Prim's Minimum Spanning Tree Algorithm

(lazy version)

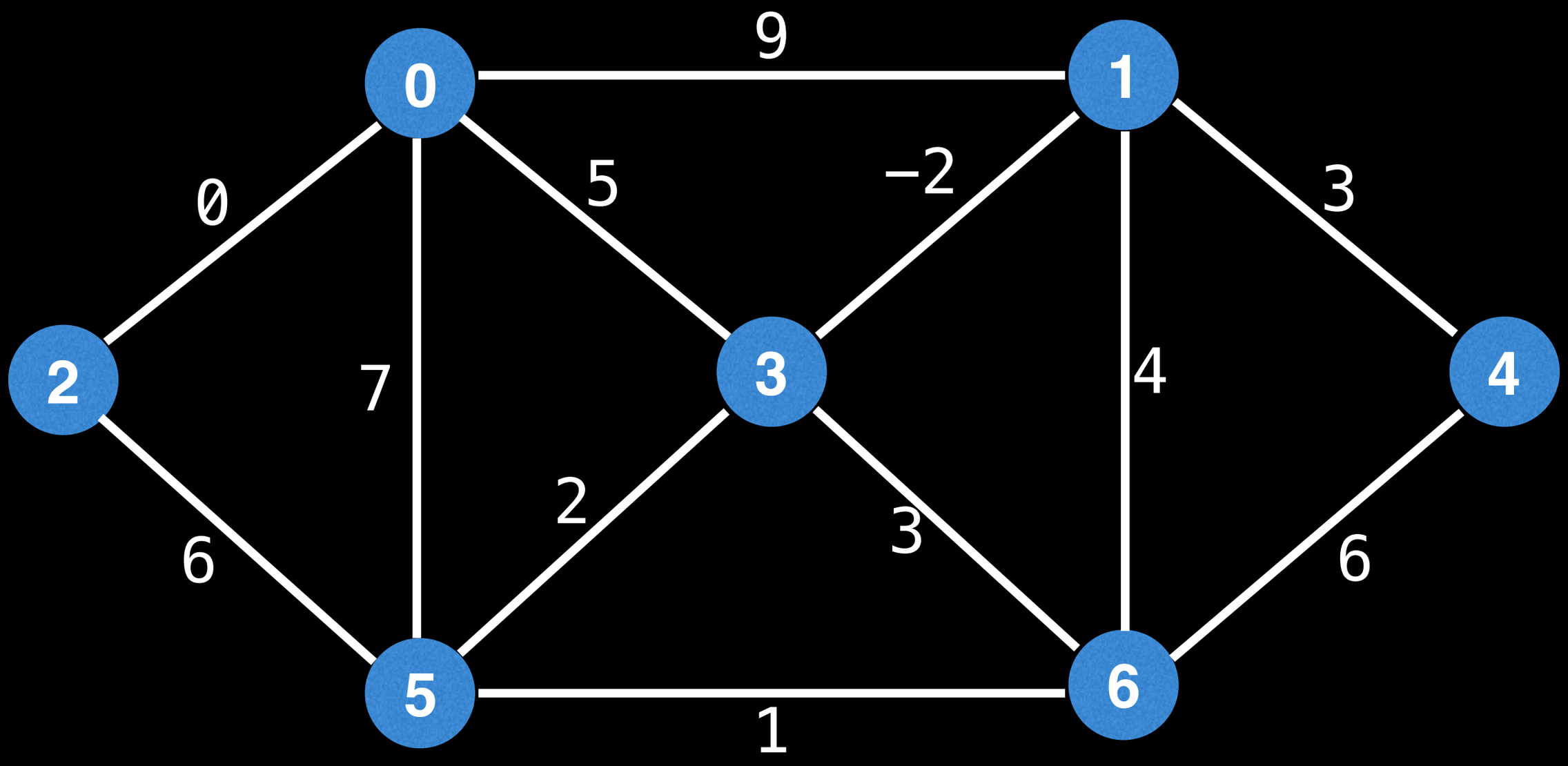
William Fiset

What is a Minimum Spanning Tree?

Given an **undirected graph** with weighted edges, a **Minimum Spanning Tree** (MST) is a subset of the edges in the graph which connects all vertices together (without creating any cycles) while minimizing the total edge cost.

What is a Minimum Spanning Tree?

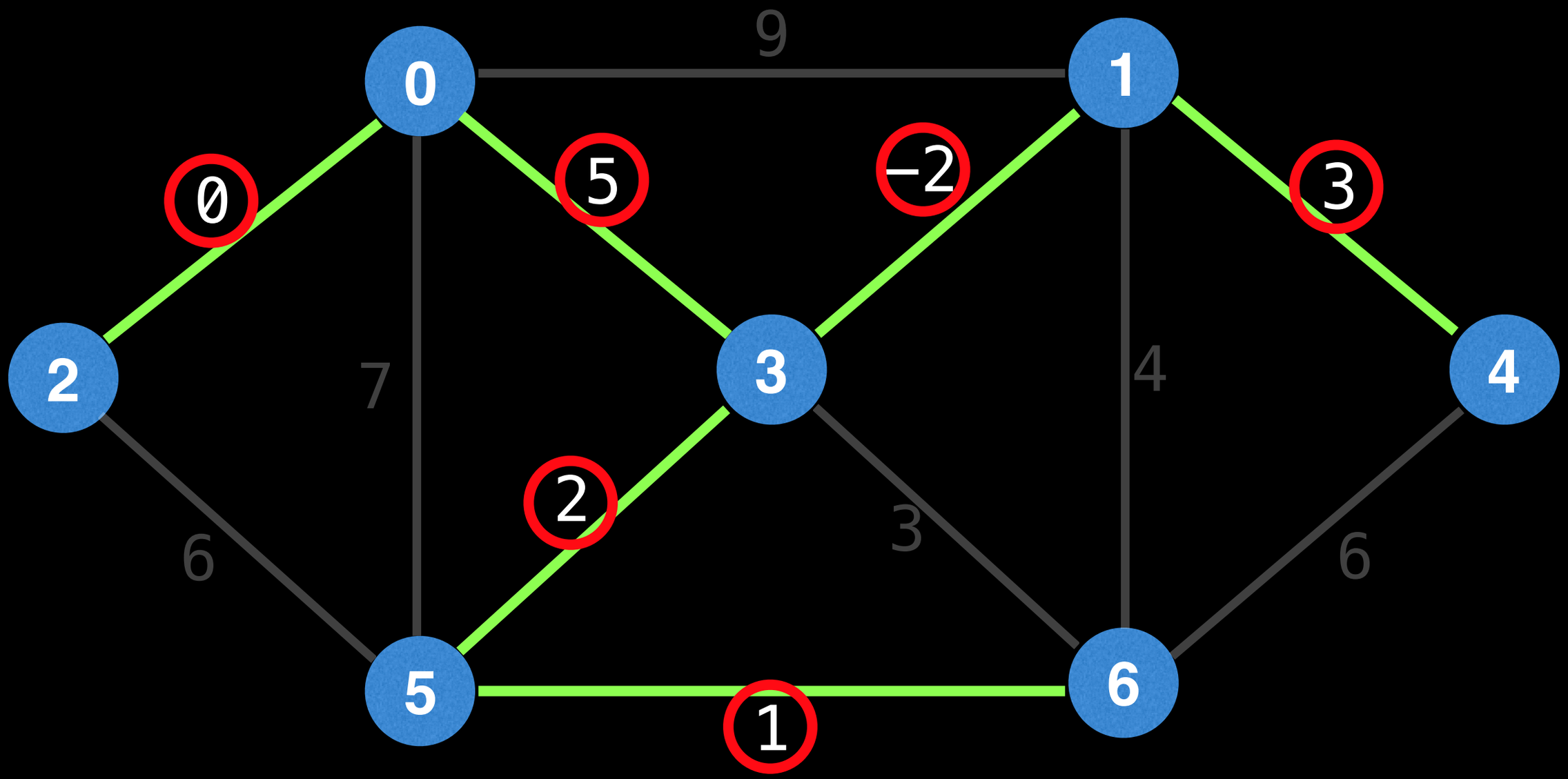
Given an **undirected graph** with weighted edges, a **Minimum Spanning Tree** (MST) is a subset of the edges in the graph which connects all vertices together (without creating any cycles) while minimizing the total edge cost.



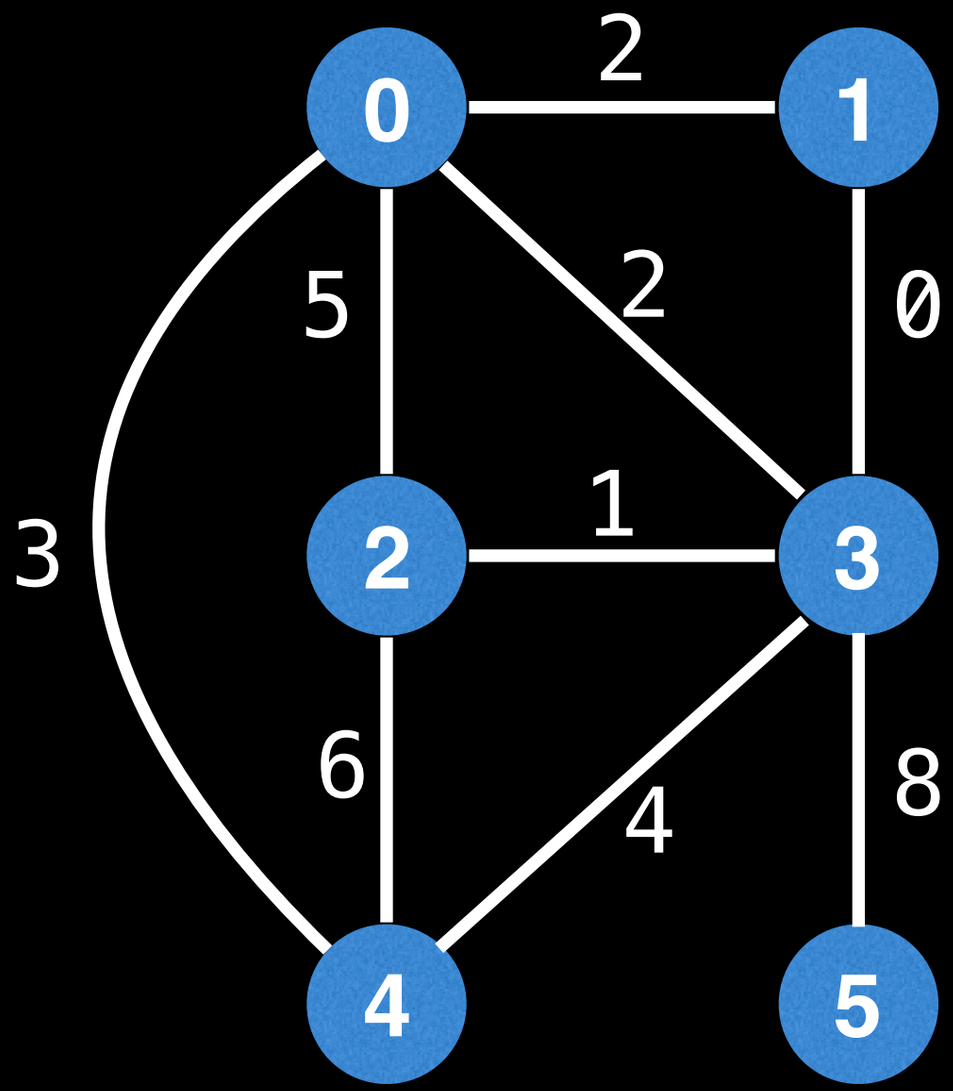
What is a Minimum Spanning Tree?

This particular graph has a unique MST highlighted in green. However, it is common for a graph to have multiple valid MSTs of equal costs.

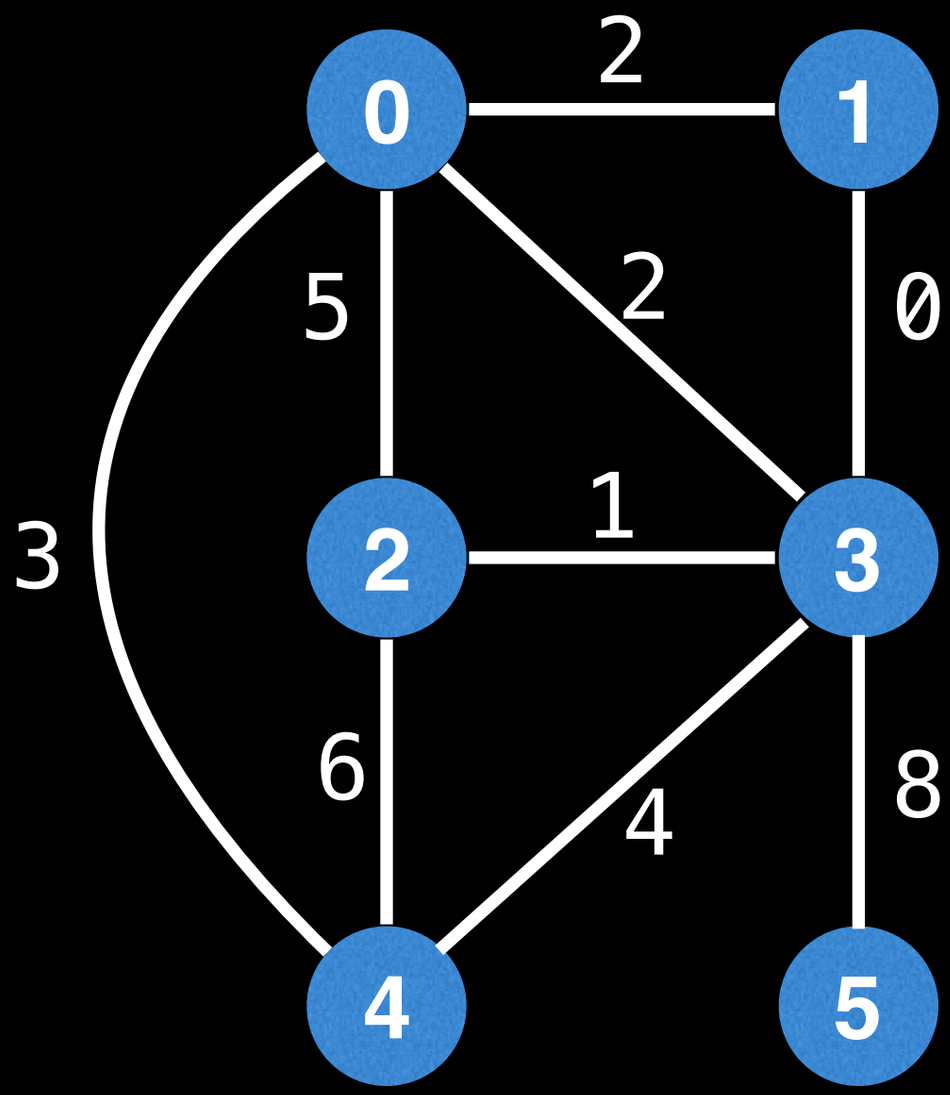
MST cost: 0 + 5 + −2 + 2 + 1 + 3 = 9

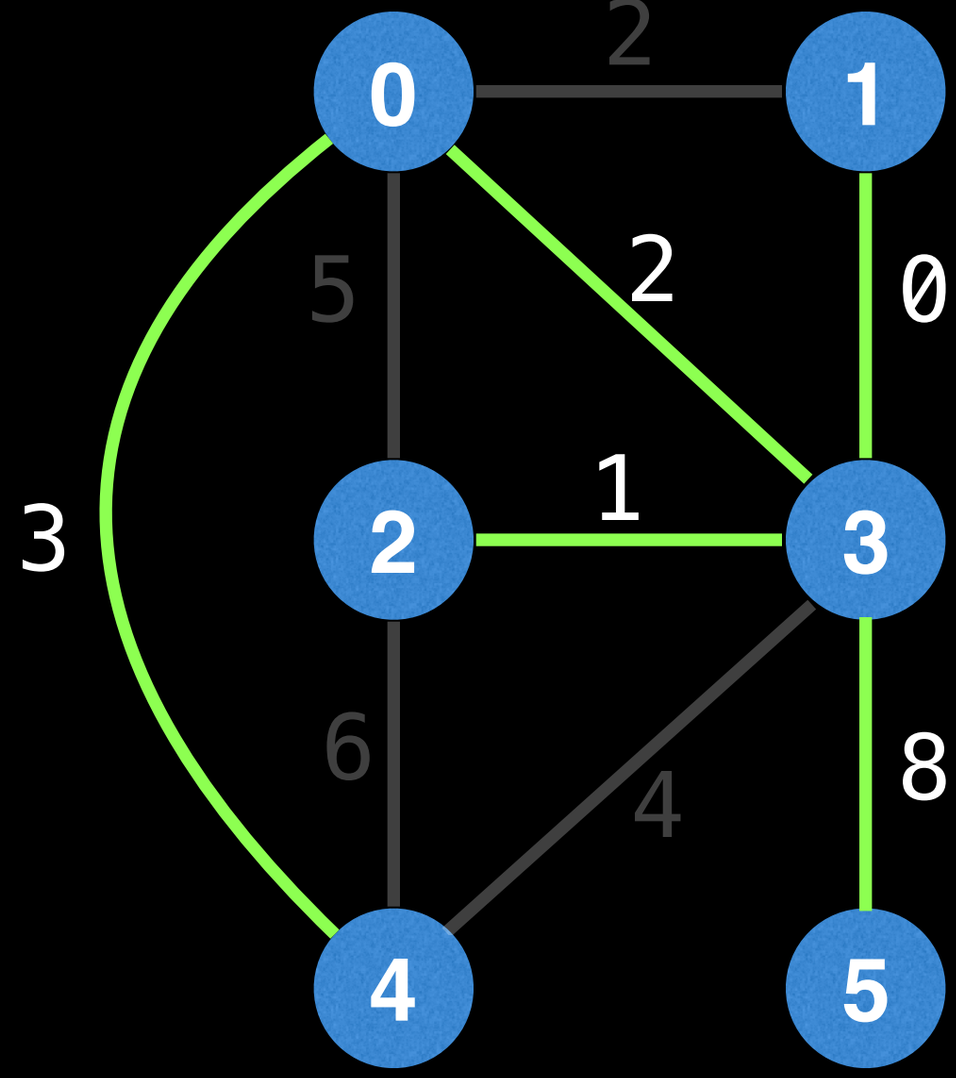


Find any MST



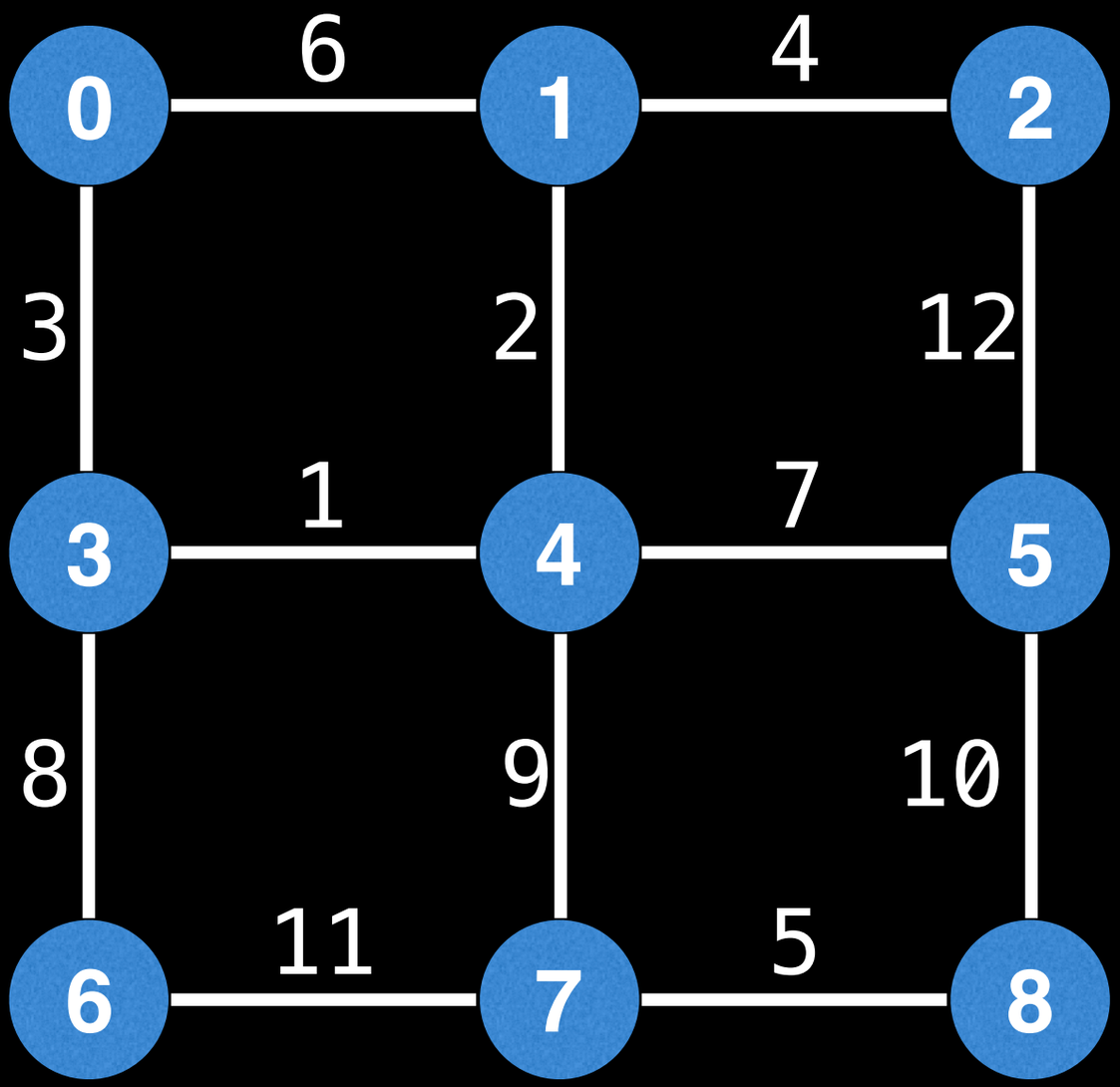
Find any MST



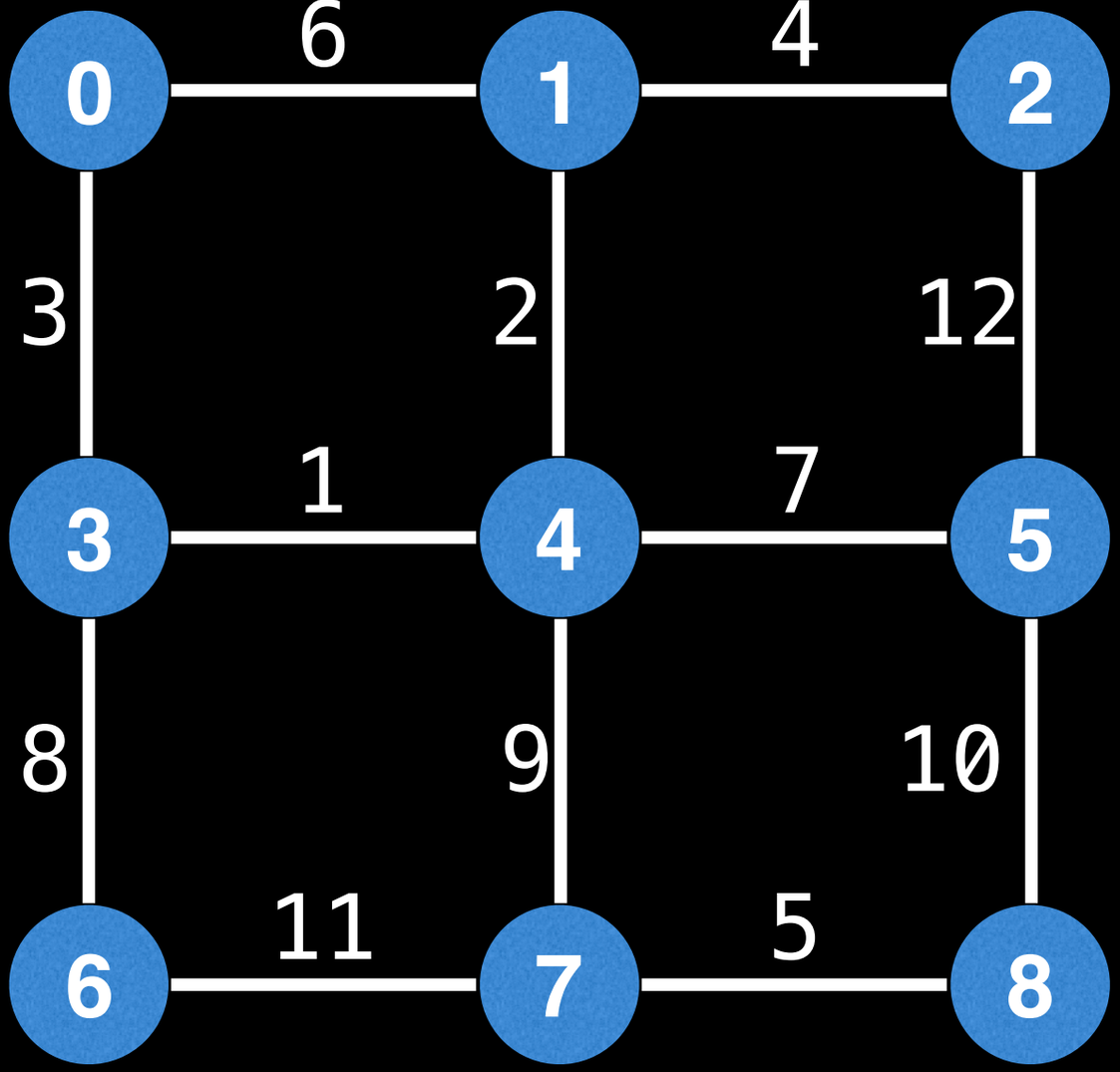


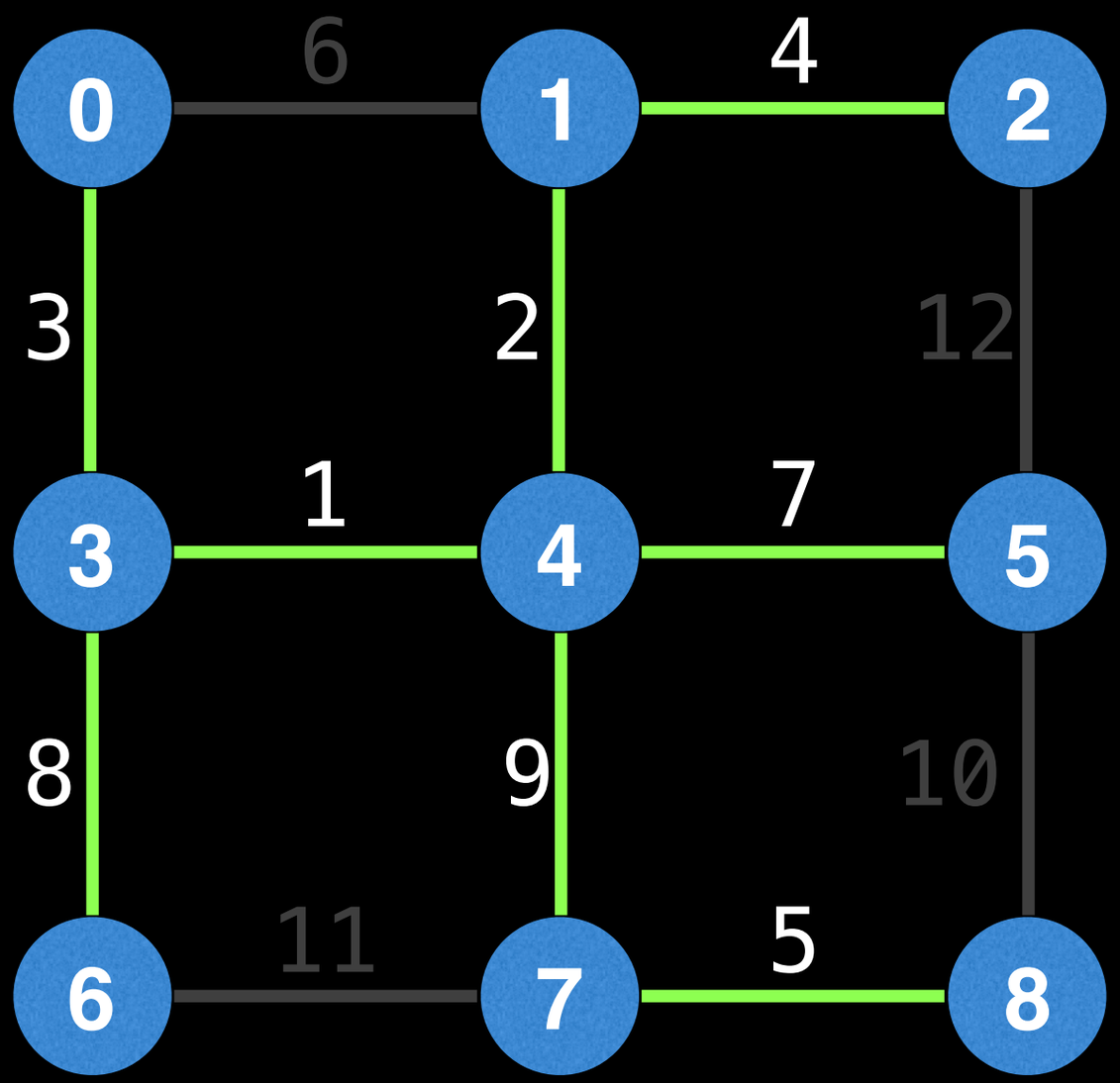
MST cost: 3 + 1 + 2 + 0 + 8 = 14

Find any MST



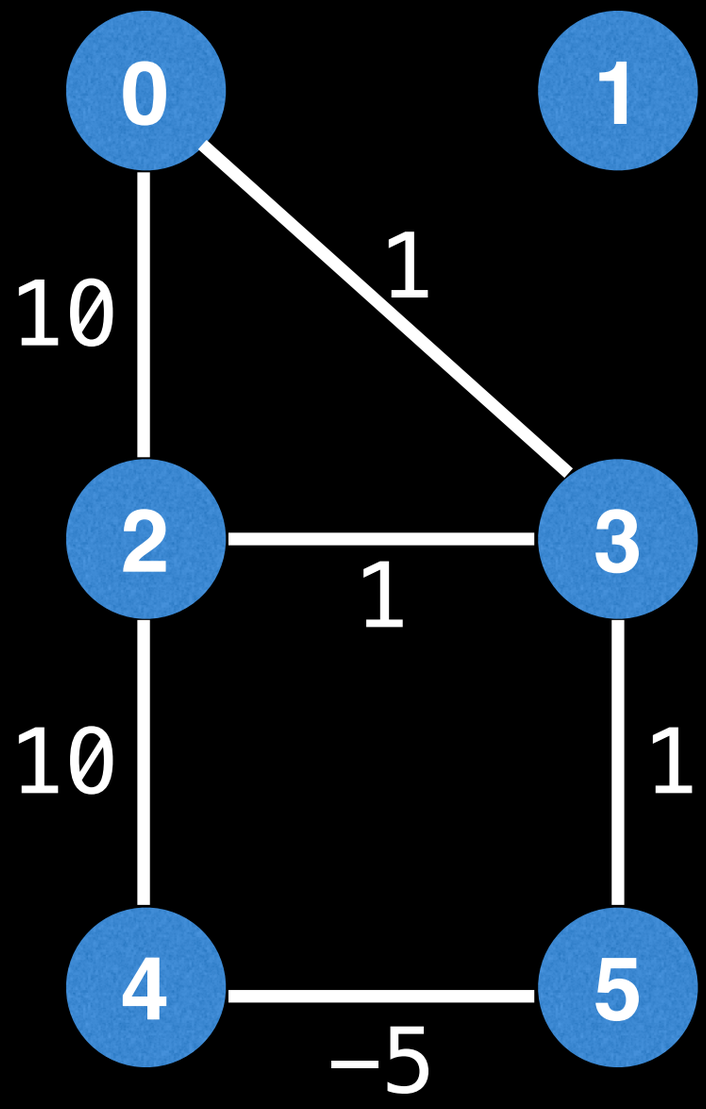
Find any MST



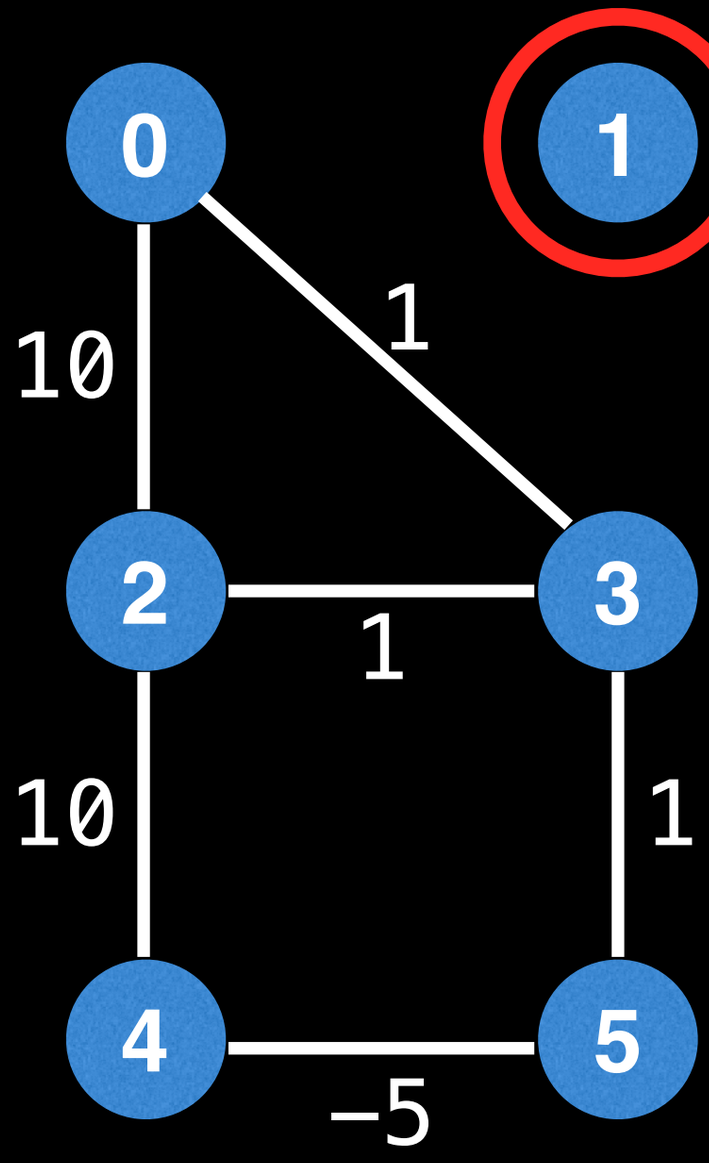


MST cost: 3 + 1 + 2 + 4 + 7 + 8 + 9 + 5 = 39

Find any MST



Find any MST



This graph has no MST!

All nodes must be connected to form a spanning tree.

Prim's MST Algorithm

Prim's is a greedy MST algorithm that works well on **dense graphs**. On these graphs, Prim's meets or improves on the time bounds of its popular rivals (Kruskal's & Borůvka's).

However, when it comes to finding the **minimum spanning forest** on a disconnected graph, Prim's cannot do this as easily (the algorithm must be run on each connected component individually).

The **lazy version** of Prim's has a runtime of **O(E*log(E))**, but the **eager version** (which we will also look at) has a better runtime of **O(E*log(V))**.

Lazy Prim’s MST Overview

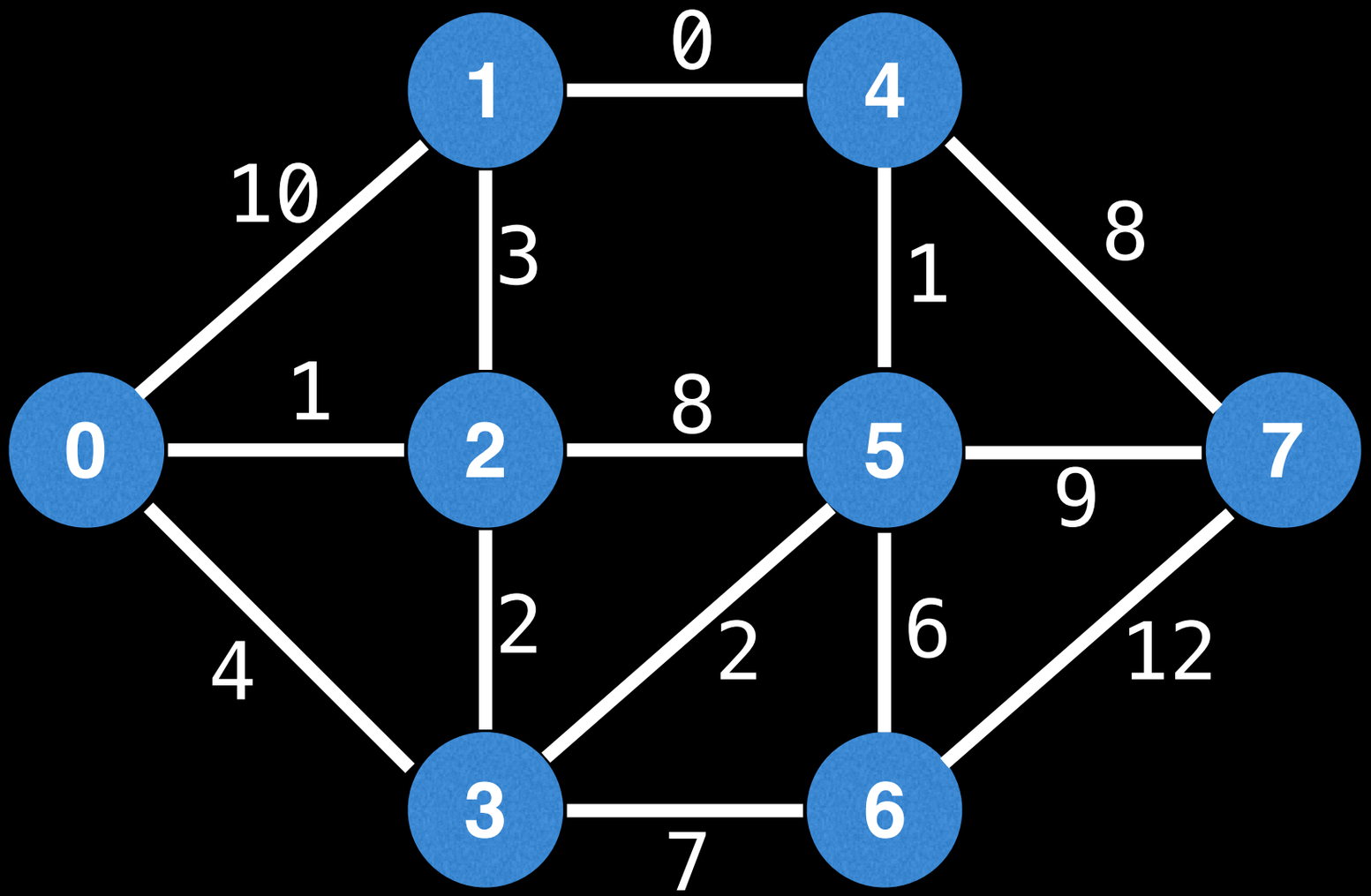
Maintain a min Priority Queue (PQ) that sorts edges based on min edge cost. This will be used to determine the next node to visit and the edge used to get there.

Start the algorithm on any node **s**. Mark **s** as visited and iterate over all edges of **s**, adding them to the PQ.

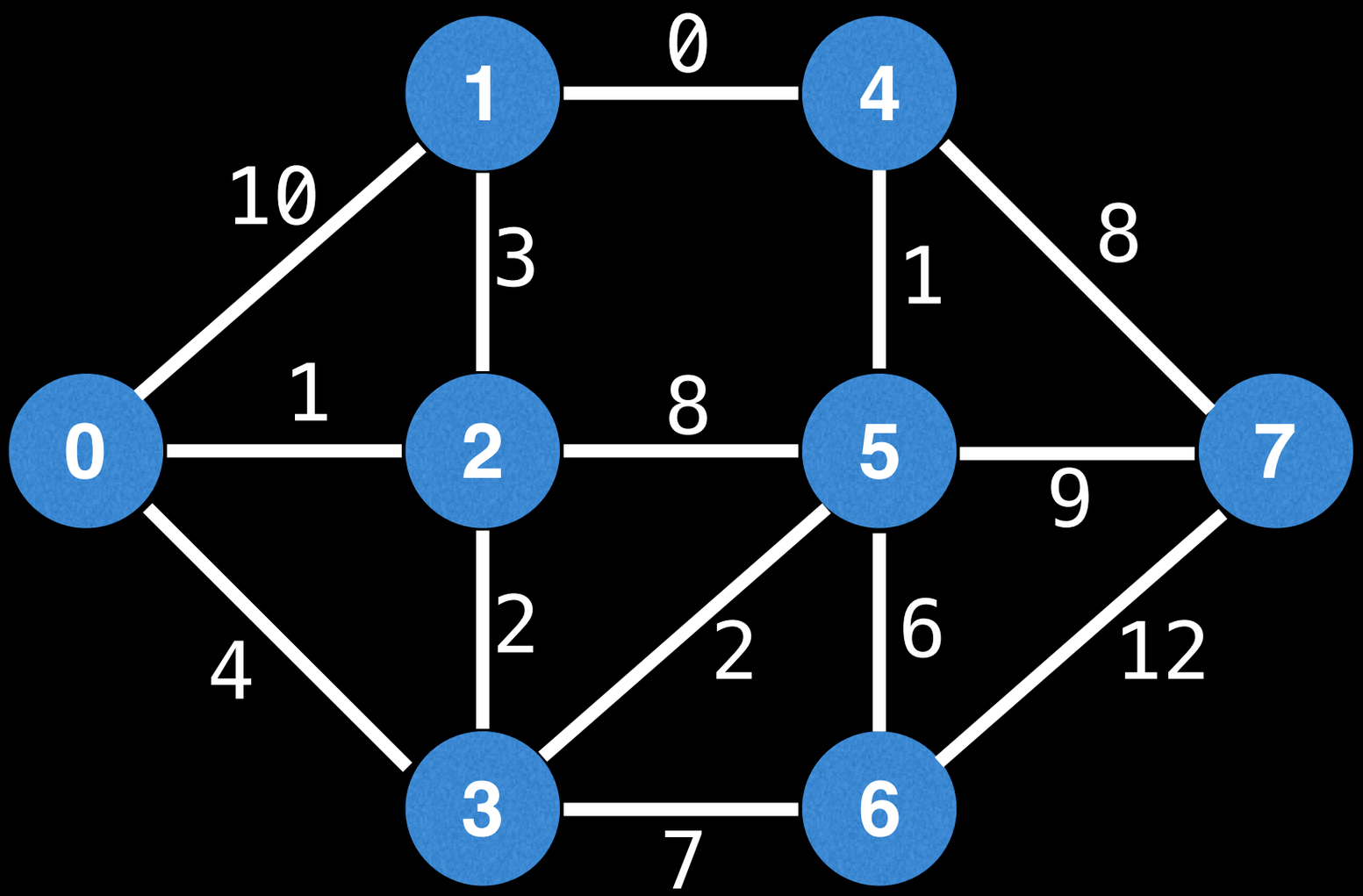
While the PQ is not empty and a MST has not been formed, dequeue the next cheapest edge from the PQ. If the dequeued edge is outdated (meaning the node it points to has already been visited) then skip it and poll again. Otherwise, mark the current node as visited and add the selected edge to the MST.

Iterate over the new current node’s edges and add all its edges to the PQ. Do not add edges to the PQ which point to already visited nodes.

Lazy Prim's

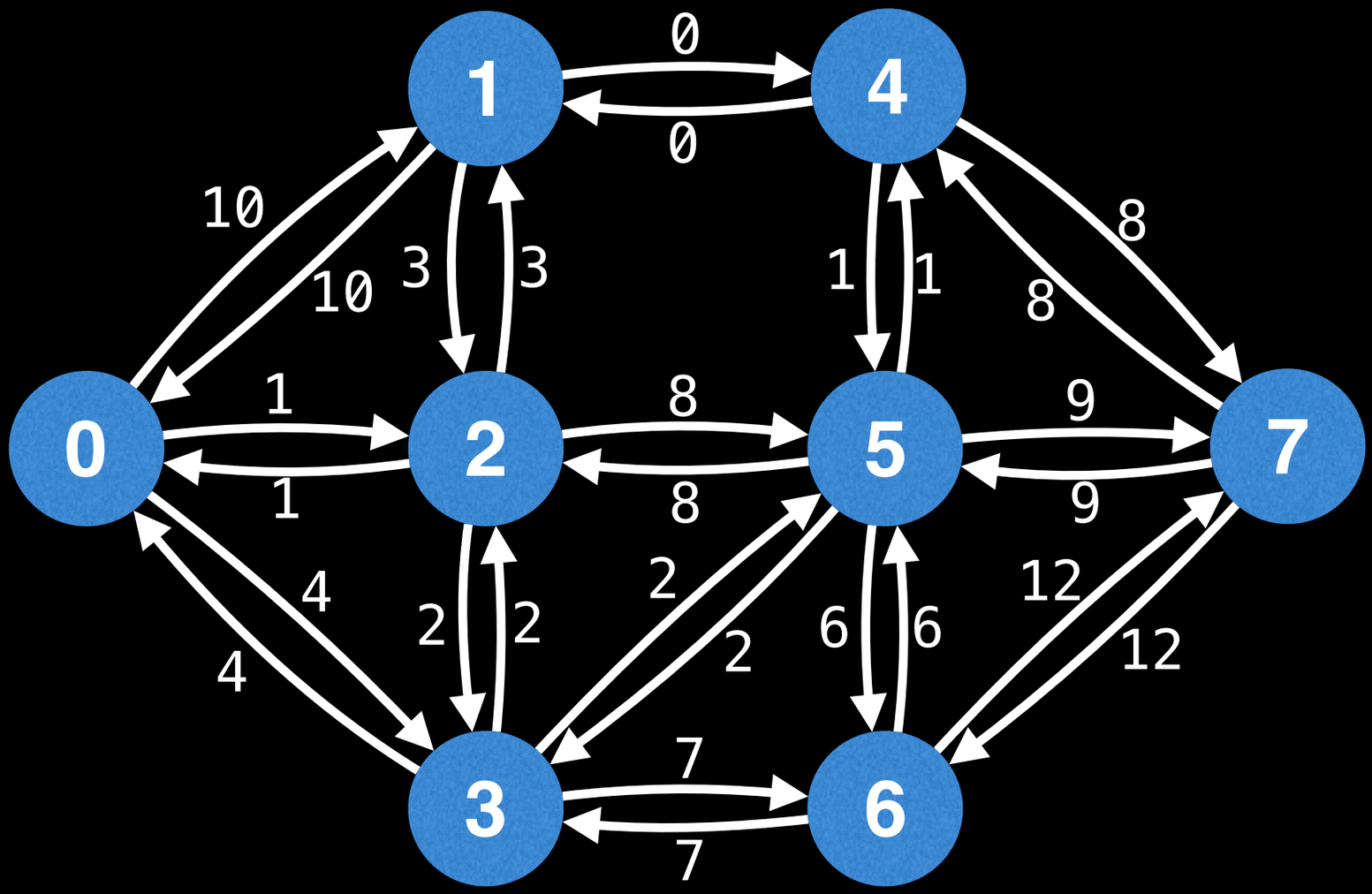


Lazy Prim's



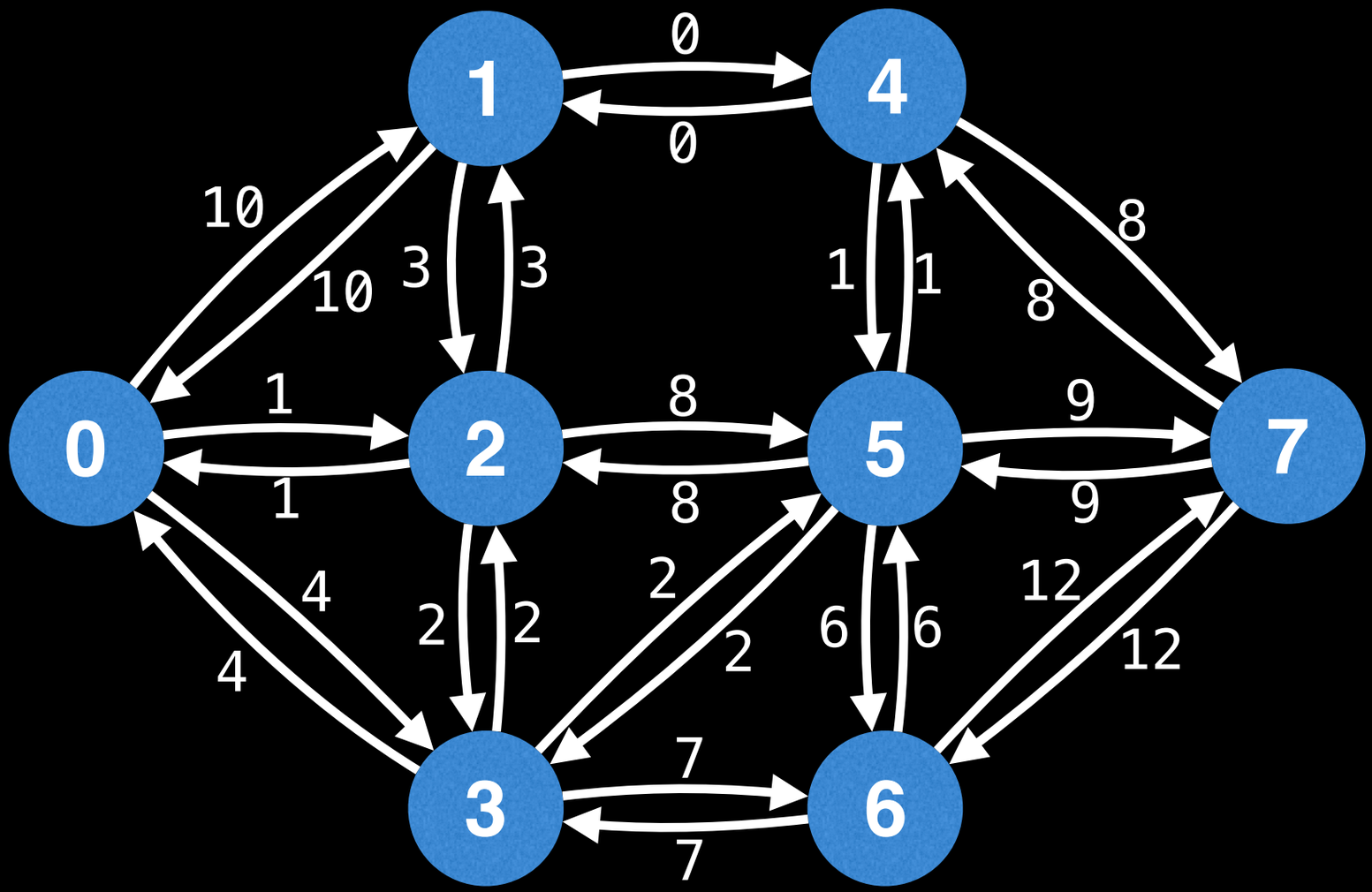
One thing to bear in mind is that although the graph above represents an **undirected graph**, the internal adjacency list representation typically has each undirected edge stored as **two directed edges**.

Lazy Prim's



The actual internal representation typically looks like this.

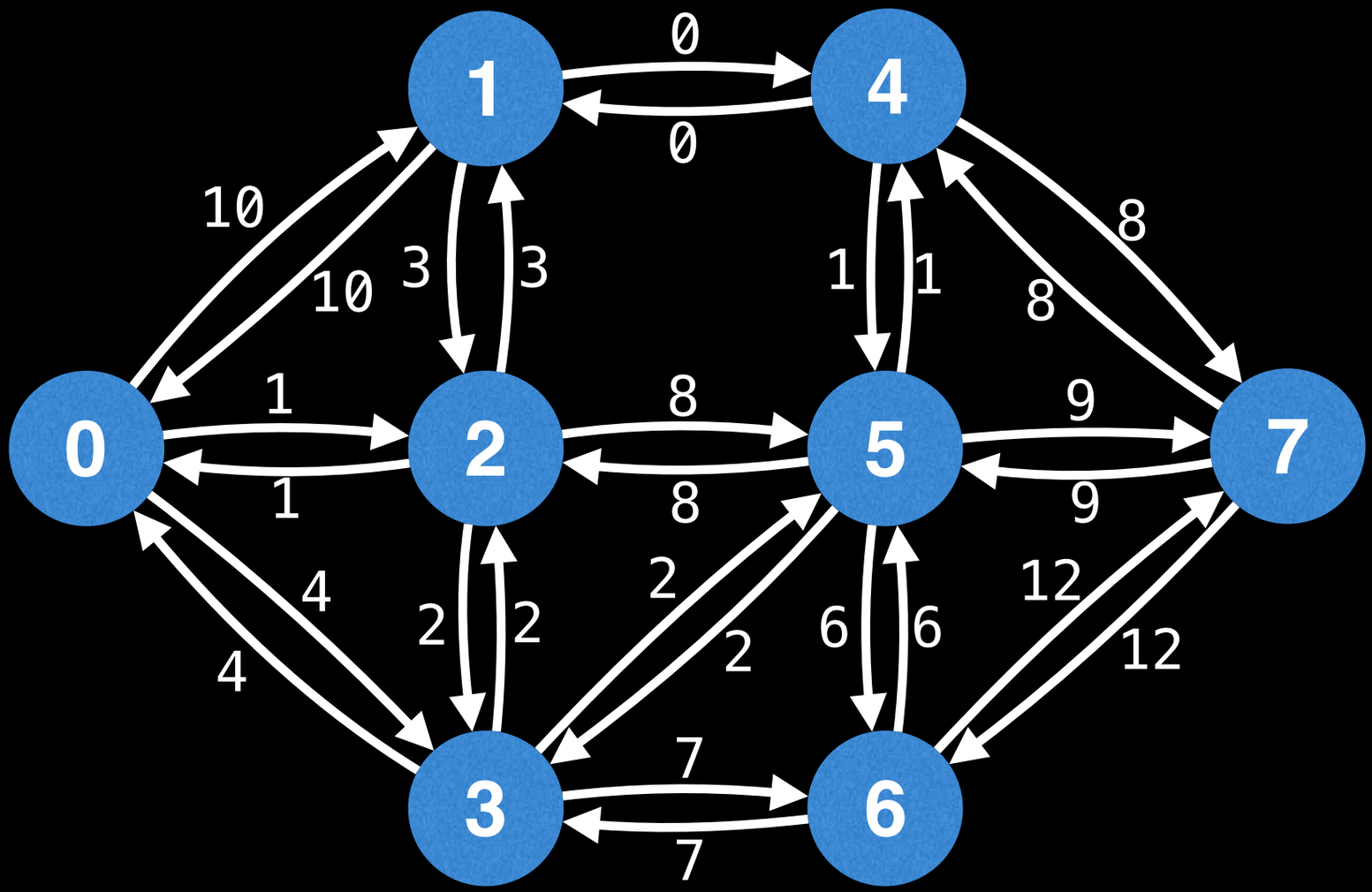
Lazy Prim's



Edges in PQ (start, end, cost)

On the right we will be keeping track of the PQ containing the edge objects as triplets:
(start node, end node, edge cost)

Lazy Prim's



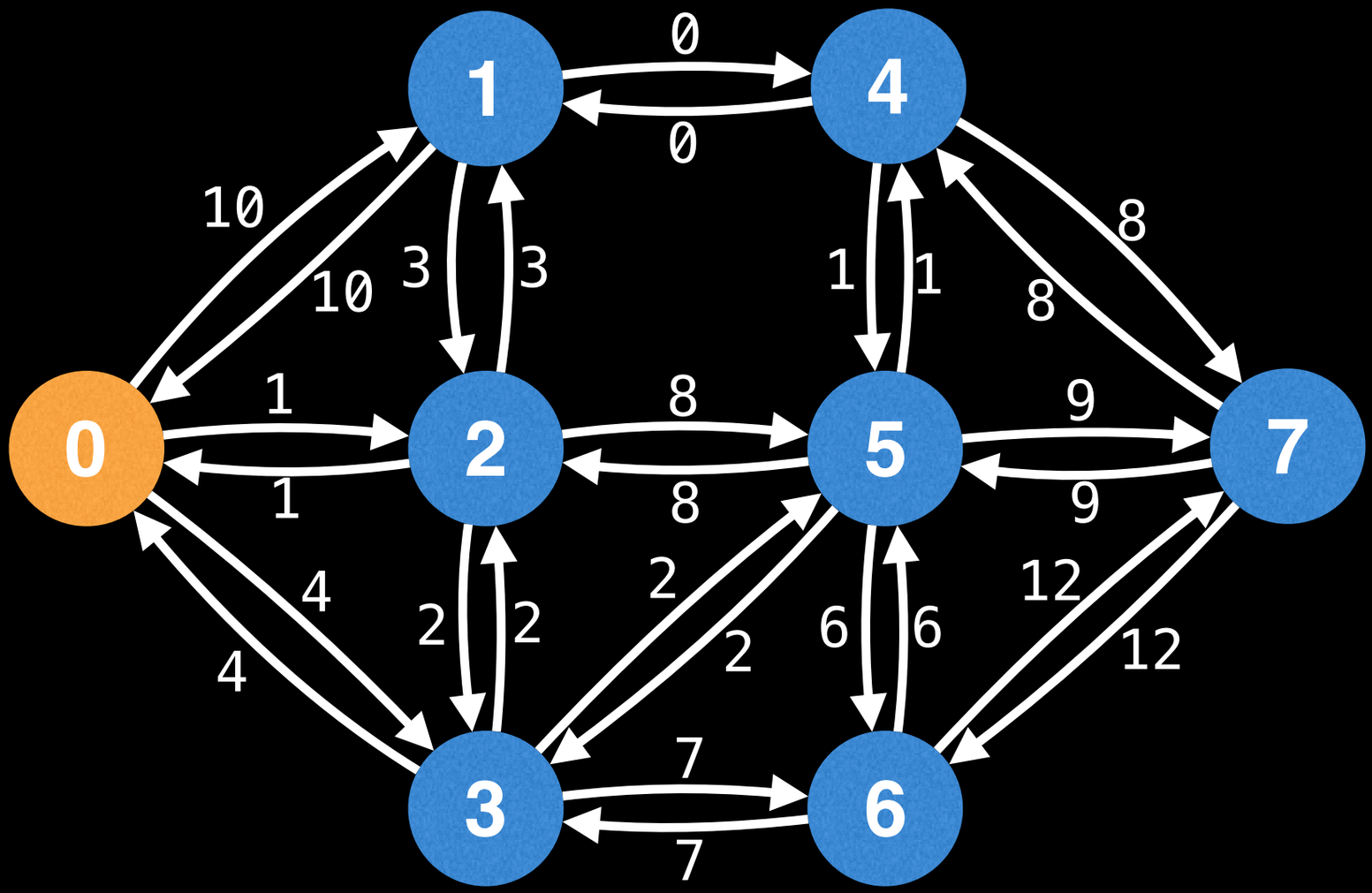
Edges in PQ
(start, end, cost)

Unvisited

Visiting

Visited

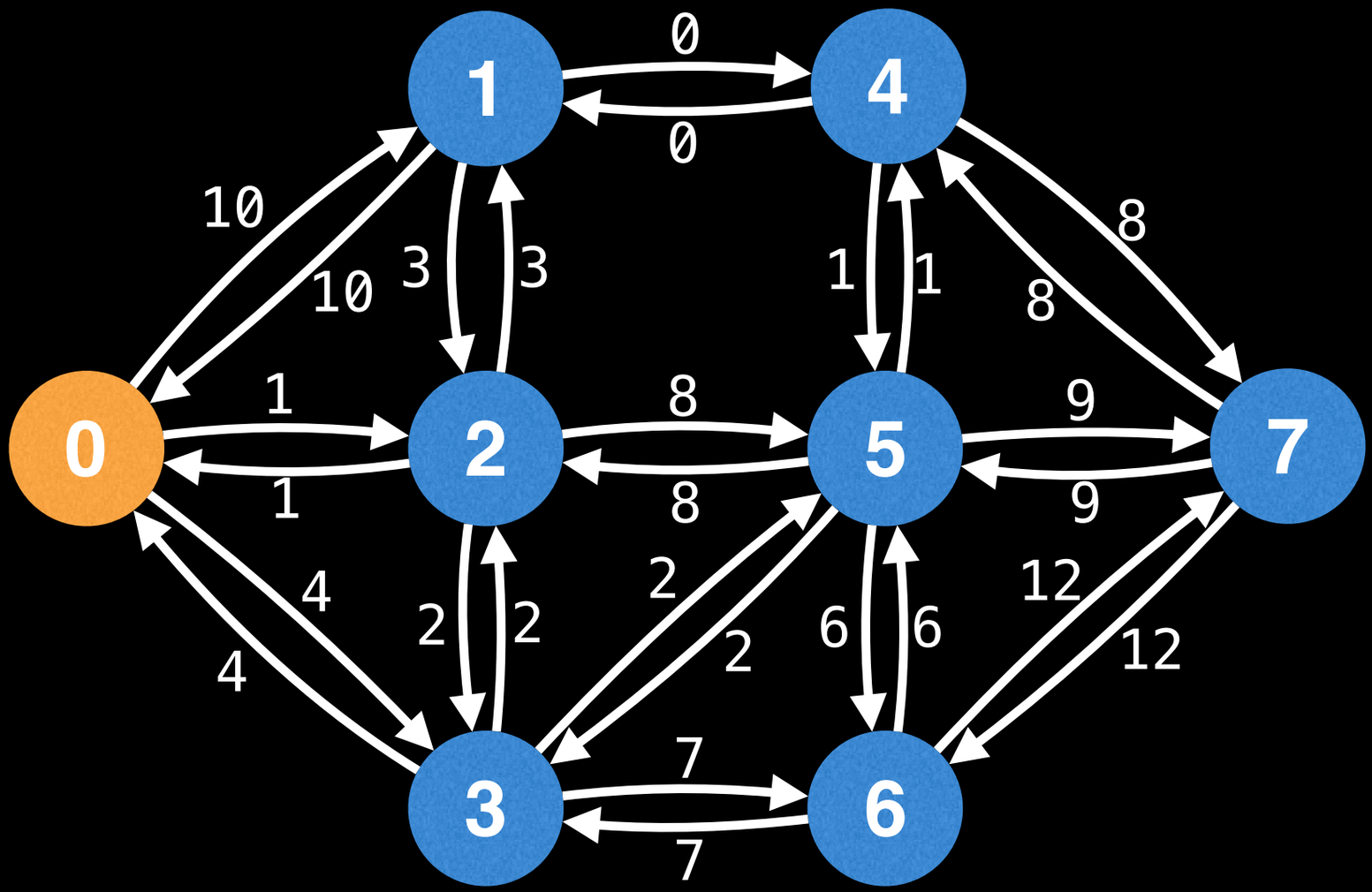
Lazy Prim’s



Edges in PQ
(start, end, cost)

Let’s begin Prim’s at node 0.

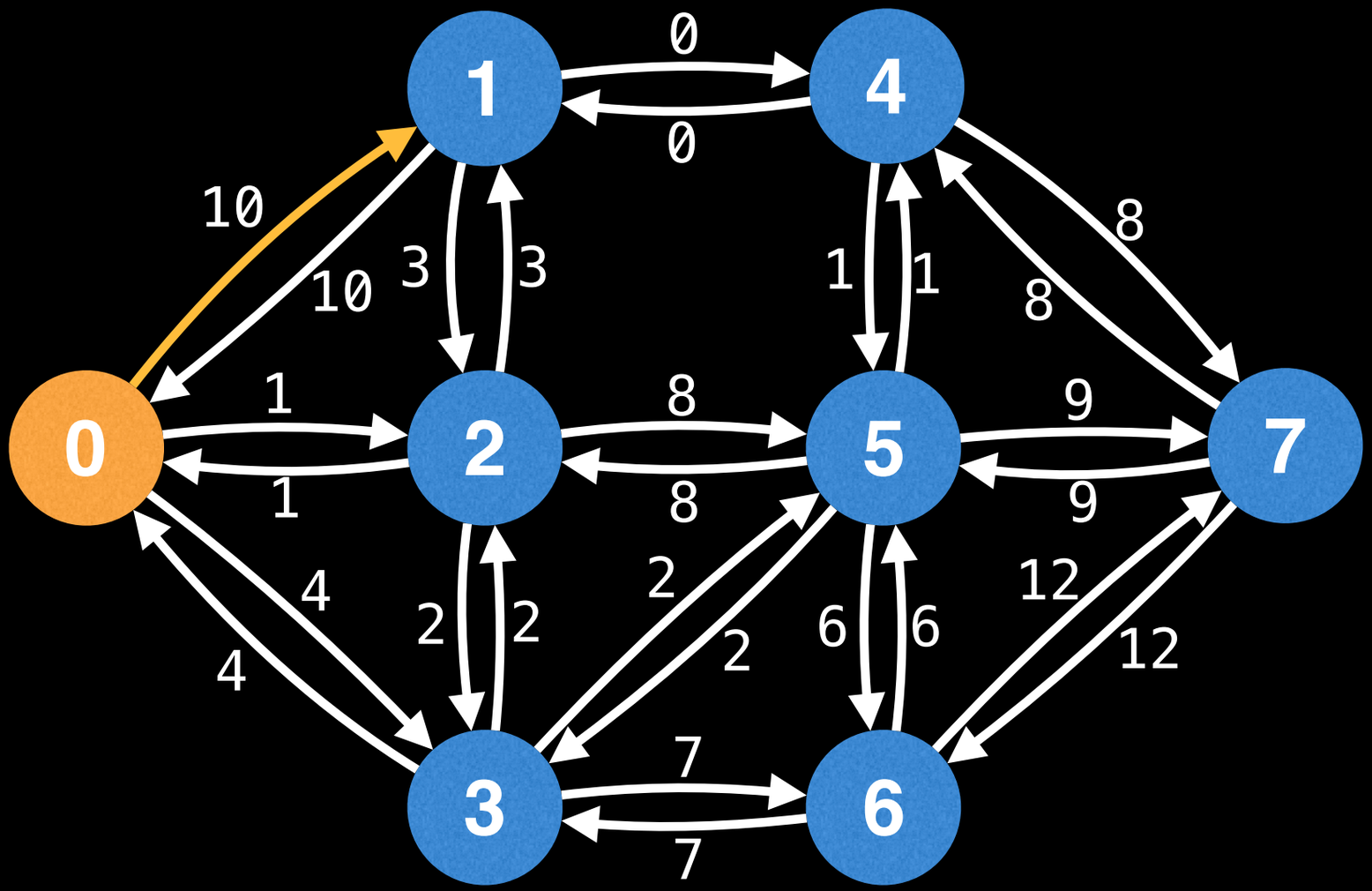
Lazy Prim's



Edges in PQ
(start, end, cost)

Iterate over all outgoing edges
and add them to the PQ.

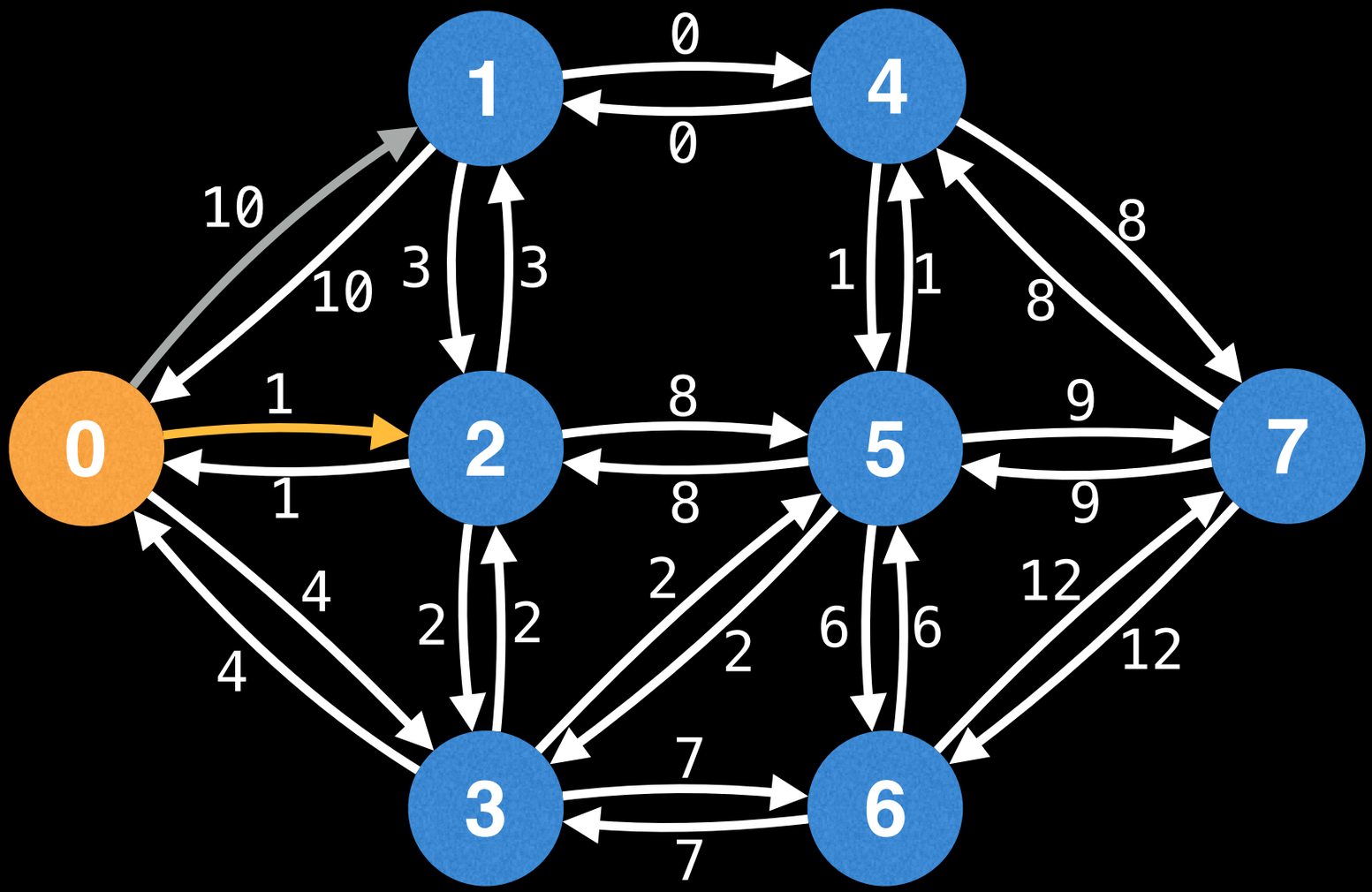
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)

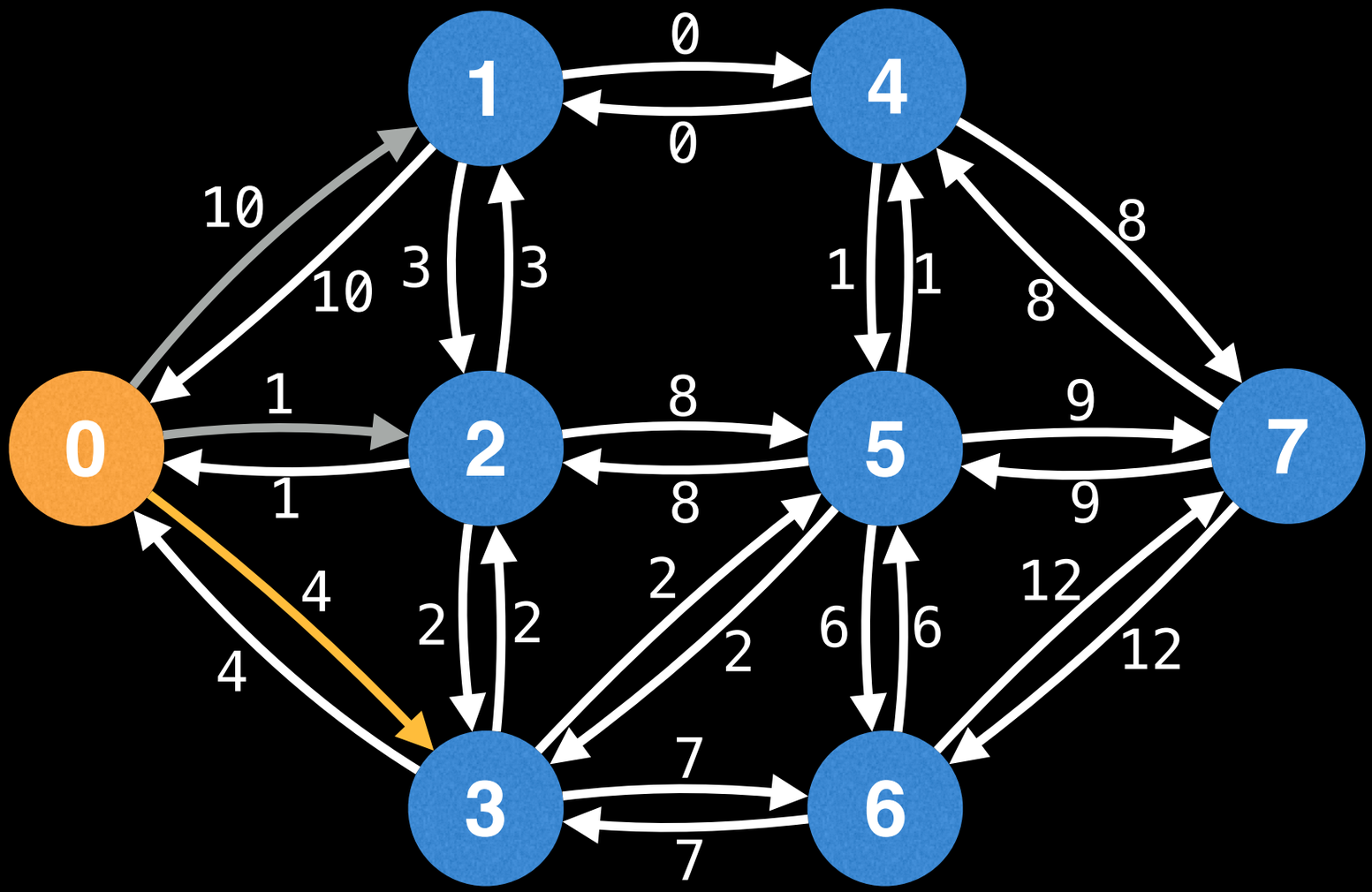
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
(0, 2, 1)

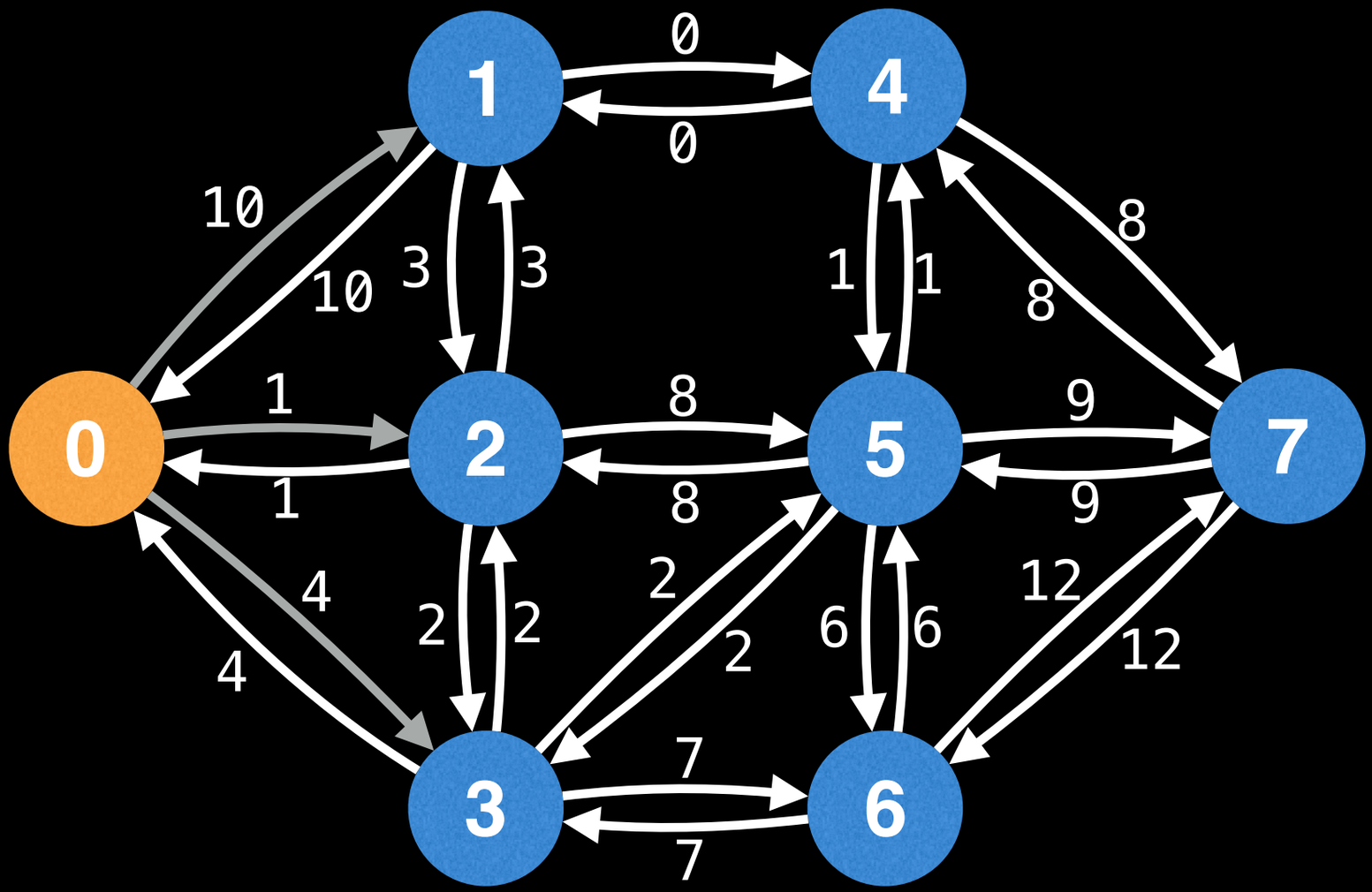
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
(0, 2, 1)
(0, 3, 4)

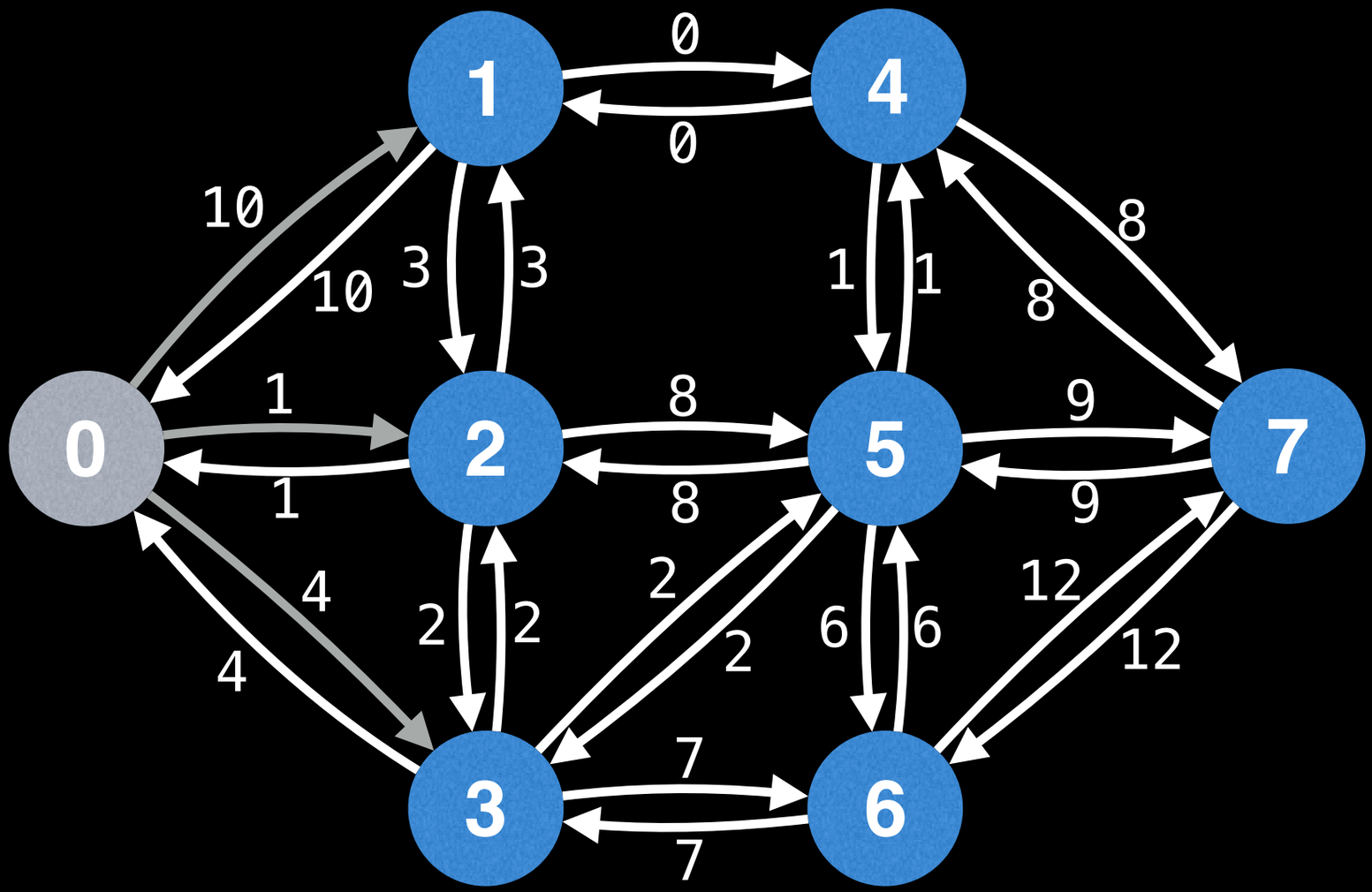
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
(0, 2, 1)
(0, 3, 4)

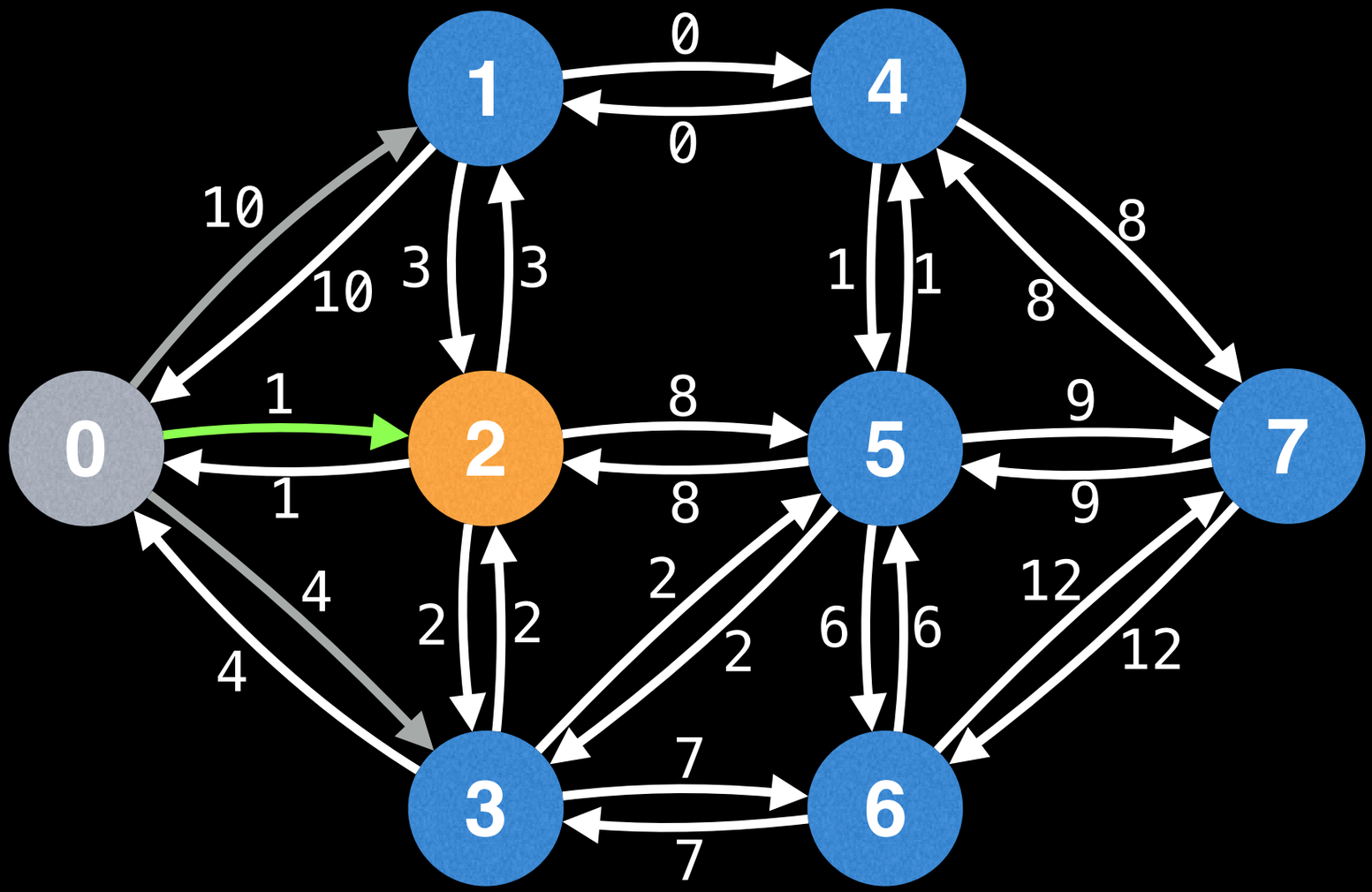
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
(0, 2, 1)
(0, 3, 4)

Lazy Prim's

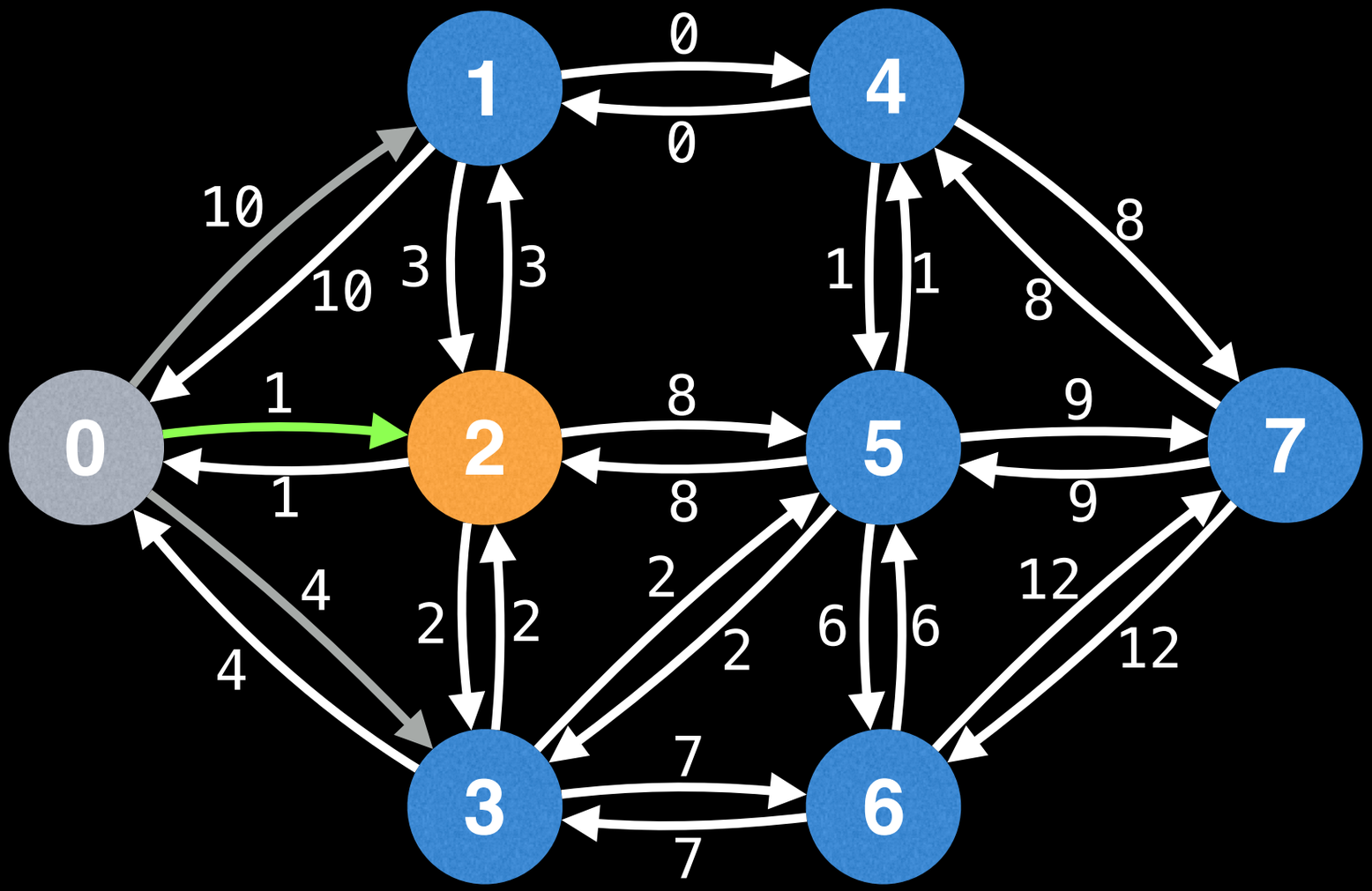


Edges in PQ
(start, end, cost)

(0, 1, 10)
→ (0, 2, 1)
(0, 3, 4)

Edge (0, 2, 1) has the lowest value in the PQ so it gets added to the MST. This also means that the next node we process is node 2.

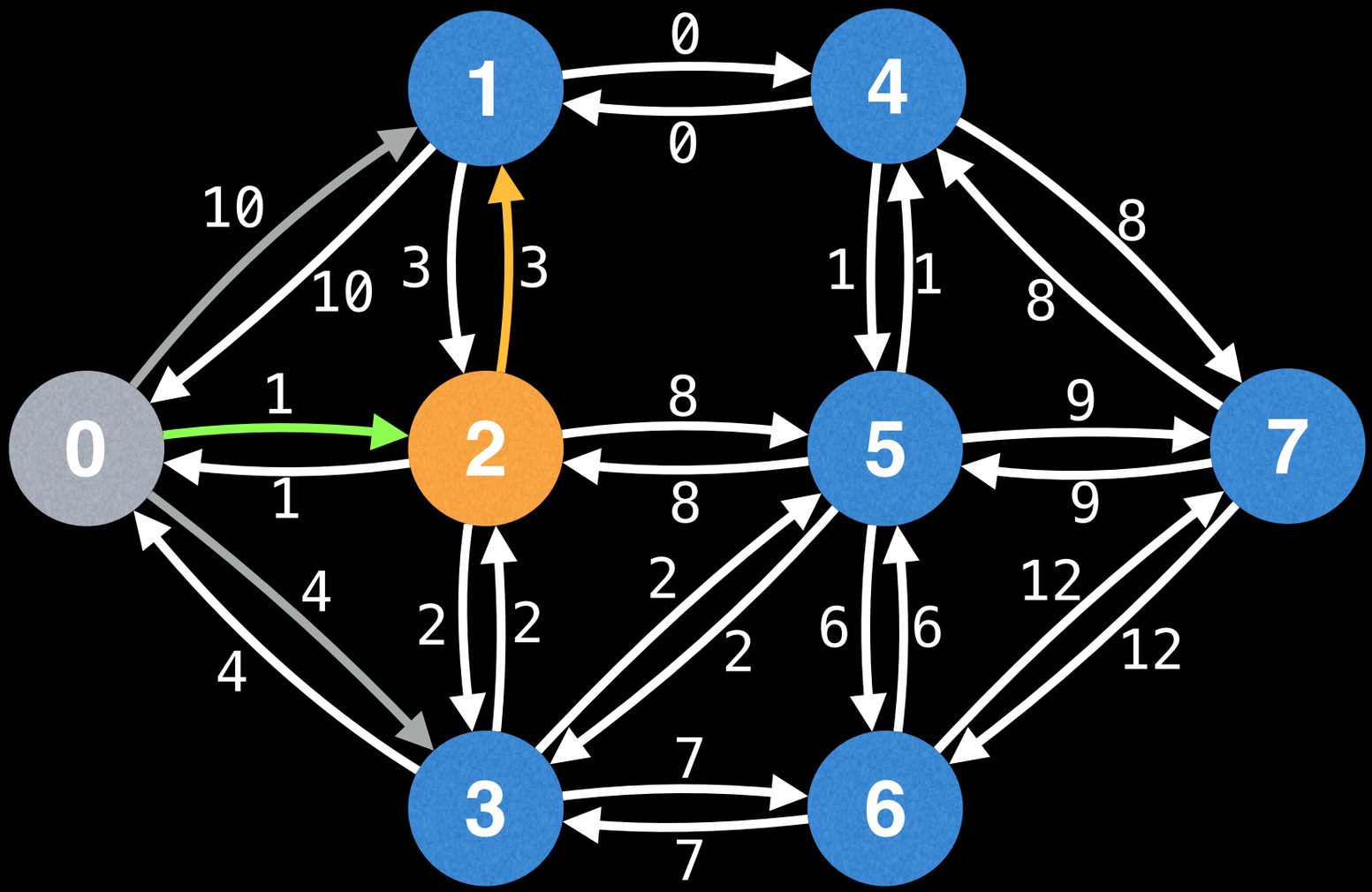
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)

Lazy Prim's

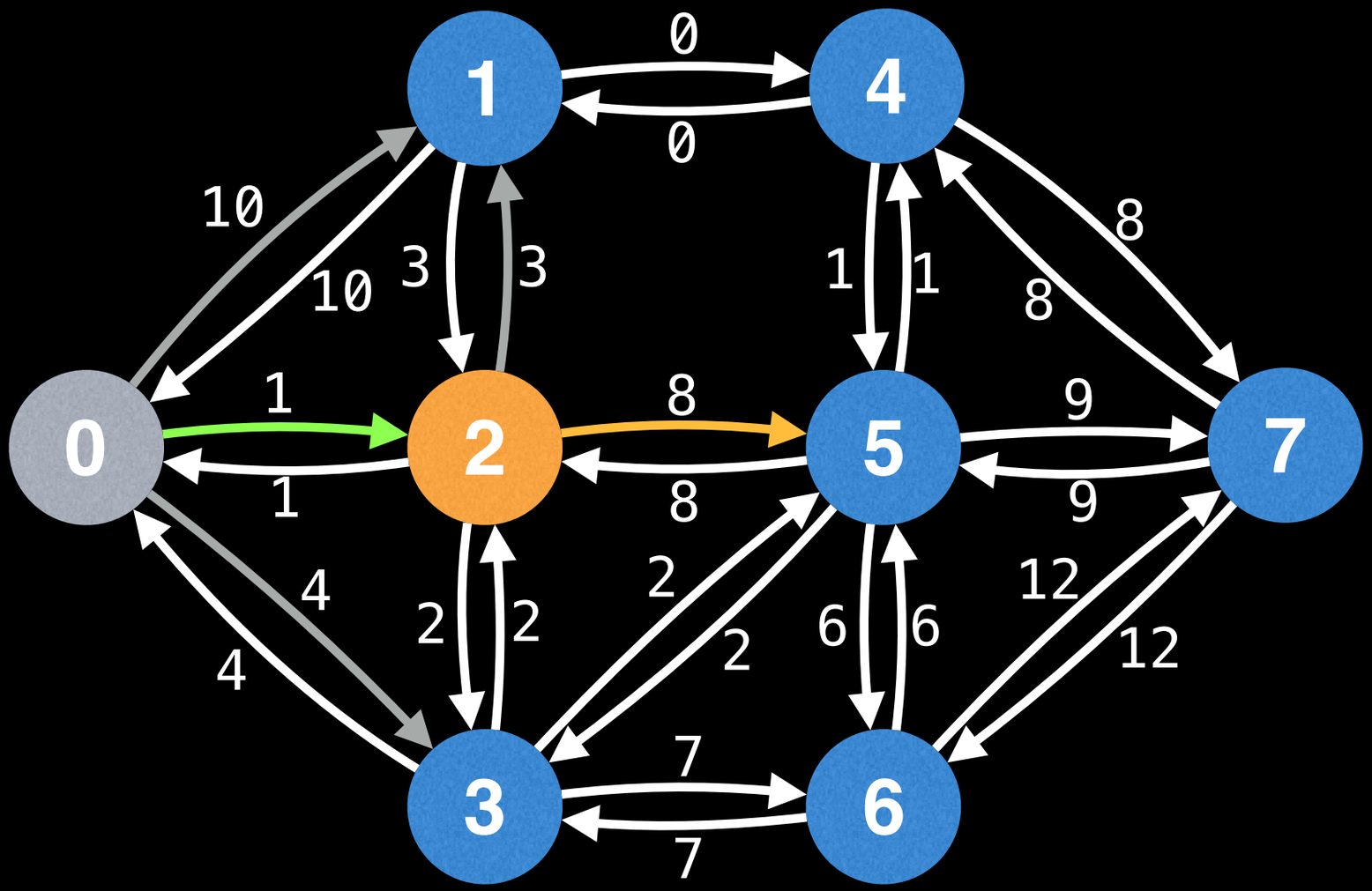


Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)

Next, iterate through all the edges of node 2 and add them to the PQ.

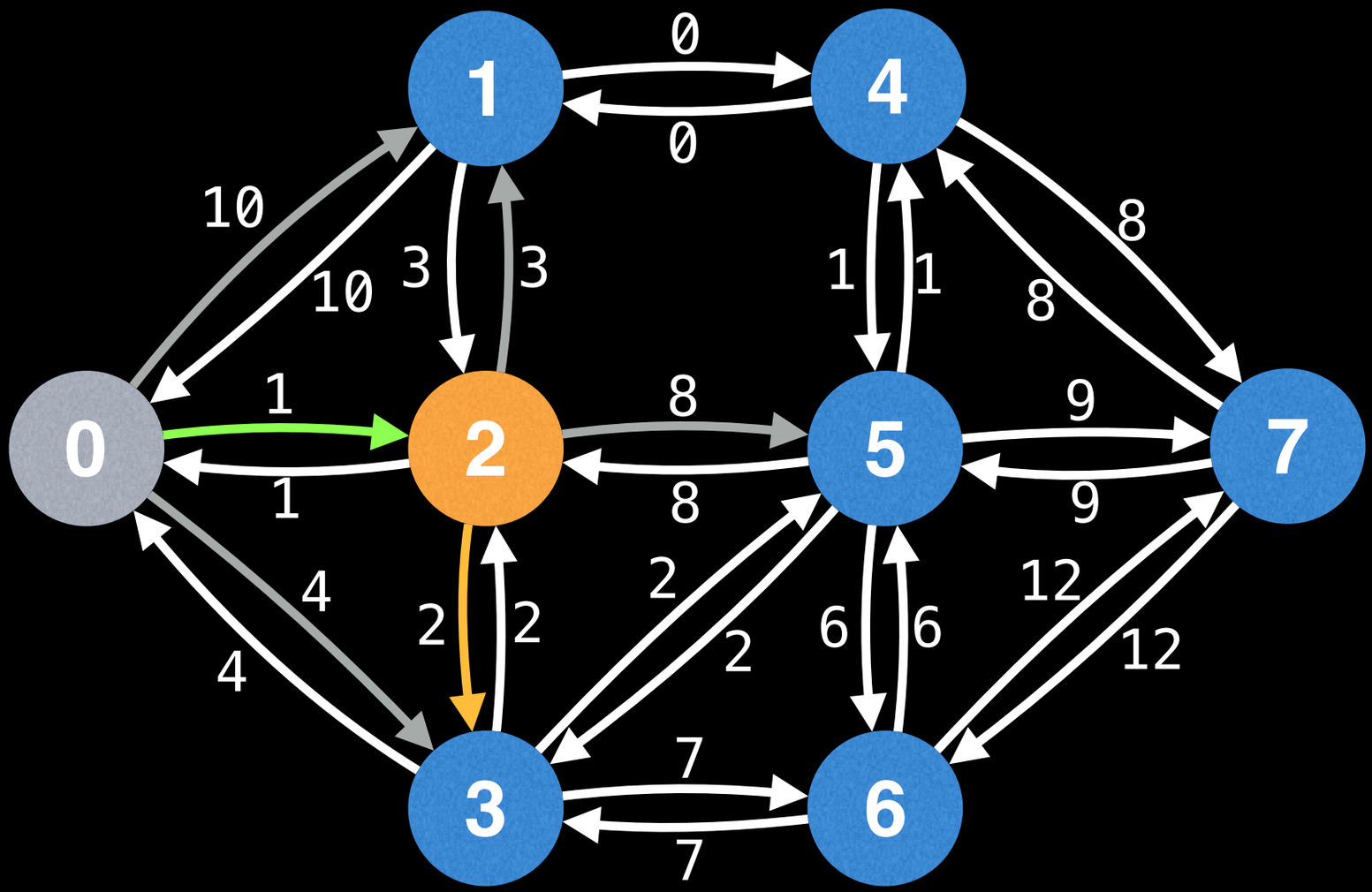
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)

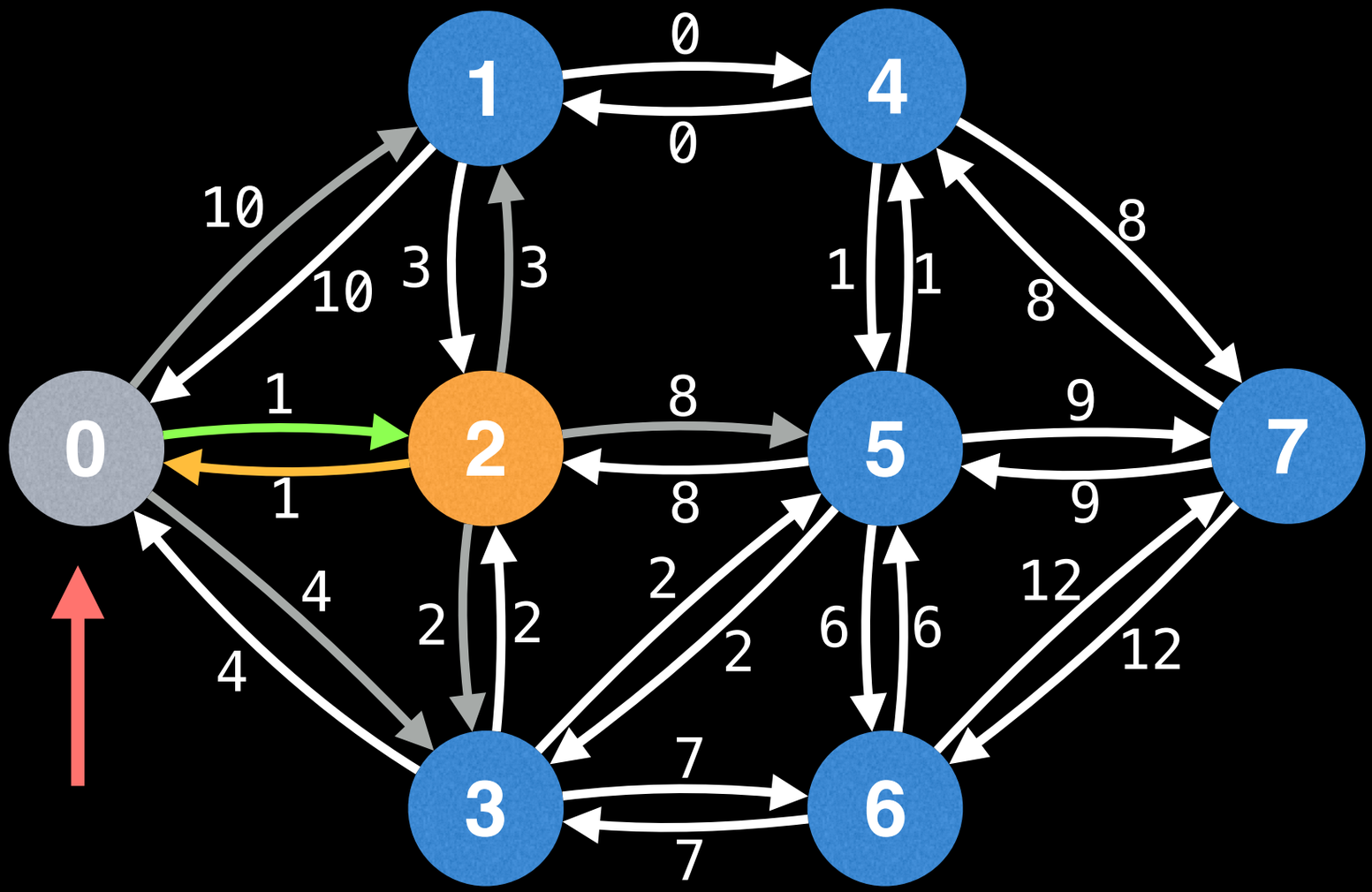
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
(2, 3, 2)

Lazy Prim’s

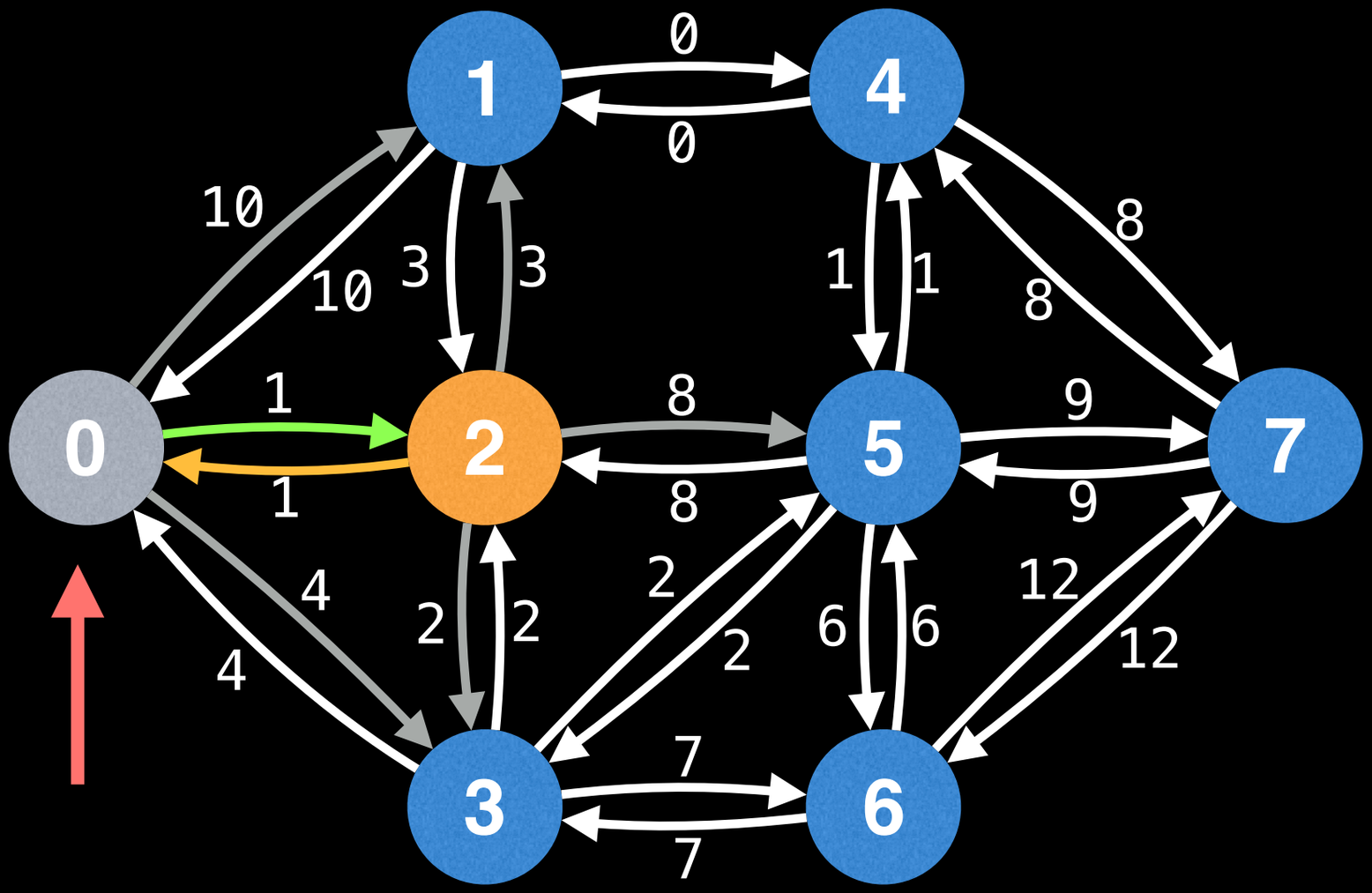


Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
(2, 3, 2)

Node 0 is already visited so we skip adding the edge (2, 0, 1) to the PQ.

Lazy Prim's

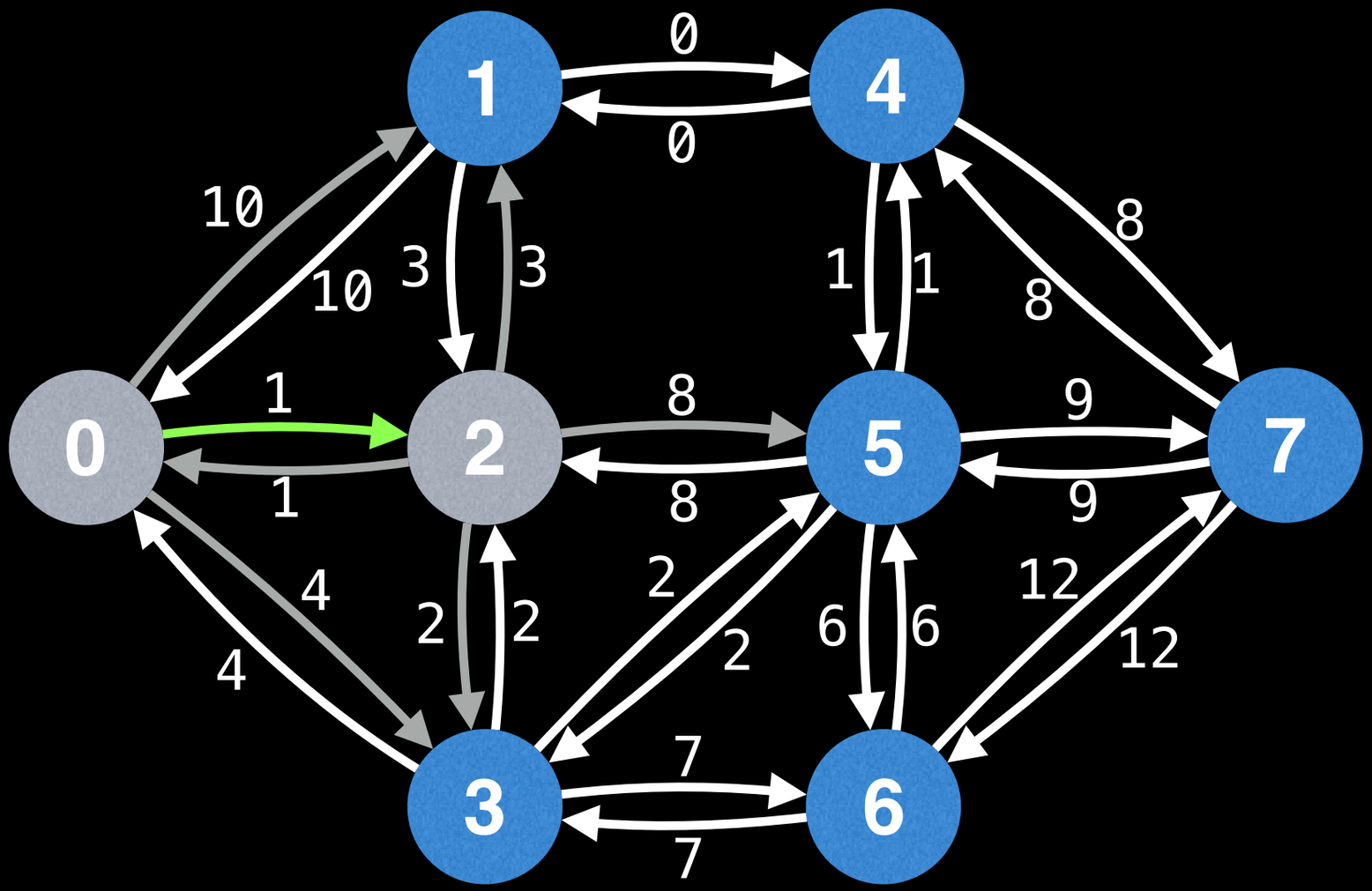


Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
(2, 3, 2)

The reason we don't include edges which point to already visited nodes is that either they overlap with an edge already part of the MST (as is the case with the edge on this slide) or it would introduce a **cycle** in the MST, if included.

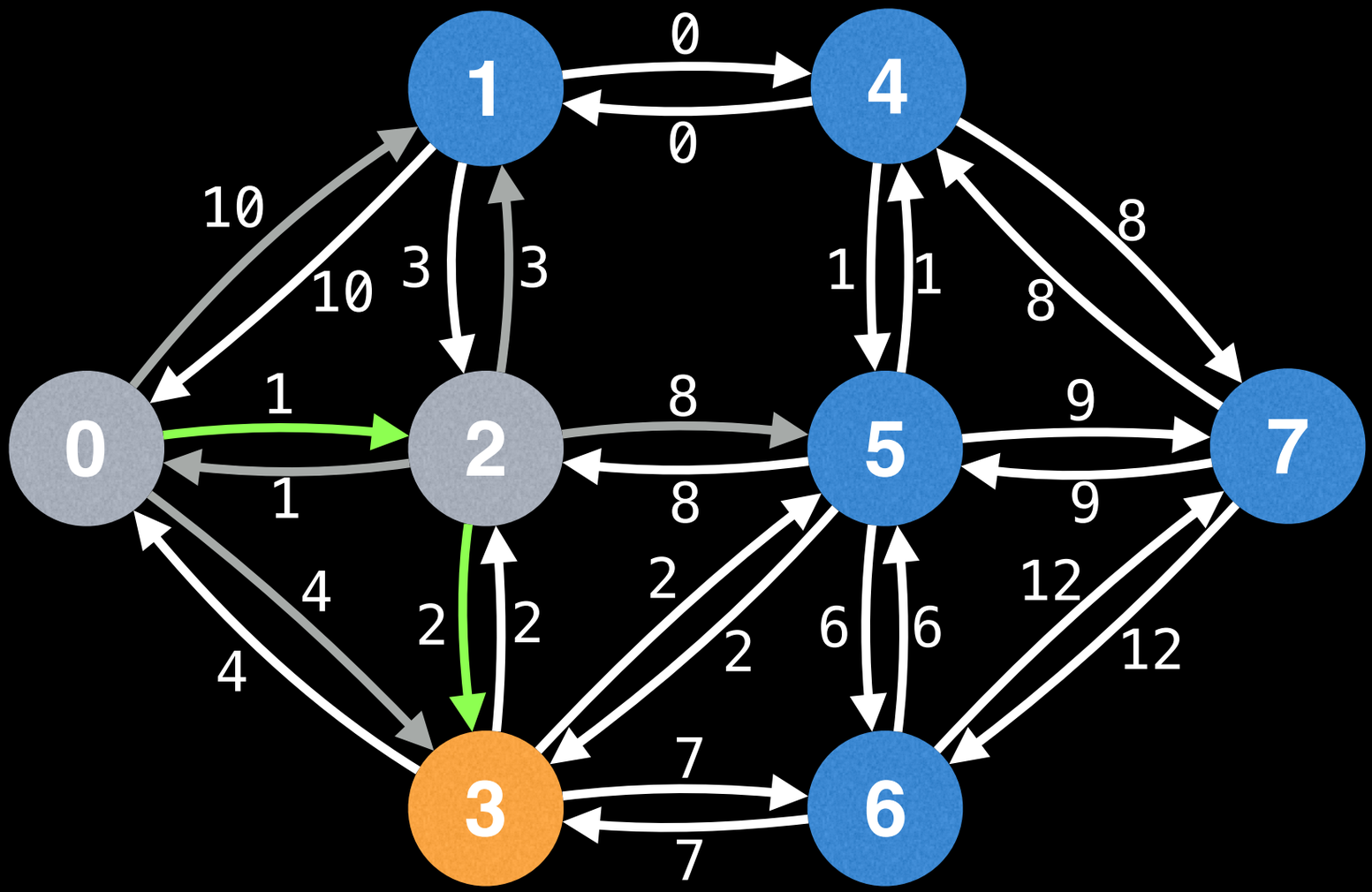
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
(2, 3, 2)

Lazy Prim's

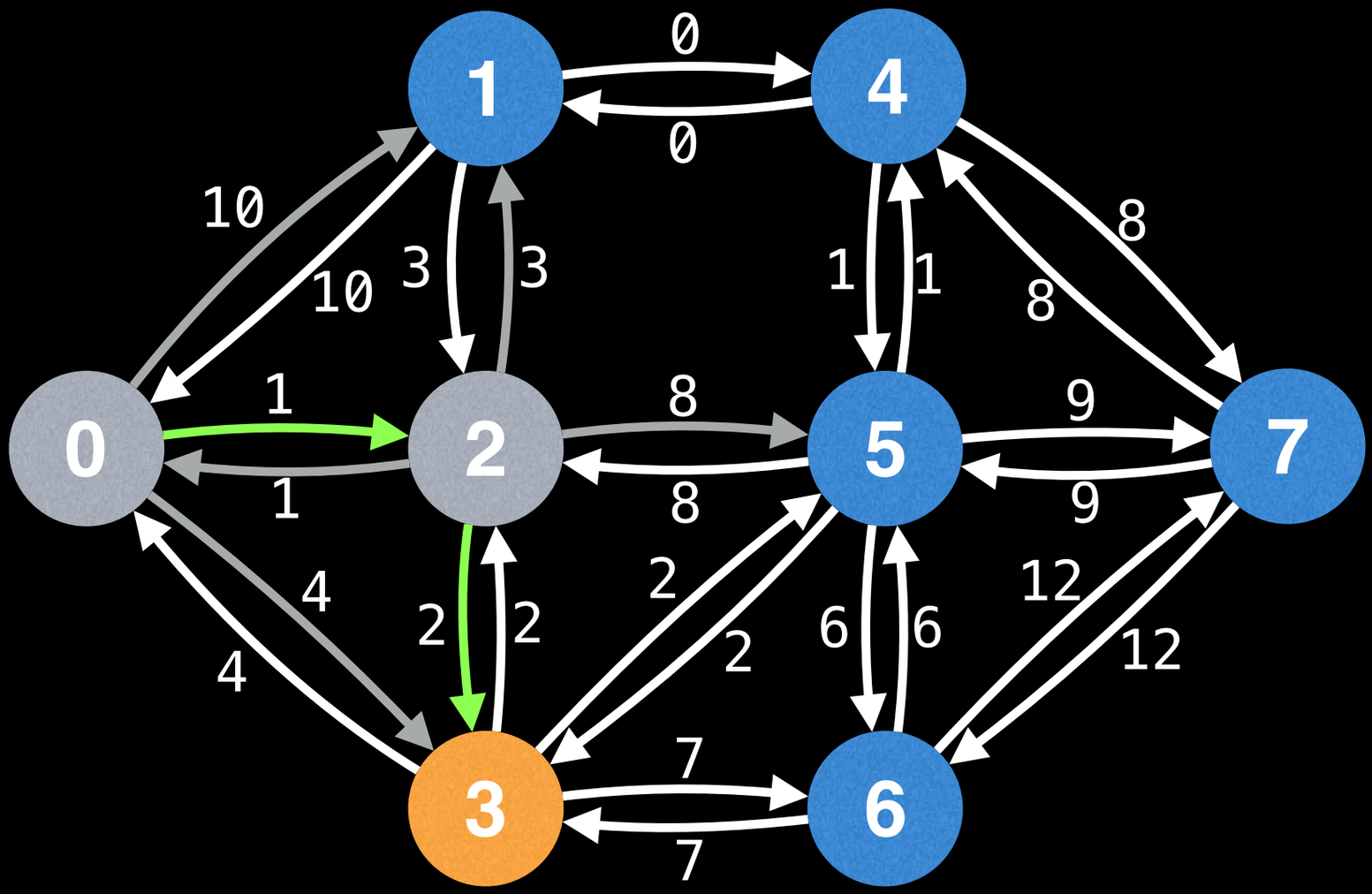


Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
→ (2, 3, 2)

Edge (2, 3, 2) has the lowest value in the PQ so it gets added to the MST. This also means that the next node we process is node 3.

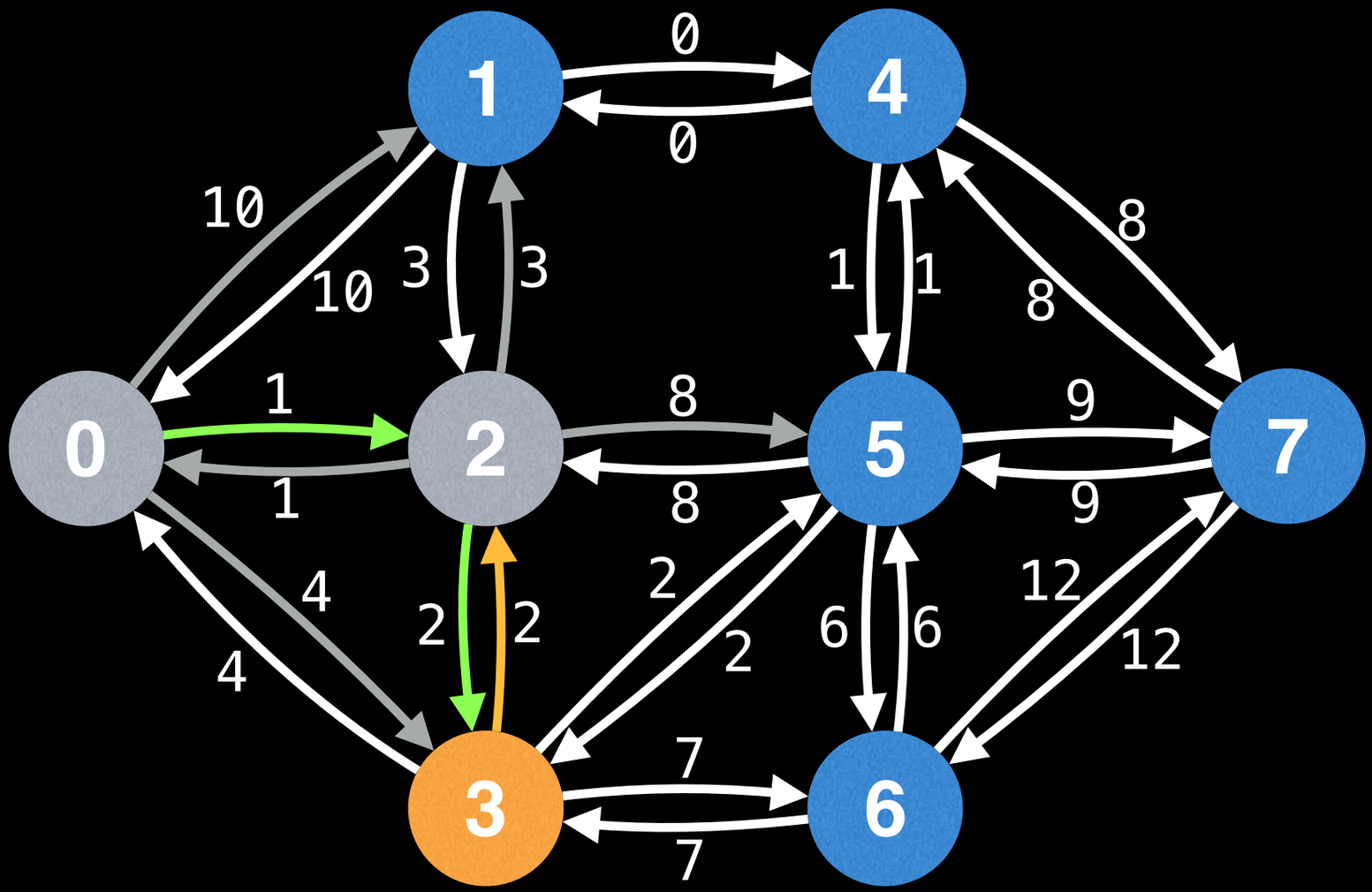
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
→ ~~(2, 3, 2)~~

Lazy Prim's

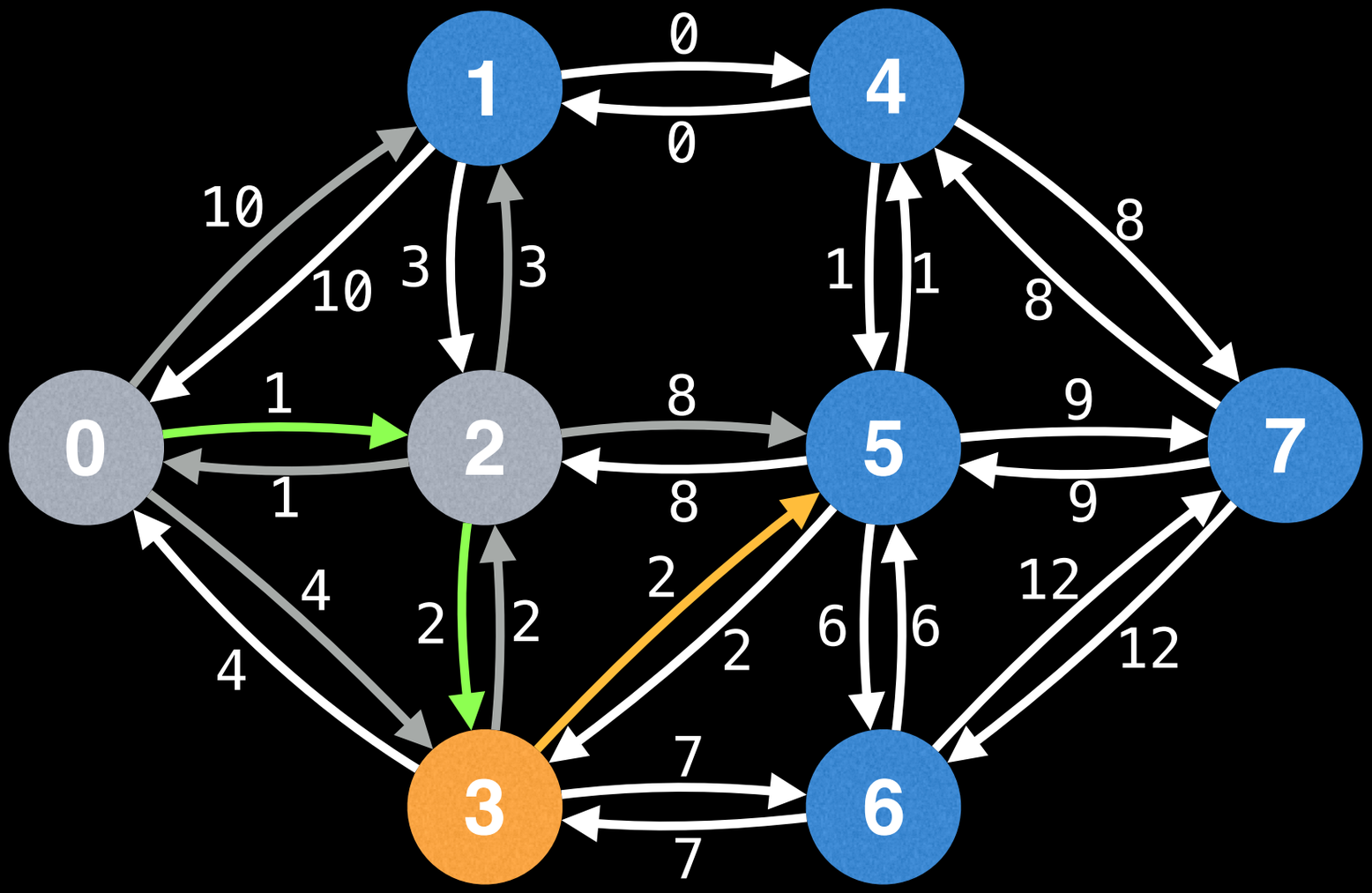


Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
→ ~~(2, 3, 2)~~

The same process of adding edges to the PQ and polling the smallest edge continues until the MST is complete.

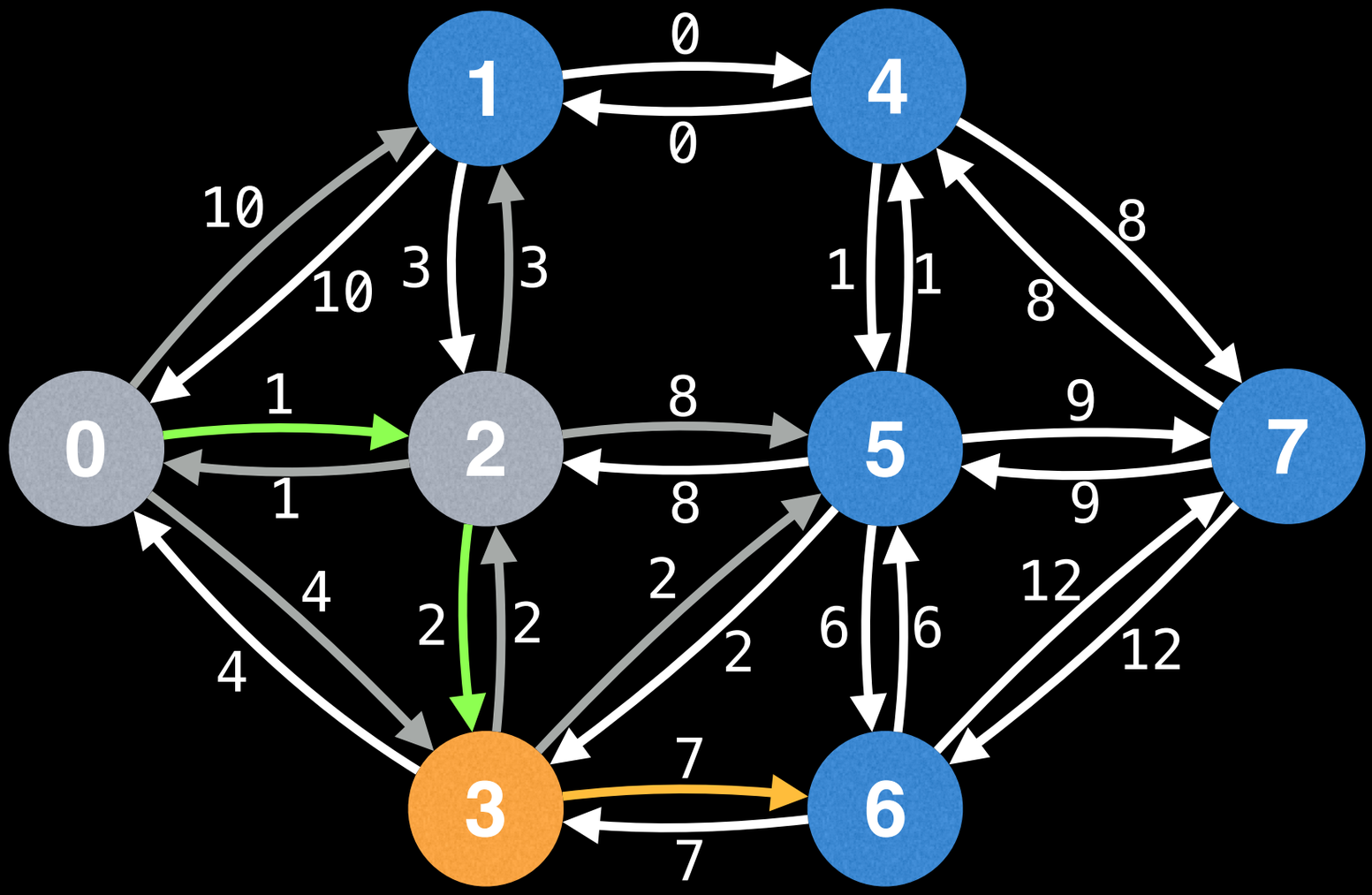
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
→ ~~(2, 3, 2)~~
(3, 5, 2)

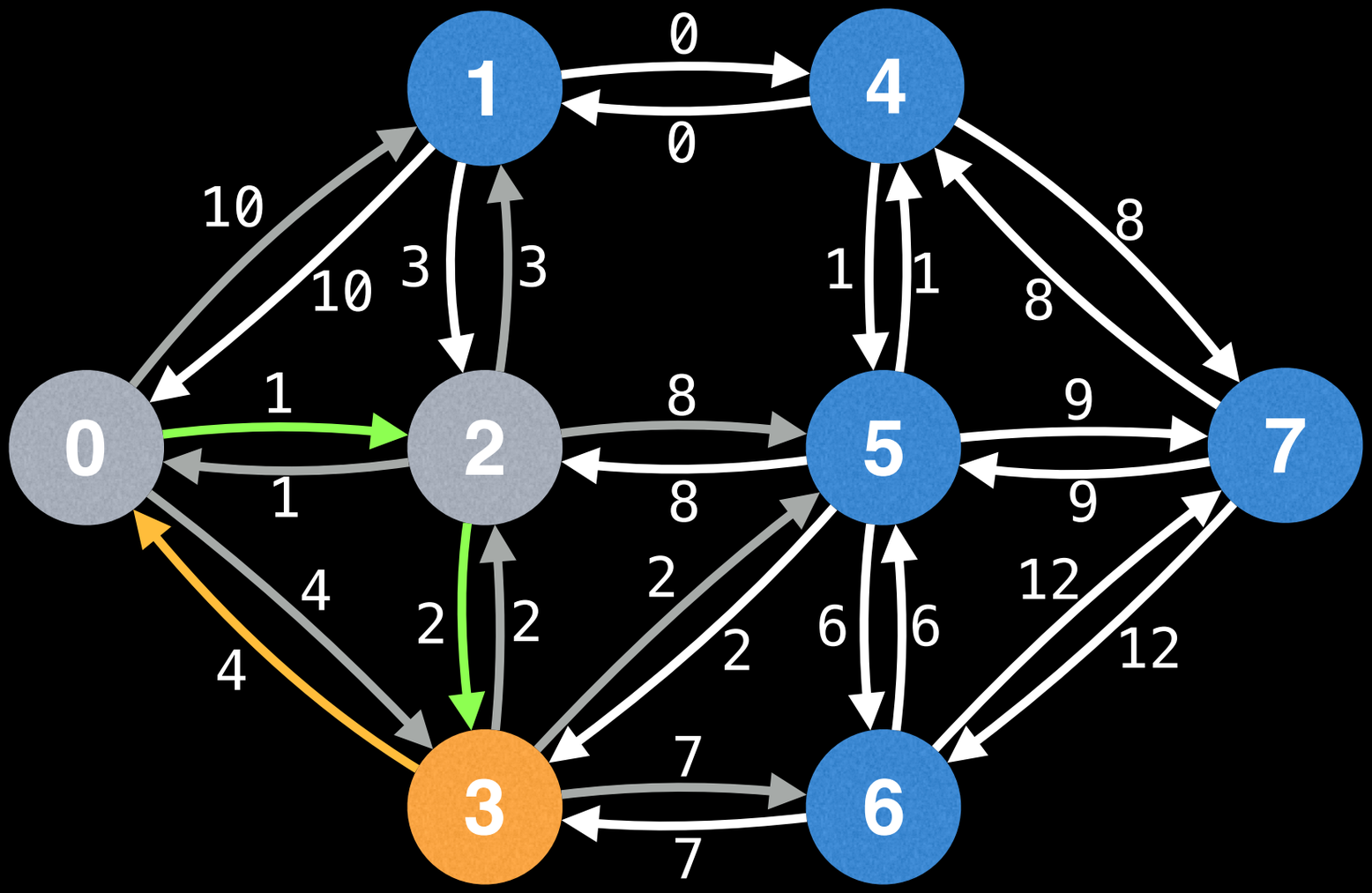
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
→ ~~(2, 3, 2)~~
(3, 5, 2)
(3, 6, 7)

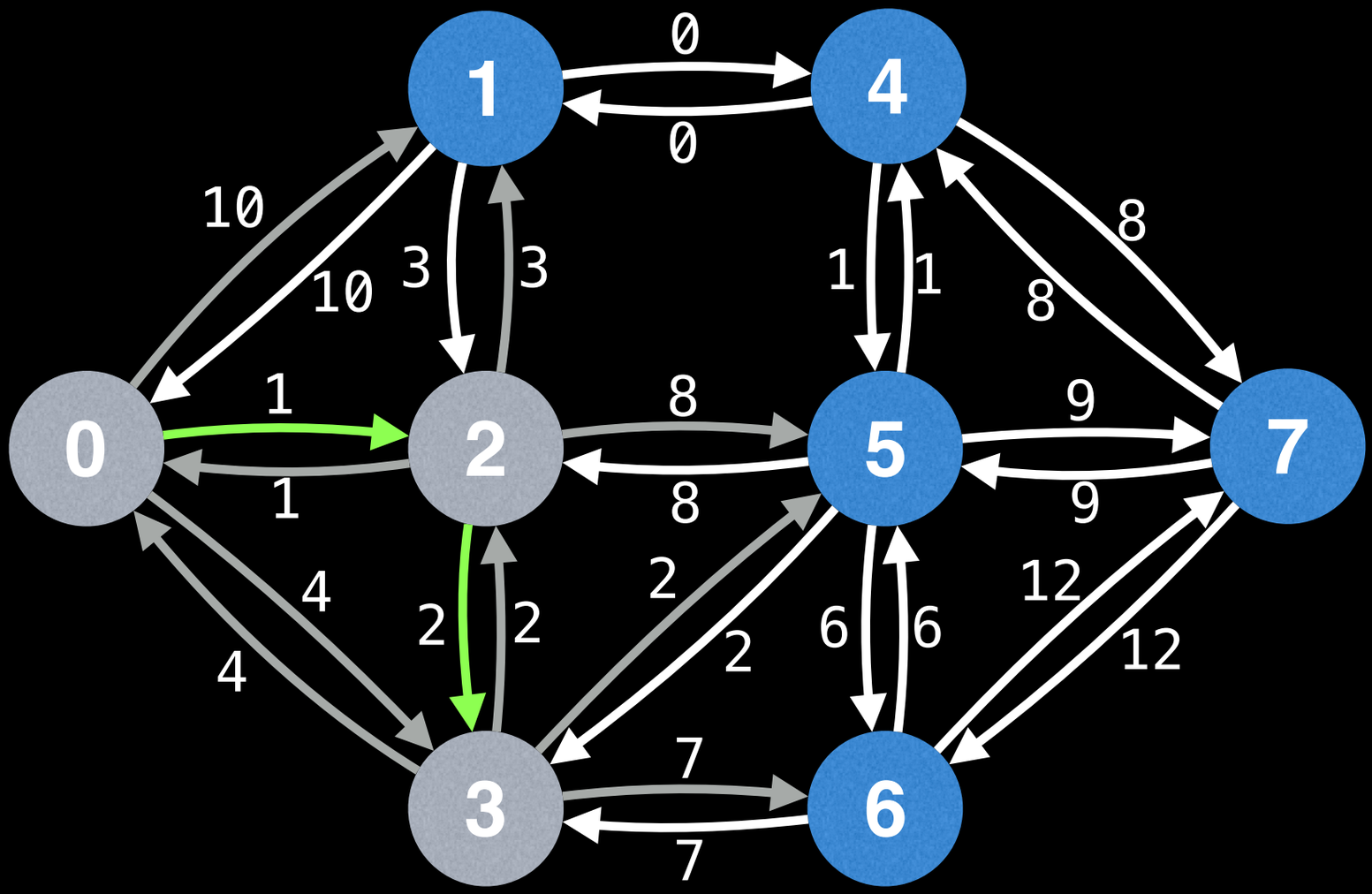
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
→ ~~(2, 3, 2)~~
(3, 5, 2)
(3, 6, 7)

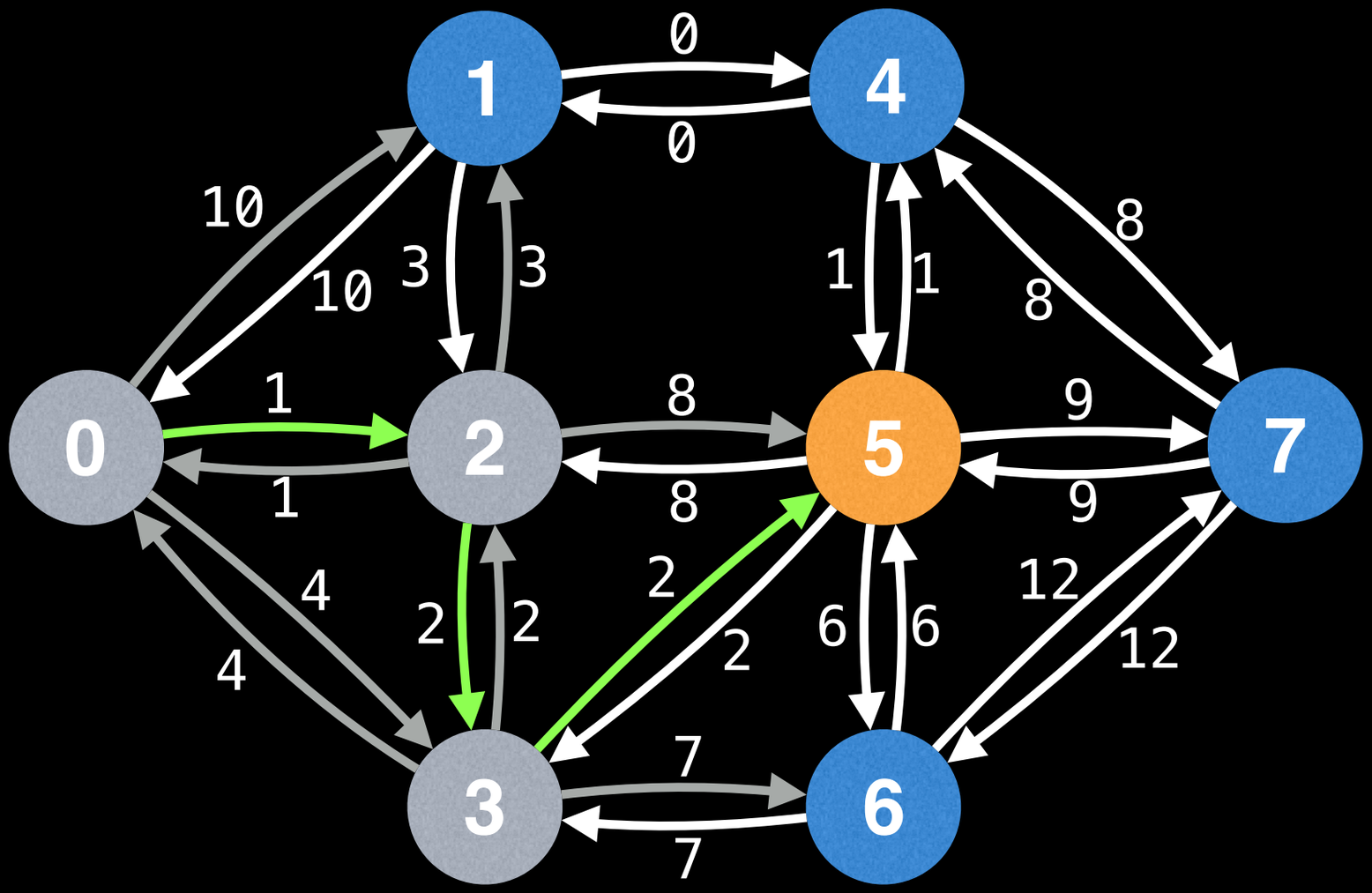
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
→ ~~(2, 3, 2)~~
(3, 5, 2)
(3, 6, 7)

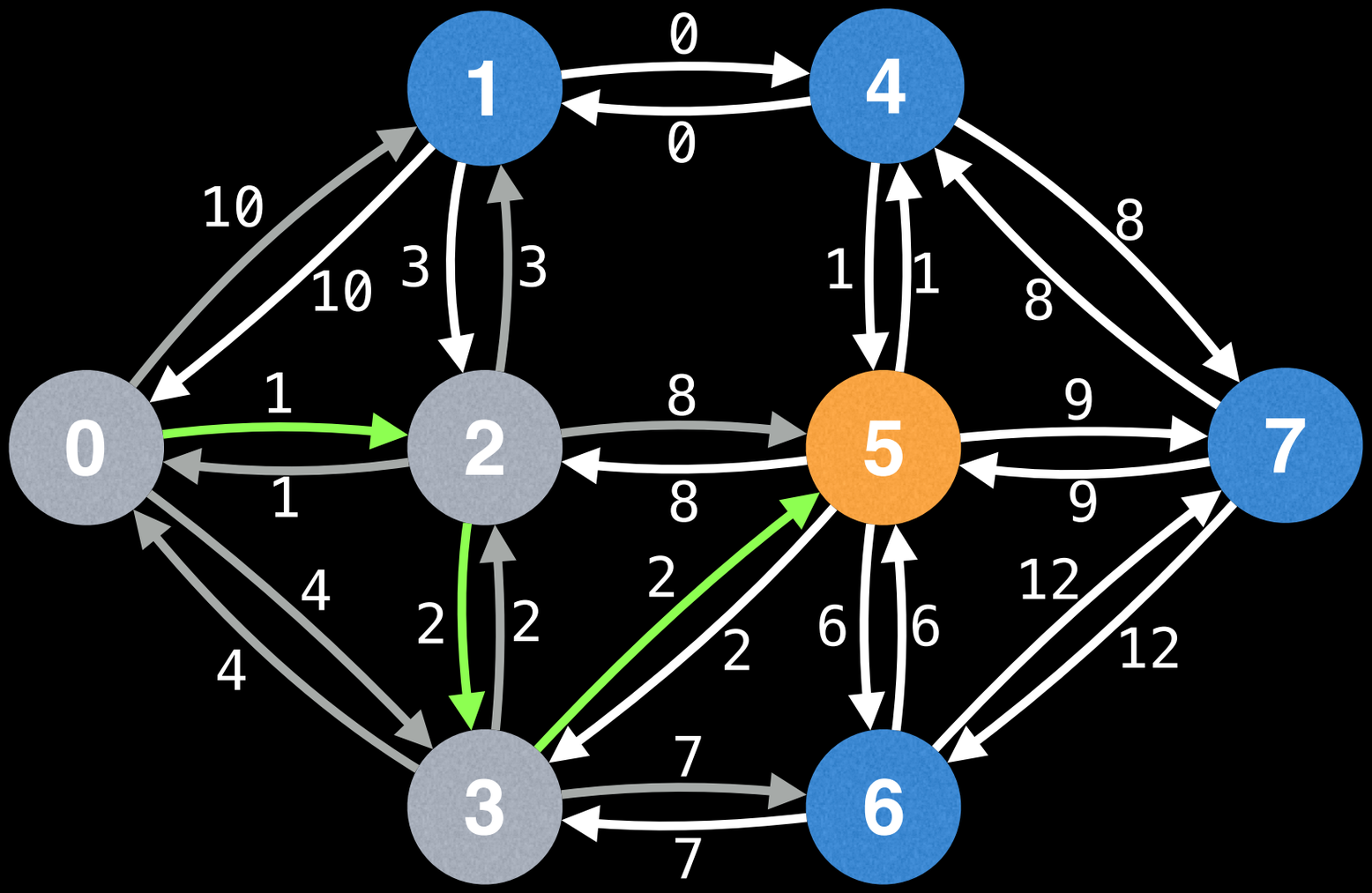
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
(3, 5, 2)
(3, 6, 7)

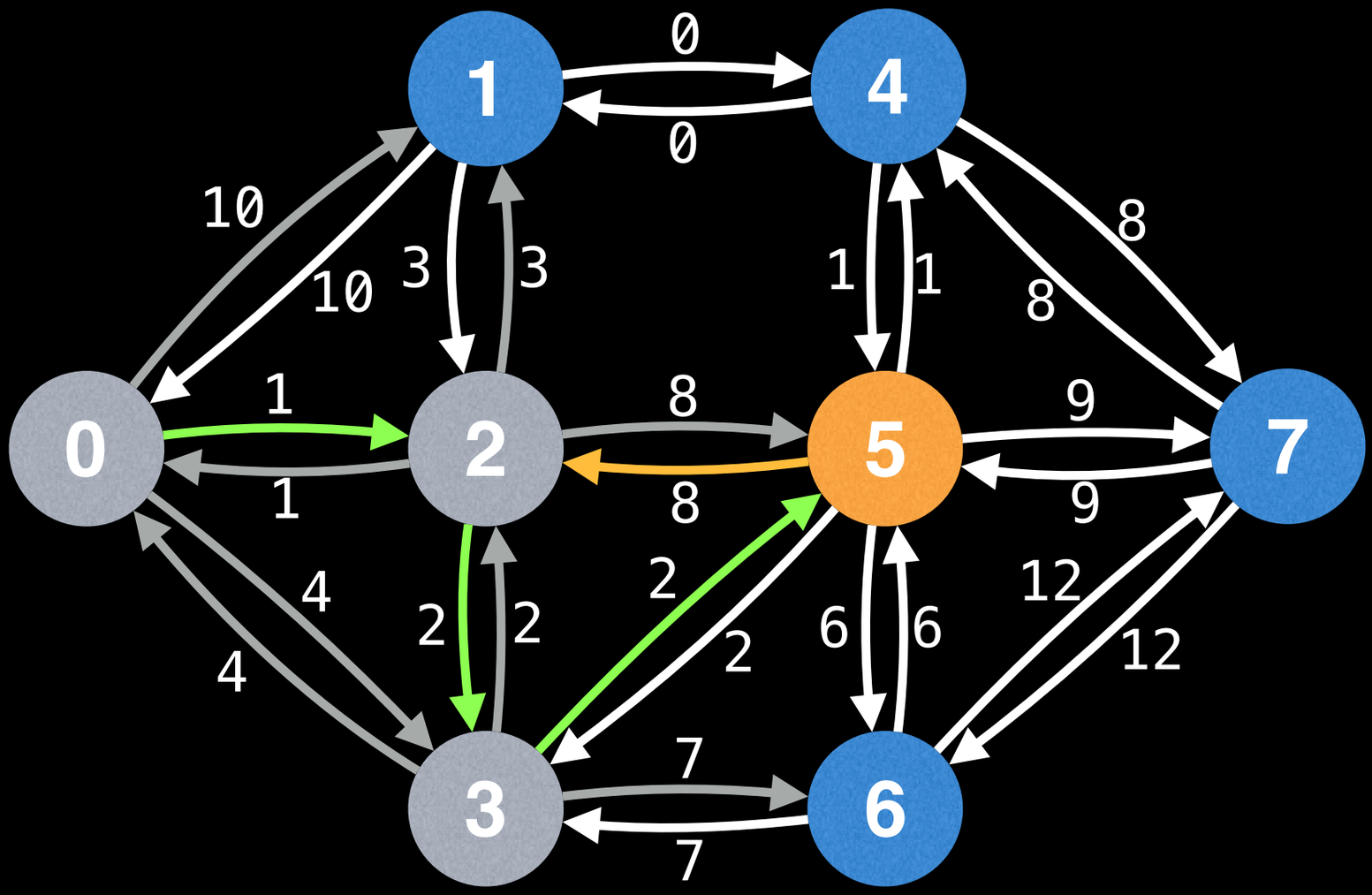
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
→ ~~(3, 5, 2)~~
(3, 6, 7)

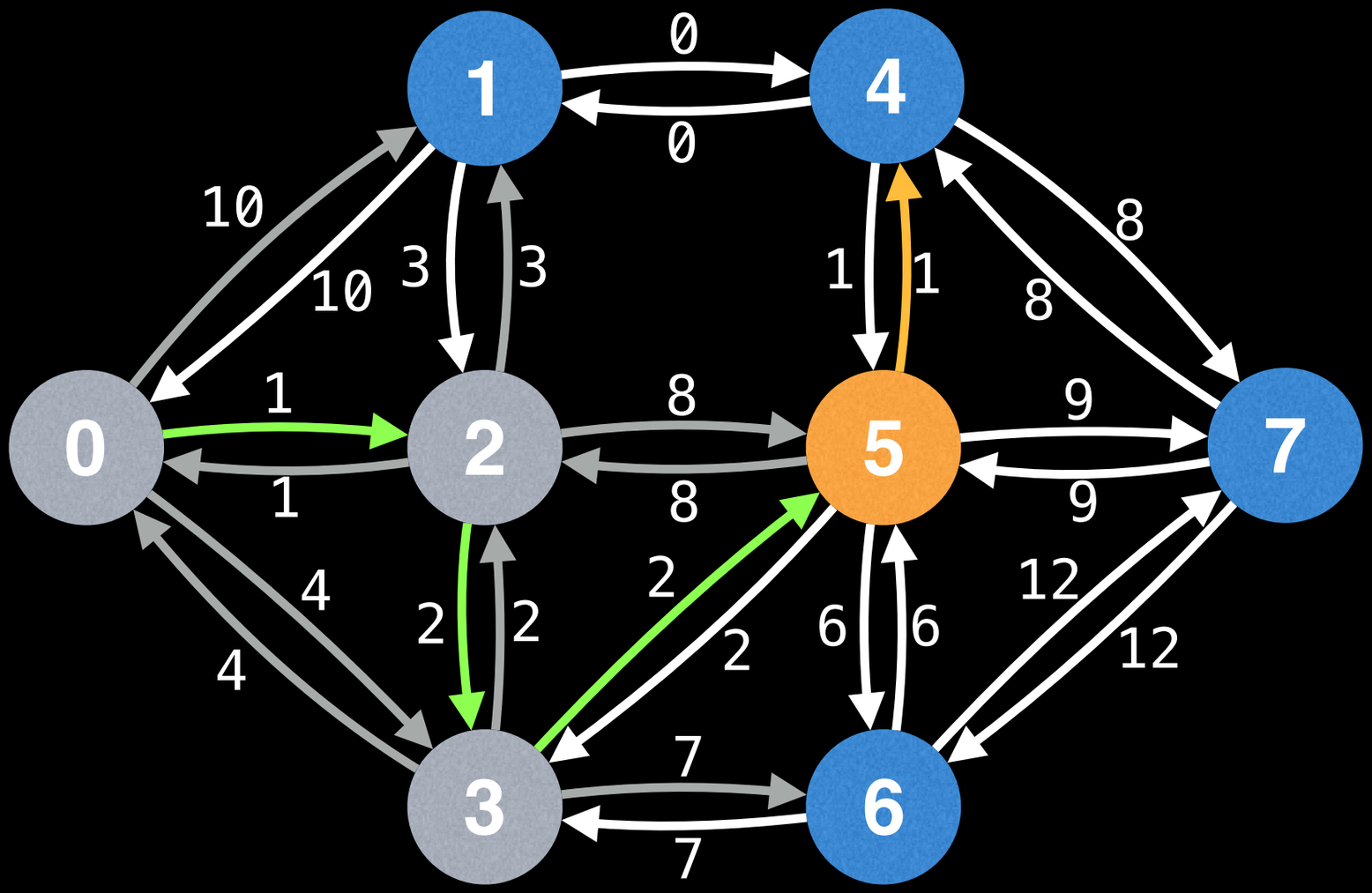
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
→ ~~(3, 5, 2)~~
(3, 6, 7)

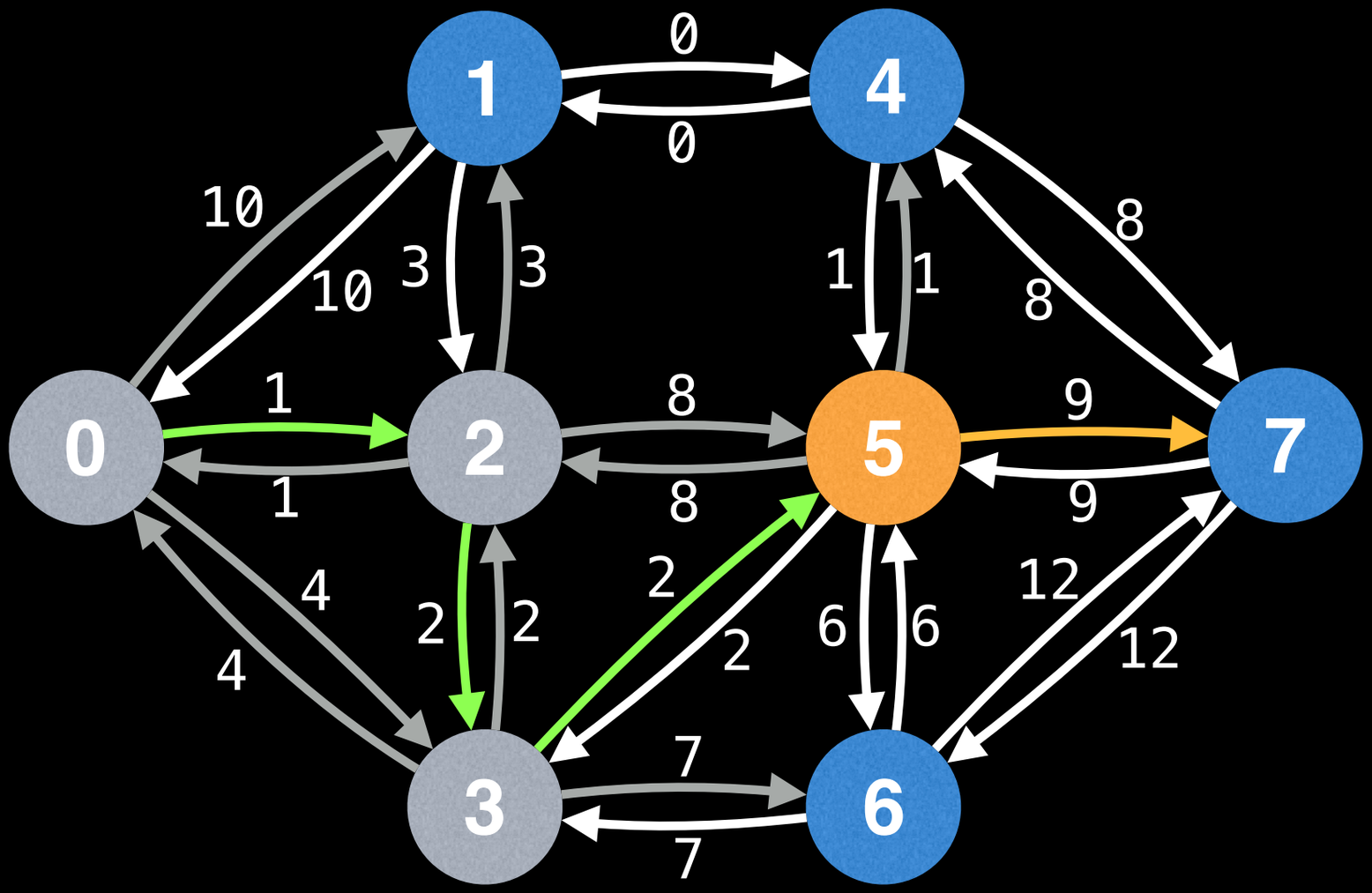
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
→ ~~(3, 5, 2)~~
(3, 6, 7)
(5, 4, 1)

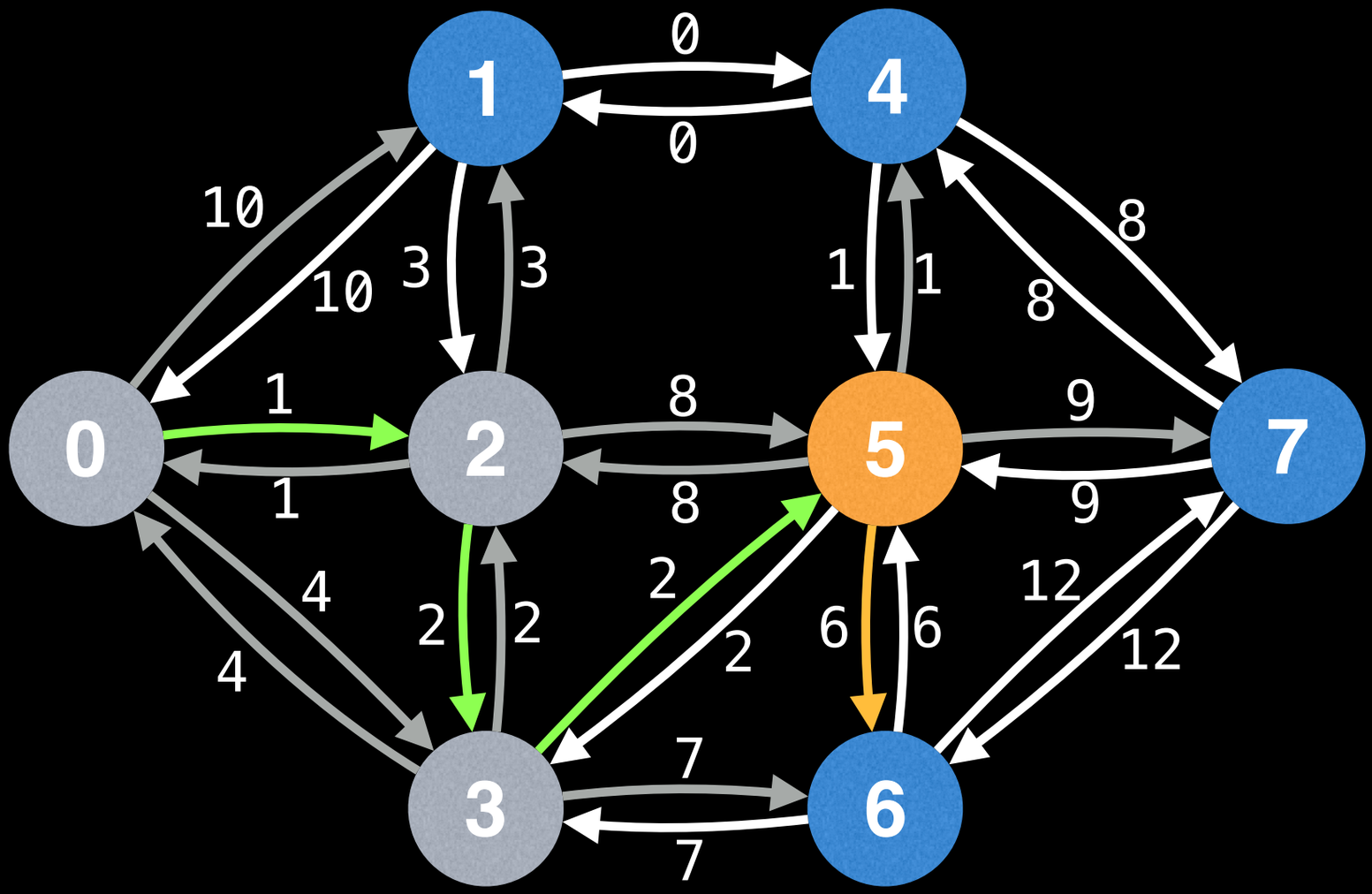
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
→ ~~(3, 5, 2)~~
(3, 6, 7)
(5, 4, 1)
(5, 7, 9)

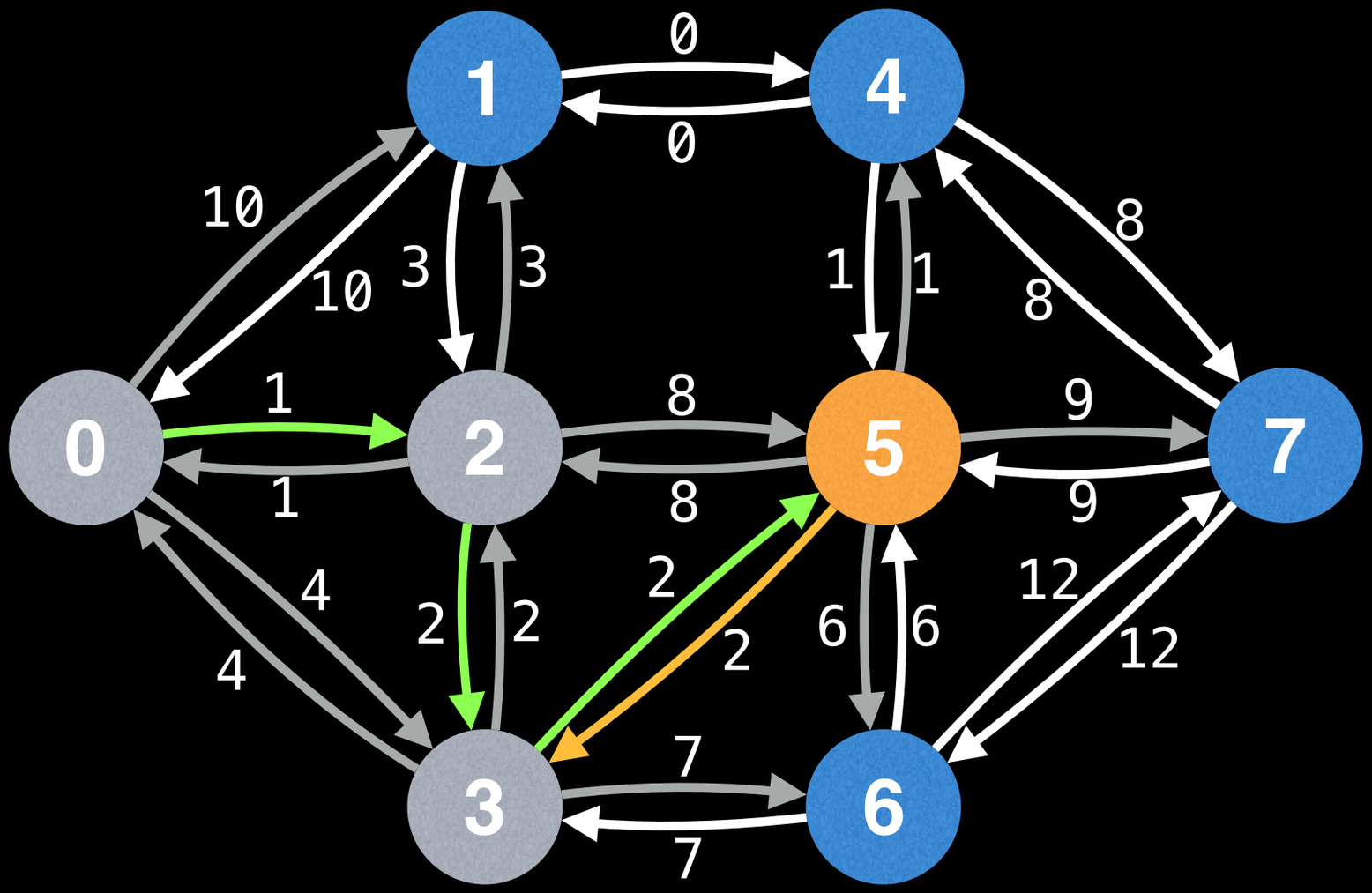
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
→ ~~(3, 5, 2)~~
(3, 6, 7)
(5, 4, 1)
(5, 7, 9)
(5, 6, 6)

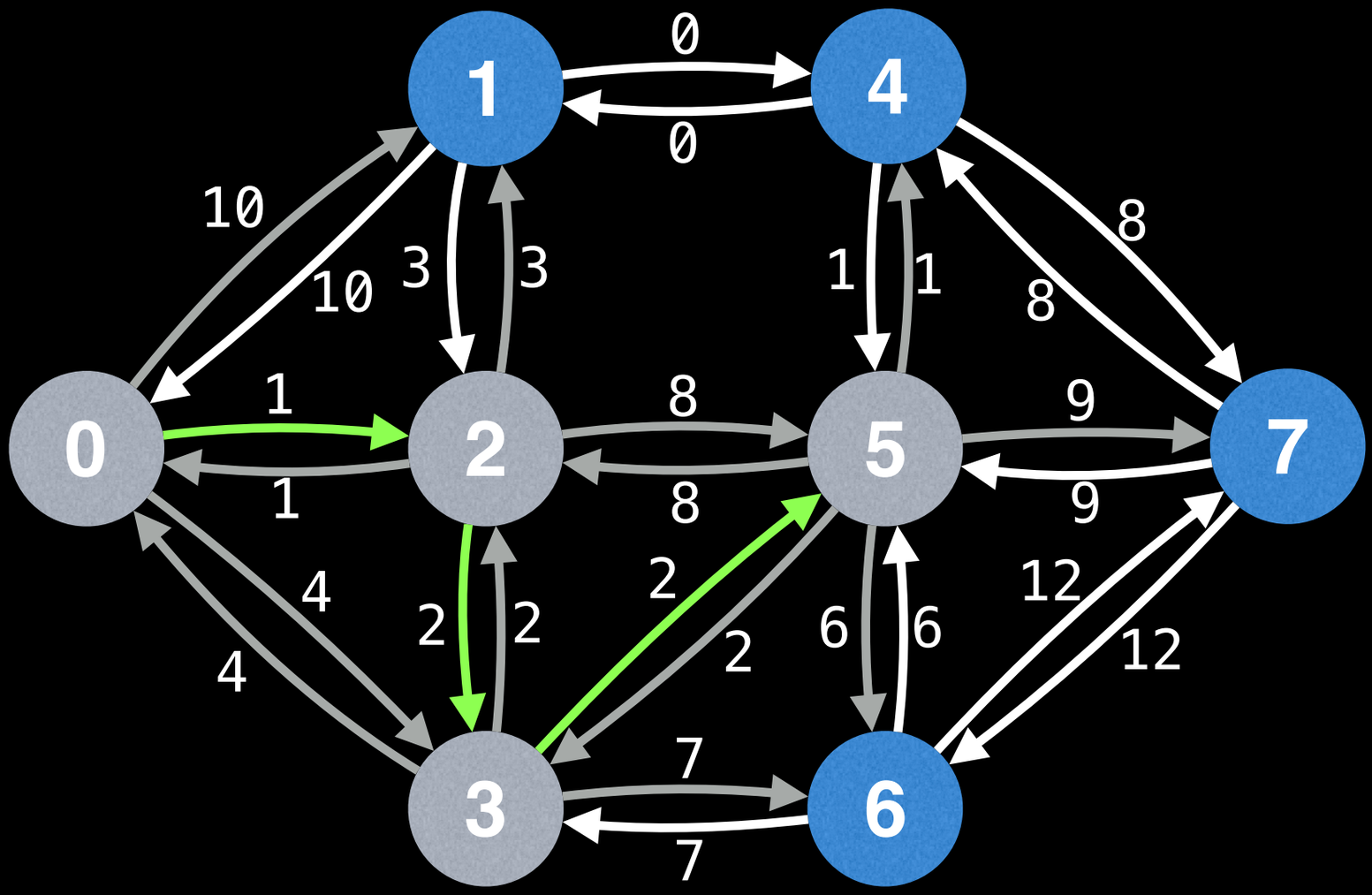
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
→ ~~(3, 5, 2)~~
(3, 6, 7)
(5, 4, 1)
(5, 7, 9)
(5, 6, 6)

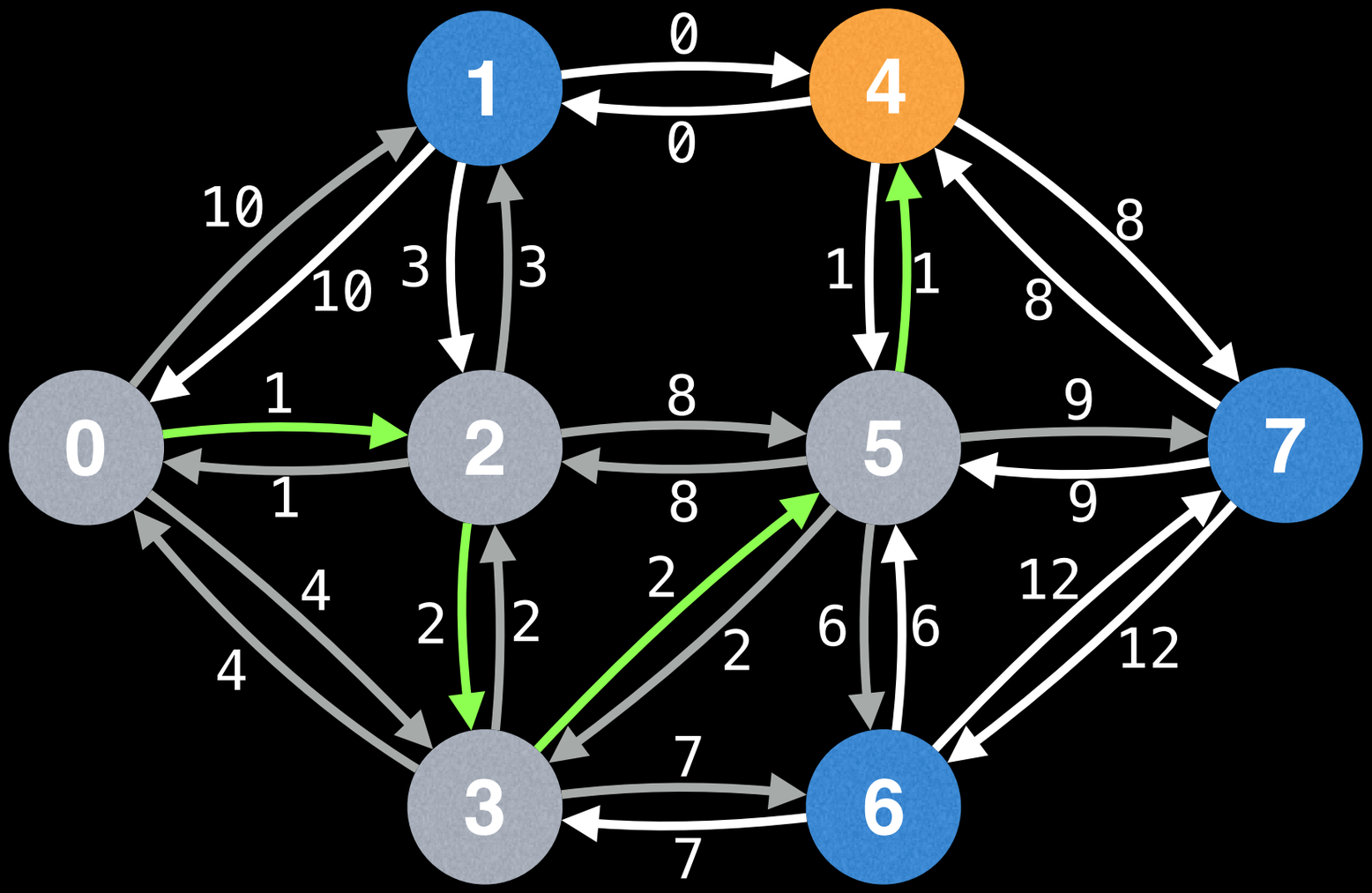
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
→ ~~(3, 5, 2)~~
(3, 6, 7)
(5, 4, 1)
(5, 7, 9)
(5, 6, 6)

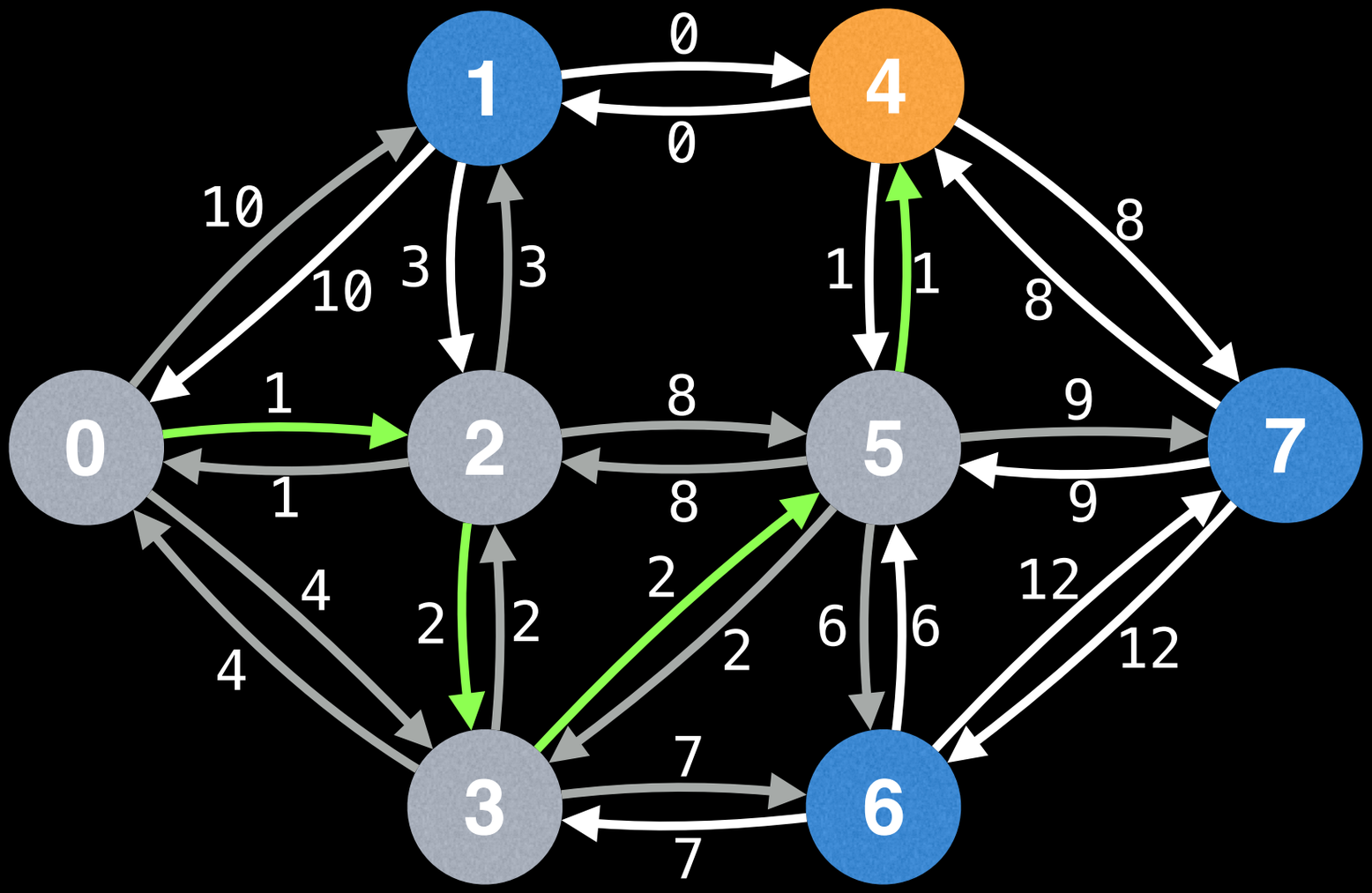
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
→ (5, 4, 1)
(5, 7, 9)
(5, 6, 6)

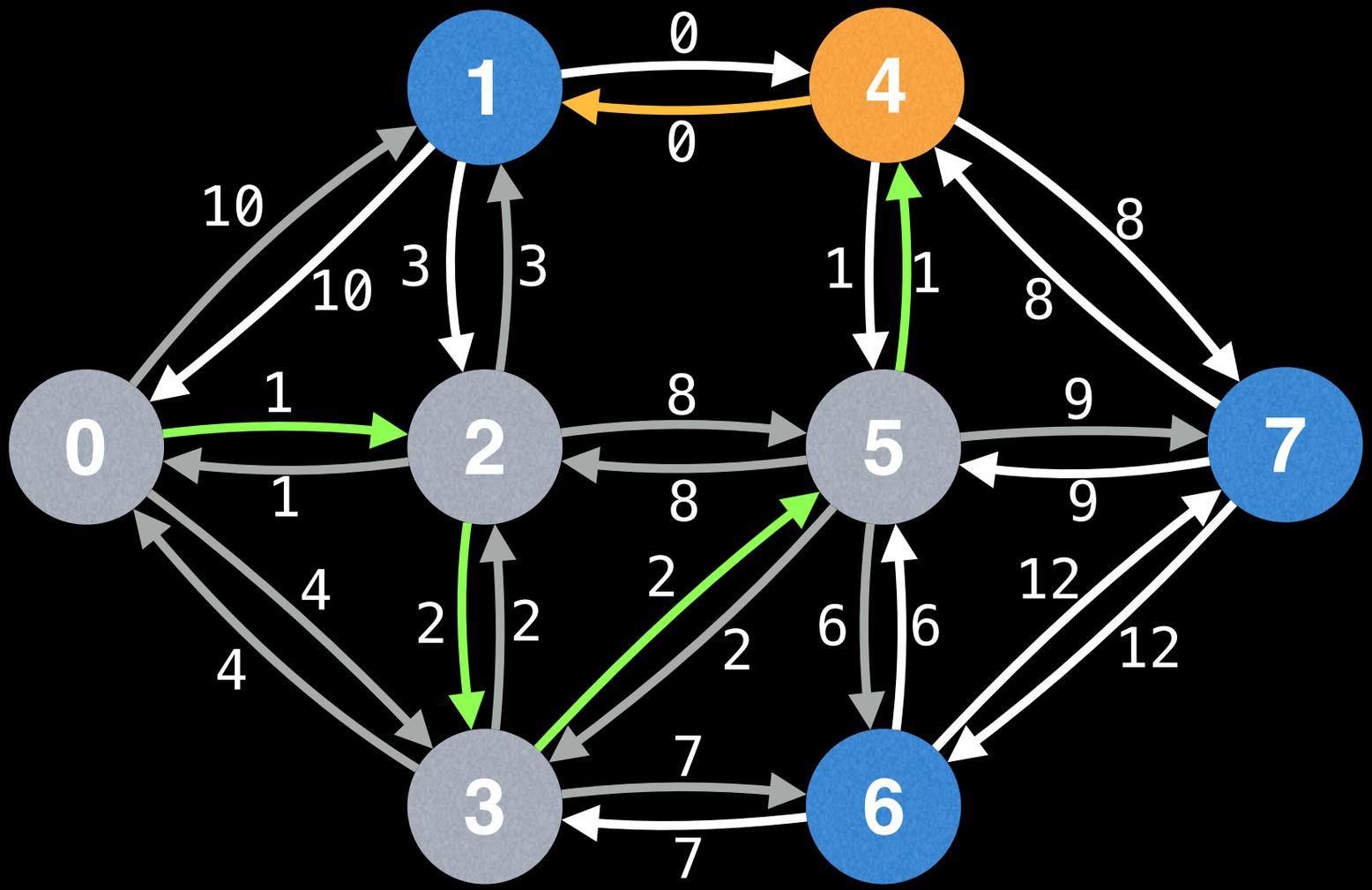
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
→ ~~(5, 4, 1)~~
(5, 7, 9)
(5, 6, 6)

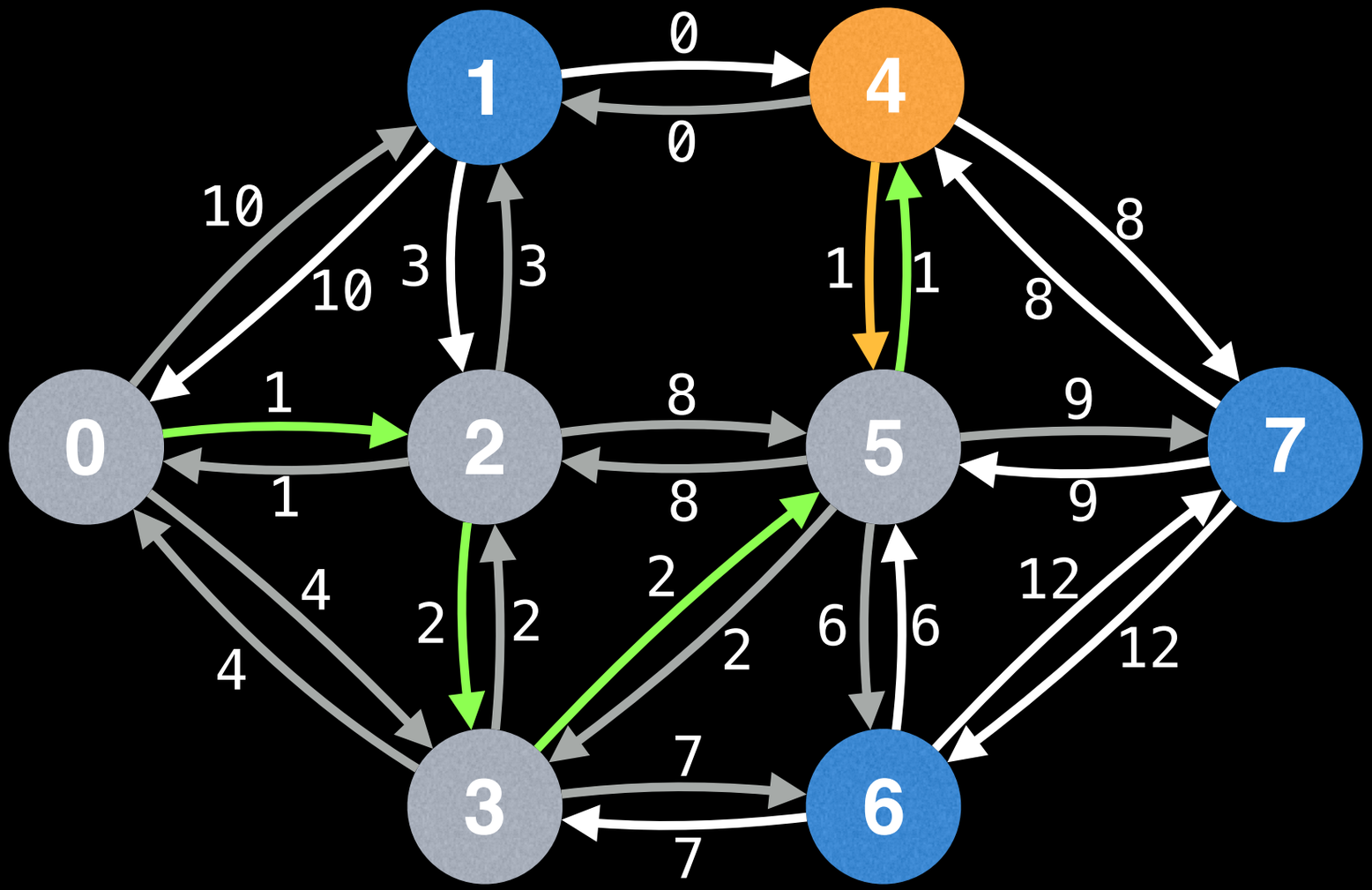
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
→ ~~(5, 4, 1)~~
(5, 7, 9)
(5, 6, 6)
(4, 1, 0)

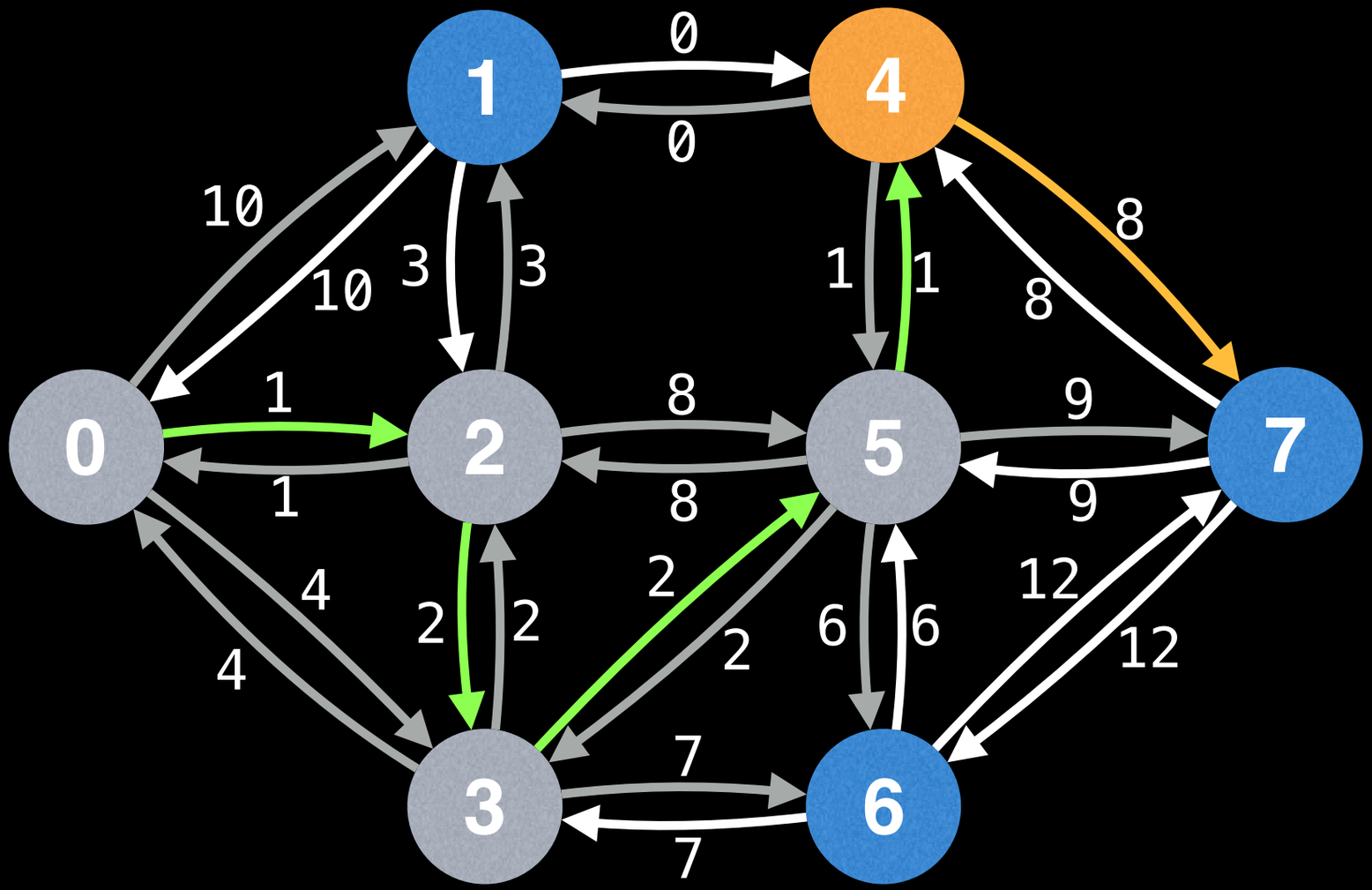
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
→ ~~(5, 4, 1)~~
(5, 7, 9)
(5, 6, 6)
(4, 1, 0)

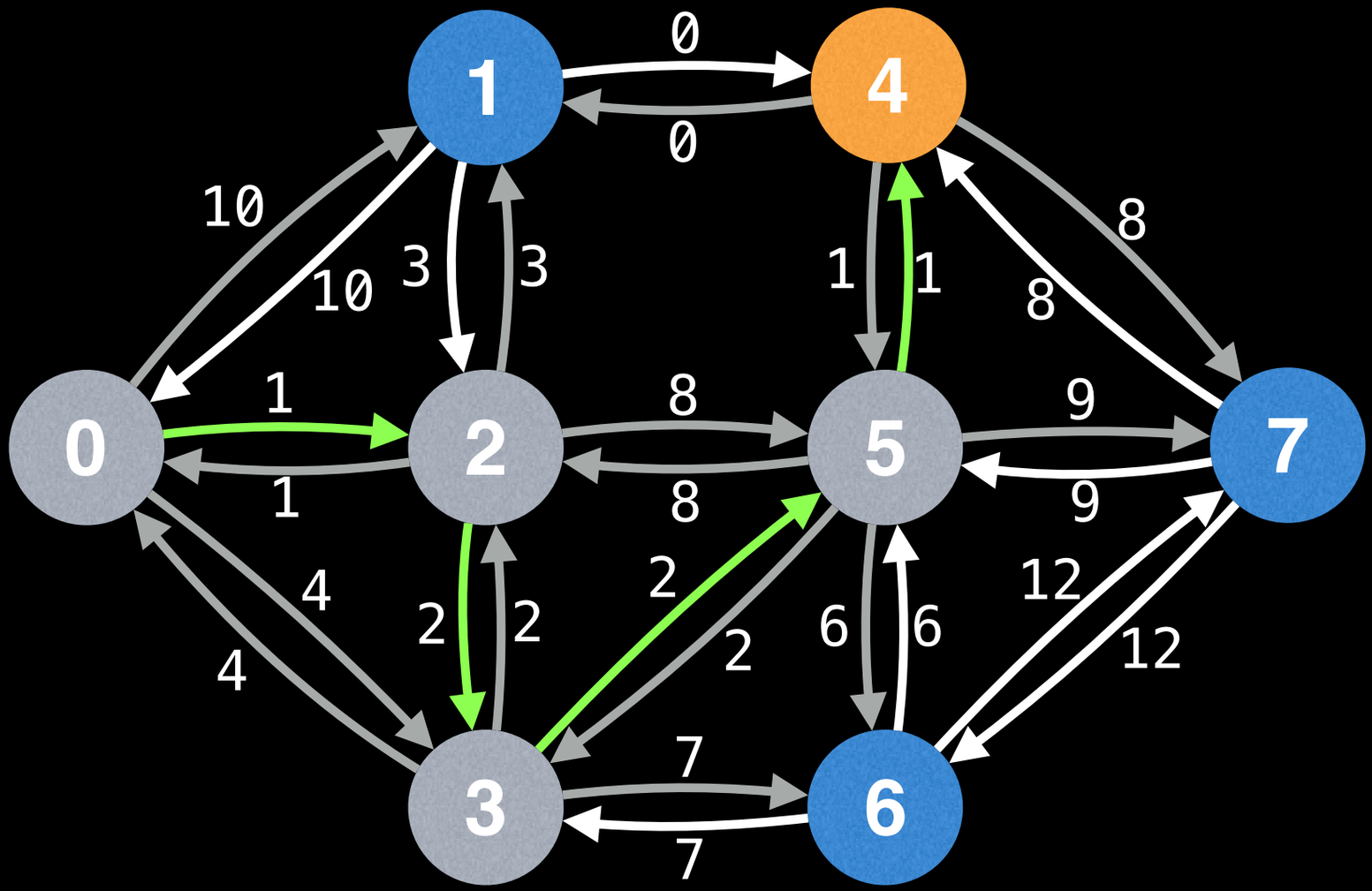
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
→ ~~(5, 4, 1)~~
(5, 7, 9)
(5, 6, 6)
(4, 1, 0)
(4, 7, 8)

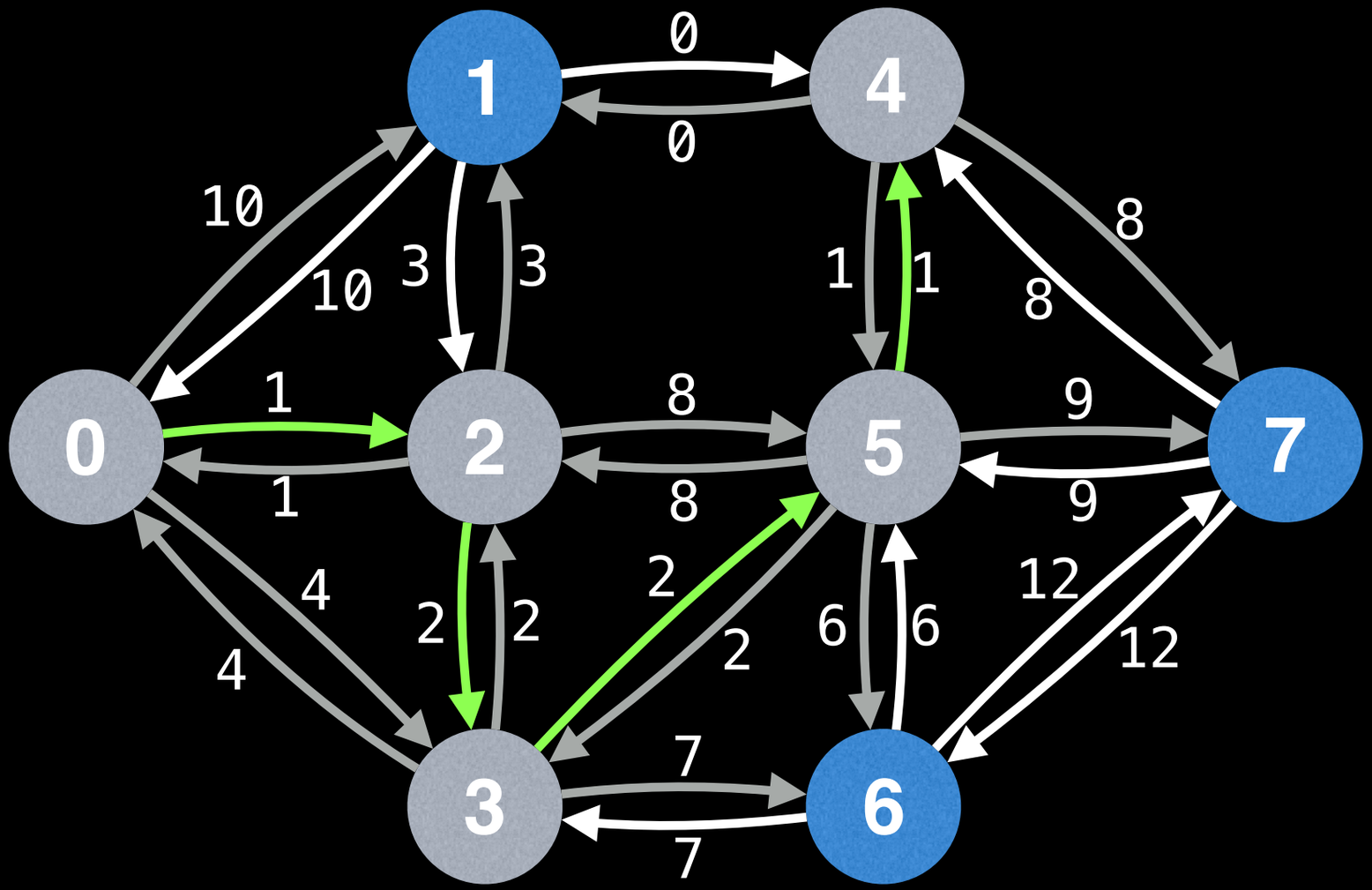
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
→ ~~(5, 4, 1)~~
(5, 7, 9)
(5, 6, 6)
(4, 1, 0)
(4, 7, 8)

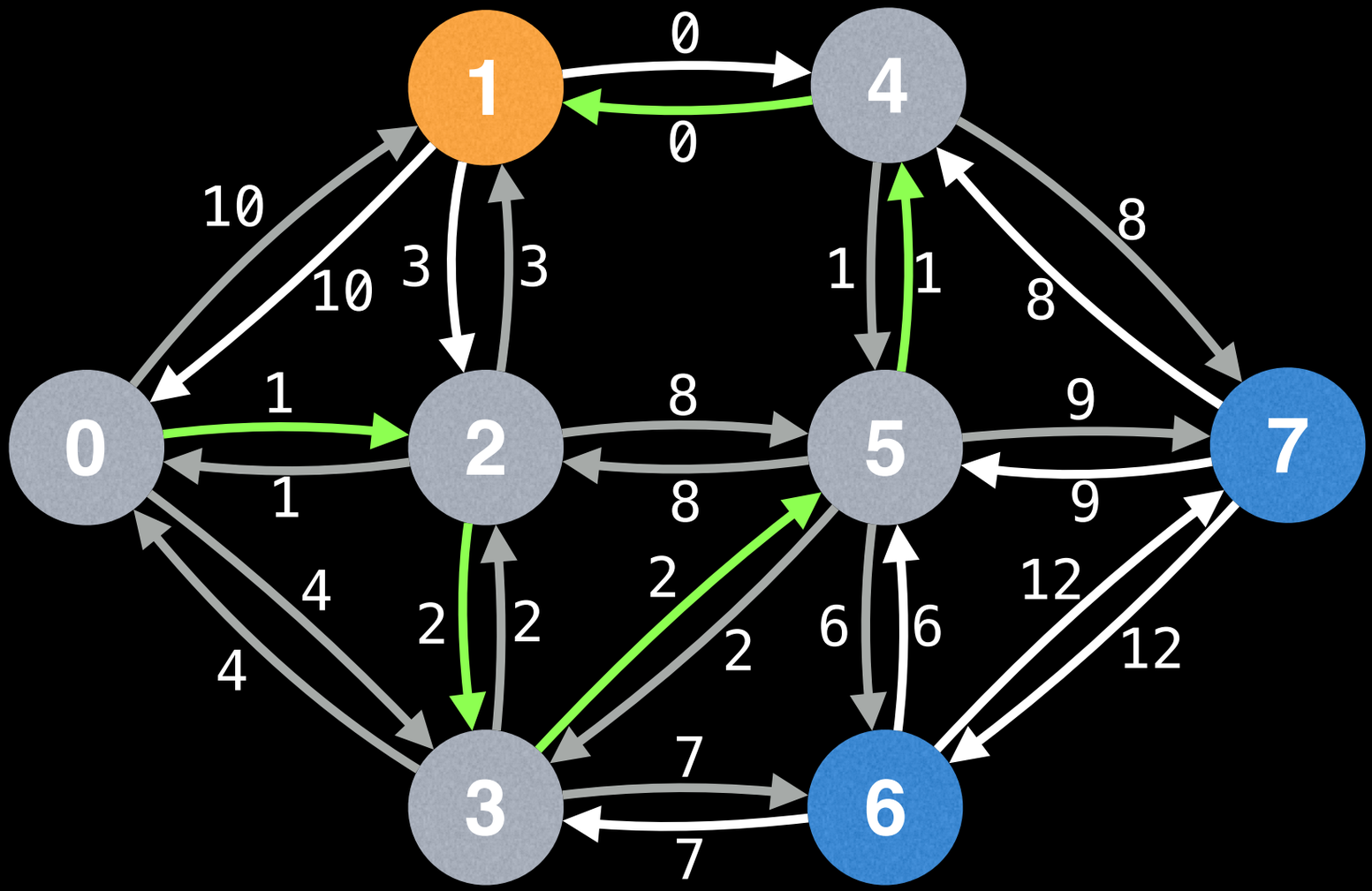
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
→ ~~(5, 4, 1)~~
(5, 7, 9)
(5, 6, 6)
(4, 1, 0)
(4, 7, 8)

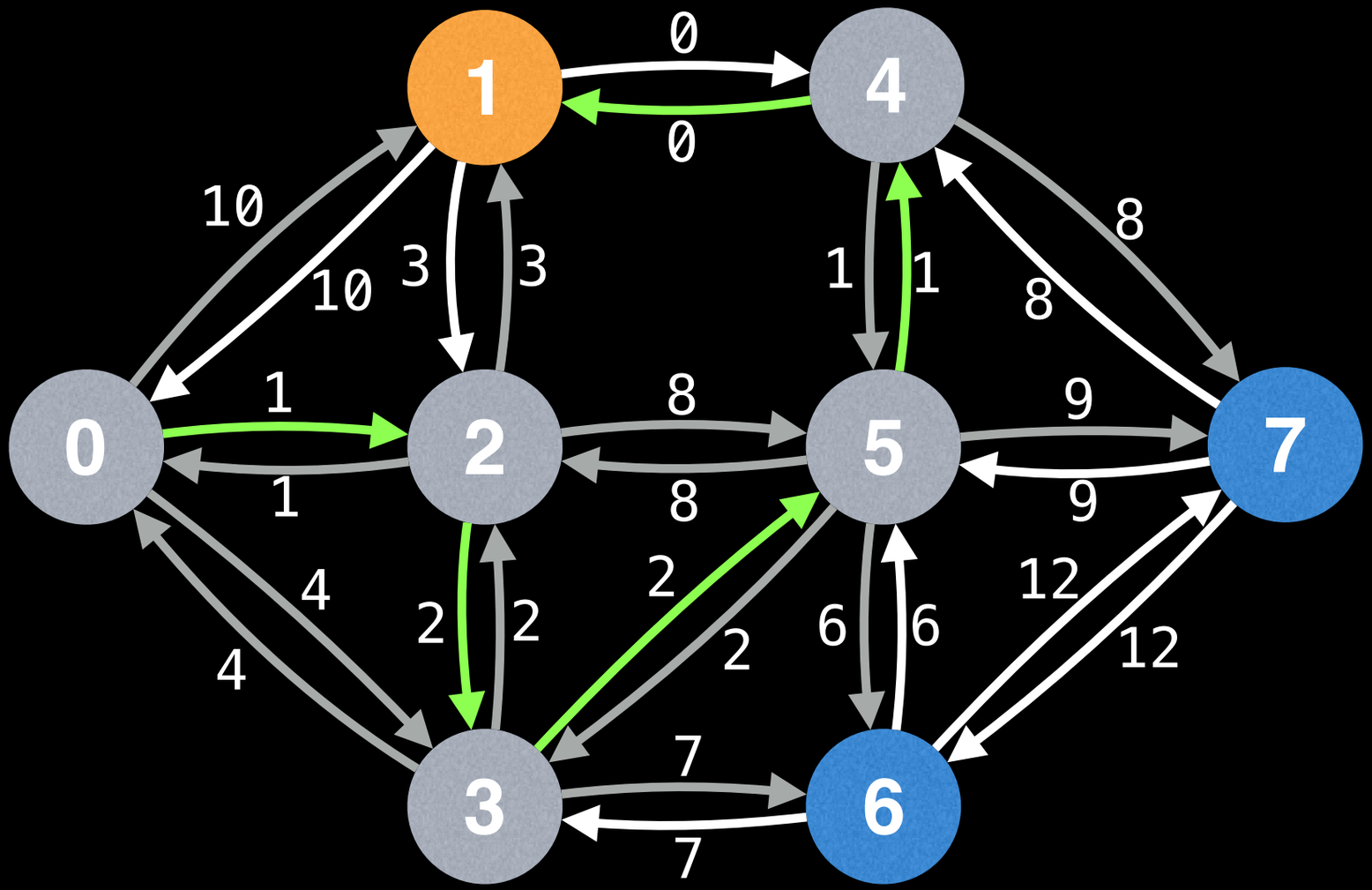
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
~~(5, 4, 1)~~
(5, 7, 9)
(5, 6, 6)
→ (4, 1, 0)
(4, 7, 8)

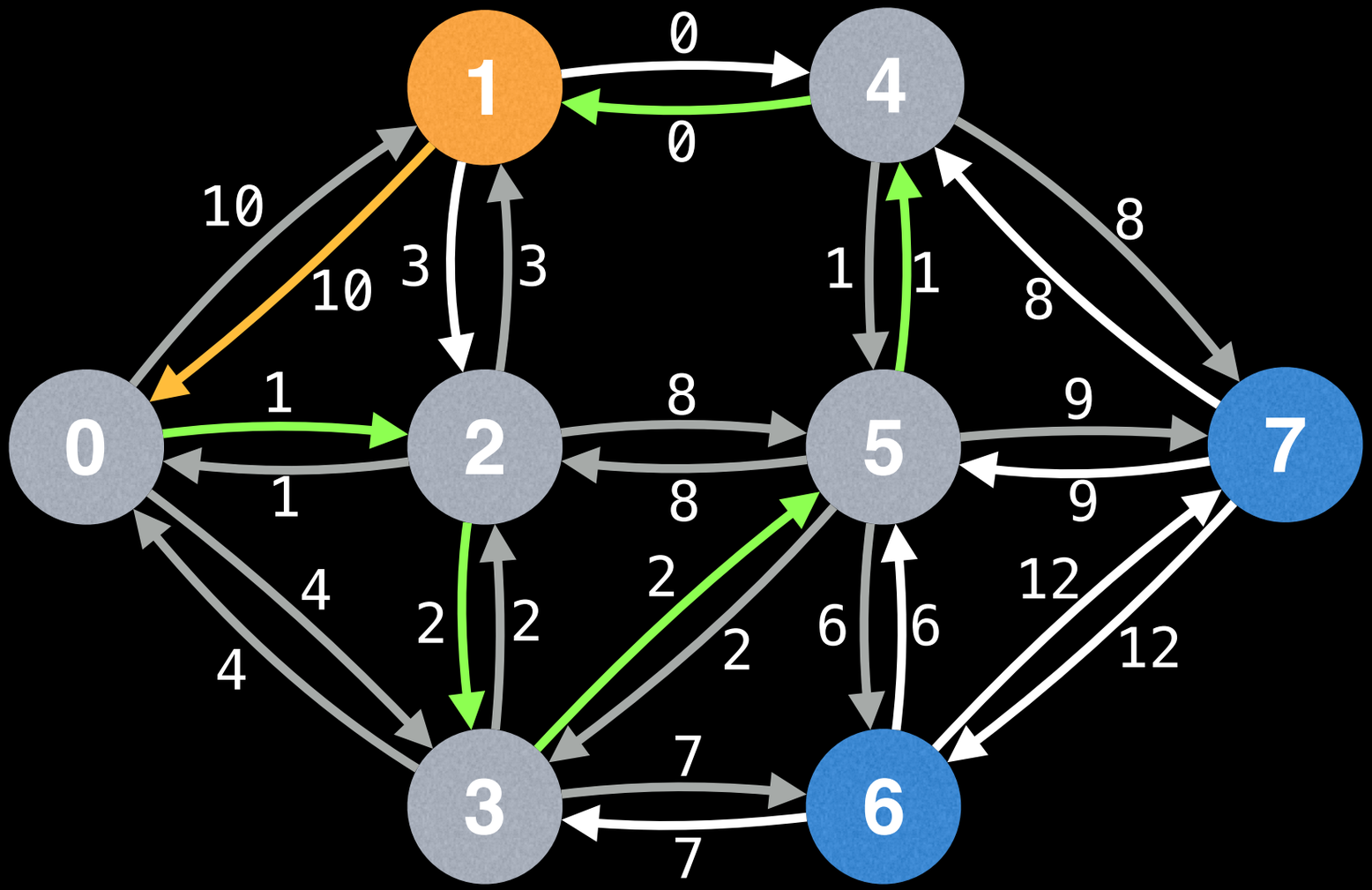
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
~~(5, 4, 1)~~
(5, 7, 9)
(5, 6, 6)
~~(4, 1, 0)~~
(4, 7, 8)

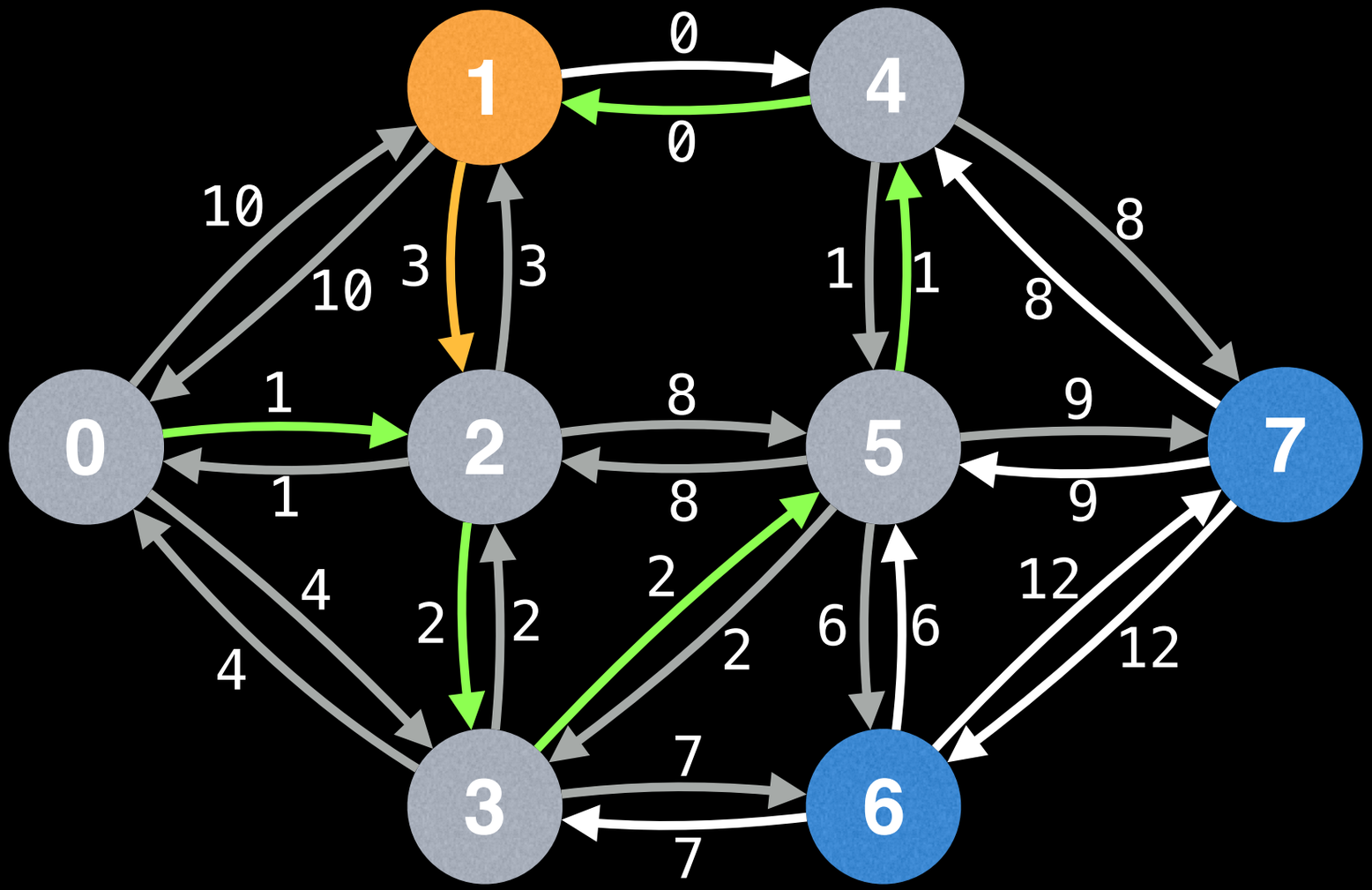
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
~~(5, 4, 1)~~
(5, 7, 9)
(5, 6, 6)
~~(4, 1, 0)~~
(4, 7, 8)

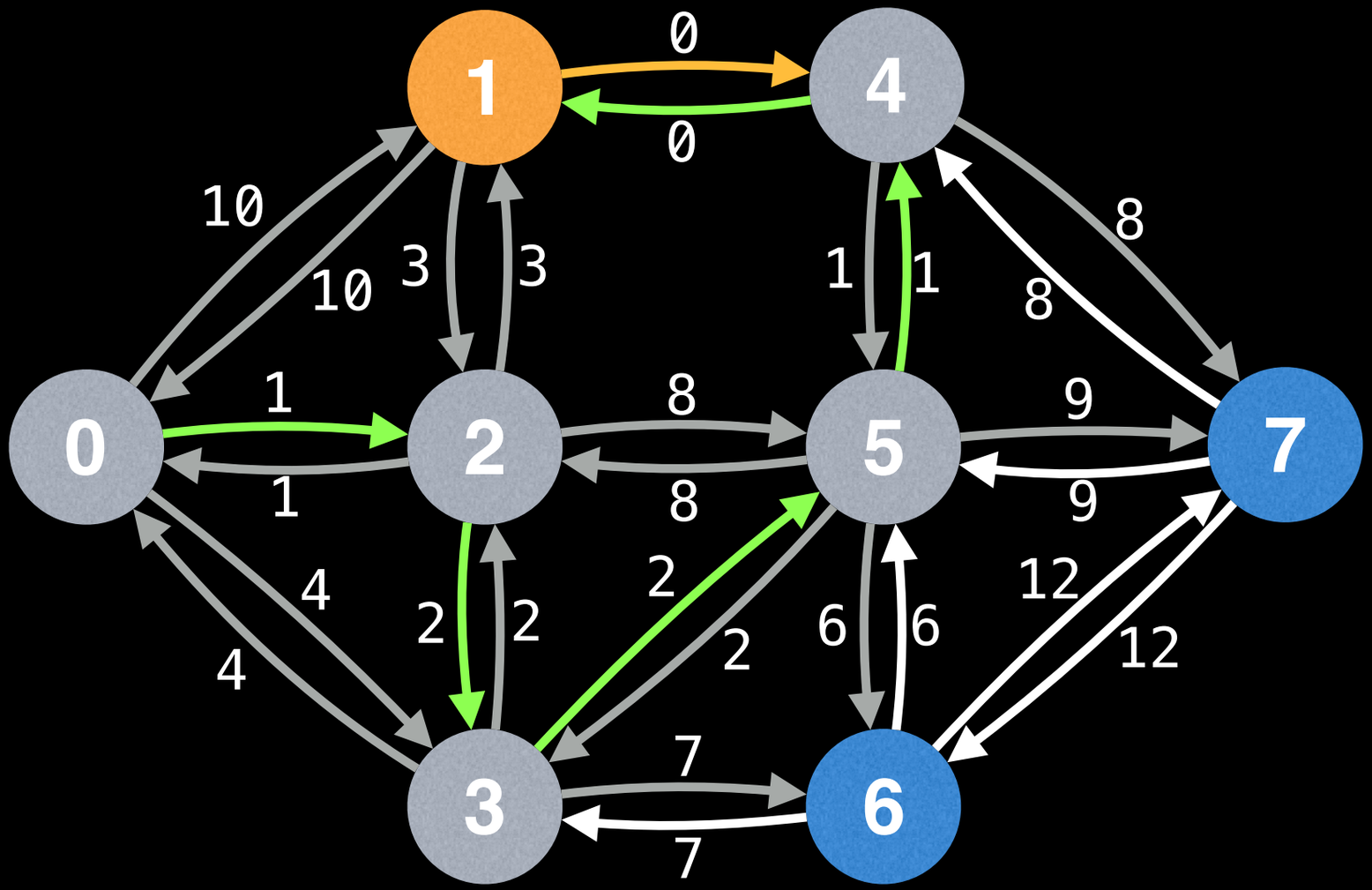
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
~~(5, 4, 1)~~
(5, 7, 9)
(5, 6, 6)
~~(4, 1, 0)~~
(4, 7, 8)

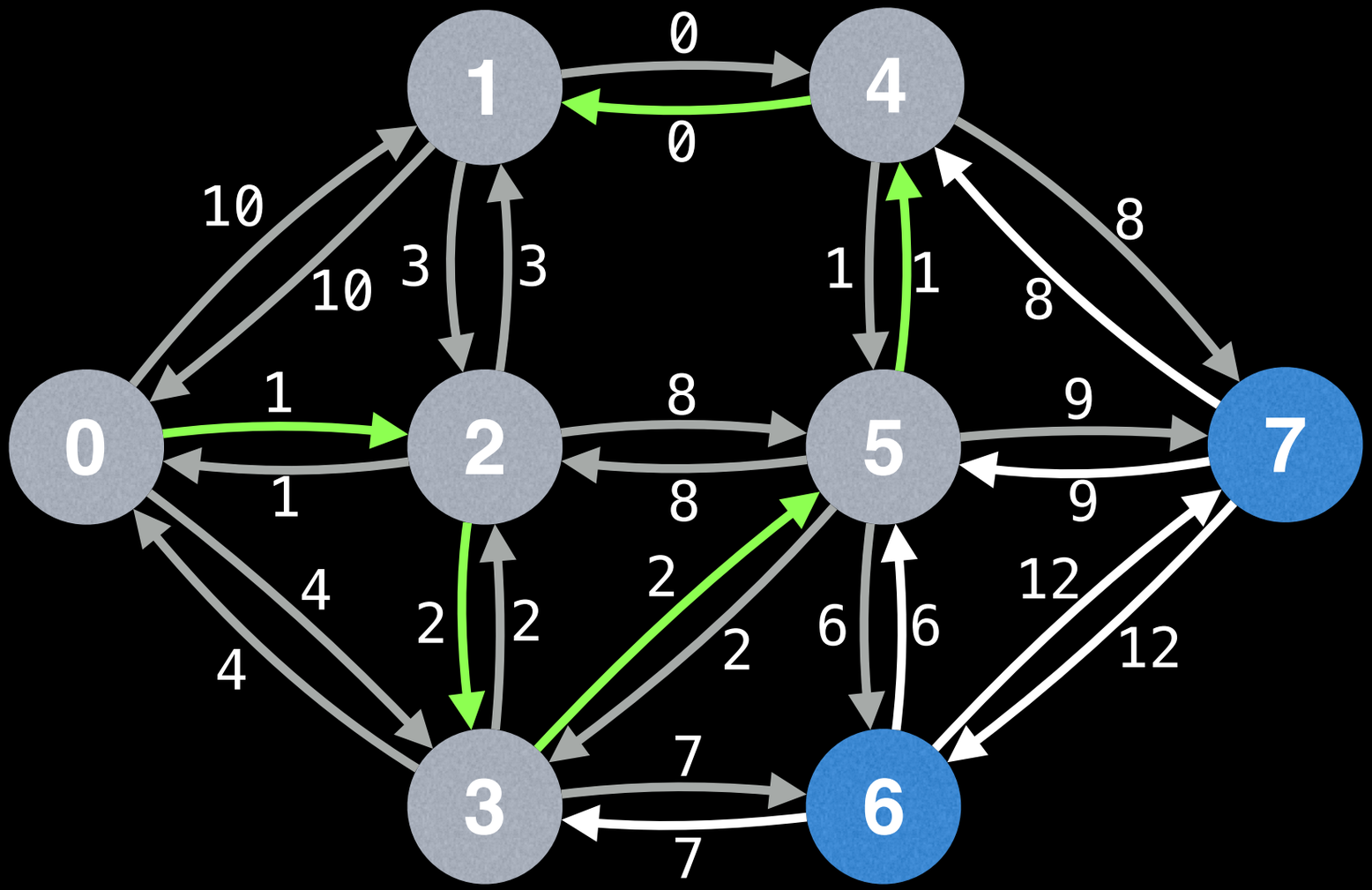
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
~~(5, 4, 1)~~
(5, 7, 9)
(5, 6, 6)
→ ~~(4, 1, 0)~~
(4, 7, 8)

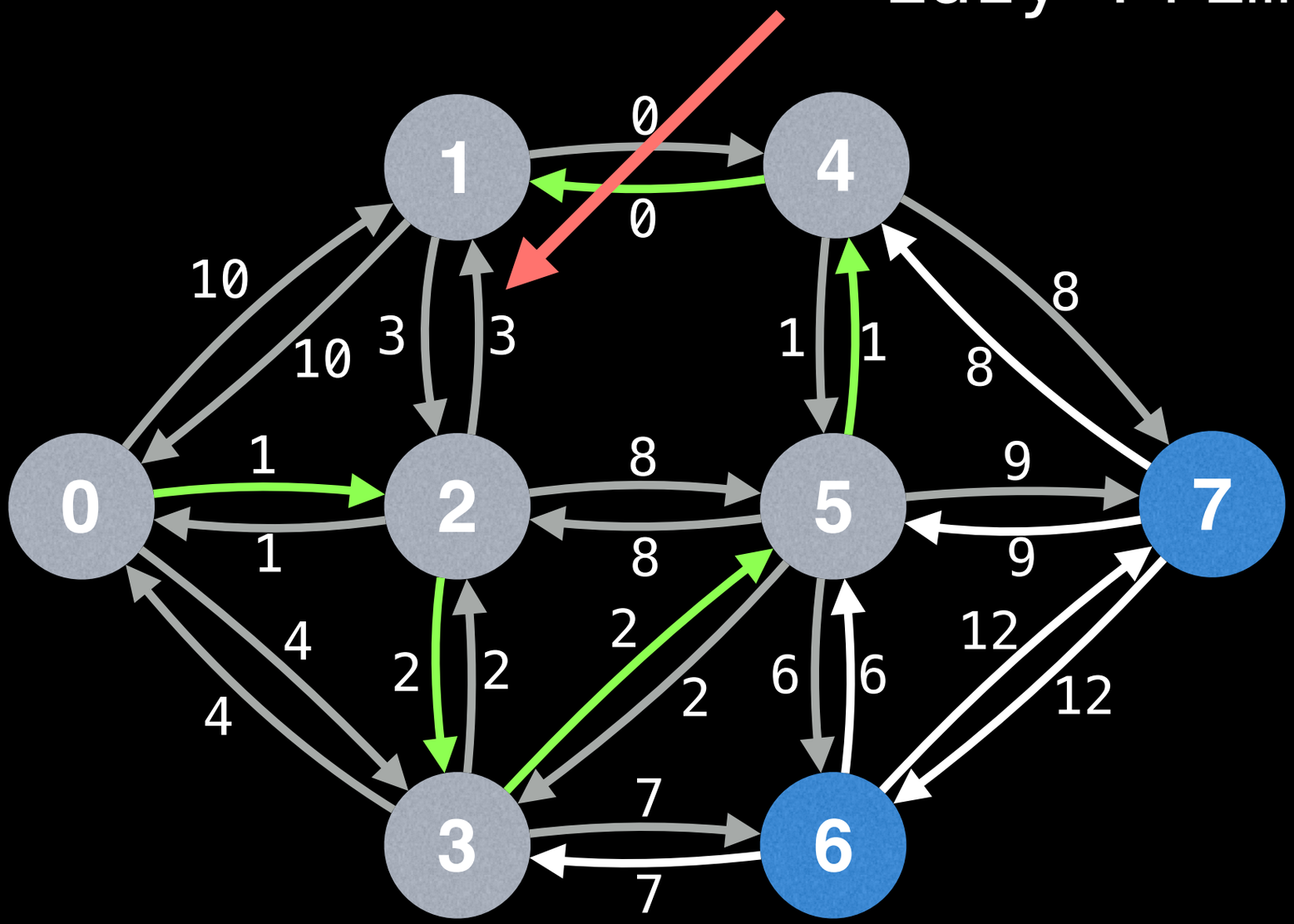
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
(2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
~~(5, 4, 1)~~
(5, 7, 9)
(5, 6, 6)
~~(4, 1, 0)~~
(4, 7, 8)

Lazy Prim's

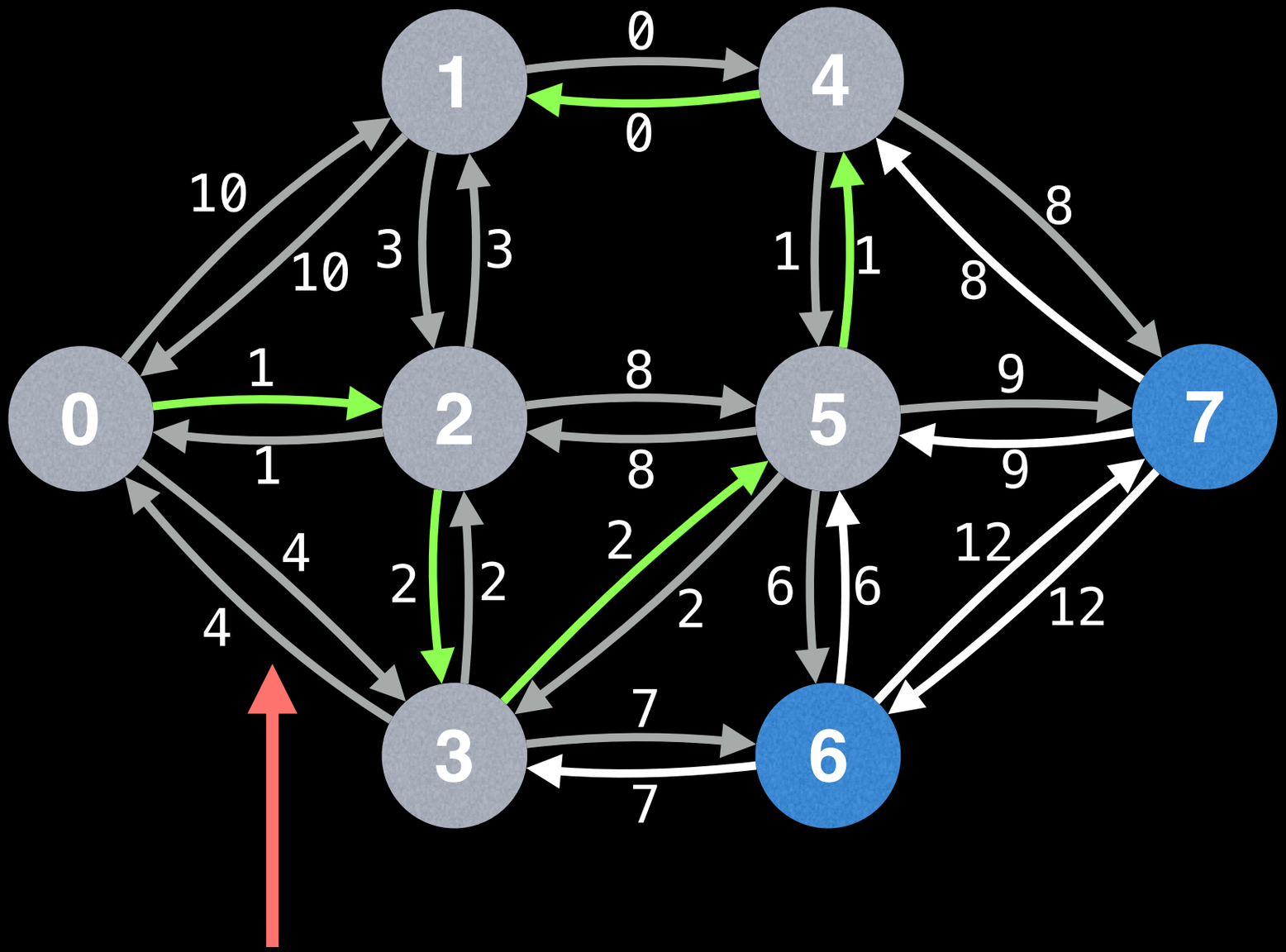


Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
→ (2, 1, 3)
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
~~(5, 4, 1)~~
(5, 7, 9)
(5, 6, 6)
~~(4, 1, 0)~~
(4, 7, 8)

The edge (2, 1, 3) is stale, poll again.

Lazy Prim's

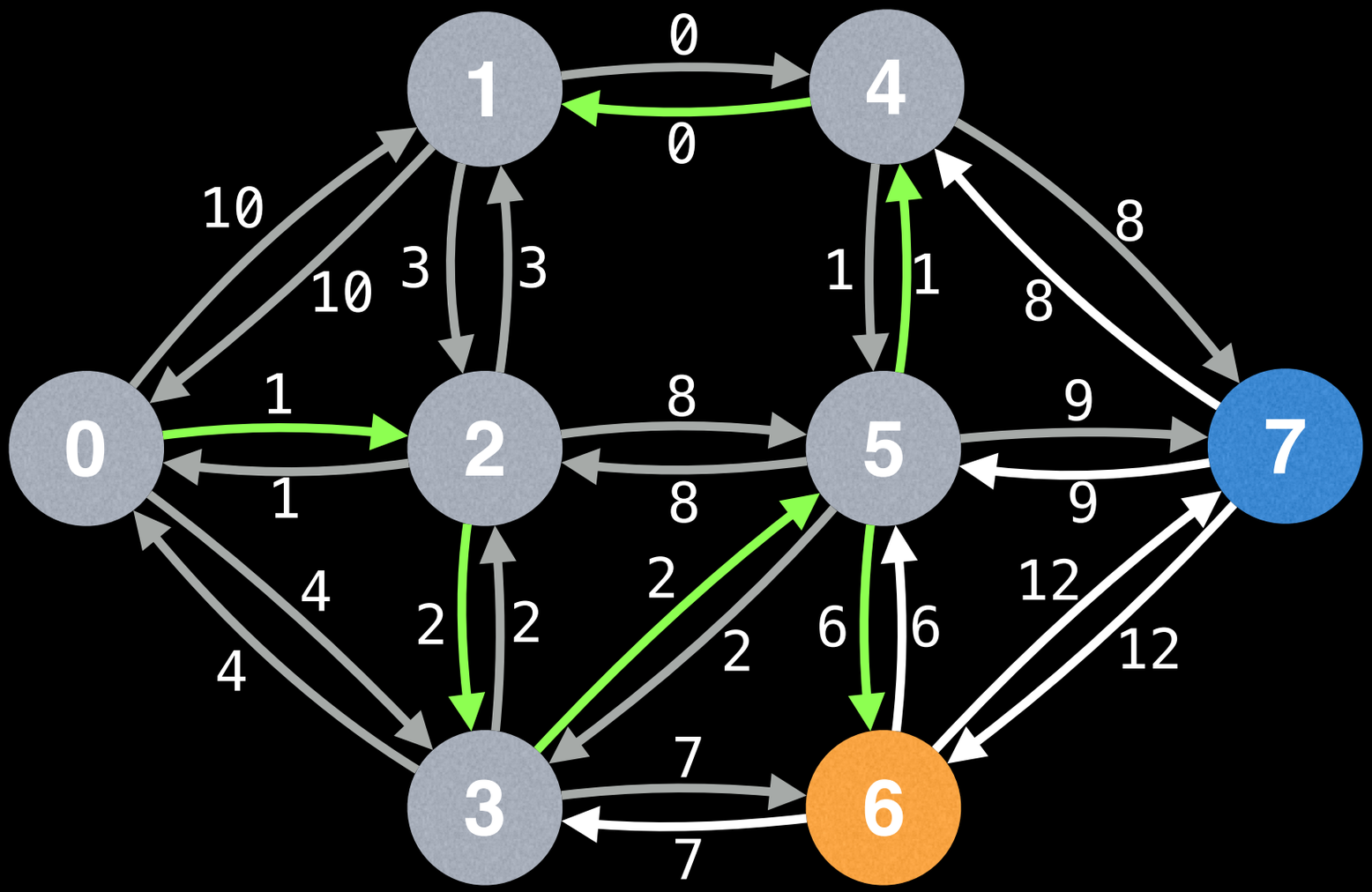


Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
(0, 3, 4)
~~(2, 1, 3)~~
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
~~(5, 4, 1)~~
(5, 7, 9)
(5, 6, 6)
~~(4, 1, 0)~~
(4, 7, 8)

The edge (0, 3, 4) is also stale, poll again.

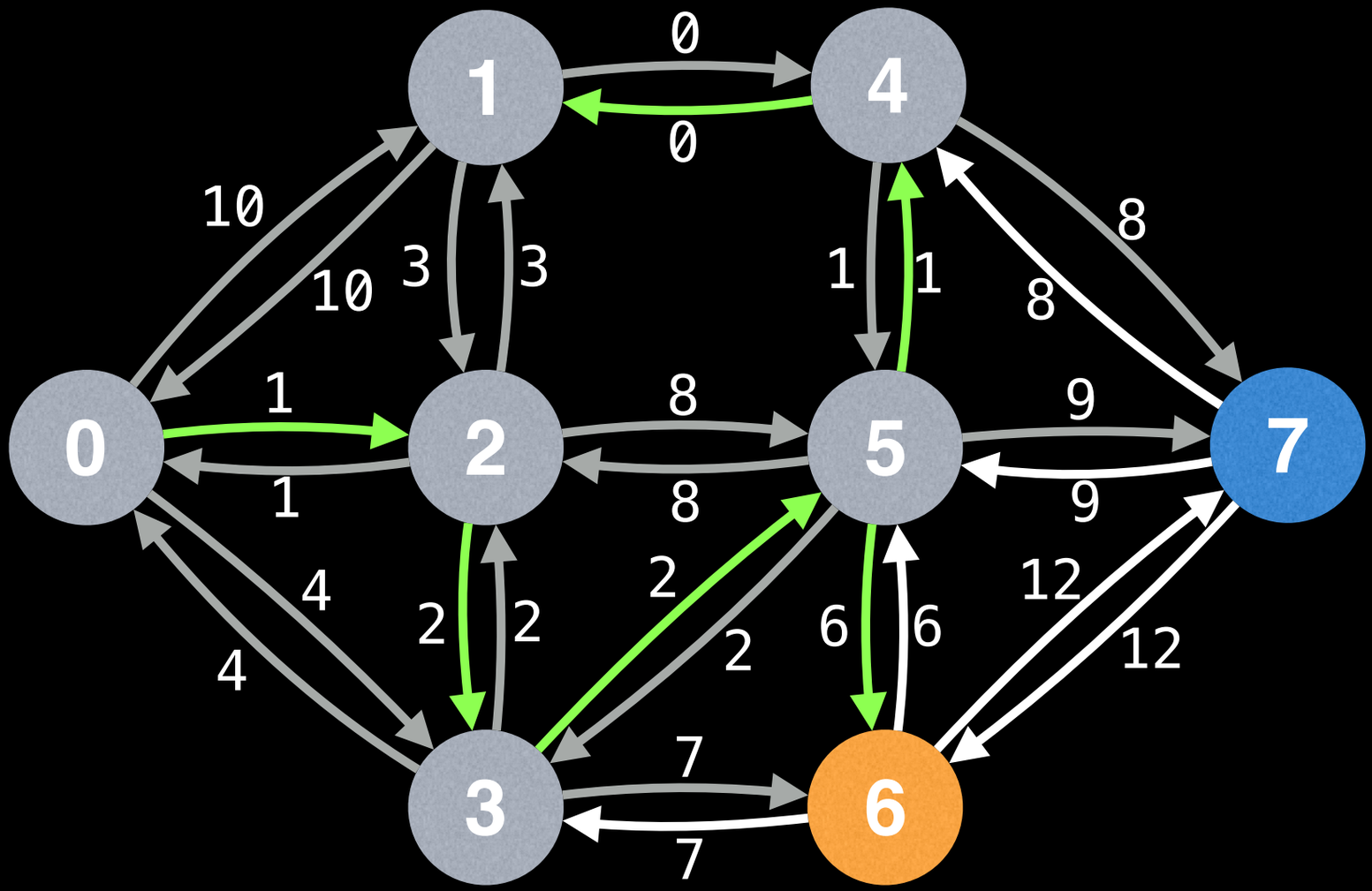
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
~~(5, 4, 1)~~
(5, 7, 9)
→ (5, 6, 6)
~~(4, 1, 0)~~
(4, 7, 8)

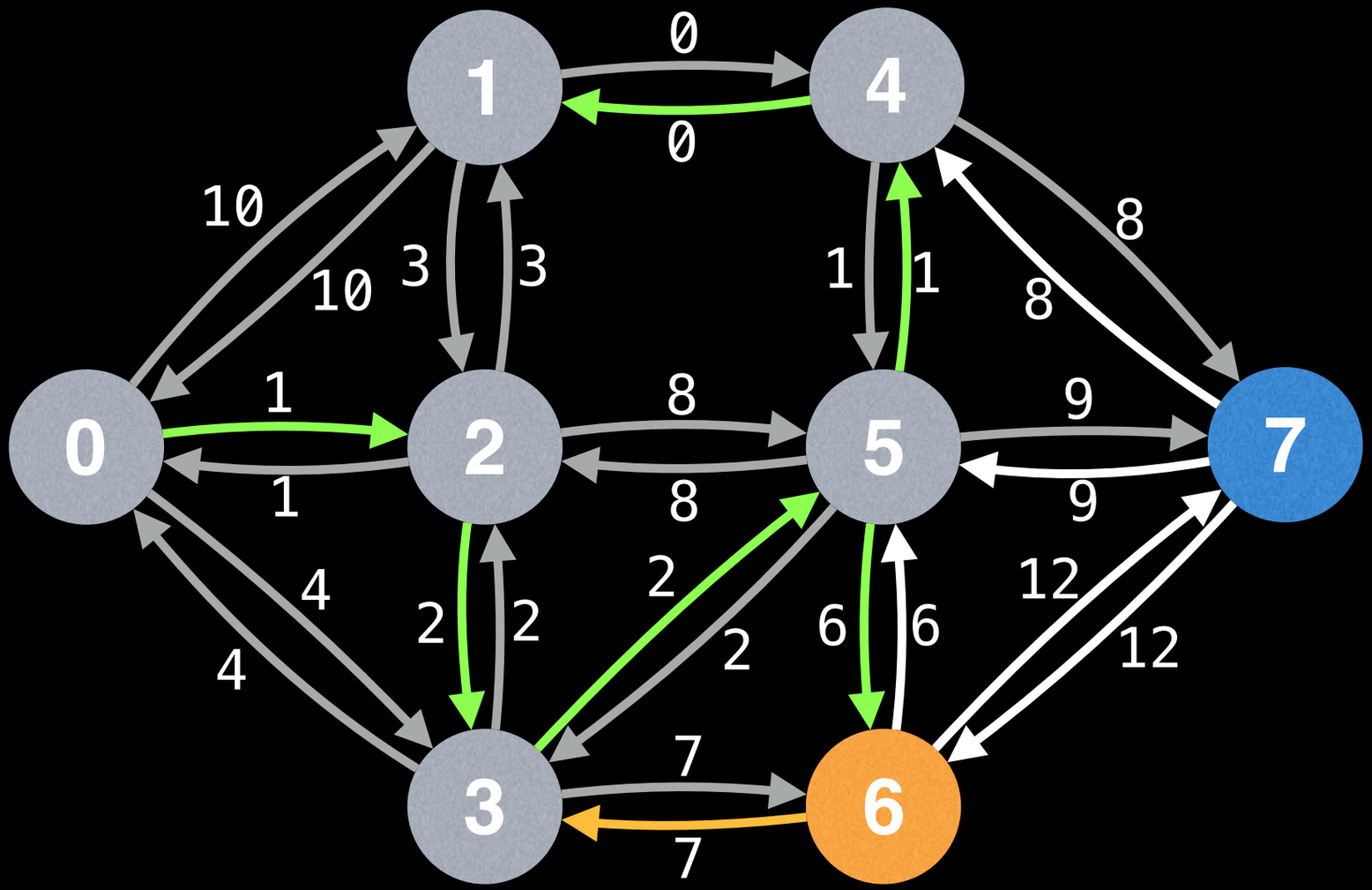
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
~~(5, 4, 1)~~
(5, 7, 9)
→ ~~(5, 6, 6)~~
~~(4, 1, 0)~~
(4, 7, 8)

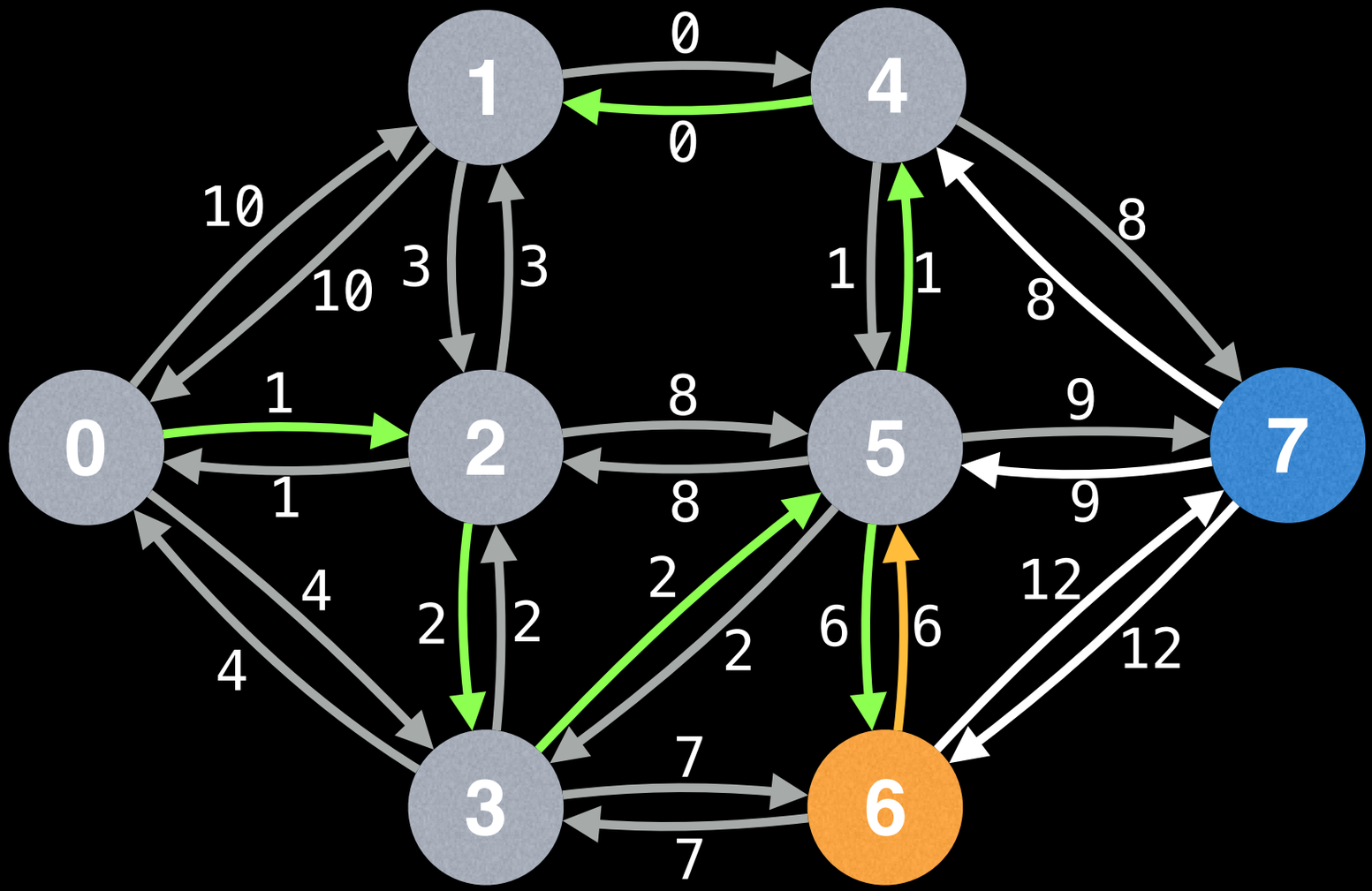
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
~~(5, 4, 1)~~
(5, 7, 9)
→ ~~(5, 6, 6)~~
~~(4, 1, 0)~~
(4, 7, 8)

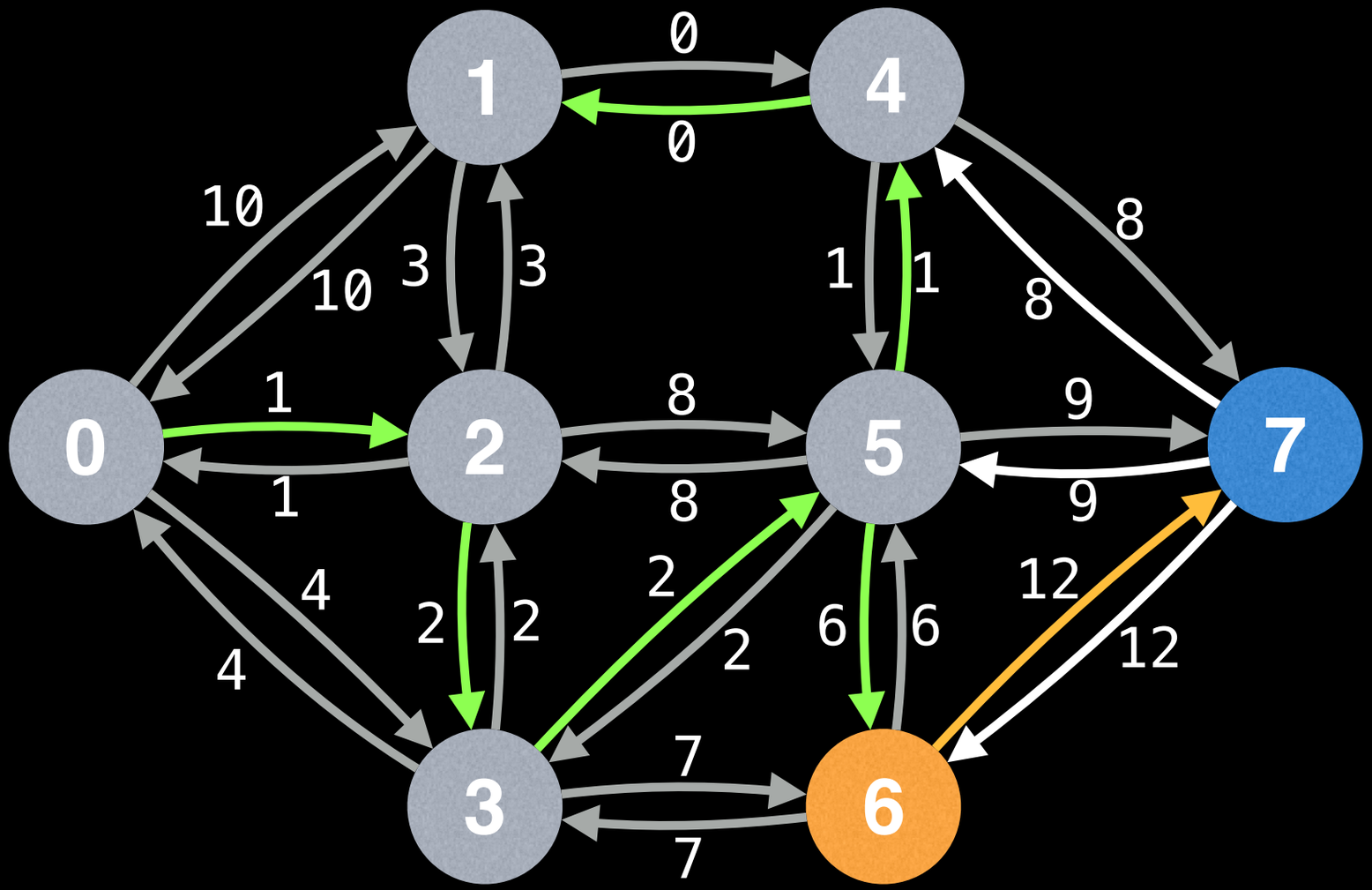
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
~~(5, 4, 1)~~
(5, 7, 9)
→ ~~(5, 6, 6)~~
~~(4, 1, 0)~~
(4, 7, 8)

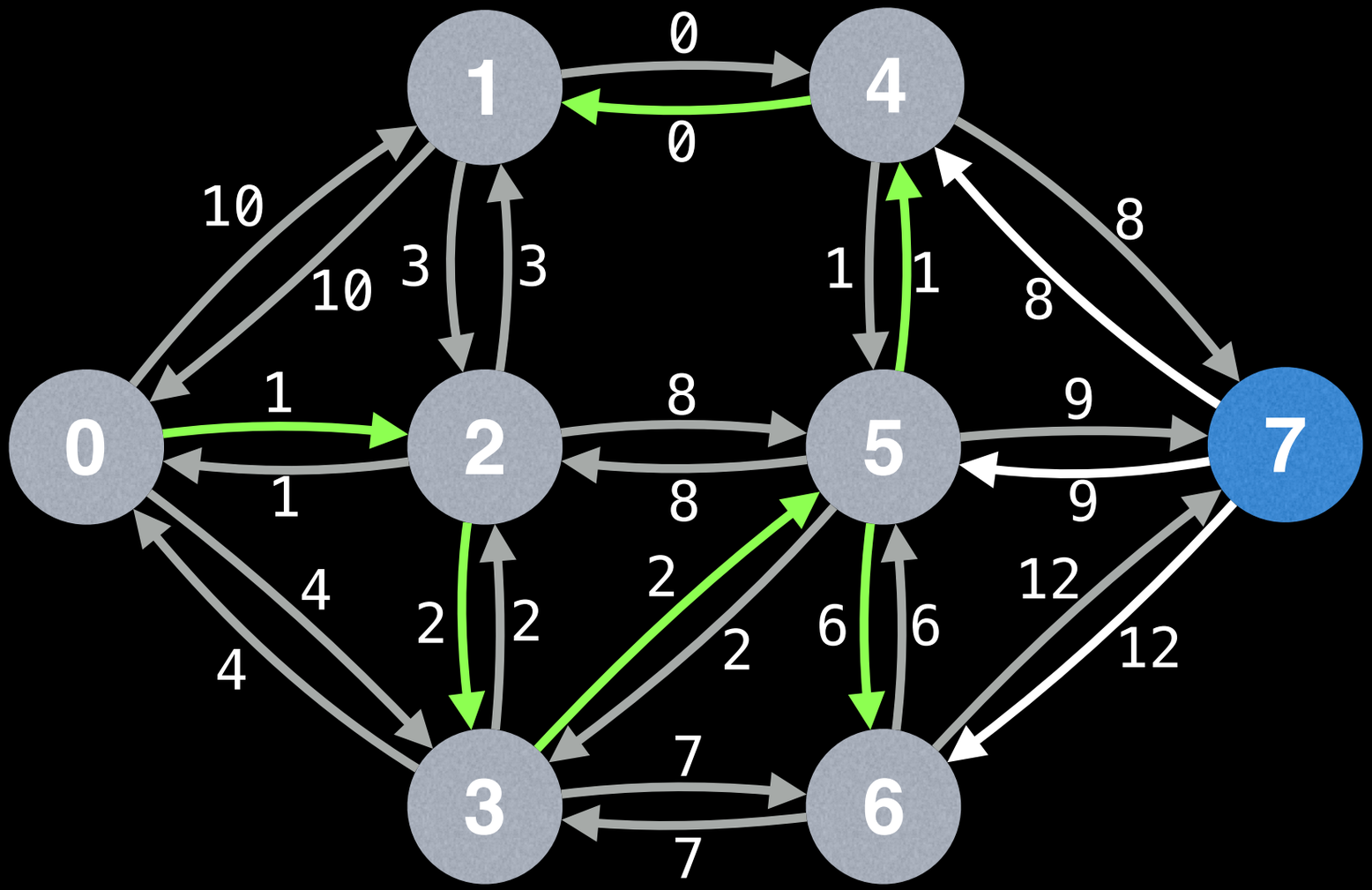
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
~~(5, 4, 1)~~
(5, 7, 9)
→ ~~(5, 6, 6)~~
~~(4, 1, 0)~~
(4, 7, 8)
(6, 7, 12)

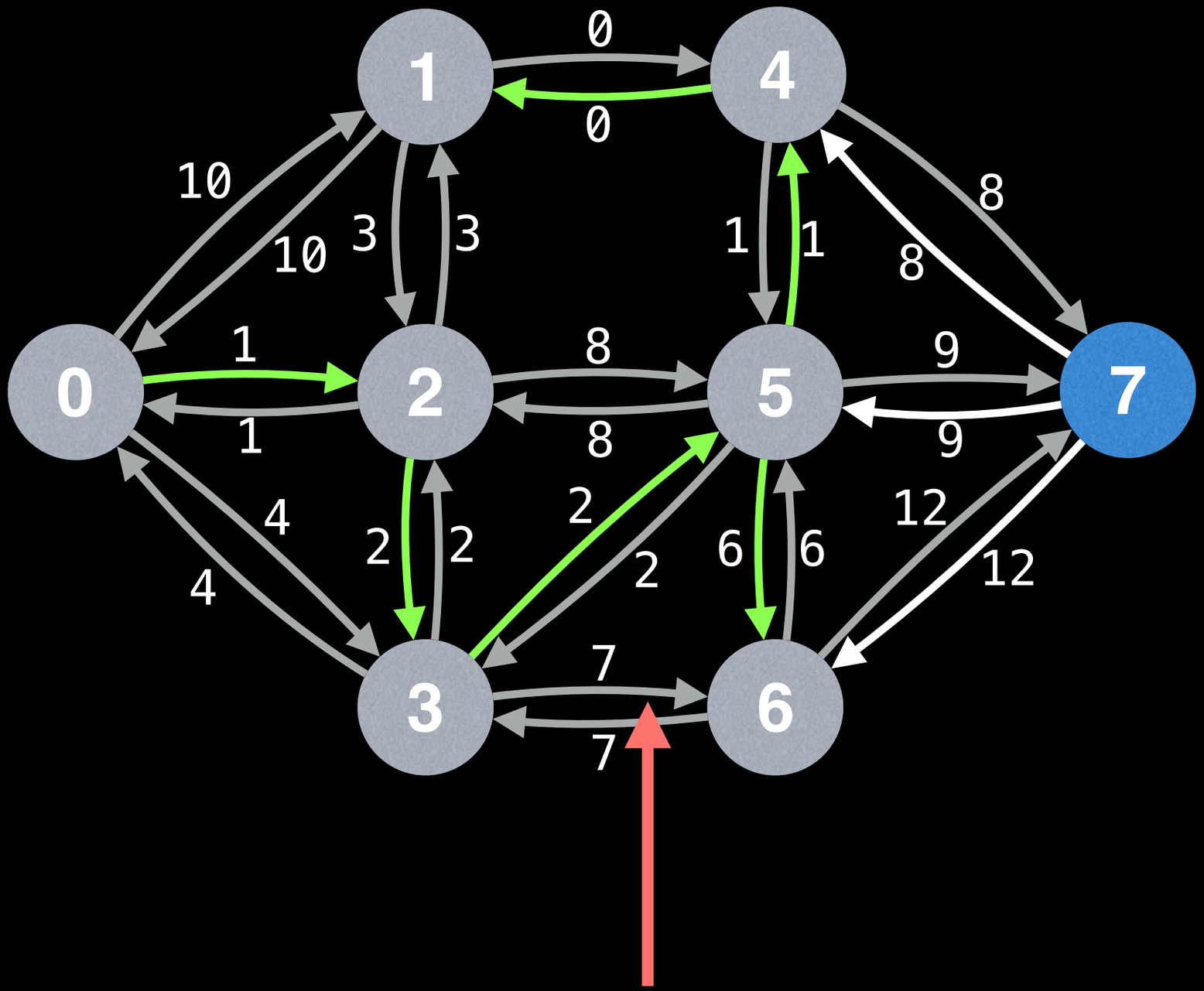
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
~~(5, 4, 1)~~
(5, 7, 9)
→ ~~(5, 6, 6)~~
~~(4, 1, 0)~~
(4, 7, 8)
(6, 7, 12)

Lazy Prim's

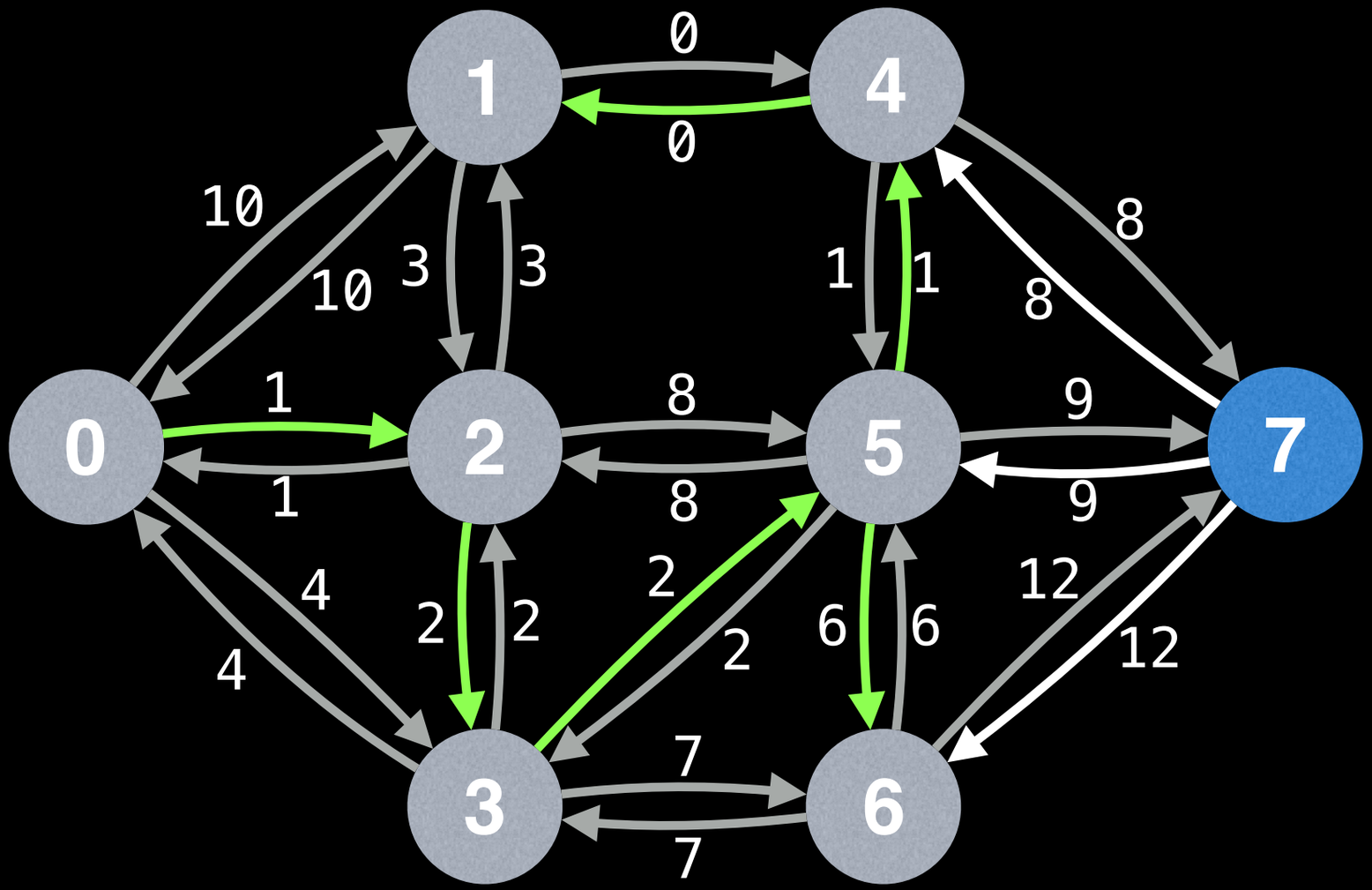


Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
(3, 6, 7)
~~(5, 4, 1)~~
(5, 7, 9)
~~(5, 6, 6)~~
~~(4, 1, 0)~~
(4, 7, 8)
(6, 7, 12)

The edge (3, 6, 7) is also stale, poll again.

Lazy Prim's

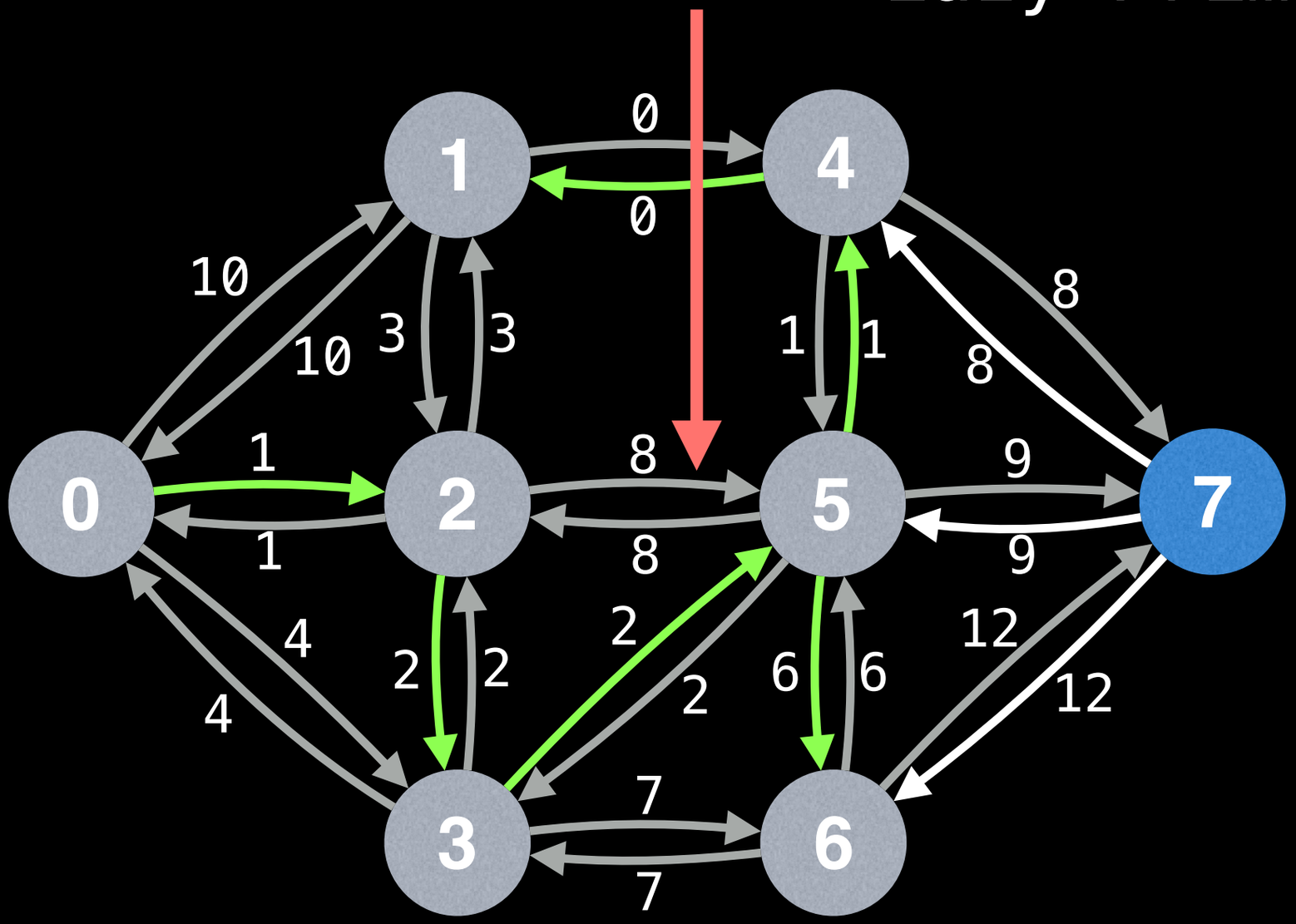


Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
→ (2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
~~(3, 6, 7)~~
~~(5, 4, 1)~~
(5, 7, 9)
~~(5, 6, 6)~~
~~(4, 1, 0)~~
(4, 7, 8)
(6, 7, 12)

There's a tie between (2, 5, 8) and (4, 7, 8) for which edge gets polled next. Since (2, 5, 8) was added first, let's assume it gets priority (it doesn't matter in practice).

Lazy Prim's

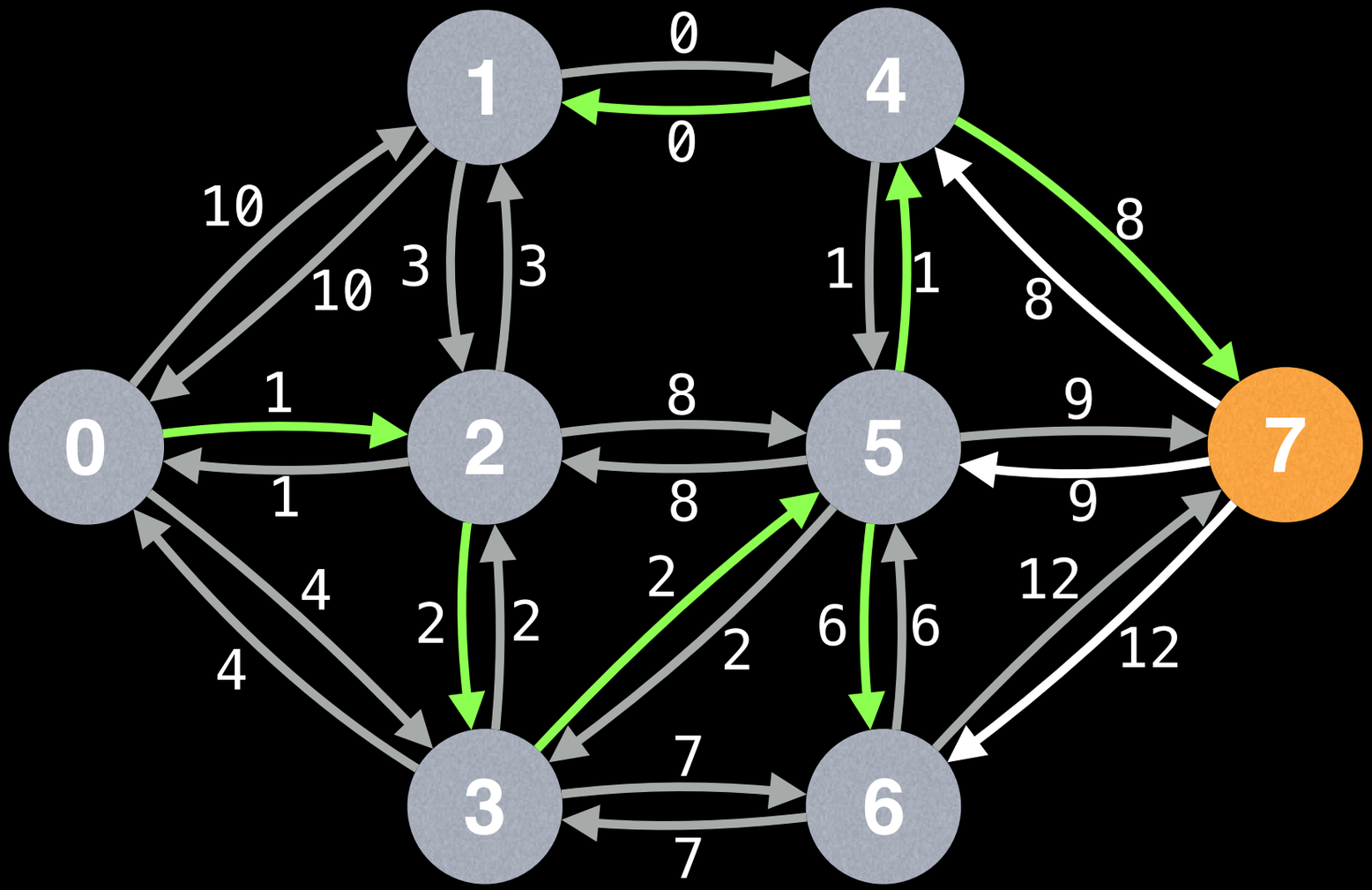


Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
(2, 5, 8)
~~(2, 3, 2)~~
~~(3, 5, 2)~~
~~(3, 6, 7)~~
~~(5, 4, 1)~~
(5, 7, 9)
~~(5, 6, 6)~~
~~(4, 1, 0)~~
(4, 7, 8)
(6, 7, 12)

The edge (2, 5, 8) is stale, poll again.

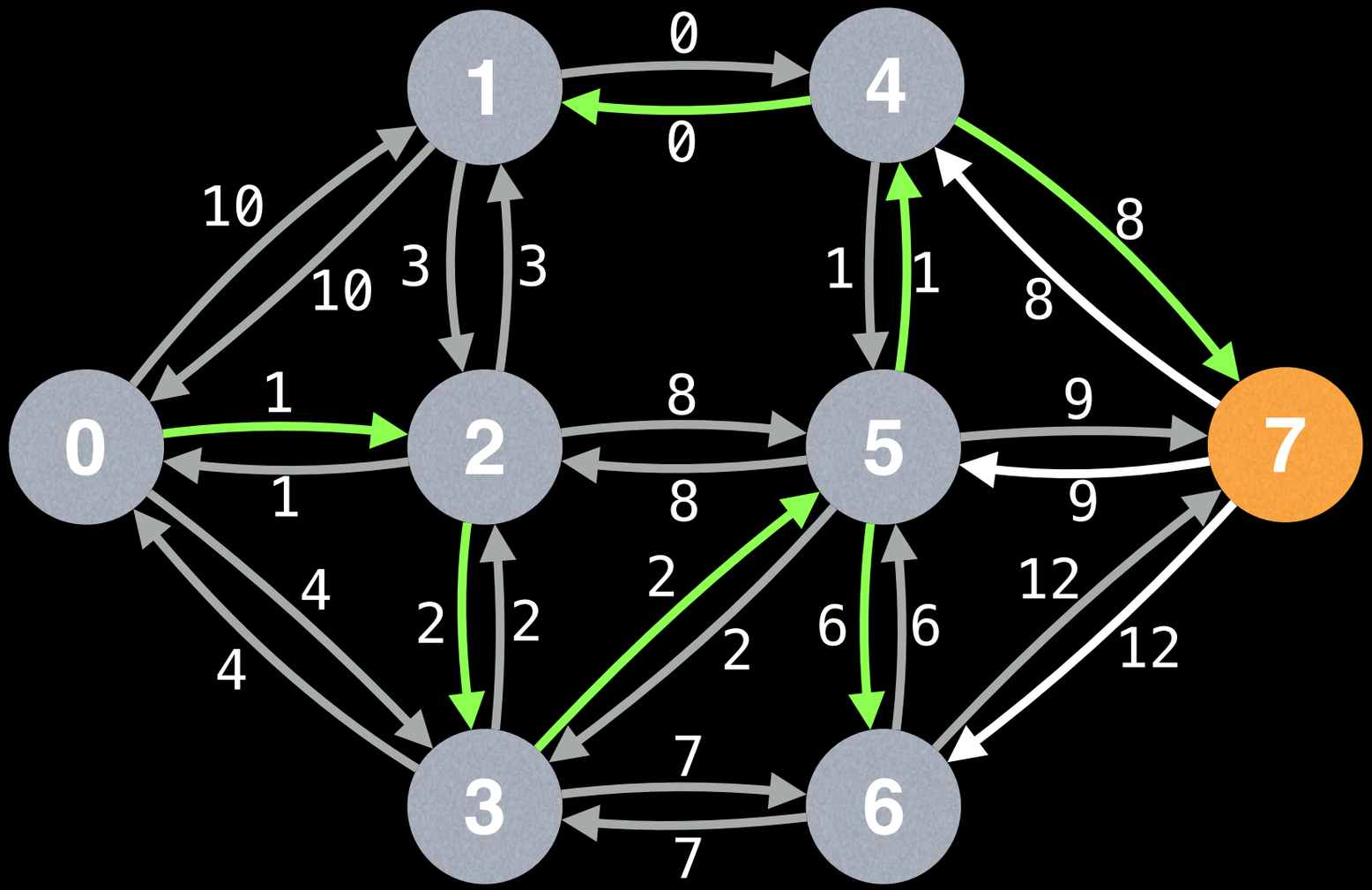
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
~~(2, 5, 8)~~
~~(2, 3, 2)~~
~~(3, 5, 2)~~
~~(3, 6, 7)~~
~~(5, 4, 1)~~
(5, 7, 9)
~~(5, 6, 6)~~
~~(4, 1, 0)~~
(4, 7, 8)
(6, 7, 12)

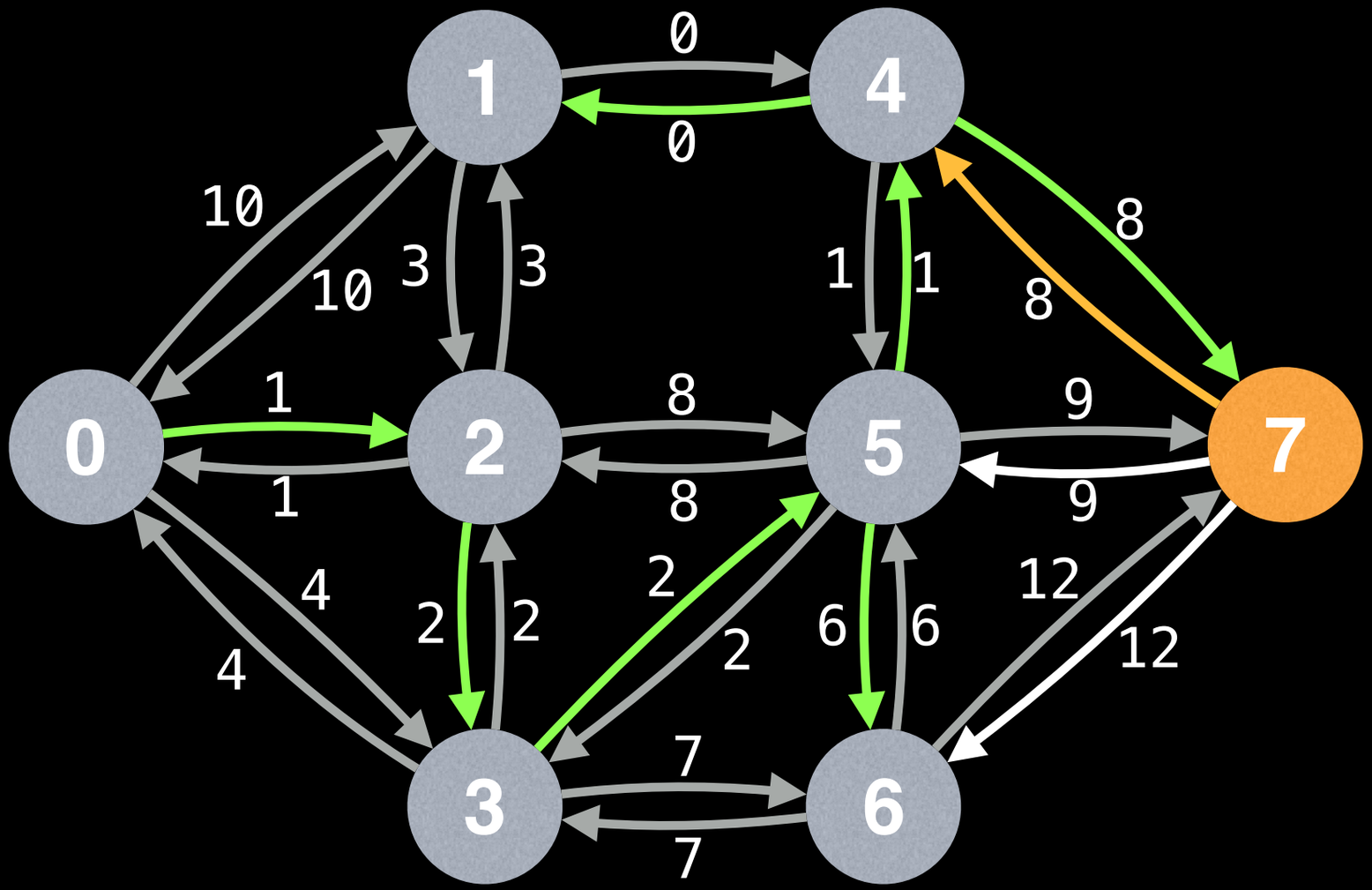
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
~~(2, 5, 8)~~
~~(2, 3, 2)~~
~~(3, 5, 2)~~
~~(3, 6, 7)~~
~~(5, 4, 1)~~
(5, 7, 9)
~~(5, 6, 6)~~
~~(4, 1, 0)~~
~~(4, 7, 8)~~
(6, 7, 12)

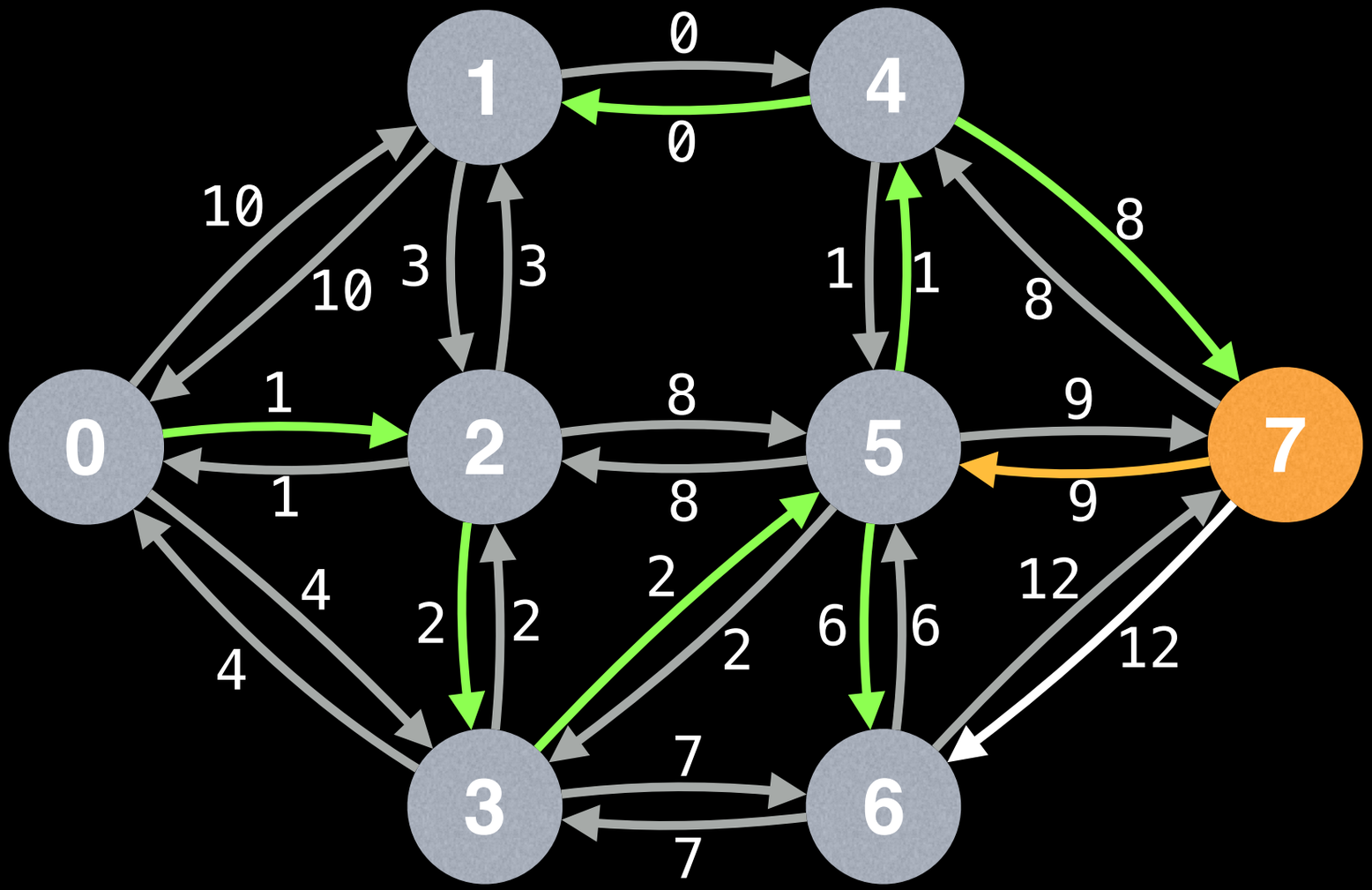
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
~~(2, 5, 8)~~
~~(2, 3, 2)~~
~~(3, 5, 2)~~
~~(3, 6, 7)~~
~~(5, 4, 1)~~
(5, 7, 9)
~~(5, 6, 6)~~
~~(4, 1, 0)~~
~~(4, 7, 8)~~
(6, 7, 12)

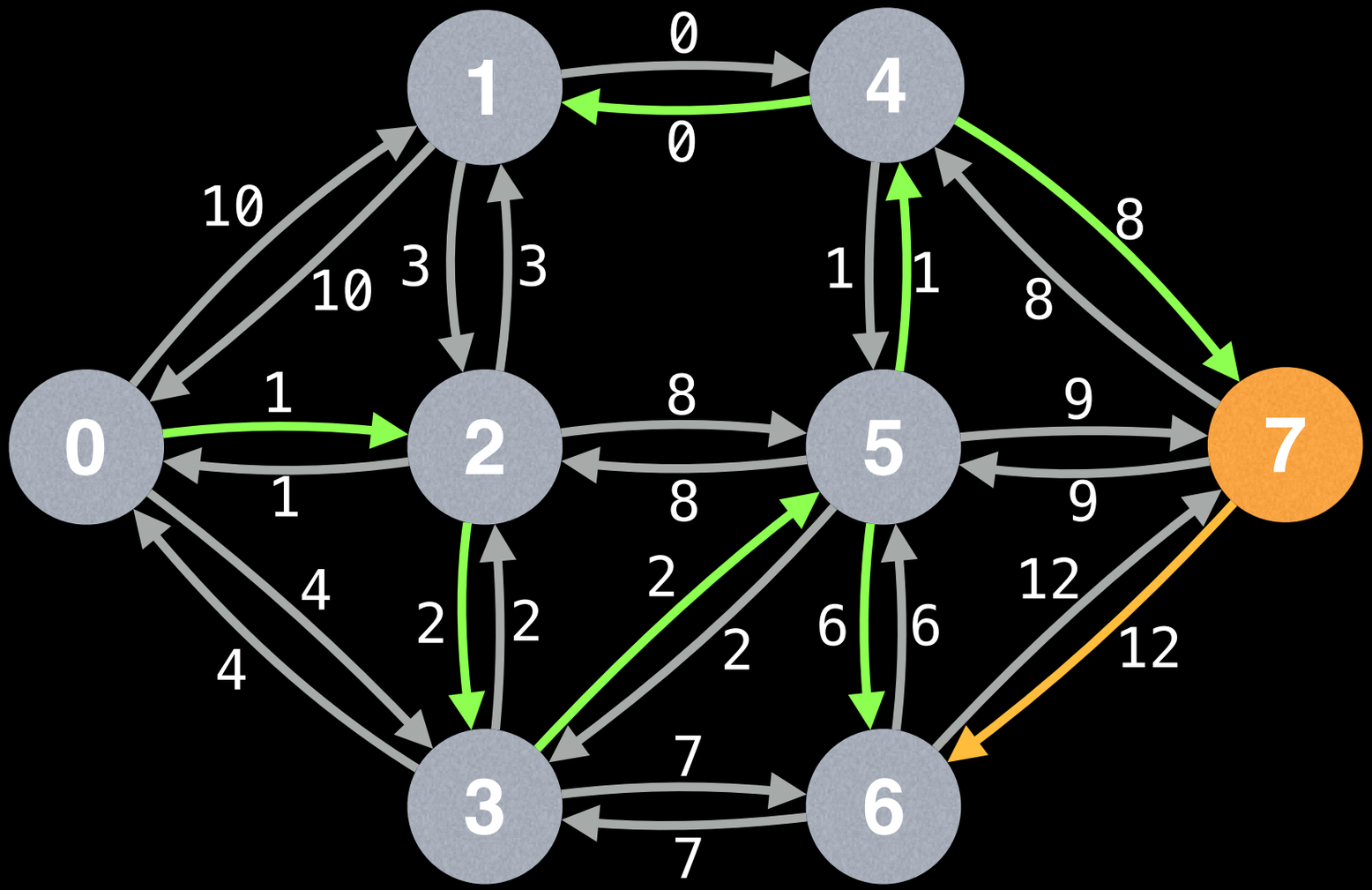
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
~~(2, 5, 8)~~
~~(2, 3, 2)~~
~~(3, 5, 2)~~
~~(3, 6, 7)~~
~~(5, 4, 1)~~
(5, 7, 9)
~~(5, 6, 6)~~
~~(4, 1, 0)~~
~~(4, 7, 8)~~
(6, 7, 12)

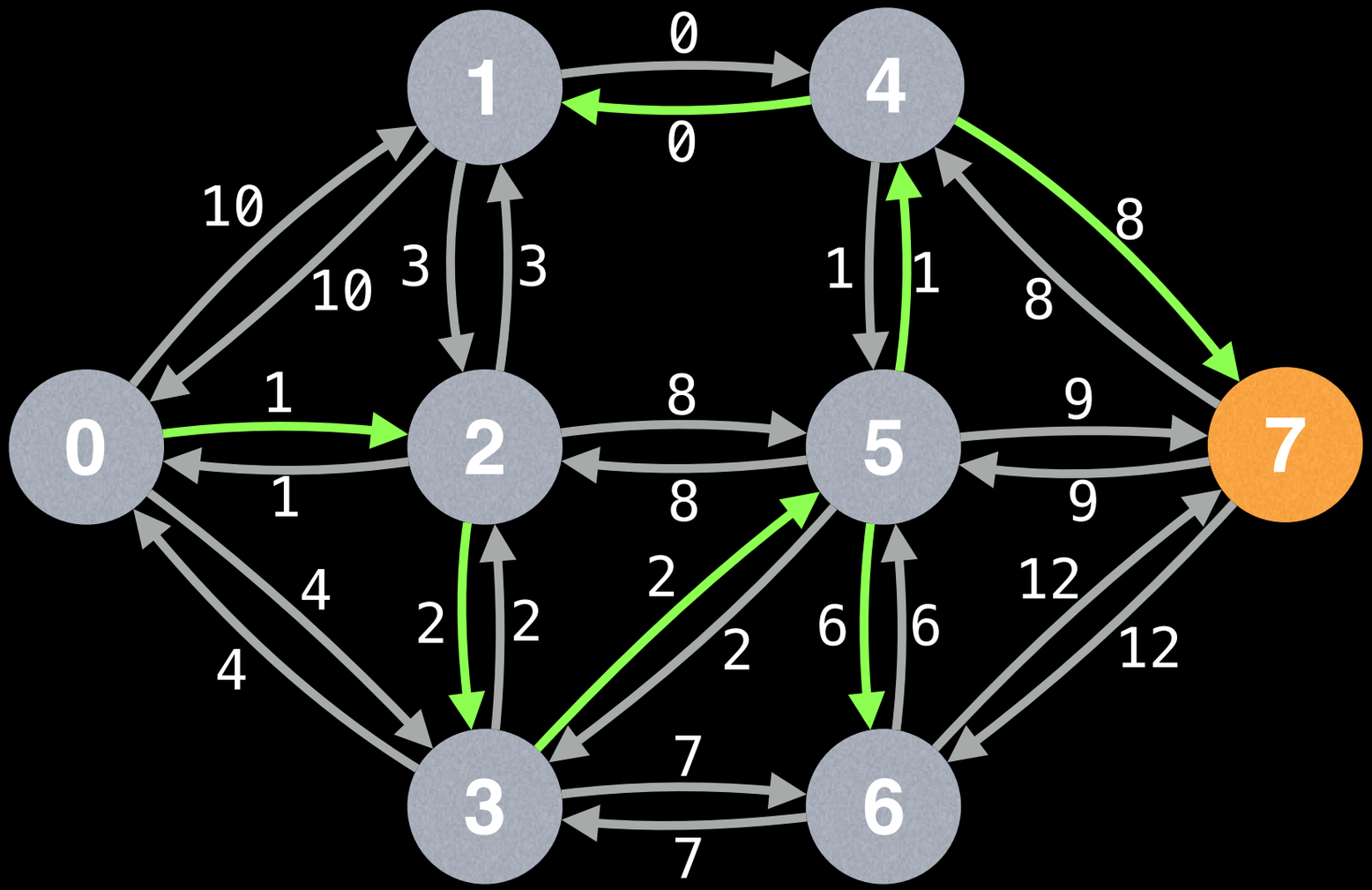
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
~~(2, 5, 8)~~
~~(2, 3, 2)~~
~~(3, 5, 2)~~
~~(3, 6, 7)~~
~~(5, 4, 1)~~
(5, 7, 9)
~~(5, 6, 6)~~
~~(4, 1, 0)~~
→ ~~(4, 7, 8)~~
(6, 7, 12)

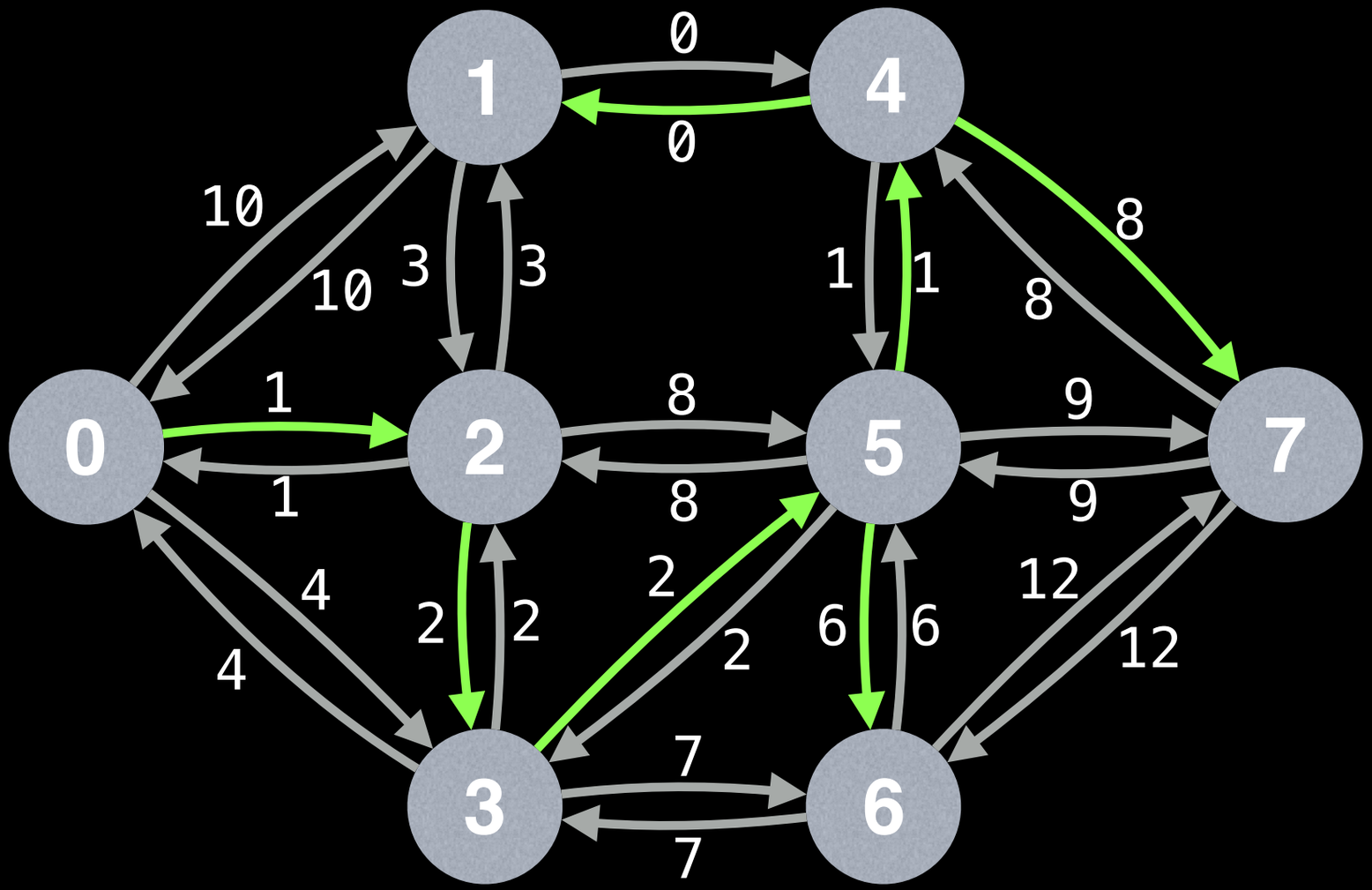
Lazy Prim's



Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
~~(2, 5, 8)~~
~~(2, 3, 2)~~
~~(3, 5, 2)~~
~~(3, 6, 7)~~
~~(5, 4, 1)~~
(5, 7, 9)
~~(5, 6, 6)~~
~~(4, 1, 0)~~
~~(4, 7, 8)~~
(6, 7, 12)

Lazy Prim's

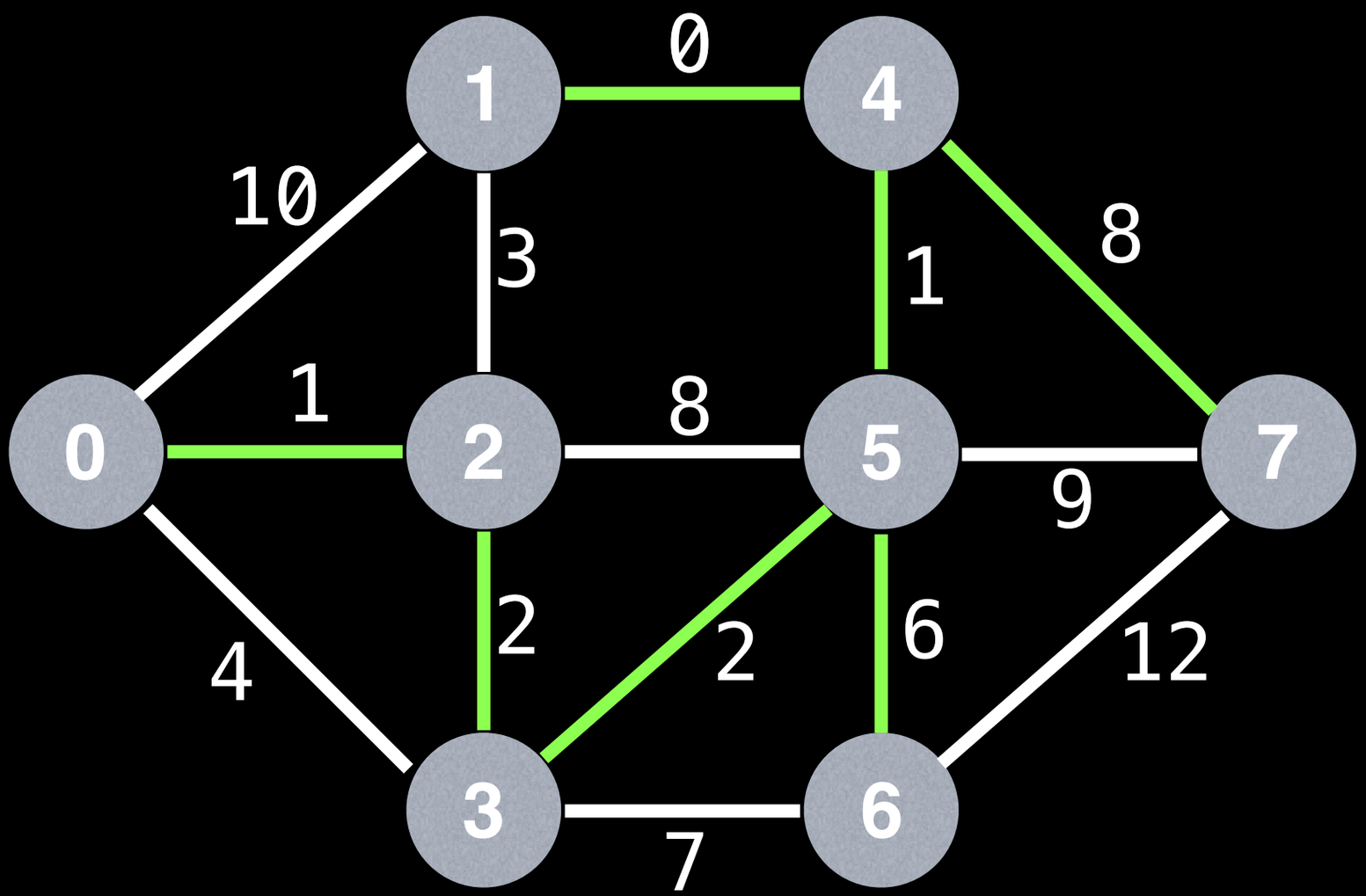


Edges in PQ
(start, end, cost)

(0, 1, 10)
~~(0, 2, 1)~~
~~(0, 3, 4)~~
~~(2, 1, 3)~~
~~(2, 5, 8)~~
~~(2, 3, 2)~~
~~(3, 5, 2)~~
~~(3, 6, 7)~~
~~(5, 4, 1)~~
(5, 7, 9)
~~(5, 6, 6)~~
~~(4, 1, 0)~~
→ ~~(4, 7, 8)~~
(6, 7, 12)

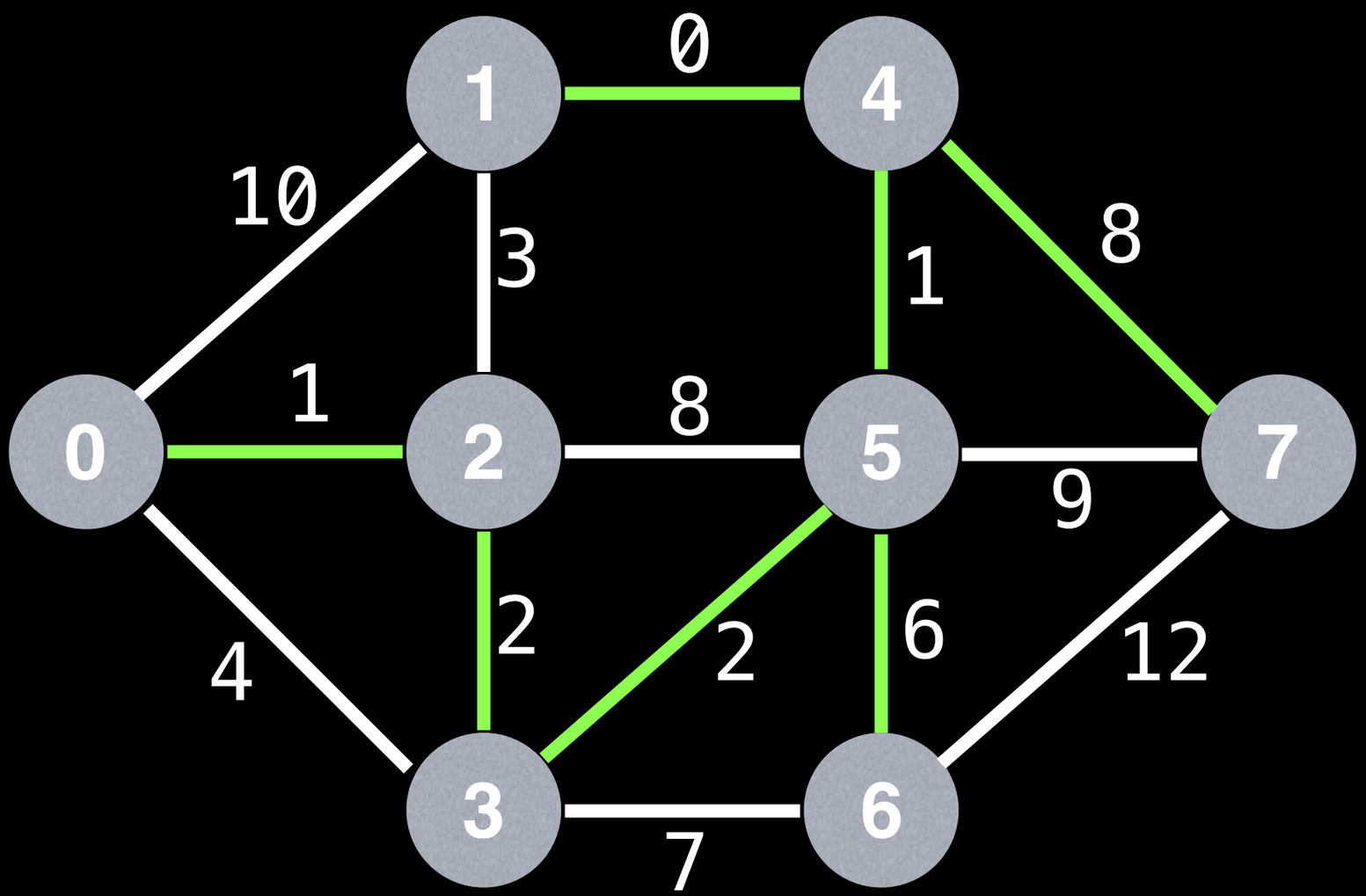
We can now stop Prim's since the MST is complete. We know the MST is complete because the number of edges in the tree is one less than the number of nodes in the graph (i.e. the definition of a tree).

Lazy Prim's

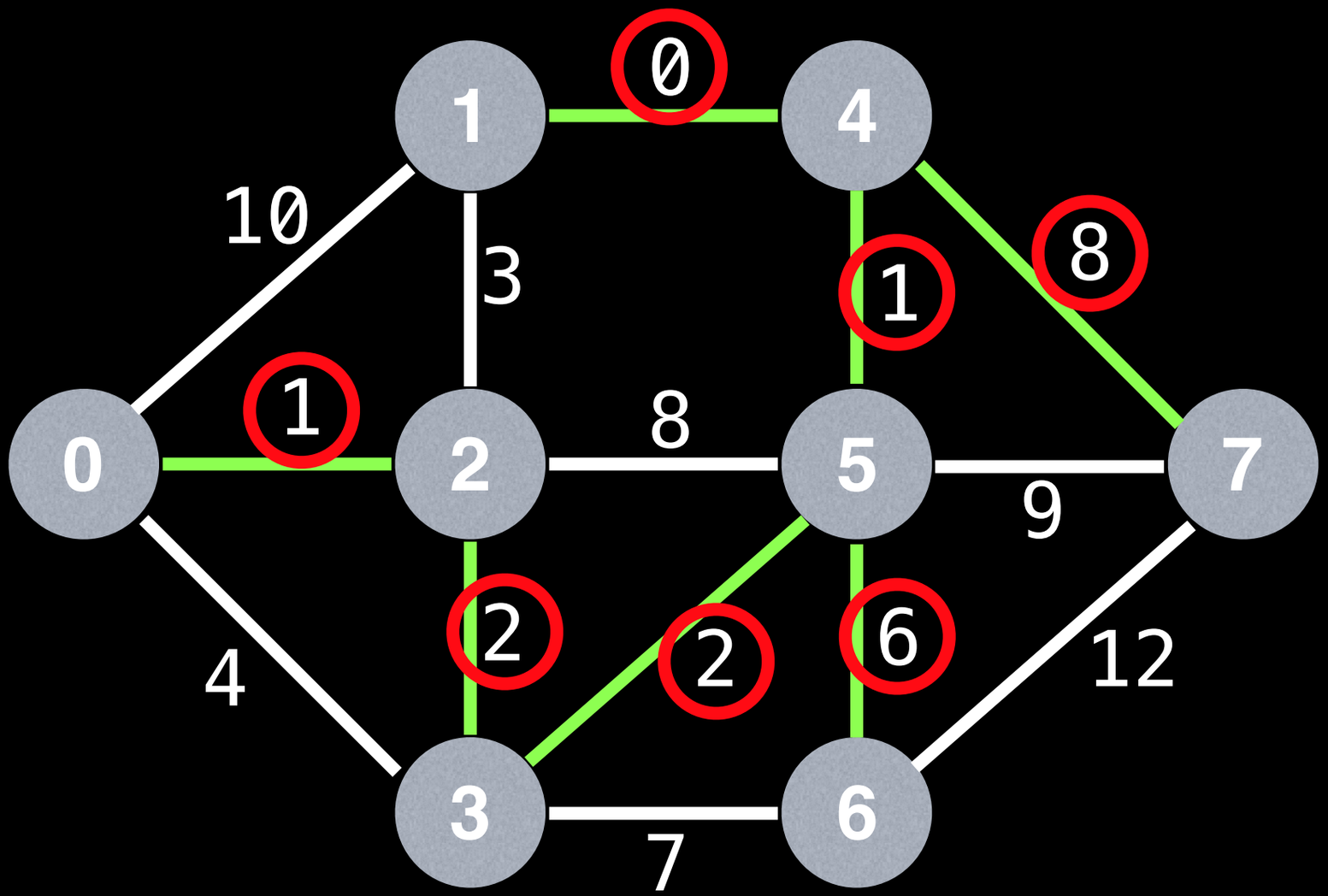


If we collapse the graph back into the undirected edge view it becomes clear which edges are included in the MST.

Lazy Prim's



Lazy Prim's



The MST cost is:

1 + 2 + 2 + 6 + 1 + 0 + 8 = 20

Lazy Prim's Pseudo Code

Let's define a few variables we'll need:

```
n = … # Number of nodes in the graph.

pq = … # PQ data structure; stores edge objects consisting of
       # {start node, end node, edge cost} tuples. The PQ sorts
       # edges based on min edge cost.

g = … # Graph representing an adjacency list of weighted edges.
      # Each undirected edge is represented as two directed
      # edges in g. For especially dense graphs, prefer using
      # an adjacency matrix instead of an adjacency list to
      # improve performance.

visited = [false, …, false] # visited[i] tracks whether node i
                            # has been visited; size n
```

Lazy Prim's Pseudo Code

Let's define a few variables we'll need:

```
n = … # Number of nodes in the graph.

pq = … # PQ data structure; stores edge objects consisting of
       # {start node, end node, edge cost} tuples. The PQ sorts
       # edges based on min edge cost.

g = … # Graph representing an adjacency list of weighted edges.
      # Each undirected edge is represented as two directed
      # edges in g. For especially dense graphs, prefer using
      # an adjacency matrix instead of an adjacency list to
      # improve performance.

visited = [false, …, false] # visited[i] tracks whether node i
                            # has been visited; size n
```

Lazy Prim's Pseudo Code

Let's define a few variables we'll need:

```
n = … # Number of nodes in the graph.

pq = … # PQ data structure; stores edge objects consisting of
       # {start node, end node, edge cost} tuples. The PQ sorts
       # edges based on min edge cost.

g = … # Graph representing an adjacency list of weighted edges.
      # Each undirected edge is represented as two directed
      # edges in g. For especially dense graphs, prefer using
      # an adjacency matrix instead of an adjacency list to
      # improve performance.

visited = [false, …, false] # visited[i] tracks whether node i
                            # has been visited; size n
```

Lazy Prim's Pseudo Code

Let's define a few variables we'll need:

```
n = … # Number of nodes in the graph.

pq = … # PQ data structure; stores edge objects consisting of
       # {start node, end node, edge cost} tuples. The PQ sorts
       # edges based on min edge cost.

g = … # Graph representing an adjacency list of weighted edges.
      # Each undirected edge is represented as two directed
      # edges in g. For especially dense graphs, prefer using
      # an adjacency matrix instead of an adjacency list to
      # improve performance.

visited = [false, …, false] # visited[i] tracks whether node i
                            # has been visited; size n
```

Lazy Prim's Pseudo Code

Let's define a few variables we'll need:

```
n = … # Number of nodes in the graph.

pq = … # PQ data structure; stores edge objects consisting of
       # {start node, end node, edge cost} tuples. The PQ sorts
       # edges based on min edge cost.

g = … # Graph representing an adjacency list of weighted edges.
      # Each undirected edge is represented as two directed
      # edges in g. For especially dense graphs, prefer using
      # an adjacency matrix instead of an adjacency list to
      # improve performance.

visited = [false, …, false] # visited[i] tracks whether node i
                            # has been visited; size n
```

```
# s - the index of the starting node (0 ≤ s < n)
function lazyPrims(s = 0):
  m = n - 1 # number of edges in MST
  edgeCount, mstCost = 0, 0
  mstEdges = [null, …, null] # size m
  addEdges(s)

  while (!pq.isEmpty() and edgeCount != m):
    edge = pq.dequeue()
    nodeIndex = edge.to

    if visited[nodeIndex]:
      continue

    mstEdges[edgeCount++] = edge
    mstCost += edge.cost

    addEdges(nodeIndex)

if edgeCount != m:
  return (null, null) # No MST exists!

return (mstCost, mstEdges)
```

```
# s - the index of the starting node (0 ≤ s < n)
function lazyPrims(s = 0):
  m = n - 1 # number of edges in MST
  edgeCount, mstCost = 0, 0
  mstEdges = [null, …, null] # size m
  addEdges(s)

  while (!pq.isEmpty() and edgeCount != m):
    edge = pq.dequeue()
    nodeIndex = edge.to

    if visited[nodeIndex]:
      continue

    mstEdges[edgeCount++] = edge
    mstCost += edge.cost

    addEdges(nodeIndex)

if edgeCount != m:
  return (null, null) # No MST exists!

return (mstCost, mstEdges)
```

```
# s - the index of the starting node (0 ≤ s < n)
function lazyPrims(s = 0):
  m = n - 1 # number of edges in MST
  edgeCount, mstCost = 0, 0
  mstEdges = [null, …, null] # size m
  addEdges(s)

  while (!pq.isEmpty() and edgeCount != m):
    edge = pq.dequeue()
    nodeIndex = edge.to

    if visited[nodeIndex]:
      continue

    mstEdges[edgeCount++] = edge
    mstCost += edge.cost

    addEdges(nodeIndex)

if edgeCount != m:
  return (null, null) # No MST exists!

return (mstCost, mstEdges)
```

```
# s - the index of the starting node (0 ≤ s < n)
function lazyPrims(s = 0):
  m = n - 1 # number of edges in MST
  edgeCount, mstCost = 0, 0
  mstEdges = [null, …, null] # size m
  addEdges(s)

  while (!pq.isEmpty() and edgeCount != m):
    edge = pq.dequeue()
    nodeIndex = edge.to

    if visited[nodeIndex]:
      continue

    mstEdges[edgeCount++] = edge
    mstCost += edge.cost

    addEdges(nodeIndex)

if edgeCount != m:
  return (null, null) # No MST exists!

return (mstCost, mstEdges)
```

Helper method to iterate over the edges of a node and add edges to the PQ:

```
function addEdges(nodeIndex):
  # Mark the current node as visited.
  visited[nodeIndex] = true

  # Iterate over all edges going outwards from the current node.
  # Add edges to the PQ which point to unvisited nodes.
  edges = g[nodeIndex]
  for (edge : edges):
    if !visited[edge.to]:
      pq.enqueue(edge)
```

Helper method to iterate over the edges of a node and add edges to the PQ:

```
function addEdges(nodeIndex):
  # Mark the current node as visited.
  visited[nodeIndex] = true

  # Iterate over all edges going outwards from the current node.
  # Add edges to the PQ which point to unvisited nodes.
  edges = g[nodeIndex]
  for (edge : edges):
    if !visited[edge.to]:
      pq.enqueue(edge)
```

Helper method to iterate over the edges of a node and add edges to the PQ:

```
function addEdges(nodeIndex):
  # Mark the current node as visited.
  visited[nodeIndex] = true
  # Iterate over all edges going outwards from the current node.
  # Add edges to the PQ which point to unvisited nodes.
  edges = g[nodeIndex]
  for (edge : edges):
    if !visited[edge.to]:
      pq.enqueue(edge)
```

Helper method to iterate over the edges of a node and add edges to the PQ:

```
function addEdges(nodeIndex):
  # Mark the current node as visited.
  visited[nodeIndex] = true

  # Iterate over all edges going outwards from the current node.
  # Add edges to the PQ which point to unvisited nodes.
  edges = g[nodeIndex]
  for (edge : edges):
    if !visited[edge.to]:
      pq.enqueue(edge)
```

```
# s - the index of the starting node (0 ≤ s < n)
function lazyPrims(s = 0):
  m = n - 1 # number of edges in MST
  edgeCount, mstCost = 0, 0
  mstEdges = [null, …, null] # size m
  addEdges(s)

  while (!pq.isEmpty() and edgeCount != m):
    edge = pq.dequeue()
    nodeIndex = edge.to

    if visited[nodeIndex]:
      continue

    mstEdges[edgeCount++] = edge
    mstCost += edge.cost

    addEdges(nodeIndex)

if edgeCount != m:
  return (null, null) # No MST exists!

return (mstCost, mstEdges)
```

```
# s - the index of the starting node (0 ≤ s < n)
function lazyPrims(s = 0):
  m = n - 1 # number of edges in MST
  edgeCount, mstCost = 0, 0
  mstEdges = [null, …, null] # size m
  addEdges(s)

  while (!pq.isEmpty() and edgeCount != m):
    edge = pq.dequeue()
    nodeIndex = edge.to

    if visited[nodeIndex]:
      continue

    mstEdges[edgeCount++] = edge
    mstCost += edge.cost

    addEdges(nodeIndex)

if edgeCount != m:
  return (null, null) # No MST exists!

return (mstCost, mstEdges)
```

```
# s - the index of the starting node (0 ≤ s < n)
function lazyPrims(s = 0):
  m = n - 1 # number of edges in MST
  edgeCount, mstCost = 0, 0
  mstEdges = [null, …, null] # size m
  addEdges(s)

  while (!pq.isEmpty() and edgeCount != m):
    edge = pq.dequeue()
    nodeIndex = edge.to

    if visited[nodeIndex]:
      continue

    mstEdges[edgeCount++] = edge
    mstCost += edge.cost

    addEdges(nodeIndex)

if edgeCount != m:
  return (null, null) # No MST exists!

return (mstCost, mstEdges)
```

```
# s - the index of the starting node (0 ≤ s < n)
function lazyPrims(s = 0):
  m = n - 1 # number of edges in MST
  edgeCount, mstCost = 0, 0
  mstEdges = [null, …, null] # size m
  addEdges(s)

  while (!pq.isEmpty() and edgeCount != m):
    edge = pq.dequeue()
    nodeIndex = edge.to

    if visited[nodeIndex]:
      continue

    mstEdges[edgeCount++] = edge
    mstCost += edge.cost

    addEdges(nodeIndex)

if edgeCount != m:
  return (null, null) # No MST exists!

return (mstCost, mstEdges)
```

Skip edge that points to a visited node.

```
# s - the index of the starting node (0 ≤ s < n)
function lazyPrims(s = 0):
  m = n - 1 # number of edges in MST
  edgeCount, mstCost = 0, 0
  mstEdges = [null, …, null] # size m
  addEdges(s)

  while (!pq.isEmpty() and edgeCount != m):
    edge = pq.dequeue()
    nodeIndex = edge.to

    if visited[nodeIndex]:
      continue

    mstEdges[edgeCount++] = edge
    mstCost += edge.cost

    addEdges(nodeIndex)

if edgeCount != m:
  return (null, null) # No MST exists!

return (mstCost, mstEdges)
```

```
# s - the index of the starting node (0 ≤ s < n)
function lazyPrims(s = 0):
  m = n - 1 # number of edges in MST
  edgeCount, mstCost = 0, 0
  mstEdges = [null, …, null] # size m
  addEdges(s)

  while (!pq.isEmpty() and edgeCount != m):
    edge = pq.dequeue()
    nodeIndex = edge.to

    if visited[nodeIndex]:
      continue

    mstEdges[edgeCount++] = edge
    mstCost += edge.cost

    addEdges(nodeIndex)

if edgeCount != m:
  return (null, null) # No MST exists!

return (mstCost, mstEdges)
```

```
# s - the index of the starting node (0 ≤ s < n)
function lazyPrims(s = 0):
  m = n - 1 # number of edges in MST
  edgeCount, mstCost = 0, 0
  mstEdges = [null, …, null] # size m
  addEdges(s)

  while (!pq.isEmpty() and edgeCount != m):
    edge = pq.dequeue()
    nodeIndex = edge.to

    if visited[nodeIndex]:
      continue

    mstEdges[edgeCount++] = edge
    mstCost += edge.cost

    addEdges(nodeIndex)

if edgeCount != m:
  return (null, null) # No MST exists!

return (mstCost, mstEdges)
```

Next Video: Eager Prim’s