# ME145 Robotic Planning and Kinematics: Lab 4

## BFS Algorithm and Trapezoidation

Eric Perez

Spring 2024

May 13, 2024

## computeBFStree

i) Pseudo code from the textbook was used to accomplish the computeBFStree. Below is the pseudo-code used.

**breadth-first search (BFS) algorithm**

**Input:** a graph $G$, a start node $v_{\text{start}}$ and goal node $v_{\text{goal}}$
**Output:** a path from $v_{\text{start}}$ to $v_{\text{goal}}$ if it exists, otherwise a failure notice

```
1: for each node v in G :
2:       parent(v) := NONE
3: parent(v_start) := SELF
4: create an empty queue Q and insert(Q, v_start)
5: while Q is not empty :
6:       v := retrieve(Q)
7:       for each node u connected to v by an edge :
8:             if parent(u) == NONE :
9:                   set parent(u) := v and insert(Q, u)
10:            if u == v_goal :
11:                  run extract-path algorithm to compute the path from start to goal
12:                  return success and the path from start to goal
13: return failure notice along with the parent values.
```

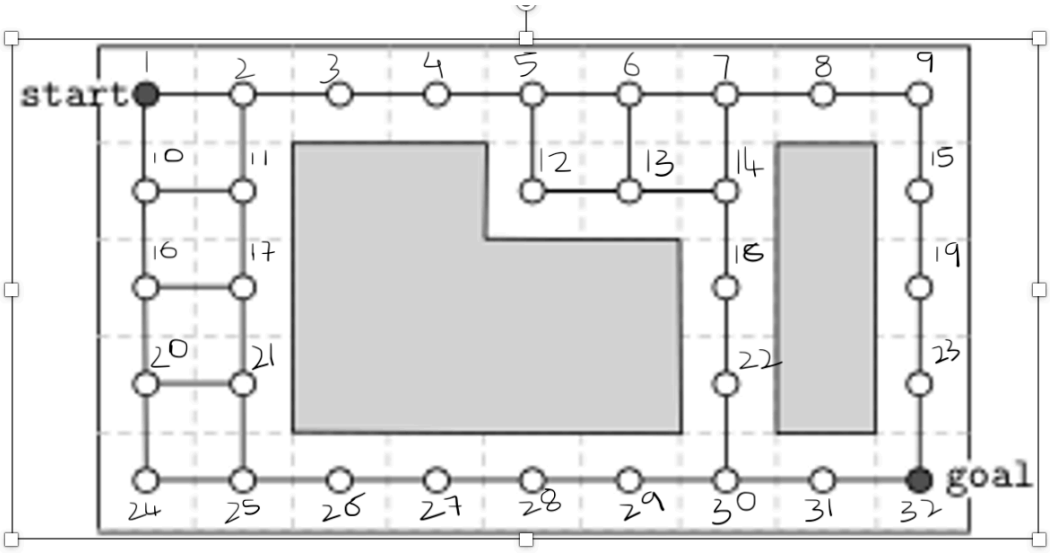
A Q array and parent array were initialized at the start with the Q array containing the starting Node and the parent array containing NANs with the number of columns matching the number of nodes in the workspace. The while loop in the code continues to run as long as the Q array is not empty, the Q array will remove the first value every loop while adding the non-visited adjacent nodes. The parent vector will look at the adjacent node slots and if there is a NAN ( aka not active) it will replace the NAN with the current node which will make that edge active. This process will continue until all possible adjacent edges are active. Below is the code I created for the computeBFS Tree.

ii)

```
% Eric Perez
% ME 145 Lab 4

function [parents] = computeBFStree(AdjTable,start)

    [~,numNodes]= size(AdjTable);

     if start > numNodes
        error('start or goal is not within range of workspace')
     end

    parents = nan(numNodes, 1);

    Q = [start];
    S = [];
    T = [];

    parents(start) = 0;


    while ~isempty(Q)
        currentNode = Q(1);
        Q(1) = [];  % Remove the first value

        adjacentNodes = AdjTable{currentNode};

        [m,n] = size(adjacentNodes);
        for i = 1:n
            node = adjacentNodes(i);

            if isnan(parents(node))
                parents(node) = currentNode;
                Q(end+1) = node;
                S(end+1) = currentNode;
                T(end+1) = node;
            end
        end
    end
G = digraph(S,T);
```

The function also has an error check that makes sure the user's start is within the nodes entered. The function also graphs the BFS tree using the digraph() function. S and T are variables used to form the graph as the variable “S” contains parent nodes to the nodes in array “T”.

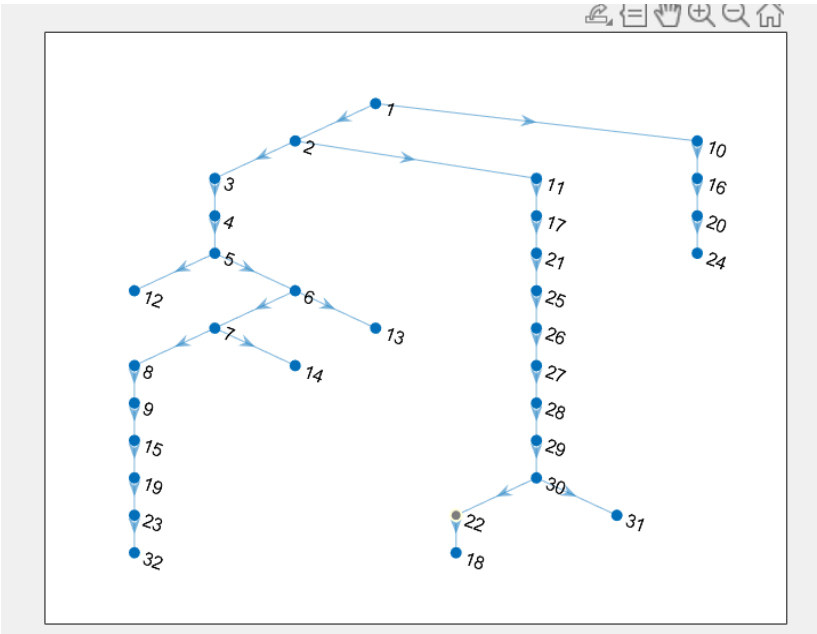
iii) Testing function on the below workspace within respected nodes labeled.



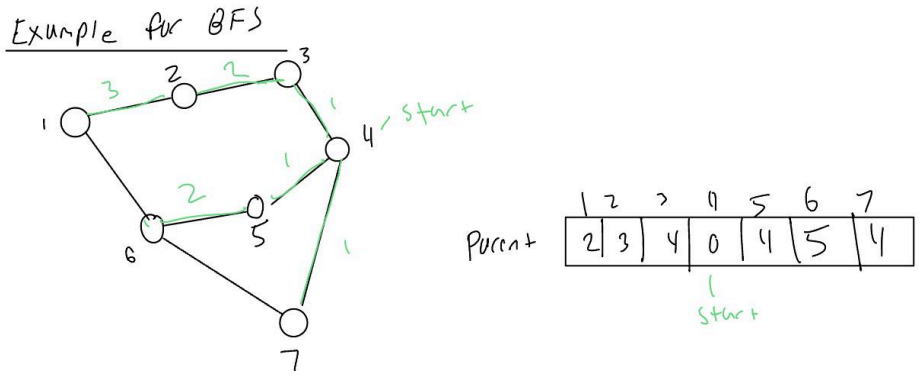

input : [Parents]=computeBFStree({[2 10],[3 11 1],[4 2],[5 3],[6 12 4], [7 13 5],[8 14 6], [9 7],[15 8], [16 11 1],[2 10 17],[13 6 5],[14 6 12],[8 13 7],[19 9],[20 17 10],[21 16 11],[22 14],[23 15],[24 21 16],[25 20 17],[30 18],[32 19],[25 20],[26 24],[27 25],[28 26],[29 27],[30 28],[31 22 29],[32 30],[23 31]},1)

Output:

```
Parents =

  Columns 1 through 18

     0     1     2     3     4     5     6     7     8     1     2     5     6     7     9    10    11    22

  Columns 19 through 32

    15    16    17    30    19    20    21    25    26    27    28    29    30    23
```



Another example used with the respected nodes and edges labeled and starting point being node 4:




Code input:

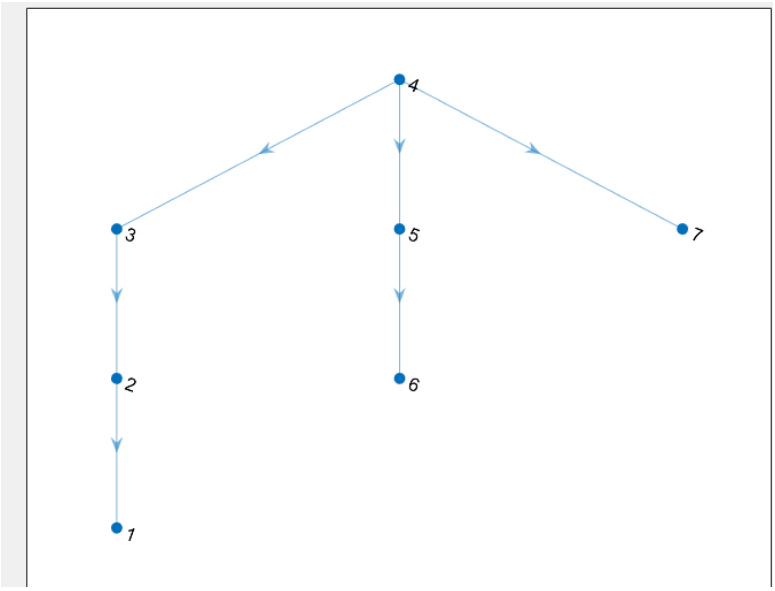
[parents] = computeBFStree({[2,6],[1,3],[2,4],[3 5 7],[6 4], [1 5 7] [4 6]},4)

input : [parents] = computeBFStree({[2,6],[1,3],[2,4],[ 3 5 7],[6 4], [1 5 7] [4 6]},4)

Output:

```
parents =
```

```
     2     3     4     0     4     5     4
```



## computeBFSpath

i) Pseudo code from the textbook was used to accomplish the computeBFSpath by adding this section into the previous code. Below is the pseudo-code used.

**extract-path algorithm**

**Input:** a goal node $v_{\text{goal}}$, and the `parent` values
**Output:** a path from $v_{\text{start}}$ to $v_{\text{goal}}$

```
create an array P := [v_goal]
set u := v_goal
while parent(u) ≠ SELF :
    u := parent(u)
    insert u at the beginning of P
return P
```

The extract-path algorithm essentially grabs the path from the parent vector by traveling backward. Once the goal is reached, the while loop is activated, and the path array will begin to fill until the nodes reach back to the start. This is accomplished by having the current node equal the node that is currently in the current node's parent vector column (the column number corresponds to the node number) and inserting that value into the path array and repeating until the current node is back at the start. This part is added to the previous code to create the computeBFSpath function.

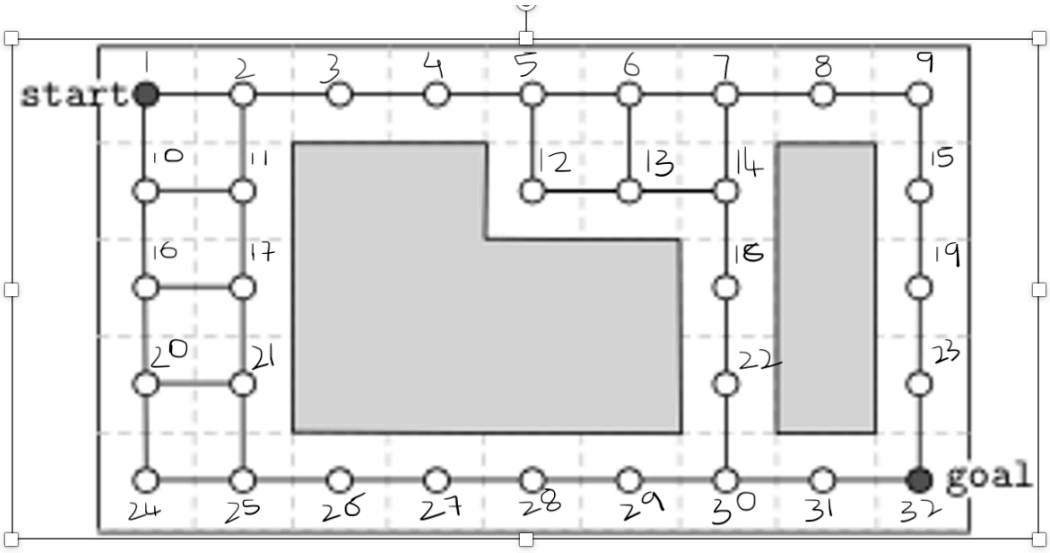
ii)

```
% Eric Perez
% ME 145 Lab 4

function [Path] = computeBFSpath(AdjTable,start,goal)

    [~,numNodes]= size(AdjTable);
    if start > numNodes || goal > numNodes
        error('start or goal is not within range of workspace')
    end
    parents = nan(numNodes, 1);


    Q = [start];
    BFSNodesPlot = []; % nodes of BFS tree
    BFSAdjNodesPlot = []; % Adj nodes

    NodesPlot = []; % nodes for workspace
   AdjNodesPlot = []; % adj nodes

    parents(start) = 0;

    while ~isempty(Q)
        currentNode = Q(1);
        Q(1) = [];  % Remove the first value

        adjacentNodes = AdjTable{currentNode};

        [m,n] = size(adjacentNodes);
        for i = 1:n
            node = adjacentNodes(i);
            AdjNodesPlot(end+1) = node;
             NodesPlot(end+1) = currentNode;
             j = length(AdjNodesPlot);

           for k = 1:j            % looks for duplicate combinations
               if NodesPlot(end) ==  AdjNodesPlot(k)
                NodesPlot(end) = [];
                 AdjNodesPlot(end) = [];
                 weight = [];
                break
               end
           end



            if isnan(parents(node))
                parents(node) = currentNode;
                Q(end+1) = node;
                 BFSNodesPlot(end+1) = currentNode;
                BFSAdjNodesPlot(end+1) = node;

            end
            if node == goal
                Path = [node]
                while parents(node) ~= 0
                    node = parents(node);
                    Path = [node,Path];
                end

            end

        end
    end

BFStreePlot = digraph(BFSNodesPlot,BFSAdjNodesPlot);
WorkSpacePlot = graph(NodesPlot,AdjNodesPlot);

L = plot(BFStreePlot);
figure
h = plot(WorkSpacePlot);
highlight(h,Path,'EdgeColor','g','NodeColor','g')
highlight(L,Path,'EdgeColor','g','NodeColor','g')
labelnode(h,[Path(1) Path(end)],{'start' 'end'})

end
```

The function also has an error check that makes sure the user's start and goal node is within the nodes entered. The function also graphs the BFS tree and the workspace tree using the digraph() function with the path highlighted on both. In order to make my code graph my workspace tree, a section was added that checks if the nodes in the workspace and the adjacent nodes to those nodes had been accounted for in the previous columns of the array.  This was done as the workspace “NodesPlot” and “AdjNodesPlot” arrays would capture two of the same combinations and the graph would repeat lines.

iii)

Testing function on the below workspace within respected nodes labeled.

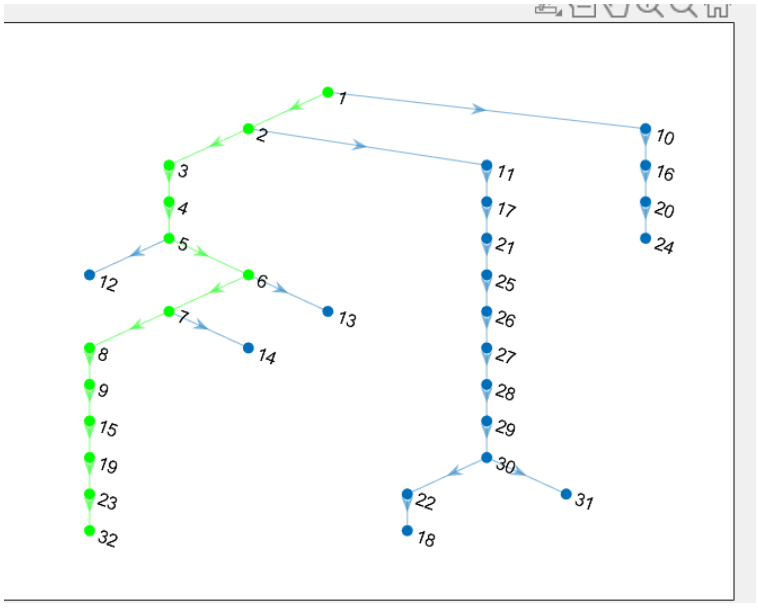



input : [Parents]=computeBFStree({[2 10],[3 11 1],[4 2],[5 3],[6 12 4], [7 13 5],[8 14 6], [9 7],[15 8], [16 11 1],[2 10 17],[13 6 5],[14 6 12],[8 13 7],[19 9],[20 17 10],[21 16 11],[22 14],[23 15],[24 21 16],[25 20 17],[30 18],[32 19],[25 20],[26 24],[27 25],[28 26],[29 27],[30 28],[31 22 29],[32 30],[23 31]},1,32)

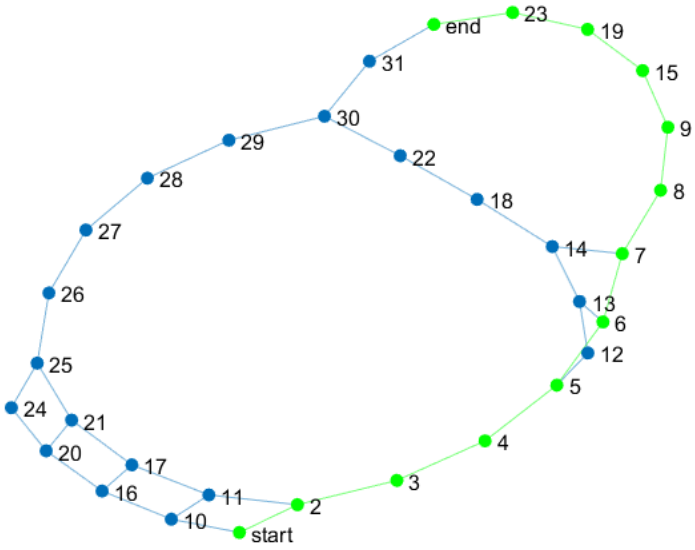
Output:

```
Path =

     1     2     3     4     5     6     7     8     9    15    19    23    32
```

BFS Tree with path highlighted:

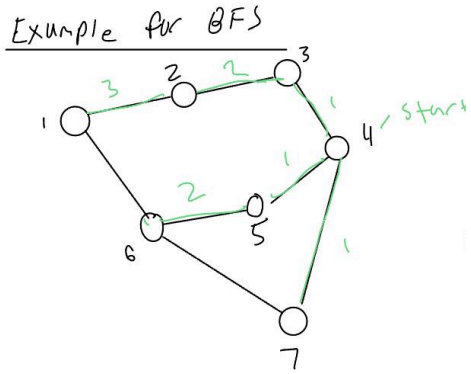


Workspace Tree with path highlighted:




Another example used with the respected nodes labeled and the starting point being

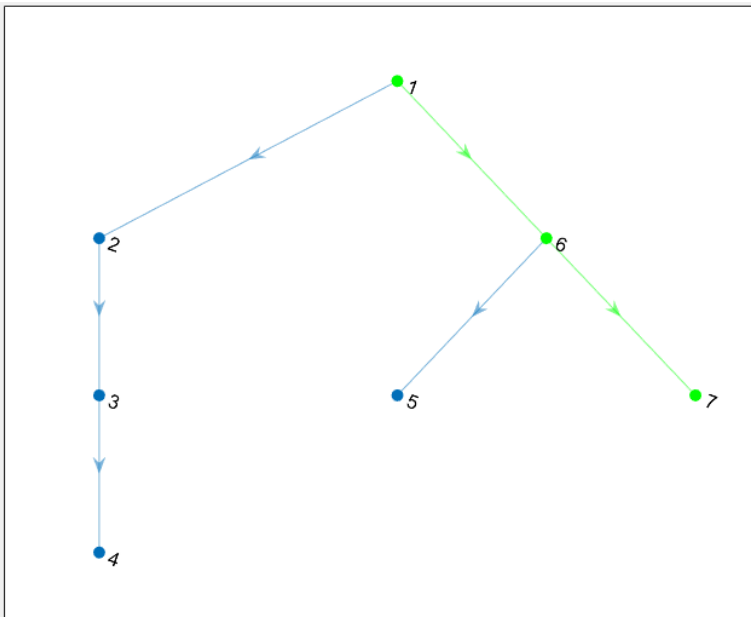
node 1 and goal being node 7:




input : [parents] = computeBFSpath({[2,6],[1,3],[2,4],[ 3 5 7],[6 4], [1 5 7] [4 6]},1,7)
Output:

```
path =

     1     6     7
```

BFS Tree with path highlighted:




Workspace Tree with path highlighted: