Bellman-Ford Algorithm

William Fiset

BF algorithm overview

In graph theory, the **Bellman-Ford (BF)** algorithm is a **Single Source Shortest Path (SSSP)** algorithm. This means it can find the shortest path from one node to any other node.

However, BF is not ideal for most SSSP problems because it has a time complexity of **O(EV)**. It is better to use Dijkstra's algorithm which is much faster. It is on the order of **0((E+V)log(V))** when using a binary heap priority queue.

BF algorithm overview

However, Dijkstra's algorithm can fail when the graph has negative edge weights. This is when BF becomes really handy because it can be used to detect **negative cycles** and **determine where they occur**.

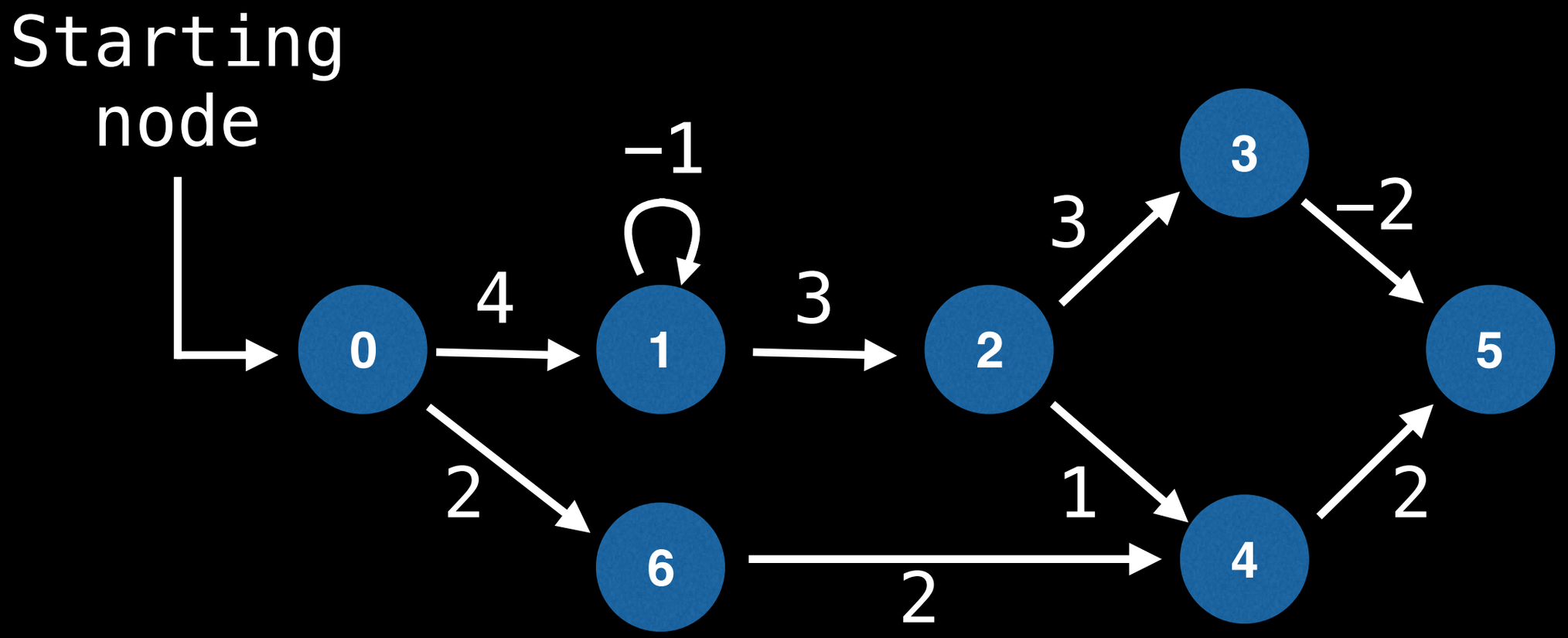
Finding negative cycles can be useful in many types of applications. One particularly neat application arises in finance when performing an **arbitrage** between two or more markets.

Negative Cycles

Negative cycles can manifest themselves in many ways…

Negative Cycles

Negative cycles can manifest themselves in many ways…



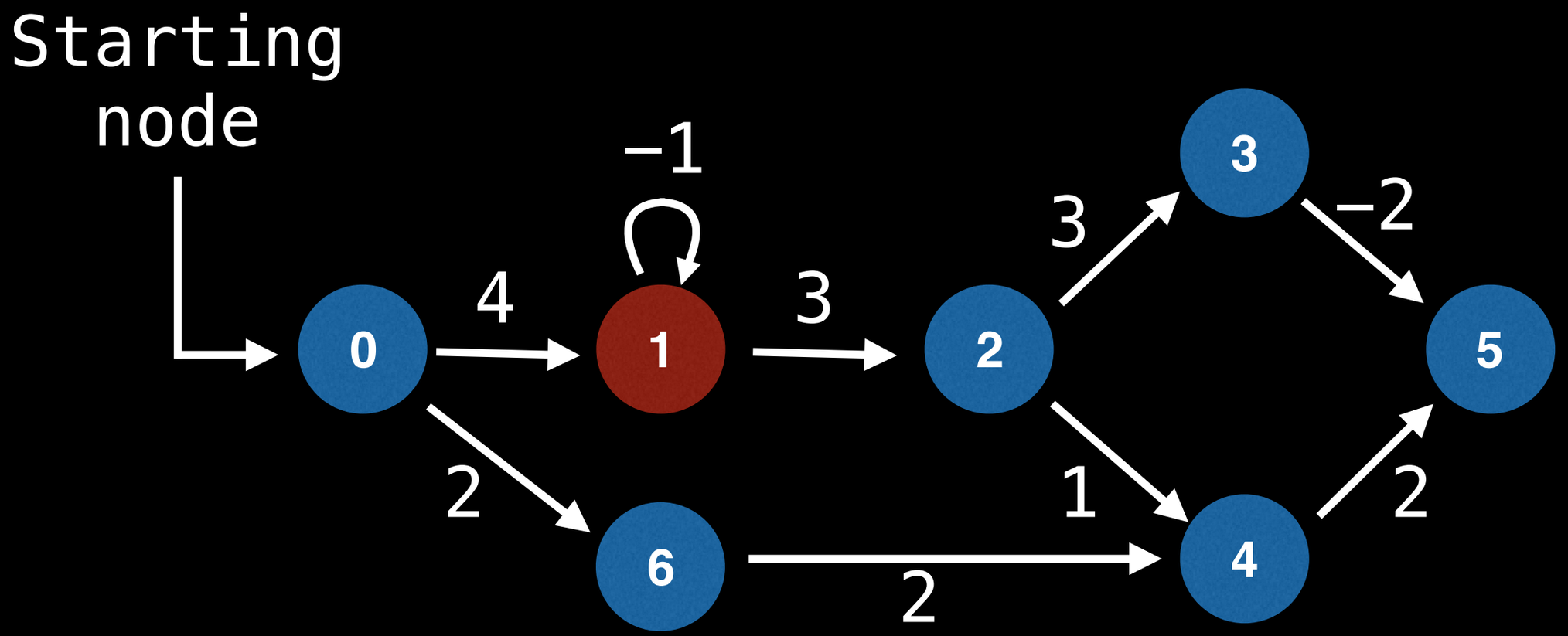
Unaffected node

Directly in negative cycle

Reachable by negative cycle

Negative Cycles

Negative cycles can manifest themselves in many ways…



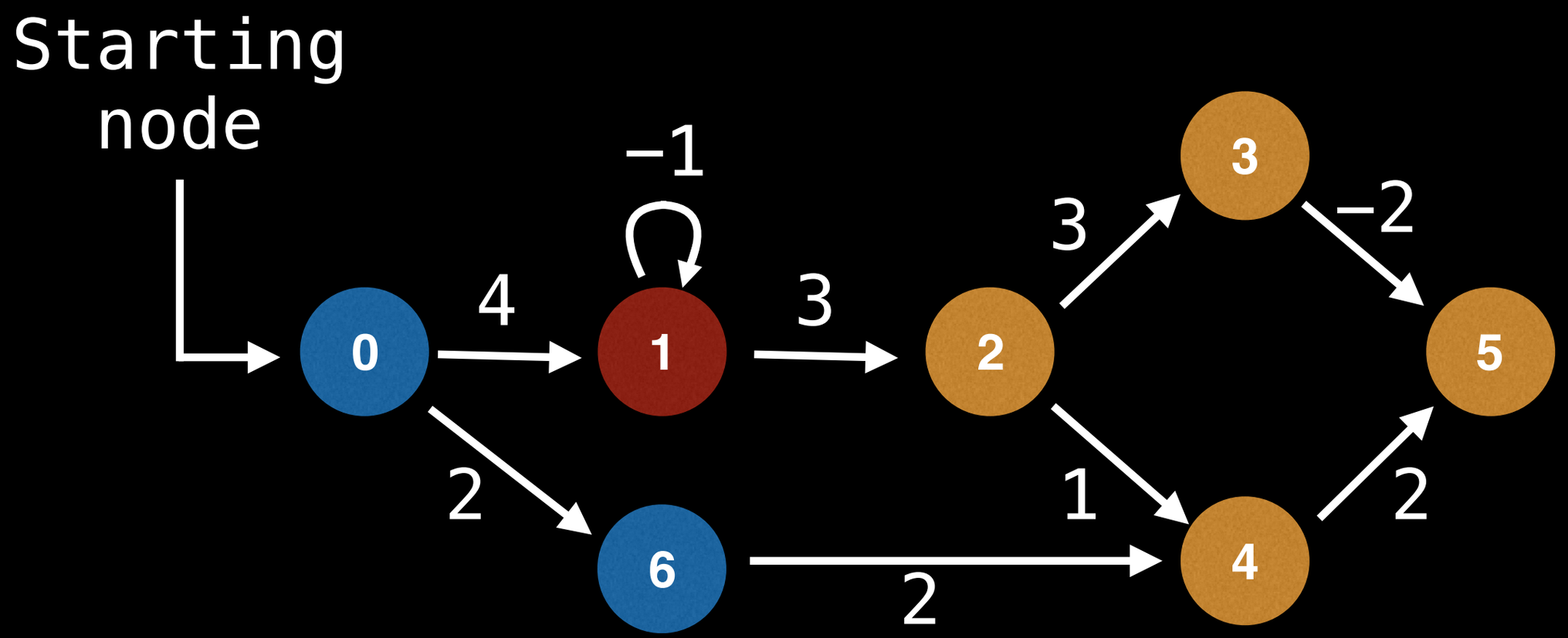
Unaffected node

Directly in negative cycle

Reachable by negative cycle

Negative Cycles

Negative cycles can manifest themselves in many ways…



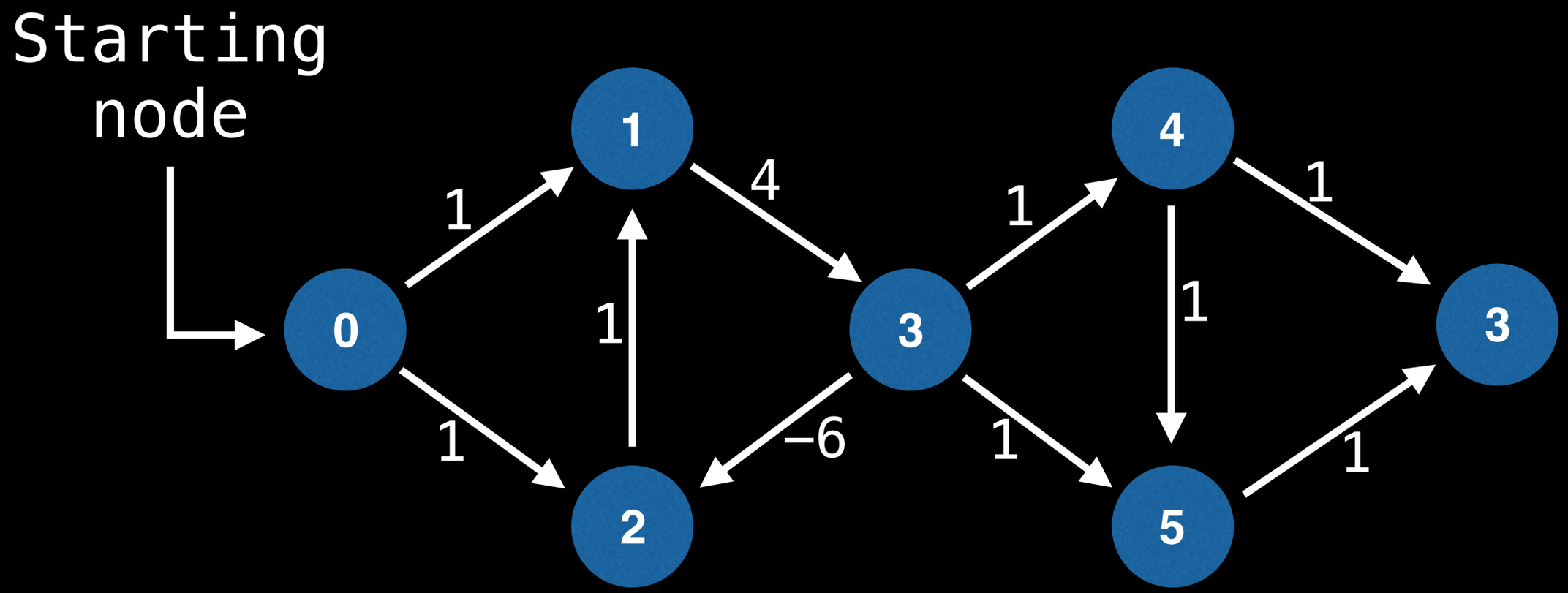
Unaffected node

Directly in negative cycle

Reachable by negative cycle

Negative Cycles

Negative cycles can manifest themselves in many ways…



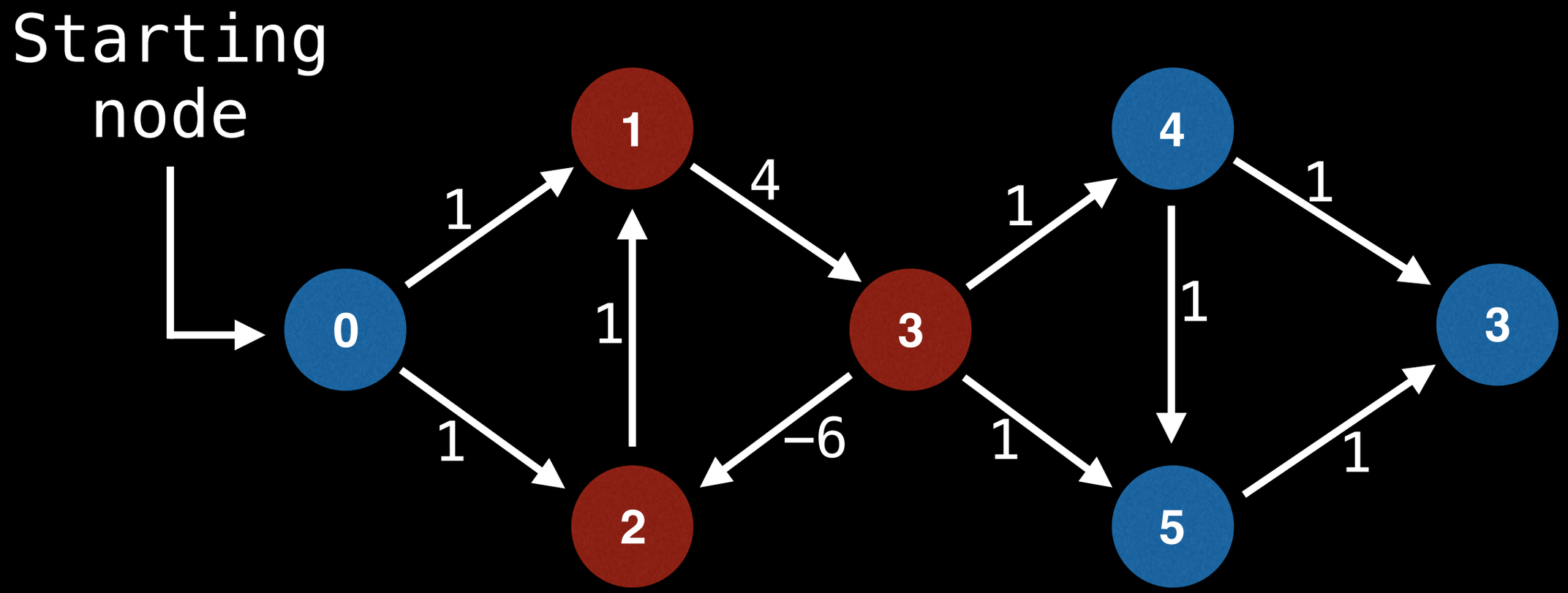
Unaffected node

Directly in negative cycle

Reachable by negative cycle

Negative Cycles

Negative cycles can manifest themselves in many ways…



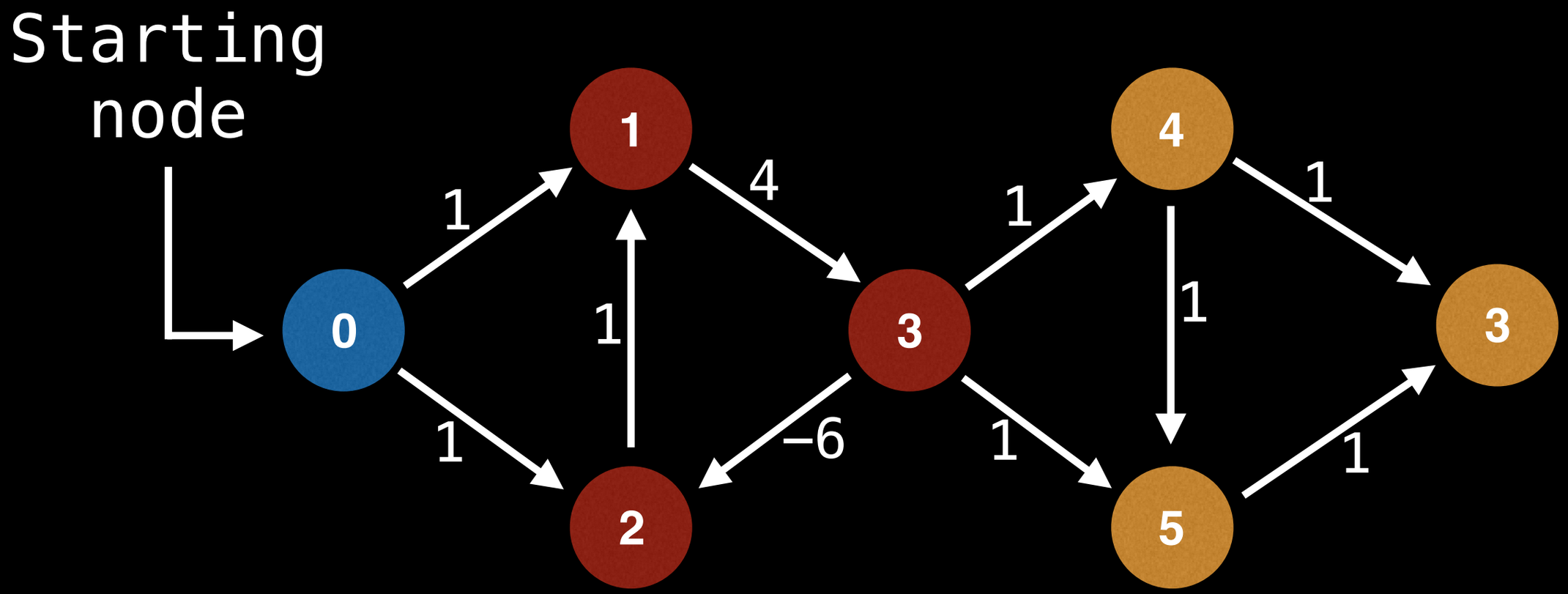
Unaffected node

Directly in negative cycle

Reachable by negative cycle

Negative Cycles

Negative cycles can manifest themselves in many ways…



Unaffected node

Directly in negative cycle

Reachable by negative cycle

BF Algorithm Steps

Let's define a few variables…

Let **E** be the number of edges.

Let **V** be the number of vertices.

Let **S** be the id of the starting node.

Let **D** be an array of size **V** that tracks the best distance from **S** to each node.

BF Algorithm Steps

1) Set every entry in **D** to +∞
2) Set **D**[**S**] = 0
3) Relax each edge V−1 times:

BF Algorithm Steps

1) Set every entry in **D** to +∞
2) Set **D**[**S**] = 0
3) Relax each edge V−1 times:

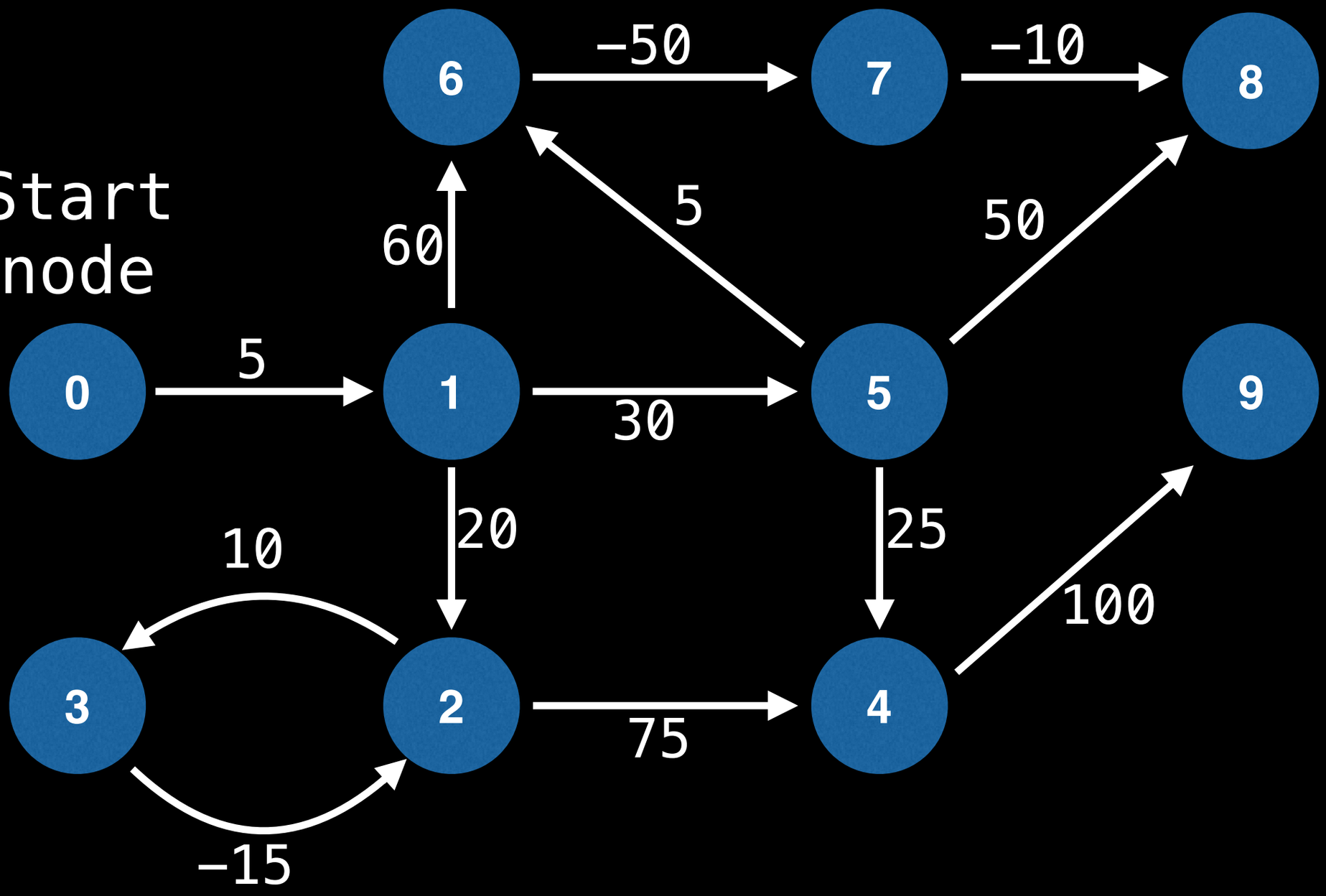
```
for (i = 0; i < V-1; i = i + 1):
  for edge in graph.edges:
    // Relax edge (update D with shorter path)
    if (D[edge.from] + edge.cost < D[edge.to])
      D[edge.to] = D[edge.from] + edge.cost
```

BF Algorithm Steps

1) Set every entry in **D** to +∞
2) Set **D**[**S**] = 0
3) Relax each edge V−1 times:

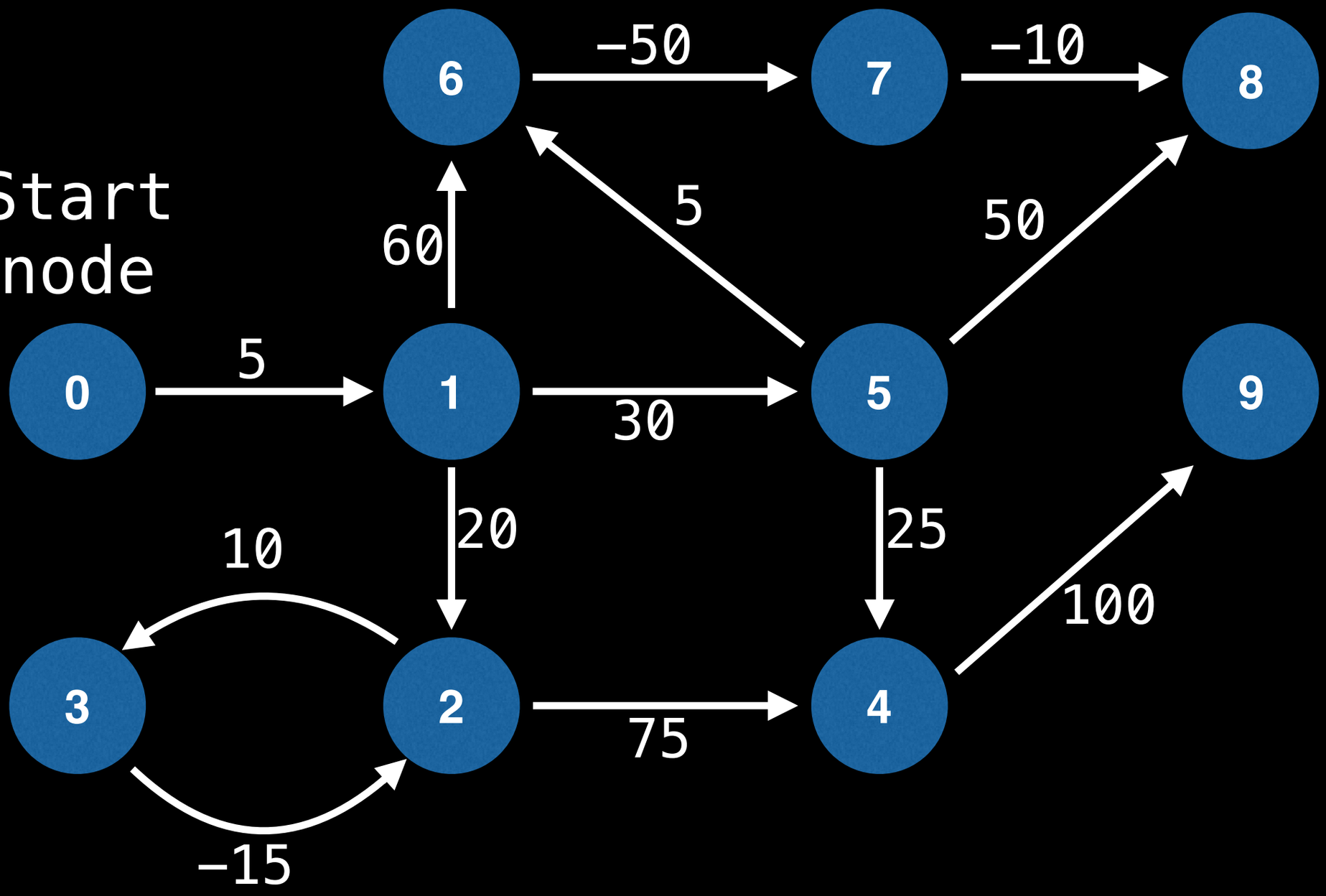
```
for (i = 0; i < V-1; i = i + 1):
  for edge in graph.edges:
    // Relax edge (update D with shorter path)
    if (D[edge.from] + edge.cost < D[edge.to])
      D[edge.to] = D[edge.from] + edge.cost

// Repeat to find nodes caught in a negative cycle
for (i = 0; i < V-1; i = i + 1):
  for edge in graph.edges:
    if (D[edge.from] + edge.cost < D[edge.to])
      D[edge.to] = -∞
```



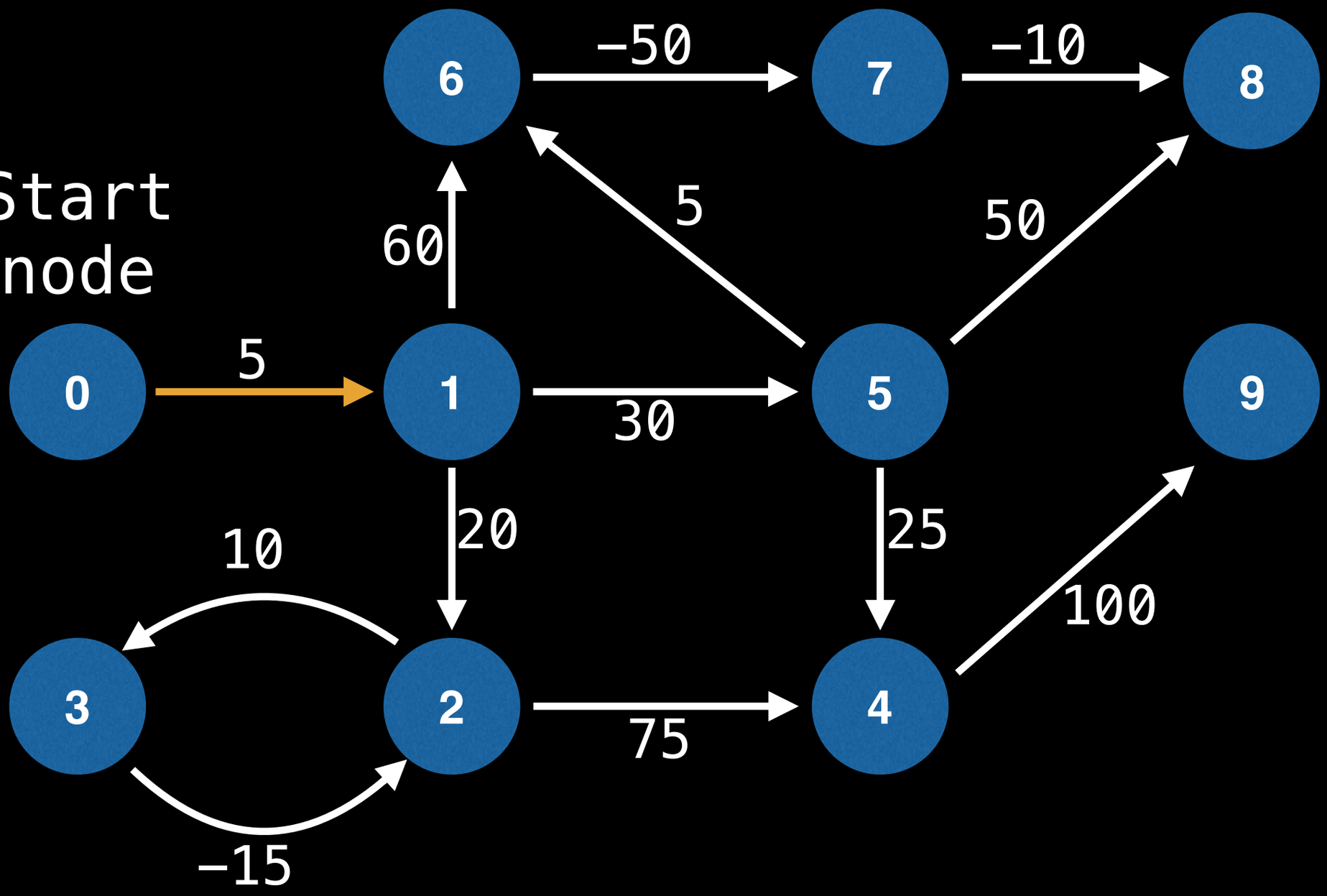
0	∞
1	∞
2	∞
3	∞
4	∞
5	∞
6	∞
7	∞
8	∞
9	∞

NOTE: The edges do not need to be chosen in any specific order.



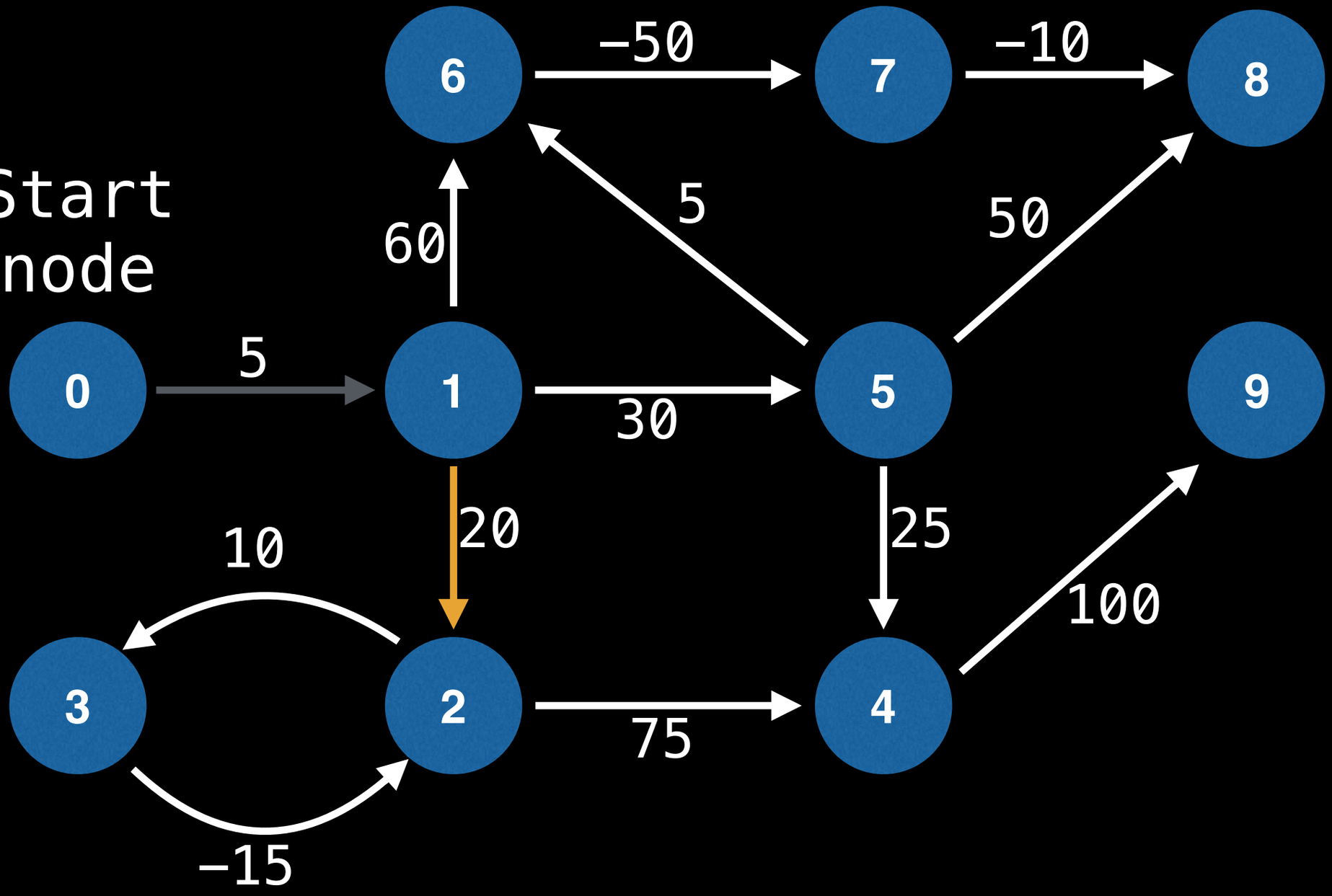
0	0
1	∞
2	∞
3	∞
4	∞
5	∞
6	∞
7	∞
8	∞
9	∞

NOTE: The edges do not need to be chosen in any specific order.



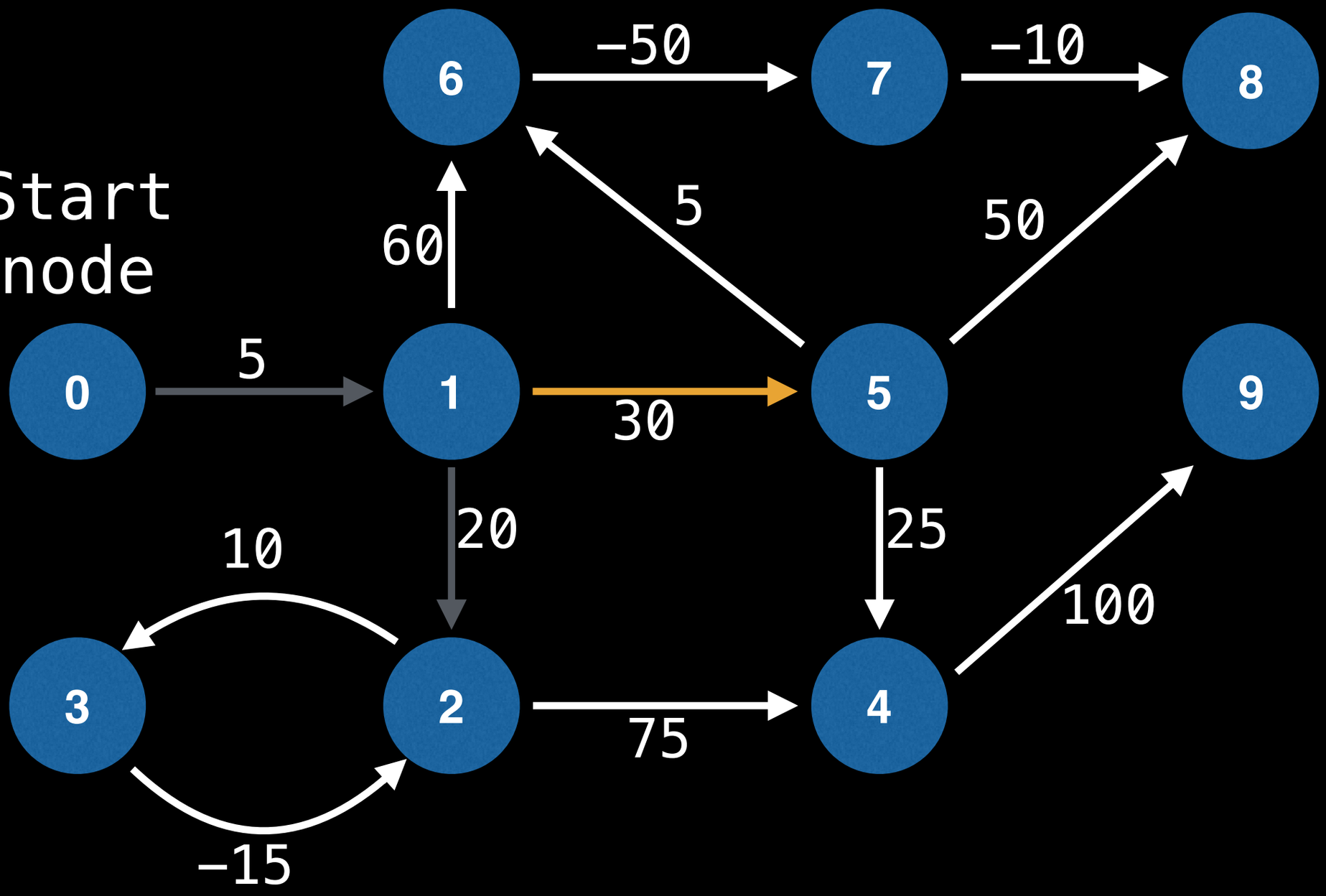
0	0
1	5
2	∞
3	∞
4	∞
5	∞
6	∞
7	∞
8	∞
9	∞

NOTE: The edges do not need to be chosen in any specific order.



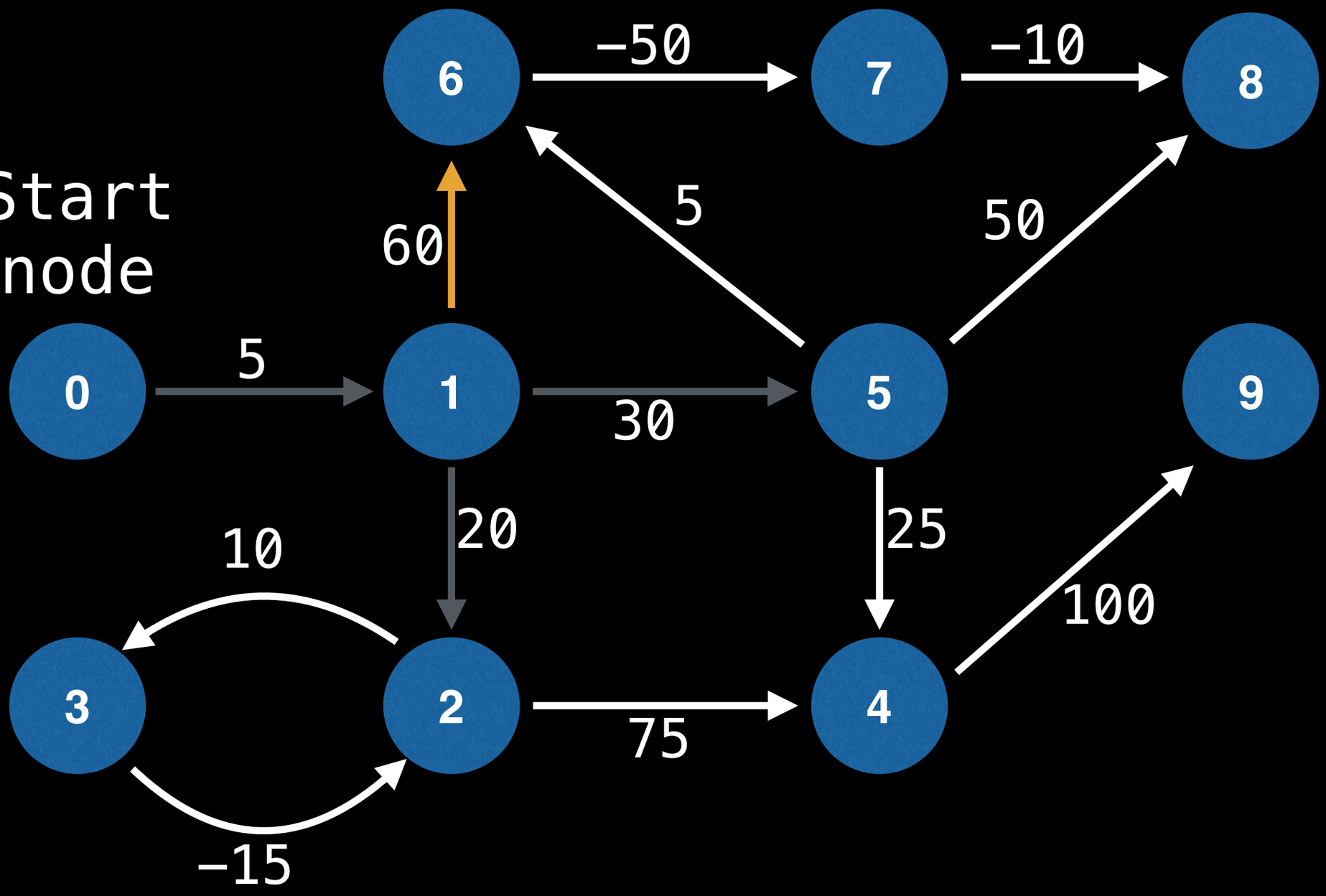
0	0
1	5
2	25
3	∞
4	∞
5	∞
6	∞
7	∞
8	∞
9	∞

NOTE: The edges do not need to be chosen in any specific order.



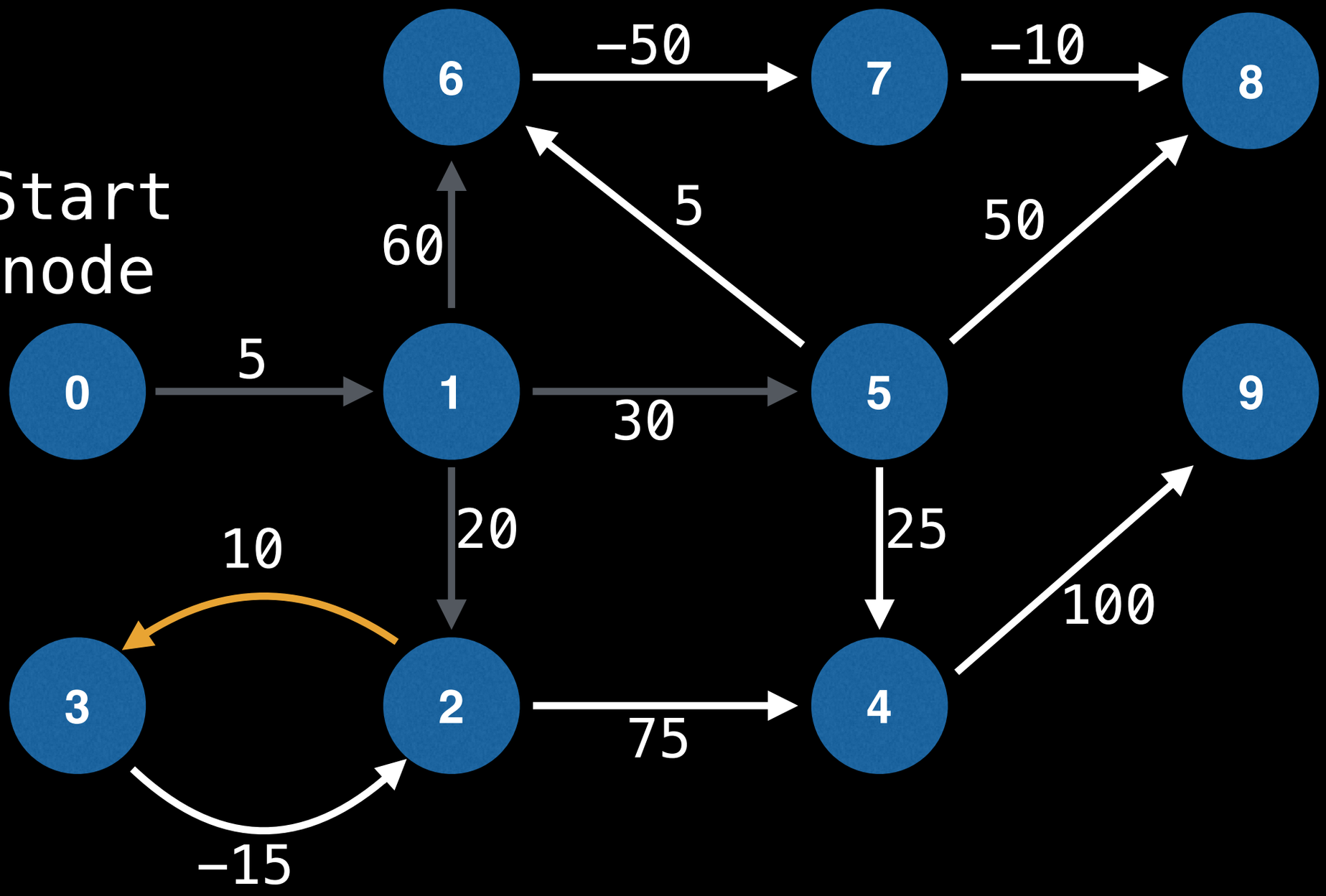
0	0
1	5
2	25
3	∞
4	∞
5	35
6	∞
7	∞
8	∞
9	∞

NOTE: The edges do not need to be chosen in any specific order.



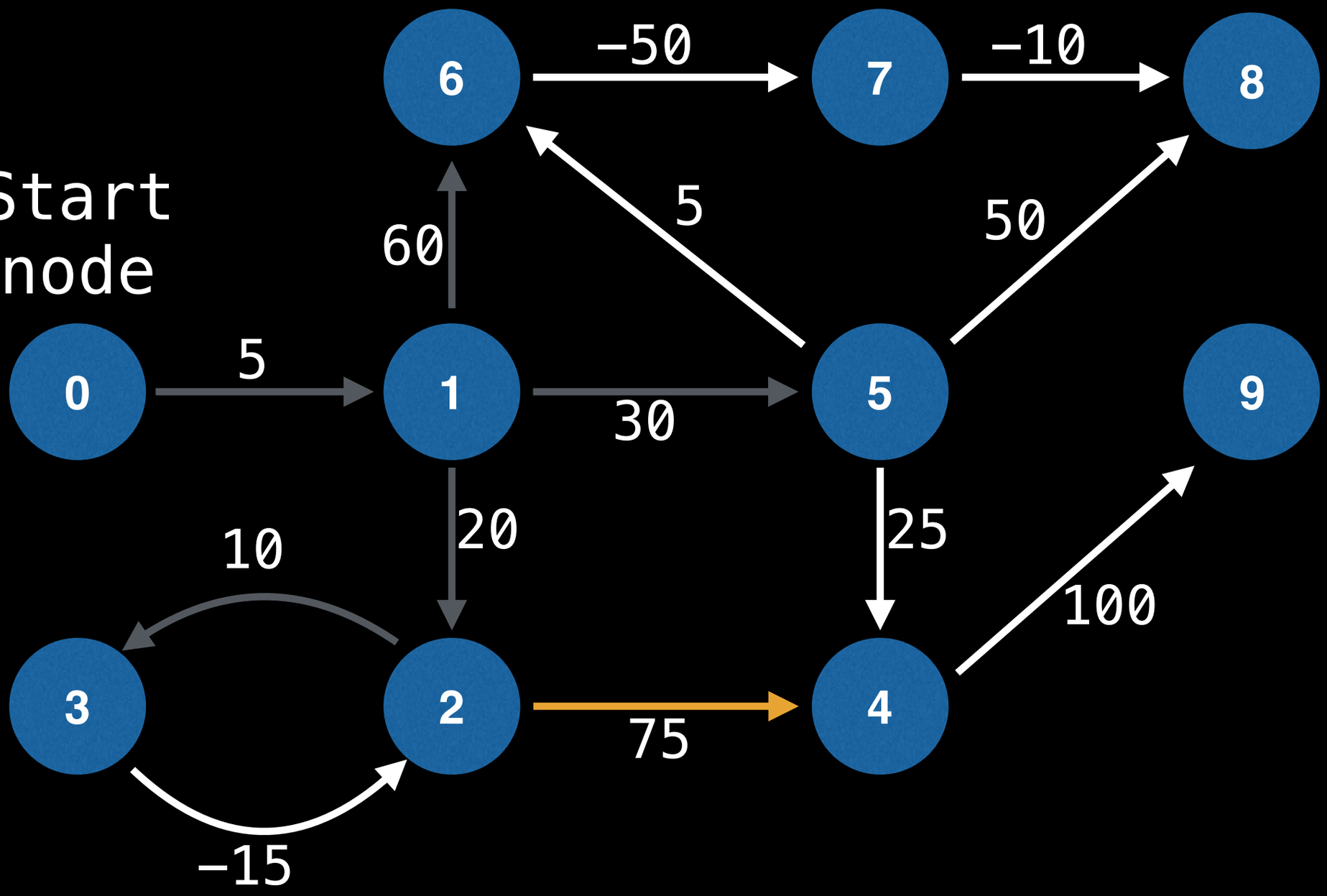
0	0
1	5
2	25
3	∞
4	∞
5	35
6	65
7	∞
8	∞
9	∞

NOTE: The edges do not need to be chosen in any specific order.



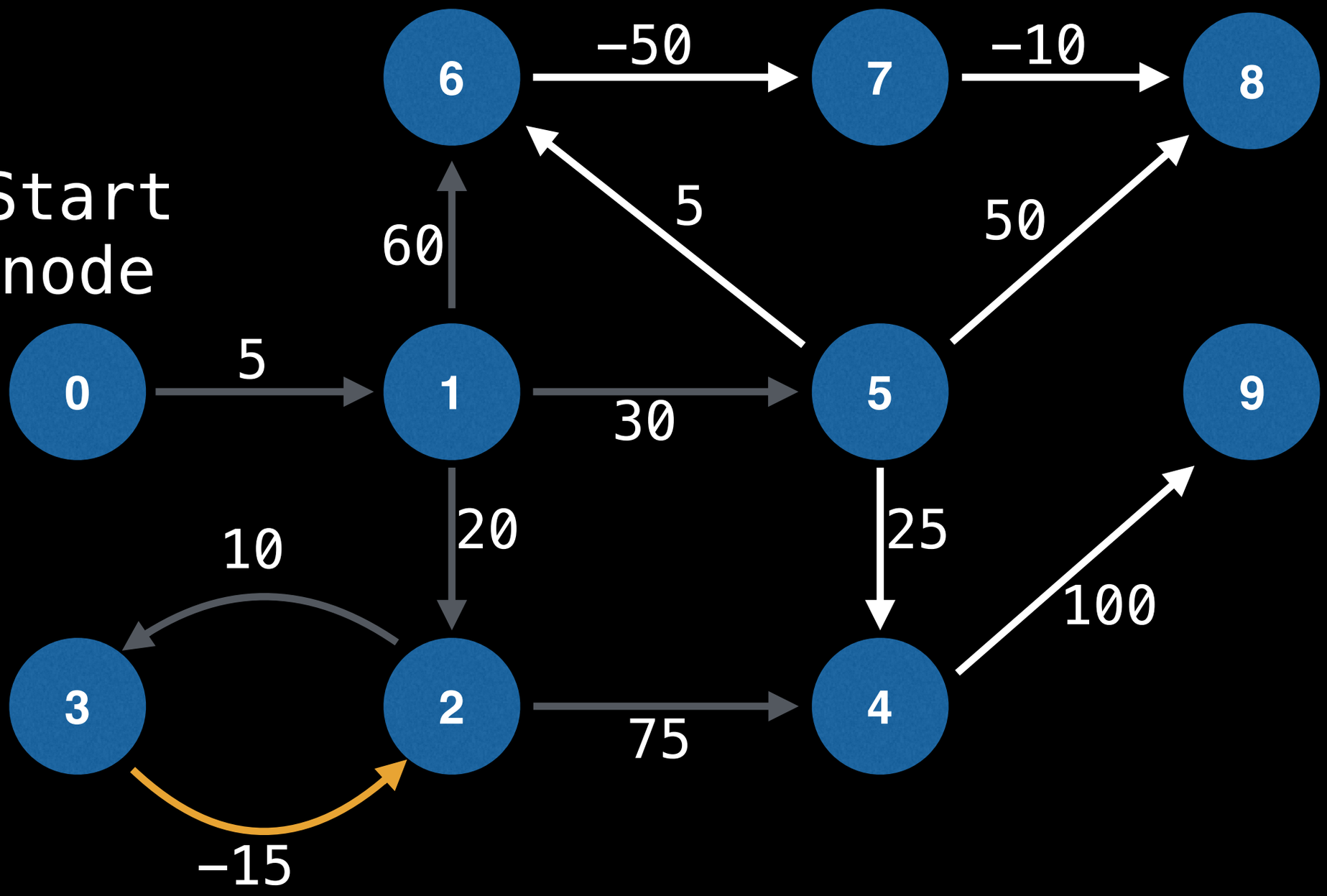
0	0
1	5
2	25
3	35
4	∞
5	35
6	65
7	∞
8	∞
9	∞

NOTE: The edges do not need to be chosen in any specific order.



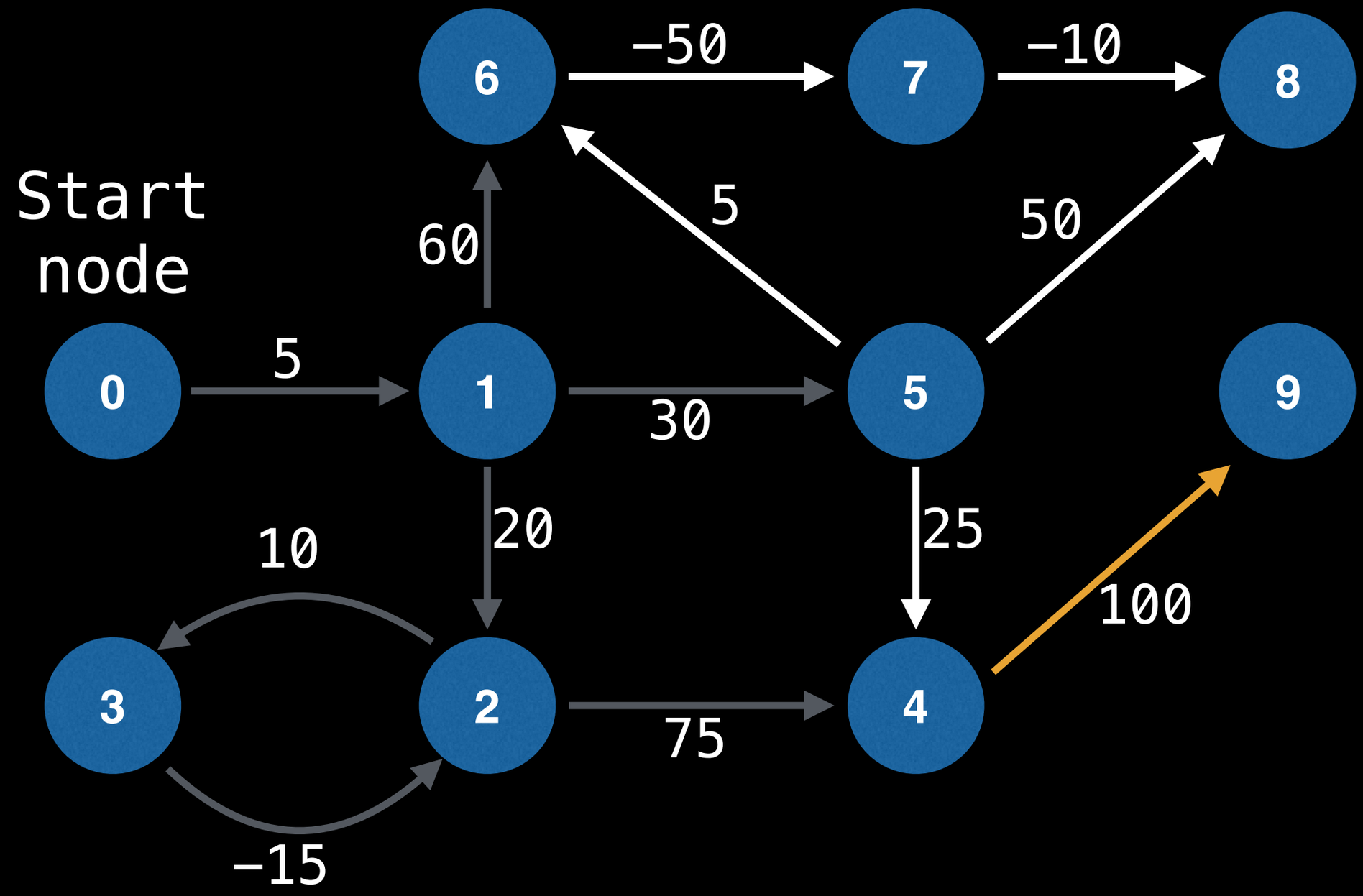
0	0
1	5
2	25
3	35
4	100
5	35
6	65
7	∞
8	∞
9	∞

NOTE: The edges do not need to be chosen in any specific order.



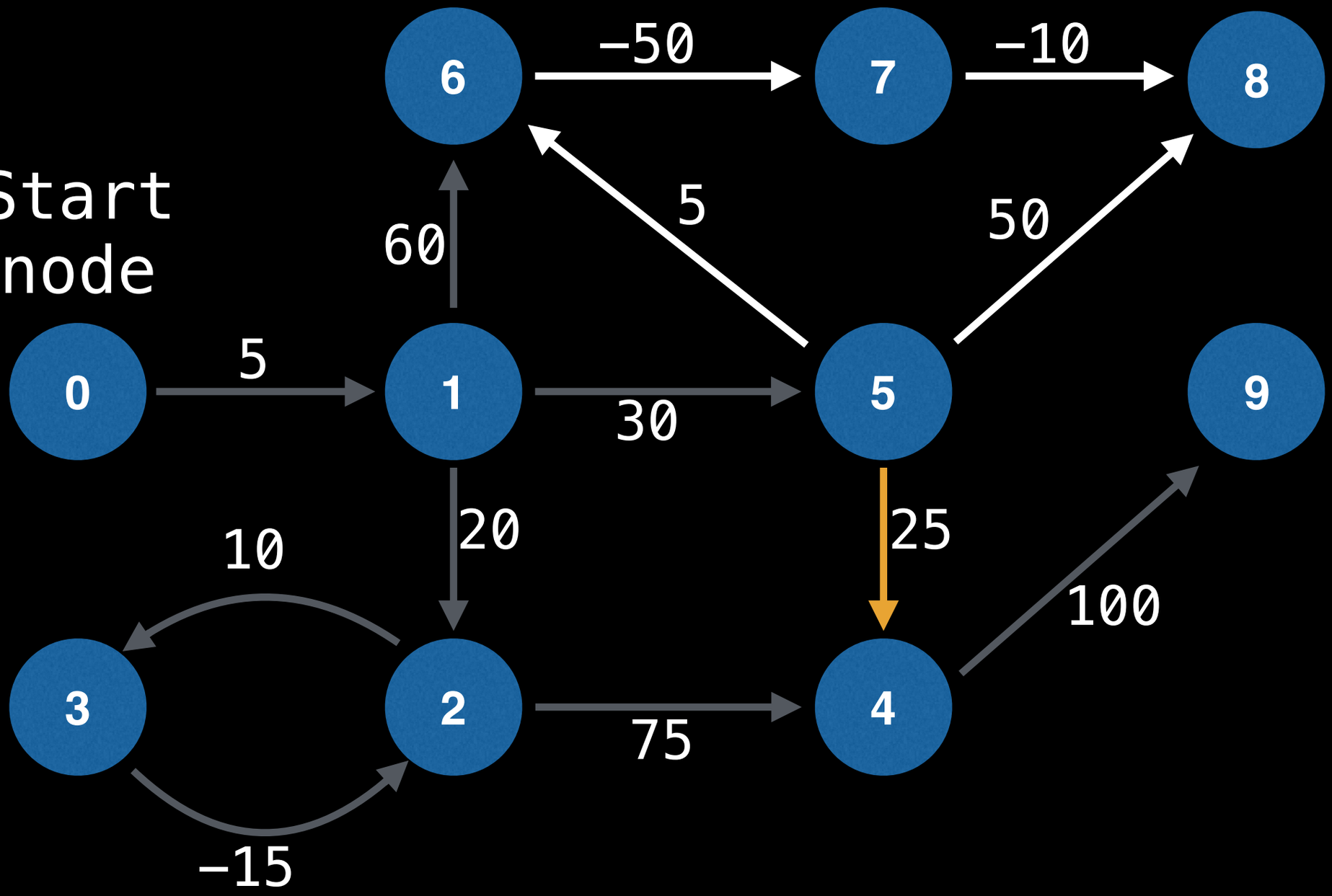
0	0
1	5
2	20
3	35
4	100
5	35
6	65
7	∞
8	∞
9	∞

NOTE: The edges do not need to be chosen in any specific order.



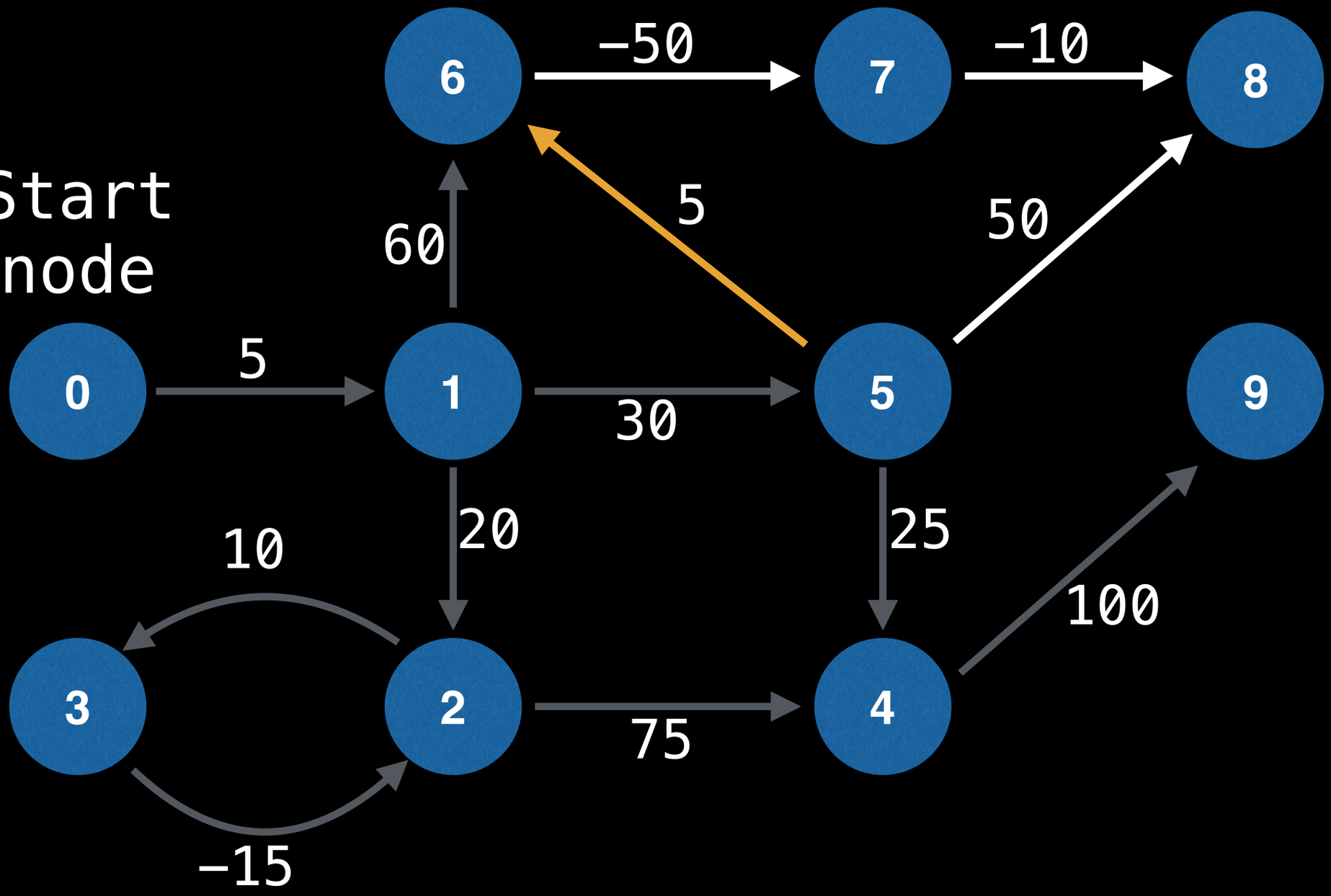
0	0
1	5
2	20
3	35
4	100
5	35
6	65
7	∞
8	∞
9	200

NOTE: The edges do not need to be chosen in any specific order.



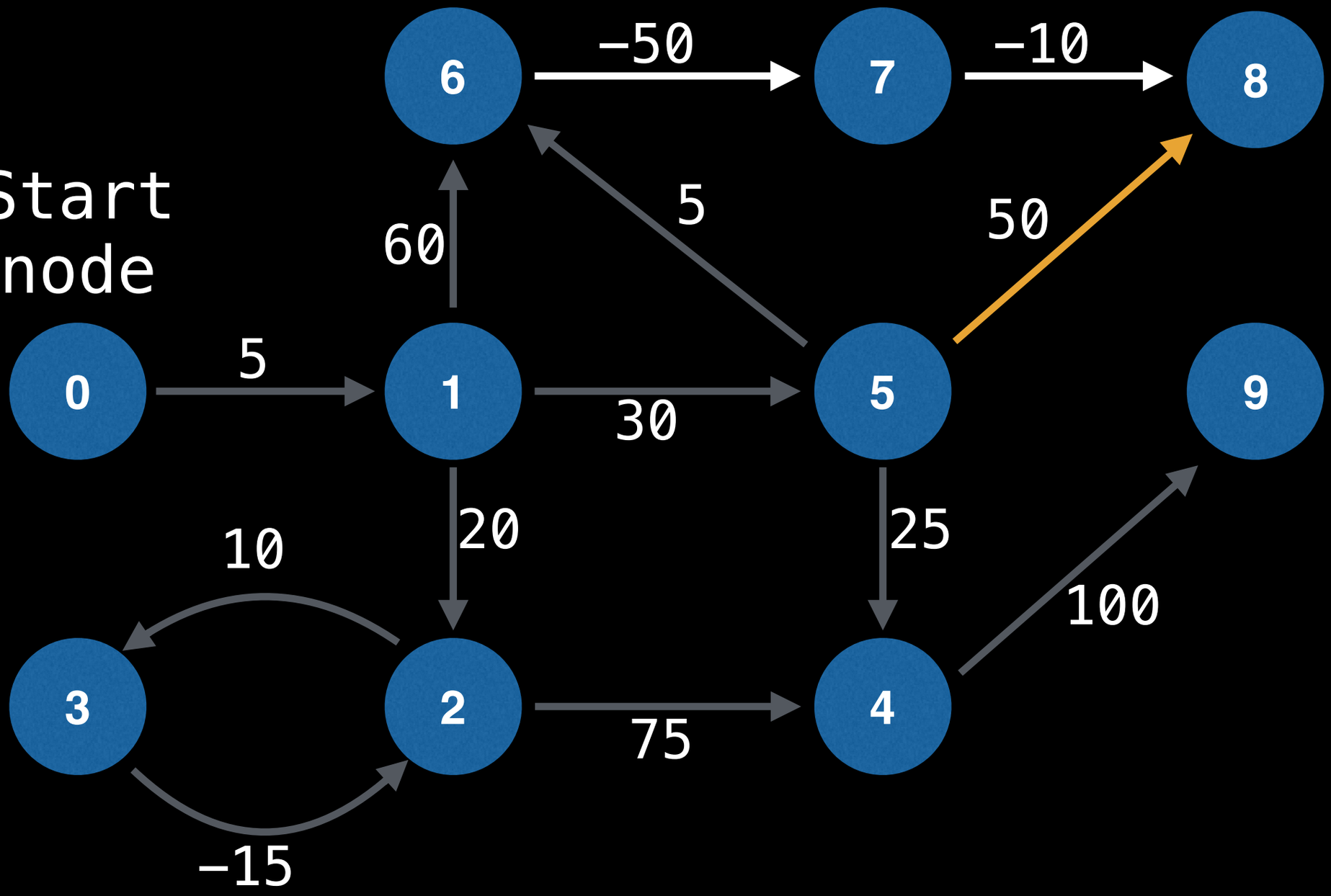
0	0
1	5
2	20
3	35
4	60
5	35
6	65
7	∞
8	∞
9	200

NOTE: The edges do not need to be chosen in any specific order.



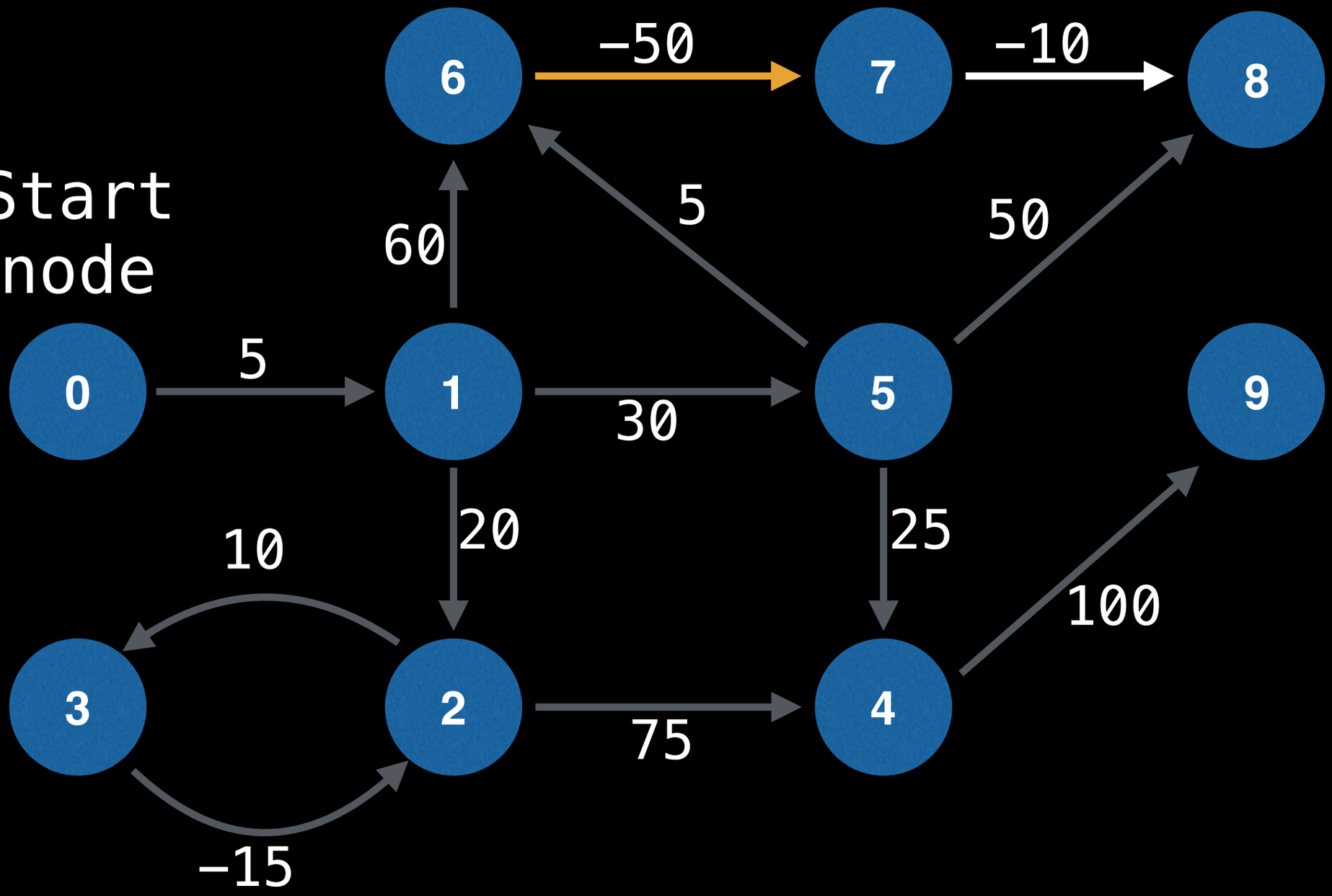
0	0
1	5
2	20
3	35
4	60
5	35
6	40
7	∞
8	∞
9	200

NOTE: The edges do not need to be chosen in any specific order.



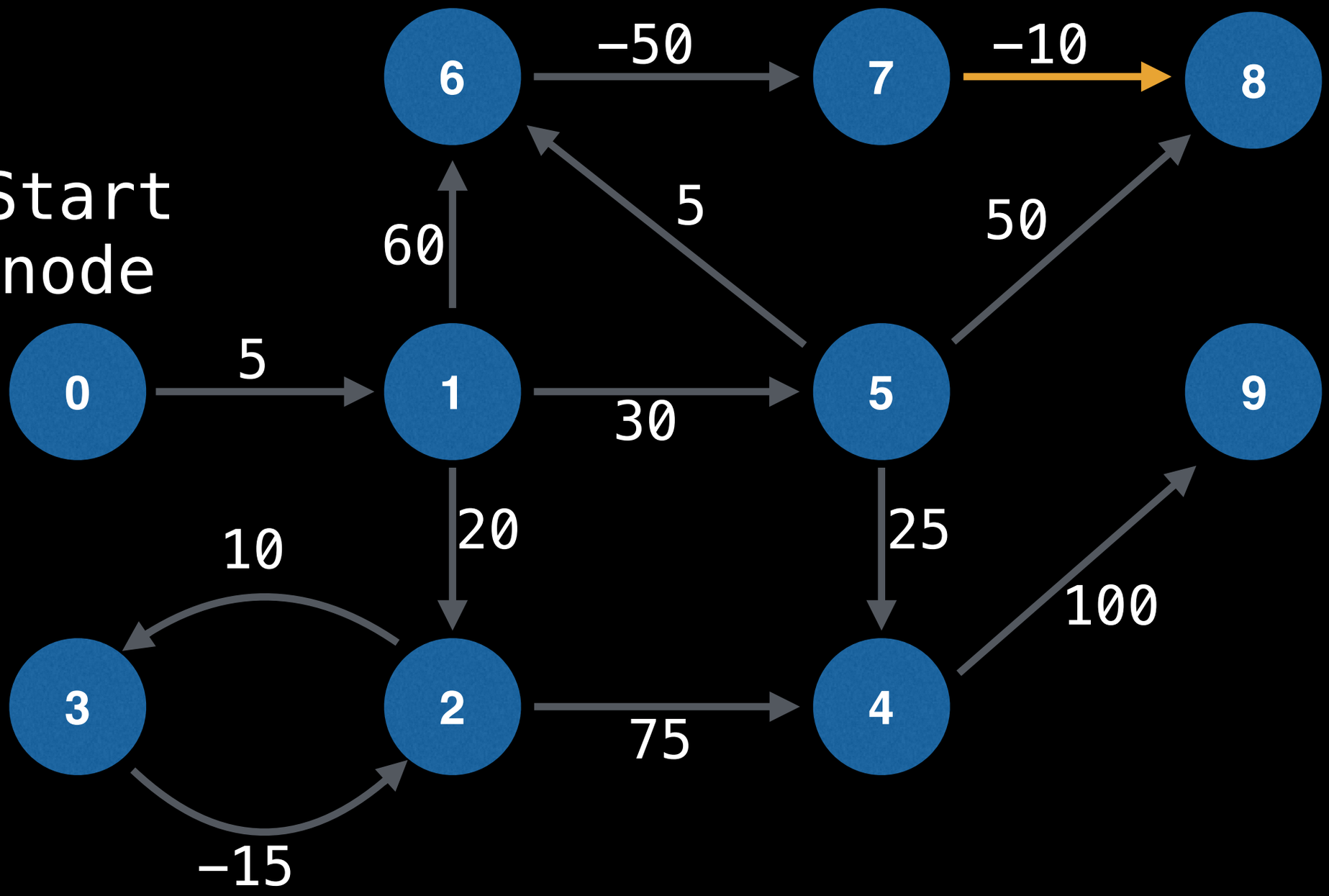
0	0
1	5
2	20
3	35
4	60
5	35
6	40
7	∞
8	85
9	200

NOTE: The edges do not need to be chosen in any specific order.



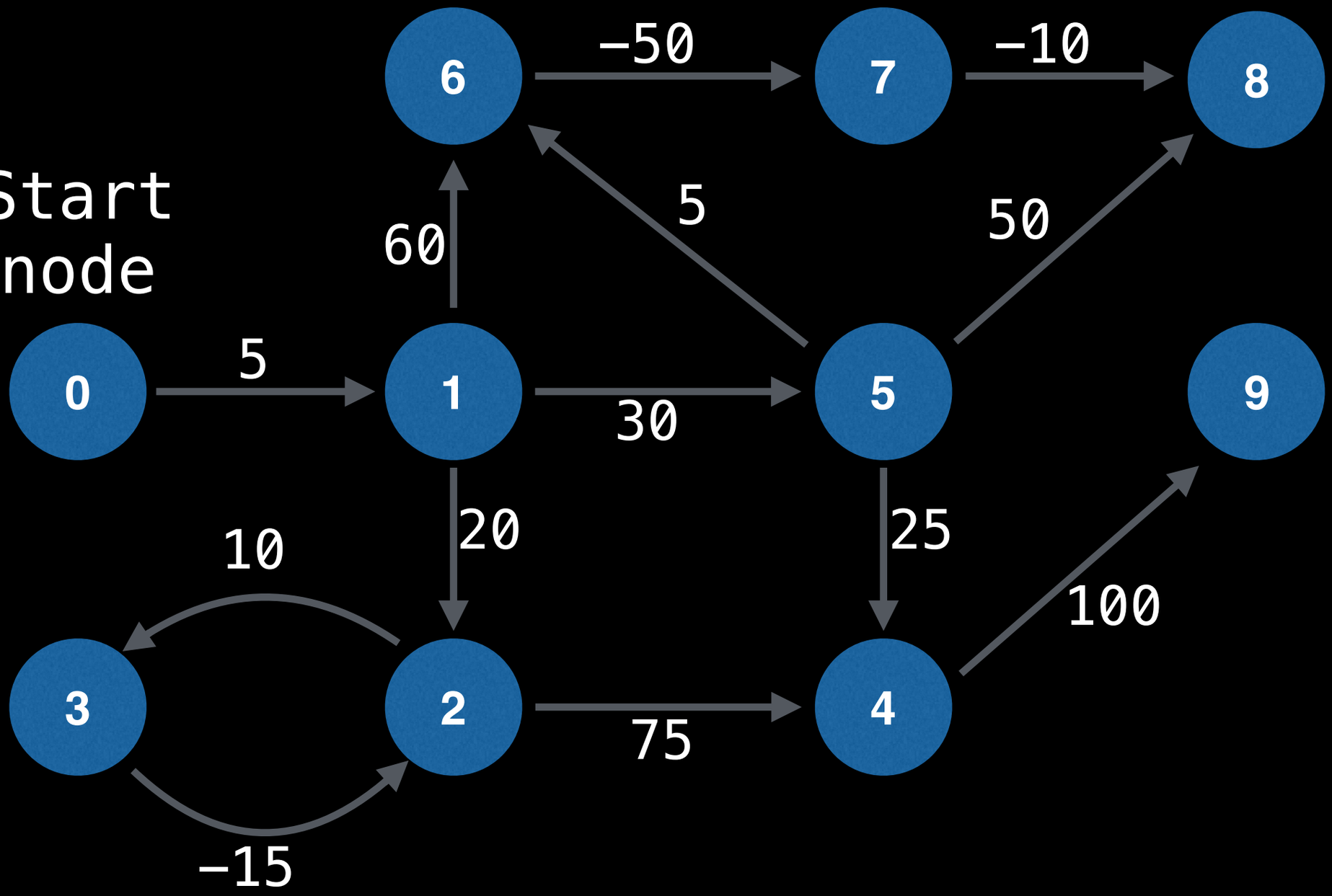
0	0
1	5
2	20
3	35
4	60
5	35
6	40
7	-10
8	85
9	200

NOTE: The edges do not need to be chosen in any specific order.



0	0
1	5
2	20
3	35
4	60
5	35
6	40
7	-10
8	-20
9	200

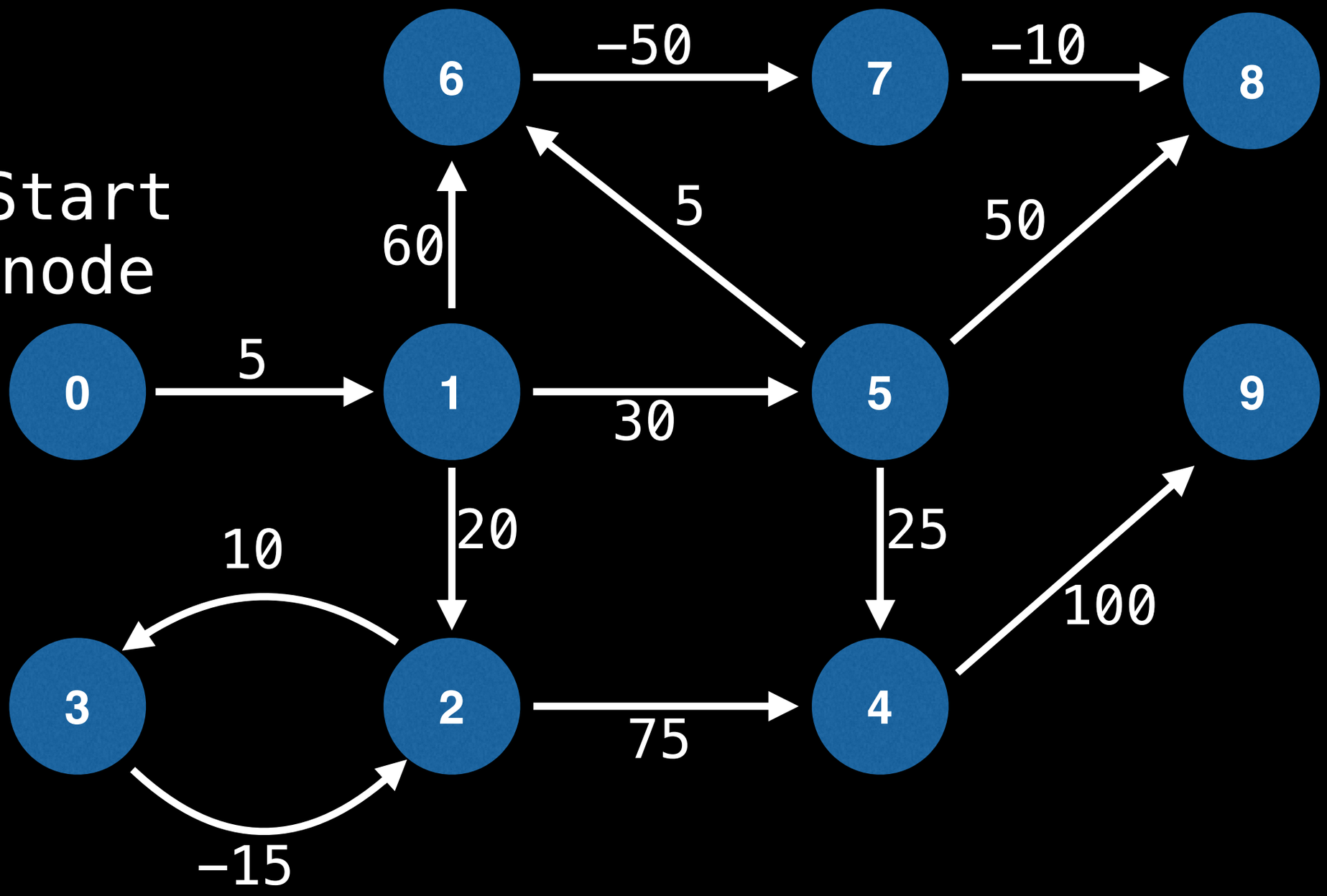
NOTE: The edges do not need to be chosen in any specific order.



0	0
1	5
2	20
3	35
4	60
5	35
6	40
7	-10
8	-20
9	200

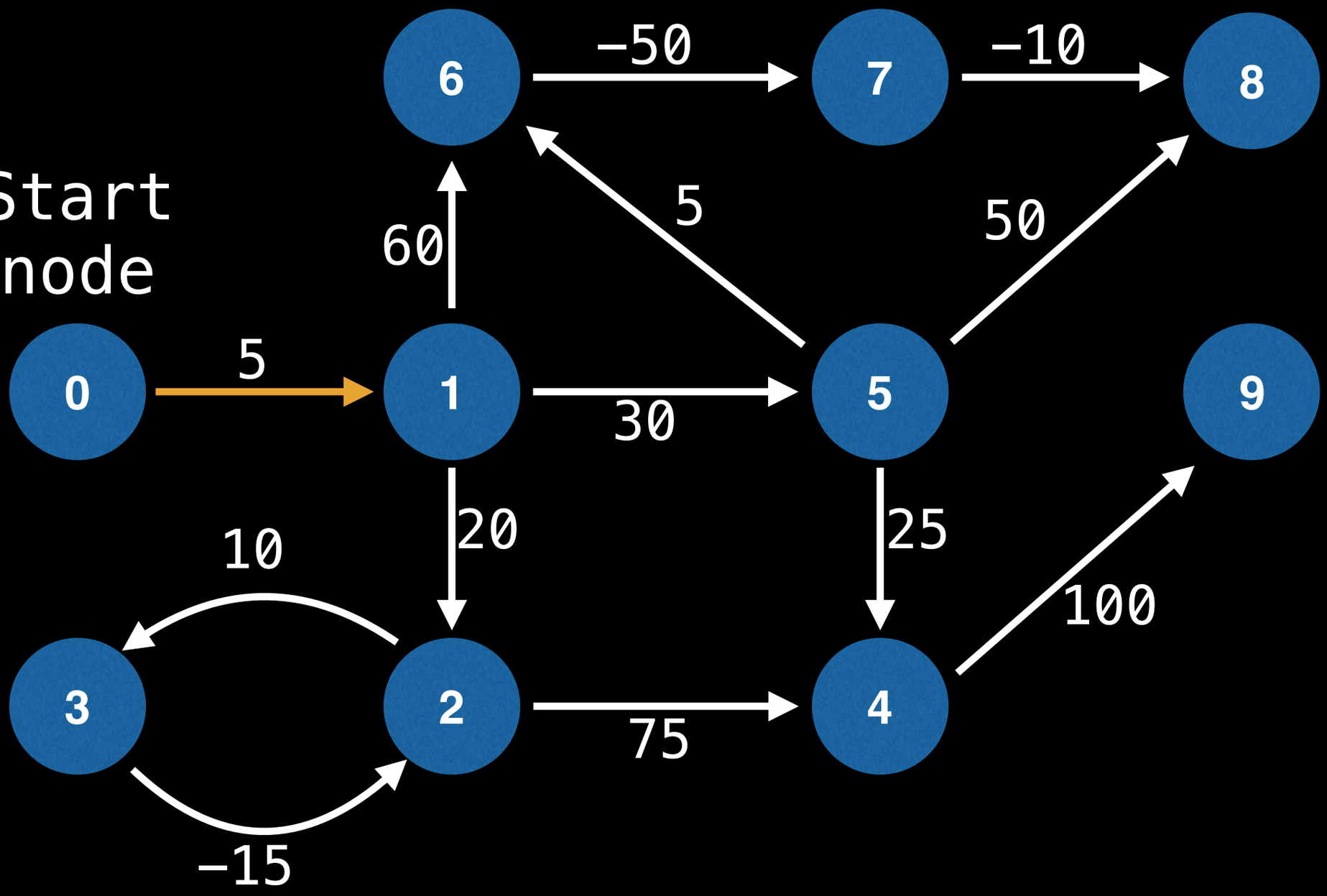
Iteration 1 complete, 8 more to go…

NOTE: The edges do not need to be chosen in any specific order.



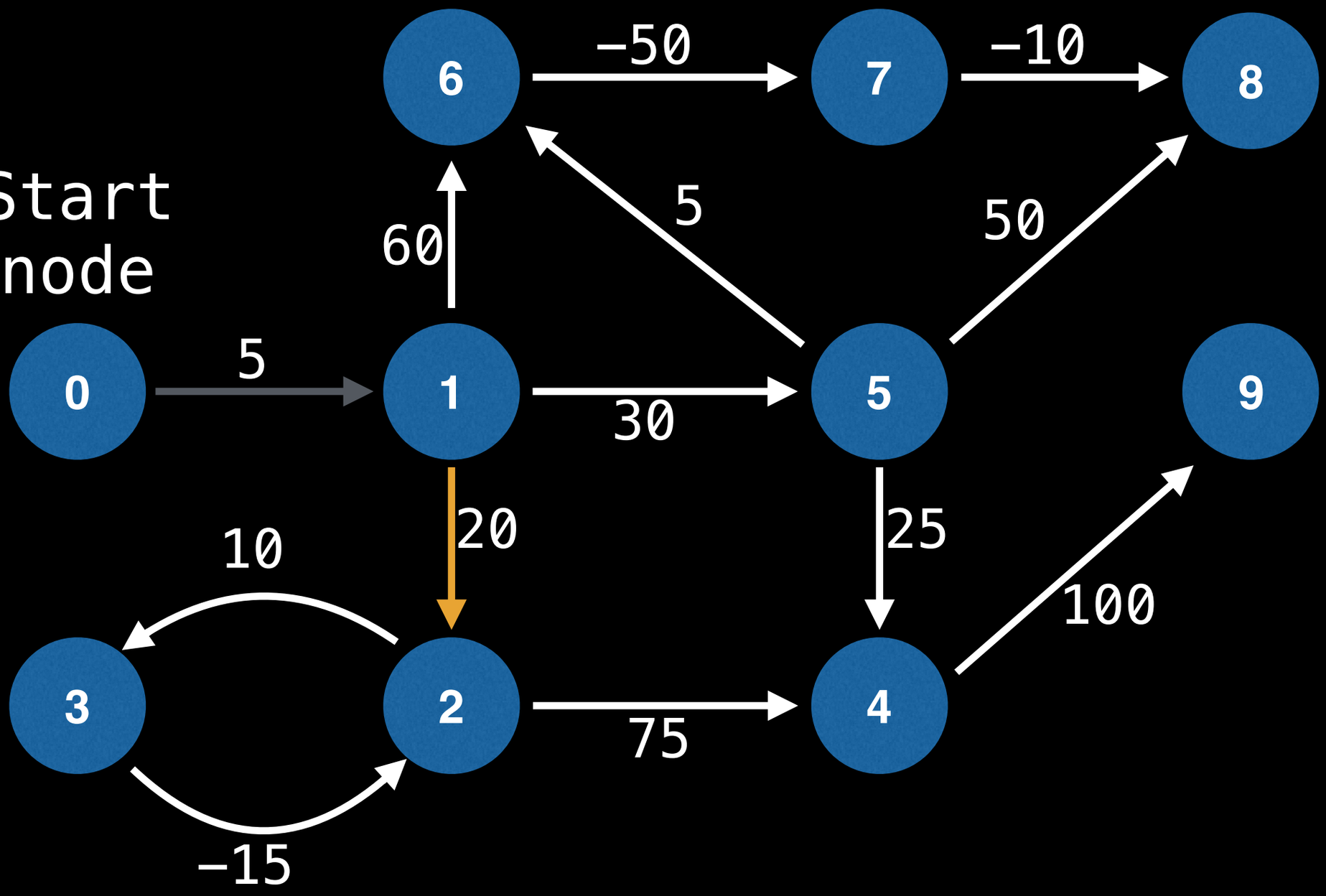
0	0
1	5
2	20
3	35
4	60
5	35
6	40
7	-10
8	-20
9	200

NOTE: The edges do not need to be chosen in any specific order.



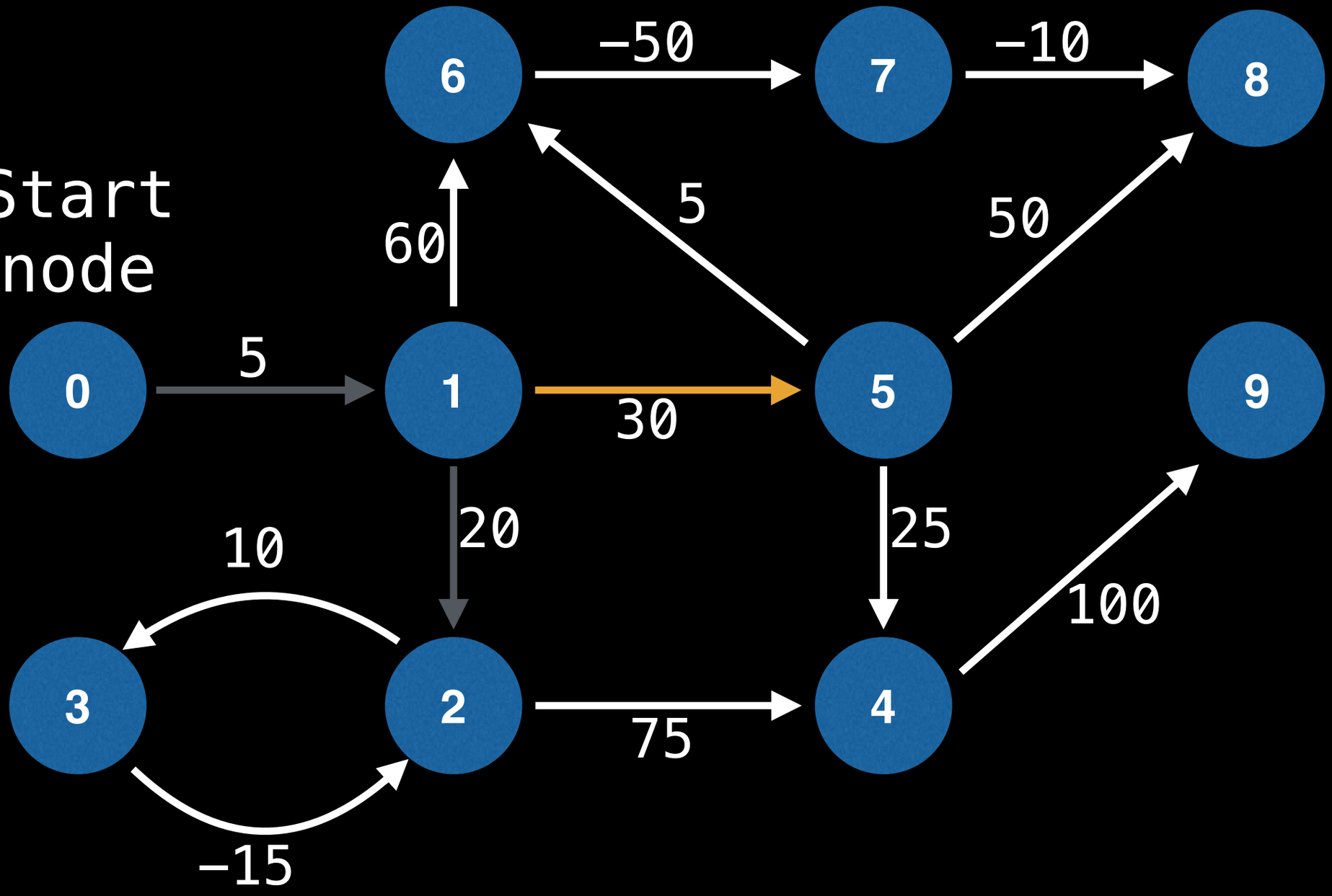
0	0
1	5
2	20
3	35
4	60
5	35
6	40
7	-10
8	-20
9	200

NOTE: The edges do not need to be chosen in any specific order.



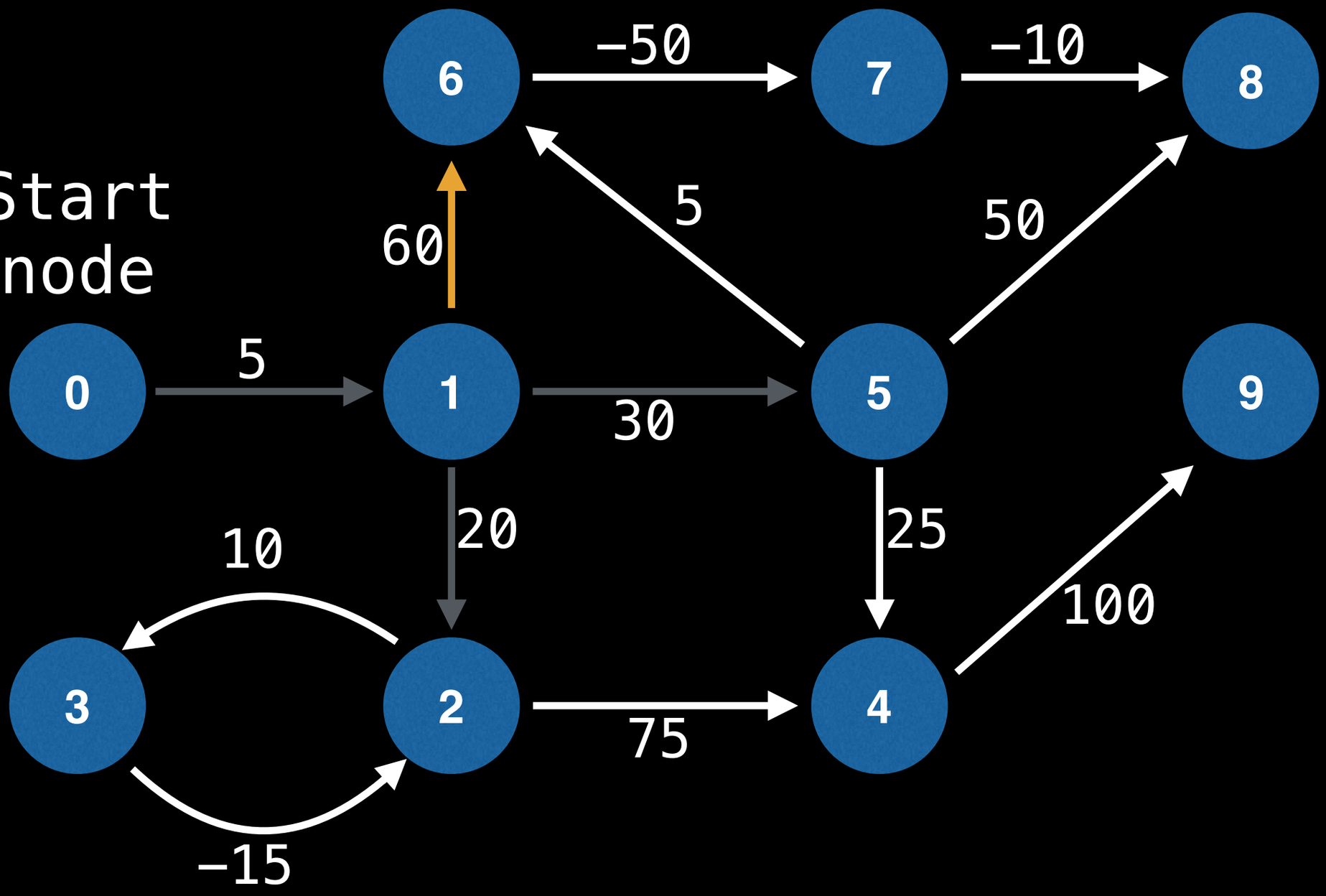
0	0
1	5
2	20
3	35
4	60
5	35
6	40
7	-10
8	-20
9	200

NOTE: The edges do not need to be chosen in any specific order.



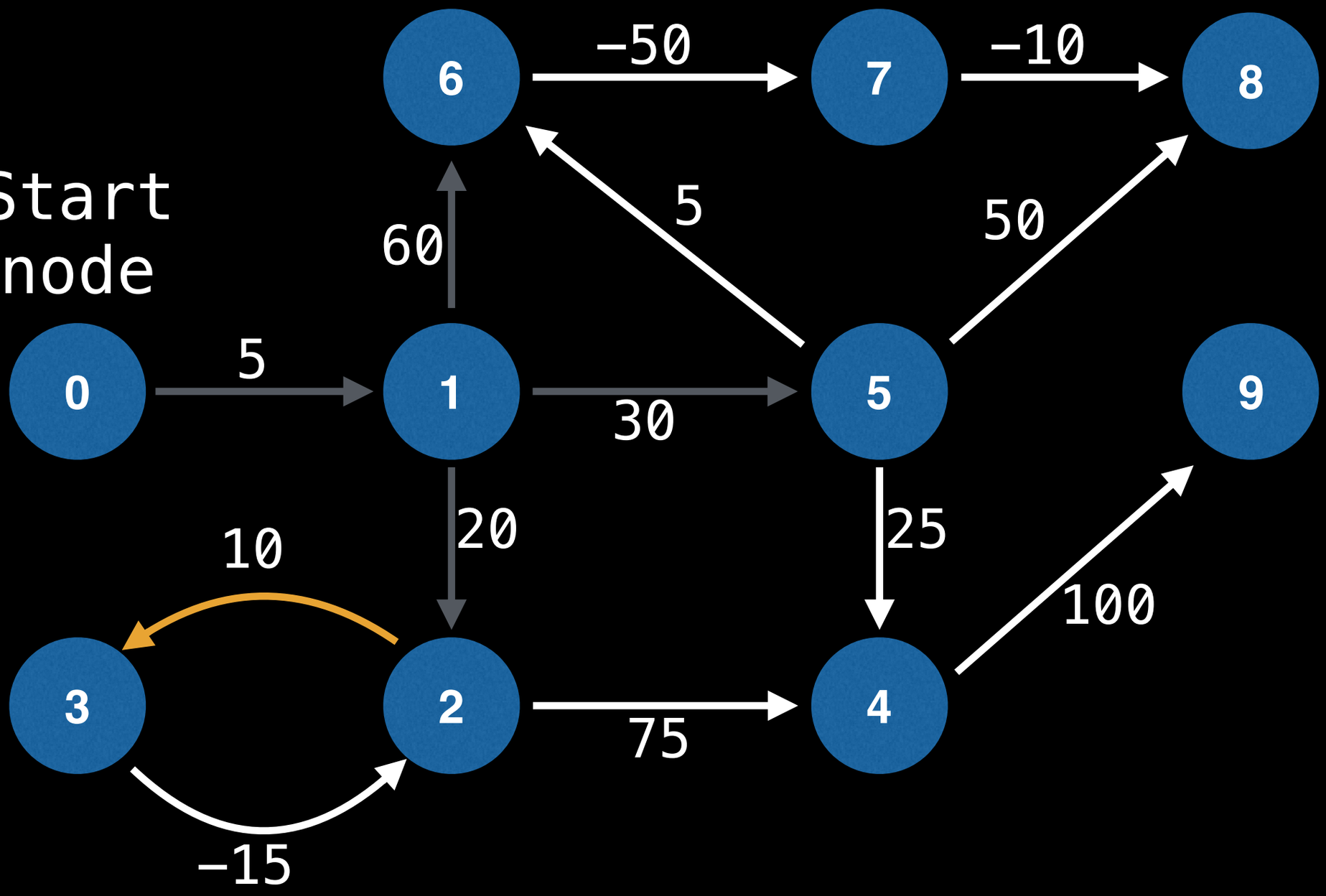
0	0
1	5
2	20
3	35
4	60
5	35
6	40
7	-10
8	-20
9	200

NOTE: The edges do not need to be chosen in any specific order.



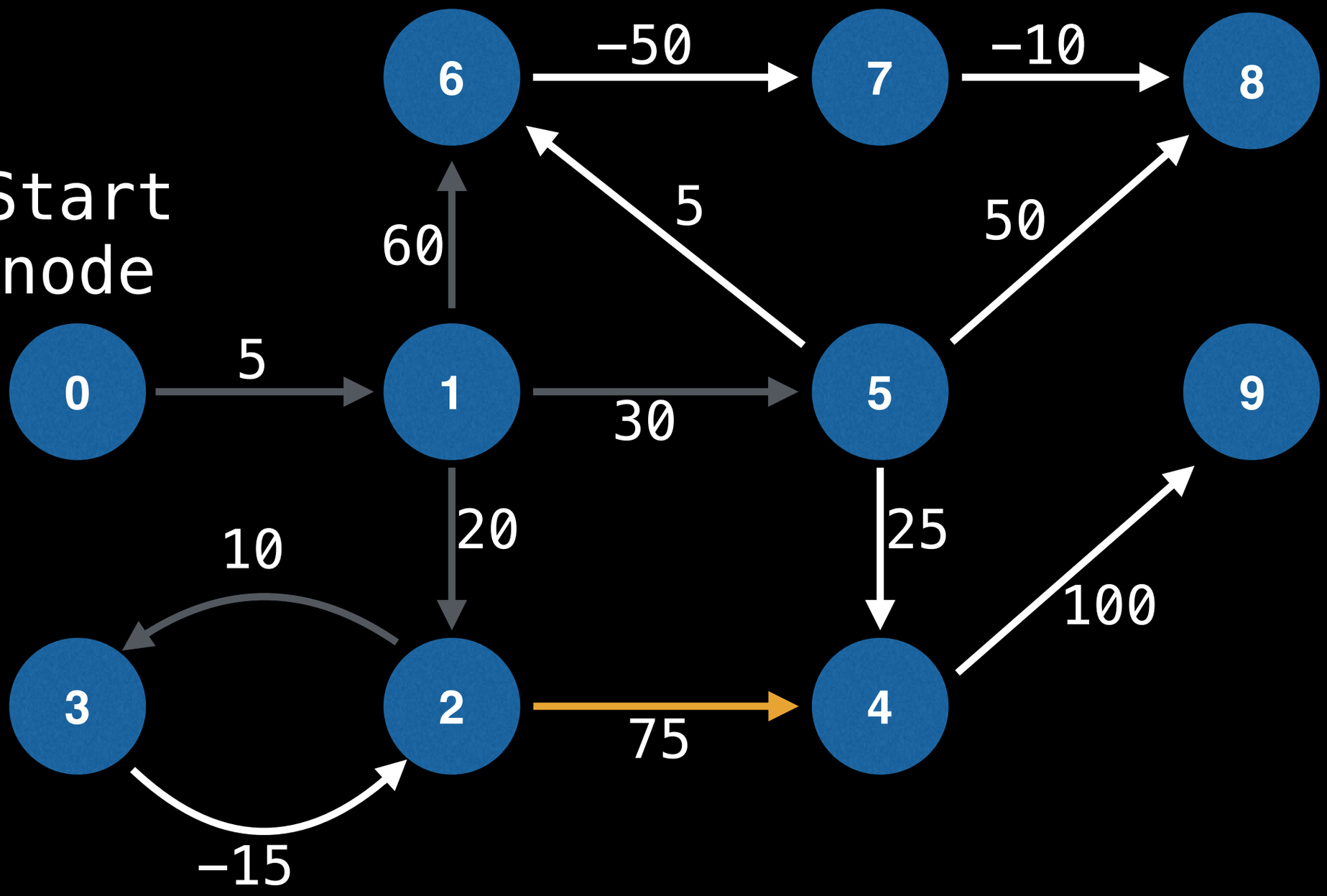
0	0
1	5
2	20
3	35
4	60
5	35
6	40
7	-10
8	-20
9	200

NOTE: The edges do not need to be chosen in any specific order.



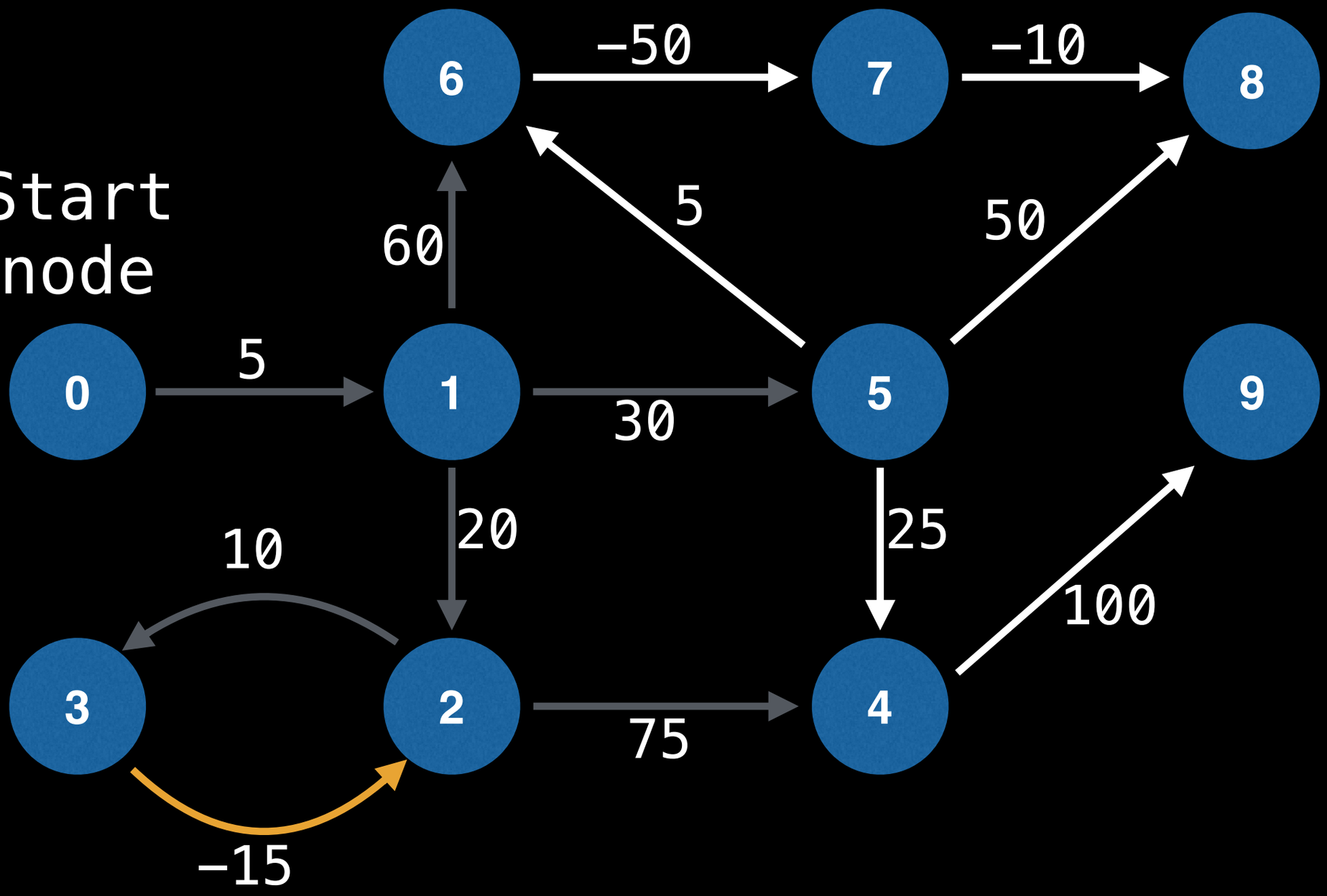
0	0
1	5
2	20
3	30
4	60
5	35
6	40
7	-10
8	-20
9	200

NOTE: The edges do not need to be chosen in any specific order.



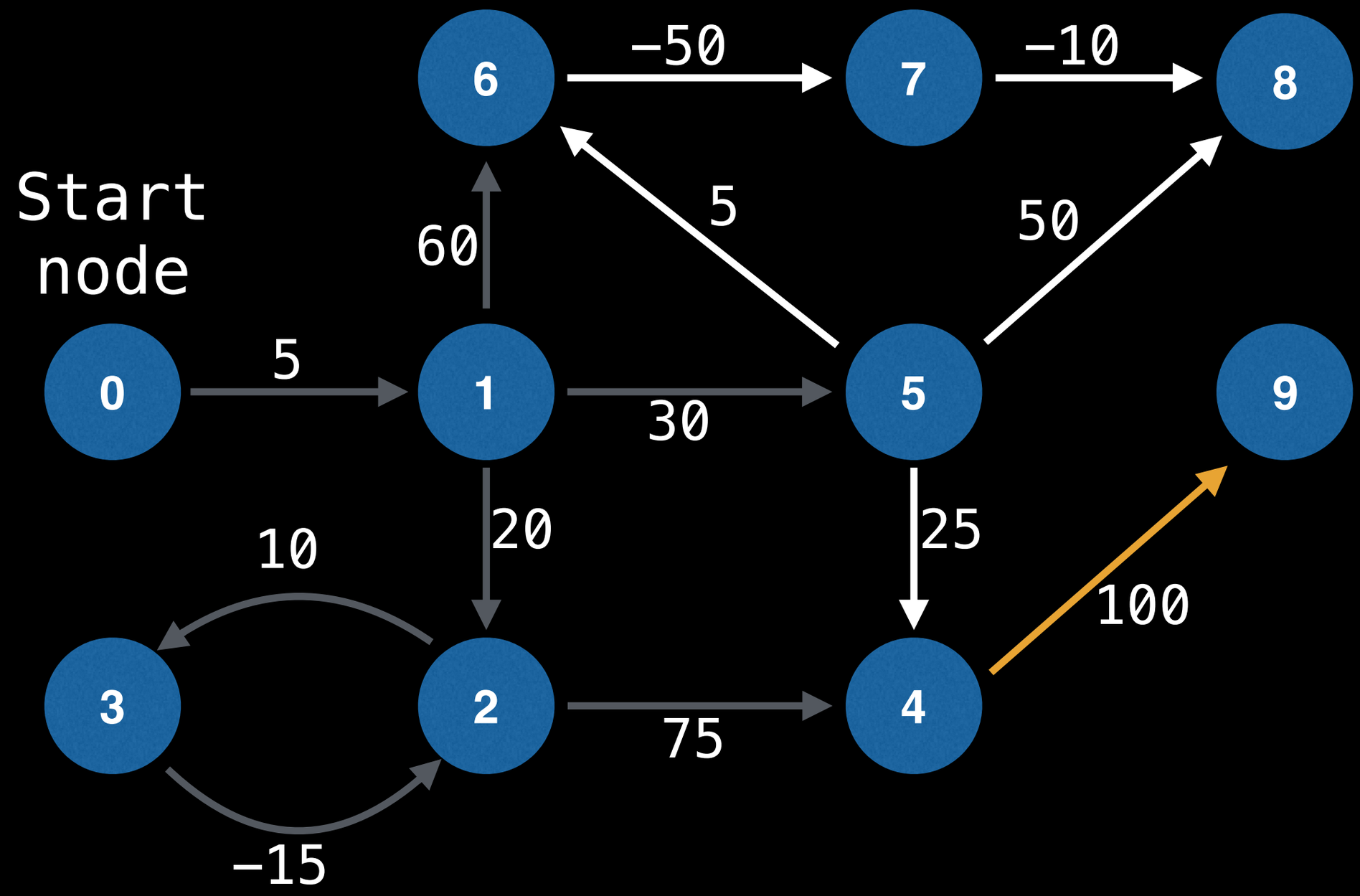
0	0
1	5
2	20
3	30
4	60
5	35
6	40
7	-10
8	-20
9	200

NOTE: The edges do not need to be chosen in any specific order.



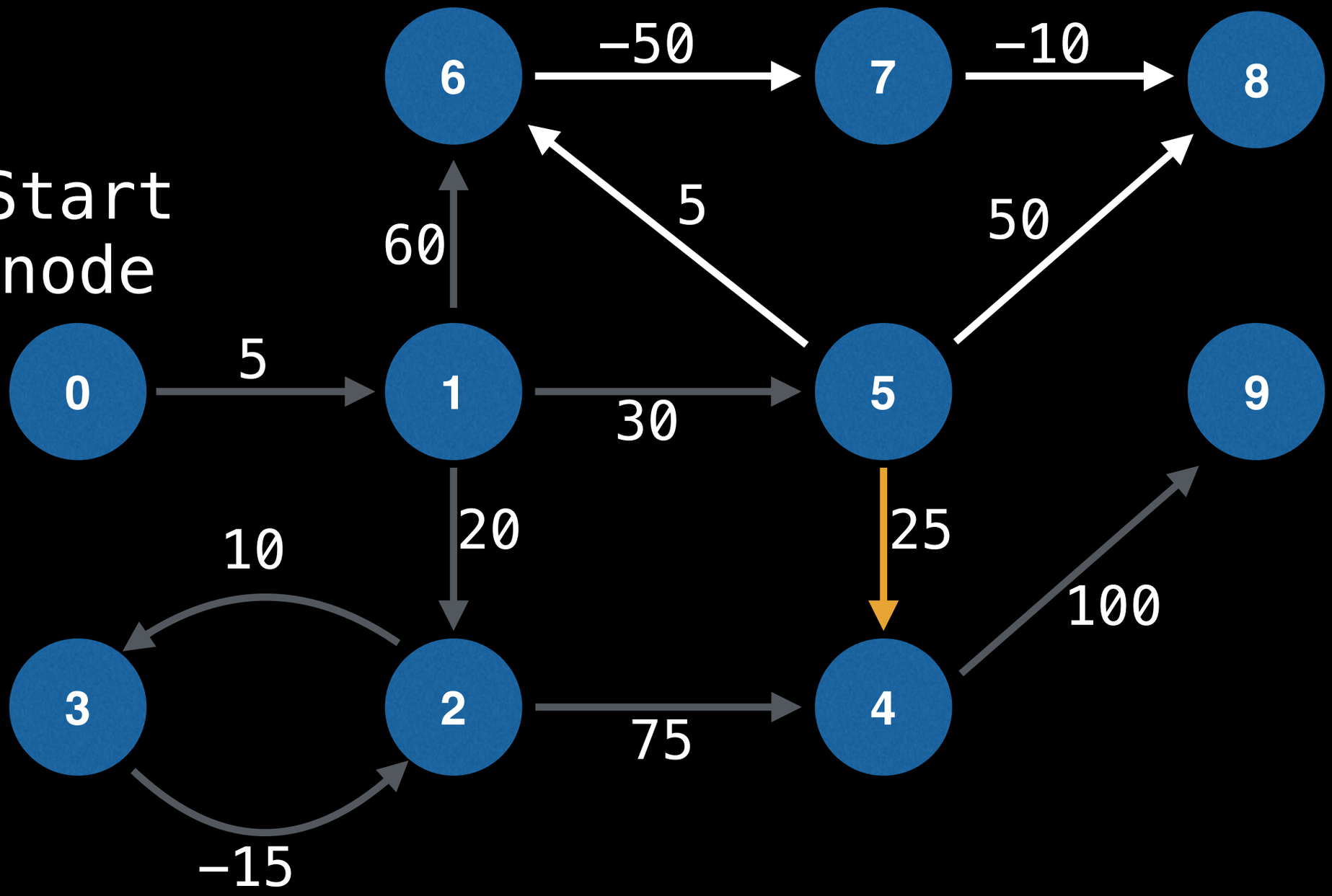
0	0
1	5
2	15
3	30
4	60
5	35
6	40
7	-10
8	-20
9	200

NOTE: The edges do not need to be chosen in any specific order.



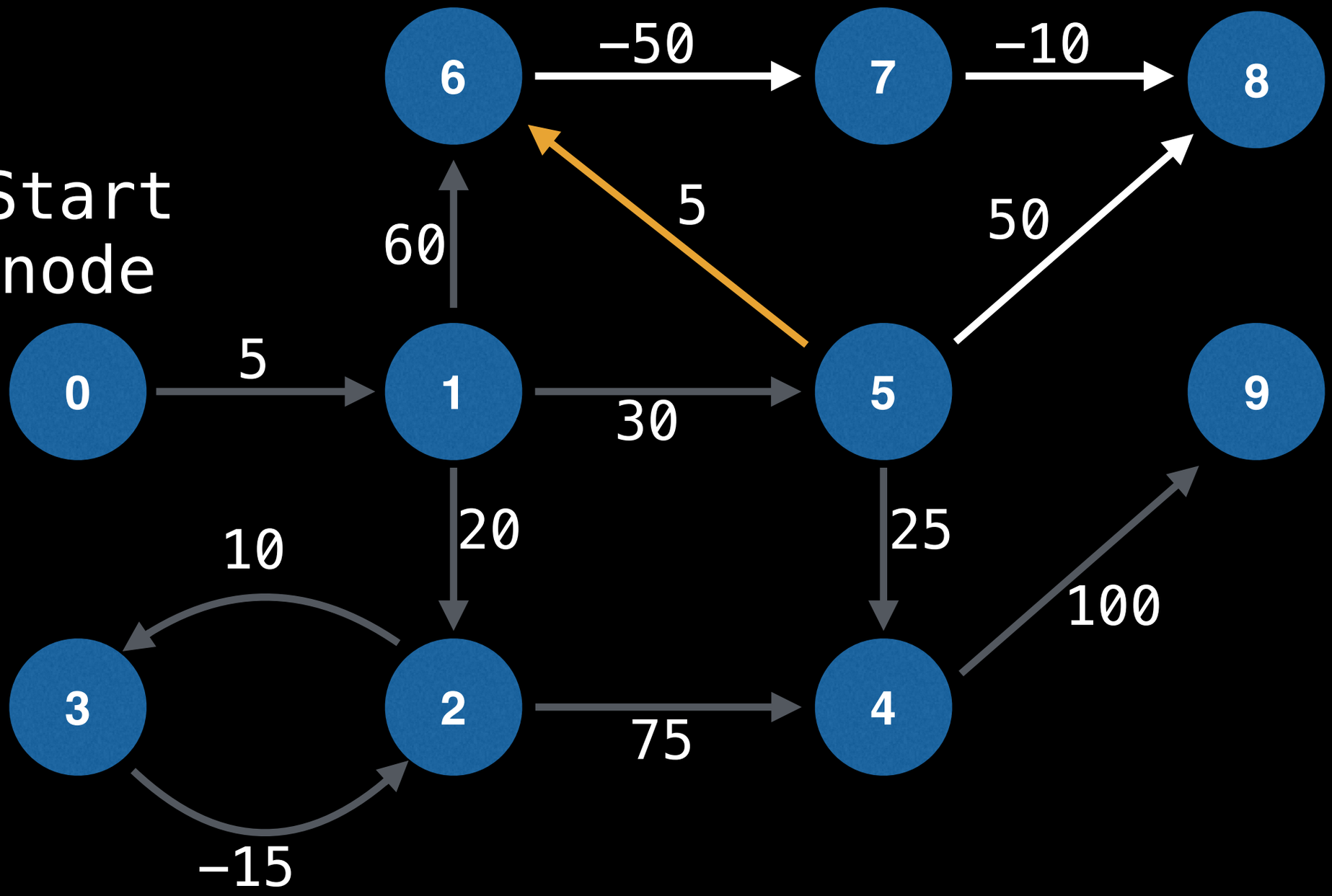
0	0
1	5
2	15
3	30
4	60
5	35
6	40
7	-10
8	-20
9	160

NOTE: The edges do not need to be chosen in any specific order.



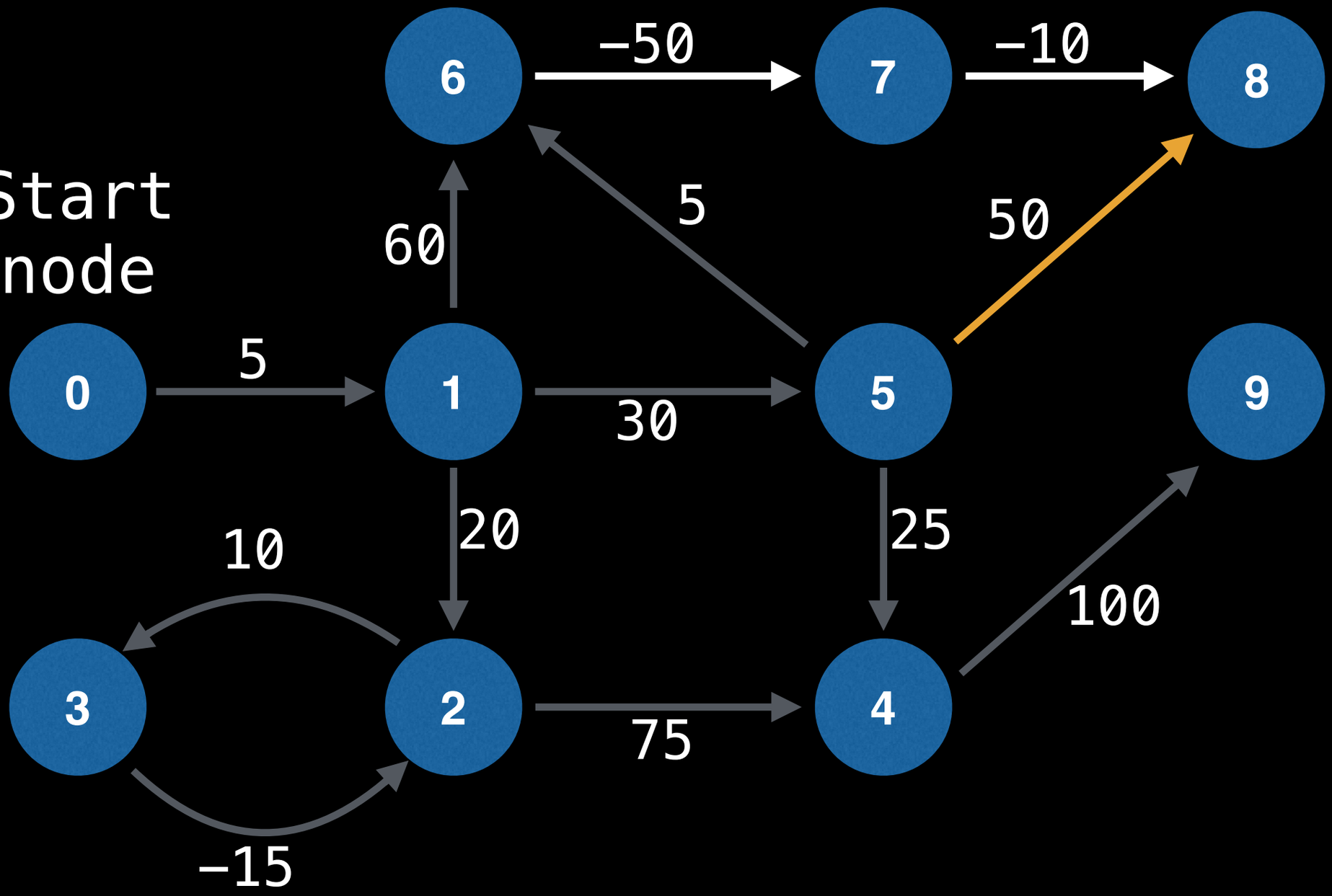
0	0
1	5
2	15
3	30
4	60
5	35
6	40
7	-10
8	-20
9	160

NOTE: The edges do not need to be chosen in any specific order.



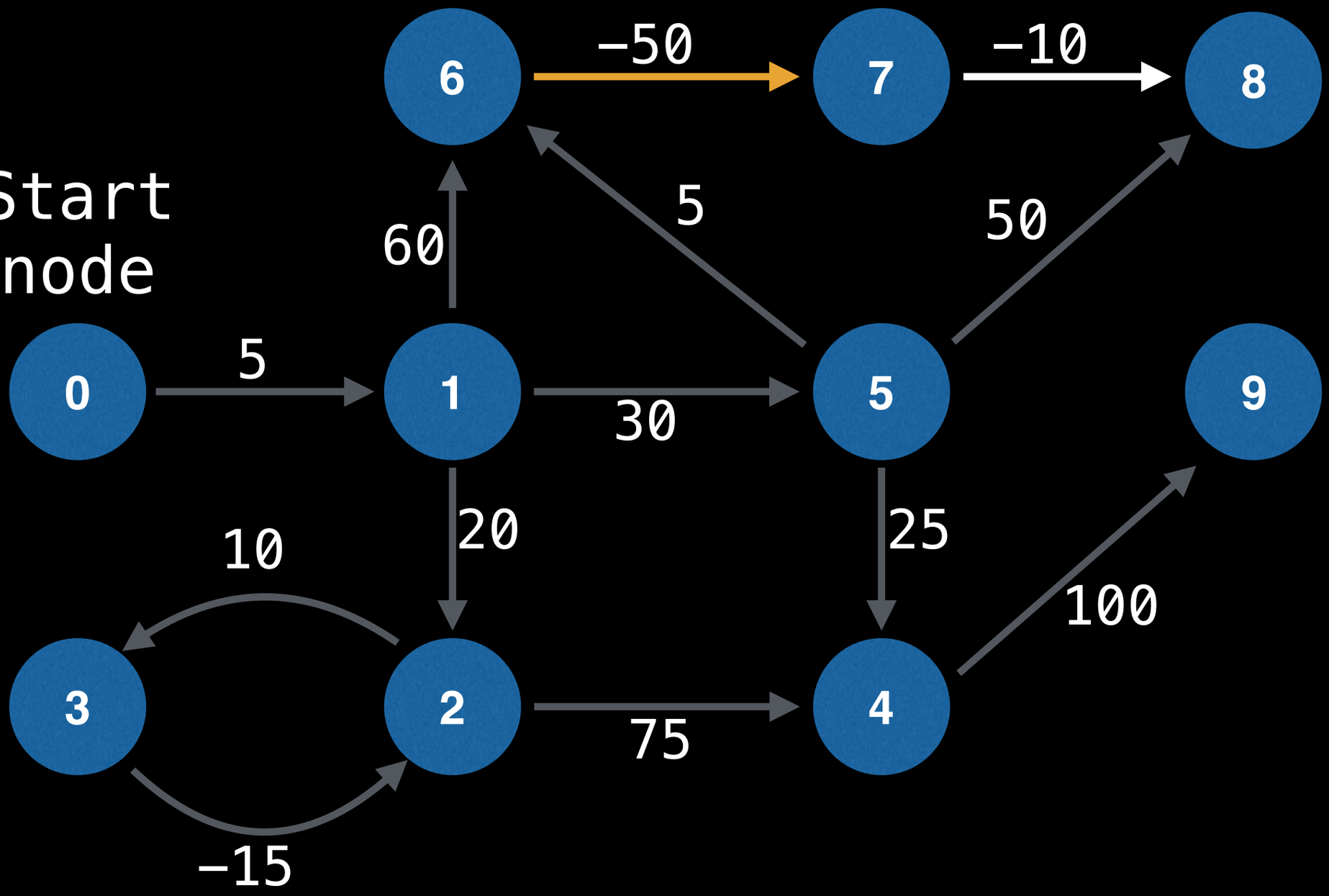
0	0
1	5
2	15
3	30
4	60
5	35
6	40
7	-10
8	-20
9	160

NOTE: The edges do not need to be chosen in any specific order.



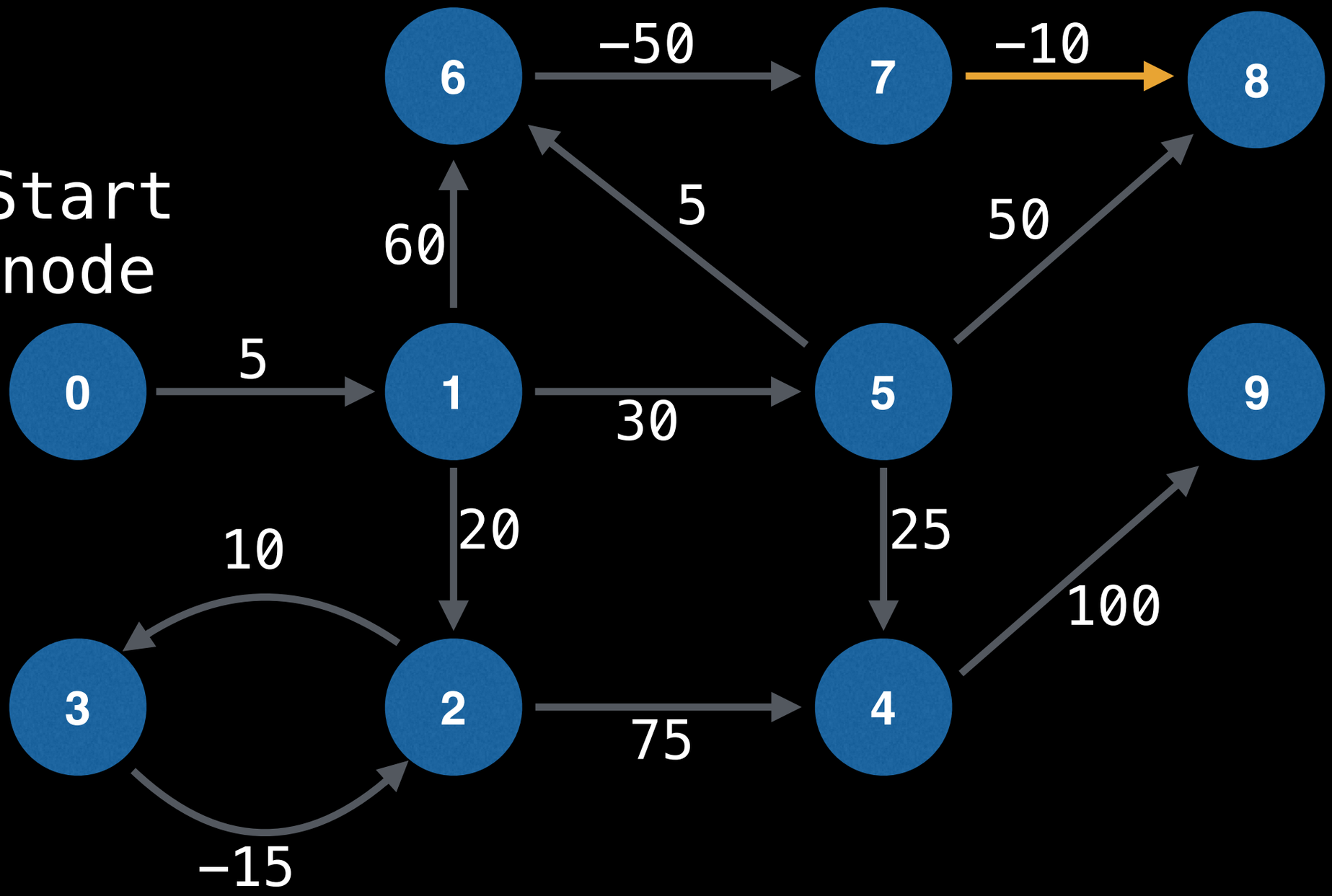
0	0
1	5
2	15
3	30
4	60
5	35
6	40
7	-10
8	-20
9	160

NOTE: The edges do not need to be chosen in any specific order.



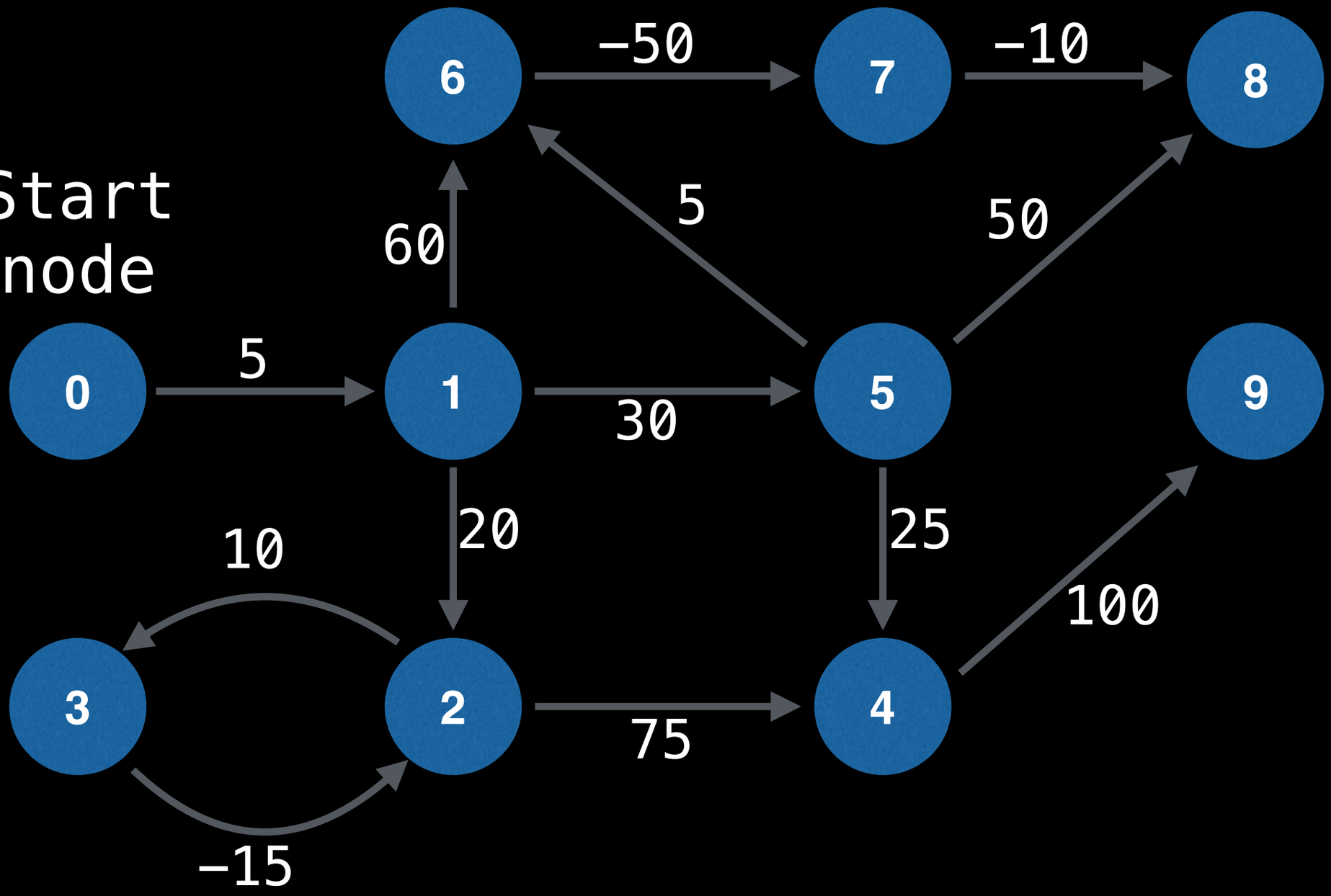
0	0
1	5
2	15
3	30
4	60
5	35
6	40
7	-10
8	-20
9	160

NOTE: The edges do not need to be chosen in any specific order.



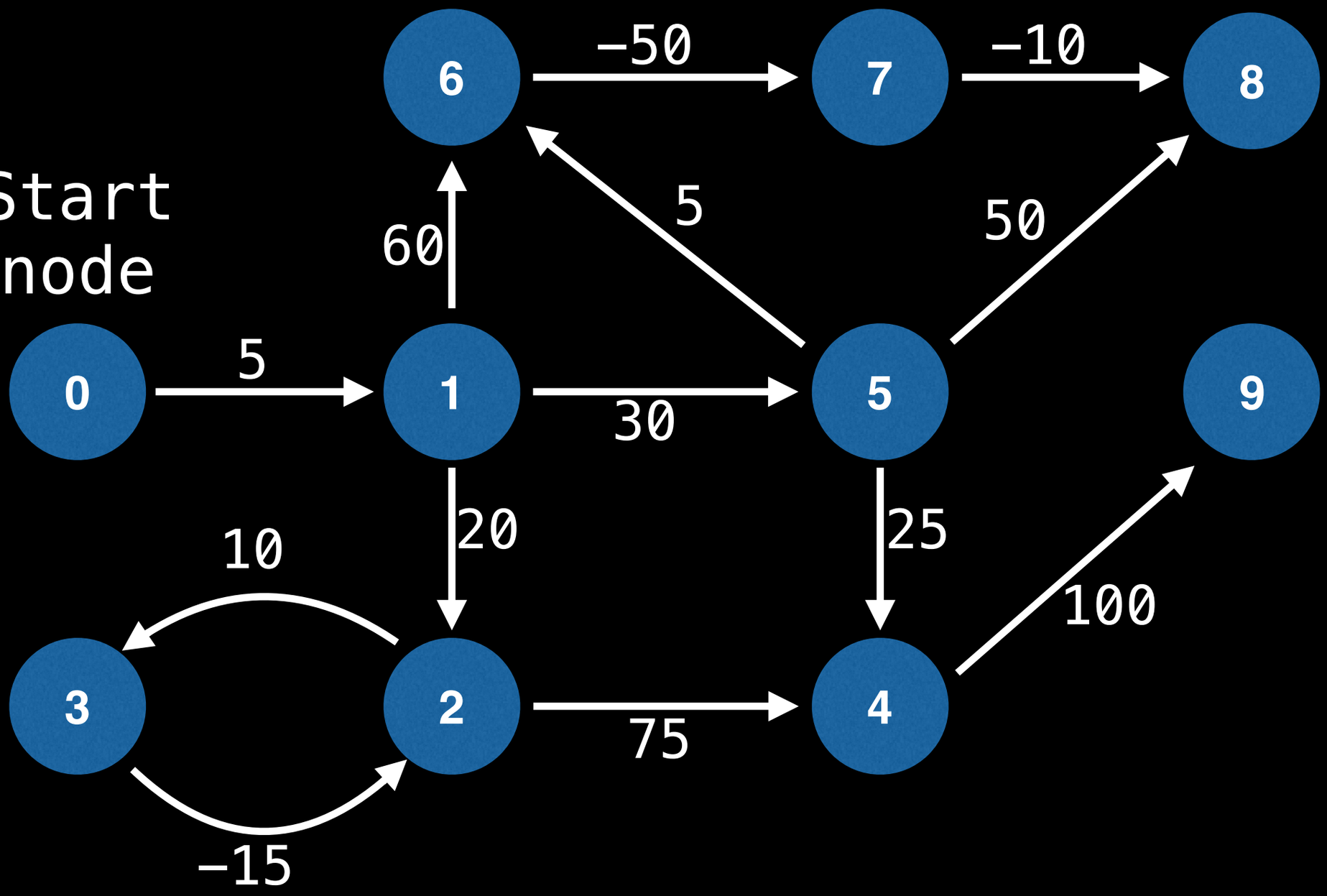
0	0
1	5
2	15
3	30
4	60
5	35
6	40
7	-10
8	-20
9	160

NOTE: The edges do not need to be chosen in any specific order.



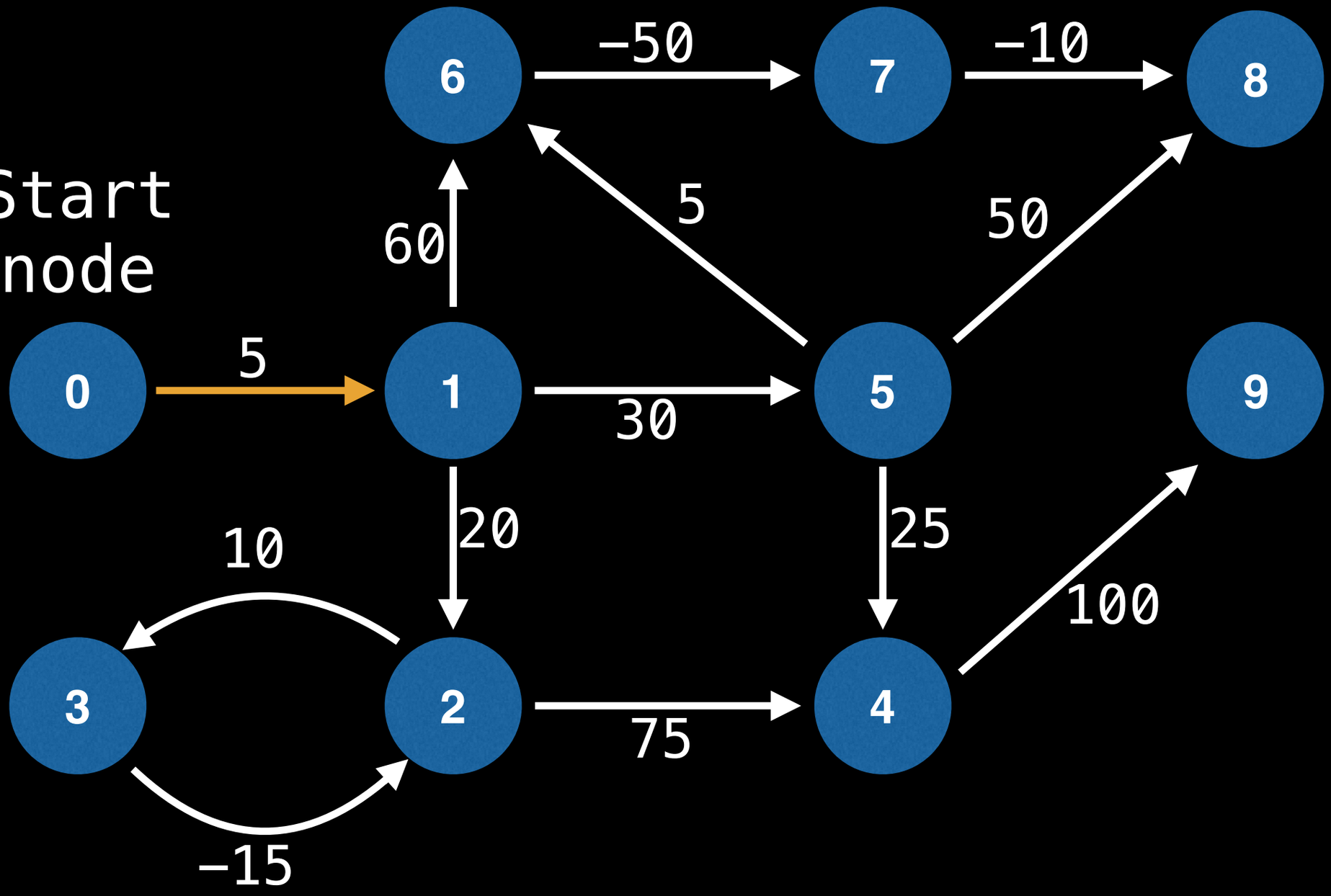
0	0
1	5
2	15
3	30
4	60
5	35
6	40
7	-10
8	-20
9	160

Iteration 2 complete, 7 more to go…
Let’s fast-forward to the end…



0	0
1	5
2	-20
3	-5
4	60
5	35
6	40
7	-10
8	-20
9	160

We’re finished with the SSSP part. Now let’s detect those negative cycles. If we can relax an edge then there’s a negative cycle.

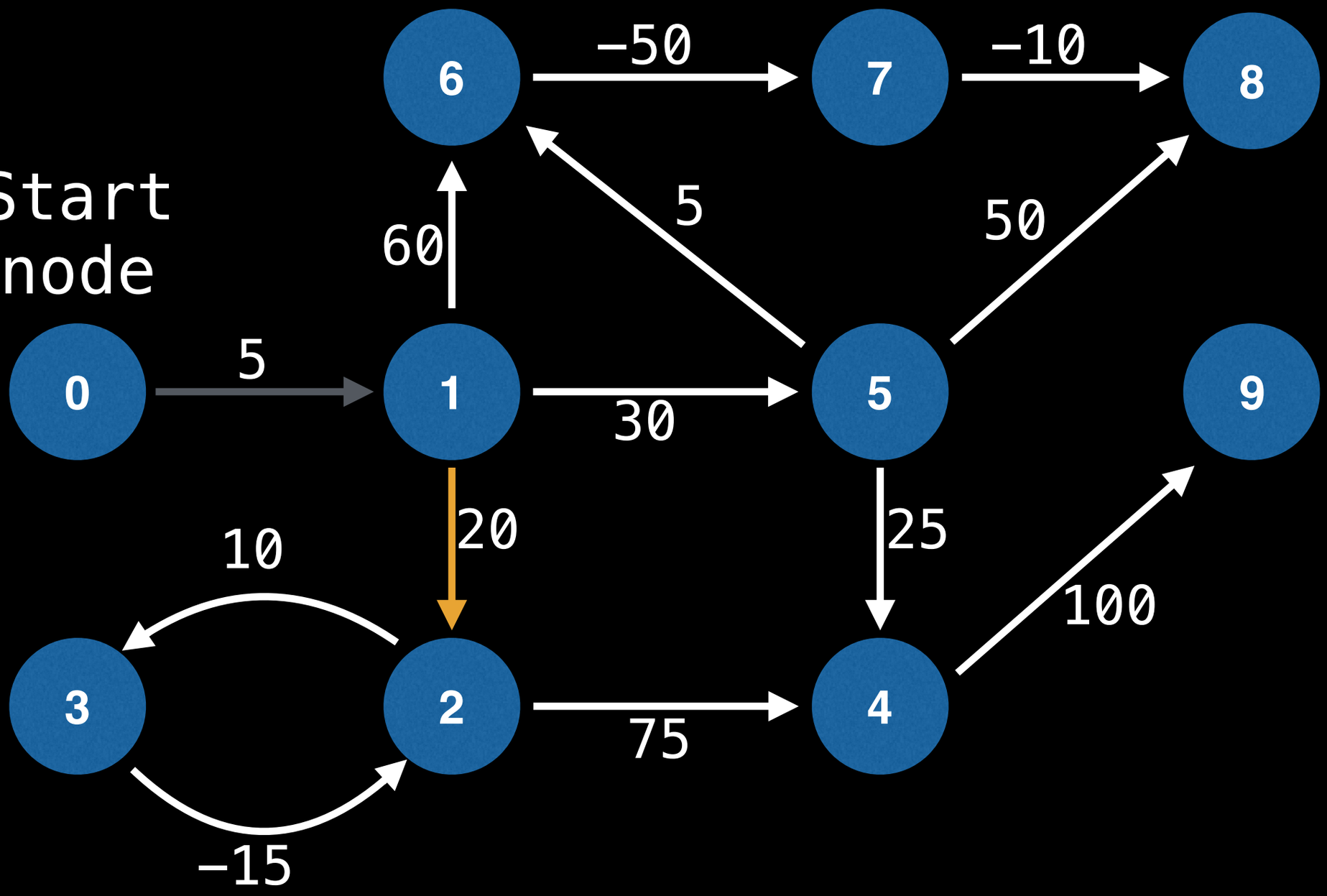


0	0
1	5
2	-20
3	-5
4	60
5	35
6	40
7	-10
8	-20
9	160

Unaffected node

Directly in negative cycle

Reachable by negative cycle

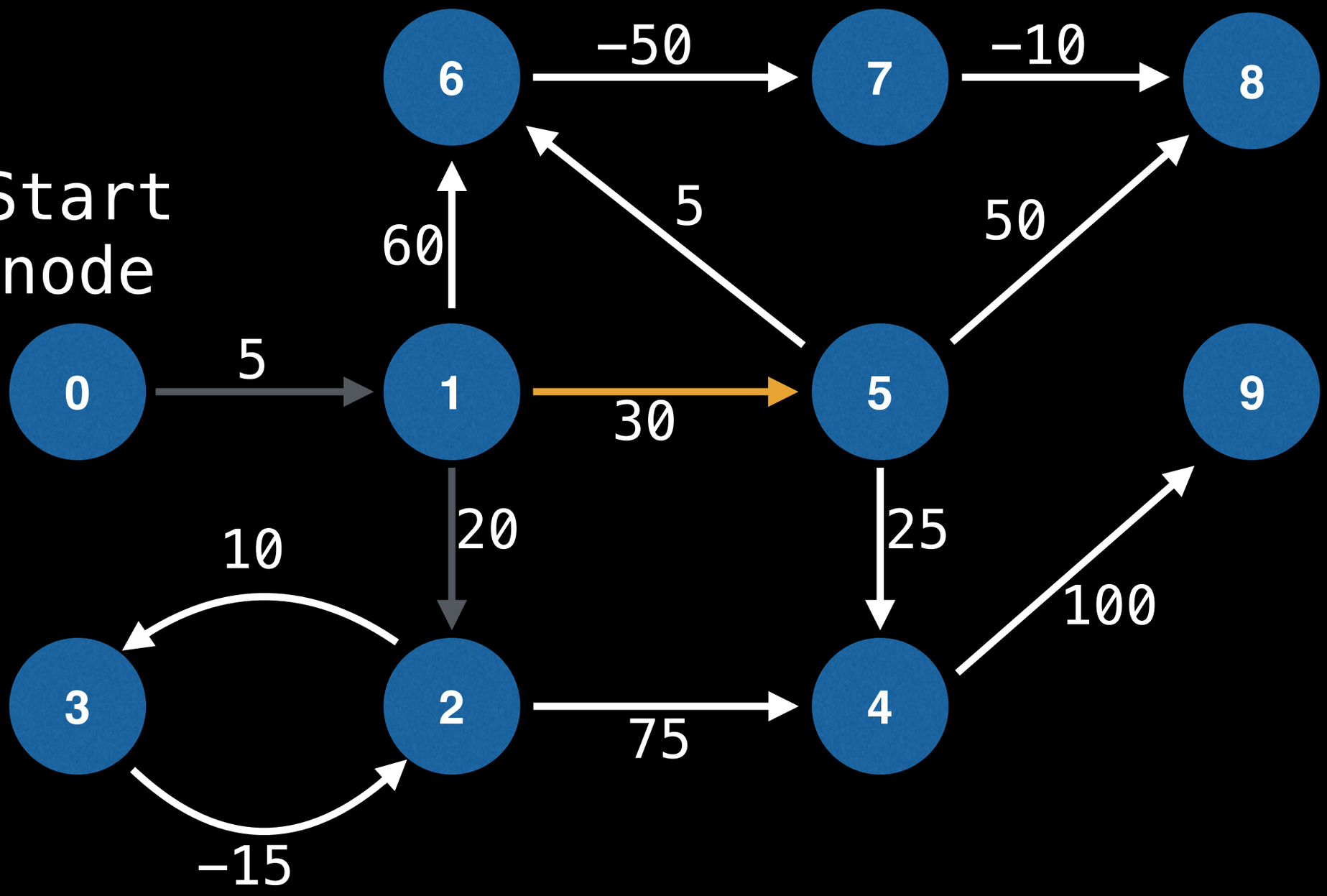


0	0
1	5
2	-20
3	-5
4	60
5	35
6	40
7	-10
8	-20
9	160

Unaffected node

Directly in negative cycle

Reachable by negative cycle

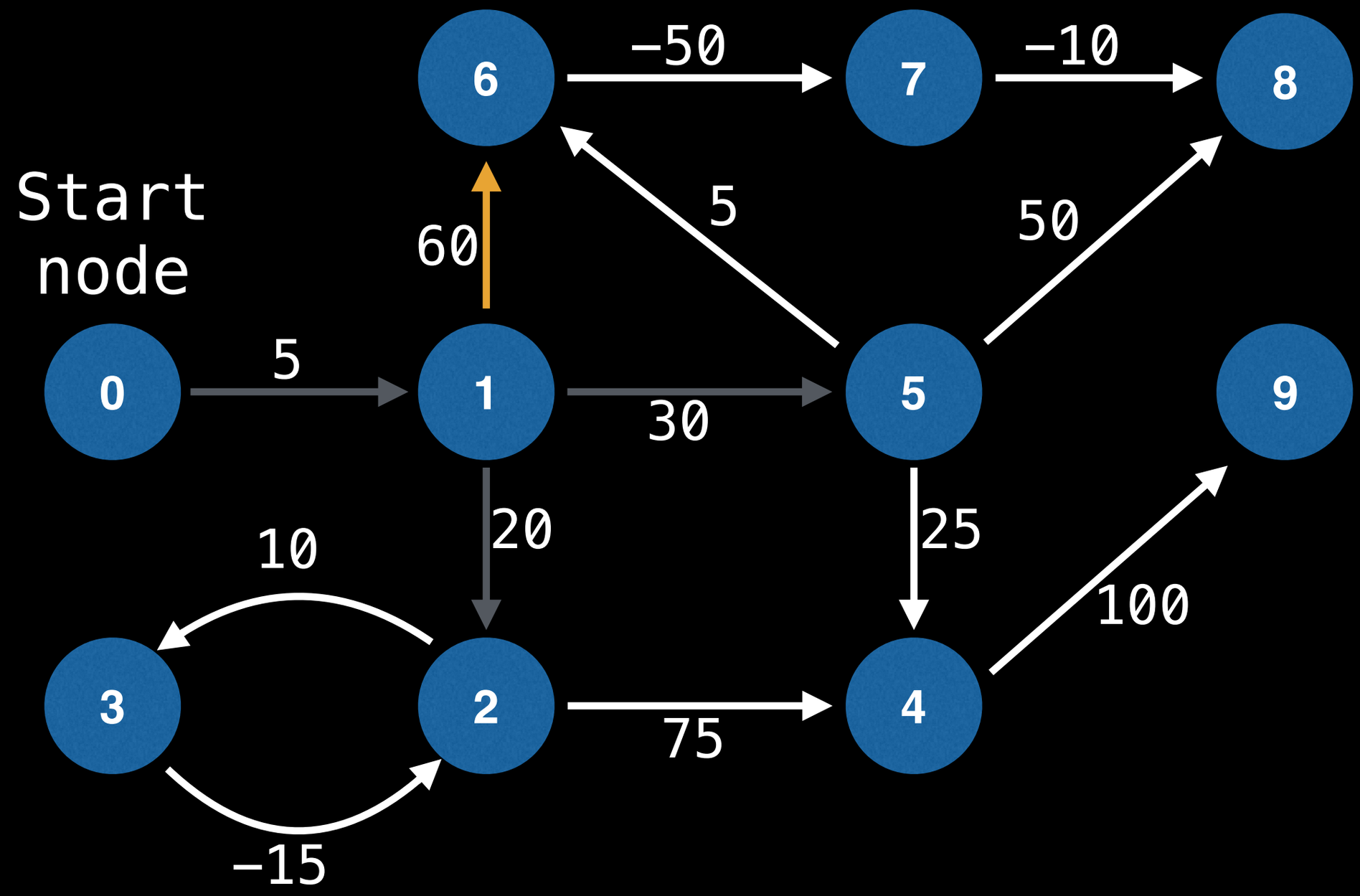


0	0
1	5
2	-20
3	-5
4	60
5	35
6	40
7	-10
8	-20
9	160

Unaffected node

Directly in negative cycle

Reachable by negative cycle

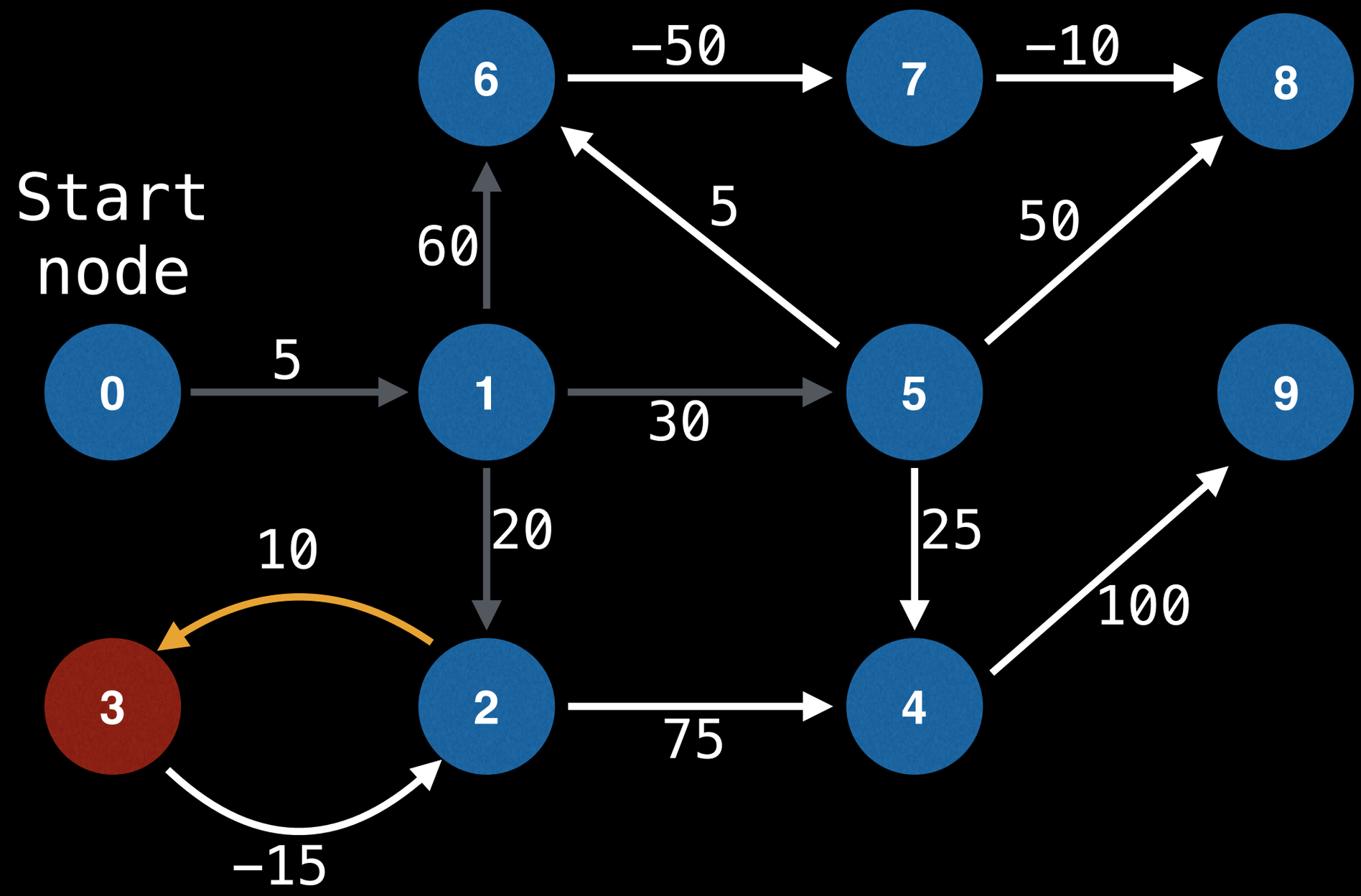


0	0
1	5
2	-20
3	-5
4	60
5	35
6	40
7	-10
8	-20
9	160

Unaffected node

Directly in negative cycle

Reachable by negative cycle

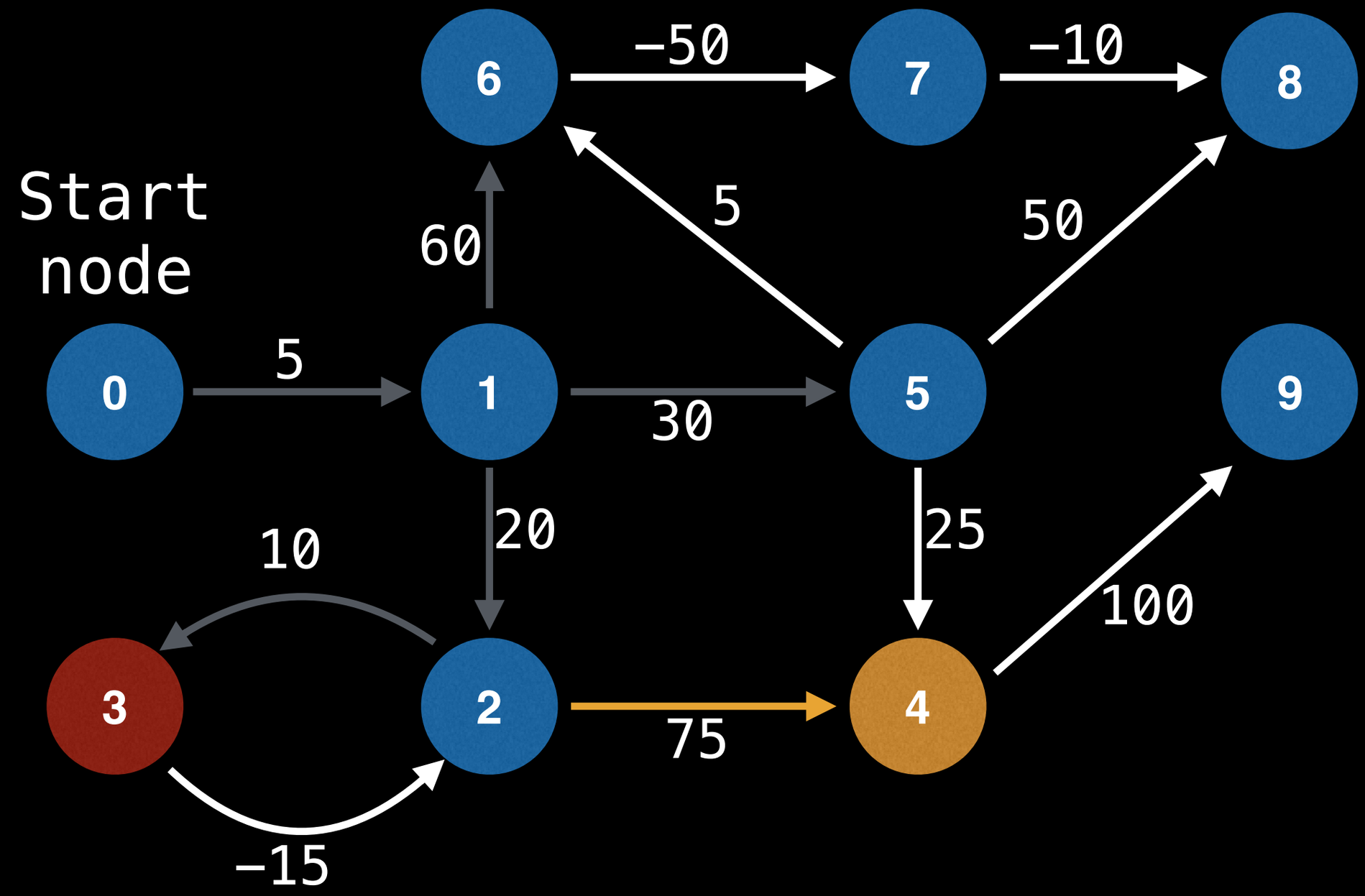


0	0
1	5
2	-20
3	-∞
4	60
5	35
6	40
7	-10
8	-20
9	160

Unaffected node

Directly in negative cycle

Reachable by negative cycle

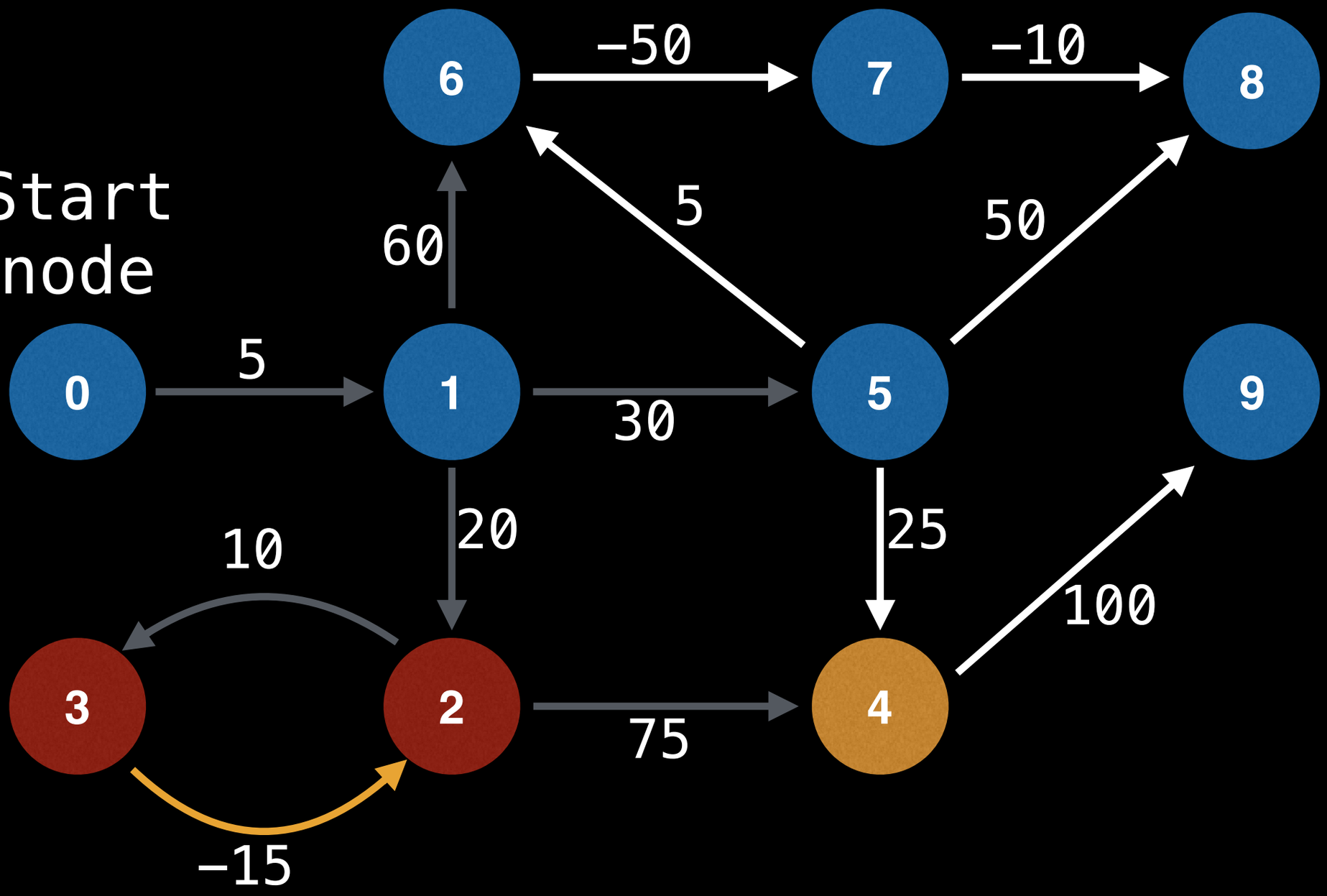


0	0
1	5
2	-20
3	-∞
4	-∞
5	35
6	40
7	-10
8	-20
9	160

Unaffected node

Directly in negative cycle

Reachable by negative cycle

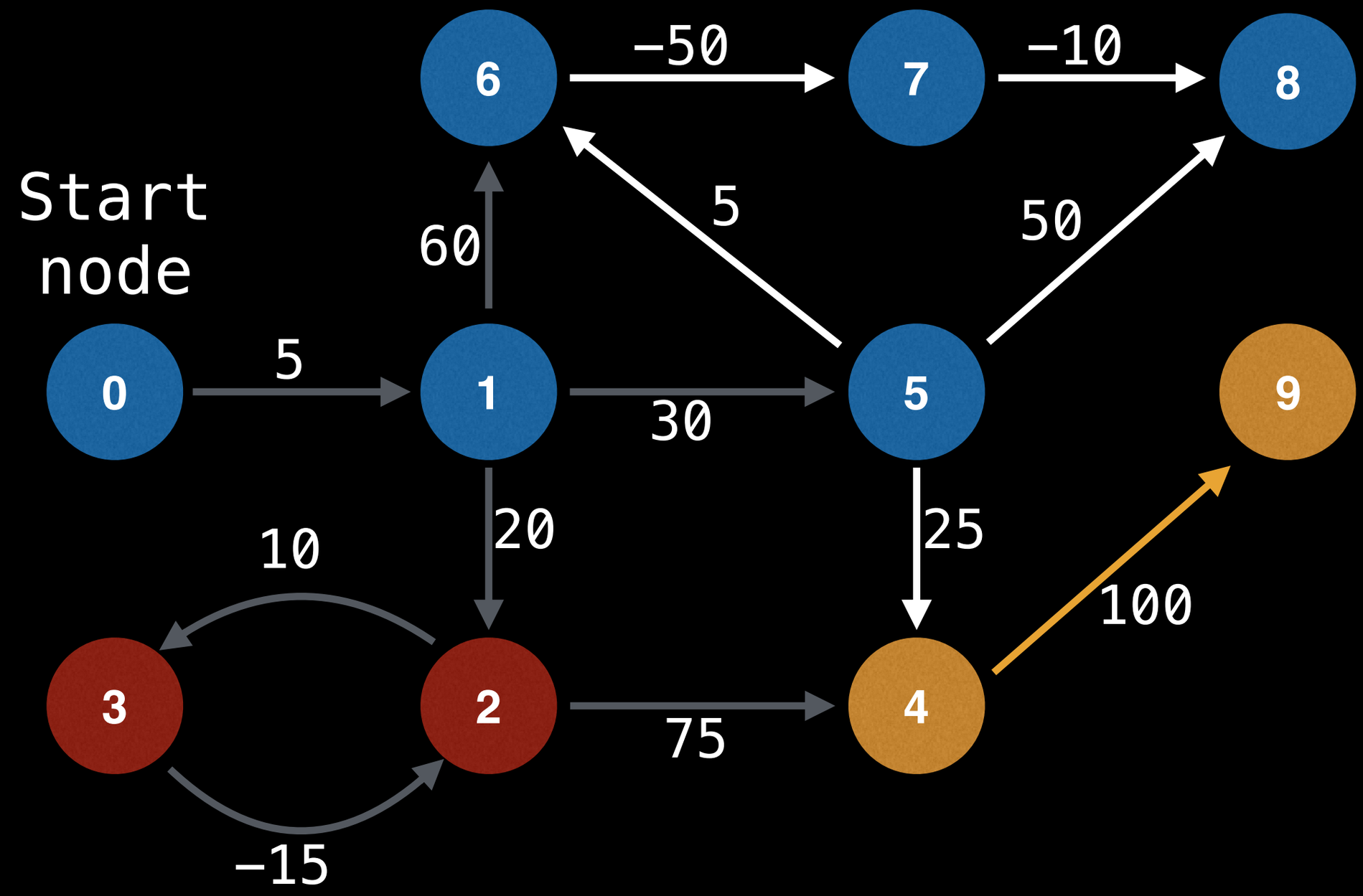


0	0
1	5
2	-∞
3	-∞
4	-∞
5	35
6	40
7	-10
8	-20
9	160

Unaffected node

Directly in negative cycle

Reachable by negative cycle

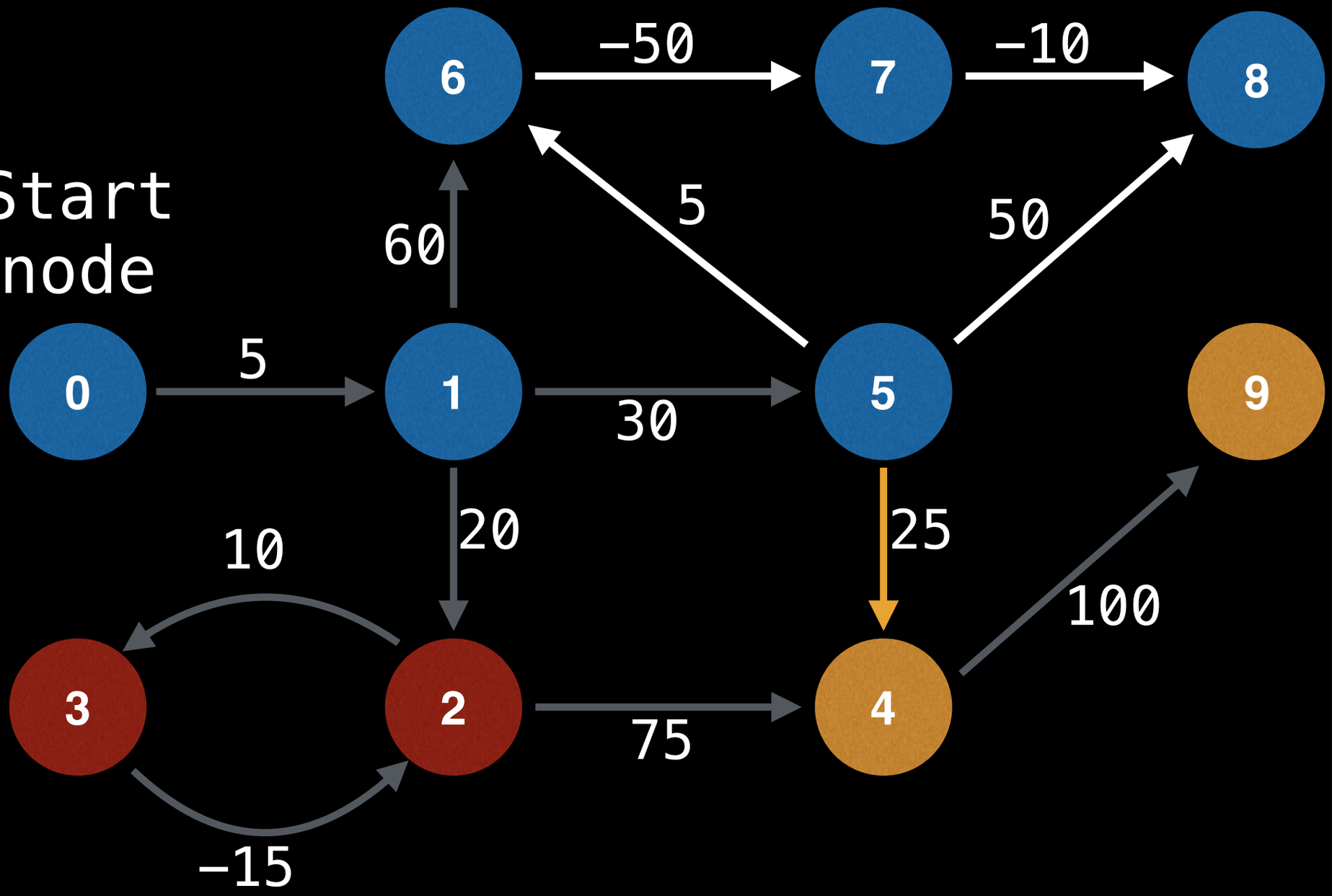


0	0
1	5
2	-∞
3	-∞
4	-∞
5	35
6	40
7	-10
8	-20
9	-∞

Unaffected node

Directly in negative cycle

Reachable by negative cycle

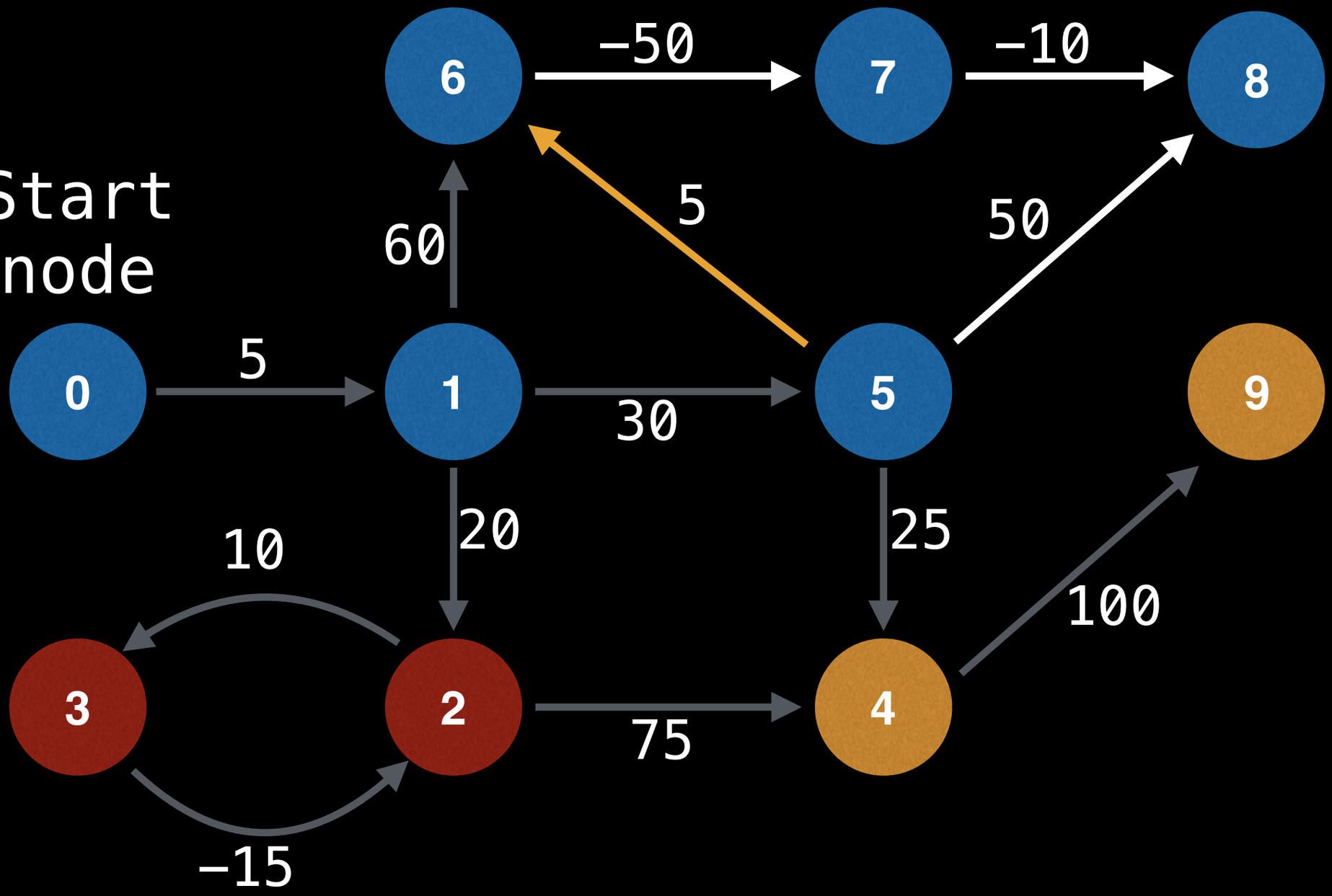


0	0
1	5
2	-∞
3	-∞
4	-∞
5	35
6	40
7	-10
8	-20
9	-∞

Unaffected node

Directly in negative cycle

Reachable by negative cycle

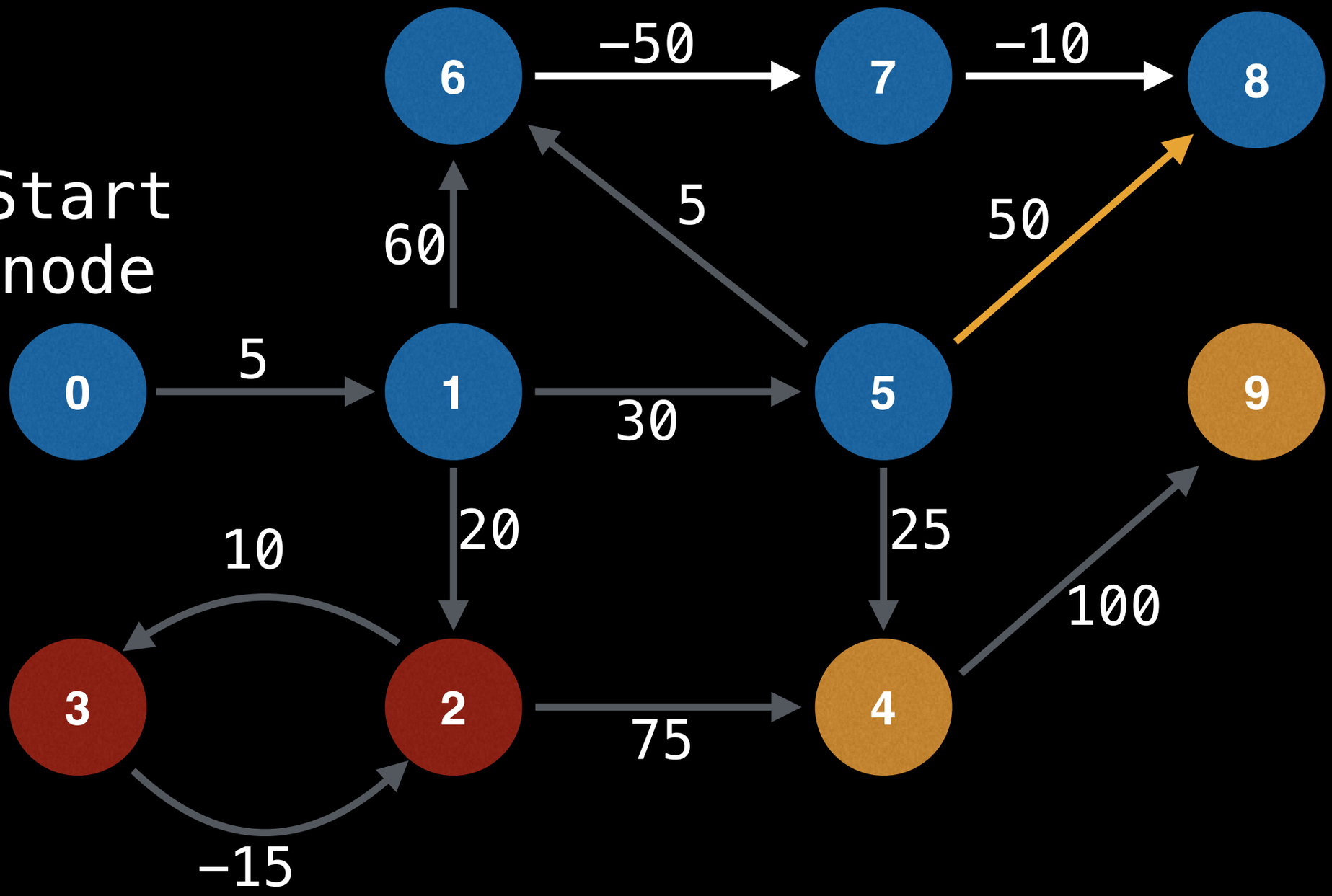


0	0
1	5
2	-∞
3	-∞
4	-∞
5	35
6	40
7	-10
8	-20
9	-∞

Unaffected node

Directly in negative cycle

Reachable by negative cycle

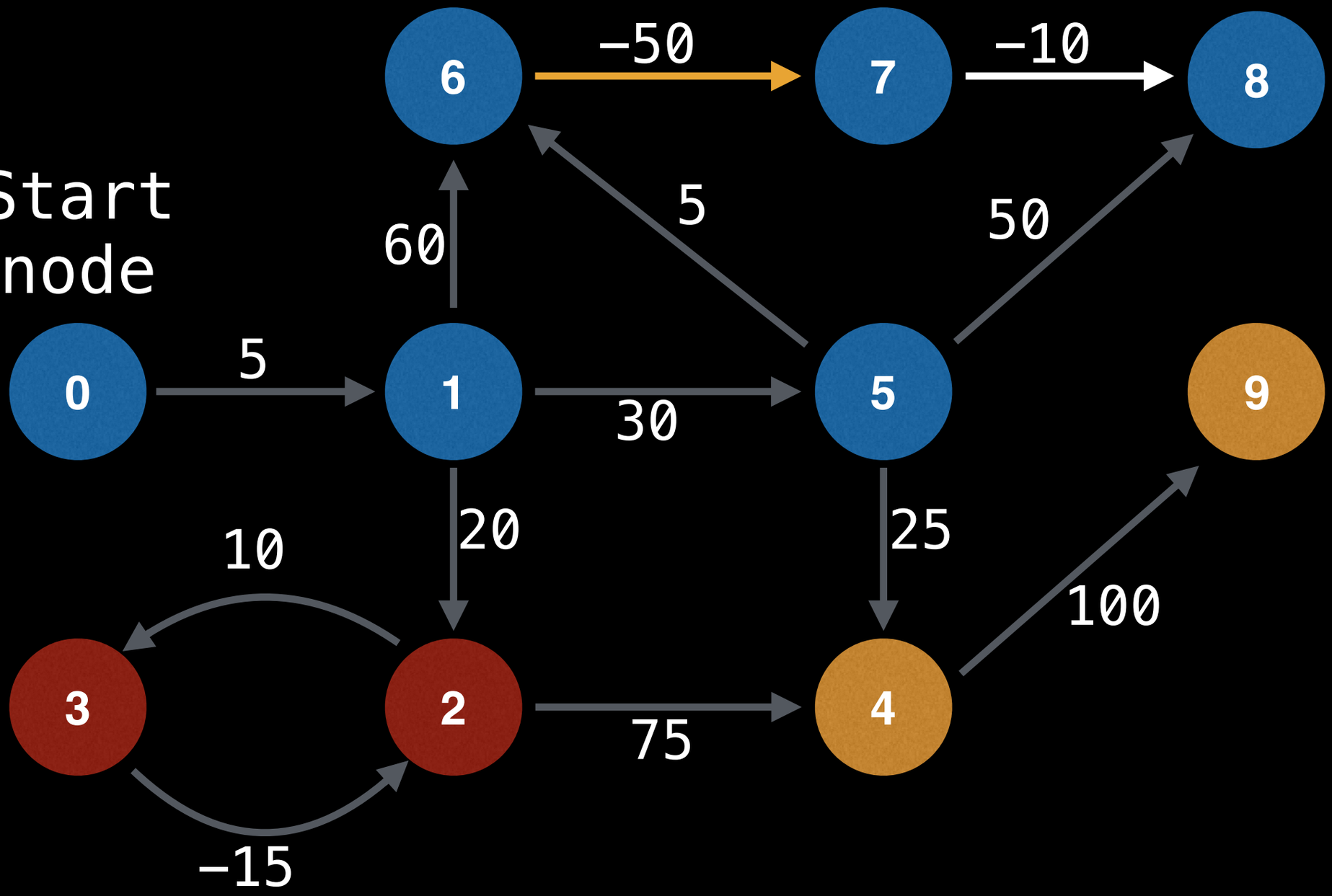


0	0
1	5
2	-∞
3	-∞
4	-∞
5	35
6	40
7	-10
8	-20
9	-∞

Unaffected node

Directly in negative cycle

Reachable by negative cycle

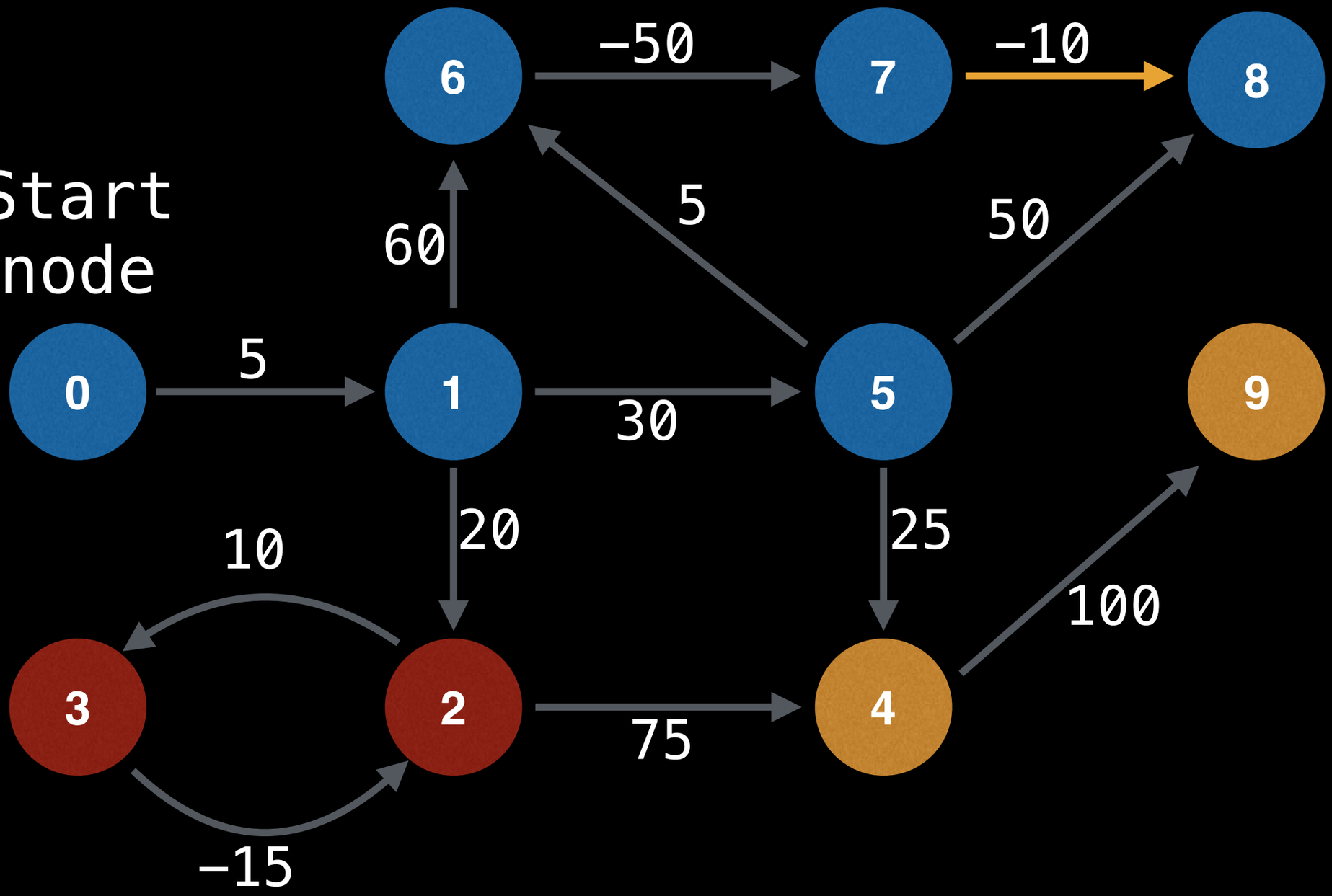


0	0
1	5
2	-∞
3	-∞
4	-∞
5	35
6	40
7	-10
8	-20
9	-∞

Unaffected node

Directly in negative cycle

Reachable by negative cycle

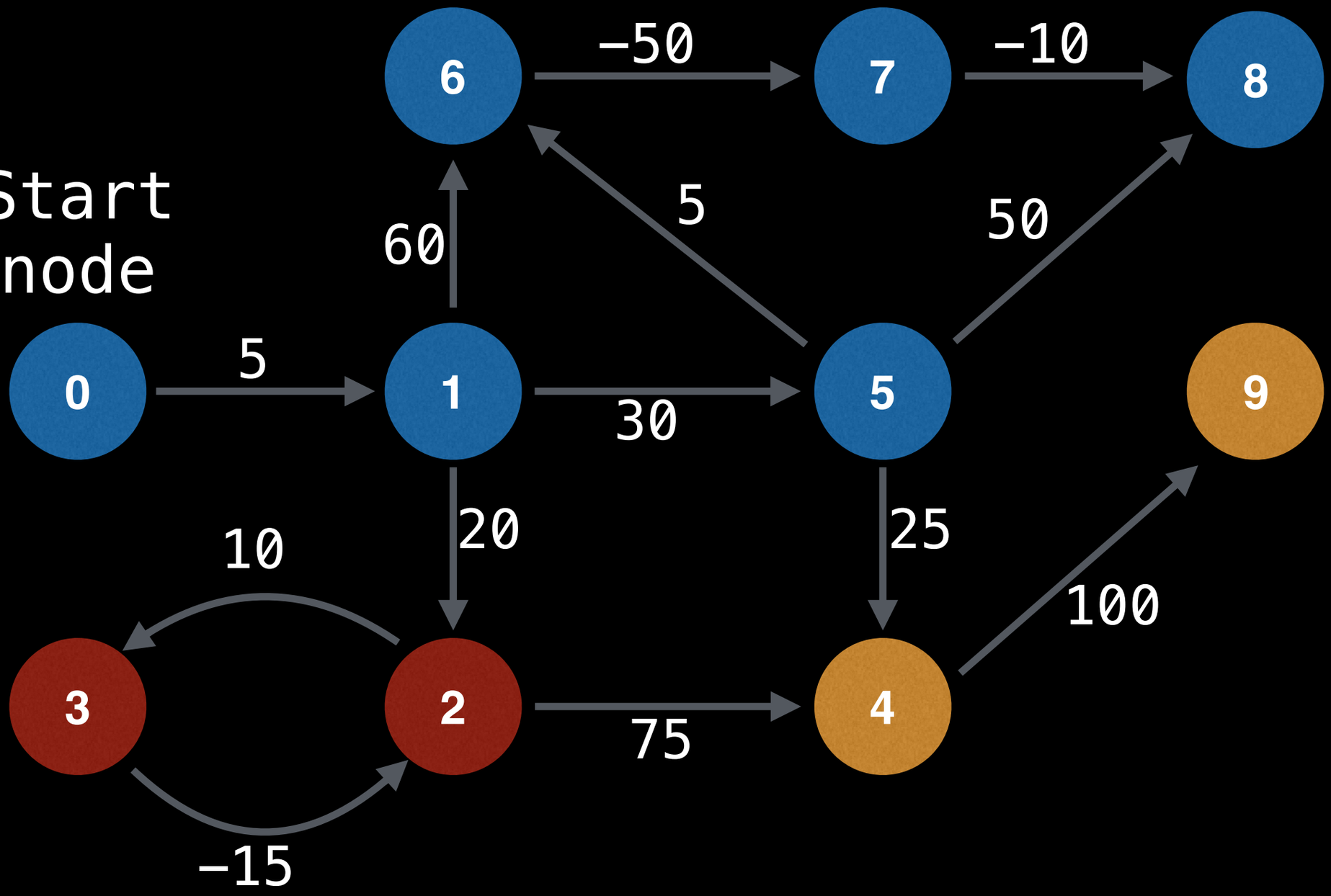


0	0
1	5
2	-∞
3	-∞
4	-∞
5	35
6	40
7	-10
8	-20
9	-∞

Unaffected node

Directly in negative cycle

Reachable by negative cycle



0	0
1	5
2	-∞
3	-∞
4	-∞
5	35
6	40
7	-10
8	-20
9	-∞

Repeat this for another 8 iterations in order to ensure the cycles fully propagate. In this example, we happened to detect all cycles on the first iteration, but this was a coincidence.