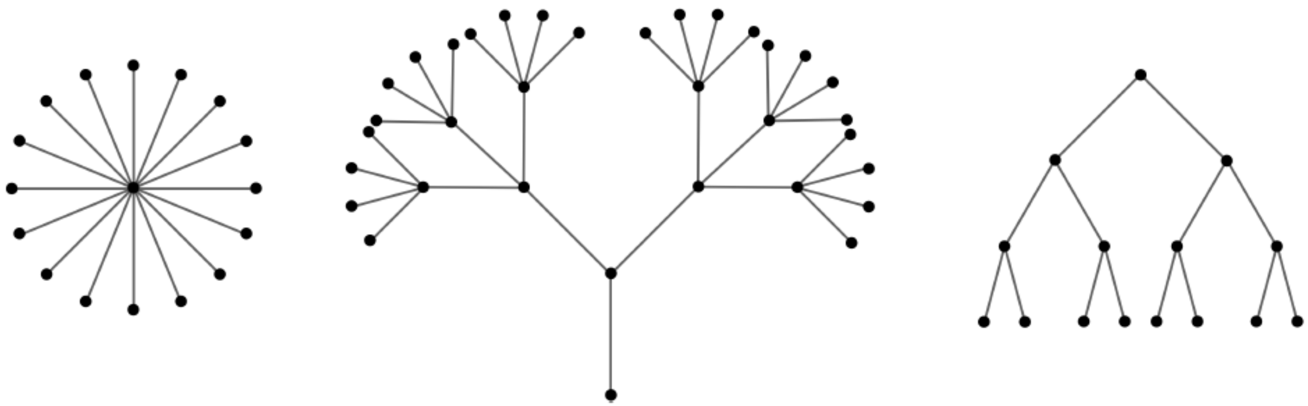
Trees

Olli-Pekka Hämäläinen

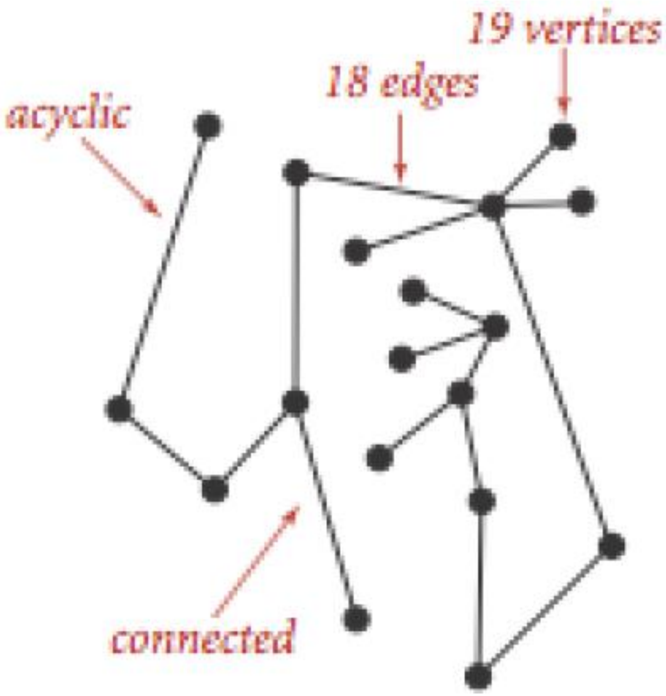
Tree

- An undirected graph which contains no cycles is called a *tree*
- All nodes of the tree are connected to each other by unique paths
- As a definition we could say that “a tree is a connected acyclic graph”



Tree

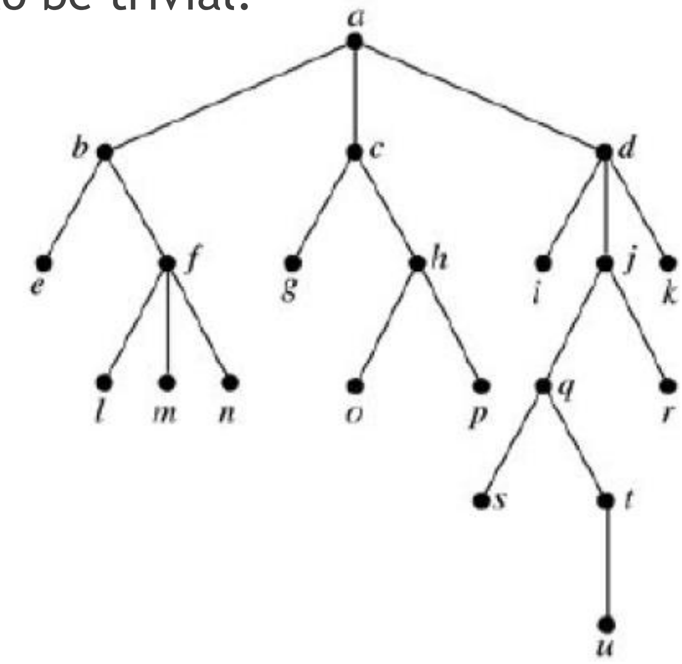
- Because all nodes of the tree are connected to each other via just one edge, we can formulate a simple rule for the number of edges and nodes (vertices):



$$\boldsymbol{Number\ of\ edges}$$
$$\boldsymbol{= Number\ of\ nodes - 1}$$

Rooted tree

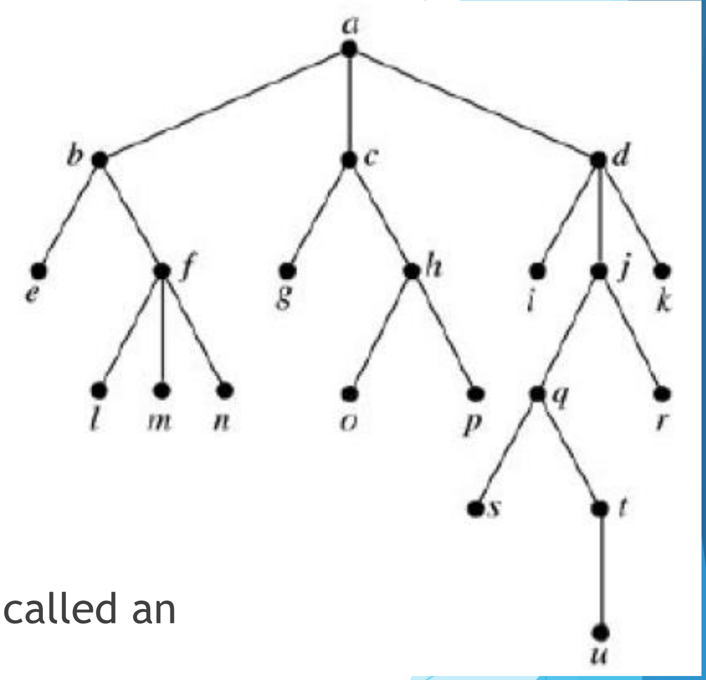
- One tree type that is essential for many applications (especially in data structures & computer science) is a *rooted tree*, where the tree originates from one node
- Rooted tree is actually directed, but the directions of edges are usually not illustrated by arrows, because the directions are thought to be trivial:
  - Root is on the top
  - Edges point down



For example, the root of this rooted tree is clearly *a*.

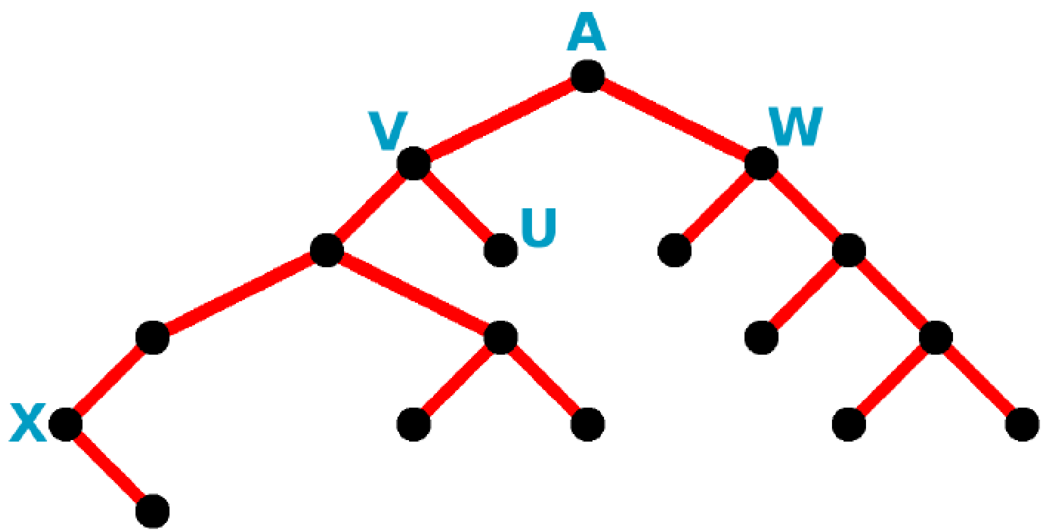
Terminology of a rooted tree

- Rooted tree has its origins in non-mathematical background - family trees!
- This shows in terminology:
  - *e* and *f* are *b's children*
  - *b* is a *parent* of *f*
  - *l*, *m* and *n* are *siblings*
  - *q*, *r*, *s*, *t* and *u* are *j's descendants*
  - *j* is the *ancestor* of nodes *q*, *r*, *s*, *t* and *u*
  - Childless nodes are called *leaves*
  - A node which is neither a root nor a leaf is called an *internal node*



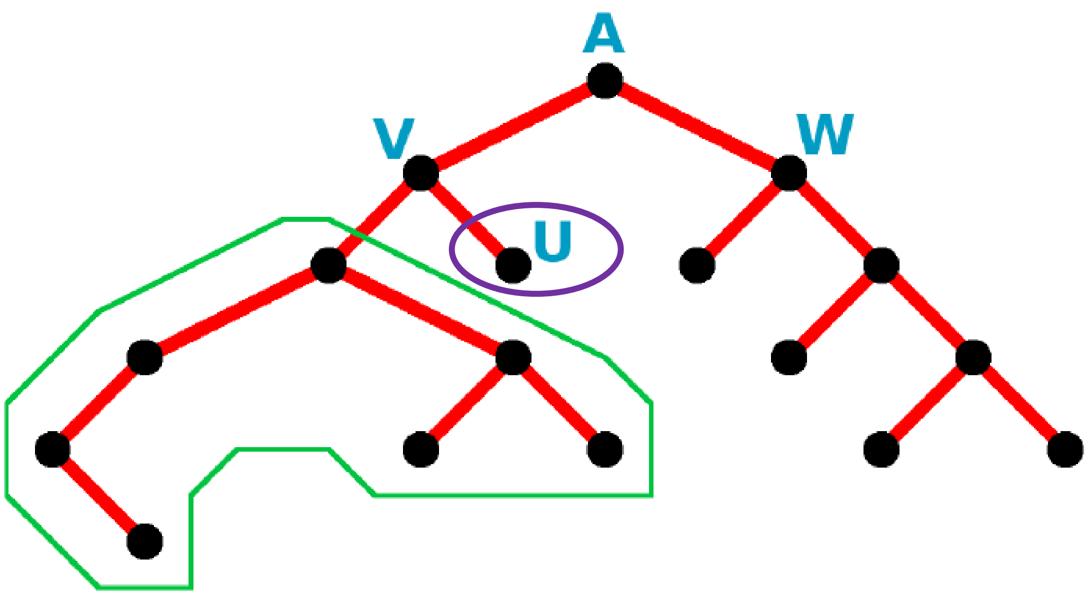
Binary tree

- A special case of a rooted tree is a *binary tree*, where each node can only have two children at max
- Children are classified as *left child* and *right child*
- For example, in the binary tree below, A has two children: V is the left child and W is the right child
- Node X only has one child (right one; unnamed here)



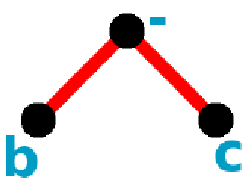
Subtree

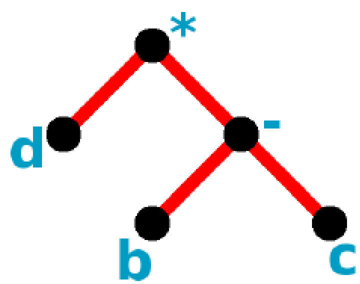
- A binary tree can be cut at any point
- The resulting cut-off parts are called *subtrees*
  - For example, if we cut the tree in here below node V, the part delimited by green line is left subtree of V
  - Right subtree of V is just the node U

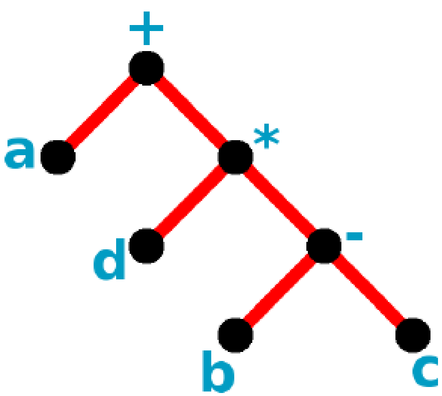


Expression tree

- One use of binary trees is to present explicitly in which order calculations are performed
- A tree that serves this purpose is called an *expression tree*
- Notice the importance of child roles!
  - Left or right?







$\boldsymbol{b - c}$

$\boldsymbol{d * (b - c)}$

$\boldsymbol{a + d * (b - c)}$

Binary search tree

- *Binary search tree* is a tree-shaped hierarchic database structure
- Ordering of the tree is based on emphasized roles of left and right child:
  - Element of “lesser value” is always set as left child and element of “greater value” is set as right child
  - In forthcoming examples, we deal with elements of numeric values, but also other kinds of data can be organized using the same principle
  - For example, names are ordered to left and right children according to alphabetical order
- Information can be added and deleted from the database
- Also, a search function must be enabled (otherwise our database would be a quite useless one)

Binary search tree

- Let's examine how the search tree is built up when elements are added to it; we compose the search tree from the following elements in respective order:

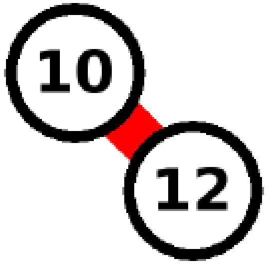
$$\{10, 12, 5, 4, 20, 8, 7, 15, 13\}.$$

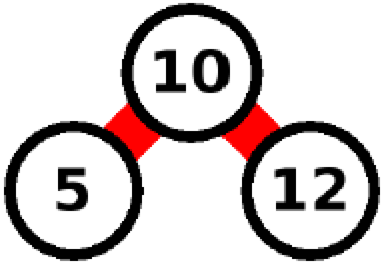
Binary search tree

- Let’s examine how the search tree is built up when elements are added to it; we compose the search tree from the following elements in respective order:

$$\{10, 12, 5, 4, 20, 8, 7, 15, 13\}.$$

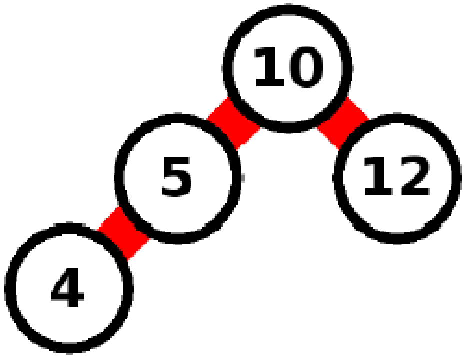
  - First the tree is empty, so the first element (10) will be the root of the tree
  - 12 is greater than 10, so it will be added as right child
  - 5 is smaller than 10, so it will be added as left child

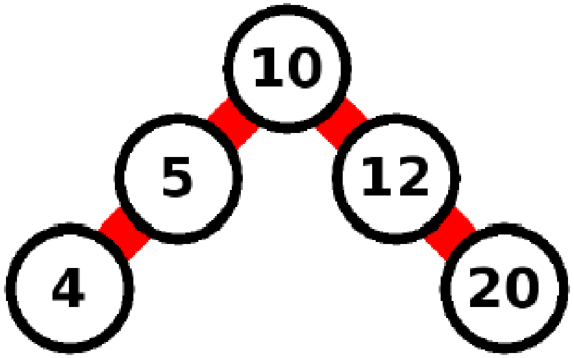




Binary search tree

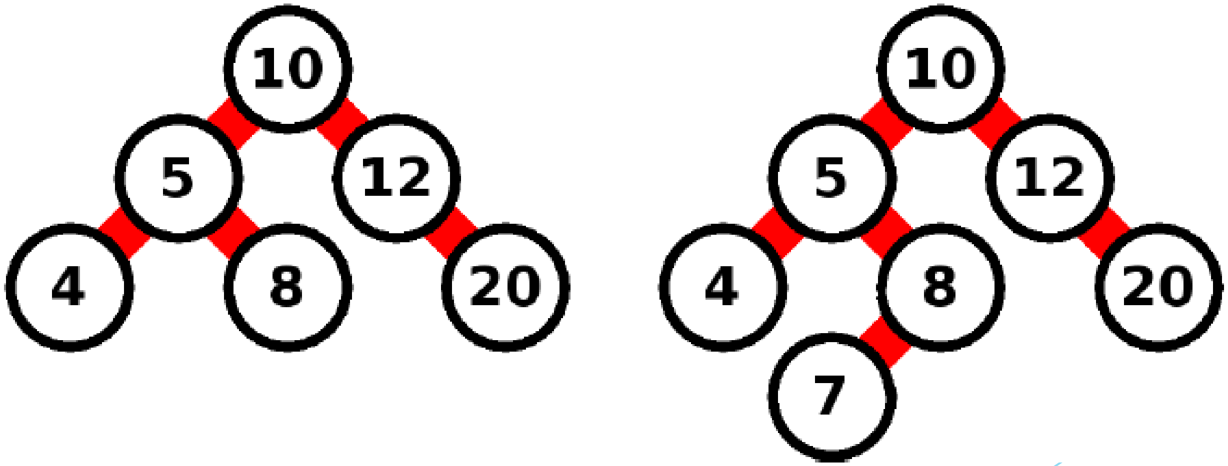
- Next we add element 4; it is smaller than 10, so it goes to left branch. Left child already exists, so we compare the new element to that: 4 < 5, so 4 will be the left child of 5
- Next we add 20; it is greater than 10, so it goes to right branch. Right child already exists, so we compare the new element to that: 20 > 12, so 20 will be the right child of 12.





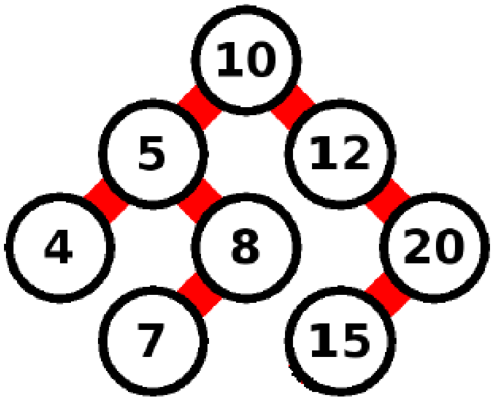
Binary search tree

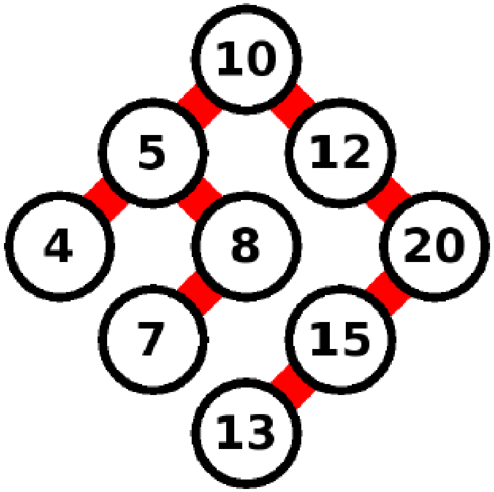
  - Next we add element 8; it is smaller than 10, so it goes to left branch, and it's greater than 5, so it takes the place of right child of 5.
  - Next we add element 7; it conquers itself the place as the left child of 8
- So, in each node we perform a comparison
  - Result defines whether we go to left or right



Binary search tree

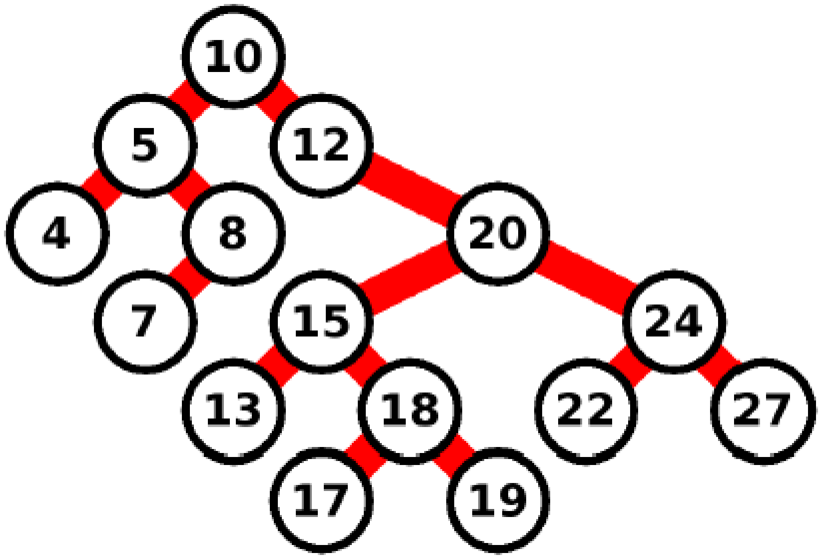
- Let’s add the last elements 15 and 13
- Comparison operation defines whether the element will be the left or right child, if that spot is available
- If the spot is already taken, we make a new comparison





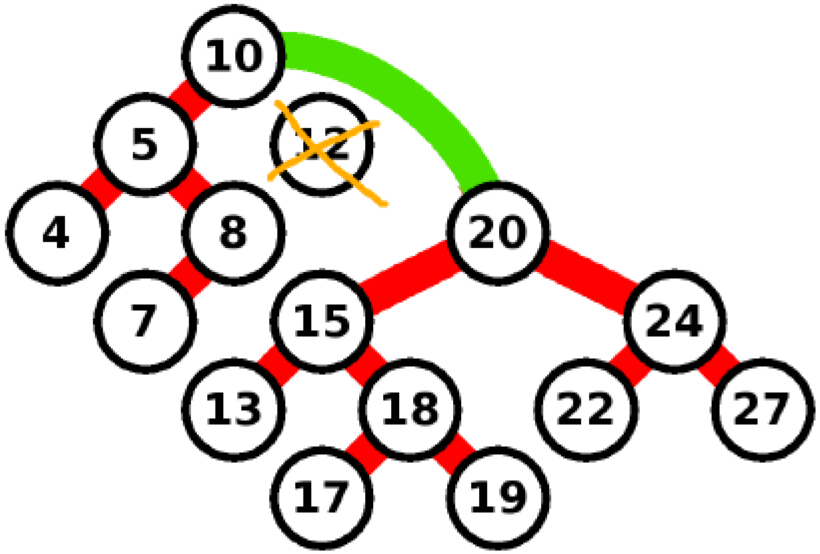
Binary search tree

- Let's add a couple of more elements: $\{18,24,17,27,22,19\}$
- By following the previously mentioned rules, we end up with the tree below
- From this tree it's easy to search for an element:
  - Perform comparison operation in each node in order to find out to which branch we target our search
  - If the desired element is not found, it's not in the tree



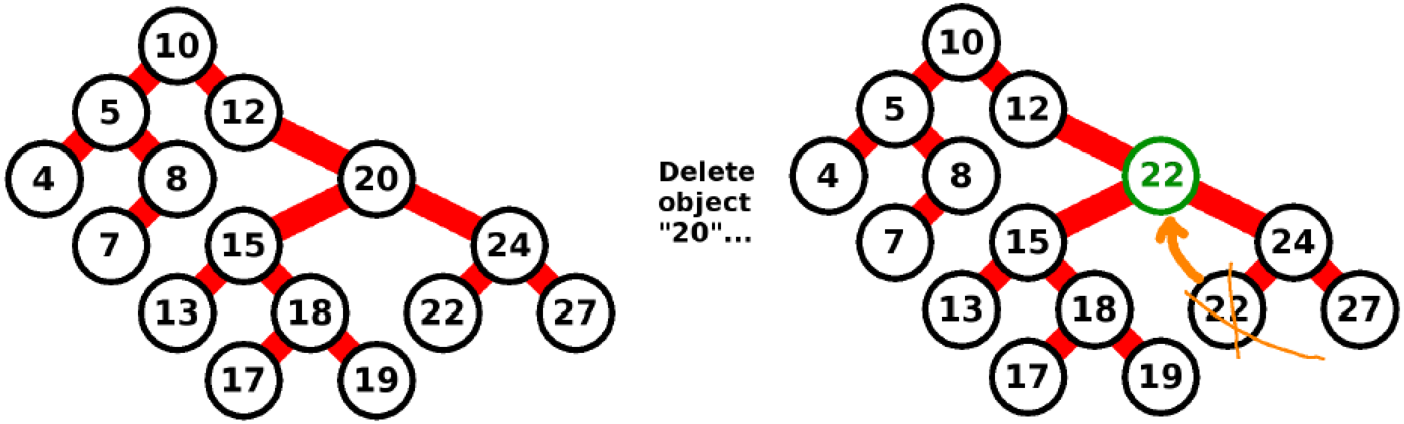
Binary search tree

- Deleting an element from the search tree is easy, if the element has no children; we can just eliminate the element without any other measures
- If the element has one child, then we have to attach the child with its subtree in the place of removed element
  - For example, if we delete element 12 from our tree:



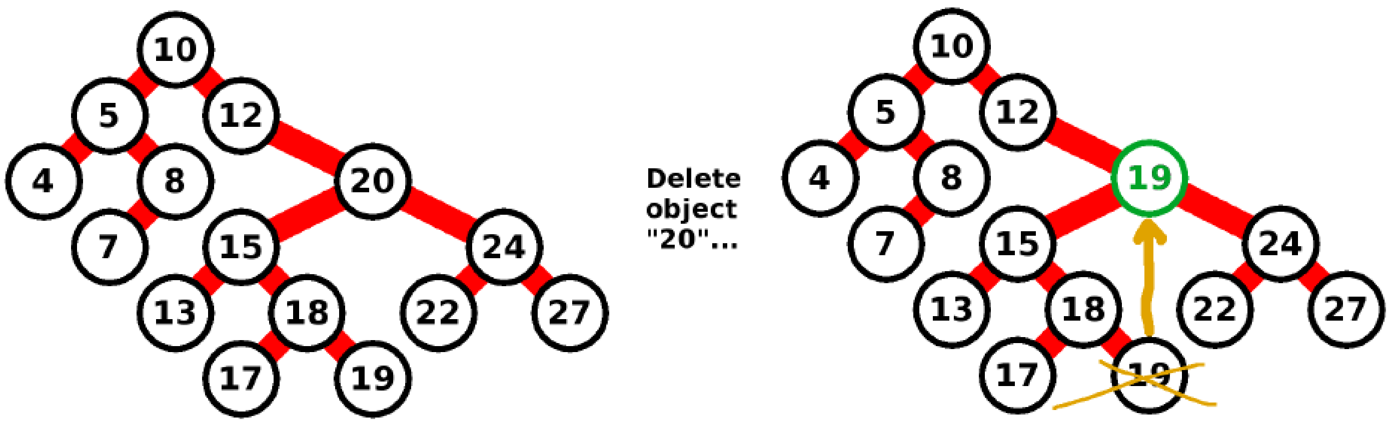
Binary search tree

- If the element to be deleted has two children, we have two alternative methods:
  - 1) We find the smallest element from the right subtree and move that in the place of removed element
  - For example, if we delete the element 20 from our search tree, we find the smallest element of the right subtree (22) and replace the 20 by that



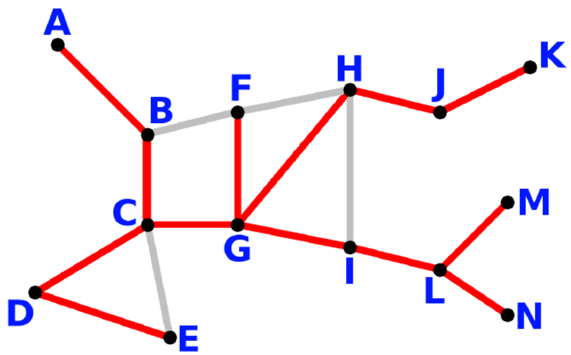
Binary search tree

- The other possibility is to examine the left subtree:
  - 2) We find the greatest element from the left subtree and move that in the place of our removed element
  - For example, in the case of deleting the 20 in our tree, we find the greatest element from our left subtree (19) and replace the 20 by that



Spanning tree

- Let's go back to our cyclic graphs: in certain situations, it makes sense to think of ways to modify the graph to as simple as possible but in such a way that there exists a connection between each node
  - For example, internet connections between cities; data moves so quickly that the physical connection doesn't need to be as short as possible – as long as it exists
- In these cases, the graph should be simplified to a tree by removing edges until all cyclic structures are gone
- This kind of tree is called the *spanning tree* of the graph



Minimal spanning tree (MST)

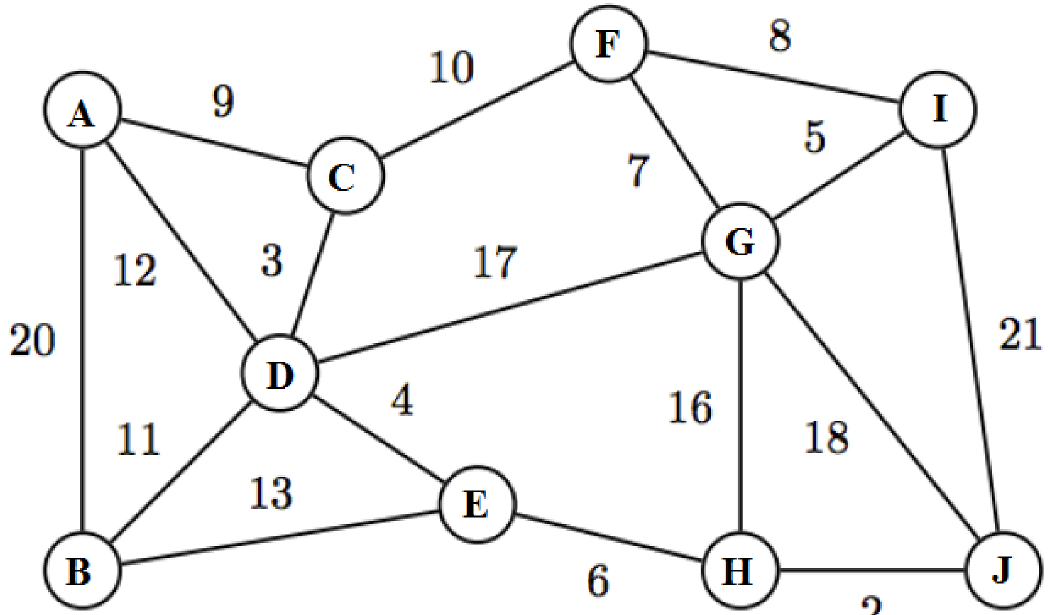
- Especially with weighted graphs, our interest lies on optimization: which of the possible spanning trees has the lightest total weight?
- This kind of lightest possible tree is called *minimal spanning tree (MST)*
- Several algorithms exist for finding such a tree:
  - Prim's algorithm
  - Kruskal's algorithm
  - etc.
- Let's go through *Prim's algorithm*, because it is reasonably efficient and the idea is easy to understand – and it can be ran using matrices!
  - In dense graphs Prim is also faster than Kruskal

Prim's algorithm

- Prim's algorithm is very simple:
  - Start from an arbitrary node (NOTE! Start node selection has no effect on the end result!)
  - Examine which edge of the ones connected to our labeled nodes is the lightest
  - Label the node where this edge goes, if said node doesn't belong to our group of labeled nodes already
  - Continue until all nodes are labeled
- Simplest to run in matrix form just like Dijkstra:
  - This time we're only interested in weights of sole edges – not the weight of the path from start node
  - Emergence of cyclic structures is prevented by removing the columns of already labeled nodes

Prim's algorithm: FBI example

- The graph below depicts FBI agents A…J
- Edge weights portray the exposure probabilities of information sharing channels between agents
- Extra connections need to be eliminated in order to reduce security risks; what's the MST?



Prim's algorithm: FBI example

- Formulate adjacency matrix from the graph
- Add two columns: "W" for weight of the selected edge and "P" for which node the edge connects to

	A	B	C	D	E	F	G	H	I	J	W	P
A		20	9	12								
B	20			11	13							
C	9			3		10						
D	12	11	3		4		17					
E		13		4				6				
F			10				7		8			
G				17		7		16	5	18		
H					6		16			2		
I						8	5			21		
J							18	2	21			

Prim's algorithm: FBI example

- Start from node A; then W = 0, and there are no edges yet
- A has now been labeled, so remove its column

	A	B	C	D	E	F	G	H	I	J	W	P
A		20	9	12							0	-
B	20			11	13							
C	9			3		10						
D	12	11	3		4		17					
E		13		4				6				
F			10				7		8			
G				17		7		16	5	18		
H					6		16			2		
I						8	5			21		
J							18	2	21			

Prim's algorithm: FBI example

- The lightest edge that is connected to A is 9, which takes us to node C; label C and remove its column

	A	B	C	D	E	F	G	H	I	J	W	P
A		20	9	12							0	-
B	20			11	13							
C	9			3		10					9	A
D	12	11	3		4		17					
E		13		4				6				
F			10				7		8			
G				17		7		16	5	18		
H					6		16			2		
I						8	5			21		
J							18	2	21			

Prim’s algorithm: FBI example

- The lightest edge that is connected to A or C is 3, which takes us to node D; label D and remove its column

	A	B	C	D	E	F	G	H	I	J	W	P
A		20	9	12							0	-
B	20			11	13							
C	9			3		10					9	A
D	12	11	3		4		17				3	C
E		13		4				6				
F			10				7		8			
G				17		7		16	5	18		
H					6		16			2		
I						8	5			21		
J							18	2	21			

Prim’s algorithm: FBI example

- Our group of labeled nodes now consists of A, C and D. The lightest edge that is connected to these is 4, which takes us to E. Label E and remove its column.

	A	B	C	D	E	F	G	H	I	J	W	P
A		20	9	12							0	-
B	20			11	13							
C	9			3		10					9	A
D	12	11	3		4		17				3	C
E		13		4				6			4	D
F			10				7		8			
G				17		7		16	5	18		
H					6		16			2		
I						8	5			21		
J							18	2	21			

Prim's algorithm: FBI example

- Group of labeled nodes consists of A, C, D and E. Lightest edge is 6 (EH), so label H and remove its column.

	A	B	C	D	E	F	G	H	I	J	W	P
A		20	9	12							0	-
B	20			11	13							
C	9			3		10					9	A
D	12	11	3		4		17				3	C
E		13		4				6			4	D
F			10				7		8			
G				17		7		16	5	18		
H					6		16			2	6	E
I						8	5			21		
J							18	2	21			

Prim's algorithm: FBI example

- Group of labeled nodes consists of A, C, D, E and H. Lightest edge is 2 (HJ), so label J and remove its column.

	A	B	C	D	E	F	G	H	I	J	W	P
A		20	9	12							0	-
B	20			11	13							
C	9			3		10					9	A
D	12	11	3		4		17				3	C
E		13		4				6			4	D
F			10				7		8			
G				17		7		16	5	18		
H					6		16			2	6	E
I						8	5			21		
J							18	2	21		2	H

Prim's algorithm: FBI example

- Group of labeled nodes consists of A, C, D, E, H and J. Lightest edge is 10 (CF), so label F and remove its column.

	A	B	C	D	E	F	G	H	I	J	W	P
A		20	9	12							0	-
B	20			11	13							
C	9			3		10					9	A
D	12	11	3		4		17				3	C
E		13		4				6			4	D
F			10				7		8		10	C
G				17		7		16	5	18		
H					6		16			2	6	E
I						8	5			21		
J							18	2	21		2	H

Prim’s algorithm: FBI example

- Group of labeled nodes consists of A, C, D, E, H, J and F. Lightest edge is 7 (FG), so label G and remove its column.

	A	B	C	D	E	F	G	H	I	J	W	P
A		20	9	12							0	-
B	20			11	13							
C	9			3		10					9	A
D	12	11	3		4		17				3	C
E		13		4				6			4	D
F			10				7		8		10	C
G				17		7		16	5	18	7	F
H					6		16			2	6	E
I						8	5			21		
J							18	2	21		2	H

Prim's algorithm: FBI example

- Group of labeled nodes consists of A, C, D, E, H, J, F and G. Lightest edge is 5 (GI), so label I and remove its column.

	A	B	C	D	E	F	G	H	I	J	W	P
A		20	9	12							0	-
B	20			11	13							
C	9			3		10					9	A
D	12	11	3		4		17				3	C
E		13		4				6			4	D
F			10				7		8		10	C
G				17		7		16	5	18	7	F
H					6		16			2	6	E
I						8	5			21	5	G
J							18	2	21		2	H

Prim's algorithm: FBI example

- Now the only node that is left to label is B. The lightest edge that takes us here is 11 (DB), so label B (and remove its column).

	A	B	C	D	E	F	G	H	I	J	W	P
A		20	9	12							0	-
B	20			11	13						11	D
C	9			3		10					9	A
D	12	11	3		4		17				3	C
E		13		4				6			4	D
F			10				7		8		10	C
G				17		7		16	5	18	7	F
H					6		16			2	6	E
I						8	5			21	5	G
J							18	2	21		2	H

Prim’s algorithm: FBI example

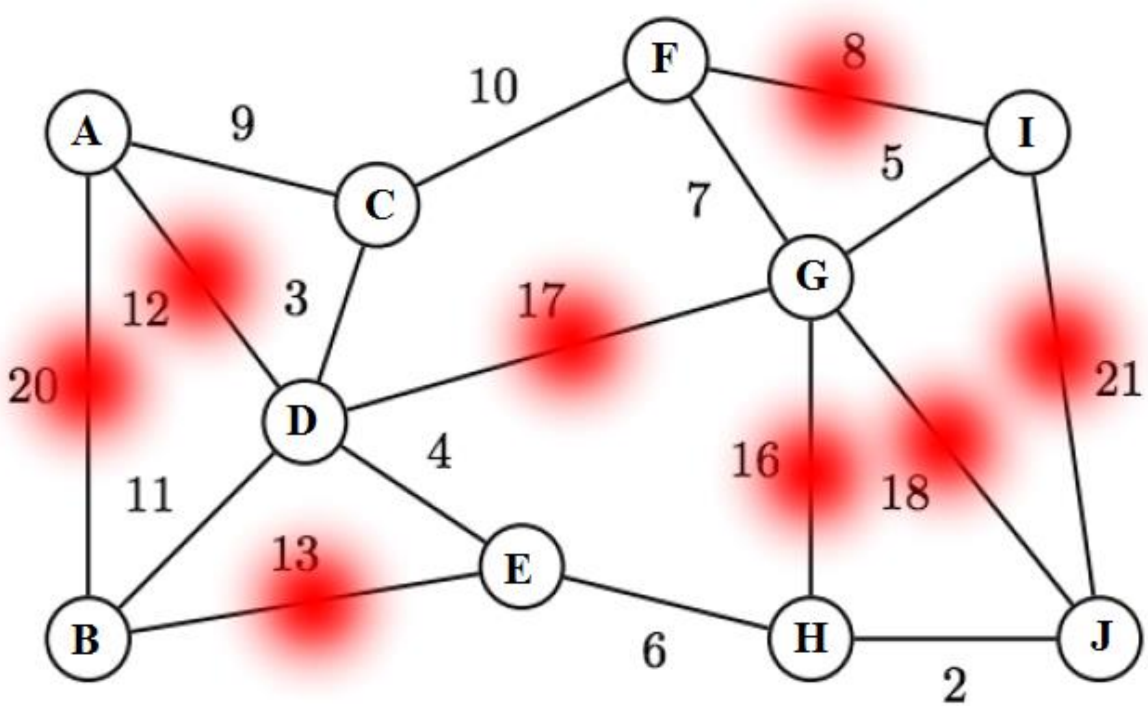
- Algorithm is done! Now the weight of our minimal spanning tree can be calculated by summing up the edge weights of column W:

	A	B	C	D	E	F	G	H	I	J	W	P
A		20	9	12							0	-
B	20			11	13						11	D
C	9			3		10					9	A
D	12	11	3		4		17				3	C
E		13		4				6			4	D
F			10				7		8		10	C
G				17		7		16	5	18	7	F
H					6		16			2	6	E
I						8	5			21	5	G
J							18	2	21		2	H

$$\sum W = 57$$

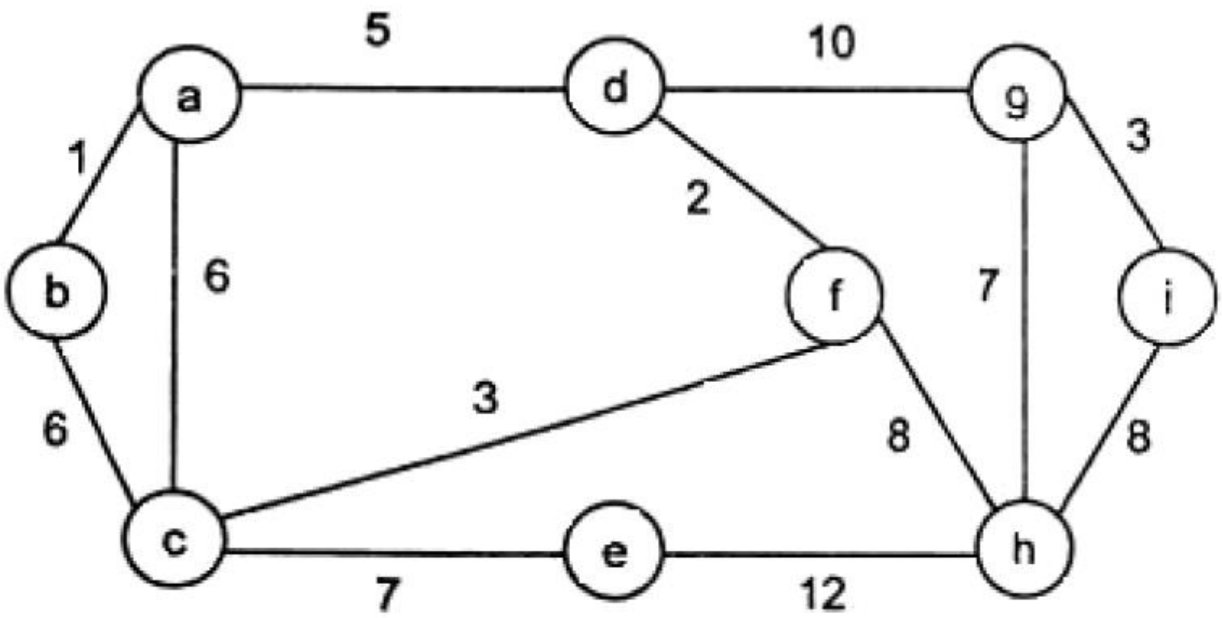
Prim’s algorithm: FBI example

- So, the connections marked in red will be eliminated
- The remaining edges formulate the MST



Prim’s algorithm: Neste example

- Neste Oyj has an oil refinery with 9 refining points
- All points must be supplied with crude oil via pipes
- What is the cheapest possible pipe tree, if edge weights represent the costs of said pipelines?



Prim’s algorithm: Neste example

- As in previous example, formulate the adjacency matrix and add weight and node columns W and P:

	A	B	C	D	E	F	G	H	I	W	P
A		1	6	5							
B	1		6								
C	6	6			7	3					
D	5					2	10				
E			7					12			
F			3	2				8			
G				10				7	3		
H					12	8	7		8		
I							3	8			

Prim's algorithm: Neste example

- Start from node A. A has been labeled, so remove its column:

	A	B	C	D	E	F	G	H	I	W	P
A		1	6	5						0	-
B	1		6								
C	6	6			7	3					
D	5					2	10				
E			7					12			
F			3	2				8			
G				10				7	3		
H					12	8	7		8		
I							3	8			

Prim's algorithm: Neste example

- Lightest edge connected to A is 1 (AB), so label B and remove the B-column:

	A	B	C	D	E	F	G	H	I	W	P
A		1	6	5						0	-
B	1		6							1	A
C	6	6			7	3					
D	5					2	10				
E			7					12			
F			3	2				8			
G				10				7	3		
H					12	8	7		8		
I							3	8			

Prim’s algorithm: Neste example

- A and B have been labeled. Lightest edge from these rows is 5 (AD), so label D and remove the D-column:

	A	B	C	D	E	F	G	H	I	W	P
A		1	6	5						0	-
B	1		6							1	A
C	6	6			7	3					
D	5					2	10			5	A
E			7					12			
F			3	2				8			
G				10				7	3		
H					12	8	7		8		
I							3	8			

Prim's algorithm: Neste example

- Labeled nodes are A, B and D. Lightest edge from these rows is 2 (DF), so label F and remove the F-column:

	A	B	C	D	E	F	G	H	I	W	P
A		1	6	5						0	-
B	1		6							1	A
C	6	6			7	3					
D	5					2	10			5	A
E			7					12			
F			3	2				8		2	D
G				10				7	3		
H					12	8	7		8		
I							3	8			

Prim's algorithm: Neste example

- Labeled nodes are A, B, D and F. Lightest edge from these rows is 3 (FC), so label C and remove the C-column:

	A	B	C	D	E	F	G	H	I	W	P
A		1	6	5						0	-
B	1		6							1	A
C	6	6			7	3				3	F
D	5					2	10			5	A
E			7					12			
F			3	2				8		2	D
G				10				7	3		
H					12	8	7		8		
I							3	8			

Prim's algorithm: Neste example

- Labeled nodes are A, B, D, F and C. Lightest edge from these rows is 7 (CE), so label E and remove the E-column:

	A	B	C	D	E	F	G	H	I	W	P
A		1	6	5						0	-
B	1		6							1	A
C	6	6			7	3				3	F
D	5					2	10			5	A
E			7					12		7	C
F			3	2				8		2	D
G				10				7	3		
H					12	8	7		8		
I							3	8			

Prim’s algorithm: Neste example

- Labeled nodes are A, B, D, F, C and E. Lightest edge from these rows is 8 (FH), so label H and remove the H-column:

	A	B	C	D	E	F	G	H	I	W	P
A		1	6	5						0	-
B	1		6							1	A
C	6	6			7	3				3	F
D	5					2	10			5	A
E			7					12		7	C
F			3	2				8		2	D
G				10				7	3		
H					12	8	7		8	8	F
I							3	8			

Prim's algorithm: Neste example

- Labeled nodes are A, B, D, F, C, E and H. Lightest edge from these rows is 7 (HG), so label G and remove the G-column:

	A	B	C	D	E	F	G	H	I	W	P
A		1	6	5						0	-
B	1		6							1	A
C	6	6			7	3				3	F
D	5					2	10			5	A
E			7					12		7	C
F			3	2				8		2	D
G				10				7	3	7	H
H					12	8	7		8	8	F
I							3	8			

Prim’s algorithm: Neste example

- Now the only node to be labeled is I. The lightest edge that takes us there is 3 (GI), so label I and remove its column:

	A	B	C	D	E	F	G	H	I	W	P
A		1	6	5						0	-
B	1		6							1	A
C	6	6			7	3				3	F
D	5					2	10			5	A
E			7					12		7	C
F			3	2				8		2	D
G				10				7	3	7	H
H					12	8	7		8	8	F
I							3	8		3	G

Prim’s algorithm: Neste example

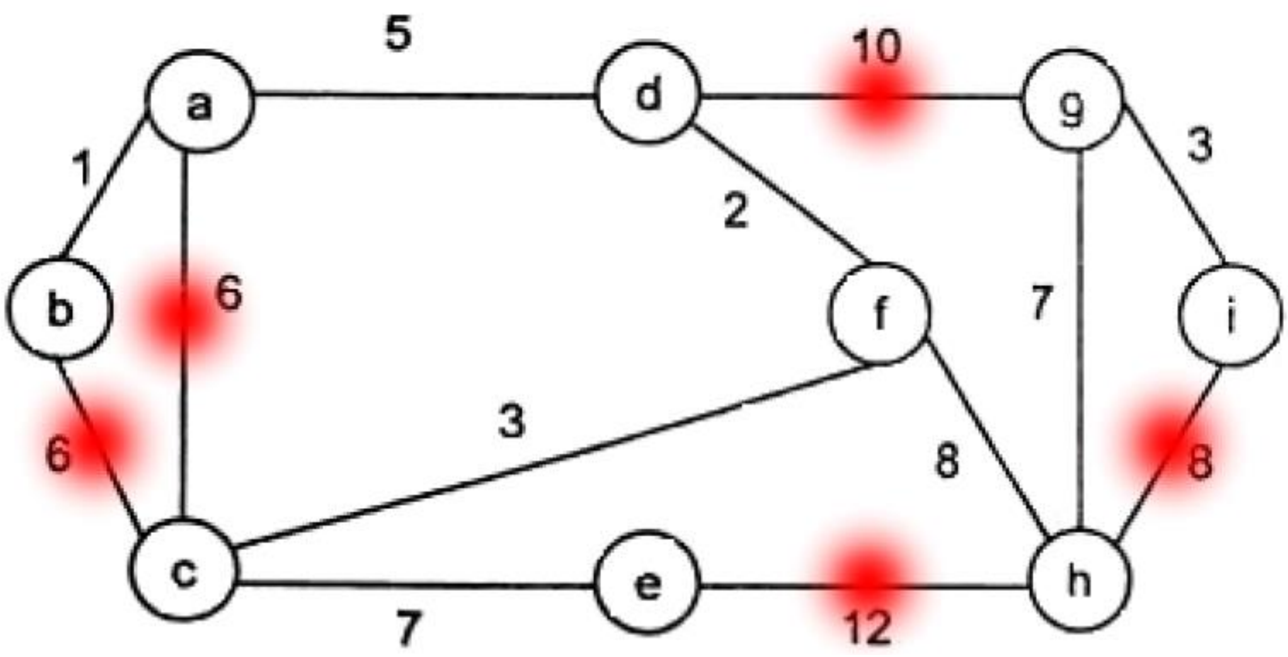
- Algorithm is ready! The total weight of the MST can be calculated by summing up the edge weights of column W:

	A	B	C	D	E	F	G	H	I	W	P
A		1	6	5						0	-
B	1		6							1	A
C	6	6			7	3				3	F
D	5					2	10			5	A
E			7					12		7	C
F			3	2				8		2	D
G				10				7	3	7	H
H					12	8	7		8	8	F
I							3	8		3	G

$$\sum W = 36$$

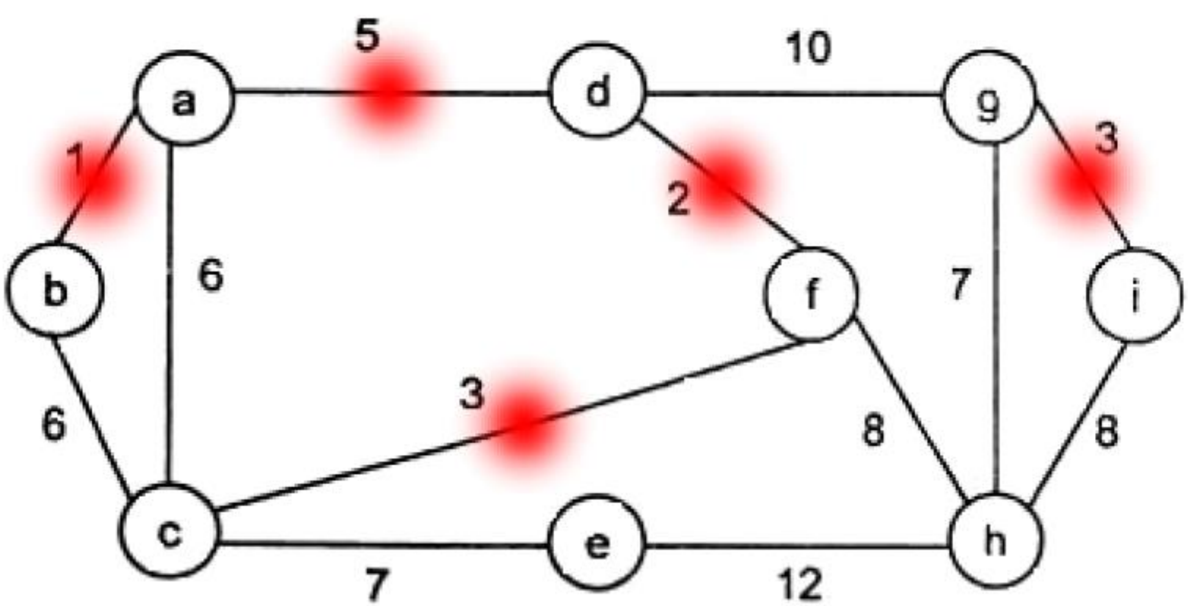
Prim’s algorithm: Neste example

- So, the pipelines marked in red will not be built
- The remaining pipelines formulate the MST



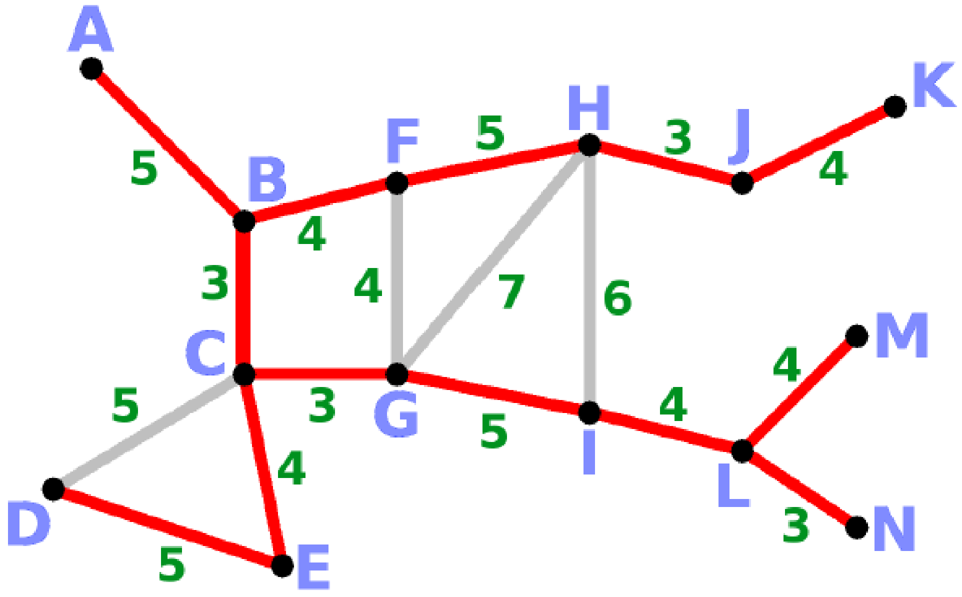
Additional notes on MST

- In some cases we might want to find out the *maximal spanning tree* instead – so, maximize the edge weights
- This can be done by Prim's algorithm as well – we just modify the edge selection criterion!
- For example, the maximal spanning tree of the graph in previous example (e.g., if the weights were pipeline capacities which we want to maximize):



Additional notes on MST

- As we noticed with Dijkstra's algorithm, the optimal solution is not always unique; there may be several solutions which are equivalently good
- For example, the MST of the graph below could as well include CD instead of DE or FG instead of FB
  - Both would produce the same total weight!



Thank you!