Math | Grade: 78.50
-----------------------------
Overall Grade Report (Merge Sort by Grade Ascending):
ID: 103 | Name: Charlie | Course: Math | Grade: 78.50
ID: 104 | Name: David | Course: Math | Grade: 85.00
ID: 101 | Name: Alice | Course: Math | Grade: 87.50
ID: 102 | Name: Bob | Course: Math | Grade: 92.00
-----------------------------
```

TEST CASES:

Input:

Alice - 101 - Math - 87.5

Bob - 102 - Math - 92.0

Charlie - 103 - Math - 78.5

David - 104 - Math - 85.0

Expected Output:

- **Quick Overview (Insertion Sort):**
  Charlie < David < Alice < Bob
- **Mark Entry (Bubble Sort by ID):**
  101 < 102 < 103 < 104
- **Course Ranking (Quick Sort):**
  Bob > Alice > David > Charlie
- **Overall Report (Merge Sort):**
  Charlie < David < Alice < Bob

RESULT:

Thus, the C program for a Student Grade Management System using different sorting algorithms was implemented successfully.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 13 a	PRODUCT INVENTORY MANAGEMENT SYSTEM - SEARCH OPERATIONS
Date:	

AIM:

To efficiently search and retrieve product details in a retail inventory using Linear Search.

ALGORITHM:

1. Scan the inventory list sequentially.
2. Compare each product ID with the search query.
3. Return the product details if found, otherwise indicate not found.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
// Structure to store product details
struct Product {
    int productID;
    char productName[50];
    float price;
};

// Linear Search Function
void linearSearch(struct Product inventory[], int n, int key) {
    for(int i = 0; i < n; i++) {
        if(inventory[i].productID == key) {
            printf("\nProduct Found: ID = %d, Name = %s, Price = %.2f\n",
                    inventory[i].productID, inventory[i].productName, inventory[i].price);
            return;
        }
    }
    printf("\nProduct ID %d not found!\n", key);
}

int main() {
    int n, key;

    // Input the number of products
    printf("\nEnter the number of products: ");
    scanf("%d", &n);
```

```
    struct Product inventory[n];

    // Enter product details
    printf("\nEnter product details (ID Name Price):\n");
    for(int i = 0; i < n; i++) {
        scanf("%d %s %f", &inventory[i].productID, inventory[i].productName, &inventory[i].price);
    }

    // Search product
    printf("\nEnter product ID to search using Linear Search: ");
    scanf("%d", &key);

    linearSearch(inventory, n, key);
    return 0;
}
```

OUTPUT:

```
Enter the number of products: 3

Enter product details (ID Name Price):
101 Pencil 5.00
102 Eraser 2.00
103 Scale 15.00

Enter product ID to search using Linear Search: 102

Product Found: ID = 102, Name = Eraser, Price = 2.00
```

TEST CASES:

1. **Input:** 7
   101 Notebook 80.0
   102 Pen 15.0
   103 Pencil 10.0
   104 Eraser 5.0
   105 Sharpener 20.0
   106 Marker 50.0
   107 Stapler 100.0
   Excepted Output: Product Found: ID = 107, Name = Stapler, Price = 100.00
2. **Input:** 4
   111 Fridge 30000.0
   222 WashingMachine 20000.0
   333 Microwave 15000.0
   444 AirConditioner 40000.0
   999
   Expected output: Product ID 999 not found!

RESULT:

Thus, the C program for a Product Inventory Management System using Linear Search was successfully implemented.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 13b	PRODUCT INVENTORY MANAGEMENT SYSTEM - SEARCH OPERATIONS
Date:	

AIM:

To efficiently search and retrieve product details in a retail inventory using Binary Search.

ALGORITHM:

1. Ensure the inventory is sorted.
2. Divide the inventory into halves.
3. Compare the middle element with the search query.
4. If found, return the product details; otherwise, continue searching in the relevant half of the list.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
// Structure to store product details
struct Product {
   int productID;
   char productName[50];
   float price;
};
// Binary Search Function (Requires Sorted Inventory)
int binarySearch(struct Product inventory[], int left, int right, int key) {
   while(left <= right) {
      int mid = left + (right - left) / 2;
      if(inventory[mid].productID == key) {
         printf("\nProduct Found: ID = %d, Name = %s, Price = %.2f\n",
               inventory[mid].productID, inventory[mid].productName, inventory[mid].price);
         return mid;
      }
      else if(inventory[mid].productID < key)
         left = mid + 1;
      else
         right = mid - 1;
   }
   printf("\nProduct ID %d not found!\n", key);
   return -1;
}
// Helper function to sort inventory for Binary Search (Bubble Sort)
void bubbleSort(struct Product inventory[], int n) {
   struct Product temp;
```

```
    for(int i = 0; i < n - 1; i++) {
        for(int j = 0; j < n - i - 1; j++) {
            if(inventory[j].productID > inventory[j + 1].productID) {
                temp = inventory[j];
                inventory[j] = inventory[j + 1];
                inventory[j + 1] = temp;
            }
        }
    }
}

int main() {
    int n, key;

    // Input the number of products
    printf("\nEnter the number of products: ");
    scanf("%d", &n);
    struct Product inventory[n];
        // Enter product details
    printf("\nEnter product details (ID Name Price):\n");
    for(int i = 0; i < n; i++) {
        scanf("%d %s %f", &inventory[i].productID, inventory[i].productName, &inventory[i].price);
    }

    // Sort inventory for Binary Search
    bubbleSort(inventory, n);

    // Search product
    printf("\nEnter product ID to search using Binary Search: ");
    scanf("%d", &key);
    binarySearch(inventory, 0, n - 1, key);
    return 0;
}
```

OUTPUT:

```
Enter the number of products: 5

Enter product details (ID Name Price):
123 Pencil 2.00
203 Eraser 5.00
516 Sharpner 3.00
798 Scale 20.00
345 Notbook 45.00

Enter product ID to search using Binary Search: 798

Product Found: ID = 798, Name = Scale, Price = 20.00
```

TEST CASES:

1. **Input:** 5
   101 Laptop 50000
   202 Smartphone 25000
   303 Tablet 30000
   404 Monitor 40000
   505 Printer 15000
   Excepted Output: Product Found: ID = 303, Name = Tablet, Price = 30000.00
2. **Input:** 4
   111 Fridge 30000.0
   222 WashingMachine 20000.0
   333 Microwave 15000.0
   444 AirConditioner 40000.0
   Expected output: Product ID 999 not found!

RESULT:

Thus, the C program for a Product Inventory Management System using Binary Search was successfully implemented.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		

Exp No. 14	DEMONSTRATION OF APPLICATIONS OF HASHING
Date:	

AIM:

This program demonstrates key applications of hashing and sparse tables through four problems: checking if an array can be sorted with a single swap, verifying anagrams using character frequency hashing, efficiently answering range minimum queries using a sparse table, and merging two sorted arrays. These tasks highlight efficient algorithmic techniques for solving common problems in C.

ALGORITHM:

Single Swap Sorted Array:

- Sort the array and compare it with the original array.
- Track indices where elements mismatch.
- If exactly two mismatched indices, check if swapping them sorts the array.

Anagram Checking

- Compare string lengths; if unequal, return false.
- Use a hash map to count character frequencies in both strings.
- If all counts match, return true; otherwise, return false.

Range Minimum Query Using Sparse Table:

- Preprocess the array to build the sparse table for range minimum queries.
- For each query, use the sparse table to efficiently find the minimum in the specified range.

Merge Two Sorted Arrays:

- Use two pointers to traverse both sorted arrays.
- Compare elements and add the smaller one to the result array.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>

// ========== 1. Single Swap Sorted Array ==========
bool canBeSortedWithOneSwap(int arr[], int n) {
    int mismatch[2], count = 0;
    int sorted[n];

    for (int i = 0; i < n; i++)
        sorted[i] = arr[i];

    // Bubble sort
    for (int i = 0; i < n - 1; i++)
```

```
        for (int j = i + 1; j < n; j++)
            if (sorted[i] > sorted[j]) {
                int temp = sorted[i];
                sorted[i] = sorted[j];
                sorted[j] = temp;
            }

    for (int i = 0; i < n; i++) {
        if (arr[i] != sorted[i]) {
            if (count >= 2) return false;
            mismatch[count++] = i;
        }
    }

    if (count == 0) return true;
    if (count == 2 && arr[mismatch[0]] == sorted[mismatch[1]] &&
                    arr[mismatch[1]] == sorted[mismatch[0]])
        return true;

    return false;
}

// ========== 2. Anagram Checking ==========
bool areAnagrams(char str1[], char str2[]) {
    int hash[256] = {0};
    if (strlen(str1) != strlen(str2)) return false;

    for (int i = 0; str1[i]; i++) {
        hash[(int)str1[i]]++;
        hash[(int)str2[i]]--;
    }

    for (int i = 0; i < 256; i++)
        if (hash[i] != 0)
            return false;

    return true;
}

// ========== 3. Range Minimum Query Using Sparse Table ==========
#define MAXN 100
#define LOG 10

int st[MAXN][LOG];
int logTable[MAXN];
```

```
void preprocess(int arr[], int n) {
    logTable[1] = 0;
    for (int i = 2; i <= n; i++)
        logTable[i] = logTable[i / 2] + 1;

    for (int i = 0; i < n; i++)
        st[i][0] = arr[i];

    for (int j = 1; (1 << j) <= n; j++) {
        for (int i = 0; i + (1 << j) <= n; i++) {
            st[i][j] = (st[i][j - 1] < st[i + (1 << (j - 1))][j - 1])
                    ? st[i][j - 1]
                    : st[i + (1 << (j - 1))][j - 1];
        }
    }
}

int query(int L, int R) {
    int j = logTable[R - L + 1];
    return (st[L][j] < st[R - (1 << j) + 1][j])
         ? st[L][j]
         : st[R - (1 << j) + 1][j];
}

// ========== 4. Merge Two Sorted Arrays ==========
void mergeArrays(int A[], int B[], int n, int m) {
    int i = 0, j = 0, k = 0;
    int result[n + m];

    while (i < n && j < m) {
        if (A[i] < B[j])
            result[k++] = A[i++];
        else
            result[k++] = B[j++];
    }
    while (i < n) result[k++] = A[i++];
    while (j < m) result[k++] = B[j++];

    printf("Merged array: ");
    for (int l = 0; l < n + m; l++)
        printf("%d ", result[l]);
    printf("\n");
}

// ========== Main Driver ==========
int main() {
```

```
    printf("---- 1. Single Swap Sort Check ----\n");
    int arr1[] = {1, 3, 5, 4, 2};
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
    if (canBeSortedWithOneSwap(arr1, n1))
        printf("Yes, array can be sorted with one swap\n");
    else
        printf("No, more than one swap needed\n");

    printf("\n---- 2. Anagram Check ----\n");
    char str1[] = "triangle", str2[] = "integral";
    if (areAnagrams(str1, str2))
        printf("Yes, the strings are anagrams\n");
    else
        printf("No, not anagrams\n");

    printf("\n---- 3. Range Minimum Query (Sparse Table) ----\n");
    int arr2[] = {1, 3, -1, 7, 9, 11, 3, 5};
    int n2 = sizeof(arr2) / sizeof(arr2[0]);
    preprocess(arr2, n2);
    printf("Minimum in range [1, 4]: %d\n", query(1, 4));
    printf("Minimum in range [3, 6]: %d\n", query(3, 6));

    printf("\n---- 4. Merge Two Sorted Arrays ----\n");
    int A[] = {2, 4, 6, 8};
    int B[] = {1, 3, 5, 7};
    mergeArrays(A, B, 4, 4);
    return 0;
}
```

OUTPUT:

```
---- 1. Single Swap Sort Check ----
No, more than one swap needed

---- 2. Anagram Check ----
Yes, the strings are anagrams

---- 3. Range Minimum Query (Sparse Table) ----
Minimum in range [1, 4]: -1
Minimum in range [3, 6]: 3

---- 4. Merge Two Sorted Arrays ----
Merged array: 1 2 3 4 5 6 7 8


...Program finished with exit code 0
Press ENTER to exit console.
```

TEST CASES:

1. Single Swap Sorted Array:

- **Input:** [1, 3, 2, 4, 5]
  Expected Output: Yes (Swapping 3 and 2 sorts the array)
- **Input:** [1, 5, 3, 4, 2]
  Expected Output: No (More than one swap needed)

2. Anagram Checking:

- **Input:** "listen", "silent"
  Expected Output: Yes (They are anagrams)
- **Input:** "hello", "world"
  Expected Output: No (Different characters)

3.Range Minimum Query Using Sparse Table:

- **Input Array:** [2, 5, 1, 4, 9, 3]
  Query (L=1, R=4):
  Expected Output: 1 (Minimum in the range [1,4])
- **Query (L=2, R=5):**
  Expected Output: 1

4. Merge Two Sorted Arrays:

- **Input:** A = [1, 3, 5], B = [2, 3, 6]
  Expected Output: [1, 2, 3, 5, 6] (Merged with duplicates removed)
- **Input:** A = [4, 6], B = [1, 2]
  Expected Output: [1, 2, 4, 6]

RESULT:

Thus, the C program demonstrating the applications of hashing—covering Single Swap Sorted Array, Anagram Checking, Range Minimum Query using Sparse Table, and Merging Two Sorted Arrays—was implemented and executed successfully.

EVALUATION:

PARAMETES	MAX MARKS	MARKS OBTAINED
Pre Lab	20	
Design	10	
Coding (with Standards)	20	
Testing	15	
Viva	10	
Total	75	
Signature of the Faculty		