A Link-State Routing Algorithm

Dijkstra's algorithm

- net topology, link costs known to all nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- computes least cost paths from one node ('source") to all other nodes
  - gives *forwarding table* for that node
- iterative: after k iterations, know least cost path to k dest.'s

notation:

- c(x,y): link cost from node x to y; = ∞ if not direct neighbors
- D(v): current value of cost of path from source to dest. v
- p(v): predecessor node along path from source to v
- N': set of nodes whose least cost path definitively known

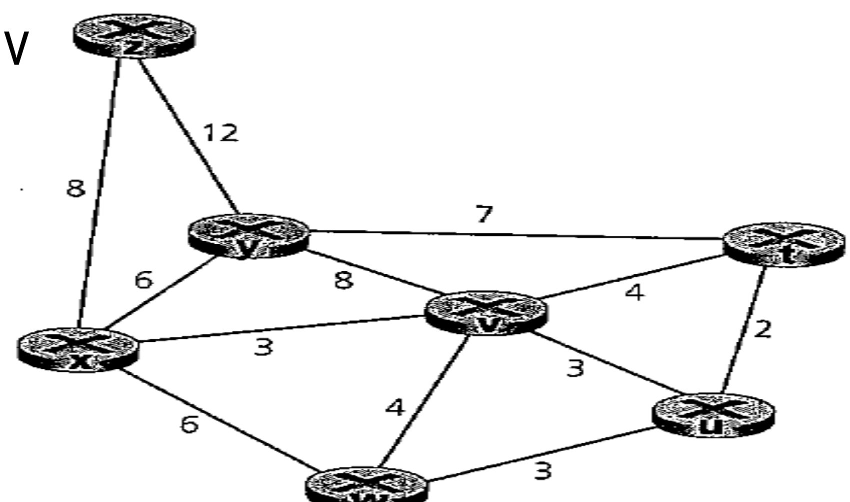
Dijsktra's Algorithm

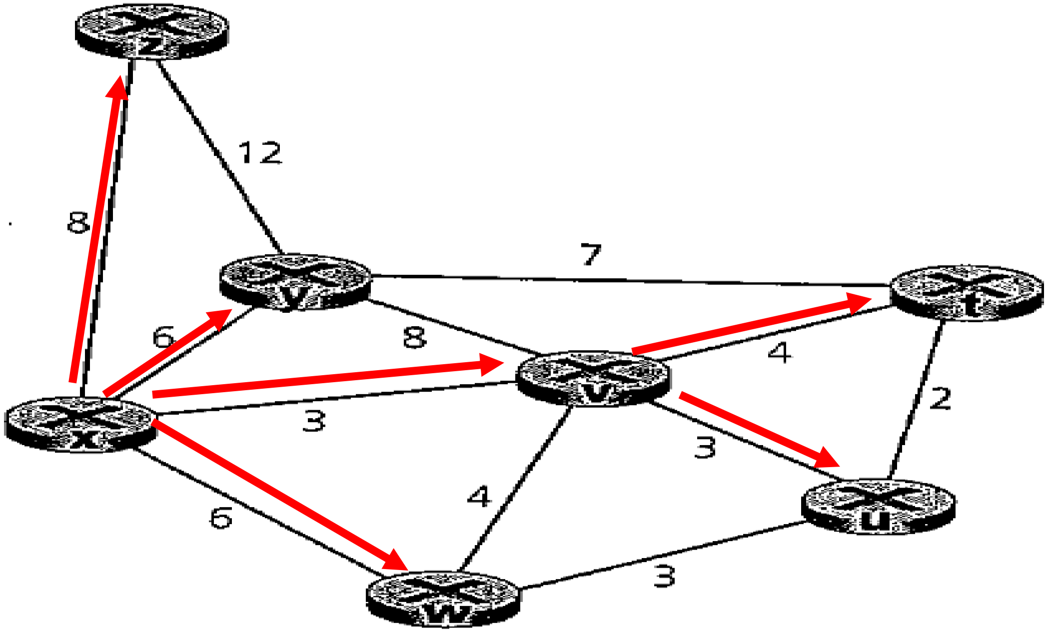
```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
        then D(v) = c(u,v)
    else D(v) = ∞

Loop
   find w not in N' such that D(w) is a minimum
   add w to N'
   update D(v) for all v adjacent to w and not in N' :
      D(v) = min( D(v), D(w) + c(w,v) )
   /* new cost to v is either old cost to v or known
    shortest path cost to w plus cost from w to v */
until all nodes in N'
```

Playground: https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html

Prob.1

Step	*N'*	*D(t),p(t)*	*D(u),p(u)*	*D(v),p(v)*	*D(w),p(w)*	*D(y),p(y)*	*D(z),p(z)*
0	x	∞	∞	3,x	6,x	6,x	8,x
1	xv	7,v	6,v		6,x	6,x	8,x
2	xvu	7,v			6,x	6,x	8,x
3	xvuw	7,v				6,x	8,x
4	xvuwy	7,v					8,x
5	xvuwyt						8,x
6	xvuwytz						




z
y
x
v
w
t
u
12
8
7
6
8
4
3
2
3
6
4
3

Distance vector algorithm

Bellman-Ford equation (dynamic programming)

let

$d_x(y)$:= cost of least-cost path from x to y

then

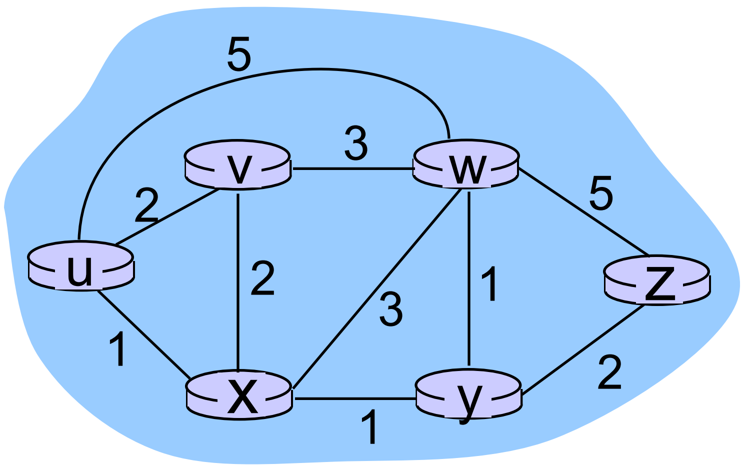
$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

cost from neighbor v to destination y

cost to neighbor v

min taken over all neighbors v of x

Bellman-Ford example



clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$
\begin{aligned}
d_u(z) = \min \{ & c(u,v) + d_v(z), \\
& c(u,x) + d_x(z), \\
& c(u,w) + d_w(z) \} \\
= \min \{ & 2 + 5, \\
& 1 + 3, \\
& 5 + 3\} = 4
\end{aligned}
$$

node achieving minimum is next
hop in shortest path, used in forwarding table

Distance vector algorithm

- Distance Vector (DV) Definition
  - $D_x(y)$ = Node x's *estimate* of least-cost path from x to y
  - x maintains DV $\mathbf{D}_x = [D_x(y): y \in N]$
- Node x's Local Knowledge
  - knows direct link cost to each neighbor v: $c(x,v)$
  - Maintains received DVs from neighbors: For each neighbor v, x maintains $\mathbf{D}_v = [D_v(y): y \in N]$
- Update Rule (Bellman-Ford Relaxation)
  - On receiving new $\mathbf{D}_v$ from neighbor

    $D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\}$ *for each node* $y \in N$
  - Periodically broadcasts own D_x to neighbors
- Convergence: Under stable topology & positive costs: Estimates $D_x(y)$ converge to true distances $d_x(y)$

Distance vector algorithm

Why Not Dijkstra?

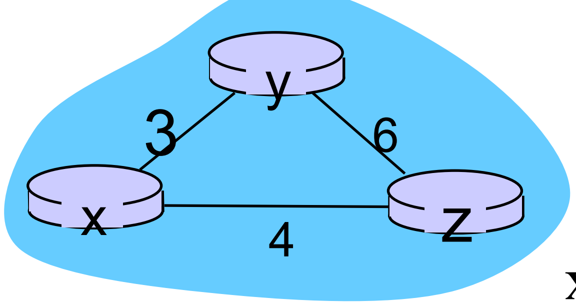
- Dijkstra's is a centralized algorithm run by a single source node with complete graph knowledge (all edges and weights).
- In DV, nodes can't afford to collect/store the entire topology—it's inefficient and unscalable for large, dynamic networks.
  - Running Dijkstra per node would require flooding the full graph, causing massive overhead.

- The DV approach, is a **distributed**, **asynchronous** implementation of the core B-F relaxation principle.
  - Work in *any order* and converge after $O(|E|)$iterations (or passes over edges)
  - **Handling Network Dynamics:** Routing networks change (links fail, costs vary), so DV/B-F supports incremental updates: A single neighbor's DV change propagates gradually, reconverging efficiently.

Dijskstra VS BF

Aspect	Bellman-Ford (for DV)	Dijkstra's (Not Suitable for DV)
Execution Model	Distributed, local relaxes	Centralized, global priority queue
Knowledge Needed	Neighbors' vectors only	Full graph topology
Update Style	Async, periodic broadcasts	Synchronous, one-shot computation
Convergence	Iterative to optimum (under mild cond.)	Immediate, but requires full restart
Overhead	Low (O(|E|) per update round)	High in distributed (needs topology sync)
DV Fit	DV is distributed B-F	Poor, shine in centralized, offline settings

P.2



Node x table

Cost to

		x	y	z
From	x	0	3	4
	y	∞	∞	∞
	z	∞	∞	∞

Cost to

		x	y	z
From	x	0	3	4
	y	3	0	6
	z	4	6	0

Node y table

Cost to

		x	y	z
From	x	∞	∞	∞
	y	3	0	6
	z	∞	∞	∞

Cost to

		x	y	z
From	x	0	3	4
	y	3	0	6
	z	4	6	0

Node z table

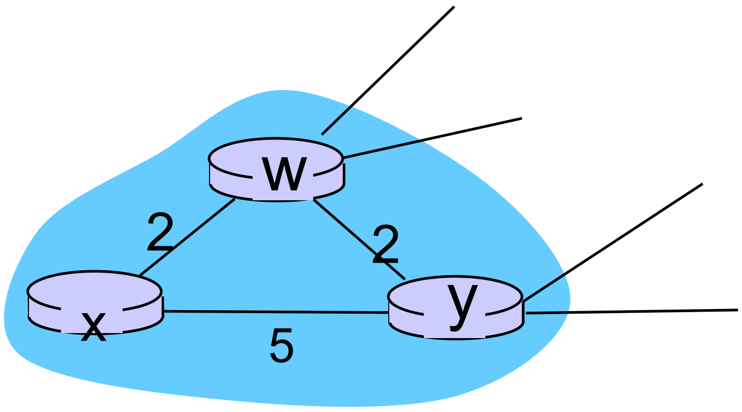
Cost to

		x	y	z
From	x	∞	∞	∞
	y	∞	∞	∞
	z	4	6	0

Cost to

		x	y	z
From	x	0	3	4
	y	3	0	6
	z	4	6	0

P.3



- Dx(w) = 2, Dx(y) = 4, Dx(u) = 7