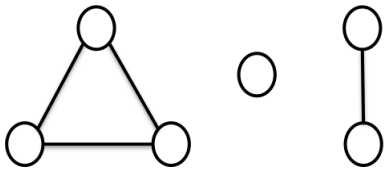
COL351: Analysis and Design of Algorithms
Instructor: Ragesh Jaiswal

1. In the lectures, we considered DFS starting from a given vertex s (we called this `DFS(`s`)`). In case the undirected graph is not strongly connected, running the DFS from a starting vertex does not explore all vertices. In such cases, it makes sense to define the following graph exploration algorithm:

```
GraphDFS(G)
  - While there exists a node u that has not been "explored"
    - DFS(u)
```

   Note that the above algorithm will result in a "DFS Forest" instead of a DFS tree as discussed in class. Run this algorithm on the graph below and show the resulting forest.

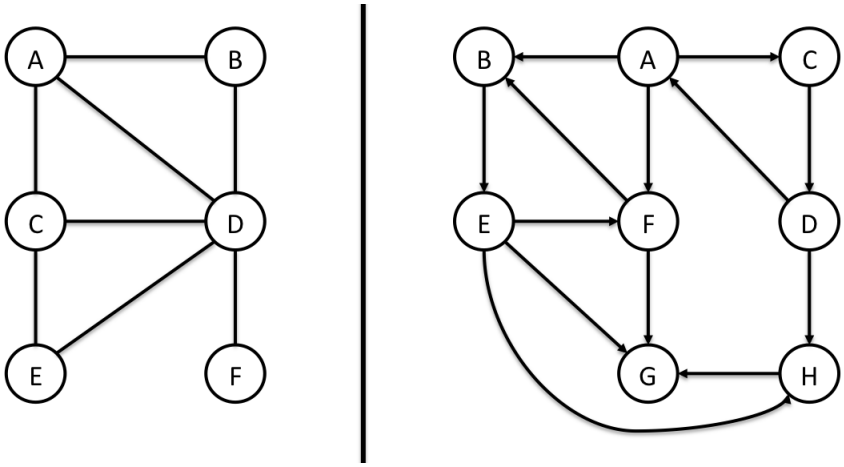


2. Consider an undirected graph G and suppose we run the GraphDFS algorithm on G. The edges in the graph may be placed in two categories depending on the `GraphDFS` execution. Some edges will be part of the resulting DFS forest. These are called *tree edges*. The remaining edges are called *back edges*.

   For the case of directed graphs, the edges may be categorized further. Let node v be reachable from node u in the DFS forest. Then v is called the *descendent* of u and u is called the *predecessor* of v. The edges can be placed in the following four categories:

   - Tree edges: These are the edges that are part of the DFS forest.
   - Forward edges: These are edges that go from a node to a non-child descendent in the DFS forest.
   - Back edges: These are edges that go from a node to its ancestor.
   - Cross edges: These are nodes that go from a node to a node that is neither its ancestor or its predecessor in the DFS forest.

   Run the `GraphDFS` algorithm in the given undirected and directed graphs below and mark the different types of edges.



3. Note that back edges are easy to detect while running the `GraphDFS` algorithm. During the execution if an edge leads to a vertex v that is still in the recursion stack (i.e., the recursive call $DFS(v)$ has not returned), then such an edge is a back edge. Show that a directed graph has a cycle if and only if one detects a back edge while running `GraphDFS`. Use this to design an algorithm to check if a given directed graph has a cycle. Discuss running time of your algorithm.

4. Use the ideas developed for finding strongly connected components of a directed graph to give another algorithm for topological ordering of vertice of a given DAG by performing a single execution of the DFS algorithm.

5. You are given a directed graph $G = (V, E)$ in which each node $u \in V$ has an associated *price*, denoted by $price(u)$, which is a positive integer. The *cost* of a node u, denoted by $cost(u)$, is defined to be the price of the cheapest node reachable from u (including u itself). Design an algorithm that computes $cost(u)$ for all $u \in V$. Discuss correctness and running time.