# ELEC 472 - Artificial Intelligence: Lab 1

## Objectives

This laboratory activity is focused on developing, programming, and visualizing graphs/trees, and performing different type of search.

## Submission and Deliverable

Please submit the following through OnQ:

- ***one report*** (with ***five*** parts) – the report must be created in MS Word and converted to PDF prior to submission,
- ***two*** MATLAB files (named: bfs.m and dls.m)

## 1. Graph Visualization

An important step in understanding an agent and its actions with respect to the environment is visualization of the state space. Any graph can be defined by its set of edges. In order to do so in MATLAB, we need two vectors of ***source*** and ***target*** nodes. You can then make a graph structure by calling the ***graph*** function and visualize it using the ***plot*** function as illustrated. Open MATLAB. Go to New > Script. In the new script, implement the code below, save, and run it:

```
clc       % clear screen
clear all % clear variables

source  = [1, 1, 1, 2, 2, 2, 2, 8];                          % Source Nodes
target  = [3, 4, 2, 6, 5, 7, 8, 9];                          % Target Nodes
names   = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I'};     % Node Names
weights = [200, 300, 900, 400, 0, 200, 100, 50];             % Edge Weights
G1      = graph(source,target,weights,names);

figure
plot(G1, 'EdgeLabel', G1.Edges.Weight)
```

Have a look at the figure. Can you see the relationship between the code and the figure? Note that, for example, node 1 (‘A’) is connected to note 3 (‘C’) with an edge of weight 200. As another example, node 2 (‘B’) is connected to node 6 (‘F’) with an edge of weight 400.

***Report - Part 1:*** Save the figure produced by this graph and include it in the ***Report***.

***Report - Part 2:*** Change the *name* of Node 7 to your own first name, and the weight of the edge from node 2 to that node to the last four digits of your student number. Then re-plot the figure. Include this new figure in the ***Report***.

## 2. Implementing Breadth First Search

In this section we will implement the BFS algorithm. Our goal is to develop the function: nodeList = bfs(source, target, startNode, targetNode), which takes the ***source*** and ***target*** nodes to create the graph, and starts from ***startNode*** to find the ***targetNode***. The output of the

function should be a list of nodes that the algorithm checks until it reaches the goal state. Here is the solution – carefully analyze and understand this code, and re-implement it. First, create a new file named bfs.m. Then, implement the following code in that file. This is your function.

```
function nodeList = bfs(source, target, startNode, targetNode)

% Initialize visited list, queue and nodeList
nodeList  = [];
visited   = [];
queue     = [];

% Set starting node as current node and add it to visited list
visited(end+1) = startNode;
queue(end+1)   = startNode;
iterations     = 0;

while ~isempty(queue)
    % Pop the first item from queue
    currentNode = queue(1);
    queue(1)    = [];    % This is how you pop the first element of a vector

    % check the current node and add it to visited list
    iterations      = iterations + 1;
    nodeList(end+1) = currentNode;
    if currentNode == targetNode
        return
    end
    visited(end+1) = currentNode;

% Get all the children of the current node and add them to the queue if not
visited
    children = getChildren(source, target, currentNode);
    for i = 1:numel(children)
        if ~any(visited==children (i))
            queue(end+1) = children(i);
        end
    end
end

nodeList = -1;
disp('Target not found!')
end

% Helper function to get the children of a node
function children = getChildren(source, target, node)
    children = target(source == node);
end
```

Now, in the command window, test the algorithm using:

```
source = [1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 5, 15, 15, 15];
target = [3, 5, 4, 2, 6, 10, 7, 9, 8, 14, 11, 12, 13, 15, 17, 16, 18];
nodeList = bfs(source, target, 1, 15);
```

Here' we are basically created a new graph and attempted to search for node 15, starting from noded 1, using our 'bfs' function.

***Report - Part 3:*** Save the output of the test above in the ***Report***. To display the output, you can (*i*) remove the semi-colon where you called the function bfs; or alternatively (*ii*) just enter the variable name that you are interested in (**`nodeList`**), in the Command Window; or alternatively (*iii*) go to the Workspace and see the variable that you are interested in (**`nodeList`**).
A screenshot of either option would suffice.

***Report - Part 4:*** Plot the graph and include the figure in the ***Report***. Have a look at the figure and double-check to see if the function has worked as expected.
To plot this graph, you can use the following script:

```
G2 = graph(source, target);
figure
plot(G2)
```

***File 1:*** Submit the function ***bfs.m*** as a **file**.

## 3. Iterative Deepening Search

In this section, we will implement iterative deepening depth first search. First, we will implement a depth-limited search function: nodeList = dls(source, target, startNode, targetNode, depth), which takes inputs similar to the bfs function and outputs nodeList (a list of nodes in the order that the algorithm searches). The input parameter ***depth*** determined the maximum depth that the algorithm explores. The function uses a recursive component to find the goal state. Here is the solution – carefully analyze and understand this code, and re-implement it:

```
function [nodeList, result] = dls(source, target, startNode, targetNode,
depth)

% Initialize nodeList and result
nodeList = startNode;
result   = 0;

% End the recursive algorithm if target has been found or depth limit has
% been reached
if startNode == targetNode
    result = 1;
    return
elseif depth == 0
    return
end


% Get the list of current node's children and recall the recursive
% algorithm on the child. Remember to store the node values in nodeList,
% also remember to set the result to 1 if target is found
```

```
children = getChildren(source, target, startNode);
for i = 1:numel(children)
    [nl, r] = dls(source, target, children(i), targetNode, depth-1);
    nodeList = [nodeList nl];
    if r == 1
        result = 1;
        return
    end
end

end

% Helper function to get the children of a node
function children = getChildren(source,target,node)
    children = target(source==node);
end
```

Now, test the algorithm using:

```
source = [1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 5, 15, 15, 15];
target = [3, 5, 4, 2, 6, 10, 7, 9, 8, 14, 11, 12, 13, 15, 17, 16, 18];
nodeList2 = dls(source, target, 1, 15, 2);  % testing the DLS for L = 2
```

Remember, using a DLS and incrementing the depth creates an IDS. To do this, we can implement the loop inside the DLS function, or alternatively include the DLS function itself in a loop (as shown below):

```
nodeList3 = [];
for L = 0:2
    nodeList3 = [nodeList3, dls(source, target, 1, 15, L)];
end
```

***Report - Part 5:*** Save the outputs *nodeList2* and *nodeList3* in the ***Report***. Double-check to see if the function has worked as expected.

***File 2:*** Submit the function ***dls.m*** as a **file**.