```
function [visitedNodes, stack, timeElapsed] =
 depth_first_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
visitedNodes = prevVisitedNodes; % It will be used to store visited
 nodes
stack = prevStack;
numTiles = length(goalState); % Total number of tiles in the puzzle
timeElapsed = 0;

%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


% MAIN LOOP
% Loop until the stack is empty
% Note also that when the goal state is discovered, the loop will be
 terminated (by an if-statement)
tic;
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];
```

```
    elseif ~any(ismember(currentNode(1:numTiles)',
 visitedNodes(1:numTiles, :)', 'rows'))
        visitedNodes = [visitedNodes currentNode];
    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);

    successorStates = successors(currentNode(1:numTiles)); % Generate
 the successors of the current node

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

    % Invesitage any unvisited successors
    for iSucc = 1: size(successorStates, 2)
        iSuccState = successorStates(:, iSucc);

        if any(ismember(iSuccState',
 visitedNodes(1:numTiles, :)', 'rows'))
            continue;
        end

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

        unvisitedSuccExists = 1; % Update the flag
        break;
    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

% Issues an error since the queue is empty and a solution could not be
 obtained
error("The DFS algorithm could not find a solution.");
end
```

Published with MATLAB® R2017b