CMPS 2200 – Fall 17

Minimum Spanning Trees

Carola Wenk

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

Minimum spanning trees

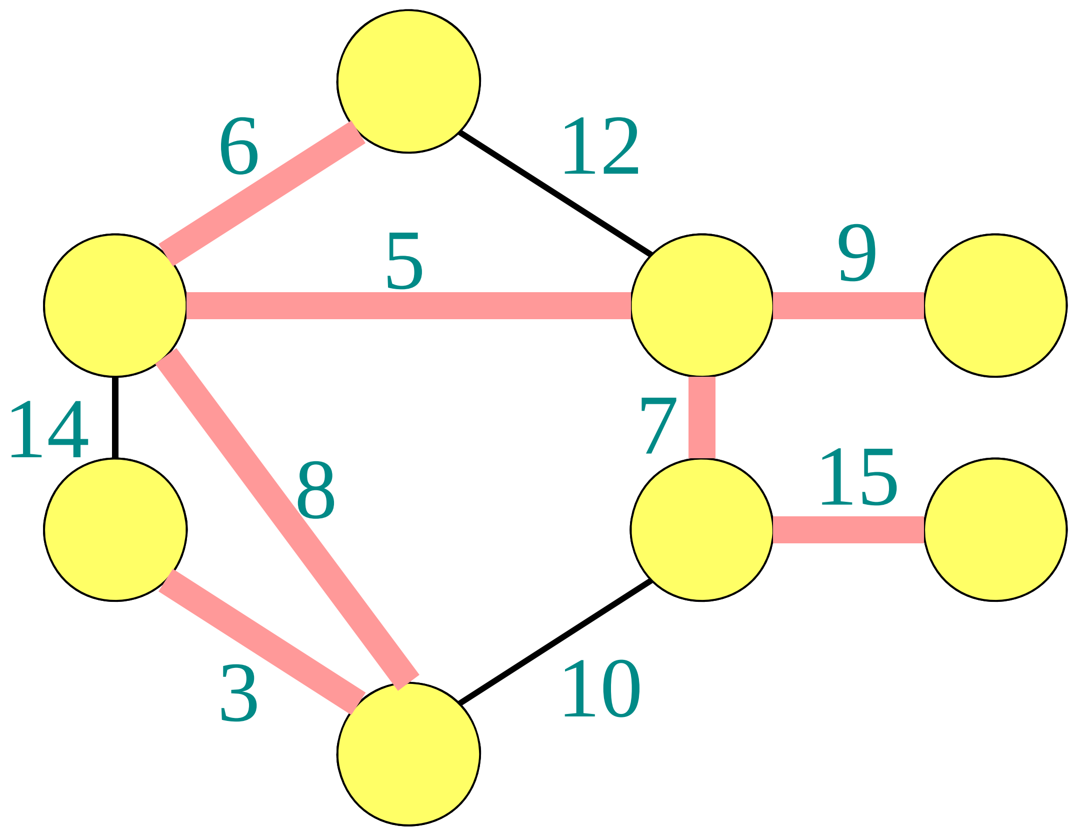
Input: A connected, undirected graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$.

- 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



Hallmark for “greedy” algorithms

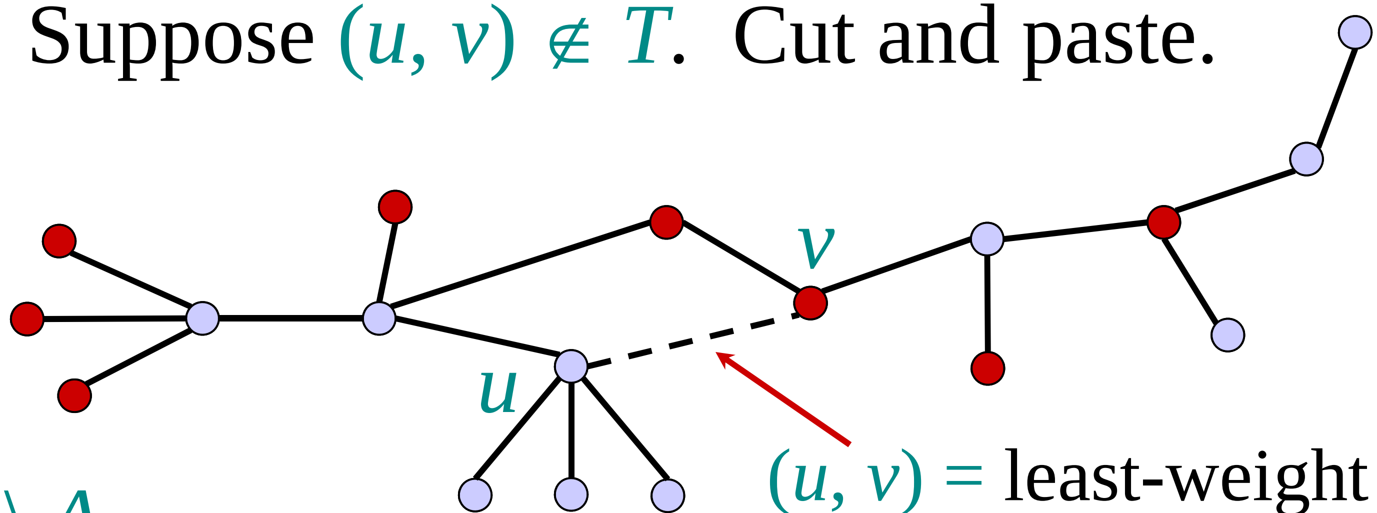
Greedy-choice property
A locally optimal choice is globally optimal.

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:



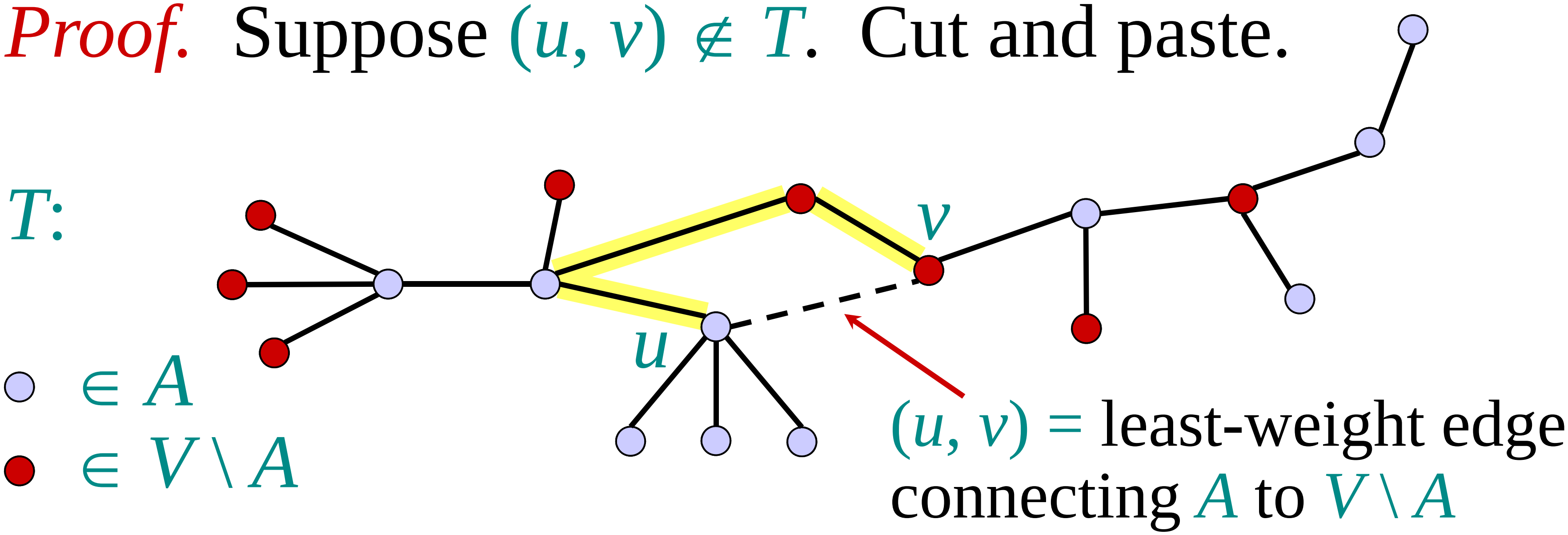
$\circ \in A$

$\bullet \in V \setminus A$

(u, v) = least-weight edge connecting A to $V \setminus A$

Proof of theorem

Proof. Suppose $(u, v) \notin T$. Cut and paste.

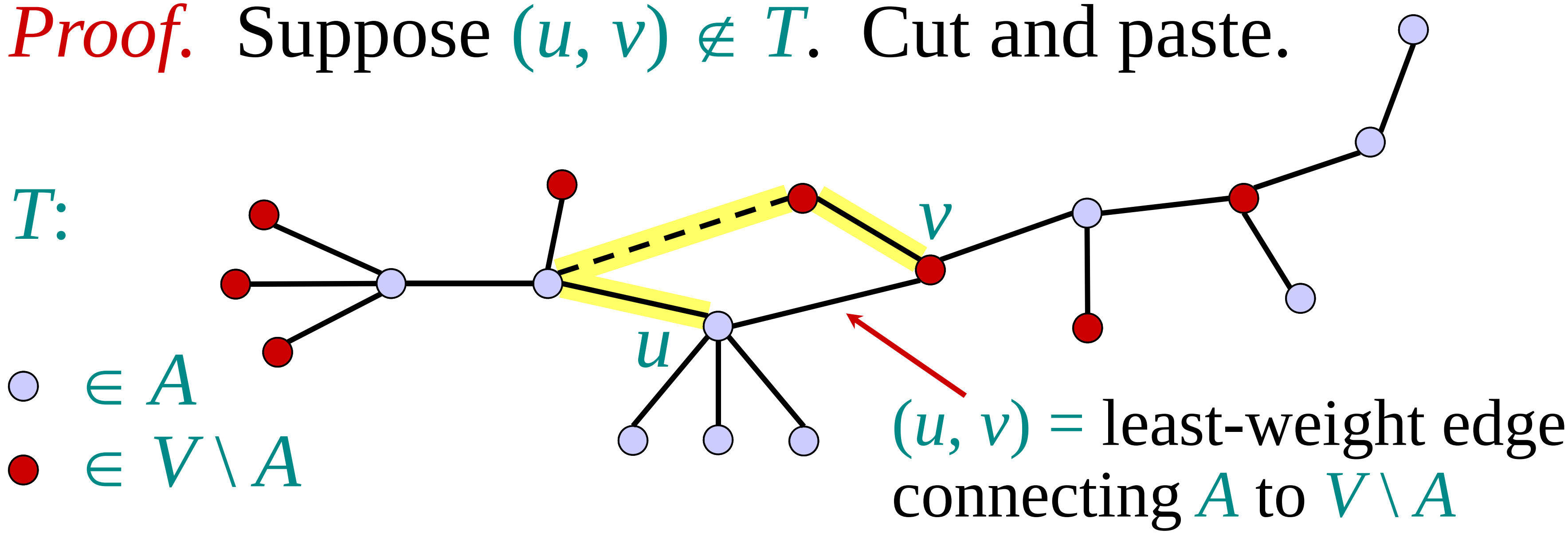


Consider the unique simple path from u to v in T.

Proof of theorem

Proof. Suppose $(u, v) \notin T$. Cut and paste.

T:

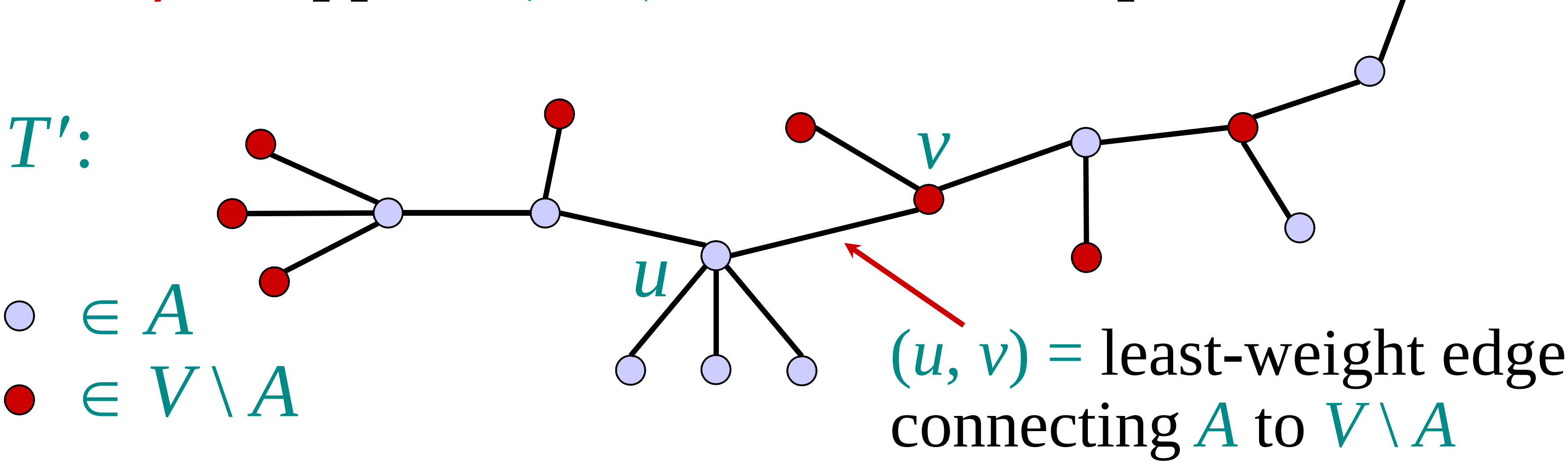


Consider the unique simple path from u to v in T.

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.



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. ☐

Prim's algorithm

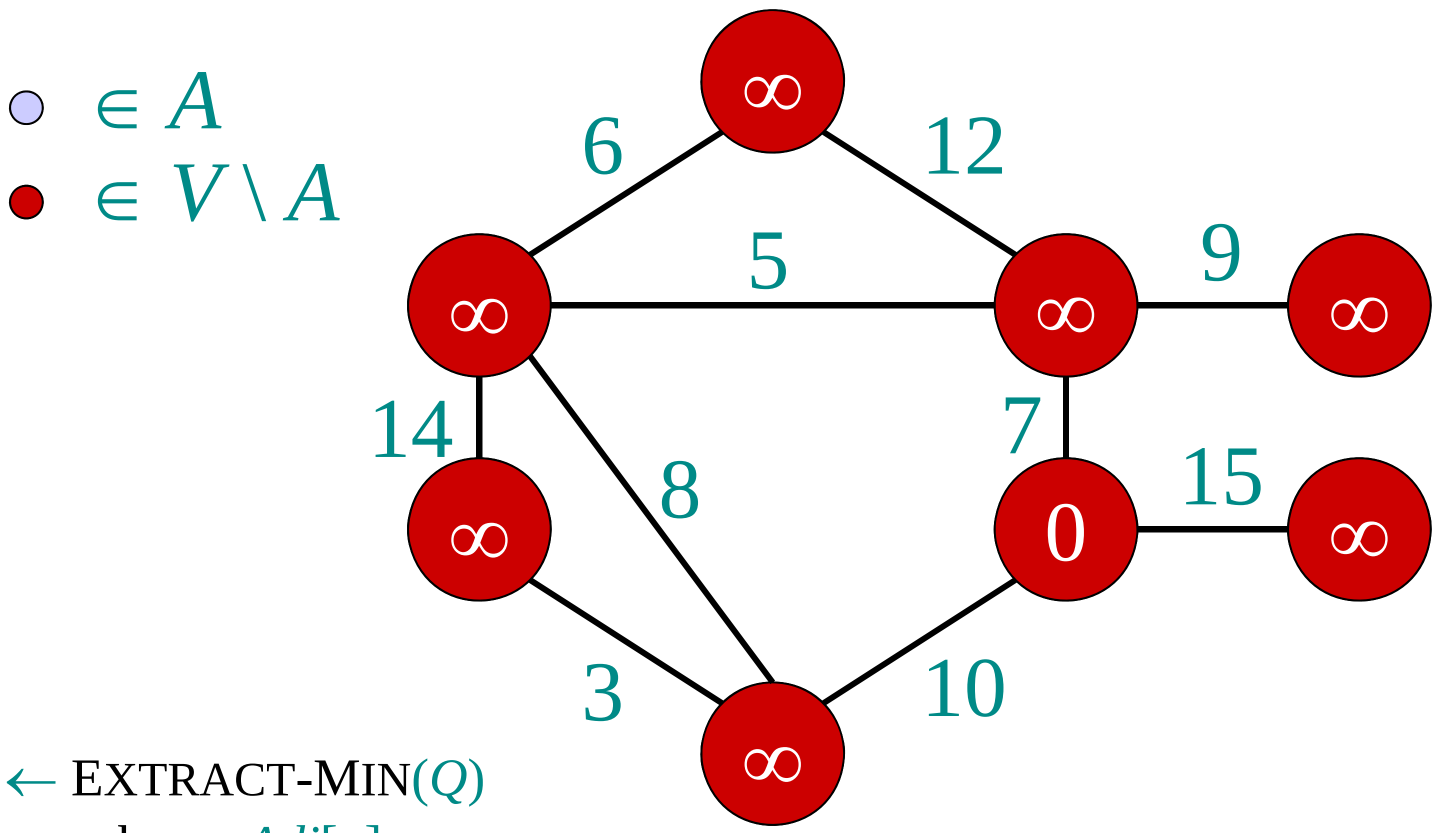
IDEA: Maintain $V \setminus A$ as a priority queue Q. Key each vertex in Q with the weight of the least-weight edge connecting it to a vertex in A.

$Q \leftarrow V$
$key[v] \leftarrow \infty$ 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**
      $d[v] \leftarrow d[u] + w(u, v)$

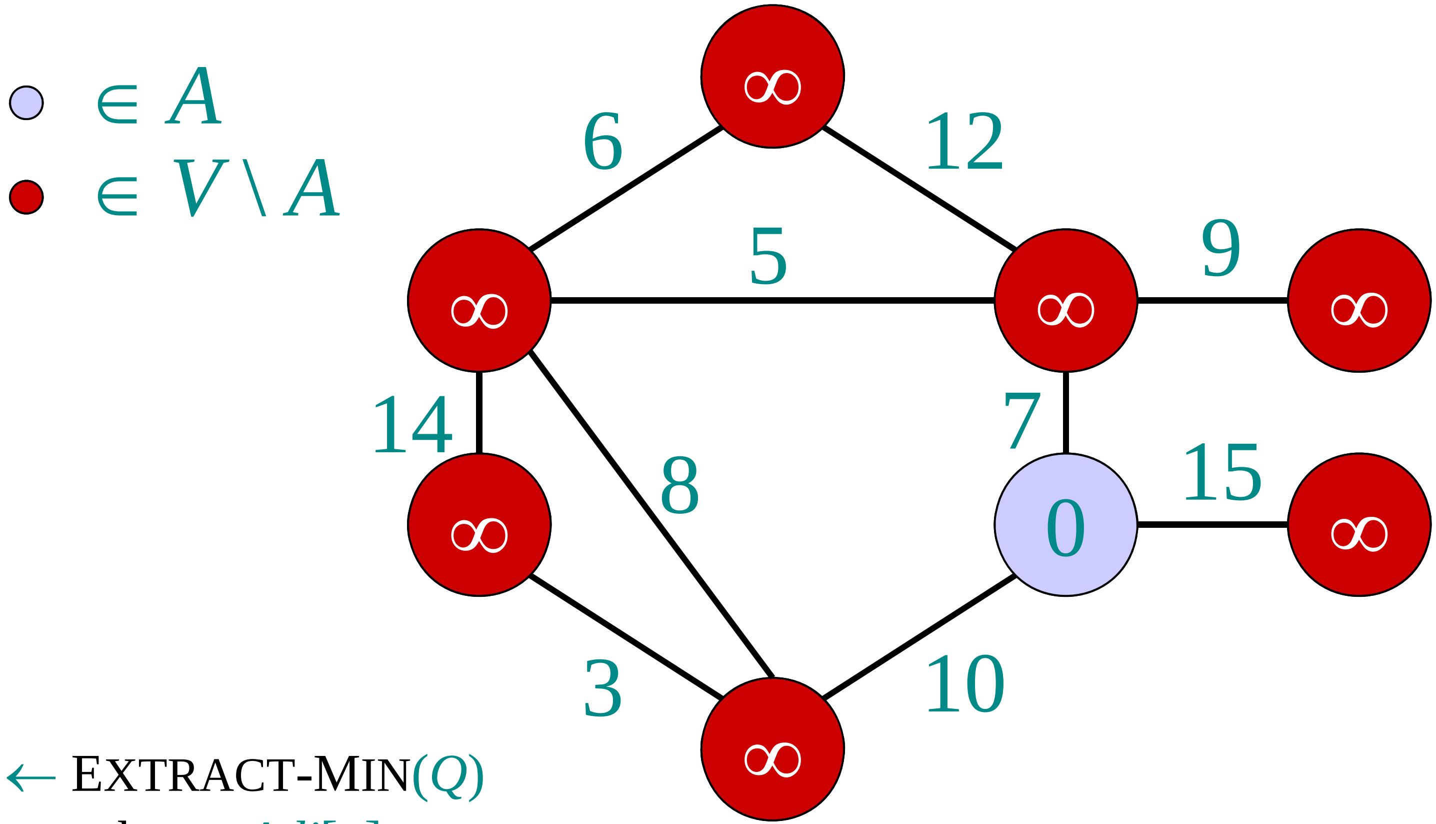
At the end, $\{(v, \pi[v])\}$ forms the MST edges.

Example of Prim's algorithm



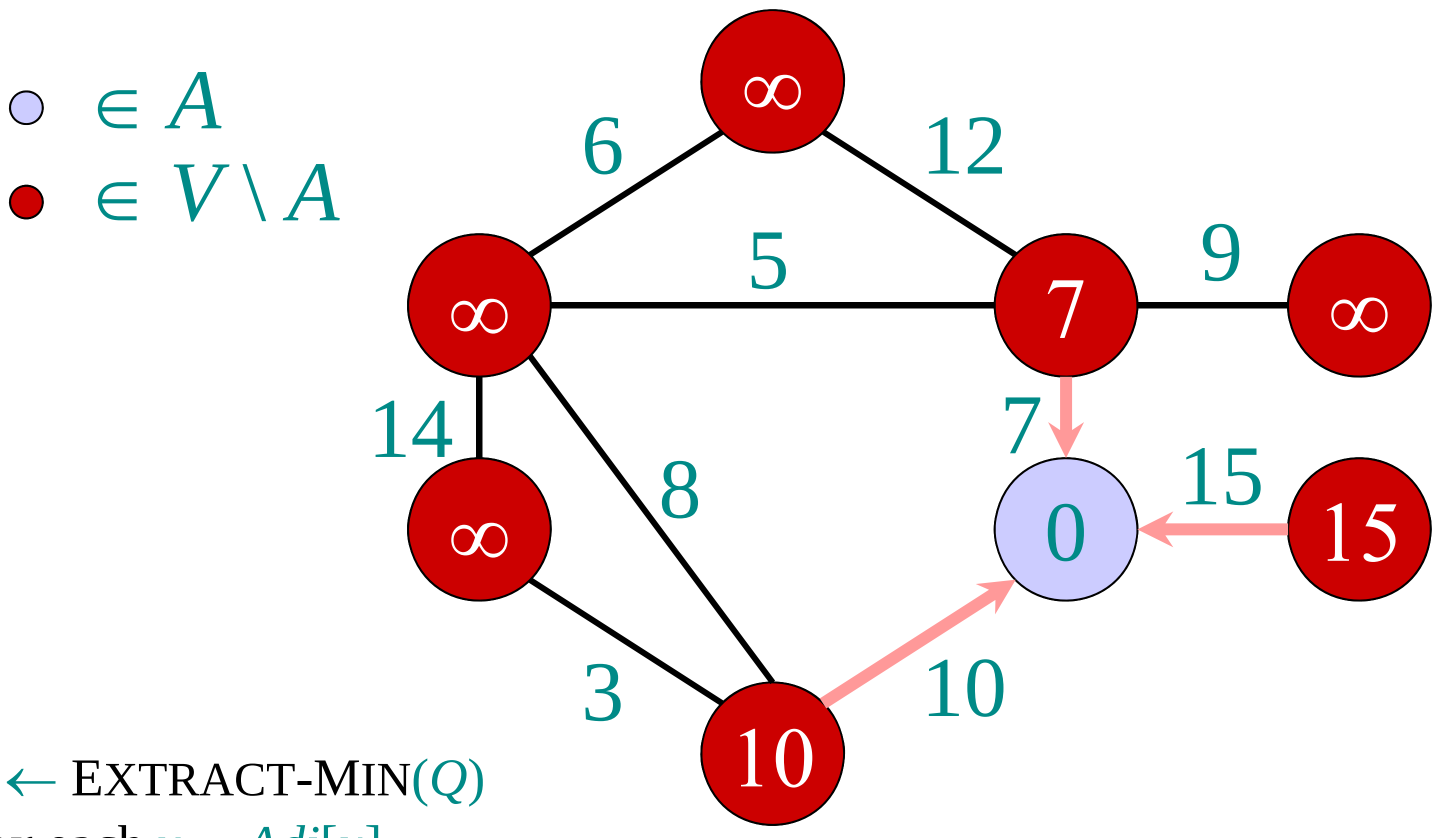
$u \leftarrow$ EXTRACT-MIN(Q)
for each $v \in Adj[u]$
do if $v \in Q$ and $w(u, v) < key[v]$
then $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
$\pi[v] \leftarrow u$

Example of Prim's algorithm



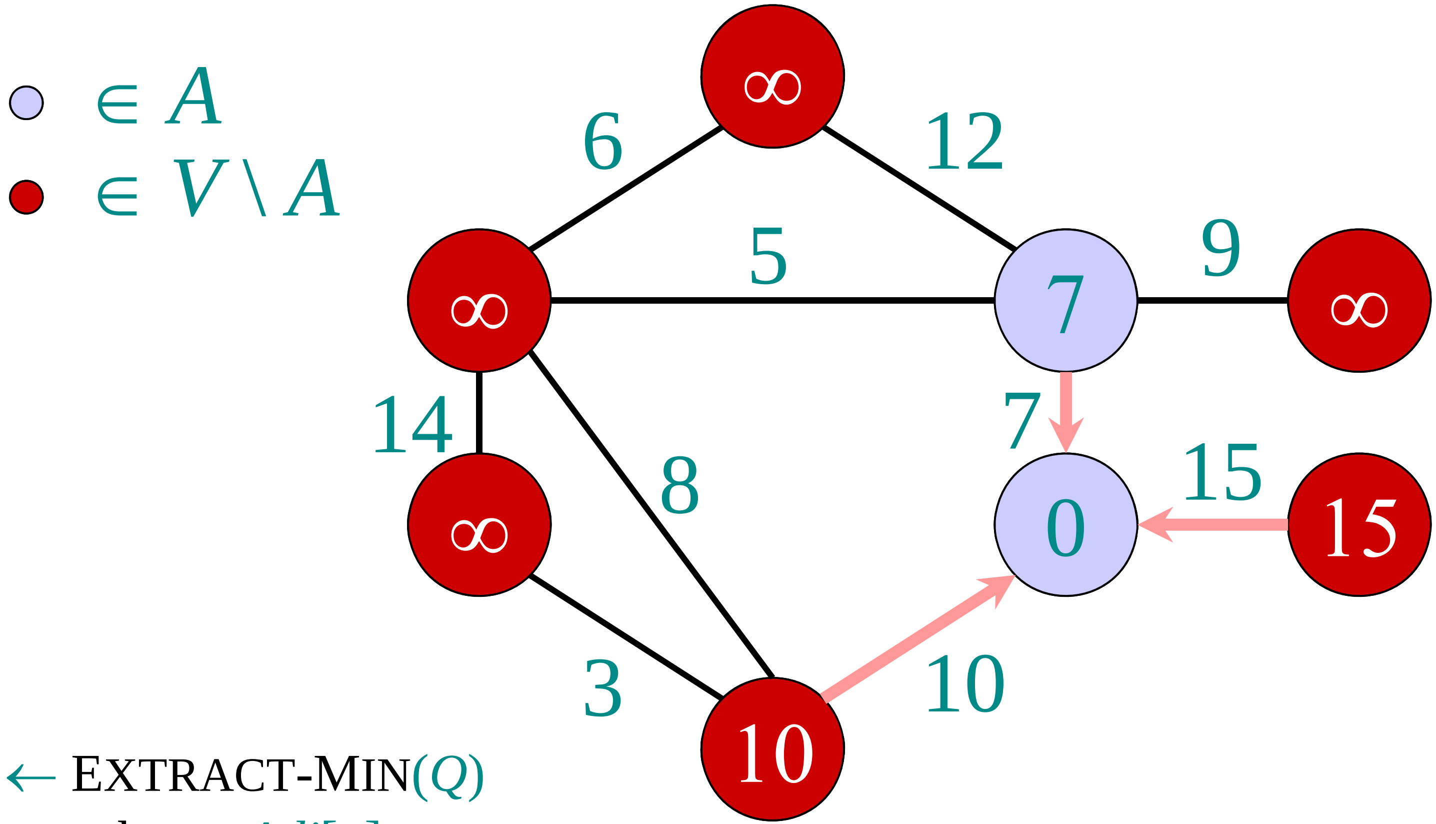
$u \leftarrow$ EXTRACT-MIN(Q)
for each $v \in Adj[u]$
    do if $v \in Q$ and $w(u, v) < key[v]$
        then $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
            $\pi[v] \leftarrow u$

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)

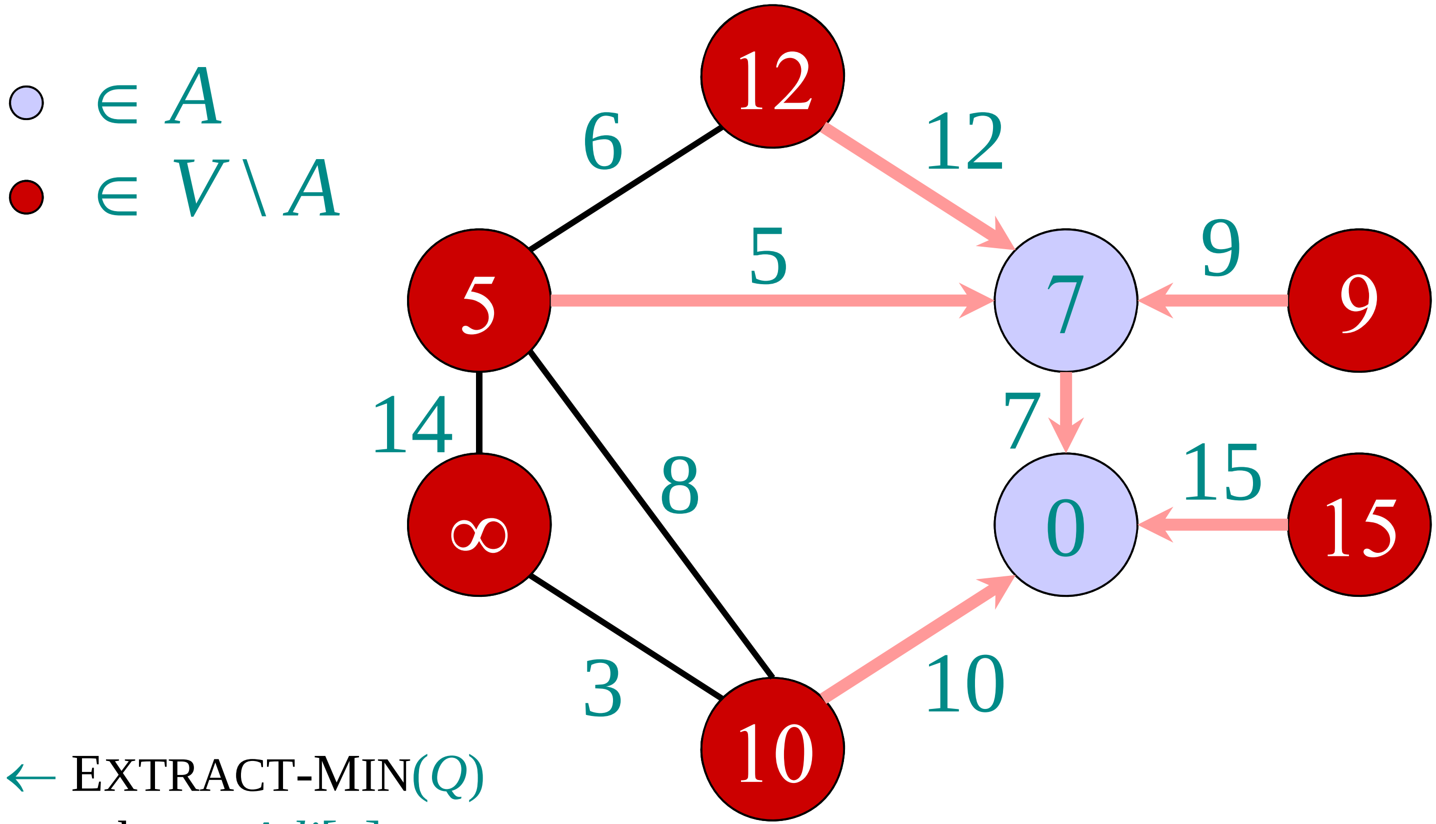
for each $v \in Adj[u]$

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

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

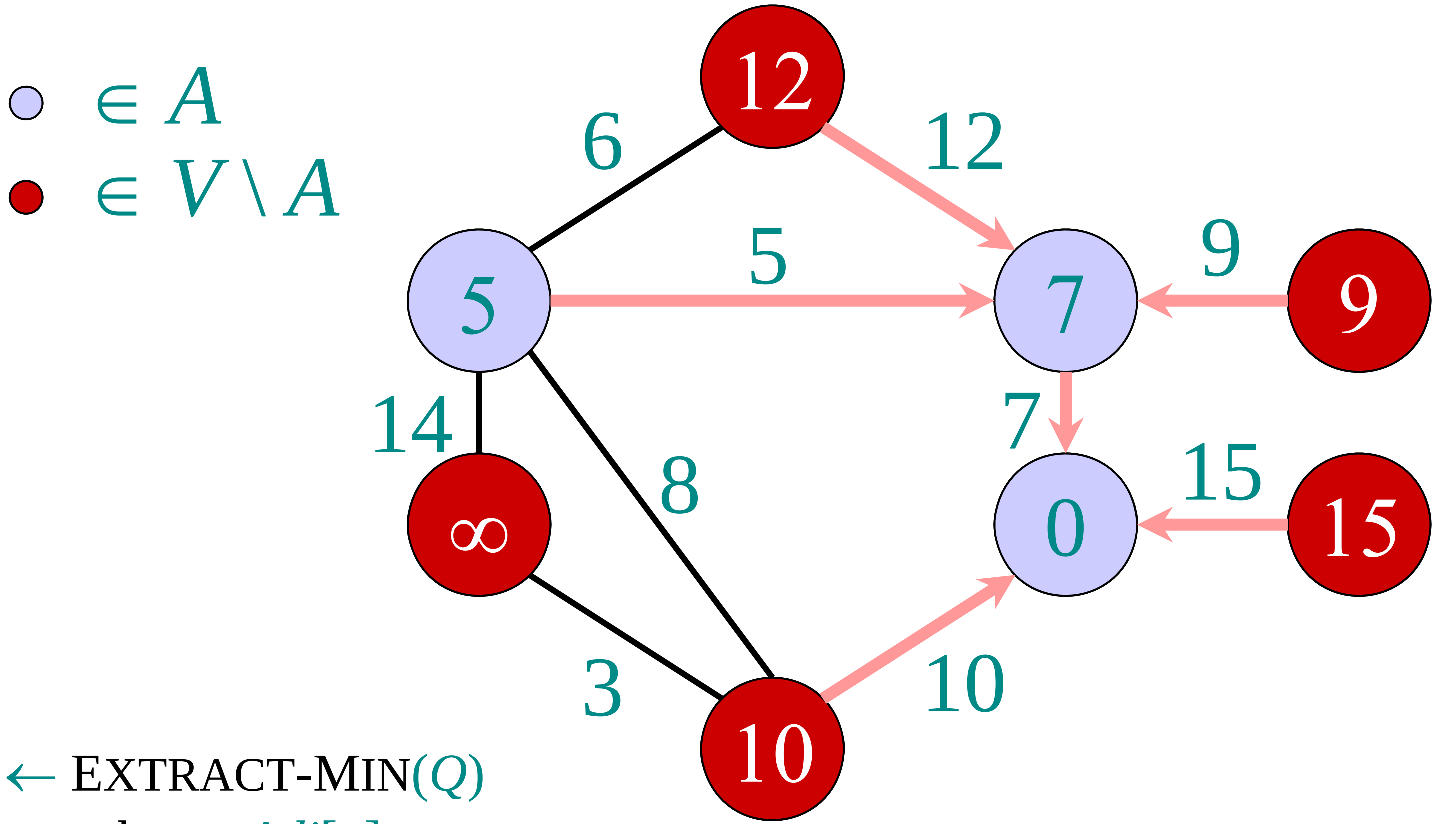
$\pi[v] \leftarrow u$

Example of Prim's algorithm



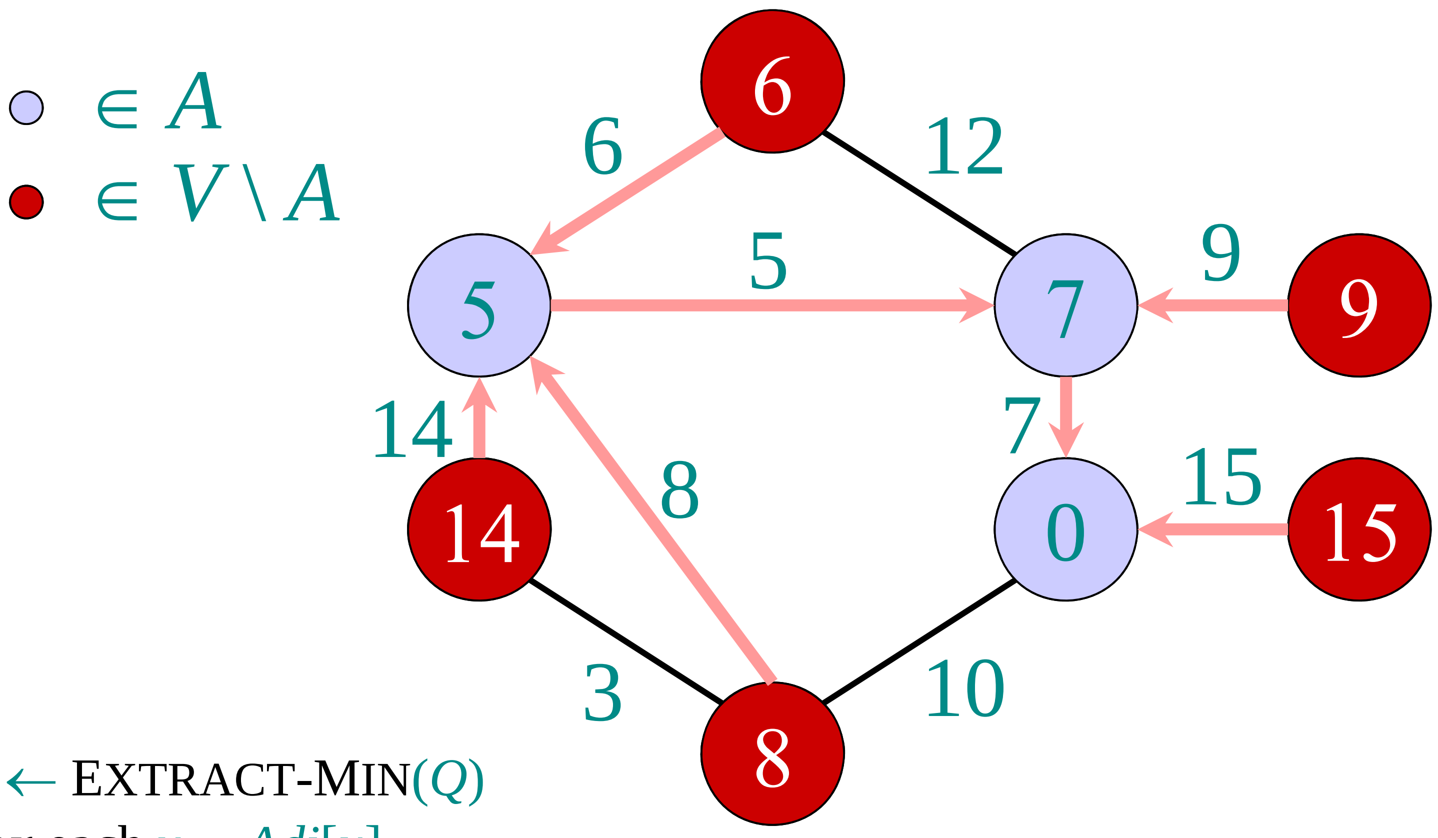
$u \leftarrow$ EXTRACT-MIN(Q)
for each $v \in Adj[u]$
    do if $v \in Q$ and $w(u, v) < key[v]$
        then $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
            $\pi[v] \leftarrow u$

Example of Prim's algorithm



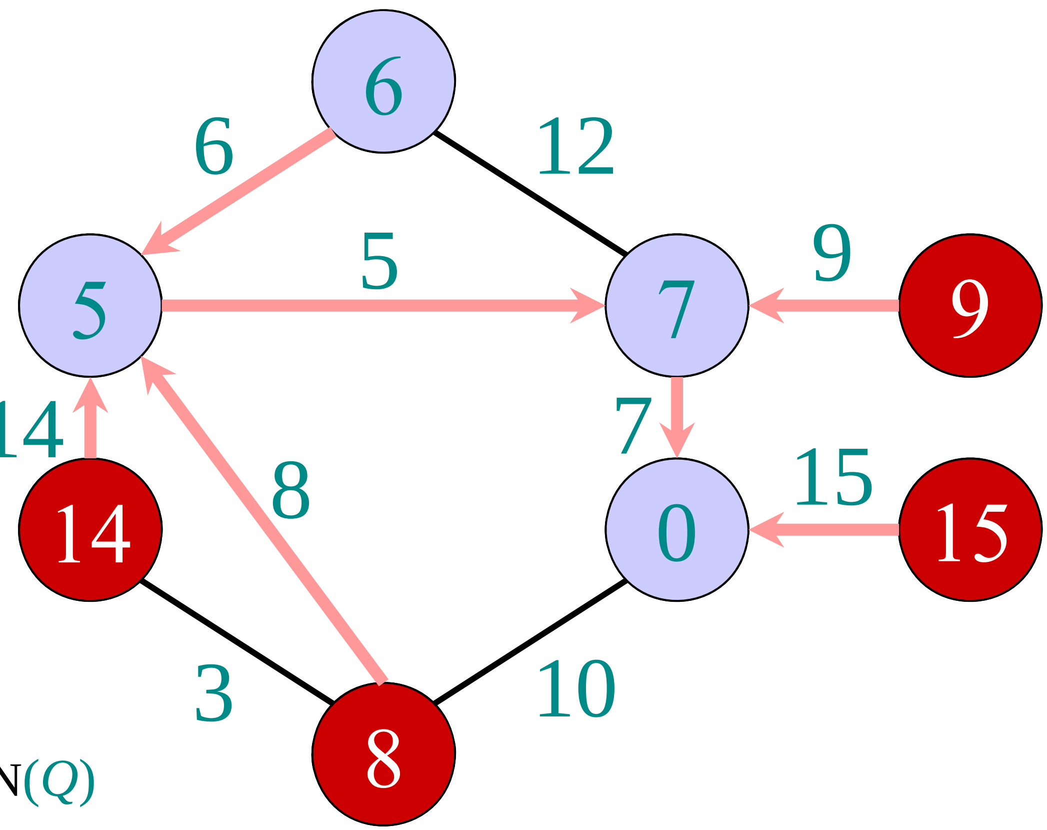
$u \leftarrow$ EXTRACT-MIN(Q)
for each $v \in Adj[u]$
    do if $v \in Q$ and $w(u, v) < key[v]$
        then $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
            $\pi[v] \leftarrow u$

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)

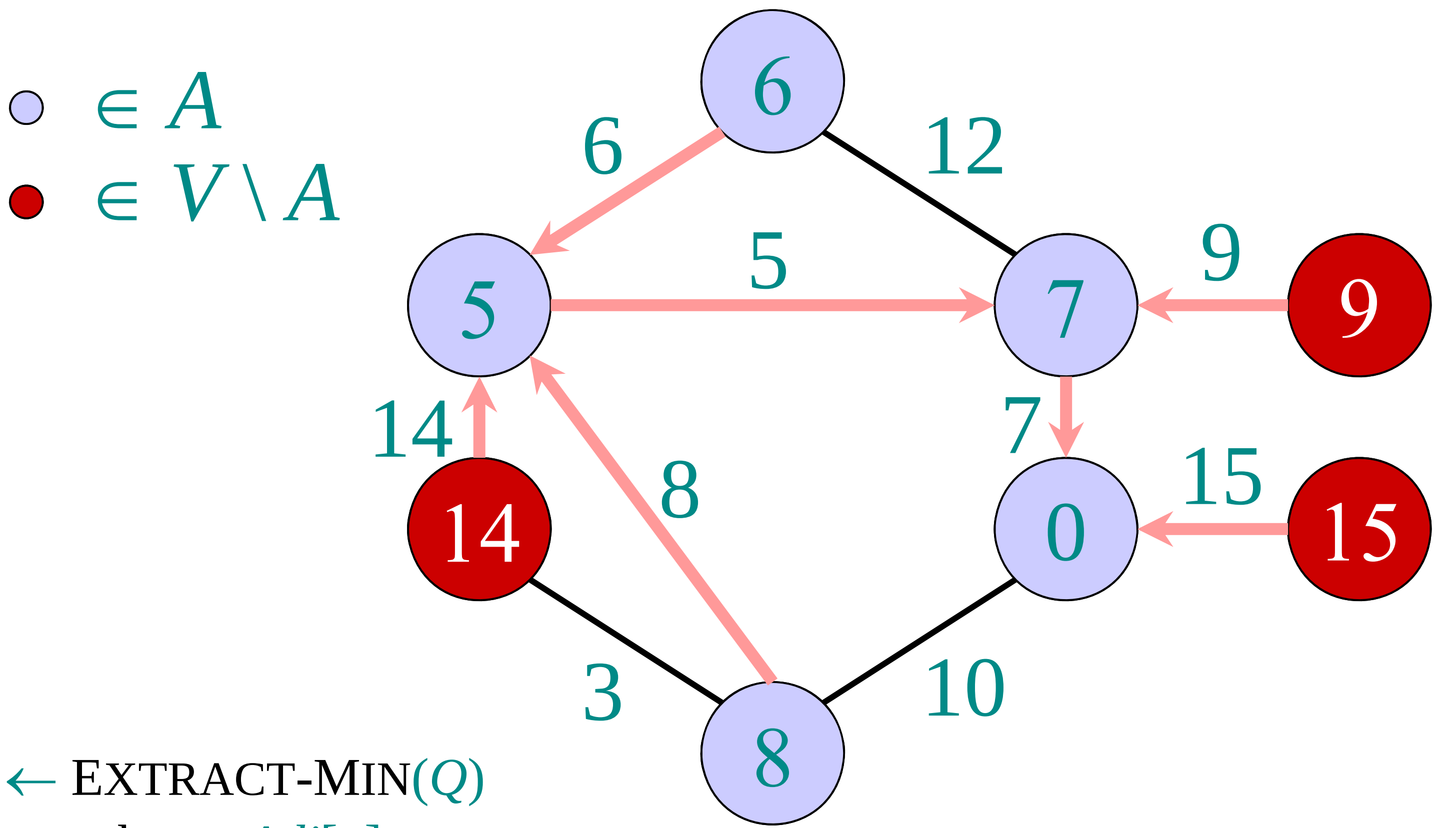
for each $v \in Adj[u]$

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

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

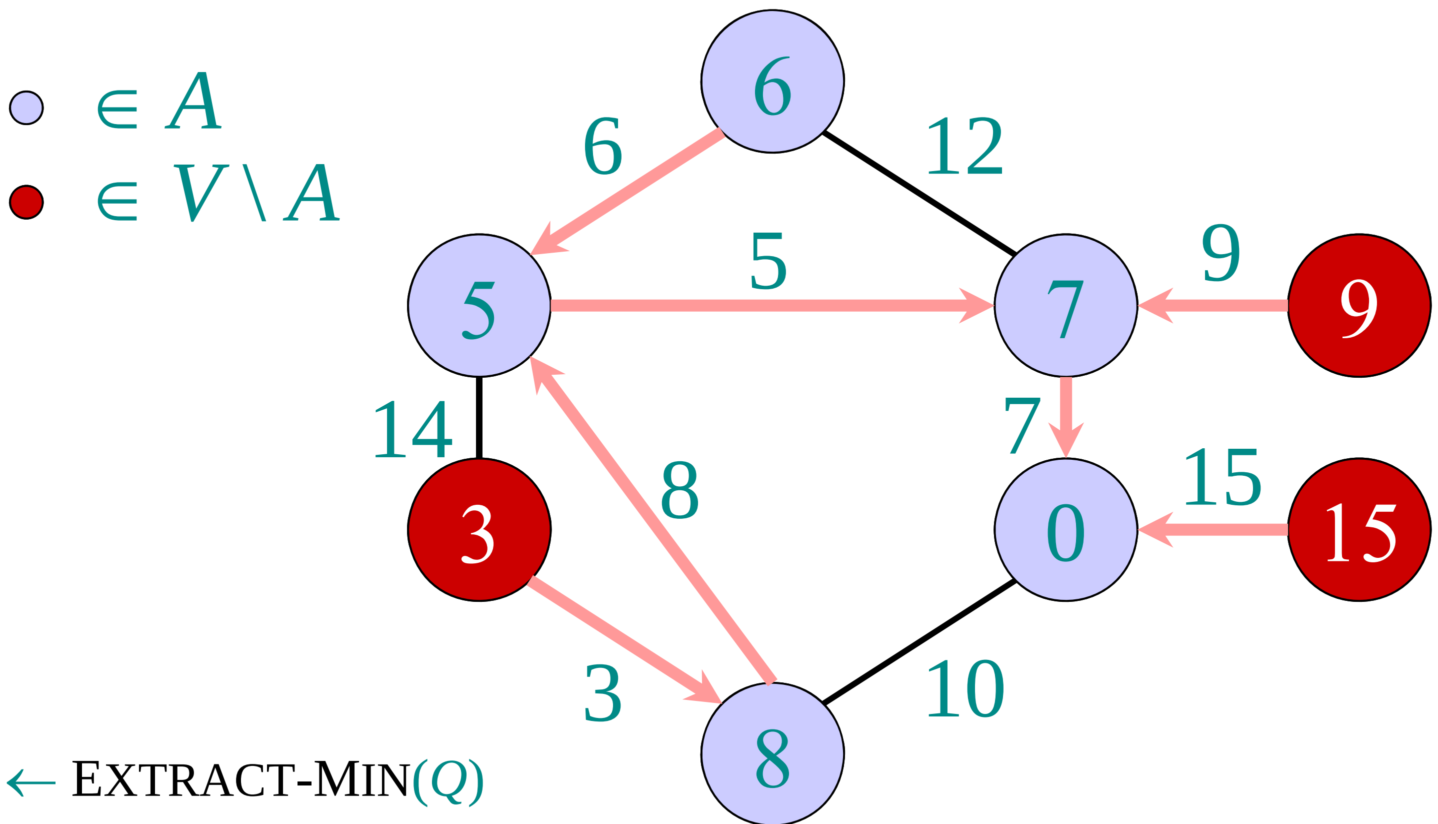
      $\pi[v] \leftarrow u$

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)

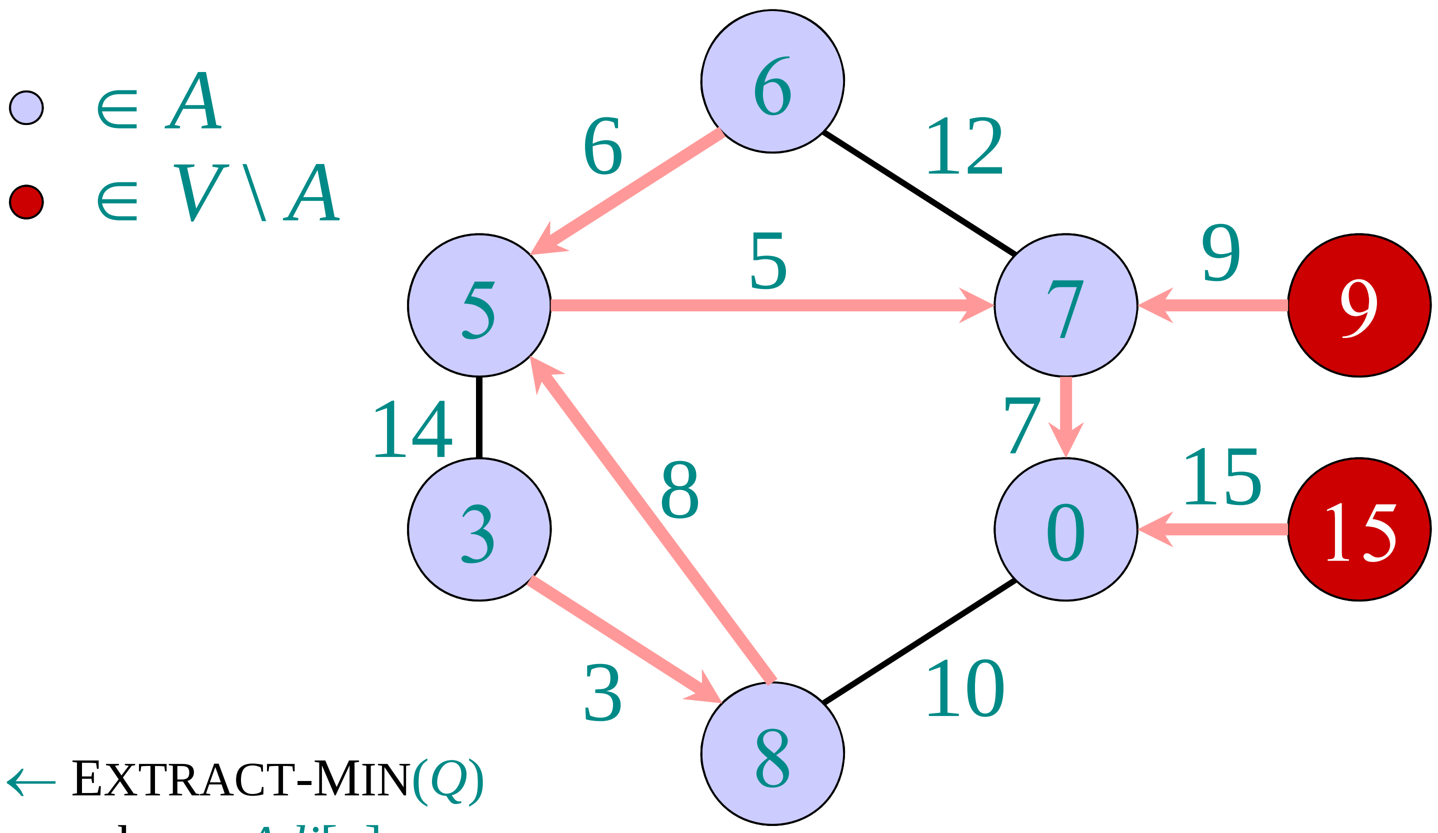
for each $v \in Adj[u]$

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

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

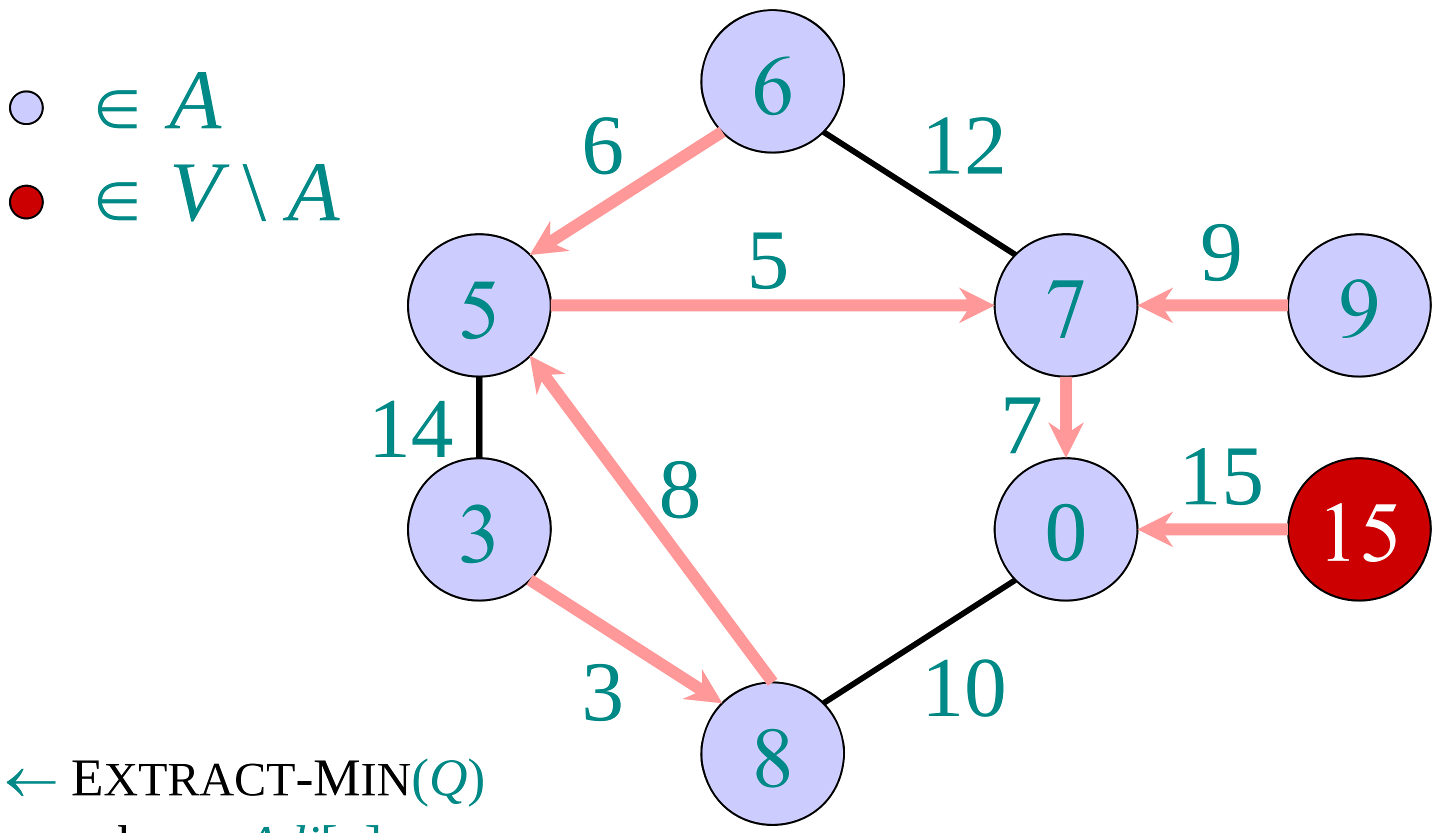
$\pi[v] \leftarrow u$

Example of Prim's algorithm



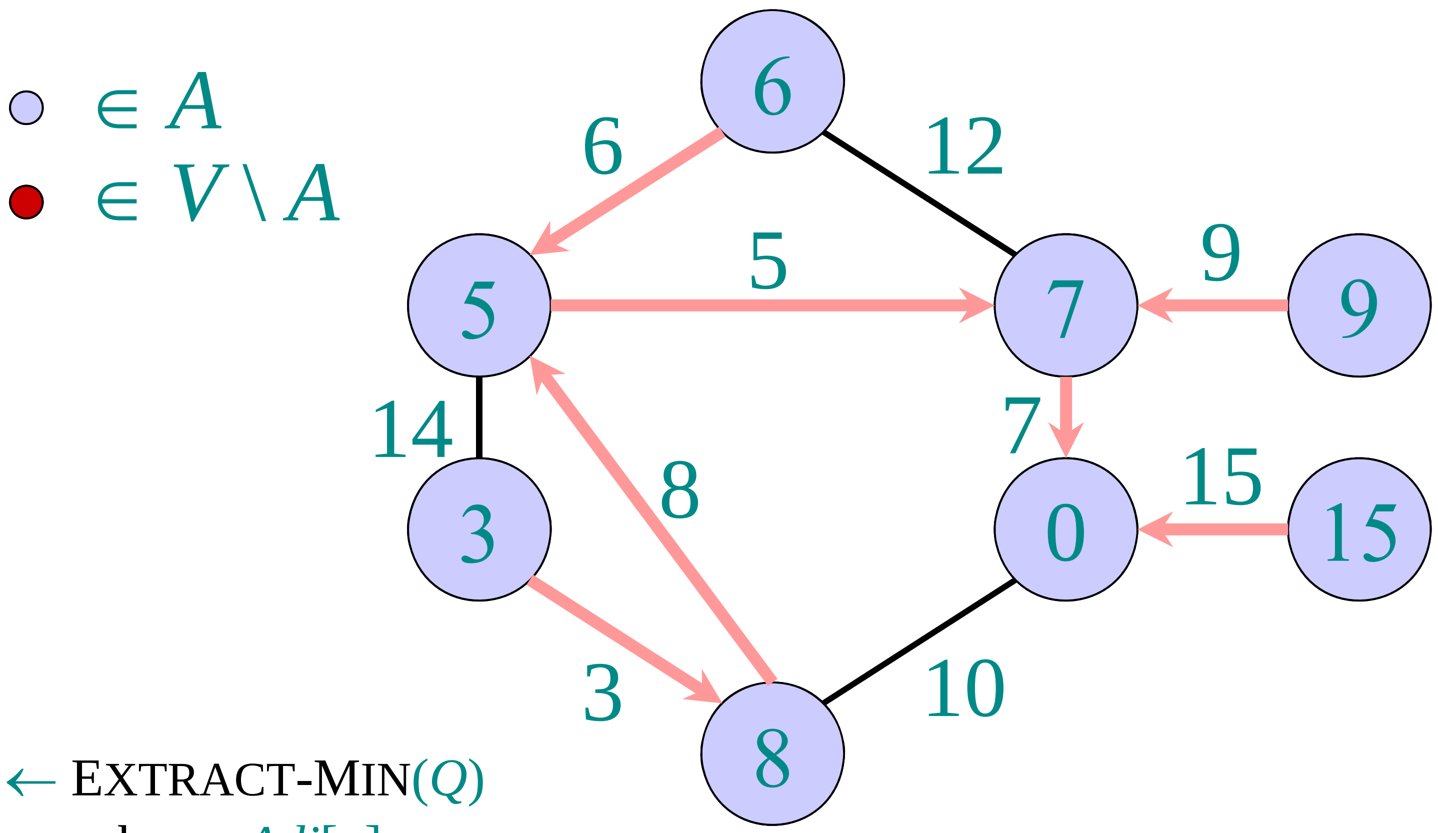
$u \leftarrow$ EXTRACT-MIN(Q)
for each $v \in Adj[u]$
    do if $v \in Q$ and $w(u, v) < key[v]$
        then $key[v] \leftarrow w(u, v)$ ⊳ DECREASE-KEY
            $\pi[v] \leftarrow u$

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



$u \leftarrow$ 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	$O(\|V\|)$	$O(1)$	$O(\|V\|^2)$
binary heap	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\|E\| \log \|V\|)$
Fibonacci heap	$O(\log \|V\|)$ amortized	$O(1)$ amortized	$O(\|E\| + \|V\| \log \|V\|)$ worst case

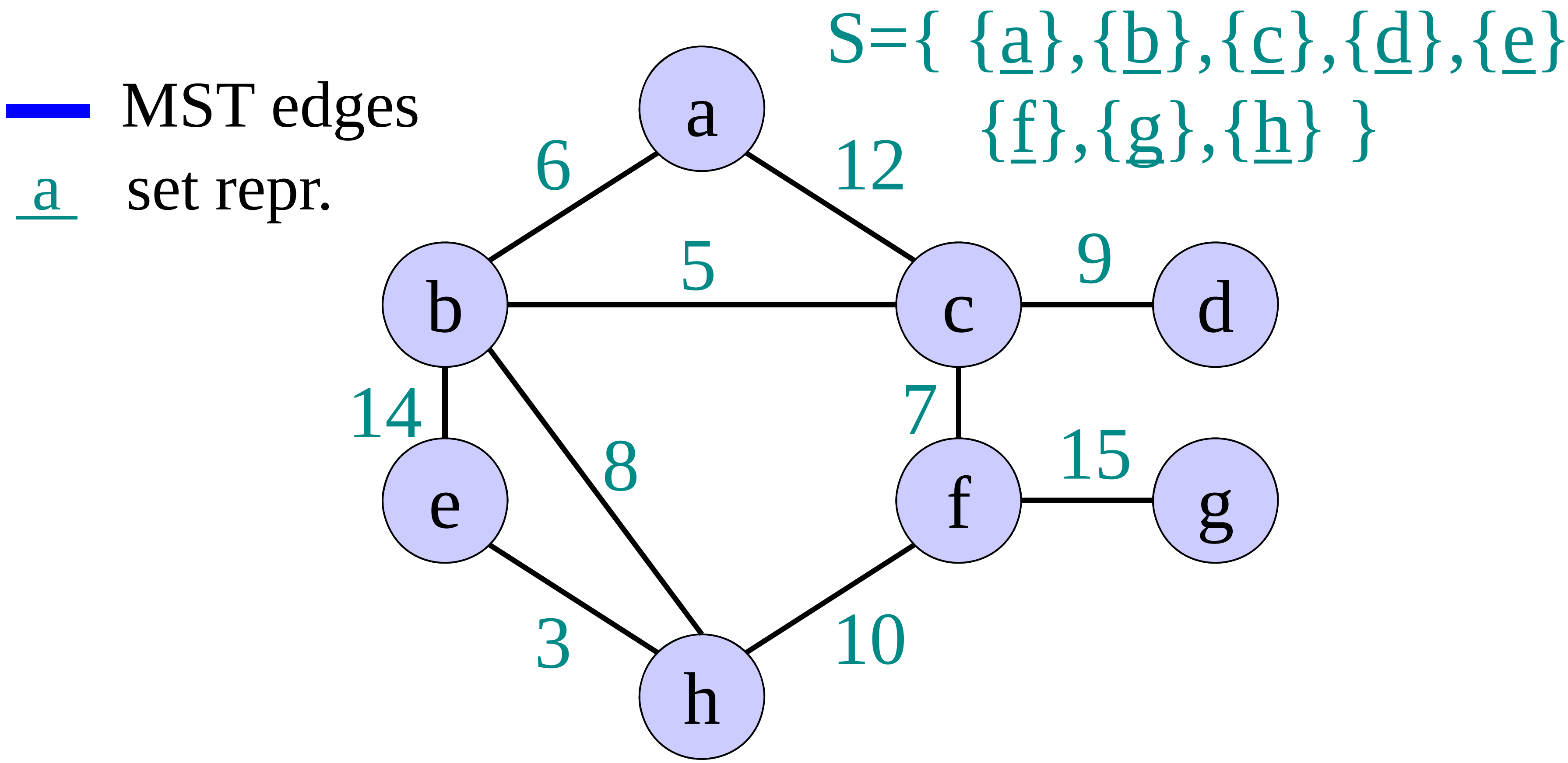
Kruskal's algorithm

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

- Correctness: Next edge e connects two components T_1, T_2. It is the lightest edge which does not produce a cycle, hence it is also the lightest edge between T_1 and $V \backslash T_1$ and therefore satisfies the cut property.

Example of Kruskal's algorithm



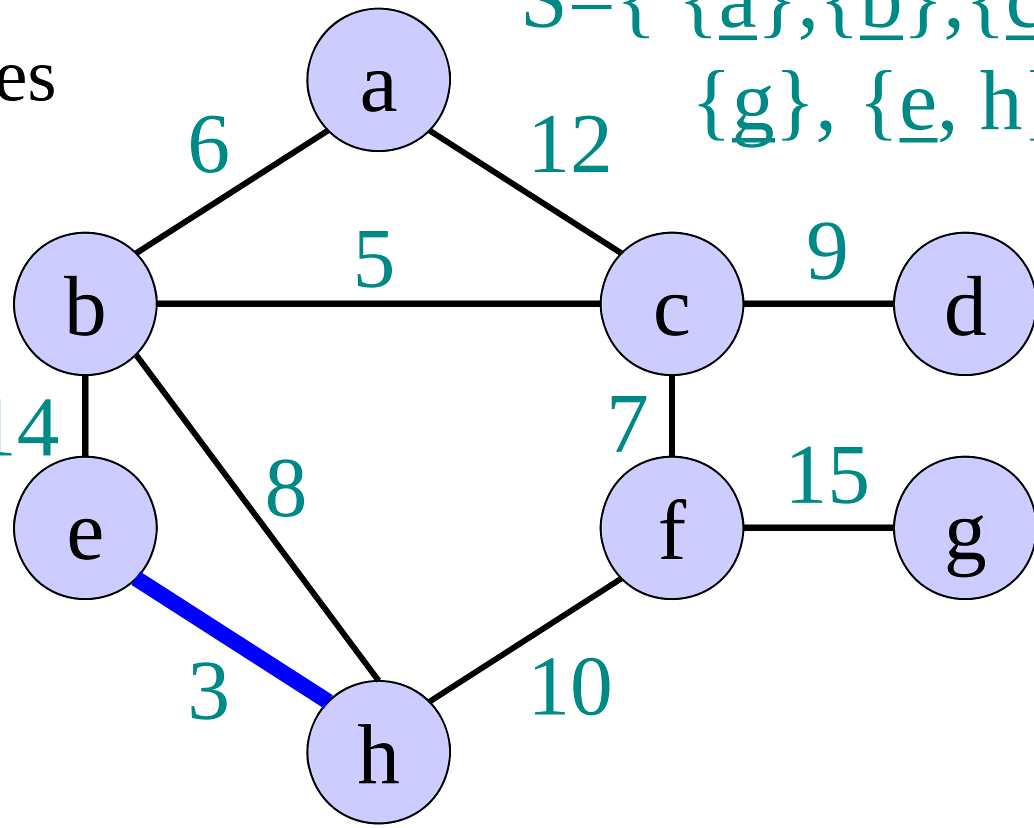
Every node is a single tree.

Example of Kruskal's algorithm

MST edges

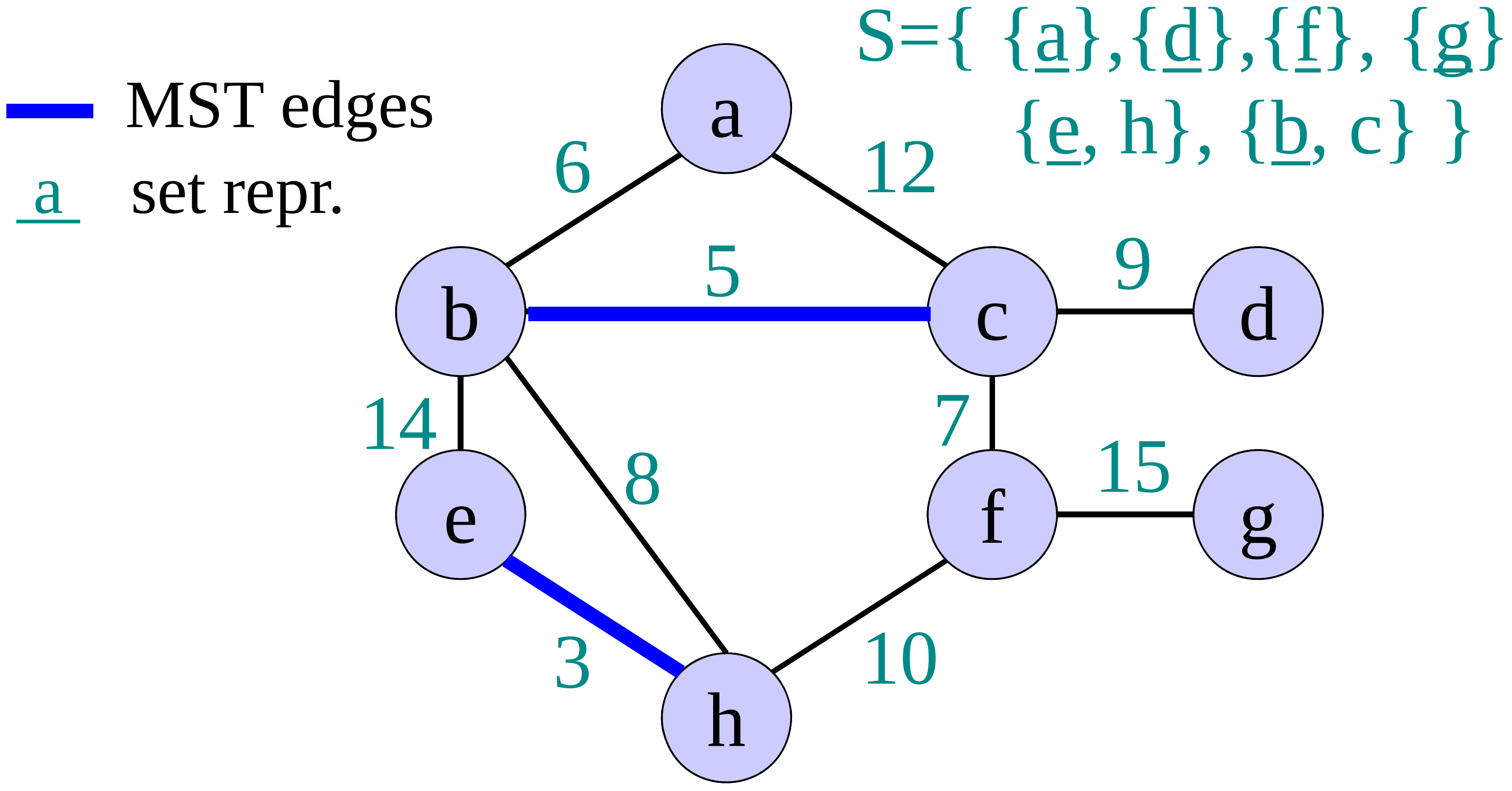
a set repr.

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



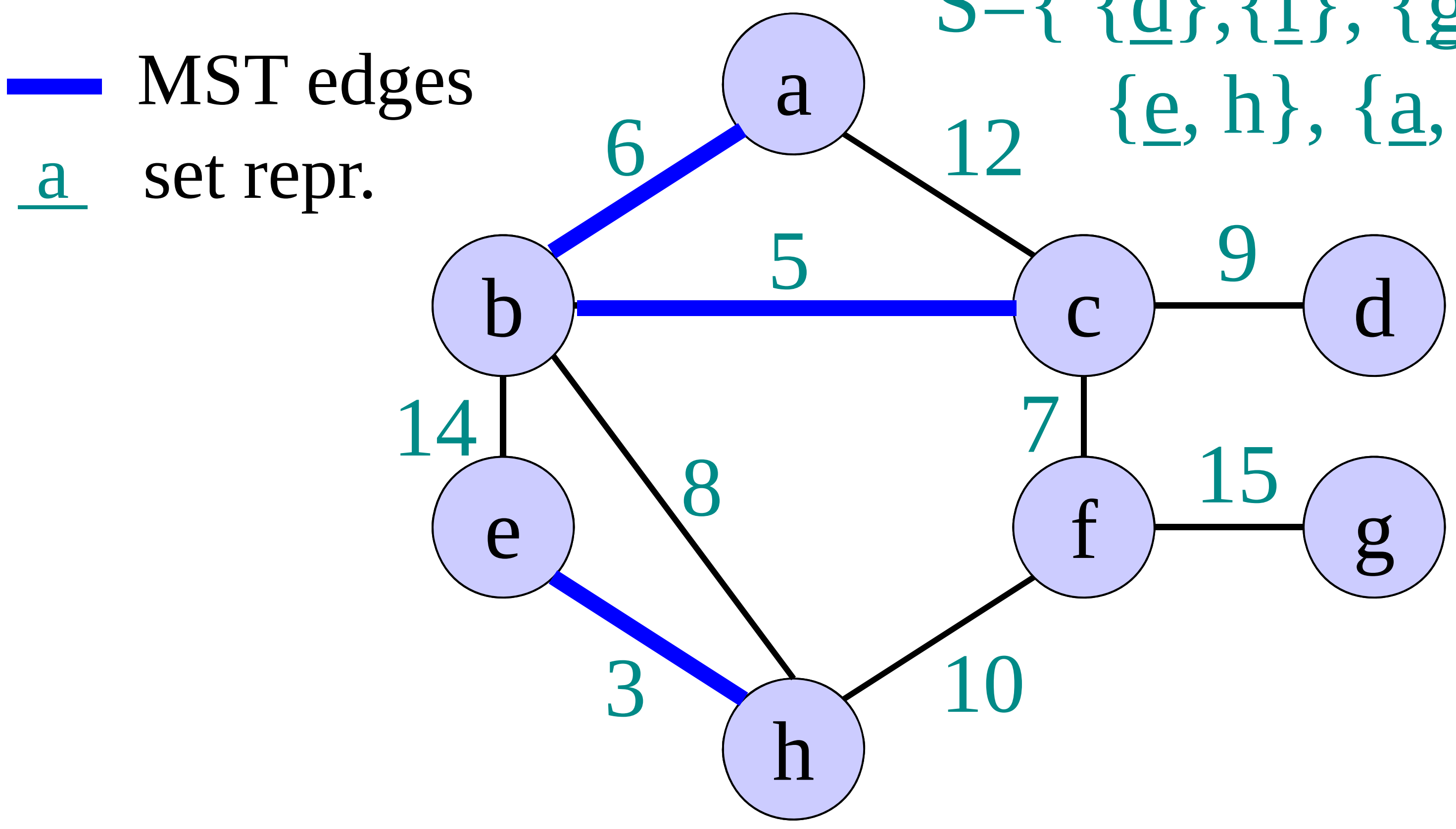
Edge 3 merged two singleton trees.

Example of Kruskal's algorithm

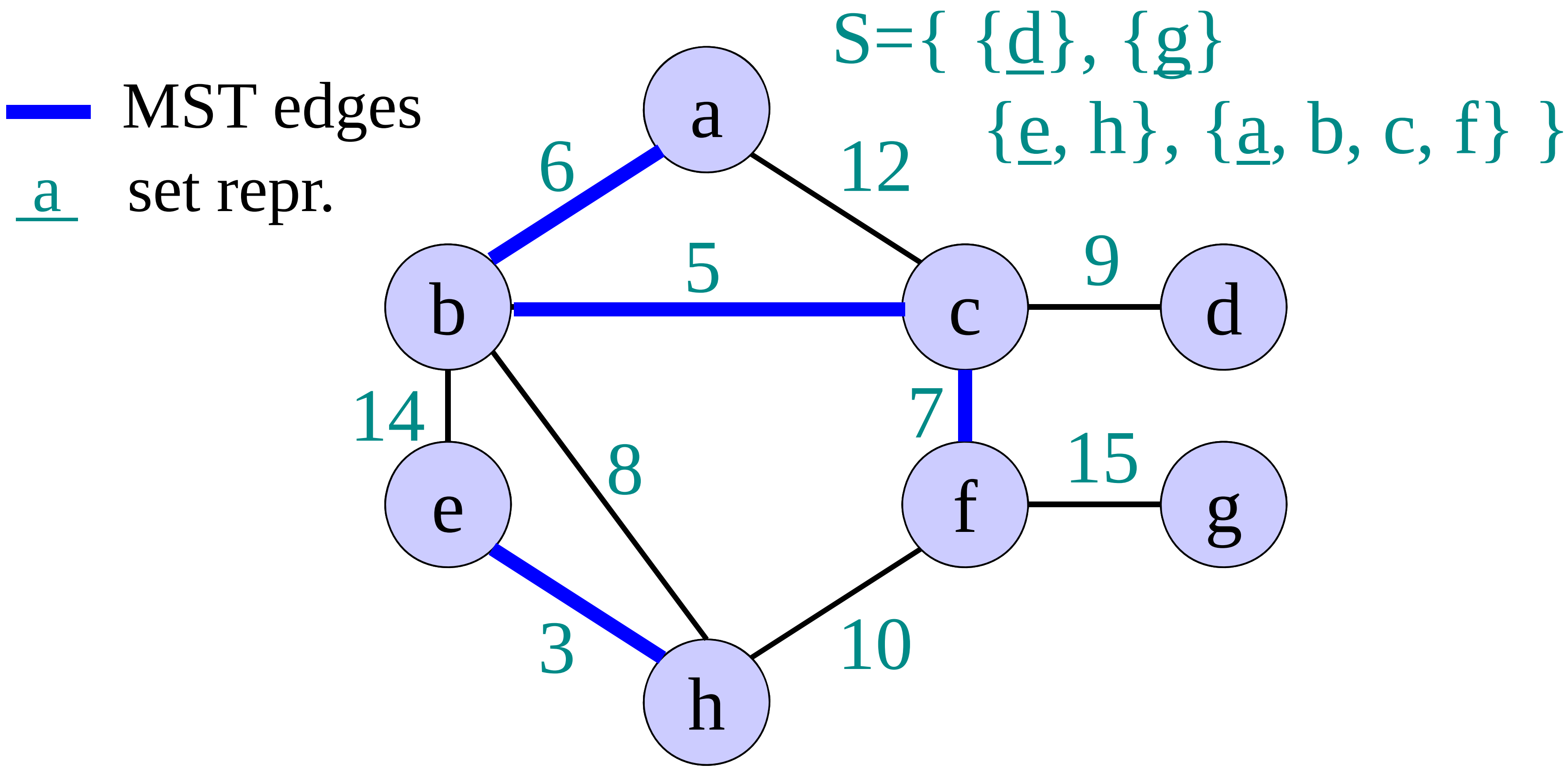


Example of Kruskal's algorithm

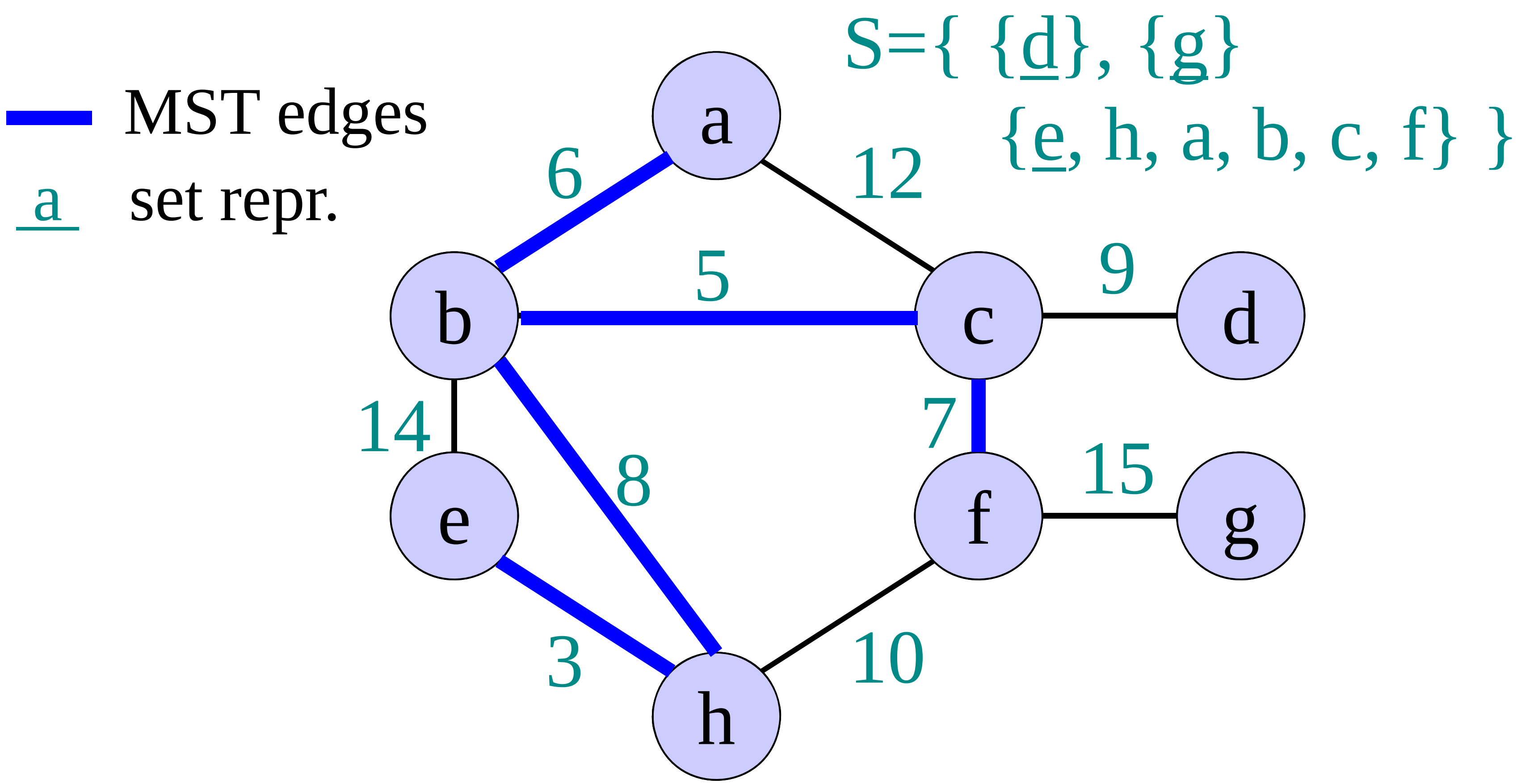
S={ {d},{f}, {g}
{e, h}, {a, b, c} }



Example of Kruskal's algorithm

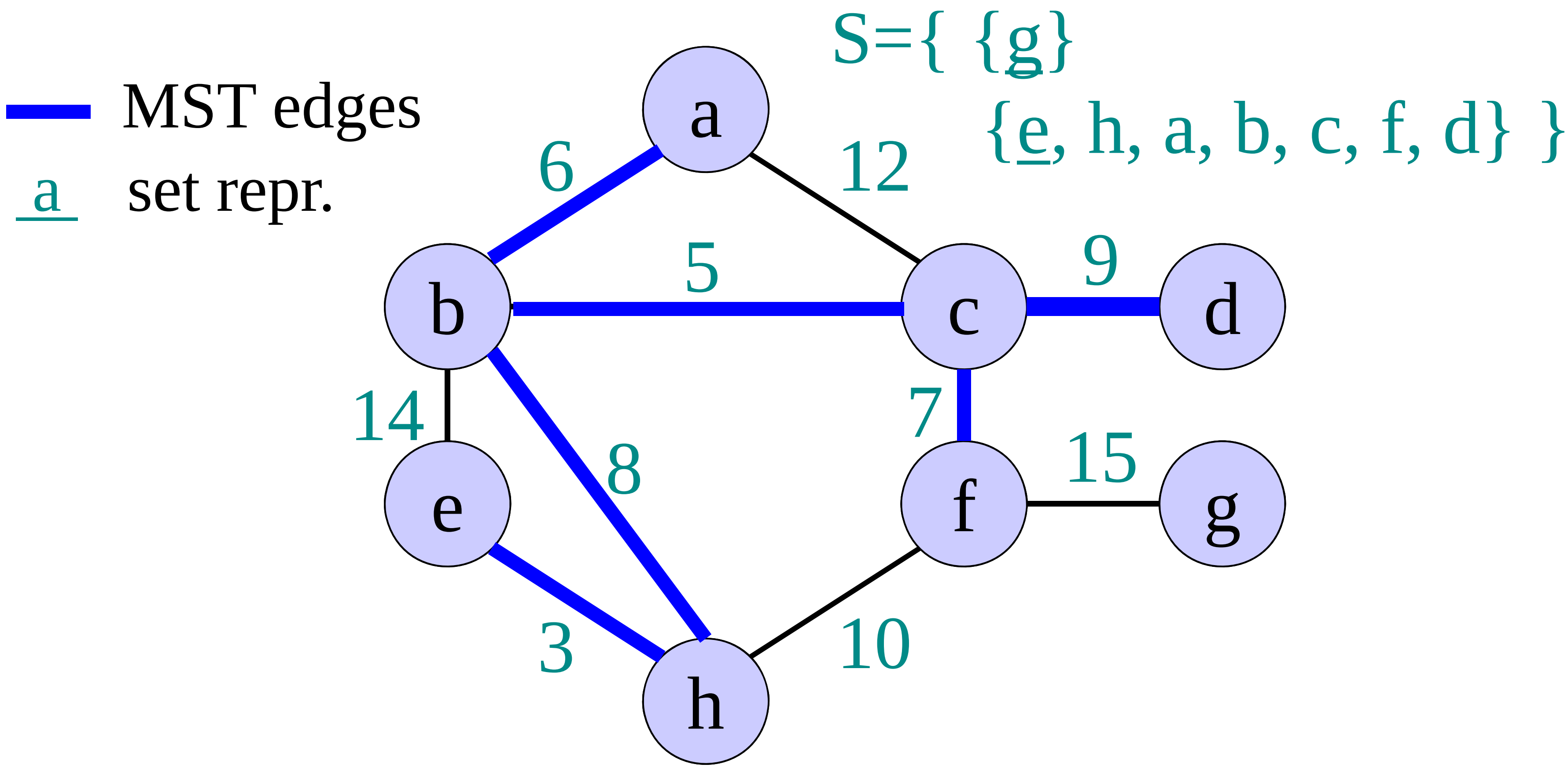


Example of Kruskal's algorithm

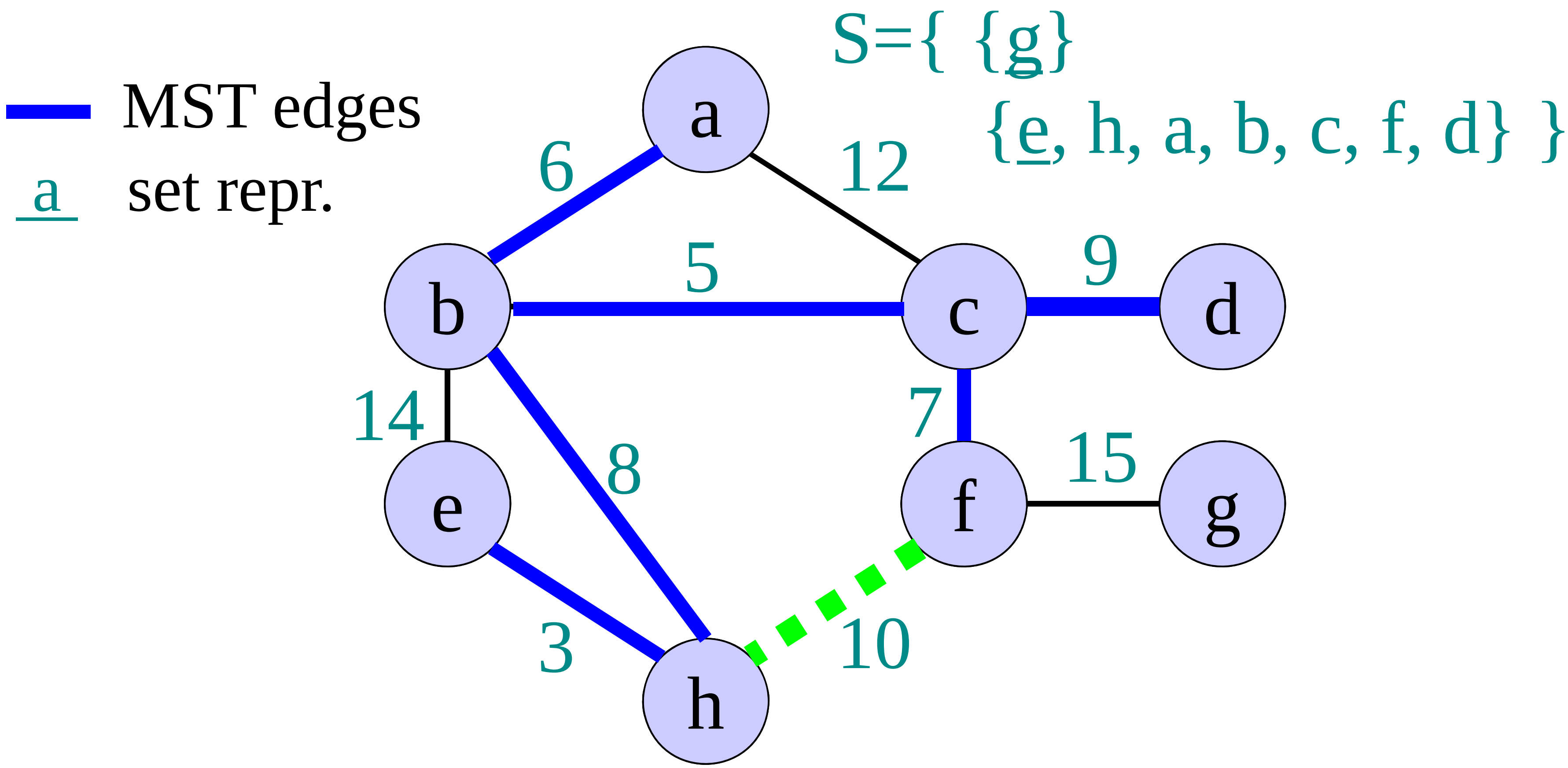


Edge 8 merged the two bigger trees.

Example of Kruskal’s algorithm



Example of Kruskal's algorithm



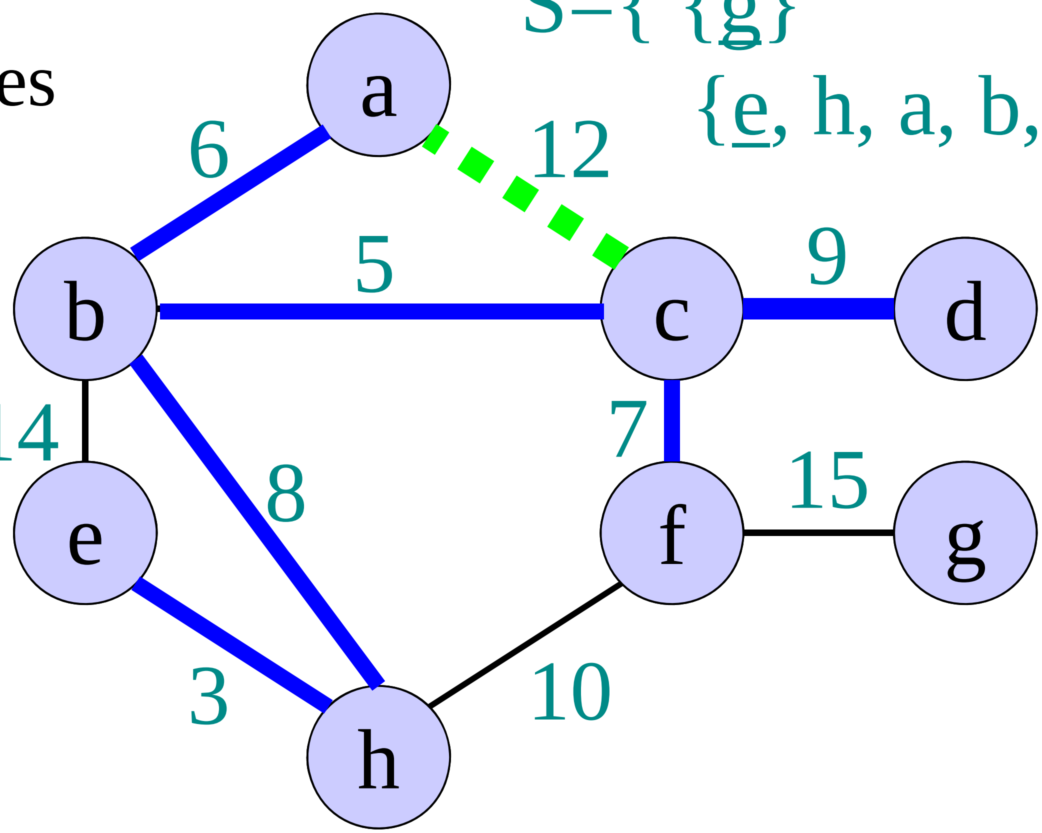
Skip edge 10 as it would cause a cycle.

Example of Kruskal's algorithm

MST edges

a set repr.

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



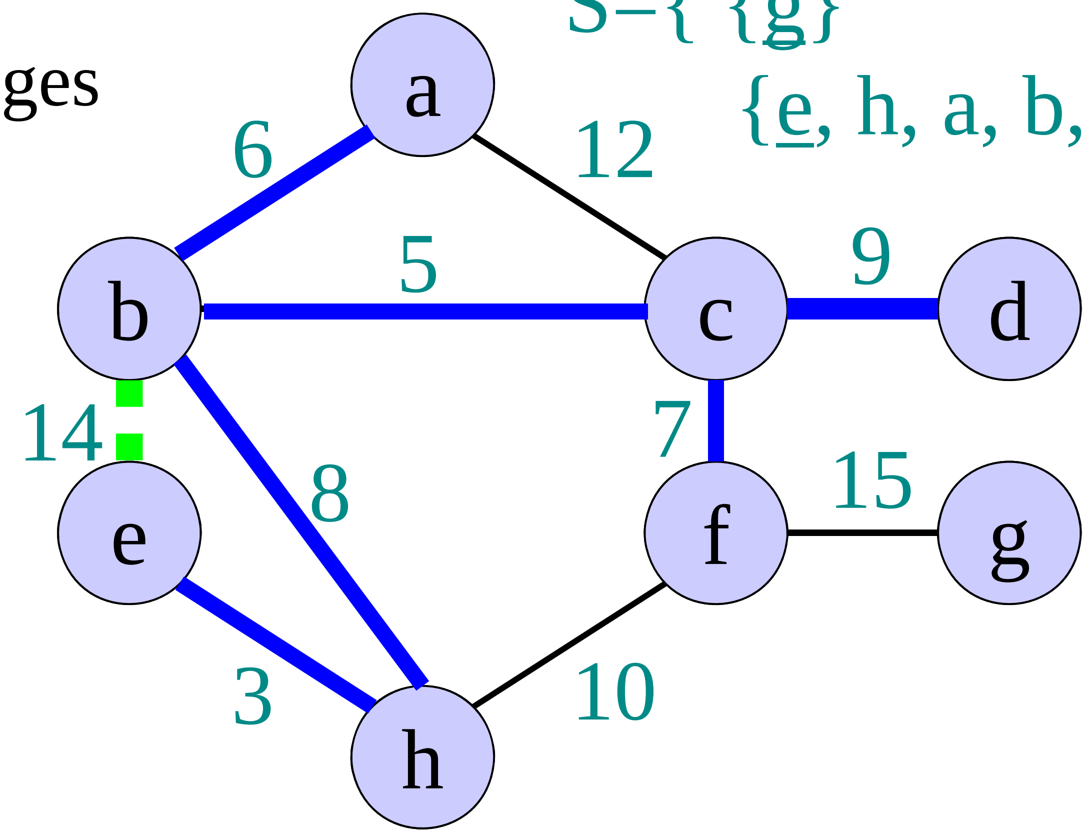
Skip edge 12 as it would cause a cycle.

Example of Kruskal's algorithm

MST edges

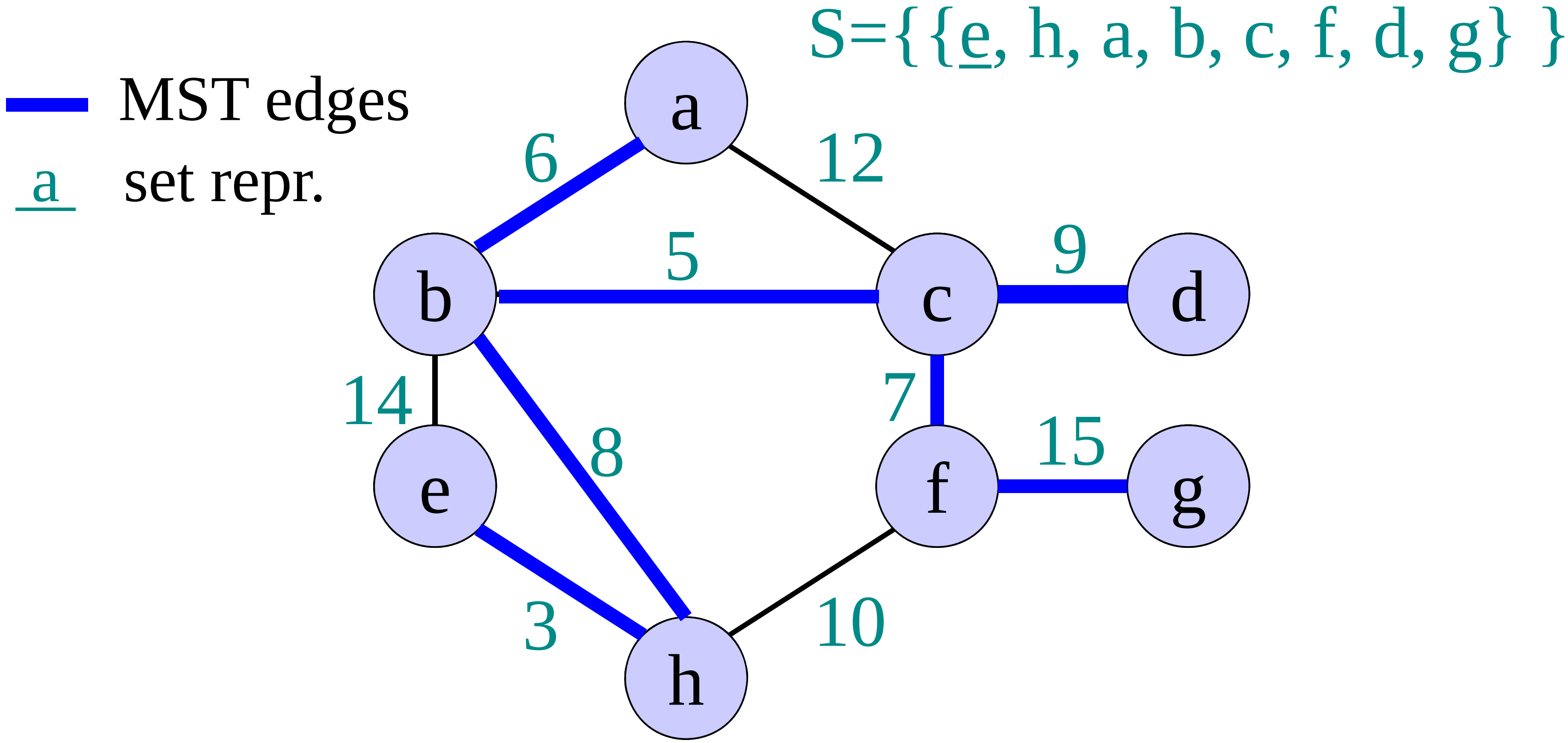
a set repr.

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



Skip edge 14 as it would cause a cycle.

Example of Kruskal's algorithm



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 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
$O(\lvert V\rvert)$	**for** each $v \in V$ **do** MAKE-SET(v)
$O(\lvert E\rvert\log\lvert E\rvert)$	Sort edges of E in non-decreasing order according to w
$O(\lvert E\rvert)$	**For** each $(u,v) \in E$ taken in this order **do**
$O(\alpha(\lvert V\rvert))$	**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 $O(|V| + |E|)$

Disjoint-set data structure (Union-Find)

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

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

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

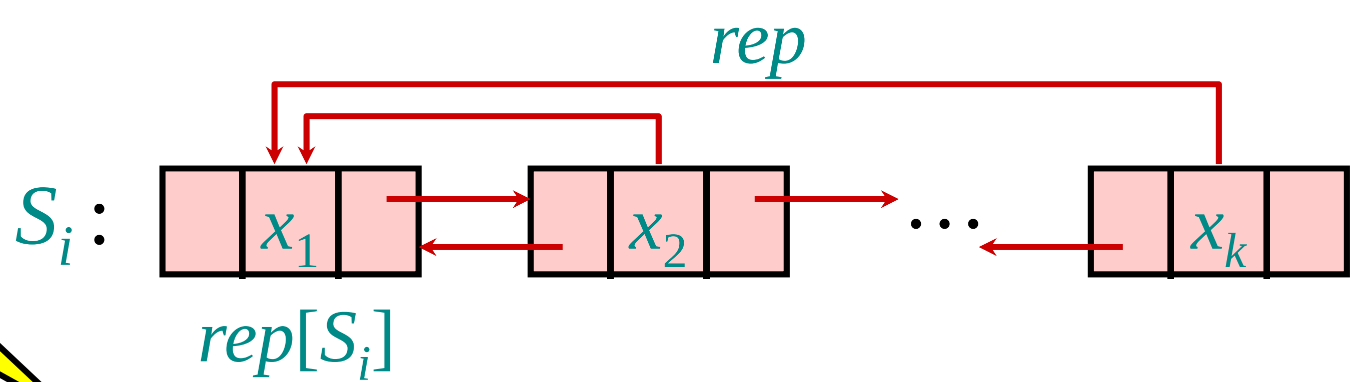
$O(\alpha(n))$ • FIND-SET(x): returns the representative of the set S_x containing element x

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

Augmented linked-list solution

Store $S_i = \{x_1, x_2, \ldots, x_k\}$ as unordered doubly linked list.
Augmentation: Each element x_j also stores pointer $rep[x_j]$ to $rep[S_i]$ (which is the front of the list, x_1).

Assume pointer to x is given.



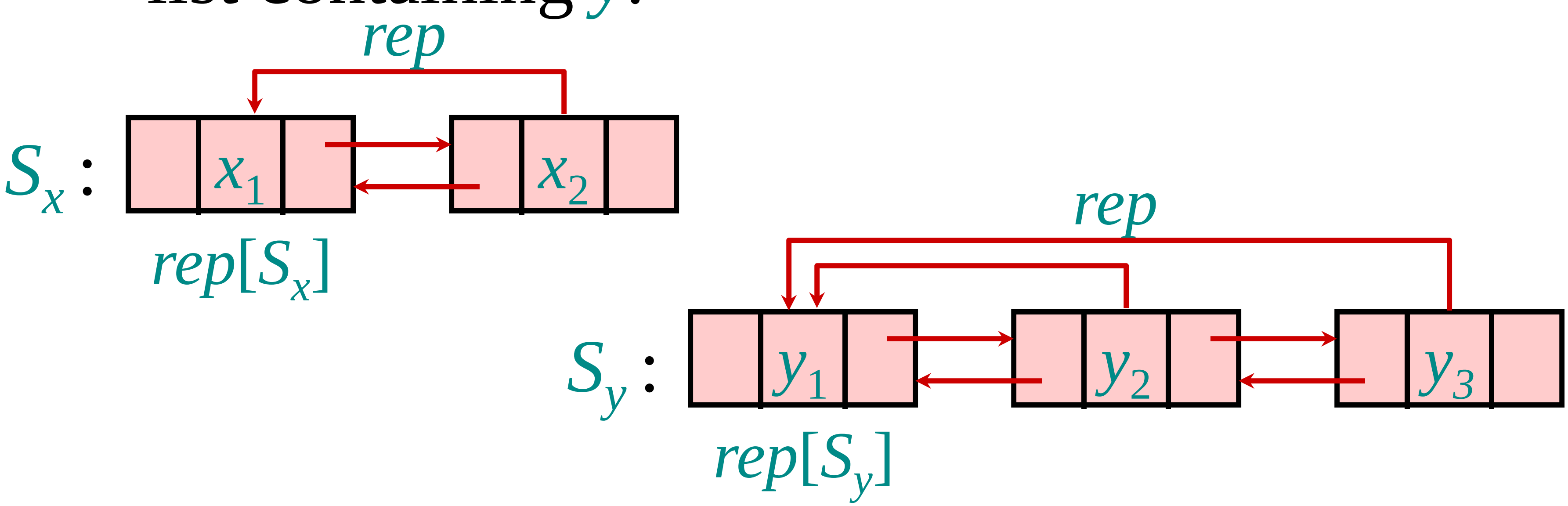
- FIND-SET(x) returns $rep[x]$. – $\Theta(1)$
- UNION(x, y) concatenates lists containing x and y and updates the rep pointers for all elements in the list containing y. – $\Theta(n)$

Example of augmented linked-list solution

Each element x_j stores pointer $rep[x_j]$ to $rep[S_i]$.

UNION(x, y)

- concatenates the lists containing x and y, and
- updates the *rep* pointers for all elements in the list containing y.



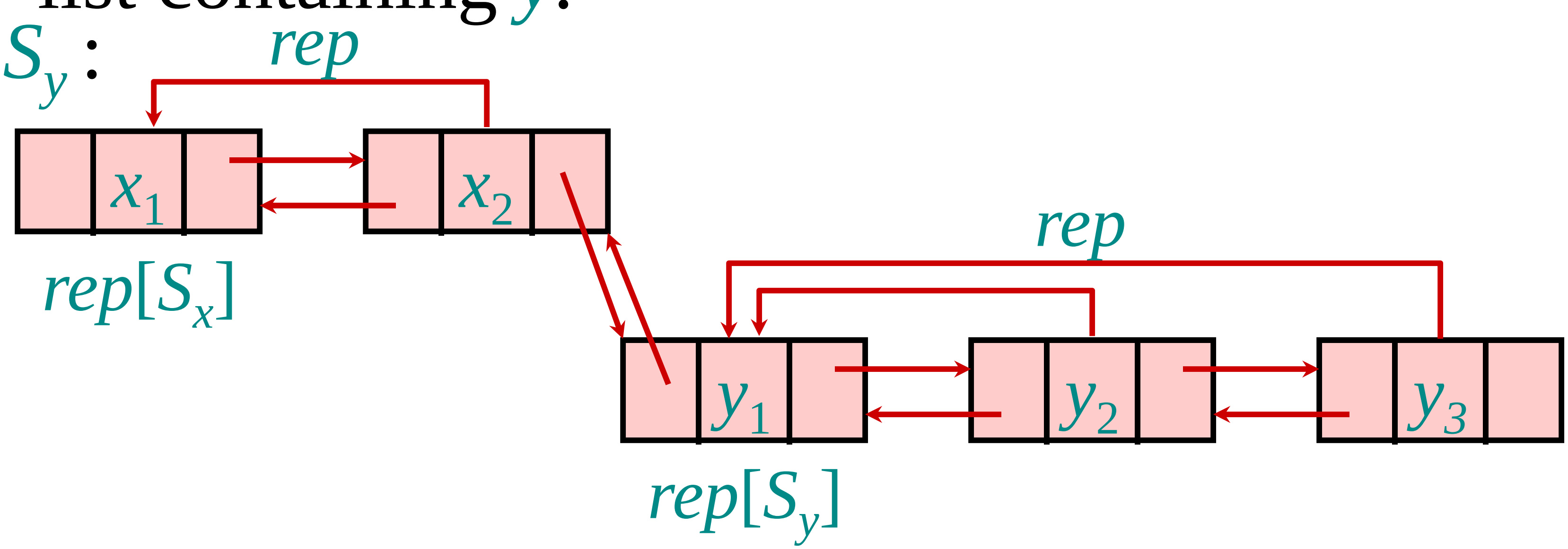
Example of augmented linked-list solution

Each element x_j stores pointer $rep[x_j]$ to $rep[S_i]$.

UNION(x, y)

- concatenates the lists containing x and y, and
- updates the *rep* pointers for all elements in the list containing y.

$S_x \cup S_y$:

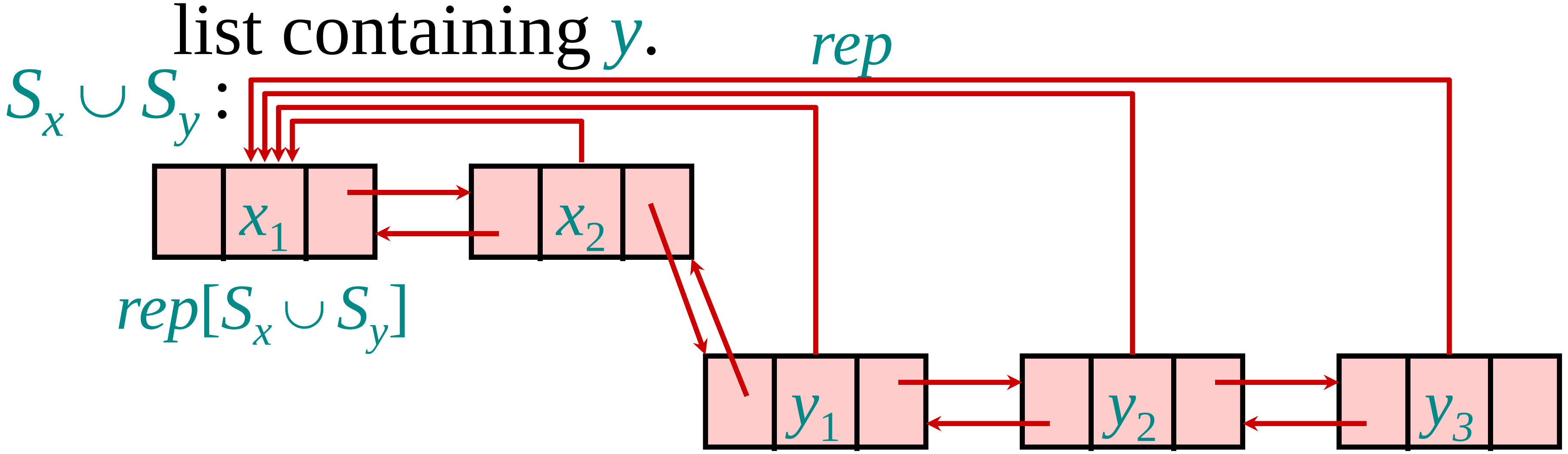


Example of augmented linked-list solution

Each element x_j stores pointer $rep[x_j]$ to $rep[S_i]$.

UNION(x, y)

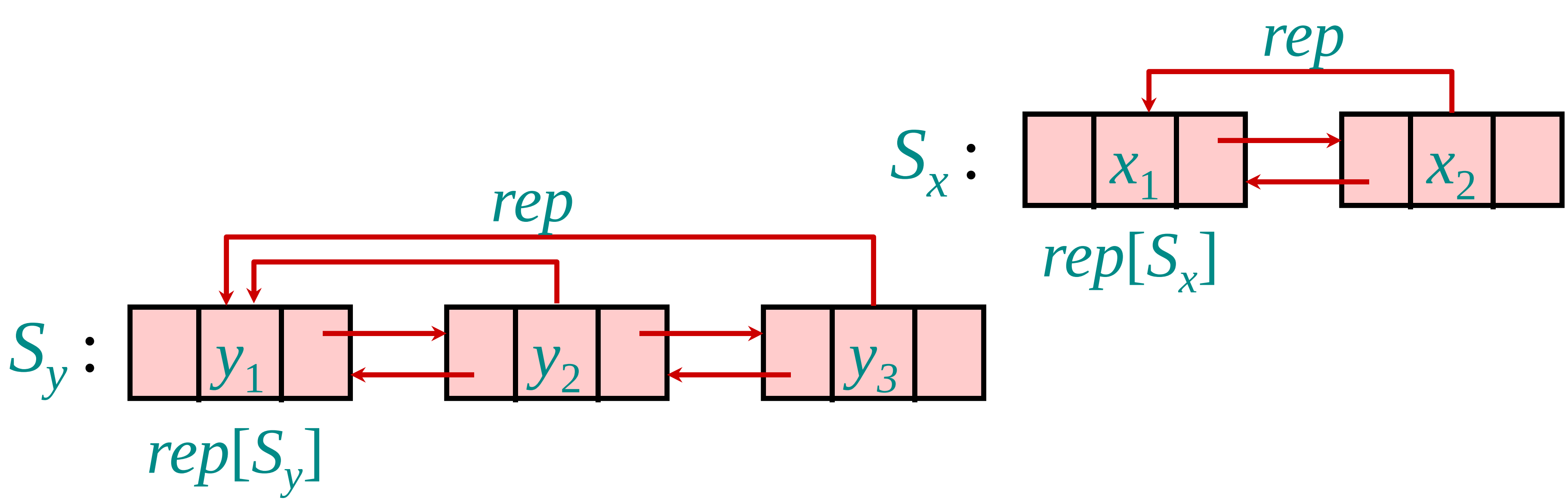
- concatenates the lists containing x and y, and
- updates the *rep* pointers for all elements in the list containing y.



Alternative concatenation

UNION(*x*, *y*) could instead

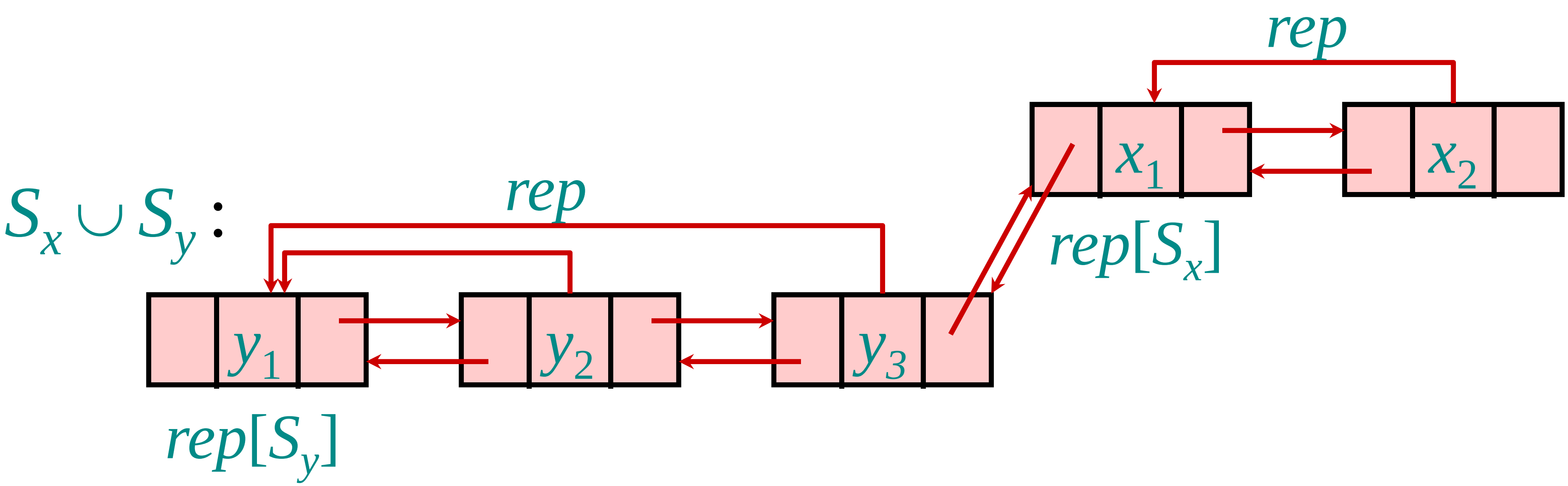
- concatenate the lists containing *y* and *x*, and
- update the *rep* pointers for all elements in the list containing *x*.



Alternative concatenation

UNION(x, y) could instead

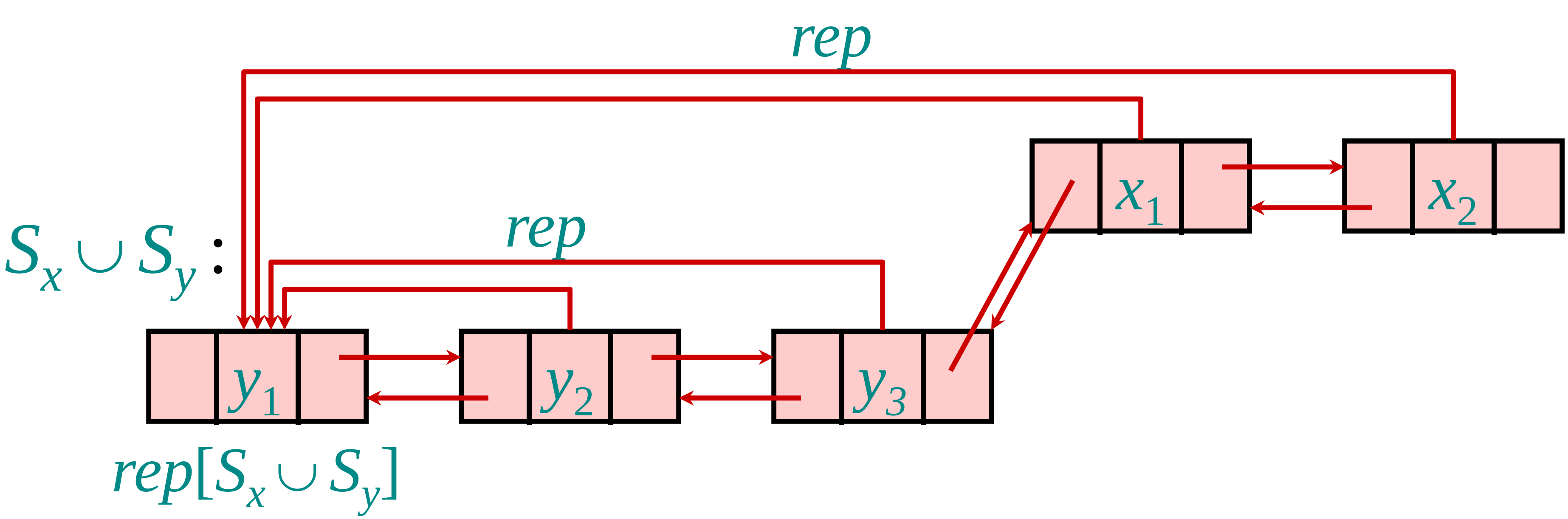
- concatenate the lists containing y and x, and
- update the *rep* pointers for all elements in the list containing x.



Alternative concatenation

UNION(x, y) could instead

- concatenate the lists containing y and x, and
- update the *rep* pointers for all elements in the list containing x.



Trick 1: Smaller into larger
(weighted-union heuristic)

To save work, concatenate the smaller list onto the end of the larger list. Cost = Θ(length of smaller list). Augment list to store its ***weight*** (# elements).

- Let n denote the overall number of elements (equivalently, the number of MAKE-SET operations).
- Let m denote the total number of operations.
- Let f denote the number of FIND-SET operations.

Theorem: Cost of all UNION's is $O(n \log n)$.

Corollary: Total cost is $O(m + n \log n)$.

Analysis of Trick 1
(weighted-union heuristic)

Theorem: Total cost of UNION's is $O(n \log n)$.

Proof. • Monitor an element x and set S_x containing it.

- After initial MAKE-SET(x), $weight[S_x] = 1$.
- Each time S_x is united with S_y:
  - if $weight[S_y] \geq weight[S_x]$:
    – pay 1 to update $rep[x]$, and
    – $weight[S_x]$ at least doubles (increases by $weight[S_y]$).
  - if $weight[S_y] < weight[S_x]$:
    – pay nothing, and
    – $weight[S_x]$ only increases.

Thus pay $\leq \log n$ for x.

Ackermann's function *A*, and it's "inverse" α

Define $A_k(j) = \begin{cases} j+1 & \text{if } k = 0, \\ A_{k-1}^{(j+1)}(j) & \text{if } k \geq 1. \end{cases}$ – iterate j+1 times

$A_0(j) = j + 1$

$A_1(j) \sim 2\,j$

$A_2(j) \sim 2j\,2^j > 2^j$

$A_3(j) > 2^{2^{2^{\cdot^{\cdot^{\cdot^{2^j}}}}}}\Big\}\, j$

$A_4(j)$ is a lot bigger.

$A_0(1) = 2$

$A_1(1) = 3$

$A_2(1) = 7$

$A_3(1) = 2047$

$A_4(1) > 2^{2^{2^{\cdot^{\cdot^{\cdot^{2^{2047}}}}}}}\Big\}$ 2048 times

Define $\alpha(n) = \min\,\{k : A_k(1) \geq n\} \leq 4$ for practical n.