BREADTH FIRST SEARCH(BFS):

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

#define MAX_VERTICES 100

// Queue structure for BFS
typedef struct Queue {
    int items[MAX_VERTICES];
    int front, rear;
} Queue;

// Initialize the queue
void initQueue(Queue* q) {
    q->front = -1;
    q->rear = -1;
}

// Check if the queue is empty
bool isEmpty(Queue* q) {
    return q->front == -1;
}

// Add an item to the queue
void enqueue(Queue* q, int item) {
    if (q->rear == MAX_VERTICES - 1) {
        printf("Queue is full\n");
        return;
    }
    if (q->front == -1) {
```

```
        q->front = 0;
    }
    q->items[++(q->rear)] = item;
}

// Remove an item from the queue
int dequeue(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    int item = q->items[q->front];
    if (q->front >= q->rear) {
        q->front = q->rear = -1; // Queue is empty
    } else {
        q->front++;
    }
    return item;
}

// Graph structure
typedef struct Graph {
    int vertices;
    bool adjMatrix[MAX_VERTICES][MAX_VERTICES];
} Graph;

// Initialize the graph
void initGraph(Graph* g, int vertices) {
    g->vertices = vertices;
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
```

```
            g->adjMatrix[i][j] = false;
        }
    }
}

// Add edge to the graph
void addEdge(Graph* g, int start, int end) {
    g->adjMatrix[start][end] = true;
    g->adjMatrix[end][start] = true; // For undirected graph
}
// Perform BFS on the graph
void bfs(Graph* g, int startVertex) {
    bool visited[MAX_VERTICES] = {false};
    Queue q;
    initQueue(&q);
    // Mark the start vertex as visited and enqueue it
    visited[startVertex] = true;
    enqueue(&q, startVertex);
    while (!isEmpty(&q)) {
        int currentVertex = dequeue(&q);
        printf("Visited %d\n", currentVertex);
        // Get all adjacent vertices of the dequeued vertex
        for (int i = 0; i < g->vertices; i++) {
            if (g->adjMatrix[currentVertex][i] && !visited[i]) {
                visited[i] = true;
                enqueue(&q, i);
            }
        }
    }
}
int main() {
```

```
    Graph g;
    int vertices = 5;
    initGraph(&g, vertices);
    addEdge(&g, 0, 1);
    addEdge(&g, 0, 4);
    addEdge(&g, 1, 2);
    addEdge(&g, 1, 3);
    addEdge(&g, 2, 4);
    addEdge(&g, 3, 4);
    printf("BFS starting from vertex 0:\n");
    bfs(&g, 0);
    return 0;
}
```

SAMPLE OUTPUT:

```
BFS starting from vertex 0:
Visited 0
Visited 1
Visited 4
Visited 2
Visited 3
```

DEPTH FIRST SEARCH(DFS):

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

#define MAX_VERTICES 100

// Stack structure for DFS
typedef struct Stack {
    int items[MAX_VERTICES];
    int top;
} Stack;
```

```
// Initialize the stack
void initStack(Stack* s) {
    s->top = -1;
}

// Check if the stack is empty
bool isEmpty(Stack* s) {
    return s->top == -1;
}

// Add an item to the stack
void push(Stack* s, int item) {
    if (s->top == MAX_VERTICES - 1) {
        printf("Stack is full\n");
        return;
    }
    s->items[++(s->top)] = item;
}

// Remove an item from the stack
int pop(Stack* s) {
    if (isEmpty(s)) {
        printf("Stack is empty\n");
        return -1;
    }
    return s->items[(s->top)--];
}

// Graph structure
typedef struct Graph {
    int vertices;
    bool adjMatrix[MAX_VERTICES][MAX_VERTICES];
} Graph;
```

```
// Initialize the graph
void initGraph(Graph* g, int vertices) {
    g->vertices = vertices;
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            g->adjMatrix[i][j] = false;
        }
    }
}

// Add edge to the graph
void addEdge(Graph* g, int start, int end) {
    g->adjMatrix[start][end] = true;
    g->adjMatrix[end][start] = true; // For undirected graph
}

// Perform DFS on the graph
void dfs(Graph* g, int startVertex) {
    bool visited[MAX_VERTICES] = {false};
    Stack s;
    initStack(&s);

    // Start DFS
    push(&s, startVertex);

    while (!isEmpty(&s)) {
        int currentVertex = pop(&s);

        if (!visited[currentVertex]) {
            printf("Visited %d\n", currentVertex);
            visited[currentVertex] = true;

            // Push all unvisited adjacent vertices to the stack
```

```
            for (int i = 0; i < g->vertices; i++) {
                if (g->adjMatrix[currentVertex][i] && !visited[i]) {
                    push(&s, i);
                }
            }
        }
    }
}

int main() {
    Graph g;
    int vertices = 5;
    initGraph(&g, vertices);

    addEdge(&g, 0, 1);
    addEdge(&g, 0, 4);
    addEdge(&g, 1, 2);
    addEdge(&g, 1, 3);
    addEdge(&g, 2, 4);
    addEdge(&g, 3, 4);

    printf("DFS starting from vertex 0:\n");
    dfs(&g, 0);

    return 0;
}
```

SAMPLE OUTPUT:

```
DFS starting from vertex 0:
Visited 0
Visited 4
Visited 3
Visited 1
Visited 2
```