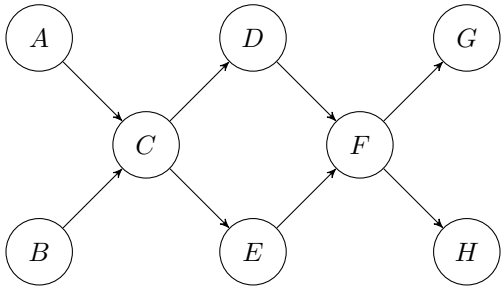
Weekly Assignment 3:
Depth-First Search

September 2023

1. Run the DFS-based topological ordering algorithm on the following graph. Whenever you have a choice of vertices to explore, always pick the one that is alphabetically first.



   How many topological orderings does this graph have?

2. Modify DFS such that each edge of the graph and its class (tree edge, forward edge, backward edge or cross edge) are displayed.

3. Design a linear-time algorithm for the following task:

   Input: A directed acyclic graph G

   Output: Does G contain a path that visits all vertices of G?

   Discuss the time complexity of your algorithm and explain why it is correct.

4. Suppose a Computer Science curriculum consists of n courses. Each course is mandatory. The prerequisite graph for the curriculum is a directed, acyclic graph (DAG) G that has a node for each course, and an edge from course v to course w if and only if v is a prerequisite for w. In such a case, a student is not allowed to take course w in the same or an earlier semester than she takes course v. A student can take any number of courses in a single semester.

   Design an algorithm to find out the minimum number of semesters necessary to complete the curriculum. The running time of your algorithm should be linear. Discuss the correctness of your solution. Your algorithm should consider the extreme case also when there are no pre-requisites at all.

5. A student has to write an algorithm to find a cycle in an directed graph, that is to say, given a graph G, find cycle $(x_1, \ldots x_{k-1}, x_k = x_1)$, or find that there is no cycle. Being smart, they found on the web that DFS is the algorithm to use. Thus, they came up with the algorithm below. Find and correct the mistakes.

Algorithm 4 Buggy student's program

```
1:  procedure findCycle(G)
2:      for all x ∈ V do
3:          mark[x] ← false
4:          parent[x] ← nil
5:      end for
6:      x ←chooseElement(V)                          ▷ choose element randomly
7:      if cc(G, x) = false then
8:          WriteLn("No loop")
9:      end if
10: end procedure

11: procedure cc(G, x)
12:     mark[x] ← true
13:     for all y ∈ neighbourhood(x, G) do
14:         if not mark[y] then
15:             parent[y] ← x
16:             if cc(G, y) = true then
17:                 return True
18:             end if
19:         else
20:             display(y)
21:             z ← x
22:             while z ≠ y do
23:                 display(z)
24:                 z ← parent[z]
25:             end while
26:             return true
27:         end if
28:     end for
29:     return false
30: end procedure
```