The Greedy Algorithm

More iterative algorithms

Outline

- Minimum Spanning Tree (MST)
  - Optimization
- The Greedy Approach
  - Marginal Gain/Loss
  - Correctness and runtime
  - Optimal for MST

Outline

- Minimum Spanning Tree (MST)
  - Optimization
- The Greedy Approach
  - Marginal Gain/Loss
  - Correctness and runtime
  - Optimal for MST

Learning objectives:
You are able to

- explain the concepts of an optimization problem and marginal gain
- describe the MST problem and the greedy algorithm for the MST problem
- analyze the runtime and correctness of Kruskal’s algorithm

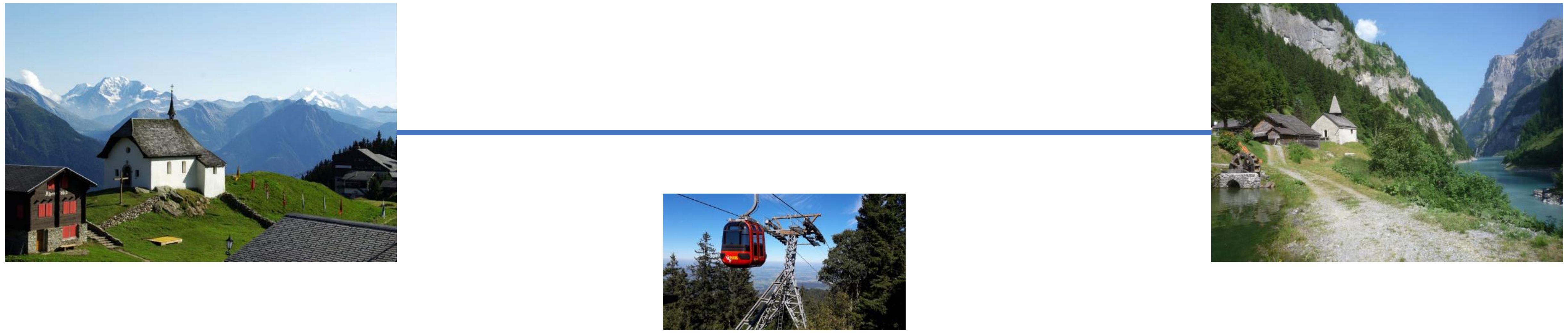
The Minimum Spanning Tree Problem

Task:
A swiss valley with n villages.
We need a transport (cable car) network that connects the villages.

Building and maintaining cable car lines is expensive. We want to minimize the costs.

The Minimum Spanning Tree Problem

Task:
A swiss valley with n villages.
We need a transport (cable car) network that connects the villages.

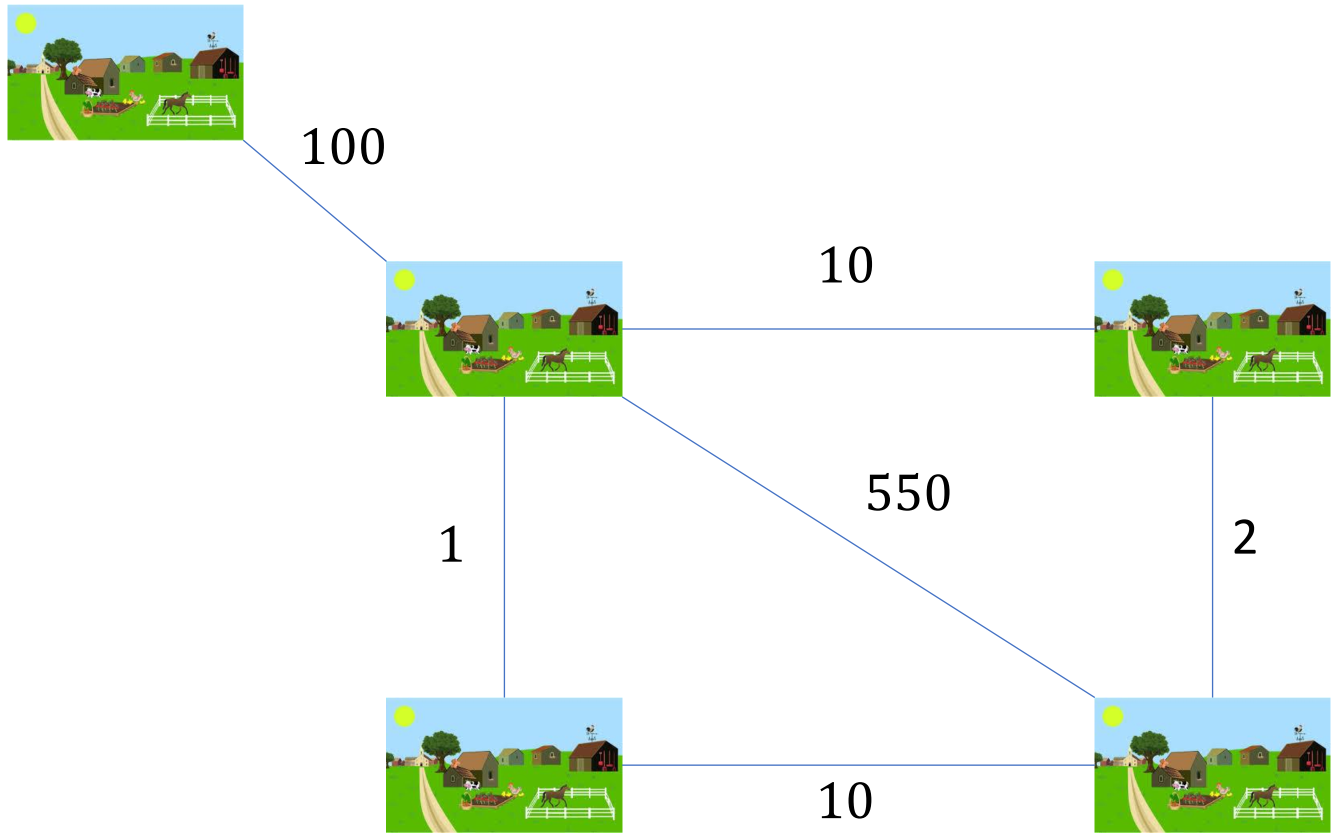


Cost $w(e)$

The Minimum Spanning Tree Problem

Task:
A swiss valley with n villages.
We need a transport (cable car) network that connects the villages.

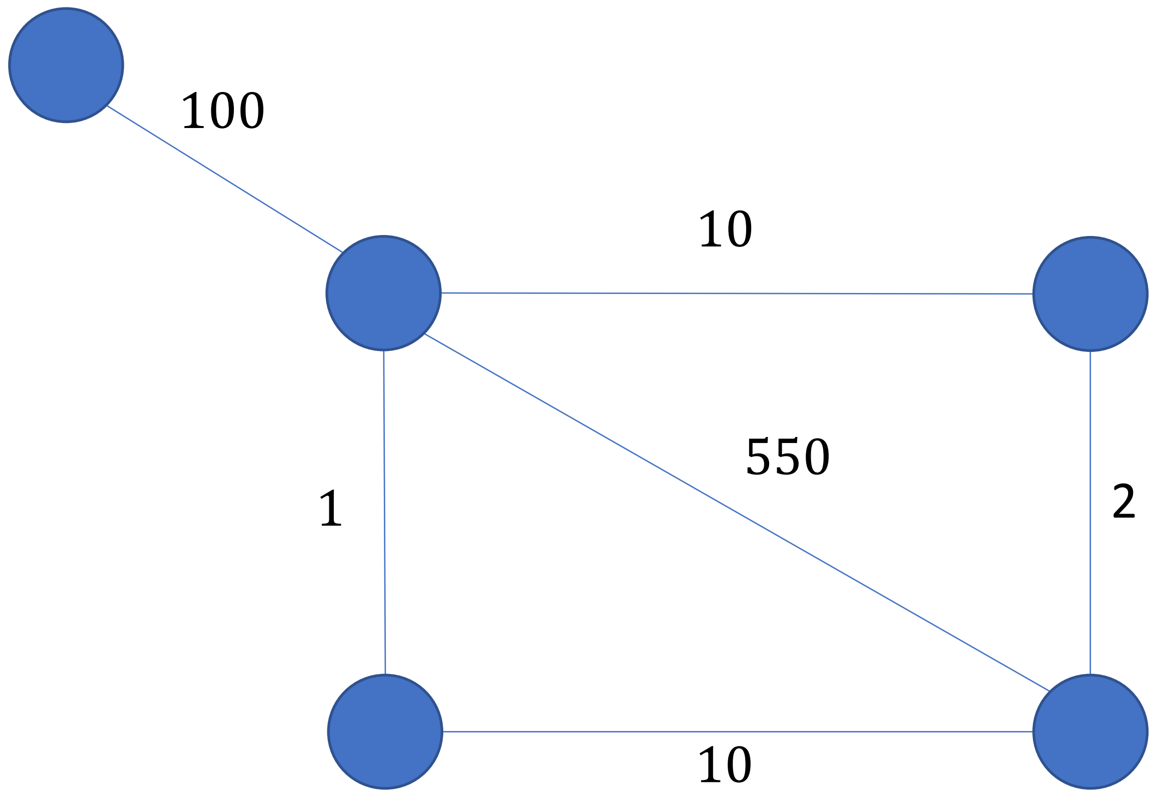
Costs are not uniform. Some villages might not even be possible to (pairwise) connect.



The Minimum Spanning Tree Problem

Task:
A swiss valley with n villages.
We need a transport (cable car) network that connects the villages.

Costs are not uniform. Some villages might not even be possible to (pairwise) connect.



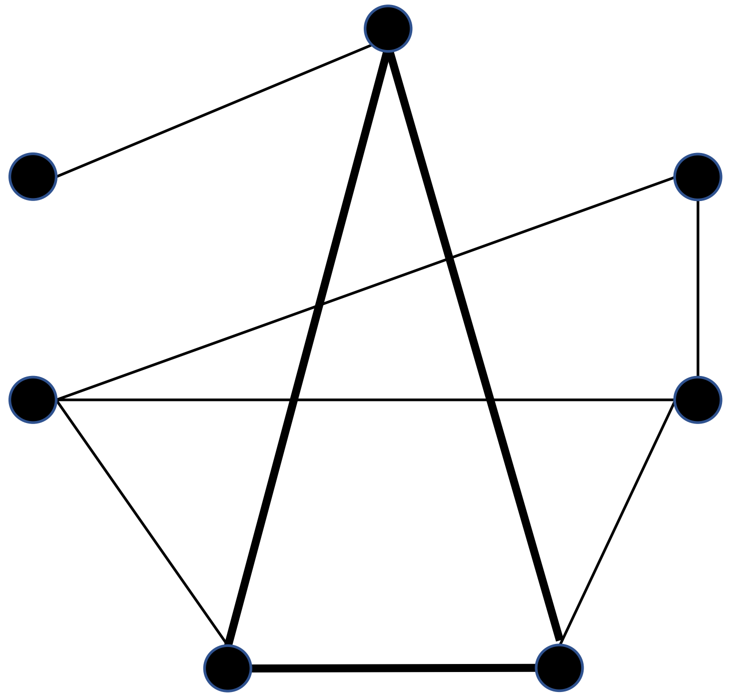
Reminder: Subgraphs and Trees

Subgraph: $G' = (V', E')$ of $G = (V, E)$
- $V' \subseteq V, E' \subseteq E.$
- G' does not need to be connected.
- Sometimes we write: $G' \subseteq G$

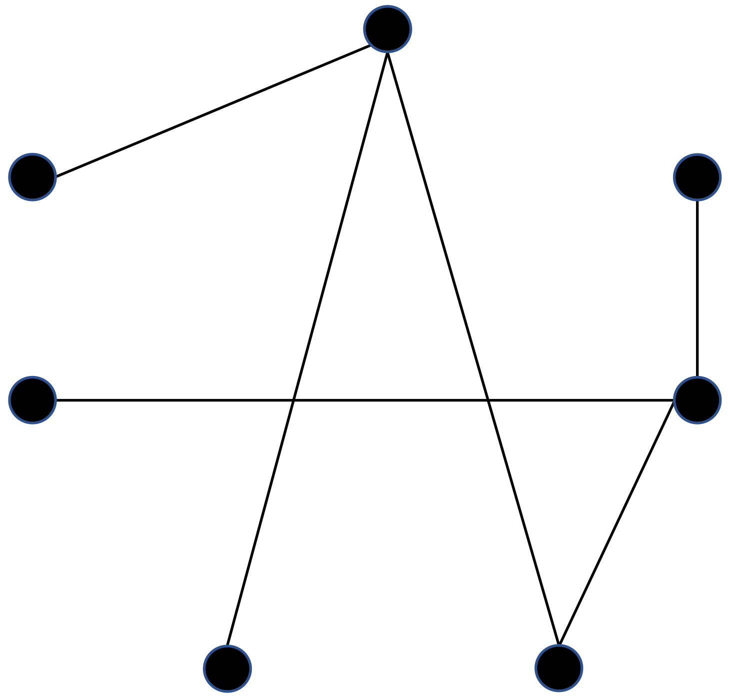
Cycle: $C = (V, E)$
- C is a connected graph
- For all $v \in V$, $\deg(v) = 2$

Tree: $T = (V, E)$
- T is a graph
- T *contains* no cycles, i.e., no subgraph of T is a cycle

Bold edges form a cycle. This graph is not a tree



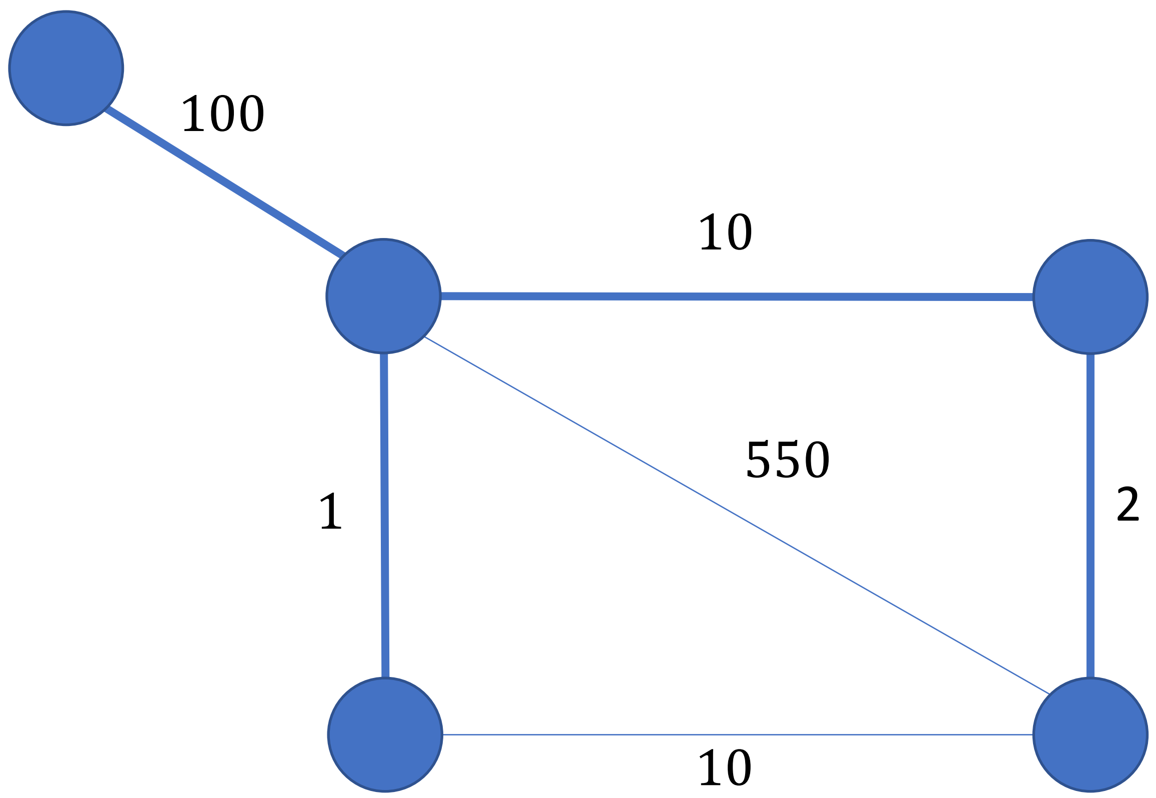
This graph is a tree.



The Minimum Spanning Tree Problem

Task:
A swiss valley with n villages.
We need a transport (cable car) network that connects the villages.

Spanning Tree:
A connected subtree that contains (spans) all nodes.



The Minimum Spanning Tree Problem

Input:
A graph $G = (V, E)$ of n nodes and m edges.
Each edge $e \in E$ is assigned a positive integer weight $w(e)$.

Output:
A subgraph that connects all nodes and has the smallest possible weight.

The Minimum Spanning **Tree** Problem

Observation:
The minimum weight spanning subgraph is a tree.

The Minimum Spanning **Tree** Problem

Observation:
The minimum weight spanning subgraph is a tree.

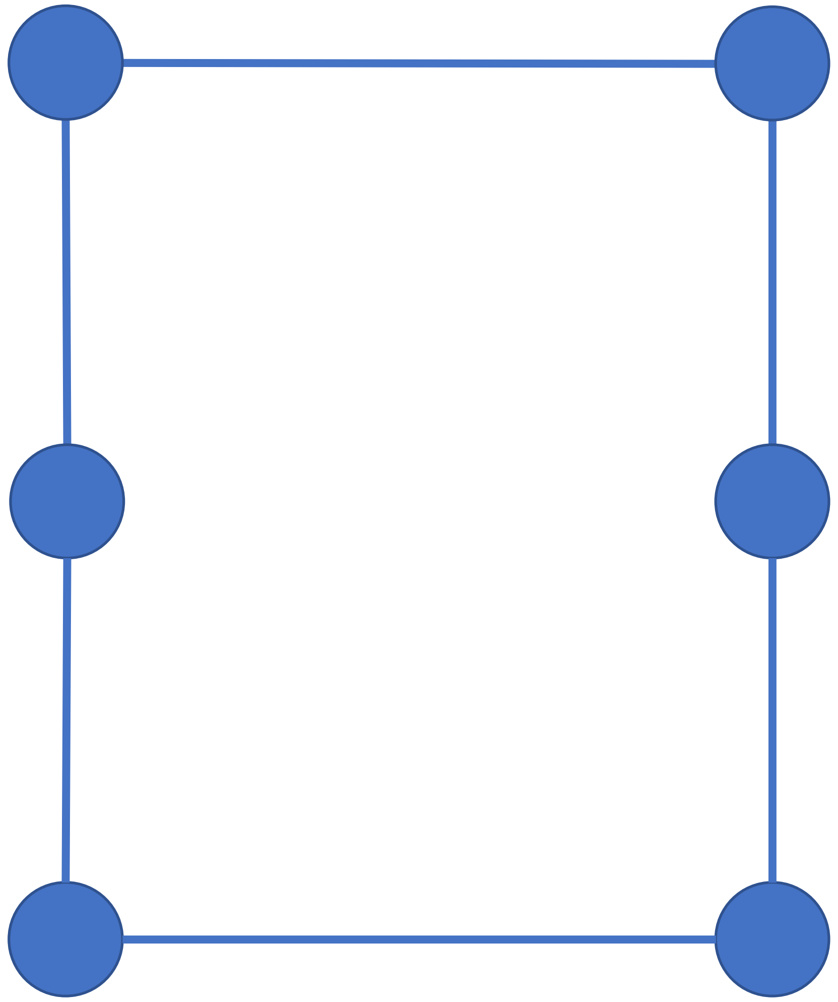
Suppose that the minimum weight spanning subgraph G' is not a tree.

By definition, G' contains a cycle

The Minimum Spanning **Tree** Problem

Suppose that the minimum weight spanning subgraph G' is not a tree.

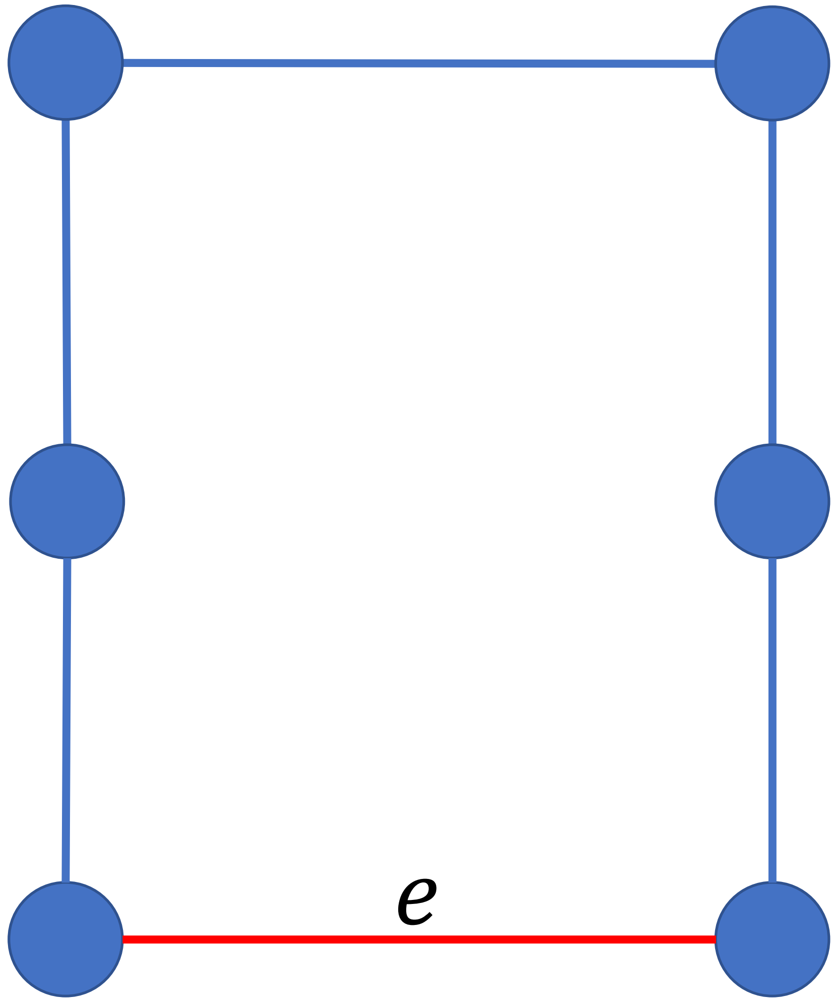
By definition, G' contains a cycle



The Minimum Spanning **Tree** Problem

Suppose that the minimum weight spanning subgraph G' is not a tree.

By definition, G' contains a cycle



Consider any edge e on the cycle

Remove edge e:
Subgraph gets lighter, stays connected and still spans all nodes.

Contradiction.

The Minimum Spanning **Tree** Problem

Suppose that the minimum weight spanning subgraph G' is not a tree.

→

Cycles can be removed: Contradiction

→

Observation:
The minimum weight spanning subgraph is a tree.

Outline

- Minimum Spanning Tree (MST)
  - Optimization

- The Greedy Approach
  - Marginal Gain/Loss
  - Correctness and runtime
  - Optimal for MST

Optimization Problems

Input:
A graph $G = (V, E)$ of n nodes and m edges.
Every edge $e \in E$ is assigned a positive integer weight $w(e)$.

Output:
A subtree that connects all nodes and has the smallest possible weight.

Optimization Problems

Minimize
x

Objective function $\boldsymbol{f(x)}$

Subject to

Constraints $\boldsymbol{g_1(x)}$
$\boldsymbol{g_2(x)}$
…
$\boldsymbol{g_m}$

Optimization Problems

Or maximize

Minimize
x

Objective function $f(x)$

Subject to

Constraints $g_1(x)$
$g_2(x)$
…
g_m

Optimization Problems

Weight of the tree $w(T)$

Minimize
T

$\boldsymbol{f}(T) = \boldsymbol{\Sigma}_{\boldsymbol{e} \in T} \boldsymbol{w}(\boldsymbol{e})$

Subject to

T is a spanning tree of input graph G.

Specify that the solution must be valid

Outline

- Minimum Spanning Tree (MST)
  - Optimization

- The Greedy Approach
  - Marginal Gain/Loss
  - Correctness and runtime
  - Optimal for MST

The Greedy Algorithm

Start with $T = \emptyset$
Marginal gain of $\boldsymbol{e}$:
$$f(T) - f(T \cup e) = -w(e)$$

Iteratively add the edge with the largest marginal gain.
Don't add edges that create cycles

A very typical iterative algorithm

The Greedy Algorithm

Or "loss"

Start with $T = \emptyset$
Marginal gain of $\boldsymbol{e}$:
$$f(T) - f(T \cup e) = -w(e)$$

Iteratively add the edge with the largest marginal gain.
Don't add edges that create cycles

A very typical iterative algorithm

The Greedy Algorithm

Or "loss"

Start with $T = \emptyset$
Marginal gain of e:
$f(T) - f(T \cup e) = -w(e)$

Iteratively add the edge with the largest marginal gain.
Don't add edges that create cycles

Read:
Add the best edge that makes sense

A very typical iterative algorithm

Kruskal’s Algorithm

Input edge list E
Forest $F = (V, \emptyset)$
While($\boldsymbol{F}$ is not a single connected component)

1. Find the edge $e \in E$ with smallest $w(e)$
2. If $F \cup e$ is a forest, $F \coloneqq F \cup e$
3. Remove e from E

Kruskal's Algorithm

Forest:
Tree that *might* not be connected. A tree is a forest.

Input edge list E
Forest $F = (V, \emptyset)$
While($\boldsymbol{F}$ is not a single connected component)

1. Find the edge $e \in E$ with smallest $w(e)$
2. If $F \cup e$ is a forest, $F \coloneqq F \cup e$
3. Remove e from E

Kruskal's Algorithm

Lemma:
Kruskal's algorithm returns a minimum weight spanning tree.

Lemma:
The runtime of Kruskal's algorithm is $O(|E| \log |E|)$

Optimal!

Kruskal’s Algorithm

Lemma:
The runtime of Kruskal’s algorithm is $O(|E| \log |E|)$

$\downarrow$ Denote #edges by m

Lemma:
The runtime of Kruskal’s algorithm is $O(m \log m)$

Kruskal's Algorithm

Lemma:
The runtime of Kruskal's algorithm is $O(m \log m)$

Input edge list E
Forest $F = (V, \emptyset)$
While($\boldsymbol{F}$ is not a single connected component)
1. Find the edge $e \in E$ with smallest $w(e)$
2. If $F \cup e$ is a forest, $F \coloneqq F \cup e$
3. Remove e from E

Kruskal's Algorithm

Lemma:
The runtime of Kruskal's algorithm is $O(m \log m)$

Input edge list E
Forest $F = (V, \emptyset)$
While(F is not a single connected component)
1. Find the edge $e \in E$ with smallest $w(e)$
2. If $F \cup e$ is a forest, $F := F \cup e$
3. Remove e from E

Naïve is bad:
- While loop $\Omega(m)$ times
- Checking the "while" – condition takes $\Omega(n)$ time.
- Search takes $\Omega(m)$ time
- Forest check takes $\Omega(n)$ time
- Leads to $\Omega(m^2)$

Kruskal's Algorithm

Lemma:
The runtime of Kruskal's algorithm is $O(m \log m)$

Naïve is bad:
- While loop $\Omega(m)$ times
- Checking the "while" – condition takes $\Omega(n)$ time.
- Search takes $\Omega(m)$ time
- Forest check takes $\Omega(n)$ time
- Leads to $\Omega(m^2)$

Input edge list E
Forest $F = (V, \emptyset)$
While(F is not a single connected component)
1. Find the edge $e \in E$ with smallest $w(e)$
2. If $F \cup e$ is a forest, $F := F \cup e$
3. Remove e from E

Kruskal's Algorithm

Lemma:
The runtime of Kruskal's algorithm is $O(m \log m)$

Naïve is bad:

- While loop $\Omega(m)$ times
- Checking the "while" – condition takes $\Omega(n)$ time.
- Search takes $\Omega(m)$ time
- Forest check takes $\Omega(n)$ time
- Leads to $\Omega(m^2)$

Input edge list E
Forest $F = (V, \emptyset)$
While(F is not a single connected component)

1. Find the edge $e \in E$ with smallest $w(e)$
2. If $F \cup e$ is a forest, $F := F \cup e$
3. Remove e from E

Kruskal’s Algorithm

Smarter implementation:
Use Union-Find data structure
(e.g., with UNION by size).

FIND(v): Returns (an identifier of) the component of v in time $O(\log n)$.

UNION(u, v): Merges the components of u and v into a single component in time $O(\log n)$.

Kruskal's Algorithm

Smarter implementation:
Use Union-Find data structure
(e.g., with UNION by size).

FIND(v): Returns (an identifier of) the component of v in time $O(\log n)$.

UNION(u, v): Merges the components of u and v into a single component in time $O(\log n)$.

Every node of the merged component agree on the identifier of the component

Kruskal’s Algorithm

Smarter implementation:
Use Union-Find data structure
(e.g., with UNION by size).

FIND(v): Returns (an identifier of) the component of v in time $O(\log n)$.

UNION(u, v): Merges the components of u and v into a single component in time $O(\log n)$.

Input edge list E
Sort E ascending according to weight
Forest $F = (V, \emptyset)$
Iterate in ascending order $\{u, v\} \in E$
    If(FIND(u) $\neq$ FIND(v))
        $F \coloneqq F \cup \{u, v\}$
        UNION(FIND(u), FIND(v))

Kruskal's Algorithm

Smarter implementation:
Use Union-Find data structure
(e.g., with UNION by size).

FIND(v): Returns (an identifier of) the component of v in time $O(\log n)$.

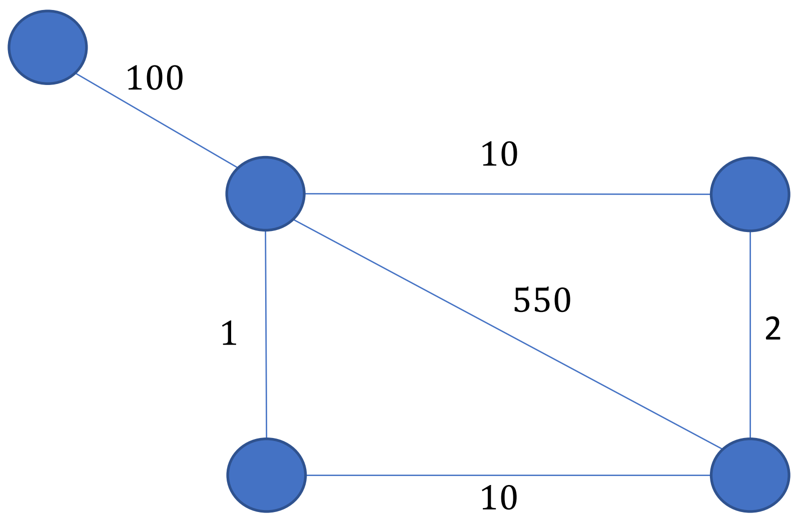
UNION(u, v): Merges the components of u and v into a single component in time $O(\log n)$.

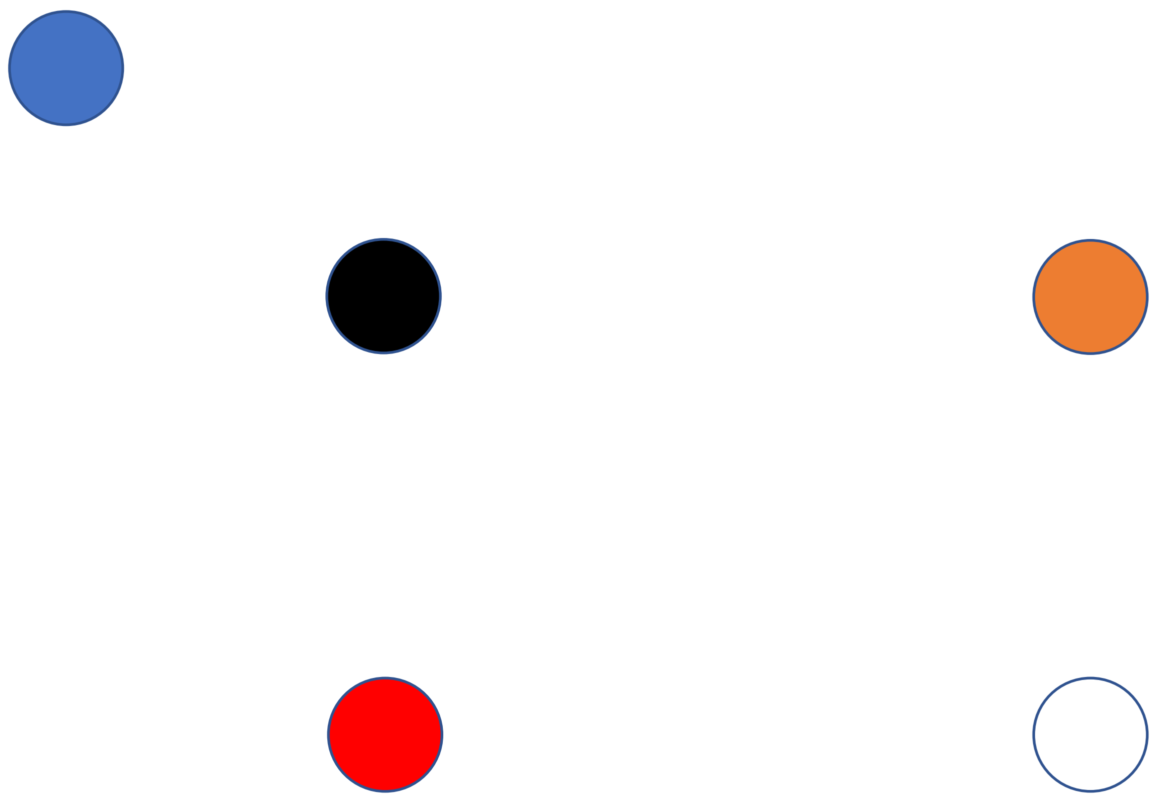
Input edge list E
Sort E ascending according to weight
Forest $F = (V, \emptyset)$
Iterate in ascending order $\{u, v\} \in E$
    If(FIND(u) $\neq$ FIND(v))
        $F \coloneqq F \cup \{u, v\}$
        UNION(FIND(u), FIND(v))

Runtime:
Sorting takes $\mathrm{O}(m \log m)$
For-loop requires $O(m \log n)$

Kruskal's Algorithm - Example

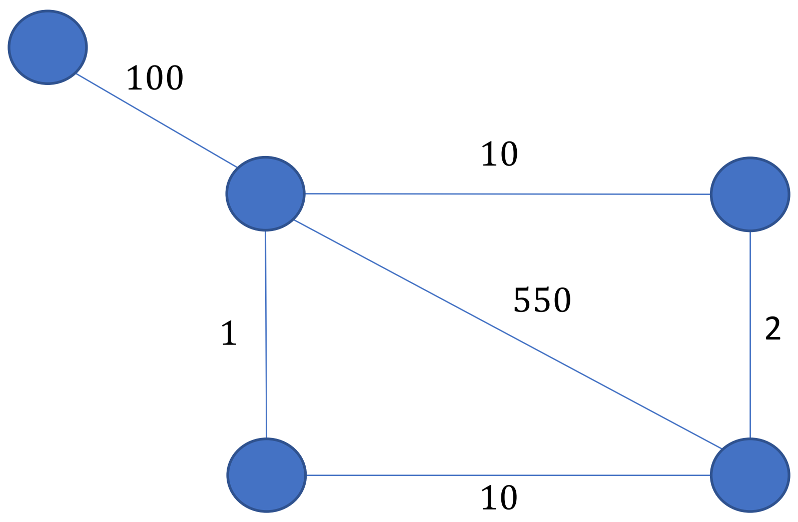
Input edge list E
Sort E ascending according to weight
Forest $F = (V, \emptyset)$
Iterate in ascending order $\{u, v\} \in E$
    If(FIND(u) $\neq$ FIND(v))
        $F := F \cup \{u, v\}$
        UNION(FIND(u), FIND(v))

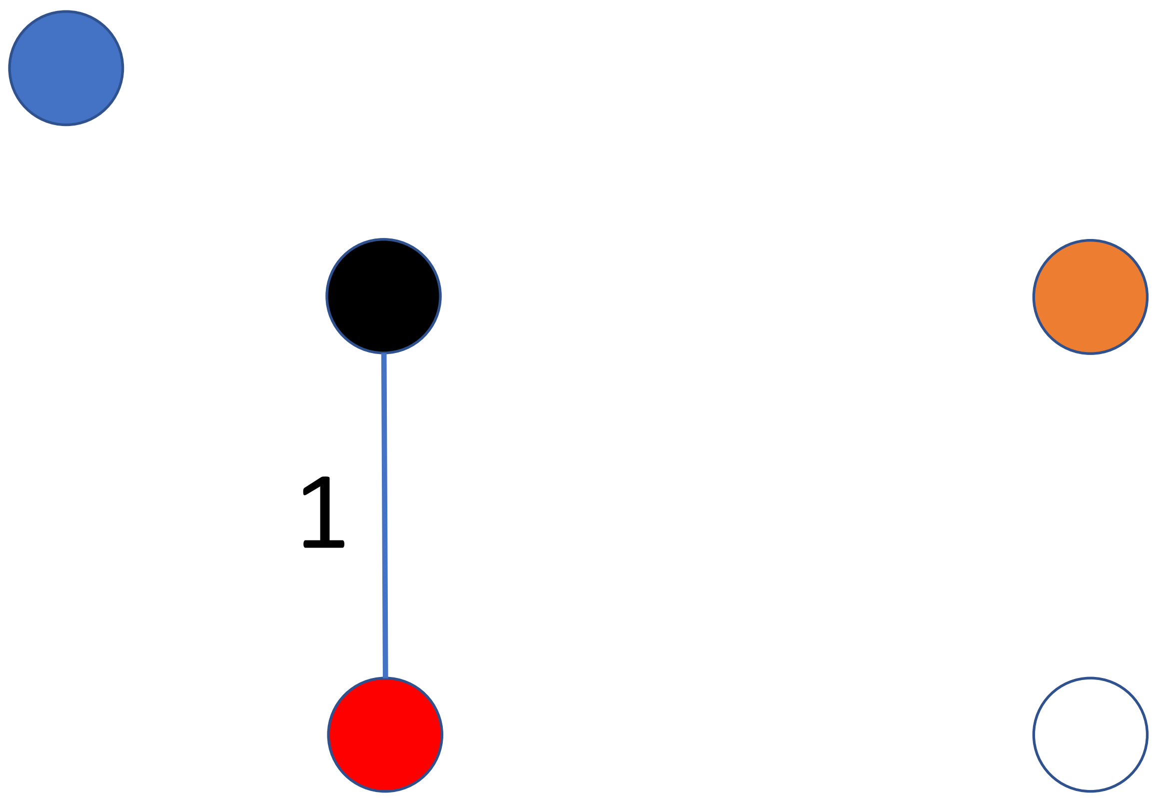




Kruskal's Algorithm - Example

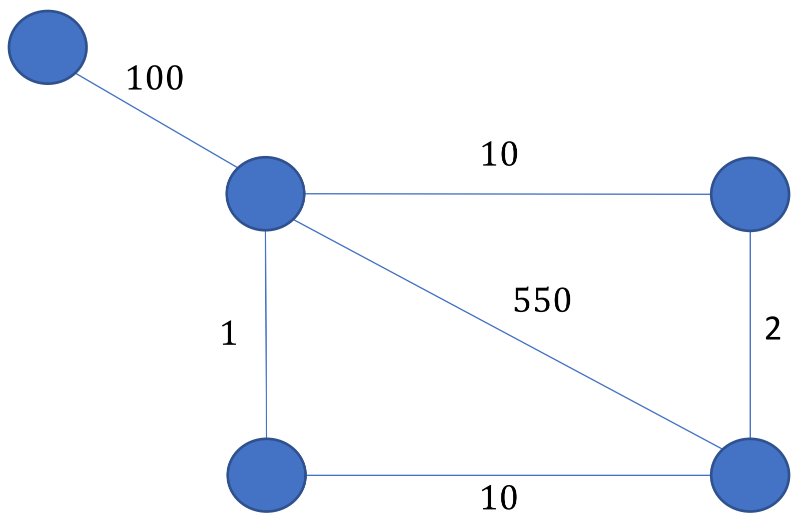
Input edge list E
Sort E ascending according to weight
Forest $F = (V, \emptyset)$
Iterate in ascending order $\{u, v\} \in E$
    If(FIND(u) $\neq$ FIND(v))
        $F := F \cup \{u, v\}$
        UNION(FIND(u), FIND(v))

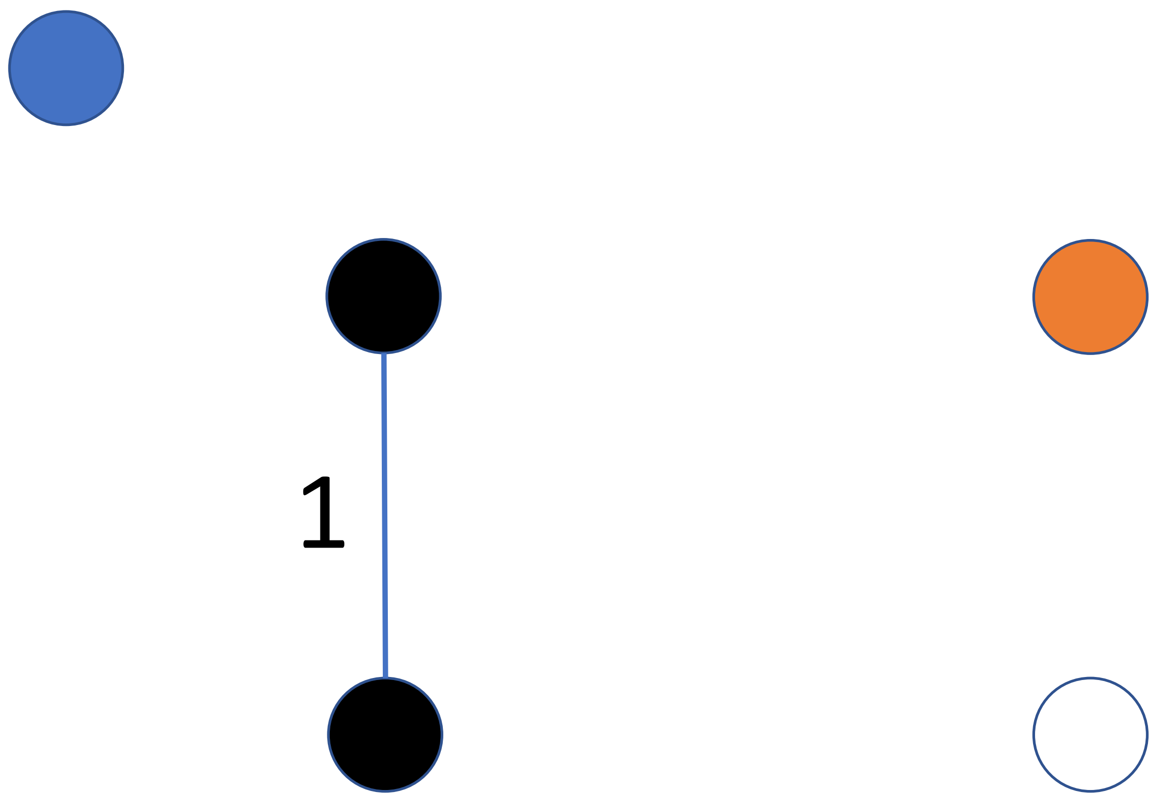




Kruskal's Algorithm - Example

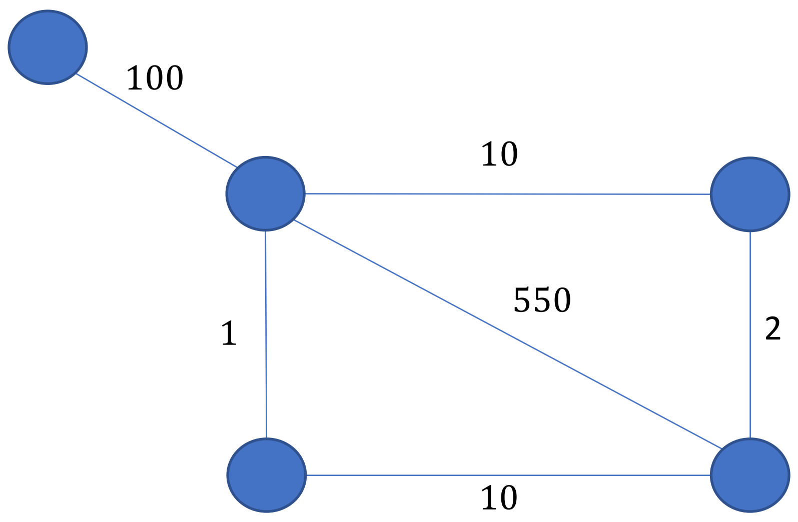
Input edge list E
Sort E ascending according to weight
Forest $F = (V, \emptyset)$
Iterate in ascending order $\{u, v\} \in E$
    If(FIND(u) $\neq$ FIND(v))
        $F \coloneqq F \cup \{u, v\}$
        UNION(FIND(u), FIND(v))

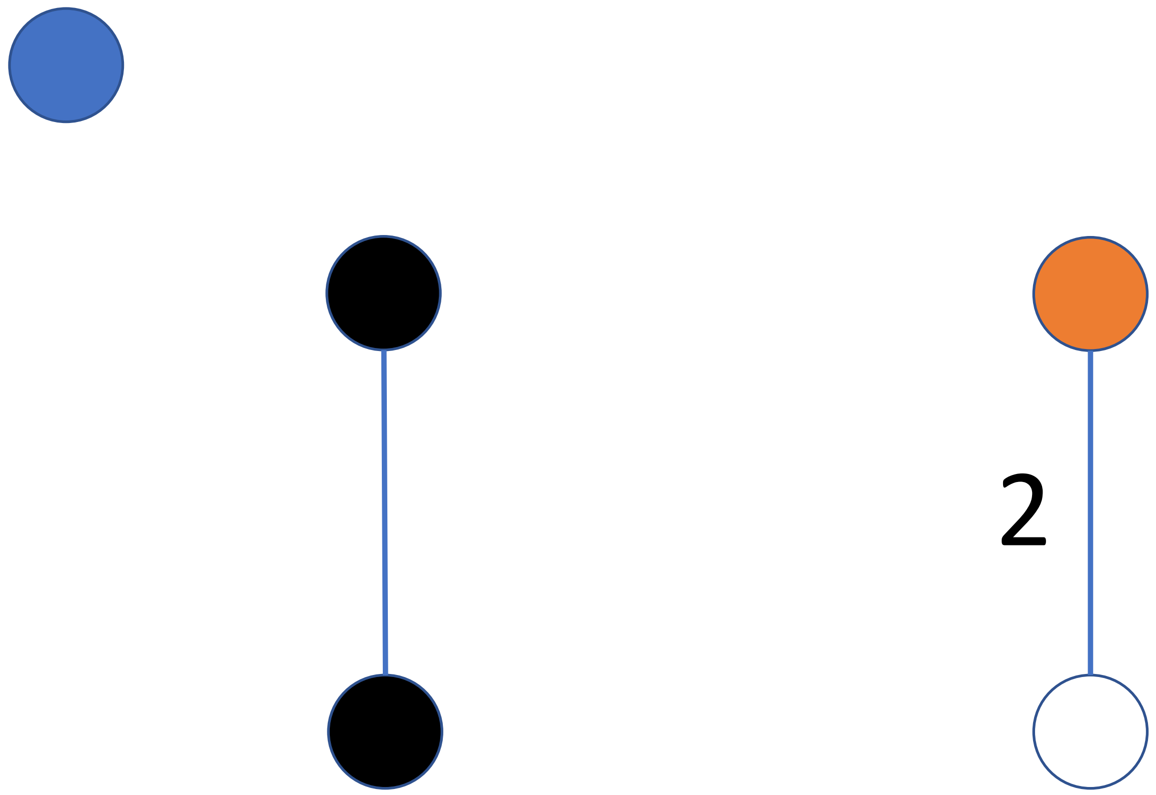




Kruskal's Algorithm - Example

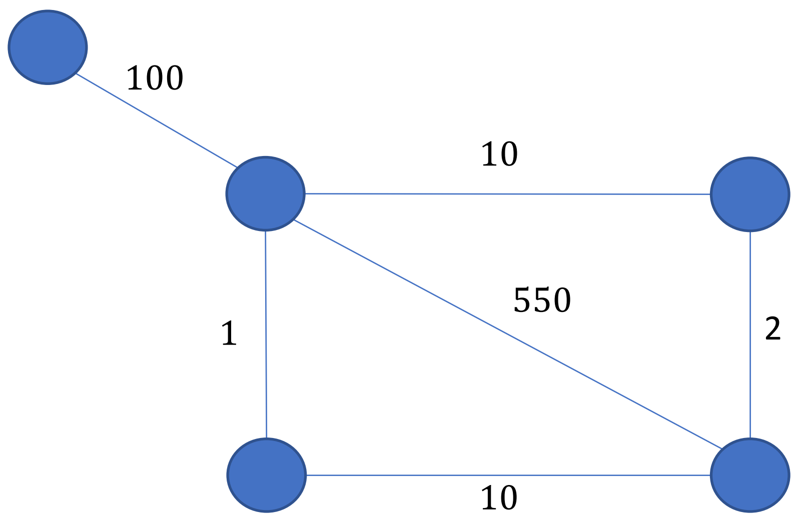
Input edge list E
Sort E ascending according to weight
Forest $F = (V, \emptyset)$
Iterate in ascending order $\{u, v\} \in E$
    If(FIND(u) $\neq$ FIND(v))
        $F := F \cup \{u, v\}$
        UNION(FIND(u), FIND(v))

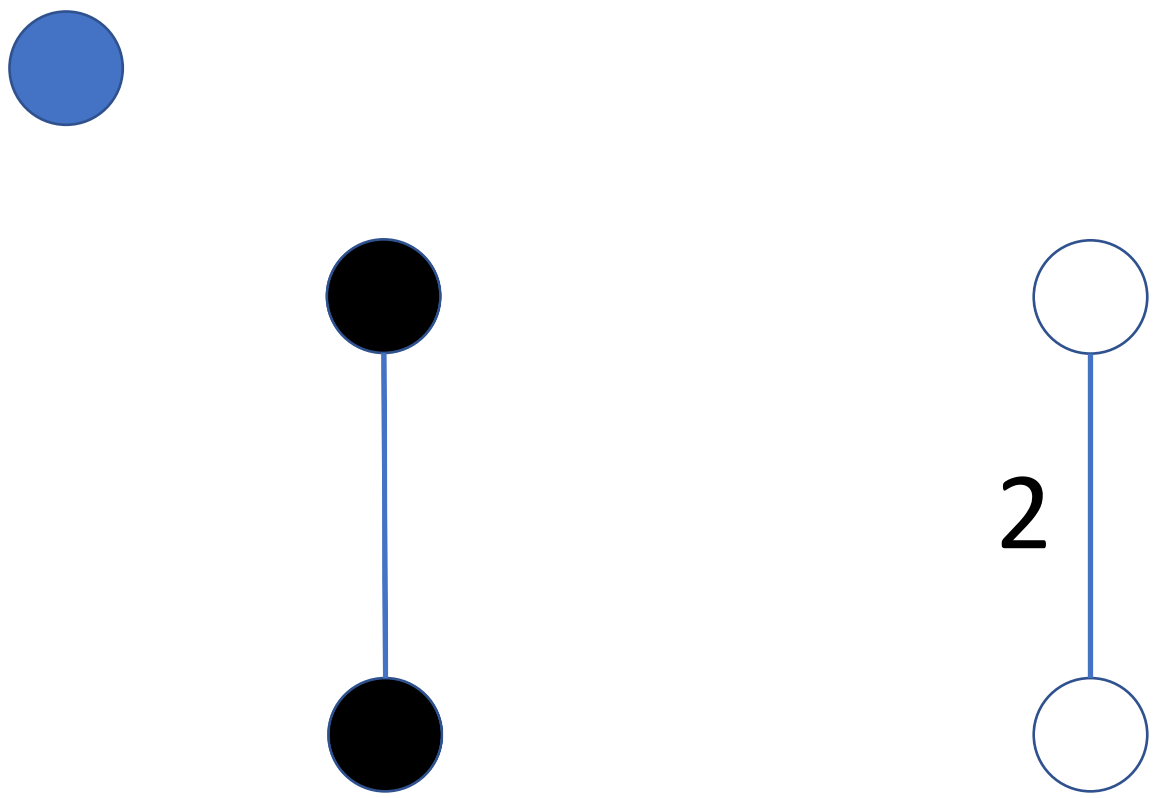




Kruskal's Algorithm - Example

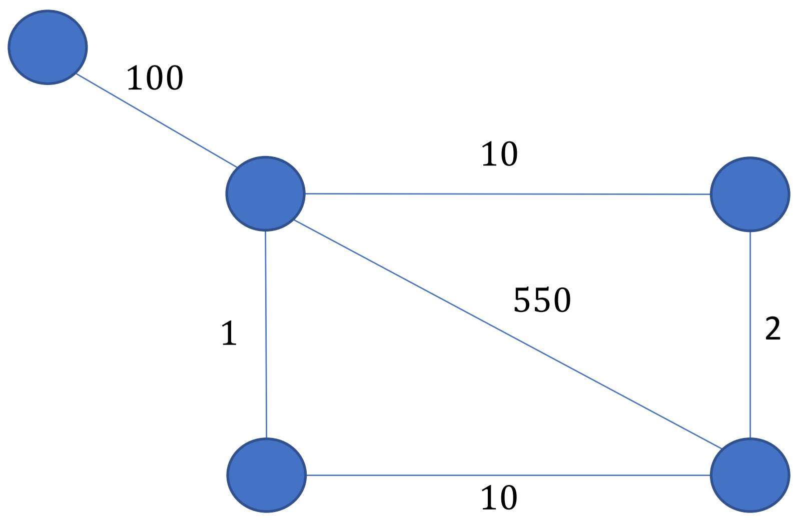
Input edge list E
Sort E ascending according to weight
Forest $F = (V, \emptyset)$
Iterate in ascending order $\{u, v\} \in E$
    If(FIND(u) $\neq$ FIND(v))
        $F \coloneqq F \cup \{u, v\}$
        UNION(FIND(u), FIND(v))

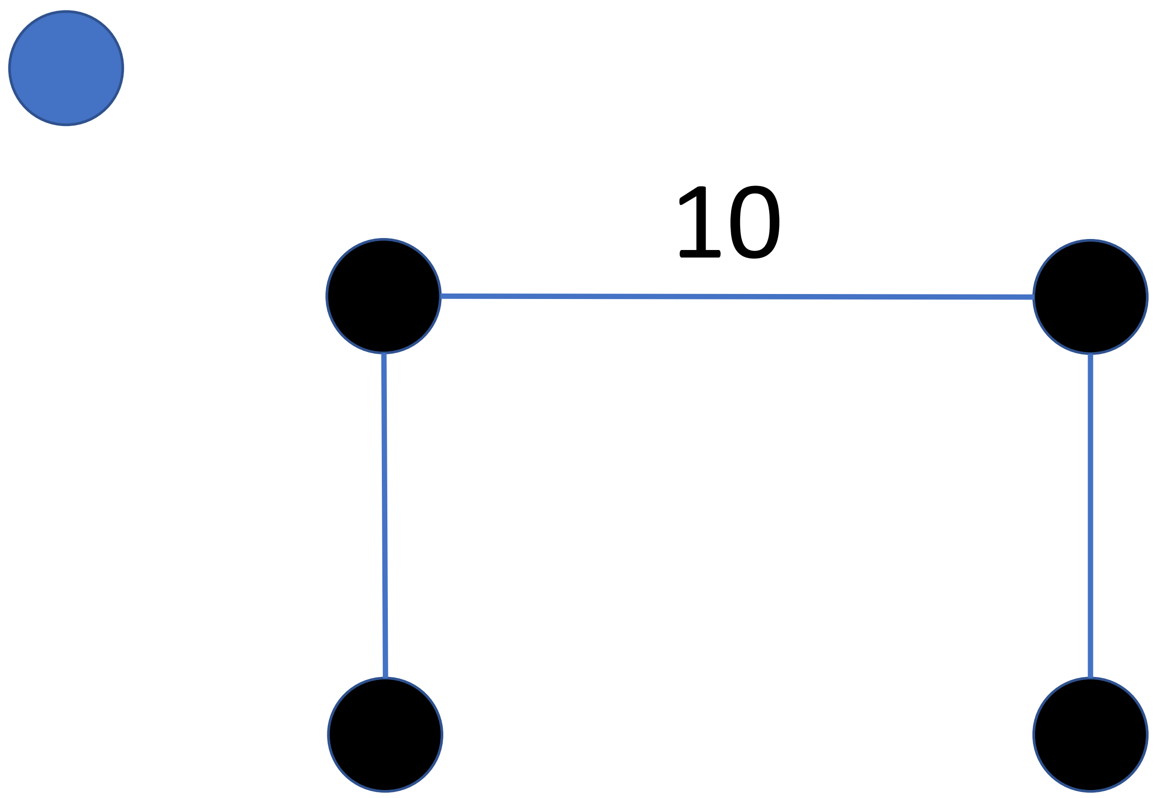




Kruskal's Algorithm - Example

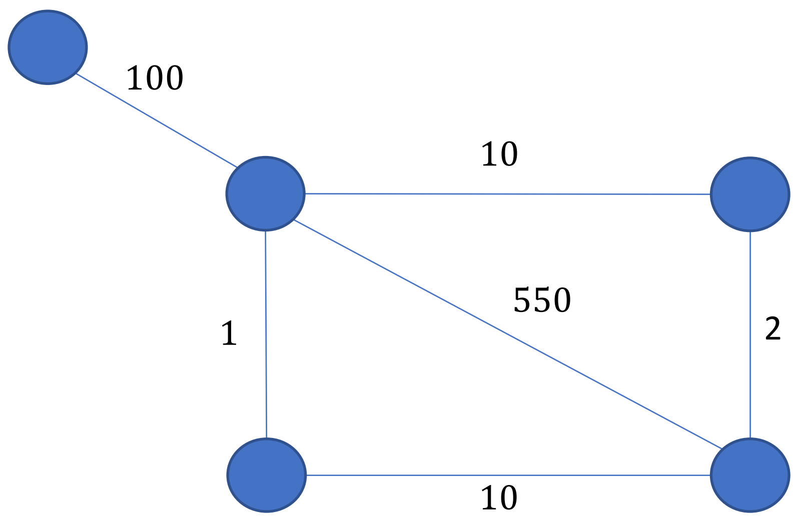
Input edge list E
Sort E ascending according to weight
Forest $F = (V, \emptyset)$
Iterate in ascending order $\{u, v\} \in E$
    If(FIND(u) $\neq$ FIND(v))
        $F := F \cup \{u, v\}$
        UNION(FIND(u), FIND(v))

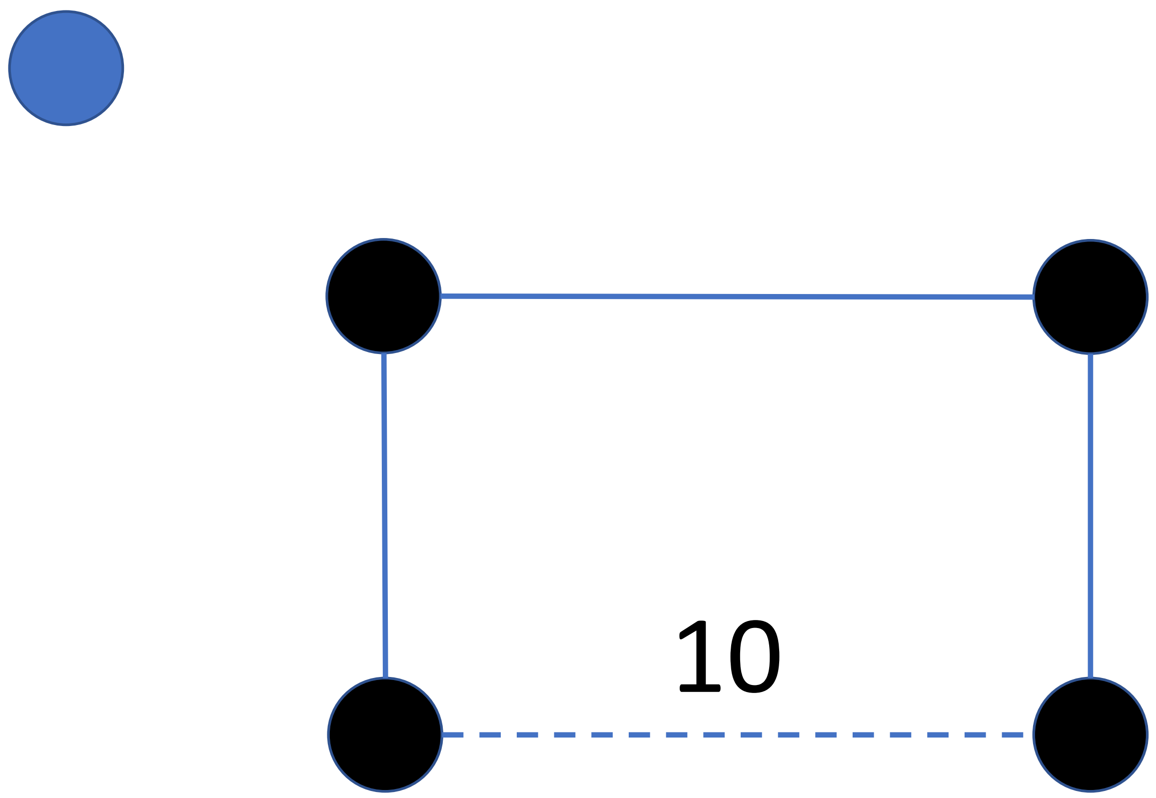




Kruskal's Algorithm - Example

Input edge list E
Sort E ascending according to weight
Forest $F = (V, \emptyset)$
Iterate in ascending order $\{u, v\} \in E$
    If(FIND(u) $\neq$ FIND(v))
        $F := F \cup \{u, v\}$
        UNION(FIND(u), FIND(v))

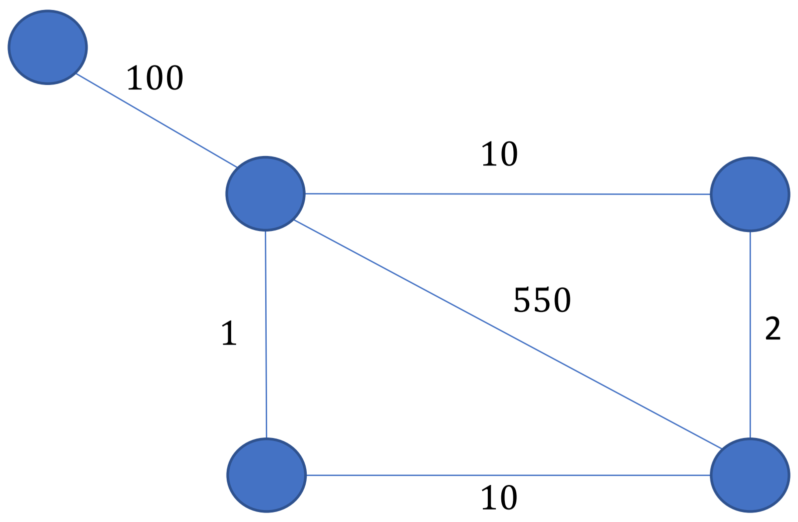


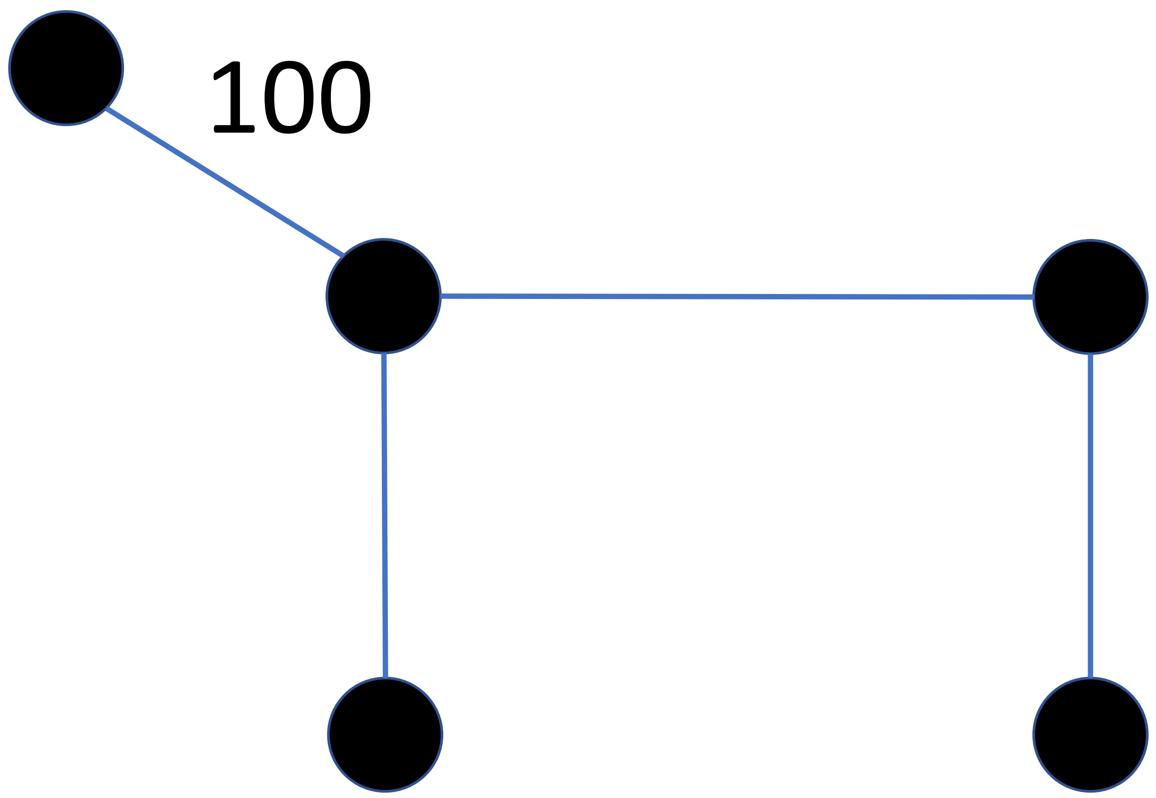


Don't connect two nodes of same color

Kruskal's Algorithm - Example

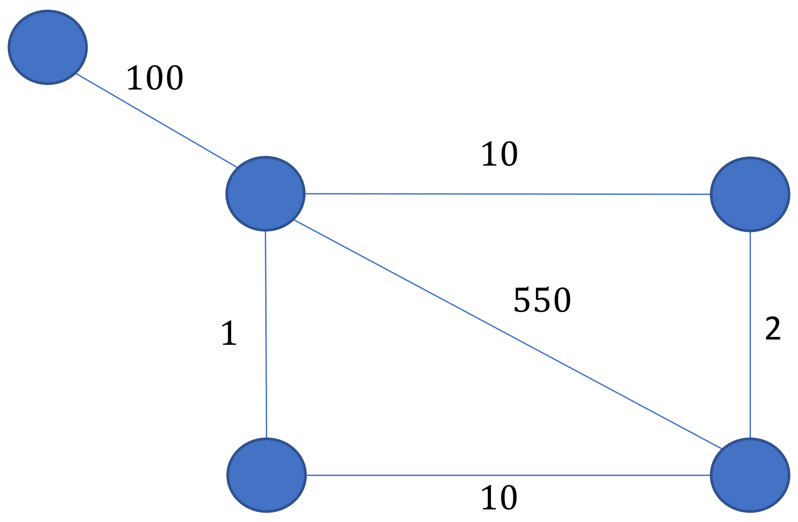
Input edge list E
Sort E ascending according to weight
Forest $F = (V, \emptyset)$
Iterate in ascending order $\{u, v\} \in E$
    If(FIND(u) $\neq$ FIND(v))
        $F := F \cup \{u, v\}$
        UNION(FIND(u), FIND(v))

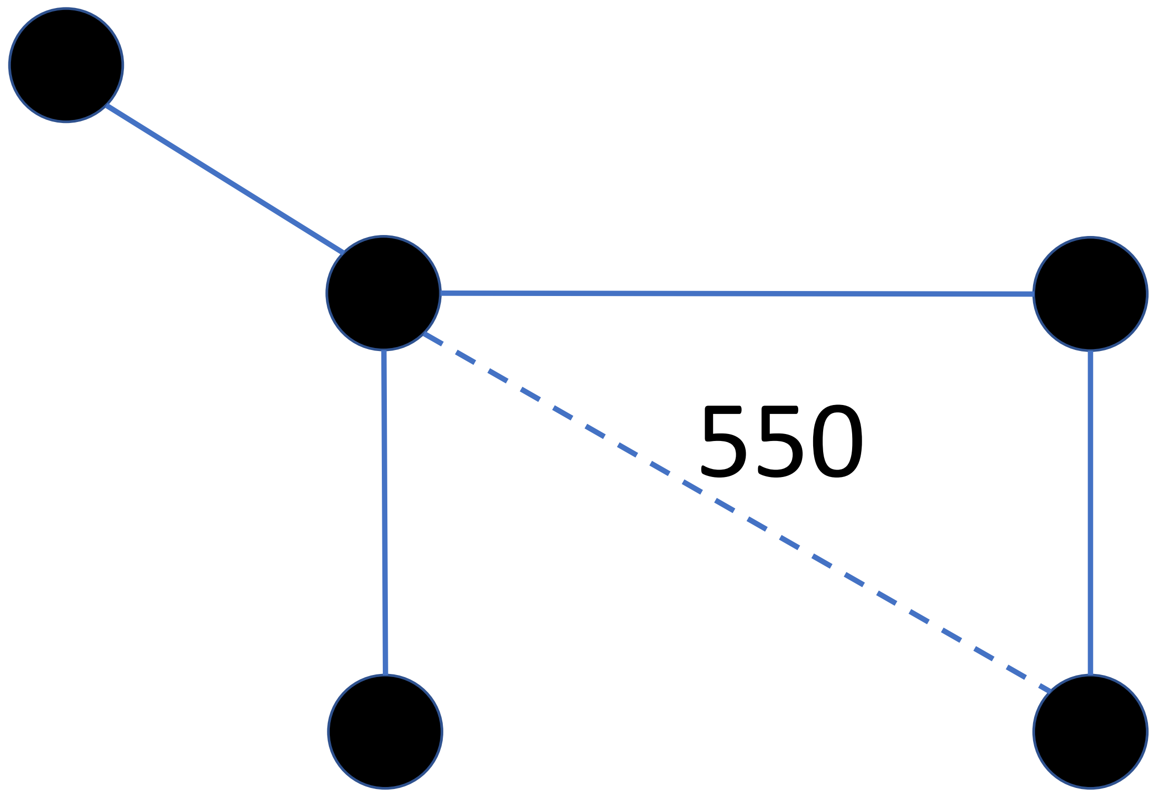




Kruskal's Algorithm - Example

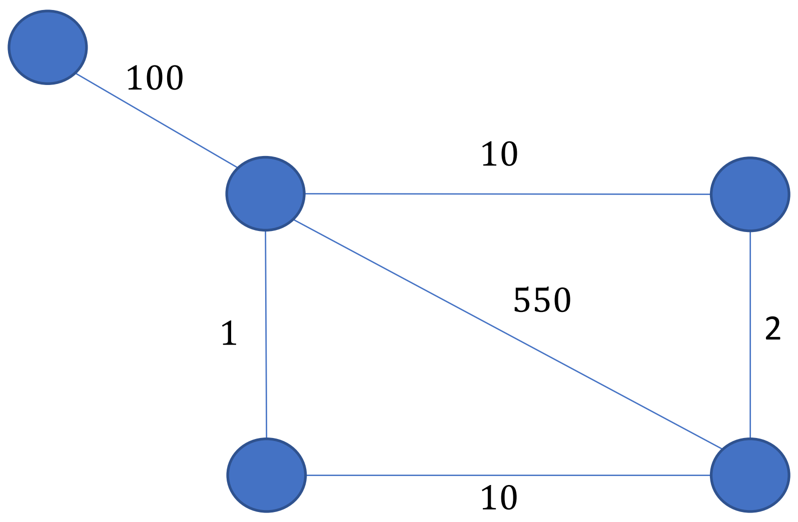
Input edge list E
Sort E ascending according to weight
Forest $F = (V, \emptyset)$
Iterate in ascending order $\{u, v\} \in E$
    If(FIND(u) $\neq$ FIND(v))
        $F \coloneqq F \cup \{u, v\}$
        UNION(FIND(u), FIND(v))

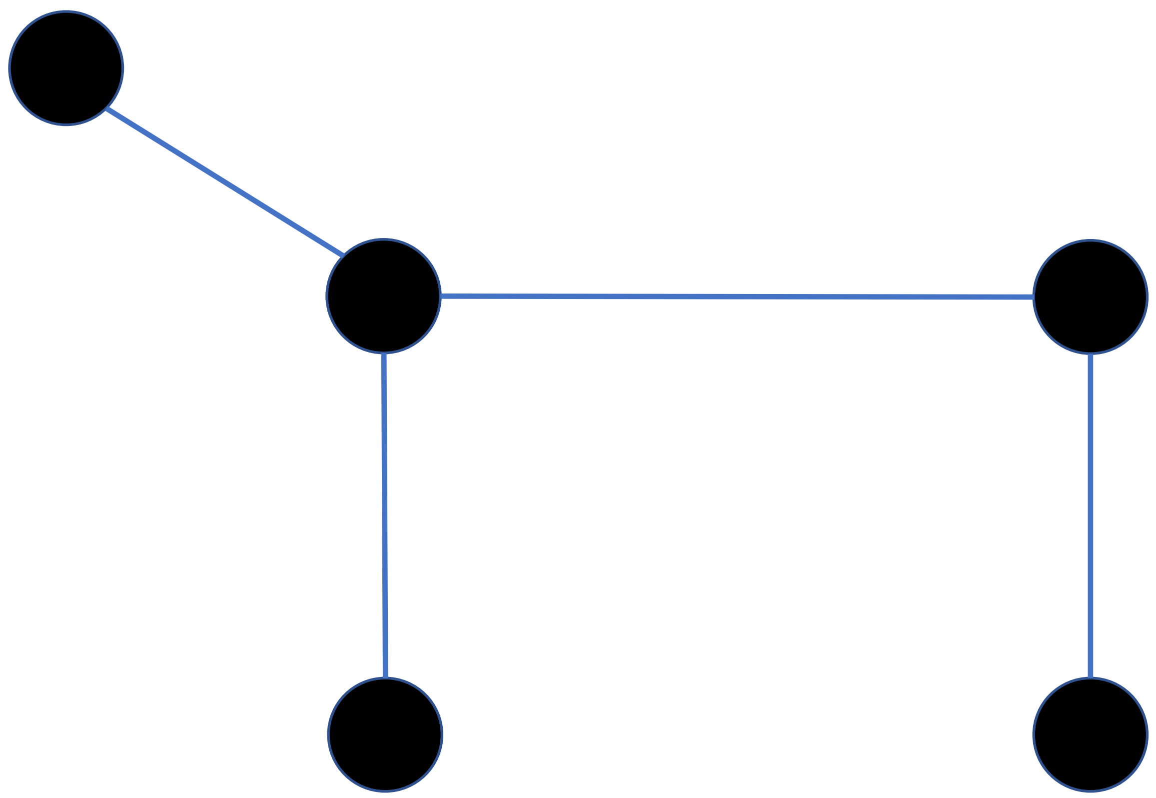




Kruskal's Algorithm - Example

Input edge list E
Sort E ascending according to weight
Forest $F = (V, \emptyset)$
Iterate in ascending order $\{u, v\} \in E$
    If(FIND(u) $\neq$ FIND(v))
        $F := F \cup \{u, v\}$
        UNION(FIND(u), FIND(v))





Idea Recap

Minimum Spanning Tree:
Start with isolated nodes as components.

Greedily merge components.

Union-Find:
Check if nodes in the same component and perform merging in $O(\log m)$ time

Union-Find Exercise:
Details of Union-Find in the tutorial exercise

Challenge (?): Can you implement it without checking the literature?

Outline

- Minimum Spanning Tree (MST)
  - Optimization
- The Greedy Approach
  - Marginal Gain/Loss
  - Correctness and runtime
  - Optimal for MST

Kruskal’s Algorithm

Lemma:
Kruskal’s algorithm returns a minimum weight spanning tree.

Kruskal's Algorithm

Lemma:
Kruskal's algorithm returns a minimum weight spanning tree.

Optimal!

Kruskal’s Algorithm

Lemma:
Kruskal’s algorithm returns a minimum weight spanning tree.

Simplification:
Suppose that the weights are unique

Kruskal's Algorithm

Suppose for a contradiction that Kruskal's does not give a minimum spanning tree.

$\rightarrow$

Let T be the spanning tree returned by Kruskal's. Order the edges according to when they are selected.

Kruskal's Algorithm

Suppose for a contradiction that Kruskal's does not give a minimum spanning tree.

Let T be the spanning tree returned by Kruskal's. Order the edges according to when they are selected.

Let T^* be the minimum weight spanning tree.

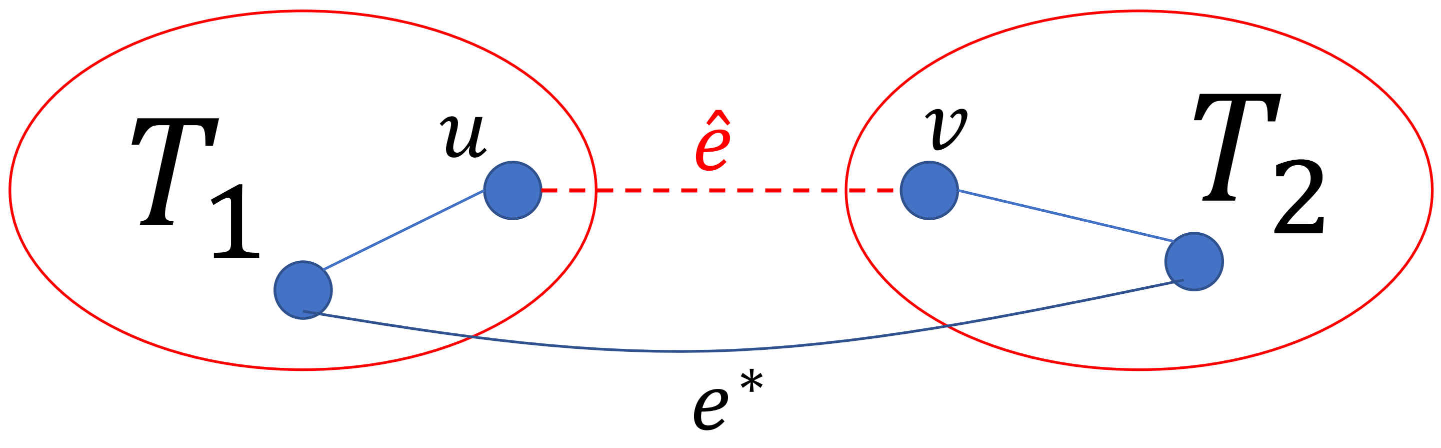
Consider the first edge $\hat{e}$ in T^* that differs from T.

Kruskal's Algorithm

Edge $\hat{e} \in T$ is the first edge not in T^*. Removing $\hat{e}$ from T divides T into two disjoint components T_1 and T_2.

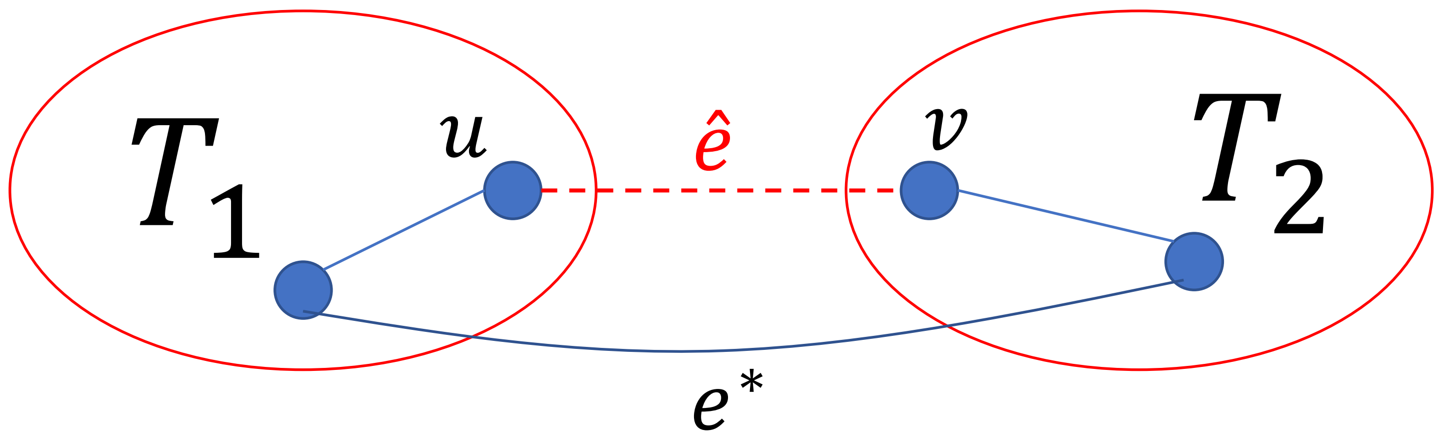
There must be a path in T^* that connects nodes u and v. On this path, there is at least one edge $e^* \in T^*$ with one endpoint in T_1 and the other in T_2.

Since Kruskal's did not pick e^*, it must be the case that $w(e^*) > w(\hat{e})$.



Kruskal's Algorithm

Edge $\hat{e} \in T$ is the first edge not in T^*. Removing $\hat{e}$ from T divides T into two disjoint components T_1 and T_2.



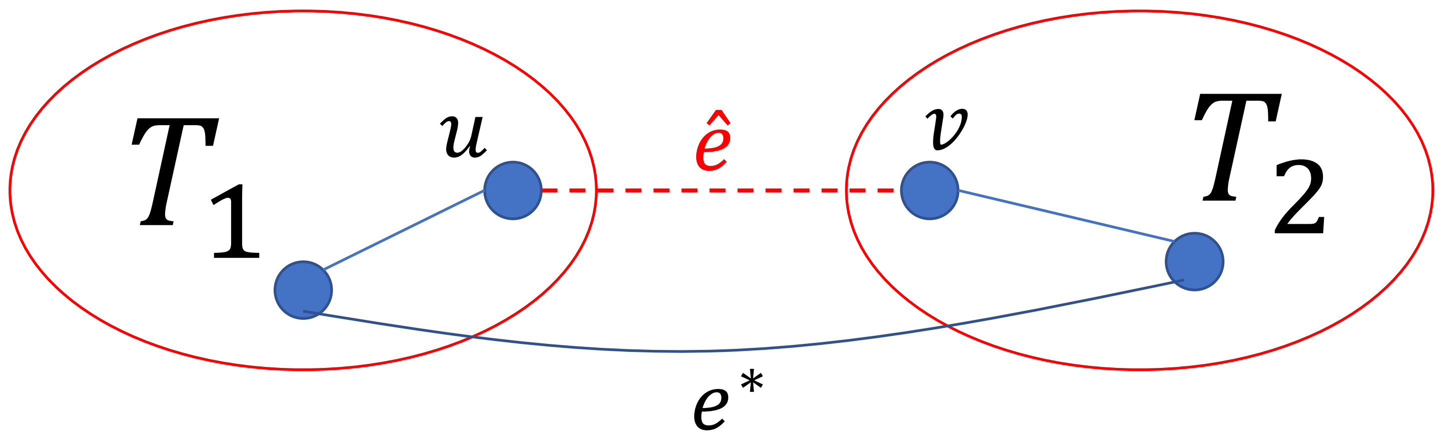
There must be a path in T^* that connects nodes u and v. On this path, there is at least one edge $e^* \in T^*$ with one endpoint in T_1 and the other in T_2.

Since Kruskal's did not pick e^*, it must be the case that $w(e^*) > w(\hat{e})$.

Removing e^* from T^* and adding $\hat{e}$ to T^* reduces its weight.

Kruskal's Algorithm

Edge $\hat{e} \in T$ is the first edge not in T^*. Removing $\hat{e}$ from T divides T into two disjoint components T_1 and T_2.



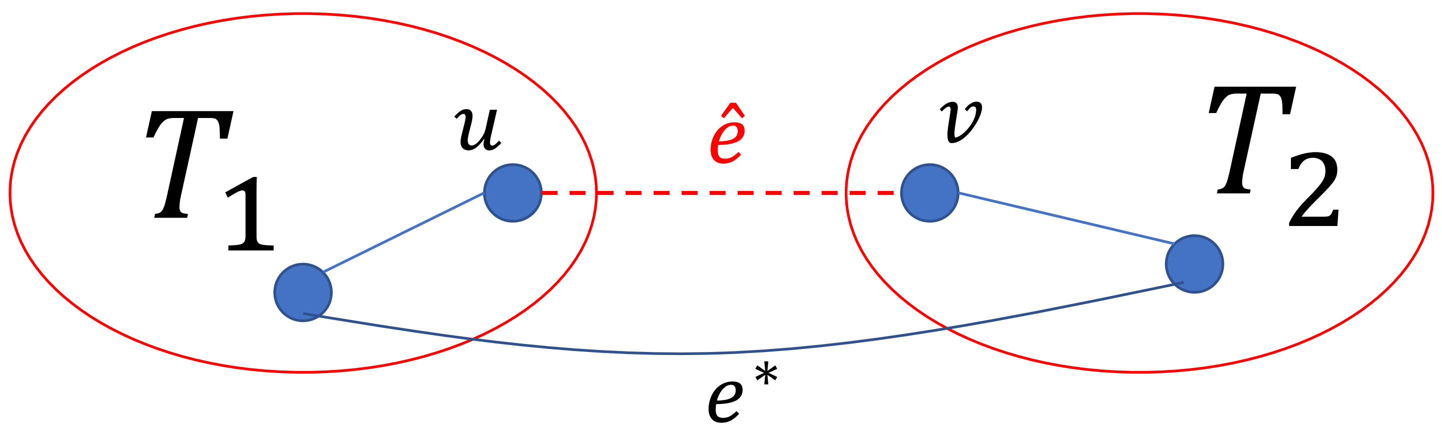
There must be a path in T^* that connects nodes u and v. On this path, there is at least one edge $e^* \in T^*$ with one endpoint in T_1 and the other in T_2.

Since Kruskal's did not pick e^*, it must be the case that $w(e^*) > w(\hat{e})$.

Removing e^* from T^* and adding $\hat{e}$ to T^* reduces its weight.

Contradiction.

Proof Recap



Consider first greedy edge $\hat{e} = \{u, v\}$ that is not in the optimal spanning tree T^*.

Without $\hat{e}$, nodes u and v are disconnected. In T^*, there is an edge e^* that connects the components of u and v.

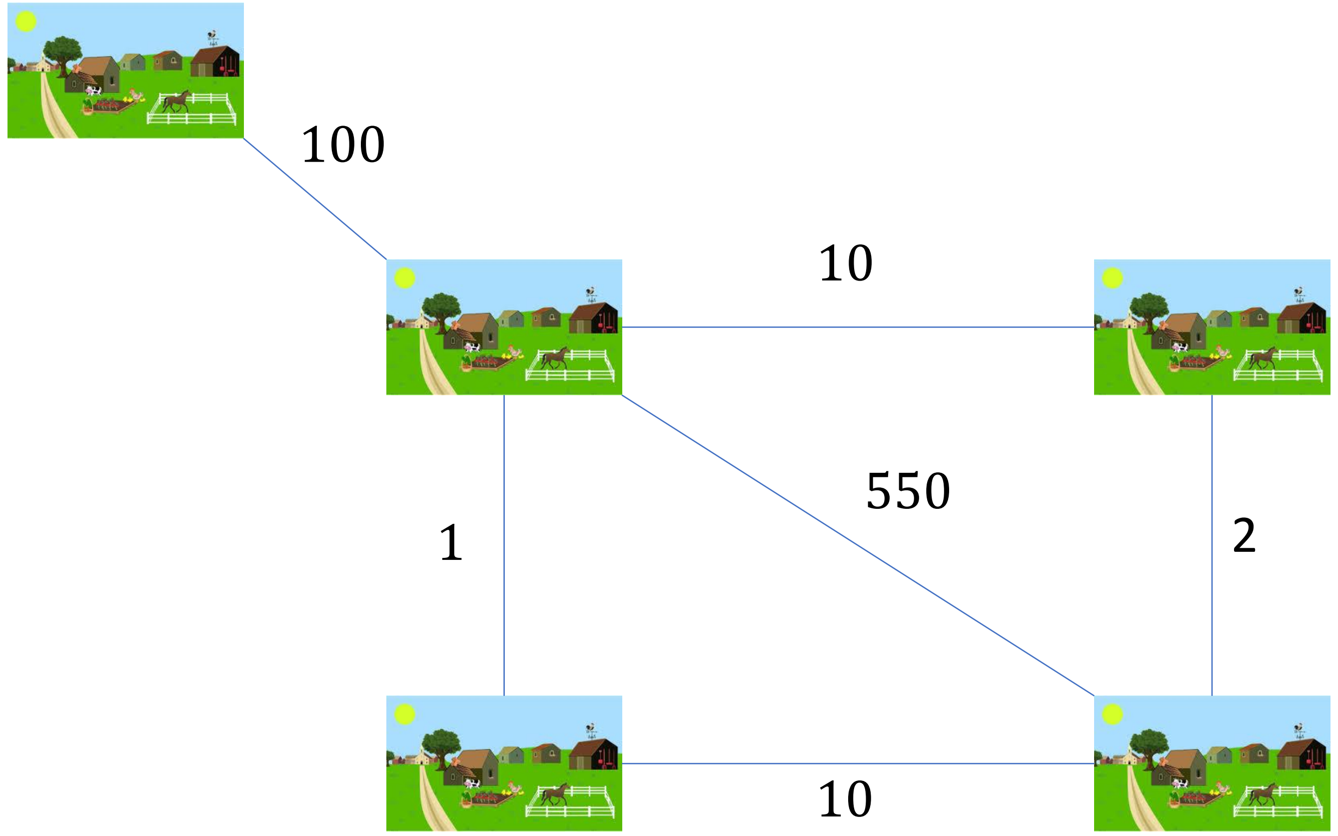
Swapping $\hat{e}$ and e^* creates a spanning tree that is cheaper than the optimum which is a contradiction.

Kruskal’s Algorithm

Lemma:
Kruskal’s algorithm returns a minimum weight spanning tree.

Lemma:
The runtime of Kruskal’s algorithm is $O(|E| \log |E|)$

Wrap-up



MST:
Cheapest connecting structure.

Kruskal's algorithm:
Greedy

Correctness

Runtime:
$O(m \log m)$