COMP 1123
Lab 11

Part 1. Queue ADT

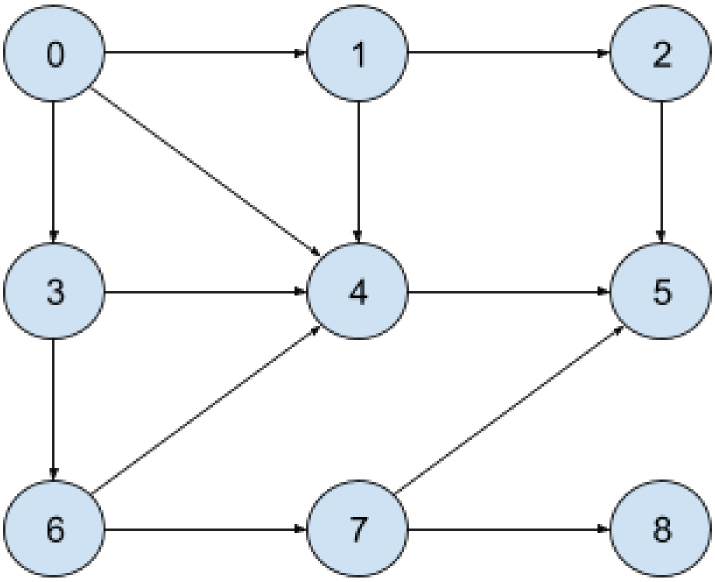
Implement a queue ADT using the below type definitions and prototypes. Write a main function to test your implementation.

```
typedef struct LINKED_QUEUE_NODE_s *LINKED_QUEUE_NODE;
typedef struct LINKED_QUEUE_NODE_s{
LINKED_QUEUE_NODE next;
    void *data;
} LINKED_QUEUE_NODE_t[1];

typedef struct LINKED_QUEUE_s *LINKED_QUEUE;
typedef struct LINKED_QUEUE_s{
    LINKED_QUEUE_NODE head;
    LINKED_QUEUE_NODE tail;
} LINKED_QUEUE_t[1];

LINKED_QUEUE linked_queue_init();
void linked_queue_free(LINKED_QUEUE queue);
int linked_queue_is_empty(LINKED_QUEUE queue);
int linked_queue_size(LINKED_QUEUE queue);
void linked_queue_enqueue(LINKED_QUEUE queue, void *data);
void *linked_queue_dequeue(LINKED_QUEUE queue);

// breadth first search
void bfs(int **adjMatrix, int numberOfVertex, int startVertex);
```

Part 2. A Queue Application: Breadth-First Traversal in Graphs

A graph is a data structure consisting of vertices (or nodes) and edges (or arcs) that connect any two vertices. Breadth-first traversal is a way to visit all the vertices in the graphs in a particular order starting from a given vertex. In this part, you are to implement the breadth-first traversal algorithm for graphs represented as matrices.

An example graph is given below.



In this graph, each circle is a vertex labeled by a nonnegative integer value and the arrows are edges which connect pairs of vertices. If there is an edge (a,b) between two vertices a and b, you can go from vertex a to vertex b but not vice versa. For example edge (7,5) means that you can go from vertex 7 to vertex 5 but you cannot go from 5 to 7 since there is no edge (5,7).

We can represent this graph in the form of a matrix as follows.

```
int N = 9;
int adjMatrix[N][N] = {
  // 0  1  2  3  4  5  6  7  8
    {0, 1, 0, 1, 1, 0, 0, 0, 0}, //0
    {0, 0, 1, 0, 1, 0, 0, 0, 0}, //1
    {0, 0, 0, 0, 0, 1, 0, 0, 0}, //2
    {0, 0, 0, 0, 1, 0, 1, 0, 0}, //3
    {0, 0, 0, 0, 0, 1, 0, 0, 0}, //4
    {0, 0, 0, 0, 0, 0, 0, 0, 0}, //5
    {0, 0, 0, 0, 1, 0, 0, 1, 0}, //6
    {0, 0, 0, 0, 0, 1, 0, 0, 1}, //7
    {0, 0, 0, 0, 0, 0, 0, 0, 0}, //8
    };
```

In the above matrix, there is an edge from vertex `i` to vertex `j` if `adjMatrix[i][j] = 1`, and no edge otherwise.

An algorithm for breadth-first traversal is given below.

```
bft(g, s) //g is a graph, s is start vertex
    q <- create a queue
    dmap <- create a map of values for the vertices in g where the
                value associated with each vertex is false
    enqueue s to q
    mark s as discovered in dmap
    while q is not empty do
        v  <-  dequeue from q
        print "v is visited."
        for all edges of the form (v,w) in g do
            if w is not discovered in dmap then
                enqueue w to q
                mark w as discovered in dmap
            endif
        endfor
    endwhile
end
```

In the above algorithm, a queue is used to keep track of discovered vertices and a table to prevent enqueueing vertices which are already discovered.

Examples:
- If the graph is breadth-first traversed starting from vertex 0, the vertices are visited in the following order: 0, 1, 3, 4, 2, 6, 5, 7, 8.
- If the graph is breadth-first traversed starting from vertex 1, the vertices are visited in the following order: 1, 2, 4, 5.