Shortest paths

Algorithm for DAGs [unknown]

Recall: DAG = Directed Acyclic Graph.

Recall: A topological sort can be found via DFS in time O(n+m).

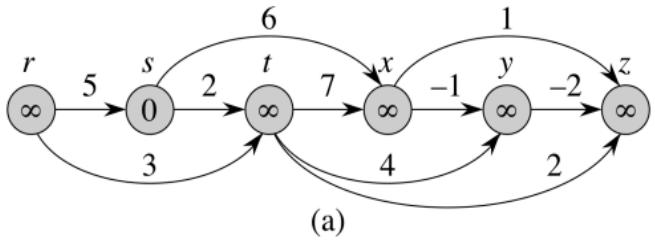
```
DAG-SHORTEST-PATHS(G, w, s)
  topologically sort the vertices
  INIT-SINGLE-SOURCE(G, s)
  for each vertex u, taken in topologically sorted order
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Running time: O(n+m).

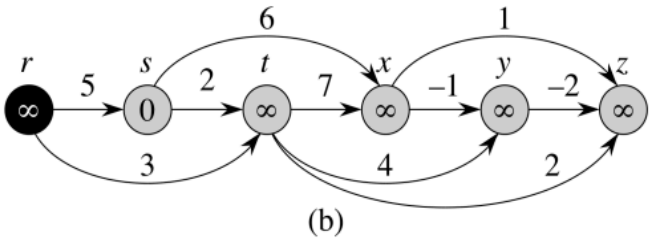
Sentence: When the algorithm stops: v.d = δ(s, v) for all v ∈ V.

Proof: For a node v for which there is a path from s to v, all nodes on some shortest path have been relaxed in order—thereby assigning the correct δ-values along that path by the Path-Relaxation Lemma. For every other node, δ(s, v) = ∞, so correctness in those cases follows from the fact that δ(s, v) ≤ v.d.

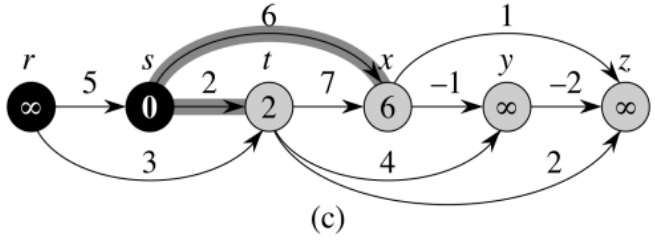
The algorithm for DAG, example



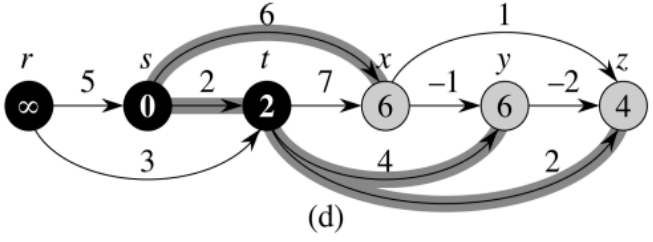
(a)



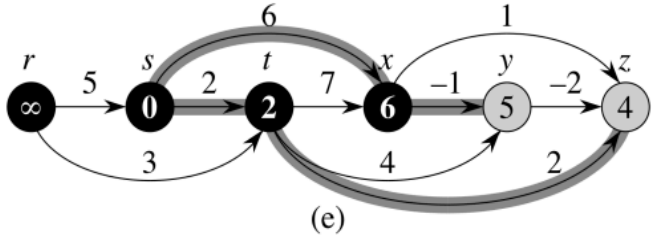
(b)



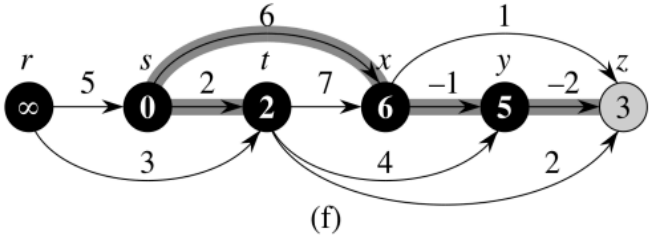
(c)



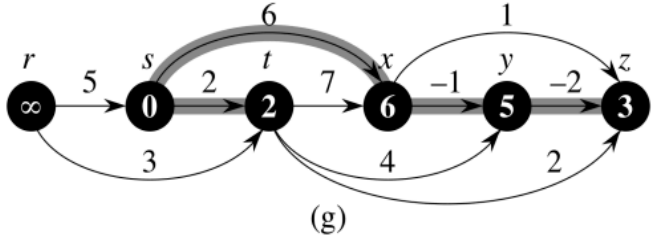
(d)



(e)



(f)



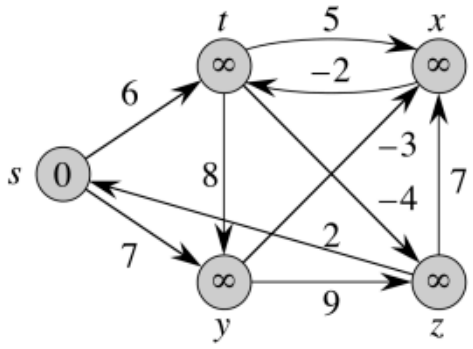
(g)

Bellman-Ford-Moore [1956-57-58]

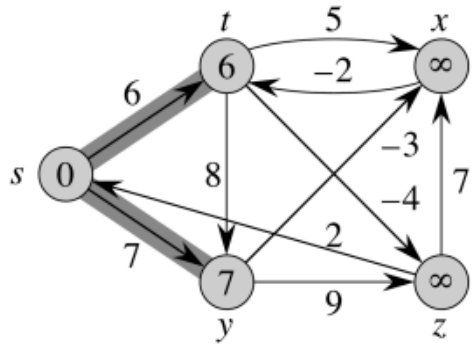
```
BELLMAN-FORD(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

Running time: O(nm)

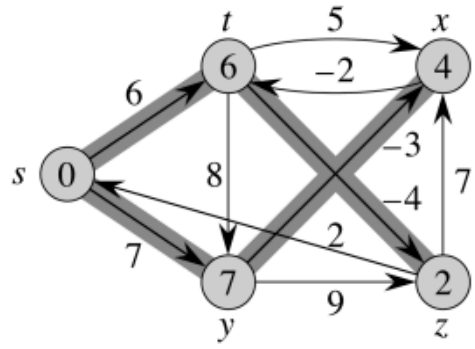
Bellman-Ford-Moore [1956-57-58]



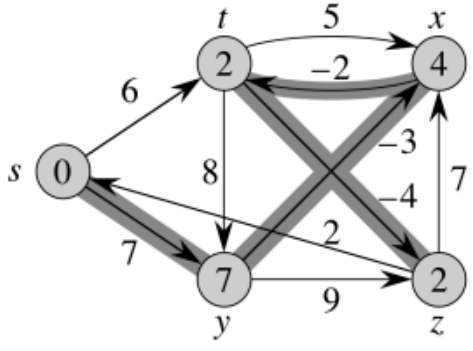
(a)



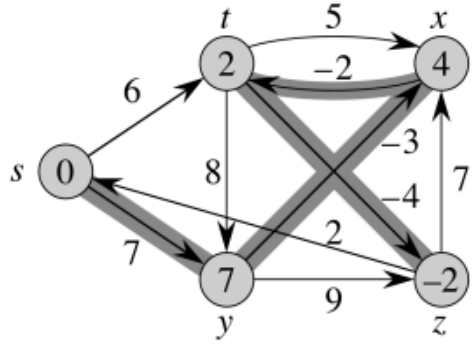
(b)



(c)



(d)



(e)

Relaxation order:
(t, x),(t, y),(t, z),(x, t),(y, x),(y, z),(z, x),(z, s),(s, t),(s, y).

Bellman-Ford, correctness

The idea behind Bellman-Ford is that it maintains the following invariant:

> After i iterations of the first **for-**loop v.d = δ(s, v) holds for every vertex v that has a shortest path from s of at most i edges.

This invariant follows directly from the path-relaxation lemma. More precisely, the following holds for Bellman-Ford:

Sentence: If there is a negative circuit that can be reached from s, Bellman-Ford returns FALSE. Otherwise it returns TRUE, and v.d=δ(s, v) for all $v \in V$ when it stops.

Shortest paths between all pairs of nodes

All-pairs shortest path problem: find $\delta(s, v)$ for all $s, v \in V$. (and an explicit path) for all $v \in V$.

* One option: run Dijkstra from each source $v \in V$ (requires non-negative weights): $O(n\ m \log n)$ time.

* Or: run Bellman–Ford–Moore from each source $v \in V$ (if there are negative weights): $O(n^2\ m)$ time.

* Another option: Floyd–Warshall's algorithm: $O(n^3)$ time, handles negative weights.

* Yet another option: Johnson's algorithm: $O(n\ m \log n)$ time, handles negative weights.

Floyd-Warshall's algorithm [1962]

Based on dynamic programming.

Uses the adjacency matrix representation (in variant with weights on edges).

Output is also in matrix form:

$D = (d_{ij})$, $d_{ij} = \delta(v_i, v_j)$ = the length of a shortest path from v_i to v_j. Set to ∞ if no path is found.

$\Pi = (\pi_{ij})$, π_{ij} = last vertex before v_j on a shortest path from node v_i to vertex v_j. Set to NIL if no path is found.

Floyd-Warshall's algorithm

(Only construction of the D-matrix is shown)

```
FLOYD-WARSHALL(W, n)
  D^(0) = W
  for k = 1 to n
      let D^(k) = (d_ij^(k)) be a new n × n matrix
      for i = 1 to n
          for j = 1 to n
              d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
  return D^(n)
```

Running time: $O(n^3)$. Space:$O(n^2)$ (only previous matrix $D^{(k)}$ needs to be saved).

Sentence: When the algorithm stops d_{ij} and π_{in} set correctly for all $v_i, v_j \in V$ (if there is no negative cycle in the graph).

Proof: The invariant is that $D^{(k)}$ contains the length of shortest paths between v_i and v_j which passes the nodes $v_1, v_2, \ldots, v_k$ (beyond the end points v_i and v_j).

Johnson's algorithm [1977]

User:

- Runs Bellman-Ford-Moore once on a slightly extended graph.
- From here, adjust the edge weights so that all become positive without essentially changing the shortest paths (see lemma on the next page).
- Runs Dijkstra from all vertices.

Running in O(n m logn + n m) = O(n m logn) time, handles negative weights.

Re-weighting

Look at the situation where we are all knotted $v \in V$ assigns a number $\phi(v)$.

From ϕ can we make new weights $\tilde{w}$ in the graph in the following way:

$$\tilde{w}(u, v) = w(u, v) + \phi(v) - \phi(u).$$

Look at a path $v_1, v_2, \ldots, v_k$. Then applies:

$$\begin{aligned} \sum_{i=1}^{k-1} \tilde{w}(v_i, v_{i+1}) &= \sum_{i=1}^{k-1} \left(w(v_i, v_{i+1}) + \phi(v_{i+1}) - \phi(v_i) \right) \\ &= \sum_{i=1}^{k-1} w(v_i, v_{i+1}) + \left(\phi(v_k) - \phi(v_1) \right) \end{aligned}$$

since the sum is telescoping.

In other words, the path length under the new weights is equal to the path length under the old weights, up to an additive correction determined by the path's endpoints. That is, this correction is the same for every path from s(=v1) to t(=vk).

Therefore, the shortest path from s to t is identical under both w and $\tilde{w}$. Moreover, there is a negative cycle under w if and only if there is a negative cycle under $\tilde{w}$ (since in any cycle $v_k = v_1$, making $\phi(v_k) - \phi(v_1) = 0$)