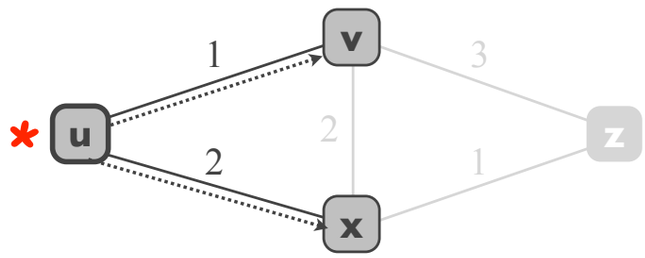
dest.	next hop	cost
z	-	-
v	v	1
x	x	2



In the first step, the algorithm considers only the the links that are directly connected to router u.

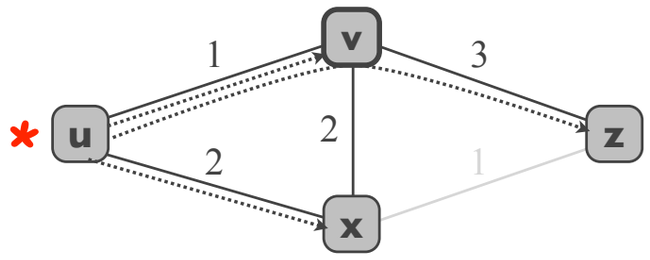
Considering only these 2 links:

- Router u cannot reach router z yet.

- Router u can reach router v through a direct link.
- The next hop is router v itself, and the cost is 1.

- Router u can reach router x through a direct link.
- The next hop is router x itself, and the cost is 2.

dest.	next hop	cost
z	~~-~~ v	~~-~~ 4
v	v	1
x	x	2



In the second step, the algorithm considers router v, and the links that are directly connected to router v.

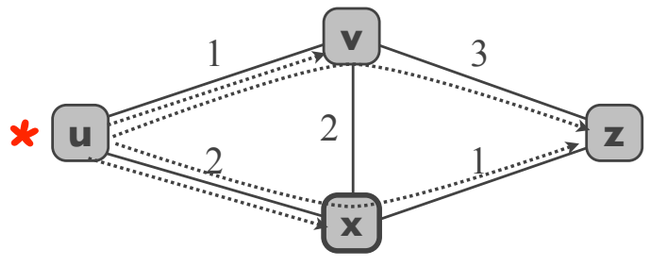
The algorithm asks: can router u improve its existing paths by routing through router v?

Let’s consider the path to destination router z:
- Can router u improve its path to destination z by routing through router v?
- Yes, it can: it previously could not reach router z at all, now it can reach router z through router v.
- So, the algorithm adds a new path through v.
- The next hop is router v, and the cost is 4.

Now let’s consider the path to destination router v:
- Can router u improve its path to destination v by routing through router v?
- No, because the path to destination router v is already through router v itself, so there is nothing to improve.

The same is true about the path to destination router x.

dest.	next hop	cost
z	~~v~~ ~~v~~ x	~~1~~ ~~4~~ 3
v	v	1
x	x	2



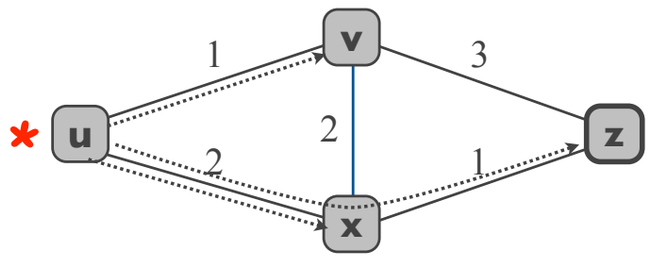
In the third step, the algorithm considers the other neighbor, router x,
and the links that are directly connected to router x.

The algorithm asks: can router u improve its existing paths by routing through router x?

Let’s consider the path to destination router z:
- Can router u improve its path to destination router z by routing through router x?
- Yes, it can, so the algorithm removes the old path through router v, and adds a new path through router x.
- The new next hop is router x, and the new cost is 3.

None of the other paths can be improved.

dest.	next hop	cost
z	~~v~~ ~~x~~ x	~~4~~ ~~4~~ 3
v	v	1
x	x	2



In the next step, the algorithm considers router z.

None of the paths can be improved, so, the algorithm is done:
it has computed the least-cost path from router u to every other router in the network.

Link-state routing algorithm for router u

- Input: router graph & link costs
- Output: least-cost path from router u to every other router

What we saw is an example of a link-state routing algorithm:

- It takes as input is the graph of all routers and the costs of all the links between them.
- It produces as output the least-cost path from a given source router to every other router in the network.

Link-state routing algorithm for router u

- “Centralized” algorithm:
  after the initial exchange of information,
  each router runs the algorithm
  without further exchange
- Still: all routers participate

A link-state routing algorithm can be characterised as a “centralized” algorithm, in the sense that, once an entity has the input (network graph and link costs), it runs the algorithm without any further communication.

One option is that that entity is the router itself, i.e., each router computes the least-cost path from itself to every other router in the network. This is what is typically done in most networks today.

Dijkstra's algorithm

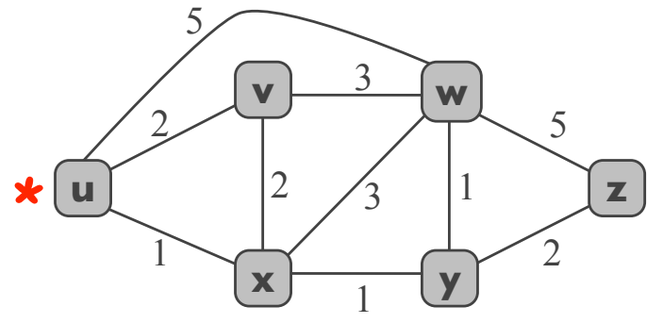
- At each step, consider a new router
  - *starting from "closest" neighbor*
- Check whether current paths can be improved
  - *by using that router as an intermediate point*
- End when no improvement is possible

The particular link-state routing algorithm that we discussed is called Dijstra's algorithm.

In summary:

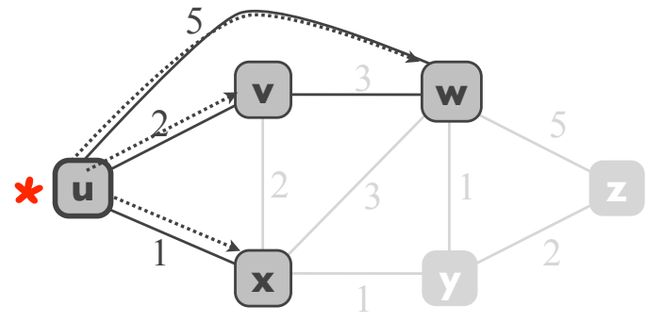
- At each step, the algorithm considers a new router.
- It checks whether it can improve the previously computed paths by routing through this new router.
- It ends when no improvement is possible.

dest.	next hop	cost
z		
w		
y		
v		
x		

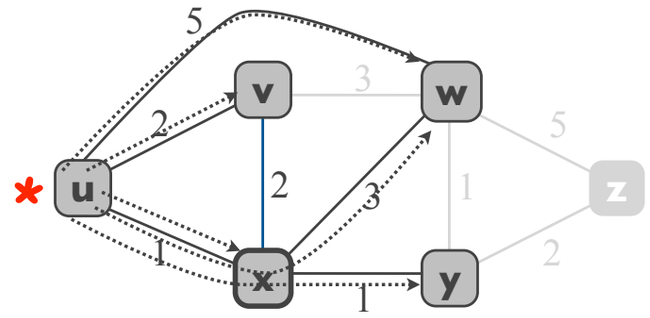


And one more example.
We always concentrate on the instance of the algorithm run by router u.

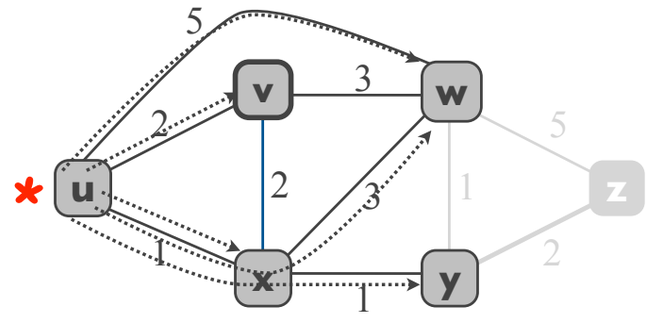
dest.	next hop	cost
z	-	-
w	w	5
y	-	-
v	v	2
x	x	1



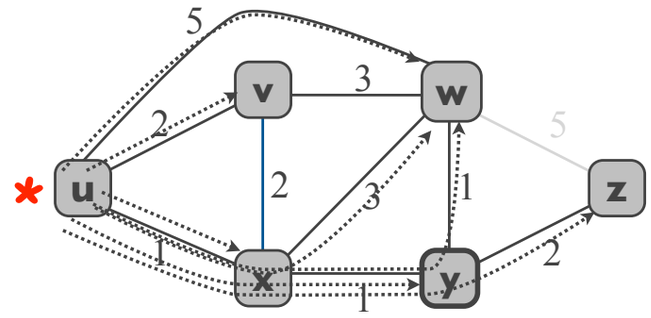
dest.	next hop	cost
z	-	-
w	~~w~~ x	~~5~~ 4
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1



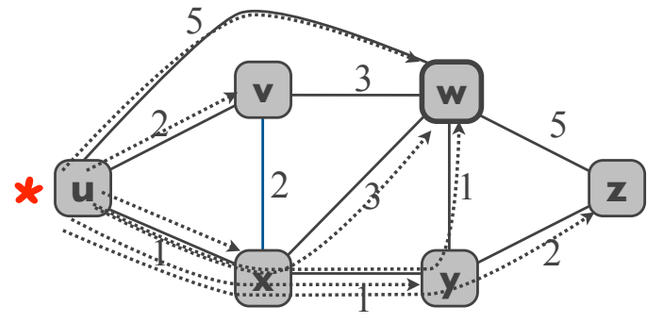
dest.	next hop	cost
z	-	-
w	~~w~~ x	~~5~~ 4
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1



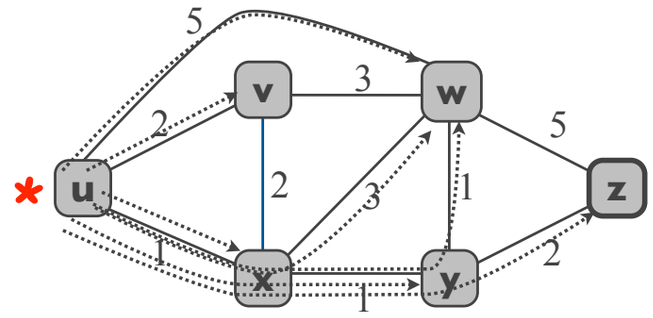
dest.	next hop	cost
z	~~-~~ x	~~-~~ 4
w	~~w~~ x	~~5~~ ~~4~~ 3
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1

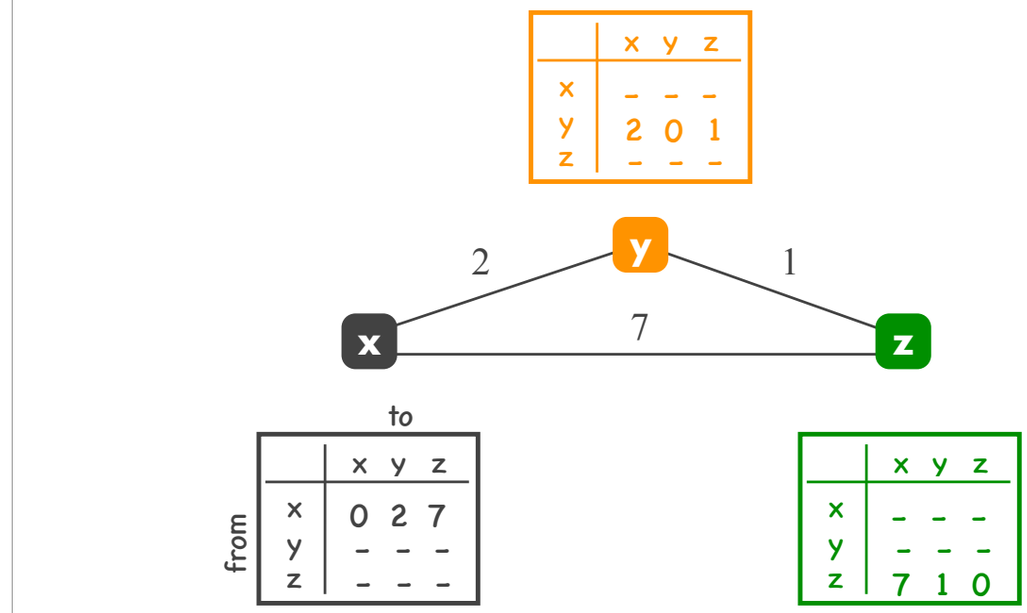


dest.	next hop	cost
z	~~-~~ x	~~-~~ 4
w	~~w~~ x	~~5~~ ~~4~~ 3
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1



dest.	next hop	cost
z	~~-~~ x	~~-~~ 4
w	~~w~~ x	~~5~~ ~~4~~ 3
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1





Now let’s look at another routing algorithm, which is quite different from Dijkstra.

This algorithm works in rounds and, in every round, each router exchanges information with its neighbours and recomputes its paths based on the exchanged information. So, it is a distributed algorithm, in the sense that, in each round, each router expects input from and provides input to other routers.

For the purposes of this algorithm, each router maintains a special table, which specifies the least cost between each pair of routers, and it is at first populated based on direct links.

For example, in the beginning, router x on the left knows that:
- it can reach itself with cost 0
- it can reach router y with cost 2 (through a direct link)
- and it can reach router z with cost 7 (through another direct link).

Similarly, in the beginning, router y in the middle knows that:
- it can reach router x with cost 2 (through a direct link)
- it can reach itself with cost 0
- and it can reach router z with cost 1 (through a direct link).