DATA STRUCTURE

1.Breadth first search

PROGRAM:

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

typedef struct Queue {

   int front, rear, size;

   unsigned capacity;

   int* array;

} Queue;

Queue* createQueue(unsigned capacity) {

   Queue* queue = (Queue*)malloc(sizeof(Queue));

   queue->capacity = capacity;

   queue->front = queue->size = 0;

   queue->rear = capacity - 1;

   queue->array = (int*)malloc(queue->capacity * sizeof(int));

   return queue;

}

 bool isFull(Queue* queue) {

   return (queue->size == queue->capacity);

}

bool isEmpty(Queue* queue) {

   return (queue->size == 0);

}

 void enqueue(Queue* queue, int item) {

   if (isFull(queue)) return;

   queue->rear = (queue->rear + 1) % queue->capacity;
```

```
    queue->array[queue->rear] = item;
    queue->size = queue->size + 1;
}
 int dequeue(Queue* queue) {
    if (isEmpty(queue)) return -1;
    int item = queue->array[queue->front];
    queue->front = (queue->front + 1) % queue->capacity;
    queue->size = queue->size - 1;
    return item;
}
 int front(Queue* queue) {
    if (isEmpty(queue)) return -1;
    return queue->array[queue->front];
}
 int rear(Queue* queue) {
    if (isEmpty(queue)) return -1;
    return queue->array[queue->rear];
}
 void BFS(int** adjMatrix, int numVertices, int startVertex) {
    bool* visited = (bool*)malloc(numVertices * sizeof(bool));
    for (int i = 0; i < numVertices; i++)
        visited[i] = false;
    Queue* queue = createQueue(numVertices);
    visited[startVertex] = true;
    enqueue(queue, startVertex);
    while (!isEmpty(queue)) {
        int currentVertex = dequeue(queue);
        printf("Visited %d\n", currentVertex);
        for (int i = 0; i < numVertices; i++) {
            if (adjMatrix[currentVertex][i] && !visited[i]) {
                visited[i] = true;
                enqueue(queue, i);
            }
        }
    }
    free(visited);
    free(queue->array);
    free(queue);
}
int main() {
    int numVertices = 5;
     int** adjMatrix = (int**)malloc(numVertices * sizeof(int*));
    for (int i = 0; i < numVertices; i++) {
        adjMatrix[i] = (int*)malloc(numVertices * sizeof(int));
        for (int j = 0; j < numVertices; j++) {
            adjMatrix[i][j] = 0;
        }
    }
     adjMatrix[0][1] = adjMatrix[1][0] = 1;
    adjMatrix[0][2] = adjMatrix[2][0] = 1;
    adjMatrix[1][3] = adjMatrix[3][1] = 1;
    adjMatrix[2][3] = adjMatrix[3][2] = 1;
    adjMatrix[3][4] = adjMatrix[4][3] = 1;
    printf("Breadth First Search starting from vertex 0:\n");
    BFS(adjMatrix, numVertices, 0);
     for (int i = 0; i < numVertices; i++) {
        free(adjMatrix[i]);
    }
    free(adjMatrix);
    return 0;
}
```

OUTPUT:

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

2.depth first search

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
typedef struct Graph {
    int numVertices;
    int** adjMatrix;
} Graph;
 Graph* createGraph(int numVertices) {
    Graph* graph = (Graph*)malloc(sizeof(Graph));
    graph->numVertices = numVertices;
    graph->adjMatrix = (int**)malloc(numVertices * sizeof(int*));
    for (int i = 0; i < numVertices; i++) {
        graph->adjMatrix[i] = (int*)malloc(numVertices * sizeof(int));
        for (int j = 0; j < numVertices; j++) {
            graph->adjMatrix[i][j] = 0; // Initialize all edges as 0
        }
    }
    return graph;
}
 void addEdge(Graph* graph, int src, int dest) {
    graph->adjMatrix[src][dest] = 1;
    graph->adjMatrix[dest][src] = 1; // Because the graph is undirected
}
void DFSUtil(Graph* graph, int vertex, bool* visited) {
    visited[vertex] = true;
    printf("Visited %d\n", vertex);
    for (int i = 0; i < graph->numVertices; i++) {
        if (graph->adjMatrix[vertex][i] == 1 && !visited[i]) {
            DFSUtil(graph, i, visited);
        }
    }
}
void DFS(Graph* graph, int startVertex) {
    bool* visited = (bool*)malloc(graph->numVertices * sizeof(bool));
    for (int i = 0; i < graph->numVertices; i++)
        visited[i] = false;
    DFSUtil(graph, startVertex, visited);
    free(visited);
}
int main() {
    int numVertices = 5;
    Graph* graph = createGraph(numVertices);
     addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 4);
    printf("Depth First Search starting from vertex 0:\n");
    DFS(graph, 0);
     for (int i = 0; i < numVertices; i++) {
        free(graph->adjMatrix[i]);
    }
    free(graph->adjMatrix);
    free(graph);
    return 0;
}
```

OUTPUT:

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