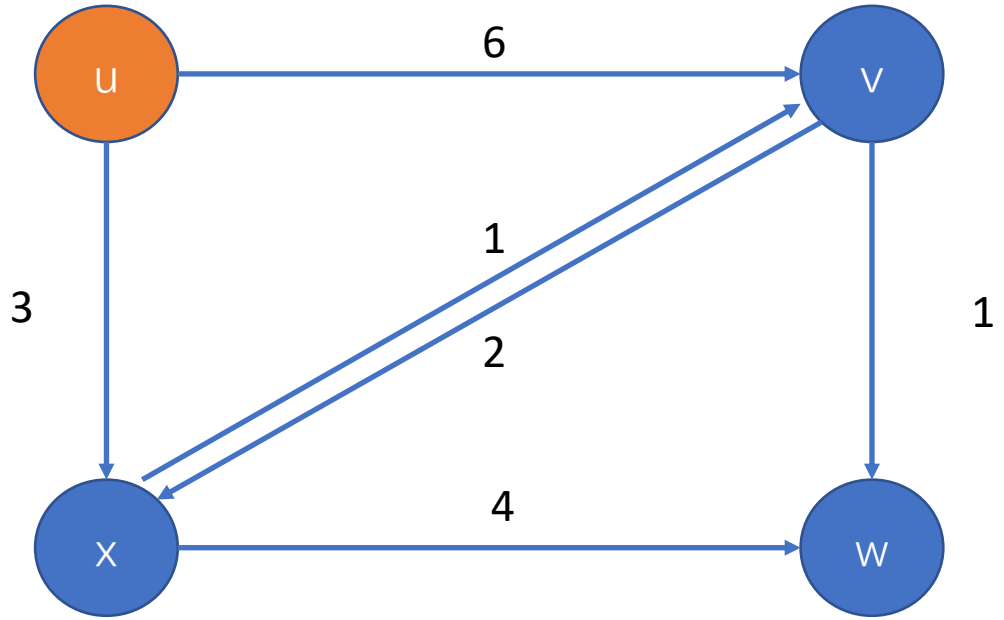
COMP 3711
Tutorial 9

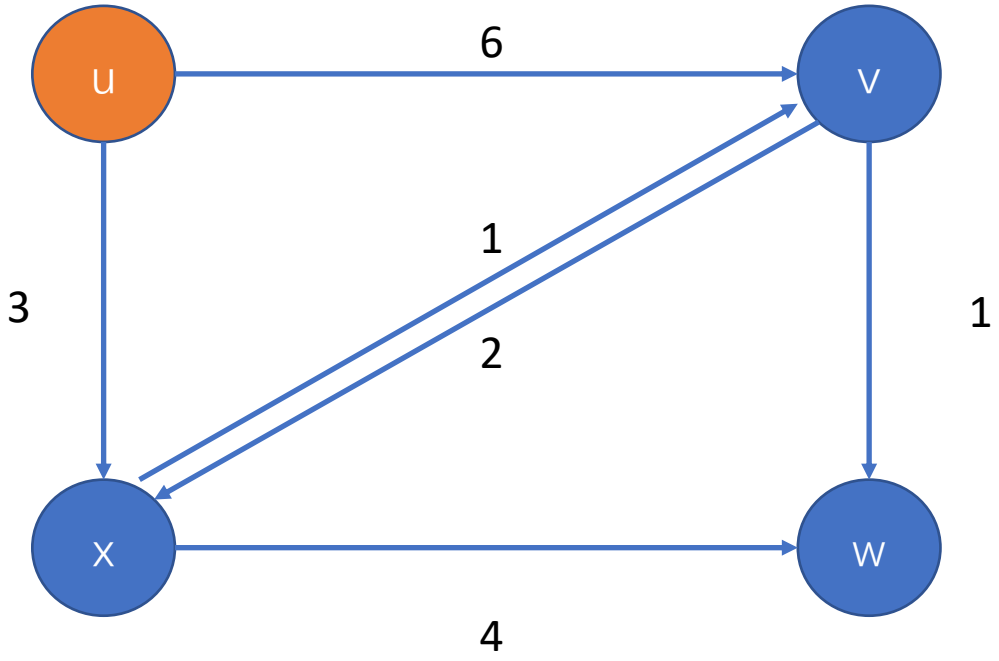
Problem 1

- Execute Dijkstra's algorithm on the following digraph, where u is the source vertex.
- You need to indicate only the following:
- (a) the order in which the vertices are removed from the priority queue
- (b) the final distance values d[] for each vertex
- (c) the different distance values d[] assigned to vertex w, as the algorithm executes



Solution

When running Dijkstra's algorithm with u as the source vertex.
(i) x is removed first
(ii) then v is removed
(iii) then w is removed



d(u) = 0

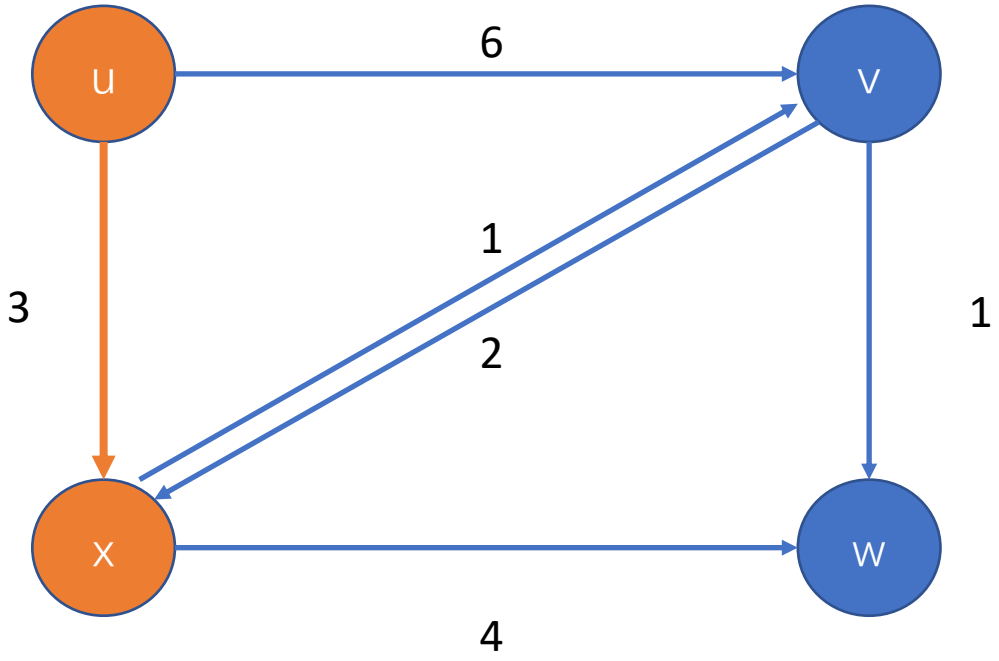
d(x) = 3

d(v) = 6

d(w) = ∞

Solution

When running Dijkstra's algorithm with u as the source vertex.
(i) x is removed first
(ii) then v is removed
(iii) then w is removed



d(u) = 0

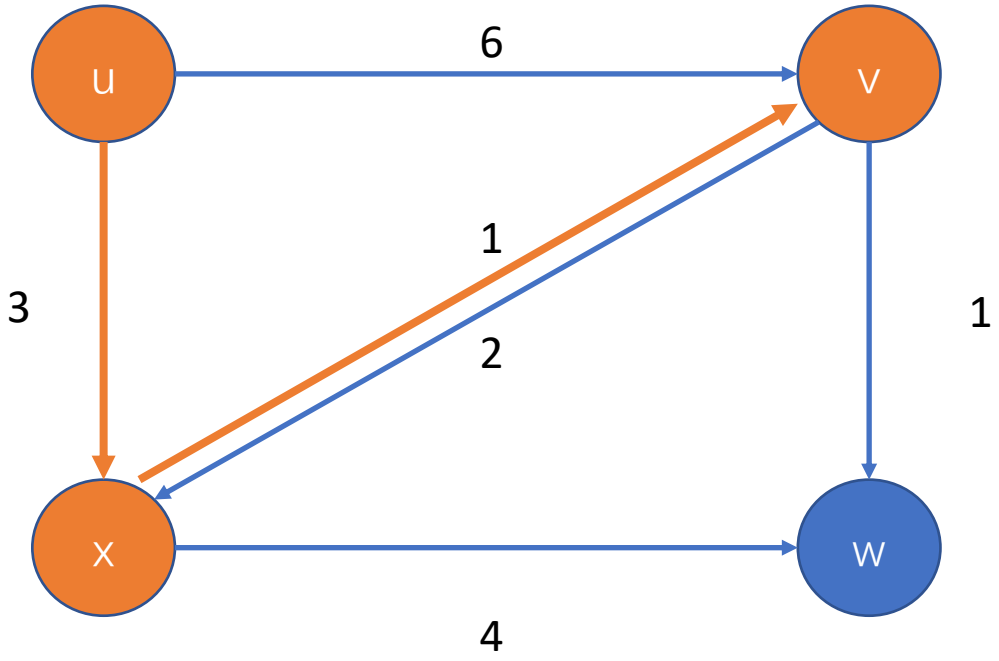
d(x) = 3

d(v) = 4

d(w) = 7

Solution

When running Dijkstra's algorithm with u as the source vertex.
(i) x is removed first
(ii) then v is removed
(iii) then w is removed



d(u) = 0

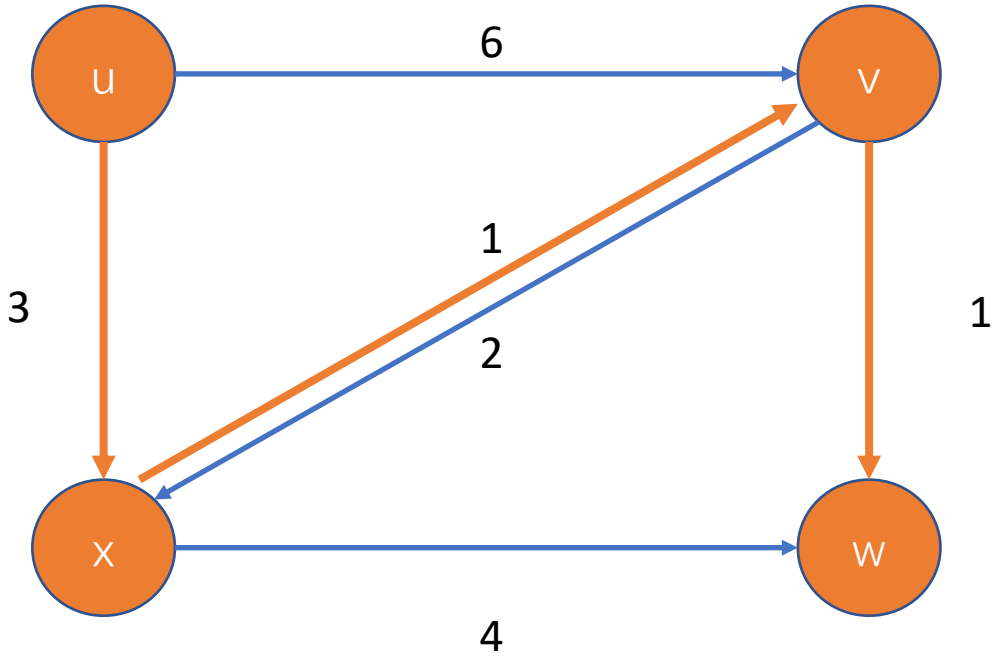
d(x) = 3

d(v) = 4

d(w) = 5

Solution

When running Dijkstra's algorithm with u as the source vertex.
(i) x is removed first
(ii) then v is removed
(iii) then w is removed



d(u) = 0

d(x) = 3

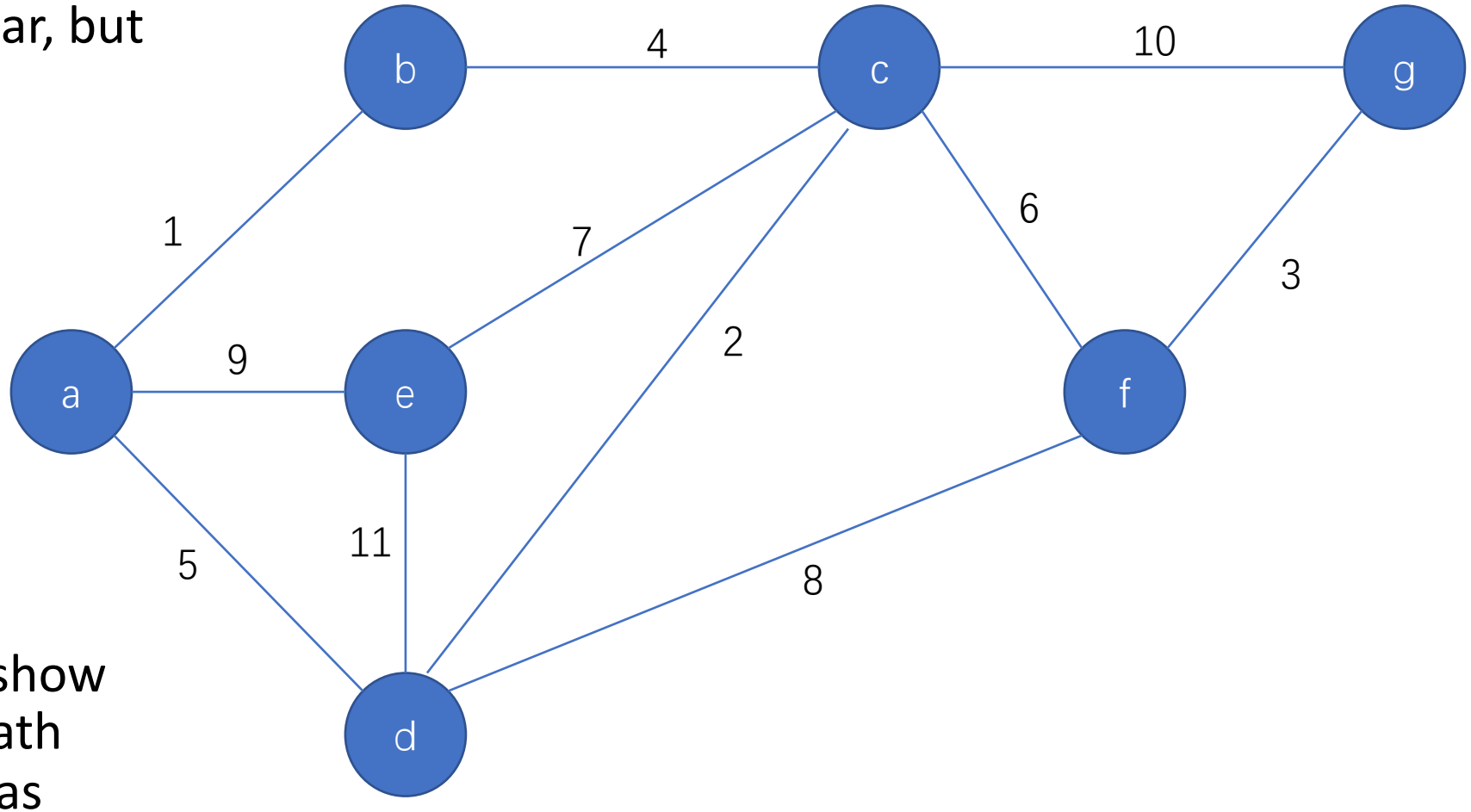
d(v) = 4

d(w) = 5

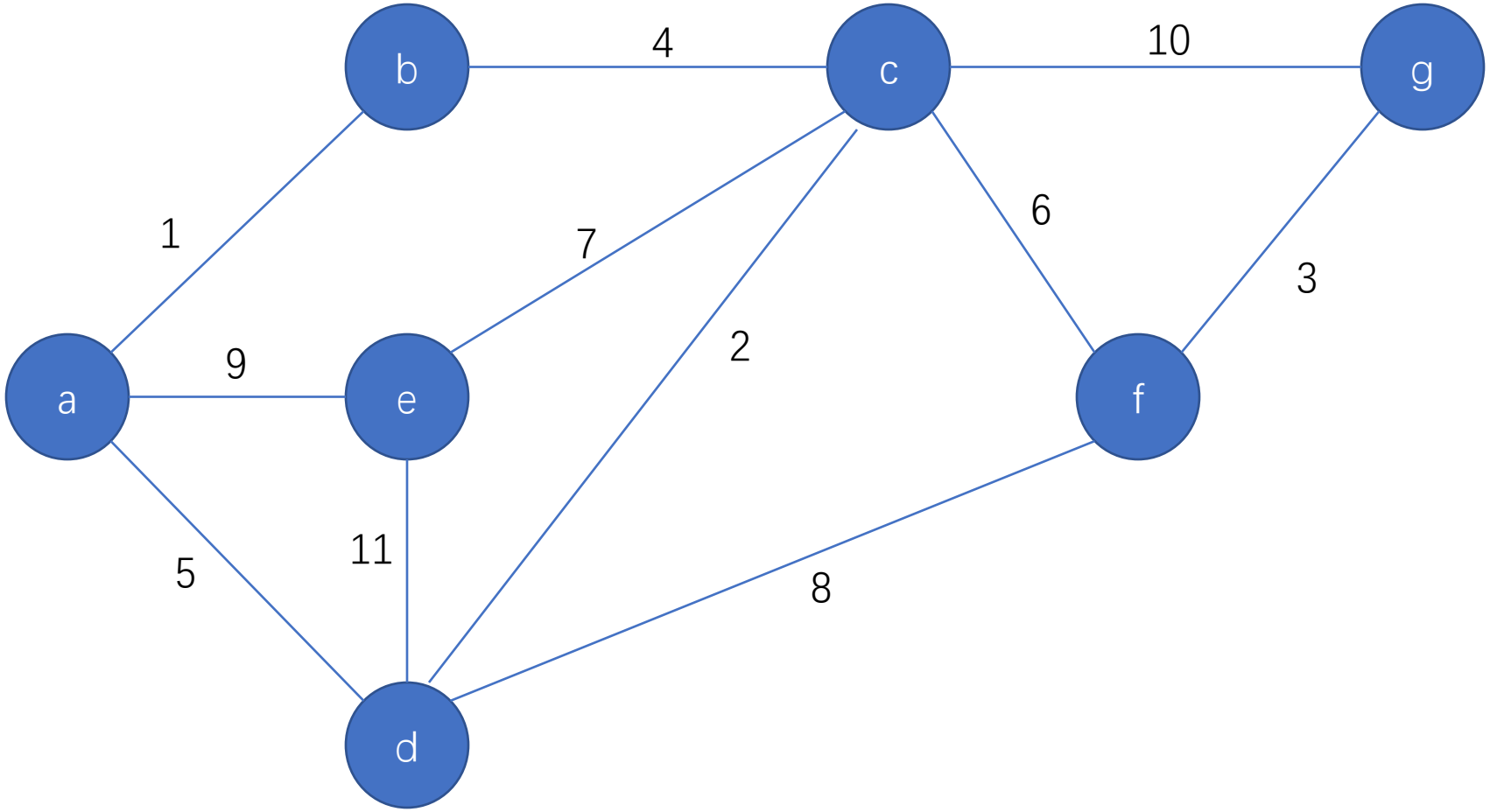
w had the values $\infty, 7, 5$

Problem 2

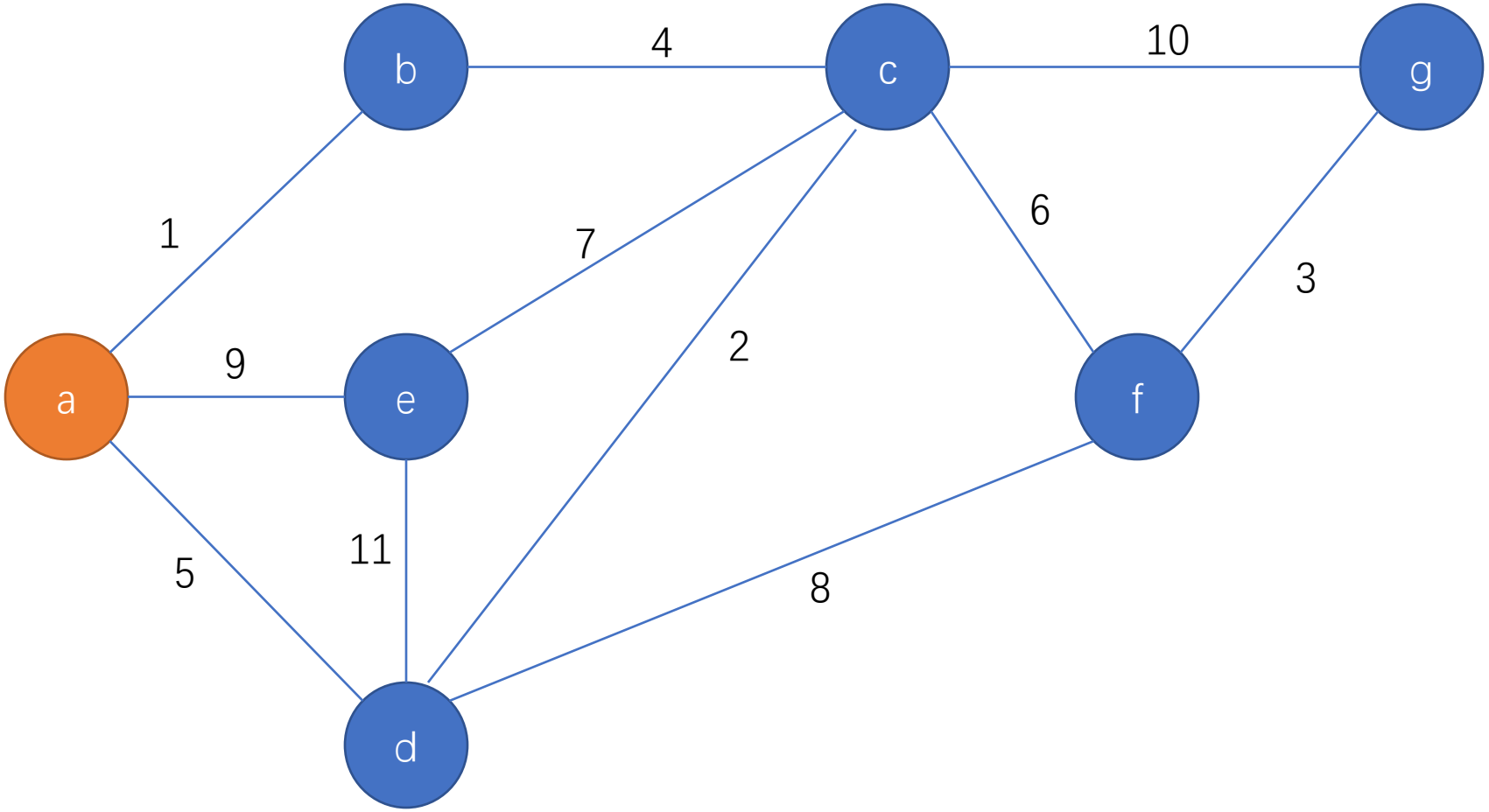
Prim's minimum spanning tree algorithm and Dijkstra's shortest path algorithm are very similar, but with crucial differences.



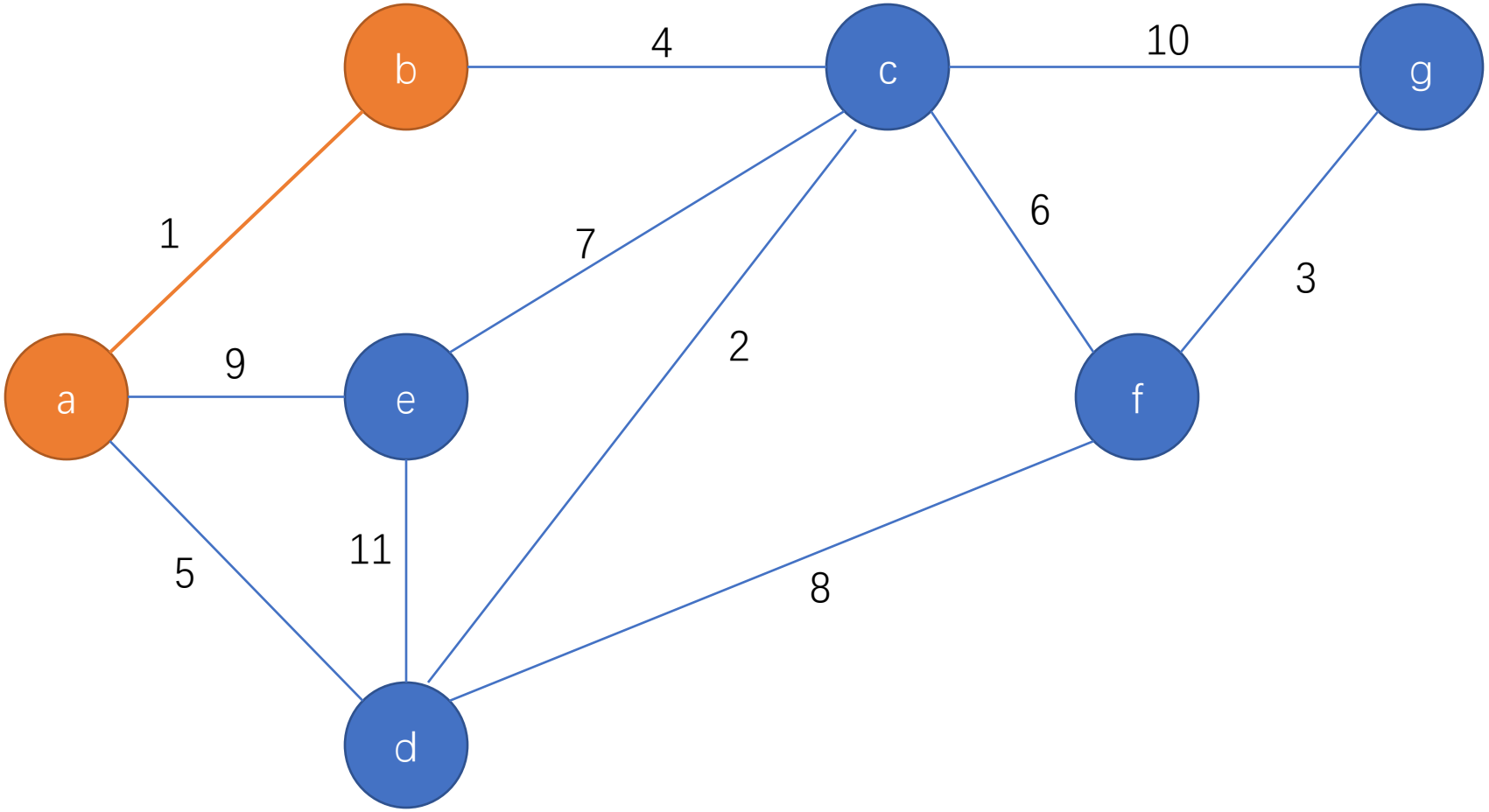
Run both algorithms on the graph (starting from a) and show the partial MST / shortest path tree after every new edge has been added.


b
4
c
10
g
1
7
6
3
a
9
e
2
f
11
5
8
d

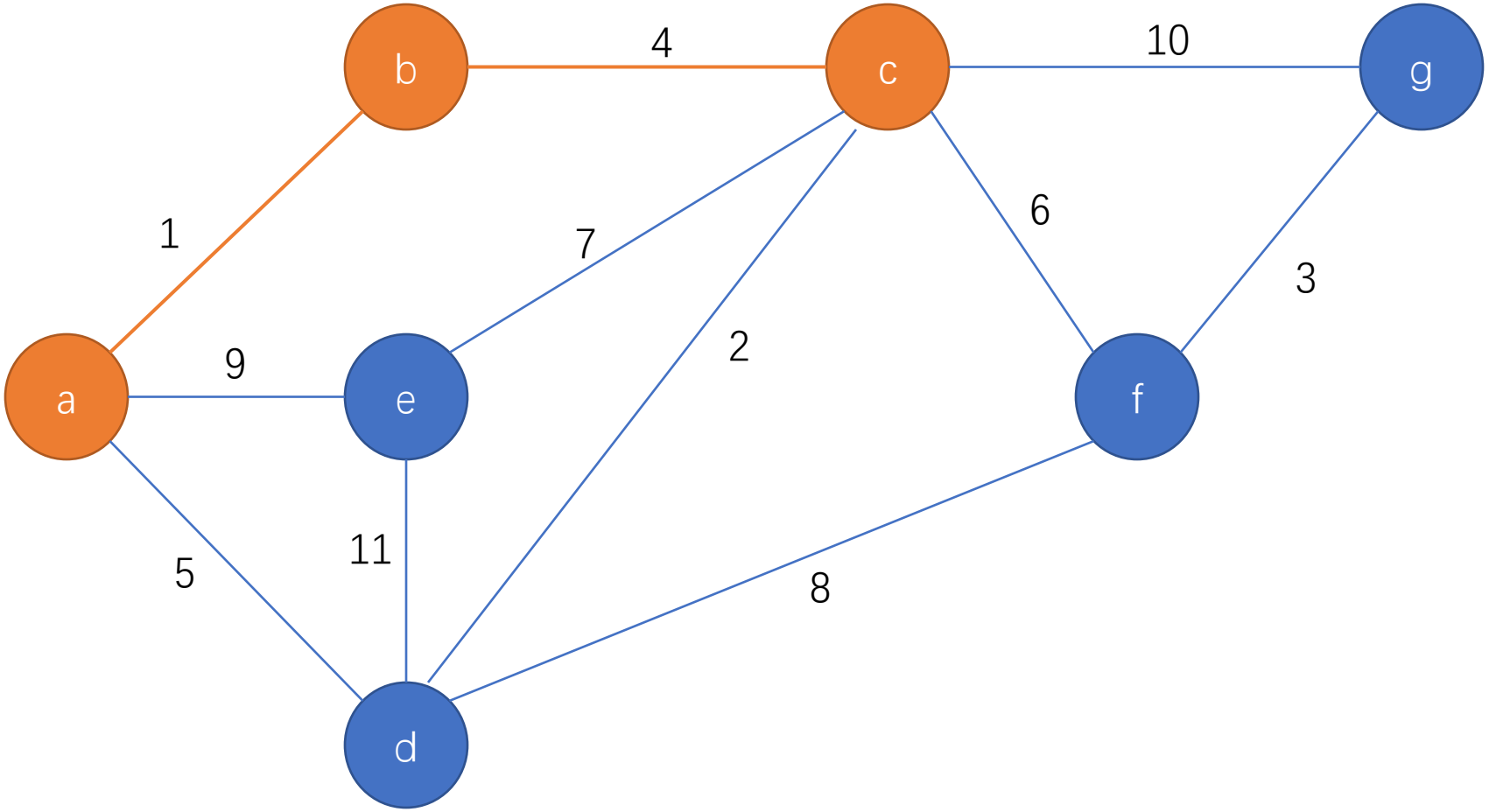
Solution: Dijkstra



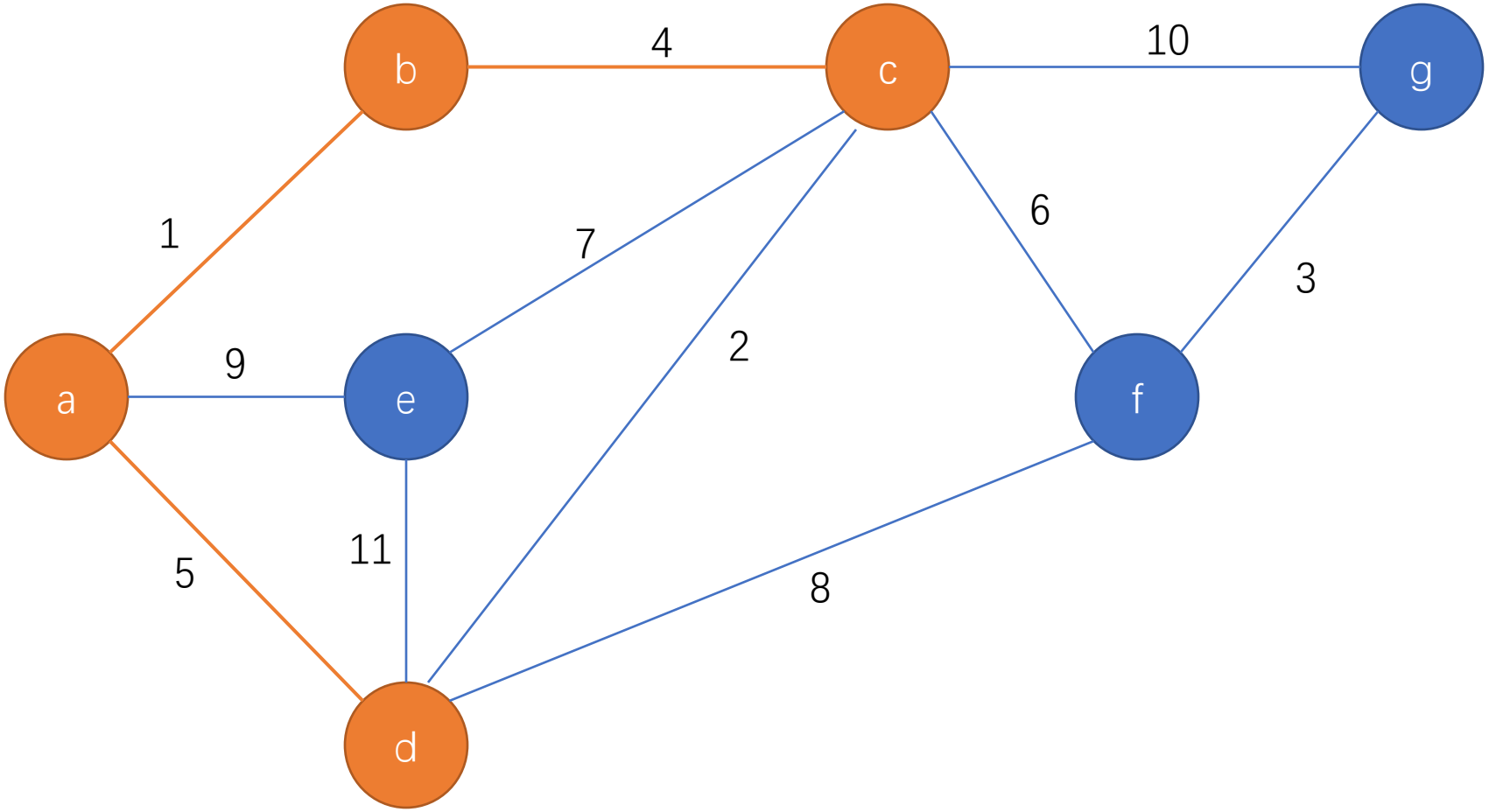
Solution: Dijkstra



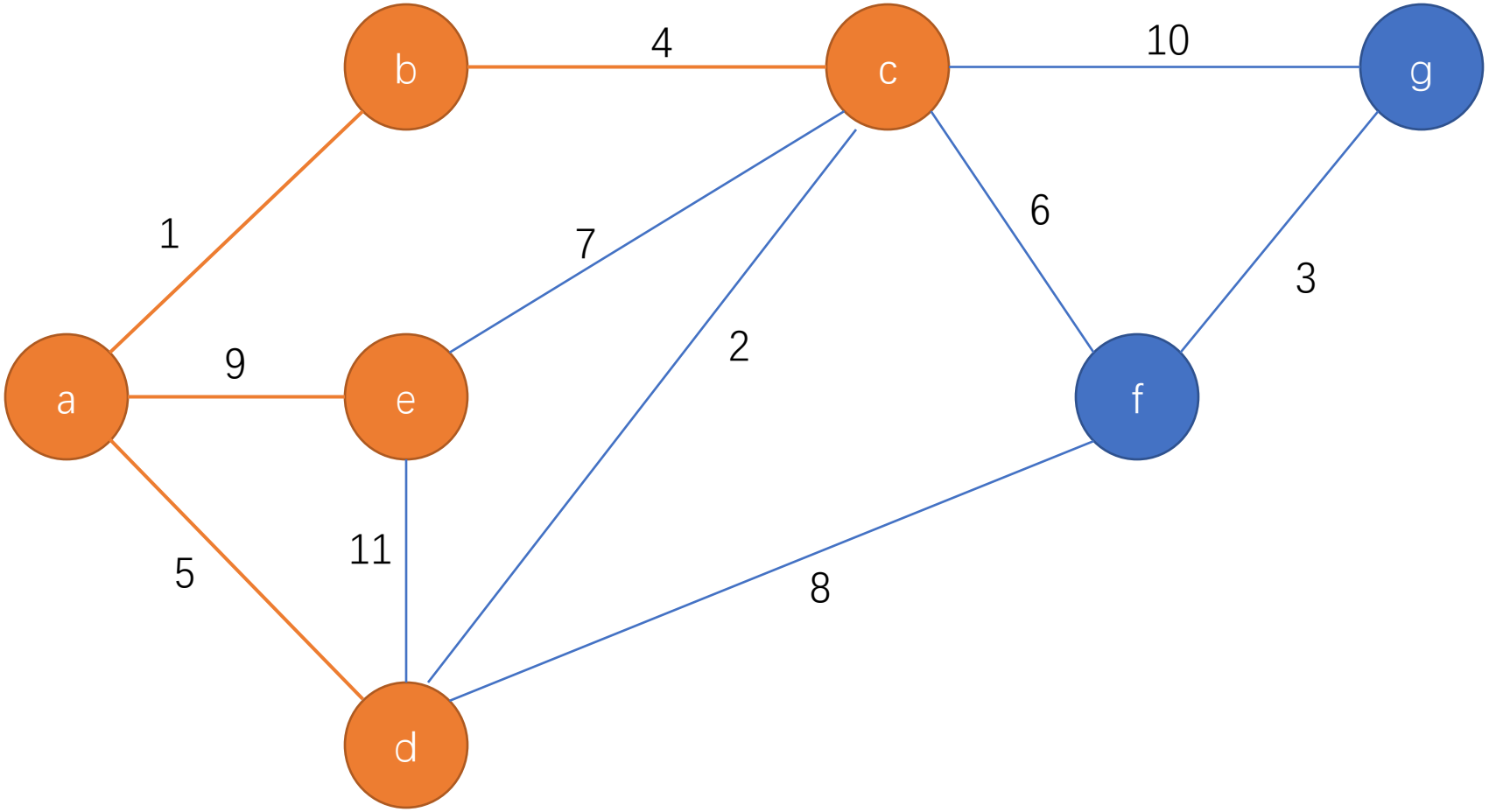
Solution: Dijkstra



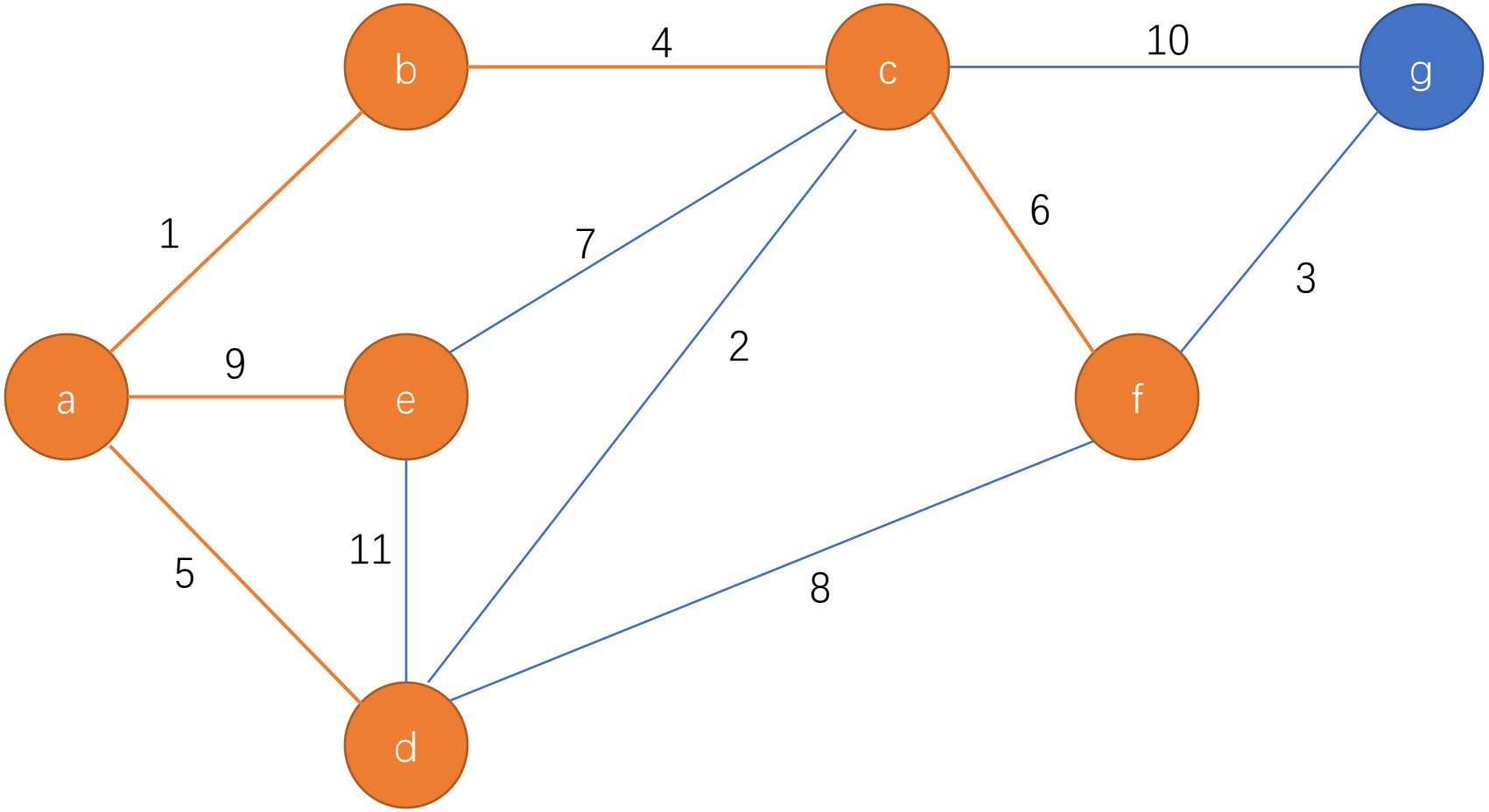
Solution: Dijkstra



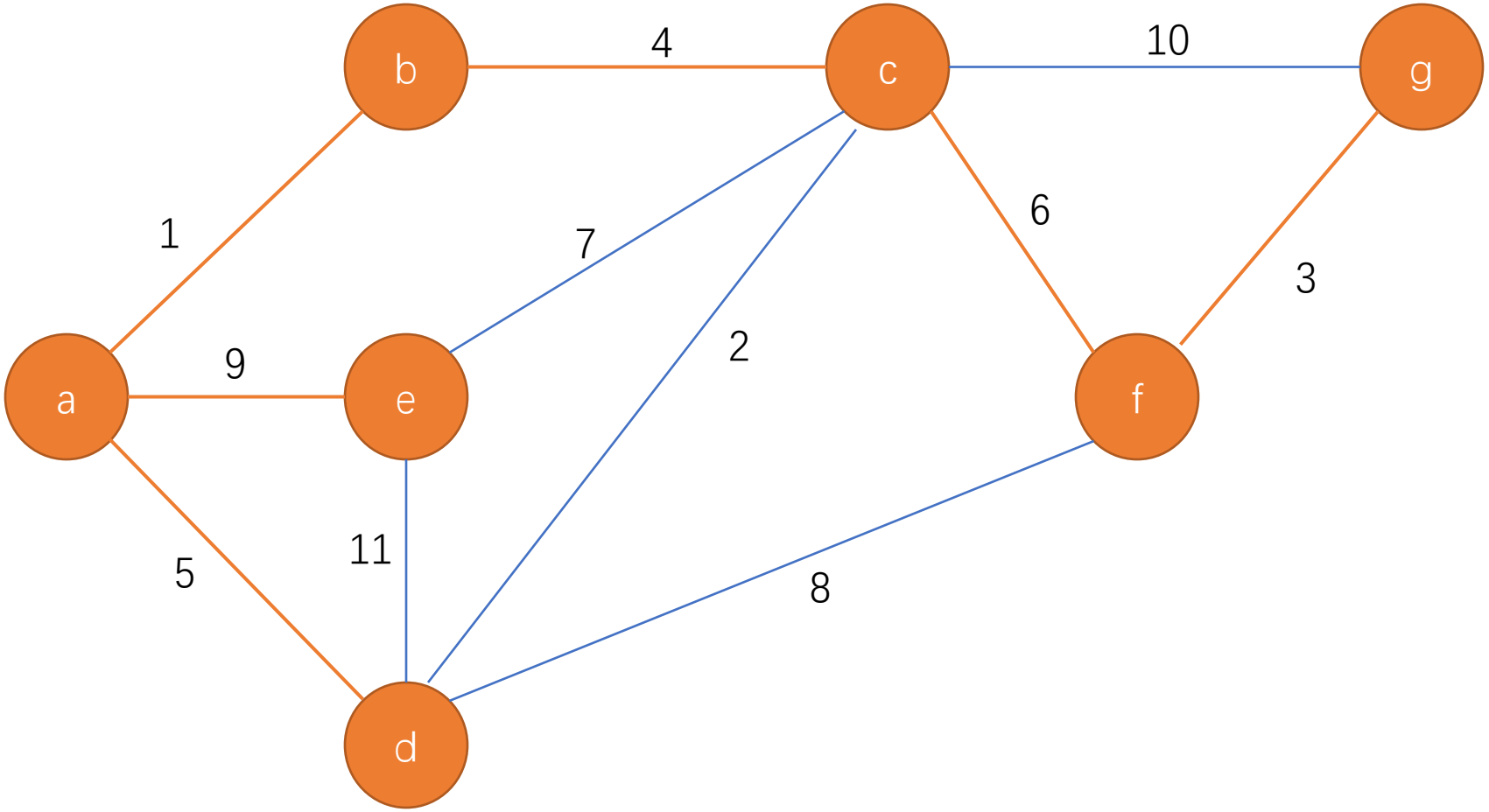
Solution: Dijkstra



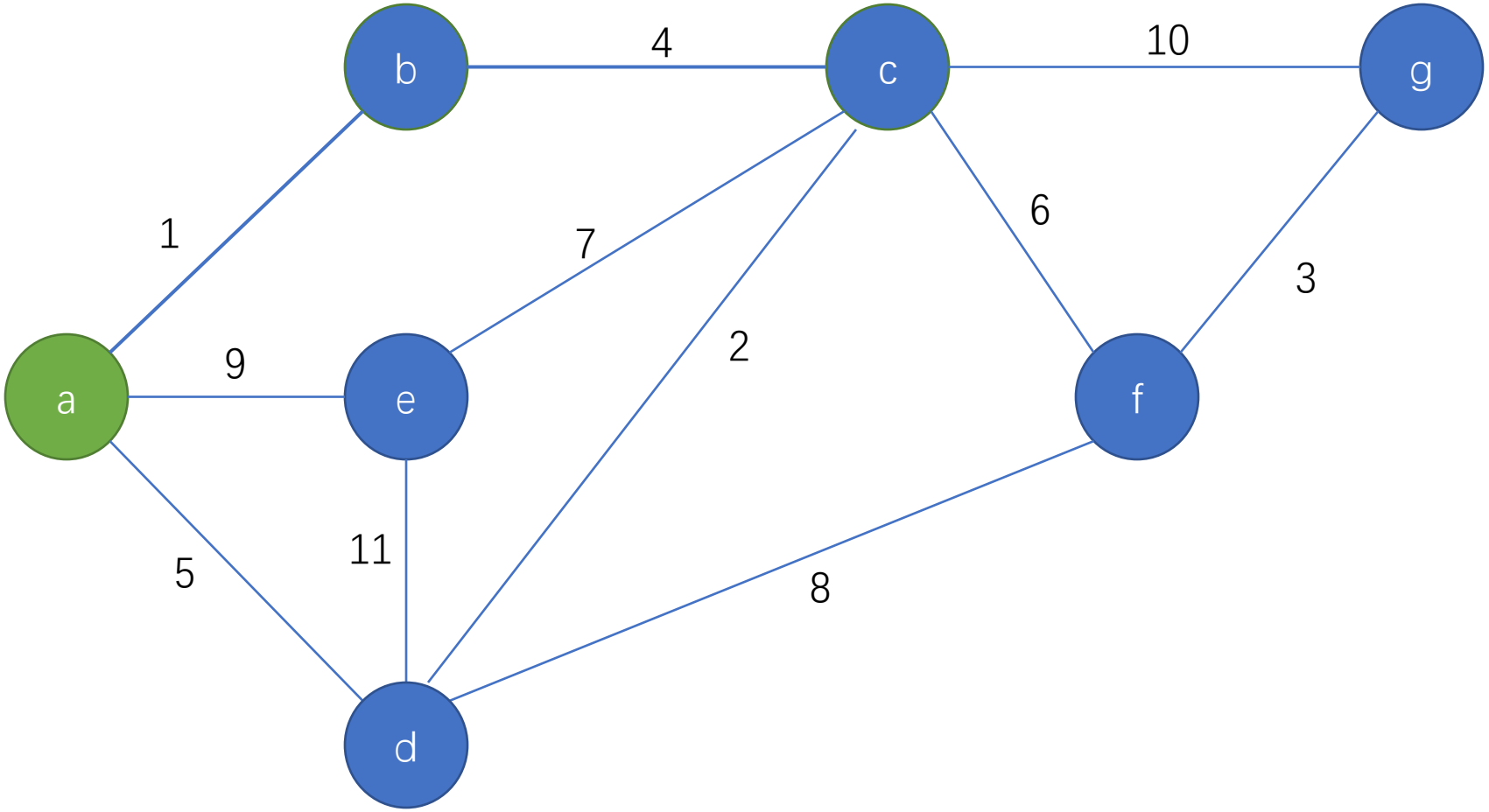
Solution: Dijkstra



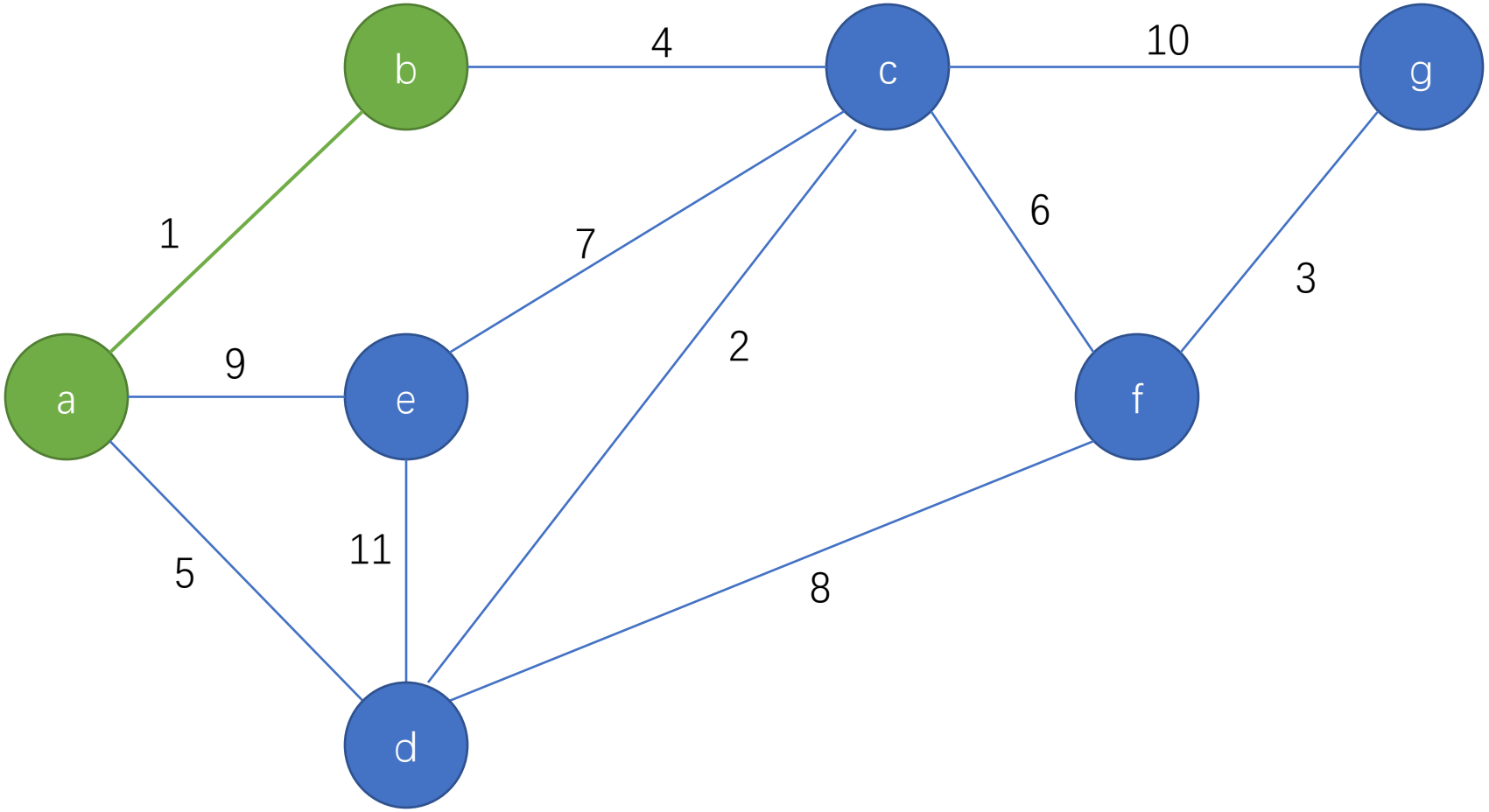
Solution: Dijkstra



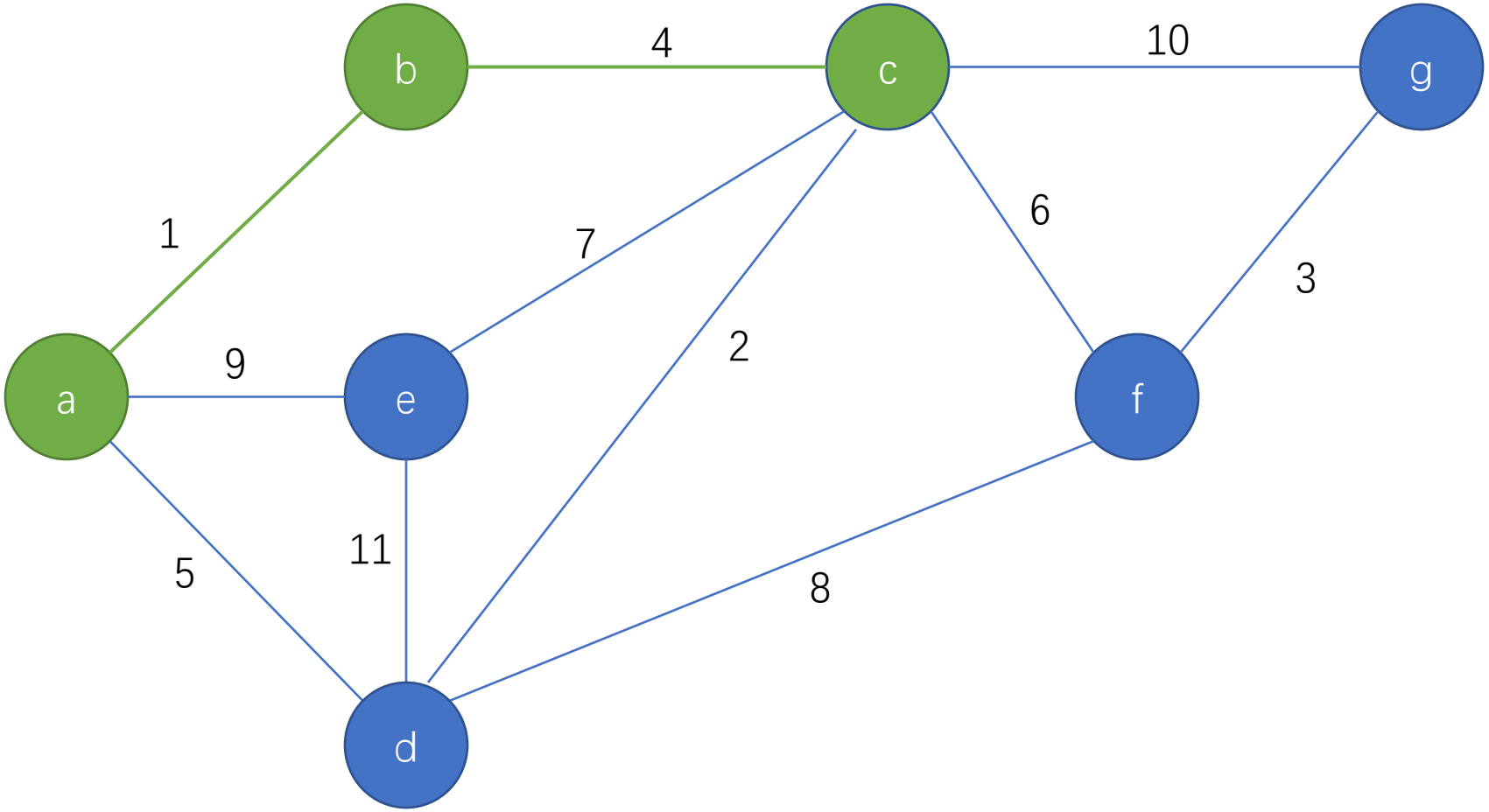
Solution:Prim



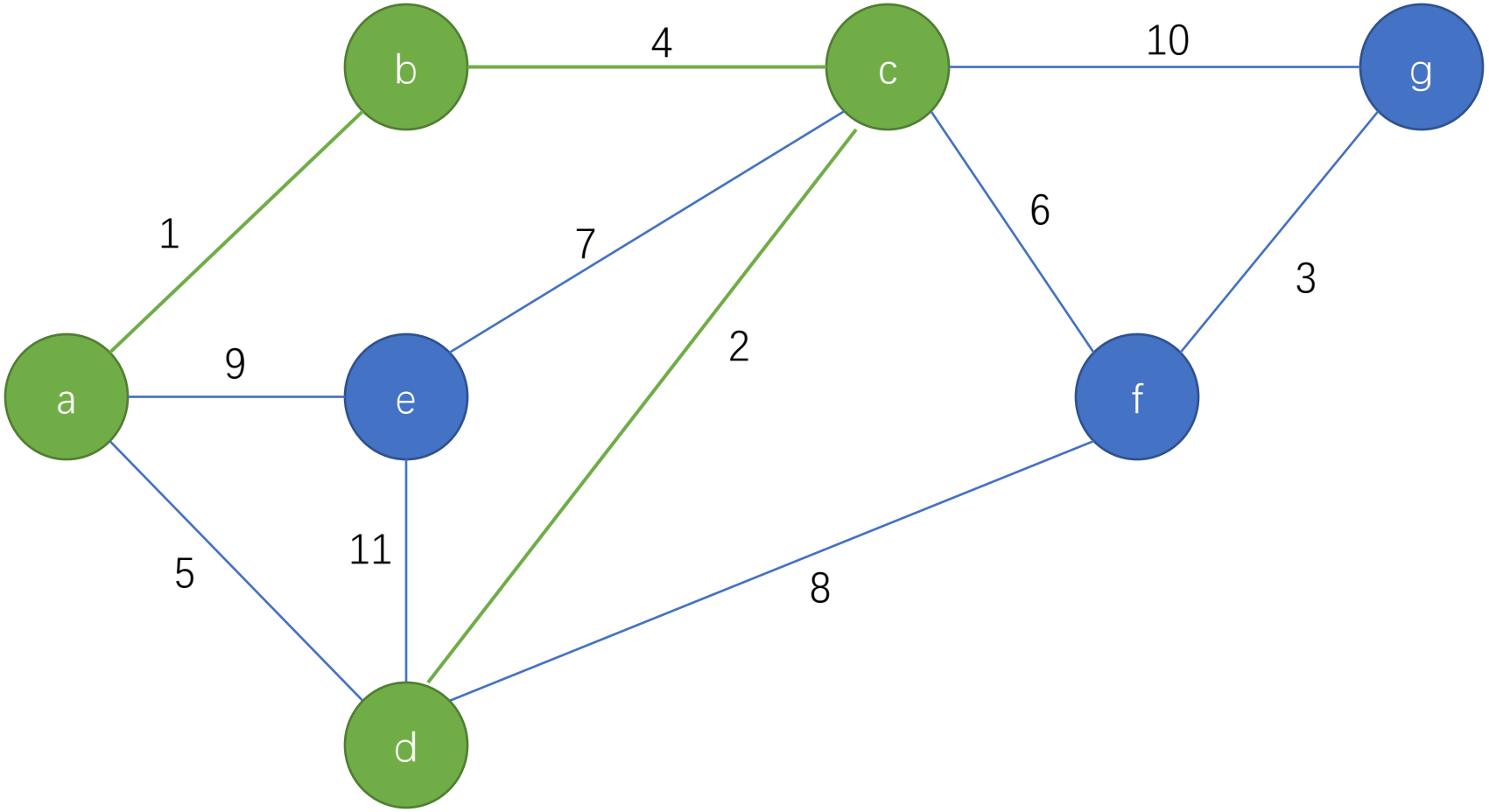
Solution:Prim



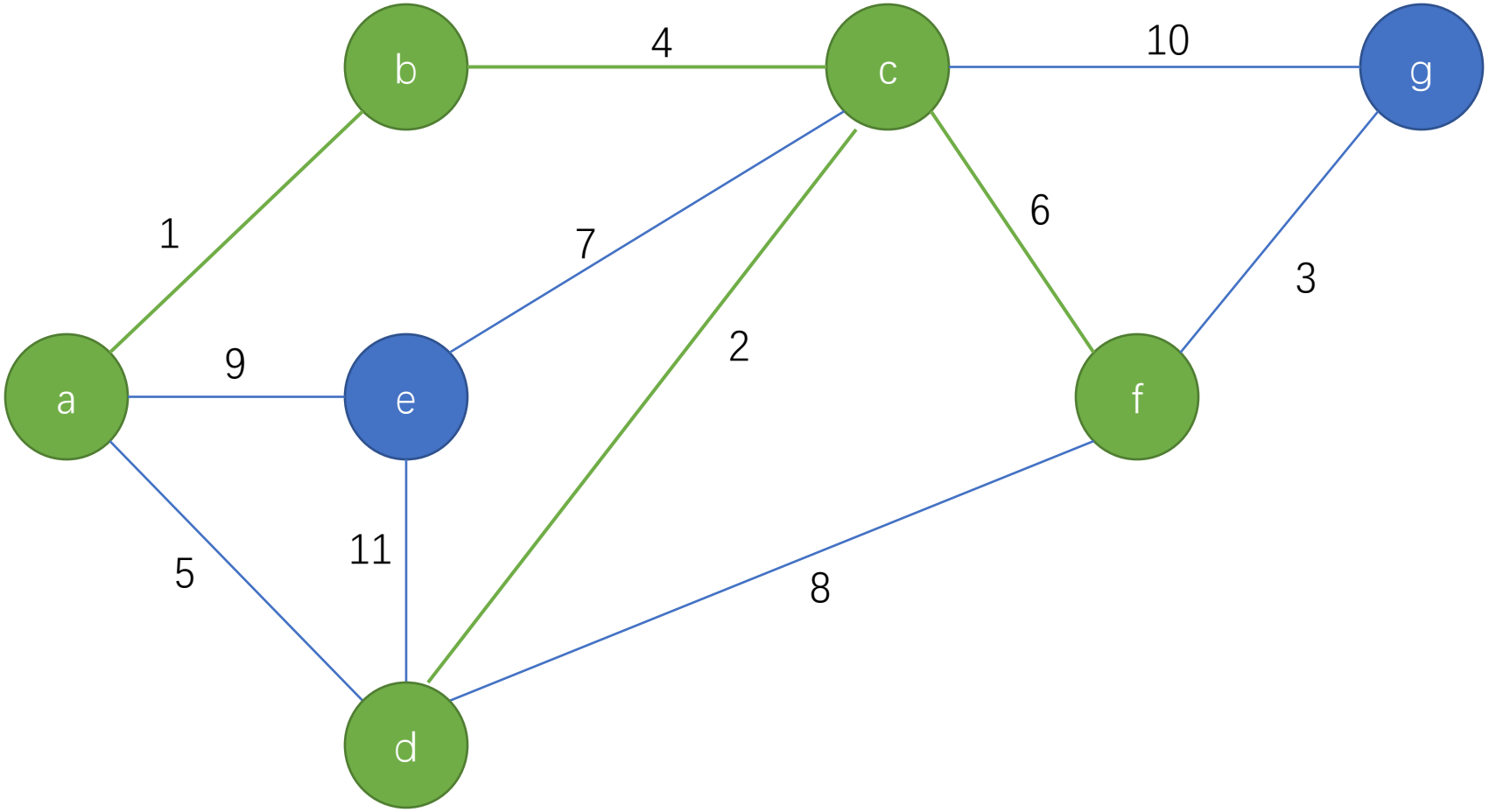
Solution:Prim



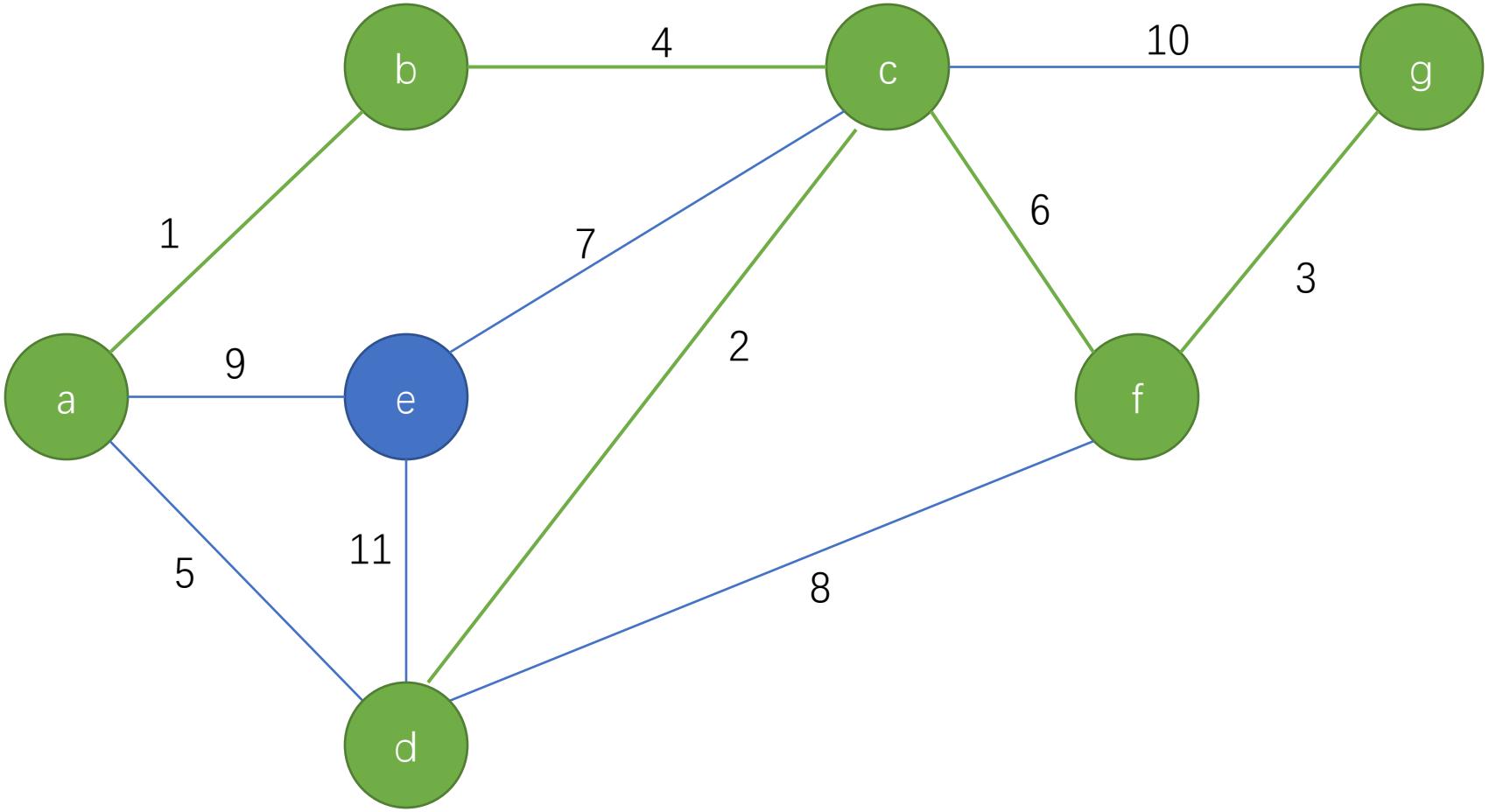
Solution:Prim



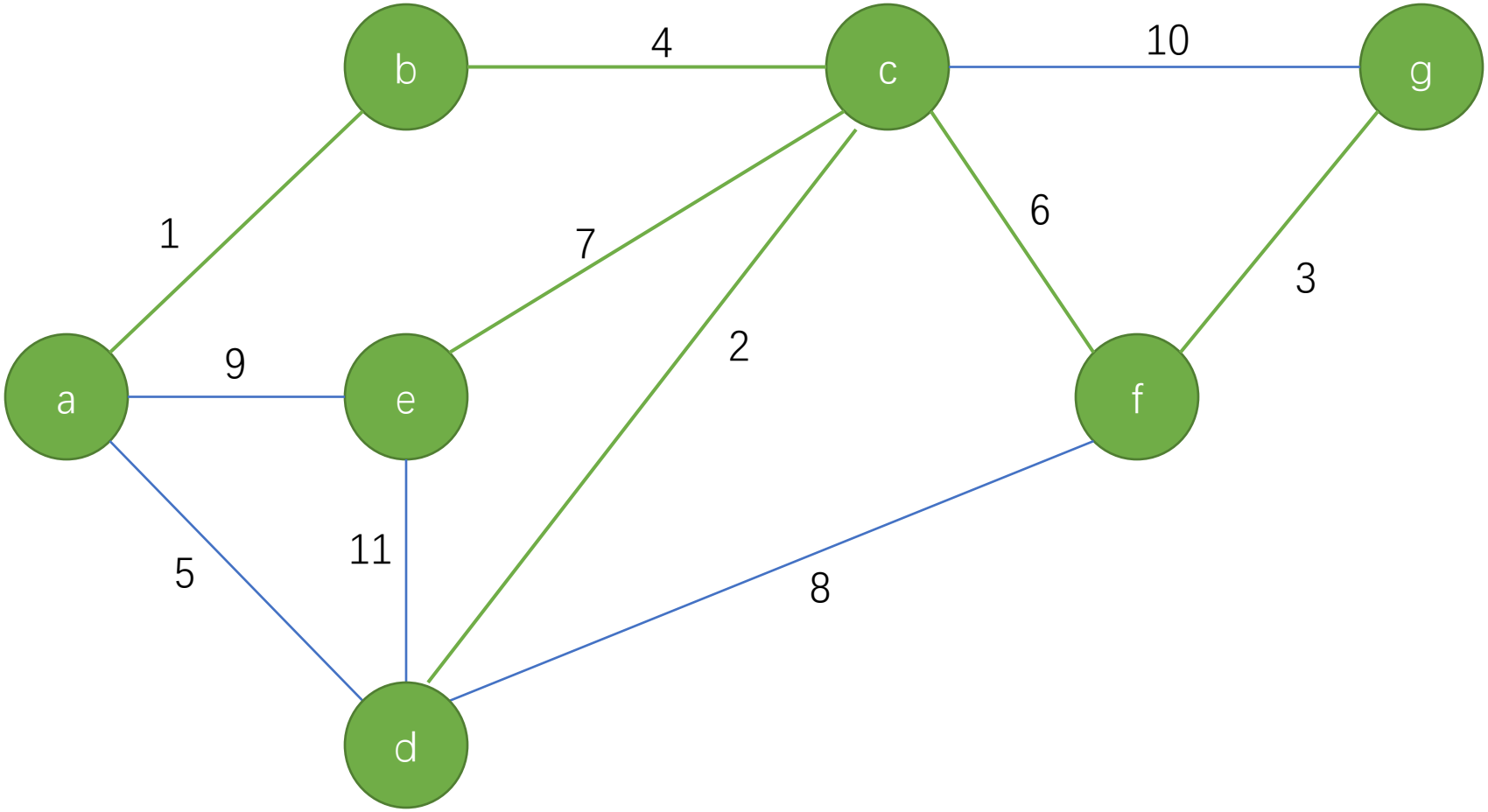
Solution:Prim



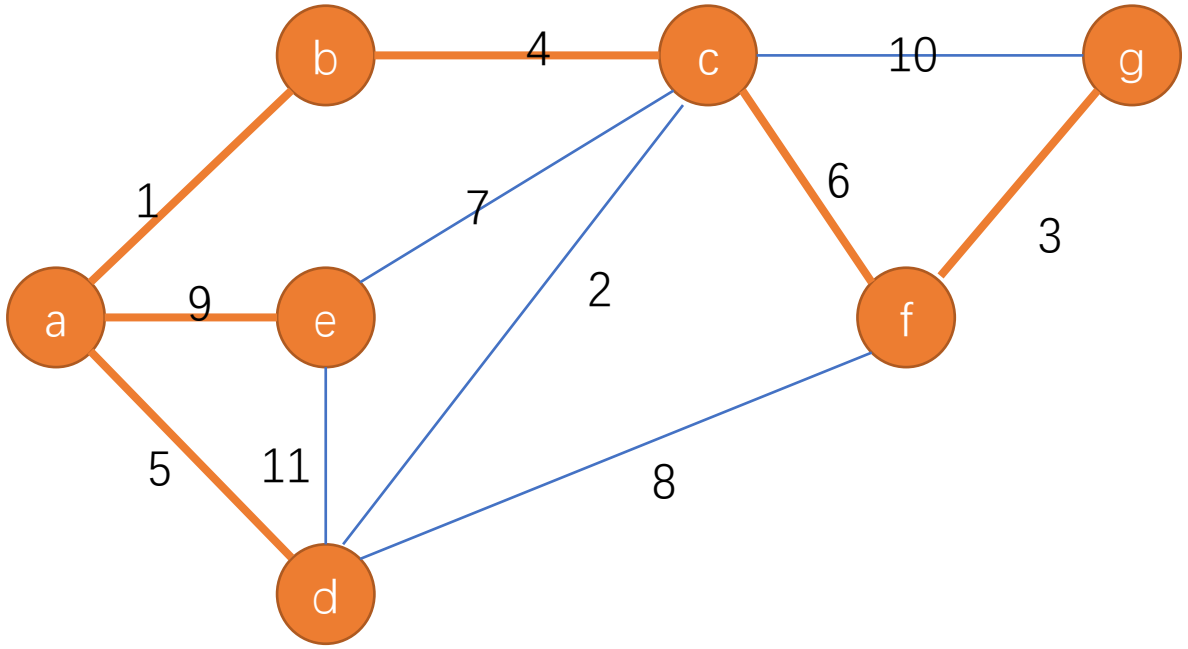
Solution:Prim



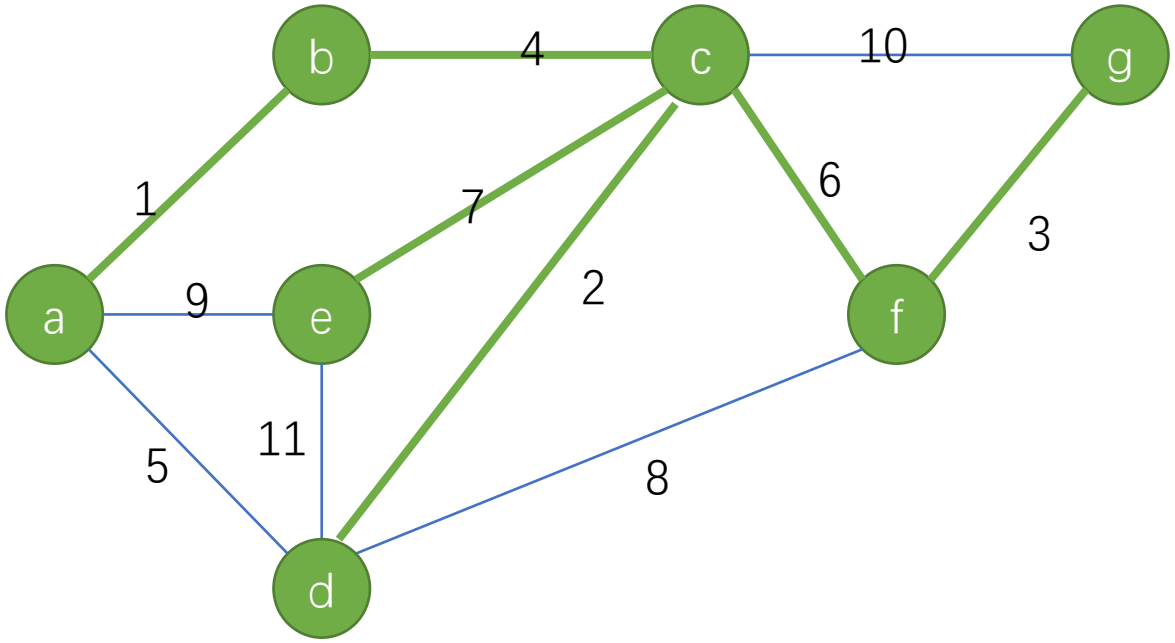
Solution:Prim



Solution



Shortest Path
(from Dijkstra)



MST
(from Prim)

Solution

Note:
Although the algorithms are very similar they output two different trees!

MST:
If the edges are all distinct, the MST is unique.
=> Prim's algorithm outputs the same (MST) Tree,
independent of the initial vertex.

Shortest Path:
The shortest path tree, is totally dependent upon its starting vertex since the tree paths are specifically the shortest paths from the starting vertex.
So, Dijkstra's algorithm can output a totally different tree for each starting vertex!

Problem 3

- Suppose that **instead of using a heap** to store the tentative vertex distances, Dijkstra's algorithm **just kept an array** in which it stored each vertex's tentative distance.
- It then finds the next vertex **by running through the entire array** and choosing the vertex with lowest tentative distance
- What would the algorithm's running time be?
- Is this better than our implementation for some graphs?

Solution

Let n be the number of vertices and m the number of edges.

- At each step, algorithm uses $O(n)$ time to scan through all of the remaining vertices.
  It also uses $O(m)$ time to update the tentative distances
  (since the cost of one update is $O(1)$ in changing an array value,
  rather than $O(\log n)$ for a decrease key).
- => The algorithm will use $O(n^2)$ time (independent of m):

- This is "better" than the $O((n+m)\log n)$ implementation we learned in class if the graph has $m = \Omega\left(\frac{n^2}{\log n}\right)$.
- For example, if $m = \Theta(n^2)$ then Dijkstra's original version runs in $O(n^2)$ time, while the version taught in class uses $O(n^2 \log n)$ time

Note: This is actually the "original" implementation of Dijkstra's algorithm by Dijkstra.
The use of priority queues to save time came later.

Extra Problem

- Suppose that a cable network of n sites is connected by duplex communication channels. Unfortunately, the communication channels are not perfect.

- The channel between sites u and v is known to fail with (given) probability $f(u, v)$

- The probabilities of failure for different channels are known to be mutually independent events.

- One of the *n* sites is the central station.

Your problem is to compute the most reliable paths from the central station to all other sites (i.e., the paths of lowest failure probabilities from the central station to all other sites).

- Design an algorithm for solving this problem, justify its correctness, and analyze its time and space complexities.

Solution

Let $f'(u,v) = 1 - f(u,v)$ which is the probability of edge (u,v) NOT failing.
The probability of a path $P = u_0, u_1, u_2, \ldots, u_t$ NOT failing
is then the product of the edge non-failures, i.e.,

$$Cost(P) = \prod_{i=0}^{t-1} f'(u_i, u_{i+1}).$$

Let u_0 be the central station.
Our goal is to find, for every v, a path from u_0 to v with MAXIMAL cost.

Finding maximum Cost(P) is the same as finding minimum $\frac{1}{Cost(P)}$

which is the same as finding the path with minimum $\log \frac{1}{Cost(P)}$.

Solution

Now set

$$w(u,v) = -\log f'(u,v) = \log \frac{1}{f'(u,v)}$$

and recall that the log of the product of values is the sum of the logarithms of the values so

$$\log \frac{1}{Cost(P)} = \sum_{i=0}^{t-1} \log \frac{1}{f'(u_i,u_{i+1})} = \sum_{i=0}^{t-1} w(u_i,u_{i+1})$$

Putting all the pieces together we have just shown that

finding a max-cost path from u to v

is exactly the same as

finding a min-distance path from u to v with edge distances $w(u,v)$.

=> we can use Dijkstra's single source shortest path algorithm to solve the problem in $O(|E|\log|V|)$ time.

Solution

- Final Note: One issue that we did not explicitly discuss is that Dijkstra's algorithm only works for edges that have non-negative weights.

- This is not a problem, though, because the $f'(u, v)$ are all probabilities between 0 and 1. Thus, the $w(u, v)$ are all non-negative and we can apply Dijkstra's algorithm.

- This transformation via logs of
  a max-weight path problem where path-weight is **product** of edge weights
  to
  a shortest-path problem, where path-length is **sum** of edge lengths,
  is a very well known and commonly used technique.