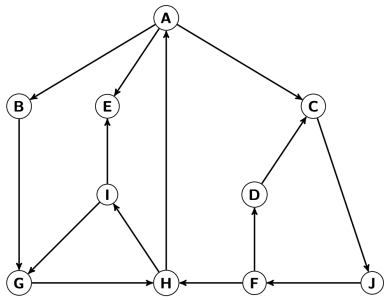
Lab 12
Graphs

Collaboration level 0 (no restrictions). Open notes.

1. Consider the graph below:

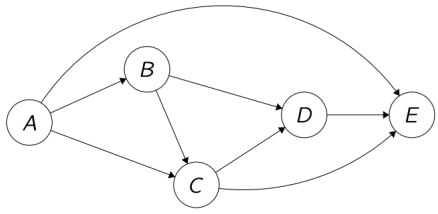


   (a) What are all the strongly connected components?

   (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 A,B, ...E 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 unvisited 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 descending finish time?

3. Explain why the following algorithmis not correct (i.e. the order produced is not a valid topological order):

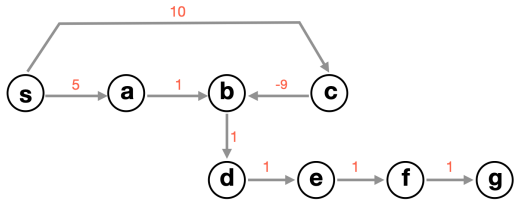
   Algorithm to order vertices topologically: Run BFS, and sort vertices by increasing distance to their respective sources.

   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 or if all edges weights were equal, we could simply run BFS form u; but BFS does not guarantee shortest paths when there are weights.

   (a) 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.



   Hint: Argue that shortest paths have optimal substructure, choose a subproblem, express the problem recursively and from here go to a recursive dynamic programming solution.

   (b) Run the algorithm from part (a) on the graph above; show how the recursion propagates, and how the shortest path from u is computed for each vertex. Consider how you'd write an algorithm to compute the shortest path from u to v without using recursion.