EXERCISE 1: Minimum Spanning Trees

Each of the figures below represents a partial spanning tree. Determine whether it could possibly be obtained from (a prematurely stopped) Kruskal’s algorithm, (a prematurely stopped) Prim’s algorithm, both or neither.

<table>
<thead>
<tr>
<th></th>
<th>Kruskal</th>
<th>Prim</th>
<th>Both</th>
<th>Neither</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>(B)</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>(C)</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>(D)</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>
EXERCISE 2: Shortest Common Ancestor

In a directed graph, a vertex $x$ is an ancestor of $v$ if there exists a (directed) path from $v$ to $x$. Given two vertices $v$ and $w$ in a rooted directed acyclic graph (DAG), a shortest common ancestor $sca(v, w)$ is a vertex $x$ which:

- is an ancestor to both $v$ and $w$;
- minimizes the sum of the distances from $v$ to $x$ and $w$ to $x$ (this path, which goes from $v$ to $x$ to $w$, is the shortest ancestral path between $v$ and $w$).

A. In the following digraph, find the shortest common ancestor of vertices 1 and 4, and give the sum of the path lengths from these vertices to all common ancestors, and then circle the shortest.

![Directed Graph]

B. Describe an algorithm for calculating the shortest common ancestor of two vertices $v$ and $w$. Your algorithm should run in linear time (proportional to $V + E$).

C. How would your algorithm differ if we are interested in the shortest ancestral path between two sets of vertices $A$ and $B$ instead of two vertices? I.e. between any vertex $v$ in $A$ and any vertex $w$ in $B$.

In the example, $A = 3, 11$ and $B = 9, 10, 13$. The shortest common ancestor is 5 (between 10 and 11).
EXERCISE 3: Detecting Directed Cycles

A. Consider the graph $G$ given below and the marked vertex $s$. Show in the given box what the output would be if depthFirstSearch is called on $G$ and $s$.

B. Consider the following modified version of the dfs method. Explain with the simplest counterexample why this code is not a correct cycle detection code.
C. Briefly describe how depth-first search could be modified to detect cycles in a digraph.

D. Fill the blank lines in the following DFS code so that it prints “Cycle found!” if and only if there is a cycle in the graph. Assume that the graph is connected.

```java
private boolean[] marked;
private boolean[] onStack;

public void checkCycles(Digraph G, int s) {
    marked = new boolean[G.V()];
    ______________________________________
    dfs(G, s);
}

private void dfs(Graph G, int v) {
    marked[v] = true;
    ______________________________________
    for (int w : G.adj(v)) {
        if (!marked[w])
            dfs(G, w);
        else if (_______________________)
            StdOut.print("Cycle found!");
    }
    ______________________________________
}```