# CS 341: Algorithms

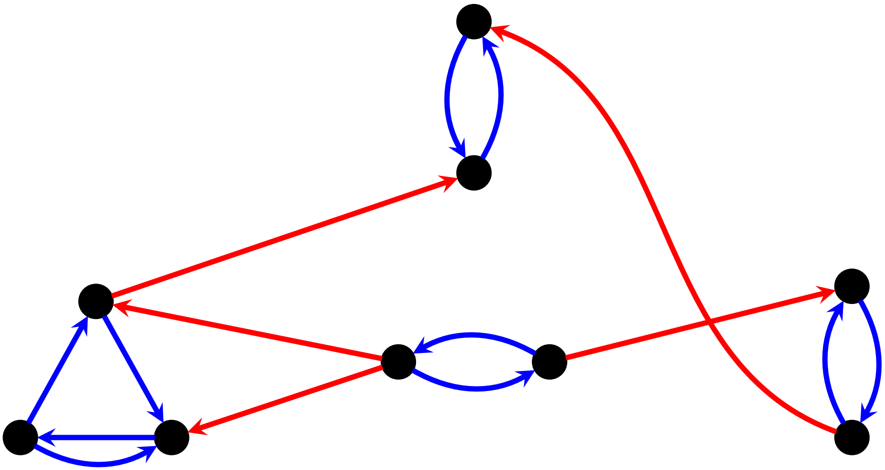
## Lecture 13: Minimum spanning trees

**Slides due to Éric Schost and based on lecture notes by many other CS341 instructors**

**David R. Cheriton School of Computer Science, University of Waterloo**

**Winter 2026**

# Kosaraju Example (SCC)

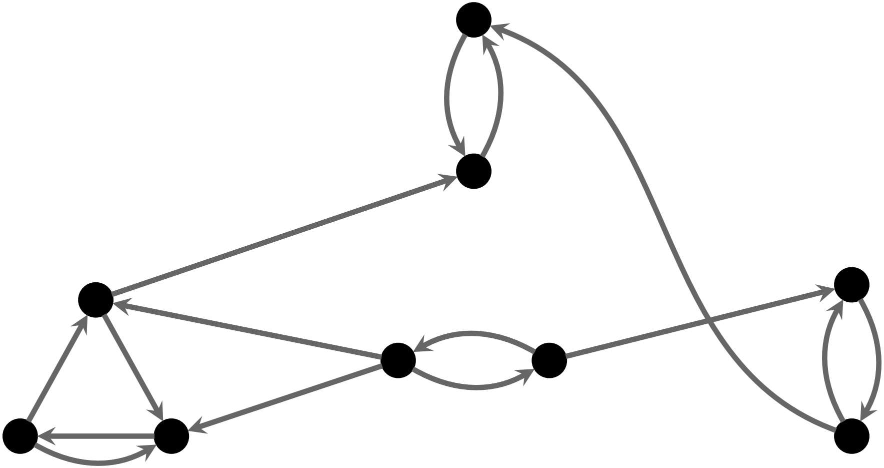


# Kosaraju's algorithm for strongly connected components

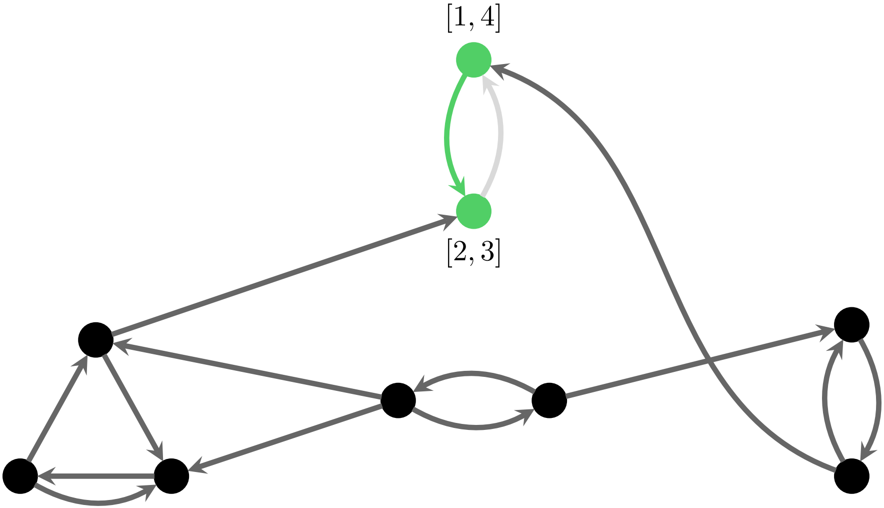
**SCC**$(G)$
1. run a DFS on $G$ and record finish times
2. run a DFS on $G^T$, with vertices ordered in **decreasing finish time**
3. return the trees in the DFS forest of $G^T$

**Runtime:** $\boldsymbol{O(n+m)}$ (don't forget the time to reverse $G$)

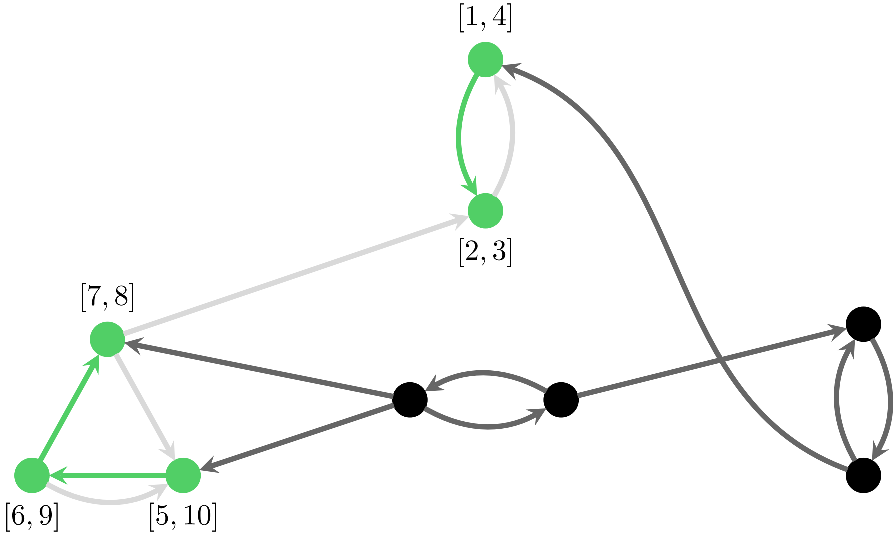
# Kosaraju Example (First DFS)



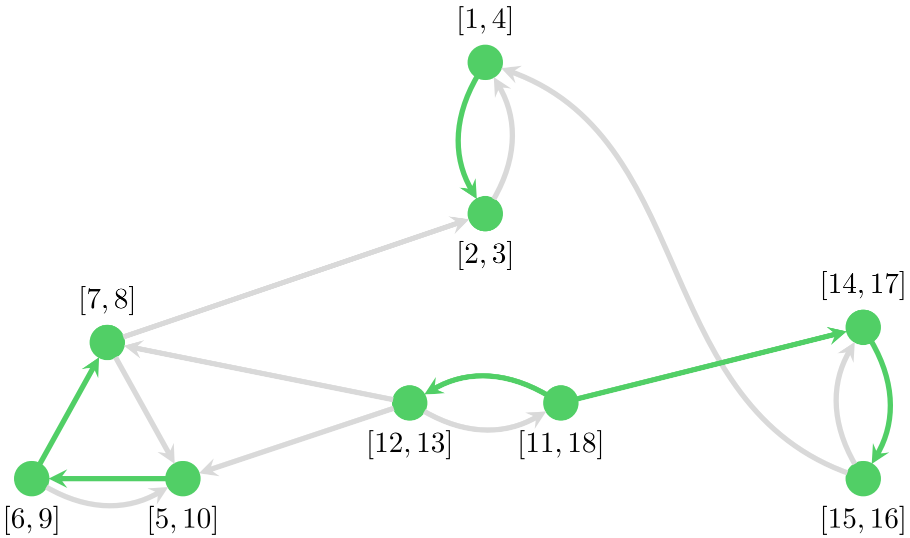
# Kosaraju Example (First DFS)



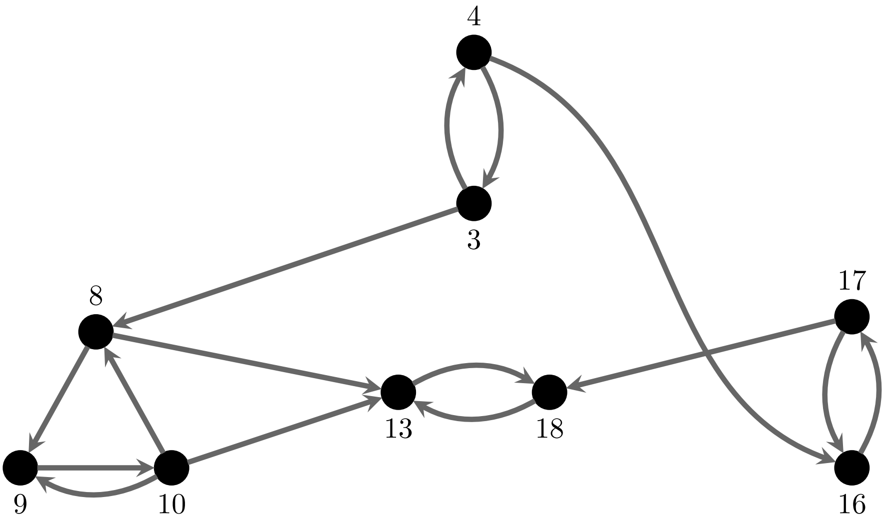
# Kosaraju Example (First DFS)



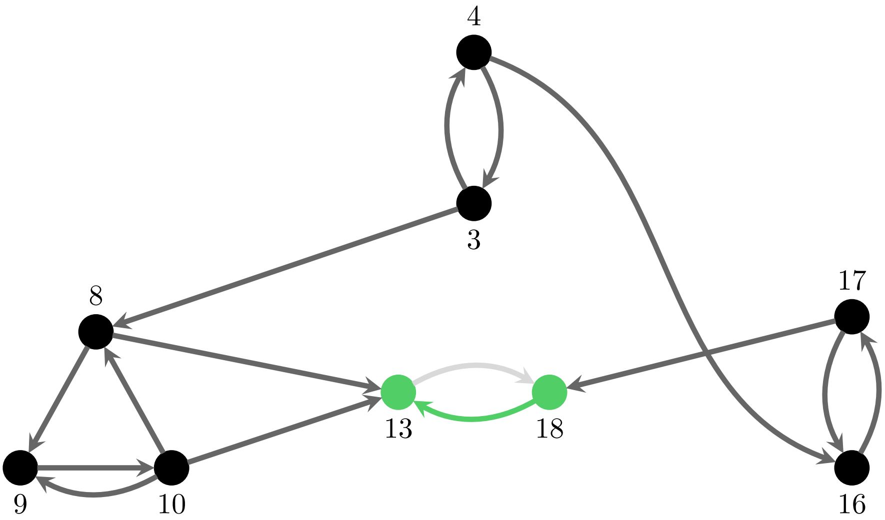
# Kosaraju Example (First DFS)



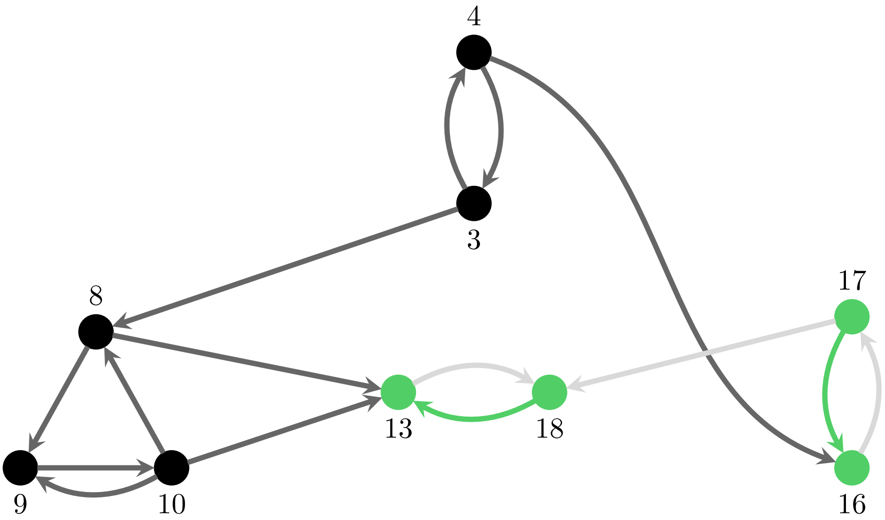
# Kosaraju Example (Second DFS)



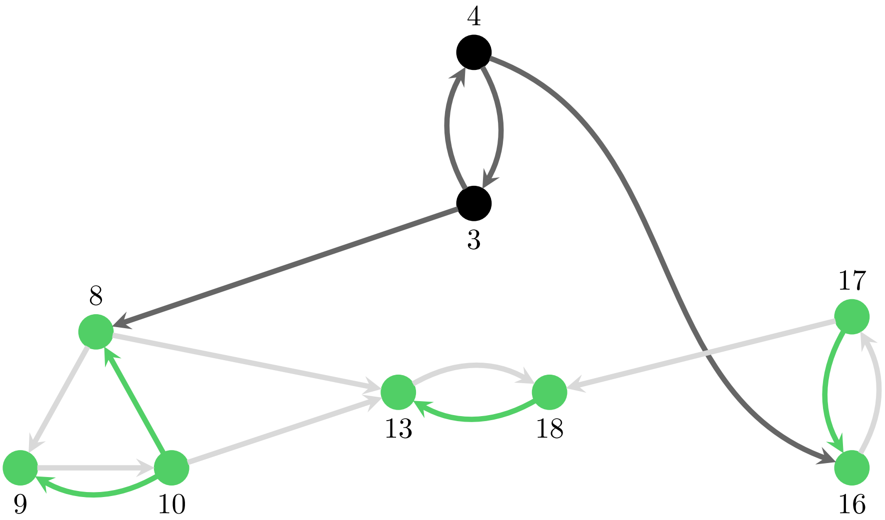
# Kosaraju Example (Second DFS)



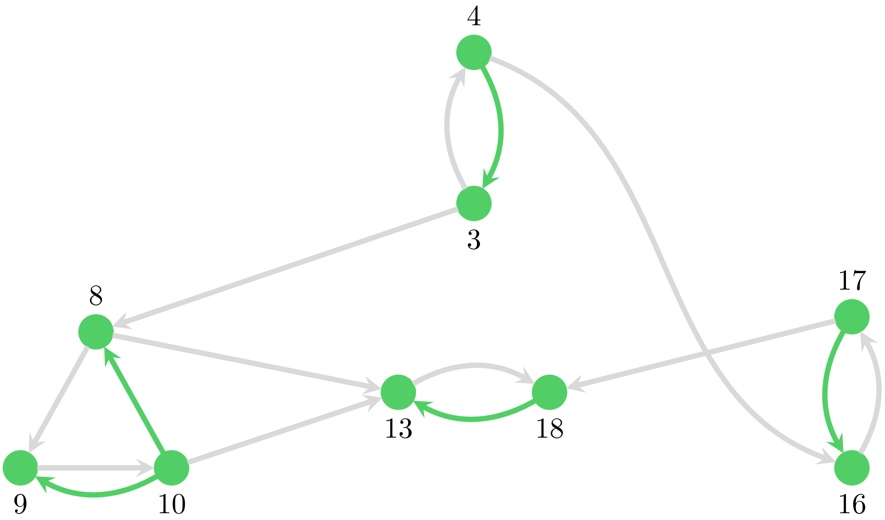
# Kosaraju Example (Second DFS)



# Kosaraju Example (Second DFS)



# Kosaraju Example (Second DFS)



# The idea behind the algorithm

**Claim**

If $S$ and $T$ are two distinct SCCs of $G$ and there is an edge $T \to S$,
**latest finish time in $S$ < latest finish time in $T$**

**Proof:**

- if we visit a vertex in $T$ first, all vertices in $S$ will be its descendants
- if we visit a vertex in $S$ first, we won't reach $T$ before $S$ is finished.

**Consequence:**

- start second run from the last-finished vertex $s$
- can prove: in $G^T$, every vertex reachable from $s$ is in the same SCC
- continue

# Spanning trees

**Input and output:**

- $G = (V, E)$ is a **weighted, connected undirected graph**
- edges have **weights** $w(e_i)$
- a **spanning tree** is a tree with edges from $E$ that covers all vertices
- examples: BFS tree, DFS tree

**Remark:** will assume $w(e_i)$ distinct, using $W(e_i) = [w(e_i), i]$ to break ties if needed

**Goal:**

- a spanning tree with **minimal weight**
- notation: $w(T) = \sum_{e \text{ edge in } T} w(e)$
- all weights fit in a word, unit cost model as usual

**Exercise**

what about maximal weight spanning trees?

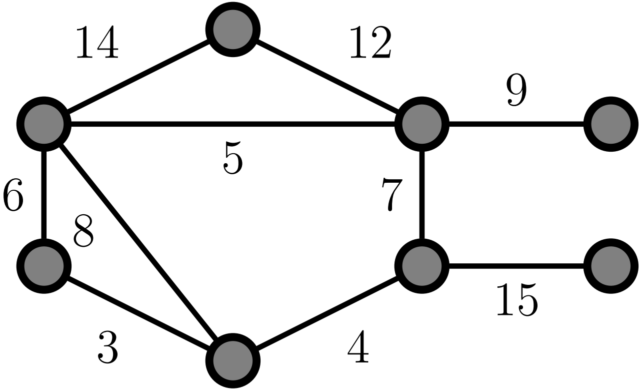
# Kruskal's algorithm

# Kruskal's algorithm

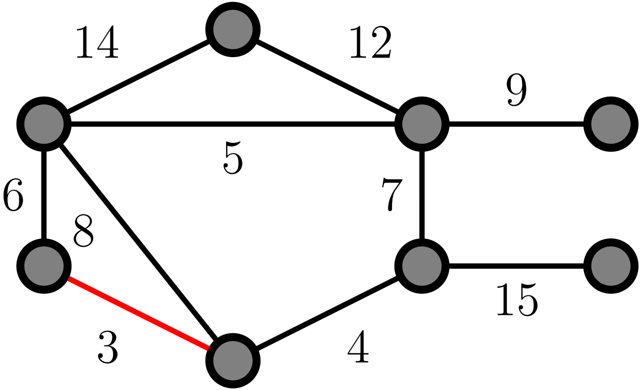
**GreedyMST**$(G)$

$F \leftarrow \{\ \}$ ($F$ is a set of edges)
sort edges by increasing weight
**for** $k = 1, \ldots, m$ **do**
    **if** $e_k$ does not create a cycle in $(V, F)$ **then**
        append $e_k$ to $F$
**return** $A = (V, F)$

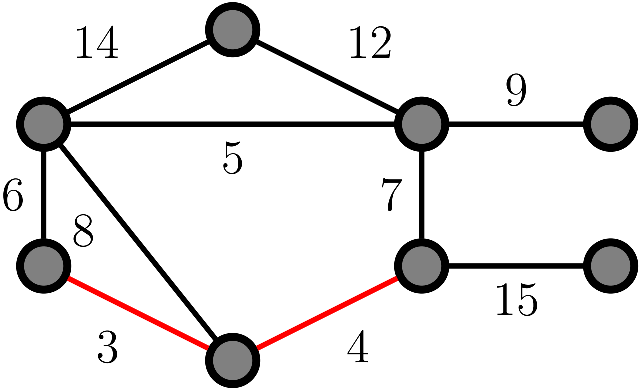
# Example



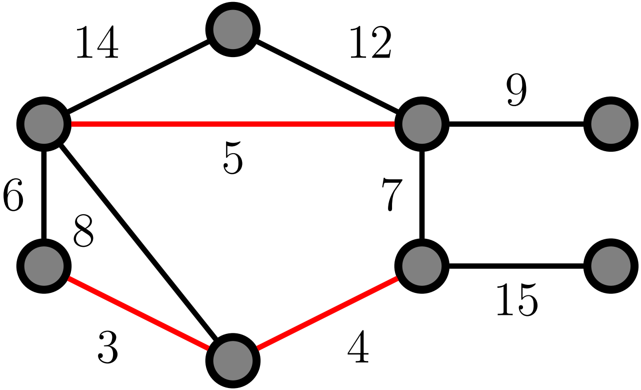
# Example



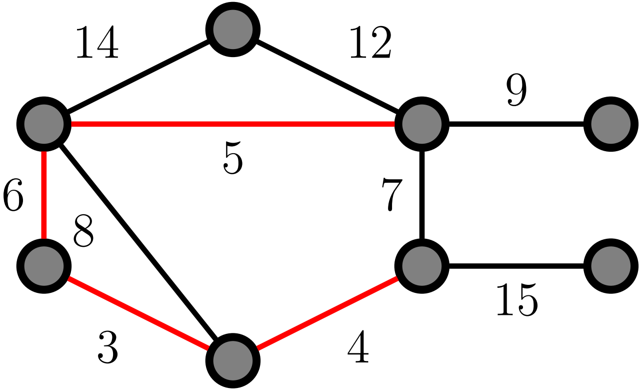
# Example



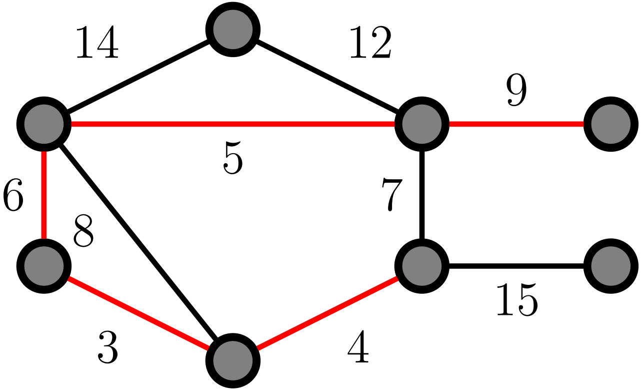
# Example



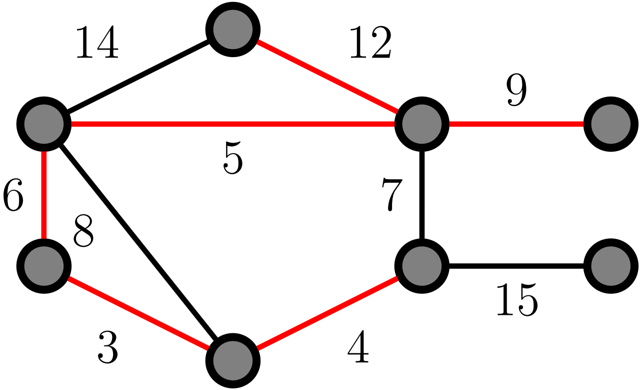
# Example



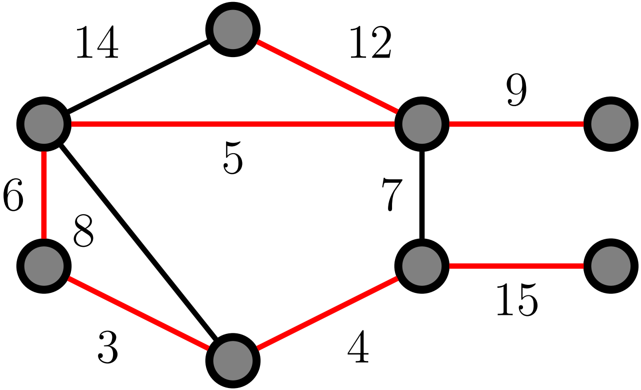
# Example



## Example



# Example



# Properties of the output

**Claim**

The output $\boldsymbol{A = (V, F)}$ **is a spanning tree**

**Proof:**

- of course, $A$ has no cycle: it is a **forest**
- suppose $A$ is **not connected**. Then, there exists an edge $e$ not in $F$, such that $(V, F \cup \{e\})$ still has no cycle (join two connected components)
- when we checked $e$, we did not include it
- that's because that it created a cycle with some edges already in $F$: **impossible**.

# The cut property

**Definition**

**cut**: a partition of the vertices into sets $\boldsymbol{S}$ and $\boldsymbol{V} - \boldsymbol{S}$

**cutset**: the edges between $\boldsymbol{S}$ and $\boldsymbol{V} - \boldsymbol{S}$

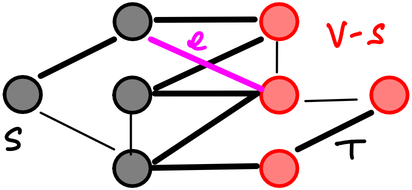
**Claim**

For **any** cut, the minimal weight edge in the cutset is in **any** minimum spanning tree.

# Proof

**For any cut, the minimal weight edge $e$ in the cutset is in any minimum spanning tree.**

- let $T$ be a minimum spanning tree **that does not contain $e$**
- adding $e$ to $T$ creates a cycle $C$, and there must be an edge $e' \neq e$ in $C$ connecting $S$ and $V - S$



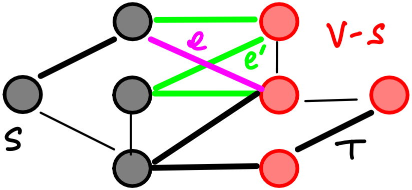

consider $\boldsymbol{T' = T - \{e'\} \cup \{e\}}$

- $w(T') < w(T)$
- but $T'$ is still a spanning tree
  - $n - 1$ edges
  - connected: can replace edge $e'$ by $C - \{e'\}$ to connect vertices
- contradiction

# Proof

**For any cut, the minimal weight edge $e$ in the cutset is in any minimum spanning tree.**

- let $T$ be a minimum spanning tree **that does not contain $e$**
- adding $e$ to $T$ creates a cycle $C$, and there must be an edge $e' \neq e$ in $C$ connecting $S$ and $V - S$



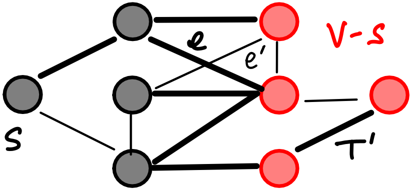

consider $\boldsymbol{T' = T - \{e'\} \cup \{e\}}$

- $w(T') < w(T)$
- but $T'$ is still a spanning tree
  - $n - 1$ edges
  - connected: can replace edge $e'$ by $C - \{e'\}$ to connect vertices
- contradiction

# Proof

**For any cut, the minimal weight edge $e$ in the cutset is in any minimum spanning tree.**

- let $T$ be a minimum spanning tree **that does not contain $e$**
- adding $e$ to $T$ creates a cycle $C$, and there must be an edge $e' \neq e$ in $C$ connecting $S$ and $V - S$




consider $\boldsymbol{T' = T - \{e'\} \cup \{e\}}$

- $w(T') < w(T)$
- but $T'$ is still a spanning tree
  - $n - 1$ edges
  - connected: can replace edge $e'$ by $C - \{e'\}$ to connect vertices
- contradiction

# Kruskal is optimal

**Claim:** every edge we add to the output is in every minimal spanning tree

**Proof:** consider $A = (V, F)$ the forest just before inserting $e = \{u, v\}$, let $S$ be the vertices in the tree containing $u$

**fact 1:** $v$ is in $V - S$ (otherwise, cycle), so $e$ is in the cutset

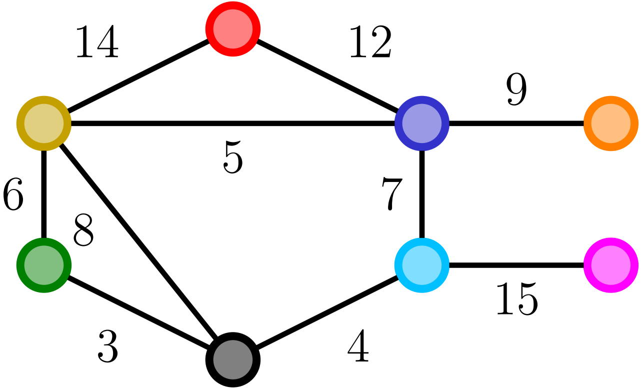
**fact 2:** if $e'$ is another edge in the cutset:

- it has not been considered yet (does not create a cycle, so would have been accepted)
- so $w(e) < w(e')$

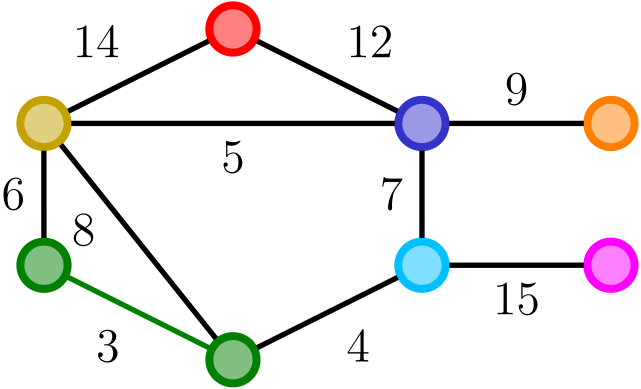
so $e$ has **minimal weight in the cutset**, and it is in every minimal spanning tree

**Remark:** this proves that the minimum spanning tree is unique

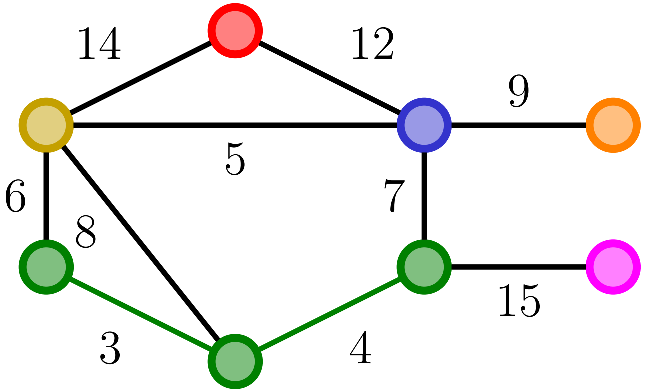
# Merging trees



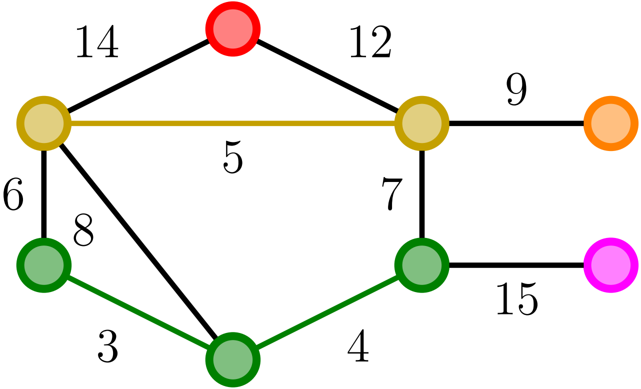
# Merging trees



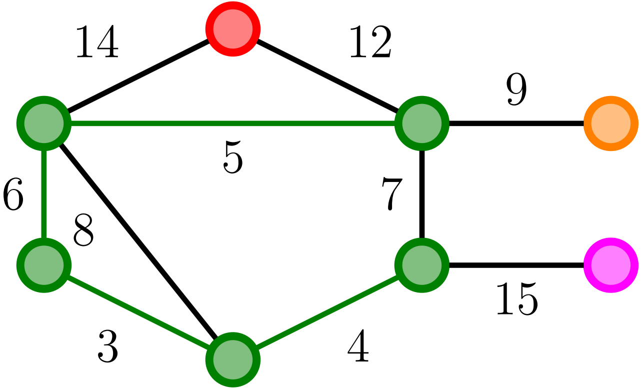
# Merging trees



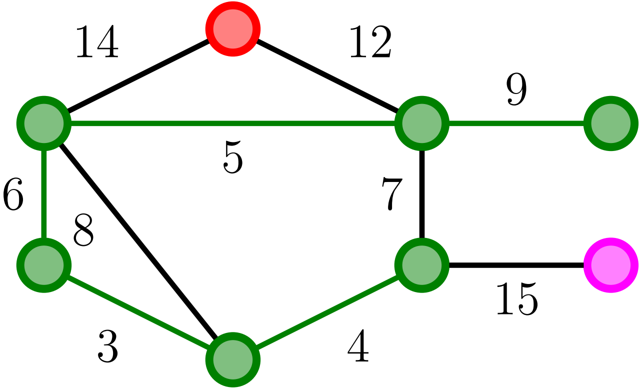
# Merging trees



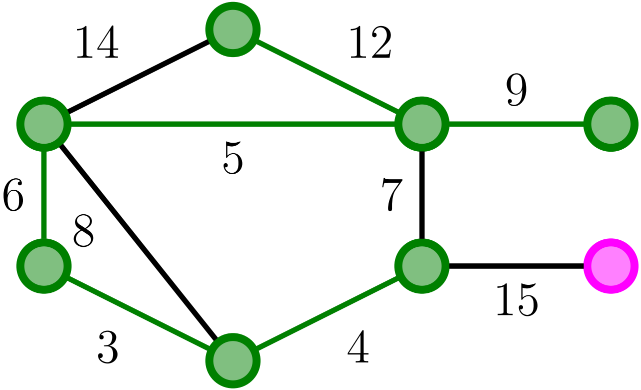
# Merging trees



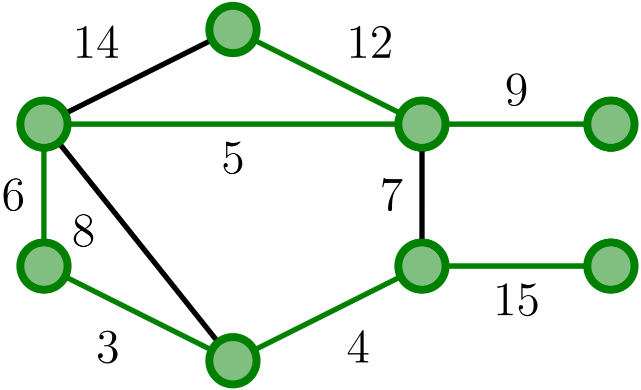
# Merging trees



# Merging trees



# Merging trees



# Data structures

Operations on **disjoint sets of vertices:**

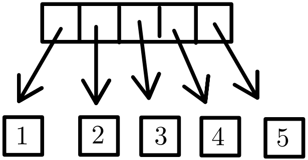
- **Find:** identify which set contains a given vertex
- **Union:** replace two sets by their union

**GreedyMST_UnionFind**$(G)$

$F \leftarrow \{\}$<br>
$S \leftarrow \{\{v_1\}, \ldots, \{v_n\}\}$<br>
sort edges by increasing weight<br>
**for** $k = 1, \ldots, m$ **do**<br>
    **if** $\textbf{find}(S, e_k.1) \neq \textbf{find}(S, e_k.2)$ **then**<br>
        $\textbf{union}(S, \textbf{find}(S, e_k.1), \textbf{find}(S, e_k.2))$<br>
        append $e_k$ to $F$

# An OK solution

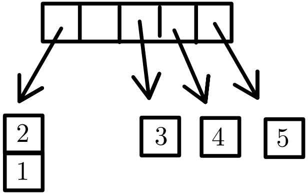
a data structure for union: an array $U$ of **linked lists**




**union_v1**$(U, s, t)$
1. **while** $U[s]$ not **NULL do**
2. $U[t] \leftarrow$ new list$(U[s]$.value$, U[t])$
3. $U[s] \leftarrow U[s]$.next

# An OK solution

a data structure for union: an array $U$ of **linked lists**

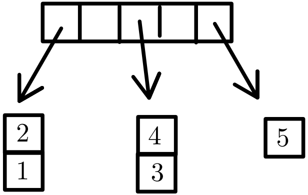



**union_v1**$(U, s, t)$
1. **while** $U[s]$ not **NULL do**
2. $U[t] \leftarrow$ new list$(U[s]$.value$, U[t])$
3. $U[s] \leftarrow U[s]$.next

**union_v1**$(U, 2, 1)$

# An OK solution

a data structure for union: an array $U$ of **linked lists**

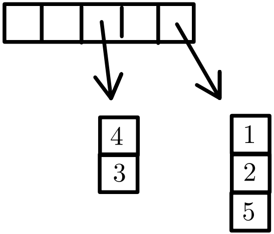



**union_v1**$(U, s, t)$
1. **while** $U[s]$ not **NULL do**
2. $U[t] \leftarrow$ new list$(U[s]$.value$, U[t])$
3. $U[s] \leftarrow U[s]$.next

**union_v1**$(U, 2, 1)$, **union_v1**$(U, 4, 3)$

# An OK solution

a data structure for union: an array $U$ of **linked lists**

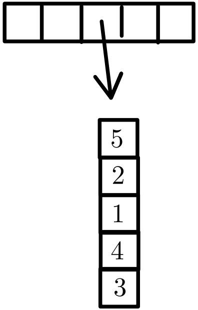



**union_v1**$(U, s, t)$
**while** $U[s]$ not **NULL do**
    $U[t] \leftarrow$ new list$(U[s]$.value$, U[t])$
    $U[s] \leftarrow U[s]$.next

**union_v1**$(U, 2, 1)$, **union_v1**$(U, 4, 3)$, **union_v1**$(U, 1, 5)$

# An OK solution

a data structure for union: an array $U$ of **linked lists**

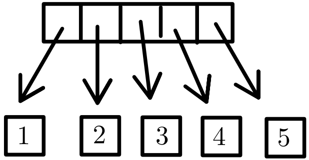



**union_v1**$(U, s, t)$
**while** $U[s]$ not **NULL do**
$U[t] \leftarrow$ new list$(U[s].\text{value}, U[t])$
$U[s] \leftarrow U[s].\text{next}$

**union_v1**$(U, 2, 1)$, **union_v1**$(U, 4, 3)$, **union_v1**$(U, 1, 5)$, **union_v1**$(U, 5, 3)$

# An OK solution

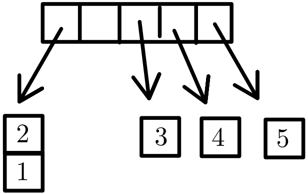
for **find**, use an **array of indices**, $X[i]$ = index of the set that contains $i$ (**find** returns $X[i]$)




$$X = [1, 2, 3, 4, 5]$$

# An OK solution

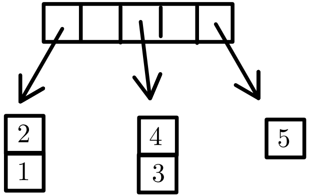
for **find**, use an **array of indices**, $X[i]$ = index of the set that contains $i$ (**find** returns $X[i]$)




$$X = [1, 1, 3, 4, 5]$$

# An OK solution

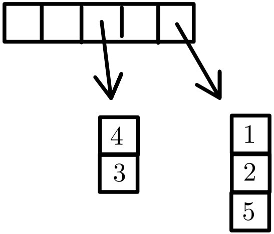
for **find**, use an **array of indices**, $X[i]$ = index of the set that contains $i$ (**find** returns $X[i]$)




$$X = [1, 1, 3, 3, 5]$$

# An OK solution

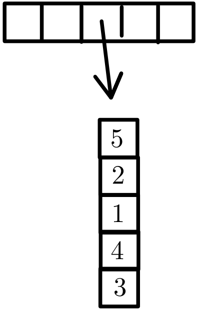
for **find**, use an **array of indices**, $X[i]$ = index of the set that contains $i$ (**find** returns $X[i]$)




$$X = [5, 5, 3, 3, 5]$$

# An OK solution

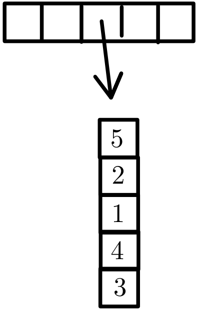
for **find**, use an **array of indices**, $X[i]$ = index of the set that contains $i$ (**find** returns $X[i]$)




$$X = [3, 3, 3, 3, 3]$$

# An OK solution

for **find**, use an **array of indices**, $X[i]$ = index of the set that contains $i$ (**find** returns $X[i]$)




**union_v2**$(X, U, s, t)$

    **while** $U[s]$ not **NULL do**

        $U[t] \leftarrow$ new list$(U[s].\text{value}, U[t])$

        $X[U[s].\text{value}] \leftarrow t$

        $U[s] \leftarrow U[s].\text{next}$

# Analysis

**Worst case:**

- **find** is $O(1)$
- **union** **traverses** one of the linked lists: worst case $\Theta(n)$

**Kruskal's algorithm:**

- sorting edges $\boldsymbol{O(m\log(m))}$
- $O(m)$ calls to **find**
- $O(n)$ calls to **union**

Worst case $\boldsymbol{O(m\log(m)+n^2)}$

# A simple heuristics for Union

**Modified Union**

- each list in $U$ keeps track of its size
- merge the **smaller list** into the **larger list**

**Key observation:** worst case for **one** union **still** $\Theta(n)$, but the amortized cost is better.

- for any vertex $v$, the size of the list containing $v$ **at least doubles** when it moves to another list
- so $v$ moves at most $\log(n)$ times
- so the **total** cost of union **per vertex** is $O(\log(n))$

**Conclusion:** $\boldsymbol{O(n \log(n))}$ for all unions and $\boldsymbol{O(m \log(m))}$ total

# Alternate amortized proof

$$
\begin{aligned}
\text{cost}(\text{all union operations}) &= \sum_{\text{union } u} \text{cost}(u) \\
&= \sum_{\text{union } u} \text{length}(\text{smaller set of } u) && \in \sum_{\text{union } u} O(n) \\
&= \sum_{\text{union } u} \sum_{\text{vertex } v} \begin{cases} 1 & \text{if } v \in \text{smaller set of } u \\ 0 & \text{otherwise.} \end{cases} \\
&= \sum_{\text{vertex } v} \sum_{\text{union } u} \begin{cases} 1 & \text{if } v \in \text{smaller set of } u \\ 0 & \text{otherwise.} \end{cases} \\
&\leq \sum_{\text{vertex } v} \log n \\
&\in O(n \log n)
\end{aligned}
$$

# Prim's algorithm (time permitting)

# The idea

### Goal

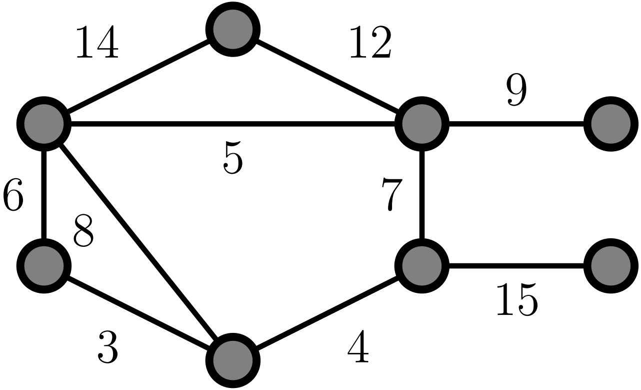
- $G$ is an undirected graph
- $w : E \rightarrow R$ a weight function
- as before, want a **minimum weight spanning tree**

### The idea:

- start from an **arbitrary source**
- **grow a tree** (connected, no cycle) edge-by-edge
- new edges chosen in a greedy manner

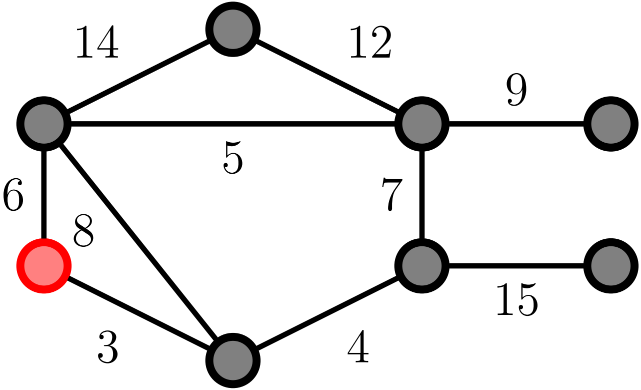
# Growing a tree

We grow $\boldsymbol{A} = (\boldsymbol{S}, \boldsymbol{F})$ by adding the **minimal weight edge** $\boldsymbol{S} \leftrightarrow (\boldsymbol{V} - \boldsymbol{S})$



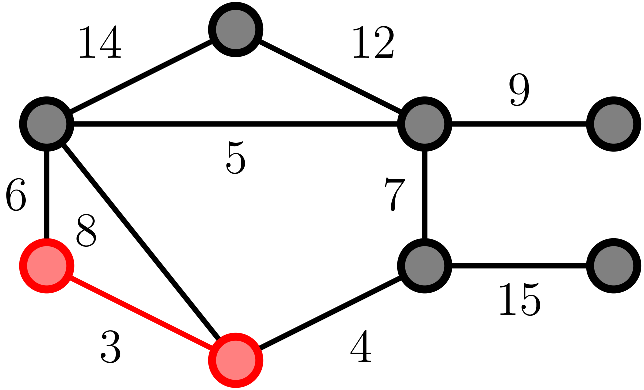
# Growing a tree

We grow $A = (S, F)$ by adding the **minimal weight edge** $S \leftrightarrow (V - S)$



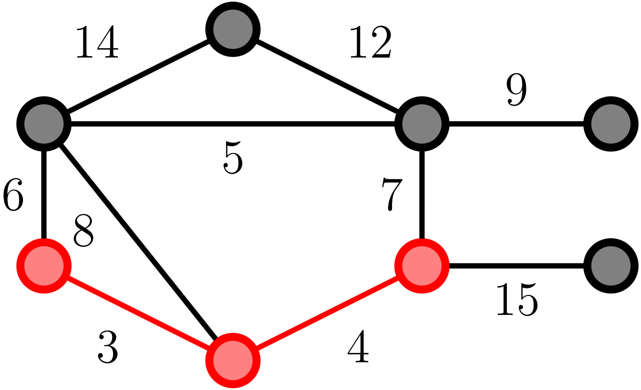
# Growing a tree

We grow $\boldsymbol{A} = (\boldsymbol{S}, \boldsymbol{F})$ by adding the **minimal weight edge** $\boldsymbol{S} \leftrightarrow (\boldsymbol{V} - \boldsymbol{S})$



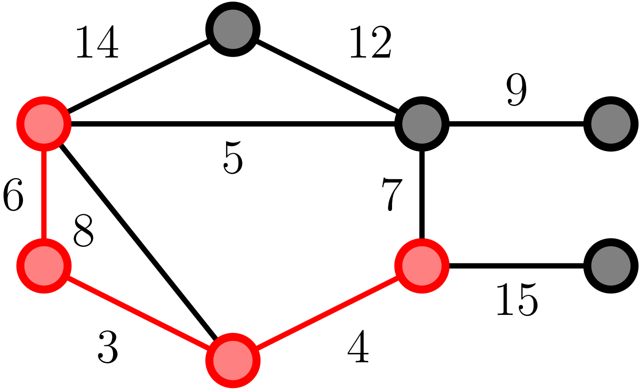
# Growing a tree

We grow $\boldsymbol{A} = (\boldsymbol{S}, \boldsymbol{F})$ by adding the **minimal weight edge** $\boldsymbol{S} \leftrightarrow (\boldsymbol{V} - \boldsymbol{S})$



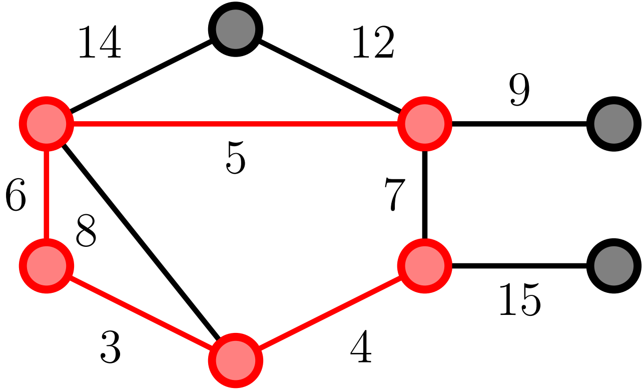
# Growing a tree

We grow $\boldsymbol{A} = (\boldsymbol{S}, \boldsymbol{F})$ by adding the **minimal weight edge** $\boldsymbol{S} \leftrightarrow (\boldsymbol{V} - \boldsymbol{S})$



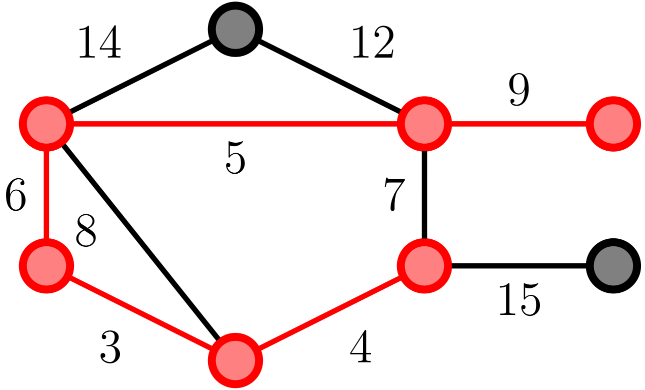
# Growing a tree

We grow $\boldsymbol{A} = (\boldsymbol{S}, \boldsymbol{F})$ by adding the **minimal weight edge** $\boldsymbol{S} \leftrightarrow (\boldsymbol{V} - \boldsymbol{S})$



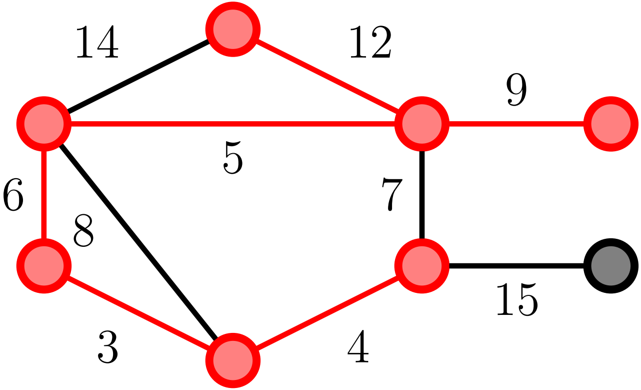
# Growing a tree

We grow $\boldsymbol{A} = (\boldsymbol{S}, \boldsymbol{F})$ by adding the **minimal weight edge** $\boldsymbol{S} \leftrightarrow (\boldsymbol{V} - \boldsymbol{S})$



# Growing a tree

We grow $\boldsymbol{A} = (\boldsymbol{S}, \boldsymbol{F})$ by adding the **minimal weight edge** $\boldsymbol{S} \leftrightarrow (\boldsymbol{V} - \boldsymbol{S})$



# Growing a tree

We grow $\boldsymbol{A} = (\boldsymbol{S}, \boldsymbol{F})$ by adding the **minimal weight edge** $\boldsymbol{S} \leftrightarrow (\boldsymbol{V} - \boldsymbol{S})$