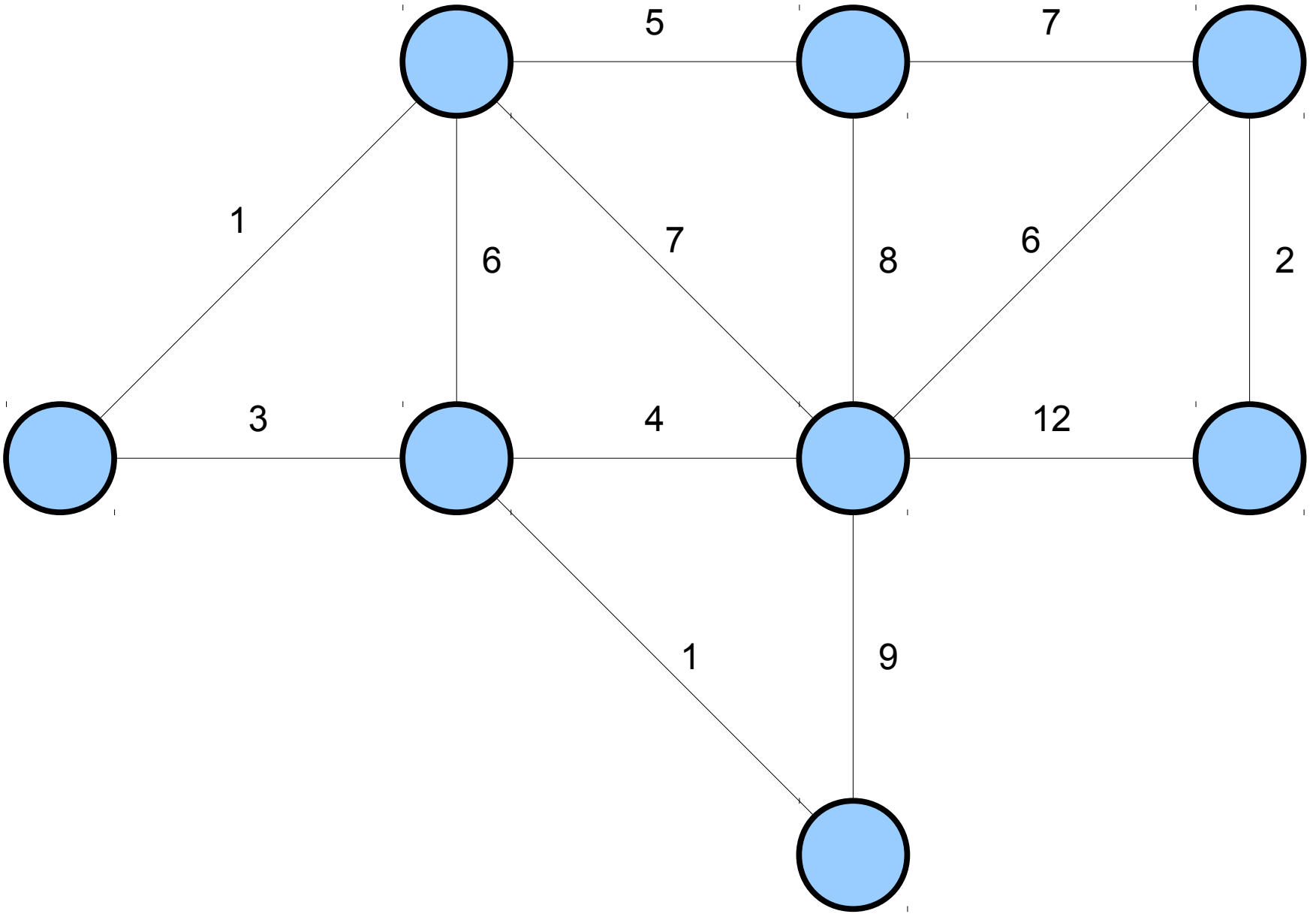
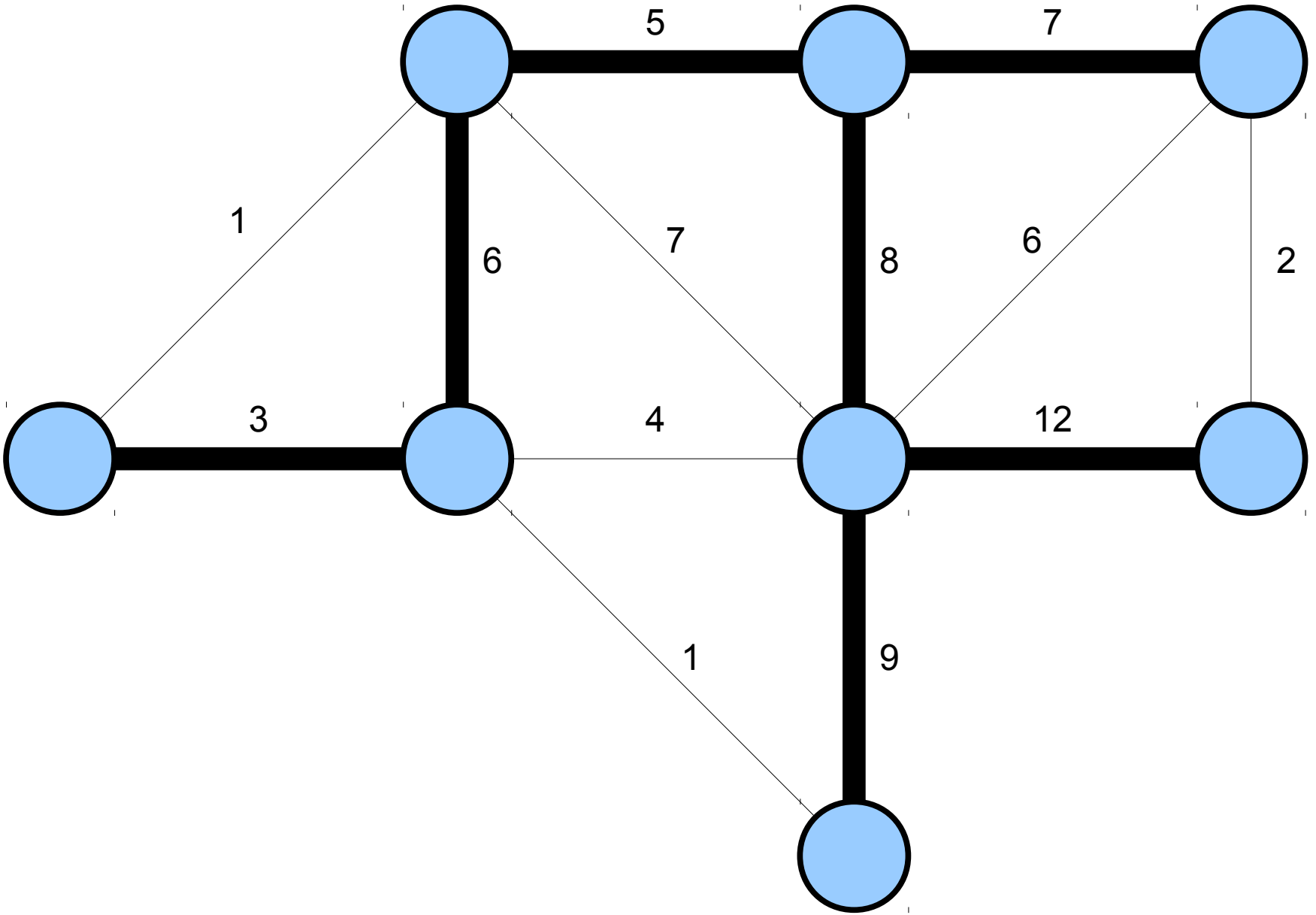
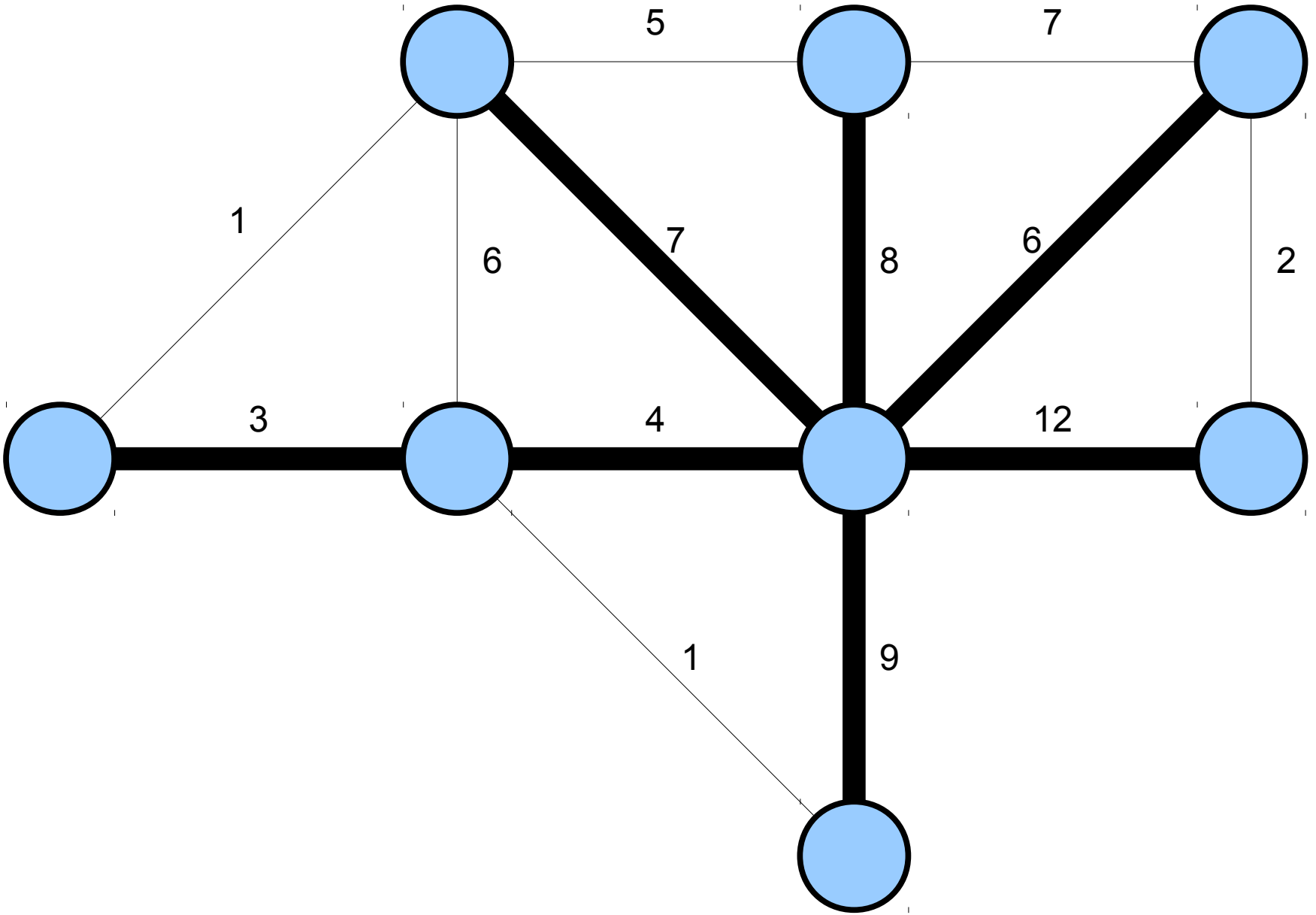
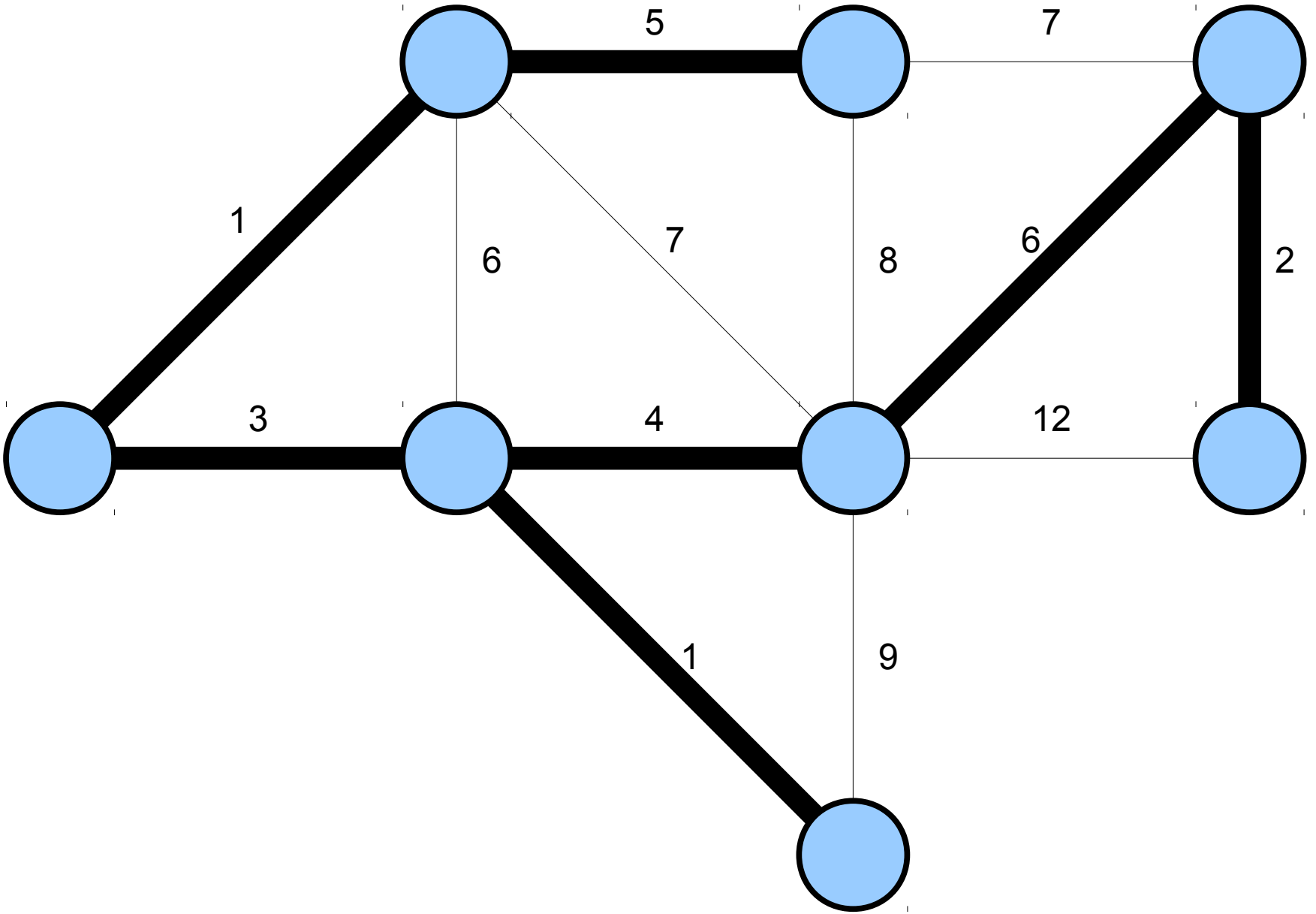
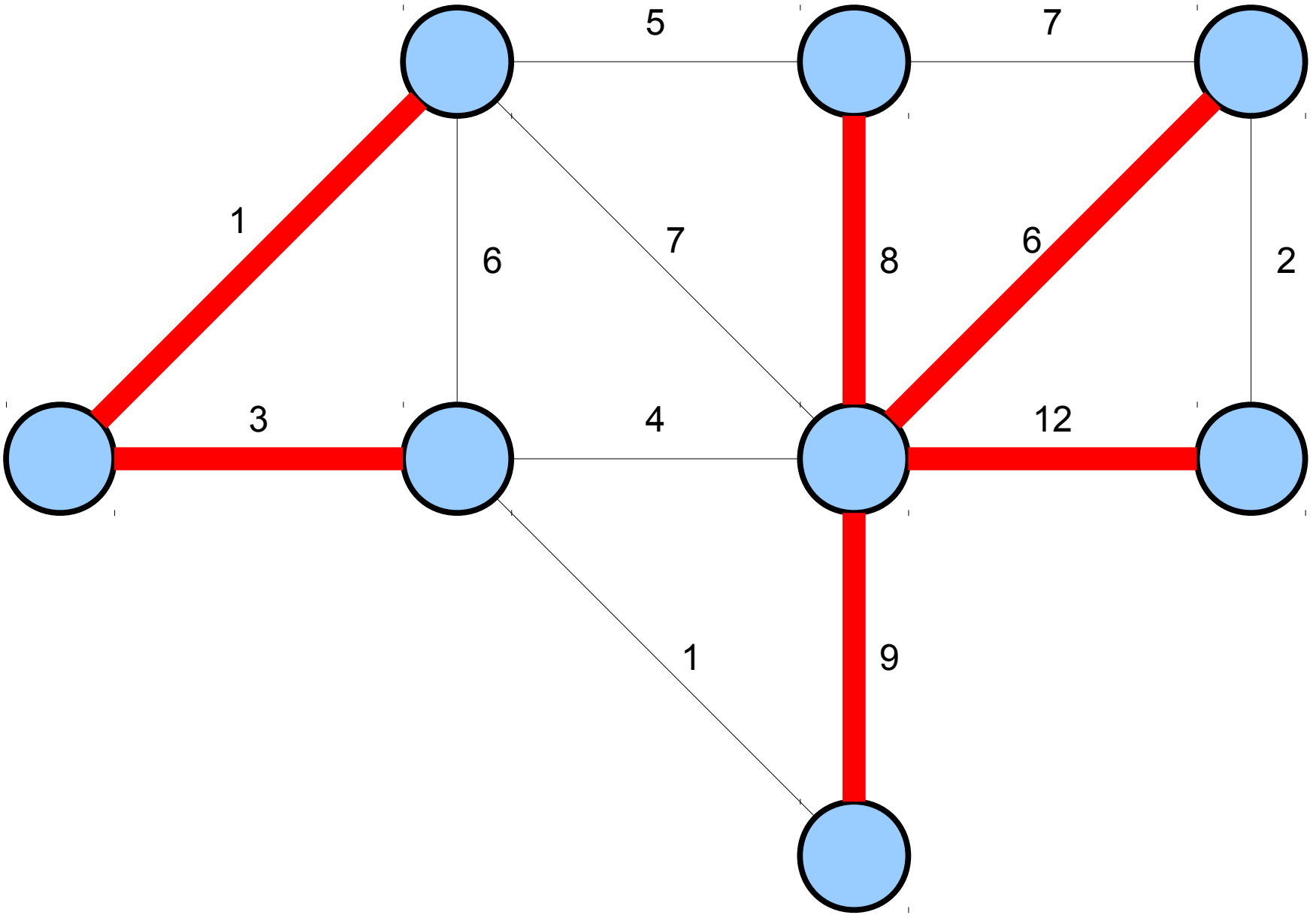
Minimum Spanning Trees

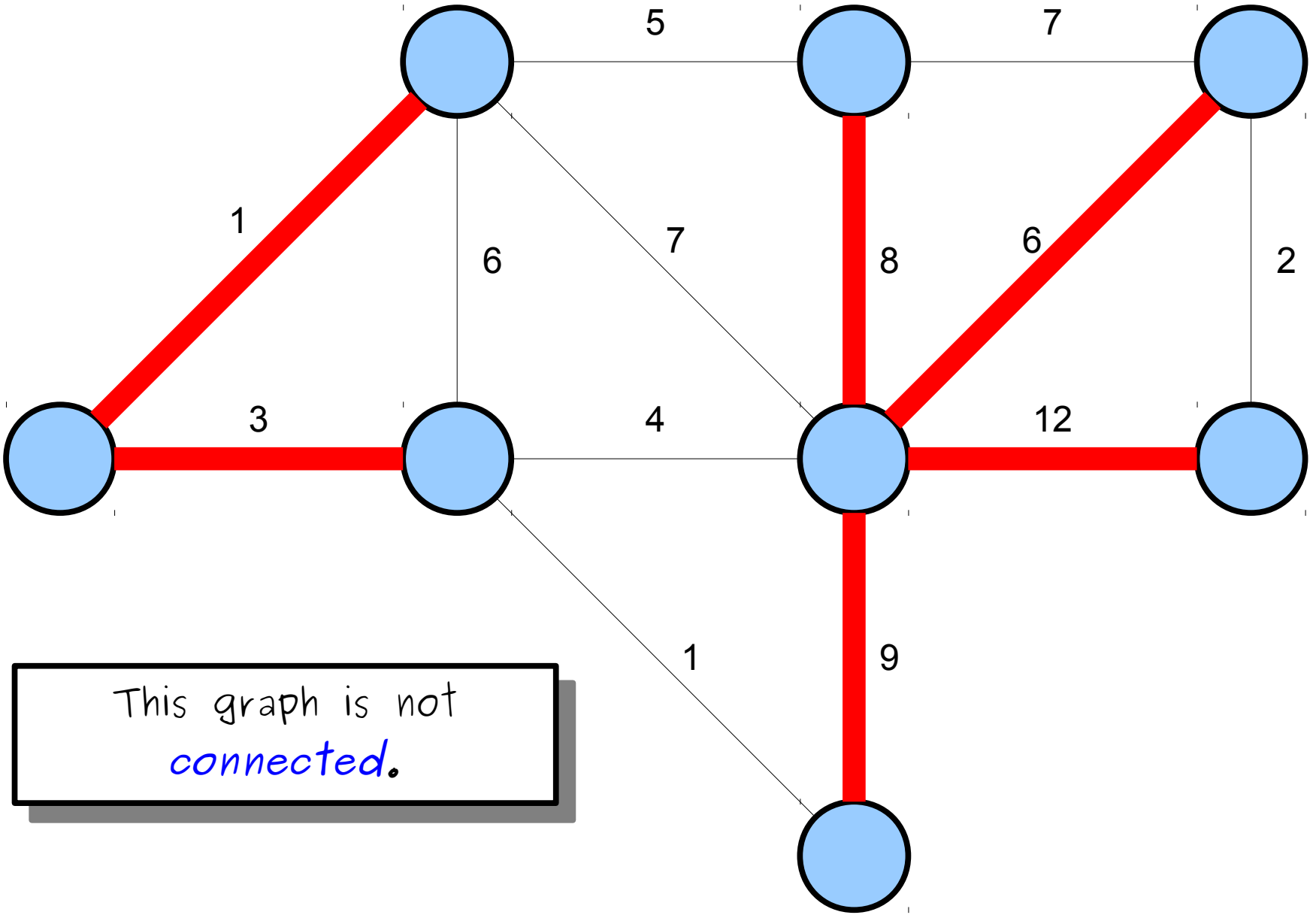

5
7
1
6
7
8
6
2
3
4
12
1
9

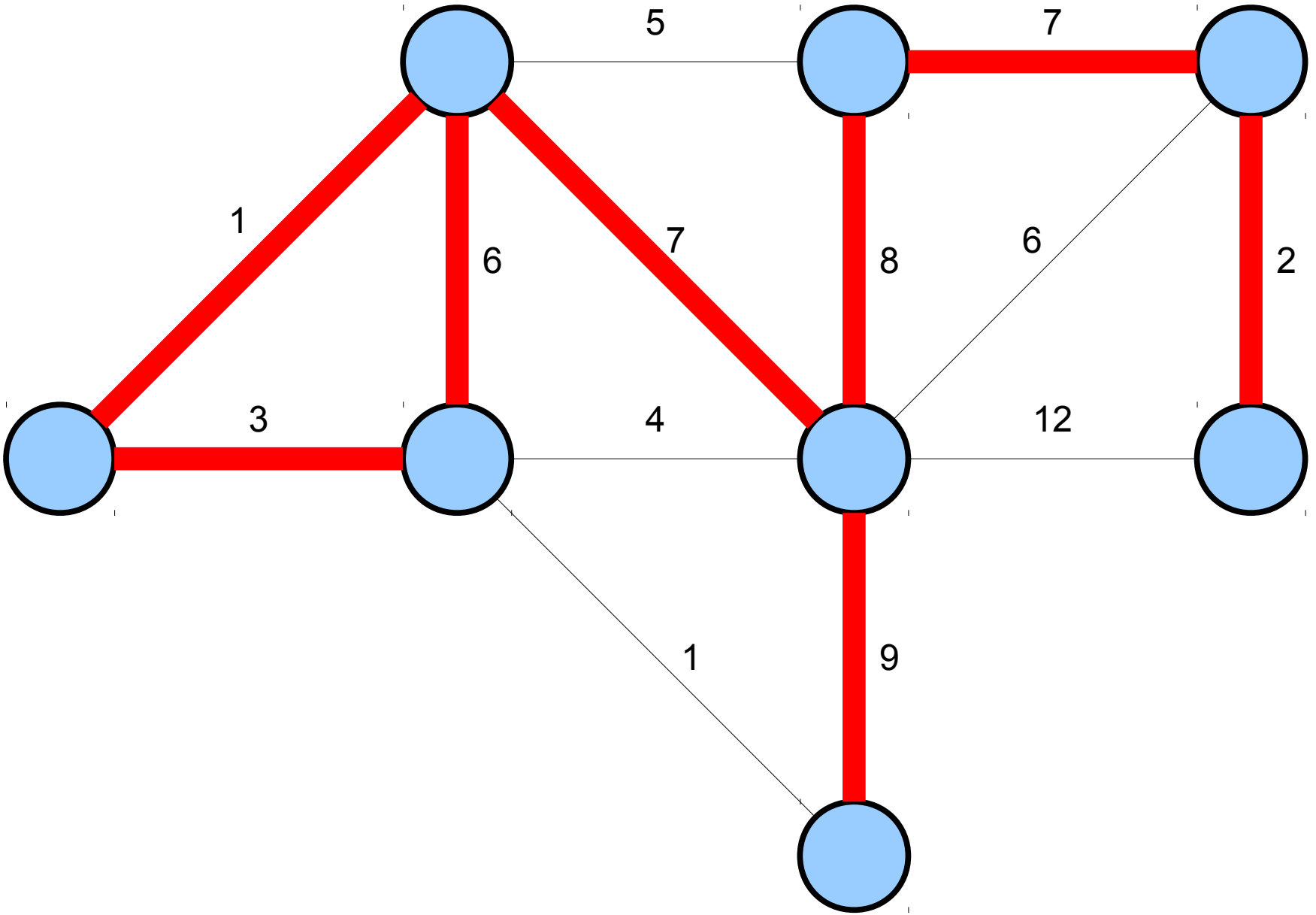

5
7
1
6
7
8
6
2
3
4
12
1
9

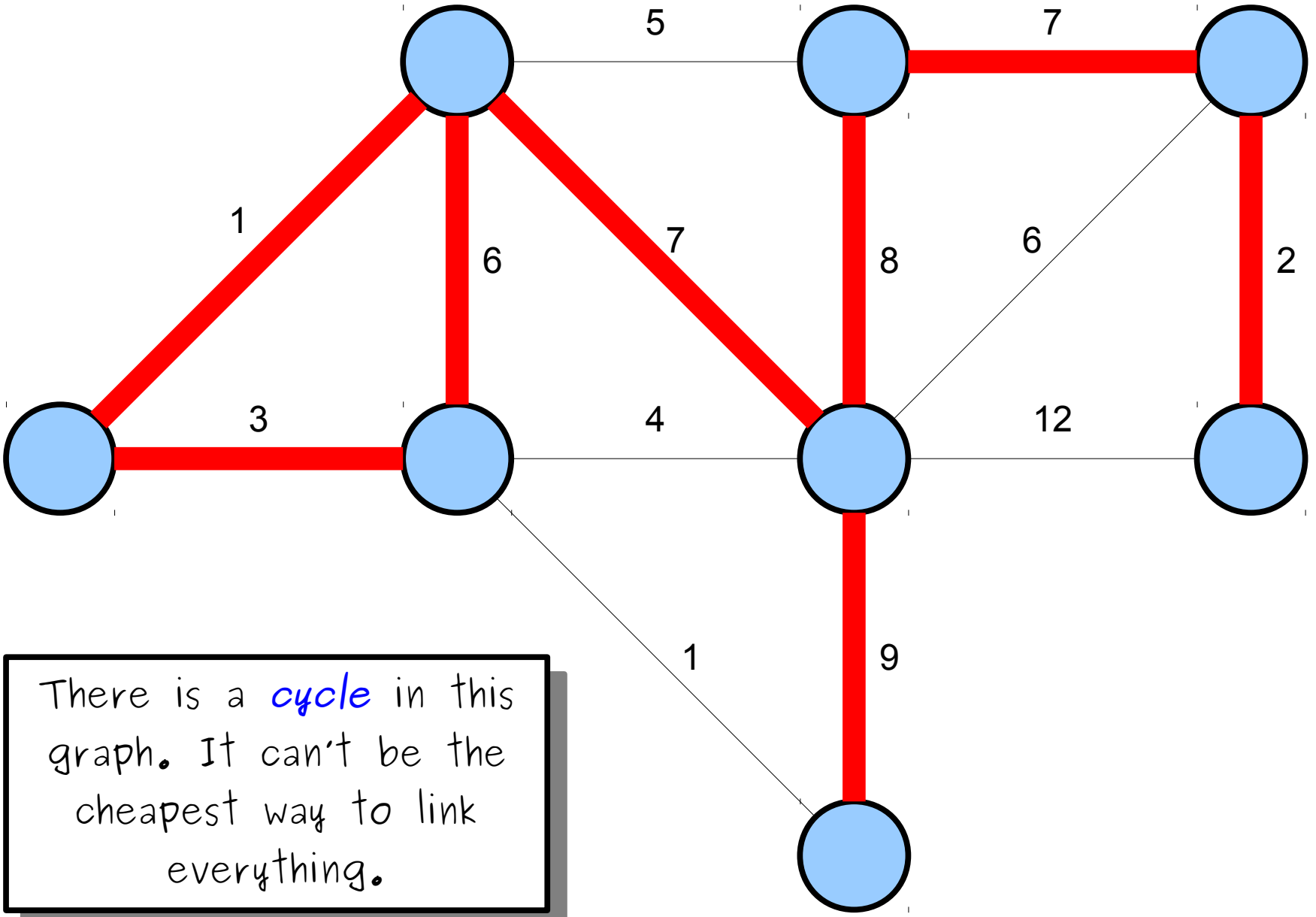

5
7
1
6
7
8
6
2
3
4
12
1
9


5
7
1
6
7
8
6
2
3
4
12
1
9


5
7
1
6
7
8
6
2
3
4
12
1
9

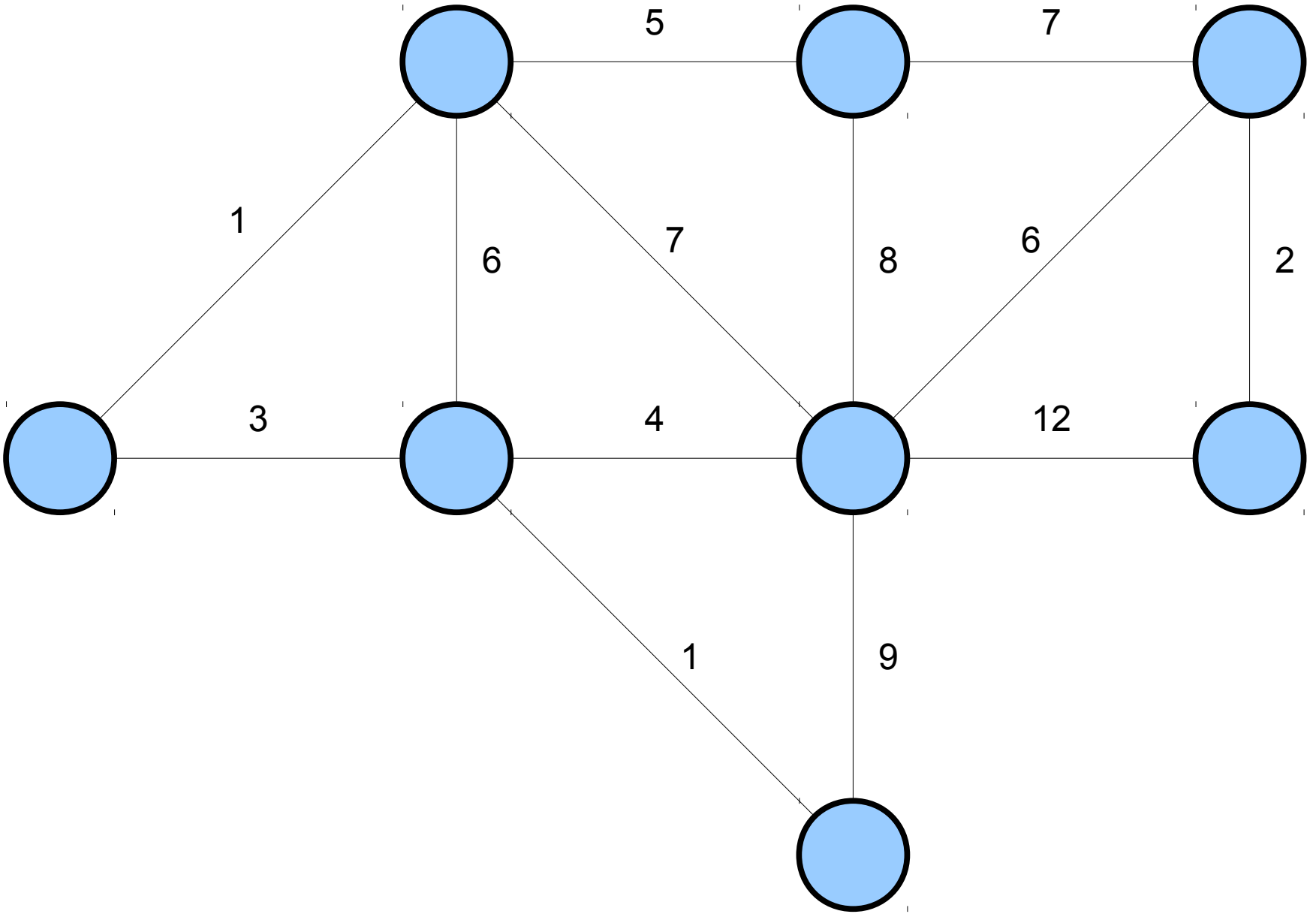

5
7
1
6
7
8
6
2
3
4
12
1
9
This graph is not
connected.

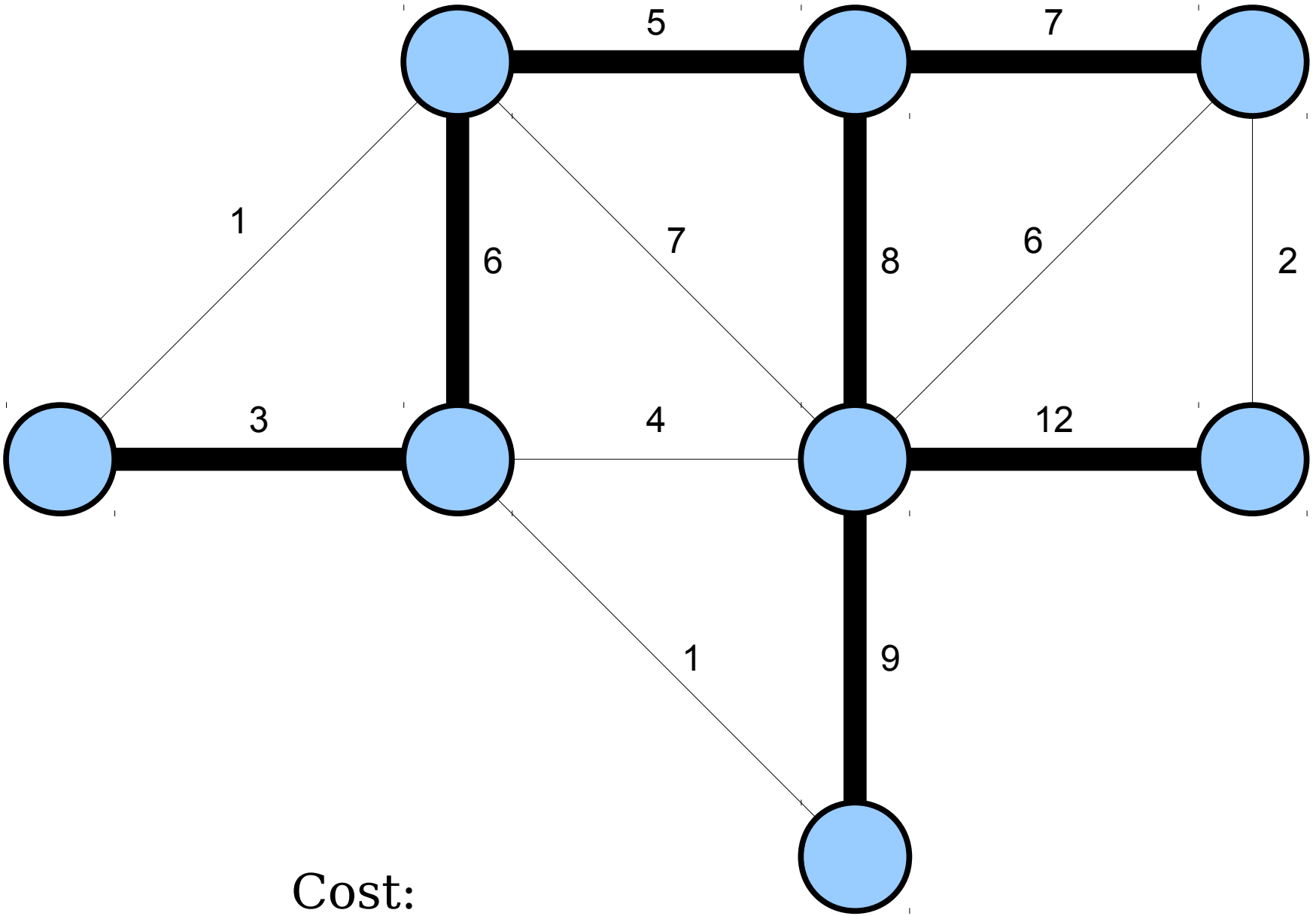

5
7
1
6
7
8
6
2
3
4
12
1
9



There is a *cycle* in this graph. It can't be the cheapest way to link everything.

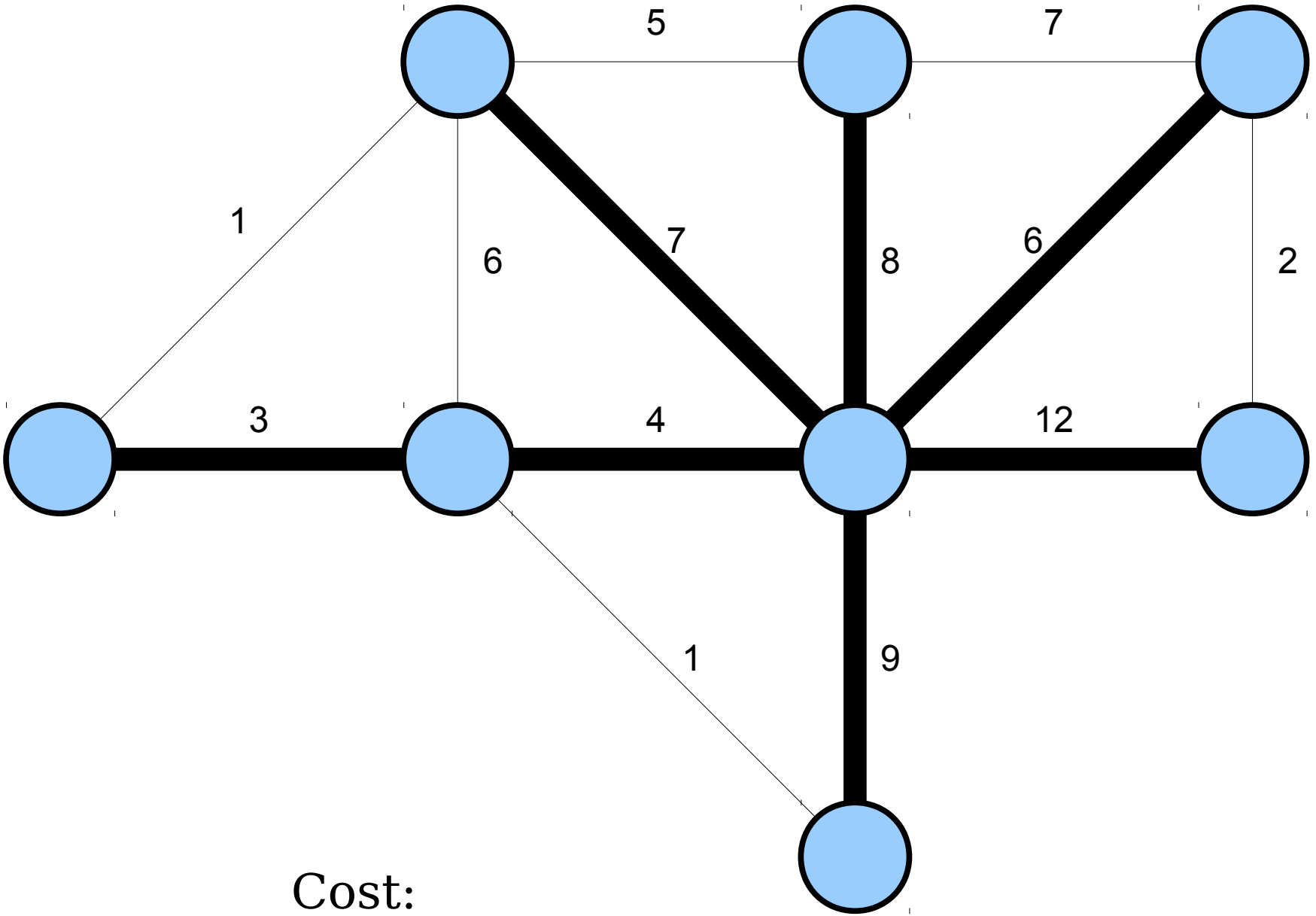
A ***spanning tree*** in an undirected graph is a set of edges with no cycles that connects all nodes.


5
7
1
6
7
8
6
2
3
4
12
1
9



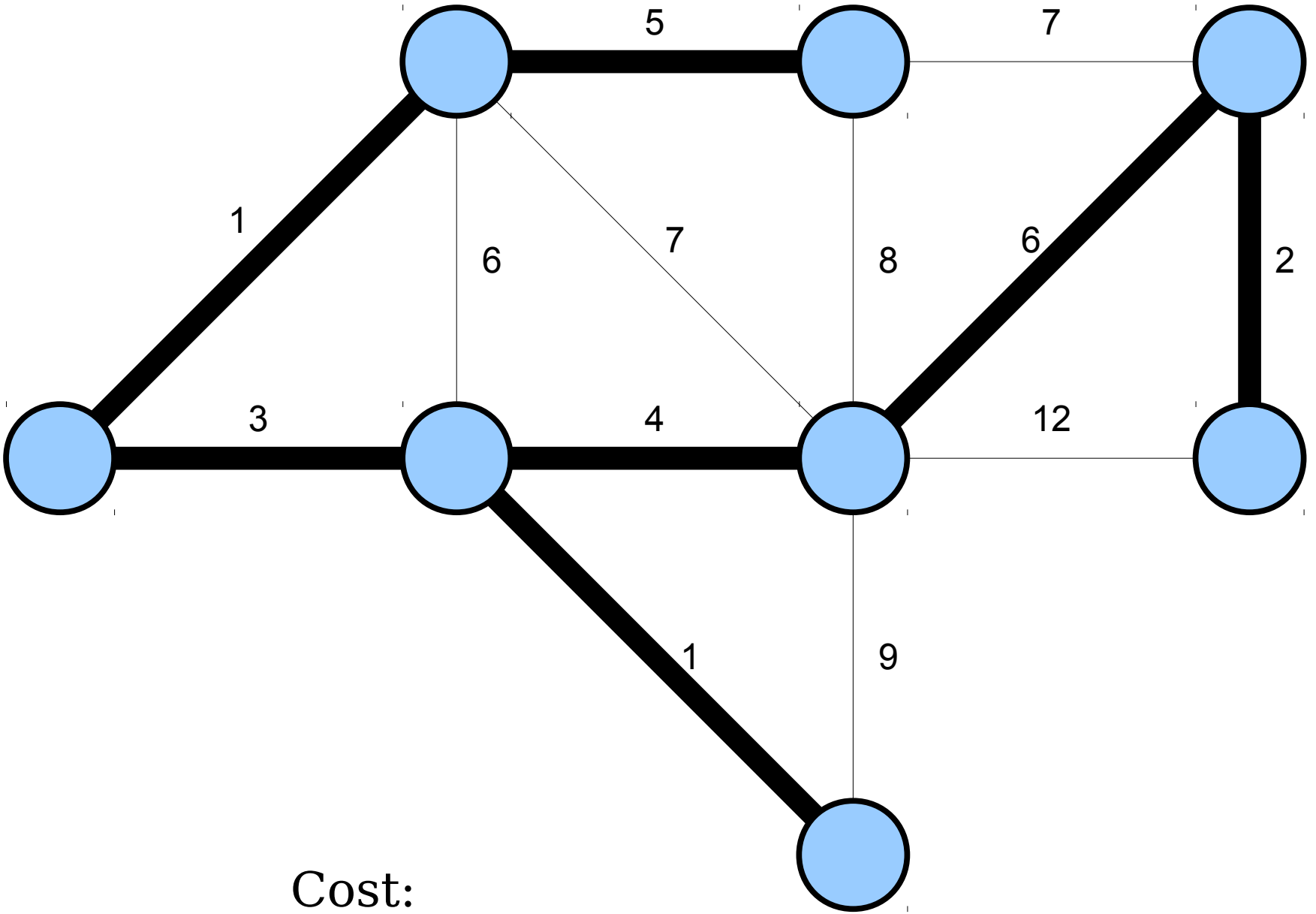
Cost:

3 + 6 + 5 + 7 + 8 + 12 + 9 = **50**



Cost:

3 + 4 + 7 + 8 + 6 + 12 + 9 = **49**



Cost:

1 + 3 + 5 + 4 + 1 + 6 + 2 = **22**

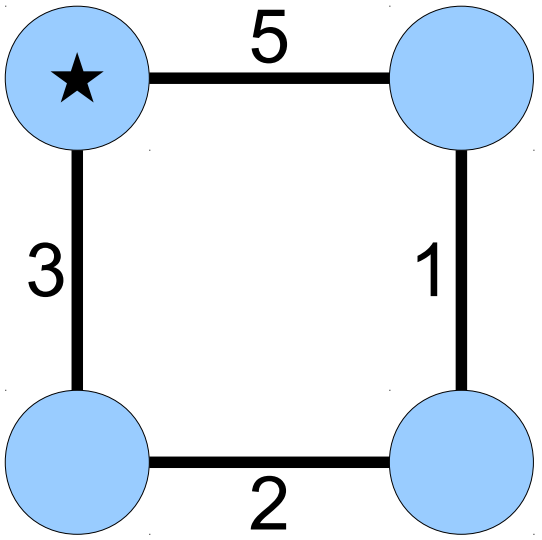
A ***minimum spanning tree*** (or ***MST***) is a spanning tree with the least total cost.

Applications

- ***Electric Grids***
  - Given a collection of houses, where do you lay wires to connect all houses with the least total cost?
  - This was the initial motivation for studying minimum spanning trees in the early 1920's. (work done by Czech mathematician ***Otakar Borůvka***)
- ***Data Clustering***
  - More on that later...
- ***Maze Generation***
  - More on that later...
- ***Computational Biology***
  - More on that later...

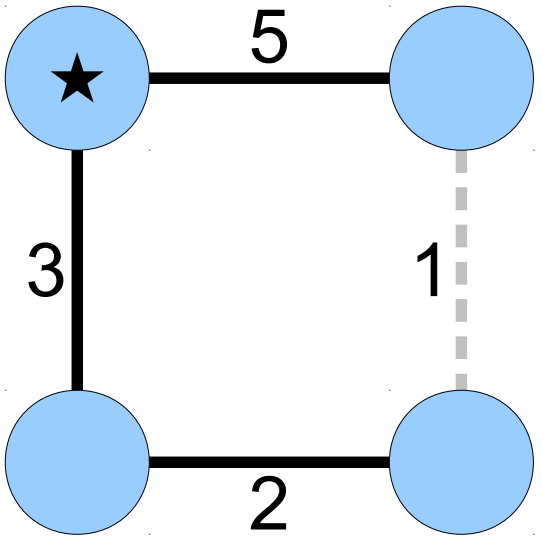
Shortest-Path Trees and MSTs

- Last time, we saw how Dijkstra's algorithm and A* search can be used to find shortest path trees in a graph.
- Note that a shortest-path tree might not be an MST and vice-versa.



Shortest-Path Trees and MSTs

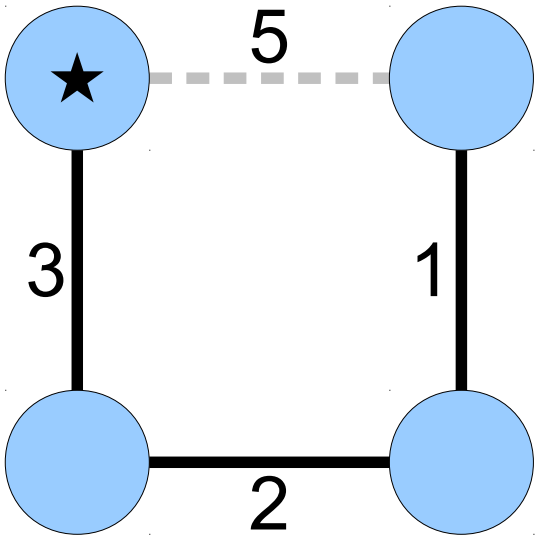
- Last time, we saw how Dijkstra's algorithm and A* search can be used to find shortest path trees in a graph.
- Note that a shortest-path tree might not be an MST and vice-versa.



Shortest-path tree for the starred node.

Shortest-Path Trees and MSTs

- Last time, we saw how Dijkstra's algorithm and A* search can be used to find shortest path trees in a graph.
- Note that a shortest-path tree might not be an MST and vice-versa.



Minimum spanning tree in the graph.

Finding an MST

MST Algorithms

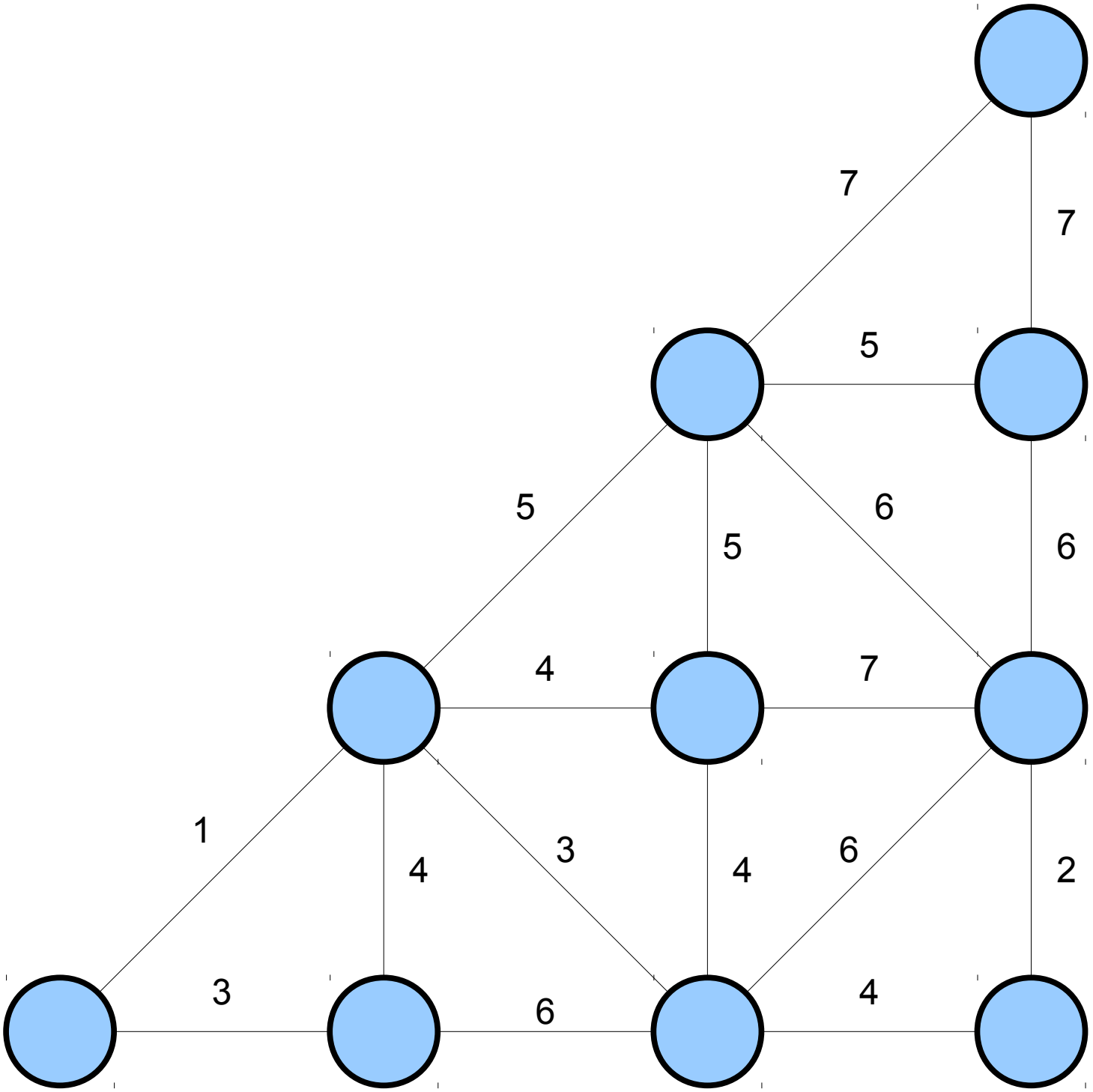
- The original MST algorithm (1926) that Borůvka proposed is now called ***Borůvka's algorithm***.
- Later, the Czech mathematician Vojtěch Jarník (1930) invented an algorithm now called ***Prim's algorithm***.
- After that, American mathematician Joseph Kruskal (1956) developed what's now called ***Kruskal's algorithm***, which is what we'll present today.
- There's been a ton of work since them – come talk to me after class for details!

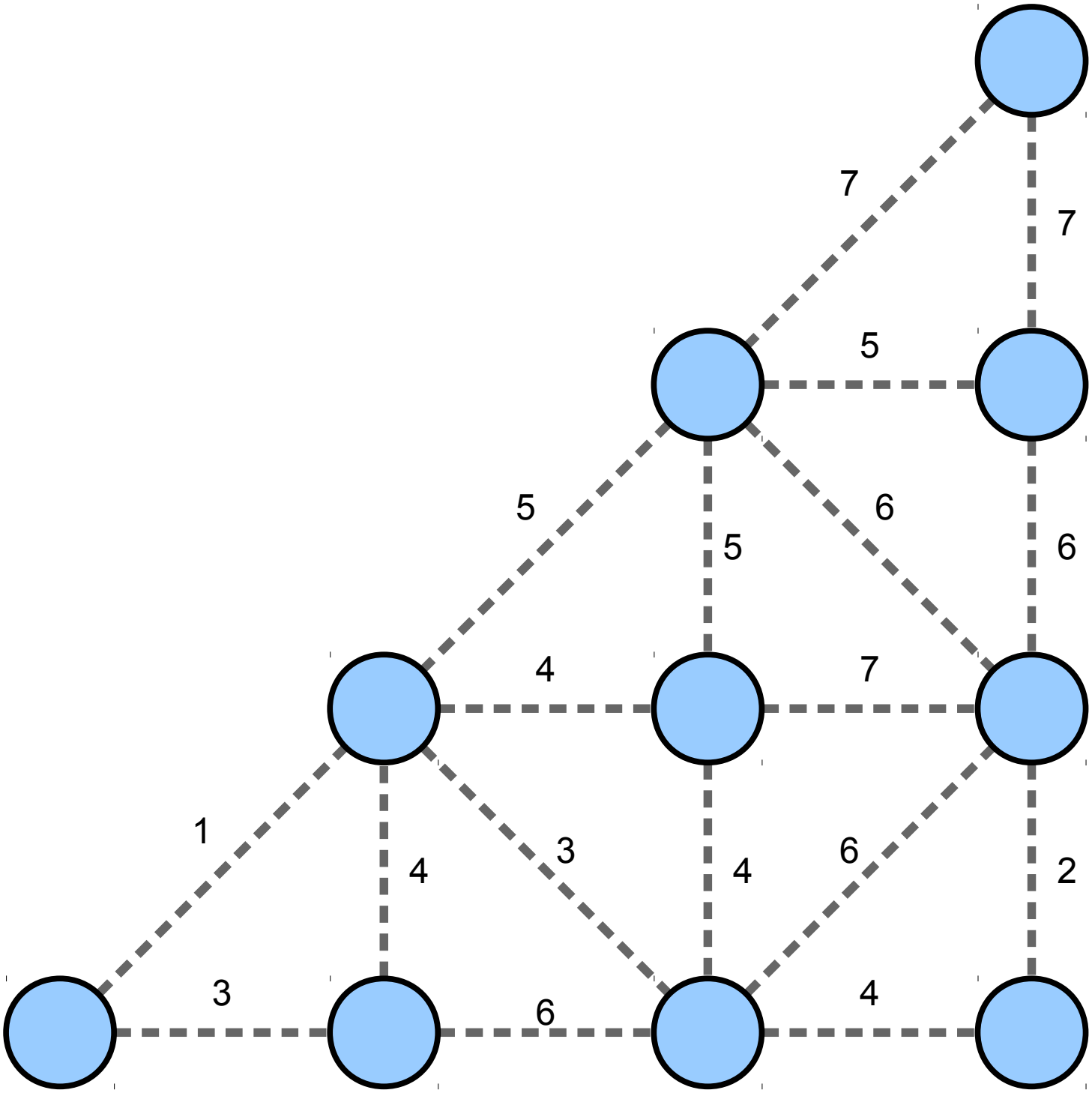
Kruskal's Algorithm:

Remove all edges from the graph.

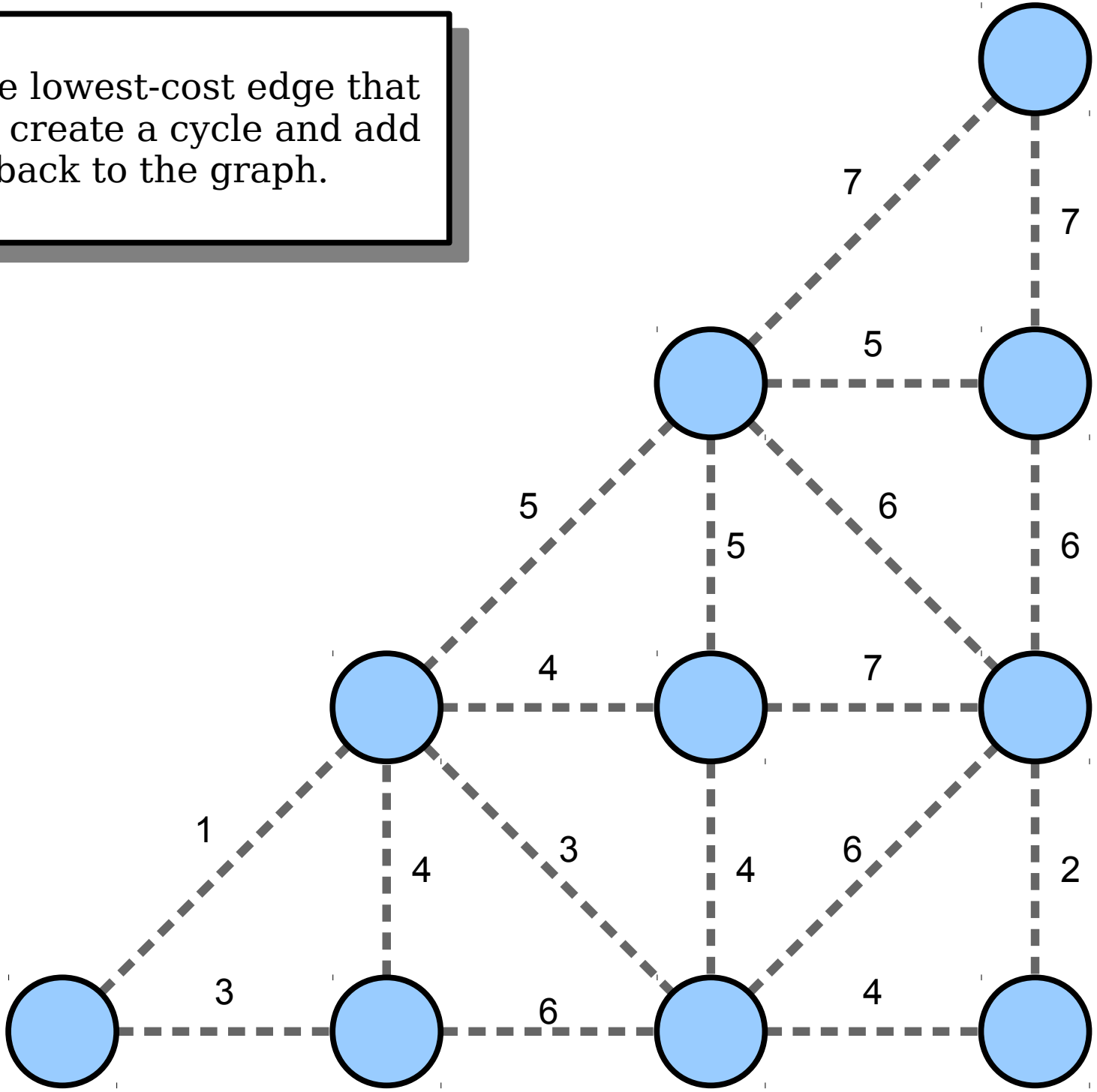
Repeatedly find the cheapest edge that doesn't create a cycle and add it back.

The result is an MST of the overall graph.

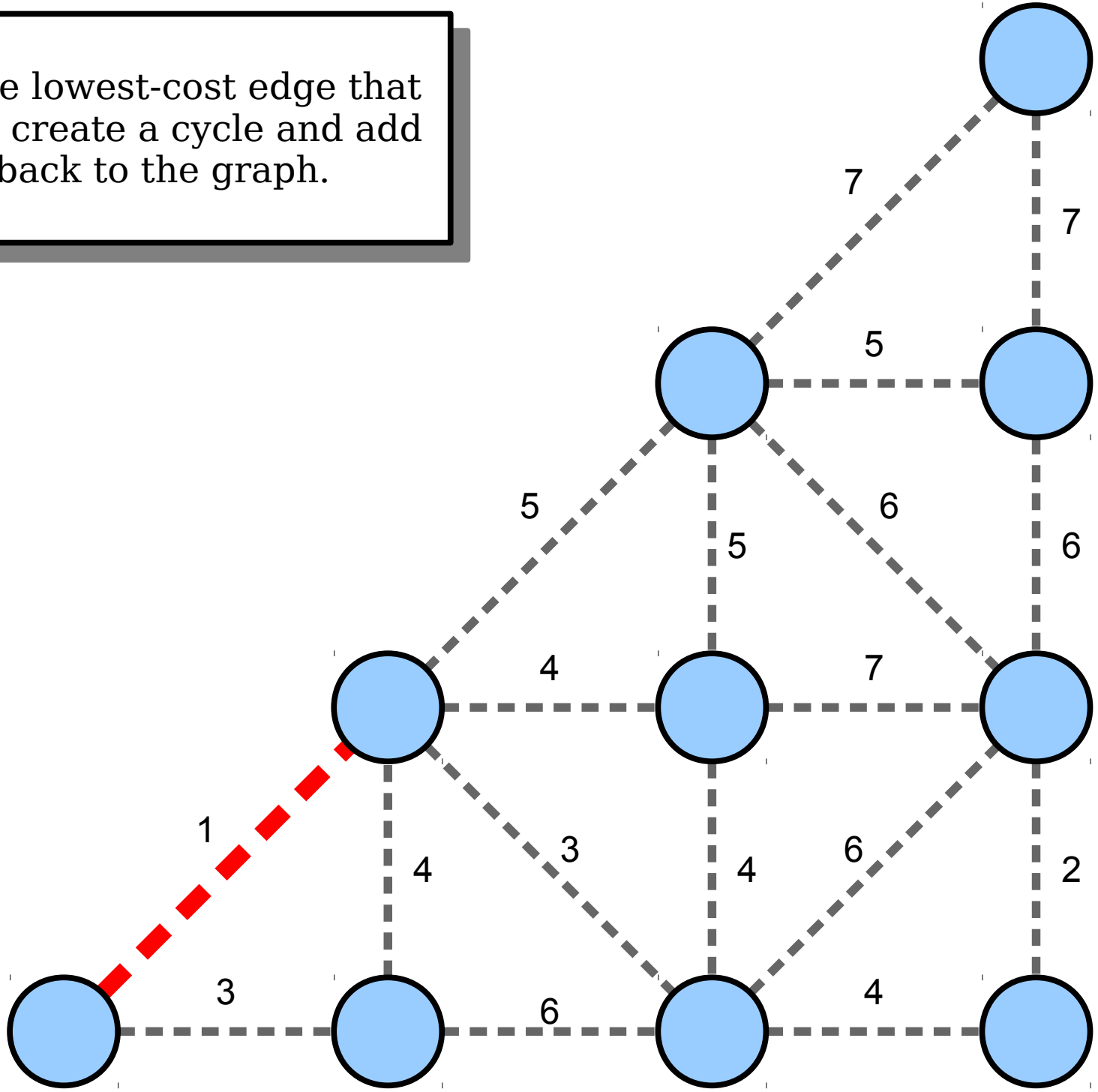

7
7
5
5
6
5
6
4
7
1
4
3
4
6
2
3
6
4


7
7
5
5
5
6
6
4
4
7
1
4
3
4
6
2
3
6
4

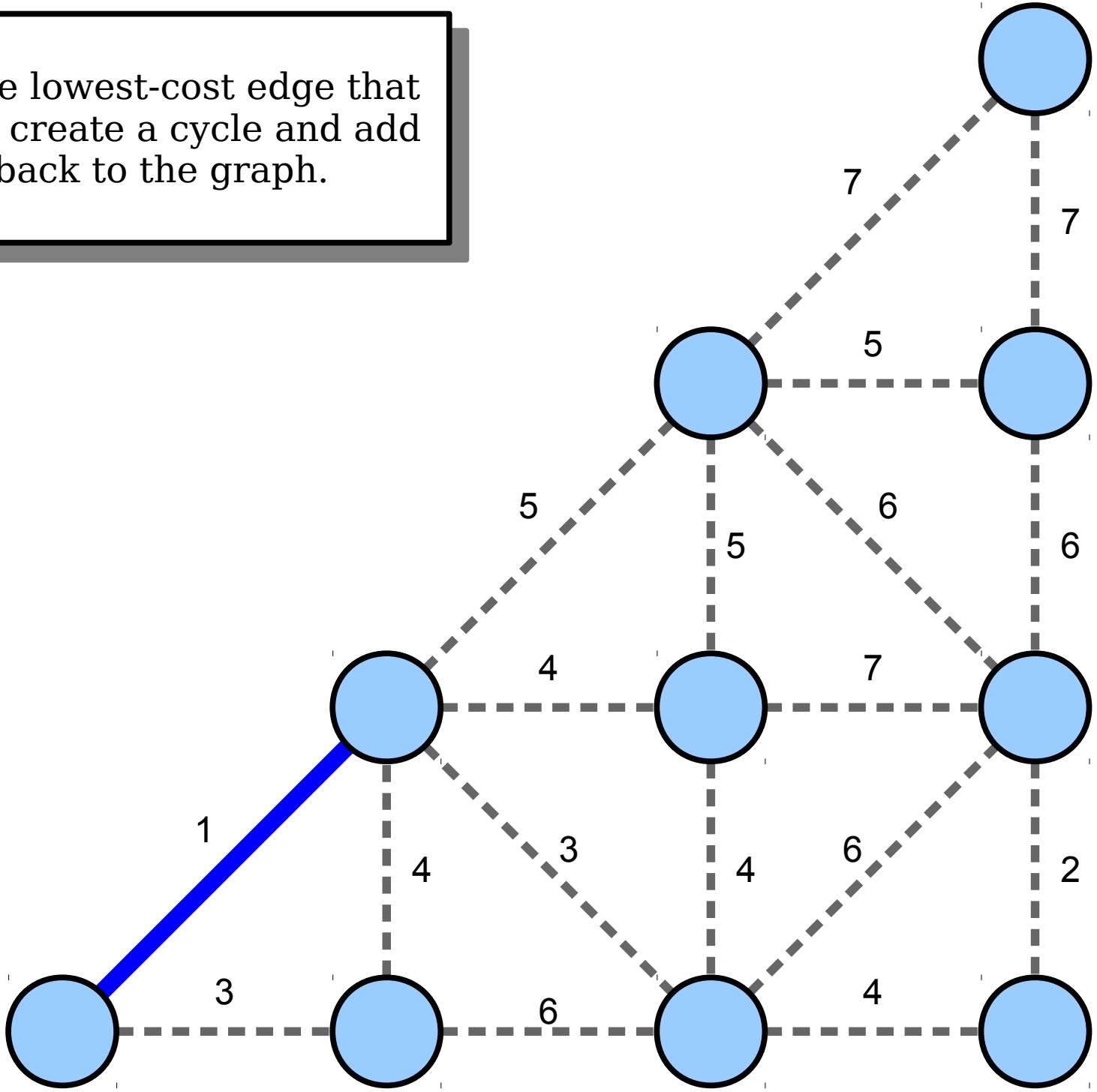
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



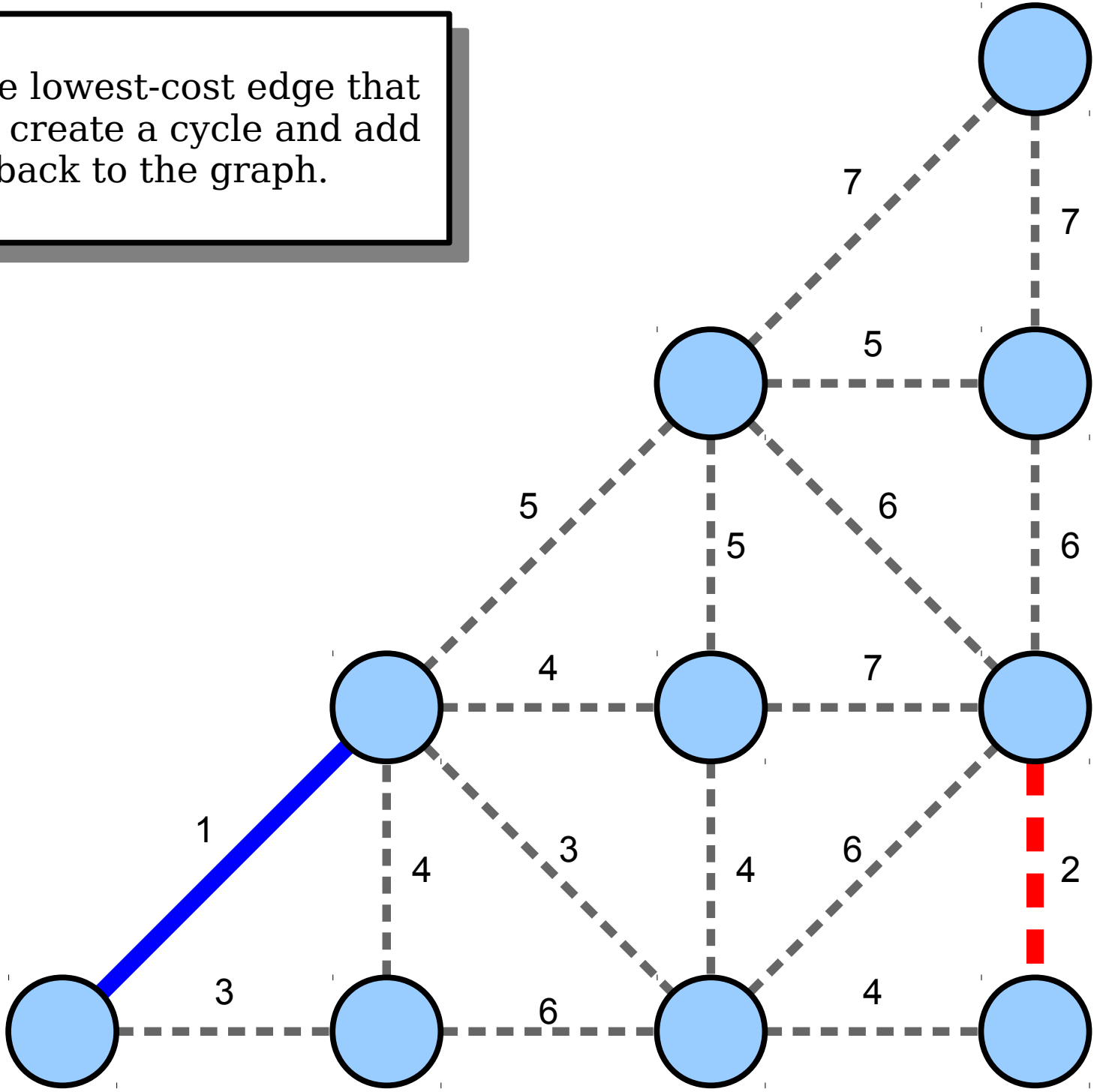
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



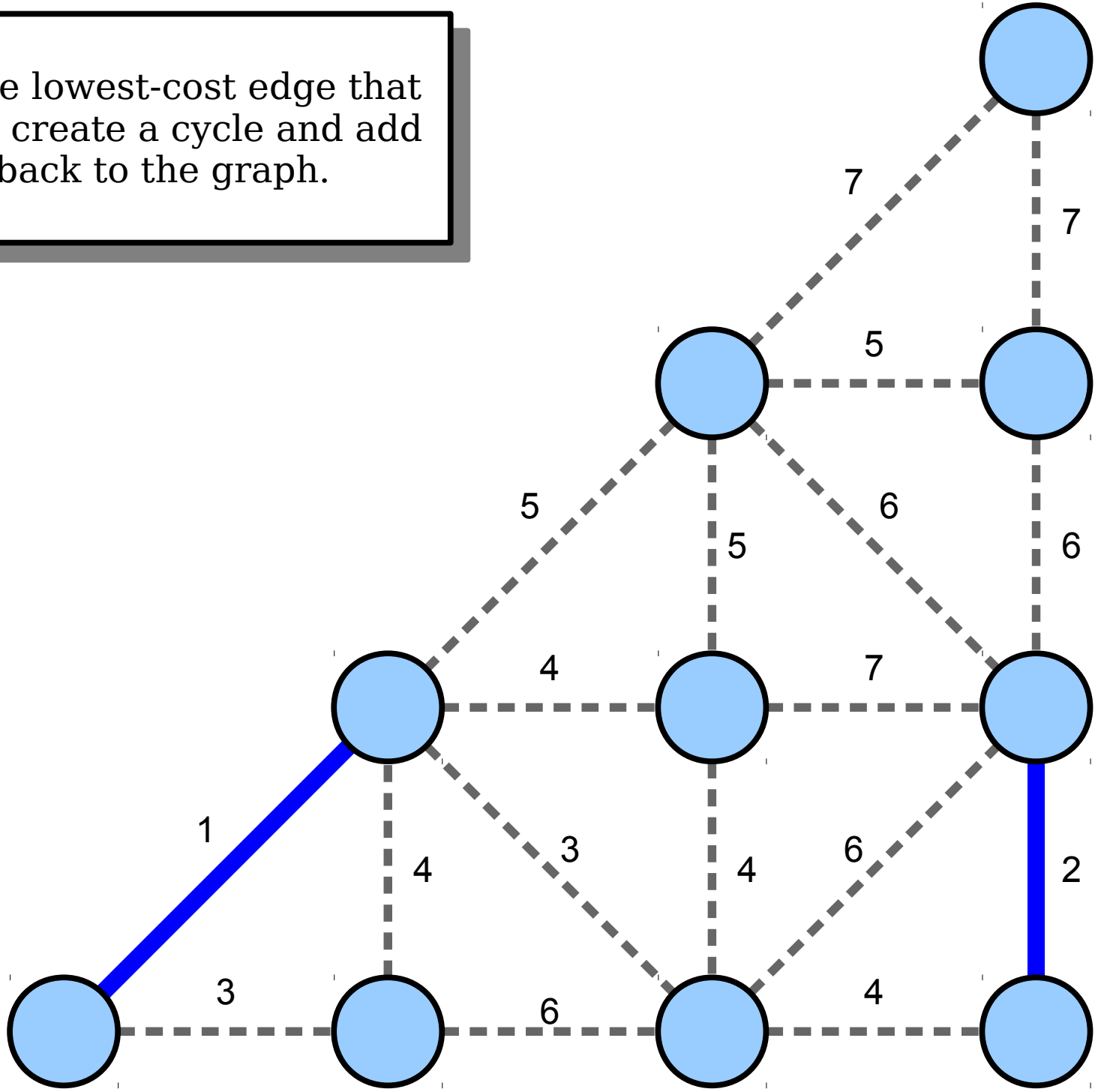
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



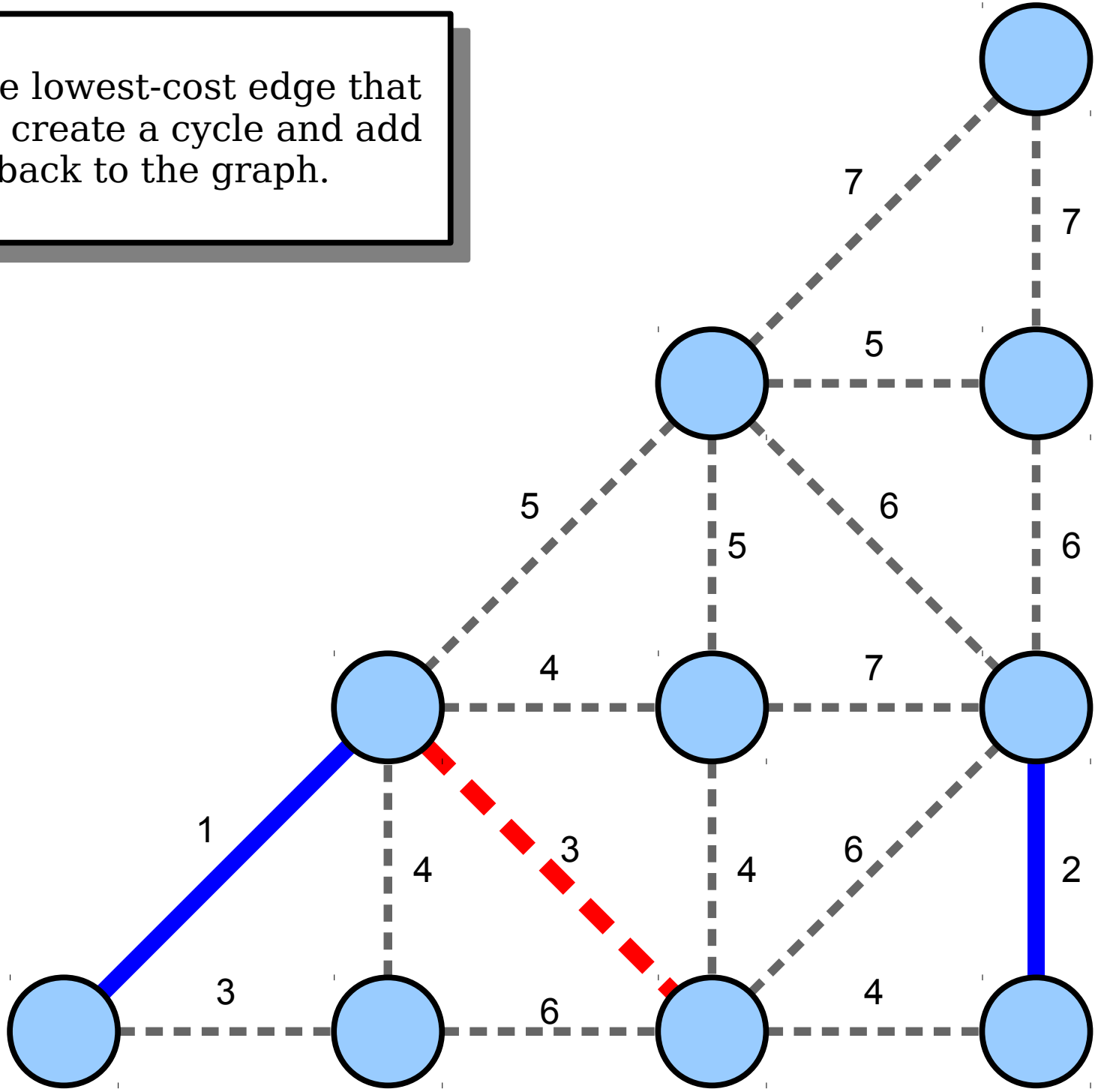
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



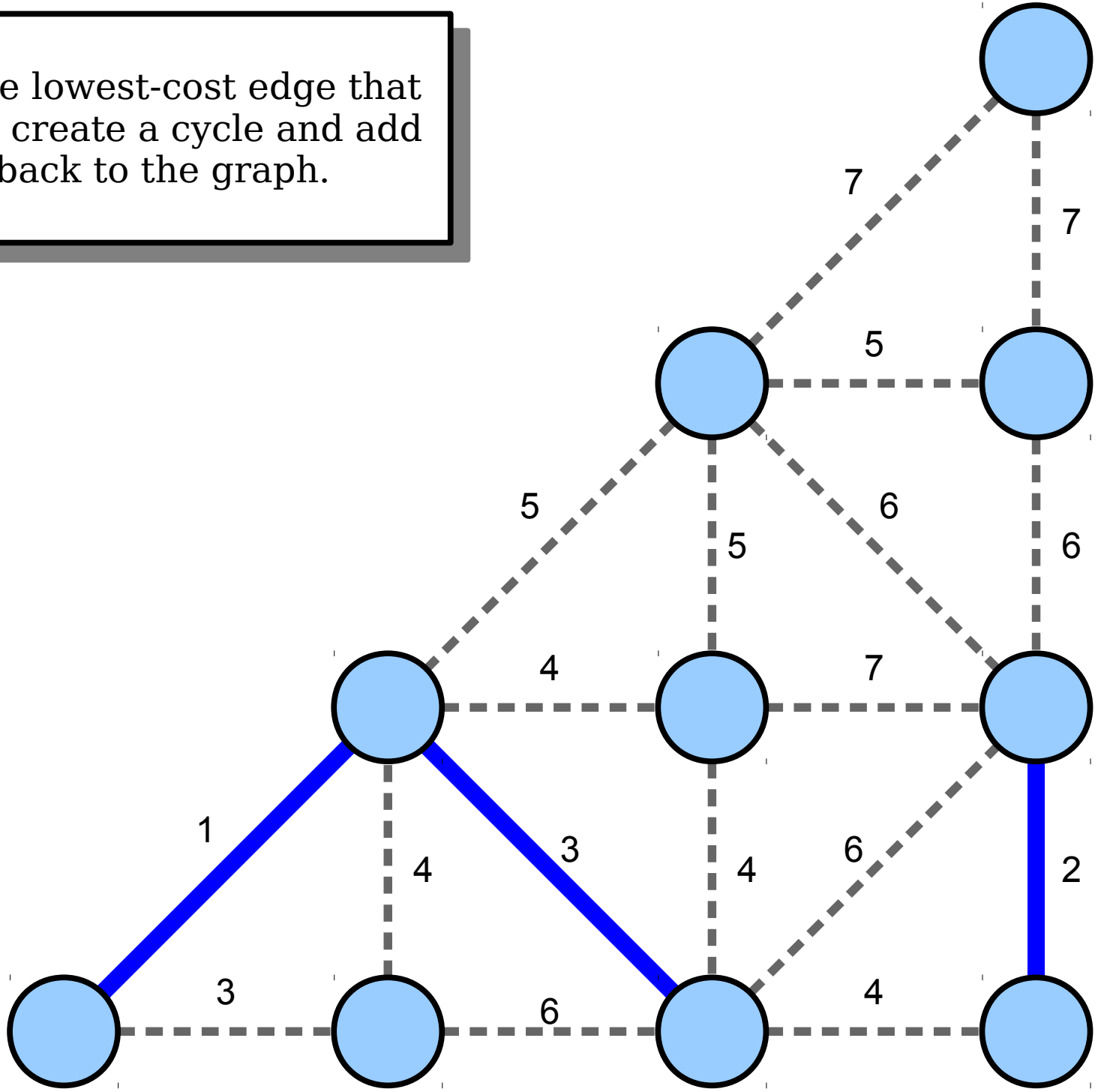
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



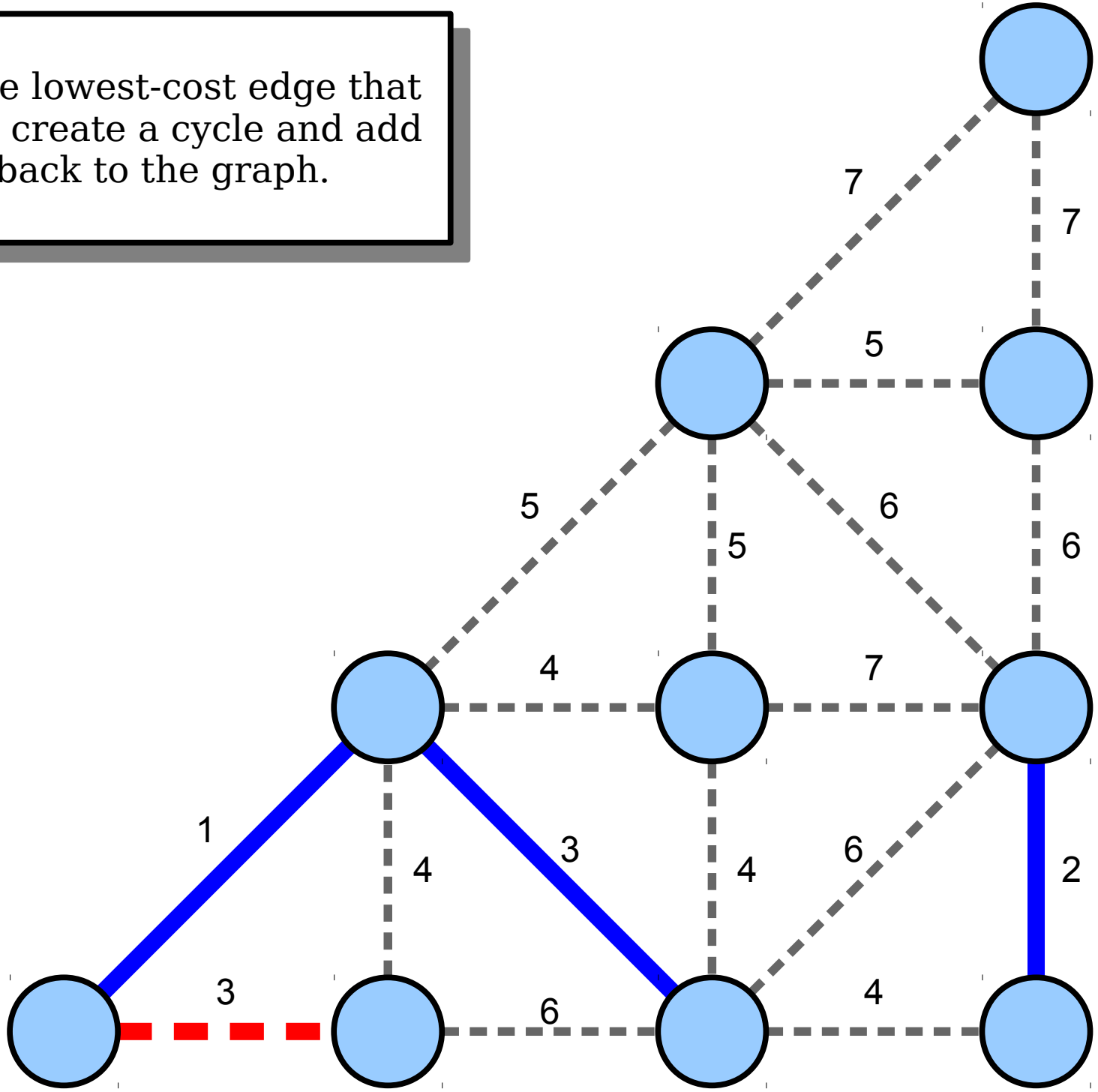
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



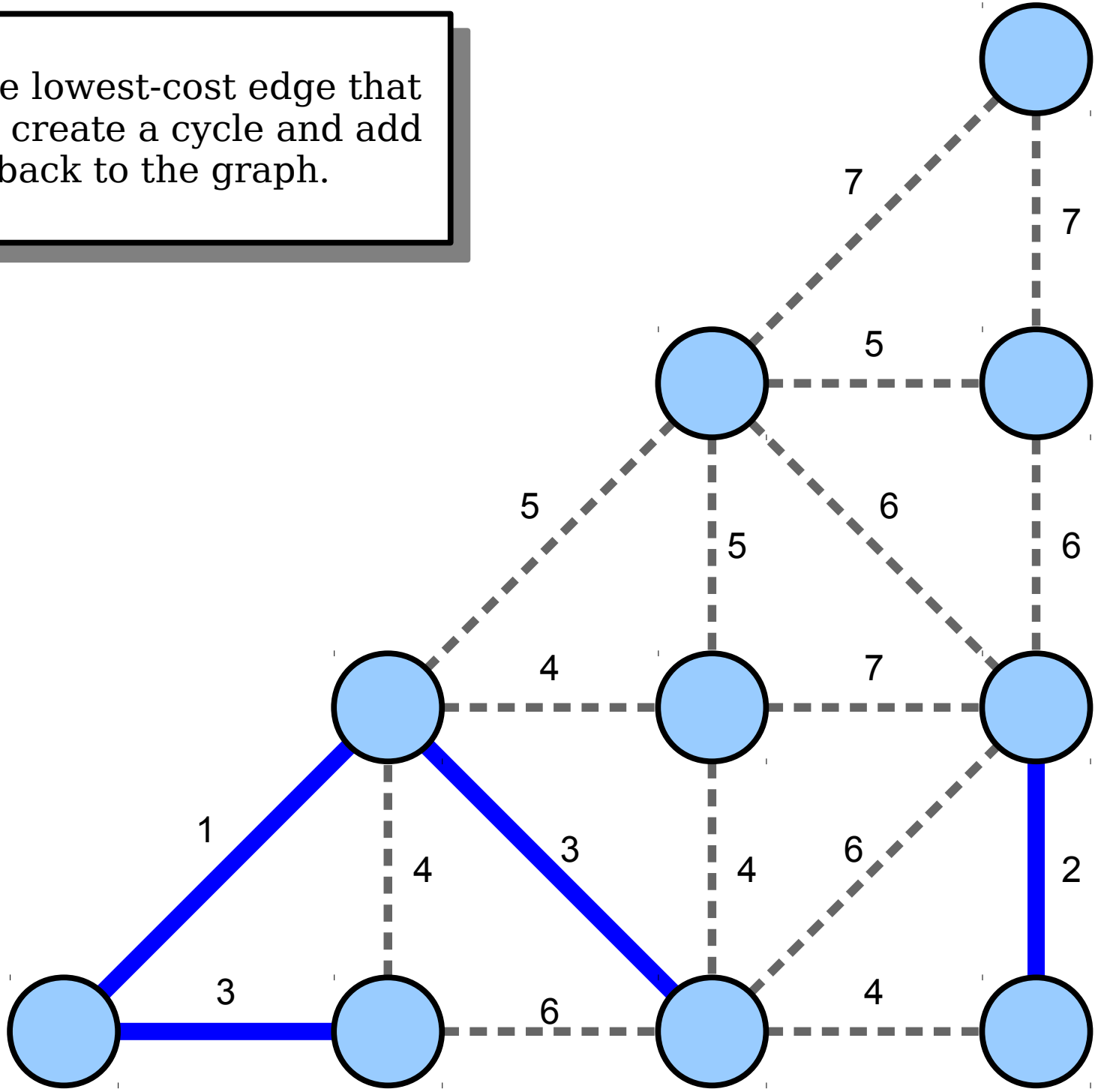
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



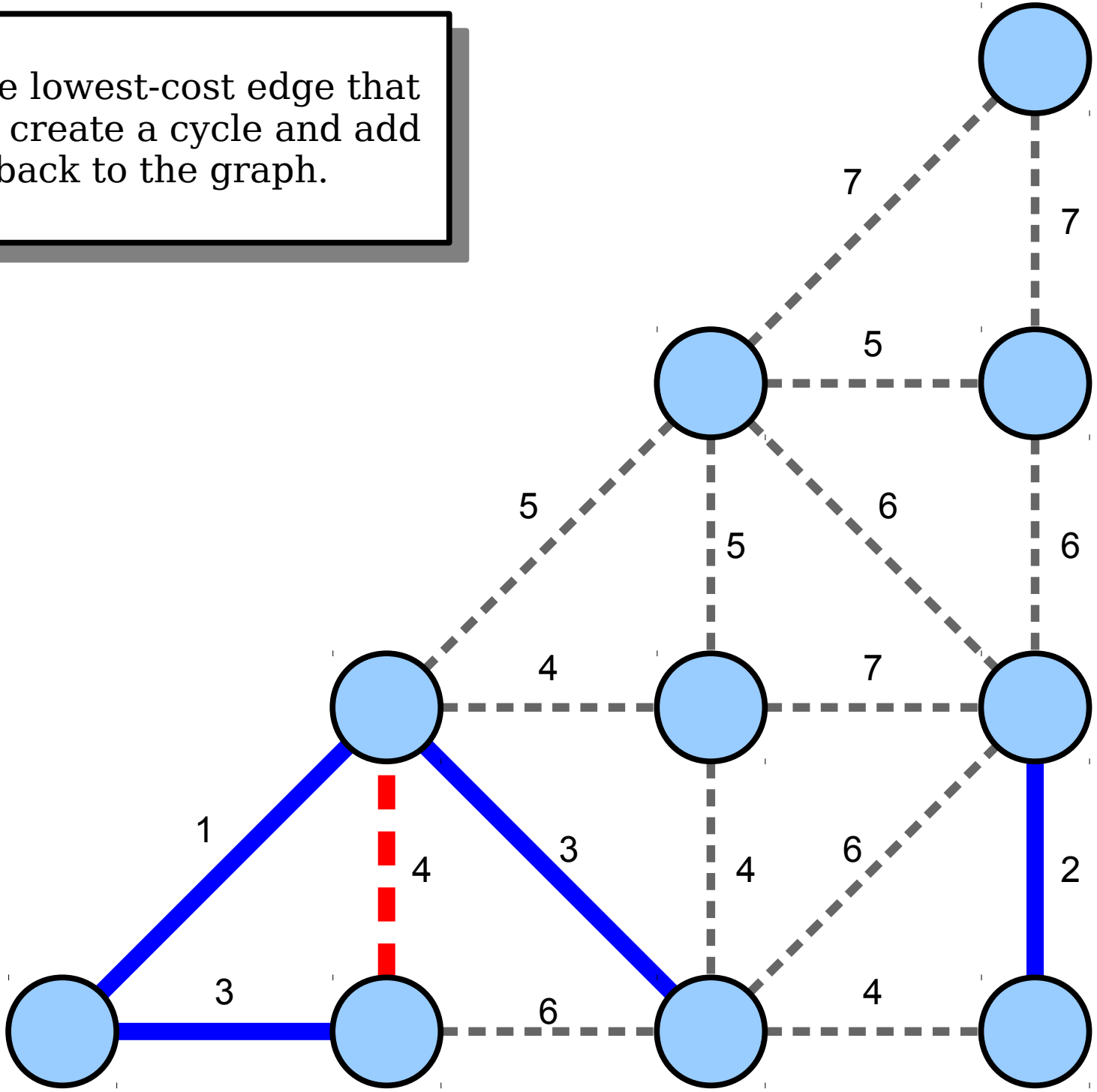
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



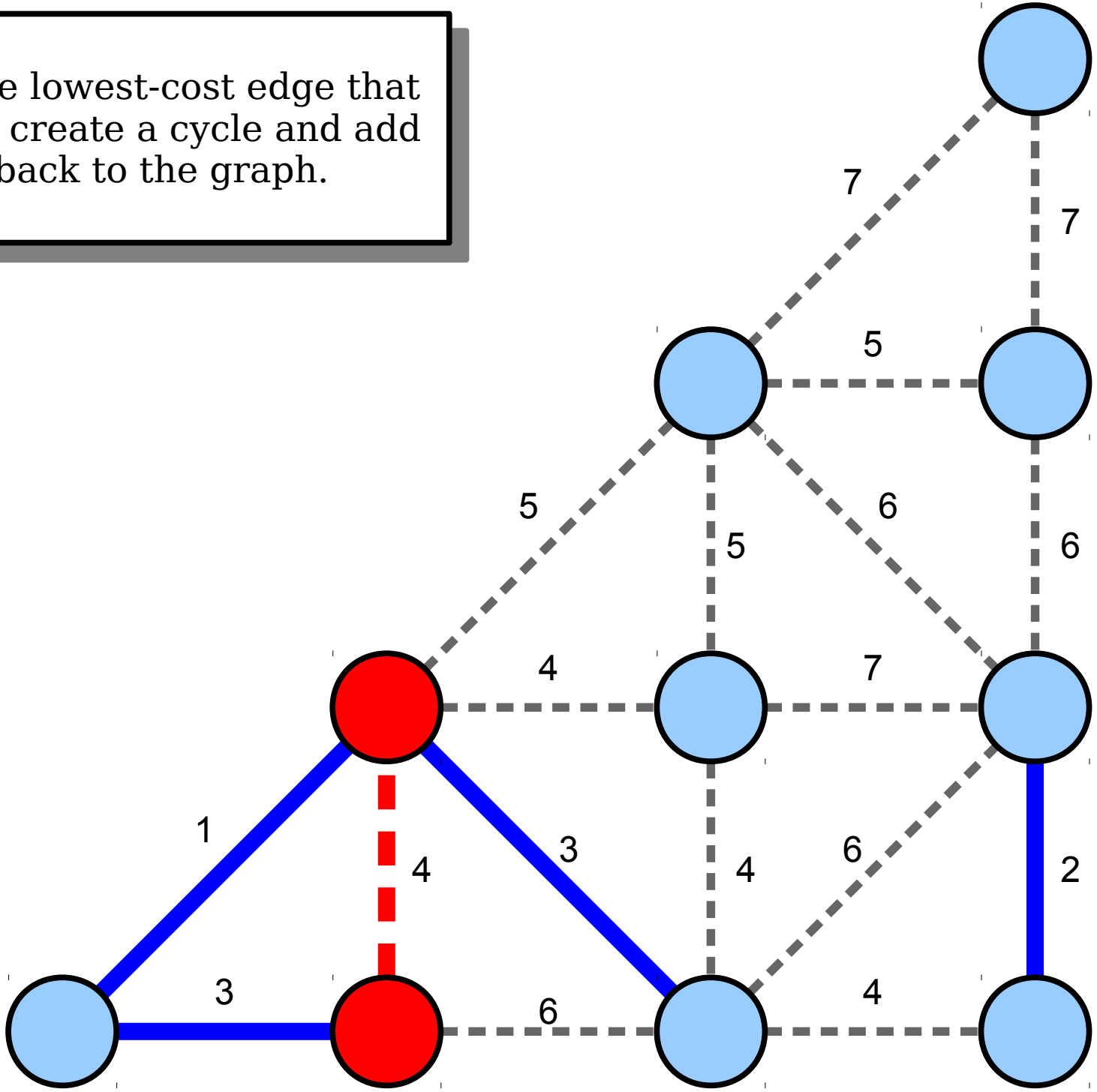
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



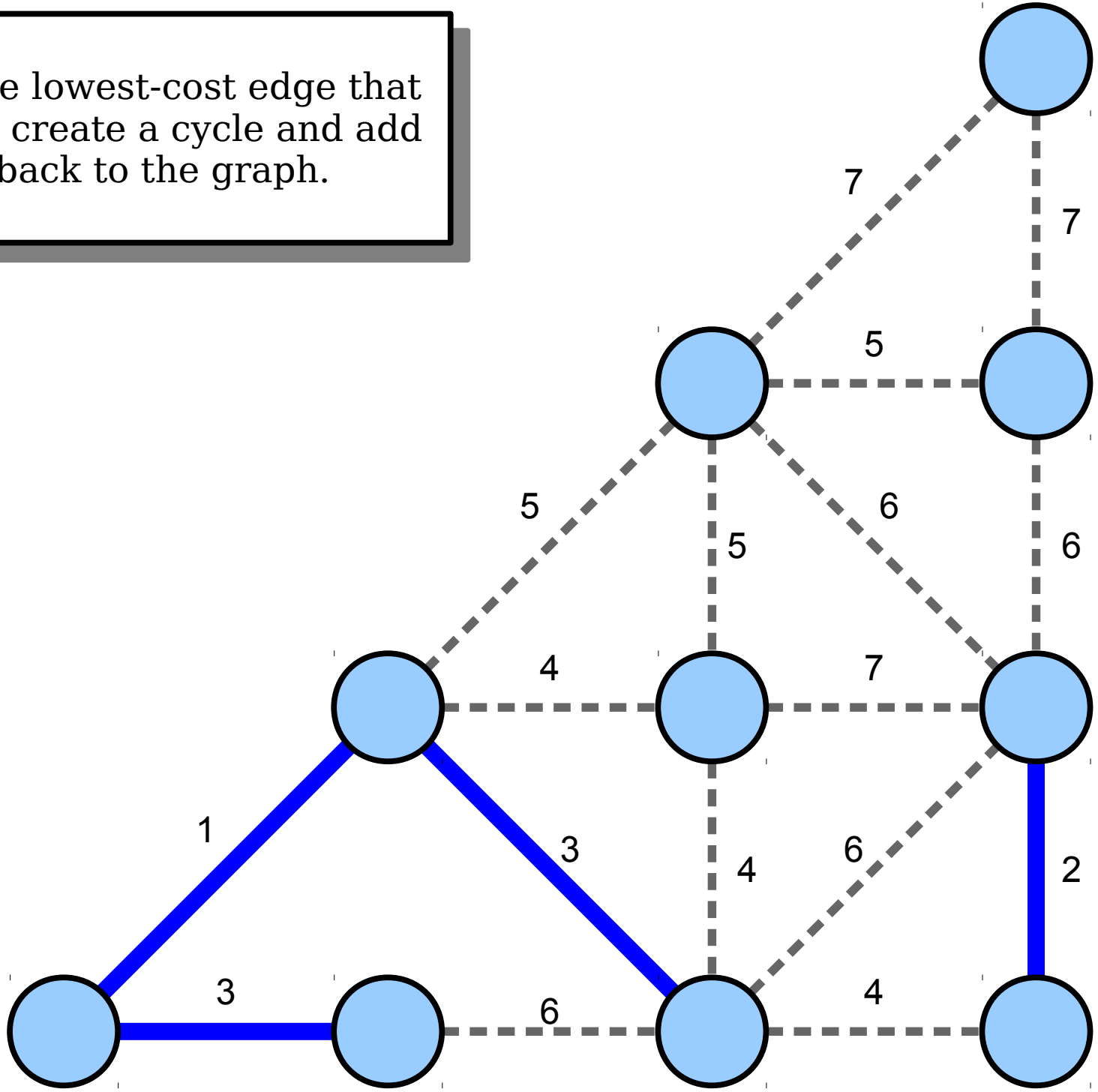
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



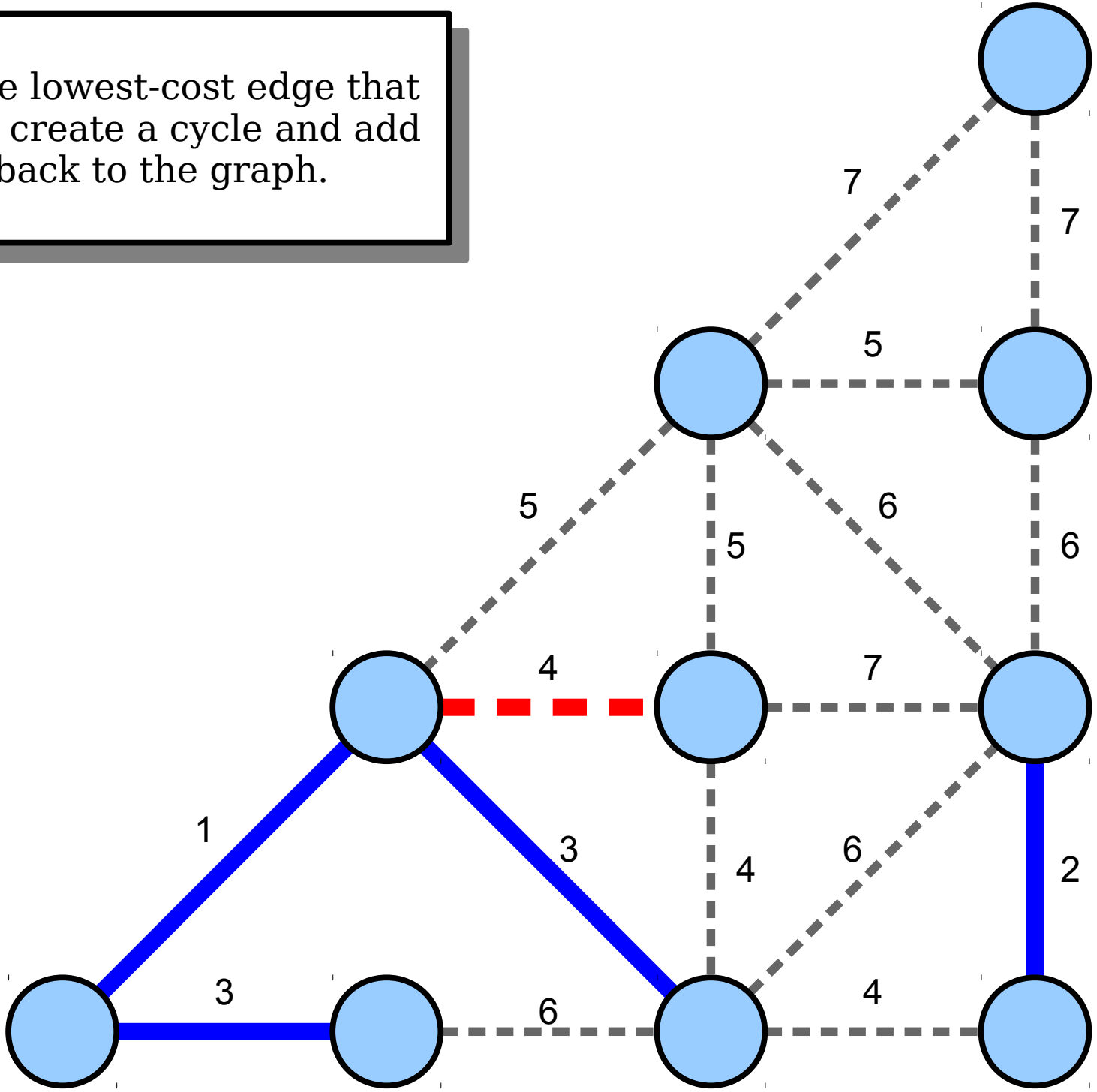
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



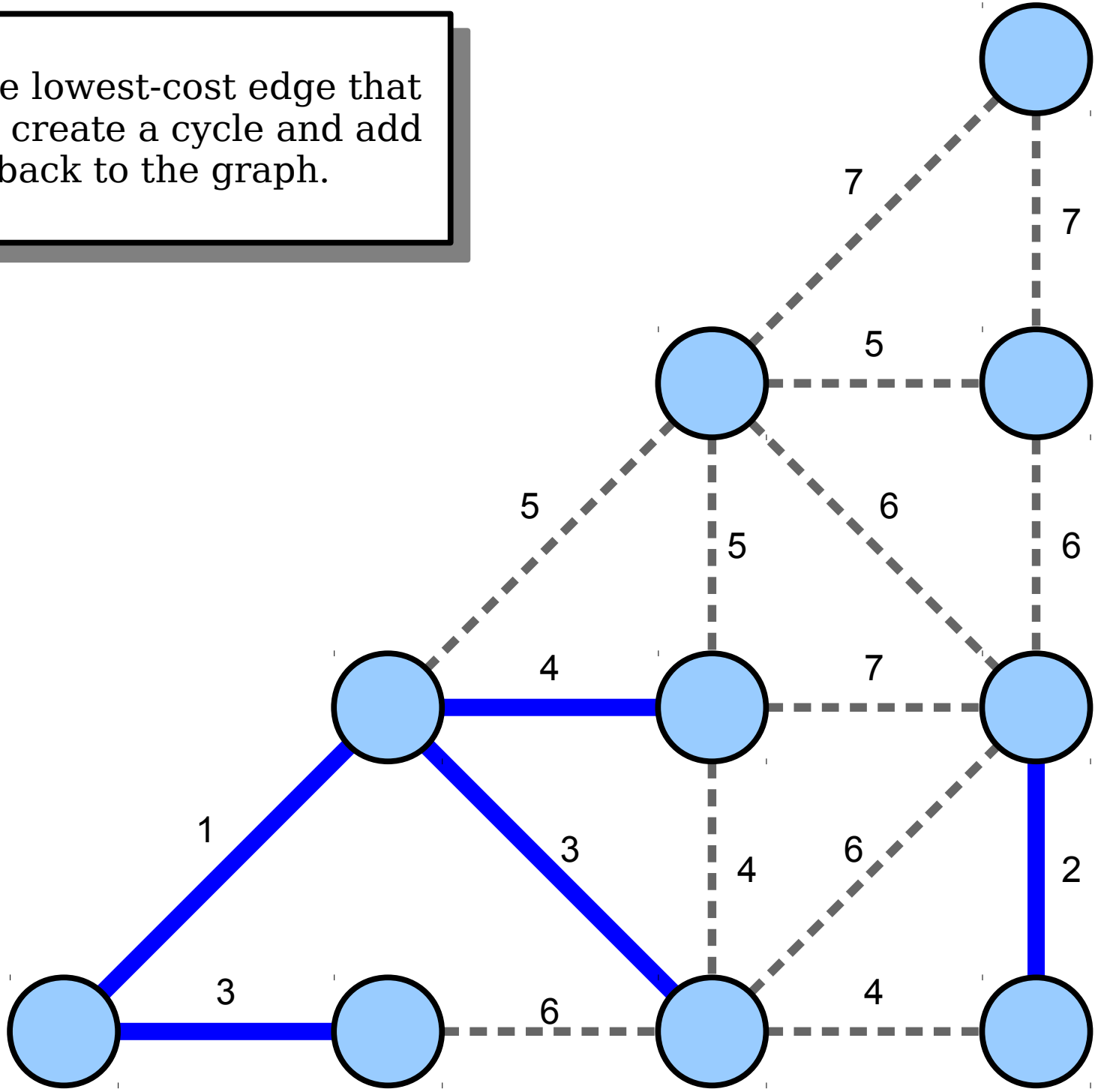
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



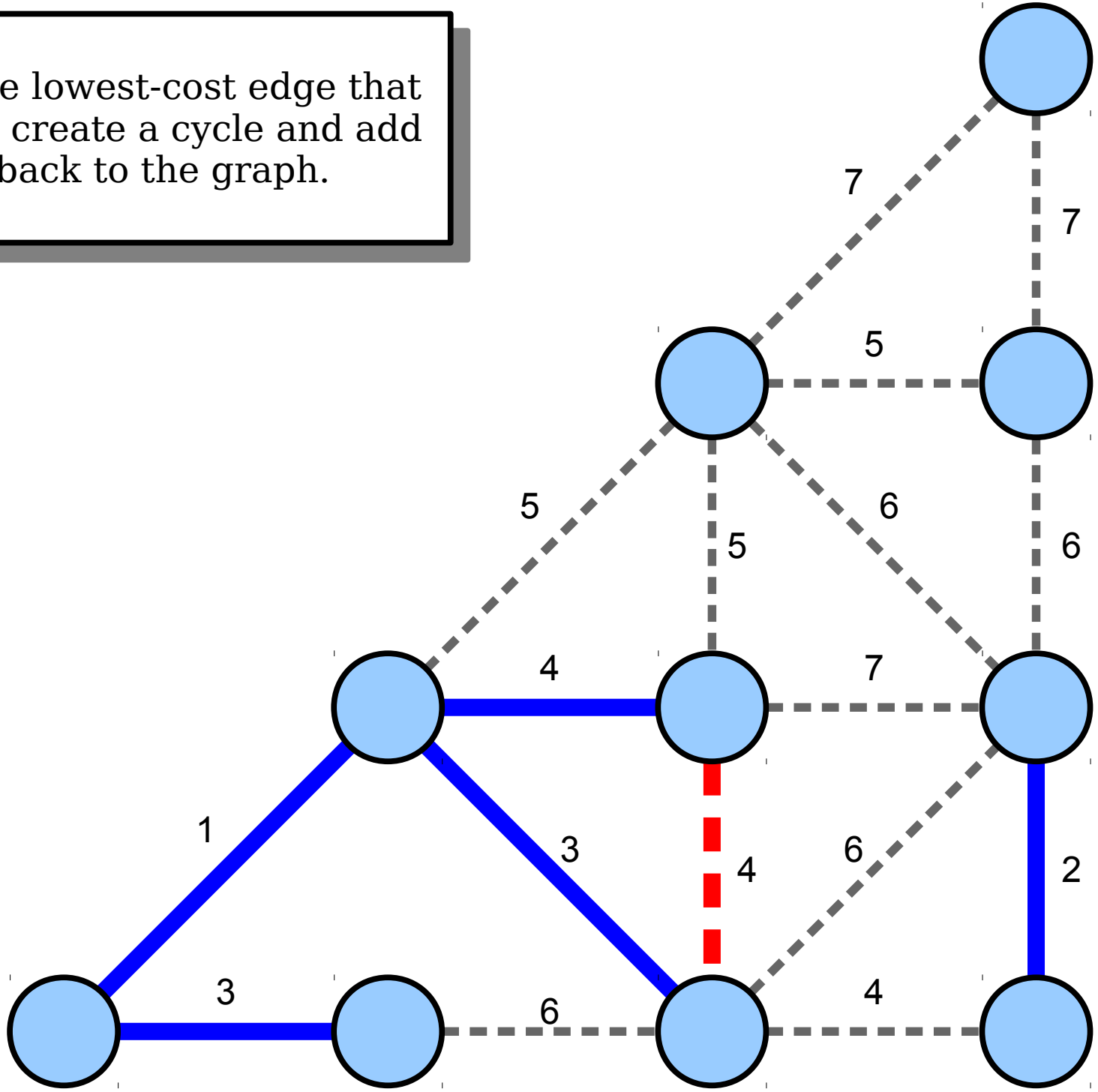
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



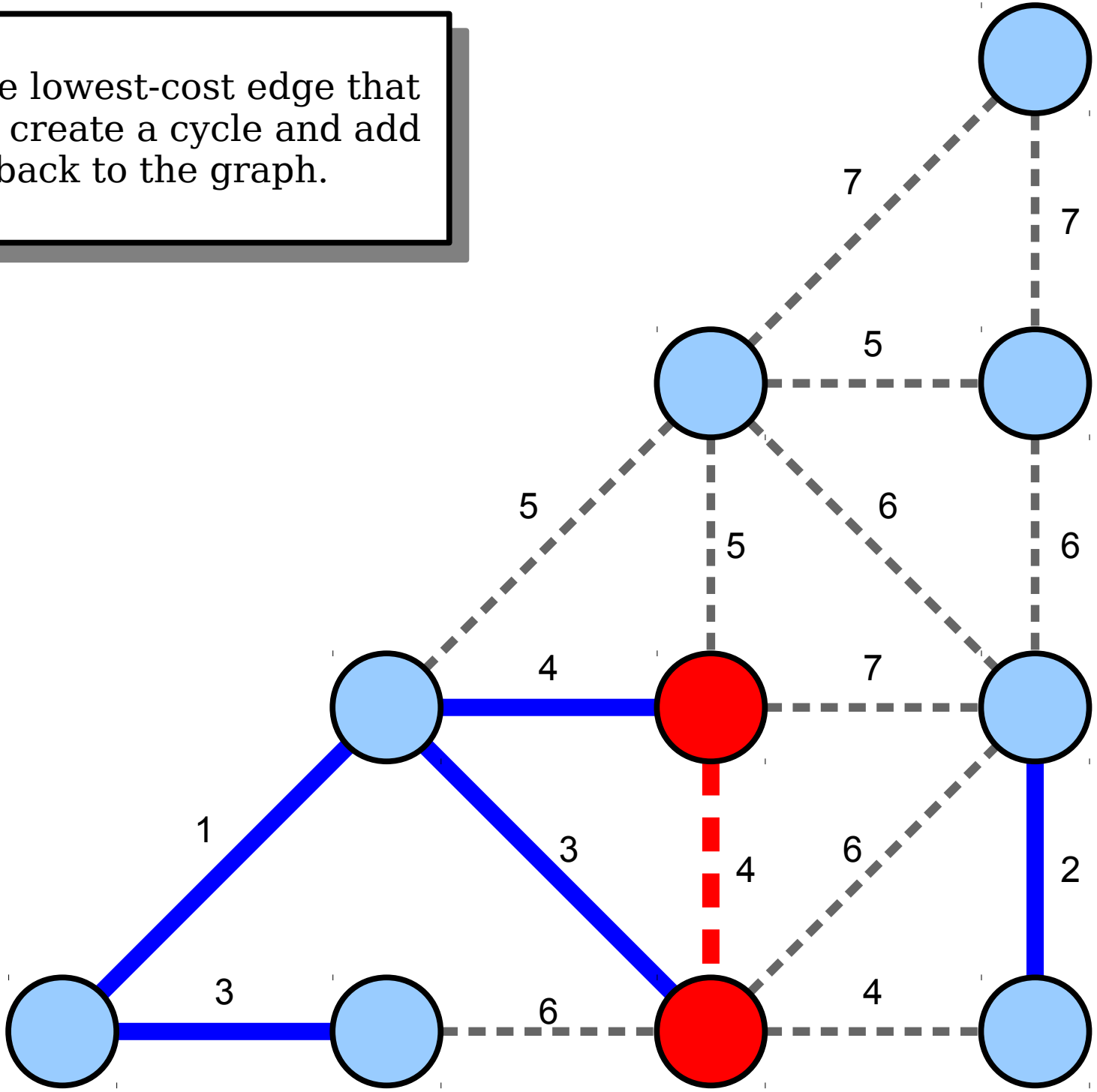
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.

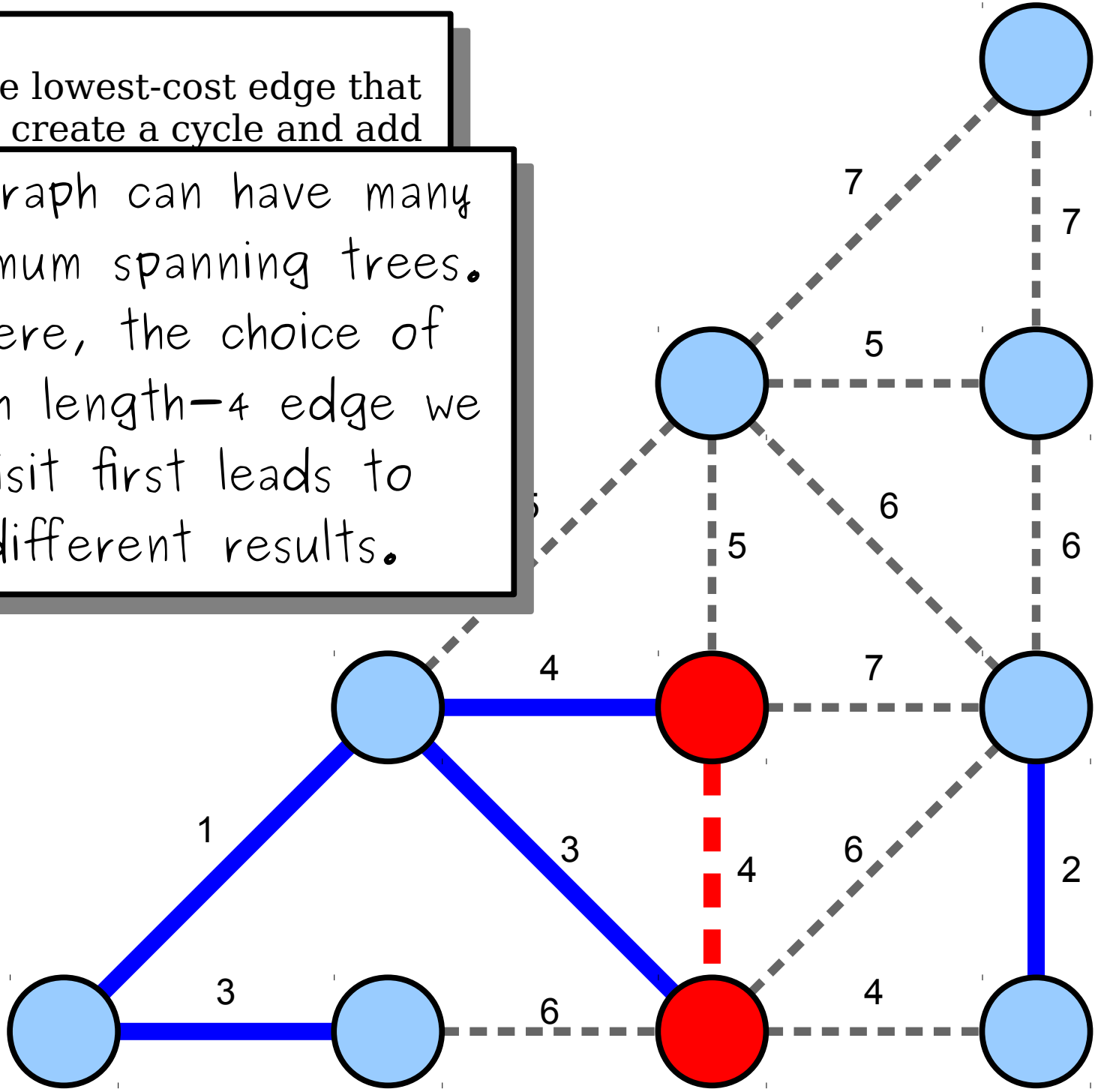


Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.

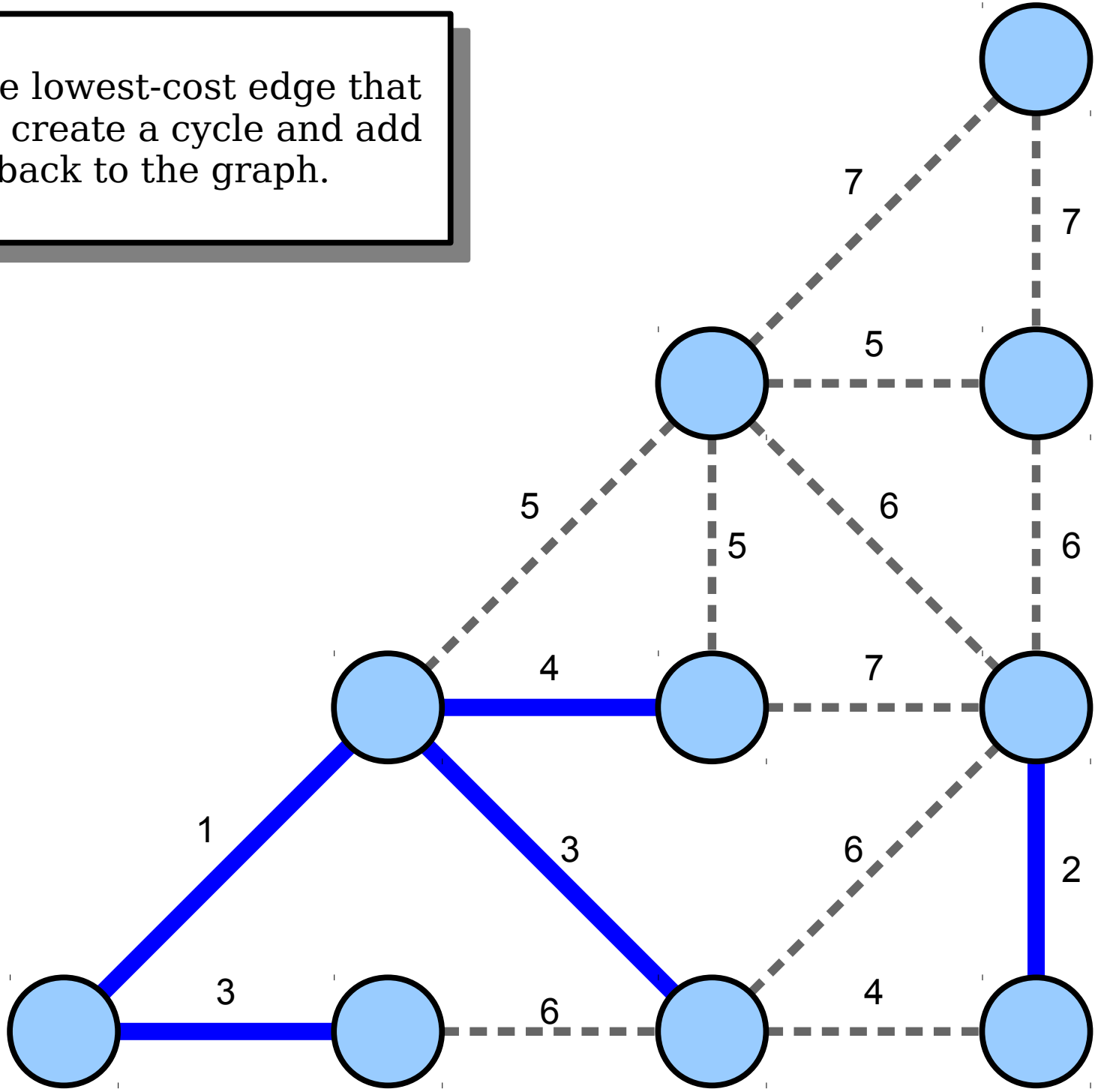


Find the lowest-cost edge that doesn't create a cycle and add

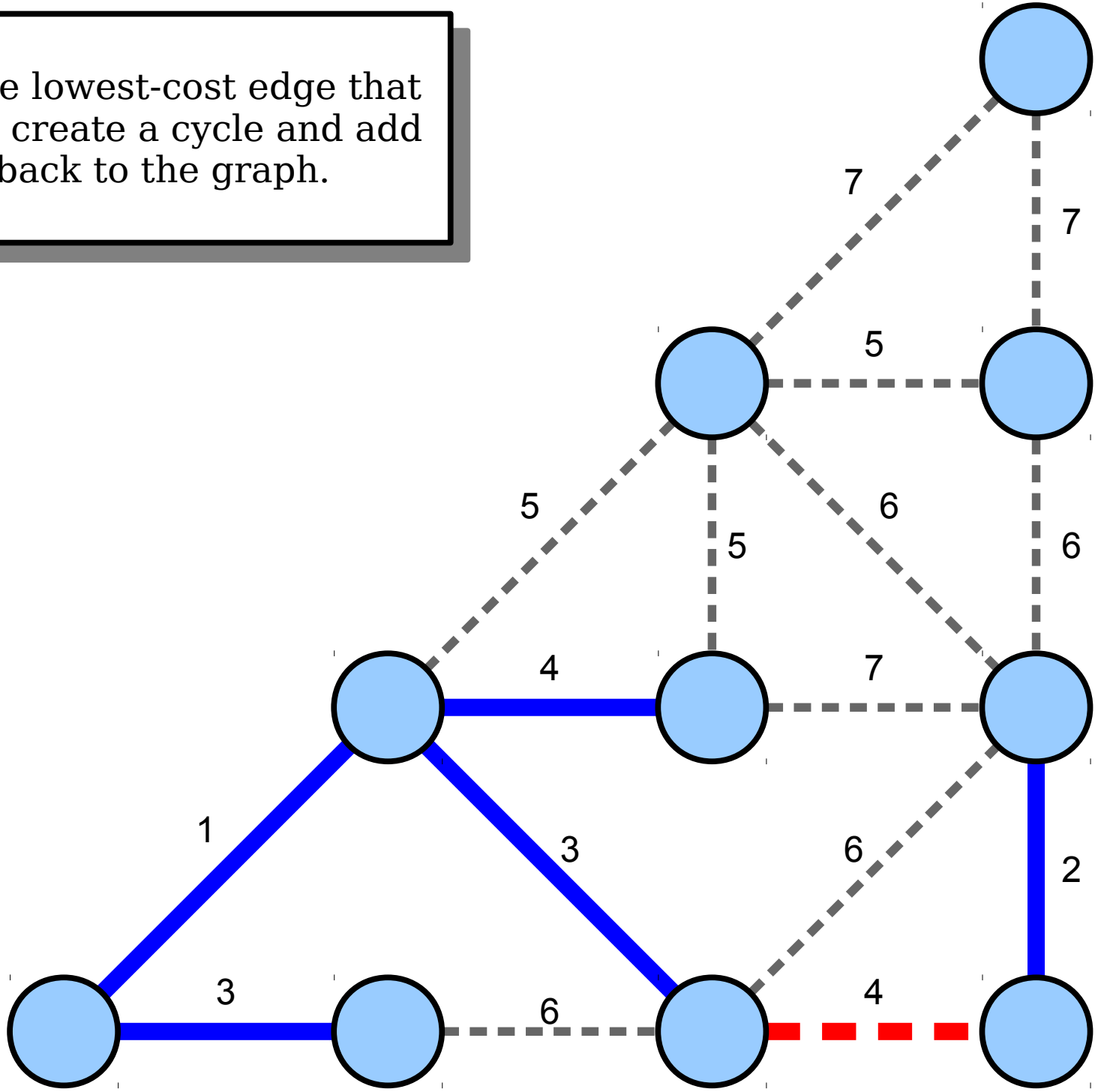
A graph can have many minimum spanning trees. Here, the choice of which length-4 edge we visit first leads to different results.



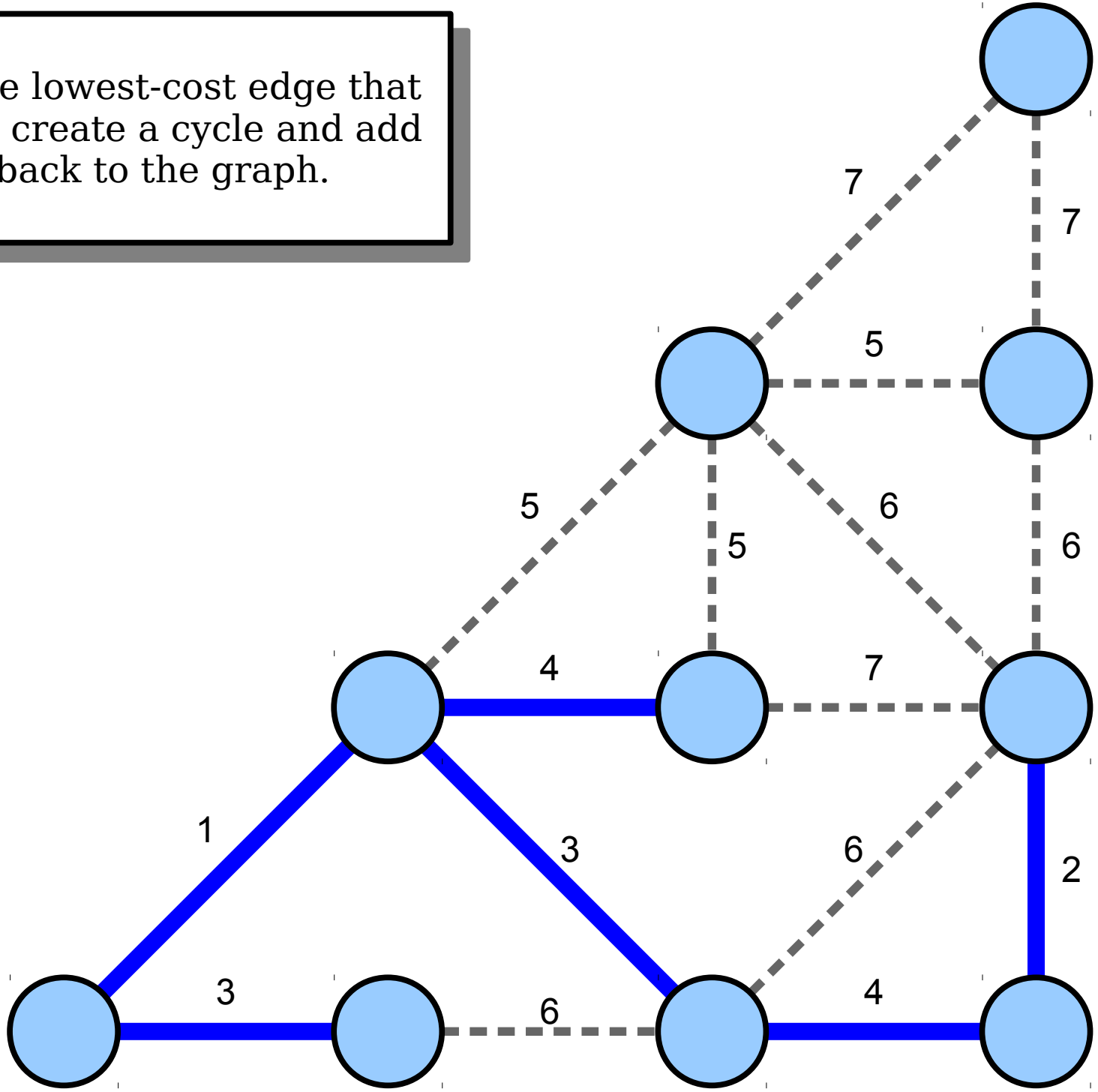
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



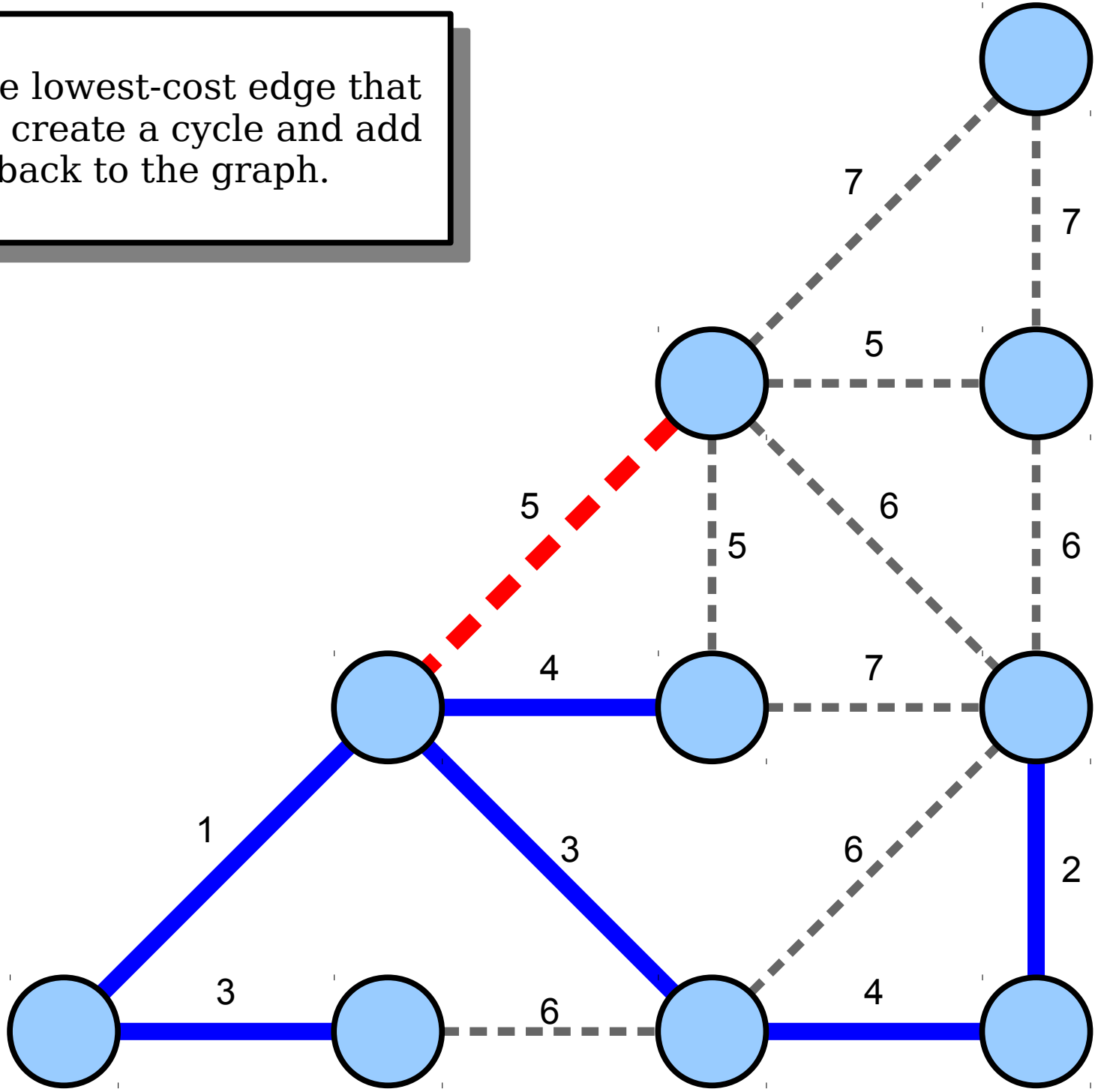
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



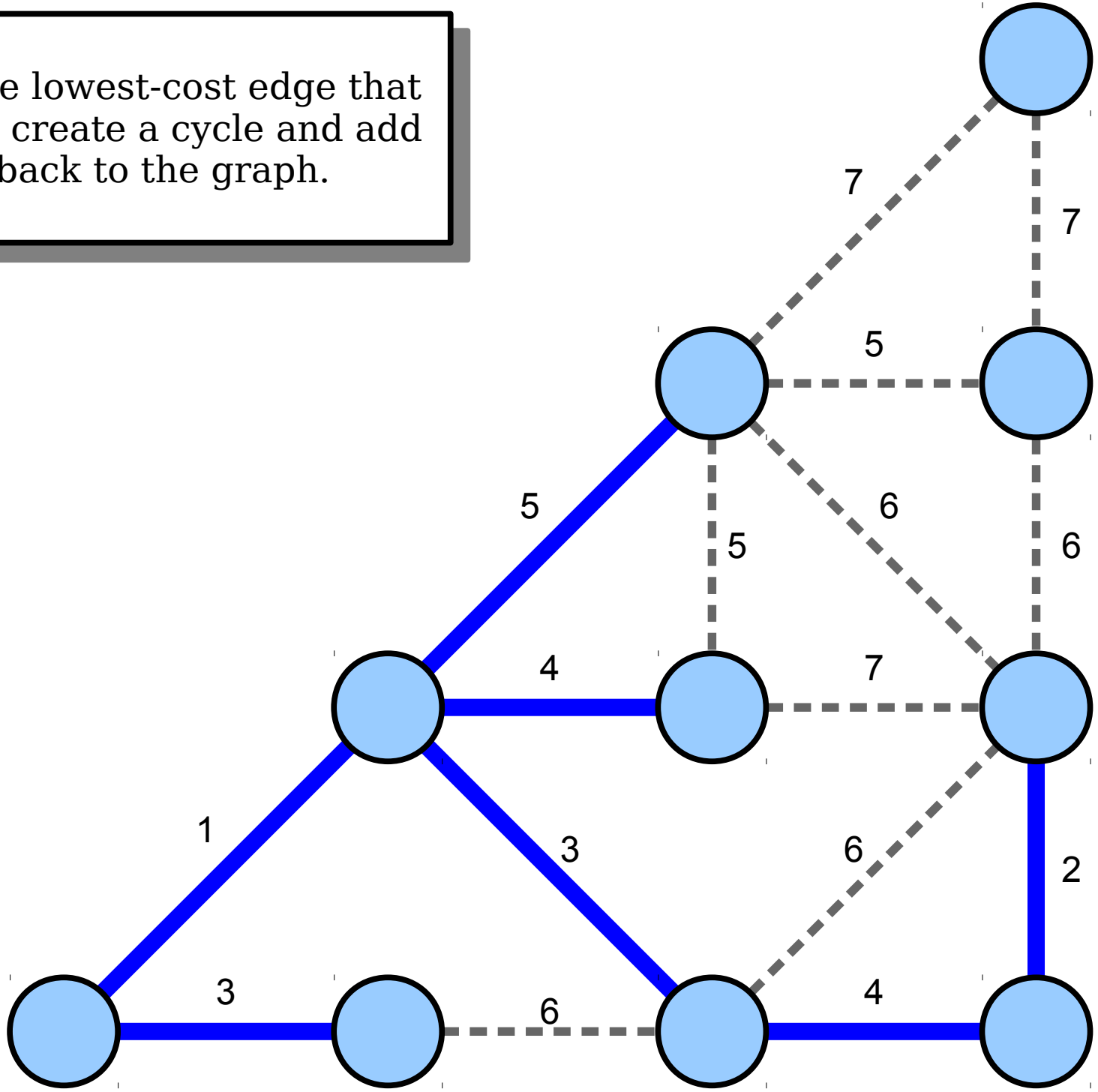
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



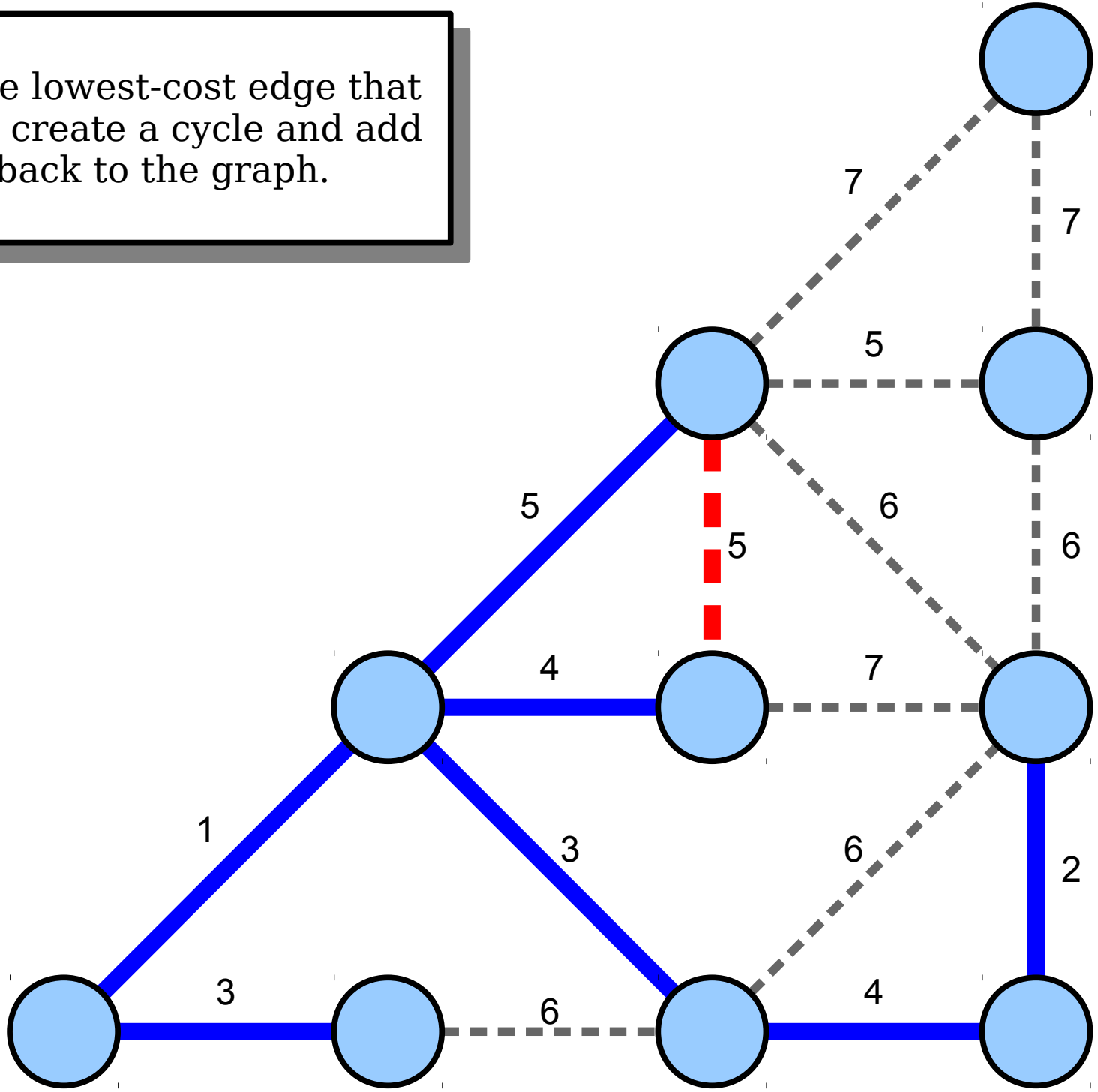
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



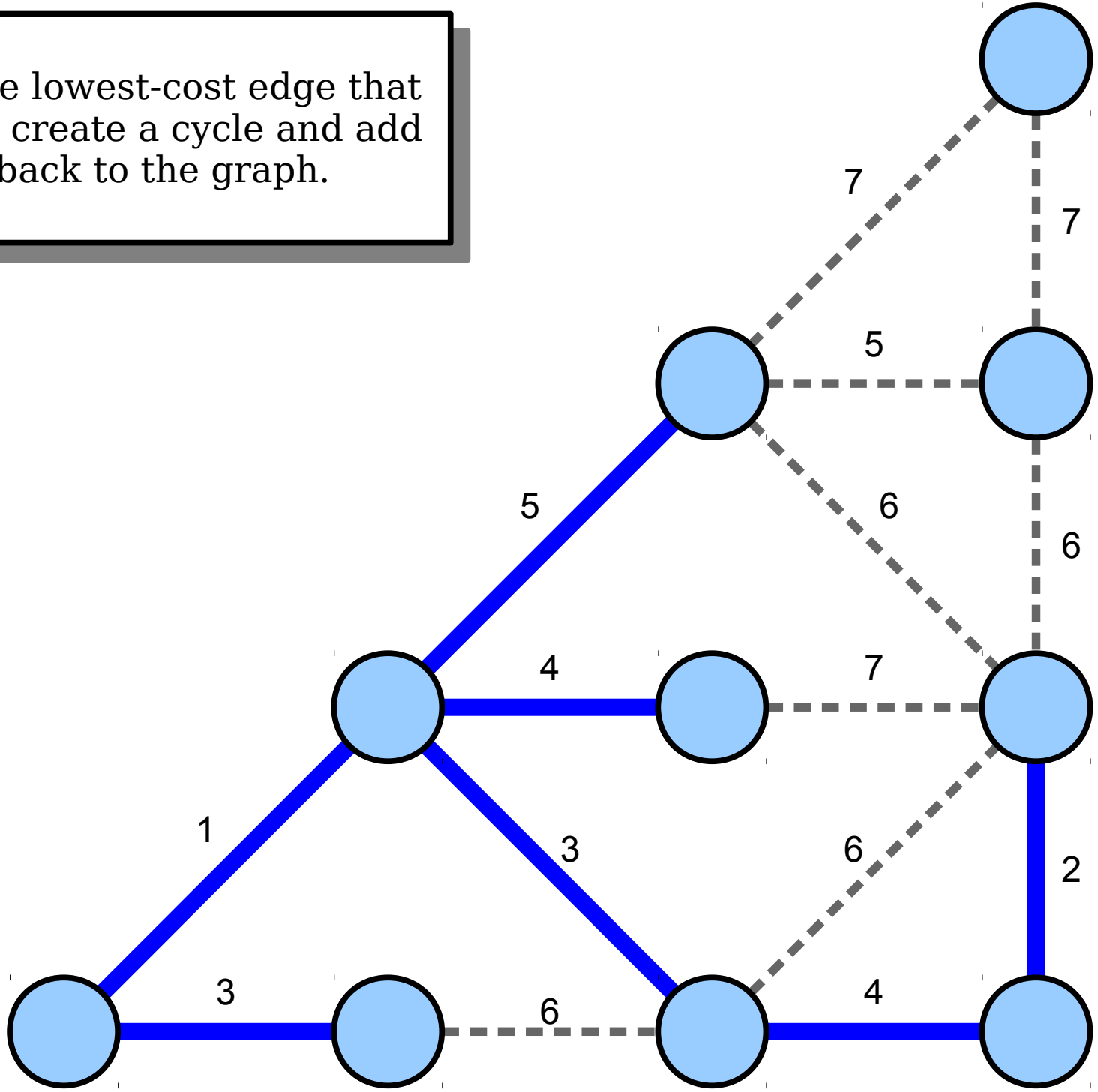
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



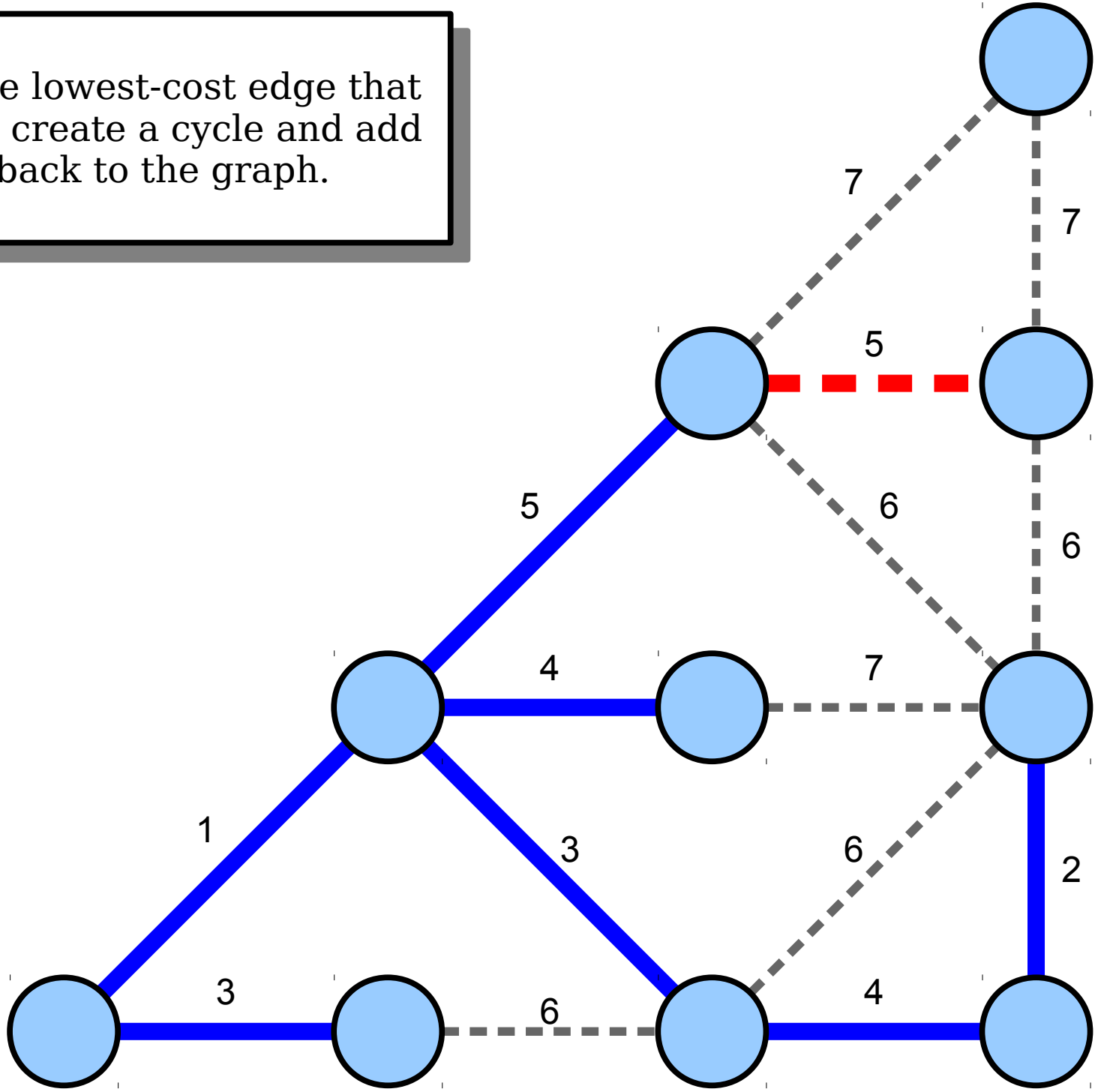
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



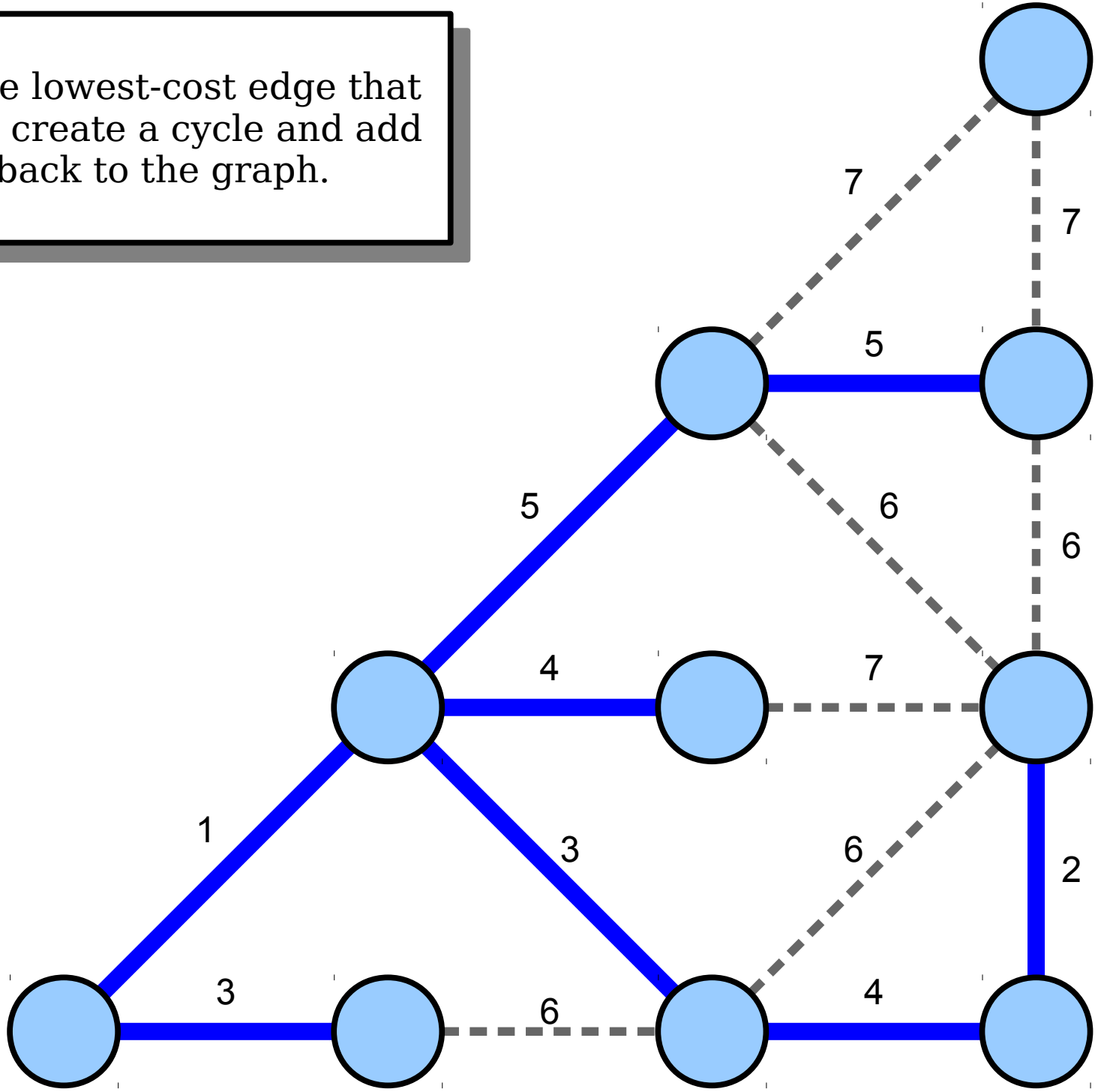
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



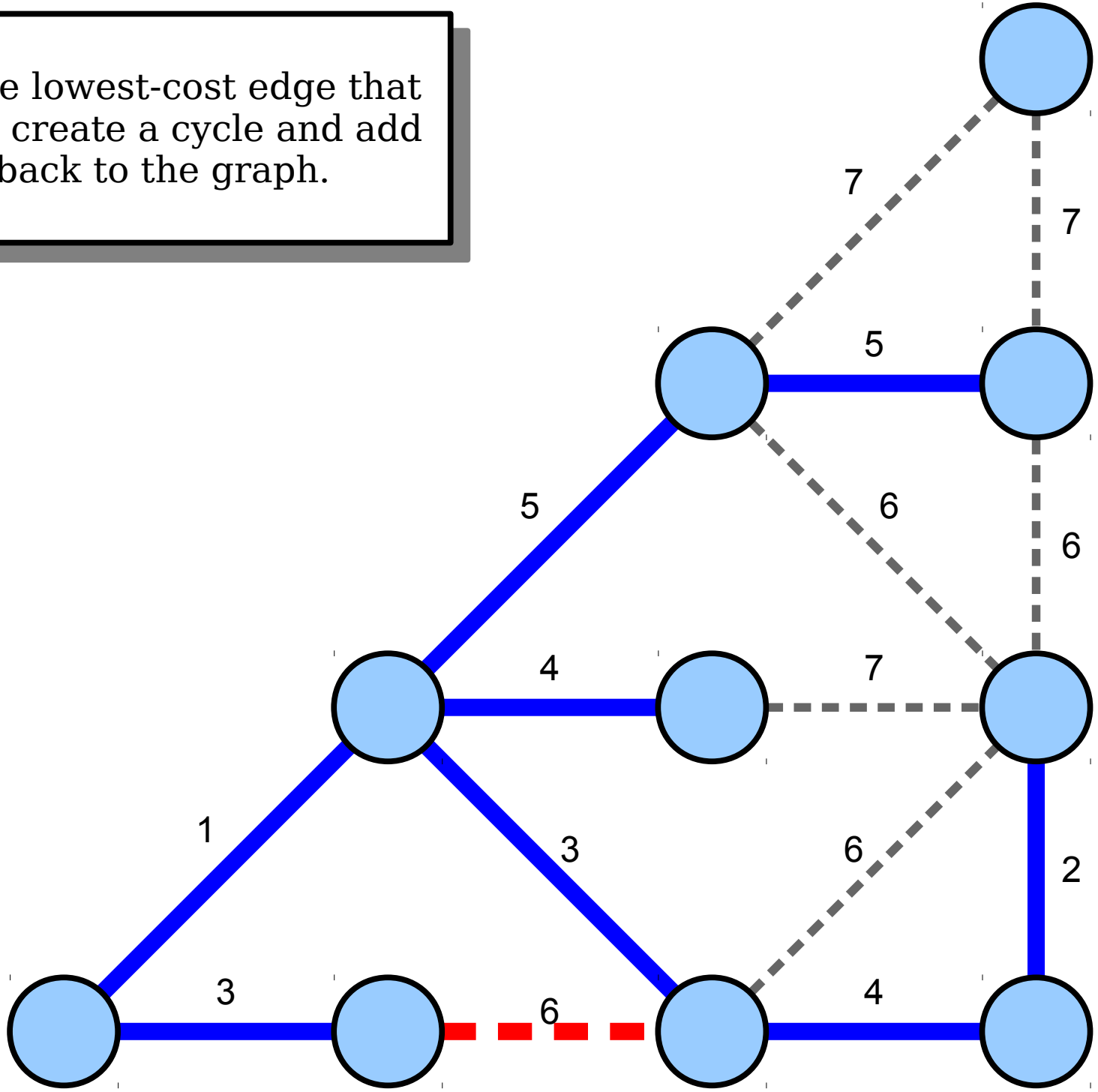
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



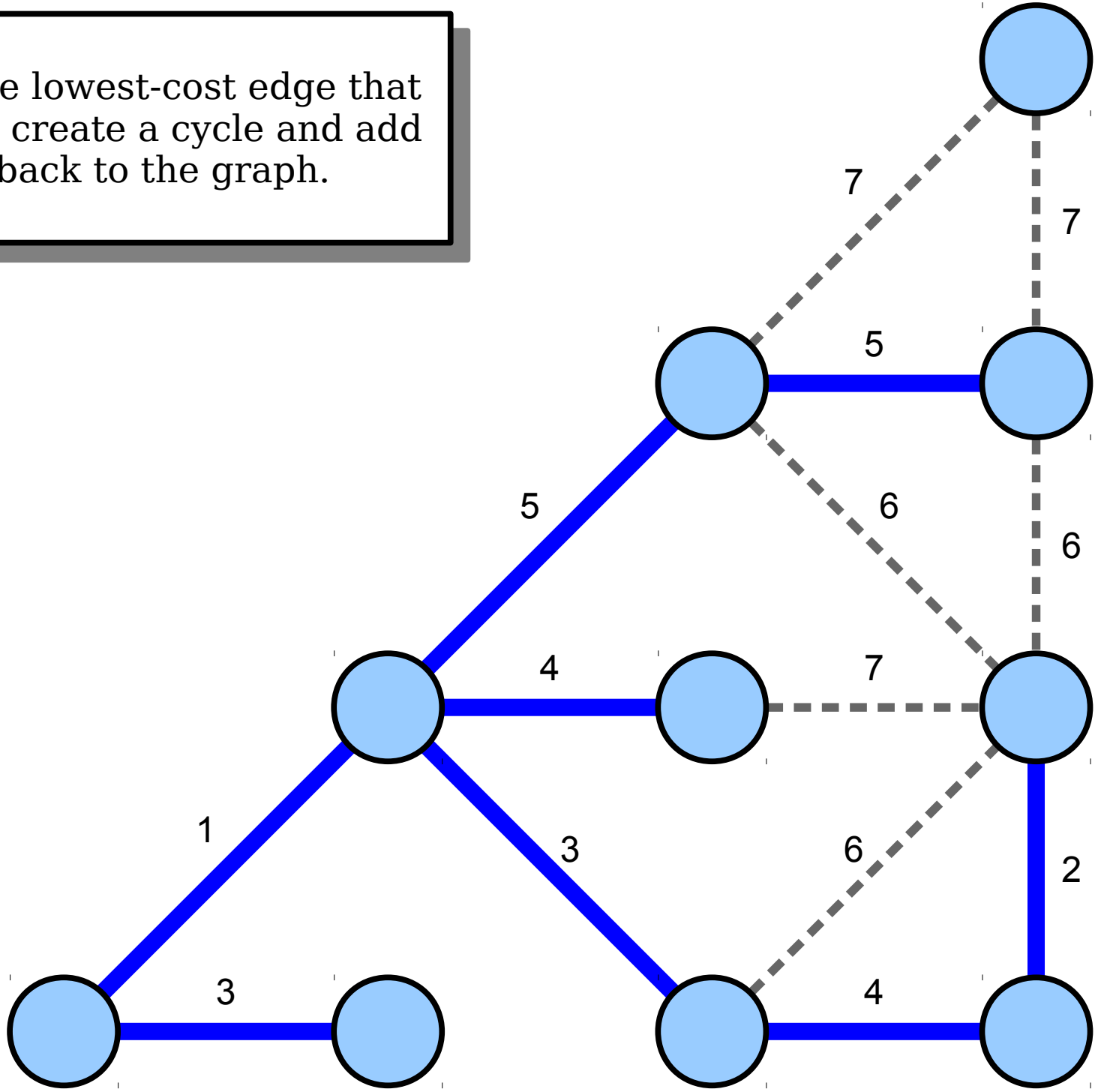
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



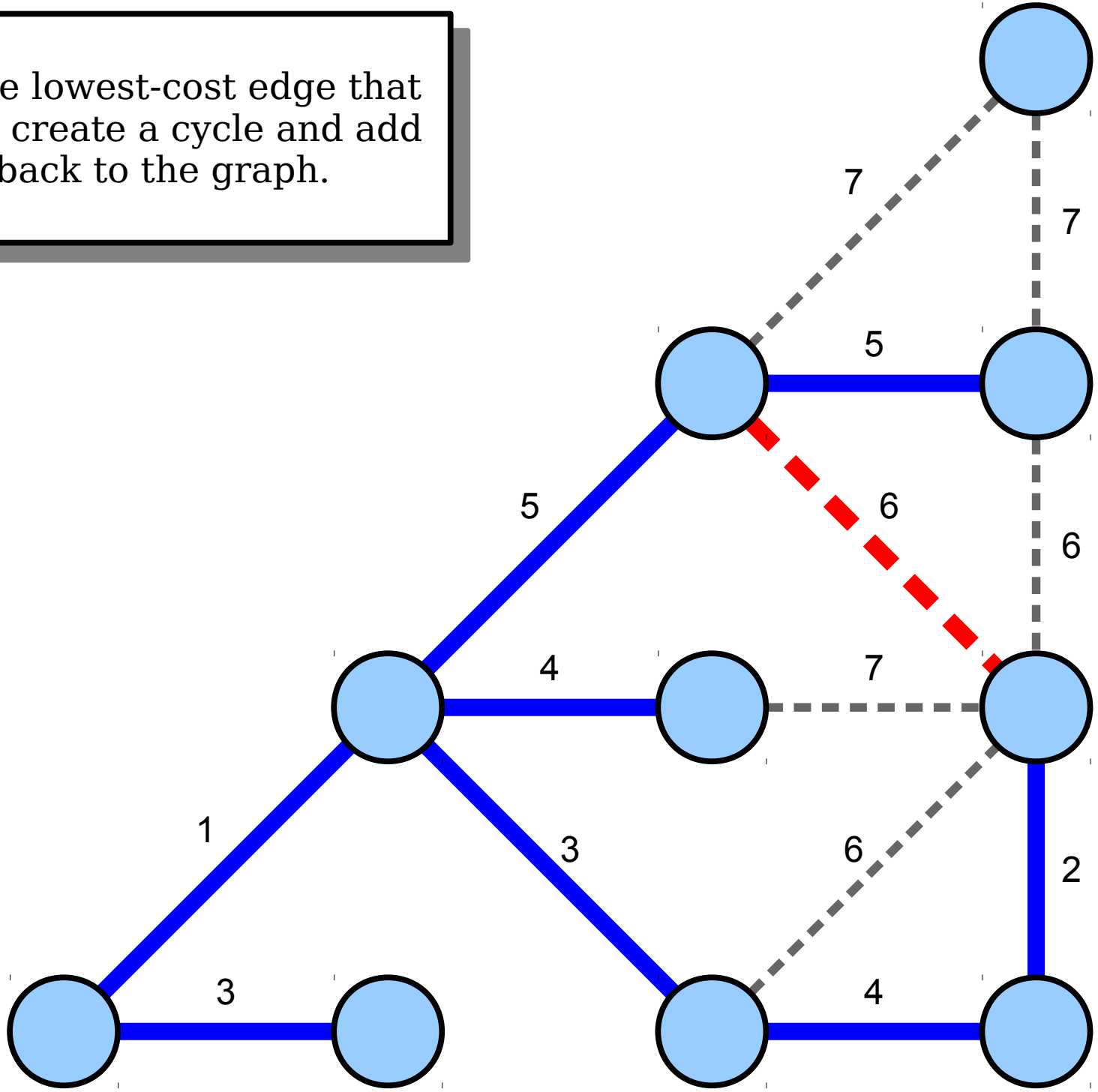
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



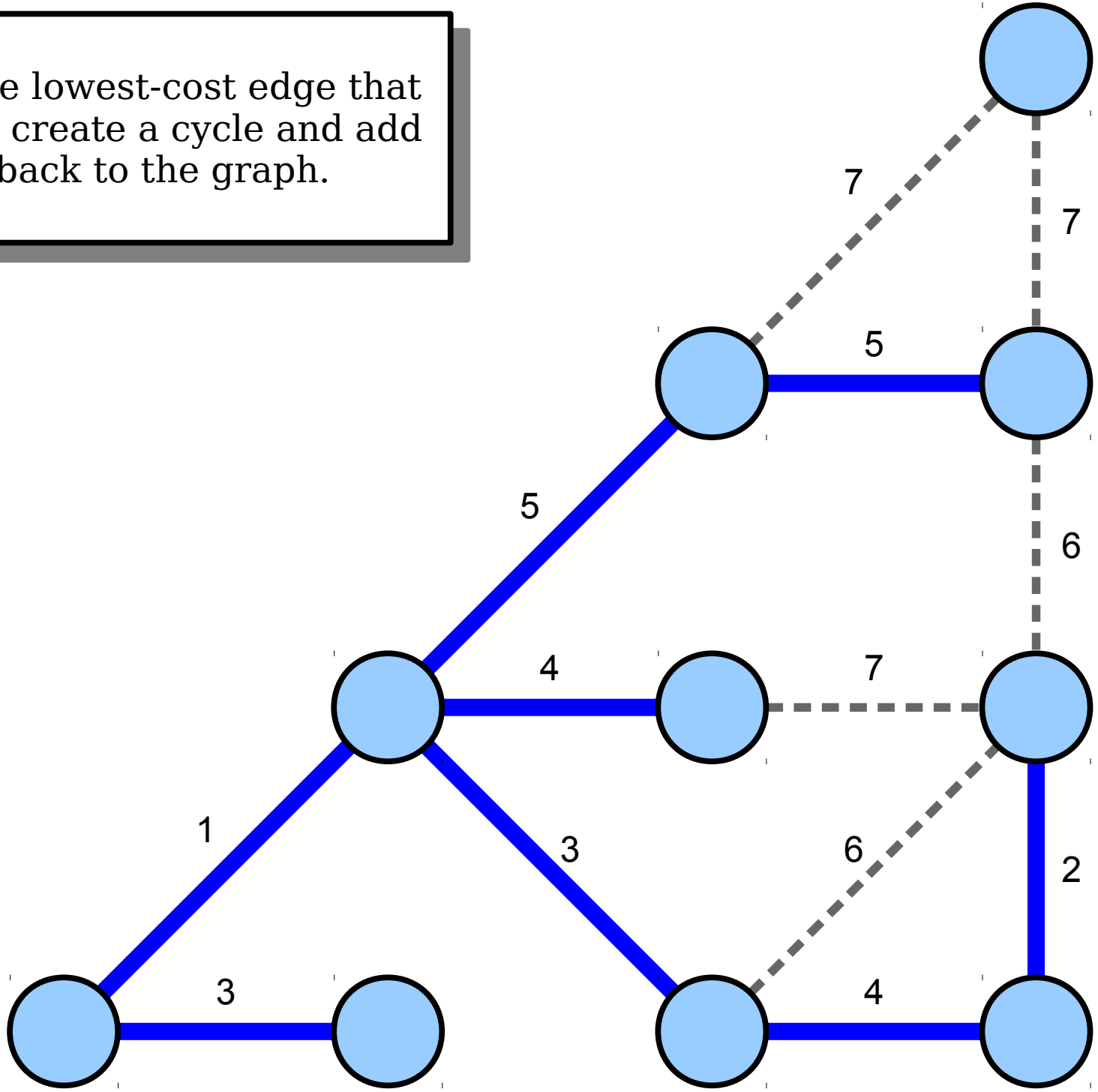
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



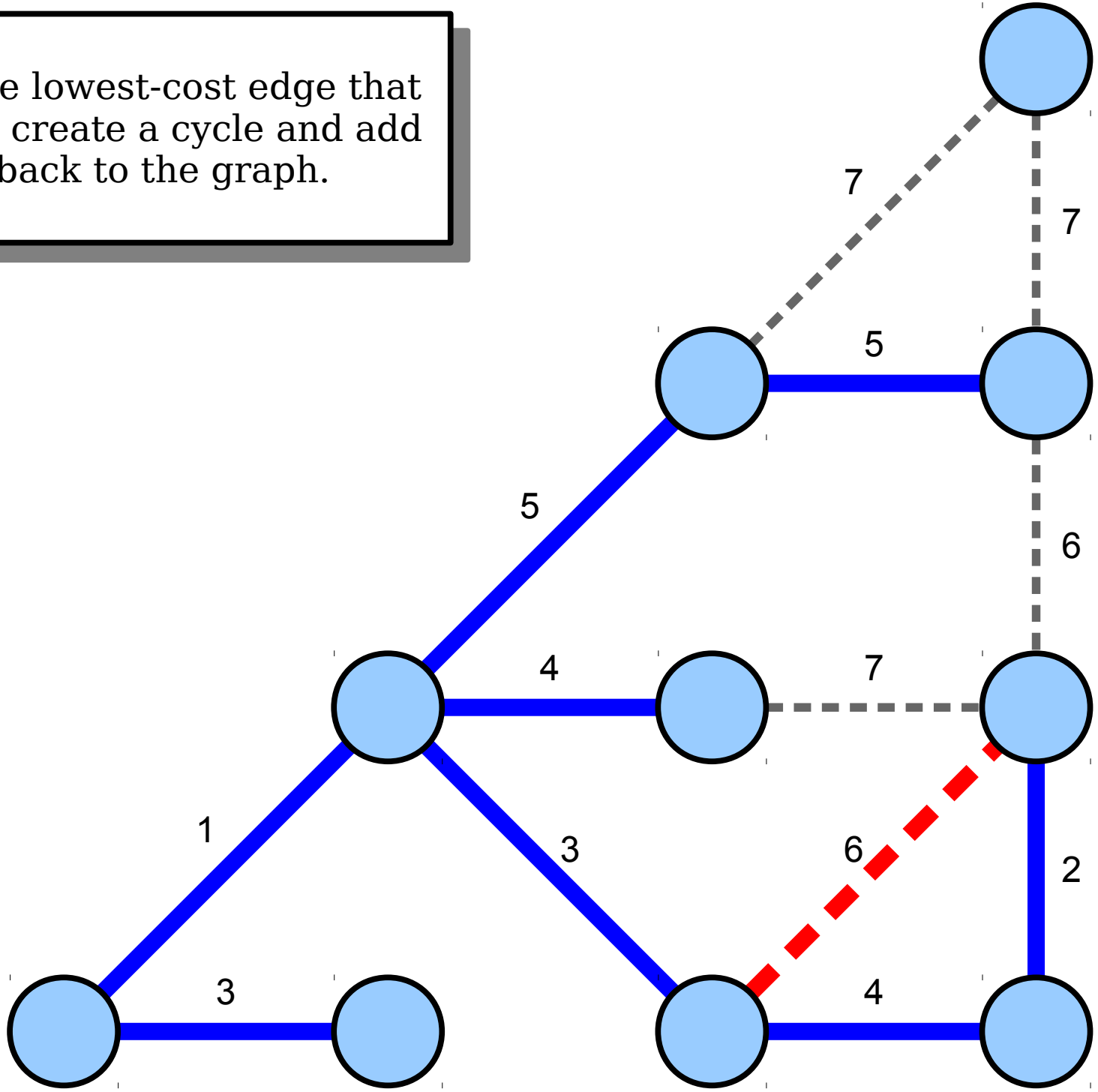
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



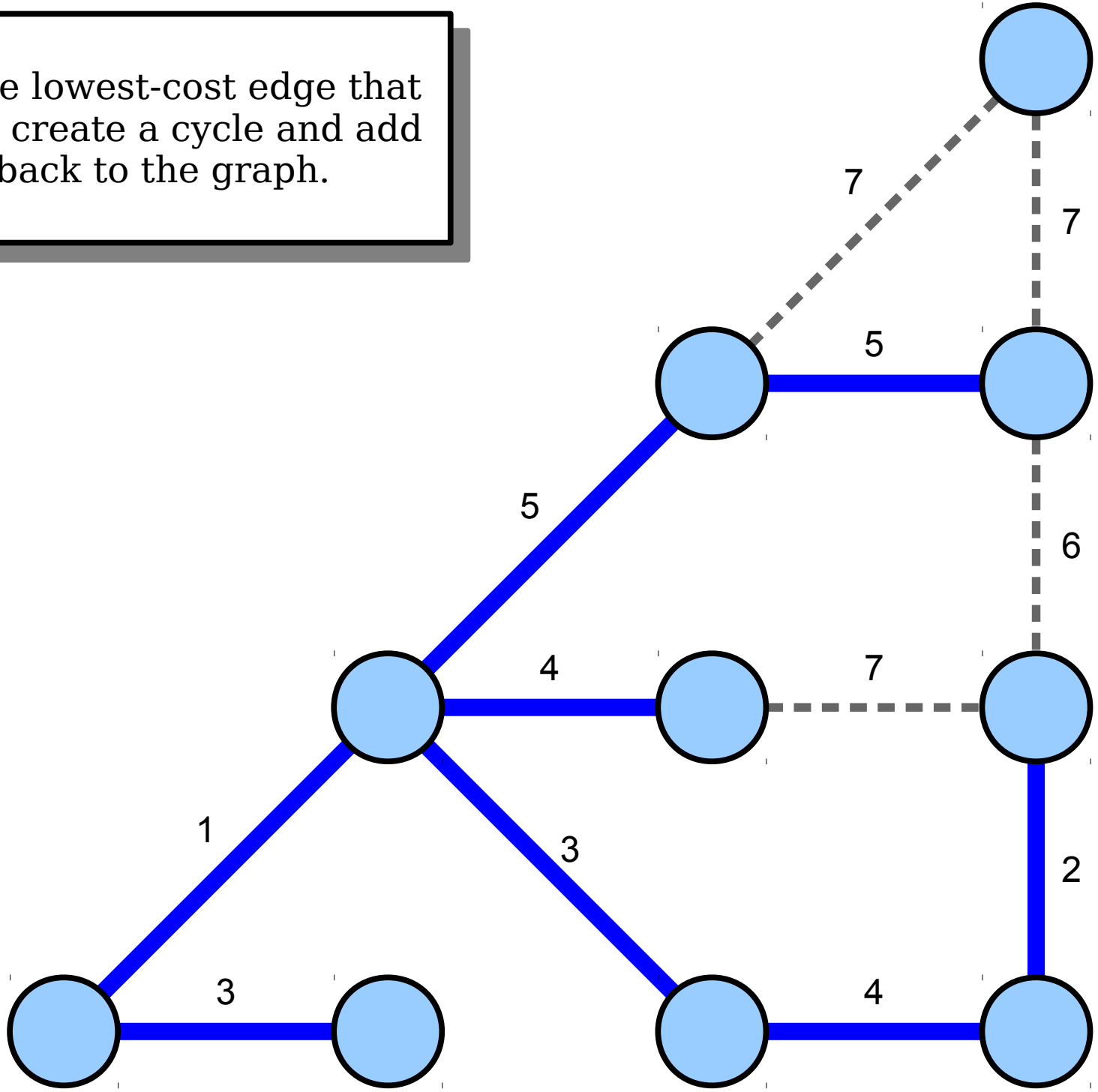
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



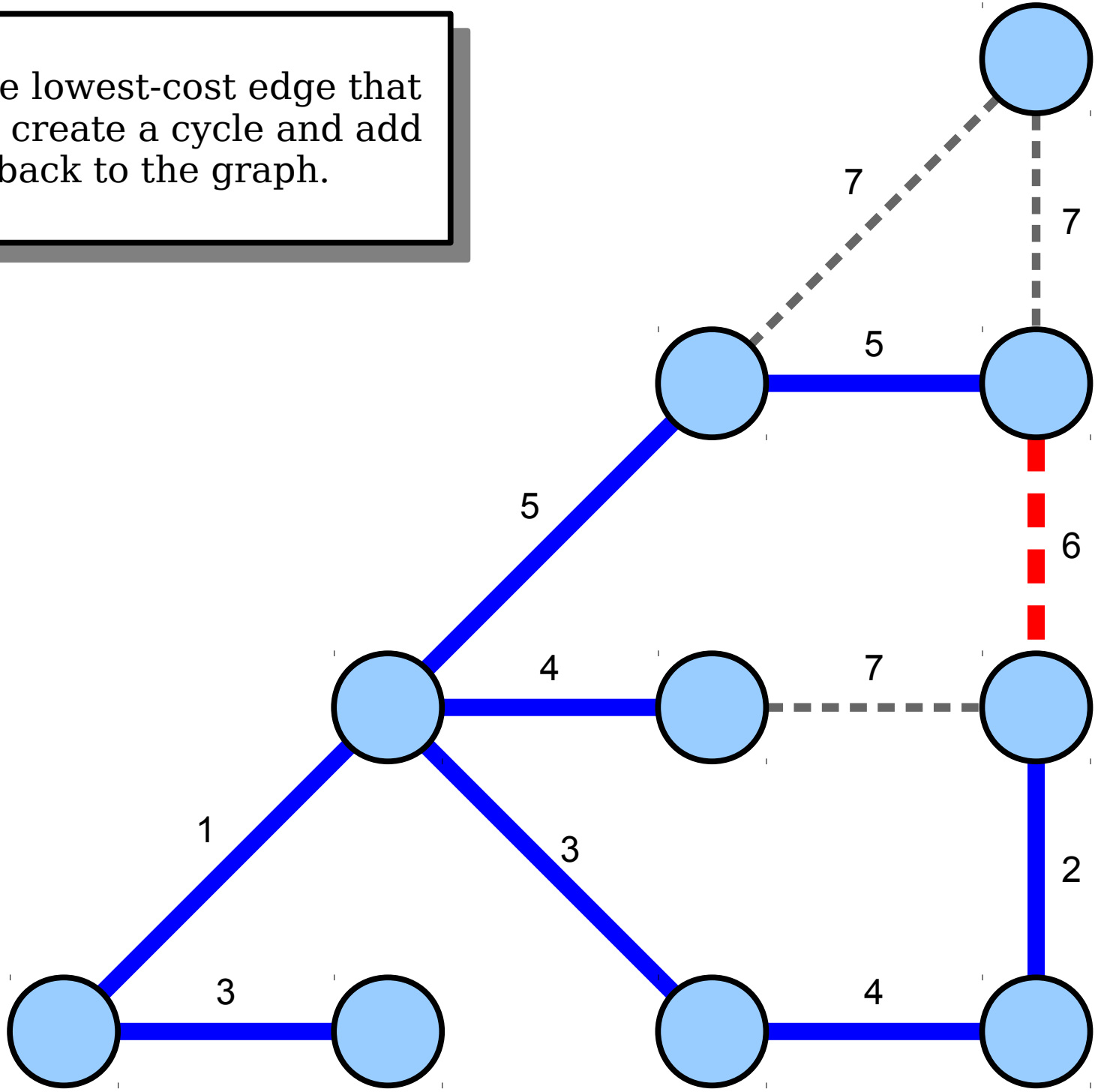
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



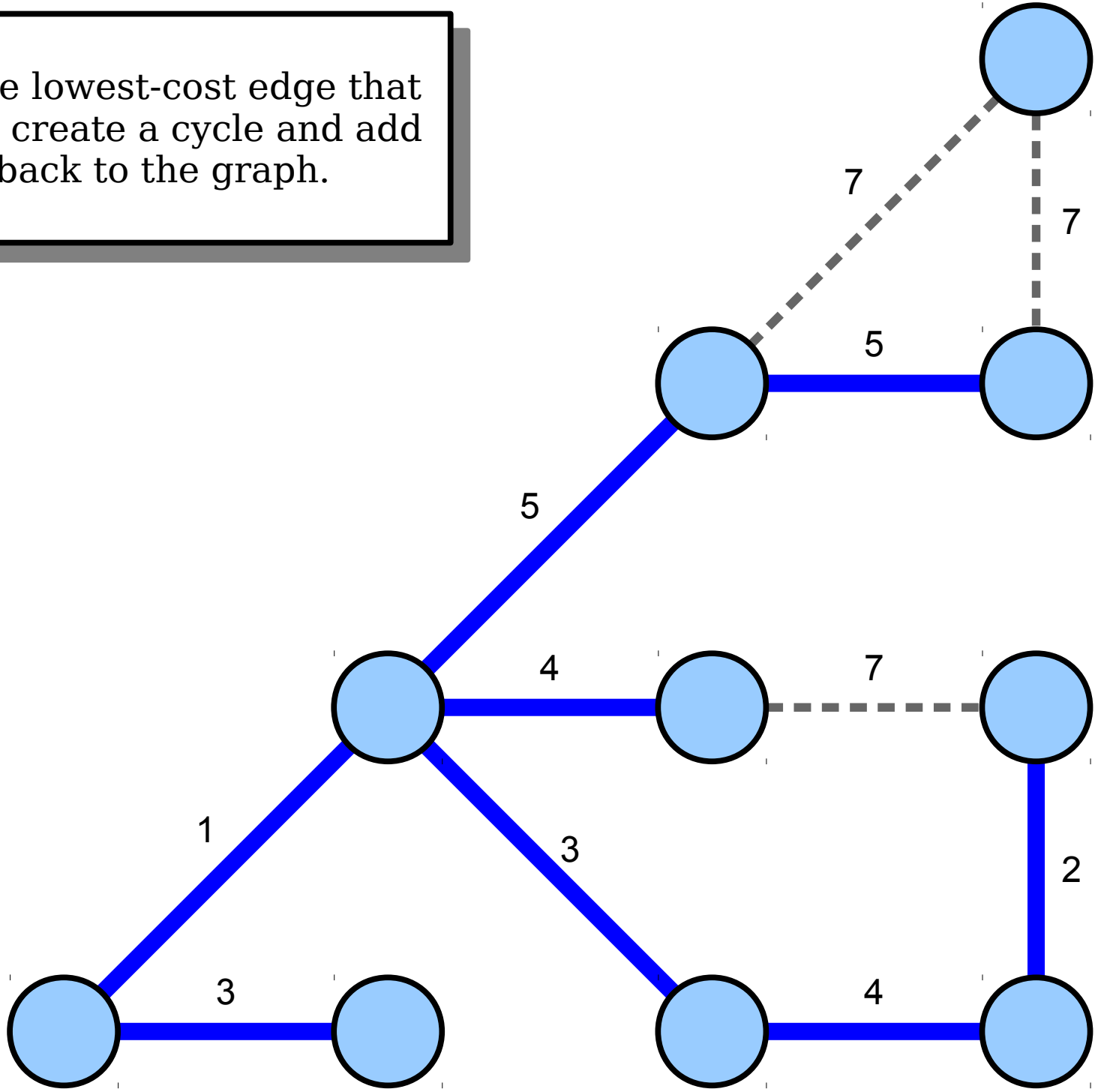
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



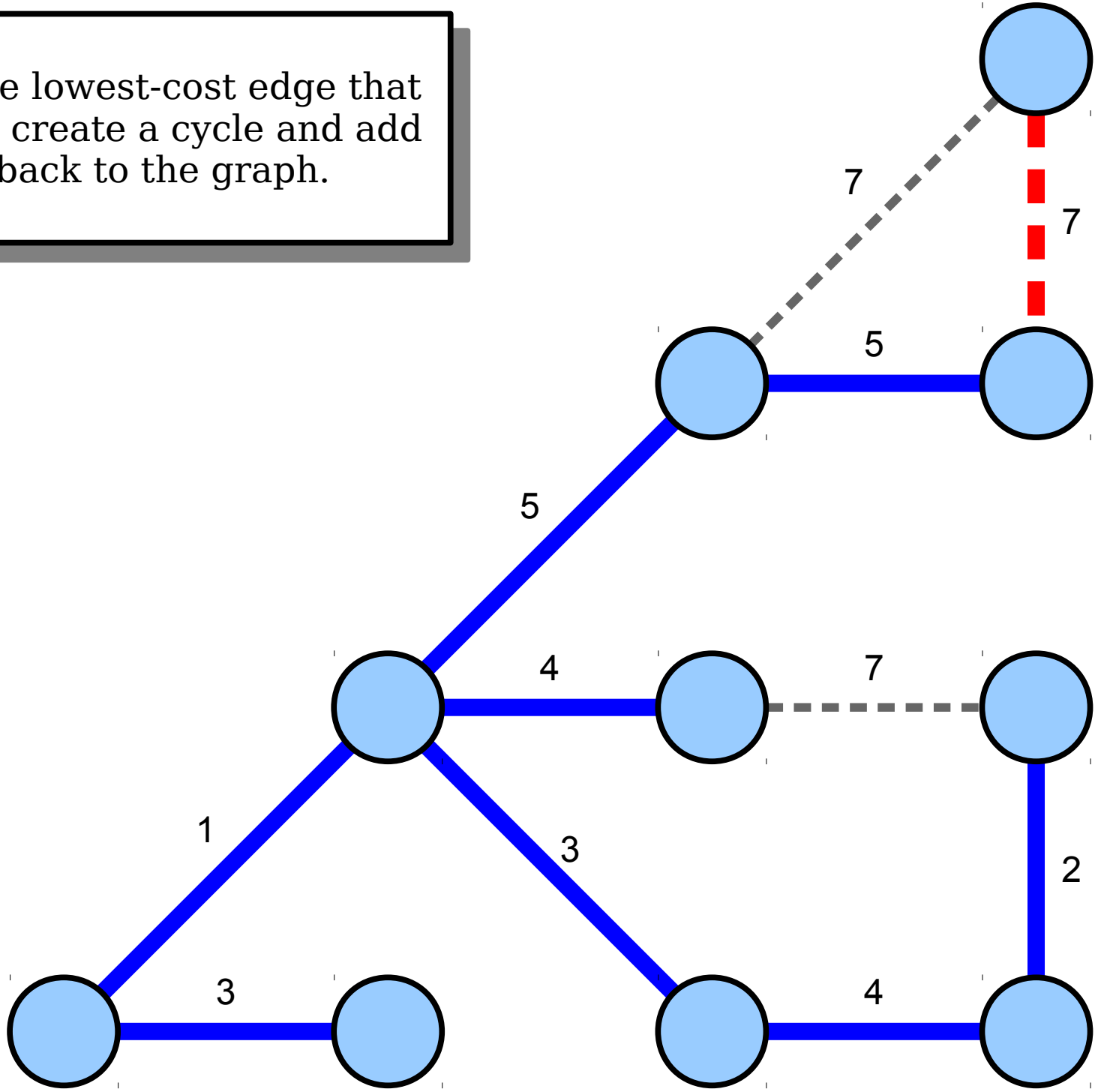
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



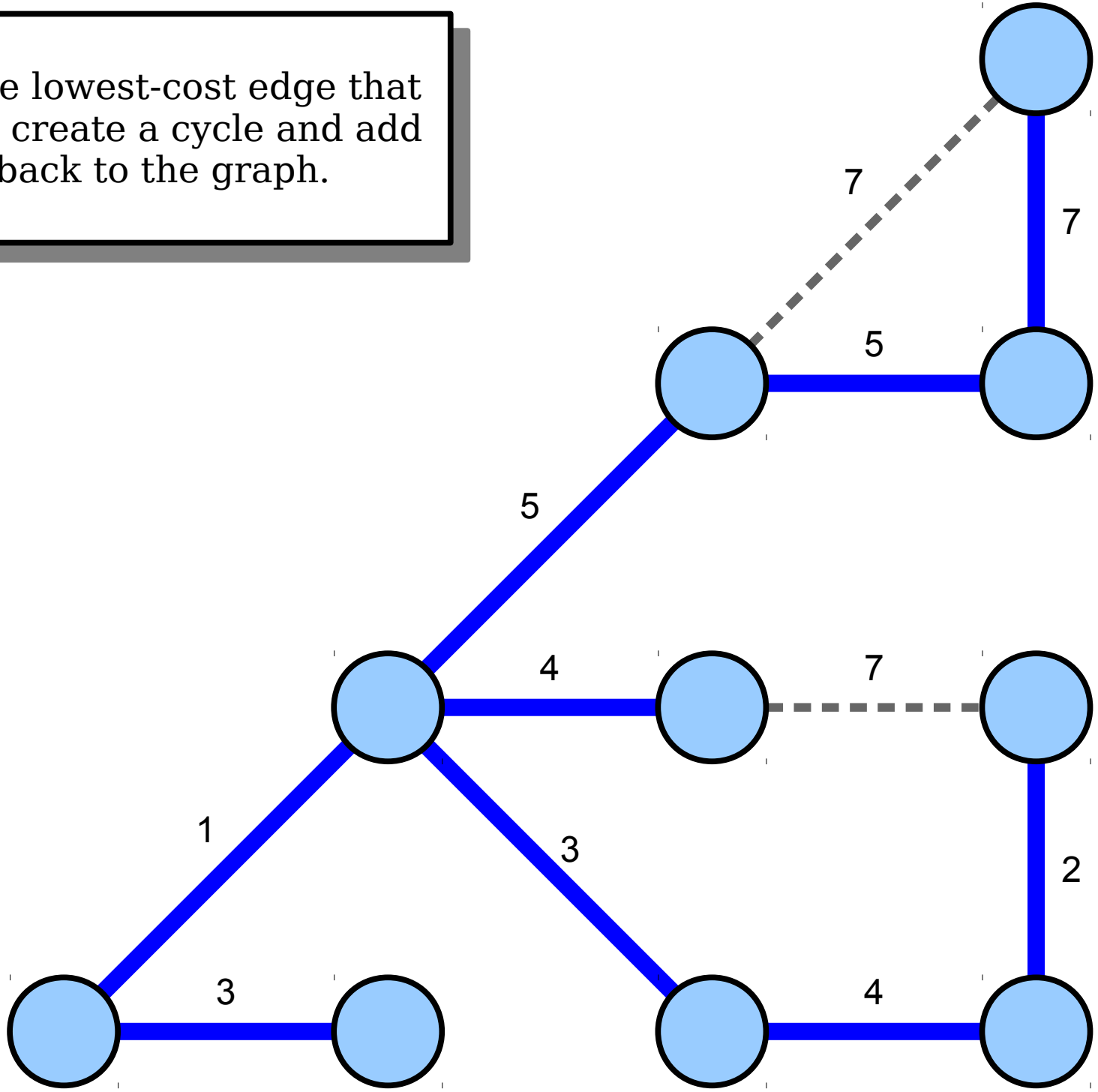
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



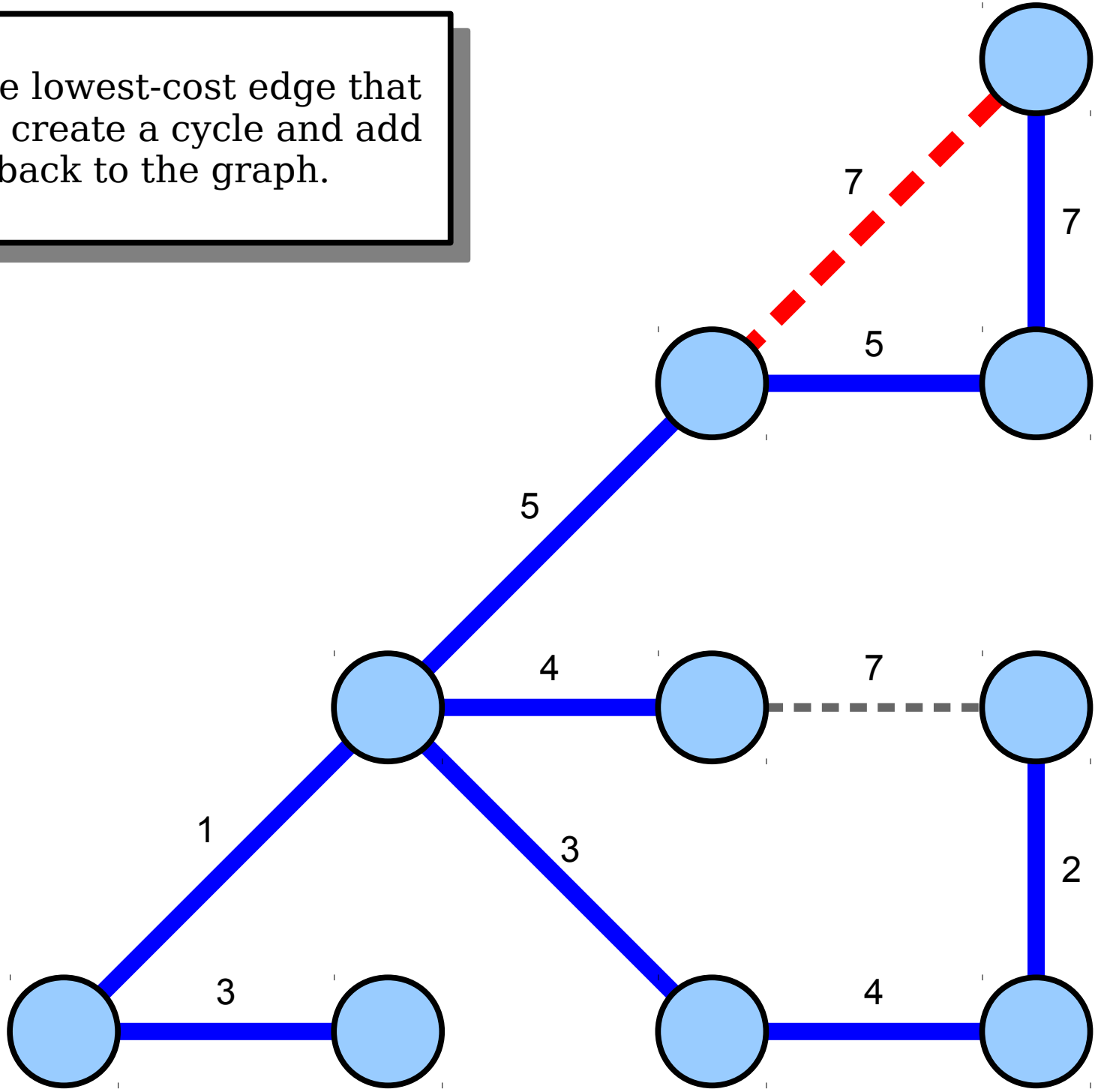
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



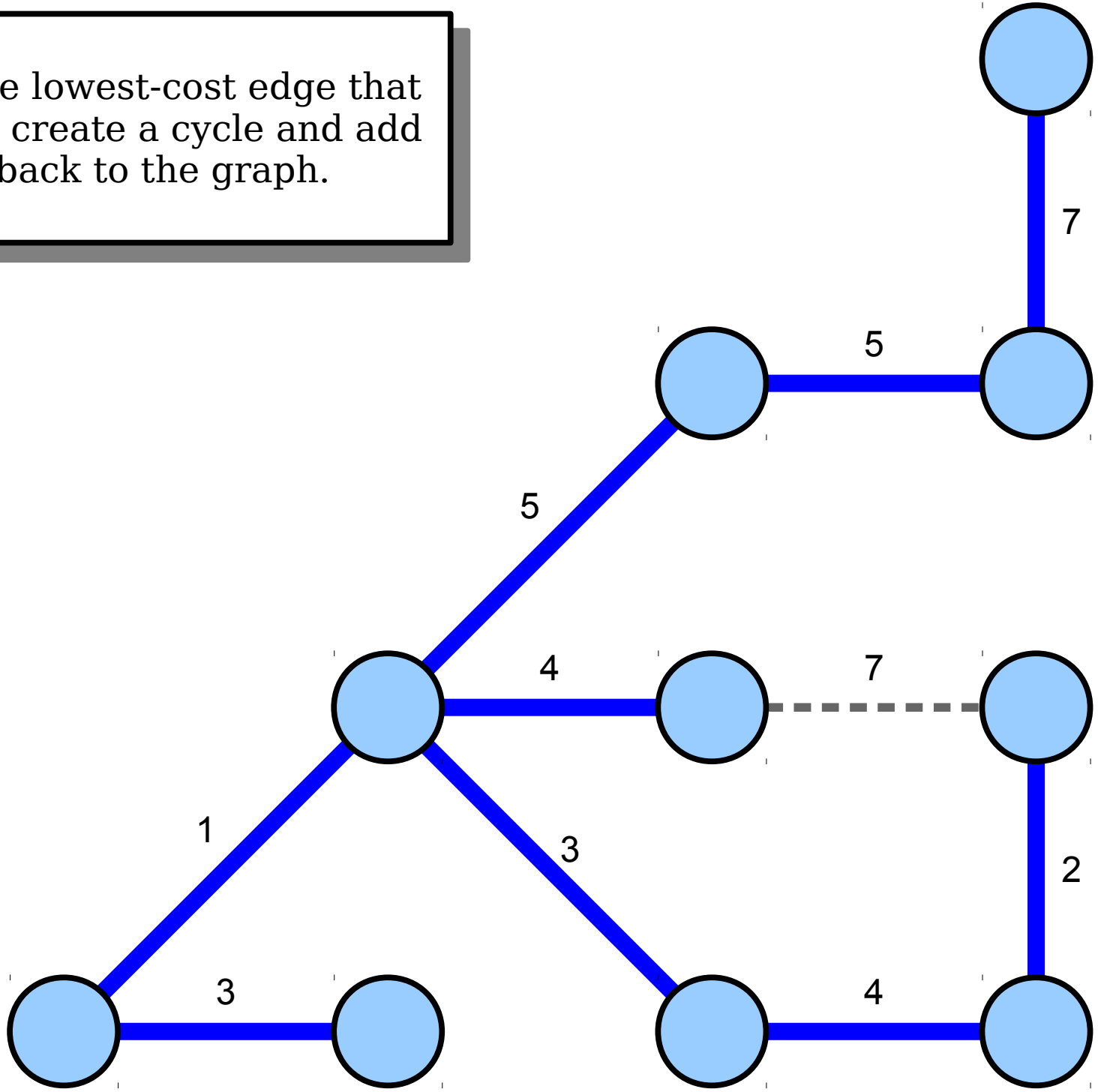
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



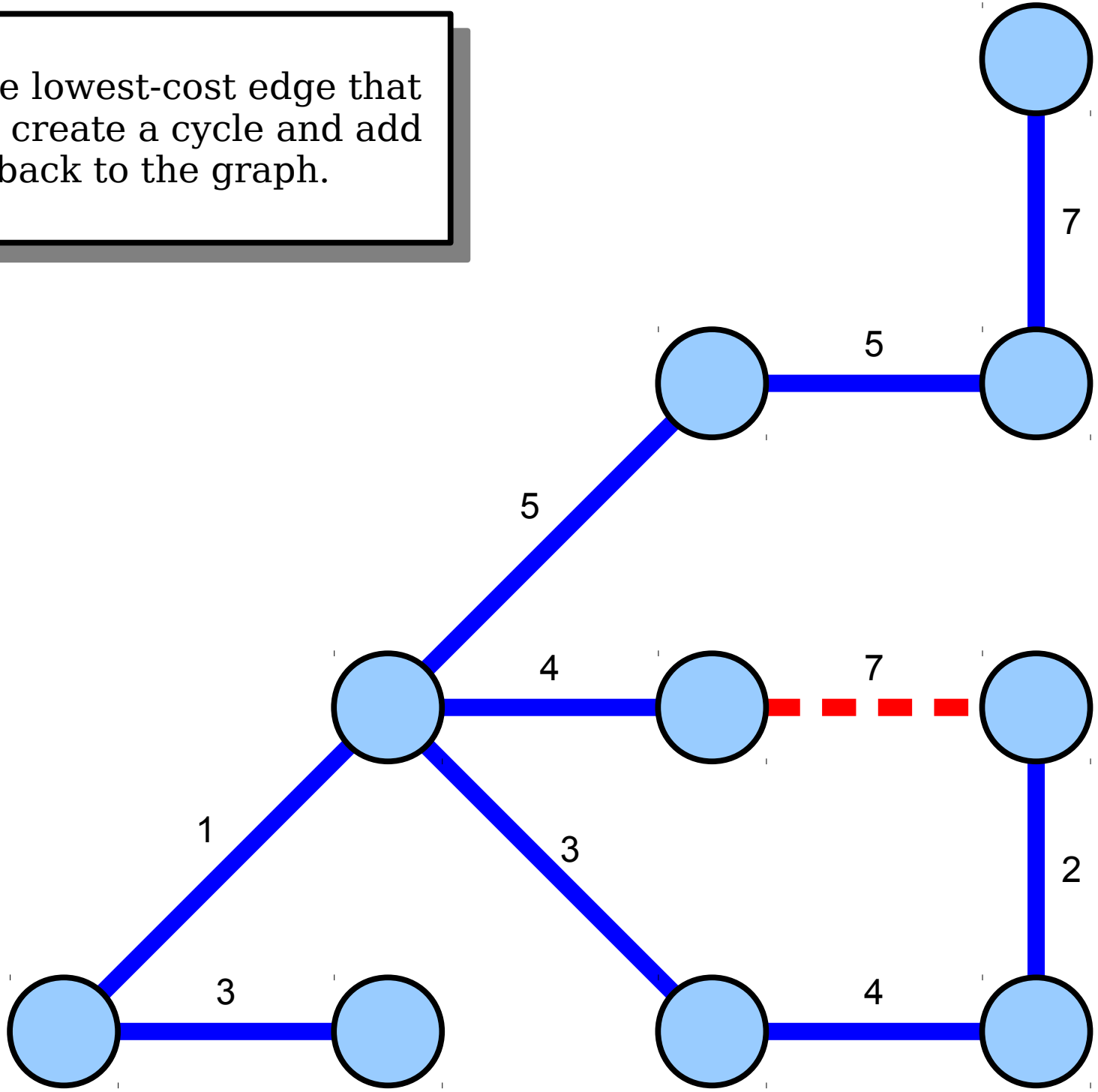
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



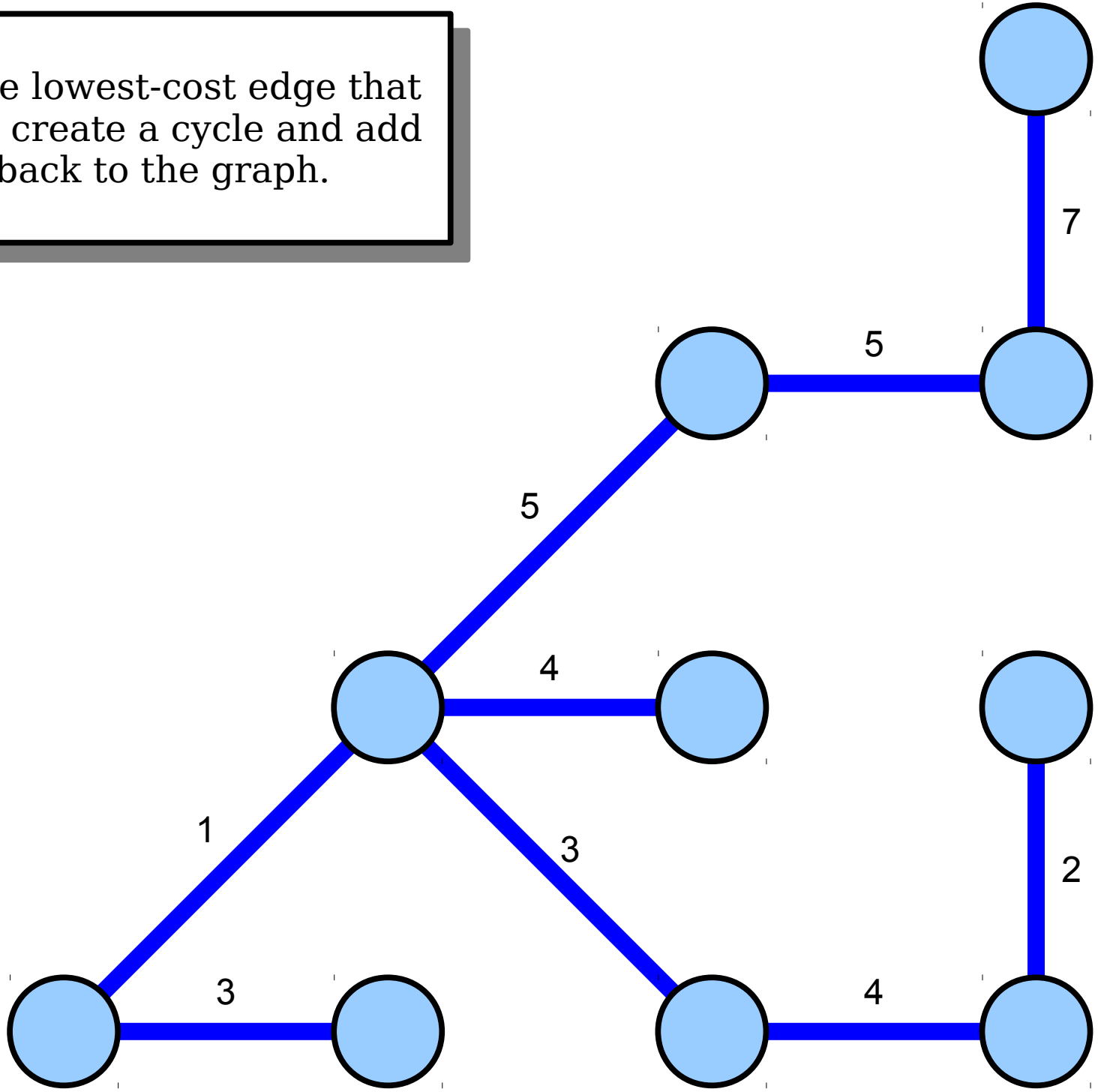
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



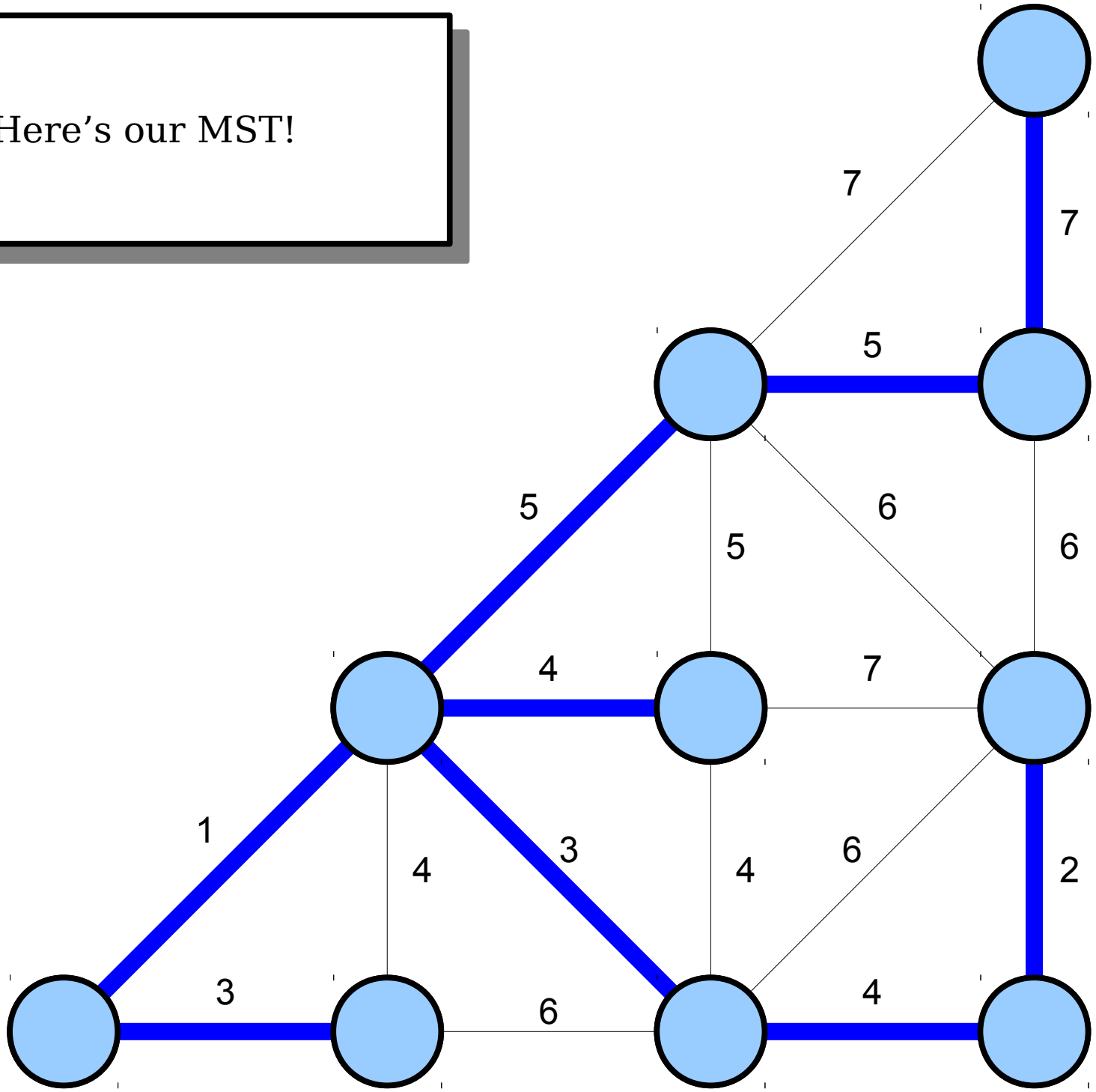
Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.



Find the lowest-cost edge that doesn't create a cycle and add it back to the graph.

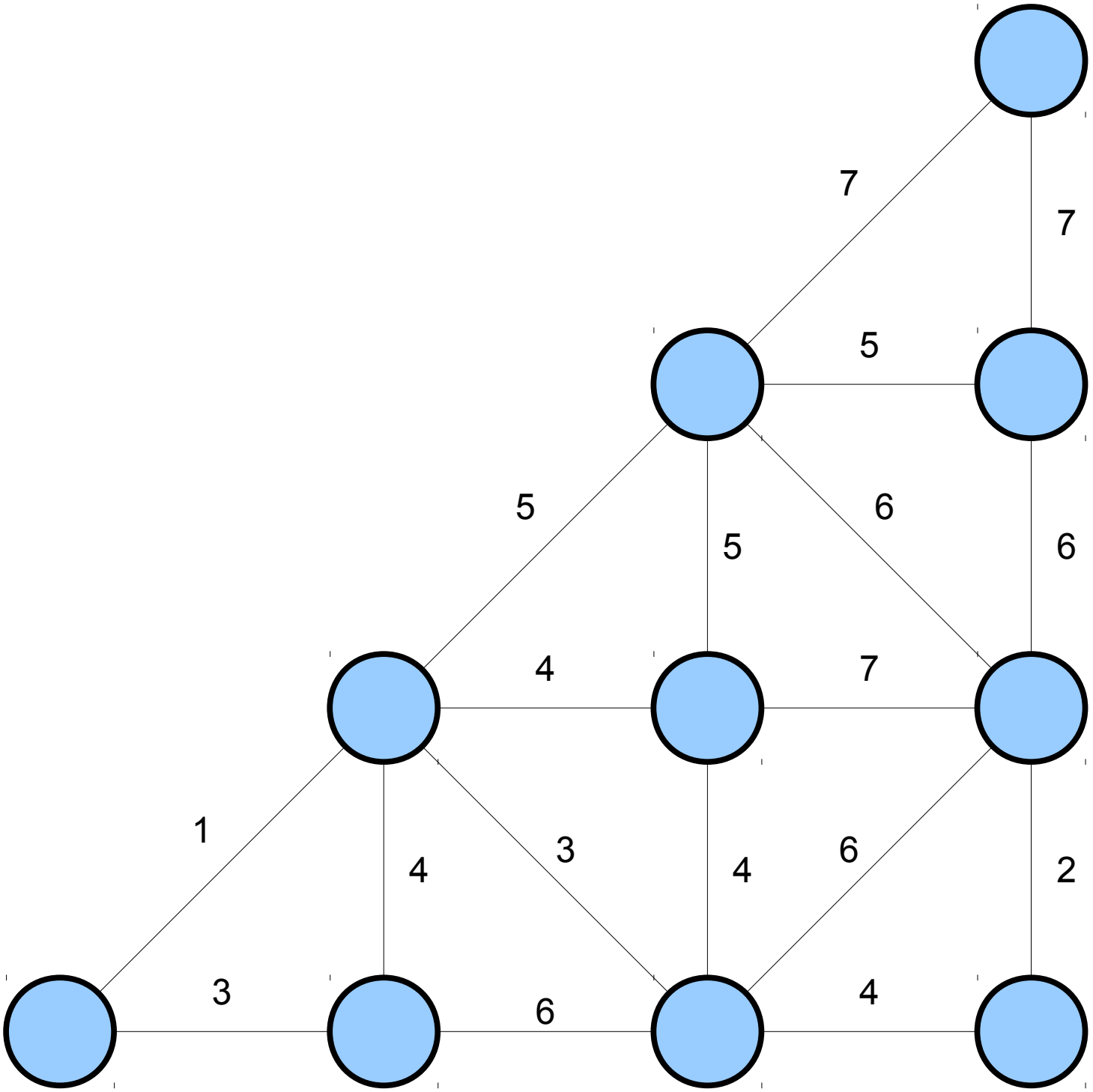


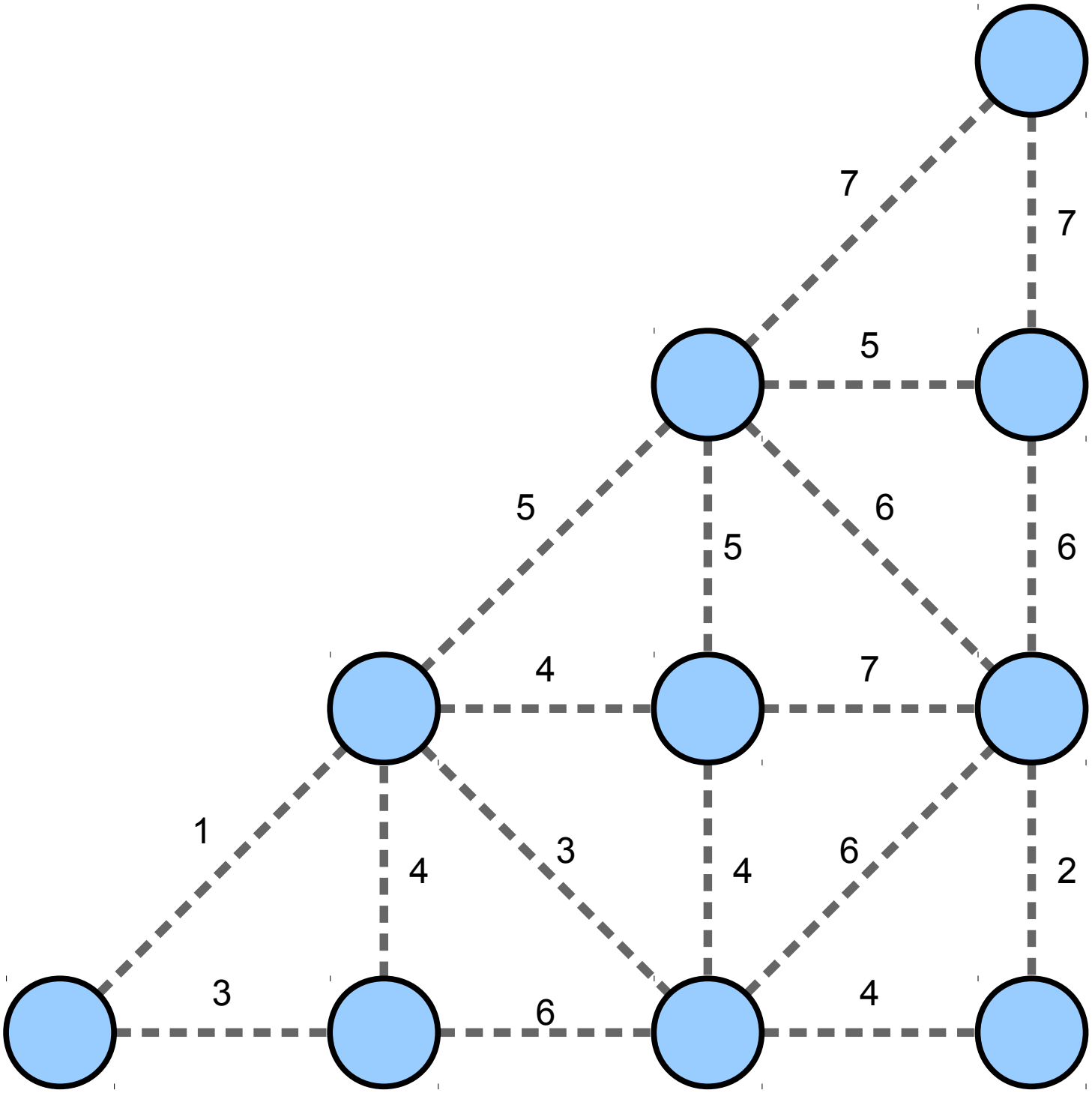
Here’s our MST!

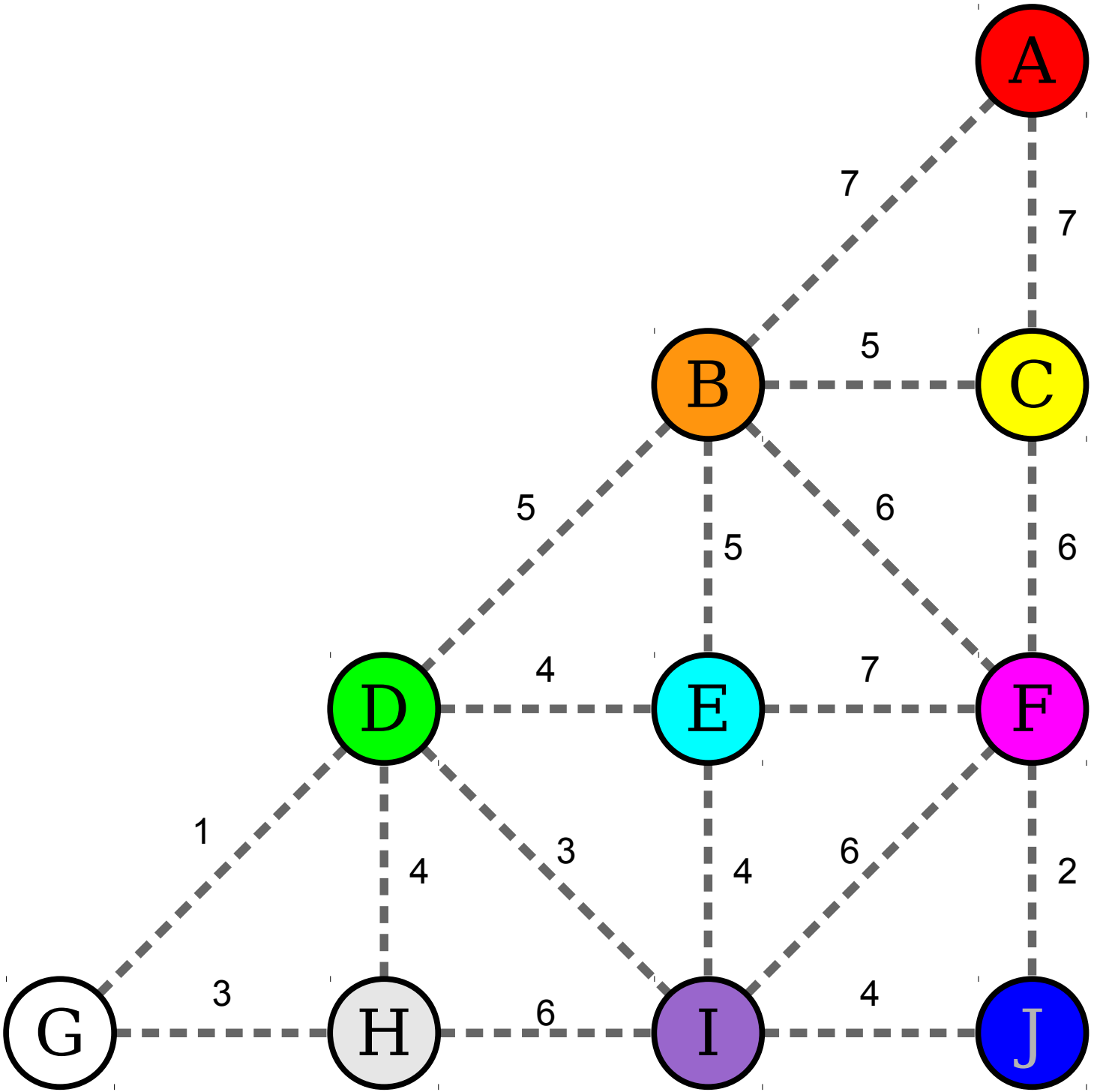


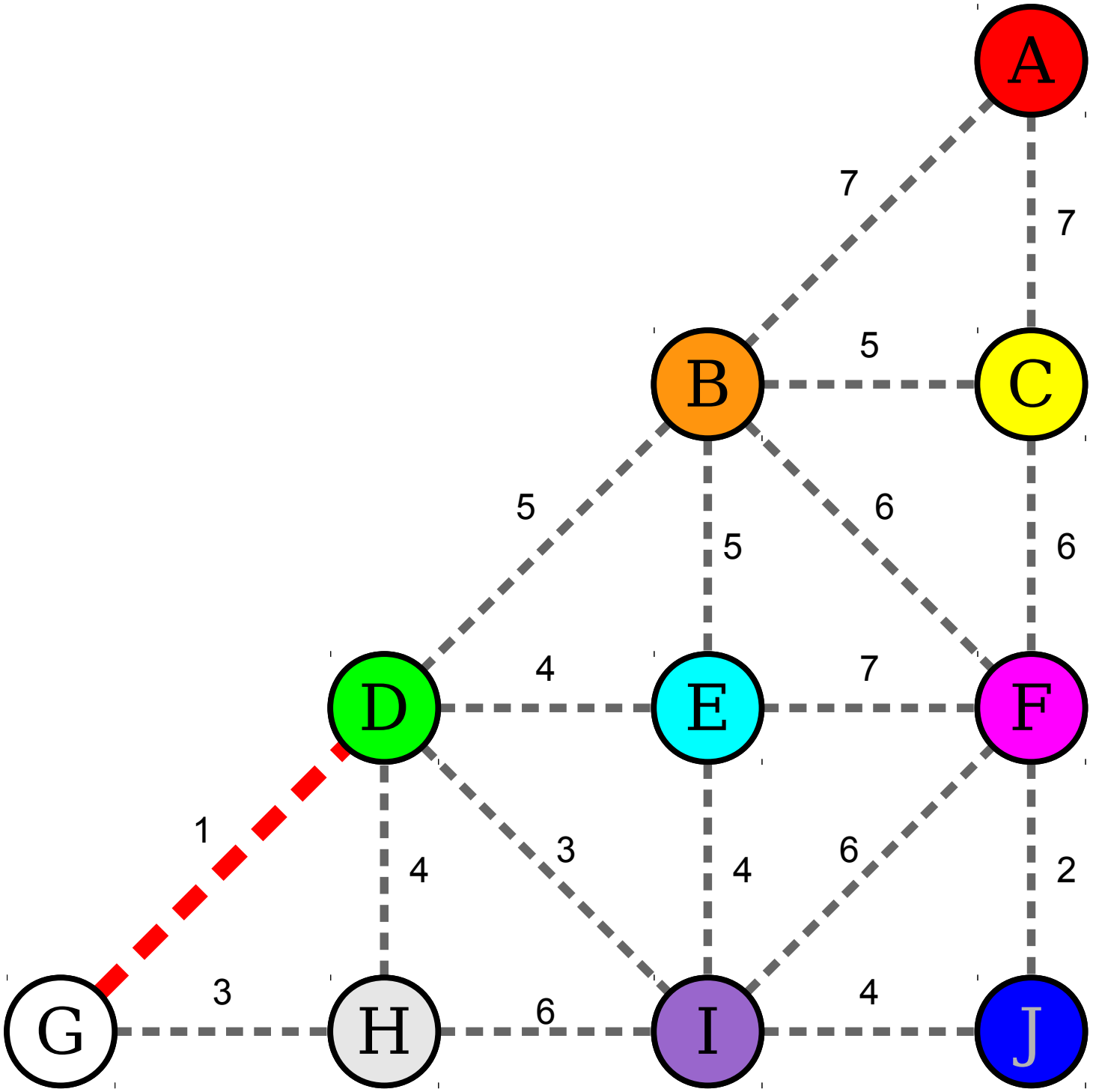
Maintaining Connectivity

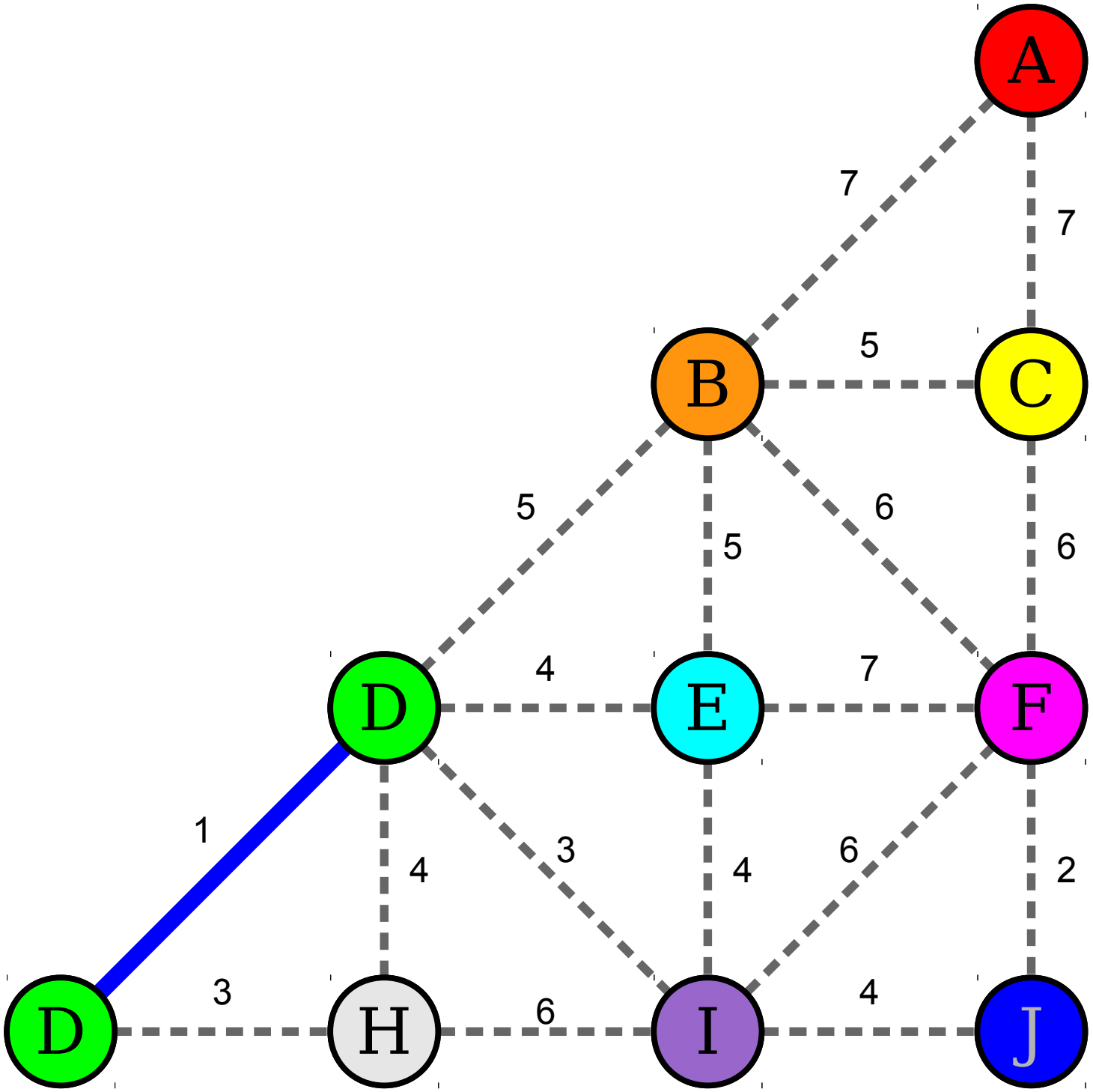
- The key step in Kruskal's algorithm is determining whether the two endpoints of an edge are already connected to one another.
- Typical approach: break the nodes apart into ***clusters***.
  - Initially, each node is in its own cluster.
  - Whenever an edge is added, the clusters for the endpoints are merged together into a new cluster.

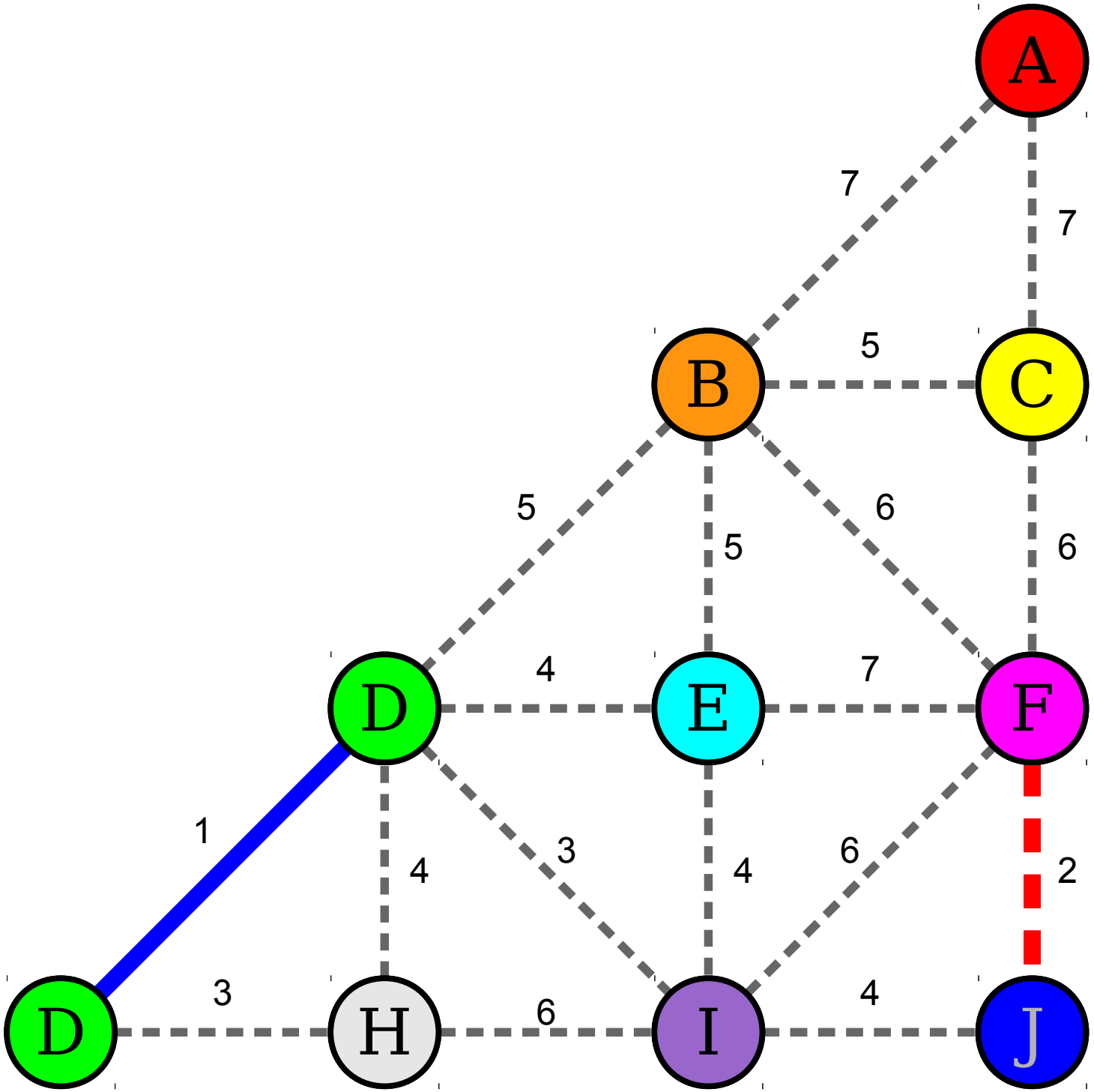

7
7
5
5
6
5
6
4
7
1
4
3
4
6
2
3
6
4

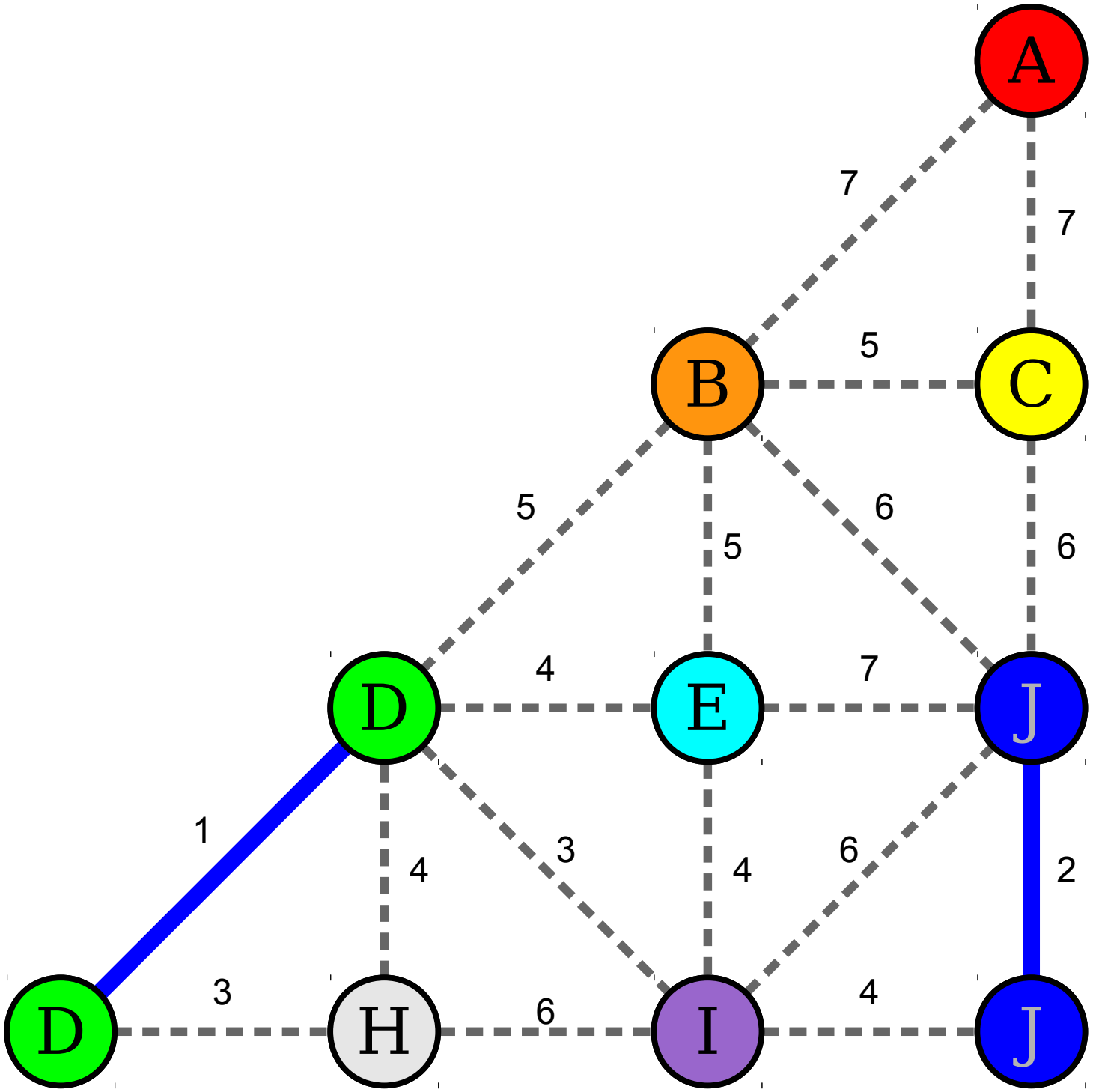

7
7
5
5
5
6
6
4
4
7
1
4
3
4
6
2
3
6
4

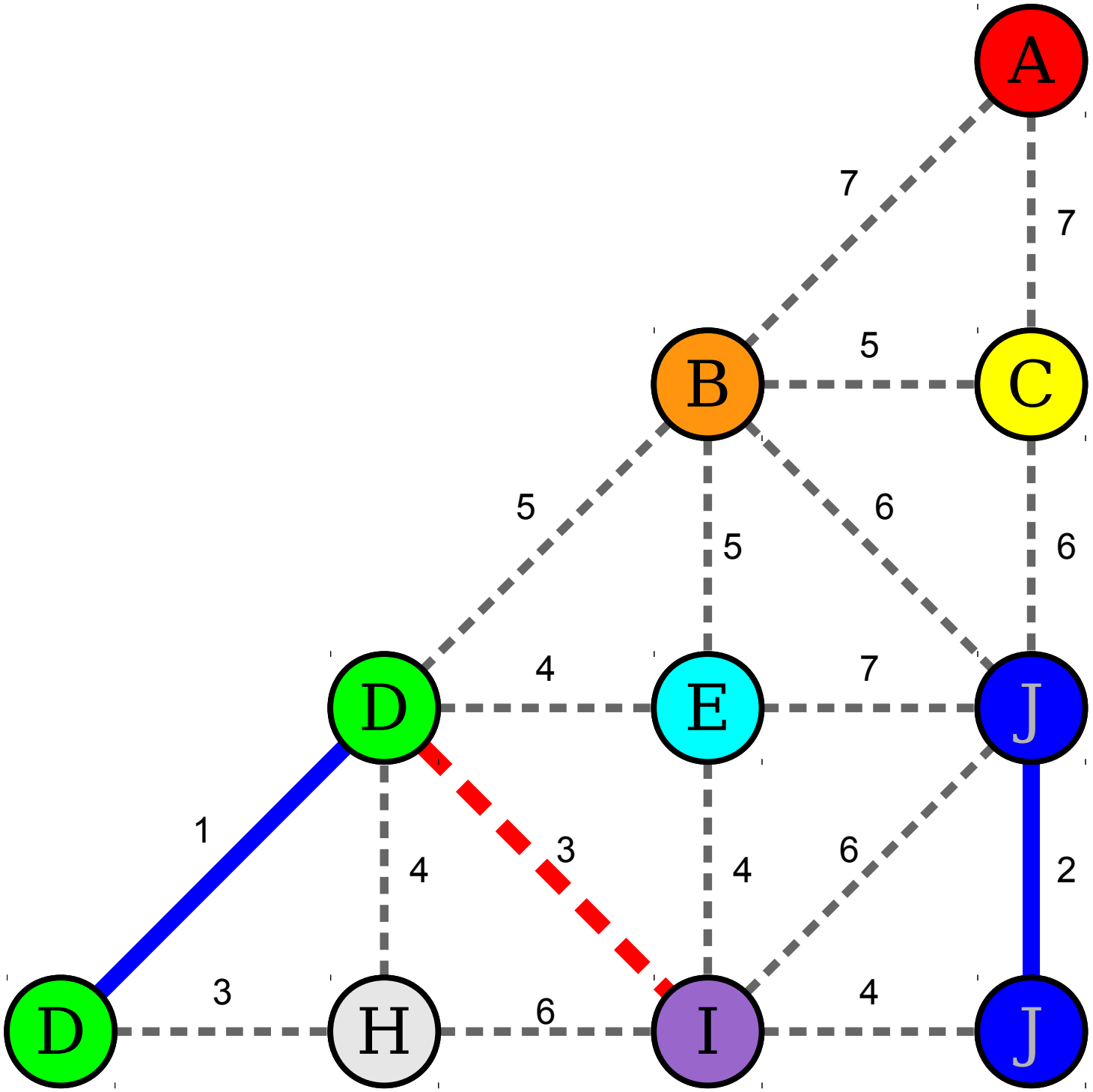

A
B
C
D
E
F
G
H
I
J
7
7
5
5
6
5
6
4
7
1
4
3
4
6
2
3
6
4

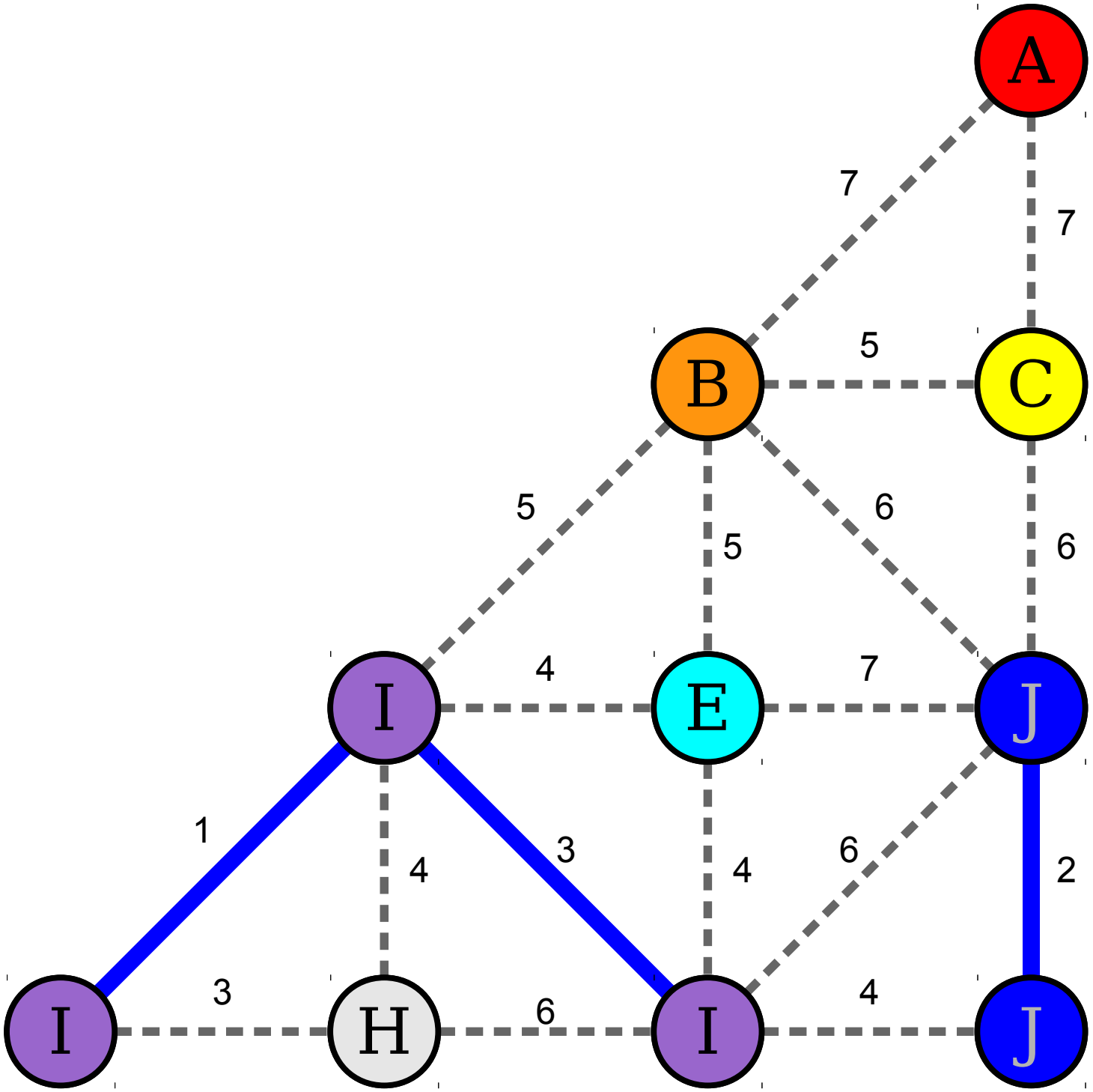

A
B
C
D
E
F
G
H
I
J
7
7
5
5
6
6
4
7
1
4
3
4
6
2
3
6
4

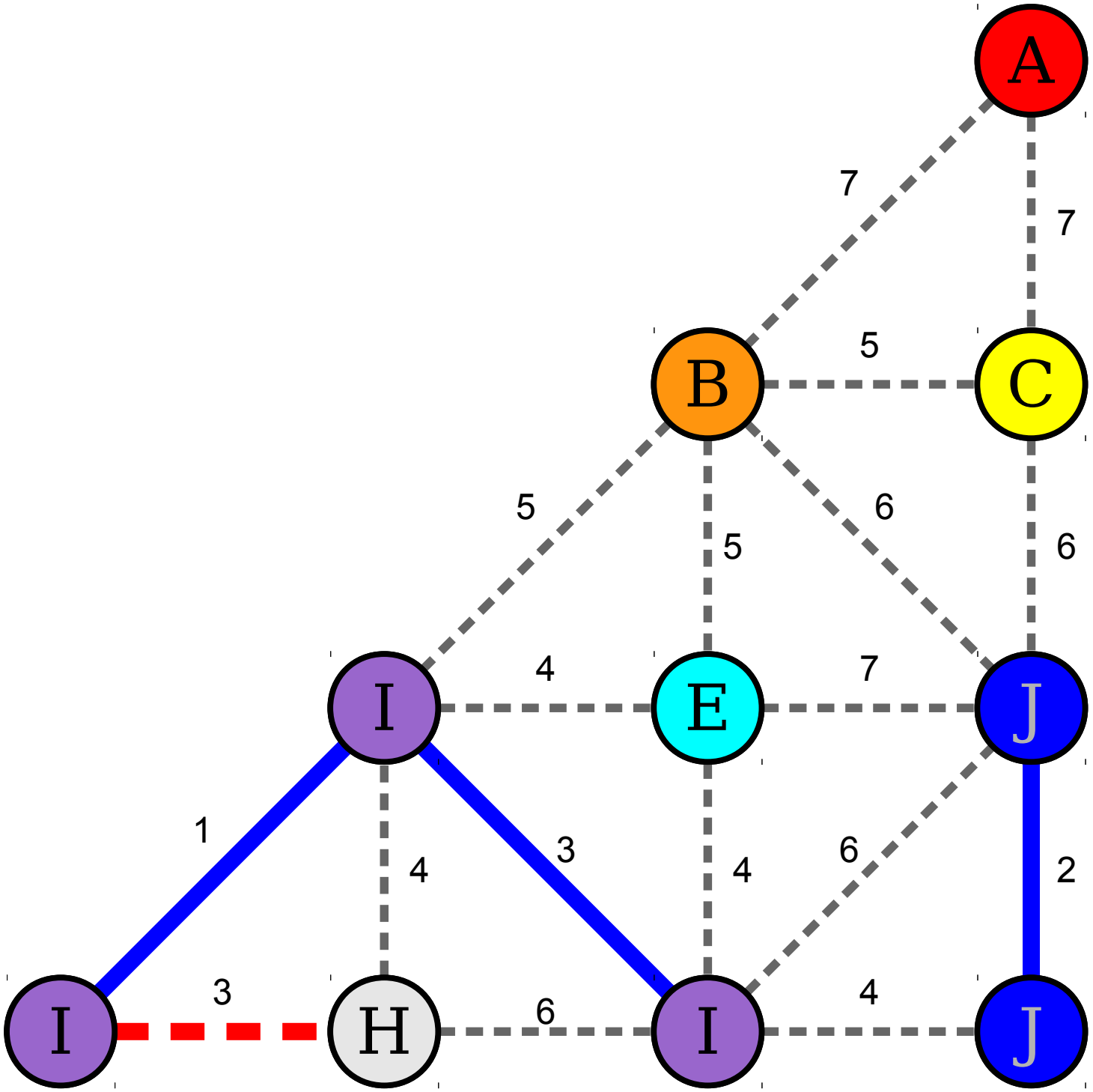

A
B
C
D
E
F
D
H
I
J
7
7
5
5
5
6
6
4
7
1
4
3
4
6
2
3
6
4

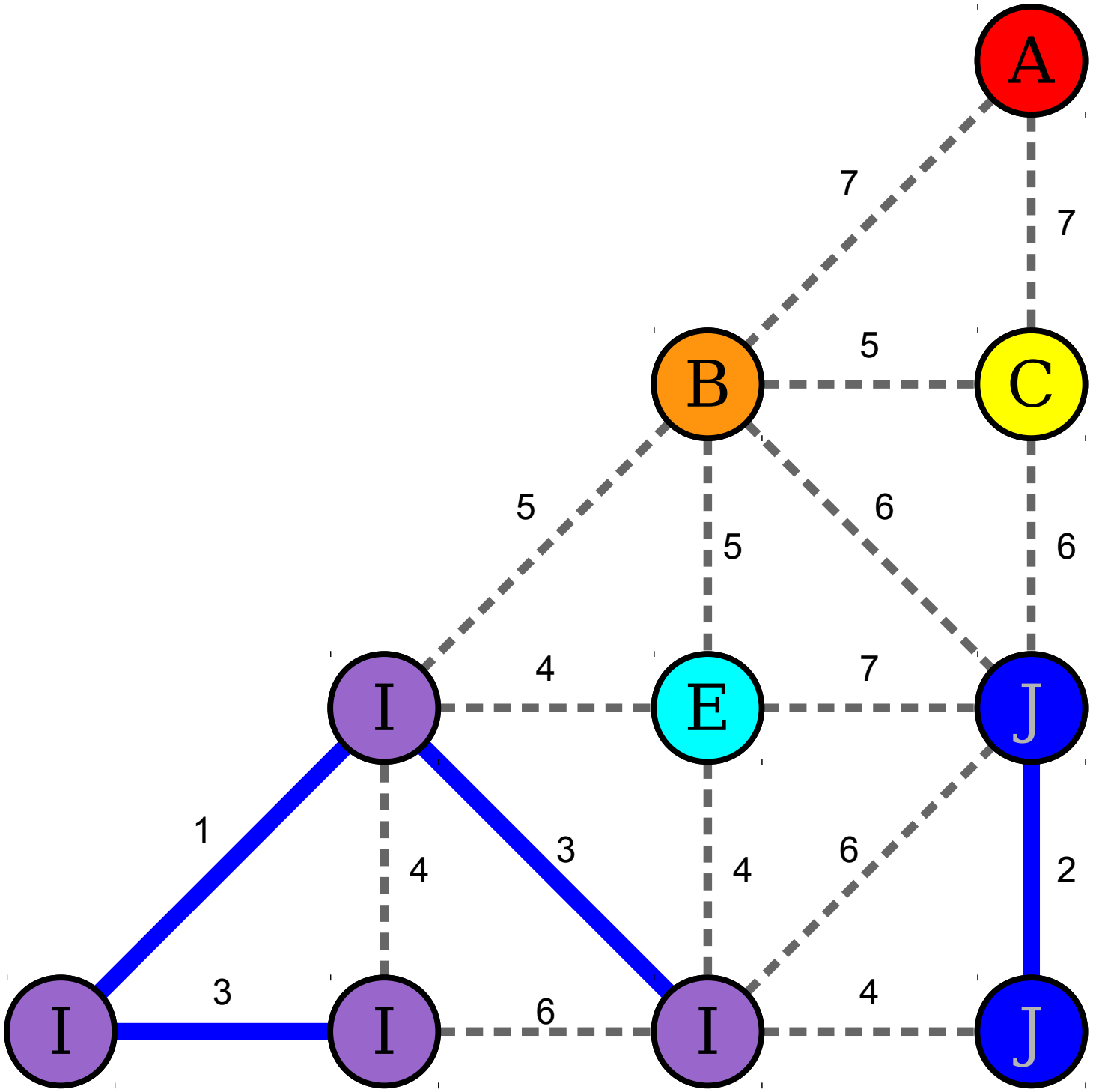

A
B
C
D
E
F
D
H
I
J
7
7
5
5
5
6
6
4
7
1
4
3
4
6
2
3
6
4

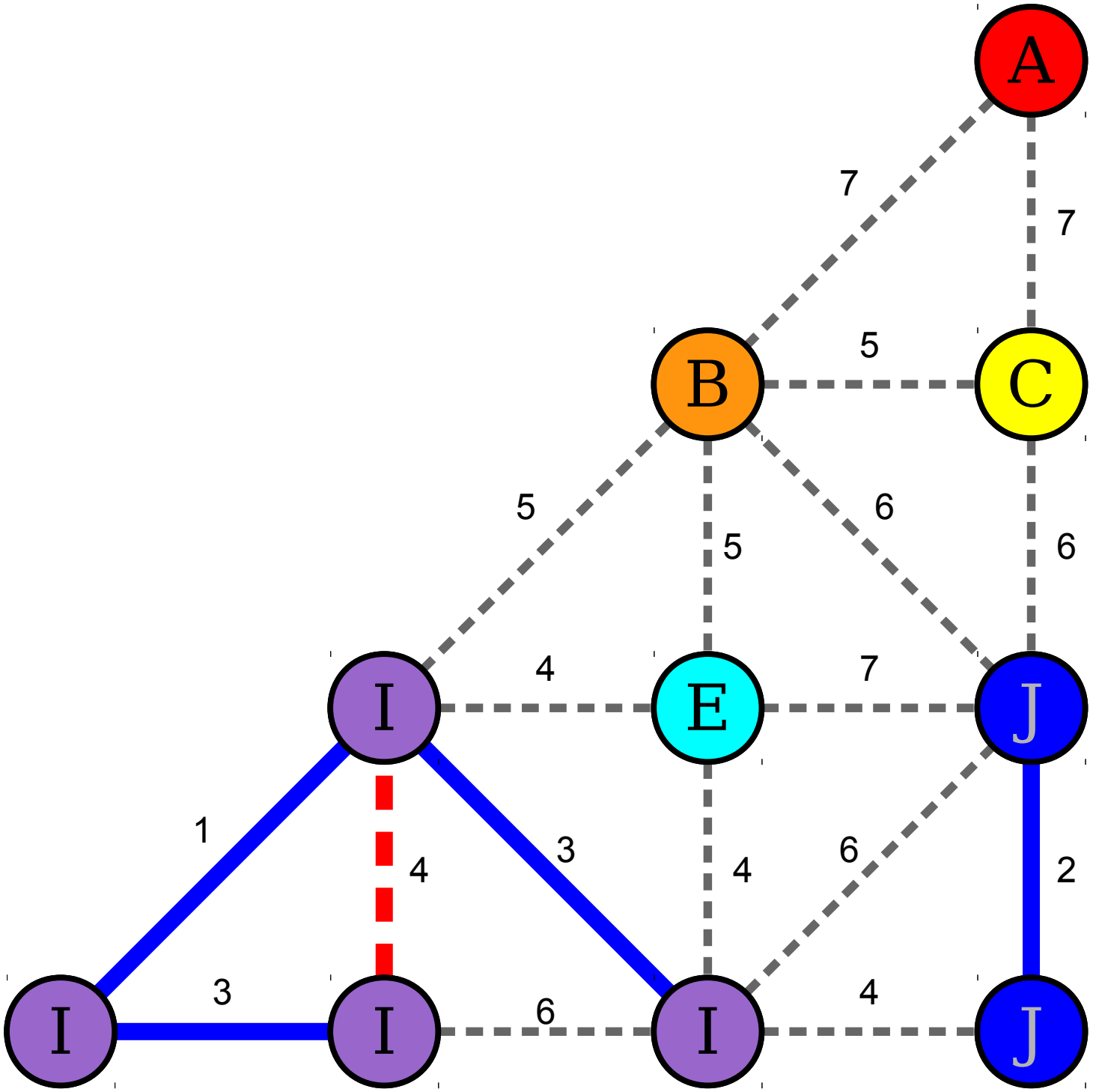

A
B
C
D
E
J
D
H
I
J
7
7
5
5
6
6
4
7
1
4
3
4
6
2
3
6
4

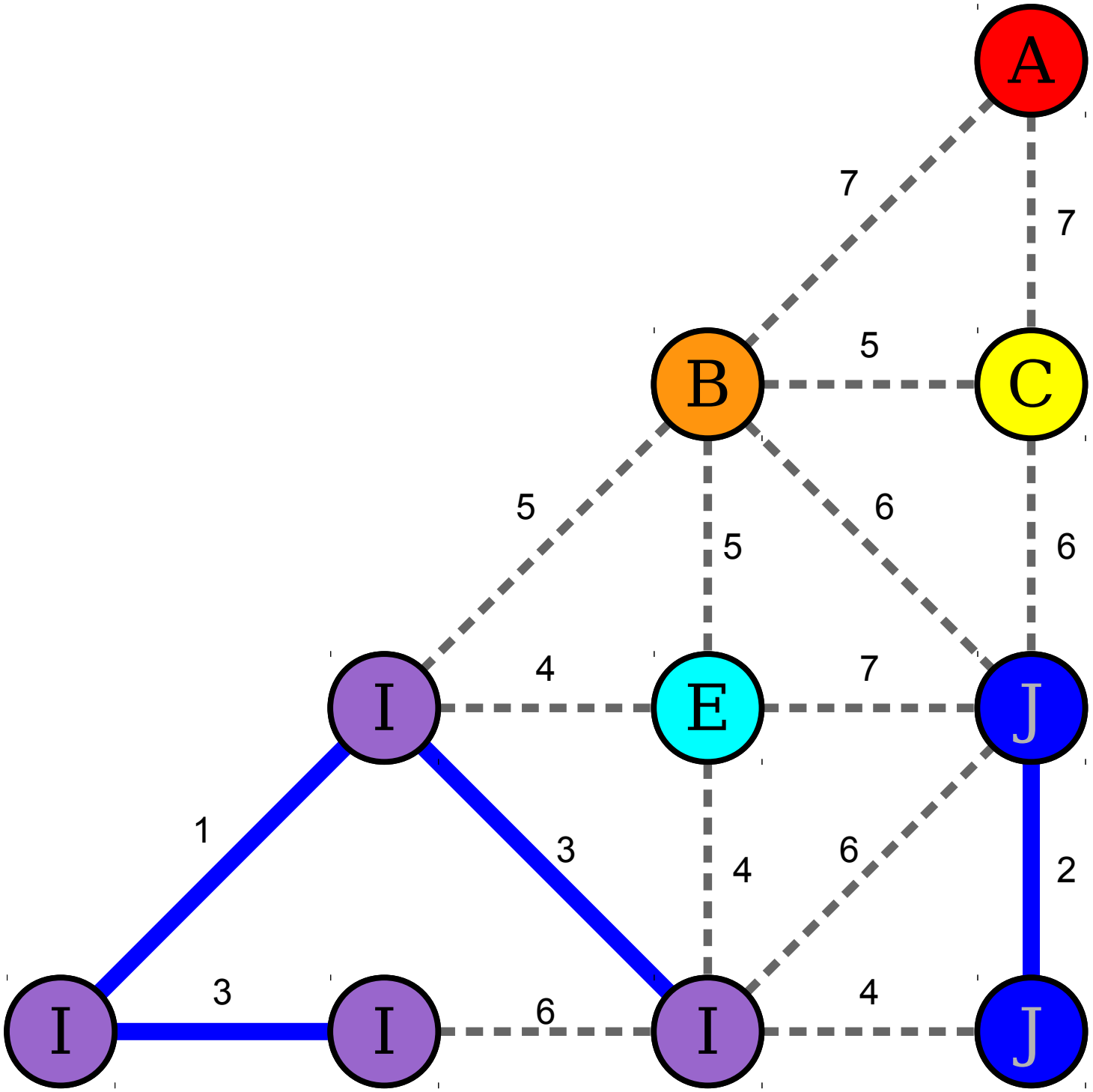

A
B
C
D
E
J
D
H
I
J
7
7
5
5
6
5
6
4
7
1
4
3
4
6
2
3
6
4

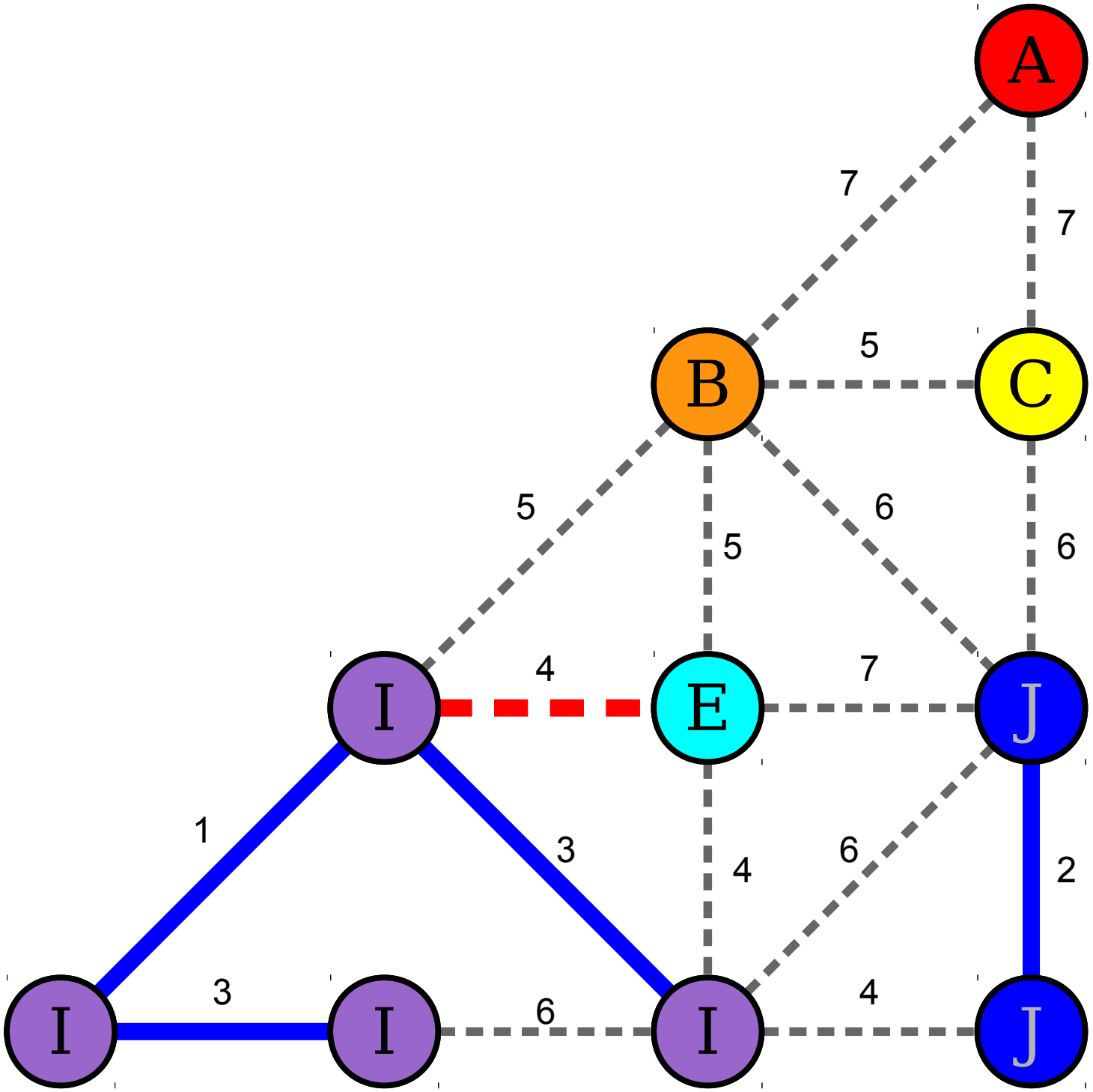

A
B
C
I
E
J
I
H
I
J
7
7
5
5
6
5
6
4
7
1
4
3
4
6
2
3
6
4

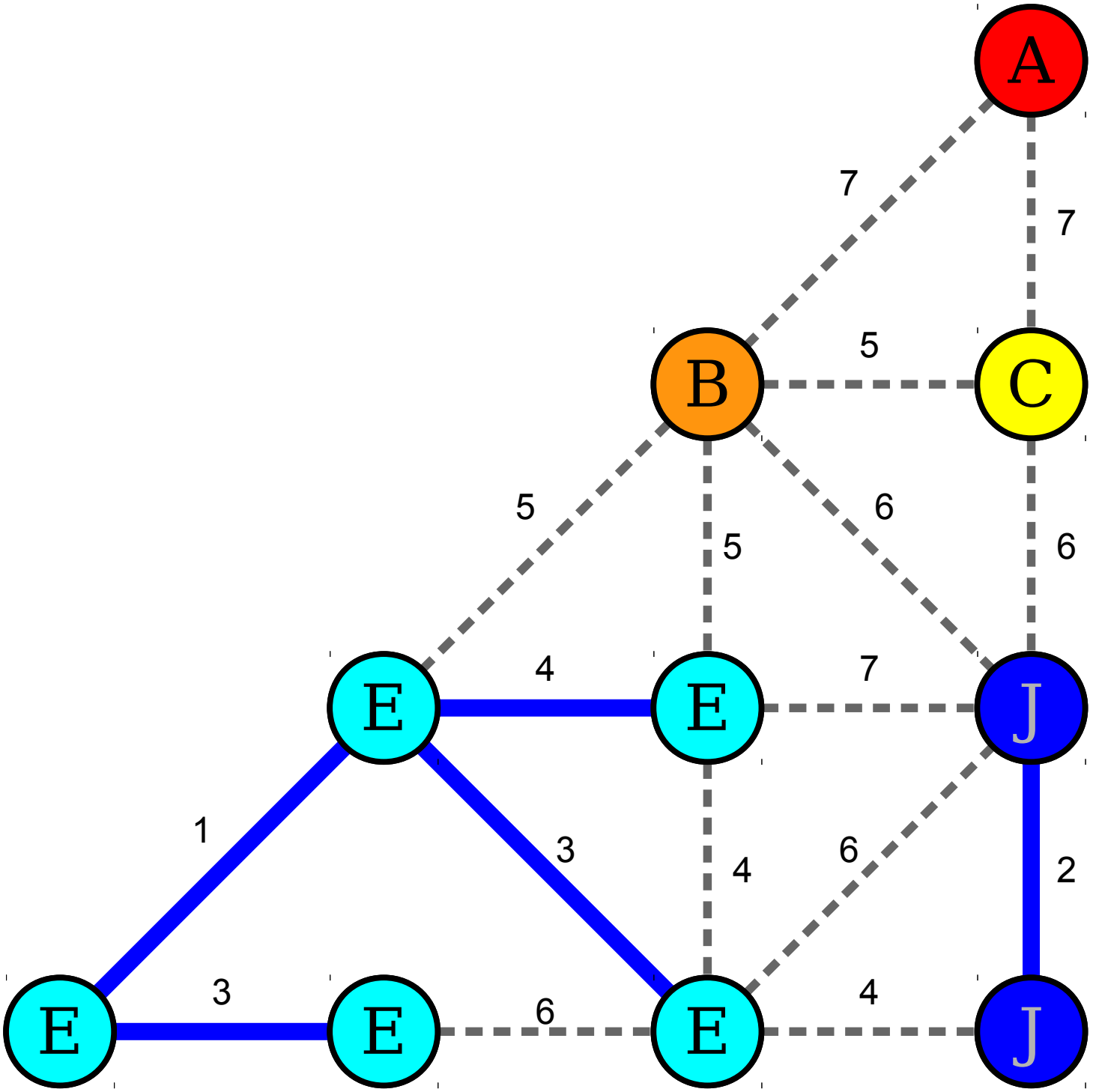

A
B
C
I
E
J
I
H
I
J
7
7
5
5
5
6
6
4
7
1
4
3
4
6
2
3
6
4

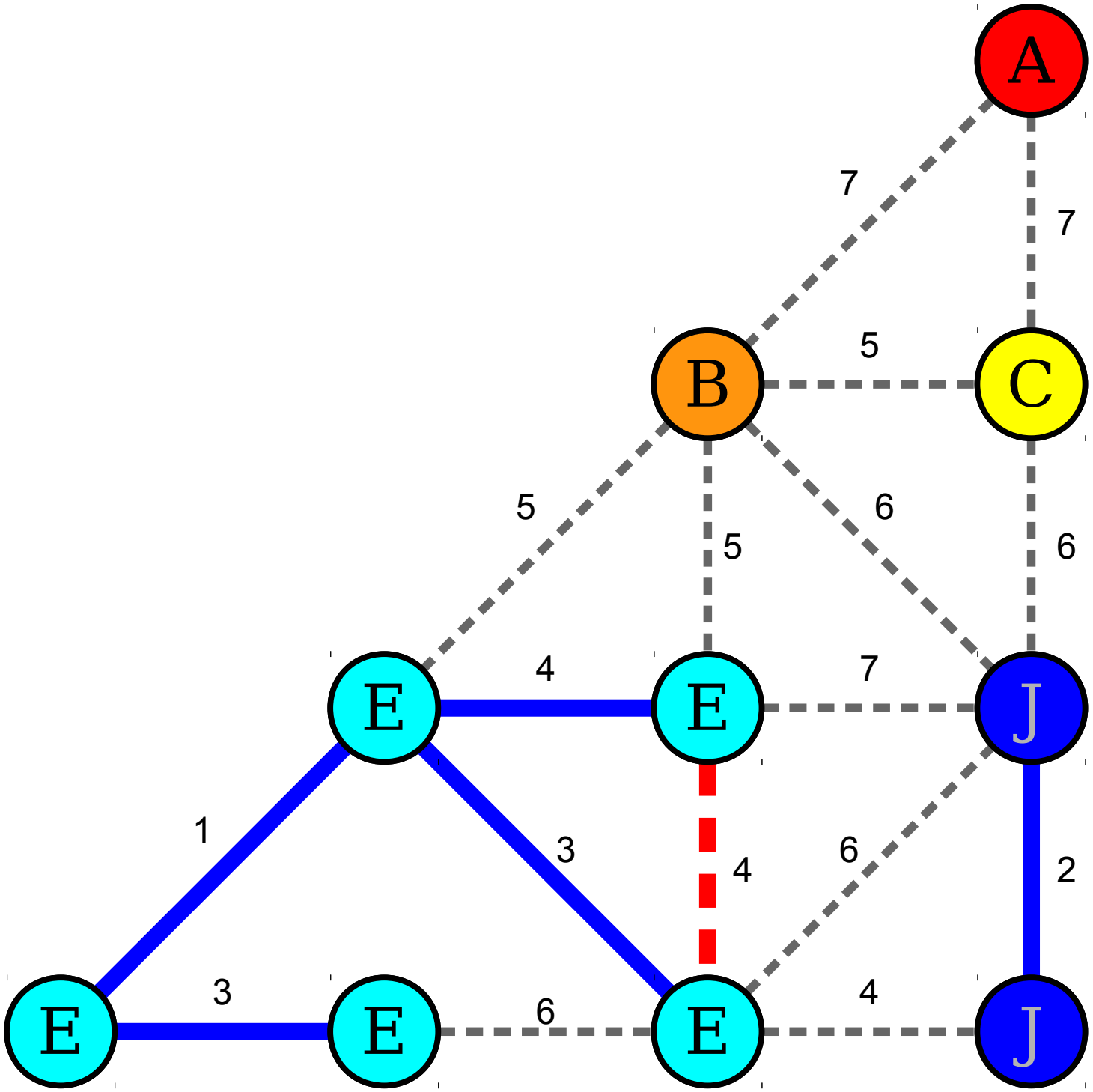

A
B
C
I
E
J
I
I
I
J
7
7
5
5
5
6
6
4
7
1
4
3
4
6
2
3
6
4

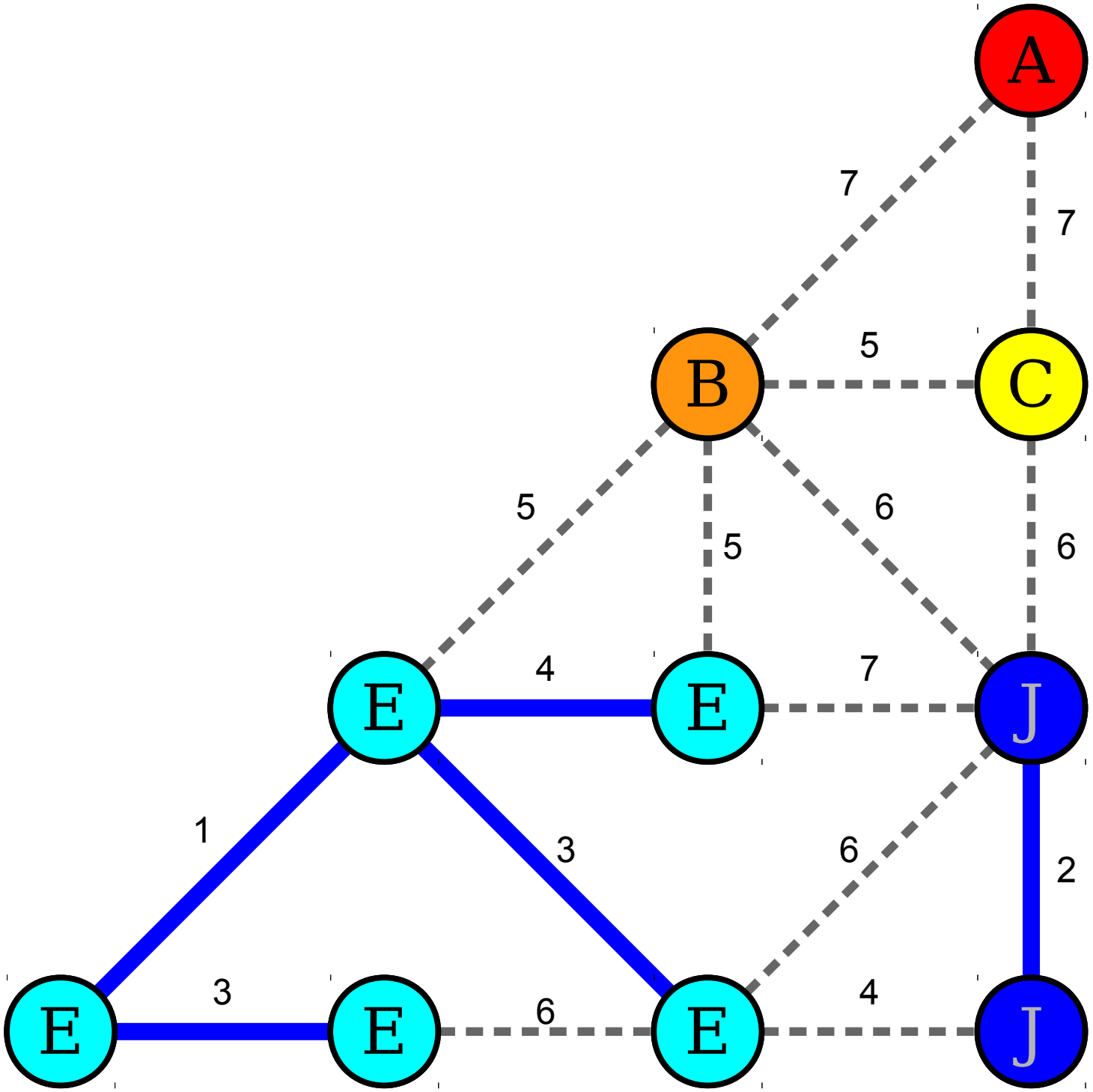

A
B
C
I
E
J
I
I
I
J
7
7
5
5
5
6
6
4
7
1
4
3
4
6
2
3
6
4

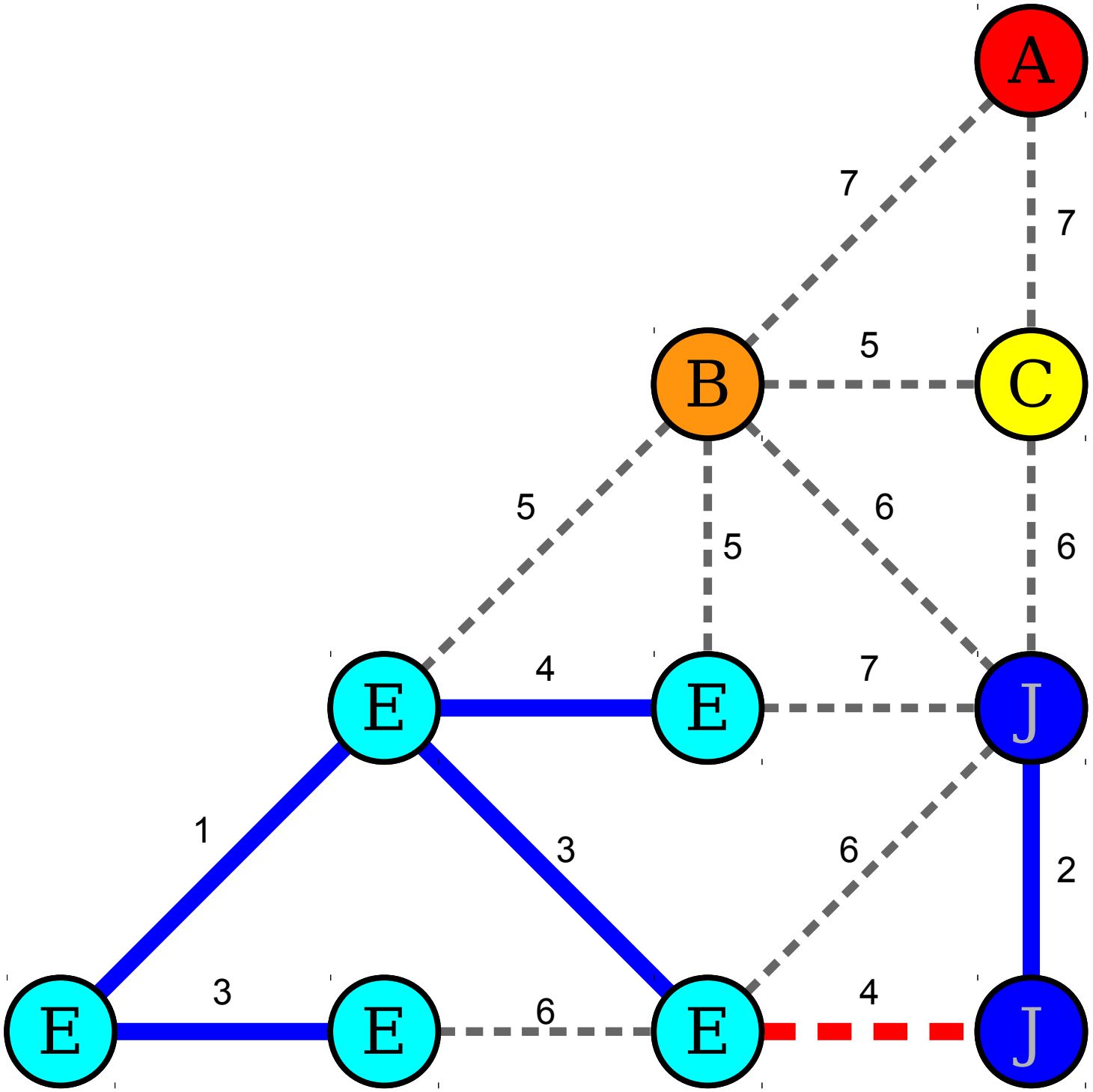

A
B
C
E
I
I
I
I
J
J
7
7
5
5
5
6
6
4
7
1
3
4
6
2
3
6
4

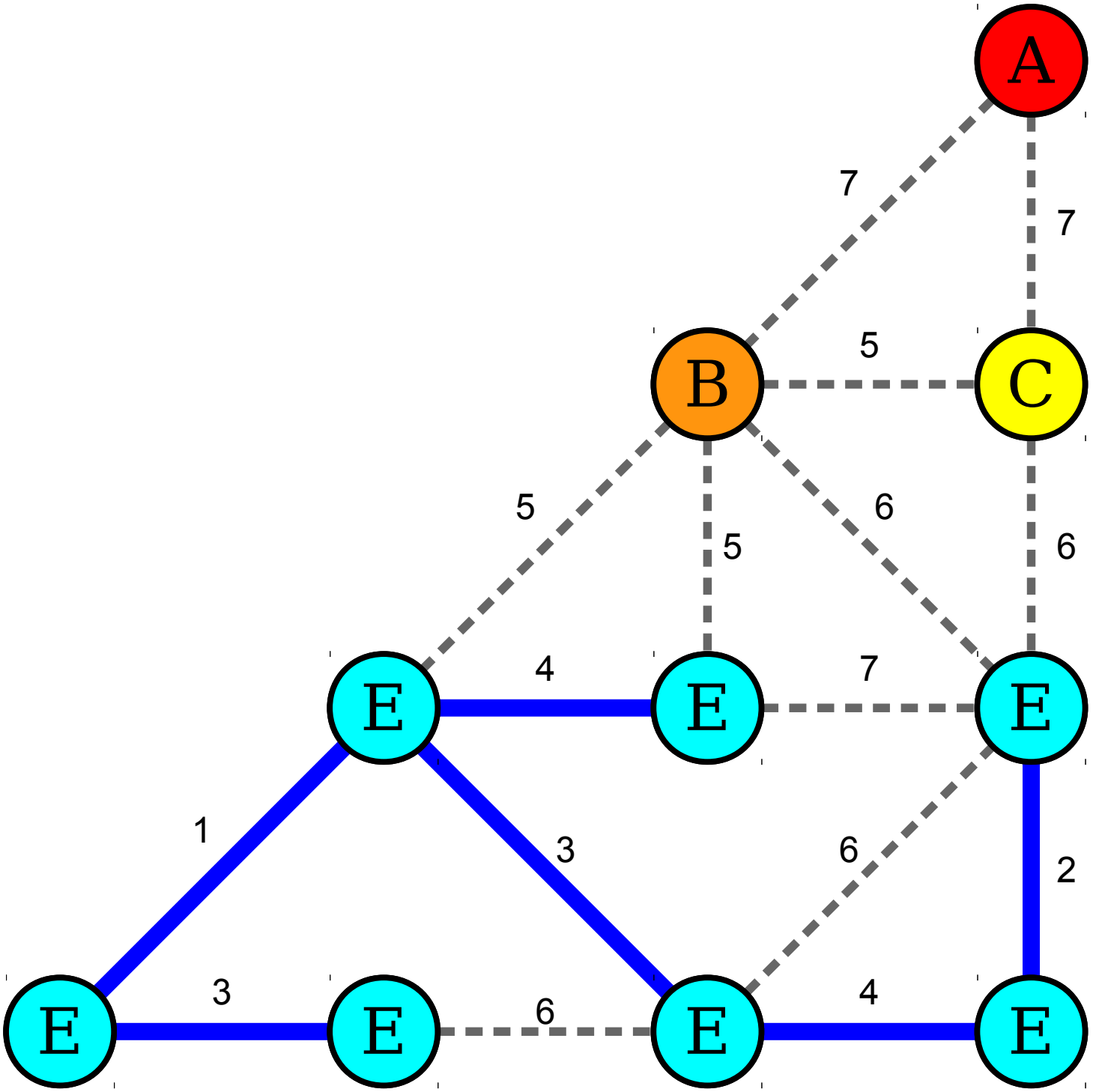

A
B
C
I
E
J
I
I
I
J
7
7
5
5
5
6
6
4
7
1
3
4
6
2
3
6
4

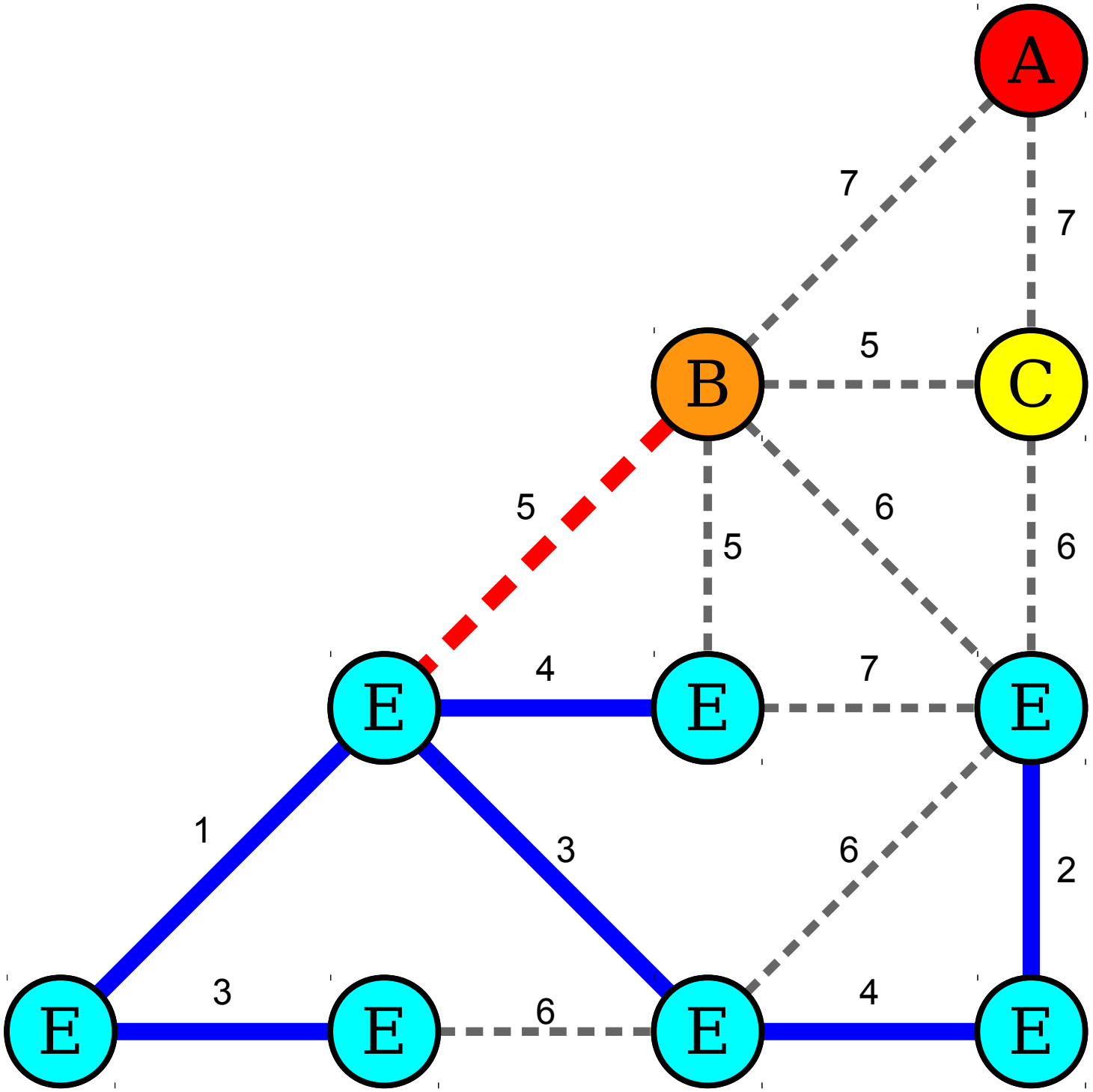

A
B
C
E
E
J
E
E
E
J
7
7
5
5
5
6
6
4
7
1
3
4
6
2
3
6
4

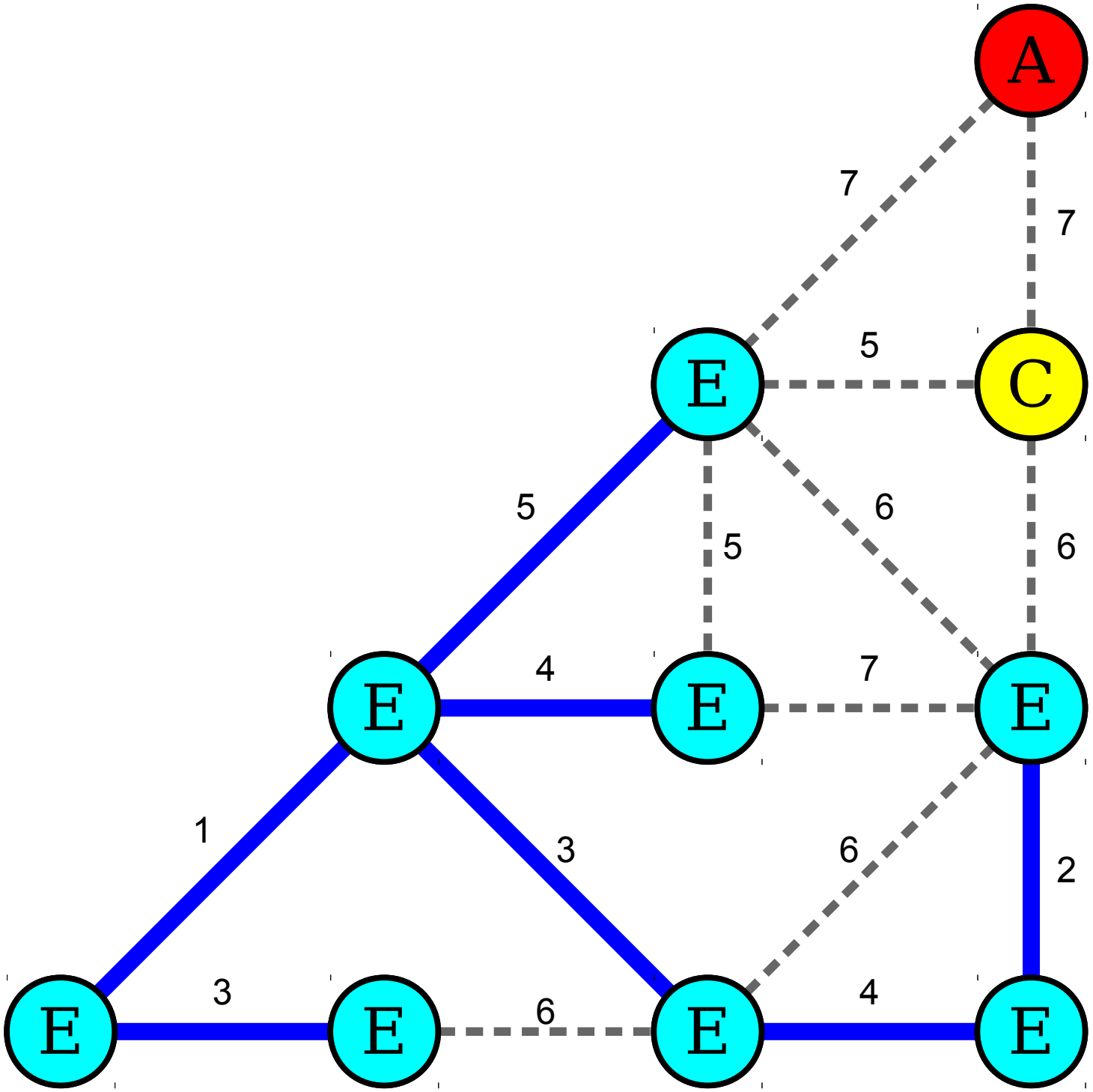

A
B
C
E
E
J
E
E
E
J
7
7
5
5
5
6
6
4
7
1
3
4
6
2
3
6
4

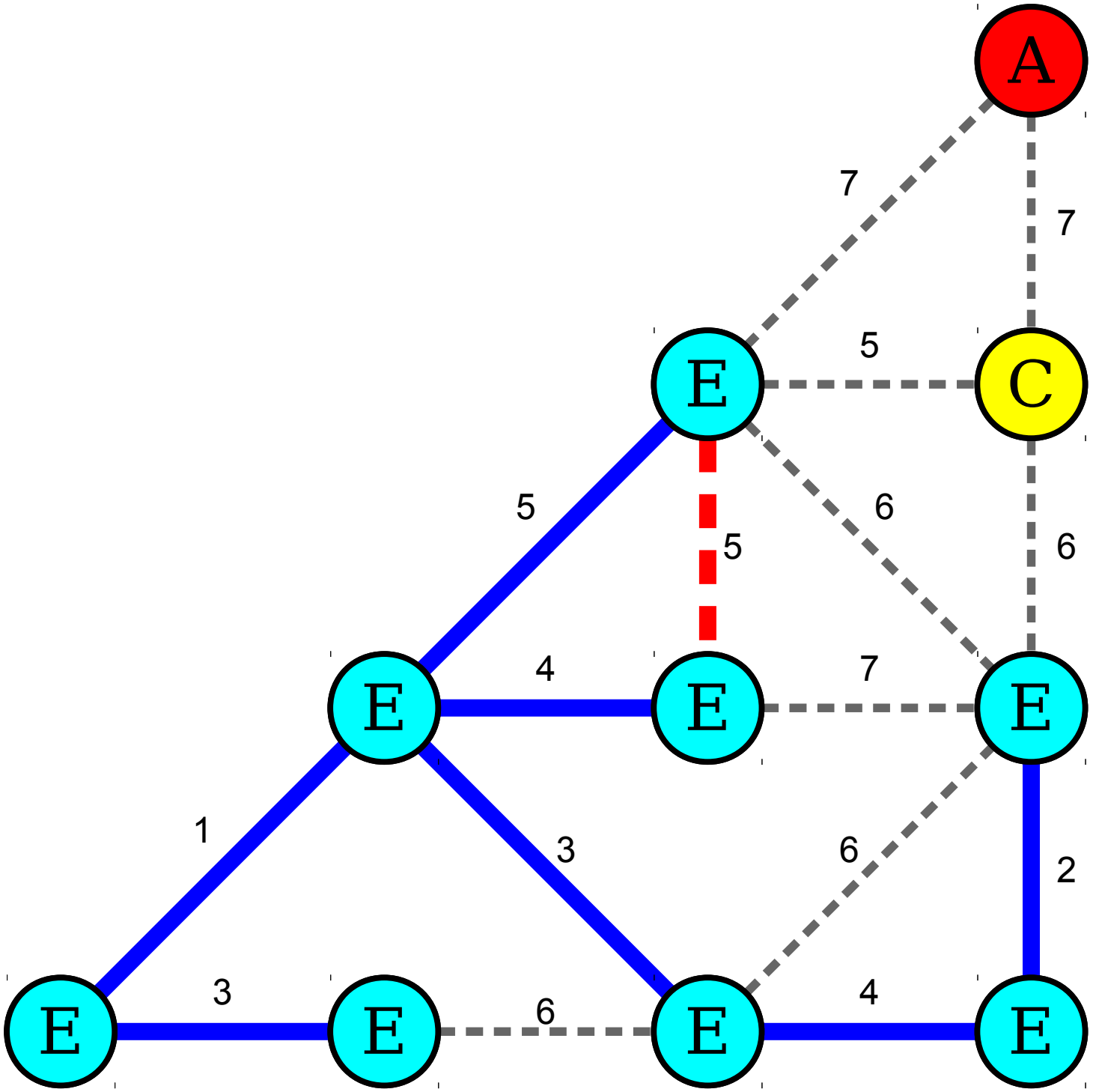

A
B
C
E
E
J
E
E
E
J
7
7
5
5
5
6
6
4
7
1
3
6
2
3
6
4

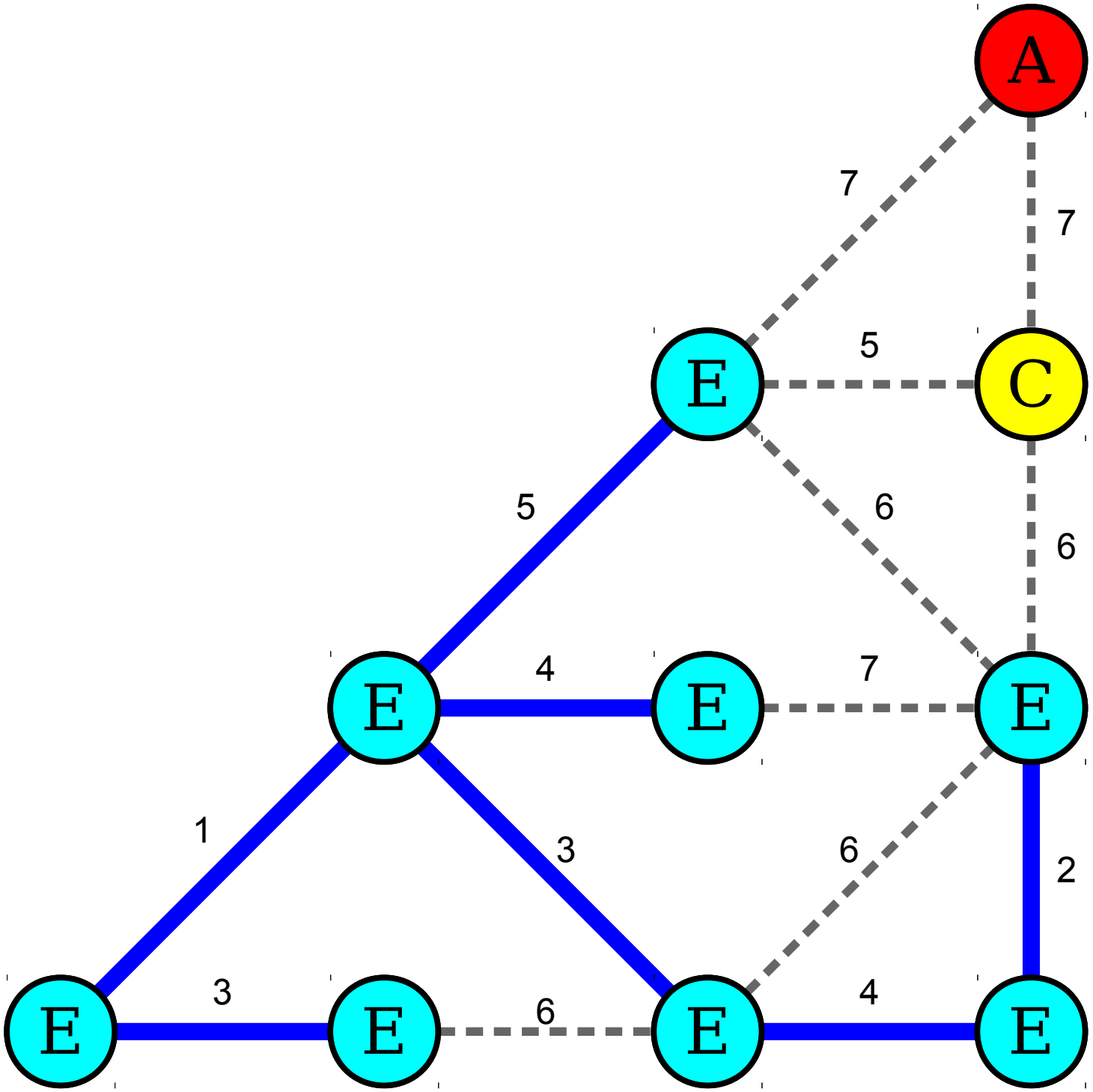

A
B
C
E
E
J
E
E
E
J
7
7
5
5
5
6
6
4
7
1
3
6
2
3
6
4

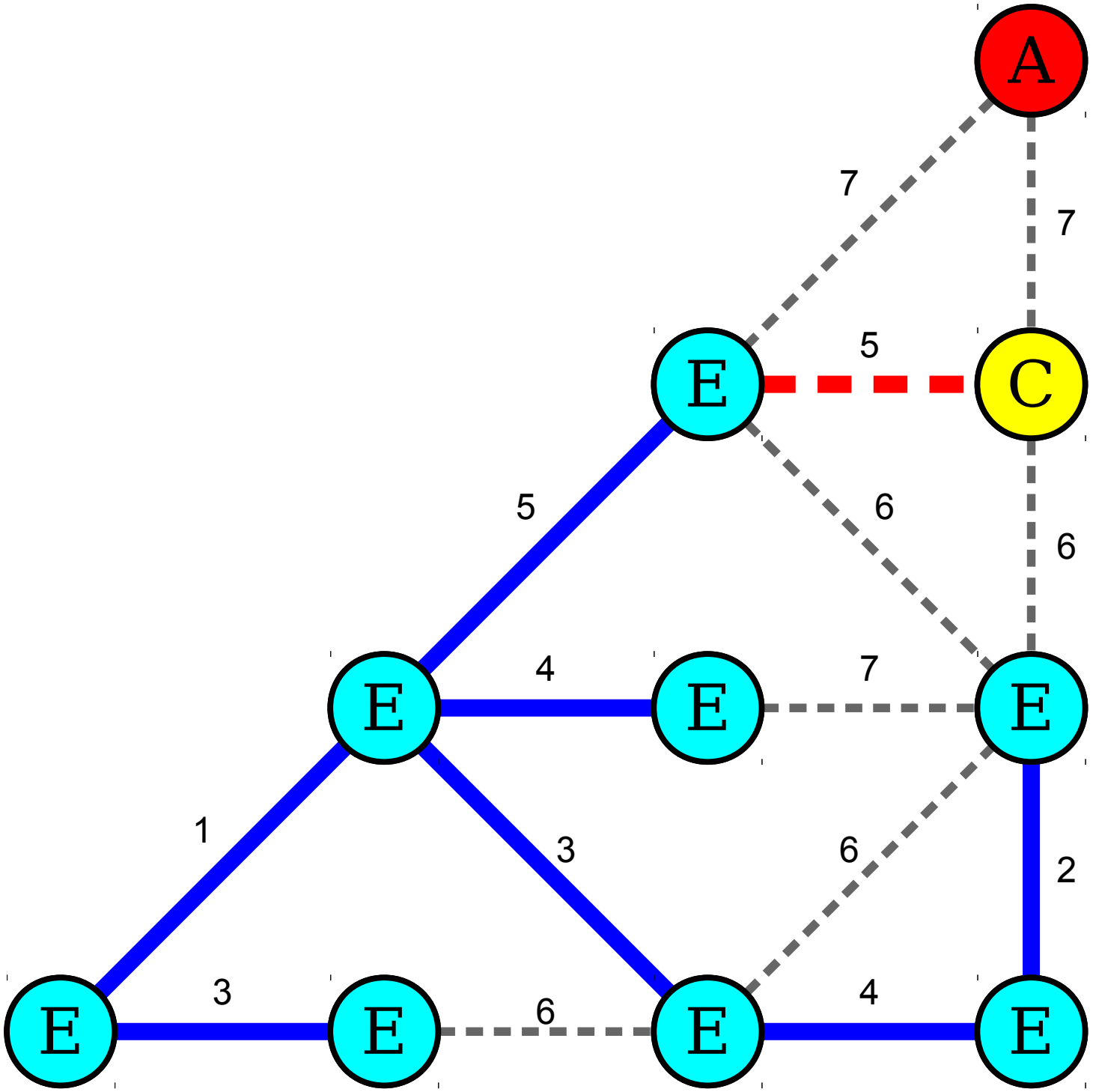

A
B
C
E
E
E
E
E
E
7
7
5
5
5
6
6
4
7
1
3
6
2
3
6
4

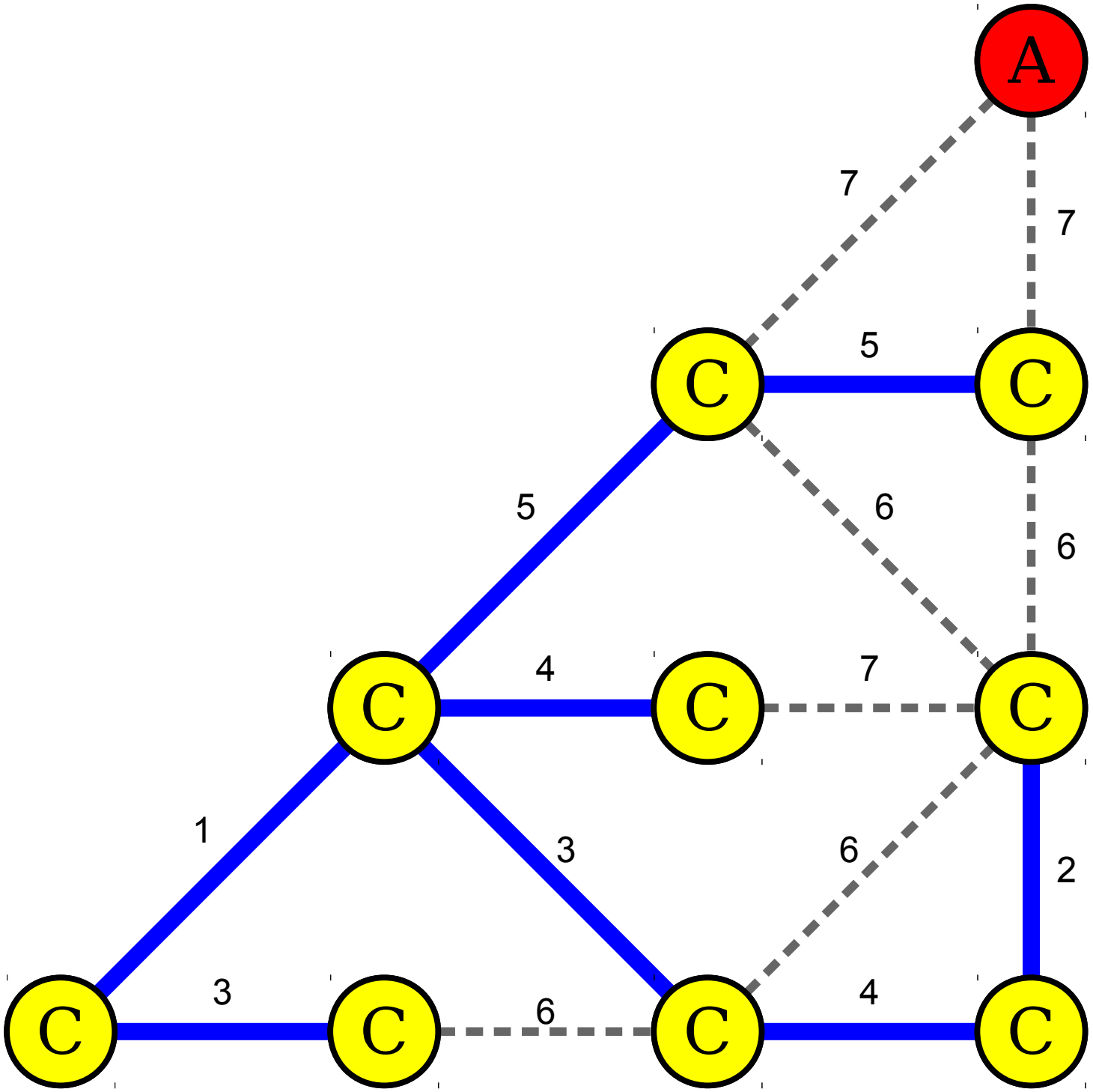

A
B
C
E
E
E
E
E
E
7
7
5
5
5
6
6
4
7
1
3
6
2
3
6
4

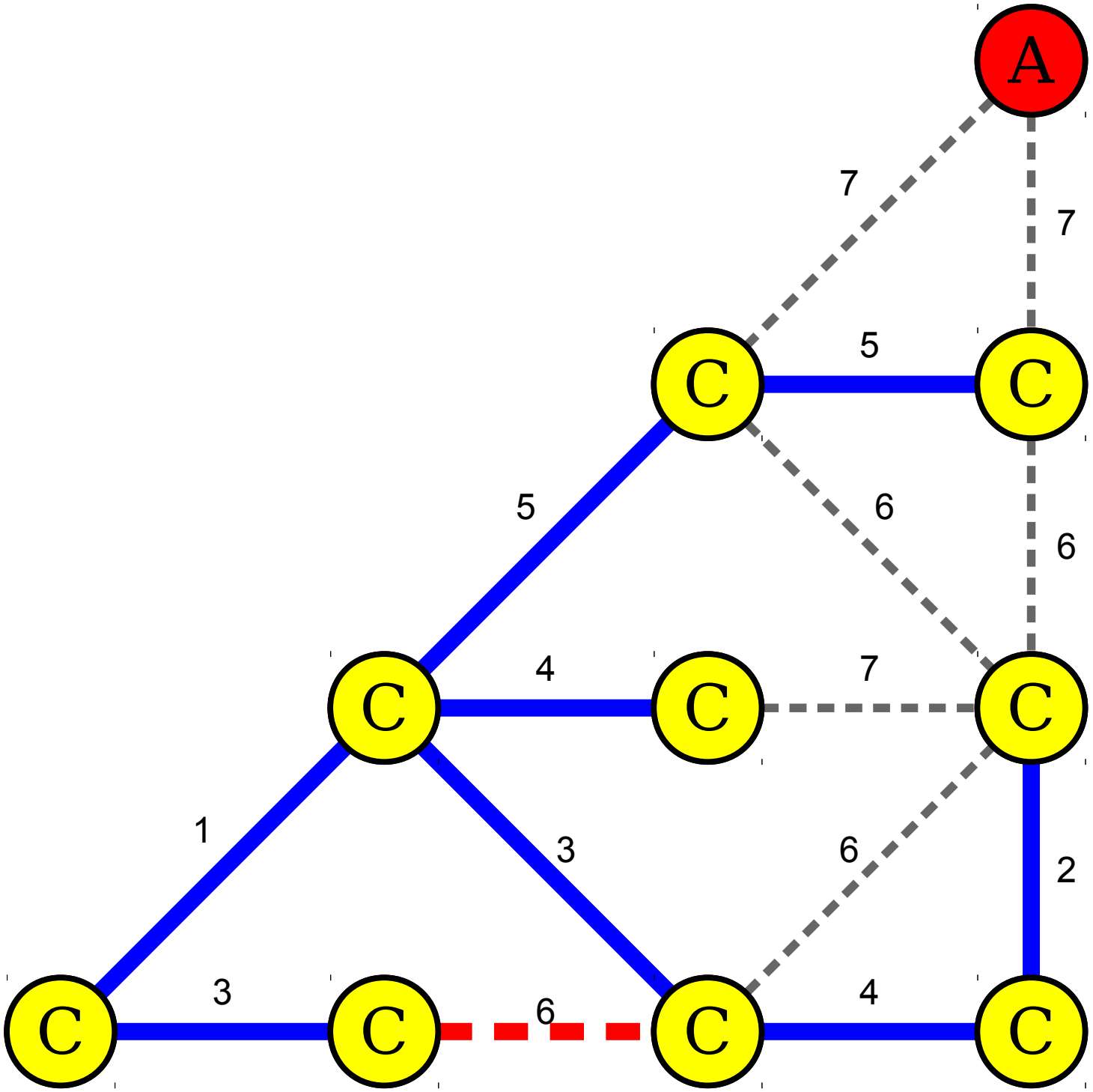

A
7
7
E
5
C
5
6
6
5
E
4
E
7
E
1
3
6
2
E
3
E
6
E
4
E

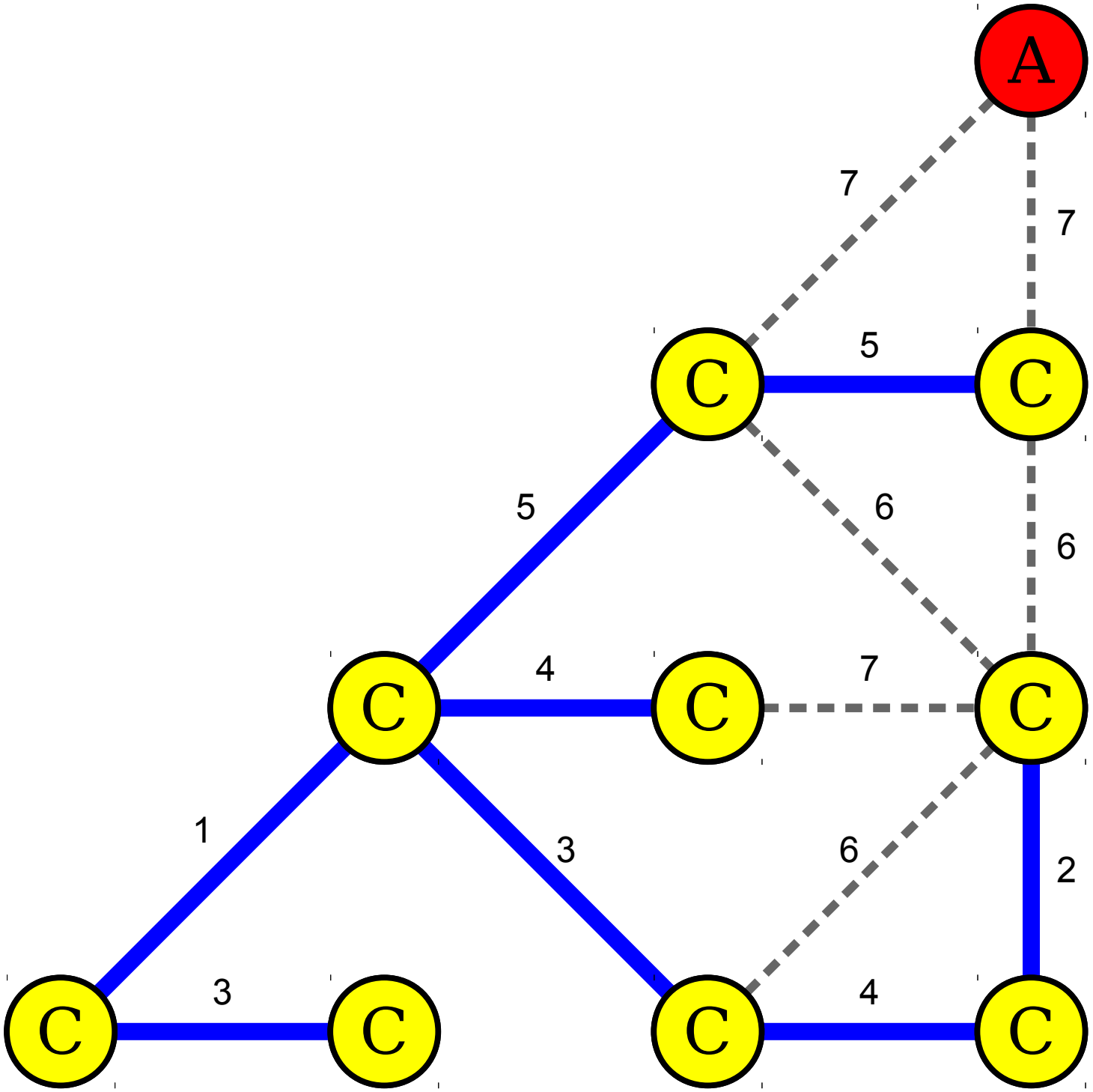

A
7
7
E
5
C
5
6
6
5
E
4
E
7
E
1
3
6
2
E
3
E
6
E
4
E

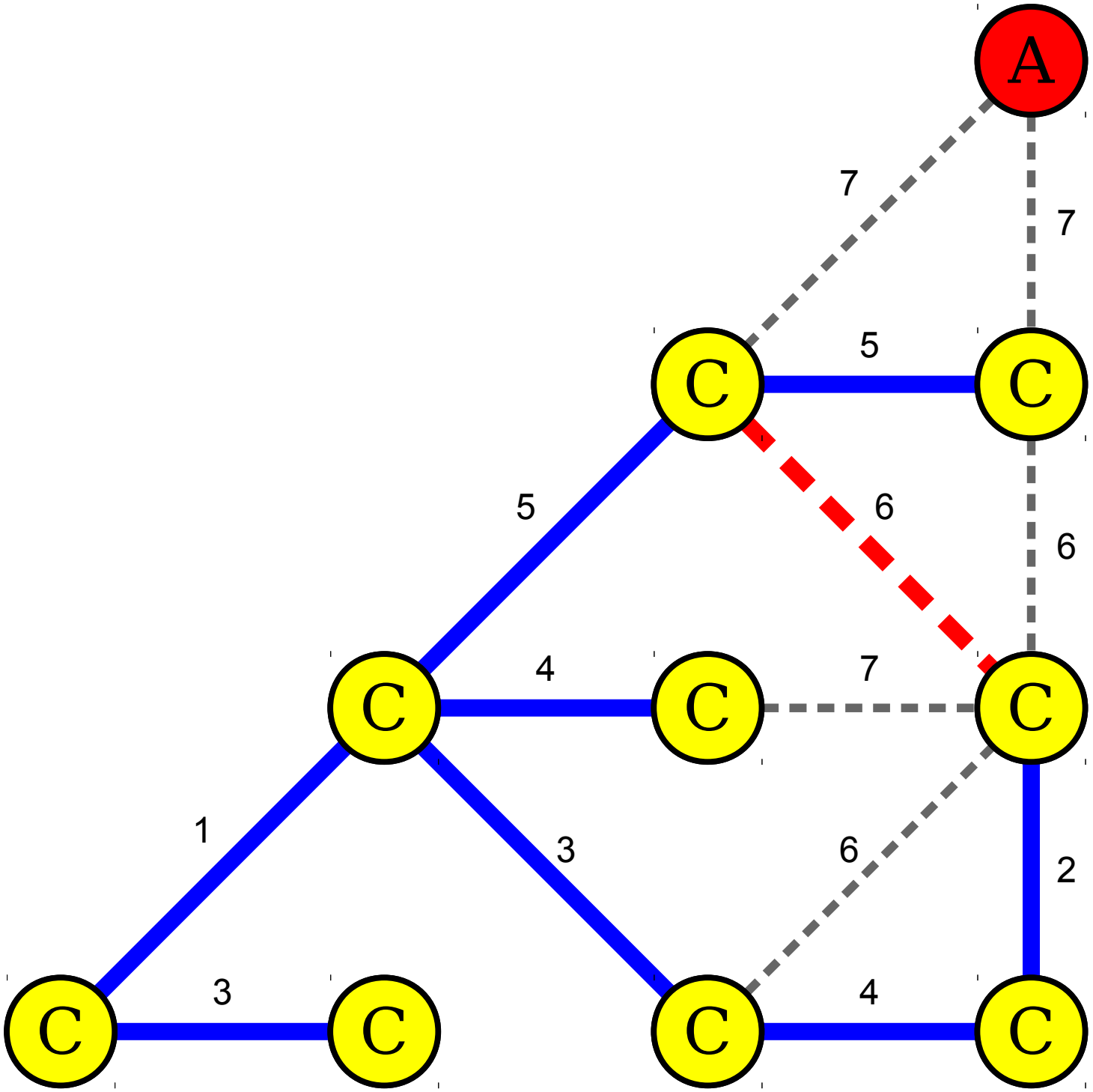

A
7
7
E
5
C
5
6
6
E
4
E
7
E
1
3
6
2
E
3
E
6
E
4
E

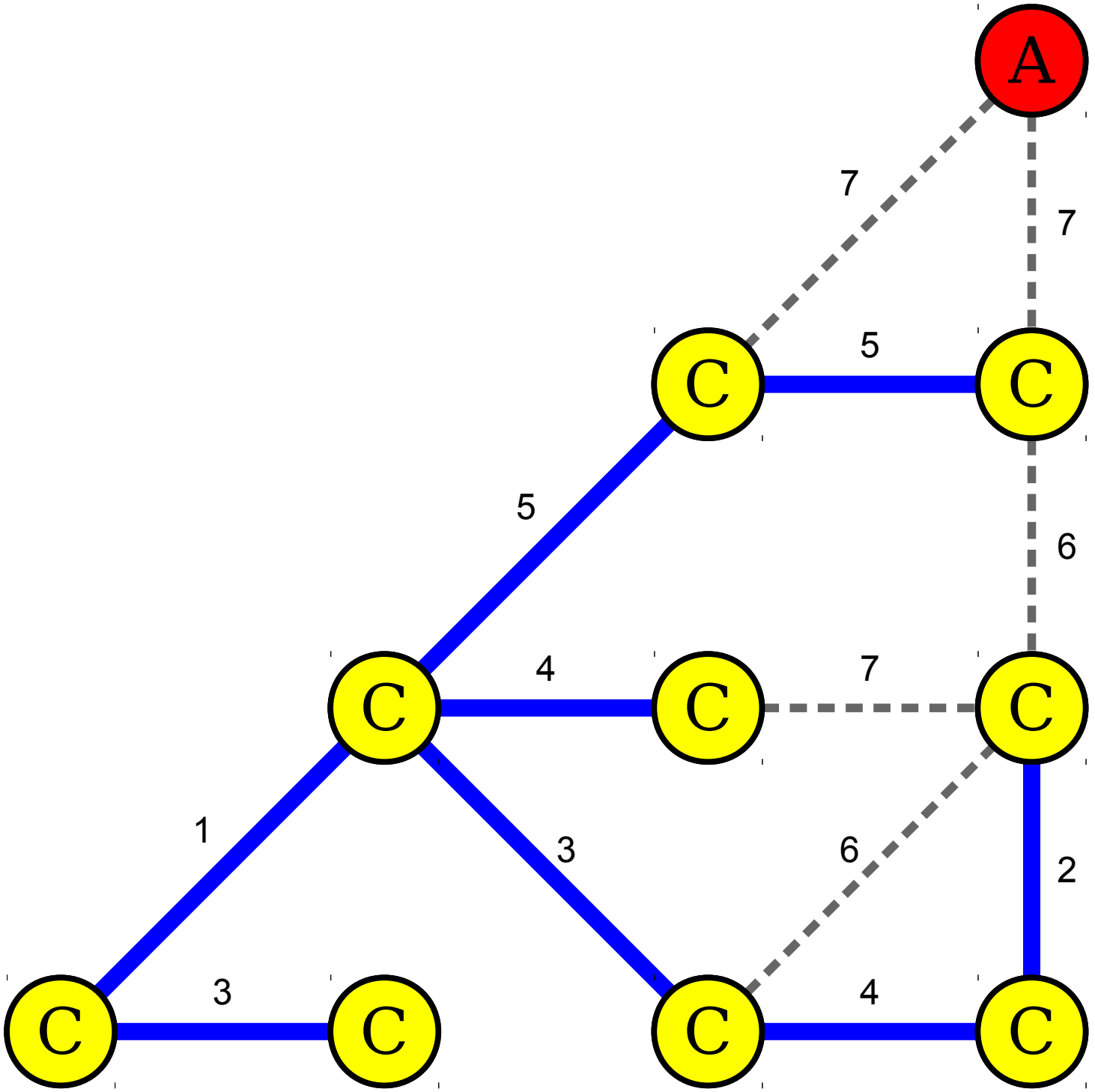

A
7
7
E
5
C
5
6
6
E
4
E
7
E
1
3
6
2
E
3
E
6
E
4
E

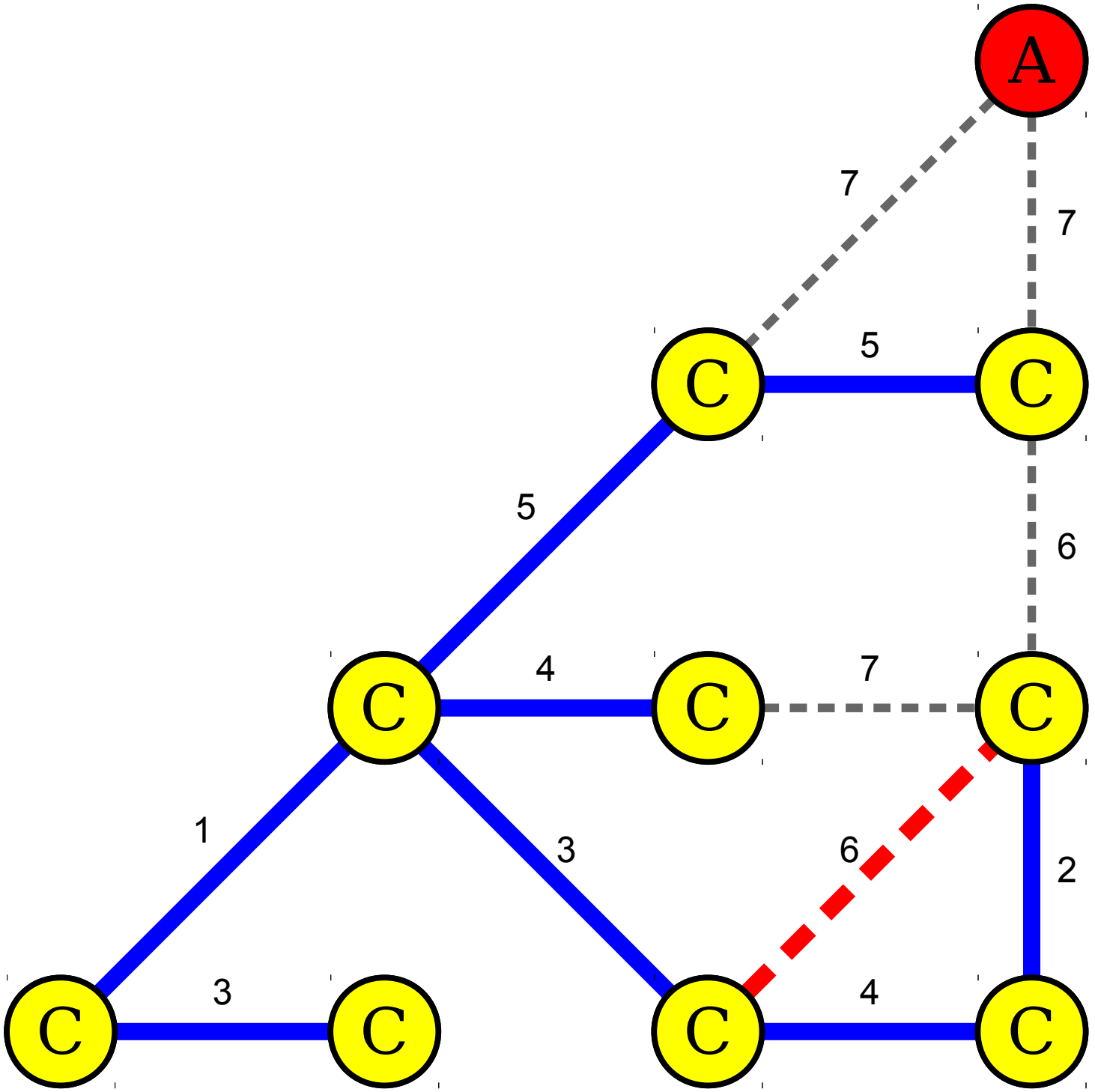

A
7
7
C
5
C
5
6
6
C
4
C
7
C
1
3
6
2
C
3
C
6
C
4
C

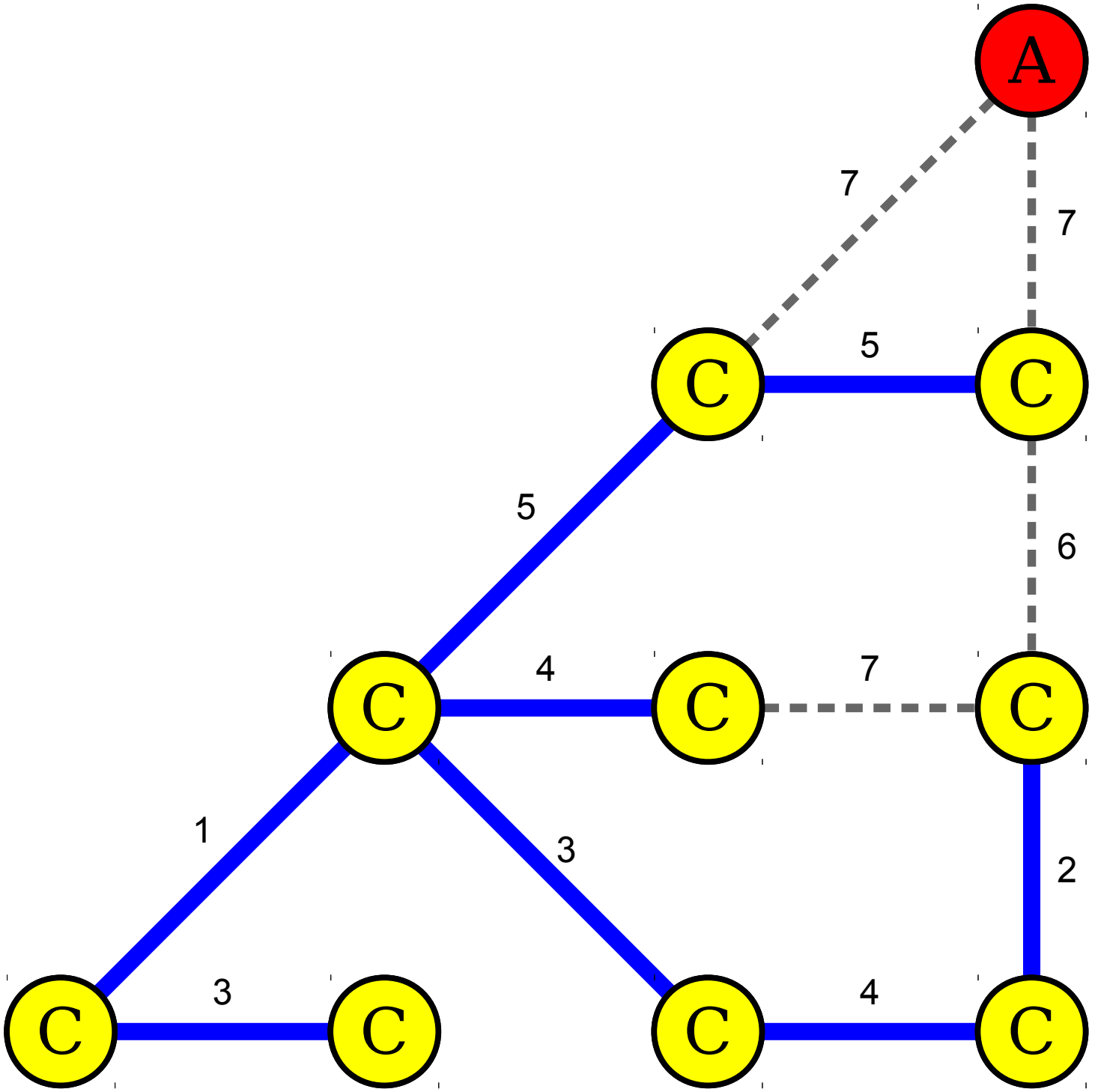

A
7
7
C
5
C
5
6
6
C
4
C
7
C
1
3
6
2
C
3
C
6
C
4
C

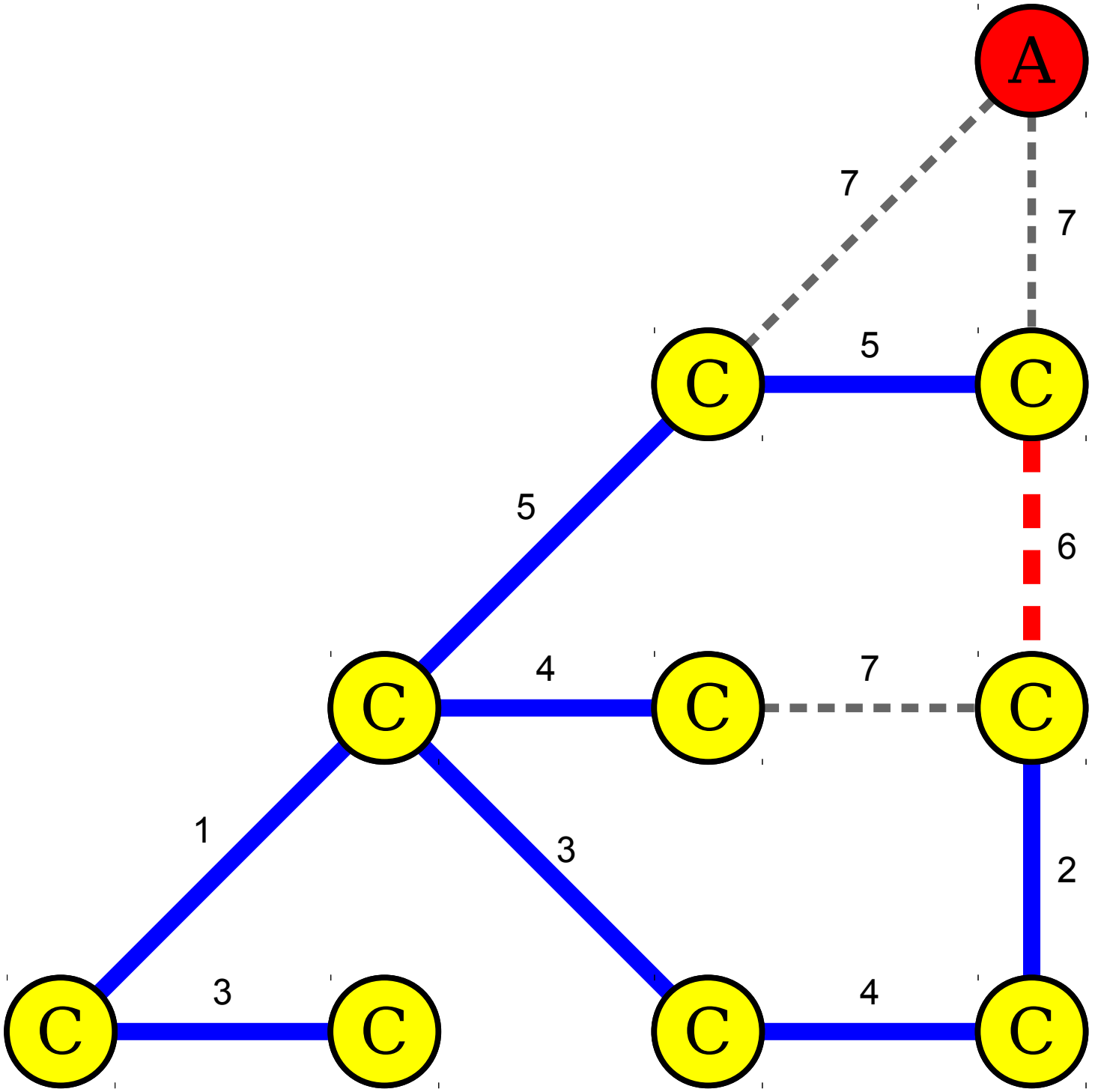

A
7
7
C
5
C
5
6
6
C
4
C
7
C
1
3
6
2
C
3
C
C
4
C

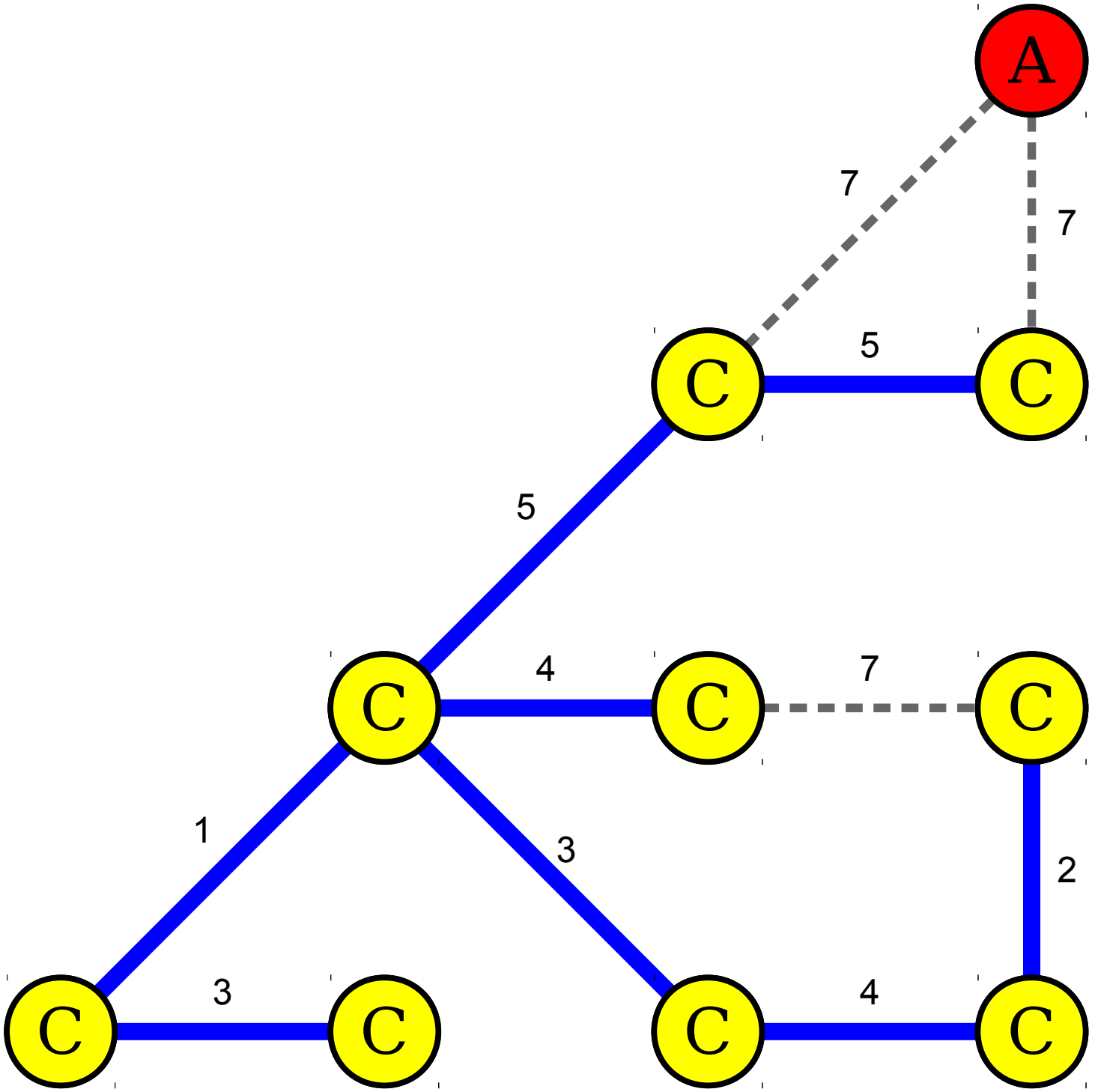

A
7
7
C
5
C
5
6
6
C
4
C
7
C
1
3
6
2
C
3
C
C
4
C

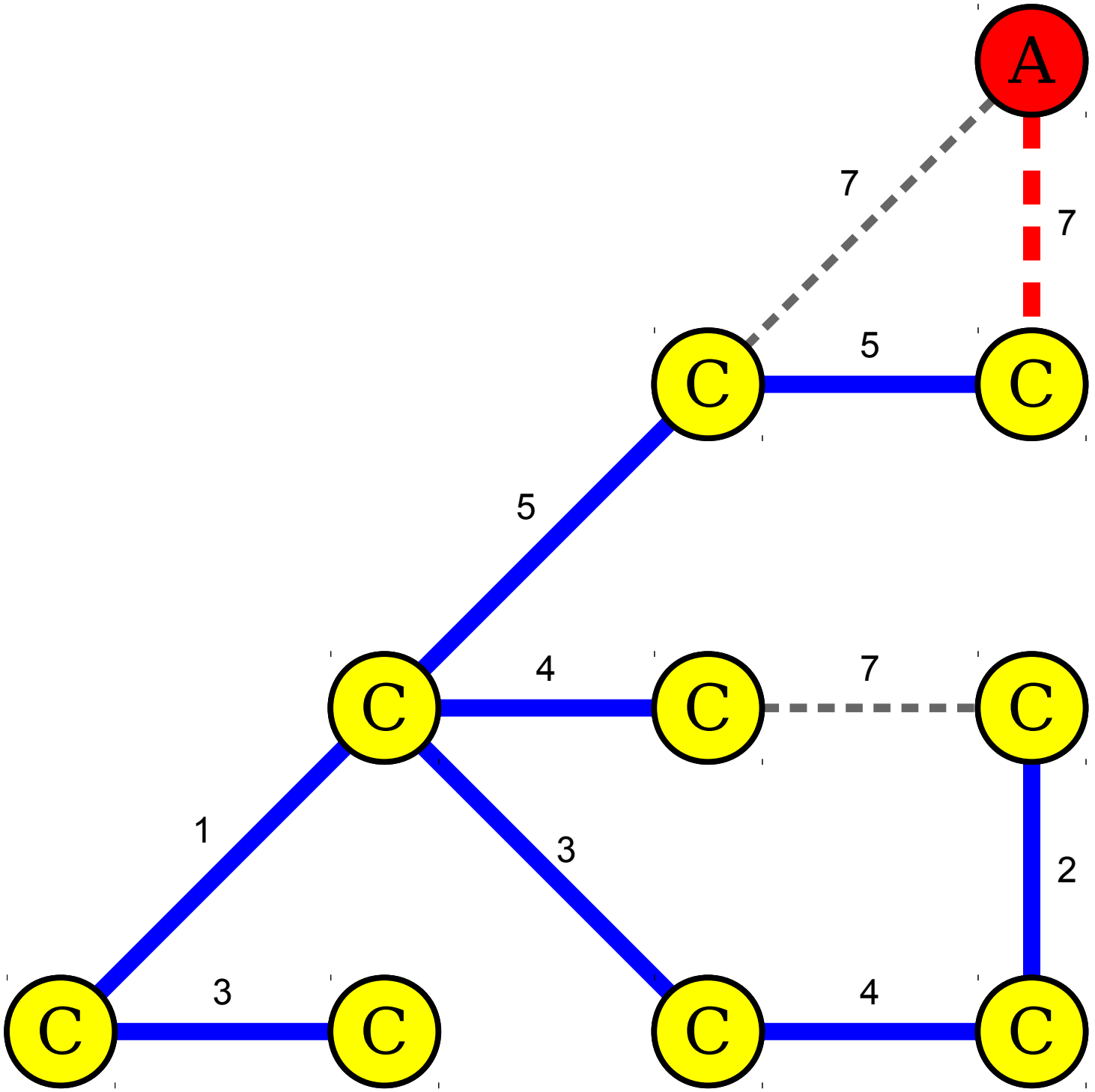

A
7
7
C
5
C
5
6
C
4
C
7
C
1
3
6
2
C
3
C
C
4
C

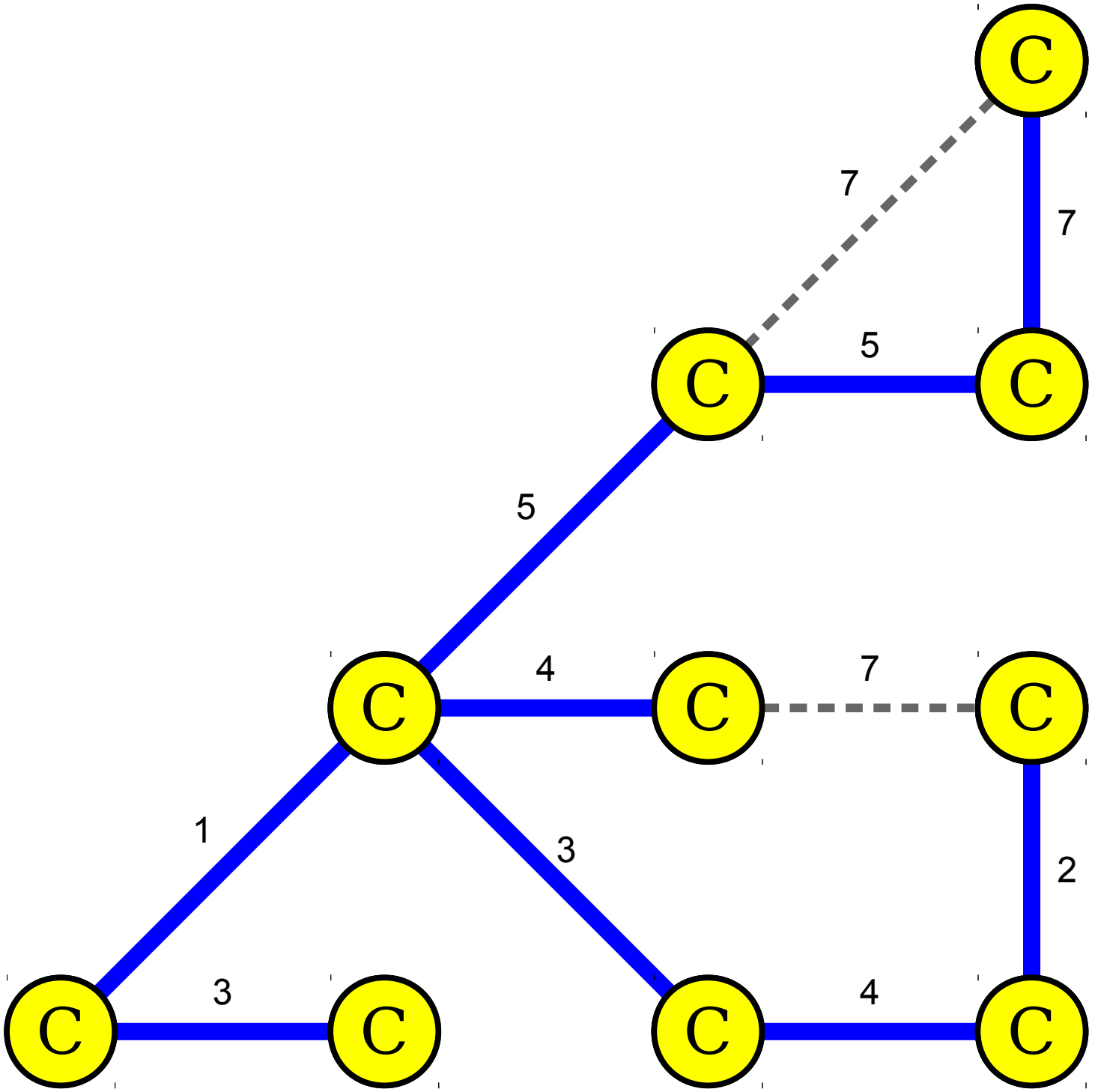

A
7
7
C
5
C
5
6
C
4
C
7
C
1
3
6
2
C
3
C
C
4
C

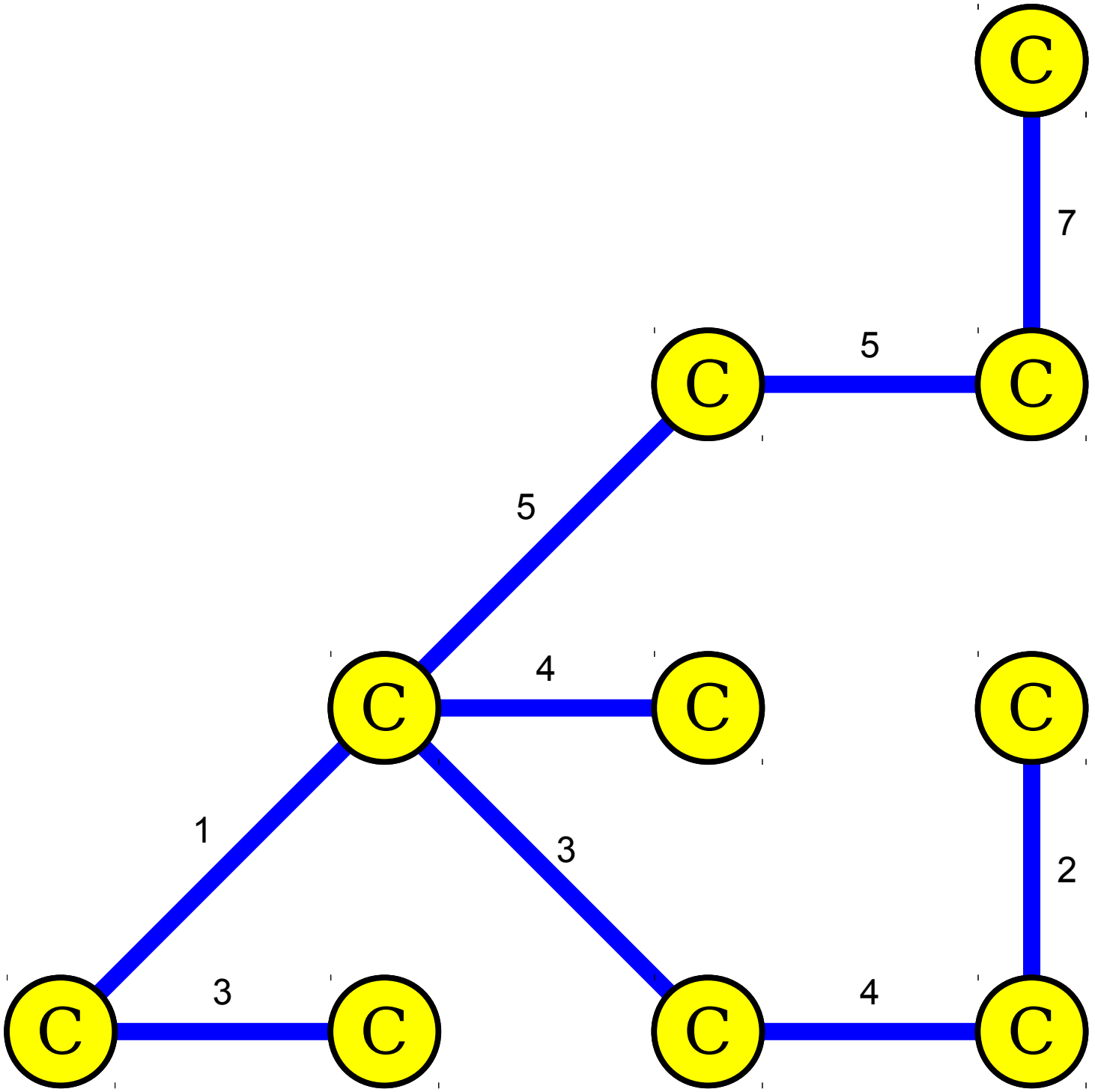

A
7
7
C
5
C
5
6
C
4
C
7
C
1
3
2
C
3
C
C
4
C


A
7
7
C
5
C
5
6
C
4
C
7
C
1
3
2
C
3
C
C
4
C


A
7
7
C
5
C
5
C
4
C
7
C
1
3
2
C
3
C
C
4
C


A
7
7
C
5
C
5
C
4
C
7
C
1
3
2
C
3
C
C
4
C


C
C
C
C
C
C
C
C
C
7
7
5
5
4
7
1
3
2
3
4


C
C
C
C
C
C
C
C
C
7
5
5
4
1
3
2
3
4

Implementing Kruskal's Algorithm

- Place every node into its own cluster.
- Place all edges into a priority queue.
- While there are two or more clusters remaining:
  - Dequeue an edge from the priority queue.
  - If its endpoints are not in the same cluster:
    - Merge the clusters containing the endpoints.
    - Add the edge to the resulting spanning tree.
- Return the resulting spanning tree.

Time-Out for Announcements!

Final Exam Logistics

- Final: Monday, March 20th, 8:30AM – 11:30AM, location TBA.
- Format is same as the midterm: closed-book, closed-computer, limited-note. You get a single, double-sided sheet of 8.5” × 11” notes decorated however you’d like.
- Cumulative exam, slightly focused on the post-midterm topics.
  - Covers topics from all assignments from this quarter.
  - Covers topics from lectures up through and including today.
- We will be holding a practice exam ***tonight*** in Hewlett 200 from 7PM – 10PM. The format of the practice final is similar to the format of the actual final exam.
- Have OAE accommodations? We’ll reach out to you soon to coordinate alternate exams.

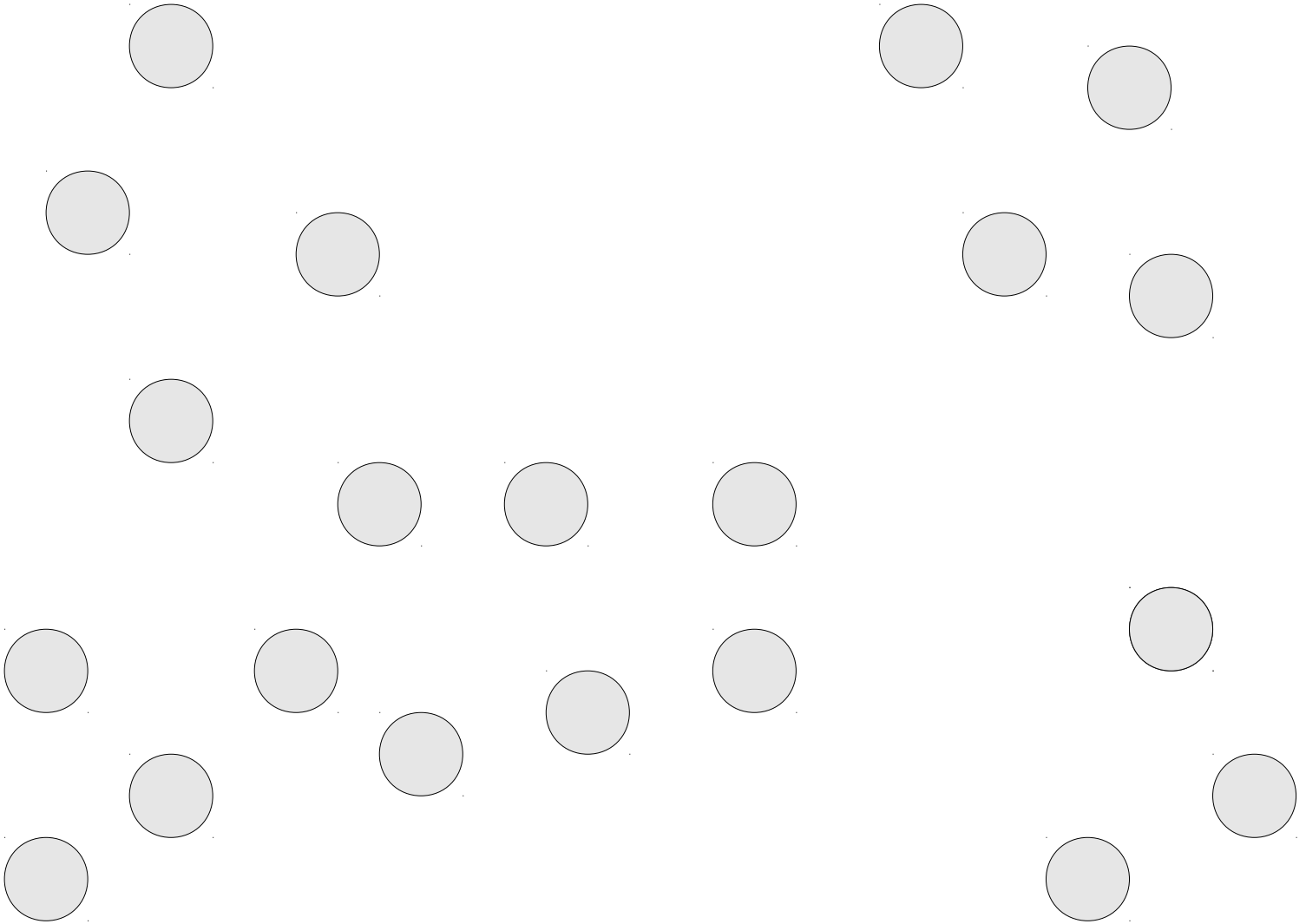
Assignment 7

- Assignment 7 is due on Friday.
  - Recommendation 1: try to complete BFS and Dijkstra's algorithm by the end of the evening.
  - Recommendation 2: try to complete A* search by tomorrow evening.
  - No late submissions will be accepted, ***even if you have remaining late days*** – sorry about that!

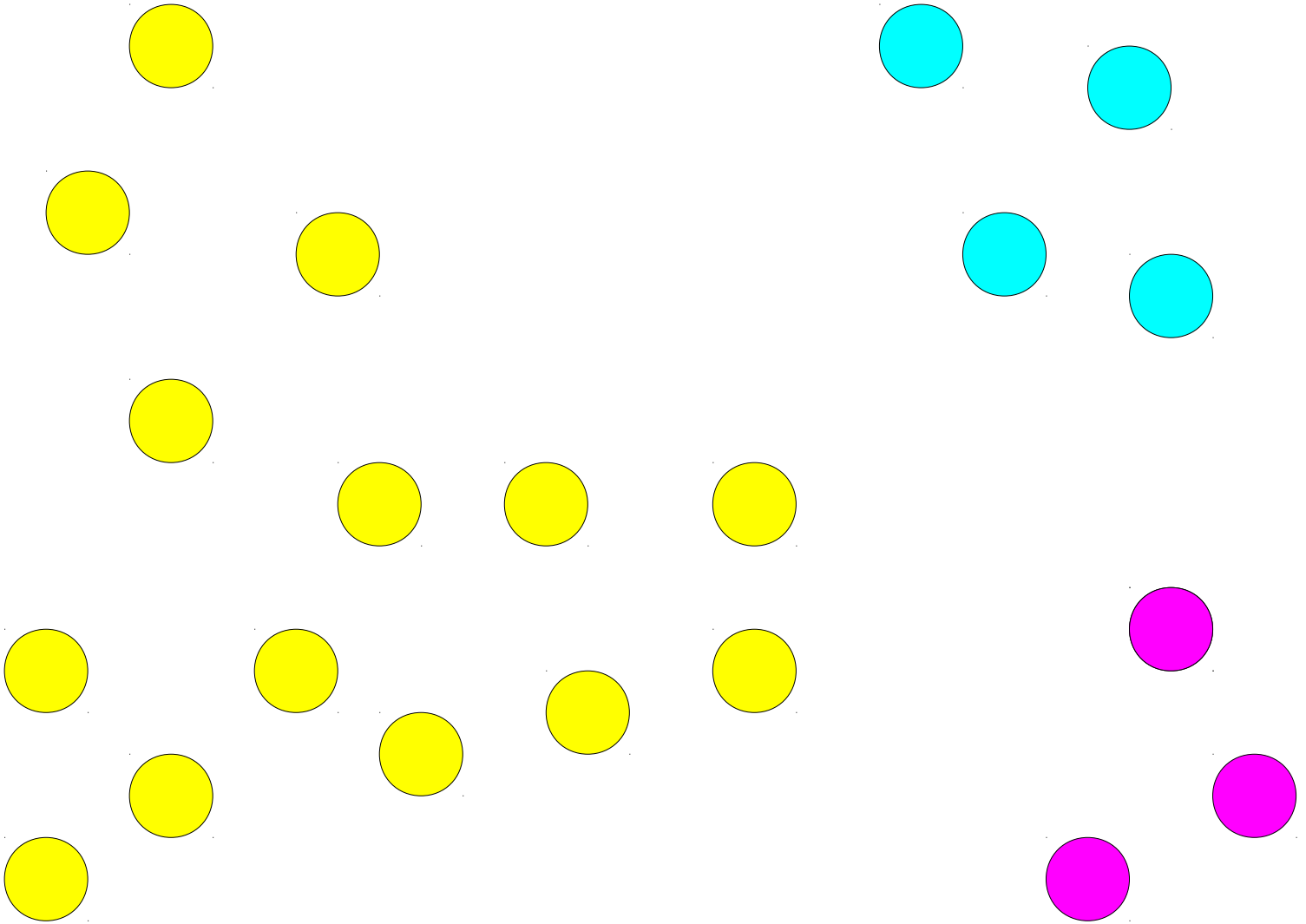
Back to CS106B!

Applications of Kruskal's Algorithm

Data Clustering



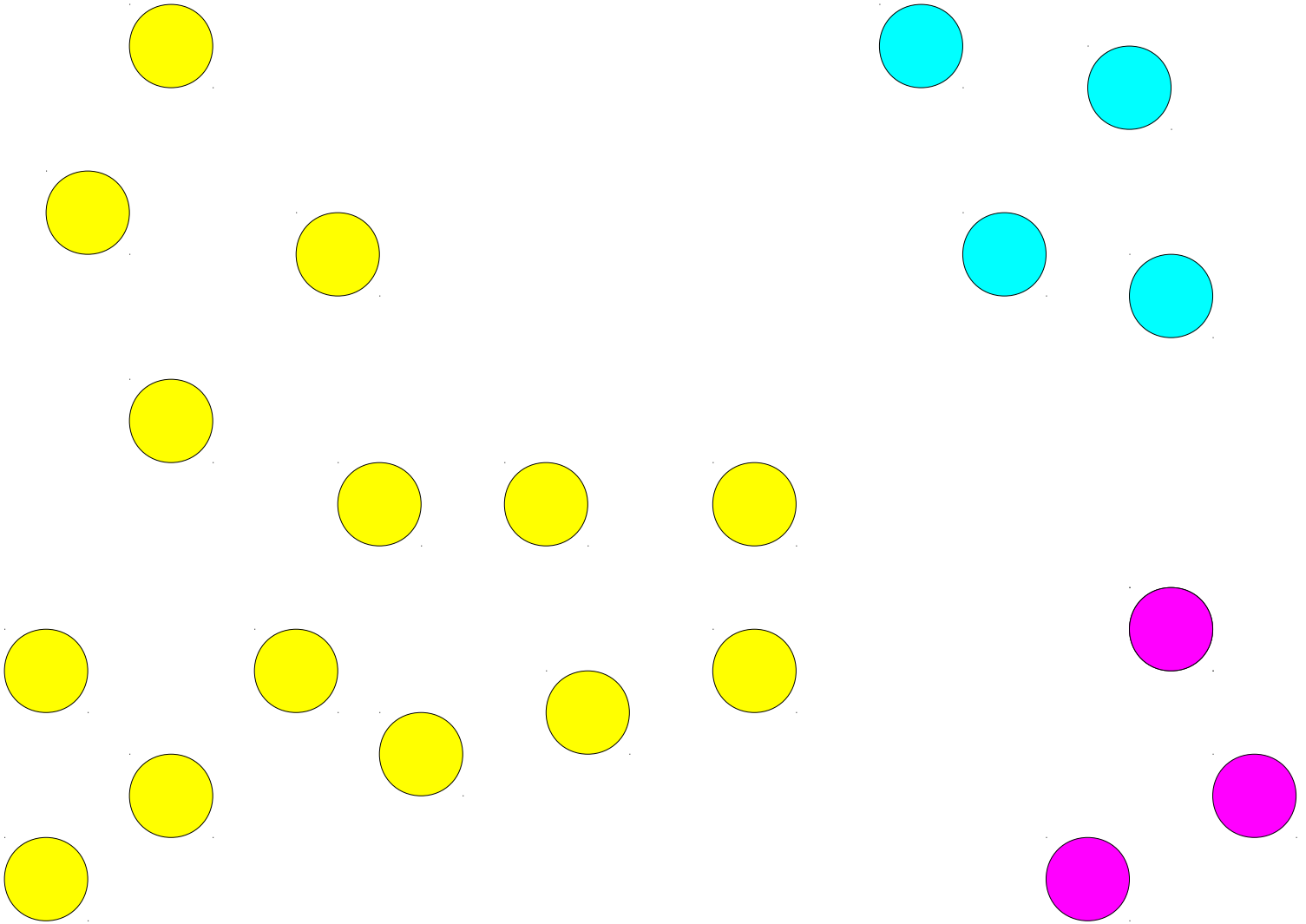
Data Clustering



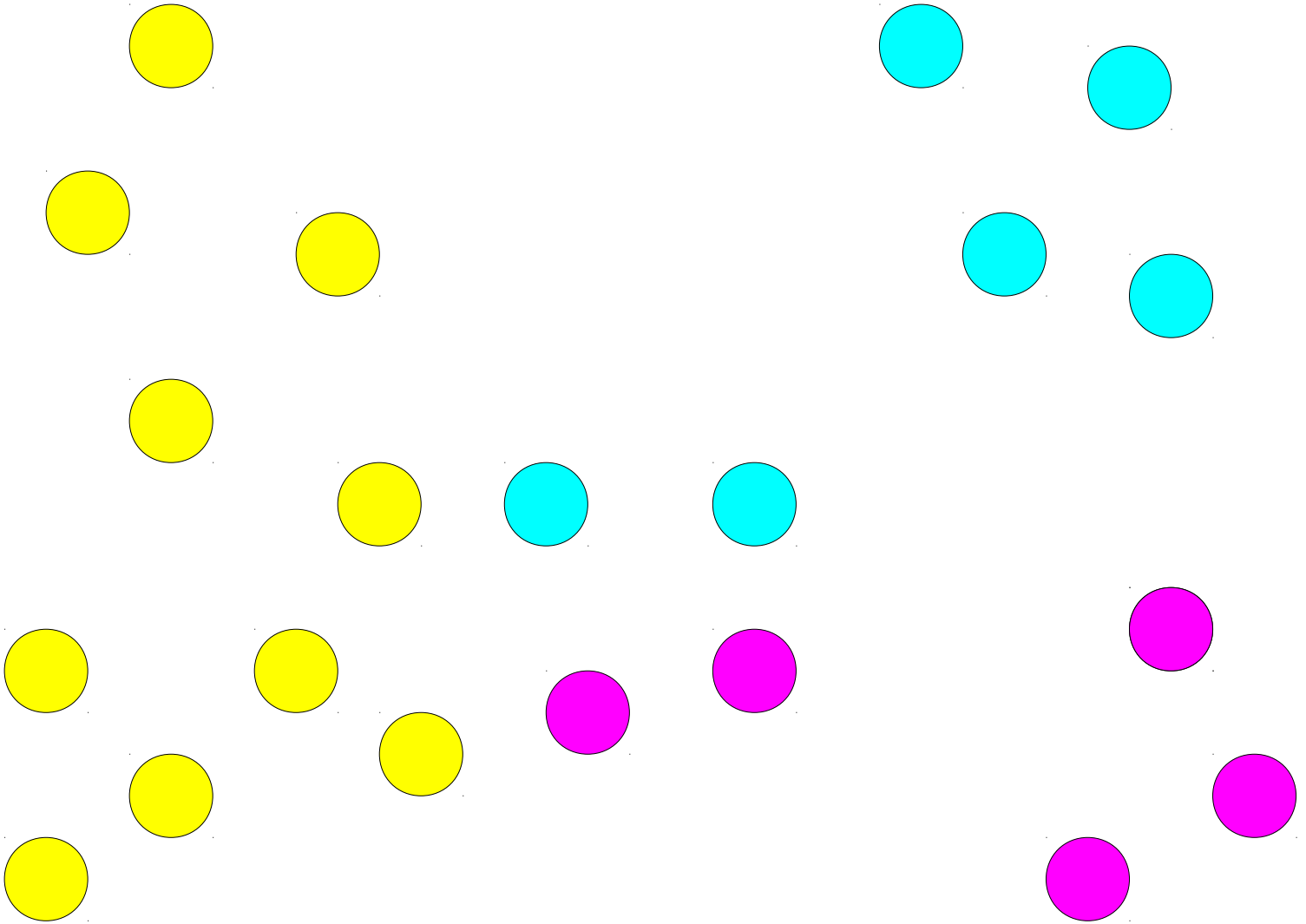
Data Clustering

- Given a set of points, break those points apart into clusters.
- Immensely useful across all disciplines:
  - Cluster individuals by phenotype to try to determine what genes influence which traits.
  - Cluster images by pixel color to identify objects in pictures.
  - Cluster essays by various features to see how students learn to write.

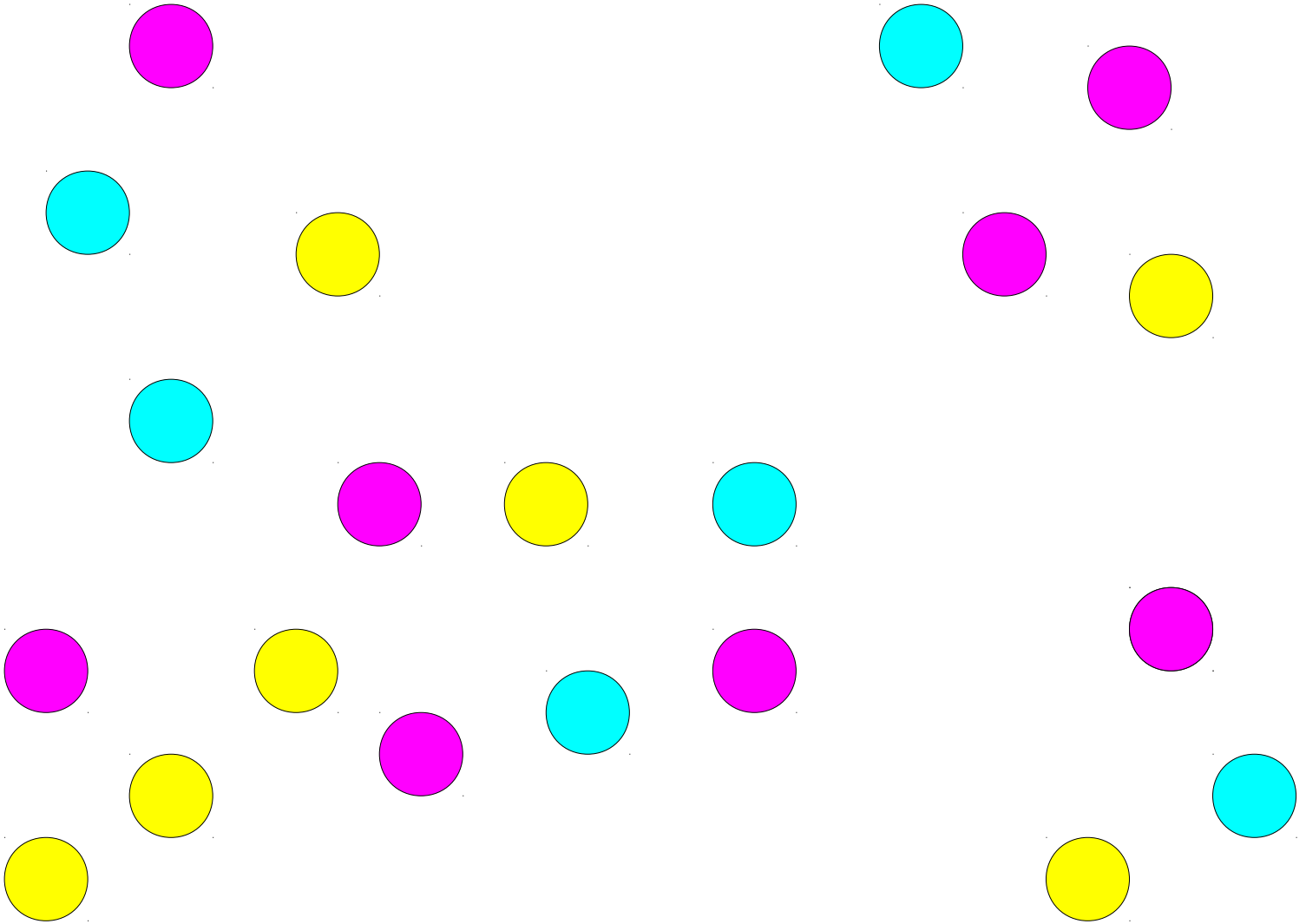
Data Clustering



Data Clustering



Data Clustering

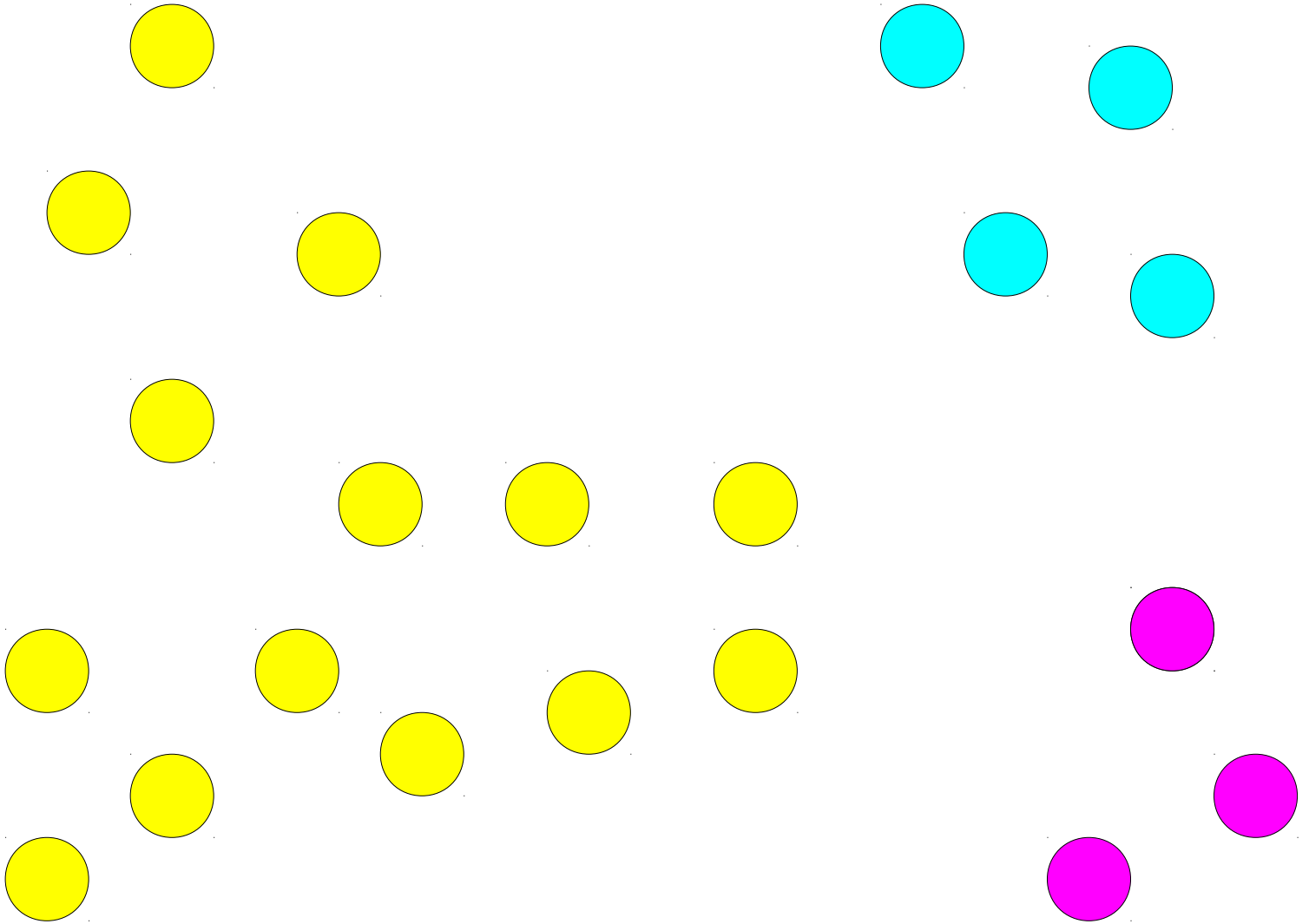


What makes a clustering “good?”

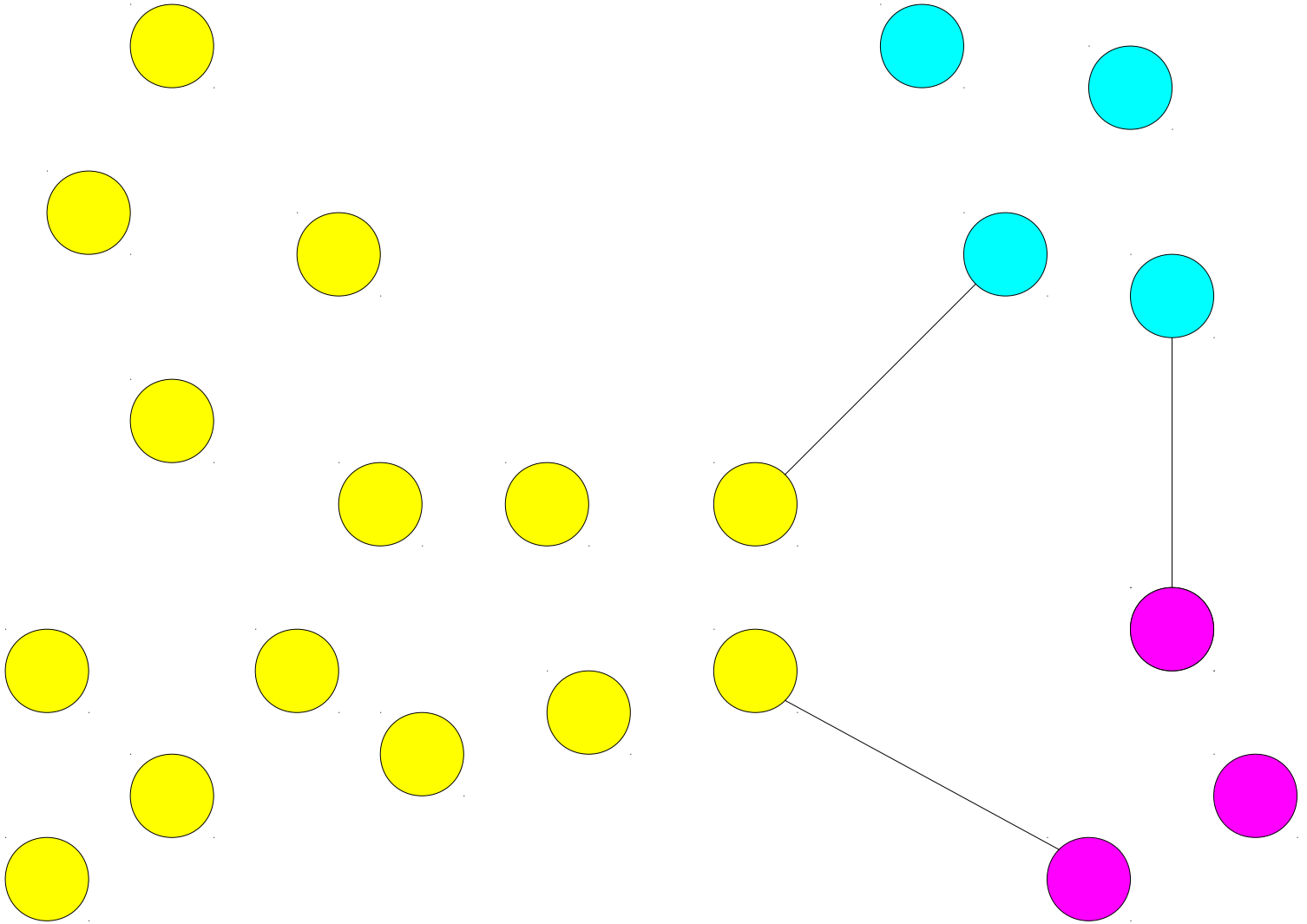
Maximum-Separation Clustering

- A ***maximum-separation clustering*** is one where the distance between the resulting clusters is as large as possible.
- Specifically, it maximizes the minimum distance between any two points of different clusters.
- Very good on many data sets, though not always ideal.

Maximum-Separation Clustering



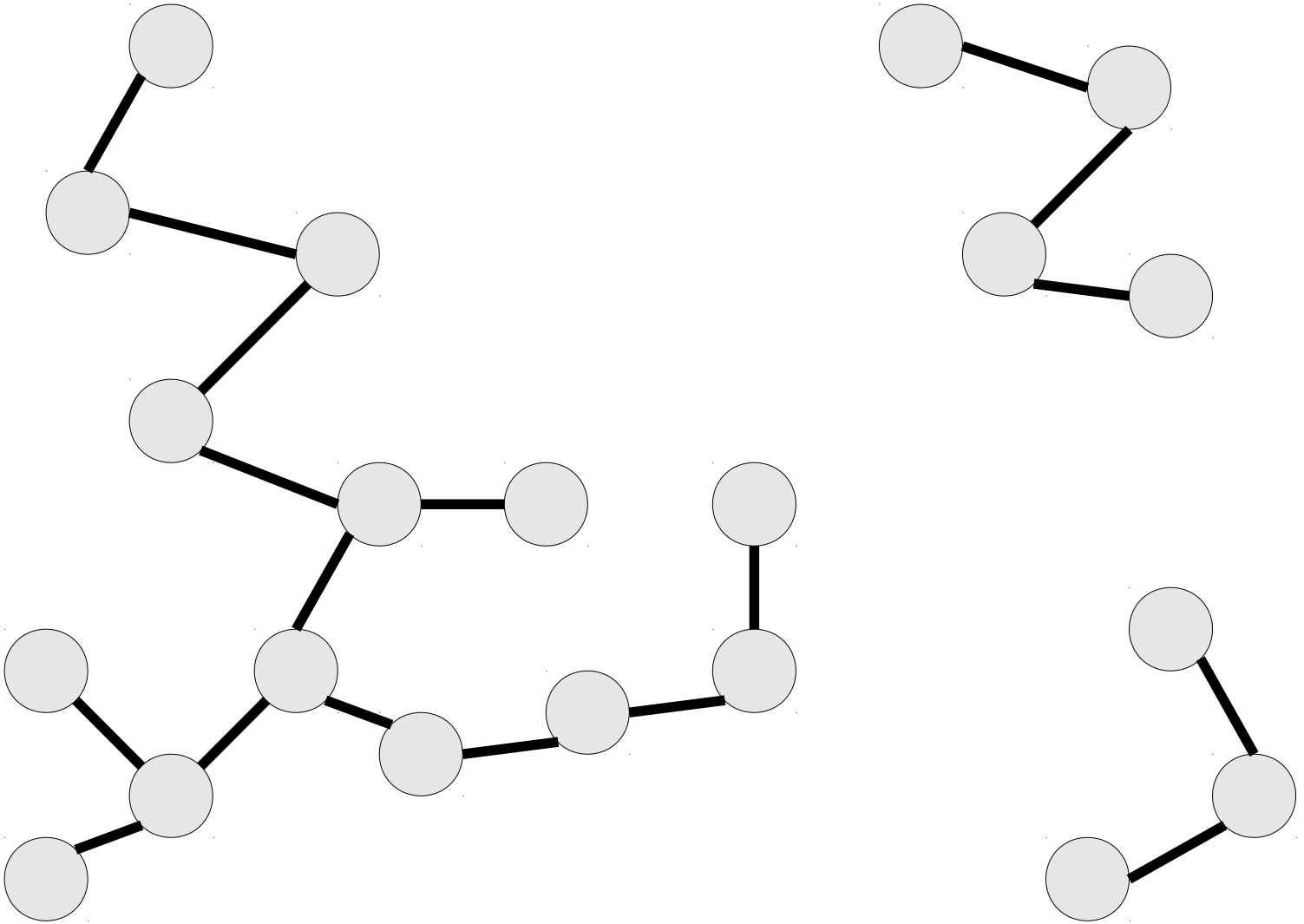
Maximum-Separation Clustering



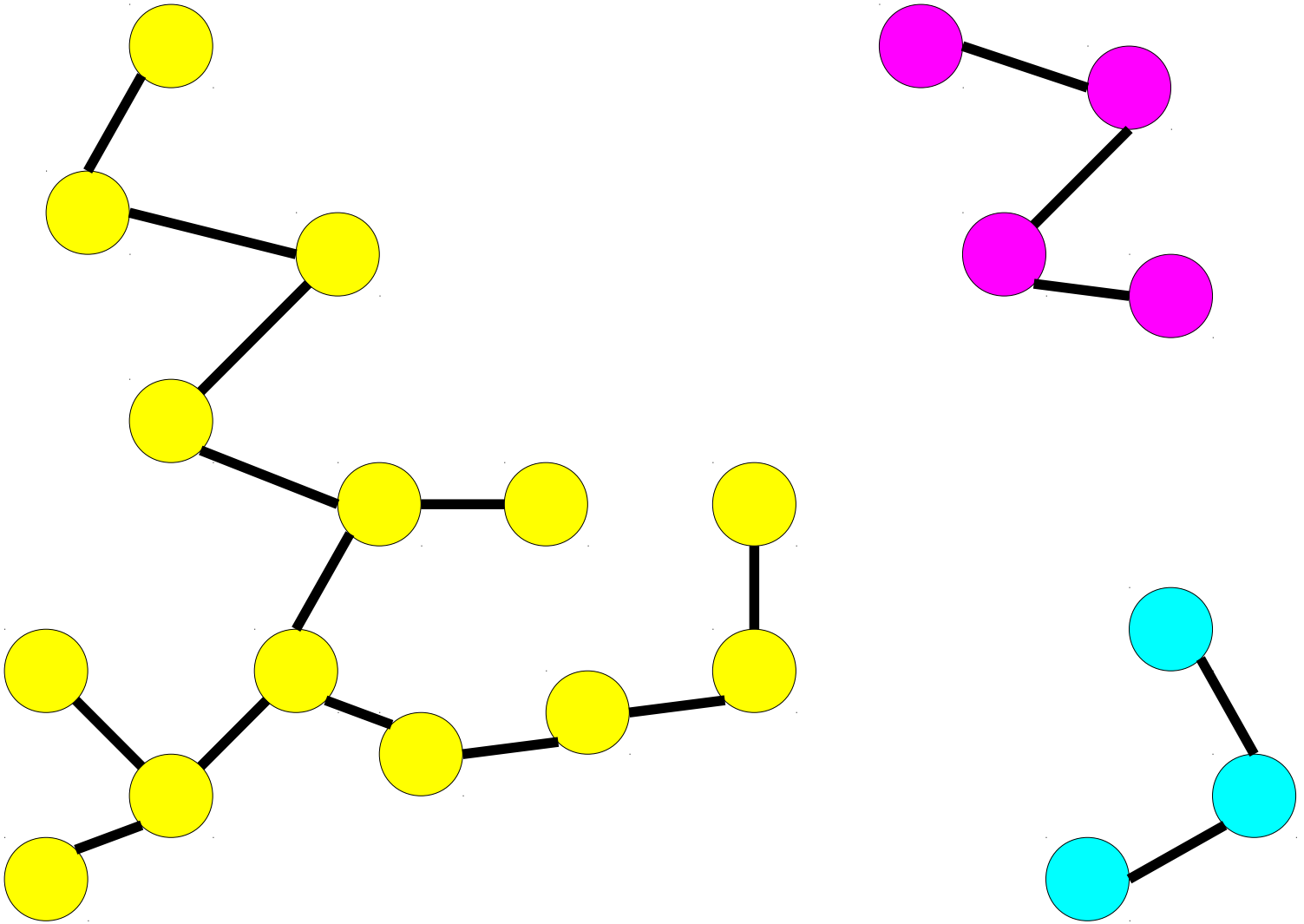
Maximum-Separation Clustering

- It is extremely easy to adopt Kruskal's algorithm to produce a maximum-separation set of clusters.
  - Suppose you want k clusters.
  - Given the data set, add an edge from each node to each other node whose length depends on their similarity.
  - Run Kruskal's algorithm until only k clusters remain.
  - The pieces of the graph that have been linked together are k maximally-separated clusters.

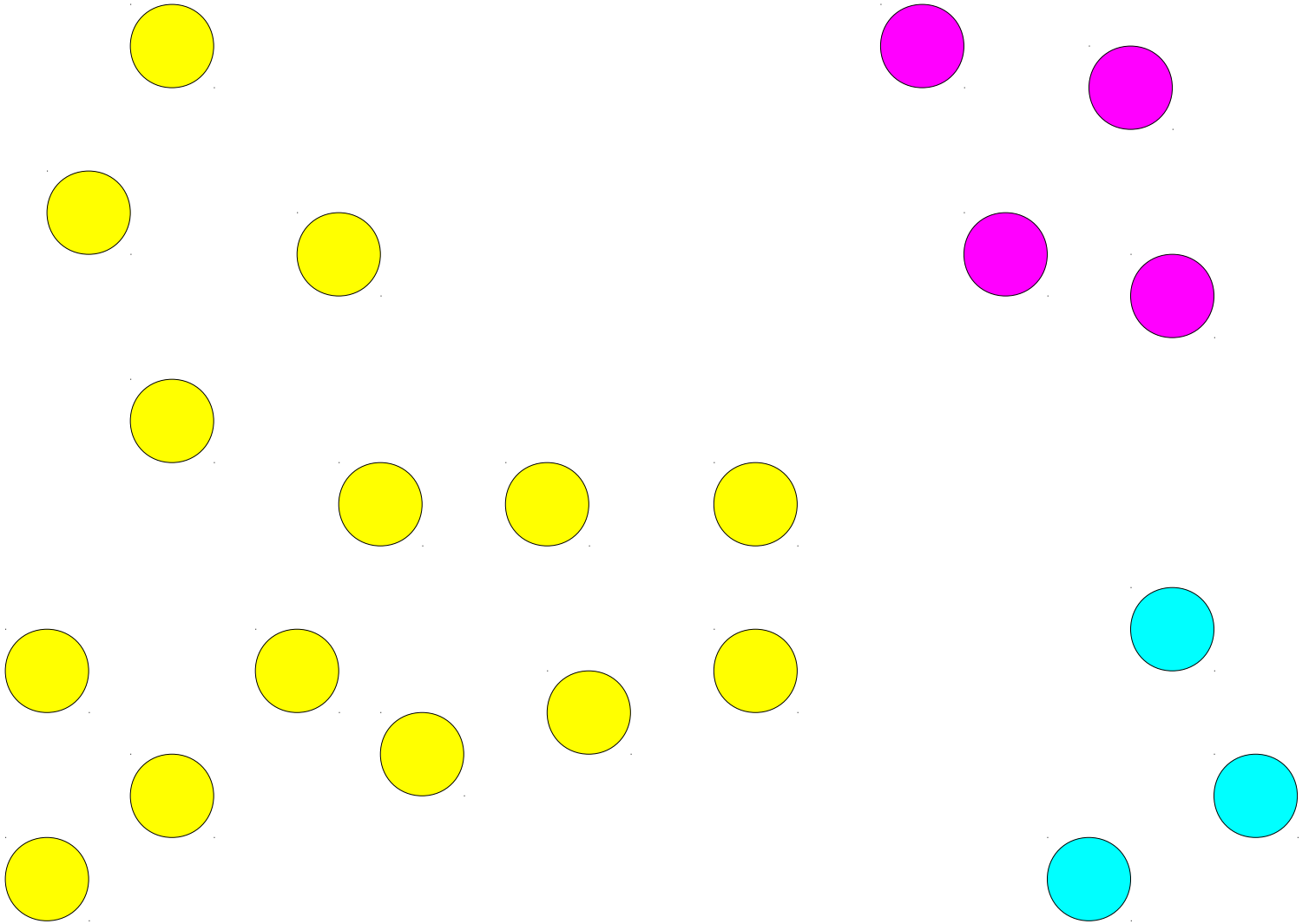
Maximum-Separation Clustering



Maximum-Separation Clustering



Maximum-Separation Clustering



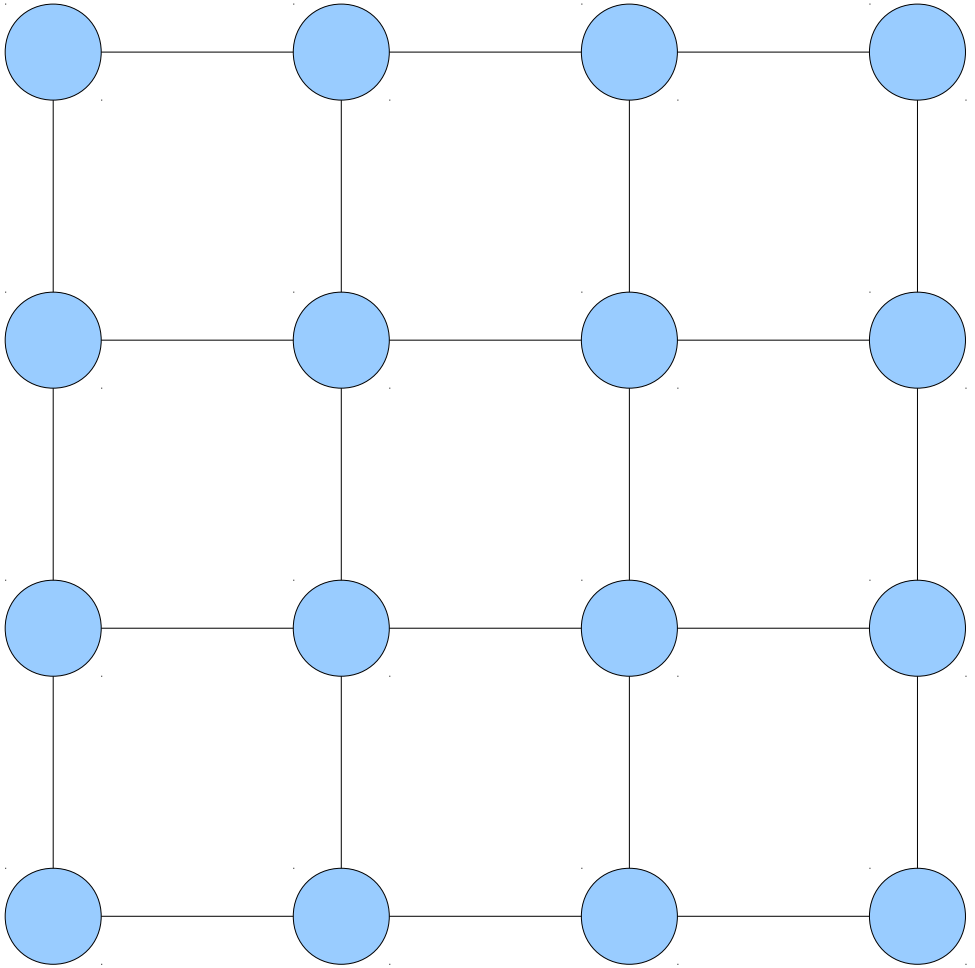
Want to learn more about clustering?

Take CS246!

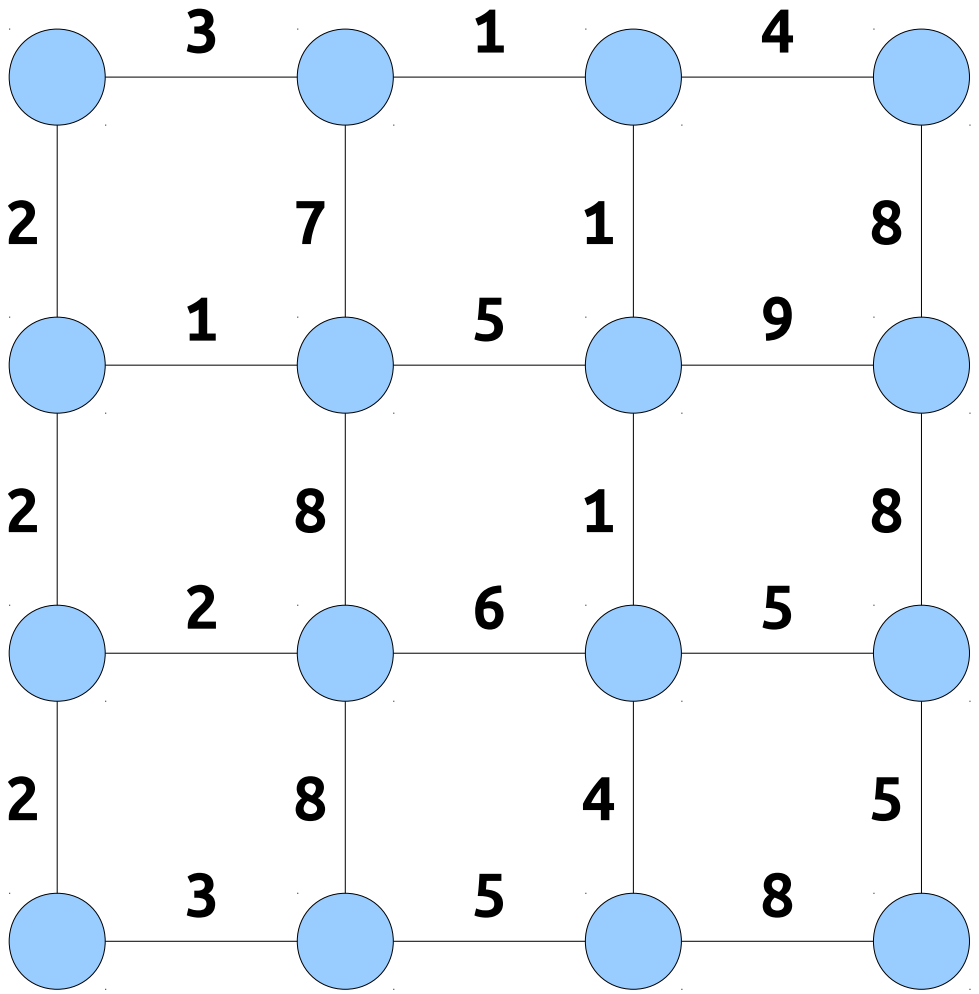
Another Application

Mazes with Kruskal's Algorithm

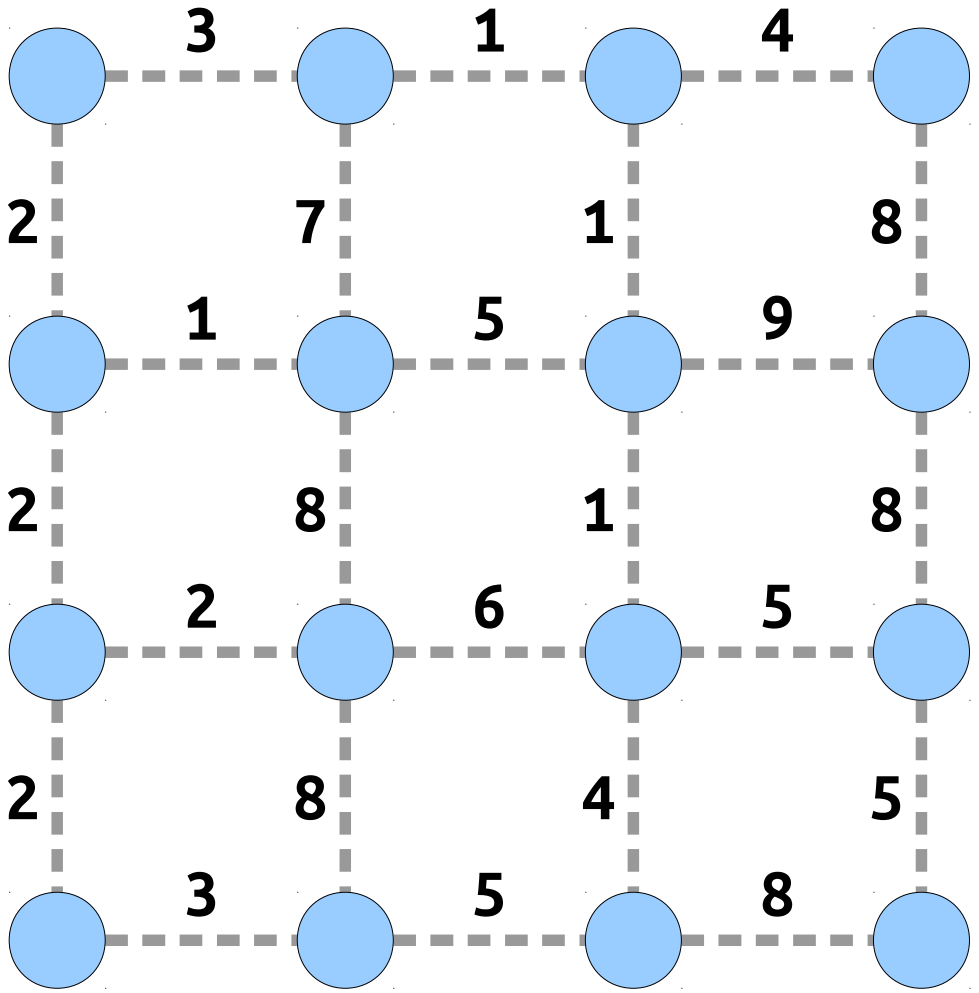
Mazes with Kruskal's Algorithm



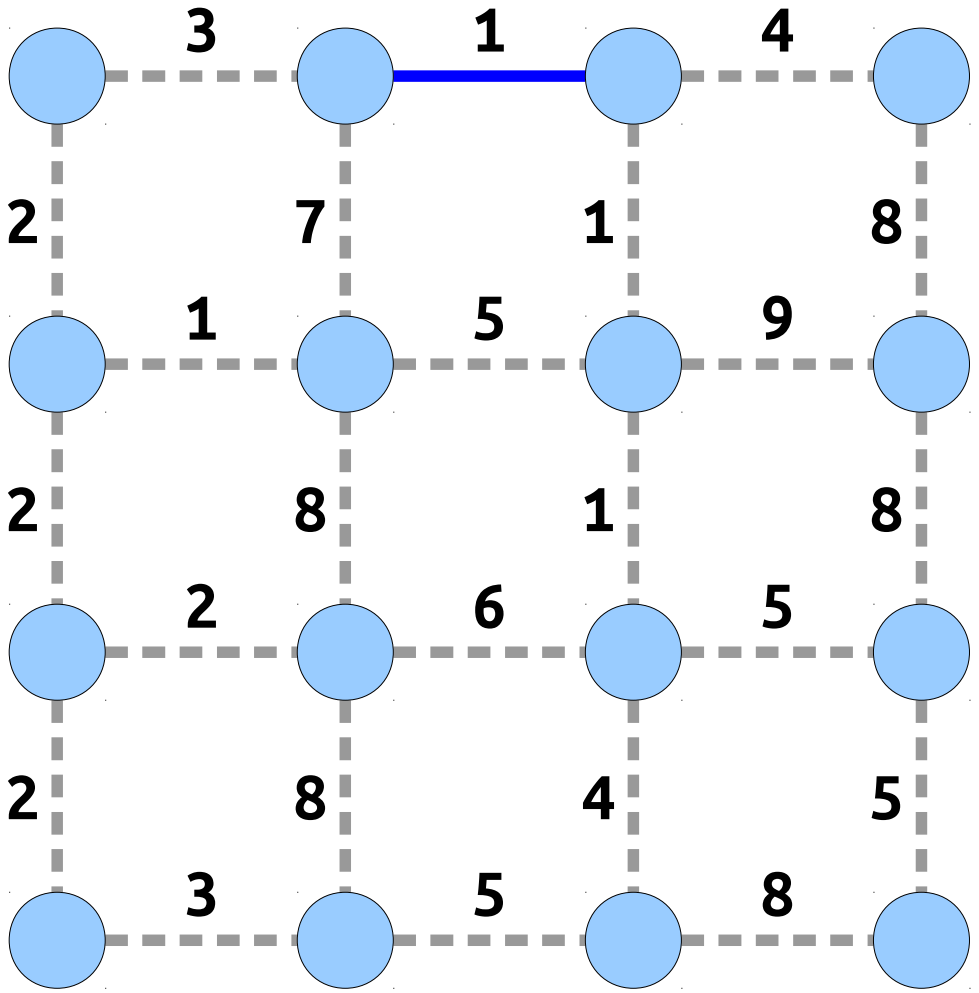
Mazes with Kruskal's Algorithm



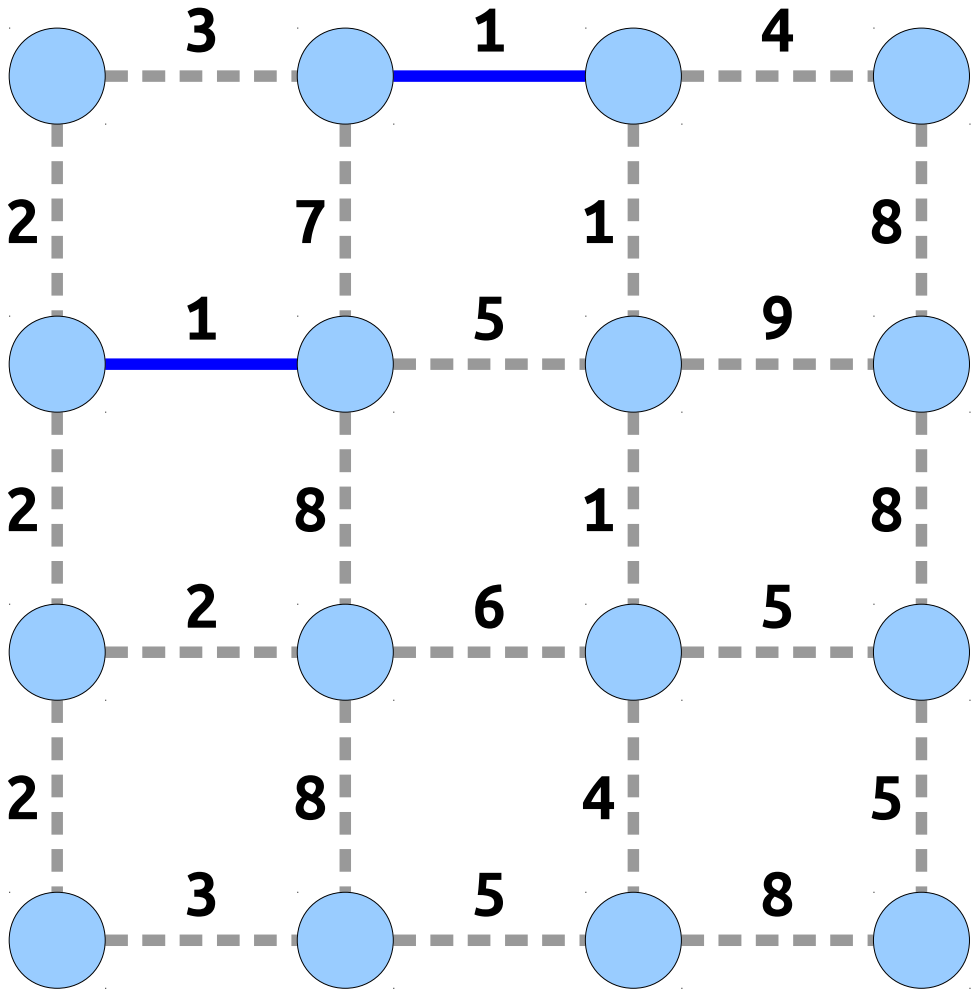
Mazes with Kruskal's Algorithm



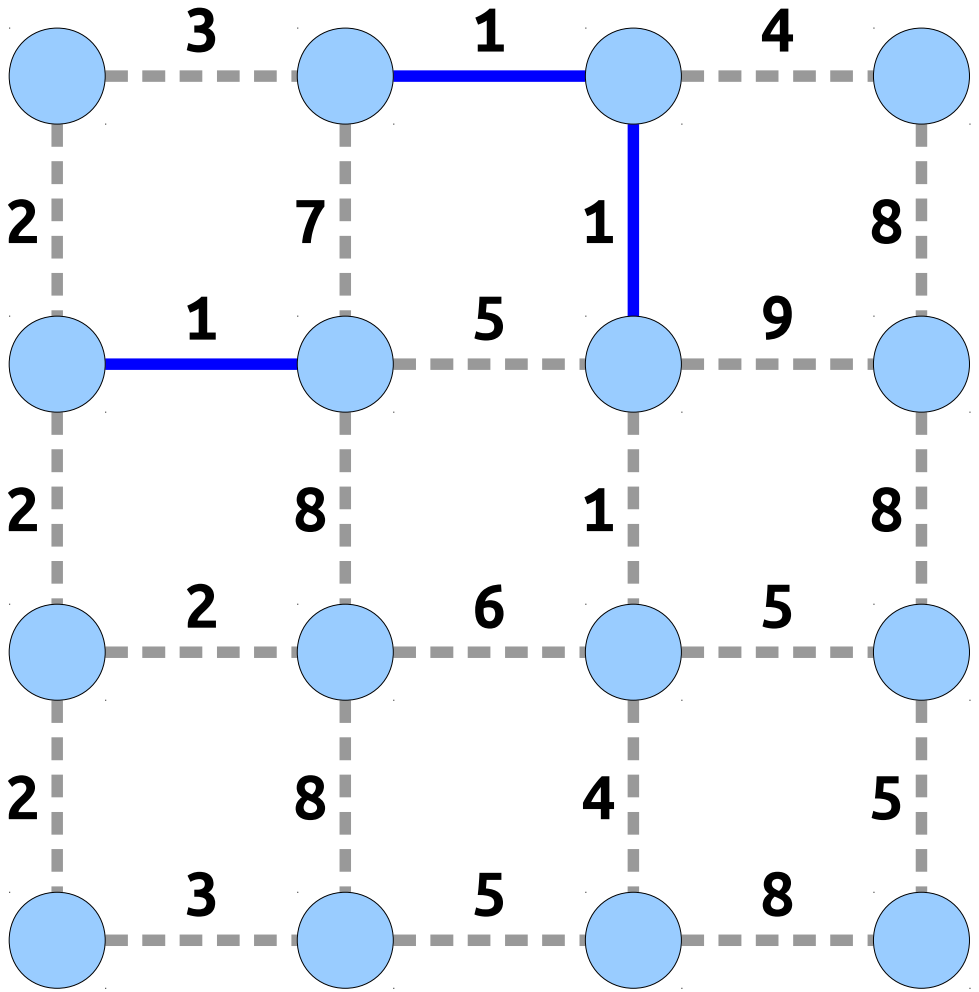
Mazes with Kruskal's Algorithm



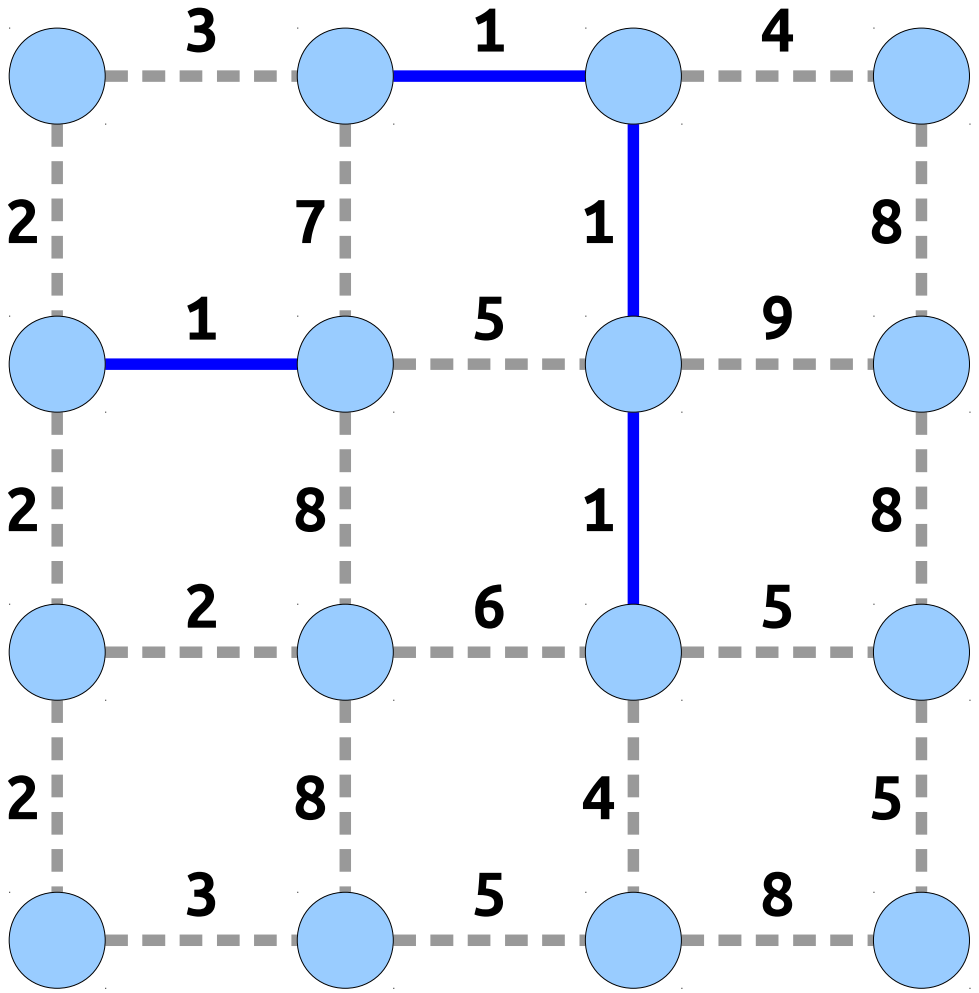
Mazes with Kruskal's Algorithm



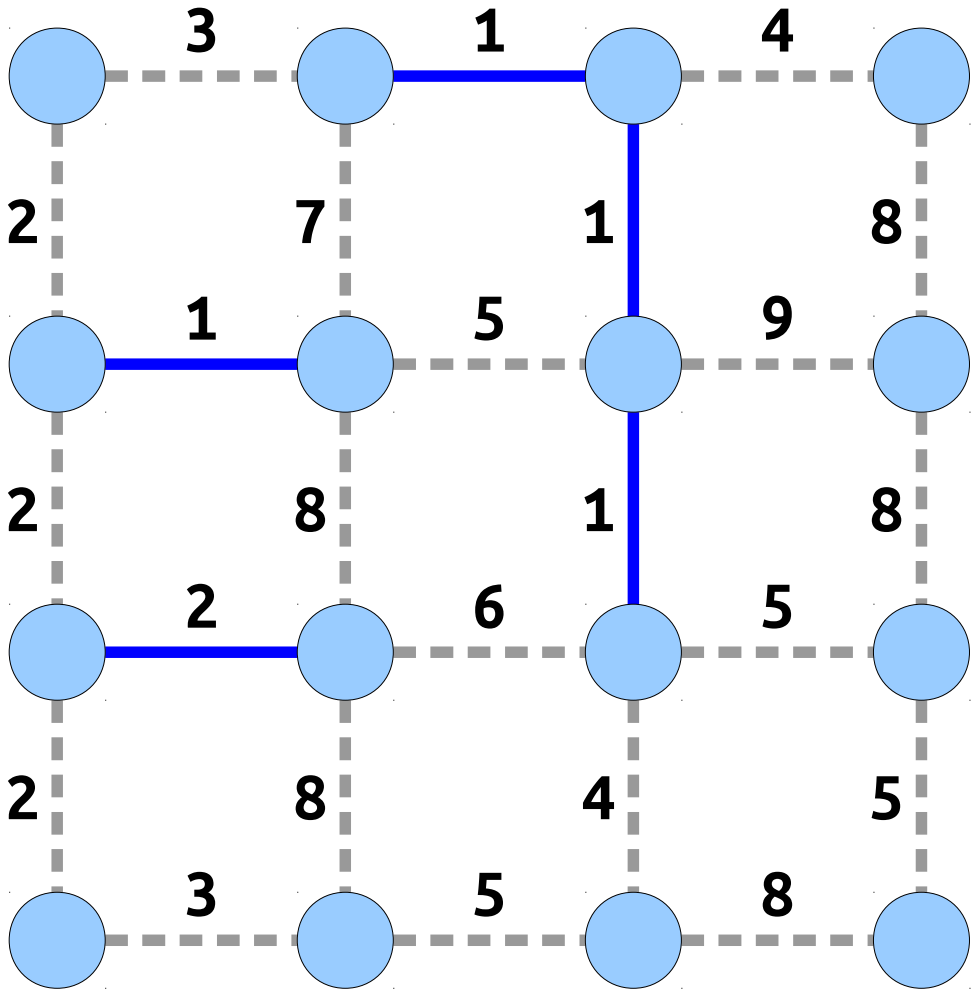
Mazes with Kruskal's Algorithm



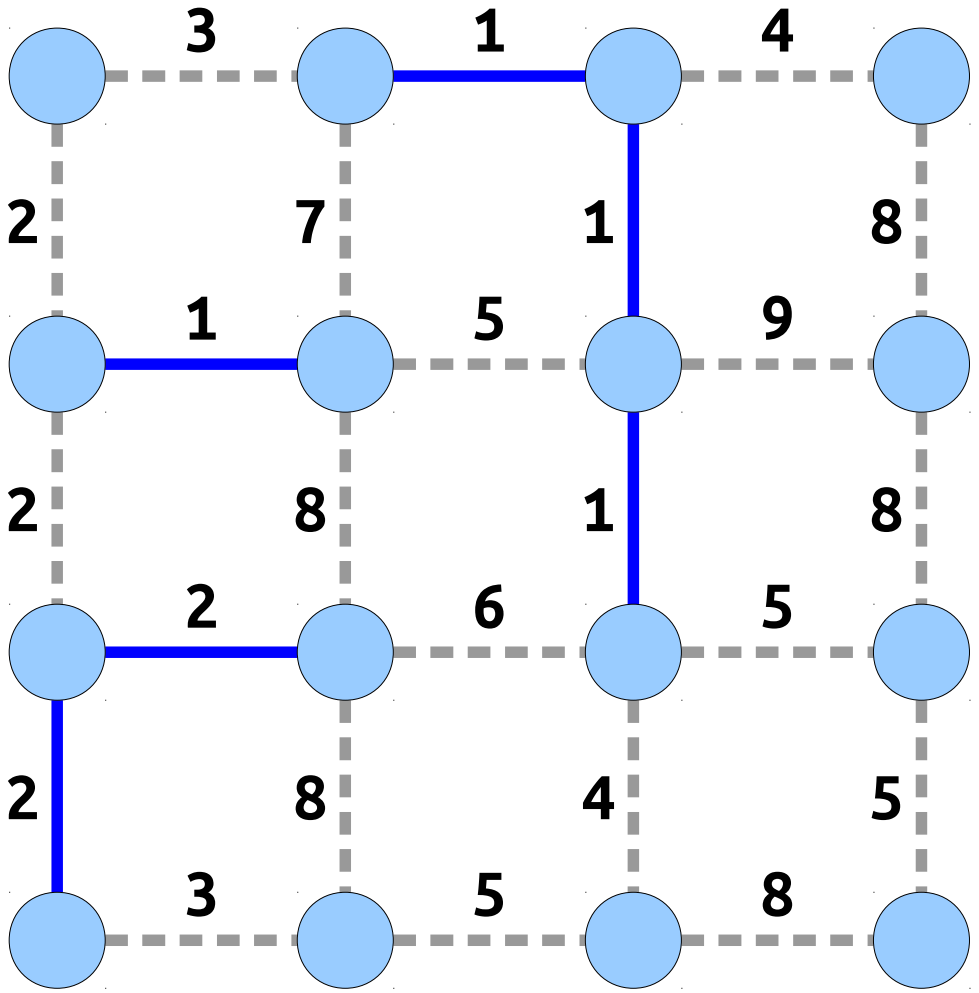
Mazes with Kruskal's Algorithm



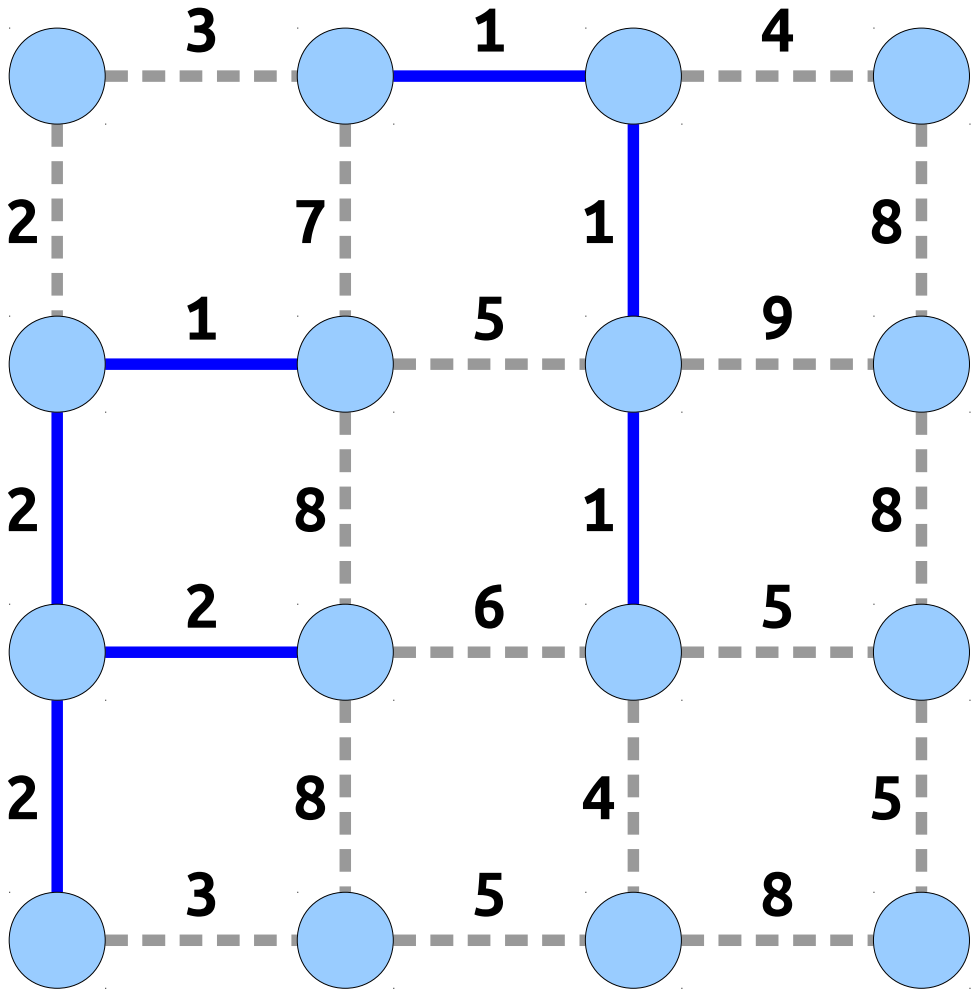
Mazes with Kruskal's Algorithm



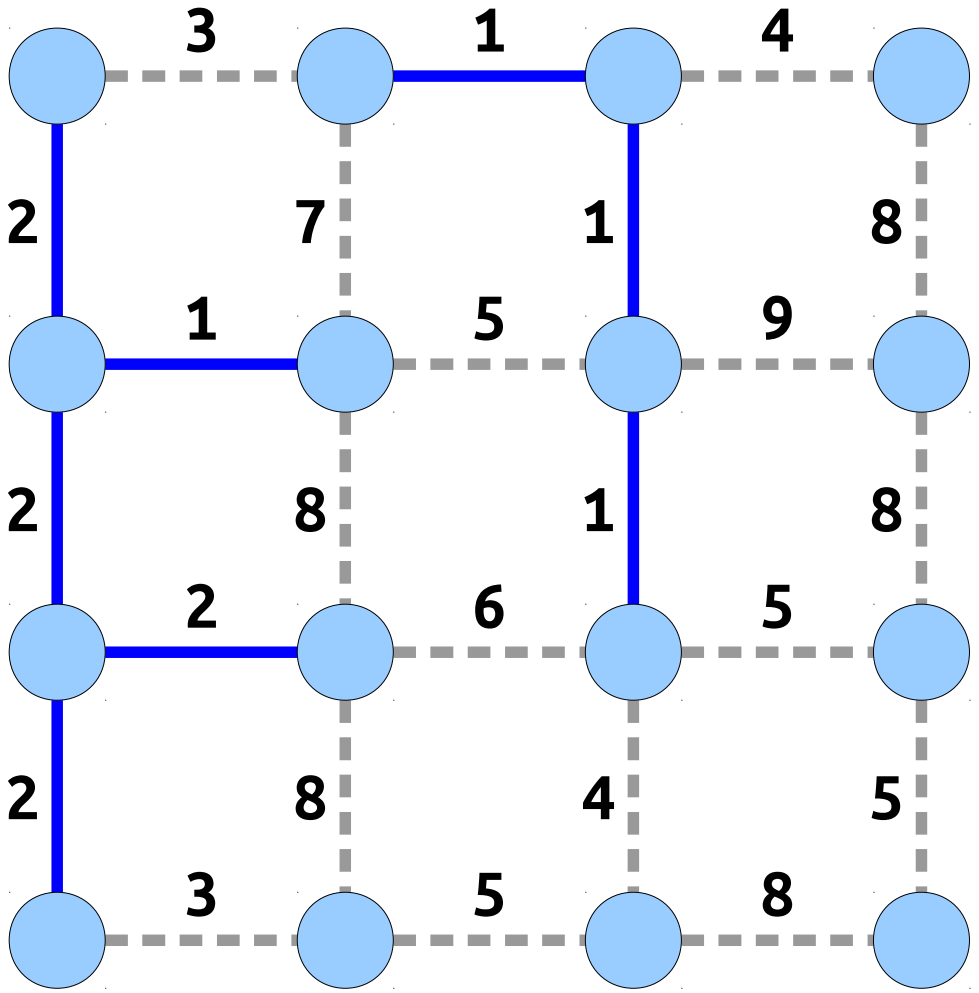
Mazes with Kruskal's Algorithm



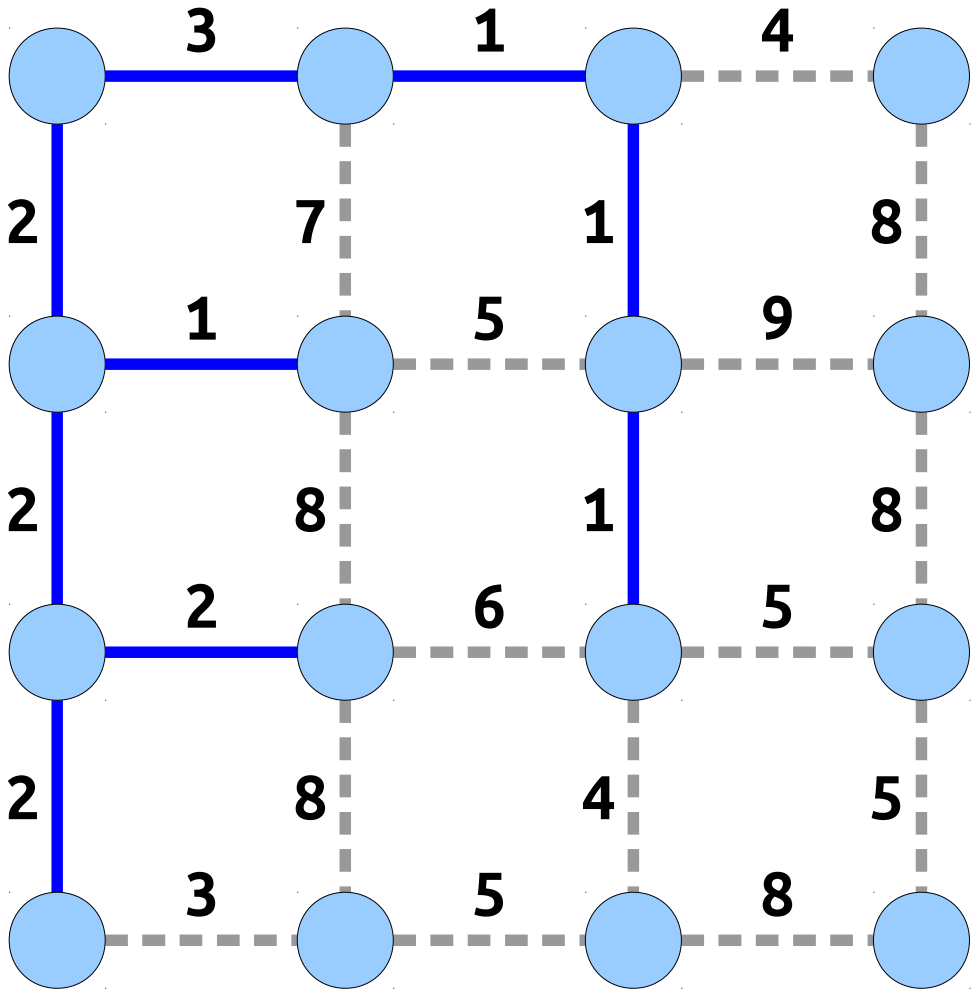
Mazes with Kruskal's Algorithm



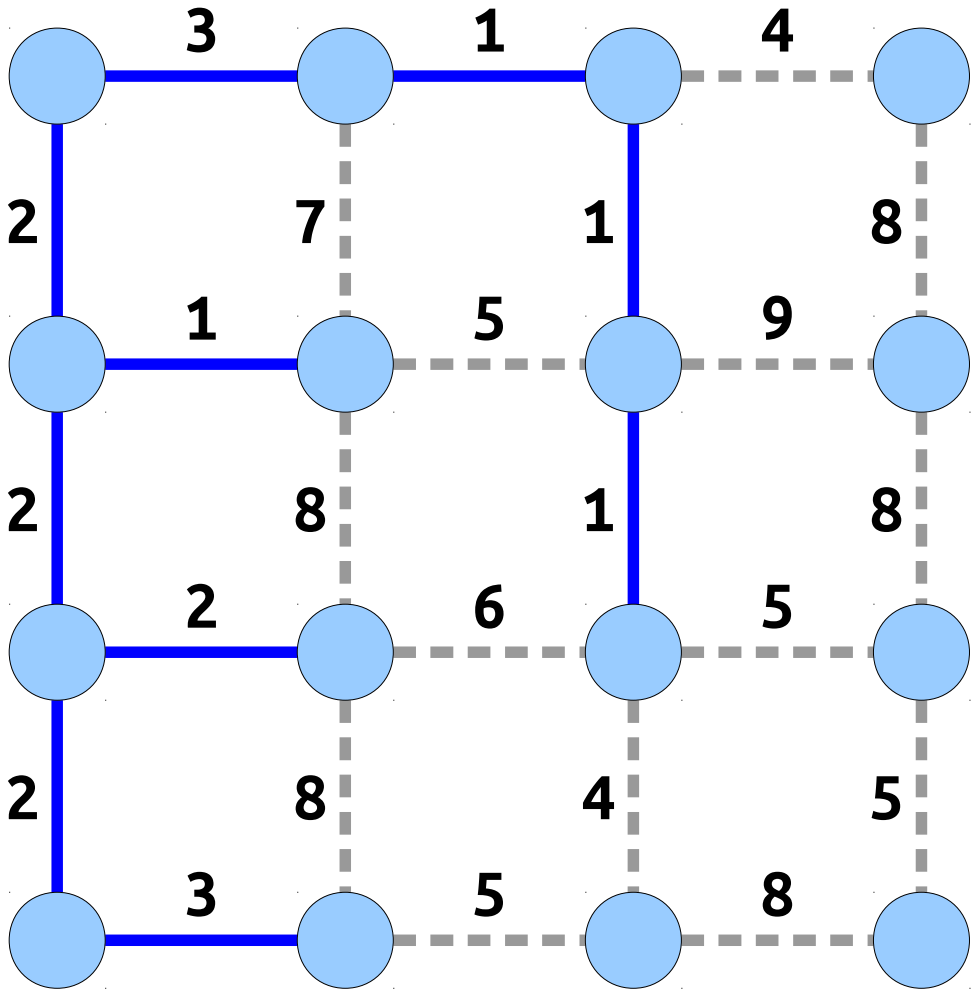
Mazes with Kruskal's Algorithm



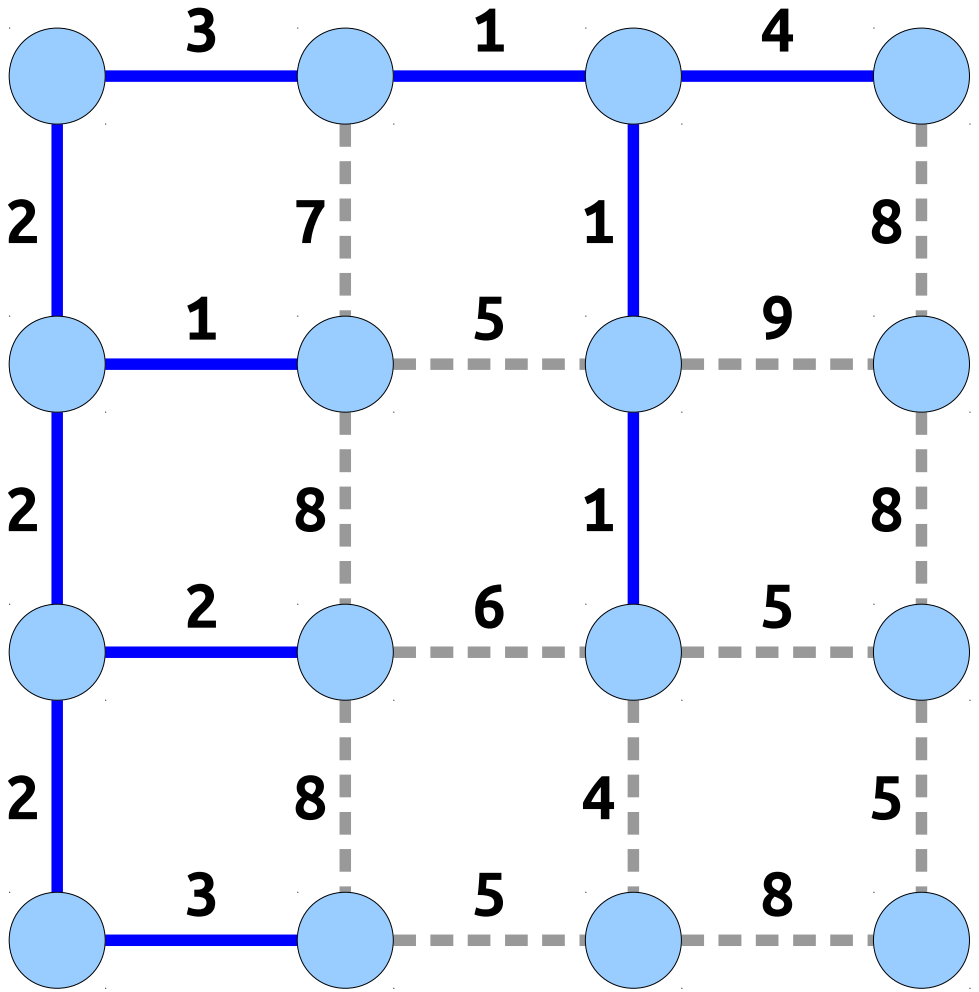
Mazes with Kruskal's Algorithm



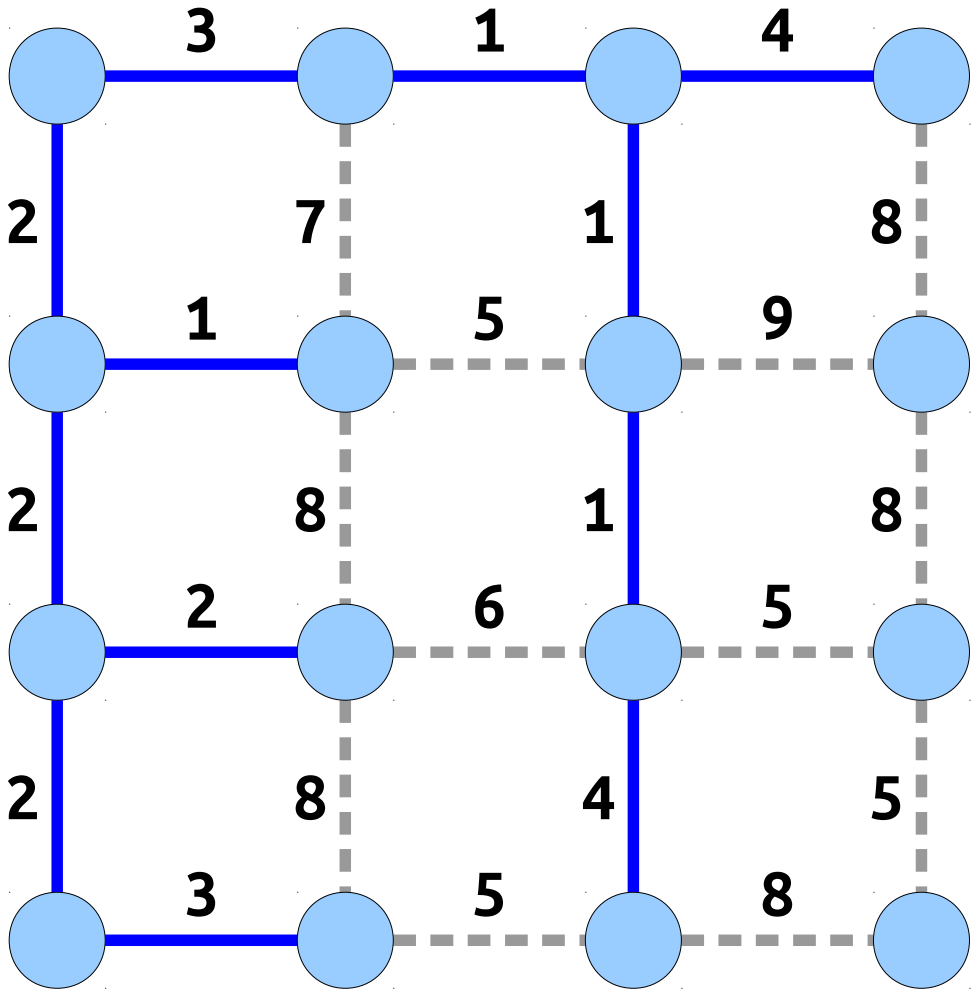
Mazes with Kruskal's Algorithm



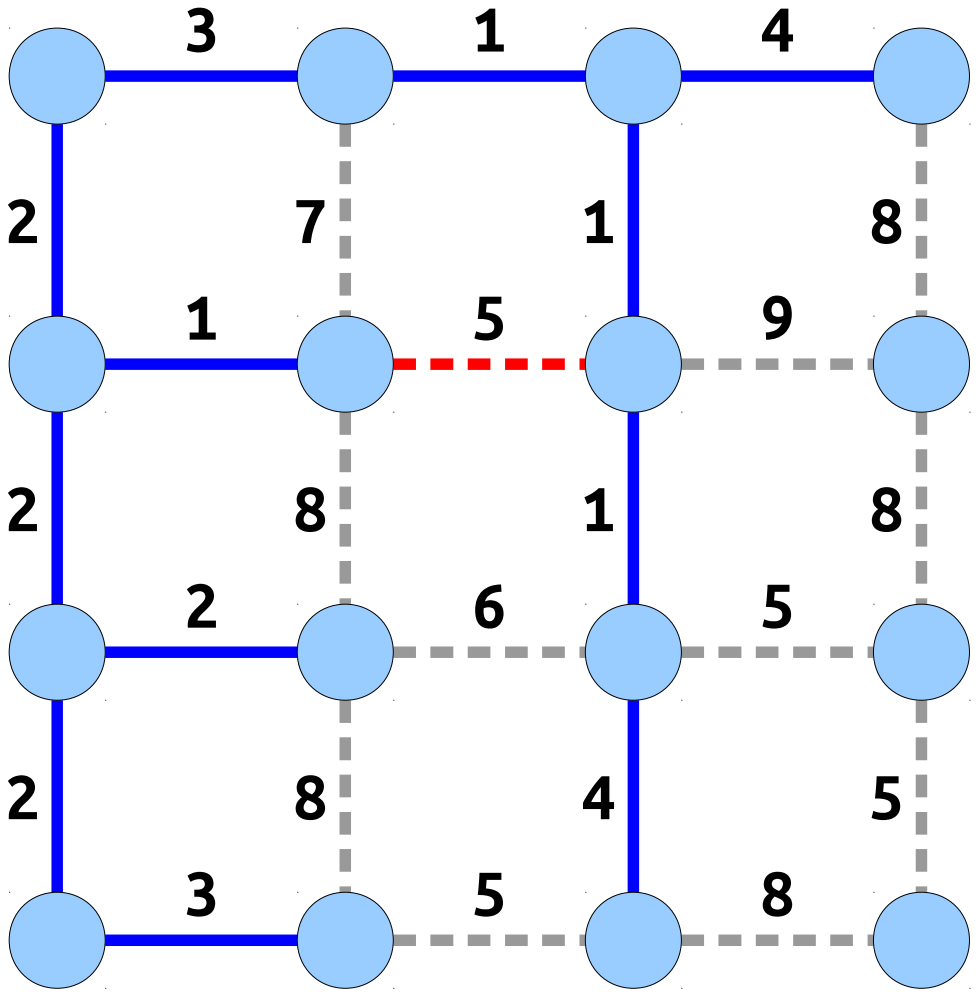
Mazes with Kruskal's Algorithm



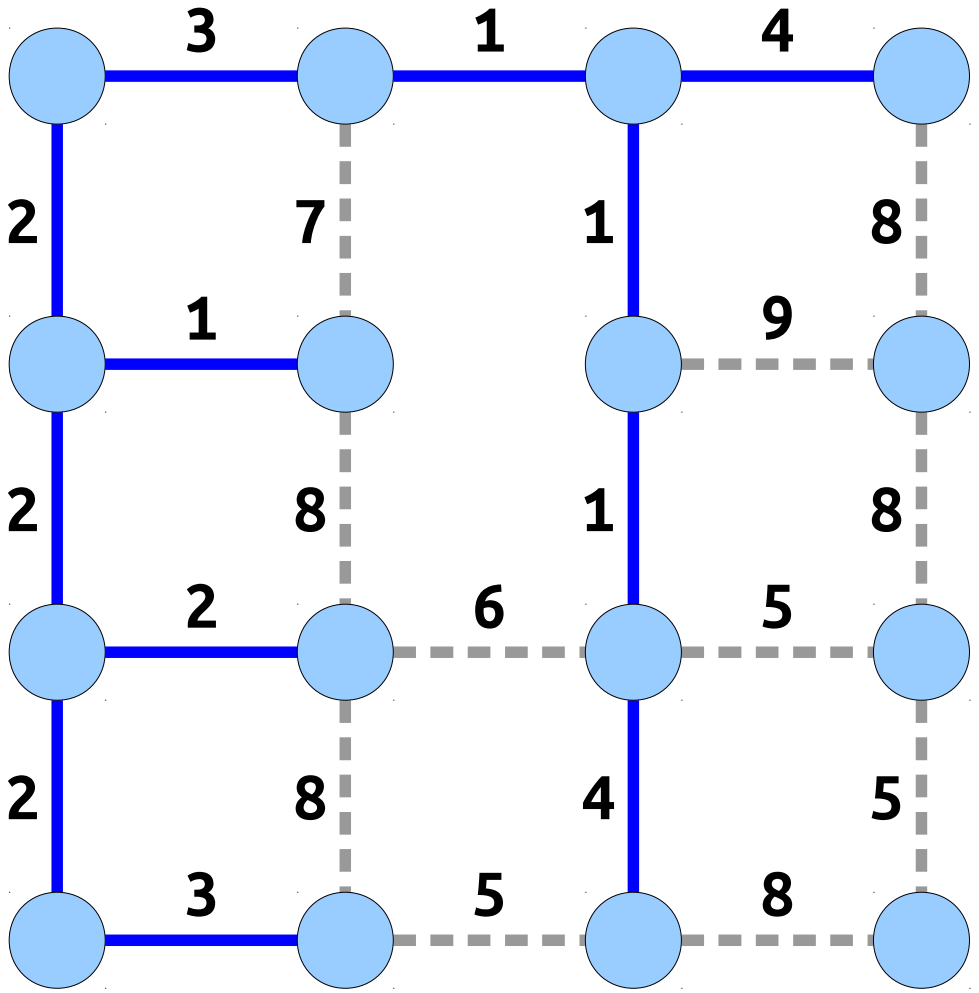
Mazes with Kruskal's Algorithm



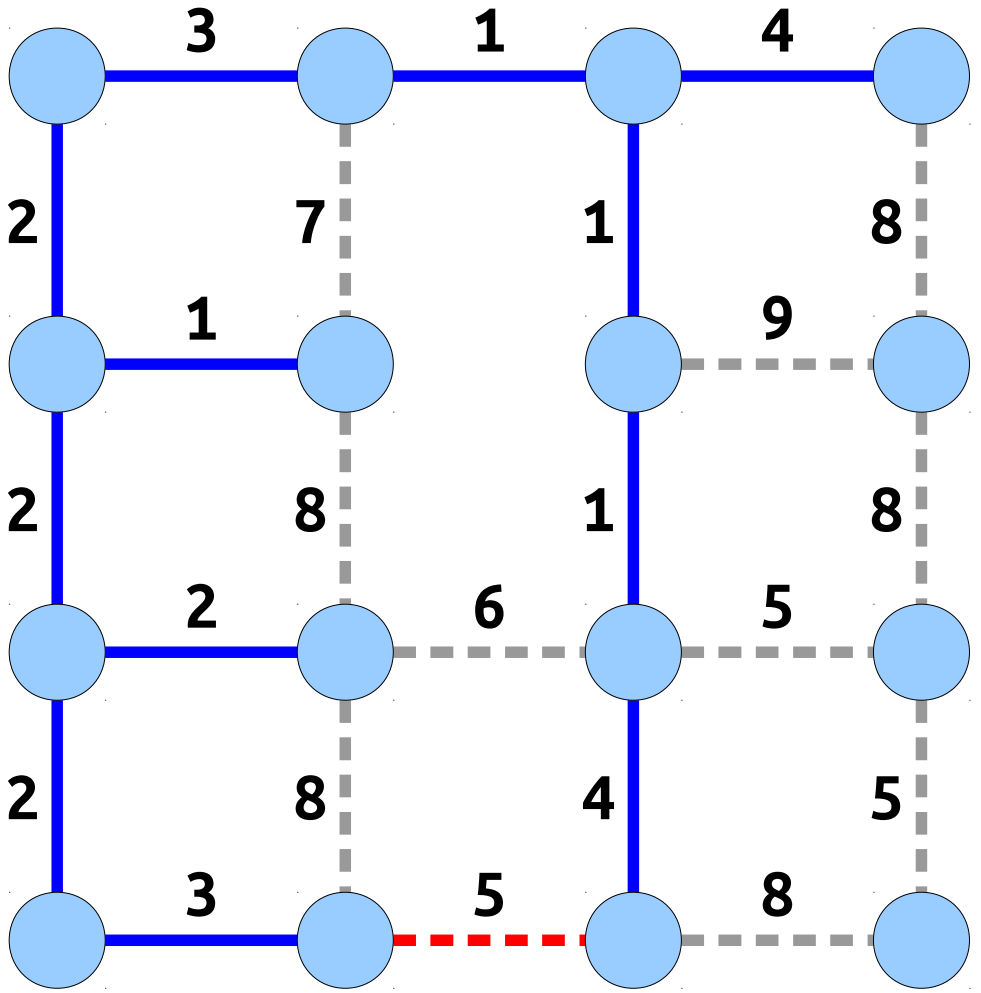
Mazes with Kruskal's Algorithm



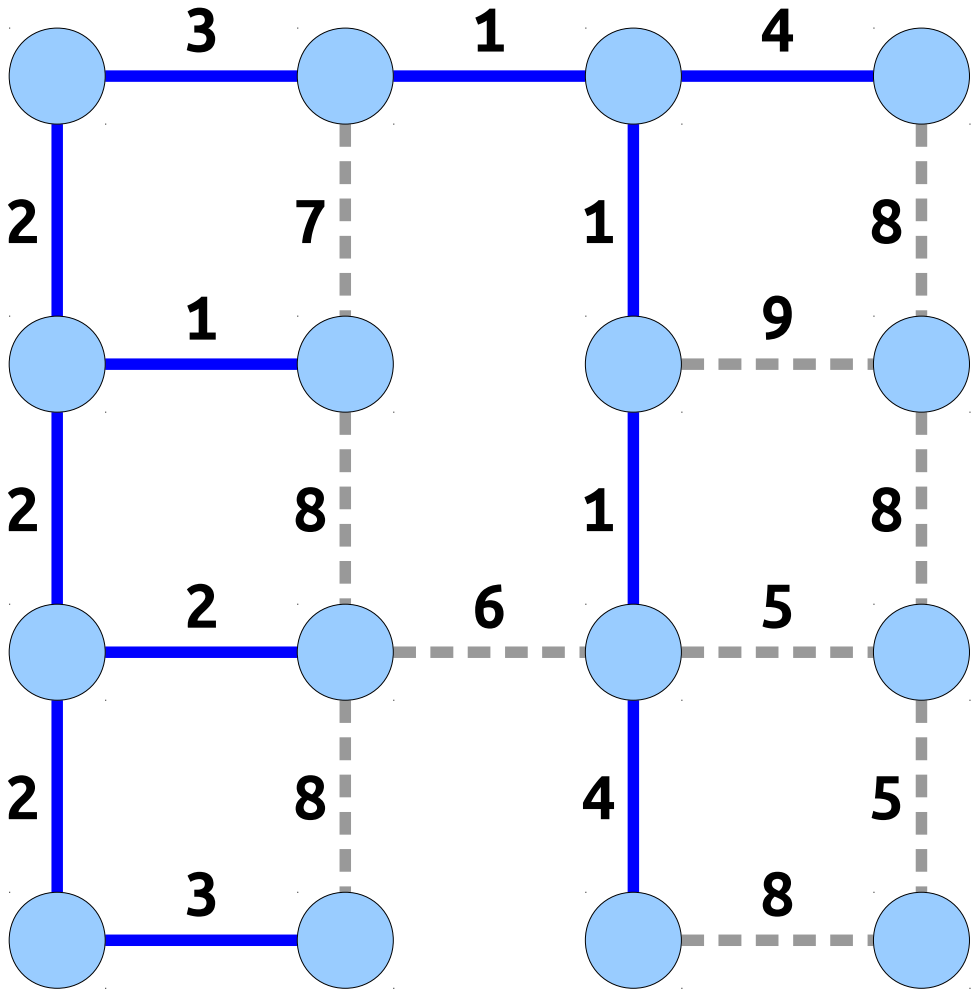
Mazes with Kruskal's Algorithm



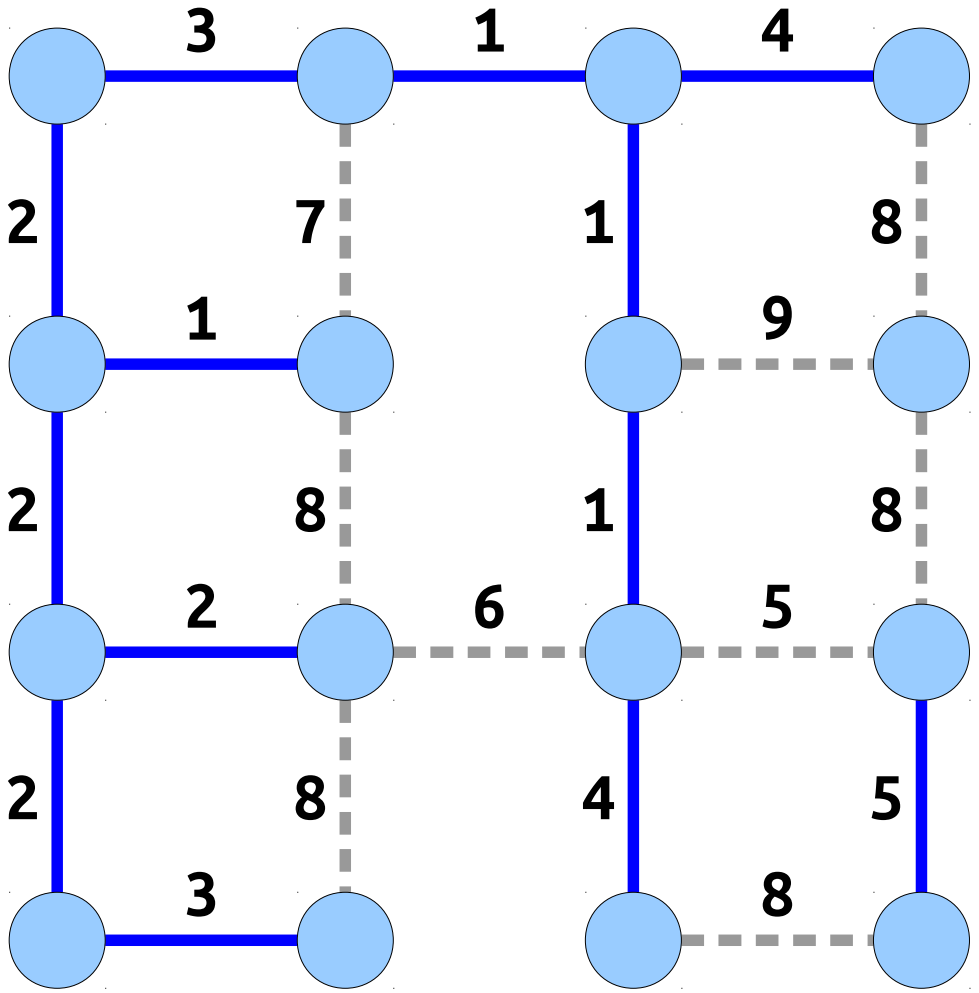
Mazes with Kruskal's Algorithm



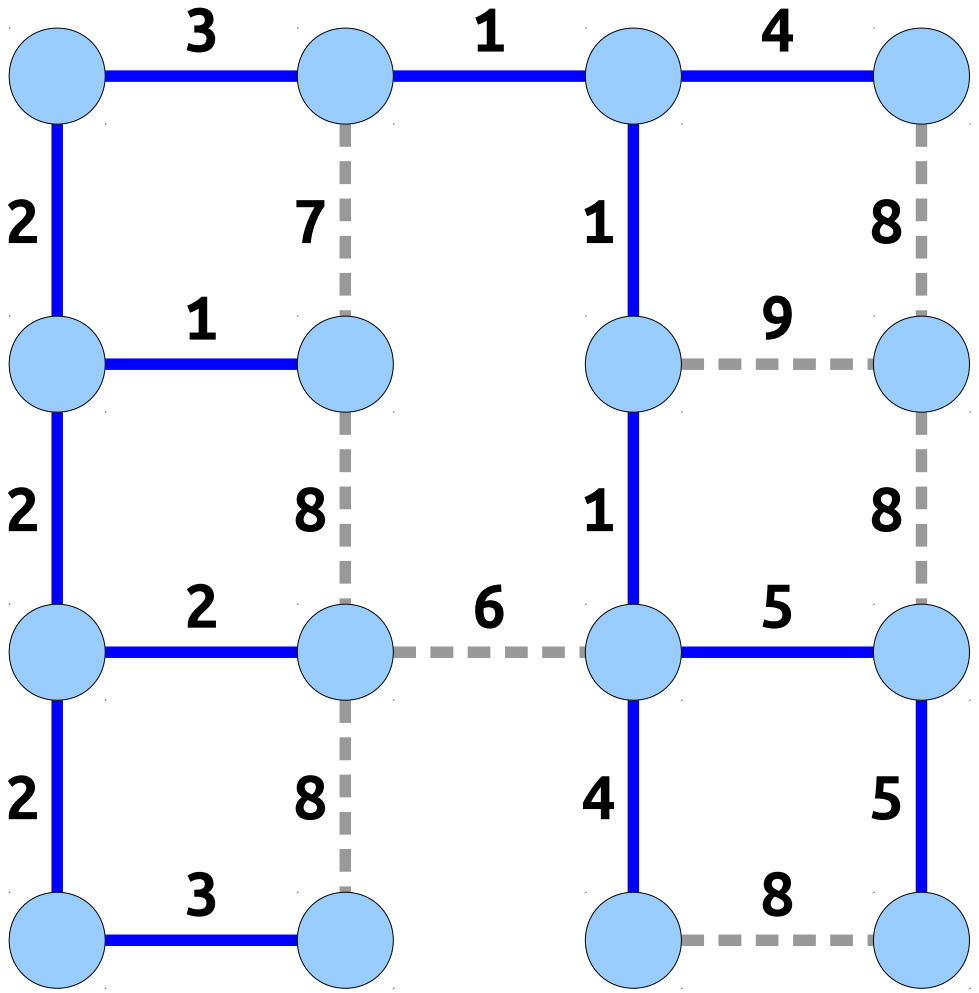
Mazes with Kruskal's Algorithm



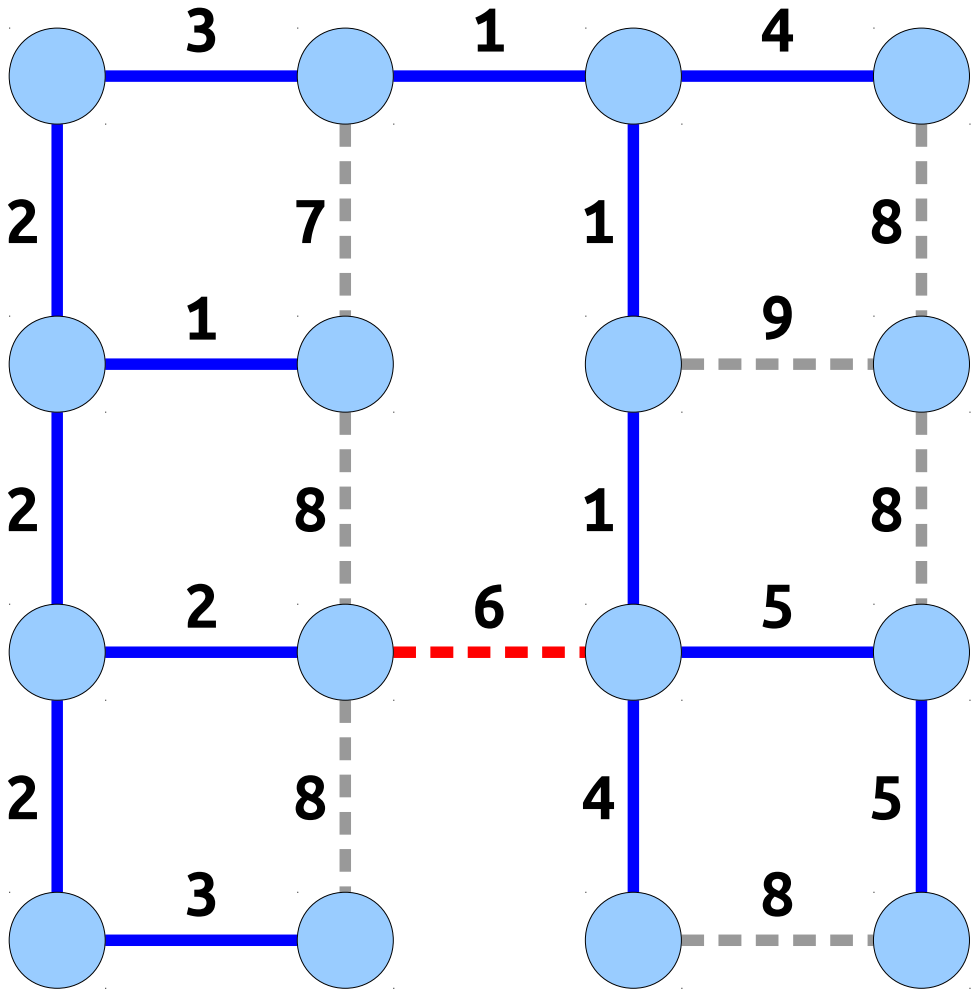
Mazes with Kruskal's Algorithm



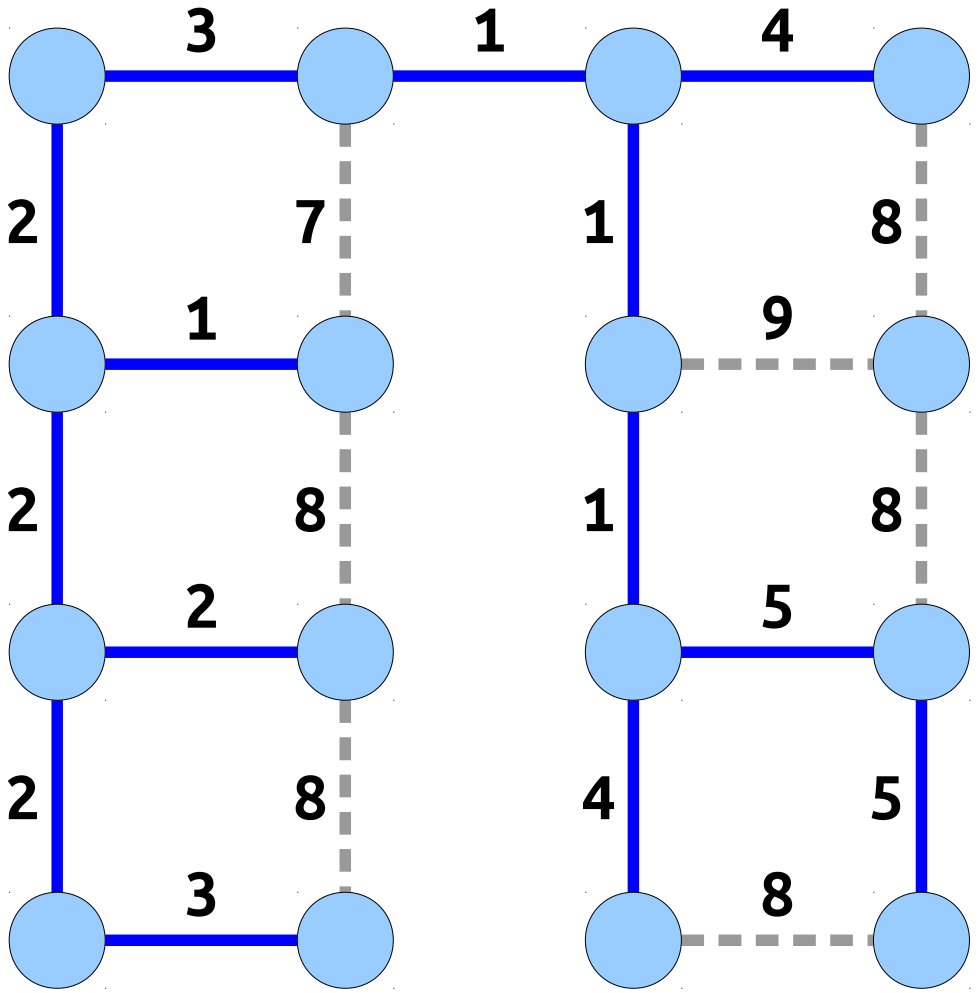
Mazes with Kruskal's Algorithm



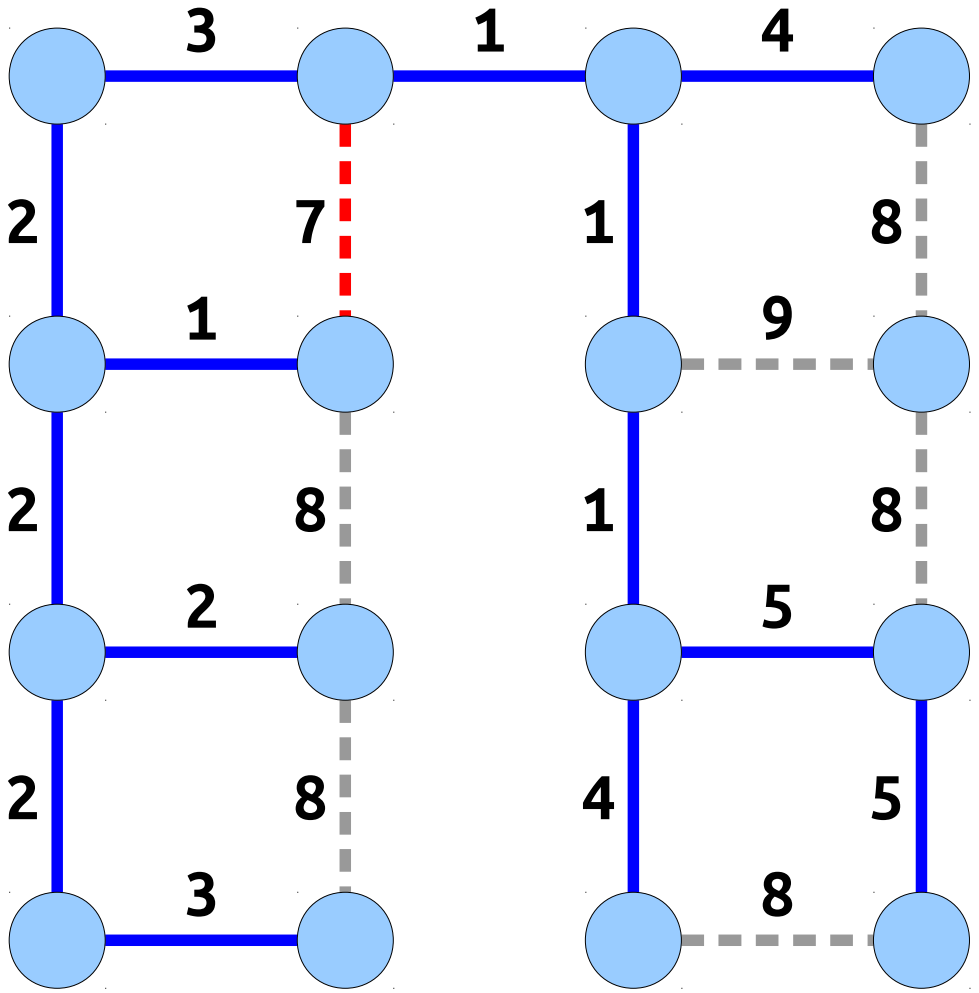
Mazes with Kruskal's Algorithm



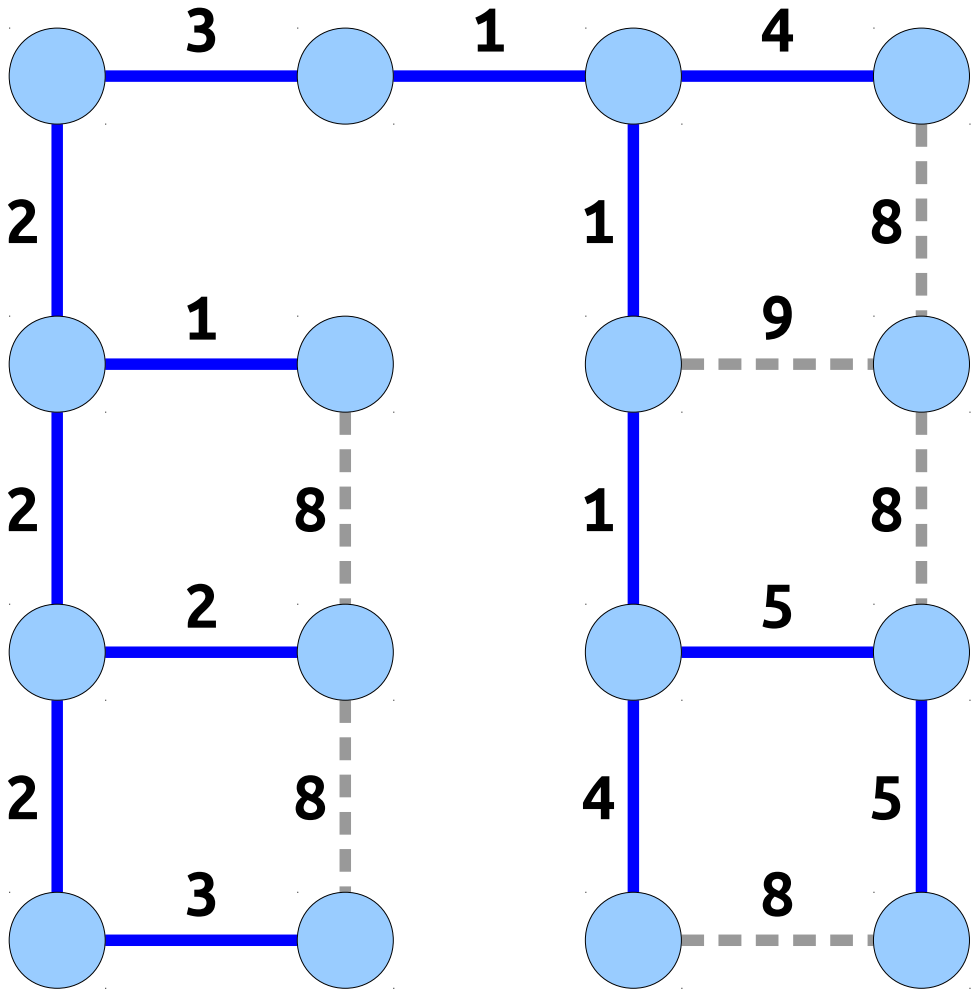
Mazes with Kruskal's Algorithm



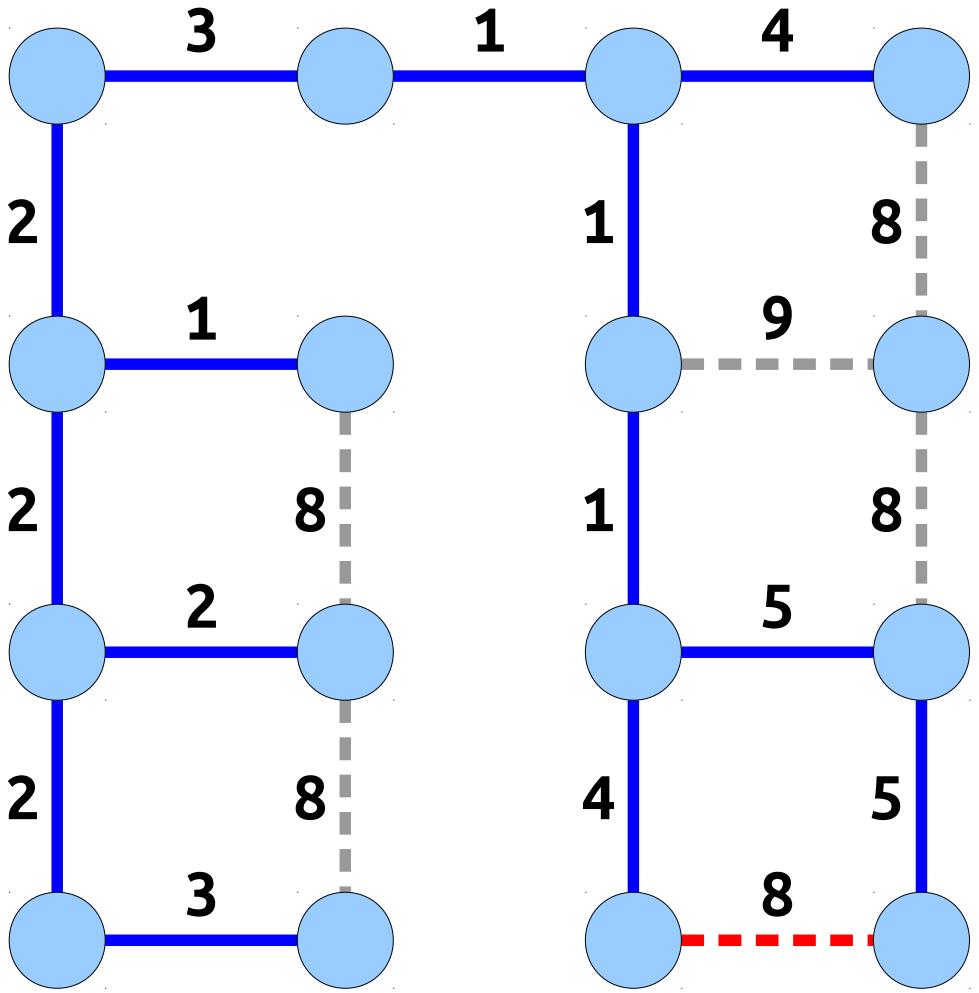
Mazes with Kruskal's Algorithm



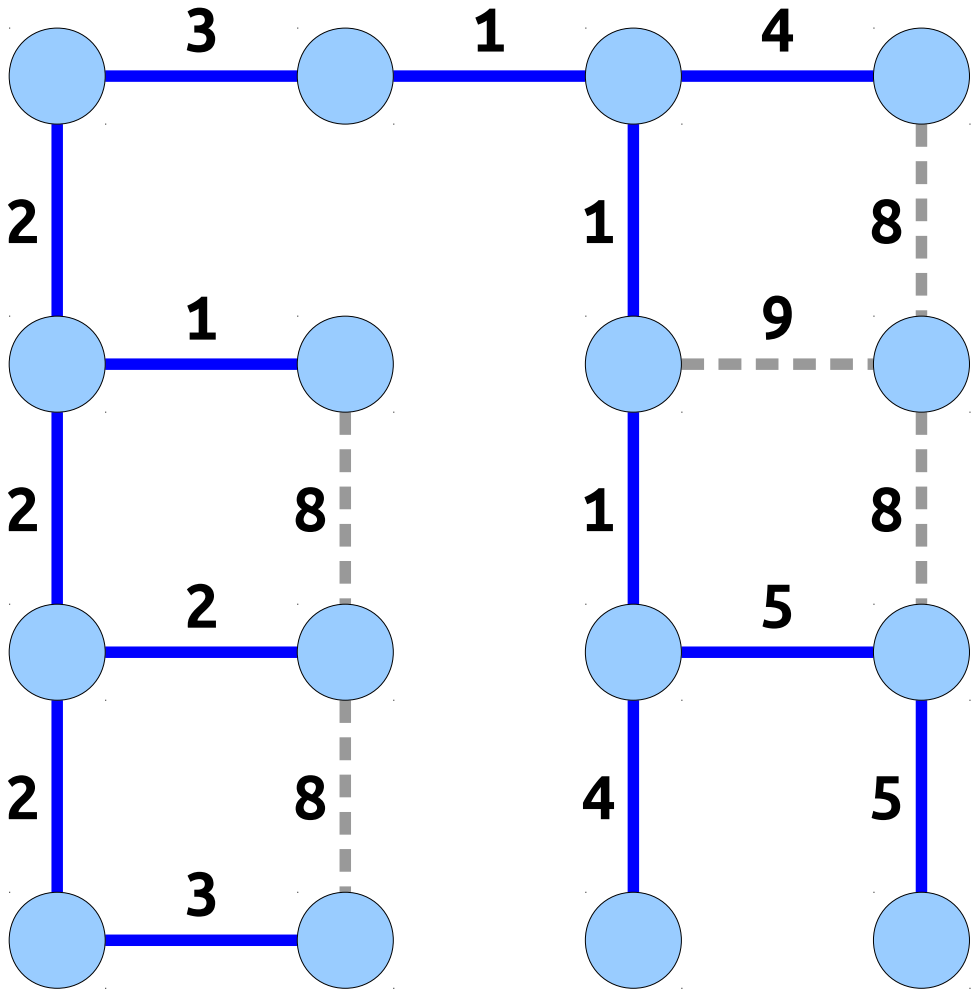
Mazes with Kruskal's Algorithm



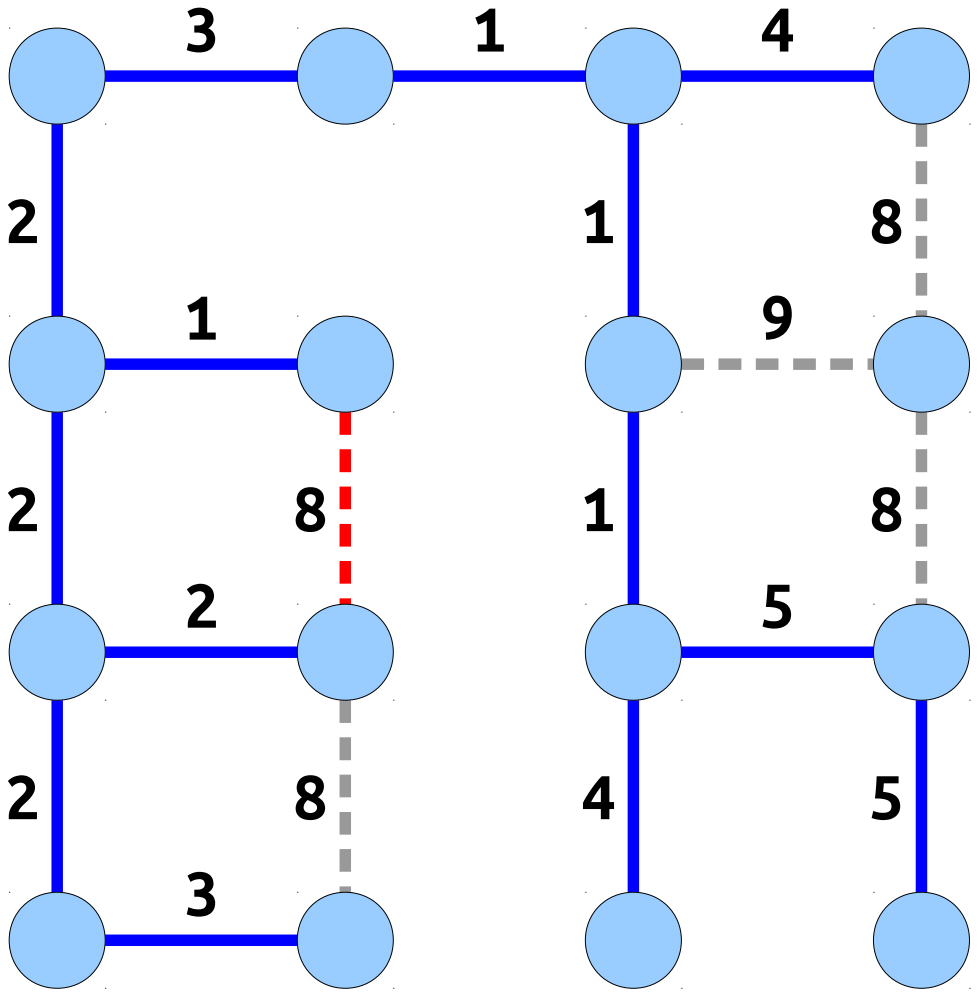
Mazes with Kruskal's Algorithm



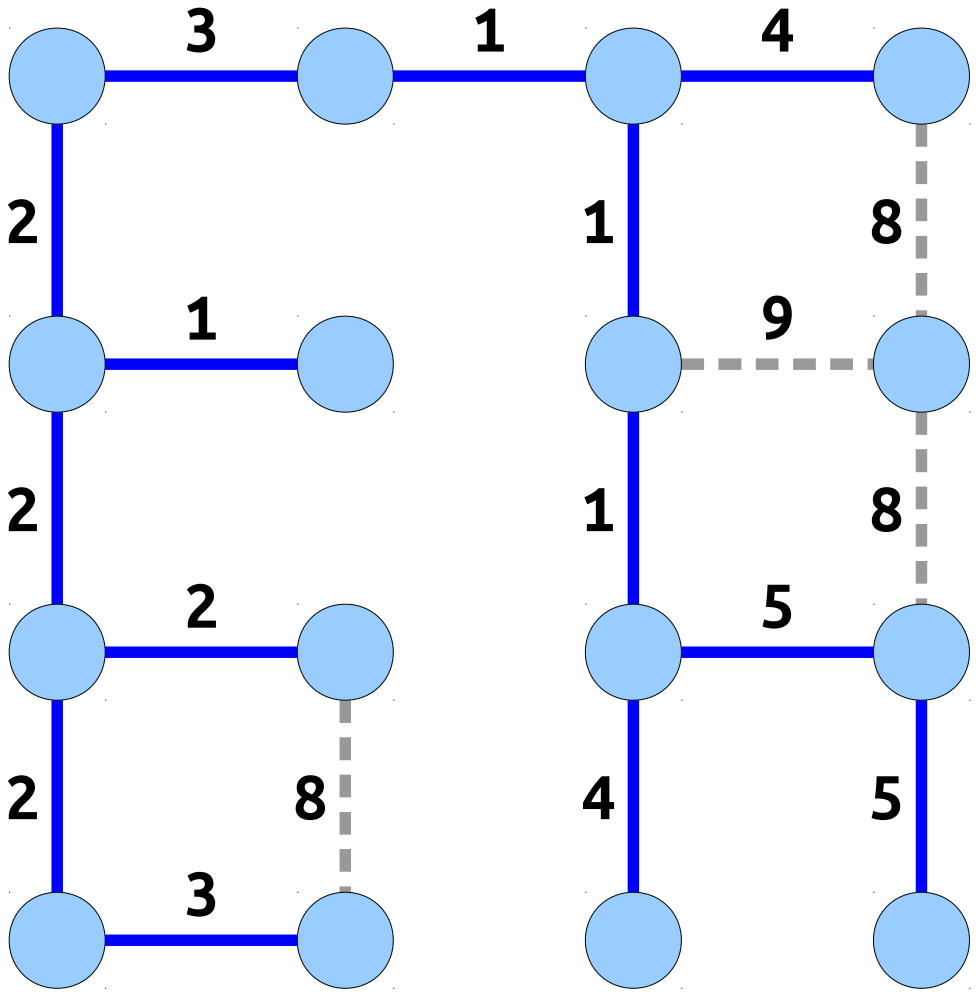
Mazes with Kruskal's Algorithm



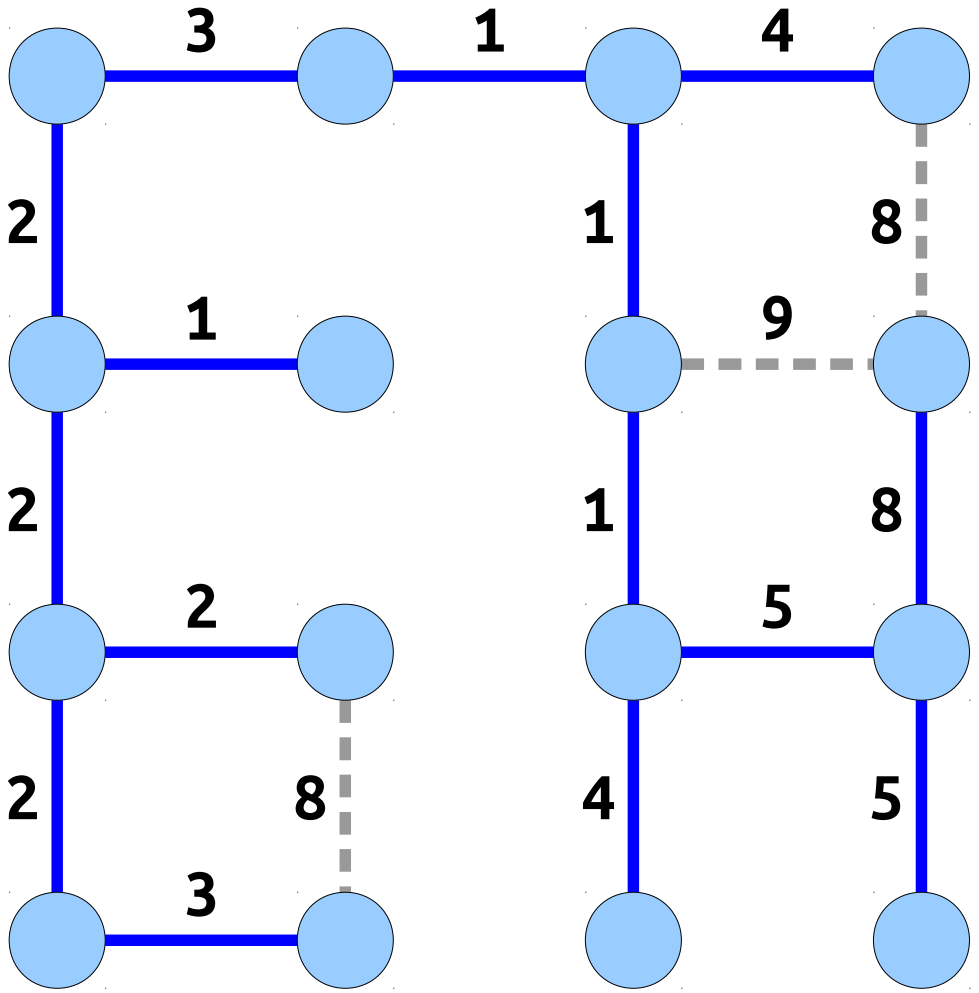
Mazes with Kruskal's Algorithm



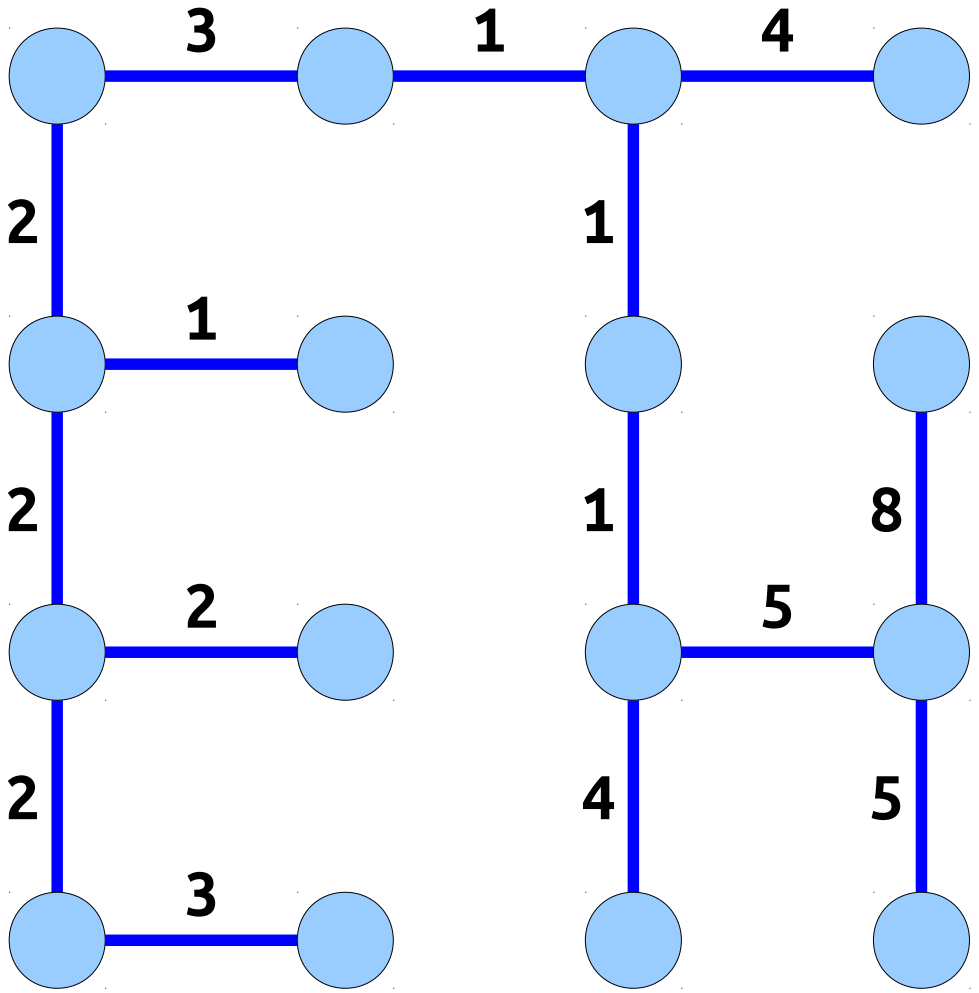
Mazes with Kruskal's Algorithm



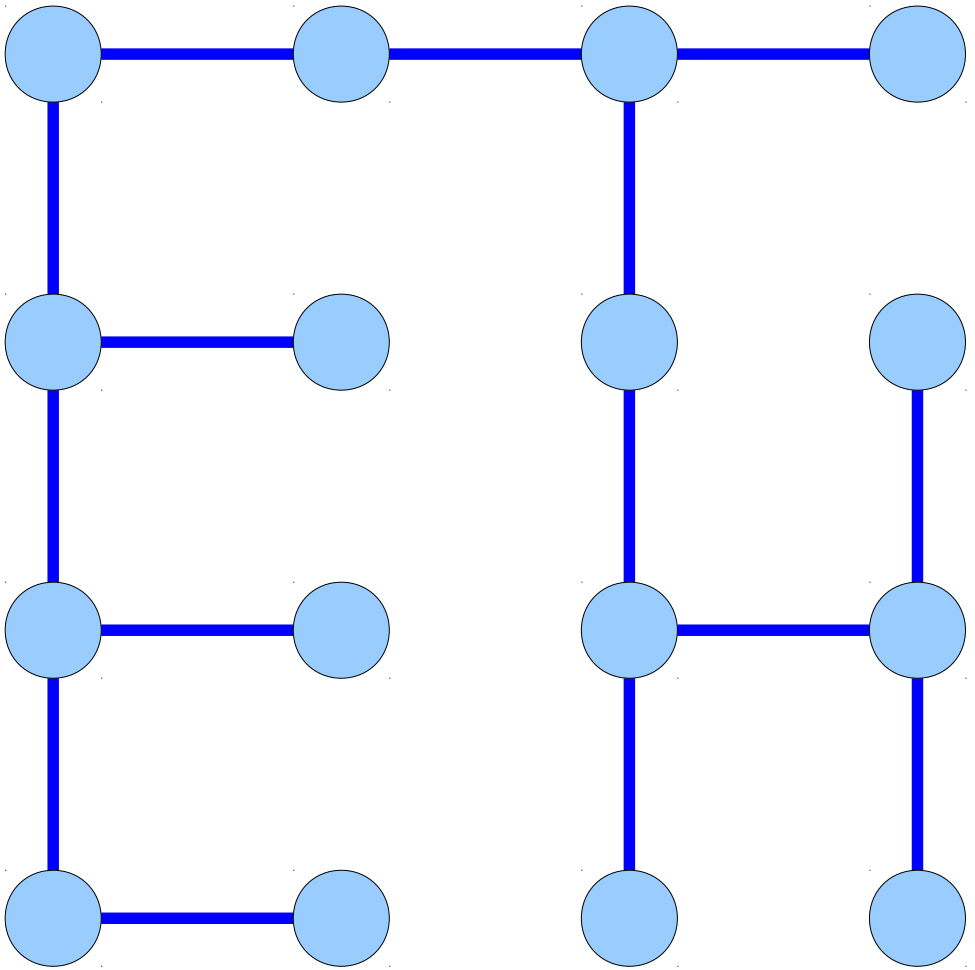
Mazes with Kruskal's Algorithm



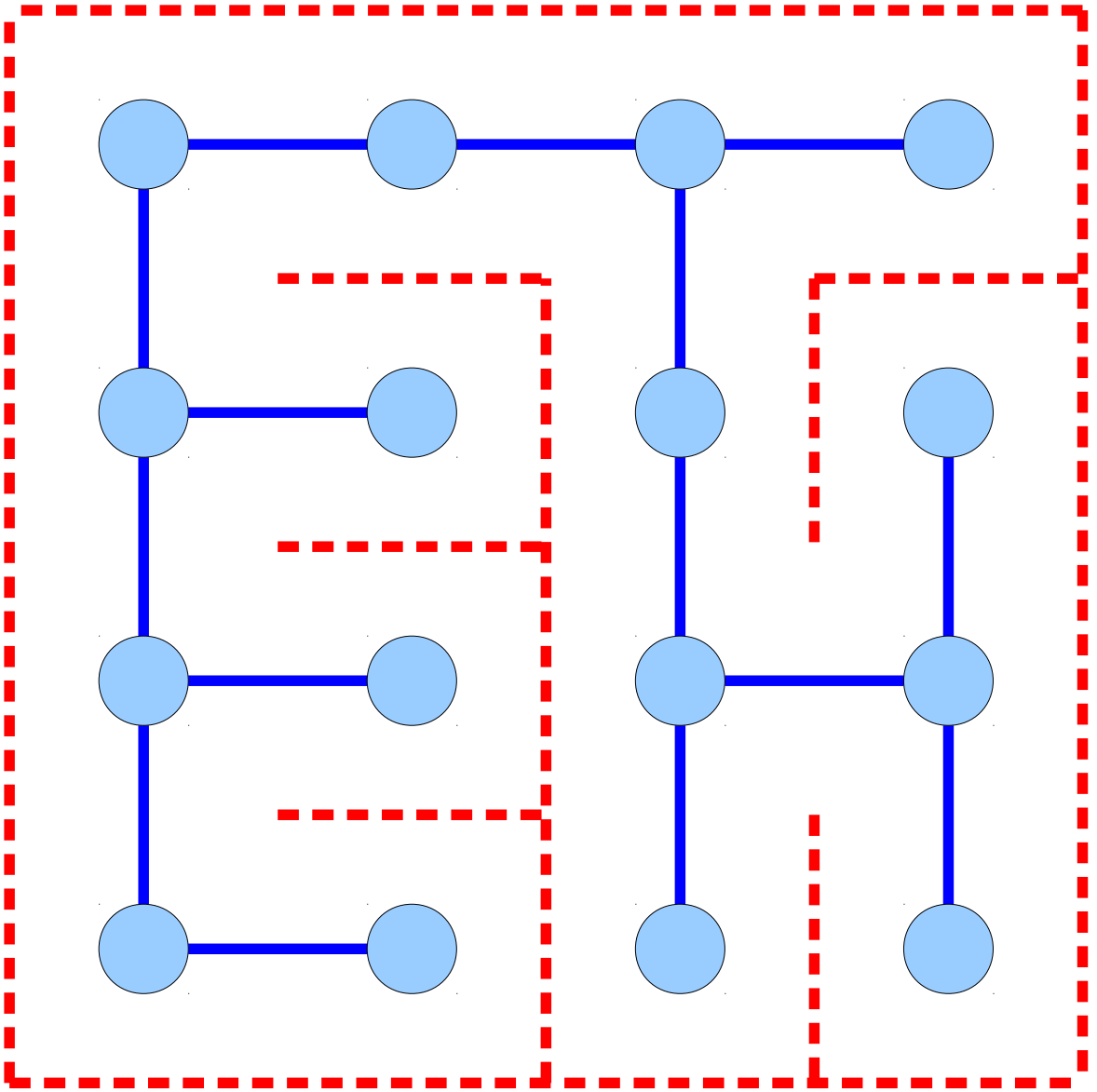
Mazes with Kruskal's Algorithm



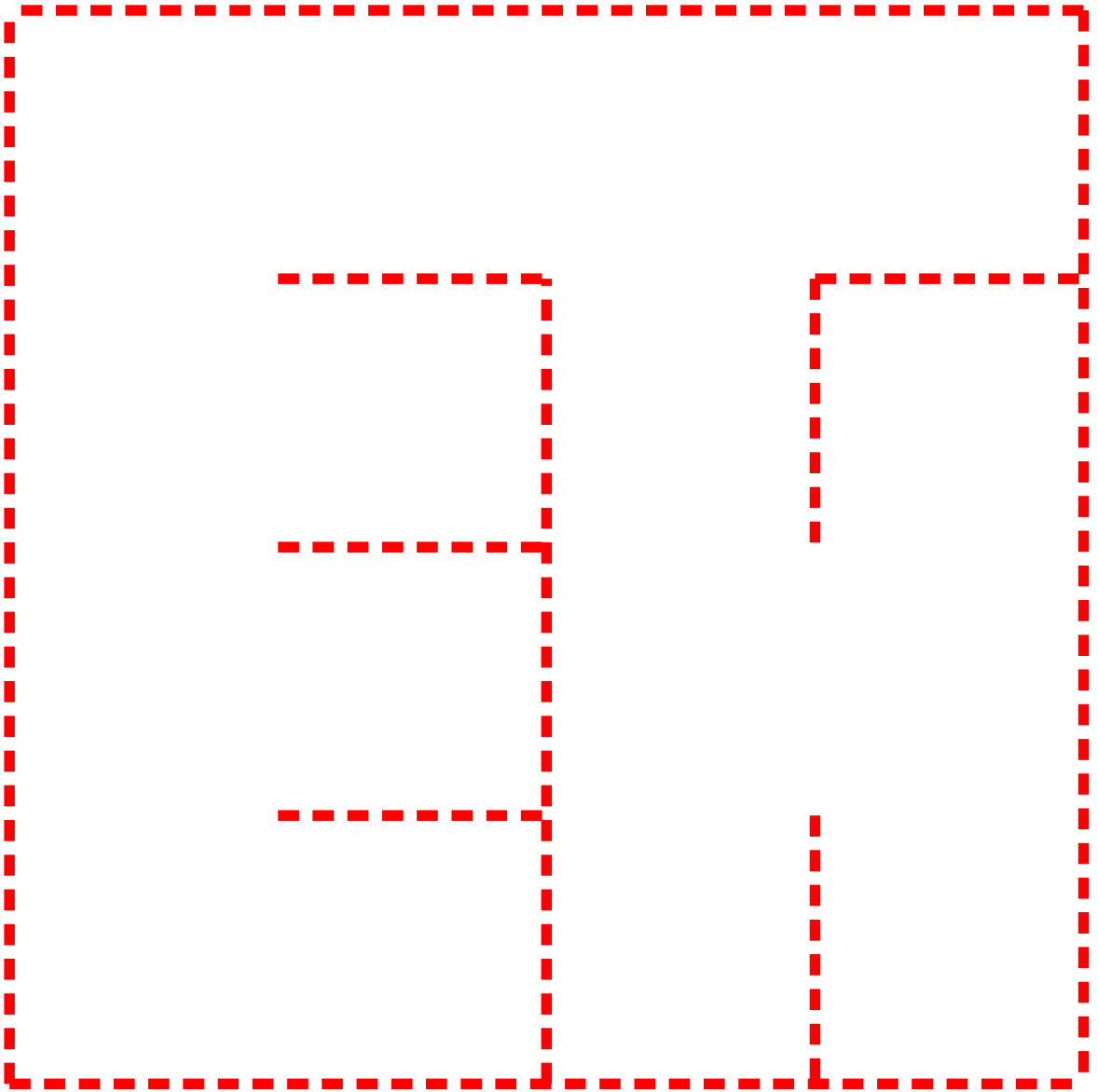
Mazes with Kruskal's Algorithm



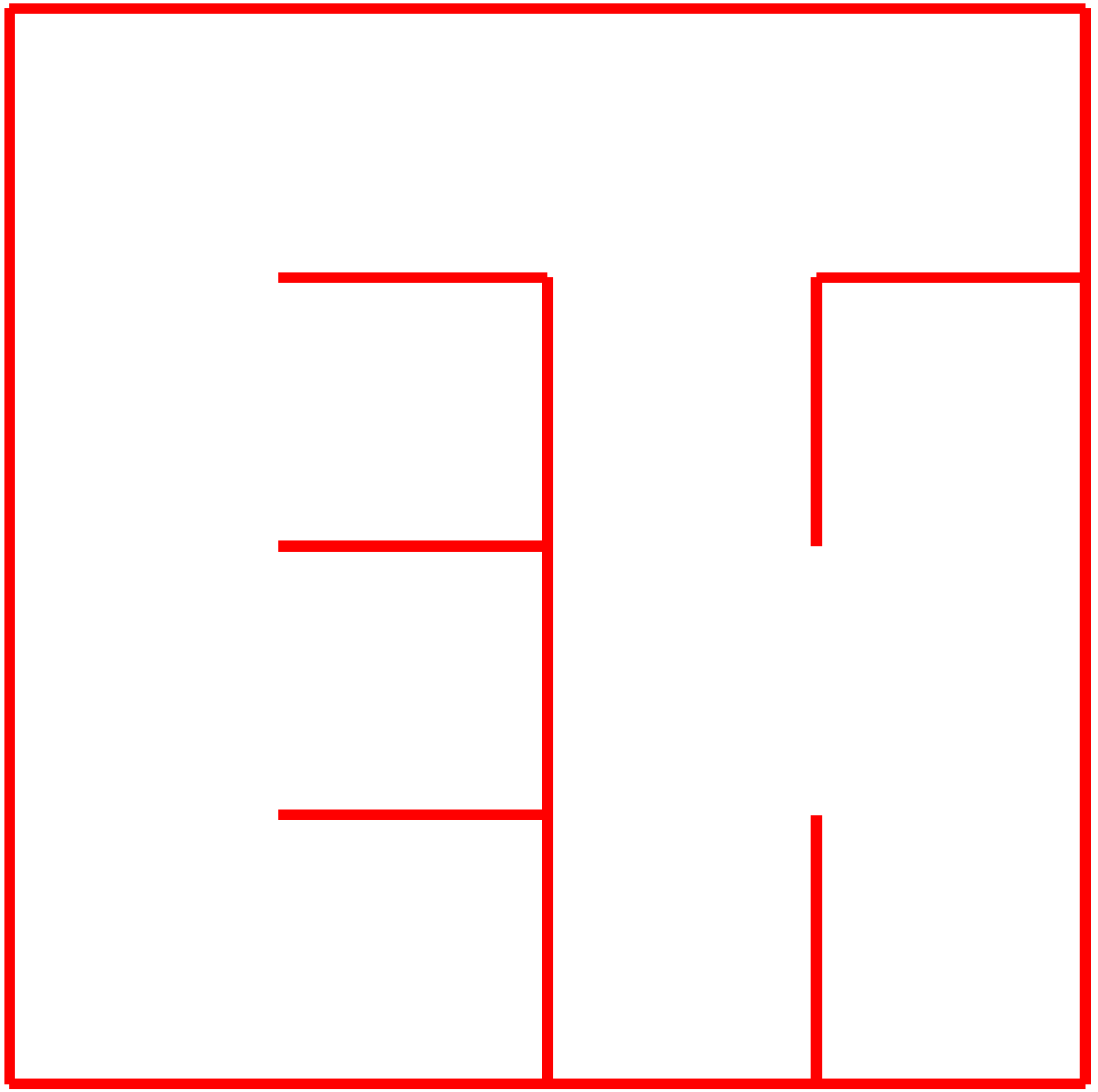
Mazes with Kruskal's Algorithm



Mazes with Kruskal's Algorithm



Mazes with Kruskal's Algorithm



Mazes with Kruskal's Algorithm

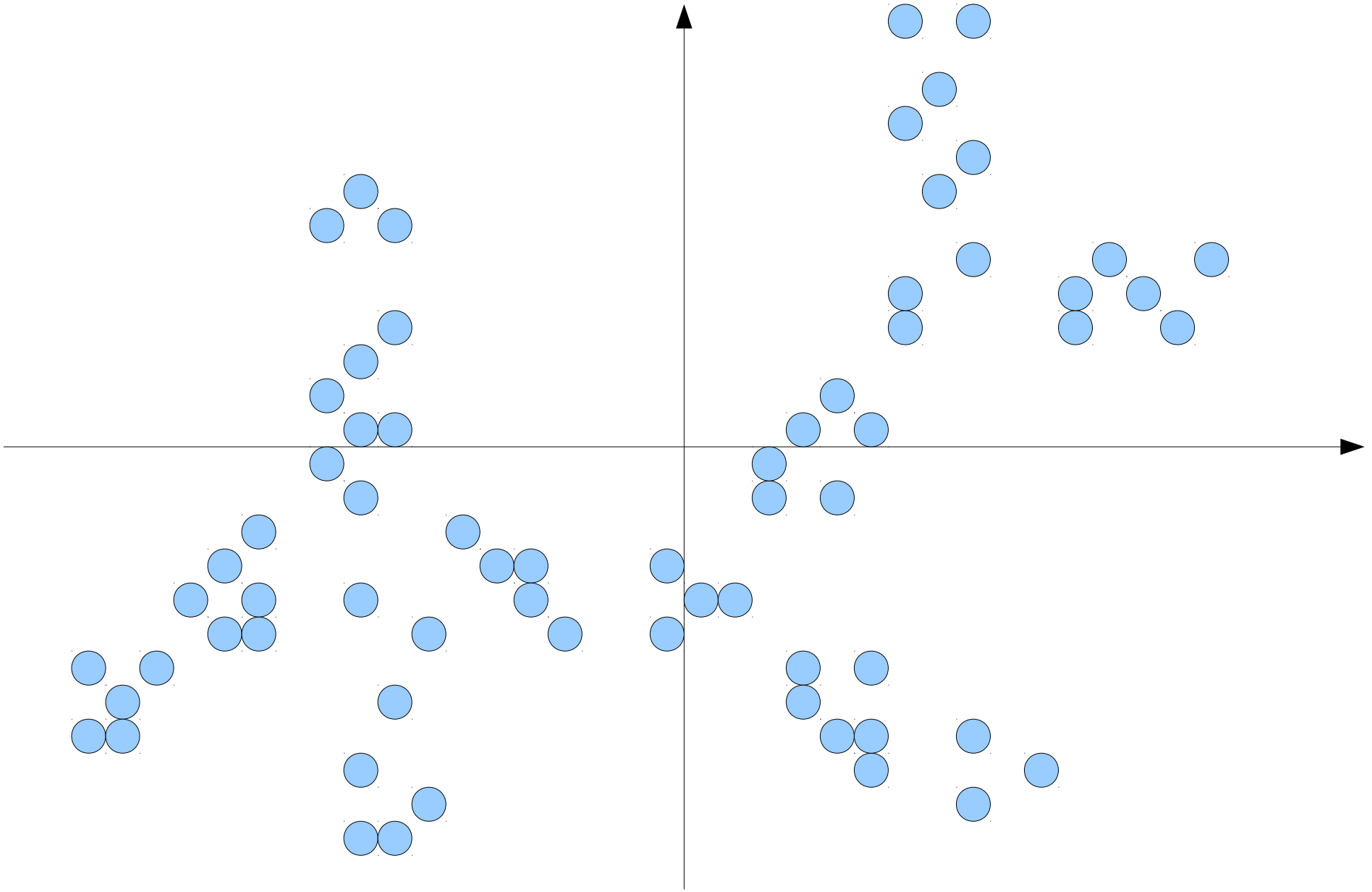
- The algorithm:
  - Create a grid graph.
  - Give each edge a random weight.
  - Compute an MST of that graph.
  - Put walls between any two cells that aren't adjacent in the MST.
- Compared with DFS-based mazes, tends to produce mazes with a high branching factor and short, twisty corridors.

Application: Stem Cells!

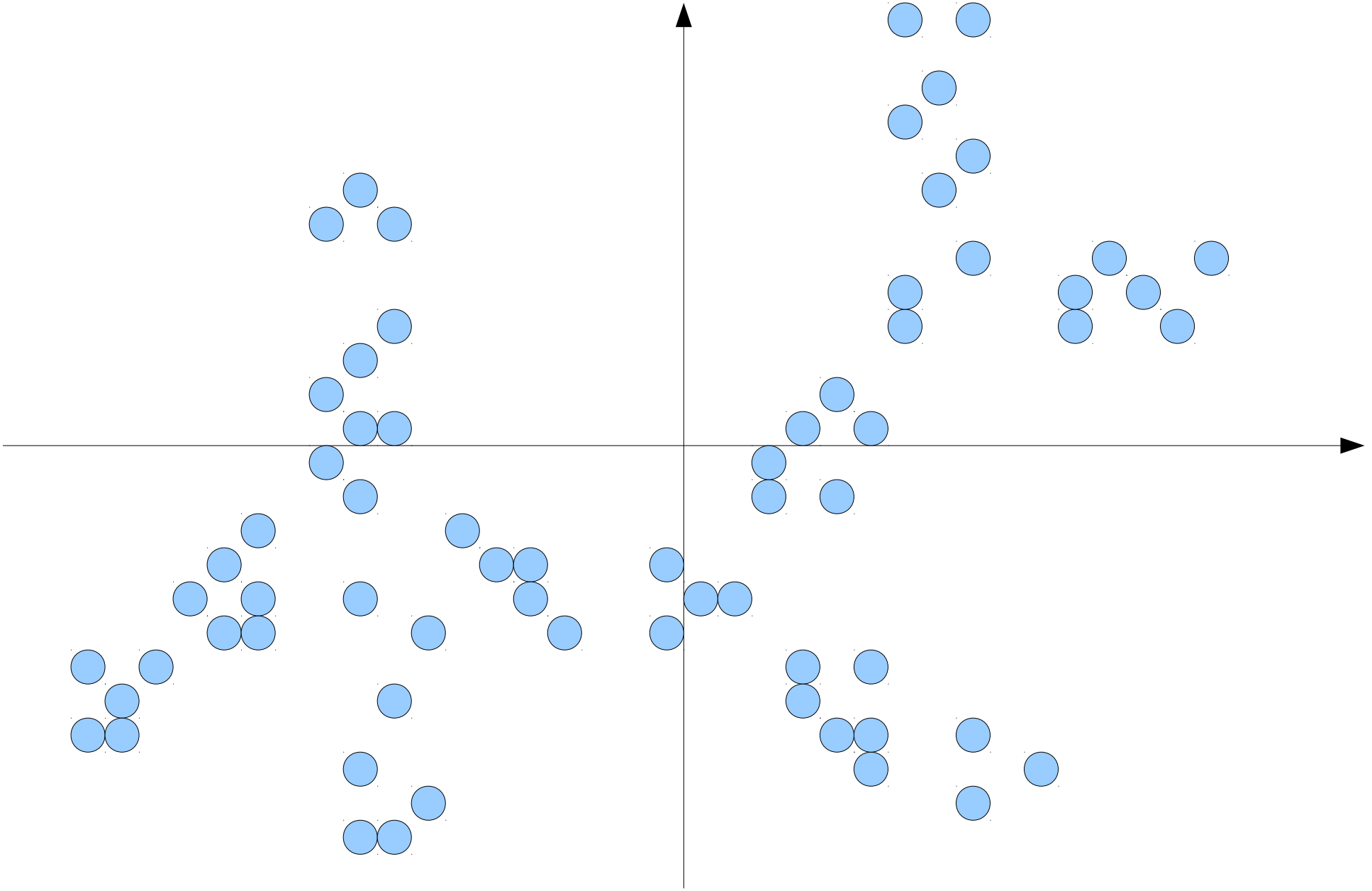
Question: How do you determine the patterns by which stem cells differentiate into specialized cells?

Step One: Grab a random collection of cells you know contains a bunch of stem cells.

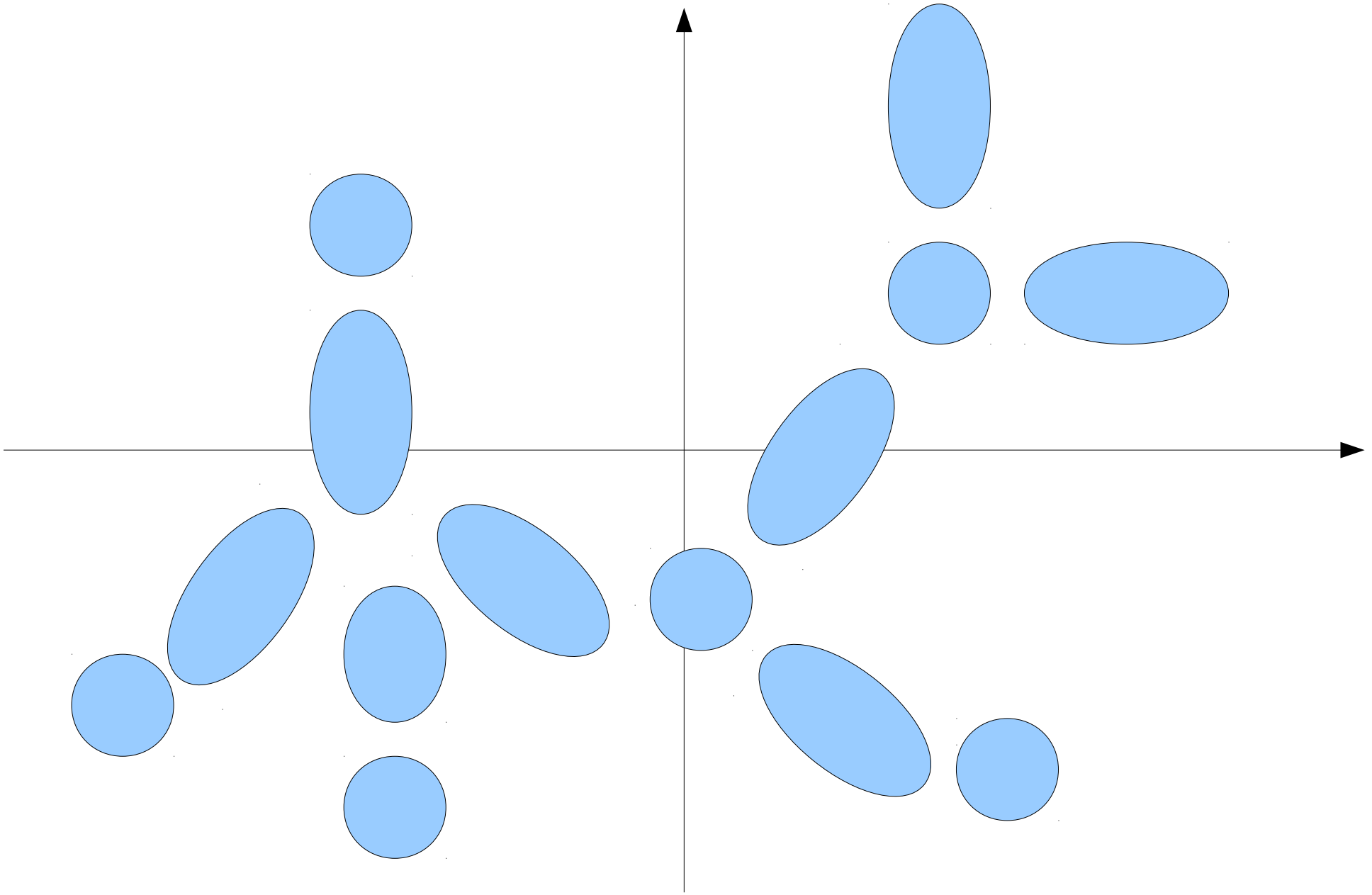
Step Two: Measure a bunch of different features from each cell and plot those features on a coordinate axis.



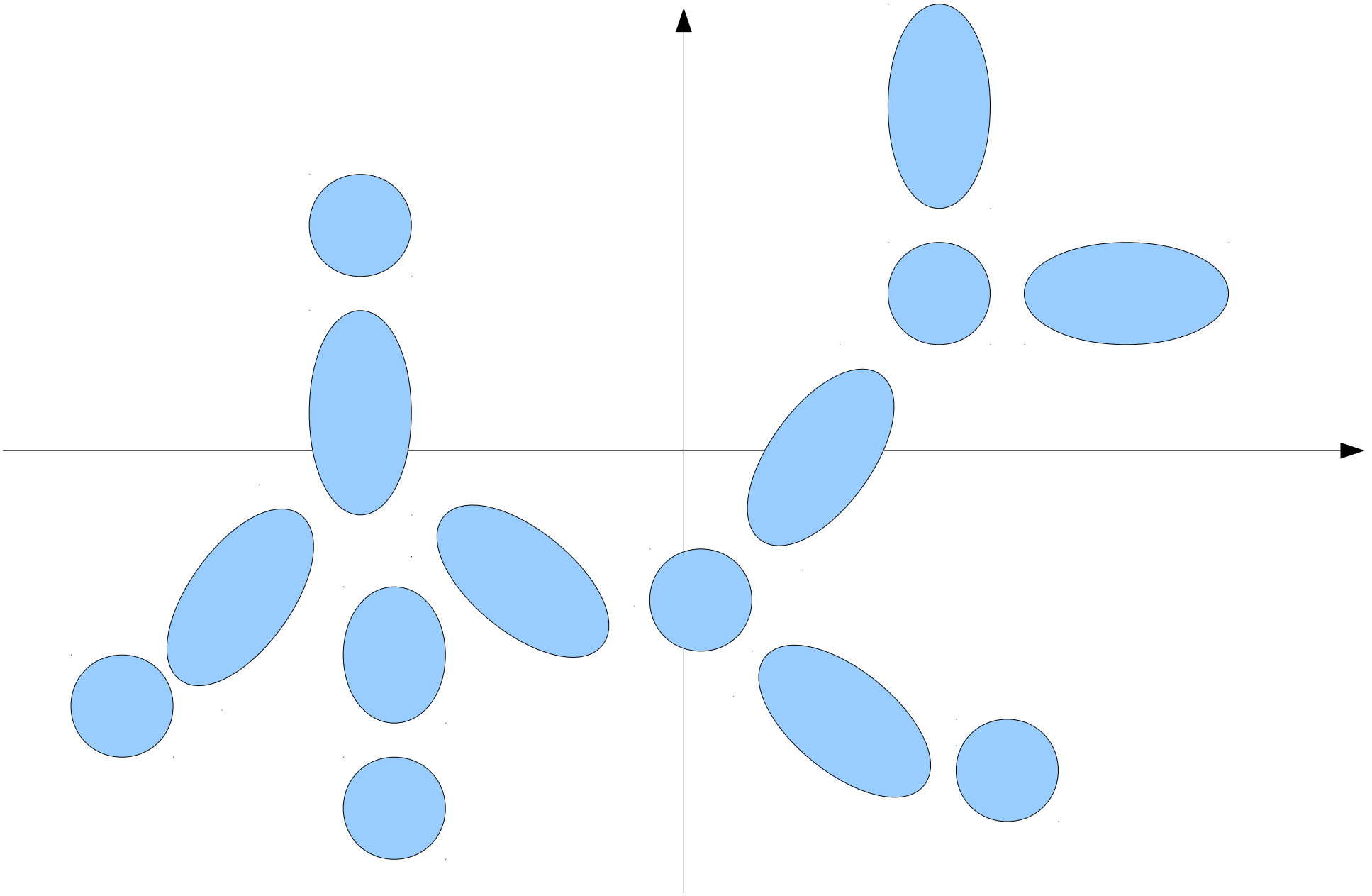
Step Two: Measure a bunch of different features from each cell and plot those features on a coordinate axis.



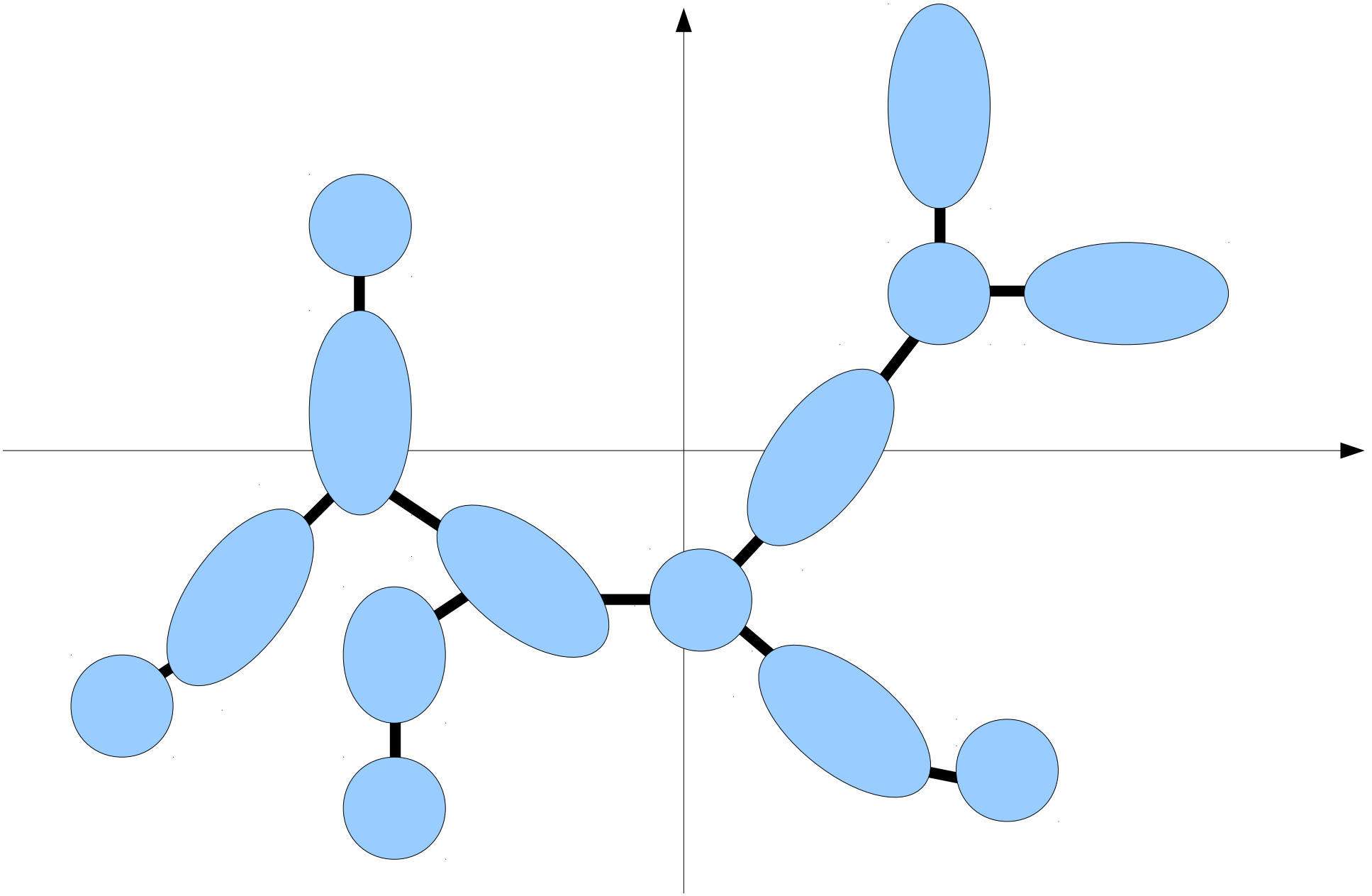
Step Three: Cluster those nodes into smaller groups, which likely represent cells of the same type.



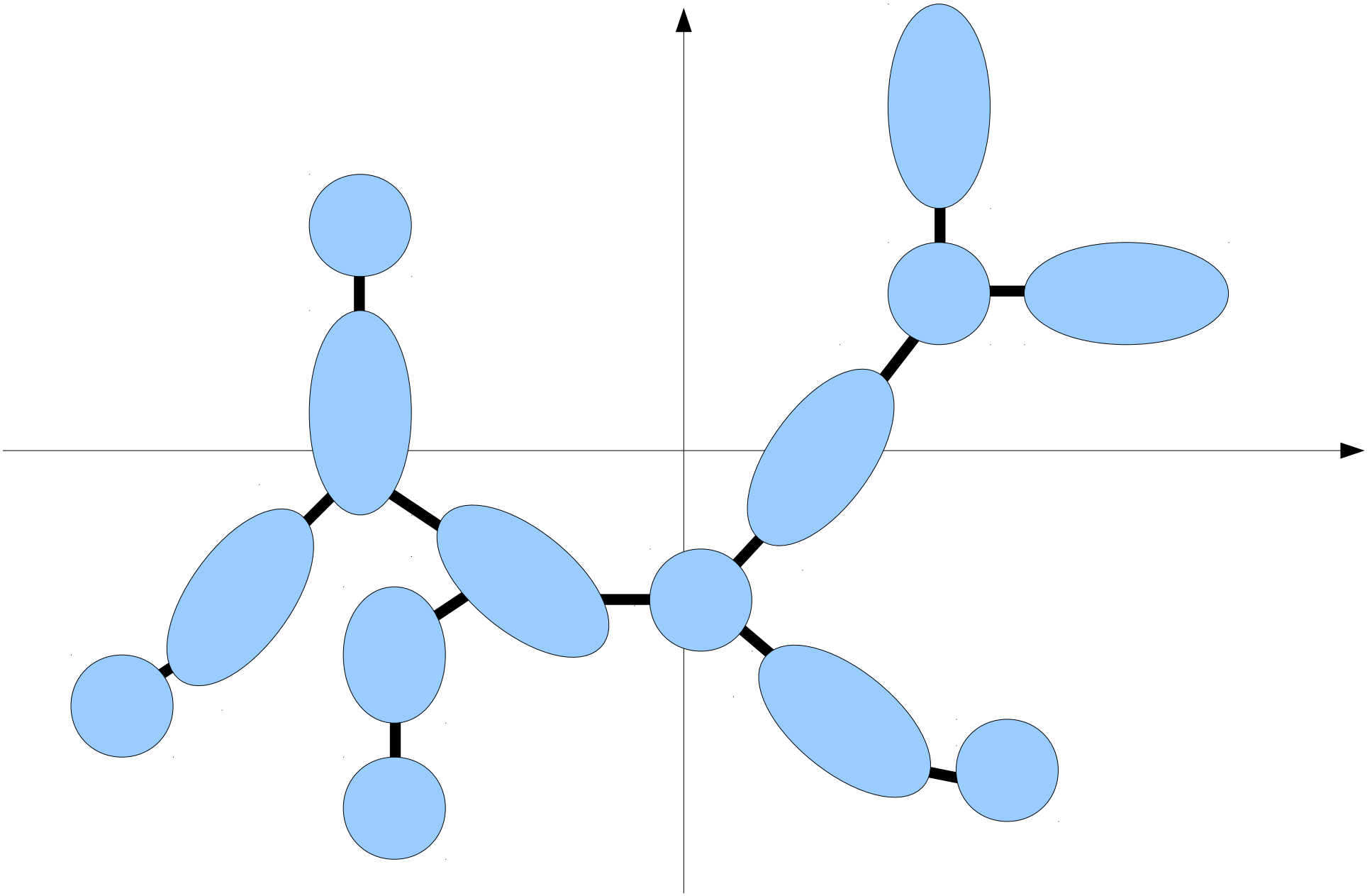
Step Three: Cluster those nodes into smaller groups, which likely represent cells of the same type.



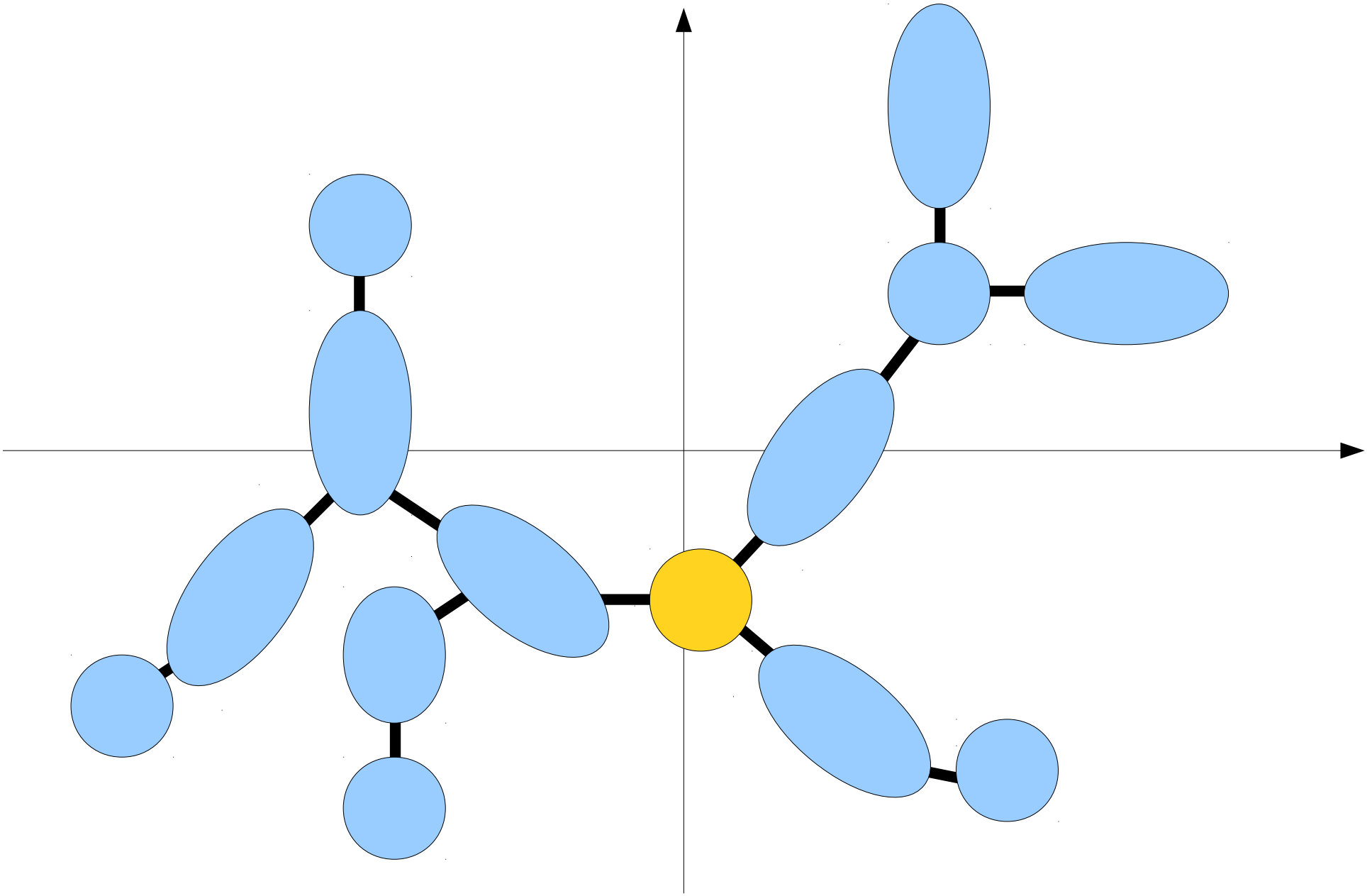
Step Four: Find an MST. Nodes are clusters and edges are distances. This is the cheapest tree connecting the clusters.



Step Four: Find an MST. Nodes are clusters and edges are distances. This is the cheapest tree connecting the clusters.



Step Five: Figure out which cluster represents the original stem cells. You now have the likely differentiation pattern!



Step Five: Figure out which cluster represents the original stem cells. You now have the likely differentiation pattern!

nature biotechnology

LETTERS

The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells

Cole Trapnell[1,2,6], Davide Cacchiarelli[1–3,6], Jonna Grimsby[2], Prapti Pokharel[2], Shuqiang Li[4], Michael Morse[1,2], Niall J Lennon[2], Kenneth J Livak[4], Tarjei S Mikkelsen[1–3] & John L Rinn[1,2,5]

Defining the transcriptional dynamics of a temporal process such as cell differentiation is challenging owing to the high variability in gene expression between individual cells. Time-series gene expression analyses of bulk cells have difficulty distinguishing early and late phases of a transcriptional cascade or identifying rare subpopulations of cells, and single-cell proteomic methods rely on a priori knowledge of key distinguishing markers[1]. Here we describe Monocle, an unsupervised algorithm that increases the temporal resolution of transcriptome dynamics using single-cell RNA-Seq data collected at multiple time points. Applied to the differentiation of primary human myoblasts, Monocle revealed switch-like changes in expression of key regulatory factors, sequential waves of gene regulation, and expression of regulators that were not known to act in differentiation. We validated some of these predicted regulators in a loss-of function screen. Monocle can in principle be used to recover single-cell gene expression kinetics from a wide array of cellular processes, including differentiation, proliferation and oncogenic transformation.

Such averaging artifacts can make factors that are correlated appear to be uncorrelated or even make positively correlated factors appear negatively correlated. As a population of cells captured at the same time may include many distinct intermediate differentiation states, considering only its average properties would mask trends occurring across individual cells. Solving this problem by experimental synchronization of cells or by stringent isolation of precursors at distinct stages is challenging and can sharply alter differentiation kinetics.

Computational analysis of gene expression data could help define biological progression between cellular states and reveal regulatory modules of genes that co-vary in expression across individual cells[9]. Previous analyses have used approaches from computational geometry[10,11] to order bulk cell populations from time-series microarray experiments by progress through a biological process independently of when the samples were collected. The recently developed SPD algorithm can resolve progression along multiple lineages arising from a progenitor cell type using supervised machine learning[12]. However, because these algorithms operate on bulk expression measurements, they are sensitive to mixture effects arising from Simpson's paradox and other averaging artifacts. Single-cell assays such as flow or

First the algorithm represents the expression profile of each cell as a point in a high-dimensional Euclidean space, with one dimension for each gene. Second, it reduces the dimensionality of this space using independent component analysis[17]. Dimensionality reduction transforms the cell data from a high-dimensional space into a low-dimensional one that preserves essential relationships between cell populations but is much easier to visualize and interpret[18]. Third, Monocle constructs a minimum spanning tree (MST) on the cells, a previously developed approach now commonly used in other single-cell settings, such as flow or mass cytometry[1,13]. Fourth, the algorithm finds the longest path through the MST, corresponding to the longest sequence of transcriptionally similar cells. Finally, Monocle uses this sequence to produce a 'trajectory' of an individual cell's progress through differentiation.

Building a repertoire of abstractions and algorithms helps you model and solve larger and larger classes of problems.

Interested in learning more?

Take CS161!