Dijkstra's Algorithm

Iterative algorithm to build shortest path tree rooted at arbitrary source node, S.

- Let nodes be n_1, n_2, …, n_L and let $A = \{n_1, n_2, \ldots, n_L\}$.
- Define $\mathbf{c}$: $A \times A \rightarrow \Re$:
    - if n_i and n_j are neighbours, $\mathbf{c}(n_i, n_j)$ is the cost associated with the link connecting n_1 and n_2;
    - if n_i and n_j are not neighbours, $\mathbf{c}(n_i, n_j) = \infty$.
- At each step node closest to S not yet in the tree is added.
- At end iteration k:
    - let $D_k(n_i)$ be the best distance known from S to n_i ;
    - N_k is set of nodes included in the tree.

Initial step: Set $N_0 = \{S\}$ and $D_0(x) = \mathbf{c}(S,x)$.

Iterative step: For $k=1, \ldots, L-1$,

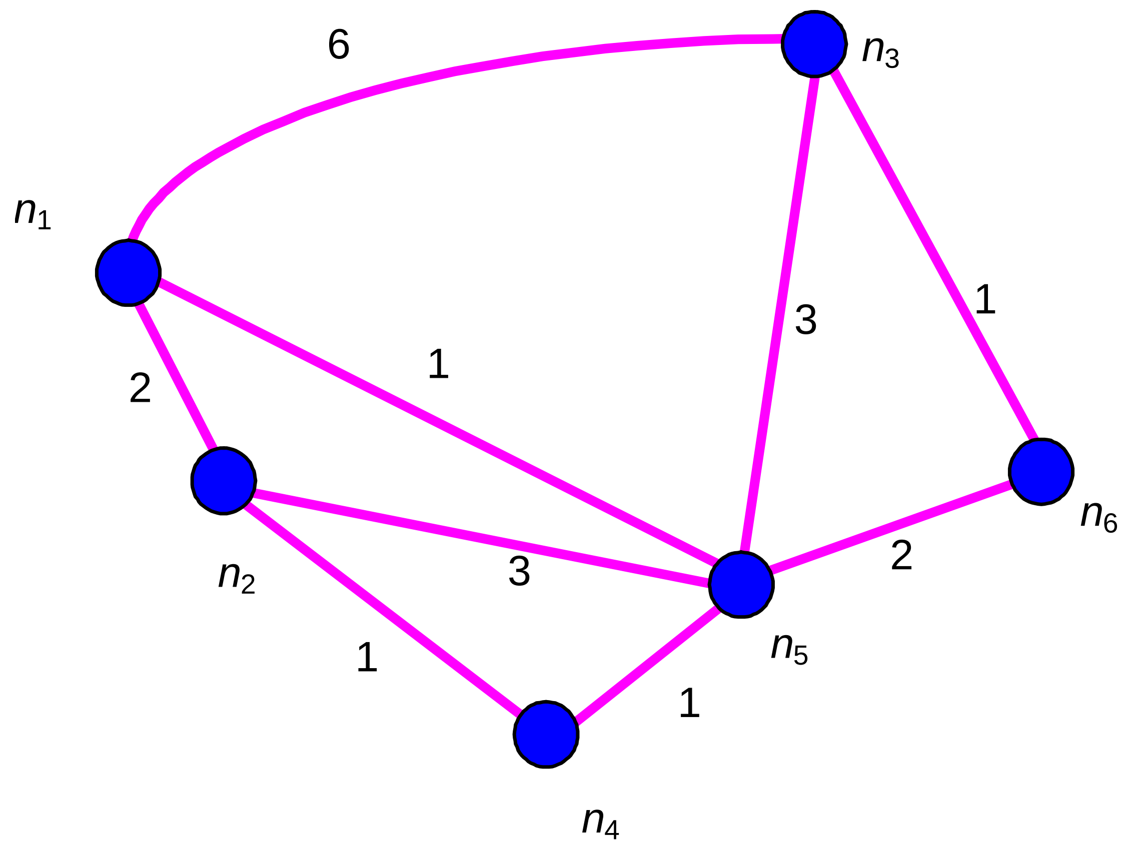
Find a node $y \notin N_{k-1}$ s.t. $D_{k-1}(y)$ is a minimum and set

$N_k := N_{k-1} \cup \{y\}$

If $k = L-1$ then ***stop***, otherwise:

$$D_k(x) := \min[D_{k-1}(x),\ D_{k-1}(y) + \mathbf{c}(y,x)] \quad \forall x \notin N_k$$

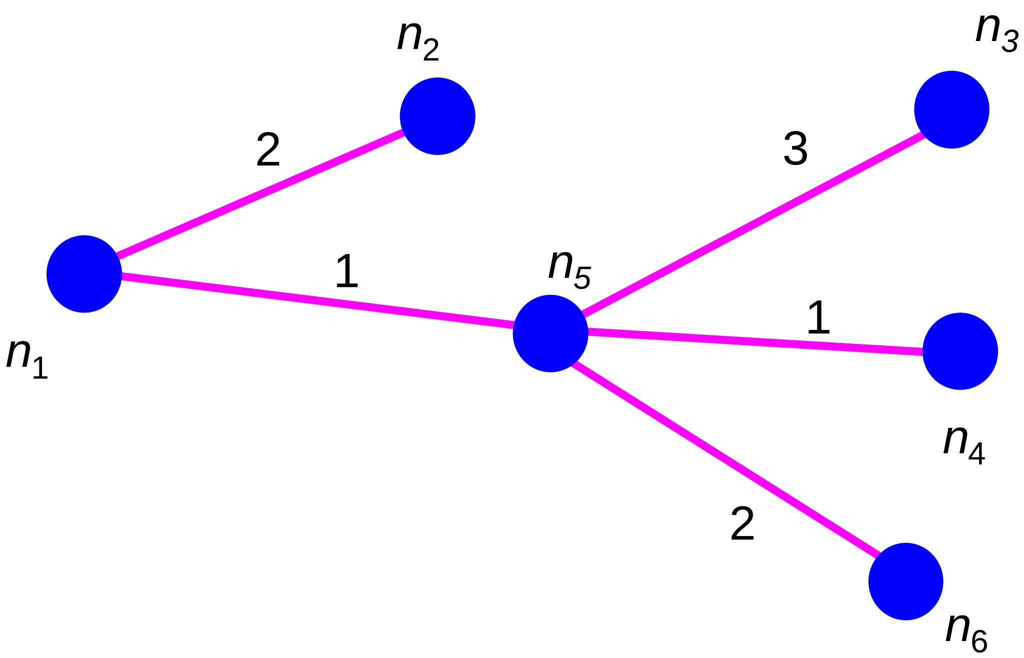
Example



k=?	N_k	n_2	n_3	n_4	n_5	n_6
Init	$\{n_1\}$	2 $(n_1\text{-}n_2)$	6 $(n_1\text{-}n_3)$	∞	**1 $(n_1\text{-}n_5)$**	∞
1	$\{n_1,n_5\}$	**2 $(n_1\text{-}n_2)$**	4 $(n_1\text{-}n_5\text{-}n_3)$	2 $(n_1\text{-}n_5\text{-}n_4)$		3 $(n_1\text{-}n_5\text{-}n_6)$
2	$\{n_1,n_5,n_2\}$		4 $(n_1\text{-}n_5\text{-}n_3)$	**2 $(n_1\text{-}n_5\text{-}n_4)$**		3 $(n_1\text{-}n_5\text{-}n_6)$
3	$\{n_1,n_5,n_2,n_4\}$		4 $(n_1\text{-}n_5\text{-}n_3)$			**3 $(n_1\text{-}n_5\text{-}n_6)$**
4	$\{n_1,n_5,n_2,n_4,n_6\}$		**4 $(n_1\text{-}n_5\text{-}n_3)$**			
5	$\{n_1,n_5,n_2,n_4,n_6,n_3\}$					

Example

Shortest Path Tree



Simplified routing table for n_1

Destination	Outgoing link	Distance
n_2	n_1—n_2	2
n_3	n_1—n_5	4
n_4	n_1—n_5	2
n_5	n_1—n_5	1
n_6	n_1—n_5	3

Bellman-Ford Algorithm

Notation. Let x and y be arbitrary nodes.

Neigh(x) is the set of neighbours of x,

$\ell(x,y)$ is the *length* of the link joining neighbours x and y.

x, maintains a distance vector, $\mathbf{D_x}$, of best known distances to every other node, and a routing vector, $\mathbf{f_x}$, of best known output links to every other node.

$$\mathbf{D_x} = \begin{bmatrix} d_1(x) \\ d_2(x) \\ . \\ . \\ . \\ d_{n(x)} \end{bmatrix}; \qquad \mathbf{f_x} = \begin{bmatrix} f_1(x) \\ f_2(x) \\ . \\ . \\ . \\ f_{n(x)} \end{bmatrix}$$

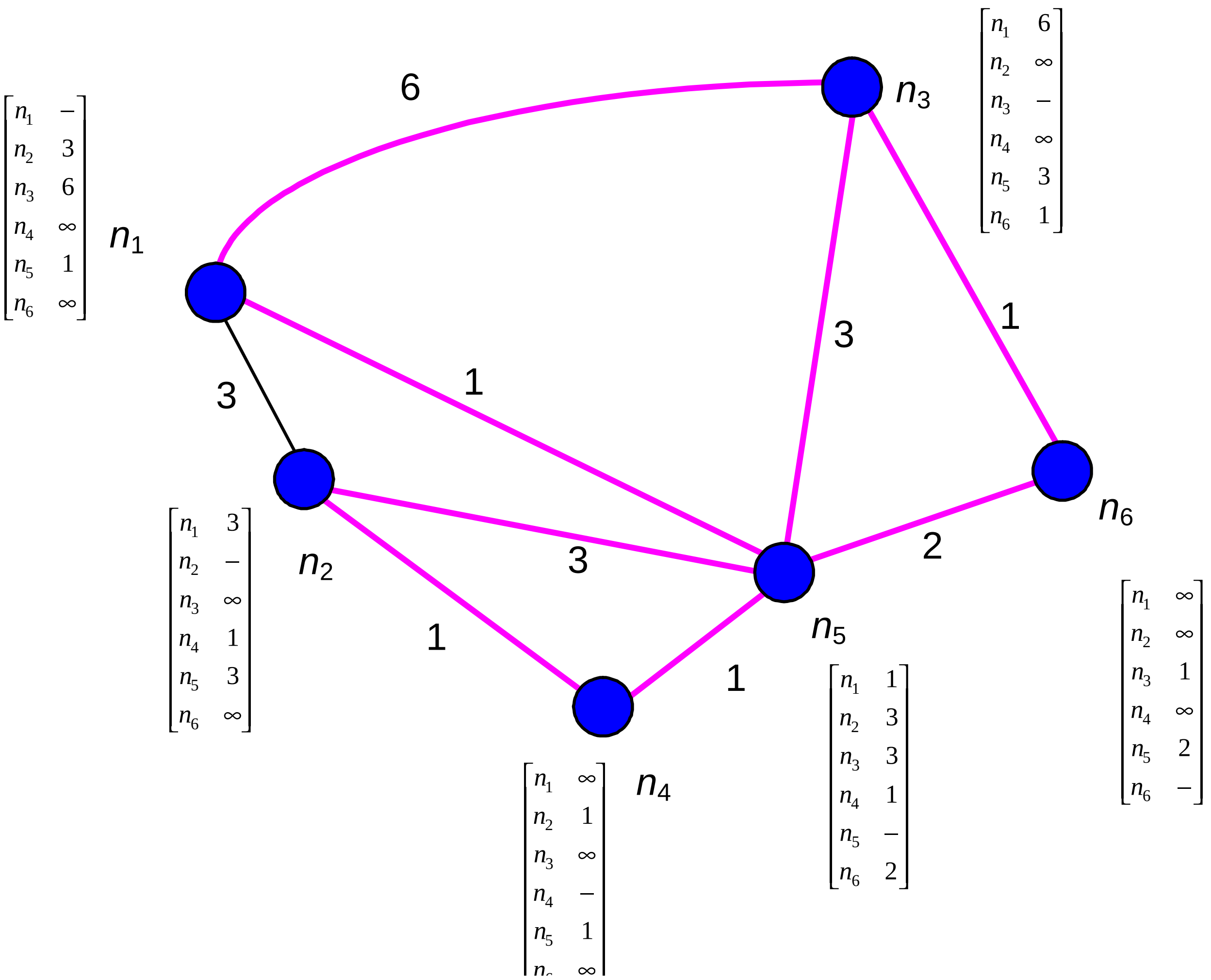
$\mathbf{D_x}$ is updated thus:

$$d_i(x) = \underset{y \in Neigh(x)}{Min} [d_i(y) + \ell(x,y)]$$

If $d_i(x)$ is a minimum for $y = y_k$, $\mathbf{f_x}$ is updated thus:

$$f_i(x) = y_k$$

Example

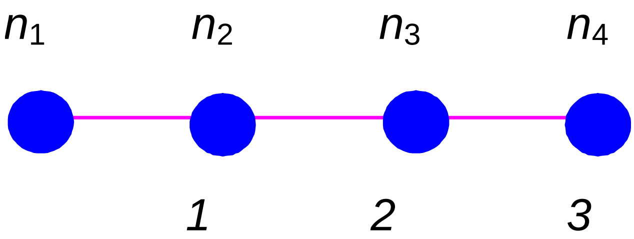


Distance vector calculation for n_1, first iteration.

Dest	Current best dist	Best dist offered by n_2	Best dist offered by n_3	Best dist offered by n_5	New vectors	
					Dist	Via
n_2	*3*	*0+3*	¥	*3+1*	*3*	n_2
n_3	*6*	¥	*0+6*	*3+1*	*4*	n_5
n_4	¥	*1+3*	¥	*1+1*	*2*	n_5
n_5	*1*	*3+3*	*3+6*	*0+1*	*1*	n_5
n_6	¥	¥	*1+6*	*2+1*	*3*	n_5

Convergence Problems

Count to infinity problem



Best distance to n_1

Assume each link has cost 1.

Suppose Link n_1—n_2 fails…

n_3 offers route to n_1 of cost 2 to node n_2. Oh dear…

It gets worse…

Distance to n_1 seen by:	n_2	n_3	n_4
After 1st exchange	3	2	3
After 2nd exchange	3	4	3
After 3rd exchange	5	4	5

Some solutions.

Split horizon: don't allow routers to advertise destinations in direction from which those destinations were learned.

Split horizon with poison reverse: advertise destinations in direction learned as distance of ∞.