Assignment 2: Solutions

TOTAL MARKS: 0

1. Solutions:

   - Problem a1898629
     Edge list: (1,2) (1,3) (2,6) (4,1) (4,6) (5,4) (6,4) (6,5)
     in-degree: 1, 1, 1, 2, 1, 2,
     out-degree: 2, 1, 0, 2, 1, 2,
     DFS (first node with out-degree == 0): 3

   - Problem a1897413
     Edge list: (1,2) (1,5) (1,6) (2,4) (3,1) (3,1) (4,3) (4,3) (5,6) (6,4)
     in-degree: 2, 1, 2, 2, 1, 2,
     out-degree: 3, 1, 2, 2, 1, 1,
     DFS (first node with out-degree == 0): Inf

   - Problem a1932791
     Edge list: (1,4) (1,5) (2,5) (3,6) (4,1) (4,2) (5,3) (5,4) (6,2)
     in-degree: 1, 2, 1, 2, 2, 1,
     out-degree: 2, 1, 1, 2, 2, 1,
     DFS (first node with out-degree == 0): Inf

   - Problem a1871789
     Edge list: (1,3) (1,5) (2,4) (3,4) (3,6) (4,1) (4,5) (5,3) (6,2)
     in-degree: 1, 1, 2, 2, 2, 1,
     out-degree: 2, 1, 2, 2, 1, 1,
     DFS (first node with out-degree == 0): Inf

   - Problem a1734046
     Edge list: (1,1) (1,2) (1,3) (1,4) (2,1) (3,6) (4,5) (5,1) (6,2)
     in-degree: 3, 2, 1, 1, 1, 1,
     out-degree: 4, 1, 1, 1, 1, 1,
     DFS (first node with out-degree == 0): Inf

   - Problem a1820798
     Edge list: (1,4) (1,6) (2,3) (3,1) (4,5) (5,6) (6,1) (6,2) (6,6)
     in-degree: 2, 1, 1, 1, 1, 3,
     out-degree: 2, 1, 1, 1, 1, 3,
     DFS (first node with out-degree == 0): Inf

   - Problem a1825938
     Edge list: (1,4) (1,5) (1,6) (2,4) (2,4) (4,1) (4,2) (4,3) (5,2) (6,1)
     in-degree: 2, 2, 1, 3, 1, 1,
     out-degree: 3, 2, 0, 3, 1, 1,
     DFS (first node with out-degree == 0): 3

   - Problem a1813487
     Edge list: (1,3) (1,5) (2,1) (2,4) (3,4) (4,5) (5,2) (5,5) (5,6) (6,2)

in-degree: 1, 2, 1, 2, 3, 1,
out-degree: 2, 2, 1, 1, 3, 1,
DFS (first node with out-degree == 0): Inf

- Problem a1871781
  Edge list: (1,4) (1,6) (2,5) (3,5) (4,1) (4,2) (5,2) (5,6) (6,3) (6,4)
  in-degree: 1, 2, 1, 2, 2, 2,
  out-degree: 2, 1, 1, 2, 2, 2,
  DFS (first node with out-degree == 0): Inf

- Problem a1907611
  Edge list: (1,2) (2,3) (3,3) (3,4) (3,4) (4,3) (4,6) (5,1) (6,5)
  in-degree: 1, 1, 3, 2, 1, 1,
  out-degree: 1, 1, 3, 2, 1, 1,
  DFS (first node with out-degree == 0): 1

- Problem a1709218
  Edge list: (1,2) (1,3) (1,4) (2,1) (2,1) (3,6) (4,2) (6,5)
  in-degree: 2, 2, 1, 1, 1, 1,
  out-degree: 3, 2, 1, 1, 0, 1,
  DFS (first node with out-degree == 0): 5

Code to solve the problem is included below. Please note that this is written to be obvious, not efficient.

Also note that the code implicitly assumes the graph is strongly connected. The graphs you were given are not strongly connected, but that's why I asked you to search for the out-degree 0 nodes, because those were the likely points at which the code would break.

In general, when you find a dfs has stopped, you would restart it again much like the connected components algorithm.

```
function [n, labels] = dfs(A, s, labels)
%
% dfs.m, (c) M Roughan, 2017
%
% created:  Thu Aug  3 2017
% author:   M Roughan
% email:    matthew.roughan@adelaide.edu.au
%
% DFS
%    DFS, presuming the graph is strongly connected
%
% INPUTS:
%      A = adjacency matrix
%             A[i,j] = 1   => there is a link from i to j
%                      n>1 => the link has multiplicity n
%      s = a start node
%      labels = a nx1 vector of 1/0 with 1 indicating that we have already explored th
%
% OUTPUTS:
%      The first node found with out-degree 0
%
%
%
if nargin < 2
  s = 1;
  labels = zeros(1,size(A,1));
end
labels(s) = 1;

tmp = A(s,:);
out_degree = sum(tmp);
if out_degree == 0
  n = s;
  return
end

k = find( tmp & (1-labels) ); % unlabelled, adjacent nodes
if isempty(k)
  n = Inf;
  return
else
  for i=1:length(k)
    [n, labels] = dfs(A, k(i), labels);
    if isfinite(n)
      return
    end
  end
end
```

2. (a) Presume that the graph is represented as a neighbourhood list, that is we have an array of nodes, each of which has an associated list of neighbours.
First, we can calculate the degrees of each node to check for balance in $O(e)$ time: for each node we add up the number of items on its neighbourhood list, but the total of these additions cannot be larger than the number of edges. If balanced, then proceed, otherwise return that there is no cycle. We could also check for connectedness here, which will be $O(n)$, but we don't need to include these in the algorithm, as we can presume that the graph is balanced and connected.
Finding a cycle involves picking an edge (easiest to be first) from unused edges at each node. If we maintain a pointer to the start of the unused part of each list, then choosing and edge, and moving the pointer are $O(1)$ operations. We must also check if we have returned to the start node, again an $O(1)$ operation. So following a path of length m takes $O(m)$ operations.
We next must find a node which has unused edges directed from it. If we search for it at this point, the worst case would be $O(n)$. However, if we also maintain a list of nodes (on the existing tour) that have unused edges, we can find a new node in $O(1)$ time.
Maintain the current path as a doubly-linked list. Then stitching together two can also be performed in $O(1)$ time, so the overall complexity is $O(e)$.
(The algorithm is called Hierholzer's algorithm)

   (b) Solutions:
   - Problem a1898629
     Edge list: (1,3) (2,6) (4,1) (4,6) (5,4) (6,4) (6,5)
     Eulerian path (generator): 2, 6, 4, 6, 5, 4, 1, 3,
     Eulerian path (alternative): 2, 6, 4, 6, 5, 4, 1, 3,

   - Problem a1897413
     Edge list: (1,5) (1,6) (2,4) (3,1) (3,1) (4,3) (4,3) (5,6) (6,4)
     Eulerian path (generator): 2, 4, 3, 1, 6, 4, 3, 1, 5, 6,
     Eulerian path (alternative): 2, 4, 3, 1, 6, 4, 3, 1, 5, 6,

   - Problem a1932791
     Edge list: (1,5) (2,5) (3,6) (4,1) (4,2) (5,3) (5,4) (6,2)
     Eulerian path (generator): 4, 1, 5, 4, 2, 5, 3, 6, 2,
     Eulerian path (alternative): 4, 1, 5, 3, 6, 2, 5, 4, 2,

   - Problem a1871789
     Edge list: (1,5) (2,4) (3,4) (3,6) (4,1) (4,5) (5,3) (6,2)
     Eulerian path (generator): 3, 6, 2, 4, 1, 5, 3, 4, 5,
     Eulerian path (alternative): 3, 6, 2, 4, 5, 3, 4, 1, 5,

   - Problem a1734046
     Edge list: (1,1) (1,2) (1,3) (2,1) (3,6) (4,5) (5,1) (6,2)
     Eulerian path (generator): 4, 5, 1, 2, 1, 1, 3, 6, 2,
     Eulerian path (alternative): 4, 5, 1, 3, 6, 2, 1, 1, 2,

   - Problem a1820798
     Edge list: (1,6) (2,3) (3,1) (4,5) (5,6) (6,1) (6,2) (6,6)
     Eulerian path (generator): 4, 5, 6, 6, 1, 6, 2, 3, 1,
     Eulerian path (alternative): 4, 5, 6, 1, 6, 2, 3, 1,

- Problem a1825938
  Edge list: (1,4) (1,5) (2,4) (2,4) (4,1) (4,2) (4,3) (5,2) (6,1)
  Eulerian path (generator): 6, 1, 5, 2, 4, 2, 4, 1, 4, 3,
  Eulerian path (alternative): 6, 1, 4, 1, 5, 2, 4, 2, 4, 3,

- Problem a1813487
  Edge list: (1,3) (2,1) (2,4) (3,4) (4,5) (5,2) (5,5) (5,6) (6,2)
  Eulerian path (generator): 5, 2, 4, 5, 5, 6, 2, 1, 3, 4,
  Eulerian path (alternative): 5, 2, 4, 5, 5, 6, 2, 1, 3, 4,

- Problem a1871781
  Edge list: (1,6) (2,5) (3,5) (4,1) (4,2) (5,2) (5,6) (6,3) (6,4)
  Eulerian path (generator): 4, 1, 6, 3, 5, 6, 4, 2, 5, 2,
  Eulerian path (alternative): 4, 1, 6, 3, 5, 2, 5, 6, 4, 2,

- Problem a1907611
  Edge list: (2,3) (3,3) (3,4) (3,4) (4,3) (4,6) (5,1) (6,5)
  Eulerian path (generator): 2, 3, 4, 3, 3, 4, 6, 5, 1,
  Eulerian path (alternative): 2, 3, 3, 4, 3, 4, 6, 5, 1,

- Problem a1709218
  Edge list: (1,2) (1,3) (2,1) (2,1) (3,6) (4,2) (6,5)
  Eulerian path (generator): 4, 2, 1, 2, 1, 3, 6, 5,
  Eulerian path (alternative): 4, 2, 1, 2, 1, 3, 6, 5,

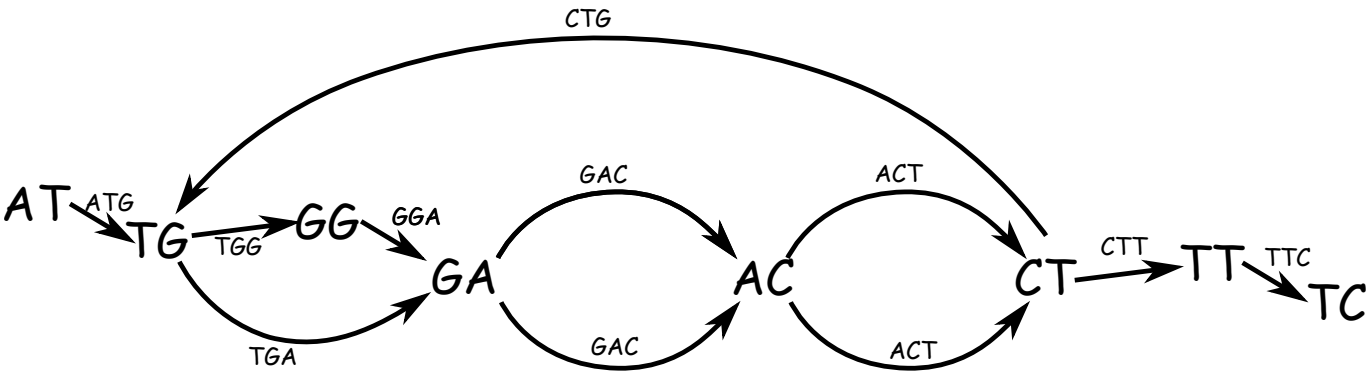
Code to solve the problem is included below. Please note that this is written to be obvious, not efficient.

```
function walk = euler_walk(A)
%
% euler.m, (c) M Roughan, 2017
%
% created:  Thu Aug  3 2017
% author:   M Roughan
% email:    matthew.roughan@adelaide.edu.au
%
% EULER
%   find an Euler path through the nodes
%
% INPUTS:
%      A = adjacency matrix
%             A[i,j] = 1   => there is a link from i to j
%                      n>1 => the link has multiplicity n
%
% OUTPUTS:
%      walk = vector containing the nodes on the Eulerian walk
%
%
[in_degree, out_degree] = degree(A);
total_edges = sum(in_degree);
```

```
d = in_degree - out_degree;
k = find( d ~= 0 );
if length(k) ~= 2
  walk = NaN;
  return; % no path walk exists
end
start = find( d == -1 );
stop = find( d == 1 );
if length(start)~=1 || length(stop)~=1
  walk = Inf;
  return; % no path walk exists
end

% find an initial walk from start to stop nodes
i = start;
walk = [i];
path_length = 0;
while i~=stop && path_length<=total_edges
  i_old = i;
  i = find(A(i_old,:)>0, 1);
  walk = [walk, i];
  A(i_old, i) = A(i_old, i) - 1;
  path_length = path_length + 1;
end
walk;
path_length;
total_edges;

% now, while we haven't finished, try again
while path_length < total_edges
  % walk
  % A

  % find a point to start a side loop
  current_nodes = unique(walk);
  tmp = sum(A,2)';
  choice = intersect(walk, find(tmp(current_nodes)>0)); % a node in current walk w
  choice = choice(1);
  k = find(choice == walk, 1);

  loop = [];
  i = 0;
  i_old = choice;
  while i~=choice
    i = find(A(i_old,:)>0, 1);
    loop = [loop, i];
    A(i_old, i) = A(i_old, i) - 1;
    path_length = path_length + 1;
    i_old = i;
  end
  % A(i, choice) = A(i, choice) - 1;
```

```
    path_length = path_length + 1;
    % loop
    if k==length(walk)
      walk = [walk(1:k), loop];
    else
      walk = [walk(1:k), loop, walk(k+1:end)];
    end
  end
```

3. The figure below shows the de Brujin graph of the reads.



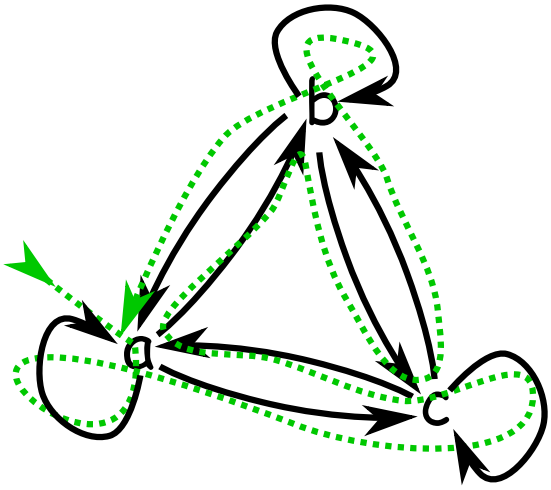
There are two possible solutions:

$$ATGGACTGACTTC$$
$$ATGACTGGACTTC$$

I used the first solution when generating this problem.

4. (a) There are n^k possible k-mers, but each must follow from the last, so each k-mer only introduces one new symbol (as prefixes must meet suffixes cyclically) into the universal string, and hence the string will be of length n^k.

   (b) Given we are dealing with k-mers, the prefixes and suffixes are a single character, *i.e.,* the set $\{a, b, c\}$, so we can create a de Brujin graph specifying all possible 2-mers as follows:



As we will generate circular strings, we could start at any node, and we trace an Eulerian walk. There are several possibilities, *e.g.,* the one corresponding to the walk above is

aaccabcbb