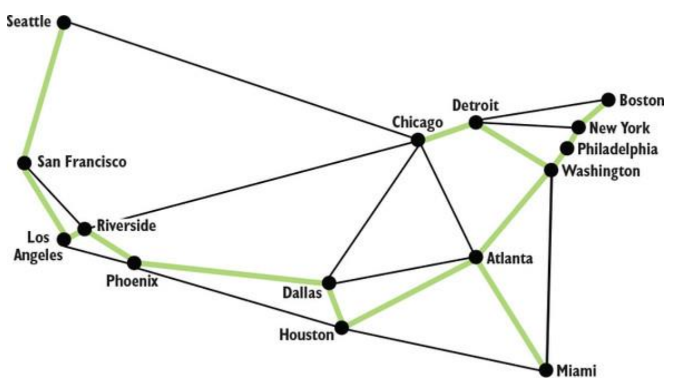
Minimum Spanning Trees

Lecture 05.03

by Marina Barsky

Motivation

- Connect all the computers in a new office building using the least amount of cable

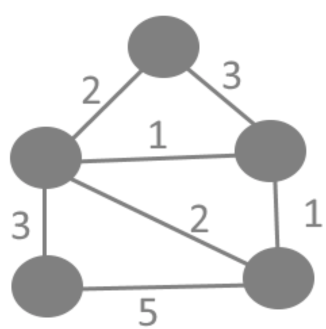


- Road repair: repair only min-cost roads such that all the cities are still connected
- Airline: downsize operations but preserve connectivity

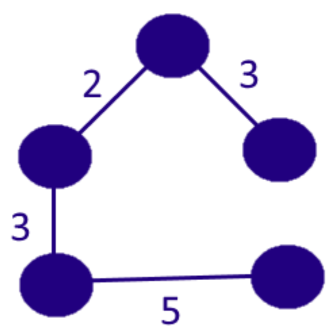
Definition

- A Spanning Tree of a graph G, is a subgraph of G which is a tree and contains all vertices of G

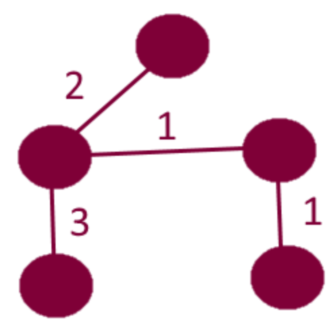
- A Minimum Spanning Tree (MST) of a **weighted** graph G is a spanning tree with the smallest total weight



Graph



Spanning Tree
Cost = 13



Minimum Spanning
Tree, Cost = 7

Problem: compute MST of Graph G

Input: undirected graph G=(V, E) and the weight w_e for each edge

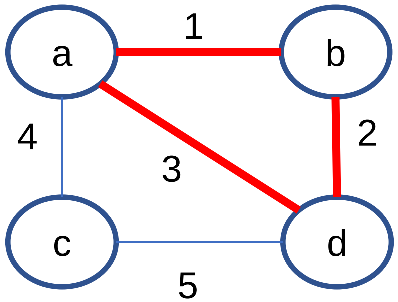
Output: minimum-cost tree $T \in$ E that spans all the vertices V

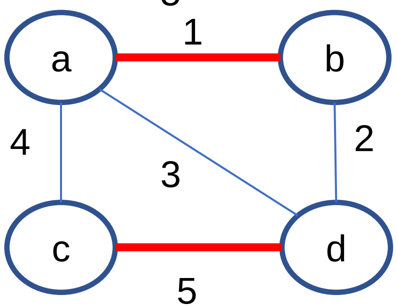
Simplifying assumptions:

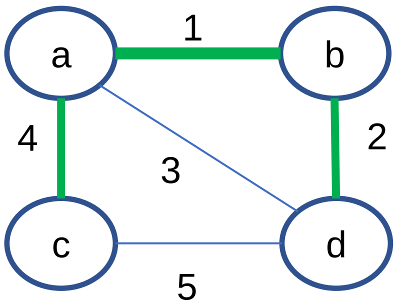
- *G* is undirected and simple (that is, it has no self-loops and no parallel edges)
- Input graph G is connected

Tree means:

- *T* has no cycles
- *T* has exactly n-1 edges
- *T* is connected (for any two nodes u, v, ∃ path u ~>v (and by design v ~> u - undirected graph)







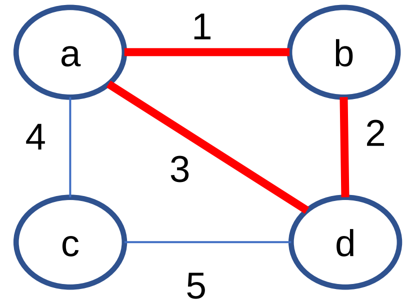
Problem: compute MST of Graph G

Input: undirected graph G=(V, E) and the weight w_e for each edge

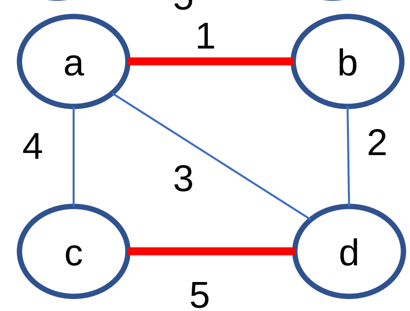
Output: minimum-cost tree $T \in$ E that spans all the vertices V

Tree means:

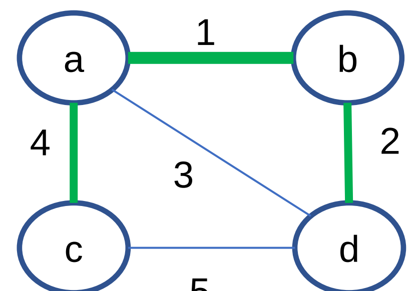
- ❑ *T* has no cycles
- ❑ *T* has exactly n-1 edges
- ❑ *T* is connected (for any two nodes u, v, ∃ path u ~>v (and by design v ~> u - undirected graph)



Not spanning, not tree



Spanning, not connected



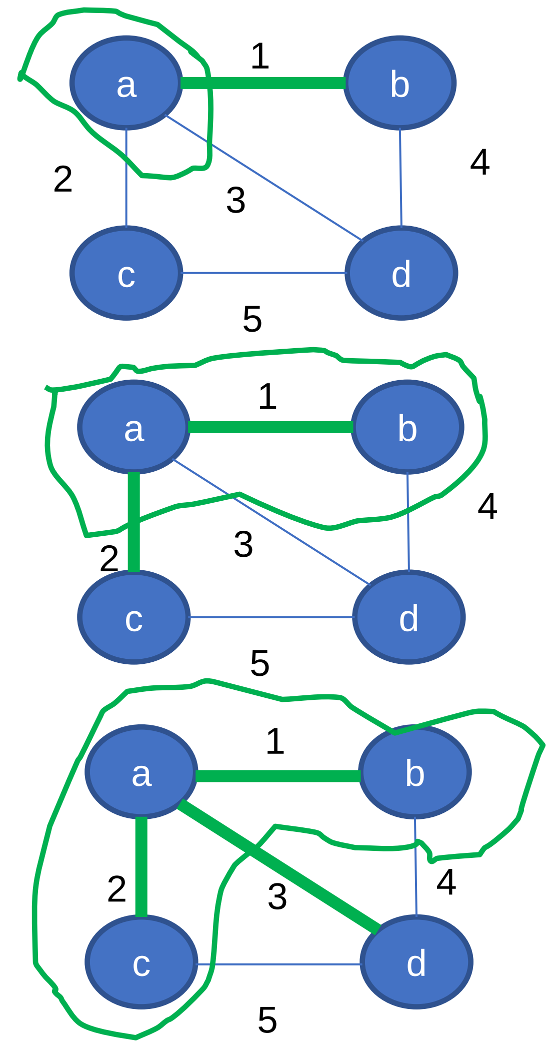
Spanning tree!

Algorithm by Prim (and Jarnik)

Works similar to Dijkstra Shortest Path algorithm.

Grows a tree starting from a single (arbitrarily selected) vertex.

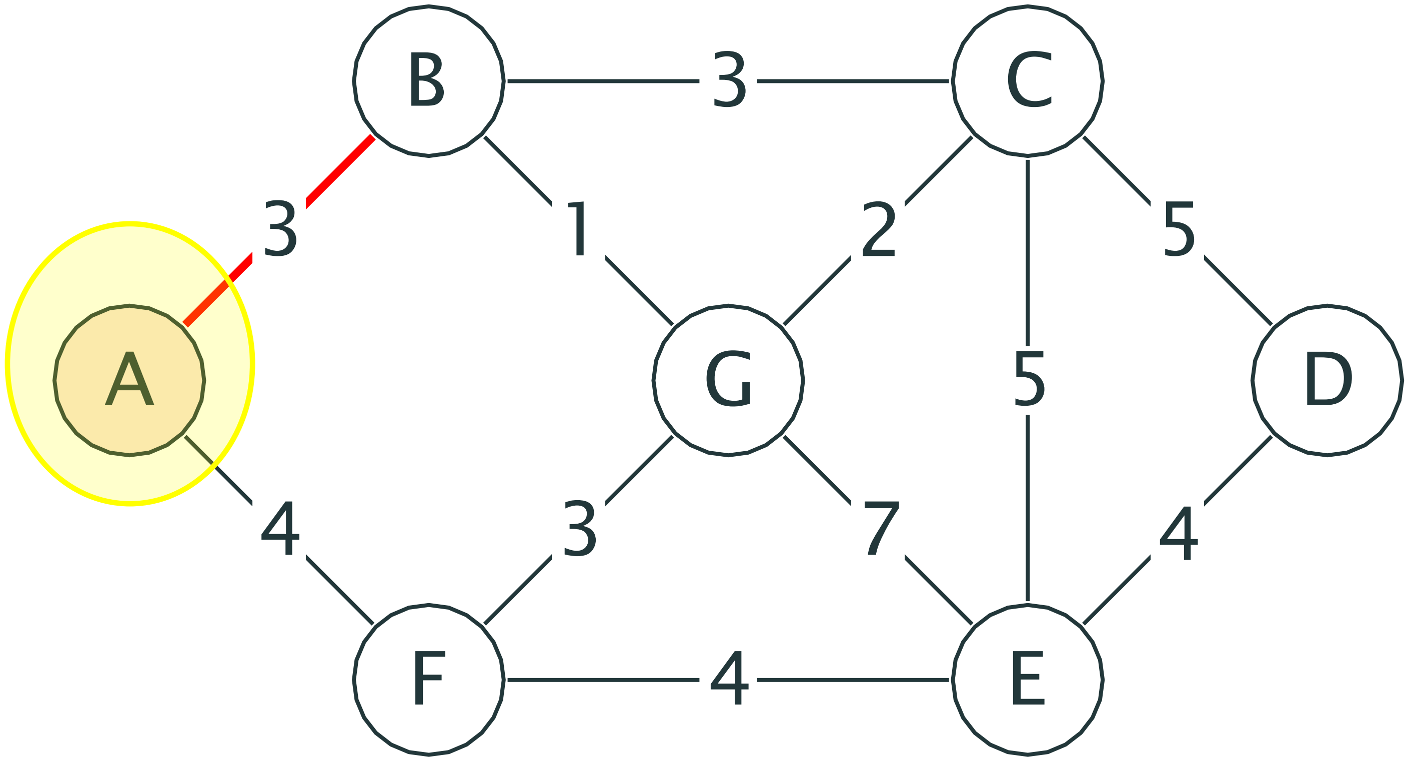
- Start from an arbitrary vertex
- Span another vertex by choosing **the edge with the min cost (greedy move)**
- Now have a tree of 2 vertices
- Check all edges out of this tree and choose the one with min-cost …



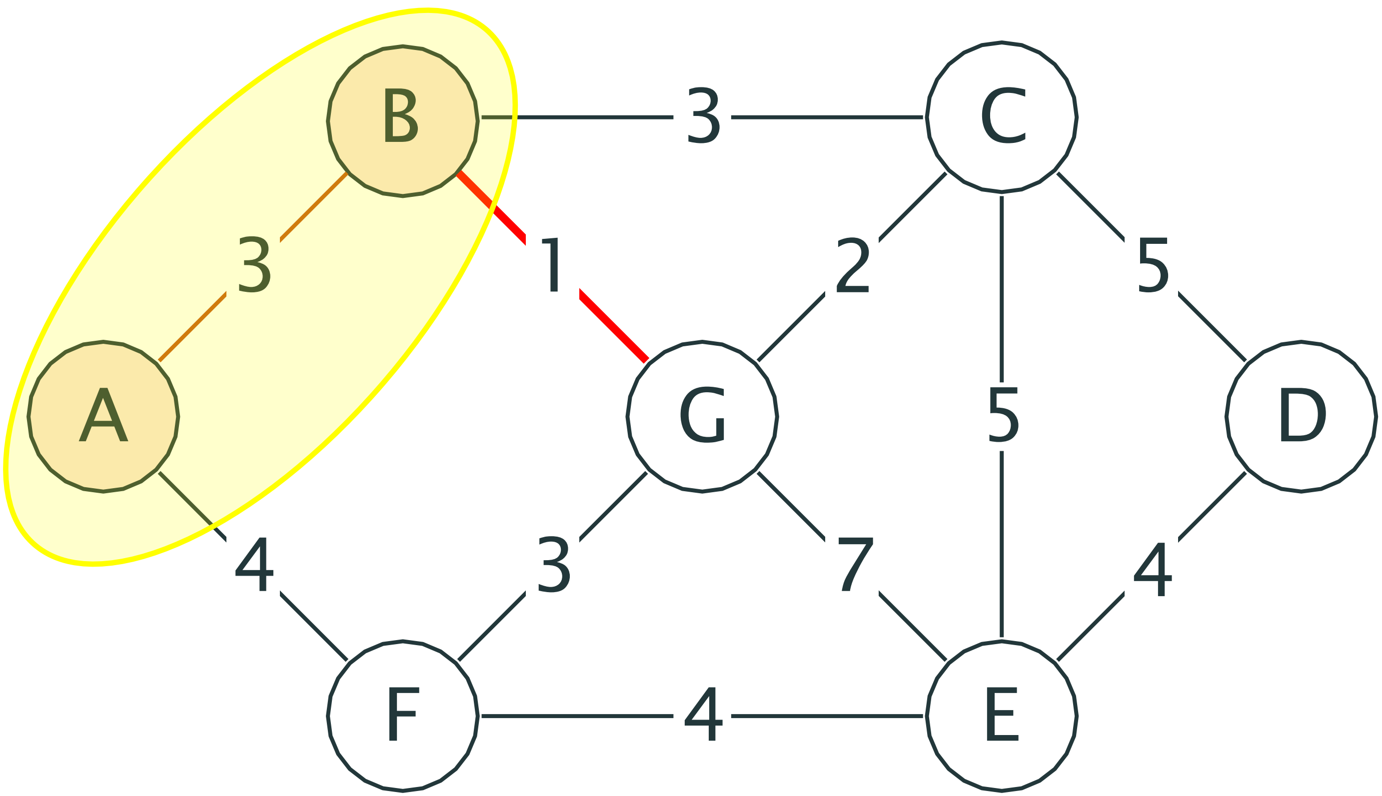
Algorithm Prim_MST (graph G(V,E))

```
initialize tree T: = ∅              # set of tree edges
X: = {vertex s}                     # s ∈ V, chosen arbitrarily
# X contains vertices spanned by the tree-so-far

while |X|!=|V|:
    let e=(u,v) be the cheapest edge of G with u ∈ X and v ∉ X
    add e to T
    add v to X
    # that increases the number of spanned vertices
```

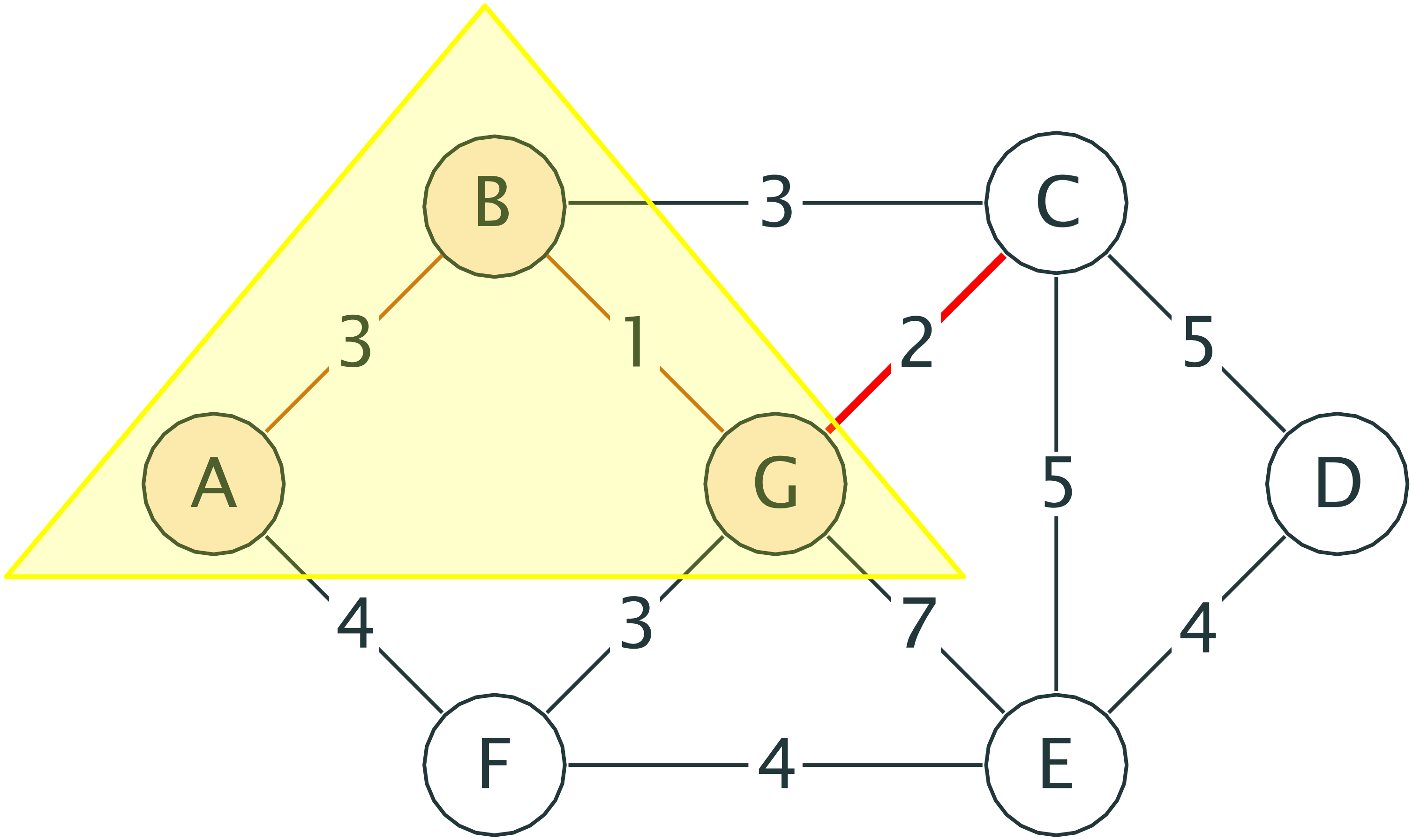
Prim: illustration



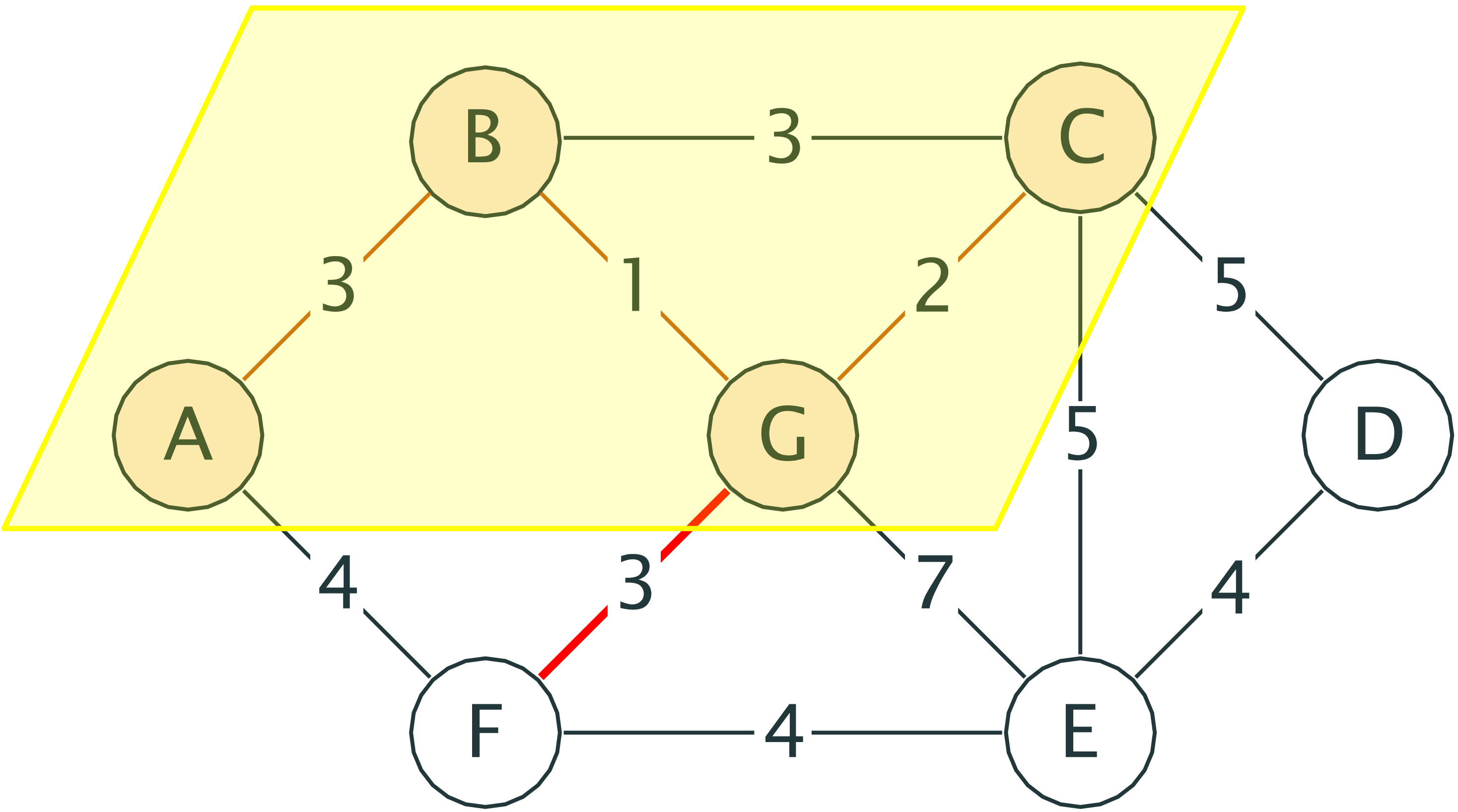
Prim: illustration



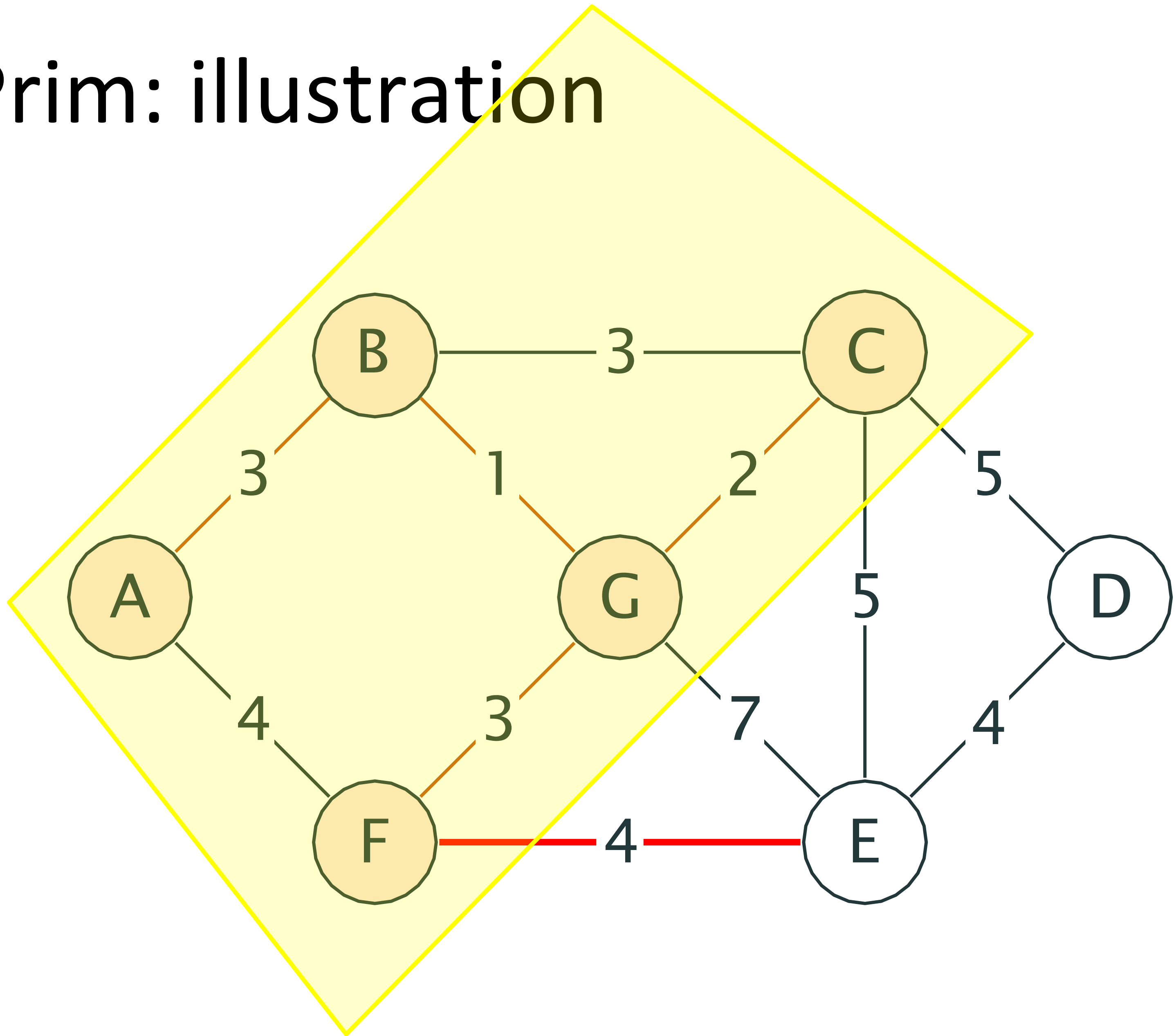
Prim: illustration



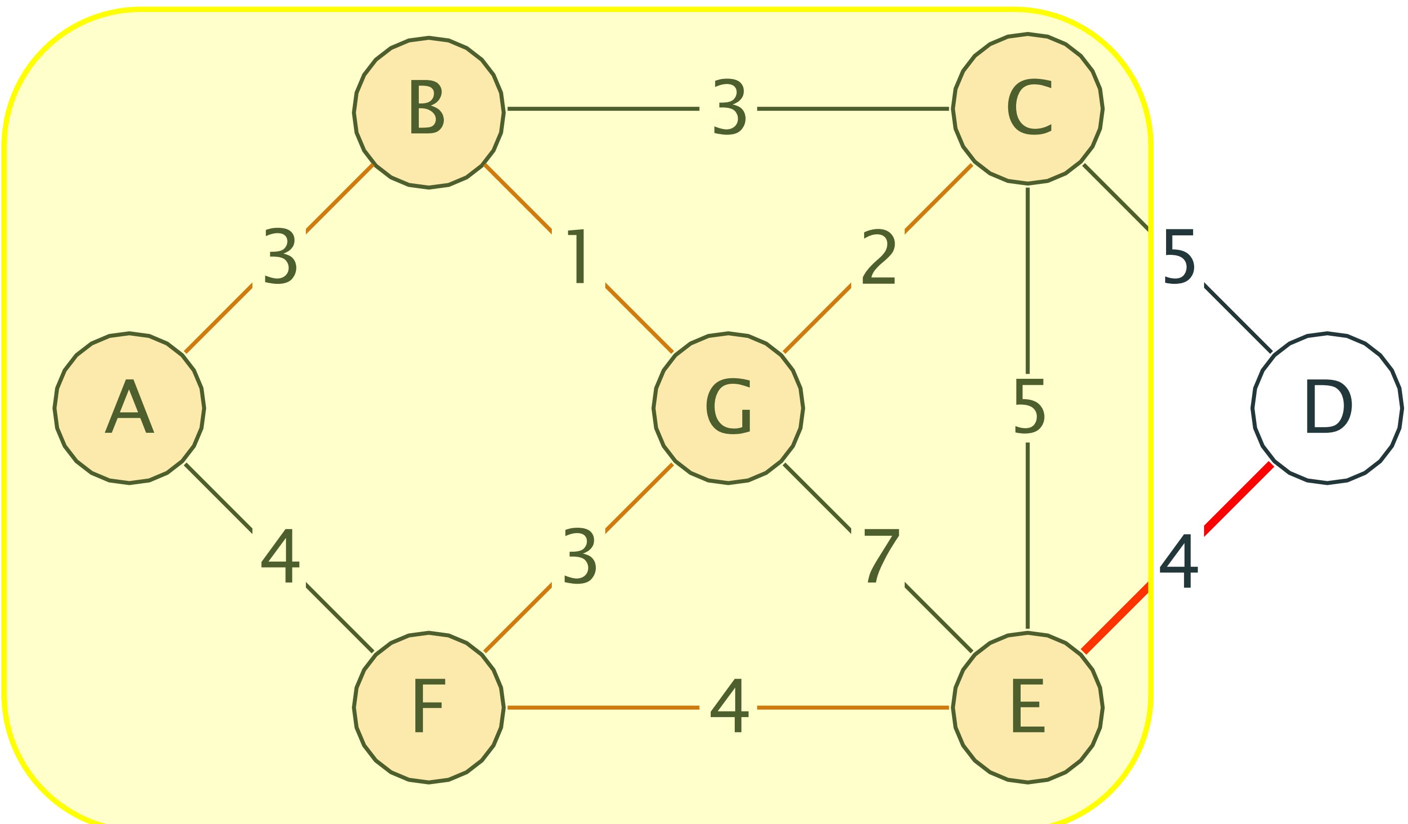
Prim: illustration



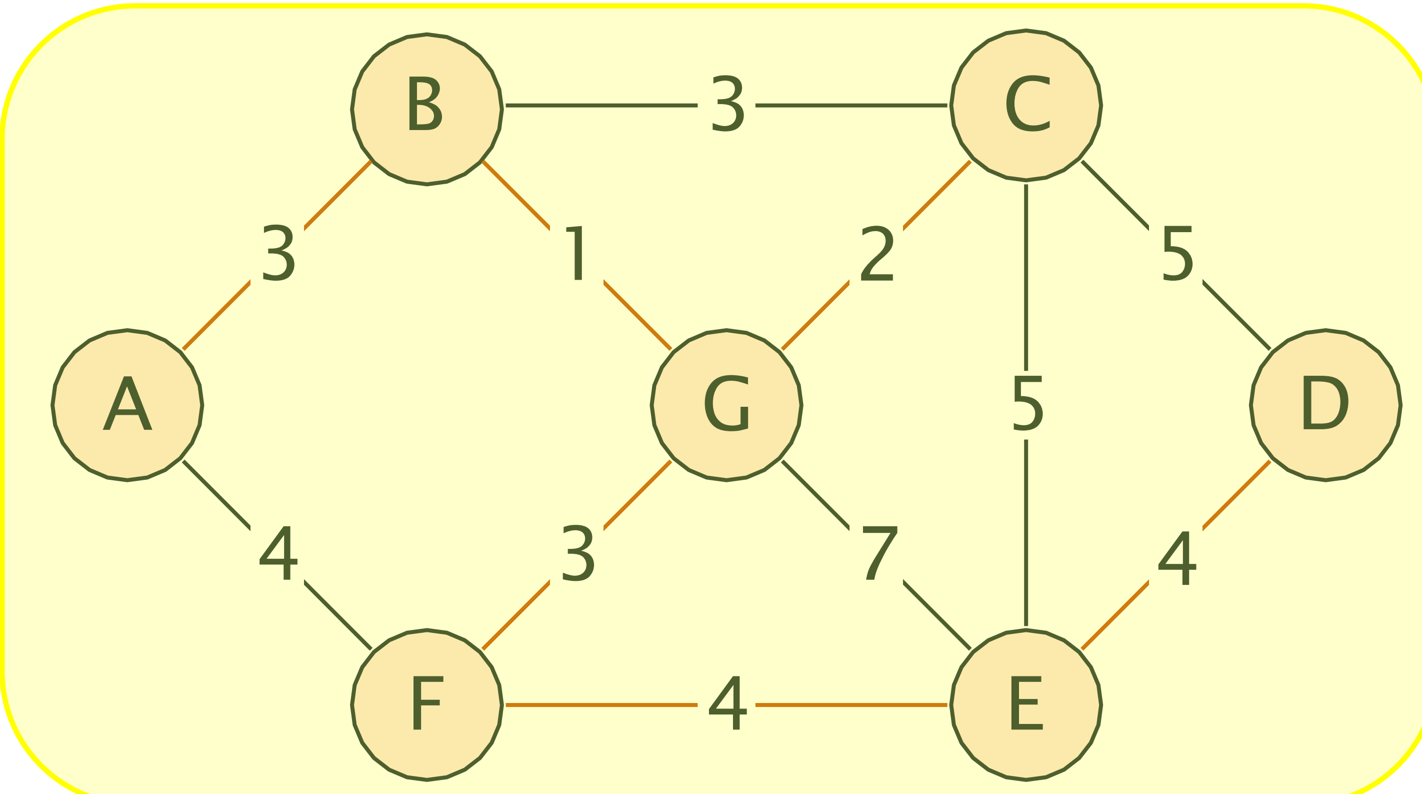
Prim: illustration



Prim: illustration



Prim: illustration



MST cost: 3 + 1 + 2 + 3 + 4 + 4 = 17

Algorithm by Kruskal

Sort all edges by weight (from smaller to larger – ascending)

Add the next smallest edge to the spanning tree, but only if adding it does not create a cycle

Sorted edges:

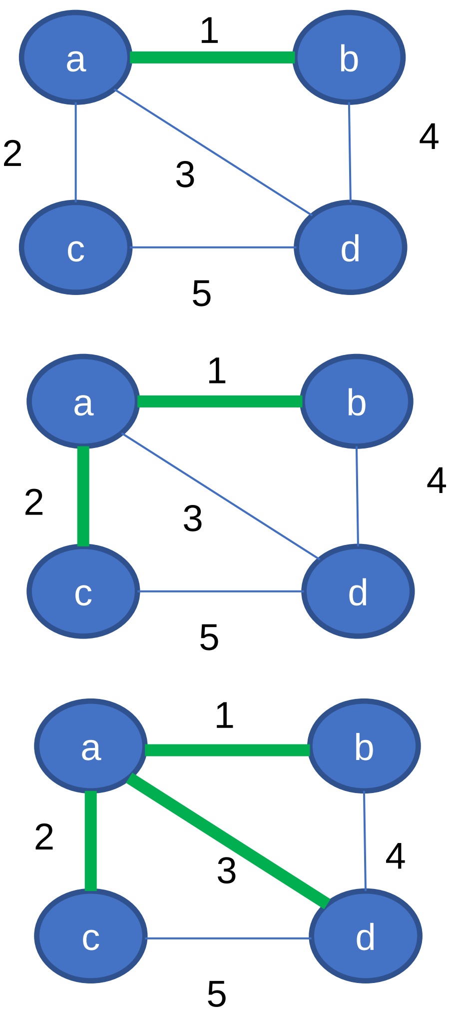
(a,b) ✓

(a,c) ✓

(a,d) ✓

(b,d) 🚫

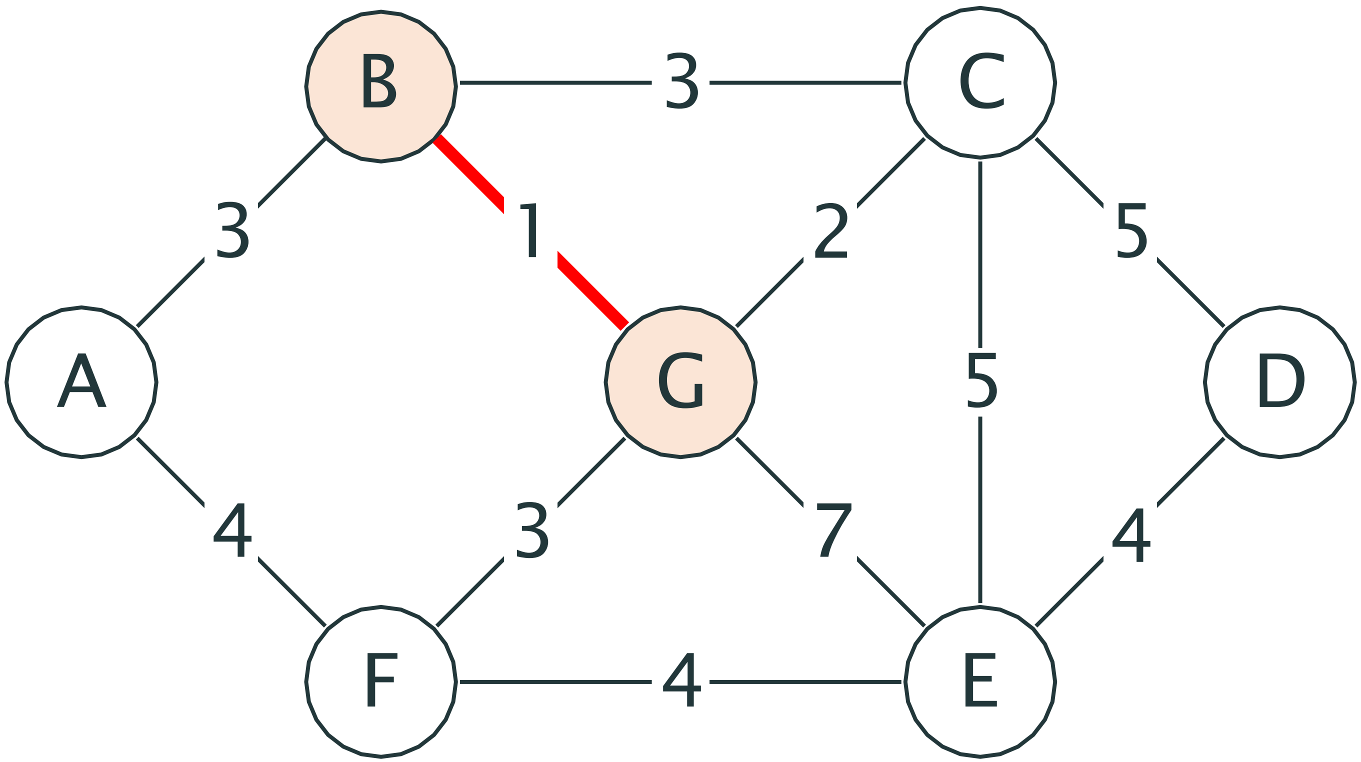
(c,d) 🚫



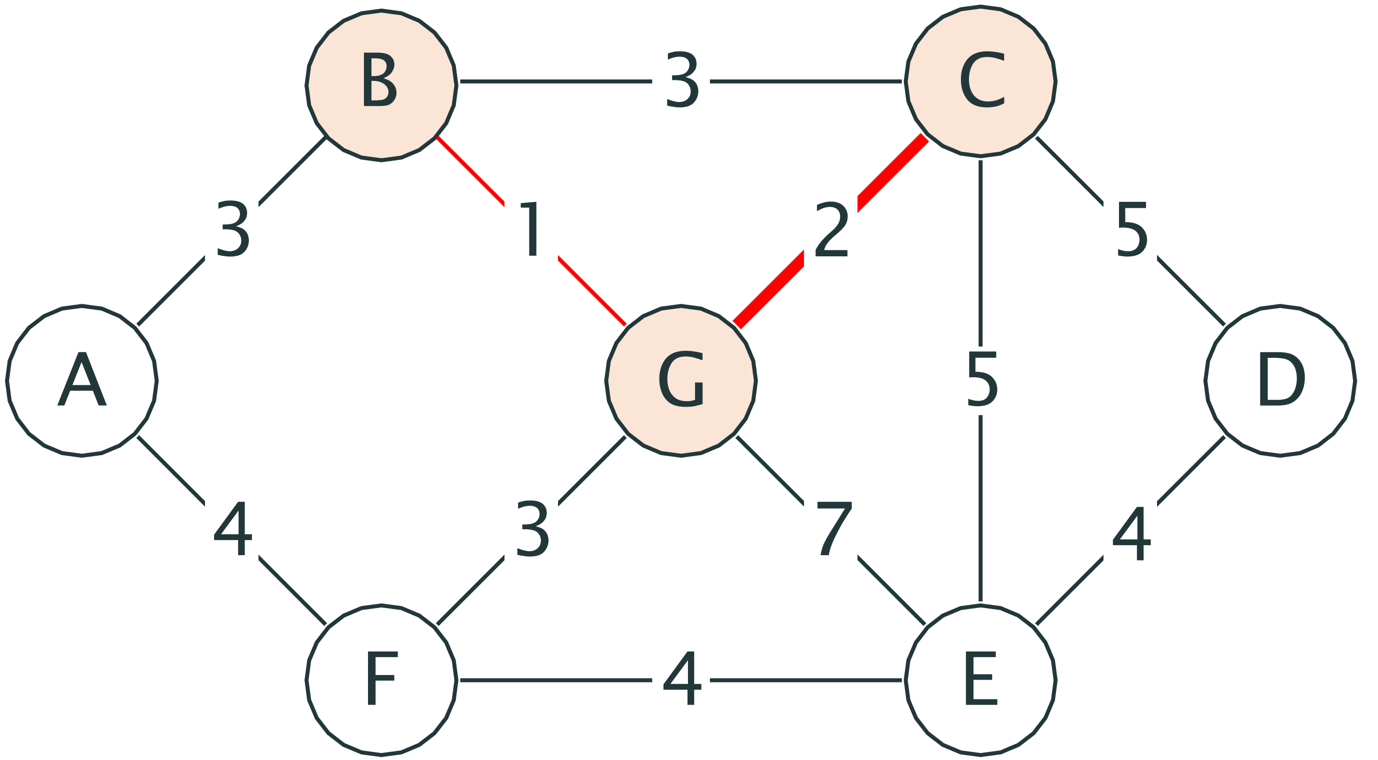
Algorithm Kruskal_MST (graph G(V,E))

```
E’ := edges of G sorted by weights
T : = Ø


for i from 1 to m:
      if T U {E’[i]} has no cycles
            add E’[i] to T


return T
```

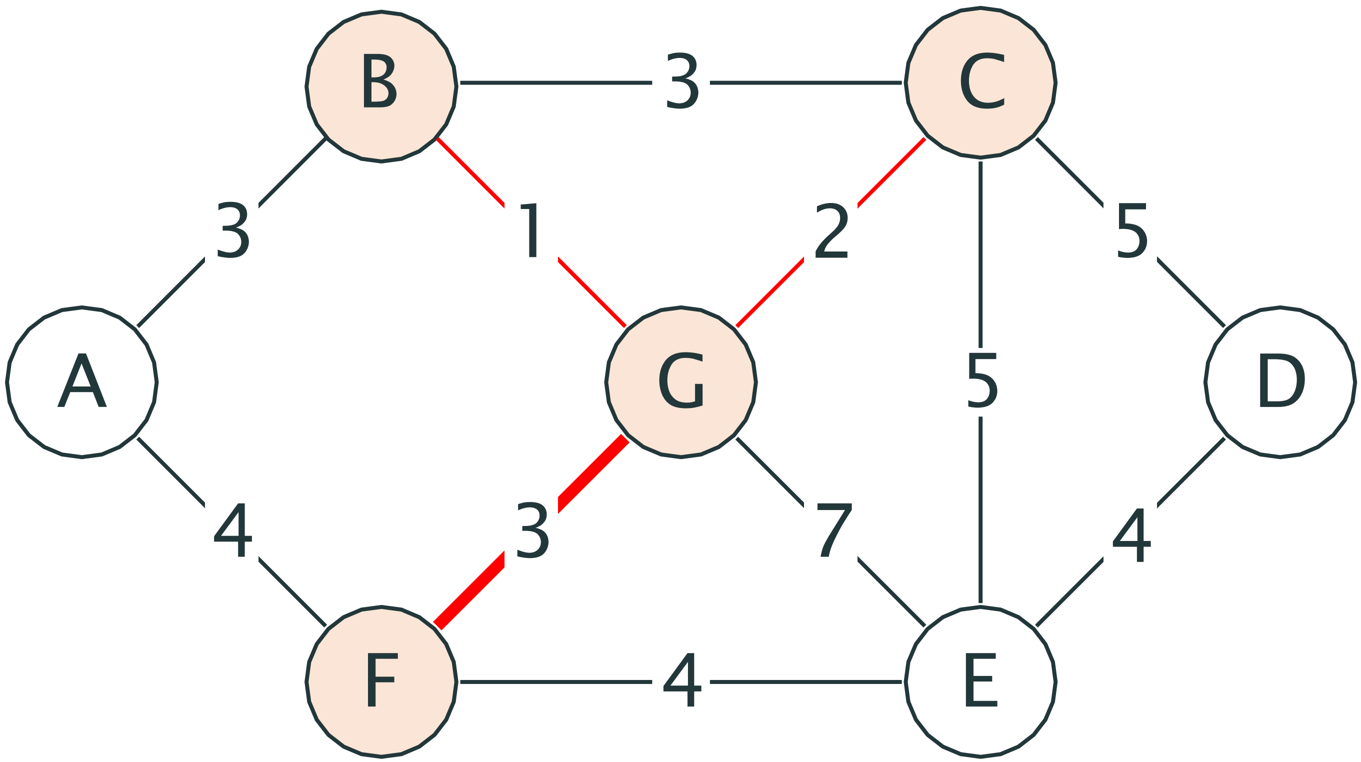
Kruskal illustration



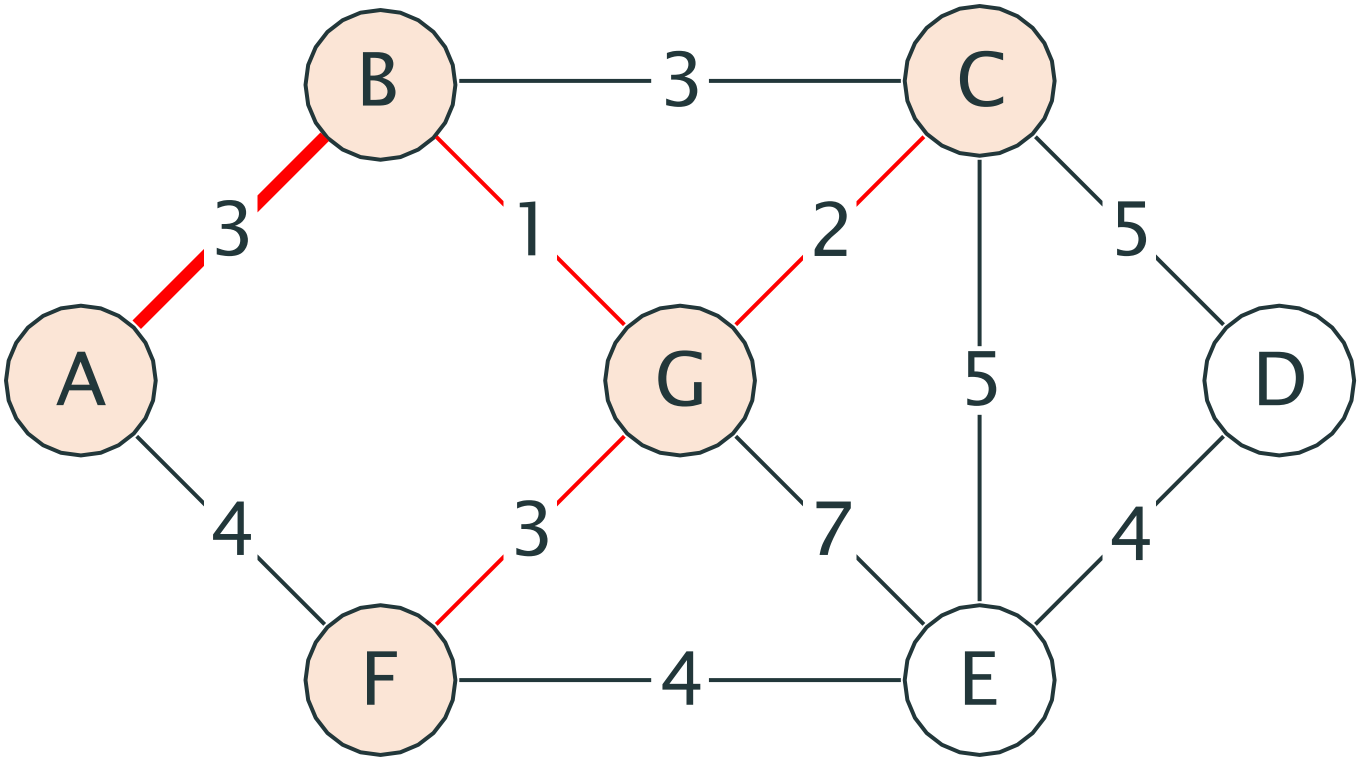
Kruskal illustration



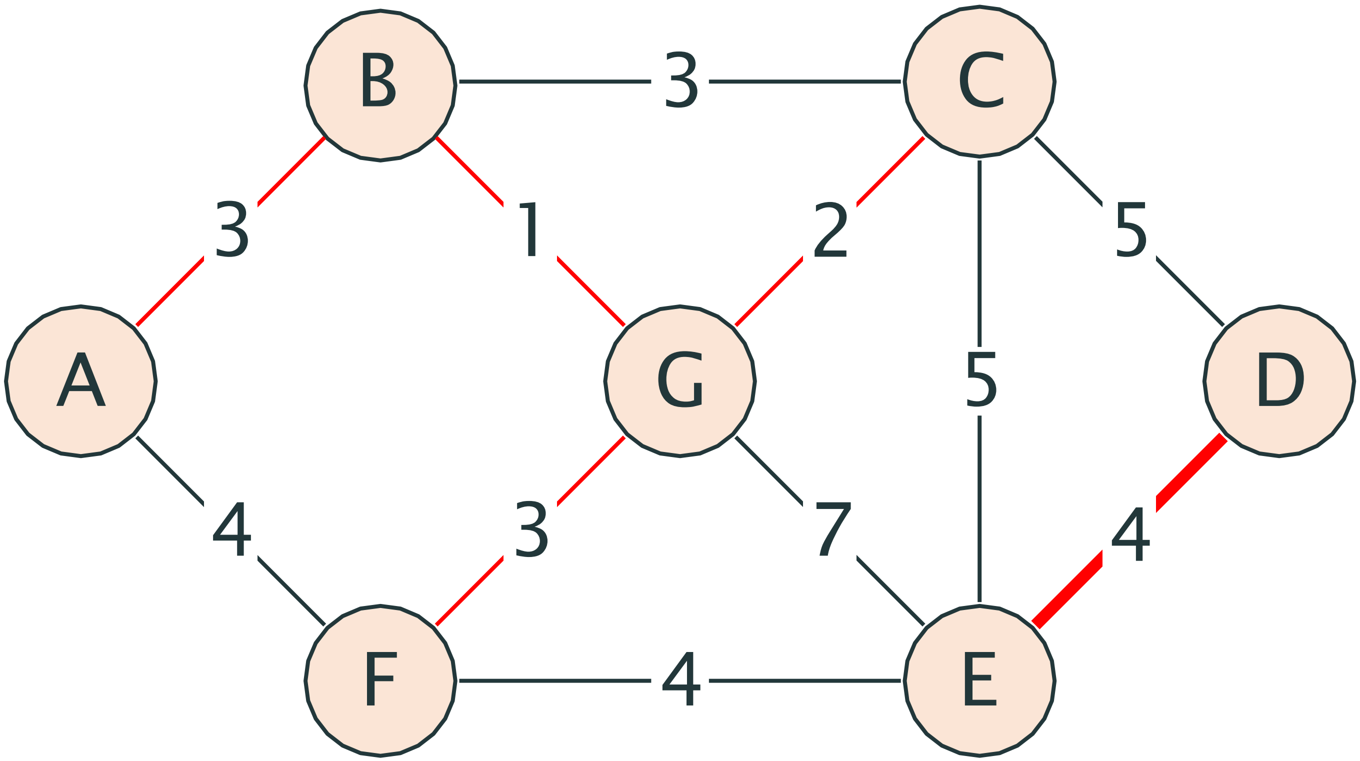
Kruskal illustration



Kruskal illustration

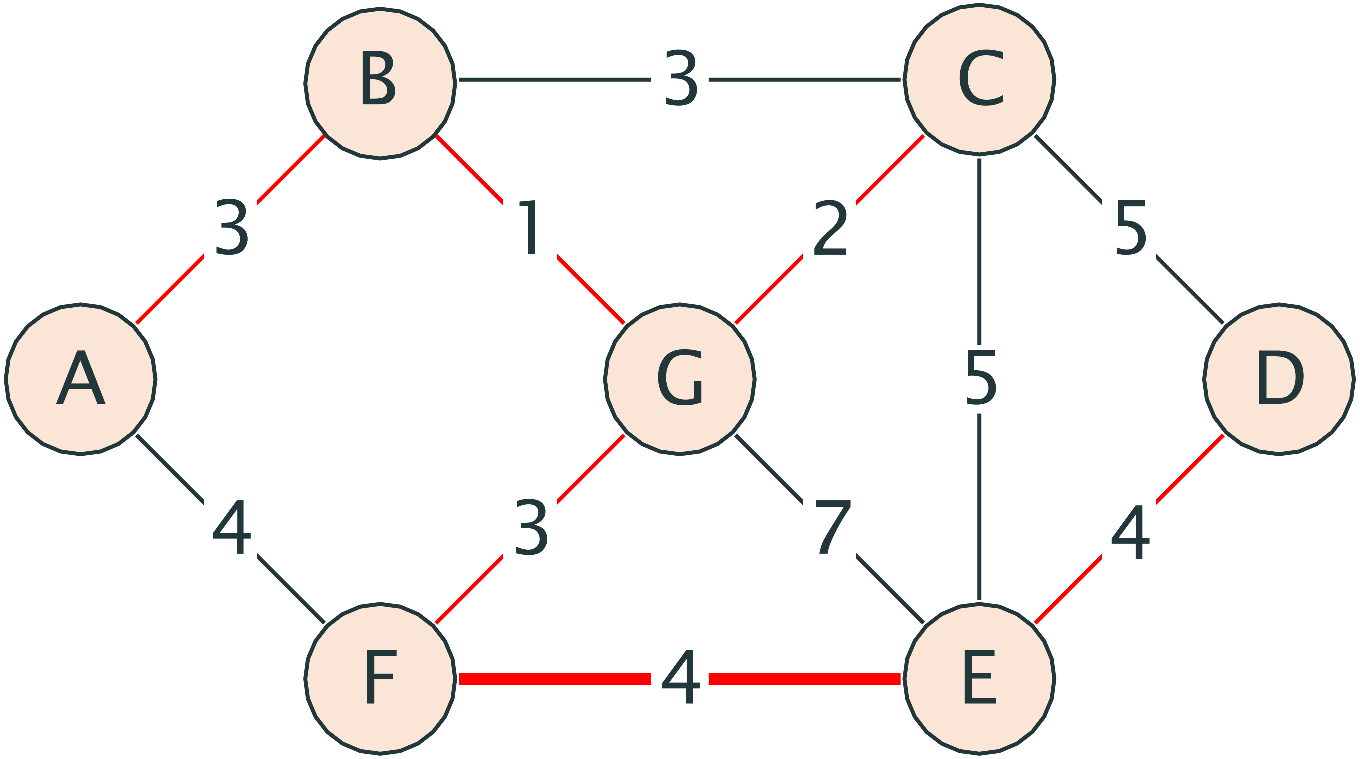


Kruskal illustration



Note that at this point T is not even a spanning tree
(not connected)

Kruskal illustration



MST cost: 1 + 2 + 3 + 3 + 4 + 4 = 17

MST algorithms are greedy

All the algorithms follow some greedy strategy.

Algorithm MST (graph G(V,E))

```
T: = ∅              # collects edges of the future MST


while |T| ≤ |V| - 1:
    select next edge e from E        # some greedy move
    T: = T ∪ e


return T
```

We must prove correctness!

See next lecture