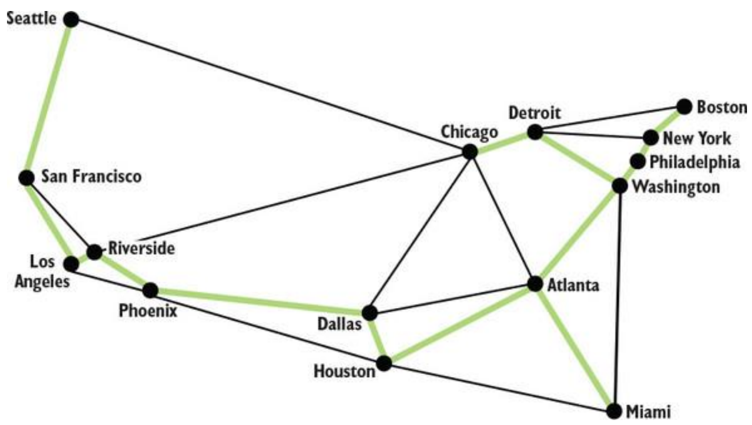
Minimum Spanning Trees

Lecture 05.03

by Marina Barsky

Motivation

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



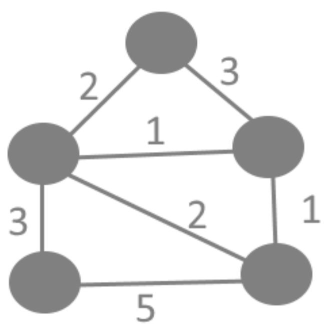
- Road repair: repair min-cost roads such that all the cities are still connected

- Airline: downsize operations but preserve connectivity

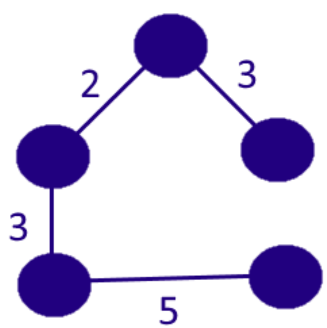
Definition

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

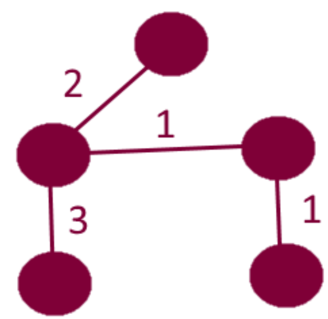
- A Minimum Spanning Tree (MST) of a weighted graph G is a spanning tree with the smallest weight



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

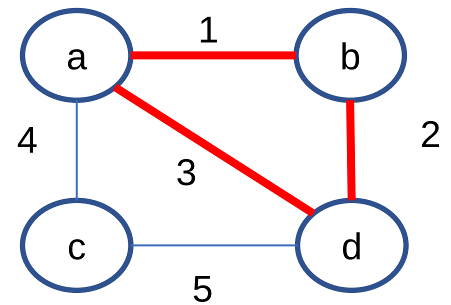
Output: minimum-cost tree $T \in$ E that spans all the vertices V

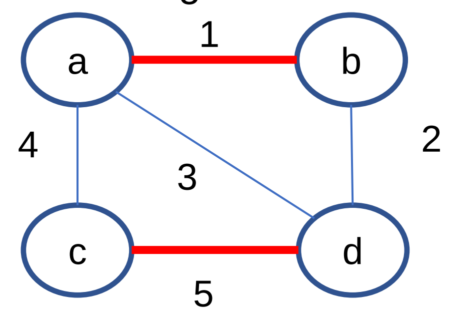
Assumptions:

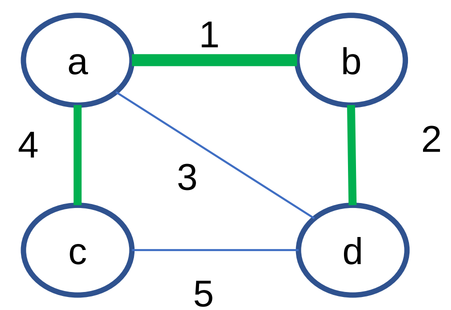
- *Input graph G is connected*

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 v ~> u, undirected graph)





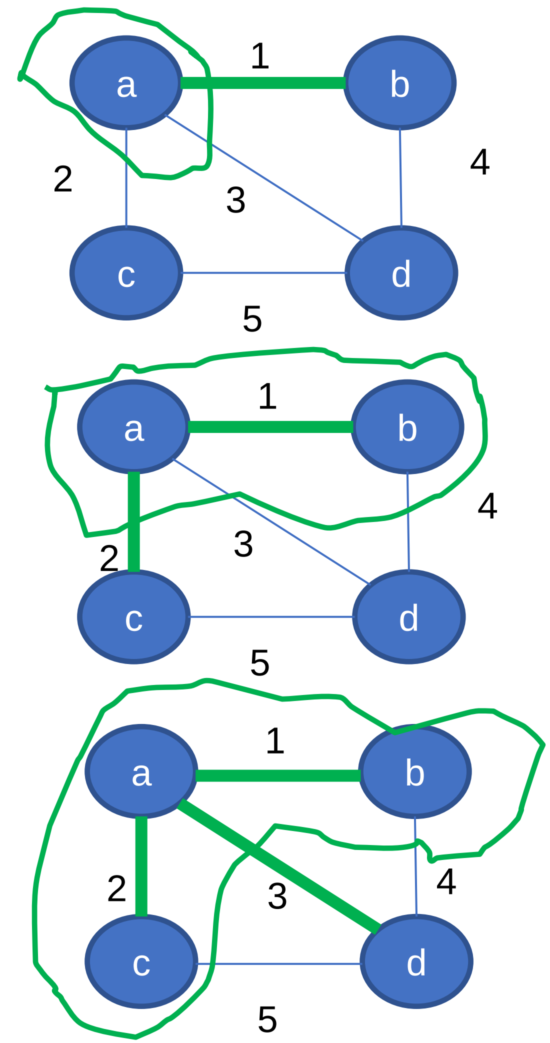


Algorithm by Prim (and Jarnik)

Works similar to Dijkstra Shortest Path algorithm

Grows a tree from a single vertex

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

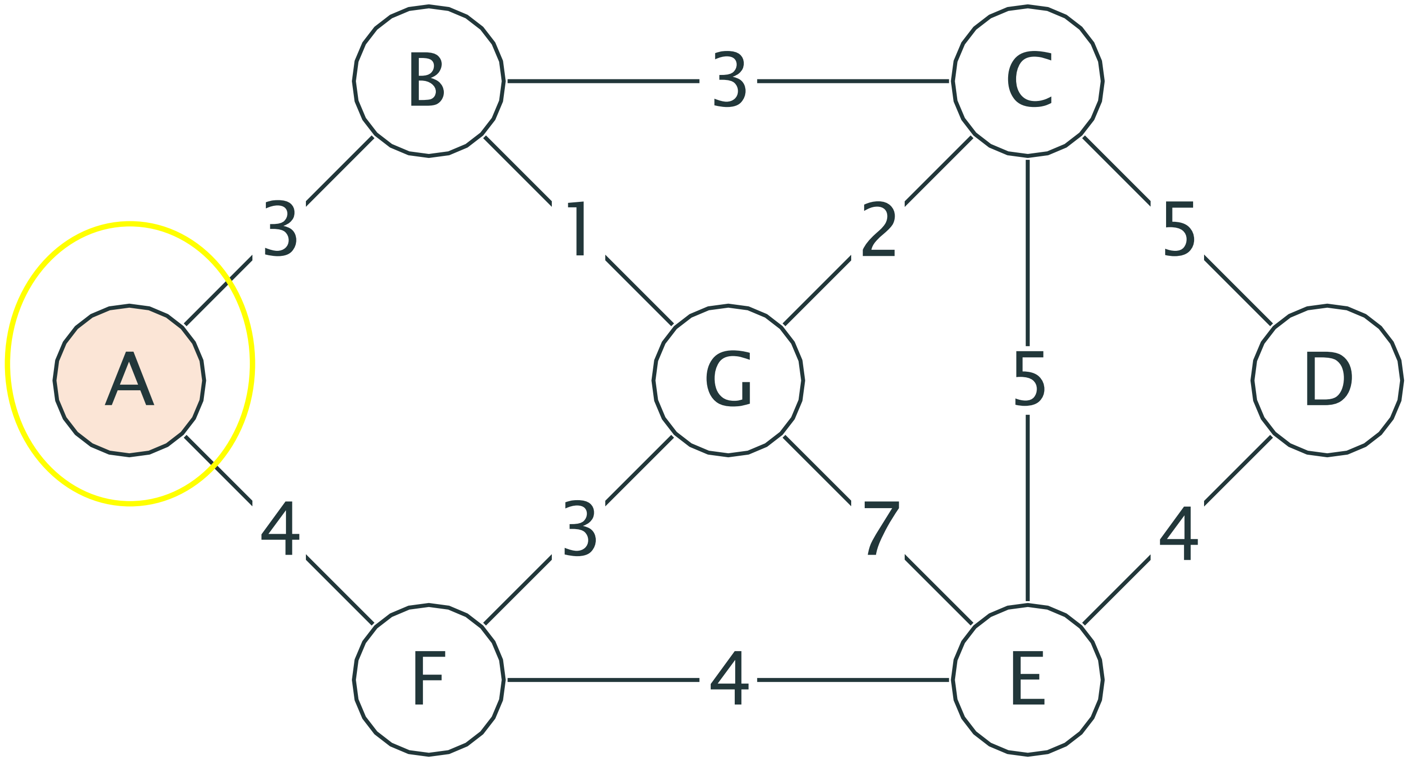


Algorithm Prim_MST (graph G(V,E))

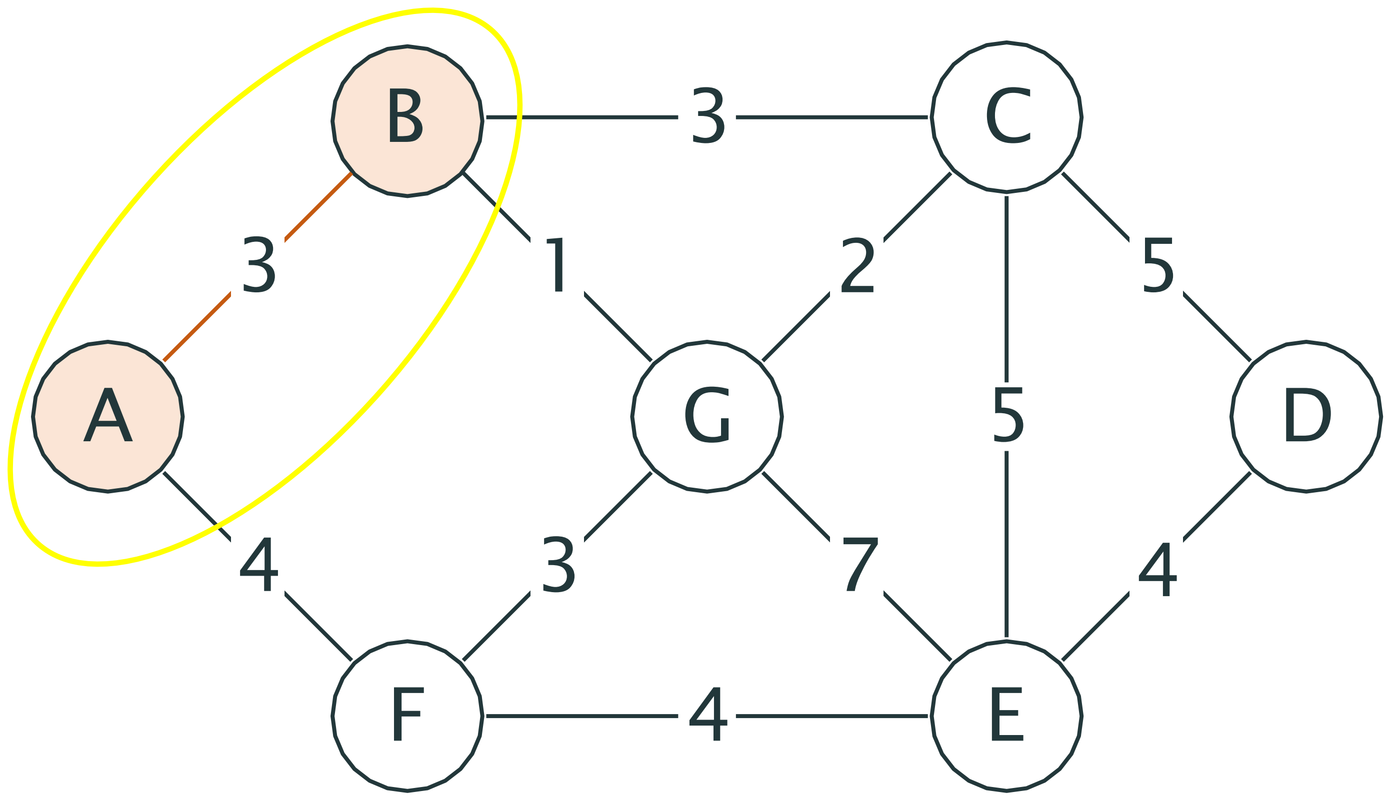
```
initialize tree T: = Ø              # set of tree edges
X: = {vertex s}                     # s ∈ V, chosen arbitrarily
# X contains vertices spanned by the tree-so-far

while |X|!=|V|:
    let e=(u,v) 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
```

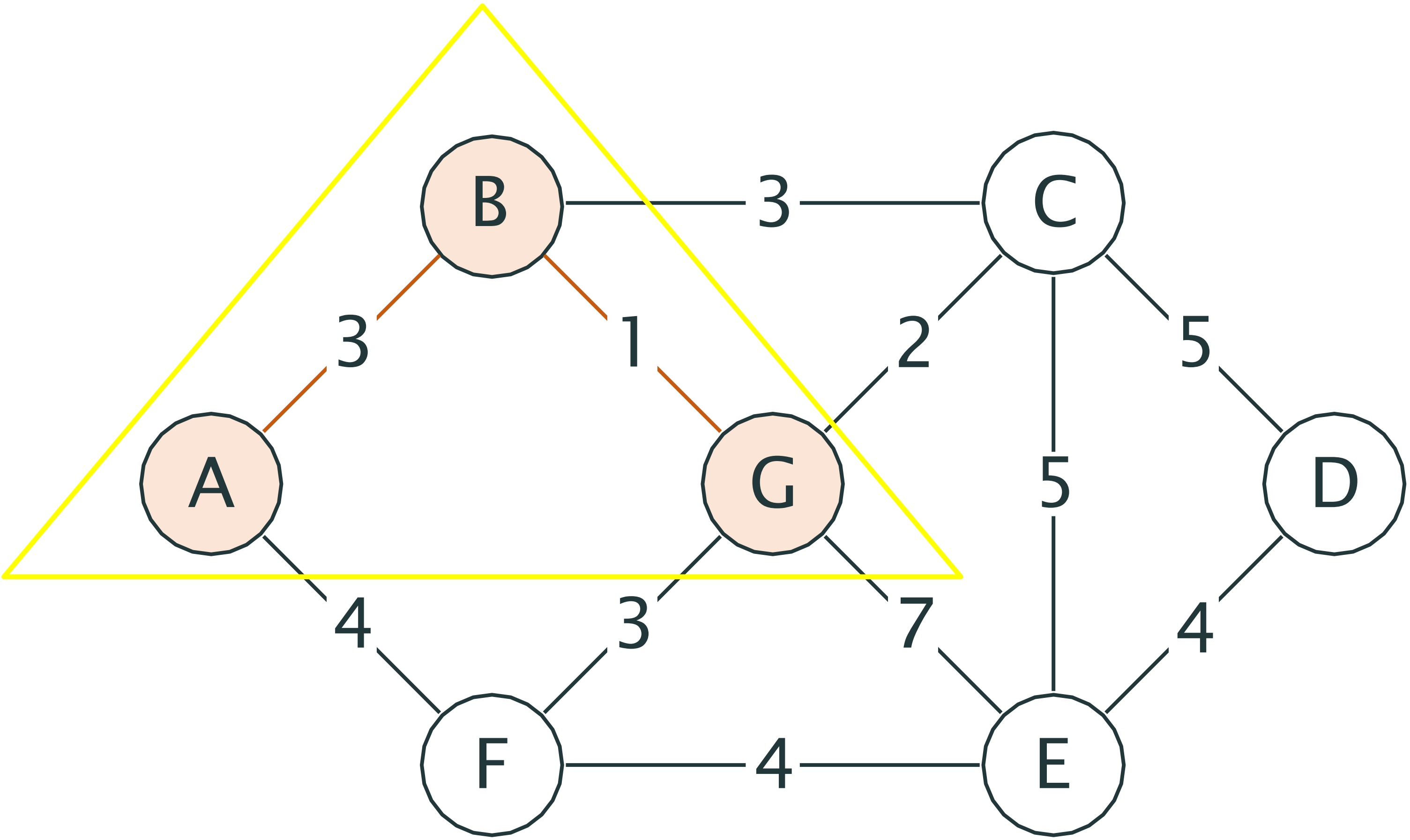
Prim: illustration



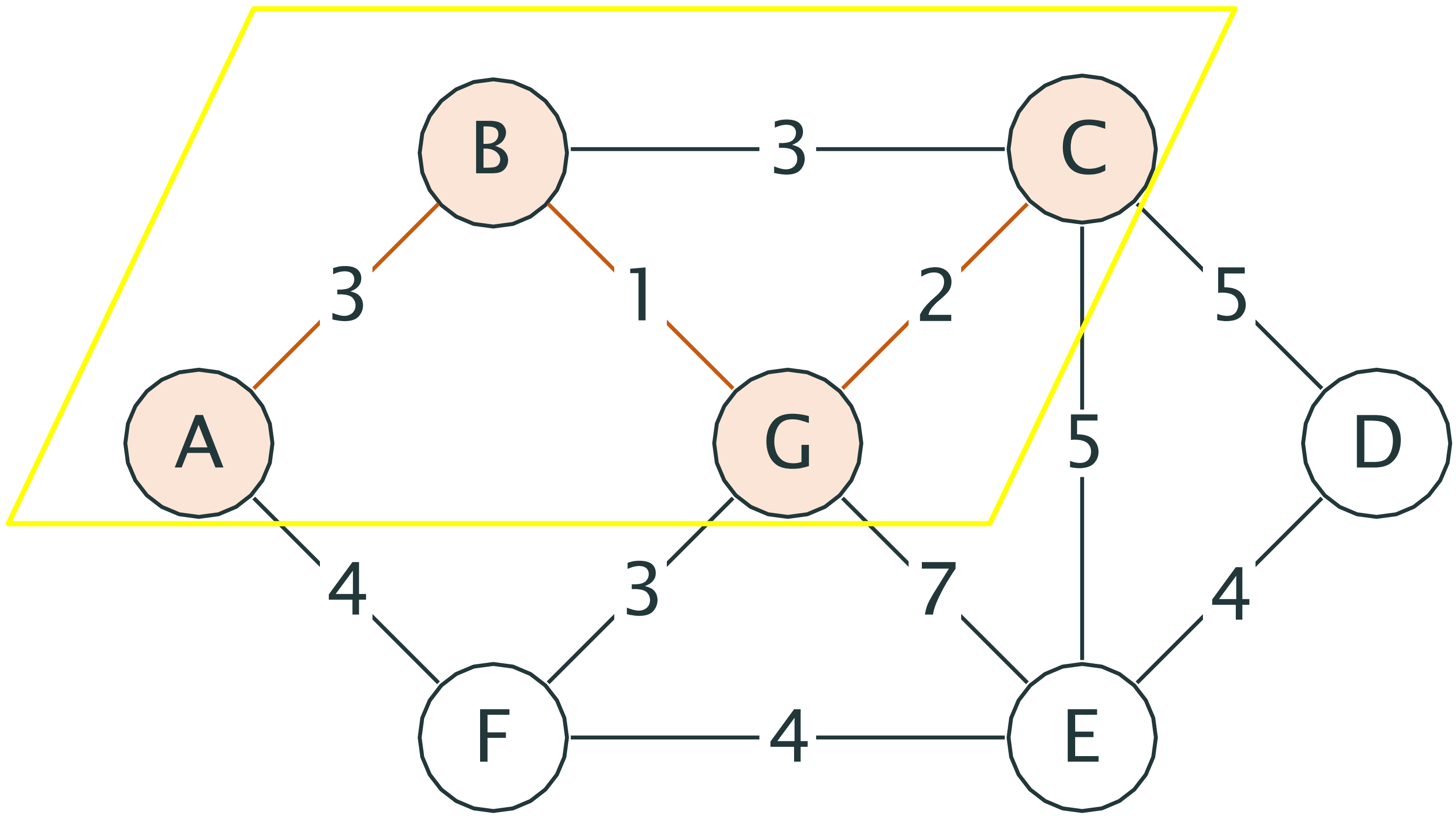
Prim: illustration



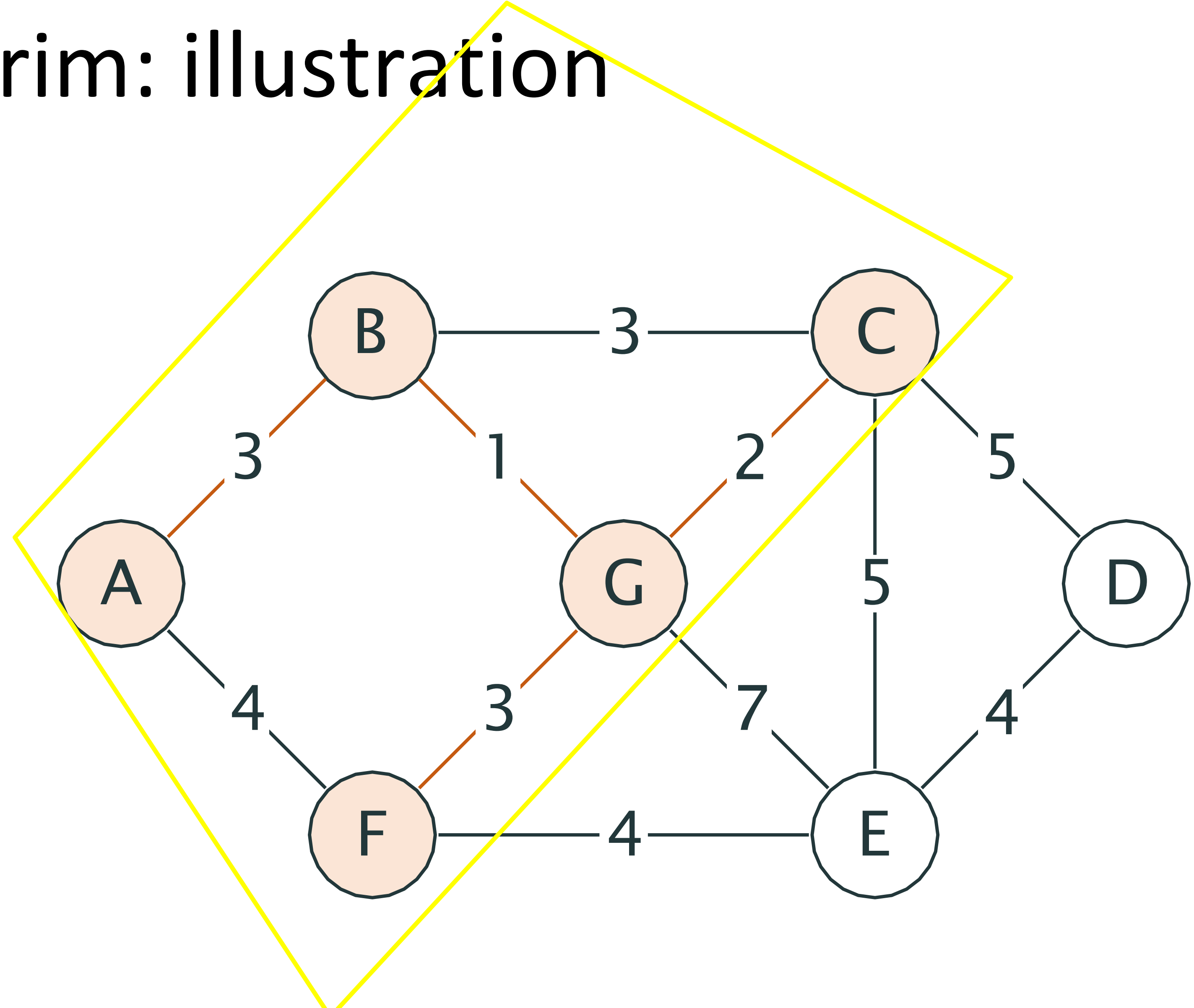
Prim: illustration



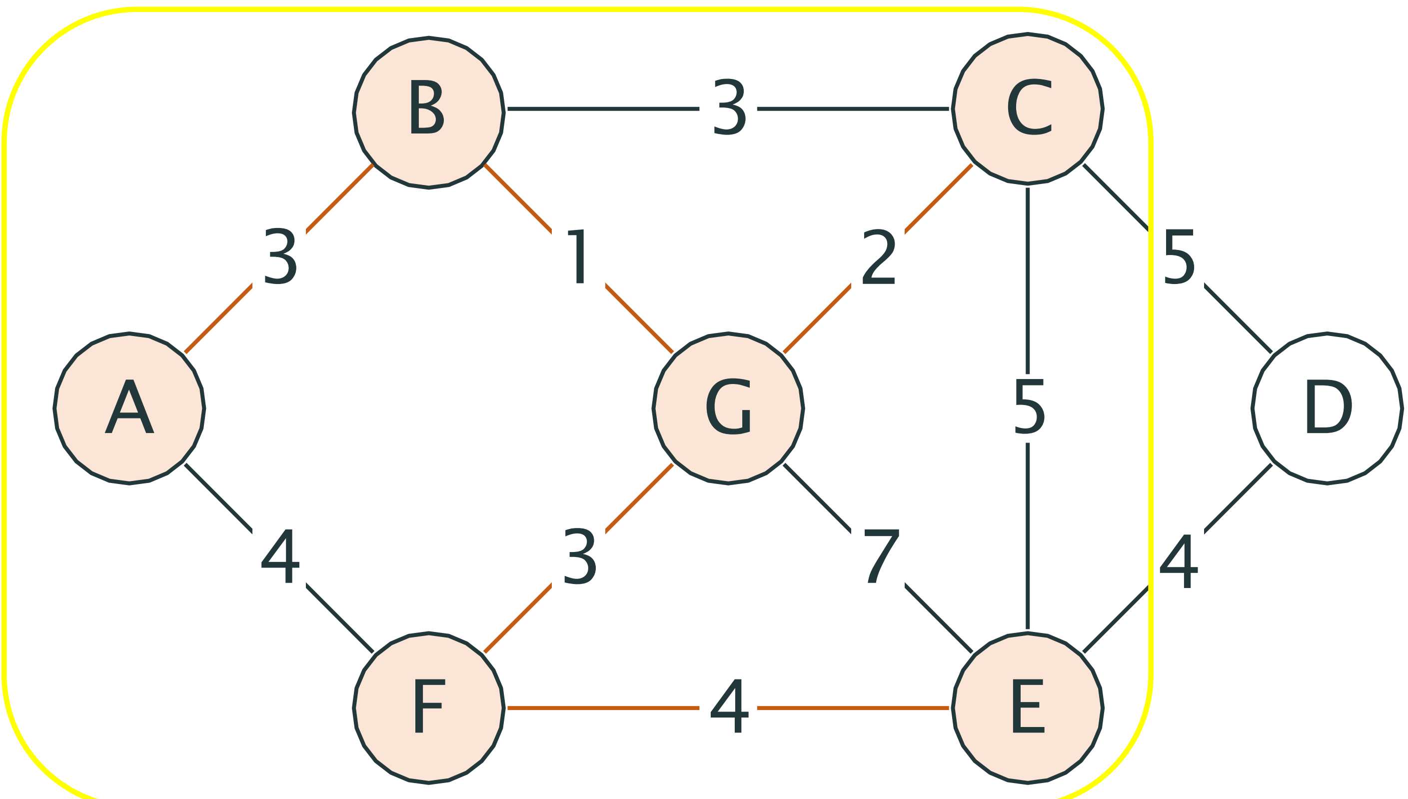
Prim: illustration



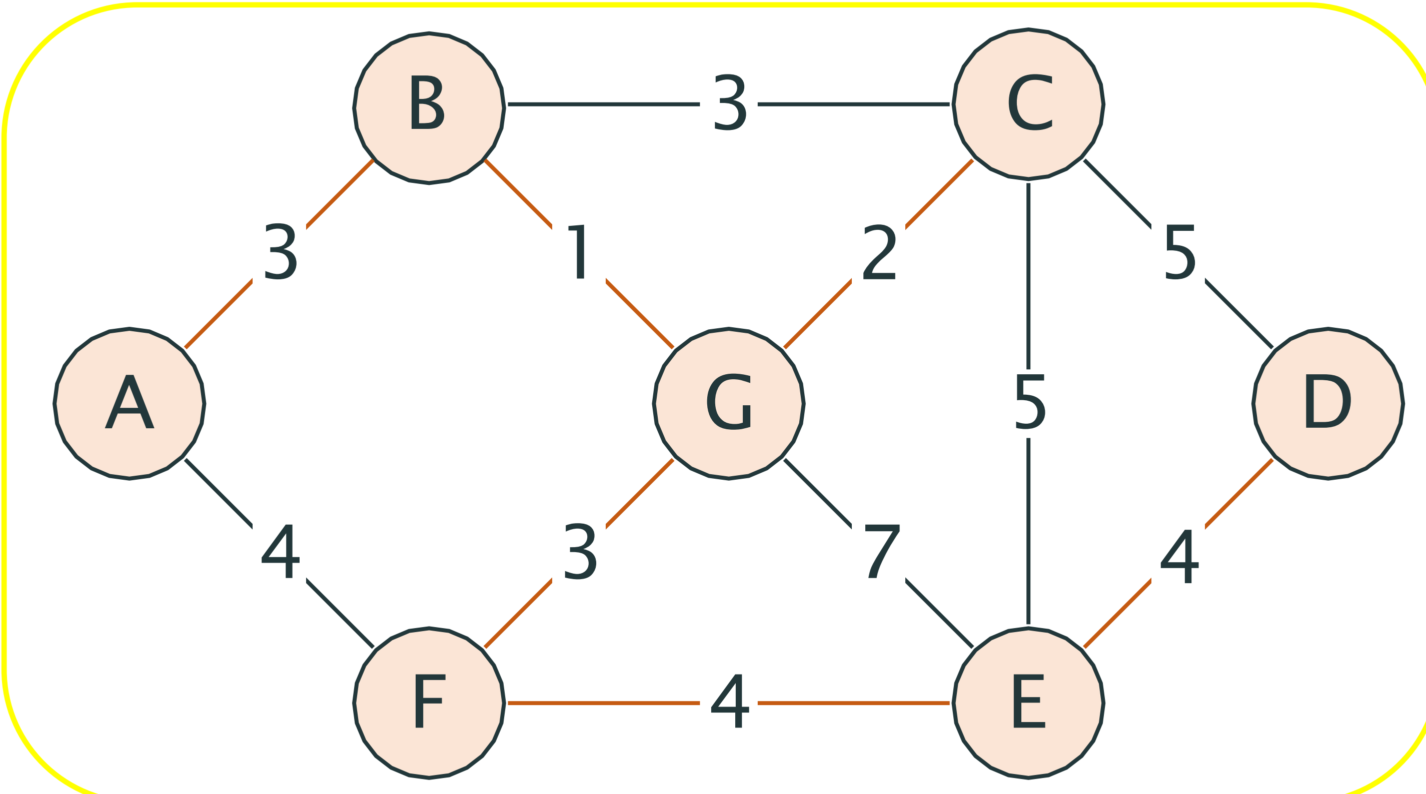
Prim: illustration



Prim: illustration



Prim: illustration



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

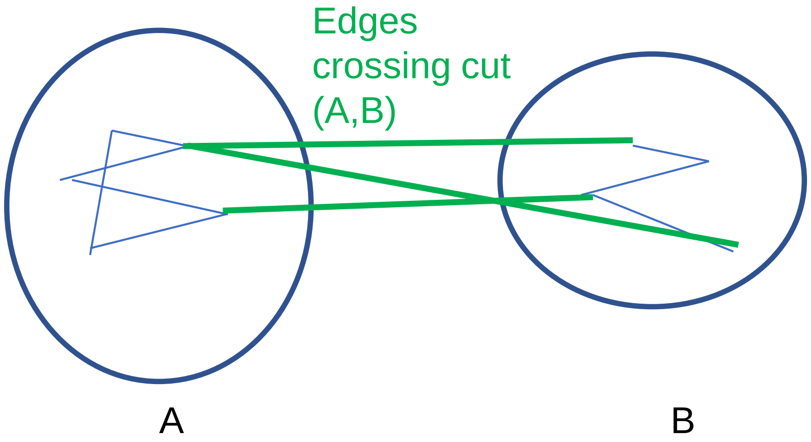
Correctness

We need to prove:

1. Prim computes a spanning tree T*
2. T* is a MST

Cuts

- A *cut* is a partition (A, B) of G into 2 non-empty subsets (proper subsets)
- How many different cuts can be in a G with n vertices? (n, n^2, 2^n)?



Empty Crossing Lemma

A graph is not connected ⇔ exists cut (A,B) with no crossing edges

Proof

<=

- Assume RHS. Pick any u ∈ A and v ∈ B. Since no edges cross the cut, there is no path from u to v => G is not connected

=>

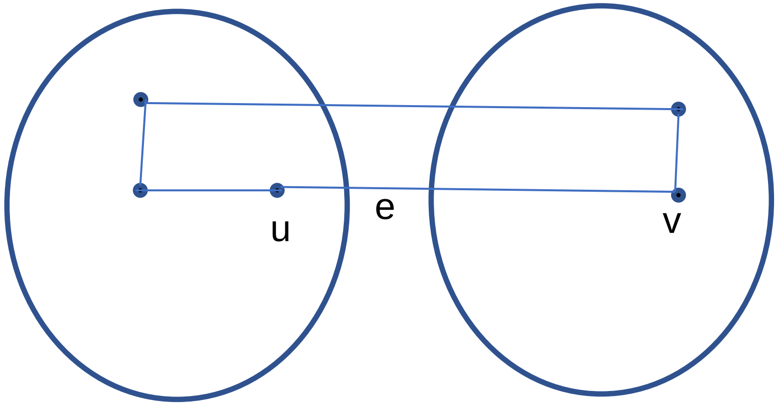
- Assume LHS. Define A = {all vertices reachable from u}, u's connected component, and B={all the remaining vertices in G}. No edges cross from A to B, otherwise A would absorb B into a single connected component

Cycle Crossing Lemma

If two vertices u and v are part of a cycle, then for a cut (A,B) such that u ∈ A and v ∈ B there must be at least two crossing edges.

Proof

If there is a path from u to v that includes one crossing edge e, then it must also be a different path from v to u to close the cycle. The only way to reach u from v is through crossing edges, thus we need at least one more crossing edge to return to A.



Lonely Edge Corollary

If there is a single crossing edge e in cut (A,B), e is not a part of any cycle

Theorem 1. Prim outputs a Spanning Tree

Proof

- Consider a cut of G into (X, V-X) at some step of the algorithm.
- Next, we add a single crossing edge to T, and the vertex on the opposite end of this edge gets added to X.
- By the Lonely Edge Corollary addition of the first crossing edge does not create a cycle in T, and we never explore other crossing edges for this cut again.
- Thus the produced T is acyclic. (1)
- T is also connected. According to the Empty Crossing Lemma there must be at least one edge from X to V-X, if G is connected. (2)
- Hence the algorithm adds n-1 edges and spans all n nodes of G: (1) acyclic, (2) connected graph with all n vertices and n-1 edges is a Spanning Tree

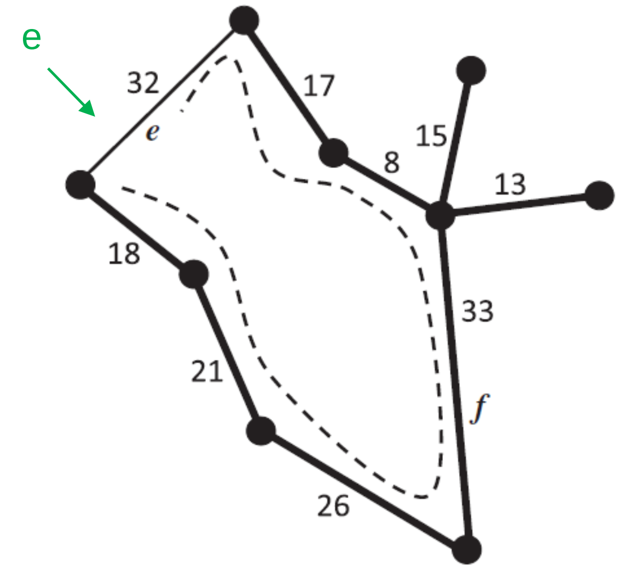
The MST Cycle Lemma

Let G be a weighted connected graph, and let T be a minimum spanning tree for G.

If e is an edge of G that is not in T, then if we add e to the tree this will create a cycle.

Proof

Because any spanning tree already contains a unique path between any pair of vertices, if we add one more edge this will create an alternative path – a cycle.



MST

The Non-MST Edge Cost Lemma

Let G be a weighted connected graph, and let T be a minimum spanning tree for G. If e is an edge of G that is not in T, the weight of e >= the weight of any edge in the cycle created by adding e to T.

Proof

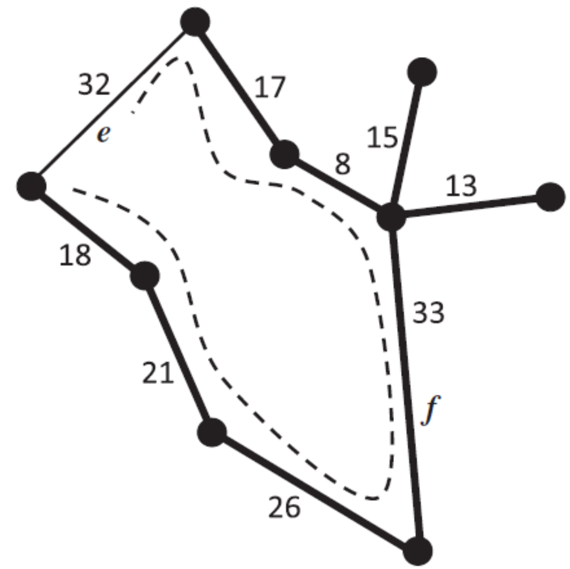
If e is not in T, then adding e to T creates a cycle, C. Suppose, for the sake of contradiction, that there is an edge f on this cycle, whose weight is: w(f) > w(e).

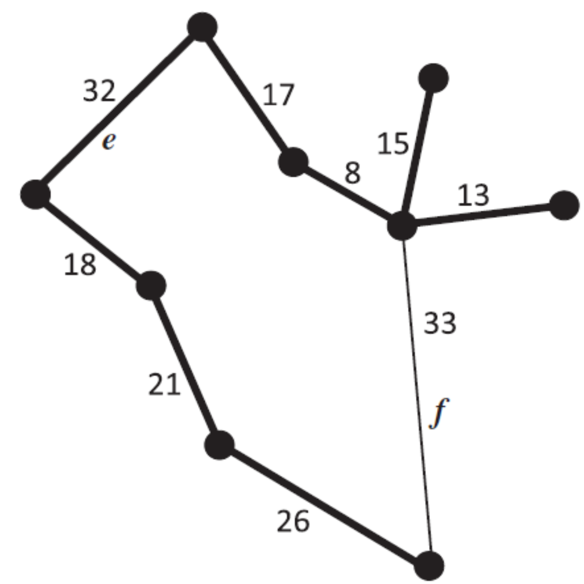
Then we can remove f from T and replace it with e, and this will result in a spanning tree, T', whose total weight is less than the total weight of T.

But the existence of such a better tree, T', would contradict the fact that T is a minimum spanning tree. So no such edge, f, can exist.

Example

- Any nontree edge must have weight that is ≥ every edge in the cycle created by that edge and a minimum spanning tree.
- Suppose edge e has weight 32 and edge f in the same cycle has weight 33. Edge f is a part of MST (shown with bold edges), and edge e is not.
- But then we could replace f by e and get a spanning tree with lower total weight, which would contradict the fact that we started with a minimum spanning tree.



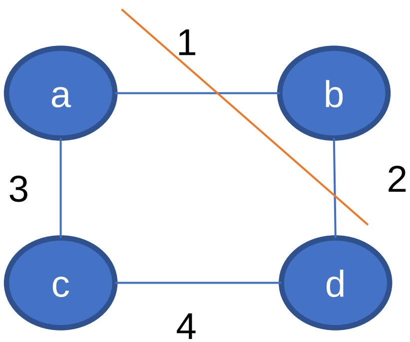


Cut Crossing Theorem

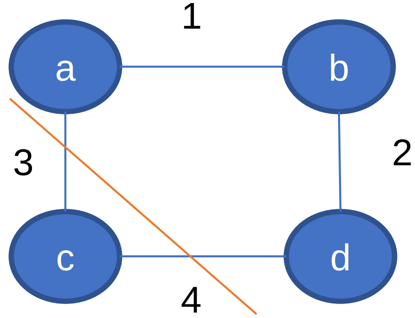
- Let G be a weighted connected graph, and let (A, B) be some possible cut of G.
- If *e* is the cheapest edge crossing cut (A, B), then *e* must be a part of some MST

What we are trying to prove

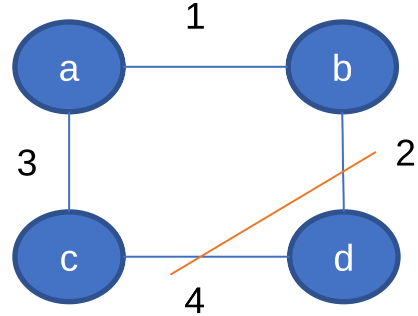
If we have an edge in a graph and you can find just a single cut for which this edge has the min cost among all edges crossing this cut, then this edge **must** belong to the MST (or one of MSTs in case when the weights are not unique)



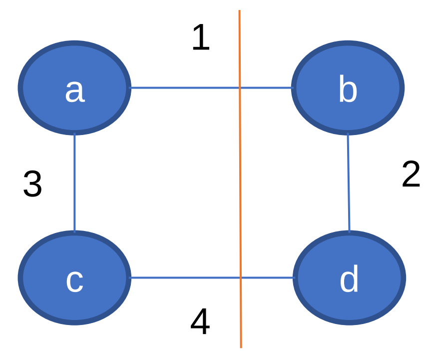
Cut 1
Edge 1 must be in MST



Cut 2
Edge 3 must be in MST



Cut 3
Edge 2 must be in MST



Cut 4
Edge 1 must be in MST

Note that edge 4 is never min of all crossing edges, no matter how we cut – so edge 4 is not in MST

Proof

- Let T be a minimum spanning tree of G. If T does not contain edge e, the addition of e to T must create a cycle.
- Therefore, there is some edge f of this cycle that has one endpoint in partition A and the other in partition B. Moreover, $w(e) \leq w(f)$.
- If we remove f from $T \cup \{e\}$, we obtain a spanning tree whose total weight is no more than before.
- Since T was a minimum spanning tree, this new tree must also be a minimum spanning tree.

In fact, if the weights in G are distinct, then the minimum spanning tree is unique

Theorem 2. Prim outputs a Minimum Spanning Tree

- If we consider a cut of G into X (MST so far) and V-X (remaining graph), then according to the Cut Crossing Theorem the cheapest edge for this cut must be a part of some MST
- Therefore, choosing the crossing edge with the minimum weight is a **safe move.**
- Because Prim's algorithm always adds a crossing edge of min-weight, the spanning tree produced by this algorithm is a Minimum Spanning Tree

Algorithm Kruskal_MST (graph G(V,E))

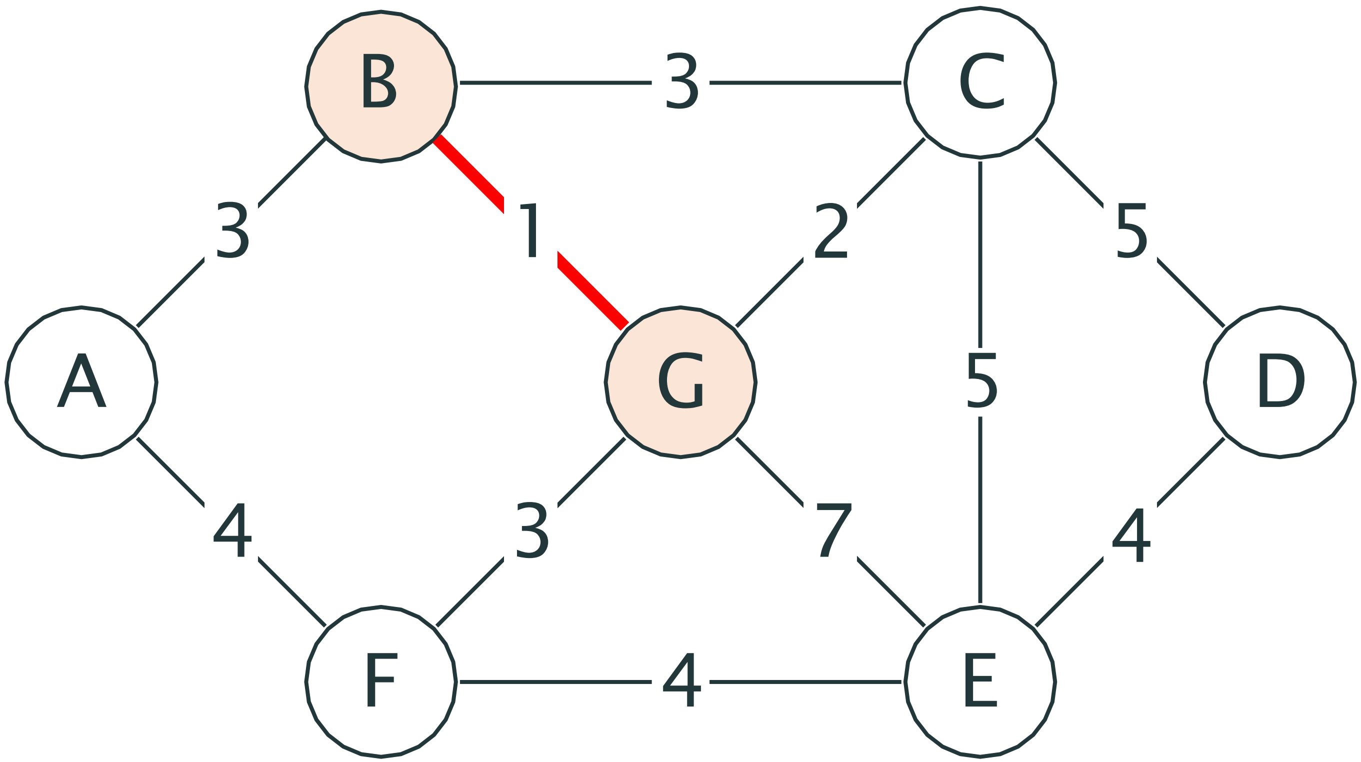
E’ := edges of G sorted by weights

T : = Ø

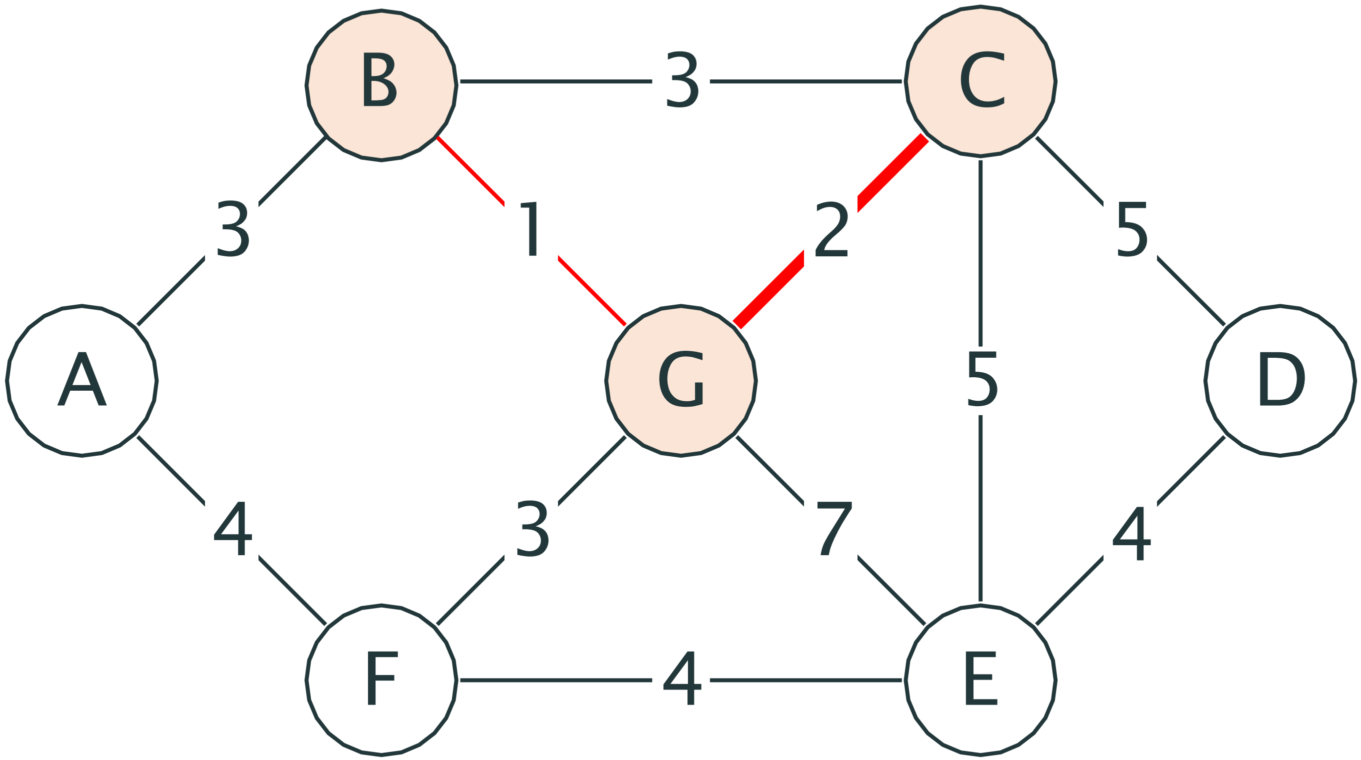
for i from 1 to m:
    if T U {E’[i]} has no cycles
        add E’[i] to T

return T

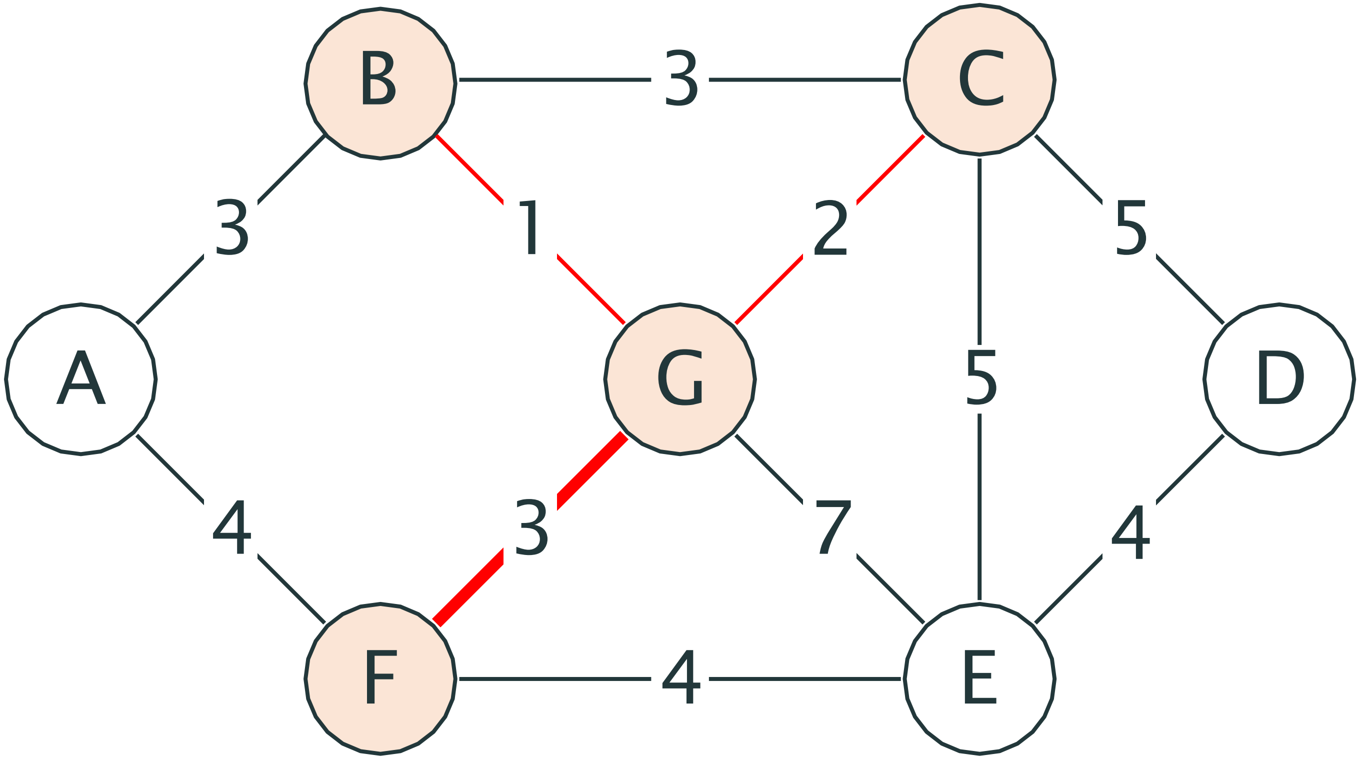
Kruskal illustration



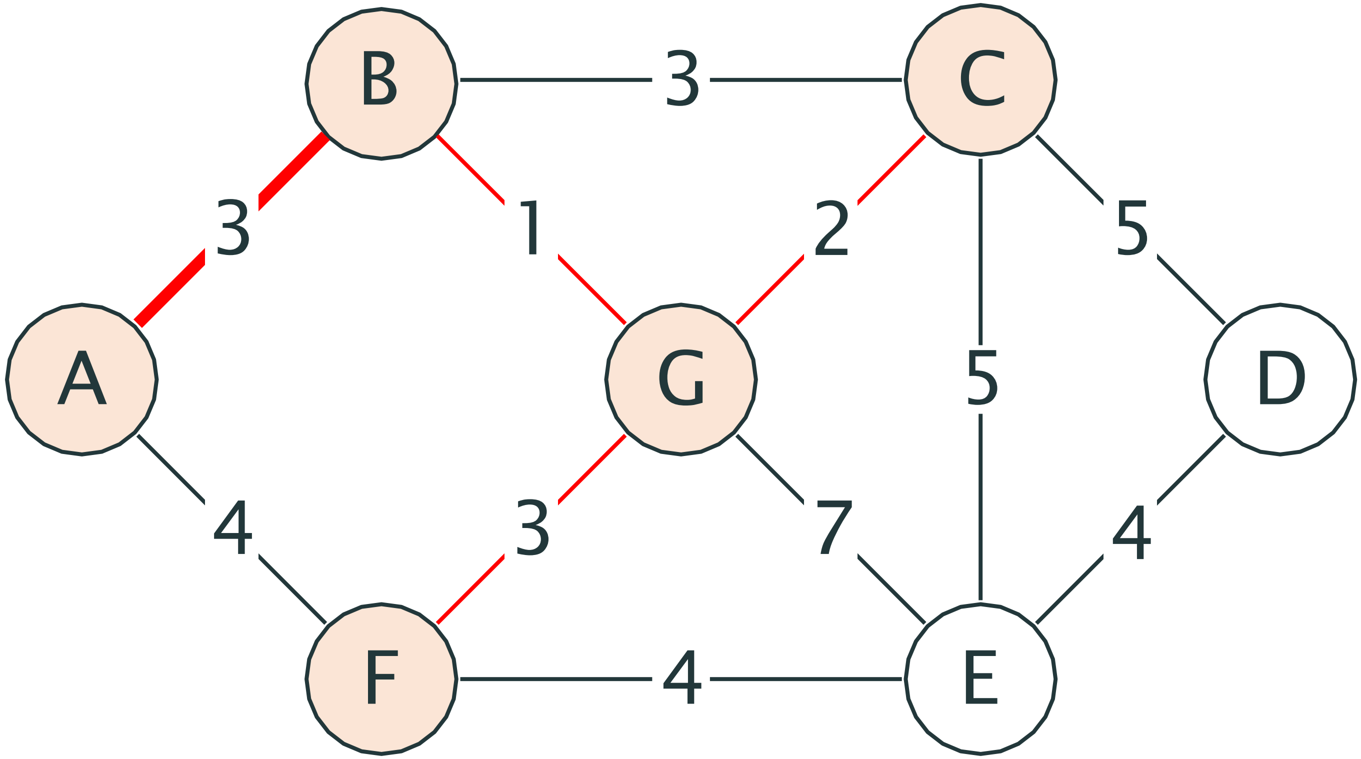
Kruskal illustration



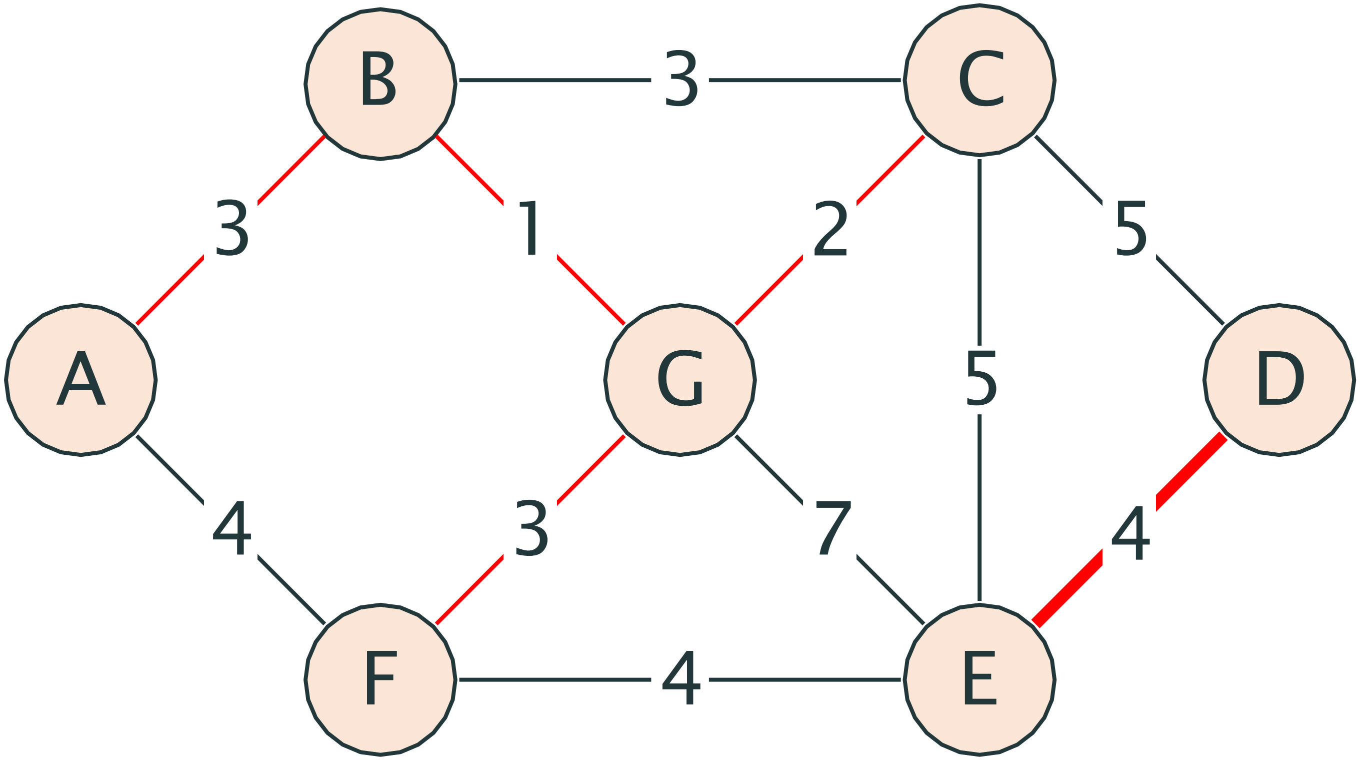
Kruskal illustration



Kruskal illustration

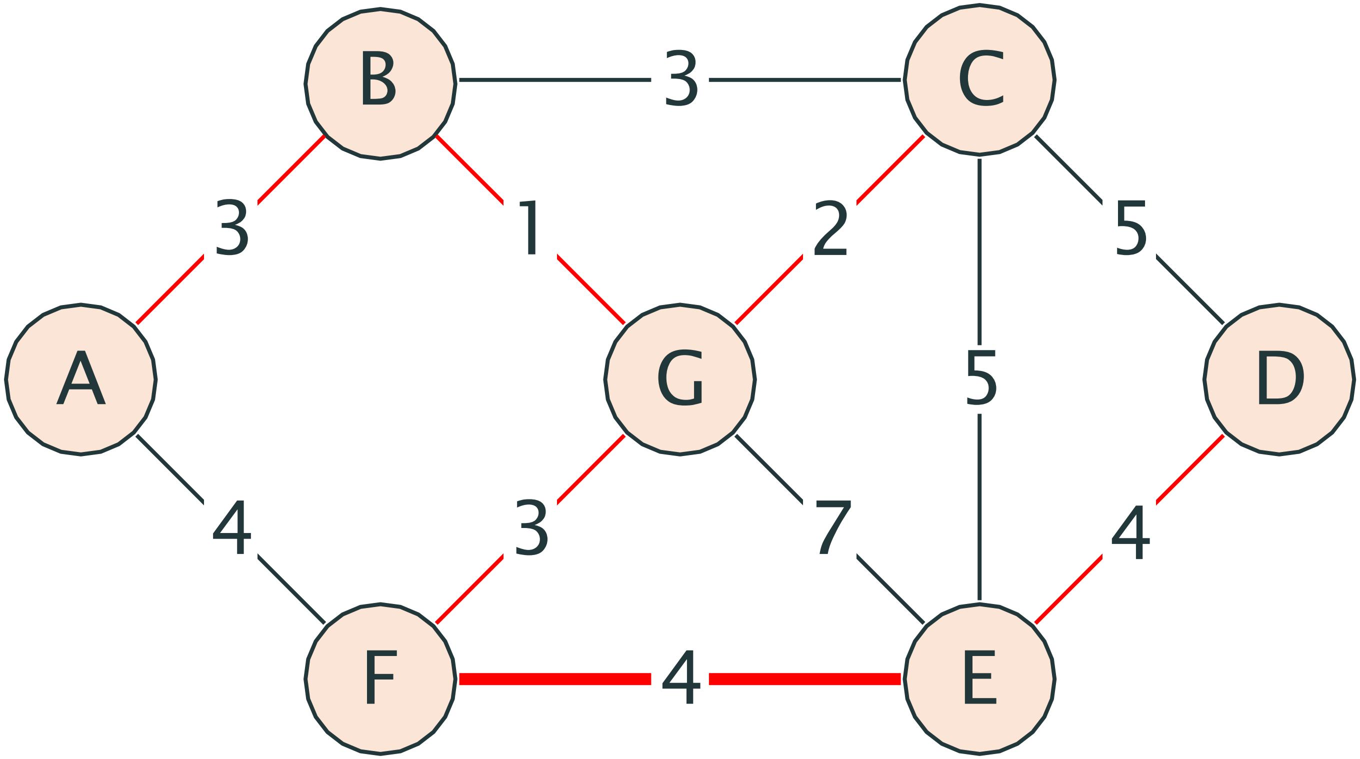


Kruskal illustration



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

Kruskal correctness (sketch)

Part I. Kruskal outputs Spanning Tree

- We explicitly check not to introduce cycles, and we add total n-1 edges connecting n nodes. Thus Kruskal produces a Spanning Tree of G

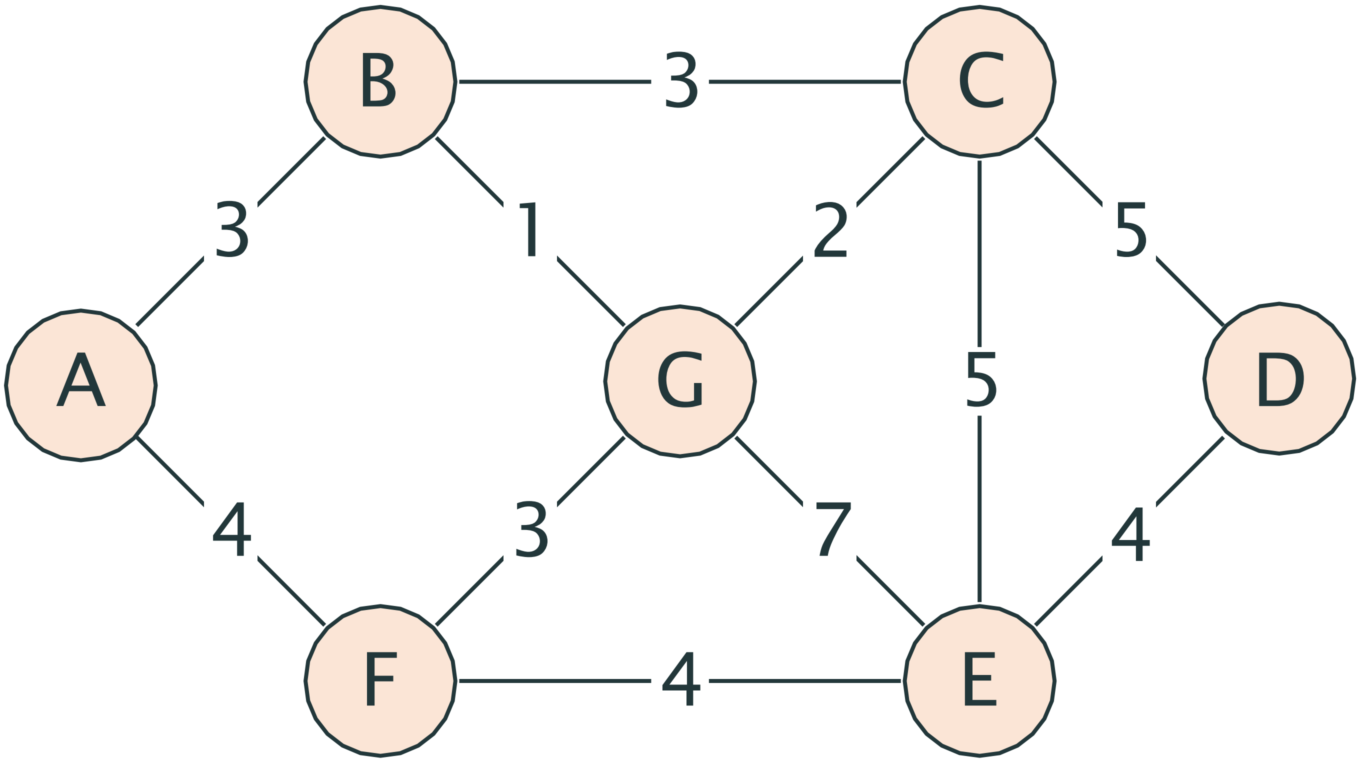
Part II. The tree is MST

- At each step, the algorithm adds a cheapest edge which does not create a cycle. This means that this is the first of crossing edges if we consider a cut of G into a connected component and remaining nodes
- By the Cut Crossing Theorem, this edge must be a part of some MST

Algorithm Baruvka_MST (graph G(V,E))

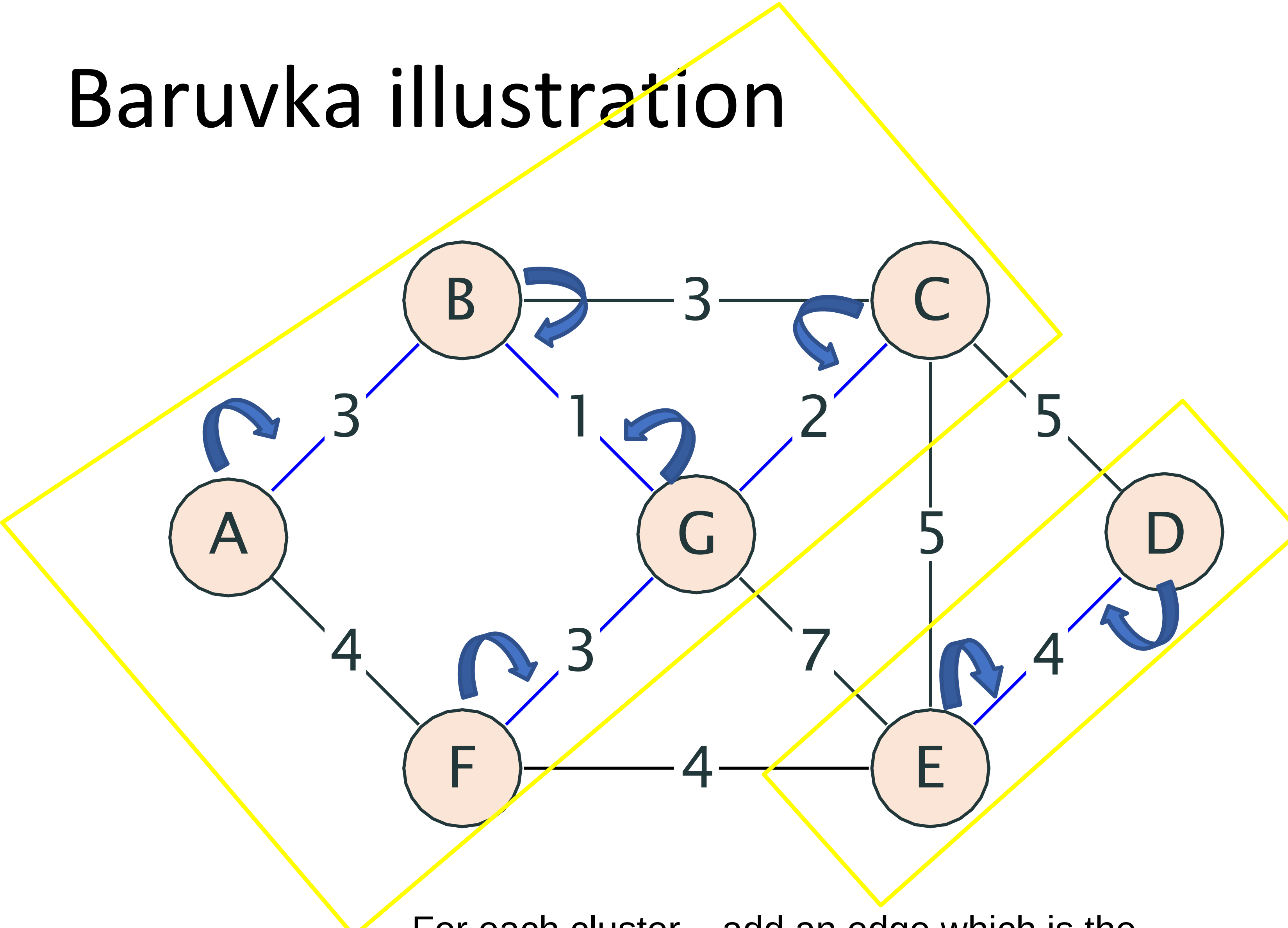
```
T: = Ø
# We create n clusters, each with a single node
assign each vertex v_i to cluster C_i
while length(clusters) != 1:
    for each cluster A in clusters:
            select min-cost of all edges (u,v)
                    such that u ∈ A and v ∉ A (v ∈ B)
            add edge (u,v) to T
            merge A and B into a single cluster A
return T
```

Baruvka illustration



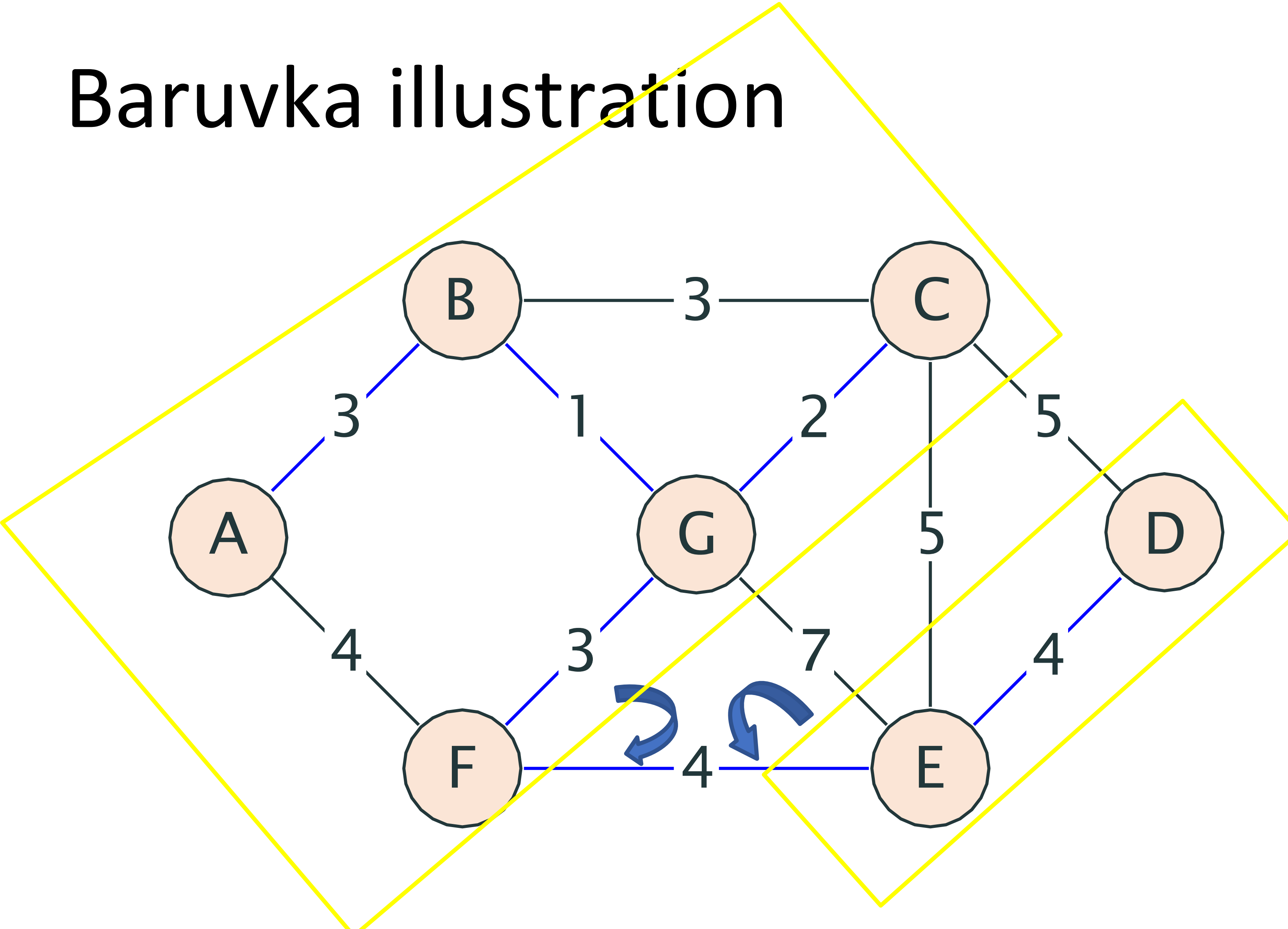
Each node as a single cluster, no edges

Baruvka illustration



For each cluster – add an edge which is the cheapest from this cluster
After first iteration: only 2 clusters remain

Baruvka illustration



After second iteration – 1 cluster:
MST cost: (3 + 1 + 2 + 3) + (4) + (4) = 17

MST algorithms: summary

All the algorithms follow some greedy strategy.

Algorithm MST (graph G(V,E))

$T := \emptyset$ # collects edges of the future MST

while $|T| \leq |V| - 1$:
    select next edge e from E # safe greedy move
    $T := T \cup e$

return T

Correctness proofs are all based on the
Cut Crossing Theorem