For questions 3 and 4 you must include the following:

- High-level description of your algorithm
- Proof of correctness
- Runtime analysis (Based on $|V|$ and $|E|$)

Note that you will also be graded on the efficiency of your algorithms. It may also be helpful to write psuedo-code for your algorithms to aid you in writing a runtime analysis and for your general understanding of your solutions.

1. Run the SCC algorithm on the following directed graph G. When doing DFS on G^R: whenever there is a choice of vertices to explore, always pick the one that is alphabetically first.

$A : D, G$
$B : F, G, L$
$C : B, E$
$D : G, H$
$E : A, J$
$F : B$
$G : H$
$H : B, L$
$I : K$
$J : F, L$
$K : D$
$L : E, H, K$

   (a) (5 points) In what order are the strongly connected components (SCCs) found?
   (b) (5 points) Which are source SCCs and which are sink SCCs?
   (c) (5 points) Draw the "metagraph" (each meta-node is an SCC of G)

2. (10 points) Consider the following problem:

   > Given a strongly connected simple *directed* graph G, determine the total number of cycles in the graph.
   > Consider the following algorithm that claims to compute the total number of cycles in the graph.
   >
   > (In a *cycle*, you cannot repeat vertices (except for the starting and ending vertices) and you cannot repeat edges.)

For each algorithm,

- Provide a runtime analysis (Based on $|V|$ and $|E|$)
- identify if it correctly solves the problem.
- If it is correct, provide a correctness proof. If it is not correct, provide a counterexample.

(a) **Algorithm1**(G; a strongly connected simple directed graph G.)

```
Run DFS(G)
c = 0
for each edge (u, v) then
    if post(v) < post(u) then
        c = c + 1
return c
```

(b) **Algorithm2**(G; a strongly connected simple directed graph G.)
[[run graphsearch and every time you encounter a vertex you have already seen before, increment your counter, c.]]

```
c = 0
for all v ∈ V:
    Status(v) = U
Pick any vertex s
Status(s) = F
Initialize a Stack: F = [s]
while |F| > 0
    w = pop(F).
    For each outgoing neighbor y of w (for each (w, y) ∈ E):
        if Status(y) ≠ U:
            c = c + 1
        else:
            Status(y) = F
            push(F, y)
    Status(w) = X
return c
```

3. (15 points) You are given a simple directed graph G with vertex set V, edge set E and vertex labels $L(v) \in \{0, 1\}$ as well as a starting and ending vertex s, t.

   Design a reasonably efficient algorithm that determines if there is a walk from s to t such that the sequence of vertex labels in the walk have exactly one occurrence of two 1's in a row.

   (Note that a *walk* is a sequence of edges from s to t such that you can repeat edges and vertices.)

   (7 points for reasonably efficient correct high level algorithm description (with correctness proof), 5 points for correct time analysis, and 3 points for efficiency of your algorithm.)

4. (15 points)

   You are given a directed graph.

   Design a reasonably efficient algorithm that *determines* if there exists a walk that goes through each vertex at least once.

   (7 points for reasonably efficient correct high level algorithm description (with correctness proof), 5 points for correct time analysis, and 3 points for efficiency of your algorithm.)