Advanced Algorithmic Techniques (COMP523)

Greedy Algorithms

Recap and plan

- **Last lecture:**
  - The Greedy approach
  - Interval Scheduling
- **This lecture:**
  - Minimum Spanning Tree
  - Kruskal’s Algorithm
  - Prim’s Algorithm

Application

- We have a set of locations.
- We want to build a communication network, joining all of them.
- We want to do it as cheaply as possible.
    - Every direct connection between two locations has a cost.
    - We want to have everything connected a the minimum cost.

Minimum Spanning Tree

- Consider a *connected* graph G=(V, E), such that for every edge e=(v,w) of E, there is an associated positive cost c_e.
- Goal: Find a subset T of E so that the graph G'=(V, T) is connected and the total cost $\sum_{e \in T} c_e$ is minimised.

Claim: T is a tree

Claim: T is a tree

- By definition, (V, T) is connected.

Claim: T is a tree

- By definition, (V, T) is connected.
- Suppose that it contained a cycle.

Claim: T is a tree

- By definition, (V, T) is connected.
- Suppose that it contained a cycle.
- Let e be an edge on that cycle.

Claim: T is a tree

- By definition, (V, T) is connected.
- Suppose that it contained a cycle.
- Let e be an edge on that cycle.
- Take (V, T-{e}).

Claim: T is a tree

- By definition, (V, T) is connected.
- Suppose that it contained a cycle.
- Let e be an edge on that cycle.
- Take (V, T-{e}).
- This is still connected.

Claim: T is a tree

- By definition, (V, T) is connected.
- Suppose that it contained a cycle.
- Let e be an edge on that cycle.
- Take (V, T-{e}).
- This is still connected.
  - All paths that used e can be rerouted through the other direction.

Claim: T is a tree

- By definition, (V, T) is connected.
- Suppose that it contained a cycle.
- Let e be an edge on that cycle.
- Take (V, T-{e}).
- This is still connected.
  - All paths that used e can be rerouted through the other direction.
- (V, T-{e}) is a valid solution, and it is cheaper. **Contradiction!**

Minimum Spanning Tree

- Consider a *connected* graph $G=(V, E)$, such that for every edge $e=(v,w)$ of E, there is an associated positive cost c_e.
- Goal: Find a subset T of E so that the graph $G'=(V, T)$ is connected and the total cost $\sum_{e \in T} c_e$ is minimised.

Minimum Spanning Tree

T is a spanning tree and the problem is called the Minimum Spanning Tree problem.

- Consider a *connected* graph G=(V, E), such that for every edge e=(v,w) of E, there is an associated positive cost c_e.
- Goal: Find a subset T of E so that the graph G'=(V, T) is connected and the total cost $\sum_{e \in T} c_e$ is minimised.

Greedy Approach #1

Greedy Approach #1

- Start with an empty set of edges T.

Greedy Approach #1

- Start with an empty set of edges T.
- Add one edge to T.

Greedy Approach #1

- Start with an empty set of edges T.
- Add one edge to T.
  - Which one?

Greedy Approach #1

- Start with an empty set of edges T.
- Add one edge to T.
  - Which one?
  - The one with the minimum cost c_e.

Greedy Approach #1

- Start with an empty set of edges T.
- Add one edge to T.
  - Which one?
  - The one with the minimum cost c_e.
- We continue like this.

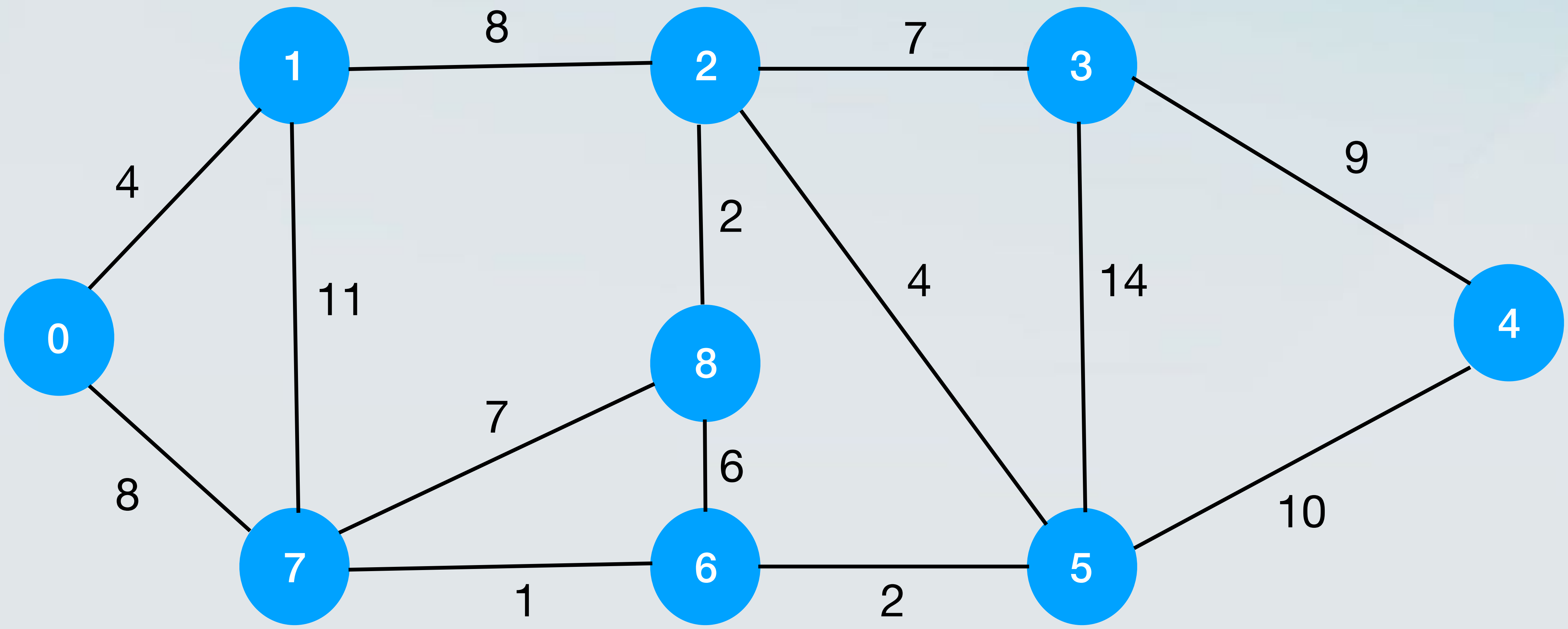
Greedy Approach #1

- Start with an empty set of edges T.
- Add one edge to T.
  - Which one?
  - The one with the minimum cost c_e.
- We continue like this.
- Do we always add the new edge e to T?

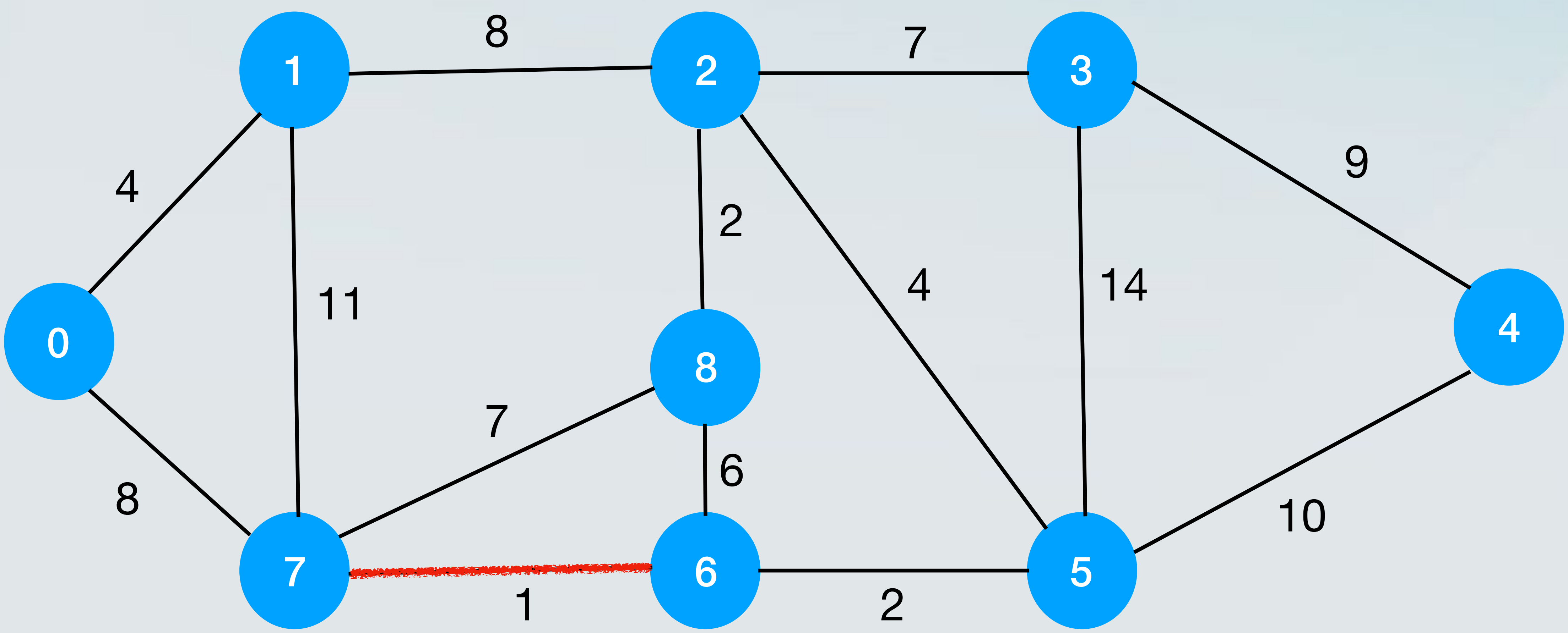
Greedy Approach #1

- Start with an empty set of edges T.
- Add one edge to T.
  - Which one?
  - The one with the minimum cost c_e.
- We continue like this.
- Do we always add the new edge e to T?
  - Only if we don't introduce any cycles.

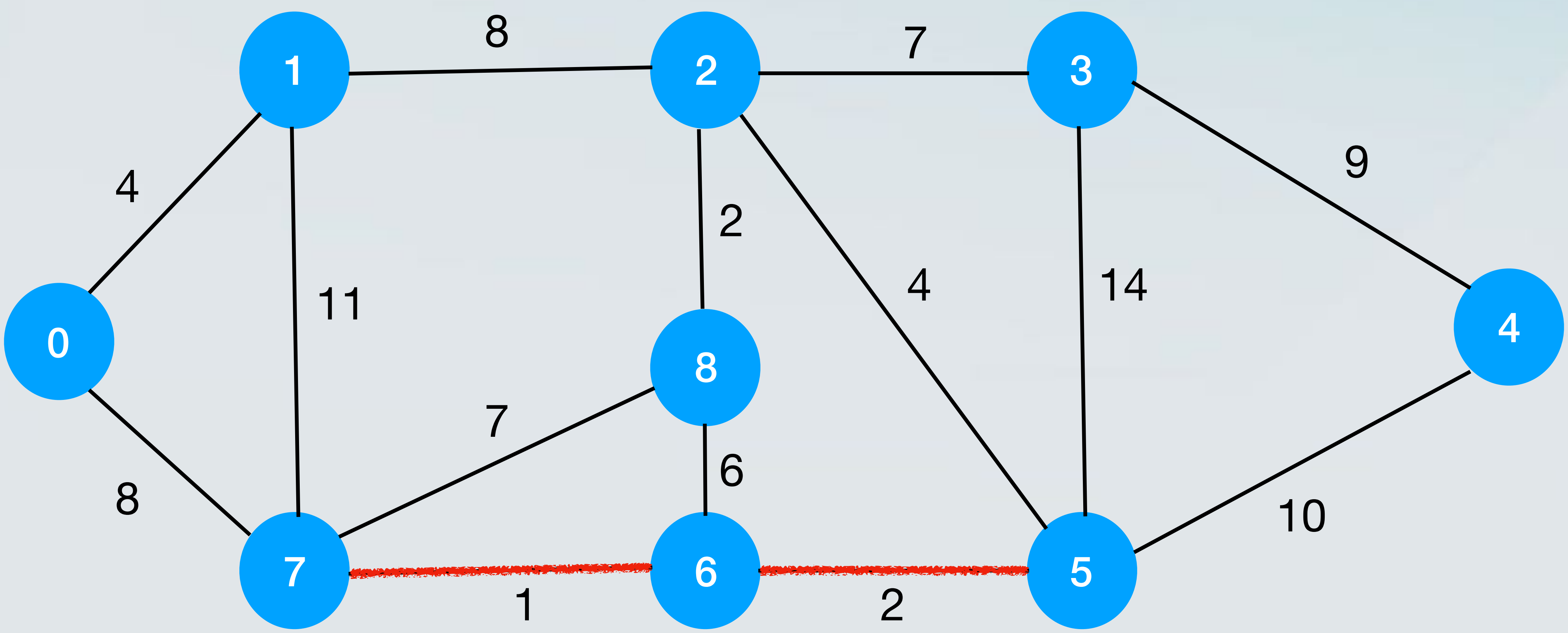
Example



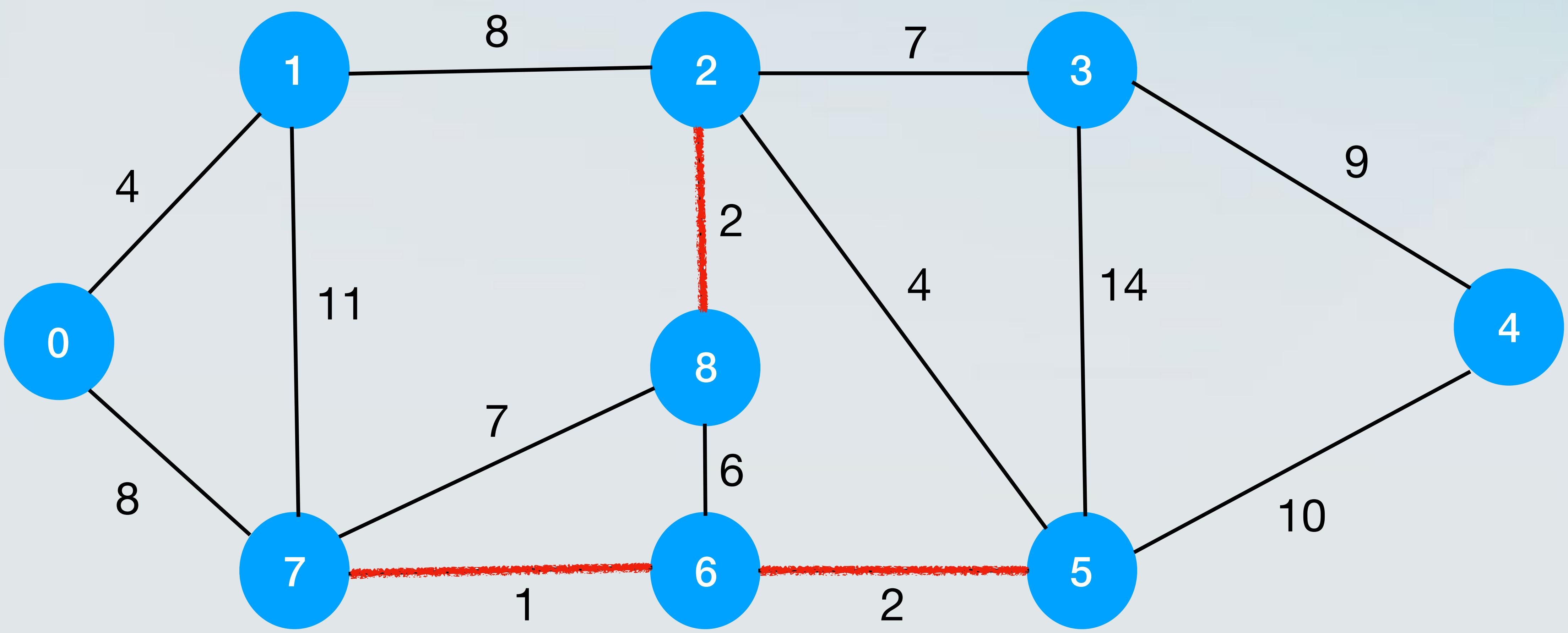
Example



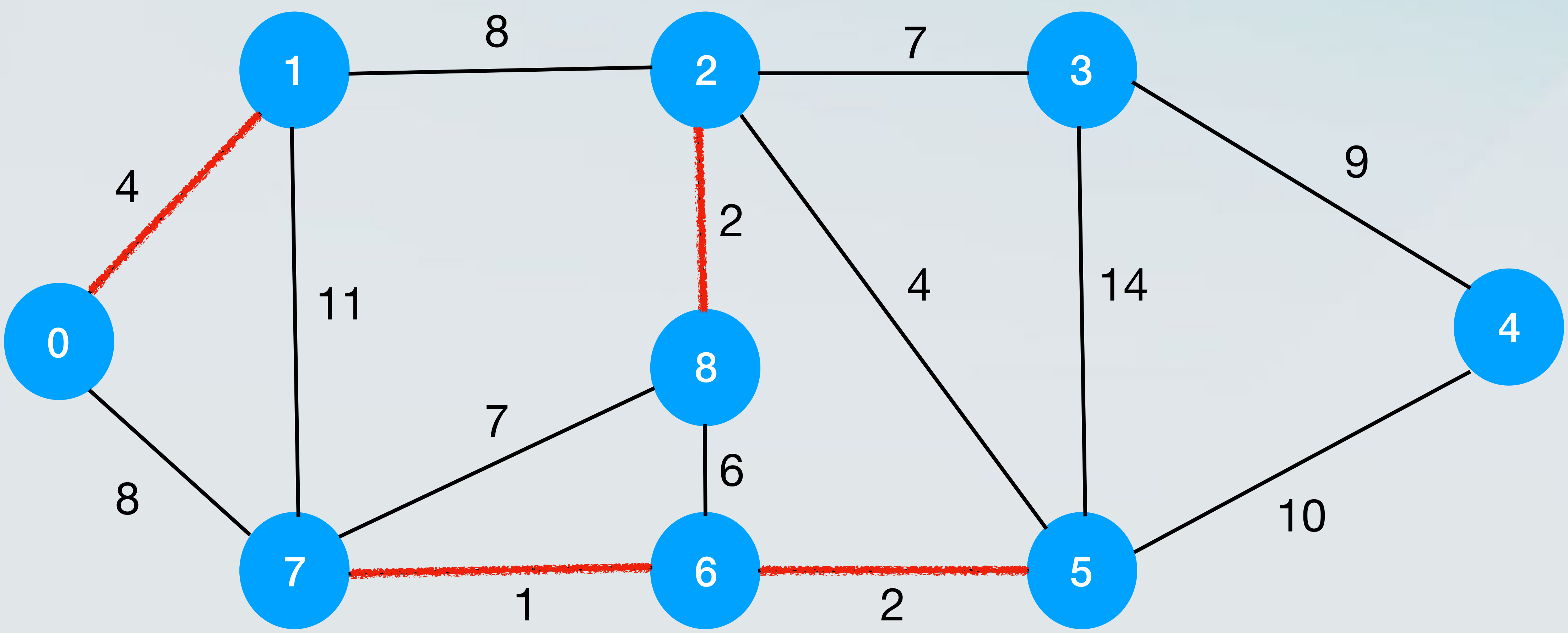
Example



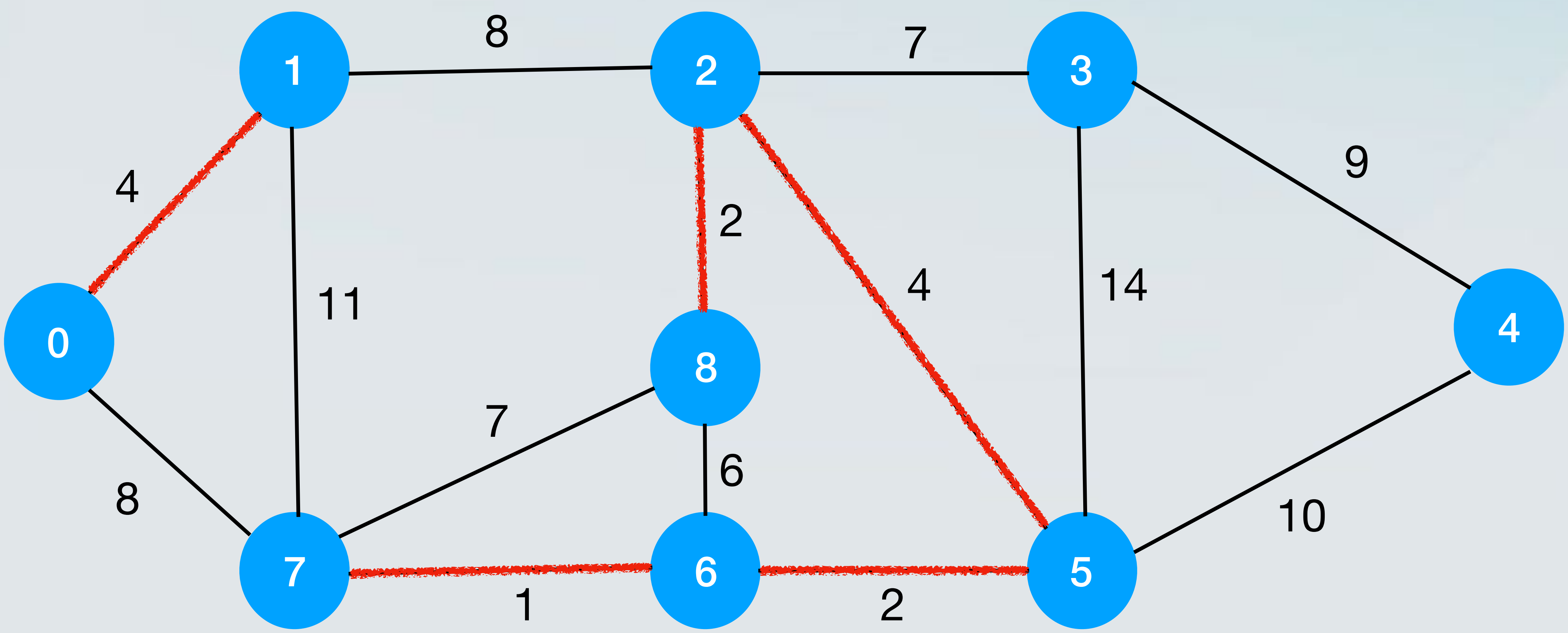
Example



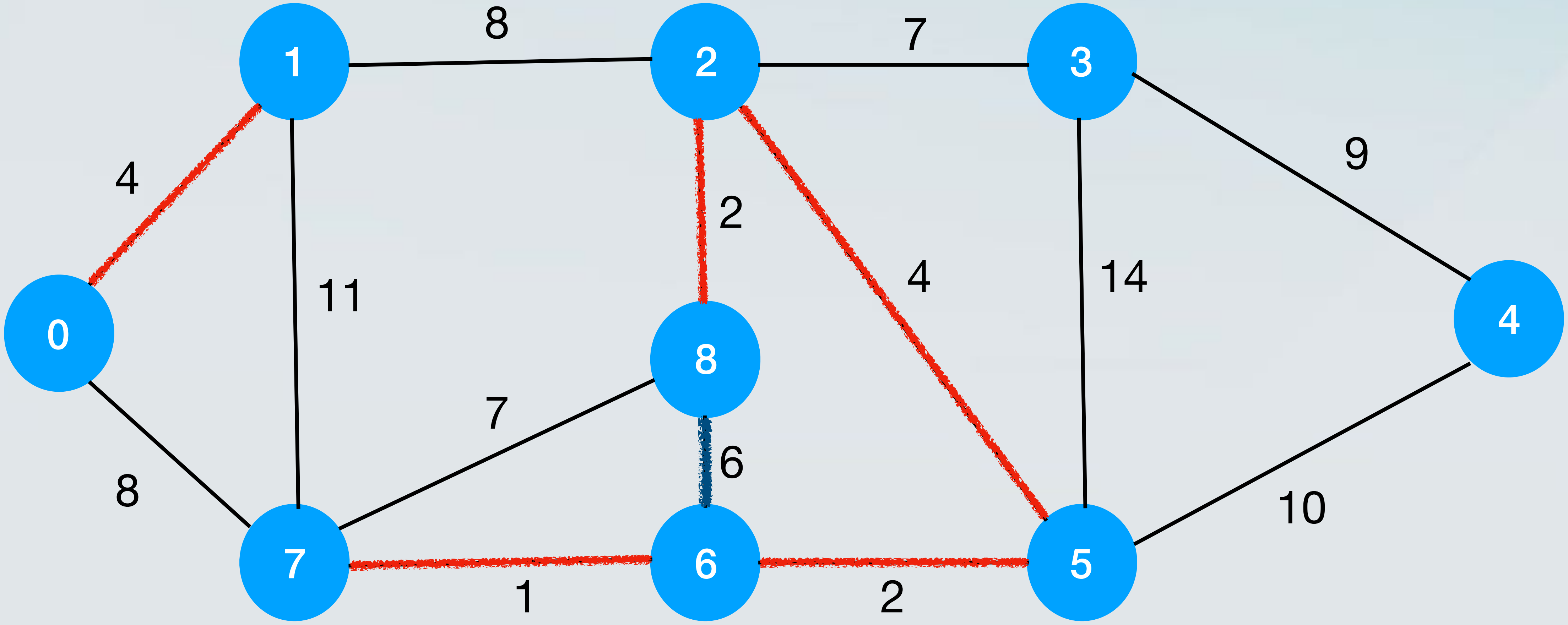
Example



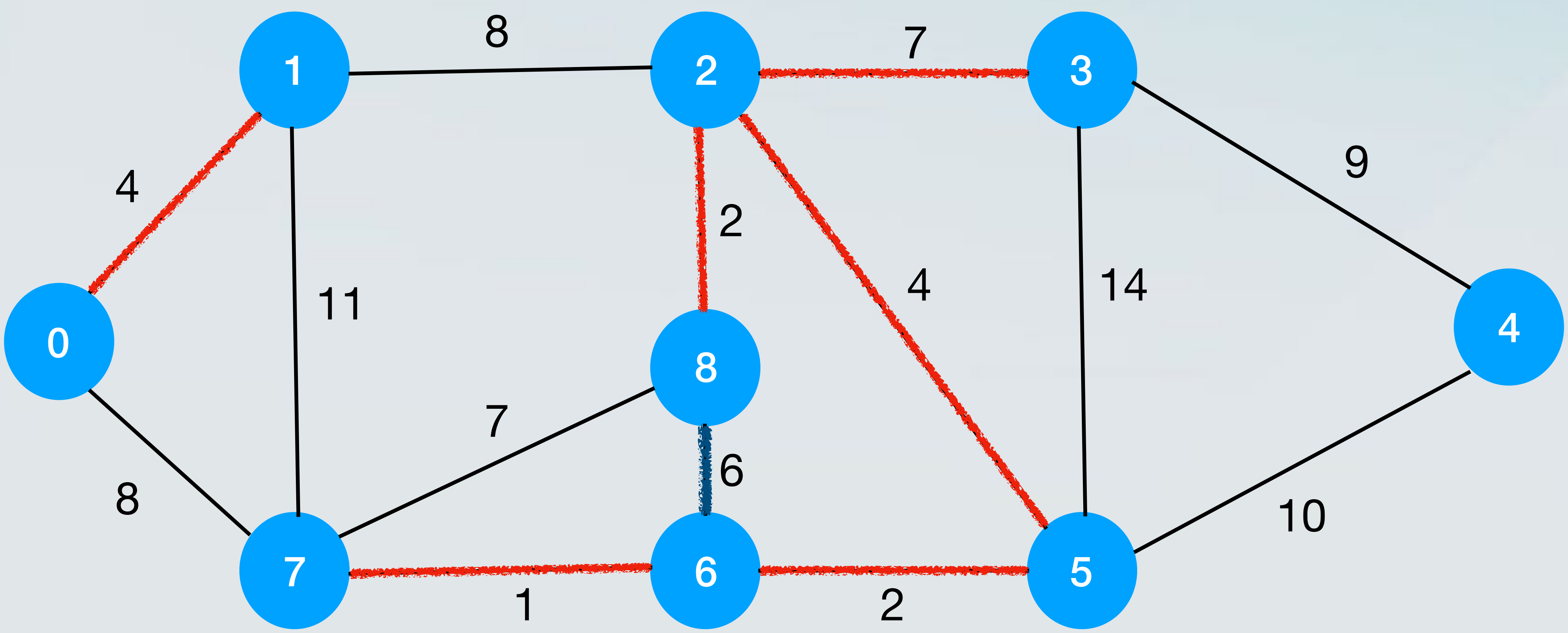
Example



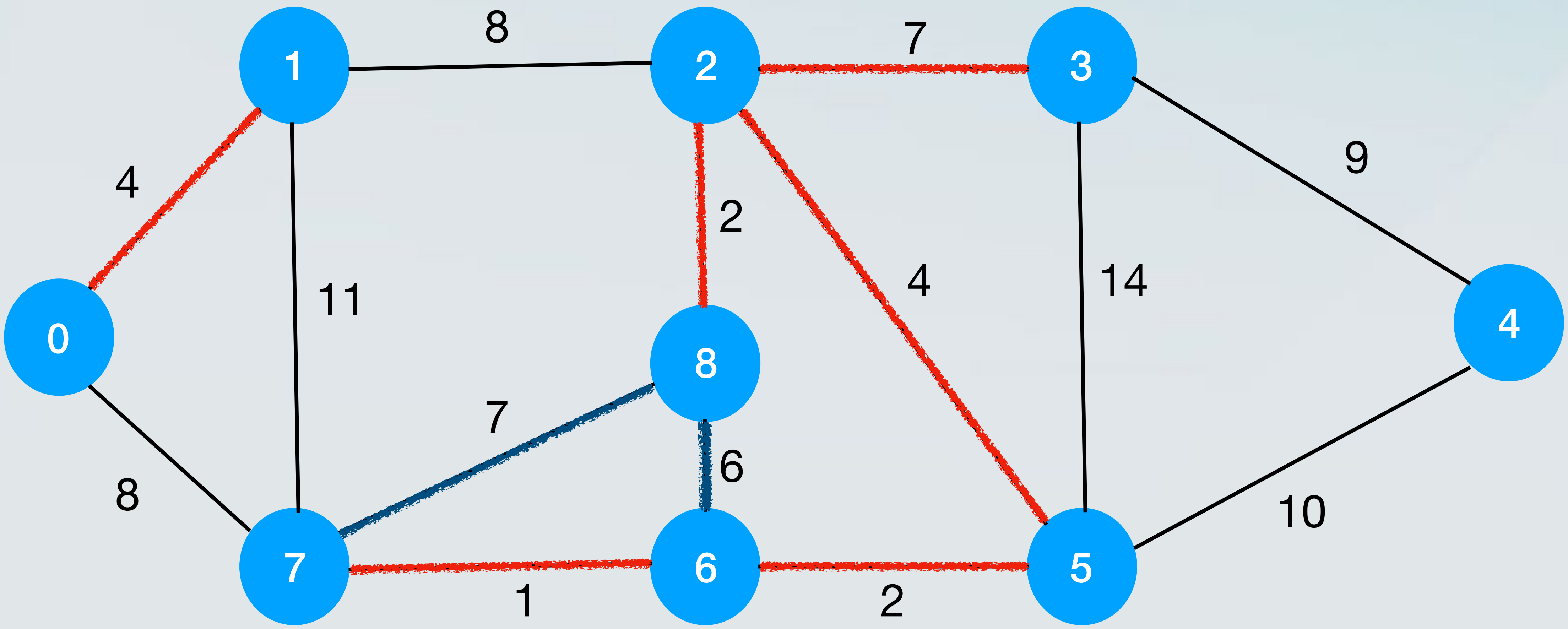
Example



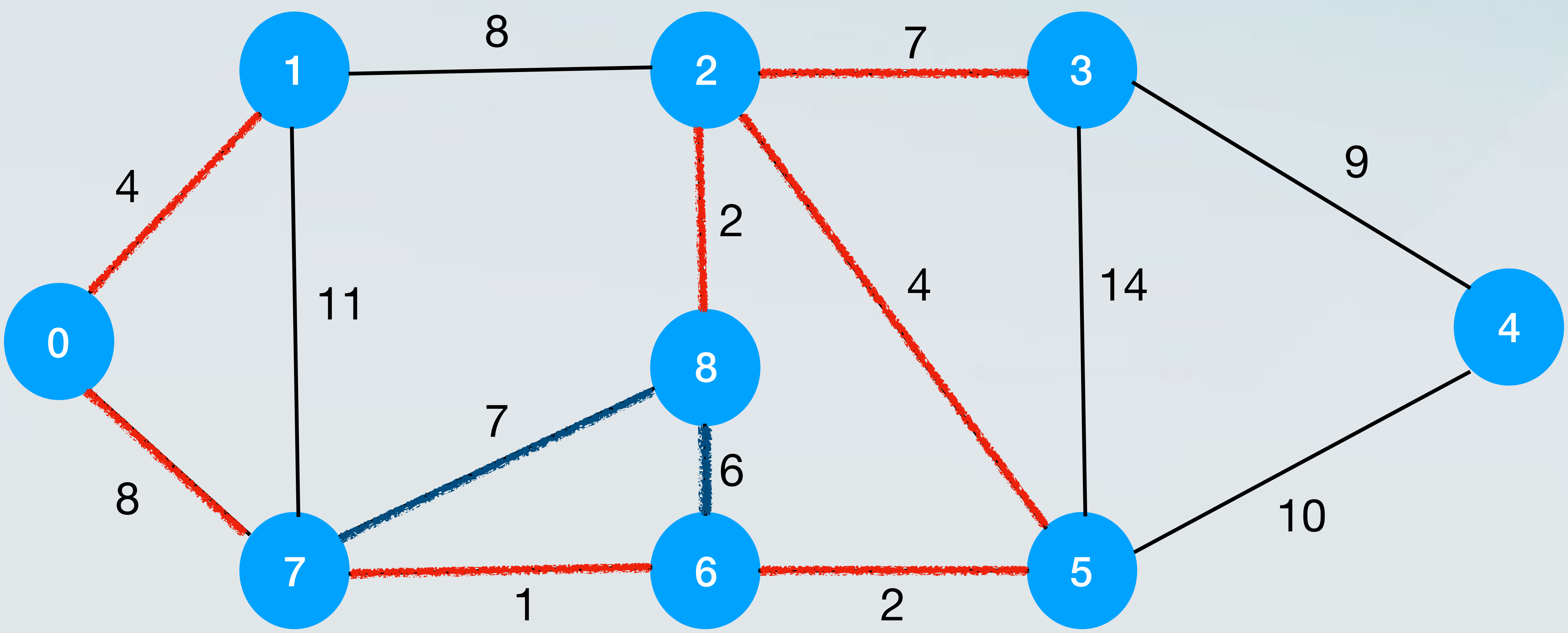
Example



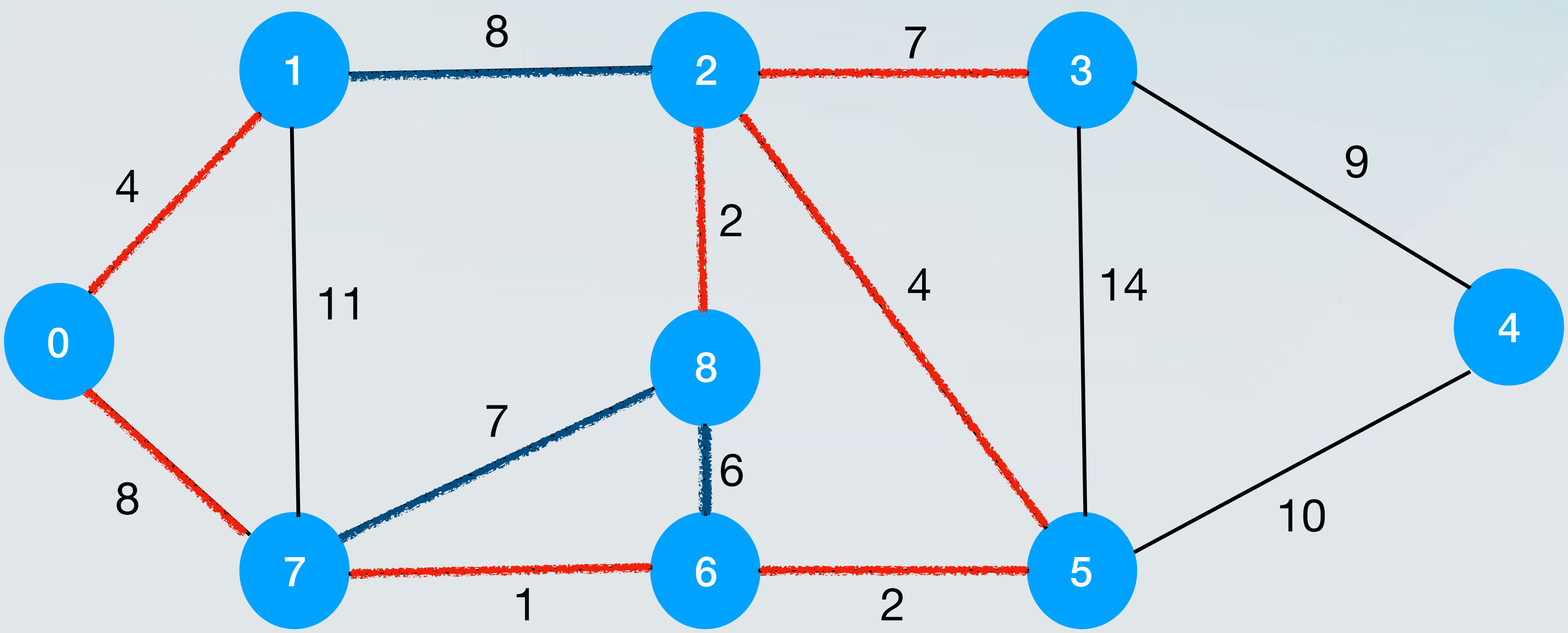
Example



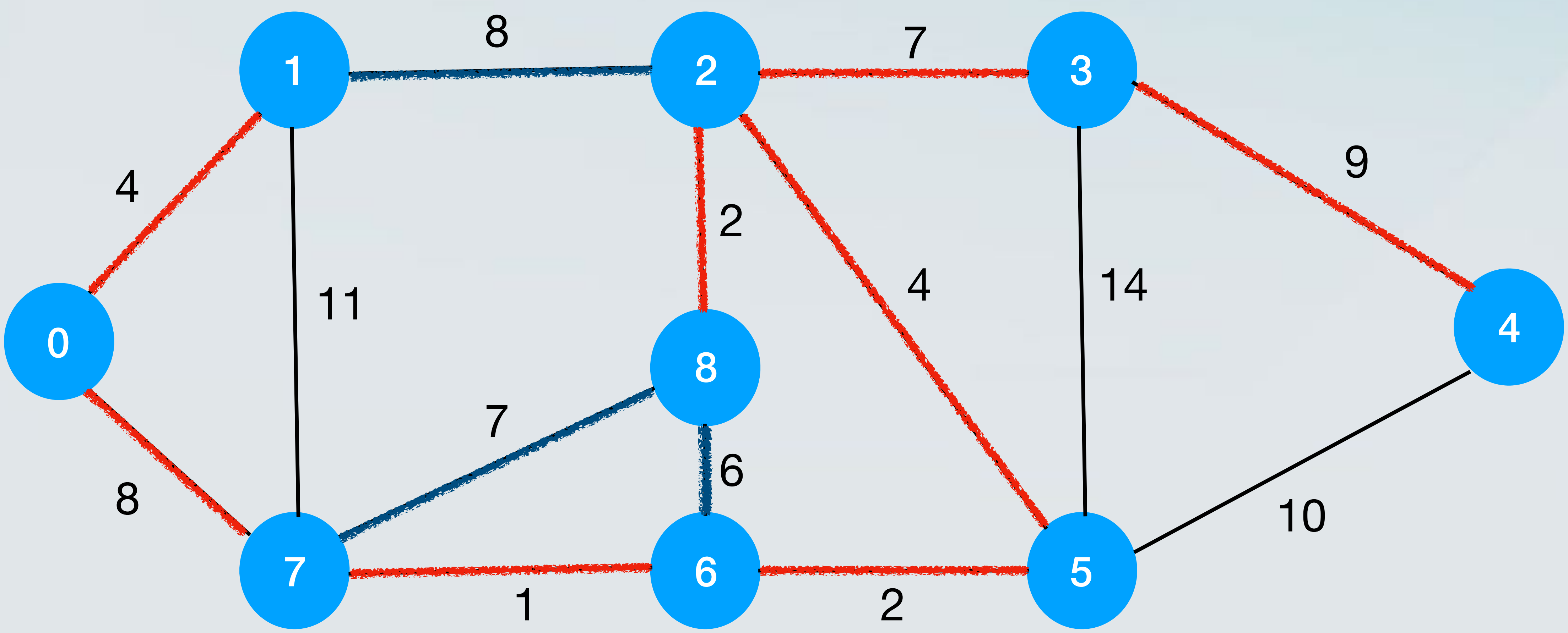
Example



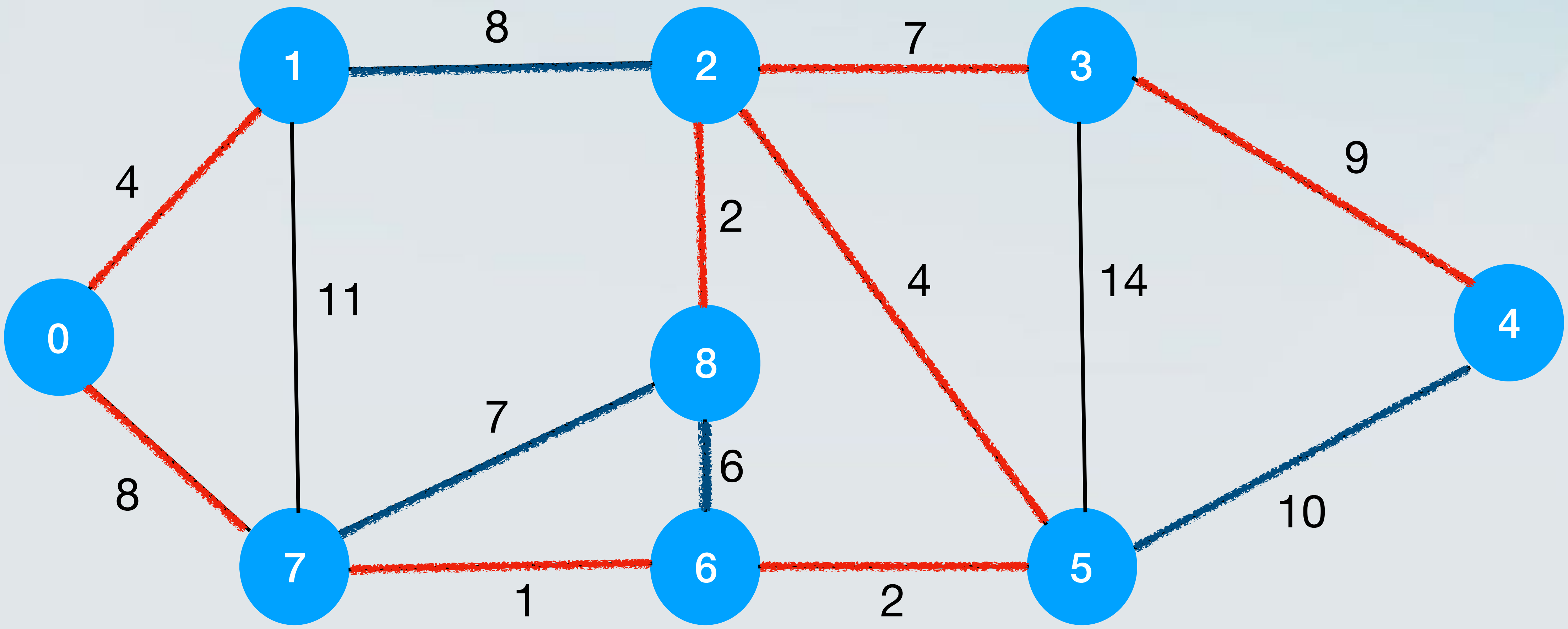
Example



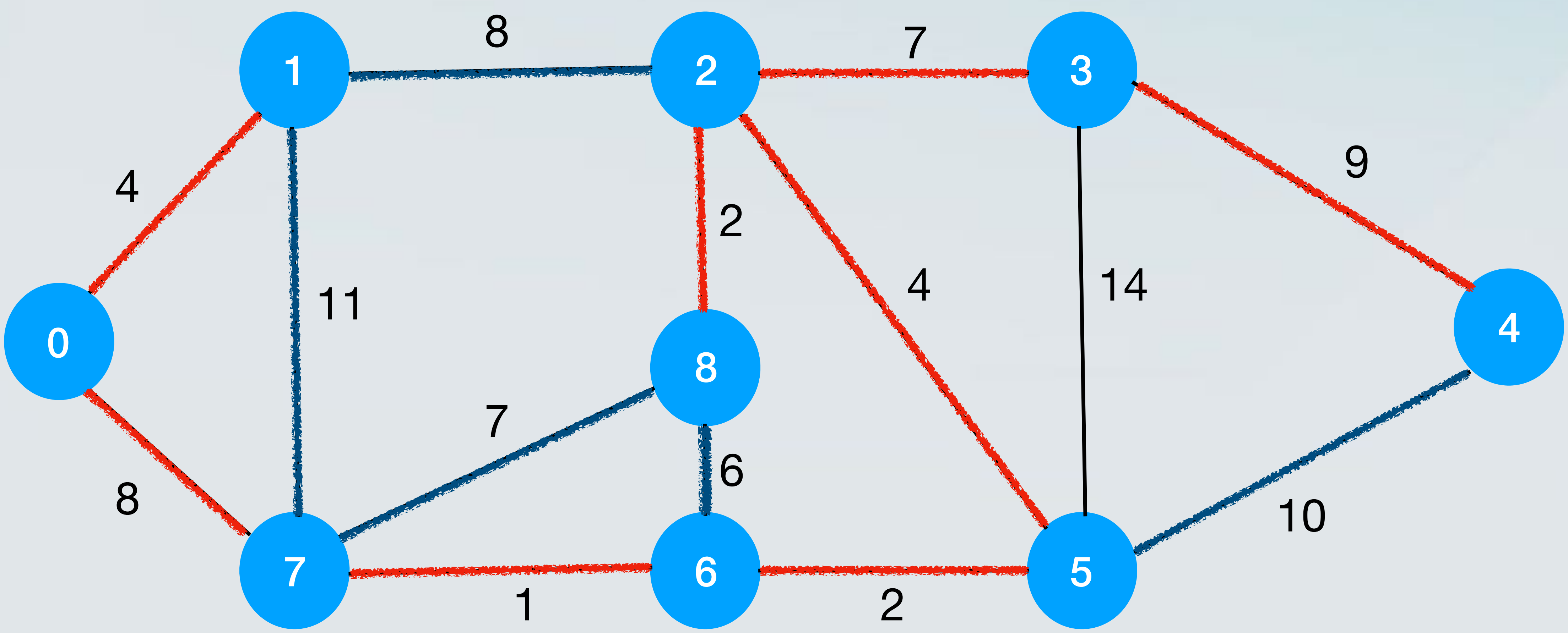
Example



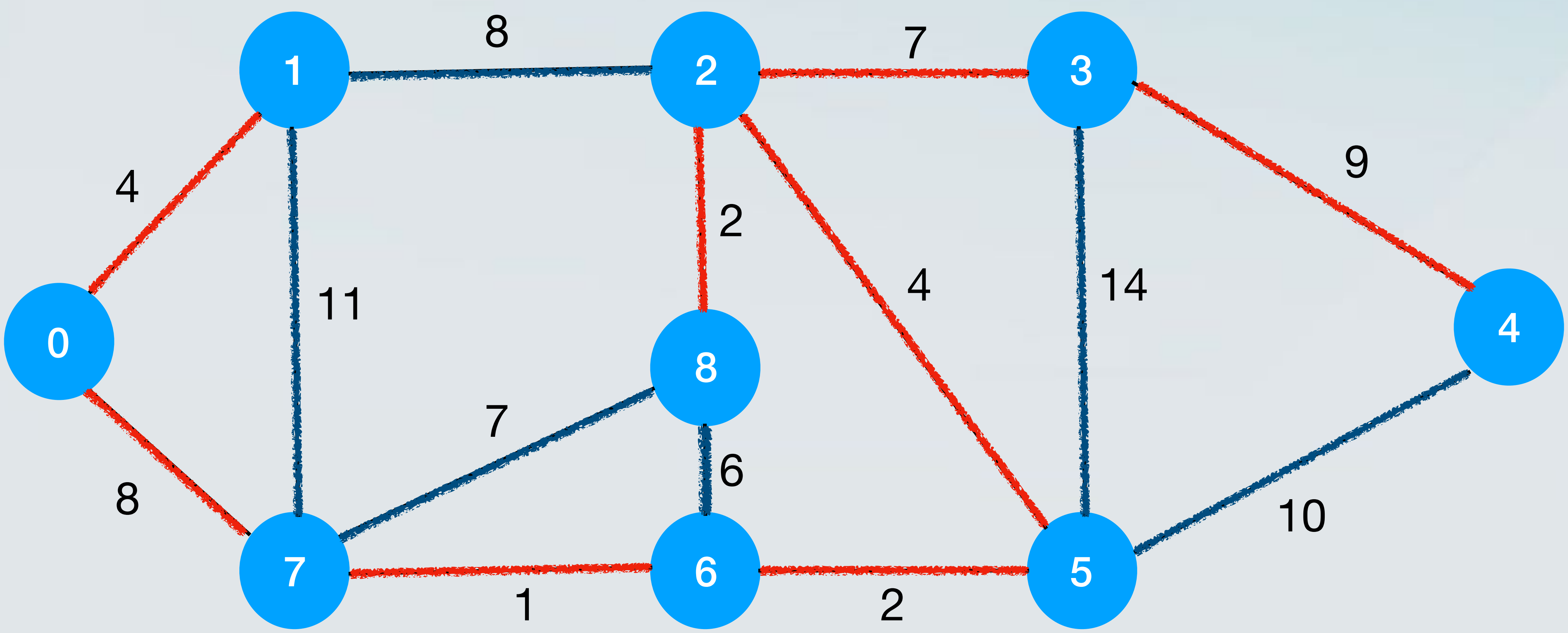
Example



Example



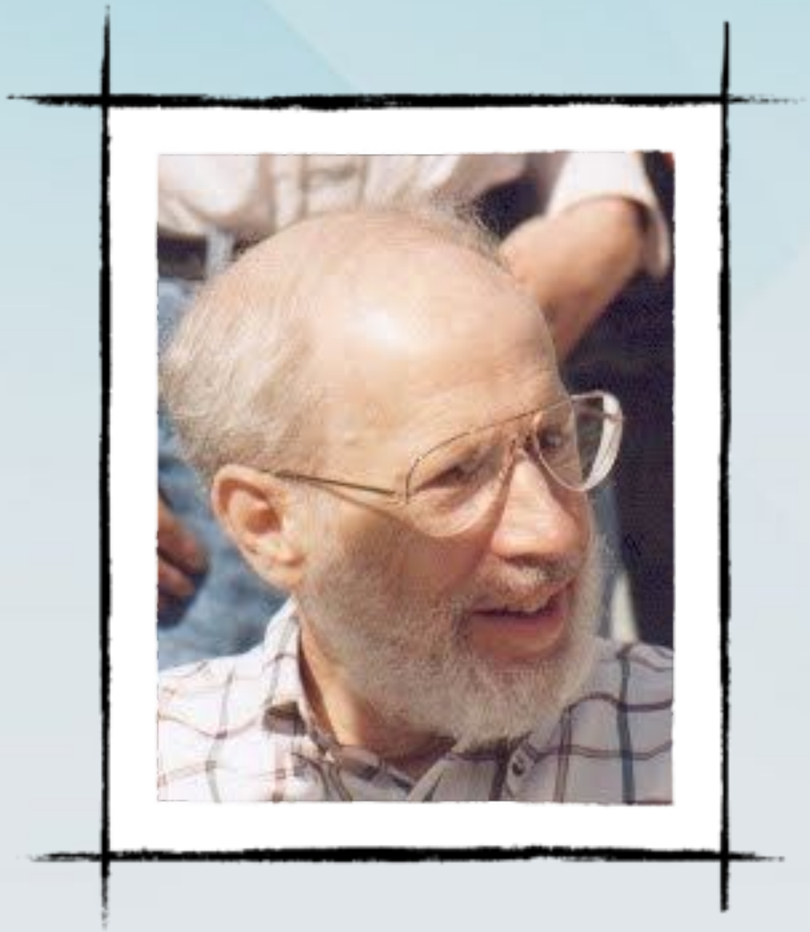
Example



Greedy Approach #1

- Start with an empty set of edges T.
- Add one edge to T.
    - Which one?
    - The one with the minimum cost c_e.
- We continue like this.
- Do we always add the new edge e to T?
    - Only if we don't introduce any cycles.

Kruskal's Algorithm



- Start with an empty set of edges T.
- Add one edge to T.
  - Which one?
  - The one with the minimum cost c_e.
- We continue like this.
- Do we always add the new edge e to T?
  - Only if we don't introduce any cycles.

Greedy Approach #2

Greedy Approach #2

- Start with an empty set of edges T.

Greedy Approach #2

- Start with an empty set of edges T.
- Start with a node s.

Greedy Approach #2

- Start with an empty set of edges T.
- Start with a node s.
  - Add an edge e=(s,w) to T.

Greedy Approach #2

- Start with an empty set of edges T.
- Start with a node s.
    - Add an edge e=(s,w) to T.
    - Which one?

Greedy Approach #2

- Start with an empty set of edges T.
- Start with a node s.
  - Add an edge e=(s,w) to T.
  - Which one?
  - The one with the minimum cost c_e.

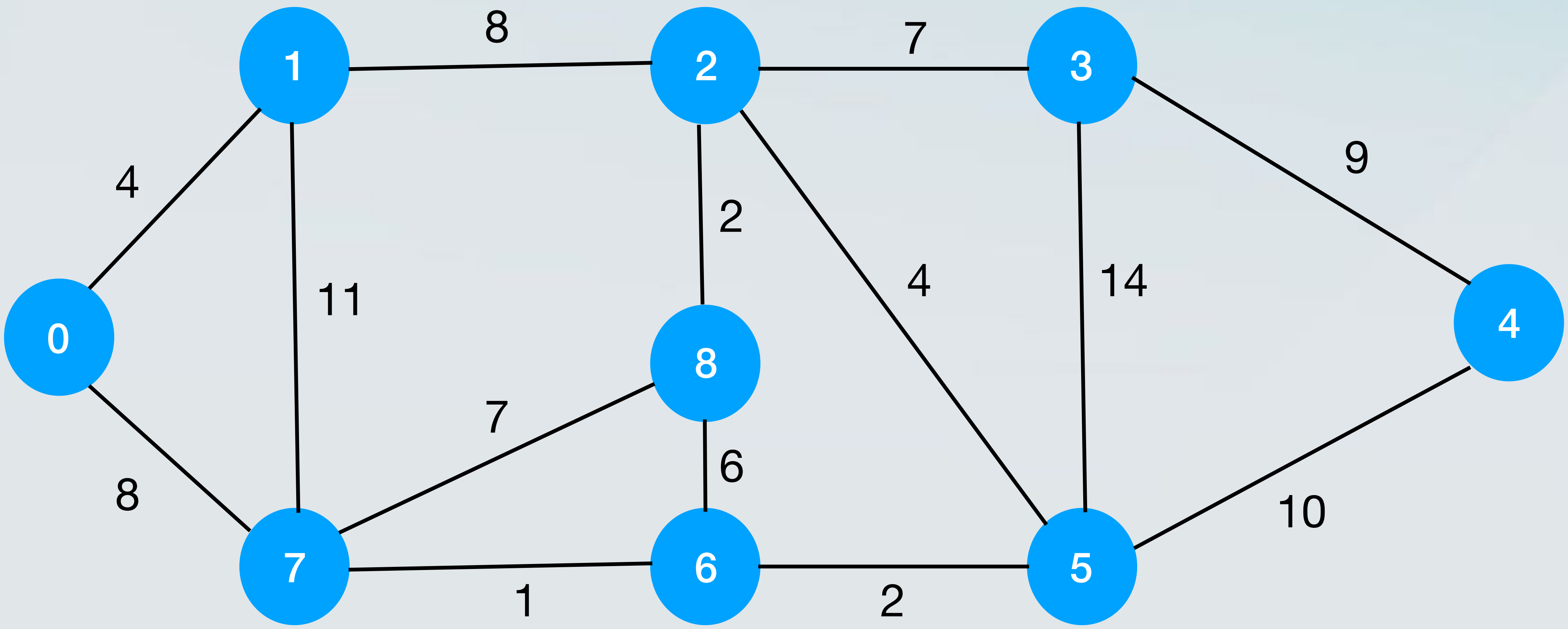
Greedy Approach #2

- Start with an empty set of edges T.
- Start with a node s.
    - Add an edge e=(s,w) to T.
    - Which one?
    - The one with the minimum cost c_e.
- We continue like this.

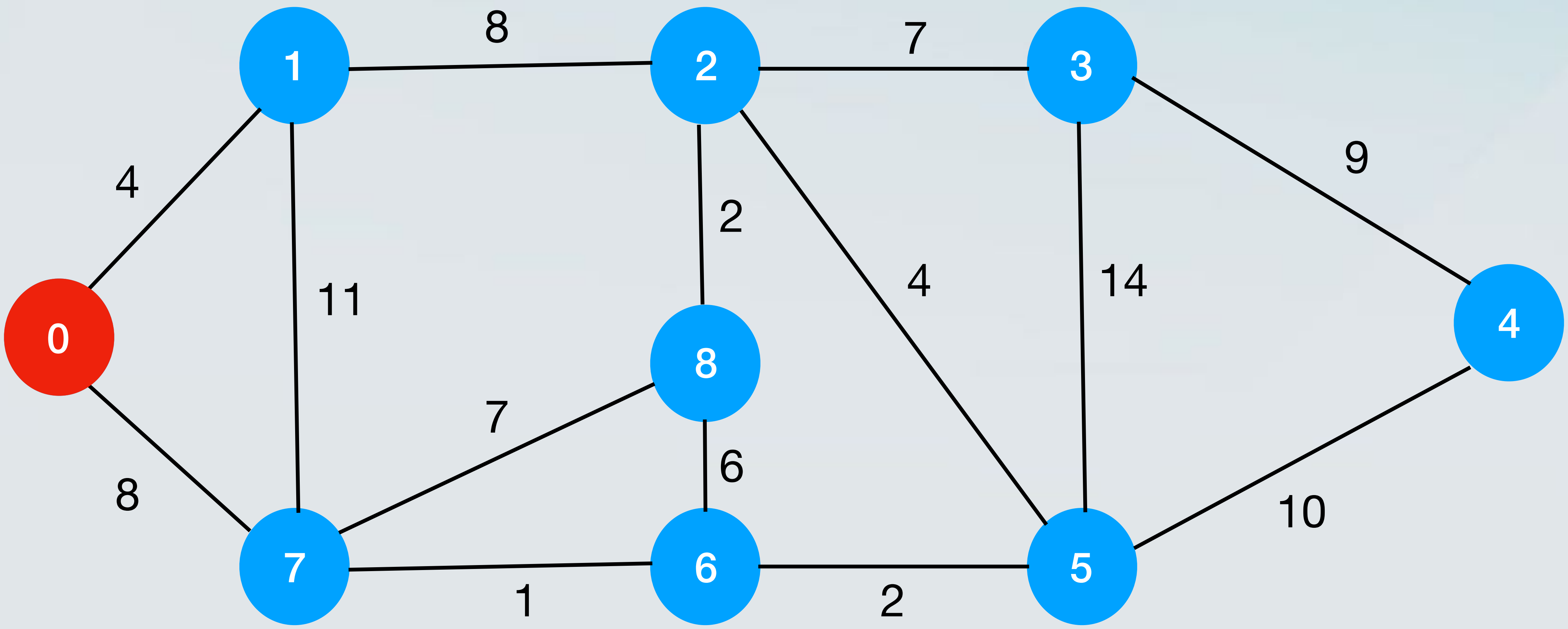
Greedy Approach #2

- Start with an empty set of edges T.
- Start with a node s.
    - Add an edge e=(s,w) to T.
    - Which one?
    - The one with the minimum cost c_e.
- We continue like this.
    - We only consider edges to neighbours that are not in the spanning tree.

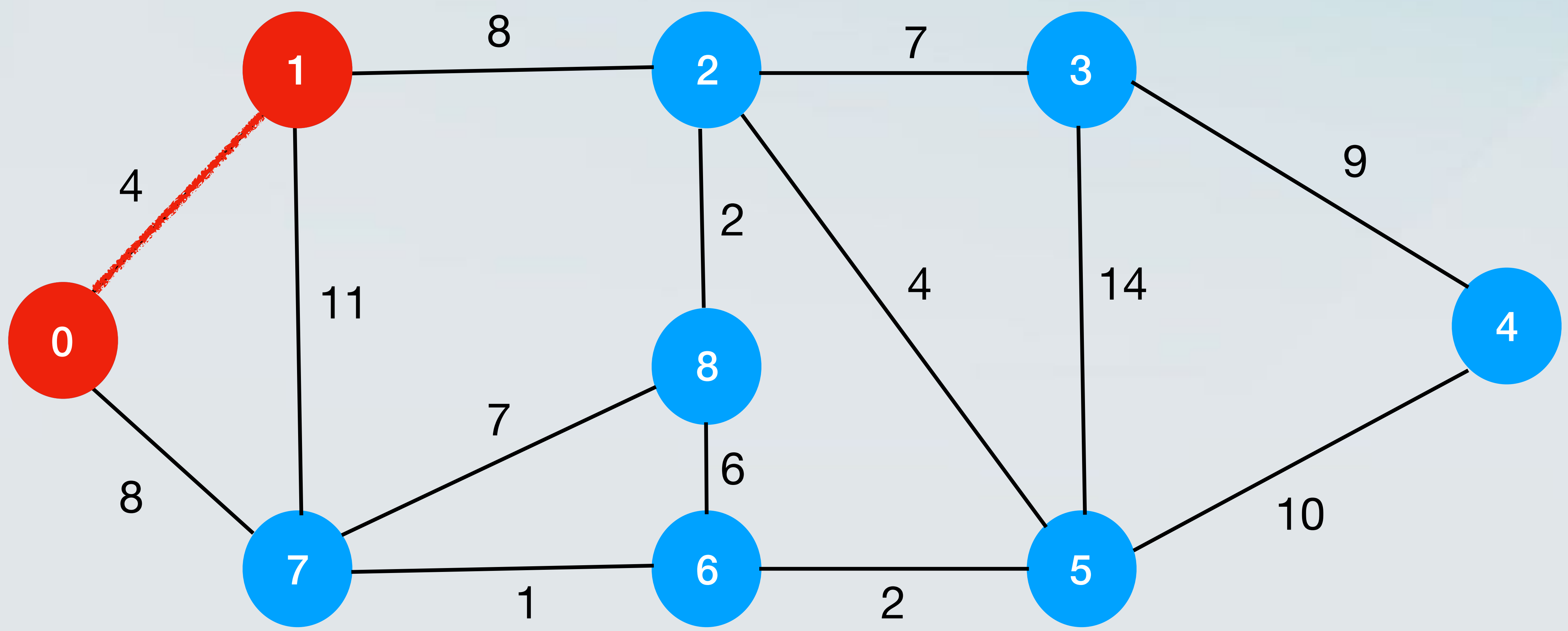
Example



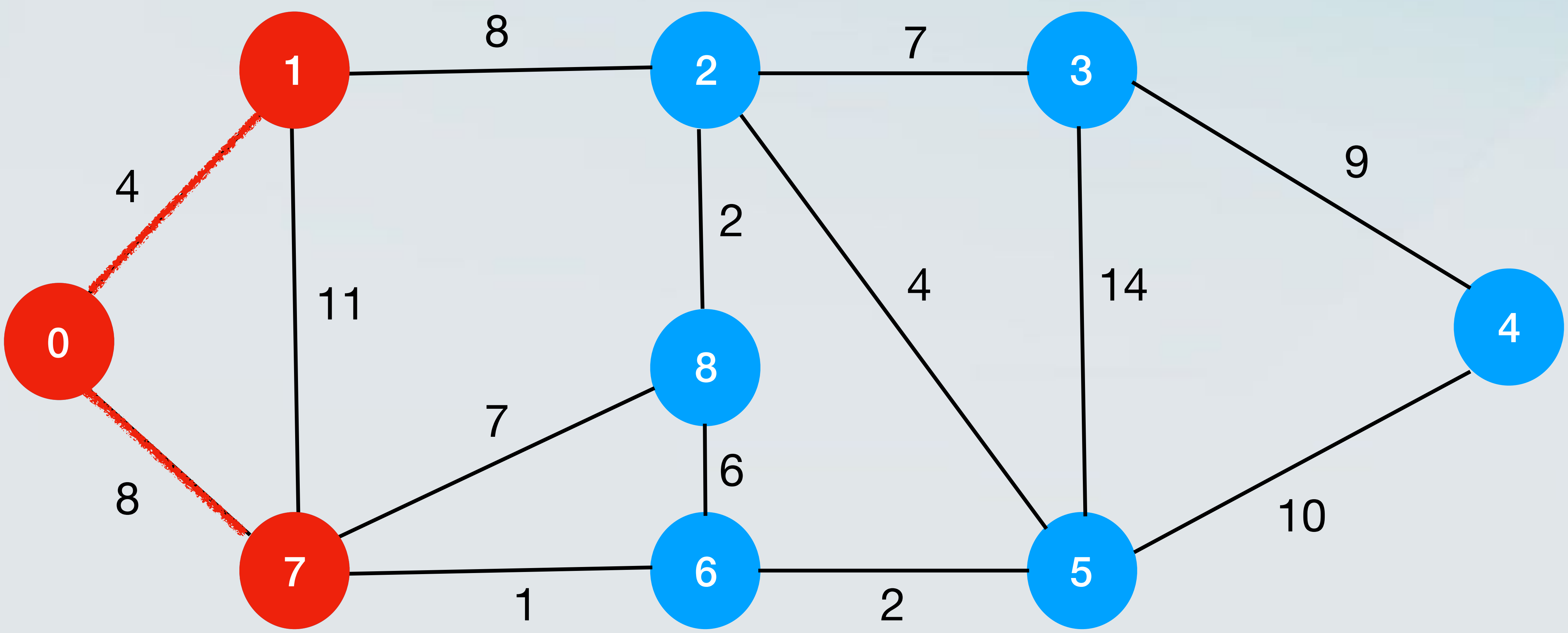
Example



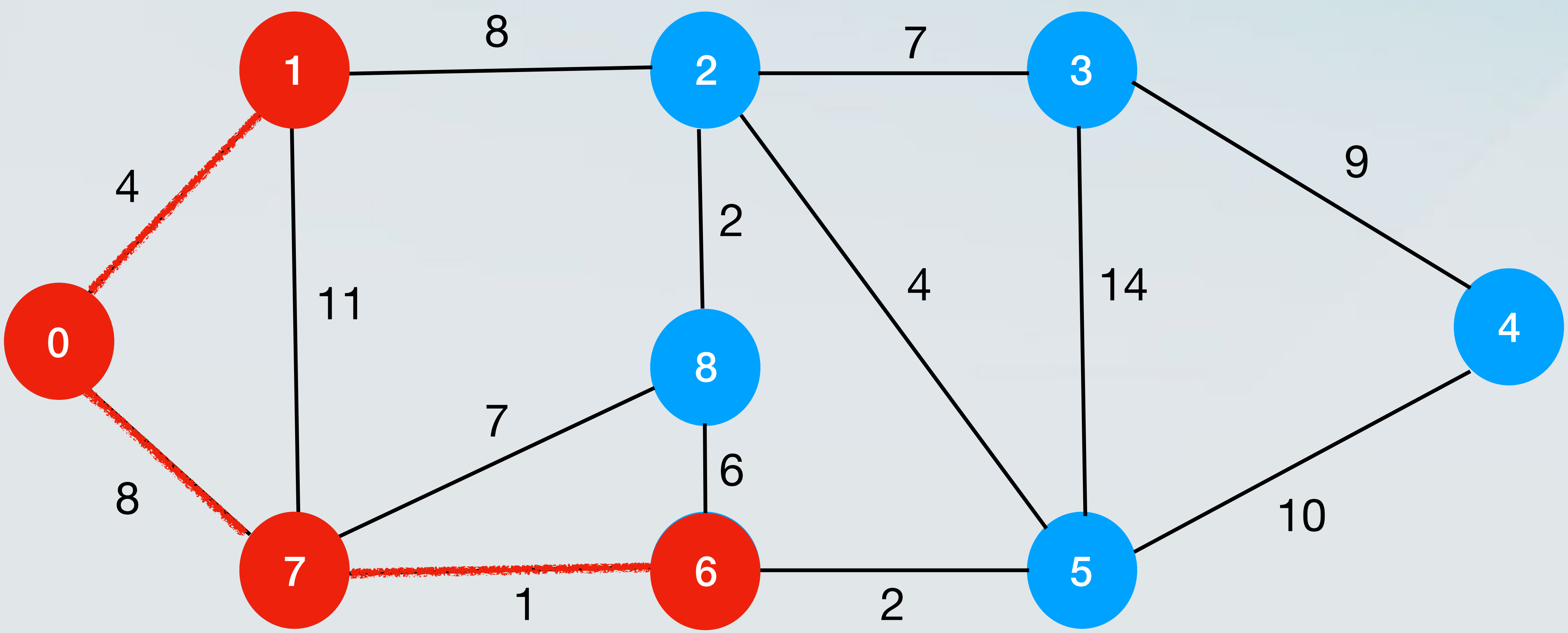
Example



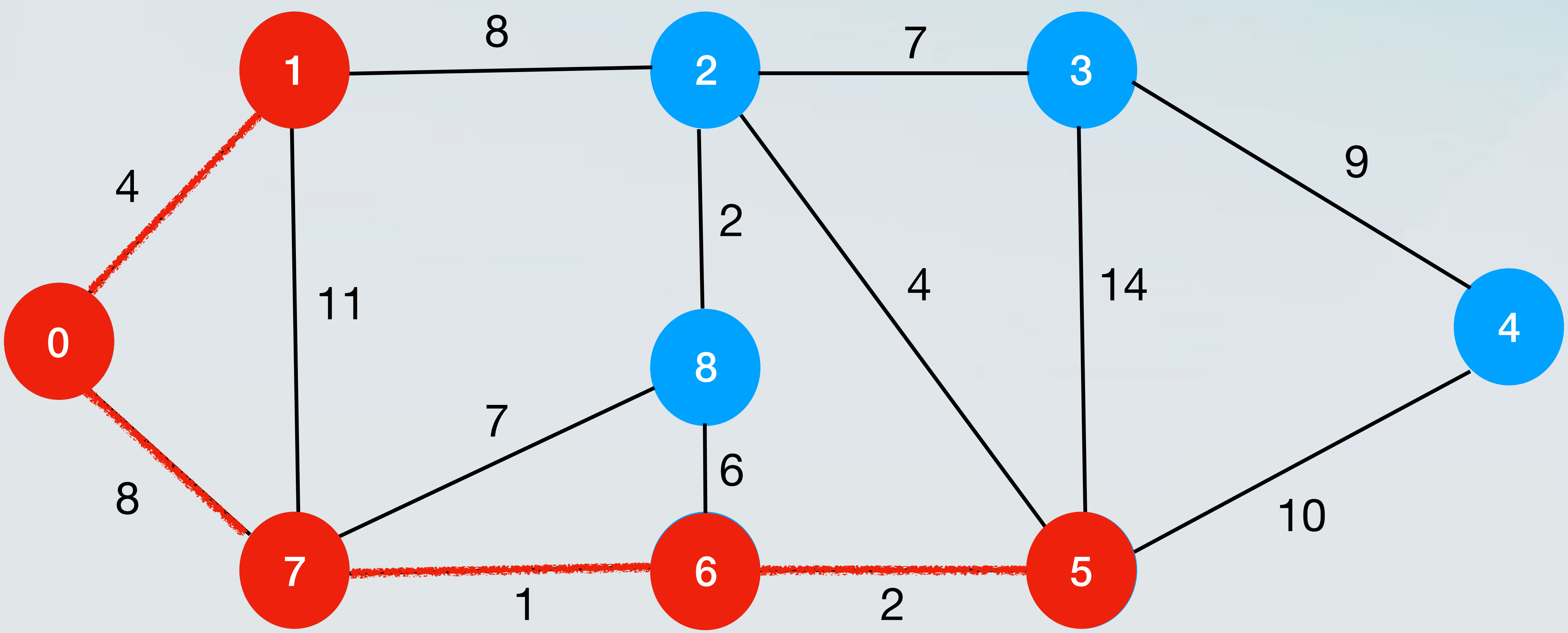
Example



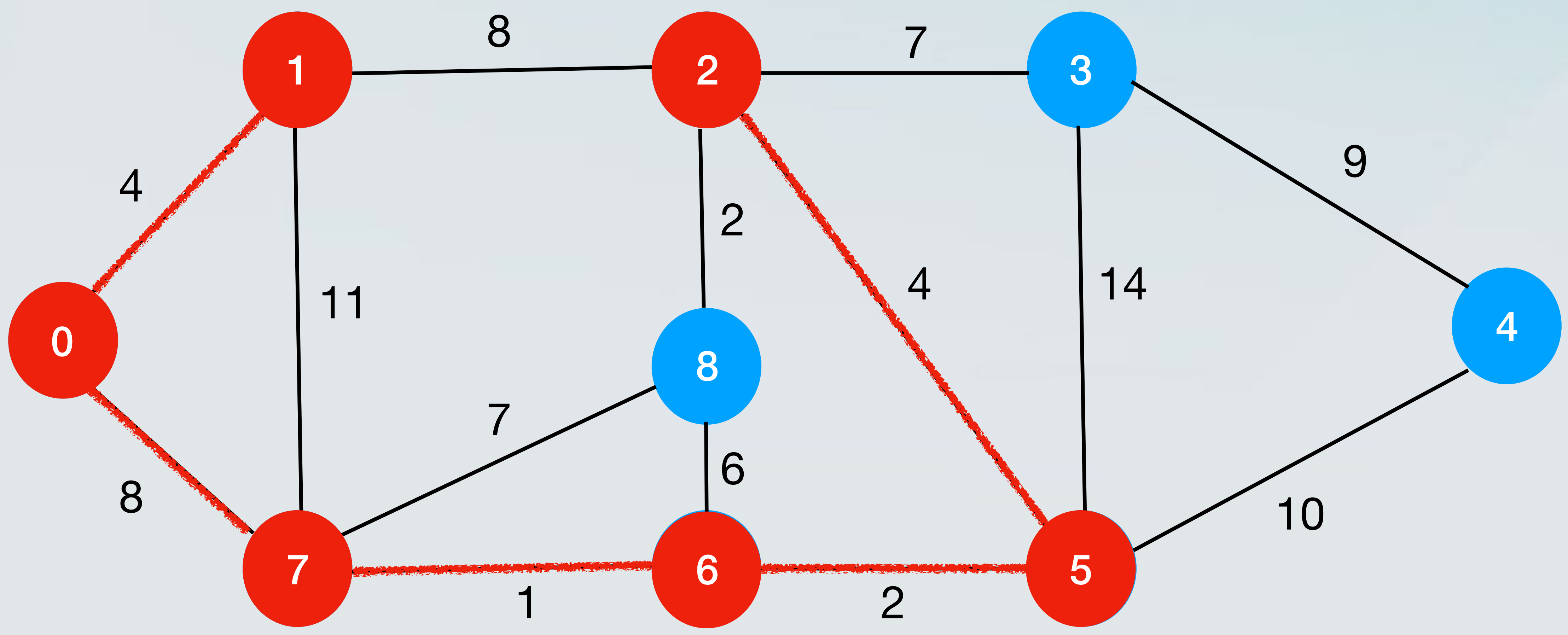
Example



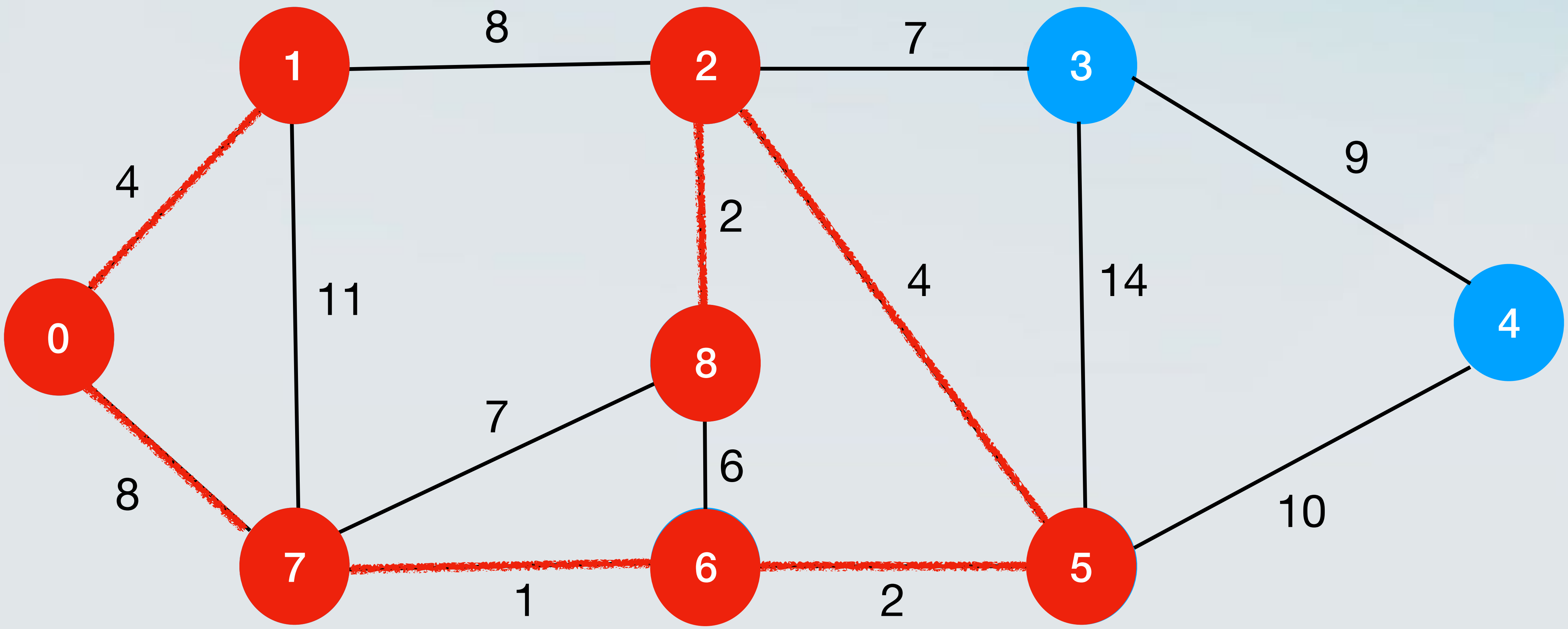
Example



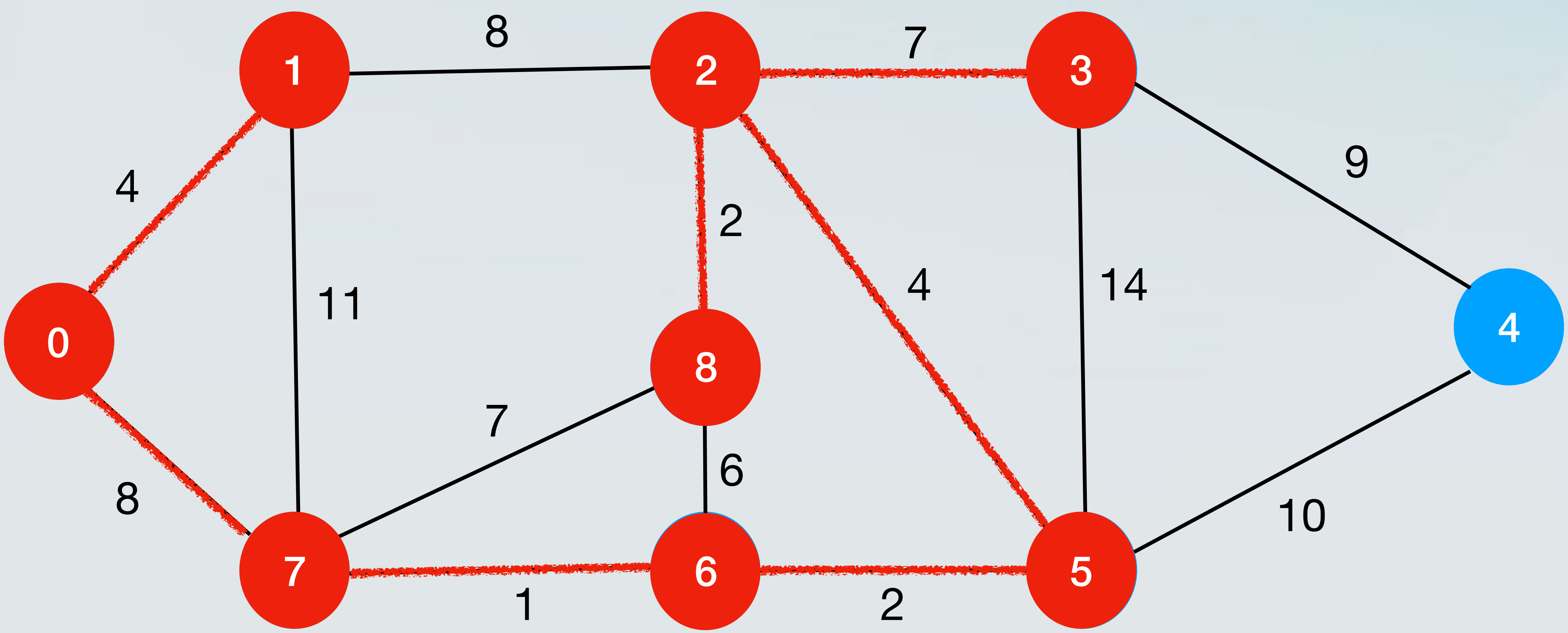
Example



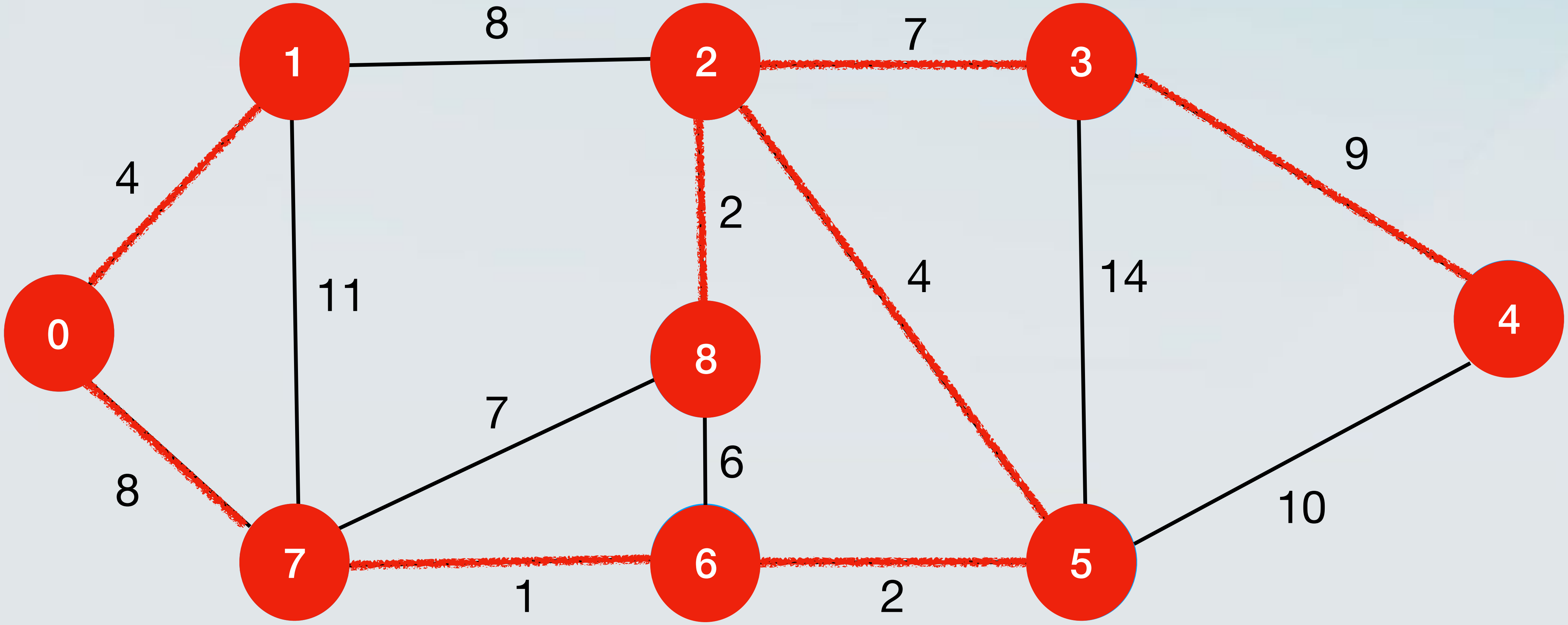
Example



Example



Example



Greedy Approach #2

- Start with an empty set of edges T.
- Start with a node s.
  - Add an edge e=(s,w) to T.
  - Which one?
  - The one with the minimum cost c_e.
- We continue like this.
  - We only consider edges to neighbours that are not in the spanning tree.

Prim's Algorithm

- Start with an empty set of edges T.
- Start with a node s.
    - Add an edge e=(s,w) to T.
    - Which one?
    - The one with the minimum cost c_e.
- We continue like this.
    - We only consider edges to neighbours that are not in the spanning tree.

Are these algorithms optimal?

Are these algorithms optimal?

- In the example, they both produced the same spanning tree.

Are these algorithms optimal?

- In the example, they both produced the same spanning tree.
- This was actually the minimum spanning tree.

Are these algorithms optimal?

- In the example, they both produced the same spanning tree.
- This was actually the minimum spanning tree.
- Do they always output the minimum spanning tree?

The cut property

The cut property

- Assume that all edge costs are distinct.

The cut property

- Assume that all edge costs are distinct.
- Let S be any subset of V,

The cut property

- Assume that all edge costs are distinct.
- Let S be any subset of V,
  - but *not empty.*

The cut property

- Assume that all edge costs are distinct.
- Let S be any subset of V,
    - but *not empty.*
    - but *not* V.

The cut property

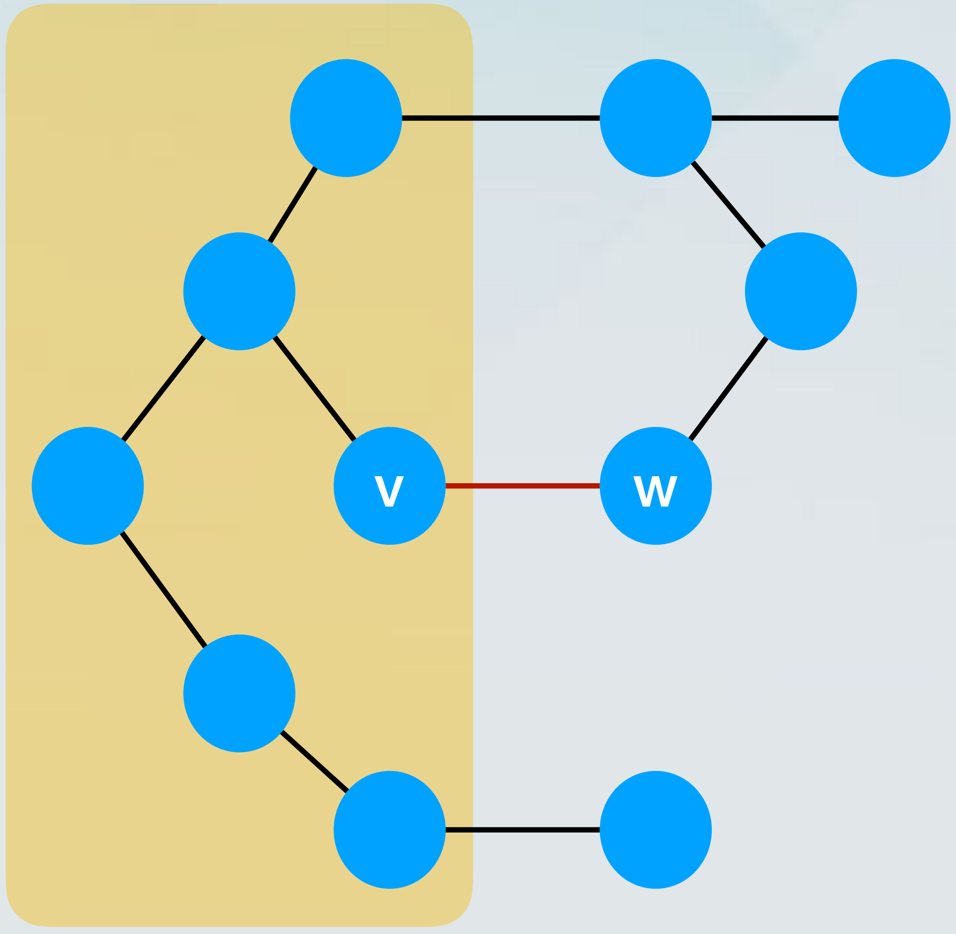
- Assume that all edge costs are distinct.
- Let S be any subset of V,
  - but *not empty.*
  - but *not* V.
- Let e=(w,v) be the minimum cost edge between S and V-S.

The cut property

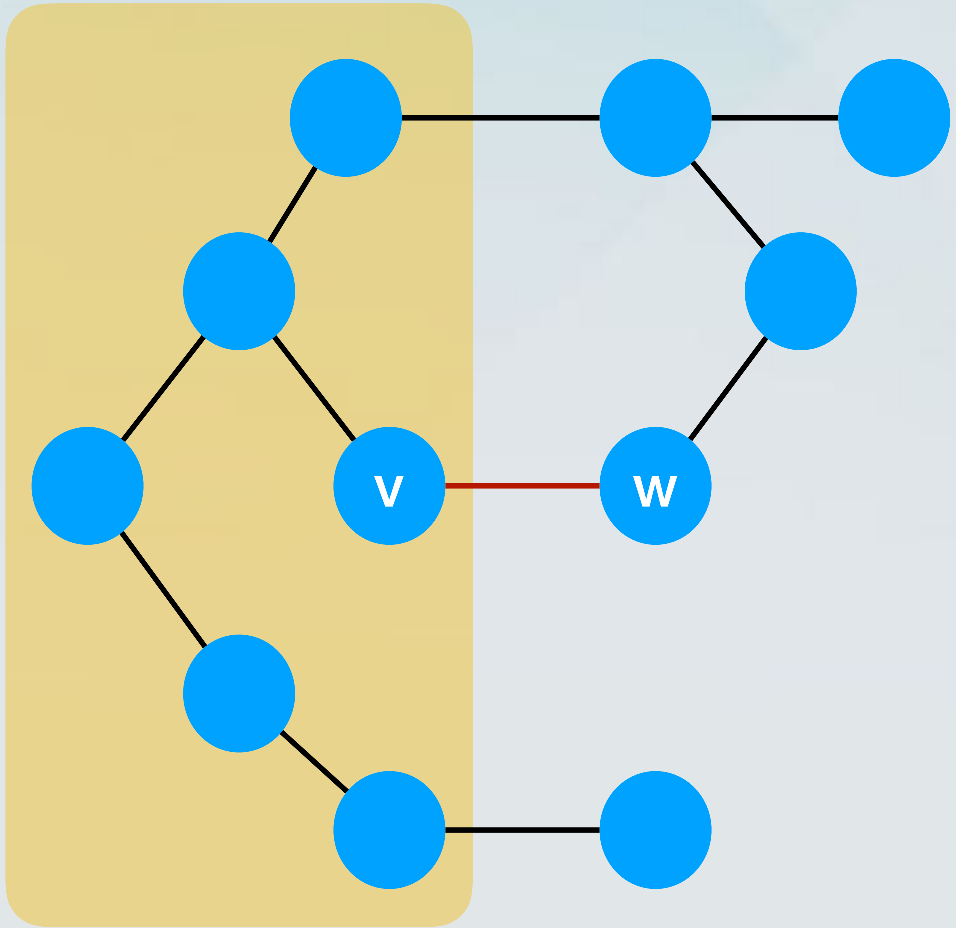
- Assume that all edge costs are distinct.
- Let S be any subset of V,
  - but *not empty.*
  - but *not* V.
- Let e=(w,v) be the minimum cost edge between S and V-S.
- Then e is contained in every minimum spanning tree.

The cut property

- Assume that all edge costs are distinct.
- Let S be any subset of V,
  - but *not empty.*
  - but *not* V.
- Let e=(w,v) be the minimum cost edge between S and V-S.
- Then e is contained in every minimum spanning tree.

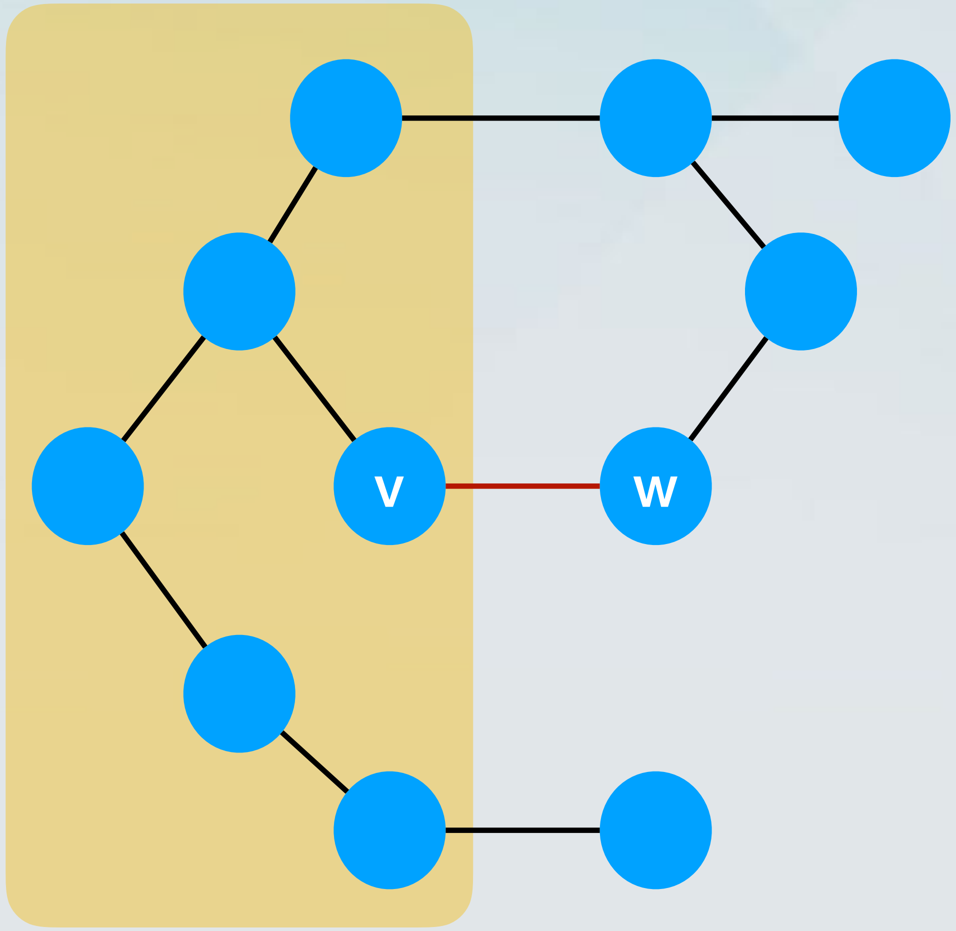


The cut property



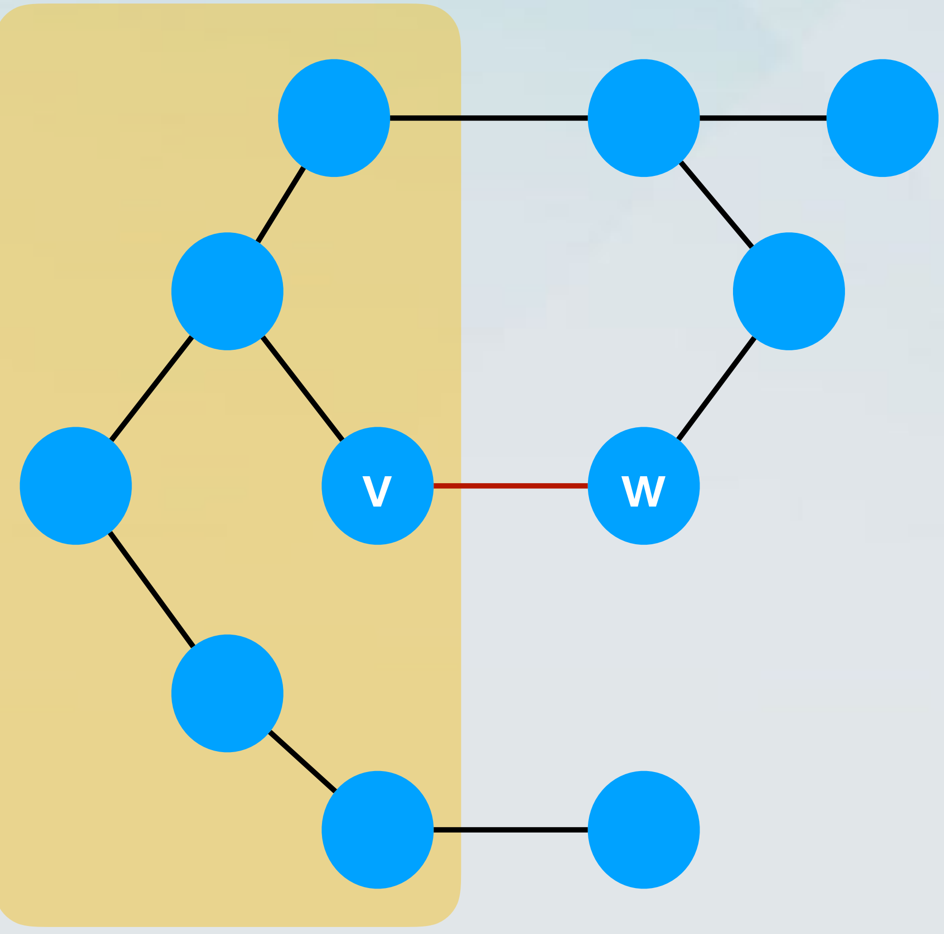
The cut property

- Then e is contained in every minimum spanning tree.



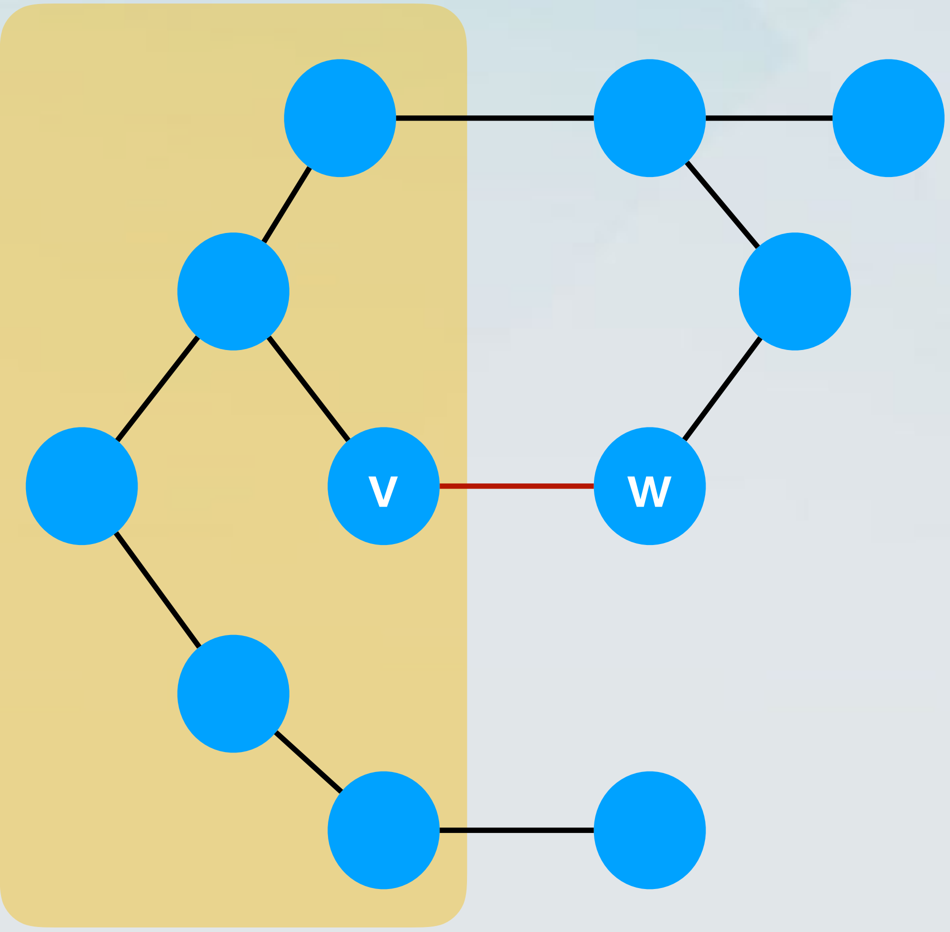
The cut property

- Then e is contained in every minimum spanning tree.
- Assume that some spanning tree T does not contain e.



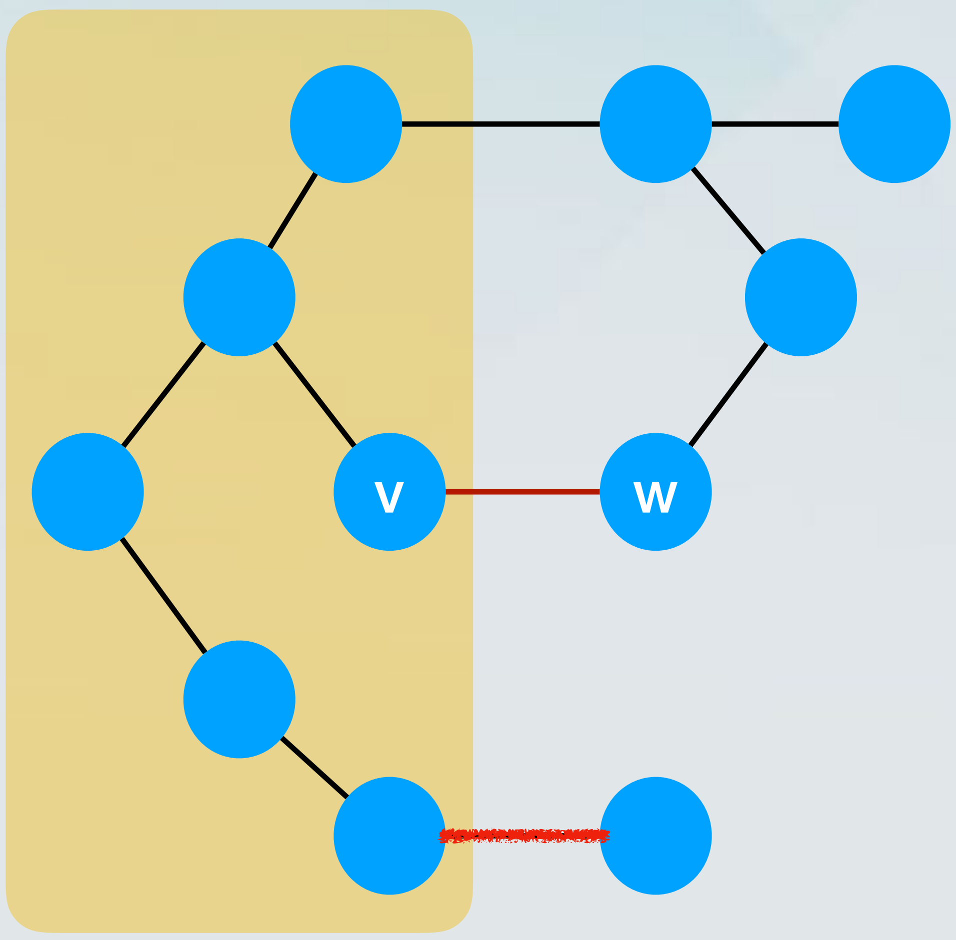
The cut property

- Then e is contained in every minimum spanning tree.
- Assume that some spanning tree T does not contain e.
- Since it is a spanning tree, it must contain some other edge f that *crosses* from S to V-S.



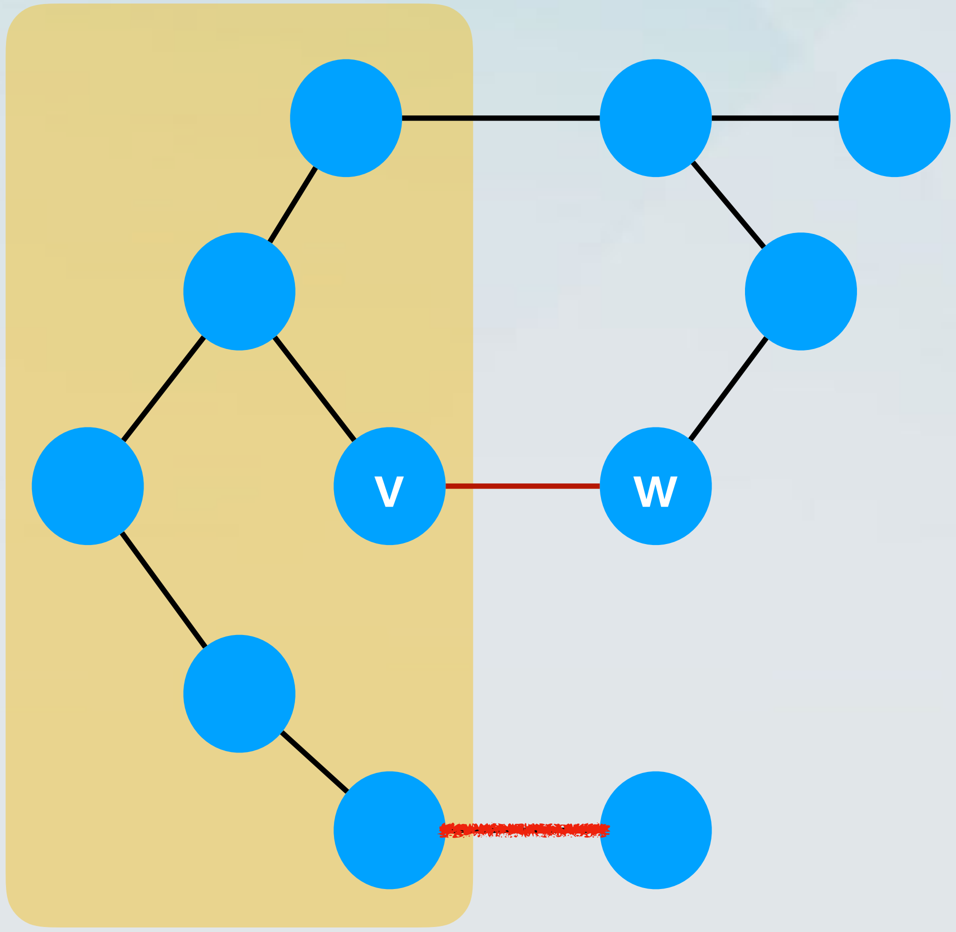
The cut property

- Then e is contained in every minimum spanning tree.
- Assume that some spanning tree T does not contain e.
- Since it is a spanning tree, it must contain some other edge f that *crosses* from S to V-S.



The cut property

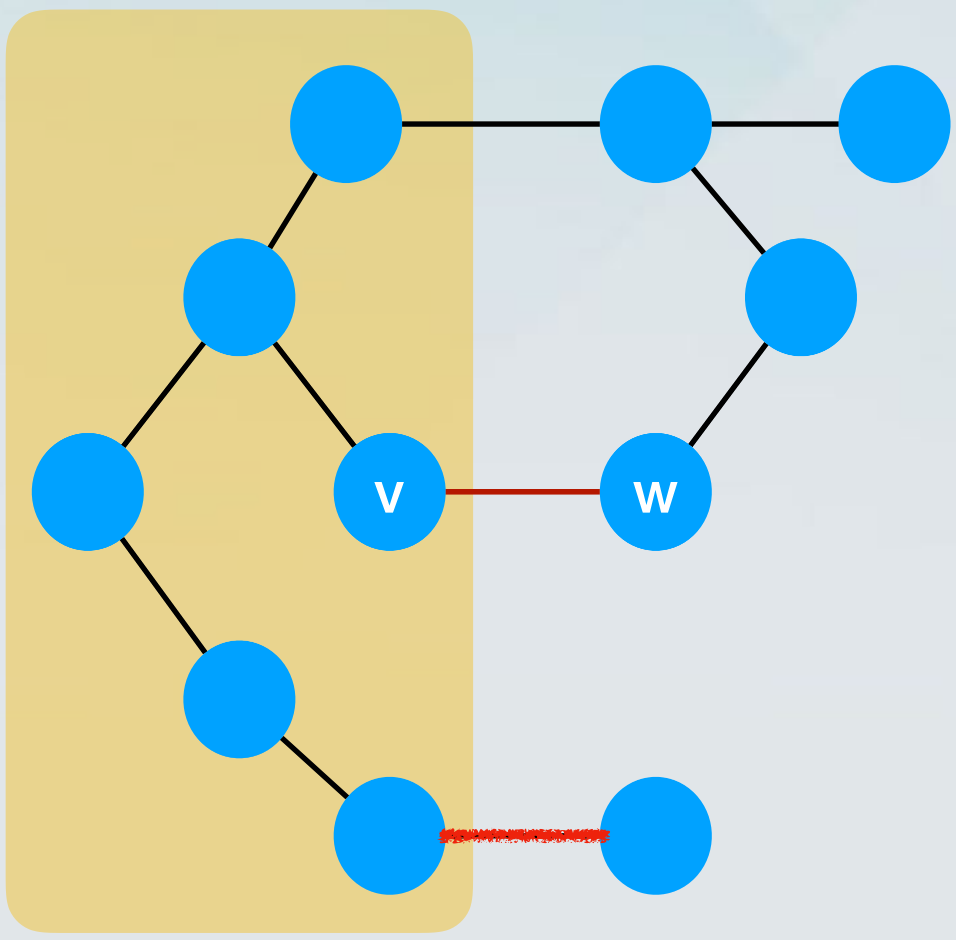
- Then e is contained in every minimum spanning tree.
- Assume that some spanning tree T does not contain e.
- Since it is a spanning tree, it must contain some other edge f that *crosses* from S to V-S.
- But $c_e \leq c_f$, so T- {f} ∪ {e} is a spanning tree of smaller cost.



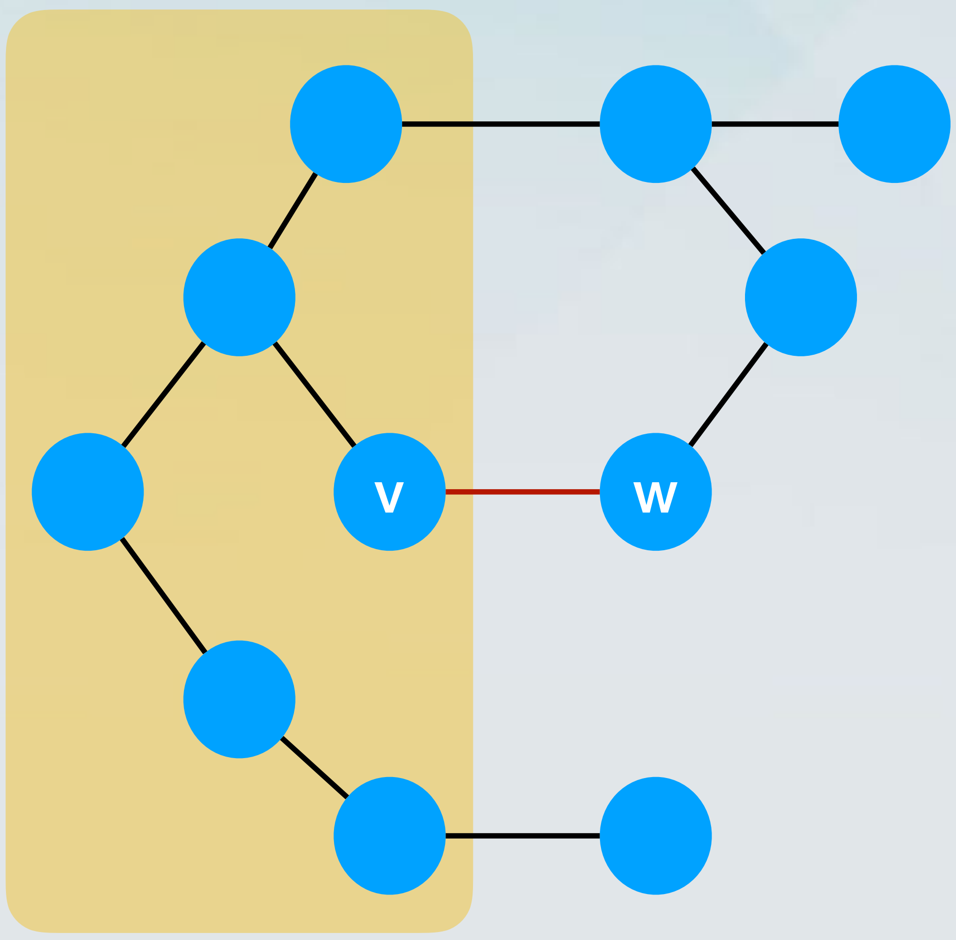
The cut property

No, $T - \{f\} \cup \{e\}$ **might not be a spanning tree!**

- Then e is contained in every minimum spanning tree.
- Assume that some spanning tree T does not contain e.
- Since it is a spanning tree, it must contain some other edge f that *crosses* from S to V-S.
- But $c_e \le c_f$, so $T - \{f\} \cup \{e\}$ is a spanning tree of smaller cost.

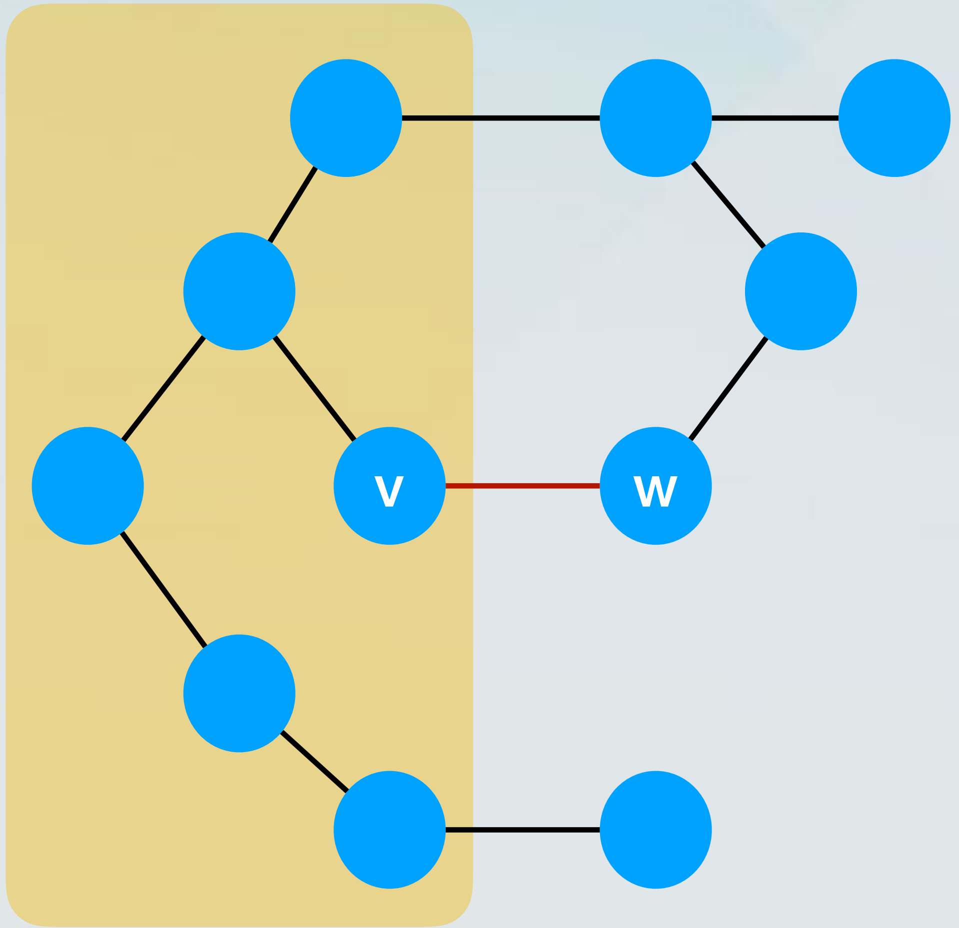


The cut property



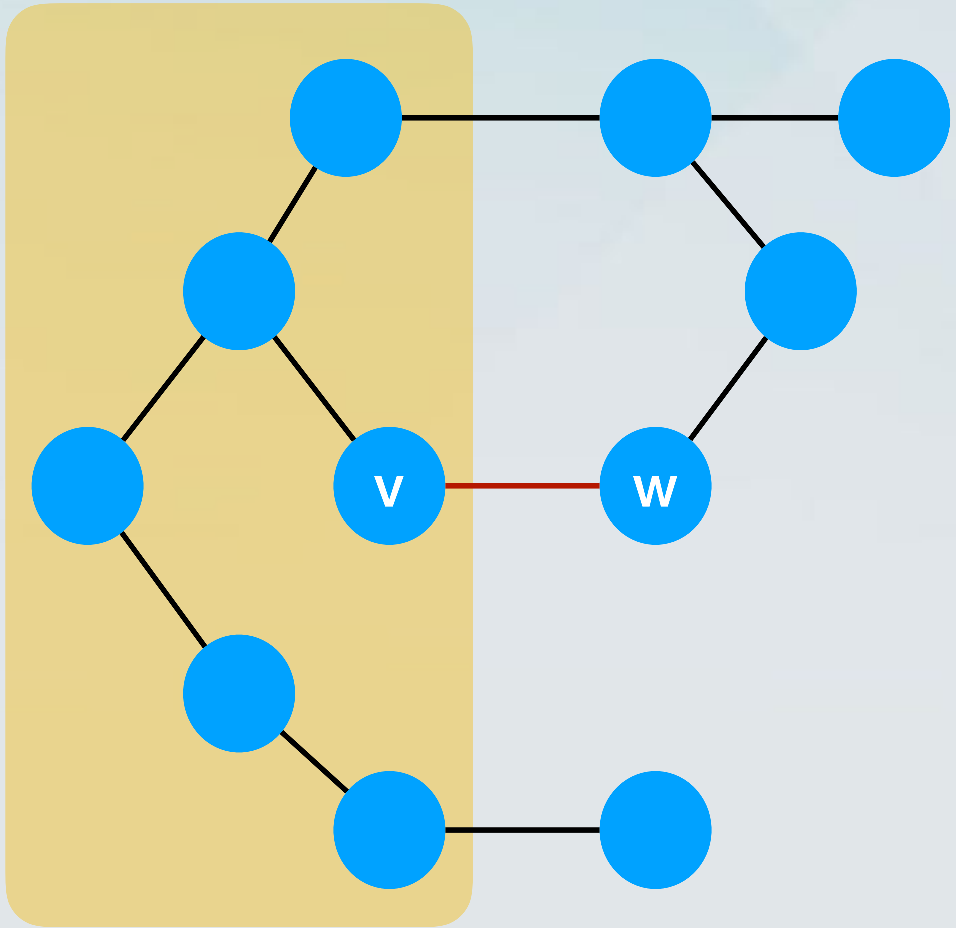
The cut property

- We can't simply select any edge.



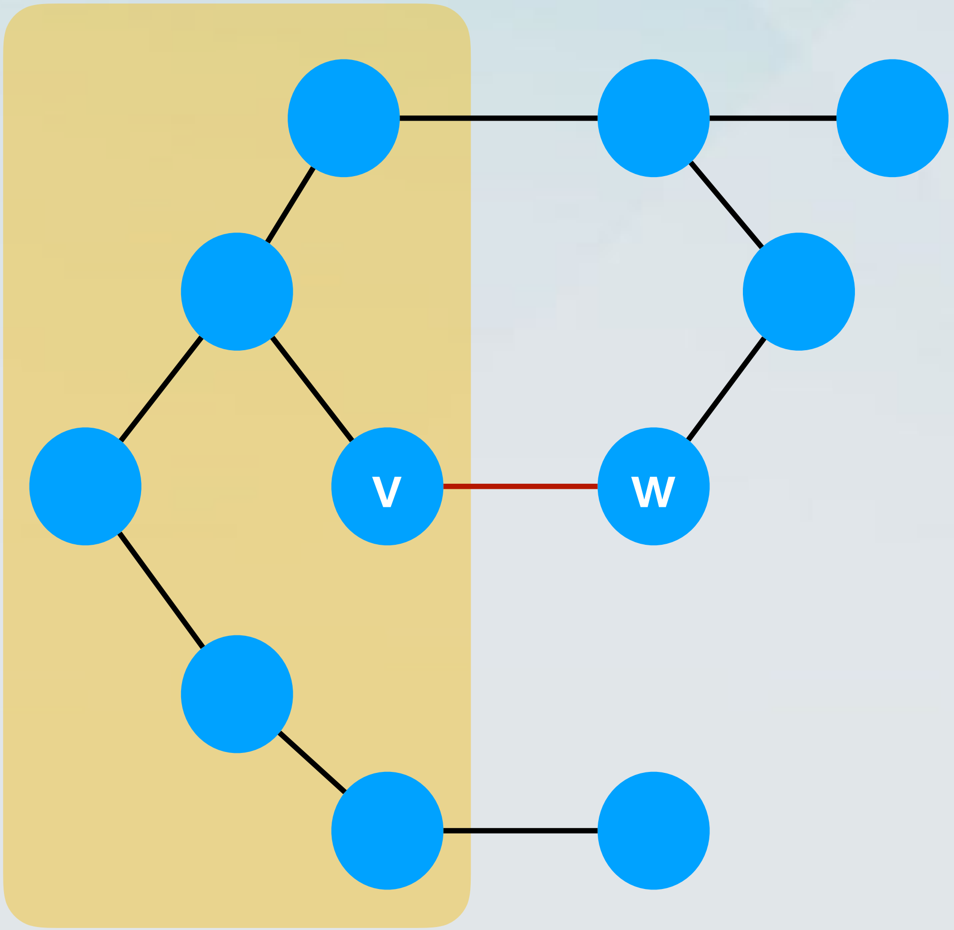
The cut property

- We can't simply select any edge.
- We need to select an edge e' which



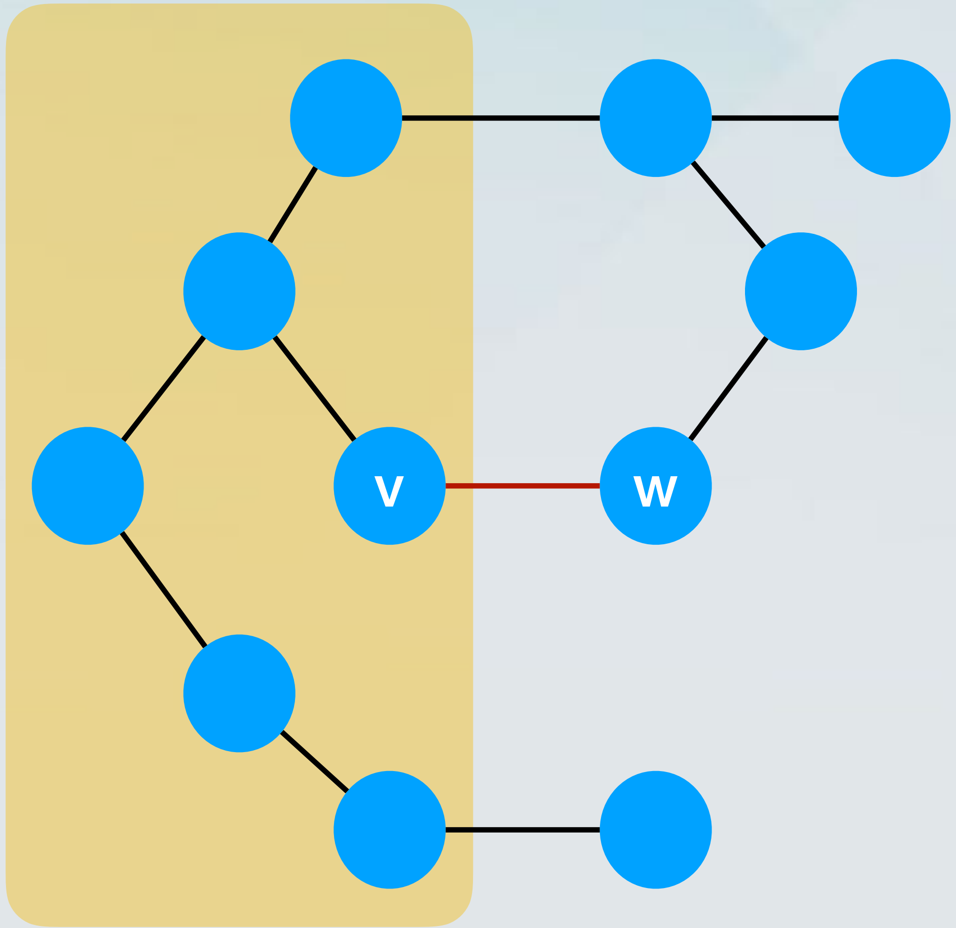
The cut property

- We can't simply select any edge.
- We need to select an edge e' which
  - is more expensive than e.



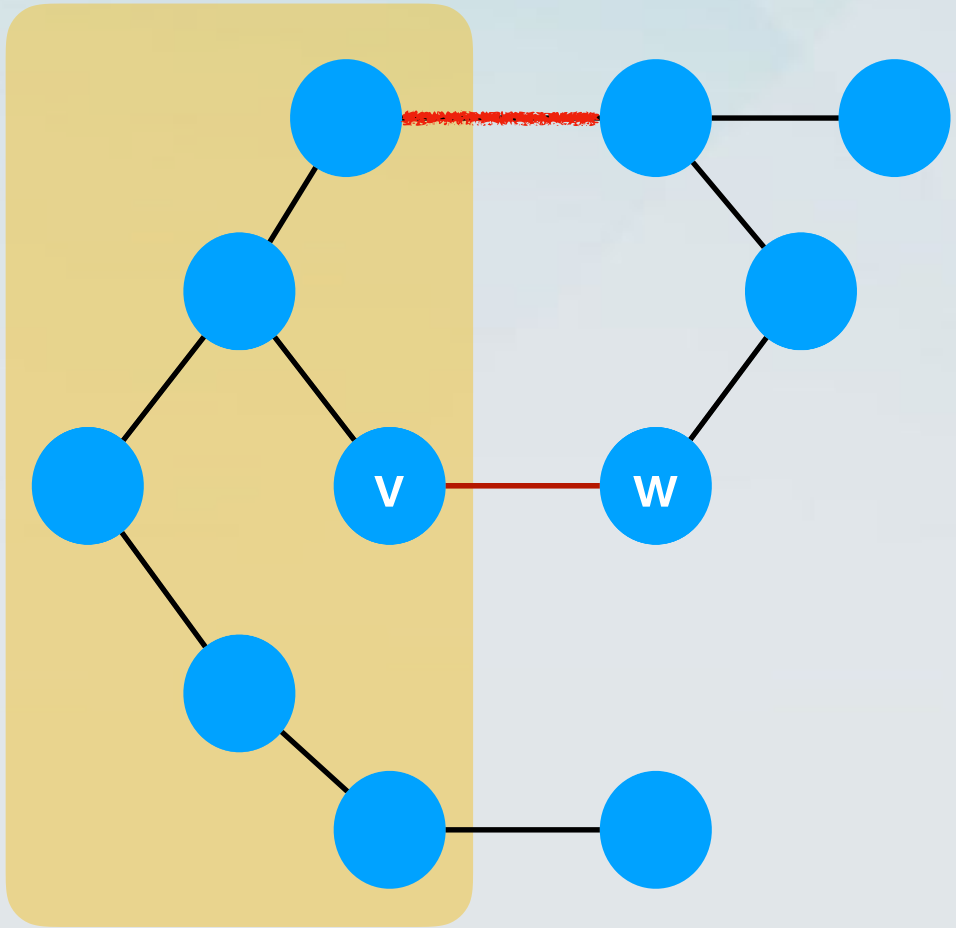
The cut property

- We can't simply select any edge.
- We need to select an edge e' which
  - is more expensive than e.
  - still results in a spanning tree, if used instead of e.

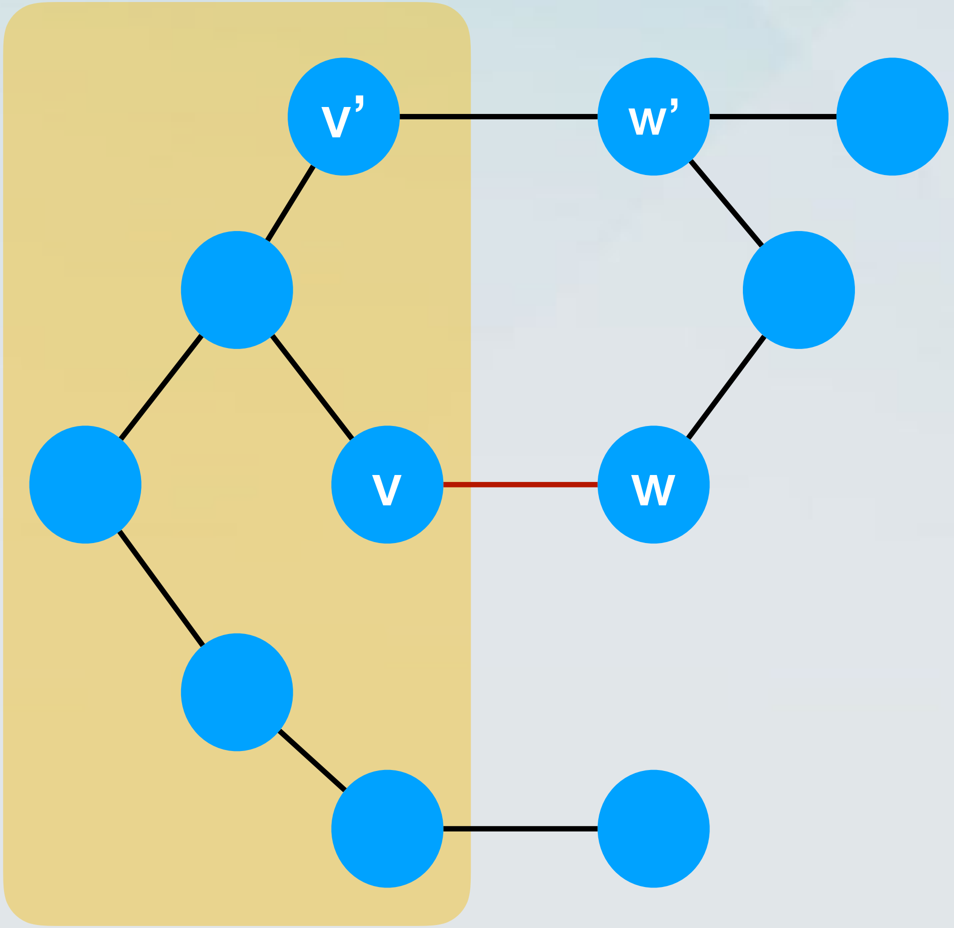


The cut property

- We can't simply select any edge.
- We need to select an edge e' which
  - is more expensive than e.
  - still results in a spanning tree, if used instead of e.

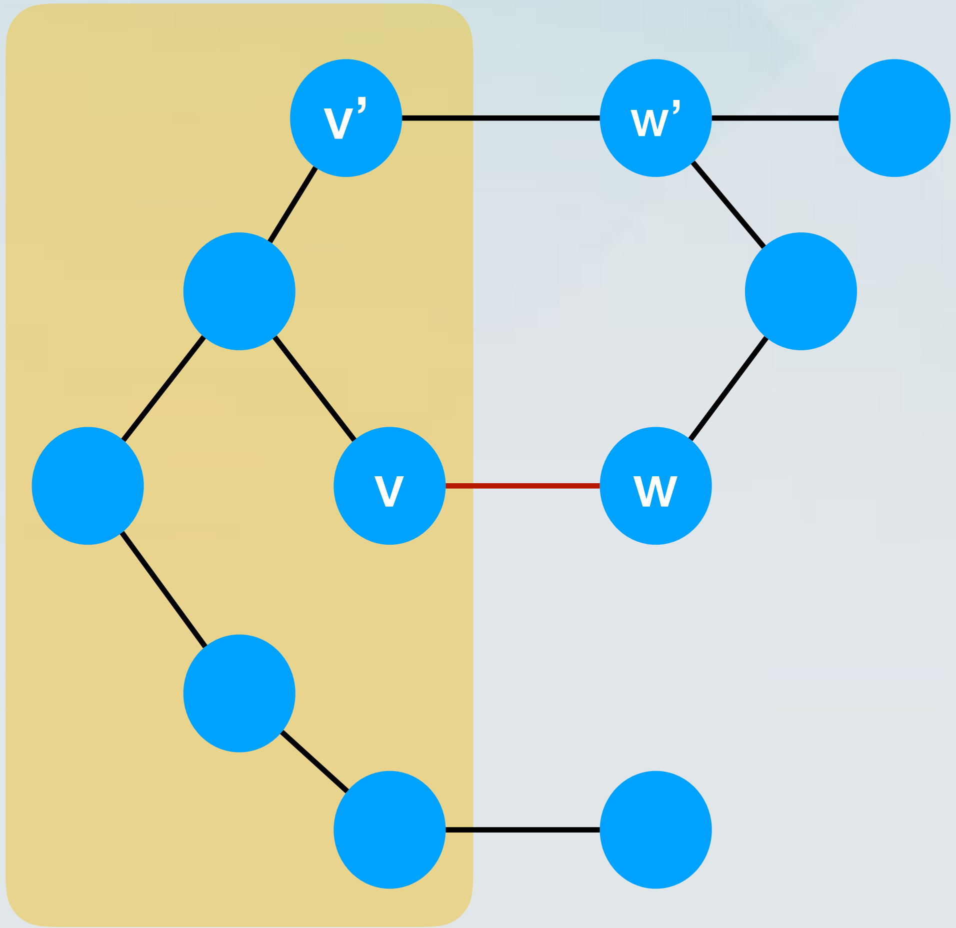


The cut property



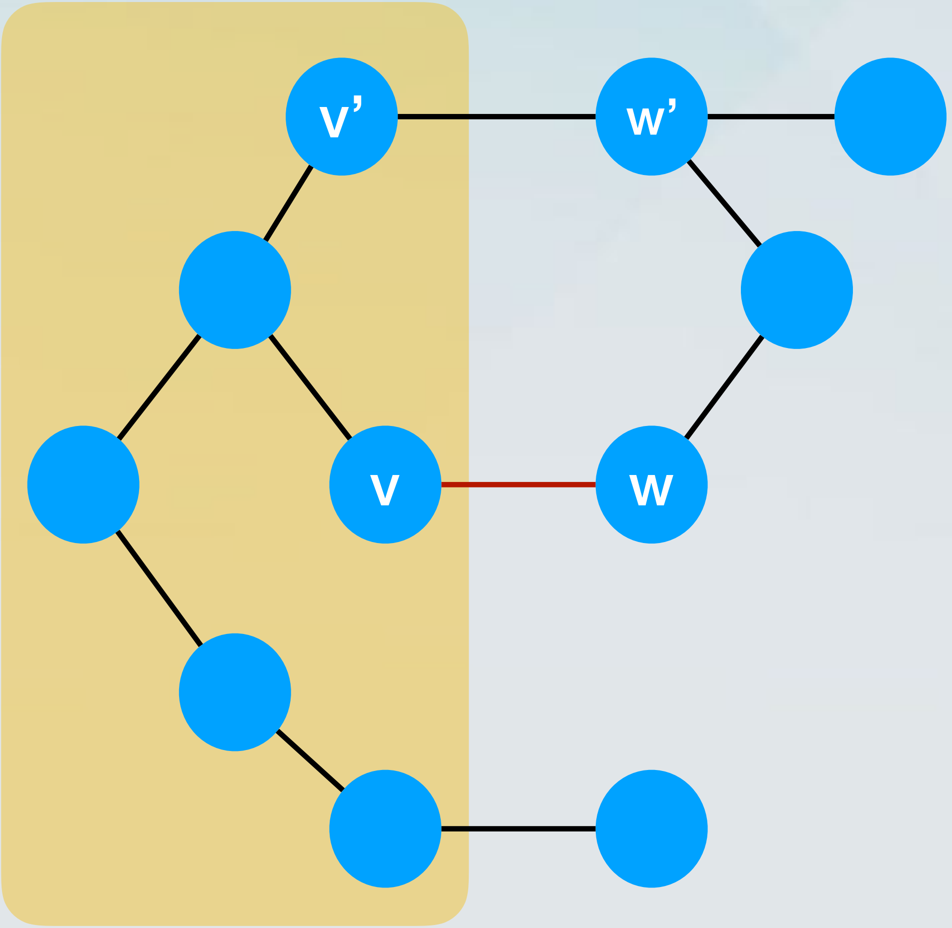
The cut property

- Let T be a minimum spanning tree which does **not** contain e=(v, w).



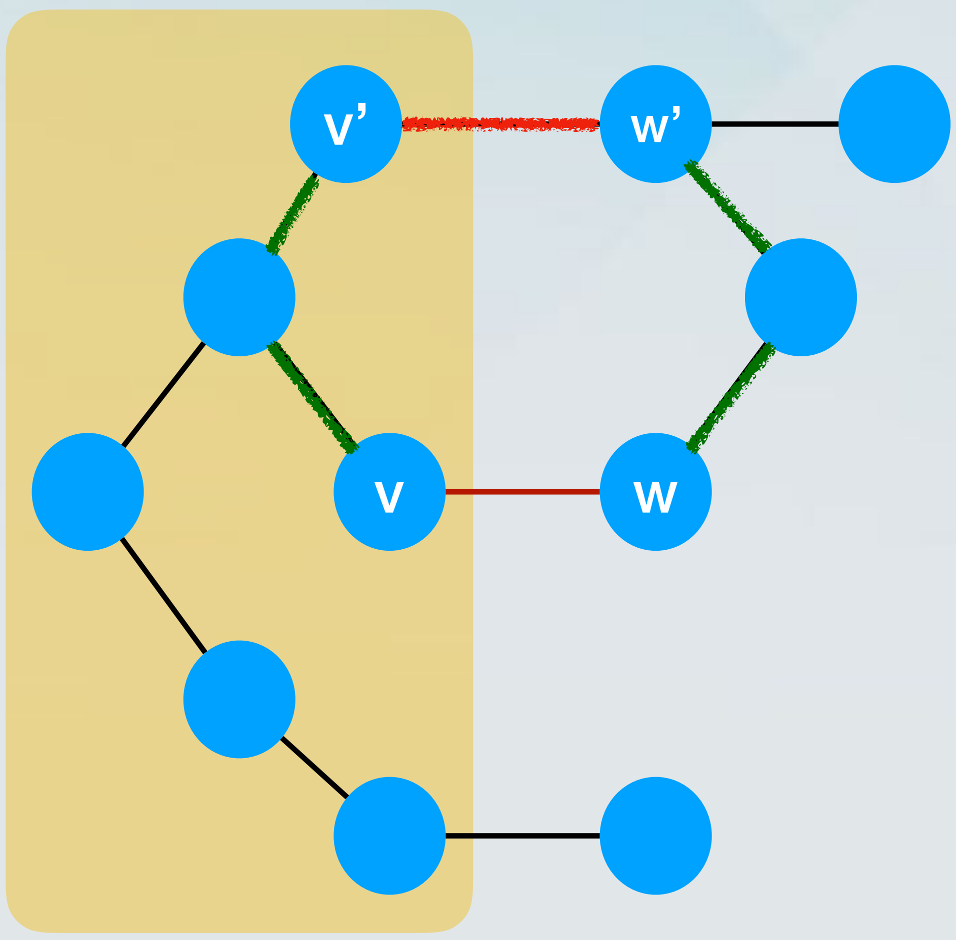
The cut property

- Let T be a minimum spanning tree which does **not** contain e=(v, w).
- Since T is a spanning tree, there is path from v to w.



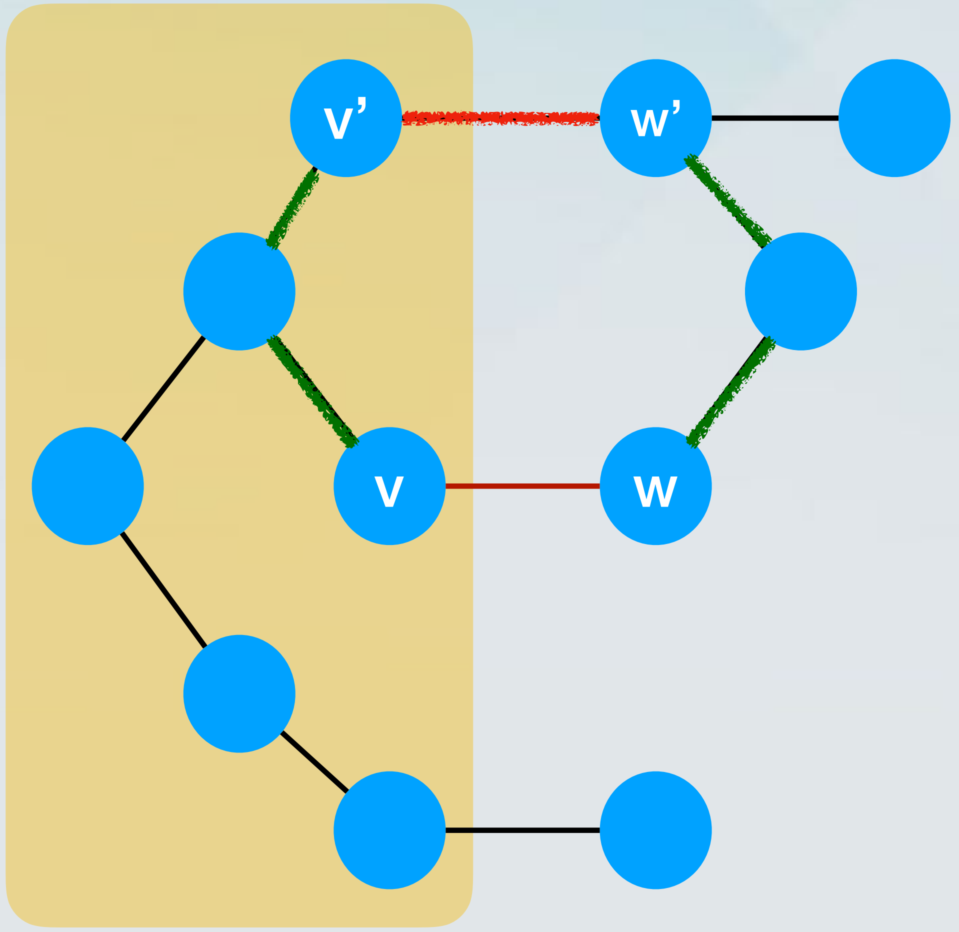
The cut property

- Let T be a minimum spanning tree which does **not** contain e=(v, w).
- Since T is a spanning tree, there is path from v to w.



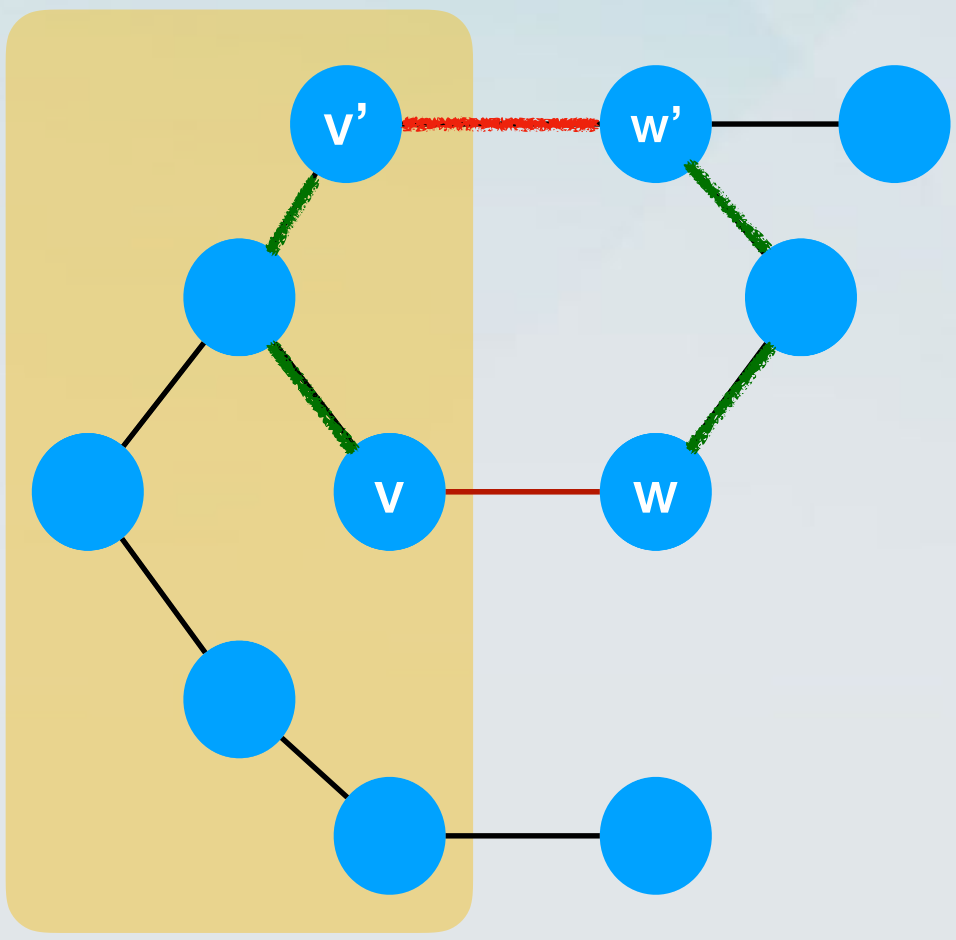
The cut property

- Let T be a minimum spanning tree which does **not** contain e=(v, w).
- Since T is a spanning tree, there is path from v to w.
- Let w' be the first node encountered in V-T and let v' be the one before it. Let e'=(v', w').



The cut property

- Let T be a minimum spanning tree which does **not** contain e=(v, w).
- Since T is a spanning tree, there is path from v to w.
- Let w’ be the first node encountered in V-T and let v’ be the one before it. Let e’=(v’, w’).
- Consider T’ = T -{e’} ∪ {e}.



Kruskal’s algorithm is optimal

Kruskal's algorithm is optimal

- Consider any edge e=(u, w) that Kruskal's algorithm adds to the output on some step.

Kruskal's algorithm is optimal

- Consider any edge e=(u, w) that Kruskal's algorithm adds to the output on some step.
- Let S be the set of nodes reachable from u just before e is added to the output.

Kruskal’s algorithm is optimal

- Consider any edge e=(u, w) that Kruskal’s algorithm adds to the output on some step.
- Let S be the set of nodes reachable from u just before e is added to the output.
- It holds that v is in S and w is in V-S. (Why?)

Kruskal's algorithm is optimal

- Consider any edge e=(u, w) that Kruskal's algorithm adds to the output on some step.
- Let S be the set of nodes reachable from u just before e is added to the output.
- It holds that v is in S and w is in V-S. (Why?)
    - Because otherwise adding e would create a cycle.

Kruskal's algorithm is optimal

- Consider any edge e=(u, w) that Kruskal's algorithm adds to the output on some step.
- Let S be the set of nodes reachable from u just before e is added to the output.
- It holds that v is in S and w is in V-S. (Why?)
  - Because otherwise adding e would create a cycle.
- The algorithm has not found any edge crossing S and V-S to the output. (Why?)

Kruskal's algorithm is optimal

- Consider any edge e=(u, w) that Kruskal's algorithm adds to the output on some step.
- Let S be the set of nodes reachable from u just before e is added to the output.
- It holds that v is in S and w is in V-S. (Why?)
  - Because otherwise adding e would create a cycle.
- The algorithm has not found any edge crossing S and V-S to the output. (Why?)
  - Such an edge would have been added to the output by the algorithm.

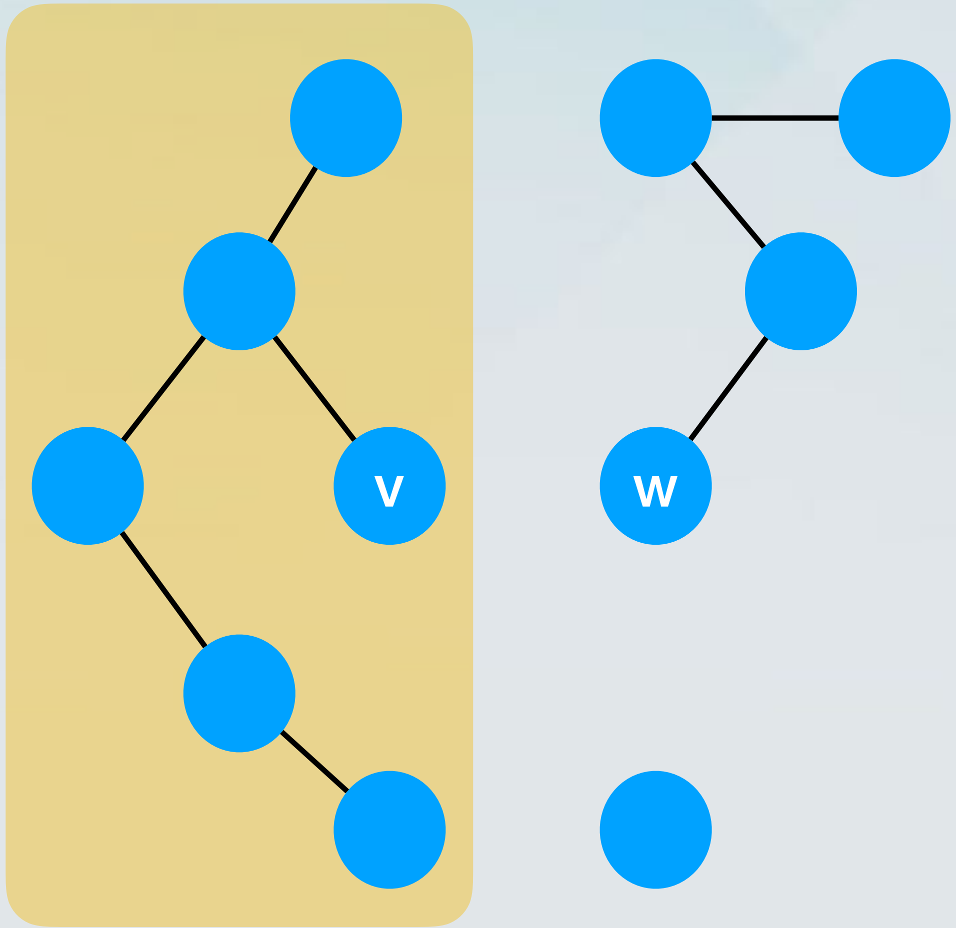
Kruskal's algorithm is optimal

- Consider any edge e=(u, w) that Kruskal's algorithm adds to the output on some step.
- Let S be the set of nodes reachable from u just before e is added to the output.
- It holds that v is in S and w is in V-S. (Why?)
  - Because otherwise adding e would create a cycle.
- The algorithm has not found any edge crossing S and V-S to the output. (Why?)
  - Such an edge would have been added to the output by the algorithm.
- The edge e must be the cheapest edge crossing S and V-S.

Kruskal's algorithm is optimal

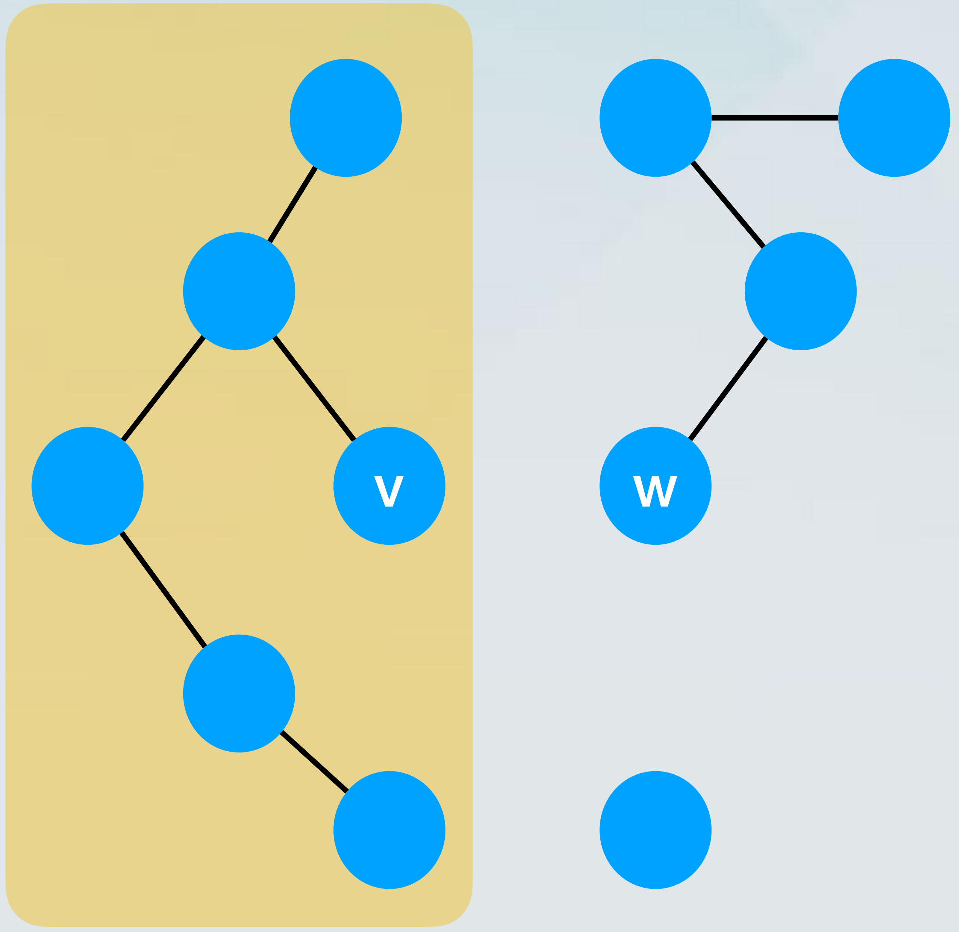
- Consider any edge e=(u, w) that Kruskal's algorithm adds to the output on some step.
- Let S be the set of nodes reachable from u just before e is added to the output.
- It holds that v is in S and w is in V-S. (Why?)
  - Because otherwise adding e would create a cycle.
- The algorithm has not found any edge crossing S and V-S to the output. (Why?)
  - Such an edge would have been added to the output by the algorithm.
- The edge e must be the cheapest edge crossing S and V-S.
- By the cut property, it belongs to every minimum spanning tree.

Is it feasible?



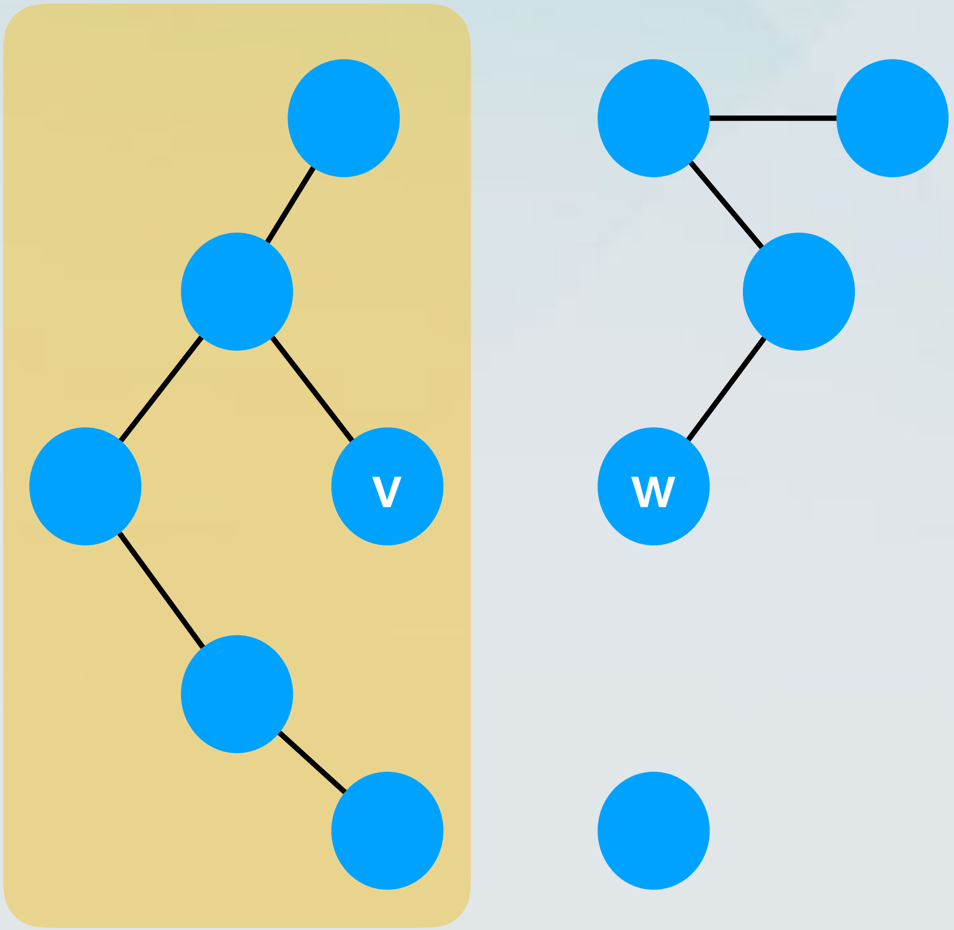
Is it feasible?

- i.e., does it always produce a spanning tree?



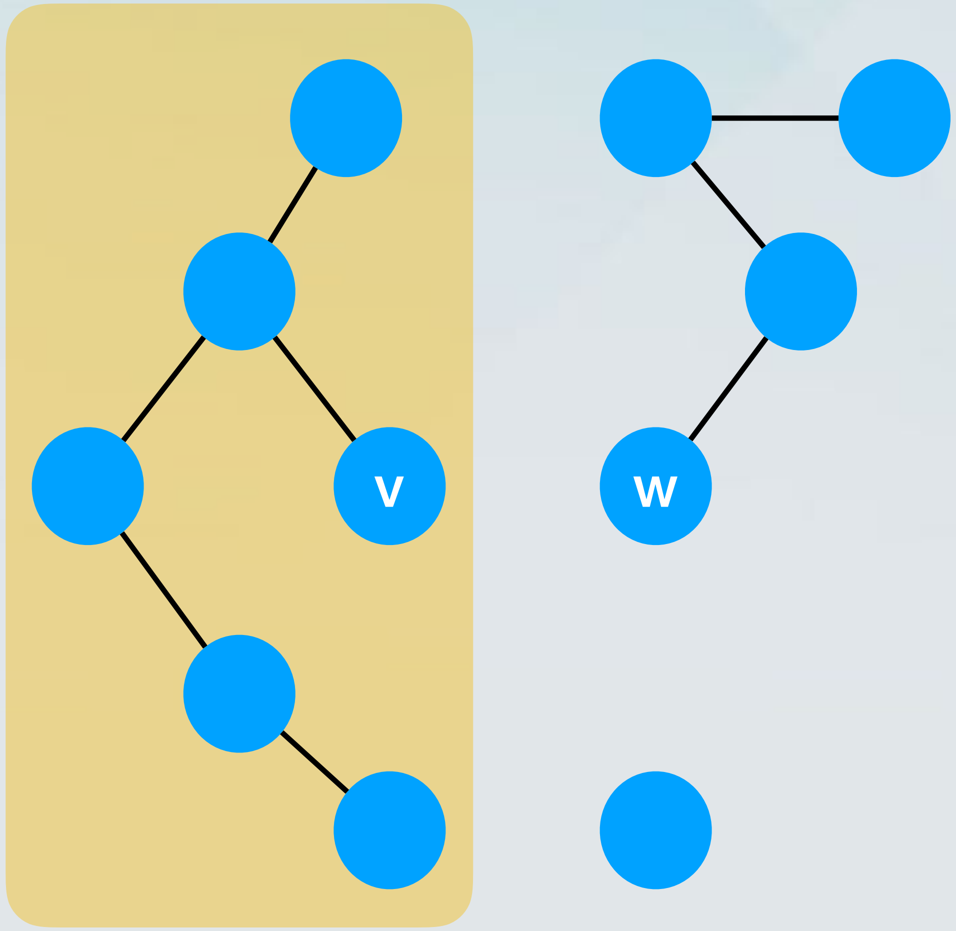
Is it feasible?

- i.e., does it always produce a spanning tree?
- The algorithm explicitly avoids cycles.



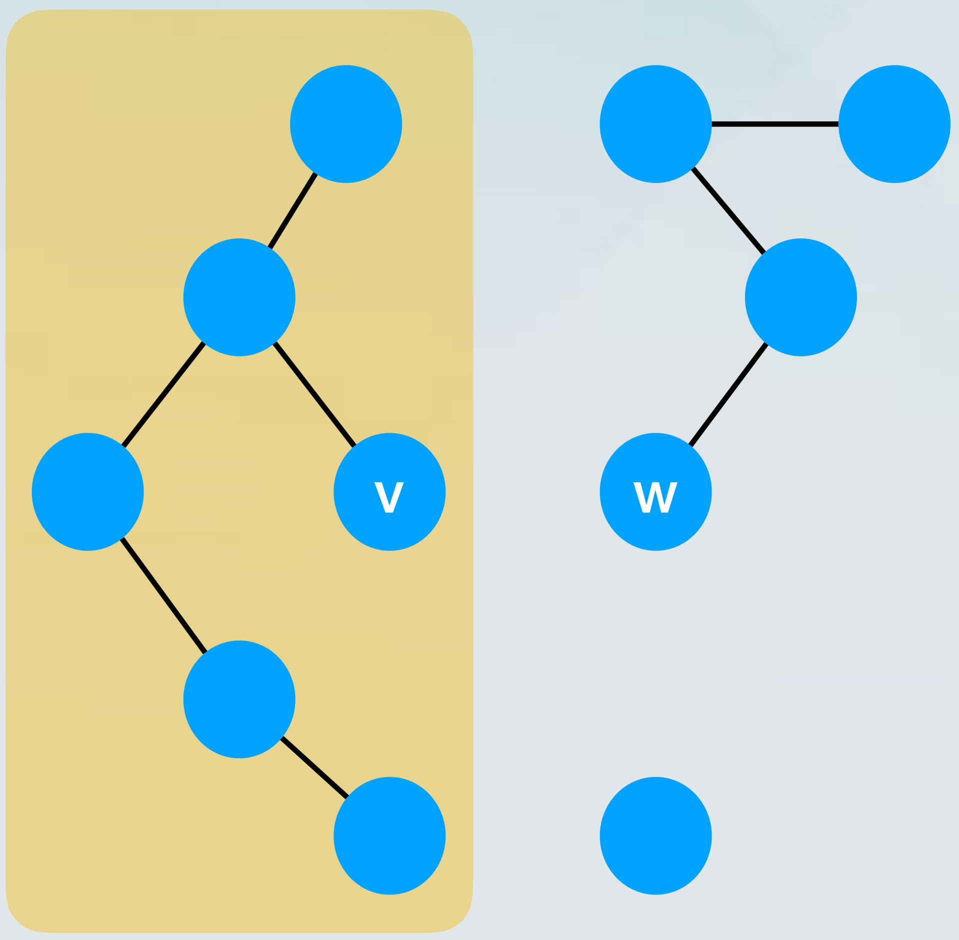
Is it feasible?

- i.e., does it always produce a spanning tree?
- The algorithm explicitly avoids cycles.
  - Output T is a forest.



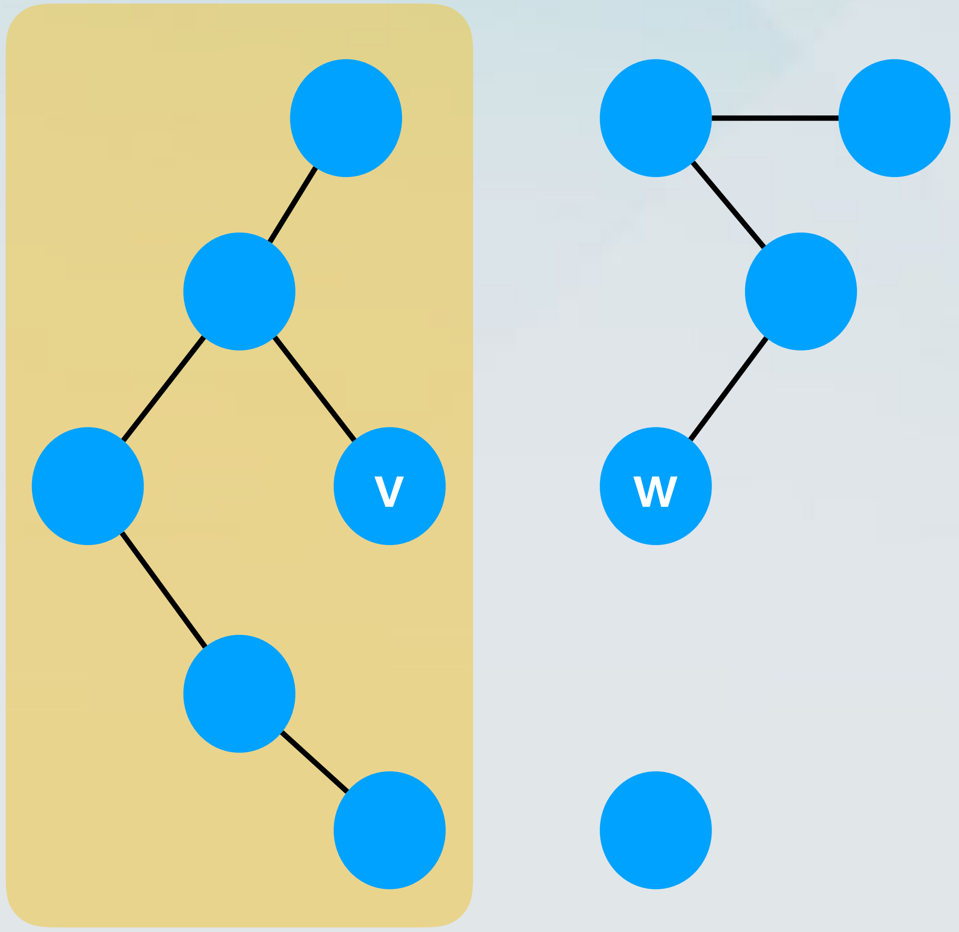
Is it feasible?

- i.e., does it always produce a spanning tree?
- The algorithm explicitly avoids cycles.
  - Output T is a forest.
- Is it a tree?



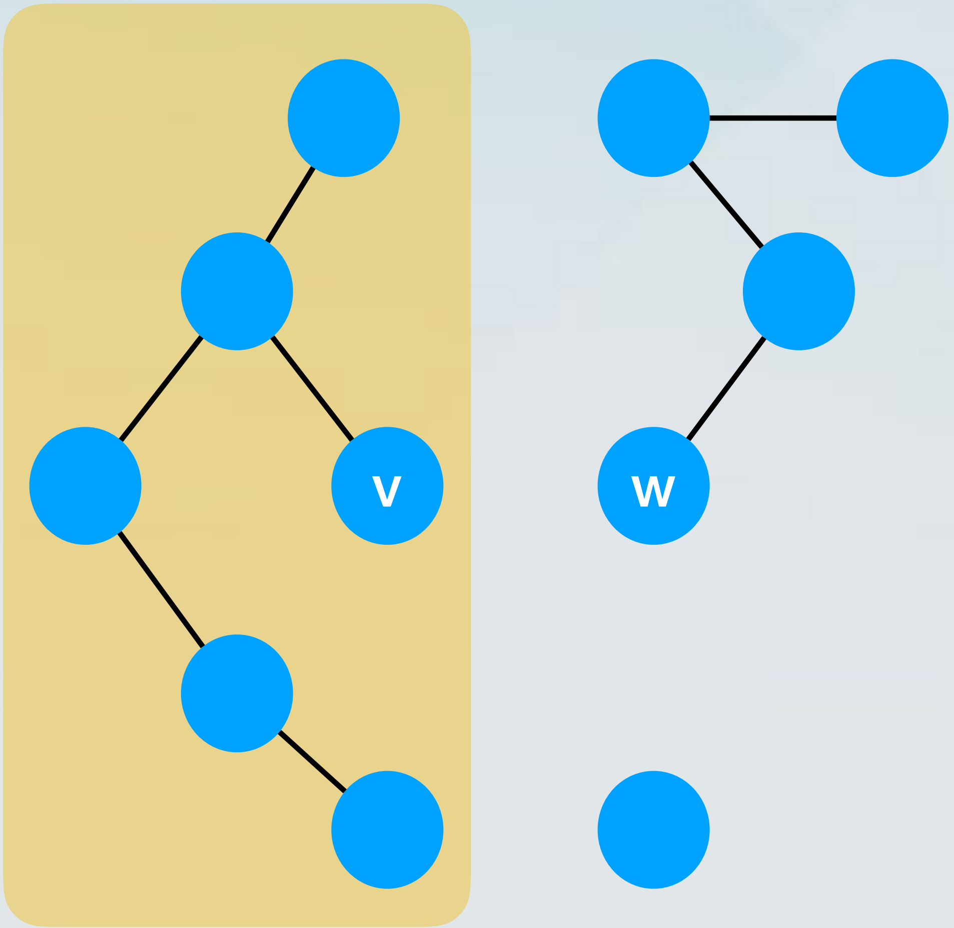
Is it feasible?

- i.e., does it always produce a spanning tree?
- The algorithm explicitly avoids cycles.
  - Output T is a forest.
- Is it a tree?
  - Is it connected?



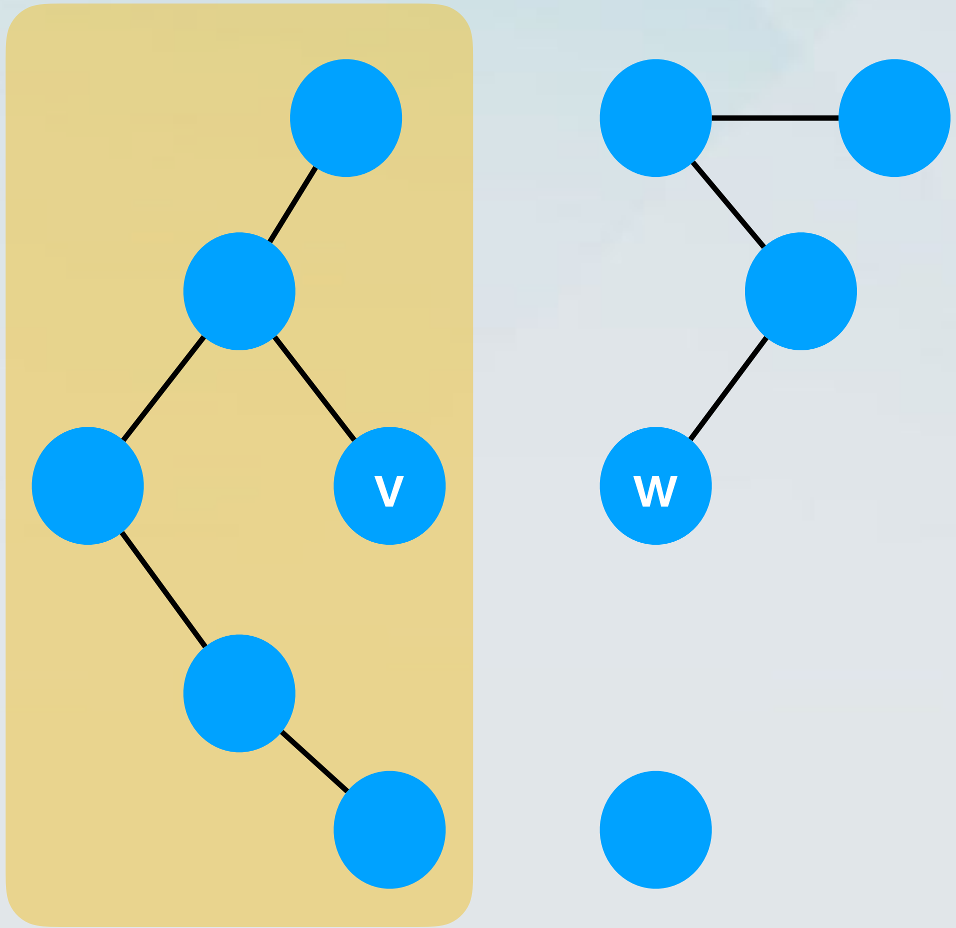
Is it feasible?

- i.e., does it always produce a spanning tree?
- The algorithm explicitly avoids cycles.
  - Output T is a forest.
- Is it a tree?
  - Is it connected?
  - G is connected.



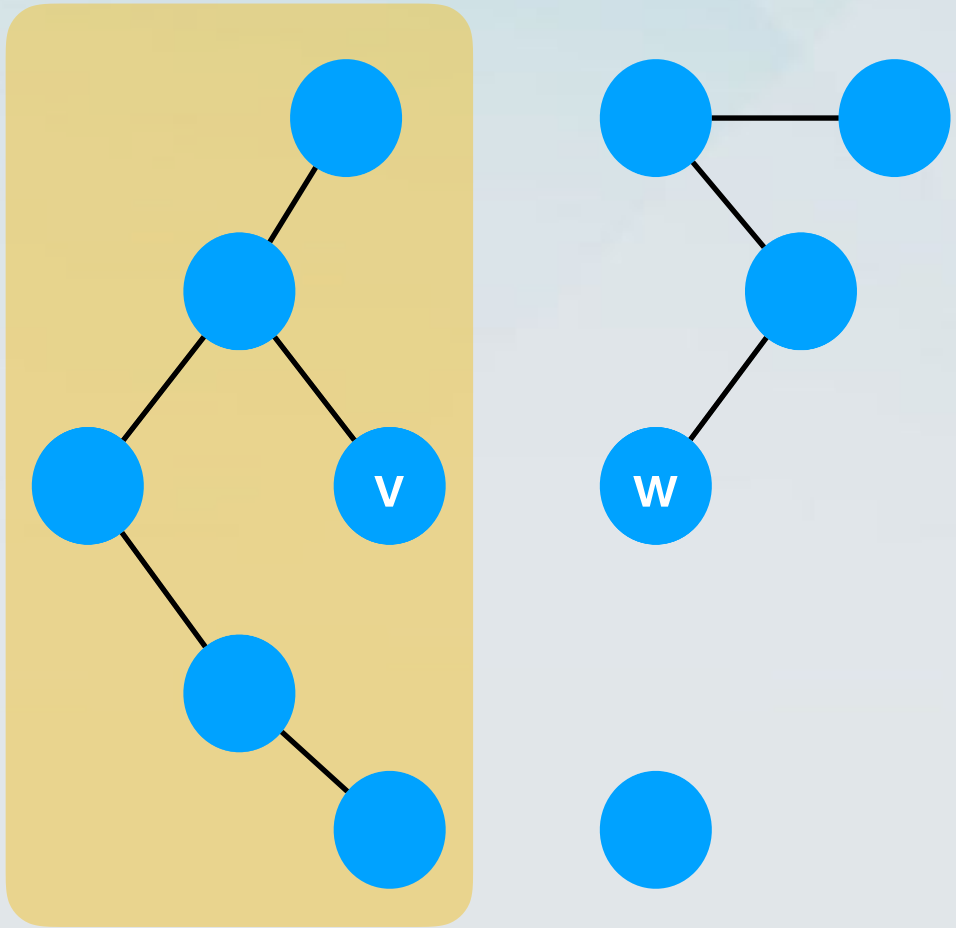
Is it feasible?

- i.e., does it always produce a spanning tree?
- The algorithm explicitly avoids cycles.
  - Output T is a forest.
- Is it a tree?
  - Is it connected?
  - G is connected.
  - Suppose by contradiction that T was not connected.



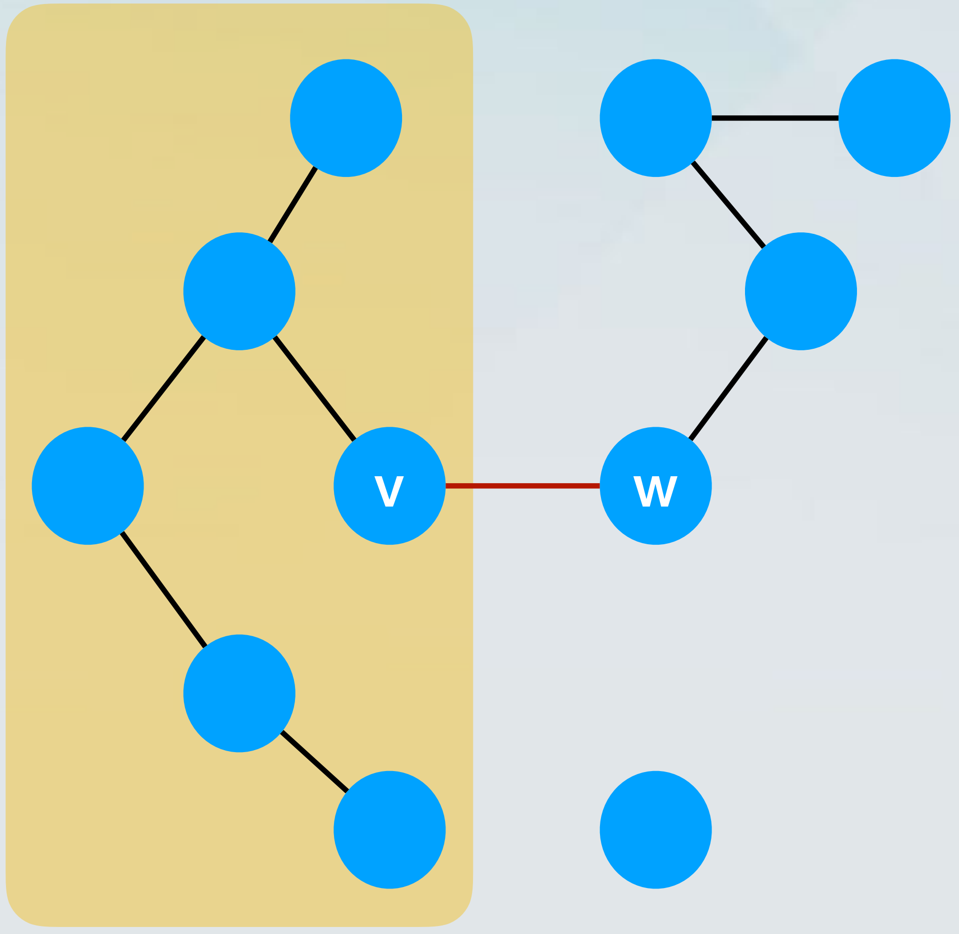
Is it feasible?

- i.e., does it always produce a spanning tree?
- The algorithm explicitly avoids cycles.
  - Output T is a forest.
- Is it a tree?
  - Is it connected?
  - G is connected.
  - Suppose by contradiction that T was not connected.
  - The algorithm would have added an edge crossing the two components.



Is it feasible?

- i.e., does it always produce a spanning tree?
- The algorithm explicitly avoids cycles.
    - Output T is a forest.
- Is it a tree?
    - Is it connected?
    - G is connected.
    - Suppose by contradiction that T was not connected.
    - The algorithm would have added an edge crossing the two components.



Prim’s algorithm is optimal

Prim’s algorithm is optimal

- In each iteration of the algorithm, there is a set S of nodes which are the nodes of a partial spanning tree.

Prim's algorithm is optimal

- In each iteration of the algorithm, there is a set S of nodes which are the nodes of a partial spanning tree.
- An edge is added to "expand" the partial spanning tree, which has the minimum cost.

Prim’s algorithm is optimal

- In each iteration of the algorithm, there is a set S of nodes which are the nodes of a partial spanning tree.
- An edge is added to “expand” the partial spanning tree, which has the minimum cost.
- This edge has one endpoint in S and one in V-S and has minimum cost.

Prim’s algorithm is optimal

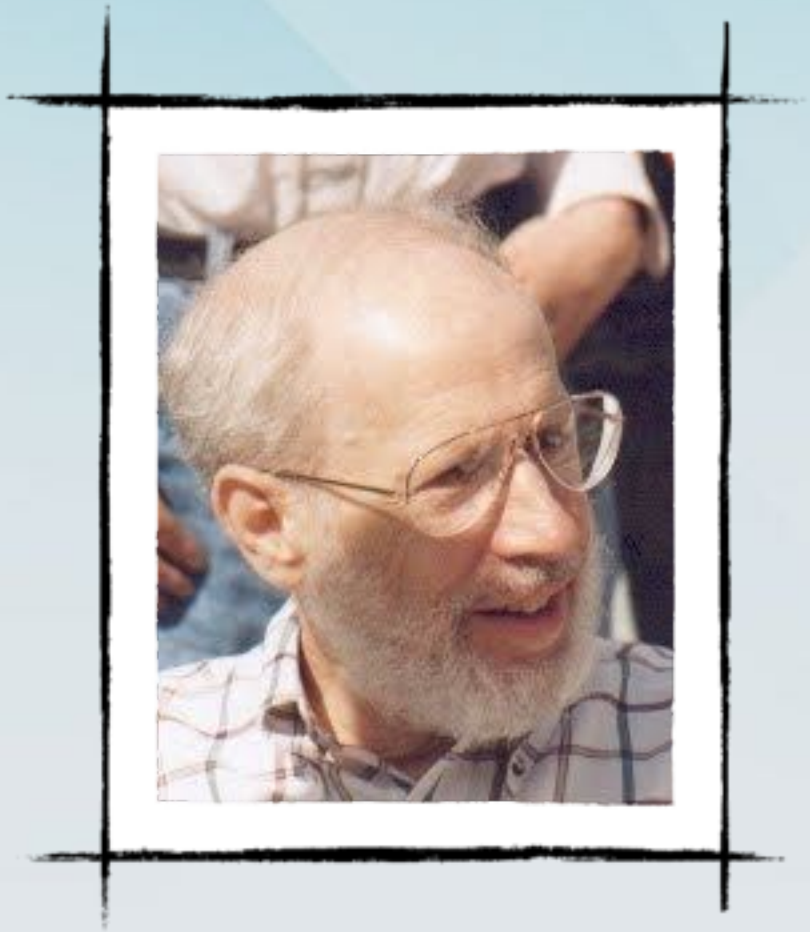
- In each iteration of the algorithm, there is a set S of nodes which are the nodes of a partial spanning tree.
- An edge is added to “expand” the partial spanning tree, which has the minimum cost.
- This edge has one endpoint in S and one in V-S and has minimum cost.
- So it must be part of every minimum spanning tree.

Greedy Approach #2

Greedy Approach #2

- Start with the full graph $G=(V, E)$.
- Delete an edge from G.
  - Which one?
  - The one with the maximum cost c_e.
- We continue like this.
  - Do we always remove the considered edge e from G?
  - As long as we don't disconnect the graph.

Reverse-Delete Algorithm

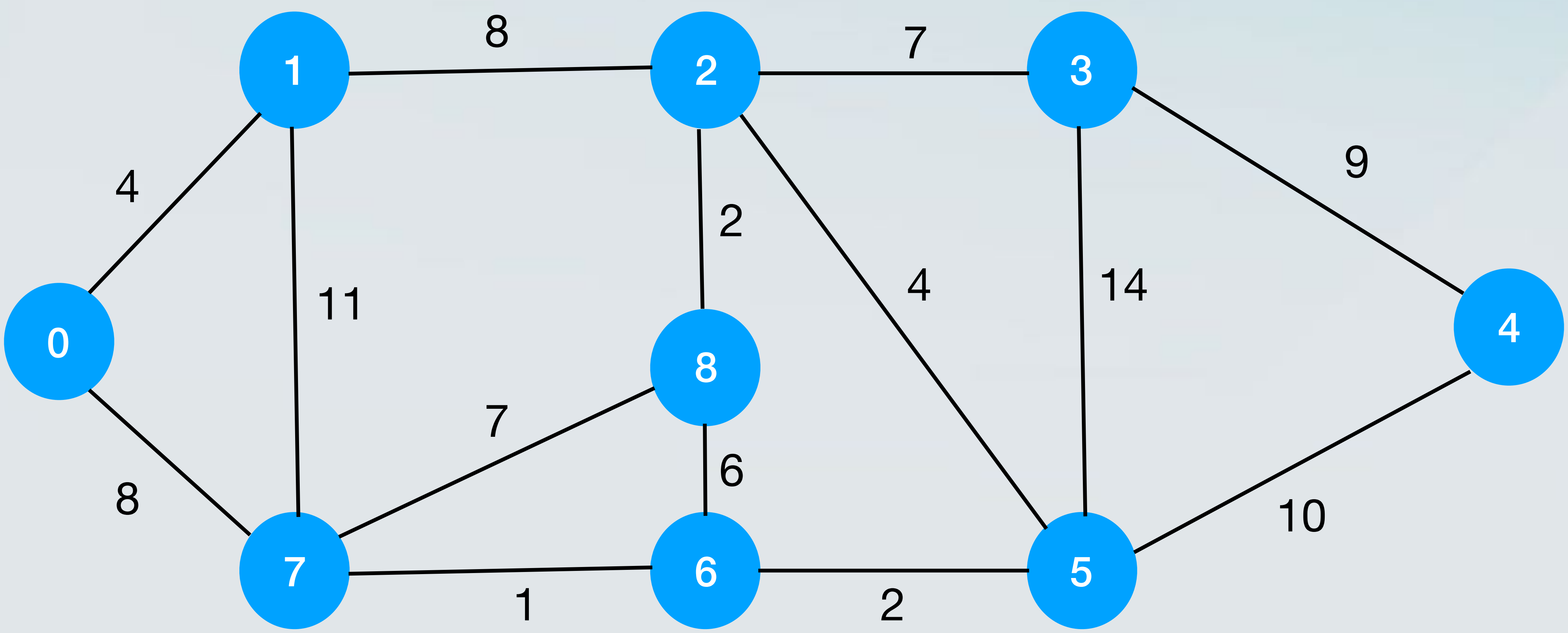


- Start with the full graph G=(V, E).
- Delete an edge from G.
    - Which one?
    - The one with the maximum cost c_e.
- We continue like this.
    - Do we always remove the considered edge e from G?
    - As long as we don't disconnect the graph.

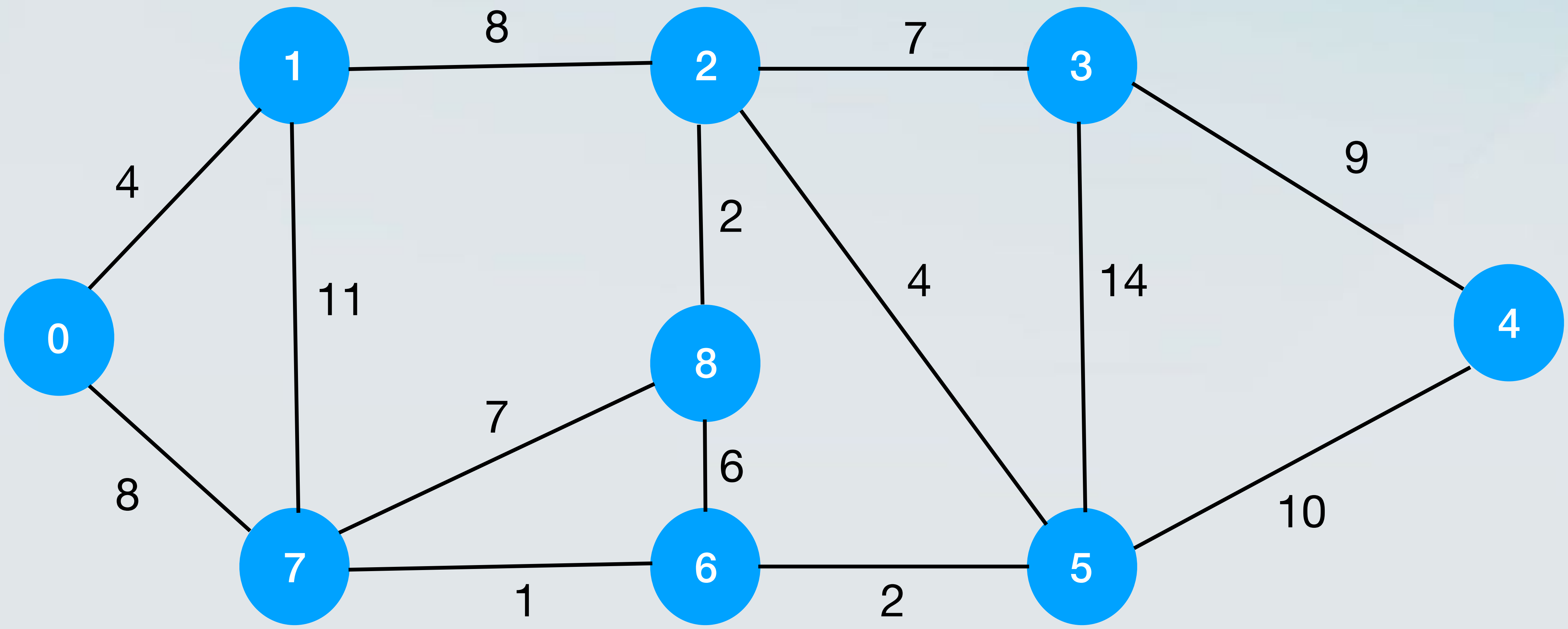
Reverse-Delete Algorithm

- Start with the full graph G=(V, E).
- Delete an edge from G.
  - Which one?
  - The one with the maximum cost c_e.
- We continue like this.
  - Do we always remove the considered edge e from G?
  - As long as we don't disconnect the graph.

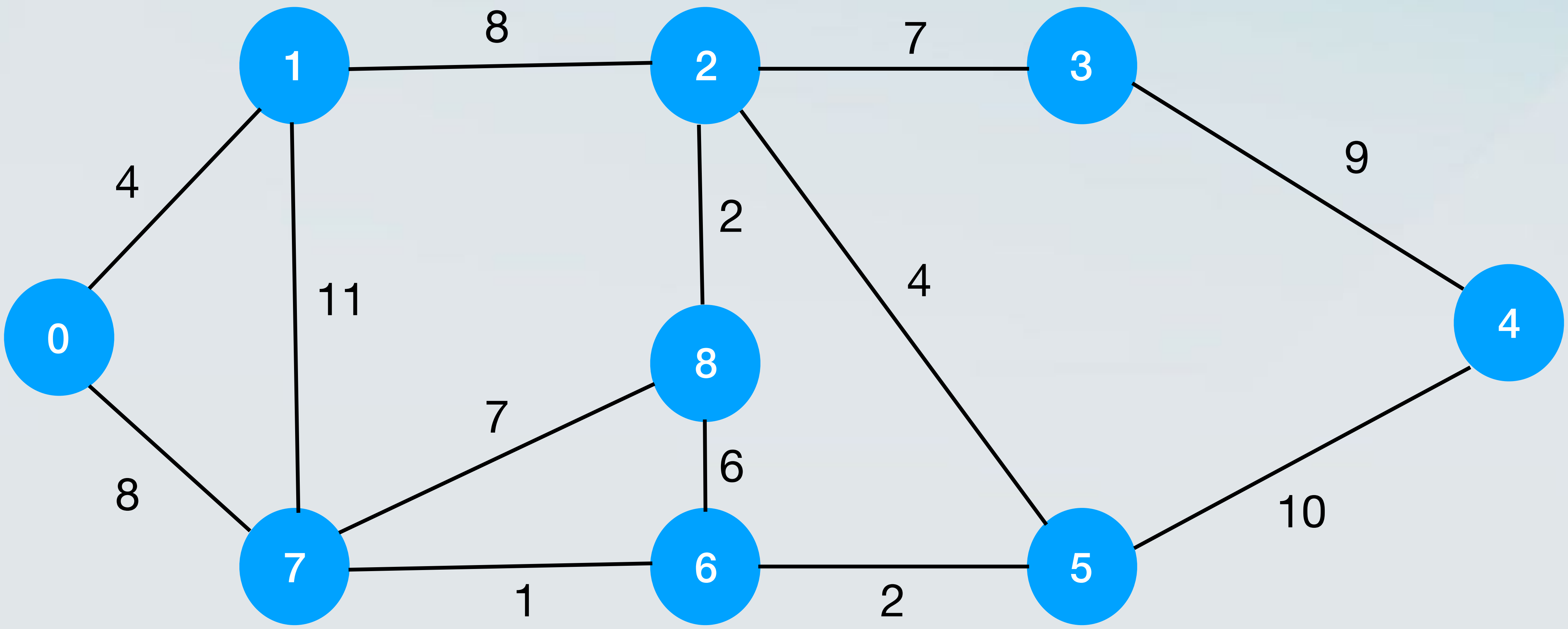
Example



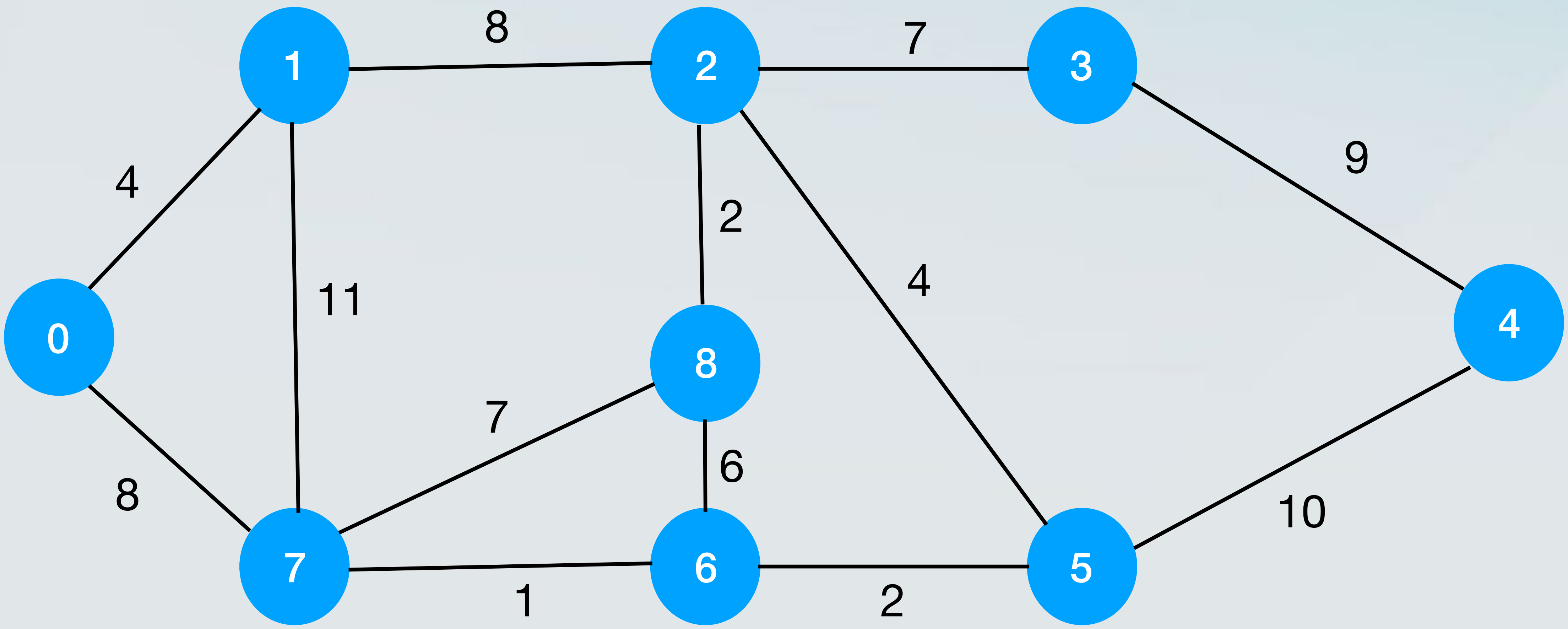
Example



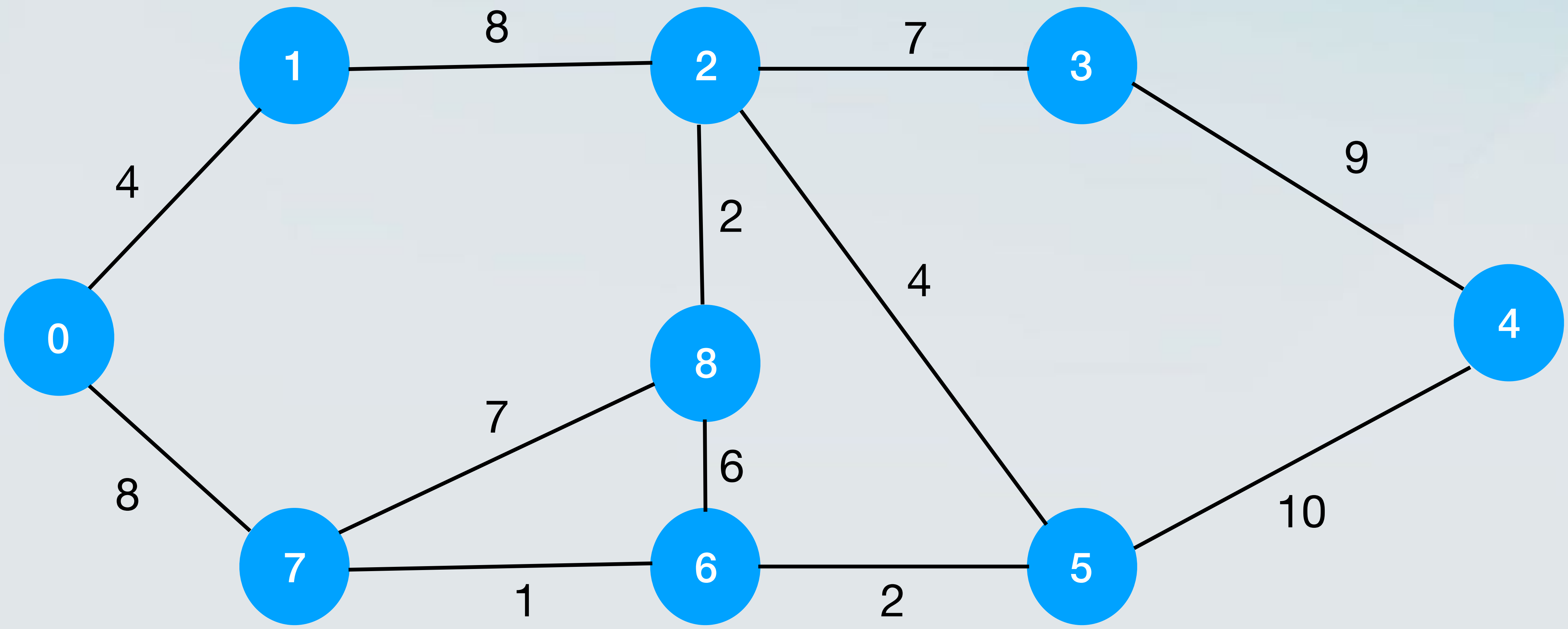
Example



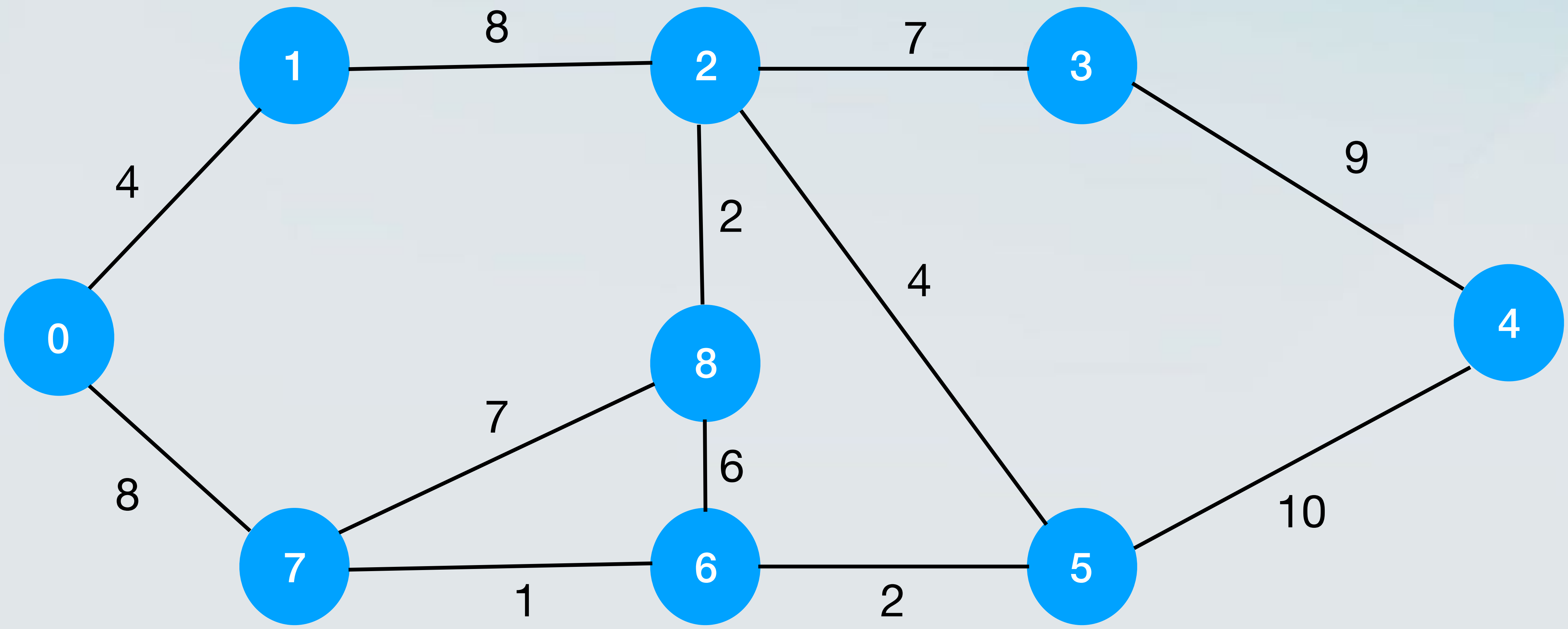
Example



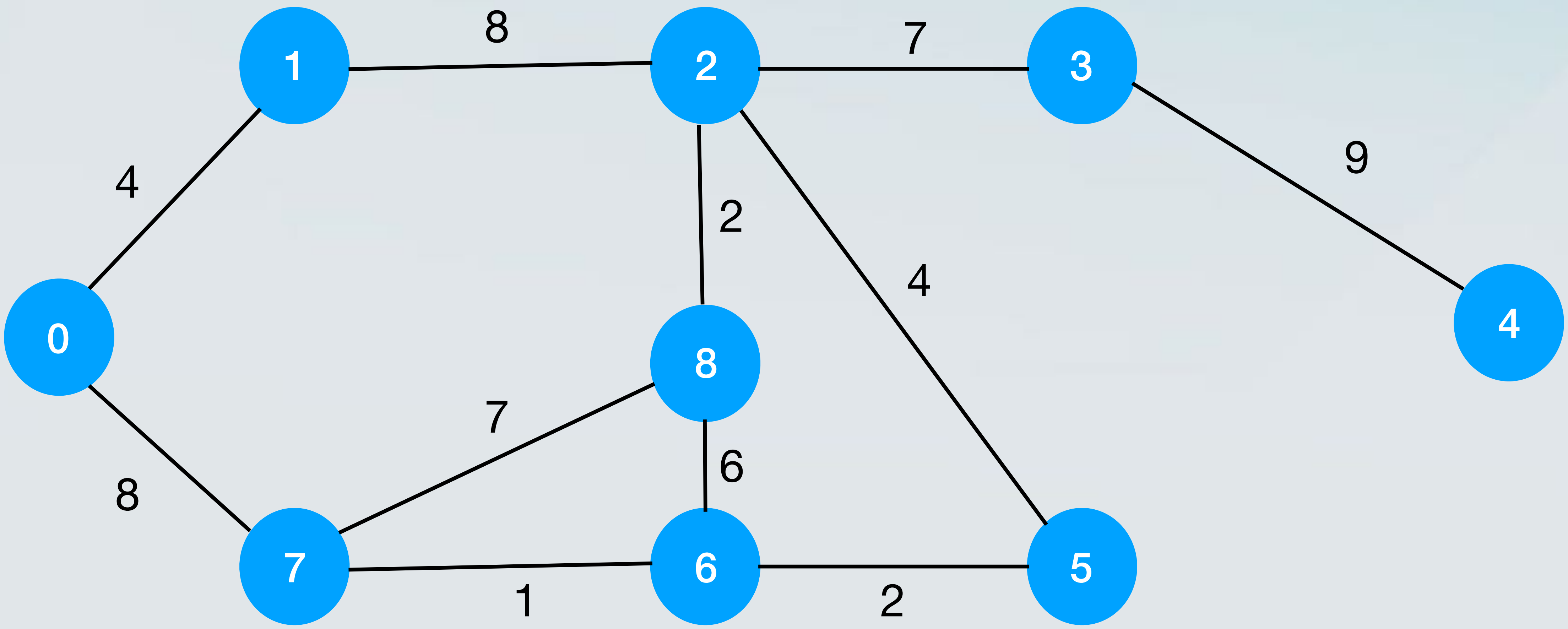
Example



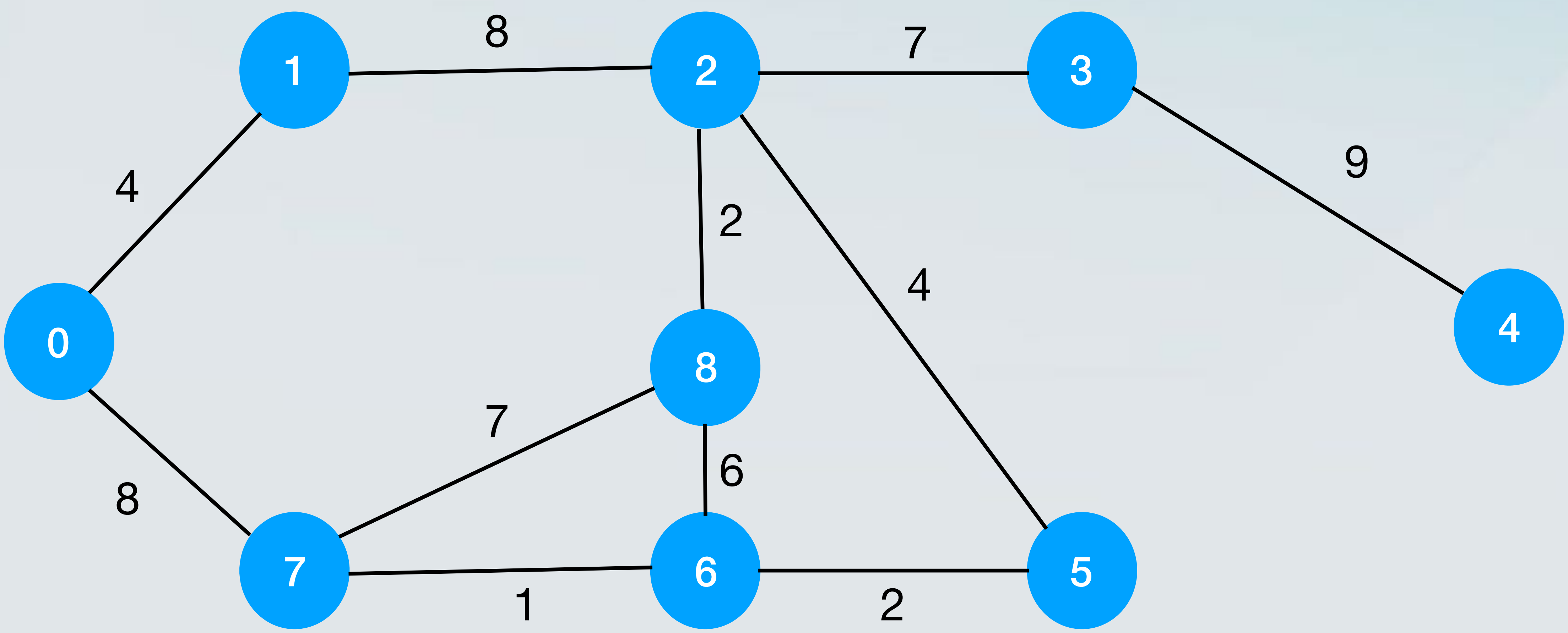
Example



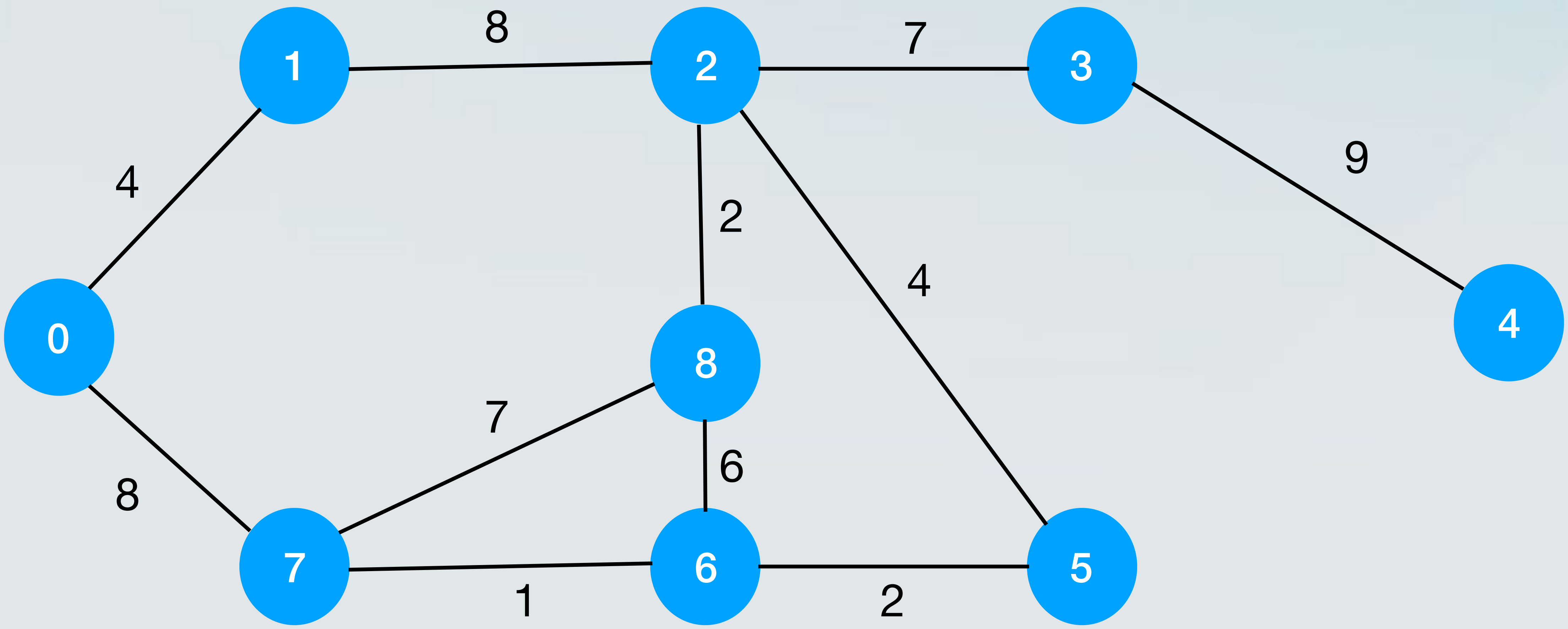
Example



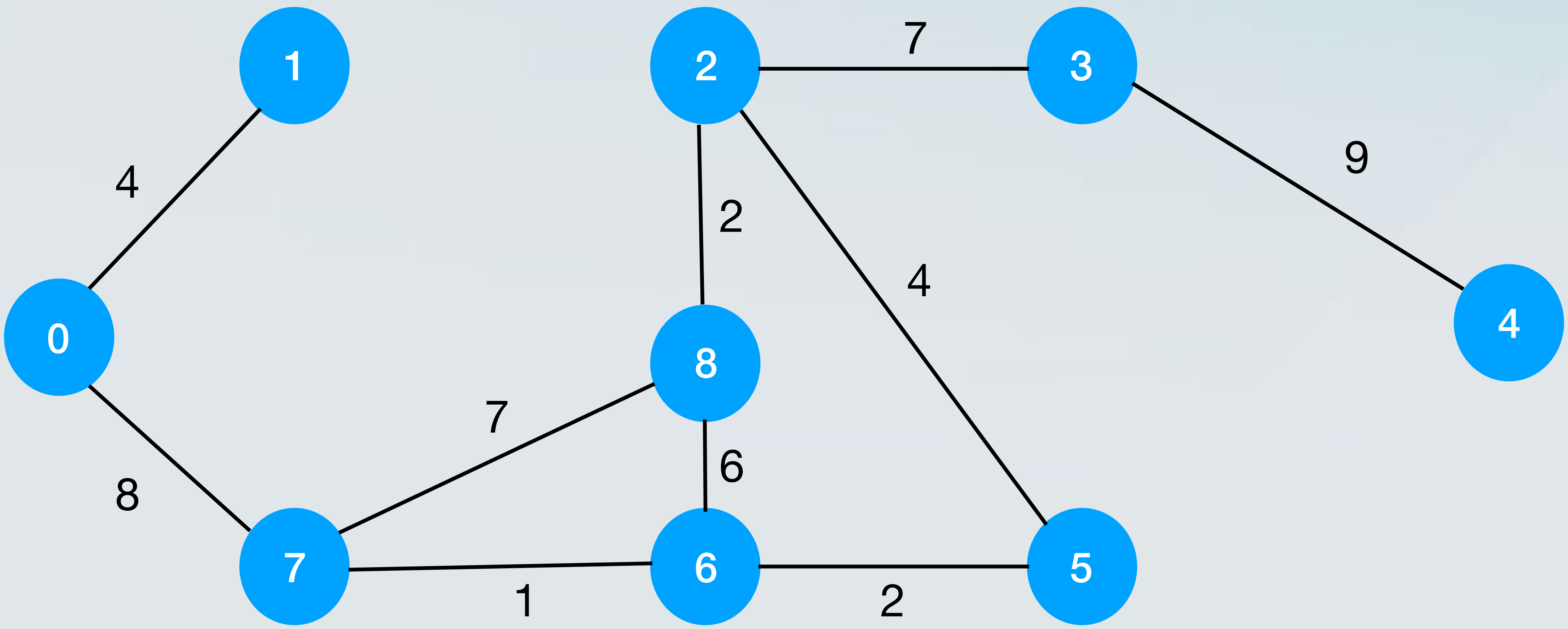
Example



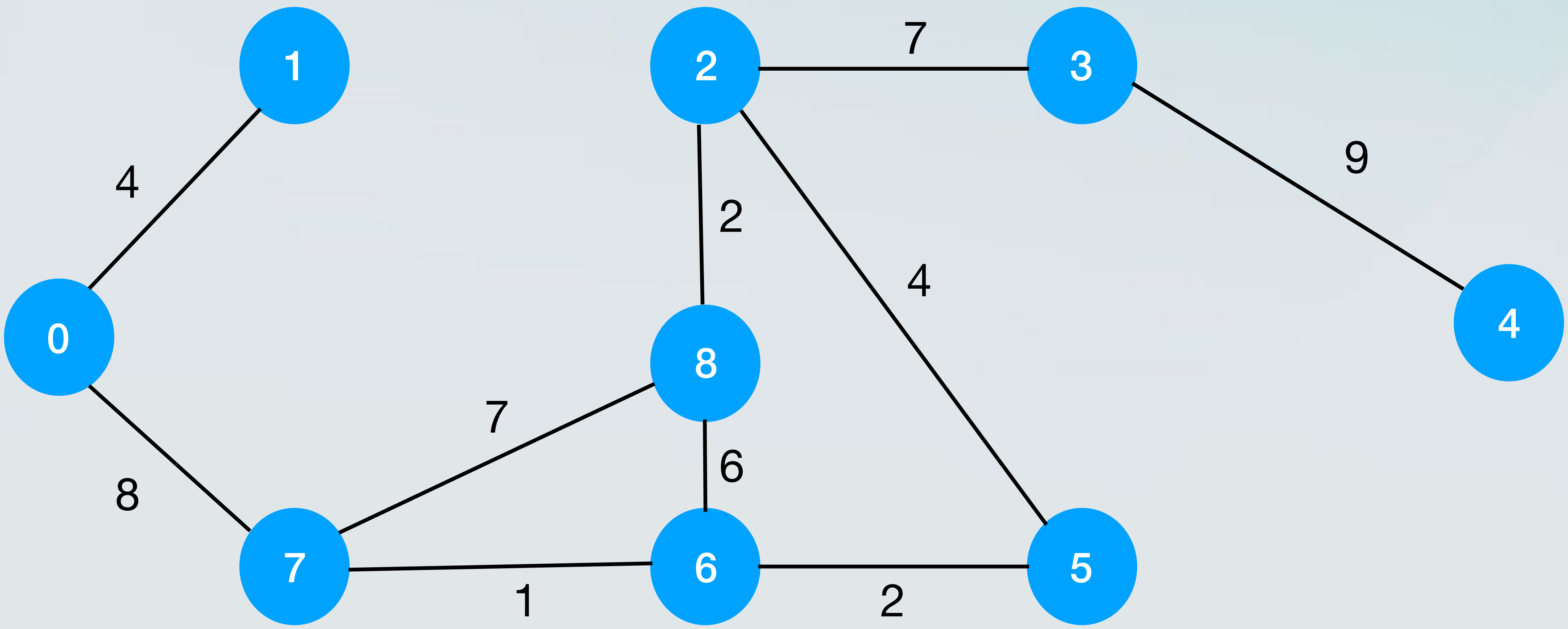
Example



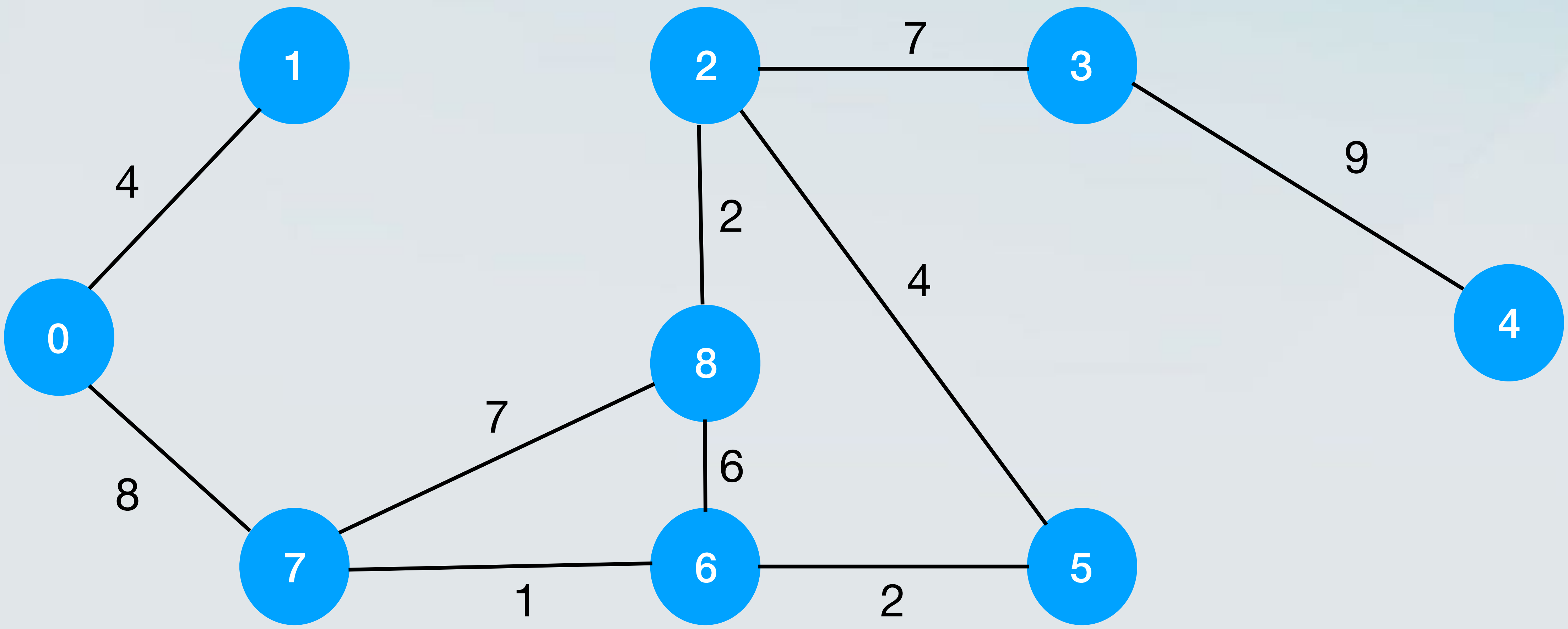
Example



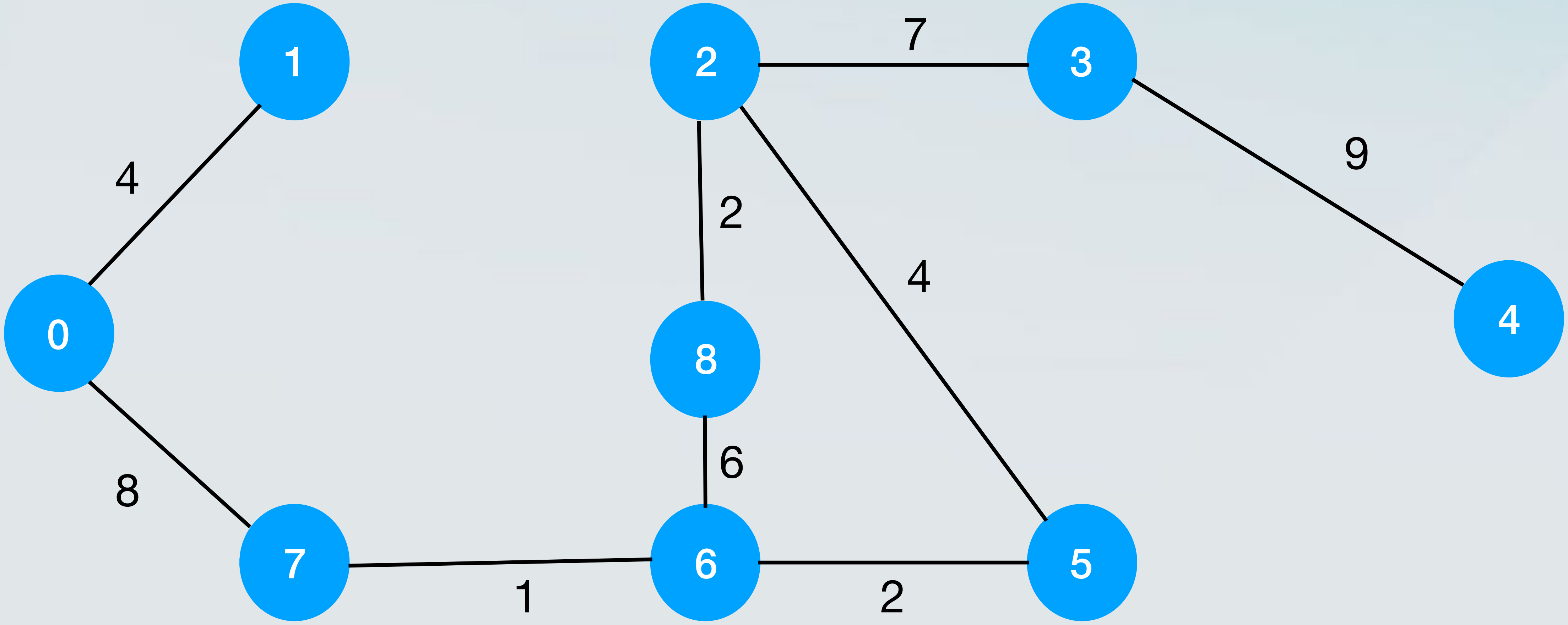
Example



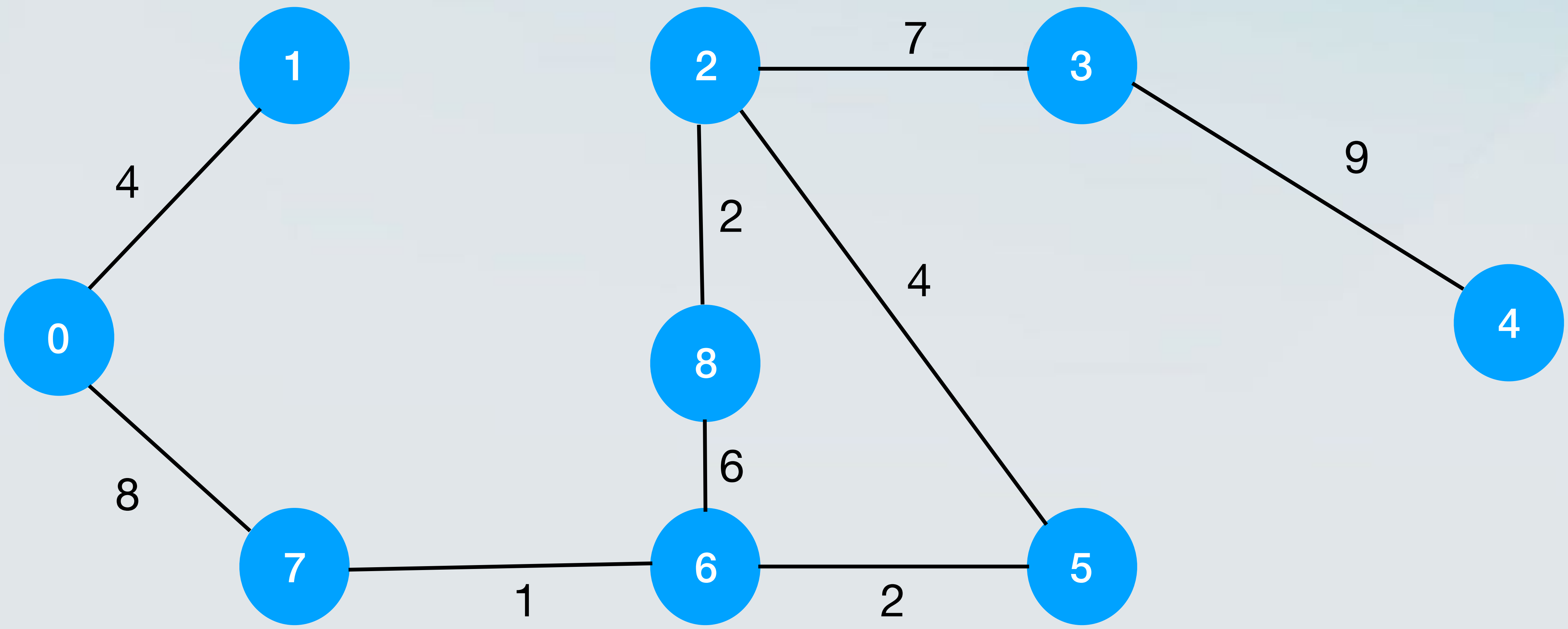
Example



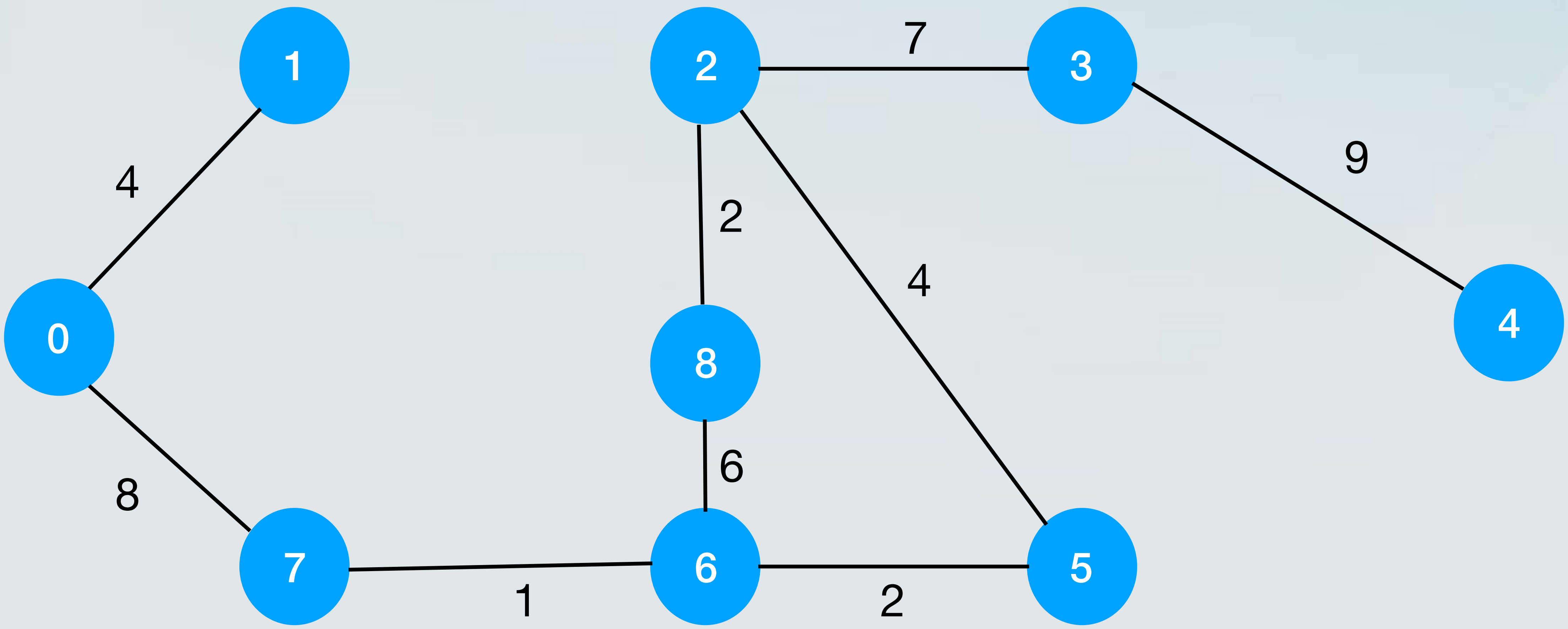
Example



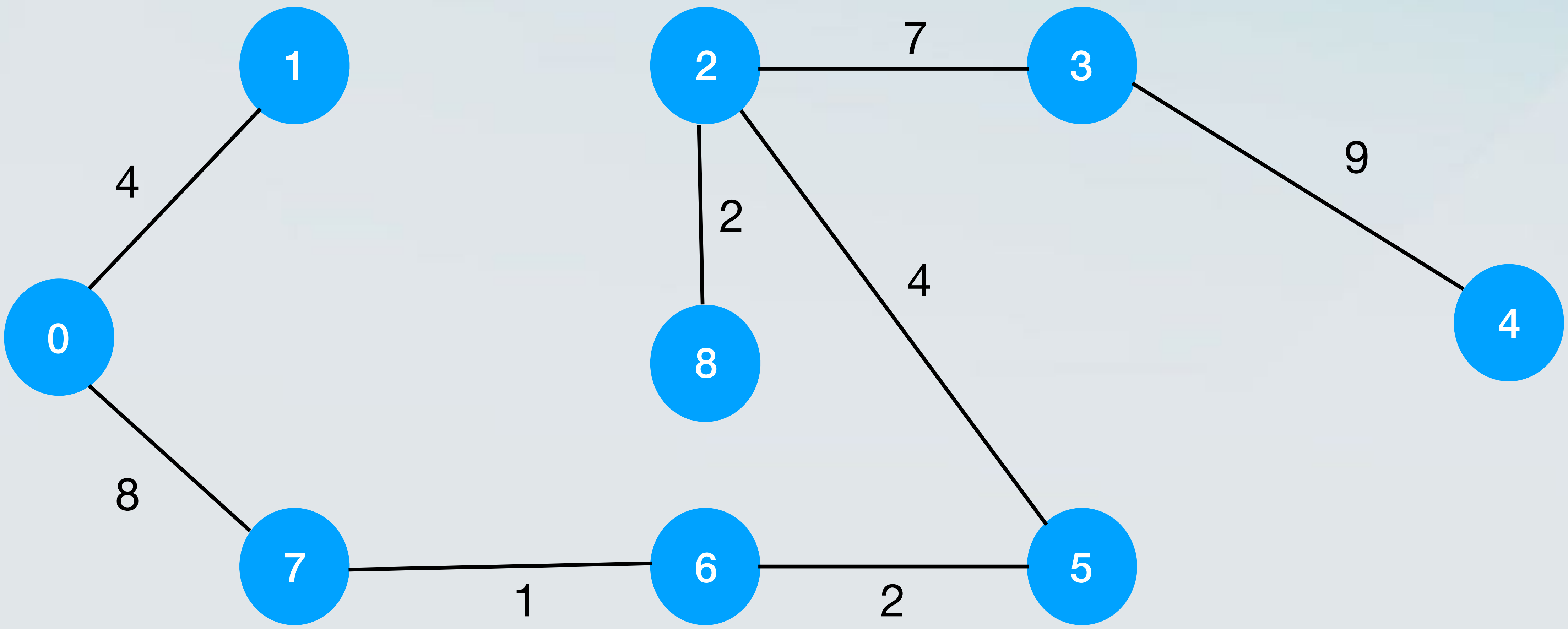
Example



Example



Example



The cycle property

The cycle property

- Assume that all edge costs are distinct.

The cycle property

- Assume that all edge costs are distinct.
- Let C be any cycle of G.

The cycle property

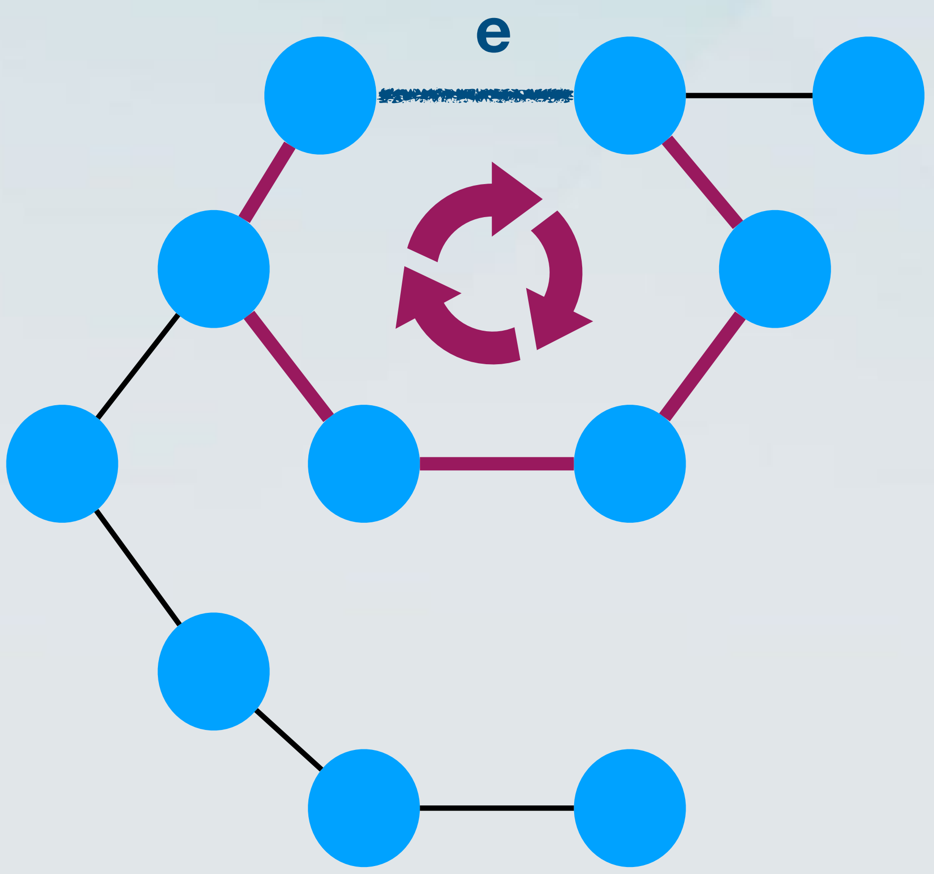
- Assume that all edge costs are distinct.
- Let C be any cycle of G.
- Let e=(w,v) be the maximum cost edge of C.

The cycle property

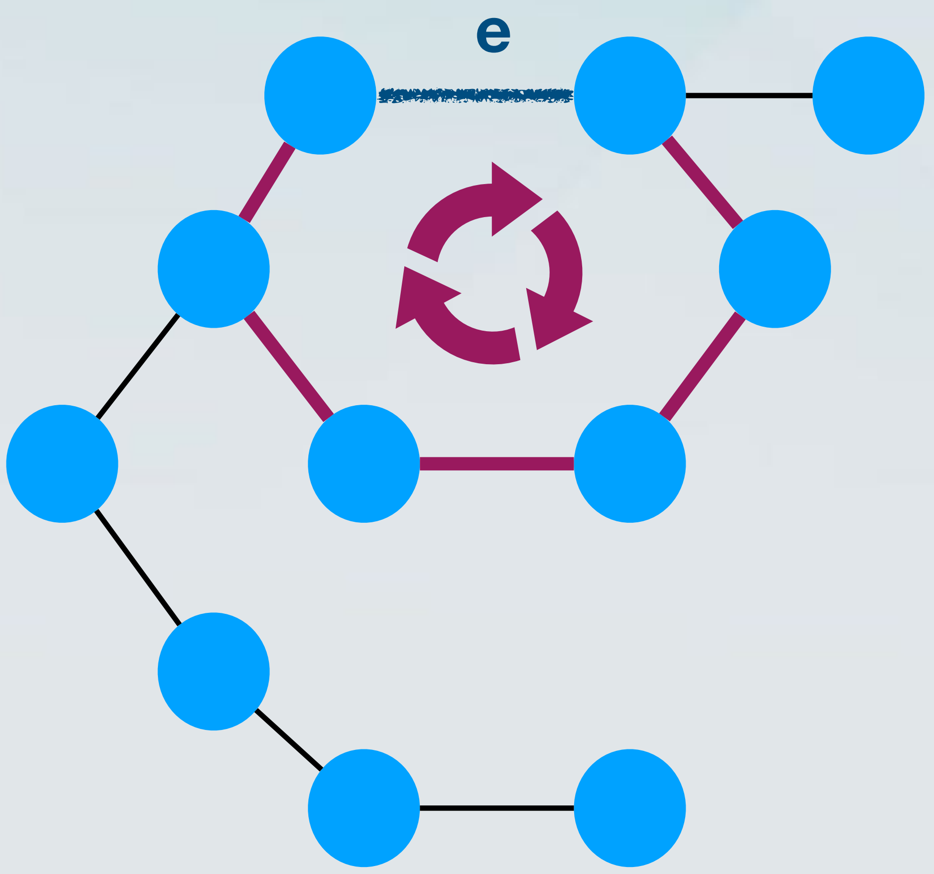
- Assume that all edge costs are distinct.
- Let C be any cycle of G.
- Let e=(w,v) be the maximum cost edge of C.
- Then e is not contained in any minimum spanning tree of G.

The cycle property

- Assume that all edge costs are distinct.
- Let C be any cycle of G.
- Let e=(w,v) be the maximum cost edge of C.
- Then e is not contained in any minimum spanning tree of G.

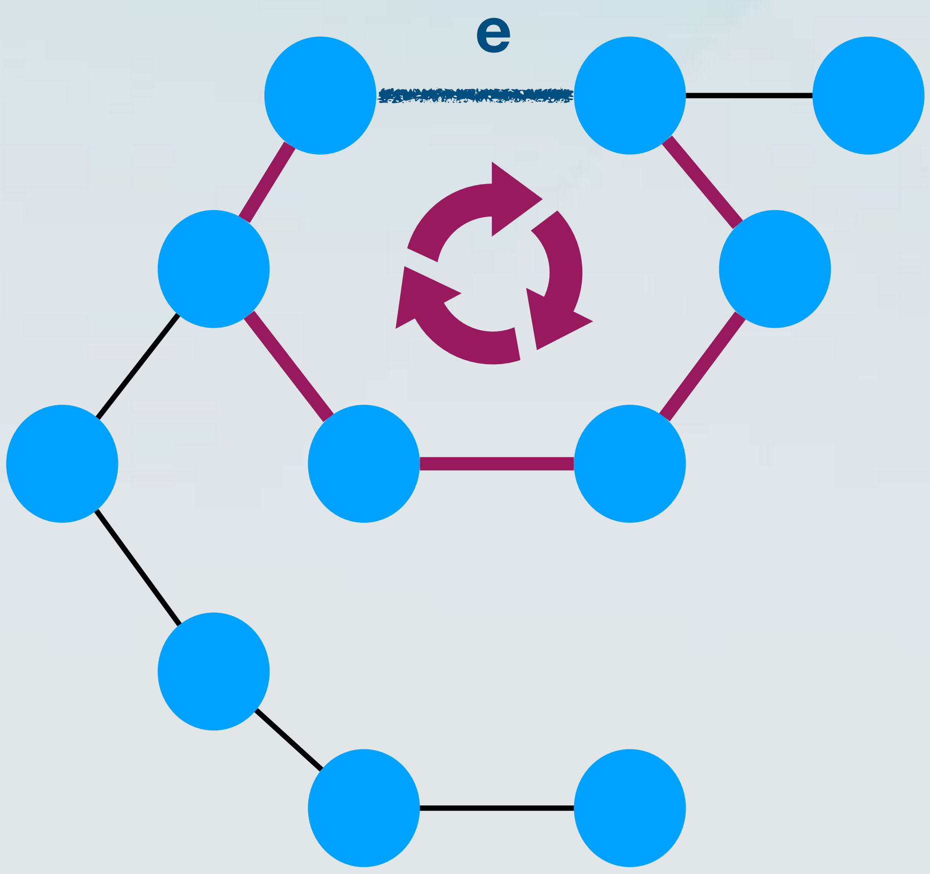


The cycle property



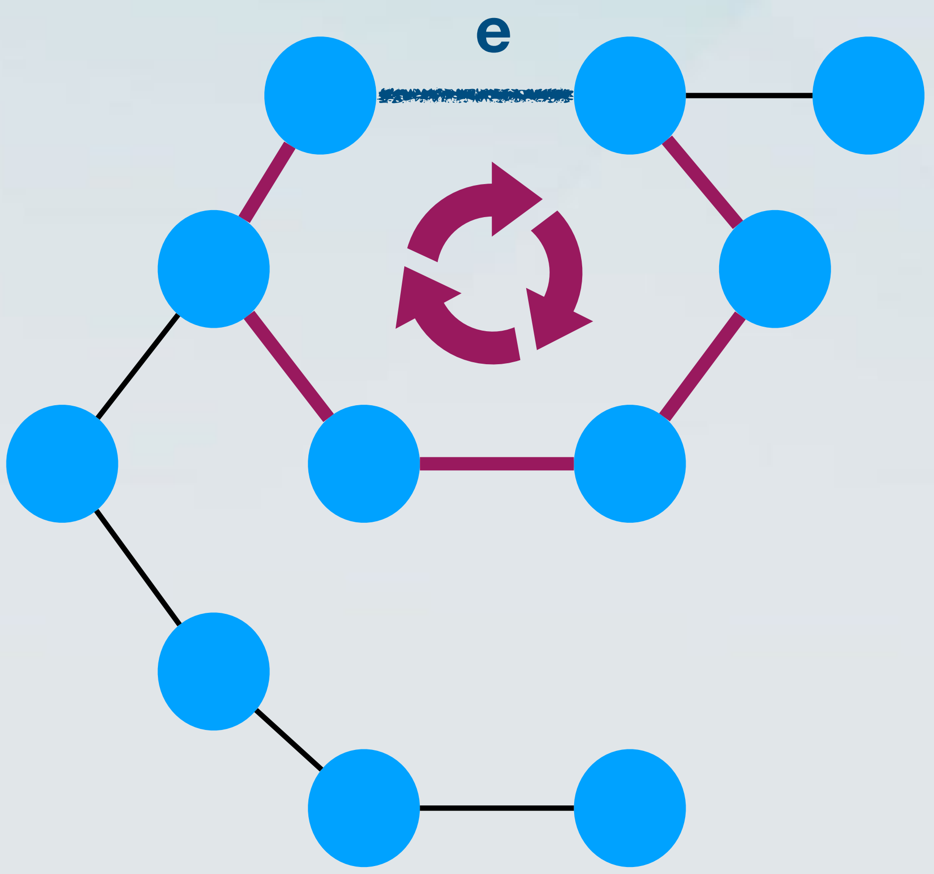
The cycle property

- Let T be a spanning tree that contains e.



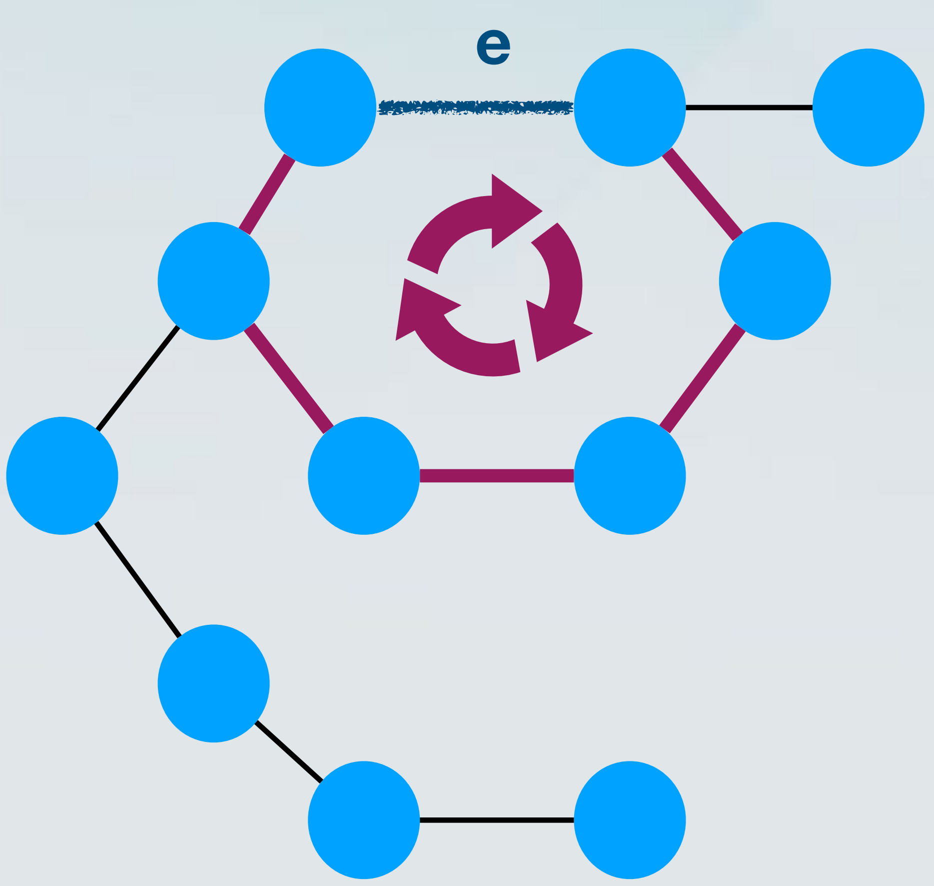
The cycle property

- Let T be a spanning tree that contains e.
  - We will show that it does not have minimum cost.



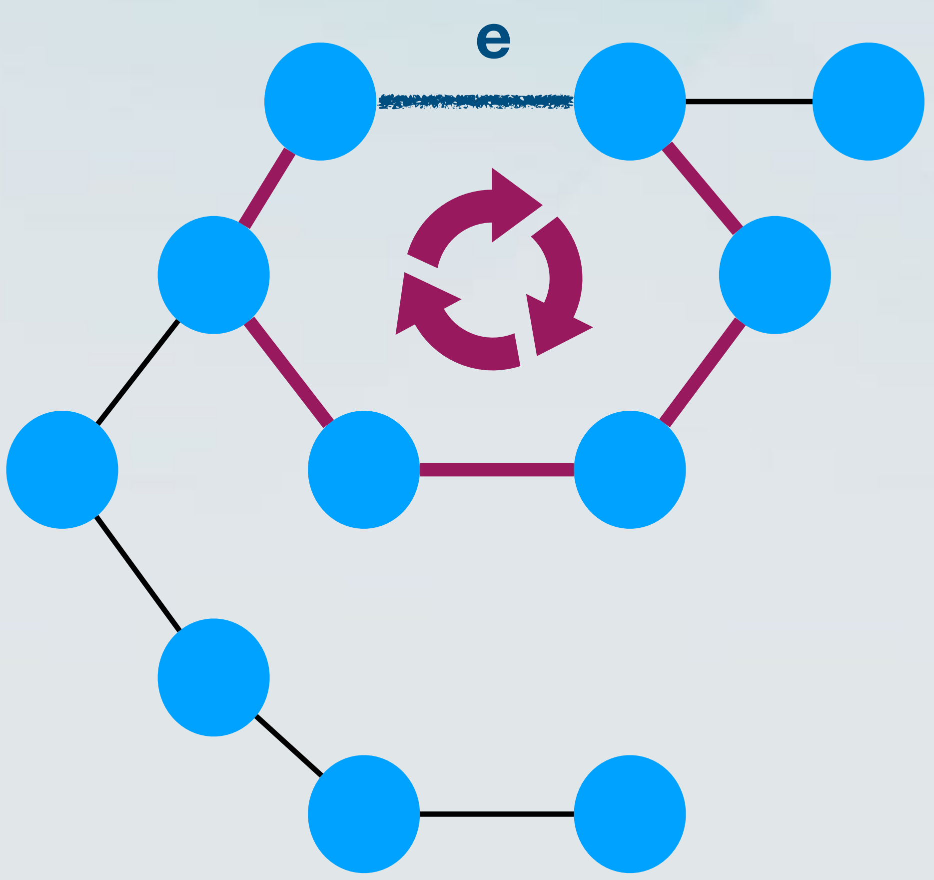
The cycle property

- Let T be a spanning tree that contains e.
  - We will show that it does not have minimum cost.
- We will substitute e with another edge e’, resulting in a cheaper spanning tree.

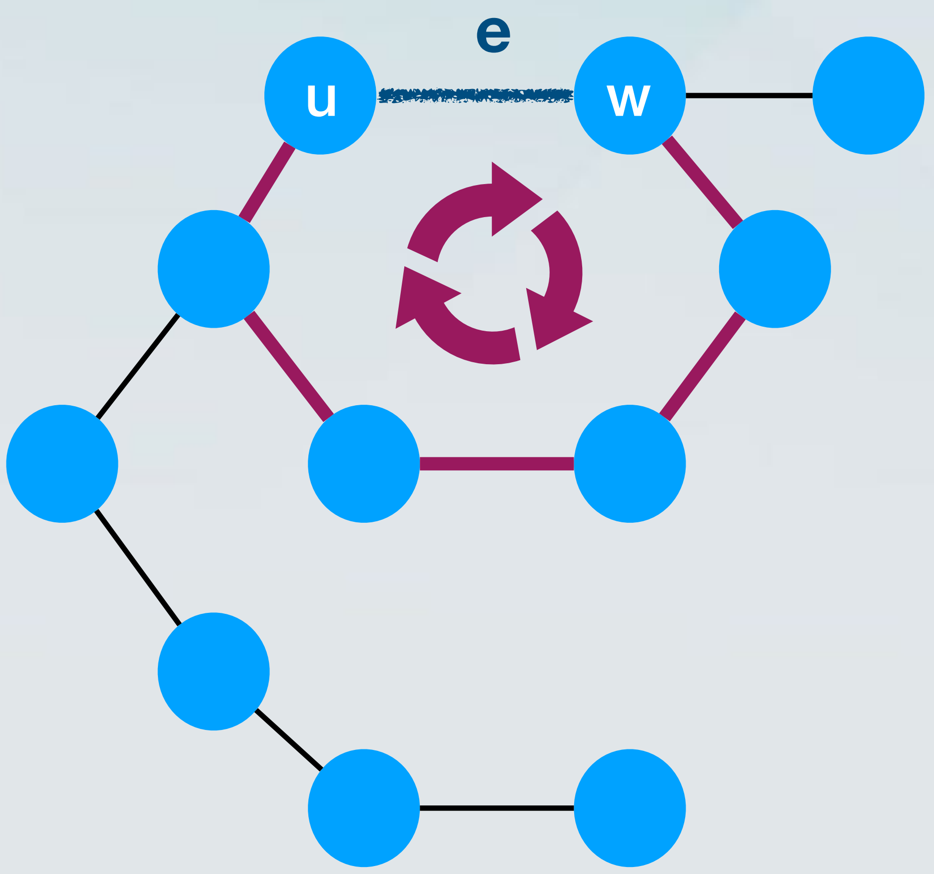


The cycle property

- Let T be a spanning tree that contains e.
  - We will show that it does not have minimum cost.
- We will substitute e with another edge e', resulting in a cheaper spanning tree.
- How to find this edge e'?

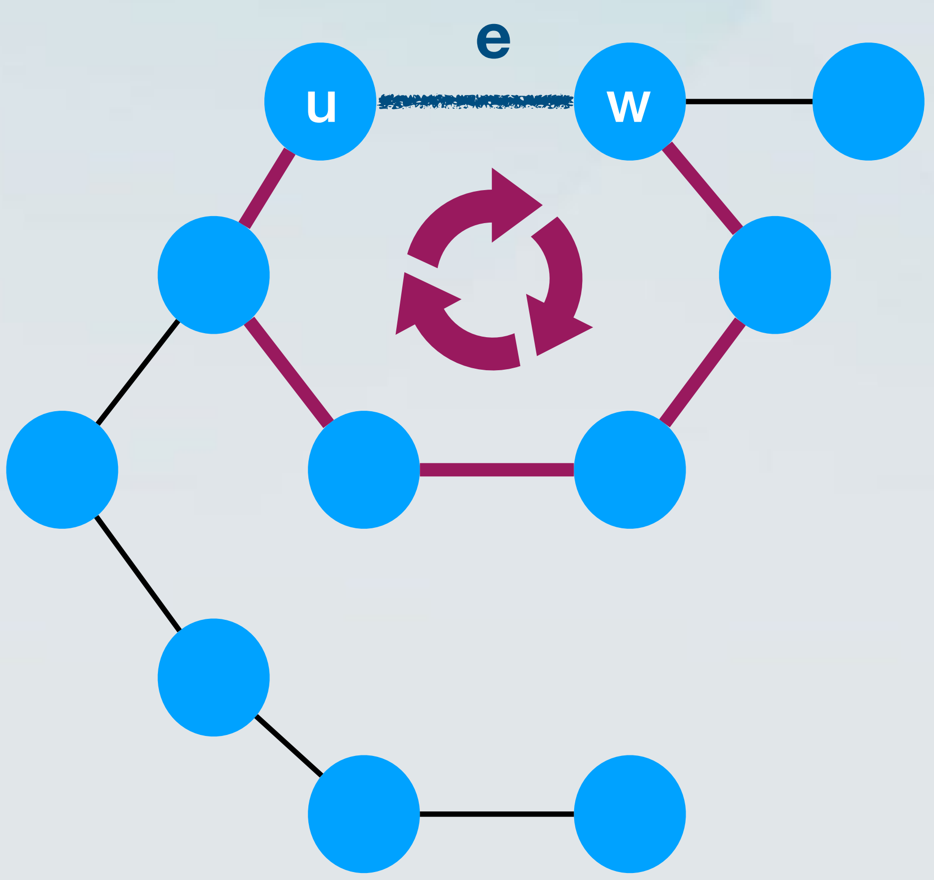


The cycle property



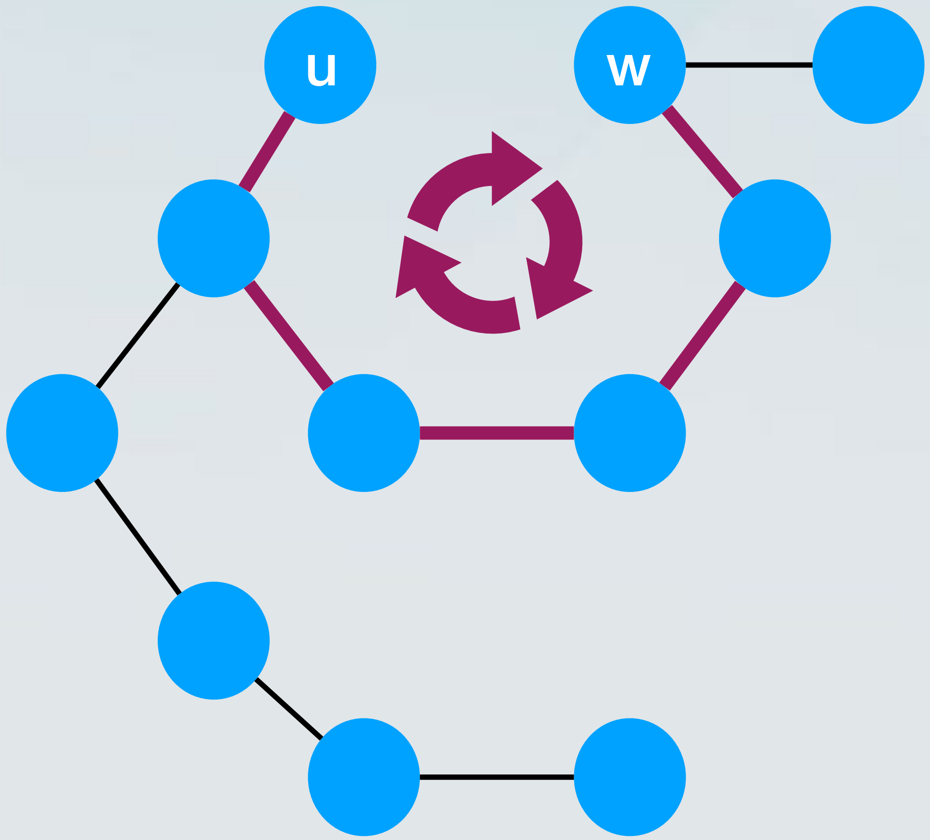
The cycle property

- We delete e from T.



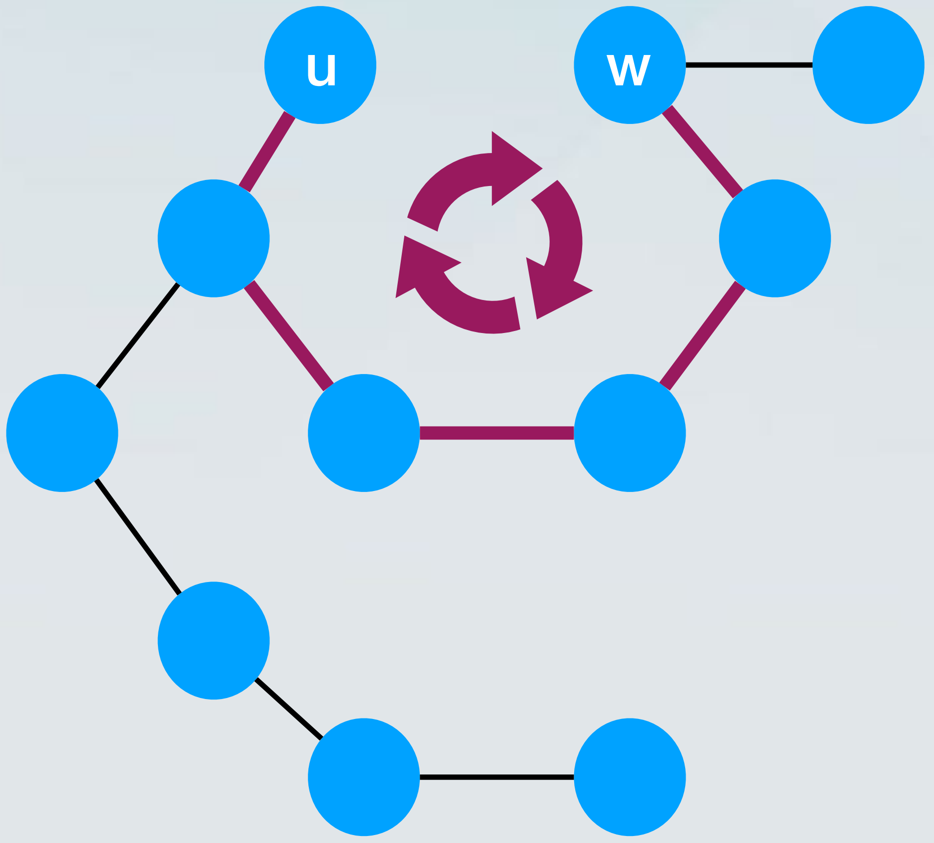
The cycle property

- We delete e from T.



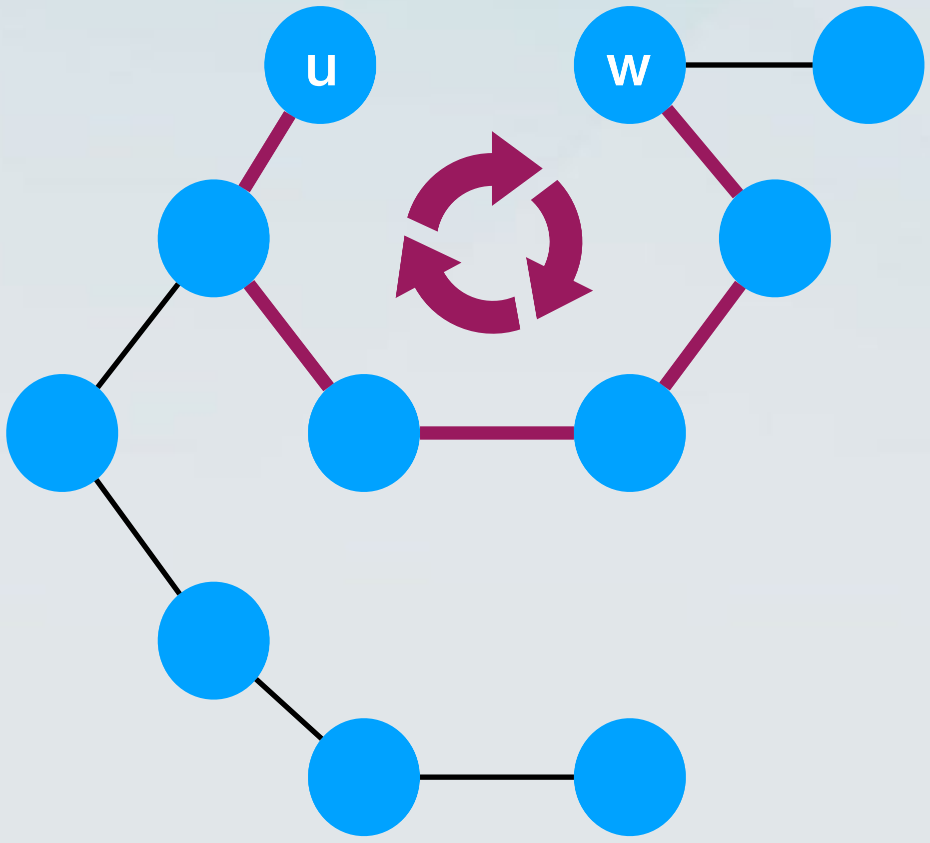
The cycle property

- We delete e from T.
- This partitions the nodes into



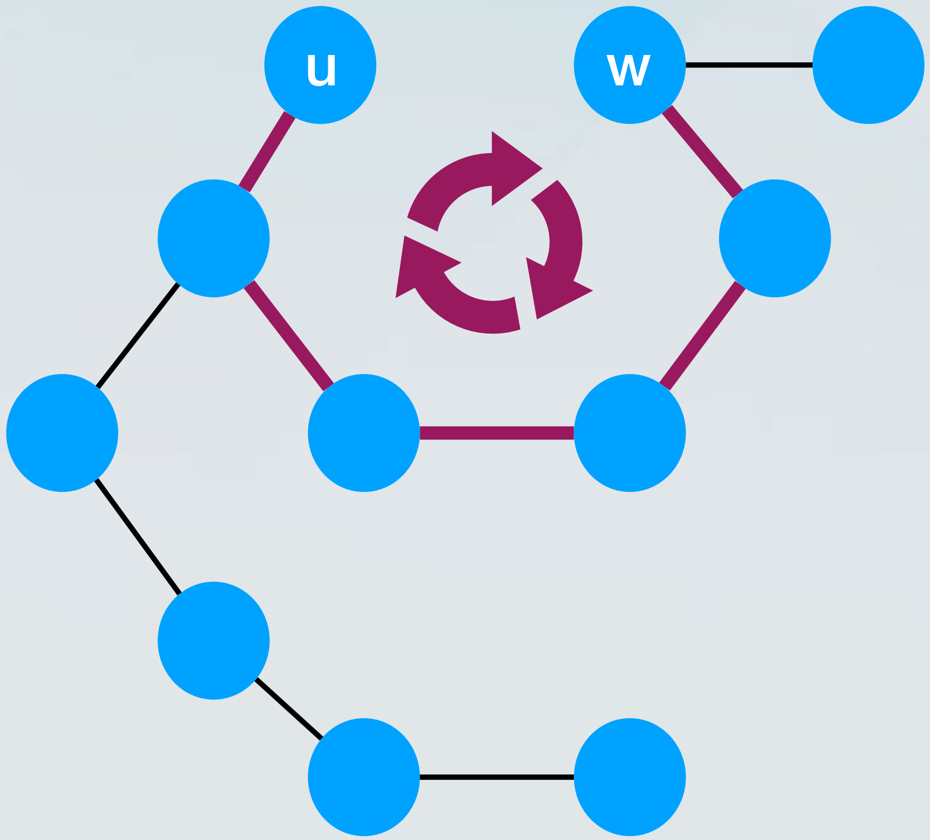
The cycle property

- We delete e from T.
- This partitions the nodes into
  - S (containing u).



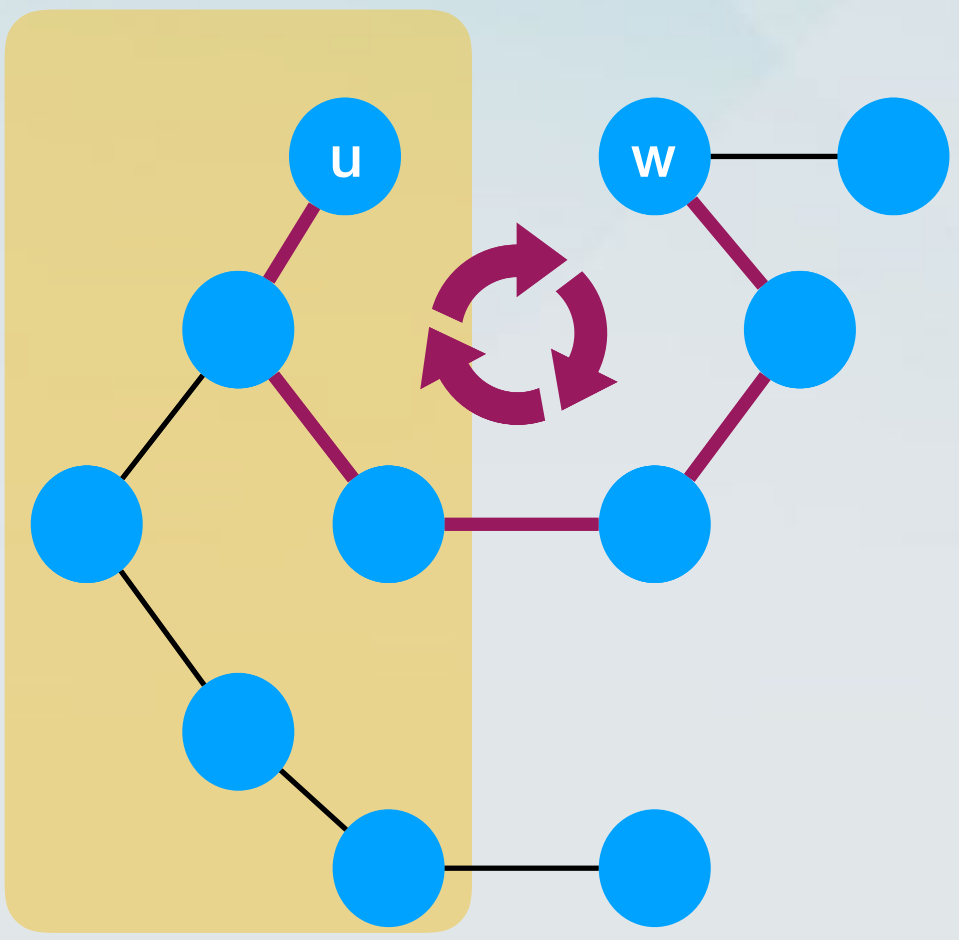
The cycle property

- We delete e from T.
- This partitions the nodes into
  - S (containing u).
  - V - S (containing w).



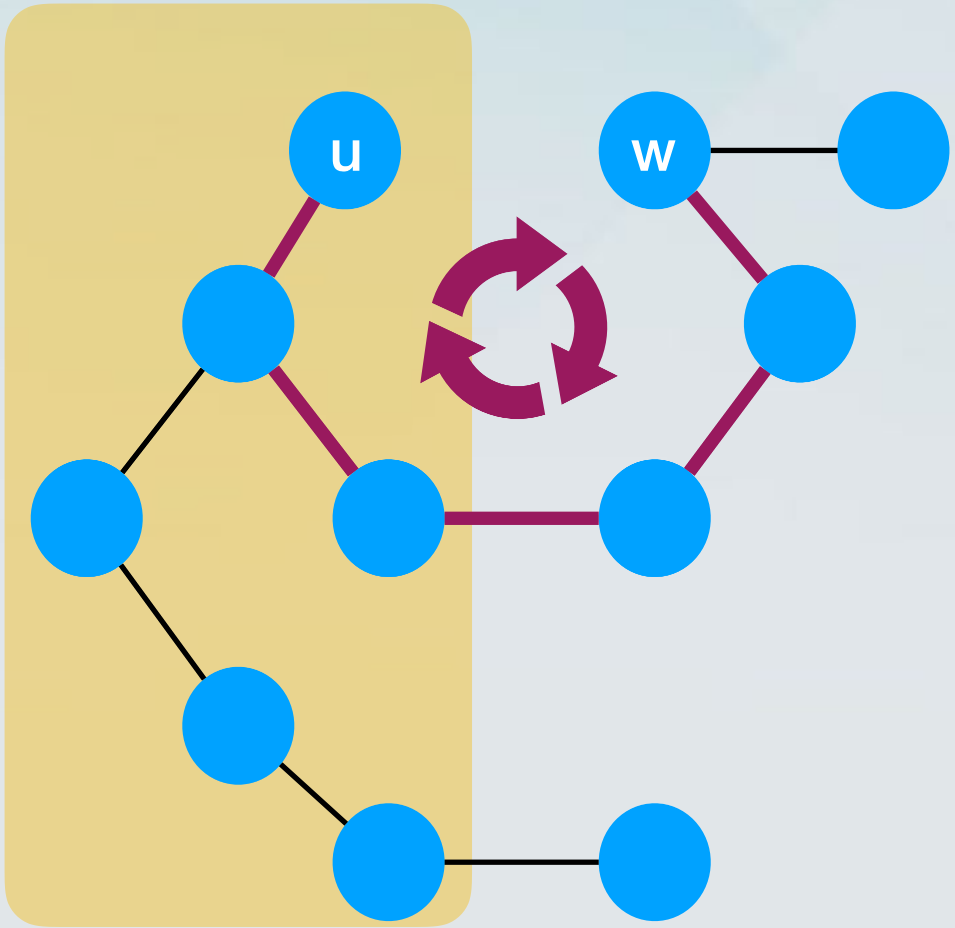
The cycle property

- We delete e from T.
- This partitions the nodes into
  - S (containing u).
  - V - S (containing w).



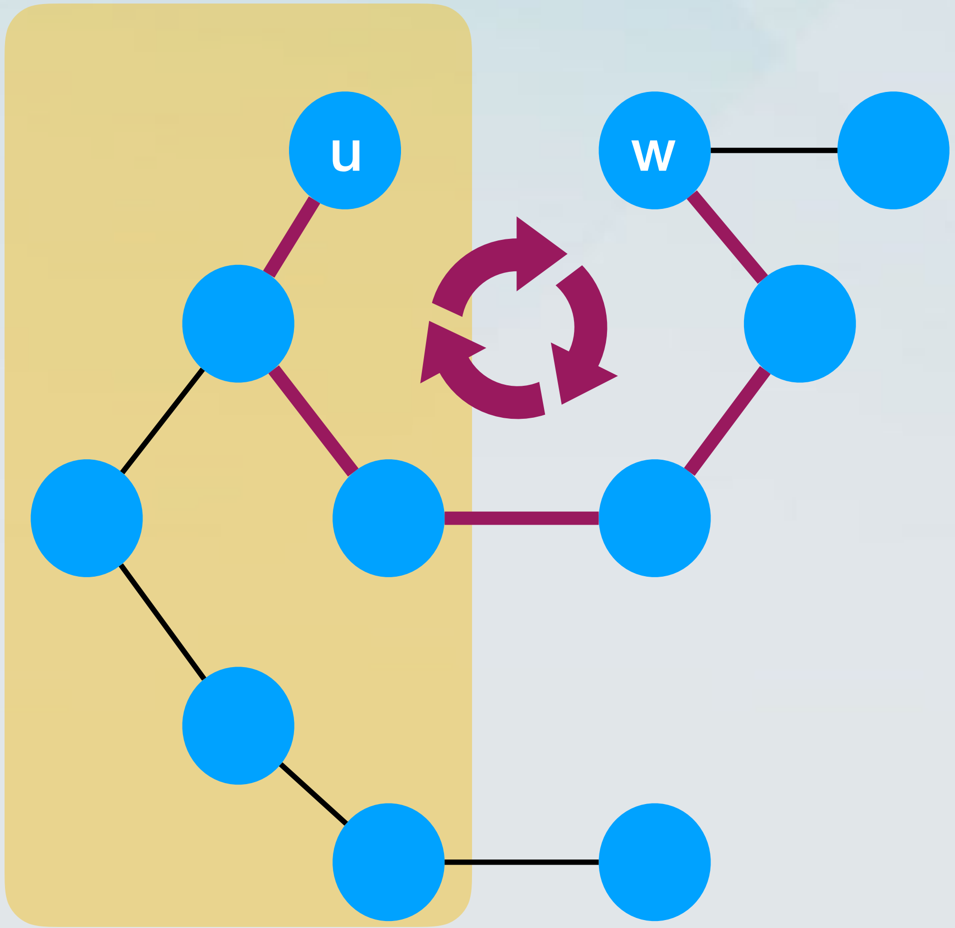
The cycle property

- We delete e from T.
- This partitions the nodes into
  - S (containing u).
  - V - S (containing w).
- We follow the other path the cycle from u to w.



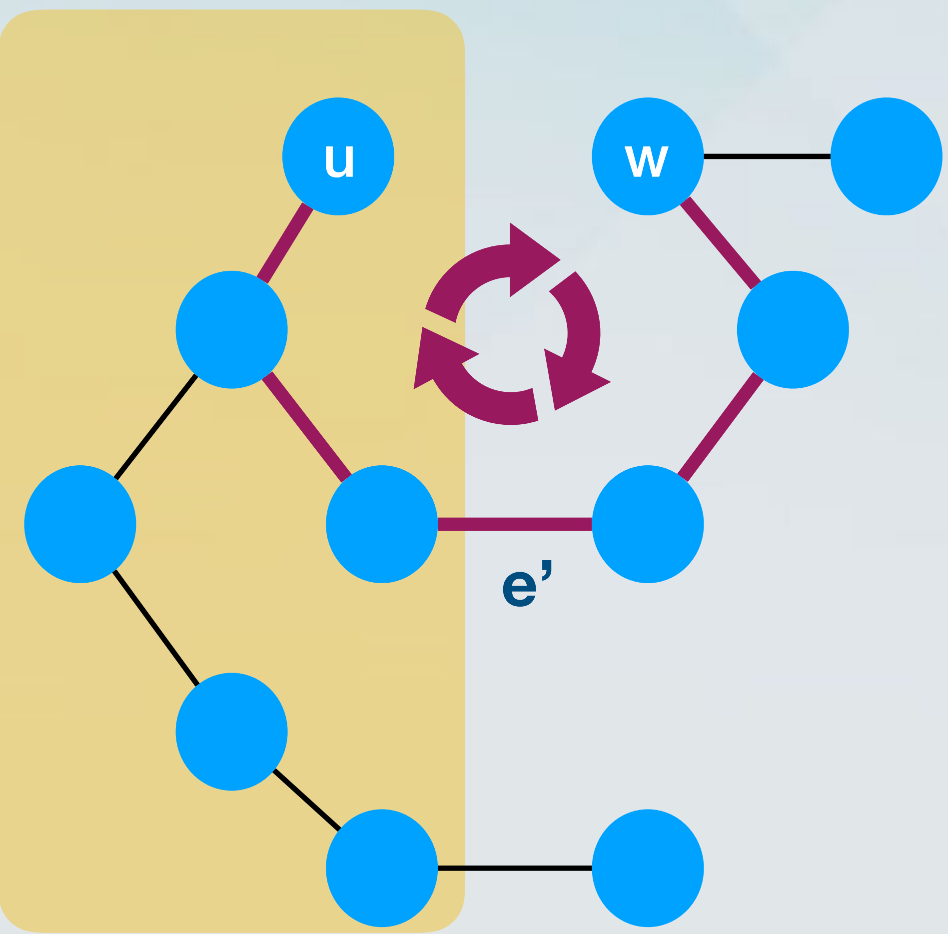
The cycle property

- We delete e from T.
- This partitions the nodes into
  - S (containing u).
  - V - S (containing w).
- We follow the other path the cycle from u to w.
- At some point we cross from S to V - S, following edge e’.



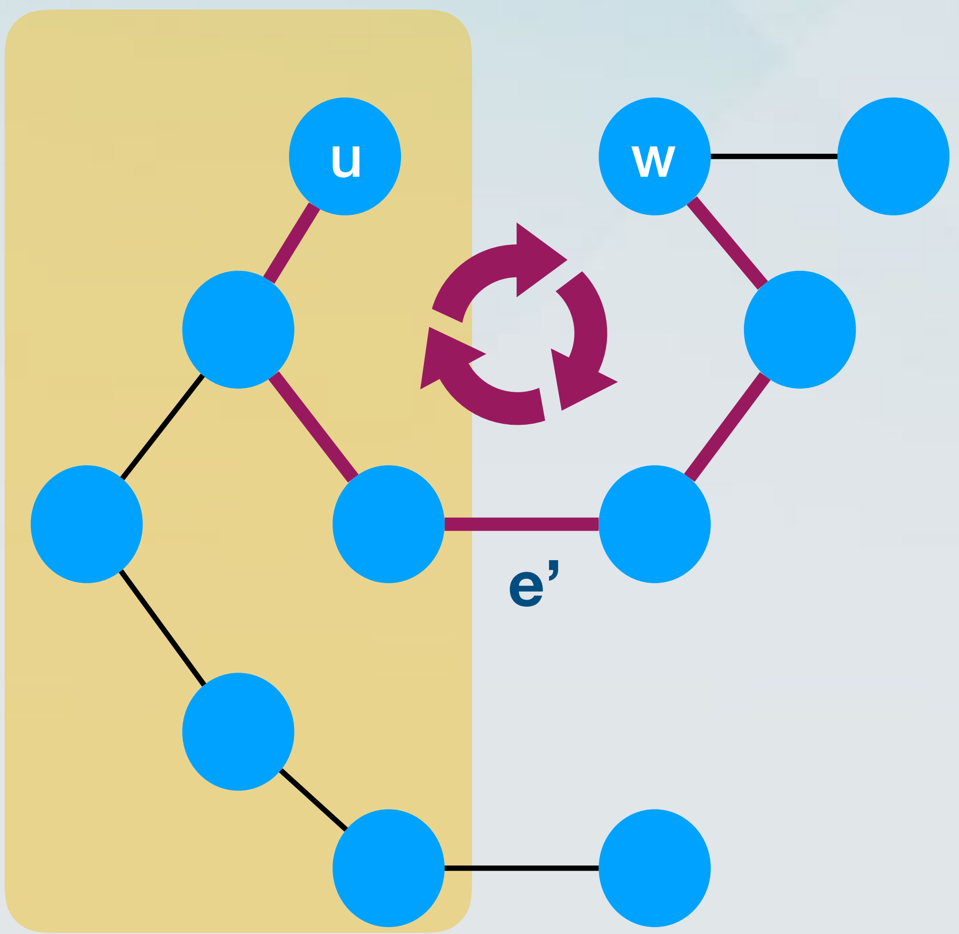
The cycle property

- We delete e from T.
- This partitions the nodes into
  - S (containing u).
  - V - S (containing w).
- We follow the other path the cycle from u to w.
- At some point we cross from S to V - S, following edge e'.



The cycle property

- We delete e from T.
- This partitions the nodes into
  - S (containing u).
  - V - S (containing w).
- We follow the other path the cycle from u to w.
- At some point we cross from S to V - S, following edge e'.
- The resulting graph is a tree with smaller cost.



Reverse-Delete is optimal

Reverse-Delete is optimal

- Consider any edge e=(v, w) which is removed by Reverse-Delete.

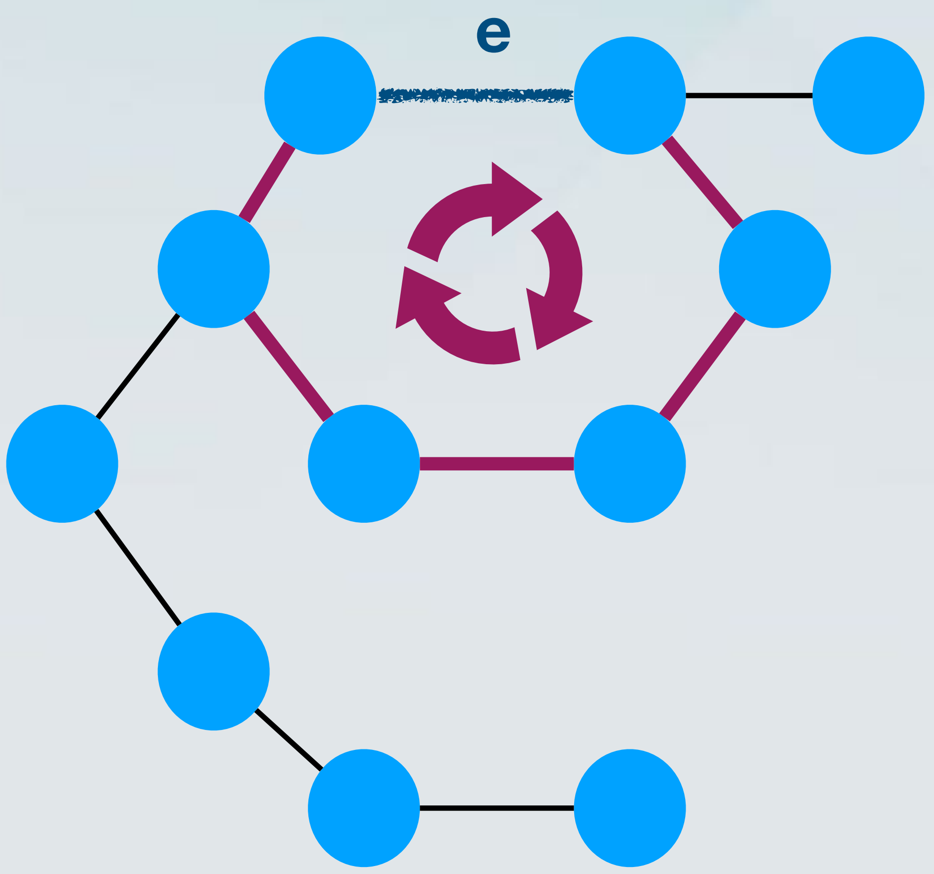
Reverse-Delete is optimal

- Consider any edge e=(v, w) which is removed by Reverse-Delete.
- Just before deleting, it lies on some cycle C.

Reverse-Delete is optimal

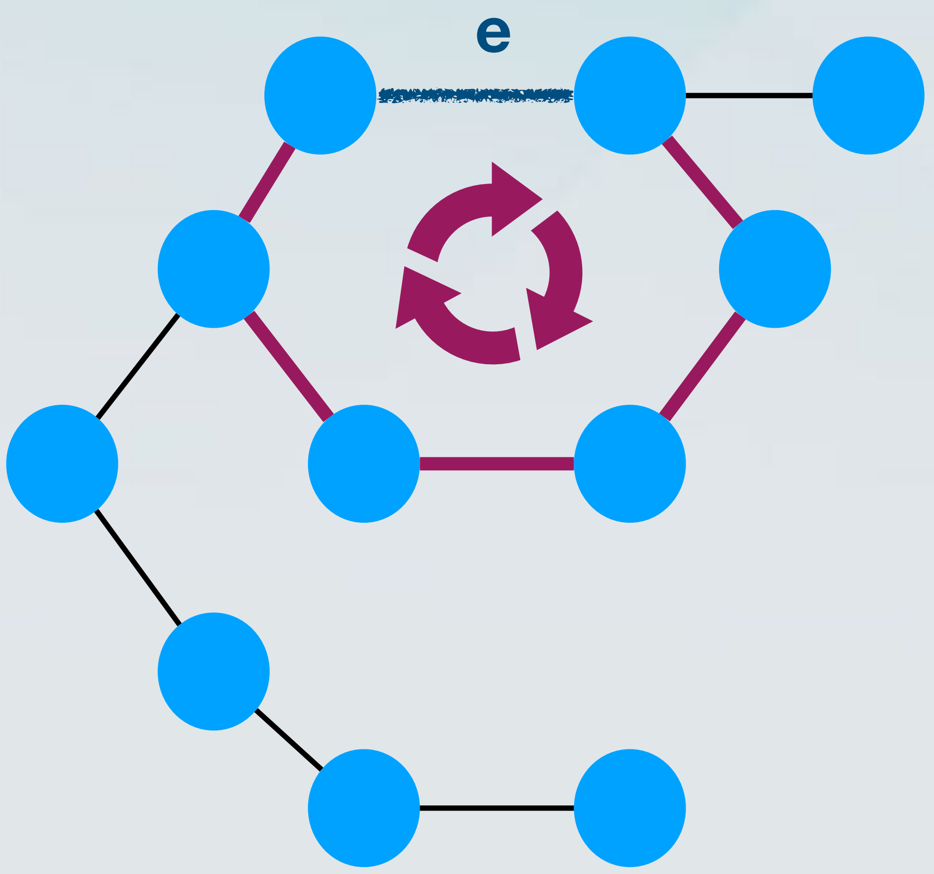
- Consider any edge e=(v, w) which is removed by Reverse-Delete.
- Just before deleting, it lies on some cycle C.
- It has the maximum cost among edges, so it cannot be part of any minimum spanning tree.

Is it feasible?



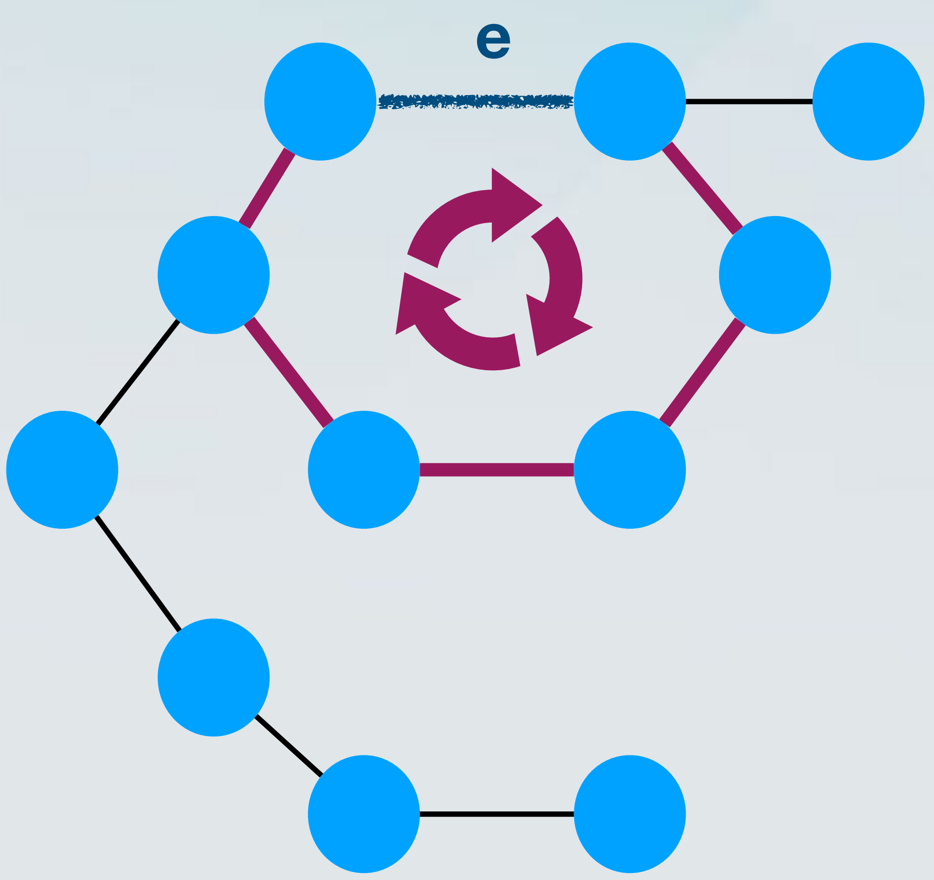
Is it feasible?

- i.e., does it always produce a spanning tree?



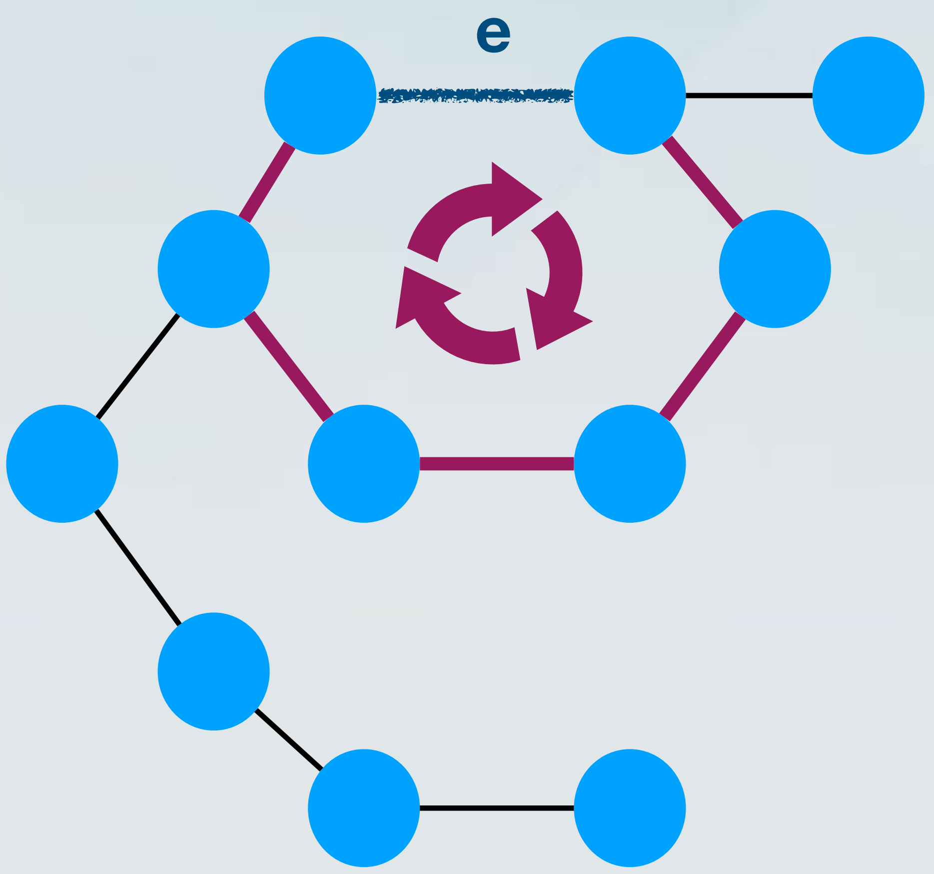
Is it feasible?

- i.e., does it always produce a spanning tree?
- Is it connected?



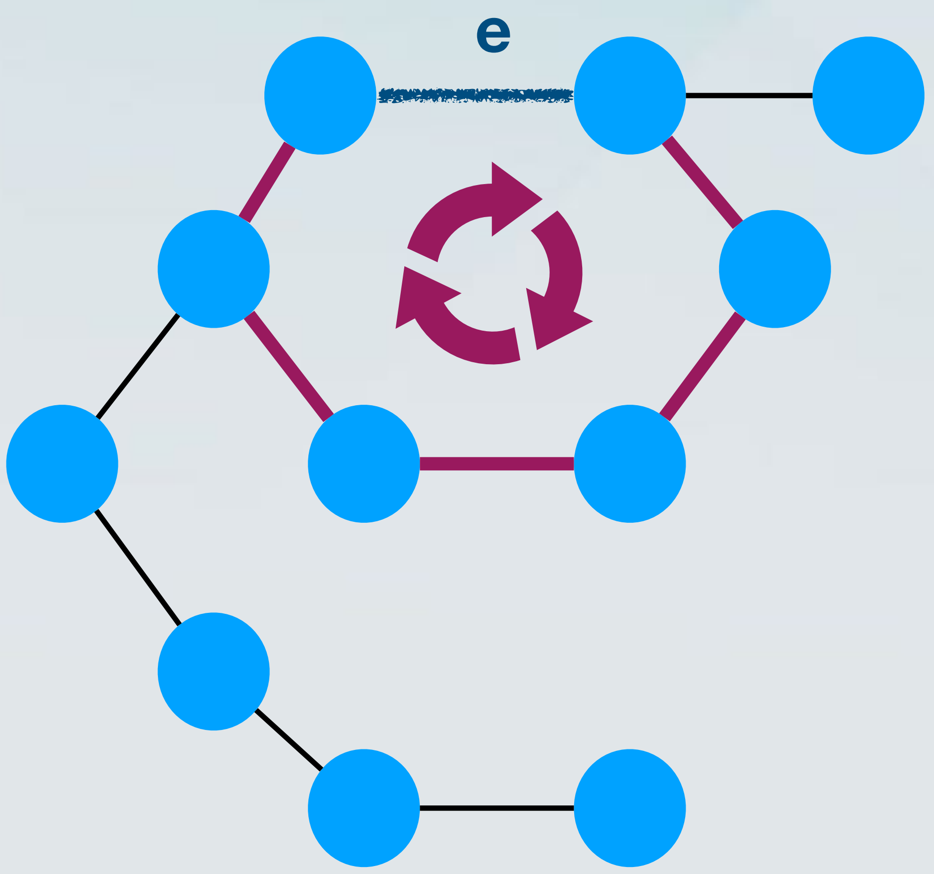
Is it feasible?

- i.e., does it always produce a spanning tree?
- Is it connected?
  - The algorithm will never disconnect the graph.



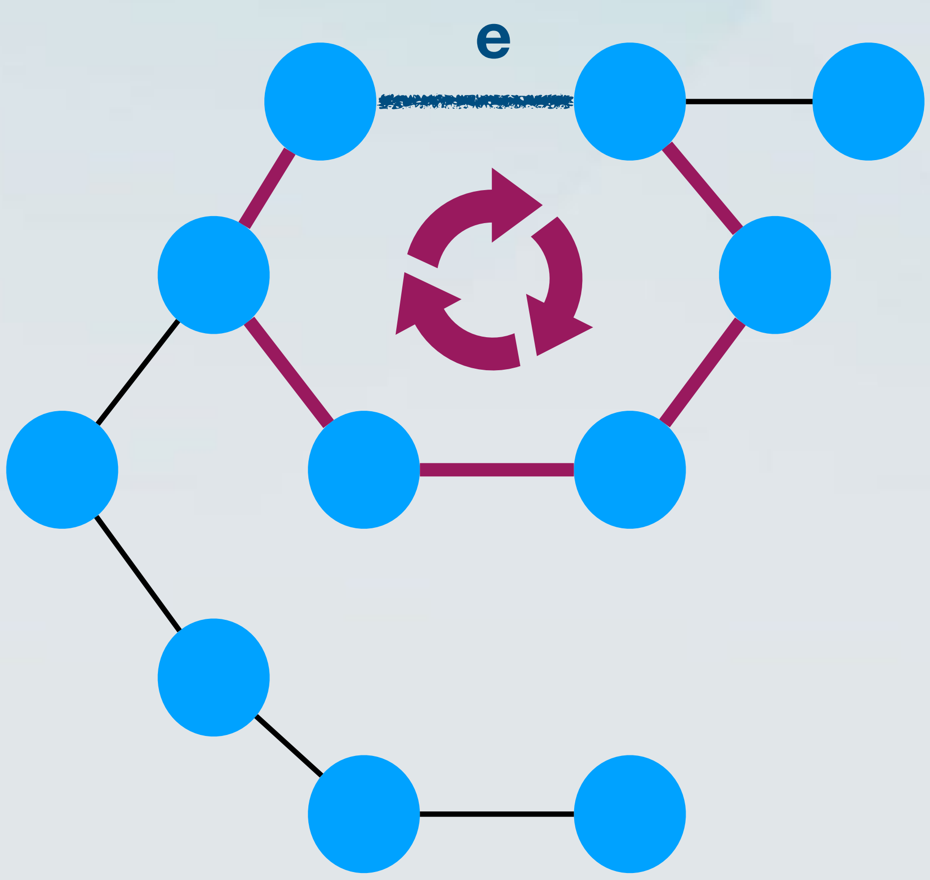
Is it feasible?

- i.e., does it always produce a spanning tree?
- Is it connected?
  - The algorithm will never disconnect the graph.
- Is it a tree?



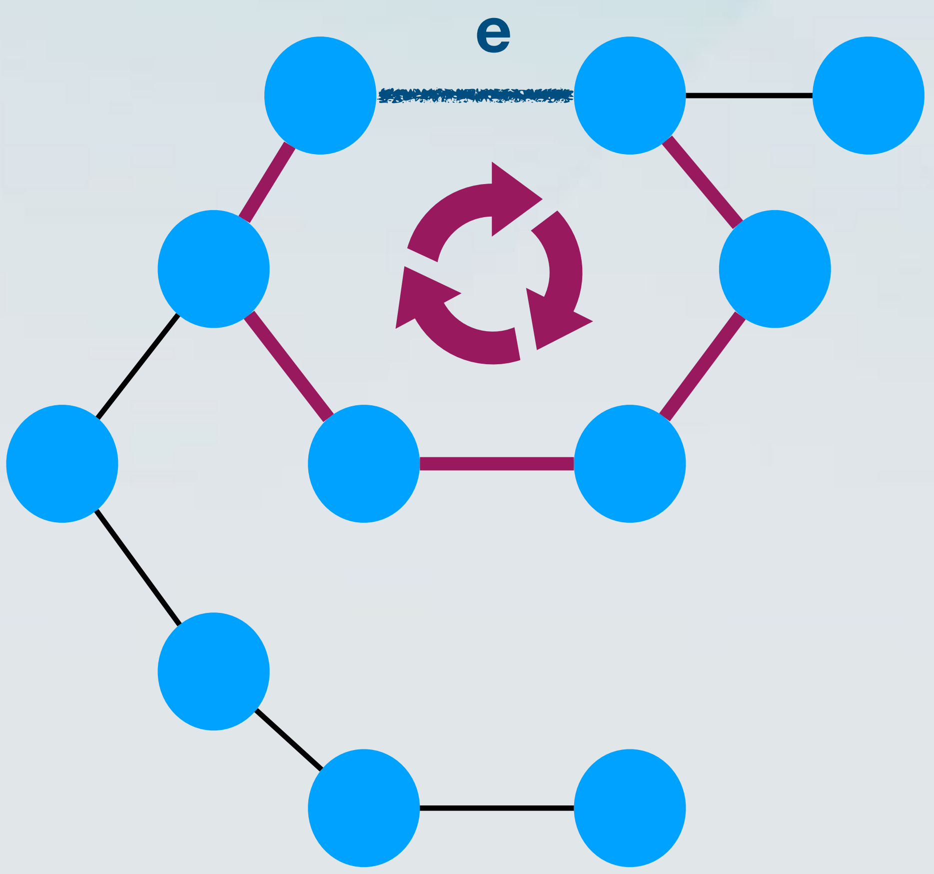
Is it feasible?

- i.e., does it always produce a spanning tree?
- Is it connected?
  - The algorithm will never disconnect the graph.
- Is it a tree?
  - Suppose that it's not.



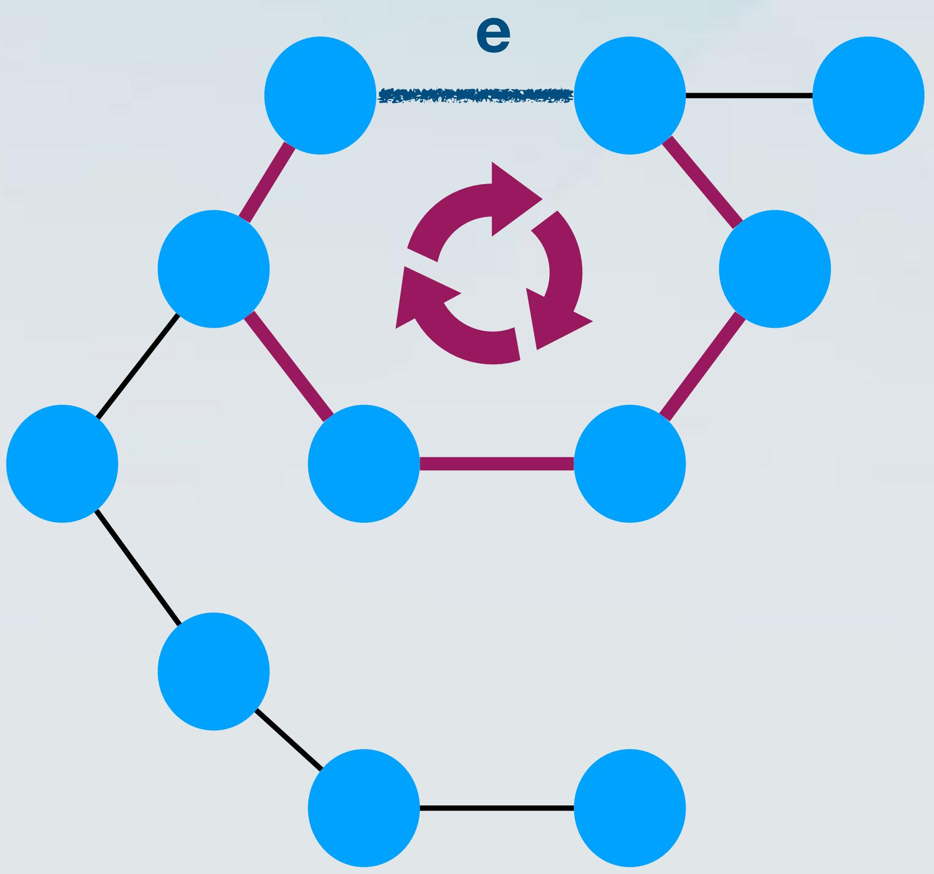
Is it feasible?

- i.e., does it always produce a spanning tree?
- Is it connected?
  - The algorithm will never disconnect the graph.
- Is it a tree?
  - Suppose that it's not.
  - Then it contains some cycle C.



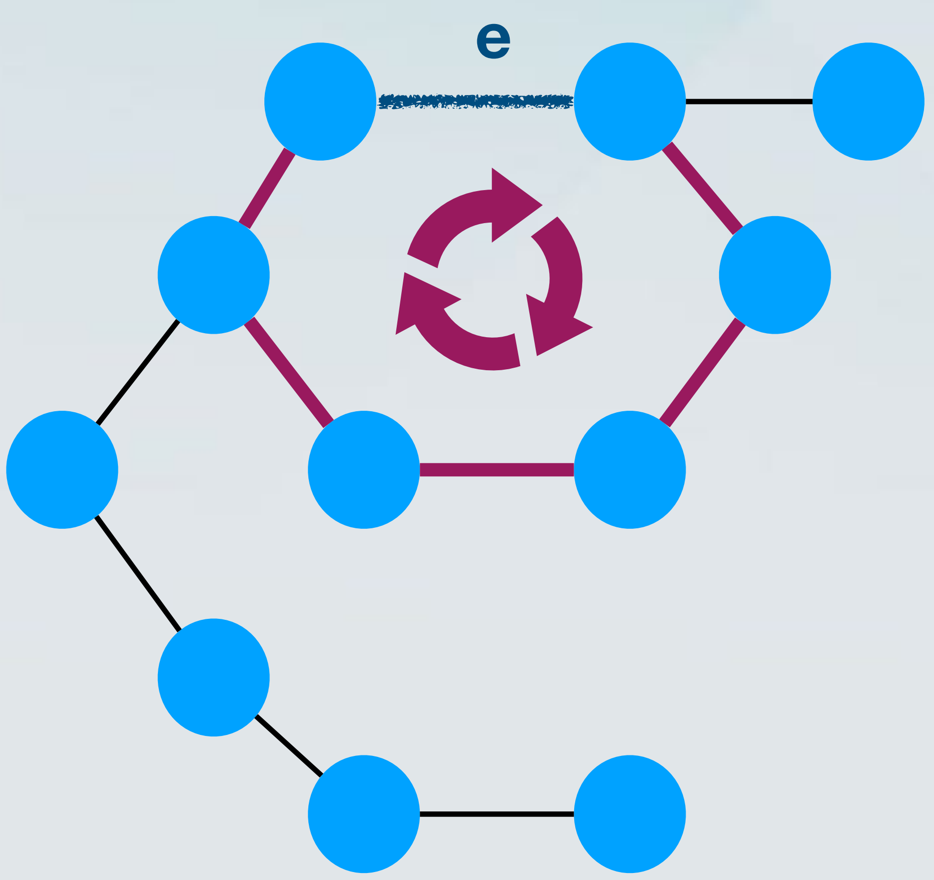
Is it feasible?

- i.e., does it always produce a spanning tree?
- Is it connected?
  - The algorithm will never disconnect the graph.
- Is it a tree?
  - Suppose that it's not.
  - Then it contains some cycle C.
  - Consider the most expensive edge e on that cycle.



Is it feasible?

- i.e., does it always produce a spanning tree?
- Is it connected?
  - The algorithm will never disconnect the graph.
- Is it a tree?
  - Suppose that it's not.
  - Then it contains some cycle C.
  - Consider the most expensive edge e on that cycle.
  - The algorithm would have removed that edge.



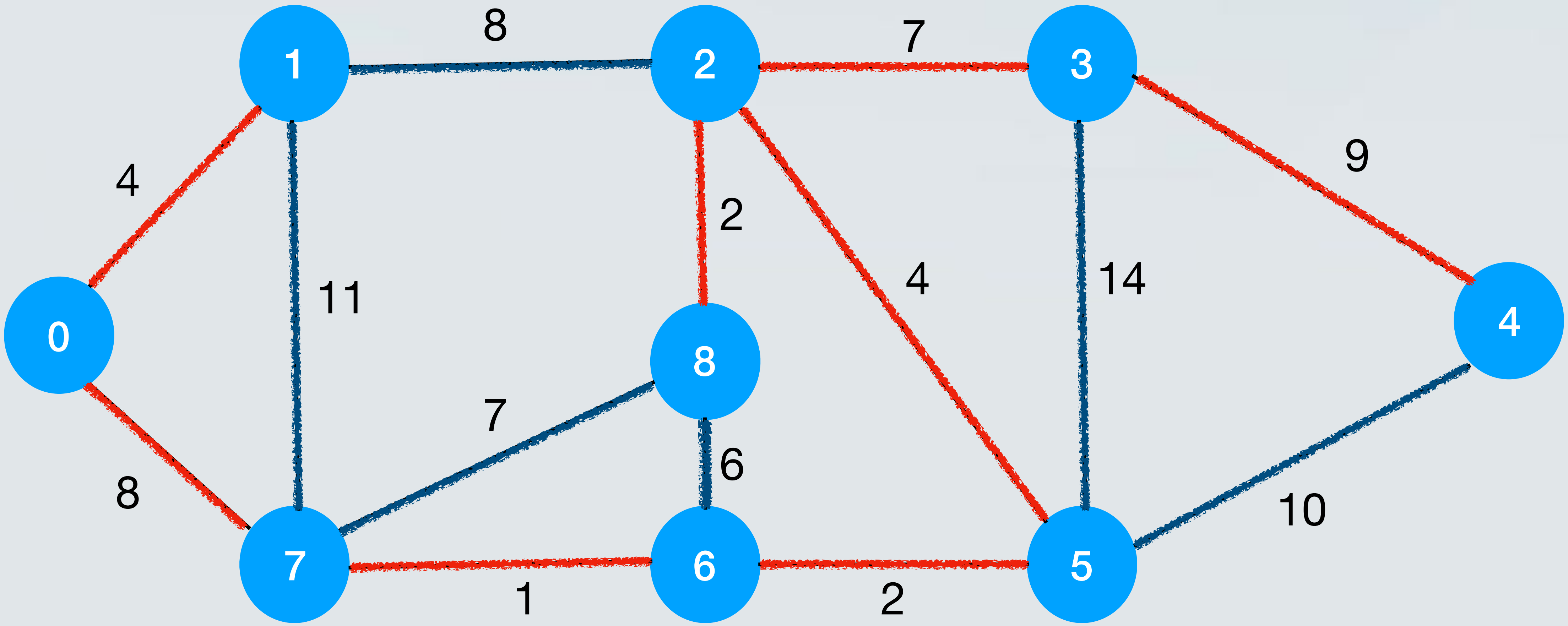
Are we done?

Are we done?

- “Assume that all edge costs are distinct”.
- What if they are not?

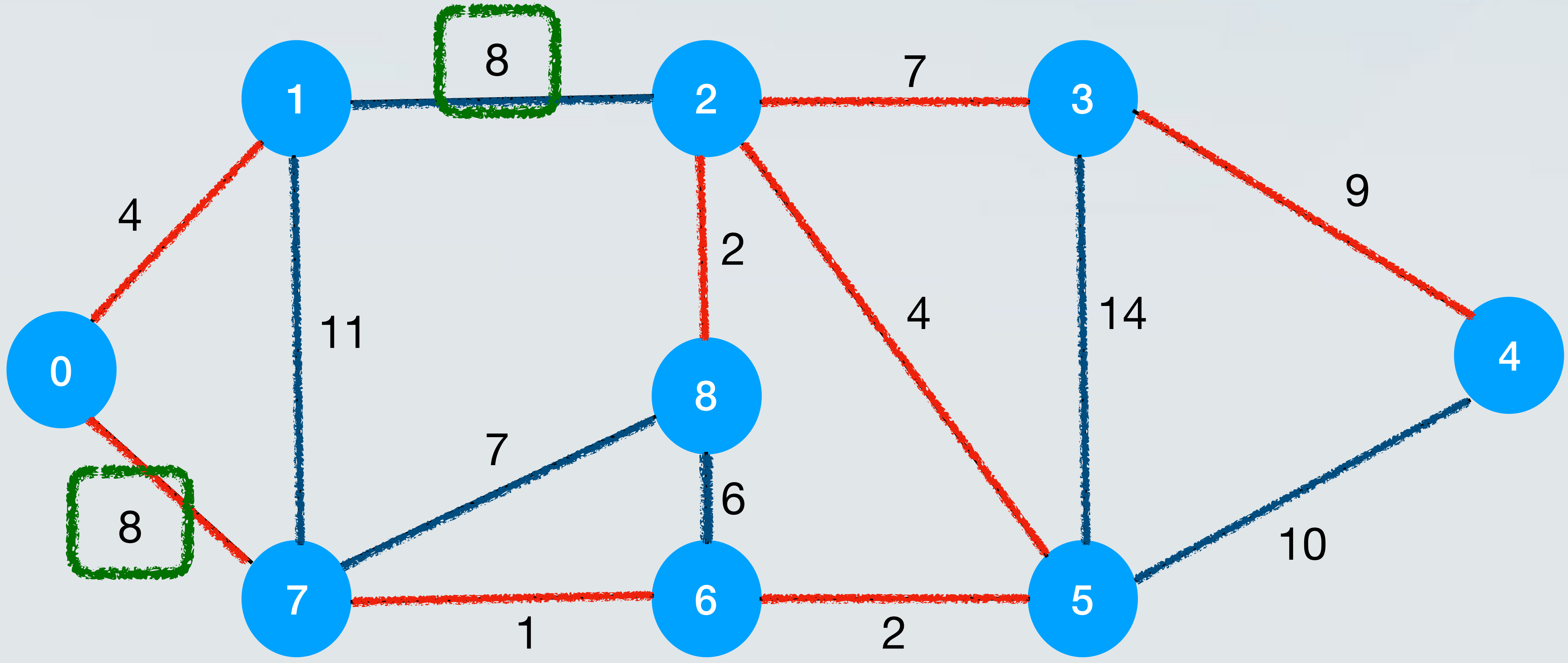
Are we done?

- “Assume that all edge costs are distinct”.
- What if they are not?



Are we done?

- “Assume that all edge costs are distinct”.
- What if they are not?



Non-distinct costs

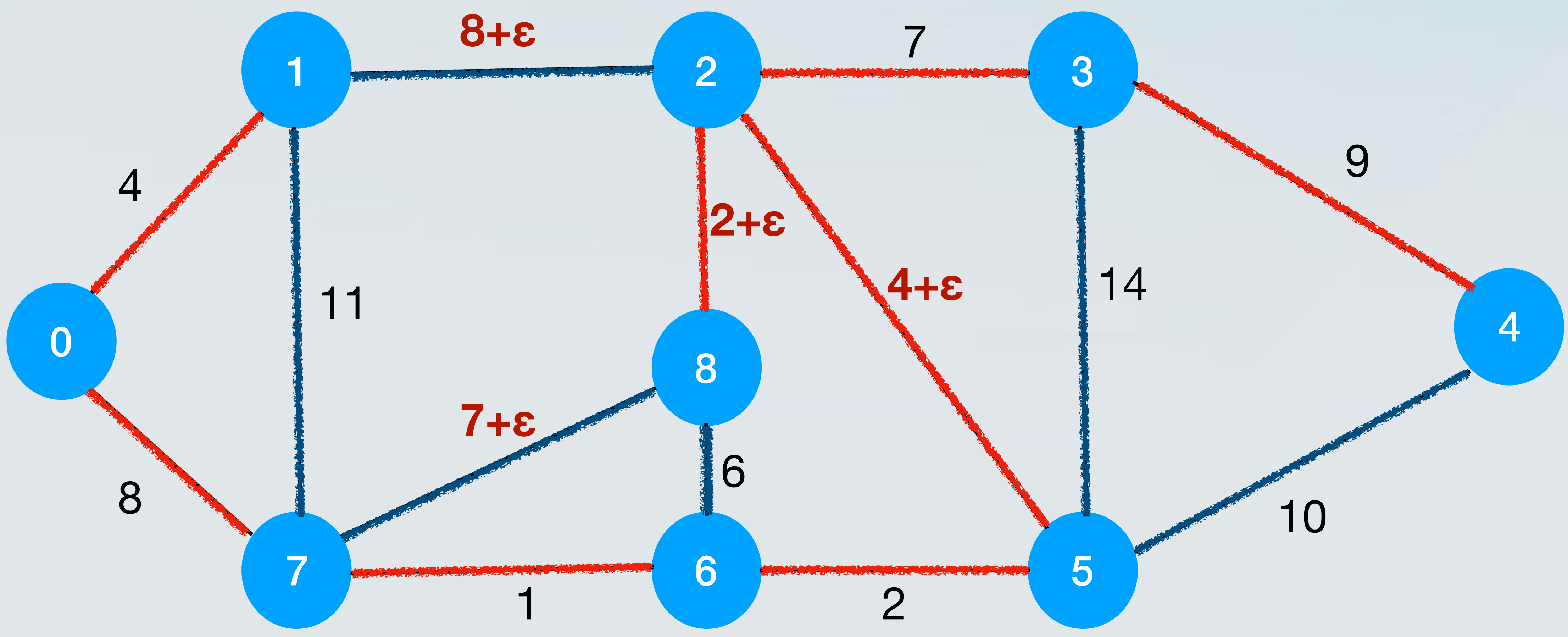
Non-distinct costs

- Take the original instance with non-distinct costs.
- Make the costs distinct by adding small numbers ε to the costs to break ties.
- Obtain a perturbed instance.
- Run the algorithm on the perturbed instance.
- Output the minimum spanning tree T.
- T is a minimum spanning tree on the original instance.

T in the original instance

- Suppose that there was a cheaper spanning tree T* on the original instance.
- If T* contains different edges with the same costs, it is not cheaper than T on the original instance.
- If T contains different edges with different costs, we can make ε small enough to make sure the ones we selected are still cheaper.

Perturbing the costs



Perturbing the costs

1, 2, 2, 4, 4, 6, 7, 7, 8, 8, 9, 10, 11, 14

1, 2, 2+ε, 4, 4+ε, 6, 7, 7+ε, 8, 8+ε, 9, 10, 11, 14

Running time?

Running time?

- Kruskal’s Algorithm

Running time?

- Kruskal’s Algorithm
  - We will not cover it, Kleinberg and Tardos Chapter 4.6.

Running time?

- Kruskal's Algorithm
  - We will not cover it, Kleinberg and Tardos Chapter 4.6.
- Prim's Algorithm

Running time?

- Kruskal's Algorithm
  - We will not cover it, Kleinberg and Tardos Chapter 4.6.
- Prim's Algorithm
  - Next lecture.