Trees

Alexander Golovnev

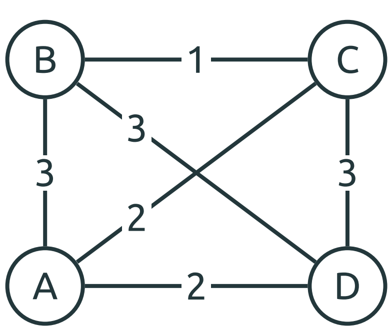
Outline

Road Repair

Trees

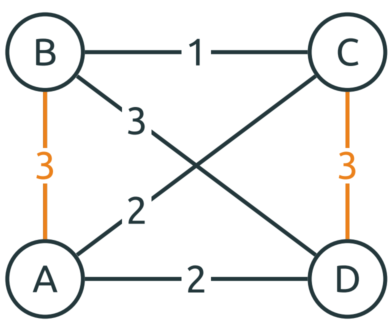
Minimum Spanning Tree

Road Repair



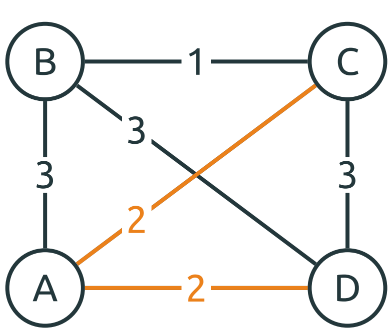
Road Repair

No pair of edges can connect all cities



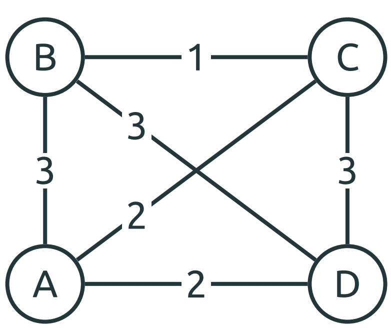
Road Repair

No pair of edges can connect all cities



Road Repair

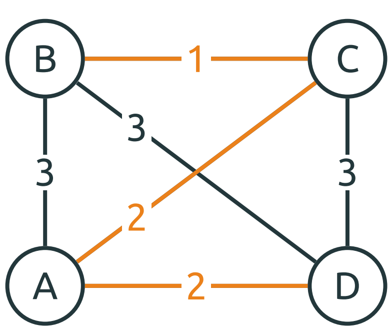
We need at least three edges



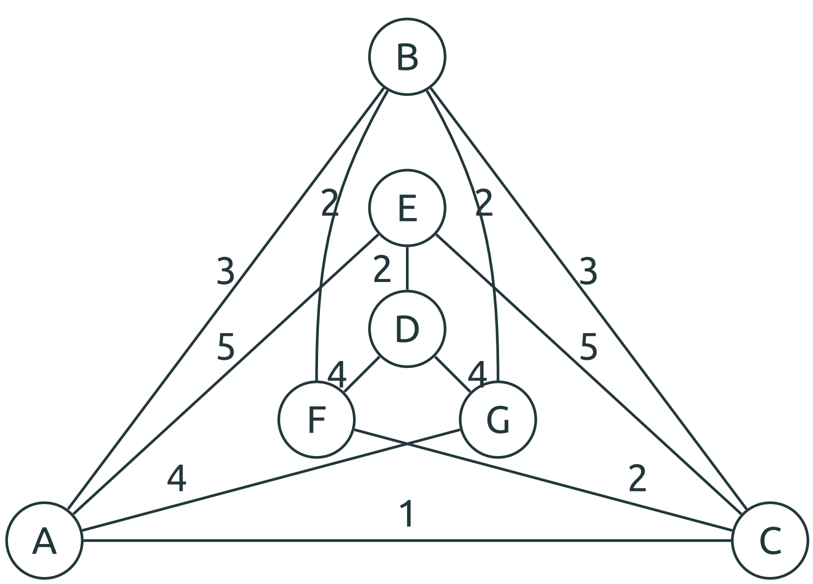
Road Repair

We need at least three edges

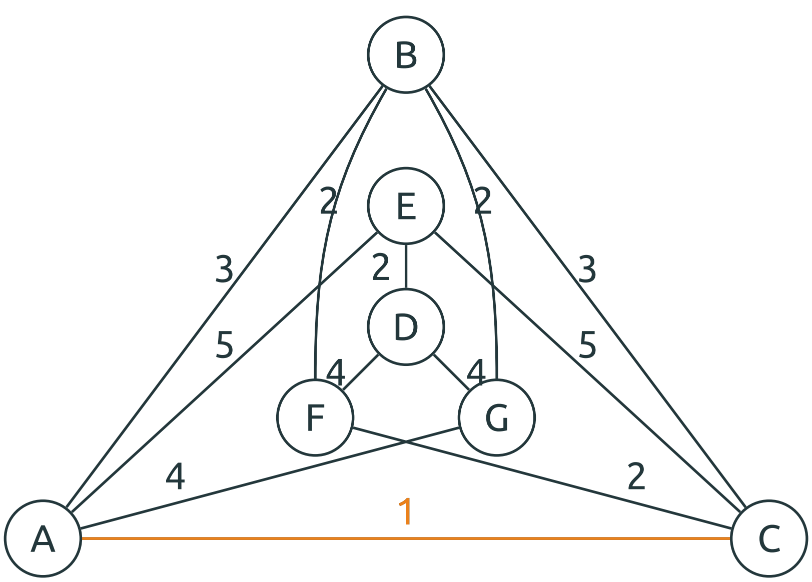
Three shortest edges work!



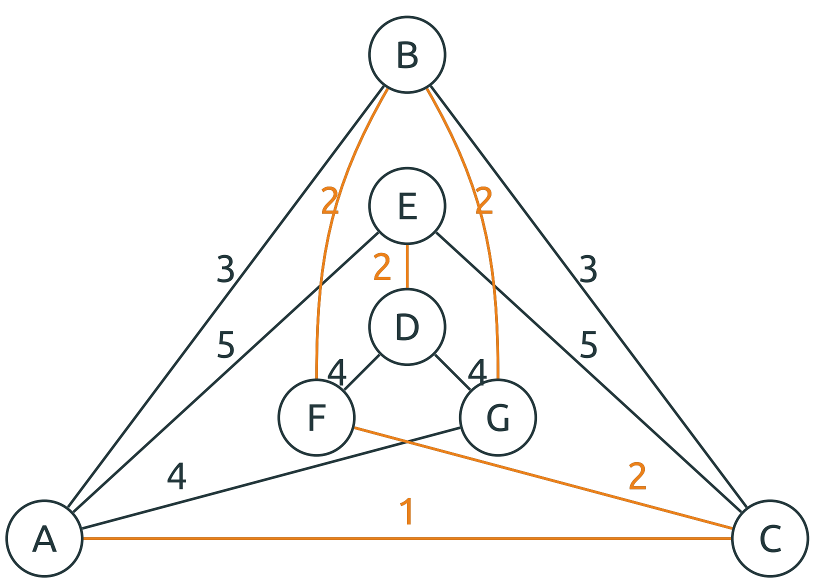
Road Repair



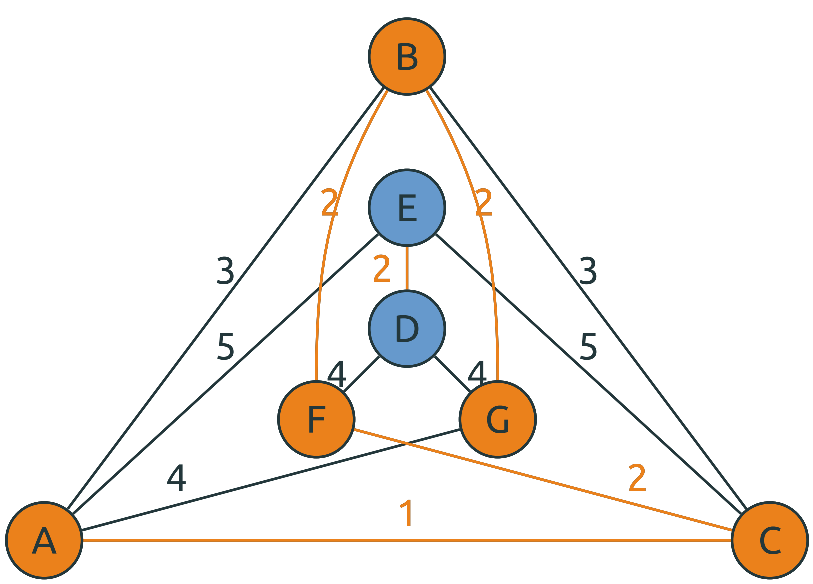
Road Repair

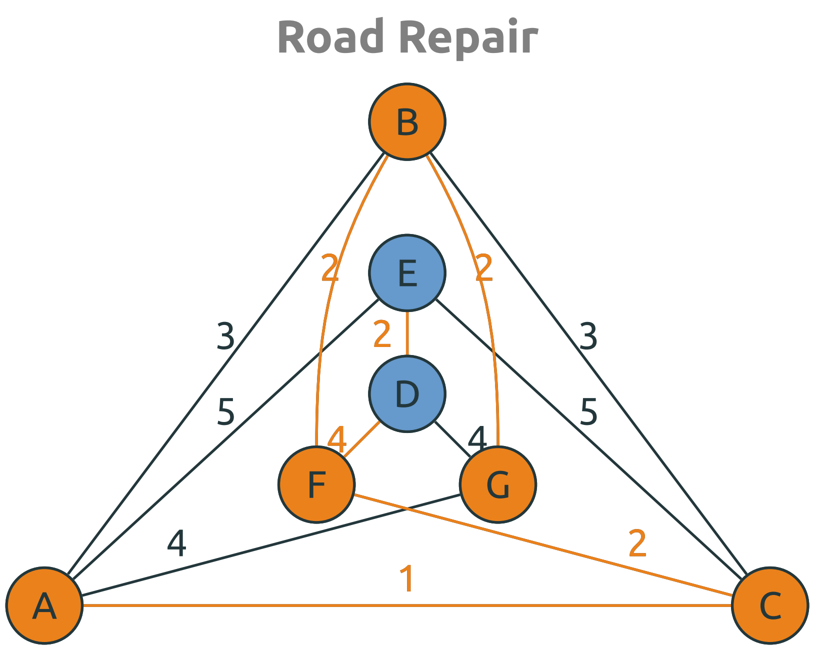


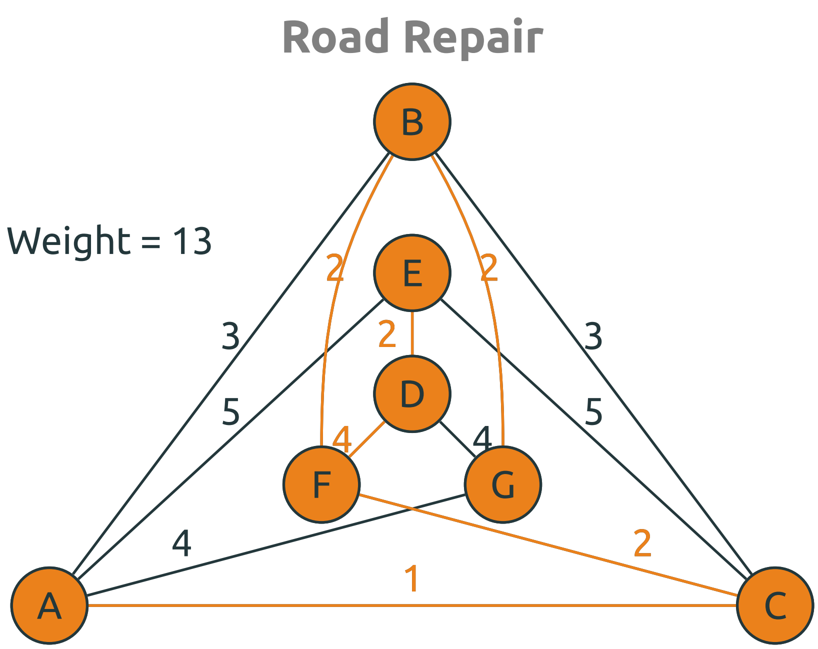
Road Repair



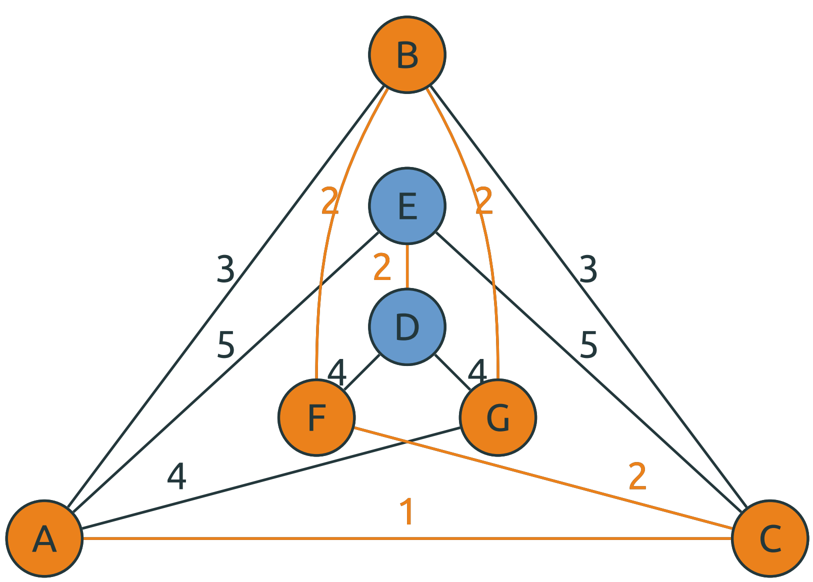
Road Repair



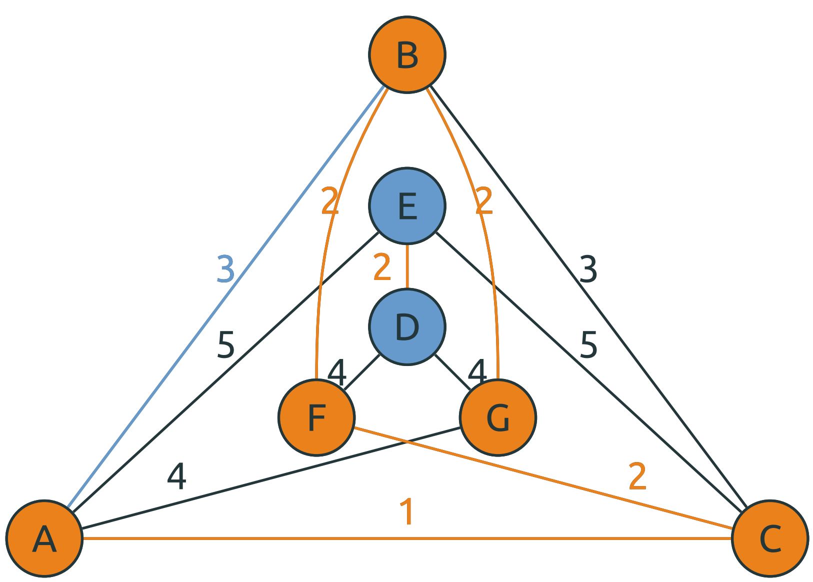

Road Repair
B
E
D
F
G
A
C
3
5
4
2
2
2
4
4
3
5
2
1


Road Repair
Weight = 13
B
E
D
F
G
A
C
2
2
2
3
3
5
5
4
4
4
2
1

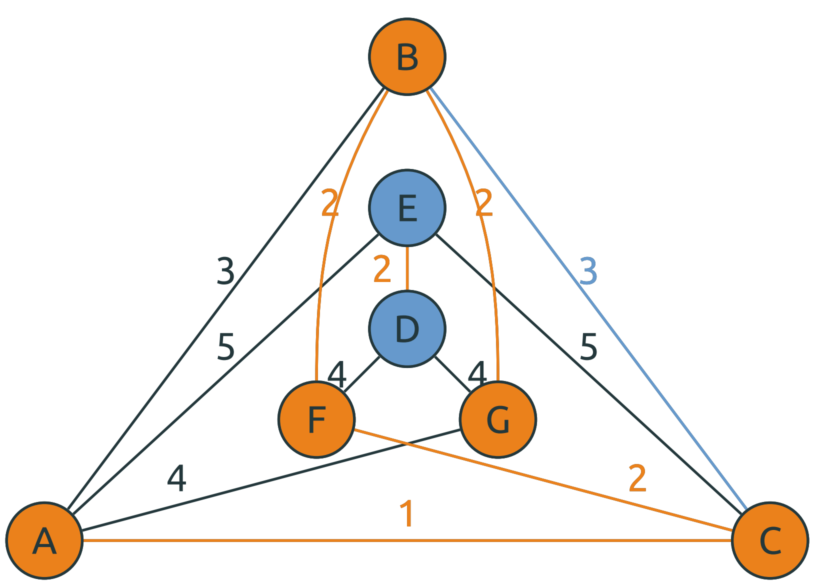
Road Repair



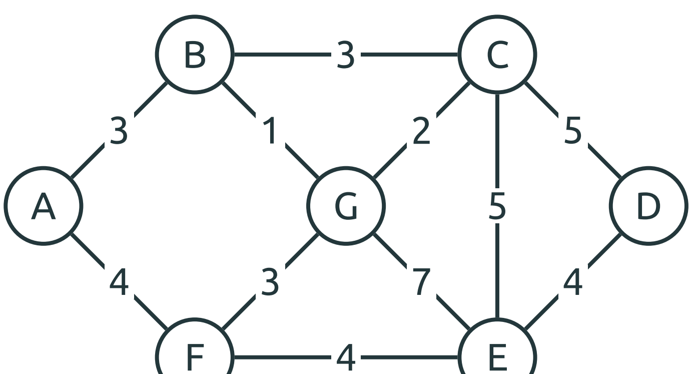
Road Repair



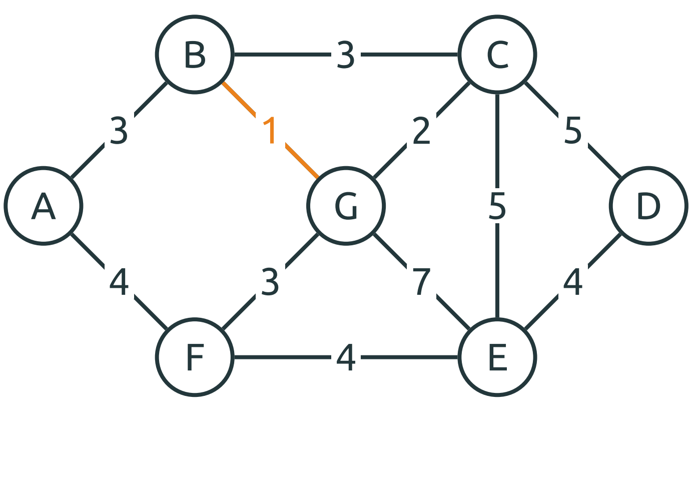
Road Repair



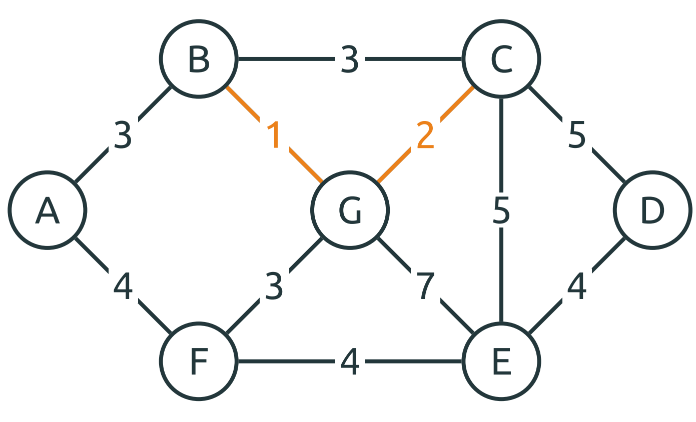
Road Repair



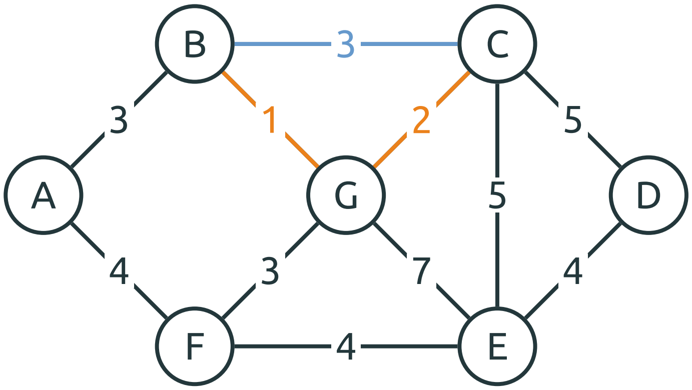
Road Repair



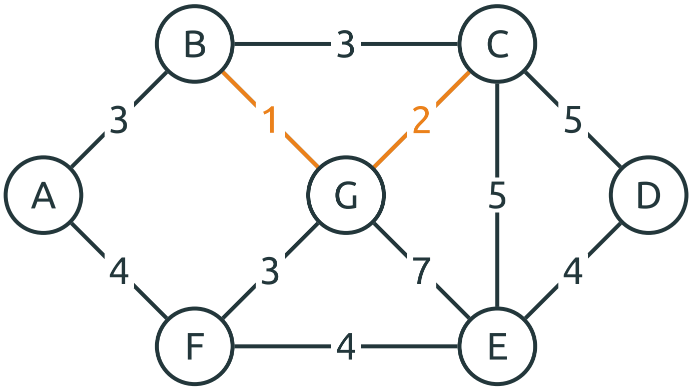
Road Repair



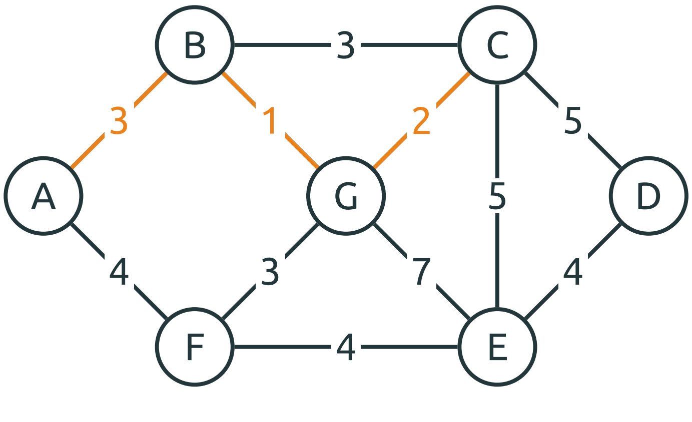
Road Repair



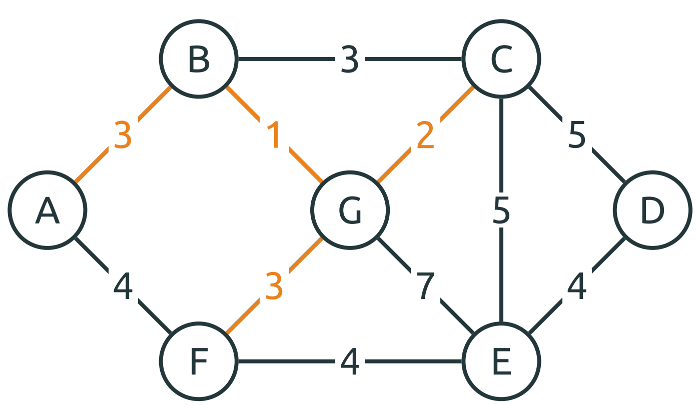
Road Repair



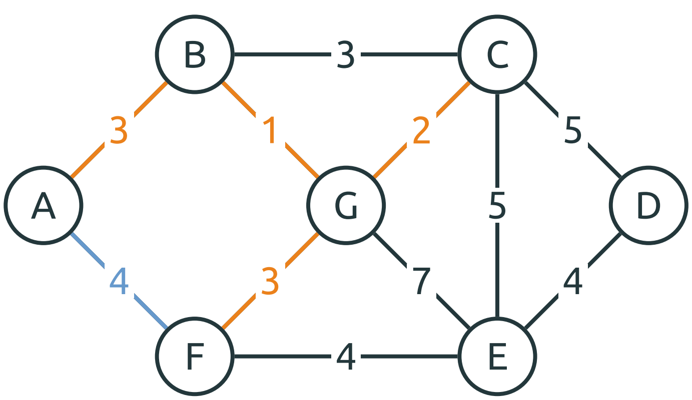
Road Repair



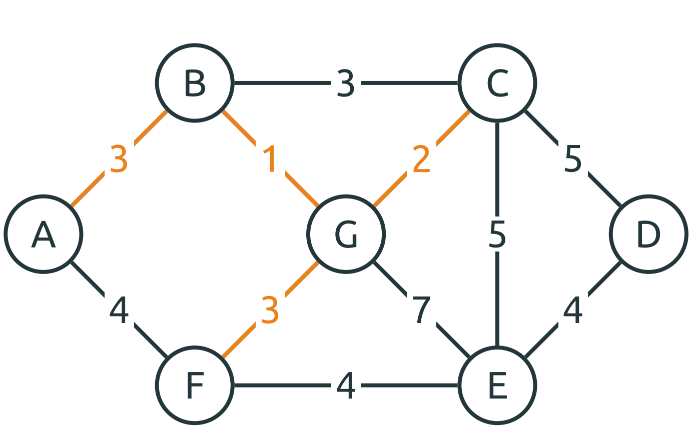
Road Repair



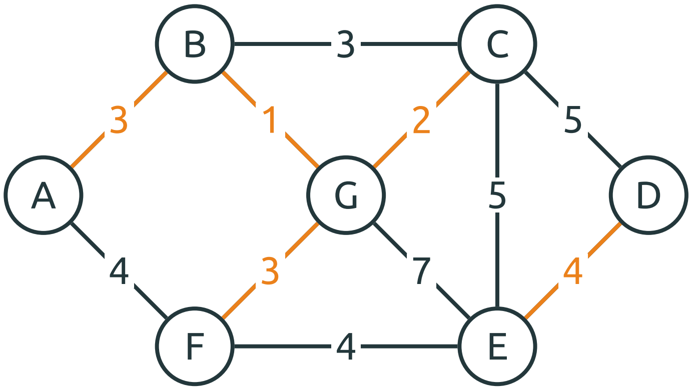
Road Repair



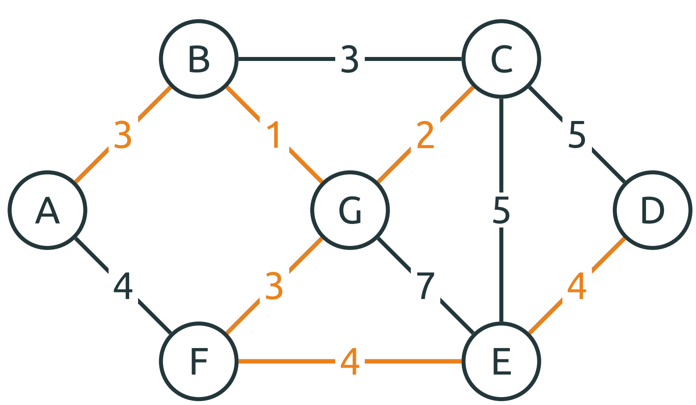
Road Repair



Road Repair



Road Repair



Outline

Road Repair

Trees

Minimum Spanning Tree

Definition

- A tree is a connected graph without cycles

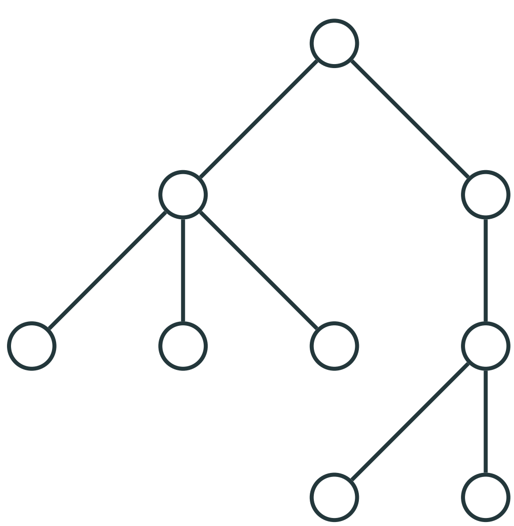
Definition

- A tree is a connected graph without cycles
- A tree is a connected graph on n vertices with $n - 1$ edges

Definition

- A tree is a connected graph without cycles
- A tree is a connected graph on n vertices with $n - 1$ edges
- A graph is a tree if and only if there is a unique simple path between any pair of its vertices

Trees: Examples



Equivalent Definitions

- (I) A tree is a connected graph without cycles
- (II) A tree is a connected graph on n vertices with $n-1$ edges
- (III) A graph is a tree if and only if there is a unique simple path between any pair of its vertices

Equivalent Definitions

- (I) A tree is a connected graph without cycles
- (II) A tree is a connected graph on n vertices with $n-1$ edges
- (III) A graph is a tree if and only if there is a unique simple path between any pair of its vertices
- We'll prove that (I) $\rightarrow$ (II) $\rightarrow$ (III) $\rightarrow$ (I)

(I) $\rightarrow$ (II)

- A connected graph on n vertices without cycles has $n - 1$ edges

(I) $\rightarrow$ (II)

- A connected graph on n vertices without cycles has $n-1$ edges
- Induction on n

(I) $\rightarrow$ (II)

- A connected graph on n vertices without cycles has $n-1$ edges
- Induction on n
- Base case. $n = 1$, 0 edges

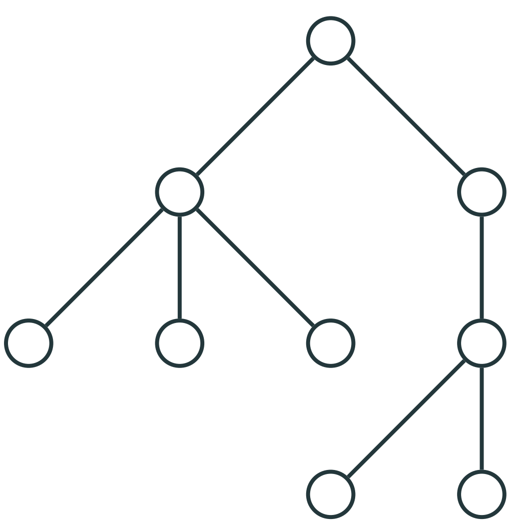
(I) $\rightarrow$ (II)

- A connected graph on n vertices without cycles has $n - 1$ edges
- Induction on n
- Base case. $n = 1$, 0 edges
- Induction hypothesis. Every connected graph on $t \leq k$ vertices has $t - 1$ edges

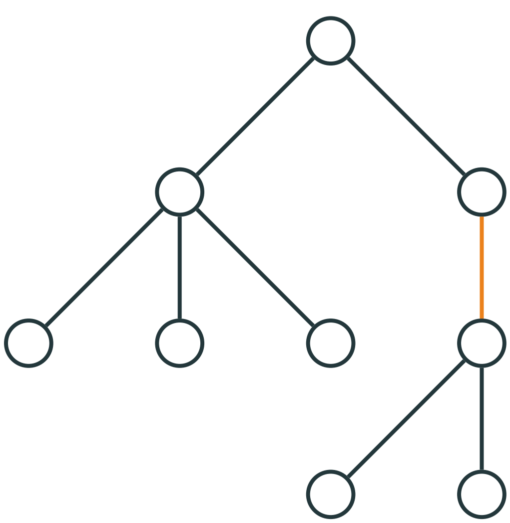
(I) $\rightarrow$ (II)

- A connected graph on n vertices without cycles has $n-1$ edges
- Induction on n
- Base case. $n = 1$, 0 edges
- Induction hypothesis. Every connected graph on $t \leq k$ vertices has $t-1$ edges
- Induction step. Every connected graph on $k+1$ vertices has k edges

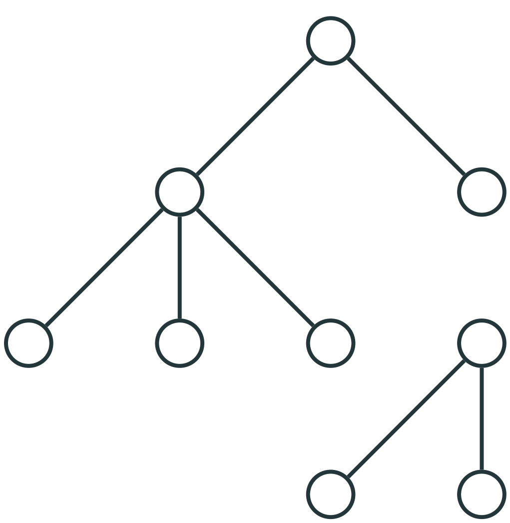
(I) $\rightarrow$ **(II)**



(I) $\rightarrow$ (II)



(I) $\rightarrow$ **(II)**



(I) $\rightarrow$ **(II)**

- Remove an edge

(I) $\rightarrow$ (II)

- Remove an edge
- Two connected graphs without cycles: with n_1 and n_2 vertices, $n_1 + n_2 = n$.

(I) $\rightarrow$ (II)

- Remove an edge
- Two connected graphs without cycles: with n_1 and n_2 vertices, $n_1 + n_2 = n$.
- By Induction hypothesis they have $n_1 - 1$ and $n_2 - 1$ edges

(I) $\rightarrow$ (II)

- Remove an edge
- Two connected graphs without cycles: with n_1 and n_2 vertices, $n_1 + n_2 = n$.
- By Induction hypothesis they have $n_1 - 1$ and $n_2 - 1$ edges
- Thus, the original graph has $(n_1 - 1) + (n_2 - 1) + 1 = n - 1$ edges

Equivalent Definitions

- (I) A tree is a connected graph without cycles
- (II) A tree is a connected graph on n vertices with $n - 1$ edges
- (III) A graph is a tree if and only if there is a unique simple path between any pair of its vertices
- We'll prove that (I) $\rightarrow$ (II) $\rightarrow$ (III) $\rightarrow$ (I)

(II) $\to$ (III)

- Assume there are two paths, they contain a cycle on m vertices and m edges

(II) $\to$ (III)

- Assume there are two paths, they contain a cycle on m vertices and m edges
- Let's recover all edges of the graph one by one starting from this cycle

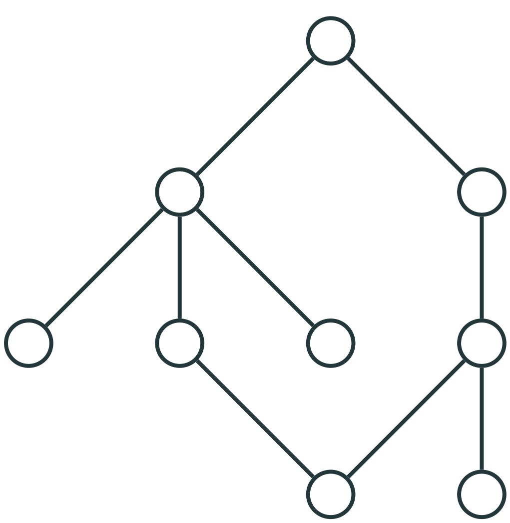
(II) $\to$ (III)

- Assume there are two paths, they contain a cycle on m vertices and m edges
- Let's recover all edges of the graph one by one starting from this cycle
- In order to make it connect all vertices we have to add $n - m$ edges

(II) $\rightarrow$ (III)

- Assume there are two paths, they contain a cycle on m vertices and m edges
- Let's recover all edges of the graph one by one starting from this cycle
- In order to make it connect all vertices we have to add $n - m$ edges
- Then the number of edges is n

(II) $\rightarrow$ **(III)**



Equivalent Definitions

- (I) A tree is a connected graph without cycles
- (II) A tree is a connected graph on n vertices with $n - 1$ edges
- (III) A graph is a tree if and only if there is a unique simple path between any pair of its vertices
- We'll prove that (I) $\rightarrow$ (II) $\rightarrow$ (III) $\rightarrow$ (I)

Equivalent Definitions

- (I) A tree is a connected graph without cycles
- (II) A tree is a connected graph on n vertices with $n-1$ edges
- (III) A graph is a tree if and only if there is a unique simple path between any pair of its vertices
- We'll prove that (I) $\rightarrow$ (II) $\rightarrow$ (III) $\rightarrow$ (I)

Outline

Road Repair

Trees

Minimum Spanning Tree

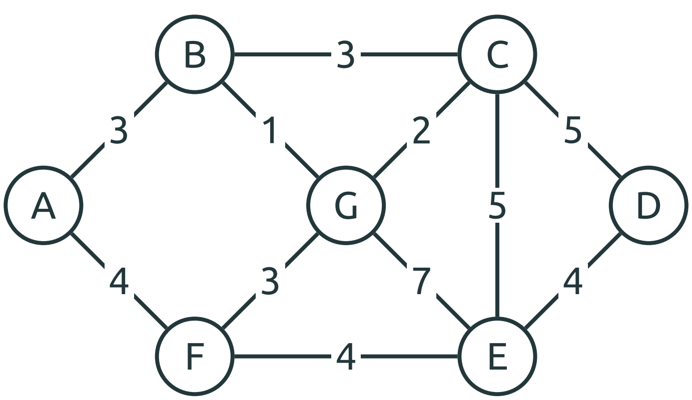
Spanning Trees

- A Spanning Tree of a graph G, is a subgraph of G which is a tree and contains all vertices of G

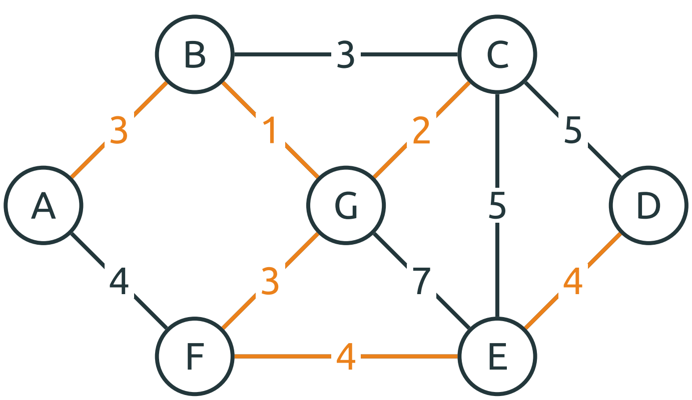
Spanning Trees

- A Spanning Tree of a graph G, is a subgraph of G which is a tree and contains all vertices of G
- A Minimum Spanning Tree of a weighted graph G is a spanning tree of the smallest weight

Minimum Spanning Tree: Examples



Minimum Spanning Tree: Examples



Kruskal's Algorithm

- Start with an empty graph T

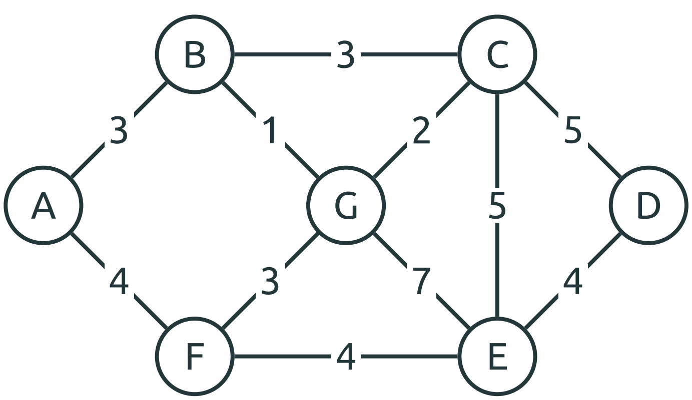
Kruskal's Algorithm

- Start with an empty graph T
- Repeat $n - 1$ times:

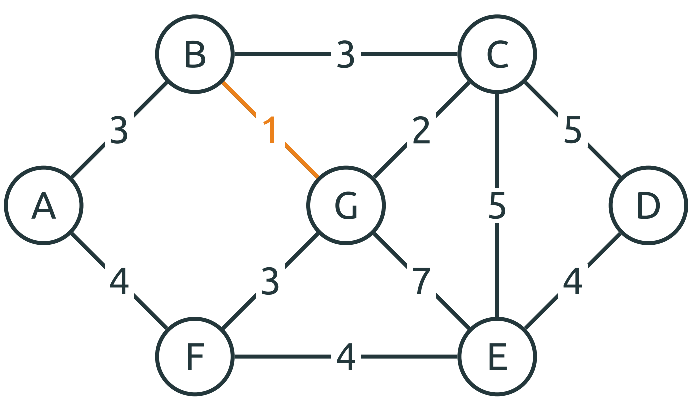
Kruskal's Algorithm

- Start with an empty graph T
- Repeat $n - 1$ times:
- Add to T an edge of the smallest weight which doesn't create a cycle in T

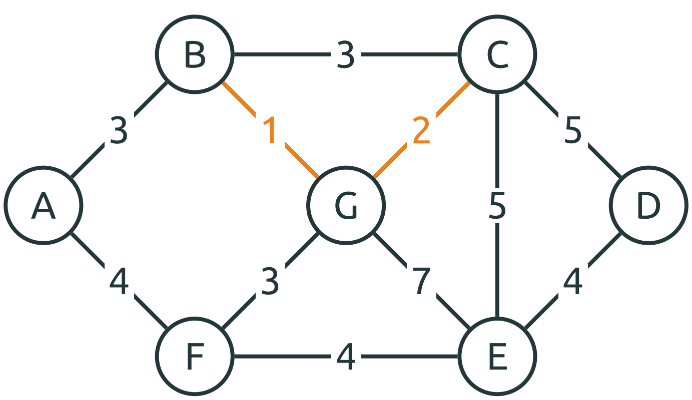
Kruskal's Algorithm: Examples



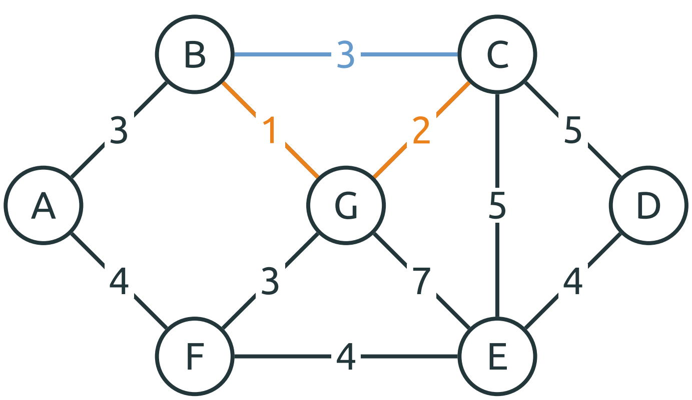
Kruskal's Algorithm: Examples



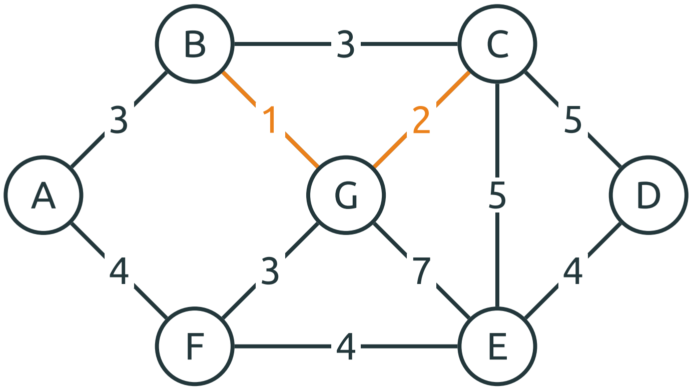
Kruskal's Algorithm: Examples



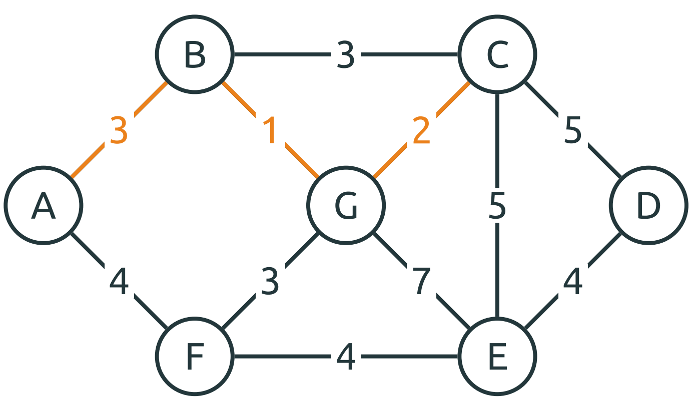
Kruskal's Algorithm: Examples



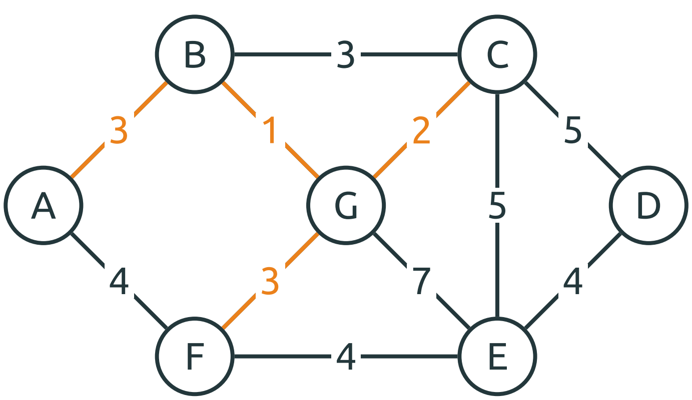
Kruskal's Algorithm: Examples



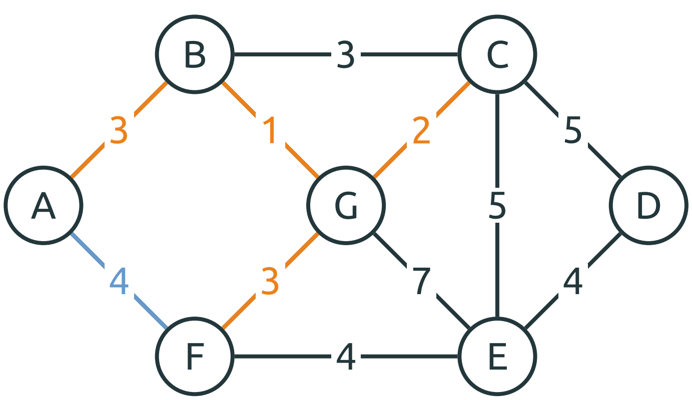
Kruskal's Algorithm: Examples



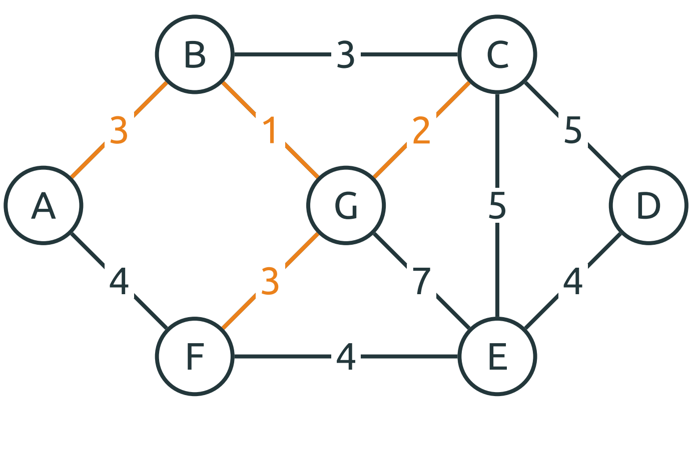
Kruskal's Algorithm: Examples



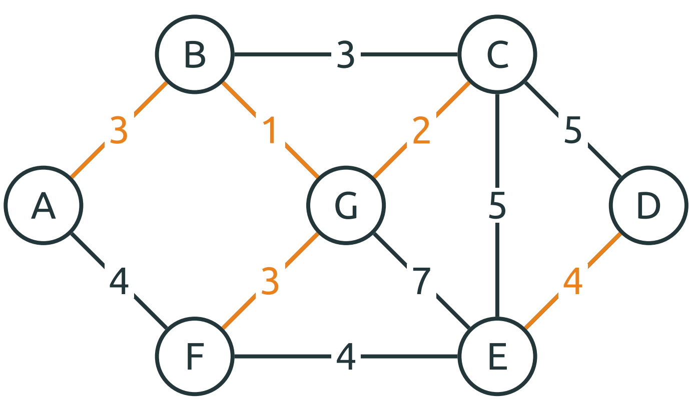
Kruskal's Algorithm: Examples



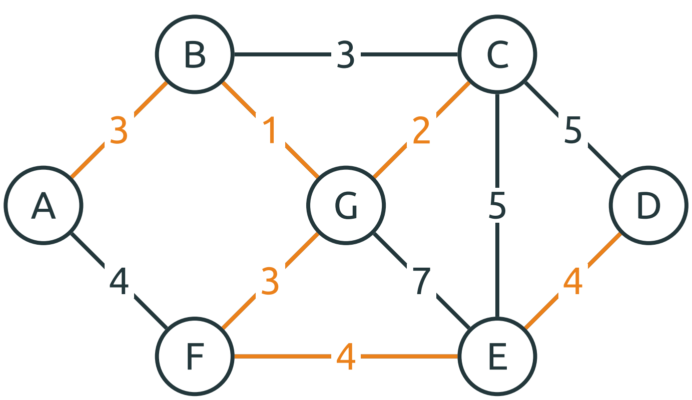
Kruskal's Algorithm: Examples



Kruskal's Algorithm: Examples



Kruskal's Algorithm: Examples



Kruskal's Algorithm: Proof

- We'll show that at every step of the algorithm, there exists a Minimum Spanning Tree which contains all currently chosen edges

Kruskal's Algorithm: Proof

- We'll show that at every step of the algorithm, there exists a Minimum Spanning Tree which contains all currently chosen edges
- Induction on Step Number s

Kruskal's Algorithm: Proof

- We'll show that at every step of the algorithm, there exists a Minimum Spanning Tree which contains all currently chosen edges
- Induction on Step Number s
- Base Case. $s = 0$, empty tree

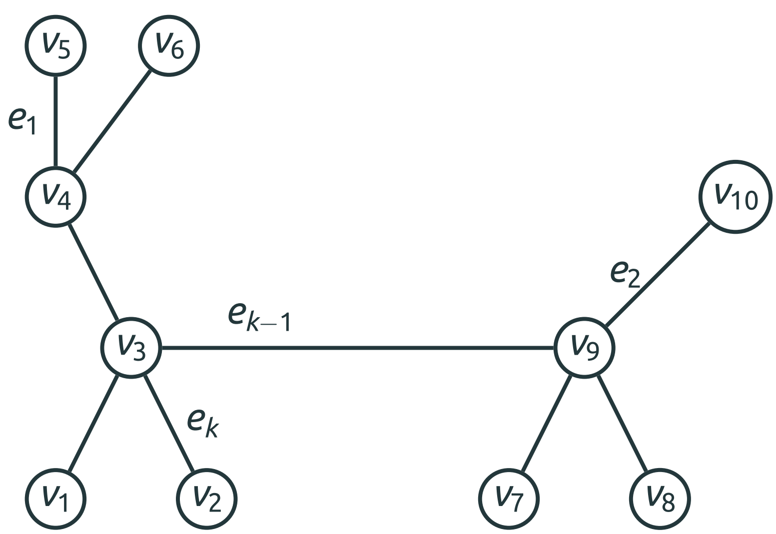
Kruskal's Algorithm: Proof

- We'll show that at every step of the algorithm, there exists a Minimum Spanning Tree which contains all currently chosen edges
- Induction on Step Number s
- Base Case. $s = 0$, empty tree
- Induction hypothesis. True for step $s = k$

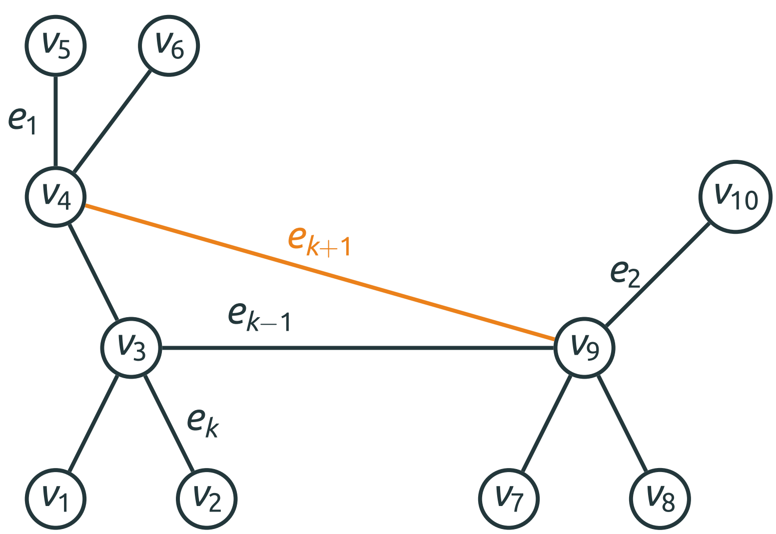
Kruskal's Algorithm: Proof

- We'll show that at every step of the algorithm, there exists a Minimum Spanning Tree which contains all currently chosen edges
- Induction on Step Number s
- Base Case. $s = 0$, empty tree
- Induction hypothesis. True for step $s = k$
- Induction step. We'll show that there exists a Minimum Spanning Tree which contains the first $k + 1$ edges of T

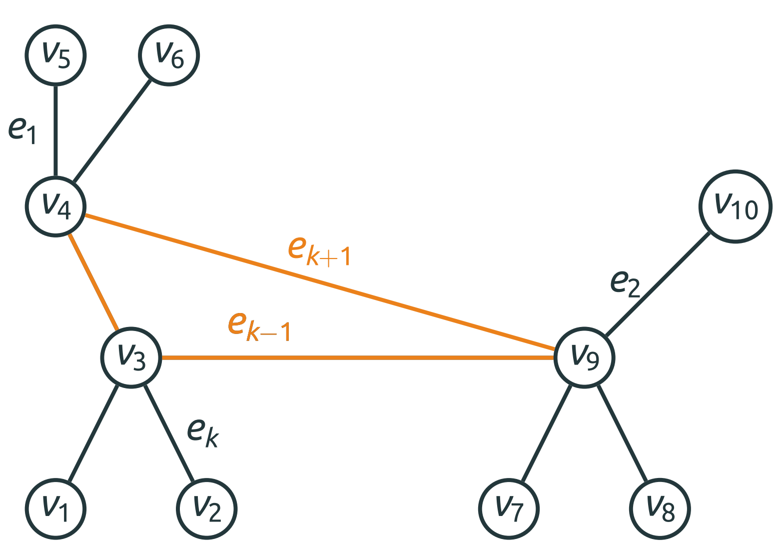
Kruskal's Algorithm: Proof



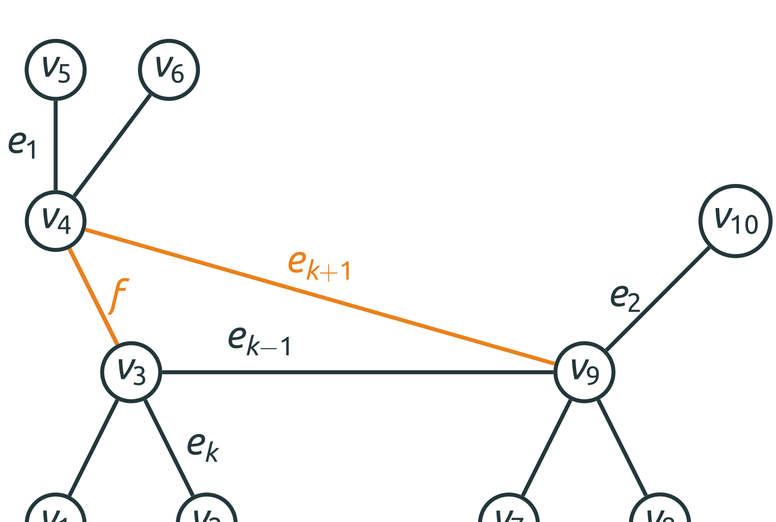
Kruskal's Algorithm: Proof



Kruskal's Algorithm: Proof

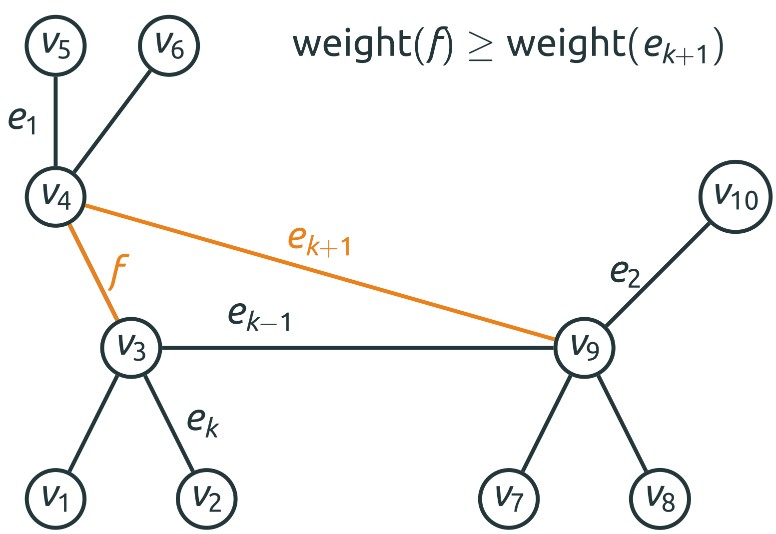


Kruskal's Algorithm: Proof



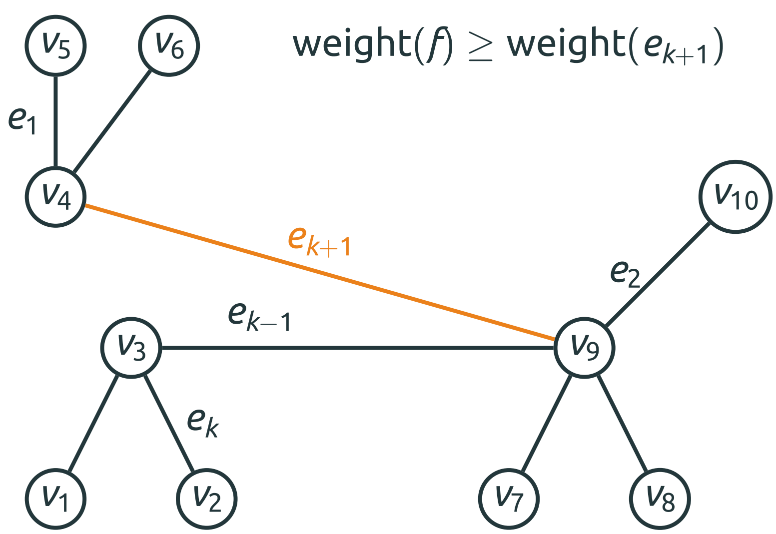
Kruskal's Algorithm: Proof

$$\text{weight}(f) \geq \text{weight}(e_{k+1})$$



Kruskal's Algorithm: Proof

$$\text{weight}(f) \geq \text{weight}(e_{k+1})$$



Kruskal's Algorithm: Proof

$$\text{weight}(f) \geq \text{weight}(e_{k+1})$$