《数据结构与算法》课程组
重庆大学计算机学院

Data Structures & Algorithms

17 MINIMUM SPANNING TREE

Outline

17.1 Definitions and Greedy Property

17.2 Prim Algorithm

17.3 Kruskal Algorithm
-- with disjoint set

17.1 Definitions and Greedy Property

Minimum spanning trees

Input:

-$G = (V, E)$ A connected, undirected graph

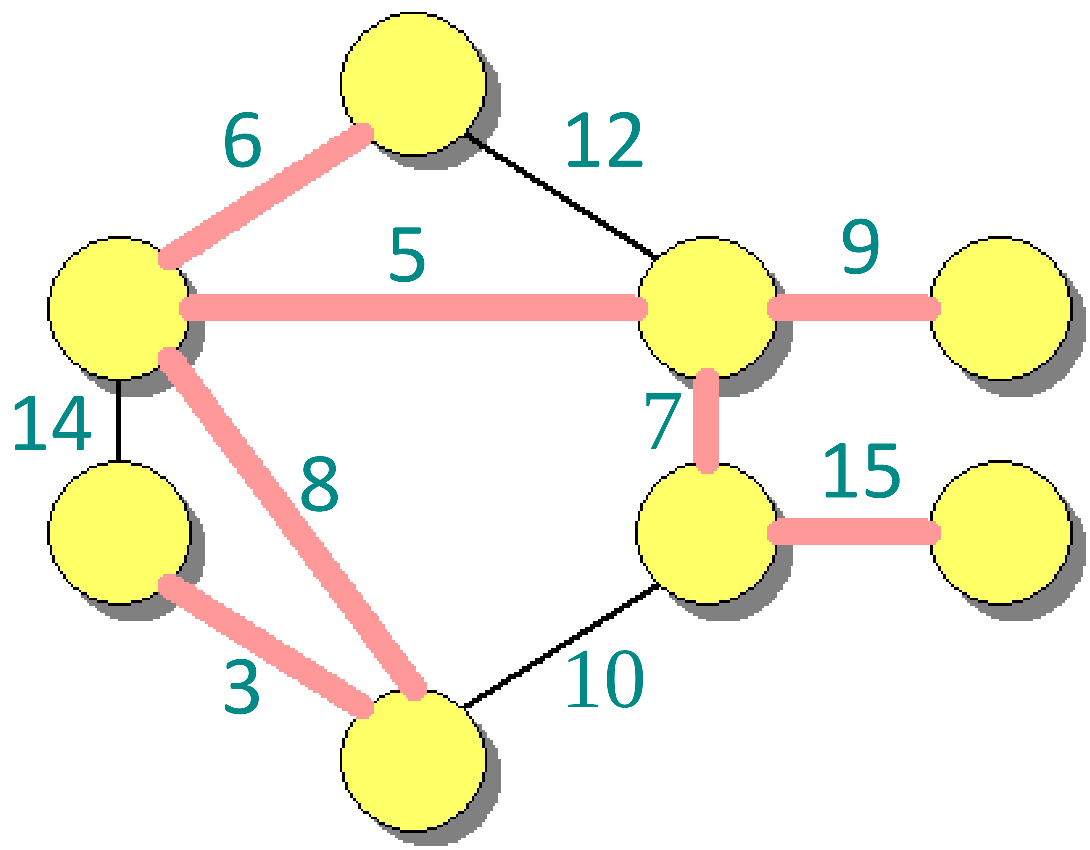
-Weight function $w: E \rightarrow R$.

- 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:

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

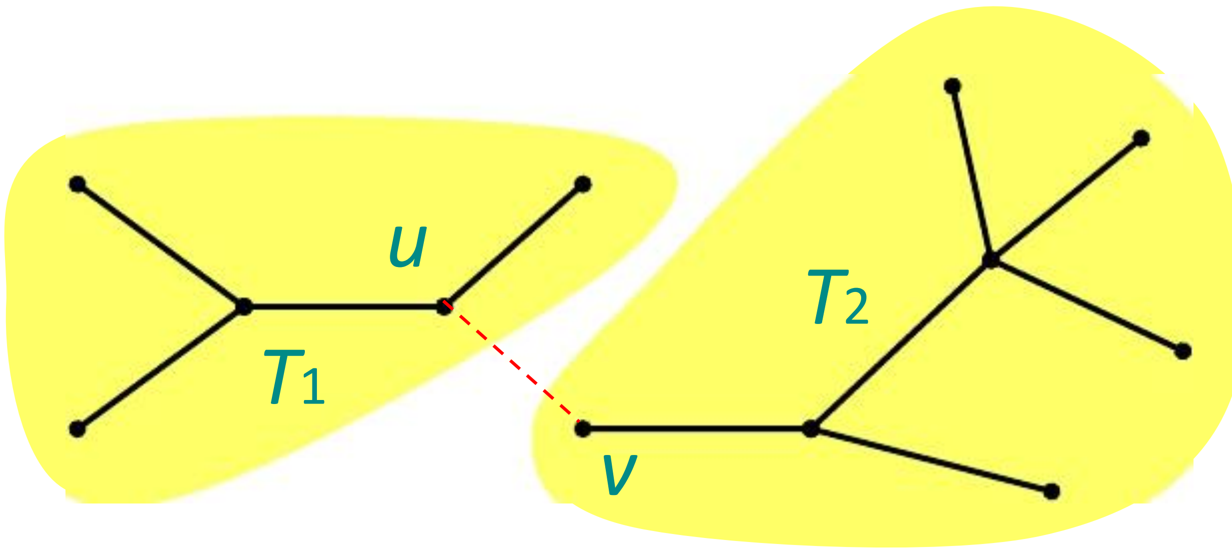
Example of MST



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 $T1$ and $T2$.

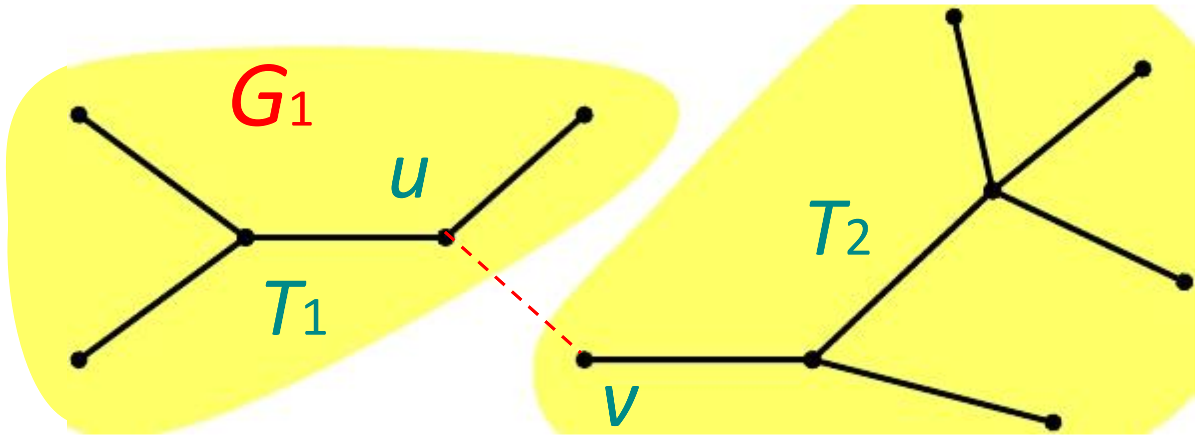
optimal substructure

Theorem.

The subtree T_1 is an MST of $G_1 = (V_1, E_1)$,where

$$V_1 = \text{vertices of } T_1,$$

$$E_1 = \{(x, y) \in E \mid x, y \in V_1\}.$$



optimal substructure

Proof.

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

If T^1 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.

Do we also have overlapping subproblems?

- Yes. Great, then dynamic programming may work!

- but MST leads to an even more efficient algorithm.

Greedy Property

Greedy-choice property

A locally optimal choice is globally optimal.

Theorem.

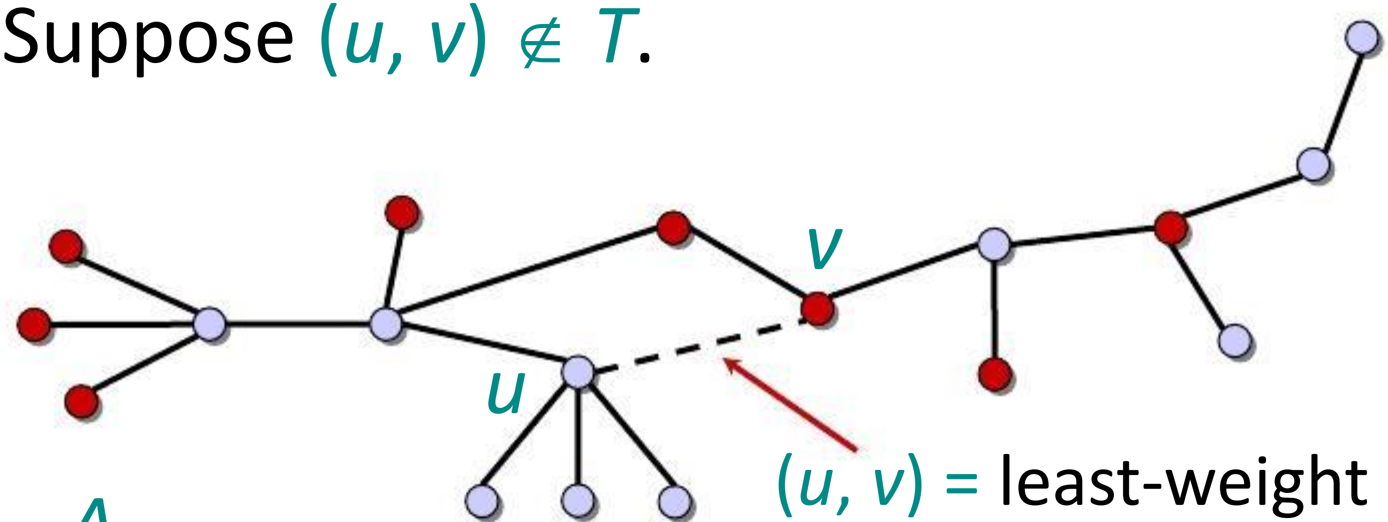
Let T be the MST of $G = (V, E)$ and let $A \subseteq V$. Suppose that $(u, v) \in E$ is the least-weight edge connecting A to $V - A$. Then, $(u, v) \in T$.

Greedy Property

Proof of theorem

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

T:



$\in A$

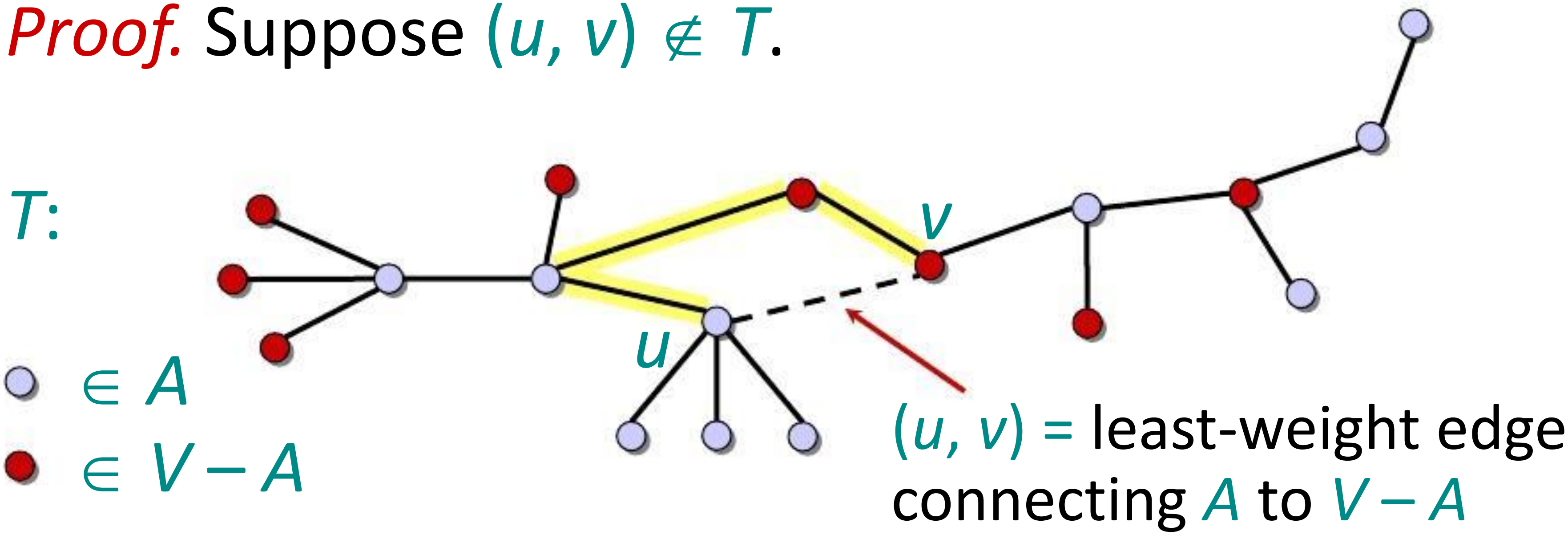
$\in V - A$

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

Greedy Property

Proof of theorem

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



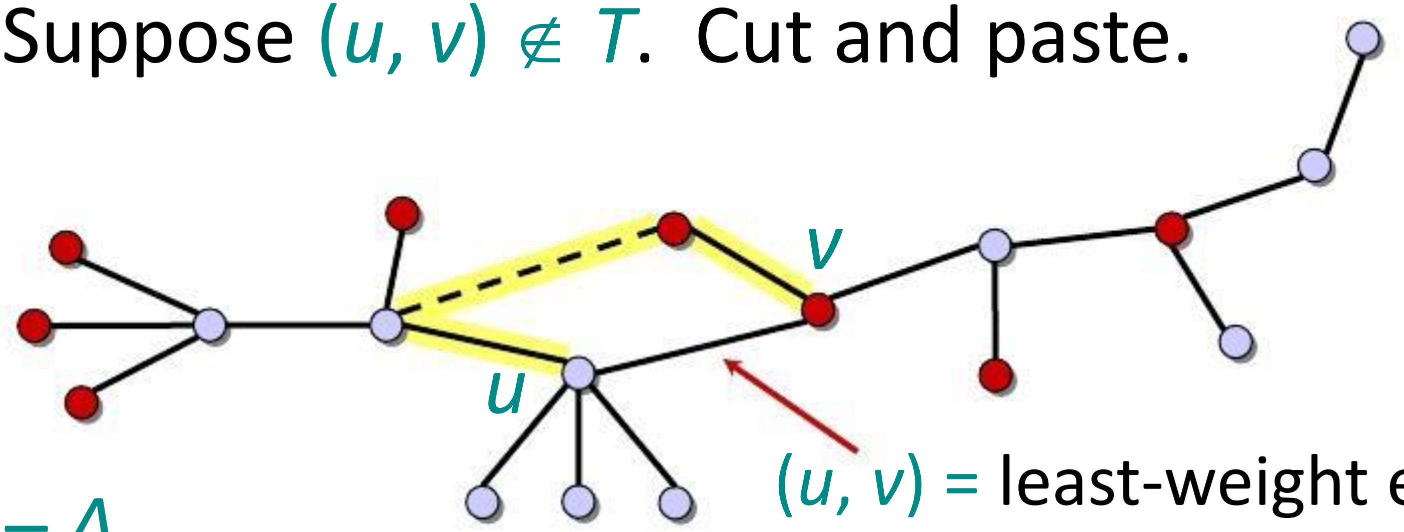
Consider the unique simple path from u to v in T.

Greedy Property

Proof of theorem

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

T:



● $\in A$

● $\in V - A$

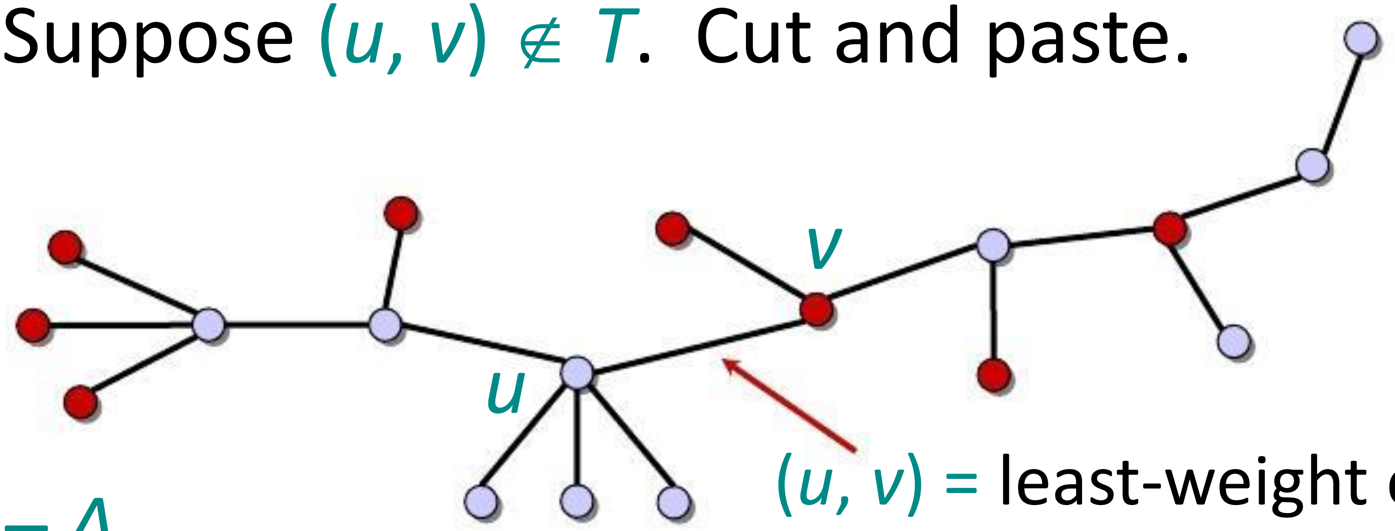
Swap (u, v) with the first edge on this path that connects a vertex in A to a vertex in $V - A$.

Greedy Property

Proof of theorem

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

$T2$:



$\in A$

$\in V - A$

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

A lighter-weight spanning tree than T results.

17.2 Prim Algorithm

Prim's algorithm

IDEA:

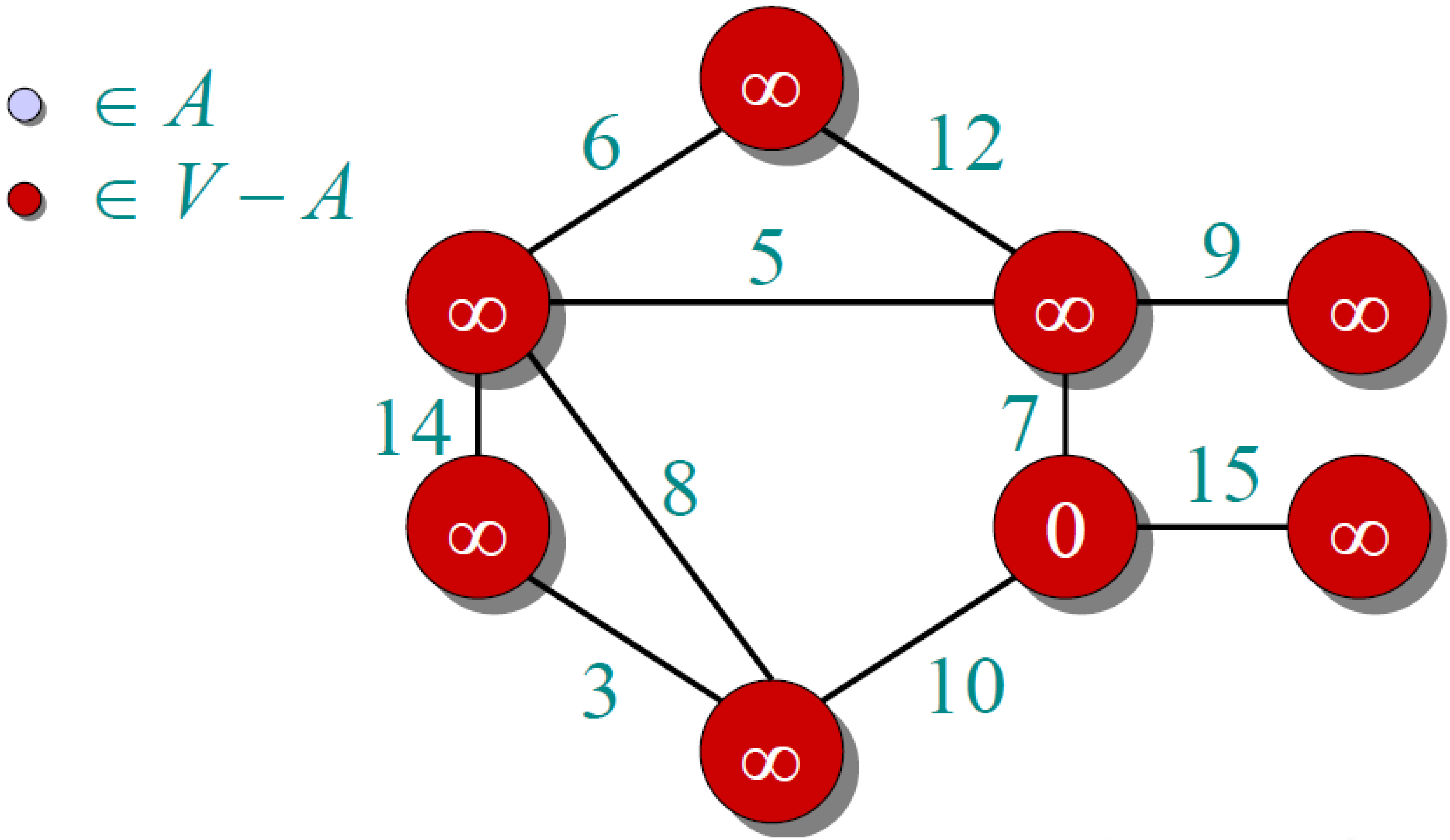
- Maintain $V - A$ as a priority queue Q.
- Key each vertex in Q with the weight of the least-weight edge connecting it to a vertex in A.

Prim's algorithm

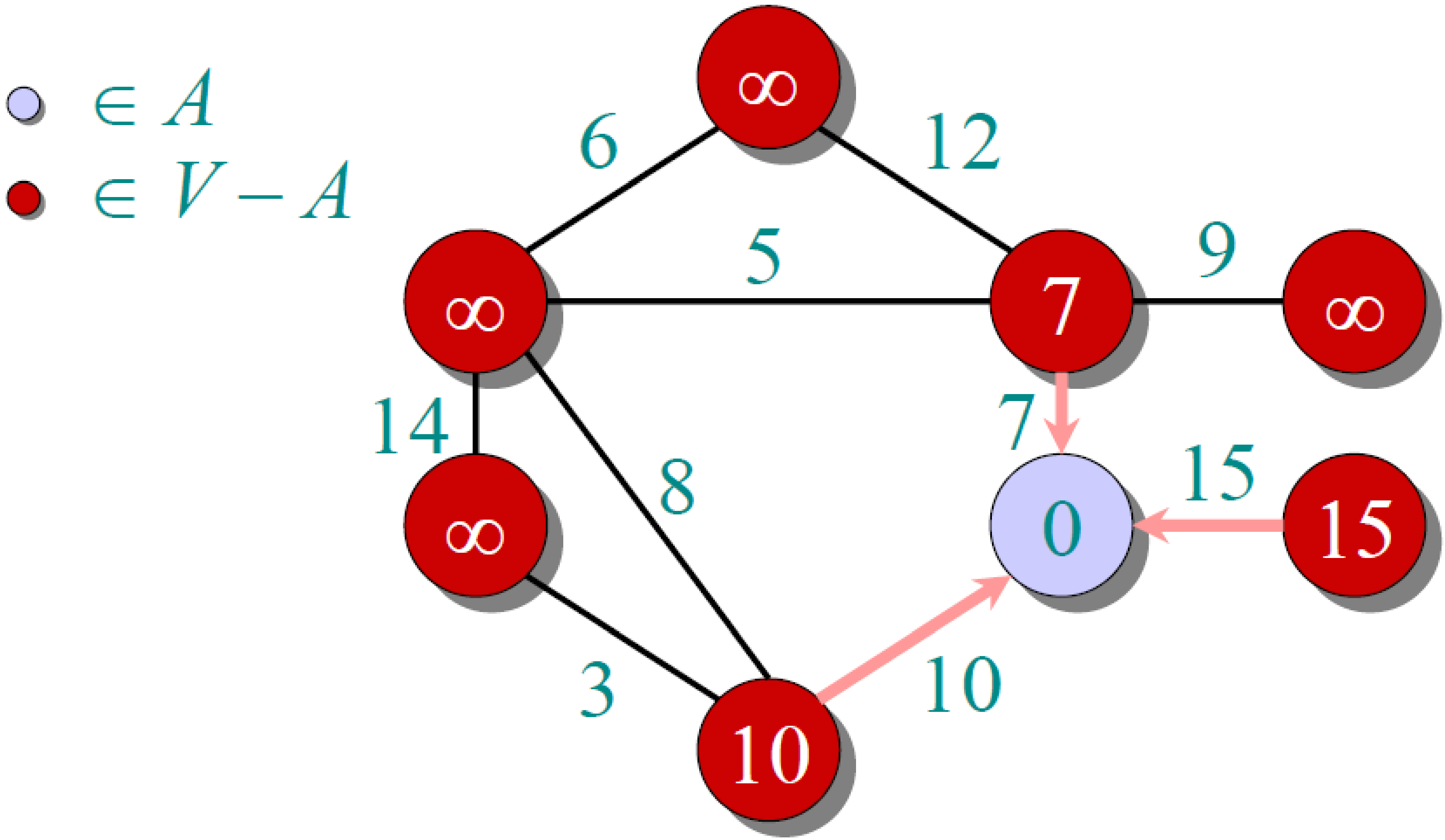
$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    for each $v \in Adj[u]$
      do if $v \in Q$ and $w(u, v) < key[v]$
        then $key[v] \leftarrow w(u, v)$ ▷ DECREASE-KEY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

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

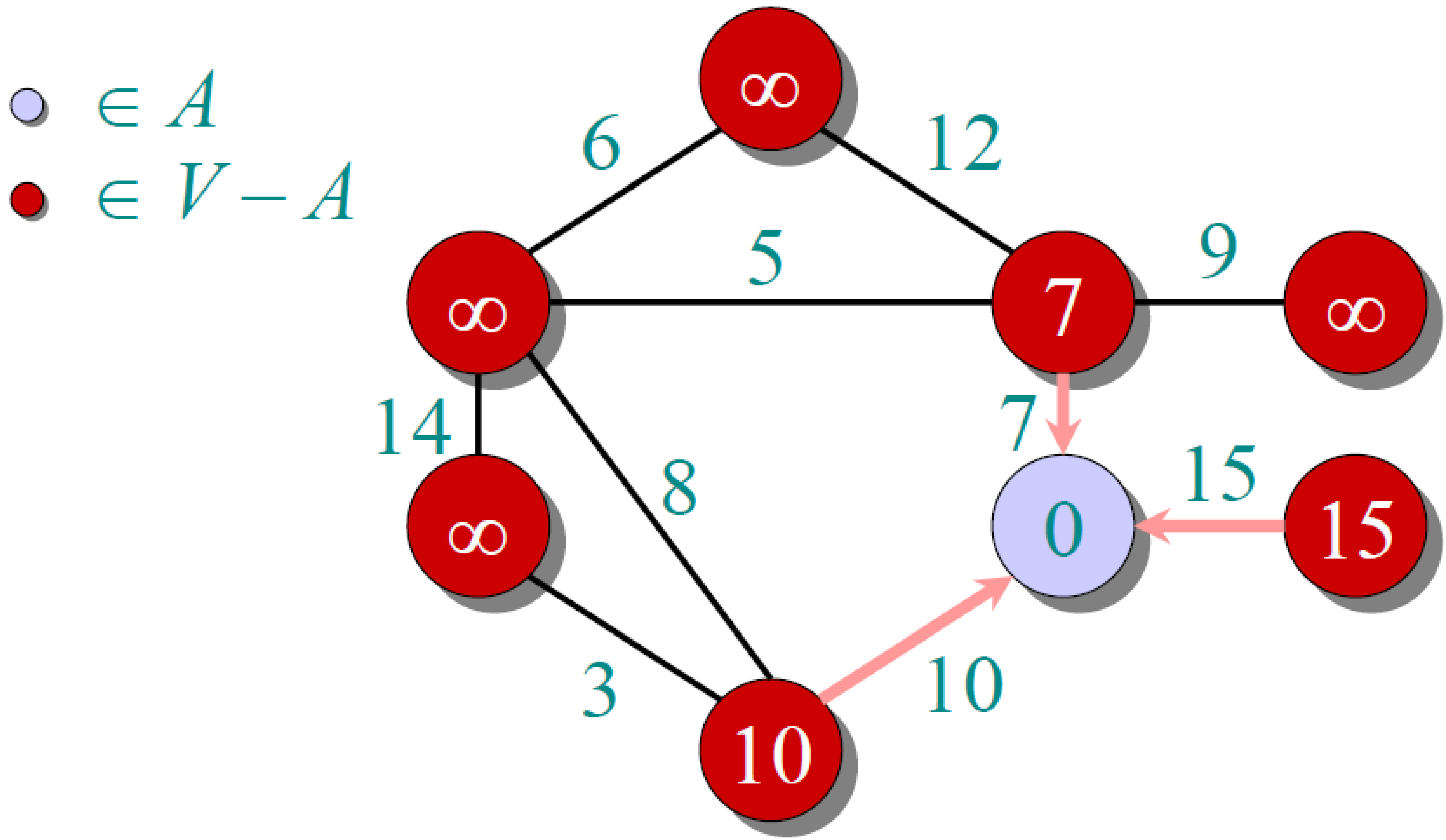
Example of Prim's algorithm



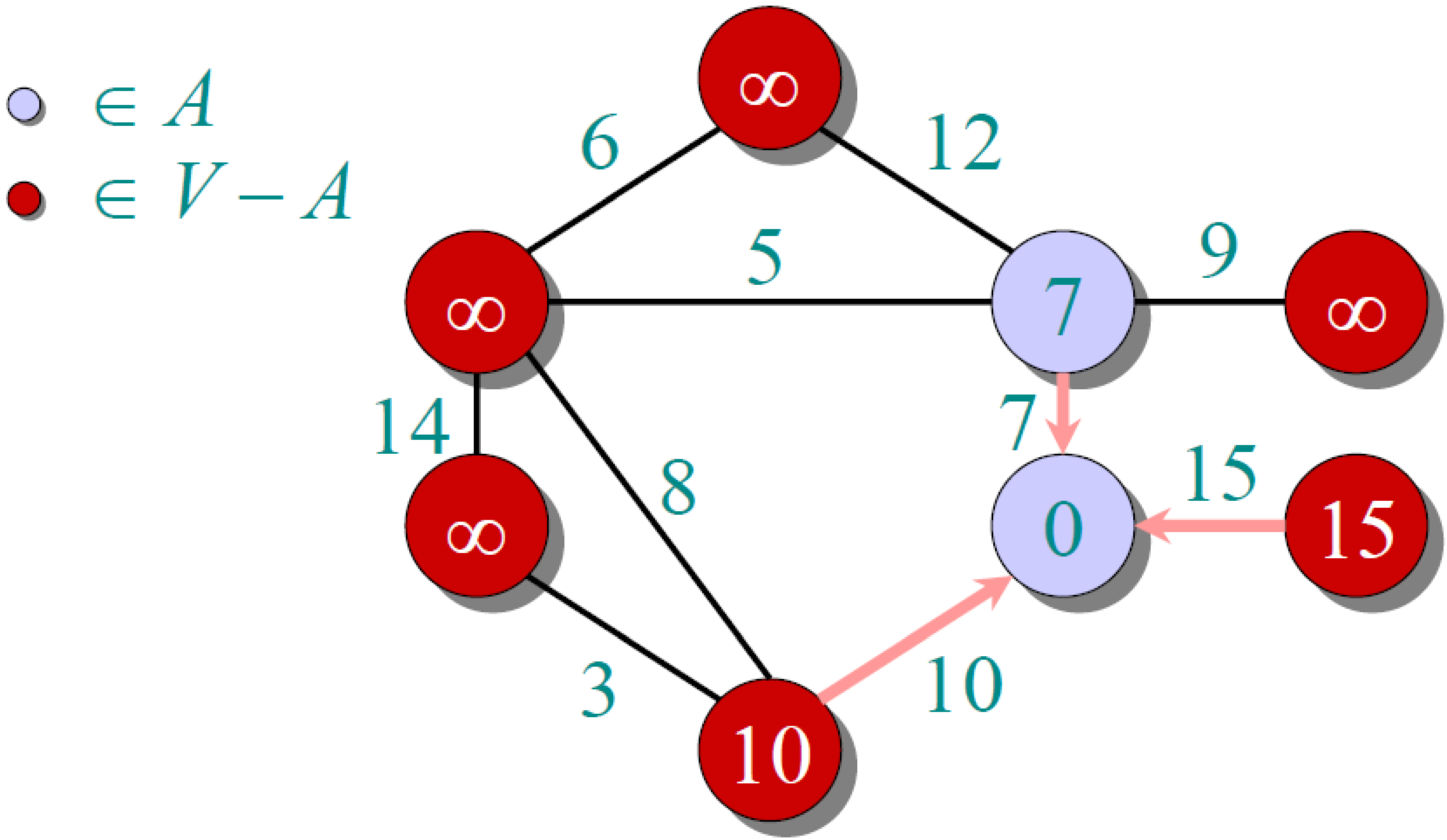
Example of Prim's algorithm



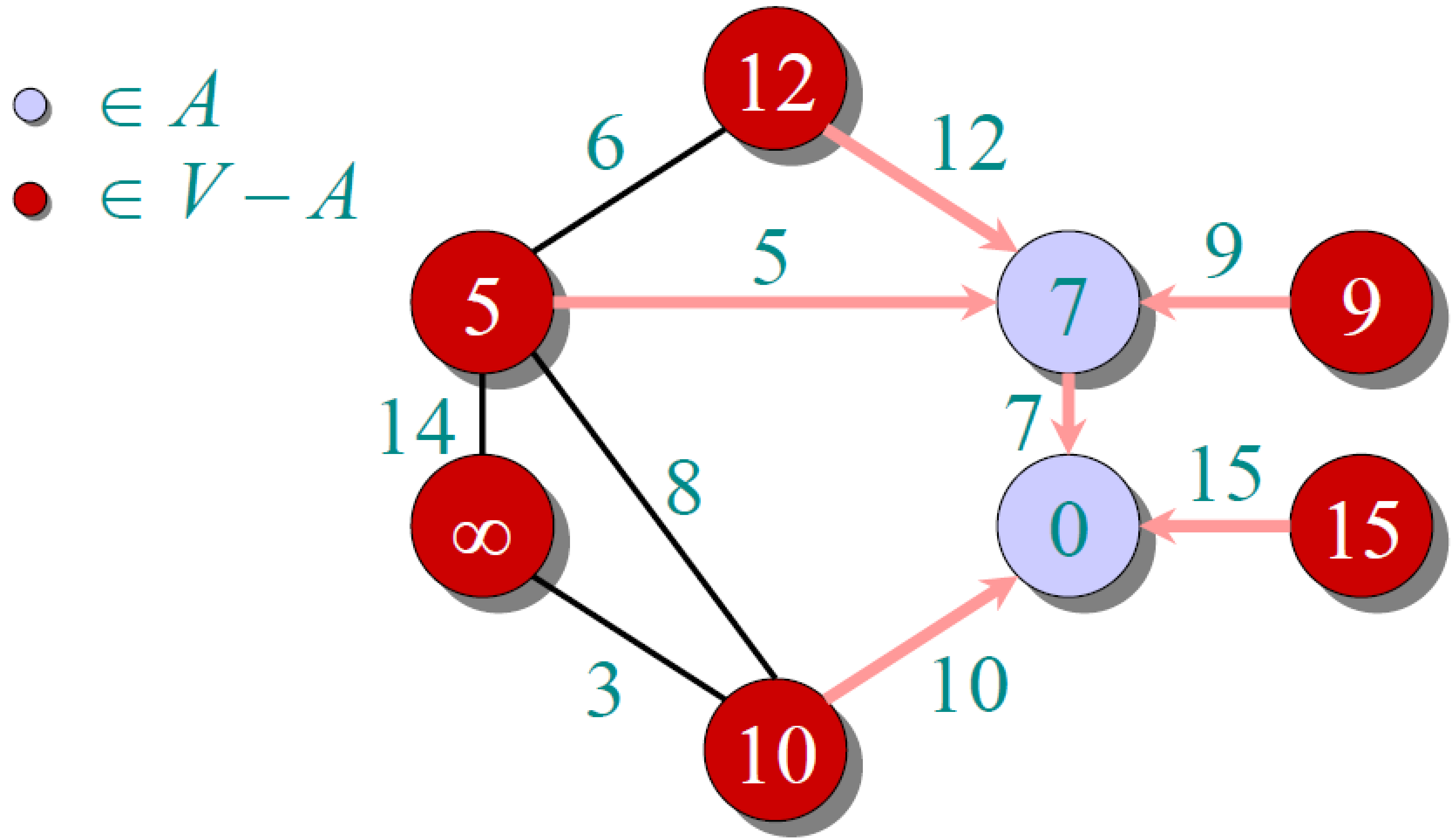
Example of Prim's algorithm



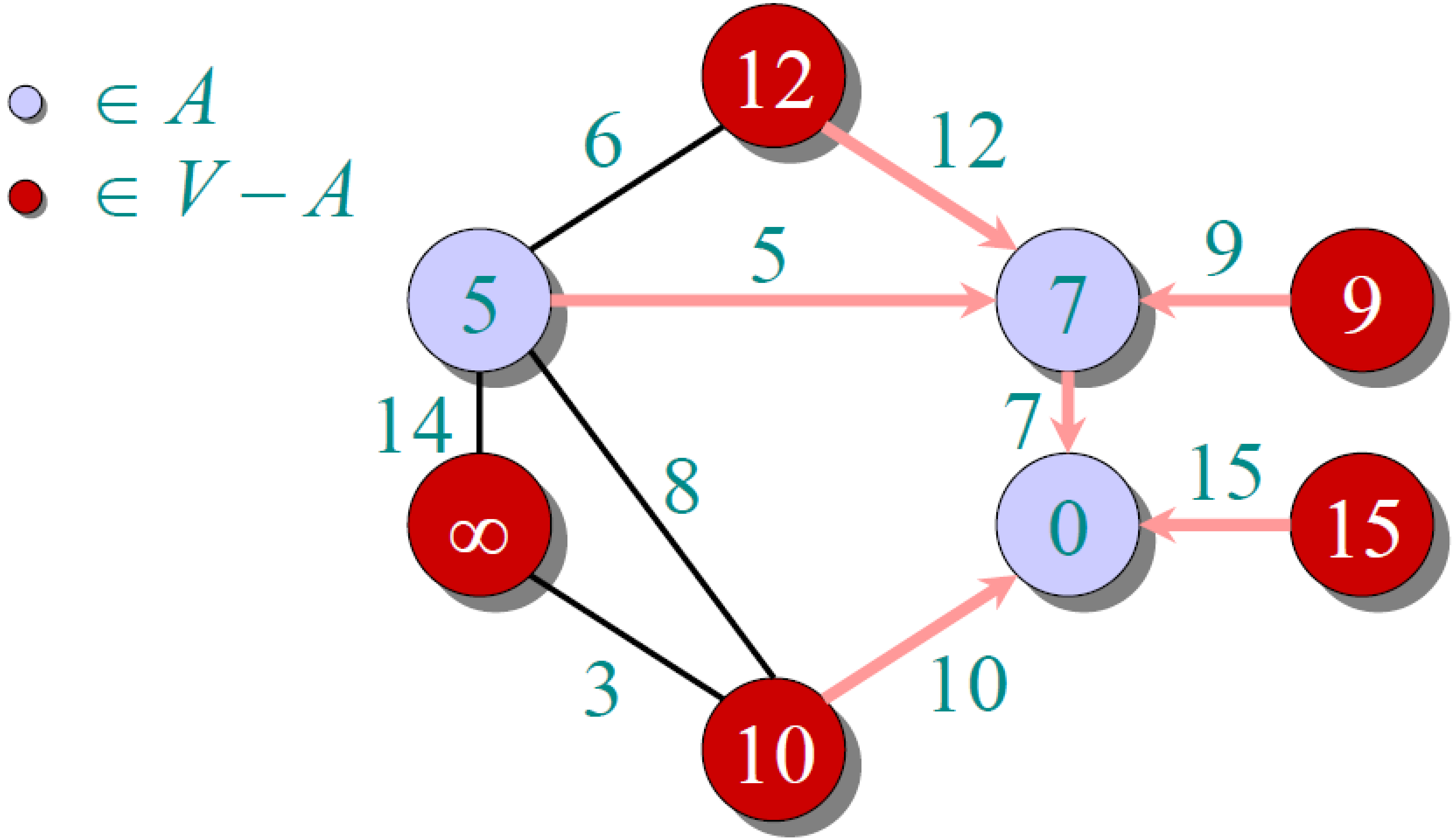
Example of Prim's algorithm



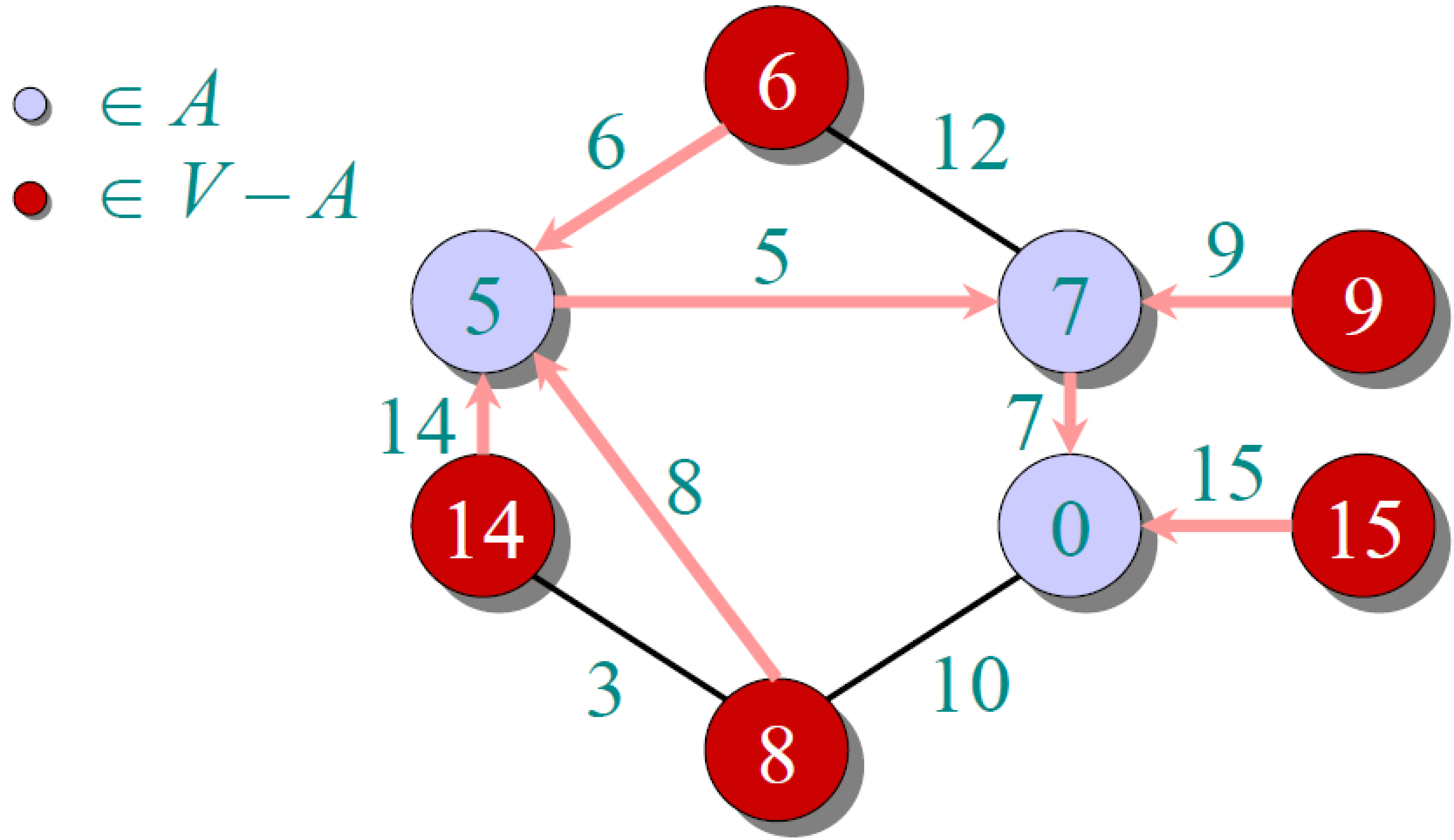
Example of Prim's algorithm



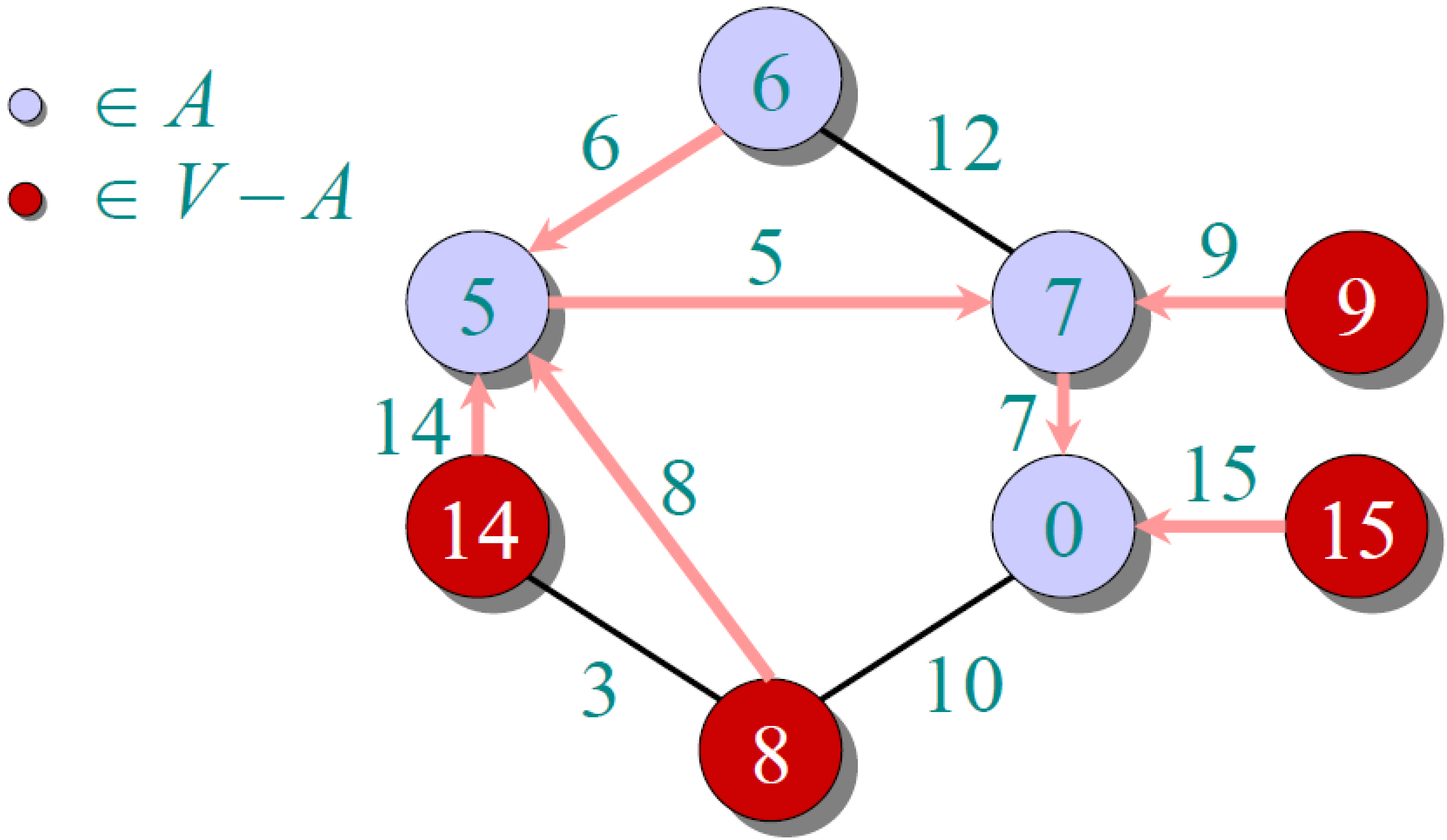
Example of Prim's algorithm



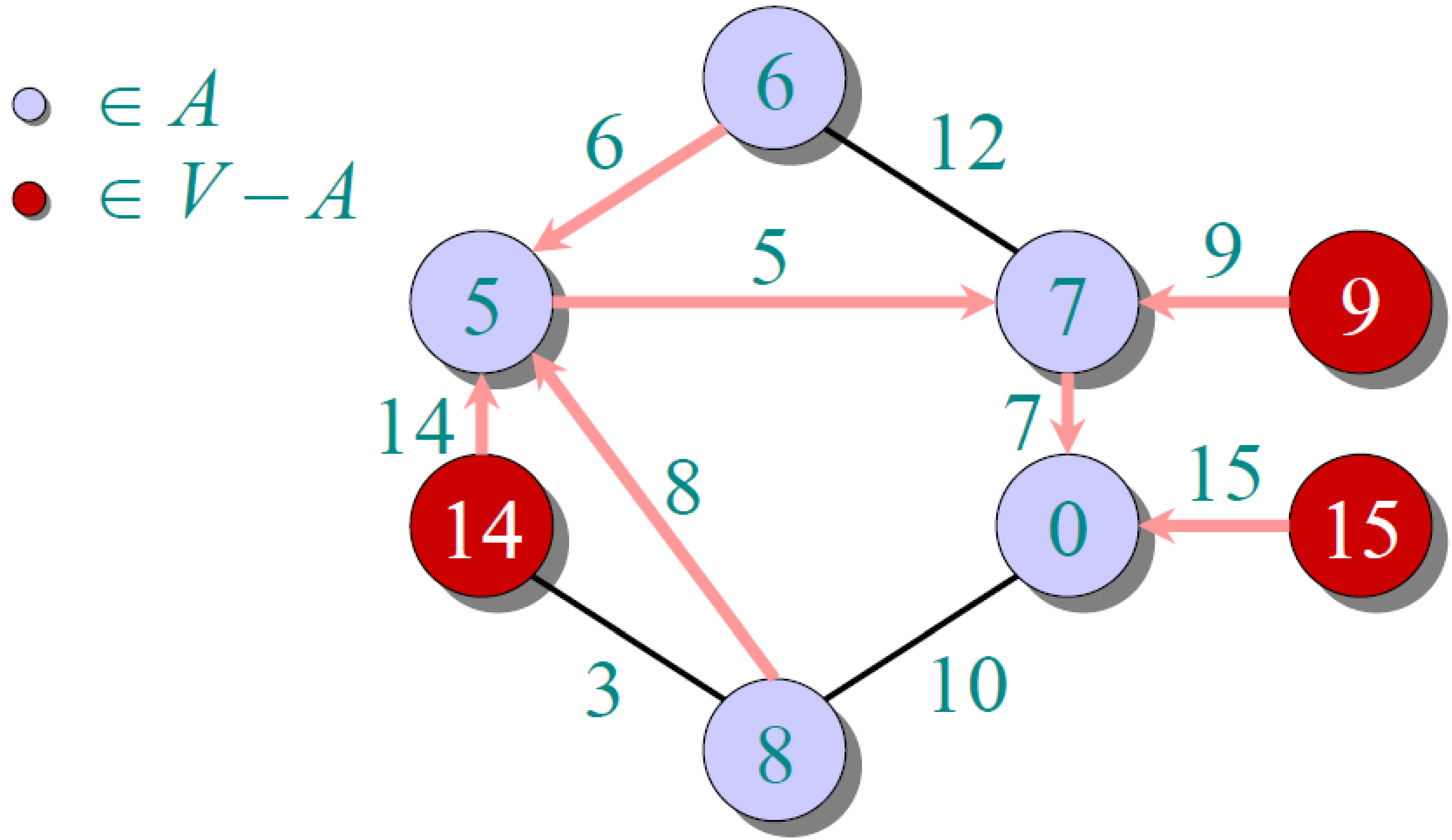
Example of Prim's algorithm



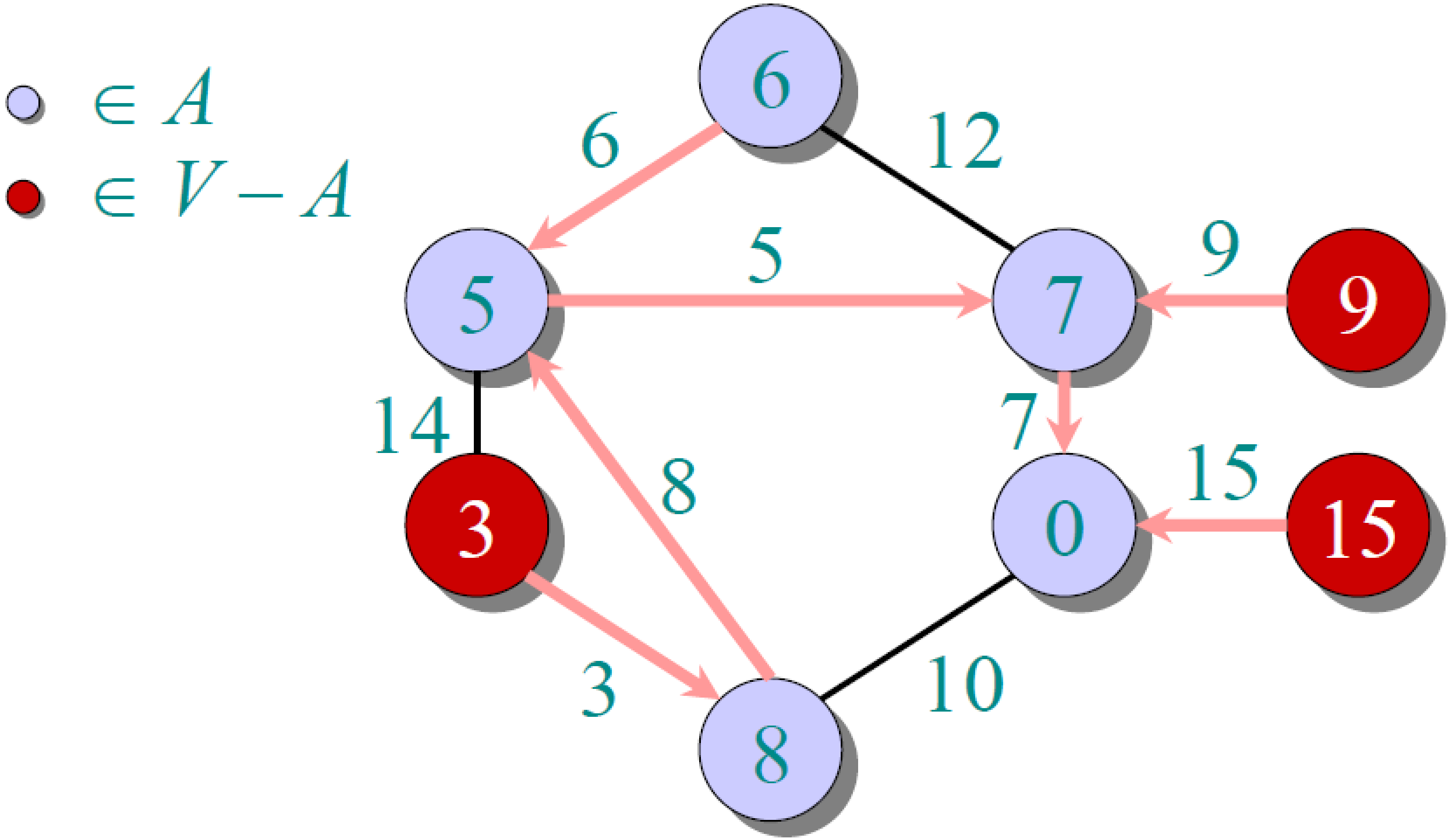
Example of Prim's algorithm



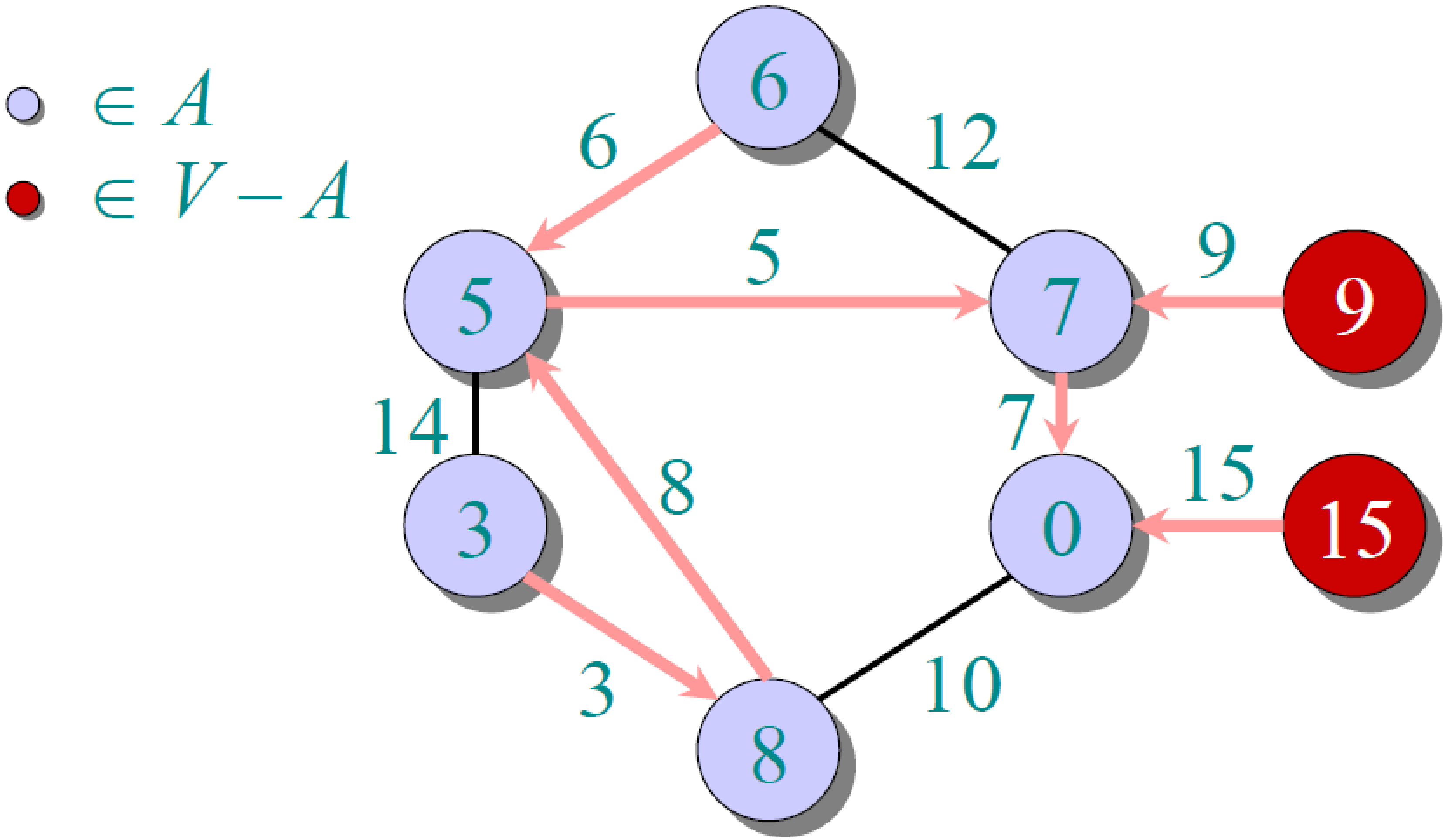
Example of Prim's algorithm



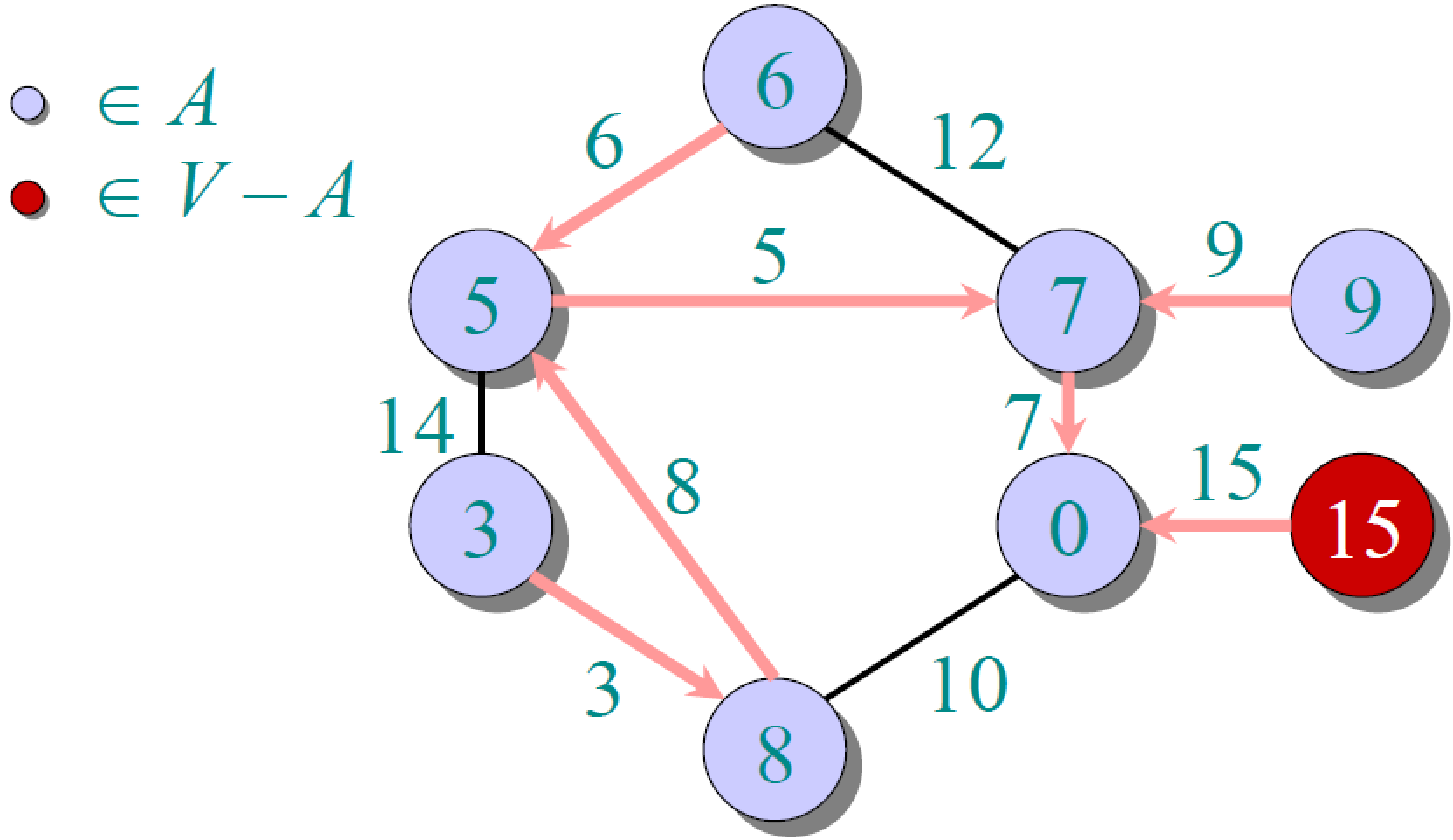
Example of Prim's algorithm



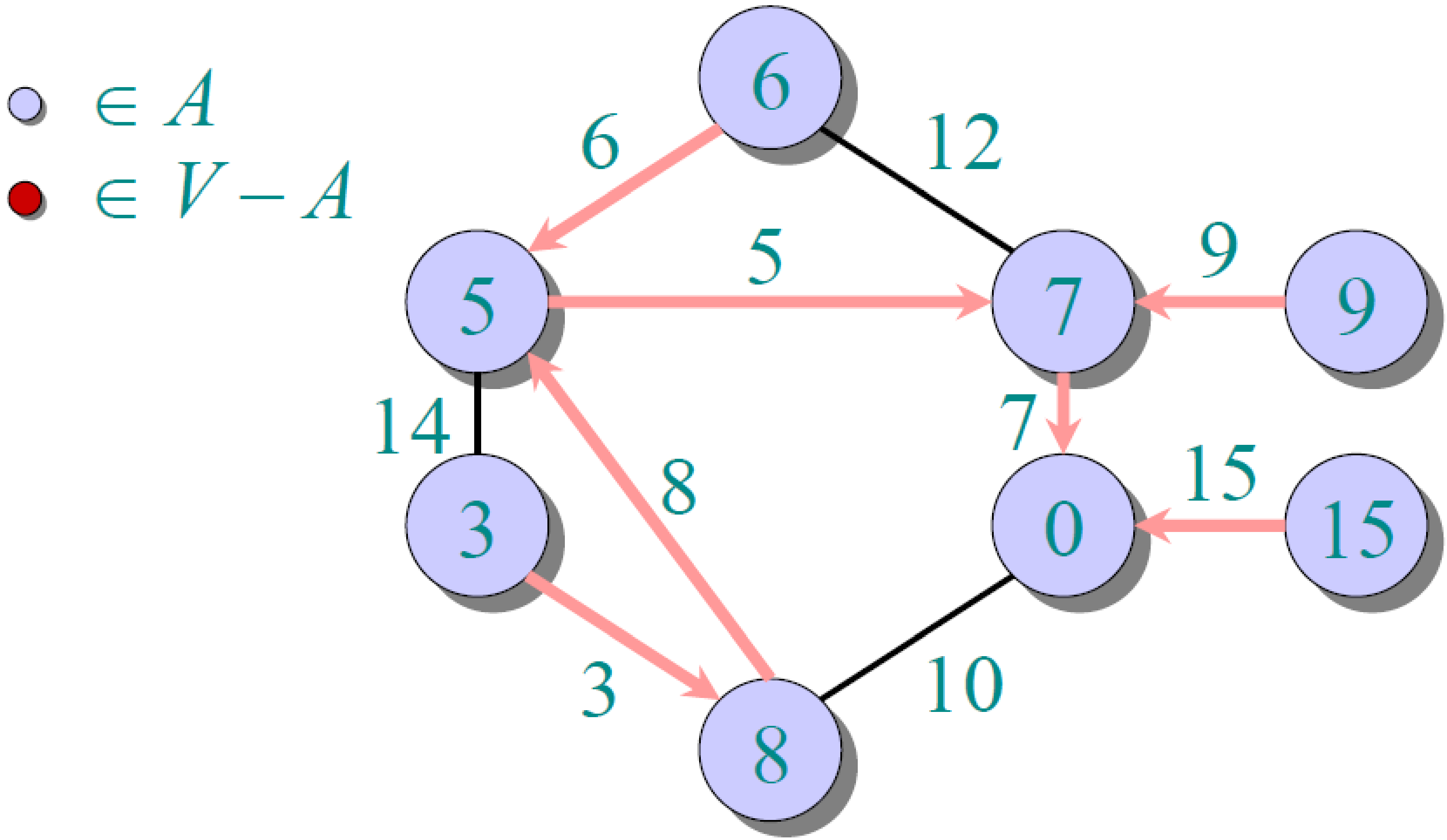
Example of Prim's algorithm



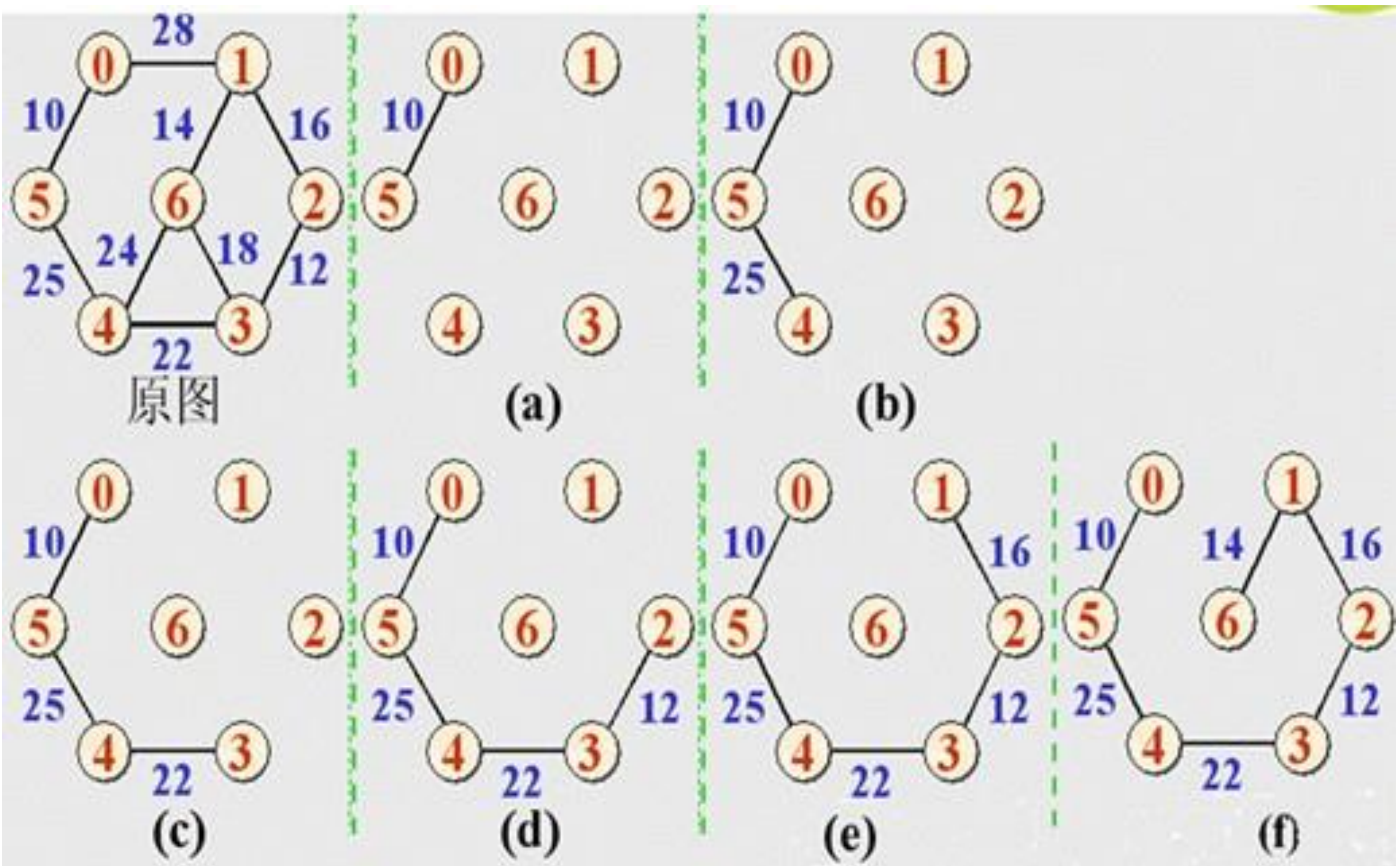
Example of Prim's algorithm



Example of Prim's algorithm



Another Example



Analysis of Prim

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

$$
\left.\begin{array}{r}|V|\\ \text{times}\end{array}\right\{
\begin{array}{l}
\textbf{while } Q \neq \varnothing \\
\quad \textbf{do } u \leftarrow \text{EXTRACT-MIN}(Q) \\
\quad \left.\begin{array}{r} degree(u) \\ \text{times}\end{array}\right\{
\begin{array}{l}
\textbf{for each } v \in Adj[u] \\
\quad \textbf{do if } v \in Q \textbf{ and } w(u,v) < key[v] \\
\quad\quad \textbf{then } \boxed{key[v] \leftarrow w(u,v)} \\
\quad\quad\quad\quad \pi[v] \leftarrow u
\end{array}
\end{array}
$$

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

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

Analysis of Prim

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

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

17.3 Kruskal Algorithm

Disjoint Set (Union-Find Set)

并查集(Disjoint-Set)是一种可以动态维护的数据结构，由不想交的集合构成，主要用于元素分组。基本运算有

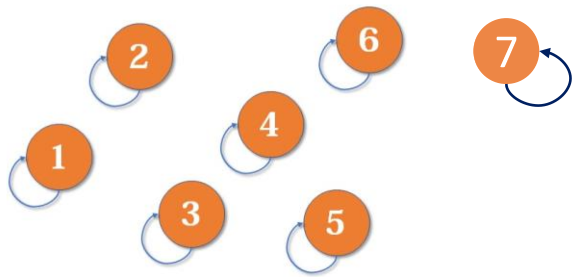
•**合并**（Union）：把两个不相交的集合合并为一个集合
•**查询**（Find）：查询两个元素是否在同一个集合中

基本数据结构：
- 每个集合由其中的一个元素代表
- 集合中的所有元素都在以代表元素为根的树上（逻辑结构）
- 数组parent[x]指向元素x在树中的父节点，如果x为根，则parent[x] = x
- 每个元素x都可以沿parent[x]在树中向上移动，直至根元素

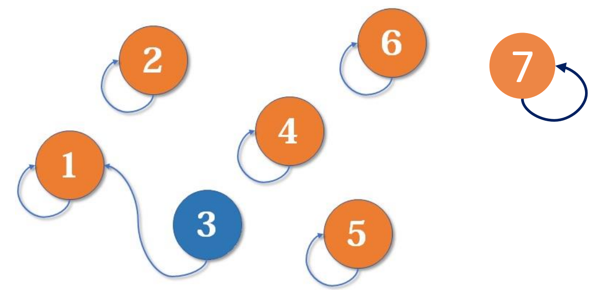
基本算法：
- 判断两个元素是否属于同一集合，只需要看它们的根节点是否相同
- 合并两个集合，只需要把根元素合并，即把其中一个根元素设置为另一个根元素的父节点

路径压缩：
- 把元素x到达根的路径上的所有元素的parent设为根节点

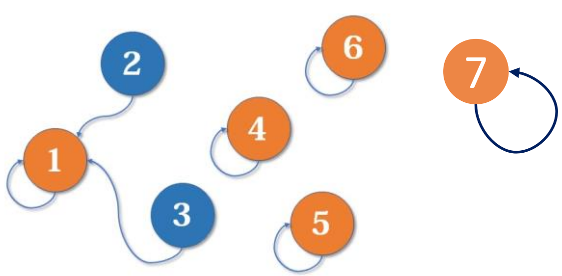
Disjoint Set (Union-Find Set)



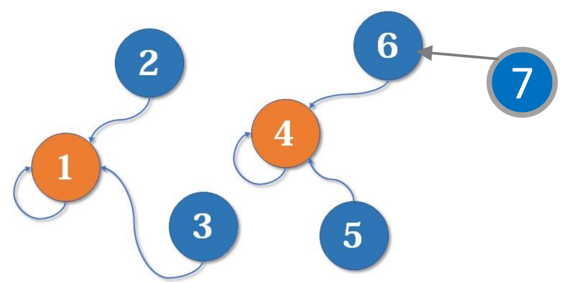
(1) initialization: 每个元素单独组成一个集合



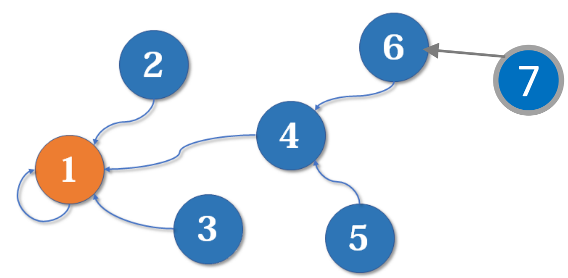
(2) union(1, 3)



(3) union(2, 3)

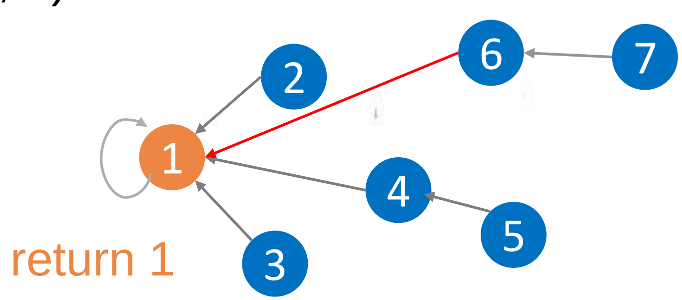


(4) union(4, 5), union(6, 7)和 union(5, 6)



(5) union(3, 6)

(6) find(6) 并压缩路径



Implementations

//**initialization**
for each $i \in \{1, \ldots n\}$
  do $parent[i] \leftarrow i$

find(a)

$root \leftarrow a$
while $parent[root] \neq root$
  do $root \leftarrow parent[root]$
 //路径压缩
while $parent[a] \neq root$
  do $temp \leftarrow parent[a]$
   $parent[a] \leftarrow root$
   $a \leftarrow temp$
return $root$

union(a, b)

$root\_a \leftarrow find(a)$
$root\_b \leftarrow find(b)$
if $root\_a \neq root\_b$
  then $parent[root\_b] \leftarrow root\_a$

Examples

1. 某市调查城镇交通状况，得到现有城镇道路统计表。表中列出了每条道路直接连通的城镇。市政府 "村村通工程" 的目标是使全市任何两个城镇间都可以实现交通（但不一定有直接的道路相连，只要相互之间可达即可）。请你计算出最少还需要建设多少条道路？

解题思路： 把已有道路连通的城镇进行合并，合并后的集合数量减去1就是需要建设的道路的最少数量

Examples

2. 把正整数区间$[a,b](1 \le a < b < 10^6)$分割成若干个不想交的集合，各集合中的任一整数满足以下条件：所有与该整数有不小于$p(1 < p \le b)$的公共质因数的整数都在同一个集合中。问最终能分成多少个集合？

解题思路：(1) 枚举 $\ge p$ 且 $\le b$的所有质数:

$$P[1] = p < P[2] < \cdots < P[m] \le b$$

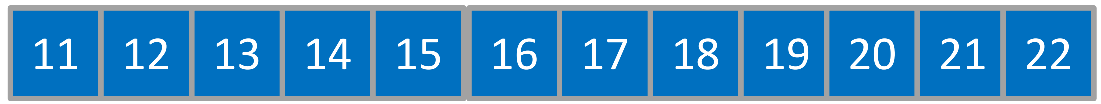
(2) for each $k \in \{1, \ldots, m\}$, 合并区间$[a,b]$中所有$P[k]$的倍数

比如分割 $[11, 22], p = 3$

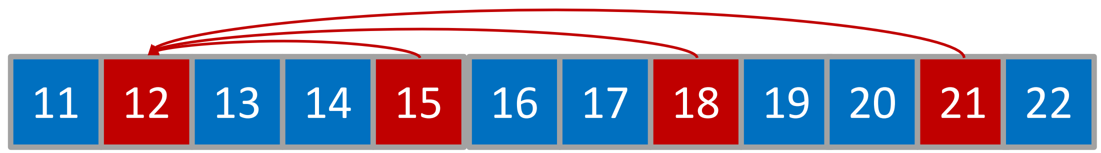
Examples

2. 把正整数区间$[a,b](1 \le a < b < 10^6)$分割成若干个集合，各集合中的任一整数满足以下条件：所有与该整数有不小于$p(1 < p \le b)$的公共质因数的整数都在同一个集合中。问最终能分成多少个集合？

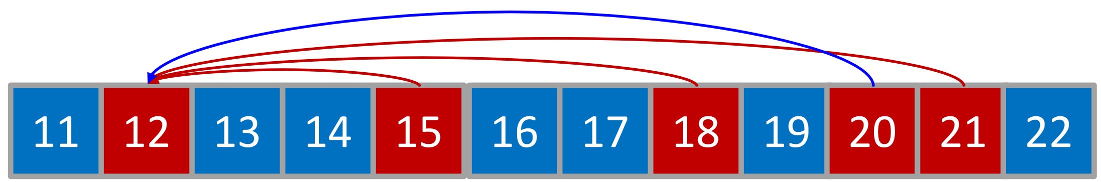
比如分割 $[11,22], p = 3$



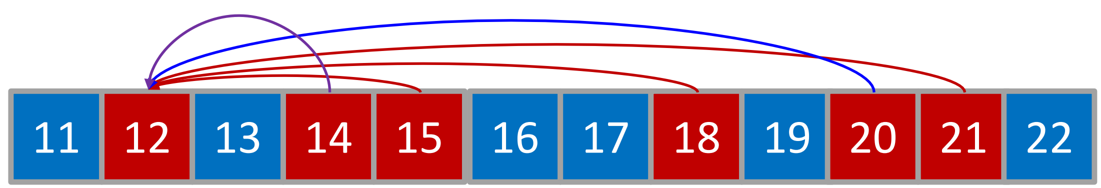
- 合并3的倍数



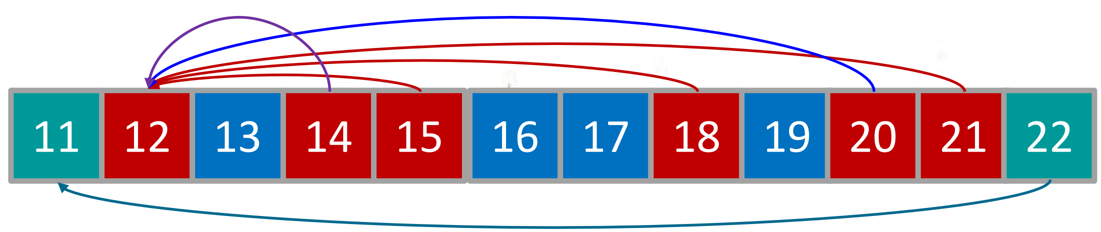
- 合并5的倍数



- 合并7的倍数



- 合并11的倍数



Examples

3. 无向图$G=(V,E)$, 从中依次删除k个节点$n_1, n_2, \ldots, n_k$, 求去掉每个节点后的连通成分数量。

解题思路： 并查集适合合并，不适合分离，所以倒序求解。先把k个节点及节点链接的边全部去掉，用并查集计算连通成分数量，再依次加入节点$n_k, n_{k-1}, \ldots, n_2$以及与图中其它节点链接的边, 求每次添加后的连通成分数量。

Kruskal's Algorithm

- **Yet another greedy algorithm**

- **Initialize all vertices to unconnected**
- **While there are still unmarked edges**
  - **Pick the lowest cost edge `e = (u, v)` and mark it**
  - **If `u` and `v` are not already connected, add `e` to the minimum spanning tree and connect `u` and `v`**

- **How is this like maze generation?**
- **How is it different?**

Kruskal's Algorithm

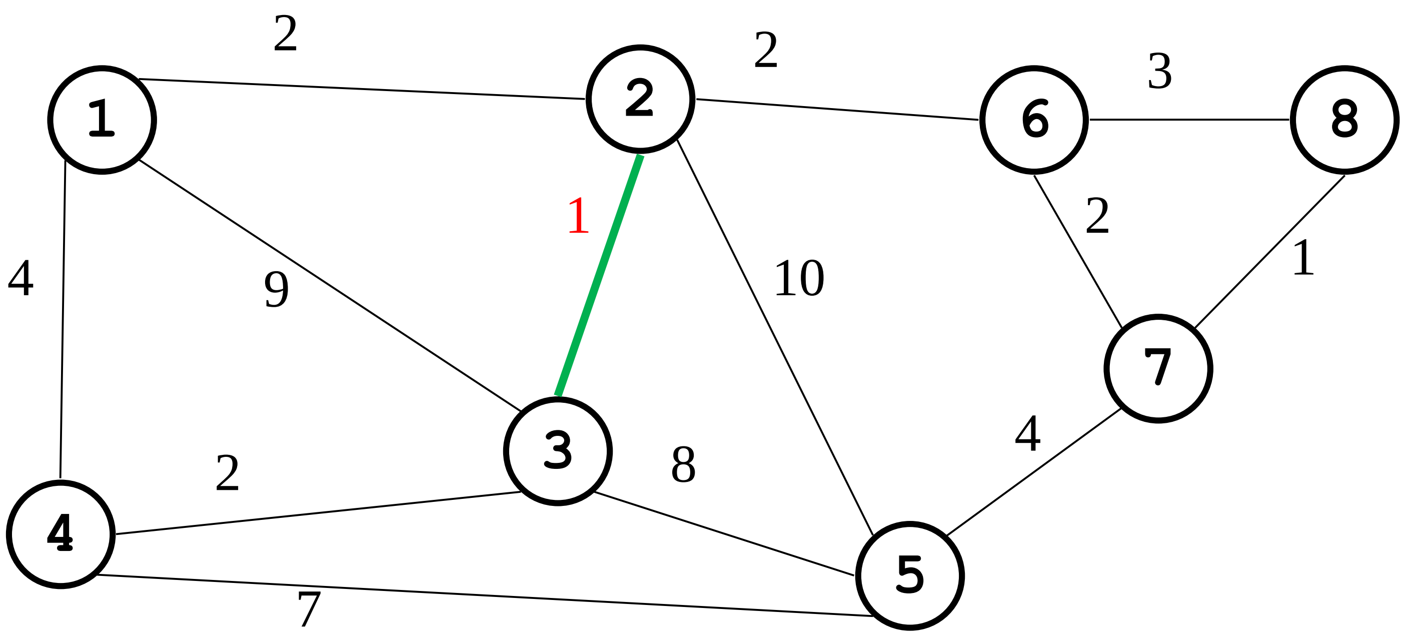
Algorithm:

```
T={ };
 while (T contains less than n-1 edges &&
              E is not empty ){
        Choose a least cost edge (v,w) from E;
        delete (v,w) from E;
        if ((v,w) does not create a cycle in T)
            add (v,w) into T;
        else discard (v,w);
 }
 if (T contains fewer than n-1 edges)
          print ("No spanning tree\n");
```

```
class KruskElem {            // An element for the heap
public:
  int from, to, distance; // The edge being stored
  KruskElem() { from = -1;  to = -1; distance = -1; }
  KruskElem(int f, int t, int d)
    { from = f; to = t; distance = d; }
};
void Kruskel(Graph* G) {   // Kruskal's MST algorithm
  ParPtrTree A(G->n());    // Equivalence class array
  KruskElem E[G->e()];     // Array of edges for min-heap
  int i;
  int edgecnt = 0;
  for (i=0; i<G->n(); i++) // Put the edges on the array
    for (int w=G->first(i); w<G->n(); w = G->next(i,w)) {
      E[edgecnt].distance = G->weight(i, w);
      E[edgecnt].from = i;
      E[edgecnt++].to = w;
    }
```

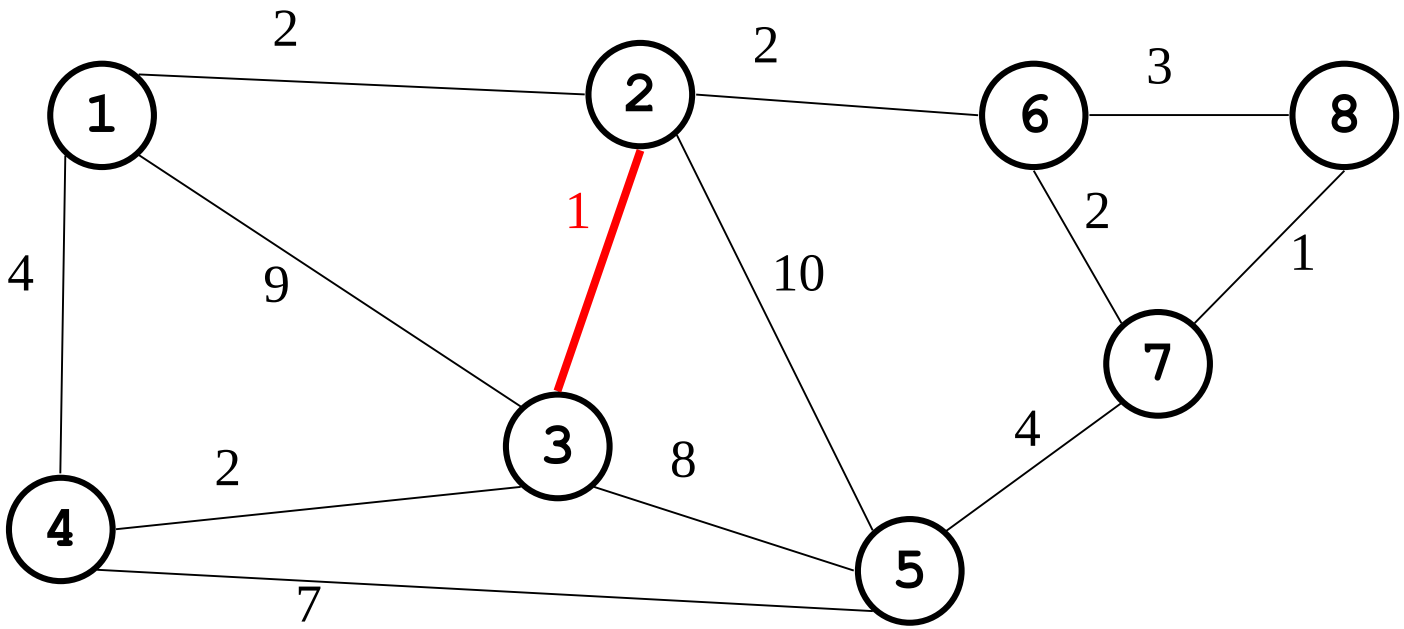
```
// Heapify the edges
heap<KruskElem, Comp> H(E, edgecnt, edgecnt);
int numMST = G->n();          // Initially n equiv classes
for (i=0; numMST>1; i++) { // Combine equiv classes
  KruskElem temp;
  temp = H.removefirst(); // Get next cheapest edge
  int v = temp.from;  int u = temp.to;
  if (A.differ(v, u)) {  // If in different equiv classes
    A.UNION(v, u);        // Combine equiv classes
    AddEdgetoMST(temp.from, temp.to);  // Add edge to MST
    numMST--;             // One less MST
  }
}
}
```

Kruskal’s Algorithm in Action



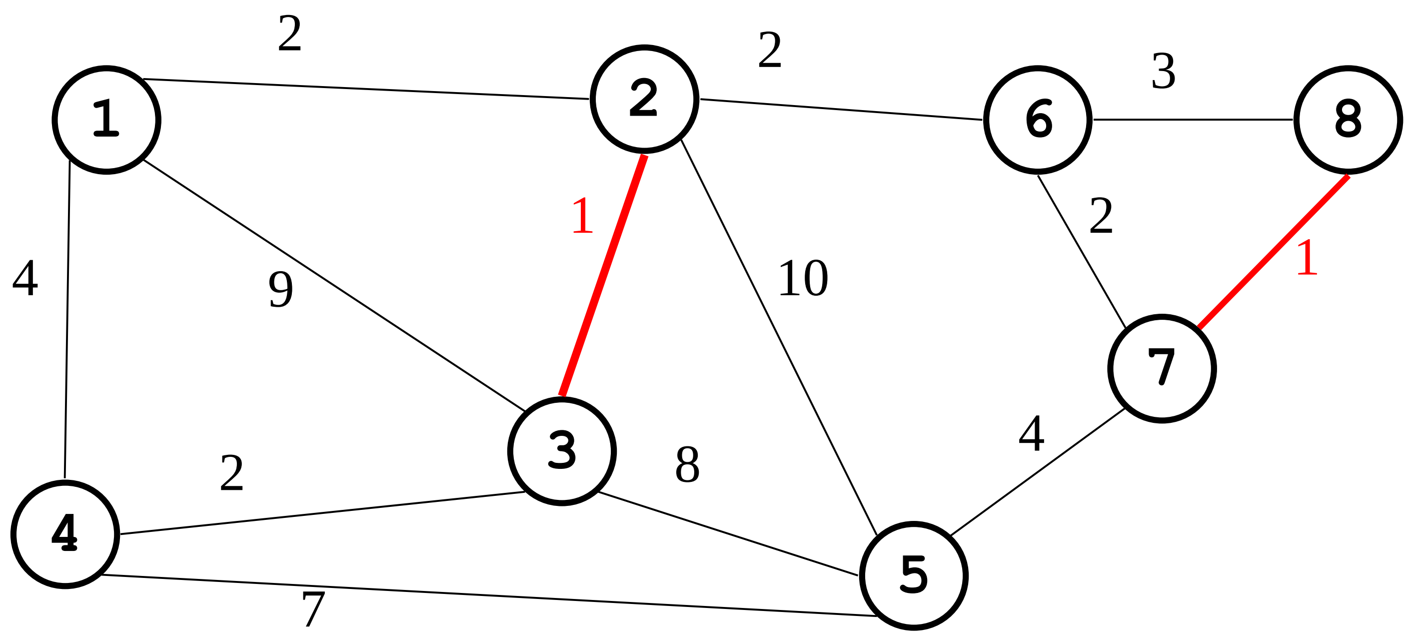
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8

Kruskal's Algorithm in Action



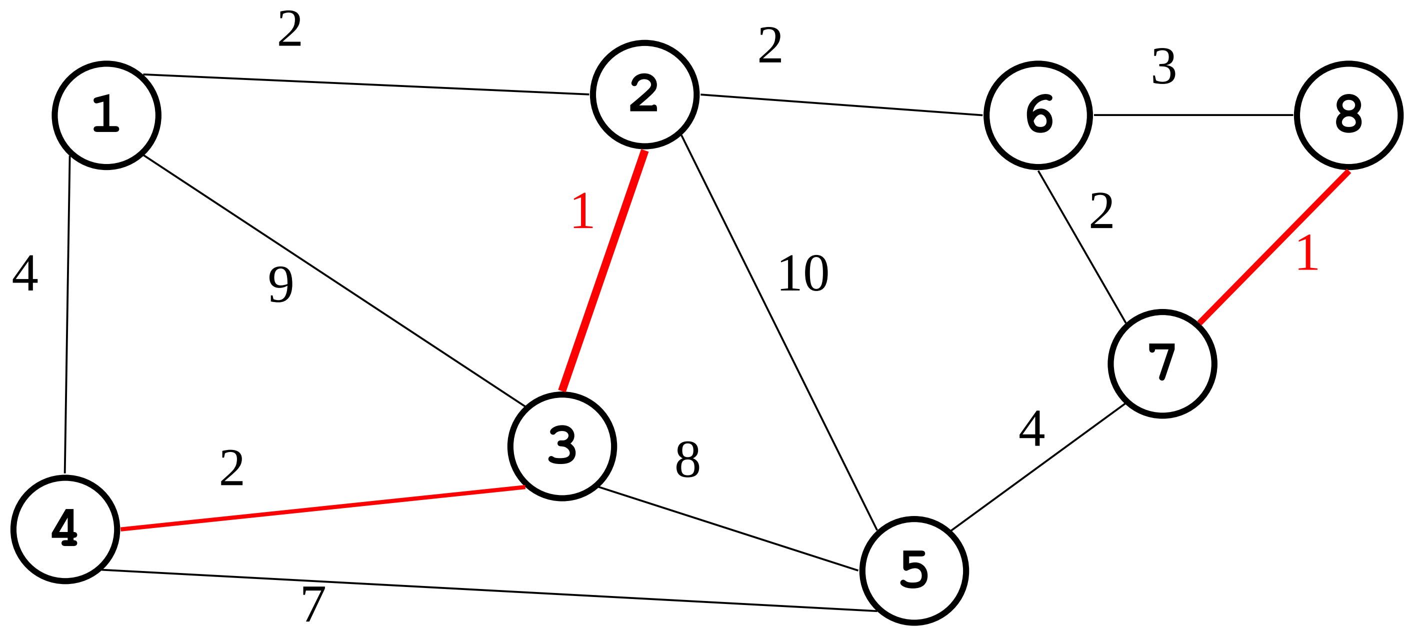
1	2	2	4	5	6	7	8
1	2	3	4	5	6	7	8

Kruskal’s Algorithm in Action



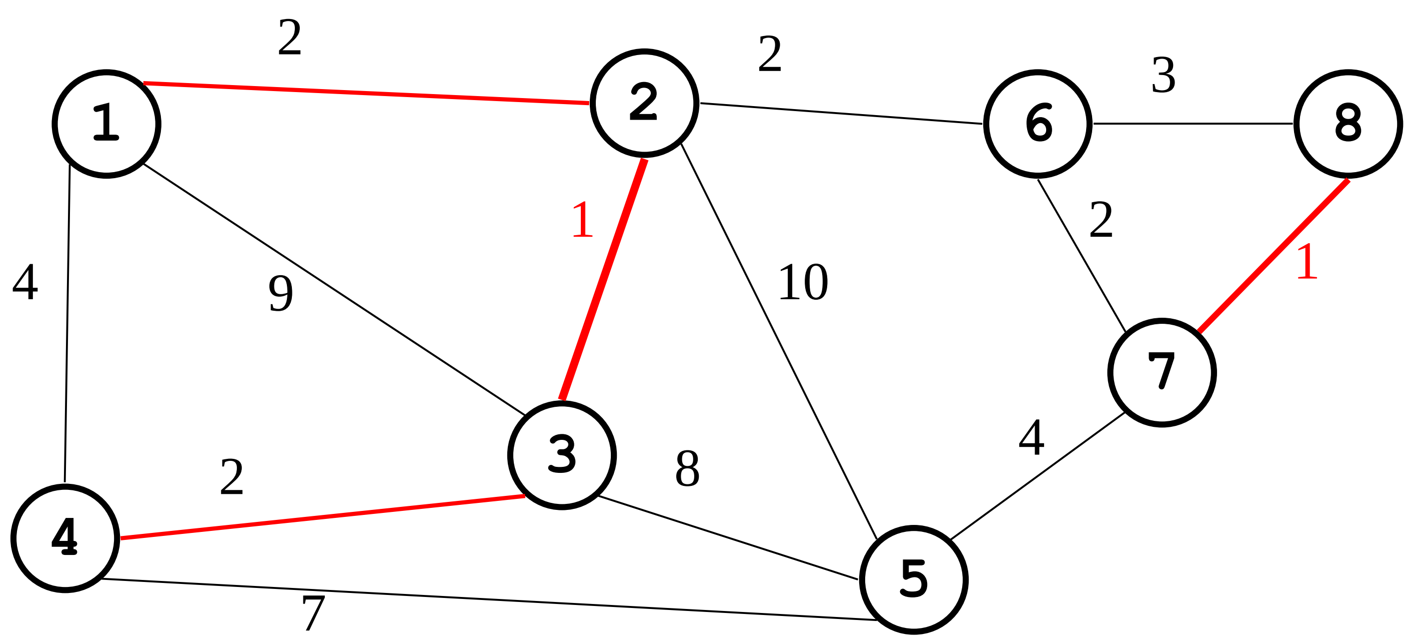
1	2	2	4	5	6	7	7
1	2	3	4	5	6	7	8

Kruskal's Algorithm in Action



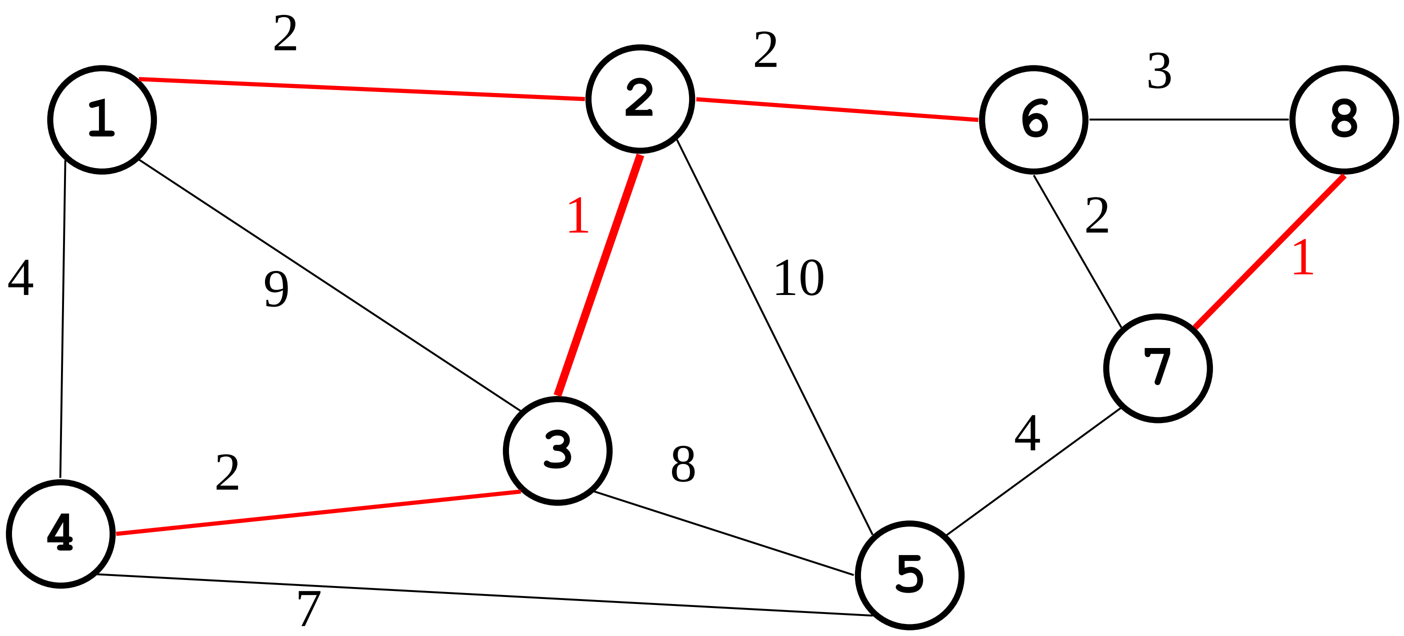
1	2	2	2	5	6	7	7
1	2	3	4	5	6	7	8

Kruskal's Algorithm in Action



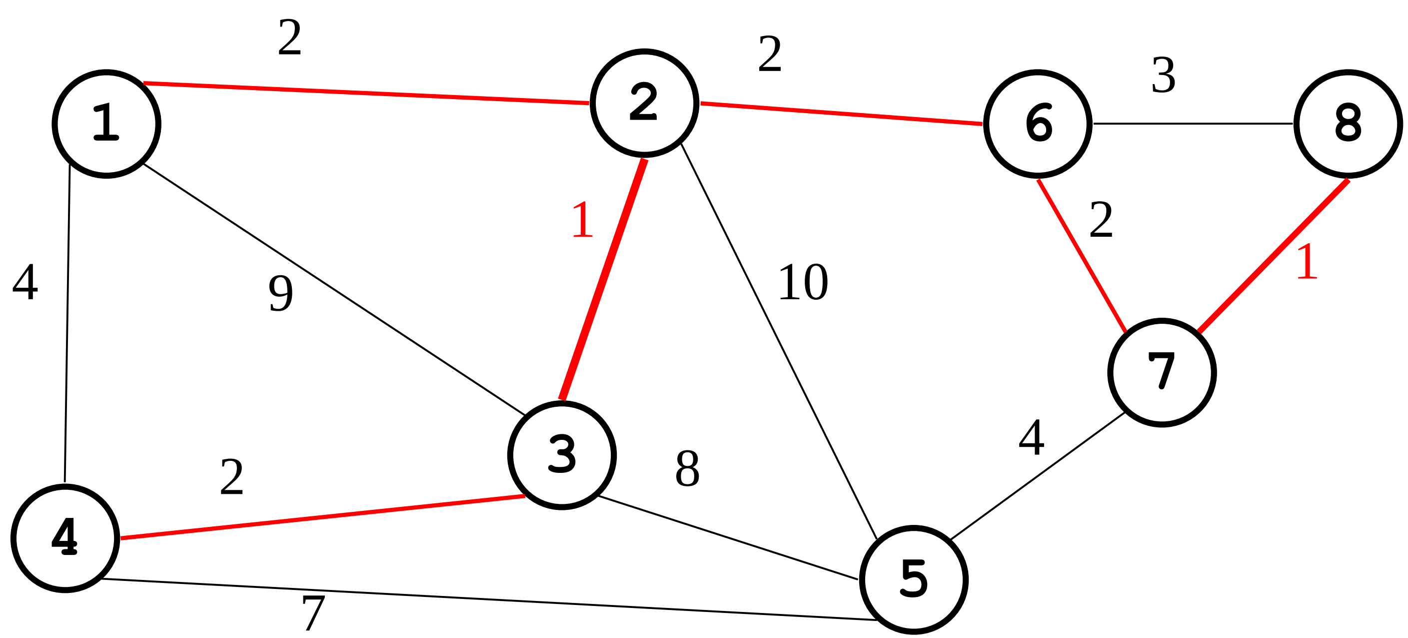
1	1	2	2	5	6	7	7
1	2	3	4	5	6	7	8

Kruskal's Algorithm in Action



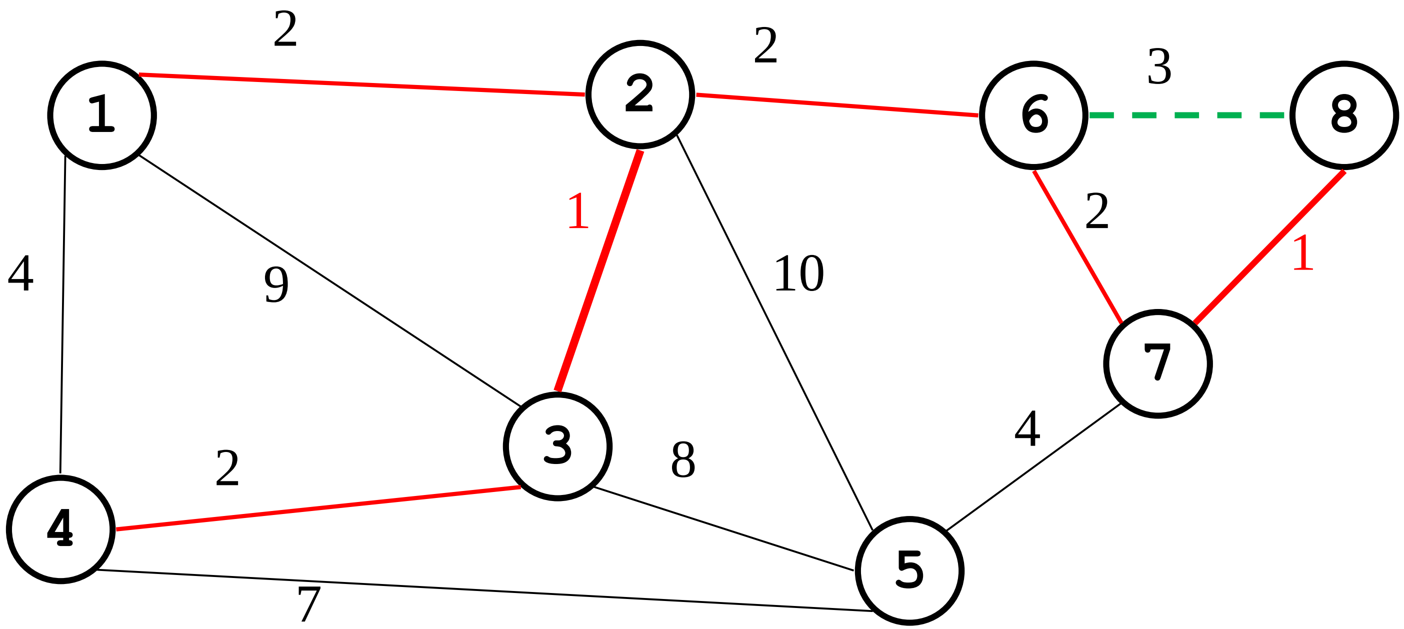
1	1	2	2	5	1	7	7
1	2	3	4	5	6	7	8

Kruskal’s Algorithm in Action



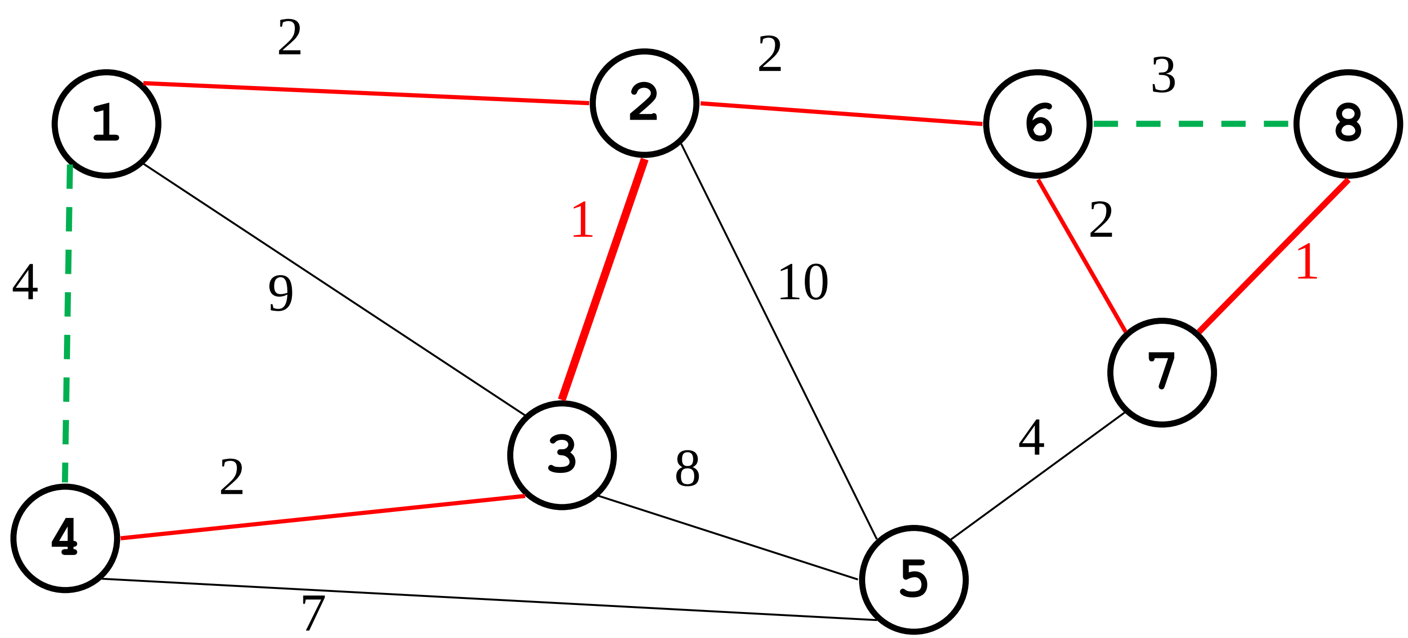
1	1	2	2	5	1	1	7
1	2	3	4	5	6	7	8

Kruskal's Algorithm in Action



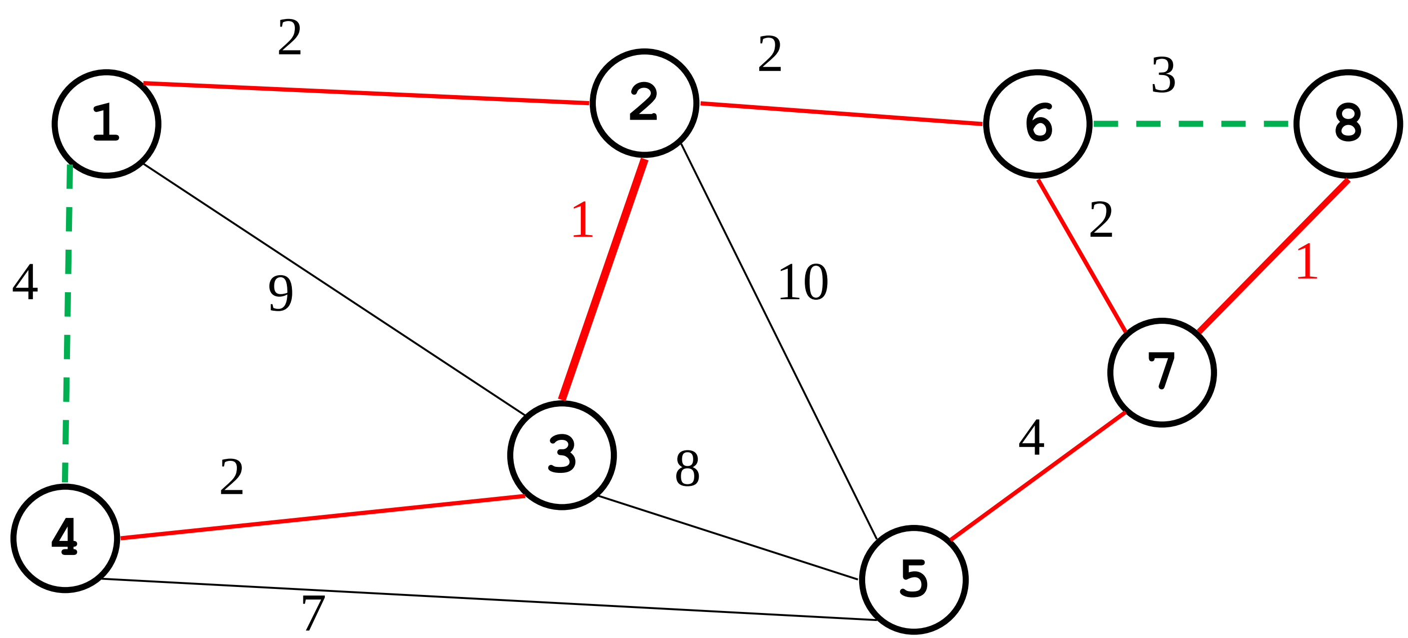
1	1	2	2	5	1	1	7
1	2	3	4	5	6	7	8

Kruskal’s Algorithm in Action



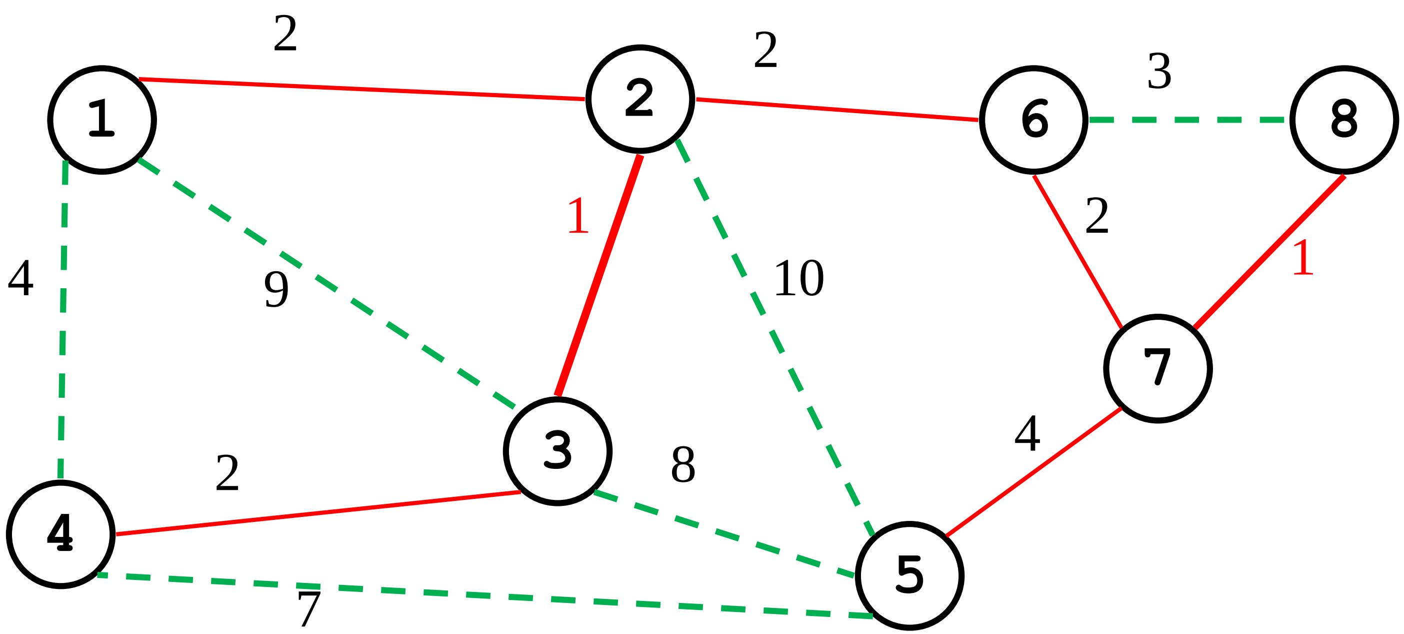
1	1	2	2	5	1	1	7
1	2	3	4	5	6	7	8

Kruskal’s Algorithm in Action



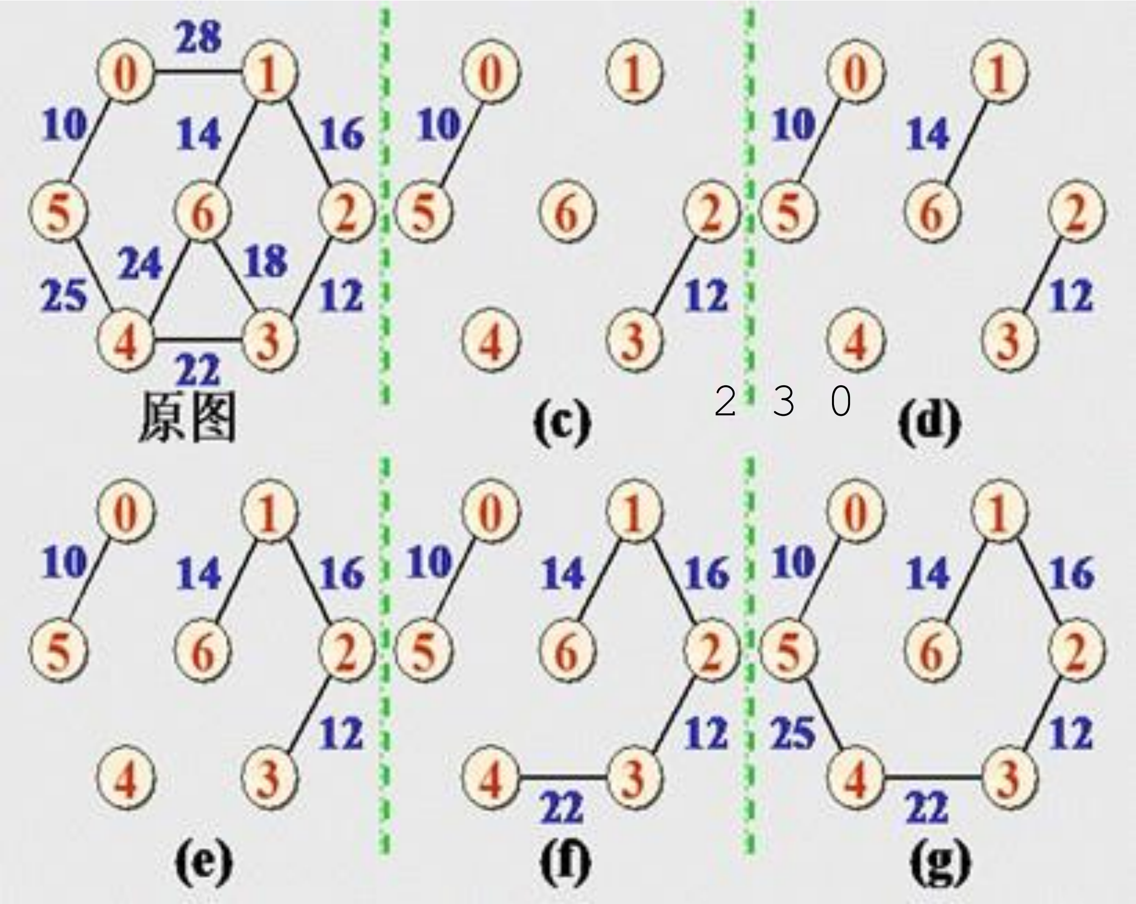
1	1	2	2	1	1	1	7
1	2	3	4	5	6	7	8

Kruskal’s Algorithm in Action



1	1	2	2	1	1	1	7
1	2	3	4	5	6	7	8

Another Example



Fundamental features of MST

- 设$G=(V,E)$为连通无向图。G中的任意一条回路上权重值最大的边，一定不在最小生成树上。

- 设$G=(V,E)$为连通无向图， 如果G中所有边的权重互异，则其MST唯一。

MST algorithms

Prim's algorithm

- Uses the ***priority queen***.
- Running time $= O(E \lg V)$.

Kruskal's algorithm

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

Best to date:

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

数据结构与算法课程组
重庆大学计算机学院

End of Section.