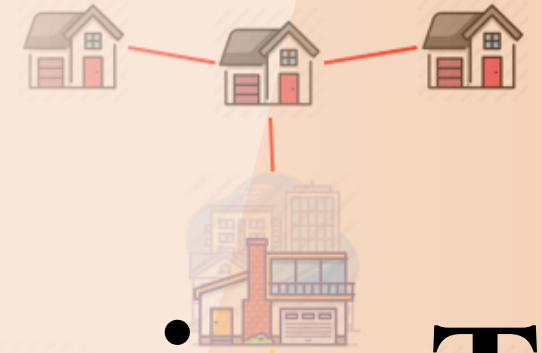


Naïve telecommunication routing, generates new connection line for each customer, which leads to huge cost.

Minimum Spanning Tree Routing generates connection line in less cost with high efficiency.

Minimum Spanning Tree

Jeff Chak Fu WONG

Department of Mathematics, Chinese University of Hong Kong, Shatin, Hong Kong

EDB HSMMC Math Modelling Advanced Training Workshop

Kruskal's Algorithm

Kruskal's algorithm is a greedy algorithm that finds a *minimum* spanning tree (MST) for a connected, undirected graph. It works by adding edges to the MST in increasing order of their weights, provided that adding a given edge does not form a cycle. The algorithm employs a disjoint-set data structure to efficiently detect cycles.

Connected graph: A graph is connected when there is a path between every pair of vertices. In a connected graph, there is no unreachable node.

Tree graph: A tree is an undirected graph in which every pair of distinct vertices is connected by exactly one path, or equivalently, a connected acyclic undirected graph.

Kruskal's Algorithm

The algorithm begins by sorting all edges of the graph in **non-decreasing order** of their weights.

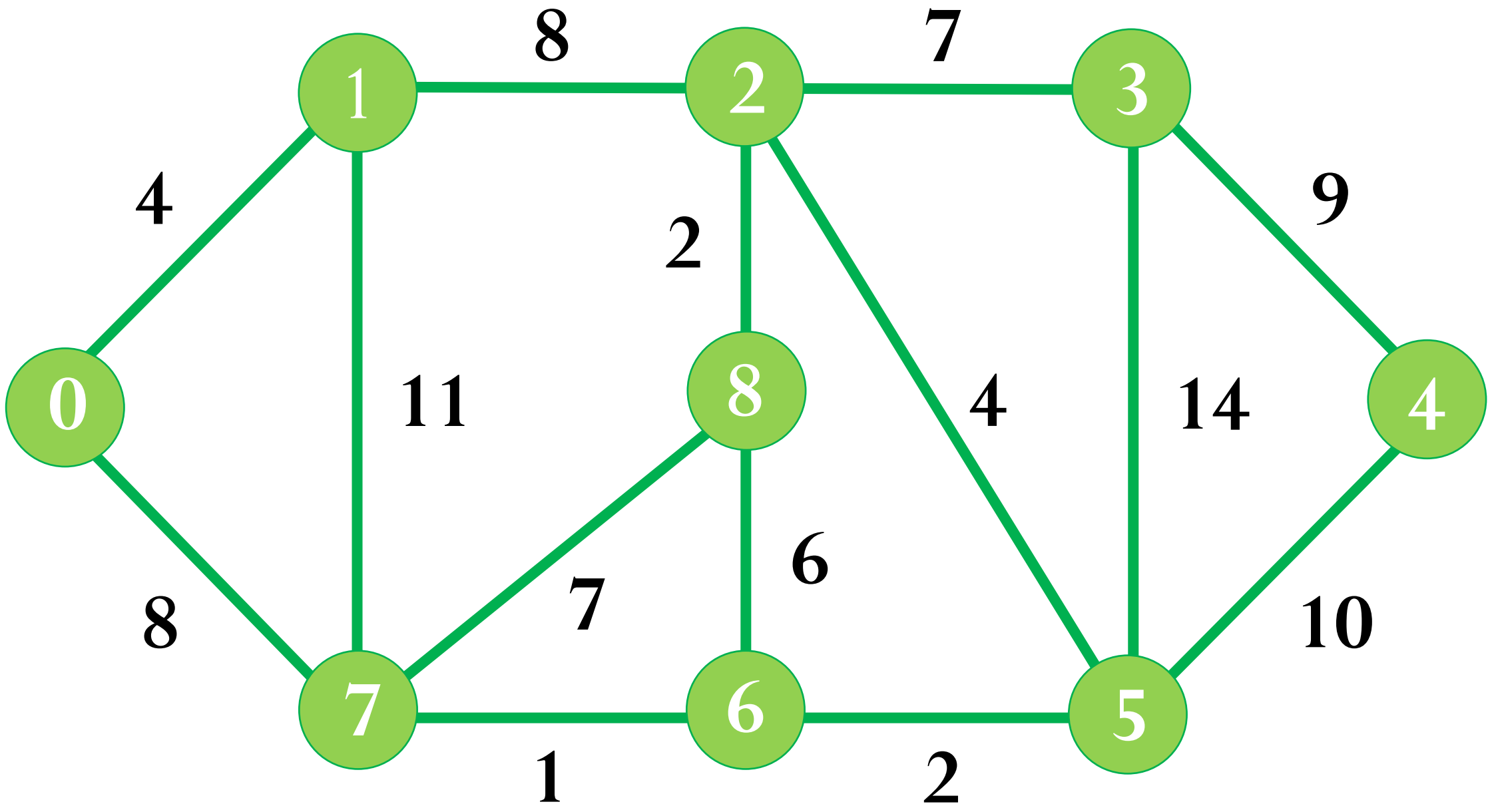
It then initializes a disjoint-set data structure with a separate set for each vertex in the graph.

Next, it iterates through the sorted list of edges.

For each edge, if the sets containing its two endpoints are distinct (meaning adding the edge does not form a cycle), the edge is added to the MST, and the corresponding sets are merged in the disjoint-set structure.

Finally, the list of edges belonging to the MST is returned.

Kruskal's Algorithm

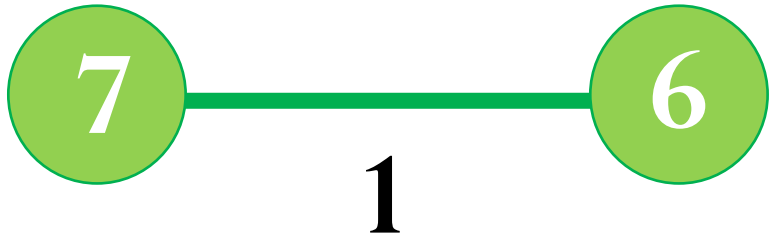


The graph contains 9 vertices and 14 edges.

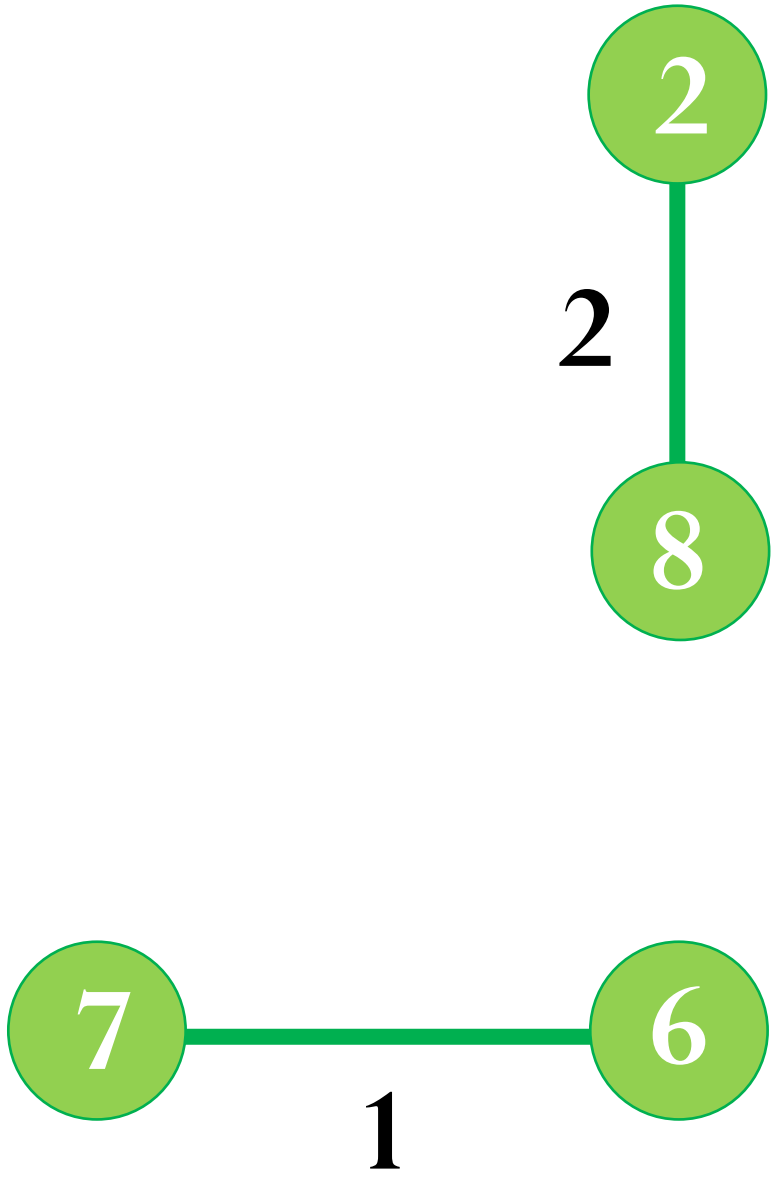
So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

Weight	1	2	2	4	4	6	7	7	8	8	9	10	11	14
Source	7	8	6	0	2	8	2	7	0	1	3	5	1	3
Destination	6	2	5	1	5	6	3	8	7	2	4	4	7	5

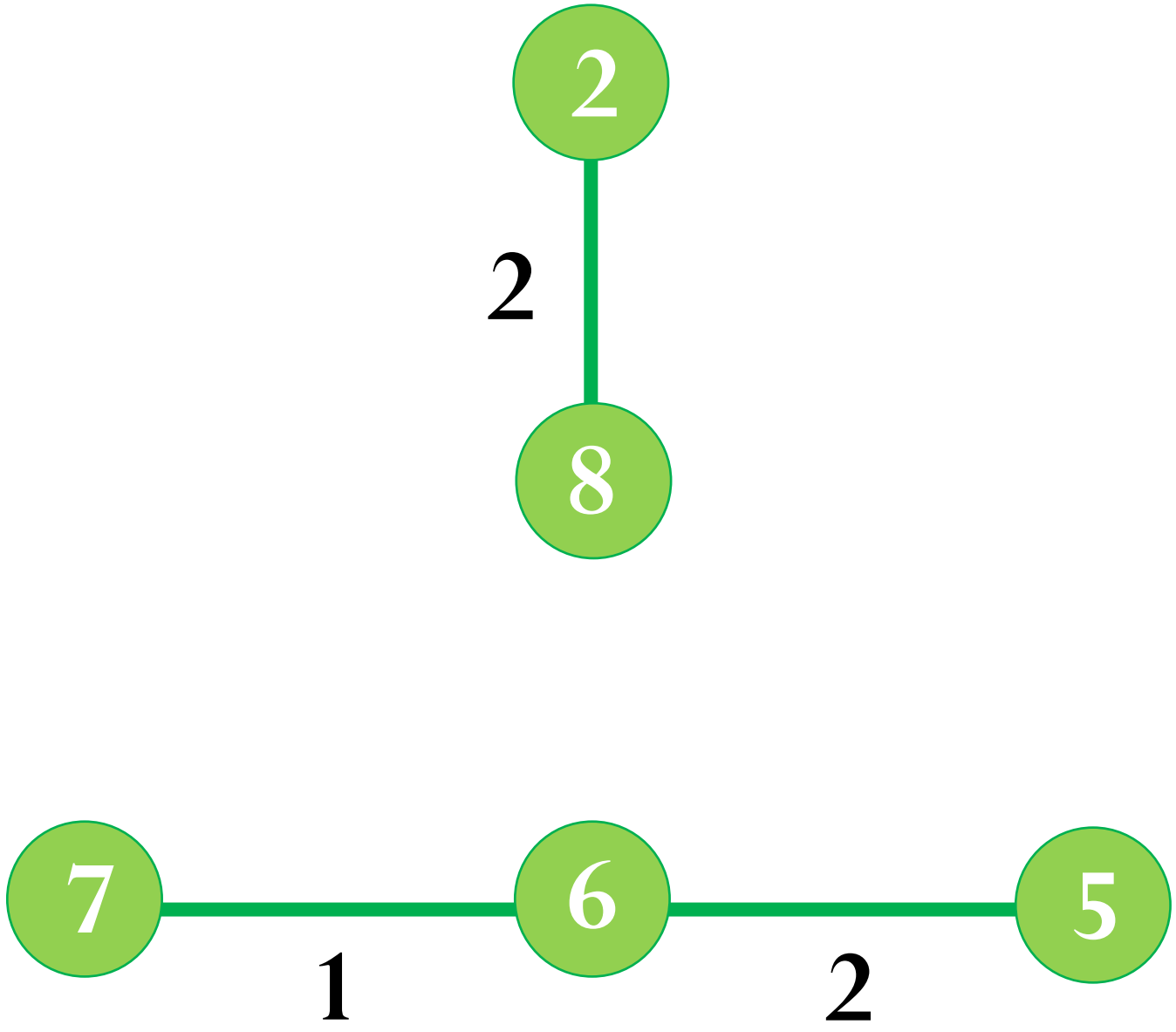
Step 1: Pick edge 7-6. No cycle is formed, include it.



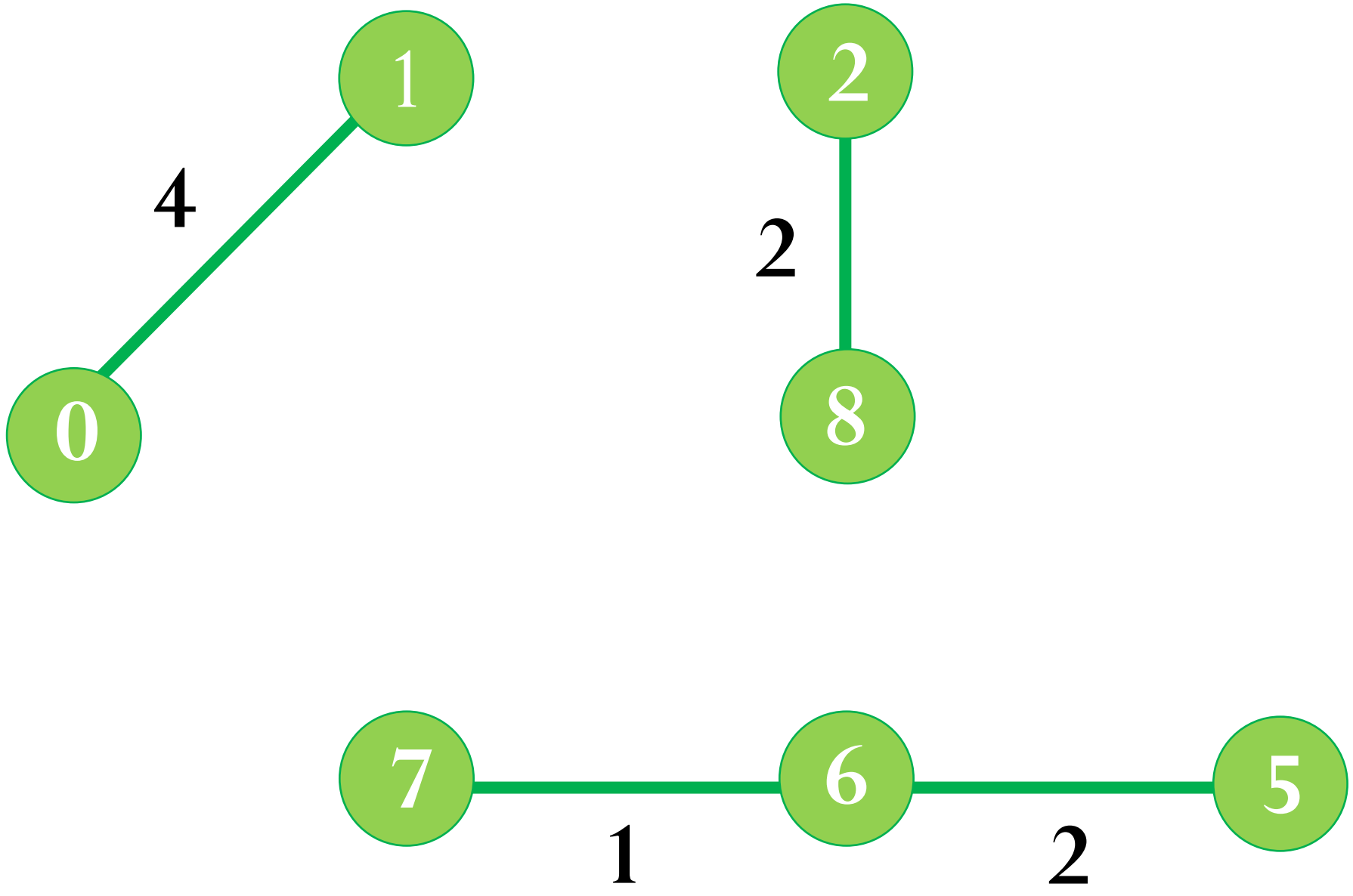
Step 2: Pick edge 8-2. No cycle is formed, include it.



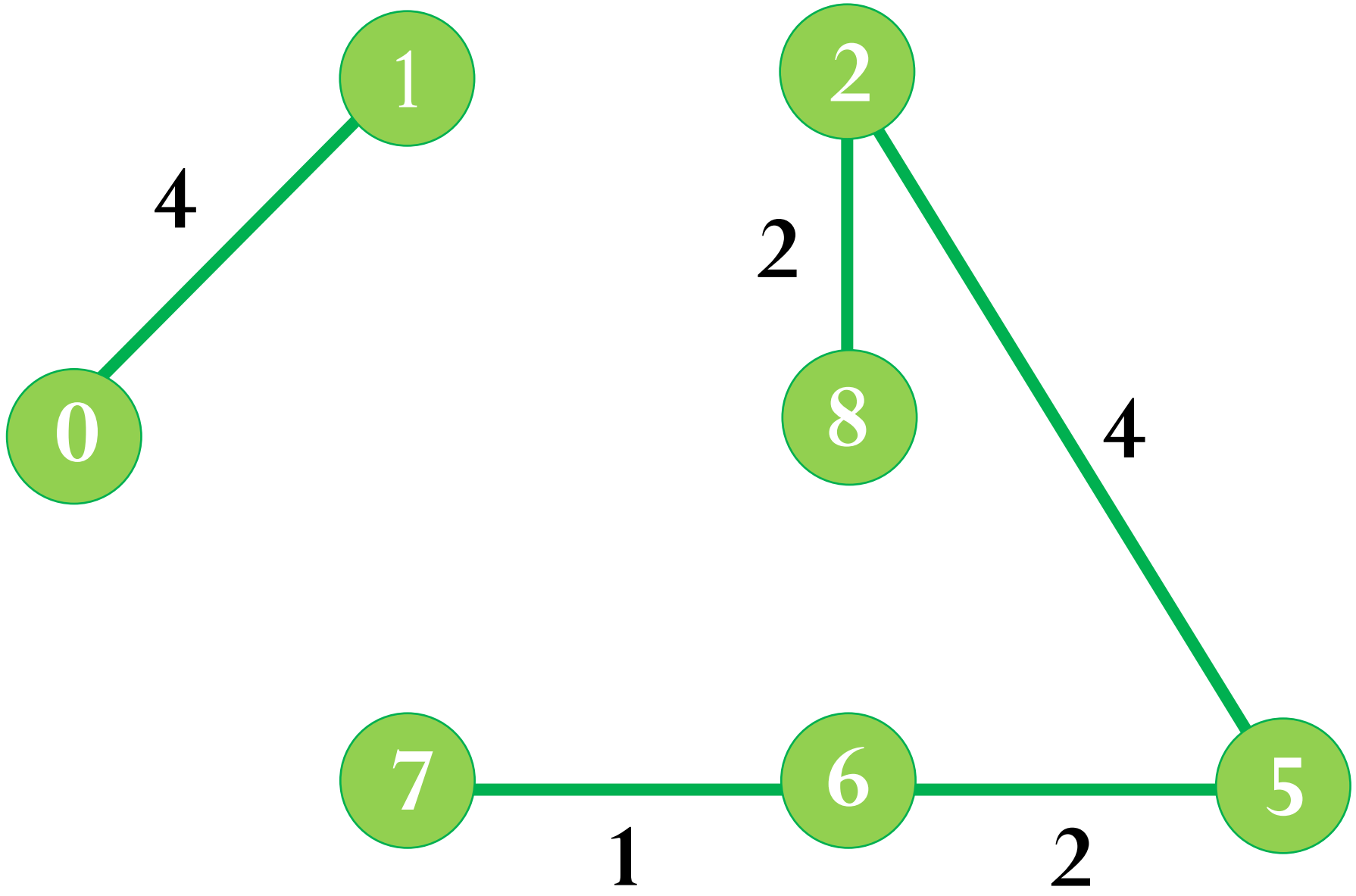
Step 3: Pick edge 6-5. No cycle is formed, include it.



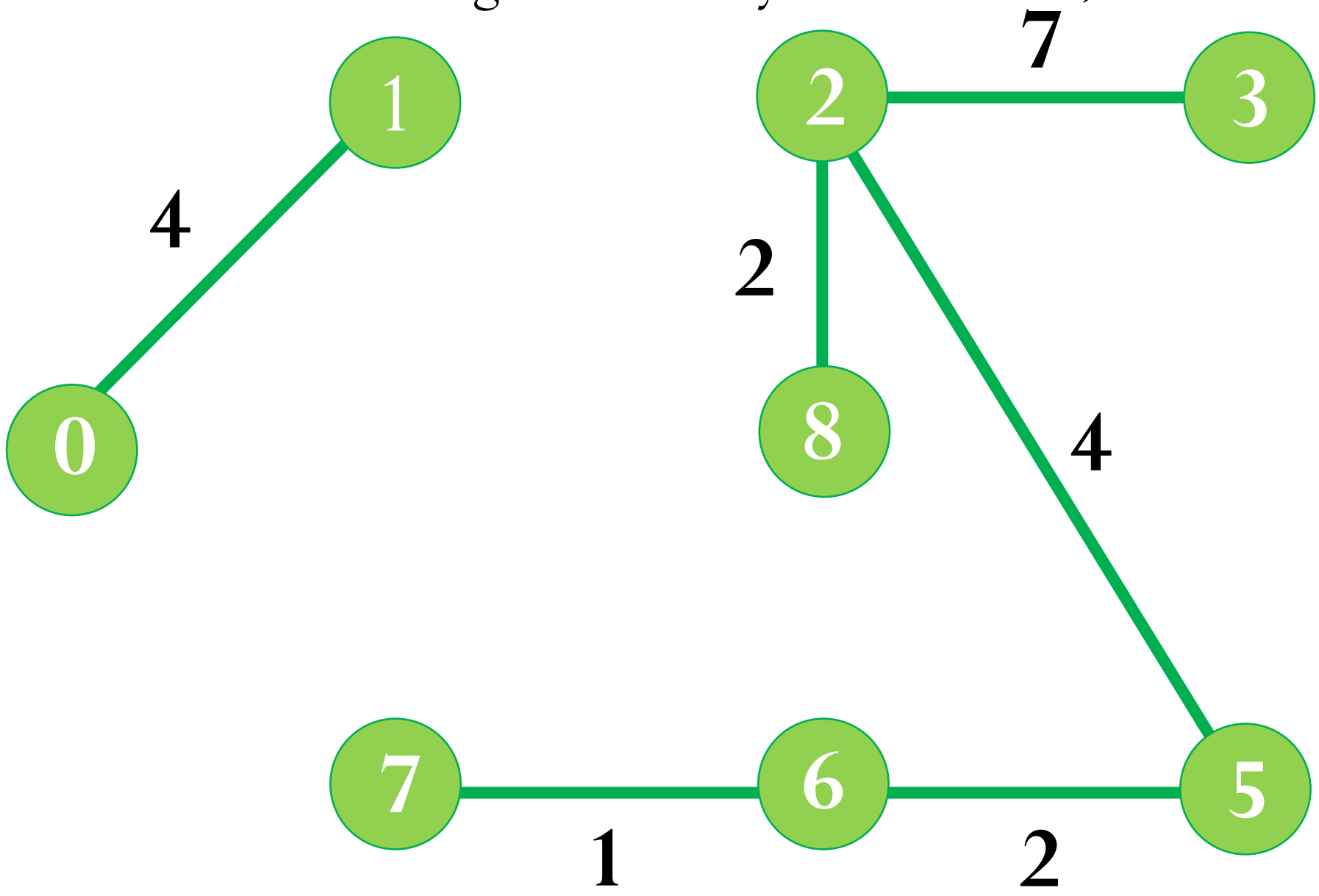
Step 4: Pick edge 0-1. No cycle is formed, include it.



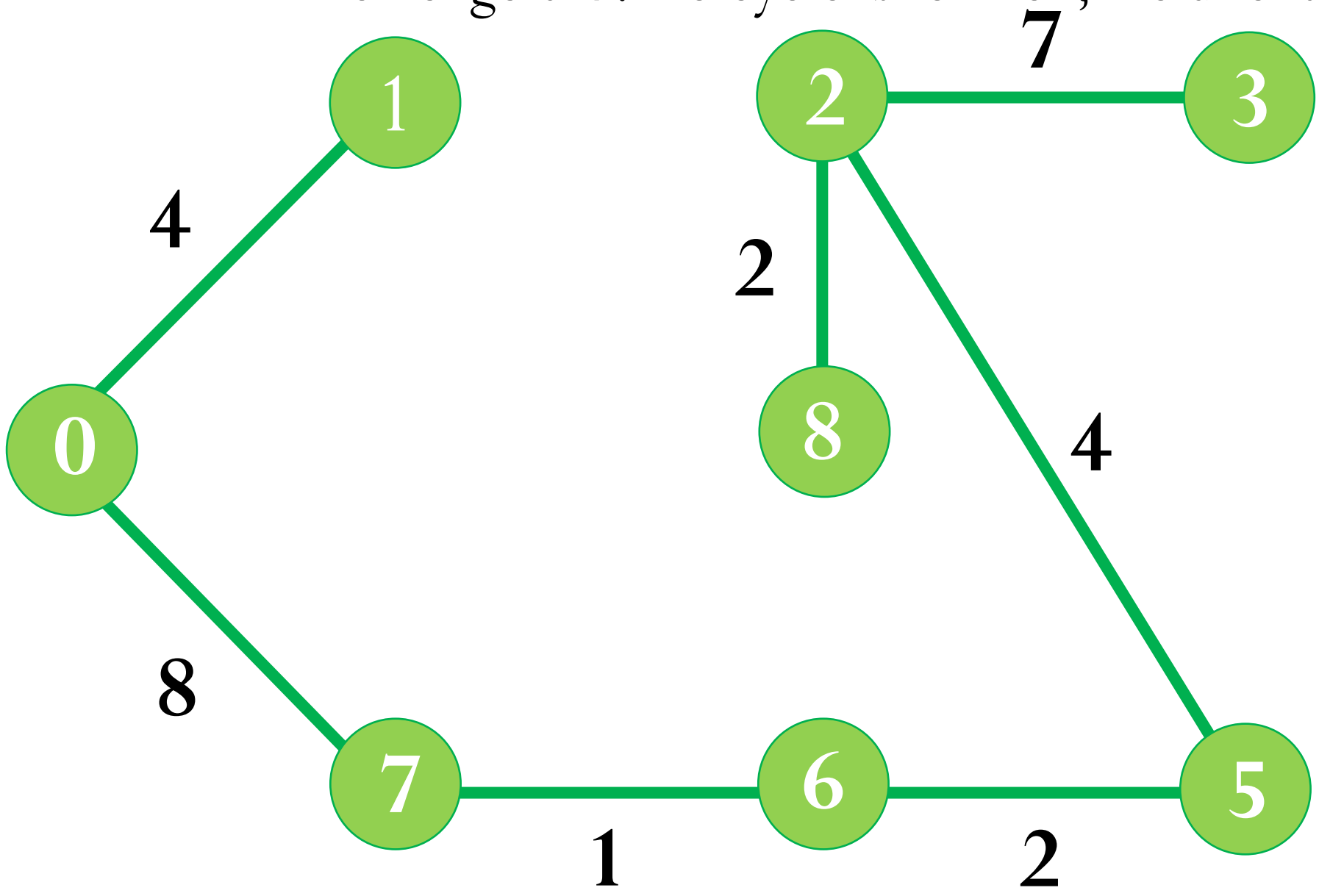
Step 5: Pick edge 2-5. No cycle is formed, include it.



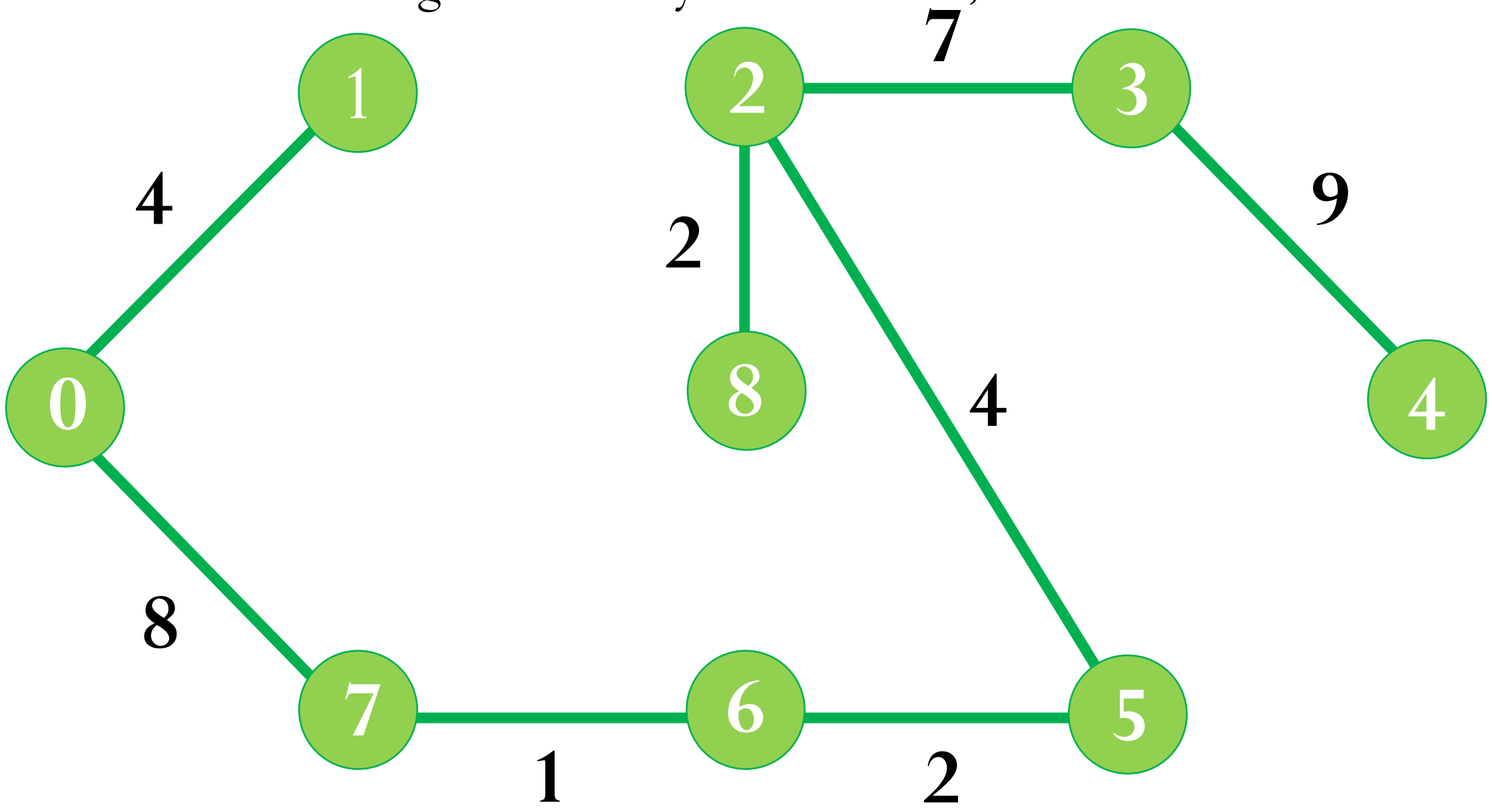
Step 6: Pick edge 8-6. Since including this edge results in the cycle, discard it.
Pick edge 2-3: No cycle is formed, include it.



Step 7: Pick edge 7-8. Since including this edge results in the cycle, discard it.
Pick edge 0-7. No cycle is formed, include it.



Step 8: Pick edge 1-2. Since including this edge results in the cycle, discard it.
Pick edge 3-4. No cycle is formed, include it.



Prim's Algorithm

Prim's algorithm is another greedy algorithm that finds a ***minimum*** spanning tree (MST) for a connected, undirected graph. It starts with an arbitrary vertex and repeatedly expands the MST *by adding* the shortest edge that connects a vertex inside the MST to a vertex outside the MST. Prim's algorithm can be implemented using a priority queue or a min-heap to efficiently select the next edge to add to the MST.

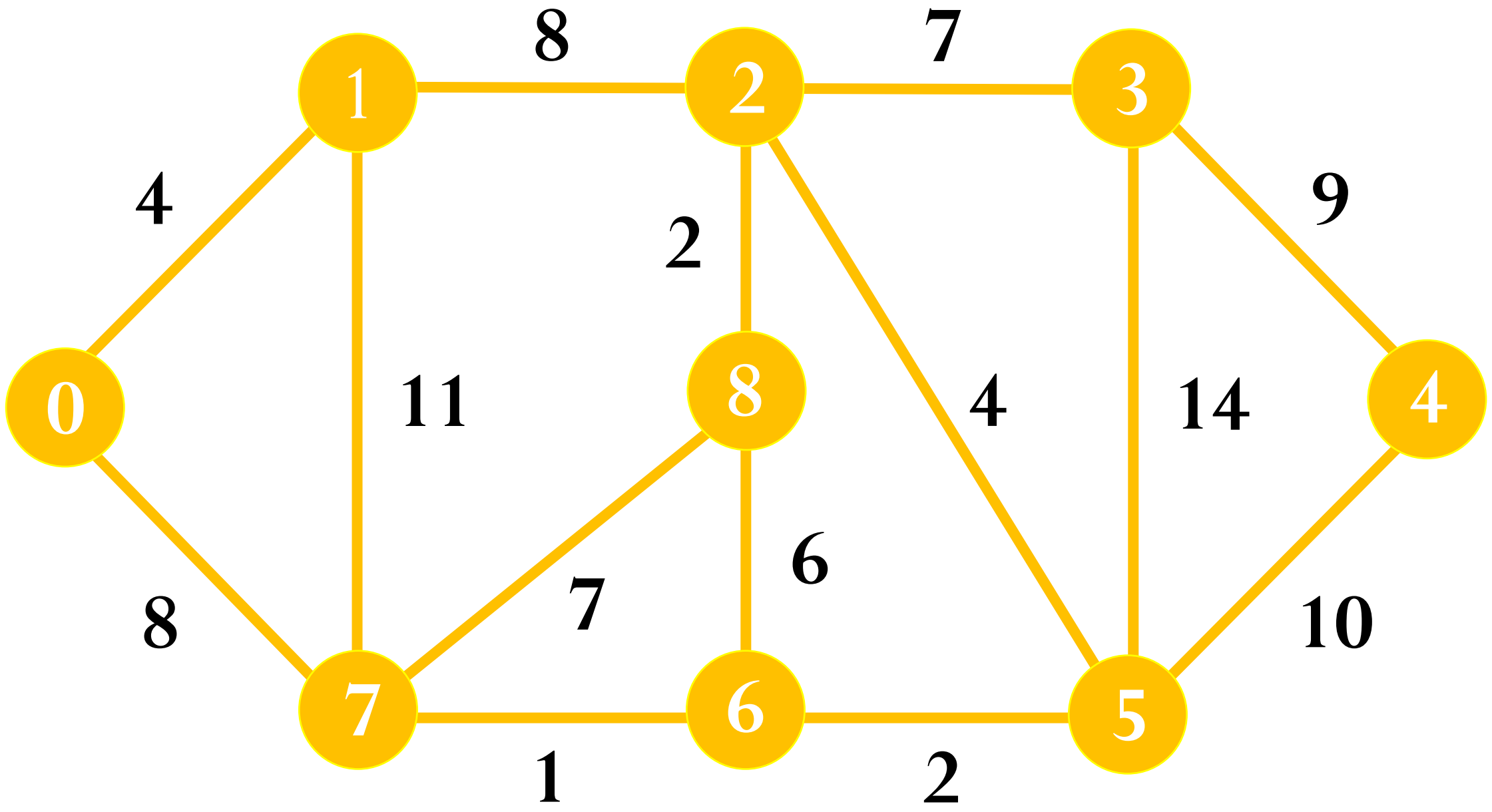
Prim's Algorithm

The algorithm begins by selecting an arbitrary starting vertex as the root of the MST and setting its key value to 0.

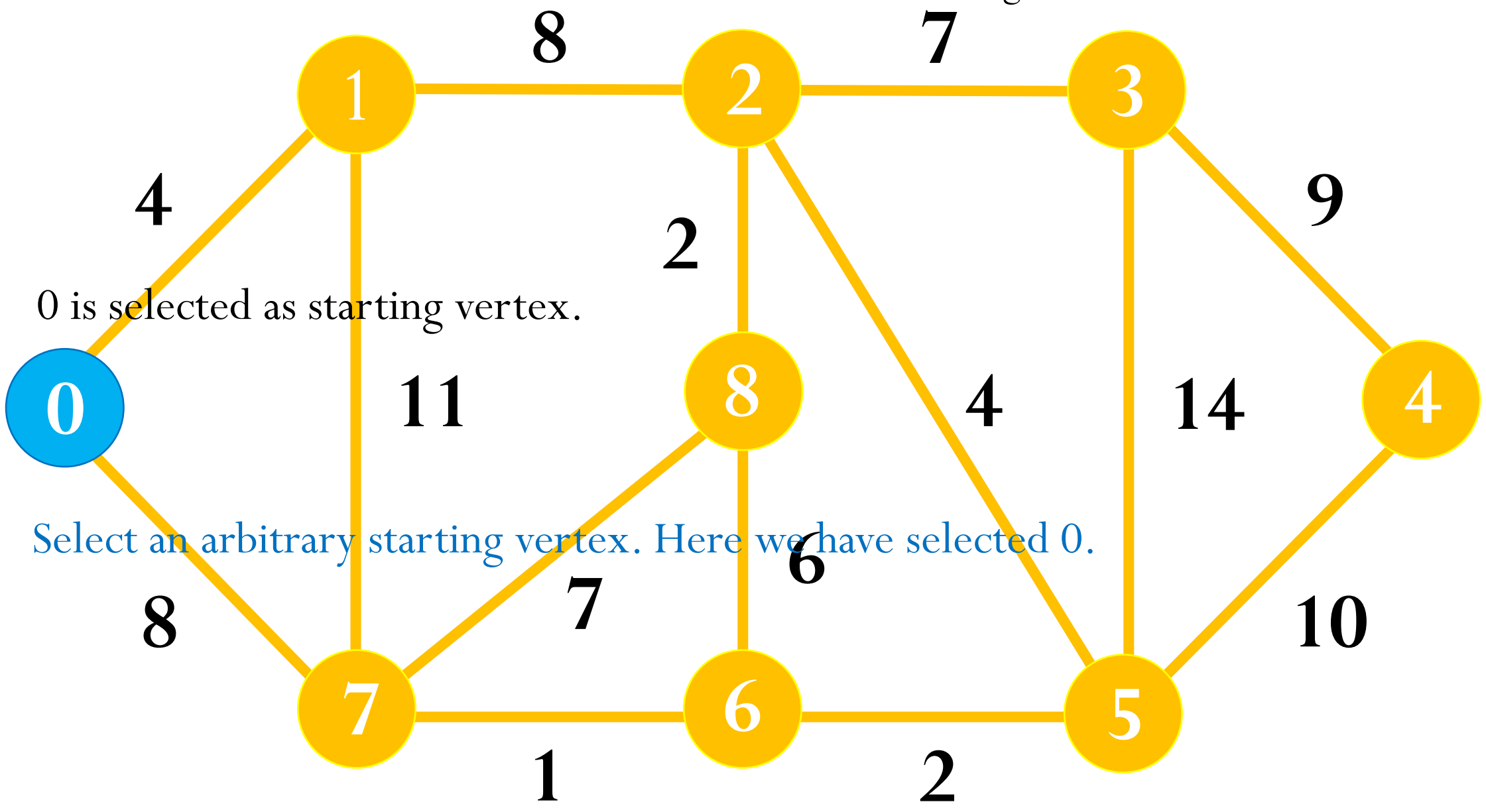
It then inserts the root vertex into a priority queue.

The algorithm repeatedly selects the vertex with the minimum key value from the priority queue and adds the corresponding edge to the MST. It updates the key values of adjacent vertices and inserts them into the priority queue if necessary. This process continues until the priority queue is empty. Finally, the list of edges in the MST is returned.

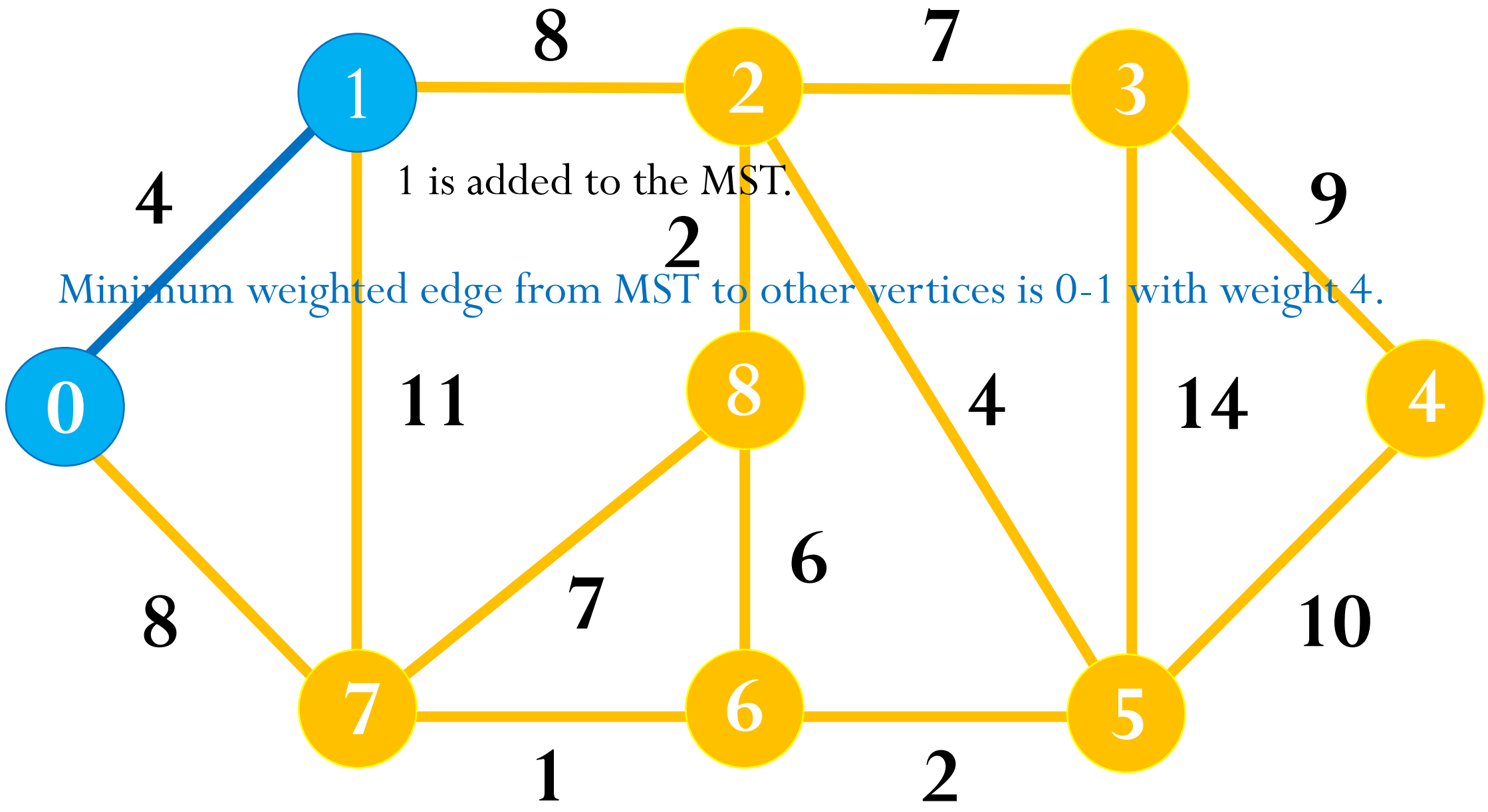
Prim's Algorithm



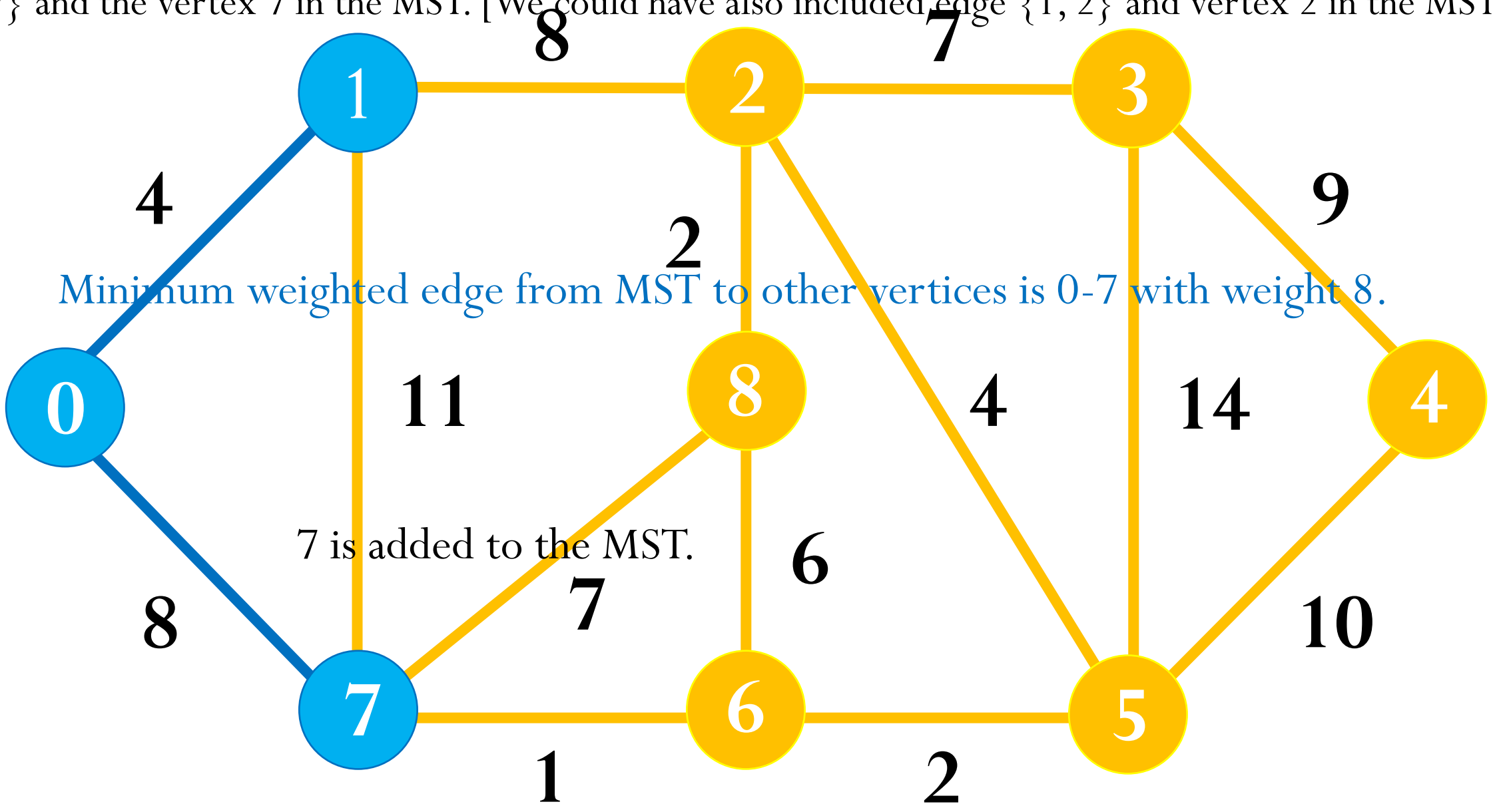
Step 1: Firstly, we select an arbitrary vertex that acts as the starting vertex of the Minimum Spanning Tree.
Here we have selected vertex 0 as the starting vertex.



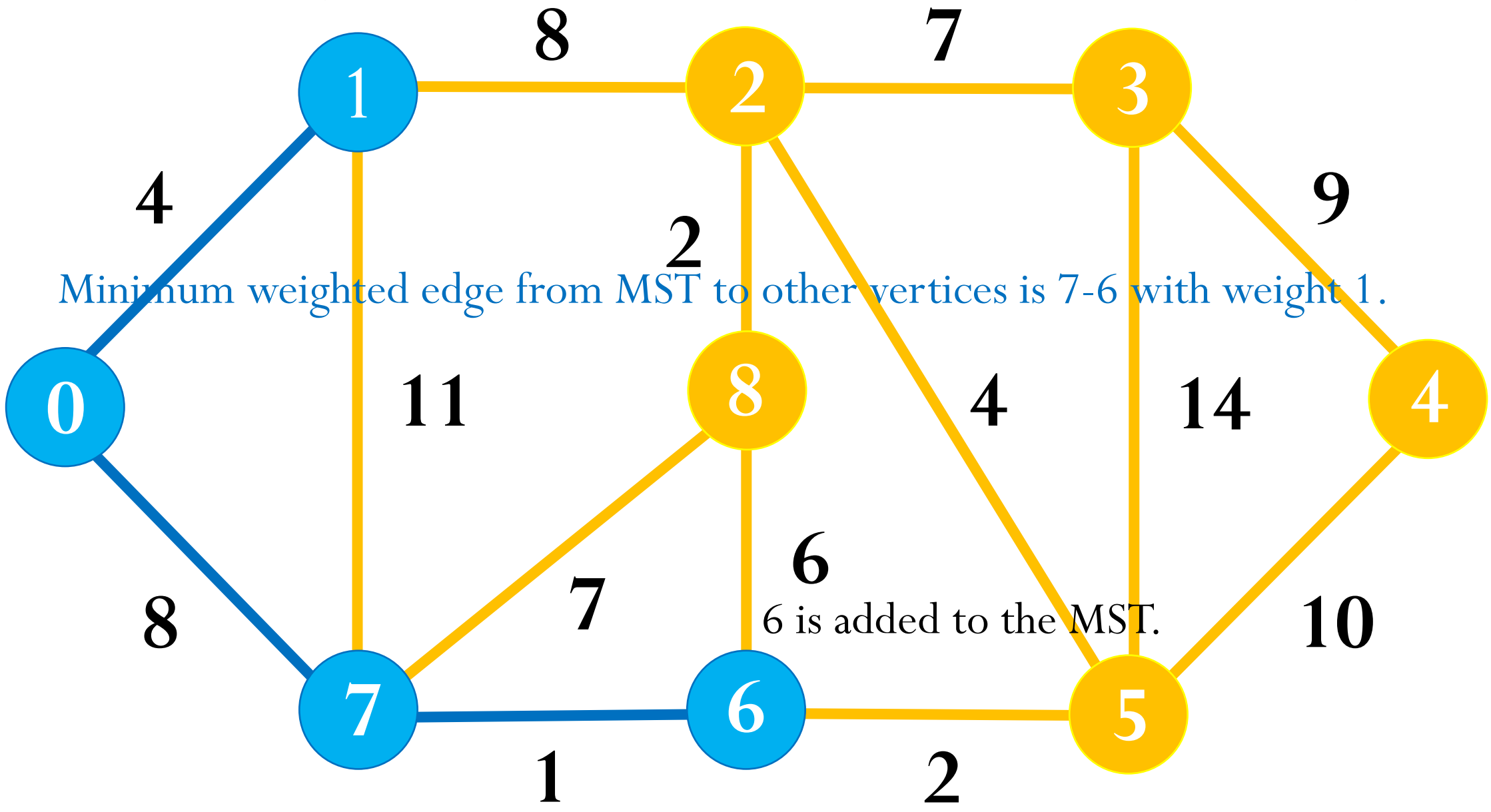
Step 2: All the edges connecting the incomplete MST and other vertices are the edges {0, 1} and {0, 7}. Between these two the edge with minimum weight is {0, 1}. So include the edge and vertex 1 in the MST.



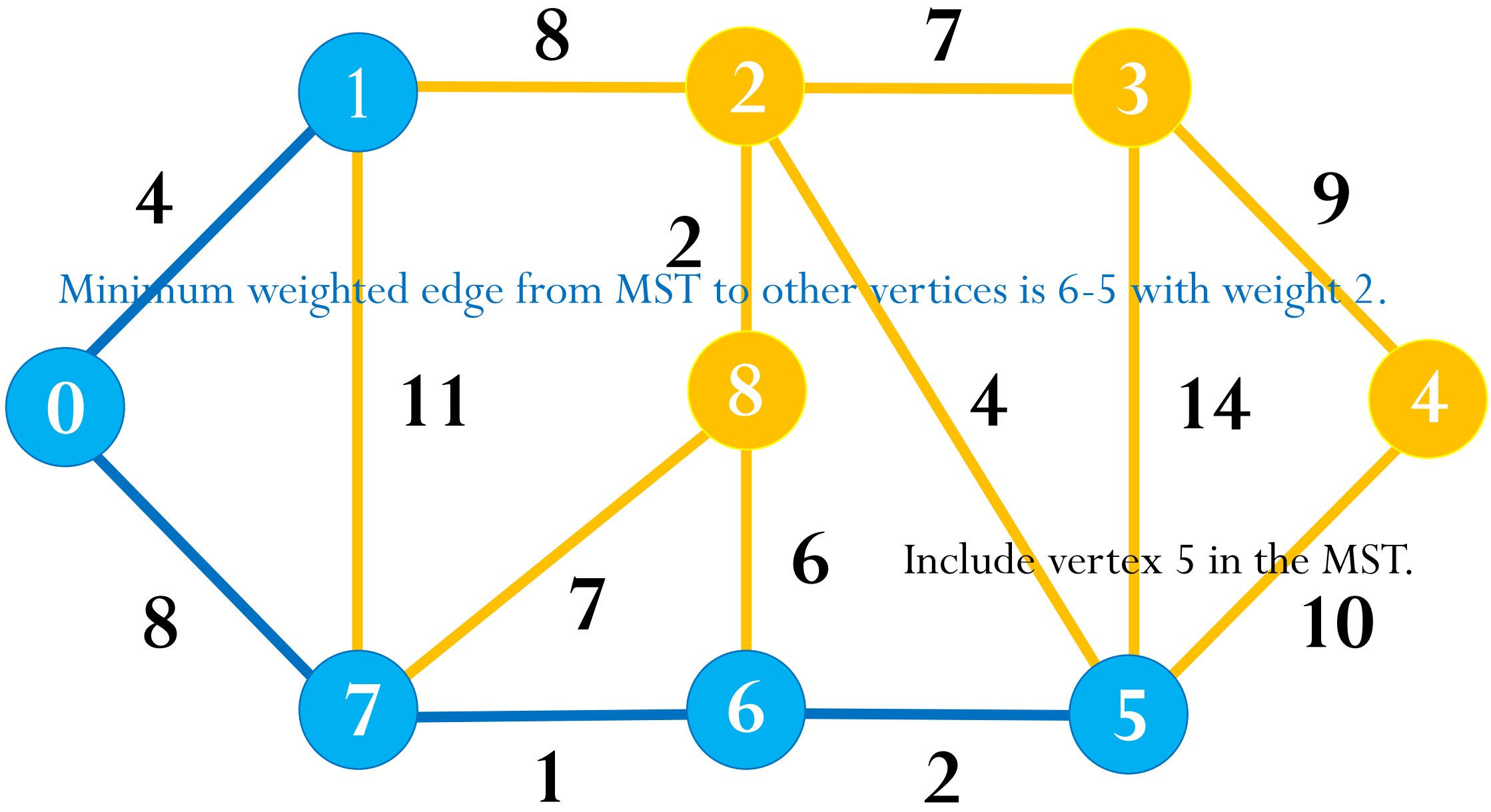
Step 3: The edges connecting the incomplete MST to other vertices are {0, 7}, {1, 7} and {1, 2}. Among these edges the minimum weight is 8 which is of the edges {0, 7} and {1, 2}. Let us here include the edge {0, 7} and the vertex 7 in the MST. [We could have also included edge {1, 2} and vertex 2 in the MST].



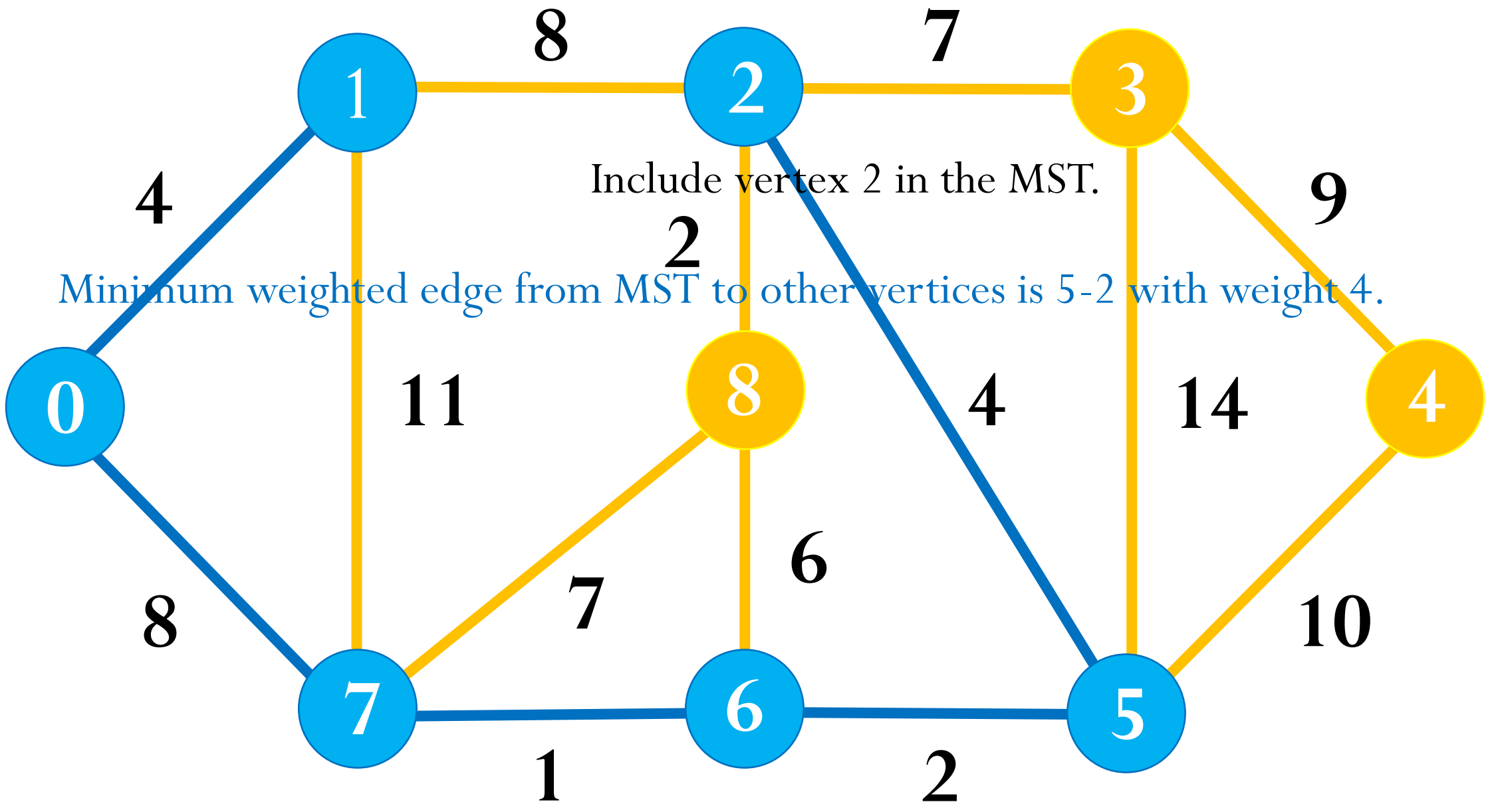
Step 4: The edges that connect the incomplete MST with the fringe vertices are {1, 2}, {7, 6} and {7, 8}. Add the edge {7, 6} and the vertex 6 in the MST as it has the least weight (i.e., 1).



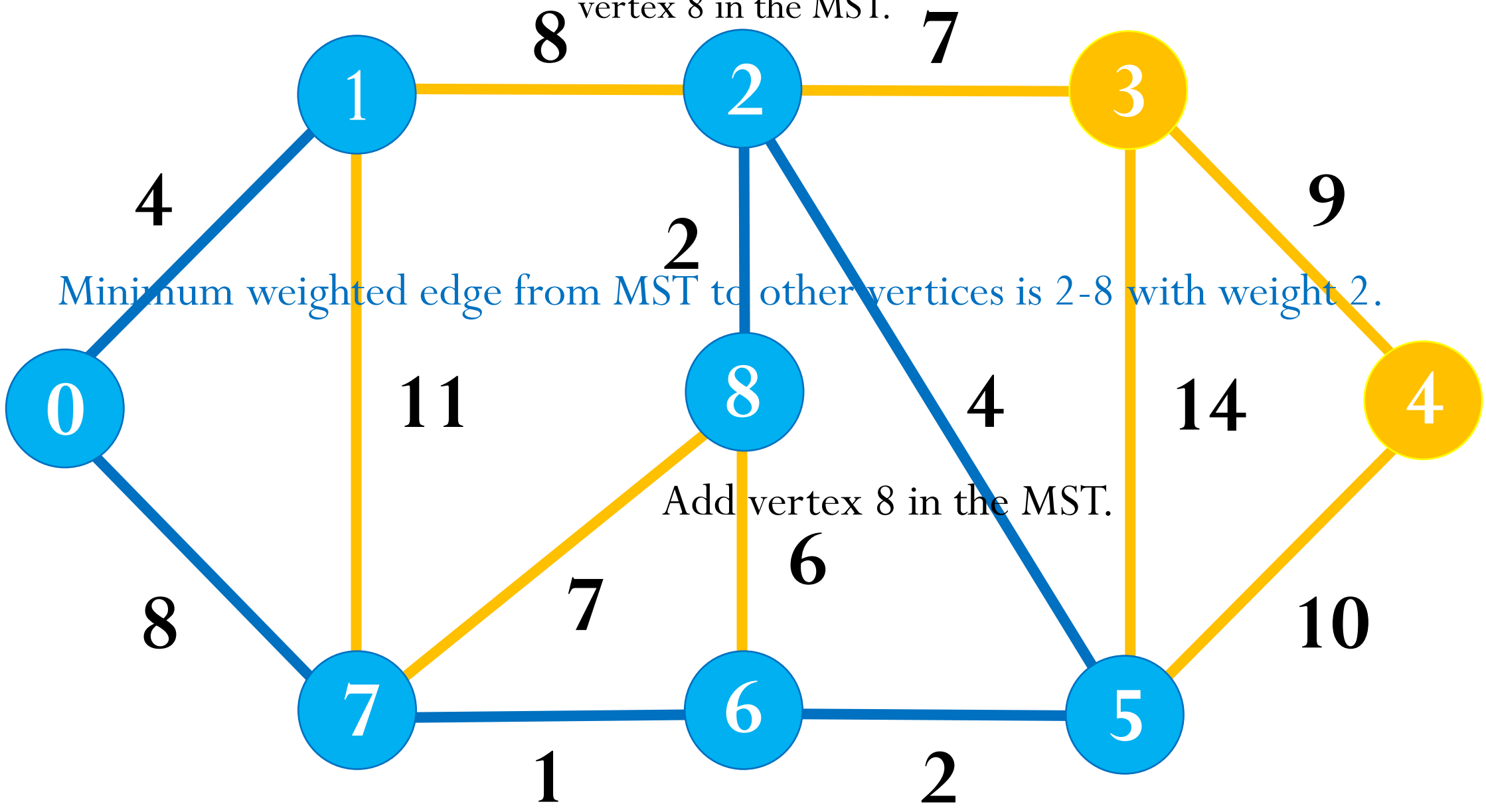
Step 5: The connecting edges now are {7, 8}, {1, 2}, {6, 8} and {6, 5}. Include edge {6, 5} and vertex 5 in the MST as the edge has the minimum weight (i.e., 2) among them.



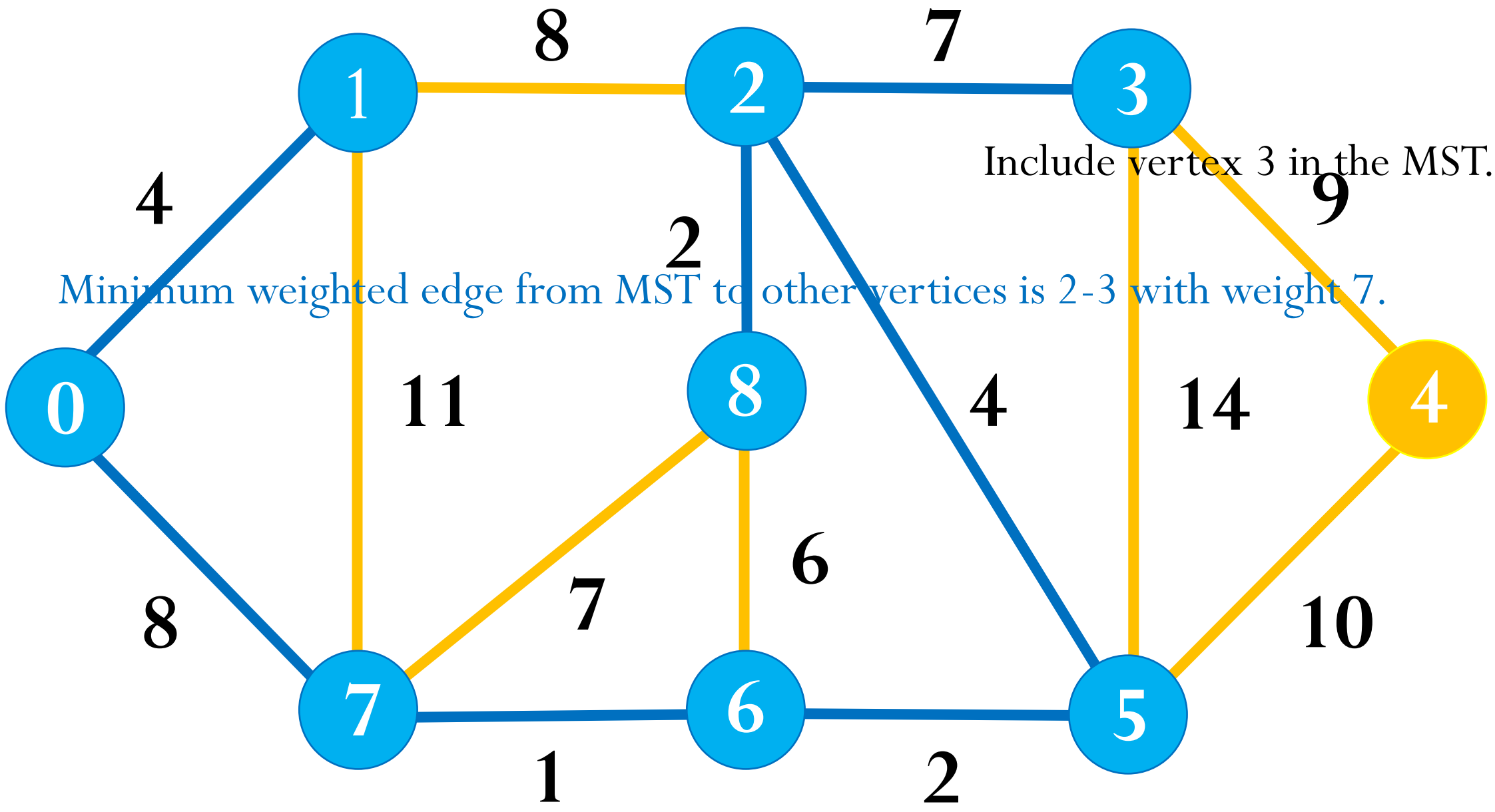
Step 6: Among the current connecting edges, the edge {5, 2} has the minimum weight. So include that edge and the vertex 2 in the MST.



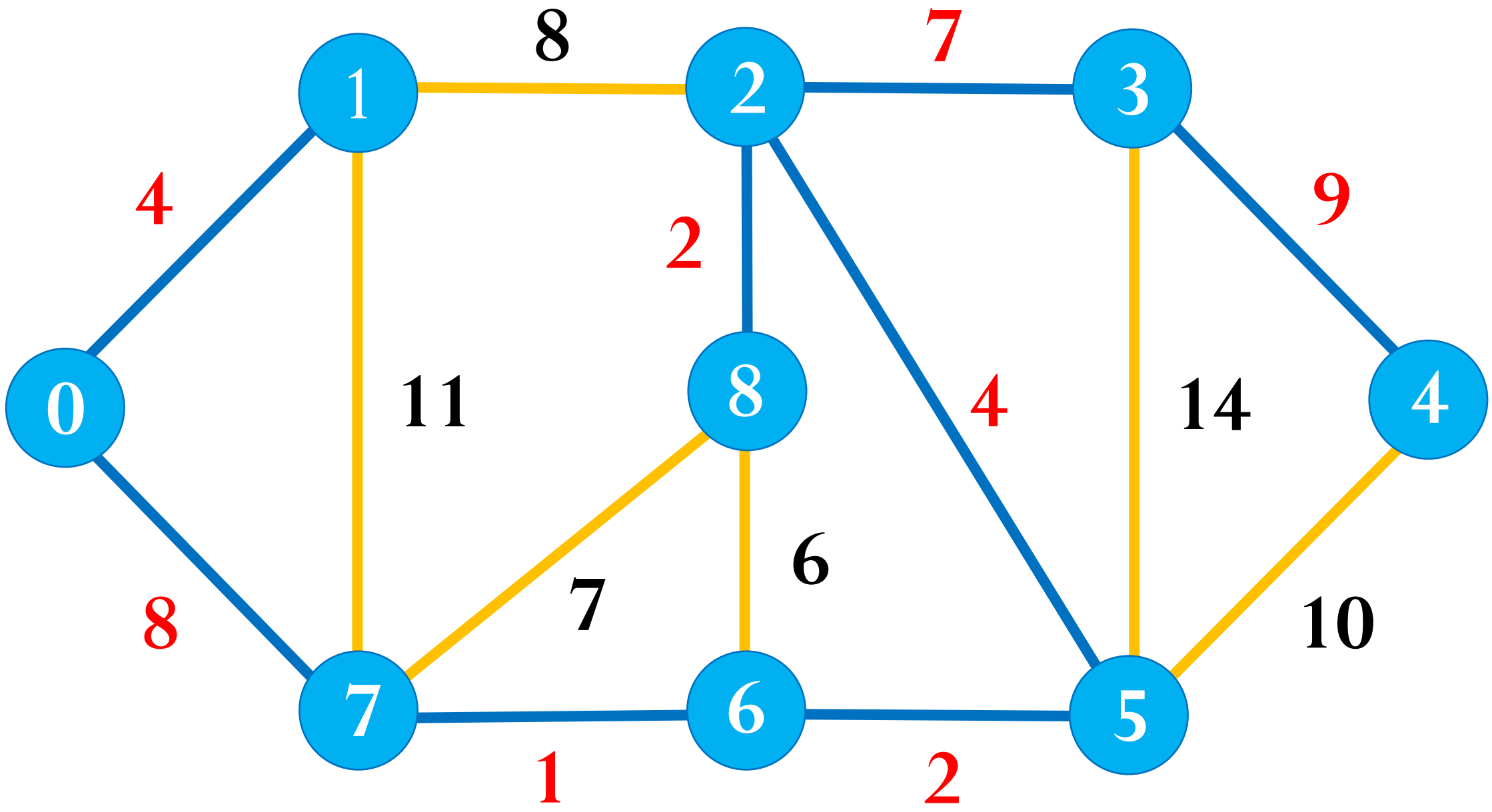
Step 7: The connecting edges between the incomplete MST and the other edges are $\{2, 8\}$, $\{2, 3\}$, $\{5, 3\}$ and $\{5, 4\}$. The edge with minimum weight is edge $\{2, 8\}$ which has weight 2. So include this edge and the vertex 8 in the MST.



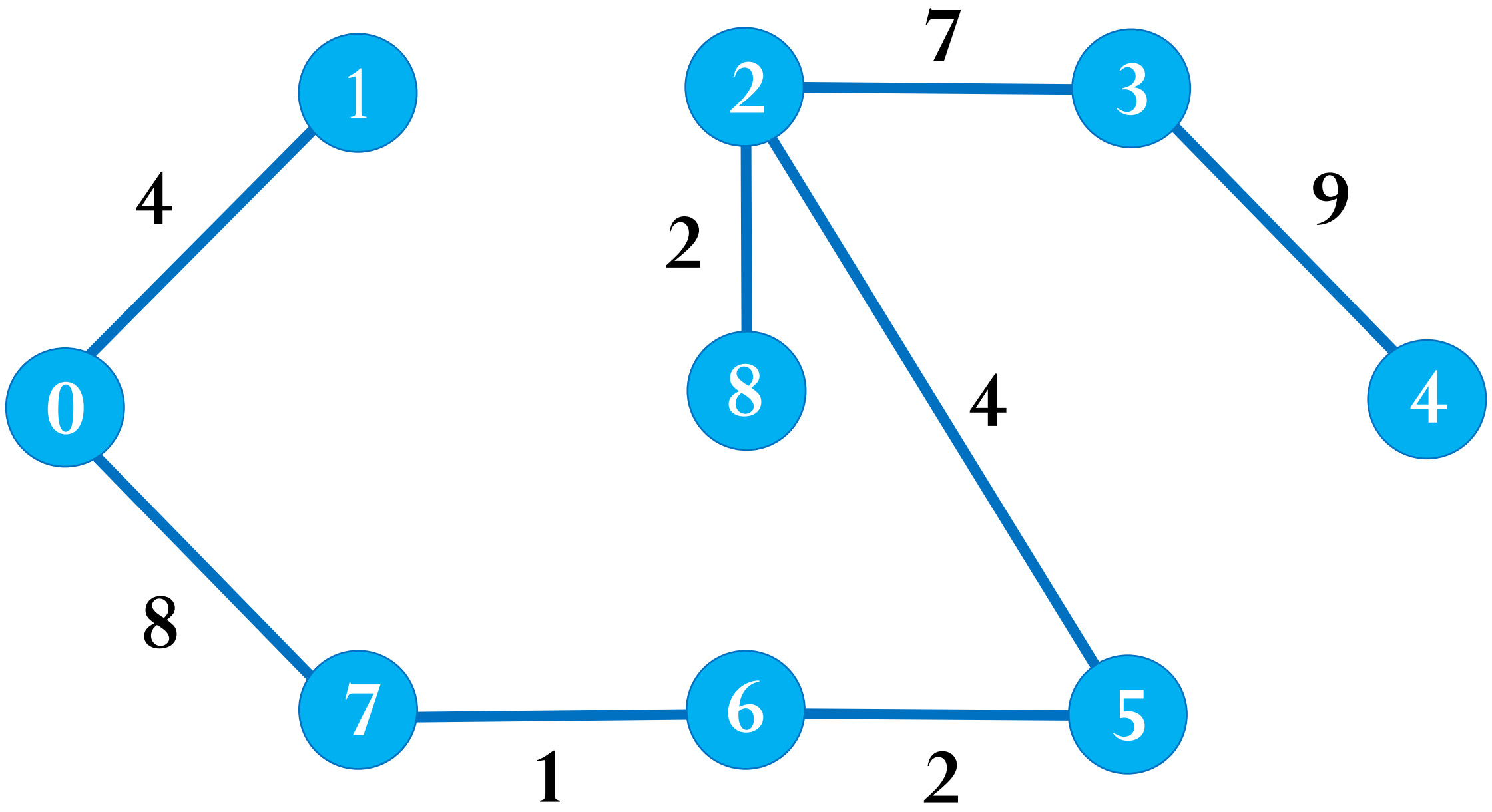
Step 8: See here that the edges {7, 8} and {2, 3} both have same weight which are minimum. But 7 is already part of MST. So we will consider the edge {2, 3} and include that edge and vertex 3 in the MST.



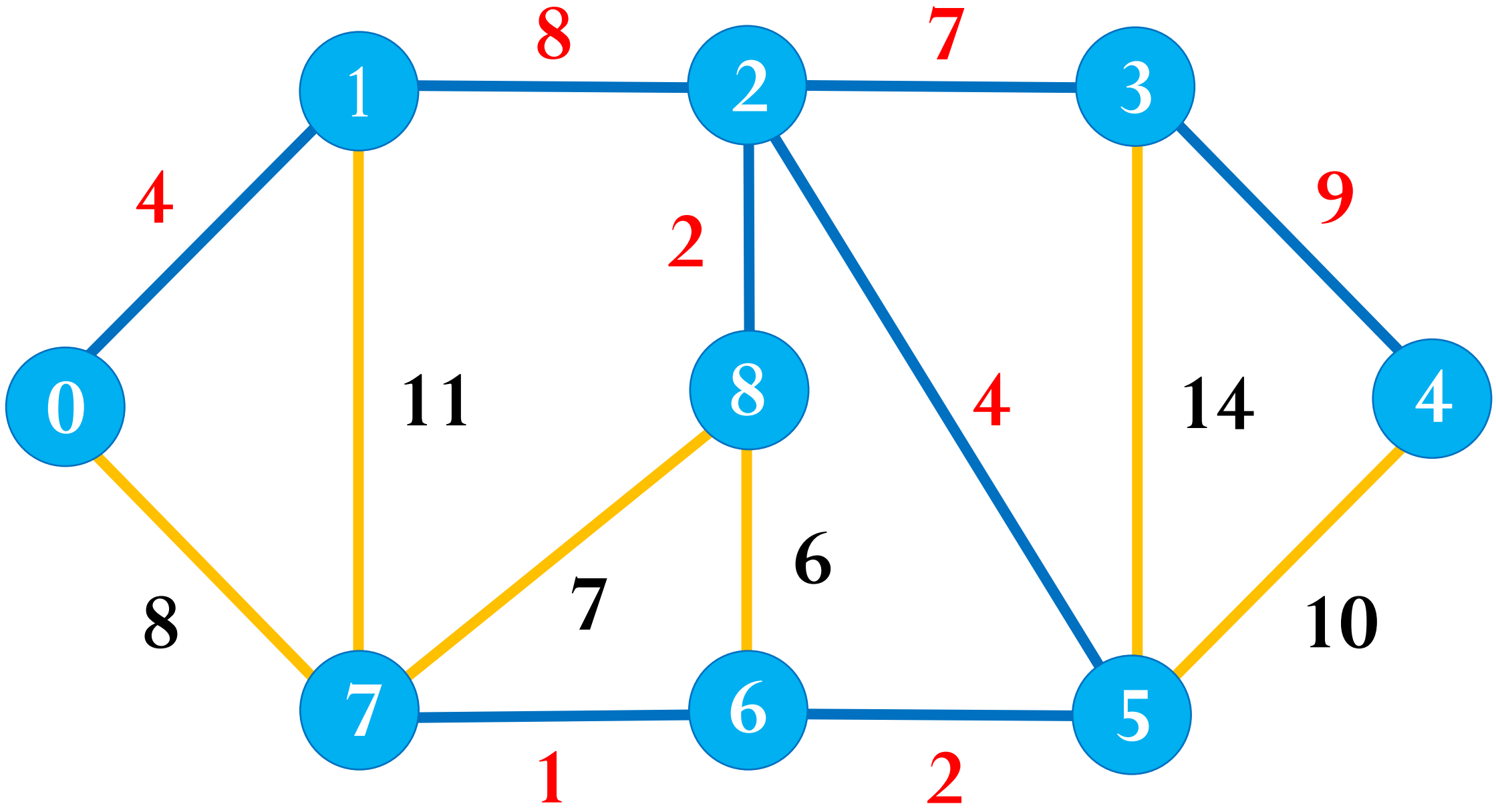
The final structure of the MST is as follows and the weight of the edges of the MST is $(4 + 8 + 1 + 2 + 4 + 2 + 7 + 9) = 37$.



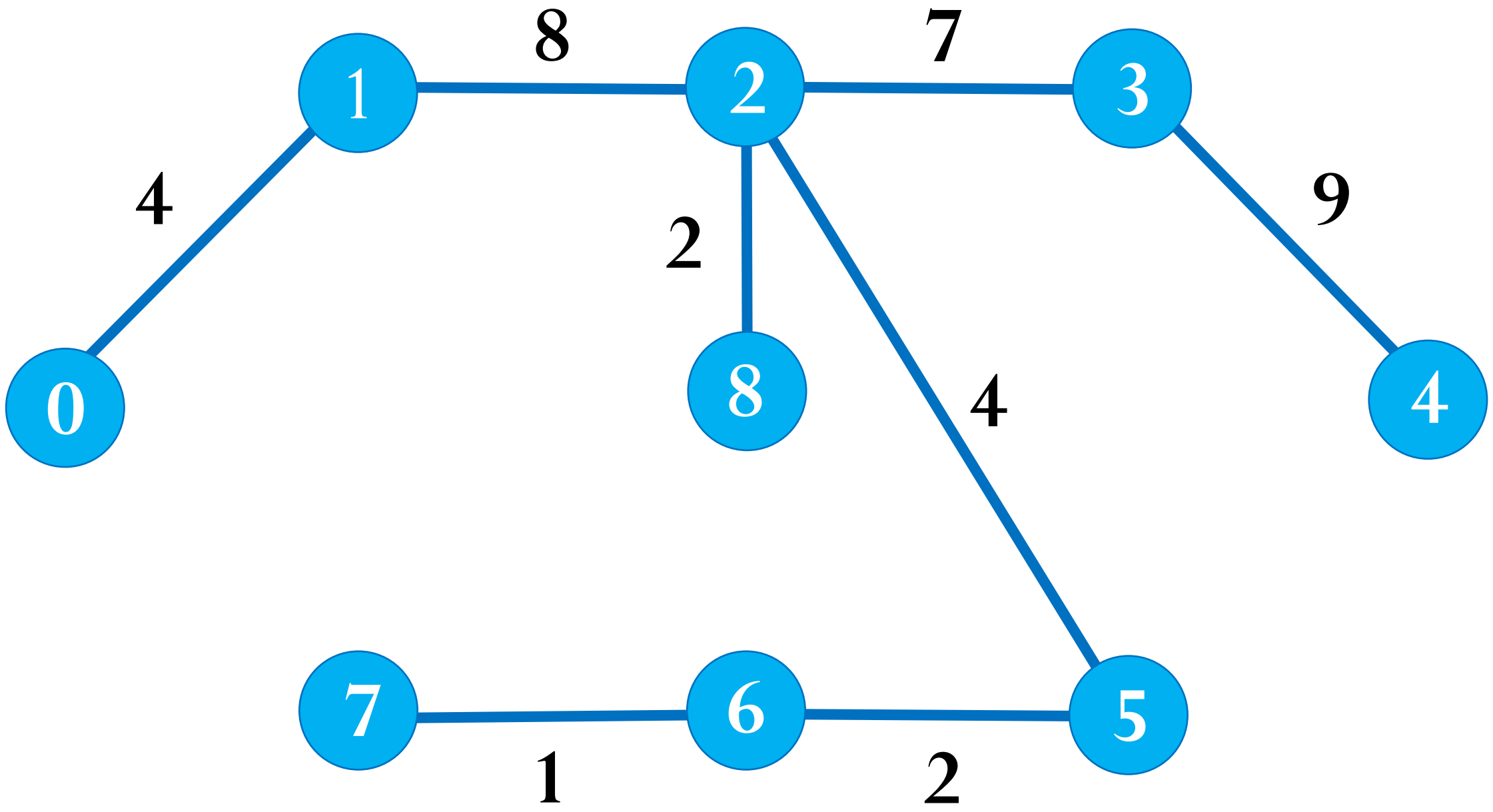
The final structure of MST



If we had selected the edge 1-2 in the third step then the MST would look like the following.



Structure of the alternate MST if we had selected edge {1, 2} in the MST

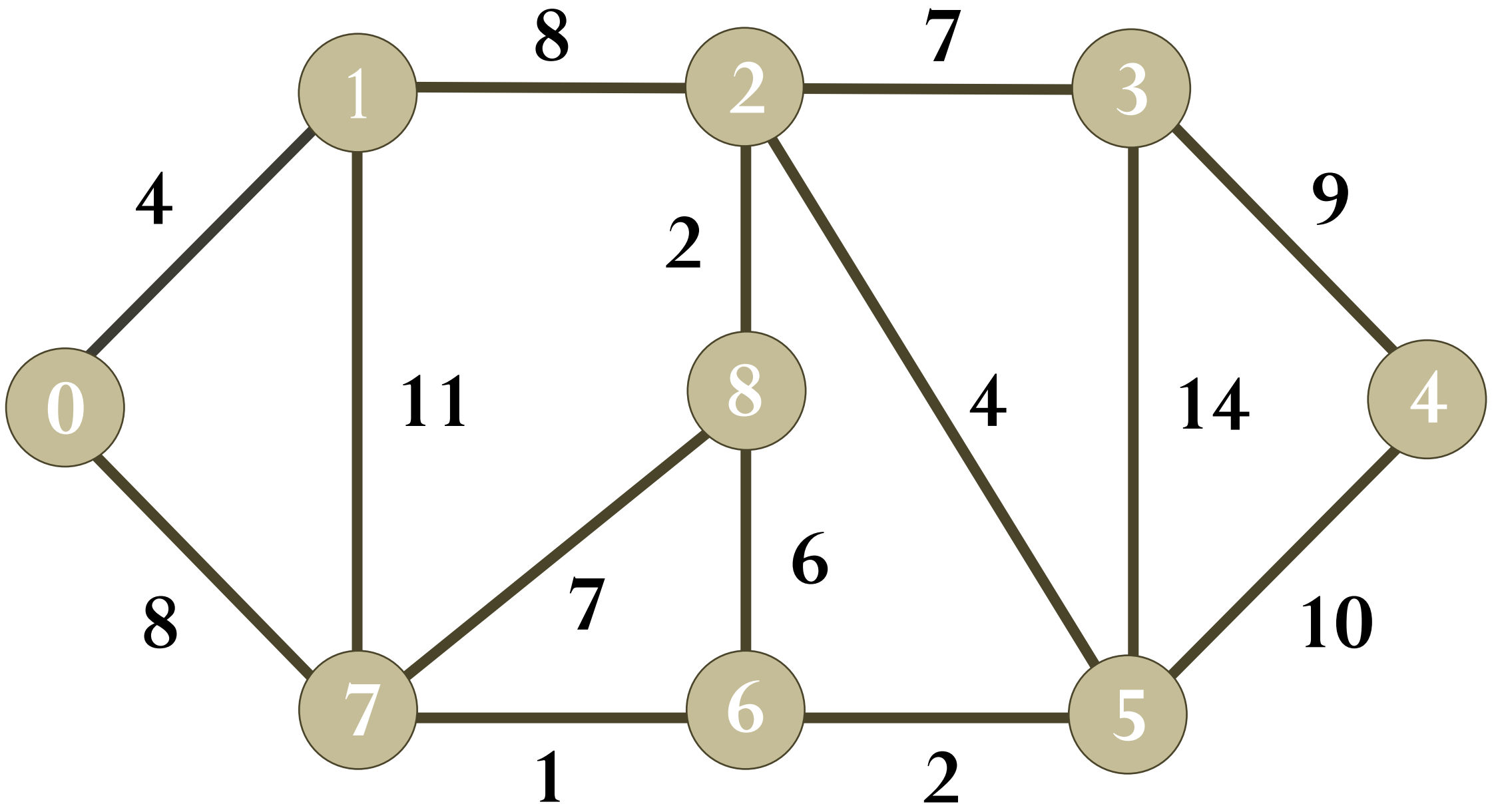


Borůvka's Algorithm

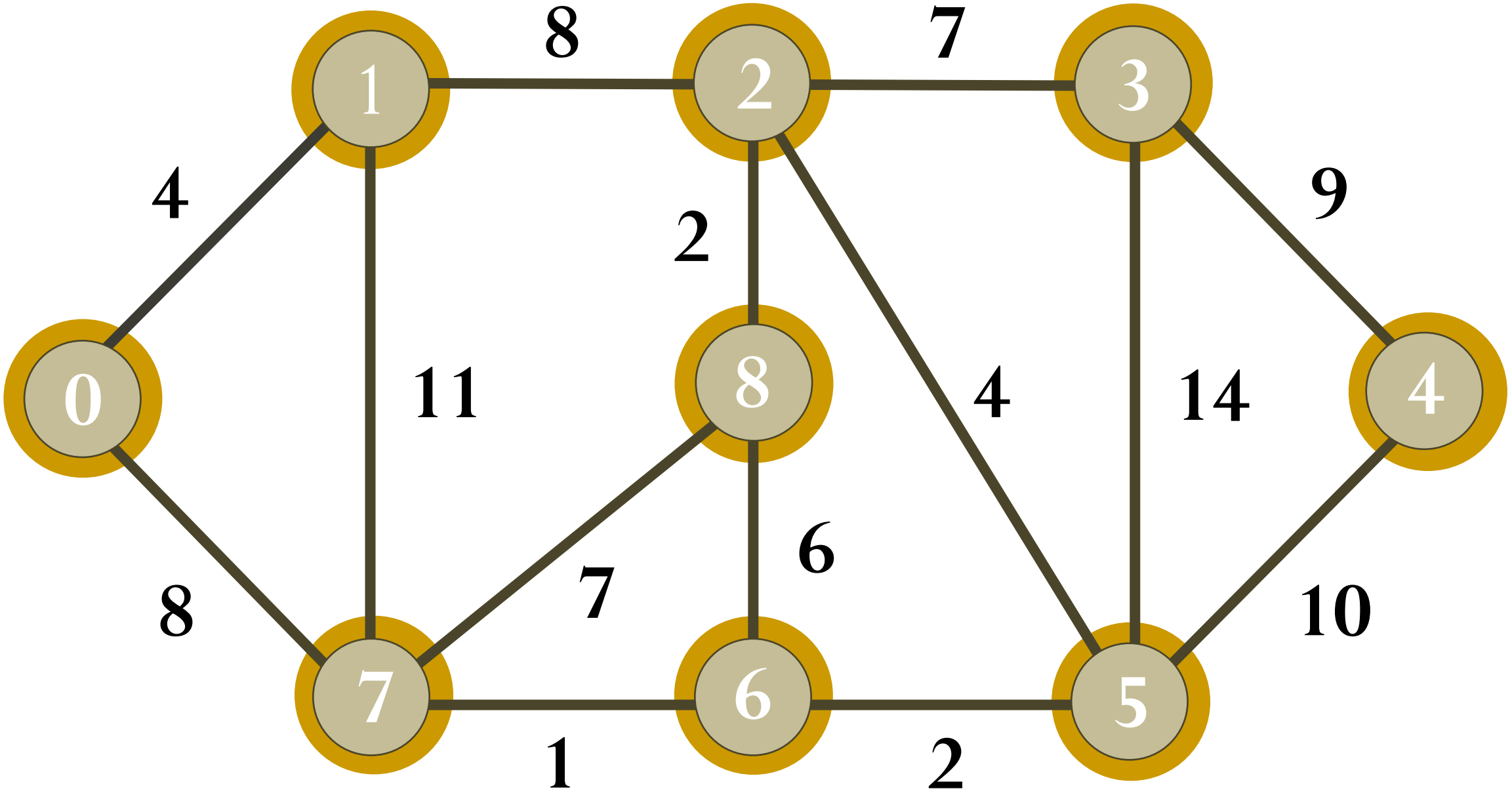
Borůvka's algorithm is a parallel algorithm that finds a **minimum** spanning tree for a connected, undirected graph. It works by repeatedly selecting the cheapest edge incident to each vertex and adding it to the MST.

Borůvka's algorithm can be parallelized to run efficiently on parallel computing architectures.

Borůvka's Algorithm



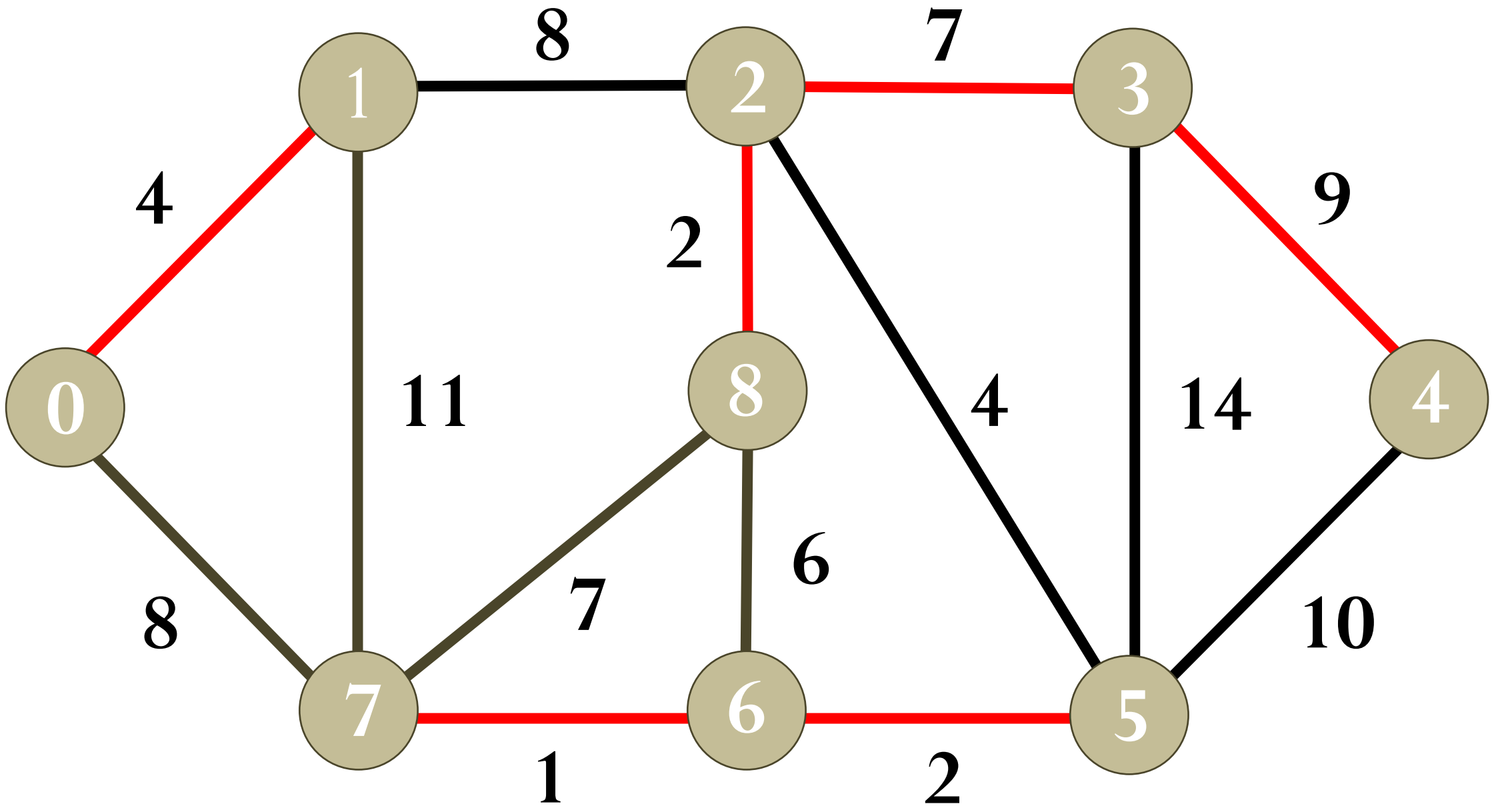
Initially, MST is empty. Every vertex is single component as highlighted in brown color in the below diagram.



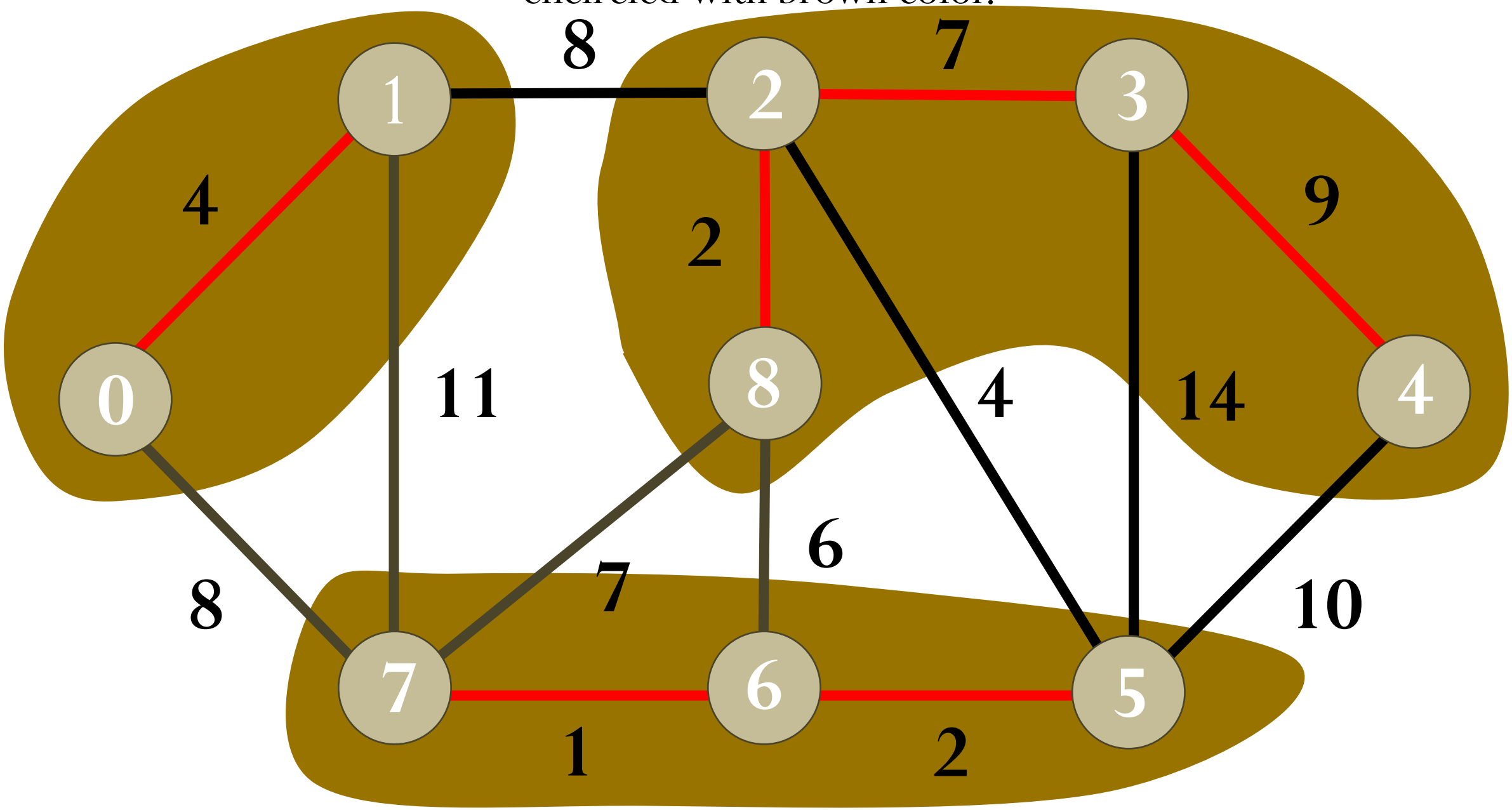
For every component, find the cheapest edge that connects it to some other component.

Component	Cheapest Edge that connects it to some other component
{0}	0-1
{1}	0-1
{2}	2-8
{3}	2-3
{4}	3-4
{5}	5-6
{6}	6-7
{7}	6-7
{8}	2-8

The cheapest edges are highlighted in red. Now MST becomes {0-1, 2-8, 2-3, 3-4, 5-6, 6-7}.



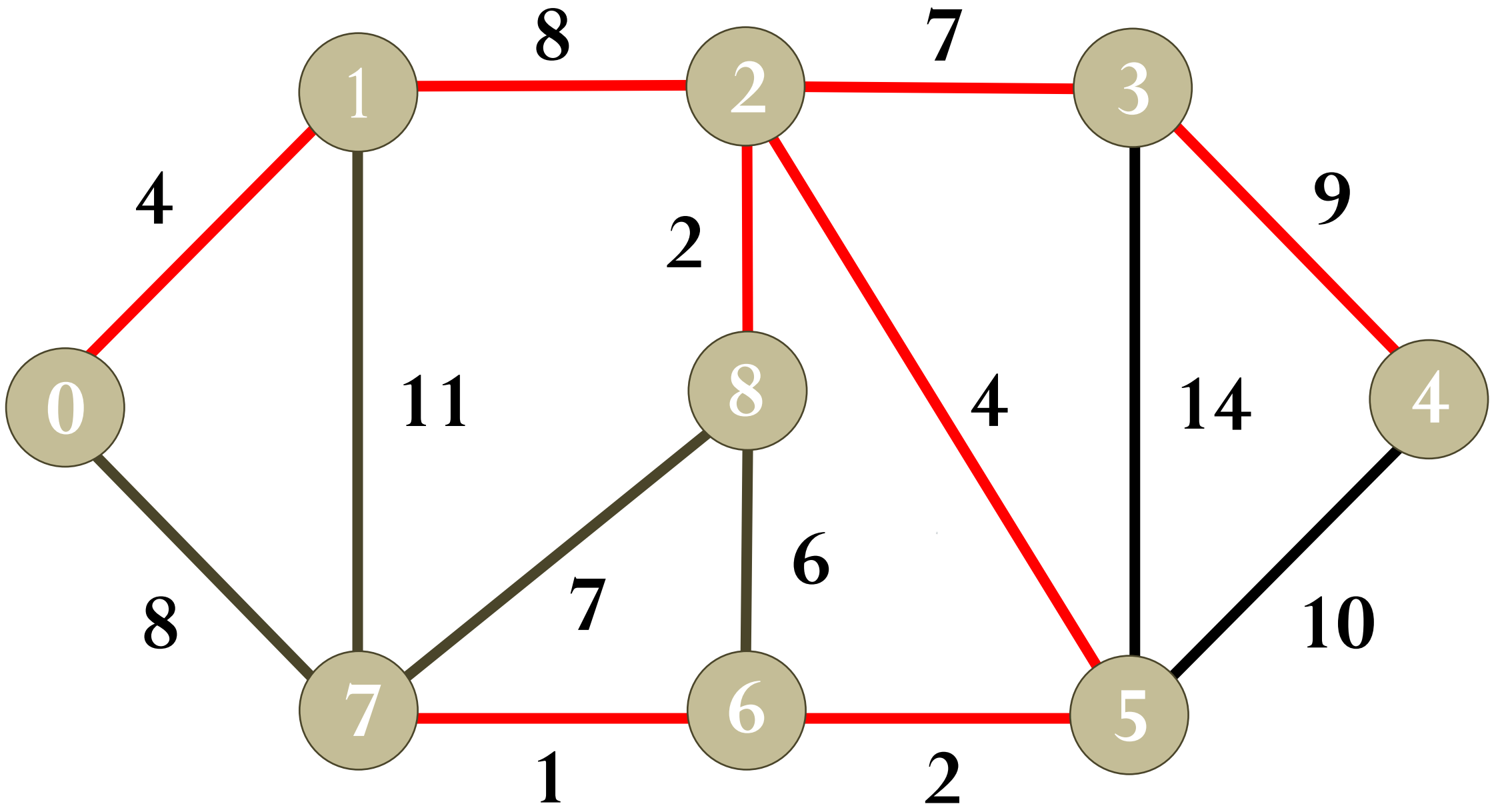
After above step, components are {{0,1}, {2,3,4,8}, {5,6,7}}. The components are encircled with brown color.



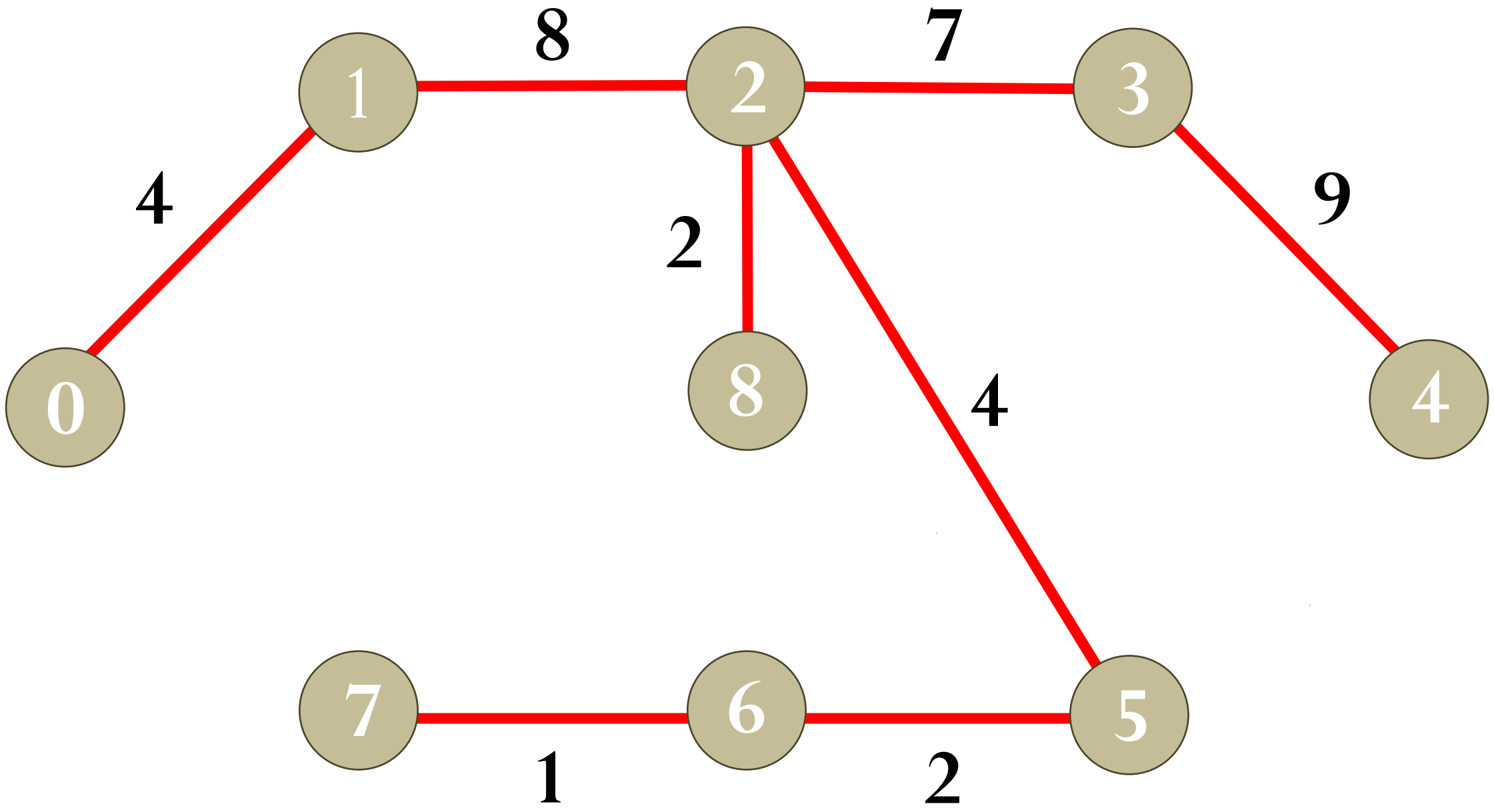
We again repeat the step, i.e., for every component, find the cheapest edge that connects it to some other component.

Component	Cheapest Edge that connects it to some other component
{0,1}	1-2 (or 0-7)
{2,3,4,8}	2-5
{5,6,7}	2-5

The cheapest edges are highlighted with red color. Now MST becomes {0-1, 2-8, 2-3, 3-4, 5-6, 6-7, 1-2, 2-5}.



At this stage, there is only one component {0, 1, 2, 3, 4, 5, 6, 7, 8} which has all edges. Since there is only one component left, we stop and return MST.

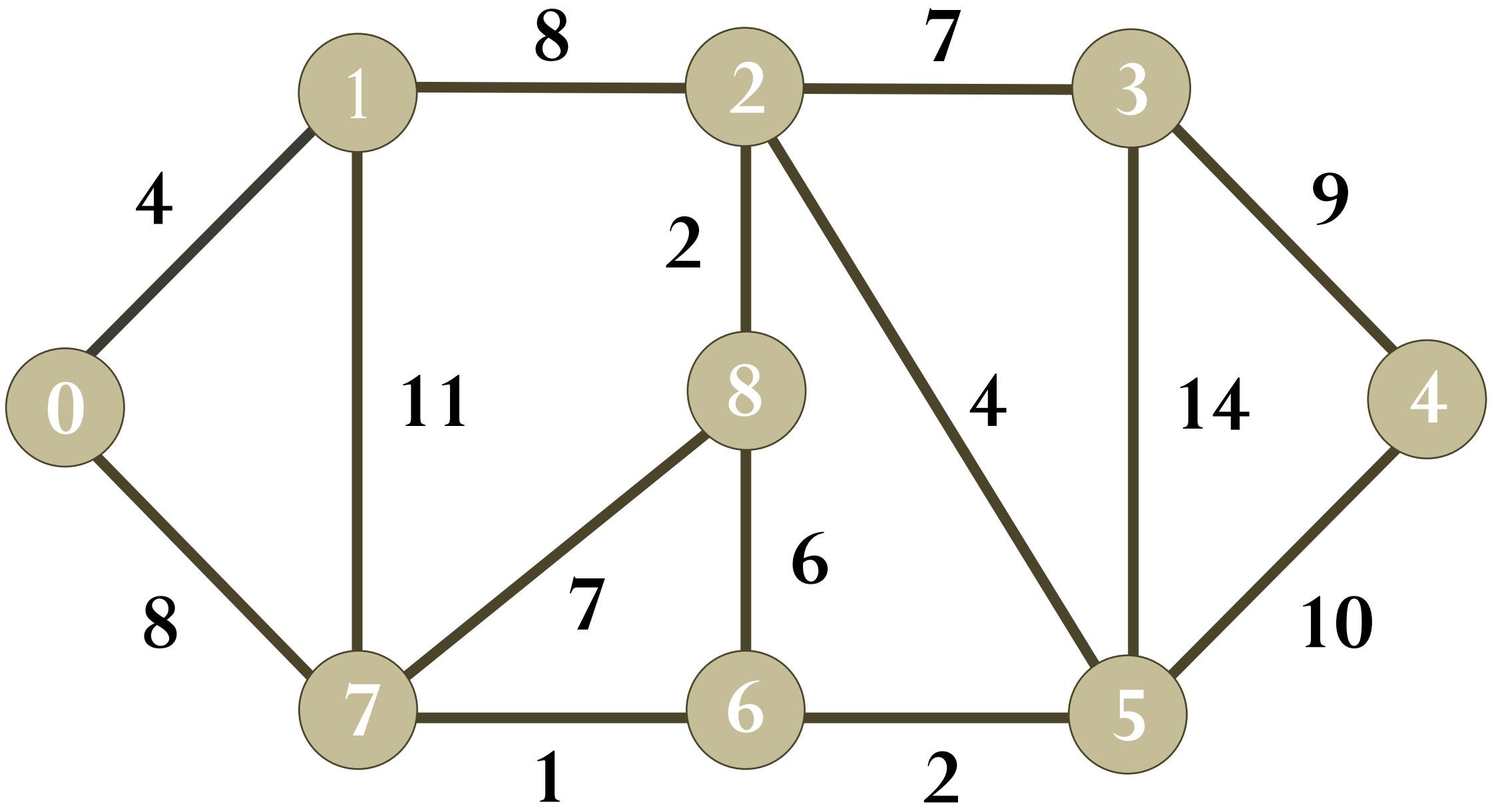


Reverse-Delete Algorithm

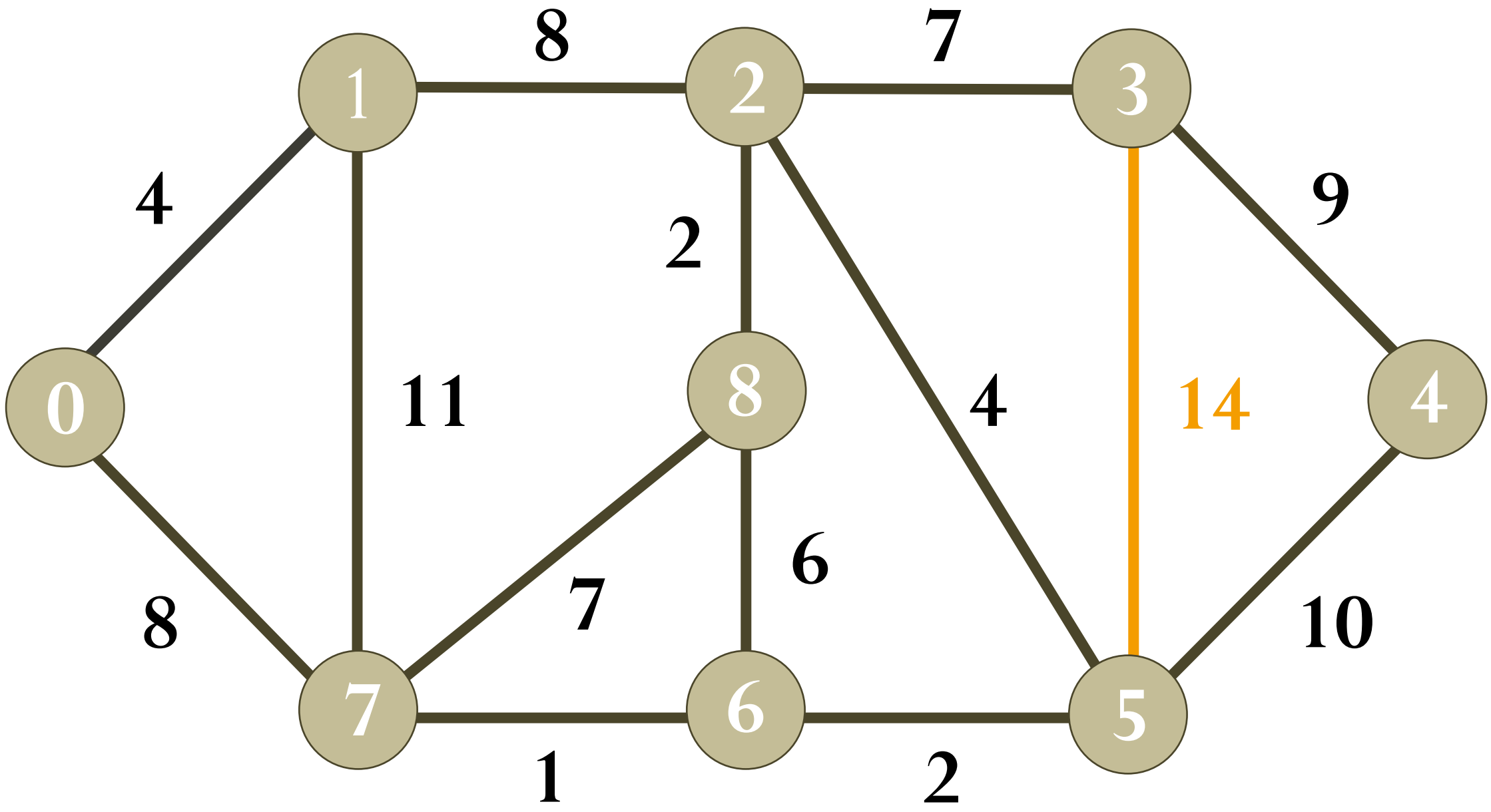
The reverse-delete algorithm is a straightforward algorithm that finds a **minimum** spanning tree for a connected, undirected graph.

It begins with all edges of the graph and repeatedly removes the most expensive edge whose **removal does not disconnect the graph**, until only the edges of the MST remain. The reverse-delete algorithm is a simple method for finding the Minimum Spanning Tree (MST) of a graph by iteratively removing edges while ensuring that the remaining edges still form a spanning tree.

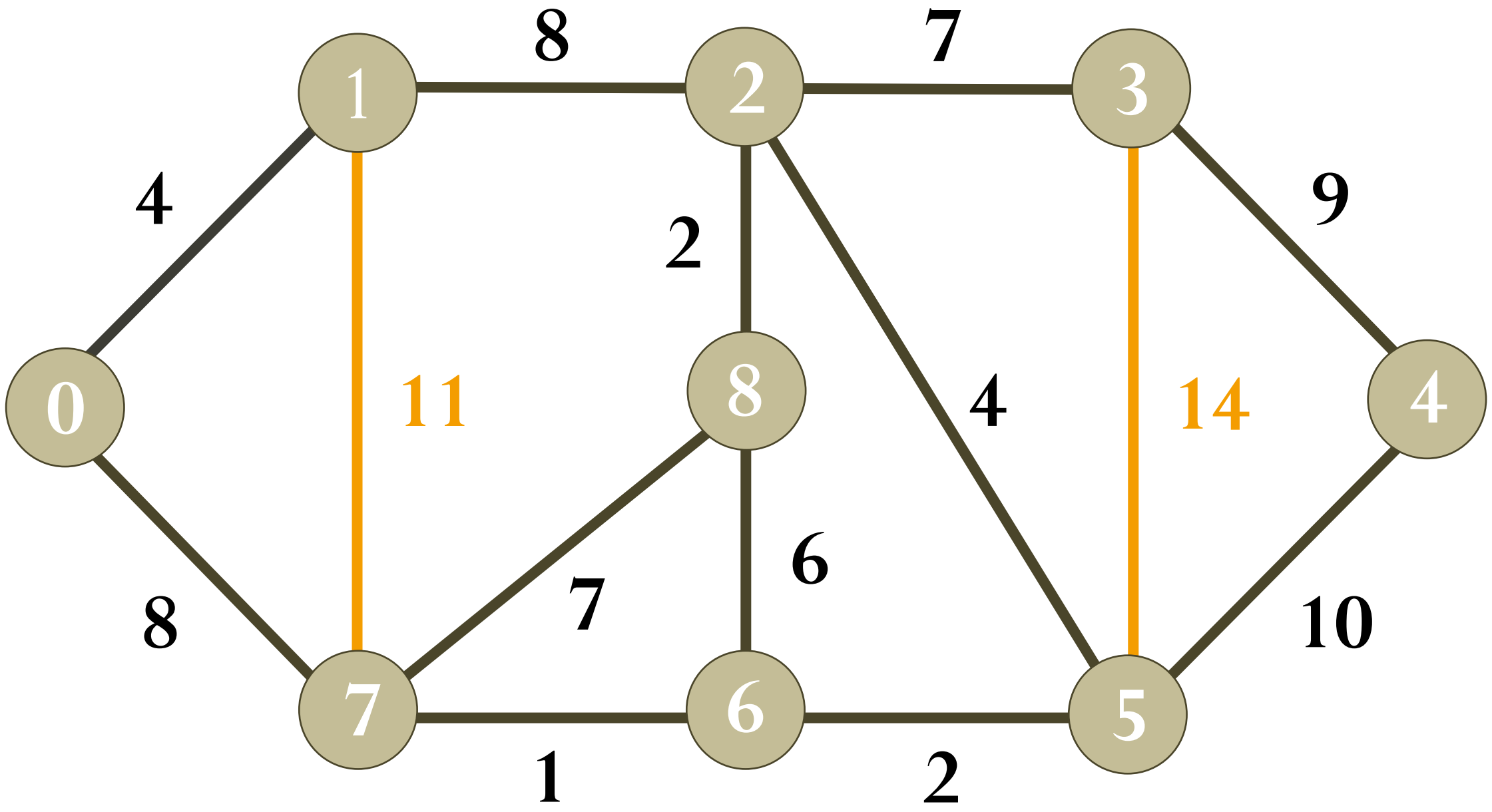
Reverse-Delete Algorithm



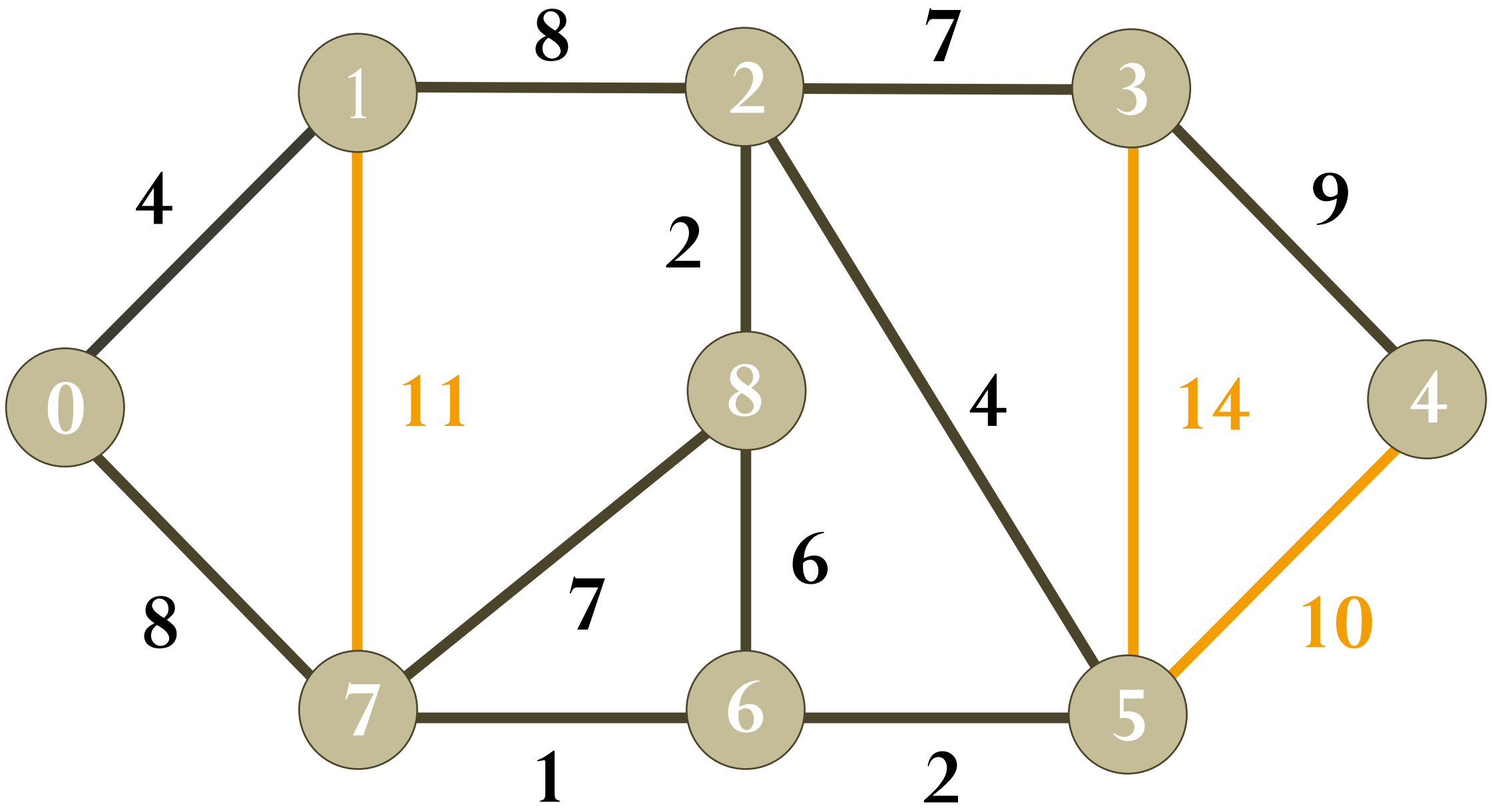
If we delete highest weight edge of weight 14, graph doesn't become disconnected, so we remove it.



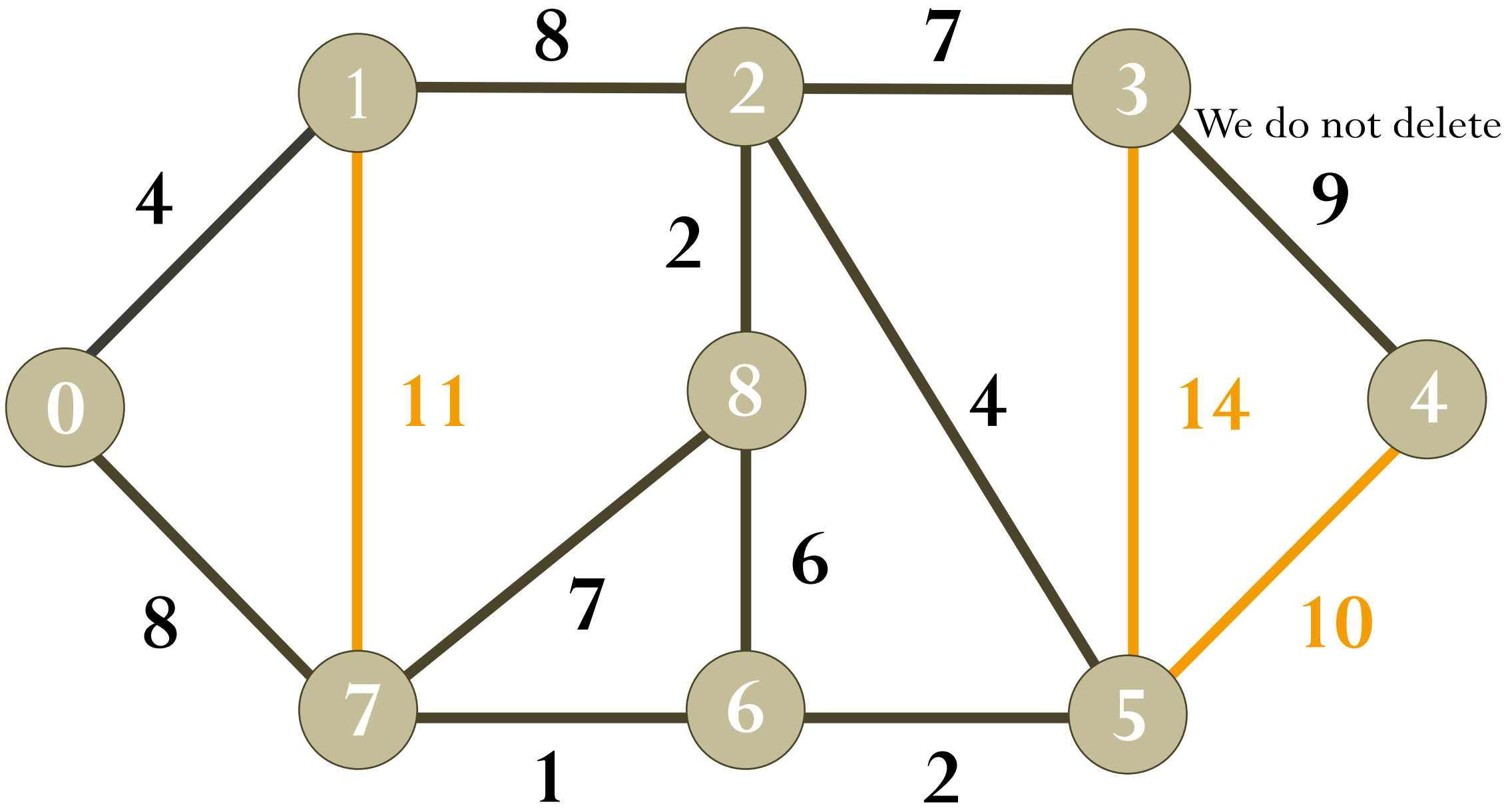
Next we delete 11 as deleting it doesn't disconnect the graph.



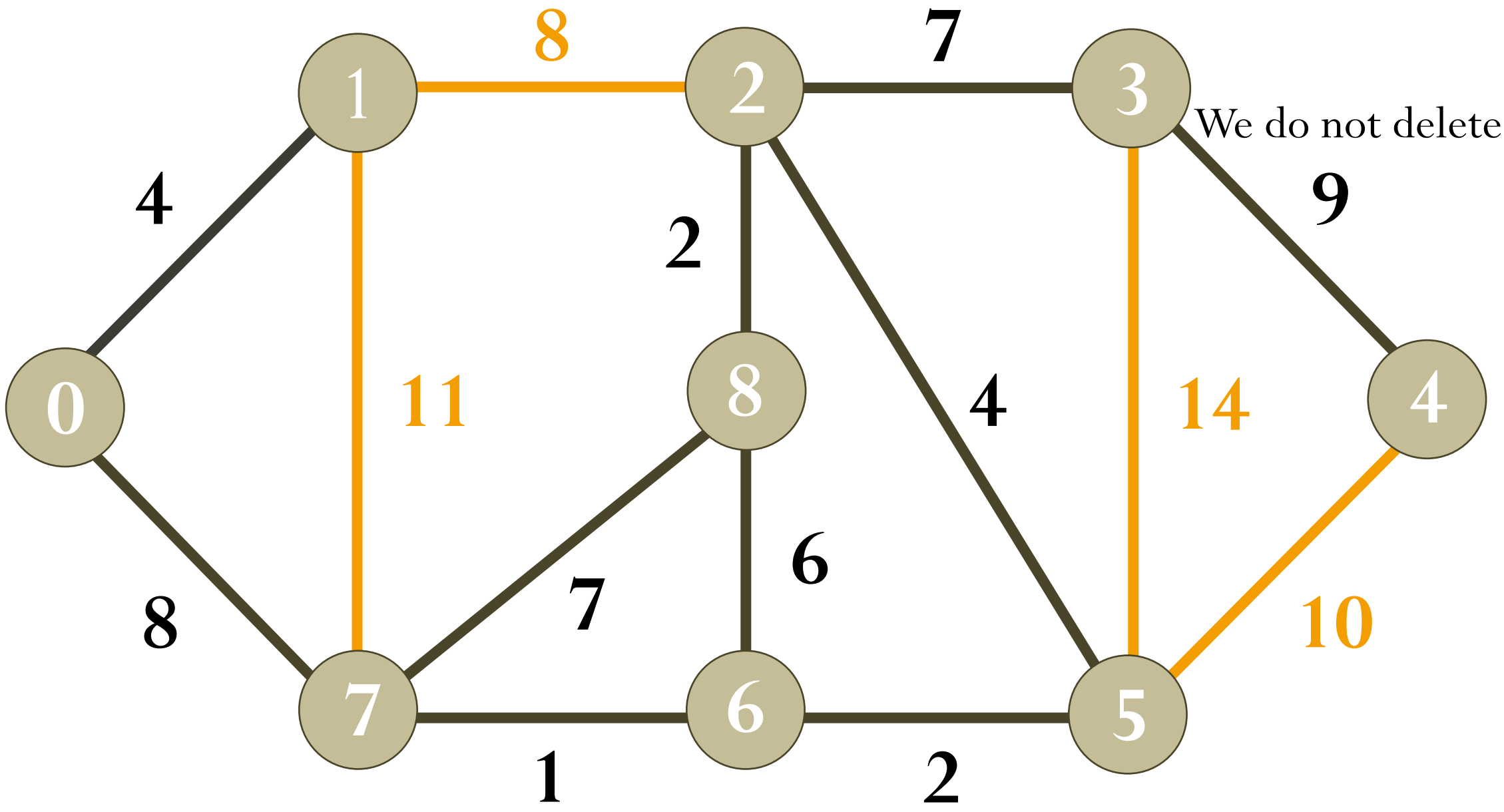
Next we delete 10 as deleting it doesn't disconnect the graph.



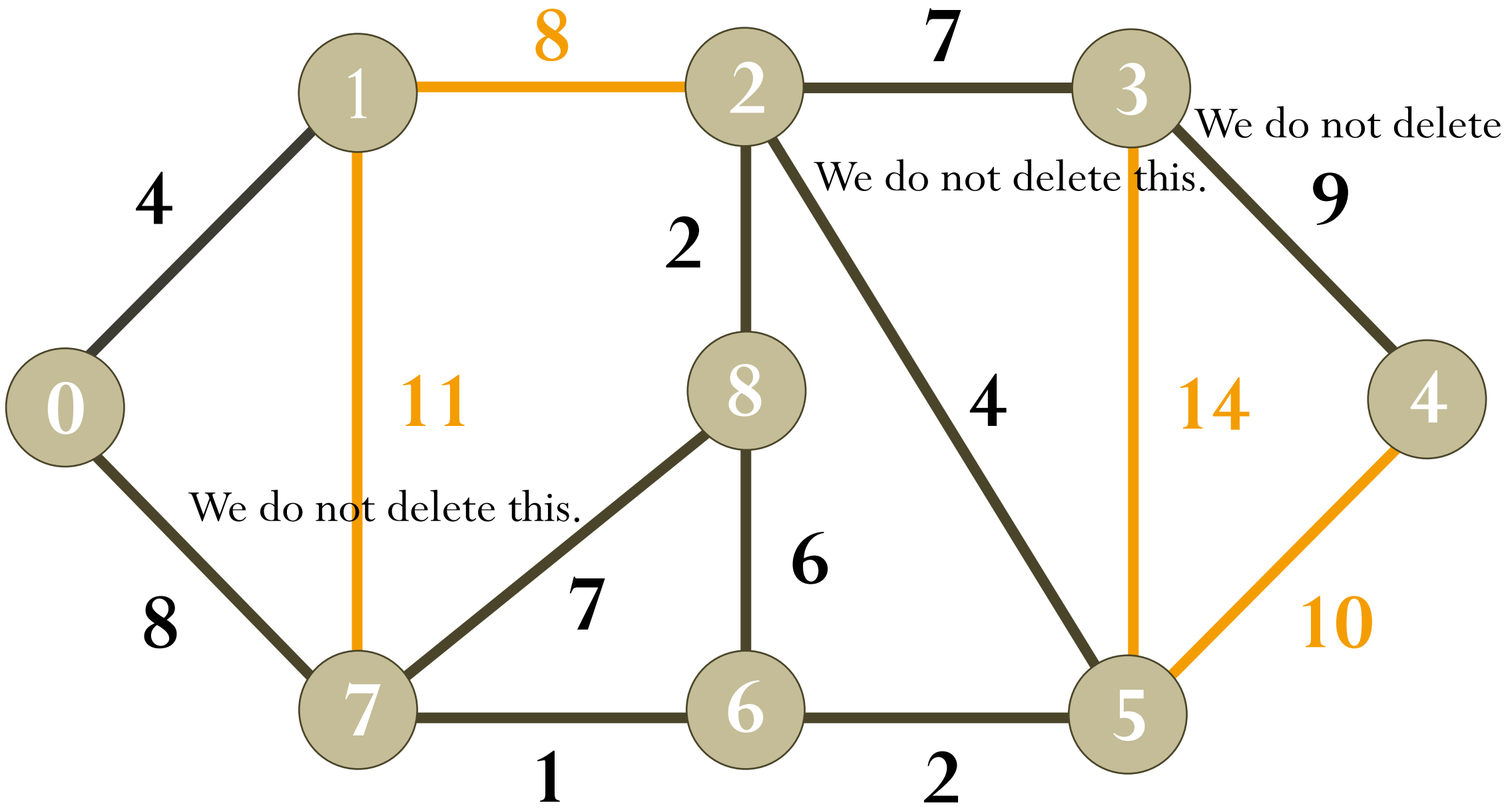
Next is 9. We cannot delete 9 as deleting it causes disconnection.



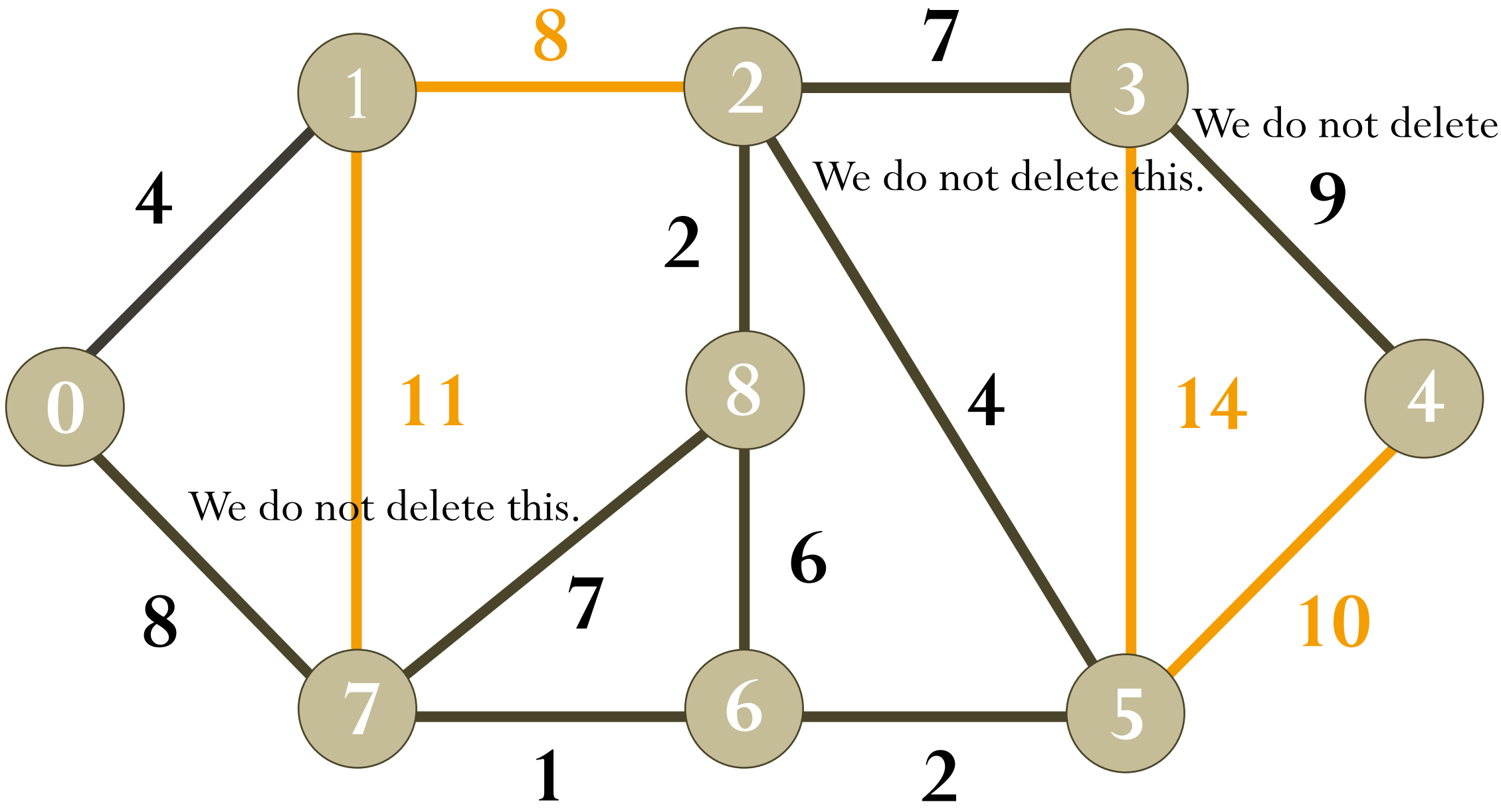
Next we delete 8 as deleting it doesn't disconnect the graph.



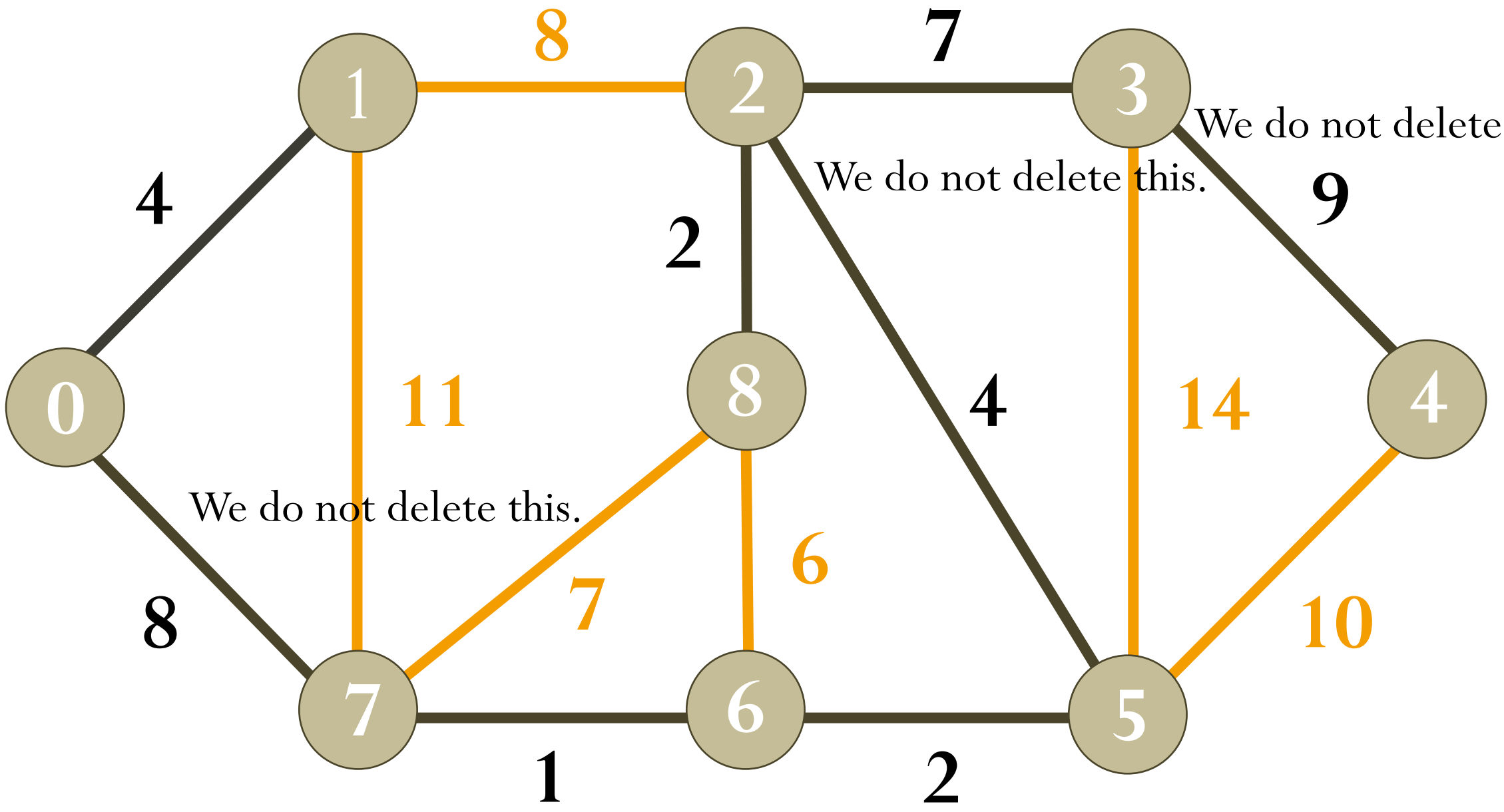
Next is 8. We cannot delete 8 as deleting it causes disconnection.



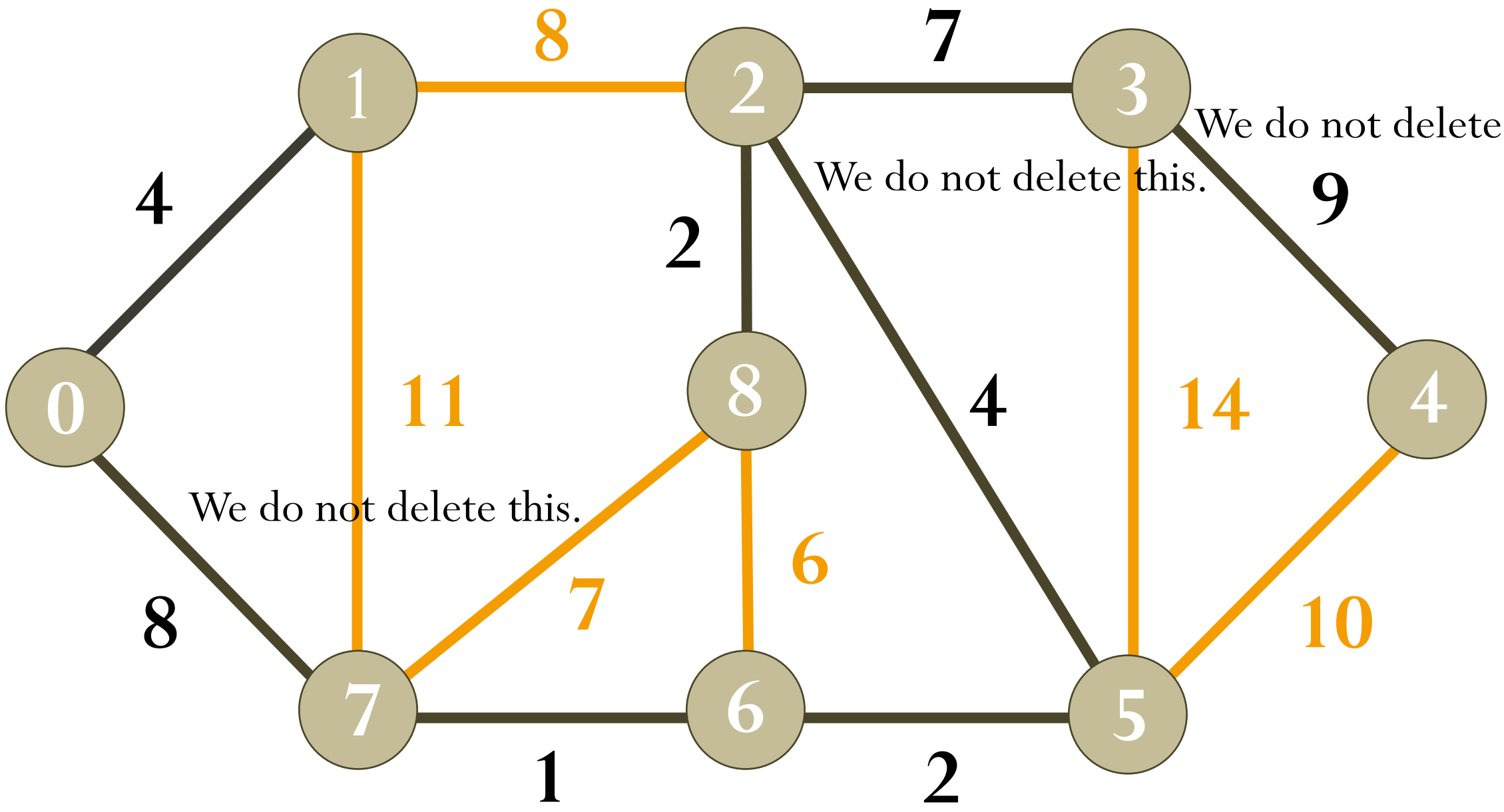
Next is 7. We cannot delete 7 as deleting it causes disconnection.

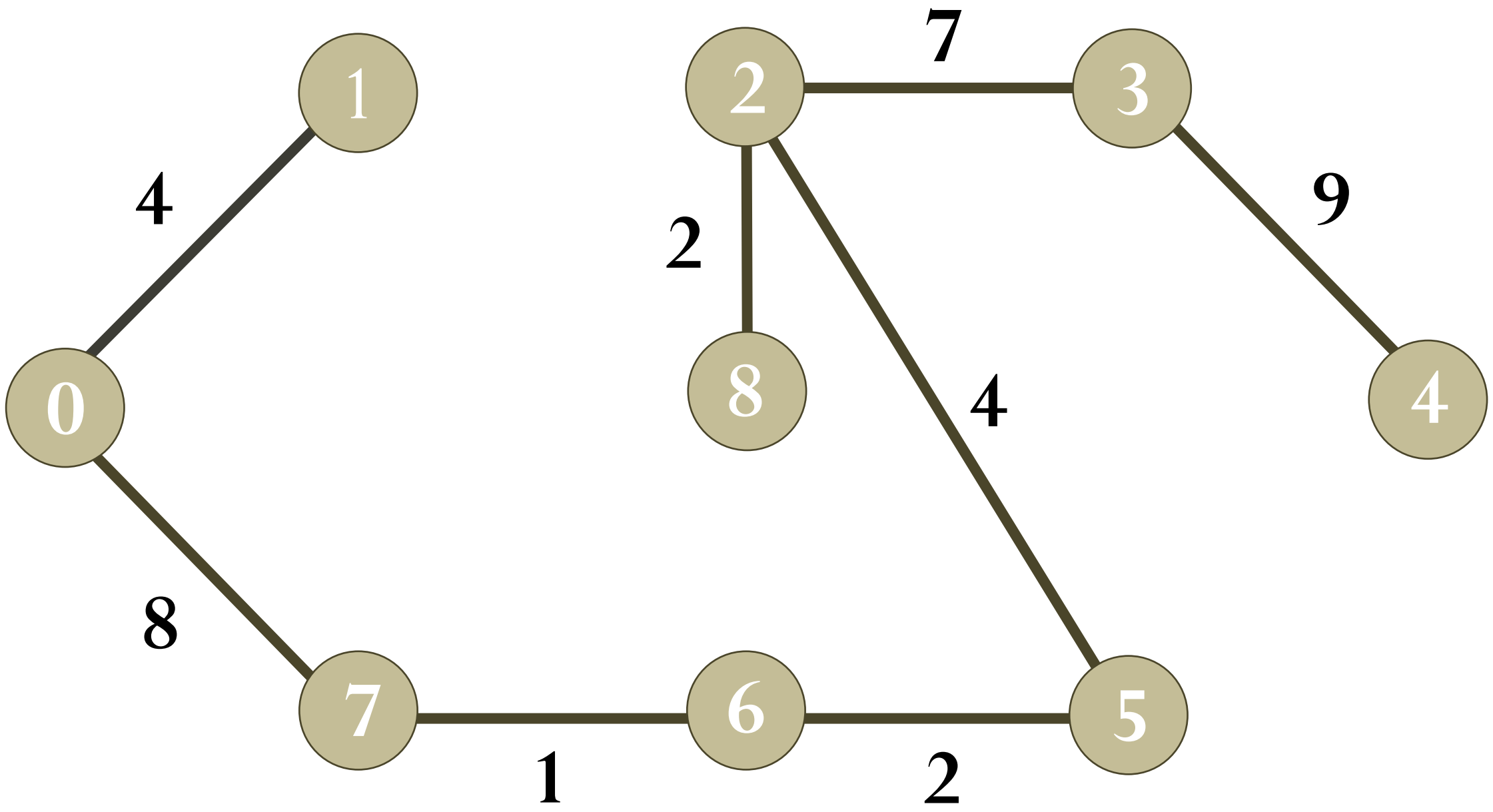


Next is 7. We cannot delete 7 as deleting it causes disconnection.



Next is 6. We cannot delete 6 as deleting it causes disconnection.




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

We continue this way and following edges remain in final MST.

Edges in MST
3-4
0-7
2-3
2-5
0-1
5-6
2-8
6-7

In case of same weight edges, we can pick any edge of the same weight edges.