Minimal Spanning Trees and Jarník-Prim's Algorithm

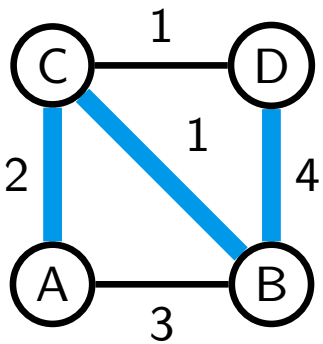
Minimal spanning tree

Assumption: Consider only *undirected* and *connected* graphs!

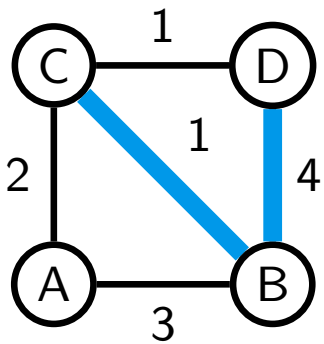
A **spanning tree** is a minimal possible selection of edges that connects all vertices. (That is, a spanning tree does not contain any cycles.)

Minimum spanning tree is a spanning tree such that the sum of weights of its edges is the minimal such.

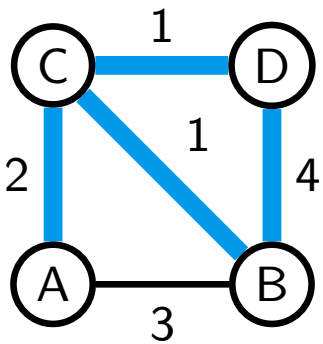
Example



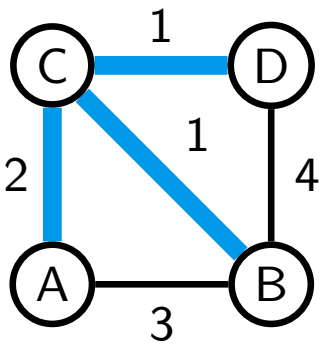
Spanning tree.



Not a spanning tree. A is not connected.



Not a spanning tree. Any of the edges CB, CD, BD could be removed.

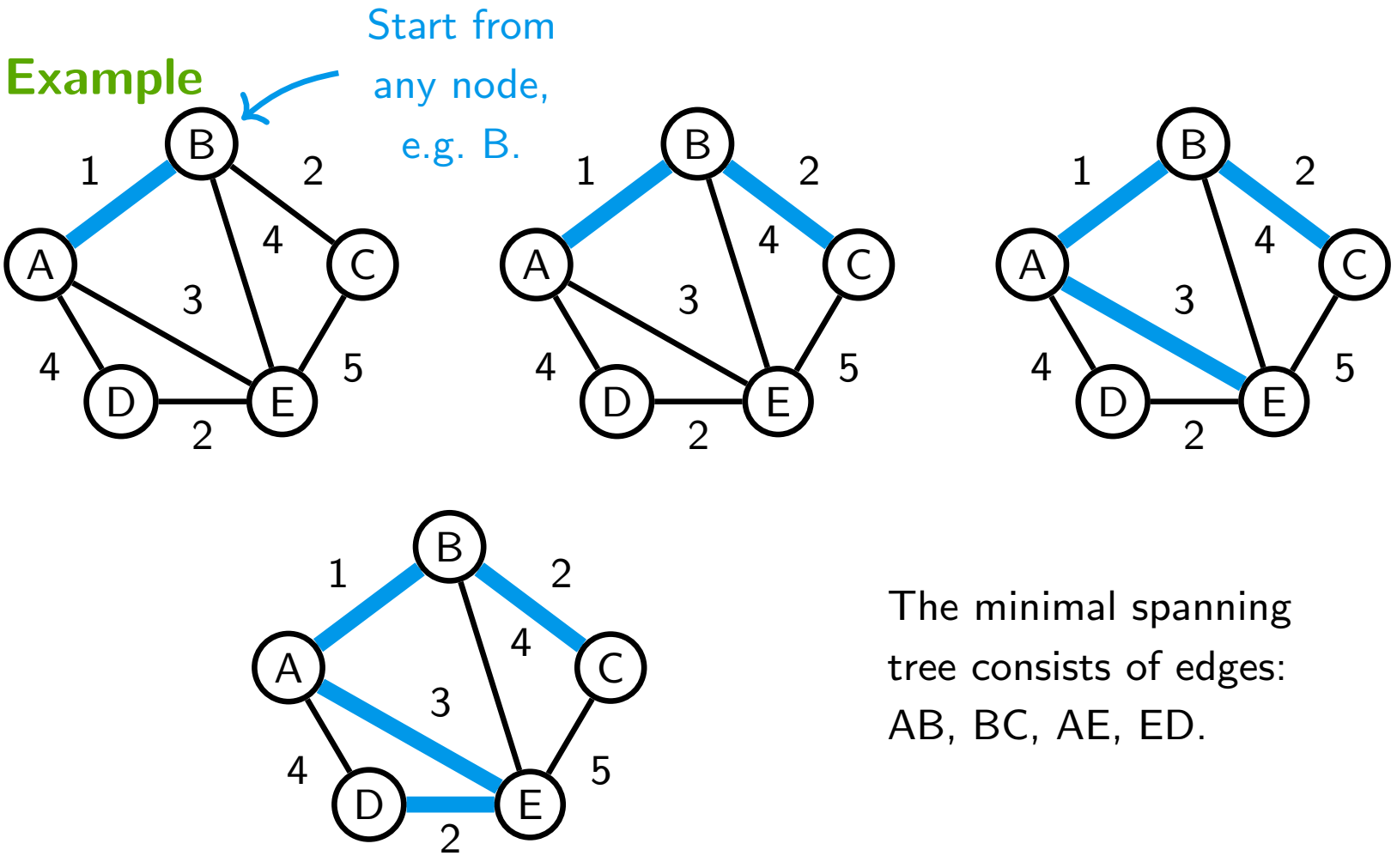


Minimum spanning tree!

Example: Execution of Jarník-Prim algorithm

Idea: Iteratively extend the tree with an edge which has the smallest weight and which connects a yet unconnected node.

Example



The minimal spanning tree consists of edges: AB, BC, AE, ED.

Jarník-Prim algorithm for finding the minimal spanning tree

For each vertex `w` of the graph, we keep track of the following:

i. `d[w] =` the current distance from the *tree* (initially: ∞)
ii. `p[w] =` the vertex which connects to the tree (initially: `w`)
iii. `f[w] =` has `w` been added to the tree? (initially: `false`)

The algorithm (idea):

1: set `d[0] = 0` (vertex `0` could be replaced by any other vertex)
2: while there are unfinished vertices:
3: set `w =` the yet unfinished vertex with the smallest `d[w]`
4: set `f[w] = true` (i.e. mark `w` as *finished*)
5: for every neighbour `u` of `w`:
6: if `weight(w,u) < d[u]`:
7: set `d[u] = weight(w,u)` and `p[u] = w`

(Where `weight(w,u)` is the weight of the edge `w` $\rightarrow$ `u`)

Jarník-Prim's algorithm works similarly to Dijkstra's algorithm. The differences are marked by red. Altough the principle is similar, the interpretation of the execution and the result is different. The main idea of Jarník-Prim is that we are growing a minimal spanning tree iteratively

The following is an invariant for Jarník-Prim:

1. The vertices marked as finished are connected/added to the tree.
2. For those vertices which are not connected yet, `d[w]` denotes the smallest weight of an edge that connects `w` to the tree.
3. `p[w]` denotes the vertex of the tree such that the edge between `w` and `p[w]` is the edge with weight `d[w]`.

In steps 5-7, we reduce the weights (from ∞) of the nodes that connect to last added finished node, so that they can be considered for adding in the next iteration.

After the algorithm finishes, i.e. all vertices are marked finished, we can read out the spanning tree from the array `p[-]`. To obtain the minimum spanning tree, for every vertex `w` (except for `w == 0`), add the edge `w` — `p[w]`.

Jarník-Prim's time complexity

The time complexity is the same as for Dijkstra's algorithm!

Adjacency matrices	Adjacency lists	
	Binary Heaps	Fibonacci Heaps
$O(n^2)$	$O((n+m)\log n)$	$O((n\log n)+m)$

Remark: Unlike Dijkstra's algorithm, Jarník-Prim's algorithm would also work for graphs with edges that have negative weights.