# CMPS 6610/4610 – Fall 2016

# *Minimum Spanning Trees*

**Carola Wenk**

Slides courtesy of Charles Leiserson with changes and additions by Carola Wenk

# Minimum spanning trees

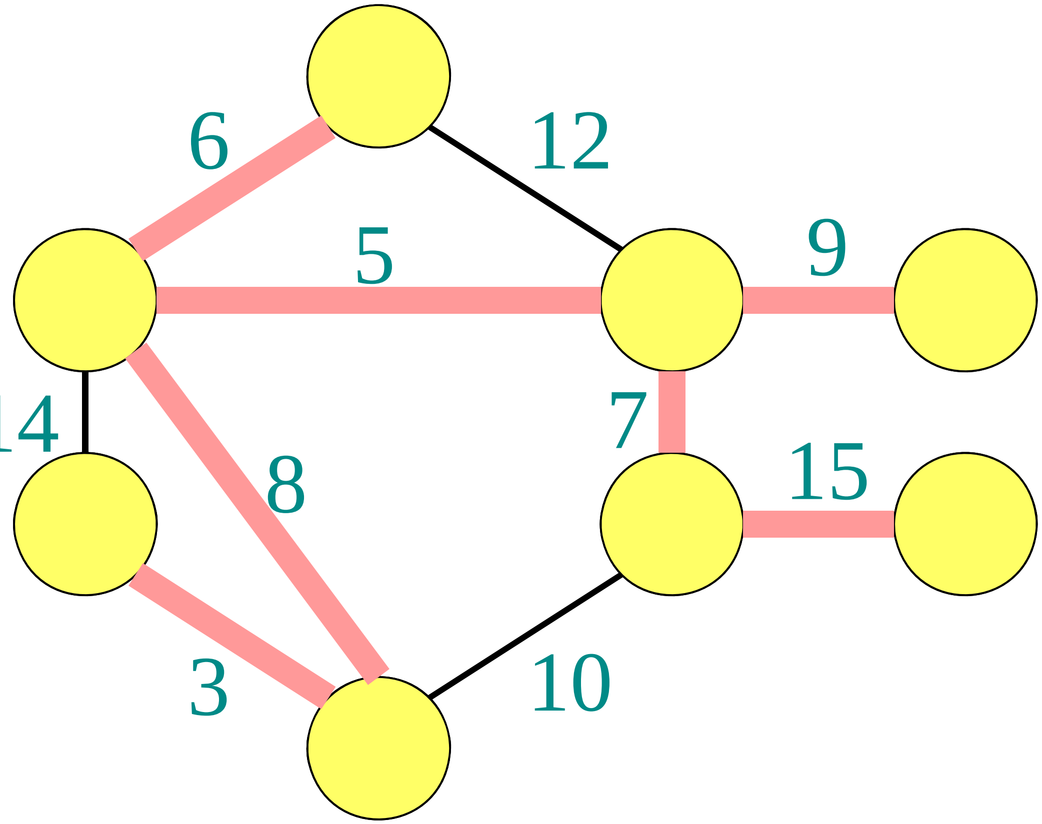
**Input:** A connected, undirected graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$.

- For simplicity, assume that all edge weights are distinct.

**Output:** A ***spanning tree*** $T$ — a tree that connects all vertices — of minimum weight:

$$w(T) = \sum_{(u,v)\in T} w(u,v).$$

# Example of MST



# Growing an MST

Grow an MST by greedily adding one edge at a time.

GENERIC-MST(G,w){
  $T \leftarrow \emptyset$
  **while** T does not form a spanning tree {
    // Maintain invariant that T is a subset of an MST for G

    Find a "safe" edge {u,v} such that $T \cup \{\{u,v\}\}$ is a subset of an MST for G
    $T \leftarrow T \cup \{\{u,v\}\}$
  }
  return A
}

# Hallmark for "greedy" algorithms

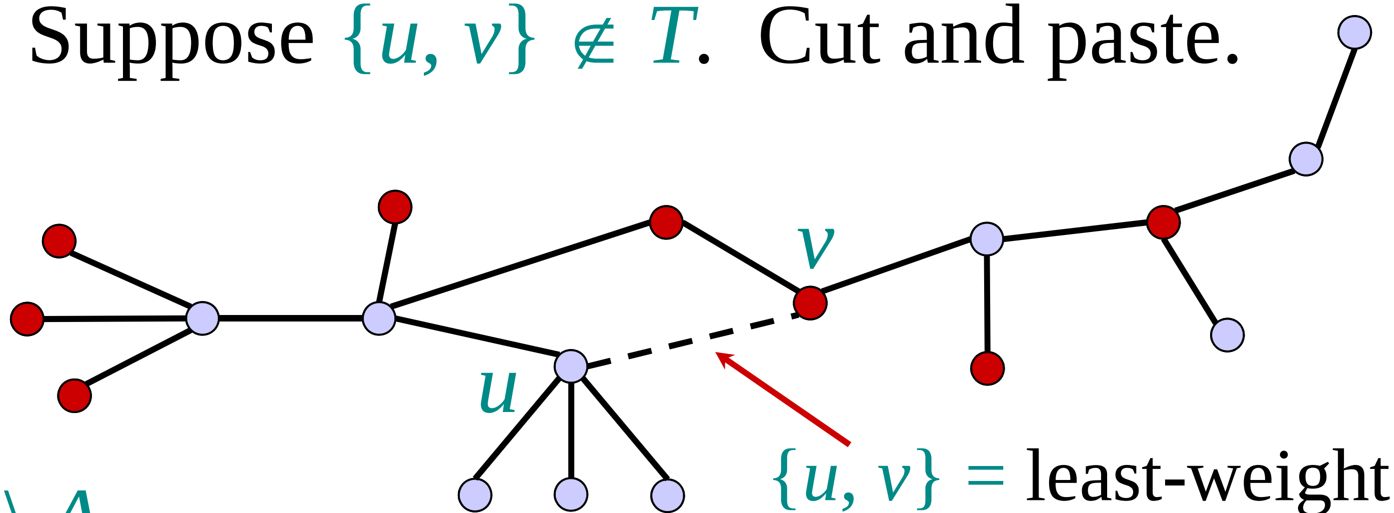
***Greedy-choice property***
*A locally optimal choice is globally optimal.*

**Theorem [Cut property].** Let $G = (V, E)$ and let $A \subseteq V$. Suppose that $\{u, v\} \in E$ is the least-weight edge connecting $A$ to $V \setminus A$. Then, $\{u, v\}$ is contained in an MST $T$ of $G$.

# Proof of theorem

*Proof*. Suppose $\{u, v\} \notin T$. Cut and paste.

$T$:



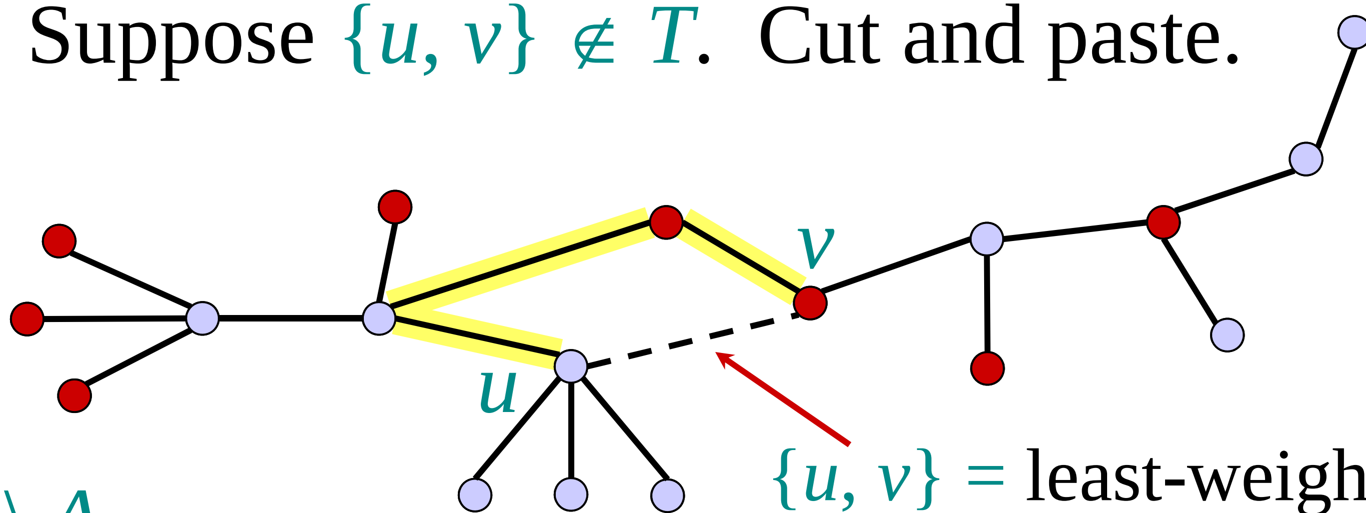

$\{u, v\}$ = least-weight edge connecting $A$ to $V \setminus A$

- ○ $\in A$
- ● $\in V \setminus A$

# Proof of theorem

*Proof.* Suppose $\{u, v\} \notin T$. Cut and paste.

$T$:




○ $\in A$

● $\in V \setminus A$

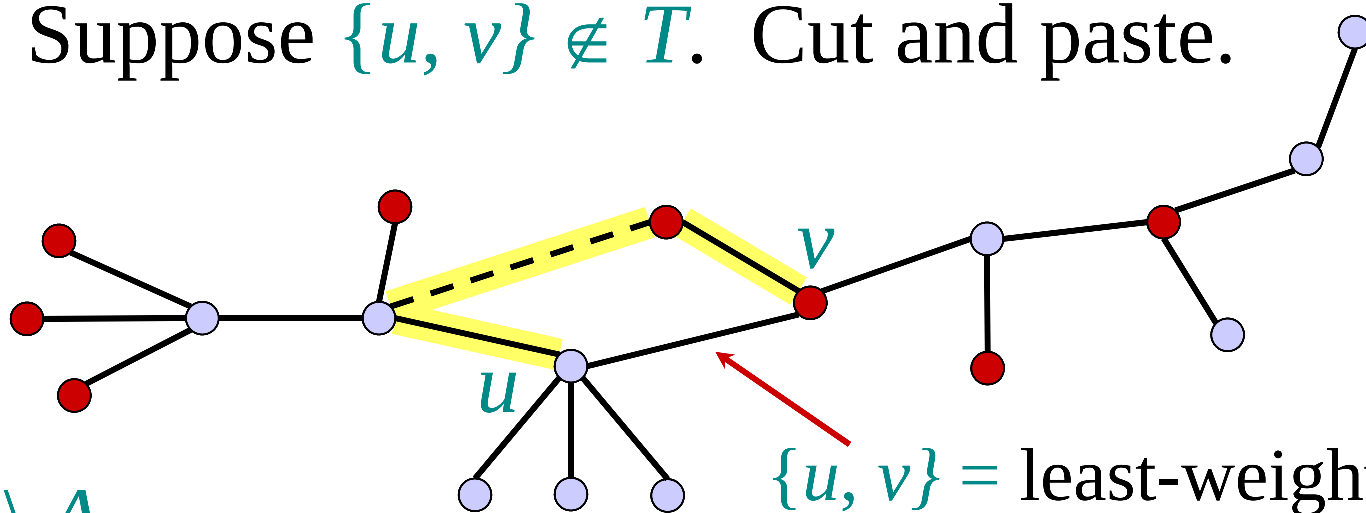
$\{u, v\}$ = least-weight edge connecting $A$ to $V \setminus A$

Consider the unique simple path from $u$ to $v$ in $T$.

# Proof of theorem

*Proof*. Suppose $\{u, v\} \notin T$. Cut and paste.

$T$:




$\in A$

$\in V \setminus A$

$\{u, v\}$ = least-weight edge connecting $A$ to $V \setminus A$

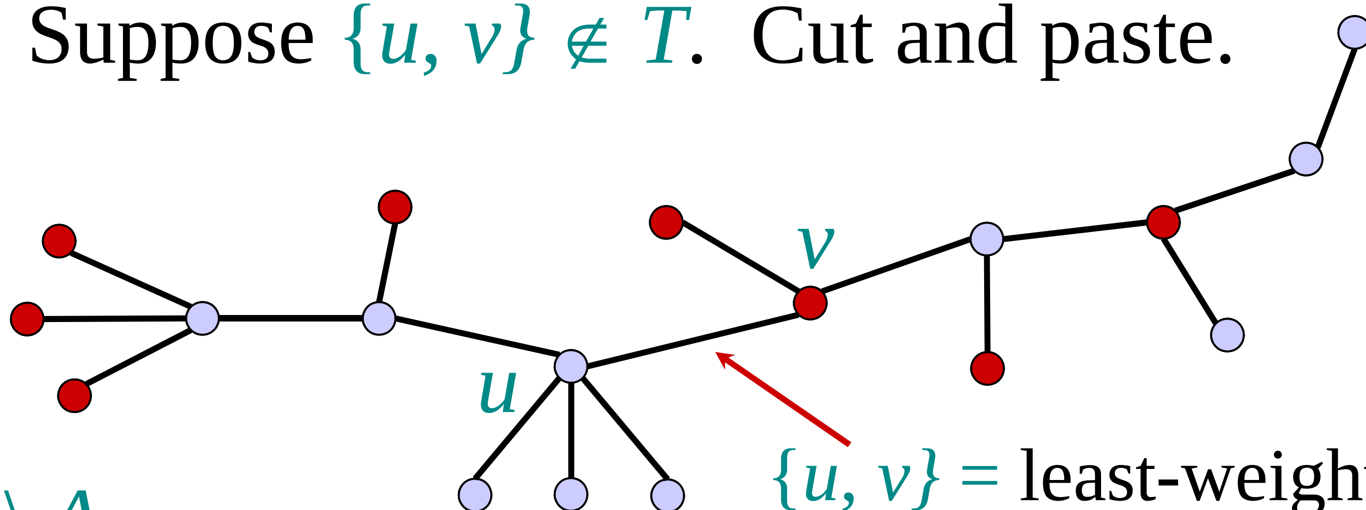
Consider the unique simple path from $u$ to $v$ in $T$.

Swap $\{u, v\}$ with the first edge on this path that connects a vertex in $A$ to a vertex in $V \setminus A$.

# Proof of theorem

*Proof.* Suppose $\{u, v\} \notin T$. Cut and paste.

$T'$:




○ $\in A$

● $\in V \setminus A$

$\{u, v\}$ = least-weight edge connecting $A$ to $V \setminus A$

Consider the unique simple path from $u$ to $v$ in $T$.

Swap $\{u, v\}$ with the first edge on this path that connects a vertex in $A$ to a vertex in $V \setminus A$.

A lighter-weight spanning tree than $T$ results. □

# MST algorithms

- Prim's algorithm:
  - Maintains one tree
  - Runs in time $O(|E| \log |V|)$ with binary heaps, in time $O(|E| + |V| \log |V|)$, with Fibonacci heaps
- Kruskal's algorithm:
  - Maintains a forest and uses the disjoint-set data structure
  - Runs in time $O(|E| \log |E|)$

# Prim's algorithm

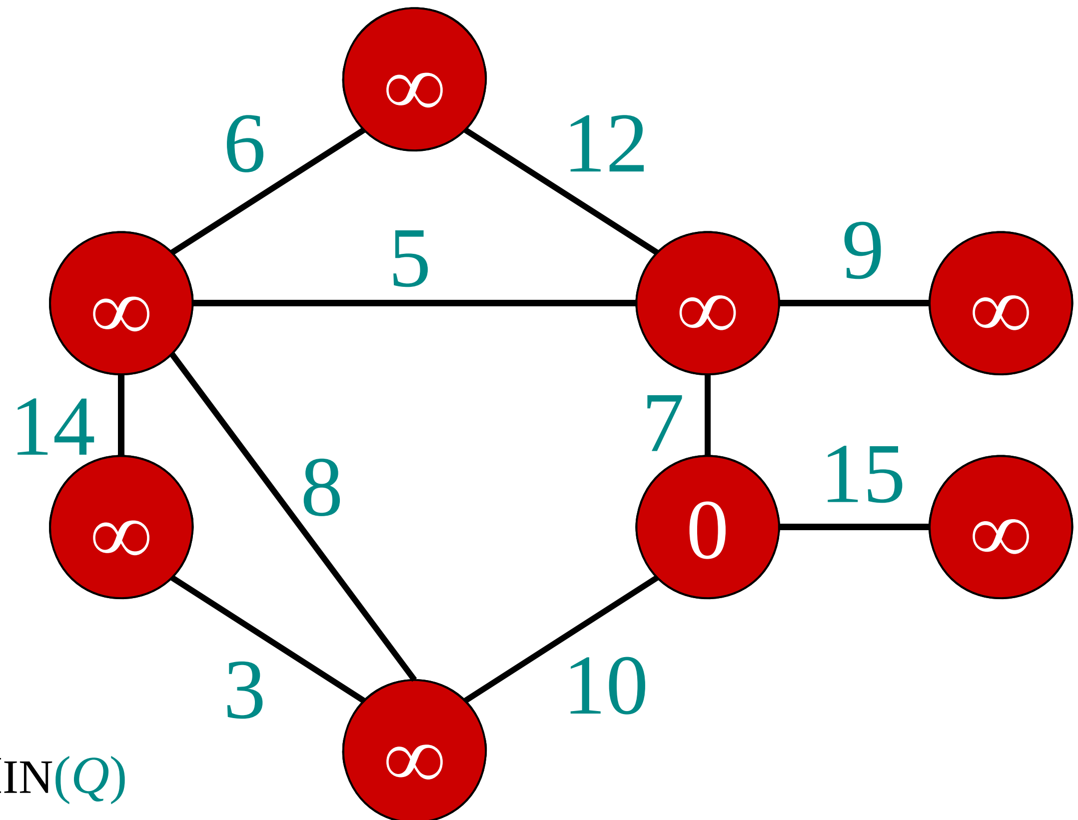
**IDEA:** Maintain $V \setminus A$ as a priority queue $Q$. Key each vertex in $Q$ with the weight of the least-weight edge connecting it to a vertex in $A$.

$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$
**while** $Q \neq \emptyset$
  **do** $u \leftarrow$ EXTRACT-MIN$(Q)$
    **for** each $v \in Adj[u]$
      **do if** $v \in Q$ and $w(u, v) < key[v]$
        **then** $key[v] \leftarrow w(u, v)$  ⊳ DECREASE-KEY
          $\pi[v] \leftarrow u$

**Dijkstra:**
**while** $Q \neq \emptyset$ **do**
  $u \leftarrow$ EXTRACT-MIN$(Q)$
  $S \leftarrow S \cup \{u\}$
  **for** each $v \in Adj[u]$ **do**
    **if** $d[v] > d[u] + w(u, v)$ **then**
      $d[v] \leftarrow d[u] + w(u, v)$

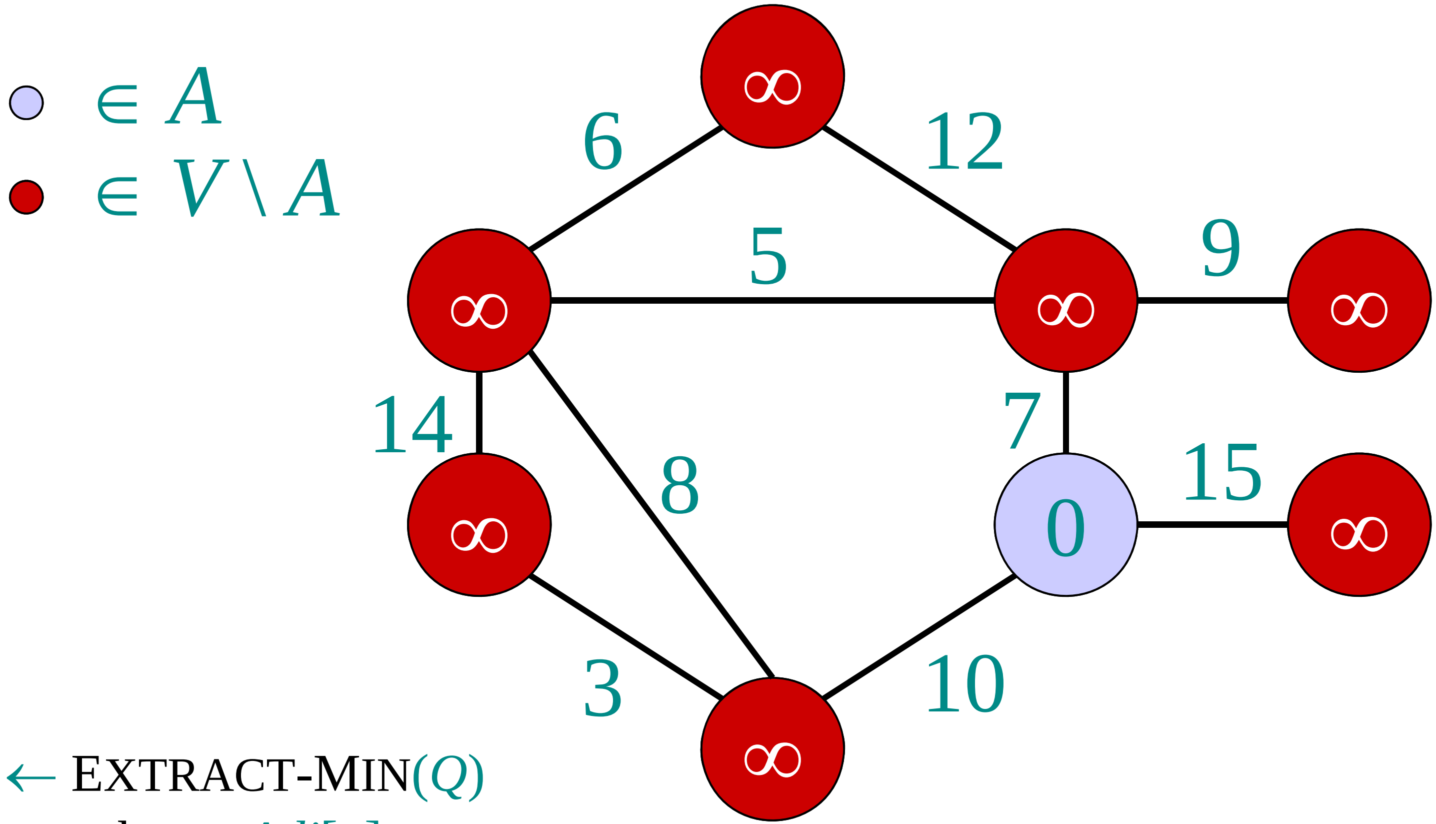
At the end, $\{(v, \pi[v])\}$ forms the MST edges.

# Example of Prim's algorithm

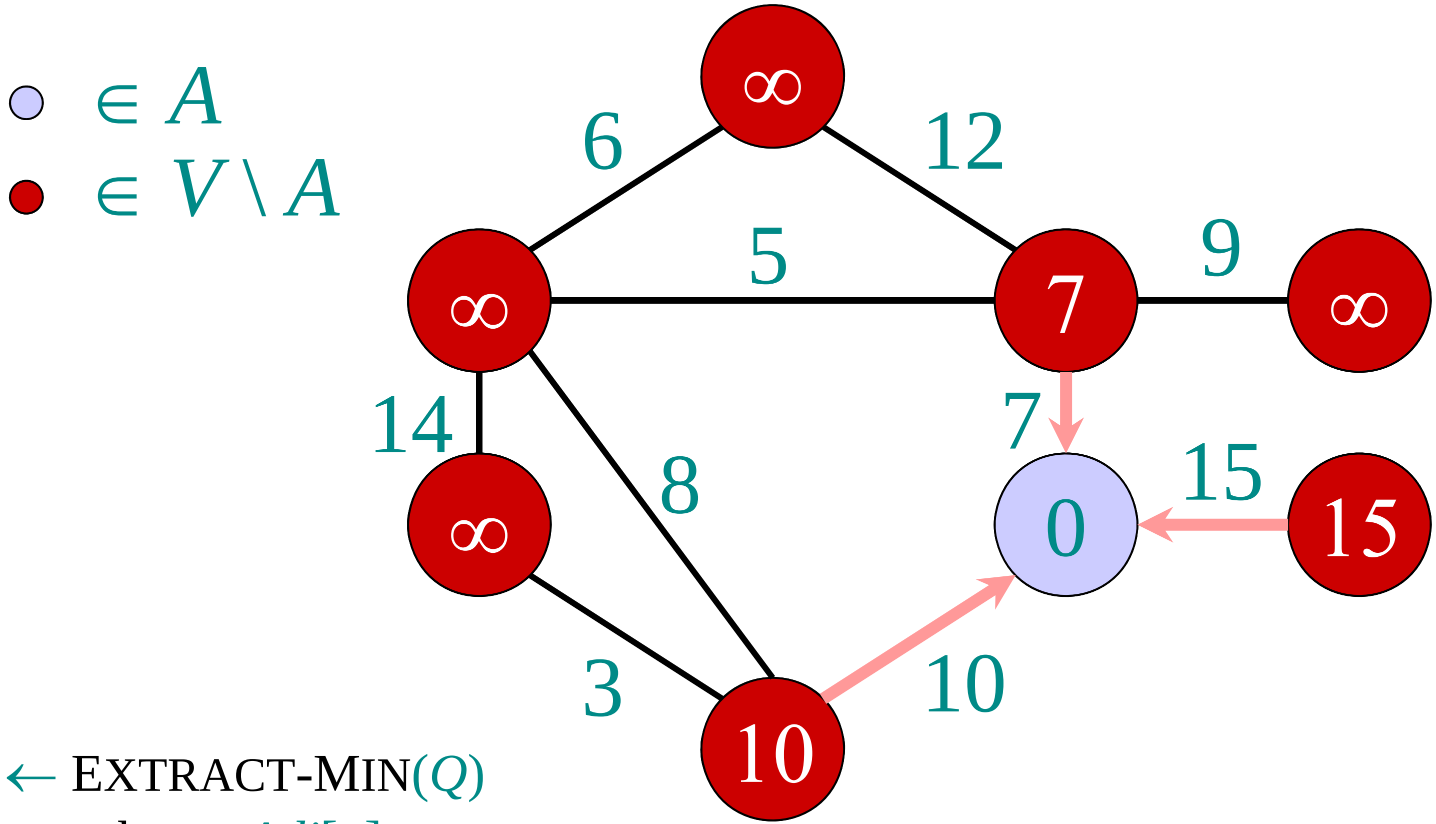

$u \leftarrow$ EXTRACT-MIN$(Q)$
**for** each $v \in Adj[u]$
**do if** $v \in Q$ and $w(u, v) < key[v]$
**then** $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
$\pi[v] \leftarrow u$

# Example of Prim's algorithm



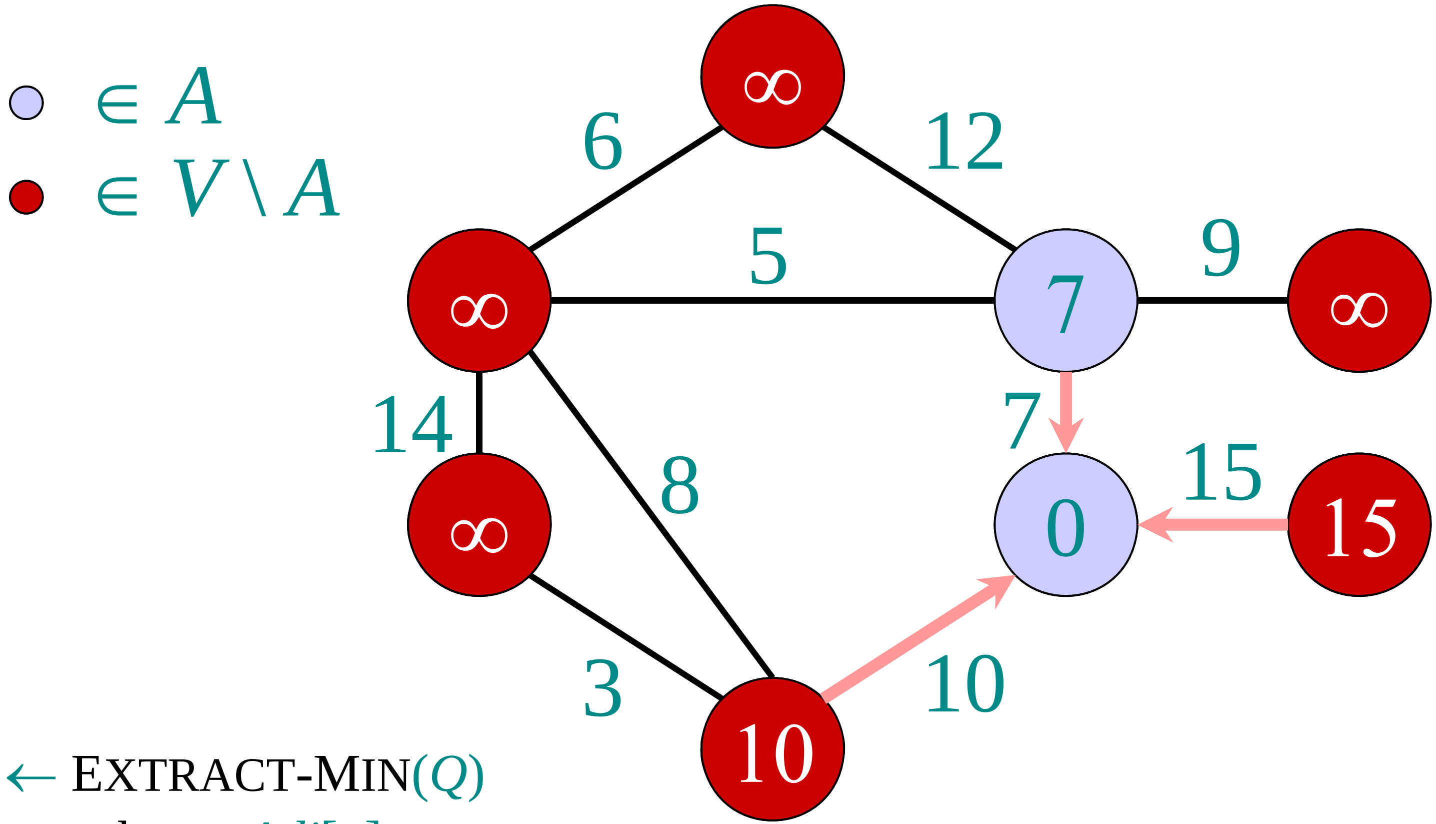

$u \leftarrow$ EXTRACT-MIN$(Q)$
**for** each $v \in Adj[u]$
**do if** $v \in Q$ and $w(u, v) < key[v]$
**then** $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
$\pi[v] \leftarrow u$

# Example of Prim's algorithm




$u \leftarrow$ EXTRACT-MIN$(Q)$
**for** each $v \in Adj[u]$
    **do if** $v \in Q$ and $w(u, v) < key[v]$
        **then** $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
            $\pi[v] \leftarrow u$

# Example of Prim's algorithm




$u \leftarrow$ EXTRACT-MIN$(Q)$

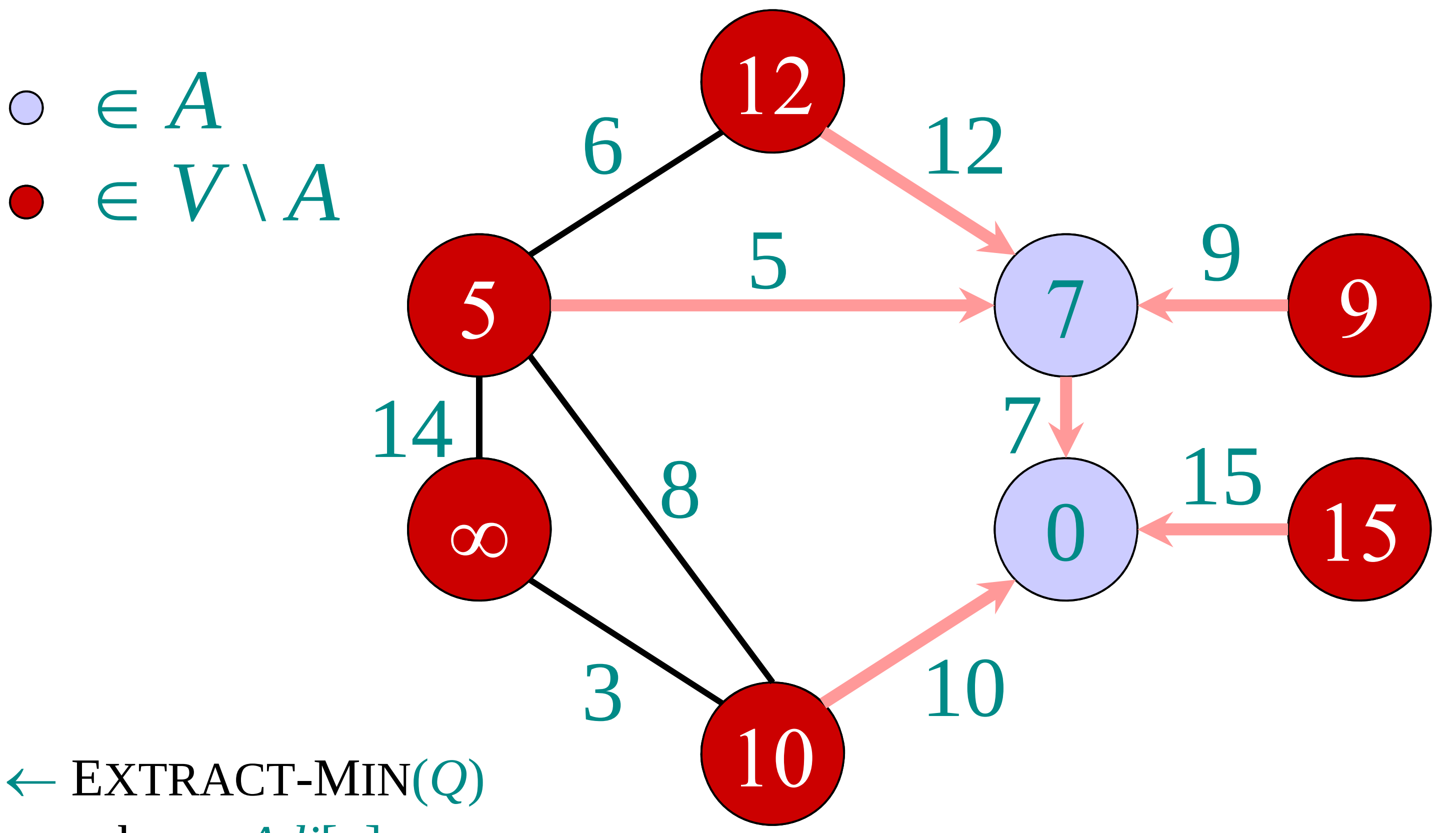
**for** each $v \in Adj[u]$

**do if** $v \in Q$ and $w(u, v) < key[v]$

**then** $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY

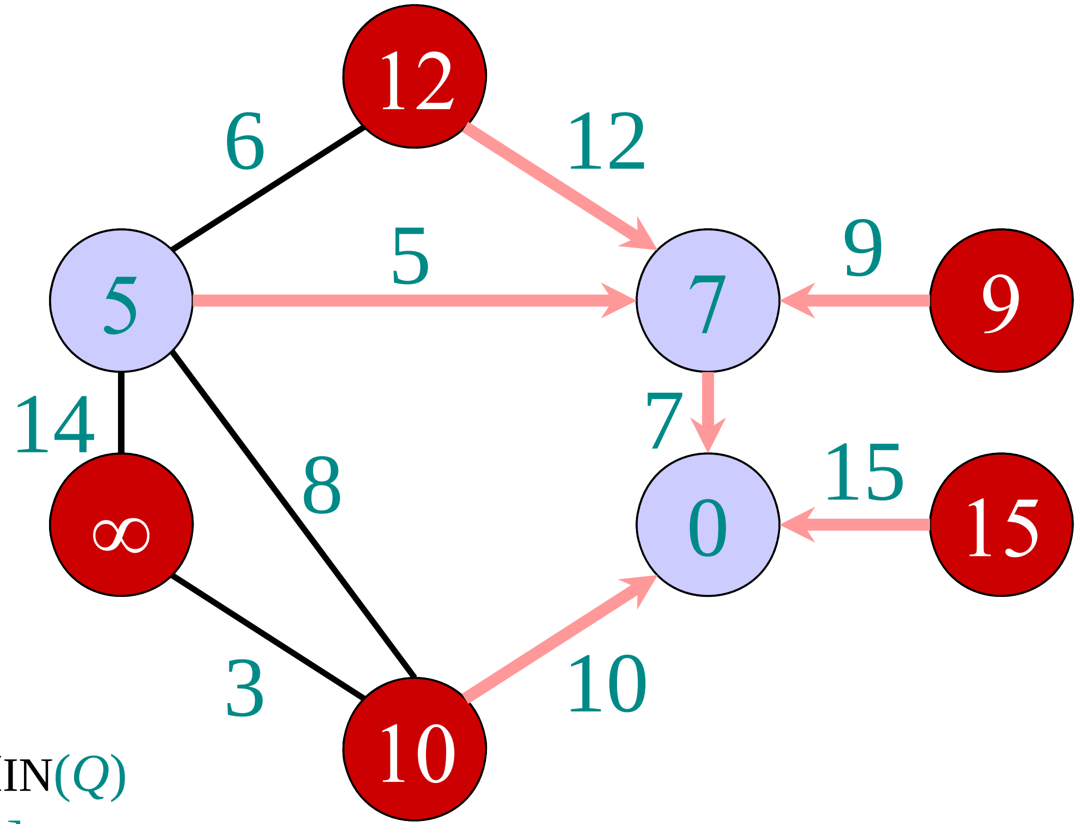
$\pi[v] \leftarrow u$

# Example of Prim's algorithm



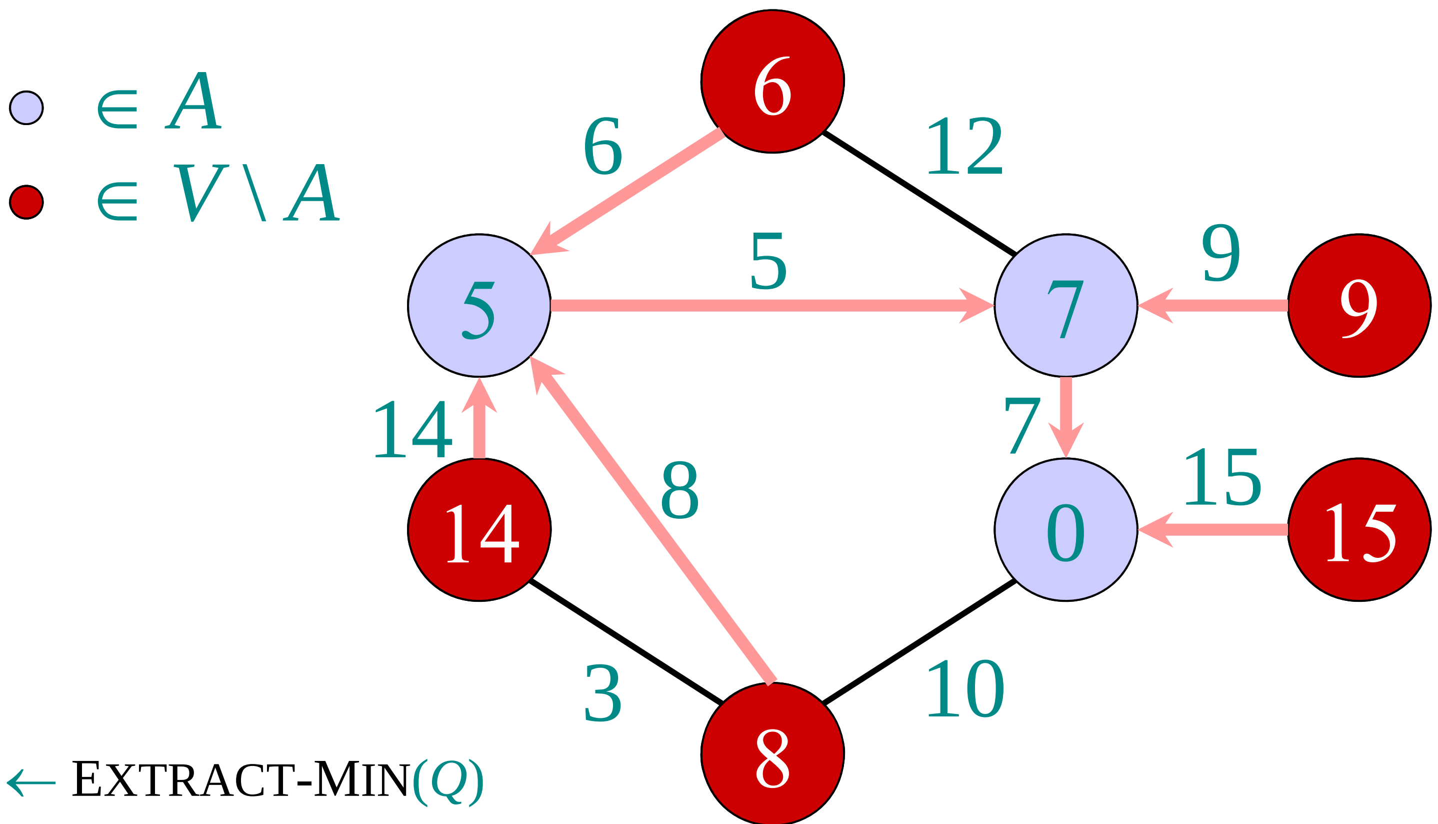

$u \leftarrow$ EXTRACT-MIN($Q$)
**for** each $v \in Adj[u]$
**do if** $v \in Q$ and $w(u, v) < key[v]$
**then** $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
$\pi[v] \leftarrow u$

# Example of Prim's algorithm

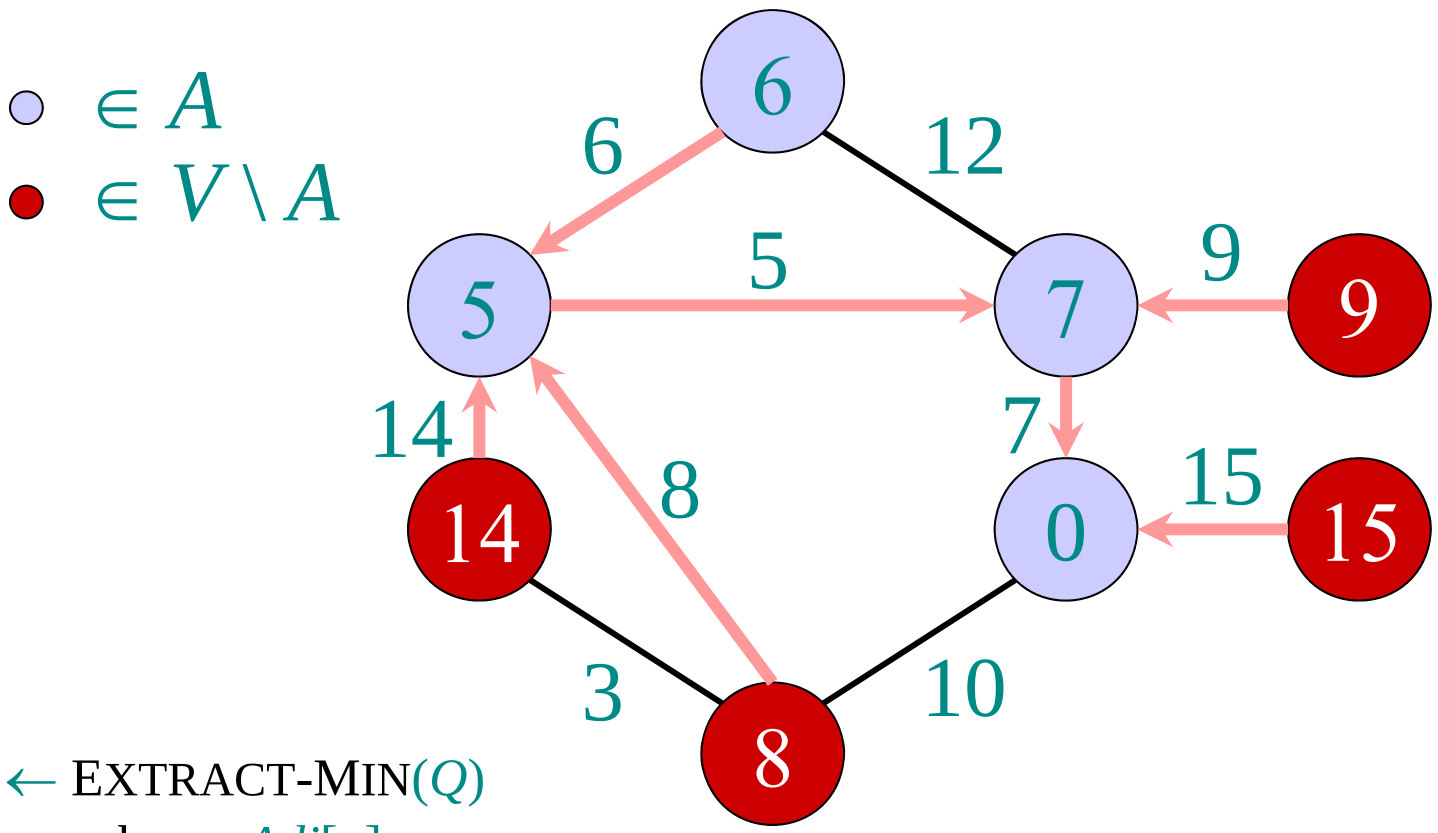

$u \leftarrow$ EXTRACT-MIN$(Q)$
**for** each $v \in Adj[u]$
**do if** $v \in Q$ and $w(u, v) < key[v]$
**then** $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
$\pi[v] \leftarrow u$

# Example of Prim's algorithm



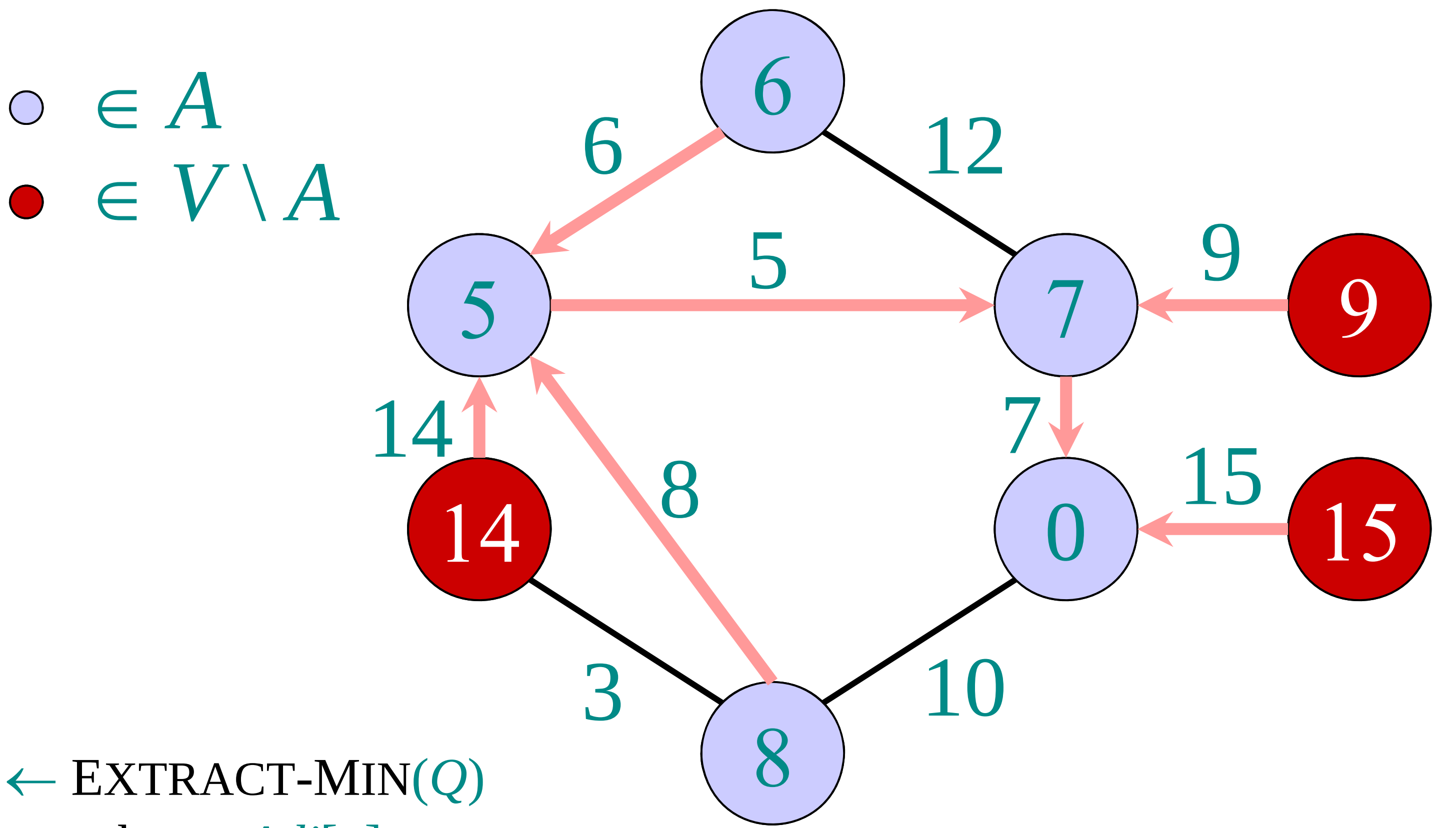

$u \leftarrow$ EXTRACT-MIN$(Q)$
**for** each $v \in Adj[u]$
  **do if** $v \in Q$ and $w(u, v) < key[v]$
    **then** $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
      $\pi[v] \leftarrow u$

# Example of Prim's algorithm



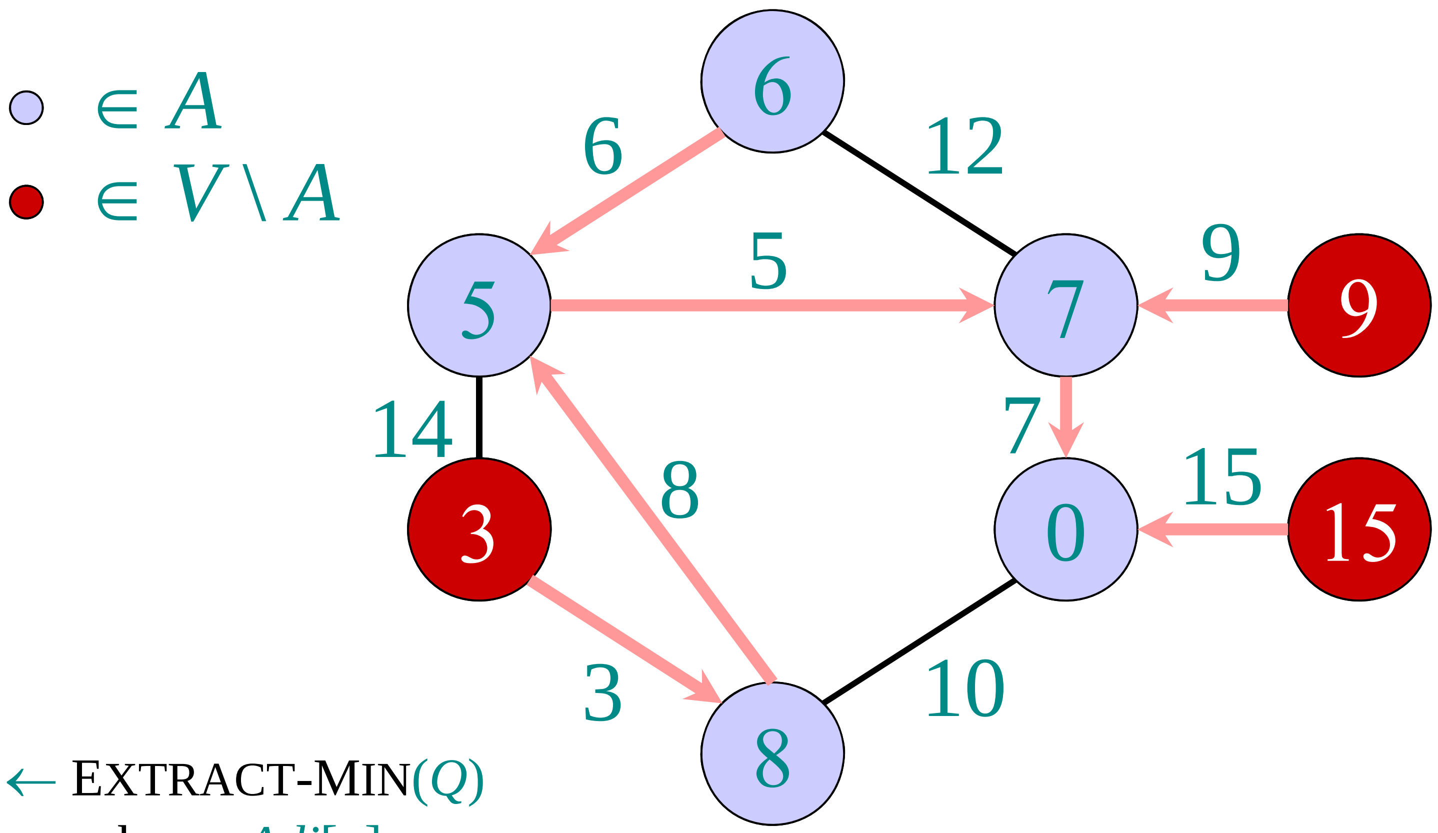

$u \leftarrow$ EXTRACT-MIN$(Q)$
**for** each $v \in Adj[u]$
  **do if** $v \in Q$ and $w(u, v) < key[v]$
    **then** $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
      $\pi[v] \leftarrow u$

# Example of Prim's algorithm



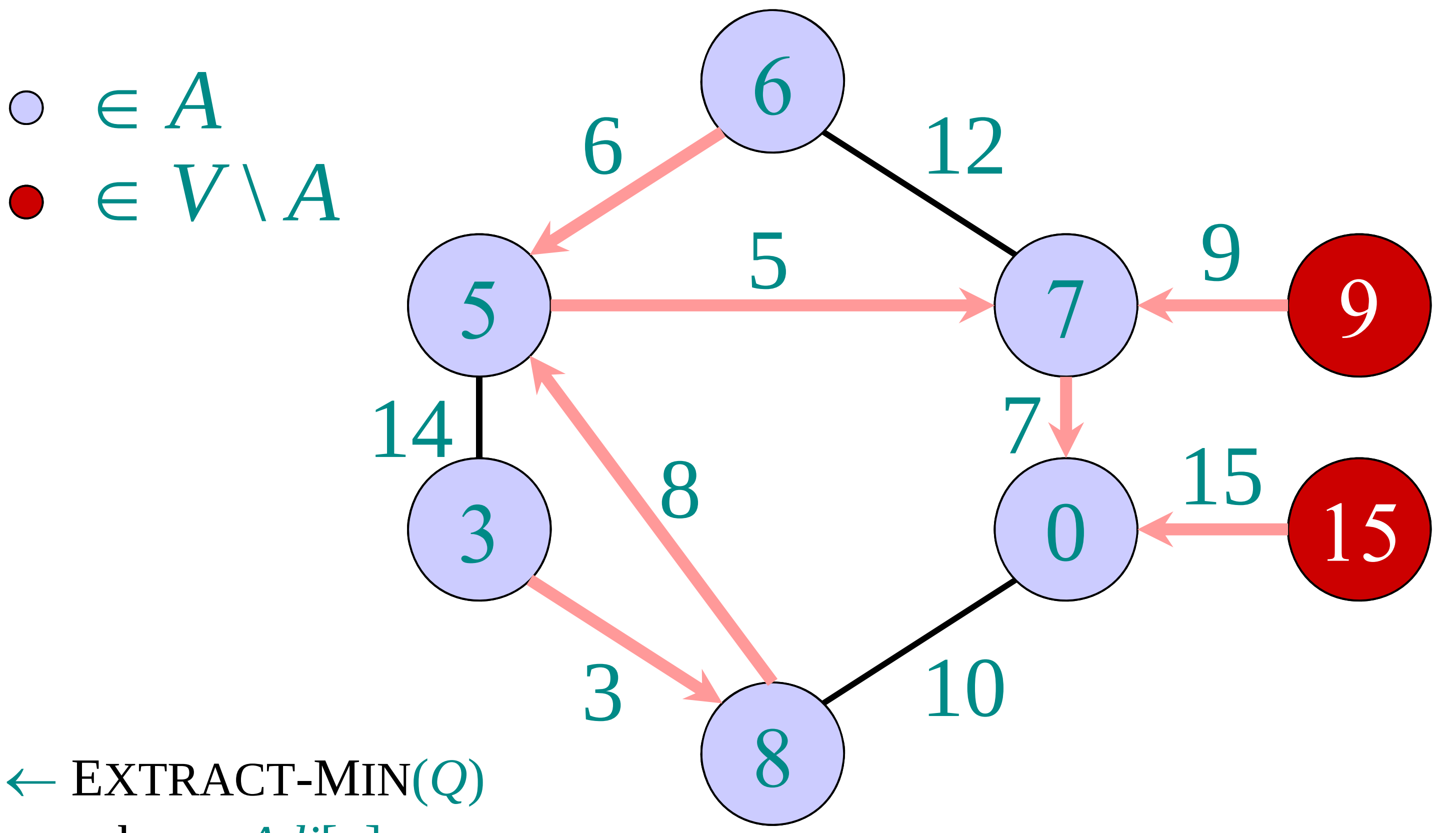

$u \leftarrow$ EXTRACT-MIN$(Q)$
**for** each $v \in Adj[u]$
  **do if** $v \in Q$ and $w(u, v) < key[v]$
    **then** $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
      $\pi[v] \leftarrow u$

# Example of Prim's algorithm



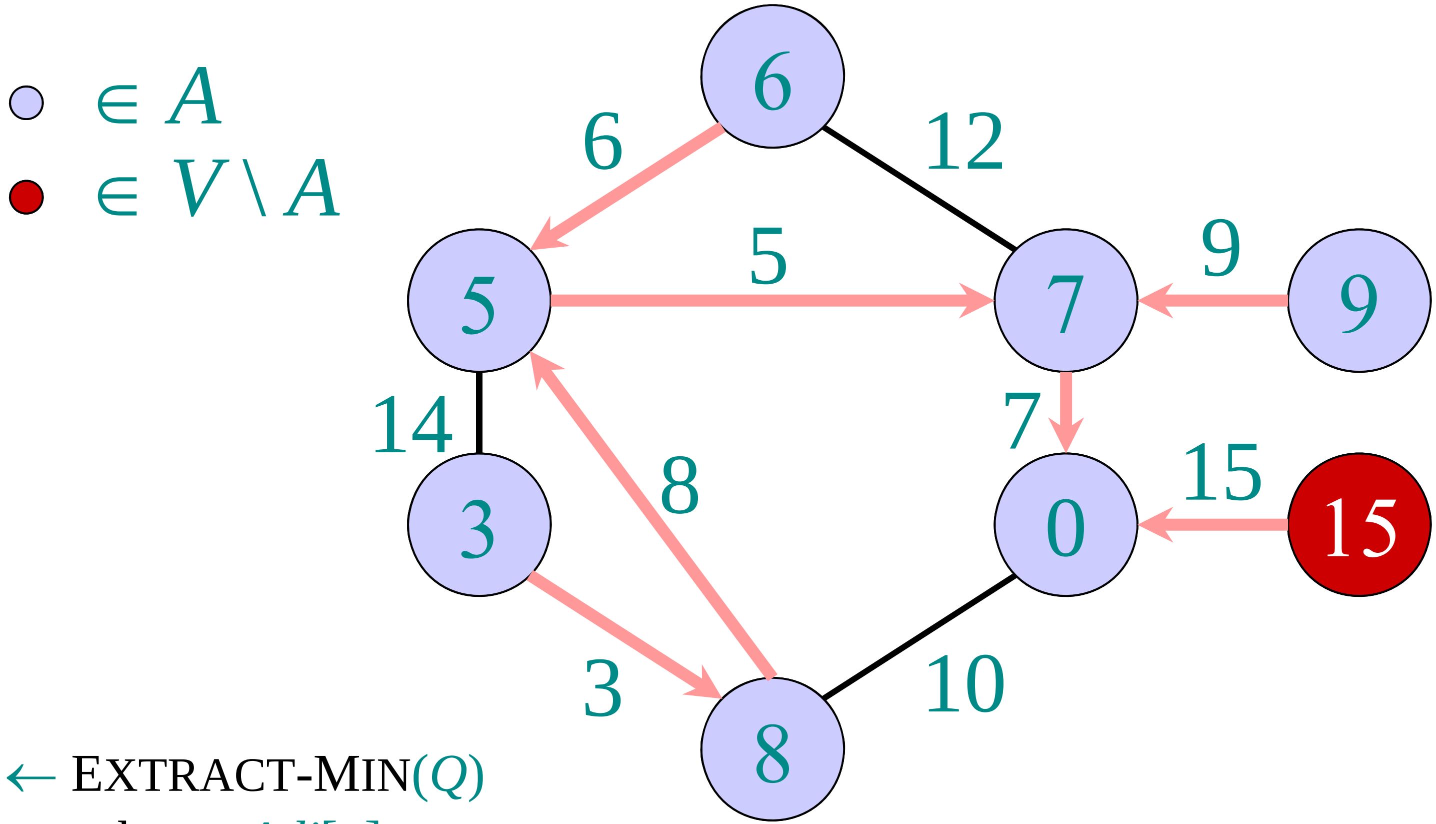

$u \leftarrow$ EXTRACT-MIN($Q$)
**for** each $v \in Adj[u]$
    **do if** $v \in Q$ and $w(u, v) < key[v]$
        **then** $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
            $\pi[v] \leftarrow u$

# Example of Prim's algorithm




$u \leftarrow$ EXTRACT-MIN($Q$)
**for** each $v \in Adj[u]$
**do if** $v \in Q$ and $w(u, v) < key[v]$
**then** $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
$\pi[v] \leftarrow u$

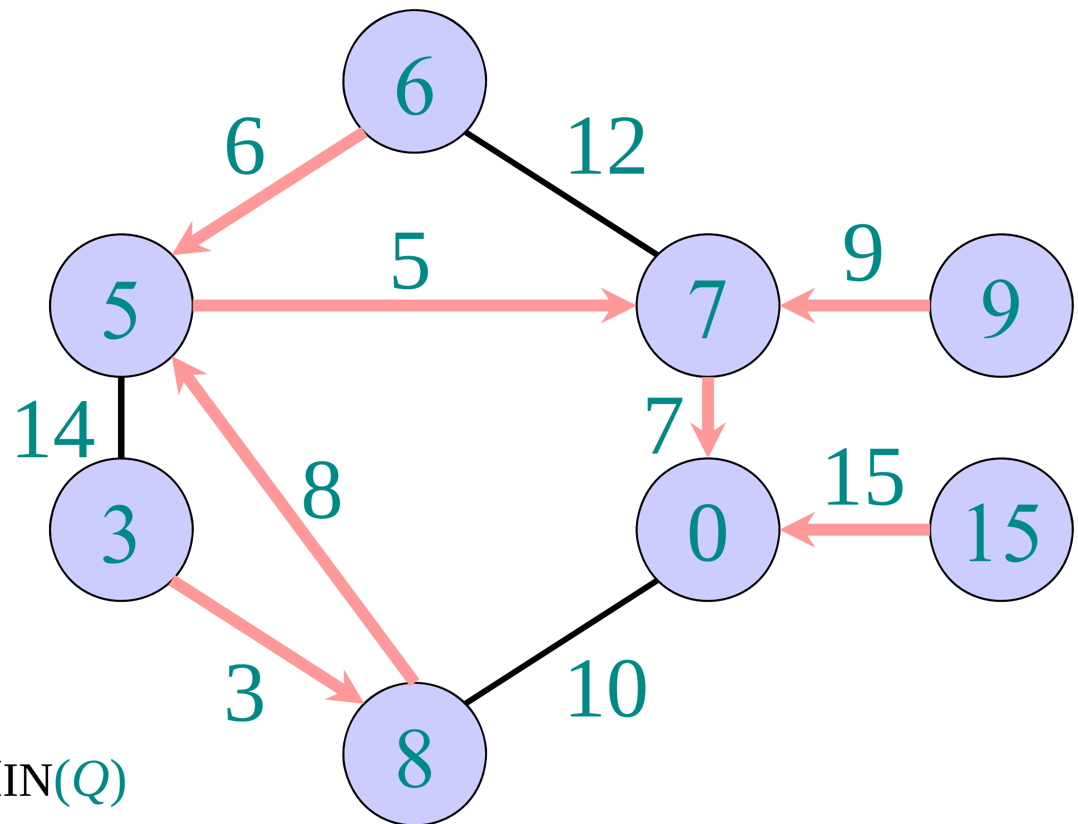
# Example of Prim's algorithm




$u \leftarrow$ EXTRACT-MIN$(Q)$
**for** each $v \in Adj[u]$
**do if** $v \in Q$ and $w(u, v) < key[v]$
**then** $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
$\pi[v] \leftarrow u$

# Example of Prim's algorithm

○ $\in A$
● $\in V \setminus A$




$u \leftarrow$ EXTRACT-MIN$(Q)$
**for** each $v \in Adj[u]$
  **do if** $v \in Q$ and $w(u, v) < key[v]$
    **then** $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
      $\pi[v] \leftarrow u$

# Analysis of Prim

$\Theta(|V|)$ total:

$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$

$|V|$ times:

**while** $Q \neq \emptyset$
  **do** $u \leftarrow$ EXTRACT-MIN$(Q)$
    $degree(u)$ times:
    **for** each $v \in Adj[u]$
      **do if** $v \in Q$ and $w(u, v) < key[v]$
        **then** $key[v] \leftarrow w(u, v)$
          $\pi[v] \leftarrow u$

Handshaking Lemma $\Rightarrow$ $\Theta(|E|)$ implicit DECREASE-KEY's.

Time = $\Theta(|V|)\cdot T_{\text{EXTRACT-MIN}} + \Theta(|E|)\cdot T_{\text{DECREASE-KEY}}$

# Analysis of Prim (continued)

Time = $\Theta(|V|)\cdot T_{\text{EXTRACT-MIN}} + \Theta(|E|)\cdot T_{\text{DECREASE-KEY}}$

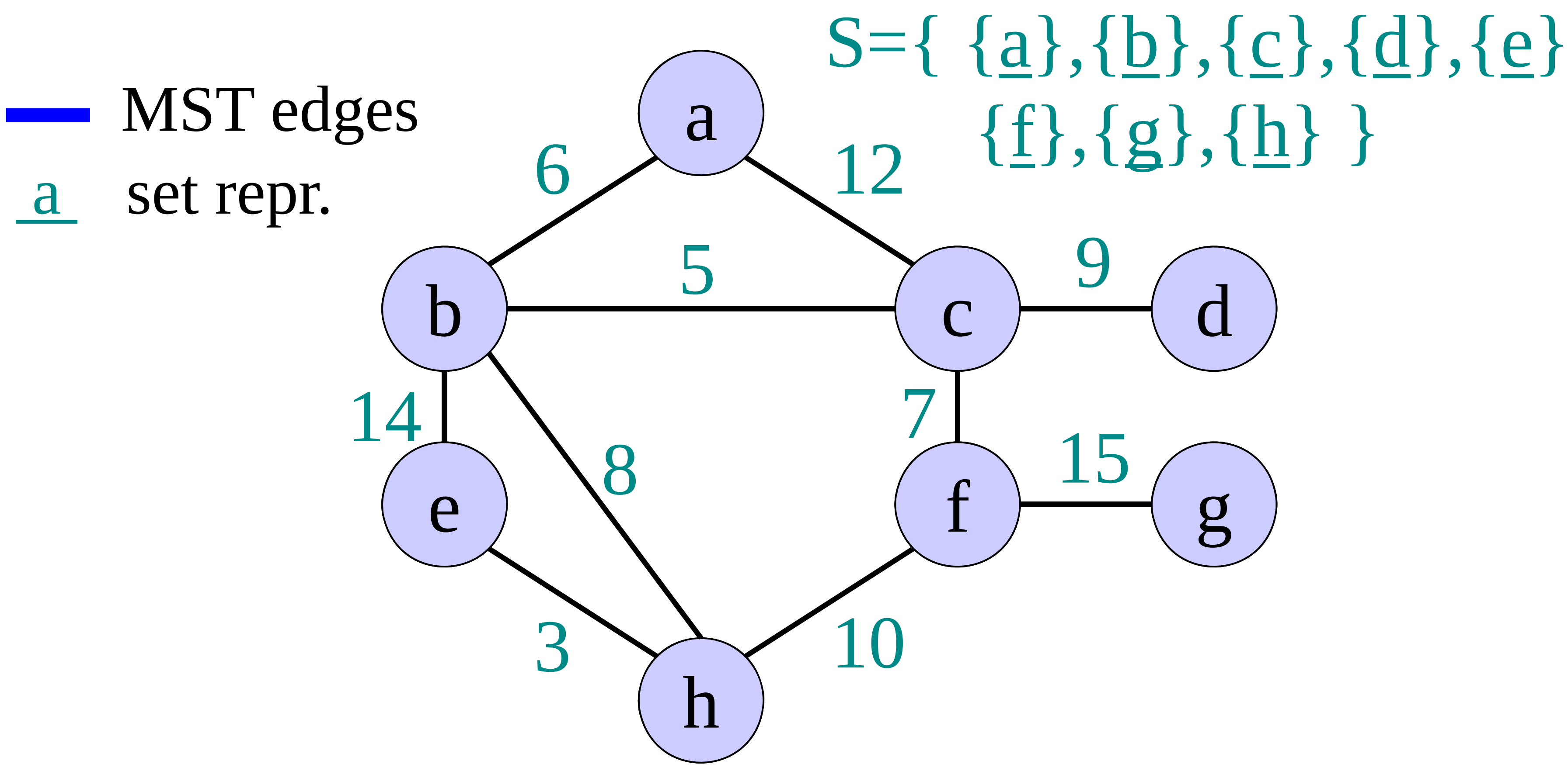
| $Q$ | $T_{\text{EXTRACT-MIN}}$ | $T_{\text{DECREASE-KEY}}$ | Total |
|---|---|---|---|
| array | $O(\lvert V\rvert)$ | $O(1)$ | $O(\lvert V\rvert^2)$ |
| binary heap | $O(\log \lvert V\rvert)$ | $O(\log \lvert V\rvert)$ | $O(\lvert E\rvert \log \lvert V\rvert)$ |
| Fibonacci heap | $O(\log \lvert V\rvert)$ amortized | $O(1)$ amortized | $O(\lvert E\rvert + \lvert V\rvert \log \lvert V\rvert)$ worst case |

# Kruskal's algorithm

**IDEA (again greedy):**
Repeatedly pick edge with smallest weight as long as it does not form a cycle.

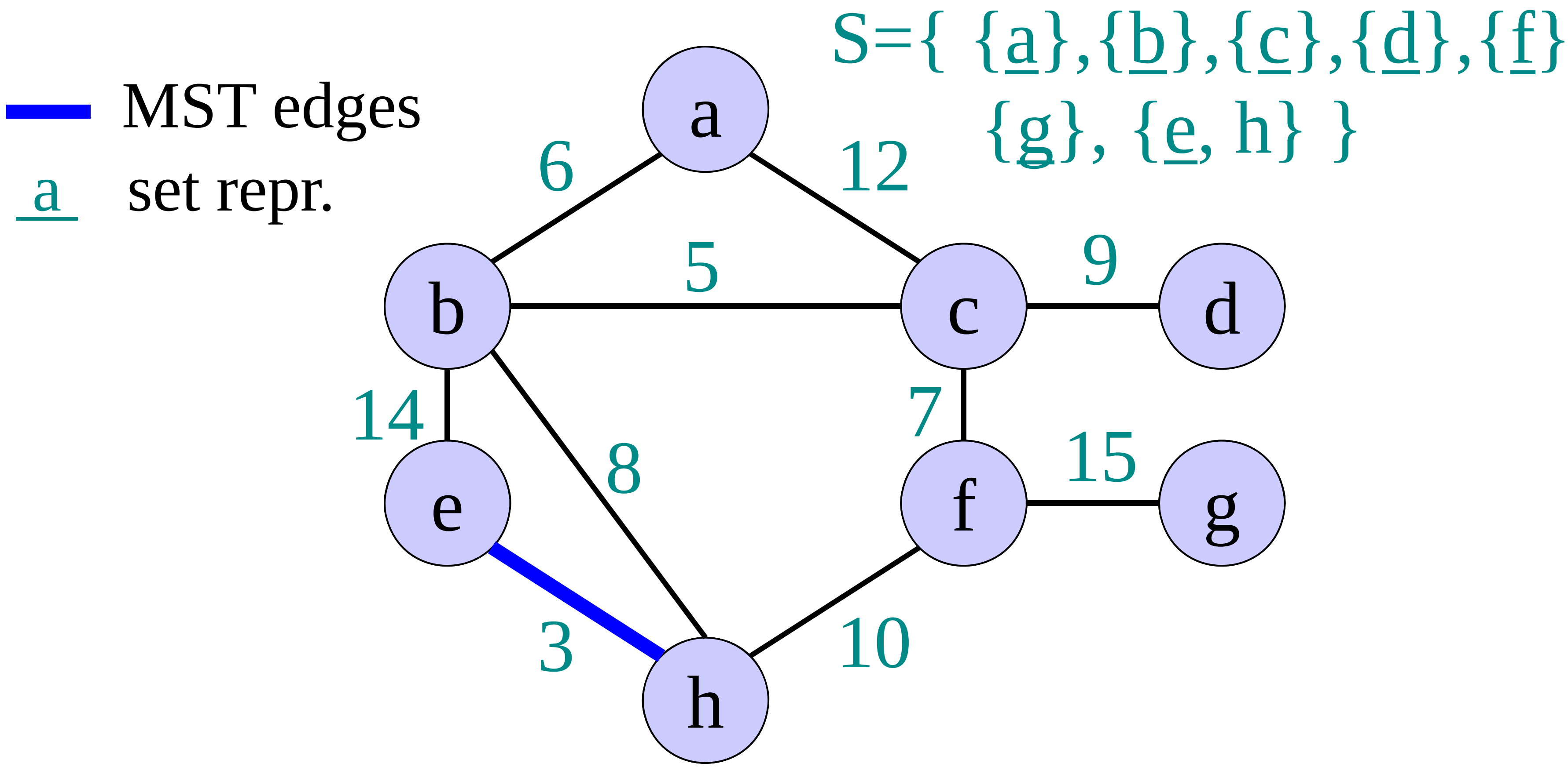
- The algorithm creates a set of trees (a **forest**)
- During the algorithm the added edges merge the trees together, such that in the end only one tree remains

# Example of Kruskal's algorithm




Every node is a single tree.

# Example of Kruskal's algorithm



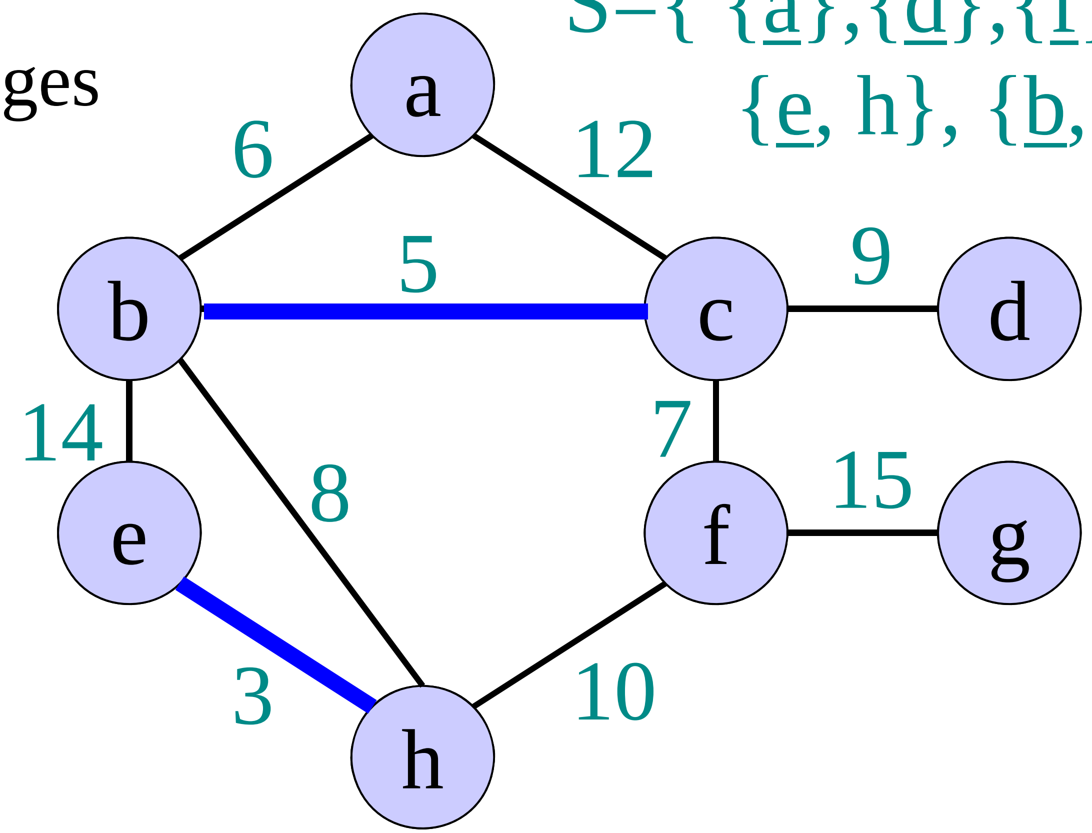

Edge 3 merged two singleton trees.

# Example of Kruskal's algorithm

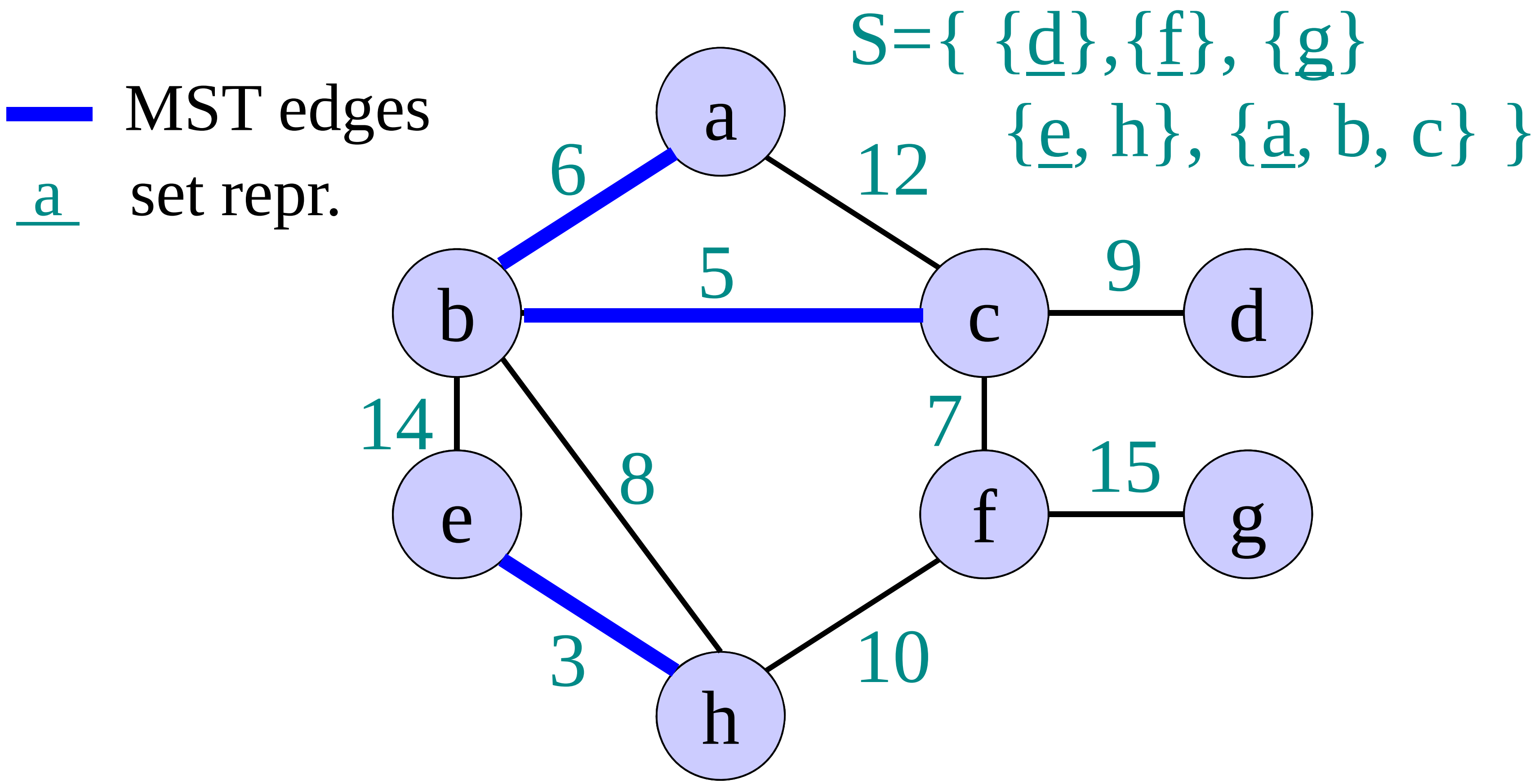
MST edges

a set repr.

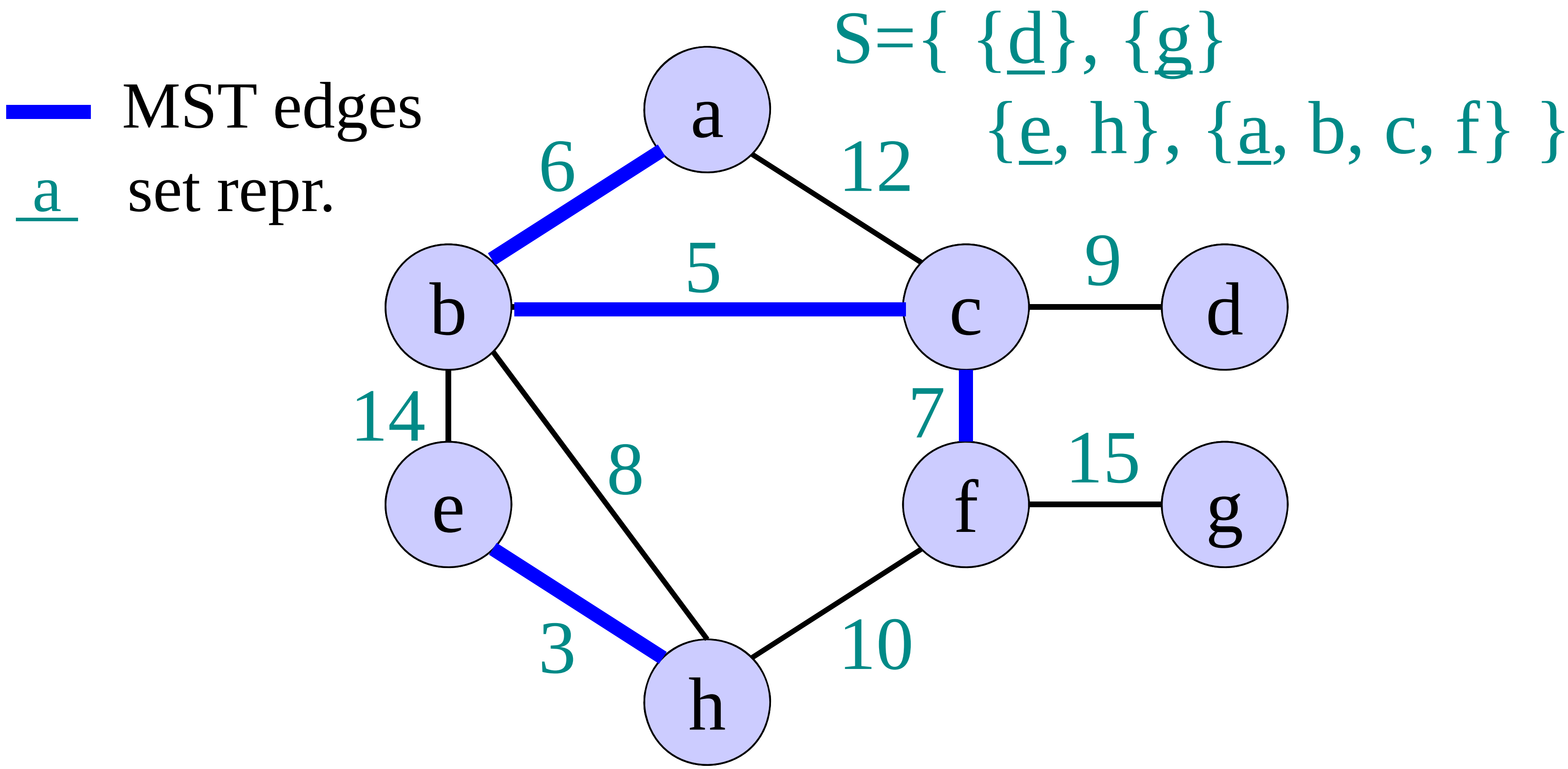
S={ {a},{d},{f}, {g}
{e, h}, {b, c} }



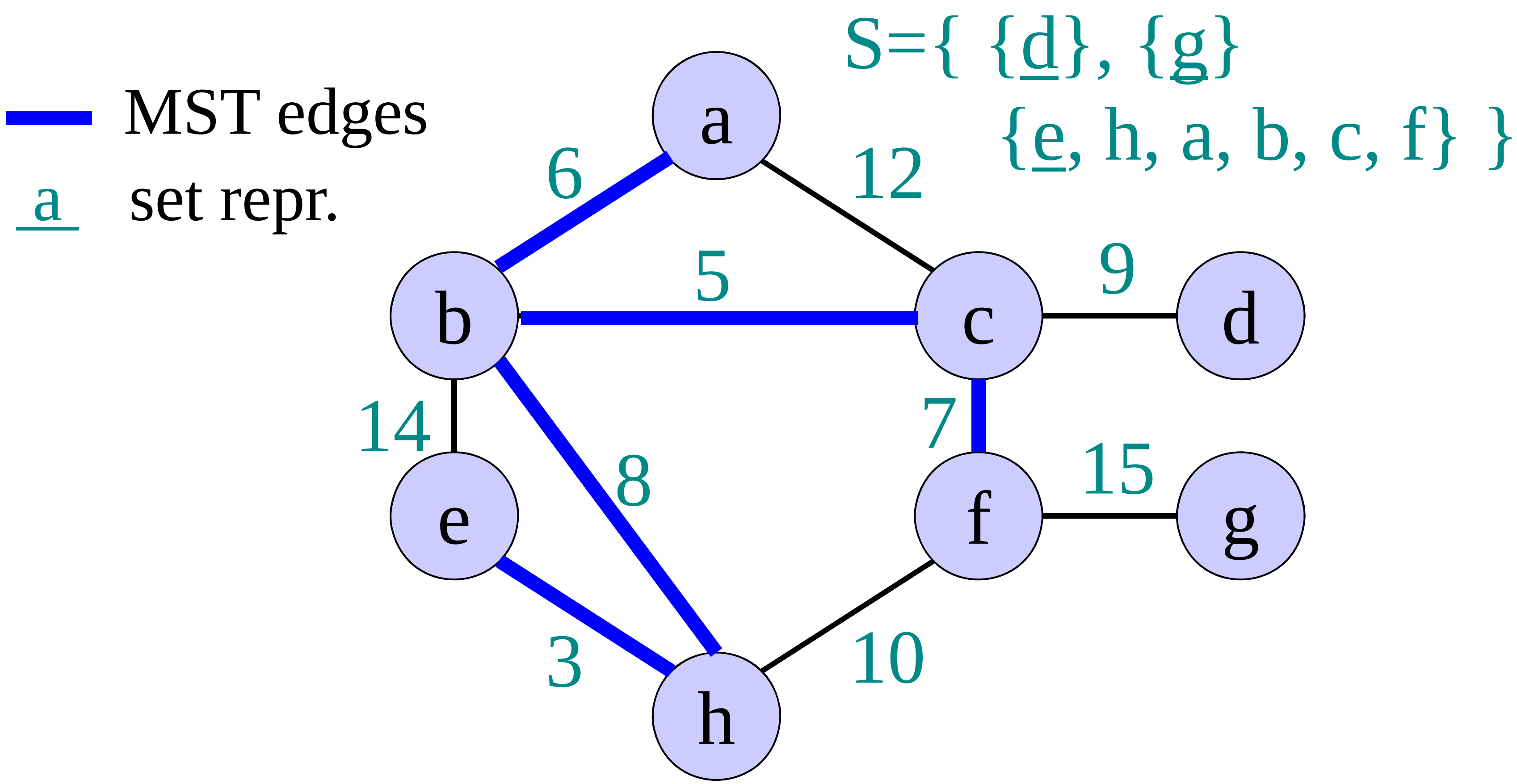
# Example of Kruskal's algorithm



# Example of Kruskal's algorithm

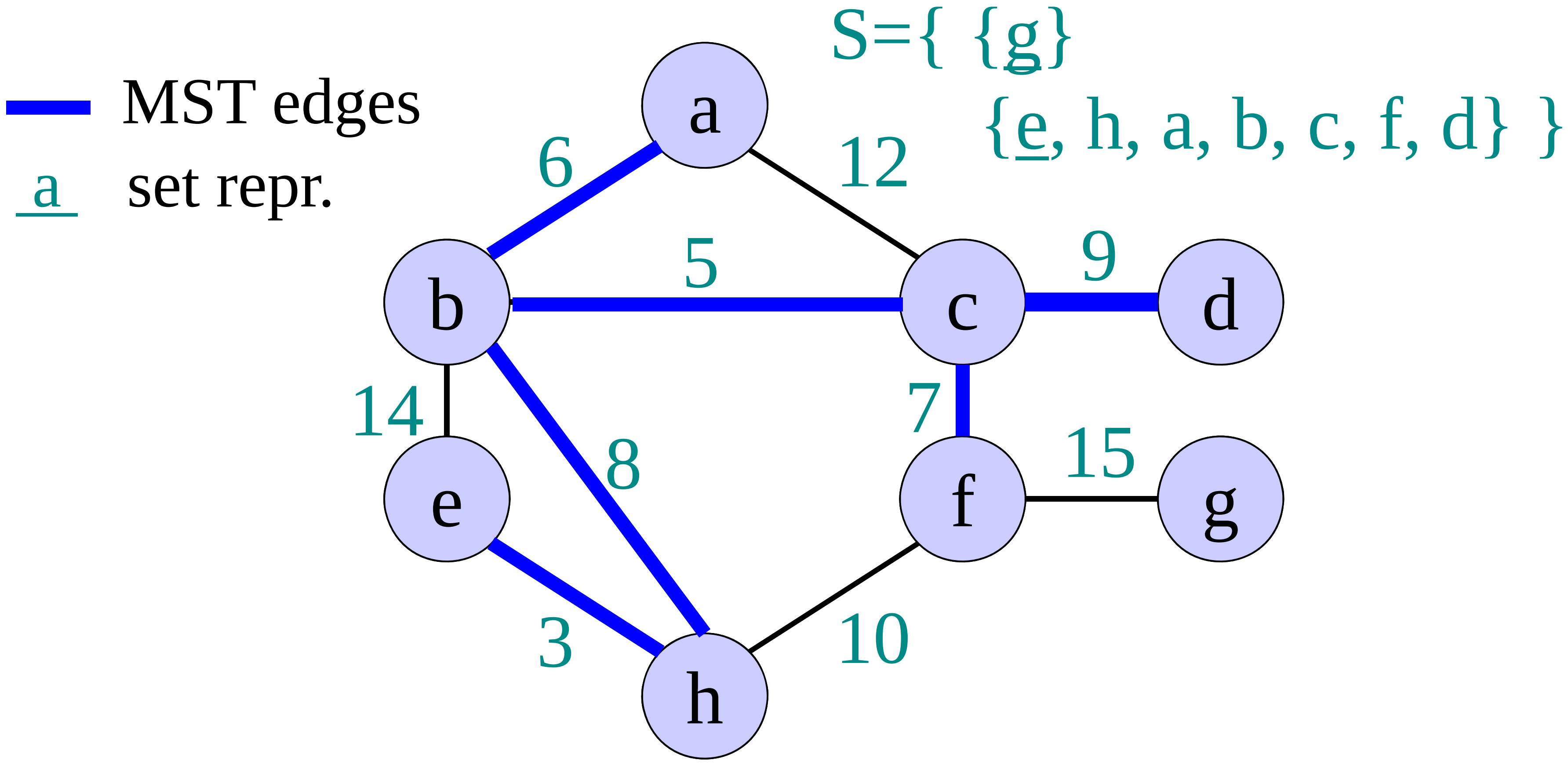


# Example of Kruskal's algorithm

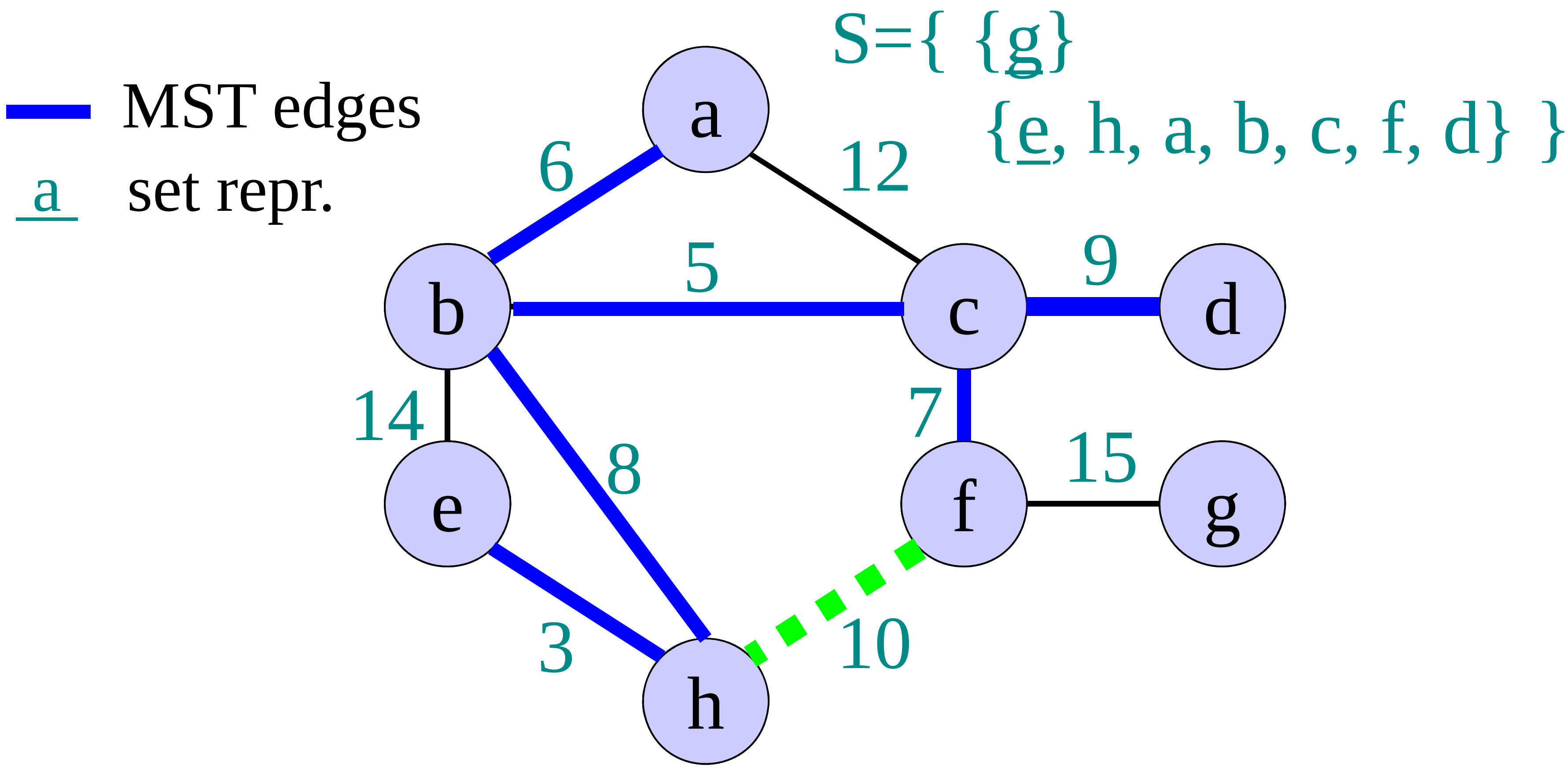



Edge 8 merged the two bigger trees.

# Example of Kruskal's algorithm

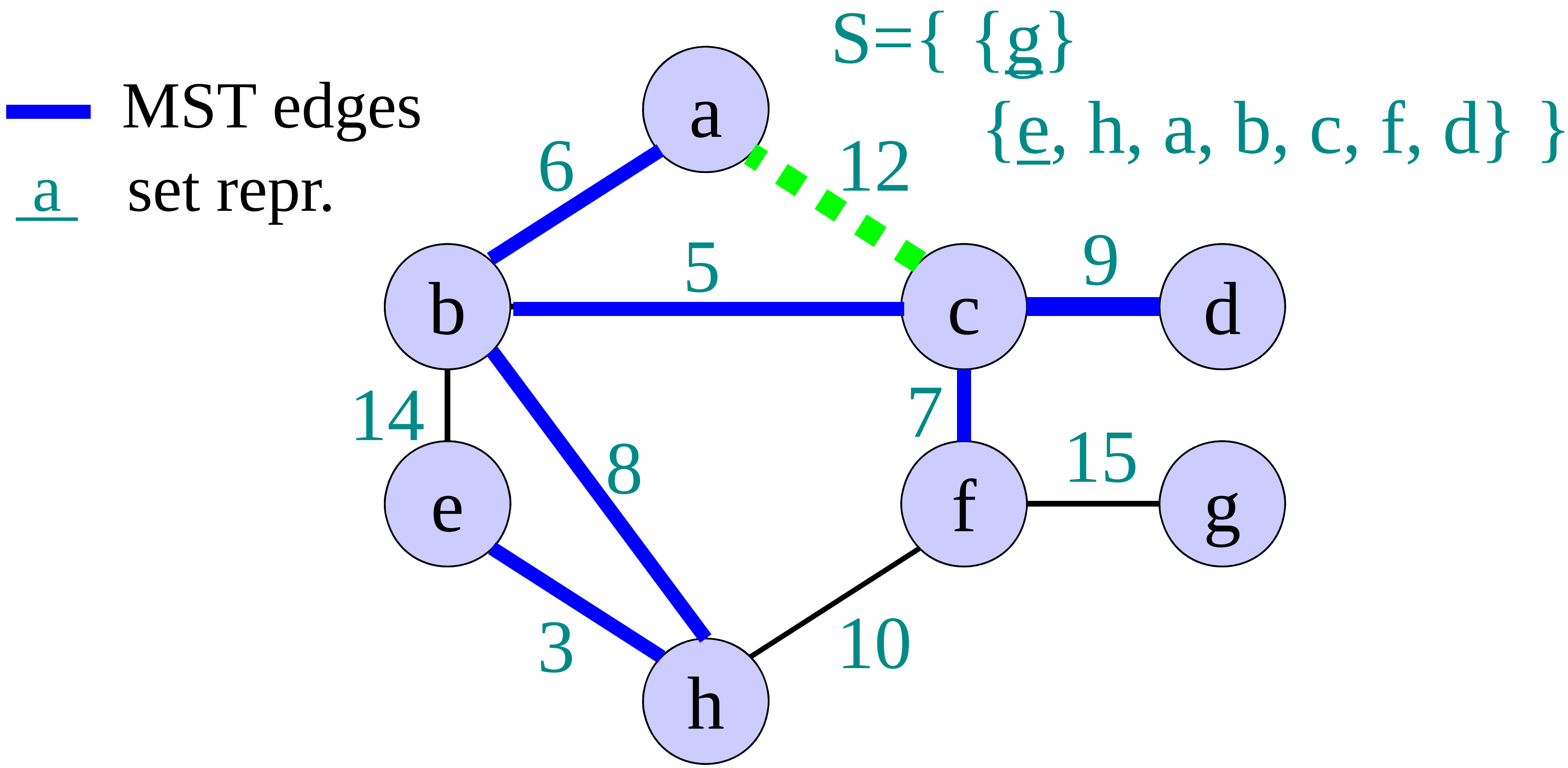


# Example of Kruskal's algorithm



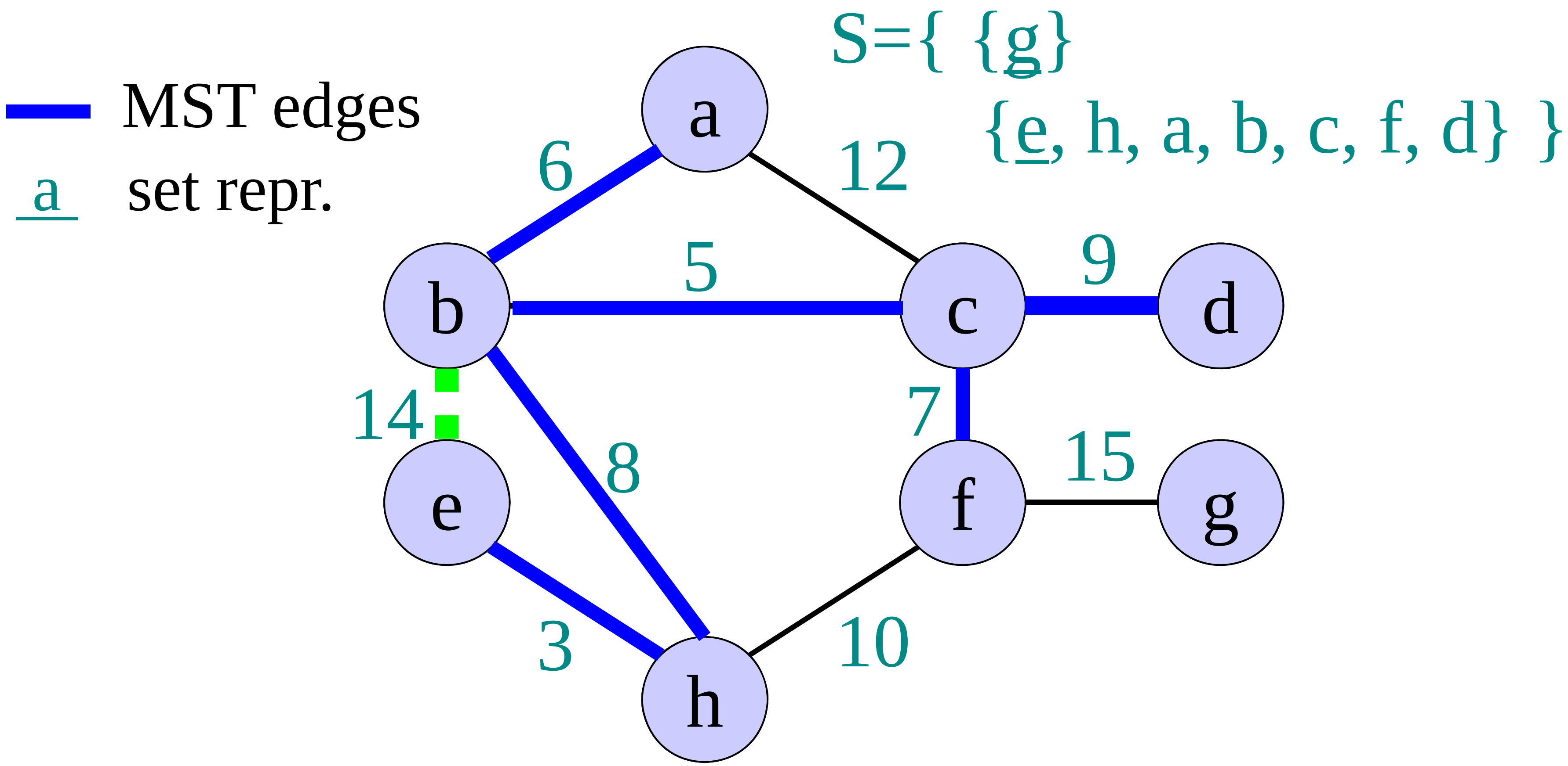

Skip edge 10 as it would cause a cycle.

# Example of Kruskal's algorithm




Skip edge 12 as it would cause a cycle.

# Example of Kruskal's algorithm



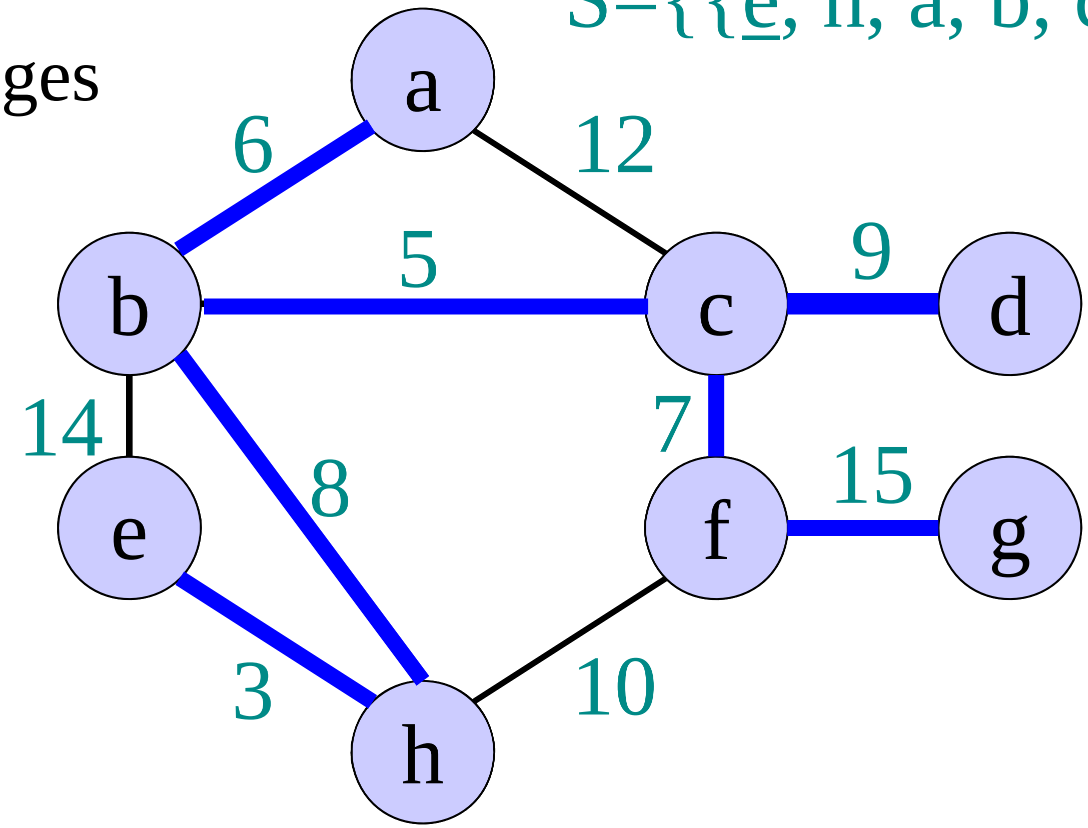

Skip edge 14 as it would cause a cycle.

# Example of Kruskal's algorithm

S={{e, h, a, b, c, f, d, g} }

MST edges

a set repr.



# Kruskal's algorithm

**IDEA (again greedy):**
Repeatedly pick edge with smallest weight as long as it does not form a cycle.

- The algorithm creates a set of trees (a **forest**)
- During the algorithm the added edges merge the trees together, such that in the end only one tree remains

- Correctness: Next edge $e$ connects two components $A_1$, $A_2$. It is the lightest edge which does not produce a cycle, hence it is also the lightest edge between $A_1$ and $V \setminus A_1$ and therefore satisfies the cut property.

# Disjoint-set data structure (Union-Find)

- Maintains a dynamic collection of *pairwise-disjoint* sets $\mathsf{S} = \{S_1, S_2, \ldots, S_r\}$.
- Each set $S_i$ has one element distinguished as the **representative** element.
- Supports operations:

$O(1)$ • MAKE-SET$(x)$: adds new set $\{x\}$ to $\mathsf{S}$

$O(\alpha(n))$ • UNION$(x, y)$: replaces sets $S_x$, $S_y$ with $S_x \cup S_y$

$O(\alpha(n))$ • FIND-SET$(x)$: returns the representative of the set $S_x$ containing element $x$

- $1 < \alpha(n) < \log^*(n) < \log(\log(n)) < \log(n)$

# Union-Find Example

The representative is underlined

| | |
|---|---|
| | $S = \{\}$ |
| MAKE-SET(2) | $S = \{\{\underline{2}\}\}$ |
| MAKE-SET(3) | $S = \{\{\underline{2}\}, \{\underline{3}\}\}$ |
| MAKE-SET(4) | $S = \{\{\underline{2}\}, \{\underline{3}\}, \{\underline{4}\}\}$ |
| FIND-SET(4) = 4 | |
| UNION(2, 4) | $S = \{\{\underline{2}, 4\}, \{\underline{3}\}\}$ |
| FIND-SET(4) = 2 | |
| MAKE-SET(5) | $S = \{\{\underline{2}, 4\}, \{\underline{3}\}, \{\underline{5}\}\}$ |
| UNION(4, 5) | $S = \{\{\underline{2}, 4, 5\}, \{\underline{3}\}\}$ |

# Kruskal's algorithm

**IDEA:** Repeatedly pick edge with smallest weight as long as it does not form a cycle.

$S \leftarrow \emptyset$ ⊳ $S$ will contain all MST edges

$O(|V|)$ **for** each $v \in V$ **do** MAKE-SET($v$)

$O(|E|\log|E|)$ Sort edges of $E$ in non-decreasing order according to $w$

$O(|E|)$ **For** each $(u,v) \in E$ taken in this order **do**

$O(\alpha(|V|))$ {
- **if** FIND-SET($u$) $\neq$ FIND-SET($v$) ⊳ $u,v$ in different trees
  - $S \leftarrow S \cup \{(u,v)\}$
  - UNION($u,v$) ⊳ Edge $(u,v)$ connects the two trees

**Runtime:** $O(|V|+|E|\log|E|+|E|\alpha(|V|)) = O(|E| \log |E|)$

# MST algorithms

- Prim's algorithm:
  - Maintains one tree
  - Runs in time $O(|E| \log |V|)$, with binary heaps.
- Kruskal's algorithm:
  - Maintains a forest and uses the disjoint-set data structure
  - Runs in time $O(|E| \log |E|)$
- Best to date: Randomized algorithm by Karger, Klein, Tarjan [1993]. Runs in expected time $O(|V| + |E|)$