9a)Breadth-First Search (BFS) traversal of a graph

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>


#define MAX 10


struct Vertex {

   char label;

   bool visited;

};


// queue variables

int queue[MAX];

int rear = -1;

int front = 0;

int queueItemCount = 0;


// graph variables

// array of vertices

struct Vertex* lstVertices[MAX];

// adjacency matrix

int adjMatrix[MAX][MAX];

// vertex count

int vertexCount = 0;
```

```
// queue functions
void insert(int data) {
   queue[++rear] = data;
   queueItemCount++;
}

int removeData() {
   queueItemCount--;
   return queue[front++];
}

bool isQueueEmpty() {
   return queueItemCount == 0;
}

// graph functions
// add vertex to the vertex list
void addVertex(char label) {
   struct Vertex* vertex = (struct Vertex*)malloc(sizeof(struct Vertex));
   vertex->label = label;
   vertex->visited = false;
   lstVertices[vertexCount++] = vertex;
}

// add edge to edge array
void addEdge(int start, int end) {
   adjMatrix[start][end] = 1;
   adjMatrix[end][start] = 1;
```

```
}

// display the vertex
void displayVertex(int vertexIndex) {
   printf("%c ", lstVertices[vertexIndex]->label);
}

// get the adjacent unvisited vertex
int getAdjUnvisitedVertex(int vertexIndex) {
   int i;
   for (i = 0; i < vertexCount; i++) {
      if (adjMatrix[vertexIndex][i] == 1 && lstVertices[i]->visited == false) {
         return i;
      }
   }
   return -1;
}

void breadthFirstSearch() {
   int i;
   // mark the first node as visited
   lstVertices[0]->visited = true;
   // display the vertex
   displayVertex(0);
   // insert vertex index in the queue
   insert(0);
   int unvisitedVertex;
   while (!isQueueEmpty()) {
        // get the unvisited vertex of the vertex which is at the front of the queue
        int tempVertex = removeData();
        // no adjacent vertex found
        while ((unvisitedVertex = getAdjUnvisitedVertex(tempVertex)) != -1) {
            lstVertices[unvisitedVertex]->visited = true;
            displayVertex(unvisitedVertex);
            insert(unvisitedVertex);
        }
    }
    // queue is empty, search is complete, reset the visited flag
    for (i = 0; i < vertexCount; i++) {
        lstVertices[i]->visited = false;
    }
}

int main() {
    int i, j;
    int edges;

    // set adjacency matrix to 0
    for (i = 0; i < MAX; i++) {
        for (j = 0; j < MAX; j++) {
            adjMatrix[i][j] = 0;
        }
    }

    printf("Enter the number of vertices (max %d): ", MAX);
    int numVertices;
    scanf("%d", &numVertices);

    for (i = 0; i < numVertices; i++) {
        char label;
        printf("Enter label for vertex %d: ", i);
        scanf(" %c", &label);
        addVertex(label);
    }

    printf("Enter the number of edges: ");
    scanf("%d", &edges);

    for (i = 0; i < edges; i++) {
        int start, end;
        printf("Enter edge (start end): ");
        scanf("%d %d", &start, &end);
        addEdge(start, end);
    }

    printf("\nBreadth-First Search: ");
    breadthFirstSearch();

    return 0;
}
```

Output

```
Enter the number of vertices (max 10): 5

Enter label for vertex 0: v

Enter label for vertex 1: f

Enter label for vertex 2: c

Enter label for vertex 3: d

Enter label for vertex 4: e

Enter the number of edges: 6

Enter edge (start end): 0 1

Enter edge (start end): 0 2

Enter edge (start end): 0 3

Enter edge (start end): 1 4

Enter edge (start end): 2 4

Enter edge (start end): 3 4

Breadth-First Search: v f c d e
```

b)Depth-First Search (DFS) traversal of a graph

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>


#define MAX 10


struct Vertex {

   char label;

   bool visited;

};


struct Vertex* lstVertices[MAX];

int adjMatrix[MAX][MAX];

int vertexCount = 0;


void addVertex(char label) {

   struct Vertex* vertex = (struct Vertex*)malloc(sizeof(struct Vertex));

   vertex->label = label;

   vertex->visited = false;

   lstVertices[vertexCount++] = vertex;

}


void addEdge(int start, int end) {

   adjMatrix[start][end] = 1;

   adjMatrix[end][start] = 1;
```

```
}

void displayVertex(int vertexIndex) {
    printf("%c ", lstVertices[vertexIndex]->label);
}

void depthFirstSearch(int vertexIndex) {
    int i;
    displayVertex(vertexIndex);
    lstVertices[vertexIndex]->visited = true;
    for (i = 0; i < vertexCount; i++) {
        if (adjMatrix[vertexIndex][i] == 1 && lstVertices[i]->visited == false) {
            depthFirstSearch(i);
        }
    }
}

int main() {
    int i, j;
    int edges;

    for (i = 0; i < MAX; i++) {
        for (j = 0; j < MAX; j++) {
            adjMatrix[i][j] = 0;
        }
    }

    printf("Enter the number of vertices (max %d): ", MAX);
    int numVertices;
    scanf("%d", &numVertices);

    for (i = 0; i < numVertices; i++) {
        char label;
        printf("Enter label for vertex %d: ", i);
        scanf(" %c", &label);
        addVertex(label);
    }
    printf("Enter the number of edges: ");
    scanf("%d", &edges);

    for (i = 0; i < edges; i++) {
        int start, end;
        printf("Enter edge (start end): ");
        scanf("%d %d", &start, &end);
        addEdge(start, end);
    }
    printf("\nDepth-First Search: ");
    for (i = 0; i < vertexCount; i++) {
        if (lstVertices[i]->visited == false) {
            depthFirstSearch(i);
        }
    }
    return 0;
}
```

Output

```
Enter the number of vertices (max 10): 5
Enter label for vertex 0: d

Enter label for vertex 1: a

Enter label for vertex 2: b

Enter label for vertex 3: r

Enter label for vertex 4: e

Enter the number of edges: 6

Enter edge (start end): 0 1

Enter edge (start end): 0 2

Enter edge (start end): 0 3

Enter edge (start end): 1 4

Enter edge (start end): 2 4

Enter edge (start end): 3 4

Depth-First Search: d a e b r
```