Lab 5: Graph Matching, Part 2 (MA438)

This version is heavily revised from the one presented in Week 8. Main changes

- Includes example of making a random simple bipartite graph
- Has fewer bugs
- Seems to actually work
- Has an exercises from Problem Set 3 at the end

Table of Contents

1. Making a "random" bipartite graph

In Lab 4, we learned several ways to construct a graph in MATLAB. In this session, we'll start by constructing our own graph, with a known matching of maximal cardinality.

We'll start by making a graph, F, on 8 vertices, with 4 edges, and 4 components. It has to be bipartite. We'll list the nodes in each part in the vectors P_1 and P_2.

```
clear;
P1 = 1:4; %[1,2,3,4, 9, 10, 12 ];
P2 = 5:8; % [5,6,7,8];
F = graph();
for i=1:length(P1) % assuming P1 and P2 have same length
   F = F.addedge(P1(i),P2(i));
end
plot(F); title("Original graph before edges added")
```

Next, we'll add some more edges. We'll make sure the graph is bipartite, by only adding edges that have a node in each of P_1 and P_2. Note that the resulting graph will always have F as a matching. We'll also ensure the the final graph is connected, and is not a tree

```
rng(1); % make "random" numbers not so random
n = 14; % miniumum number of edges in final graph
G = F;
while ( (G.numedges < n) || (max(G.conncomp) > 1) ...
      || (~G.hascycles))
   s = P1(randi(length(P1))); % randomly chosen element of P1
   t = P2(randi(length(P2))); % randomly chosem element of P2
   G = simplify(G.addedge(s,t)); % simplify removed duplicate edges
```

```
end
g = plot(G);
highlight(g,F);
title('Graph G with matching highlighted')
```

2. The Matching Algorithm: Step 0

We start with an empty matching for G, which we'll call M. That is, Mwill have the same nodes as G, but no edges.

```
M = graph();
M = M.addnode(G.numnodes); % M has same nodes as as G
```

We also need a list of vertices that are not yet matched. Since M is empty, that's just all the nodes in G.

```
M_notmatched = 1:G.numnodes; % M_notmatched= [1,2,3, ..., n]
```

3. The Matching Algorithm: Step 1

Next step is to make a directed graph, with edges between nodes if they are in the matching.

We start with an empty graph, D, with the same nodes as G. Then, for every edge $e = \{v_1, v_2\}$ in G, where v_1 is in P_1 and v_2 is in P_2, we'll add the directed edge (v_1, v_2) to D if it is also in M. Otherwise we add the edge (v_1, v_2) to D.

Some features of MATLAB's graph obect that we'll use:

- If `G` is a graph, then `G.numedges` is the number of edges in `G`.
- `G.Edges` returns a "table" representing the edges. Since we haven't used tables, we'll access the associated $n \times 2$ matrix, where `n=G.numedge.` Do that with the `G.Edges.endnodes` matrix.
- `Row` i of this matrix contains the start end nodes of Edge i of `G`.

```
D = digraph(); % D1 is a *directed* graph
D = D.addnode(G.numnodes);
% Iterate over the edges of H
n = G.numedges;
for i = 1:n
   e = G.Edges.EndNodes(i,:); % e is vector
   if ( iselement(P1, e(1)))
      v1=e(1); v2=e(2);
   else
      v1=e(2); v2=e(1);
   end
   if (M.findedge(v1,v2)) % is (e(1),e(2)) in matching?
      D = D.addedge(v1, v2);
   else
      D = D.addedge(v2, v1);
   end
end
```

```
plot(D); title("D (step 1)");
```

Now we need to find a path in D between any pair of unmatched nodes, i.e., vertices in the list `M_notmatched`. There could be numerous such paths. Although it is quite inefficient, we'll check all, and choose the longest one.

```
P = [];
for u1=M_notmatched % iterate over all matched nodes
   for u2 = setdiff(M_notmatched, u1) % iterate over those other than i
      p1 = D.shortestpath(u1,u2);
      if (length(p1) > length(P))
         P=p1;
      end
   end
end
fprintf("New path is : "); disp(P)
```

Next we add every odd numbered this path to M, removing the even numbered ones. The first and last nodes in P will no longer be unmatched, so we remove them.

```
for i=1:2:length(P)-1
   M=M.addedge(P(i),P(i+1));
end
for i=2:2:length(P)-1
   M=M.rmedge(P(i),P(i+1));
end
M_notmatched = setdiff(M_notmatched, P(1)); % remove the 1st
M_notmatched = setdiff(M_notmatched, P(end)); % and last nodes in P

g = plot(G);
highlight(g,M);
title('Graph G with matching highlighted (step 1)')
```

4. Next step

If you run the code in the section above, it should add new vertices to the matching every time, until such time as all edges are matched. Then it breaks because P will be empty, and so `P(1)` is not legal.

Exercise 3.8 from Problem Set 3

This is an exercise from Problem Set 3.

Rather than repeatedly running the code in Section 3 until it breaks, write a `while()` loop that repeats it. When doing that, it might be helpful to remove all text from that section (so that all the code is in a single cell). What is a good termination requirement?

Also: change the graph generated in Section 1 so that it has at least 20 vertices.

Upload your live script to Blackboard (Lab 5). Deadline: to be discussed.

Bonus

If you'd like to show off your MATLAB programming skills,

- *write a function that creates a random bipartite graph.*
- write a function that computes a matching for any input graph.

Function

Check if a node is in a list

```
function YN = iselement(A, s)
if isempty(find(A==s,1))
   YN = false;
else
   YN = true;
end
end
```