MINIMUM SPANNING TREES

David Kauchak
CS 140 – Spring 2020

Admin

Assignment 8

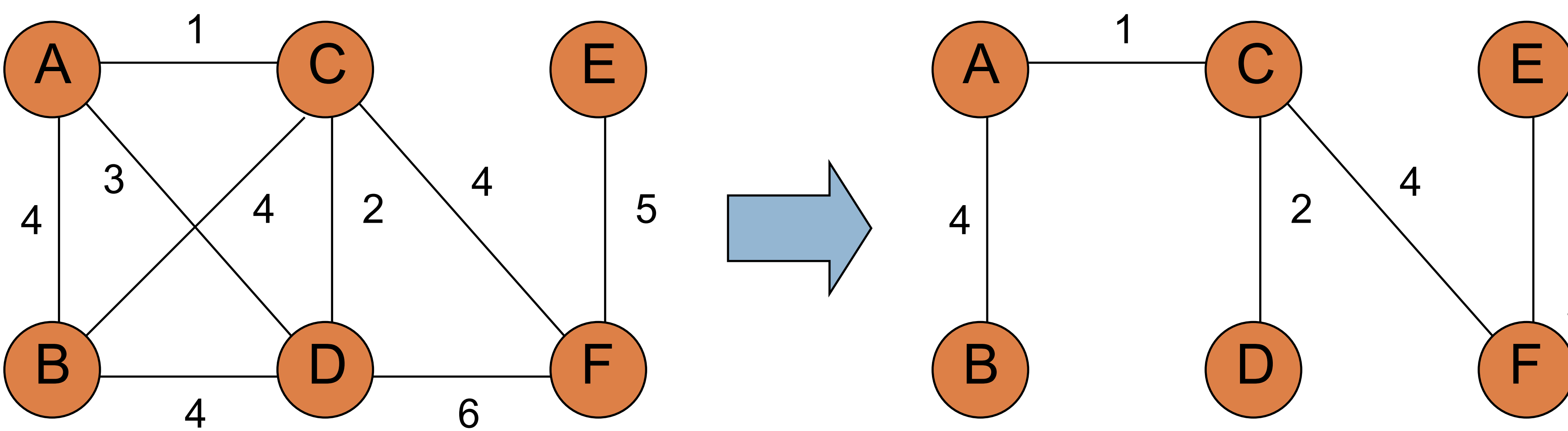
Lab tomorrow:

- Course feedback
- Summary/Review
- Interview programming questions (optional)

A few last shortest paths things

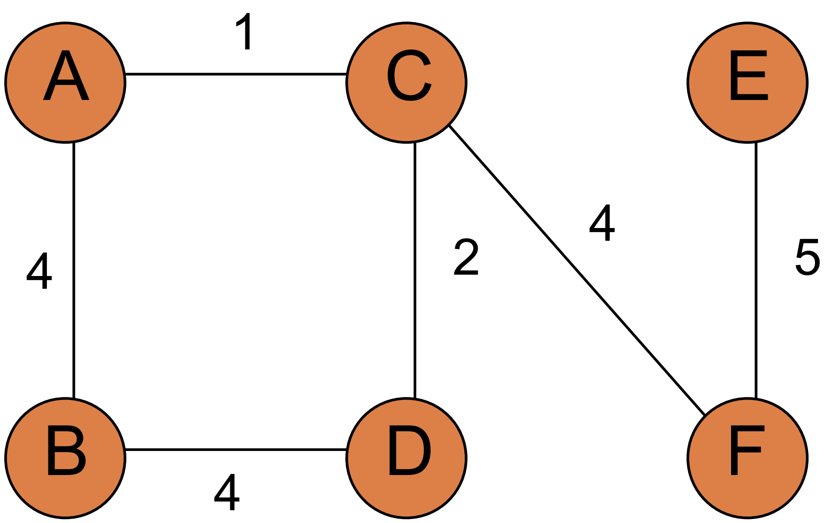
Minimum spanning trees (MST)

The lowest weight set of edges that connects all vertices of an undirected graph with positive weights



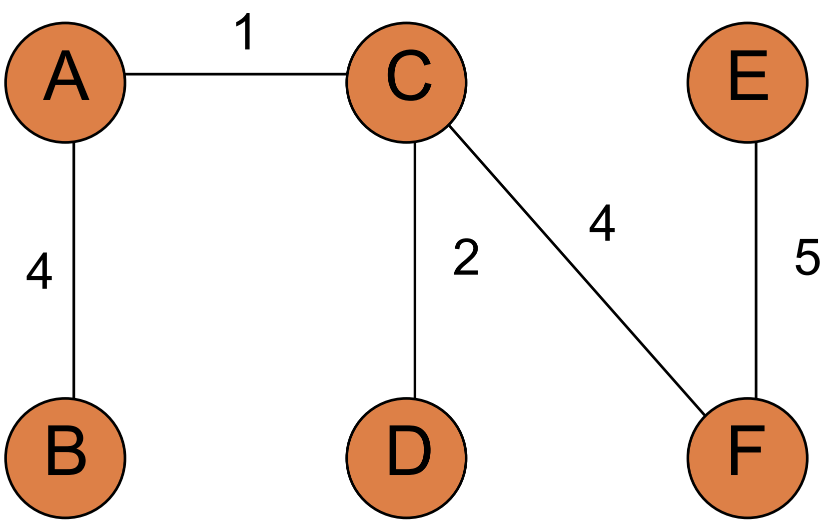
MSTs

Can an MST have a cycle?



MSTs

Can an MST have a cycle?



Applications?

Connectivity

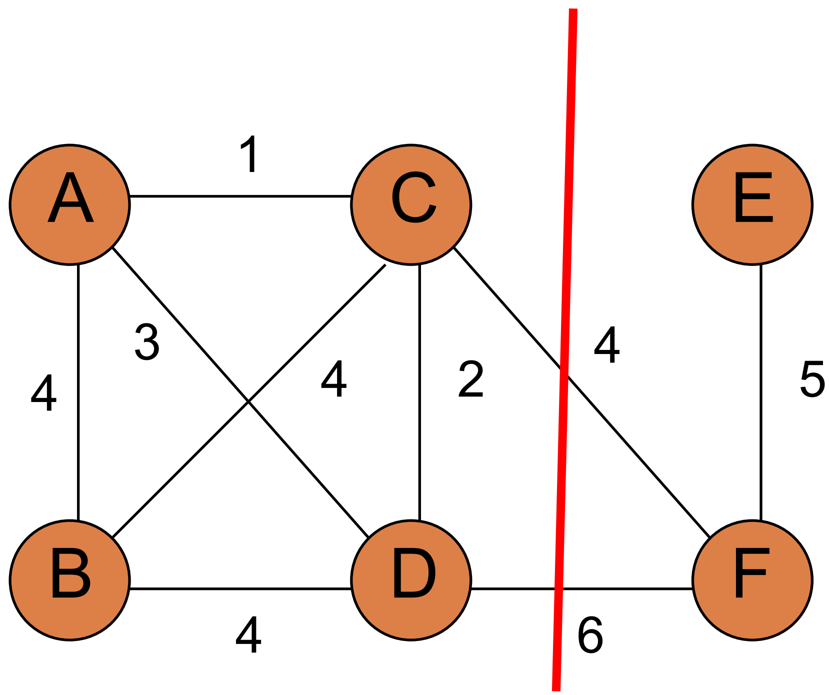
- Networks (e.g. communications)
- Circuit design/wiring

hub/spoke models (e.g. flights, transportation)

Cuts

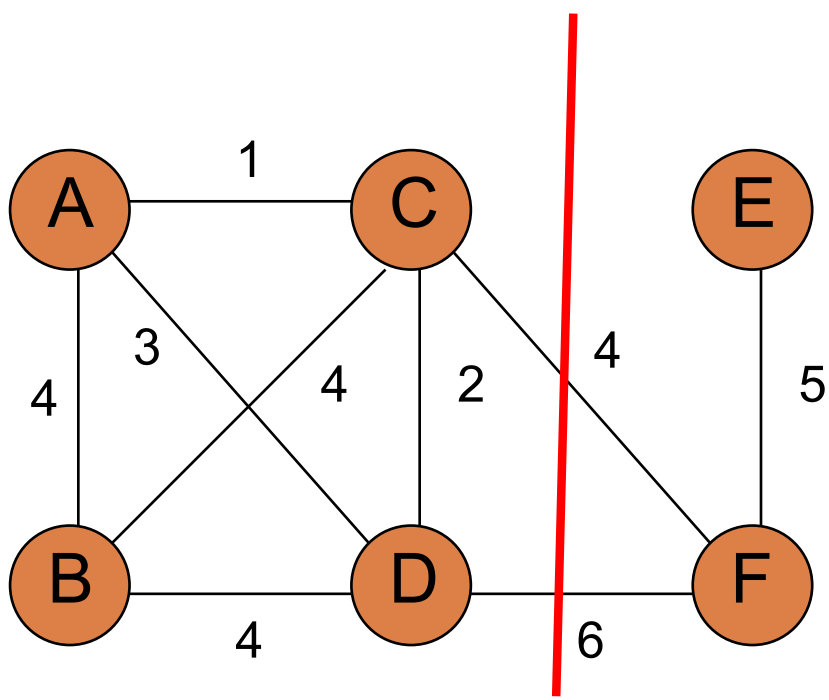
A cut is a partitioning of the vertices into two sets S and V-S

An edge "crosses" the cut if it connects a vertex $u \in V$ and $v \in V\text{-}S$



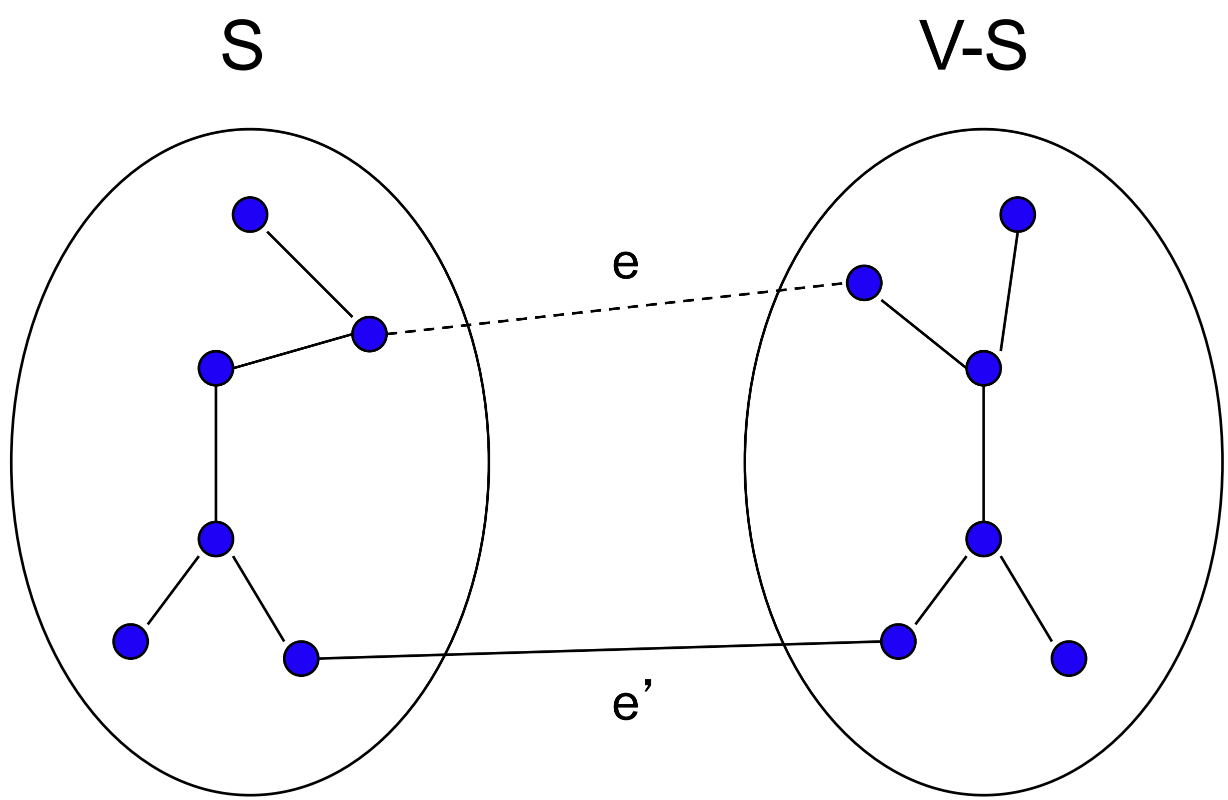
Minimum cut property

Given a partition S, let edge e be the minimum cost edge that **crosses** the partition. *Every* minimum spanning tree contains edge e.



Minimum cut property

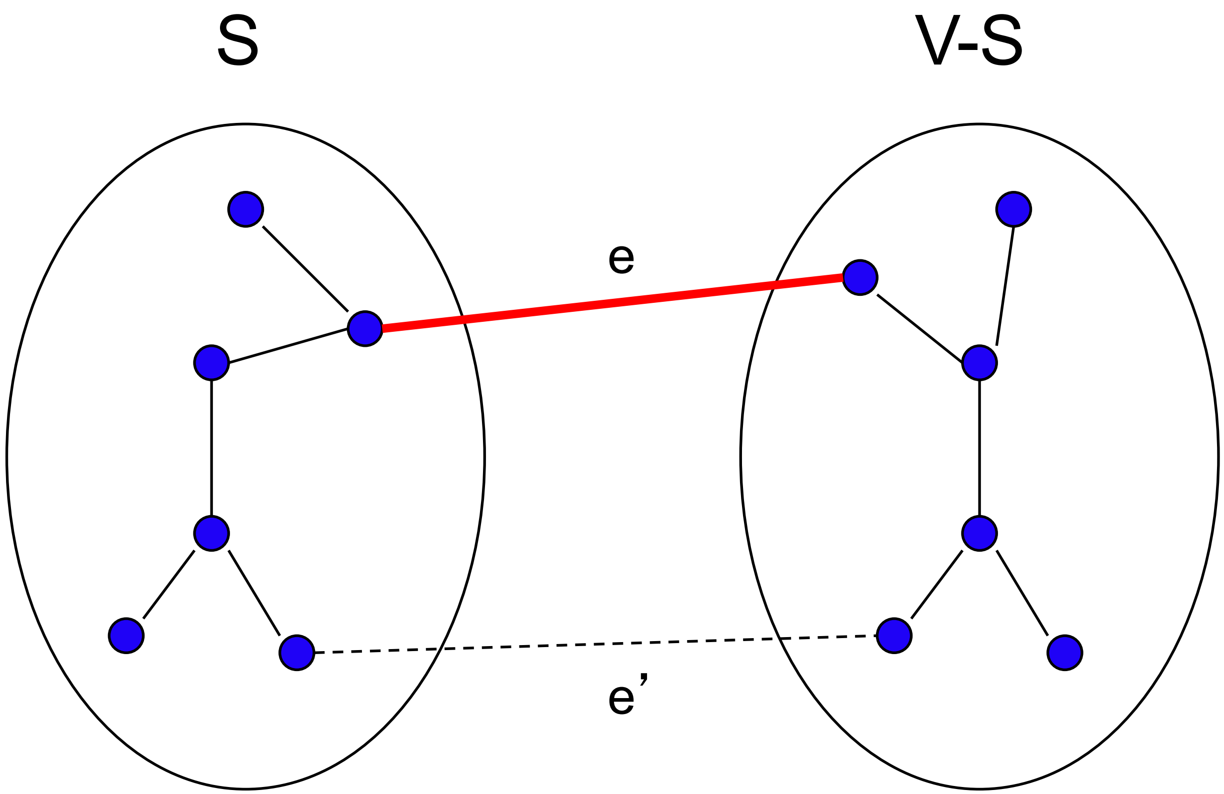
Given a partition S, let edge e be the minimum cost edge that **crosses** the partition. *Every* minimum spanning tree contains edge e.



Consider an MST with edge e' that is not the minimum edge

Minimum cut property

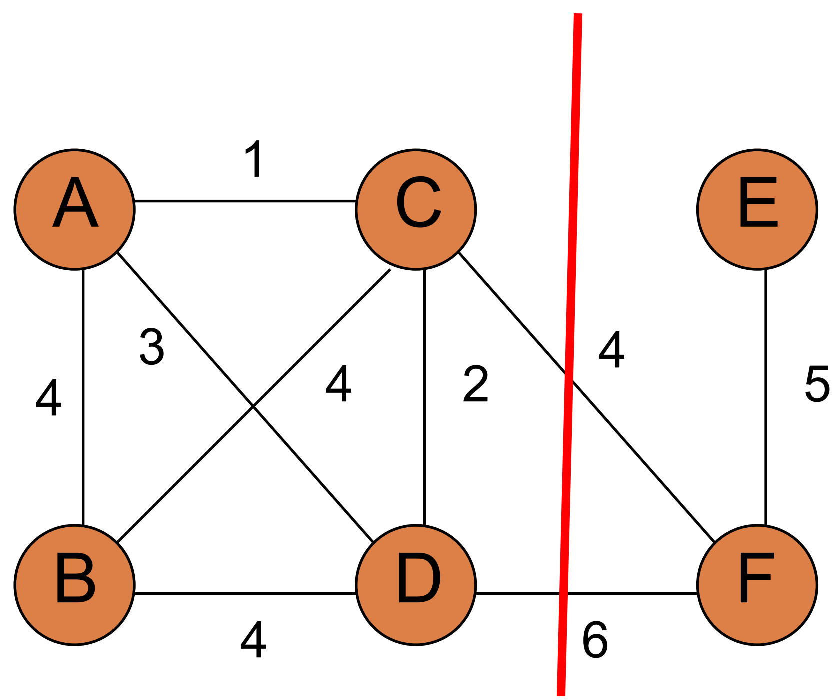
Given a partition *S*, let edge e be the minimum cost edge that **crosses** the partition. *Every* minimum spanning tree contains edge e.



Using e instead of e', still connects the graph, but produces a tree with smaller weights

Minimum cut property

If the minimum cost edge that **crosses** the partition is not unique, then *some* minimum spanning tree contains edge e.



Kruskal's algorithm

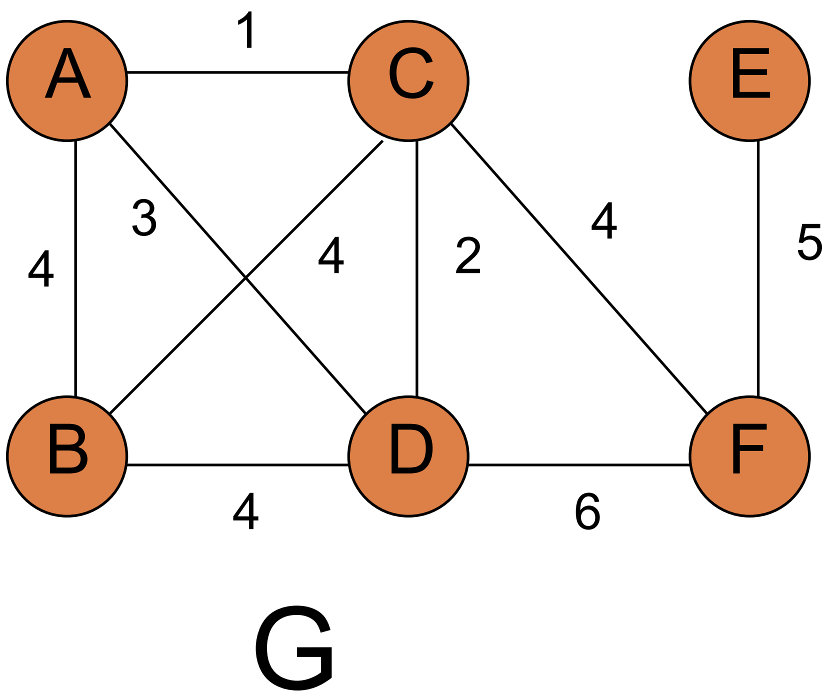
Given a partition S, let edge e be the minimum cost edge that **crosses** the partition. *Every* minimum spanning tree contains edge e.

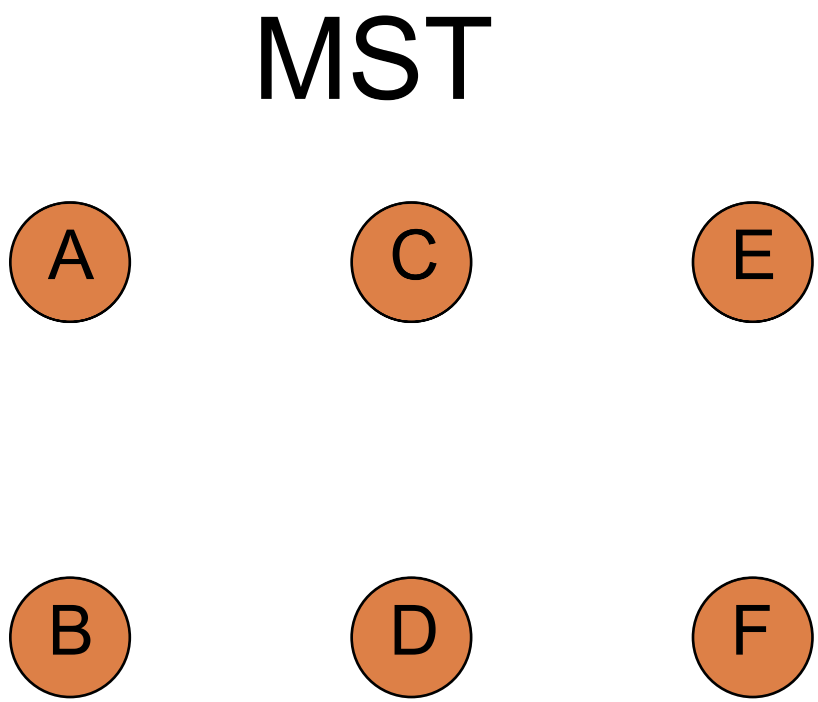
Kruskals:

- Sort edges by increasing weight
- for each edge (by increasing weight):
    - check if adding edge to MST creates a cycle
    - if not, add edge to MST

Kruskal's algorithm

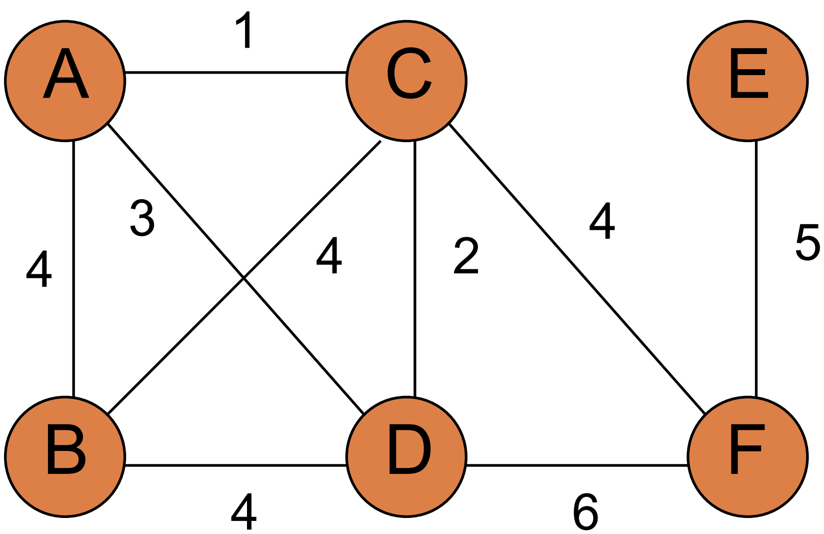
Add smallest edge that doesn't create a cycle





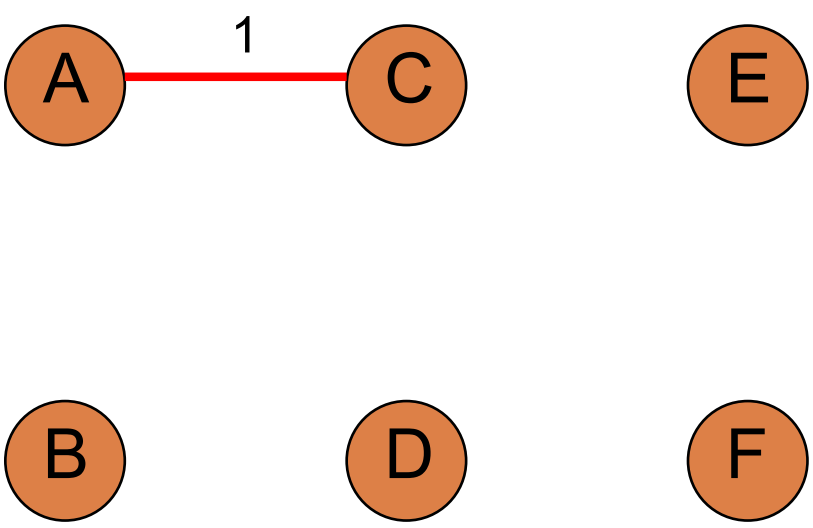
Kruskal's algorithm

Add smallest edge that doesn't create a cycle



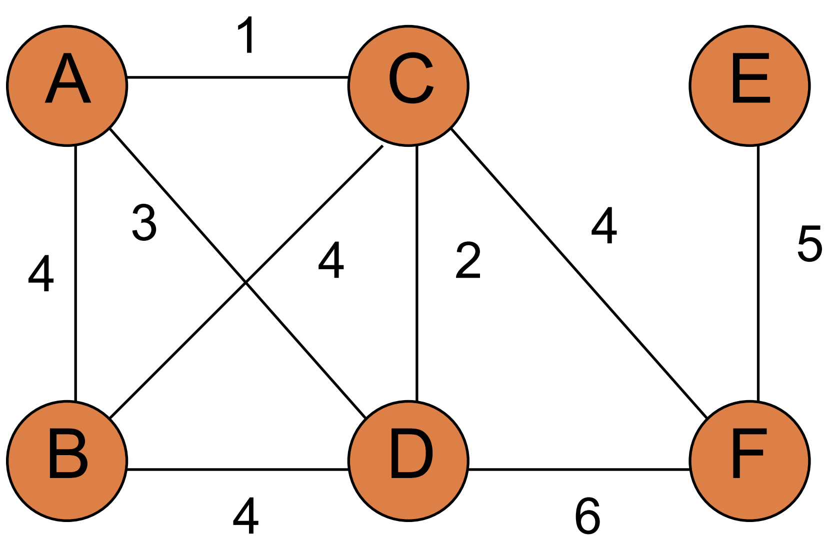
G

MST



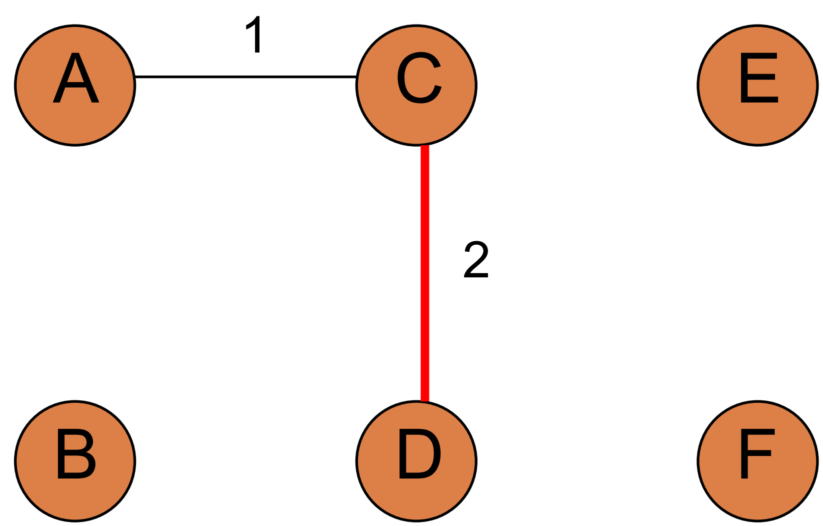
Kruskal's algorithm

Add smallest edge that doesn't create a cycle



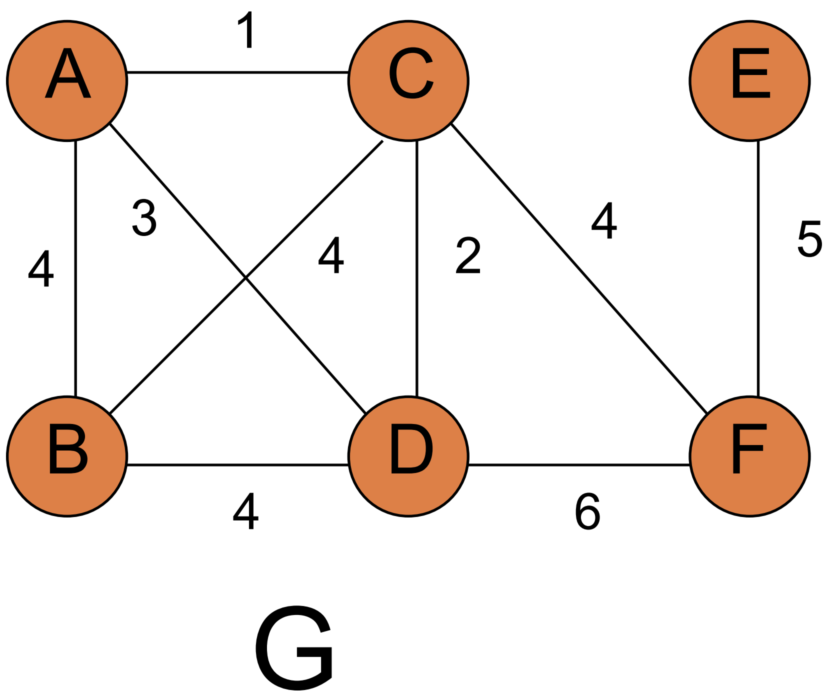
G

MST

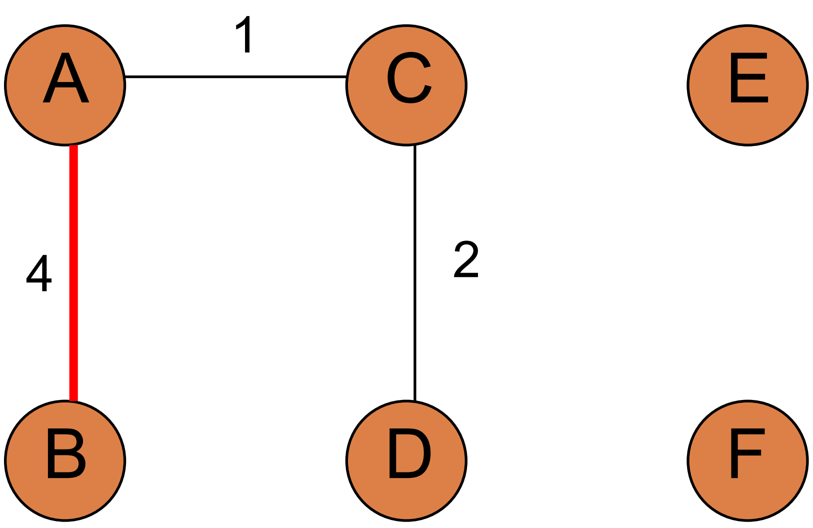


Kruskal's algorithm

Add smallest edge that doesn't create a cycle

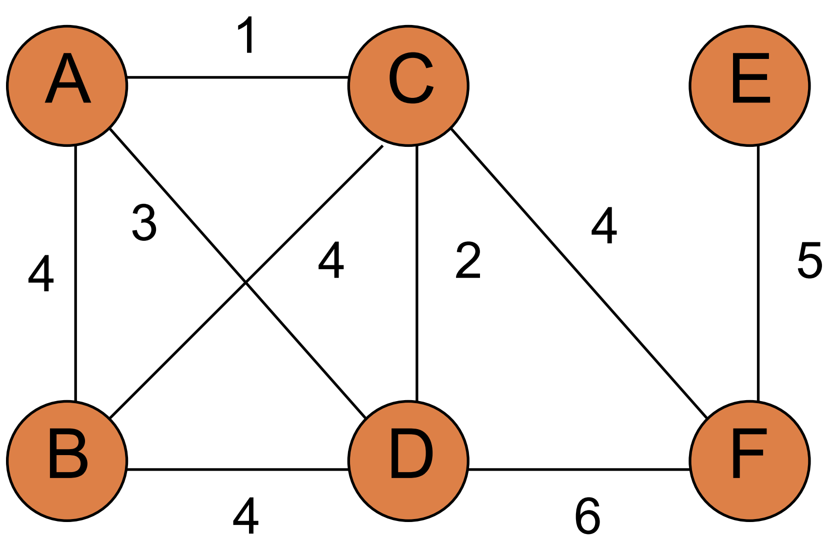


MST



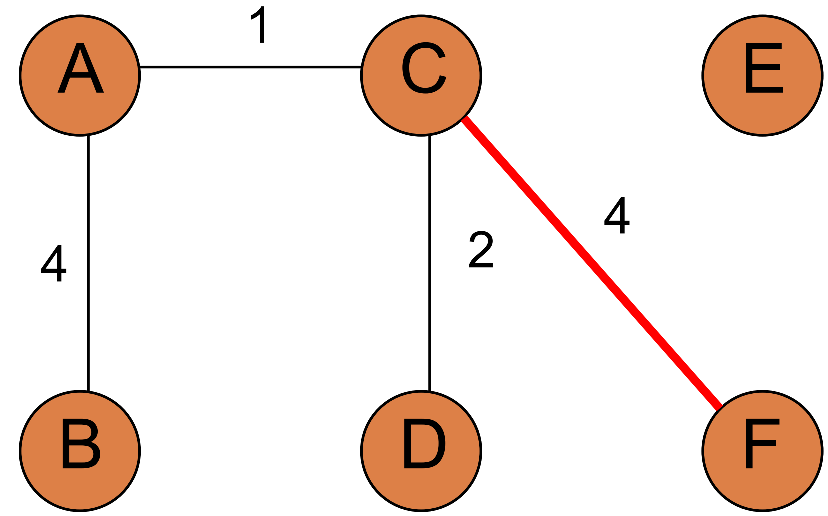
Kruskal's algorithm

Add smallest edge that doesn't create a cycle



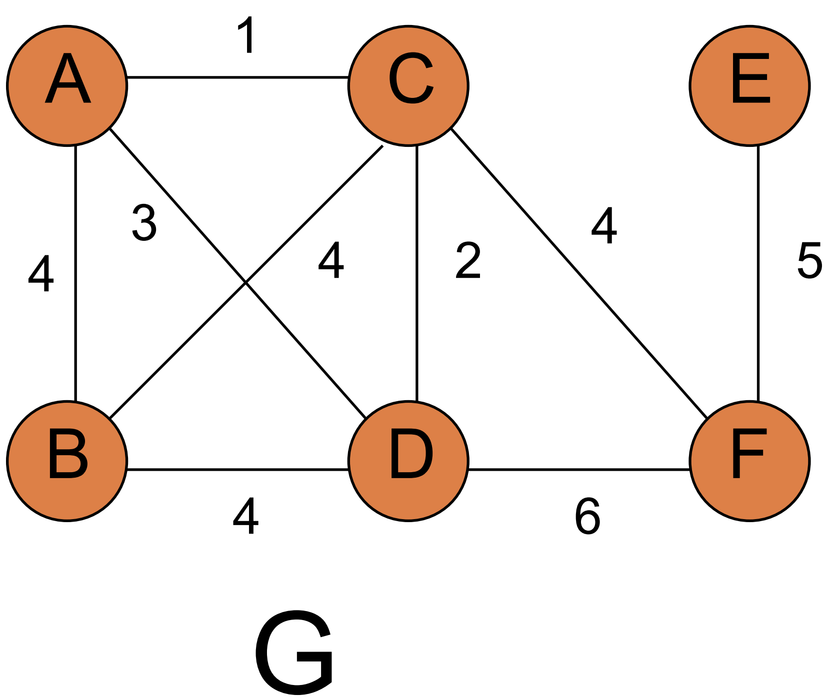
G

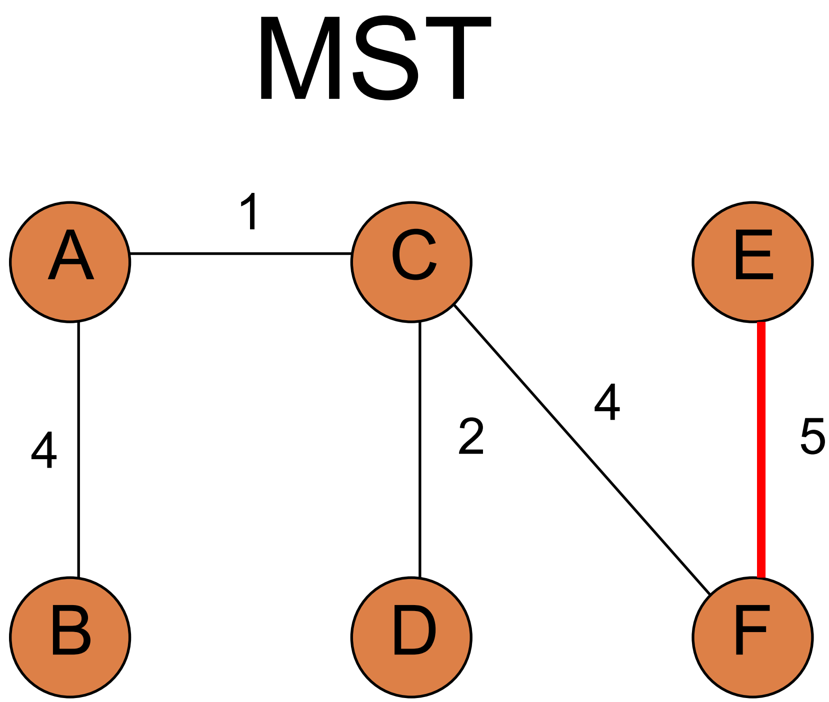
MST



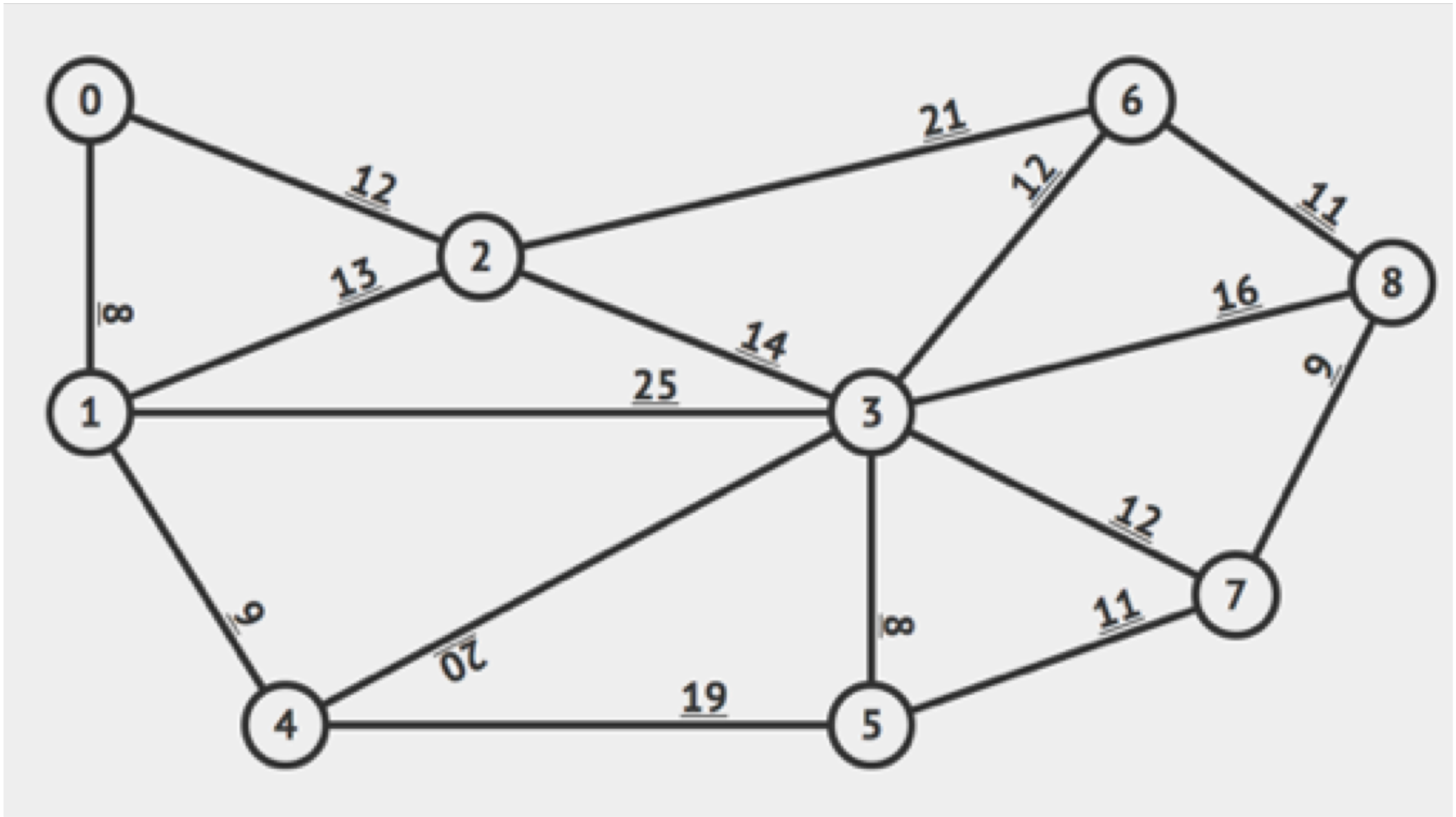
Kruskal's algorithm

Add smallest edge that doesn't create a cycle

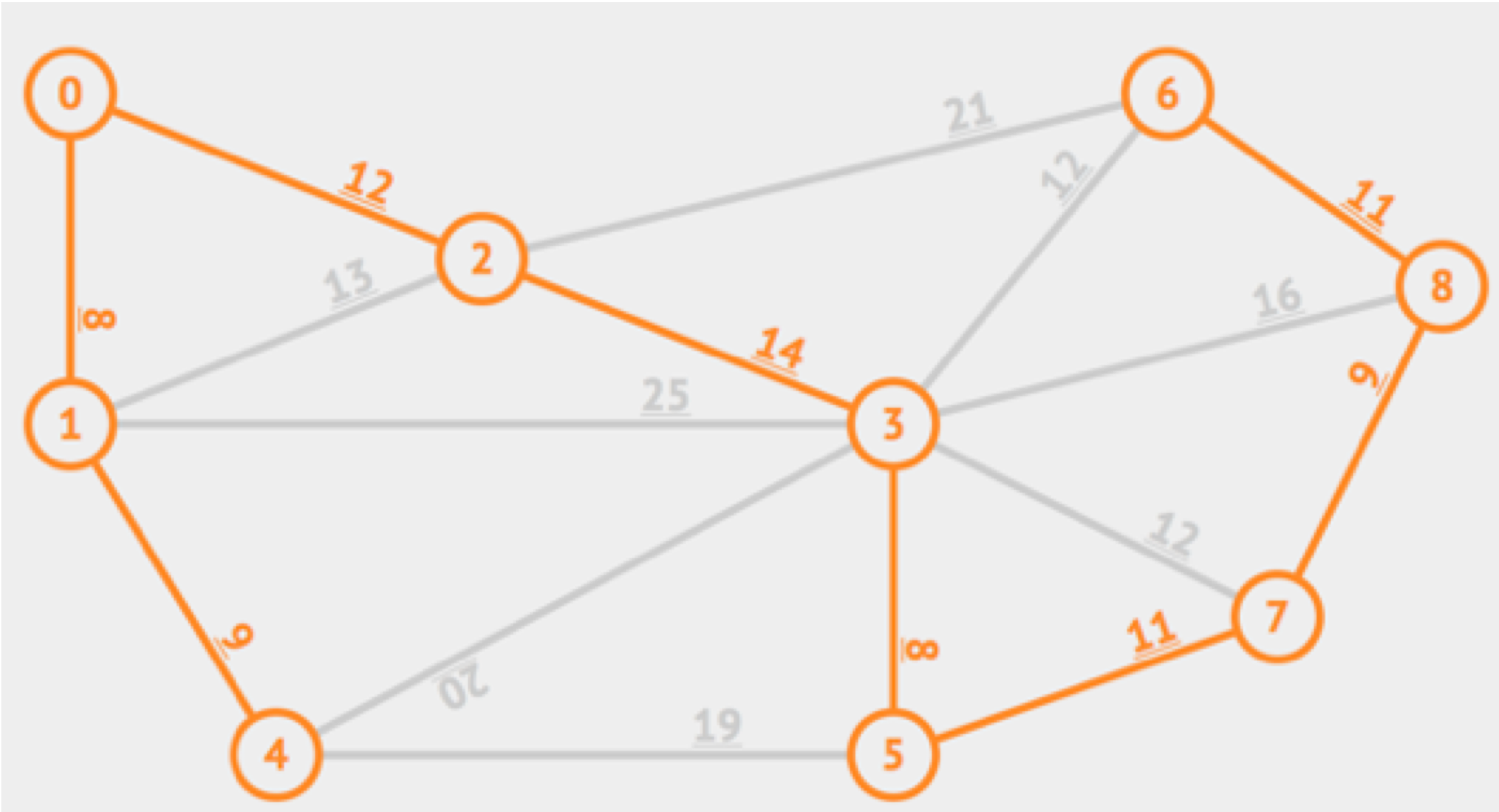




Practice



Solution



Sum = 8 + 8 + 9 + 9 + 11 + 11 + 12 + 14 = 82

Why does Kruskal's work?

Never adds an edge that creates a cycle

Therefore, always adds lowest cost edge to connect two connected components. By min cut property, that edge must be part of the MST

Kruskals:
- Sort edges by increasing weight
- for each edge (by increasing weight):
  - check if adding edge to MST creates a cycle
  - if not, add edge to MST

Kruskal's details

Uses a data structure called "disjoint set" to efficiently check whether adding an edge creates a cycle

Run-time: O(E log E) (bounded by the sort)

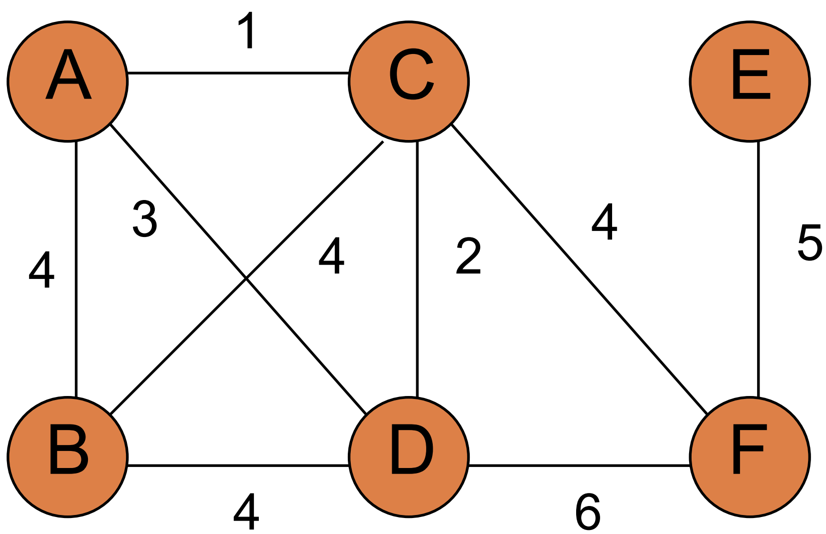
Prim's algorithm

Greedily grow the MST starting at a vertex:

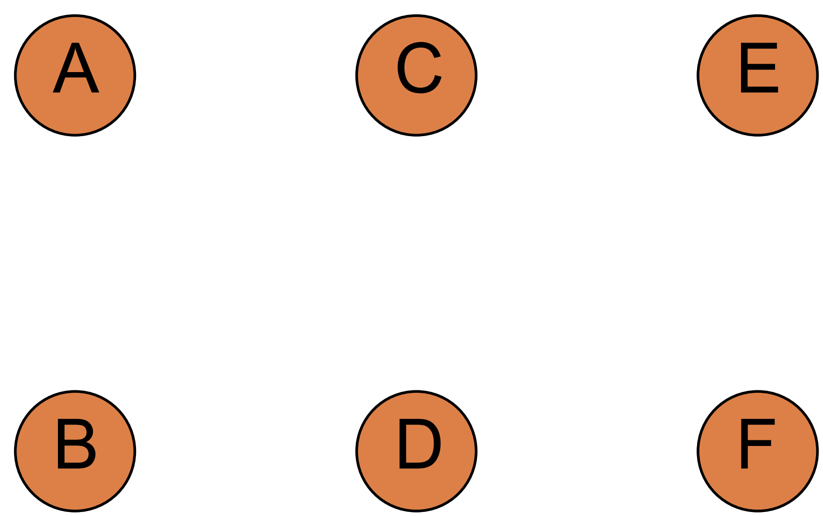
- Start with a random vertex and count that vertex as connected by the MST
- Add the edge with the smallest weight that connects a vertex not onnected by the MST
- Repeat until we've added V-1 edges

Prim's

Add the edge with the smallest weight that connects a vertex not connected by the MST

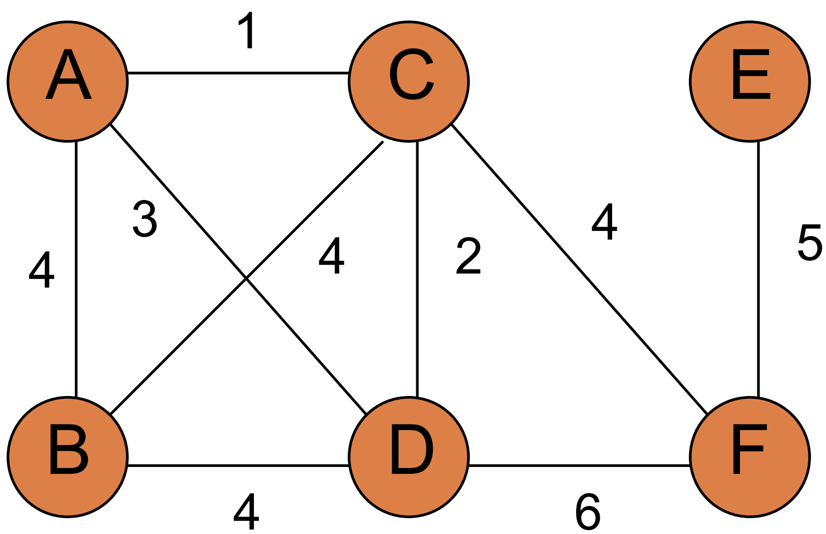


MST

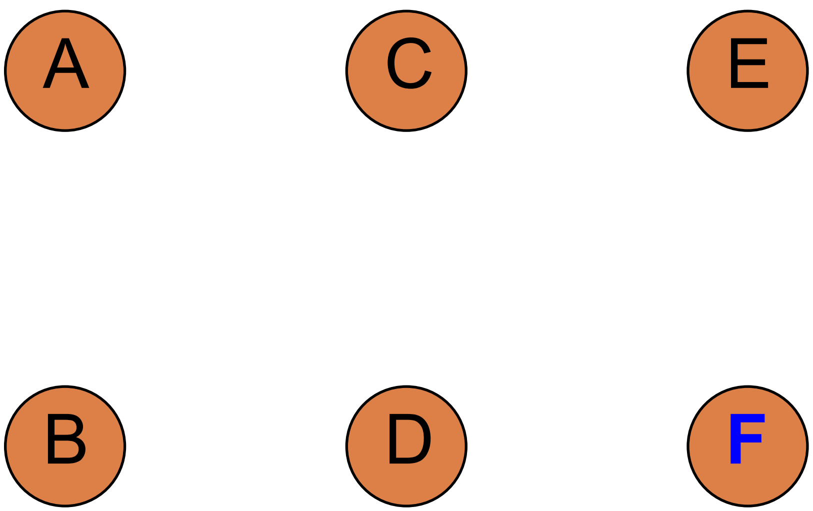


Prim's

Add the edge with the smallest weight that connects a vertex not connected by the MST

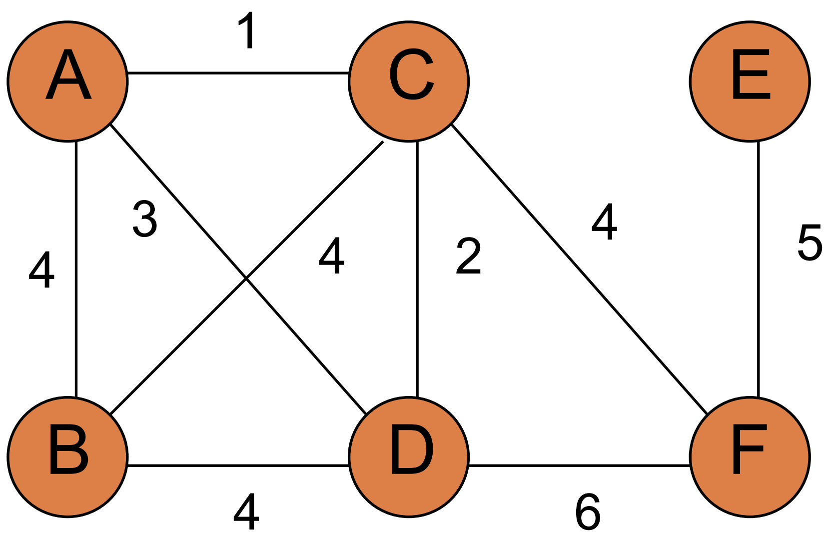


MST

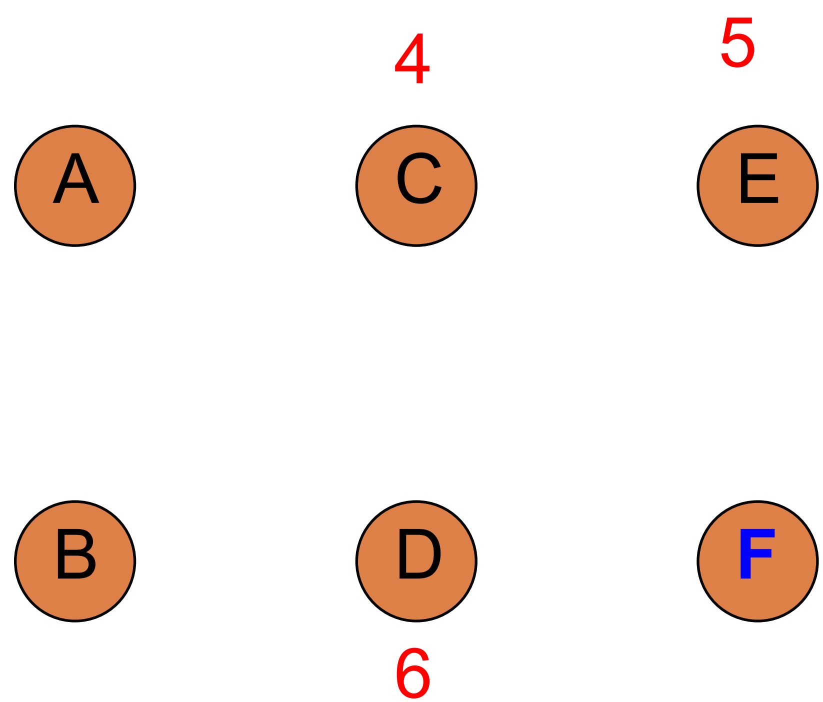


Prim's

Add the edge with the smallest weight that connects a vertex not connected by the MST

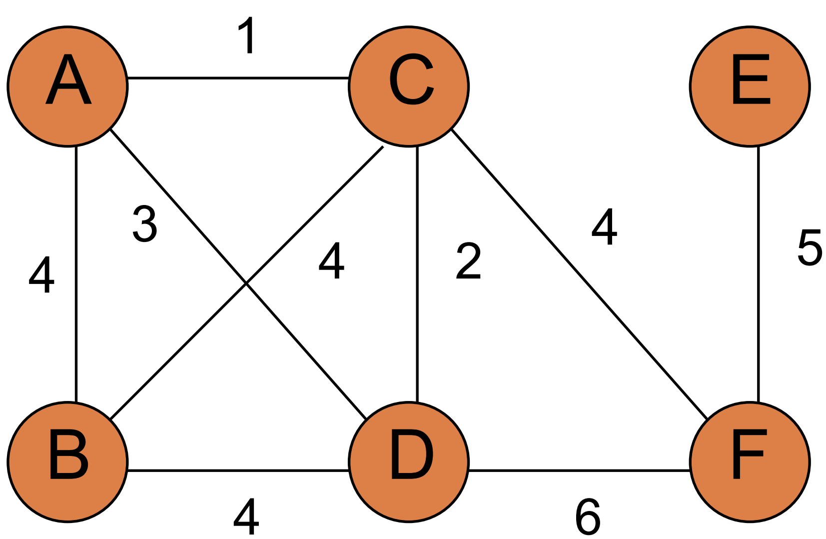


MST

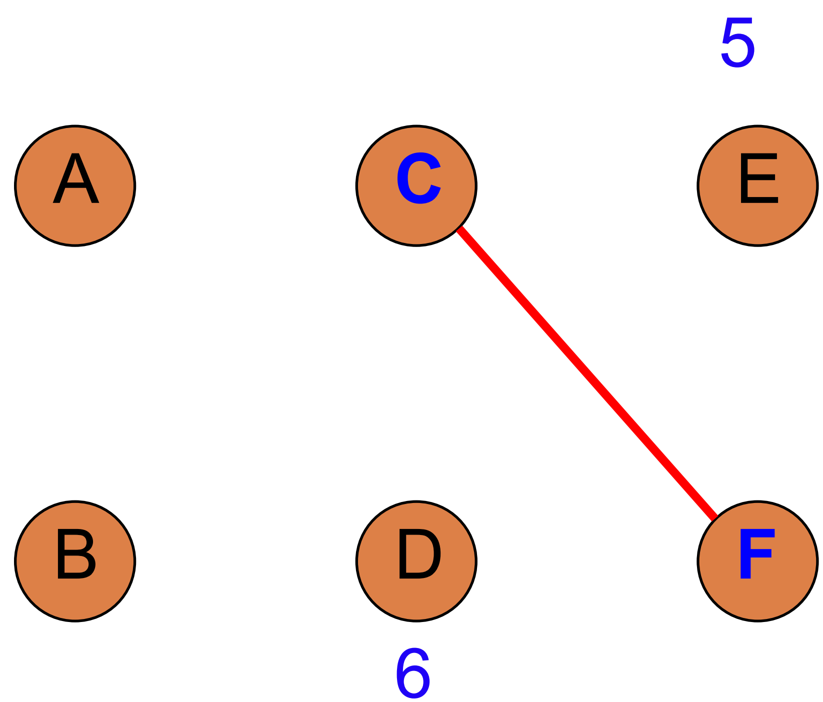


Prim's

Add the edge with the smallest weight that connects a vertex not connected by the MST

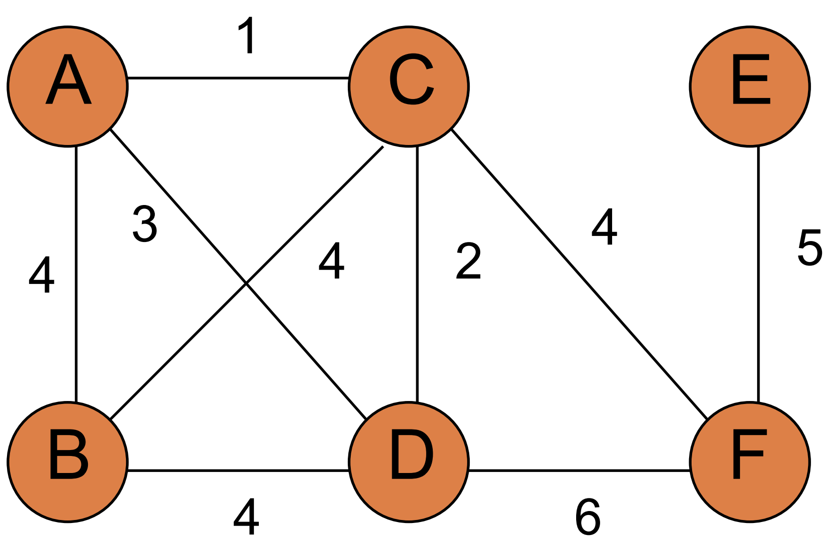


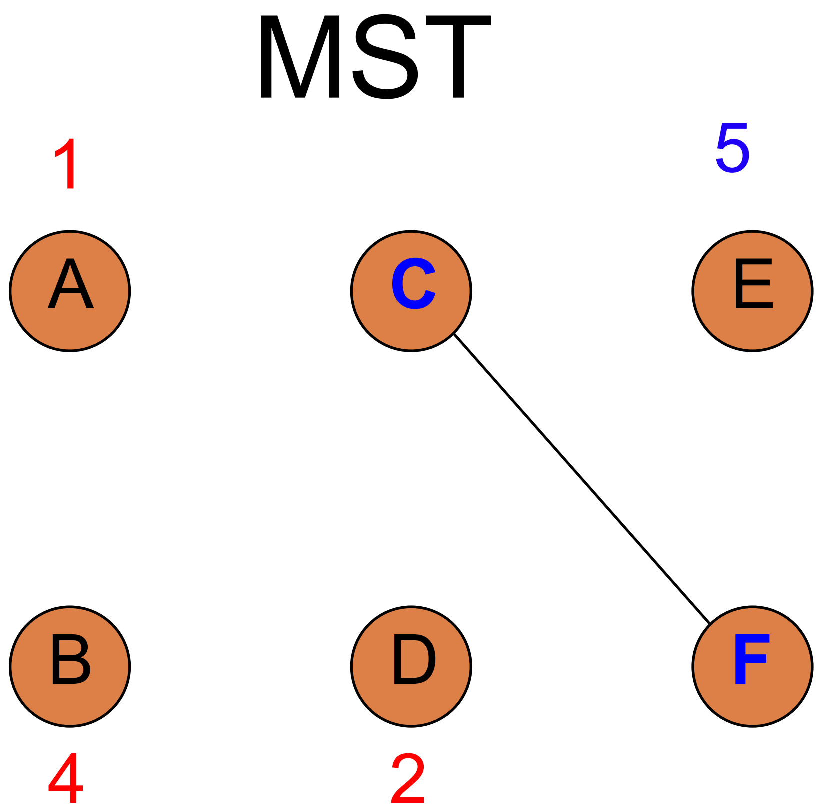
MST



Prim's

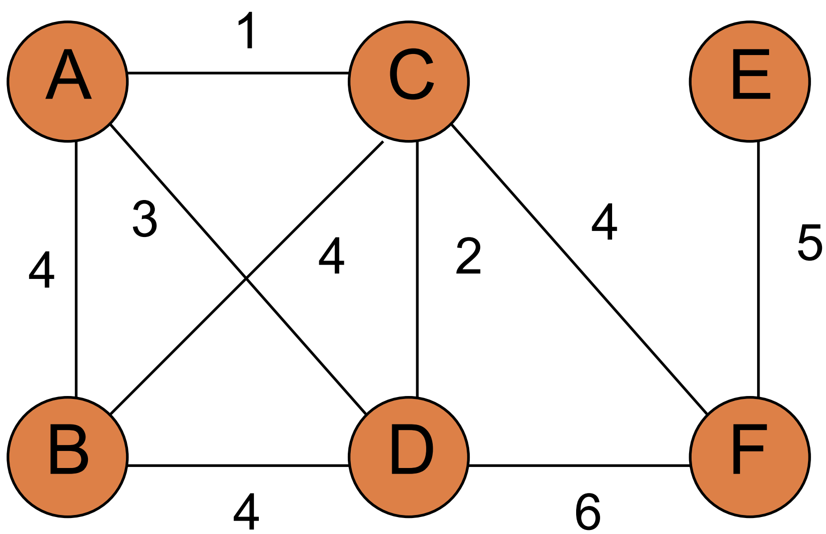
Add the edge with the smallest weight that connects a vertex not connected by the MST

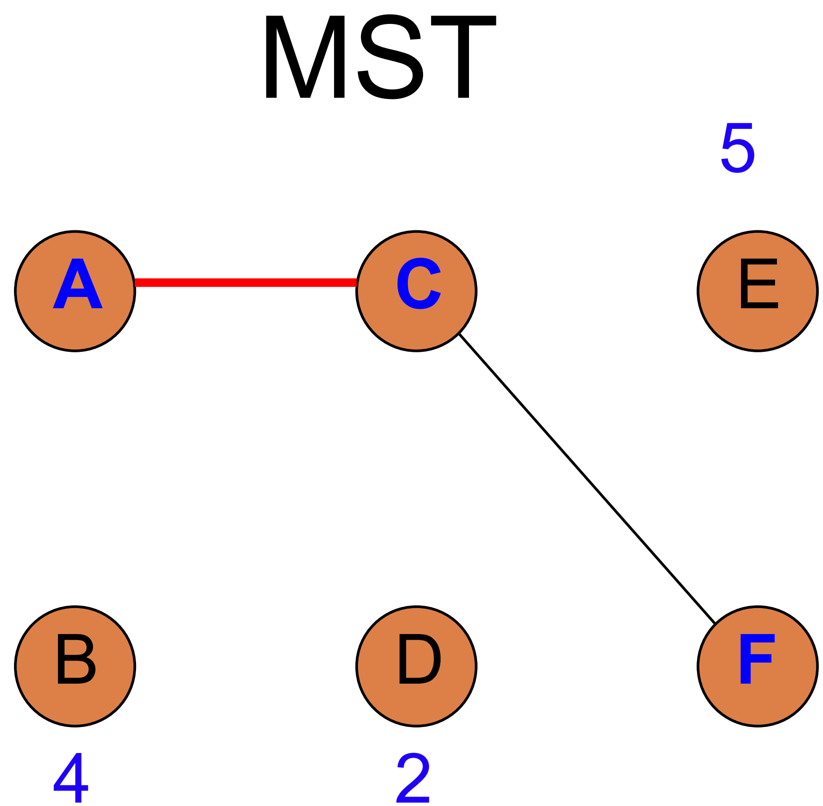




Prim's

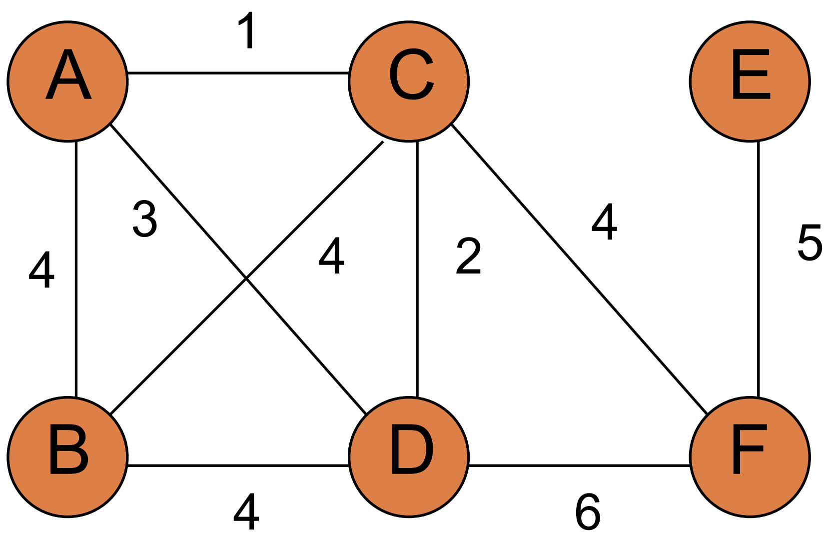
Add the edge with the smallest weight that connects a vertex not connected by the MST



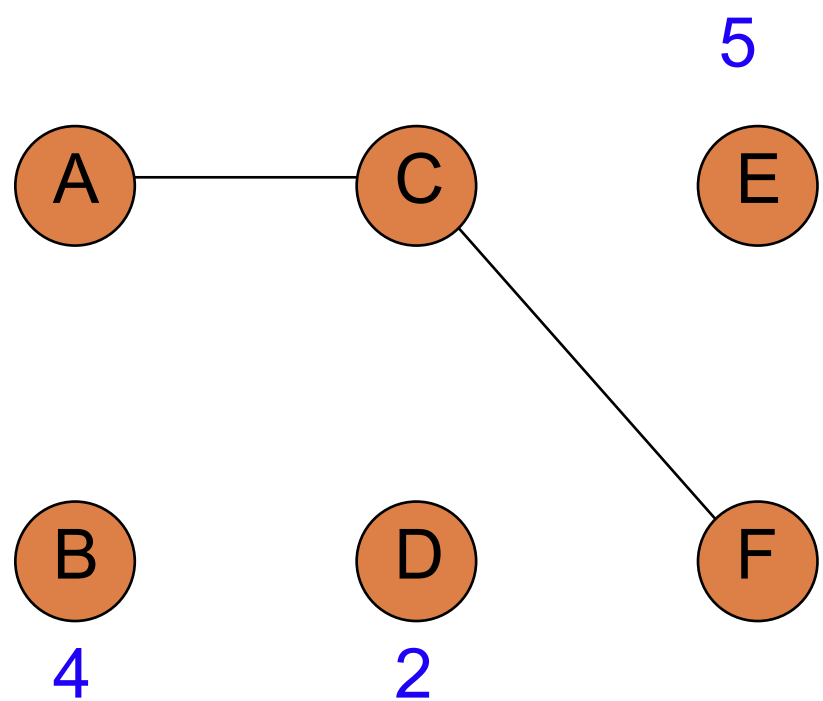


Prim's

Add the edge with the smallest weight that connects a vertex not connected by the MST

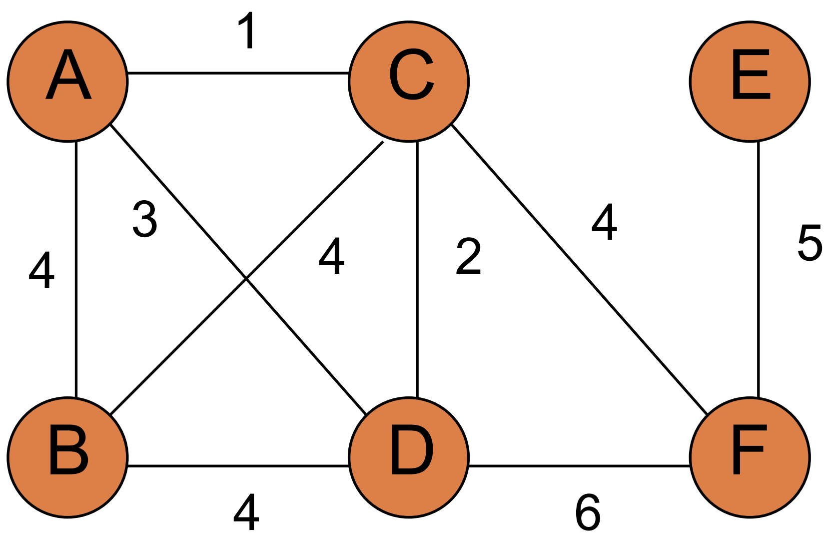


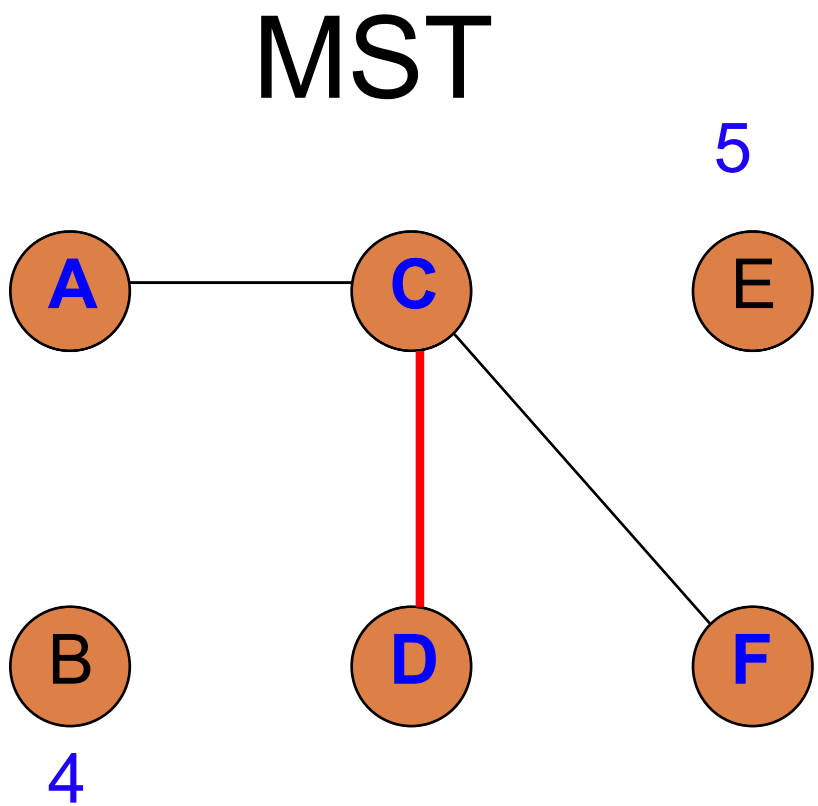
MST



Prim's

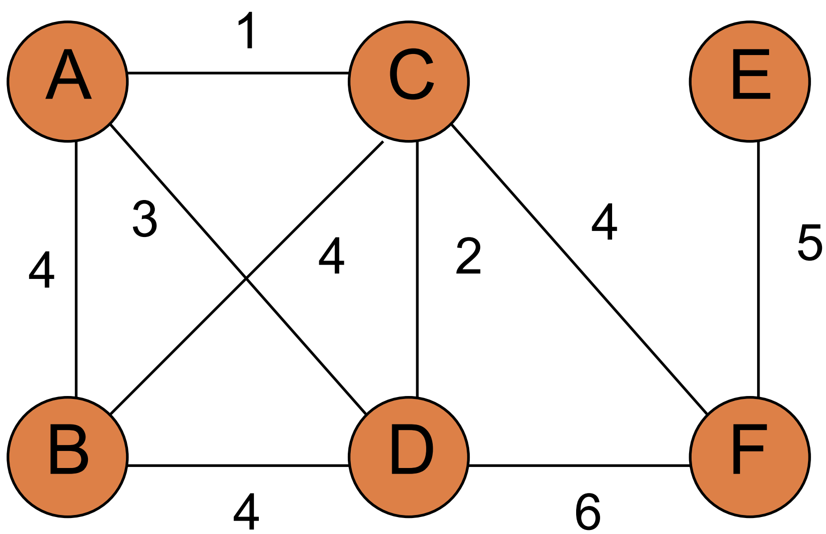
Add the edge with the smallest weight that connects a vertex not connected by the MST



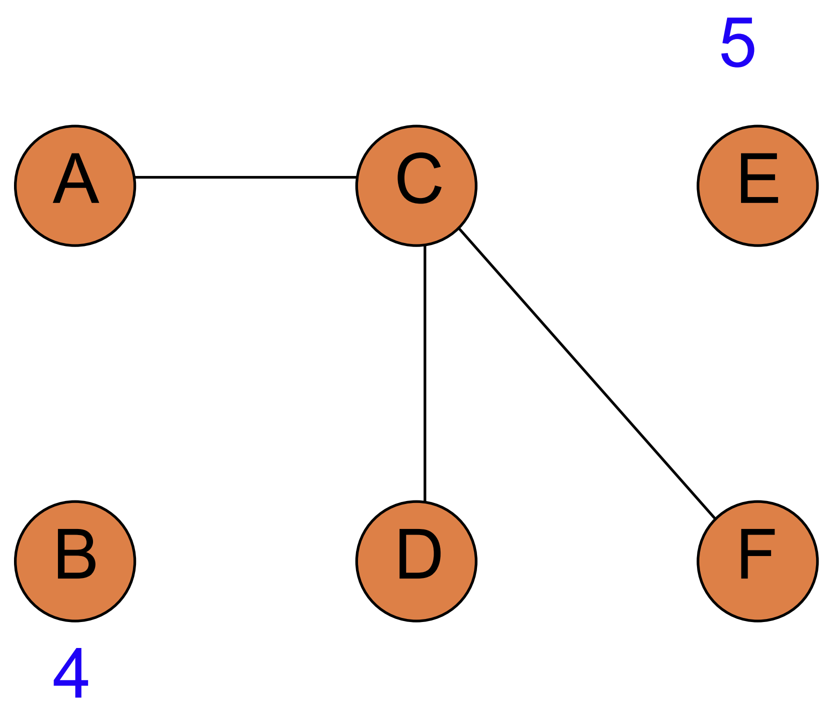


Prim's

Add the edge with the smallest weight that connects a vertex not connected by the MST

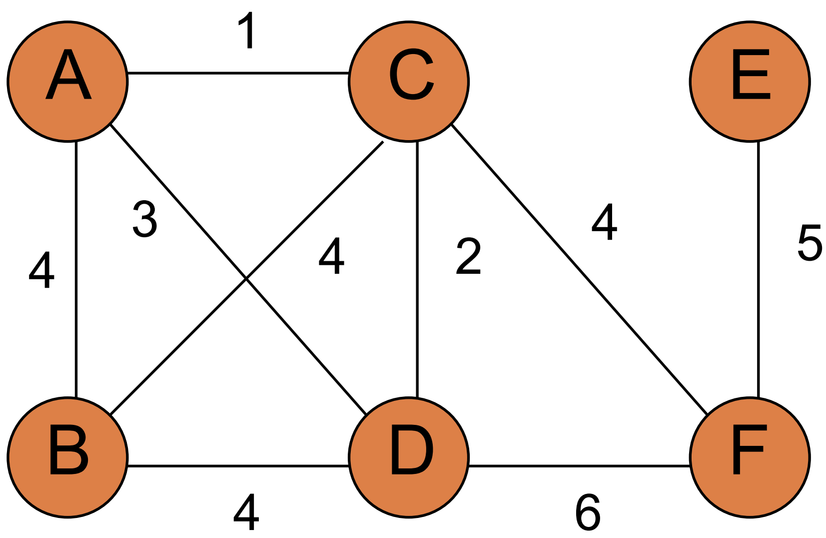


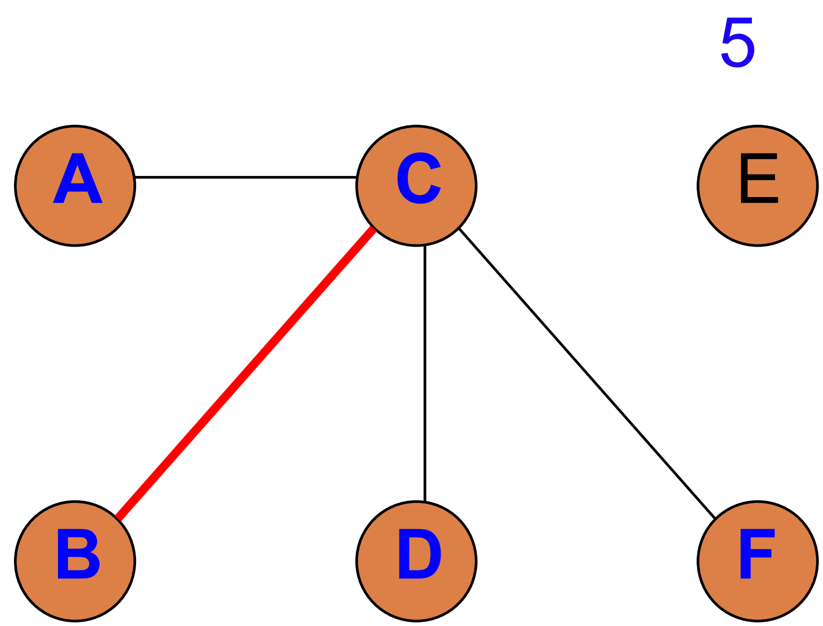
MST



Prim's

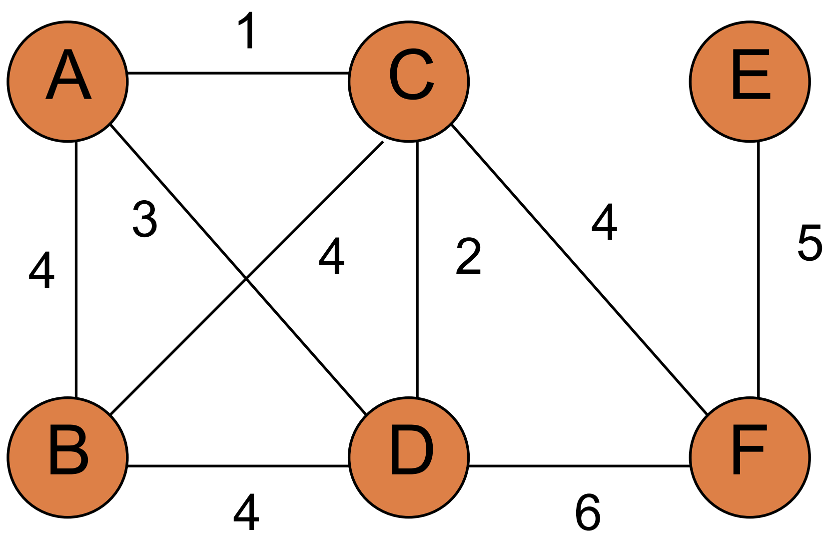
Add the edge with the smallest weight that connects a vertex not connected by the MST

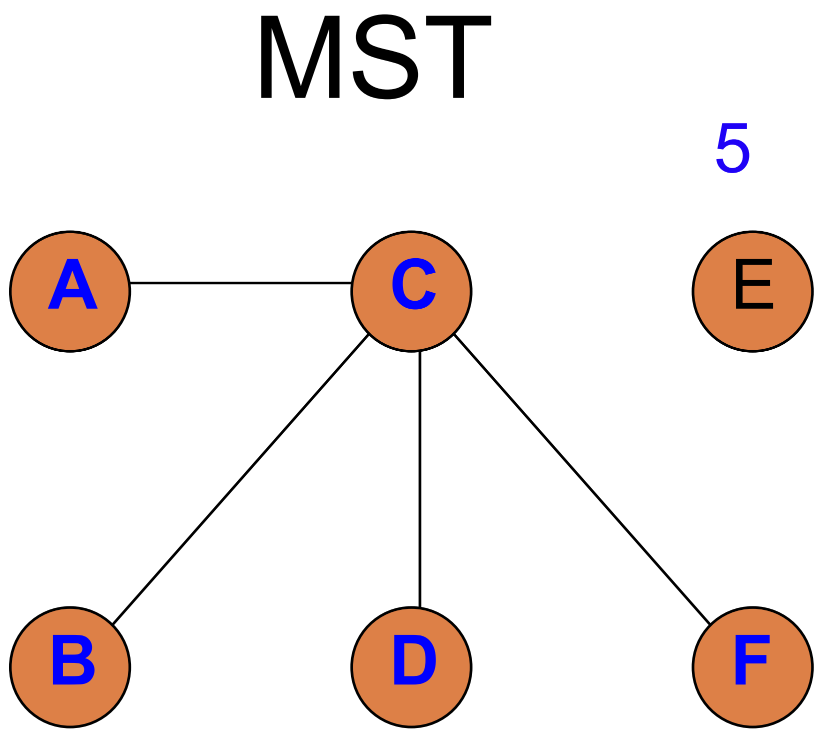




Prim's

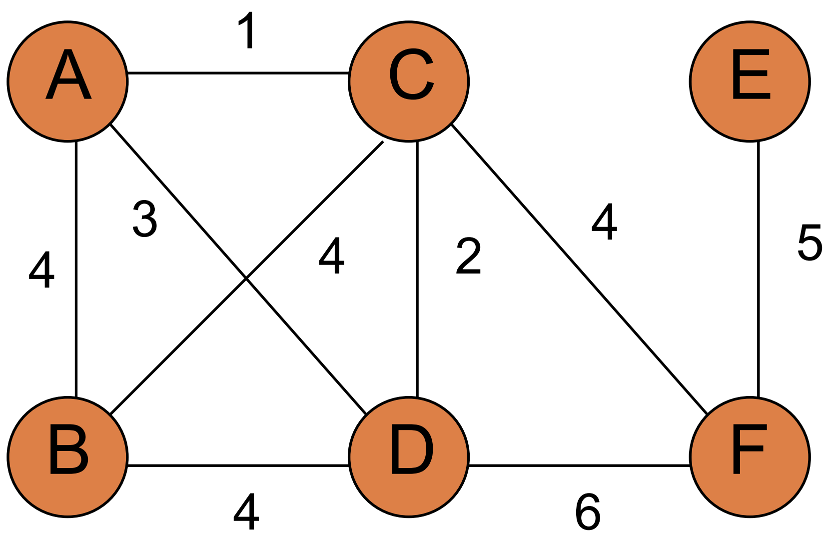
Add the edge with the smallest weight that connects a vertex not connected by the MST



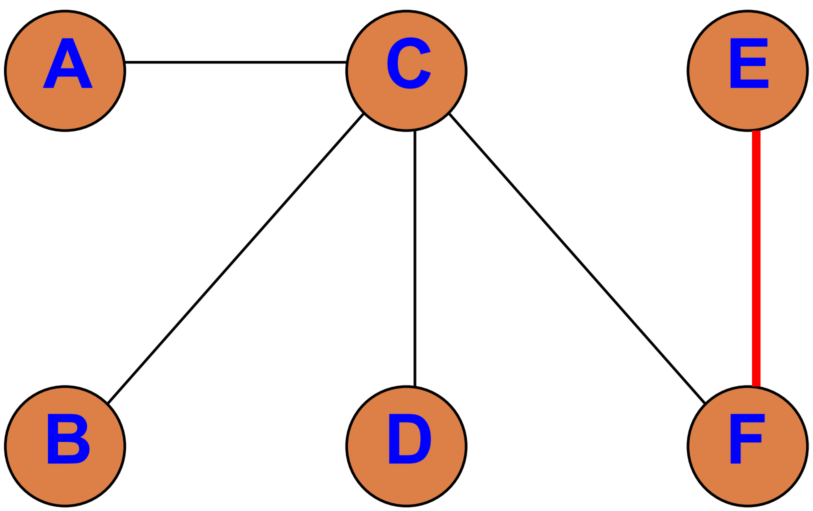


Prim's

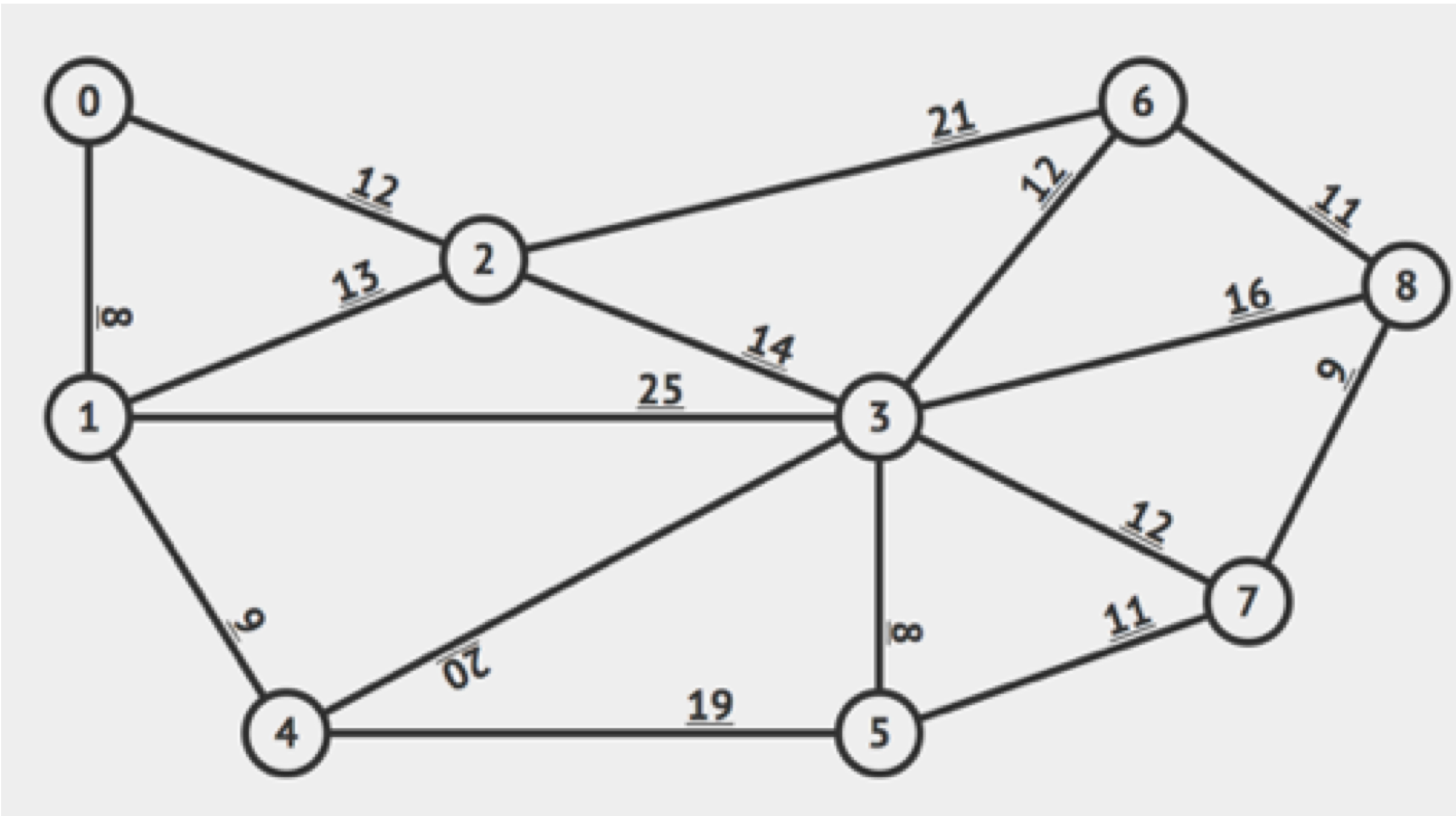
Add the edge with the smallest weight that connects a vertex not connected by the MST



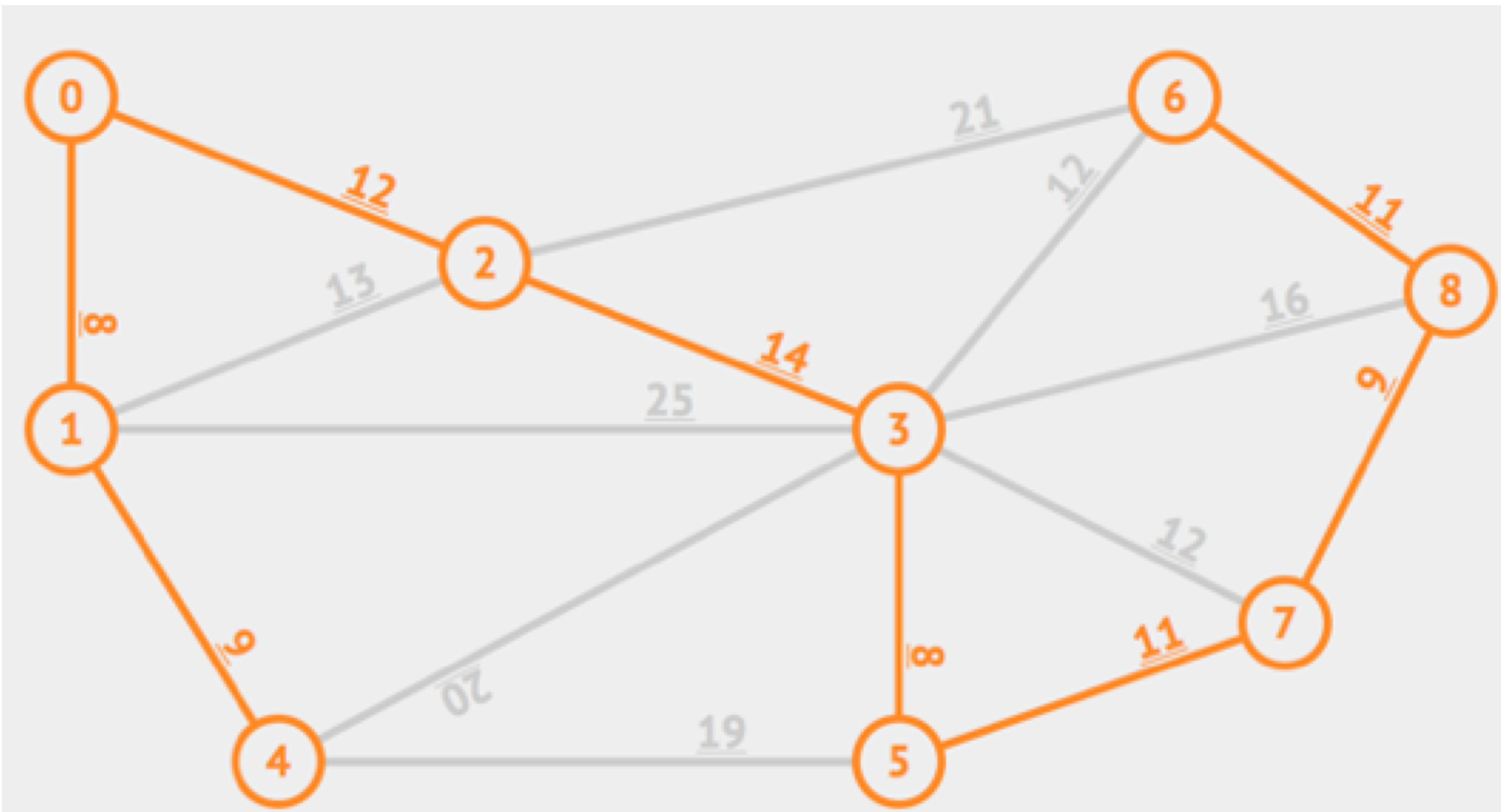
MST



Practice: start at vertex 0



Solution



Sum = 8 + 8 + 9 + 9 + 11 + 11 + 12 + 14 = 82

Why does Prim's work?

Given a partition S, let edge e be the minimum cost edge that **crosses** the partition. *Every* minimum spanning tree contains edge e.

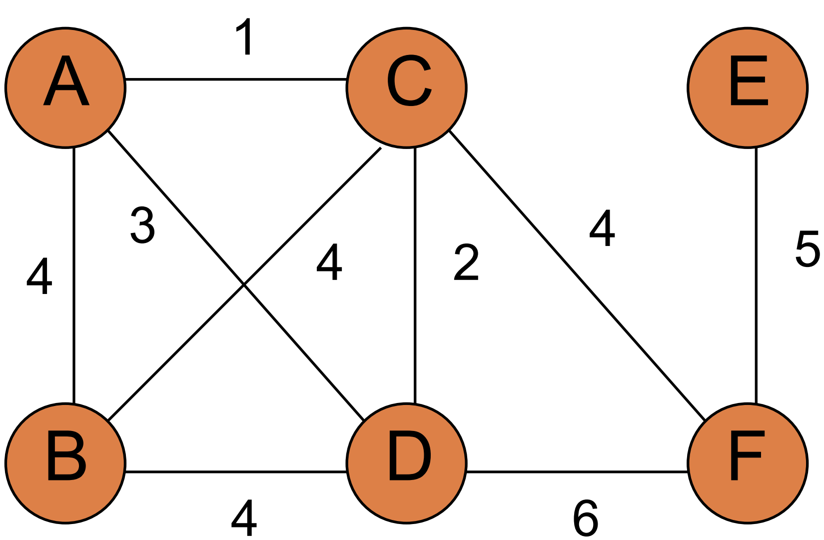
Let S be the set of vertices visited so far

The only time we add a new edge is if it's the lowest weight edge from S to V-S

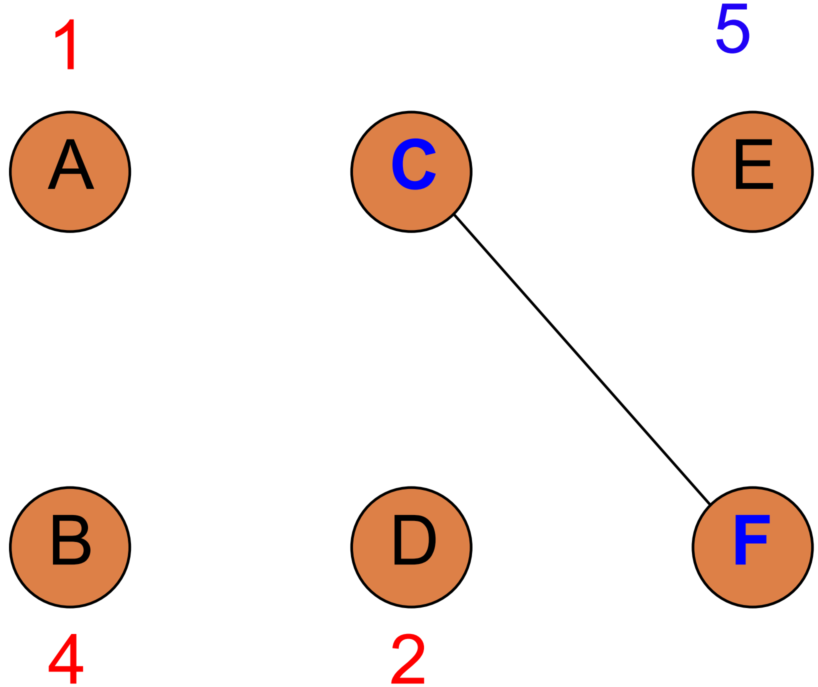
Prim's

Add the edge with the smallest weight that connects a vertex not connected by the MST

How do we find the smallest weight edge? Or, how could we keep track of it?



MST

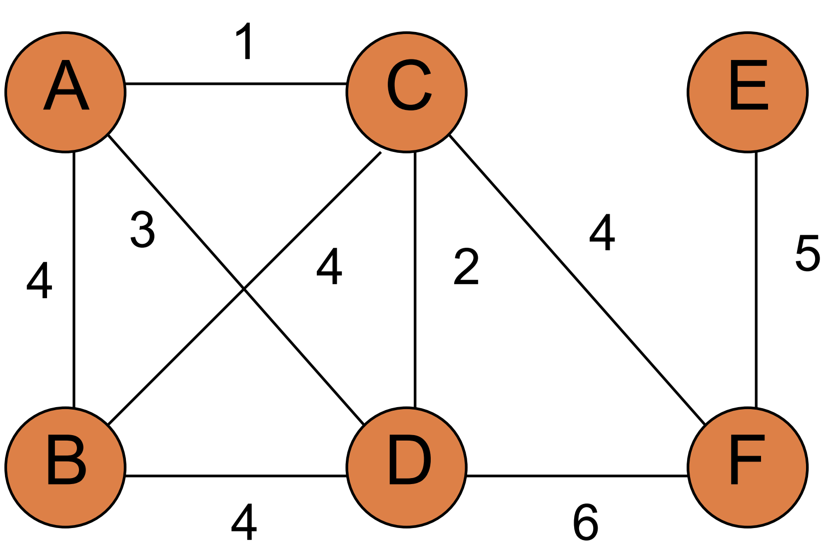


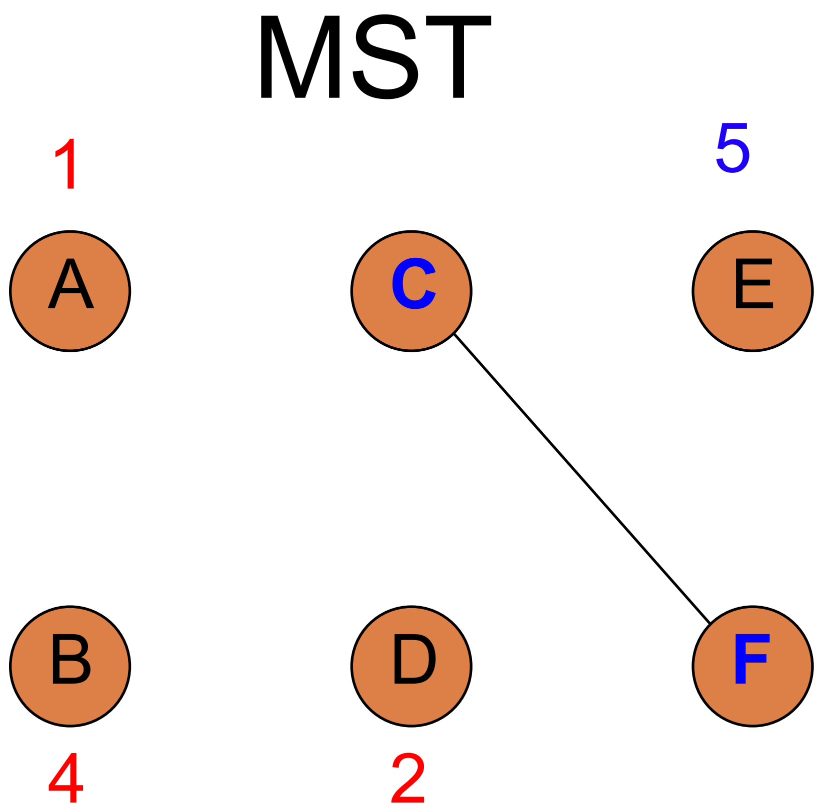
Prim's

Add the edge with the smallest weight that connects a vertex not connected by the MST

Very similar implementation to Dijksra's!

Use a priority queue





Running time of Prim's

Varies depending on the priority queue implementation

Practical version: O(E log V)