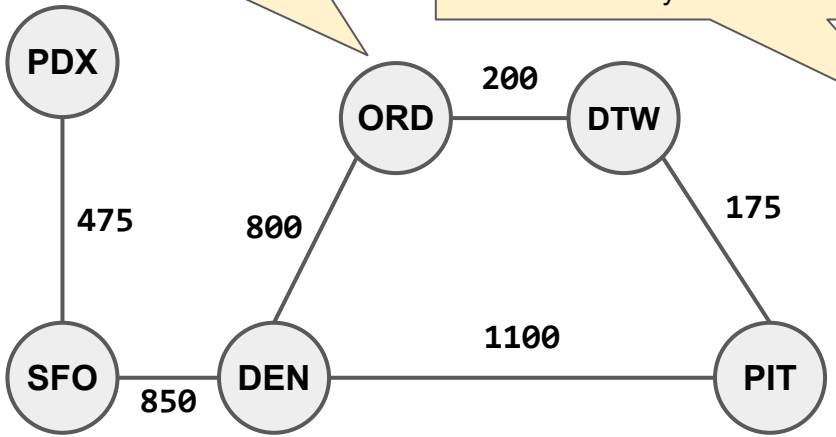
Dijkstra’s Algorithm Example

We want to find the shortest path from ORD to SFO

For each vertex in the graph, we need to keep track of the best distance (so far) from ORD to that vertex. Initially, all distances are set to “infinity”



Distance	
ORD	∞
DTW	∞
PIT	∞
DEN	∞
SFO	∞
PDX	∞

Prev	
ORD	
DTW	
PIT	
DEN	
SFO	
PDX	

We will also use a priority queue. We will add vertices, and their priority will be their distance to ORD. We can implement this priority queue with a min-heap

We also need to keep track of the “previous vertex” (on the shortest path to SFO)

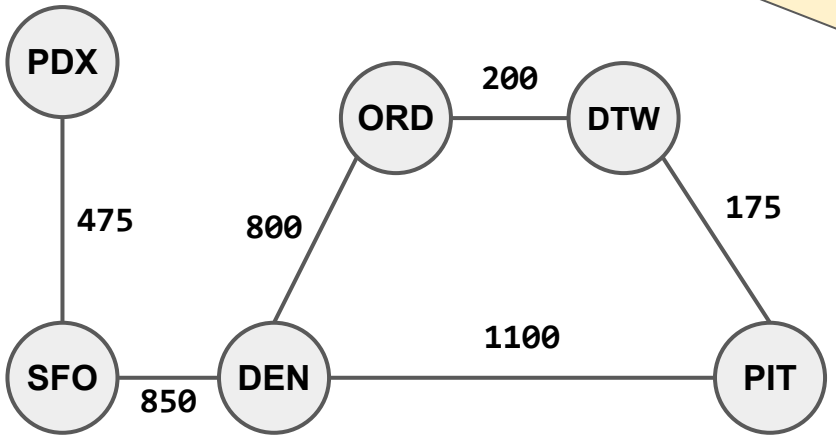
Back

Priority Queue

Front

Current Vertex

We initialize the algorithm by setting the distance to the origin vertex to 0.



Distance	
ORD	0
DTW	∞
PIT	∞
DEN	∞
SFO	∞
PDX	∞

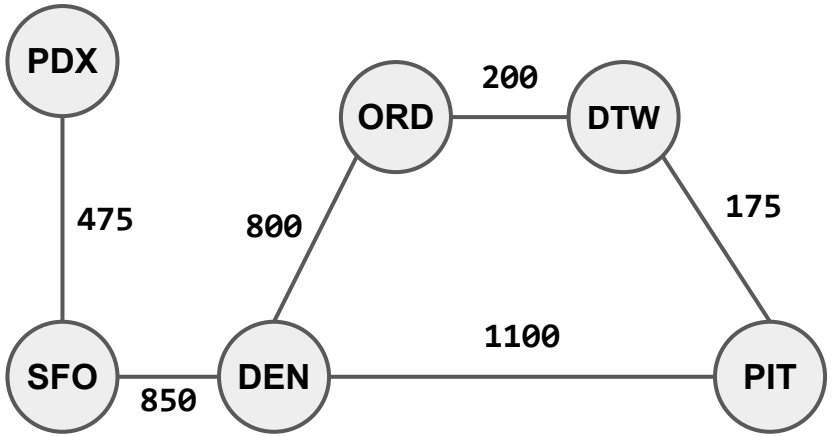
Prev	
ORD	
DTW	
PIT	
DEN	
SFO	
PDX	

And adding it to the priority queue, with priority zero (the distance from ORD to itself is zero)

Back **Priority Queue** Front

(0 ,ORD)

Current Vertex



Distance

ORD	0
DTW	∞
PIT	∞
DEN	∞
SFO	∞
PDX	∞

Prev

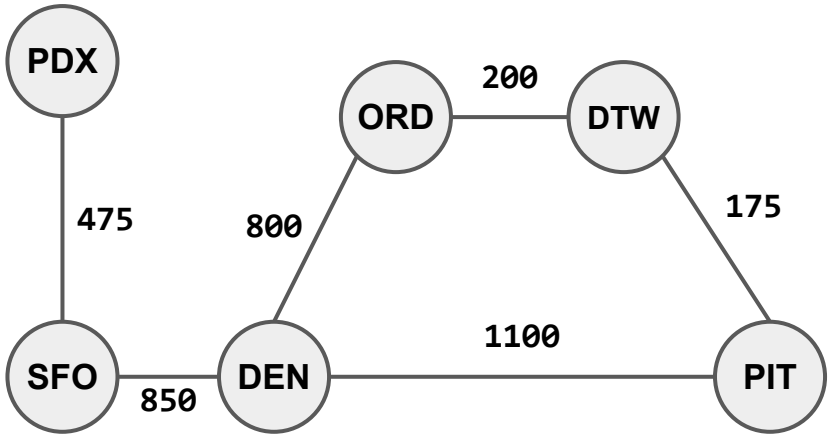
ORD	
DTW	
PIT	
DEN	
SFO	
PDX	

This is the initialization portion of the algorithm. Now we are ready to begin the main Dijkstra loop.

Back **Priority Queue** Front

(0 ,ORD)

Current Vertex



Distance	
ORD	0
DTW	∞
PIT	∞
DEN	∞
SFO	∞
PDX	∞

Prev	
ORD	
DTW	
PIT	
DEN	
SFO	
PDX	

We extract the lowest-priority value from the priority queue. It is not our target vertex (SFO), so we're not done yet.

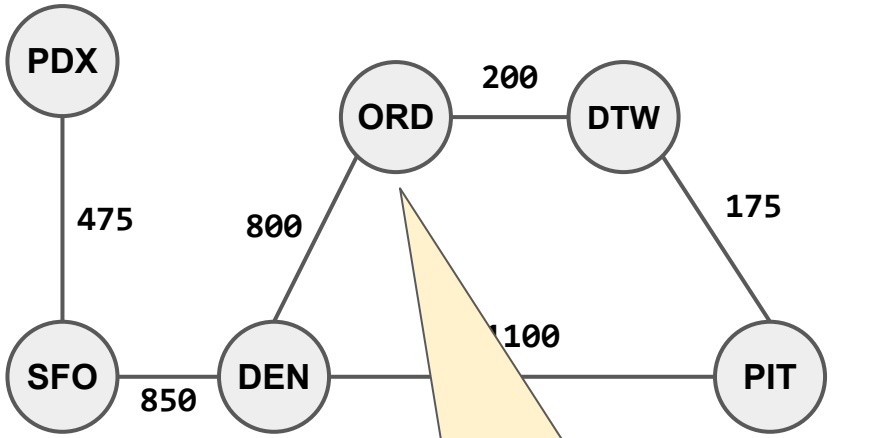
Back

Priority Queue

Front

Current Vertex

(0 ,ORD)



For every neighbor of the current vertex, we compute the distance from the origin vertex (ORD), and may update the priority queue based on that distance.

Distance

ORD	0
DTW	∞
PIT	∞
DEN	∞
SFO	∞
PDX	∞

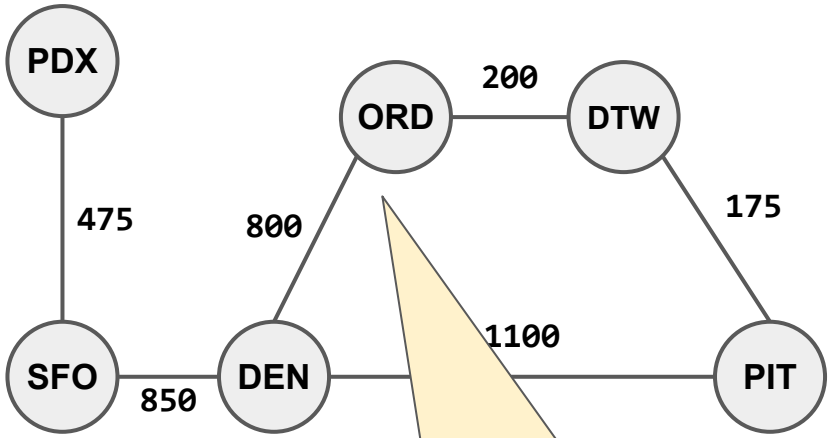
Prev

ORD	
DTW	
PIT	
DEN	
SFO	
PDX	

Back **Priority Queue** Front

Current Vertex

(0 ,ORD)



We happen to be processing ORD (the origin vertex) but, in general, we will be using `Distance[Current Vertex]` to obtain the distance to the origin vertex.

Distance	
ORD	0
DTW	∞
PIT	∞
DEN	∞
SFO	∞
PDX	∞

Prev	
ORD	
DTW	
PIT	
DEN	
SFO	
PDX	

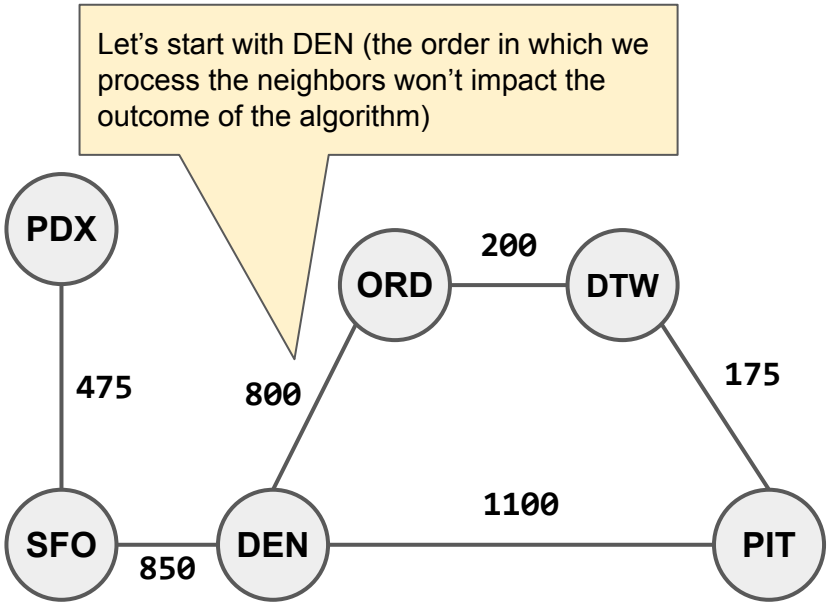
Back

Priority Queue

Front

Current Vertex

(0 ,ORD)



Distance	
ORD	0
DTW	∞
PIT	∞
DEN	∞
SFO	∞
PDX	∞

Prev	
ORD	
DTW	
PIT	
DEN	
SFO	
PDX	

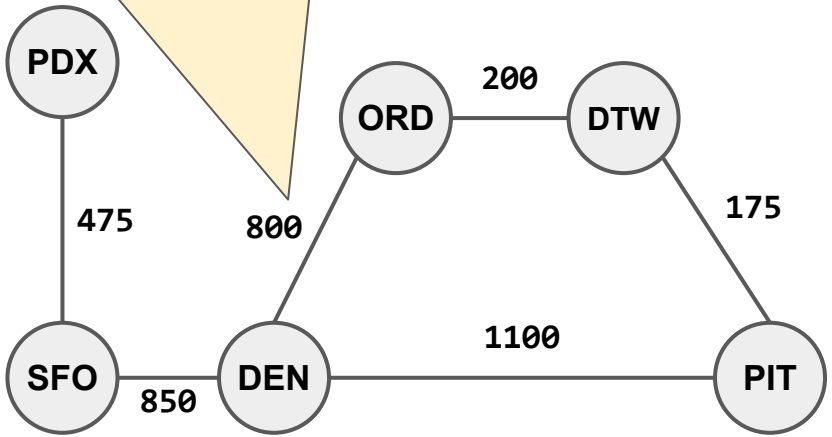
Back **Priority Queue** Front

Current Vertex

(0 ,ORD)

Its distance to the origin vertex is:

`Distance[ORD] + Weight(ORD, DEN) = 0 + 800 = 800`



Distance

ORD	0
DTW	∞
PIT	∞
DEN	∞
SFO	∞
PDX	∞

Prev

ORD	
DTW	
PIT	
DEN	
SFO	
PDX	

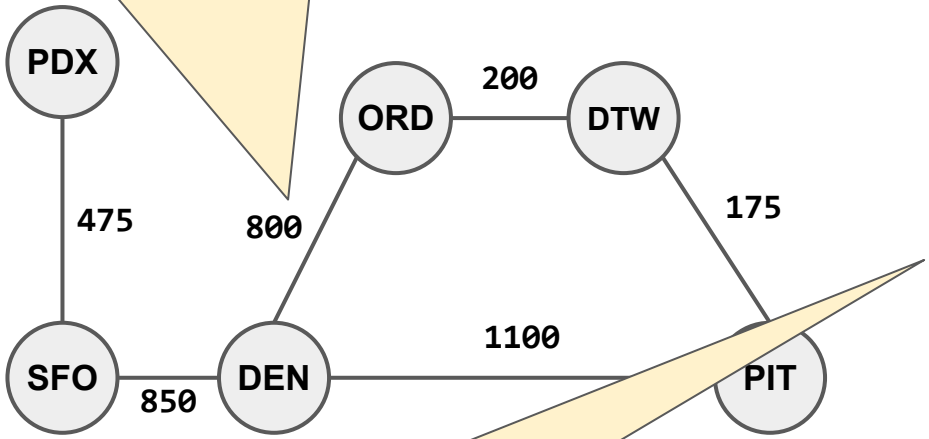
Back **Priority Queue** Front

Current Vertex

(0 ,ORD)

Its distance to the origin vertex is:

`Distance[ORD] + Weight(ORD, DEN) = 0 + 800 = 800`



This is better than the best distance we had so far (infinity), so we update the Distance table.

Distance

ORD	**0**
DTW	∞
PIT	∞
DEN	**800**
SFO	∞
PDX	∞

Prev

ORD	
DTW	
PIT	
DEN	
SFO	
PDX	

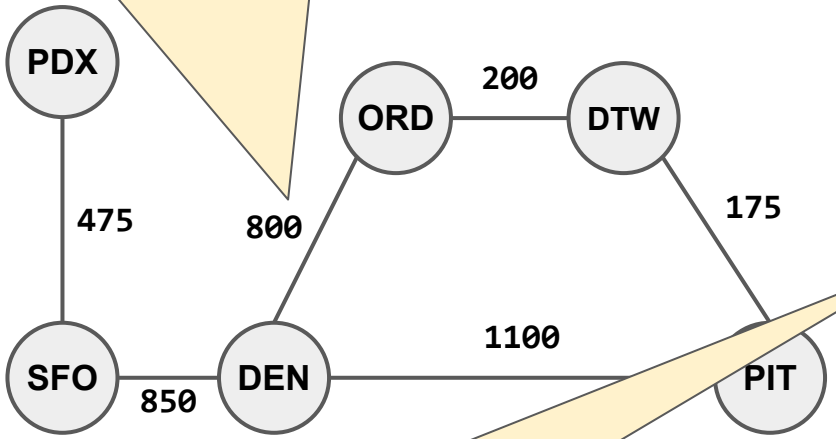
Back **Priority Queue** Front

Current Vertex

(0 ,ORD)

Its distance to the origin vertex is:

`Distance[ORD] + Weight(ORD, DEN) = 0 + 800 = 800`



Distance

ORD	0
DTW	∞
PIT	∞
DEN	800
SFO	∞
PDX	∞

Prev

ORD	
DTW	
PIT	
DEN	ORD
SFO	
PDX	

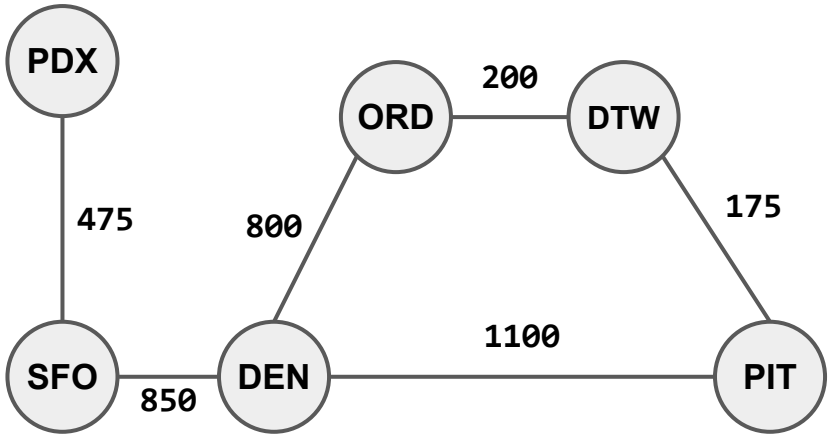
This is better than the best distance we had so far (infinity), so we update the Distance table.

We also set Prev[DEN] to ORD (the current vertex). This reflects that (as far as we know), the previous vertex to DEN (on the shortest path) will be ORD.

Back **Priority Queue** Front

Current Vertex

(0 ,ORD)



Distance	
ORD	0
DTW	∞
PIT	∞
DEN	800
SFO	∞
PDX	∞

Prev	
ORD	
DTW	
PIT	
DEN	ORD
SFO	
PDX	

We also update its priority in the queue. Since the vertex was not in the queue, it's not really a priority update but an insertion

Back

Priority Queue

Front

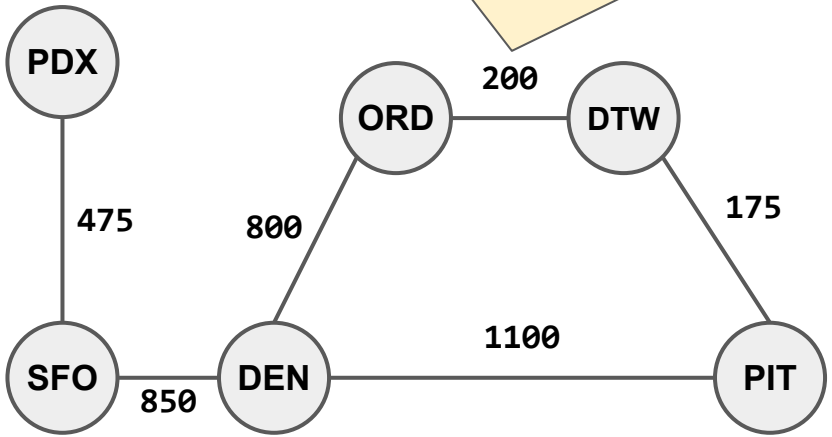
(800, DEN)

Current Vertex

(0 ,ORD)

Now we process DTW. Its distance to the origin vertex is:

`Distance[ORD] + Weight(ORD, DTW) = 0 + 200 = 200`



Distance

ORD	0
DTW	∞
PIT	∞
DEN	800
SFO	∞
PDX	∞

Prev

ORD	
DTW	
PIT	
DEN	ORD
SFO	
PDX	

Back **Priority Queue** Front

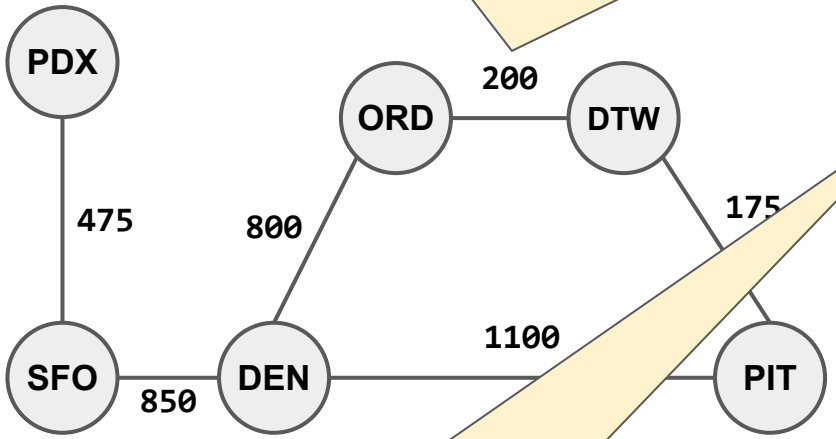
(800, DEN)

Current Vertex

(0 ,ORD)

Now we process DTW. Its distance to the origin vertex is:

`Distance[ORD] + Weight(ORD, DTW) = 0 + 200 = 200`



Distance

ORD	0
DTW	200
PIT	∞
DEN	800
SFO	∞
PDX	∞

Prev

ORD	
DTW	ORD
PIT	
DEN	ORD
SFO	
PDX	

This is better than the best distance we had so far (infinity), so we update the Distance and Prev tables.

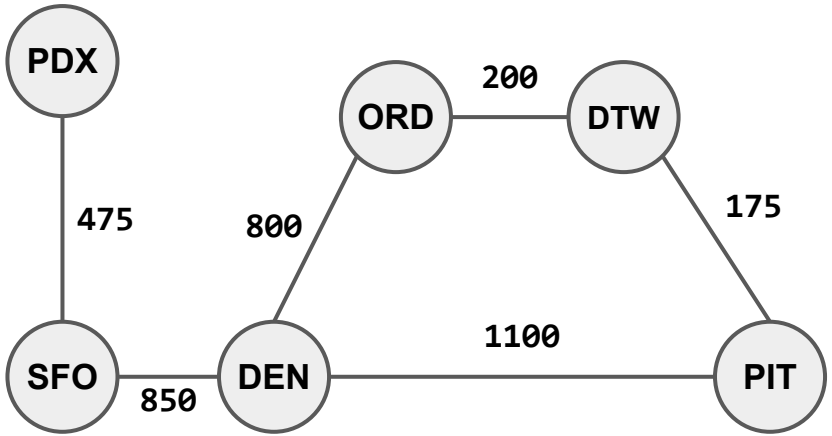
And we insert it into the priority queue. Notice that, being a priority queue, DTW goes to the front of the queue because it has a lower priority.

Back **Priority Queue** Front

(800, DEN) (200, DTW)

Current Vertex

(0 ,ORD)



Distance	
ORD	0
DTW	200
PIT	∞
DEN	800
SFO	∞
PDX	∞

Prev	
ORD	
DTW	ORD
PIT	
DEN	ORD
SFO	
PDX	

We've processed all of ORD's neighbors, so we're done processing ORD

Back

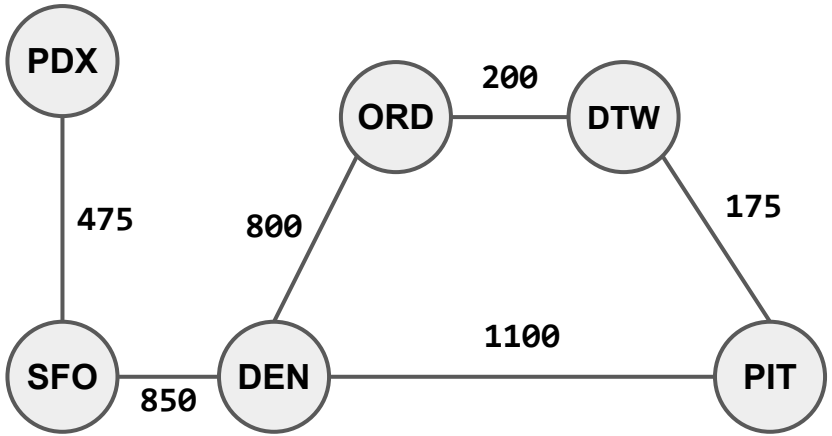
Priority Queue

Front

(800, DEN) (200, DTW)

Current Vertex

(0 ,ORD)



Distance	
ORD	0
DTW	200
PIT	∞
DEN	800
SFO	∞
PDX	∞

Prev	
ORD	
DTW	ORD
PIT	
DEN	ORD
SFO	
PDX	

We extract the lowest-priority value from the priority queue. It is not our target vertex (SFO), so we're not done yet.

Back

Priority Queue

Front

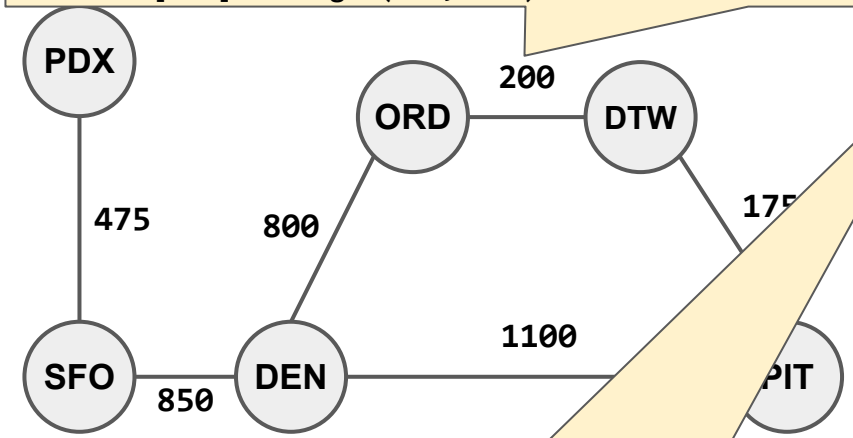
Current Vertex

(800, DEN)

(200, DTW)

We check its neighbors, starting with ORD. The distance to the origin vertex is:

`Distance[DTW] + Weight(DTW, ORD) = 200 + 200 = 400`



400 represents the distance I'd have to travel to get to ORD, starting at ORD, but travelling to DTW and back. This is clearly worse than the distance I already had (which represents staying put at ORD), so we don't update the Distance or Prev tables.

Distance

ORD	0
DTW	200
PIT	∞
DEN	800
SFO	∞
PDX	∞

Prev

ORD	
DTW	ORD
PIT	
DEN	ORD
SFO	
PDX	

Back **Priority Queue** Front

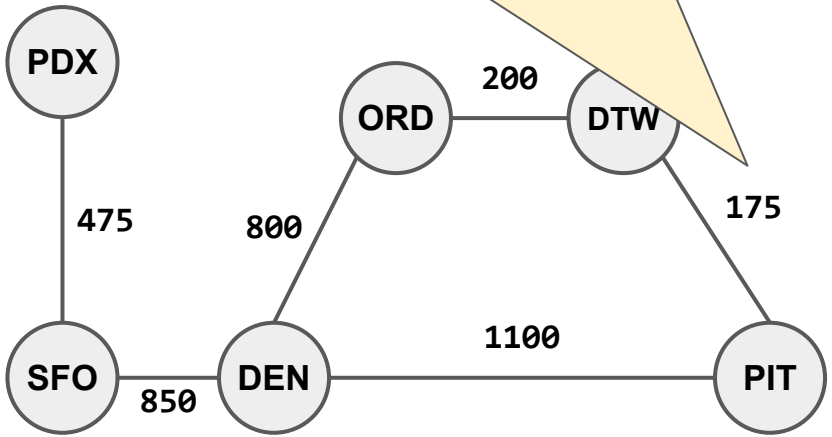
(800, DEN)

Current Vertex

(200, DTW)

Now we process PIT. It's distance to the origin vertex is:

`Distance[DTW] + Weight(DTW, PIT) = 200 + 175 = 375`



Distance

ORD	0
DTW	200
PIT	∞
DEN	800
SFO	∞
PDX	∞

Prev

ORD	
DTW	ORD
PIT	
DEN	ORD
SFO	
PDX	

Back **Priority Queue** Front

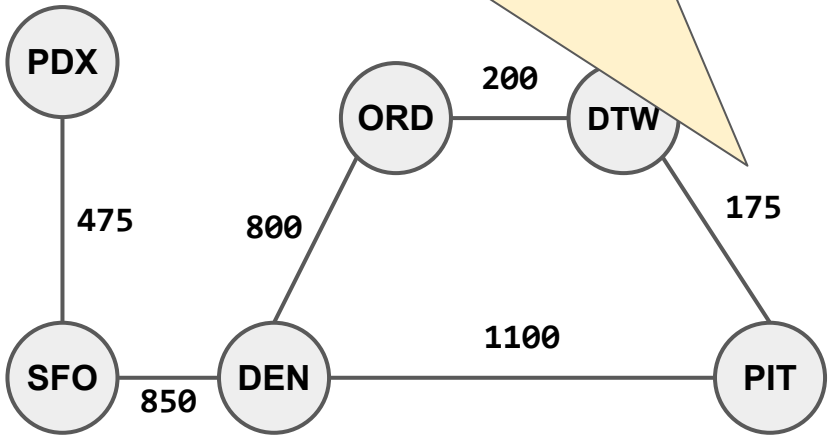
(800, DEN)

Current Vertex

(200, DTW)

Now we process PIT. It's distance to the origin vertex is:

`Distance[DTW] + Weight(DTW, PIT) = 200 + 175 = 375`



Distance

ORD	0
DTW	200
PIT	375
DEN	800
SFO	∞
PDX	∞

Prev

ORD	
DTW	ORD
PIT	DTW
DEN	ORD
SFO	
PDX	

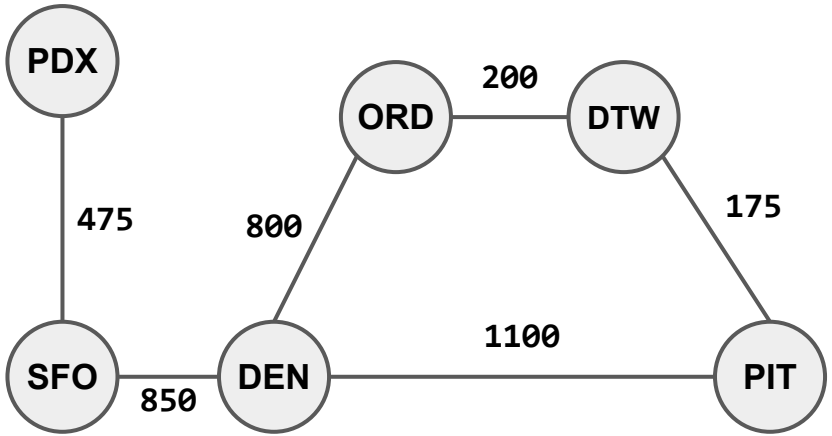
This is better than the distance we had so far, so we update the Distance and Prev tables, and add PIT to the priority queue (with this best distance so far). Again, notice how the queue is ordered by priority.

Back **Priority Queue** Front

(800, DEN) (375, PIT)

Current Vertex

(200, DTW)



Distance	
ORD	0
DTW	200
PIT	375
DEN	800
SFO	∞
PDX	∞

Prev	
ORD	
DTW	ORD
PIT	DTW
DEN	ORD
SFO	
PDX	

We're done processing DTW, and now PIT is our current vertex.

Back

Priority Queue

Front

Current Vertex

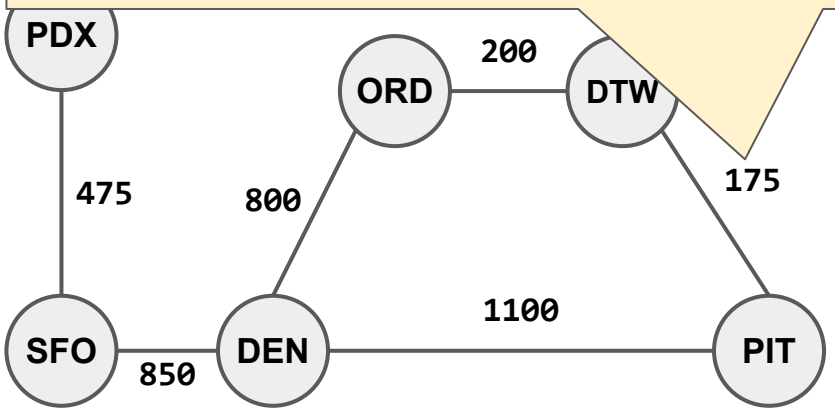
(800, DEN)

(375, PIT)

We check its neighbors, starting with DTW. The distance to the origin vertex is:

`Distance[PIT] + Weight(PIT, DTW) = 375 + 175 = 550`

This is worse than the best distance to DTW so far, so we do nothing



Distance

ORD	0
DTW	200
PIT	375
DEN	800
SFO	∞
PDX	∞

Prev

ORD	
DTW	ORD
PIT	DTW
DEN	ORD
SFO	
PDX	

Back **Priority Queue** Front

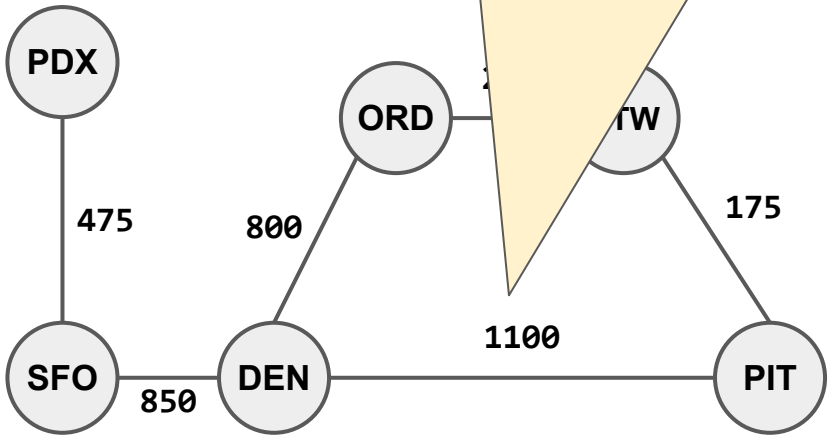
(800, DEN)

Current Vertex

(375, PIT)

Now we process DEN. It's distance to the origin vertex is:

`Distance[PIT] + Weight(PIT, DEN) = 375 + 1100 = 1475`



Distance

ORD	0
DTW	200
PIT	375
DEN	800
SFO	∞
PDX	∞

Prev

ORD	
DTW	ORD
PIT	DTW
DEN	ORD
SFO	
PDX	

This isn't better than the distance we have so far, so we also do nothing.

Back

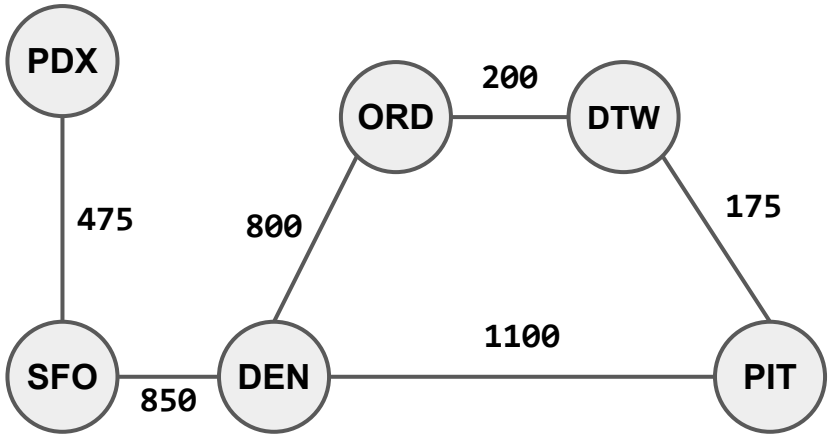
Priority Queue

Front

(800, DEN)

Current Vertex

(375, PIT)



Distance	
ORD	0
DTW	200
PIT	375
DEN	800
SFO	∞
PDX	∞

Prev	
ORD	
DTW	ORD
PIT	DTW
DEN	ORD
SFO	
PDX	

We're done processing PIT, and now DEN is our current vertex.

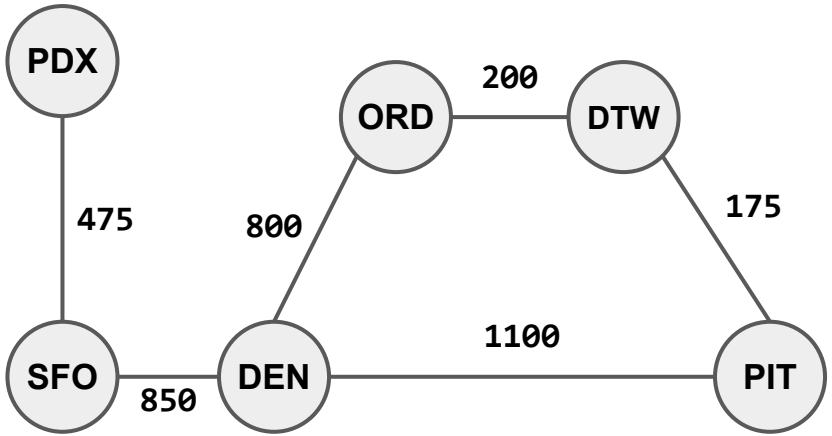
Back

Priority Queue

Front

Current Vertex

(800, DEN)



Distance	
ORD	0
DTW	200
PIT	375
DEN	800
SFO	∞
PDX	∞

Prev	
ORD	
DTW	ORD
PIT	DTW
DEN	ORD
SFO	
PDX	

We process its neighbors. Travelling to ORD and PIT won't give us better distances from ORD, so we skip those.

Back

Priority Queue

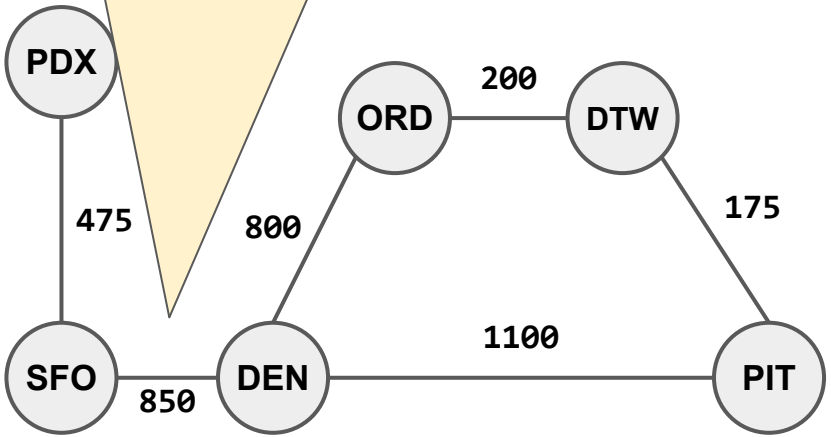
Front

Current Vertex

(800, DEN)

Now we process SFO. Its distance to the origin vertex is:

`Distance[DEN] + Weight(DEN, SFO) = 800 + 850 = 1650`



Distance

ORD	0
DTW	200
PIT	375
DEN	800
SFO	∞
PDX	∞

Prev

ORD	
DTW	ORD
PIT	DTW
DEN	ORD
SFO	
PDX	

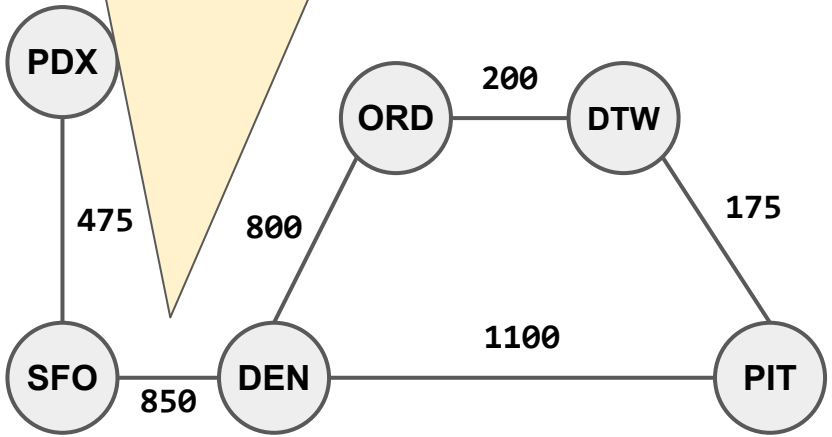
Back **Priority Queue** Front

Current Vertex

(800, DEN)

Now we process SFO. It's distance to the origin vertex is:

`Distance[DEN] + Weight(DEN, SFO) = 800 + 850 = 1650`



Distance

ORD	0
DTW	200
PIT	375
DEN	800
SFO	1650
PDX	∞

Prev

ORD	
DTW	ORD
PIT	DTW
DEN	ORD
SFO	DEN
PDX	

This is better than the distance we had so far, so we update the Distance and Prev tables, and add SFO to the priority queue

Back

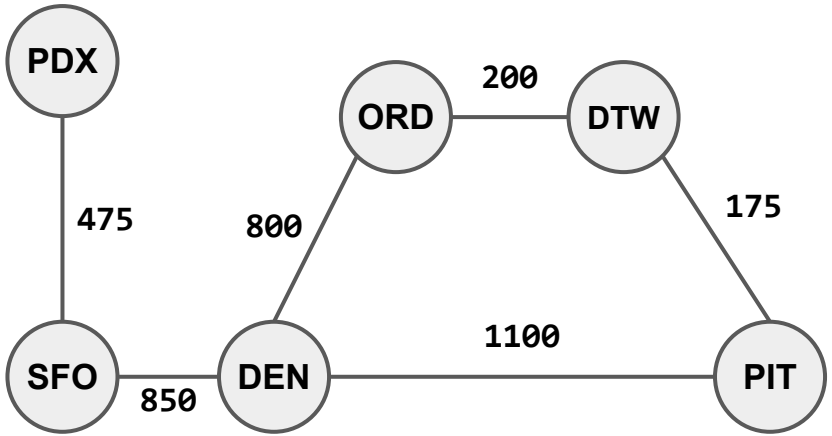
Priority Queue

Front

(1650, SFO)

Current Vertex

(800, DEN)



Distance	
ORD	0
DTW	200
PIT	375
DEN	800
SFO	1650
PDX	∞

Prev	
ORD	
DTW	ORD
PIT	DTW
DEN	ORD
SFO	DEN
PDX	

We're done processing DEN, and now SFO is our current vertex.

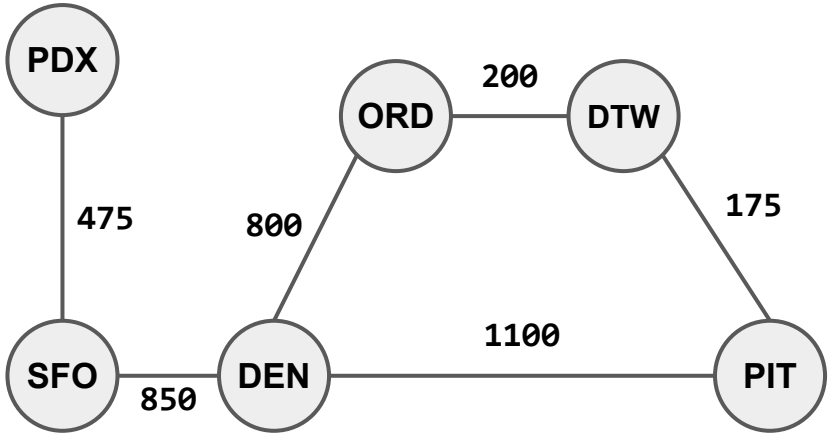
Back

Priority Queue

Front

Current Vertex

(1650, SFO)



Distance	
ORD	0
DTW	200
PIT	375
DEN	800
SFO	1650
PDX	∞

Prev	
ORD	
DTW	ORD
PIT	DTW
DEN	ORD
SFO	DEN
PDX	

Once the target vertex is the current vertex, we are done.

This is the shortest distance from ORD to SFO

The shortest path can be extracted by following the “prev” vertices starting at SFO.

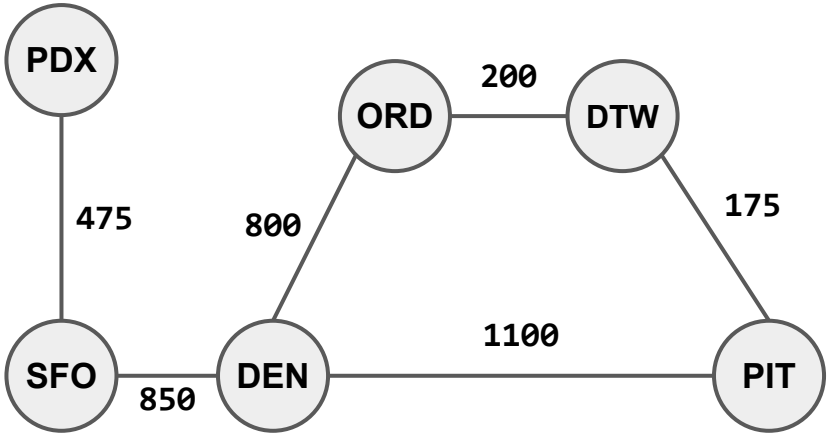
Back

Priority Queue

Front

Current Vertex

(1650, SFO)



Distance	
ORD	0
DTW	200
PIT	375
DEN	800
SFO	1650
PDX	∞

Prev	
ORD	
DTW	ORD
PIT	DTW
DEN	ORD
SFO	DEN
PDX	

Note: Dijkstra also gives us the shortest distance from ORD to all other nodes. We could keep going to find the shortest distance to PDX.

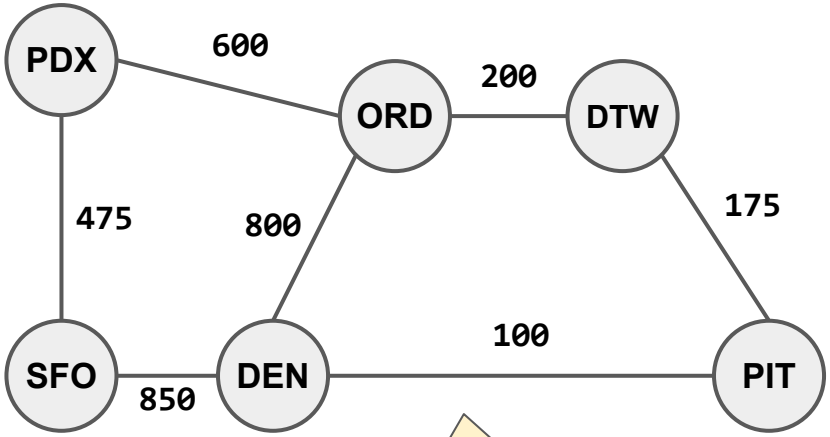
Back **Priority Queue** Front

Current Vertex

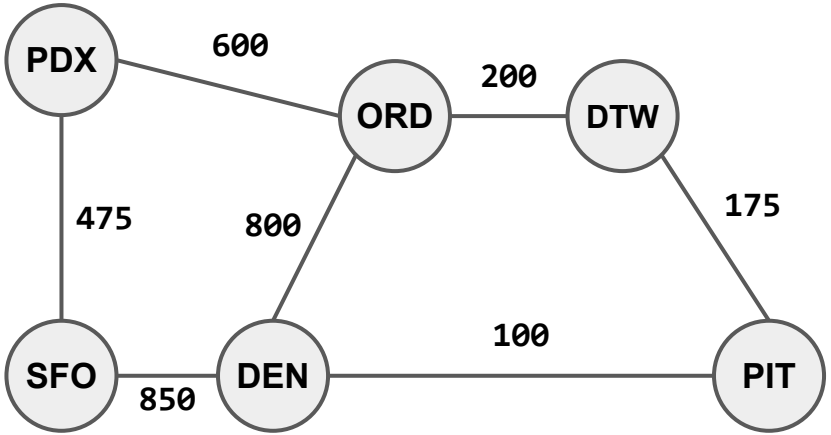
(1650, SFO)

Priority Update Example

In the previous example, we always inserted new vertices into the priority queue, and didn’t see an example of a priority update. Let’s tweak the graph to see what this would look like.



Suppose our graph looked like this. The distance from PIT to DEN is now 100, and there is a flight from ORD to PDX with a distance of 600. These are not the actual distances, but let's assume that wormholes have appeared along those routes that make the flights considerably shorter.



Distance	
ORD	0
DTW	200
PIT	375
DEN	800
SFO	∞
PDX	600

Prev	
ORD	
DTW	ORD
PIT	DTW
DEN	ORD
SFO	
PDX	ORD

Let's fast-forward to the point where PIT is the current vertex.

Like before, the algorithm has been exploring the path through DTW, and the priority queue reflects that there are two other (seemingly less promising) paths to explore through PDX and DEN

Back **Priority Queue** Front

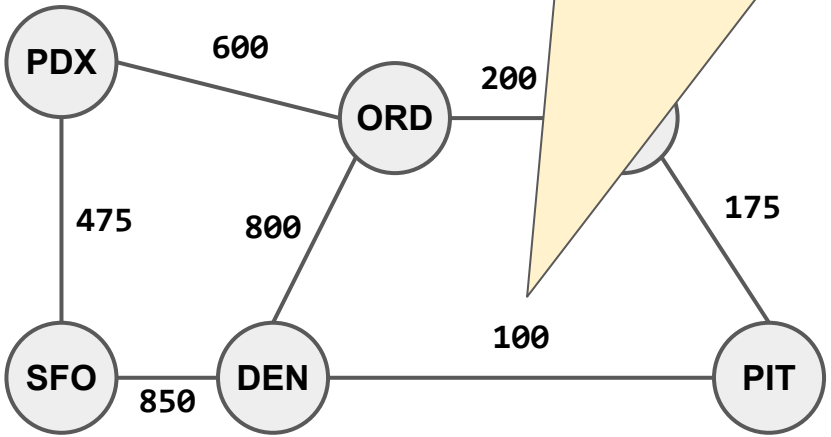
(800, DEN) (600, PDX)

Current Vertex

(375, PIT)

When processing DEN, we find its distance is:

`Distance[PIT] + Weight(PIT, DEN) = 375 + 100 = 475`



Distance

ORD	0
DTW	200
PIT	375
DEN	800
SFO	∞
PDX	600

Prev

ORD	
DTW	ORD
PIT	DTW
DEN	ORD
SFO	
PDX	ORD

Back **Priority Queue** Front

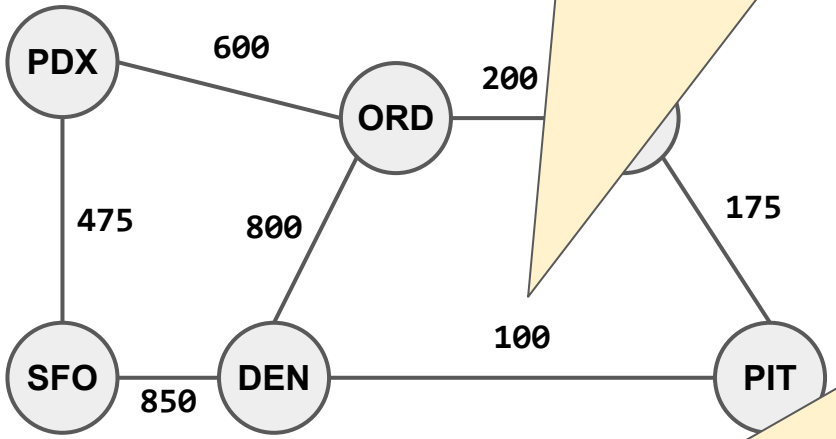
(800, DEN) (600, PDX)

Current Vertex

(375, PIT)

When processing DEN, we find its distance is:

`Distance[PIT] + Weight(PIT, DEN) = 375 + 100 = 475`



Distance

ORD	0
DTW	200
PIT	375
DEN	475
SFO	∞
PDX	600

Prev

ORD	
DTW	ORD
PIT	DTW
DEN	PIT
SFO	
PDX	ORD

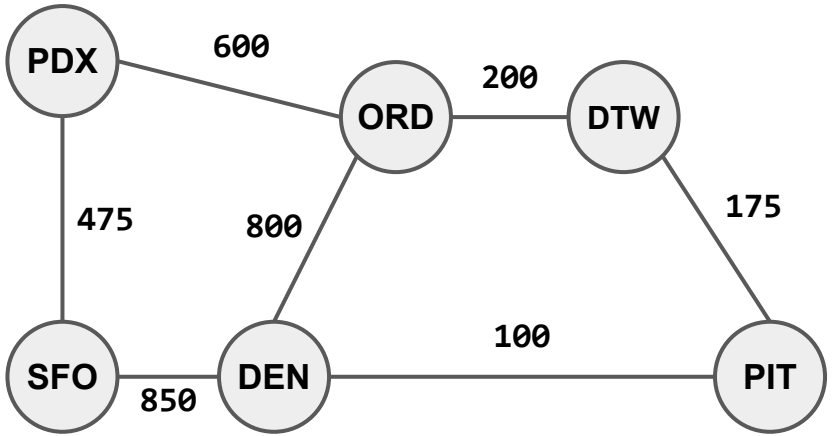
This is **better** than the distance we had so far (800), so we update the Distance and Prev tables to reflect that there is a shorter path to DEN.

Back **Priority Queue** Front

(800, DEN) (600, PDX)

Current Vertex

(375, PIT)



Distance	
ORD	0
DTW	200
PIT	375
DEN	475
SFO	∞
PDX	600

Prev	
ORD	
DTW	ORD
PIT	DTW
DEN	PIT
SFO	
PDX	ORD

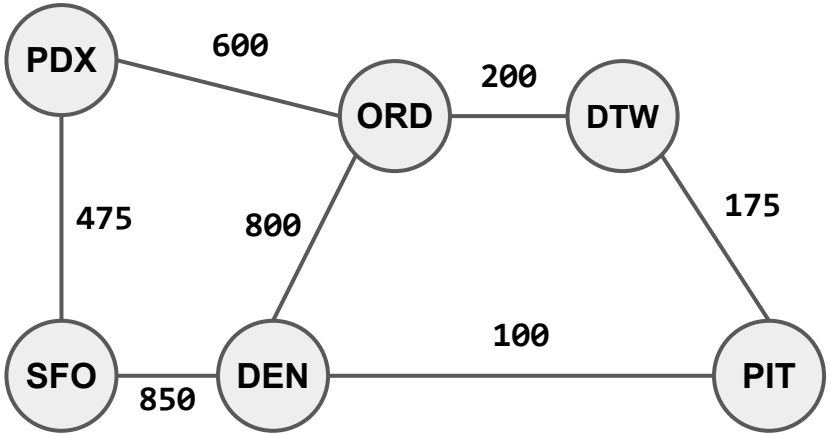
We also update DEN's priority in the queue with this new distance, which means it will now skip ahead of PDX in the queue.

Back **Priority Queue** Front

(600, PDX) (475, DEN)

Current Vertex

(375, PIT)



Distance	
ORD	0
DTW	200
PIT	375
DEN	475
SFO	∞
PDX	600

Prev	
ORD	
DTW	ORD
PIT	DTW
DEN	PIT
SFO	
PDX	ORD

We're done processing PIT, and now DEN is our current vertex.

Priority Queue

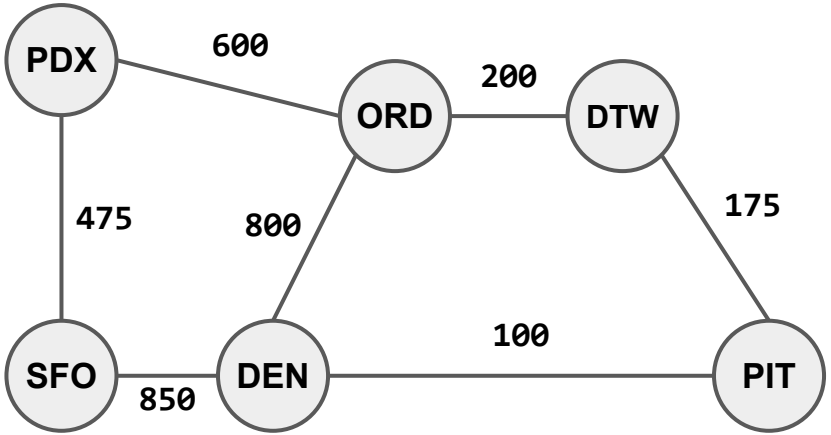
Back

(600, PDX)

Front

Current Vertex

(475, DEN)



Distance	
ORD	0
DTW	200
PIT	375
DEN	475
SFO	∞
PDX	600

Prev	
ORD	
DTW	ORD
PIT	DTW
DEN	PIT
SFO	
PDX	ORD

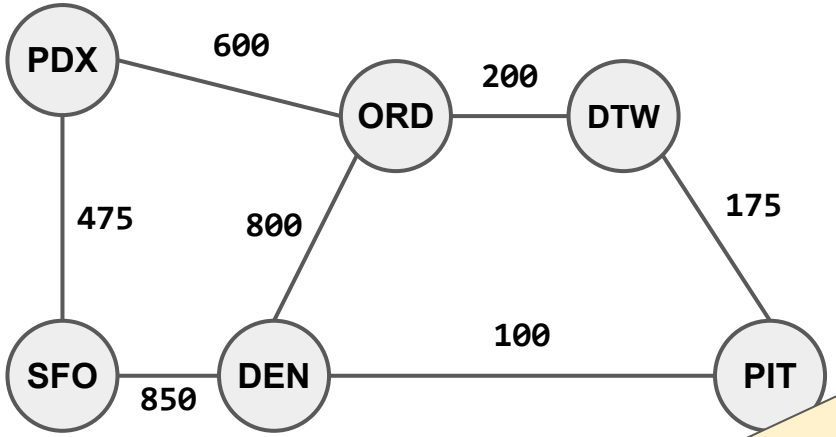
Like before, going from DEN to PIT or from DEN to ORD isn't going to improve our distances from ORD, so we skip those.

Back **Priority Queue** Front

(600, PDX)

Current Vertex

(475, DEN)



Distance

ORD	0
DTW	200
PIT	375
DEN	475
SFO	1325
PDX	600

Prev

ORD	
DTW	ORD
PIT	DTW
DEN	PIT
SFO	DEN
PDX	ORD

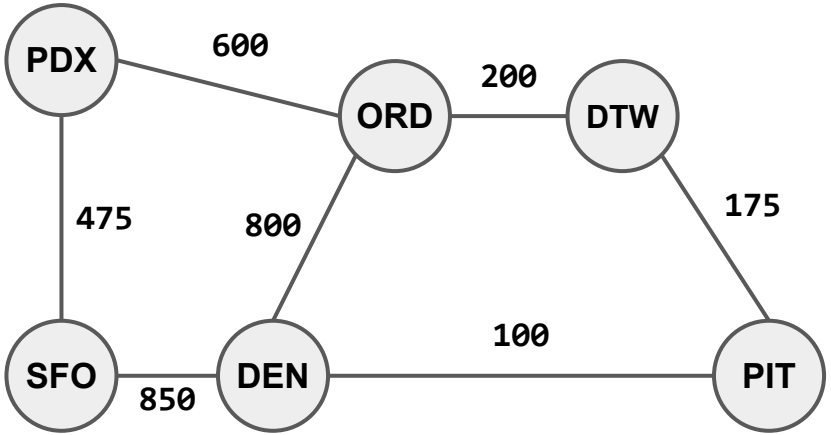
The distance to SFO (1325) is better than the distance we had so far (infinity), so we update the Distance and Prev tables.

Back **Priority Queue** Front

(600, PDX)

Current Vertex

(475, DEN)



Distance	
ORD	0
DTW	200
PIT	375
DEN	475
SFO	1325
PDX	600

Prev	
ORD	
DTW	ORD
PIT	DTW
DEN	PIT
SFO	DEN
PDX	ORD

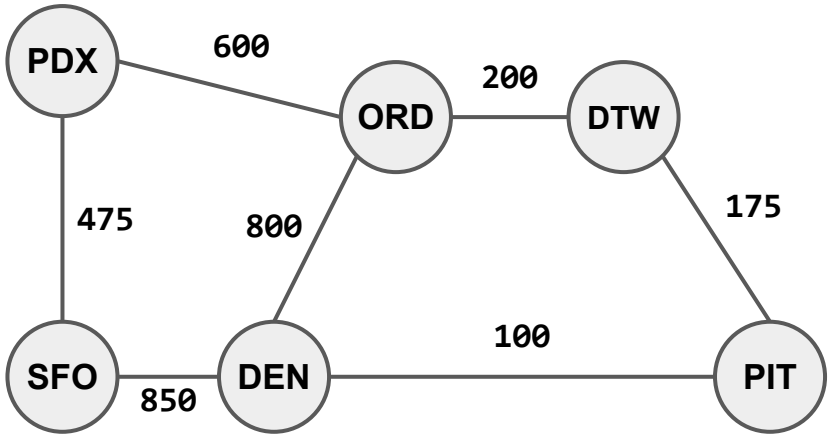
Unlike our previous example, we won't be done after processing DEN, because we still have to explore the path through PDX. Remember: we're done when the target vertex becomes the current vertex, *not* when we encounter the target vertex for the first time.

Back **Priority Queue** Front

(1325, SFO) (600, PDX)

Current Vertex

(475, DEN)



Distance	
ORD	0
DTW	200
PIT	375
DEN	475
SFO	1325
PDX	600

Prev	
ORD	
DTW	ORD
PIT	DTW
DEN	PIT
SFO	DEN
PDX	ORD

In fact, if we kept running the algorithm, we would discover that there is a shorter path through PDX.

Back **Priority Queue** Front

(1325, SFO) (600, PDX)

Current Vertex

(475, DEN)