PRACTICAL - 13

AIM : To implement DFS and BFS

PSEUDO CODE :

DFS_Traversal(adj[][], visited[], start_vertex)
Step 1.Start.
Step 2. Input the number of vertices "n" and number of edges "e".
Step 3. Create a 2D adjacency matrix "adj" of size n x n and initialize all entries to 0.
Step 4. Create a boolean array "visited" of size "n" and initialize all entries to false.
Step 5. For each edge from 1 to e:
a. Input the two connected vertices "x" and "y".
b. Set adj[x][y] and adj[y][x] to 1.
Step 6. Start DFS traversal from vertex 0:
a. Mark the current node "v" as visited: visited[v] = true.
b. Print the current node "v".
c. For each vertex "i" from 0 to n-1:
i. If adj[v][i] equals 1 and vertex "i" is not visited:
1. Recursively call step 6 for vertex "i".
Step 7. End.

BFS_Traversal(adj[][], start_vertex)
Step 1. Start.
Step 2. Input the number of vertices "n" and number of edges "e".
Step 3. Create a 2D adjacency matrix "adj" of size n x n and initialize all entries to 0.
Step 4. Create a boolean array "visited" of size "n" and initialize all entries to false.
Step 5. For each edge from 1 to e:
a. Input the two connected vertices "x" and "y".
b. Set adj[x][y] and adj[y][x] to 1.
Step 6. Create a queue "q" and enqueue start_vertex onto "q".
Step 7. Mark the start_vertex as visited: visited[start_vertex] = true.
Step 8. While "q" is not empty:
a. Dequeue a vertex "v" from "q" and print it.
b. For each vertex "i" from 0 to n-1:
i. If adj[v][i] equals 1 and vertex "i" is not visited:
1. Mark vertex "i" as visited: visited[i] = true.
2. Enqueue vertex "i" onto "q".
Step 9. End.

CODE :

(c) DFS (Depth First Search)

```
#include <stdio.h>
#include <stdbool.h>
#define MAX 100
int adj[MAX][MAX];
bool visited[MAX];
int n;
void DFS(int v) {
visited[v] = true;
printf("%d "
, v);
for (int i = 0; i < n; i++) {
if (adj[v][i] && !visited[i]) {
DFS(i);
}
}
}
int main() {
int edges, x, y;
printf("Rishita Chaubey A2305221265\n");
printf("Enter the number of vertices and edges: ");
scanf("%d %d"
, &n, &edges);
printf("Enter the edges (format: vertex1 vertex2):\n");for (int i = 0; i < edges; i++) {
scanf("%d %d"
, &x, &y);
adj[x][y] = 1;
adj[y][x] = 1;
}
for (int i = 0; i < n; i++) {
visited[i] = false;
}
printf("DFS traversal order starting from vertex 0: ");
DFS(0);
return 0;
}
```

(d) BFS (Breadth First Search)

```
#include <stdio.h>
#include <stdbool.h>
#define MAX 100
int adj[MAX][MAX];
bool visited[MAX];
int n;
int queue[MAX], front = -1, rear = -1;
void enqueue(int vertex) {
if (rear == MAX - 1) return;
if (front == -1) front = 0;
rear++;
queue[rear] = vertex;
}
int dequeue() {
if (front == -1) return -1;
int item = queue[front];
front++;
if (front > rear) front = rear = -1;
return item;
}void BFS(int v) {
enqueue(v);
visited[v] = true;
while (front != -1) {
int current = dequeue();
printf("%d "
, current);
for (int i = 0; i < n; i++) {
if (adj[current][i] && !visited[i]) {
enqueue(i);
visited[i] = true;
}
}
}
}
int main() {
int edges, x, y;
printf("Rishita Chaubey A2305221265\n");
printf("Enter the number of vertices and edges: ");
```

```
scanf("%d %d"
, &n, &edges);
printf("Enter the edges (format: vertex1 vertex2):\n");
for (int i = 0; i < edges; i++) {
scanf("%d %d"
, &x, &y);
adj[x][y] = 1;
adj[y][x] = 1;
}
for (int i = 0; i < n; i++) {
visited[i] = false;
}
printf("BFS traversal order starting from vertex 0: ");
BFS(0);
return 0;
}
```

OUTPUT

(a) DFS

```
Rishita Chaubey A2305221265
Enter the number of vertices and edges: 5 4
Enter the edges (format: vertex1 vertex2):
0 1
0 2
1 3
3 4
DFS traversal order starting from vertex 0: 0 1 3 4 2
```

(b) BFS

```
Rishita Chaubey A2305221265
Enter the number of vertices and edges: 5 4
Enter the edges (format: vertex1 vertex2):
0 1
0 2
1 3
3 4
BFS traversal order starting from vertex 0: 0 1 2 3 4
```

PRACTICAL - 14

AIM : To implement N-queen Problem

PSEUDO CODE :

NQueens(board[], row, n)

Step 1. Start.
Step 2. If row is equal to n, print the board configuration.
Step 3. For each column col from 0 to n-1:
a. If it's safe to place a queen at board[row][col]:
i. Place a queen at board[row][col].
ii. Recursively call NQueens for next row.
iii. If placement results in a solution, return true.
iv. If no place results in a solution, remove the queen (backtrack).
Step 4. If no queen can be placed in this row, return false.
Step 5. End.

CODE :

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
bool isSafe(int board[], int row, int col, int N) {
for (int i = 0; i < col; i++) {
if (board[i] == row || board[i] - i == row - col || board[i] + i == row + col)
return false;
}
return true;
}
bool solveNQueens(int board[], int col, int N) {
if (col >= N) {
for (int i = 0; i < N; i++) {
for (int j = 0; j < N; j++) {
printf("%d ", (board[i] == j) ? 1 : 0);
```

```
}
printf("\n");
}
return true;
}
for (int i = 0; i < N; i++) {
if (isSafe(board, i, col, N)) {
board[col] = i;
if (solveNQueens(board, col + 1, N)) return true;
board[col] = -1; // backtrack
}
}
return false;
}
int main() {
int N;
printf("Aabhas Kumar Jha A2305221279\n");
printf("Enter the number of queens (board size): ");
scanf("%d"
, &N);
int* board = (int*)malloc(N * sizeof(int));
for (int i = 0; i < N; i++) {
board[i] = -1;
}
if (!solveNQueens(board, 0, N)) {
printf("Solution does not exist.");
}
free(board);
return 0;
}
```

OUTPUT :

```
Aabhas Kumar Jha A2305221279
Enter the number of queens (board size): 8
1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
```