Algorithms I – supervision 8

James Wood

May 7, 2015

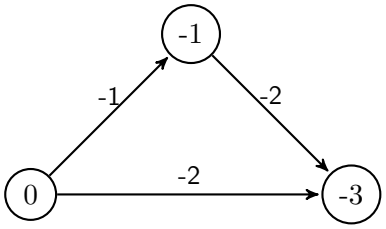
1 Graphs

(a) Start at a given vertex, assigning it a weight of 0, and add all edges leaving it to a priority queue ordered by the weight of the edge plus the weight of the vertex at their start (ascending). Remove edges from the queue until the taken edge ends at a vertex without an assigned weight. Give this vertex the weight stored as the key of the priority queue item and repeat the process until all vertices have been assigned a weight.

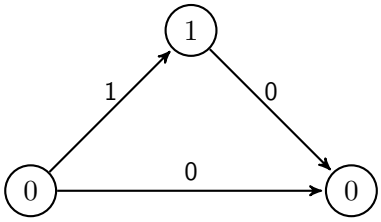
Dijkstra's algorithm is a greedy algorithm, so chooses the route that minimises weight, under the assumption that worse paths cannot be made better by adding more edges. This assumption does not hold in a general graph with negative-weight edges.

(b) A dummy vertex q is added to the graph, with edges going from it to each other vertex, each edge having weight 0. Then, the Bellman-Ford algorithm is run, starting at q. From this, we obtain a function $\delta : \text{Vertex} \to \text{Weight}$ describing how far the given vertex is away from q. We then produce a new graph based on the original graph (with weight function w) with weight function $w' : \text{Vertex} \times \text{Vertex} \to \text{Weight}$, where $w'(u, v) = w(u, v) + \delta u - \delta v$. Since $\delta v \leq \delta u + w(u, v)$, all edges in the new graph have positive weight. Hence, we can run Dijkstra's algorithm (which is quicker than Bellman-Ford) on each vertex.

(c) Consider this graph:



The shortest path from left to right goes via the middle node. Adding 2 to the weight of each arc gives this:

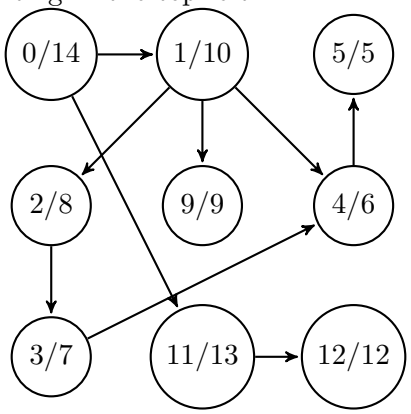


The shortest path from left to right is now to go there directly. Hence, this tactic does not work.

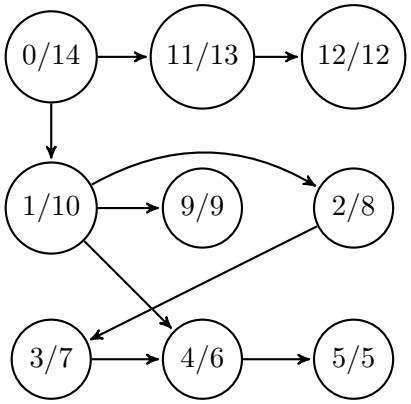
(d) A connected directed graph can be created such that certain notes are unreachable from others. In this case, starting from an existing node, in the worst case, won't assign weights to nodes, as needed for the reweighting of edges. Hence, Johnson's algorithm will fail.

2 DAG

Starting in the top-left:



This gives (read left-to-right, then top-to-bottom):



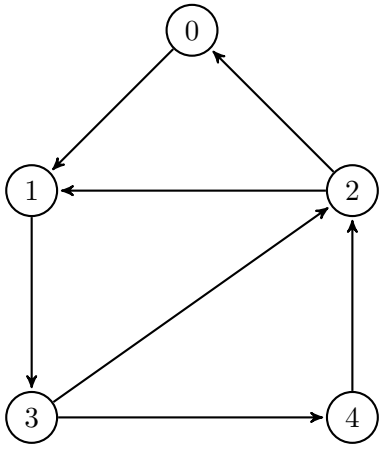
And indeed, the arrows do go forward.

3 Topological sort

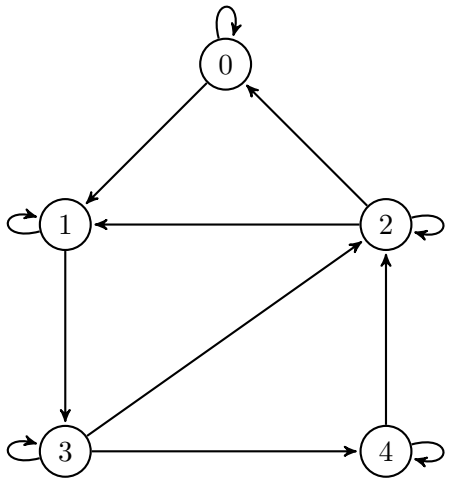
After a correctly computed topological sort, item A precedes item B if there is a path from A to B. After running the topological sort algorithm, A precedes B iff A's finish time is later than B's, according to a depth-first search. Hence, we are required to prove that if there is a path from A to B, A's finish time is later than B's. By definition, using the fact that A has children, A finishes once all of its children have finished. The child containing B (if B is not a child) will finish after B, and so on. Hence, A finishes after B, as required.

4 Draw graphs

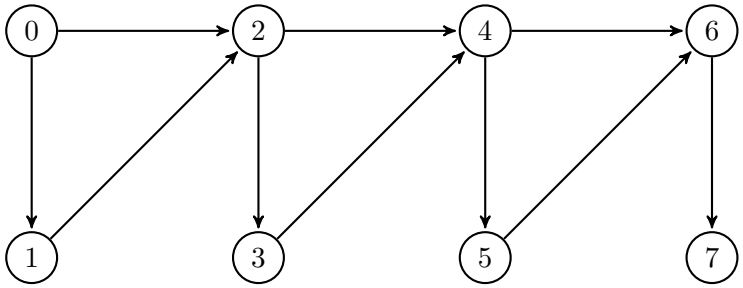
(a)



(b)



(c)



(d) In a tree, $|V| = |E| + 1$.

(e) A tree is defined as an undirected graph without cycles.

(f) 

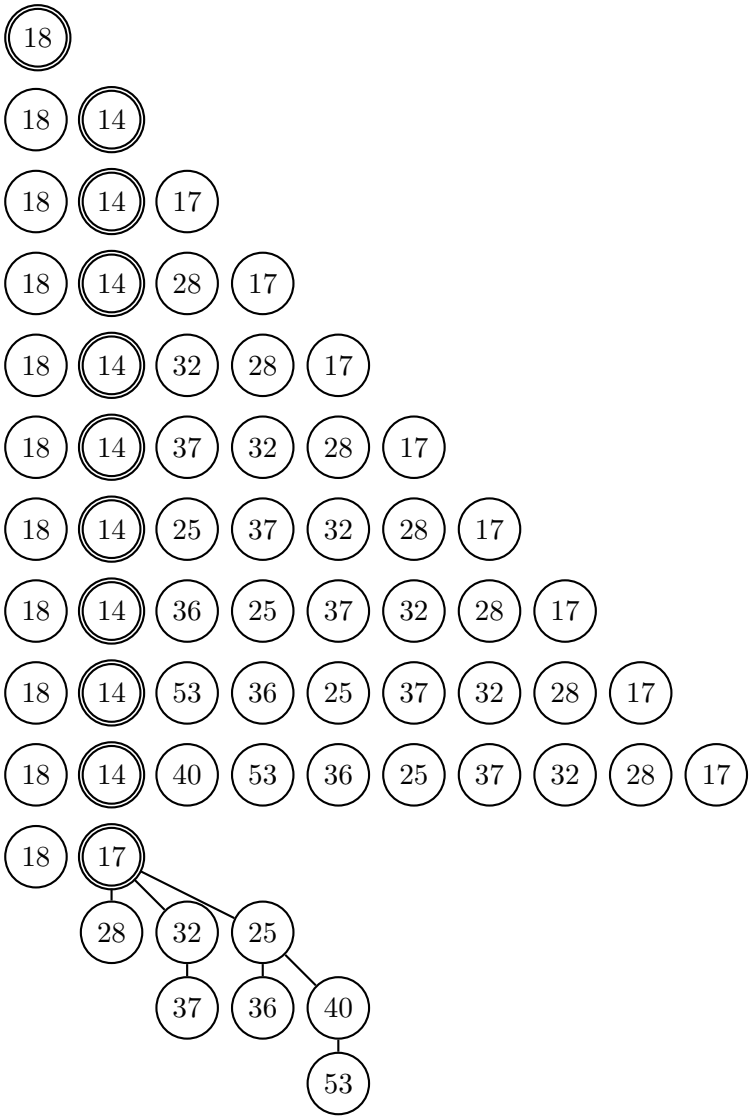
5 Matrices

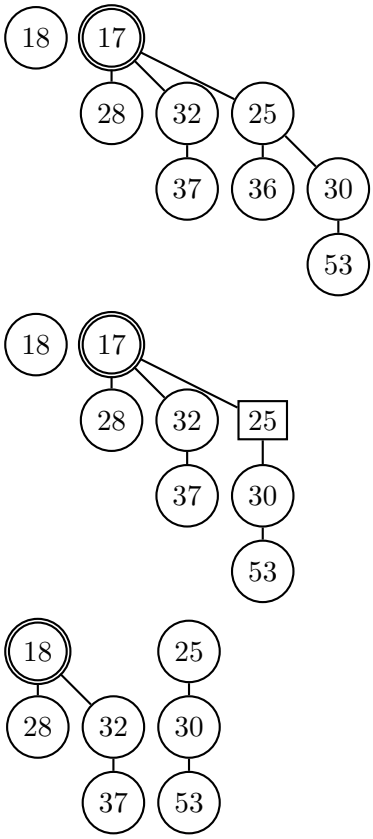
(a) $L^{(0)} = \begin{pmatrix} 0 & \infty & \dots & \infty \\ \infty & 0 & \dots & \infty \\ \vdots & \vdots & \ddots & \vdots \\ \infty & \infty & \dots & 0 \end{pmatrix}$

(b) $L^{(0)}$ acts like $\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$, the identity matrix.

6 Fibonacci heaps

(a)





(b) ?