```
function [visitedNodes, stack, timeElapsed] =
 iterative_deepening_search(goalState, mode, prevStack,
 prevVisitedNodes)
% This function realizes Depth First Search algorithm.

% goalState" is a column vector respresenting goal configuration.

% "mode": can take values of 'single_step' or 'complete'
...'single_step': take one step and returns
    ...'complete': tries to solve the puzzle completely.

% "prevStack": is the last snapshot of the stack.

% "prevVisitedNodes": is the last snapshot of the visitedNodes.

% INITIALIZE VARIABLES
maxDepth = 20; % The algorithm will not search beyond this level
numTiles = length(goalState); % Total number of tiles in the puzzle
timeElapsed = 0;


% Determine the minimum depth considering past invesigations
if strcmp(mode, 'complete')
    minDepth = 1;
elseif isempty(prevVisitedNodes)
    minDepth = 1;
else
    minDepth  = max(prevVisitedNodes(numTiles+3, :))+1;
end

% MAIN LOOP
% While incrementing the allowed depth of search, run DFS for each
 iteration
tic;
for iDepth = minDepth:maxDepth
    visitedNodes = prevVisitedNodes; % It will be used to store
 visited nodes
    stack = prevStack;

    % Find the ID number to be assigned
    if isempty(visitedNodes)
        idAssignedLast = max(stack(numTiles+1,:));
    else
        idAssignedLast = max([visitedNodes(numTiles+1, :)
 stack(numTiles+1,:)]);
    end
    idTobeAssigned = idAssignedLast + 1; % Update the id to be
 assigned to the next node

    % Loop until the stack is empty
    iIteration = 0;
    while (~isempty(stack))
```

```
        % If the mode is 'single_step', then stop search after one
    iteration
        if strcmp(mode, 'single_step') && (iIteration == 1)
            return;
        end

        currentNode = stack(:, 1); % currentNode represents the node
    to processed at current iteration

        % Mark currentNode as visited if appropriate
        if iIteration == 0
            visitedNodes = [visitedNodes currentNode];
        else
            [isVisited, loc] = ismember(currentNode(1:numTiles)',
    visitedNodes(1:numTiles, :)', 'rows');
            if isVisited
                % Compare the costs, if we just discovered a node
    version
                % with lower cost
                if visitedNodes(numTiles+3, loc) >
    currentNode(numTiles+3)
                    visitedNodes(:, loc) = currentNode;
                else
                    % We have already discovered this node (with lower
    cost), do nothing.
                end
            else
                visitedNodes = [visitedNodes currentNode];
            end
        end

        % When the algorithm reaches the goal state, return
        if all(currentNode(1:numTiles) == goalState)
            timeElapsed = toc;
            return;
        end

        % Investigate details of the current node
        currentID = currentNode(numTiles+1);
        currentCost = currentNode(numTiles+3);

        unvisitedSuccExists = 0; % A flag that can be modeified
    afterwards

        % If the node is within allowed depth
        if (currentCost < iDepth)
            % Generate the successors of the current node
            successorStates = successors(currentNode(1:numTiles));

            for iSucc = 1: size(successorStates, 2)
                succState = successorStates(:, iSucc);
```

```
                [isSuccVisited, loc] = ismember(succState',
 visitedNodes(1:numTiles, :)', 'rows');

                succCost = currentCost + 1;
                if isSuccVisited
                    if visitedNodes(numTiles+3, loc) > succCost
                        % Do nothing this successor nedds to be
 processed
                        % again
                    else
                        continue;
                    end
                end

                succNode = [succState; idTobeAssigned; currentID;
 succCost] ; % Construct the node
                idTobeAssigned = idTobeAssigned + 1; % Update the id
 to be assigned to the next node

                unvisitedSuccExists = 1; % Update the flag
                break;
            end
        end

        % If there is no unvisited successor
        if ~unvisitedSuccExists
            stack(:, 1) = []; % Pop currentNode out of stack

            % Else push the unvisited successor into the stack
        else
            stack = [succNode stack];
        end
        iIteration = iIteration + 1;
    end
end

if iDepth == maxDepth
    error("The IDDFS algorithm could not find a solution.");
end

end
```

Published with MATLAB® R2017b