Graphs II. Depth-First Search

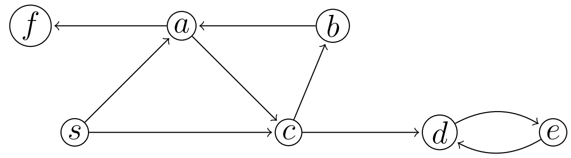

Figure 1: Graph H.

1. 1. Describe the strongly connected components of the graph H.

2. Perform depth-first search starting from vertex b.

3. Using the discovery and finish times found by depth-first search, find the strongly connected components (via the algorithm). Construct the condensate H' of the graph H.

4. Perform a topological sort of the graph H'.

2. Give a counterexample to the conjecture that if a directed graph G contains a path from u to v then any depth-first search must result in $d[v] < f[u]$.

3. A graph G has been traversed via DFS. The discovery and finish times of the vertices are stored in the arrays d and f. Construct an algorithm that, using only the data from the arrays d and f (and the description of the graph), checks whether the edge e of the graph G is a backword edge; is a tree edge. Note that the predecessor subgraph is unknown.

4. Construct an algorithm that checks if an undirected graph $G(V, E)$ is bipartite. A graph is bipartite if its vertices can be partitioned into sets $L, R : L \cup R = V, L \cap R = \varnothing$ such that each edge has endpoints in both L and R.

5. Professor Bacon claims that the algorithm for strongly connected components would be simpler if it used the original (instead of the transpose) graph in the second depth-first search and scanned the vertices in order of increasing finishing times. Does this simpler algorithm always produce correct results?

6. An Euler tour of a strongly connected, directed graph $G(V, E)$ is a cycle that traverses each edge of G exactly once, although it may visit a vertex more than once.

1. Show that G has an Euler tour if and only if $d_{in}(v) = d_{out}(v)$ for each vertex $v \in V$.

2. Describe an $O(|E|)$ algorithm to find an Euler tour of G if one exists. (Hint: Merge edge-disjoint cycles.)

7. Construct an algorithm that finds the shortest weighted paths from a vertex u to all vertices of a weighted directed acyclic graph (DAG) reachable from u and estimate its complexity. The input of the problem is a description of DAG G and a list of edges, each edge is given by a triple of integers (i, j, w): G has an edge from the vertex i to the vertex j of weight w. The length of the path from vertex u to v is the sum of the weights on the path from u to v.