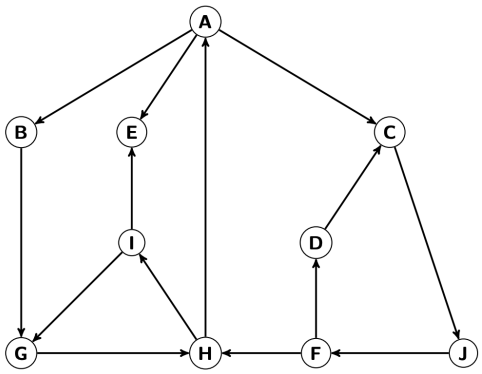
Week 12: Lab

Module: Graphs

Collaboration level 0 (no restrictions). Open notes.

1. Consider the graph below:

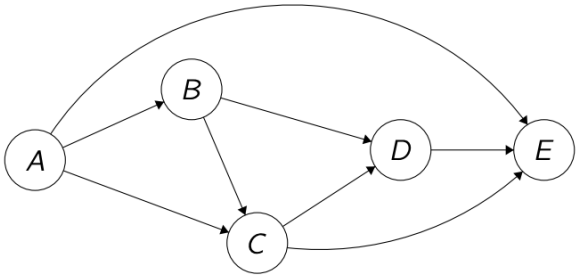


   (a) What are all the strongly connected components?

   (b) Perform DFS on the graph above starting from vertex A. Assume that the adjacency lists are ordered in alphabetical order. As you go, label each vertex with the start and finish time. Draw the DFS-tree and highlight the tree edges on the graph.

   (c) Perform BFS on the graph above starting from vertex A. Again, assume the adjacency lists are ordered in alphabetical order. Draw the BFS-tree and highlight the tree edges on the graph.

2. Consider the graph below:



   (a) Without using any specific algorithm, 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 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.

- What do you get when you order the vertices by ascending start time?
- What do you get when you order the vertices by descending finish time?

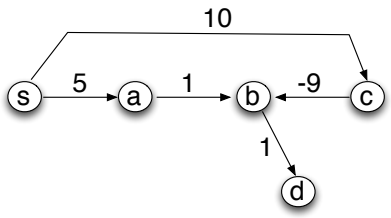
(c) Now run DFS starting at vertex C, but consider the edges in the adjacency lists in in reverse alphabetical order.

- What do you get when you order the vertices by ascending start time?
- 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. Consider the directed acyclic graph below and assume you want to compute SSSP(s).



   Run the iterative algorithm for computing SSSP(s) on a DAG that we discussed in class. Show the order in which the edges are relaxed, and show how the distances $d[x]$ change after each relaxation. What is the running time of this algorithm?

5. Consider a directed graph G, and assume that instead of shortest paths we want to compute *longest paths*. Longest paths are defined in the natural way, i.e. the longest path from u to v is the path of maximum weight among all possible paths from u to v. Note that if the graph contains a positive cycle, then longest paths are not well defined (for the same reason that shortest paths are not well defined when the graph has a negative cycle). So what we mean is the *longest simple path*, (a path is called *simple* if it contains no vertex more than once).

   Show that the the *longest simple path* problem does not have optimal substructure by coming up with a small graph that provides a counterexample. Note: Finding longest (simple) paths is a classical *hard* problem, and it is known to be NP-complete.