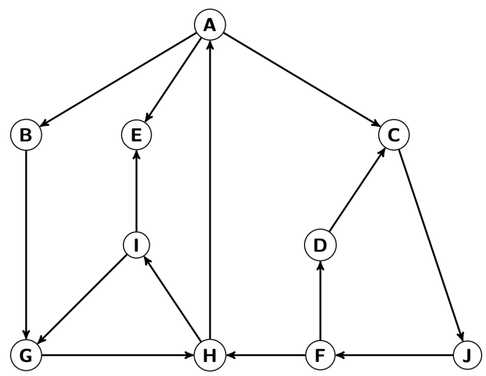
Week 12: Lab

Module: Graphs

Collaboration level 0 (no restrictions). Open notes.

1. Consider the graph below:

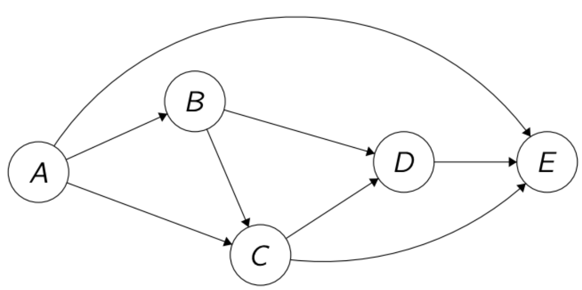


(a) What are all the strongly connected components?

(b) Perform DFS on the graph above starting from vertex A. Use lexicographical ordering to break vertex ties. As you go, label each vertex with the start and finish time. Draw the DFS-tree generated from the search and highlight them on the graph.

(c) Perform BFS on the graph above starting from vertex A. Use lexicographical ordering to break vertex ties. Draw the BFS-tree generated from the search and highlight them on the graph.

2. Consider the graph below:



(a) Without using any specific algoorithm, come up with a topological order for the graph. How many different orders can you come up with?

(b) Run DFS on the whole graph starting at vertex C, and consider the the edges in the adjacency lists in alphabetical order. Recall that when you run DFS on the graph, if it stops but has not yet explored the whoole graph, it will start again at the next unexplored (white) vertex.

(i) What do you get when you order the vertices by ascending start time?

(ii) What do you get when you order the vertices by descending finish time?

(c) Now run DFS starting at vertex C, but consider the edges in the adjacency lists in in reverse alphabetical order.

(I) What do you get when you order the vertices by ascending start time?

(ii) What do you get when you order the vertices by descending finish time?

3. Explain why the following algorithm does not necessarily produce a topological order: Run BFS, and label the vertices by increasing distance to their respective sources.

   Note: To prove that a certain algorithm does not work, it is sufficient to show a counter-example.

4. **Shortest paths on DAGs:** Consider a directed acyclic graph G. Unlike the problems so far, in which we considered that the edges are "equal" and we counted the weight of a path as the numbr of edges on the path, in this problem we'll assume that the graph is *weighted*. What that means is that each edge (u, v) has a weight which we denote by w_{uv}. The weight represents the cost of traversing that edge (for e.g. the weight could be the distance beween the vertices). The weight of a path in a weighted graph is defined as the sum of the weights of the edges alog the path.

   So: given a weighted DAG and two arbitrary vertices u and v, the goal is to compute the shortest path from u to v. Note that if the graph was not weighted, we could simply run BFS form u, but a breadth-first approach does not work when there are weights.

   Explore an algorithm for finding the shortest path from u to v. You can assume, for simplicity, that u is a vertex without any incoming edges, and is the first in a topological order.