Bansilal Ramnath Agarwal Charitable Trust's
VISHWAKARMA INSTITUTE OF TECHNOLOGY – PUNE

ARTIFICIAL INTELLIGENCE

Division	A
Batch	2
GR-no	12311493
Roll no	54
Name	Atharva Kangralkar

Assignment 2:

1.A Implement BFS and DFS algorithm on Graph (Using Array)

1.B Implement BFS and DFS algorithm on Tree(Using linked list)

1. A Implement BFS and DFS algorithm on Graph(Using Array)

Description:-

BFS (Breadth-First Search): This program implements BFS using an adjacency matrix and a queue for level-wise traversal. It starts from a given node, enqueues it, and explores all its unvisited neighbors before moving deeper. The queue ensures FIFO (First-In-First-Out) order, preventing redundancy. The algorithm prints nodes as they are visited.

DFS (Depth-First Search): This program implements DFS using an adjacency matrix and recursion for depth-first traversal. It starts from a given node, marks it as visited, and recursively explores all its unvisited adjacent nodes. DFS follows backtracking to explore deep paths first. It ensures each node is visited once, avoiding infinite loops in cyclic graphs.

- **Code:**

```
#include<stdio.h>

int g[100][100],v[100],q[100],f=0,r=-1,n;

// Function to perform Breadth-First Search (BFS)
void bfs(int s){
   int i;
   q[++r]=s, v[s]=1;
   printf("BFS: ");
   while(f<=r){
      s=q[f++];
      printf("%d ",s);
      for(i=0;i<n;i++)
         if(g[s][i]&&!v[i])  // If an edge exists and the node is not visited
            q[++r]=i, v[i]=1;  // Enqueue the node and mark it as visited
   }
}
```

```
// Function to perform Depth-First Search (DFS)
void dfs(int s){
   int i;
   printf("%d ",s), v[s]=1;
   for(i=0;i<n;i++)
      if(g[s][i]&&!v[i])  // If an edge exists and the node is not visited
         dfs(i);  // Recursively call DFS for the adjacent node
}

int main(){
   int e,i,a,b,s;  // e: number of edges, a & b: edge endpoints, s: starting node
   printf("Enter number of nodes: ");
   scanf("%d",&n);
   printf("Enter number of edges: ");
   scanf("%d",&e);
   for(i=0;i<e;i++){
      printf("Enter edge %d: ",i+1);
      scanf("%d%d",&a,&b);
      g[a][b]=g[b][a]=1;
   }

   // Print the adjacency matrix
   printf("\nAdjacency Matrix:\n");
   for(i=0;i<n;i++){
      for(int j=0;j<n;j++)
         printf("%d ",g[i][j]);
      printf("\n");
   }

   printf("\nEnter start node for traversal: ");
   scanf("%d",&s);
   bfs(s);  // Perform BFS from the given starting node

   // Reset visited array for DFS
   for(i=0;i<n;i++) v[i]=0;

   printf("\nDFS: ");
   dfs(s);  // Perform DFS from the given starting node
}
```

Screenshots/Output:

```
Enter number of nodes: 5
Enter number of edges: 3
Enter edge 1: 0 1
Enter edge 2: 0 3
Enter edge 3: 1 3

Adjacency Matrix:
0 1 0 1 0
1 0 0 1 0
0 0 0 0 0
1 1 0 0 0
0 0 0 0 0

Enter start node for traversal: 0
BFS: 0 1 3
DFS: 0 1 3
```

1.B Implement BFS and DFS algorithm on Tree(Using linked list)

Description:- This program implements BFS and DFS using an adjacency list for efficient graph representation. BFS uses a queue for level-order traversal, while DFS is implemented using recursion for depth-first exploration. The adjacency list is sorted during insertion for ordered traversal. The program reads user input for vertices, edges, and the starting node, then performs BFS and DFS.

- **Code:**

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

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

struct Node* front = NULL;
struct Node* rear = NULL;


void Enqueue(struct Node* x) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = x->data;
    temp->left = x->left;
    temp->right = x->right;
    temp->next = NULL;

    if (front == NULL && rear == NULL) {
        front = rear = temp;
        return;
    }
```

```
    rear->next = temp;
    rear = temp;
}

struct Node* Dequeue() {
    if (front == NULL) return NULL;

    struct Node* temp = front;

    if (front == rear) {
        front = rear = NULL;
    } else {
        front = front->next;
    }

    return temp;
}

int isQueueEmpty() {
    return front == NULL;
}


struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = newNode->next = NULL;
    return newNode;
}

struct Node* buildTree() {
    int data;
    printf("Enter data (-1 for NULL): ");
    scanf("%d", &data);
    if (data == -1) return NULL;

    struct Node* root = createNode(data);
    printf("Enter left child of %d:\n", data);
    root->left = buildTree();
```

```
    printf("Enter right child of %d:\n", data);
    root->right = buildTree();
    return root;
}


void bfs(struct Node* root) {
    if (root == NULL) return;

    front = rear = NULL;
    Enqueue(root);

    while (!isQueueEmpty()) {
        struct Node* current = Dequeue();
        printf("%d ", current->data);
        if (current->left) Enqueue(current->left);
        if (current->right) Enqueue(current->right);
    }
}

void inorder(struct Node* root) {
    if (root) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

void preorder(struct Node* root) {
    if (root) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(struct Node* root) {
    if (root) {
        postorder(root->left);
        postorder(root->right);
```

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


int main() {
    printf("Build the binary tree:\n");
    struct Node* root = buildTree();

    int choice;
    do {
        printf("\nMENU\n");
        printf("1. BFS (Level Order)\n");
        printf("2. DFS - Inorder\n");
        printf("3. DFS - Preorder\n");
        printf("4. DFS - Postorder\n");
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        printf("Traversal Output: ");
        switch (choice) {
            case 1: bfs(root); break;
            case 2: inorder(root); break;
            case 3: preorder(root); break;
            case 4: postorder(root); break;
            case 5: printf("Exiting\n"); break;
            default: printf("Invalid choice!\n");
        }
        printf("\n");
    } while (choice != 5);

    return 0;
}
```

Screenshots/Output:

```
Build the binary tree:
Enter data (-1 for NULL): 5
Enter left child of 5:
Enter data (-1 for NULL): 3
Enter left child of 3:
Enter data (-1 for NULL): 7
Enter left child of 7:
Enter data (-1 for NULL): -1
Enter right child of 7:
Enter data (-1 for NULL): 8
Enter left child of 8:
Enter data (-1 for NULL): -1
Enter right child of 8:
Enter data (-1 for NULL): -1
Enter right child of 3:
Enter data (-1 for NULL): 2
Enter left child of 2:
Enter data (-1 for NULL): -1
Enter right child of 2:
Enter data (-1 for NULL): -1
Enter right child of 5:
Enter data (-1 for NULL): 6
Enter left child of 6:
Enter data (-1 for NULL): -1
Enter right child of 6:
Enter data (-1 for NULL): -1
```

```
MENU
1. BFS (Level Order)
2. DFS - Inorder
3. DFS - Preorder
4. DFS - Postorder
5. Exit
Enter choice: 1
Traversal Output: 5 3 6 7 2 8

MENU
1. BFS (Level Order)
2. DFS - Inorder
3. DFS - Preorder
4. DFS - Postorder
5. Exit
Enter choice: 2
Traversal Output: 7 8 3 2 5 6

MENU
1. BFS (Level Order)
2. DFS - Inorder
3. DFS - Preorder
4. DFS - Postorder
5. Exit
Enter choice: 3
Traversal Output: 5 3 7 8 2 6
```

```
MENU
1. BFS (Level Order)
2. DFS - Inorder
3. DFS - Preorder
4. DFS - Postorder
5. Exit
Enter choice: 4
Traversal Output: 8 7 2 3 6 5

MENU
1. BFS (Level Order)
2. DFS - Inorder
3. DFS - Preorder
4. DFS - Postorder
5. Exit
Enter choice: 5
Traversal Output: Exiting
```