

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

On

DATA STRUCTURES (23CS3PCDST)

Submitted by

TANMAYI S BALIJA (1BM22CS359)

**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 **TANMAYI S BALIJA (1BM22CS359)**, who is a 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

Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak

Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Working of stack using an array	4-5
2	WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression	6-8
3	WAP to simulate the working of a queue of integers using an array. WAP to simulate the working of a circular queue of integers using an array	8-13
4	WAP to Implement Singly Linked List (Insertion)	13-16
5	WAP to Implement Singly Linked List (Deletion)	16-18
6	WAP to Implement Single Link List (Sorting, Reversing and Concatenation). WAP to implement Stack & Queues using Linked Representation .	18-24
7	WAP to Implement doubly link list with primitive operations +leetcode	24-26 26-28
8	Constructing Binary Search Tree(BST)	28-34
9	BFS and DFS + leetcode	35-41
10	Hash Function	42-43

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 n 5
int top=-1; stack[n];

void push(int a)
{
    if(top==n-1)
    {
        printf("Stack is full");
    }
    else{
        top++;
        stack[top] = a;
    }
}

int pop()
{
    int a;
    if(top==--1)
    {
        printf("underflow");
    }
    else{
        a=stack[top];
        printf("%d is popped",a);
        top--;
    }
}

void display()
{
    if(top==--1)
```

```

void display()
{
    if(top== -1)
    {
        printf("no elements");
    }
    else{
        while(top!= -1)
        {
            printf("%d \n", stack[top]);
            top--;
        }
    }
}

void main()
{
    int a, choice;
    printf("Enter 1 Push, 2 Pop, 3 Display 4 Exit \n");
    while(1)
    {
        printf("Enter choice \t");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: printf("Enter number to be inserted \t");
                    scanf("%d", &a);
                    push(a);
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
            case 4: exit(0);
        }
    }
}

```

Output:

```

Enter 1 Push, 2 Pop, 3 Display 4 Exit
Enter choice    1

Enter number to be inserted    20
Enter choice    1

Enter number to be inserted    50
Enter choice    1

Enter number to be inserted    80
Enter choice    2
80 is poppedEnter choice    1

Enter number to be inserted    67
Enter choice    3
Stack elements:
67
50
20

```

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 <ctype.h>

#define MAX_SIZE 100

int precedence(char symbol);
void push(char item);
char pop();
void infixToPostfix(char infix[], char postfix[]);

char stack[MAX_SIZE];
int top = -1;

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

    printf("Enter infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

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

    return 0;
}

int precedence(char symbol) {
    switch(symbol) {
        case '^':
            return 3;
    }
}

int precedence(char symbol) {
    switch(symbol) {
        case '^':
            return 3;
        case '*':
        case '/':
            return 2;
        case '+':
        case '-':
            return 1;
        default:
            return 0;
    }
}

void push(char item) {
    if (top == MAX_SIZE - 1) {
        printf("Stack Overflow\n");
        exit(EXIT_FAILURE);
    }
    stack[++top] = item;
}

char pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
        exit(EXIT_FAILURE);
    }
    return stack[top--];
}
```

```

void infixToPostfix(char infix[], char postfix[]) {
    int i = 0, j = 0;
    char symbol, x;

    push('(');

    while ((symbol = infix[i++]) != '\0') {
        if (symbol == '(') {
            push(symbol);
        } else if (isalnum(symbol)) {
            postfix[j++] = symbol;
        } else if (symbol == ')') {
            while (stack[top] != '(') {
                postfix[j++] = pop();
            }
            x = pop();
        } else {
            while (precedence(stack[top]) >= precedence(symbol)) {
                postfix[j++] = pop();
            }
            push(symbol);
        }
    }

    while (stack[top] != '(') {
        postfix[j++] = pop();
    }
    x = pop();
    postfix[j] = '\0';
}

```

Output:

```

Enter the expression : 4+(3//7)-(6*8)+2/8-4
4 3 / 7 / + 6 8 * - 2 8 / + 4 -
-----
Process exited after 12.95 seconds with return value 0
Press any key to continue . . . |

```

Lab program 3a:

WAP to simulate the working of a queue of integers using an array. Provide the following operations

- a) Insert
- b) Delete
- c) Display

The program should print appropriate messages for queue empty and queue overflow conditions.

```
#include <stdio.h>
#define size 5
int queue[size], front=-1, rear=-1;
void enqueue(int a){
    if(rear==size-1){
        printf("Queue is full/overflow\n");
        return;
    }
    else if(front==--1 && rear==--1){
        front=0;
        rear=0;
    }
    else{
        rear=rear+1;
    }
    queue[rear]=a;
}
int dequeue(){
    int a;
    if((front==--1 && rear==--1)||front>rear){
        printf("Queue is empty/underflow\n");
    }
    else{
        a=queue[front];
        front++;
    }
    return a;
}
void display(){
    if((front==--1 && rear==--1)||front>rear){
        printf("Queue is empty/underflow\n");
    }
    else{
        for(int i=front; i<=rear; i++){
            printf("%d\t", queue[i]);
        }
    }
}
```



```

        front++;
    }
    return a;
}

void display() {
    if((front==--1 && rear==--1)||front>rear){
        printf("Queue is empty/underflow\n");
    }
    else{
        for(int i=front;i<=rear;i++){
            printf("%d\t",queue[i]);
        }
    }
}

void main() {
    int op,n;
    while(1){
        printf("\nEnter 1.Enqueue\n2.Dequeue\n3.Display\n4.-1 to stop execution\n");
        scanf("%d",&op);
        if(op==--1){
            break;
        }
        switch(op){
            case 1:printf("Enter no\n");
                    scanf("%d",&n);
                    enqueue(n);
                    break;
            case 2:n=dequeue();
                    printf("%d is Dequeued\n",n);
                    break;
            case 3:display();
                    break;
            default:printf("Invalid choice\n");
        }
    }
}

```

Output:

```
Vismays-MacBook-Pro:LN code notes  
/vismaypawar/Desktop/All notes/dsa
```

1. Enqueue
2. Dequeue
3. Display
4. Exit

```
Enter your choice: 1  
Enter element to enqueue: 2  
Inserted 2 into the queue.
```

1. Enqueue
2. Dequeue
3. Display
4. Exit

```
Enter your choice: 1  
Enter element to enqueue: 3  
Inserted 3 into the queue.
```

1. Enqueue
2. Dequeue
3. Display
4. Exit

```
Enter your choice: 1  
Enter element to enqueue: 4  
Inserted 4 into the queue.
```

1. Enqueue
2. Dequeue
3. Display
4. Exit

```
Enter your choice: 1  
Enter element to enqueue: 5  
Inserted 5 into the queue.
```

1. Enqueue
2. Dequeue
3. Display
4. Exit

```
Enter your choice: 1  
Enter element to enqueue: 6  
Inserted 6 into the queue.
```

1. Enqueue
2. Dequeue
3. Display
4. Exit

```
Enter your choice: 1  
Queue Overflow. Cannot enqueue.
```

```
Vismays-MacBook-Pro:LN code notes  
/vismaypawar/Desktop/All notes/dsa
```

1. Enqueue
2. Dequeue
3. Display
4. Exit

```
Enter your choice: 1  
Enter element to enqueue: 2  
Inserted 2 into the queue.
```

1. Enqueue
2. Dequeue
3. Display
4. Exit

```
Enter your choice: 1  
Enter element to enqueue: 3  
Inserted 3 into the queue.
```

1. Enqueue
2. Dequeue
3. Display
4. Exit

```
Enter your choice: 1  
Enter element to enqueue: 4  
Inserted 4 into the queue.
```

1. Enqueue
2. Dequeue
3. Display
4. Exit

```
Enter your choice: 1  
Enter element to enqueue: 5  
Inserted 5 into the queue.
```

1. Enqueue
2. Dequeue
3. Display
4. Exit

```
Enter your choice: 1  
Enter element to enqueue: 6  
Inserted 6 into the queue.
```

1. Enqueue
2. Dequeue
3. Display
4. Exit

```
Enter your choice: 1  
Queue Overflow. Cannot enqueue.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 2 3 4 5 6
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 2 from the queue.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 3 from the queue.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 4 from the queue.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 5 from the queue.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 6 from the queue.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue Underflow! Cannot delete element.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue is empty.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 2 3 4 5 6
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 2 from the queue.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 3 from the queue.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 4 from the queue.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 5 from the queue.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Deleted 6 from the queue.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue Underflow! Cannot delete element.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue is empty.
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting...
```

Lab program 3b:

WAP to simulate the working of a circular queue of integers using an array. Provide the following operations.

a) Insert

b) Delete

c) Display

The program should print appropriate messages for queue empty and queue overflow conditions.

```
}
else if (front==rear)
{
    front=rear=-1;
}
else{
    printf("%d is popped \n",queue[front]);
    front=(front+1)%n;
}
}

}

void display()
{
    int i=front;
    if (front==rear)
    {
        printf("no elements \n");
    }
    else{
        while(i!=rear)
        {
            printf("%d \n",queue[i]);
            i=(i+1)%n;
        }
        break;
    }
}

void main()
{
    int a,choice;
    printf("Enter 1 enqueue, 2 dequeue, 3 Display 4 Exit \n");
    while(1)
    {
        printf("Enter choice \n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: printf("Enter number to be inserted \n");
                    scanf("%d",&a);
                    enqueue(a);
                    break;
            case 2: dequeue();
                    break;
            case 3: display();
                    break;
            case 4: exit(0);
        }
    }
}
```

```

#include<stdio.h>
#include<stdlib.h>
#define n 3
int front=-1, rear=-1, queue[n];

void enqueue(int a)
{
    if(rear==-1 && front== -1)
    {
        rear=front=0;
        queue[rear]=a;
    }
    else if((rear+1)%n==front)
    {
        printf("Queue is full \n");
    }
    else{
        rear=(rear+1)%n;
        queue[rear] = a;
    }
}

void dequeue()
{
    int a;
    if(front==-1 && rear == -1)
    {
        printf("underflow \n");
    }
    else if(front==rear)
    {
        front=rear=-1;
    }
    else{
        printf("%d is popped \n",queue[front]);
        front=(front+1)%n;
    }
}

void display()
{
    int i=front;
    if(front==-1)
    {
        printf("no elements \n");
    }
    else{
        while(i!=rear)
        {
            printf("%d \n",queue[i]);
            i=(i+1)%n;
        }
        break;
    }
}

```

Output:

```
C:\Users\vigne\OneDrive\Des  ×  +  ∨  
1  
Enter a value:20  
  
1.insert 2.delete 3.DISPLAY 4.EXIT:  
1  
Enter a value:30  
  
1.insert 2.delete 3.DISPLAY 4.EXIT:  
1  
Enter a value:40  
  
1.insert 2.delete 3.DISPLAY 4.EXIT:  
2  
20 deleted  
1.insert 2.delete 3.DISPLAY 4.EXIT:  
2  
30 deleted  
1.insert 2.delete 3.DISPLAY 4.EXIT:  
1  
Enter a value:30  
  
1.insert 2.delete 3.DISPLAY 4.EXIT:  
3  
40      30  
1.insert 2.delete 3.DISPLAY 4.EXIT:  
4  
  
-----  
Process exited after 51.07 seconds with return value 0  
Press any key to continue . . . |
```

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.**
- c) Display the contents of the linked list.**

```

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

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

struct node * create (int a)
{
    struct node* newnode = (struct node*) malloc (sizeof(struct node));
    newnode->data=a;
    newnode->next=0;
    return newnode;
};

struct node * insertatbeg(int a, struct node* head)
{
    struct node * newnode =create(a);
    newnode->next=head;
    head=newnode;
    return head;
};

struct node * insertatpos(int a, struct node *head, int pos )
{
    struct node*temp=head;
    int i=1;
    while(i>pos-1)
    {
        temp=temp->next;
    }
    struct node* newnode = create(a);
    newnode->next=temp->next;
    temp->next=newnode;
};

struct node * insertatend(int a, struct node* head)
{
    struct node*temp=head;
    while(temp->next!=NULL)
    {
        temp=temp->next;
    }
    struct node* newnode = create(a);
    temp->next=newnode;
};

struct node * insertatend(int a, struct node* head)
{
    struct node*temp=head;
    while(temp->next!=NULL)
    {
        temp=temp->next;
    }
    struct node* newnode = create(a);
    temp->next=newnode;
};

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

void main()
{
    int choice, pos,value;
    struct node*head=NULL;
    printf("enter 1.beg 2.end, 3.pos 4.display 5.exit");
    do{

```

```

{
    struct node* temp= head;
    while(temp->next!=NULL)
    {
        printf("%d \t -> \t",temp->data);
        temp=temp->next;
    }
    printf("%d",temp->data);
}

void main()
{
    int choice, pos,value;
    struct node*head=NULL;
    printf("enter 1.beg 2.end, 3.pos 4.display 5.exit");
    do{

        printf("\n Enter choice \t");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:printf("\n Enter the element to be inserted at beginning \t");
                    scanf("%d",&value);
                    head=insertatbeg(value,head);
                    break;
            case 2: printf("\n Enter the element to be inserted at end \t");
                    scanf("%d",&value);
                    insertatend(value,head);
                    break;
            case 3:printf("\n Enter the element to be inserted at position \t");
                    scanf("%d",&value);
                    printf("\n Enter pos");
                    scanf("%d",&pos);
                    insertatpos(value,head,pos);
                    break;
            case 4: display(head);
                    break;
            case 5: exit(0);

        }
    }while(choice != 6);
}

```

Output:

```

^C:\Users\vigne\OneDrive\Do  x  +  v
enter 1.beg 2.end, 3.pos 4.display 5.exit
Enter choice  1

Enter the element to be inserted at beginning  20
Enter choice  1

Enter the element to be inserted at beginning  30
Enter choice  2

Enter the element to be inserted at end          45
Enter choice  2

Enter the element to be inserted at end          60
Enter choice  3

Enter the element to be inserted at position  3
Enter pos3

Enter choice  4
30  ->  3      ->  20      ->  45      ->  60
Enter choice  5

Process returned 0 (0x0)   execution time : 27.893 s
Press any key to continue.

```


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 *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;
}

struct node *insertAtBeginning(struct node *head, int data) {
    struct node *newNode = createNode(data);
    if (head == NULL) {
        head = newNode;
    } else {
        newNode->next = head;
        head = newNode;
    }
    return head;
}
```

```

struct node *deleteAtBeginning(struct node *head) {
    if (head == NULL) {
        printf("List is empty\n");
    } else {
        struct node *temp = head;
        head = head->next;
        free(temp);
        printf("Node deleted from the beginning\n");
    }
    return head;
}

```

```

struct node *deleteAtEnd(struct node *head) {
    if (head == NULL) {
        printf("List is empty\n");
    } else if (head->next == NULL) {
        free(head);
        head = NULL;
        printf("Node deleted from the end\n");
    } else {
        struct node *temp = head;
        struct node *prev = NULL;
        while (temp->next != NULL) {
            prev = temp;
            temp = temp->next;
        }
        prev->next = NULL;
        free(temp);
        printf("Node deleted from the end\n");
    }
    return head;
}

```

```

struct node *deleteAtPosition(struct node *head, int position) {
    if (head == NULL) {
        printf("List is empty\n");
    } else if (position == 1) {
        head = deleteAtBeginning(head);
    } else {
        struct node *temp = head;
        struct node *prev = NULL;
        int count = 1;
        while (temp != NULL && count < position) {
            prev = temp;
            temp = temp->next;
            count++;
        }
        if (temp == NULL) {
            printf("Invalid position\n");
        } else {
            prev->next = temp->next;
            free(temp);
            printf("Node deleted from position %d\n", position);
        }
    }
    return head;
}

```

```

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

```

```

int main() {
    struct node *head = NULL;
    head = insertAtEnd(head, 10);
    head = insertAtEnd(head, 20);
    head = insertAtEnd(head, 30);
    head = insertAtEnd(head, 40);
    head = insertAtEnd(head, 50);

    int choice, position;

    do {
        printf("\n1. Delete at beginning\n");
        printf("2. Delete at end\n");
        printf("3. Delete at a specific position\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                head = deleteAtBeginning(head);
                break;
            case 2:
                head = deleteAtEnd(head);
                break;
            case 3:
                printf("Enter the position to delete: ");
                scanf("%d", &position);
                head = deleteAtPosition(head, position);
                break;
            case 4:
                display(head);
                break;
            case 5:
                printf("Exiting the program\n");
                break;
        }
    } while (choice != 5);
}

```

```

C:\Users\vigne\OneDrive\Do x + v
1. Delete at beginning
2. Delete at end
3. Delete at a specific position
4. Display
5. Exit
Enter your choice: 4
Linked list: 10 20 30 40 50

1. Delete at beginning
2. Delete at end
3. Delete at a specific position
4. Display
5. Exit
Enter your choice: 1
Node deleted from the beginning

1. Delete at beginning
2. Delete at end
3. Delete at a specific position
4. Display
5. Exit
Enter your choice: 2
Node deleted from the end

1. Delete at beginning
2. Delete at end
3. Delete at a specific position
4. Display
5. Exit
Enter your choice: 4
Linked list: 20 30 40

1. Delete at beginning
2. Delete at end
3. Delete at a specific position
4. Display
5. Exit
Enter your choice: 3
Enter the position to delete: 2
Node deleted from position 2

1. Delete at beginning
2. Delete at end
3. Delete at a specific position
4. Display
5. Exit
Enter your choice: 4
Linked list: 20 40

1. Delete at beginning
2. Delete at end
3. Delete at a specific position
4. Display
5. Exit
Enter your choice: |

```

Lab program 6a:

WAP to Implement Single Link List with following operations

a) Sort the linked list.

b) Reverse the linked list.

c) Concatenation of two linked lists

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

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

struct Node *head = NULL;
struct Node *newnode = NULL;
struct Node *p = NULL;
struct Node *q = NULL;
struct Node *prevnode = NULL;
struct Node *currentnode = NULL;
struct Node *temp = NULL;
struct Node *i = NULL;
struct Node *j = NULL;

void insertatbeg(int data) {
    newnode = (struct Node*)malloc(sizeof(struct Node));
    newnode->data = data;
    newnode->next = head;
    head = newnode;
}

void insertatend(int data) {
    newnode = (struct Node*)malloc(sizeof(struct Node));
    newnode->data = data;
    newnode->next = NULL;

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

    struct Node *current = head;
    while (current->next != NULL) {
        current = current->next;
    }

    current->next = newnode;
}

void concatenate(struct Node *p, struct Node *q) {
    if (head == NULL) {
        head = p;
    } else {
        struct Node *current = head;
        while (current->next != NULL) {
            current = current->next;
        }

        current->next = p;
    }

    while (q != NULL) {
        insertatend(q->data);
        q = q->next;
    }
}

void reverse() {
    prevnode = NULL;
    currentnode = head;
    newnode = NULL;

    while (currentnode != NULL) {
        newnode = currentnode->next;
        currentnode->next = prevnode;
        prevnode = currentnode;
        currentnode = newnode;
    }

    head = prevnode;
}
```

```

void sortlist() {
    i = head;
    while (i != NULL) {
        j = head;
        while (j->next != NULL) {
            if (j->data > j->next->data) {
                int temp = j->data;
                j->data = j->next->data;
                j->next->data = temp;
            }
            j = j->next;
        }
        i = i->next;
    }
}

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

int main() {
    int choice;
    int data;
    printf("\n1. Insert at Beginning\n2. Insert at End\n3. Sort List\n4. Reverse\n5. Concatenate\n6. Display\n7. Exit\n");

    while (1) {
        printf("Enter your choice: ");
        scanf("%d", &choice);

```

```

1. Insert at Beginning
2. Insert at End
3. Sort List
4. Reverse
5. Concatenate
6. Display
7. Exit
Enter your choice: 1
Enter data: 45
Enter your choice: 1
Enter data: 80
Enter your choice: 2
Enter data: 60
Enter your choice: 2
Enter data: 12
Enter your choice: 6
80 -> 45 -> 60 -> 12 -> NULL
Enter your choice: 4
Enter your choice: 6
12 -> 60 -> 45 -> 80 -> NULL
Enter your choice: 3
Enter your choice: 6
12 -> 45 -> 60 -> 80 -> NULL
Enter your choice: 5
Enter the first linked list: 12 -> 45 -> 60 -> 80 -> NULL
Enter the second linked list: 50 60 12 13 -1
After concatenating the two lists, the concatenated list is: 50 -> 60 -> 12 -> 13 -> NULL
Enter your choice: 7
Exiting the program...

Process returned 0 (0x0) execution time : 61.745 s
Press any key to continue.

```

Lab program 6b:

WAP to implement Stack & Queues using Linked Representation a)Stack

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

struct Node {
    int data;
    struct Node *link;
};

void display(struct Node *top) {
    if (top != NULL) {
        printf("Stack elements are:\n");
        while (top != NULL) {
            printf("%d\n", top->data);
            top = top->link;
        }
        printf("\n");
    } else {
        printf("Stack is empty\n");
    }
}

struct Node *push(struct Node *top, int x) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Stack Overflow\n");
        return top;
    }

    newNode->data = x;
    newNode->link = top;
    top = newNode;

    return top;
}

struct Node *pop(struct Node *top, int *poppedElement) {
    if (top == NULL) {
        printf("Stack Underflow\n");
        *poppedElement = -1;
        return NULL;
    }

    struct Node *temp = top;
    *poppedElement = temp->data;
    top = top->link;
    free(temp);

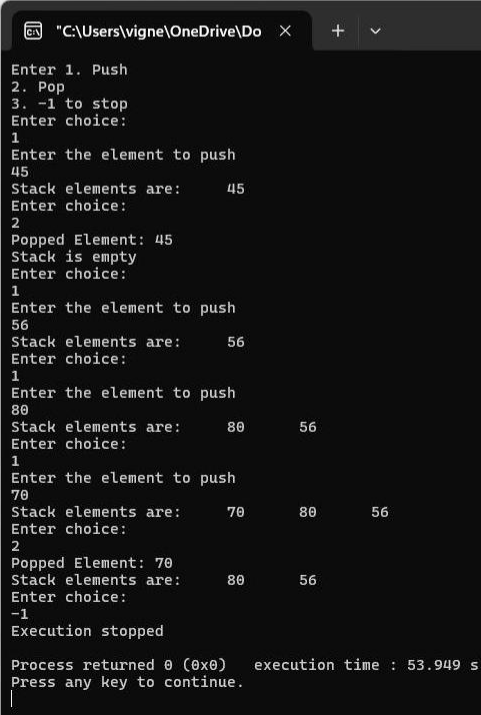
    return top;
}

int main() {
    int choice, n, poppedElement;
    struct Node *top = NULL;
    printf("Enter 1. Push\n2. Pop\n3. -1 to stop\n");
    while (1) {
        printf("Enter choice:\n");
        scanf("%d", &choice);

        if (choice == -1) {
            printf("Execution stopped\n");
            break;
        }

        switch (choice) {
            case 1:
                printf("Enter the element to push\n");
                scanf("%d", &n);
                top = push(top, n);
                break;
            case 2:
                top = pop(top, &poppedElement);
                if (poppedElement != -1) {
                    printf("Popped Element: %d\n", poppedElement);
                }
                display(top);
            }
    }

    return 0;
}
```



```
C:\Users\vigne\OneDrive\Do x + v
Enter 1. Push
2. Pop
3. -1 to stop
Enter choice:
1
Enter the element to push
45
Stack elements are: 45
Enter choice:
2
Popped Element: 45
Stack is empty
Enter choice:
1
Enter the element to push
56
Stack elements are: 56
Enter choice:
1
Enter the element to push
80
Stack elements are: 80 56
Enter choice:
1
Enter the element to push
70
Stack elements are: 70 80 56
Enter choice:
2
Popped Element: 70
Stack elements are: 80 56
Enter choice:
-1
Execution stopped

Process returned 0 (0x0) execution time : 53.949 s
Press any key to continue.
```

b)Queue

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

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

void display(struct Node* front) {
    if (front == NULL) {
        printf("Queue is empty\n");
        return;
    }

    struct Node* temp = front;
    printf("Queue elements are:\t");
    while (temp != NULL) {
        printf("%d\t", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

void enqueue(struct Node* front, struct Node* rear, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Queue Overflow\n");
        return;
    }

    newNode->data = data;
    newNode->next = NULL;

    if (rear == NULL) {
        front = rear = newNode;
        return;
    }

    if (rear == NULL) {
        front = rear = newNode;
        return;
    }

    rear->next = newNode;
    rear = newNode;
}

int dequeue(struct Node* front, struct Node* rear) {
    if (front == NULL) {
        printf("Queue Underflow\n");
        return -1;
    }

    struct Node* temp = front;
    int dequeuedData = temp->data;

    front = front->next;

    if (front == NULL) {
        rear = NULL;
    }

    free(temp);
    return dequeuedData;
}

int main() {
    int choice, n, dequeuedElement;
    struct Node* front = NULL;
    struct Node* rear = NULL;
    printf("Enter 1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
    while (1) {
        printf("Enter choice\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element to enqueue\n");
                scanf("%d", &n);
                enqueue(front, rear, n);
                break;
        }
    }
}
```

```

int main() {
    int choice, value;

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

        switch (choice) {
            case 1:
                printf("Enter the value to be inserted: ");
                scanf("%d", &value);
                enqueue(value);
                break;
            case 2:
                value = dequeue();
                if (value != -1) {
                    printf("Deleted value: %d\n", value);
                }
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}

```

Output:

Lab program 7:

WAP to Implement doubly link list with primitive operations

- a) Create a doubly linked list.
- b) Insert a new node to the left of the node.
- c) Delete the node based on a specific value
- d) Display the contents of the list

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

struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};

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

void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = create(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        newNode->next = *head;
        (*head)->prev = newNode;
        *head = newNode;
    }
}

void deleteNode(struct Node** head, int value) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = *head;
    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Value not found in the list\n");
        return;
    }
}
```

```

    }
    if (temp == NULL) {
        printf("Value not found in the list\n");
        return;
    }
    if (temp->prev == NULL) {
        *head = temp->next;
        if (temp->next != NULL) {
            temp->next->prev = NULL;
        }
    } else {
        temp->prev->next = temp->next;
        if (temp->next != NULL) {
            temp->next->prev = temp->prev;
        }
    }
    free(temp);
}

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

int main() {
    struct Node* head = NULL;
    int choice, data;
    printf("1. Insert at beginning\n");
    printf("2. Delete node based on specific value\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    while (1) {
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the data to be inserted: ");
                scanf("%d", &data);
                insertAtBeginning(&head, data);
                break;
            case 2:
                printf("Enter the value to be deleted: ");
                scanf("%d", &data);
                deleteNode(&head, data);
                break;
            case 3:
                display(head);
                break;
            case 4:
                printf("Exiting...\n");
                exit(EXIT_SUCCESS);
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}

```

```
/tmp/oIoPCUSYwa.o
```

1. Insert at beginning
2. Delete node based on specific value
3. Display
4. Exit

```
Enter your choice: 1
```

```
Enter the data to be inserted: 40
```

```
Enter your choice: 1
```

```
Enter the data to be inserted: 50
```

```
Enter your choice: 1
```

```
Enter the data to be inserted: 30
```

```
Enter your choice: 2
```

```
Enter the value to be deleted: 50
```

```
Enter your choice: 3
```

```
List elements: 30 40
```

```
Enter your choice: 4
```

```
Exiting...
```

```
|
```

LeetCode Problem:

ScoreOfParentheses:

856. Score of Parentheses

Medium

Given a balanced parentheses string s , return the **score** of the string.

The **score** of a balanced parentheses string is based on the following rule:

- $()$ has score 1.
- AB has score $A + B$, where A and B are balanced parentheses strings.
- (A) has score $2 \times A$, where A is a balanced parentheses string.

Example 1:

Input: $s = "()"$
Output: 1

Example 2:

Input: $s = "(())"$
Output: 2

```
1 int scoreOfParentheses(char* s) {
2     int n=strlen(s),ans=0;
3     int d=0,i=0;
4     while(i<n) {
5         if(s[i]=='(') d++;
6         else {
7             d--;
8             if(i>0 && s[i-1]=='(') ans+=1<<d;
9         }
10        i++;
11    }
12    return ans;
13 }
```

Output:

☒ Testcase | [Test Result](#)

• Case 1 • Case 2 • Case 3

Input

$s =$

$"()"$

Output

1

Expected

1

☒ Testcase | [Test Result](#)

• Case 1 • Case 2 • Case 3

Input

$s =$

$"(())"$

Output

2

Expected

2

☒ Testcase | [> Test Result](#)

• Case 1

• Case 2

• Case 3

Input

s =

"() ()"

Output

2

Expected

2

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>

typedef struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
} TreeNode;

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

TreeNode* insertNode(TreeNode* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else if (data > root->data) {
        root->right = insertNode(root->right, data);
    }
    return root;
}
```

```

void inorderTraversal(TreeNode* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

void preorderTraversal(TreeNode* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

void postorderTraversal(TreeNode* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}

void displayTree(TreeNode* root) {
    printf("Elements in the tree (inorder traversal): ");
    inorderTraversal(root);
    printf("\n");
}

int main() {
    TreeNode* root = NULL;
    int choice, data;
    printf("\n1. Insert\n2. Inorder Traversal\n3. Preorder Traversal\n4. Postorder Traversal\n5. Display Tree\n6. Exit\n");
    do {
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter data to insert into the tree: ");
                scanf("%d", &data);
                root = insertNode(root, data);
                break;
            case 2:
                printf("Inorder Traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;
            case 3:
                printf("Preorder Traversal: ");
                preorderTraversal(root);
                printf("\n");
                break;
            case 4:
                printf("Postorder Traversal: ");
                postorderTraversal(root);
                printf("\n");
                break;
            case 5:
                displayTree(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. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 1
Enter data to insert into the tree: 5
```

```
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 1
Enter data to insert into the tree: 6
```

```
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 1
Enter data to insert into the tree: 4
```

```
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 1
Enter data to insert into the tree: 7
```

```
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 1
Enter data to insert into the tree: 3
```

```
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 5
Elements in the tree (inorder traversal): 3 4 5 6 7
```



```
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 2
Inorder Traversal: 3 4 5 6 7
```

```
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 3
Preorder Traversal: 5 4 3 6 7
```

```
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 4
Postorder Traversal: 3 4 7 6 5
```

```
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 6
Exiting...
Vismays-MacBook-Pro:LNM code no
```

Leet Code Problem:

Delete the Middle Node Of a Linked List:

2095. Delete the Middle Node of a Linked List Solved

Medium Topics Companies Hint

You are given the **head** of a linked list. **Delete the middle node**, and return the **head** of the modified linked list.

The **middle node** of a linked list of size n is the $\lfloor n / 2 \rfloor^{\text{th}}$ node from the **start** using **0-based indexing**, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x .

- For $n = 1, 2, 3, 4$, and 5 , the middle nodes are $0, 1, 1, 2$, and 2 , respectively.

Example 1:

Diagram of a linked list with nodes: 1 (index 0) → 3 (index 1) → 4 (index 2) → 7 (index 3, highlighted in red) → 1 (index 4) → 2 (index 5) → 6 (index 6).

Input: head = [1,3,4,7,1,2,6]
Output: [1,3,4,1,2,6]

Explanation:
The above figure represents the given linked list. The indices of the nodes are written below.
Since $n = 7$, node 3 with value 7 is the middle node, which is marked in red.
We return the new list after removing this node.

```
1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     struct ListNode *next;
6  * };
7  */
8 struct ListNode* deleteMiddle(struct ListNode* head) {
9     struct ListNode *fast, *slow, *prev;
10    fast=head;
11    slow=head;
12    while(fast != NULL && fast->next != NULL)
13    {
14        fast=fast->next->next;
15        prev=slow;
16        slow=slow->next;
17    }
18    prev->next=slow->next;
19    free(slow);
20    return head;
21 }
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Output:

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

head =
[1,3,4,7,1,2,6]

Output

[1,3,4,1,2,6]

Expected

[1,3,4,1,2,6]

Testcase

Test Result

Accepted Runtime: 5 ms

Case 1

Case 2

Case 3

Input

head =
[1,2,3,4]

Output

[1,2,4]

Expected

[1,2,4]

Contribute a testcase

Accepted Runtime: 5 ms

Case 1

Case 2

Case 3

Input

head =
[2,1]

Output

[2]

Expected

[2]

Contribute a test

Odd Even Linked List

328. Odd Even Linked List

Medium Topics Companies

Given the **head** of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return the **reordered list**.

The **first** node is considered **odd**, and the **second** node is **even**, and so on.

Note that the relative order inside both the even and odd groups should remain as it was in the input.

You must solve the problem in $O(1)$ extra space complexity and $O(n)$ time complexity.

Example 1:

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

Example 2:

```

1  struct ListNode* next;
2  * };
3  */
4  struct ListNode* oddEvenList(struct ListNode* head) {
5      if (head == NULL || head->next == NULL) {
6          return head;
7      }
8      struct ListNode *odd = head;
9      struct ListNode *even = head->next;
10     struct ListNode *evenHead = even;
11     struct ListNode *oddHead = odd;
12
13     while (even != NULL && even->next != NULL) {
14         odd->next = even->next;
15         odd = odd->next;
16
17         if (odd != NULL) {
18             even->next = odd->next;
19             even = even->next;
20         }
21     }
22     odd->next = evenHead;
23
24     return head;
25 }

```

Saved to local

Ln 28, Col 26

Testcase Test Result

Accepted Runtime: 5 ms

Output:

✓

Testcase

>

Test Result

Accepted

Runtime: 0 ms

• Case 1

• Case 2

Input

head =
[1,2,3,4,5]

Output

[1,3,5,2,4]

Expected

[1,3,5,2,4]

The screenshot shows a test runner interface with a dark theme. At the top, there is a header bar with a green checkmark icon, the text "Testcase", and a green arrow icon followed by "Test Result". Below this, the word "Accepted" is displayed in large green font, and "Runtime: 0 ms" is shown in white. There are two tabs: "Case 1" (active, highlighted with a green dot) and "Case 2" (inactive, highlighted with a grey dot). Under the "Input" section, the text "head =" is followed by the array "[2,1,3,5,6,4,7]" in a light blue font. The "Output" section shows the array "[2,3,6,7,1,5,4]" in a light blue font. The "Expected" section shows the array "[2,3,6,7,1,5,4]" in a light blue font.

Lab program 9: Write a Program to traverse a graph using BFS method.

```
#include <stdio.h>

void bfs(int a[10][10], int n, int u) {
    int f = 0, r = -1, q[10] = {0}, v, s[10] = {0};
    printf("The nodes visited from %d: ", u);

    q[++r] = u;
    s[u] = 1;
    printf("%d ", u);

    while (f <= r) {
        u = q[f++];

        for (v = 0; v < n; v++) {
            if (a[u][v] == 1 && s[v] == 0) {
                printf("%d ", v);
                s[v] = 1;
                q[++r] = v;
            }
        }
    }
    printf("\n");
}
```

```
int main() {
    int n, a[10][10], source, i, j;
    printf("\nEnter the number of nodes: ");
    scanf("%d", &n);
    printf("\nEnter the adjacency matrix:\n");

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &a[i][j]);
        }
    }
}
```

```
q[++r] = u;
s[u] = 1;
printf("%d ", u);

while (f <= r) {
    u = q[f++];

    for (v = 0; v < n; v++) {
        if (a[u][v] == 1 && s[v] == 0) {
            printf("%d ", v);
            s[v] = 1;
            q[++r] = v;
        }
    }
    printf("\n");
}

int main() {
    int n, a[10][10], source, i, j;
    printf("\nEnter the number of nodes: ");
    scanf("%d", &n);
    printf("\nEnter the adjacency matrix:\n");

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    for (source = 0; source < n; source++) {
        bfs(a, n, source);
    }

    return 0;
}
```

Output:

```
Enter the number of nodes: 4
```

```
Enter the adjacency matrix:
```

```
0 1 1 0
```

```
1 0 1 1
```

```
1 1 0 1
```

```
0 1 1 0
```

```
The nodes visited from 0: 0 1 2 3
```

```
The nodes visited from 1: 1 0 2 3
```

```
The nodes visited from 2: 2 0 1 3
```

```
The nodes visited from 3: 3 1 2 0
```

○ Vismays-MacBook-Pro: LNM code notes

```

#define MAX_SIZE 100

int n;
int a[MAX_SIZE][MAX_SIZE];
int s[MAX_SIZE];

void dfs(int v) {
    s[v] = 1;
    for (int i = 1; i <= n; i++) {
        if (a[v][i] && !s[i]) {
            dfs(i);
        }
    }
}

int main() {
    int i, j, count = 0;

    printf("\nEnter number of vertices: ");
    scanf("%d", &n);

    for (i = 1; i <= n; i++) {
        s[i] = 0;
        for (j = 1; j <= n; j++) {
            a[i][j] = 0;
        }
    }

    printf("Enter the adjacency matrix:\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    dfs(1);
    for (i = 1; i <= n; i++) {
        if (s[i]) {
            count++;
        }
    }
}

```

```

if (a[v][i] && !s[i]) {
    dfs(i);
}
}

int main() {
    int i, j, count = 0;

    printf("\nEnter number of vertices: ");
    scanf("%d", &n);
    for (i = 1; i <= n; i++) {
        s[i] = 0;
        for (j = 1; j <= n; j++) {
            a[i][j] = 0;
        }
    }

    printf("Enter the adjacency matrix:\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    dfs(1);
    for (i = 1; i <= n; i++) {
        if (s[i]) {
            count++;
        }
    }

    if (count == n) {
        printf("Graph is connected\n");
    } else {
        printf("Graph is not connected\n");
    }

    return 0;
}

```

Output:

```

Enter number of vertices: 4
Enter the adjacency matrix:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
Graph is connected
Vismays-MacBook-Pro:LN code r

```


LeetCode Problem:

a)Delete Node In BST

Description | Editorial | Solutions | Submissions

450. Delete Node in a BST

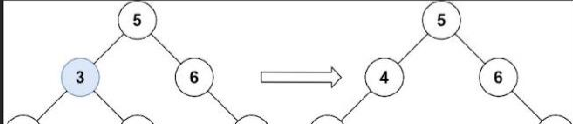
Medium Topics Companies

Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return the *root node reference* (possibly updated) of the BST.

Basically, the deletion can be divided into two stages:

1. Search for a node to remove.
2. If the node is found, delete the node.

Example 1:



</> Code

```
11 //
12 struct TreeNode* deleteNode(struct TreeNode* root, int key) {
13     if (root) {
14         if (key < root->val)
15             root->left = deleteNode(root->left, key);
16         else if (key > root->val)
17             root->right = deleteNode(root->right, key);
18         else {
19             if (!root->left && !root->right)
20                 return NULL;
21             if (!root->left || !root->right)
22                 return root->left ? root->left : root->right;
23             struct TreeNode* temp = root->left;
24             while (temp->right != NULL)
25                 temp = temp->right;
26             root->val = temp->val;
27             root->left = deleteNode(root->left, temp->val);
28         }
29     }
30     return root;
31 }
32
```

Output:

Testcase | Test Result

Accepted Runtime: 6 ms

Case 1 Case 2 Case 3

Input

root =
[5,3,6,2,4,null,7]

key =
3

Output

[5,2,6,null,4,null,7]

Expected

[5,4,6,2,null,null,7]

Testcase | Test Result

Accepted Runtime: 6 ms

Case 1 Case 2 Case 3

Input

root =
[5,3,6,2,4,null,7]

key =
0

Output

[5,3,6,2,4,null,7]

Expected

[5,3,6,2,4,null,7]

Testcase

>_ Test Result

Accepted Runtime: 6 ms

Case 1

Case 2

Case 3

Input

root =

[]

key =

0

Output

[]

Expected

[]

b)Find Bottom Left Tree Value

513. Find Bottom Left Tree Value

Medium

Topics

Companies

Given the `root` of a binary tree, return the leftmost value in the last row of the tree.

Example 1:

```

graph TD
    2((2)) --> 1((1))
    2 --> 3((3))
          
```

Input: root = [2,1,3]

Output: 1

Example 2:

```

graph TD
    1((1))
          
```

Input: root = [1]

Output: 1

C++

Auto

```

1  /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     TreeNode *left;
6  *     TreeNode *right;
7  *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8  *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9  *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
10 * };
11 */
12 int findBottomLeftValue(struct TreeNode* root) {
13     int value=root->val;
14     int mdepth=0;
15     void transverse(struct TreeNode* p,int depth){
16         if(!p)
17             return;
18         if(depth>mdepth){
19             mdepth=depth;
20             value=p->val;
21         }
22         transverse(p->left,depth+1);
23         transverse(p->right,depth+1);
24     }
25     transverse(root,0);
26     return value;
27 }
          
```

Saved to local

Ln 25, Col 20

Output:

☑ Testcase | >_ Test Result

Accepted Runtime: 3 ms

• Case 1 • Case 2

Input

root =
[2,1,3]

Output

1

Expected

1

☑ Testcase | >_ Test Result

Accepted Runtime: 3 ms

• Case 1 • Case 2

Input

root =
[1,2,3,4,null,5,6,null,null,7]

Output

7

Expected

7

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: $K \rightarrow L$ as $H(K) = K \bmod m$ (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

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

#define MAX_EMPLOYEES 100
#define HT_SIZE 10

typedef struct {
    int key;
} Employee;

typedef struct {
    Employee* entries[HT_SIZE];
} HashTable;

int hash(int key) {
    return key % HT_SIZE;
}

void initHashTable(HashTable* ht) {
    for (int i = 0; i < HT_SIZE; i++) {
        ht->entries[i] = NULL;
    }
}

void insertEmployee(HashTable* ht, Employee* emp) {
    int index = hash(emp->key);
    while (ht->entries[index] != NULL) {
        index = (index + 1) % HT_SIZE;
    }
    ht->entries[index] = emp;
}

void displayHashTable(HashTable* ht) {
    printf("\nHash Table:\n");
    for (int i = 0; i < HT_SIZE; i++) {
        if (ht->entries[i] != NULL) {
            printf("Index %d: Key %d\n", i, ht->entries[i]->key);
        } else {
            printf("Index %d: Empty\n", i);
        }
    }
}
```

```

while (ht->entries[index] != NULL) {
    index = (index + 1) % HT_SIZE;
}

ht->entries[index] = emp;
}

void displayHashTable(HashTable* ht) {
    printf("\nHash Table:\n");
    for (int i = 0; i < HT_SIZE; i++) {
        if (ht->entries[i] != NULL) {
            printf("Index %d: Key %d\n", i, ht->entries[i]->key);
        } else {
            printf("Index %d: Empty\n", i);
        }
    }
}

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

    int n;
    printf("Enter the number of employee records: ");
    scanf("%d", &n);

    printf("Enter the employee keys:\n");
    for (int i = 0; i < n; i++) {
        Employee* emp = (Employee*)malloc(sizeof(Employee));
        if (emp == NULL) {
            printf("Memory allocation failed!\n");
            exit(1);
        }
        scanf("%d", &emp->key);
        insertEmployee(&ht, emp);
    }
    displayHashTable(&ht);
    return 0;
}

```

Output:

```

C:\Users\vigne\OneDrive\Do x + v
Enter the number of employee records: 7
Enter the employee keys:
43
23
76
35
123
78
32

Hash Table:
Index 0: Empty
Index 1: Empty
Index 2: Key 32
Index 3: Key 43
Index 4: Key 23
Index 5: Key 35
Index 6: Key 76
Index 7: Key 123
Index 8: Key 78
Index 9: Empty

Process returned 0 (0x0)   execution time : 16.799 s
Press any key to continue.
|

```