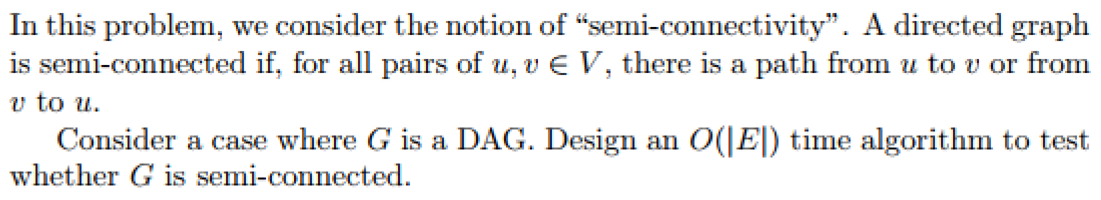
W21 CS180 Homework 3 Rubric

Problem 1



In this problem, we consider the notion of "semi-connectivity". A directed graph is semi-connected if, for all pairs of $u, v \in V$, there is a path from u to v or from v to u.

Consider a case where G is a DAG. Design an $O(|E|)$ time algorithm to test whether G is semi-connected.

Topologically sorting G into a list L. For vertices v_i, v_{i+1}, if no edge exists, then G is not semi-connected. (**6 pts**)

Explain why time complexity is O(|E|) (**4 pts**)

Partial credits awarded for the algorithm.

Problem 2

Suppose a directed graph G is decomposed to SCCs $G_1, .., G_n$ by Kosaraju's algorithm. $G_1, \ldots, G_n$ form a partition. (**2 pts**)

Contract each strongly connected component into a single vertex, the resulting graph is a DAG (**4 pts**). Name the resulting graph G*.

Apply the algorithm in problem 1 to G*, and we can test the semi-connectivity (**2 pts**).

Provide explanations on time complexity (**2 pts**).

You should provide explanations on "the graph of strongly connected component is acyclic.", i.e. explain which graph (or component) is acyclic in your algorithm.

Problem 3

part a. Recursive algorithm

You should state whether your graph is directed or undirected, because the algorithm for a directed or an undirected graph can be very different. I assume for part a and part b, your graph is under the same assumption, unless otherwise stated. ***If during the exam, you did not explicitly make such statements, you may lose points.***

A short generalization of your strategy (**1 pt**). A correct recursive algorithm (**4 pts**).

Eulerian Path in undirected graph

- An undirected graph has an eulerian path if and only if it is connected and all vertices except 2 have even degree. One of those 2 vertices that have an odd degree must be the start vertex, and the other one must be the end vertex.

`main algorithm`

```
Start vertex:

    case 1: an odd vertex (if there are odd vertices).
    case 2: If there are zero odd vertices, we start from any vertex .

For current vertex $u$:

    traverse all adjacent vertices of $u$ to find an edge to the next vertex
$v$,

        case 1: if there is only one adjacent vertex $v$ , we immediately
consider it.

        case 2: If there are more than one adjacent vertices, we consider an
adjacent $v$ only if edge u-v is not a bridge.

    add u-v to the path, remove u-v from the edge list. choose v to be the next
vertex to be processed.
```

`helper function: isBridge(u,v)`

```
count number of vertices reachable from u, denoted count1. (Use DFS to count the
reachable vertices.)

remove edge u-v and again count number of reachable vertices from u, denoted
count2.

If count1 > count2:

    u-v is a bridge

else:

    u-v is not a bridge
```

Eulerian Path in directed graph

- A directed graph has an eulerian path if and only if it is **connected** and each vertex except 2 have the same in-degree as out-degree, and one of those 2 vertices has out-degree with one greater than in-degree (this is the start vertex), and the other vertex has in-degree with one greater than out-degree (this is the end vertex)

The idea is to keep following unused edges and removing them until we get stuck. Once we get stuck, we backtrack to the nearest vertex in our current path that has unused edges, and we repeat the process until all the edges have been used.

```
Choose any starting vertex v
follow a trail of edges from that vertex until returning to v.
    The tour formed in this way is a cycle, but may not cover all the vertices
and edges    of the initial graph.
As long as there exists a vertex u that belongs to the current tour, but that has
adjacent edges not part of the tour, start another trail from u, following unused
edges until returning to u, and join the tour formed in this way to the previous
tour.
```

Note:

  Claim: We find all simple cycles and combine them into one.

This is partially correct, because an Eulerian path does not have to be an Eulerian cycle. If your algorithm is based on this assumption/strategy, *you will get one point off.*

part b. Iterative algorithm

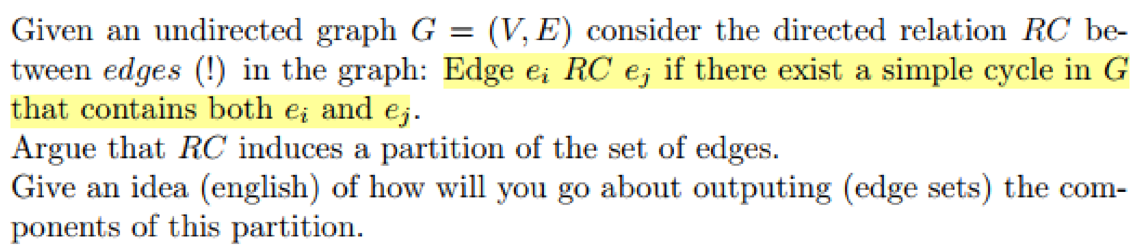
A correct iterative algorithm (**4 pts**)

On directed graph: use stack

```
stack St;
put start vertex in St;
until St is empty
  let V be the value at the top of St;
  if degree(V) = 0, then
    add V to the answer;
    remove V from the top of St;
  otherwise
    find any edge coming out of V;
    remove it from the graph;
    put the second end of this edge in St;
```

Explanations on time complexity (**1 pt**)

Problem 4



Given an undirected graph $G = (V, E)$ consider the directed relation RC between *edges* (!) in the graph: Edge e_i RC e_j if there exist a simple cycle in G that contains both e_i and e_j.
Argue that RC induces a partition of the set of edges.
Give an idea (english) of how will you go about outputing (edge sets) the components of this partition.

If a relation is reflexive, symmetric and transitive, then it is an equivalent relation. Each equivalence relation provides a partition of the underlying set into disjoint equivalent classes (**4 pts**).

To represent the output set, as long as your algorithm works, you get full points (**6 pts**). Partials are awarded.