

## EXPERIMENT-12

### Program:

Implementation of breadth first search by using c/c++ and write it's complexity.

### Pseudo Code:

```
BFS(G, s)
    for each vertex  $u \in G.V - \{s\}$ 
         $u.color = WHITE$ 
         $u.d = \infty$ 
         $u.\pi = NIL$ 
     $s.color = GRAY$ 
     $s.d = 0$ 
     $s.\pi = NIL$ 
     $Q = \emptyset$ 
    ENQUEUE(Q, s)
    while  $Q \neq \emptyset$ 
         $u = DEQUEUE(Q)$ 
        for each  $v \in G.Adj[u]$ 
            if  $v.color == WHITE$ 
                 $v.color = GRAY$ 
                 $v.d = u.d + 1$ 
                 $v.\pi = u$ 
                ENQUEUE(Q, v)
         $u.color = BLACK$ 
```

Here, G is the graph, s is the source node, V is the set of vertices in G, and Adj[u] is the adjacency list of vertex u. The algorithm starts by initializing all vertices to white colour, infinite distance, and no parent. The source node is coloured Gray, distance is set to 0, and parent is set to null. A queue Q is initialized with the source node. The algorithm then dequeues a vertex from the queue and colours it black. For each adjacent vertex that is white, it colours it Gray, sets its distance to the distance of its parent plus 1, sets its parent to the dequeued vertex, and enqueues it. The algorithm continues until the queue is empty.

### Input:

```
#include <stdio.h>

#include <stdlib.h>
```

```

#define MAX_NODES 100

int queue[MAX_NODES];

int front = -1, rear = -1;

void enqueue(int node) {
    if (rear == MAX_NODES - 1) {
        printf("Queue is full. Cannot enqueue %d.\n", node);
    } else {
        if (front == -1) {
            front = 0;
        }
        rear++;
        queue[rear] = node;
    }
}

int dequeue() {
    int node;
    if (front == -1) {
        printf("Queue is empty.\n");
        return -1;
    } else {
        node = queue[front];
        front++;
        if (front > rear) {
            front = rear = -1;
        }
        return node;
    }
}

int isEmpty() {
    return front == -1;
}

// BFS algorithm

void BFS(int adjList[][MAX_NODES], int nodes, int start) {

```

```

int visited[MAX_NODES] = {0};

    enqueue(start);
visited[start] = 1;
printf("Breadth-First Search starting from node %d: ", start);
while (!isEmpty()) {
    int current = dequeue();
    printf("%d ", current);
    for (int i = 0; i < nodes; i++) {
        if (adjList[current][i] && !visited[i]) {
            enqueue(i);
            visited[i] = 1;
        }
    }
}

int main() {
    int nodes, edges;
    printf("Enter the number of nodes: ");
    scanf("%d", &nodes);
    printf("Enter the number of edges: ");
    scanf("%d", &edges);
    int adjList[MAX_NODES][MAX_NODES] = {0};
    printf("Enter the edges (format: from to):\n");
    for (int i = 0; i < edges; i++) {
        int from, to;
        scanf("%d %d", &from, &to);
        adjList[from][to] = 1;
        adjList[to][from] = 1; // For undirected graph
    }
    int startNode;
    printf("Enter the starting node for BFS: ");
    scanf("%d", &startNode);
    BFS(adjList, nodes, startNode);
}

```

```
    return 0;  
}
```

**Output:**

```
Boddu Asmitha Bhavya_A2305221386  
Enter the number of nodes: 10  
Enter the number of edges: 13  
Enter the edges (format: from to):  
1 2  
2 3  
1 3  
1 4  
4 5  
5 7  
7 6  
6 9  
9 8  
8 4  
9 10  
6 5  
6 8  
Enter the starting node for BFS: 1  
Breadth-First Search starting from node 1: 1 2 3 4 5 8 6 7 9
```

**Complexity:**

$O(V+E)$ , where  $V$  is the number of nodes and  $E$  is the number of edges.

## EXPERIMENT-13

### Program:

Implementation of Depth first search by using c/c++ and write the it's complexity.

### Pseudo Code:

DFS(G, u)

    u.visited = true

    for each v  $\in$  G.Adj[u]

        if v.visited == false

            DFS(G,v)

init() {

    For each u  $\in$  G

        u.visited = false

    For each u  $\in$  G

        DFS(G, u)

}

### Input:

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

```
#include <stdbool.h>
```

```
#define MAX_NODES 100
```

```
bool visited[MAX_NODES];
```

```
int adjacency_matrix[MAX_NODES][MAX_NODES];
```

```
int num_nodes;
```

```
void initialize() {
```

```
    for (int i = 0; i < MAX_NODES; i++) {
```

```
        visited[i] = false;
```

```
        for (int j = 0; j < MAX_NODES; j++) {
```

```
            adjacency_matrix[i][j] = 0;
```

```
        }
```

```
    }
```

```
}
```

```

void addEdge(int start, int end) {
    adjacency_matrix[start][end] = 1;
    adjacency_matrix[end][start] = 1;
}

void dfs(int node) {
    printf("%d ", node);
    visited[node] = true;
    for (int i = 0; i < num_nodes; i++) {
        if (adjacency_matrix[node][i] && !visited[i]) {
            dfs(i);
        }
    }
}

int main() {
    int num_edges;
    int start, end;

    printf("Boddu Asmitha Bhavya_A2305221386");
    printf("\nEnter the number of nodes: ");
    scanf("%d", &num_nodes);
    initialize();
    printf("Enter the number of edges: ");
    scanf("%d", &num_edges);
    printf("Enter the edges (start end):\n");
    for (int i = 0; i < num_edges; i++) {
        scanf("%d %d", &start, &end);
        addEdge(start, end);
    }

    int start_node;
    printf("Enter the starting node for DFS: ");
    scanf("%d", &start_node);
    printf("Depth First Search (DFS) starting from node %d: ", start_node);

```

```
    dfs(start_node);  
  
    return 0;  
}
```

**Output:**

```
Boddu Asmitha Bhavya_A2305221386  
Enter the number of nodes: 10  
Enter the number of edges: 13  
Enter the edges (start end):  
1 2  
1 3  
2 3  
1 4  
4 5  
5 7  
7 6  
6 9  
9 8  
8 4  
8 6  
6 5  
9 10  
Enter the starting node for DFS: 4  
Depth First Search (DFS) starting from node 4: 4 1 2 3 5 6 7 8 9
```

**Complexity:**

$O(V + E)$ , where  $V$  is the number of vertices and  $E$  is the number of edges in the graph.

## EXPERIMENT-14

### Program:

Implementation of n-Queen problem using c/c++ and write it's complexity.

### Pseudo Code:

```
check_row(nxnarray, row, n){
    for(j=0;j<n;j++){//this should be same for most languages
        if(nxnarray[row][j]==1){
            return false;
        }
    }
    return true;
}bool placeQueens(int current column, int current row) {
    // Base case.
    if all columns are filled, then return true
    for each row of the board, do
        if isValid (board, i, col), then set queen at place (i, col) in the board
        if solveNQueen (board, col+1) = true, then return true
        otherwise remove queen from place (i, col) from board.
    done
    return false
}Algorithm NQueens (k, n) //Prints all Solution to the n-queens problem {
    for i := 1 to n do {
        if Place (k, i) then {
            x [k] := i;
            if (k = n) then write (x [1 : n])
            else NQueens (k+1, n);
        }
    }
}Algorithm Place (k, i) {
    for j := 1 to k-1 do
        if ((x [ j ] = // in the same column or (Abs (x [ j ] - i) =Abs (j - k))) // or in the same diagonal
            then return false;
    return true;}
```



**Input:**

```
#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <math.h>

int x[10];

bool canPlace(int k, int i) {
    for (int j = 1; j < k; j++) {
        if (x[j] == i || abs(x[j] - i) == abs(j - k)) {
            return false;
        }
    }
    return true;
}

void printSolution(int n) {
    printf("Solution found:\n");
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (x[i] == j) {
                printf("Q ");
            } else {
                printf(". ");
            }
        }
        printf("\n");
    }
    printf("\n");
}

void placeQueens(int k, int n) {
    for (int i = 1; i <= n; i++) {
        if (canPlace(k, i)) {
            x[k] = i;
            if (k == n) {
                printSolution(n);
            } else {
                placeQueens(k + 1, n);
            }
        }
    }
}
```

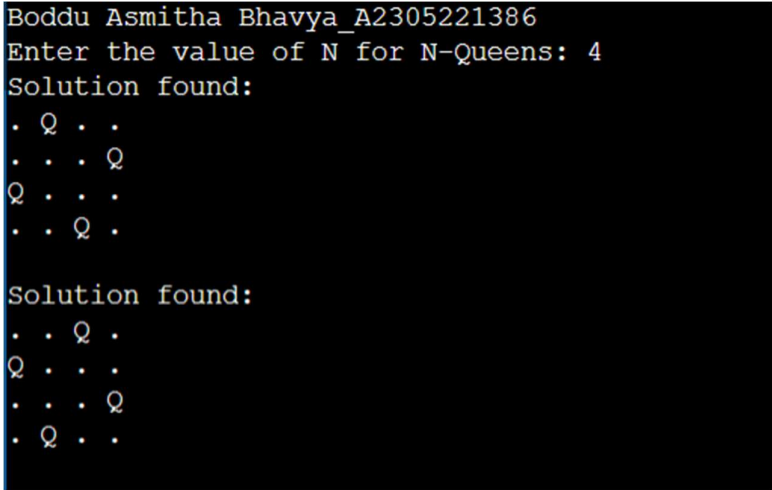
```

        }    }  }}

void solveNQueens(int n) {
    placeQueens(1, n);
}int main() {
    int n;
    printf("Boddu Asmitha Bhavya_A2305221386");
    printf("\nEnter the value of N for N-Queens: ");
    scanf("%d", &n);
    if (n <= 0) {
        printf("Please enter a positive integer.\n");
        return 1;
    }
    if (n > 10) {
        printf("This program is designed for N <= 10 due to performance limitations.\n");
        return 1;
    }
    solveNQueens(n);
    return 0;}

```

#### Output:



```

Boddu Asmitha Bhavya_A2305221386
Enter the value of N for N-Queens: 4
Solution found:
. Q . .
. . . Q
Q . . .
. . Q .

Solution found:
. . Q .
Q . . .
. . . Q
. Q . .

```

#### Complexity:

Time complexity is  $O(N!)$