Design and Analysis of Algorithms

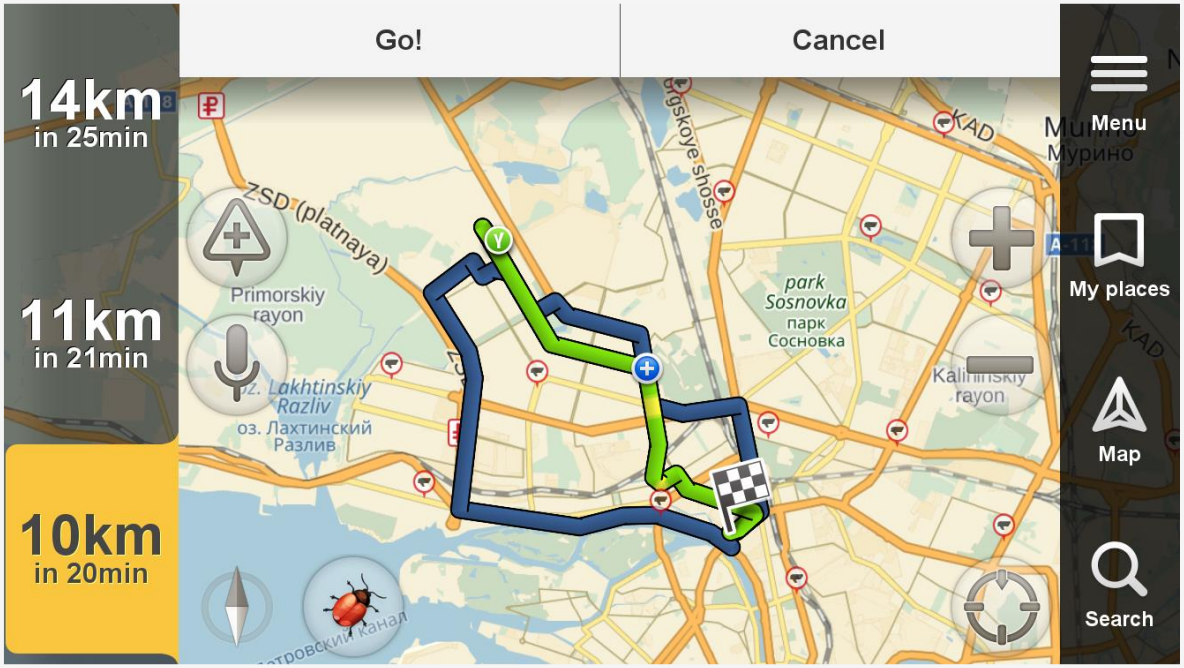
06-02
Paths in Graph

Fastest Route: Dijkstra's Algorithm

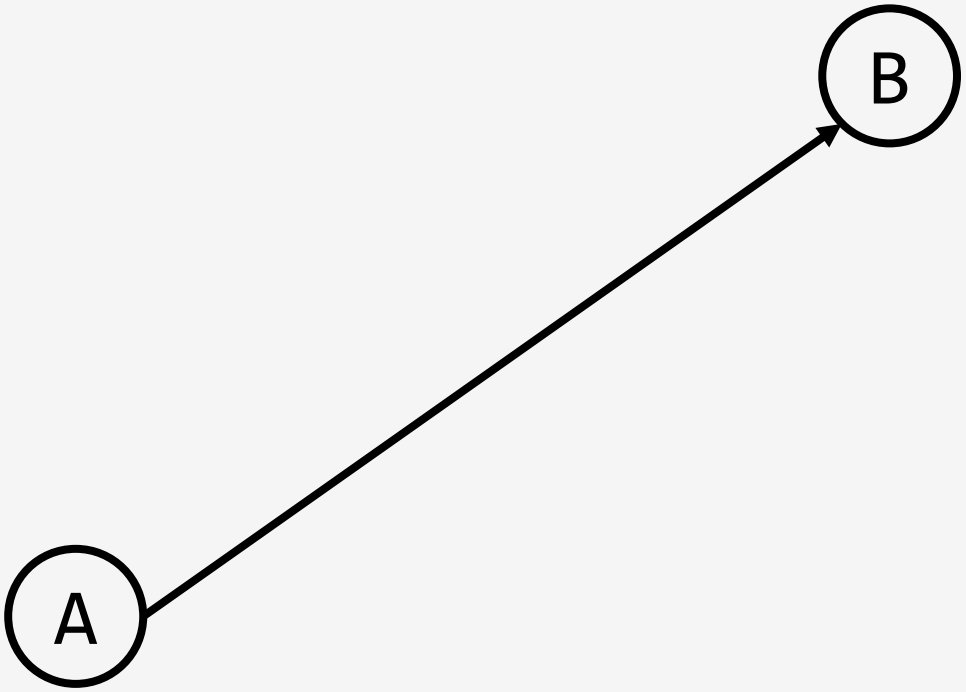
Imran Ihsan
Assistant Professor, Department of Computer Science
Air University, Islamabad, Pakistan
www.imranihsan.com

Fastest Route

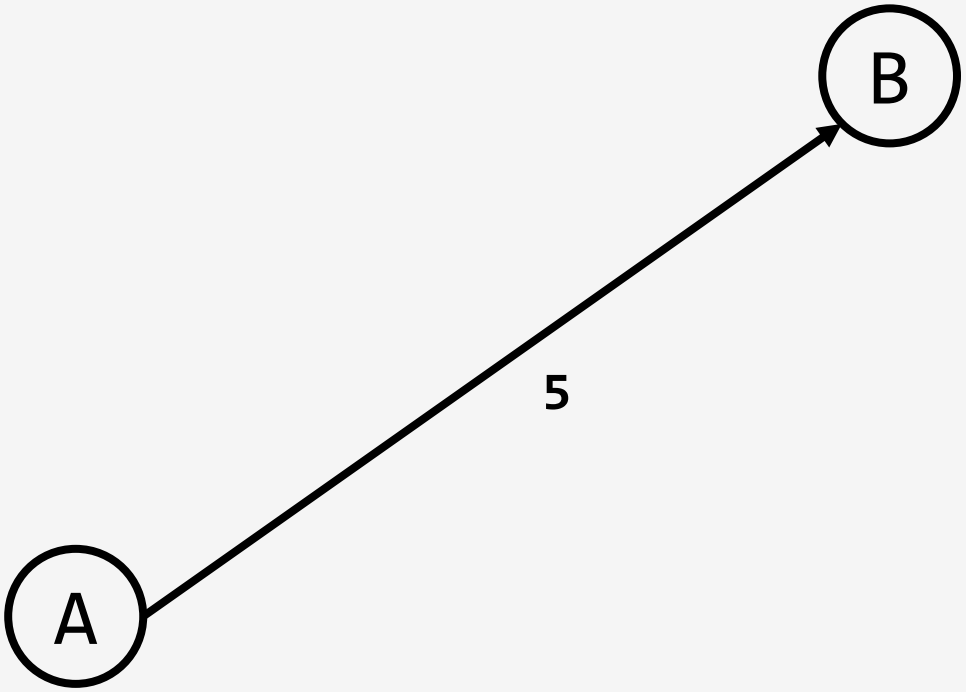
What is the fastest route to get home from work?



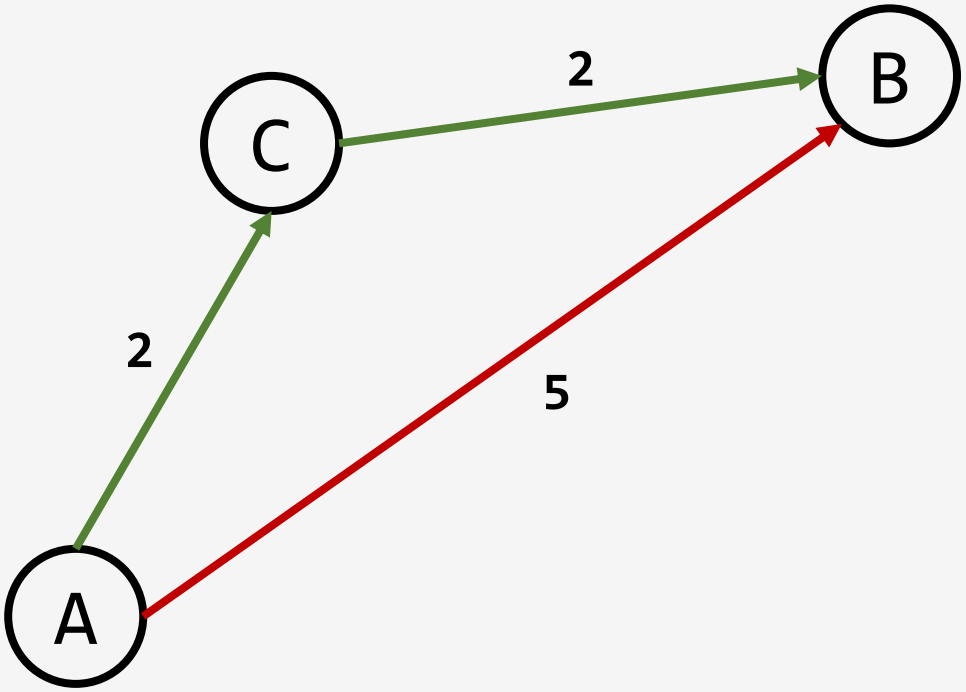
Fastest Route



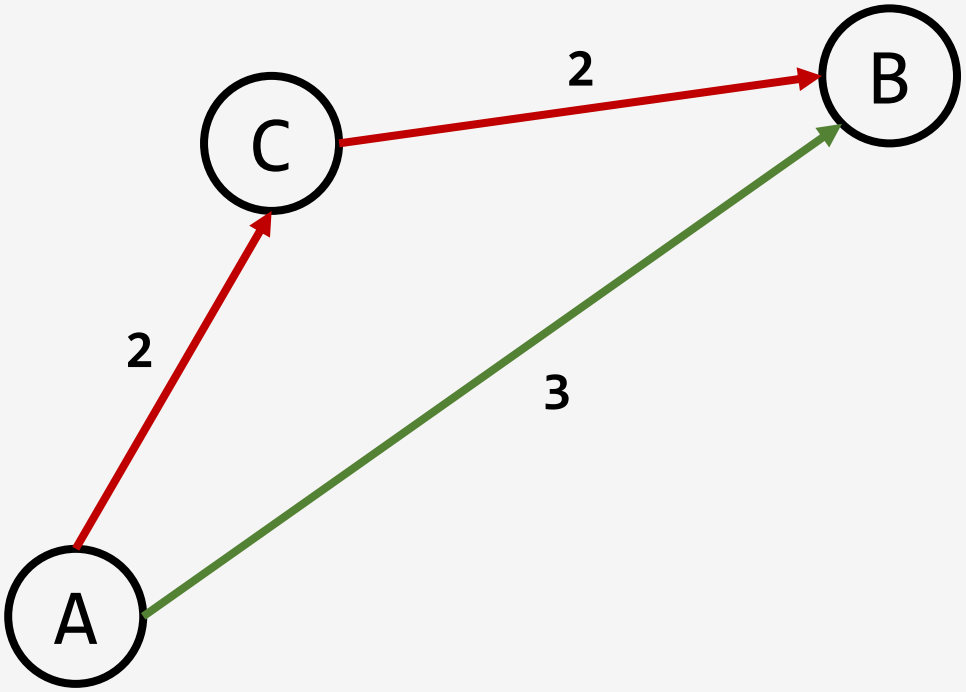
Fastest Route



Fastest Route

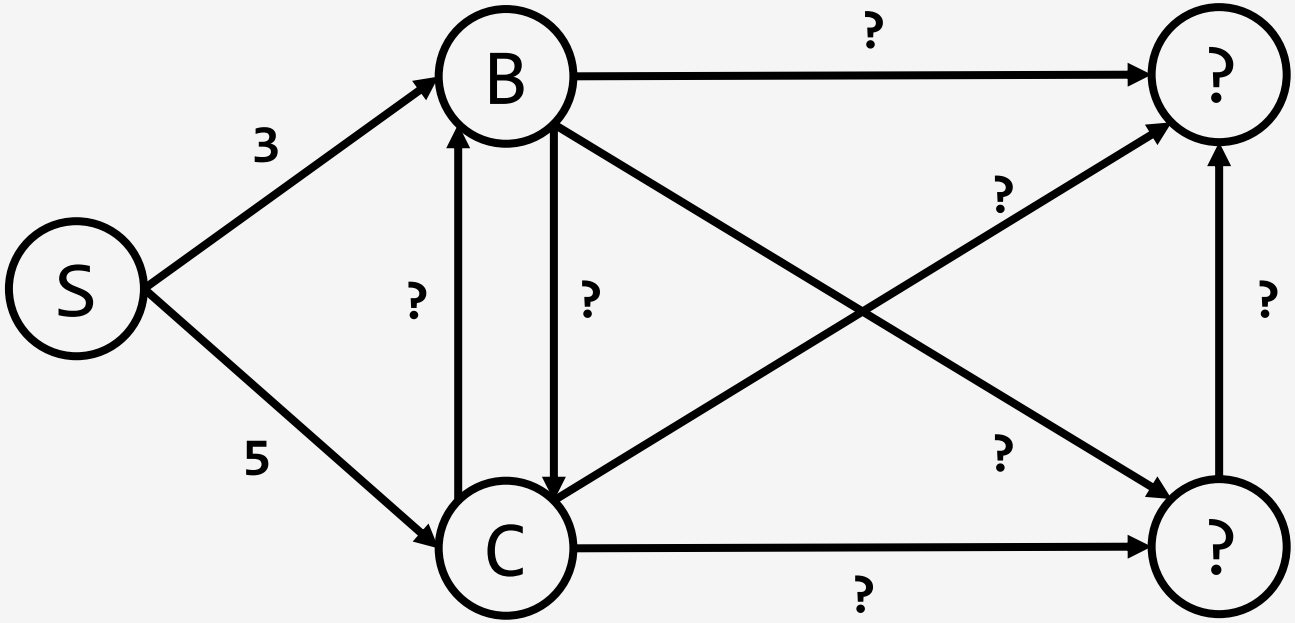


Fastest Route



Intuition

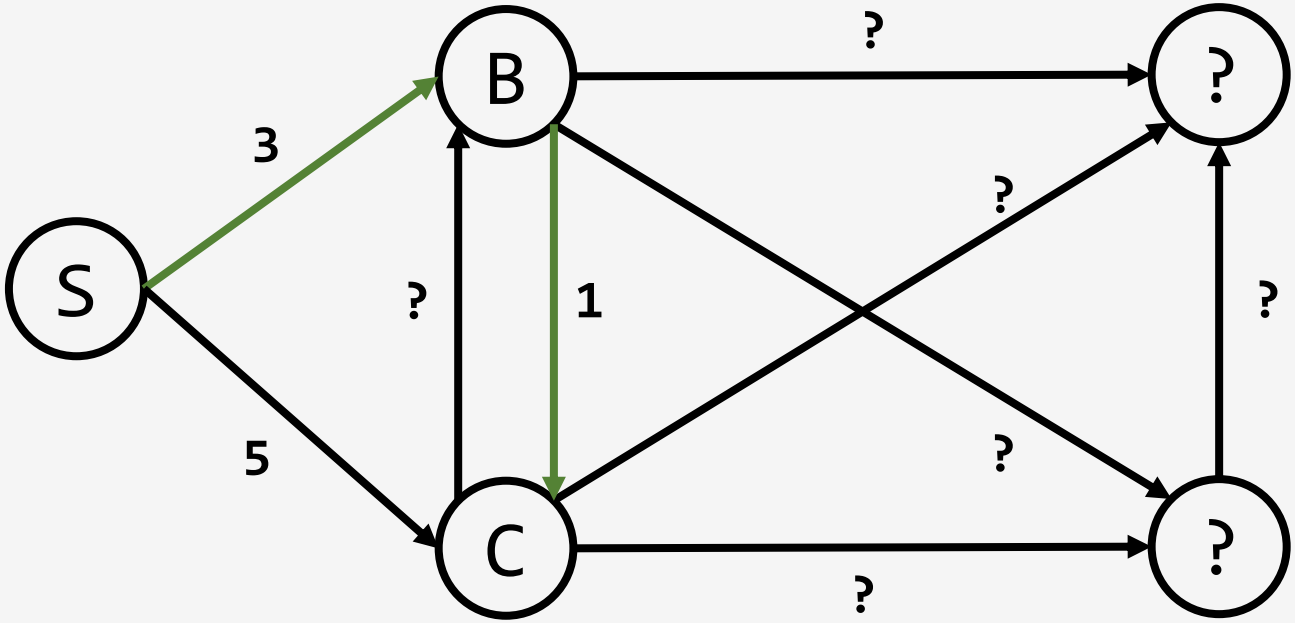
Assume that we stay at S and observe two outgoing edges:



Can we be sure that the distance from S to C is 5?

Intuition

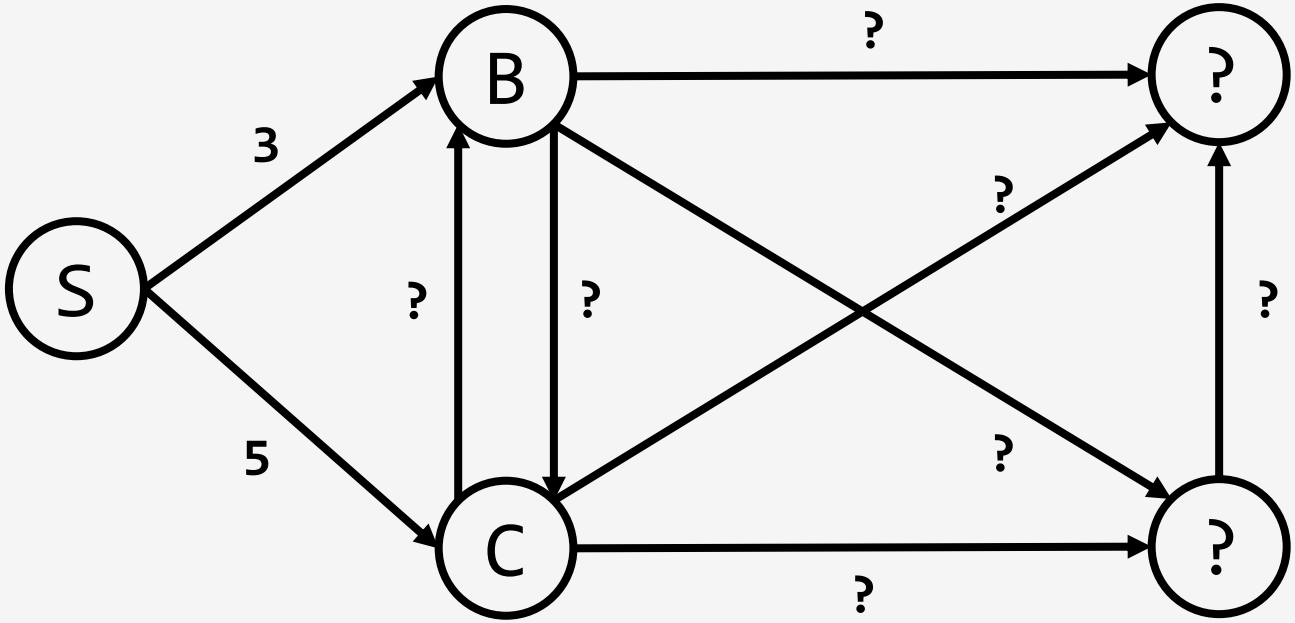
Can we be sure that the distance from S to C is 5?



No, because the weight of the edge (B, C) might be equal to, say, 1.

Intuition

Can we be sure that the distance from S to B is 3?



Yes, because there are no negative weight edges.

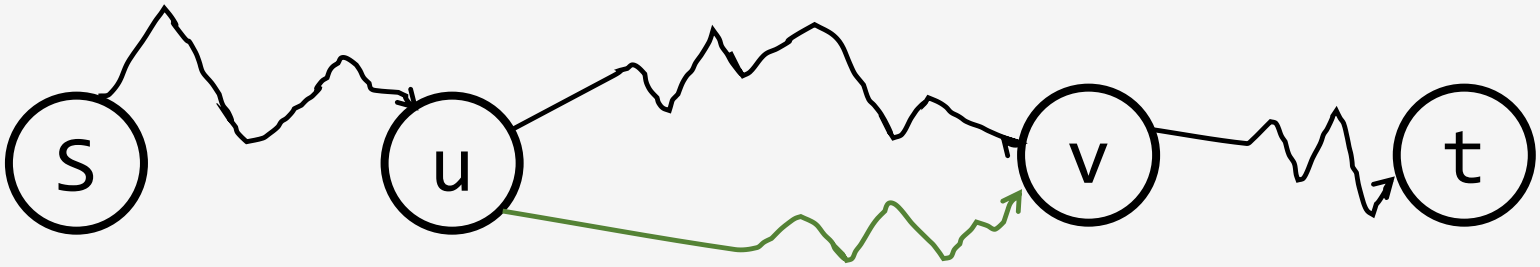
Naïve Algorithm

Optimal substructure

Any sub-path of an optimal path is also optimal.

Proof

Consider an optimal path from S to t and two vertices u and v on this path. If there were a shorter path from u to v, we would get a shorter path from S to t.



Corollary

If $S \rightarrow \ldots \rightarrow u \rightarrow t$ is a shortest path from S to t, then

$$d(S, t) = d(S, u) + w(u, t)$$

Edge Relaxation

dist[v] will be an upper bound on the actual distance from S to v.

The edge relaxation procedure for an edge (u, v) just checks whether going from S to v through u improves the current value of dist[v].

Edge Relaxation

```
Relax((u, v) ∈ E)
        if dist[v] > dist[u] + w(u, v):
            dist[v] ← dist[u] + w(u, v)
            prev [v] ← u
```

Naïve Approach

```
Naïve( G, S)
        for all u ∈ V :
            dist[u] ← ∞
            prev [u] ← nil
        dist[S] ← 0
        do:
            relax all the edges
        while at least one dist changes
```

Correct Distances

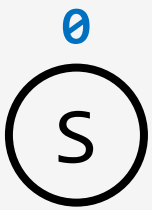
Lemma

After the call to Naïve algorithm all the distances are set correctly.

Dijkstra's Algorithm

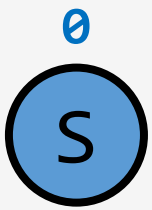
Intuition

initially, we only know the distance to S



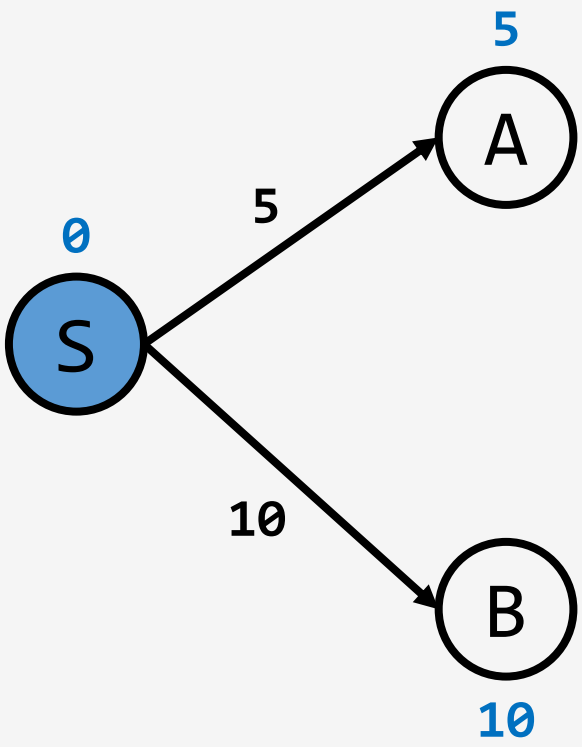
Intuition

let's relax all the edges from S



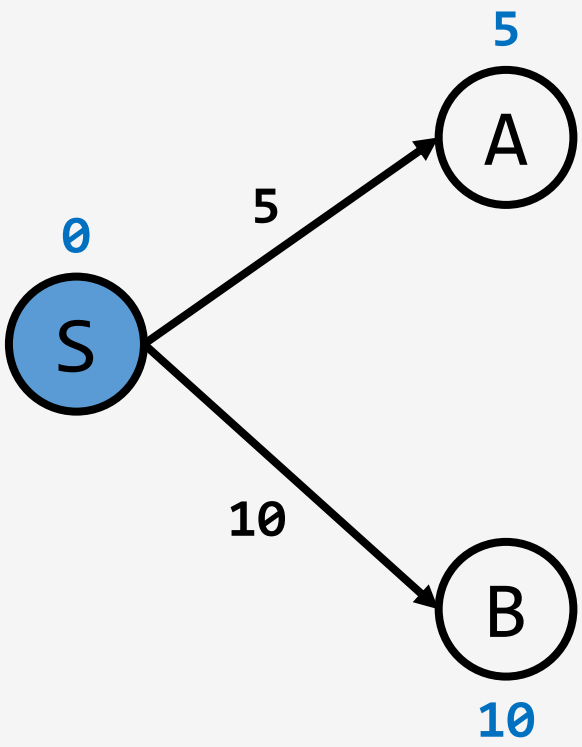
Intuition

let's relax all the edges from S

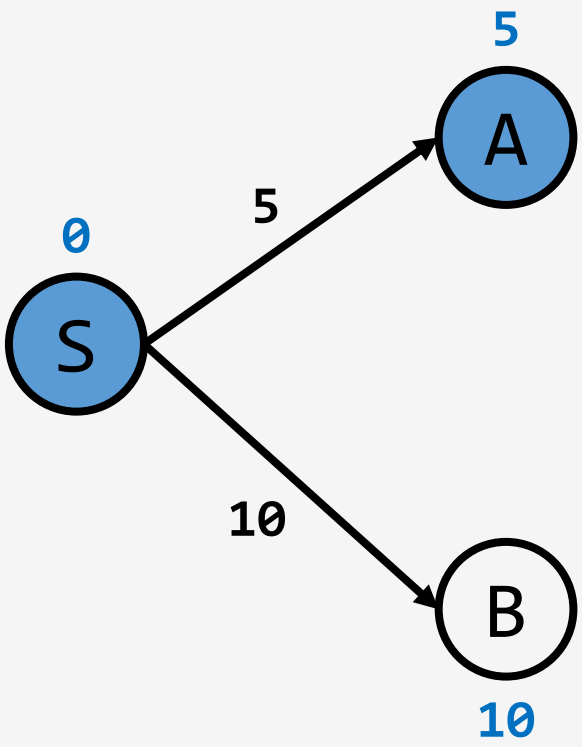


Intuition

we now know the distance for A

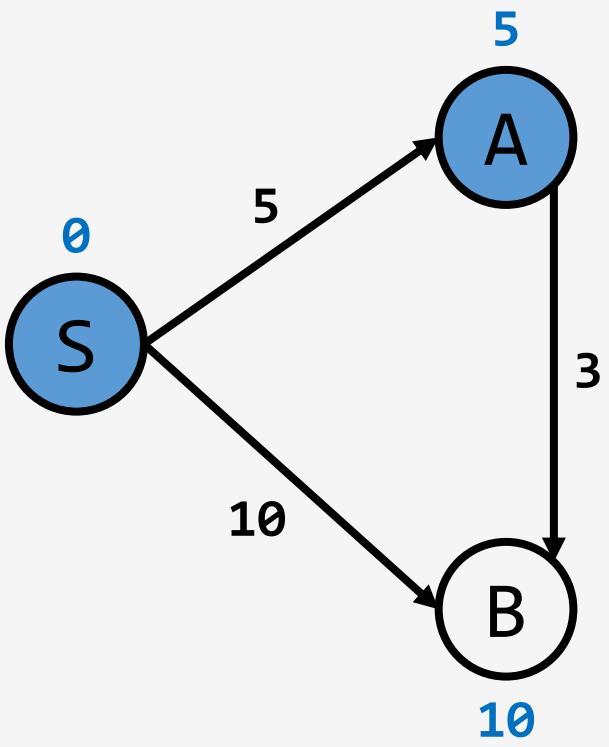


Intuition



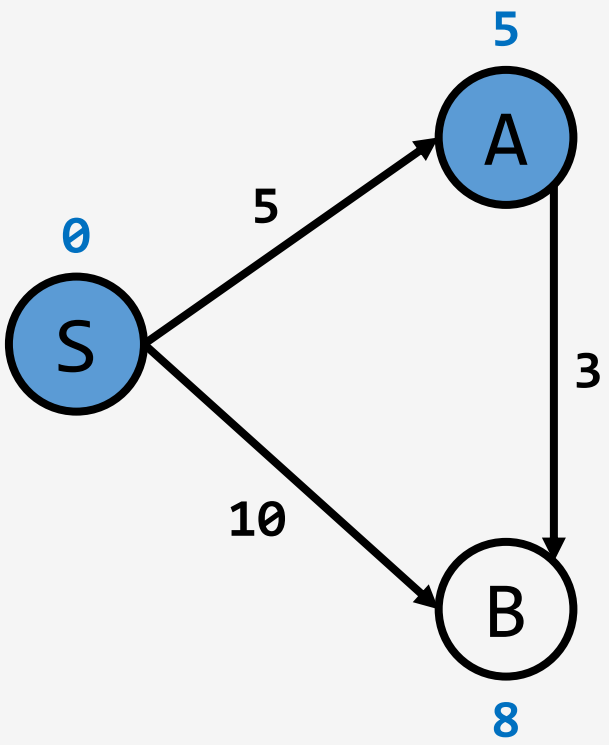
Intuition

now, let's relax all the edges from A



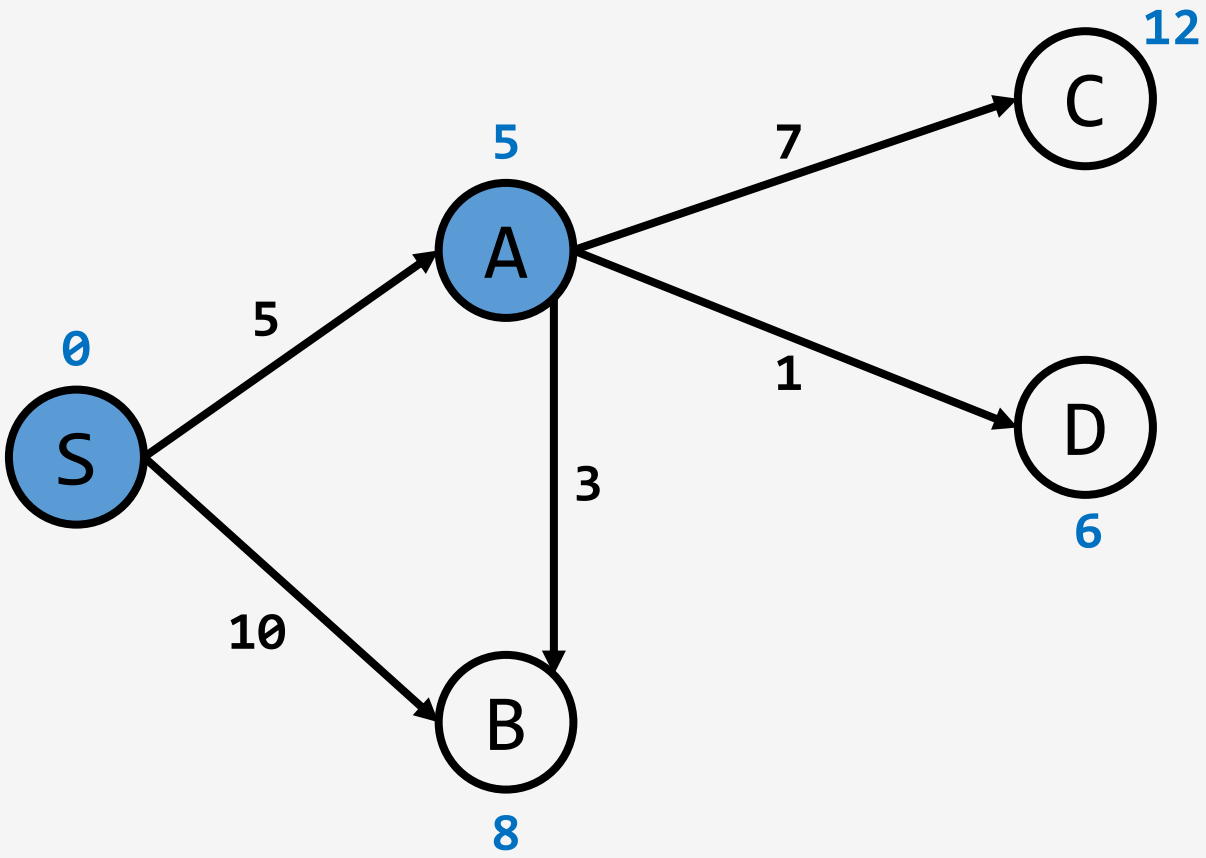
Intuition

we discover an edge (A, B) of weight 3 that updates dist[B]



Intuition

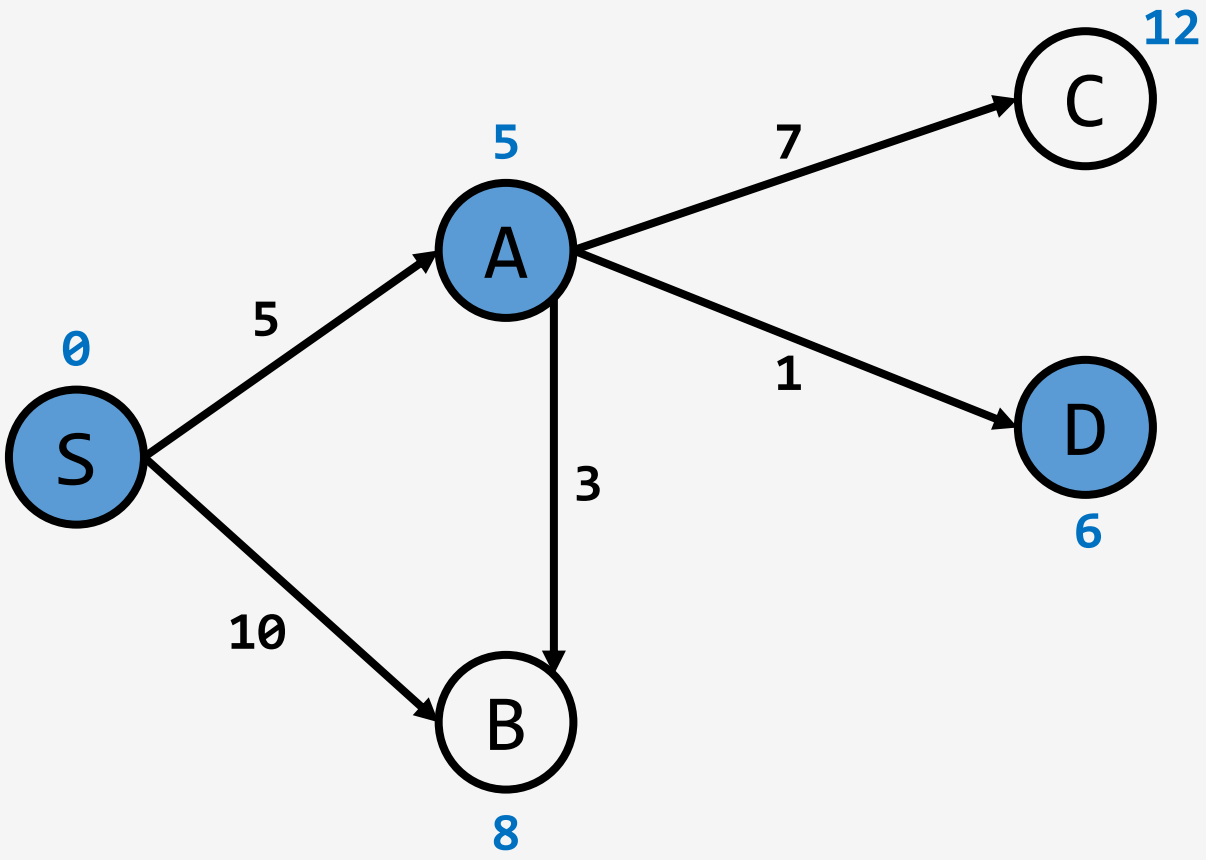
we also discover a few more outgoing edges



what is the next vertex for which we already know the correct distance?

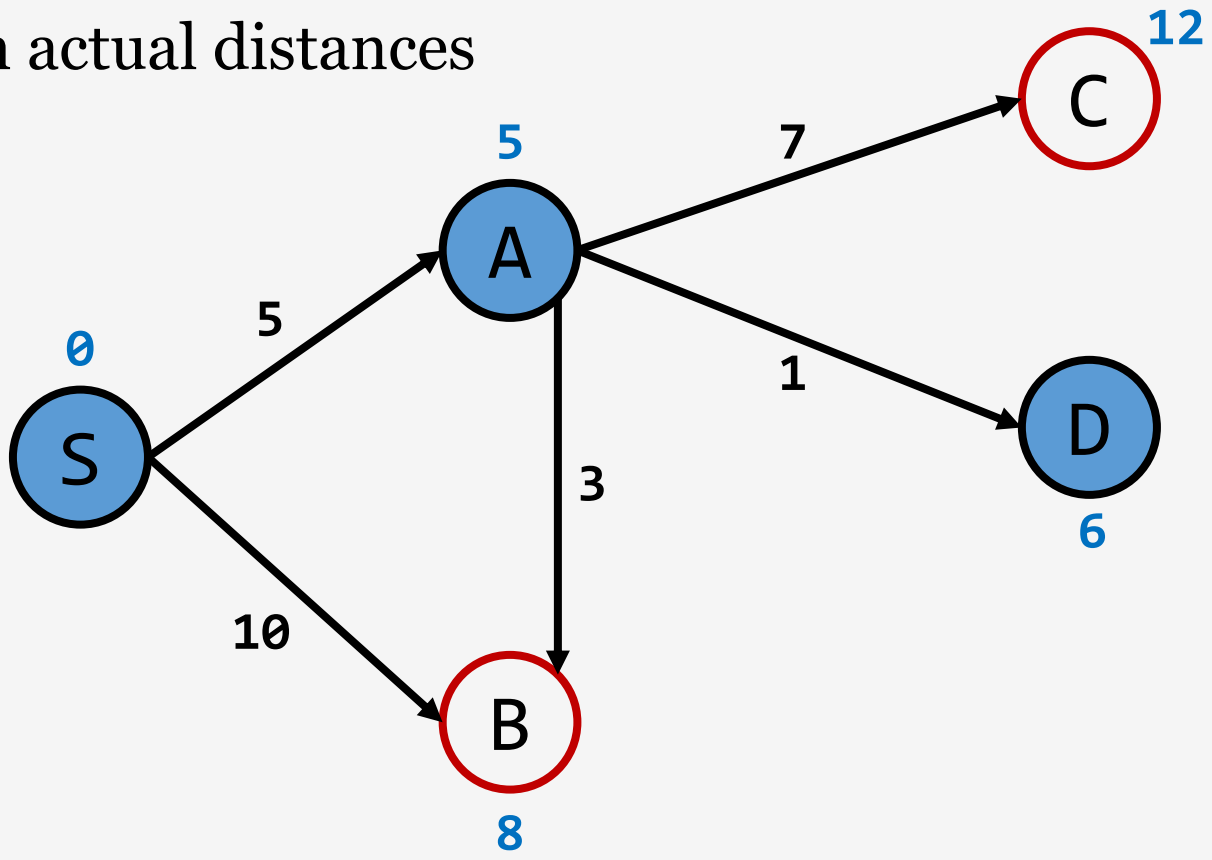
Intuition

It is D



Intuition

while for B and C it is possible that their dist values are larger than actual distances



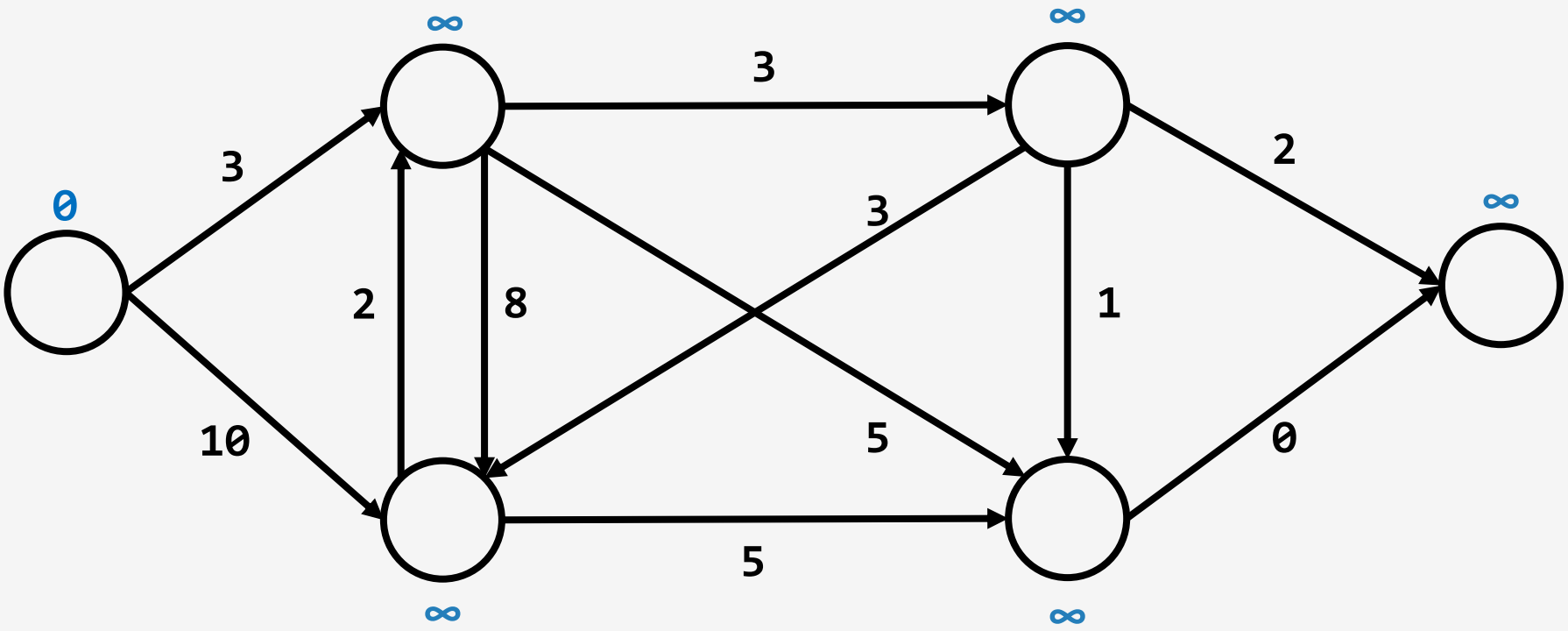
Main ideas of Dijkstra's Algorithm

We maintain a set R of vertices for which dist is already set Correctly ("known region").

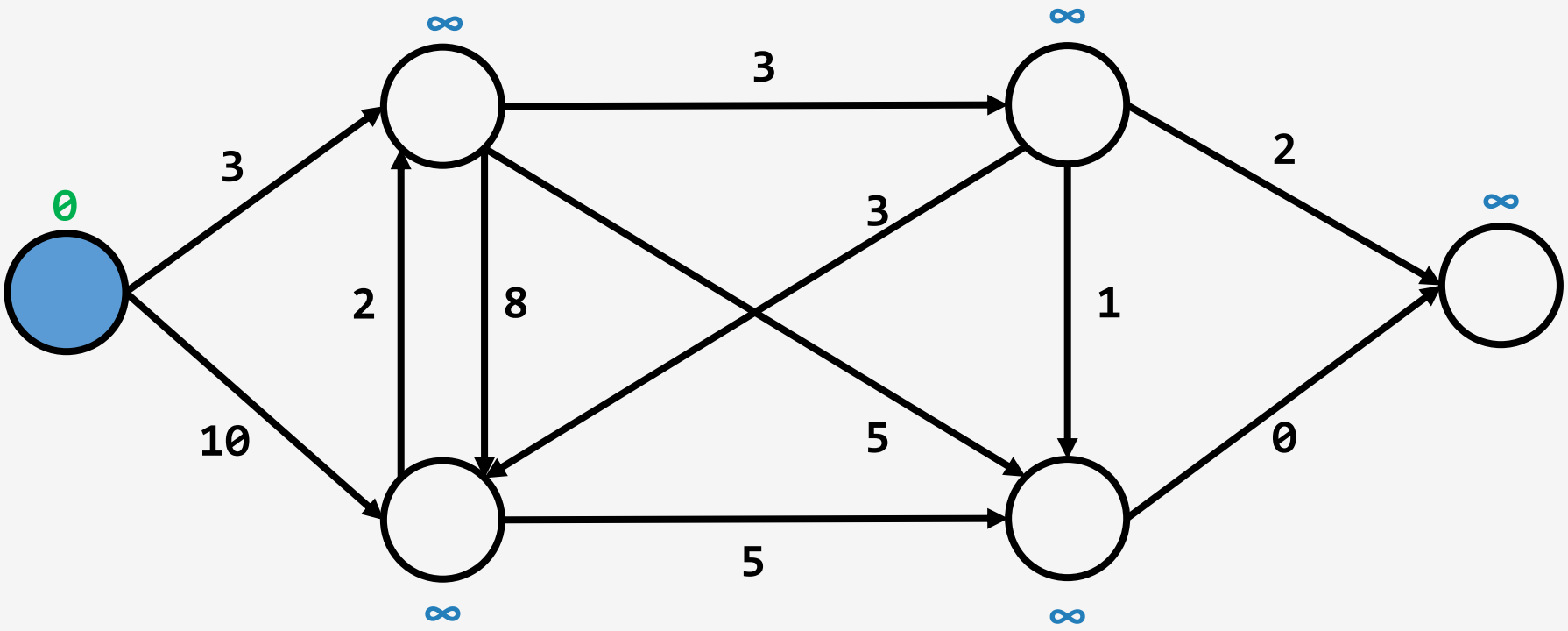
The first vertex added to R is S.

On each iteration we take a vertex outside of R with the minimal dist-value, add it to R, and relax all its outgoing edges.

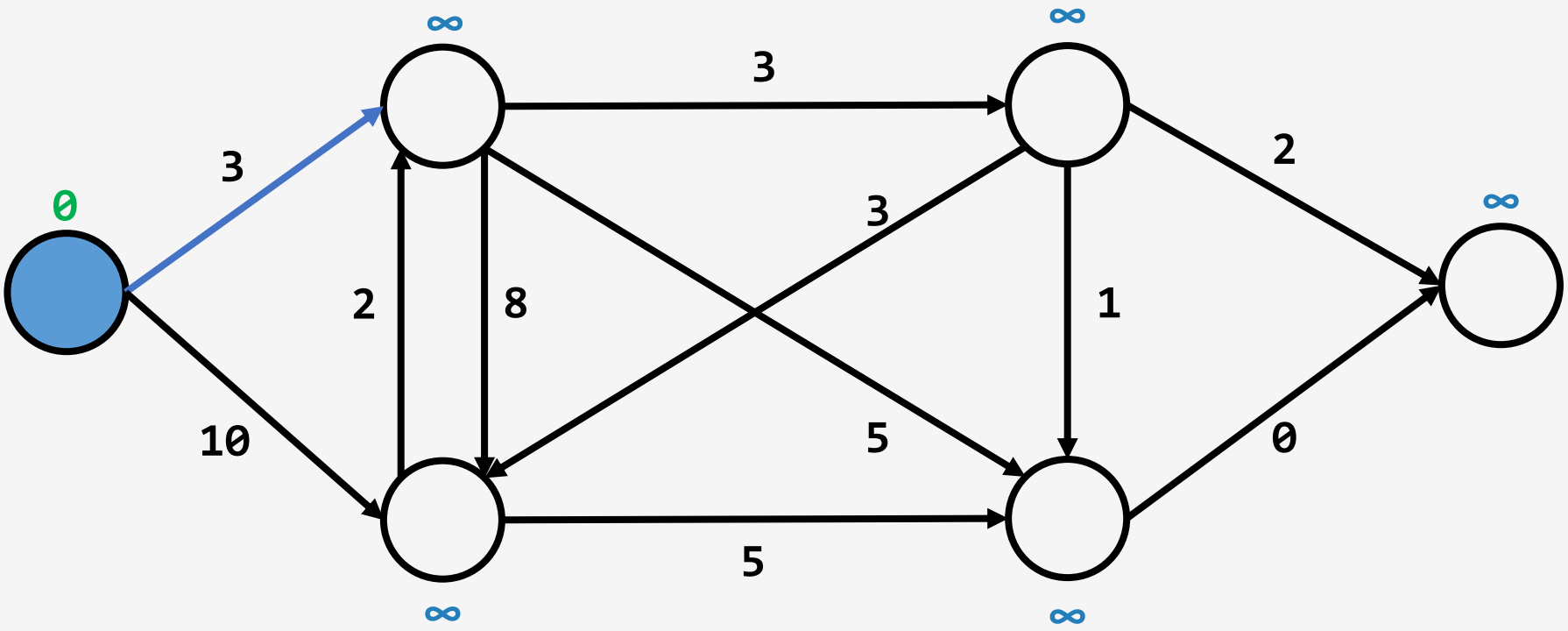
Example: Dijkstra's Algorithm



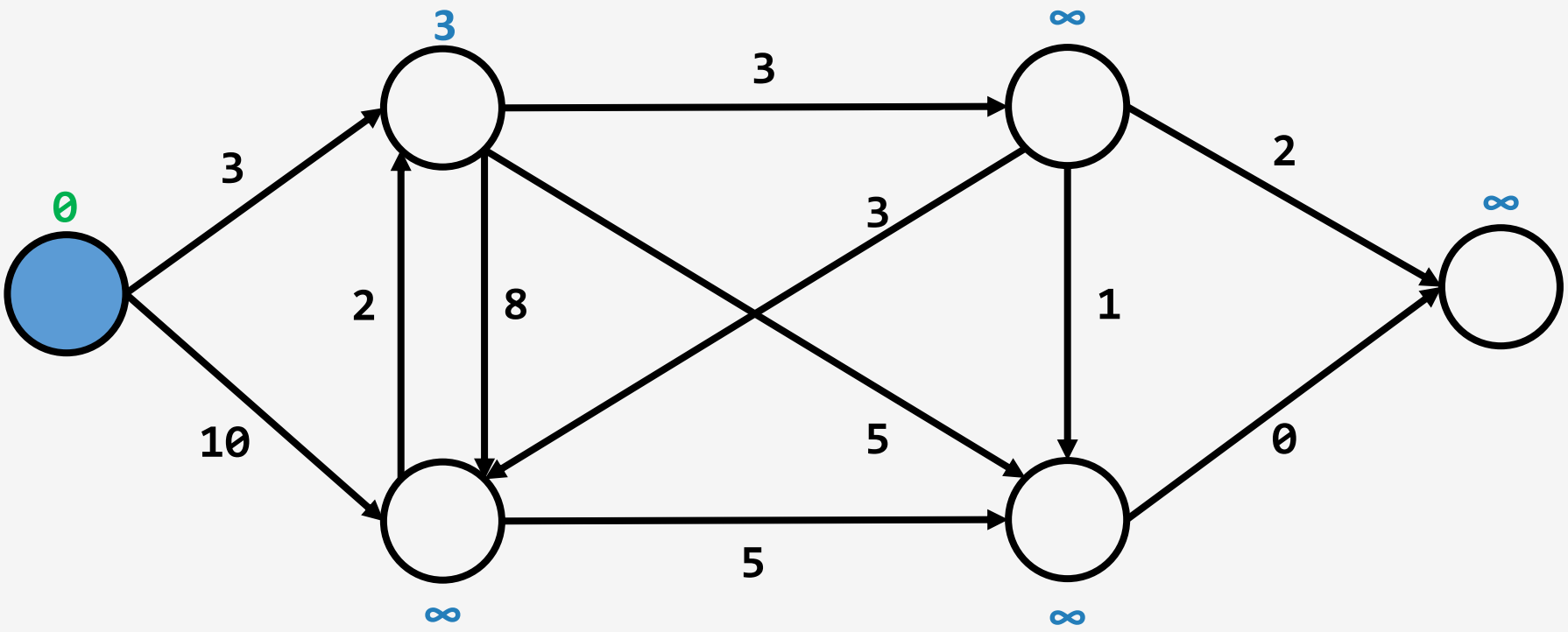
Example: Dijkstra's Algorithm



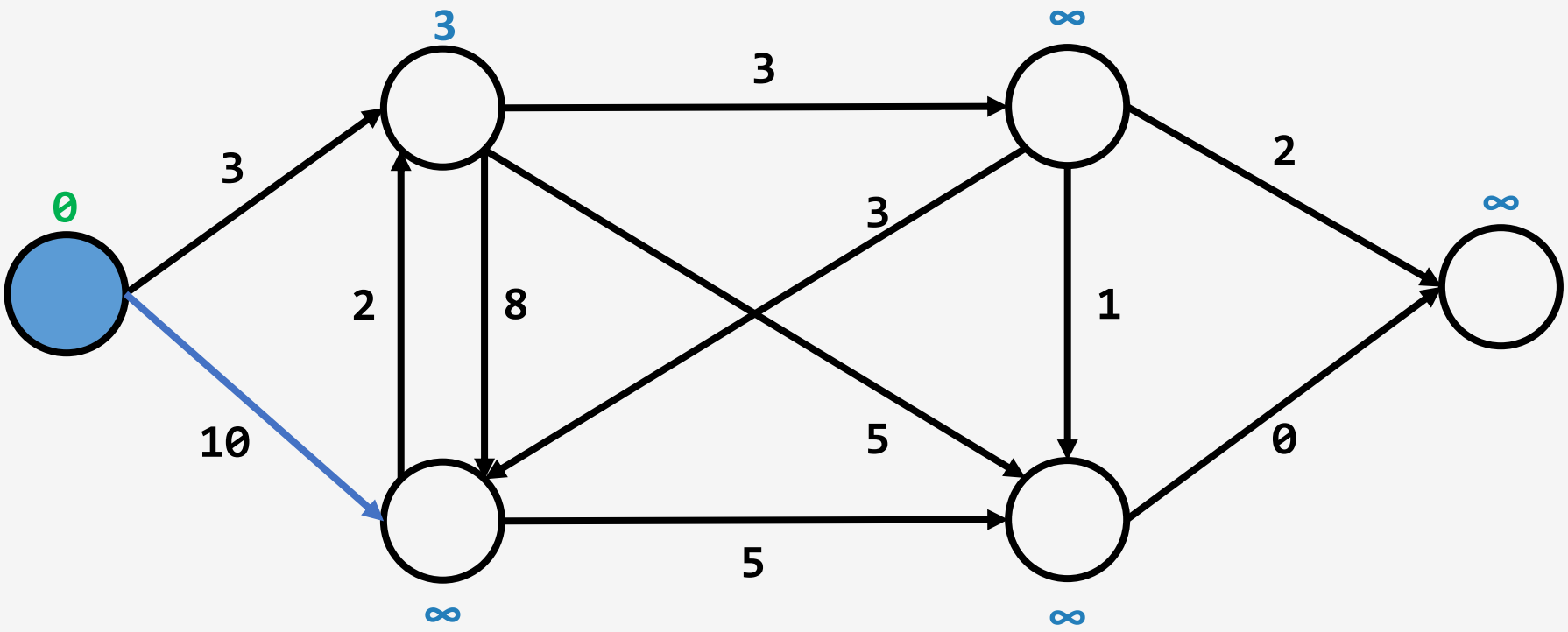
Example: Dijkstra's Algorithm



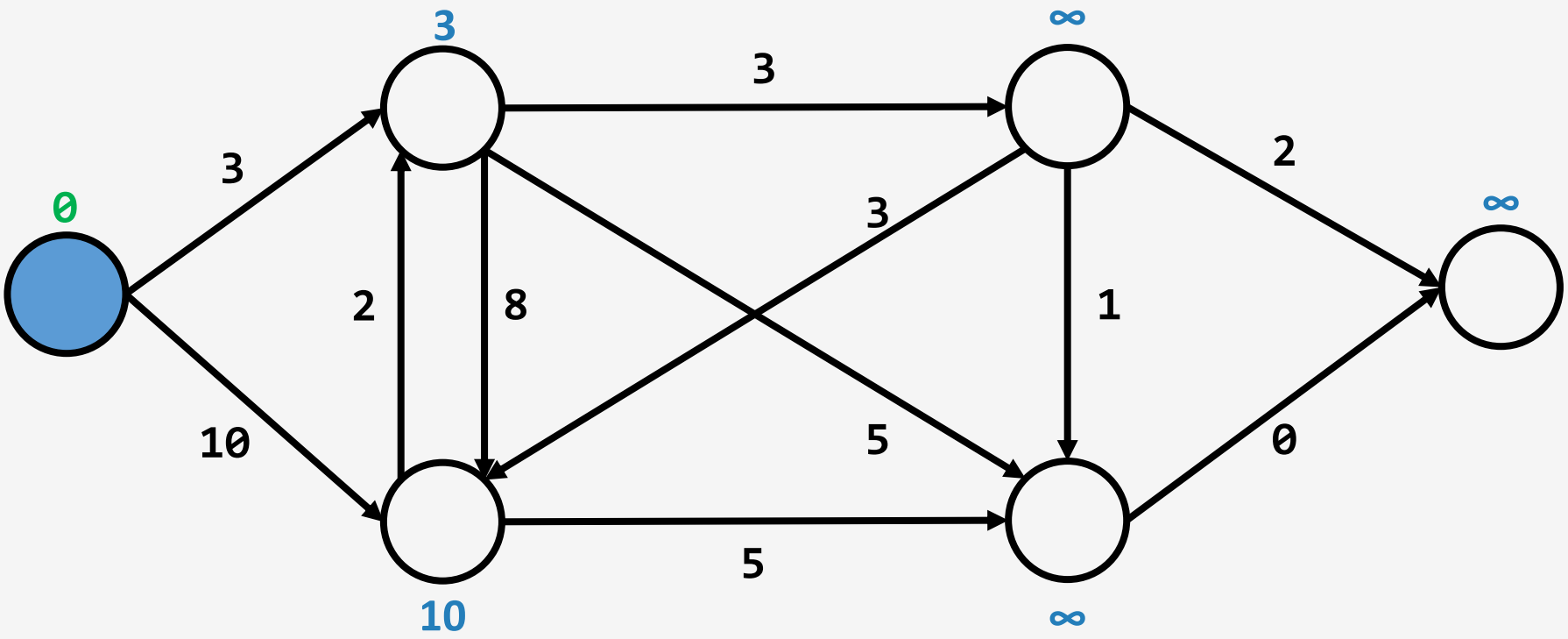
Example: Dijkstra's Algorithm



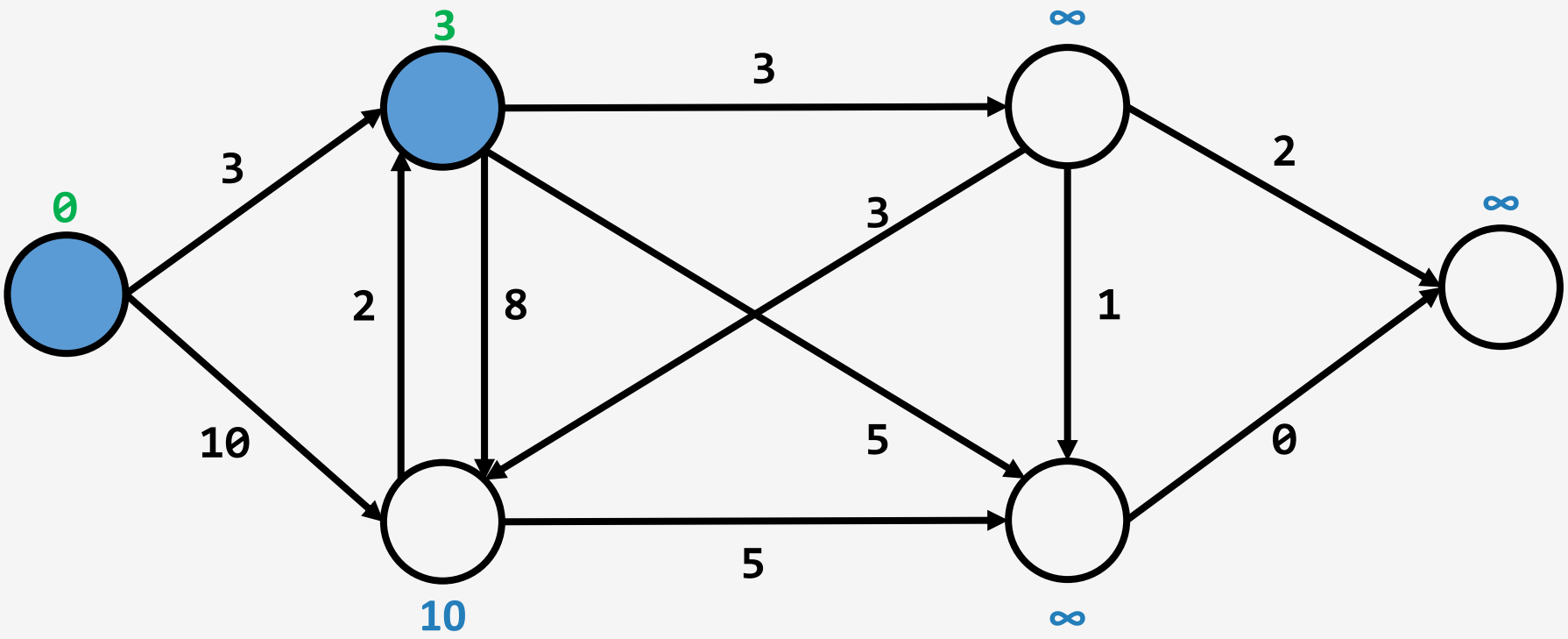
Example: Dijkstra's Algorithm



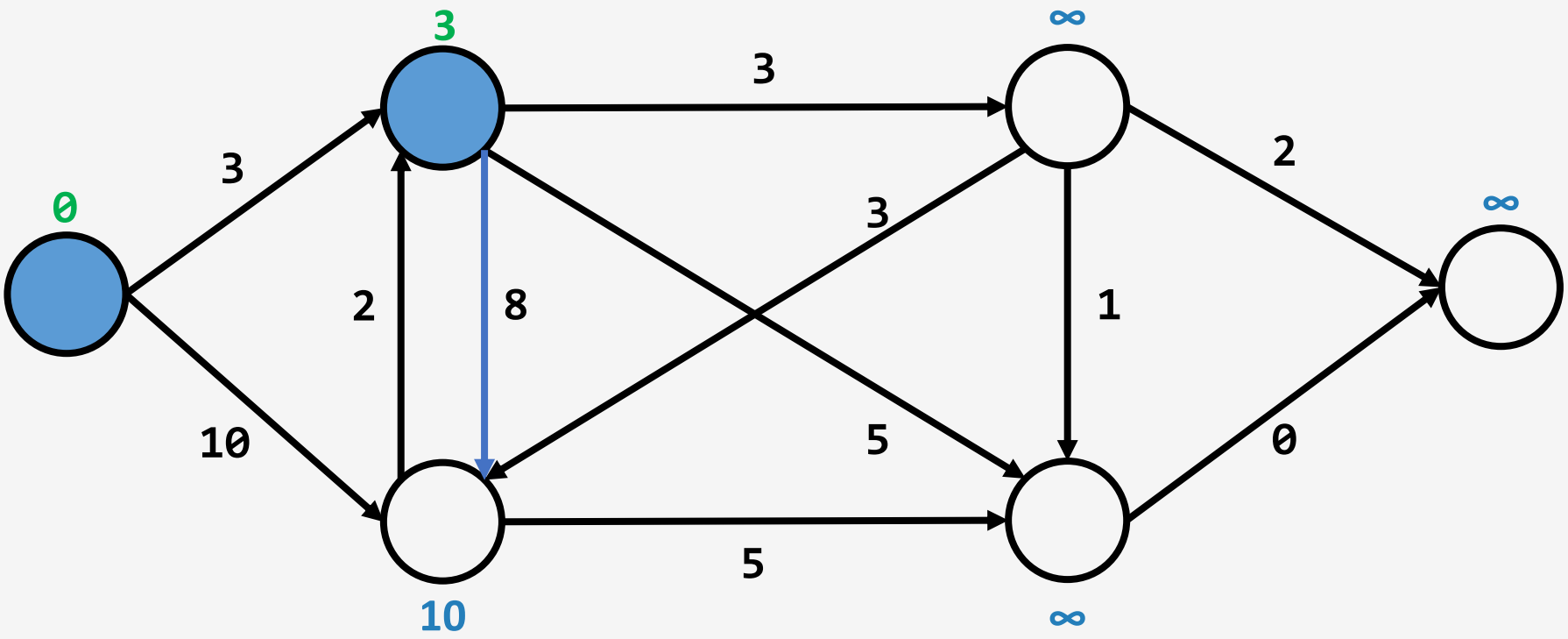
Example: Dijkstra's Algorithm



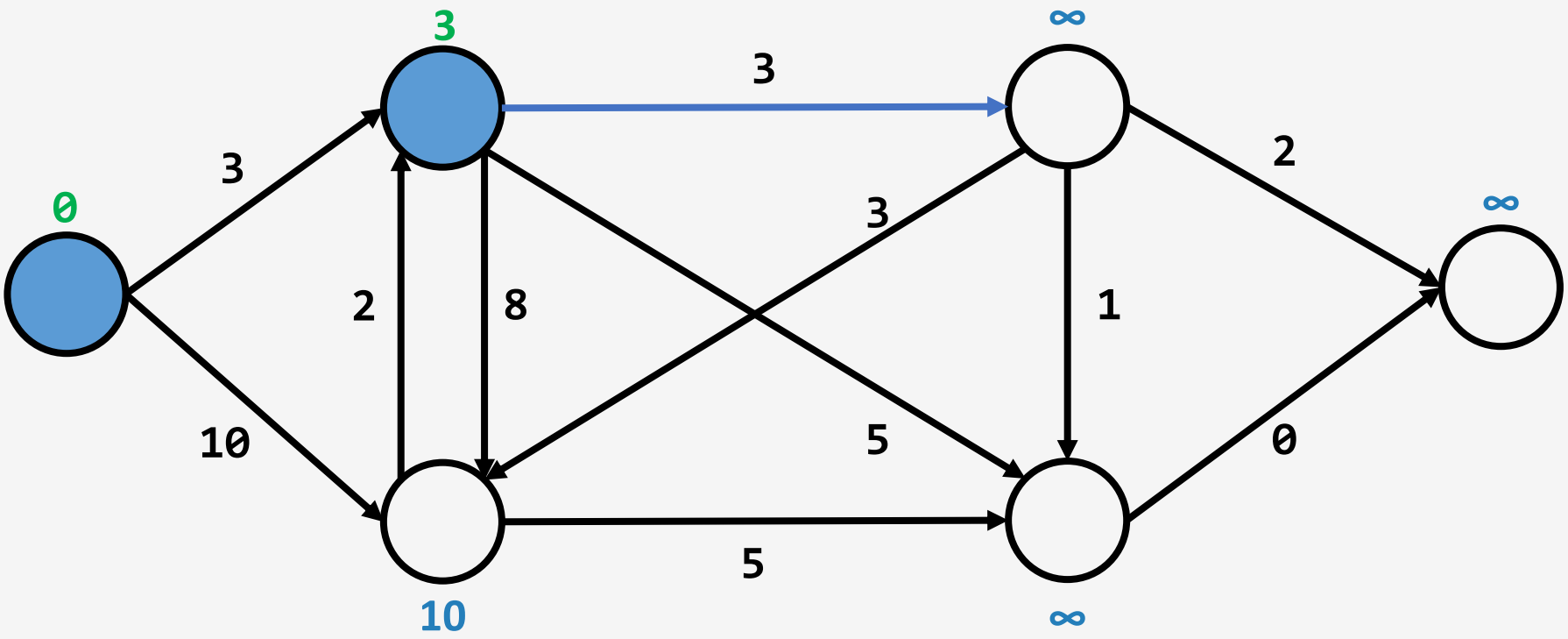
Example: Dijkstra's Algorithm



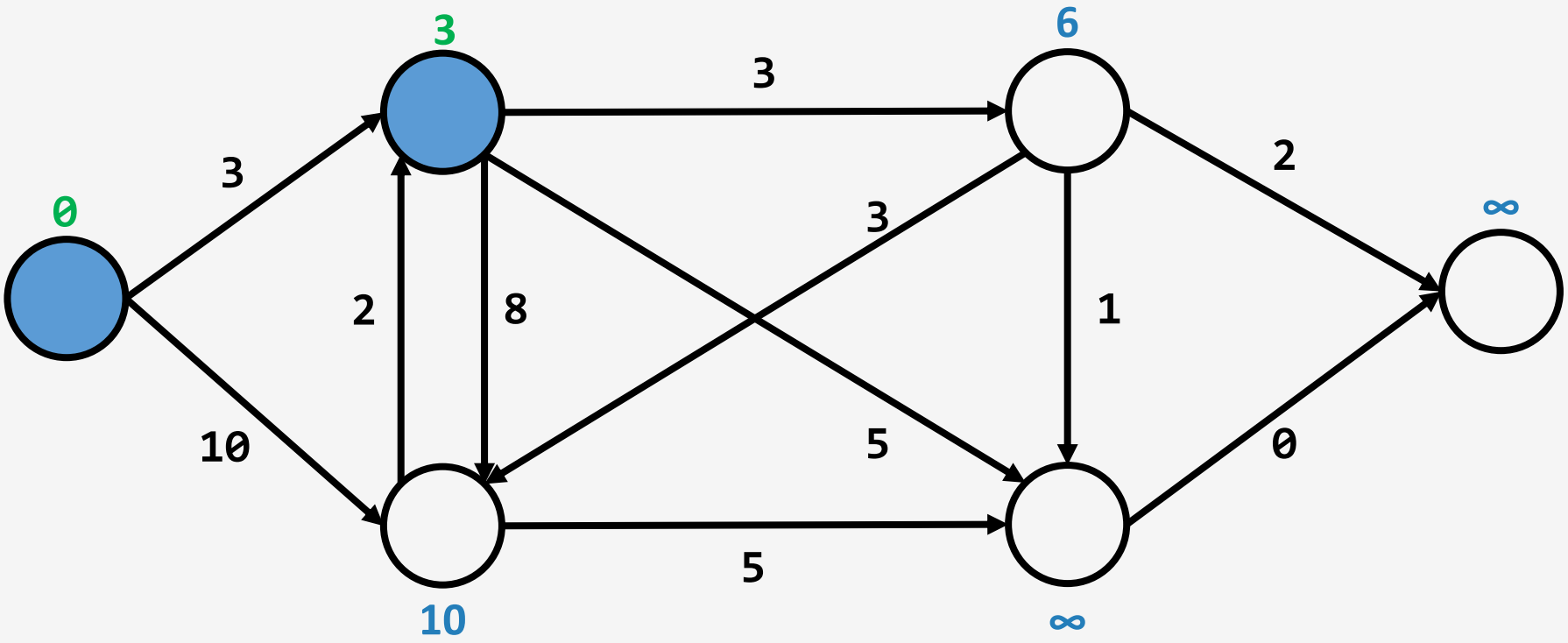
Example: Dijkstra's Algorithm



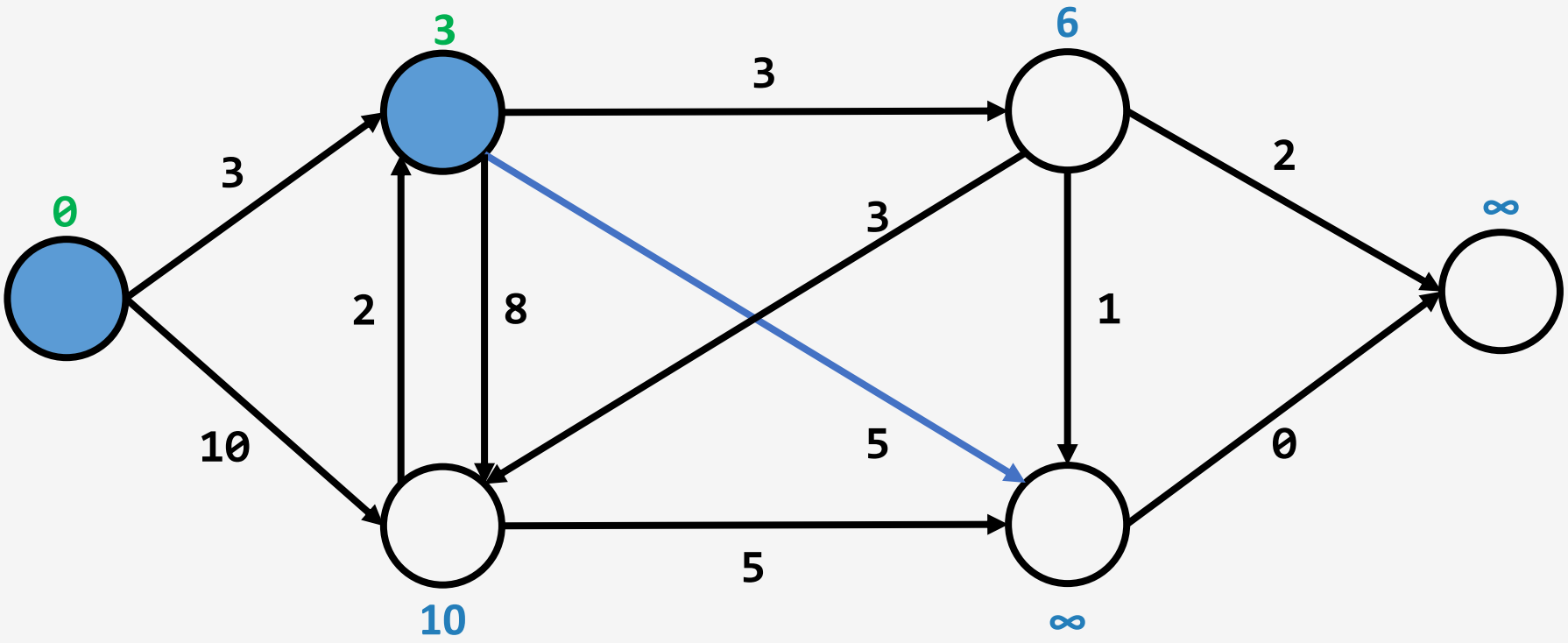
Example: Dijkstra's Algorithm



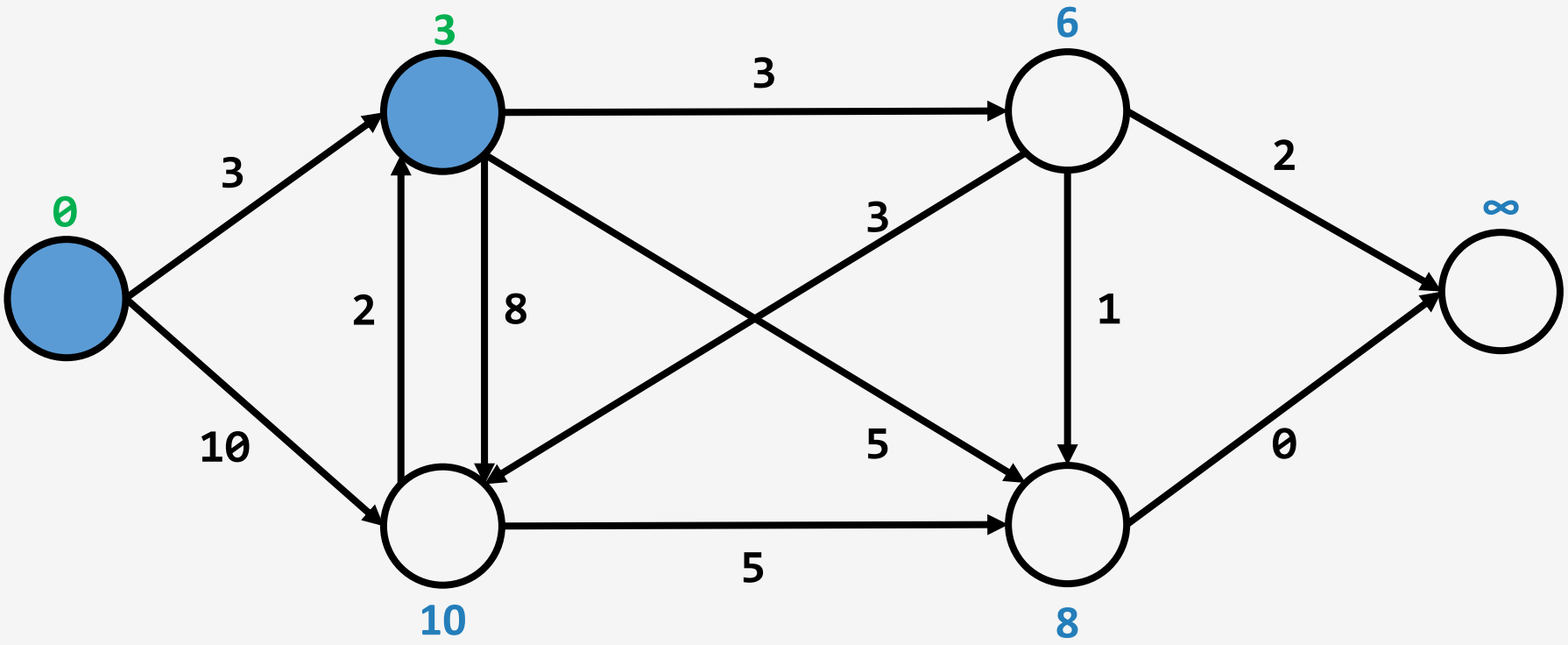
Example: Dijkstra's Algorithm



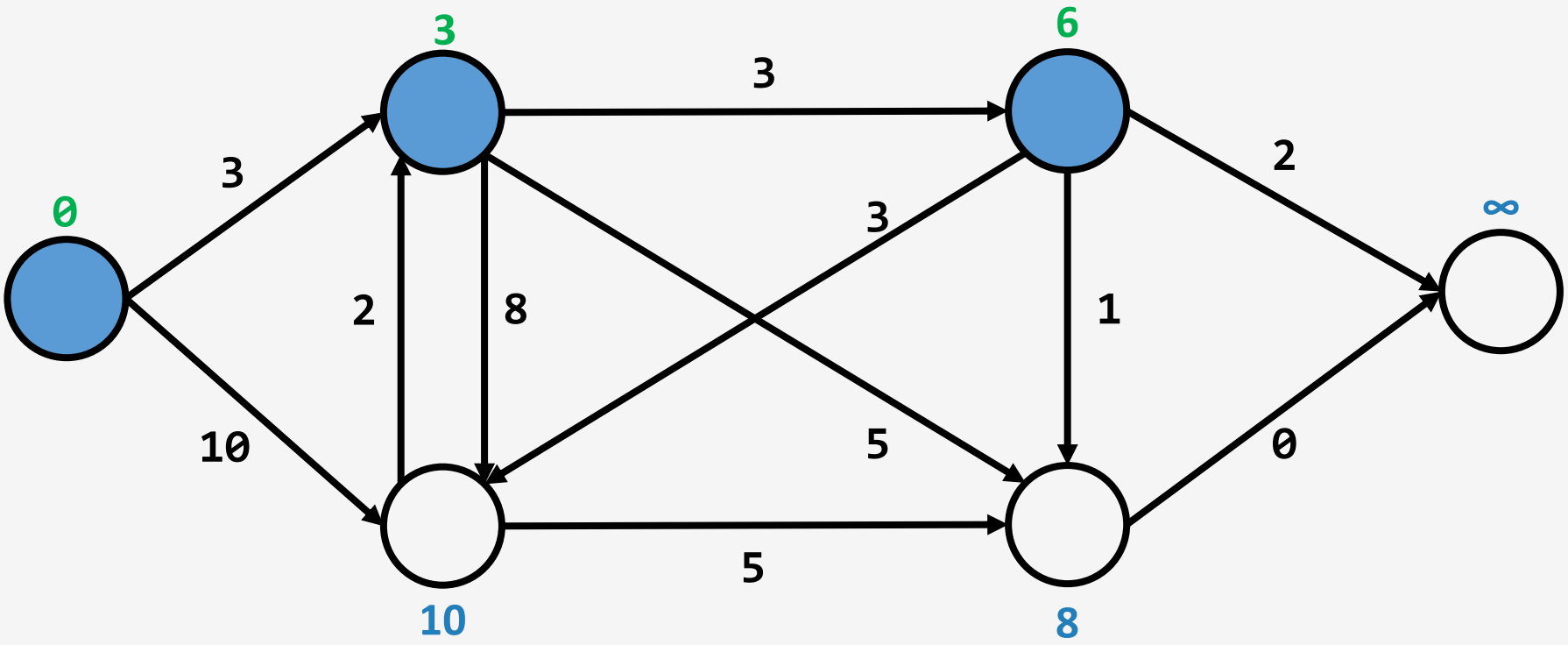
Example: Dijkstra's Algorithm



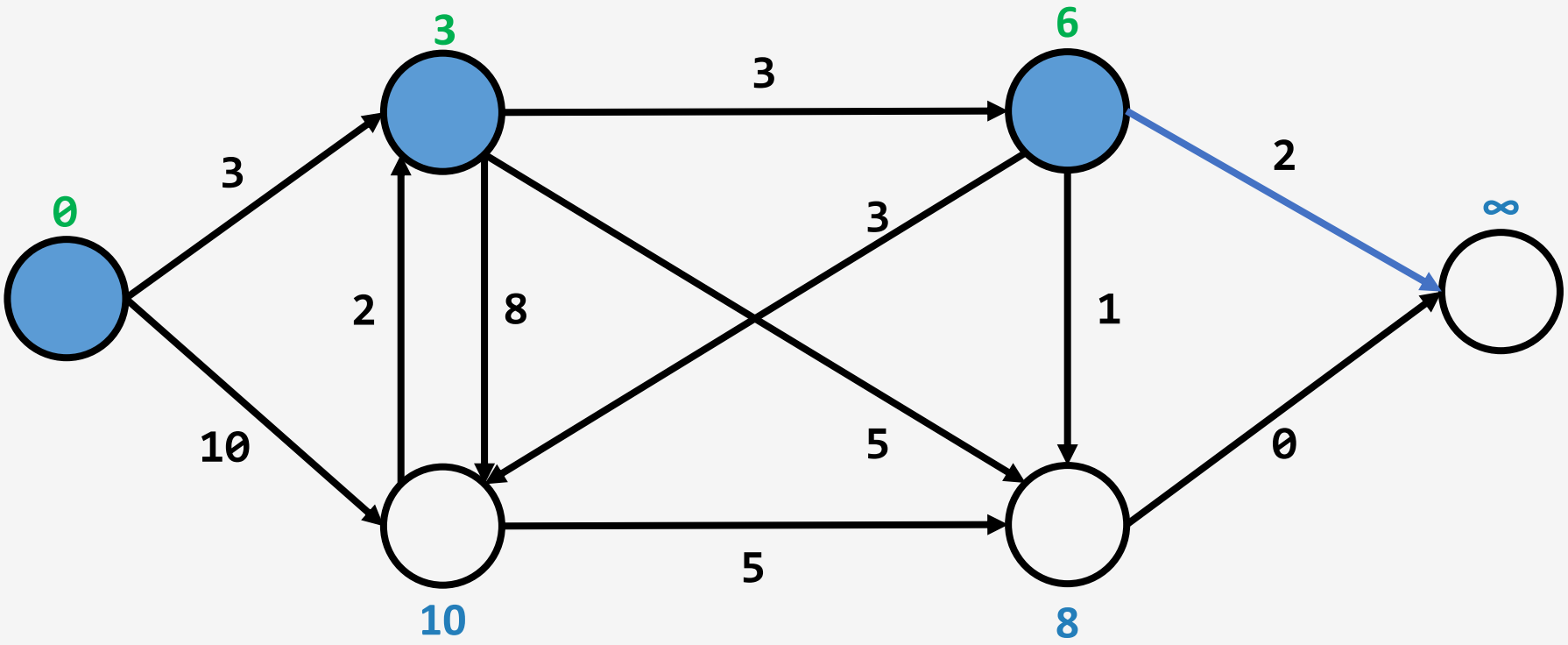
Example: Dijkstra's Algorithm



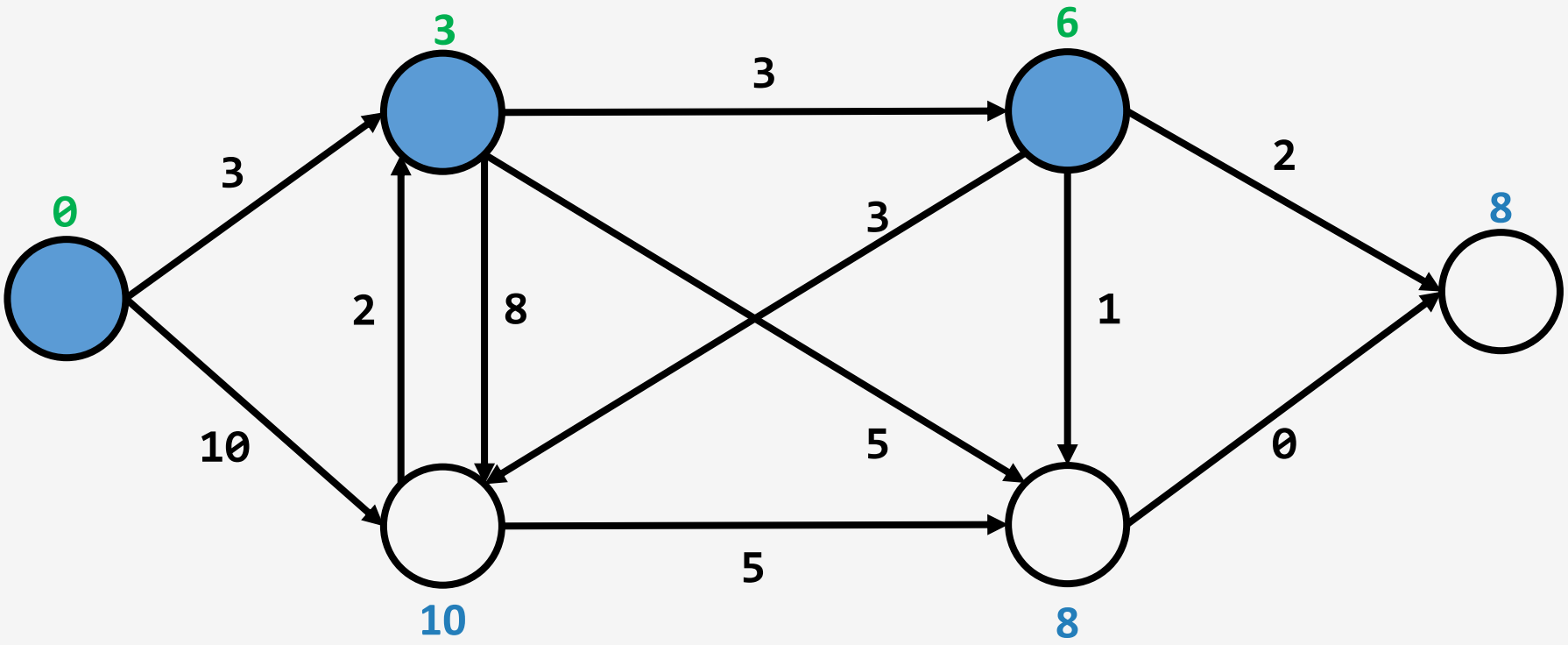
Example: Dijkstra's Algorithm



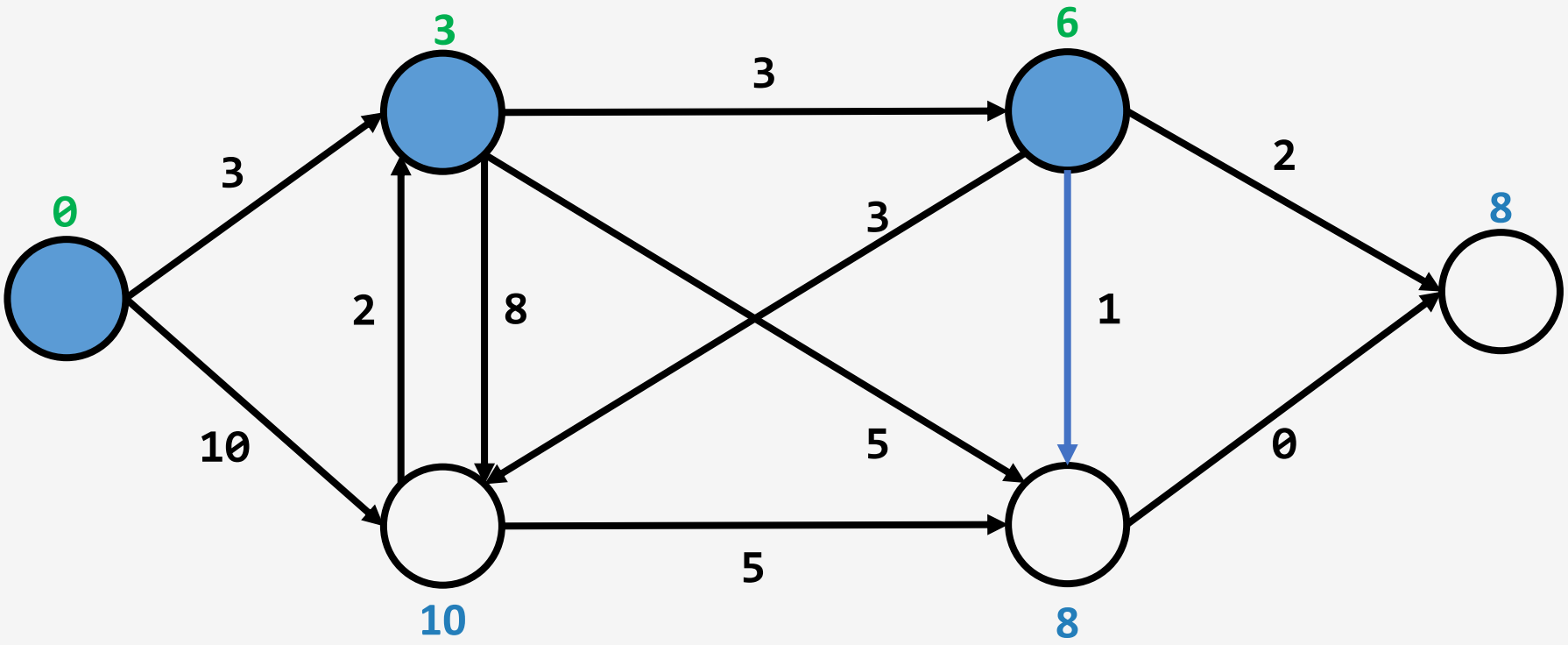
Example: Dijkstra's Algorithm



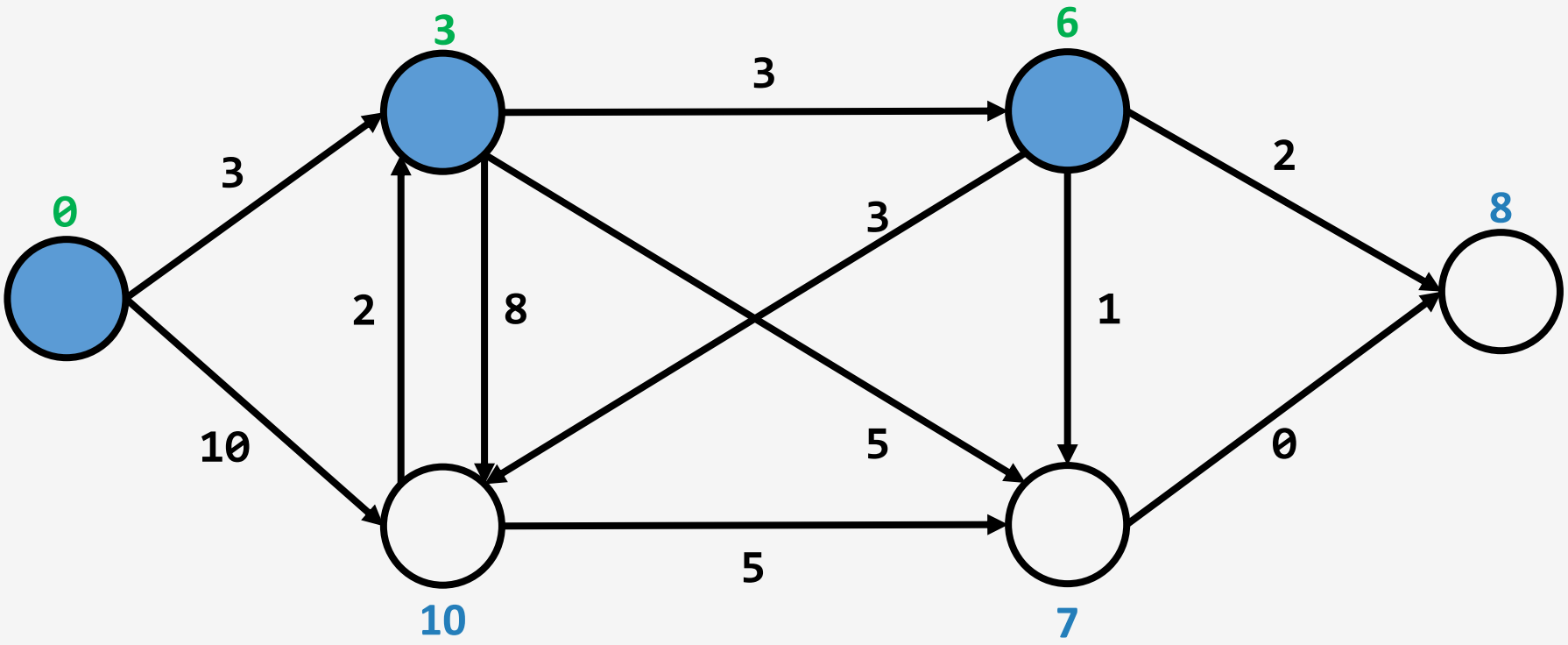
Example: Dijkstra's Algorithm



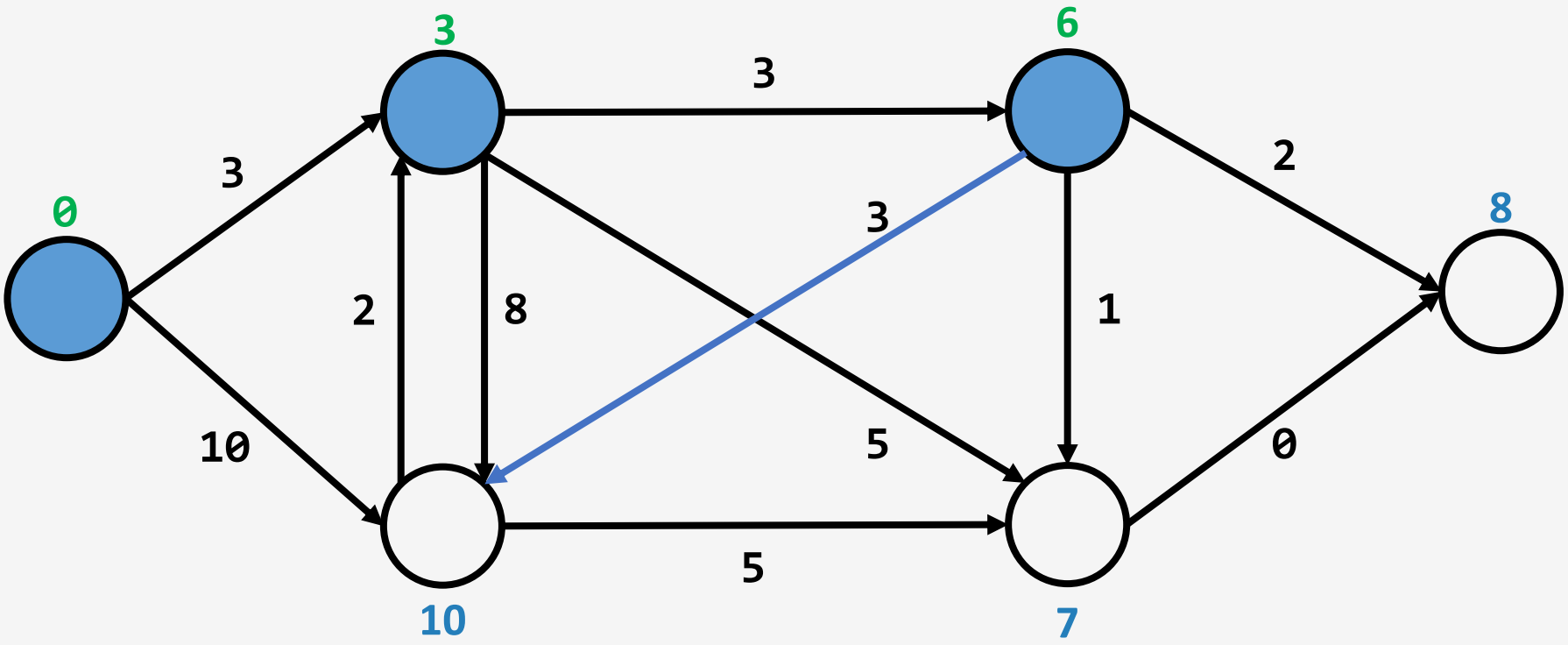
Example: Dijkstra's Algorithm



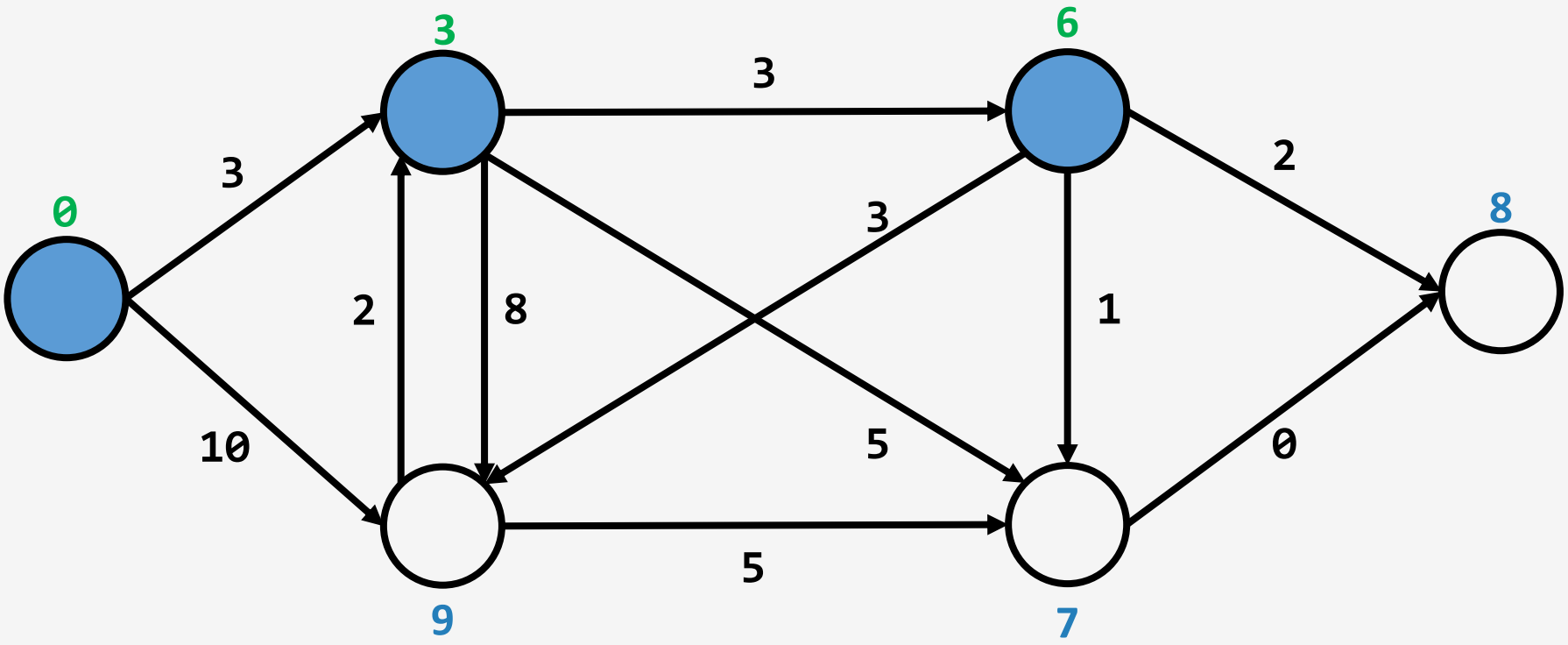
Example: Dijkstra's Algorithm



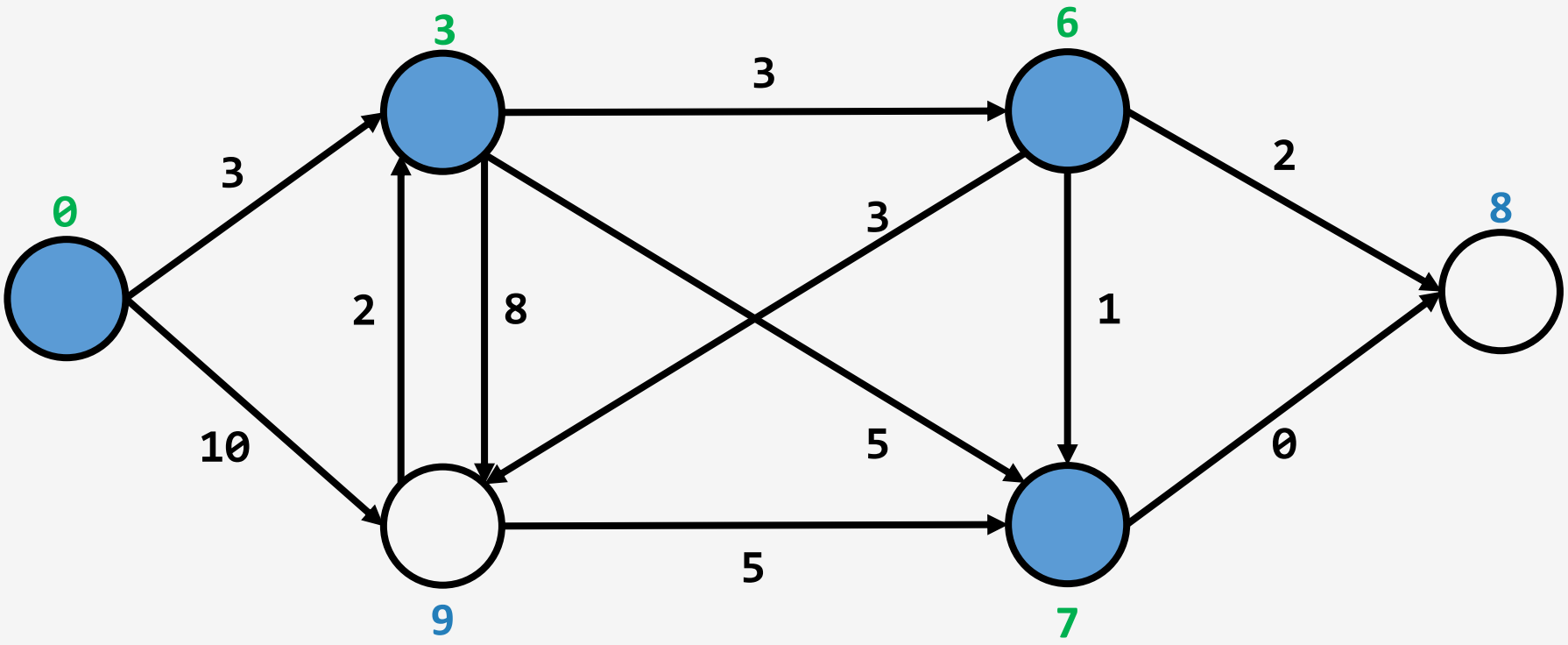
Example: Dijkstra's Algorithm



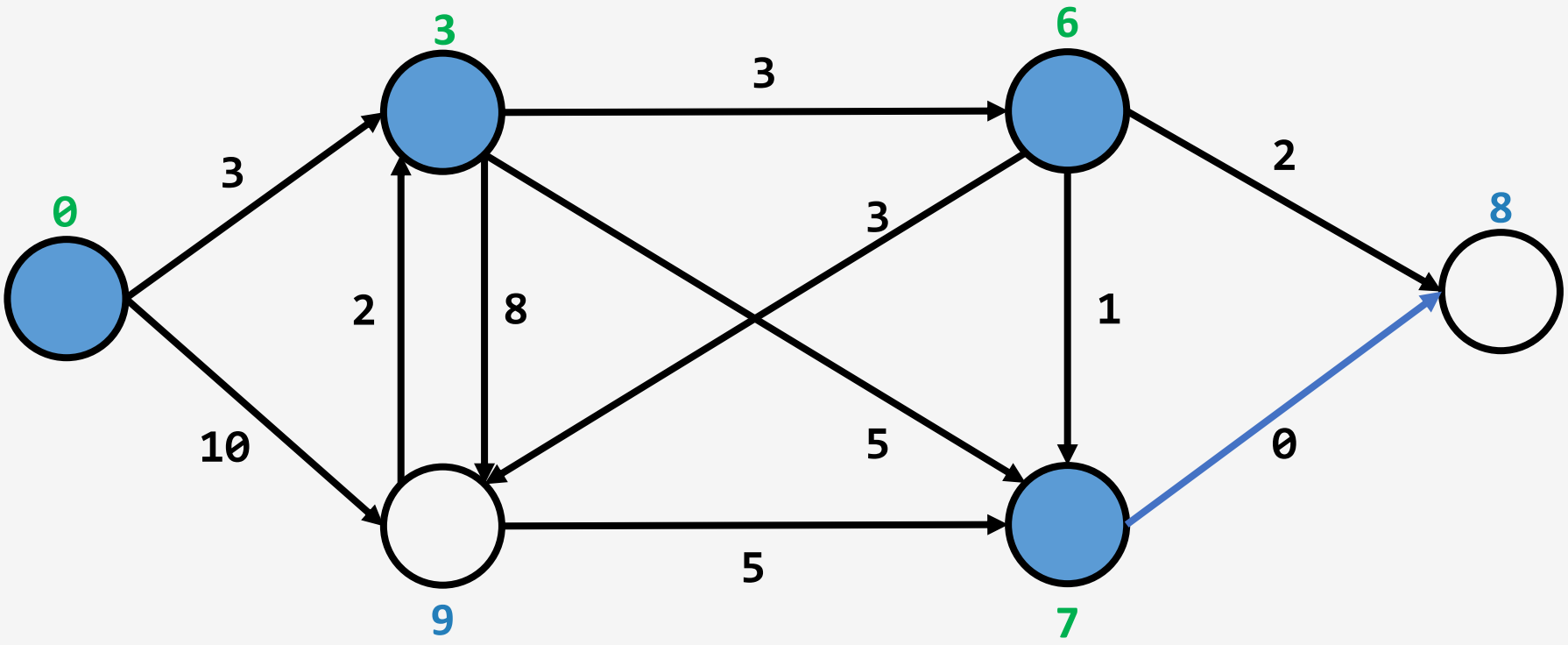
Example: Dijkstra's Algorithm



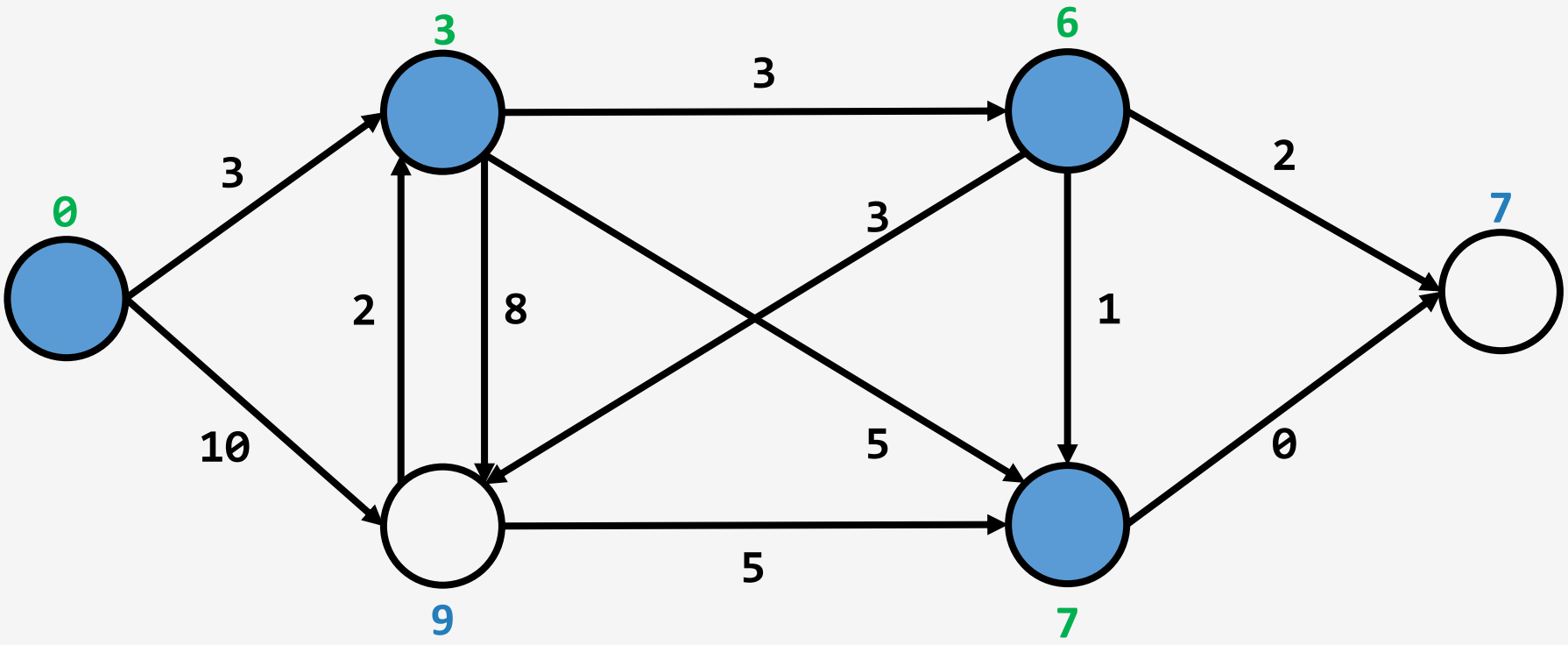
Example: Dijkstra's Algorithm



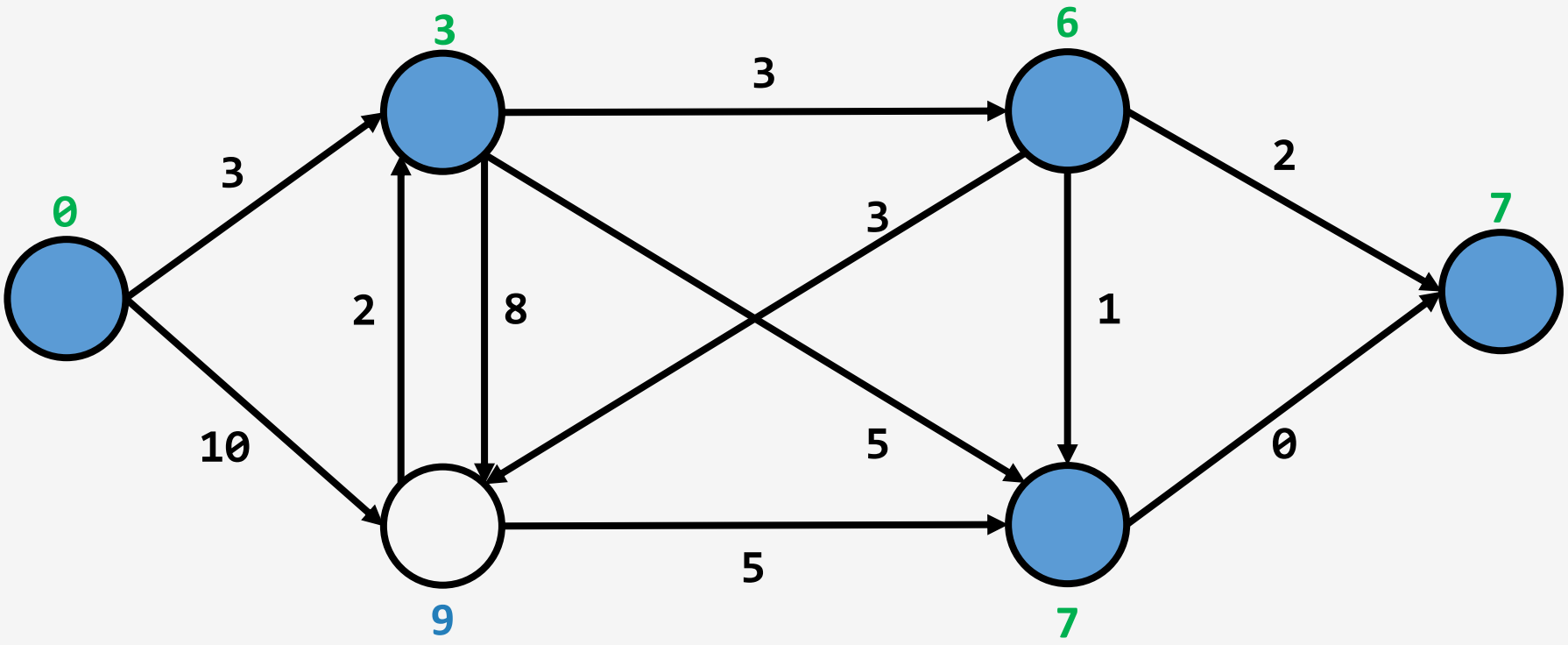
Example: Dijkstra's Algorithm



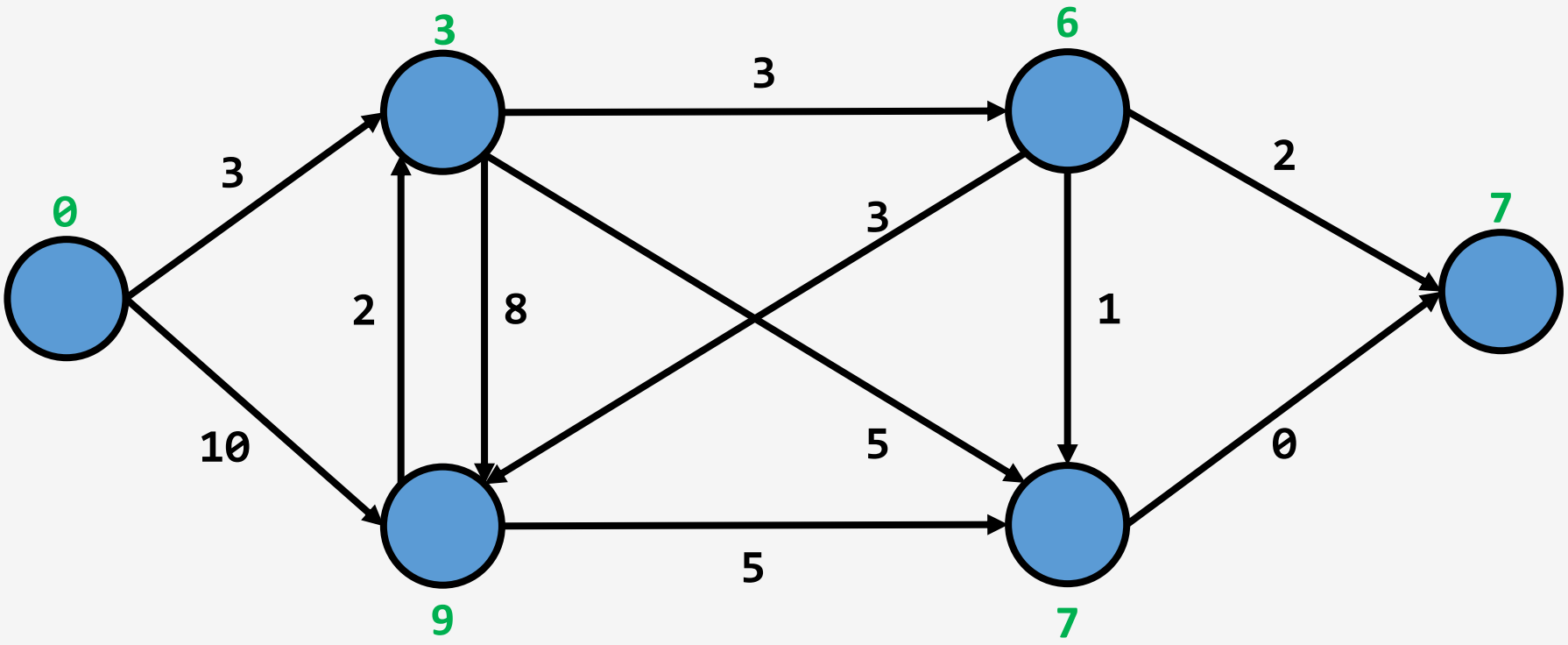
Example: Dijkstra's Algorithm



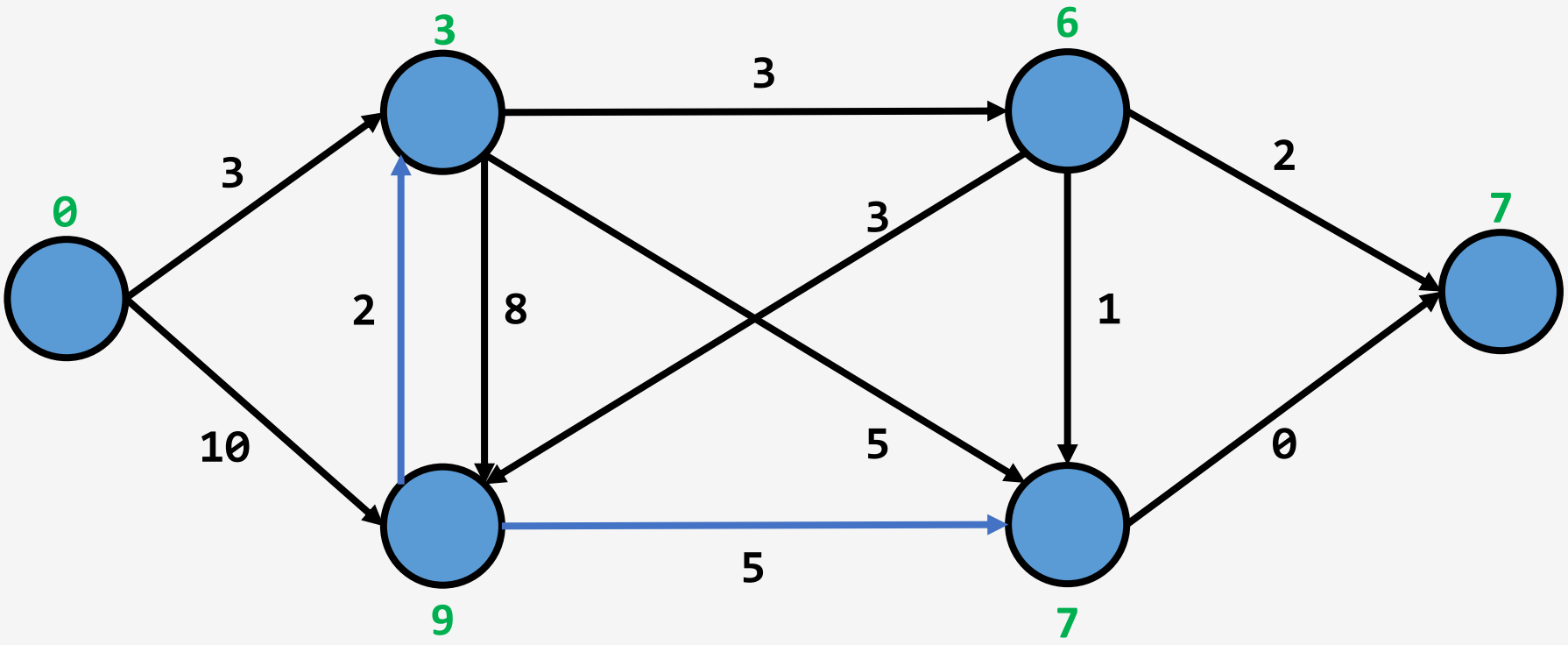
Example: Dijkstra's Algorithm



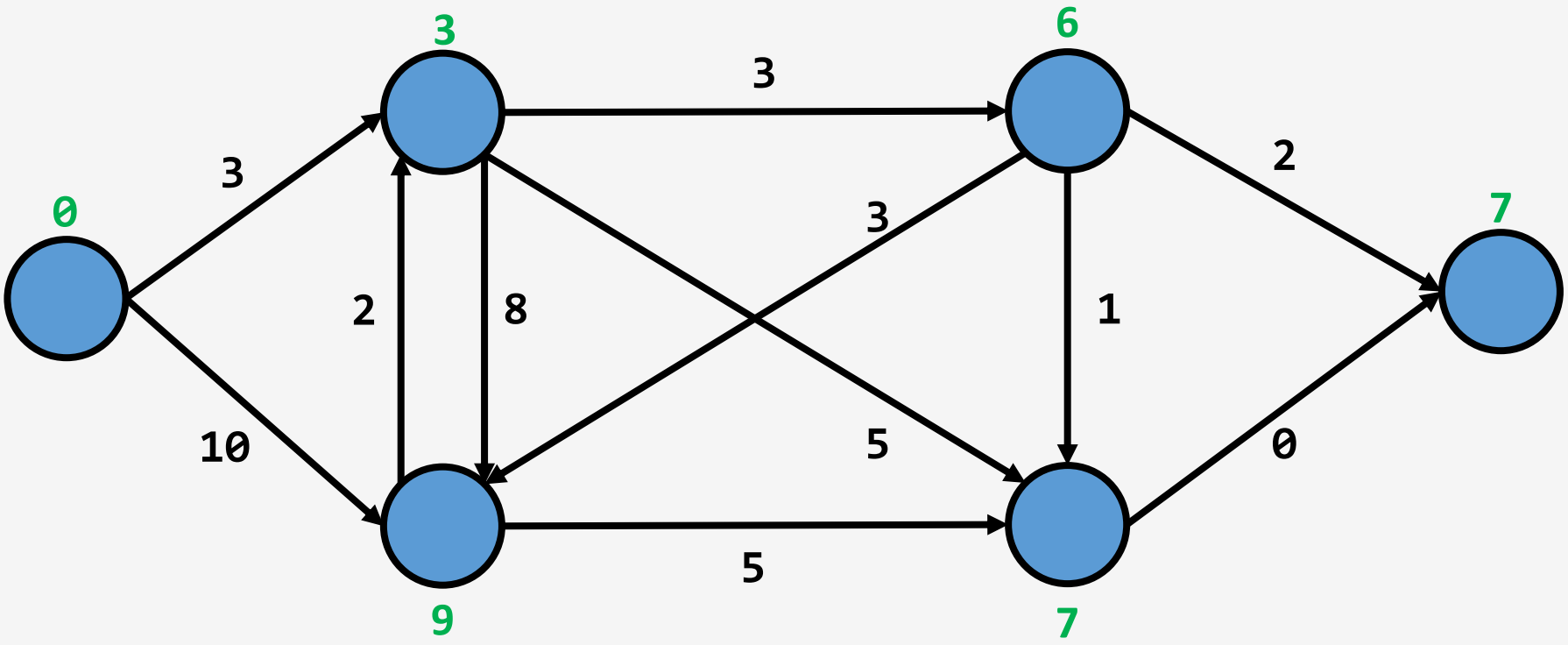
Example: Dijkstra's Algorithm



Example: Dijkstra's Algorithm



Example: Dijkstra's Algorithm



Pseudocode

```
Dijkstra(G, S)
      for all u ∈ V:
         dist[u] ← ∞, prev[u] ← nil
      dist[S] ← 0
      H ← MakeQueue(V) {dist-values as keys}
      while H is not empty:
         u ← ExtractMin(H)
         for all (u, v) ∈ E:
            if dist[v] > dist[u] + w(u, v):
               dist[v] ← dist[u] + w(u, v)
               prev [v] ← u
               ChangePriority(H, v, dist[v]).
```

Conclusion

Can find the minimum time to get from work to home

Can find the fastest route from work to home

Works for any graph with non-negative edge weights

Works in $O(|V|^2)$ or $O((|V| + |E|) \log(|V|))$
depending on the implementation