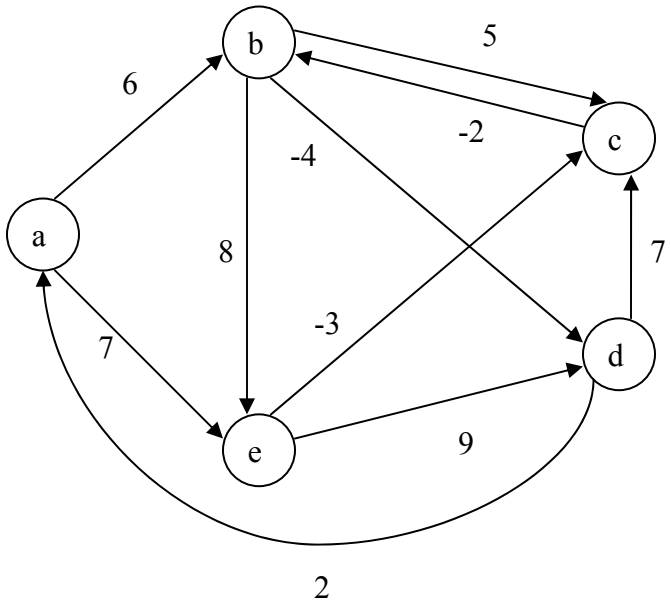
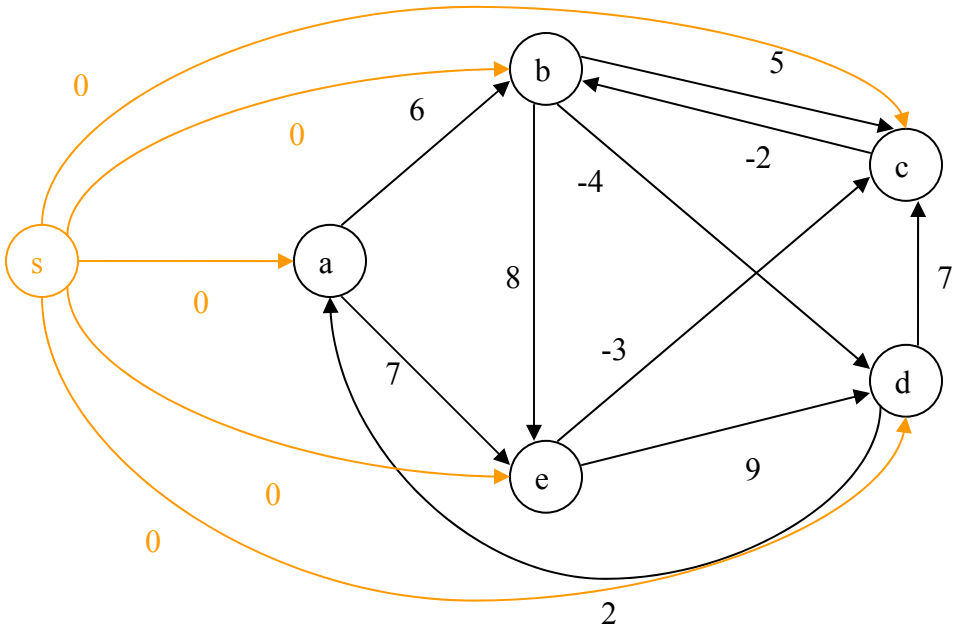
COMP 360: Algorithm Design Techniques
Tutorial given on February 18, 2004
Prepared by Michel Langlois

Example of Johnson's algorithm

Given a graph G = (V, E), with weighting function w:



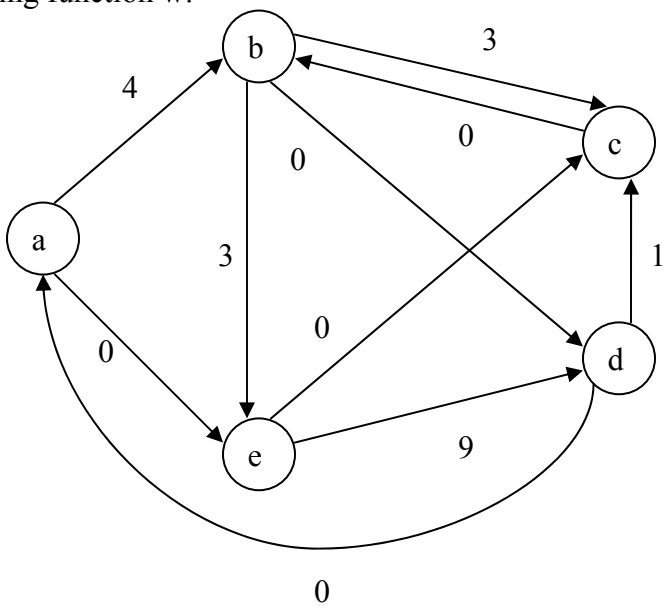
Step 1. Transform G into G' = (V', E'), where V' = V ∪ {s}, and E' = E ∪ {(s, v) with weight 0 s.t. v is in V}. Here is G':



Step 2. Find $h(v) = \delta(s, v)$, the shortest directed path between s and each vertex v, using Bellman-Ford algorithm. BF will also detect a negative weight cycle if there is one.

Vertex v	$h(v) = \delta(s, v)$	Through
a	-7	e, c, b, d
b	-5	e, c
c	-3	e
d	-9	e, c, b
e	0	none

Step 3. Reweight edges of G: $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$. Here is G with the new weighting function $\hat{w}$:



Step 4. For each vertex u of G:

Run Dijkstra's algorithm with source u to get $\delta'(u, v)$, the shortest directed path between u and v using the new weights.

Set $d_{uv} = \delta'(u, v) + h(v) - h(u)$.

Here is the result for u=a:

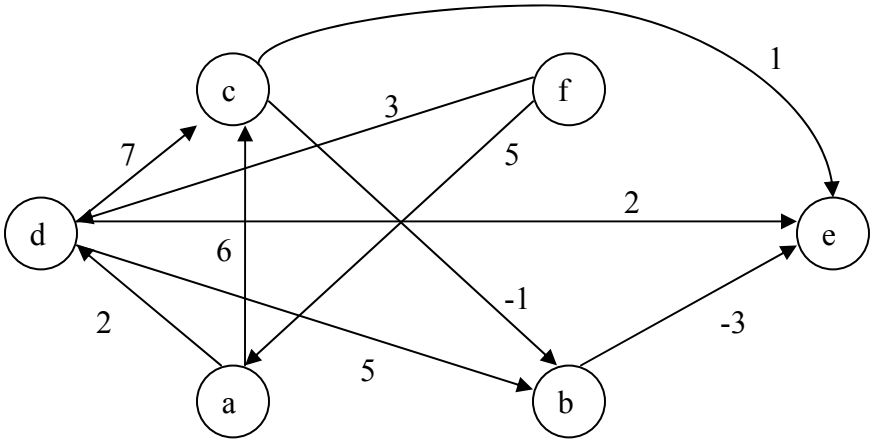
Vertex v	$\delta'(a, v)$	Through	$\delta'(a, v) + h(v) - h(a) = d_{av}$
a	0	none	0 + (-7) – (-7) = 0
b	0	e, c	0 + (-5) – (-7) = 2
c	0	e	0 + (-3) – (-7) = 4
d	0	e, c, b	0 + (-3) – (-7) = -2
e	0	none	0 + 0 – (-7) = 7

The column "Through" gives the shortest paths between vertex 'a' and every other vertex in G, as found by Dijkstra's algorithm. The column d_{av} gives the lengths of those paths.

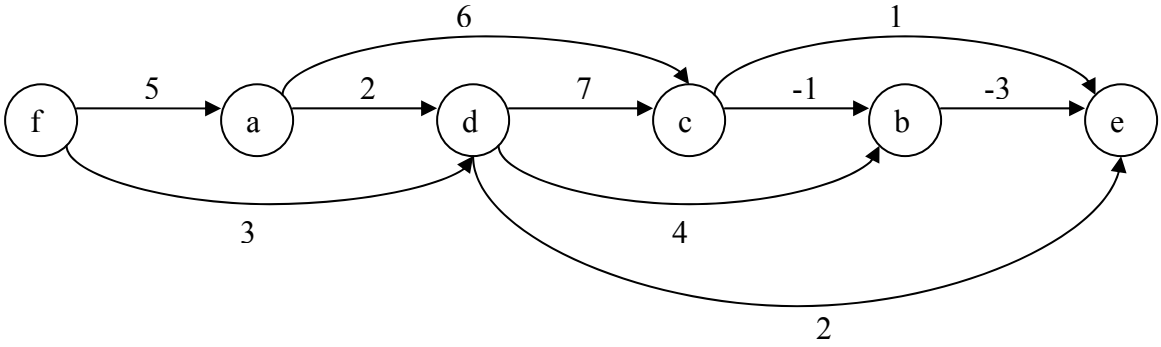
Johnson's algorithm runs in $O(V^2 \log V + V*E)$. On sparse graphs (i.e. E not too close to V^2), this is better than the Floyd-Warshall algorithm, which runs in $\theta(V^3)$.

Shortest paths on DAGs (directed acyclic graphs)

Here is a DAG:



Step 1. Topological sort of the vertices, such that vertex u is before v vertex iff there is an edge between u and v:



Step 2. Choose a source. I will use vertex f.

Step 3. Starting from the chosen source s, process each vertex v in sorted order:
Relax each edge leaving v

Remark: We can start processing from s because there are no paths from the source to any vertex that has been sorted before s. Also, we can omit the last vertex because it is either a sink or not connected to the rest of the graph.

Recall the rule for relaxing an edge (u,v):

If $d[v] > d[u] + w(u,v)$ then
$d[v] \leftarrow d[u] + w(u,v)$
$\pi[v] \leftarrow u$

In the following table, each column corresponds to processing one vertex. The first row indicates which vertex is being processed in each step. d indicates the length of the best known path between s and vertex v so far, and π is vertex v's predecessor in that path.

	Initially		s=f		a		d		c		b	
	d	π	d	π	d	π	d	π	d	π	d	π
s=f	0	Nil										
a	∞	Nil	5	s=f								
d	∞	Nil	3	**s=f**								
c	∞	Nil			11	a	10	d				
b	∞	Nil					7	**d**				
e	∞	Nil					5	d			**4**	**b**

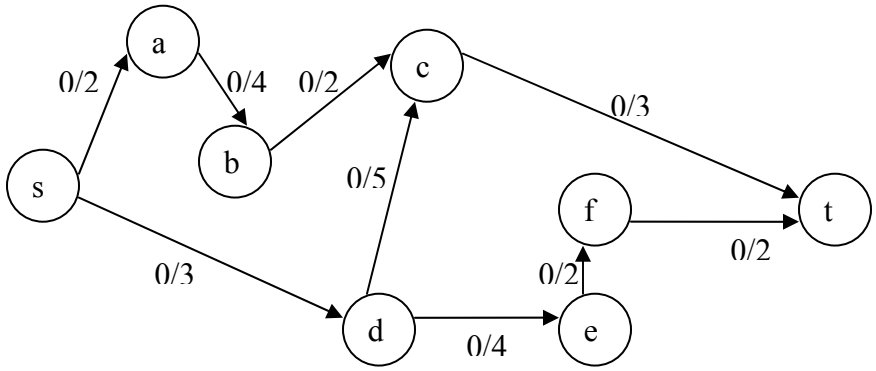
To find the shortest path between f and some vertex v, look up the rightmost numerical value on the row corresponding to v, and backtrack the chain of predecessors.

For example, the shortest path between f and vertex e is f→d→b→e, with length 4, and the relevant entries are shown in bold in the table.

Max flow problem using residual graph

This is not exactly the graph I worked on in the tutorial, but I want to show an example where we end up decreasing the flow along a back edge to augment the flow in the net.

Let's start with some graph G = (V, E) with capacitated edges:



Here are the rules for building the residual graph:

For each e = (u,v):

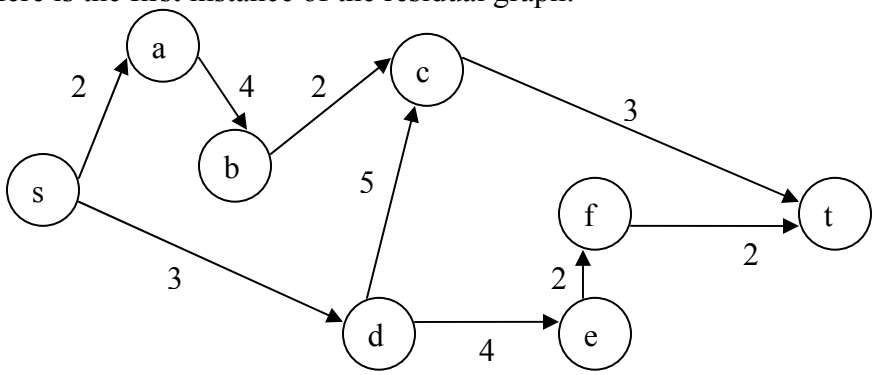
If cap(u, v) > flow(u, v) then

Add (u, v) to the residual graph with label cap(u, v) – flow(u, v)

If flow(u, v) > 0 then

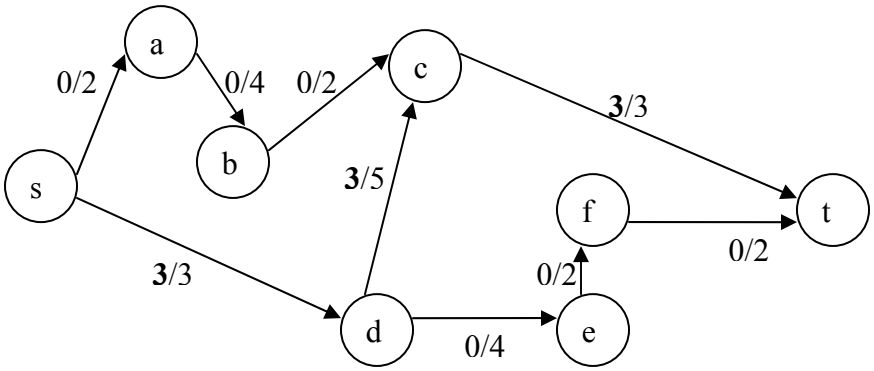
Add (v, u) to the residual graph with label flow(u, v)

Here is the first instance of the residual graph:

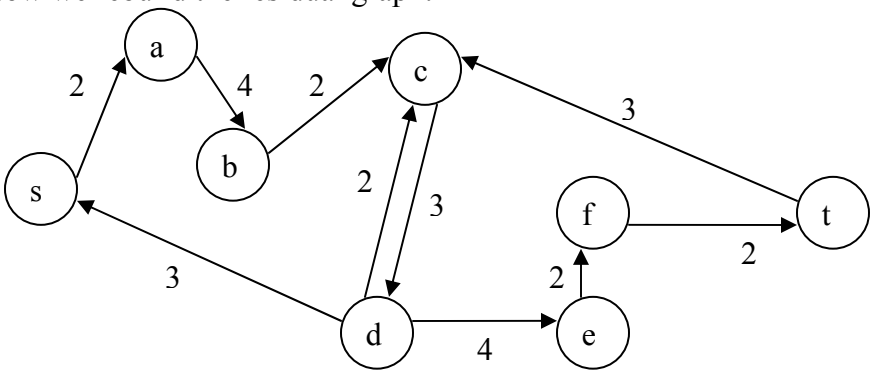


Using BFS, we look for the shortest (shortest in terms of number of segments) forward path from s to t in the residual graph. That path is s→d→c→t.

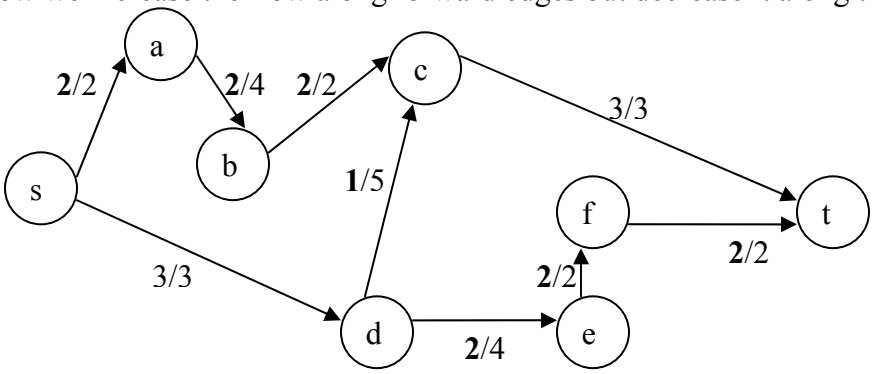
The minimum label along that path is 3, so that's the amount by which we'll increase the flow (the updated flows are shown in bold):



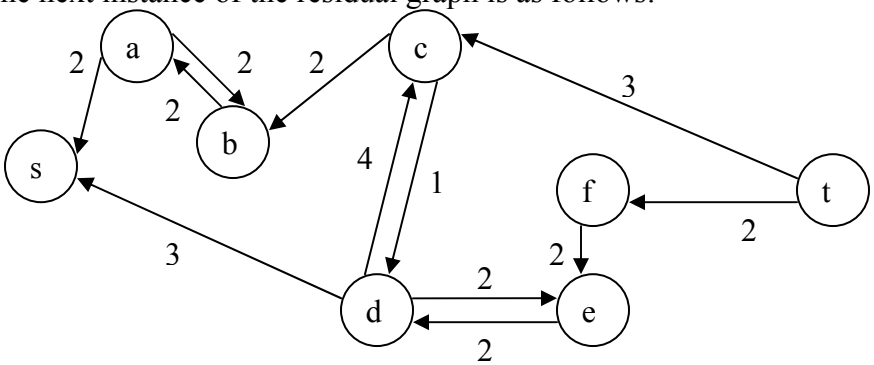
Now we rebuild the residual graph:



This time, the shortest (and only) forward path from s to t is s→a→b→c→d→e→f→t. The smallest label along that path is 2, that's by how much we increase the flow (notice how we increase the flow along forward edges but decrease it along the back edge):



The next instance of the residual graph is as follows:



This time, there is no forward path from s to t in the residual graph, so we are done. The max flow is the total flow leaving s in G, or the total flow arriving at t, 5 in both cases.