# Algorithms and Data Structures

## The Greedy Approach and Minimum Spanning Trees

# The Greedy approach

- The goal is to come up with a global solution.
- The solution will be built up in small consecutive steps.
- For each step, the solution will be the best possible myopically, according to some criterion.

# Graph Theory Basics

# Graph Definitions

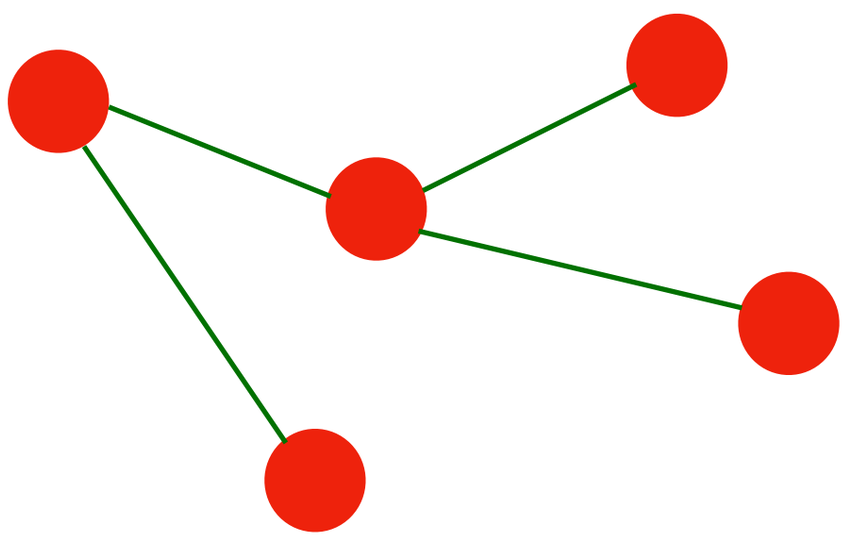
Graph $G = (V,E)$

Set of nodes (or vertices) $V$, with $|V| = n$

Set of edges $E$, with $|E| = m$

Undirected: edge $e = \{v,w\}$

Directed: edge $e = (v,w)$



# Graph Definitions

**Neighbours of $v$ :** Set of nodes connected by an edge with $v$

**Degree of a node:** number of neighbours

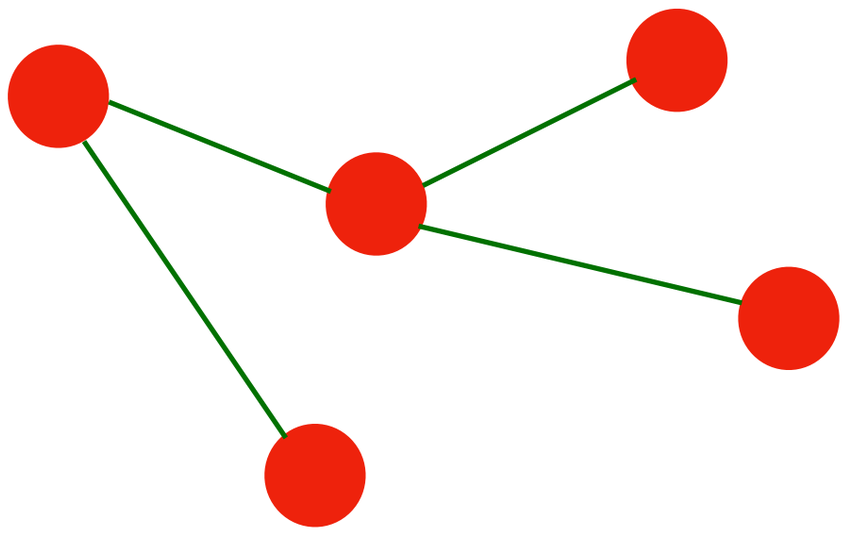
Directed graphs: *in-degree* and *out-degree*

**Path:** A sequence of (non-repeating) nodes with consecutive nodes being connected by an edge.

Length: # nodes - 1

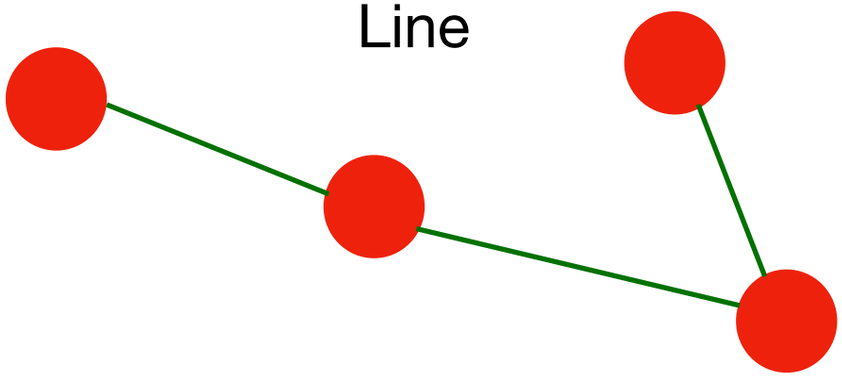
**Distance between $u$ and $v$ :** length of the shortest path $u$ and $v$,

**Graph diameter:** The longest distance in the graph

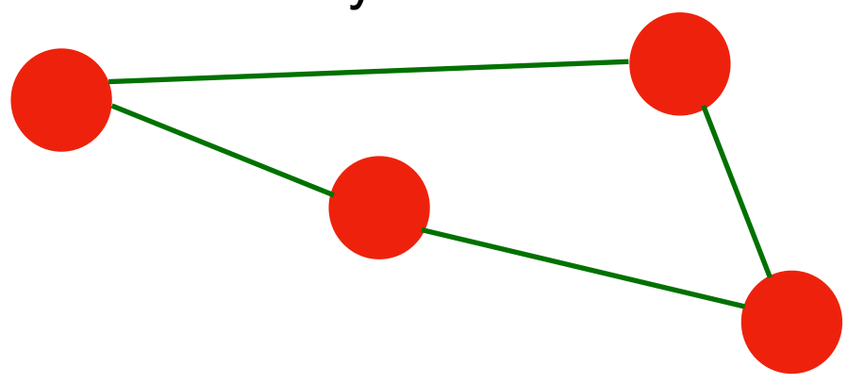


# Lines, cycles, trees and cliques

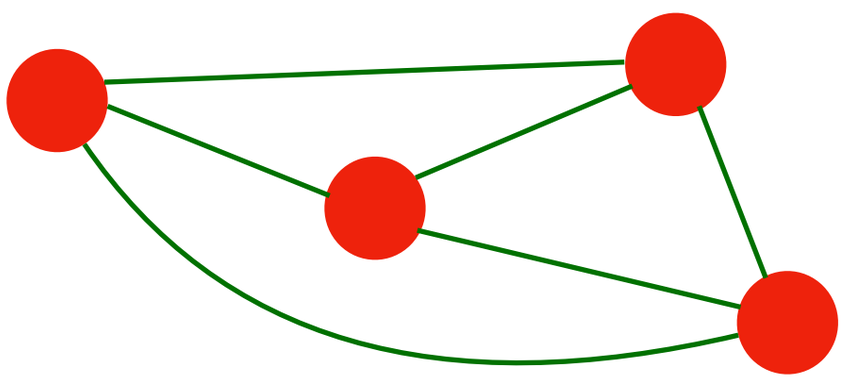
Line



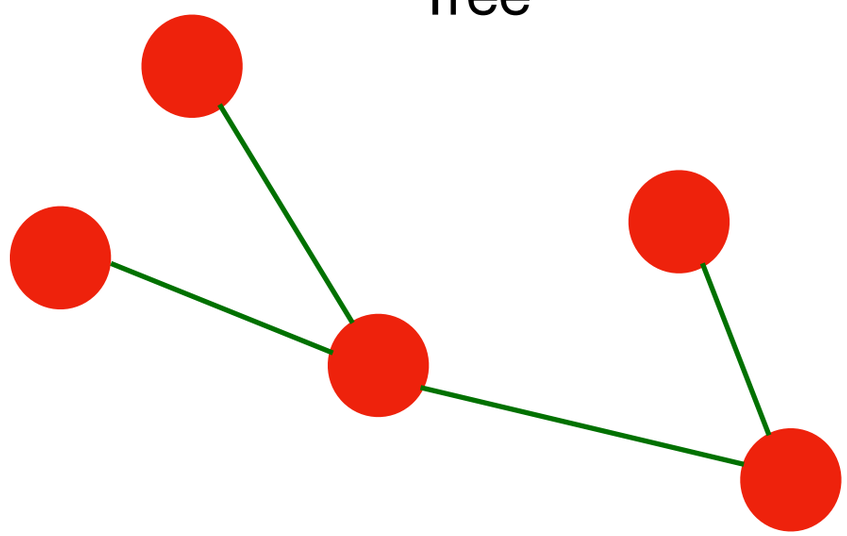
Cycle



Clique

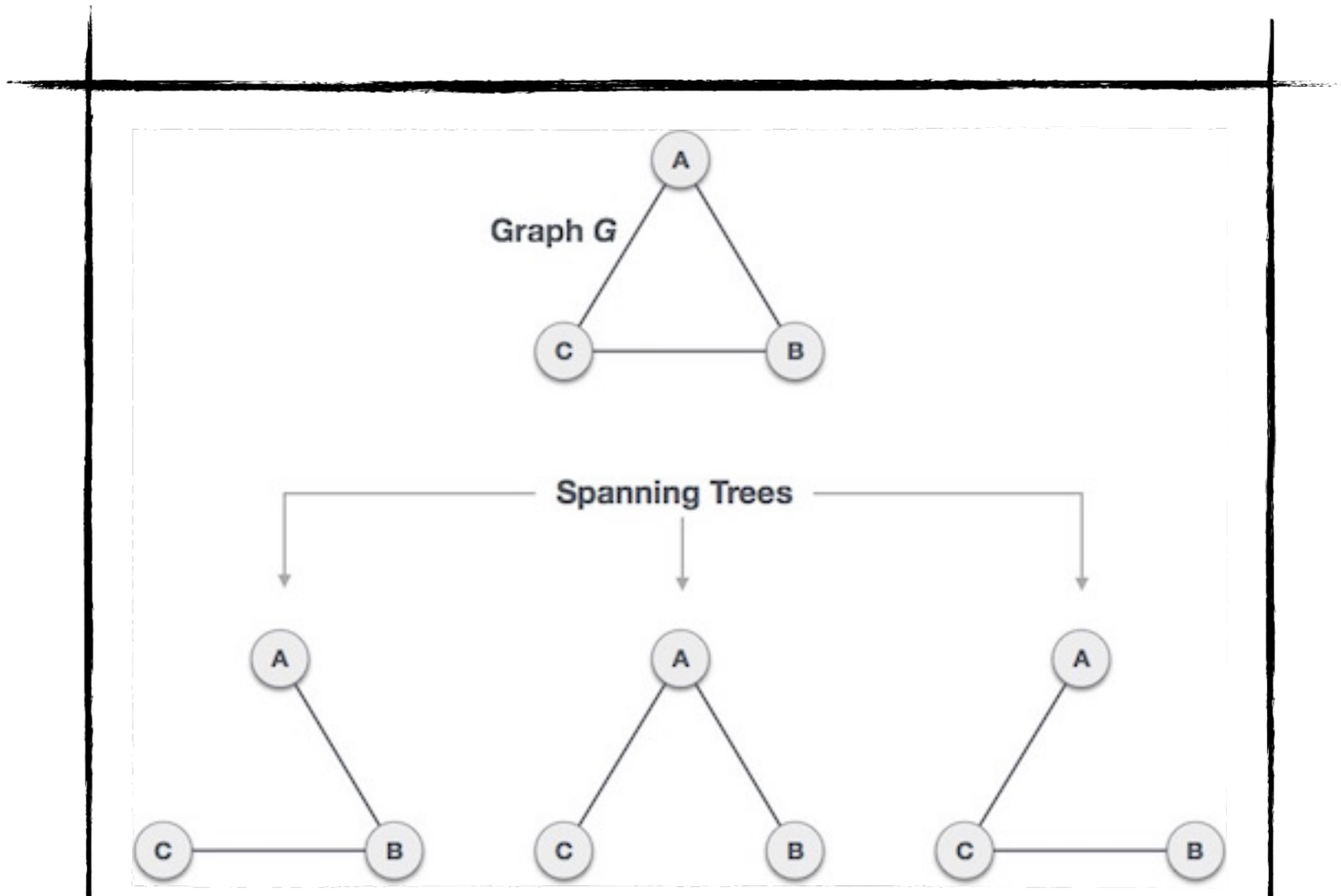


Tree



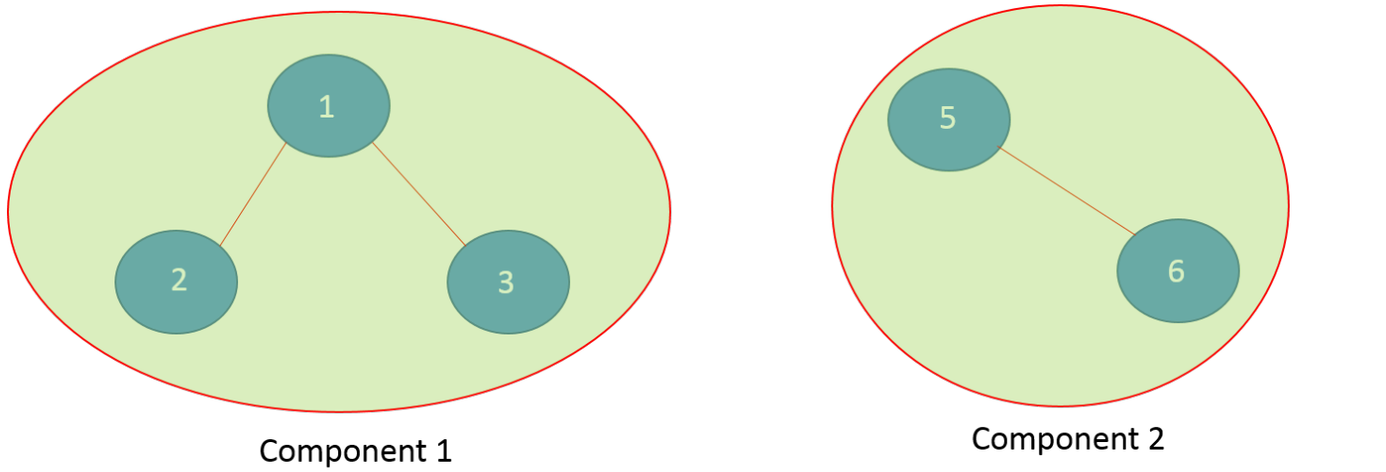
# Definitions

A **spanning tree** of a graph $G$ is a tree containing all the nodes of $G$.



# Definitions

A **connected component** of a graph $G$ is subgraph such that any two vertices are connected via some path.



# Graph Representations

How do we represent a graph $G = (V,E)$?

Adjacency Matrix

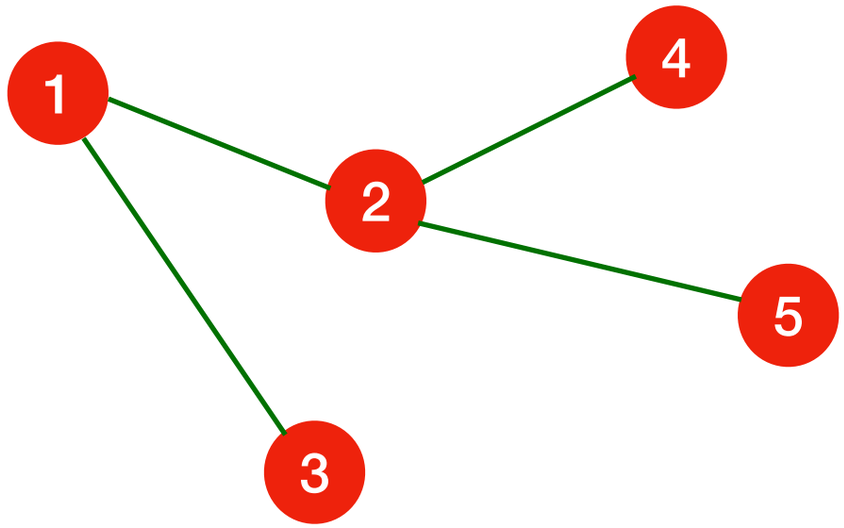
Adjacency List

# Adjacency Matrix **A**

The $i^{\text{th}}$ node corresponds to the $i^{\text{th}}$ row and the $i^{\text{th}}$ column.

If there is an edge between $i$ and $j$ in the graph, then we have **A**[$i$,$j$] = 1, otherwise **A**[$i$,$j$] = 0.

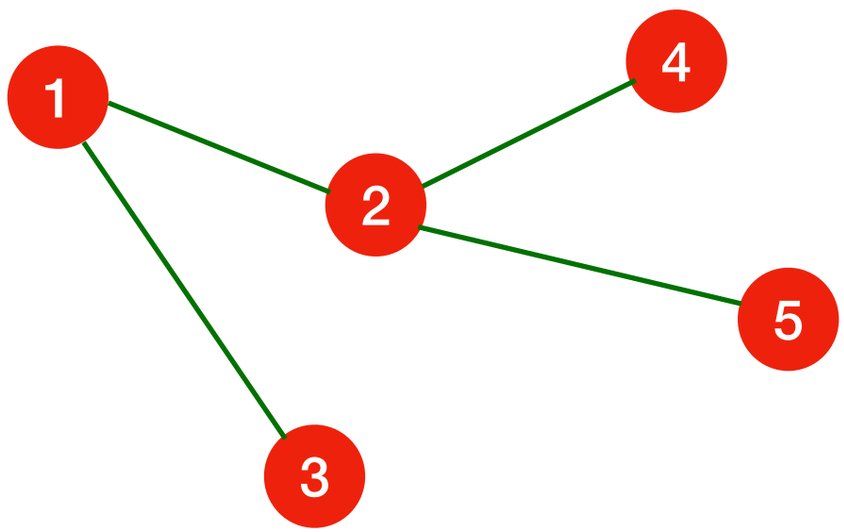
For undirected graphs, necessarily **A**[$i$,$j$] = **A**[$j$,$i$]. For directed graphs, it could be that **A**[$i$,$j$] ≠ **A**[$j$,$i$].

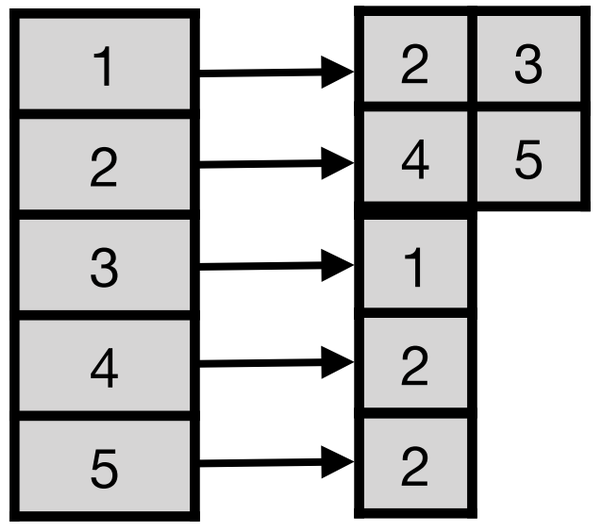



| | | | | |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |

# Adjacency List L

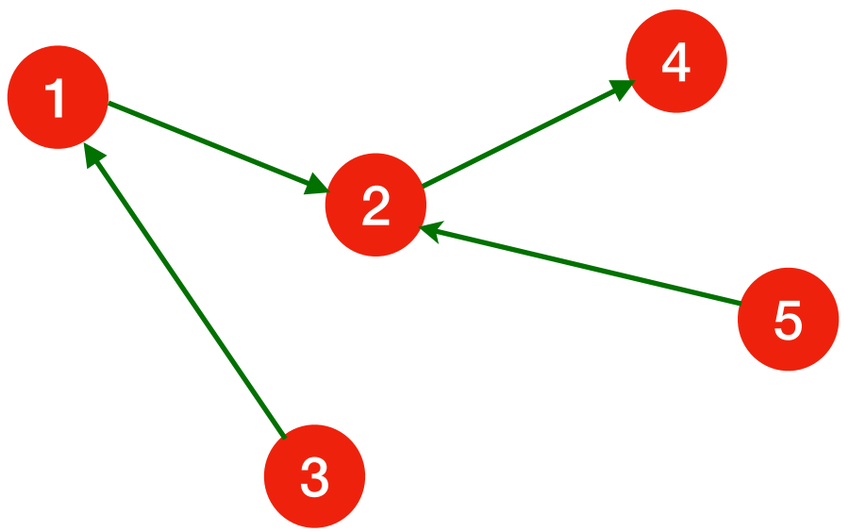
- Nodes are arranged as a list, each node points to the neighbours.
- For undirected graphs, the node points only in one direction.
- For directed graphs, the node points in two directions, for in-degree and for out-degree

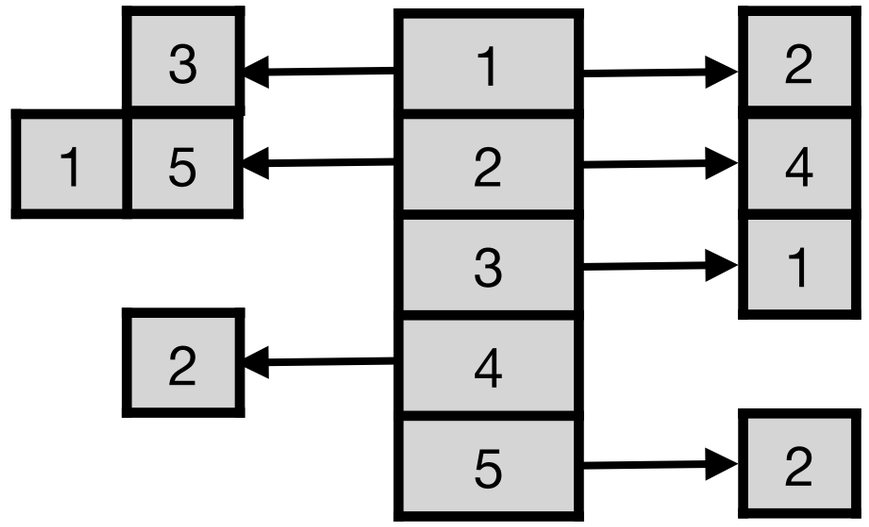


# Adjacency List L

- Nodes are arranged as a list, each node points to the neighbours.
- For undirected graphs, the node points only in one direction.
- For directed graphs, the node points in two directions, for in-degree and for out-degree.



# Adjacency Matrix vs Adjacency List

**Adjacency Matrix**

Memory: $O(n^2)$

Checking *adjacency* of $u$ and $v$
Time: O(1)

Finding *all adjacent nodes* of $u$
Time: O(n)

**Adjacency List**

Memory: $O(m+n)$

Checking *adjacency* of $u$ and $v$
Time: $O(\min(\deg(u),\deg(v))$

Finding *all adjacent nodes* of $u$
Time: $O(\deg(u))$

# Graph Traversal (Search)

We would like to go over all the possible nodes of an (undirected) graph.

There are different ways of doing that.

Two systematic ways:

- Depth-First Search
- Breadth-First Search

# Graph Traversal (Search)

We would like to go over all the possible nodes of an (undirected) graph.

There are different ways of doing that.

Two systematic ways:

- Depth-First Search
- Breadth-First Search

KT Chapter 3.2.

CLRS Chapter 20.2, 20.3

# Minimum Spanning Tree via Greedy

# 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 at 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: ($V$, $T$) is a tree

# Claim: ($V$, $T$) is a tree

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

# Claim: ($V$, $T$) is a tree

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

Suppose that it contained a cycle (i.e., it is not a tree).

# Claim: ($V$, $T$) is a tree

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

Suppose that it contained a cycle (i.e., it is not a tree).

Let $e$ be an edge on that cycle.

# Claim: ($V$, $T$) is a tree

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

Suppose that it contained a cycle (i.e., it is not a tree).

Let $e$ be an edge on that cycle.

Take ($V$, $T$-{$e$}).

# Claim: ($V$, $T$) is a tree

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

Suppose that it contained a cycle (i.e., it is not a tree).

Let $e$ be an edge on that cycle.

Take ($V$, $T$-{$e$}).

This is still connected.

# Claim: ($V$, $T$) is a tree

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

Suppose that it contained a cycle (i.e., it is not a tree).

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: ($V$, $T$) is a tree

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

Suppose that it contained a cycle (i.e., it is not a tree).

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

$G'=(V, T)$ 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?

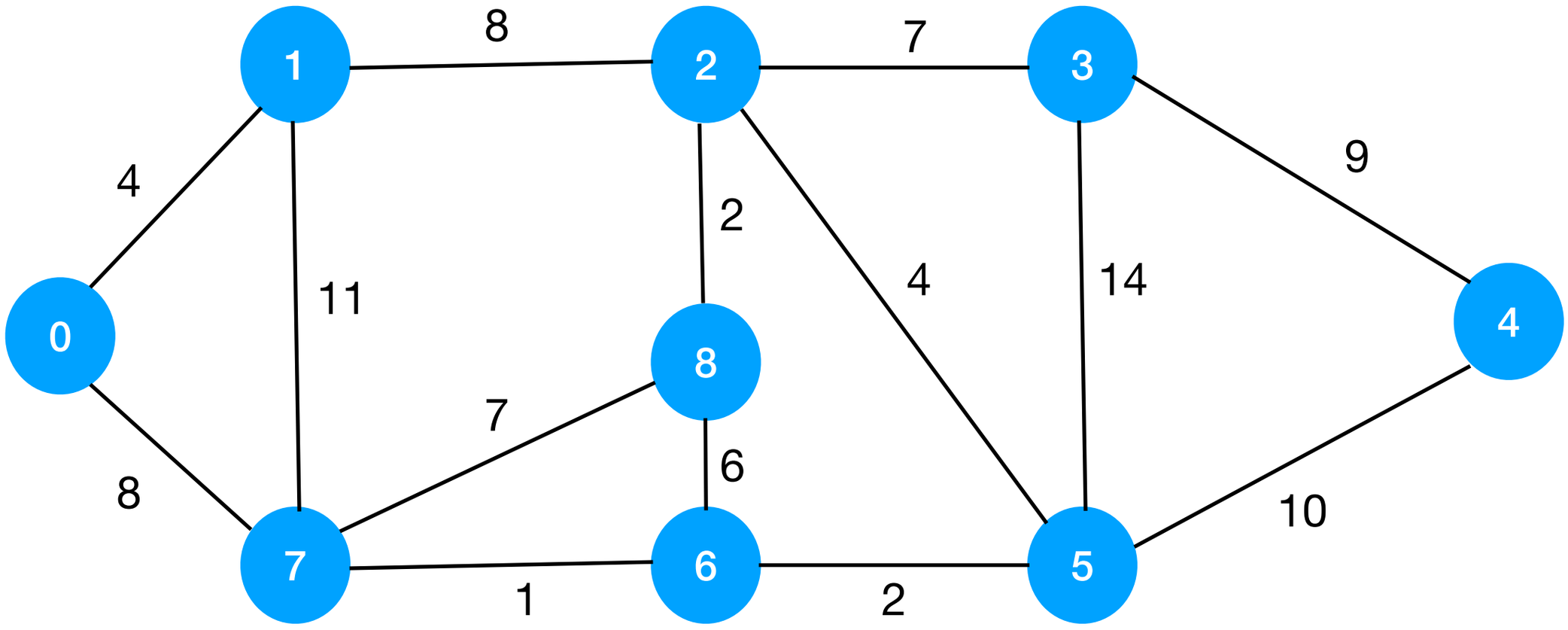
The one with the minimum cost $c_e$.

We continue like this.

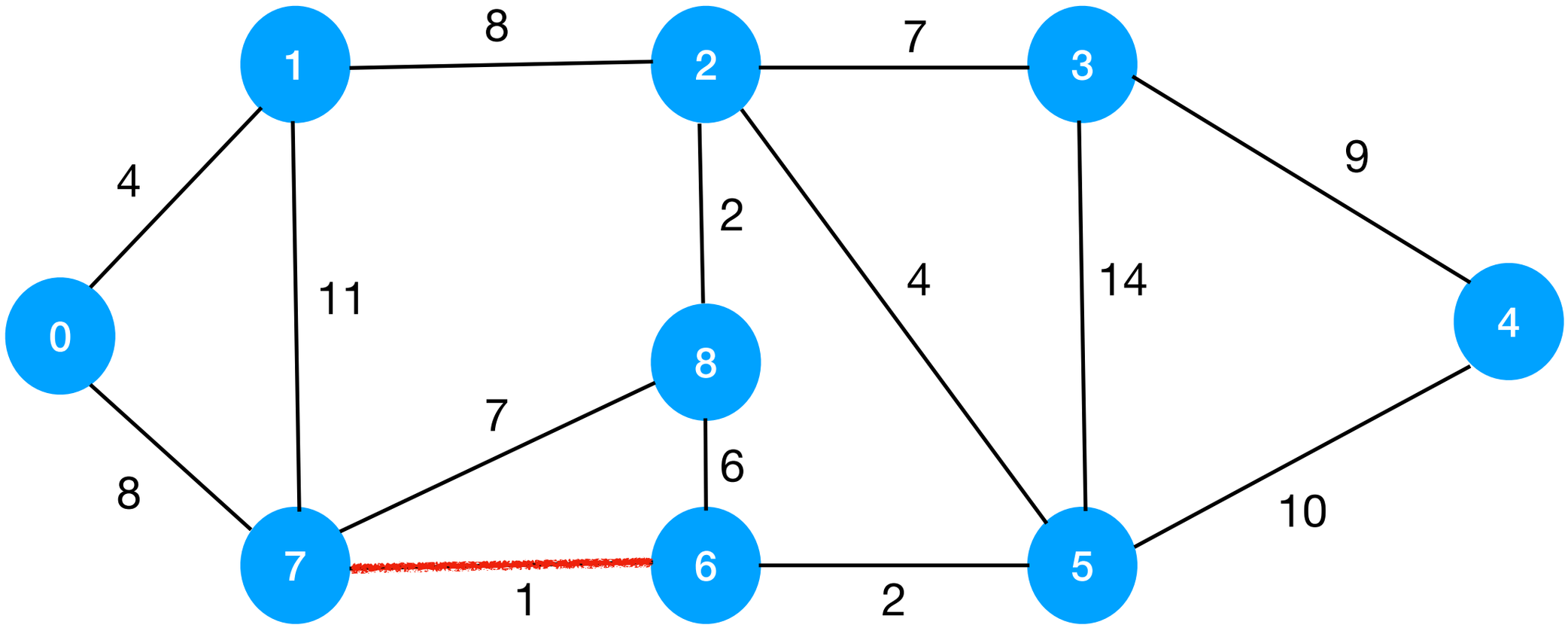
Do we always add the new edge $e$ to $T$ ?

Only if we don’t introduce any cycles.

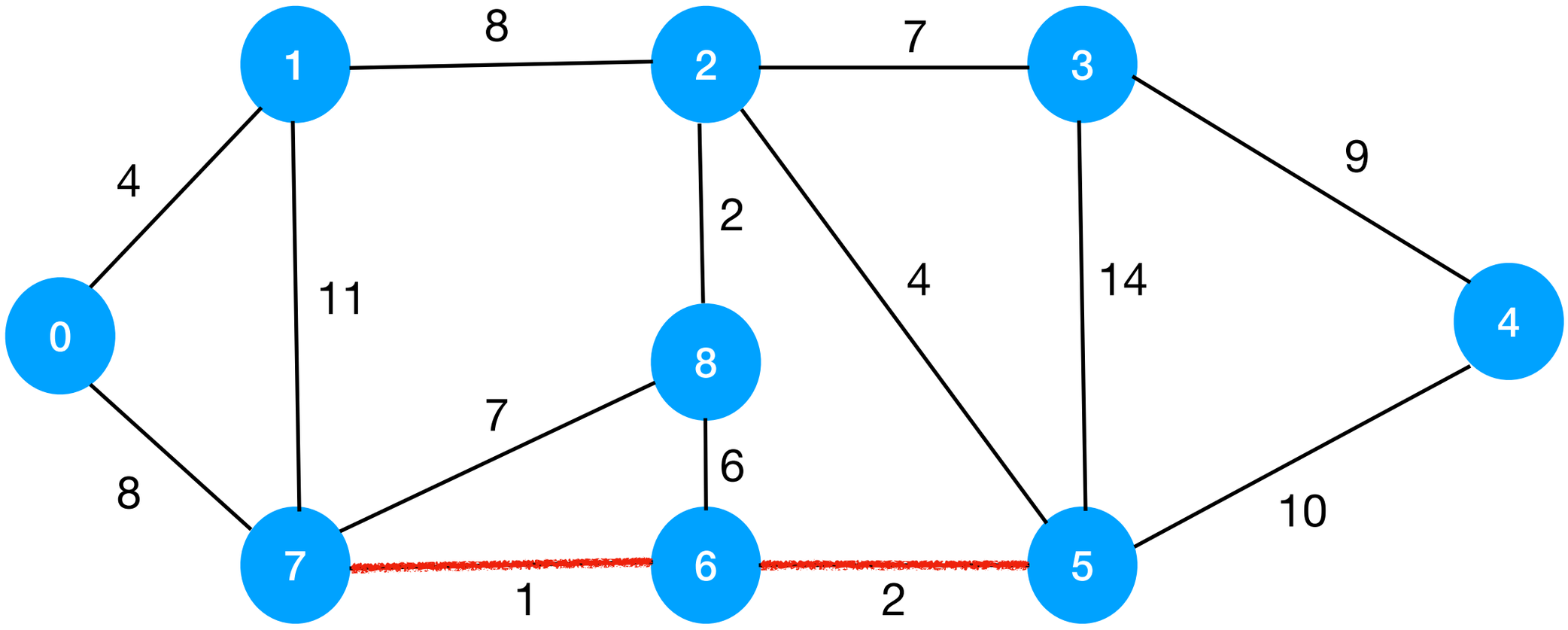
# Example



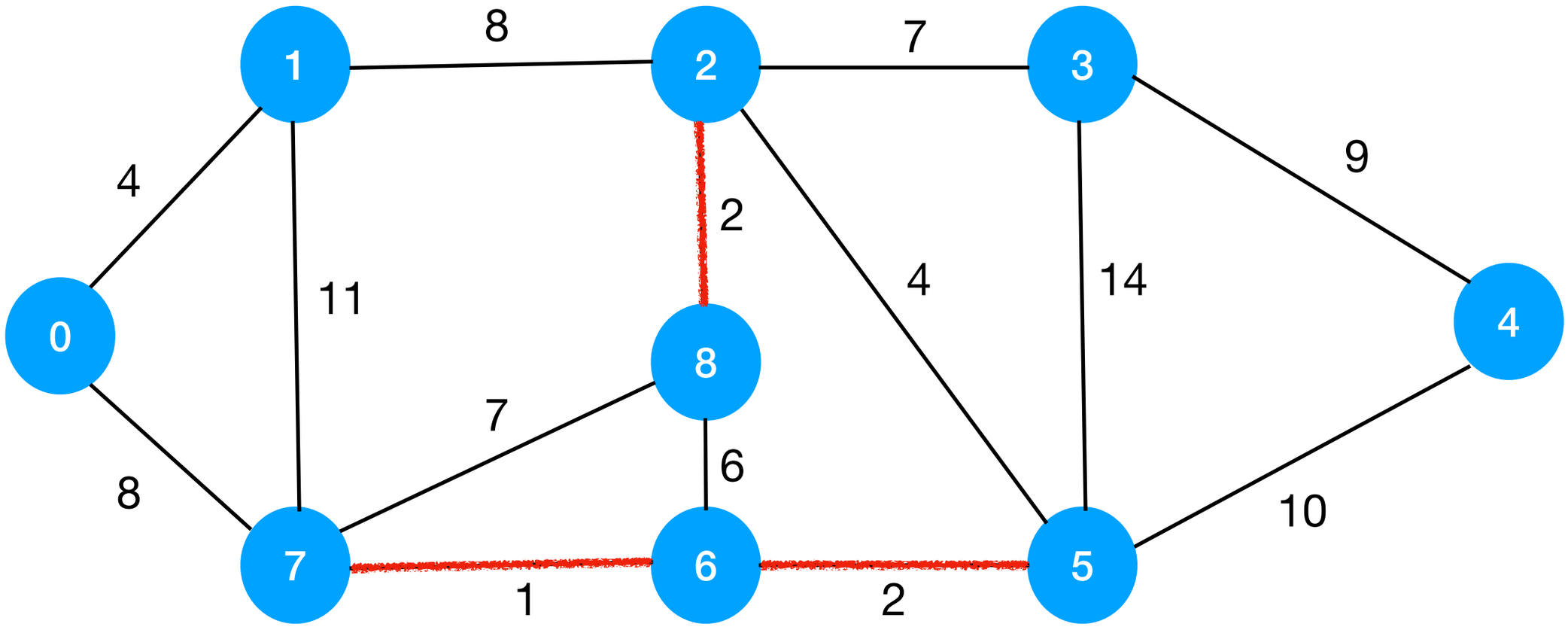
# Example



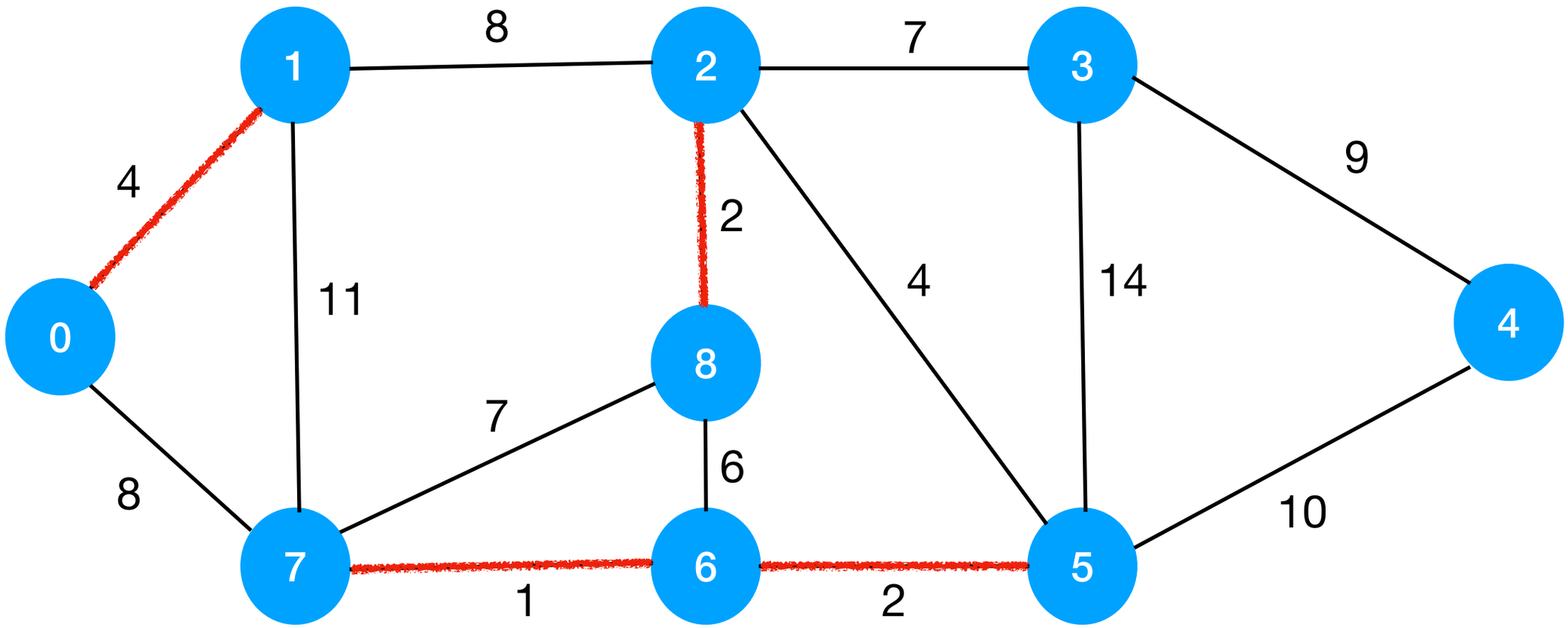
# Example



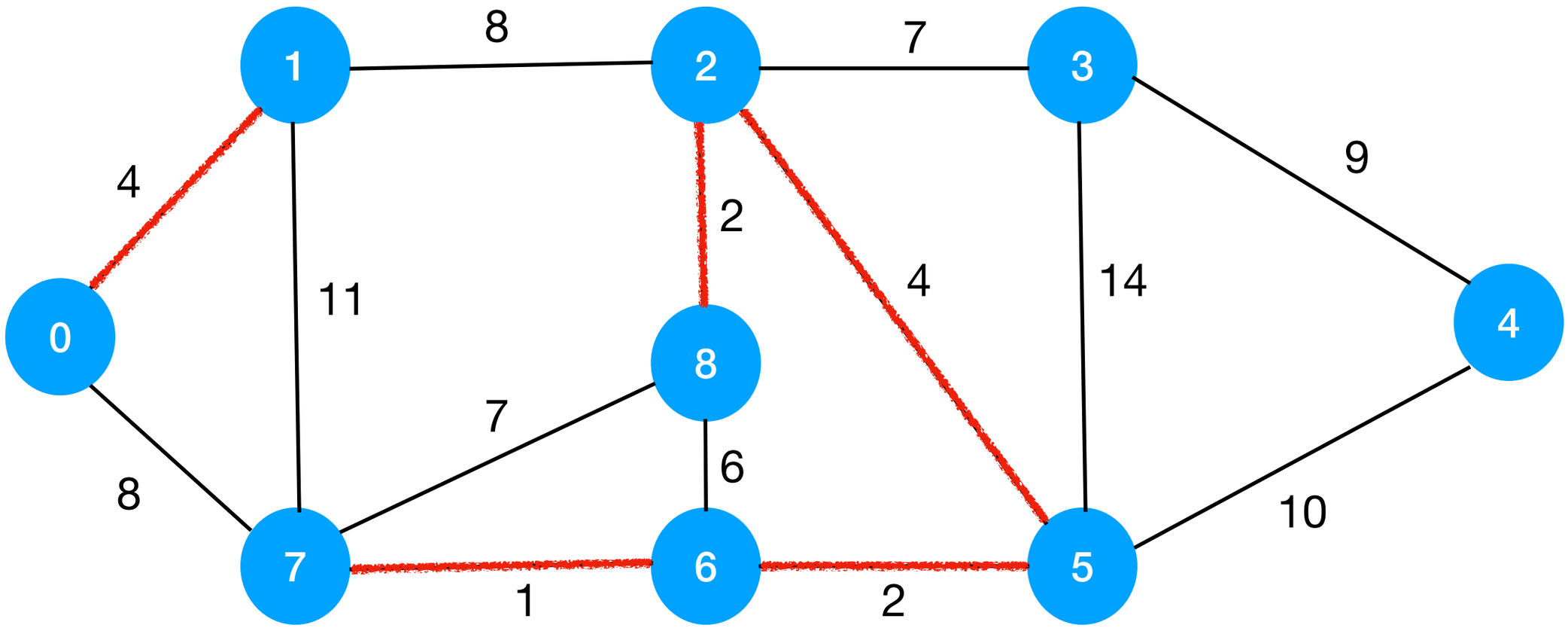
# Example



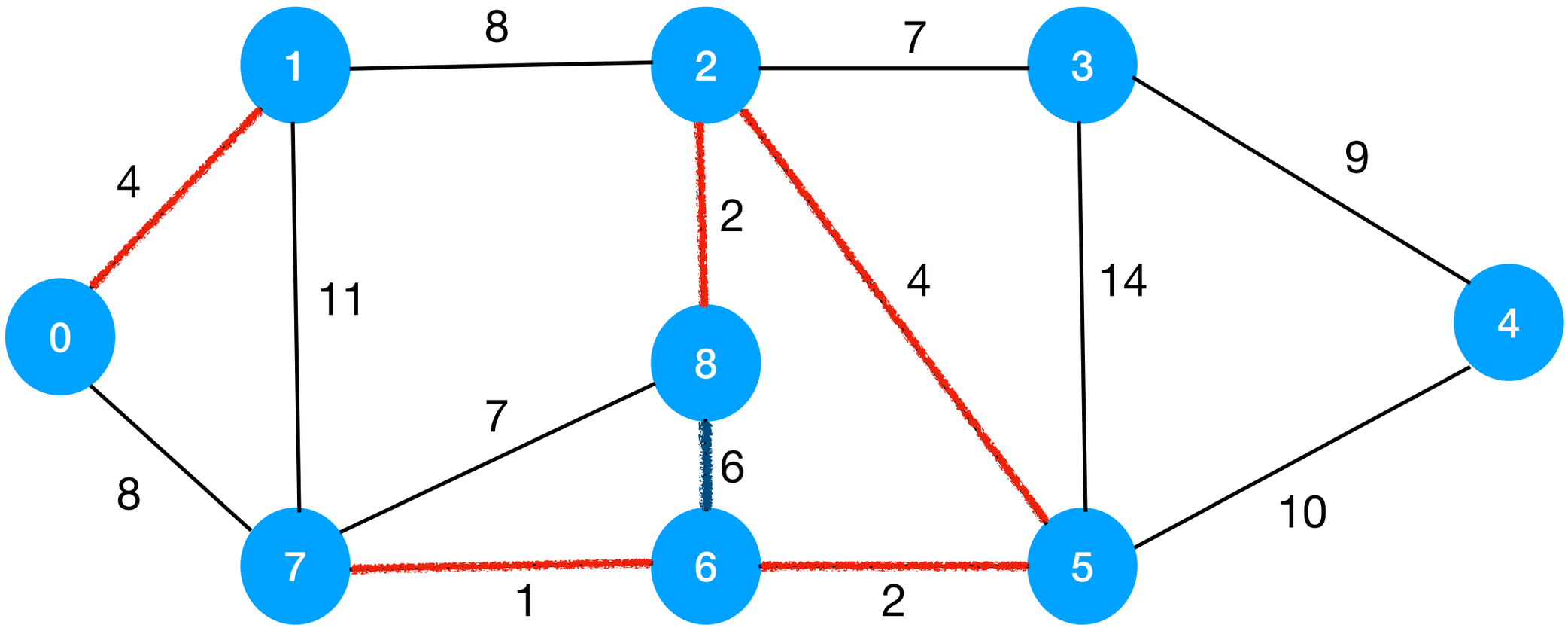
# Example



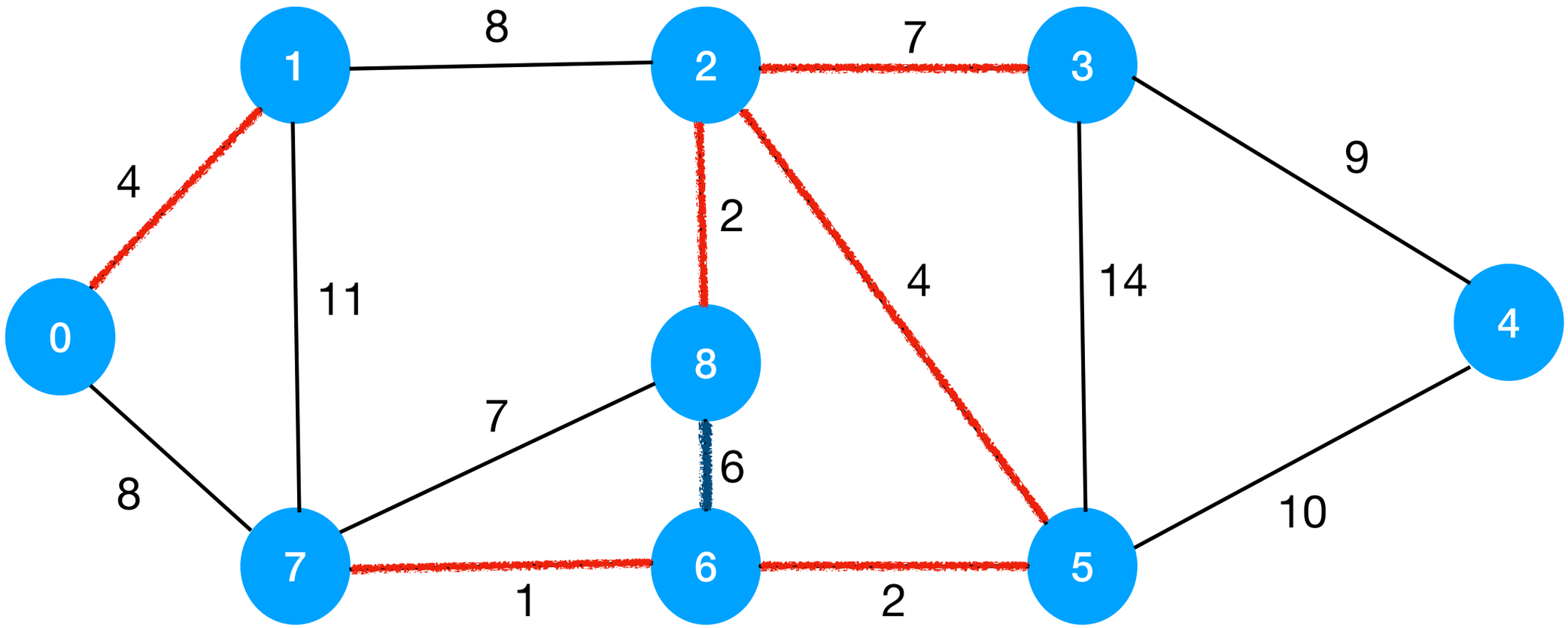
# Example



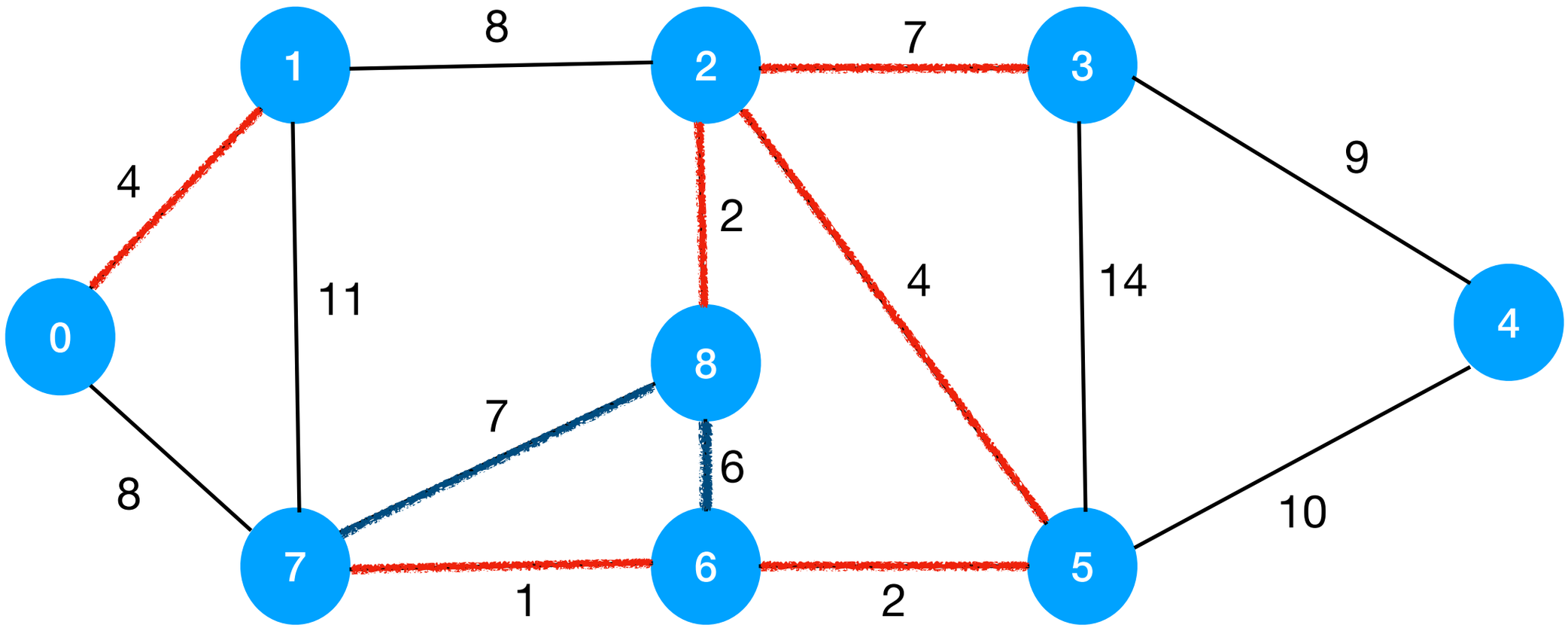
# Example



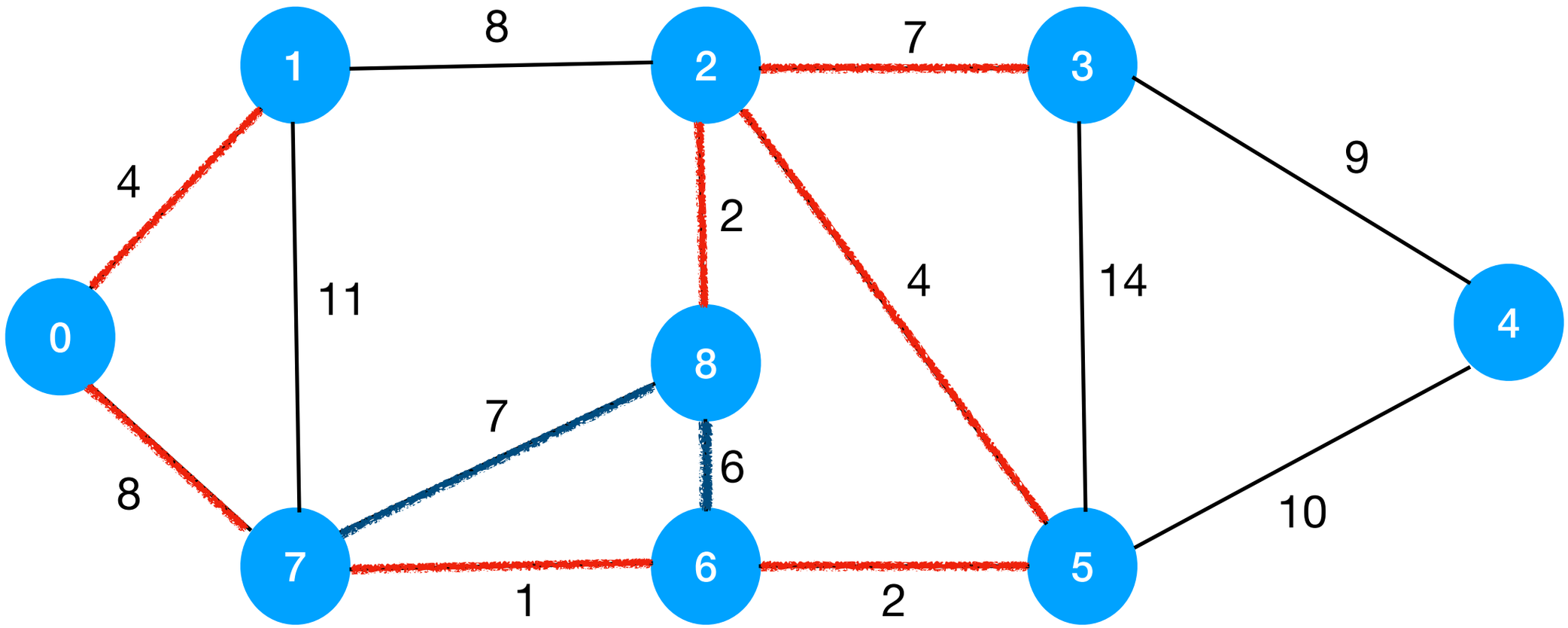
# Example



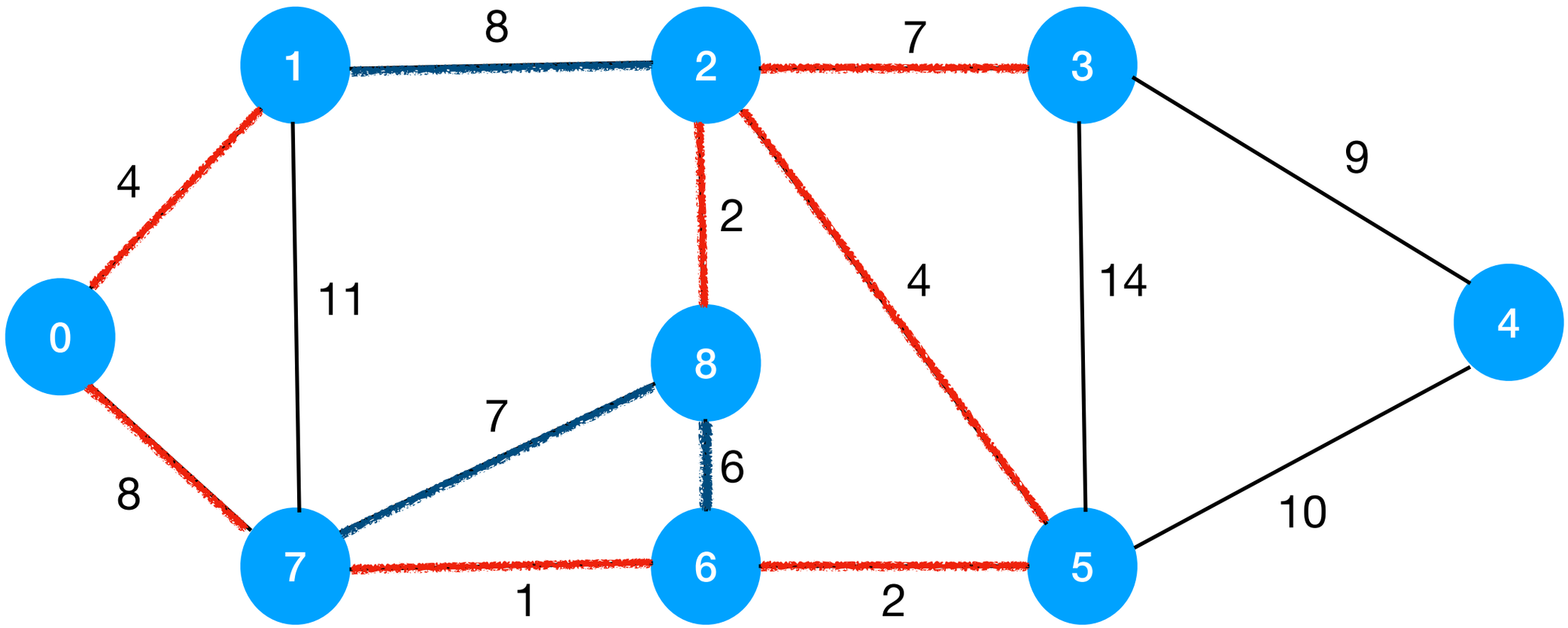
# Example



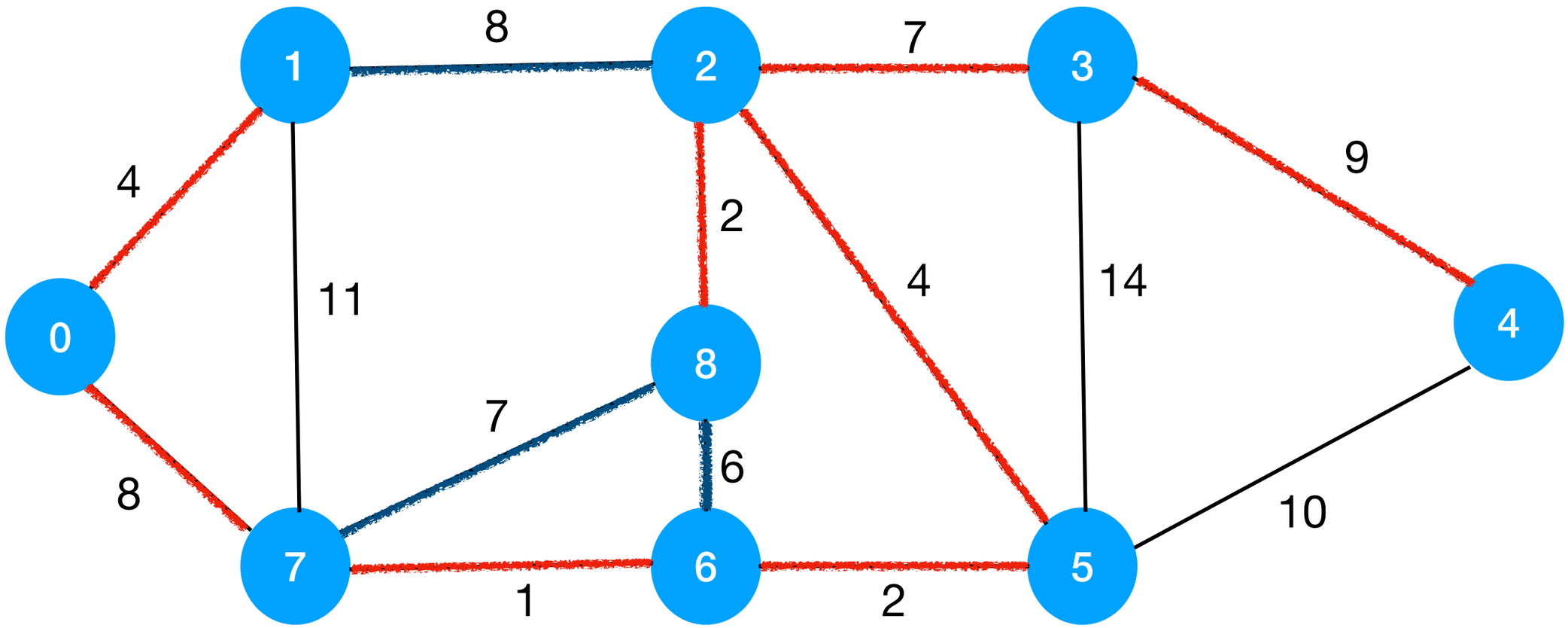
# Example



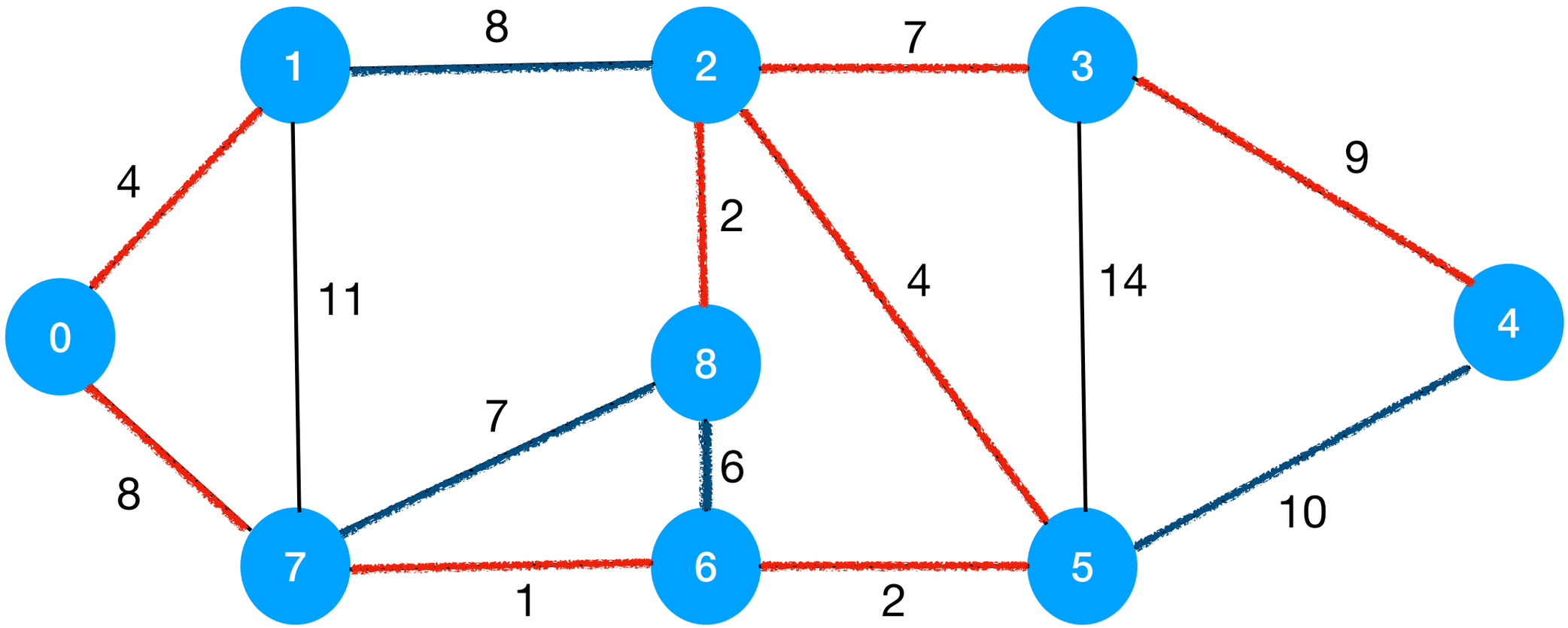
# Example



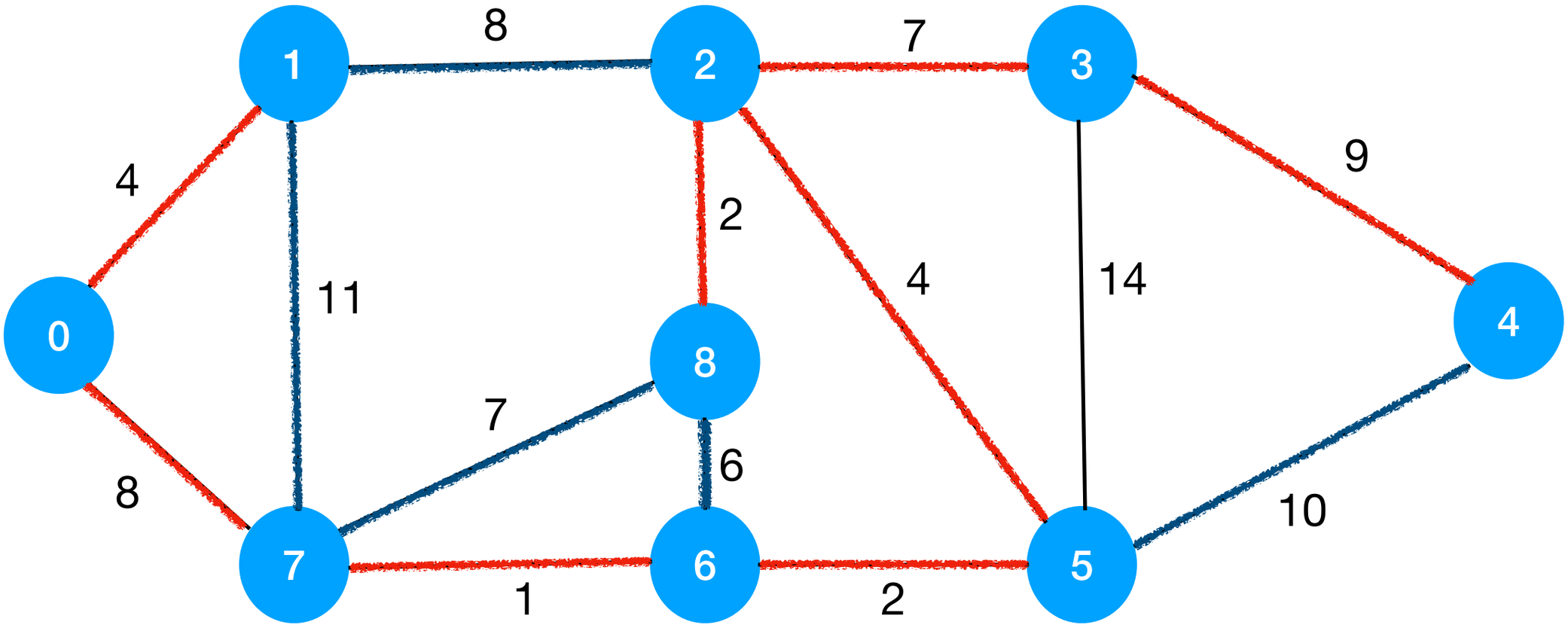
# Example



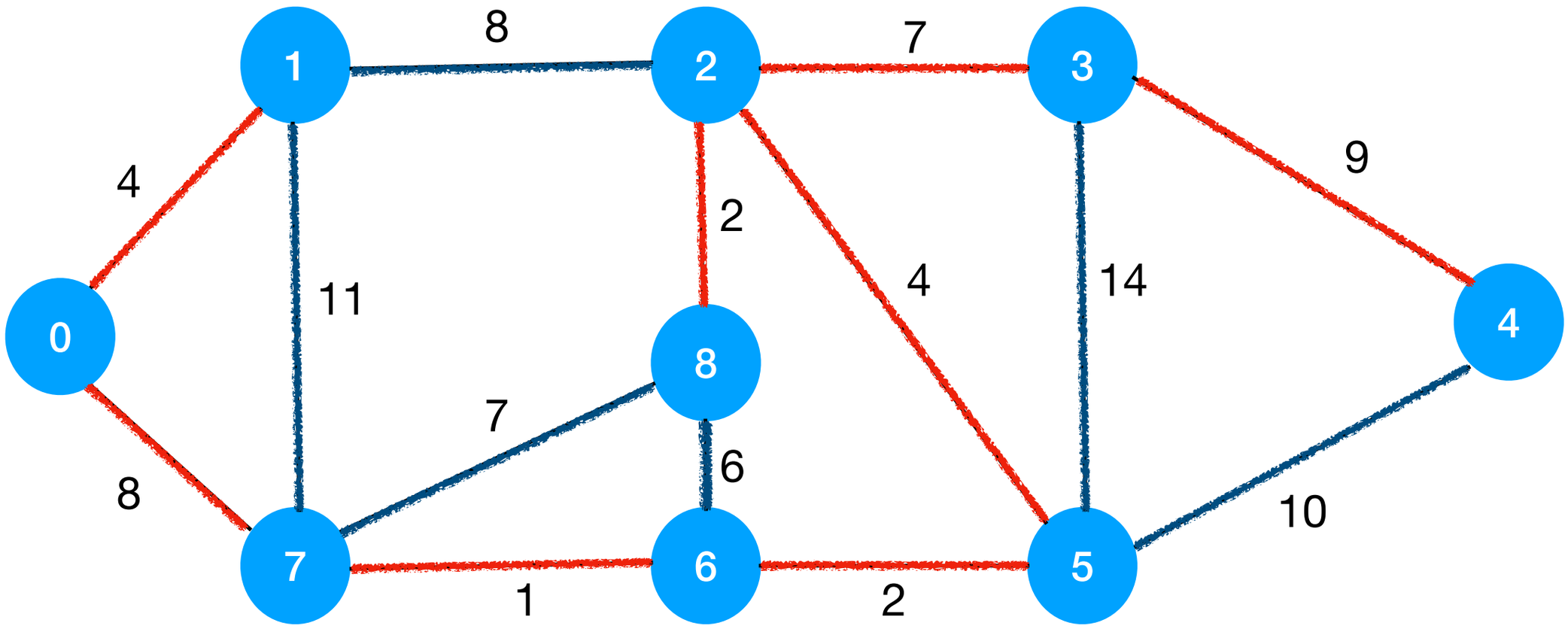
# Example



# Example



# Example



# Greedy Approach #1

# 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?

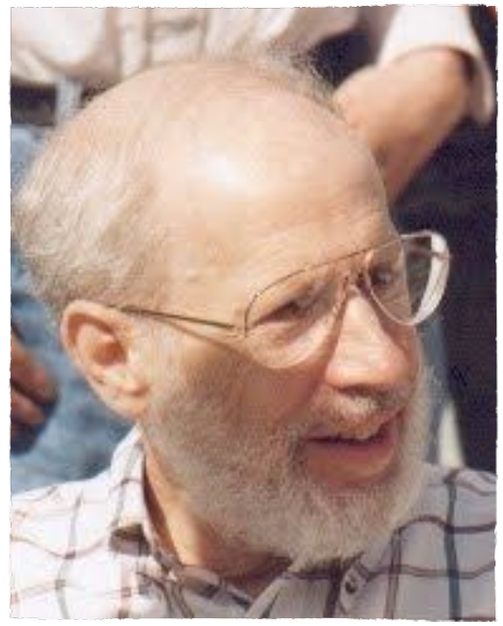
The one with the minimum cost $c_e$.

We continue like this.

Do we always add the new edge $e$ to $T$ ?

Only if we don’t introduce any cycles.

# 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 $T$.

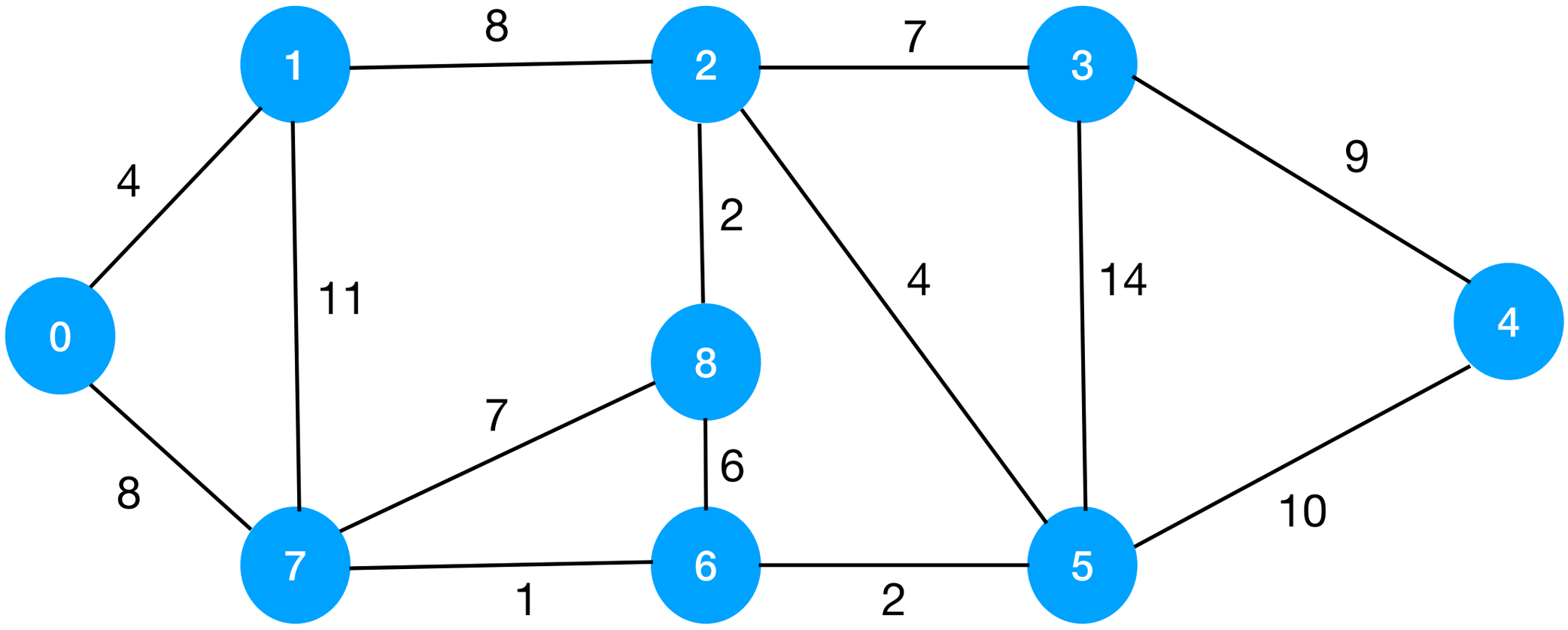
Which one?

The one with the minimum cost $c_e$.

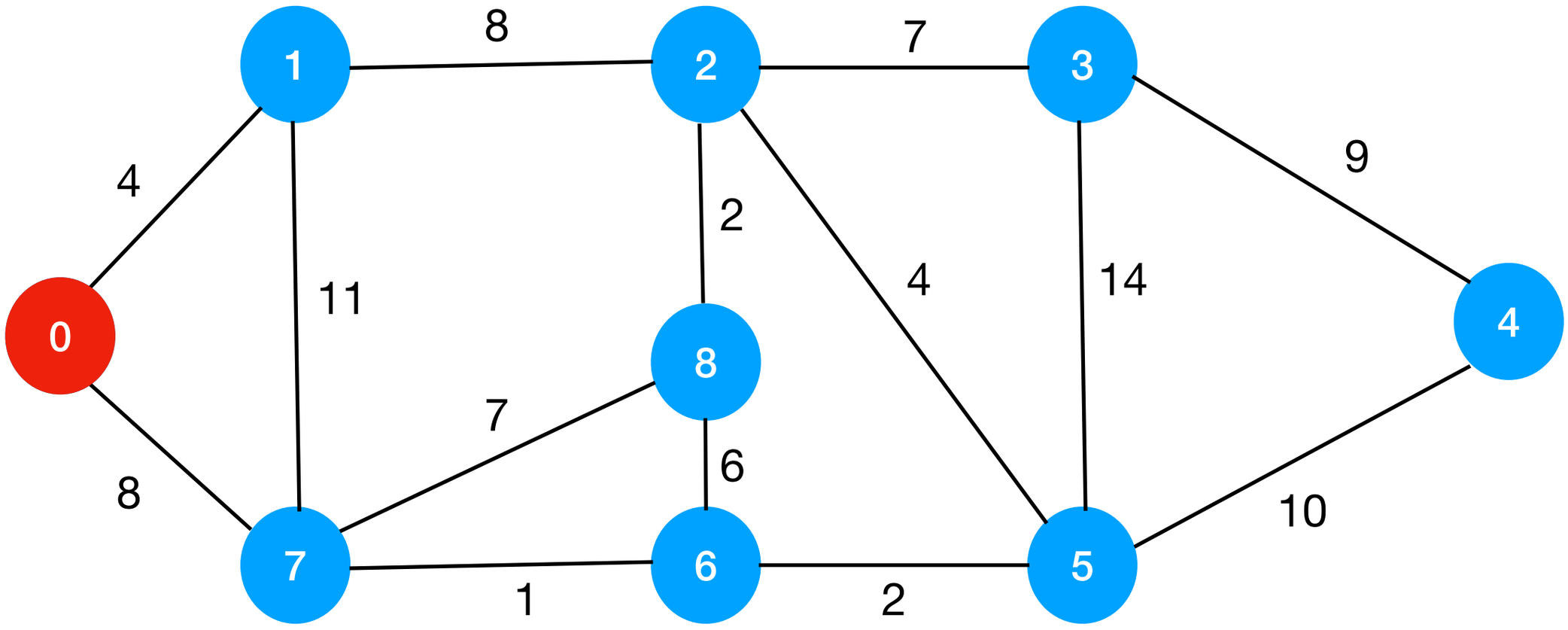
We continue like this.

We only consider edges to neighbours that are not in the spanning tree.

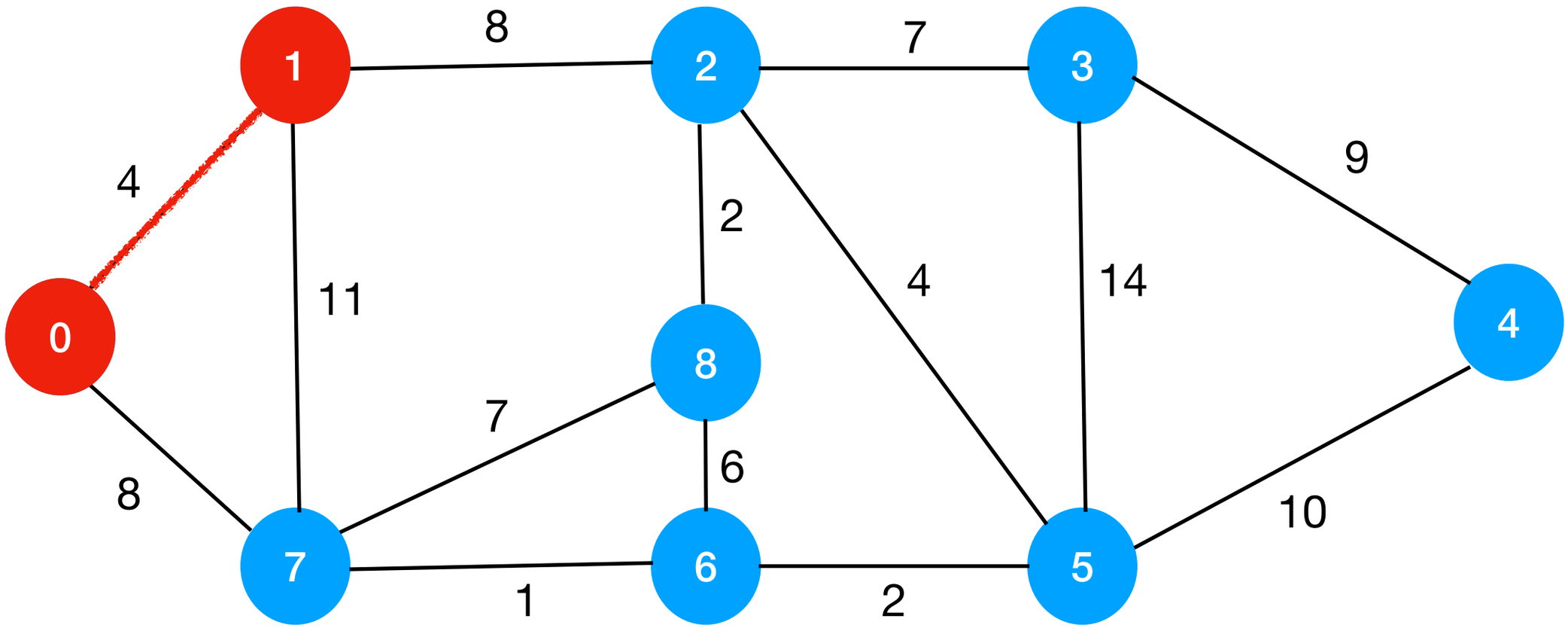
# Example



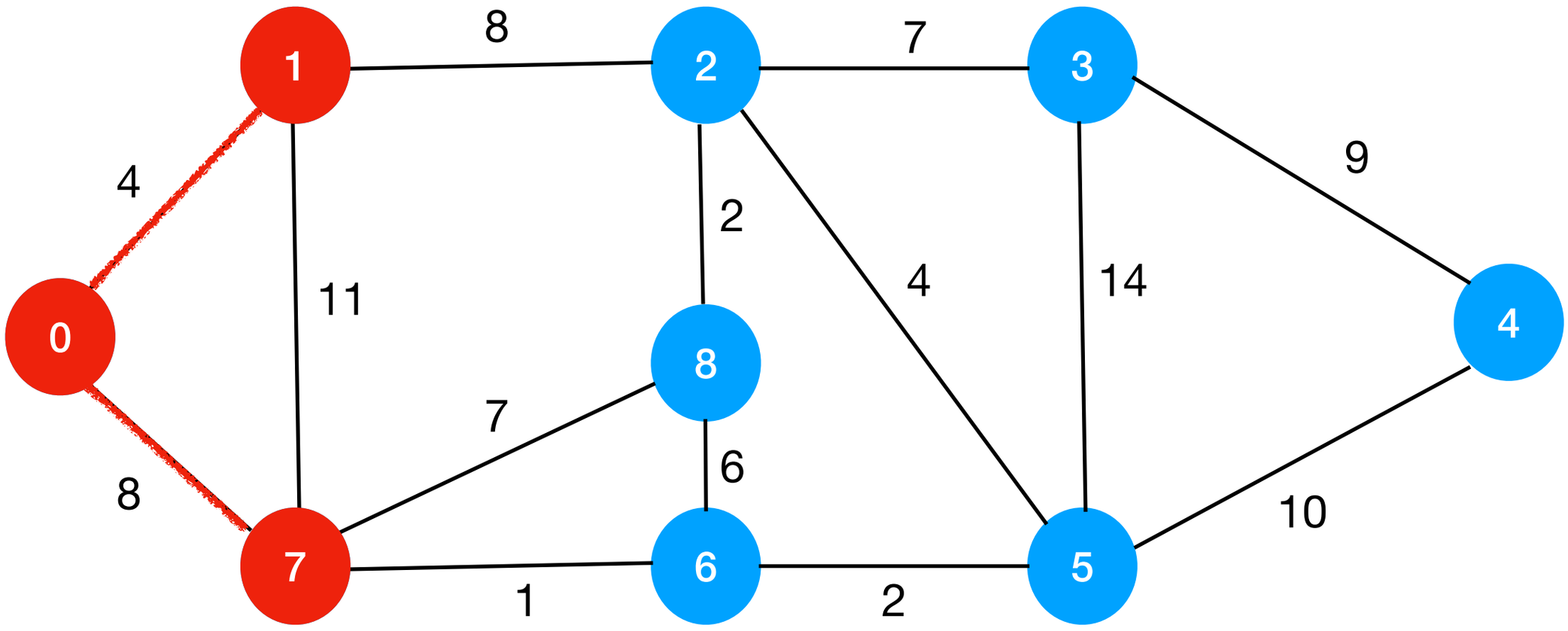
# Example



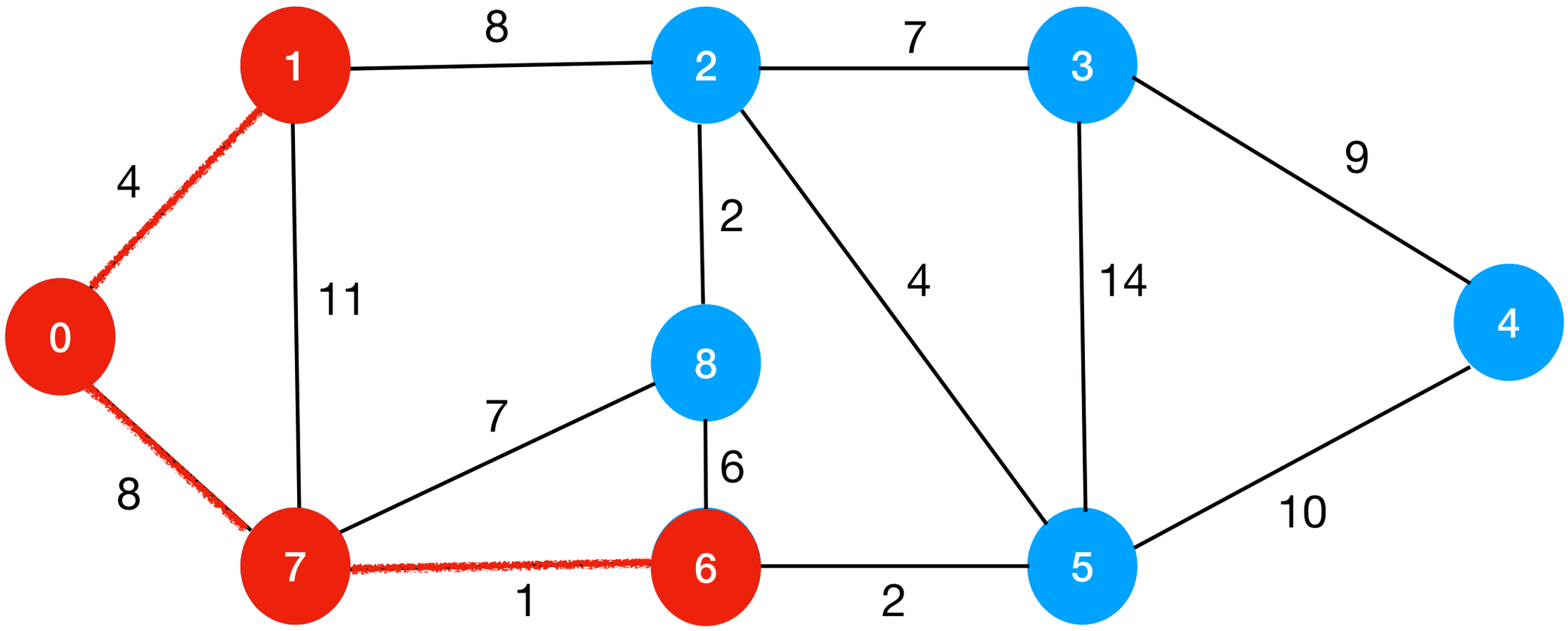
# Example



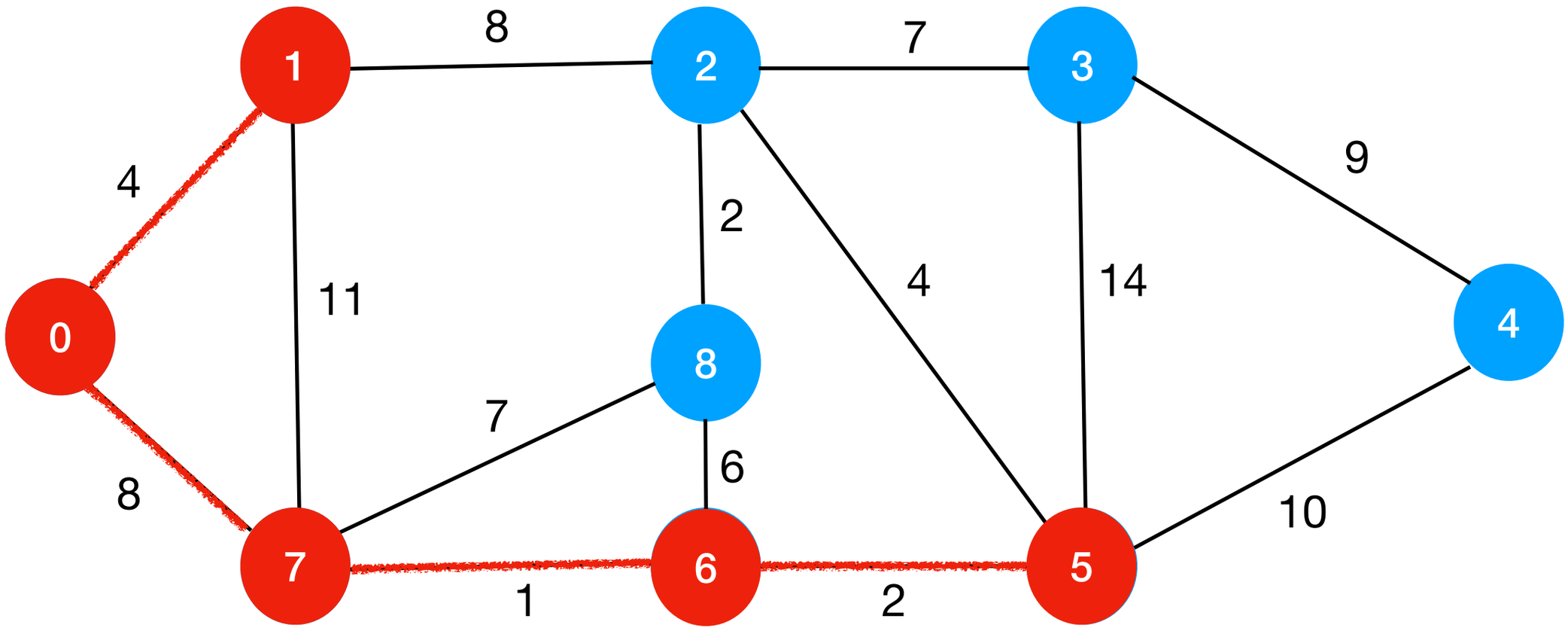
# Example



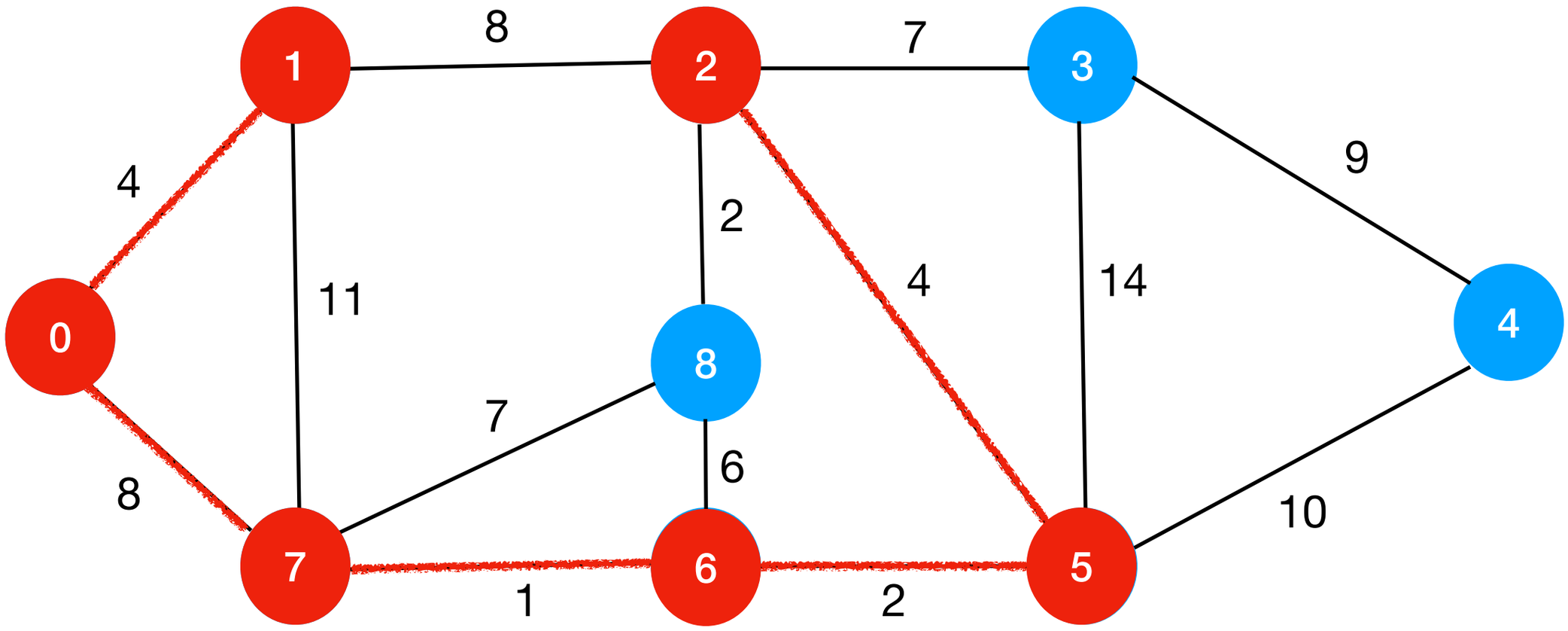
# Example



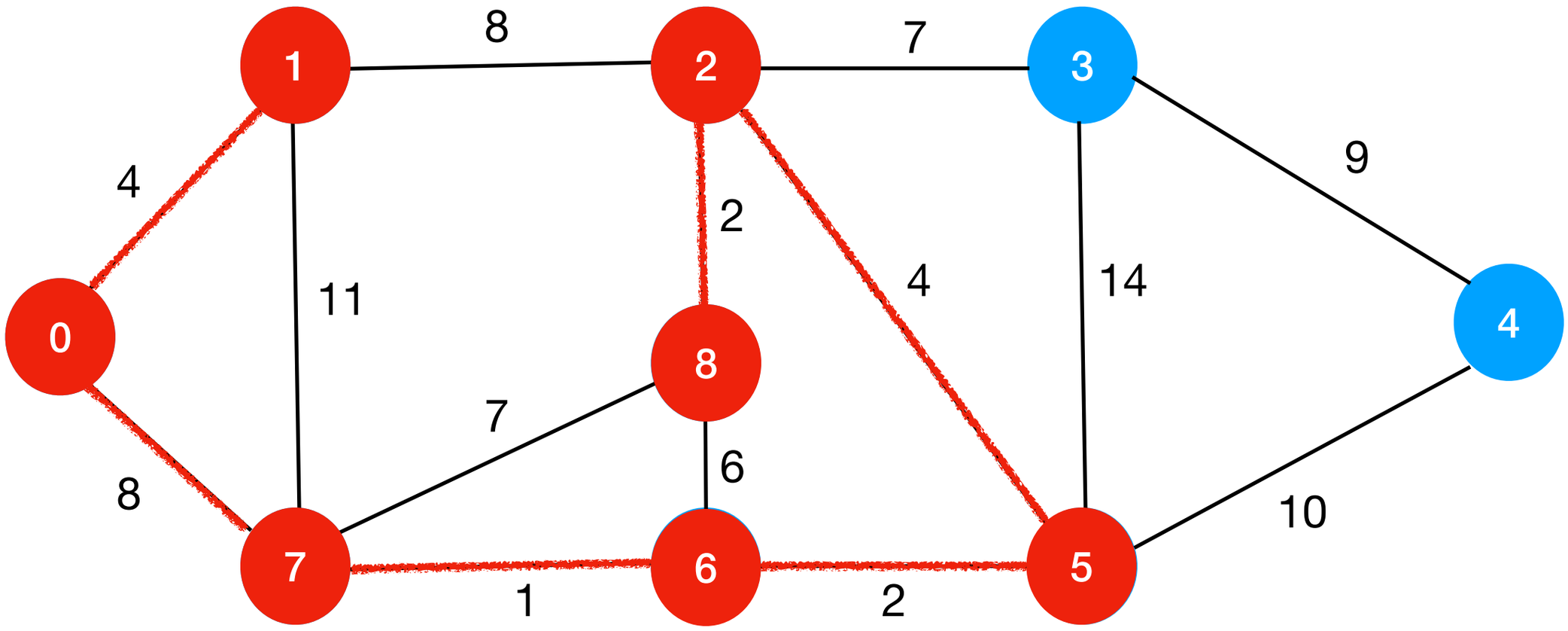
# Example



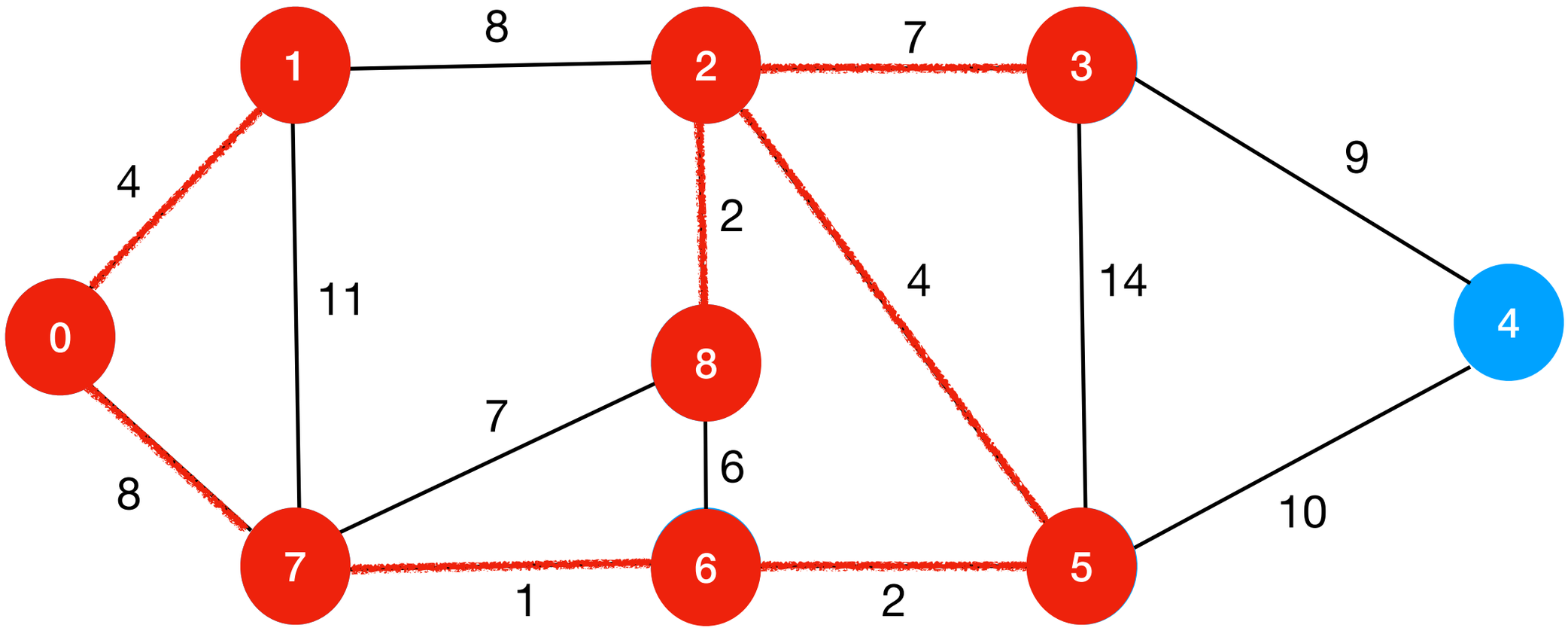
# Example



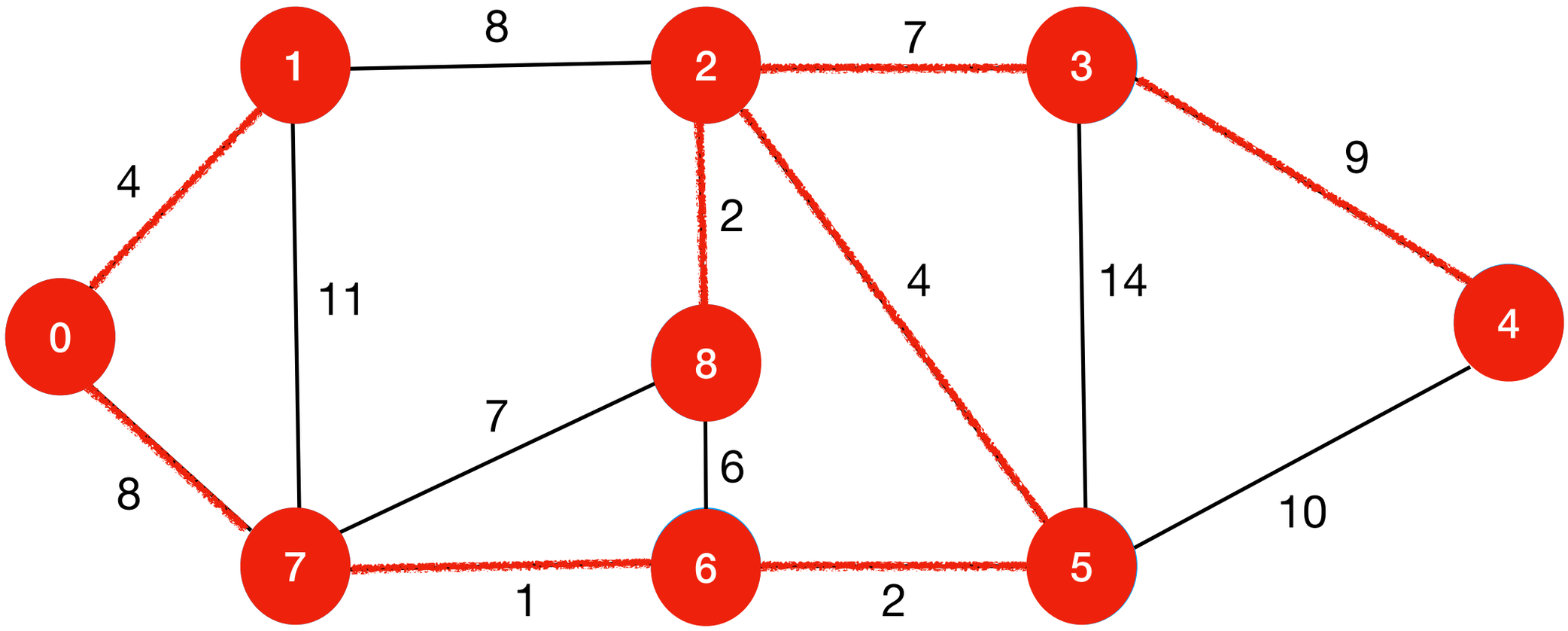
# Example



# 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.

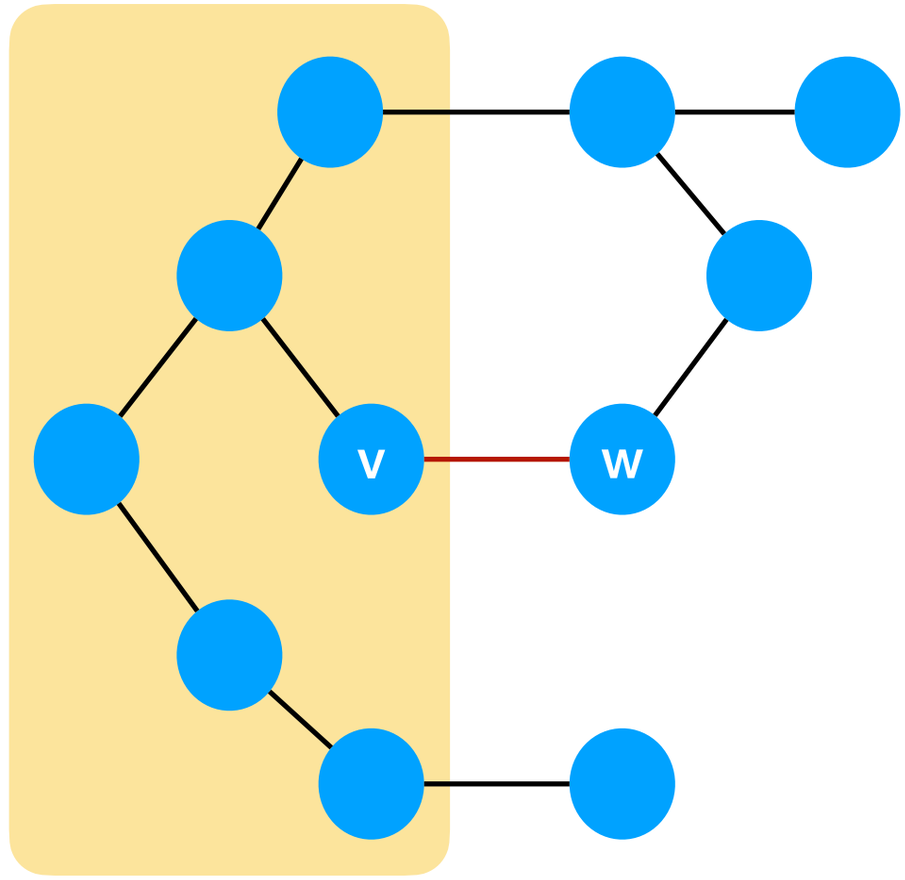
Let $S$ be any subset of $V$,

but *not empty.*

but *not* $V$.

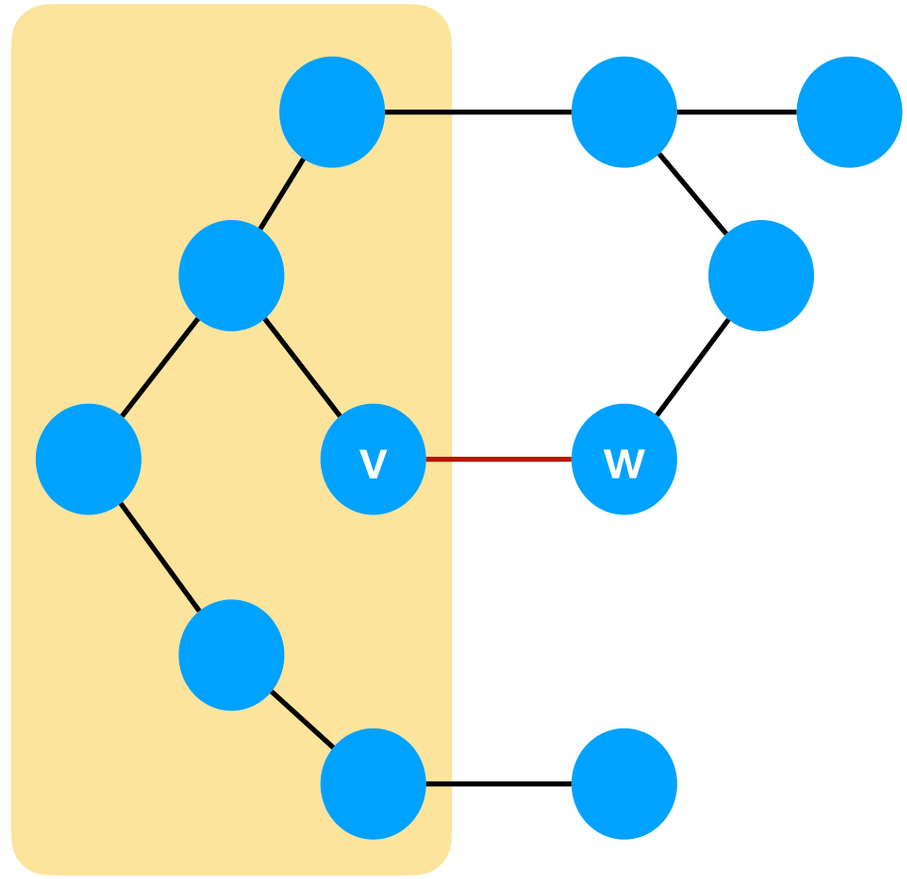
Let $e=\{w,v\}$ be the minimum cost edge between $S$ and $V$-$S$.

Then $e$ is contained in every minimum spanning tree.



# The cut property

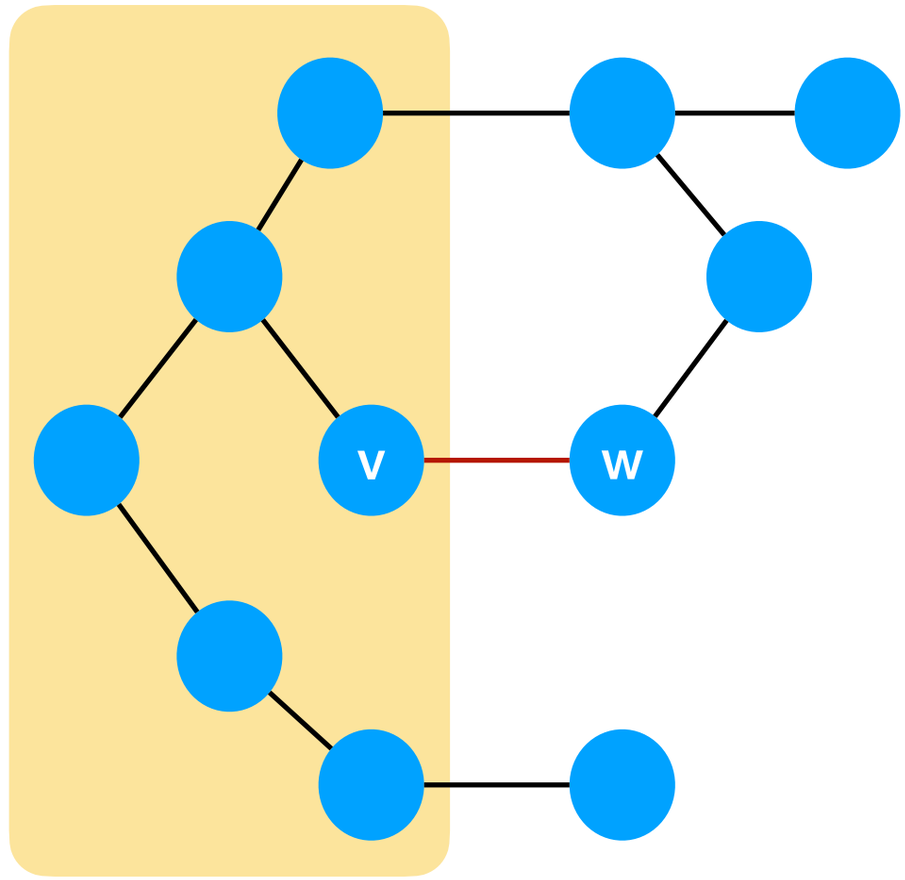
Then $e$ is contained in every minimum spanning tree.



# The cut property

Then $e$ is contained in every minimum spanning tree.

Assume that some spanning tree $G_T$ does not contain $e$.

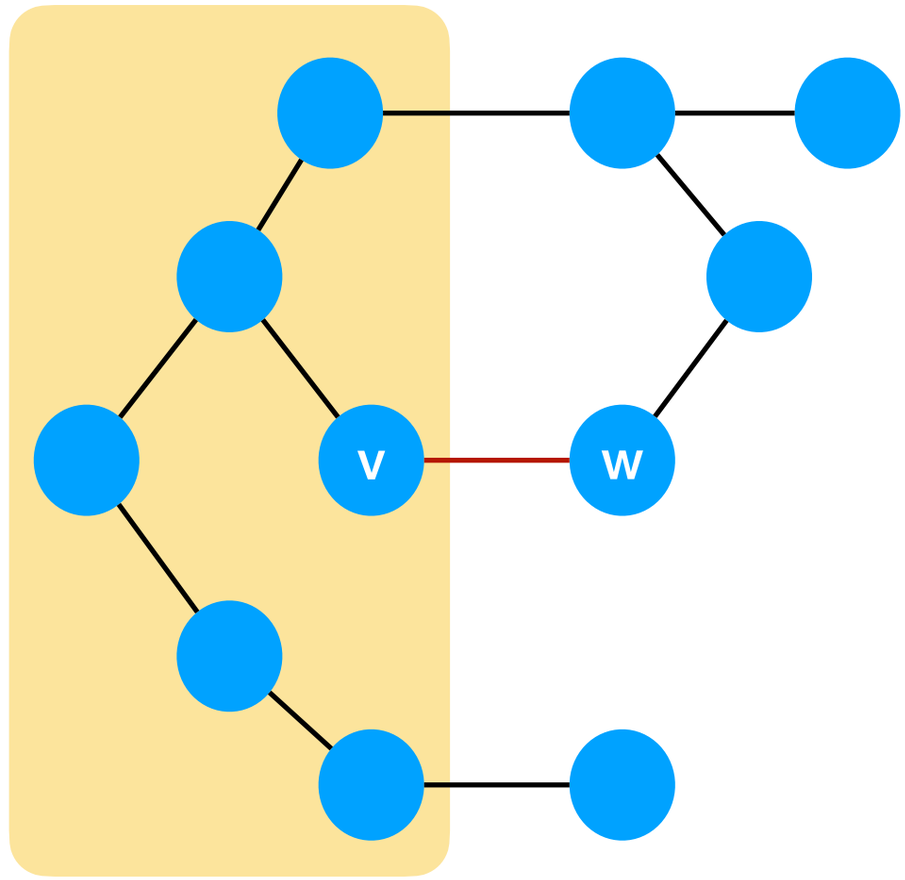


# The cut property

Then $e$ is contained in every minimum spanning tree.

Assume that some spanning tree $G_T$ does not contain $e$.

Since it is a spanning tree, it must contain some other edge $f$ that *crosses* from $S$ to $V$-$S$.

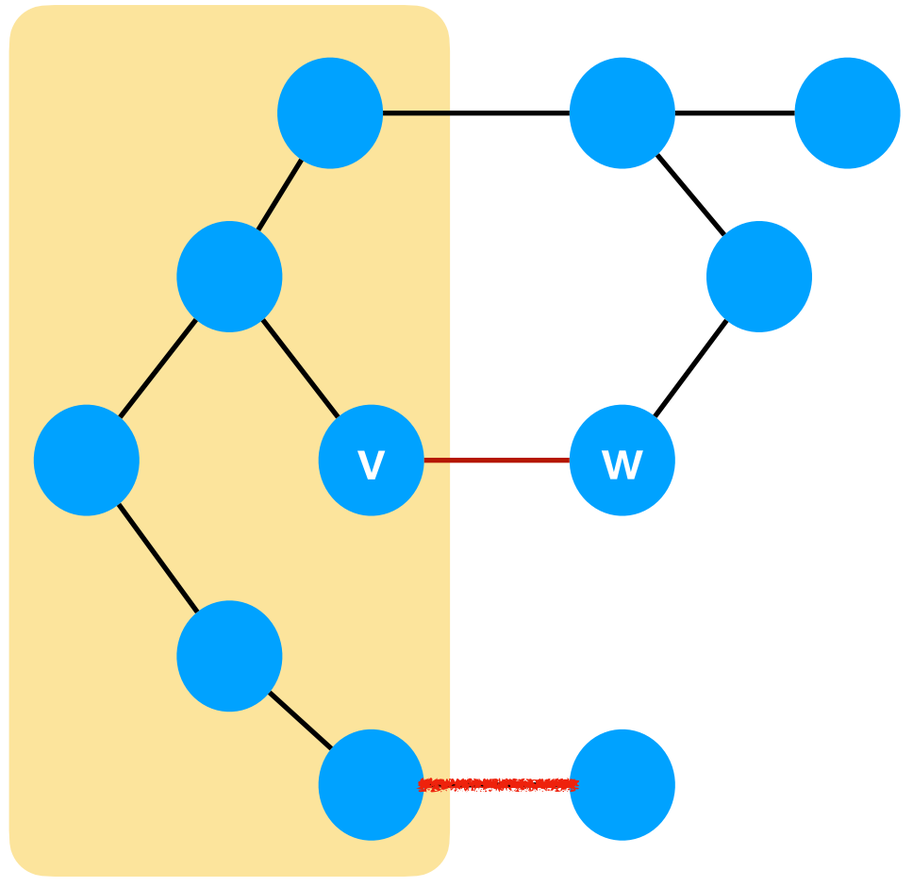


# The cut property

Then $e$ is contained in every minimum spanning tree.

Assume that some spanning tree $G_T$ does not contain $e$.

Since it is a spanning tree, it must contain some other edge $f$ that *crosses* from $S$ to $V$-$S$.



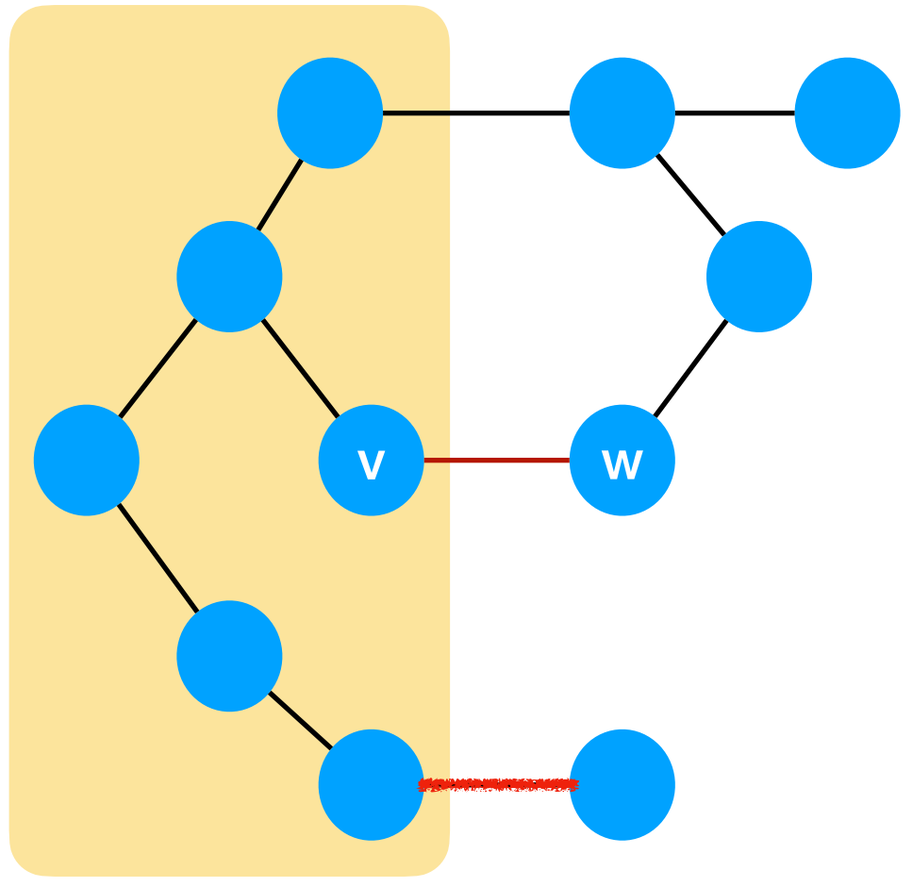
# The cut property

Then $e$ is contained in every minimum spanning tree.

Assume that some spanning tree $G_T$ 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 $G_T$ - { $f$ } ∪ { $e$ } is a spanning tree of smaller cost.



# The cut property

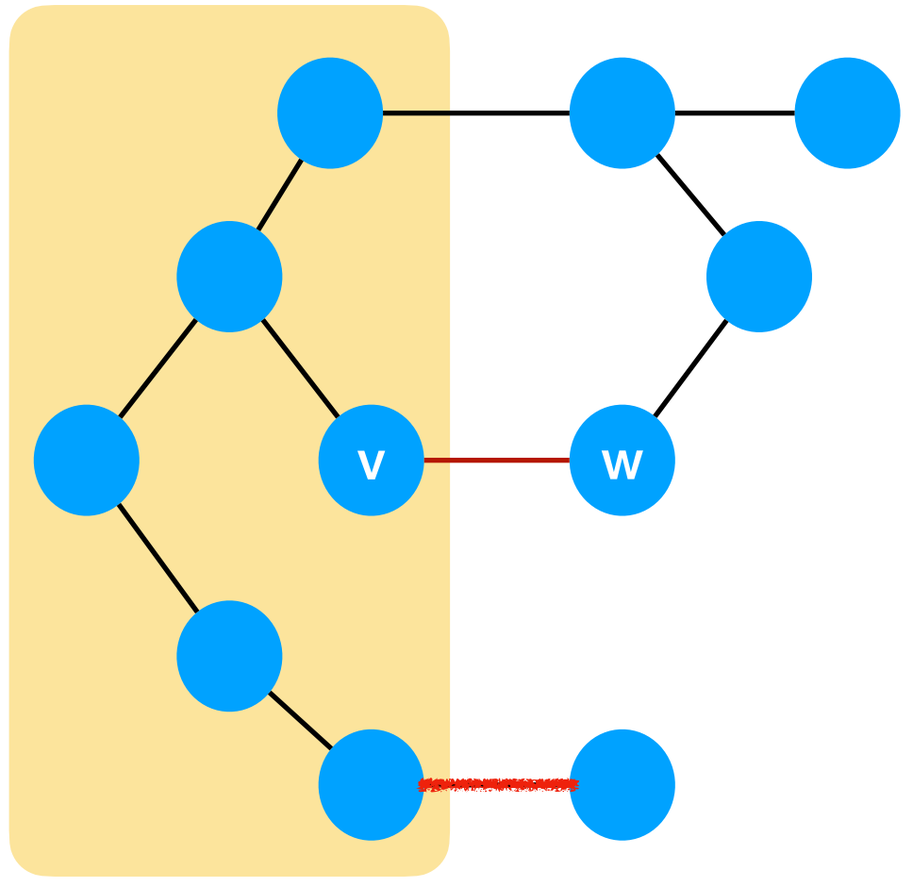
**No, $G_T$ - { $f$ } U { $e$ } might not be a spanning tree!**

Then $e$ is contained in every minimum spanning tree.

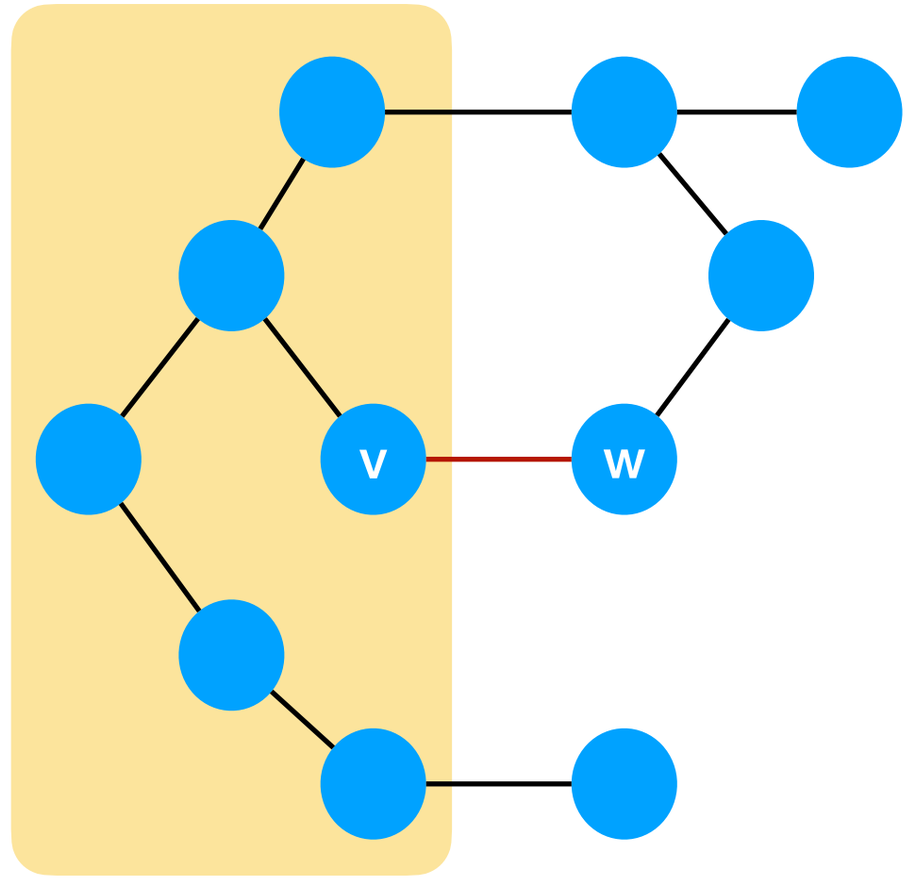
Assume that some spanning tree $G_T$ does not contain $e$.

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 $G_T$ - { $f$ } ∪ { $e$ } is a spanning tree of smaller cost.

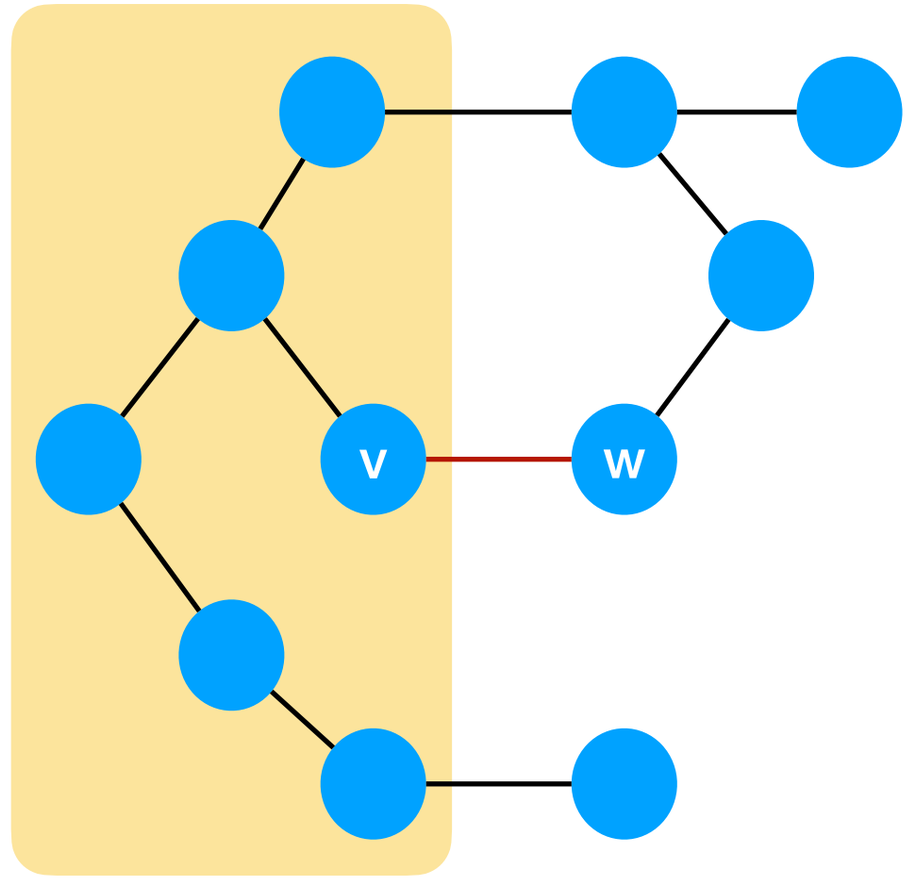


# The cut property



# The cut property

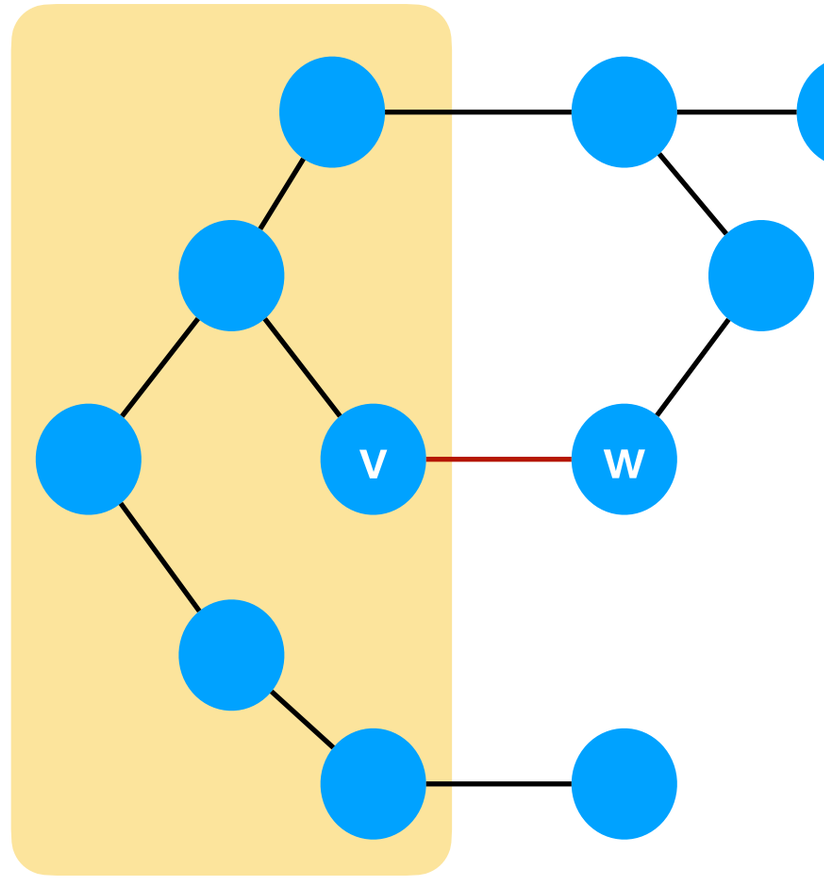
We can’t simply select any edge.



# The cut property

We can’t simply select any edge.

We need to select an edge $e'$ which

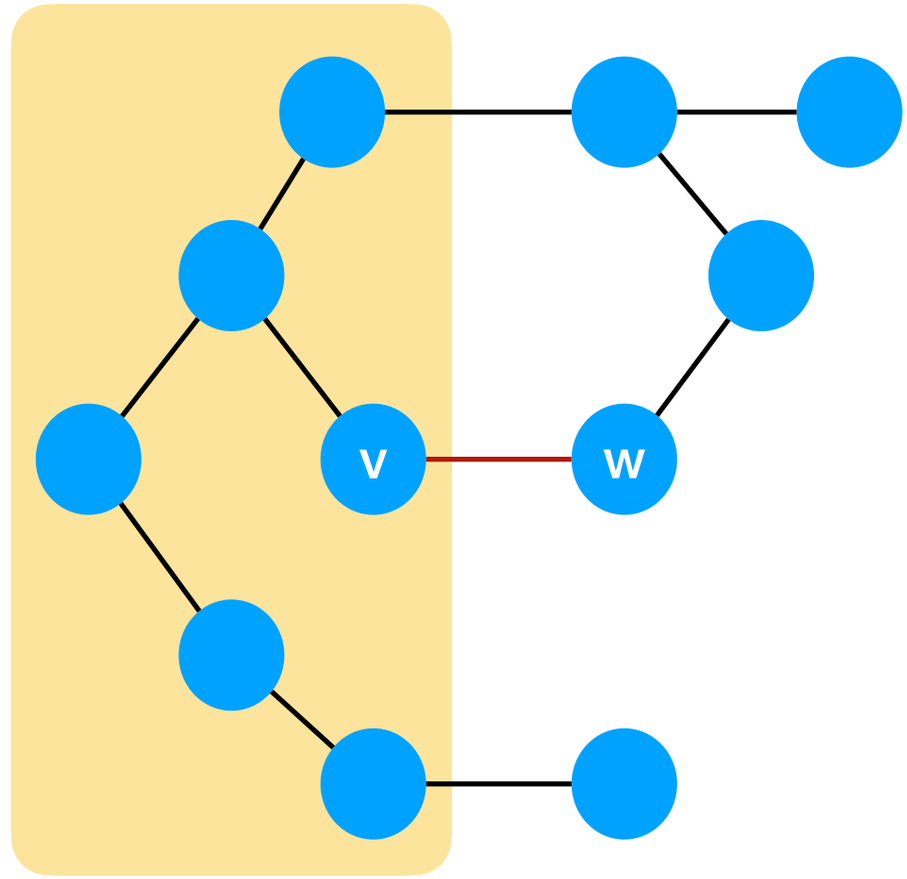


# The cut property

We can’t simply select any edge.

We need to select an edge $e'$ which

- is more expensive than $e$.

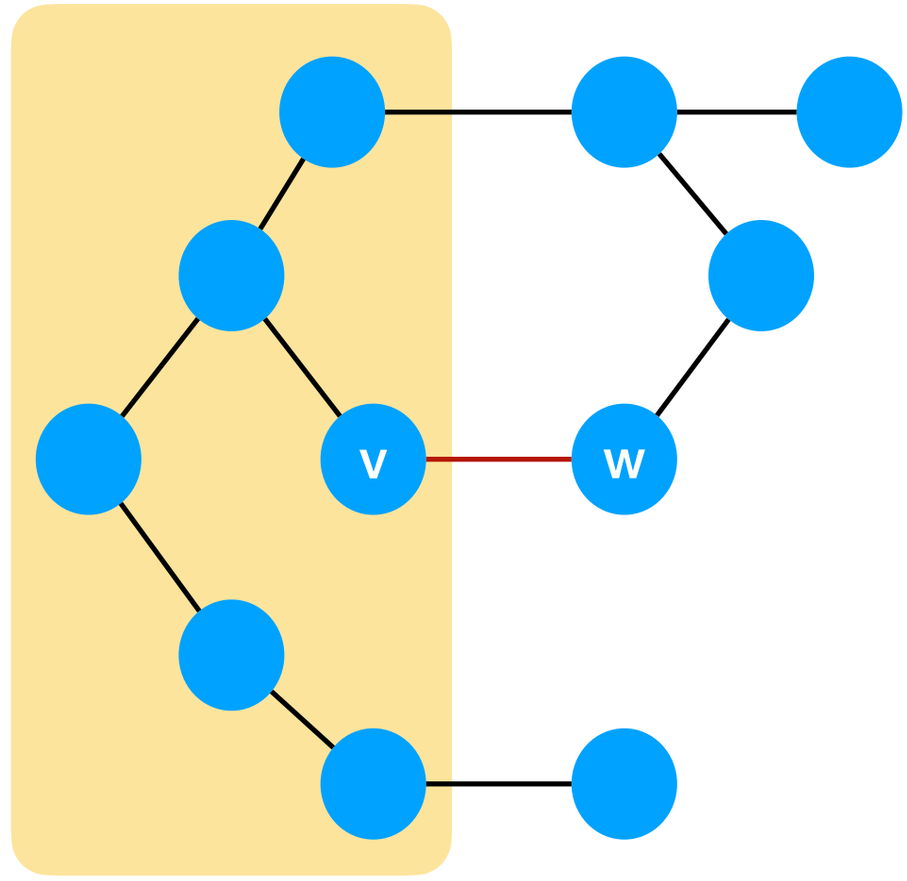


# The cut property

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 $e$.

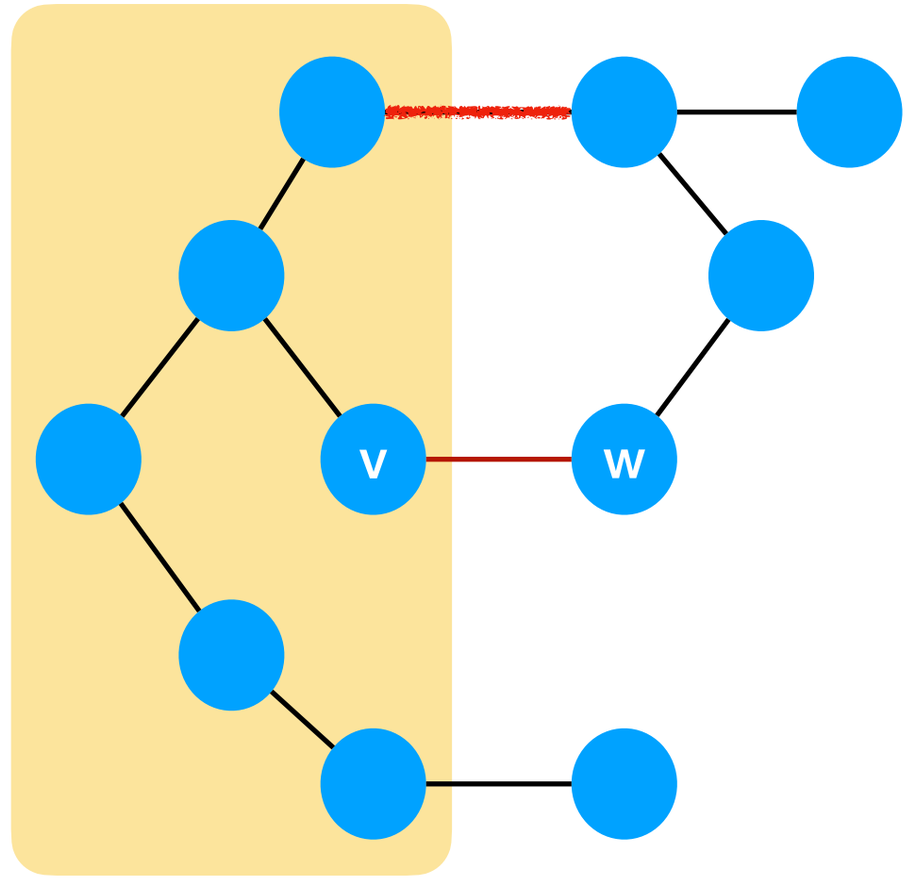


# The cut property

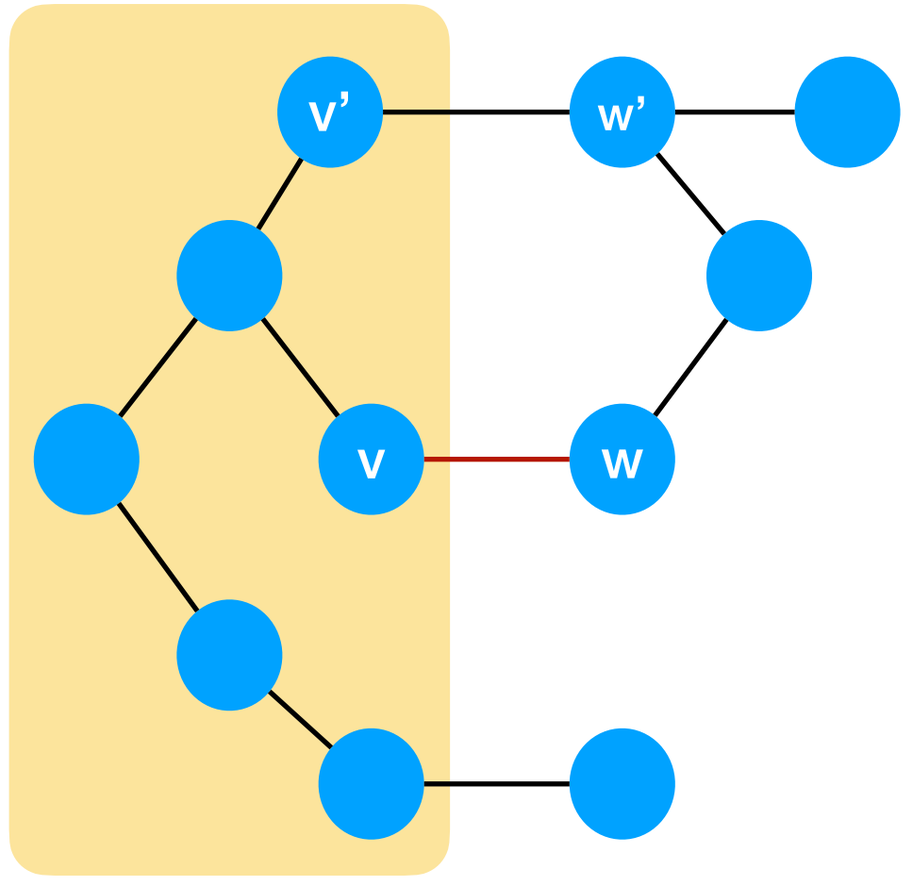
We can’t simply select any edge.

We need to select an edge $e'$ which

- is more expensive than $e$.
- still results in a spanning tree, if used instead of $e$.

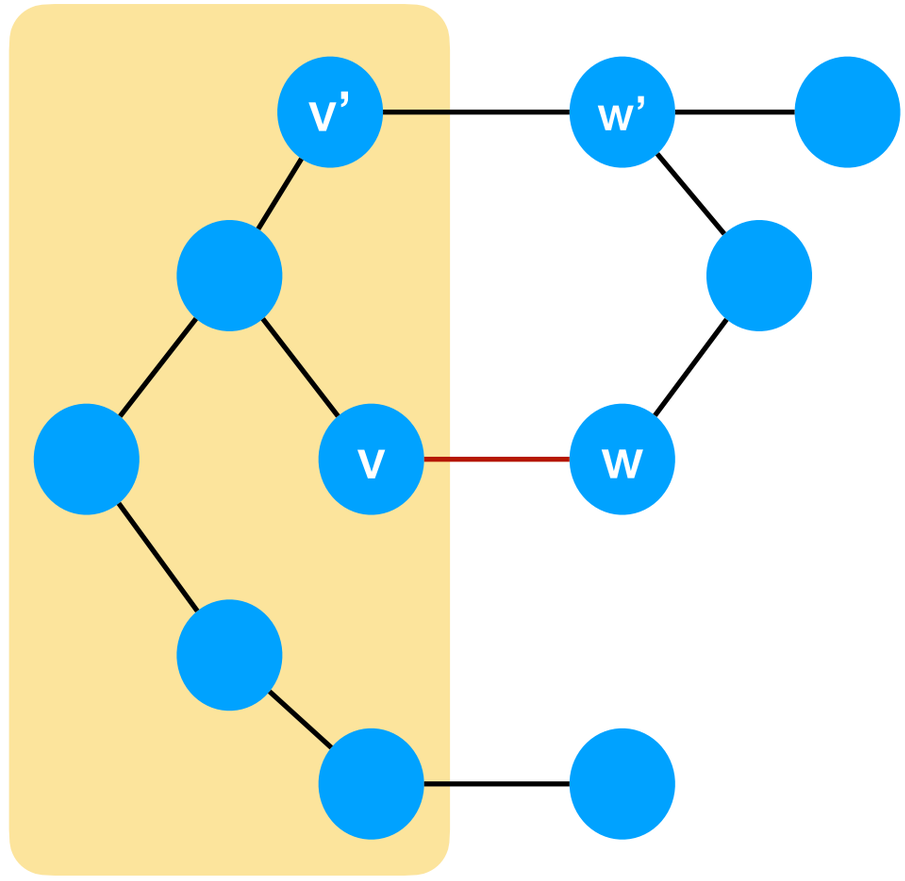


# The cut property



# The cut property

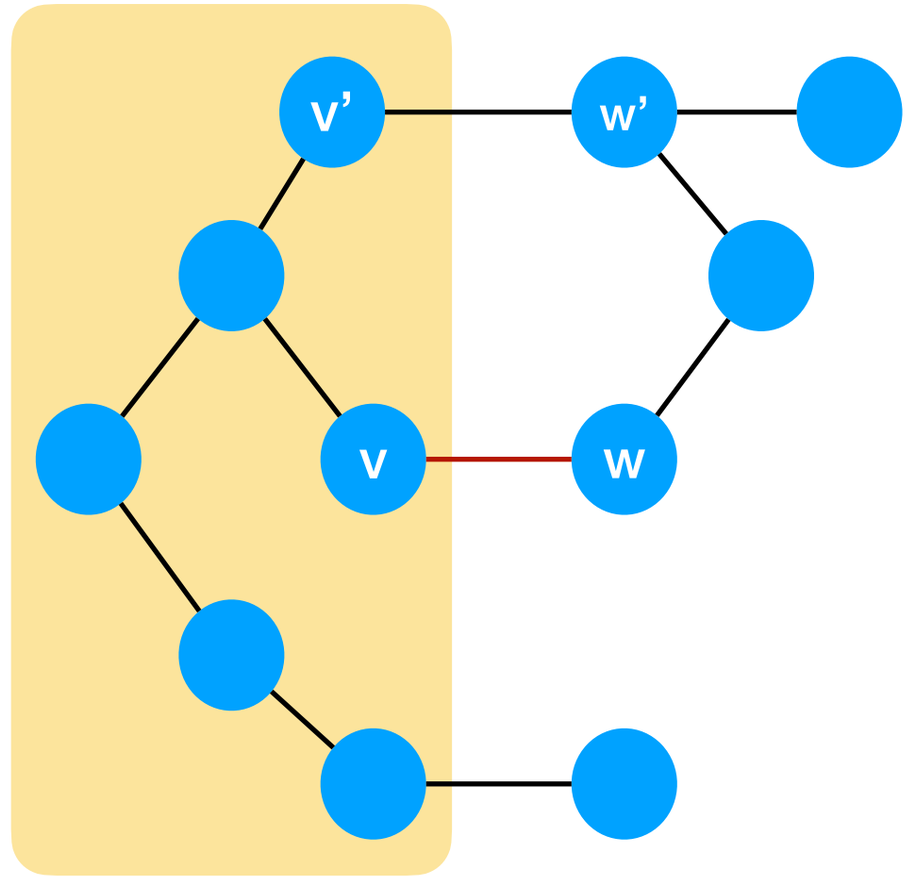
Let $G_T$ be a minimum spanning tree which does ***not*** contain $e=\{v,w\}$.



# The cut property

Let $G_T$ be a minimum spanning tree which does ***not*** contain $e=\{v,w\}$.

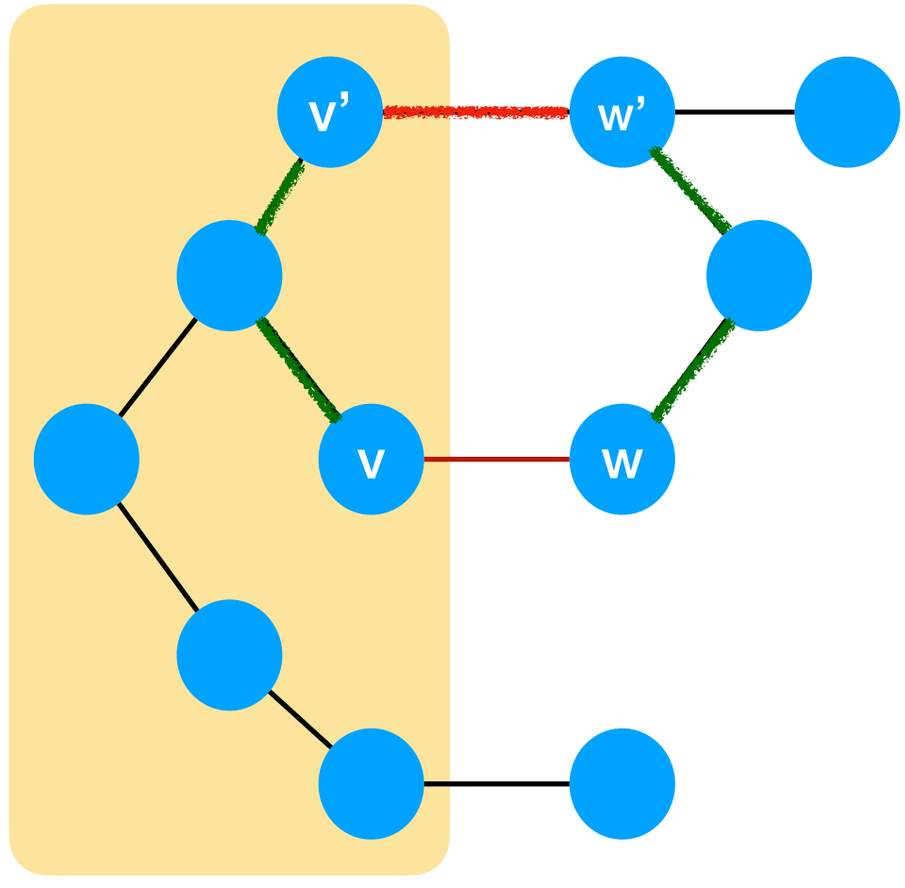
Since $G_T$ is a spanning tree, there is path from $v$ to $w$.



# The cut property

Let $G_T$ be a minimum spanning tree which does ***not*** contain $e=\{v,w\}$.

Since $G_T$ is a spanning tree, there is path from $v$ to $w$.

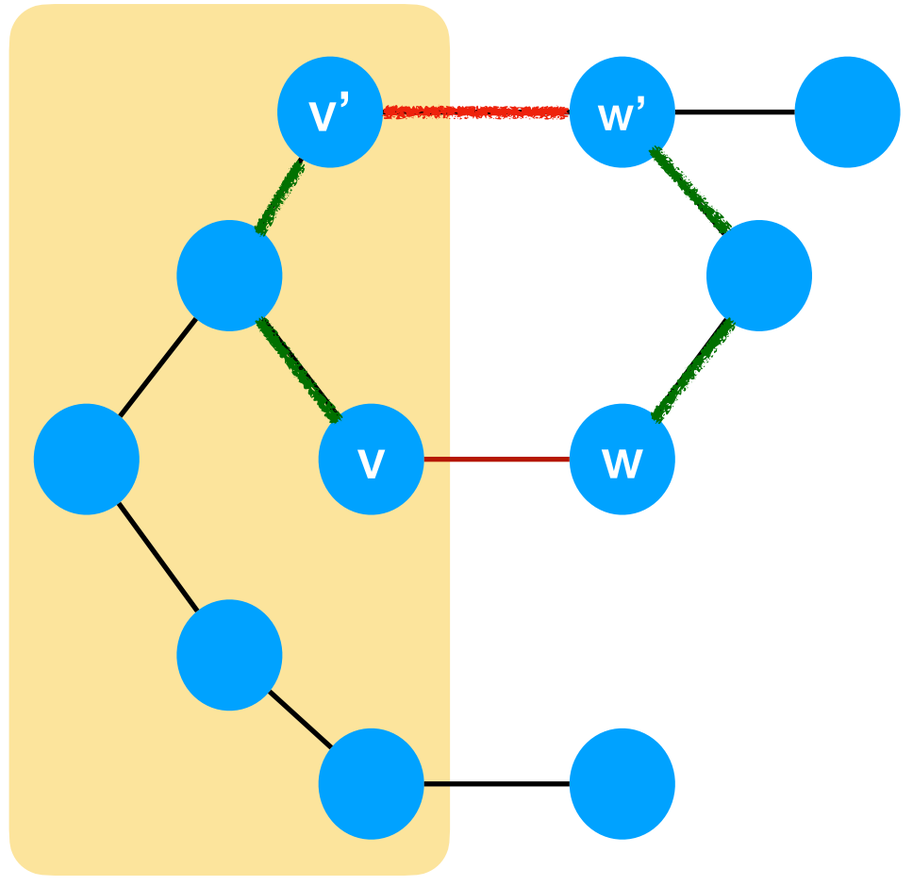


# The cut property

Let $G_T$ be a minimum spanning tree which does ***not*** contain $e=\{v,w\}$.

Since $G_T$ is a spanning tree, there is path from $v$ to $w$.

Let $w'$ be the first node encountered on this path in $V$-$S$ and let $v'$ be the one before it. Let $e'=\{v',w'\}$.



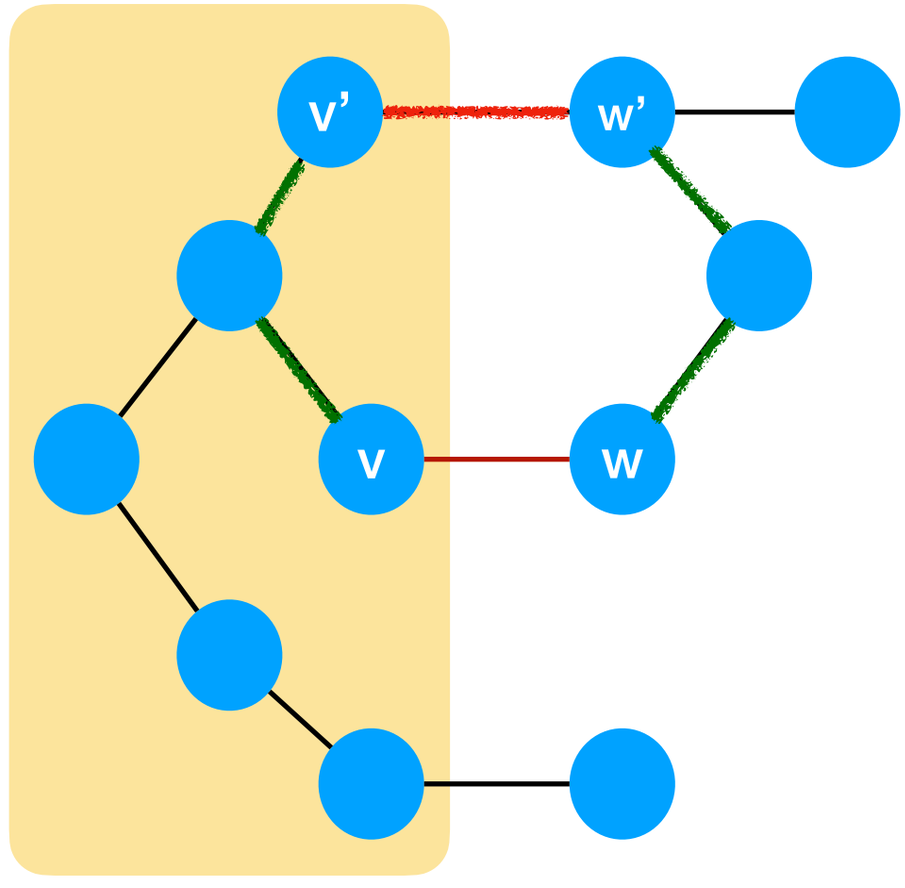
# The cut property

Let $G_T$ be a minimum spanning tree which does ***not*** contain $e=\{v,w\}$.

Since $G_T$ is a spanning tree, there is path from $v$ to $w$.

Let $w'$ be the first node encountered on this path in $V$-$S$ and let $v'$ be the one before it. Let $e'=\{v',w'\}$.

Consider $G'_T = G_T - \{ e' \} \cup \{ e \}$.



# 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 $u$ 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 $u$ 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 $u$ is in $S$ and $w$ is in $V$-$S$. (Why?)

Because otherwise adding $e$ would create a cycle.

The algorithm has not before encountered any other edge crossing $S$ and $V$-$S$. (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 $u$ is in $S$ and $w$ is in $V$-$S$. (Why?)

Because otherwise adding $e$ would create a cycle.

The algorithm has not before encountered any other edge crossing $S$ and $V$-$S$. (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 $u$ is in $S$ and $w$ is in $V$-$S$. (Why?)

Because otherwise adding $e$ would create a cycle.

The algorithm has not before encountered any other edge crossing $S$ and $V$-$S$. (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 $u$ is in $S$ and $w$ is in $V$-$S$. (Why?)

Because otherwise adding $e$ would create a cycle.

The algorithm has not before encountered any other edge crossing $S$ and $V$-$S$. (Why?)

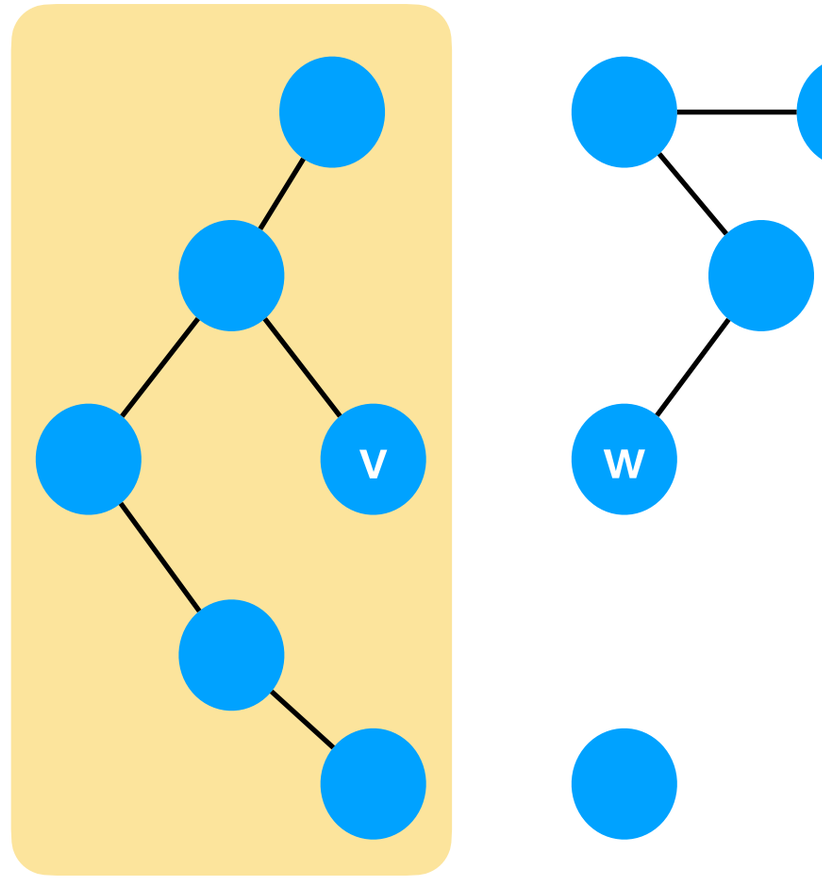
Such an edge would have been added to the output by the algorithm.

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?

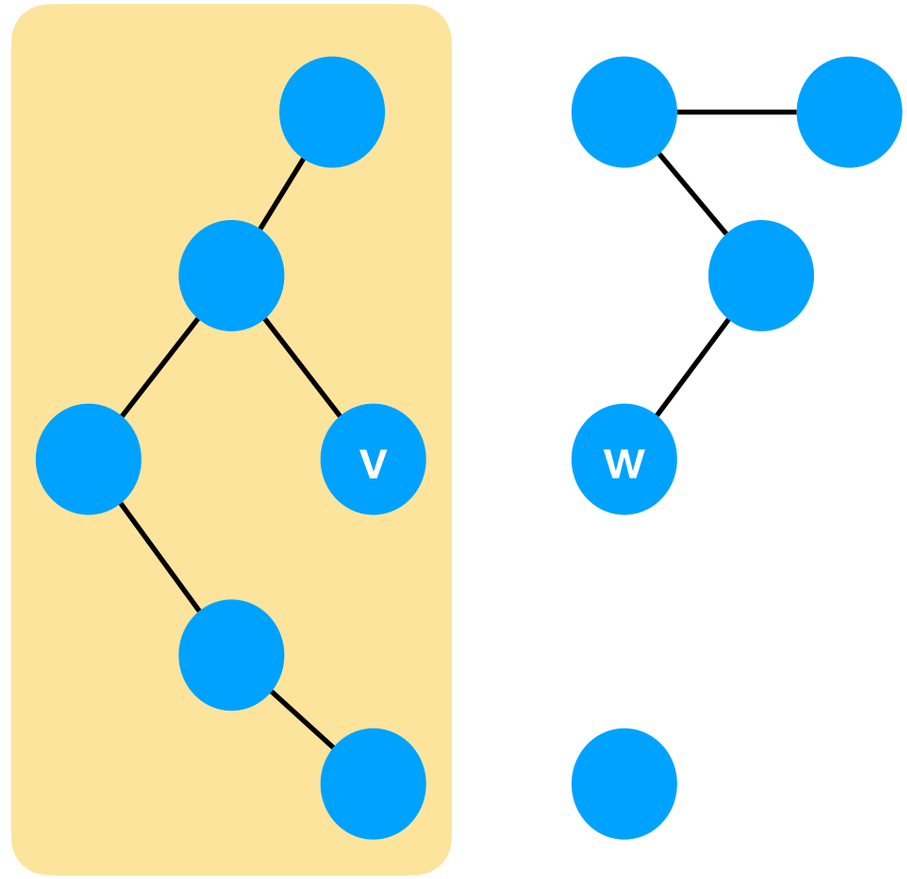
i.e., does it always produce a spanning tree?



# Is it feasible?

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

The algorithm explicitly avoids cycles.

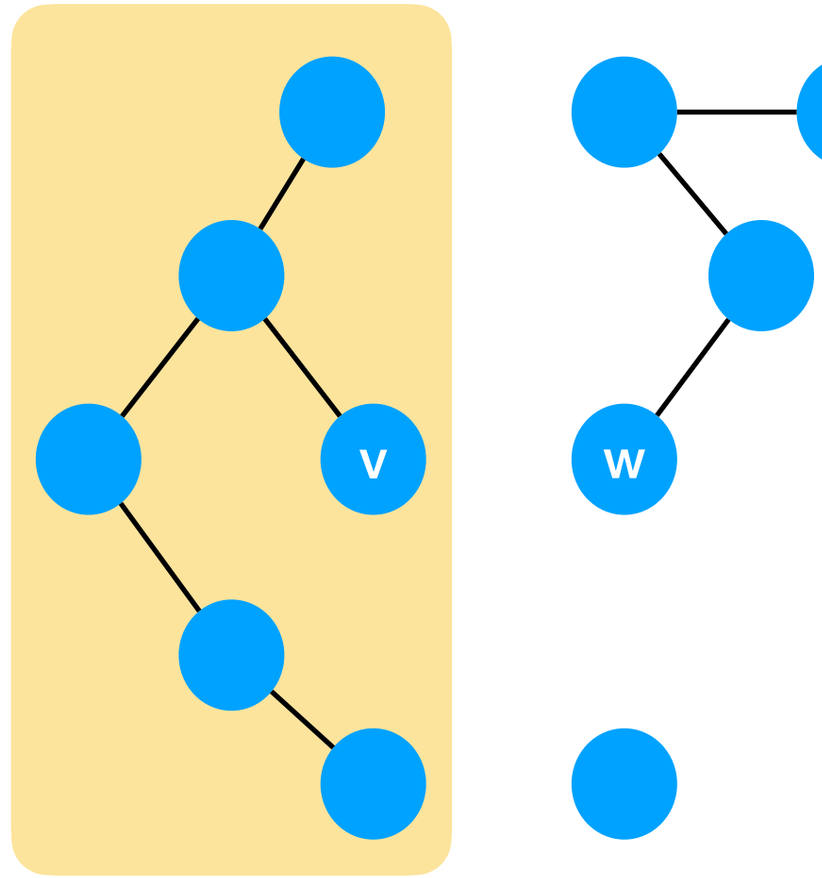


# Is it feasible?

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

The algorithm explicitly avoids cycles.

Output $G_T$ is a forest.



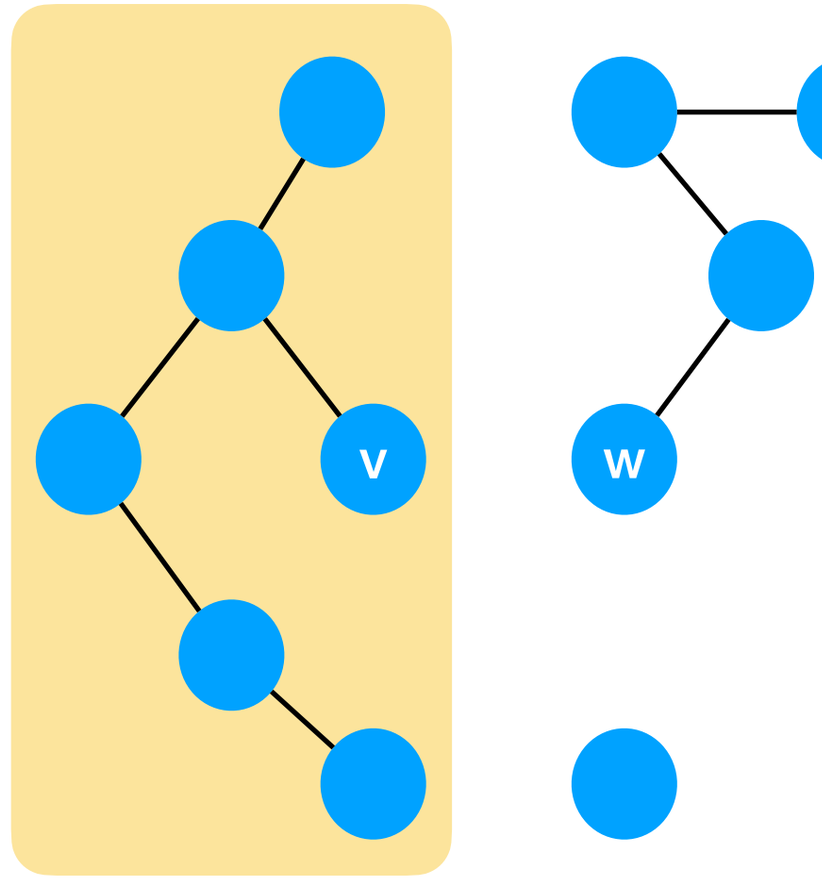
# Is it feasible?

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

The algorithm explicitly avoids cycles.

Output $G_T$ is a forest.

Is it a tree?



# Is it feasible?

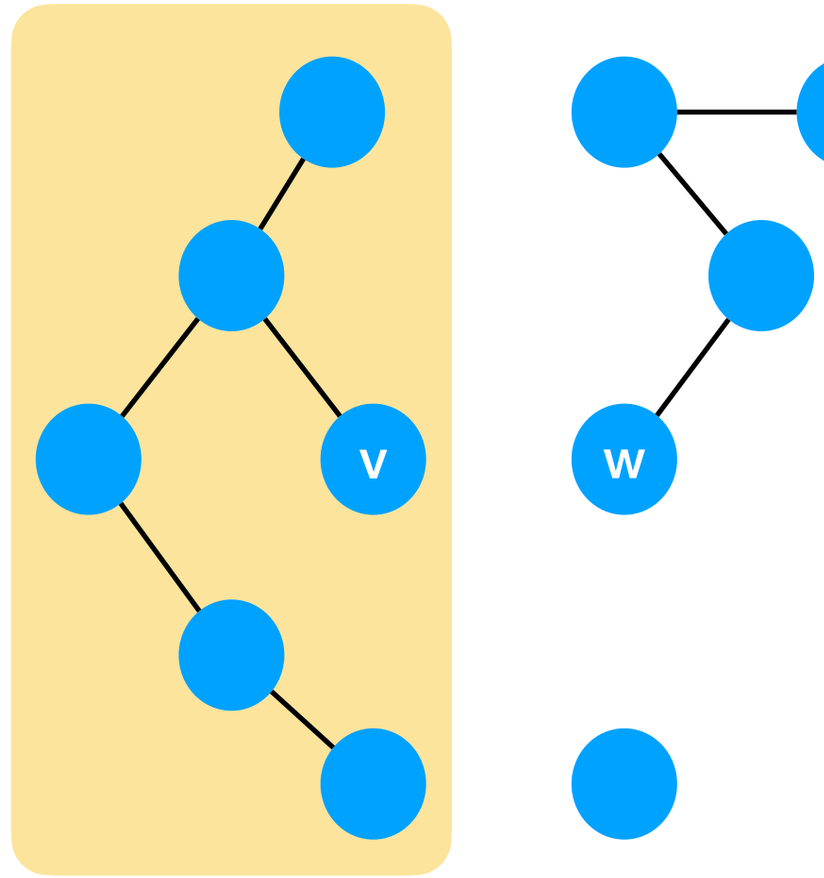
i.e., does it always produce a spanning tree?

The algorithm explicitly avoids cycles.

Output $G_T$ is a forest.

Is it a tree?

Is it connected?



# Is it feasible?

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

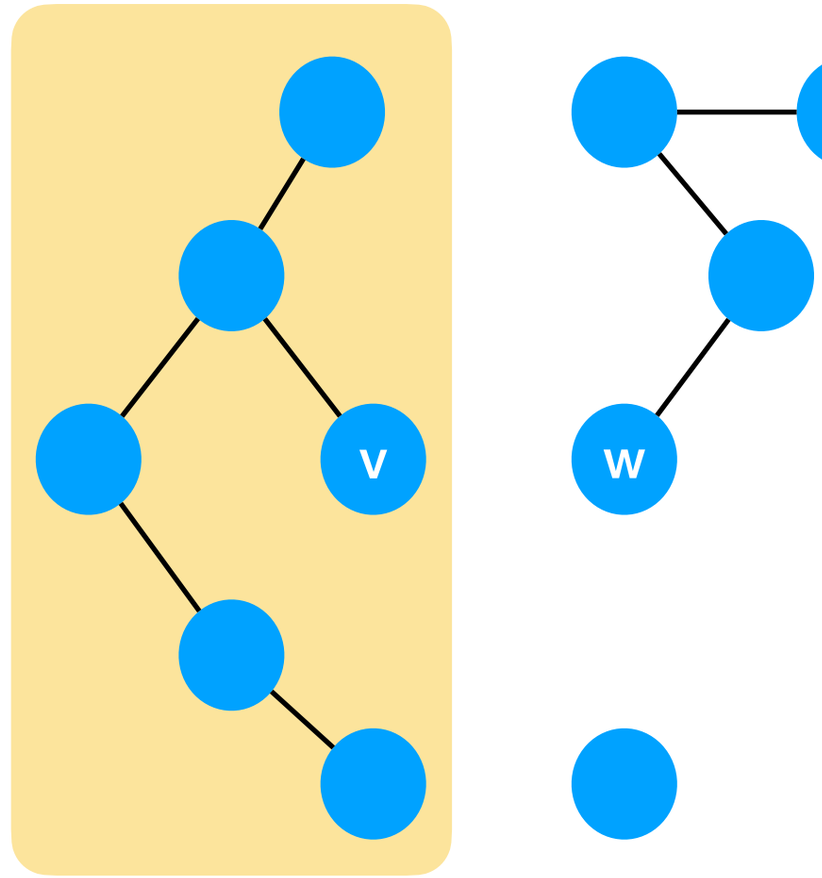
The algorithm explicitly avoids cycles.

Output $G_T$ 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.

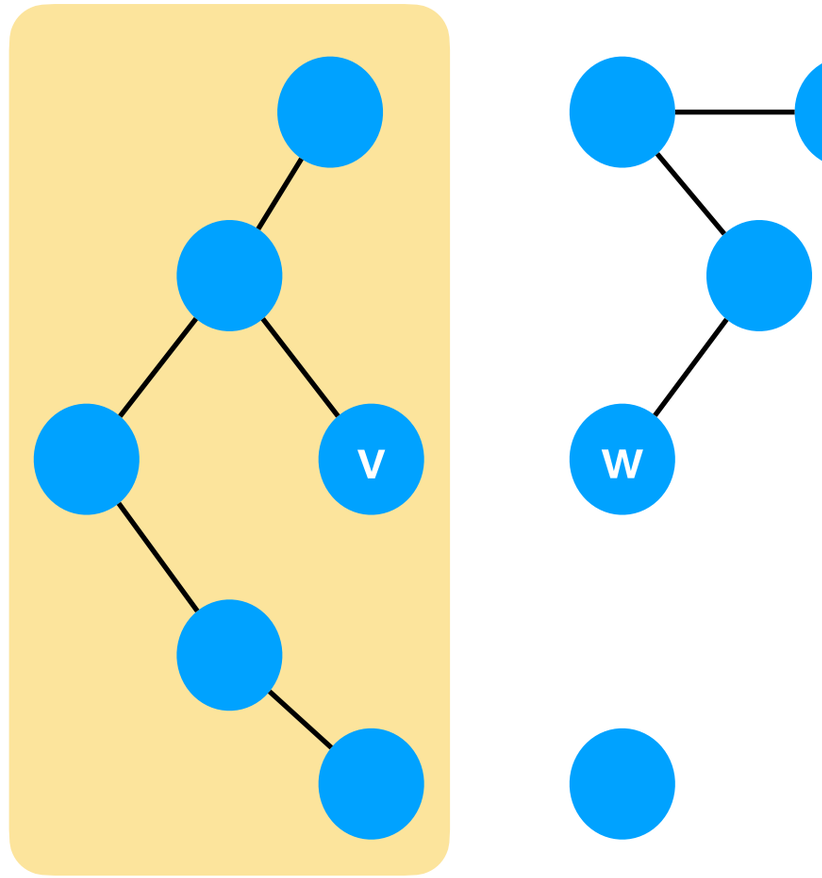
Output $G_T$ is a forest.

Is it a tree?

Is it connected?

$G$ is connected.

Suppose by contradiction that $G_T$ was not connected.



# Is it feasible?

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

The algorithm explicitly avoids cycles.

Output $G_T$ is a forest.

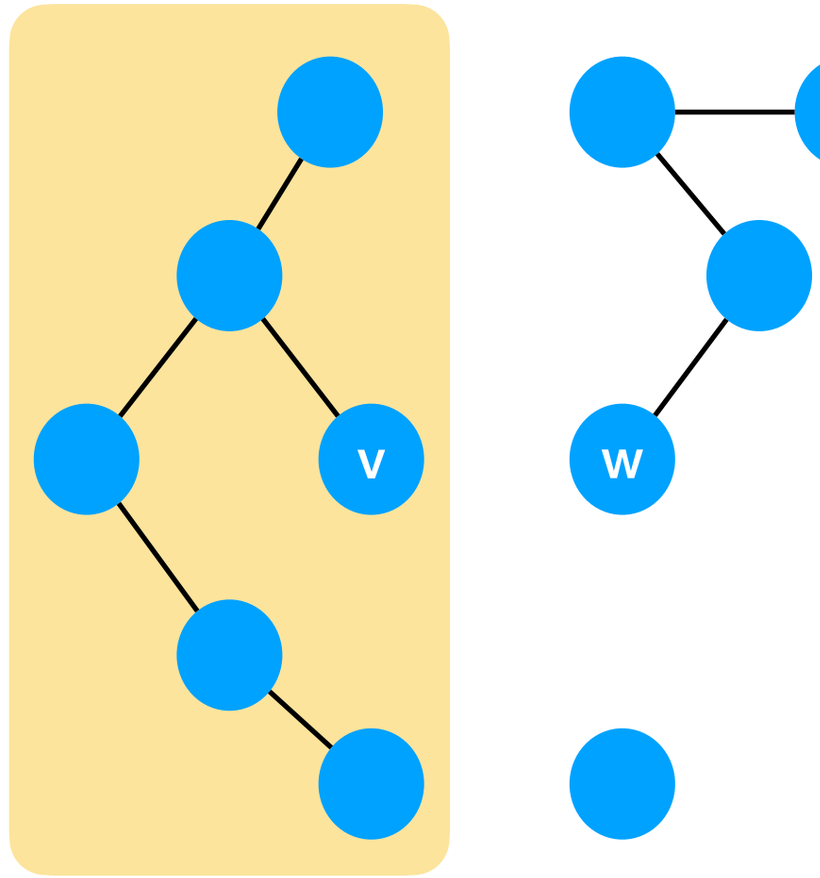
Is it a tree?

Is it connected?

$G$ is connected.

Suppose by contradiction that $G_T$ 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 $G_T$ is a forest.

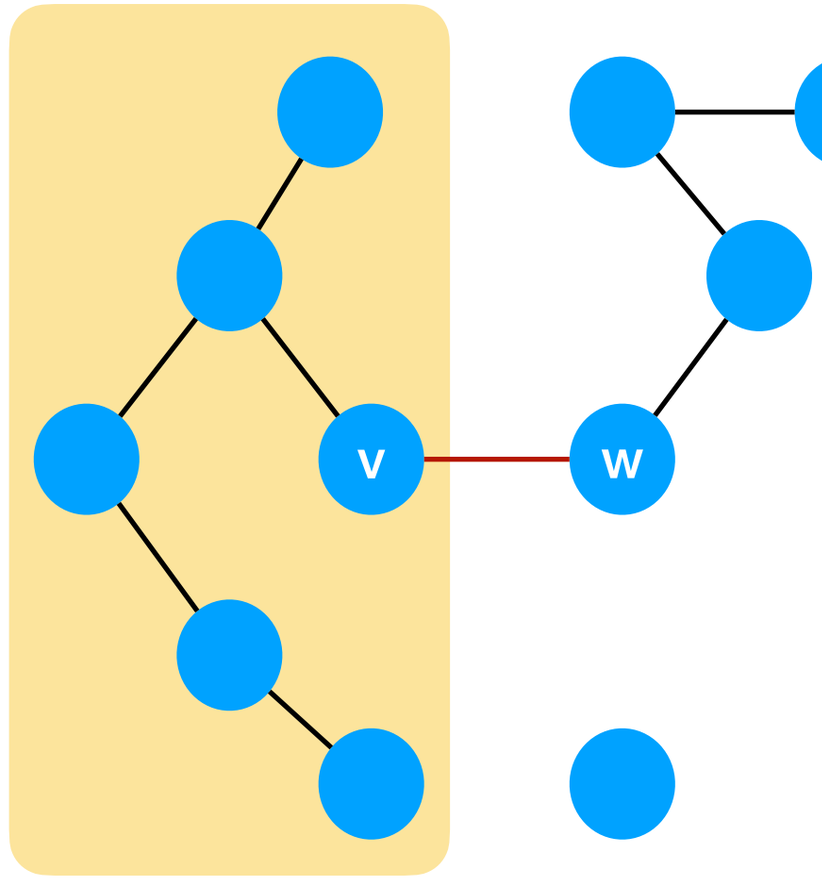
Is it a tree?

Is it connected?

$G$ is connected.

Suppose by contradiction that $G_T$ 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, by the cut property.

# Greedy Approach #3

# Greedy Approach #3

Start with the full graph $G=(V, E)$.

# Greedy Approach #3

Start with the full graph $G$=($V$, $E$ ).

Delete an edge from $G$.

# Greedy Approach #3

Start with the full graph $G$=($V$, $E$ ).

Delete an edge from $G$.

Which one?

# Greedy Approach #3

Start with the full graph $G$=($V$, $E$ ).

Delete an edge from $G$.

Which one?

The one with the maximum cost $c_e$.

# Greedy Approach #3

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.

# Greedy Approach #3

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 $G$?

# Greedy Approach #3

Start with the full graph $G$=($V$, $E$ ).

Delete an edge from $G$.

Which one?

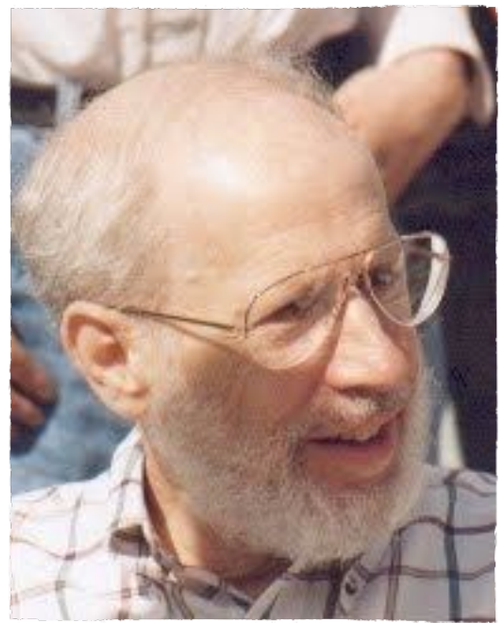
The one with the maximum cost $c_e$.

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?

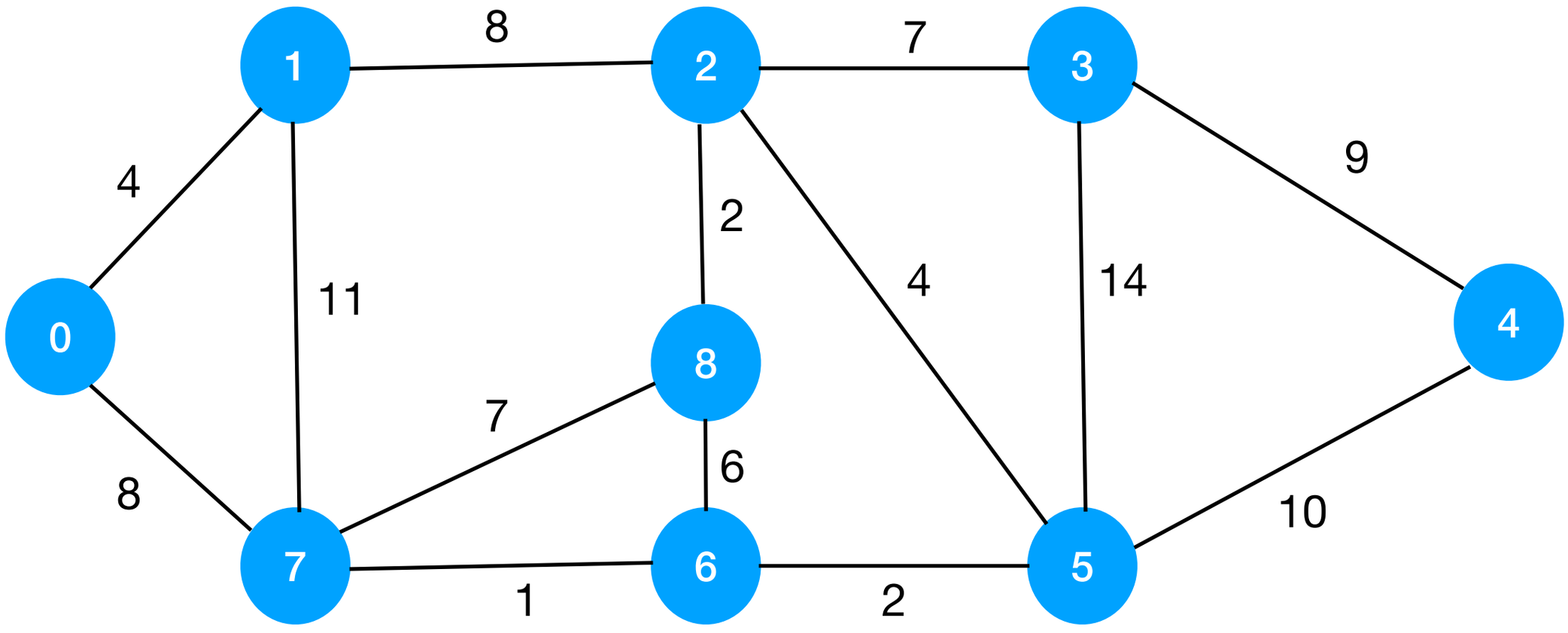
The one with the maximum cost $c_e$.

We continue like this.

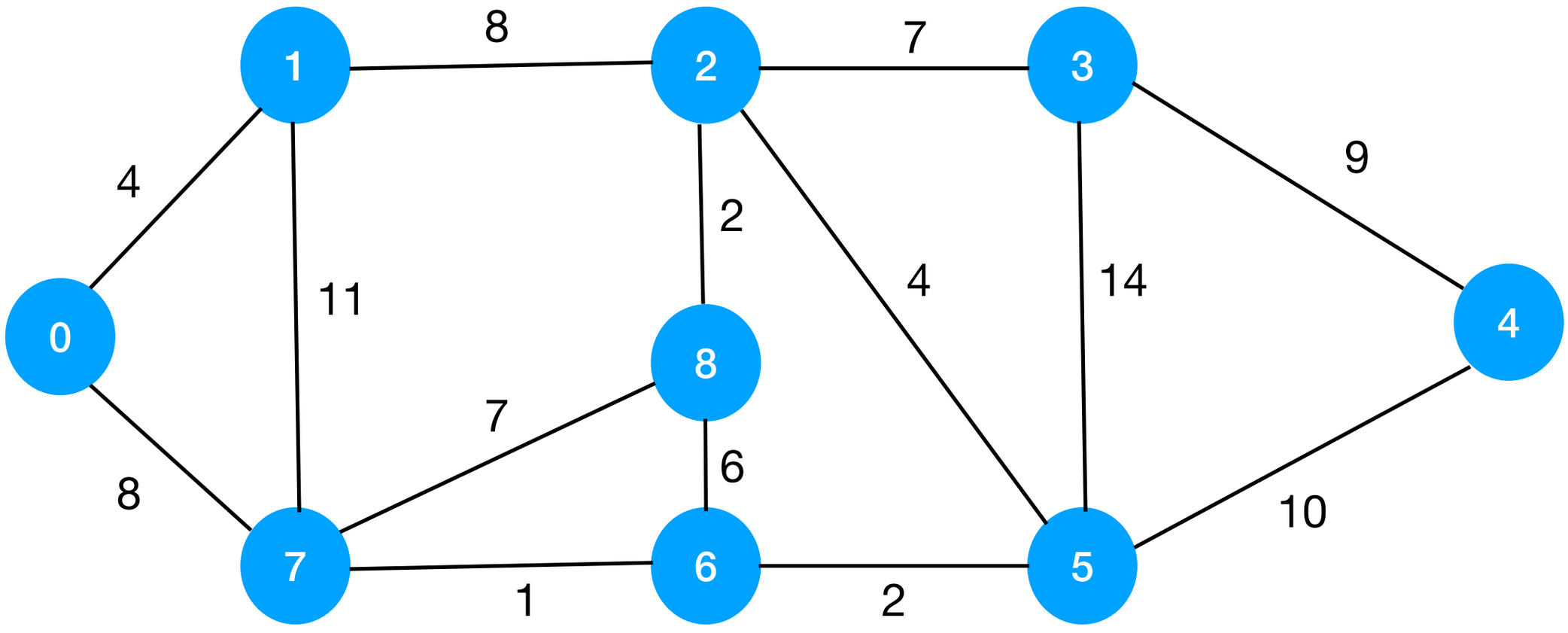
Do we always remove the considered edge $e$ from $G$?

As long as we don’t disconnect the graph.

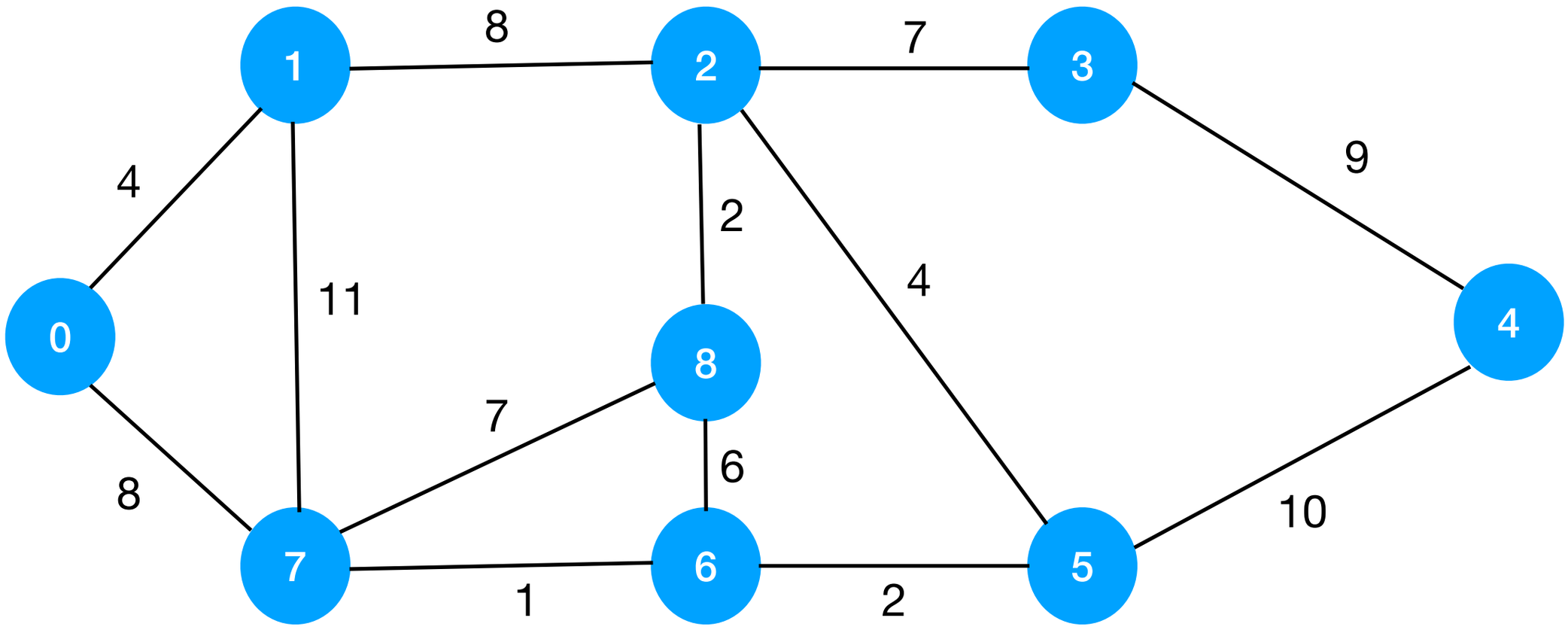
# Example



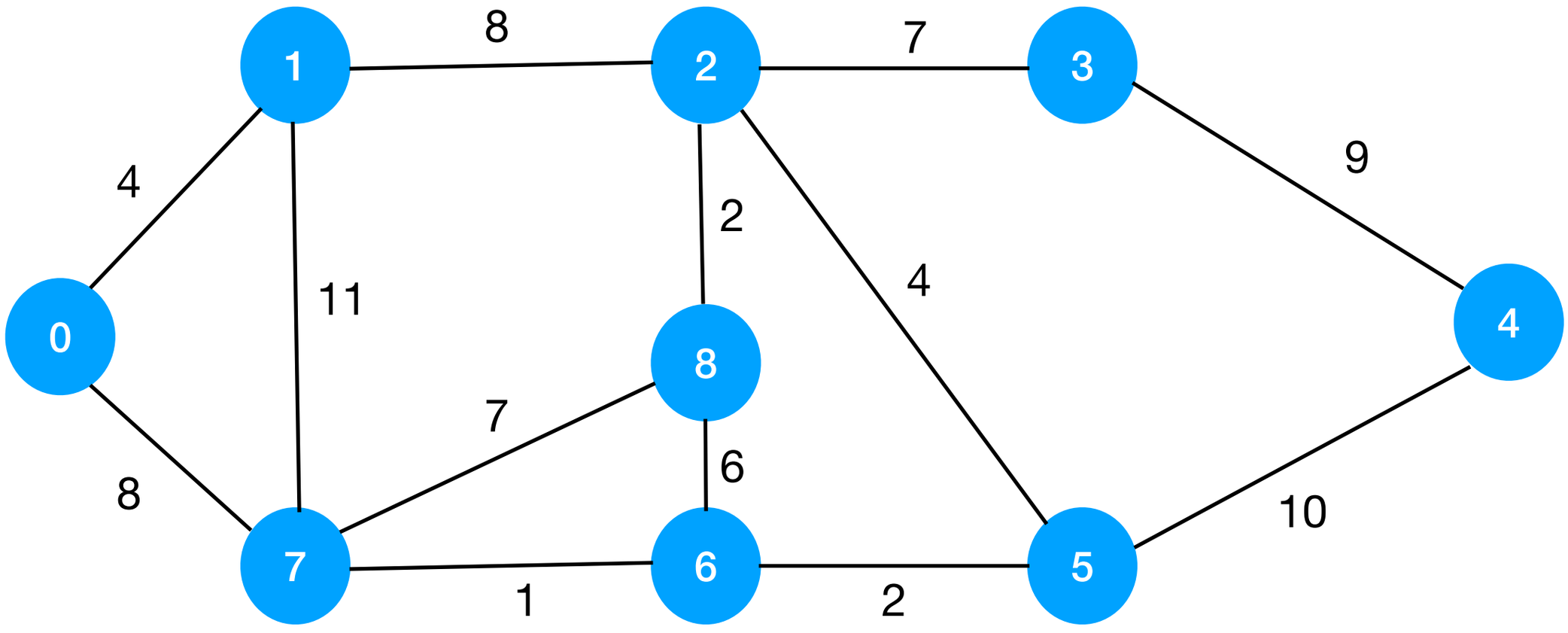
# Example



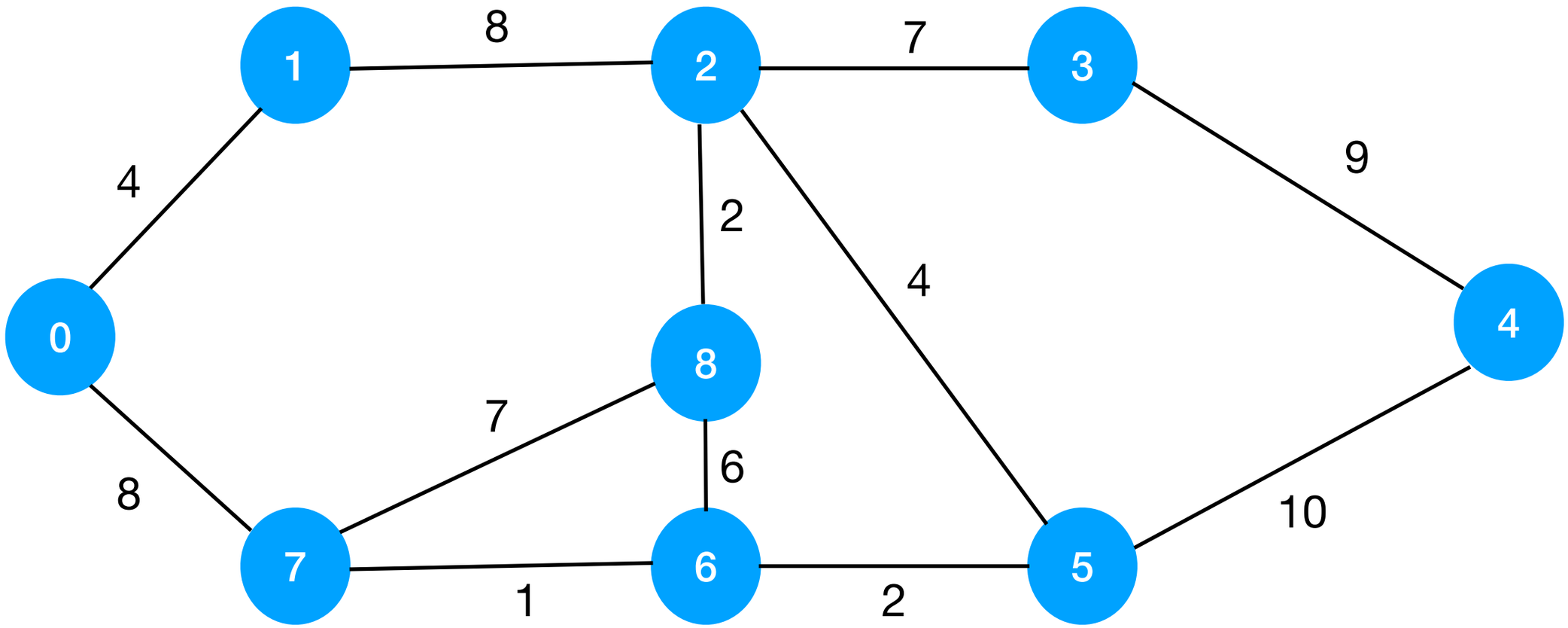
# Example



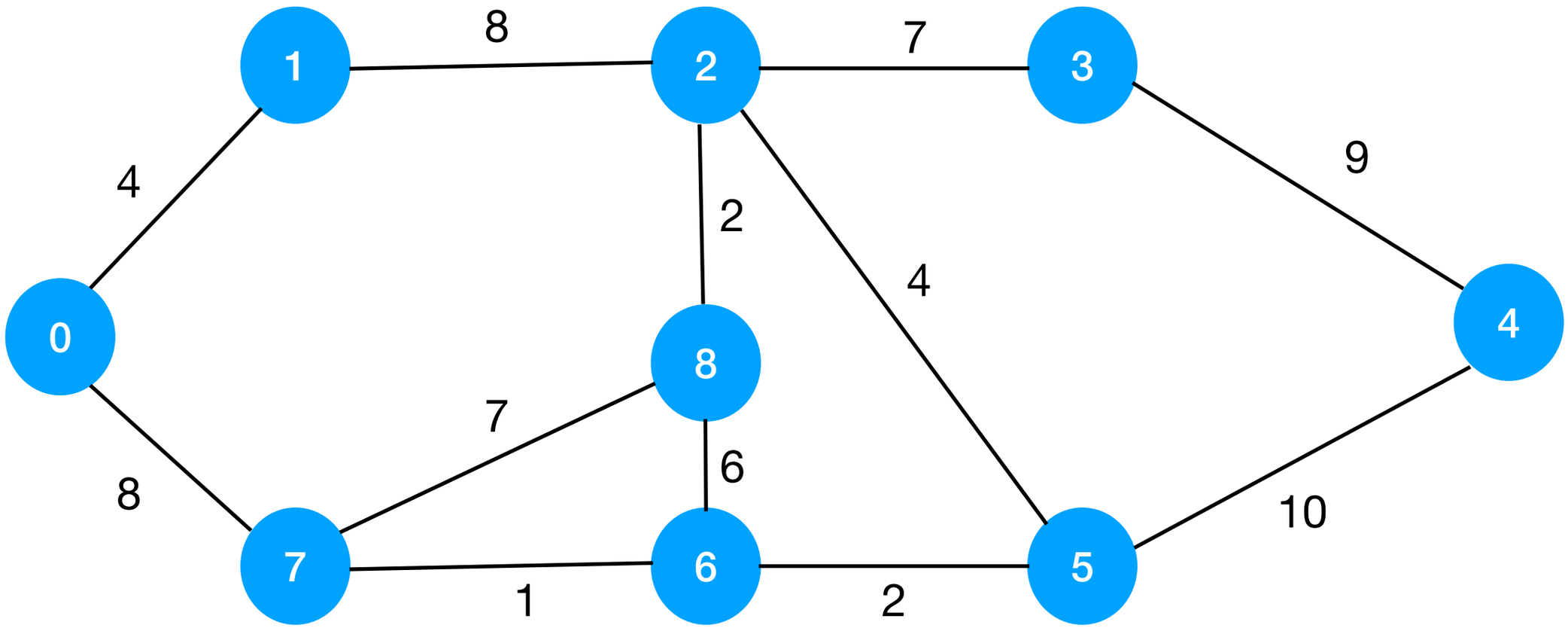
# Example



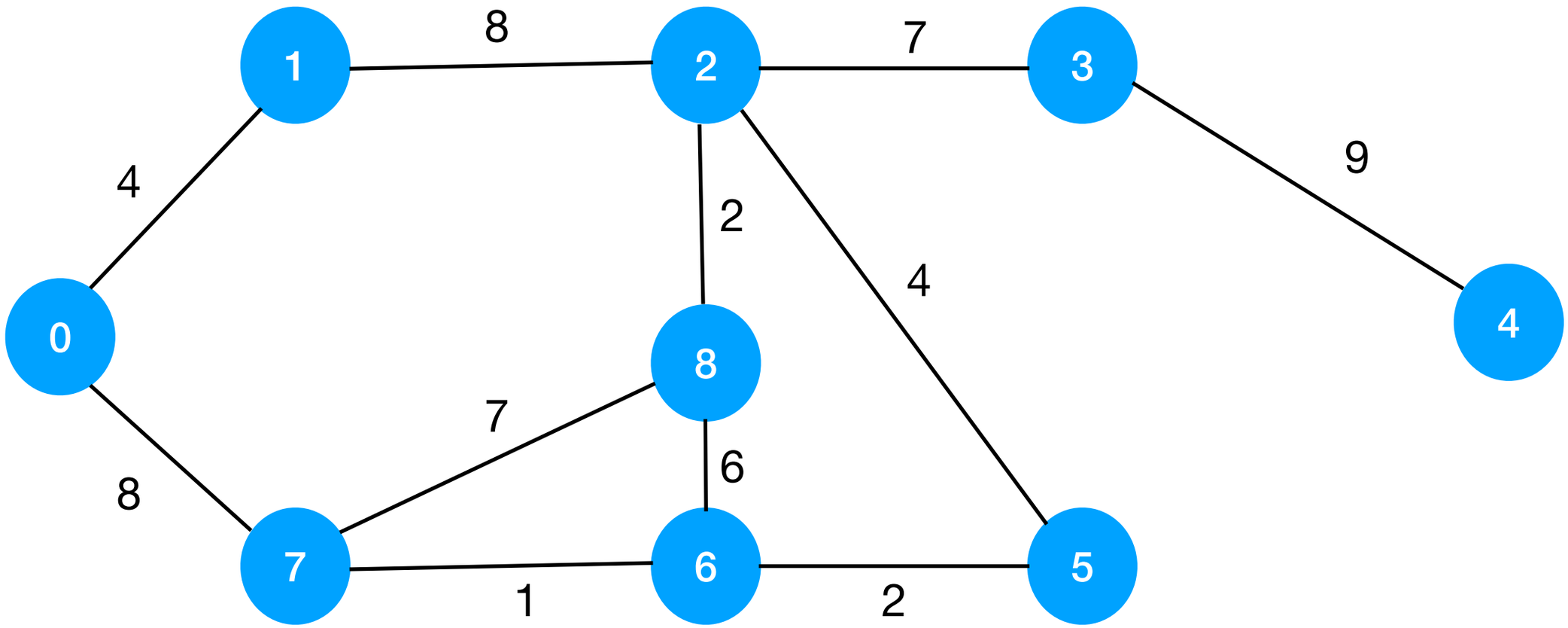
# Example



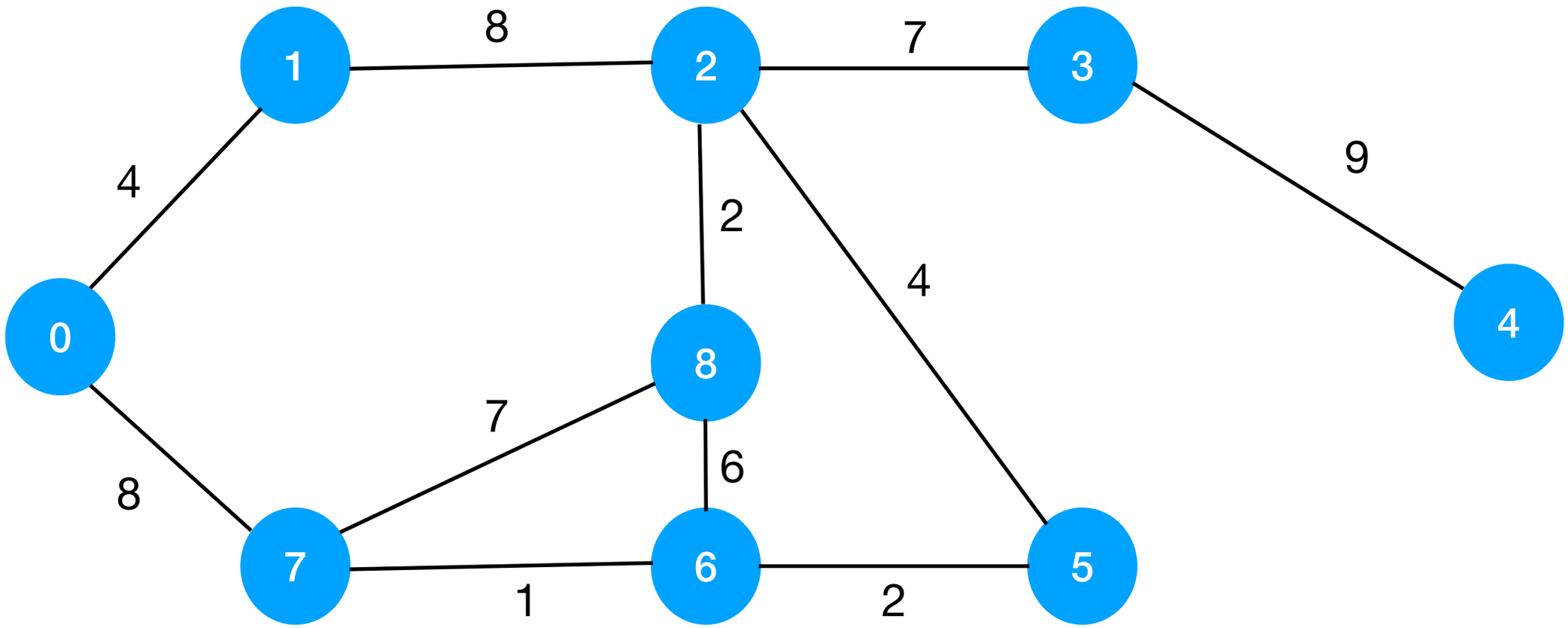
# Example



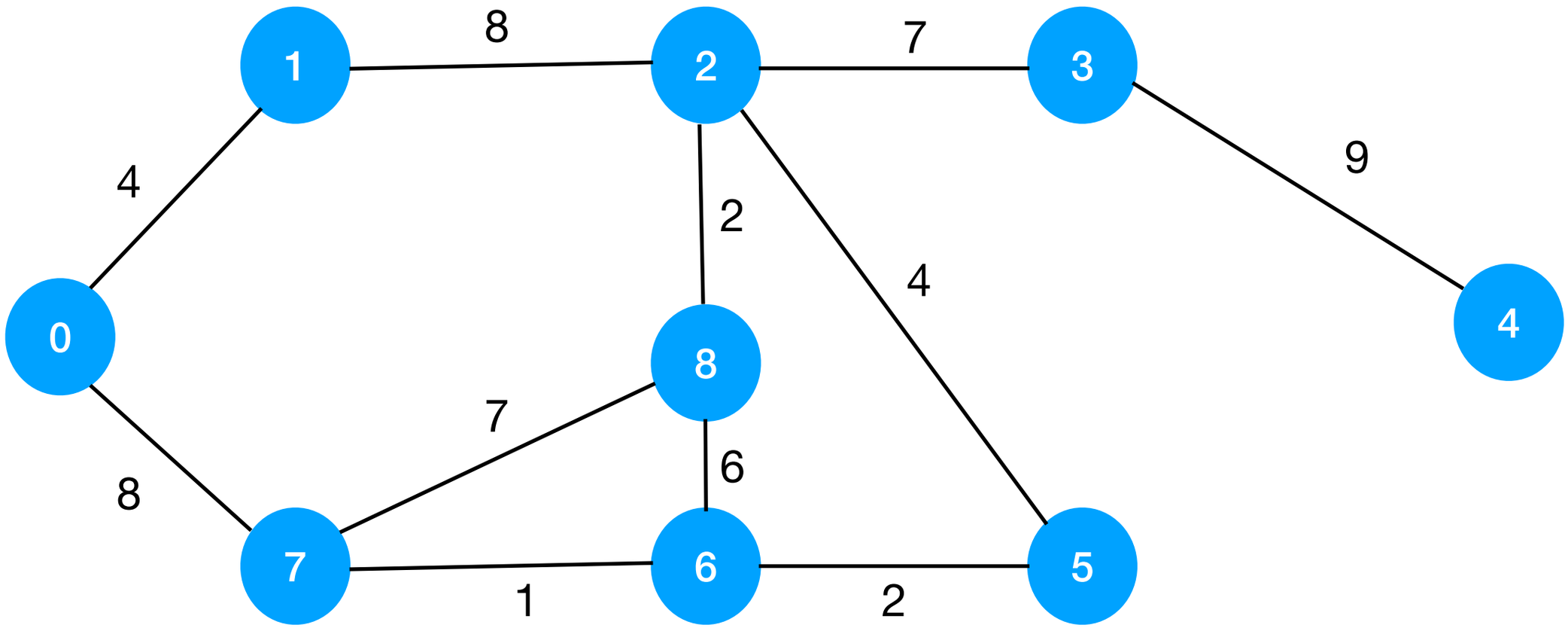
# Example



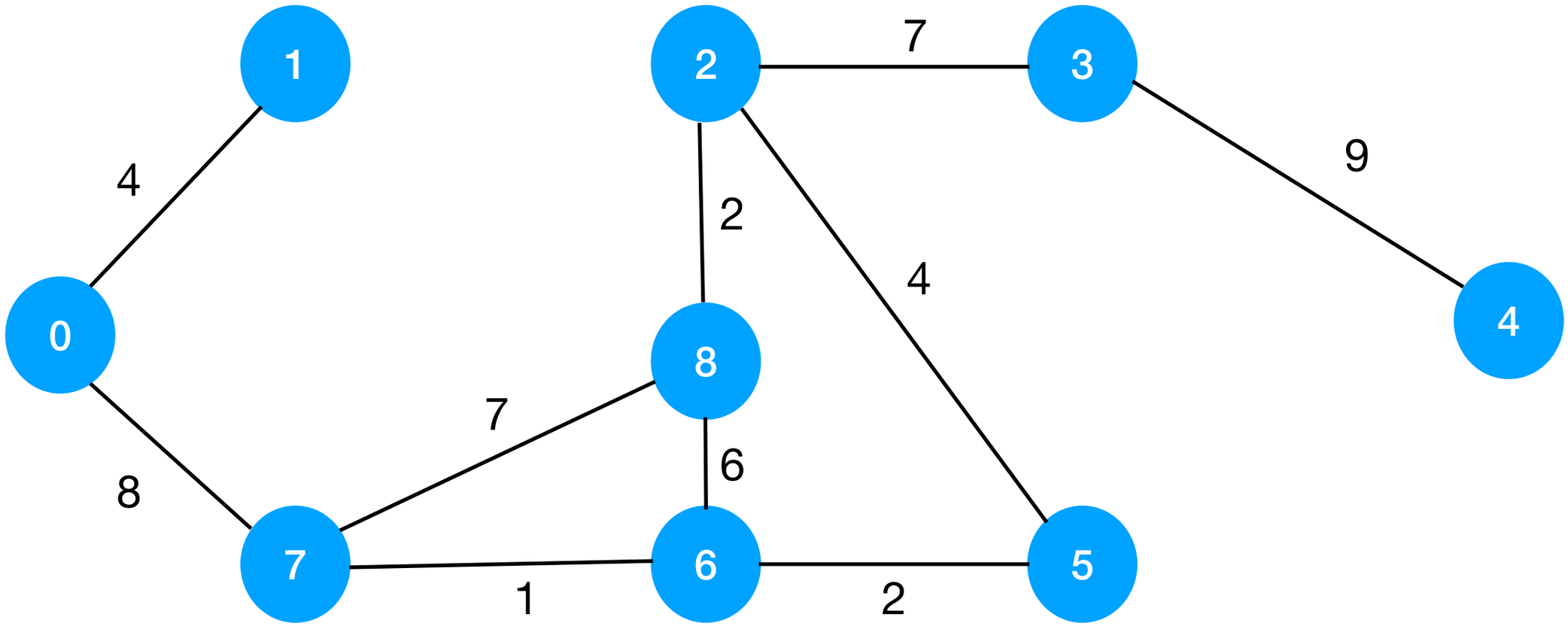
# Example



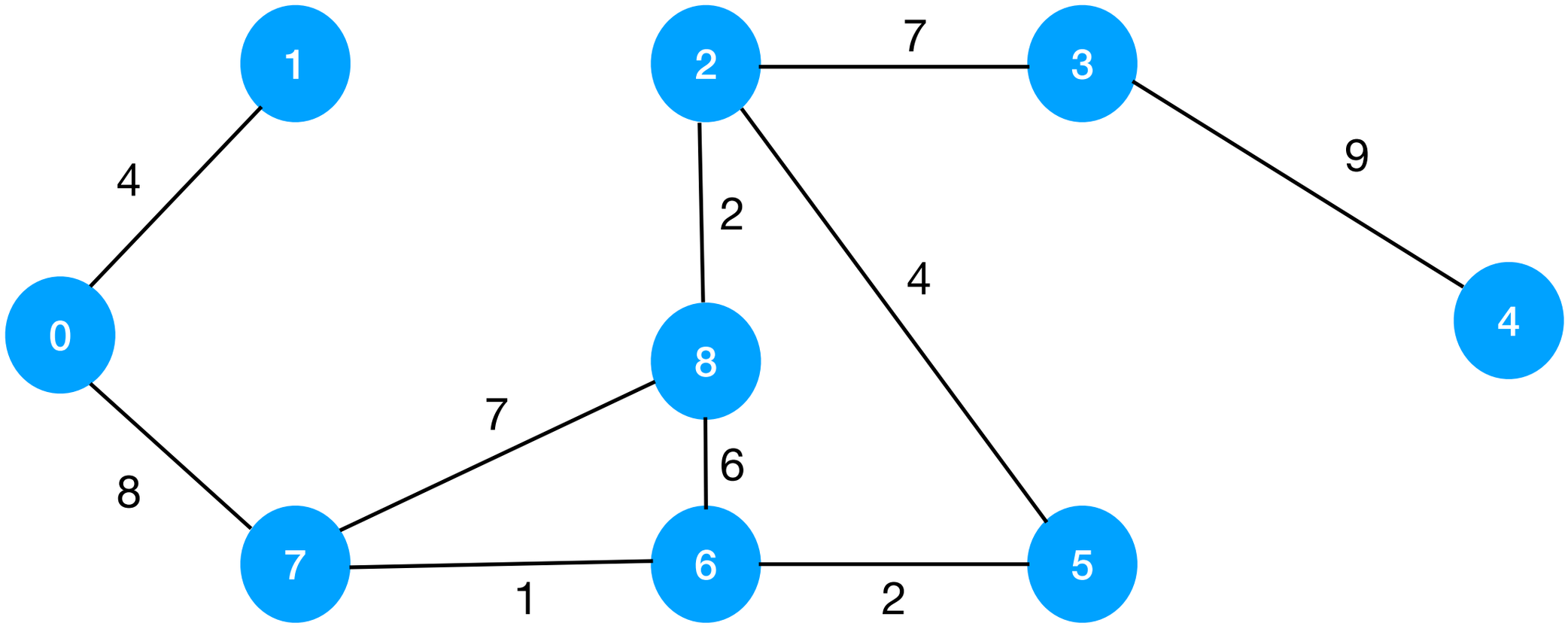
# Example



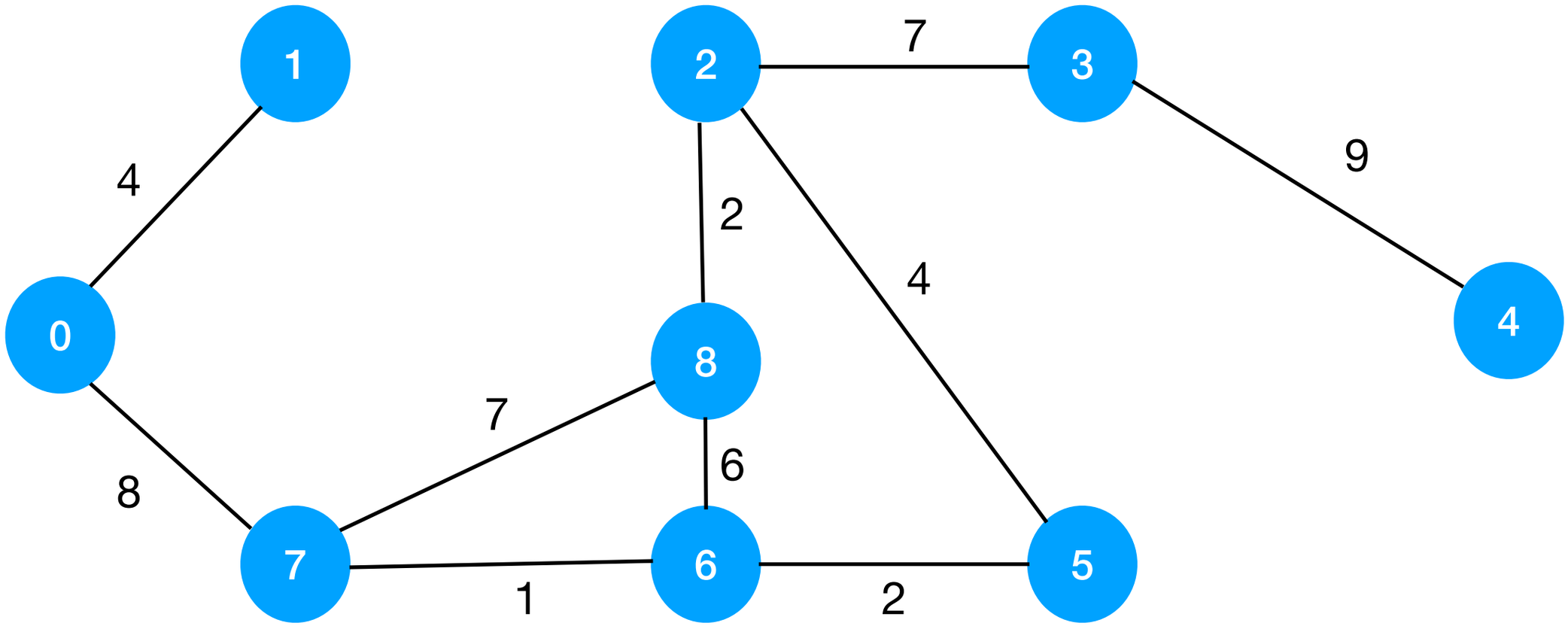
# Example



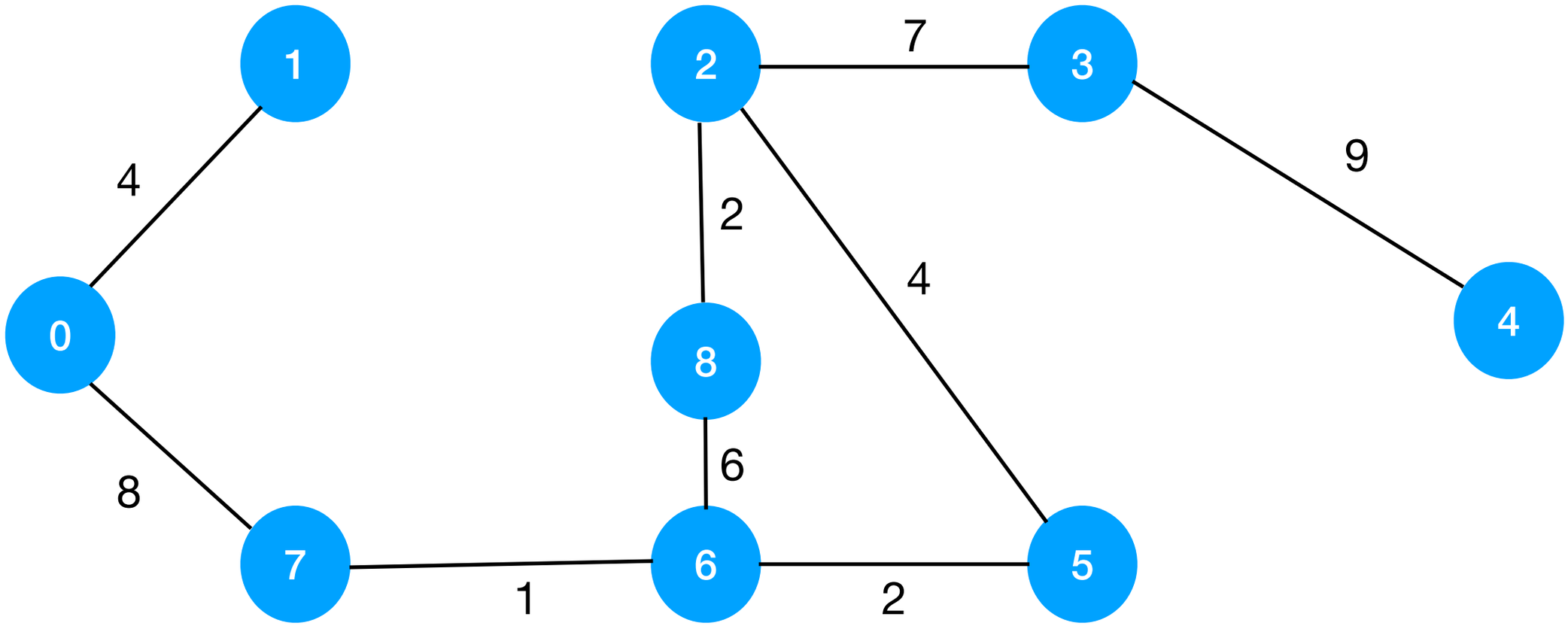
# Example



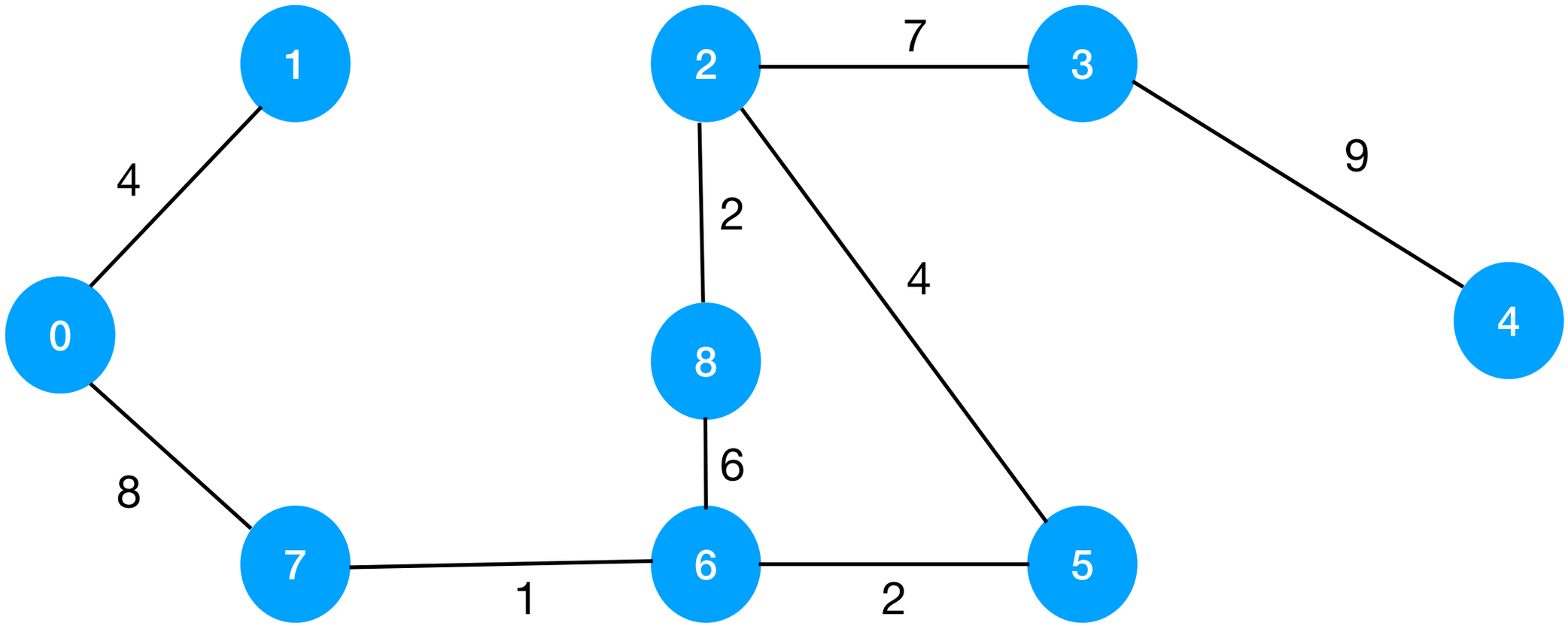
# Example



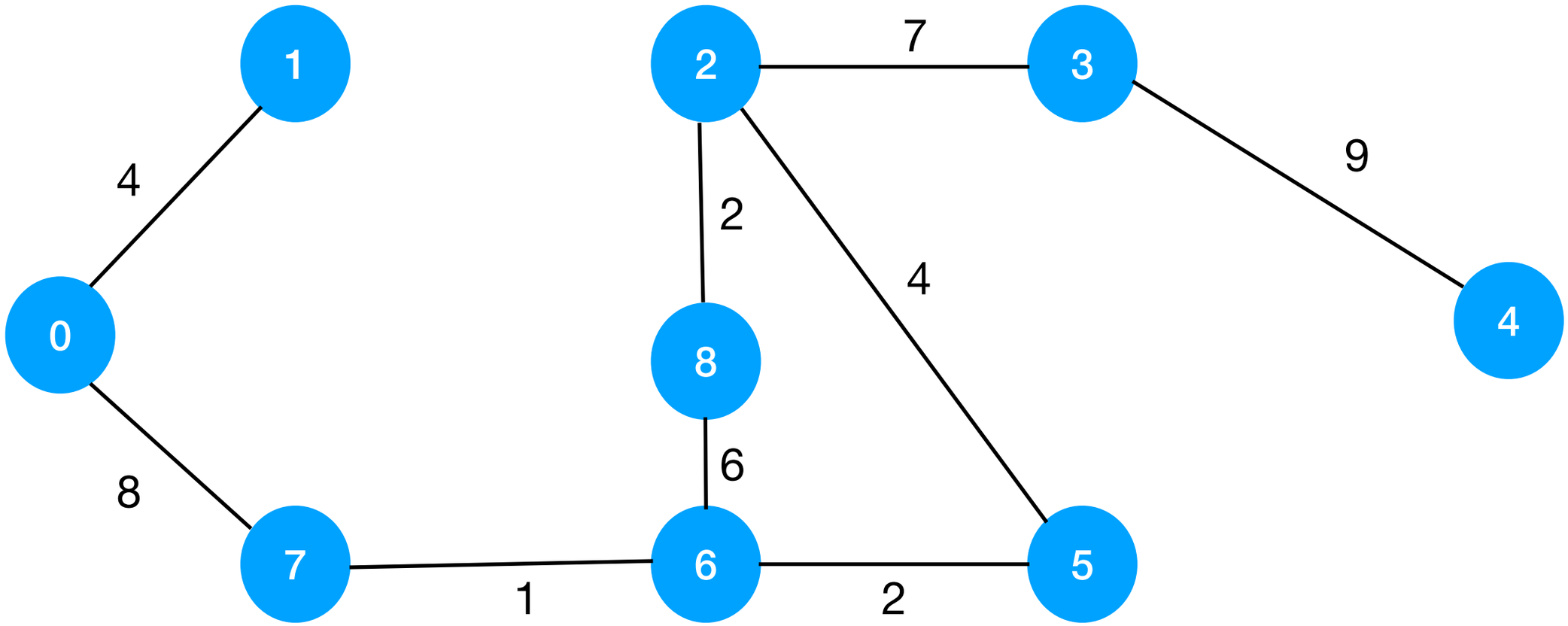
# Example



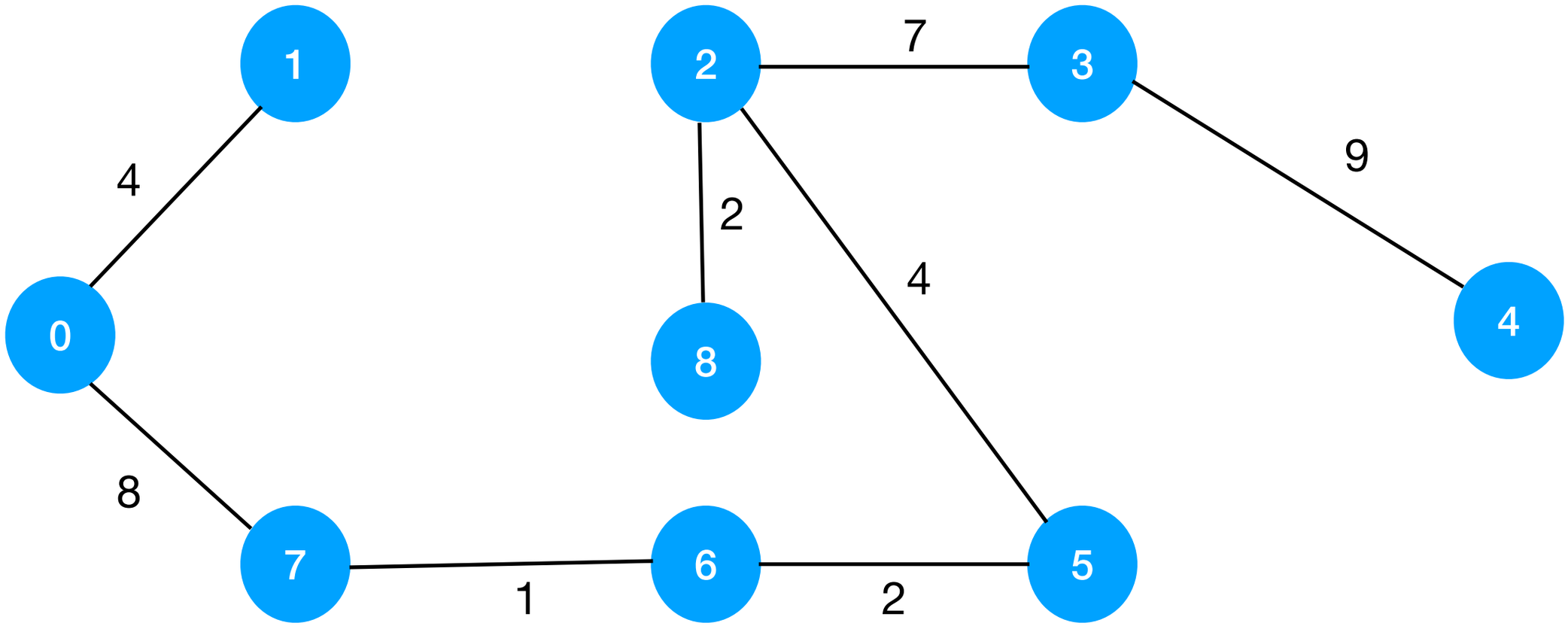
# Example



# 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 $G$.

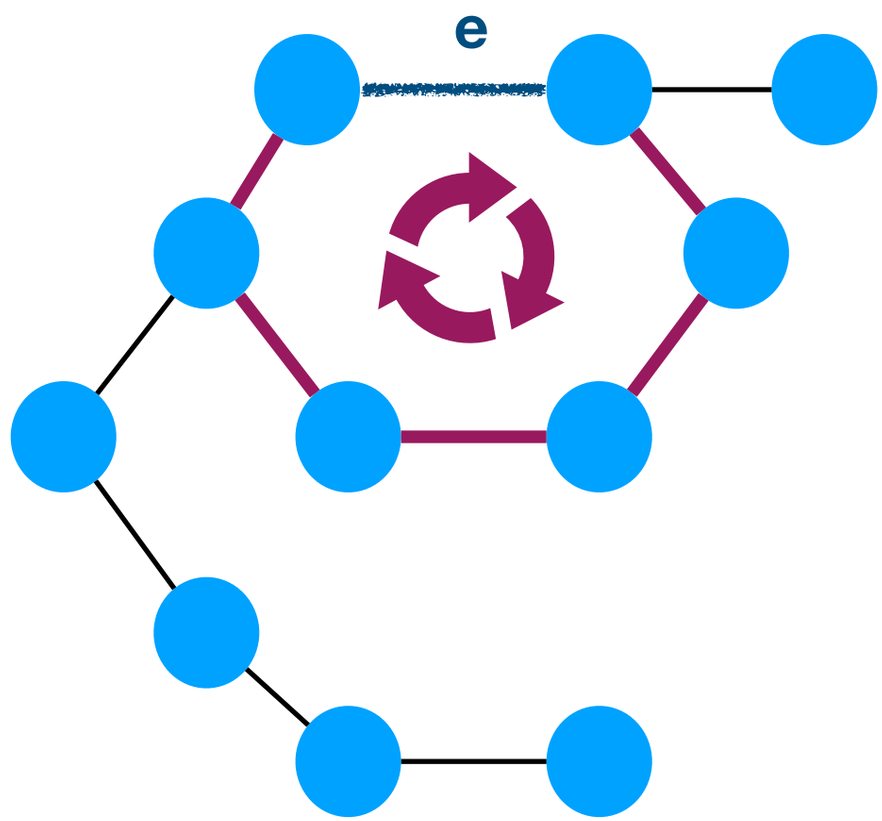
# The cycle property

Assume that all edge costs are distinct.

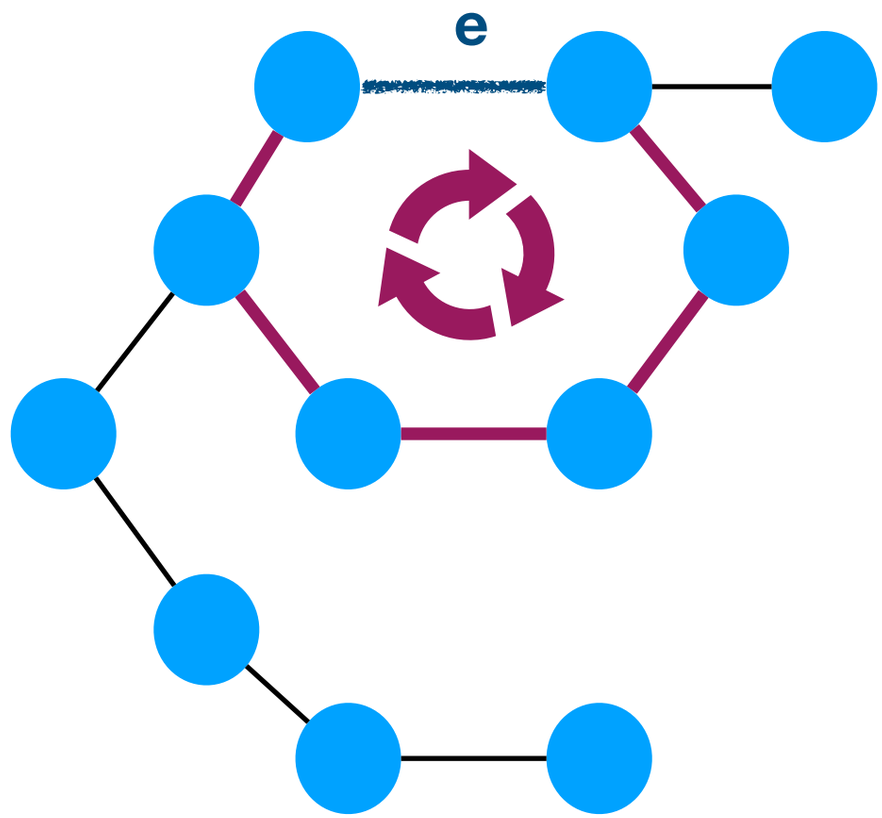
Let $C$ be any cycle of $G$.

Let $e$=($w$,$v$) be the maximum cost edge of $C$.

Then $e$ is not contained in any minimum spanning tree of $G$.

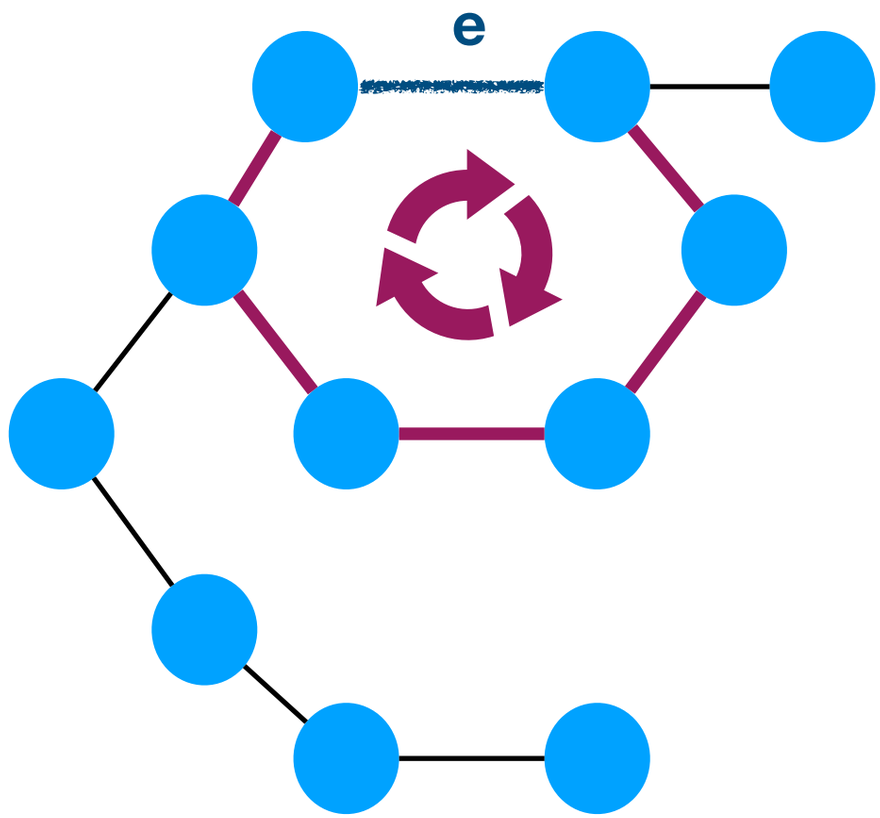


# The cycle property



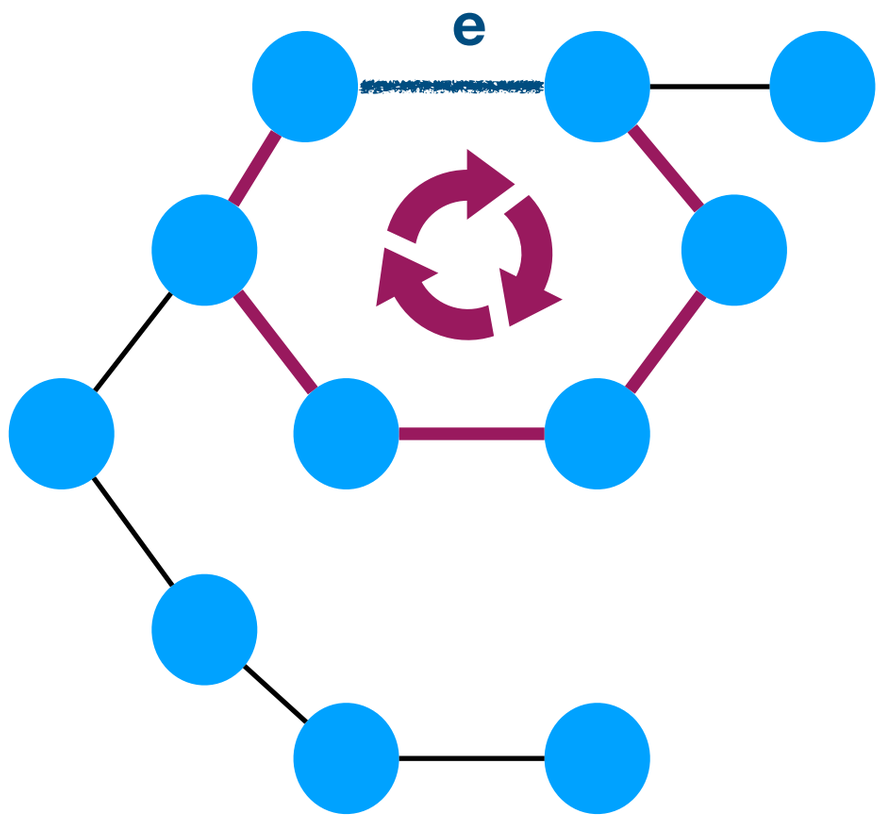
# The cycle property

- Let T be a spanning tree that contains e.



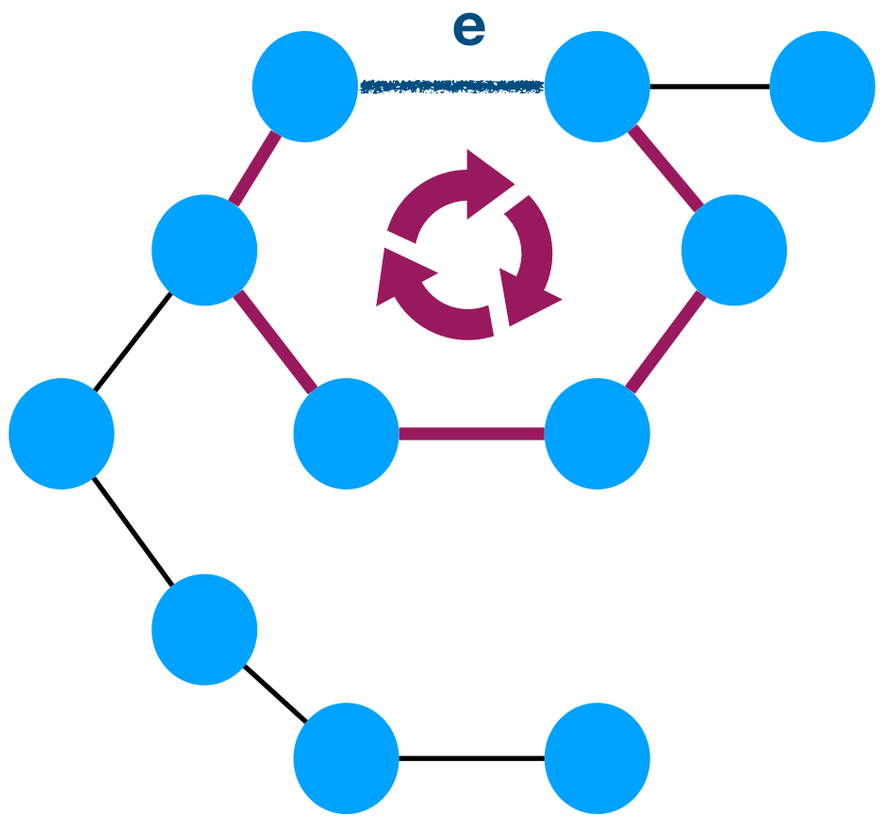
# The cycle property

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



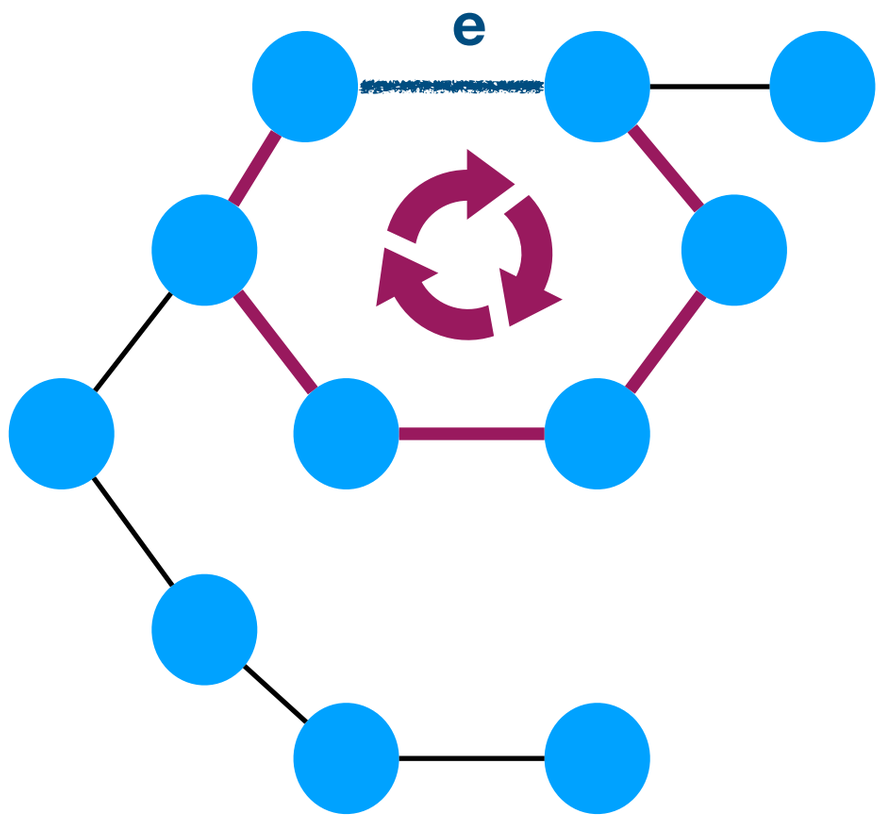
# The cycle property

- 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.

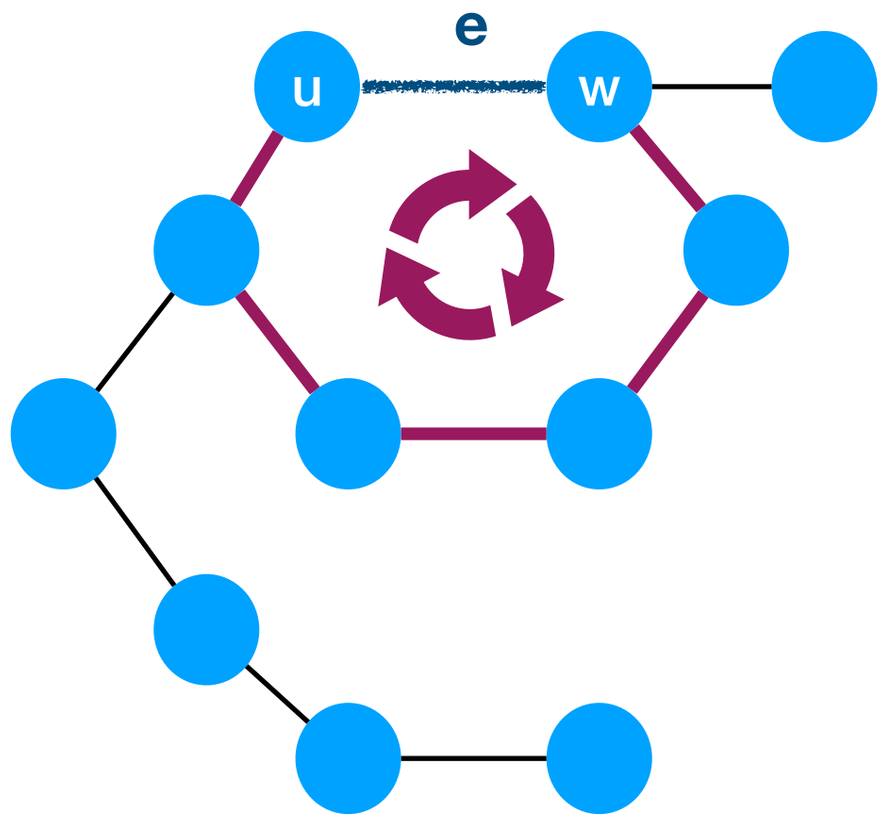


# The cycle property

- 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'?

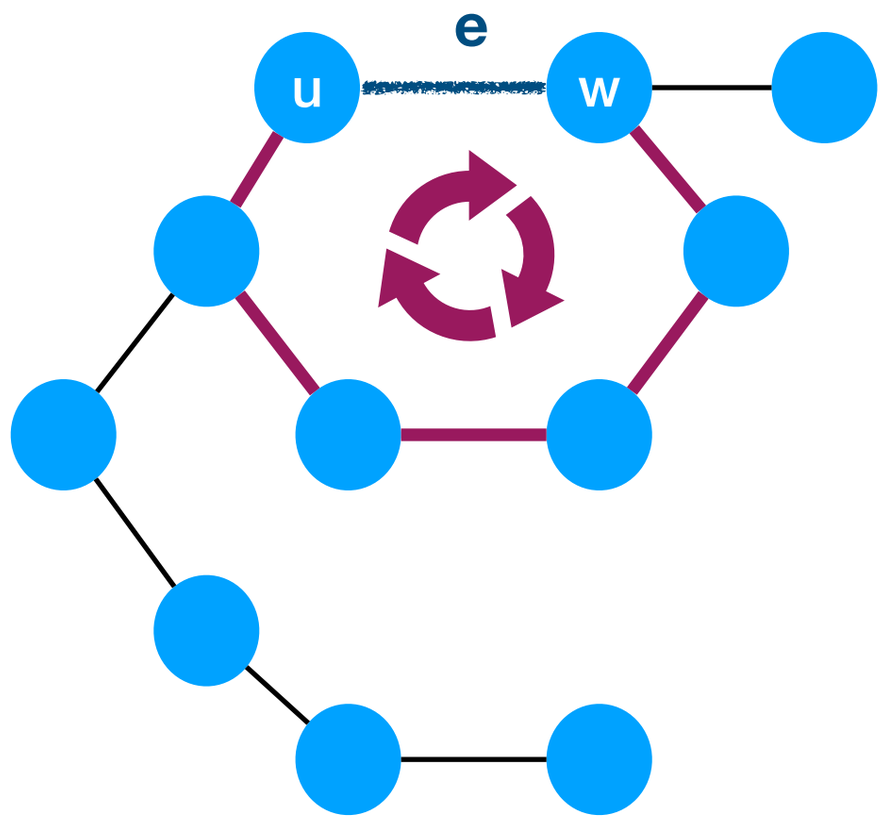


# The cycle property



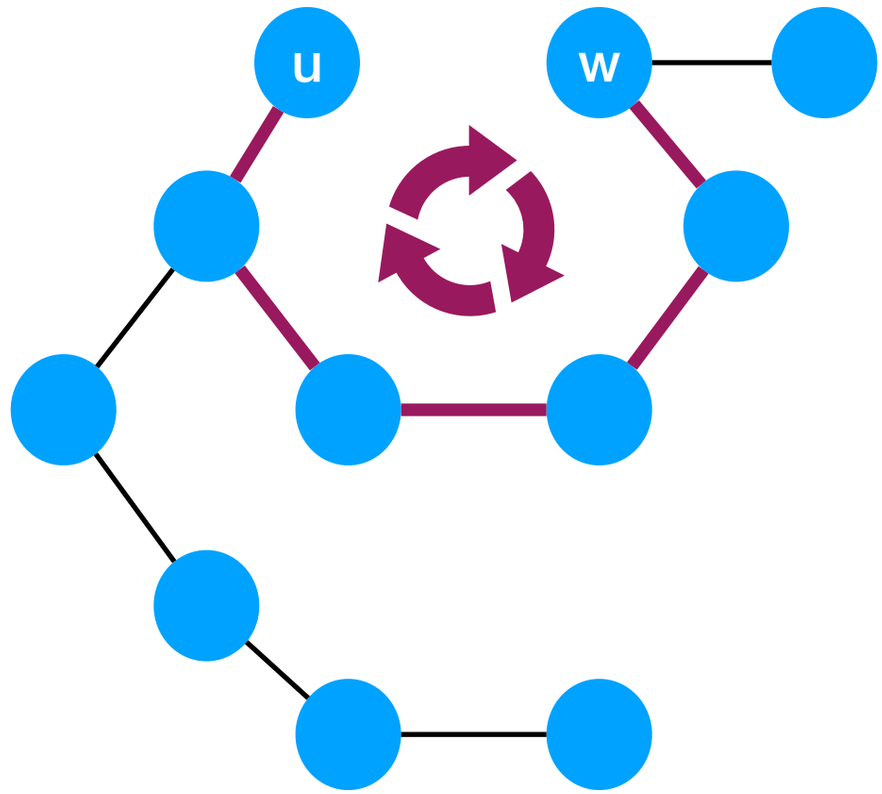
# The cycle property

We delete $e$ from T.



# The cycle property

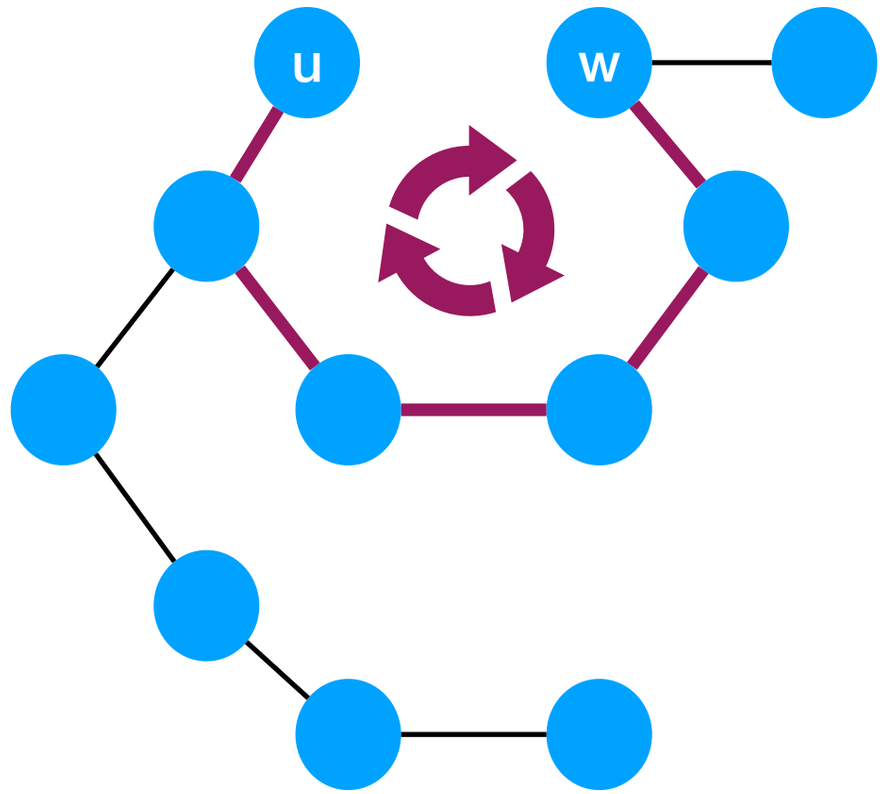
We delete $e$ from T.



# The cycle property

We delete $e$ from T.

This partitions the nodes into

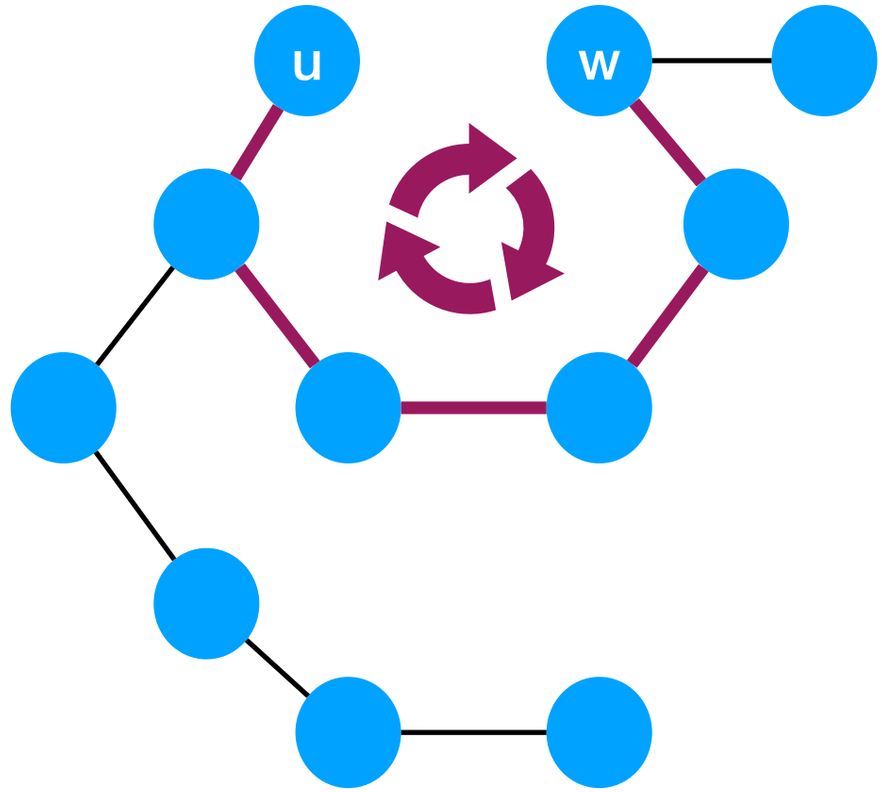


# The cycle property

We delete $e$ from T.

This partitions the nodes into

$S$ (containing $u$).



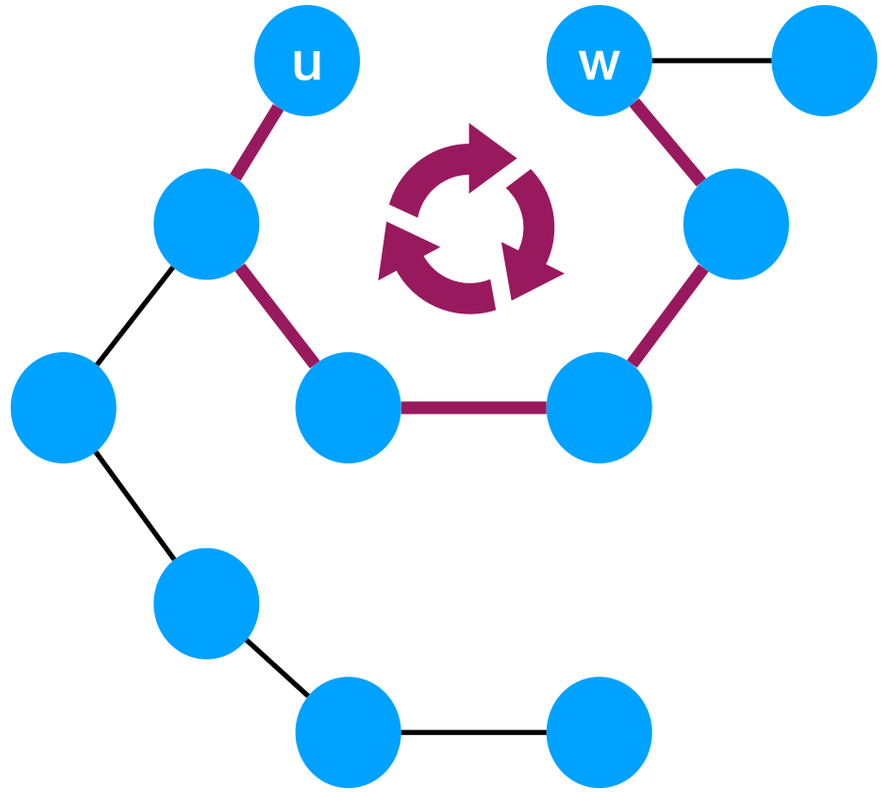
# The cycle property

We delete $e$ from T.

This partitions the nodes into

$S$ (containing $u$).

$V$ - $S$ (containing $w$).



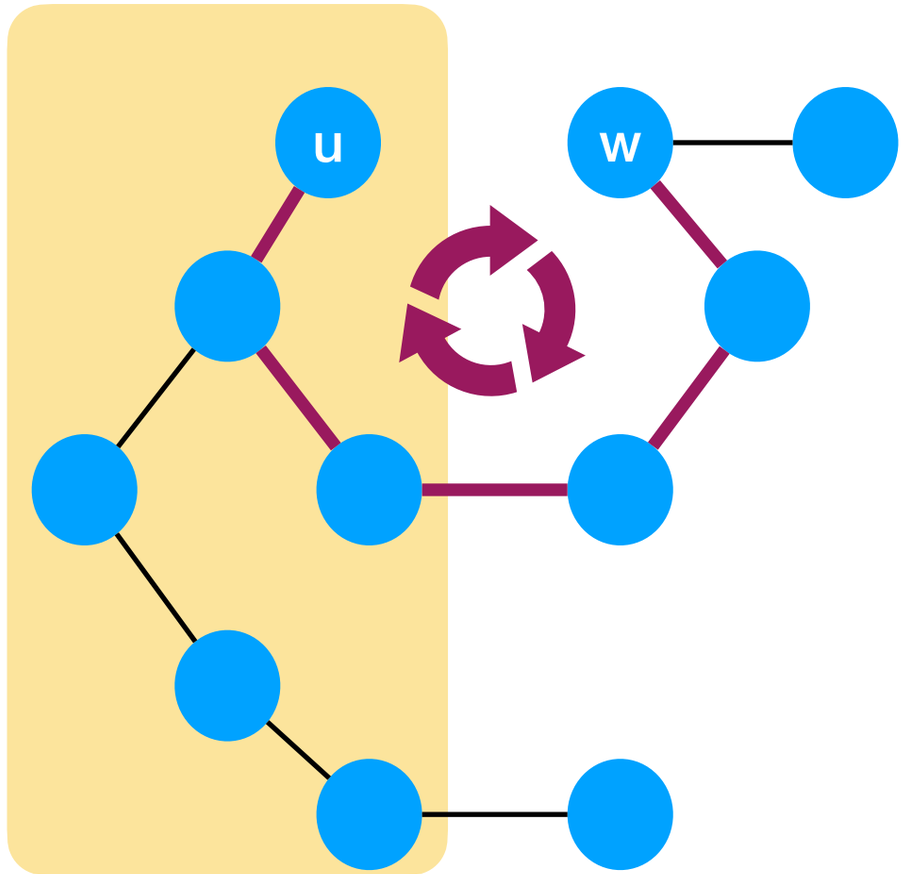
# The cycle property

We delete $e$ from T.

This partitions the nodes into

$S$ (containing $u$).

$V$ - $S$ (containing $w$).



# The cycle property

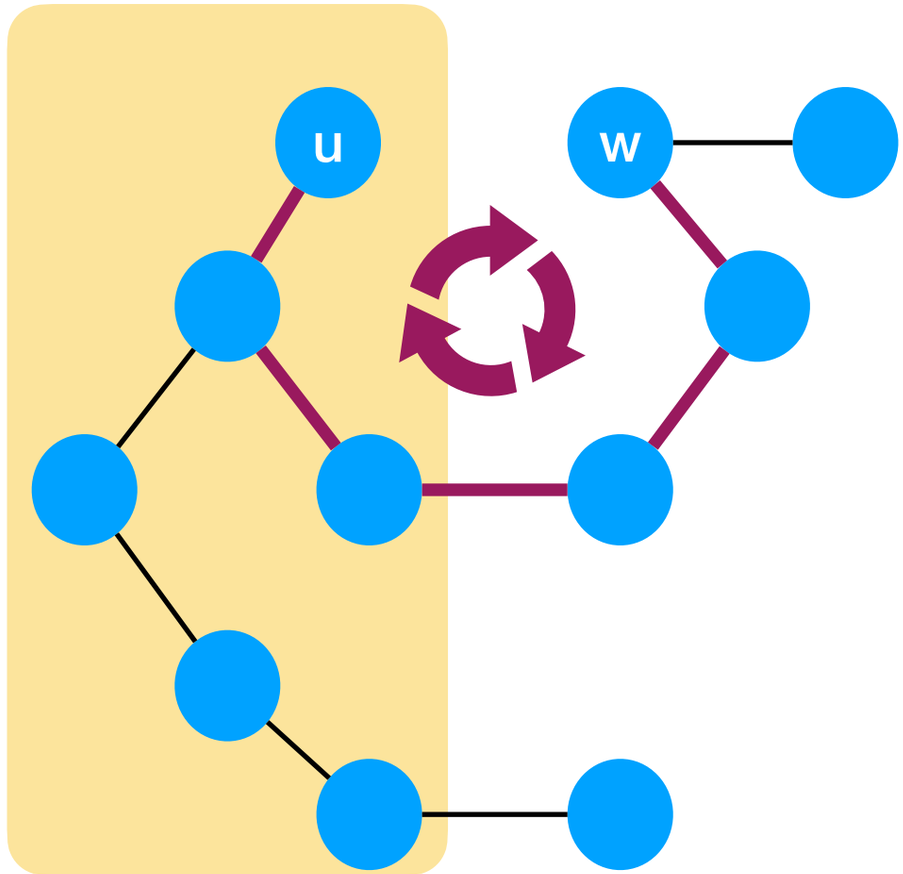
We delete $e$ from T.

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.

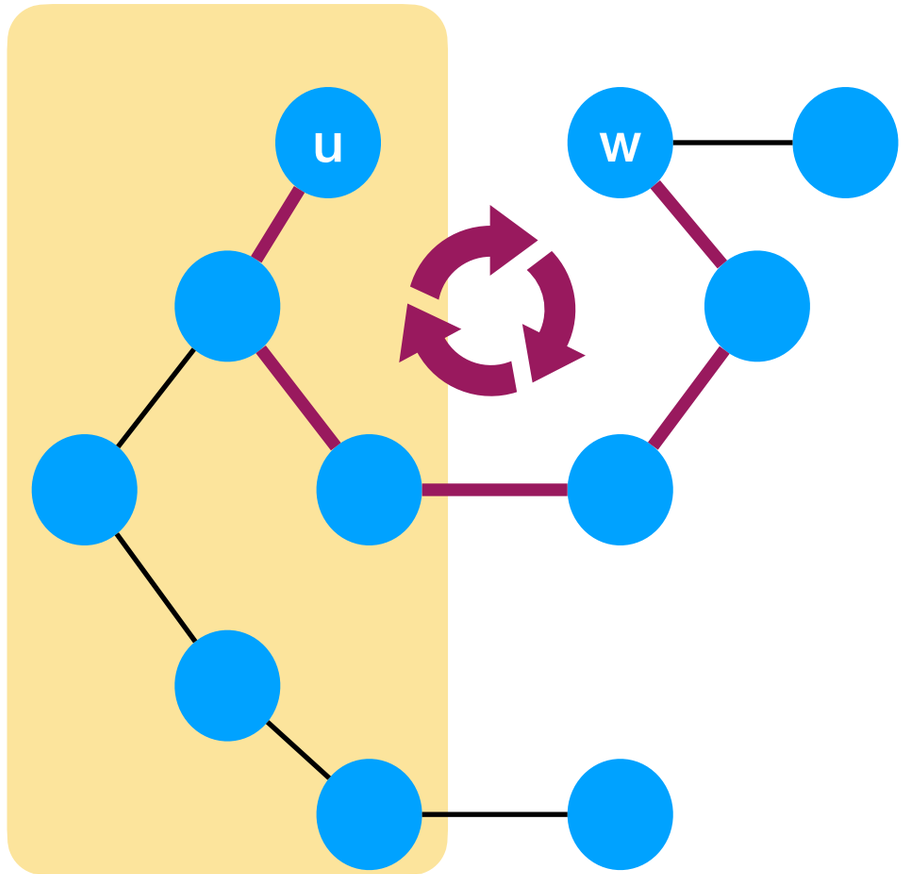
This partitions the nodes into

$S$ (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.

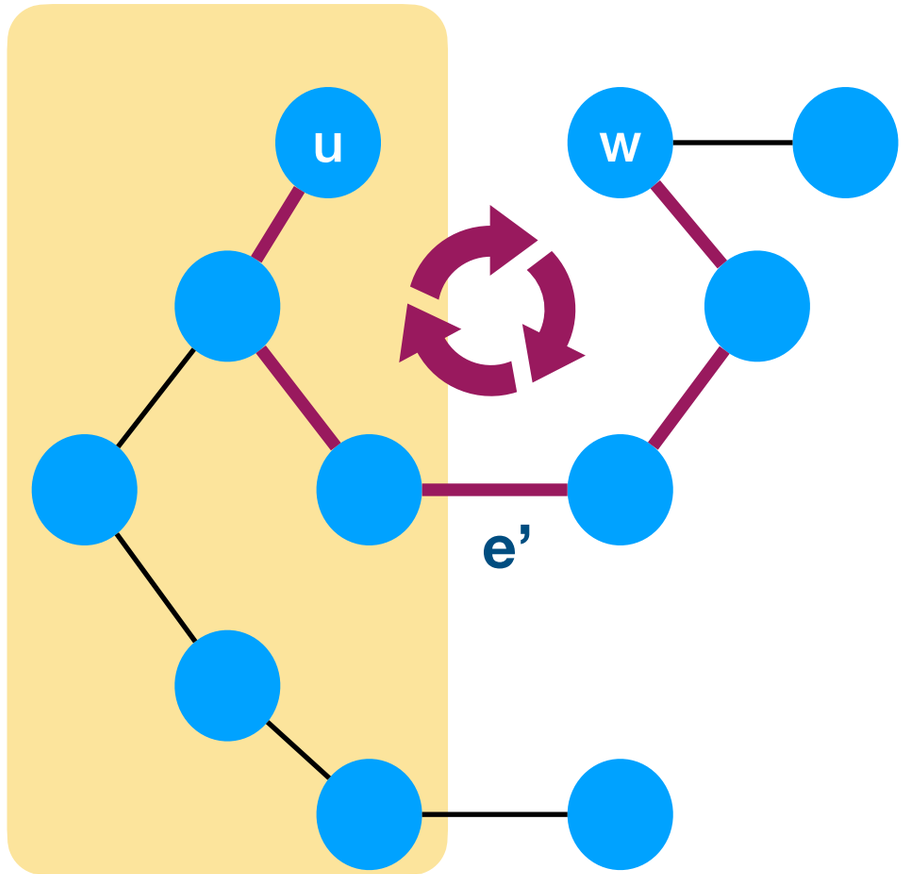
This partitions the nodes into

$S$ (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

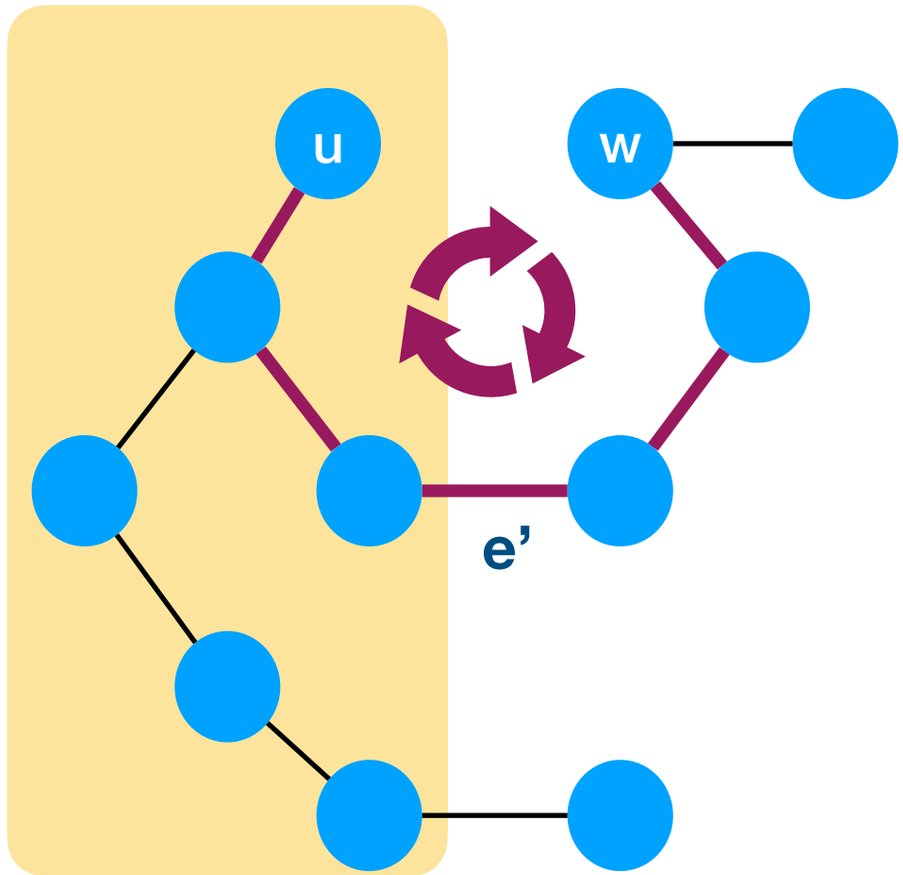
$S$ (containing $u$).

$V$ - $S$ (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

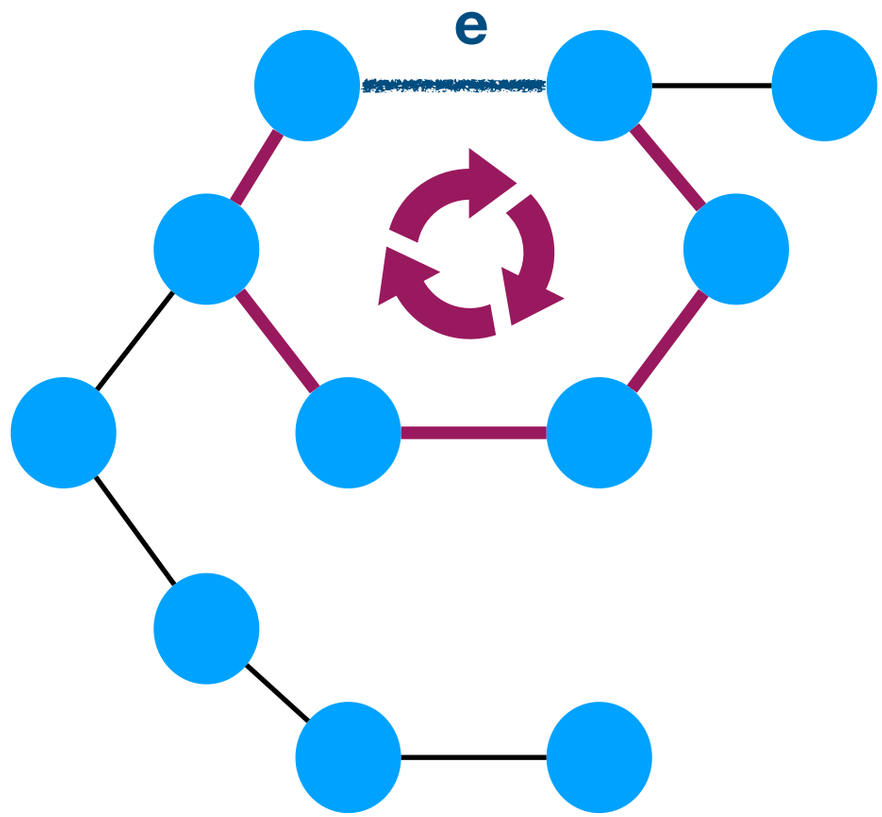
Consider any edge $e$=($v$, $w$) which is removed by Reverse-Delete.

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?

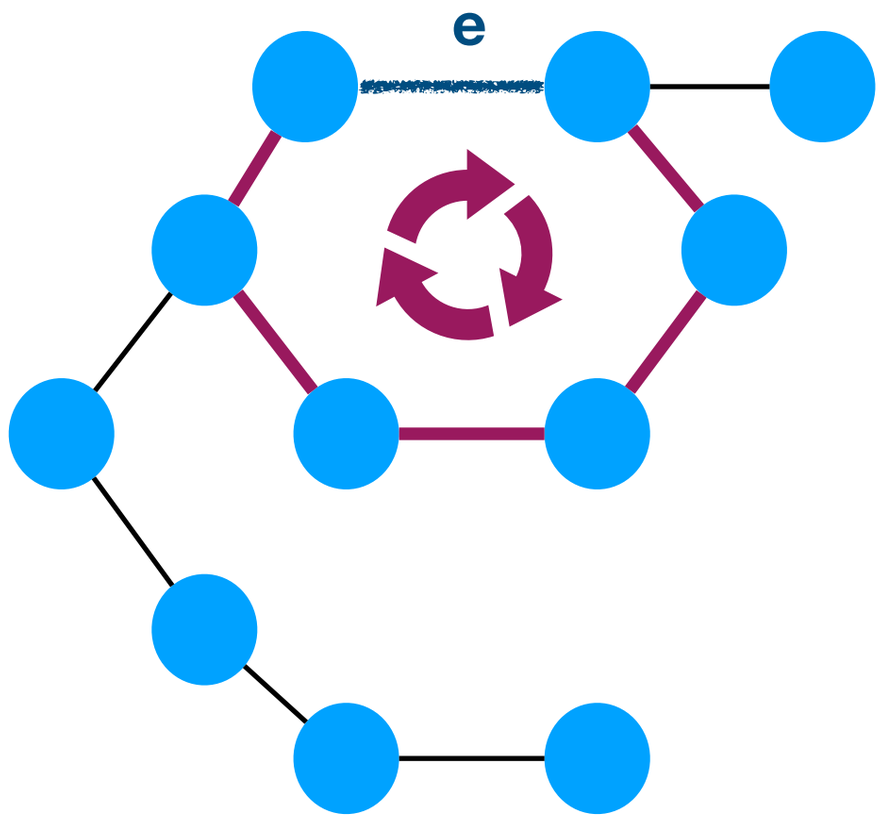
i.e., does it always produce a spanning tree?



# Is it feasible?

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

Is it connected?

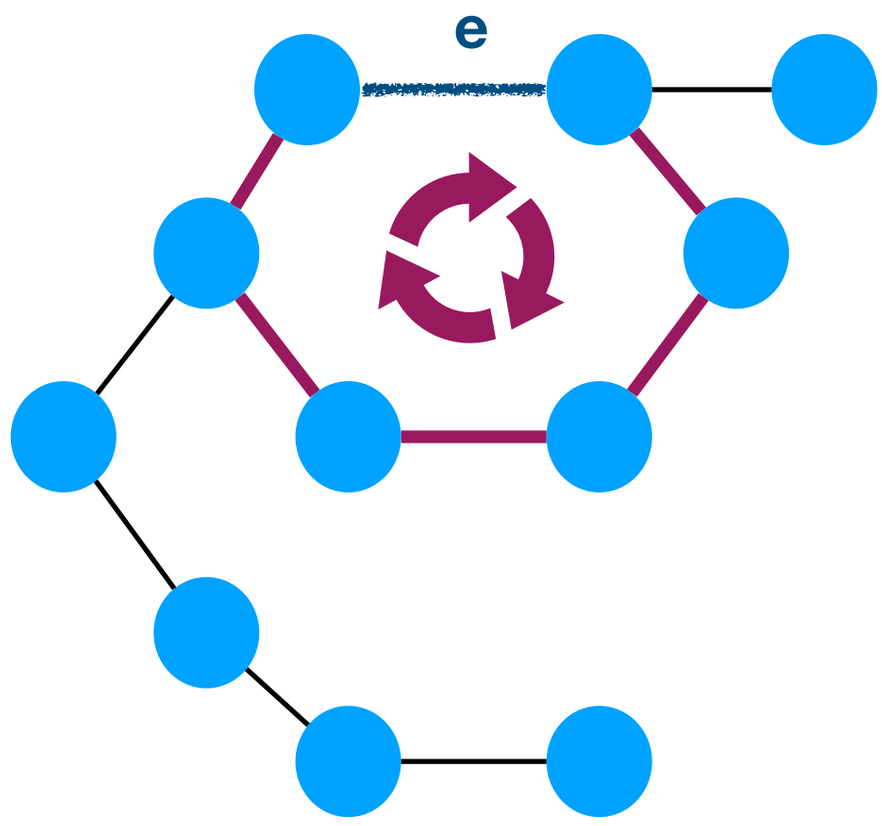


# Is it feasible?

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

Is it connected?

The algorithm will never disconnect the graph.



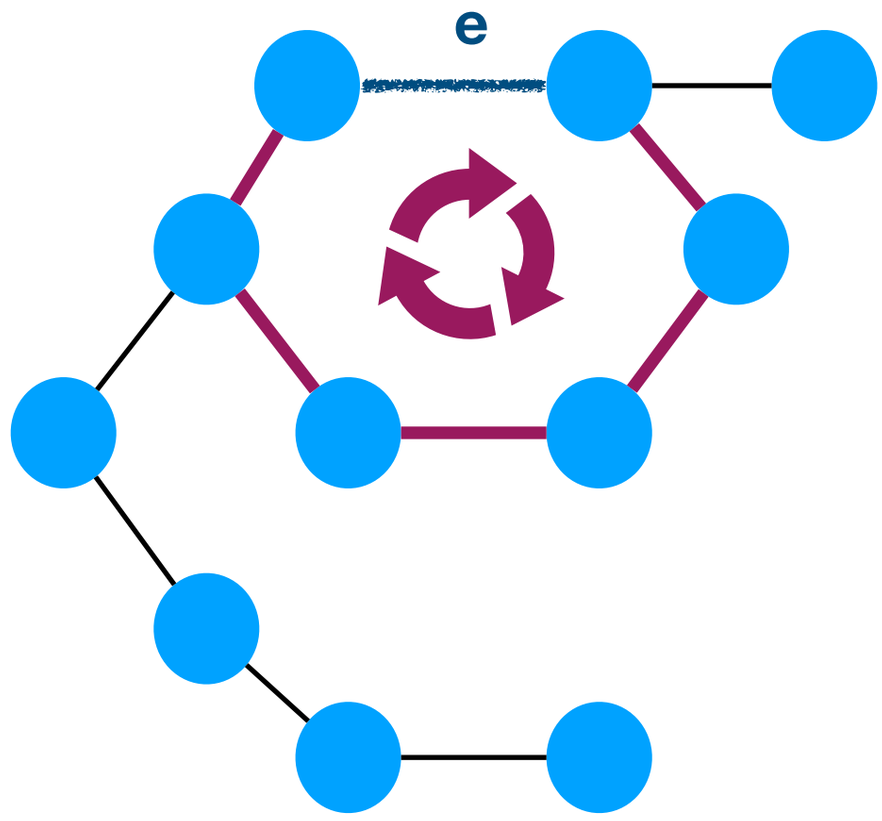
# Is it feasible?

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?

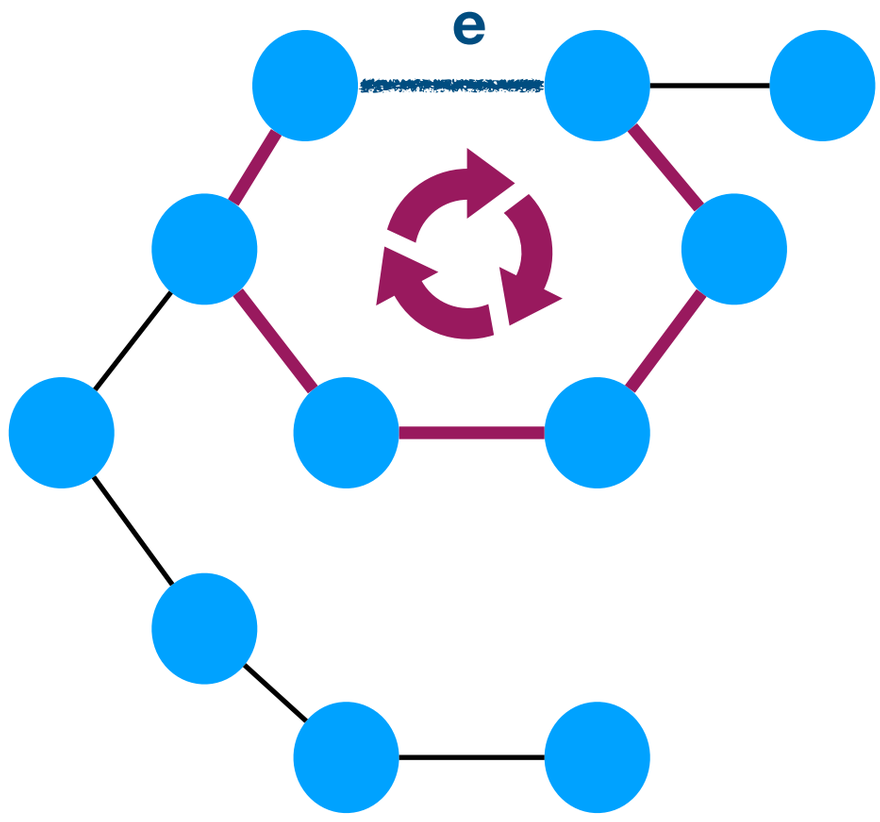
i.e., does it always produce a spanning tree?

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?

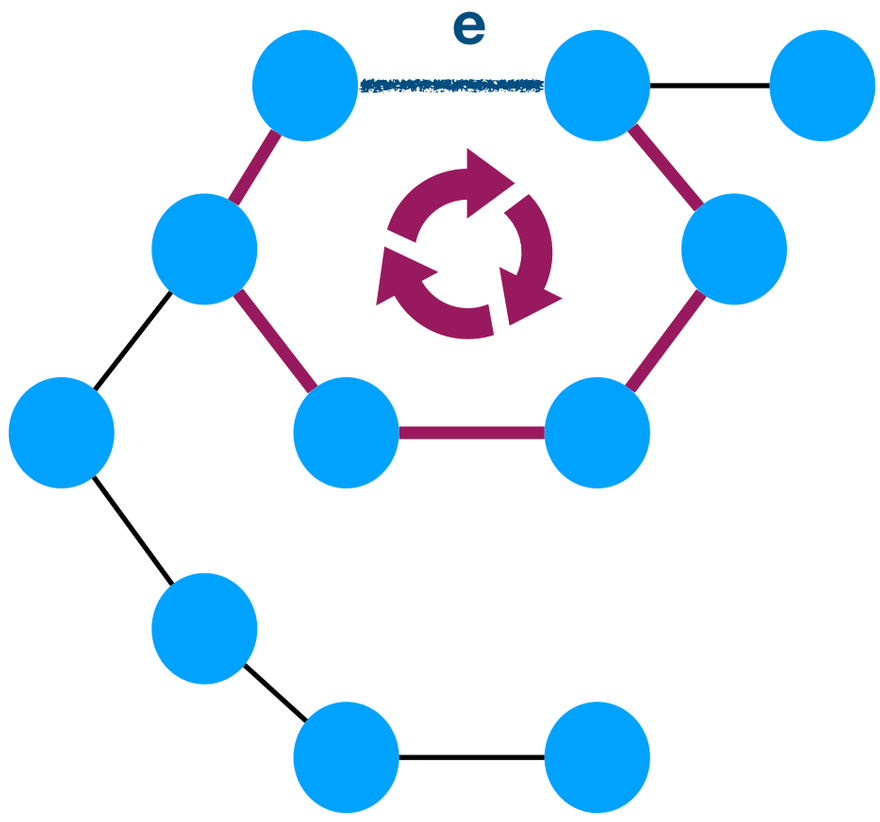
Is it connected?

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?

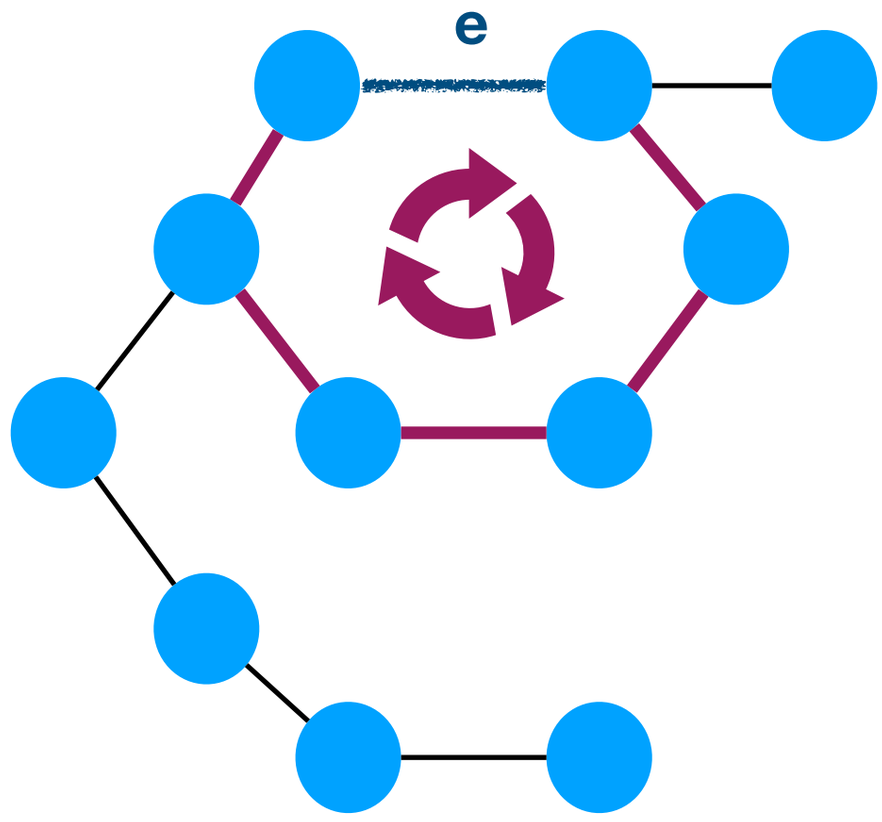
The algorithm will never disconnect the graph.

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.

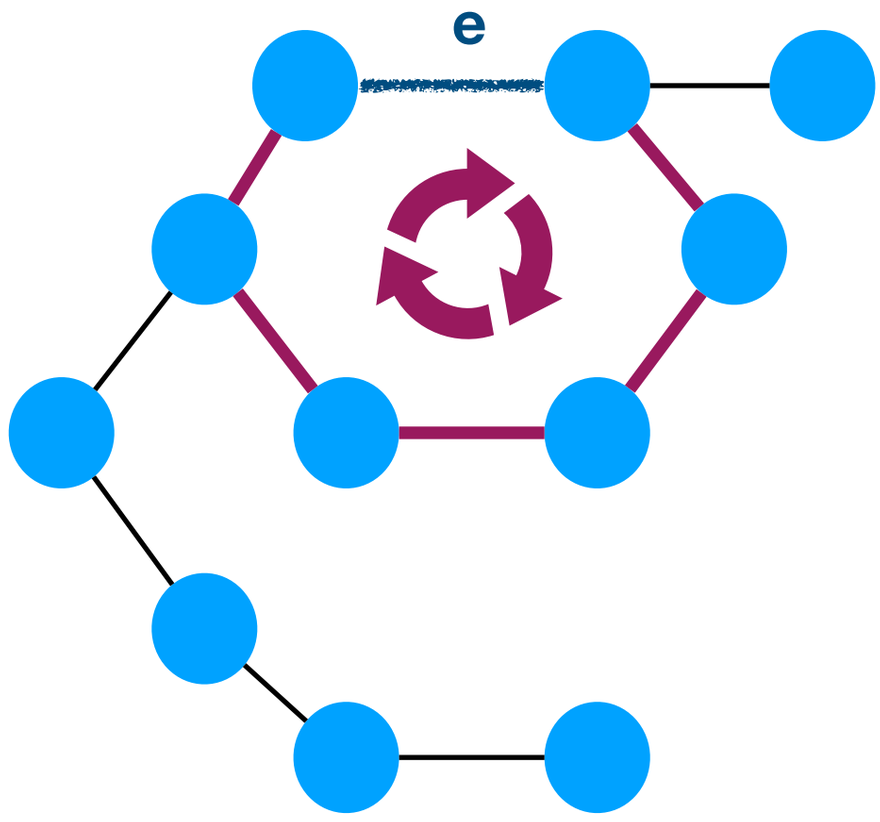
Is it a tree?

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?

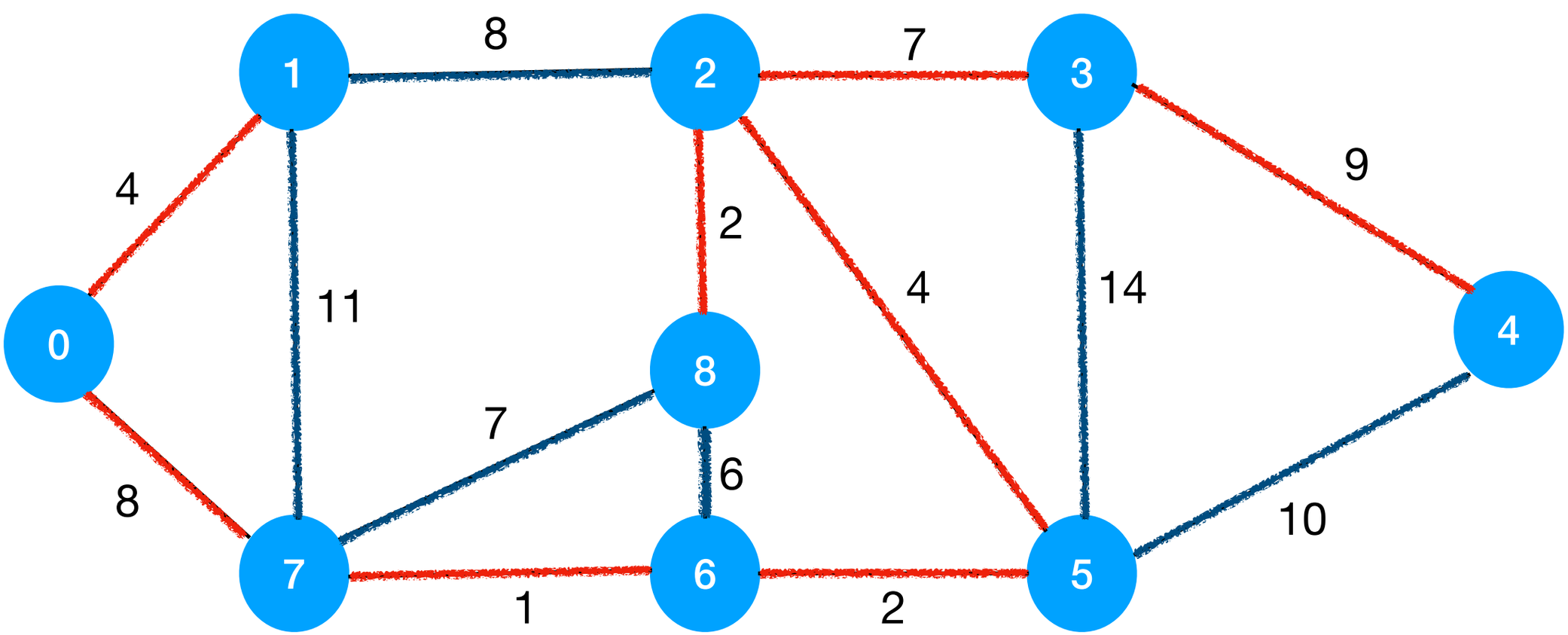
“Assume that all edge costs are distinct”.

What if they are not?

# Are we done?

“Assume that all edge costs are distinct”.

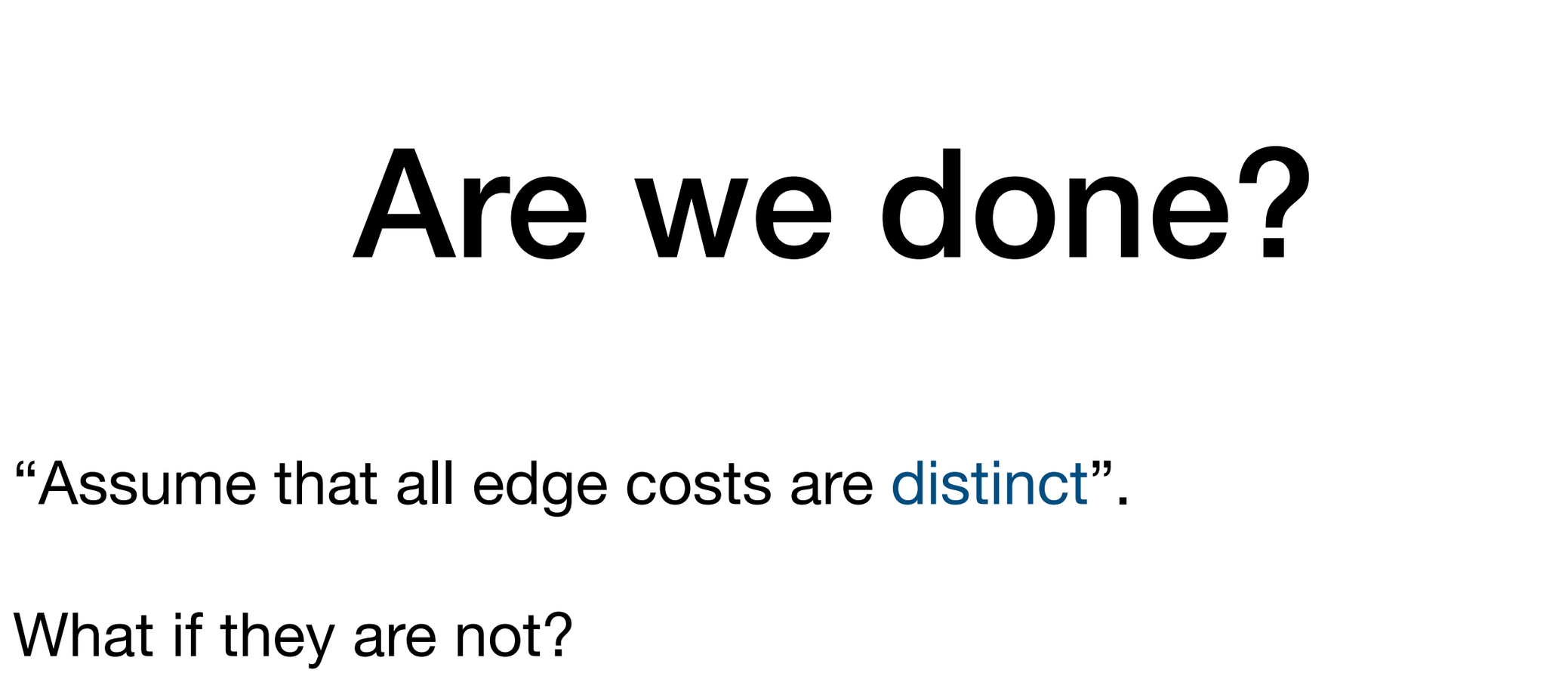
What if they are not?



# 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.

# Non-distinct costs

Take the original instance with non-distinct costs.

Make the costs distinct by adding small numbers $\varepsilon$ to the costs to break ties.

# Non-distinct costs

Take the original instance with non-distinct costs.

Make the costs distinct by adding small numbers $\varepsilon$ to the costs to break ties.

Obtain a perturbed instance.

# Non-distinct costs

Take the original instance with non-distinct costs.

Make the costs distinct by adding small numbers $\varepsilon$ to the costs to break ties.

Obtain a perturbed instance.

Run the algorithm on the perturbed instance.

# Non-distinct costs

Take the original instance with non-distinct costs.

Make the costs distinct by adding small numbers $\varepsilon$ to the costs to break ties.

Obtain a perturbed instance.

Run the algorithm on the perturbed instance.

Output the minimum spanning tree $G_T$.

# Non-distinct costs

Take the original instance with non-distinct costs.

Make the costs distinct by adding small numbers $\varepsilon$ to the costs to break ties.

Obtain a perturbed instance.

Run the algorithm on the perturbed instance.

Output the minimum spanning tree $G_T$.

$G_T$ is a minimum spanning tree on the original instance.

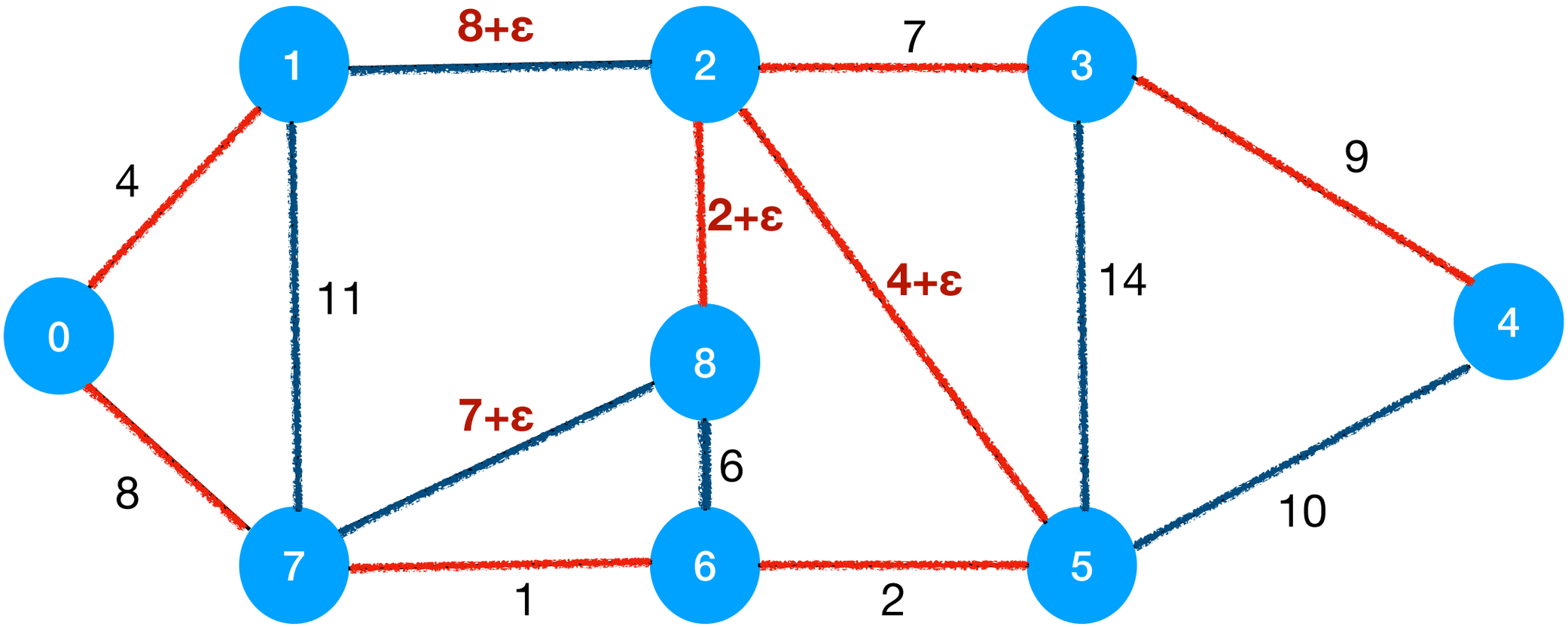
# $G_T$ in the original instance

Suppose that there was a cheaper spanning tree $G_T^*$ on the original instance.

If $G_T^*$ contains different edges with the same costs, it is not cheaper than $G_T^*$ on the original instance.

If $G_T^*$ contains different edges with different costs, we can make $\varepsilon$ 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?

Next lecture!