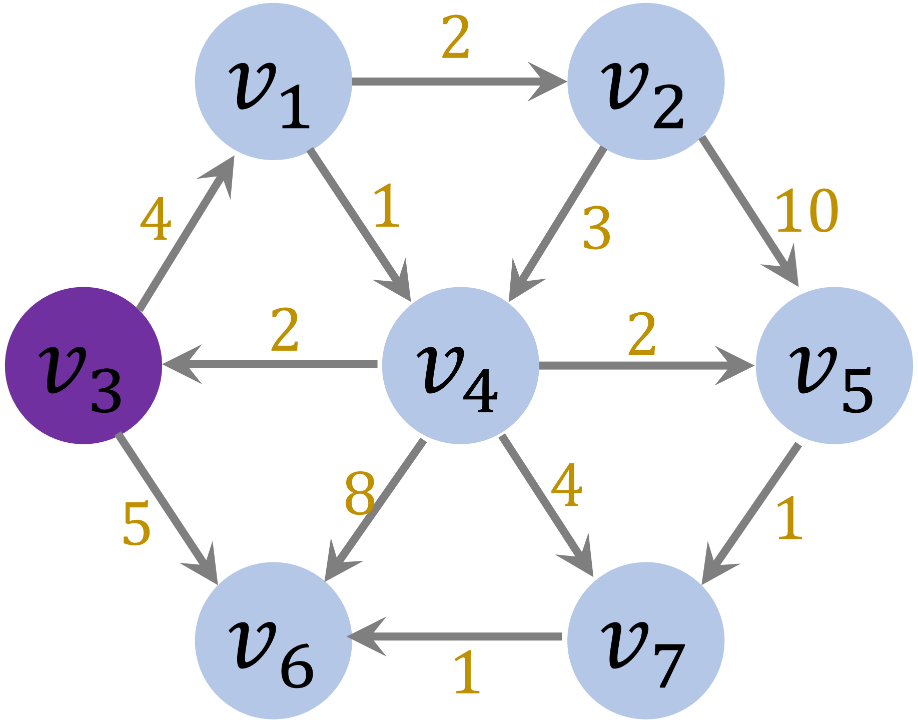
Finding Shortest-Path in Weighted Graphs

Shusen Wang

Single-Source Shortest Path in Weighted Graph

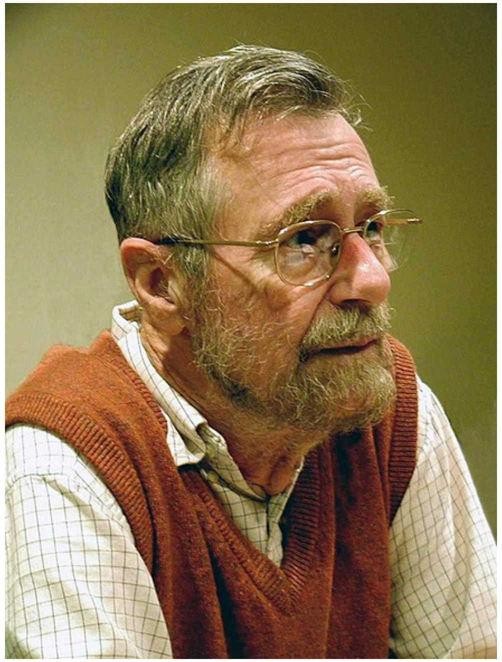


vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	**7**	v_4
v_6	**5**	v_3
v_7	**9**	0

- v_3 is the source.

Dijkstra’s Algorithm

Dijkstra’s Algorithm



Edsger W. Dijkstra

1930 – 2002

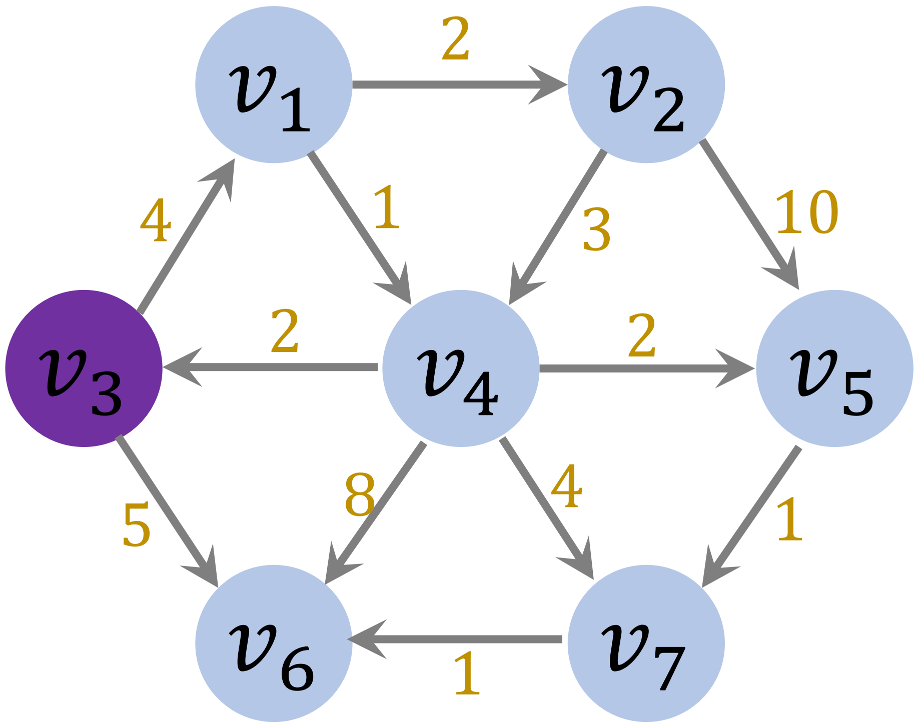
Won Turing Award in 1972

- Dijkstra’s algorithm is for solving the single-source shortest path problem.
- Published in 1959 [1].

Reference

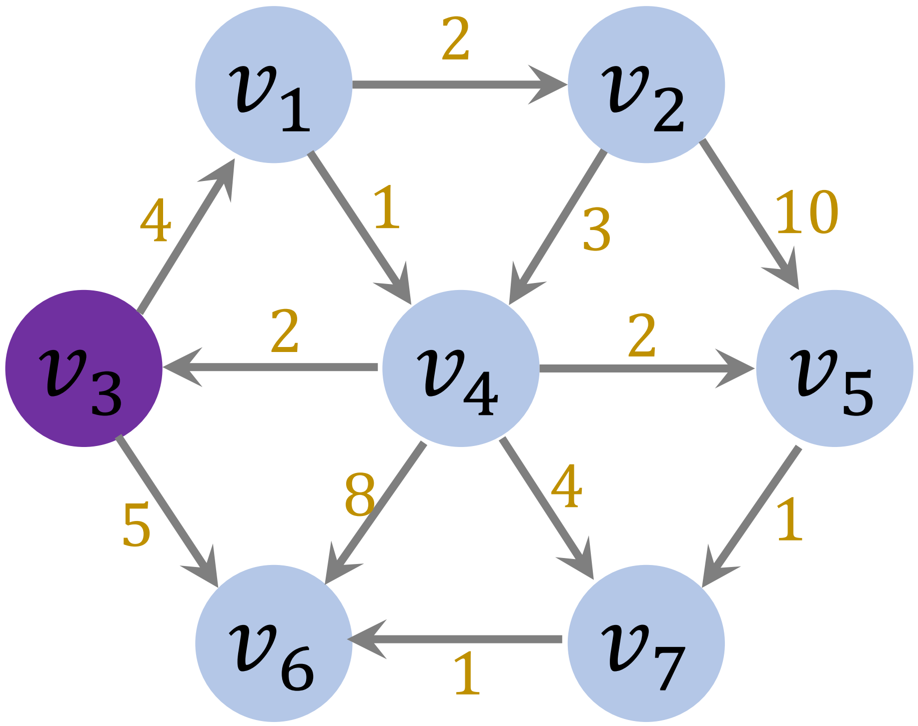
1. E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*. 1: 269–271, 1959.

Preparations



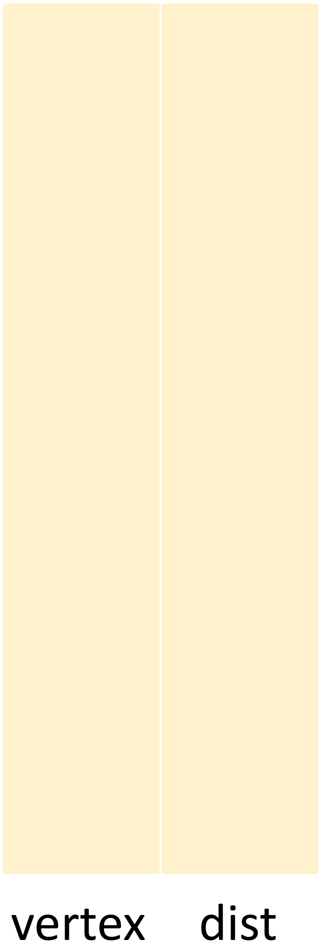
- v_3 is the source.

Preparations



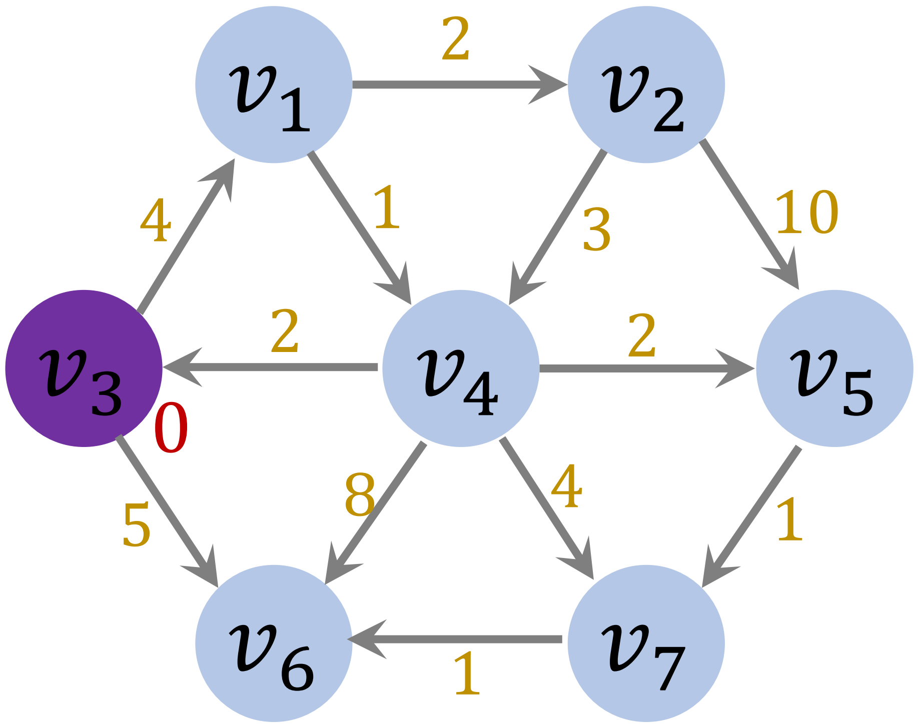
- v_3 is the source.

Priority Queue:



vertex	dist	path
v_1	∞	0
v_2	∞	0
v_3	∞	0
v_4	∞	0
v_5	∞	0
v_6	∞	0
v_7	∞	0

Initial State



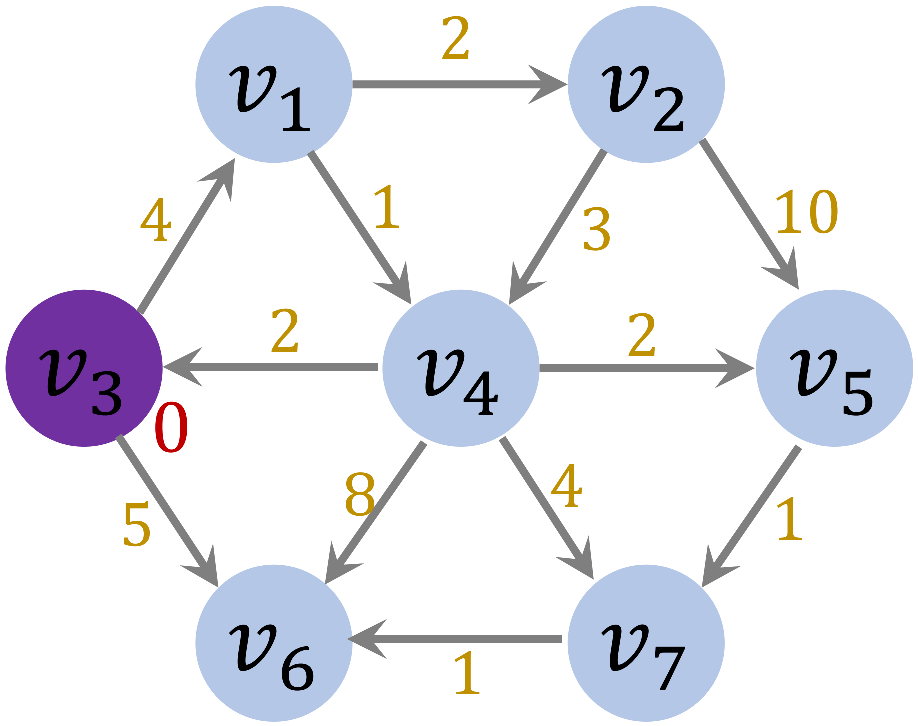
- v_3 is the source.
- Set v_3's distance to 0.

Priority Queue:

vertex	dist

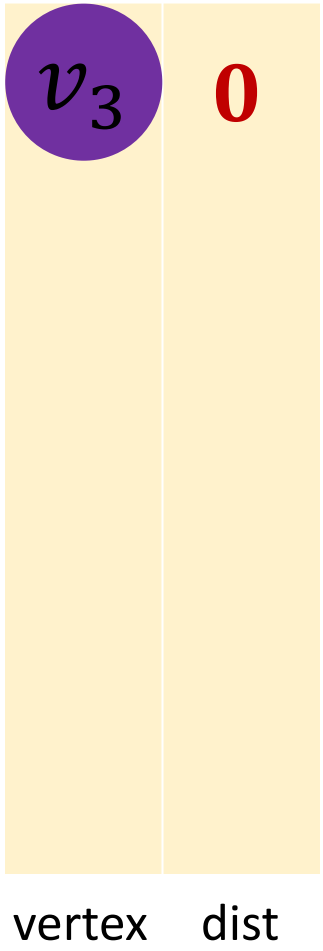
vertex	dist	path
v_1	∞	0
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	∞	0
v_7	∞	0

Initial State



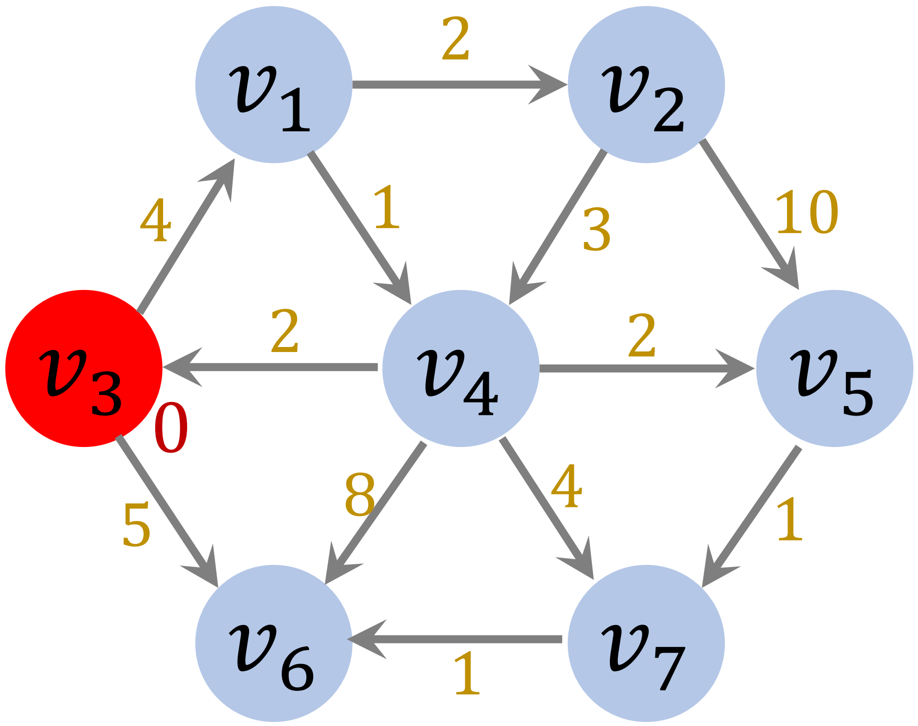
- enqueue(v_3, 0).

Priority Queue:



vertex	dist	path
v_1	∞	0
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	∞	0
v_7	∞	0

Iteration 1



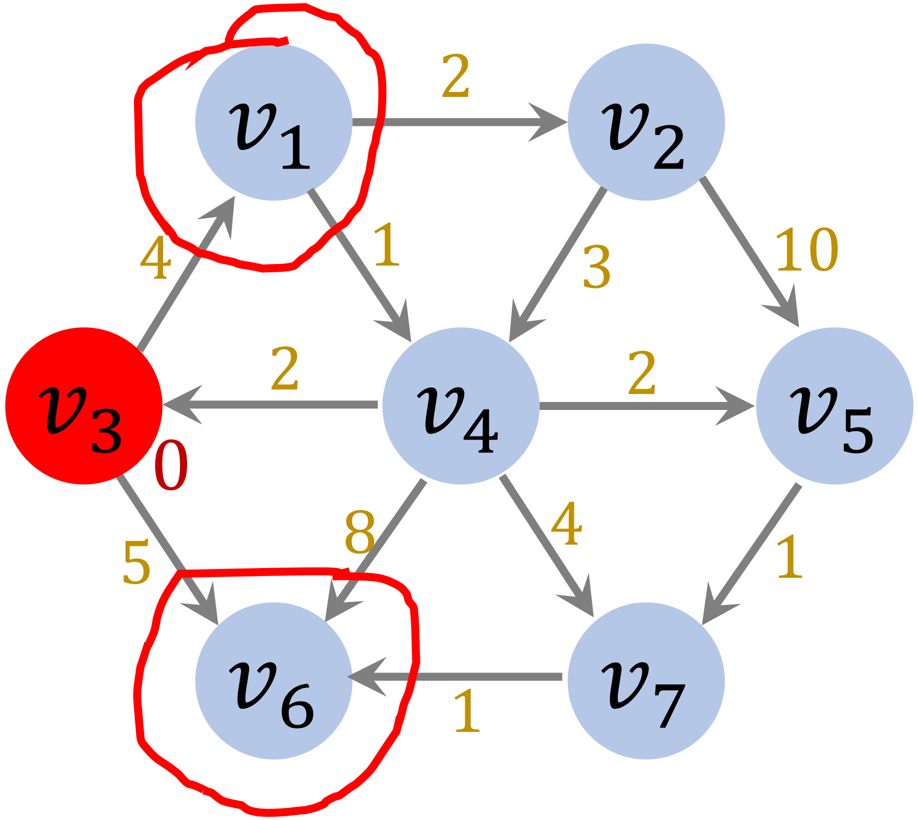
- $v_3 \leftarrow$ dequeue().

Priority Queue:

vertex	dist
v_3	**0**

vertex	dist	path
v_1	∞	0
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	∞	0
v_7	∞	0

Iteration 1



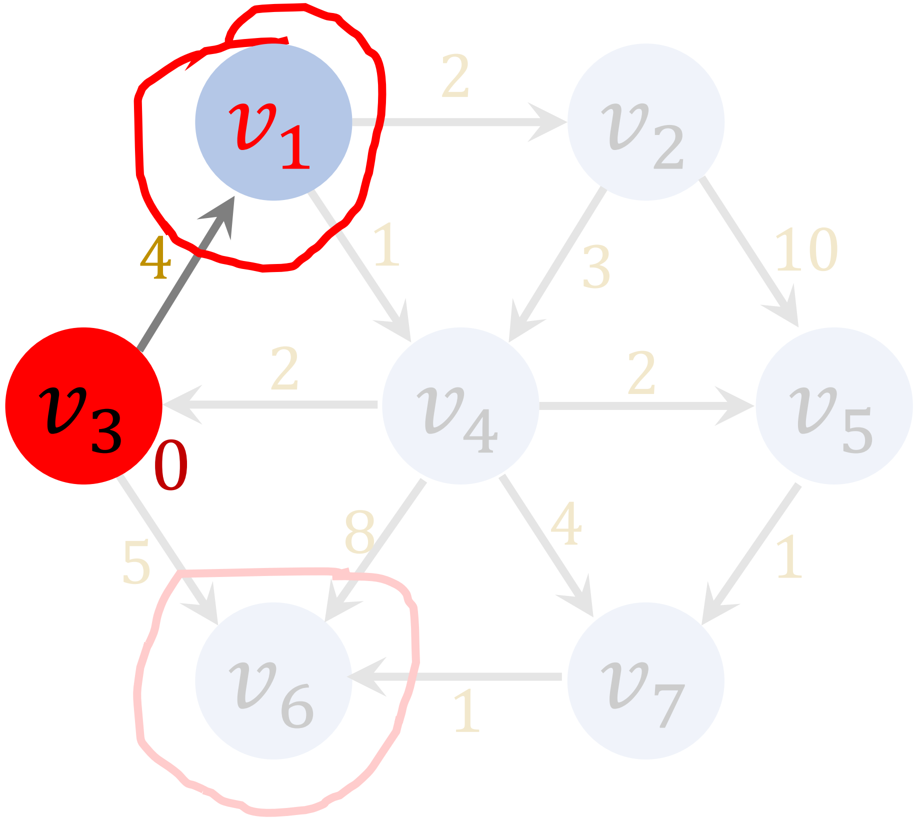
- $v_3 \leftarrow$ dequeue().
- Find adjacent vertices of v_3: v_1 and v_6.

Priority Queue:

vertex	dist

vertex	dist	path
v_1	∞	0
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	∞	0
v_7	∞	0

Iteration 1(A)



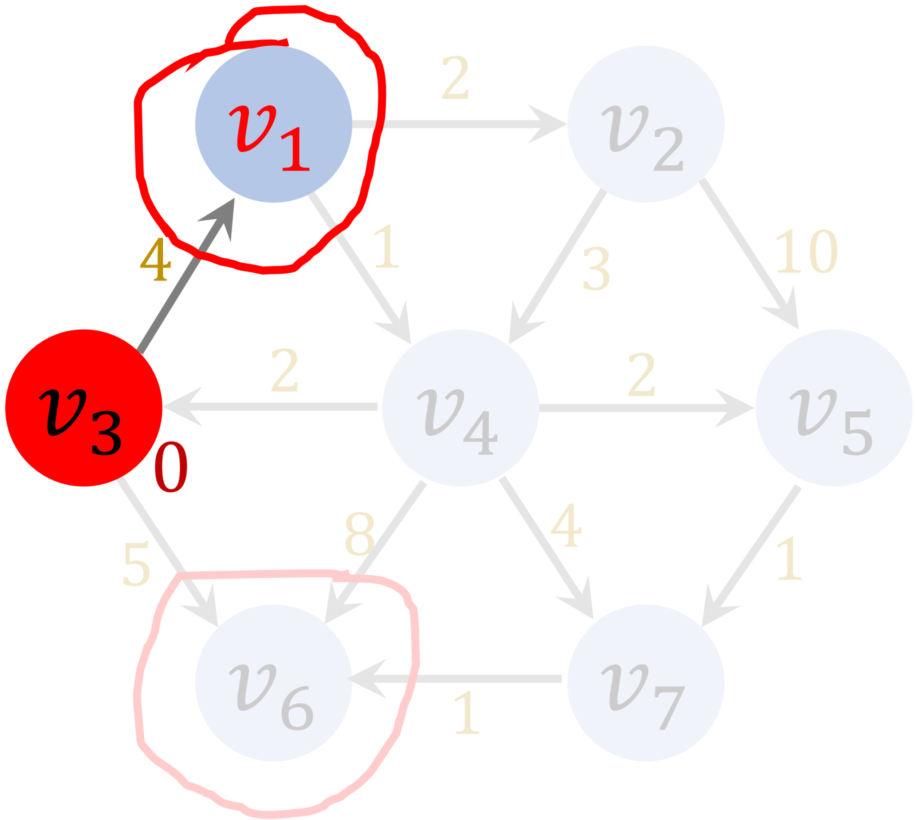
- Work on v_1.

Priority Queue:

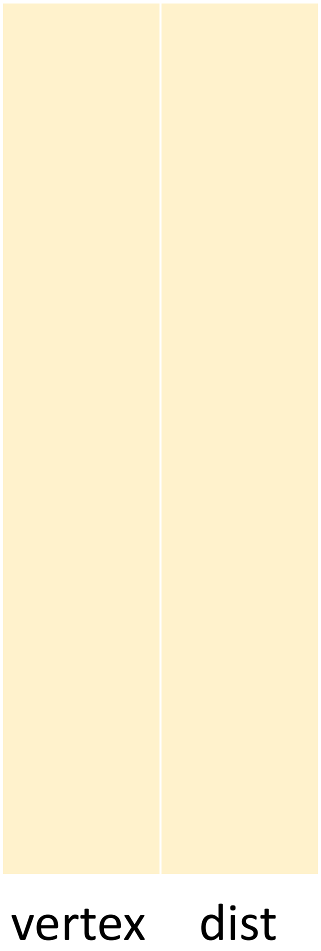
vertex	dist

vertex	dist	path
v_1	∞	0
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	∞	0
v_7	∞	0

Iteration 1(A)



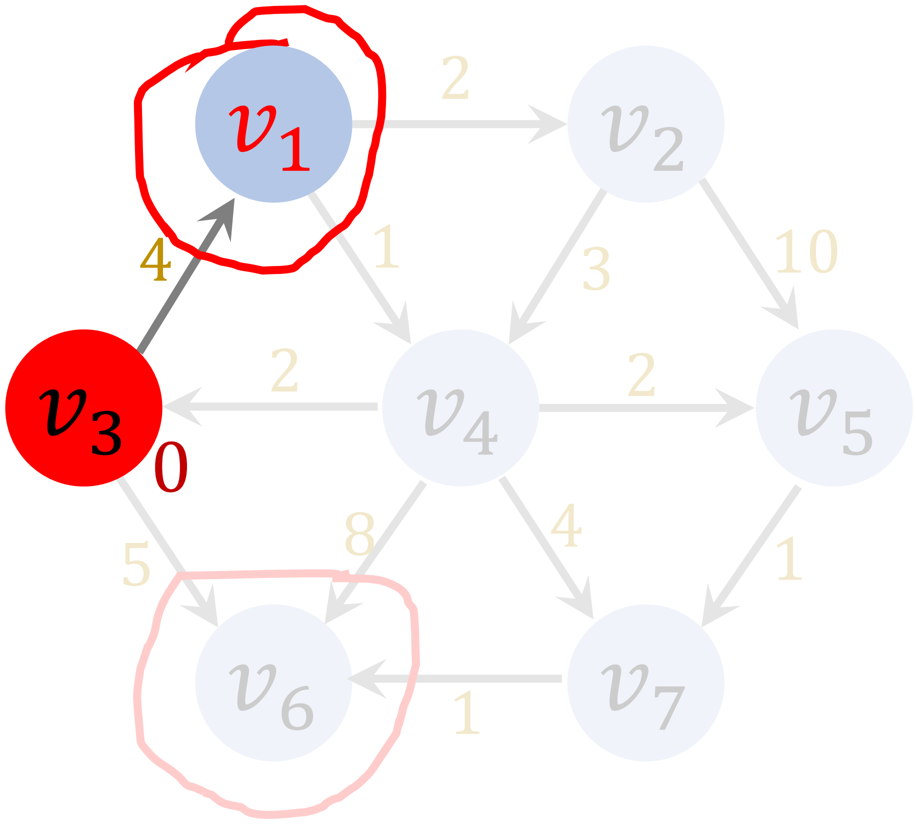
Priority Queue:



vertex	dist	path
v_1	∞	0
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	∞	0
v_7	∞	0

- $d = \text{dist}[3] + 4 = 4.$
- Since $d < \infty$, update the table.

Iteration 1(A)



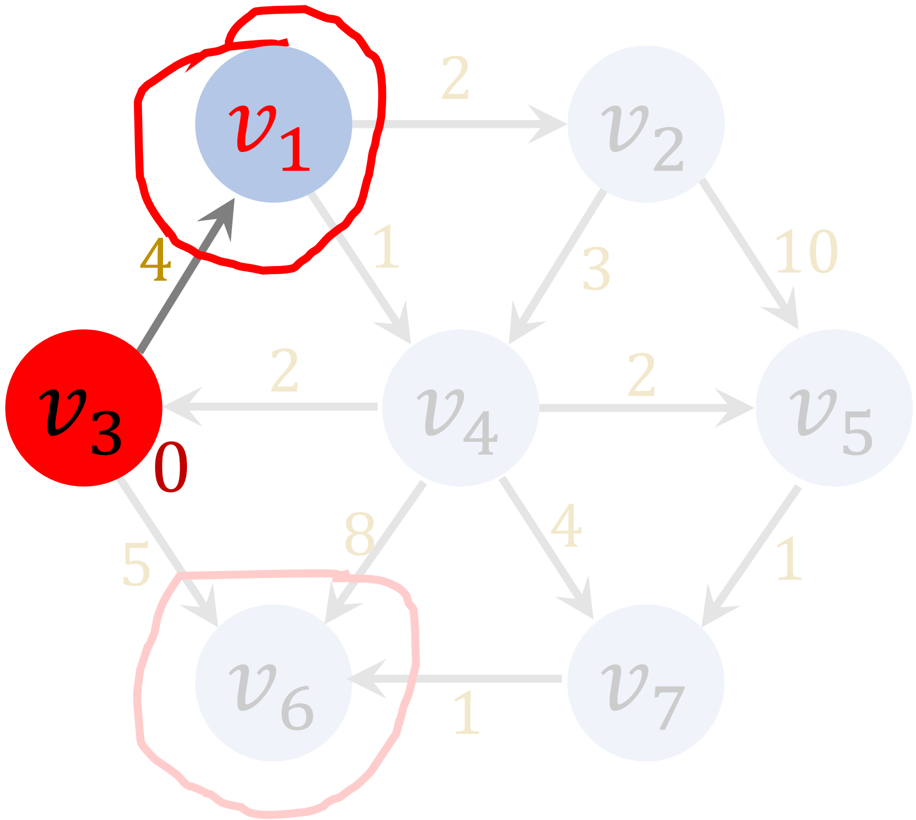
- $\text{dist}[1] = 4.$

Priority Queue:

vertex	dist

vertex	dist	path
v_1	4	0
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	∞	0
v_7	∞	0

Iteration 1(A)



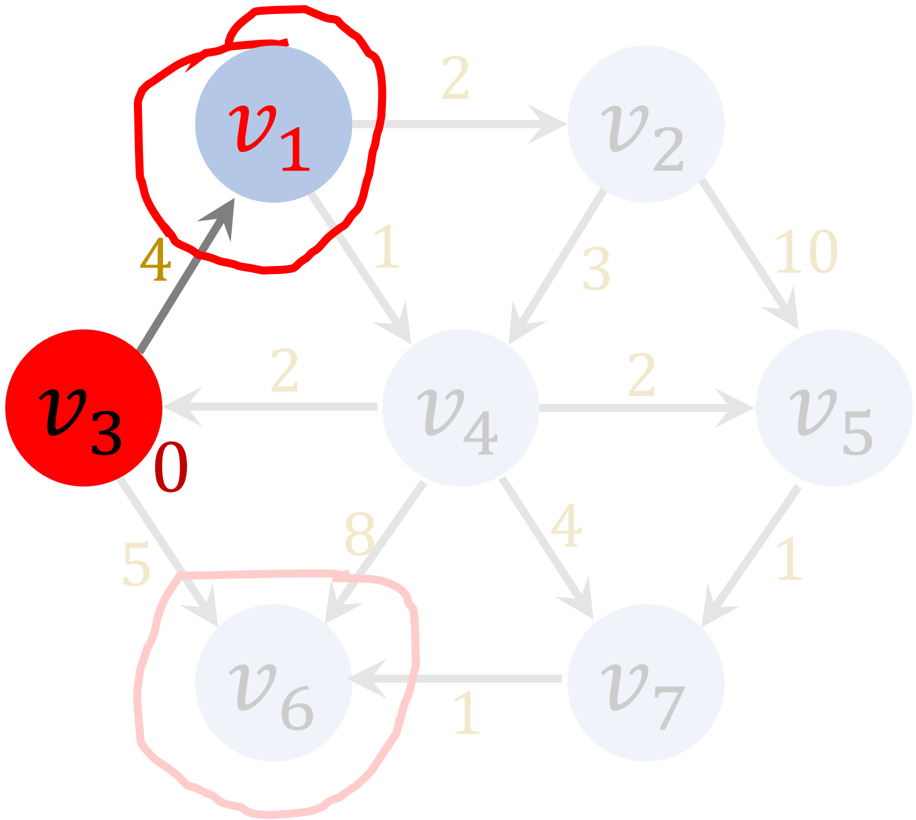
- $\text{dist}[1] = 4$.
- $\text{path}[1] = v_3$.

Priority Queue:

vertex	dist

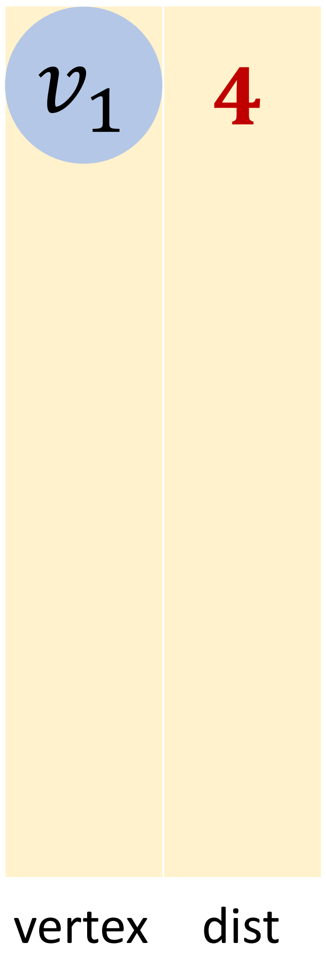
vertex	dist	path
v_1	**4**	v_3
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	∞	0
v_7	∞	0

Iteration 1(A)



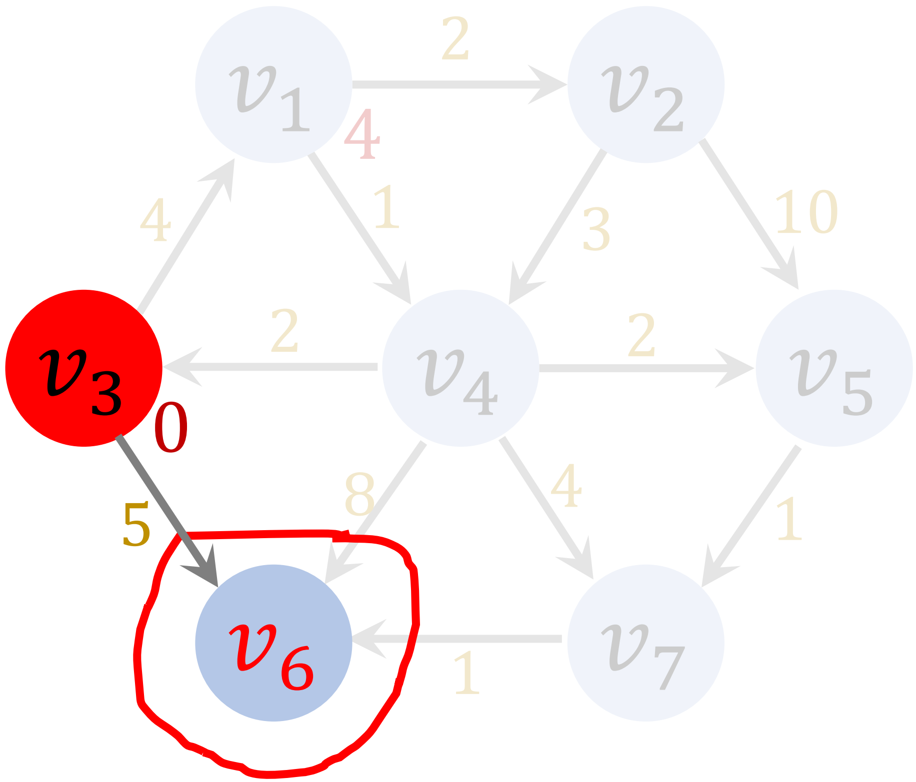
- enqueue(v_1, 4).

Priority Queue:



vertex	dist	path
v_1	**4**	v_3
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	∞	0
v_7	∞	0

Iteration 1(B)



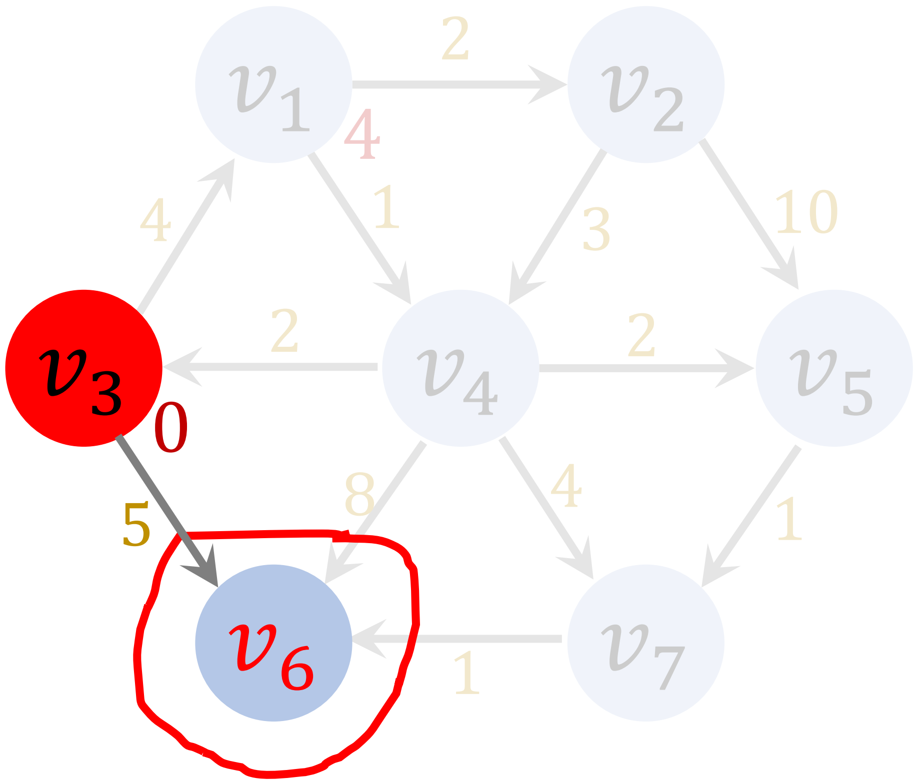
- Work on v_6.

Priority Queue:

vertex	dist
v_1	**4**

vertex	dist	path
v_1	**4**	v_3
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	∞	0
v_7	∞	0

Iteration 1(B)



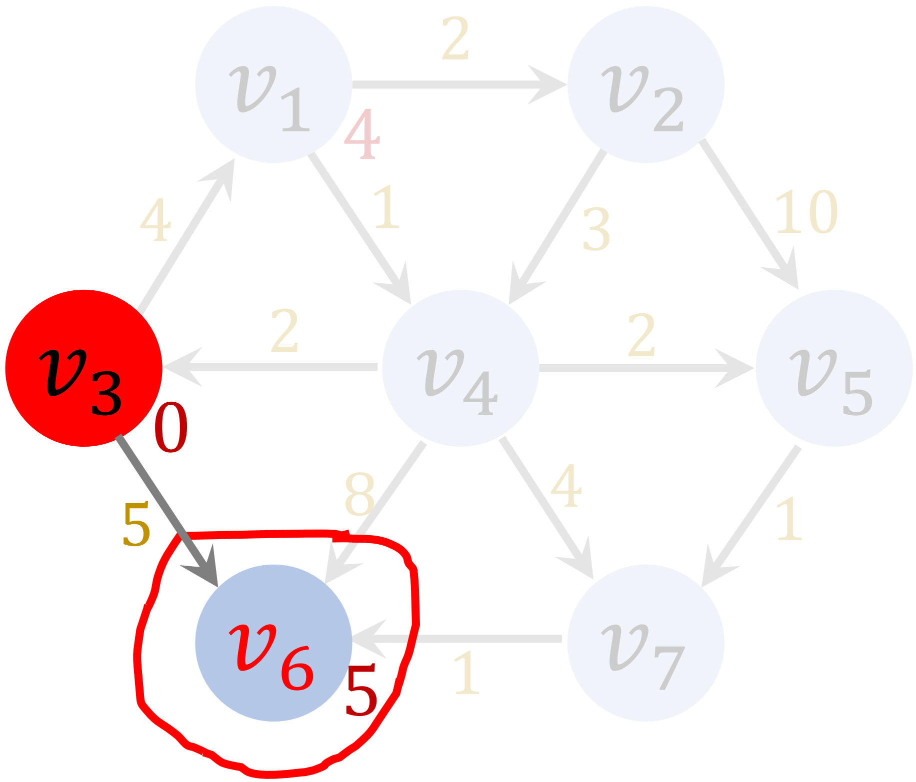
- $d = \text{dist}[3] + 5 = 5.$
- Since $d < \infty$, update the table.

Priority Queue:

vertex	dist
v_1	4

vertex	dist	path
v_1	4	v_3
v_2	∞	0
v_3	0	0
v_4	∞	0
v_5	∞	0
v_6	∞	0
v_7	∞	0

Iteration 1(B)



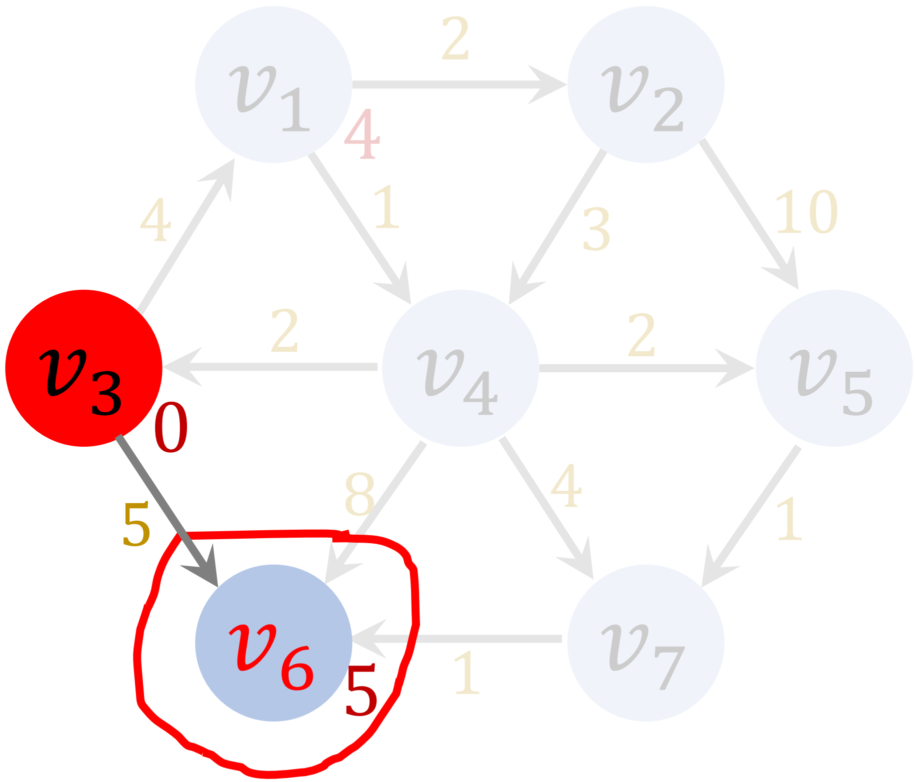
- $\text{dist}[6] = 5$.

Priority Queue:

vertex	dist
v_1	4

vertex	dist	path
v_1	4	v_3
v_2	∞	0
v_3	0	0
v_4	∞	0
v_5	∞	0
v_6	5	0
v_7	∞	0

Iteration 1(B)



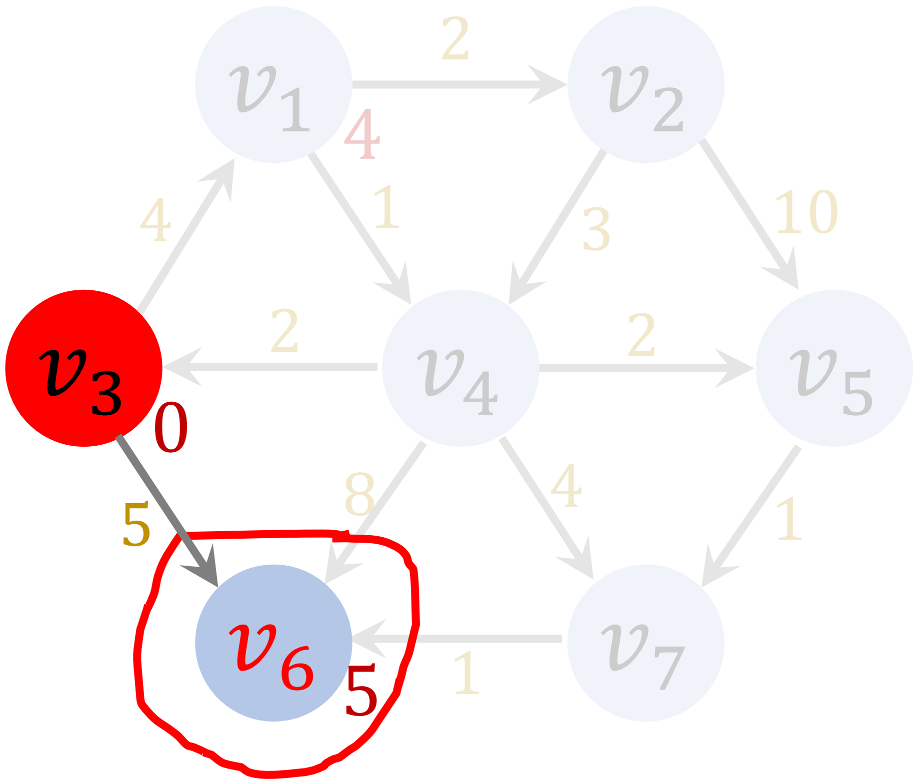
- $\text{dist}[6] = 5.$
- $\text{path}[6] = v_3.$

Priority Queue:

vertex	dist
v_1	**4**

vertex	dist	path
v_1	**4**	v_3
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 1(B)



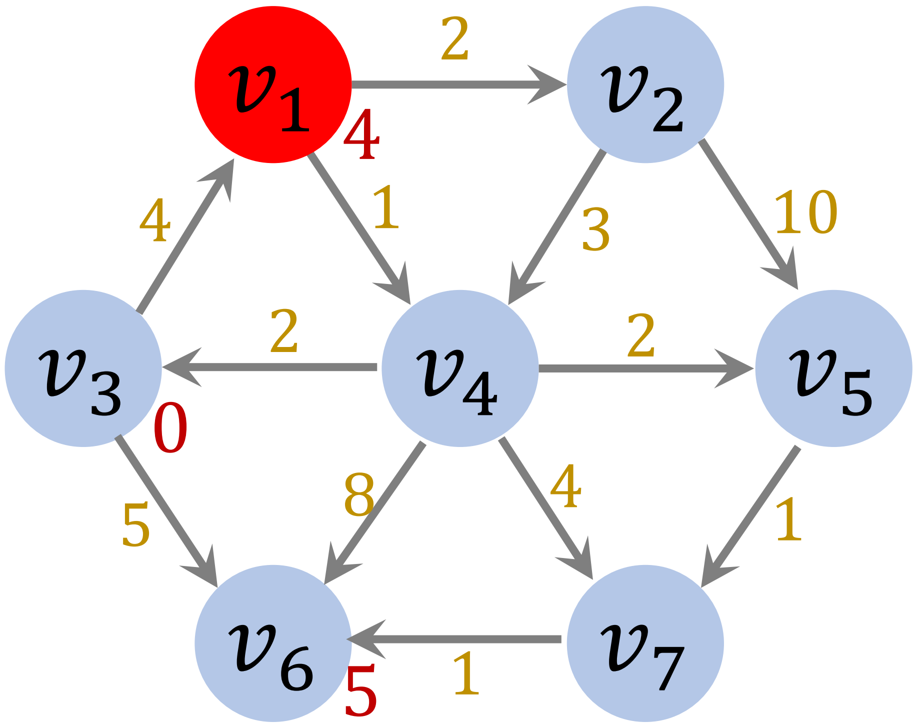
- enqueue(v_6, 5).

Priority Queue:

vertex	dist
v_1	**4**
v_6	**5**

vertex	dist	path
v_1	**4**	v_3
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 2



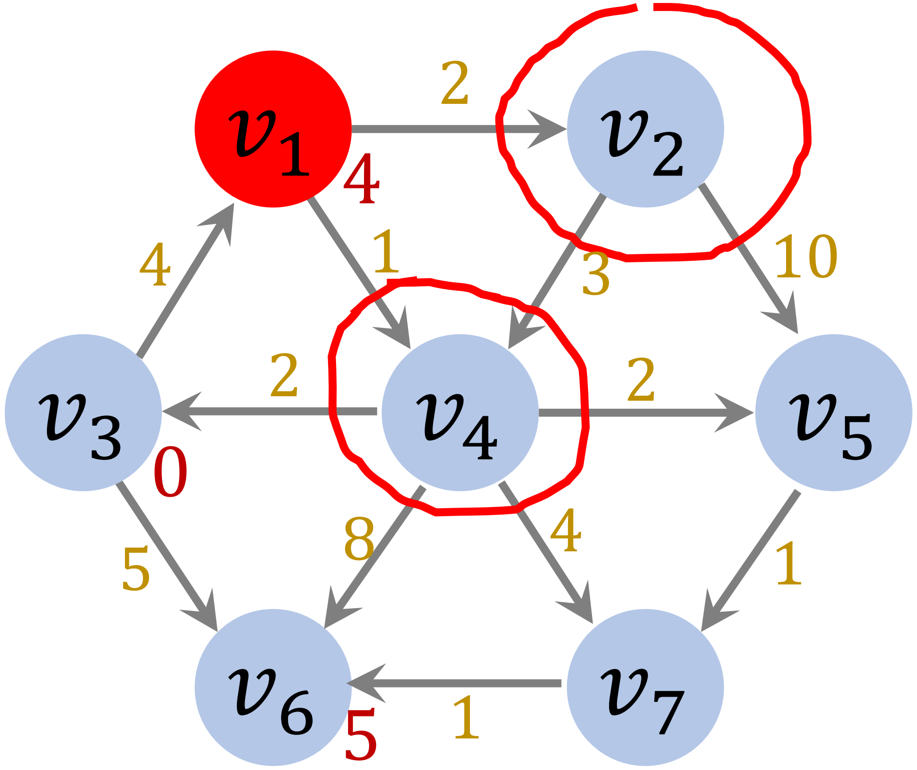
- $v_1 \leftarrow$ dequeue().

Priority Queue:

vertex	dist
v_1	**4**
v_6	**5**

vertex	dist	path
v_1	**4**	v_3
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 2



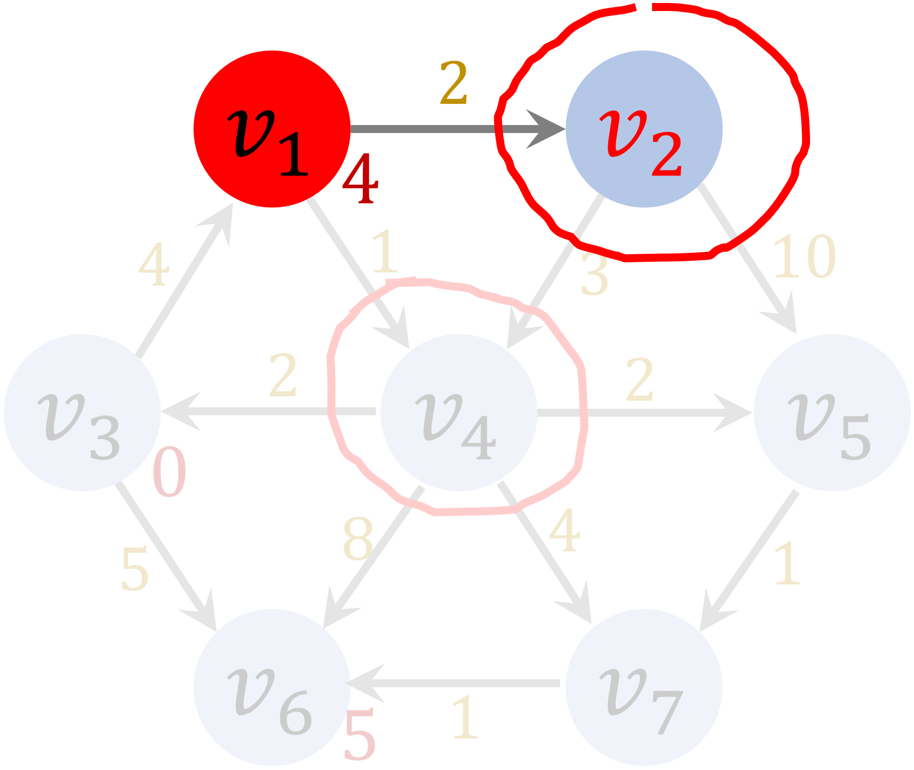
- $v_1 \leftarrow$ dequeue().
- Find adjacent vertices of v_1: v_2 and v_4.

Priority Queue:

vertex	dist
v_6	**5**

vertex	dist	path
v_1	**4**	v_3
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 2(A)



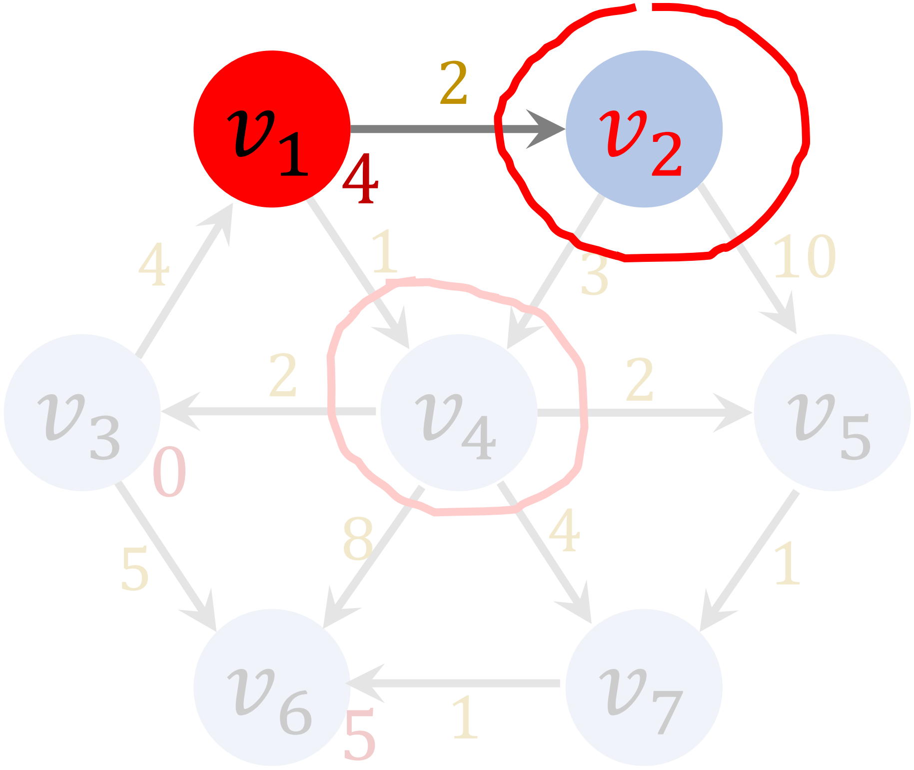
- Work on v_2.

Priority Queue:

vertex	dist
v_6	5

vertex	dist	path
v_1	**4**	v_3
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 2(A)



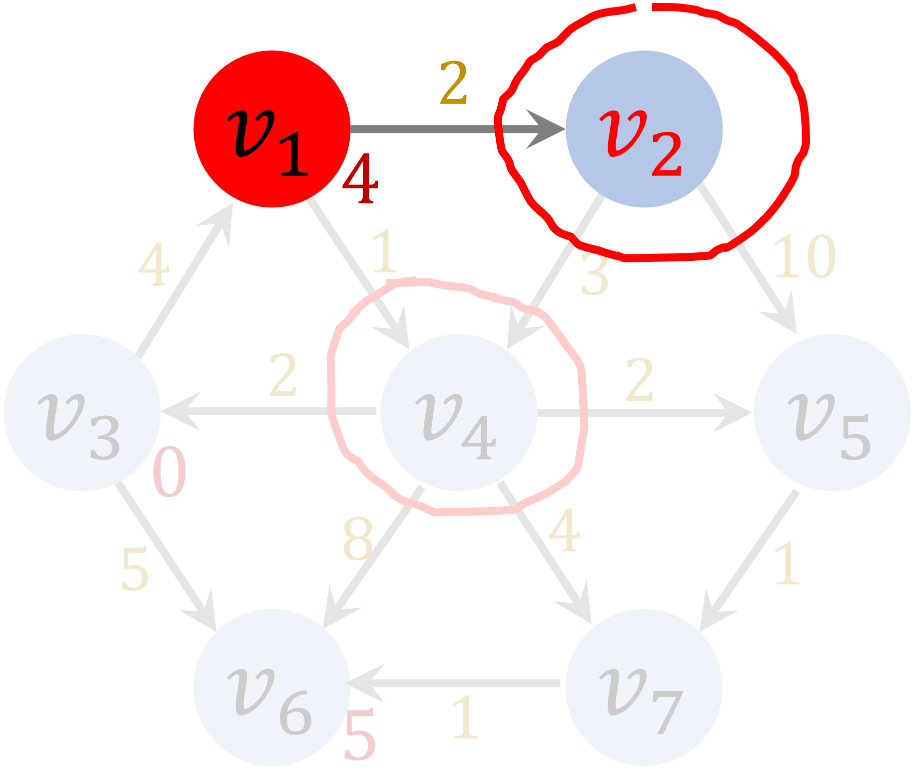
Priority Queue:

vertex	dist
v_6	**5**

vertex	dist	path
v_1	**4**	v_3
v_2	∞	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

- $d = \text{dist}[1] + 2 = 6.$
- Since $d < \infty$, update the table.

Iteration 2(A)



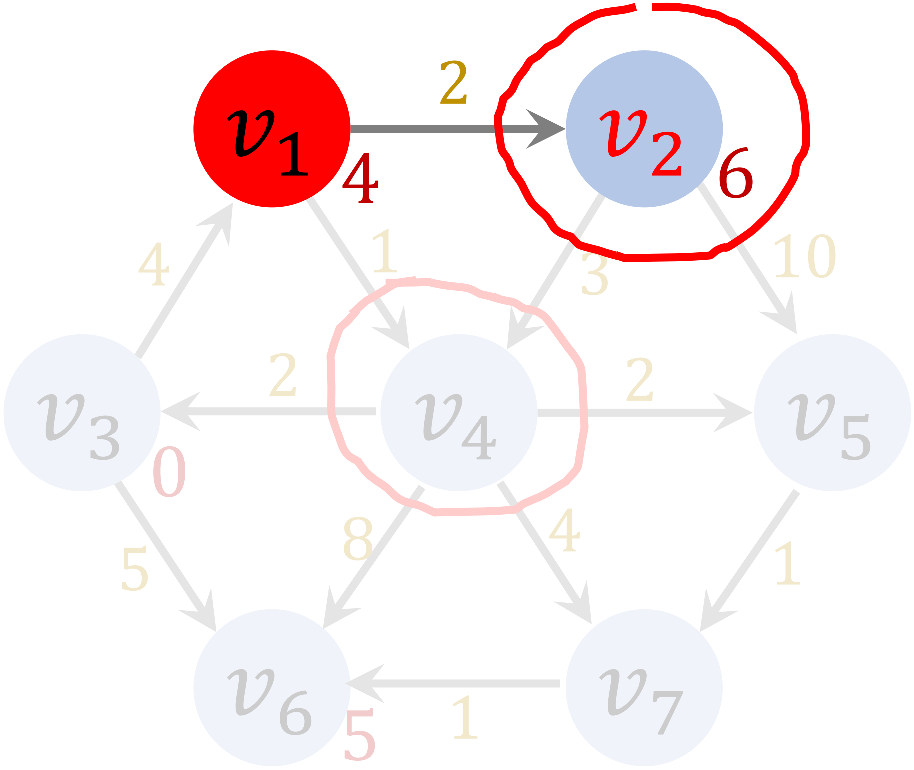
- dist[2] = 6.

Priority Queue:

vertex	dist
v_6	**5**

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	0
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 2(A)



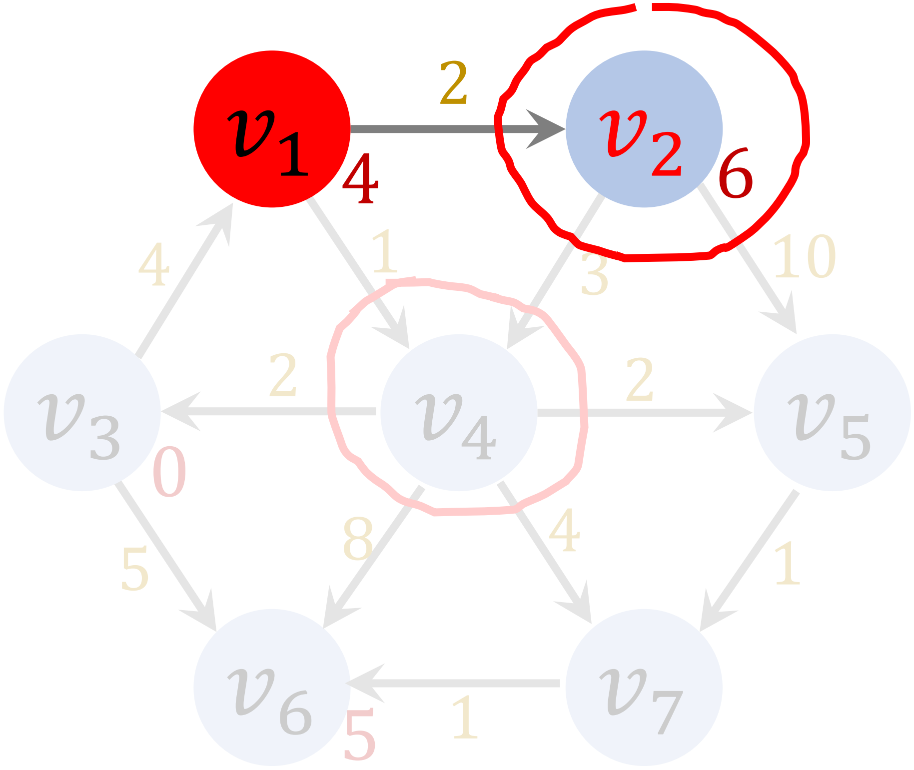
Priority Queue:

vertex	dist
v_6	**5**

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

- $\text{dist}[2] = 6$.
- $\text{path}[2] = v_1$.

Iteration 2(A)



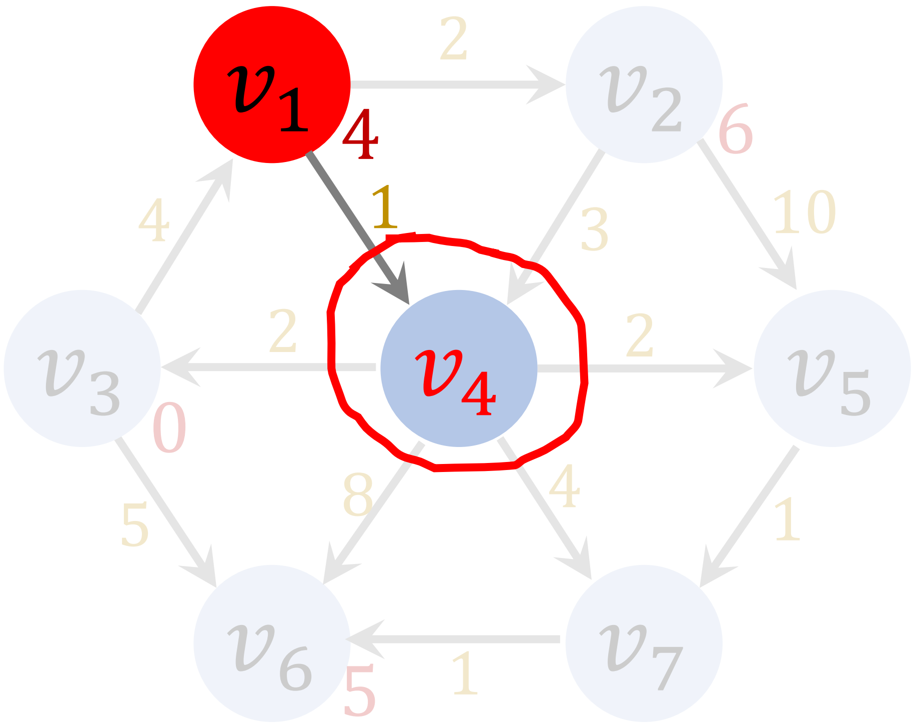
- enqueue(v_2, 6).

Priority Queue:

vertex	dist
v_6	**5**
v_2	**6**

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 2(B)



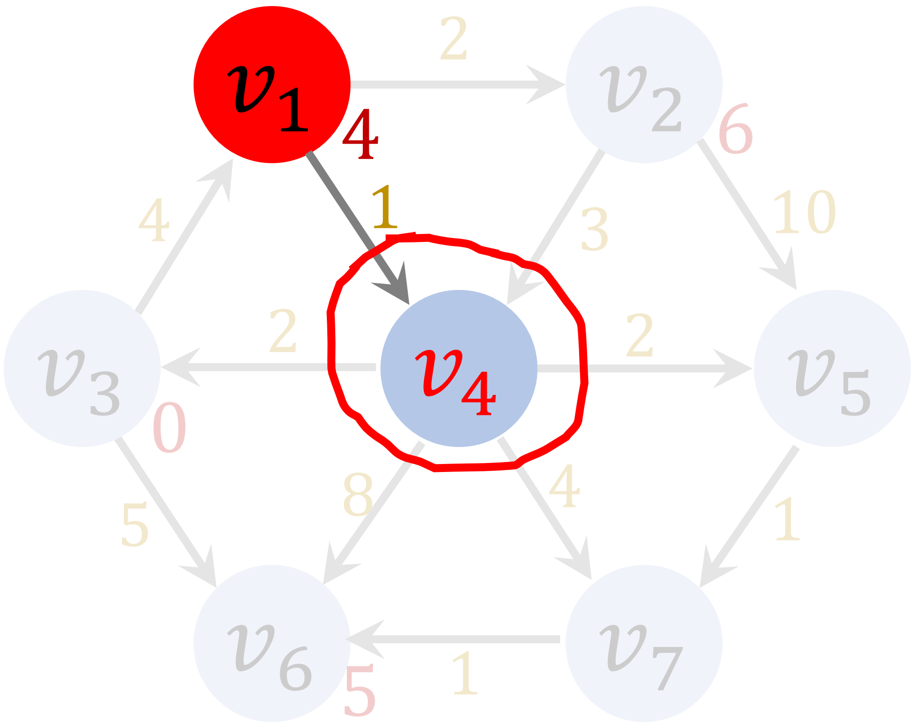
- Work on v_4.

Priority Queue:

vertex	dist
v_6	**5**
v_2	**6**

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 2(B)



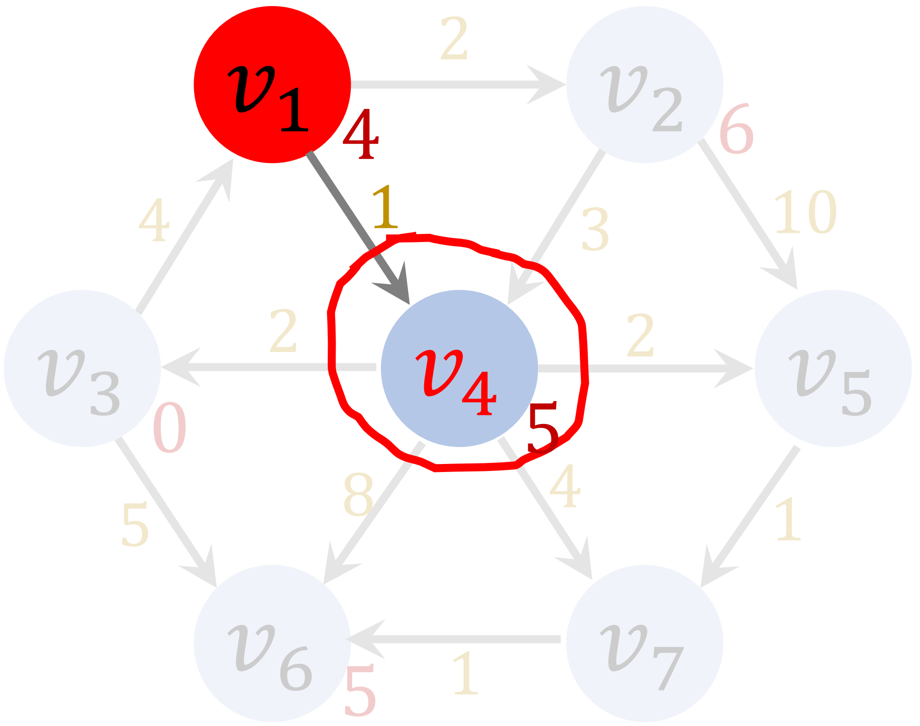
Priority Queue:

vertex	dist
v_6	**5**
v_2	**6**

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	∞	0
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

- $d = \text{dist}[1] + 1 = 5.$
- Since $d < \infty$, update the table.

Iteration 2(B)



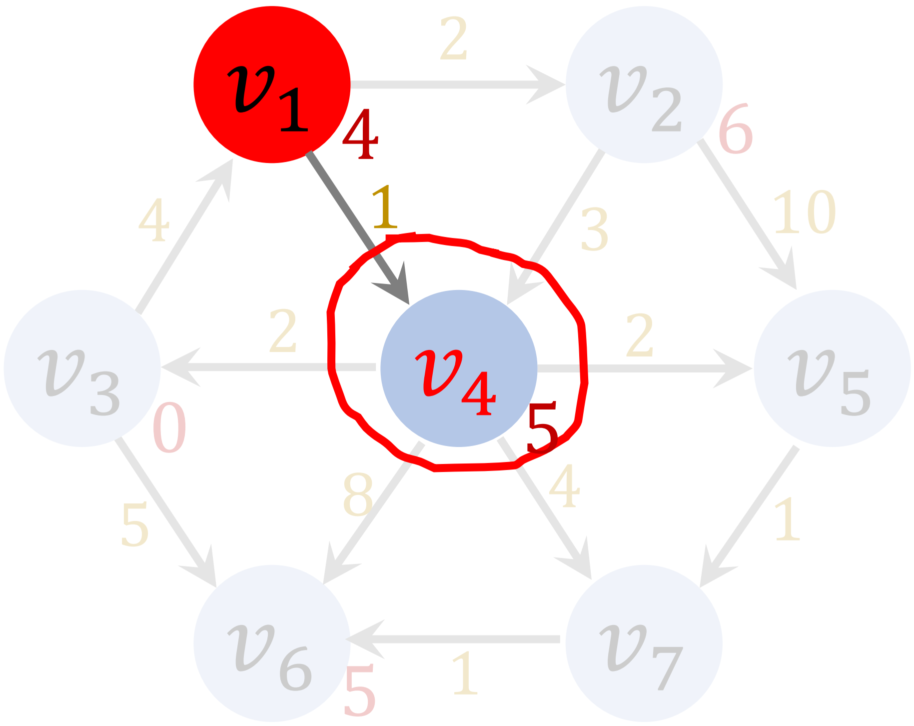
- $\text{dist}[4] = 5.$

Priority Queue:

vertex	dist
v_6	**5**
v_2	**6**

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	0
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 2(B)



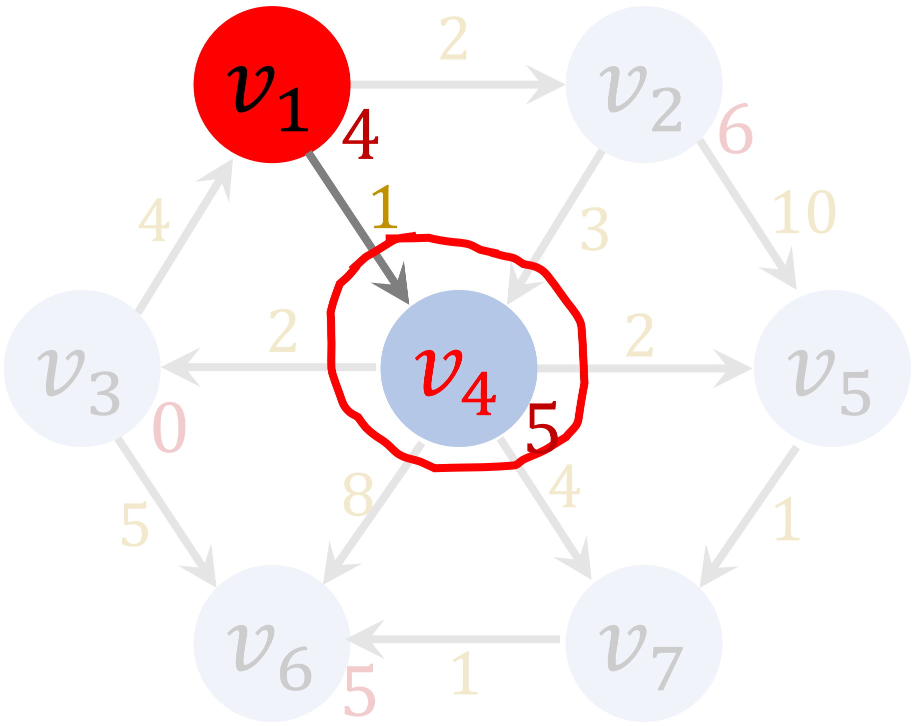
- dist[4] = 5.
- path[4] = v_1.

Priority Queue:

vertex	dist
v_6	**5**
v_2	**6**

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 2(B)



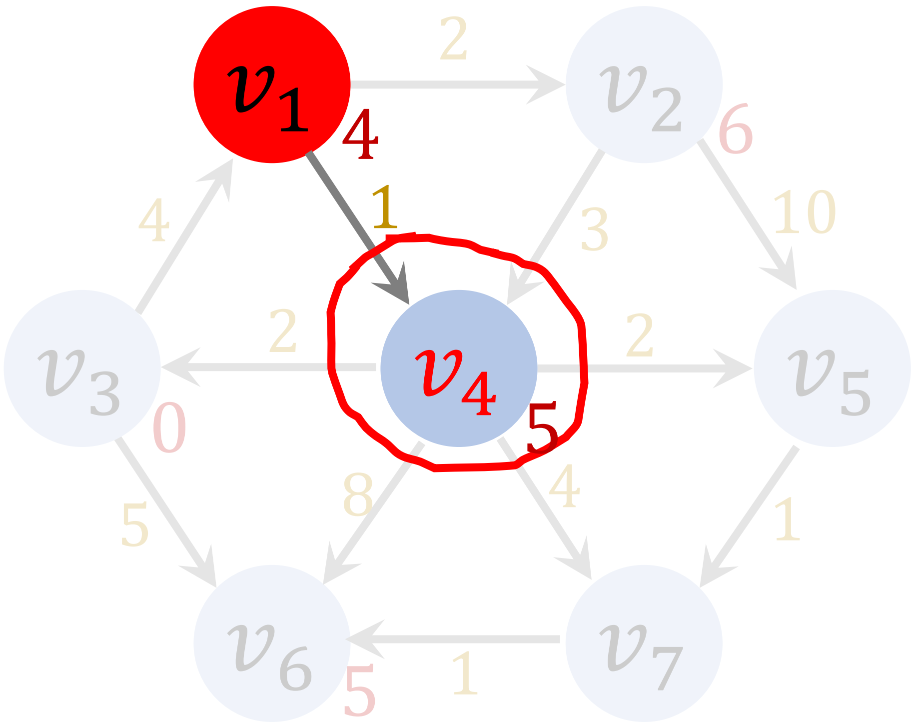
- enqueue(v_4, 5).

Priority Queue:

vertex	dist
v_6	**5**
v_2	**6**

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 2(B)



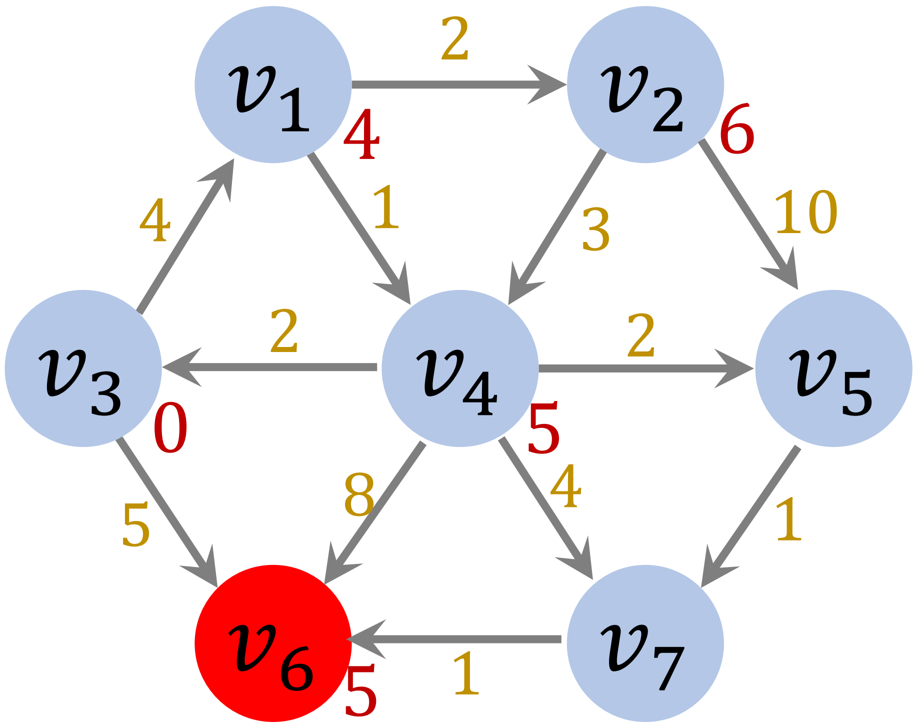
- enqueue(v_4, 5).

Priority Queue:

vertex	dist
v_6	**5**
v_4	**5**
v_2	**6**

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 3



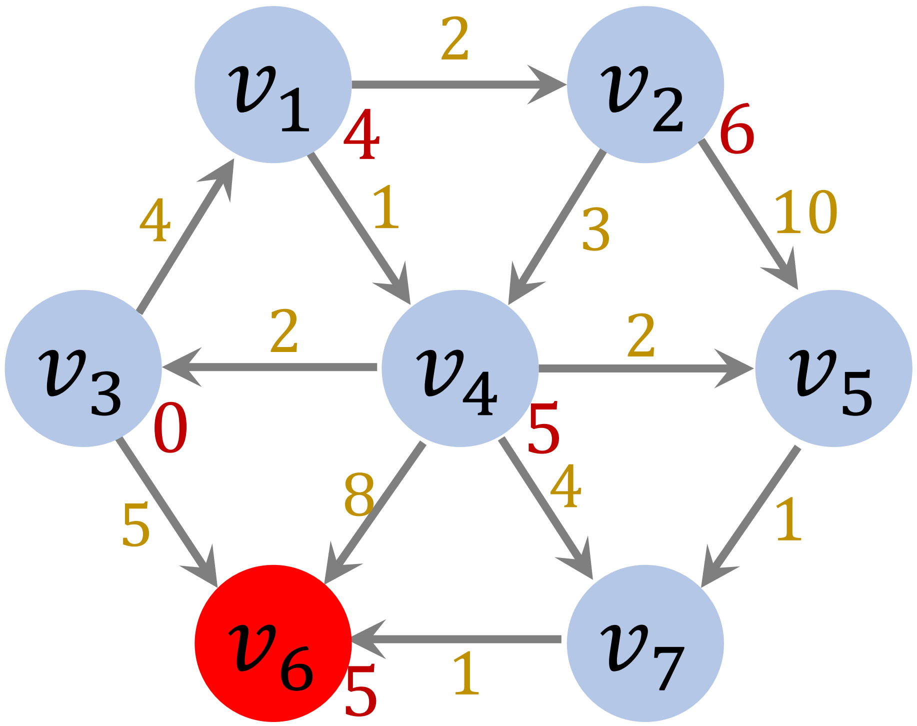
- $v_6 \leftarrow$ dequeue().

Priority Queue:

vertex	dist
v_6	**5**
v_4	**5**
v_2	**6**

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 3



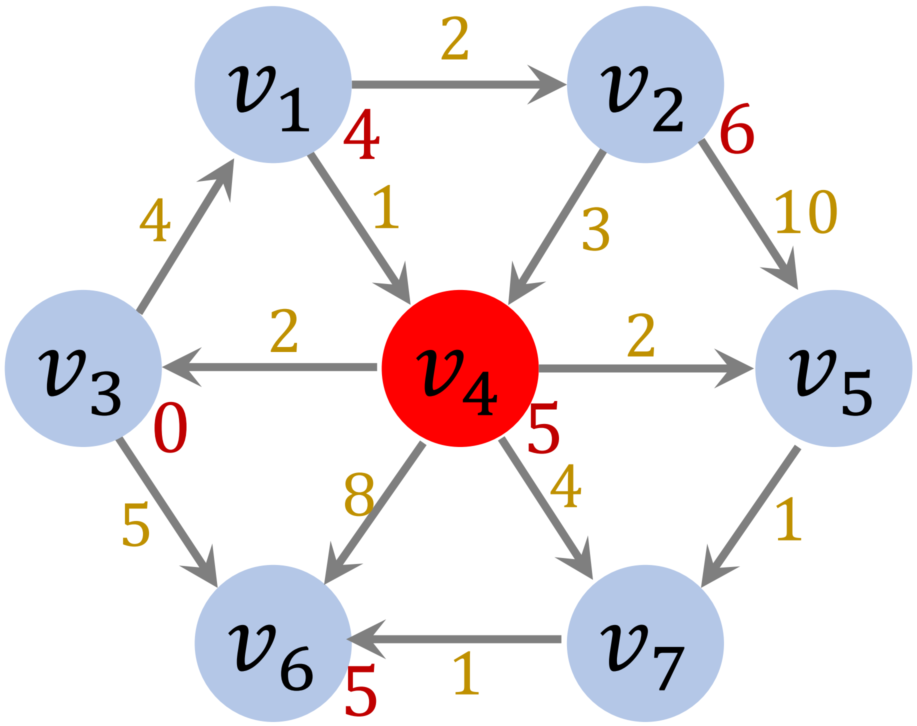
- $v_6 \leftarrow$ dequeue().
- v_6 has no adjacent vertex.
- ➔ Ignore v_6.

Priority Queue:

vertex	dist
v_4	**5**
v_2	**6**

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 4



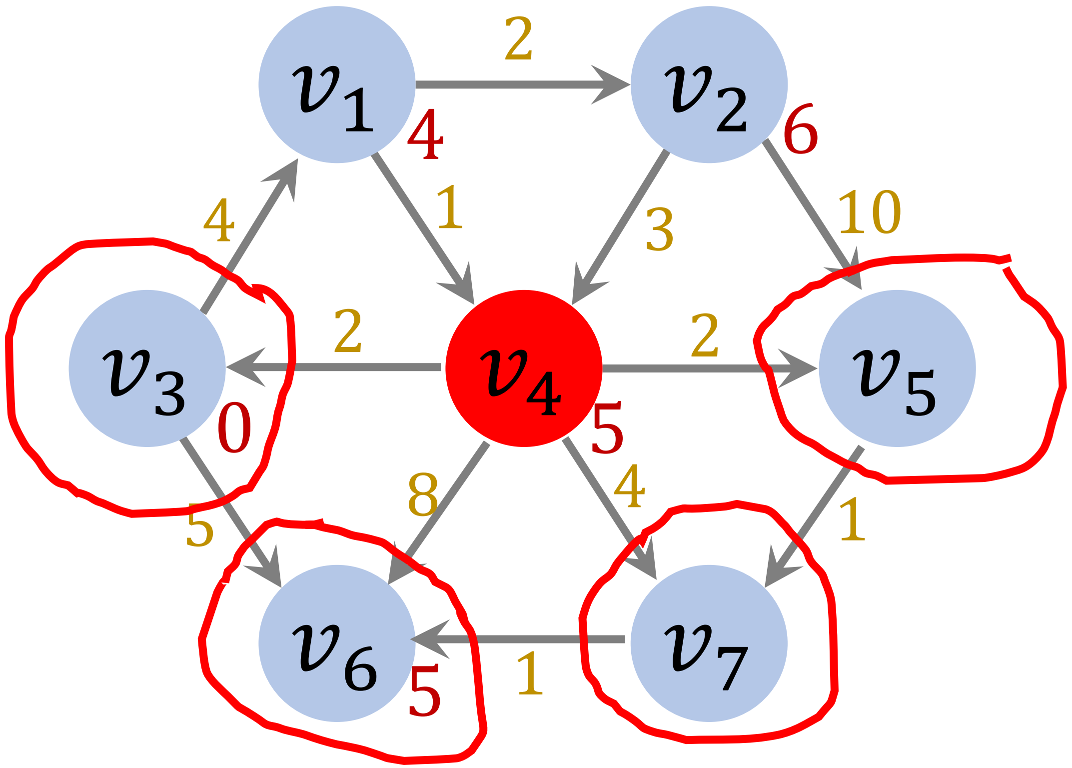
- $v_4 \leftarrow$ dequeue().

Priority Queue:

vertex	dist
v_4	5
v_2	6

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 4



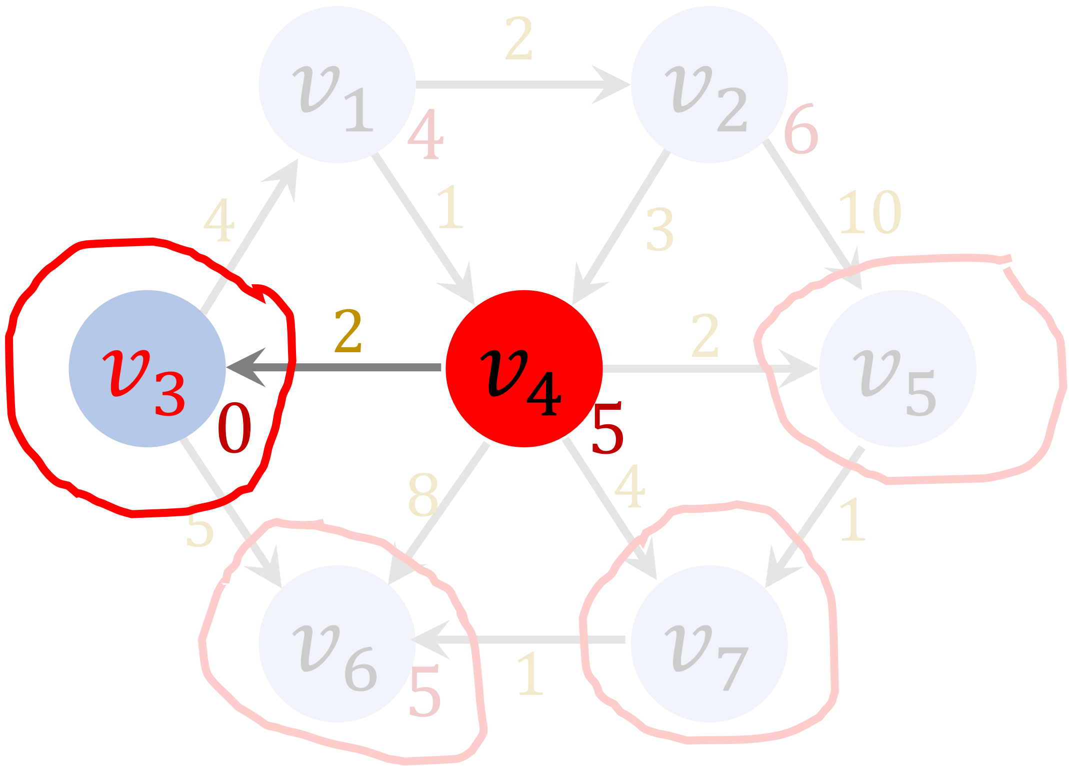
- $v_4 \leftarrow$ dequeue().
- Find adjacent vertices of v_4:
  v_3, v_5, v_6, and v_7.

Priority Queue:

vertex	dist
v_2	**6**

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 4(A)



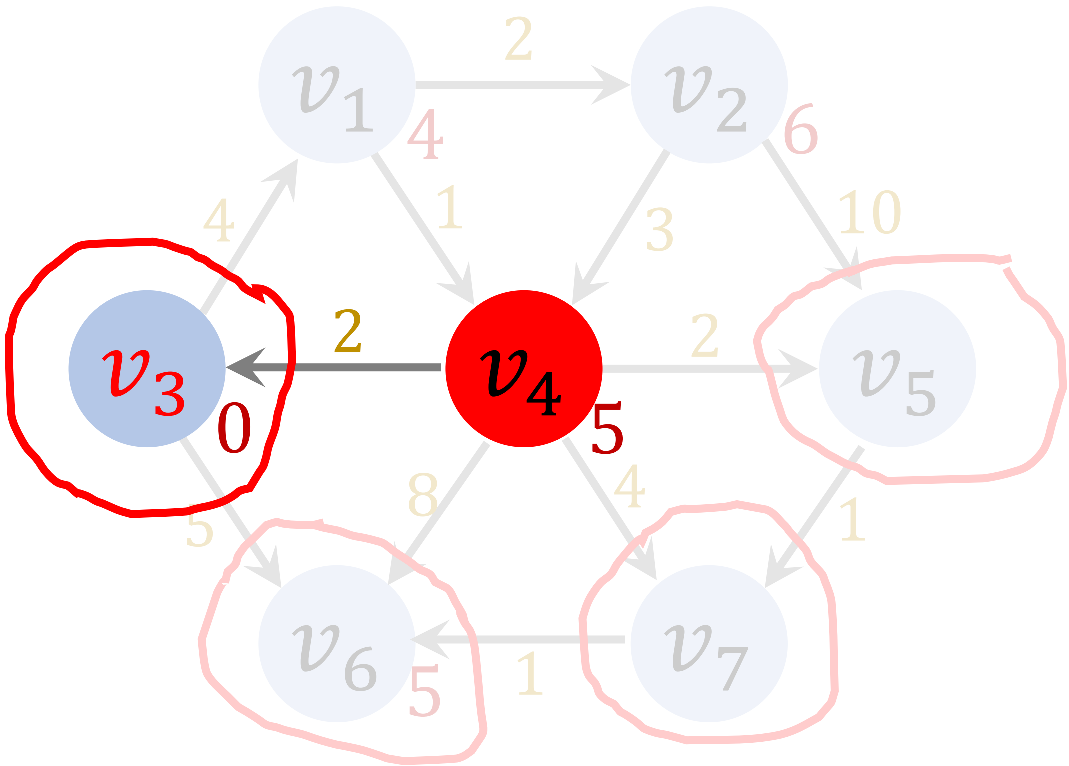
- Work on v_3.

Priority Queue:

vertex	dist
v_2	6

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 4(A)



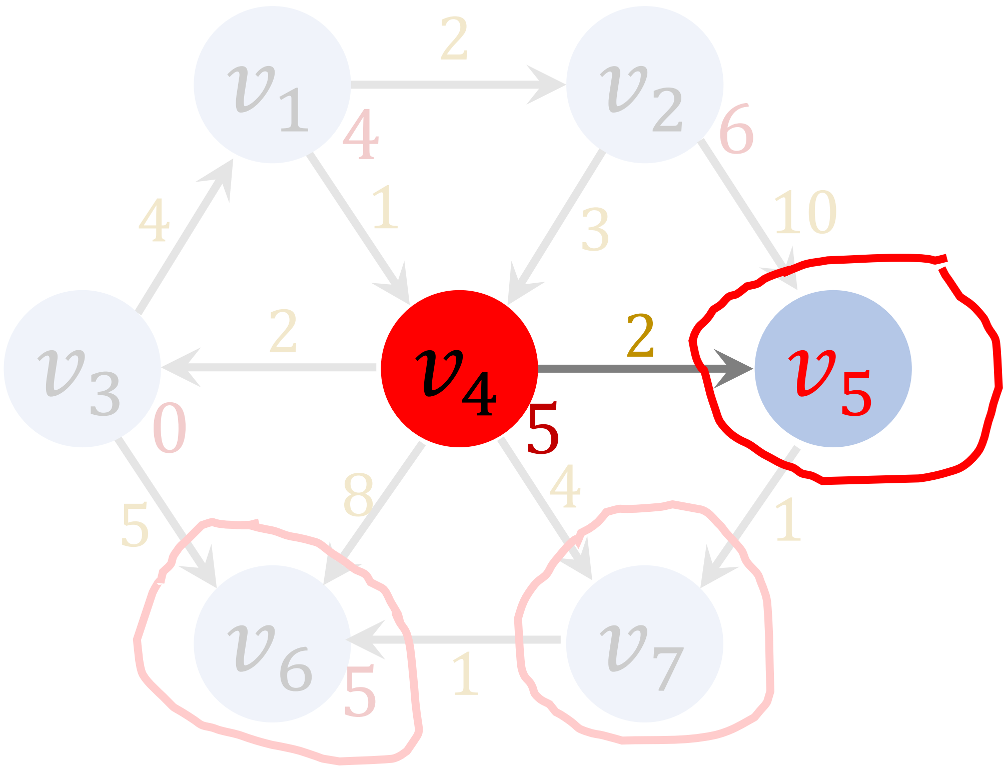
Priority Queue:

vertex	dist
v_2	6

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	∞	0
v_6	5	v_3
v_7	∞	0

- $d = \text{dist}[4] + 2 = 7.$
- dist[3] cannot get smaller.
- ➔ Do not update the table.

Iteration 4(B)



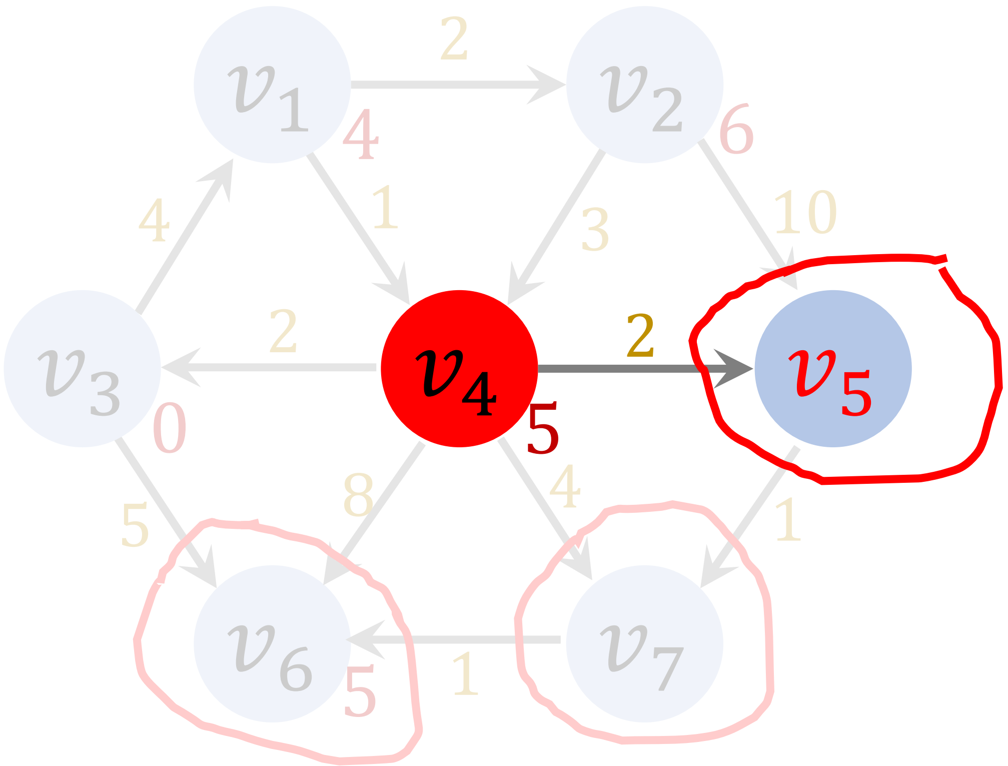
- Work on v_5.

Priority Queue:

vertex	dist
v_2	6

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	∞	0
v_6	**5**	v_3
v_7	∞	0

Iteration 4(B)



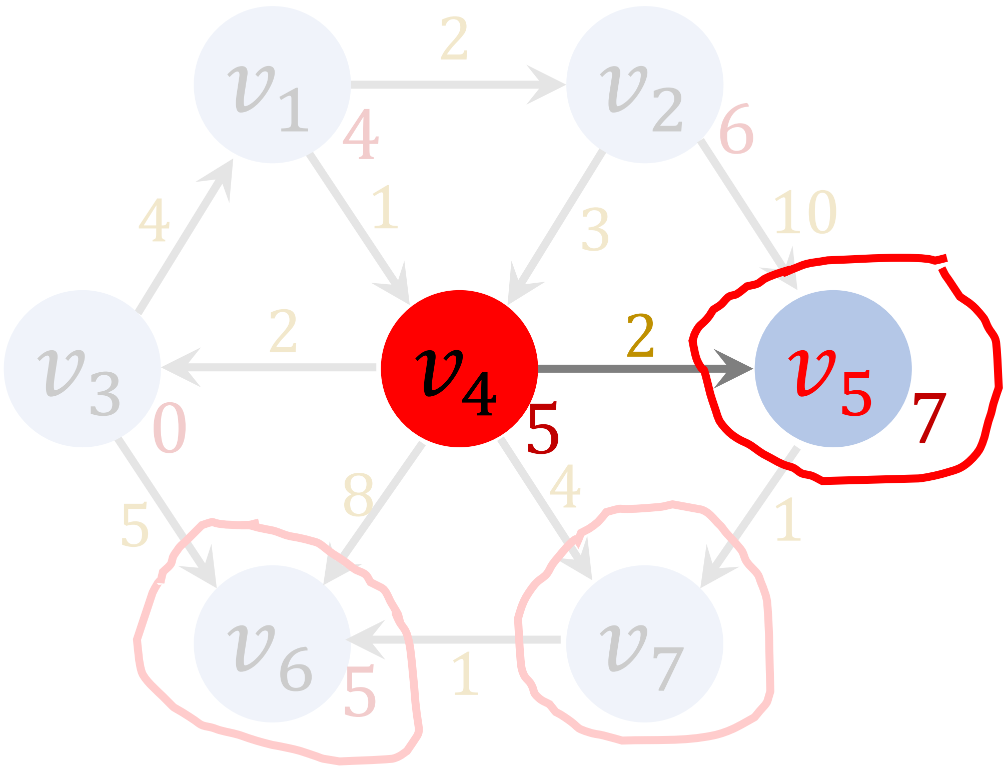
Priority Queue:

vertex	dist
v_2	6

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	∞	0
v_6	5	v_3
v_7	∞	0

- $d = \text{dist}[4] + 2 = 7.$
- Since $d < \infty$, update the table.

Iteration 4(B)



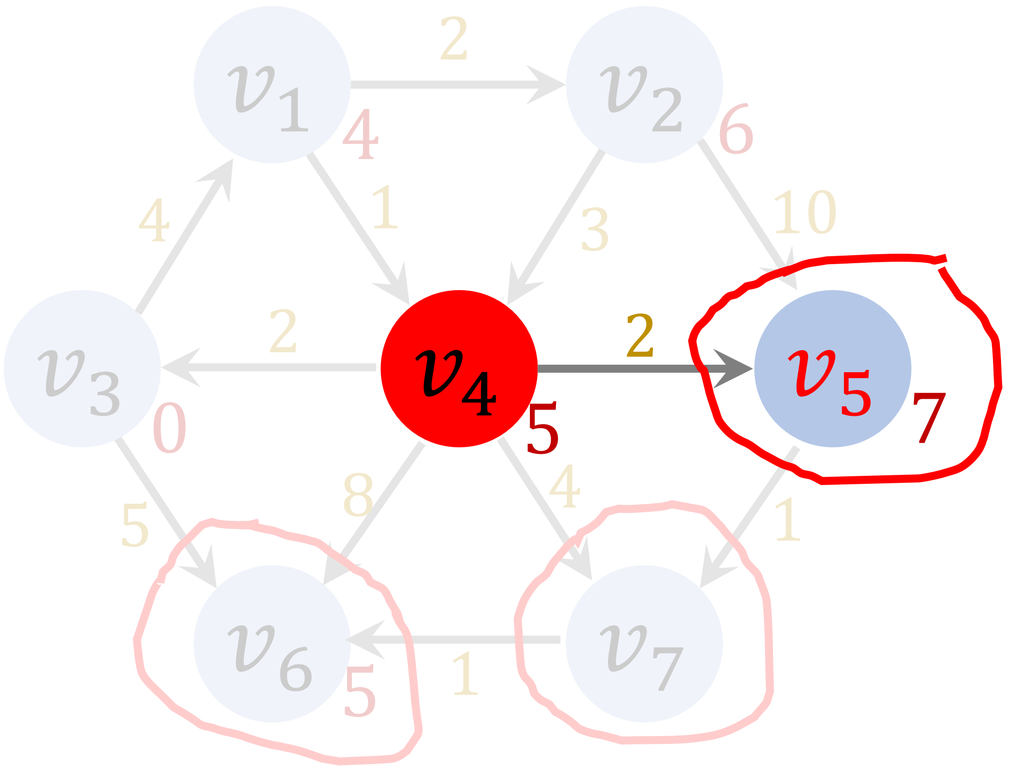
- dist[5] = 7.

Priority Queue:

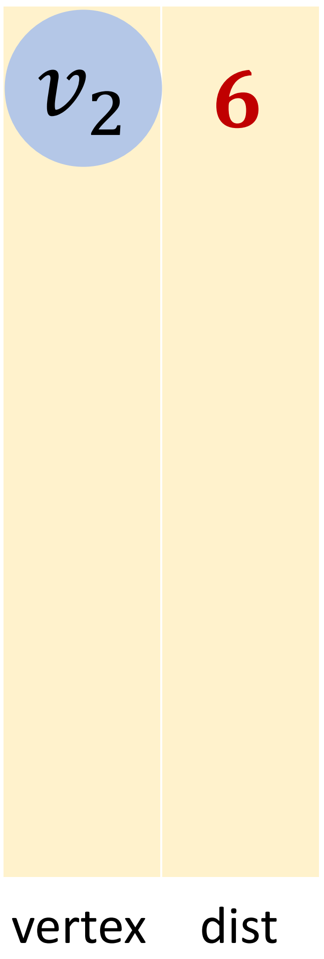
vertex	dist
v_2	6

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	∞	0

Iteration 4(B)



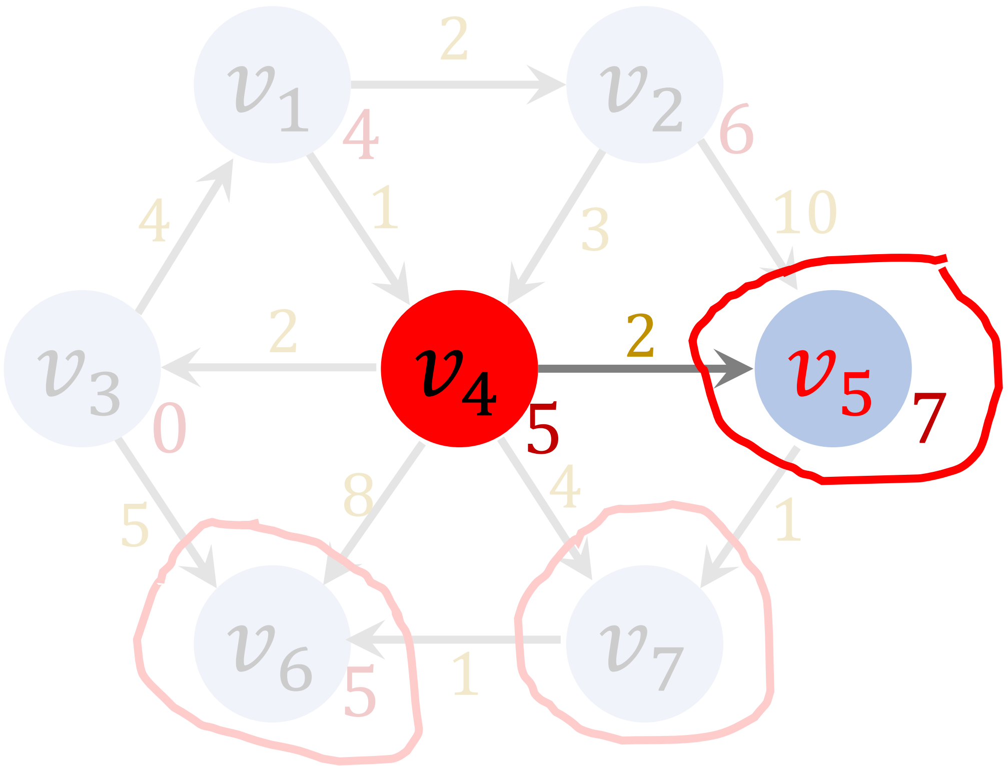
Priority Queue:



vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	∞	0

- $\text{dist}[5] = 7$.
- $\text{path}[5] = v_4$.

Iteration 4(B)



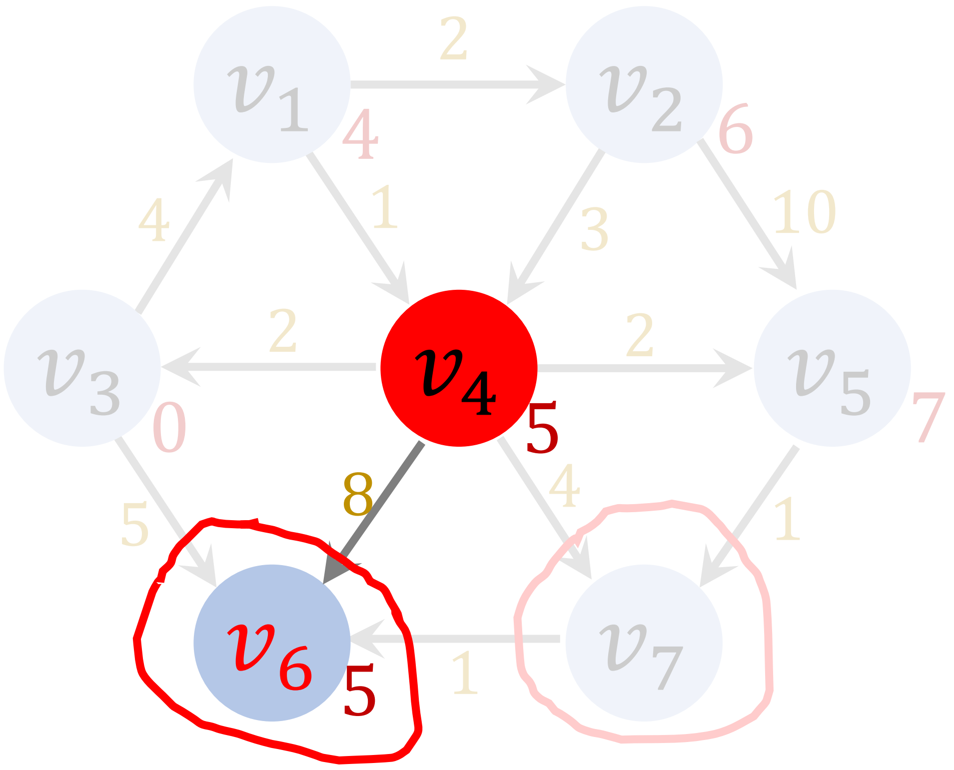
- enqueue(v_5, 7).

Priority Queue:

vertex	dist
v_2	6
v_5	7

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	∞	0

Iteration 4(C)



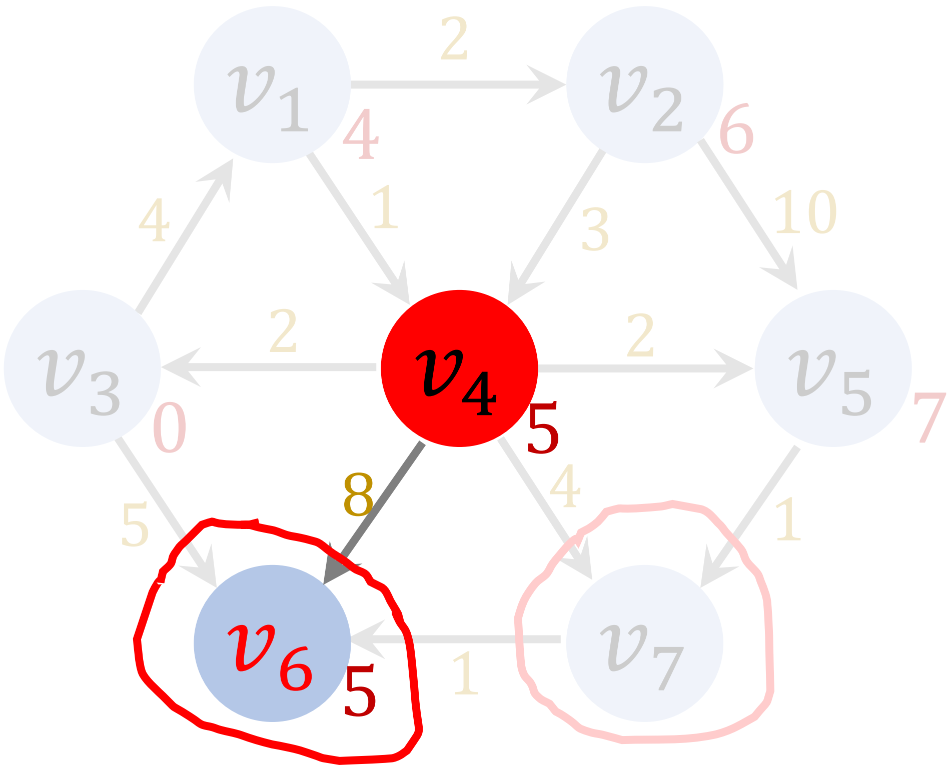
- Work on v_6.

Priority Queue:

vertex	dist
v_2	6
v_5	7

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	**7**	v_4
v_6	**5**	v_3
v_7	∞	0

Iteration 4(C)



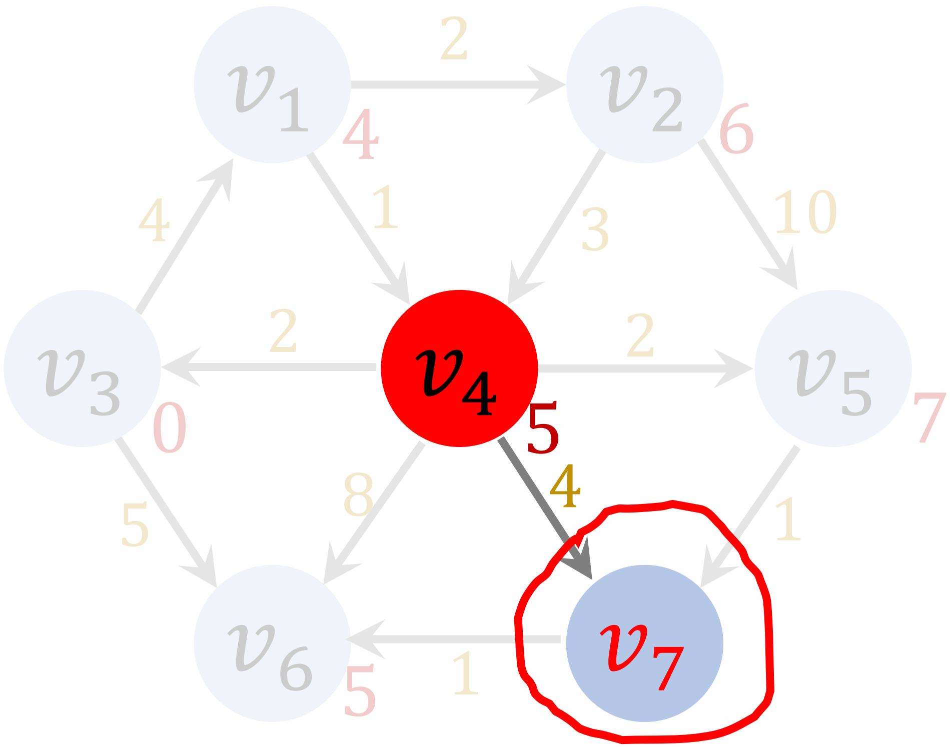
- $d = \text{dist}[4] + 8 = 13.$
- dist[6] cannot get smaller.
- ➔ Do not update the table.

Priority Queue:

vertex	dist
v_2	6
v_5	7

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	$\boxed{5}$	v_3
v_7	∞	0

Iteration 4(D)



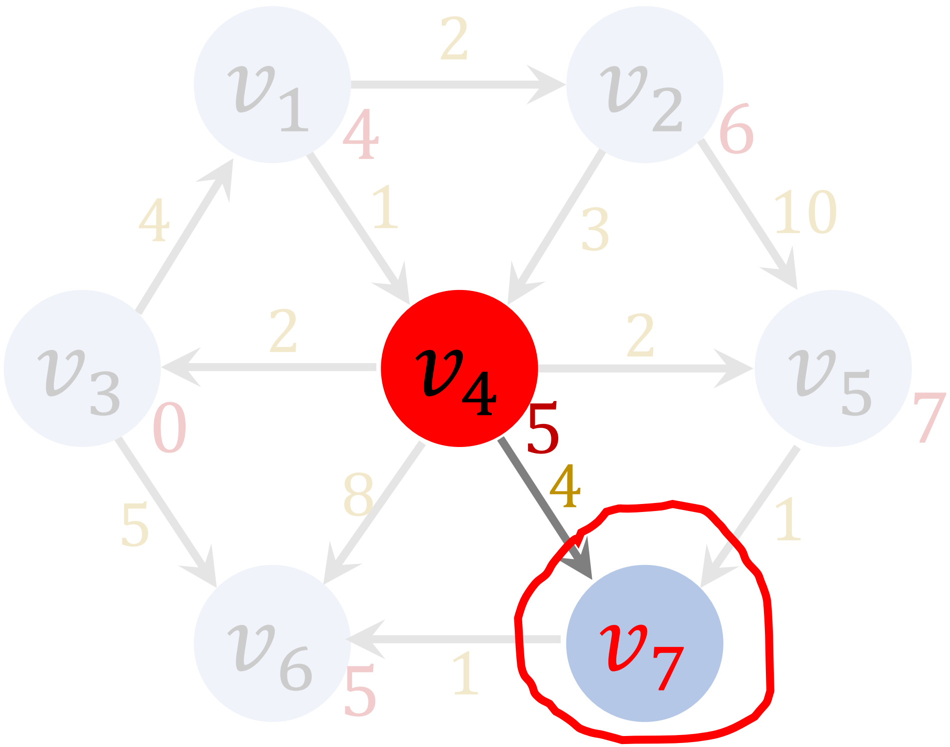
- Work on v_7.

Priority Queue:

vertex	dist
v_2	6
v_5	7

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	∞	0

Iteration 4(D)



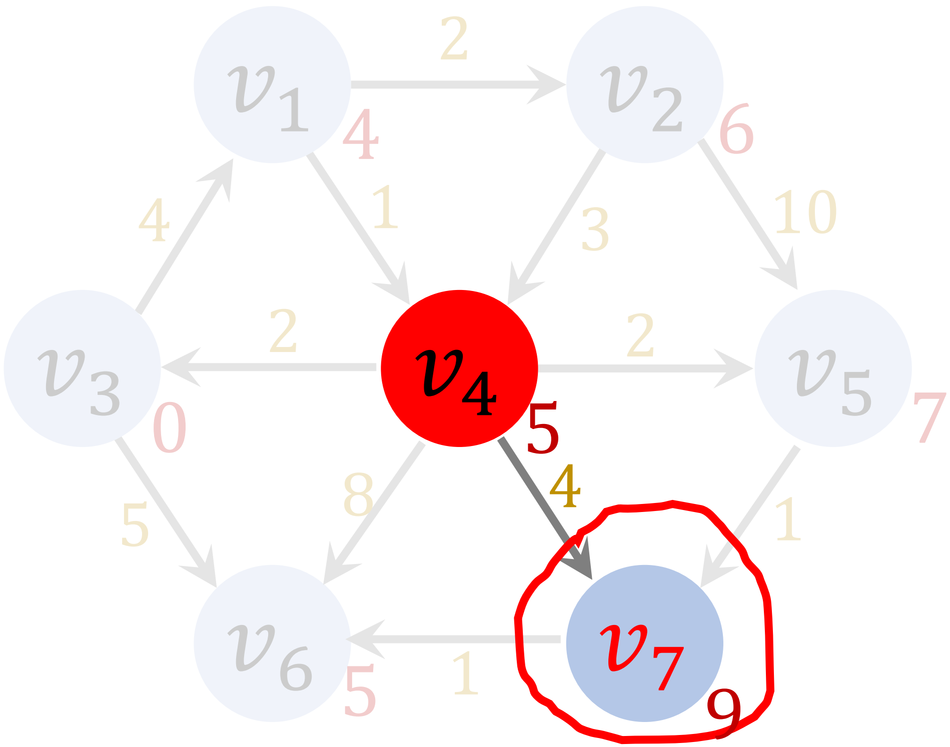
- $d = \text{dist}[4] + 4 = 9.$
- Since $d < \infty$, update the table.

Priority Queue:

vertex	dist
v_2	6
v_5	7

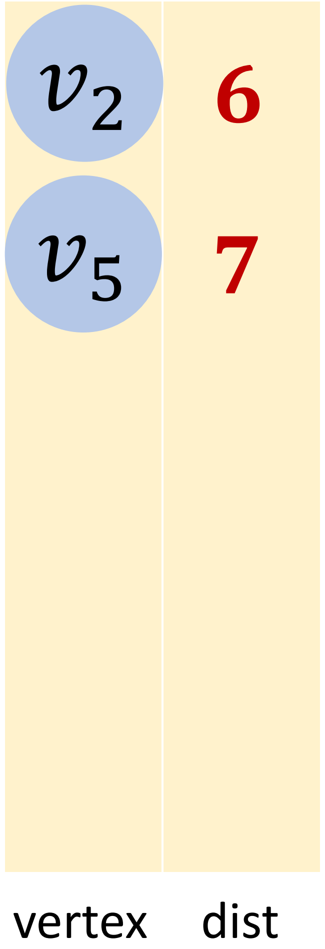
vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	∞	0

Iteration 4(D)



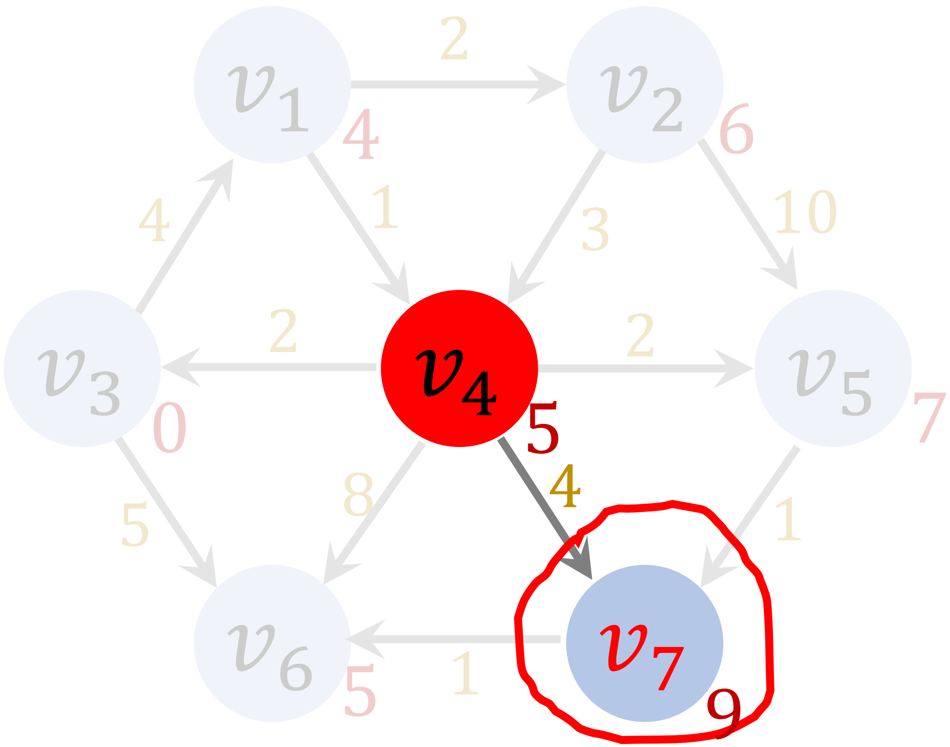
- dist[7] = 9.

Priority Queue:



vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	9	0

Iteration 4(D)



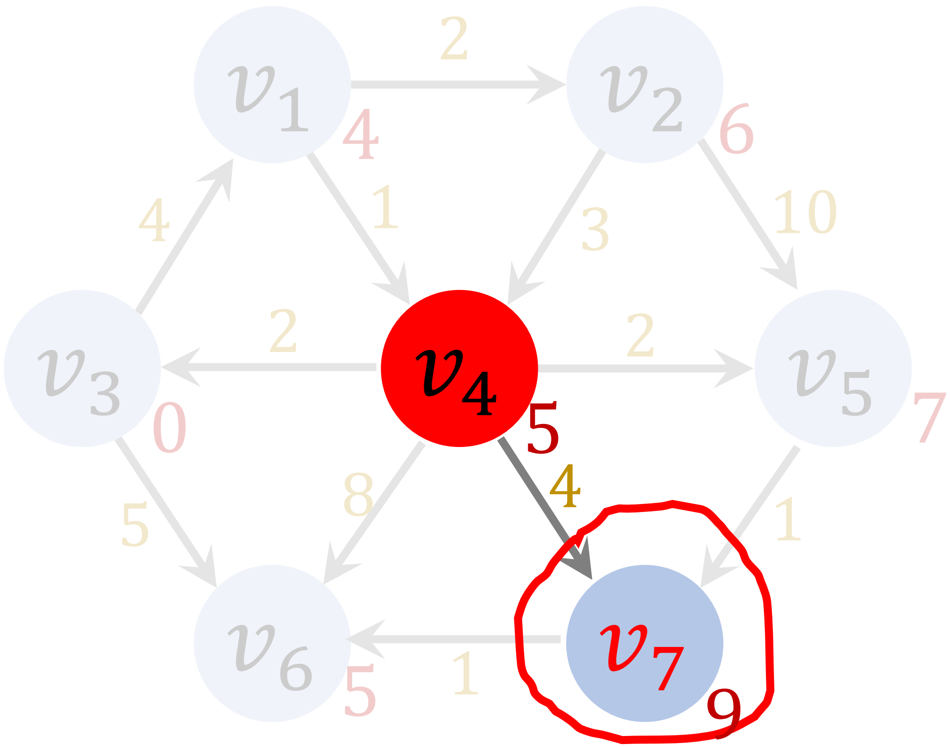
Priority Queue:

vertex	dist
v_2	**6**
v_5	**7**

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	**7**	v_4
v_6	**5**	v_3
v_7	**9**	v_4

- $\text{dist}[7] = 9$.
- $\text{path}[7] = v_4$.

Iteration 4(D)



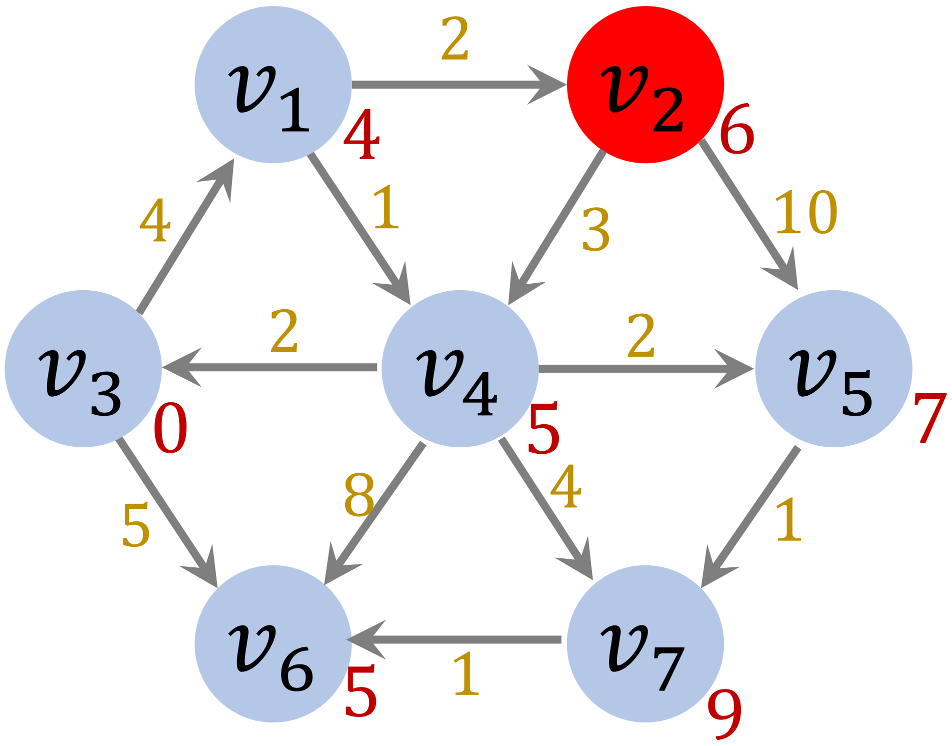
- enqueue(v_7, 9).

Priority Queue:

vertex	dist
v_2	6
v_5	7
v_7	9

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	9	v_4

Iteration 5



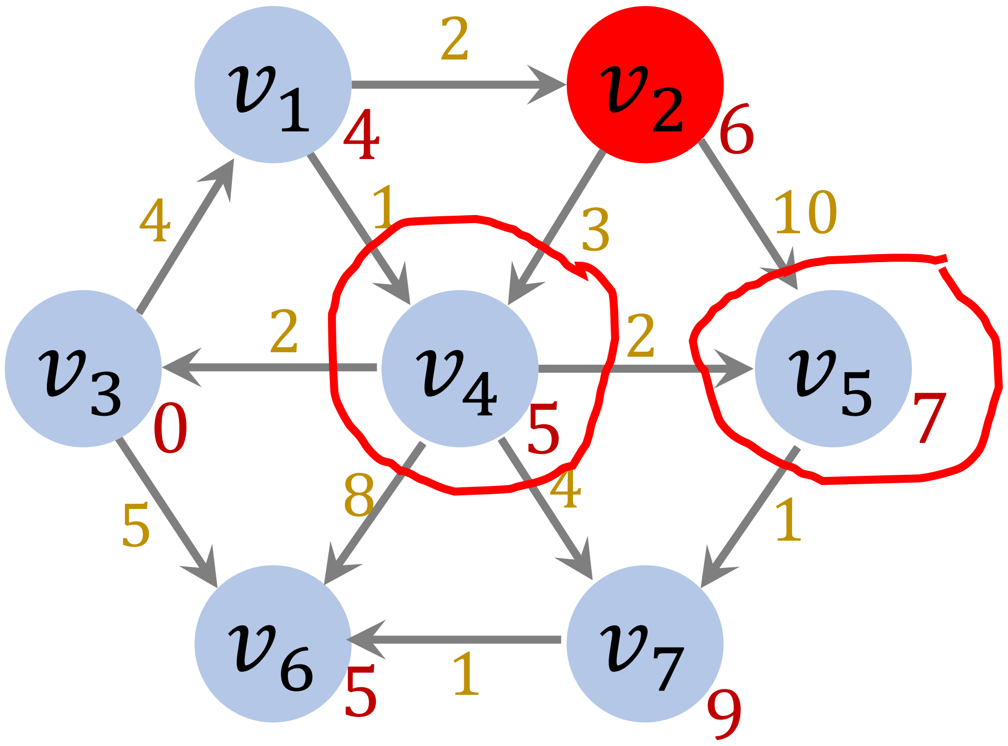
- $v_2 \leftarrow$ dequeue().

Priority Queue:

vertex	dist
v_2	6
v_5	7
v_7	9

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	**7**	v_4
v_6	**5**	v_3
v_7	**9**	v_4

Iteration 5



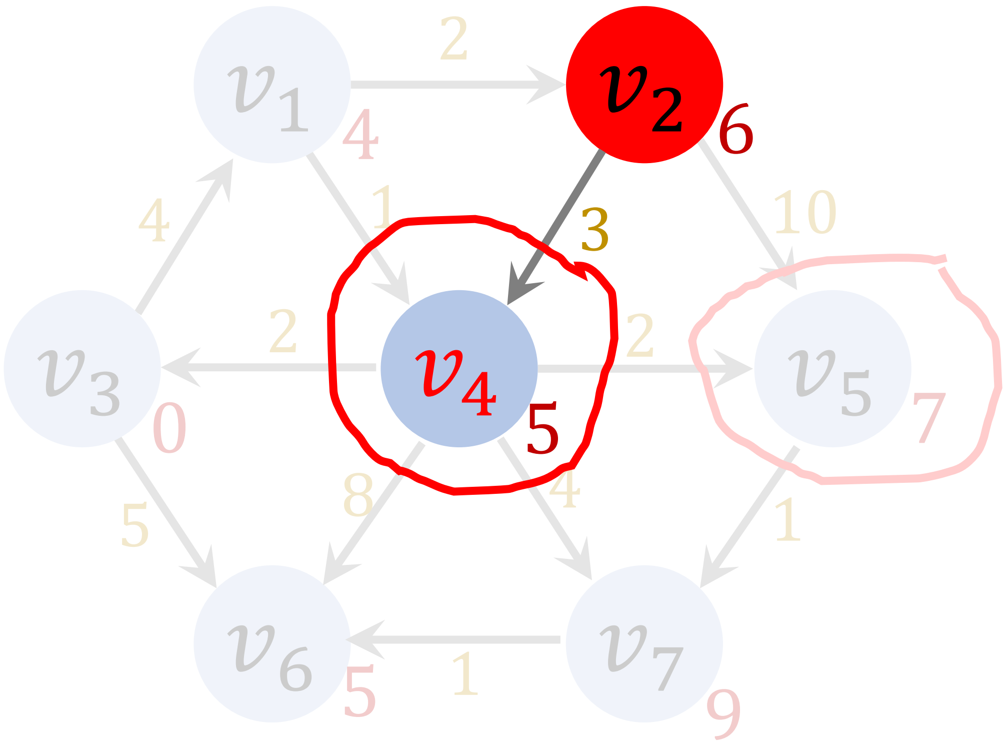
- $v_2 \leftarrow$ dequeue().
- Find adjacent vertices of v_4: v_4 and v_5.

Priority Queue:

vertex	dist
v_5	7
v_7	9

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	9	v_4

Iteration 5(A)



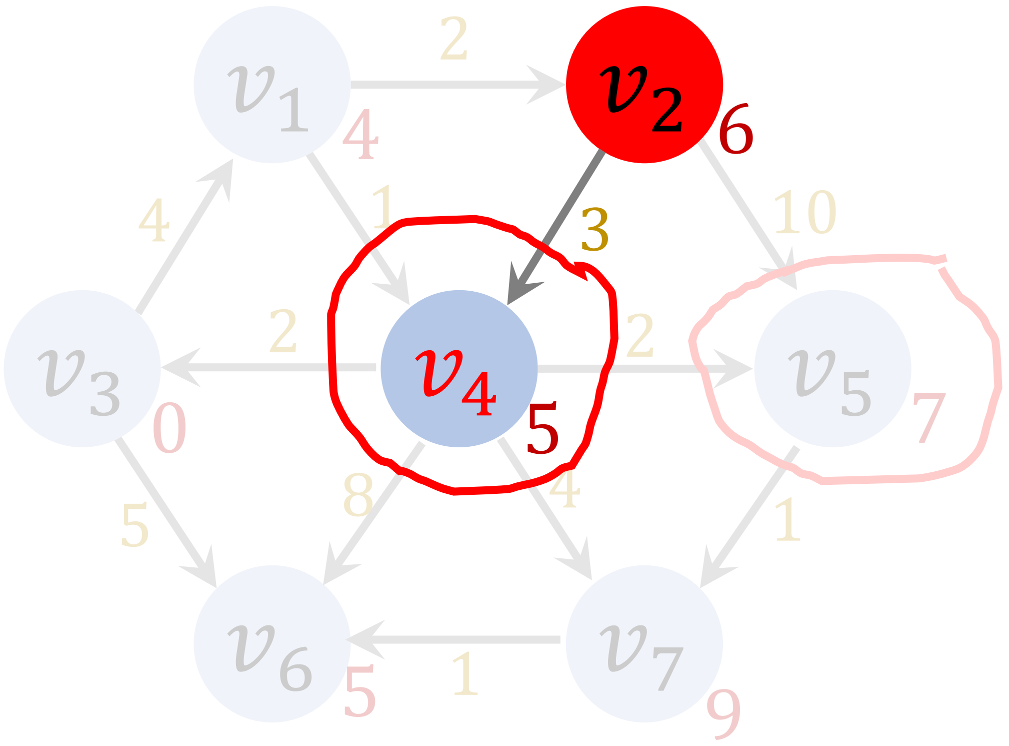
Priority Queue:

vertex	dist
v_5	7
v_7	9

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	9	v_4

- Work on v_4.

Iteration 5(A)



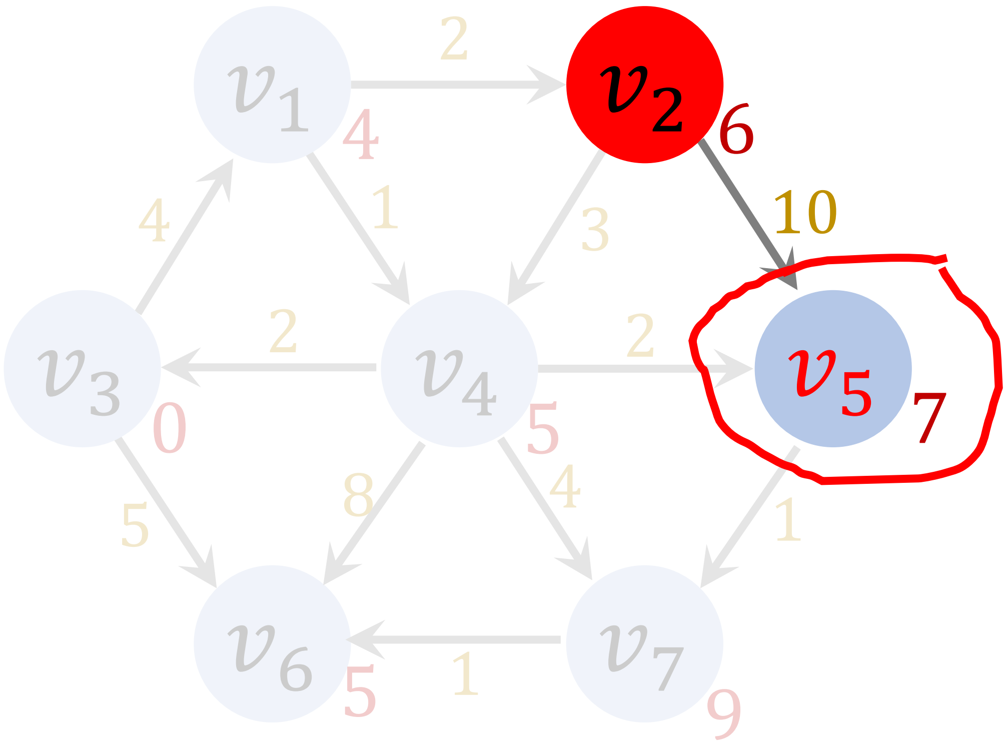
Priority Queue:

vertex	dist
v_5	7
v_7	9

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	9	v_4

- $d = \text{dist}[2] + 3 = 9.$
- $\text{dist}[4]$ cannot get smaller.
- ➔ Do not update the table.

Iteration 5(B)



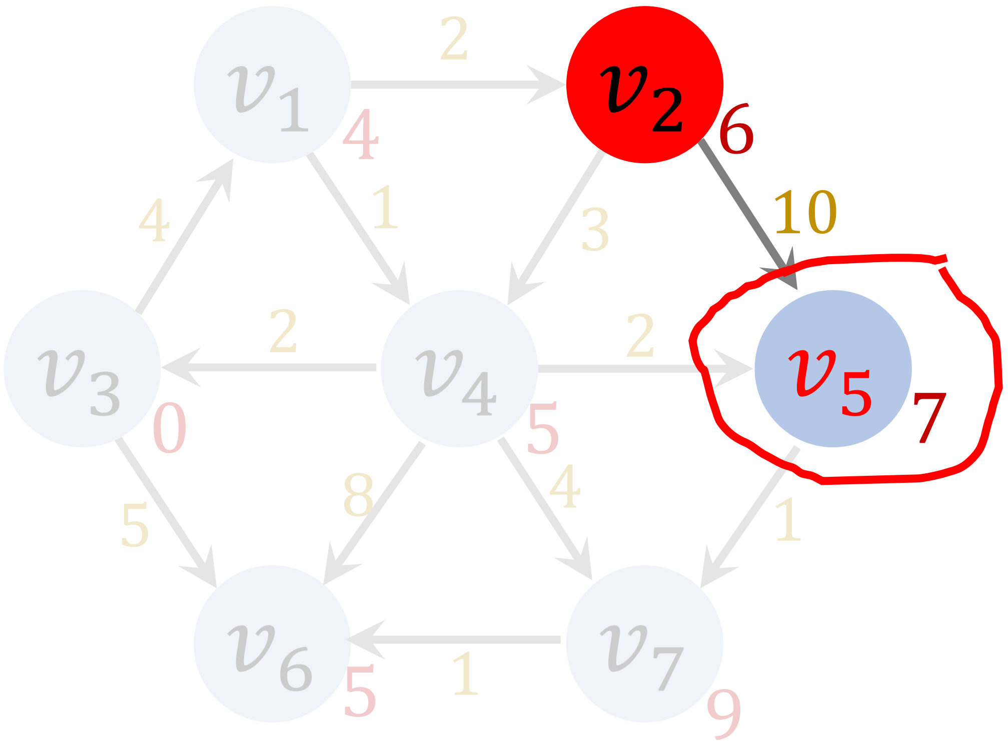
- Work on v_5.

Priority Queue:

vertex	dist
v_5	7
v_7	9

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	9	v_4

Iteration 5(B)



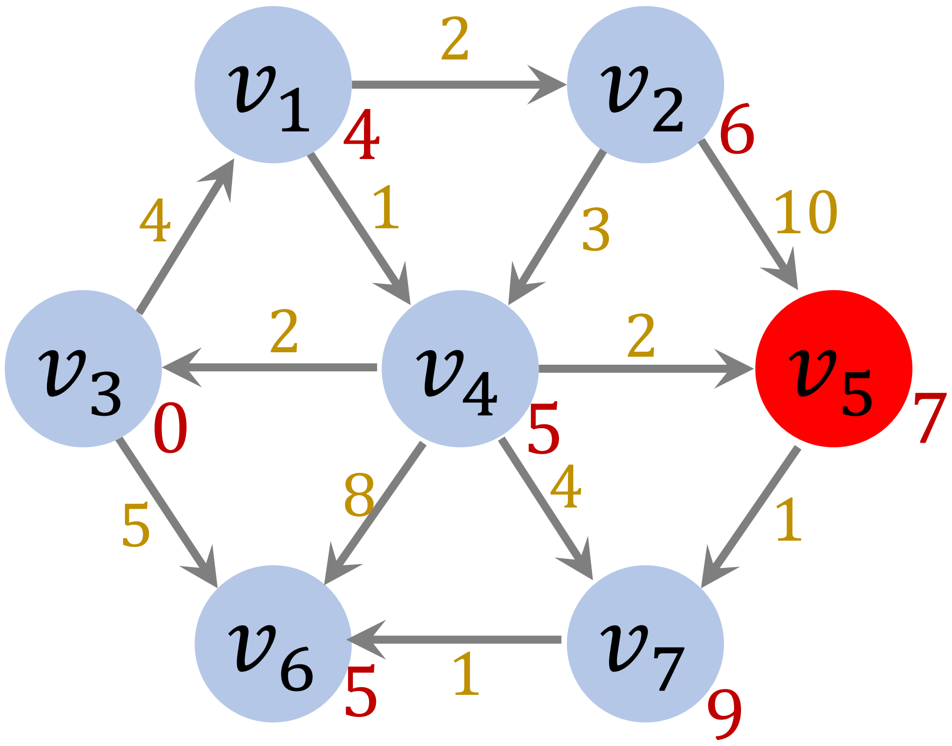
- $d = \text{dist}[2] + 10 = 16.$
- dist[5] cannot get smaller.
- ➔ Do not update the table.

Priority Queue:

vertex	dist
v_5	7
v_7	9

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	9	v_4

Iteration 6



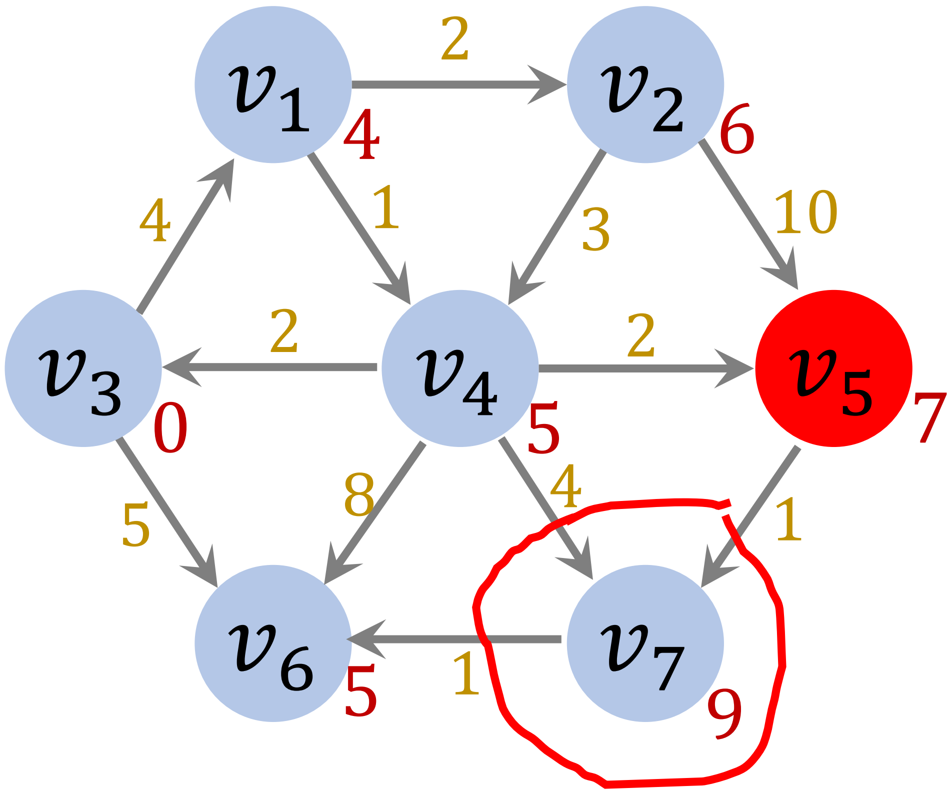
- $v_5 \leftarrow$ dequeue().

Priority Queue:

vertex	dist
v_5	7
v_7	9

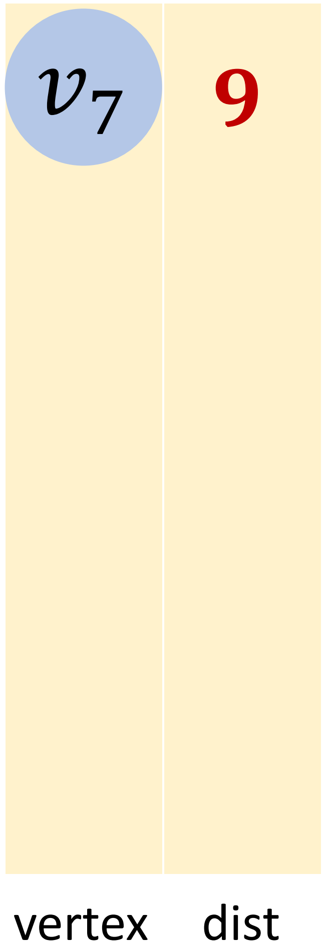
vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	**7**	v_4
v_6	**5**	v_3
v_7	**9**	v_4

Iteration 6



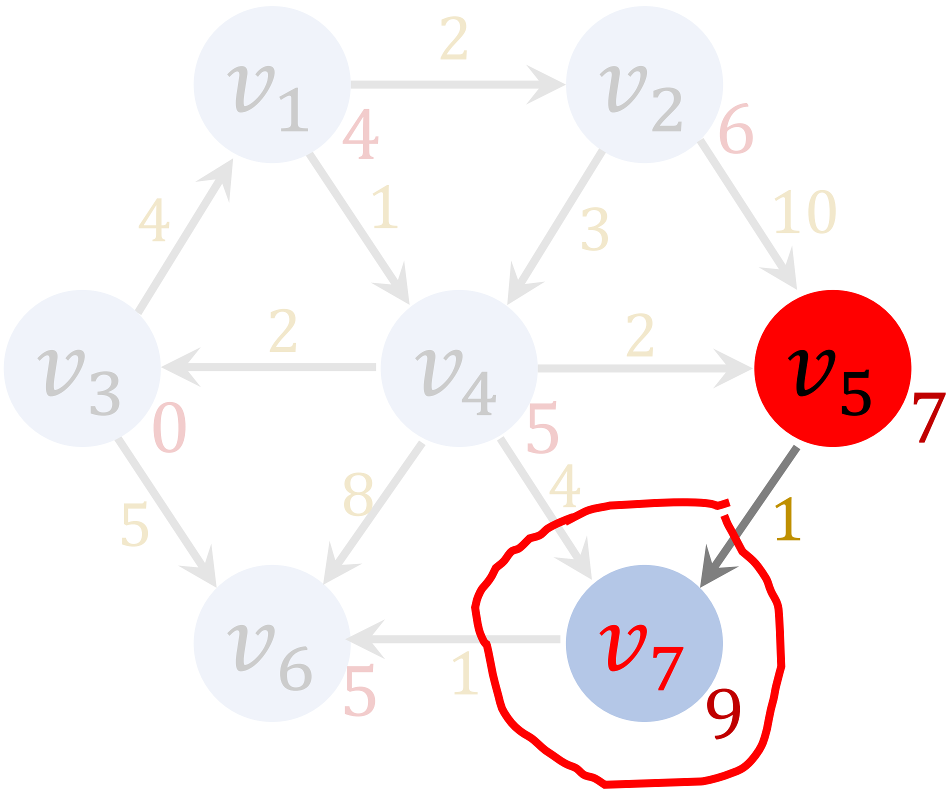
- $v_5 \leftarrow$ dequeue().
- Find adjacent vertices of v_5: v_7.

Priority Queue:



vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	9	v_4

Iteration 6(A)



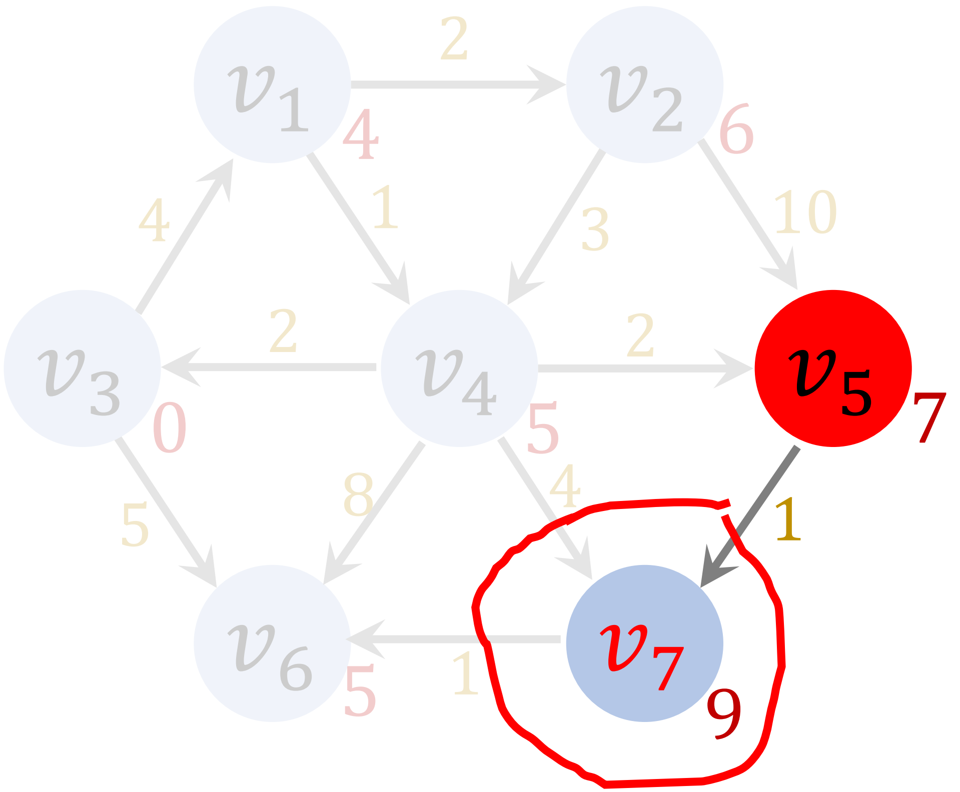
- Work on v_7.

Priority Queue:

vertex	dist
v_7	9

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	**7**	v_4
v_6	**5**	v_3
v_7	**9**	v_4

Iteration 6(A)



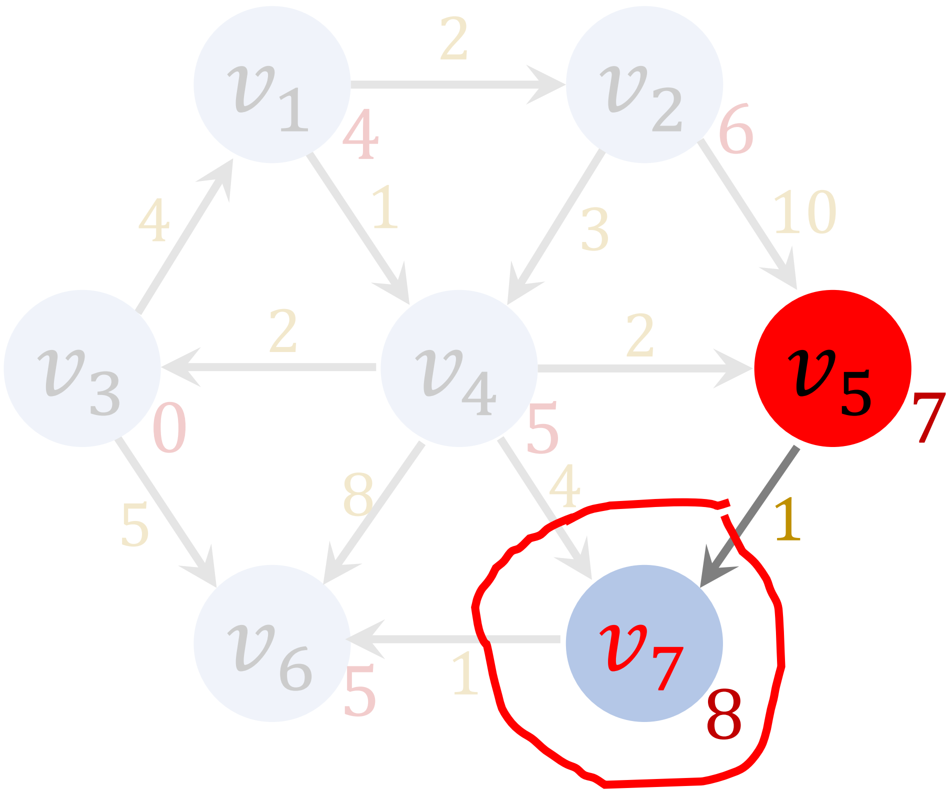
- $d = \text{dist}[5] + 1 = 8$.
- Since $d < 9$, update the table.

Priority Queue:

vertex	dist
v_7	9

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	**7**	v_4
v_6	**5**	v_3
v_7	**9**	v_4

Iteration 6(A)



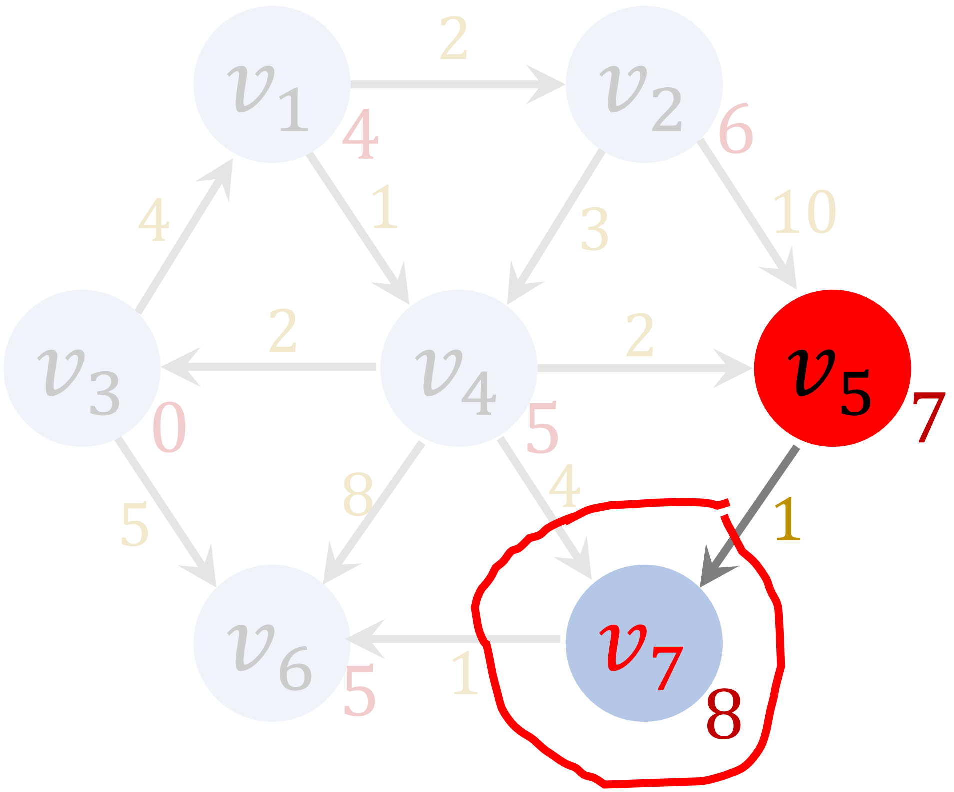
- dist[7] = 8.

Priority Queue:

vertex	dist
v_7	9

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	8	v_4

Iteration 6(A)



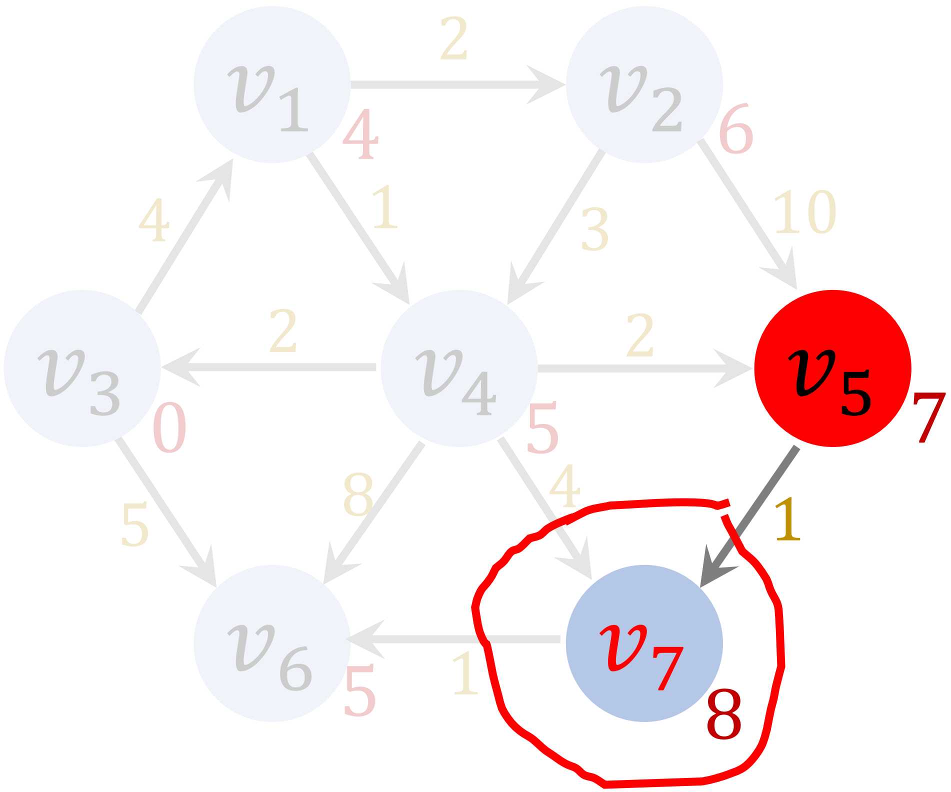
- dist[7] = 8.
- path[7] = v_5.

Priority Queue:

vertex	dist
v_7	**9**

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	**7**	v_4
v_6	**5**	v_3
v_7	**8**	v_5

Iteration 6(A)



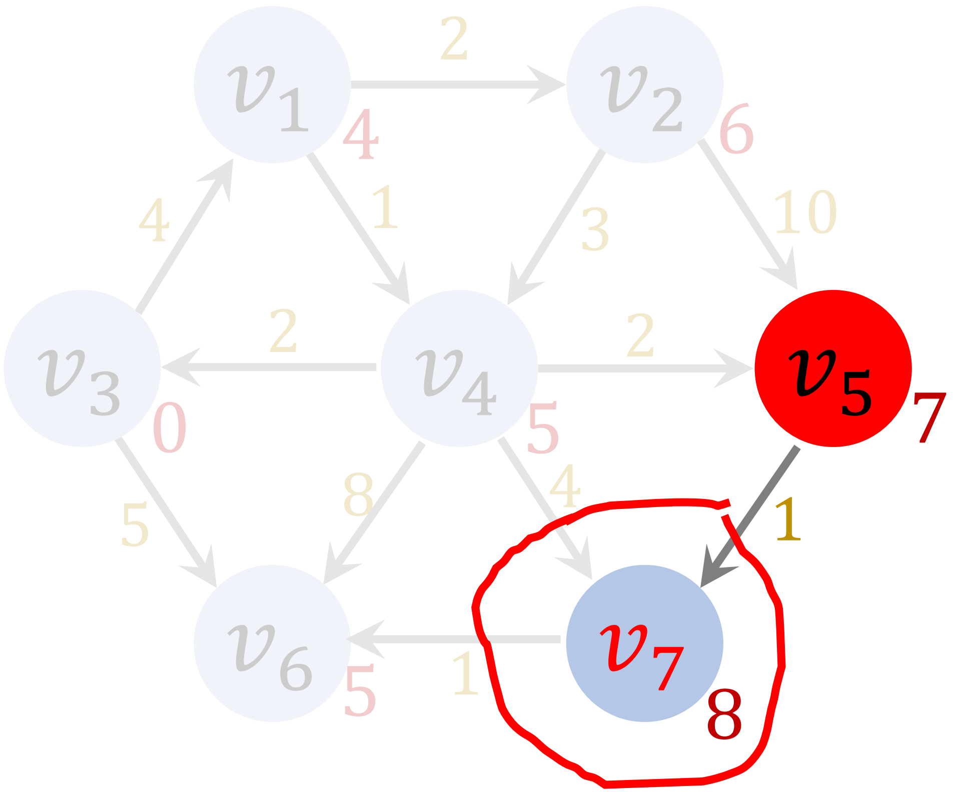
- enqueue(v_7, 8).

Priority Queue:

vertex	dist
v_7	9

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	8	v_5

Iteration 6(A)



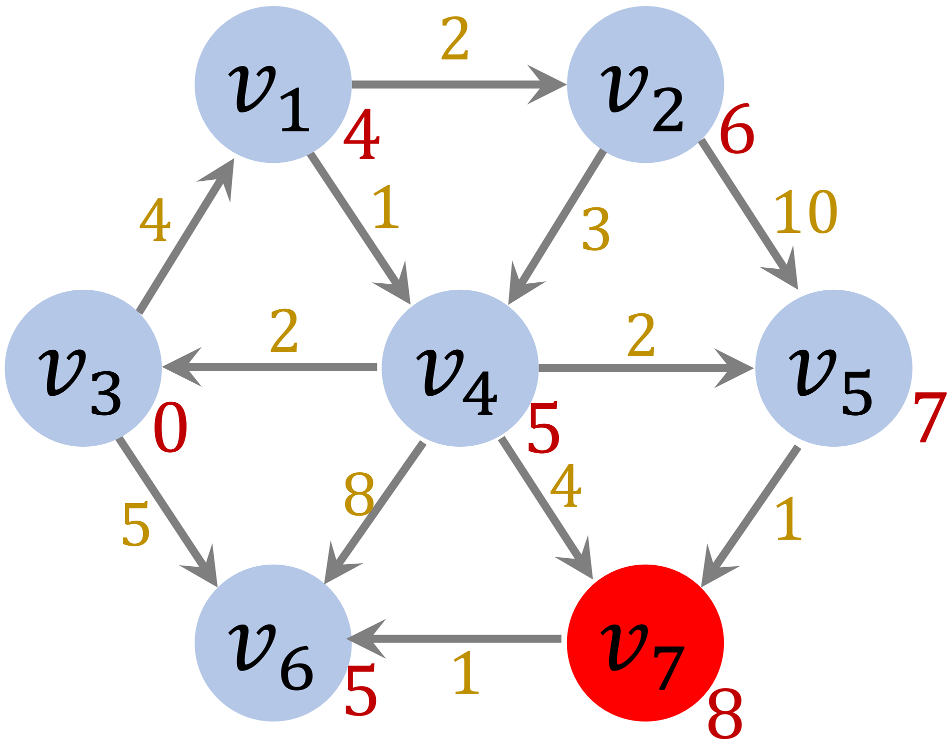
- enqueue(v_7, 8).

Priority Queue:

vertex	dist
v_7	8

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	8	v_5

Iteration 7



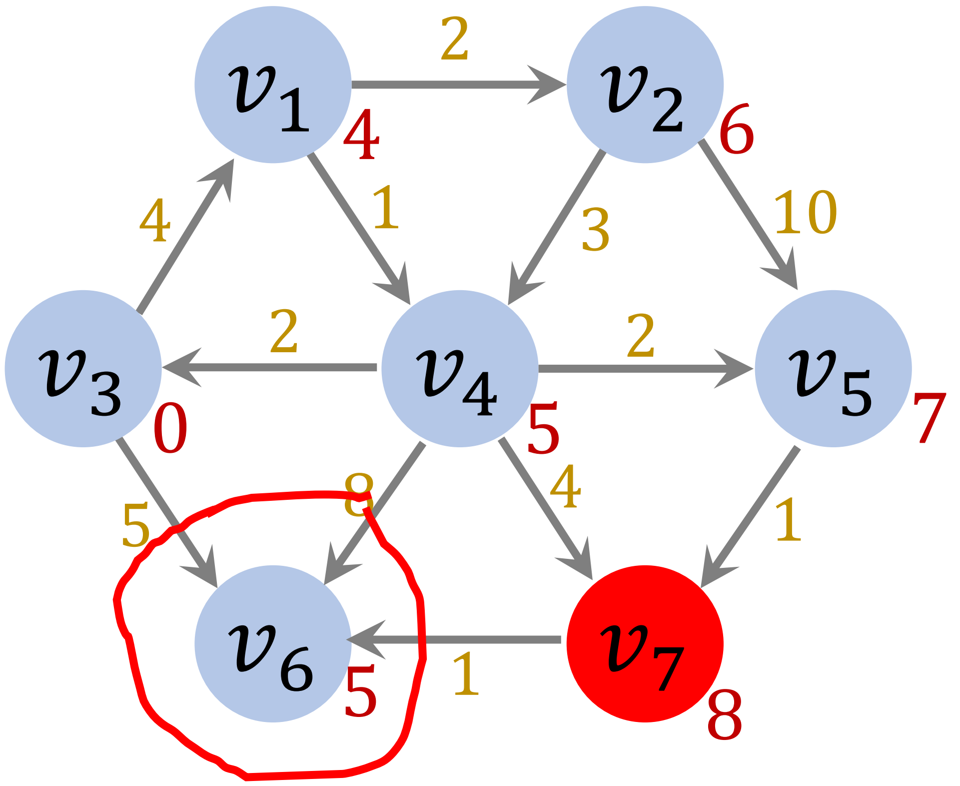
- $v_7 \leftarrow$ dequeue().

Priority Queue:

vertex	dist
v_7	8

vertex	dist	path
v_1	4	v_3
v_2	6	v_1
v_3	0	0
v_4	5	v_1
v_5	7	v_4
v_6	5	v_3
v_7	8	v_5

Iteration 7



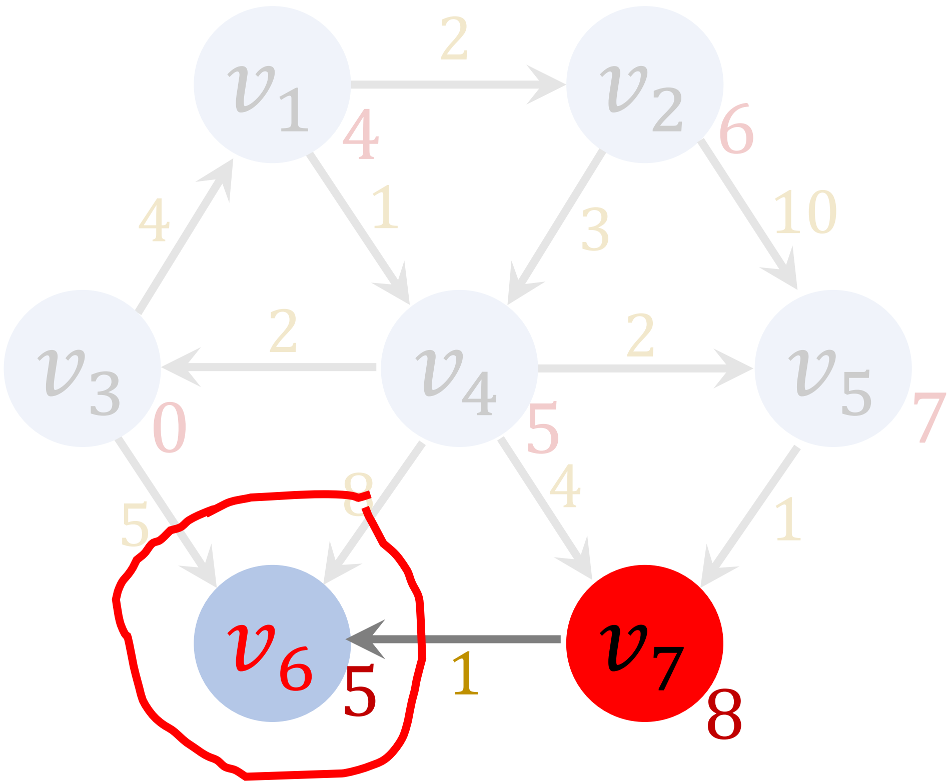
- $v_7 \leftarrow$ dequeue().
- Find adjacent vertices of v_7: v_6.

Priority Queue:

vertex	dist

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	**7**	v_4
v_6	**5**	v_3
v_7	**8**	v_5

Iteration 7(A)



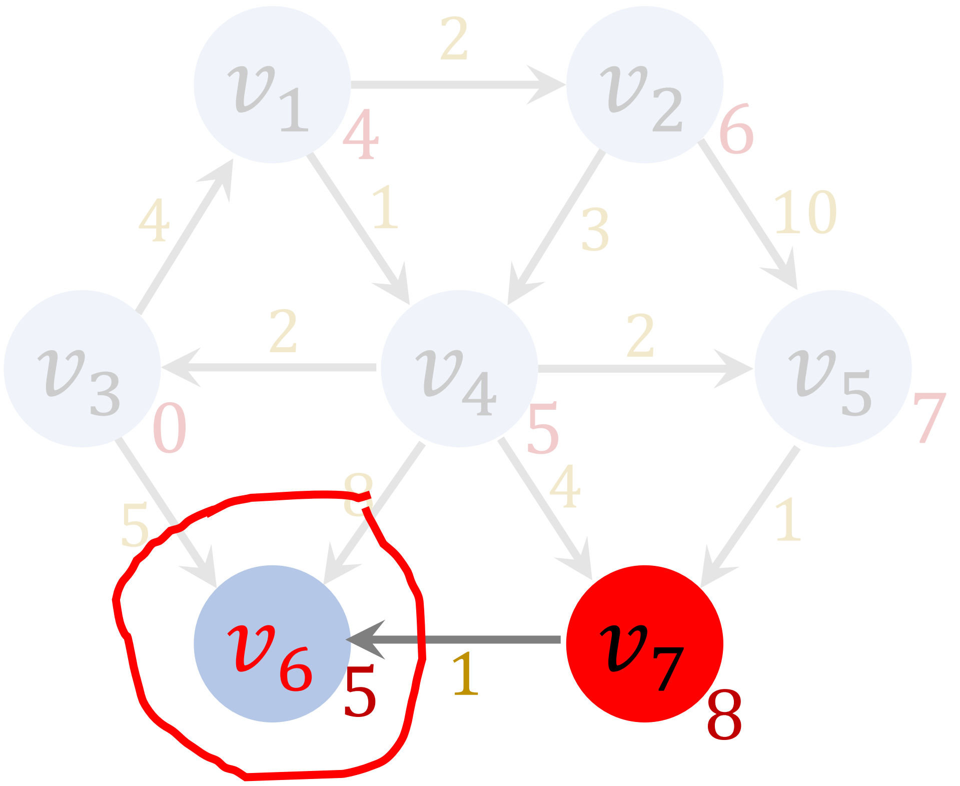
- Work on v_6.

Priority Queue:

vertex	dist

vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	**7**	v_4
v_6	**5**	v_3
v_7	**8**	v_5

Iteration 7(A)



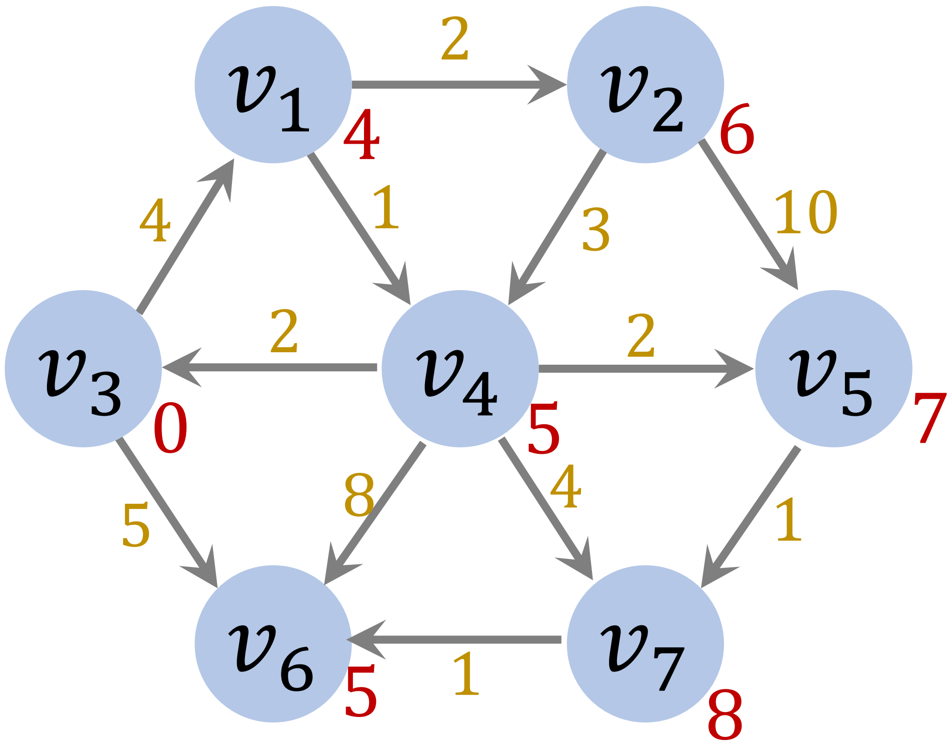
- $d = \text{dist}[7] + 1 = 9.$
- dist[6] cannot get smaller.
- ➔ Do not update the table.

Priority Queue:

vertex	dist

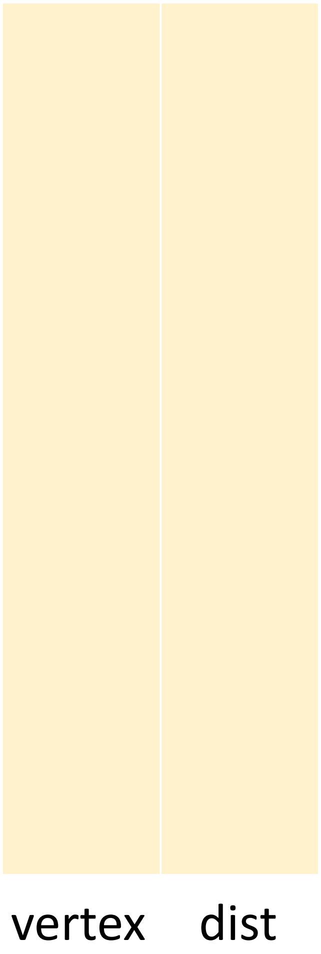
vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	**7**	v_4
v_6	**5**	v_3
v_7	**8**	v_5

End of Procedure



- The queue is empty.
- ➔ End of procedure.

Priority Queue:



vertex	dist	path
v_1	**4**	v_3
v_2	**6**	v_1
v_3	**0**	0
v_4	**5**	v_1
v_5	**7**	v_4
v_6	**5**	v_3
v_7	**8**	v_5

Pseudo Code

Inputs: vertices $\mathcal{V}$, edges $\mathcal{E}$, and the source vertex s.

1. Initialize an empty priority queue.

Pseudo Code

Inputs: vertices $\mathcal{V}$, edges $\mathcal{E}$, and the source vertex s.

1. Initialize an empty priority queue.
2. For each vertex $v \in \mathcal{V}$:
   a. Set $\text{dist}[v] = \infty$.
   b. Set $\text{path}[v] = 0$.

vertex	dist	path
v_1	∞	0
v_2	∞	0
$\vdots$	$\vdots$	$\vdots$
v_n	∞	0

Pseudo Code

Inputs: vertices $\mathcal{V}$, edges $\mathcal{E}$, and the source vertex s.

1. Initialize an empty priority queue.
2. For each vertex $v \in \mathcal{V}$:
   a. Set $\text{dist}[v] = \infty$.
   b. Set $\text{path}[v] = 0$.
3. Set $\text{dist}[s] = 0$.
4. $\text{enqueue}(s, 0)$.

Pseudo Code (Cont.)

5. While the priority queue is not empty:
   a. $v \leftarrow \text{dequeue}()$.
   b. $\mathcal{S} \leftarrow \{ u \mid e_{vu} \in \mathcal{E} \}$.
   c. For each $u \in \mathcal{S}$:
      i. $\mathrm{d}_{\text{new}} = \text{dist}[v] + e_{vu}$.
      ii. If $\mathrm{d}_{\text{new}} < \text{dist}[u]$:
         - Set $\text{dist}[u] = \mathrm{d}_{\text{new}}$ and $\text{path}[u] = v$.
         - $\text{enqueue}(u, \mathrm{d}_{\text{new}})$.

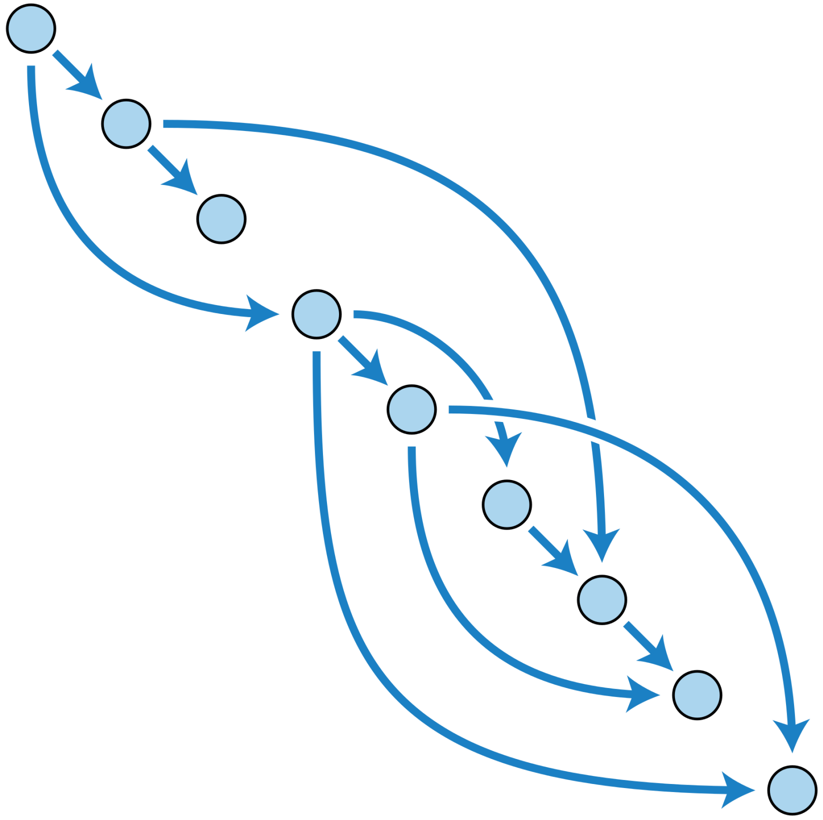
Outputs: $\text{dist}[v]$ and $\text{path}[v]$, for all $v \in \mathcal{V}$.

Time Complexity

Time Complexity

- Assume all the weights are nonnegative; otherwise, Dijkstra's algorithm does not work.
- Totally $O(|\mathcal{V}| + |\mathcal{E}|)$ enqueue and dequeue operations.
- Enqueue and dequeue operations both have $O(\log|\mathcal{V}|)$ time complexity.
- Thus, the overall time complexity is $O\big((|\mathcal{V}| + |\mathcal{E}|) \cdot \log|\mathcal{V}|\big)$.

Directed Acyclic Graph (DAG)

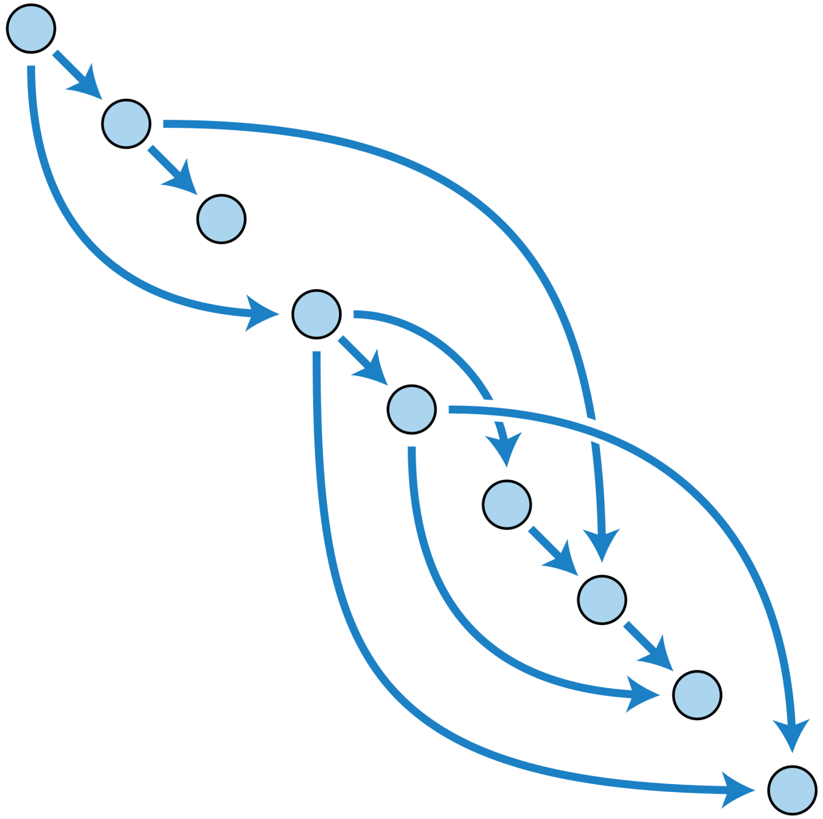


This is a DAG

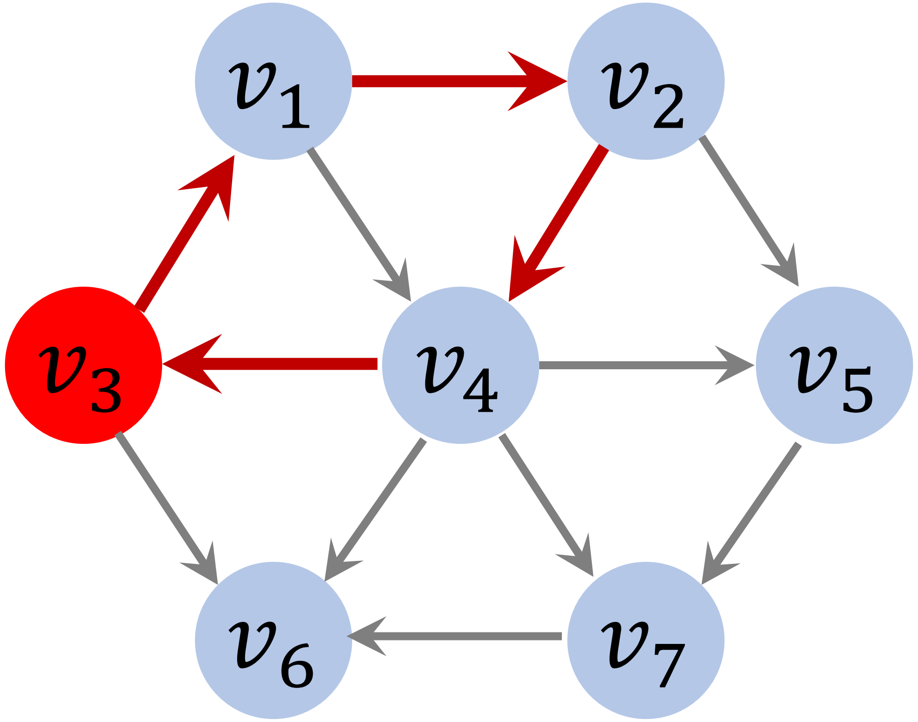
Definition of DAG

- DAG is a directed graph with no directed cycles.
- There is no way to start at any vertex v and follow a path that eventually loops back to v again.

Directed Acyclic Graph (DAG)



This is a DAG



This is not a DAG

Directed Acyclic Graph (DAG)

- If the graph is a DAG, we can use queue instead of priority queue.
- Enqueue and dequeue for standard queue cost only $O(1)$ time.
- The time complexity is $O(|\mathcal{V}| + |\mathcal{E}|)$. (The same as unweighted graph.)

Thank You!