

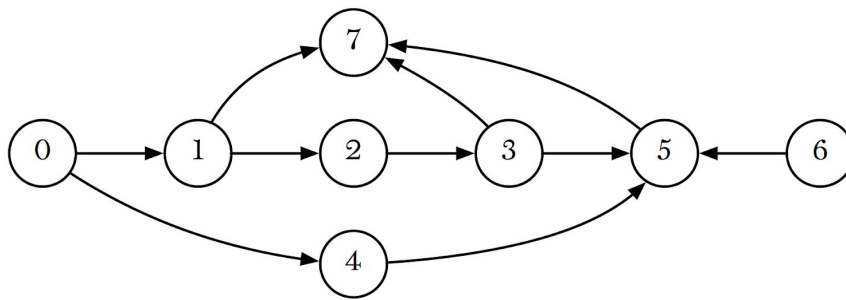
EXERCISE 1: Shortest Common Ancestor

The following exercise looks at the algorithm required to compute the *shortest common ancestor* in a directed acyclic graph, which is an essential component of the WordNet assignment's `ShortestCommonAncestor` class.

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:

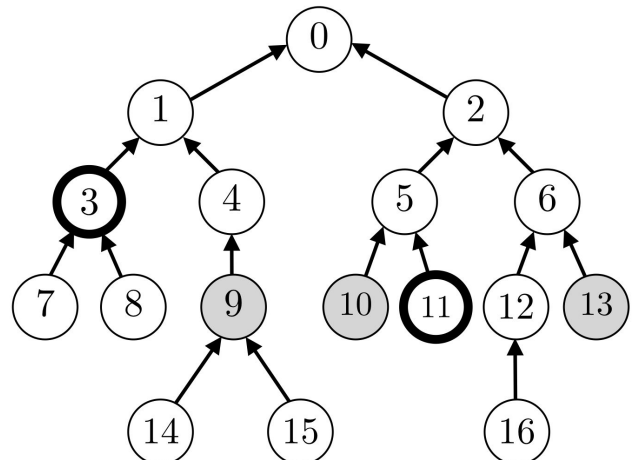
- is an ancestor to both v and w ;
- minimizes the sum of the distances from v to x and w to x (this path, which goes from v to x to w , is the shortest ancestral path between v and w).

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.



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 $V + E$).

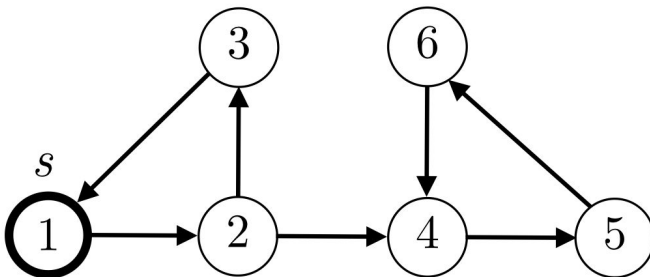
C. A shortest ancestral path of two subsets of vertices A and B over all pairs of vertices v and w , with v in A and w in B . How would your algorithm differ if A and B were two sets of vertices rather than single vertices? In the example, $A = \{3, 11\}$ and $B = \{9, 10, 13\}$.



EXERCISE 2: 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 ?

```
1| private boolean[] marked;
2|
3| public void depthFirstSearch(Graph G, int s) {
4|     marked = new boolean[G.V()];
5|     dfs(G, s);
6| }
7|
8| private void dfs(Graph G, int v) {
9|     marked[v] = true;
10|    StdOut.println("Starting " + v);
11|    for (int w : G.adj(v)) {
12|        if (!marked[w])
13|            dfs(G, w);
14|    }
15|    StdOut.println("Finished " + v);
16| }
```



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.

```
8| private void dfs(Graph G, int v) {
9|     marked[v] = true;
10|
11|    for (int w : G.adj(v)) {
12|        if (!marked[w])
13|            dfs(G, w);
14|        else StdOut.print("Cycle found!");
15|    }
16| }
```

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 cycle in the graph.

```
1| private boolean[] marked;
2| private boolean[] onStack;
3|
4| public void depthFirstSearch(Graph G, int s) {
5|     marked = new boolean[G.V()];
6|     _____
7|     dfs(G, s);
8| }
9|
10| private void dfs(Graph G, int v) {
11|     marked[v] = true;
12|     _____
13|     for (int w : G.adj(v)) {
14|         if (!marked[w])
15|             dfs(G, w);
16|         else if (_____)
17|             StdOut.print("Cycle found!");
18|     }
19|     _____
20| }
```