```
function [route,numExpanded] = DijkstraGrid (input_map, start_coords, dest_coords)

% Run Dijkstra's algorithm on a grid.

% Inputs :

%   input_map : a logical array where the freespace cells are false or 0 and

%   the obstacles are true or 1

%   start_coords and dest_coords : Coordinates of the start and end cell

%   respectively, the first entry is the row and the second the column.

% Output :

%    route : An array containing the linear indices of the cells along the

%    shortest route from start to dest or an empty array if there is no

%    route. This is a single dimensional vector

%    numExpanded: Remember to also return the total number of nodes

%    expanded during your search. Do not count the goal node as an expanded node.


% set up color map for display

% 1 - white - clear cell

% 2 - black - obstacle

% 3 - red = visited

% 4 - blue  - on list

% 5 - green - start

% 6 - yellow - destination


cmap = [1 1 1; ...

        0 0 0; ...

        1 0 0; ...
```

```
        0 0 1; ...
        0 1 0; ...
        1 1 0; ...
            0.5 0.5 0.5];

colormap(cmap);

% variable to control if the map is being visualized on every
% iteration
drawMapEveryTime = true;

[nrows, ncols] = size(input_map);

% map - a table that keeps track of the state of each grid cell
map = zeros(nrows,ncols);

map(~input_map) = 1;   % Mark free cells
map(input_map)  = 2;   % Mark obstacle cells

% Generate linear indices of start and dest nodes
start_node = sub2ind(size(map), start_coords(1), start_coords(2));
dest_node  = sub2ind(size(map), dest_coords(1),  dest_coords(2));

map(start_node) = 5;
map(dest_node)  = 6;
```

```
% Initialize distance array
distanceFromStart = Inf(nrows,ncols);

% For each grid cell this array holds the index of its parent
parent = zeros(nrows,ncols);

distanceFromStart(start_node) = 0;

% keep track of number of nodes expanded
numExpanded = 0;

% Main Loop
while true

  % Draw current map
  map(start_node) = 5;
  map(dest_node) = 6;

  % make drawMapEveryTime = true if you want to see how the
  % nodes are expanded on the grid.
  if (drawMapEveryTime)
    image(1.5, 1.5, map);
    grid on;
    axis image;
```

```
    drawnow;
end

% Find the node with the minimum distance
[min_dist, current] = min(distanceFromStart(:));

if ((current == dest_node) || isinf(min_dist))
    break;
end;

% Update map
%  map(current) = 3;        % mark current node as visited
% distanceFromStart(current) = Inf; % remove this node from further consideration

% Compute row, column coordinates of current node
[i, j] = ind2sub(size(distanceFromStart), current);

% *********************************************************************
% YOUR CODE BETWEEN THESE LINES OF STARS
%
% Visit each neighbor of the current node and update the map, distances
% and parent tables appropriately.
%
numExpanded = numExpanded + 1;      % collect the "expanded" (red) cells
```

```
    % neighbors exclude "red" cells (already visited) but include "blue" cells
    n = DijNeighbors (map, i, j);


    k = distanceFromStart(n) > (distanceFromStart(current) + 1);    % which cell has higher distance value
    indx = find (k ==1);                                  % indx is the index in n of those cells
    distanceFromStart(n(indx)) = distanceFromStart(current) + 1;    % update distance
    distanceFromStart(current) = Inf;                          % remove this node from further consideration
    map(current) = 3;                                     % mark current node as "expanded" (red)
    parent(n(indx)) = current;
    map(n(indx)) = 4;                                     % blue (4) = on list
    %*******************************************************************

end


%% Construct route from start to dest by following the parent links
if (isinf(distanceFromStart(dest_node)))
    route = [];
else
    route = [dest_node];


    while (parent(route(1)) ~= 0)
        route = [parent(route(1)), route];
    end


        % Snippet of code used to visualize the map and the path
```

```
    for k = 2:length(route) - 1
        map(route(k)) = 7;
        pause(0.1);
        image(1.5, 1.5, map);
        grid on;
        axis image;
    end
end


end


function n = DijNeighbors (G, I, J)
% find the neighbors of cell (I, J) on a 2D grid for Dijkstra's Algorithm
% Input: G - the 2D grid presented as a matrix
%       (I, J) - the coordinate of the cell on the grid, measured from the top-left
%               corner as (1, 1), I = #row, J = #column
% Output: a list of linear positions of the neighbors on the grid
nrows = size(G, 1);
ncols = size(G, 2);
n = [];
i = 1;
if I - 1 > 0 && (G(I-1, J) ~= 2 && G(I-1, J) ~= 3 && G(I-1, J) ~= 5)         % if the upper cell is inside of the grid and it is not an obstacle and not the visited cell and not the start node
    n(i) = sub2ind(size(G), I-1, J);
    i = i+1;
end
```

```
if I + 1 <= nrows && (G(I+1, J) ~= 2 && G(I+1, J) ~= 3 && G(I+1, J) ~= 5)    % if the lower cell is inside the grid and it is not ...

    n(i) = sub2ind(size(G), I+1, J);

    i = i+1;

end

if J - 1 > 0 && (G(I, J-1) ~= 2 && G(I, J-1) ~= 3 && G(I, J-1) ~= 5)

    n(i) = sub2ind(size(G), I, J-1);

    i = i+1;

end

if J + 1 <= ncols && (G(I, J+1) ~= 2 && G(I, J+1) ~= 3 && G(I, J+1) ~= 5)

    n(i) = sub2ind(size(G), I, J+1);

    i = i+1;

end

end
```