```
clear;clc;
```

Chapter 10 - 10.3 Project: A* Search In this programming assignment, you will implement A* search. Given a graph, the start node, and the goal node, your program will search the graph for a minimum-cost path from the start to the goal. Your program will either return a sequence of nodes for a minimum-cost path or indicate that no solution exists.

Data Input:

```
% Populating the 'Nodes' Matrix with the "nodes.csv" file:
    Nodes = readmatrix('nodes.csv');
% Populating the 'Edges' Matrix with the "edges.csv" file:
  % !!! ATTN !!!: As per the assignments instructions, the edge from point
  % 4 to point 7 has been removed from the "edges.csv" file:
    Edges = readmatrix('edges.csv');

% Creating an obstacles matrix from the "obstacles.csv":
    Obstacles = readmatrix('obstacles.csv');
```

Matrix Initialization:

```
% The number of nodes in the "Nodes" Matrix:
    N = size(Nodes,1); % N is also our goal position (i.e. Node #12)

% Creating the Past Cost Matrix:
    Past_Cost = [1,0];
    for i = 1:N-1
        Past_Cost(i+1,:) = [i+1,inf];
    end

% Creating the Heuristic-Cost-To-Go (Optimistic Cost) Matrix:
%   Optimistic_Cost = [1,0];
    for i = 1:N
        Optimistic_Cost(i,:) = [i, Nodes(i,4)];
    end

% Creating the Estimate Total Cost Matrix:
    for i = 1:N
        Est_Total_Cost(i,:) = [i, Past_Cost(i,2) + Optimistic_Cost(i,2)];
    end

%  Creating the 'Open' Matrix; Initializing 'Closed', 'Path', and
% 'Tentative_Past_Cost':
    Open = [1,Est_Total_Cost(1,2)];
    Closed = []; % Creating an empty variable
%    Path = [1]; % The Path will always begin with the start Node #1
    Tentative_Past_Cost = []; % Creating an empty variable
    nbrlist = []; % Creating an empty variable
    Parent = [1, 0]; % There will never be a parent to Node #1
```

A* Search Algorithm:

```
while isempty(Open) == 0  % Run until 'Open' is empty.
        % Note: isempty returns a 1 (true) if the matrix is empyty, and a 0
```

```
        % (False), if the matrix is NOT empty
    Current = Open(1,1);  % The first node in 'Open'
    Open(1,:)=[];  % Deleting the Current Node (1st row) from 'Open'
    Closed(end+1,:) = Current;  % Adding 'Current' to the 'Closed' List
    if Current == N  % N is our goal state (Last Node)
        disp ('PATH SUCCESSFULLY FOUND') % Displays if Current = goal state
        break
    end

  % Finding All Neighbors to 'Current"
    nbrlist_all = Edges( Edges(:,1)==Current | Edges(:,2)==Current);
        % Notes: This is using logical indexing
        %   The "|" symbol means "or"
        %   This is returning the neighboring nodes to 'Current' that are
        %   listed in the 'Edges' matrix.
        %   If any row of column 1 or 2 of 'Edges' contains the 'Current'
        %   node value, then this function will store the neighbor nodes as
        %   a list.
    nbrlist = sort(nbrlist_all(~ismember(nbrlist_all, Closed)));
    % Only keeping neighbors of 'Current' that are NOT in 'Closed',
    % and sorting the list from smallest to largest.

    for i = 1:size(nbrlist,1) % Running for loop for each node in 'nbrlist'
        nbr = nbrlist(i,:);
        % Determining the cost to move from node 'Current' to 'nbr':
        Cost = Edges( Edges(:,1)==nbr & Edges(:,2)==Current, 3);
          % NOTES: this is searching for the row within  the 'Edges' Matrix
          % where the first column contains the value of 'nbr' and the
          % second column contains the value of 'current', and returning
          % the cost value (column 3).
        Tentative_Past_Cost = Past_Cost(Current,2) + Cost;
        if Tentative_Past_Cost <= Past_Cost(nbr,2)
            Past_Cost(nbr,2) = Tentative_Past_Cost;
            Parent(nbr,:) = [nbr, Current];
            % Updating 'Est_Total_Cost' with this information:
            Est_Total_Cost(nbr,2) = Past_Cost(nbr,2) + ...
                Optimistic_Cost(nbr, 2);
            % Adding the estimated cost to the 'Open' Matrix:
            Open(end+1,:) = [nbr, Est_Total_Cost(nbr,2)];
            % Sorting the 'Open' Matrix based on the lowest
            % 'Est_Total_Cost' value (column 2):
            Open = sortrows(Open,2);
        end
    end
end
% End of A* Search Algorithm
```

PATH SUCCESSFULLY FOUND

Building the 'Path' vector using the 'Parent' Matrix, starting with the goal node:

```
  % Initializing the variables:
    Path = 12;
    j = 12;
```

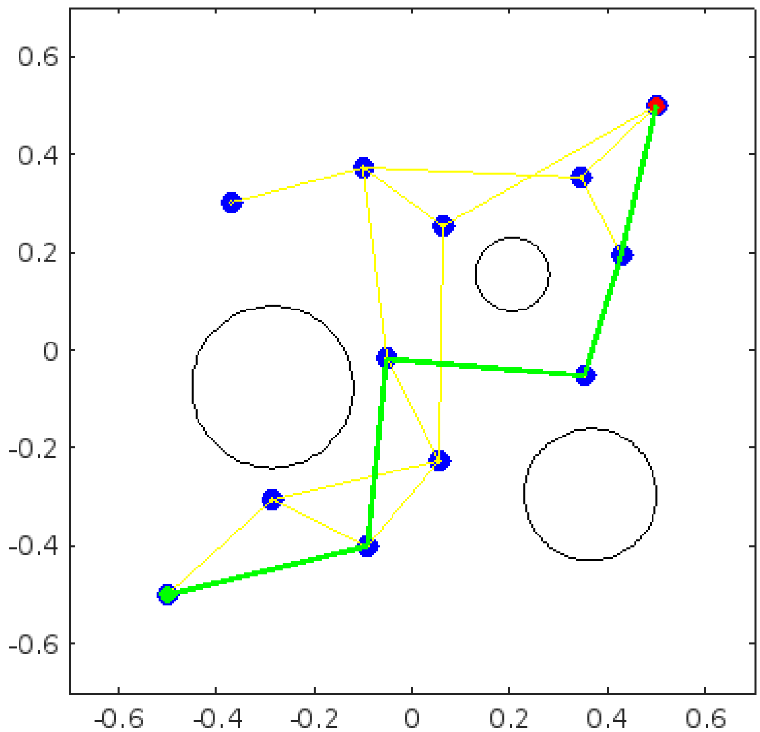
```
    while j > 1
        Path(end+1) = Parent(j,2); %Adding the Parent node to 'Path'
        j = Parent(j,2); % Setting j = current Parent Node
    end

% Reversing the order of 'Path', since it was built from the goal node to
% the start node:
    Path = fliplr(Path)

% Exporting the 'Path' vector as a *.csv file:
    writematrix(Path,'path.csv')
```

```
Path =

     1     2     5     7    10    12
```

NOT REQUIRED: Creating a Plot to Visuallize the Map and Path:

```
  % Plotting the Nodes:
    plot (Nodes(:,2),Nodes(:,3),'bo','MarkerFaceColor','#0000FF',...
        'MarkerSize',8)
    hold on;

  % Plotting the Edges:
    for edge = 1:size(Edges,1)
      n1 = Edges(edge,1);
        x1 = Nodes(n1,2);
        y1 = Nodes(n1,3);
      n2 = Edges(edge,2);
        x2 = Nodes(n2,2);
        y2 = Nodes(n2,3);
      plot([x1 x2],[y1 y2],'yo-','MarkerSize',1)
    end

  % Plotting the Circular Obstacles:
    theta=0:pi/50:2*pi; % Entering Theta values to plot circles
    for obs = 1:size(Obstacles,1) % For each obstacle
        C = Obstacles(obs,1:2); % center of the circular obstacles
        r = Obstacles(obs,3)/2; % The diameter was guve
        plot(C(1)+r*cos(theta),C(2)+r*sin(theta),'k-')
    end

  % Plotting the Calculated Path:
    % Building a new matrix which contains the node number, along with its
    % x and y coordinates:
    PathPlot = transpose(Path);
    for k = 1:size(PathPlot,1)
        x = PathPlot(k);
        PathPlot(k, 2:3) = Nodes(Nodes(:,1)==x, 2:3);
          % Note: This is finding the row within 'Nodes' containing the
          % valuethat matches 'x' and returning columns 2:3.
```

```
    end
    s=size(PathPlot,1);
    % Plotting the Start Node in Green:
        plot(PathPlot(1,2),PathPlot(1,3),'go','MarkerFaceColor','#00FF00')
    % Plotting the End Node in Red:
        plot(PathPlot(s,2),PathPlot(s,3),'ro','MarkerFaceColor','#FF0000')

    % Generating the X & Y values for the start and end points of each path
    % segment:
    for n = 1:size(PathPlot,1)-1
        px1 = PathPlot(n,2); py1 = PathPlot(n,3);
        px2 = PathPlot(n+1,2); py2 = PathPlot(n+1,3);
        plot([px1, px2], [py1, py2],'g','LineWidth',2)
    end

  % Additional Plot Options:
    axis equal; % Making the plot square
    Axis_Extension = 0.2;
    xlim([min(Nodes(:,2))-Axis_Extension,max(Nodes(:,2))+Axis_Extension])
    ylim([min(Nodes(:,3))-Axis_Extension,max(Nodes(:,3))+Axis_Extension])
    %legend ('Nodes','Obstacles')
```



Published with MATLAB® R2023a