**DSA Assignment**

**Agnim Gupta**

**2028083**

**CSSE**

**Q1**

#include <stdio.h>

#include <stdlib.h>

#define SIZE 40

struct queue

{

int items[SIZE]; int front;

int rear;

};

struct queue\* createQueue();

void enqueue(struct queue\* q, int);

int dequeue(struct queue\* q);

void display(struct queue\* q);

int isEmpty(struct queue\* q);

void printQueue(struct queue\* q);

struct node

{

int vertex;

struct node\* next;

};

struct node\* createNode(int);

struct Graph

{

    int numVertices;

    struct node\*\* adjLists;

    int\* visited;

};

void bfs(struct Graph\* graph, int startVertex)

{

    struct queue\* q = createQueue();

    graph->visited[startVertex] = 1;

    enqueue(q, startVertex);

    while (!isEmpty(q))

    {

        printQueue(q);

        int currentVertex = dequeue(q); printf("Visited %d\n", currentVertex);

        struct node\* temp = graph->adjLists[currentVertex];

        while (temp)

        {

            int adjVertex = temp->vertex;

            if (graph->visited[adjVertex] == 0)

            {

                graph->visited[adjVertex] = 1;

                enqueue(q, adjVertex);

            }

            temp = temp->next;

        }

    }

}

struct node\* createNode(int v)

{

    struct node\* newNode = malloc(sizeof(struct node)); newNode->vertex = v;

    newNode->next = NULL; return newNode;

}

struct Graph\* createGraph(int vertices)

{

    struct Graph\* graph = malloc(sizeof(struct Graph));

    graph->numVertices = vertices;

    graph->adjLists = malloc(vertices \* sizeof(struct node\*));

    graph->visited = malloc(vertices \* sizeof(int));

    int i;

    for (i = 0; i < vertices; i++)

    {

        graph->adjLists[i] = NULL;

        graph->visited[i] = 0;

    }

    return graph;

}

void addEdge(struct Graph\* graph, int src, int dest)

{

    struct node\* newNode = createNode(dest);

    newNode->next = graph->adjLists[src];

    graph->adjLists[src] = newNode;

    newNode = createNode(src);

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

    graph->adjLists[dest] = newNode;

}

struct queue\* createQueue()

{

    struct queue\* q = malloc(sizeof(struct queue)); q->front = -1;

    q->rear = -1; return q;

}

int isEmpty(struct queue\* q)

{

    if (q->rear == -1)

    return 1; else return 0;

}

void enqueue(struct queue\* q, int value)

{

    if (q->rear == SIZE - 1)

    printf("\nQueue is Full!!");

    else

    {

        if (q->front == -1) q->front = 0;

        q->rear++;

        q->items[q->rear] = value;

    }

}

int dequeue(struct queue\* q)

{

    int item;

    if (isEmpty(q))

    {

        printf("Queue is empty"); item = -1;

    }

    else {

        item = q->items[q->front]; q->front++;

        if (q->front > q->rear)

        {

            printf("Resetting queue "); q->front = q->rear = -1;

        }

    }

    return item;

}

void printQueue(struct queue\* q)

{

    int i = q->front;

    if (isEmpty(q))

    {

        printf("Queue is empty");

    }

    else

    {

    printf("\nQueue contains \n");

    for (i = q->front; i < q->rear + 1; i++)

    {

        printf("%d ", q->items[i]);

    }

    }

}

int main() {

struct Graph\* graph = createGraph(6); addEdge(graph, 0, 1);

addEdge(graph, 0, 2);

addEdge(graph, 1, 2);

addEdge(graph, 1, 4);

addEdge(graph, 1, 3);

addEdge(graph, 2, 4);

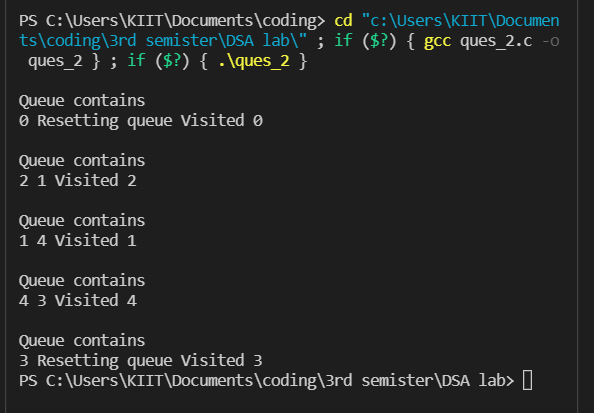
addEdge(graph, 3, 4);

bfs(graph, 0);

return 0;

}

**Output**

****

**Q2**

#include<stdio.h>

#include<stdlib.h>

struct node

{

    int value;

    struct node \*left\_child, \*right\_child;

};

struct node \*new\_node(int value)

{

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

    tmp->value = value;

    tmp->left\_child = tmp->right\_child = NULL; return tmp;

}

void print(struct node \*root\_node)

{

    if (root\_node != NULL)

    {

        print(root\_node->left\_child); printf("%d \n", root\_node->value); print(root\_node->right\_child);

    }

}

struct node\* insert\_node(struct node\* node, int value)

{

    if (node == NULL) return new\_node(value); if (value < node->value)

    {

    node->left\_child = insert\_node(node->left\_child, value);

    }

    else if (value > node->value)

    {

    node->right\_child = insert\_node(node->right\_child, value);

    }

    return node;

}

int main()

{

    printf("TechVidvan Tutorial: Implementation of a Binary Tree in C!\n\n"); struct node \*root\_node = NULL;

    root\_node = insert\_node(root\_node, 1); insert\_node(root\_node, 1);

    insert\_node(root\_node, 2);

    insert\_node(root\_node, 3);

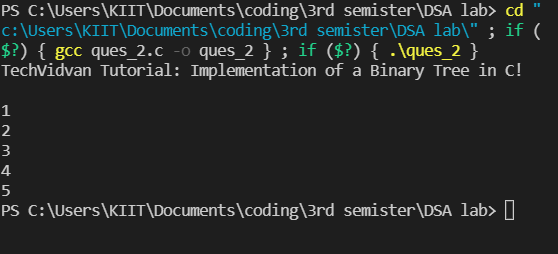
    insert\_node(root\_node, 4);

    insert\_node(root\_node, 5); print(root\_node);

    return 0;

}

**Output**

****

**Q3**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct node

{

    int data;

    struct node \*left;

    struct node \*right;

};

struct node \*root = NULL;

struct node\* createNode(int data)

{

    struct node \*newNode = (struct node\*)malloc(sizeof(struct node)); newNode->data = data;

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

}

struct queue

{

    int front, rear, size; struct node\* \*arr;

};

struct queue\* createQueue()

{

    struct queue\* newQueue = (struct queue\*) malloc(sizeof( struct queue )); newQueue->front = -1;

    newQueue->rear = 0;

    newQueue->size = 0;

    newQueue->arr = (struct node\*) malloc(100 \* sizeof( struct node )); return newQueue;

}

void enqueue(struct queue\* queue, struct node \*temp)

{

    queue->arr[queue->rear++] = temp;

    queue->size++;

}

struct node dequeue(struct queue queue)

{

    queue->size--;

    return queue->arr[++queue->front];

}

void insertNode(int data)

{

    struct node \*newNode = createNode(data); if(root == NULL)

    {

        root = newNode; return;

    }

    else

    {

        struct queue\* queue = createQueue();

        enqueue(queue, root);

        while(true)

        {

            struct node \*node = dequeue(queue);

            if(node->left != NULL && node->right != NULL) { enqueue(queue, node->left); enqueue(queue, node->right);

            }

            else

            {

            if(node->left == NULL)

            {

                node->left = newNode;

                enqueue(queue, node->left);

            }

            else

            {

                node->right = newNode; enqueue(queue, node->right);

            }

            break;

            }

        }

    }

}

void inorderTraversal(struct node \*node)

{

    if(root == NULL)

    {

        printf("Tree is empty\n"); return;

    }

    else

    {

        if(node->left != NULL) inorderTraversal(node->left);

        printf("%d ", node->data); if(node->right != NULL)

        inorderTraversal(node->right);

    }

}

int main()

{

    insertNode(1);

    printf("Binary tree after insertion: \n"); inorderTraversal(root); insertNode(2);

    insertNode(3);

    printf("\nBinary tree after insertion: \n"); inorderTraversal(root);

    insertNode(4); insertNode(5);

    printf("\nBinary tree after insertion: \n"); inorderTraversal(root);

    insertNode(6); insertNode(7);

    printf("\nBinary tree after insertion: \n"); inorderTraversal(root);

    return 0;

}

**Q4**

#include <stdio.h>

#include <stdlib.h>

struct node

{

    int data;

    struct node\* left;

    struct node\* right;

};

struct node\* newNode(int data)

{

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

    node->data = data; node->left = NULL; node->right = NULL; return (node);

}

void printPostorder(struct node\* node)

{

    if (node == NULL) return;

    printPostorder(node->left);

    printPostorder(node->right);

    printf("%d ", node->data);

}

void printInorder(struct node\* node)

{

    if (node == NULL) return;

    printInorder(node->left);

    printf("%d ", node->data);

    printInorder(node->right);

}

void printPreorder(struct node\* node)

{

    if (node == NULL)

    return;

    printf("%d ", node->data);

    printPreorder(node->left);

    printPreorder(node->right);

}

int main()

{

    struct node\* root = newNode(1);

    root->left = newNode(2);

    root->right = newNode(3);

    root->left->left = newNode(4);

    root->left->right = newNode(5);

    printf("\nPreorder traversal of binary tree is \n");

    printPreorder(root);

    printf("\nInorder traversal of binary tree is \n");

    printInorder(root);

    printf("\nPostorder traversal of binary tree is \n");

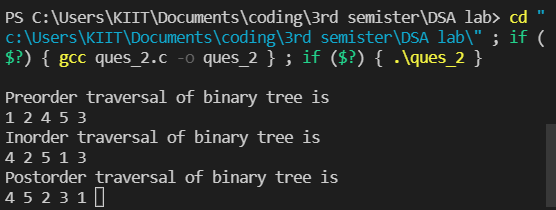
    printPostorder(root);

    getchar();

    return 0;

}

**Output**

****

**Q5**

#include<stdio.h>

#include<stdlib.h>

struct node

{

    int value;

    struct node \*left\_child, \*right\_child;

};

struct node \*new\_node(int value)

{

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

    tmp->value = value;

    tmp->left\_child = tmp->right\_child = NULL;

    return tmp;

}

void print(struct node \*root\_node)

{

    if (root\_node != NULL)

    {

        print(root\_node->left\_child);

        printf("%d \n", root\_node->value);

        print(root\_node->right\_child);

    }

}

struct node\* insert\_node(struct node\* node, int value)

{

    if (node == NULL) return new\_node(value); if (value < node->value)

    {

        node->left\_child = insert\_node(node->left\_child, value);

    }

    else if (value > node->value)

    {

        node->right\_child = insert\_node(node->right\_child, value);

    }

    return node;

}

int main()

{

    printf("TechVidvan Tutorial: Implementation of a Binary Tree in C!\n\n"); struct node \*root\_node = NULL;

    root\_node = insert\_node(root\_node, 1); insert\_node(root\_node, 1);

    insert\_node(root\_node, 2);

    insert\_node(root\_node, 3);

    insert\_node(root\_node, 4);

    insert\_node(root\_node, 5);

    insert\_node(root\_node, 6);

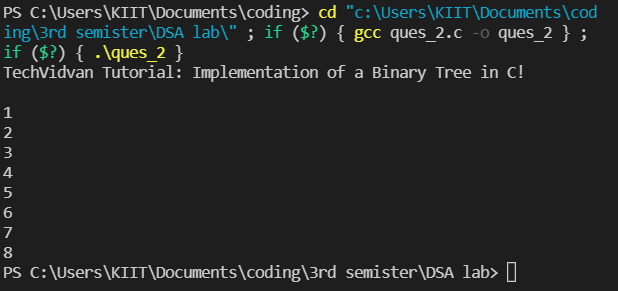
    insert\_node(root\_node, 7);

    insert\_node(root\_node, 8); print(root\_node);

    return 0;

}

**Output**

****