```
function [visitedNodes, queue, timeElapsed] = a_star_search(goalState,
 mode, prevQueue, prevVisitedNodes, heuristicType)
% This function realizes A* 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.

% "heuristic_type": 'Heuristic Manhattan' or 'Heuristic Misplaced'

% INITIALIZE VARIABLES
visitedNodes = prevVisitedNodes; % It will be used to store visited
 nodes
queue = prevQueue;
numTiles = length(goalState); % Total number of tiles in the puzzle
timeElapsed = 0;

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


% MAIN LOOP
% Loop until the queue 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(queue))

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

    % Dequeue the node with minimum f = g + h (g:Cost, h:Heuristic),
 since
    % the queue is already ordered in an ascending manner:
    currentNode = queue(:,1);
```

```
   queue(:,1) = [];

   % Investigate details of the current node
   curState = currentNode(1:numTiles);
   curID = currentNode(numTiles+1);
   curCost = currentNode(numTiles+3);

   % Insert currentNode in visitedNodes
   % But, first check if it was visited before
   if isempty(visitedNodes)
       visitedNodes = [visitedNodes currentNode];
   else
       [curNodeVisited, curLoc] = ismember(curState',
visitedNodes(1:numTiles, :)', 'rows');
       if curNodeVisited
           visitedNodes(:, curLoc) = currentNode;
       else
           visitedNodes = [visitedNodes currentNode];
       end
   end

   % When the goal state is popped out of queue than finish the
search.
   if (currentNode(1:numTiles) == goalState)
       timeElapsed = toc;
       return;
   end

   % Find successors of the currentNode
   successorStates = successors(currentNode(1:numTiles)); % Find
successors of the parent

   for iSuccessor = 1:size(successorStates,2)
       iSuccState = successorStates(:, iSuccessor);

       % Calculate the heuristic for the successor
       switch heuristicType
           case 'Heuristic Manhattan'
               iSuccHeuristic = heuristic_manhattan(iSuccState,
goalState);
           case 'Heuristic Misplaced'
               iSuccHeuristic = heuristic_misplaced(iSuccState,
goalState);
       end

       % Cost is calculated by incrementing parent cost by 1.
       iSuccCost = curCost + 1;
       iSuccF = iSuccCost + iSuccHeuristic;

       % Build the successor node by obtained properties
       iSuccNode = [iSuccState; idTobeAssigned; curID; iSuccCost;
iSuccHeuristic; iSuccF];
       idTobeAssigned = idTobeAssigned + 1;
```

```
        % Check if the successor has already been labelled, if yes
 compare the costs
        [isSuccVisited, locVisited] = ismember(iSuccState',
 visitedNodes(1:numTiles, :)', 'rows');

        % Check if the successor is already in the queue
        [isSuccQueued, locQueue] = ismember(iSuccState',
 queue(1:numTiles, :)', 'rows');

        % If previous cost (in visited nodes) was larger than put this
 successor in the queue
        if isSuccVisited
            oldCost = visitedNodes(numTiles+3, locVisited);
            if oldCost > iSuccCost
                queue = [queue iSuccNode];
            end
        end

        % If previous f (in queue) was larger than overwrite it with
 this successor
        if isSuccQueued
            oldF = queue(numTiles+5, locQueue);
            if oldF > iSuccF
                queue(:, locQueue) = iSuccNode;
            end
        end

        % If this is the first time we come across with this
 successor, insert it into
        ... the queue.
            if ~isSuccQueued && ~isSuccVisited
            queue = [queue iSuccNode];
            end
    end

    % Sort queue with respect to f
    queue = sortrows(queue', numTiles+5)';

    iIteration = iIteration + 1;
end

% Issue an error, since the queue is empty and the algorith could not
 find a solution.
error("The A* algorithm could not find a solution.");

end
```

Published with MATLAB® R2017b