組別: 姓名: 學號: 分數: 批改者:

一人寫一份，可討論。上台講解且正確者，全組自行加一分。

Algorithms

Classwork # Video 7.1-7.4

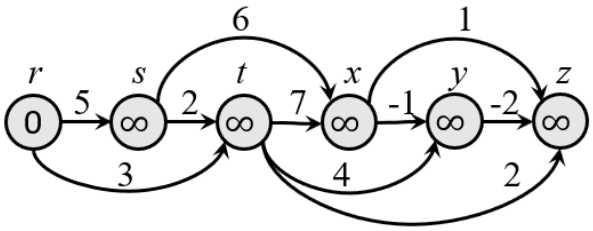
Each problem is 3 points.

3 = correct. 2 = one mistake. 1 = many mistakes. 0 = no answer.

1. Single Source Shortest Paths in Directed Acyclic Graph [7.4]

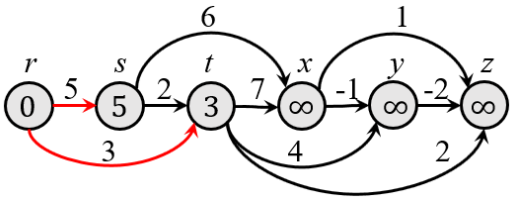
Run DAG-SHORTEST-PATH on the directed graph of the following figure, using vertex *r* as the source.

Note: Please show your procedure **step-by-step**, as figures on p.4 – p.7 of Lecture 7.4. Please always relax adjacent vertices in topological order.

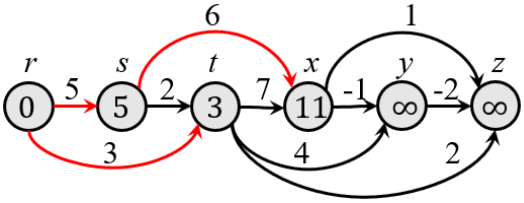


Sol:

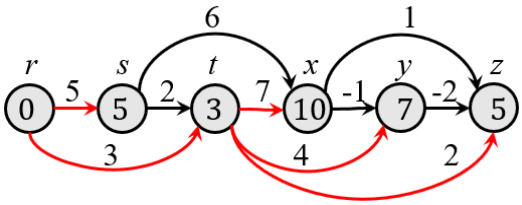
1



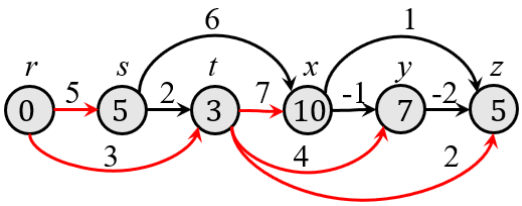
2



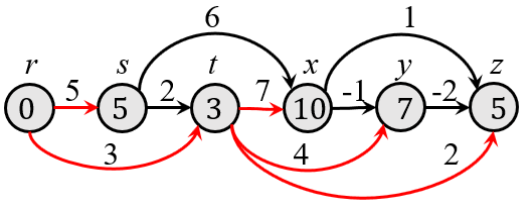
3



4



5



2. Dijkstra's Algorithm Verification [7.3, 7.2]

Professor Gaedel has written a program that he claims to implement Dijkstra's algorithm. The program produces $v.d$ and $v.\pi$ for each vertex $v \in V$. As his research assistant, you should help check if professor's program is right, *i.e.*, verify that d and π attributes for each vertex v match those of some shortest-path tree. Please fill the blanks in the procedures shown below to check the program, and make sure the time complexity of your verification program is $O(V + E)$. You may assume that all edge weights are non-negative.

- ***a.*** Verify that $s.d = 0$ and $s.\pi =$ **NIL**.
- ***b.*** Verify that $v.d = v.\pi.d + w(\square, \square)$ for all $v \neq s$. Note that $w(u, v)$ is the weight of edge from vertex u to v.
- ***c.*** Verify that $v.d = \infty$ iff $v.\pi = \square$ for all $v \neq s$
- ***d.*** Continued to problem 3

3. Continued from problem 2

If any one of the above verification fails, declare the program to be incorrect. Otherwise, please fill the folllowing blank to verify the result is a shortest-path tree. (HINT: see the final check in Bellman-Ford algorithm)

Sol:

a. Verify that s. d = 0 and s. π = NIL

b. Verify that v. d = v. π. d + w(v. π, v) for all v ≠ s

c. Verify that v. d = ∞ if and only if v. π = NIL for all v ≠ s

d. If any of above verification tests fail, declare the output to be incorrect. Otherwise, run one pass of Bellman-Ford, i.e. relax each edge(u,v) ∈ E one time. If any values of v. d changes, then declare the output to be incorrect; otherwise, declare the output to be correct.

4. Dijkstra's Algorithm Verification (con't) [7.3]

Professor Newman thinks that he has worked out a simpler proof of correctness for Dijkstra's algorithm. He claims that Dijkstra's algorithm relaxes the edges of every shortest path in the graph in the order in which they appear on the path, and therefore the path-relaxation property applies to every vertex reachable from the source. Show that the professor is mistaken by constructing a directed graph for which Dijkstra's algorithm could relax the edges of a shortest path out of order.

Sol:

1. Let the graph have vertices s, x, y, t and edges (s, x), (s, y), (x, t), (y, x), and let every edge have weight 0.
2. Dijkstra's algorithm could relax edge in the order (s, x), (s, y), (x, t), (y, x).
3. In the graph, there are two shortest paths with weight 0 from s to t: <s, x, t> and <s, y, x, t>.
4. The shortest path <s, y, x, t> are relaxed out of order because (y, x) is relax after (x, t).

5. Arbitrage [7.2]

Arbitrage is the use of discrepancies in currency exchange rates to transform one unit of a currency into more than one unit of the same currency. For example, suppose that 1 U.S. dollar buys 49 Indian rupees, 1 Indian rupee buys 2 Japanese yen, and 1 Japanese yen buys 0.0107 U.S. dollars. Then, by converting currencies, a trader can start with 1 U.S. dollar and buy $49 \times 2 \times 0.0107 = 1.0486$ U.S. dollars, thus turning a profit of 4.86 percent.

Suppose that we are given n currencies $c_1, c_2, \ldots, c_n$ and an $n \times n$ table R of exchange rates, such that one unit of currency c_i buys $R = [i, j]$ units of currency c_j.

Give an efficient algorithm to determine whether or not there exists a sequence of currencies $\langle c_{i_1}, c_{i_2}, \ldots, c_{i_k} \rangle$ such that

$\mathrm{R}[i_1, i_2] \cdot \mathrm{R}[i_2, i_3] \cdots \mathrm{R}[i_{k-1}, i_k] \cdot \mathrm{R}[i_k, i_1] > 1$

Analyze the running time of your algorithm.

Sol:

We can use the Bellman-Ford algorithm on a suitable weighted, directed graph $\mathrm{G} = (V, E)$, which we form as follows. There is one vertex in V for each currency, and for each pair of currencies c_i and c_j, there are directed edges (v_i, v_j) and (v_j, v_i). (Thus, $|V| = n$ and $|E| = \binom{n}{2}$.)

To determine edge weights, we start by observing that
$\mathrm{R}[i_1, i_2] \cdot \mathrm{R}[i_2, i_3] \cdots \mathrm{R}[i_{k-1}, i_k] \cdot \mathrm{R}[i_k, i_1] > 1$
if and only if
$$\frac{1}{\mathrm{R}[i_1, i_2]} \cdot \frac{1}{\mathrm{R}[i_2, i_3]} \cdots \frac{1}{\mathrm{R}[i_{k-1}, i_k]} \cdot \frac{1}{\mathrm{R}[i_k, i_1]} < 1$$
Taking logs of both sides of the inequality above, we express this condition as
$$\lg \frac{1}{\mathrm{R}[i_1, i_2]} + \lg \frac{1}{\mathrm{R}[i_2, i_3]} \cdots \lg \frac{1}{\mathrm{R}[i_{k-1}, i_k]} + \lg \frac{1}{\mathrm{R}[i_k, i_1]} < 0$$
Therefore, if we define the weight of edge (v_i, v_j) as
$$\mathrm{w}(v_i, v_j) = \lg \frac{1}{R[i, j]} = -\lg R[i, j]$$
then we want to find whether there exists a negative-weight cycle in *G* with these edge weights.

We can determine whether there exists a negative-weight cycle in *G* by adding an extra vertex v_0 with 0-weight edges (v_0, v_i) for all $v_i \in V$, running BELLMAN-FORD from v_0, and using the boolean result of BELLMAN-FORD (which is TRUE if there are no negative-weight cycles and FALSE if there is a negative-weight cycle) to guide our answer. That is, we invert the boolean result of BELLMAN-FORD.

This method works because adding the new vertex v_0 with 0-weight edges from v_0 to all other vertices cannot introduce any new cycles, yet it ensures that all negative-weight cycles are reachable from v_0.

It takes $\Theta(n^2)$ time to create *G*, which has $\Theta(n^2)$ edges. Then it takes $O(n^3)$ time to run BELLMAN-FORD. Thus, the total time is $O(n^3)$.

Another way to determine whether a negative-weight cycle exists is to create *G* and, without adding v_0 and its incident edges, run either of the all-pairs shortest-paths algorithms. If the resulting shortest-path distance matrix has any negative values on the diagonal, then there is a negative-weight cycle.