

# Minimum Spanning Tree (5A)

---

Copyright (c) 2015 - 2018 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to [youngwlim@hotmail.com](mailto:youngwlim@hotmail.com).

This document was produced by using LibreOffice and Octave.

# Minimum Spanning Tree

---

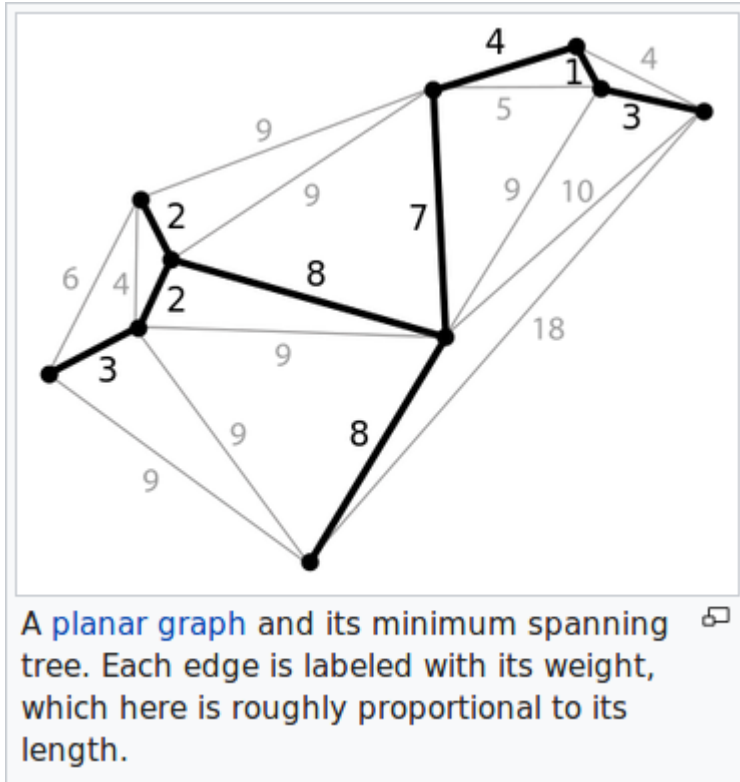
a **subset** of the **edges** of a connected, edge-weighted (un)directed graph that connects **all** the **vertices** together, without any **cycles** and with the **minimum** possible total edge **weight**.

a spanning tree whose sum of edge weights is as small as possible.

More generally, any edge-weighted undirected graph (not necessarily connected) has a minimum spanning **forest**, which is a **union** of the minimum spanning **trees** for its connected components.

[https://en.wikipedia.org/wiki/Minimum\\_spanning\\_tree](https://en.wikipedia.org/wiki/Minimum_spanning_tree)

# Types of Shortest Path Problems



[https://en.wikipedia.org/wiki/Minimum\\_spanning\\_tree](https://en.wikipedia.org/wiki/Minimum_spanning_tree)

# Properties (1)

## Possible multiplicity

If there are **n vertices** in the graph, then each spanning tree has **n-1 edges**.

## Uniqueness

If each edge has a distinct weight then there will be only one, unique minimum spanning tree. this is true in many realistic situations

## Minimum-cost subgraph

If the weights are positive, then a minimum spanning tree is in fact a minimum-cost subgraph connecting **all vertices**, since subgraphs containing cycles necessarily have more total weight.

[https://en.wikipedia.org/wiki/Minimum\\_spanning\\_tree](https://en.wikipedia.org/wiki/Minimum_spanning_tree)

# Properties (2)

## Cycle Property

For any **cycle C** in the graph, if the weight of an **edge e** of **C** is larger than the individual weights of all other edges of **C**, then this edge cannot belong to a MST.

## Cut property

For any **cut C** of the graph, if the weight of an **edge e** in the **cut-set** of **C** is strictly smaller than the weights of all other edges of the **cut-set** of **C**, then this edge belongs to all MSTs of the graph.

[https://en.wikipedia.org/wiki/Minimum\\_spanning\\_tree](https://en.wikipedia.org/wiki/Minimum_spanning_tree)

# Properties (3)

## Minimum-cost edge

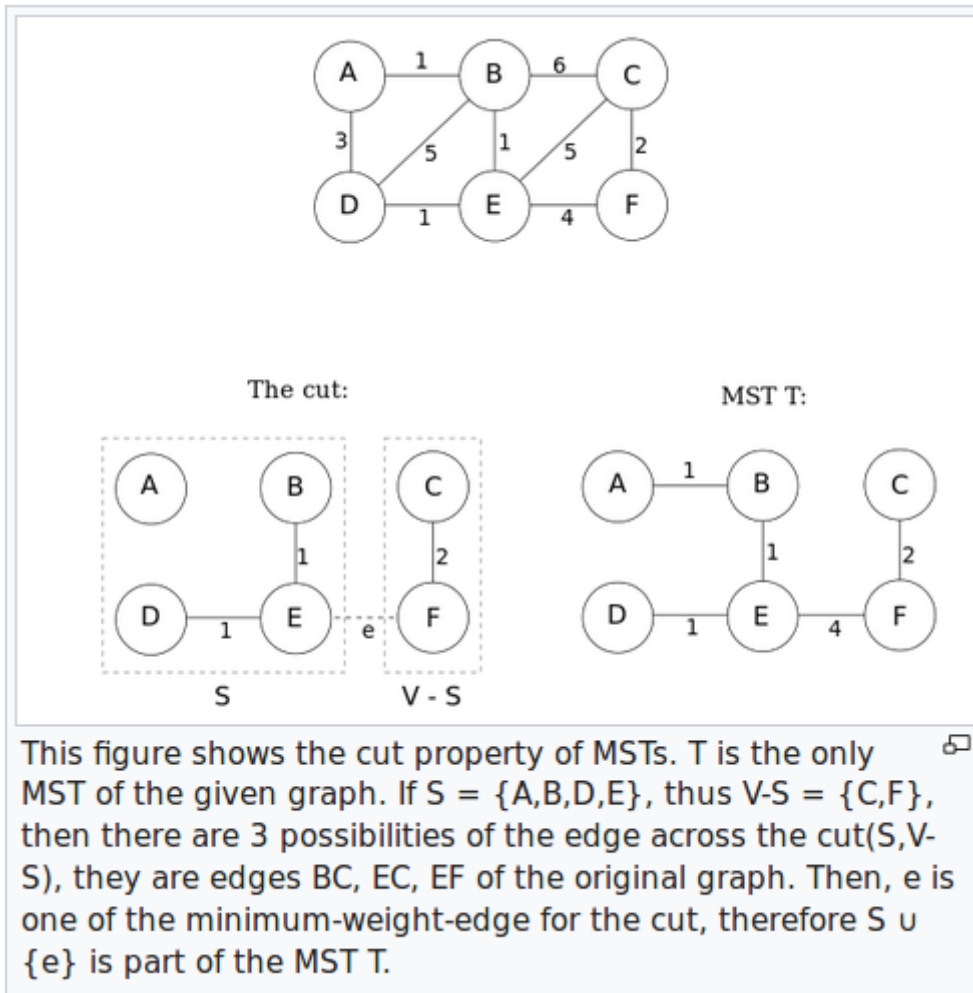
If the minimum cost **edge**  $e$  of a graph is unique, then this edge is included in any MST.

## Contraction

If  $T$  is a **tree** of **MST edges**, then we can contract  $T$  into a single vertex while maintaining the invariant that the MST of the contracted graph plus  $T$  gives the MST for the graph before contraction.

[https://en.wikipedia.org/wiki/Minimum\\_spanning\\_tree](https://en.wikipedia.org/wiki/Minimum_spanning_tree)

# Cut property examples



[https://en.wikipedia.org/wiki/Minimum\\_spanning\\_tree](https://en.wikipedia.org/wiki/Minimum_spanning_tree)



# Borůvka's algorithm

**Input:** A graph  $G$  whose edges have distinct weights  
Initialize a forest  $F$  to be a set of one-vertex trees,  
one for each vertex of the graph.

**While**  $F$  has more than one component:

Find the connected components of  $F$  and  
label each vertex of  $G$  by its component

Initialize the cheapest edge for each component to "None"

**For each** edge  $uv$  of  $G$ :

**If**  $u$  and  $v$  have different component labels:

**If**  $uv$  is cheaper than the cheapest edge  
for the component of  $u$ :

Set  $uv$  as the cheapest edge for the component of  $u$

**If**  $uv$  is cheaper than the cheapest edge  
for the component of  $v$ :

Set  $uv$  as the cheapest edge for the component of  $v$

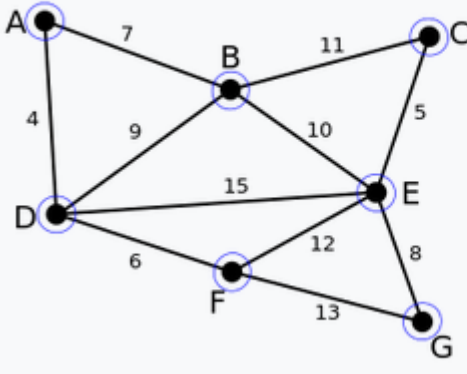
**For each** component whose cheapest edge  
is not "None":

Add its cheapest edge to  $F$

**Output:**  $F$  is the minimum spanning forest of  $G$ .

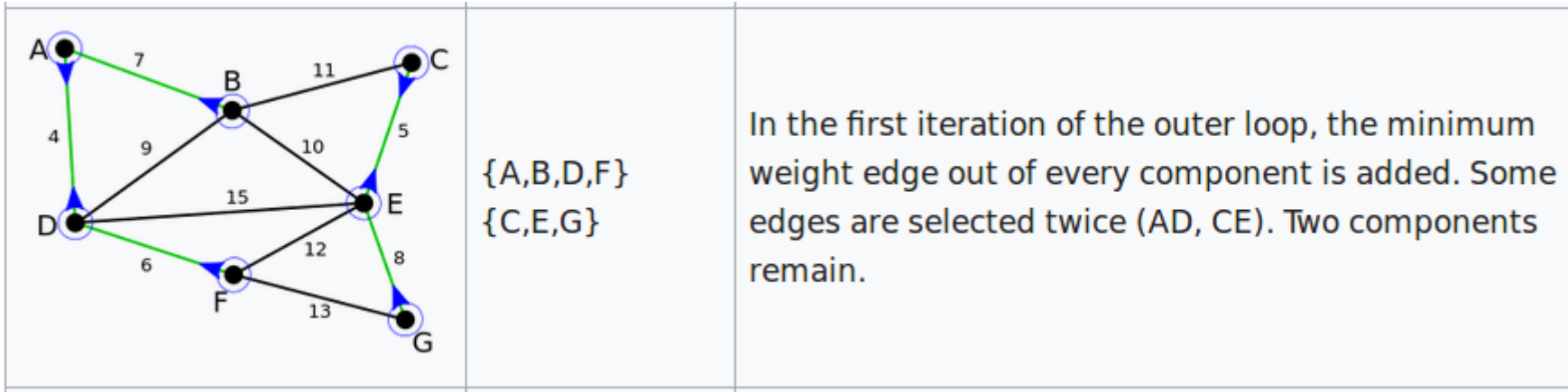
[https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s\\_algorithm](https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm)

# Borůvka's algorithm examples (1)

Image	components	Description
	<ul style="list-style-type: none"><li>{A}</li><li>{B}</li><li>{C}</li><li>{D}</li><li>{E}</li><li>{F}</li><li>{G}</li></ul>	<p>This is our original weighted graph. The numbers near the edges indicate their weight. Initially, every vertex by itself is a component (blue circles).</p>

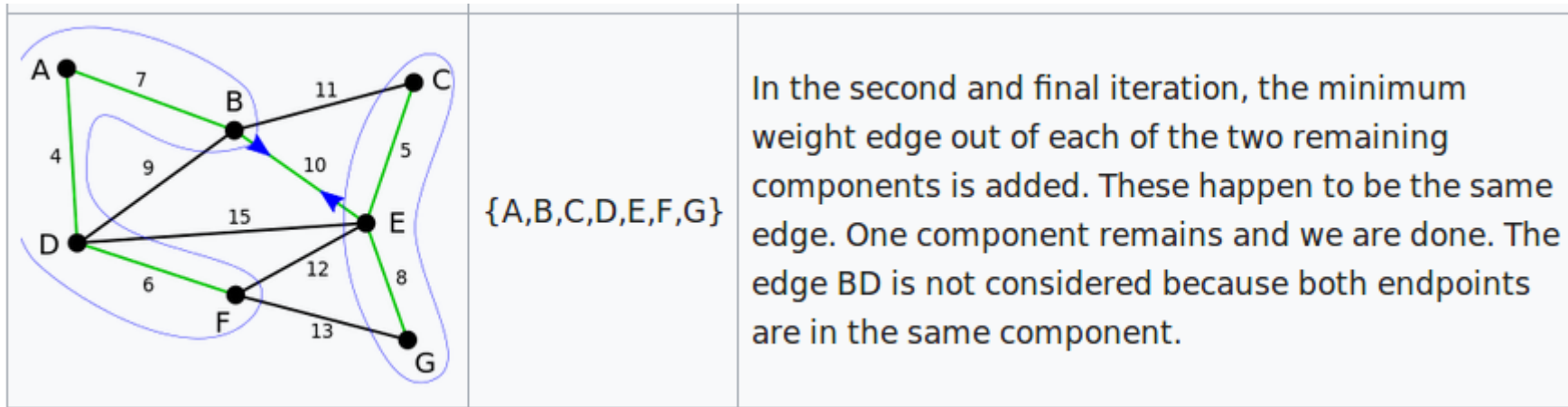
[https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s\\_algorithm](https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm)

## Borůvka's algorithm examples (2)



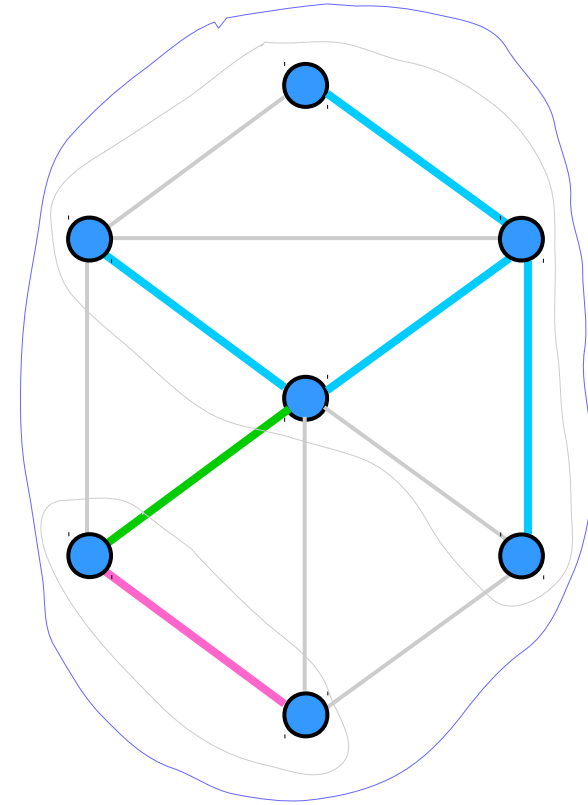
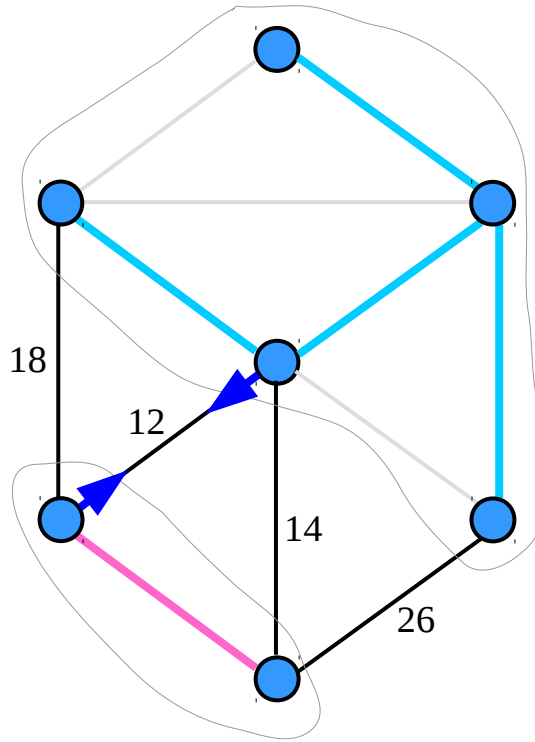
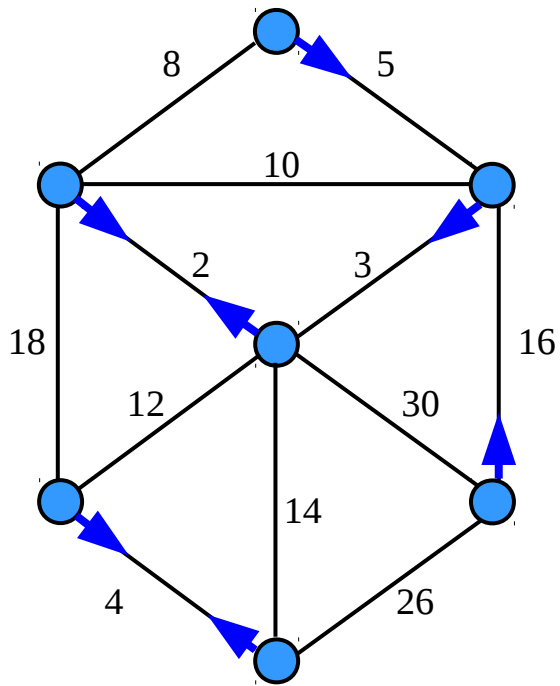
[https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s\\_algorithm](https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm)

# Borůvka's algorithm examples (3)



[https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s\\_algorithm](https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm)

# Borůvka's algorithm examples (4)



<http://jeffe.cs.illinois.edu/teaching/algorithms/notes/20-mst.pdf>

# Kruskal's algorithm

KRUSKAL(G):

1  $A = \emptyset$

2 **foreach**  $v \in G.V$ :

3   MAKE-SET( $v$ )

4 **foreach**  $(u, v)$  in  $G.E$  ordered by  $\text{weight}(u, v)$ , increasing:

5   if  $\text{FIND-SET}(u) \neq \text{FIND-SET}(v)$ :

6      $A = A \cup \{(u, v)\}$

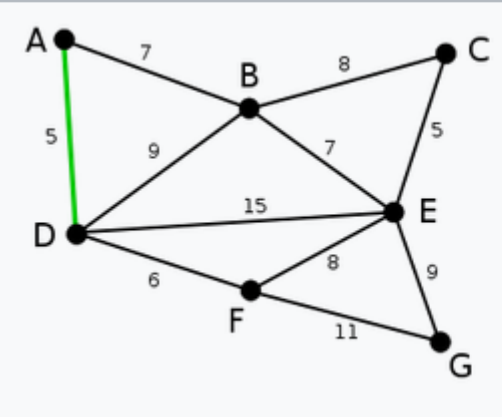
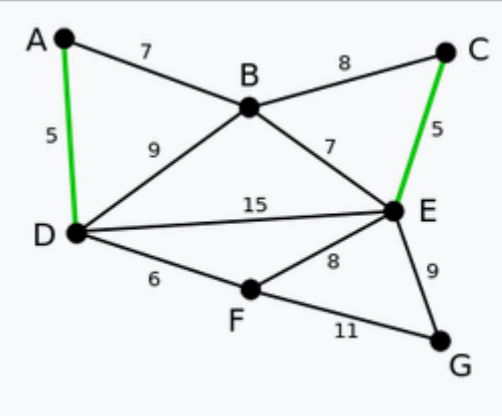
7     UNION( $u, v$ )

8 **return**  $A$

Scan all edges in increasing weight order; if an edge is safe, add it to  $A$

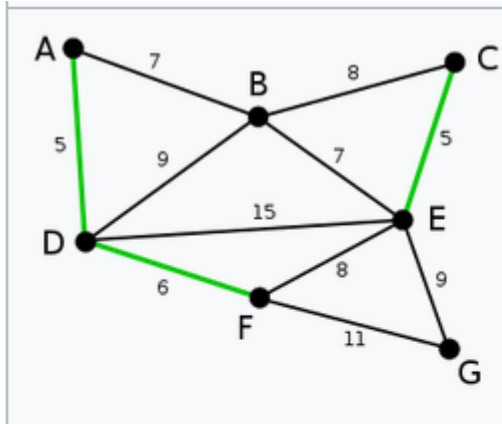
[https://en.wikipedia.org/wiki/Kruskal%27s\\_algorithm](https://en.wikipedia.org/wiki/Kruskal%27s_algorithm)

# Kruskal's algorithm examples (1)

	<p>{ 5, 5, 6, 7, 7, 8, 8, 9, 9, 11, 15 }</p> <p><b>AD</b> and <b>CE</b> are the shortest edges, with length 5, and <b>AD</b> has been arbitrarily chosen, so it is highlighted.</p>
	<p>{ 5, 5, 6, 7, 7, 8, 8, 9, 9, 11, 15 }</p> <p><b>CE</b> is now the shortest edge that does not form a cycle, with length 5, so it is highlighted as the second edge.</p>

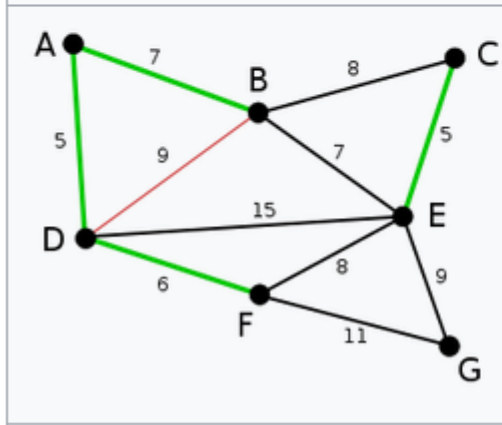
[https://en.wikipedia.org/wiki/Kruskal%27s\\_algorithm](https://en.wikipedia.org/wiki/Kruskal%27s_algorithm)

# Kruskal's algorithm examples (2)



{ 5, 5, 6, 7, 7, 8, 8, 9, 9, 11, 15 }

The next edge, **DF** with length 6, is highlighted using much the same method.



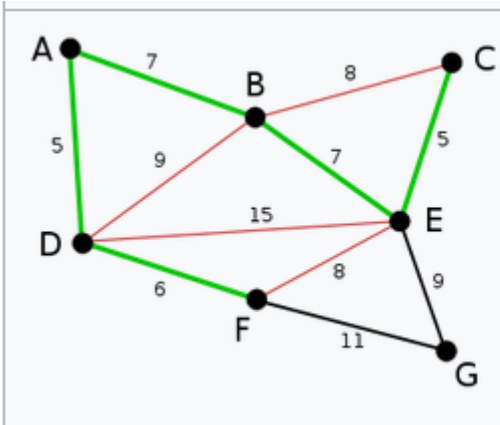
{ 5, 5, 6, 7, 7, 8, 8, ~~9~~, 9, 11, 15 }

The next-shortest edges are **AB** and **BE**, both with length 7. **AB** is chosen arbitrarily, and is highlighted. The edge **BD** has been highlighted in red, because there already exists a path (in green) between **B** and **D**, so it would form a cycle (**ABD**) if it were chosen.

[https://en.wikipedia.org/wiki/Kruskal%27s\\_algorithm](https://en.wikipedia.org/wiki/Kruskal%27s_algorithm)

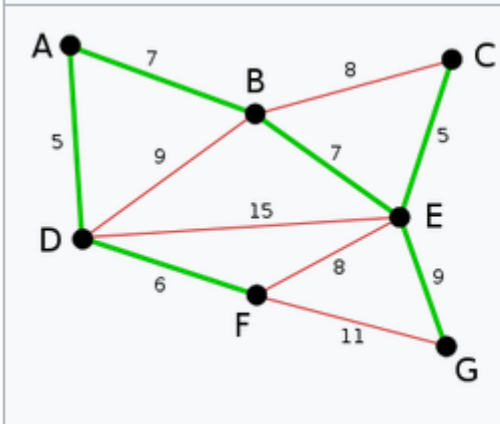


# Kruskal's algorithm examples (3)



{ 5, 5, 6, 7, 7, ~~8~~, ~~8~~, ~~9~~, 9, 11, 15 }

The process continues to highlight the next-smallest edge, **BE** with length 7. Many more edges are highlighted in red at this stage: **BC** because it would form the loop **BCE**, **DE** because it would form the loop **DEBA**, and **FE** because it would form **FEBAD**.



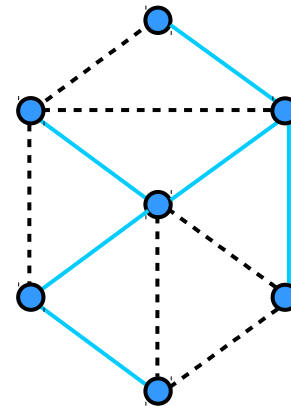
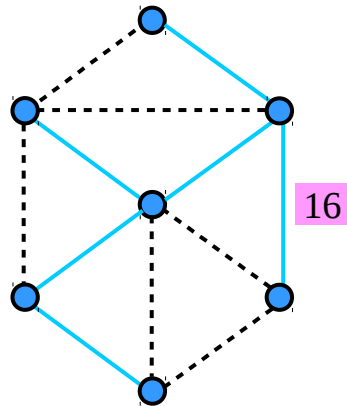
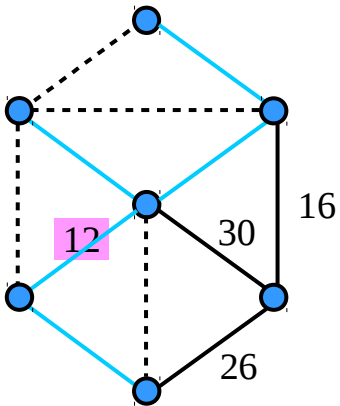
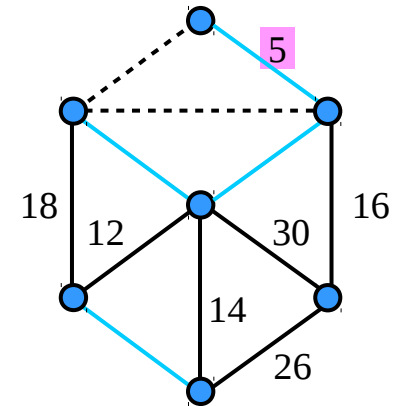
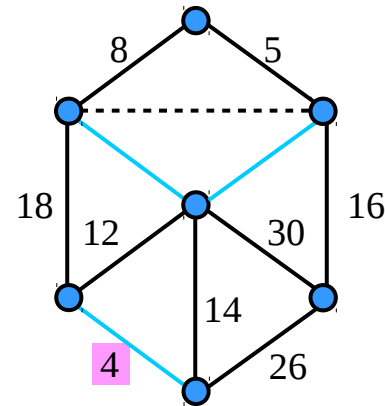
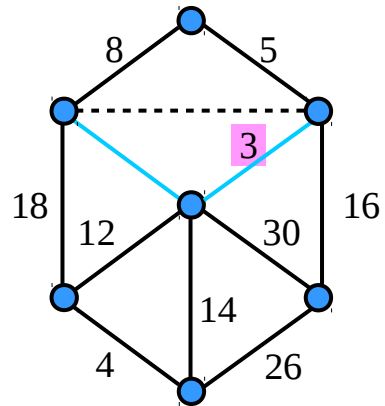
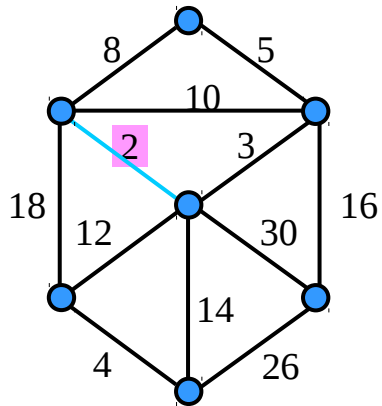
{ 5, 5, 6, 7, 7, ~~8~~, ~~8~~, ~~9~~, 9, ~~11~~, ~~15~~ }

Finally, the process finishes with the edge **EG** of length 9, and the minimum spanning tree is found.

[https://en.wikipedia.org/wiki/Kruskal%27s\\_algorithm](https://en.wikipedia.org/wiki/Kruskal%27s_algorithm)

# Kruskal's algorithm examples (4)

{2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30}



<http://jeffe.cs.illinois.edu/teaching/algorithms/notes/20-mst.pdf>

# Prim's algorithm

a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph.

operates by building this tree one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.

Repeatedly add a safe edge to the tree

1. Initialize a tree with a single vertex, chosen arbitrarily from the graph.
2. Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree.
3. Repeat step 2 (until all vertices are in the tree).

[https://en.wikipedia.org/wiki/Prim%27s\\_algorithm](https://en.wikipedia.org/wiki/Prim%27s_algorithm)

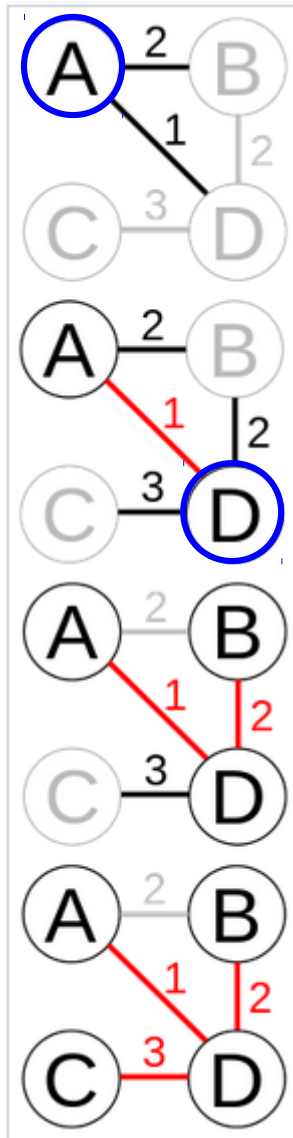
# Prim's algorithm

1. Associate with each vertex  $v$  of the graph a number  $C[v]$  (the cheapest cost of a connection to  $v$ ) and an edge  $E[v]$  (the cheapest edge).  
Initial values:  $C[v] = +\infty$ ,  $E[v] = \text{flag for no connection}$
2. Initialize an empty **forest**  $F$  and a **set**  $Q$  of **vertices** that have not yet been included in  $F$
3. Repeat the following steps until  $Q$  is empty:
  - a. Find and remove a vertex  $v$  from  $Q$  having the minimum possible value of  $C[v]$
  - b. Add  $v$  to  $F$  and, if  $E[v]$  is not the special flag value, also add  $E[v]$  to  $F$
  - c. Loop over the edges  $vw$  connecting  $v$  to other vertices  $w$ . For each such edge, if  $w$  still belongs to  $Q$  and  $vw$  has smaller weight than  $C[w]$ , perform the following steps:
    - I) Set  $C[w]$  to the cost of edge  $vw$
    - II) Set  $E[w]$  to point to edge  $vw$ .

Return  $F$

[https://en.wikipedia.org/wiki/Prim%27s\\_algorithm](https://en.wikipedia.org/wiki/Prim%27s_algorithm)

# Prim's algorithm



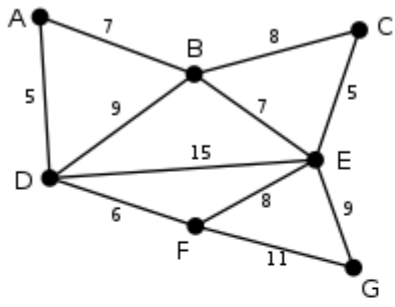
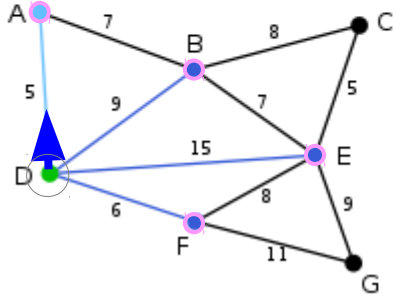
Prim's algorithm starting at vertex A.

In the third step, edges BD and AB both have weight 2, so BD is chosen arbitrarily.

After that step, AB is no longer a candidate for addition to the tree because it links two nodes that are already in the tree.

[https://en.wikipedia.org/wiki/Kruskal%27s\\_algorithm](https://en.wikipedia.org/wiki/Kruskal%27s_algorithm)

# Prim's algorithm examples (1)

Image	Description	Not seen	In the graph	In the tree
	<p>This is the initial weighted graph. It is not a tree, since to be a tree it is required that there are no cycles, and in this case there is. The numbers near the edges indicate the weight. None of the edges is marked, and vertex <b>D</b> has been chosen arbitrarily as the starting point.</p>	C, G	A, B, E, F	D
	<p>The second vertex is closest to <b>D</b> : <b>A</b> is 5 away, <b>B</b> is 9, <b>E</b> is 15, and <b>F</b> is 6. Of these, 5 is the smallest value, so we mark the <b>DA</b> edge.</p> <p>{5,6,9,15}</p>	C, G	B, E, F	A, D

[https://es.wikipedia.org/wiki/Algoritmo\\_de\\_Prim](https://es.wikipedia.org/wiki/Algoritmo_de_Prim)

# Prim's algorithm examples (2)

Image	Description	Not seen	In the graph	In the tree
	<p>The next vertex to choose is the closest to <b>D</b> or <b>A</b>. <b>B</b> is 9 away from <b>D</b> and 7 away from <b>A</b>, <b>E</b> is at 15, and <b>F</b> is at 6. 6 is the smallest value, so we mark the vertex <b>F</b> and the edge <b>DF</b>.</p>	C	B, E, G	A, D, F
	<p>The algorithm continues. The vertex <b>B</b>, which is at a distance of 7 from <b>A</b>, is the next one marked. At this point the edge <b>DB</b> is marked in red because its two ends are already in the tree and therefore can not be used.</p>	null	C, E, G	A, D, F, B

[https://es.wikipedia.org/wiki/Algoritmo\\_de\\_Prim](https://es.wikipedia.org/wiki/Algoritmo_de_Prim)

# Prim's algorithm examples (3)

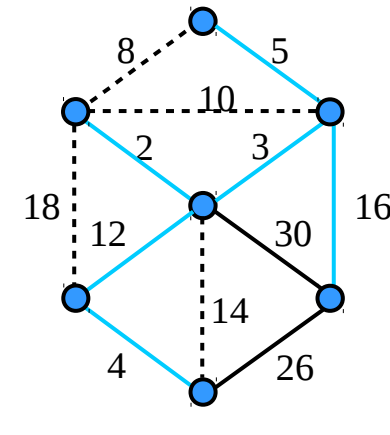
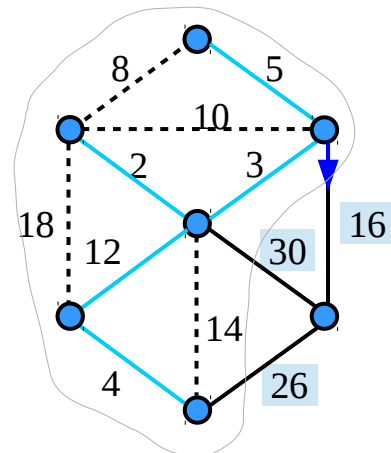
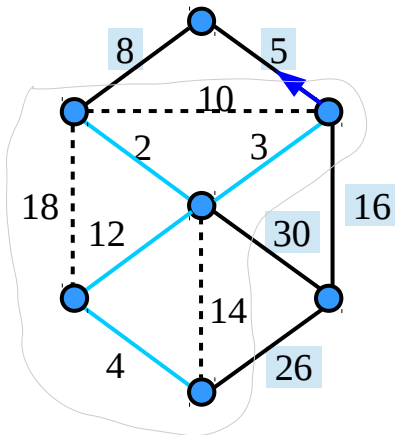
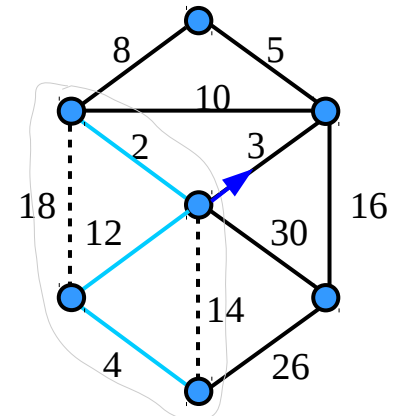
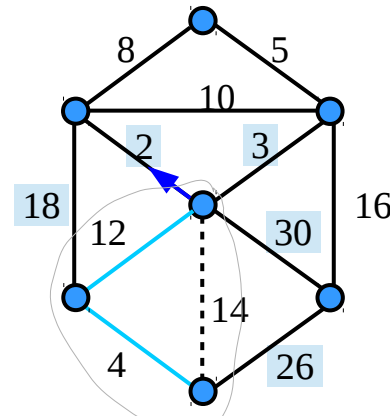
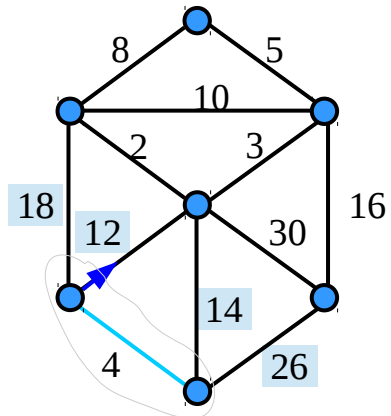
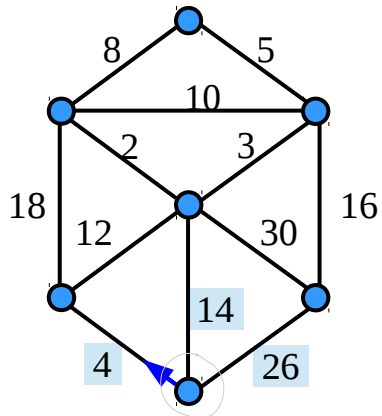
Image	Description	Not seen	In the graph	In the tree
	<p>Here you have to choose between <b>C</b> , <b>E</b> and <b>G</b> . <b>C</b> is 8 away from <b>B</b> , <b>E</b> is 7 away from <b>B</b> , and <b>G</b> is 11 away from <b>F</b> . <b>E</b> is closer, so we mark the vertex <b>E</b> and the edge <b>EB</b> . Two other edges were marked in red because both vertices that join were added to the tree.</p>	null	C, G	A, D, F, B, E
	<p>Only <b>C</b> and <b>G</b> are available. <b>C</b> is 5 away from <b>E</b> , and <b>G</b> is 9 away from <b>E</b> . Choose <b>C</b> , and mark with the arc <b>EC</b> . The <b>BC</b> arc is also marked with red.</p>	null	G	A, D, F, B, E, C
	<p><b>G</b> is the only outstanding vertex, and it is closer to <b>E</b> than to <b>F</b> , so <b>EG</b> is added to the tree. All vertices are already marked, the <b>minimum expansion tree</b> is shown in green. In this case with a weight of 39.</p>	null	null	A, D, F, B, E, C, G

[https://es.wikipedia.org/wiki/Algoritmo\\_de\\_Prim](https://es.wikipedia.org/wiki/Algoritmo_de_Prim)



# Prim's algorithm examples (4)

{2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30}



<http://jeffe.cs.illinois.edu/teaching/algorithms/notes/20-mst.pdf>

## References

- [1] <http://en.wikipedia.org/>
- [2]