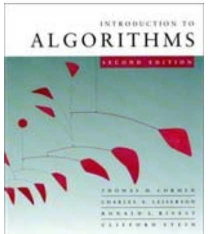
CS60020: Foundations of Algorithm Design and Machine Learning

Sourangshu Bhattacharya



Graphs (review)

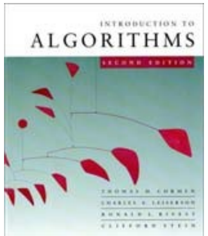
Definition. A ***directed graph (digraph)*** $G = (V, E)$ is an ordered pair consisting of

- a set V of ***vertices*** (singular: ***vertex***),
- a set $E \subseteq V \times V$ of ***edges***.

In an ***undirected graph*** $G = (V, E)$, the edge set E consists of *unordered* pairs of vertices.

In either case, we have $|E| = O(V^2)$. Moreover, if G is connected, then $|E| \geq |V| - 1$, which implies that $\lg |E| = \Theta(\lg V)$.

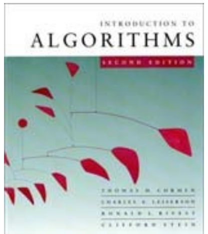
(Review CLRS, Appendix B.)



Adjacency-matrix representation

The ***adjacency matrix*** of a graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, is the matrix $A[1 \,..\, n, 1 \,..\, n]$ given by

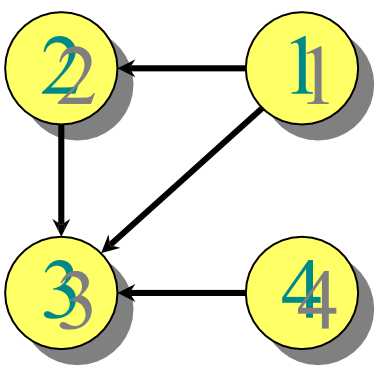
$$A[i, j] = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{if } (i, j) \notin E. \end{cases}$$



Adjacency-matrix representation

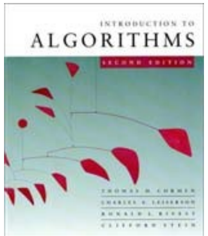
The ***adjacency matrix*** of a graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, is the matrix $A[1 \,..\, n, 1 \,..\, n]$ given by

$$A[i, j] = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{if } (i, j) \notin E. \end{cases}$$



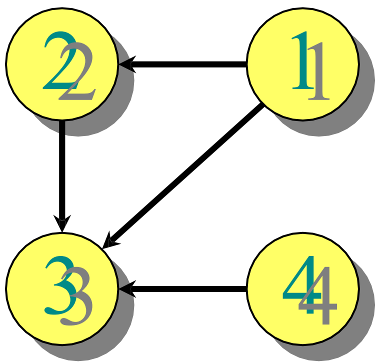
A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

$\Theta(V^2)$ storage $\Rightarrow$ ***dense*** representation.

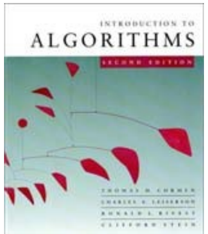


Adjacency-list representation

An ***adjacency list*** of a vertex $v \in V$ is the list $Adj[v]$ of vertices adjacent to v.

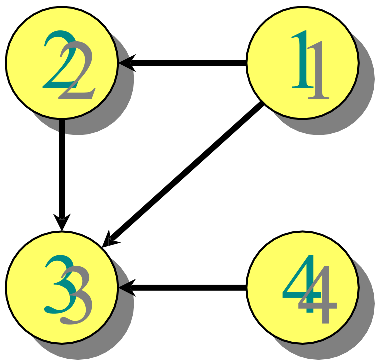


$Adj[1] = \{2, 3\}$
$Adj[2] = \{3\}$
$Adj[3] = \{\}$
$Adj[4] = \{3\}$



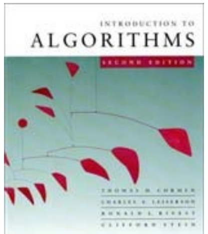
Adjacency-list representation

An ***adjacency list*** of a vertex $v \in V$ is the list $Adj[v]$ of vertices adjacent to v.



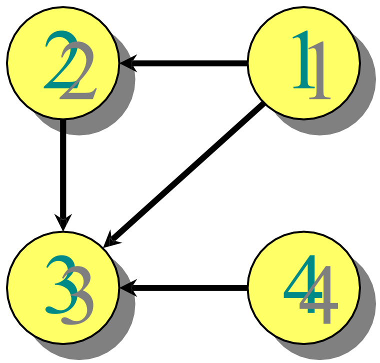
$Adj[1] = \{2, 3\}$
$Adj[2] = \{3\}$
$Adj[3] = \{\}$
$Adj[4] = \{3\}$

For undirected graphs, $|Adj[v]| = degree(v)$.
For digraphs, $|Adj[v]| = out\text{-}degree(v)$.



Adjacency-list representation

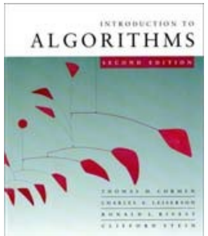
An ***adjacency list*** of a vertex $v \in V$ is the list $Adj[v]$ of vertices adjacent to v.



$Adj[1] = \{2, 3\}$
$Adj[2] = \{3\}$
$Adj[3] = \{\}$
$Adj[4] = \{3\}$

For undirected graphs, $|Adj[v]| = degree(v)$.
For digraphs, $|Adj[v]| = out\text{-}degree(v)$.

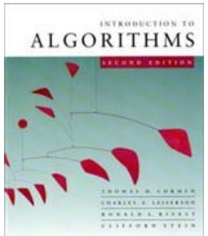
Handshaking Lemma: $\sum_{v \in V} \mathrm{Adj}[v] = 2\,|\mathrm{E}|$ for undirected graphs $\Rightarrow$ adjacency lists use $\Theta(V + E)$ storage — a ***sparse*** representation (for either type of graph).



Minimum spanning trees

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

- For simplicity, assume that all edge weights are distinct. (CLRS covers the general case.)



Minimum spanning trees

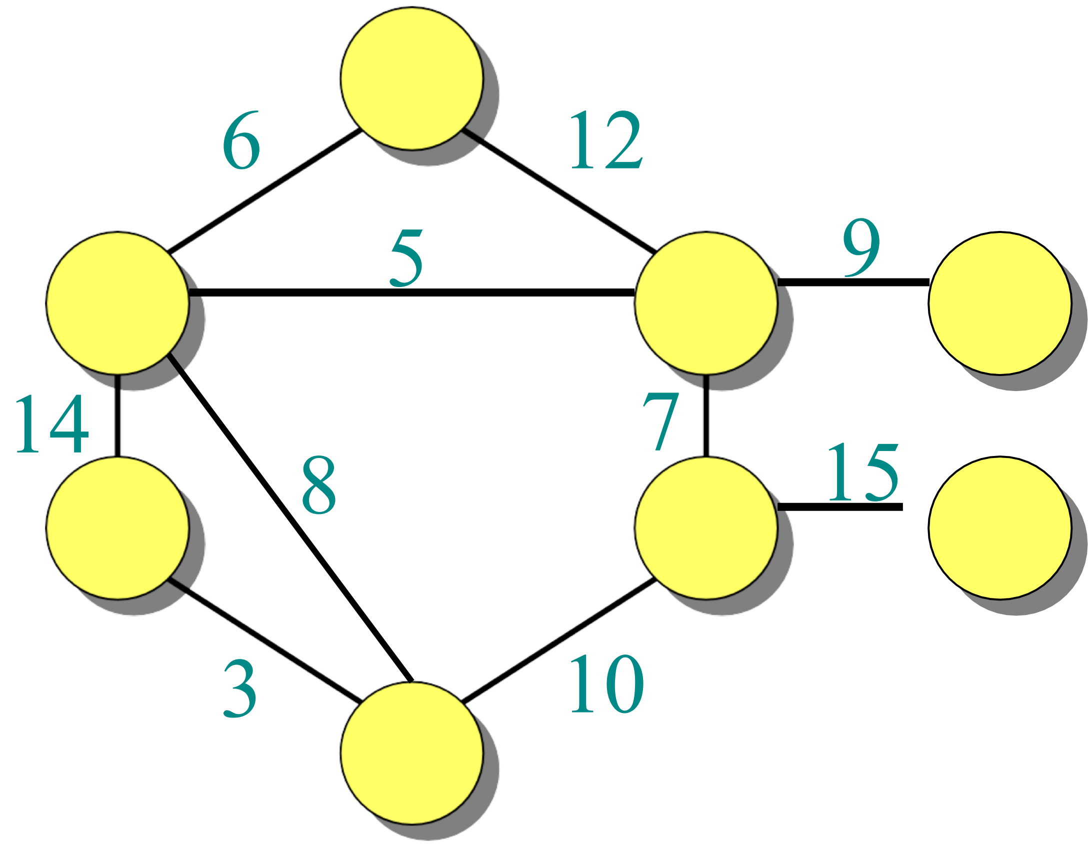
Input: A connected, undirected graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$.

- For simplicity, assume that all edge weights are distinct. (CLRS covers the general case.)

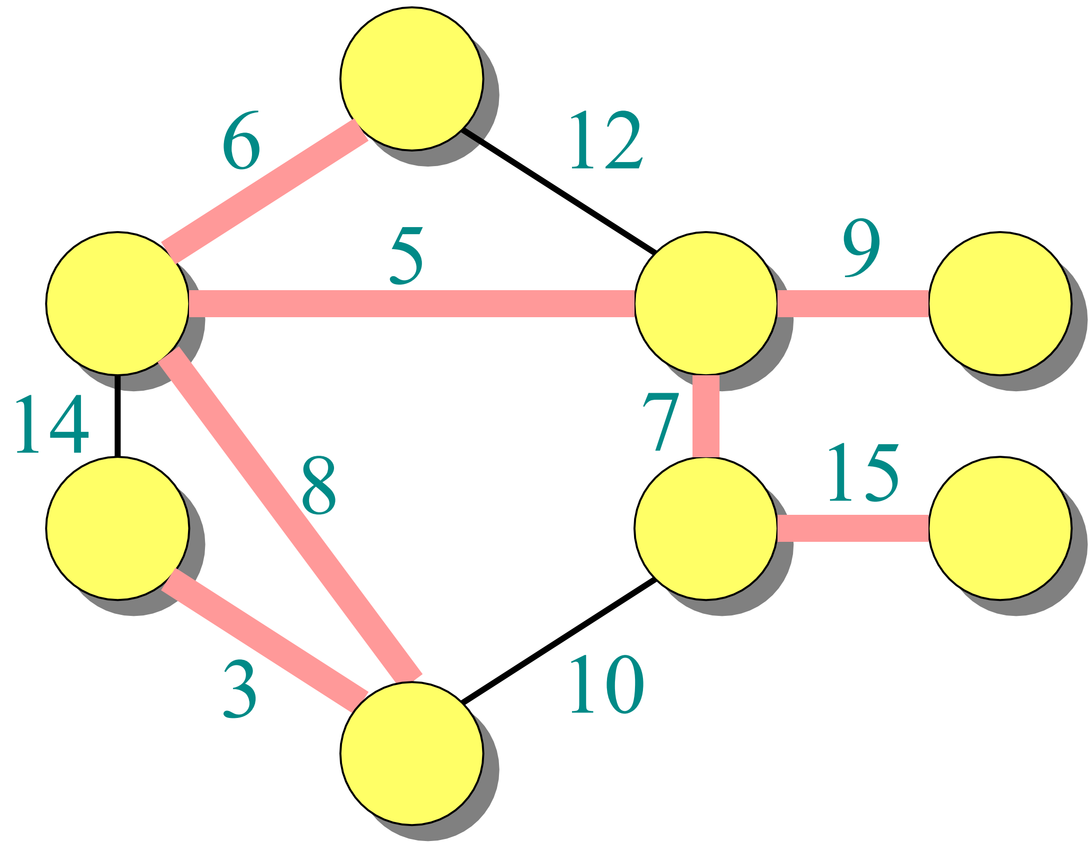
Output: A ***spanning tree*** T — a tree that connects all vertices — of minimum weight:

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

Example of MST



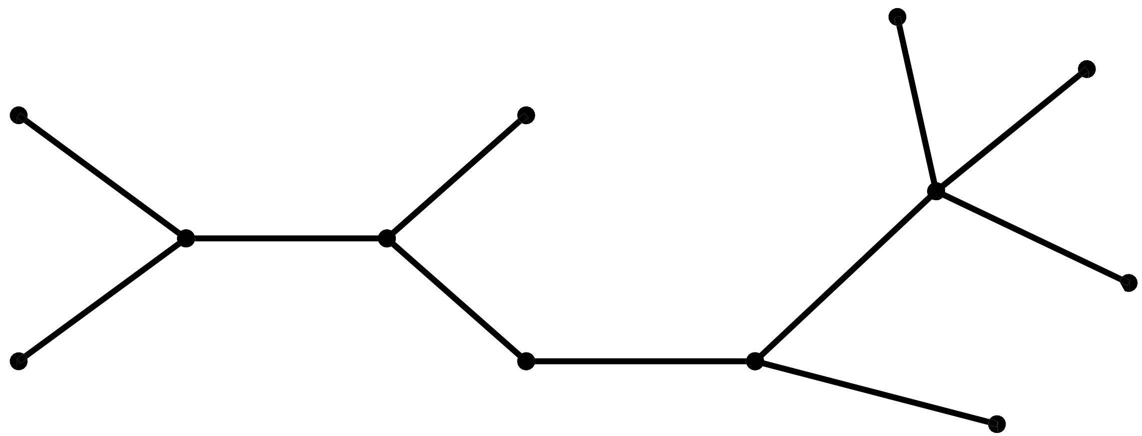
Example of MST



Optimal substructure

MST T:

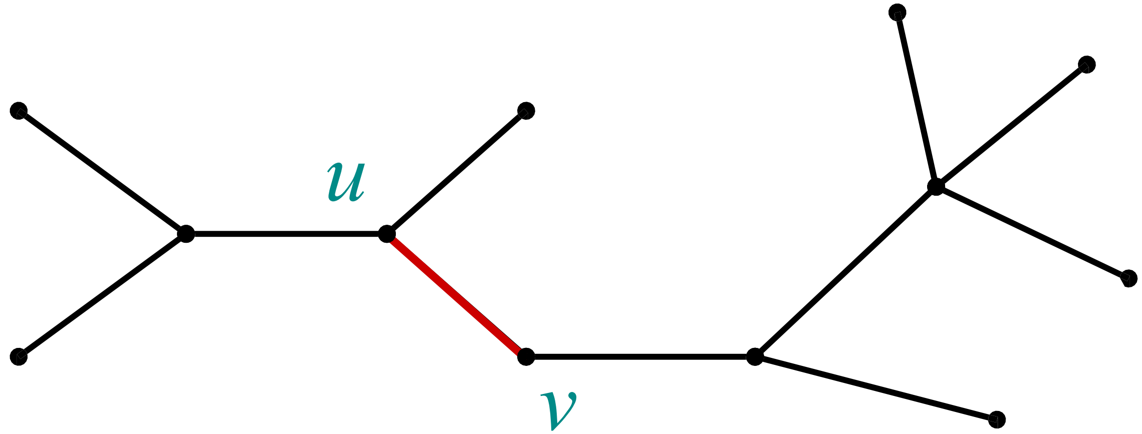
(Other edges of G are not shown.)



Optimal substructure

MST T:

(Other edges of G are not shown.)

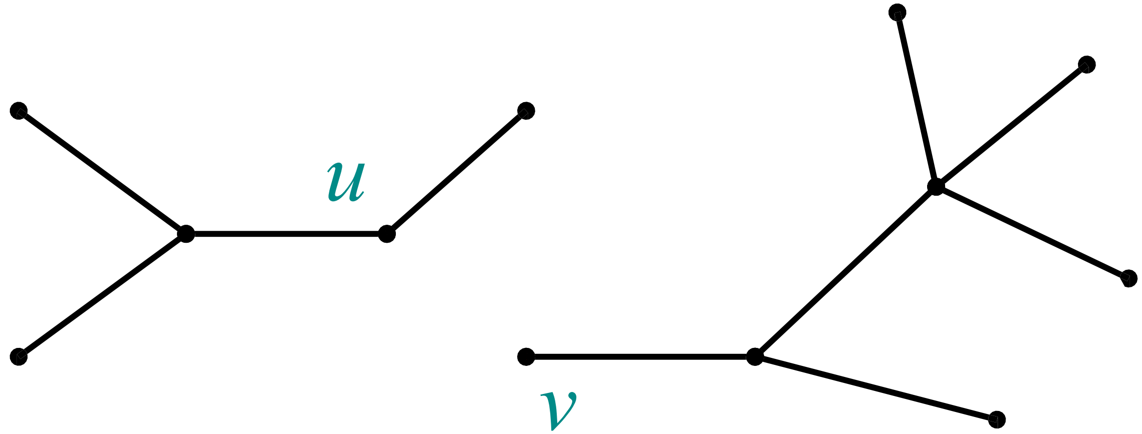


Remove any edge $(u, v) \in T$.

Optimal substructure

MST T:

(Other edges of G are not shown.)

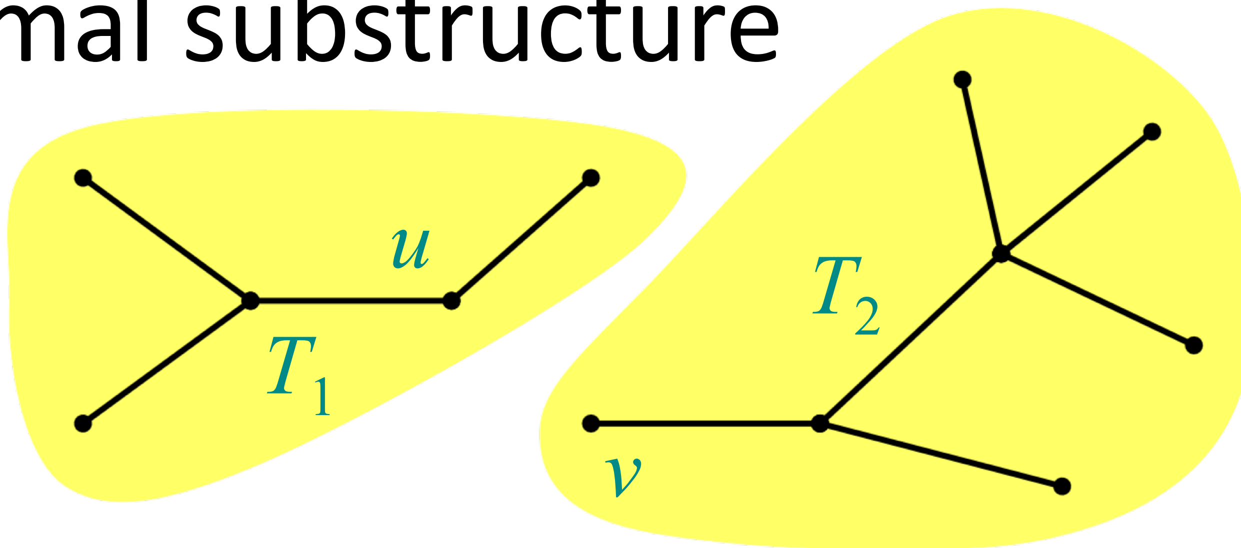


Remove any edge $(u, v) \in T$.

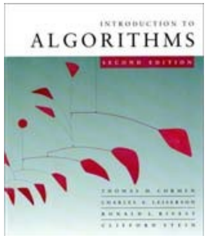
Optimal substructure

MST T:

(Other edges of G are not shown.)



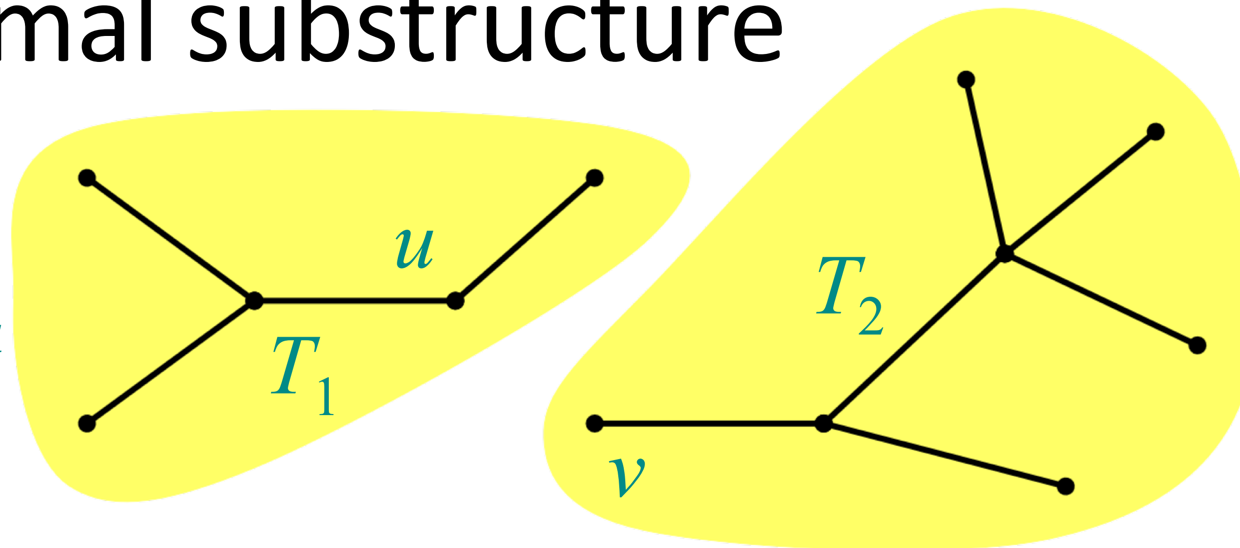
Remove any edge $(u, v) \in T$. Then, T is partitioned into two subtrees T_1 and T_2.



Optimal substructure

MST T:

(Other edges of G are not shown.)

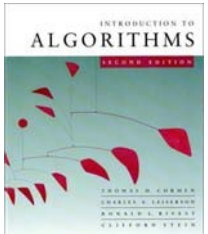


Remove any edge $(u, v) \in T$. Then, T is partitioned into two subtrees T_1 and T_2.

Theorem. The subtree T_1 is an MST of $G_1 = (V_1, E_1)$, the subgraph of G ***induced*** by the vertices of T_1:

$$V_1 = \text{vertices of } T_1,$$
$$E_1 = \{ (x, y) \in E : x, y \in V_1 \}.$$

Similarly for T_2.



Proof of optimal substructure

Proof. Cut and paste:

$$w(T) = w(u, v) + w(T_1) + w(T_2).$$

If T_1' were a lower-weight spanning tree than T_1 for G_1, then $T' = \{(u, v)\} \cup T_1' \cup T_2$ would be a lower-weight spanning tree than T for G. □

Proof of optimal substructure

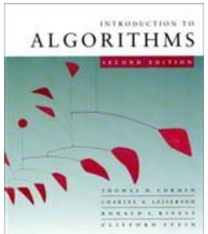
Proof. Cut and paste:

$$w(T) = w(u, v) + w(T_1) + w(T_2).$$

If T_1' were a lower-weight spanning tree than T_1 for G_1, then $T' = \{(u, v)\} \cup T_1' \cup T_2$ would be a lower-weight spanning tree than T for G. $\square$

Do we also have overlapping subproblems?

- Yes.



Proof of optimal substructure

Proof. Cut and paste:

$$w(T) = w(u, v) + w(T_1) + w(T_2).$$

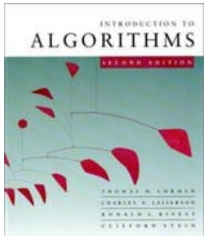
If T_1' were a lower-weight spanning tree than T_1 for G_1, then $T' = \{(u, v)\} \cup T_1' \cup T_2$ would be a lower-weight spanning tree than T for G. $\square$

Do we also have overlapping subproblems?

- Yes.

Great, then dynamic programming may work!

- Yes, but MST exhibits another powerful property which leads to an even more efficient algorithm.



Hallmark for “greedy” algorithms

Greedy-choice property

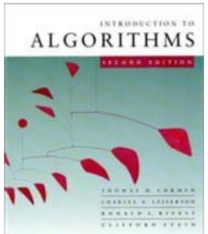
A locally optimal choice is globally optimal.

Hallmark for "greedy" algorithms

Greedy-choice property

A locally optimal choice is globally optimal.

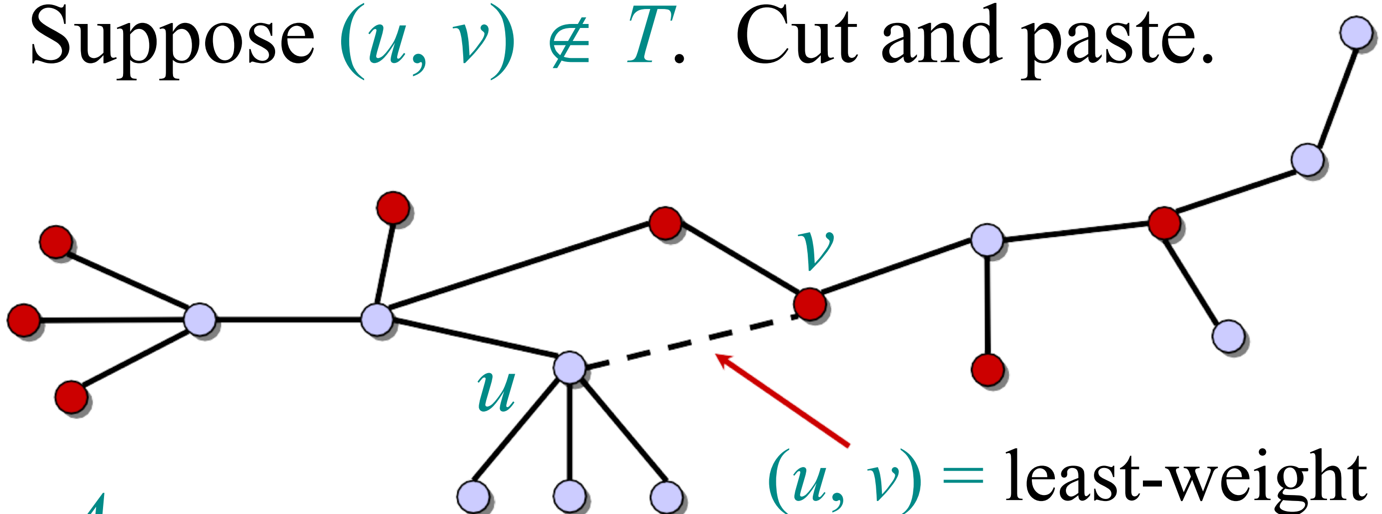
Theorem. Let T be the MST of $G = (V, E)$, and let $A \subseteq V$. Suppose that $(u, v) \in E$ is the least-weight edge connecting A to $V - A$. Then, $(u, v) \in T$.



Proof of theorem

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

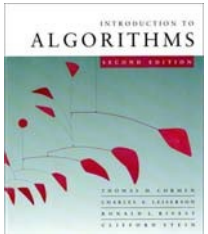
T:



$\in A$

$\in V - A$

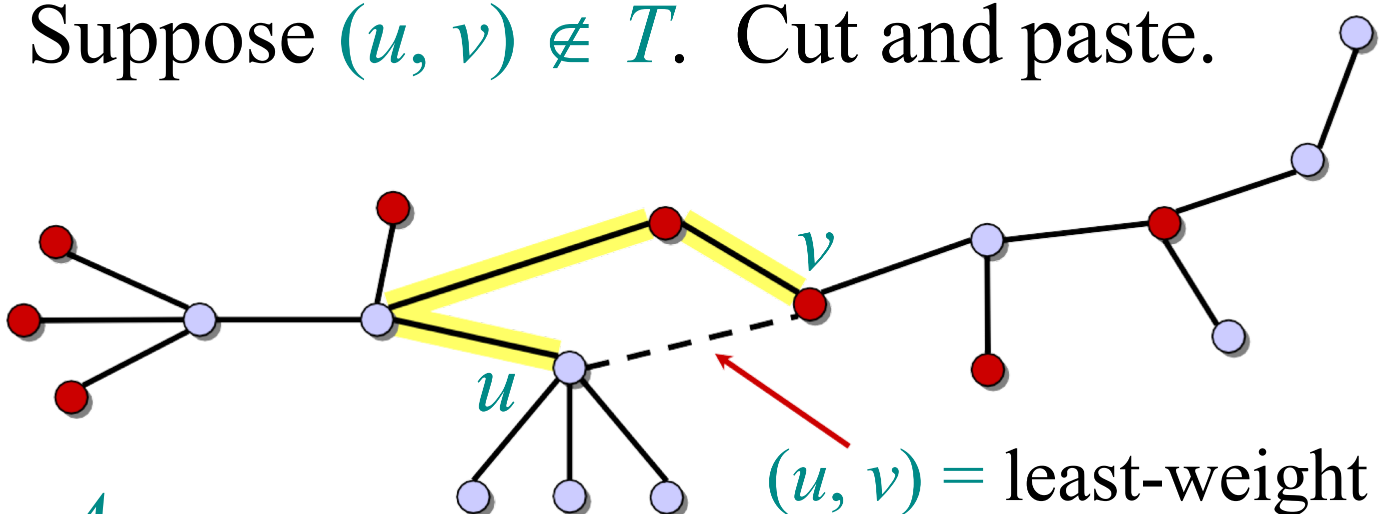
(u, v) = least-weight edge connecting A to $V - A$



Proof of theorem

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

T:

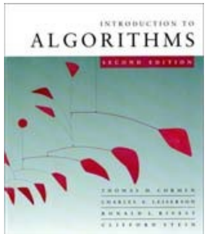


(u, v) = least-weight edge connecting A to $V - A$

● $\in A$

● $\in V - A$

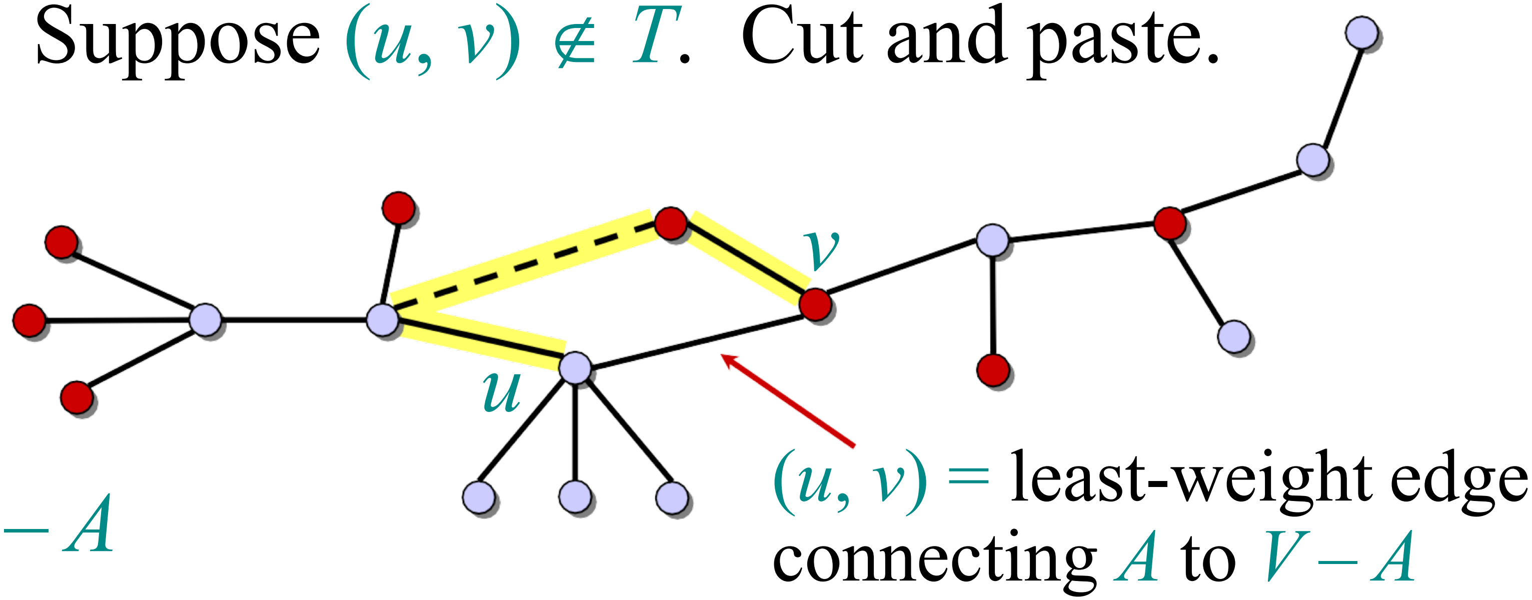
Consider the unique simple path from u to v in T.



Proof of theorem

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

T:

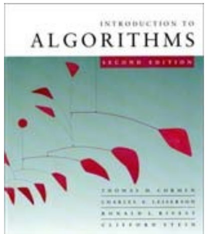


● $\in A$

● $\in V - A$

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

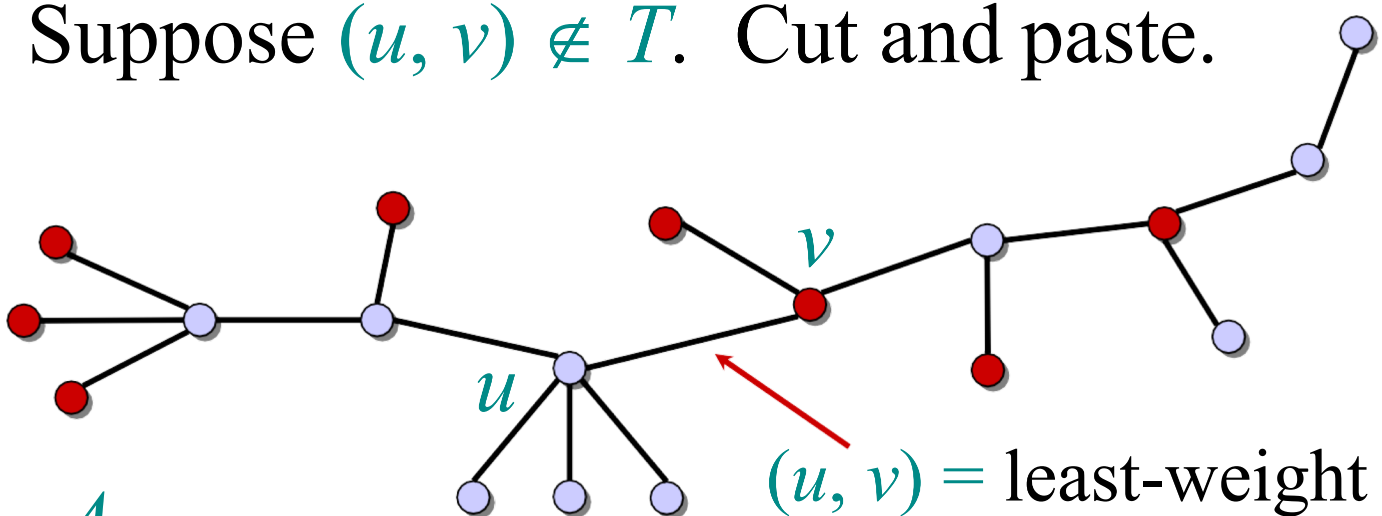
Swap (u, v) with the first edge on this path that connects a vertex in A to a vertex in $V - A$.



Proof of theorem

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

T:



$\in A$

$\in V - A$

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

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

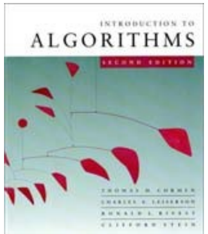
Swap (u, v) with the first edge on this path that connects a vertex in A to a vertex in $V - A$.

A lighter-weight spanning tree than T results. ∎

Kruskal's Algorithm

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```



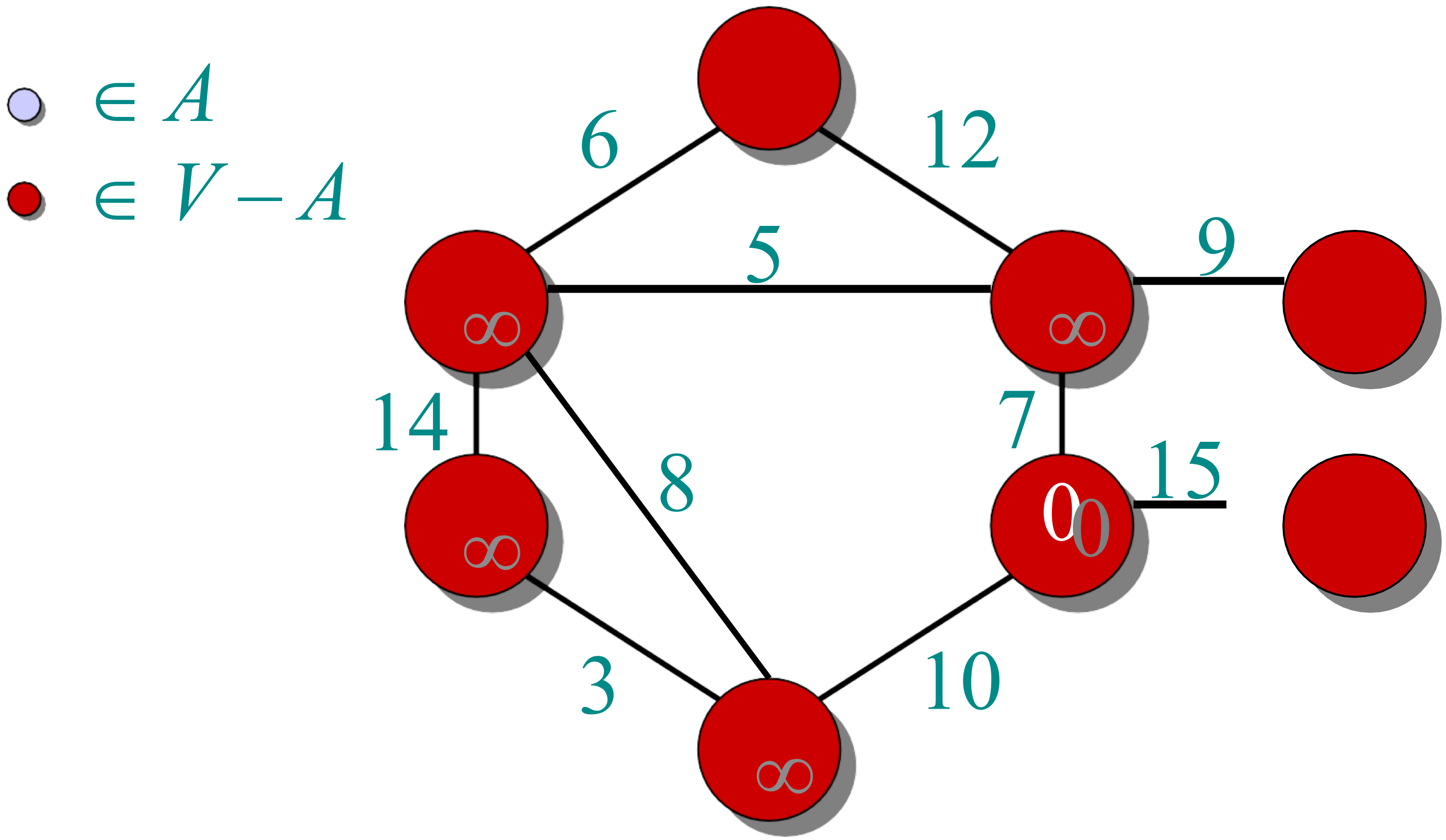
Prim's algorithm

IDEA: Maintain $V - A$ 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.

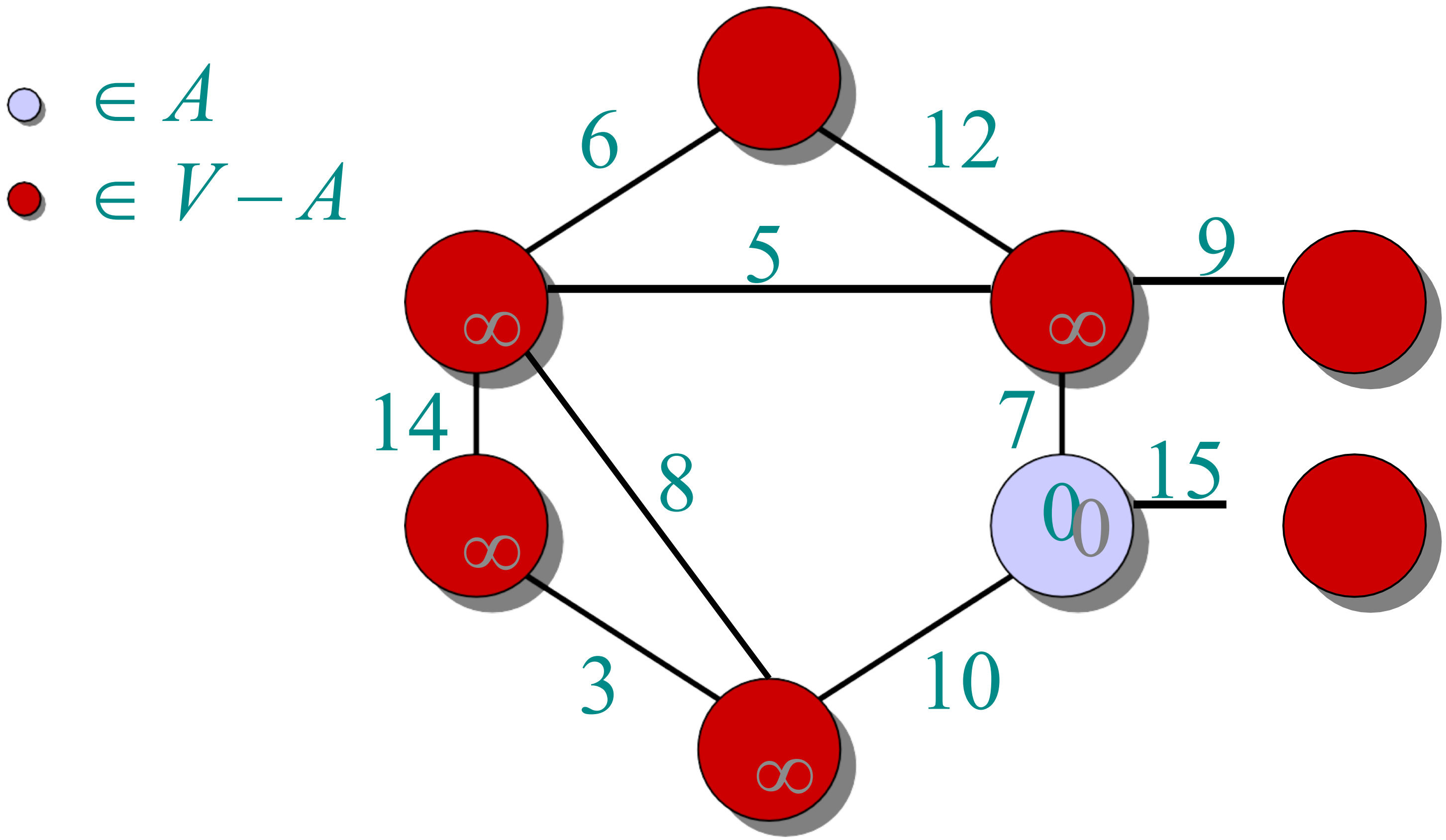
$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        for each $v \in Adj[u]$
            do if $v \in Q$ and $w(u, v) < key[v]$
                then $key[v] \leftarrow w(u,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

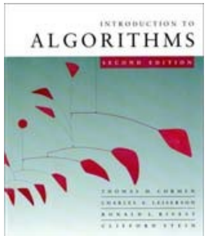
At the end, $\{(v, \pi[v])\}$ forms the MST.

Example of Prim's algorithm



Example of Prim's algorithm

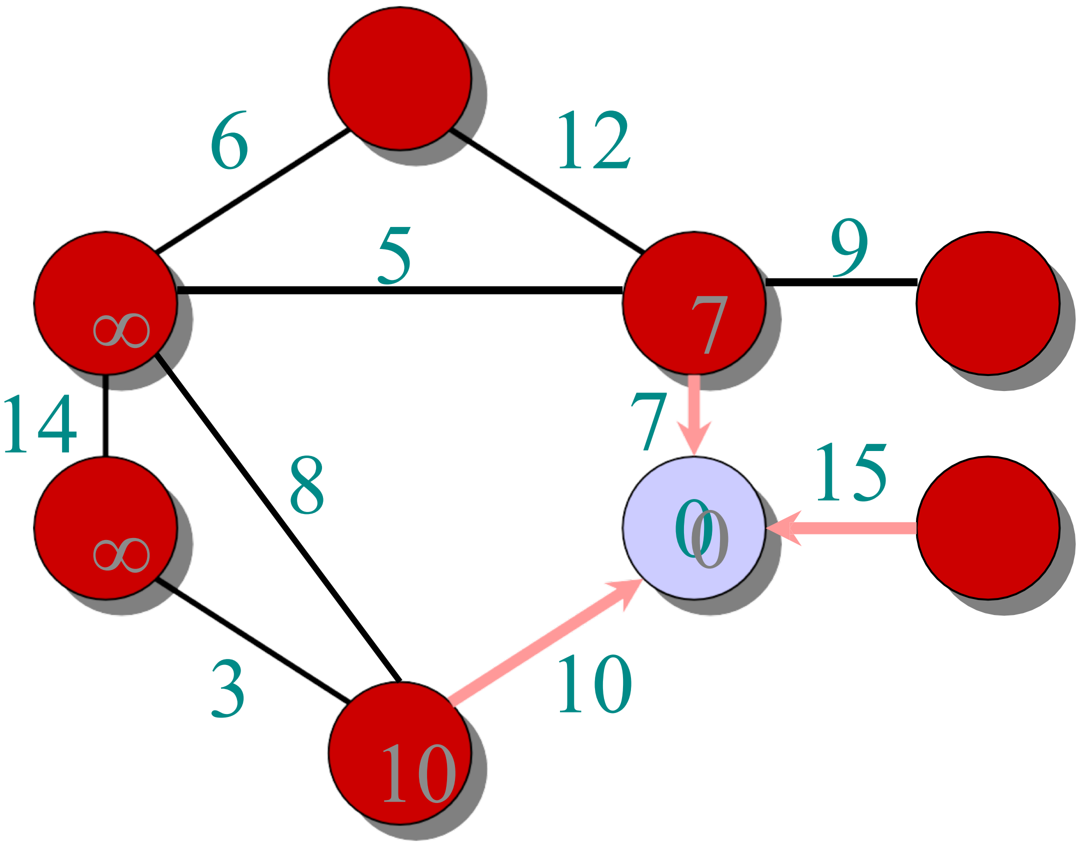


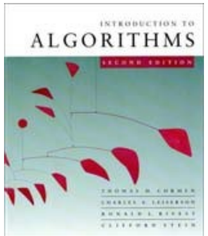


Example of Prim’s algorithm

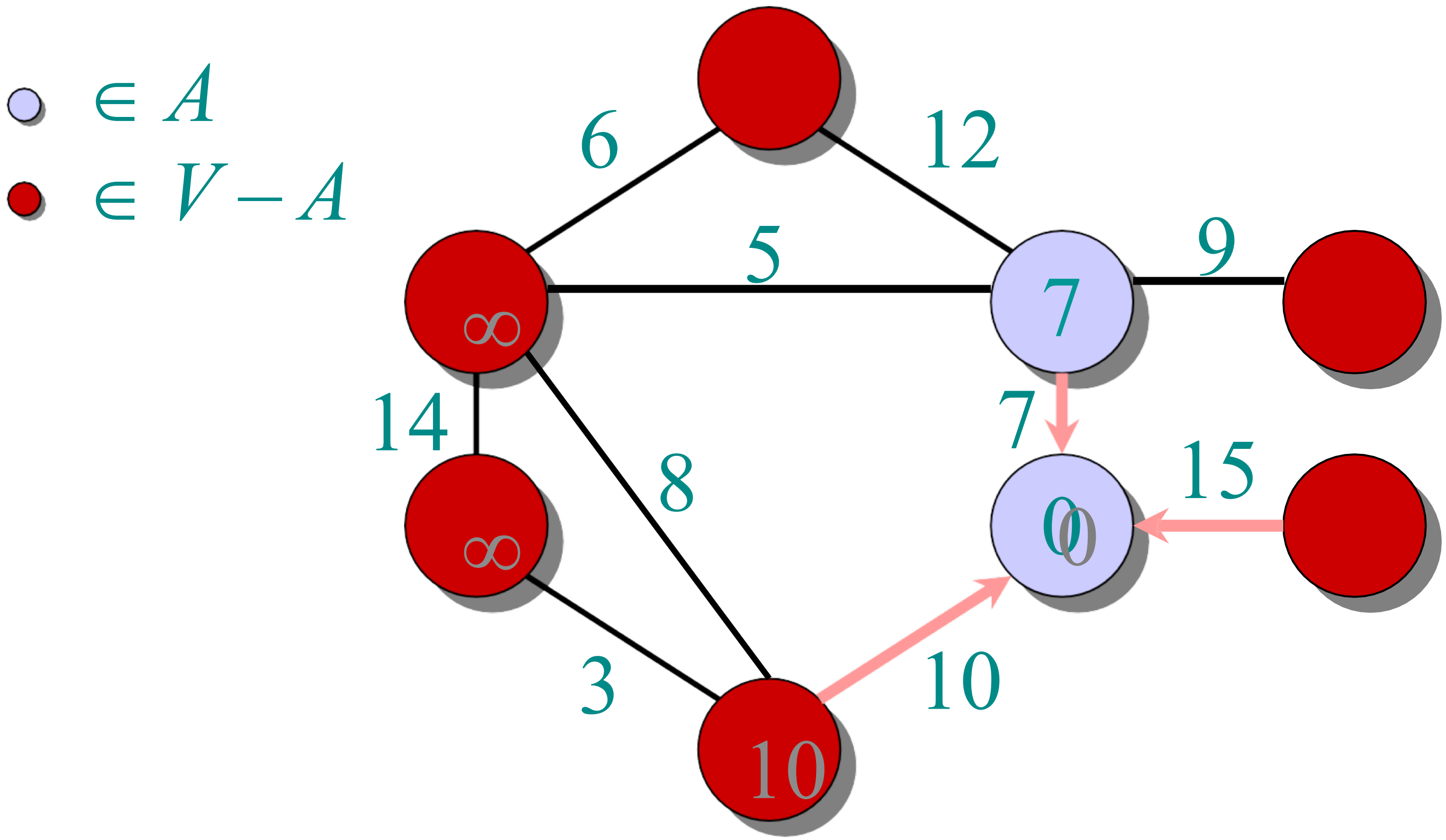
$\in A$

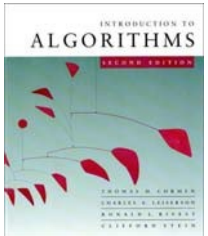
$\in V - A$



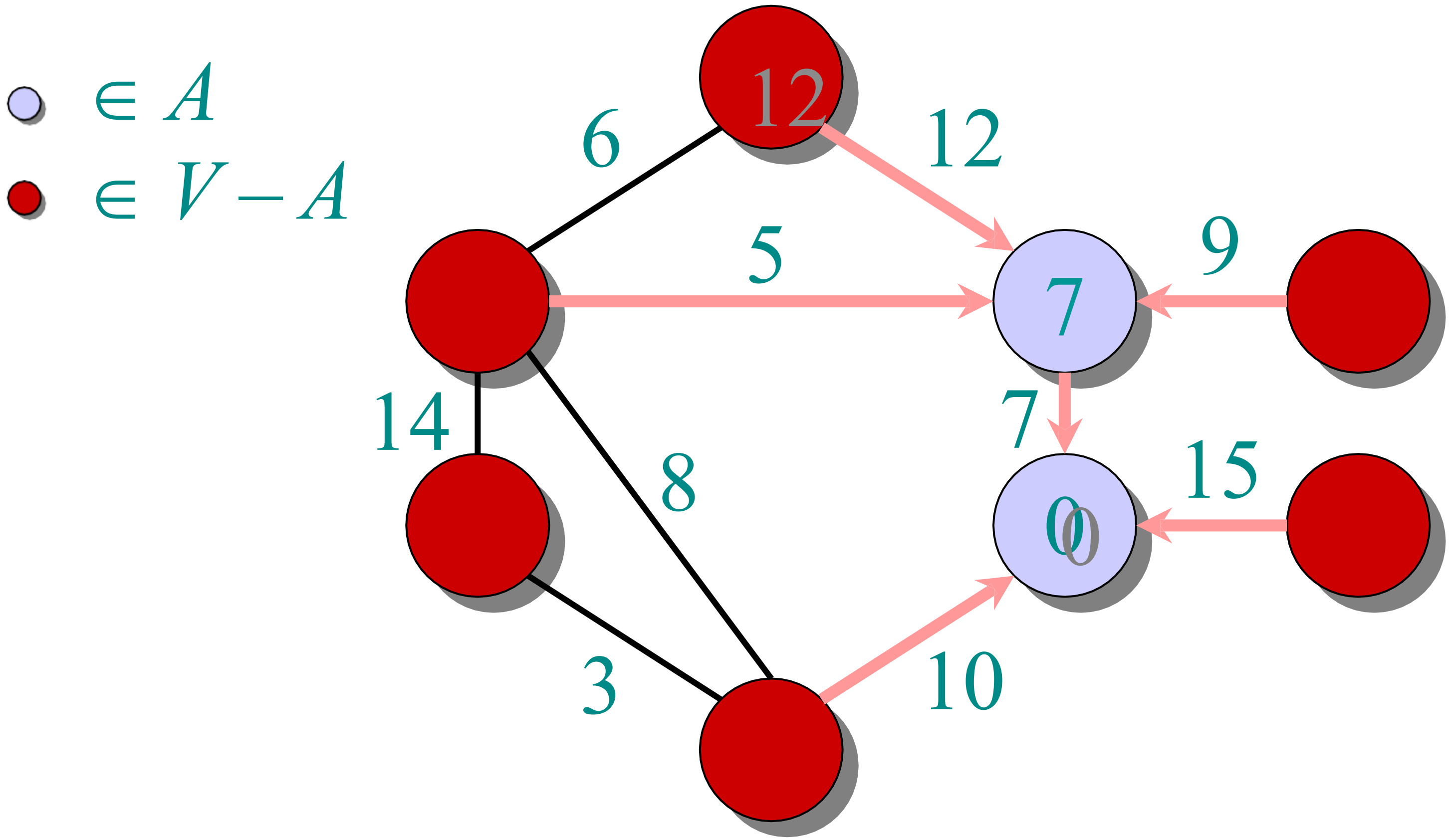


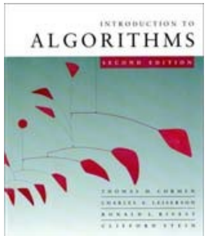
Example of Prim’s algorithm



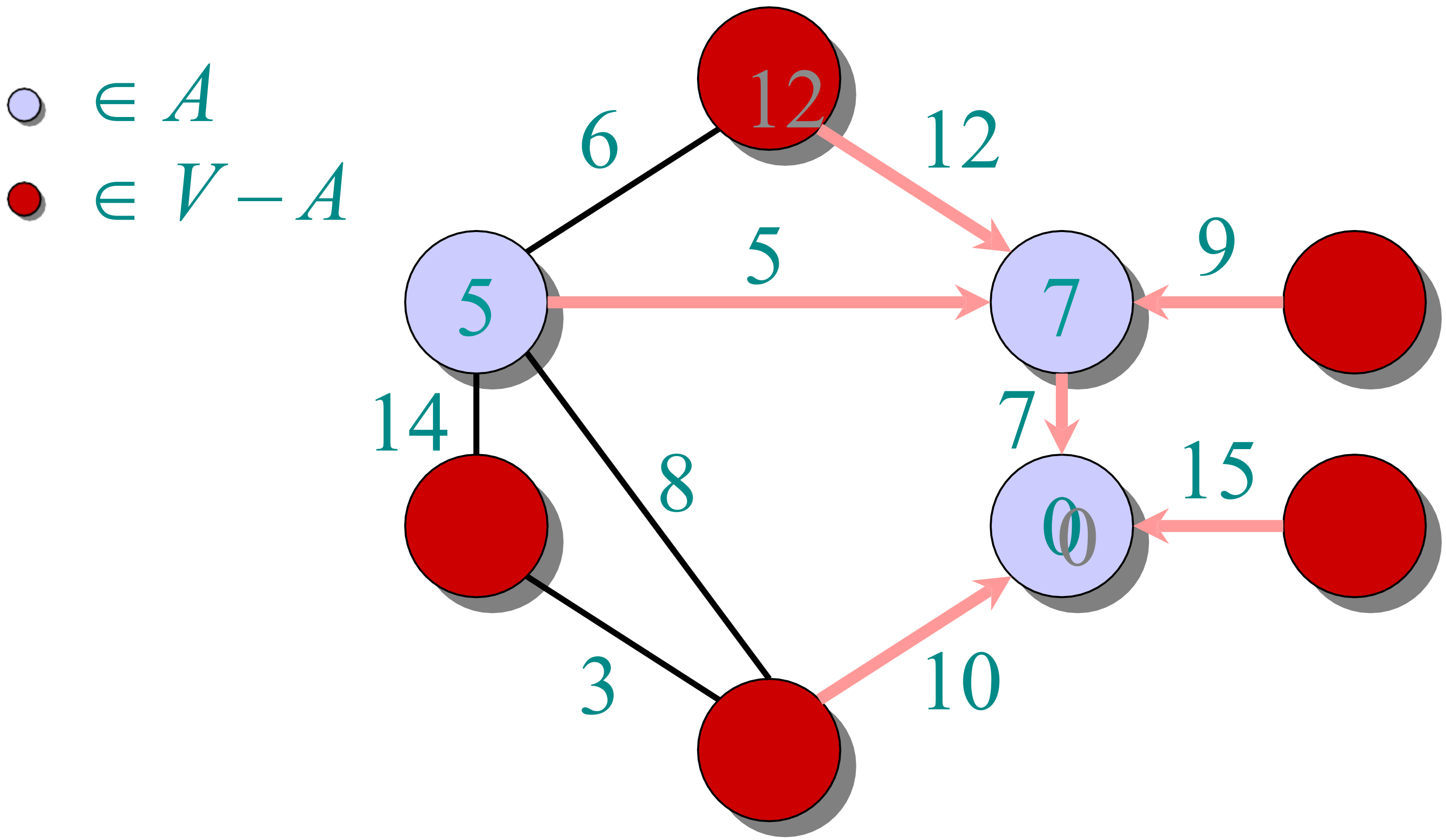


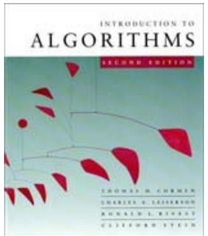
Example of Prim's algorithm



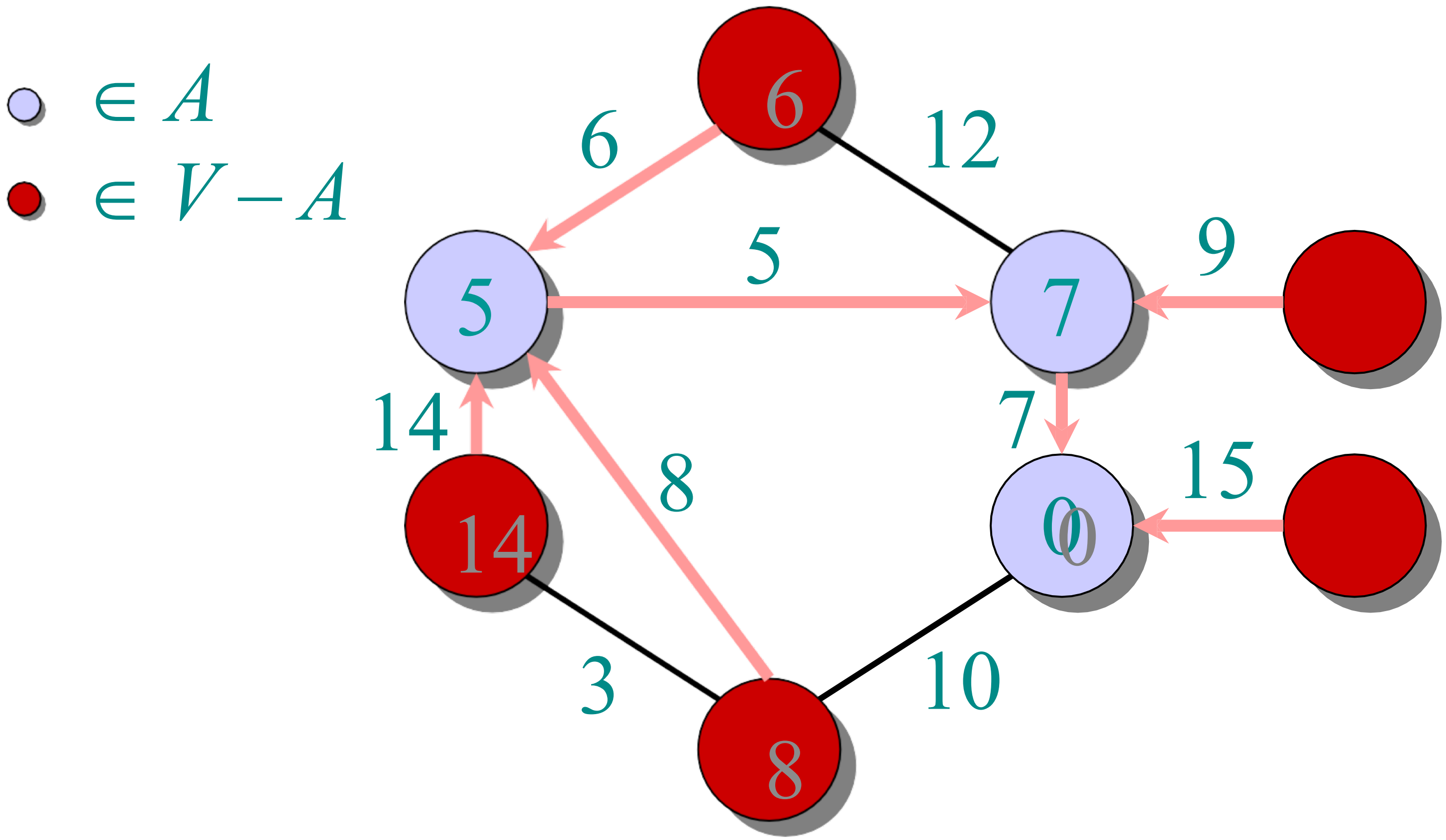


Example of Prim's algorithm

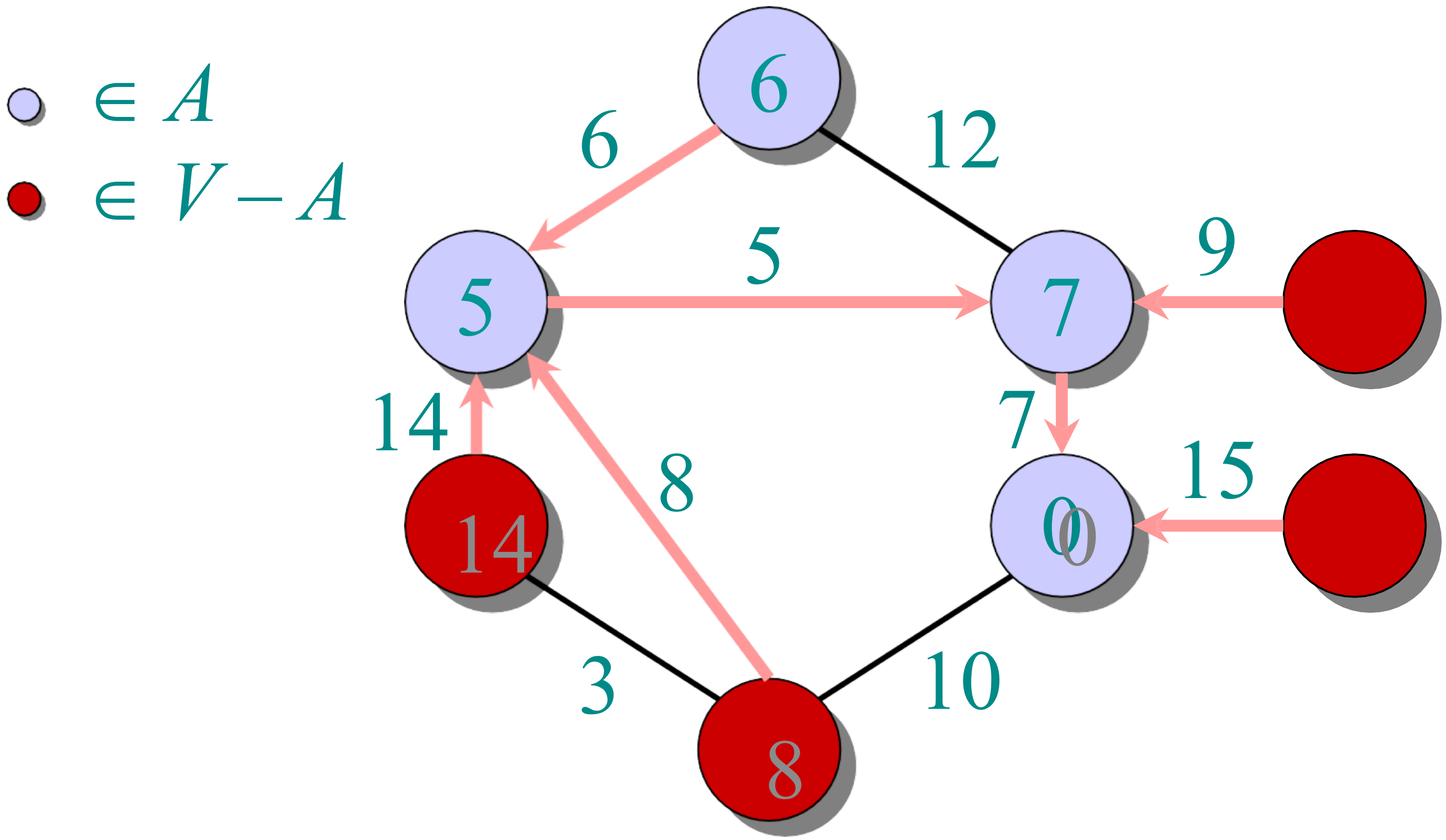


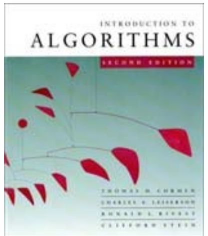


Example of Prim’s algorithm

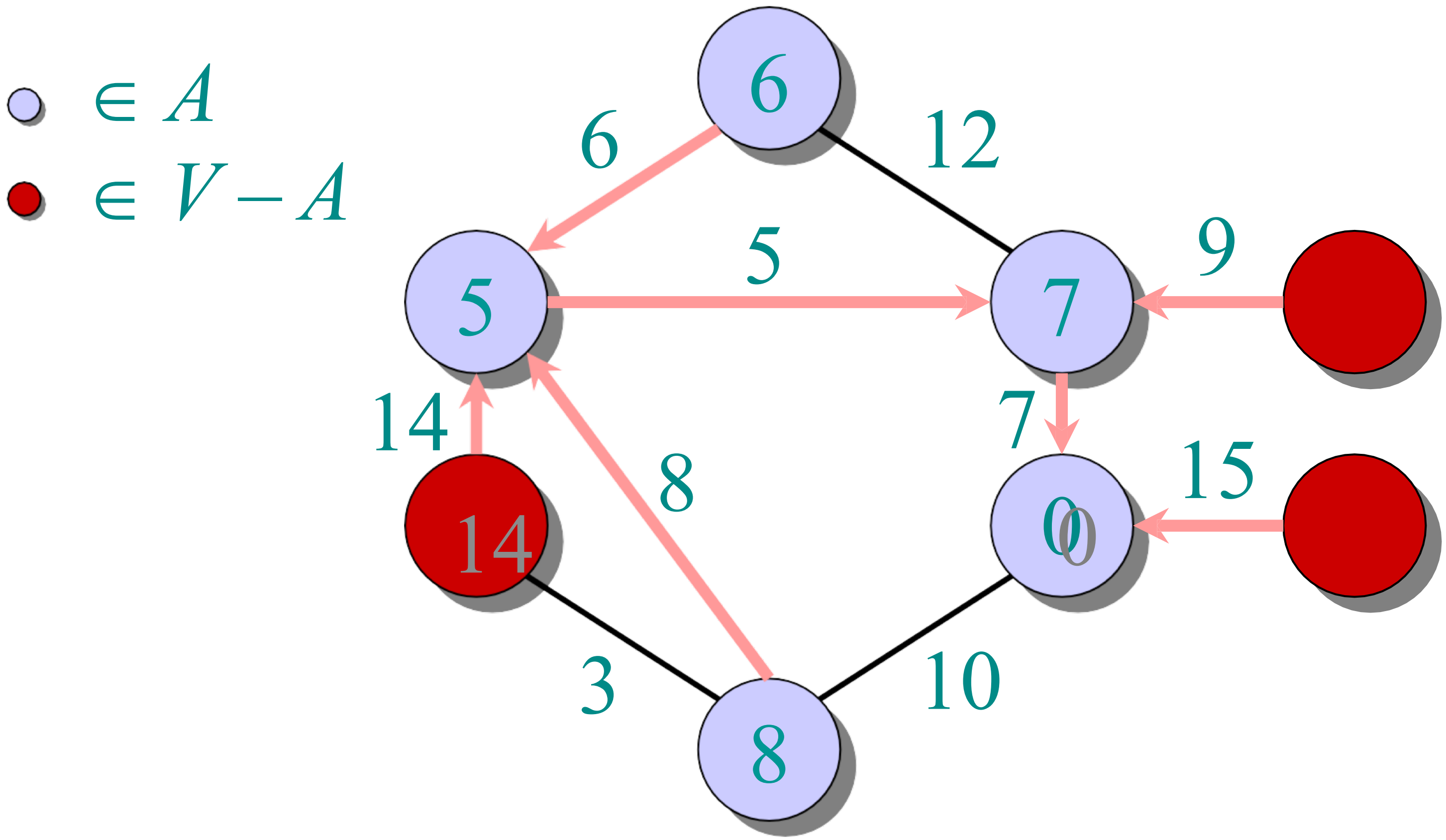


Example of Prim’s algorithm

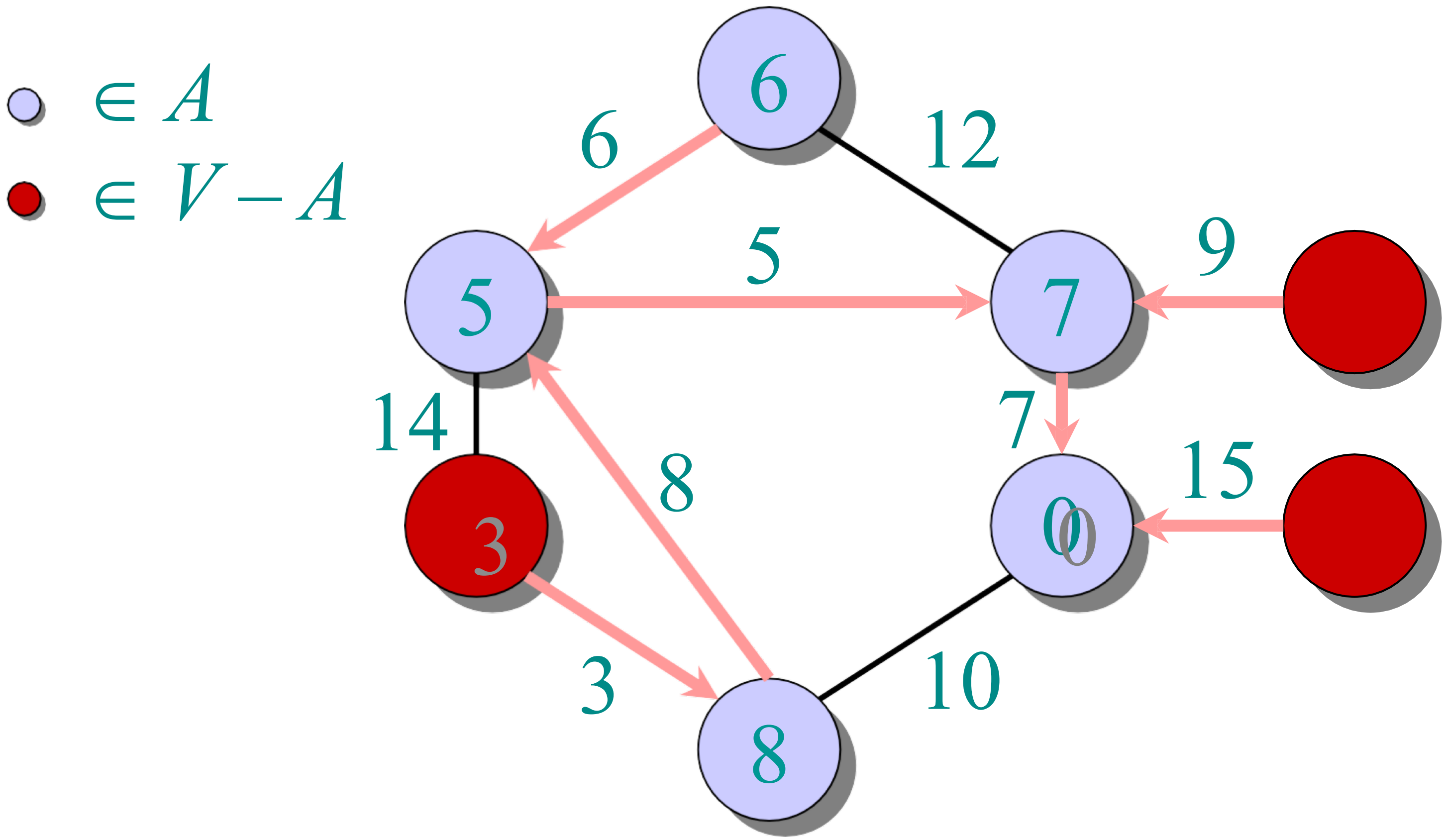




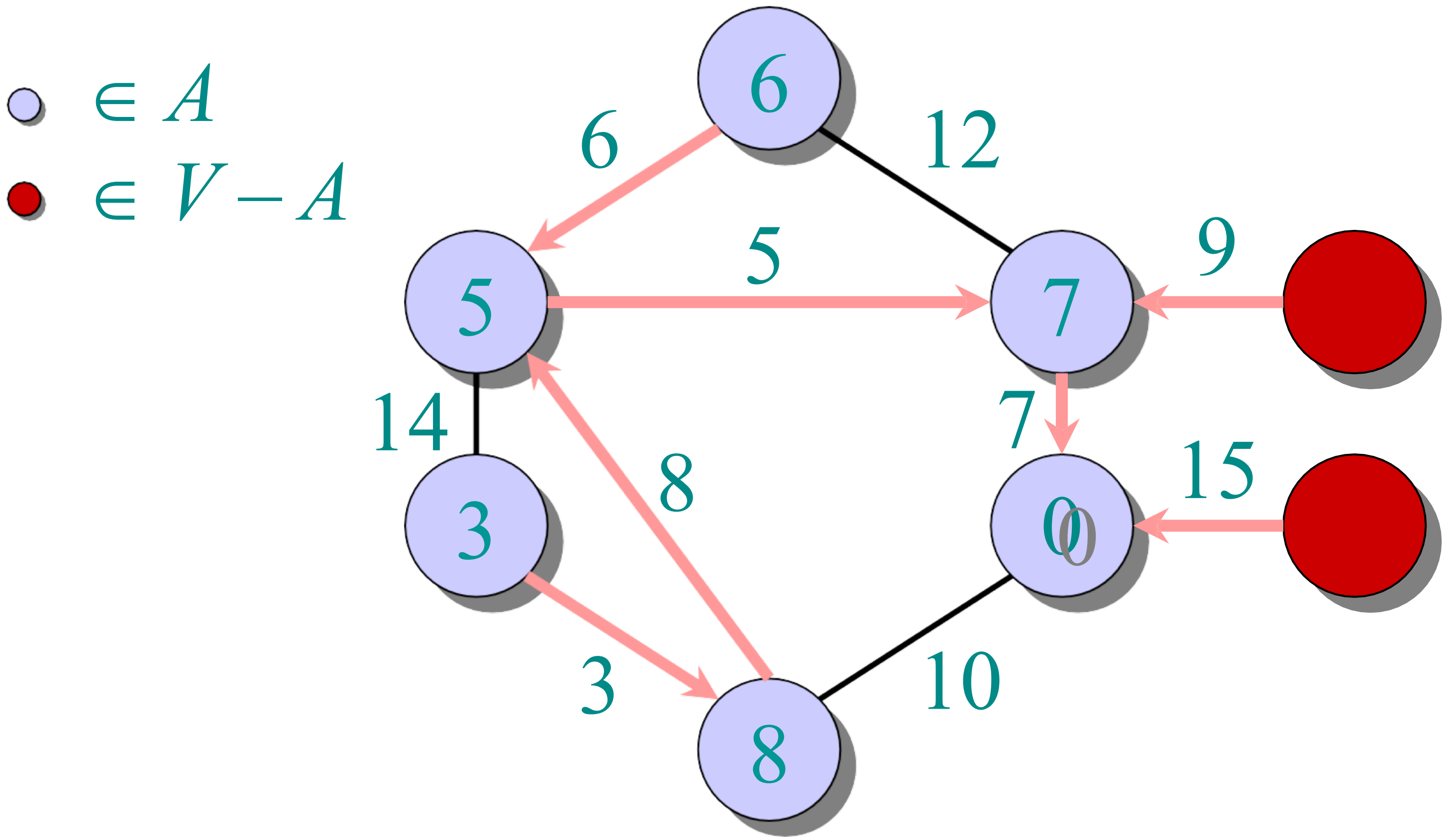
Example of Prim's algorithm



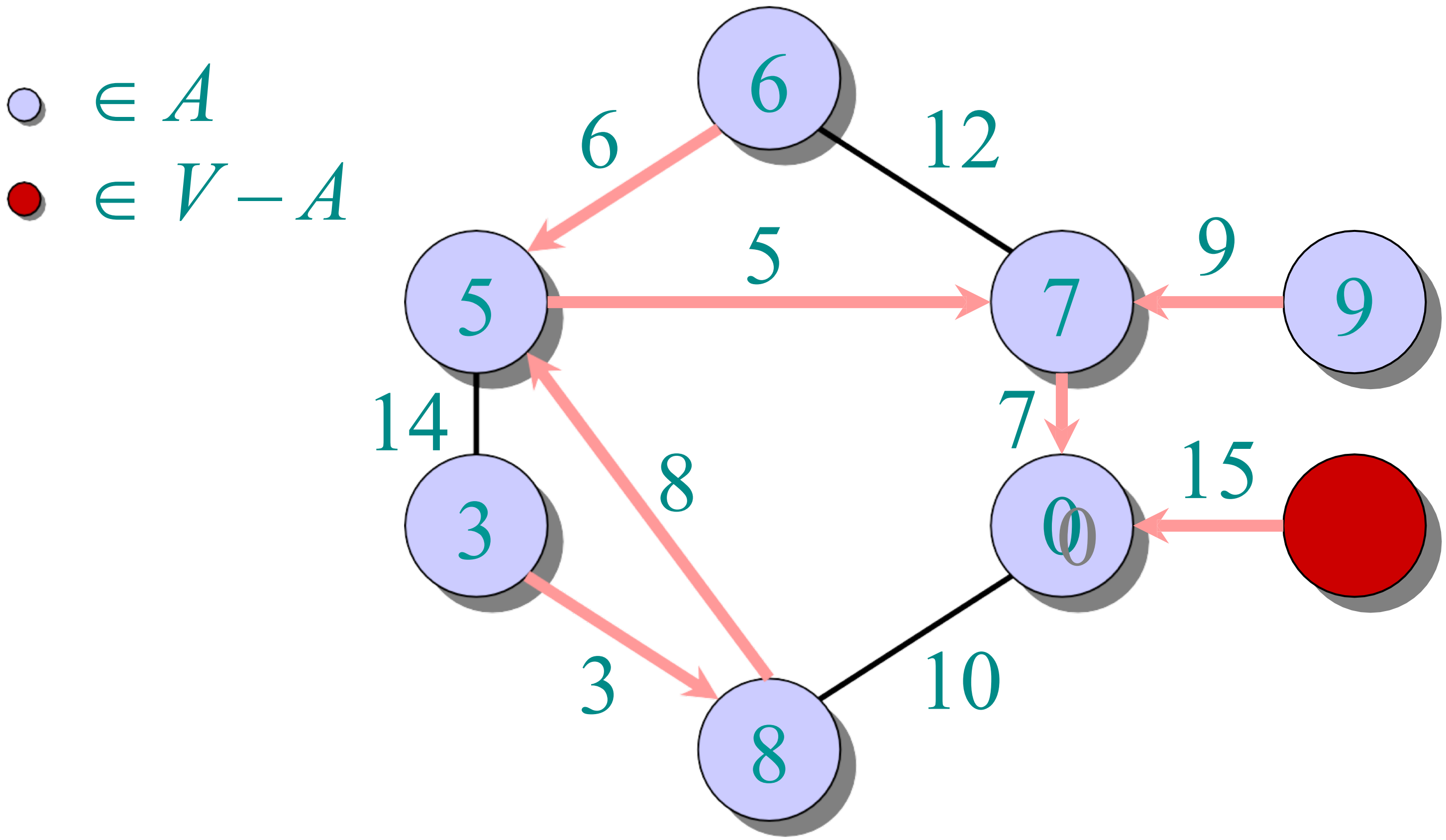
Example of Prim's algorithm

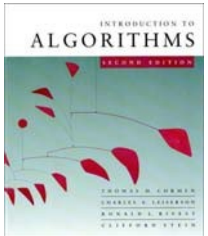


Example of Prim's algorithm

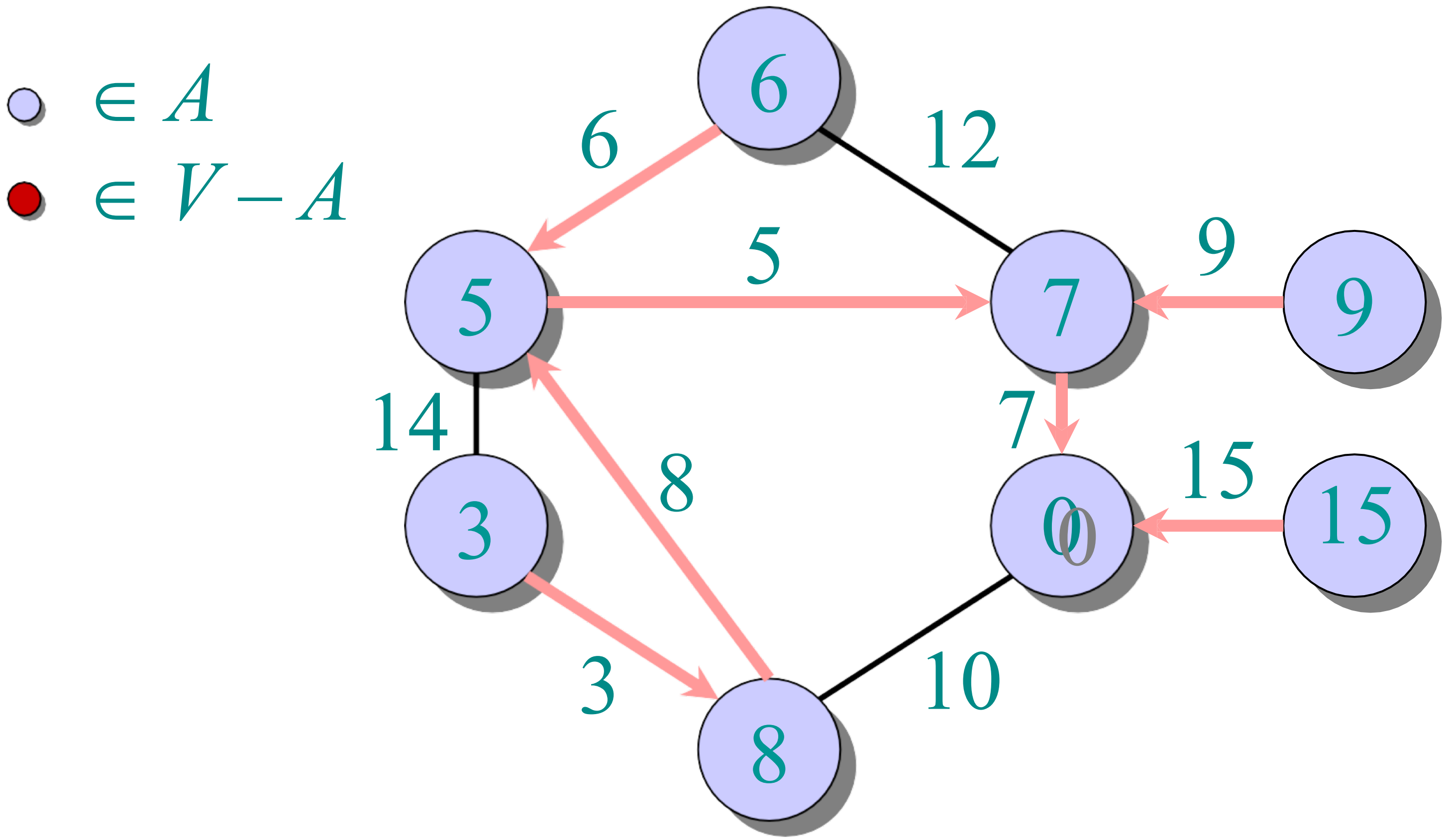


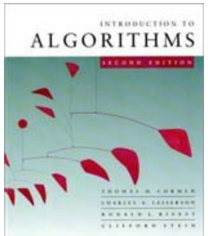
Example of Prim's algorithm





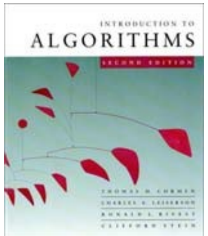
Example of Prim's algorithm





Analysis of Prim

$Q \leftarrow V$
$key[v] \leftarrow \infty$ 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 $key[v] \leftarrow w(u, v)$
                    $\pi[v] \leftarrow u$



Analysis of Prim

$\Theta(V)$ total $\left\{ \begin{array}{l} Q \leftarrow V \\ key[v] \leftarrow \infty \text{ for all } v \in V \\ key[s] \leftarrow 0 \text{ for some arbitrary } s \in V \end{array} \right.$

while $Q \neq \emptyset$

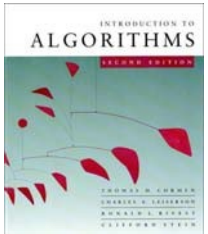
  do $u \leftarrow$ EXTRACT-MIN(Q)

    for each $v \in Adj[u]$

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

        then $key[v] \leftarrow w(u, v)$

          $\pi[v] \leftarrow u$



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 $Q \neq \emptyset$

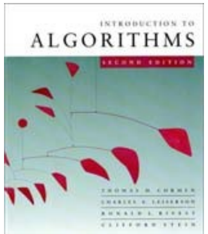
  do $u \leftarrow$ EXTRACT-MIN(Q)

    for each $v \in Adj[u]$

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

        then $key[v] \leftarrow w(u, v)$

          $\pi[v] \leftarrow u$



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 $s \in V$

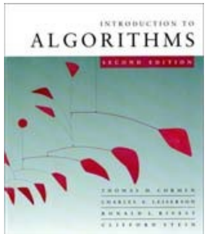
$|V|$ times:

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** $key[v] \leftarrow w(u, v)$
- $\pi[v] \leftarrow u$



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:

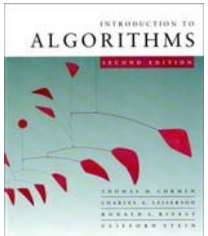
while $Q \neq \emptyset$

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.



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 $Q \neq \emptyset$

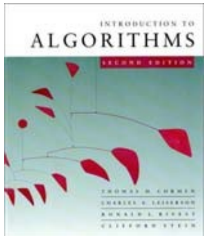
do $u \leftarrow$ EXTRACT-MIN(Q)

$degree(u)$ times:

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

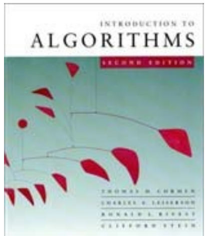
Handshaking Lemma $\Rightarrow \Theta(E)$ implicit DECREASE-KEY's.

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



Analysis of Prim (continued)

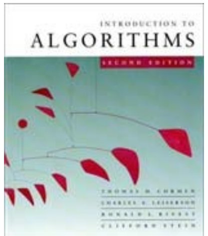
Time $= \Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}}$



Analysis of Prim (continued)

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

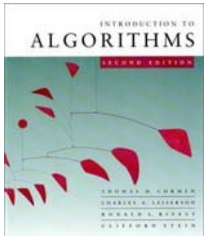
Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total



Analysis of Prim (continued)

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

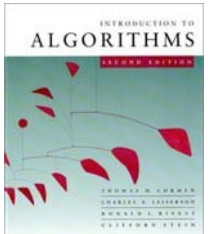
Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$



Analysis of Prim (continued)

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

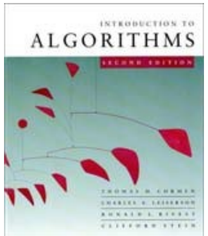
Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$



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	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$ amortized	$O(1)$ amortized	$O(E + V \lg V)$ worst case



MST algorithms

Kruskal's algorithm (see CLRS):

- Uses the ***disjoint-set data structure*** (Lecture 10).
- Running time $= O(E \lg V)$.

MST algorithms

Kruskal's algorithm (see CLRS):

- Uses the ***disjoint-set data structure*** (Lecture 10).
- Running time = $O(E \lg V)$.

Best to date:

- Karger, Klein, and Tarjan [1993].
- Randomized algorithm.
- $O(V + E)$ expected time.