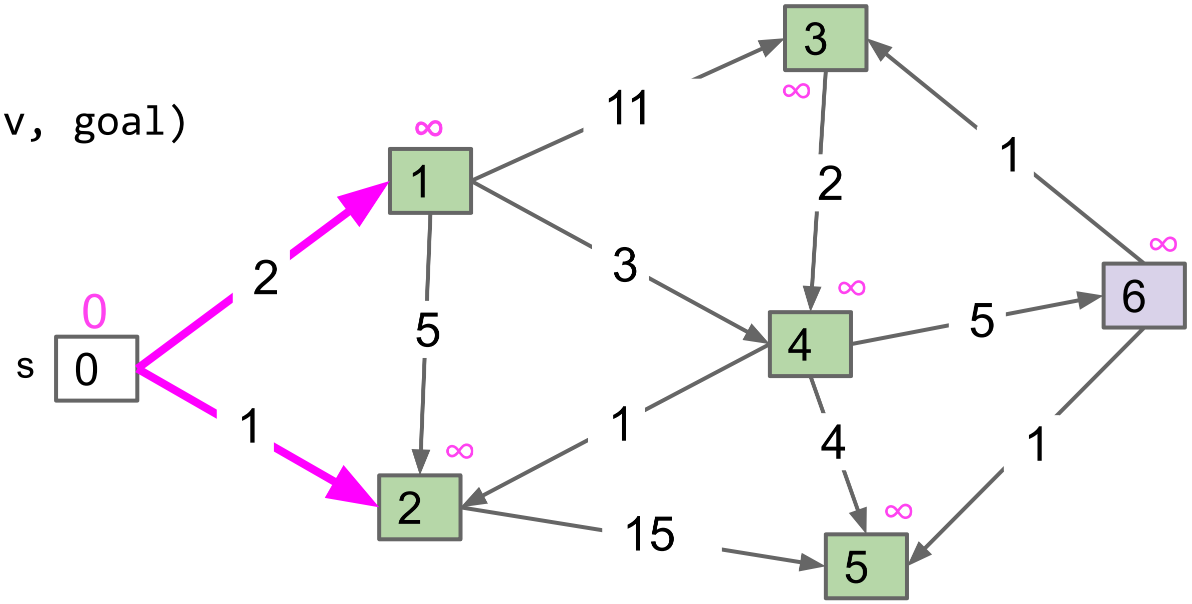
A* Demo, with s = 0, goal = 6.

Insert all vertices into fringe PQ, storing vertices in order of d(source, v) + h(v, goal).

Repeat: Remove best vertex v from PQ, and relax all edges pointing from v.

#	distTo	edgeTo	h(v, goal)
0	0	-	1
1	∞	-	3
2	∞	-	15
3	∞	-	1
4	∞	-	1
5	∞	-	∞
6	∞	-	0



h(v, goal) is arbitrary. In this example, it's the min weight edge out of each vertex.

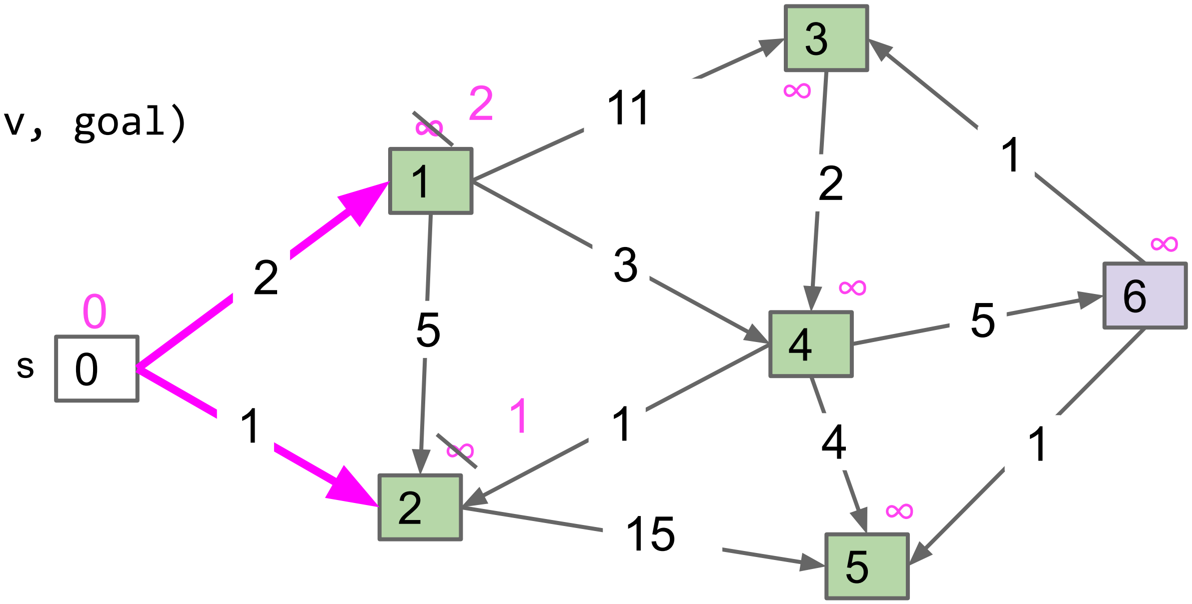
```
Fringe: [(1: ∞), (2: ∞), (3: ∞), (4: ∞), (5: ∞), (6: ∞)]
```

A* Demo, with s = 0, goal = 6.

Insert all vertices into fringe PQ, storing vertices in order of d(source, v) + h(v, goal).

Repeat: Remove best vertex v from PQ, and relax all edges pointing from v.

#	distTo	edgeTo	h(v, goal)
0	0	-	1
1	2	0	3
2	1	0	15
3	∞	-	1
4	∞	-	1
5	∞	-	∞
6	∞	-	0



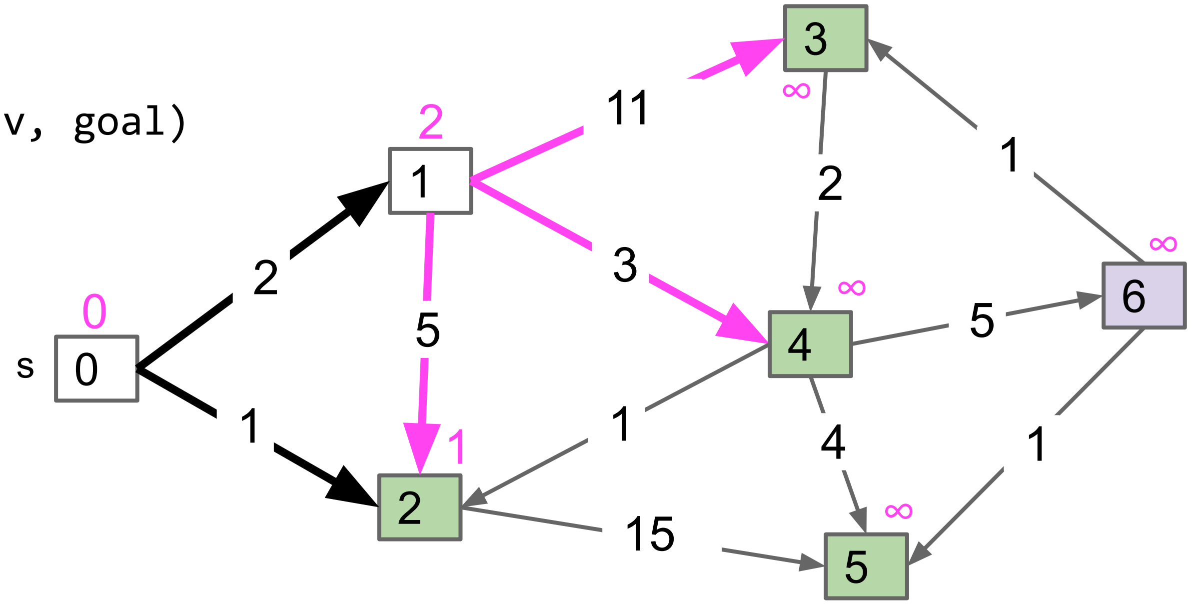
Fringe: [(1: 5), (2: 16), (3: ∞), (4: ∞), (5: ∞), (6: ∞)]

A* Demo, with s = 0, goal = 6.

Insert all vertices into fringe PQ, storing vertices in order of d(source, v) + h(v, goal).

Repeat: Remove best vertex v from PQ, and relax all edges pointing from v.

#	distTo	edgeTo	h(v, goal)
0	0	-	1
1	2	0	3
2	1	0	15
3	∞	-	2
4	∞	-	1
5	∞	-	∞
6	∞	-	0



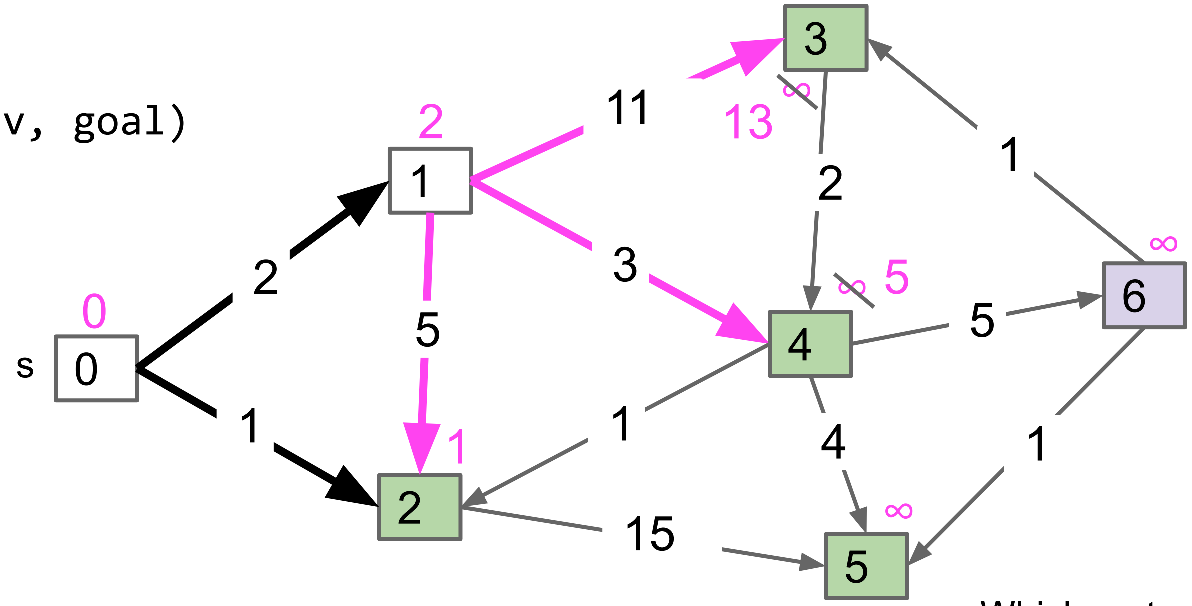
Fringe: [(2: 16), (3: ∞), (4: ∞), (5: ∞), (6: ∞)]

A* Demo, with s = 0, goal = 6.

Insert all vertices into fringe PQ, storing vertices in order of d(source, v) + h(v, goal).

Repeat: Remove best vertex v from PQ, and relax all edges pointing from v.

#	distTo	edgeTo	h(v, goal)
0	0	-	1
1	2	0	3
2	1	0	15
3	13	1	2
4	5	1	1
5	∞	-	∞
6	∞	-	0



Fringe: [(4: 6), (3: 15), (2: 16), (5: ∞), (6: ∞)]

Which vertex is removed next?

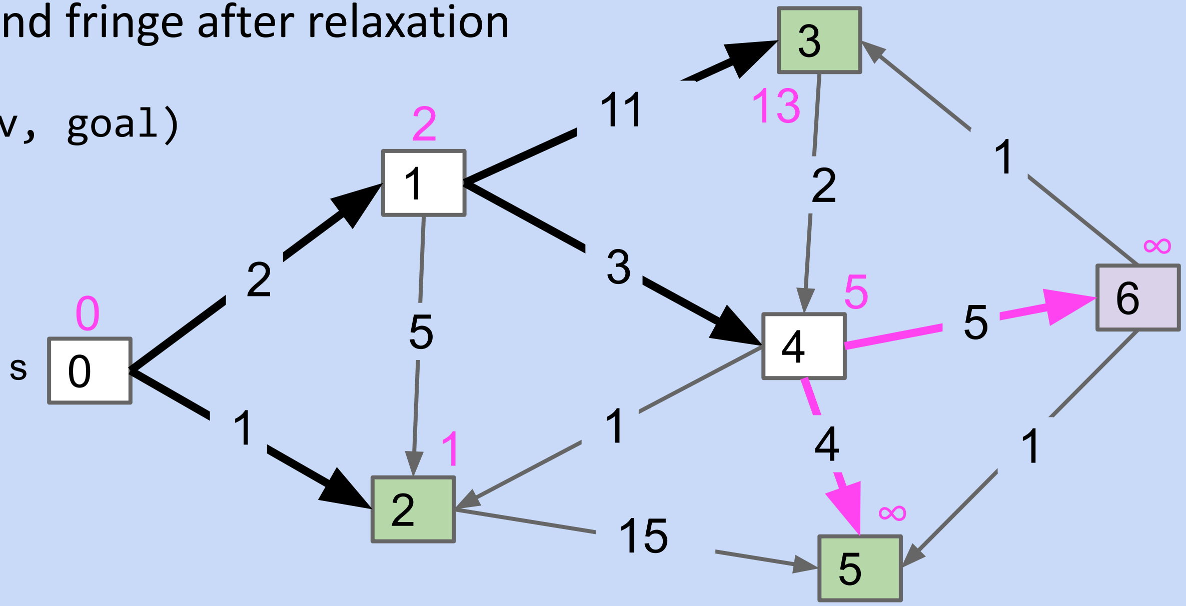
A* Demo, with s = 0, goal = 6.

Insert all vertices into fringe PQ, storing vertices in order of d(source, v) + h(v, goal).

Repeat: Remove best vertex v from PQ, and relax all edges pointing from v.

- Give `distTo`, `edgeTo`, and fringe after relaxation

#	distTo	edgeTo	h(v, goal)
0	0	-	1
1	2	0	3
2	1	0	15
3	13	1	2
4	5	1	1
5	∞	-	∞
6	∞	-	0



`Fringe: [(3: 15), (2: 16), (5: ∞), (6: ∞)]`

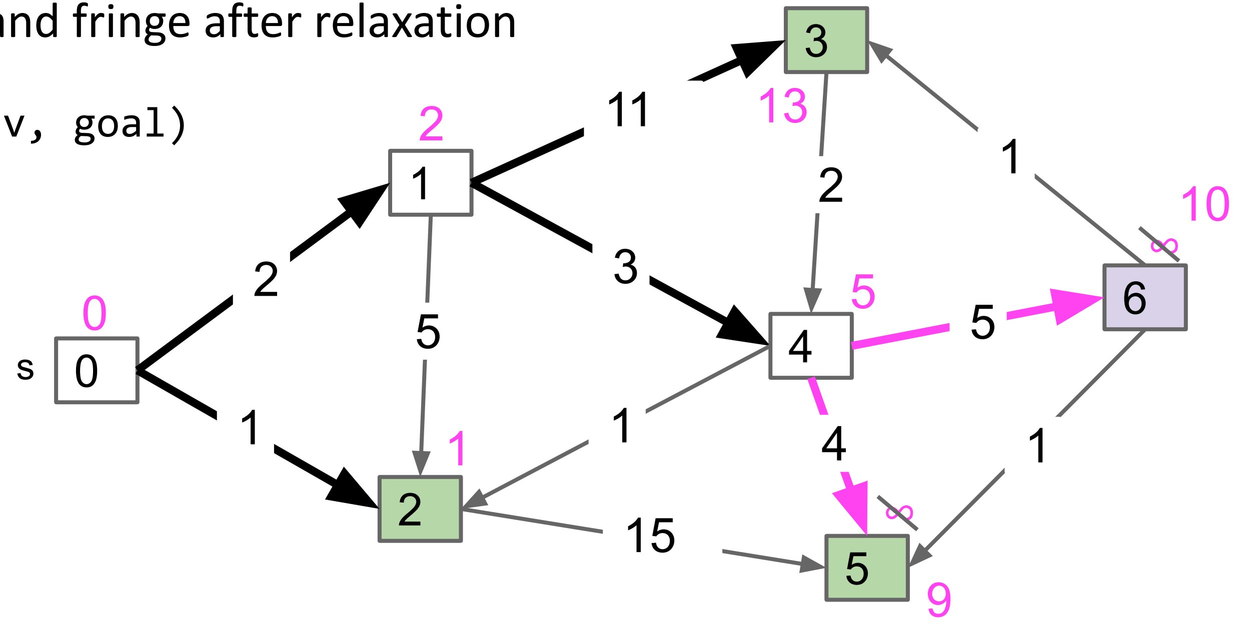
A* Demo, with s = 0, goal = 6.

Insert all vertices into fringe PQ, storing vertices in order of d(source, v) + h(v, goal).

Repeat: Remove best vertex v from PQ, and relax all edges pointing from v.

- Give `distTo`, `edgeTo`, and fringe after relaxation

#	distTo	edgeTo	h(v, goal)
0	0	-	1
1	2	0	3
2	1	0	15
3	13	1	2
4	5	1	1
5	9	4	∞
6	10	4	0



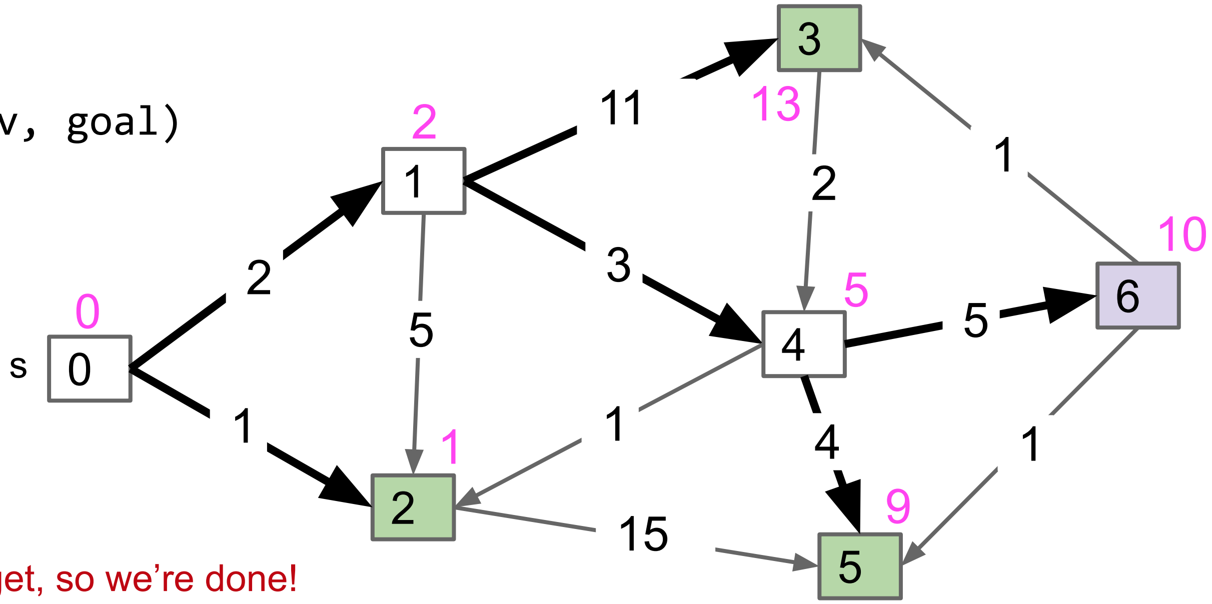
`Fringe: [(6: 10), (3: 15), (2: 16), (5: ∞)]`

A* Demo, with s = 0, goal = 6.

Insert all vertices into fringe PQ, storing vertices in order of d(source, v) + h(v, goal).

Repeat: Remove best vertex v from PQ, and relax all edges pointing from v.

#	distTo	edgeTo	h(v, goal)
0	0	-	1
1	2	0	3
2	1	0	15
3	13	1	2
4	5	1	1
5	9	4	∞
6	10	4	0



Next vertex to be dequeued is our target, so we're done!

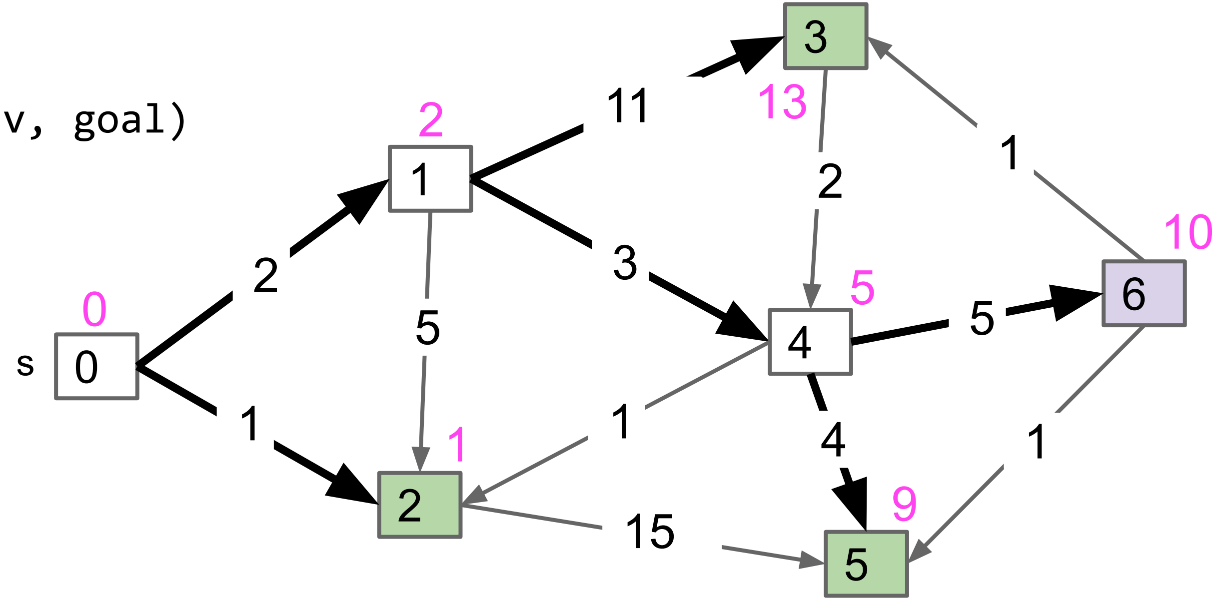
Fringe: [(6: 10), (3: 15), (2: 16), (5: ∞)]

A* Demo, with s = 0, goal = 6.

Insert all vertices into fringe PQ, storing vertices in order of d(source, v) + h(v, goal).

Repeat: Remove best vertex v from PQ, and relax all edges pointing from v.

#	distTo	edgeTo	h(v, goal)
0	0	-	1
1	2	0	3
2	1	0	15
3	13	1	2
4	5	1	1
5	9	4	∞
6	10	4	0



Observations:

- Not every vertex got visited.
- Result is not a shortest paths tree for vertex zero (path to 3 is suboptimal!), but that's OK because we only care about path to 6.