Breadth first search:

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX_VERTICES 100

typedef struct Node {

    int vertex;

    struct Node* next;

} Node;

typedef struct Queue {

    int items[MAX_VERTICES];

    int front, rear;

} Queue;

Node* createNode(int vertex);

Queue* createQueue();

bool isQueueEmpty(Queue* q);

void enqueue(Queue* q, int value);

int dequeue(Queue* q);

void bfs(int graph[MAX_VERTICES][MAX_VERTICES], int startVertex, int numVertices);


int main() {

    int numVertices = 5;

    int graph[MAX_VERTICES][MAX_VERTICES] = {

        {0, 1, 1, 0, 0},

        {1, 0, 1, 1, 0},

        {1, 1, 0, 1, 0},

        {0, 1, 1, 0, 1},

        {0, 0, 0, 1, 0}

    };

    int startVertex = 0;

    bfs(graph, startVertex, numVertices);


    return 0;

}

Node* createNode(int vertex) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->vertex = vertex;
    newNode->next = NULL;
    return newNode;
}
Queue* createQueue() {
    Queue* q = (Queue*)malloc(sizeof(Queue));
    q->front = 0;
    q->rear = -1;
    return q;
}
bool isQueueEmpty(Queue* q) {
    return q->rear < q->front;
}

void enqueue(Queue* q, int value) {
    if (q->rear == MAX_VERTICES - 1) {
        printf("Queue is full\n");
        return;
    }
    q->items[++(q->rear)] = value;
}
int dequeue(Queue* q) {
    if (isQueueEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    return q->items[(q->front)++];
}
void bfs(int graph[MAX_VERTICES][MAX_VERTICES], int startVertex, int numVertices) {
    bool visited[MAX_VERTICES] = {false};
    Queue* q = createQueue();
    visited[startVertex] = true;
    enqueue(q, startVertex);
    printf("Breadth-First Search starting from vertex %d:\n", startVertex);
    while (!isQueueEmpty(q)) {
        int currentVertex = dequeue(q);
        printf("%d ", currentVertex);
        for (int i = 0; i < numVertices; i++) {
            if (graph[currentVertex][i] == 1 && !visited[i]) {
                visited[i] = true;
                enqueue(q, i);
            }
        }
    }
    printf("\n");
    free(q);
}
```

Output:

```
Breadth-First Search starting from vertex 0:
0 1 2 3 4
```

Depth first search:

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX_VERTICES 100
typedef struct Stack {
    int items[MAX_VERTICES];
    int top;
} Stack;
Stack* createStack();
bool isStackEmpty(Stack* s);
void push(Stack* s, int value);
int pop(Stack* s);
void dfs(int graph[MAX_VERTICES][MAX_VERTICES], int startVertex, int numVertices);

int main() {
    int numVertices = 5;
    int graph[MAX_VERTICES][MAX_VERTICES] = {
        {0, 1, 1, 0, 0},
        {1, 0, 1, 1, 0},
        {1, 1, 0, 1, 0},
        {0, 1, 1, 0, 1},
        {0, 0, 0, 1, 0}
    };
    int startVertex = 0;
    dfs(graph, startVertex, numVertices);
    return 0;
}
Stack* createStack() {
    Stack* s = (Stack*)malloc(sizeof(Stack));
    s->top = -1;
    return s;
}
bool isStackEmpty(Stack* s) {
    return s->top == -1;
}
void push(Stack* s, int value) {
    if (s->top == MAX_VERTICES - 1) {
        printf("Stack overflow\n");
        return;
    }
    s->items[++(s->top)] = value;
}
int pop(Stack* s) {
    if (isStackEmpty(s)) {
        printf("Stack underflow\n");
        return -1;
    }
    return s->items[(s->top)--];
}
void dfs(int graph[MAX_VERTICES][MAX_VERTICES], int startVertex, int numVertices) {
    bool visited[MAX_VERTICES] = {false};
    Stack* s = createStack();
    push(s, startVertex);
    printf("Depth-First Search starting from vertex %d:\n", startVertex);

    while (!isStackEmpty(s)) {
        int currentVertex = pop(s);
        if (!visited[currentVertex]) {
            visited[currentVertex] = true;
            printf("%d ", currentVertex);
            for (int i = numVertices - 1; i >= 0; i--) {
                if (graph[currentVertex][i] == 1 && !visited[i]) {
                    push(s, i);
                }
            }
        }
    }
    printf("\n");
    free(s);
}
```

Output:

```
Depth-First Search starting from vertex 0:
0 1 2 3 4
```