MST Algorithms: correctness

Lecture 05.04

by Marina Barsky

Problem: compute MST of Graph G

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

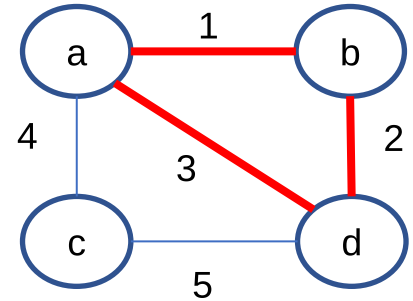
Output: minimum-cost tree $T \in$ E that spans all the vertices V

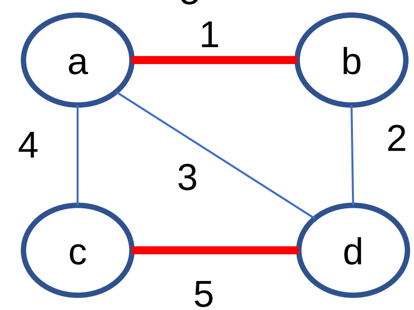
Assumptions:

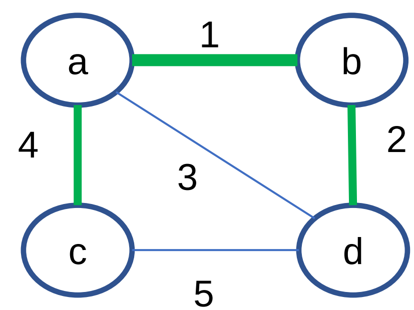
- ❑ *Input graph G is connected*

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 Prim_MST (graph G(V,E))

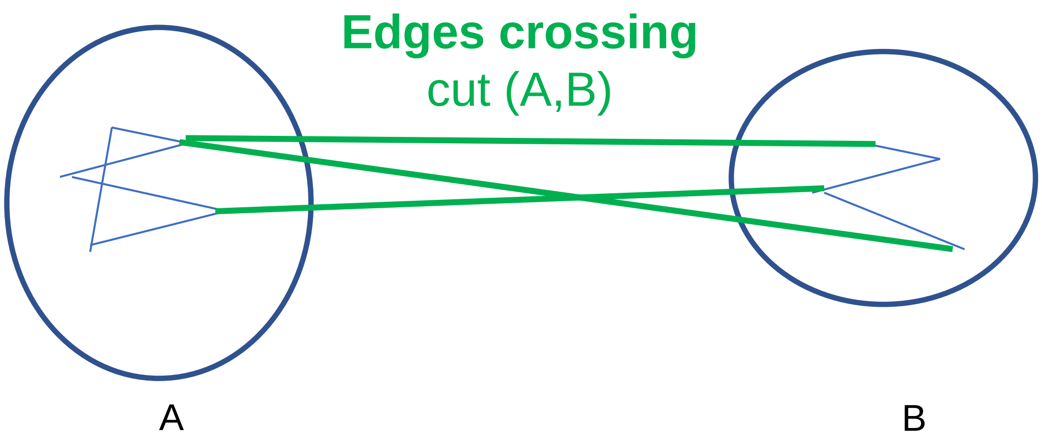
```
initialize tree T: = ∅              # set of tree edges
X: = {vertex s}                     # s ∈ V, chosen arbitrarily
# X contains vertices spanned by the tree-so-far


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

Selecting the cheapest edge sticking out of the current spanning tree (X,T) is **a greedy move**.

We need to prove that this move is a **safe move!**

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)? $2^n - 2$

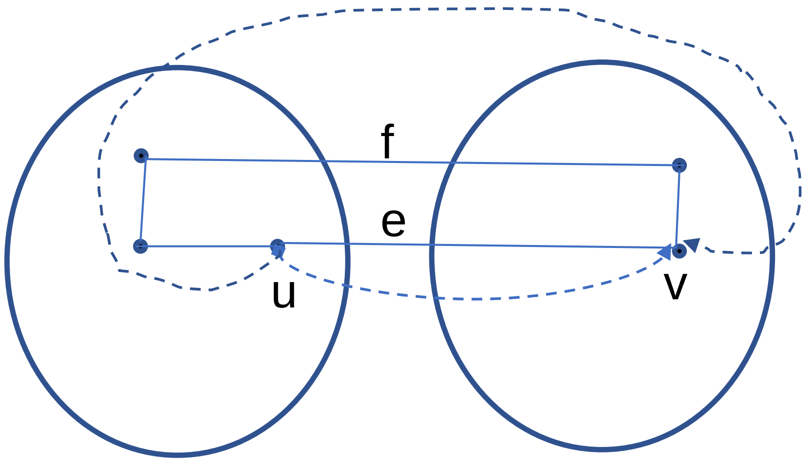


Crossing Edges Lemma

If there are (at least) two crossing edges for a cut (A,B) in an undirected connected graph, then these edges must be a part of some cycle.

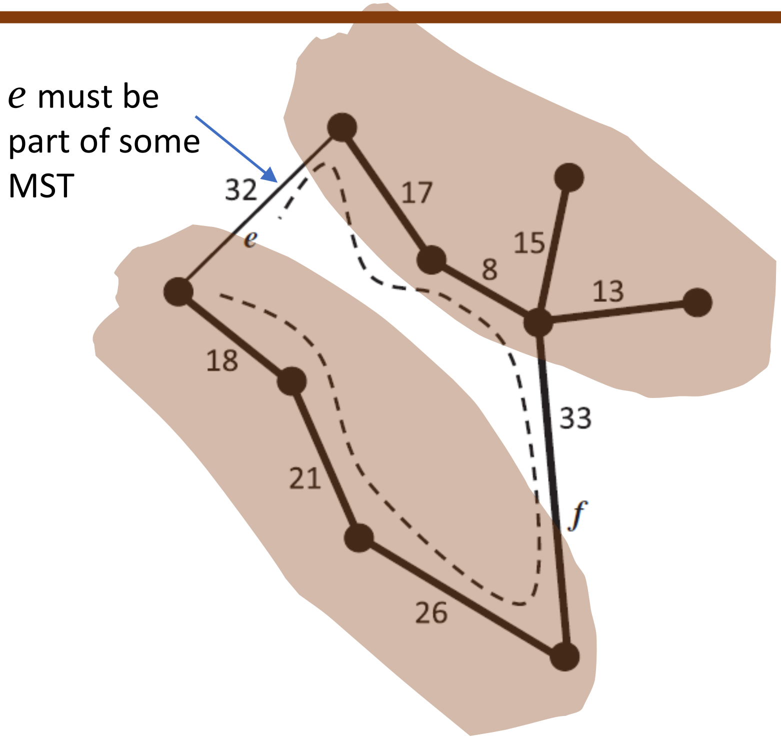
Proof

If there is a path from u to v from to two different partitions that includes the first crossing edge e, then the second crossing edge f offers an alternative path from u to v, thus closing the cycle on vertex v.



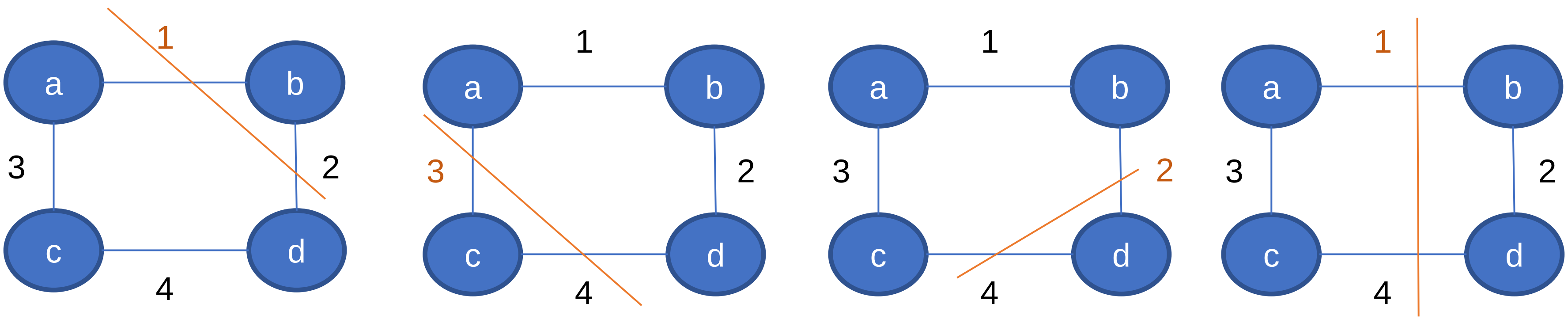
Cut Crossing Theorem

- 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

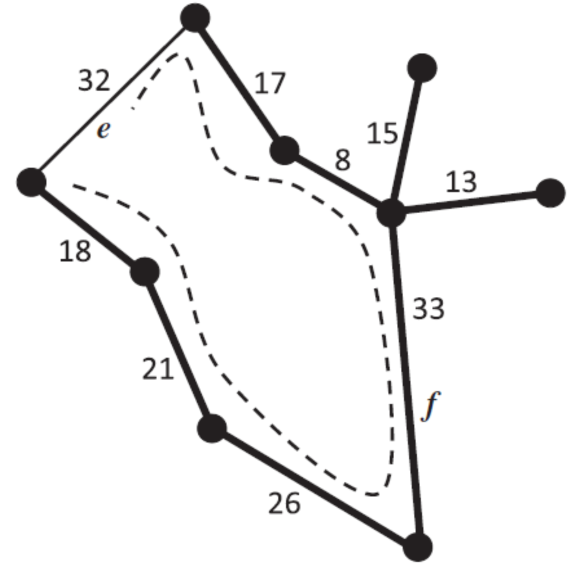
Proof

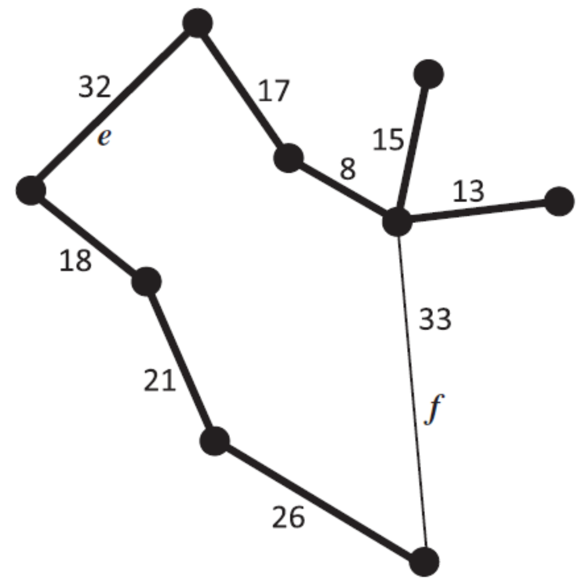
- Let T be a full minimum spanning tree of G that does not contain edge e. Then the addition of e to T must create a cycle. This is because all nodes in MST are already connected, and addition of edge e will offer an alternative path between some nodes.
- Let then consider a cut (A,B) of this MST, which has two crossing edges e and f, both on the same cycle. Edge f belongs to MST and edge e does not. Now let's assume that $w(e) \leq w(f)$.
- If we remove f from T and replace it with e, then 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 all distinct, then the minimum spanning tree is unique, and it *must* contain edge e instead of f

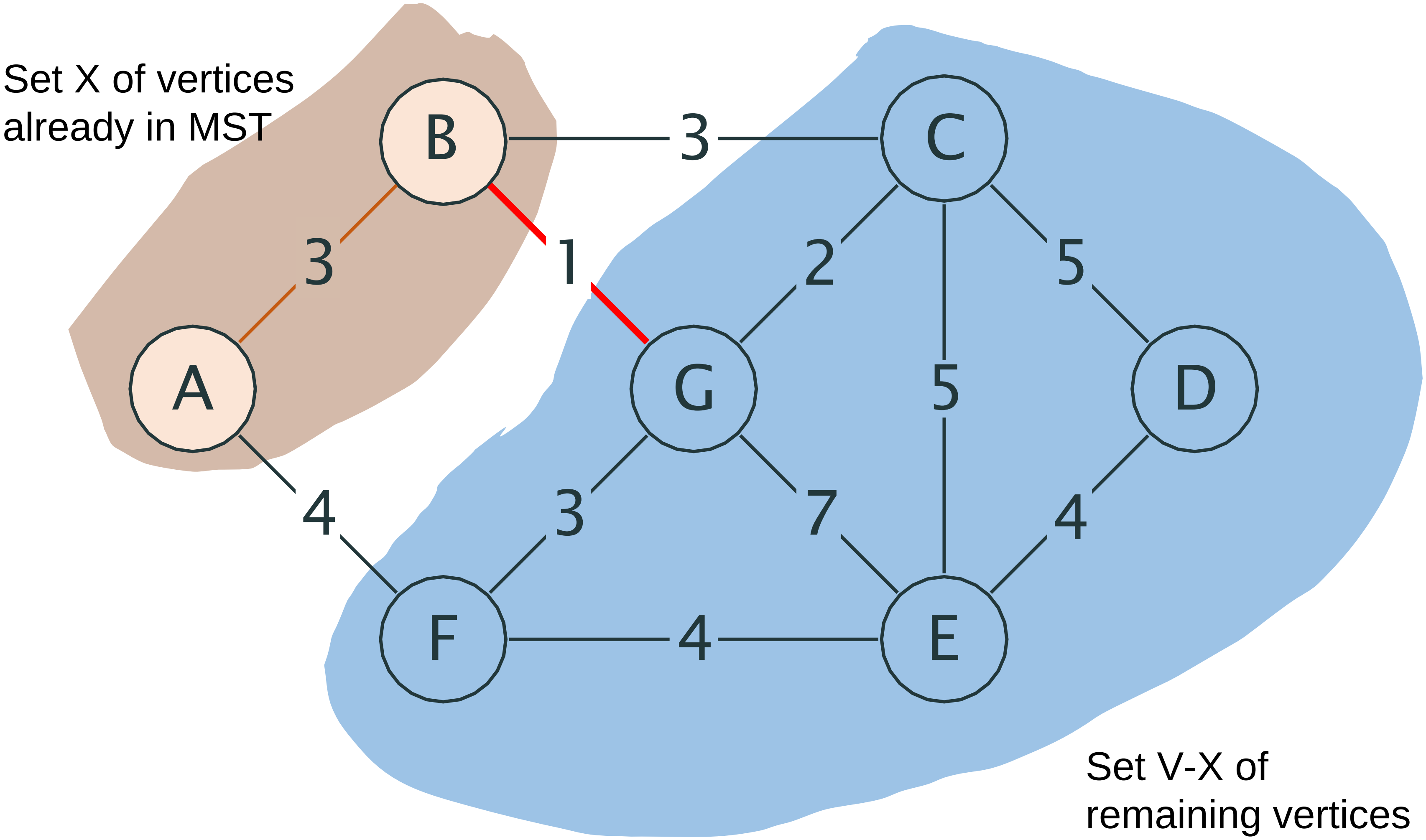
Exchange argument!

- 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.





Prim: cut


Set X of vertices
already in MST
B
3
C
3
1
2
5
A
G
5
D
4
3
7
4
F
4
E
Set V-X of
remaining vertices

Theorem: 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 crossing 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 : = ∅


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


return T
```

Kruskal: correctness (sketch)

Part I. Kruskal outputs a 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 for some cut of G
- By the **Cut Crossing Theorem**, this edge must be a part of some MST

MST algorithms: summary

All the algorithms follow some greedy strategy.

Algorithm MST (graph G(V,E))

T : = Ø   # collects edges of the future MST

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

return T

Correctness proofs are all based on the
Cut Crossing Theorem