Working with Spanning Tree Algorithms

Janani Ravi

CO-FOUNDER, LOONYCORN

www.loonycorn.com

Three Common Graph Problems

Establishing precedence

Getting from point A to point B

Covering all nodes in a graph

Three Common Graph Problems

Establishing precedence

Topological sort

Getting from point A to point B

Shortest path algorithms

Covering all nodes in a graph

Minimum spanning tree algorithms

Three Common Graph Operations

Topological sort

Computation graphs in neural networks

Shortest path

Deliveries from warehouses to customers

Minimum spanning tree

Planning railway lines

Three Common Graph Operations

Topological sort

Computation graphs in neural networks

Shortest path

Deliveries from warehouses to customers

Minimum spanning tree

Planning railway lines

Overview

Spanning tree algorithms seek to find the shortest way to cover all nodes

Such algorithms are used when start and end nodes do not matter

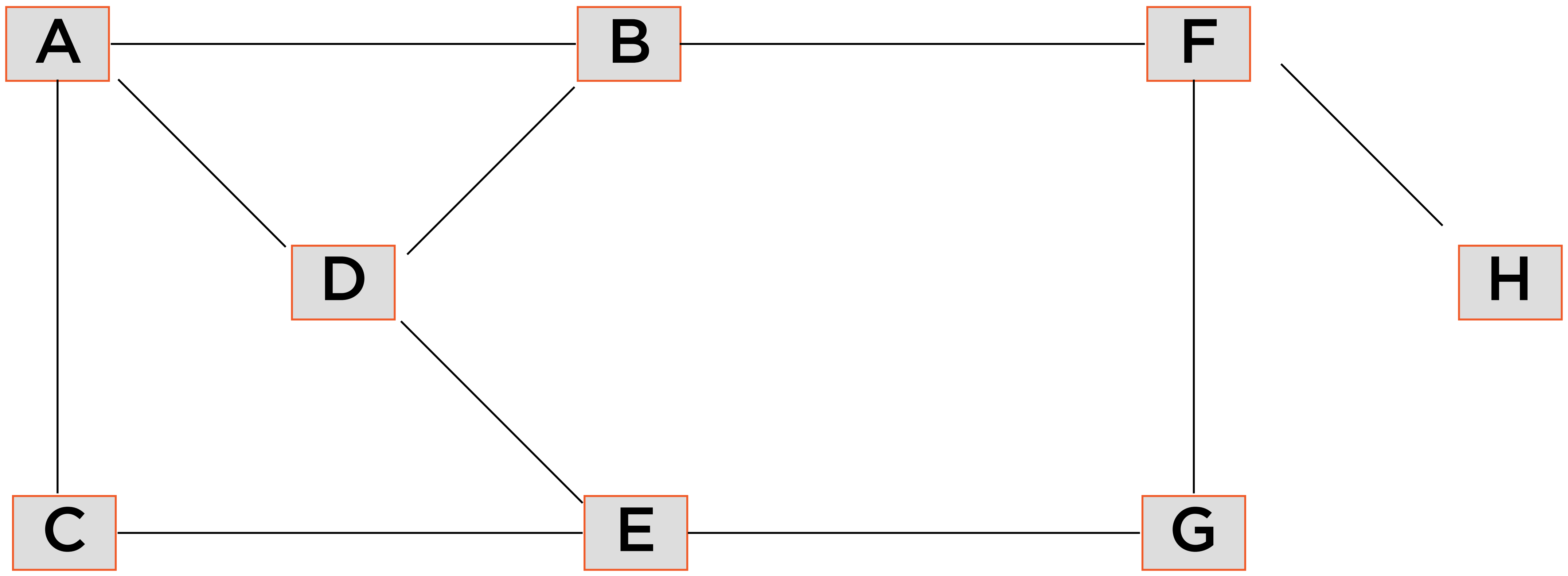
Prim's algorithm works for connected graphs

Kruskal's algorithm works even for disconnected graphs

Graph (V,E)

A set of vertices (V) and edges (E)

An Undirected Graph

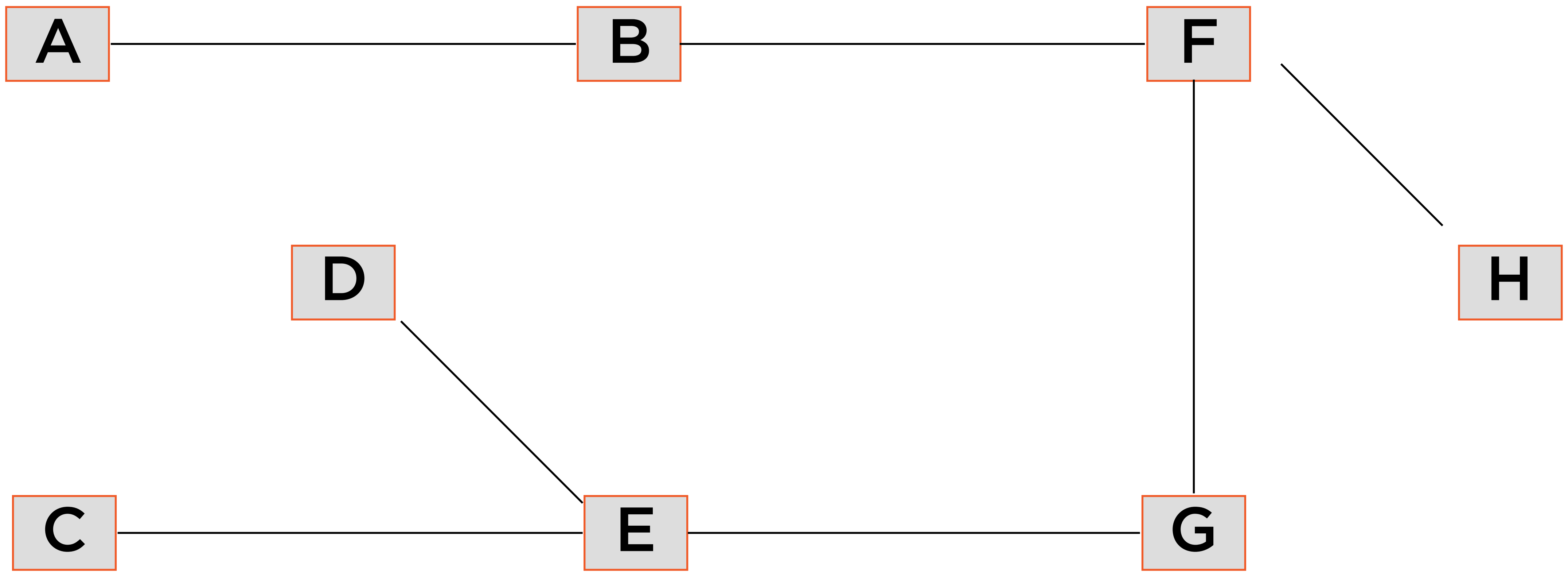


V = {A, B, C, D, E, F, G, H}

Tree

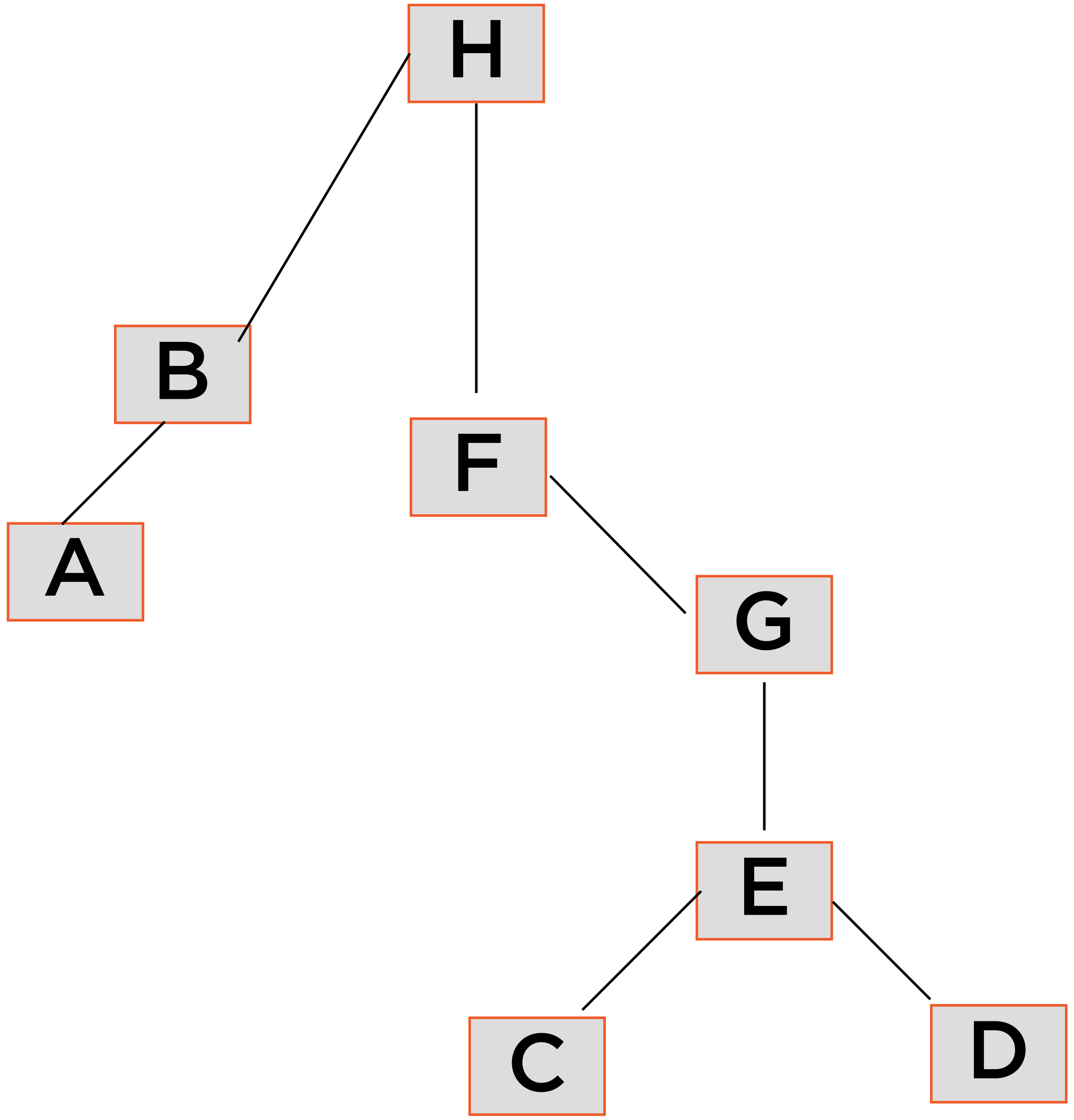
A connected graph with no cycles

Connected Graph with no Cycle



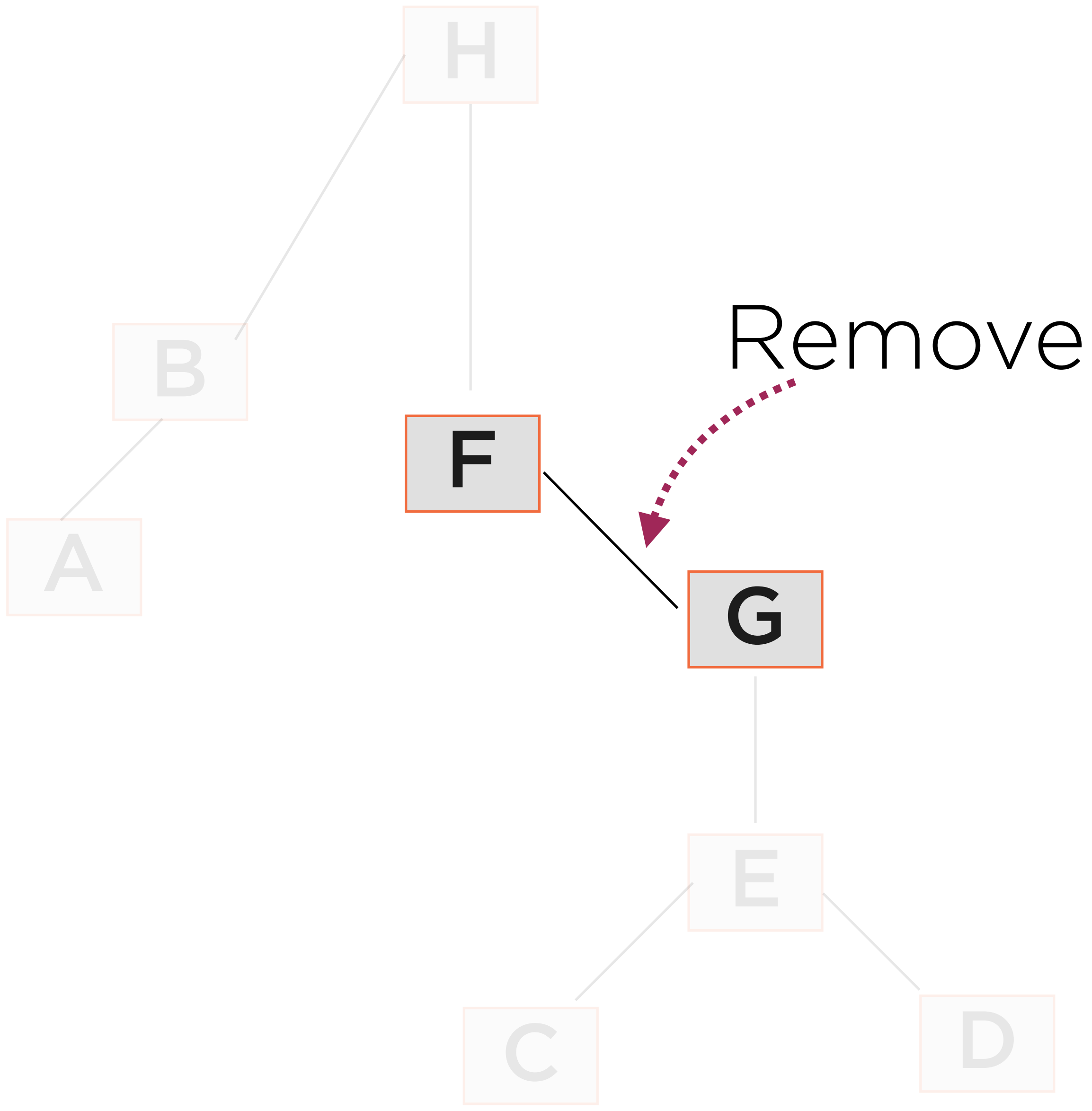
Such a graph is called a tree

Forest: Set of Disjoint Trees



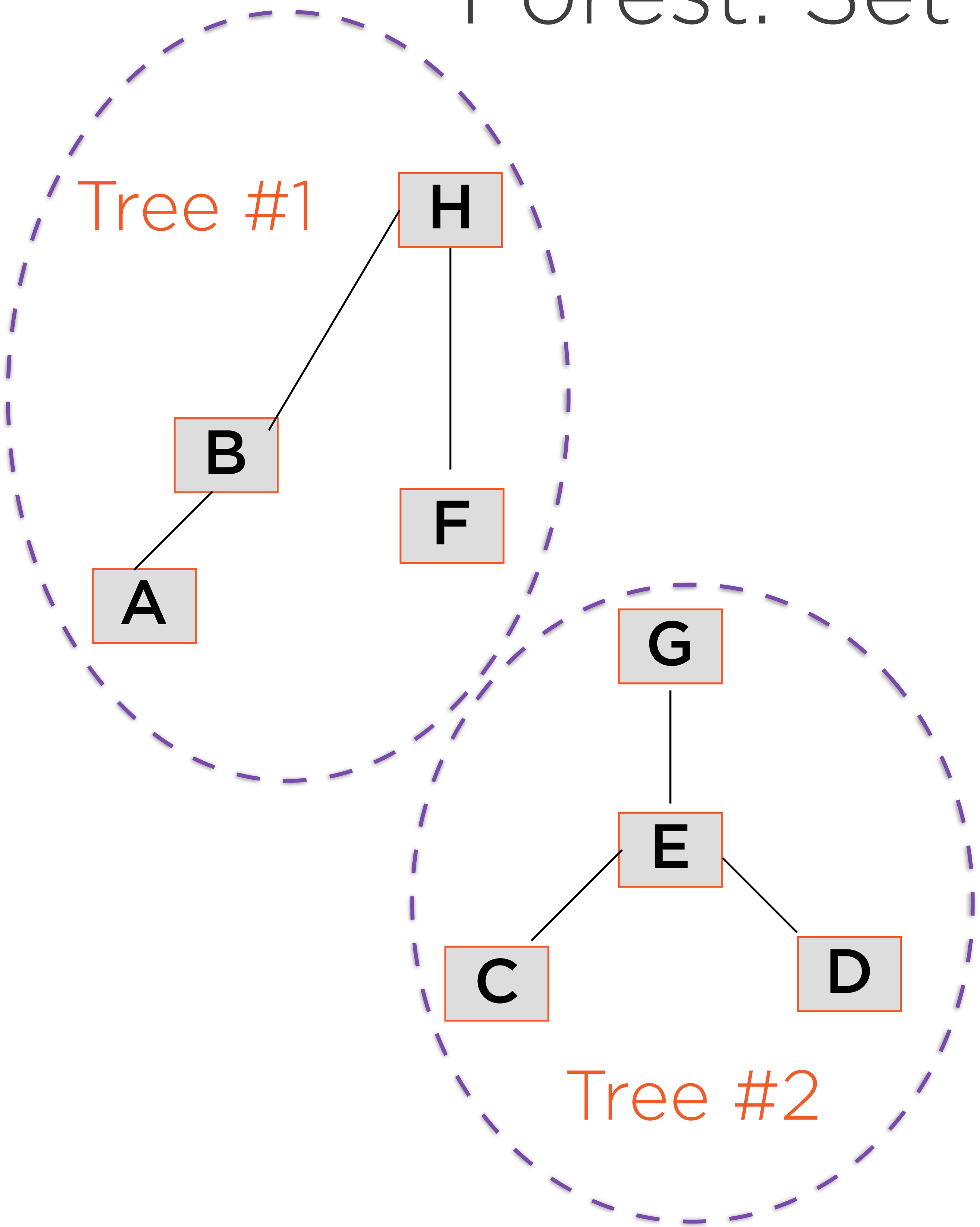
Trees are great for depicting hierarchical relationships

Forest: Set of Disjoint Trees



Removing F - G divides the original graph into two disjoint graphs

Forest: Set of Disjoint Trees



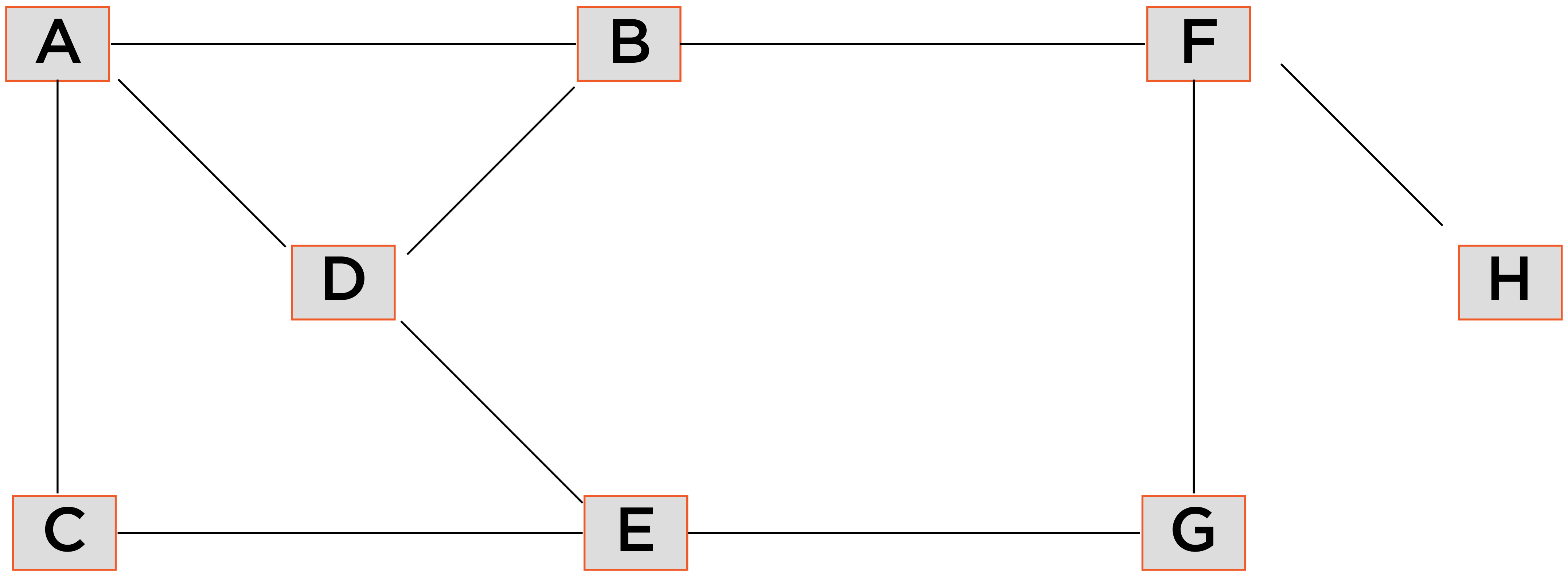
Such a set of disjoint trees is called a forest

Spanning Tree of a Graph

Any tree that includes all of the vertices of the graph

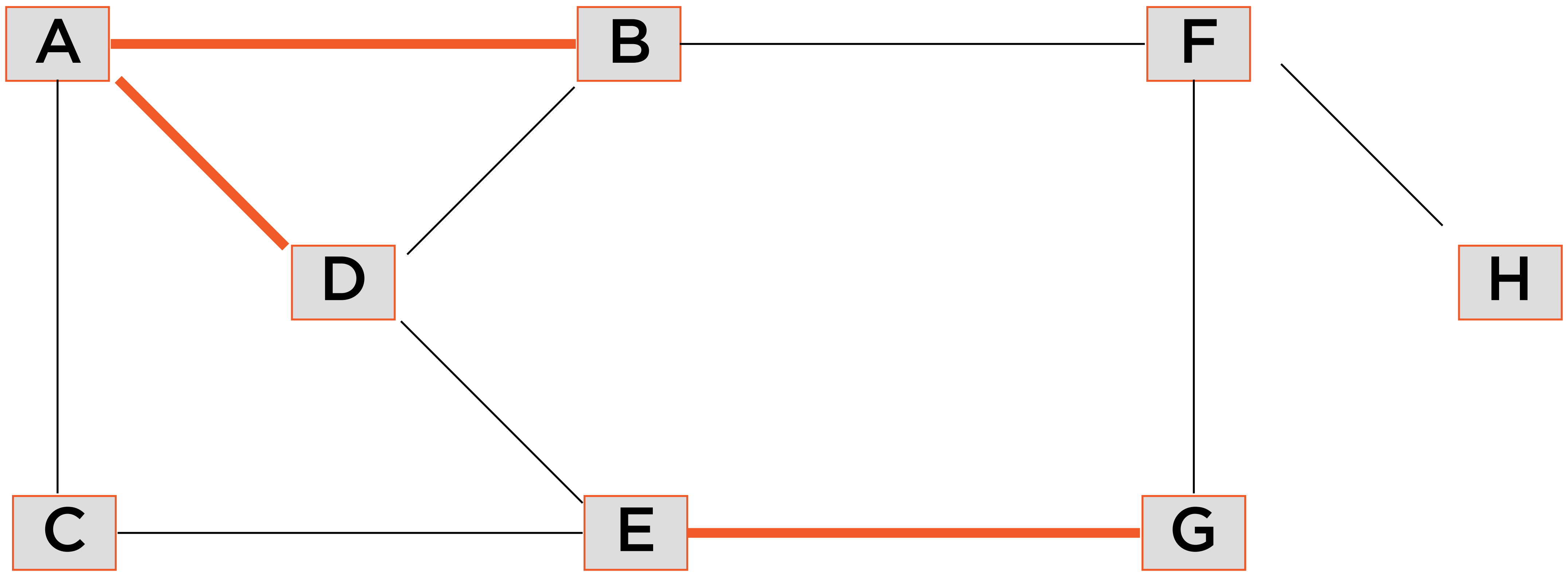
(Given a graph with N vertices, any spanning tree has N-1 edges)

An Undirected Graph



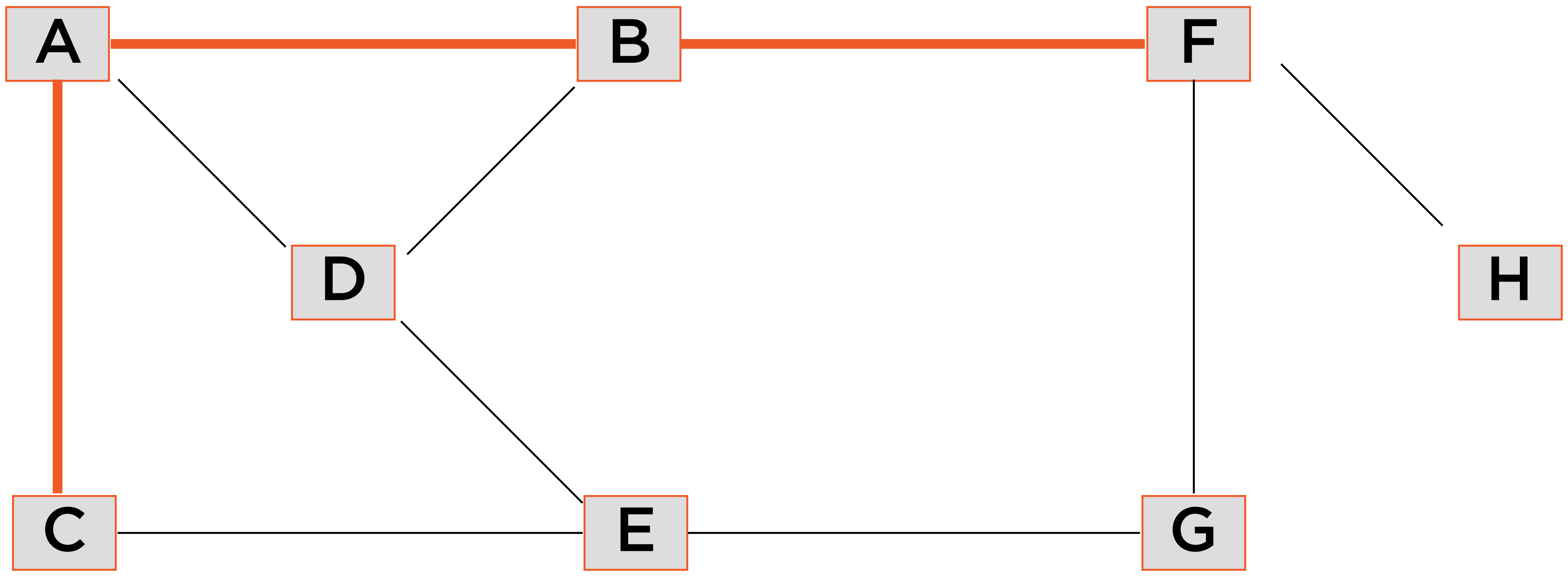
V = {A, B, C, D, E, F, G, H}

A Spanning Tree



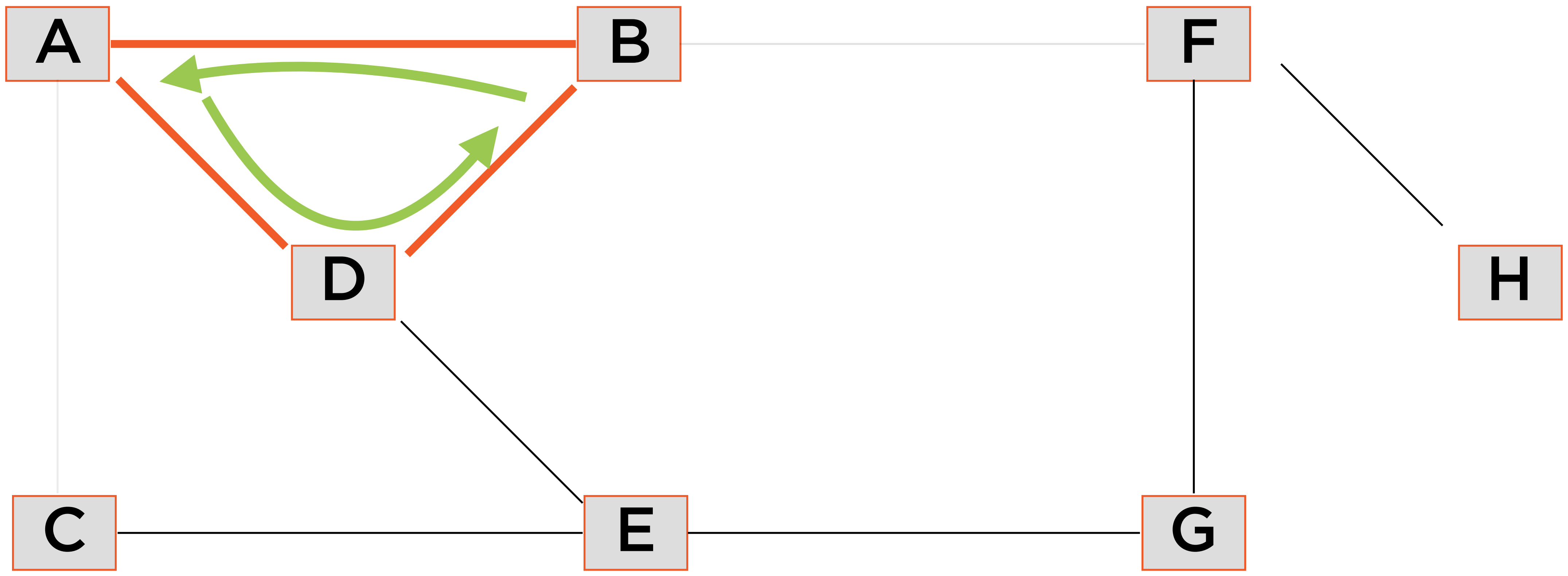
Eliminating edges A - B, A - D, E - G yields a spanning tree

Another Spanning Tree of Graph



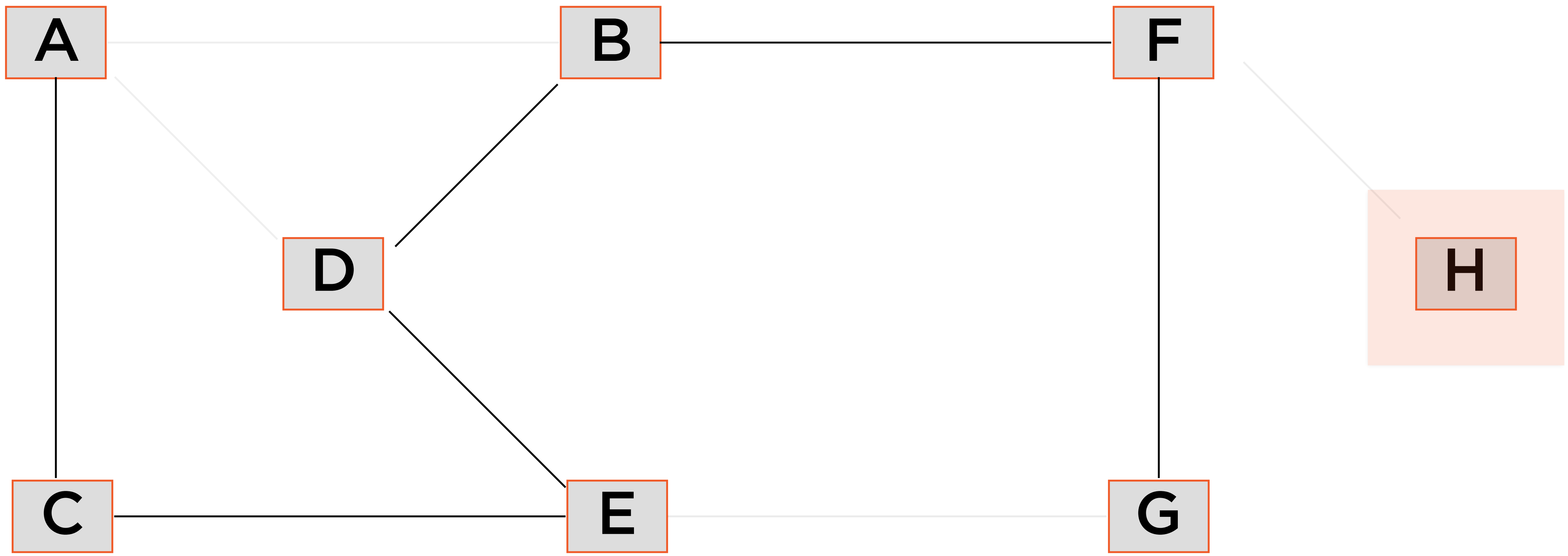
Eliminating edges A - C, A - B, B - F yields a different spanning tree

Not a Spanning Tree



This is not a spanning tree, because A - B - D - A forms a cycle

Not a Spanning Tree



This is not a spanning tree, because node H is not included in the tree

Minimum Spanning Tree of a Graph

Spanning tree with the lowest weight

Prim’s Algorithm

Two Minimum Spanning Tree Algorithms

Prim's Algorithm

Works with connected graphs

Kruskal's Algorithm

Works even with disconnected graphs

Two Minimum Spanning Tree Algorithms

Prim's Algorithm

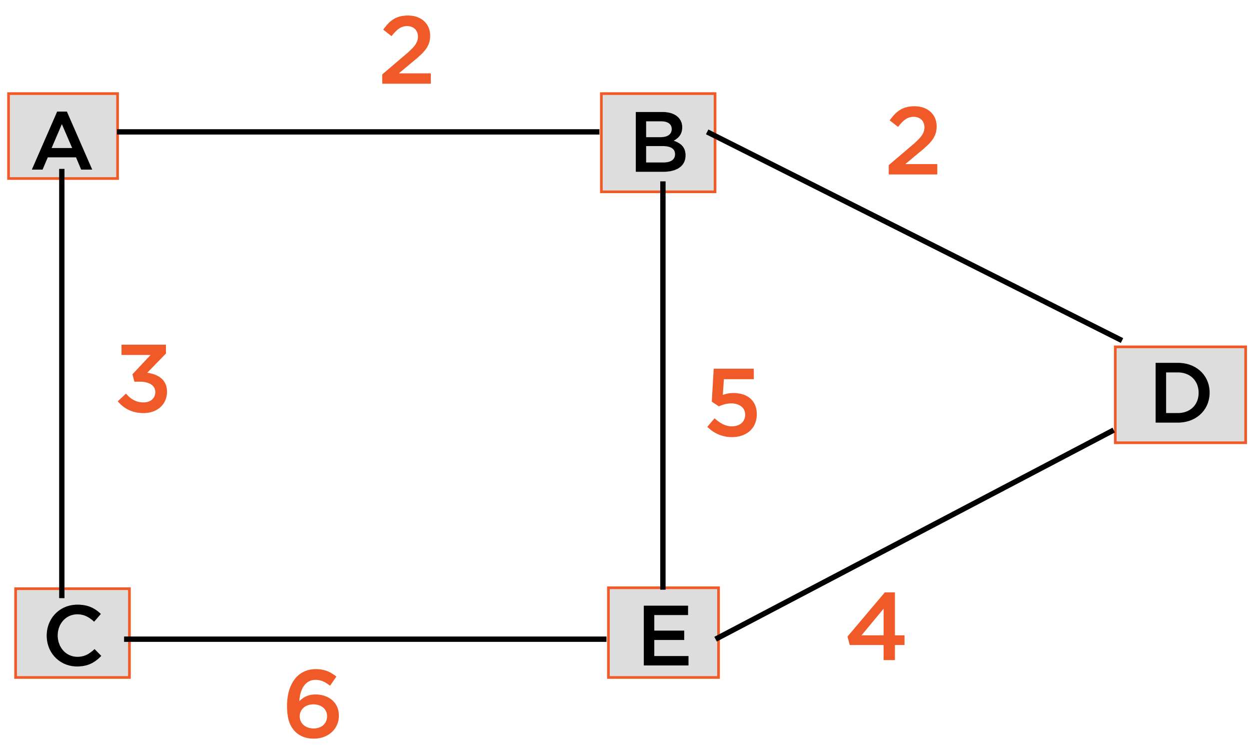
Works with connected graphs

Kruskal's Algorithm

Works even with disconnected graphs

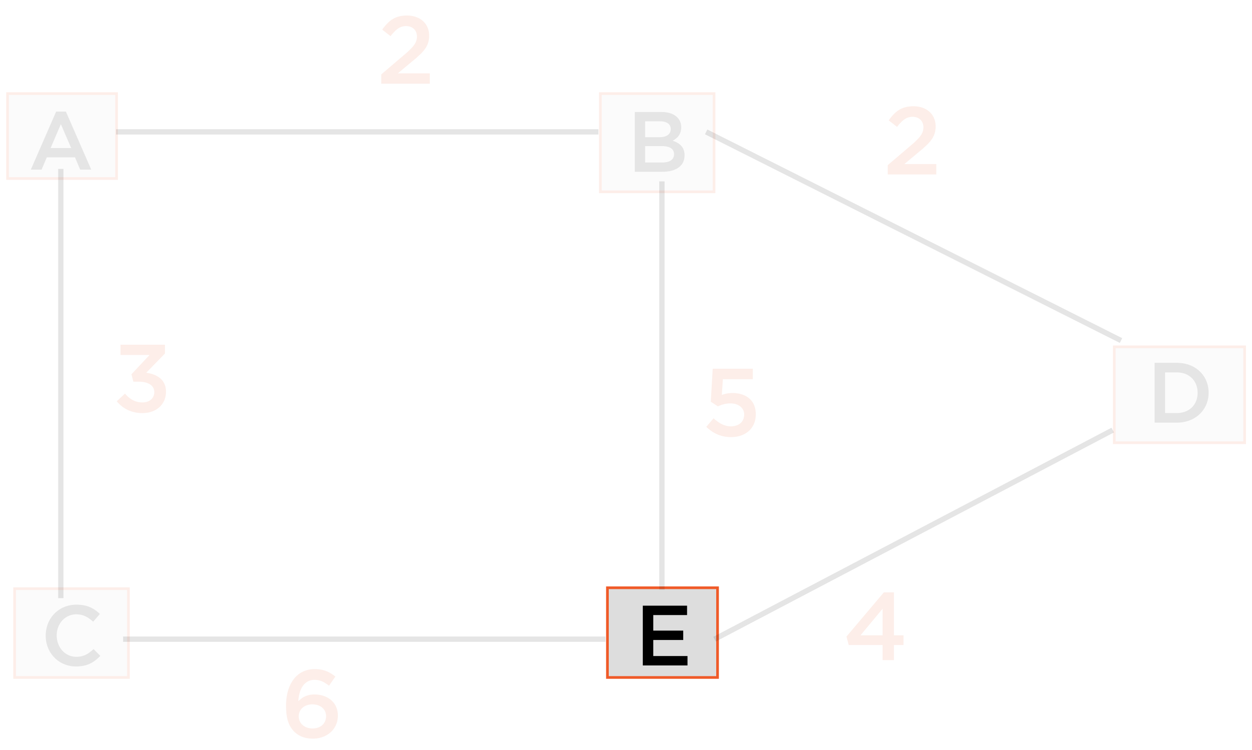
Prim’s algorithm is a **greedy** algorithm to find a minimal spanning tree for a **weighted** **undirected** graph

Prim's Algorithm



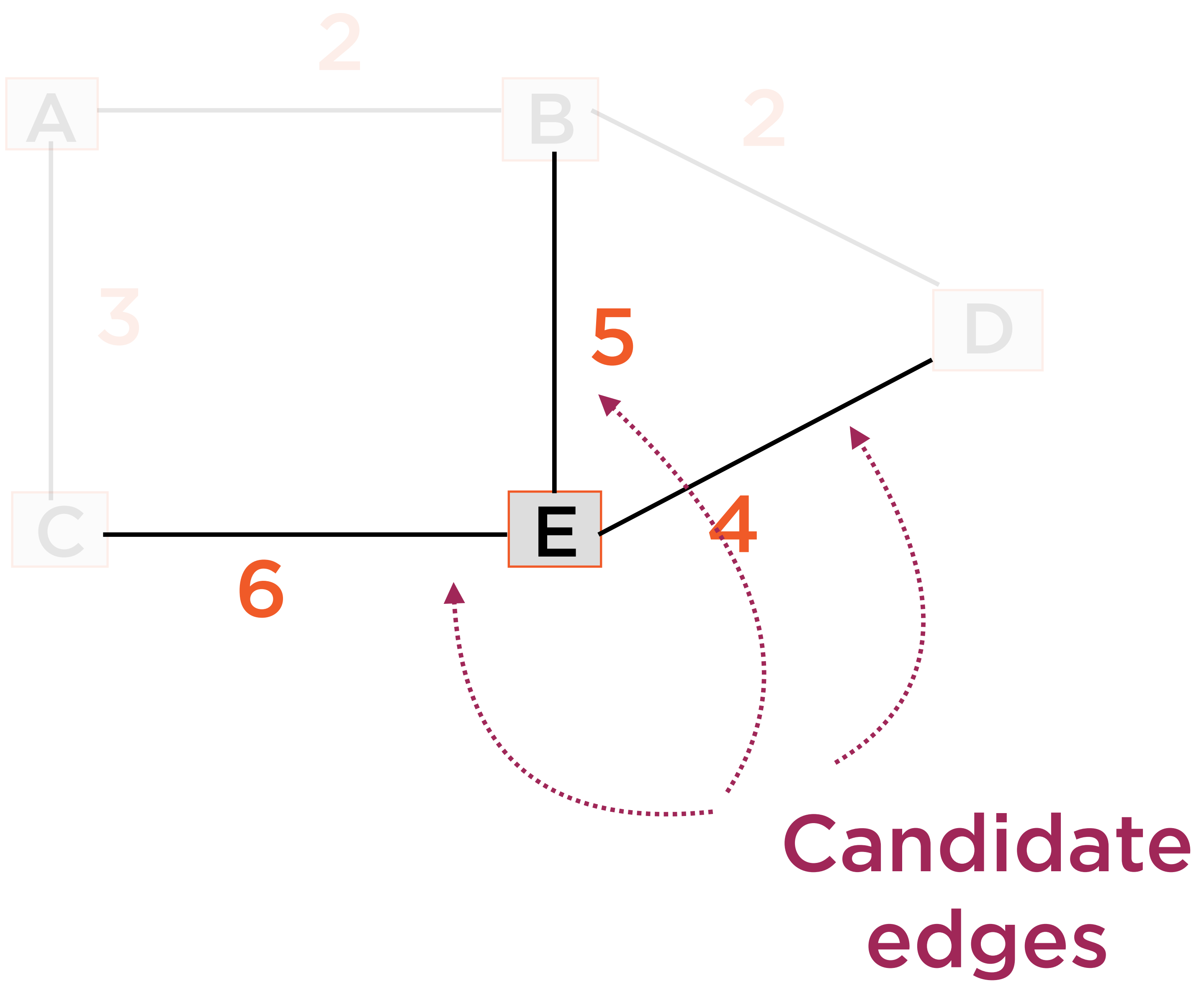
Start anywhere, pick a node at random

Prim’s Algorithm



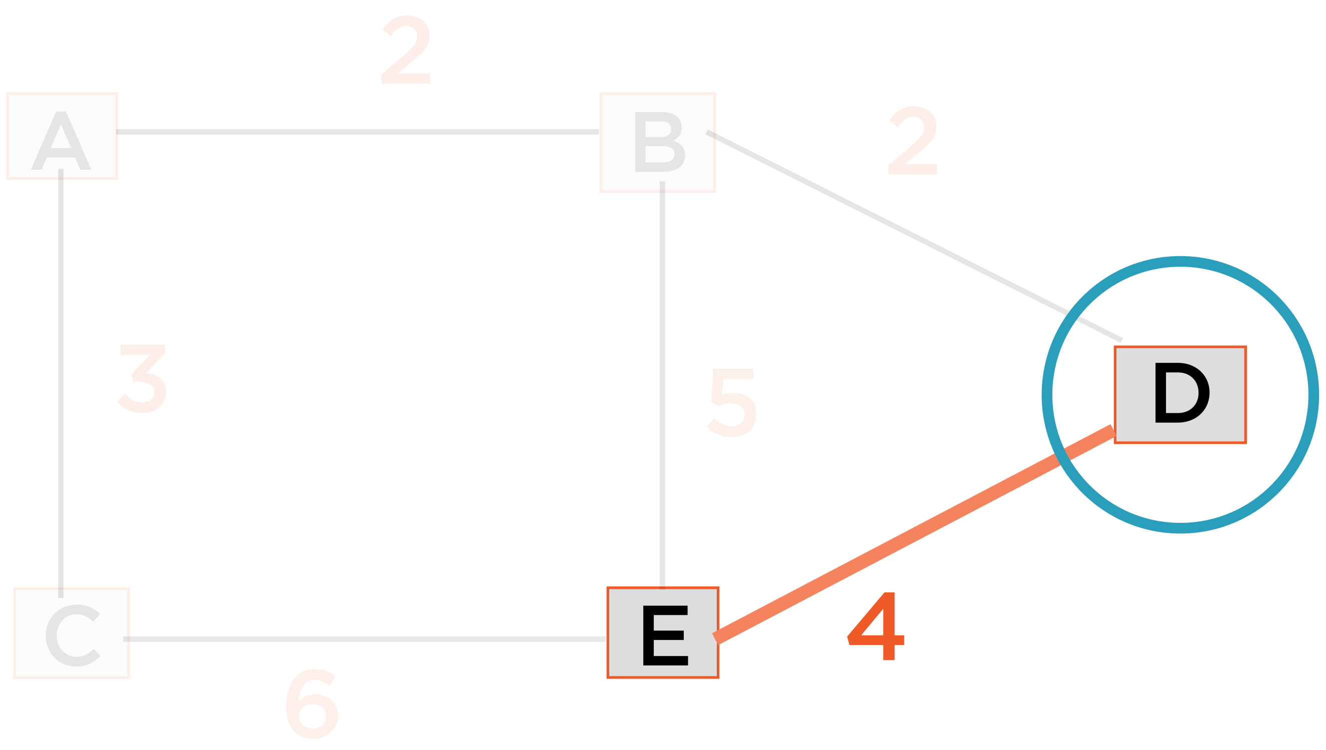
Start anywhere, pick a node at random

Prim's Algorithm



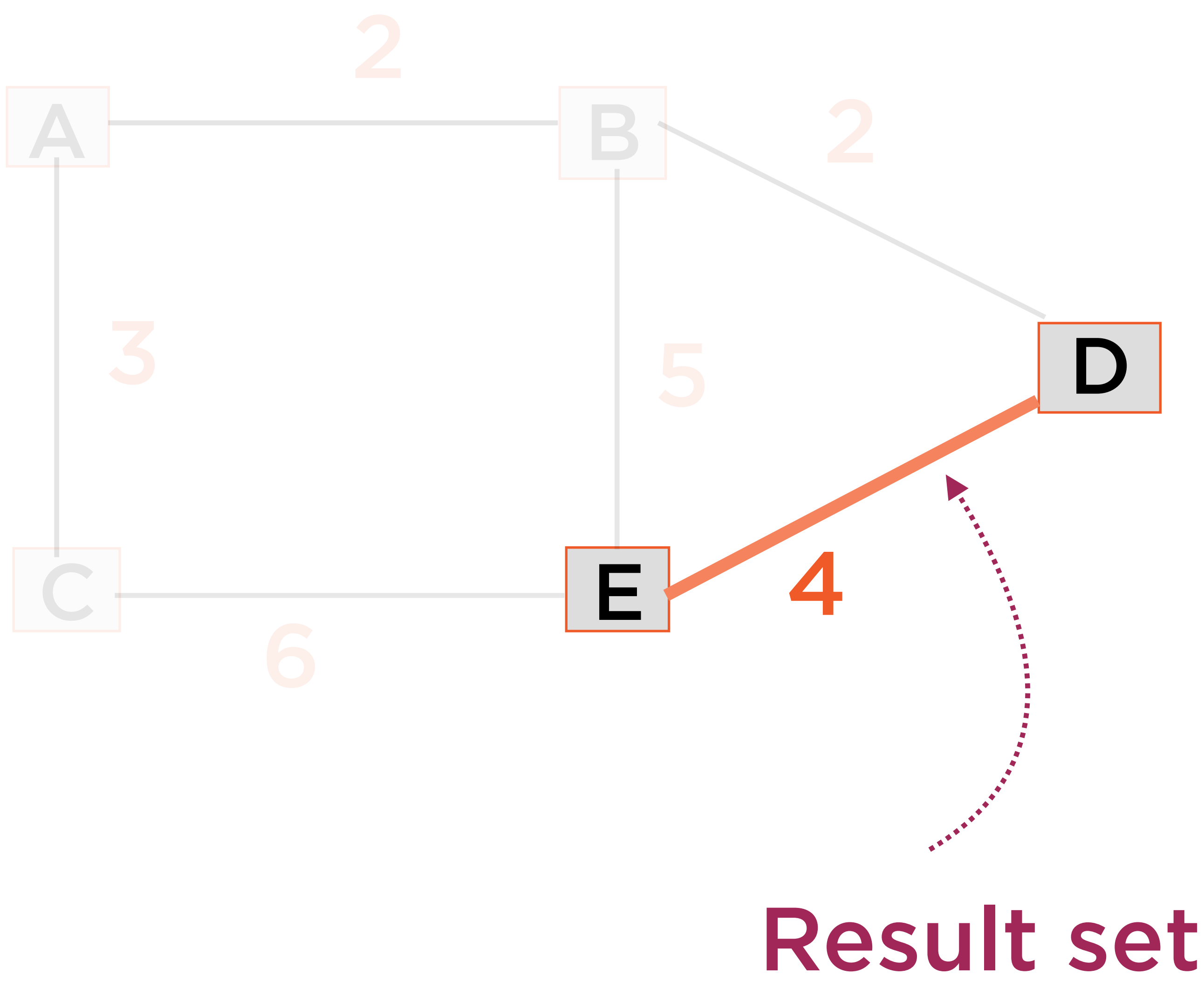
Find the lowest weight edge out of that node

Prim’s Algorithm



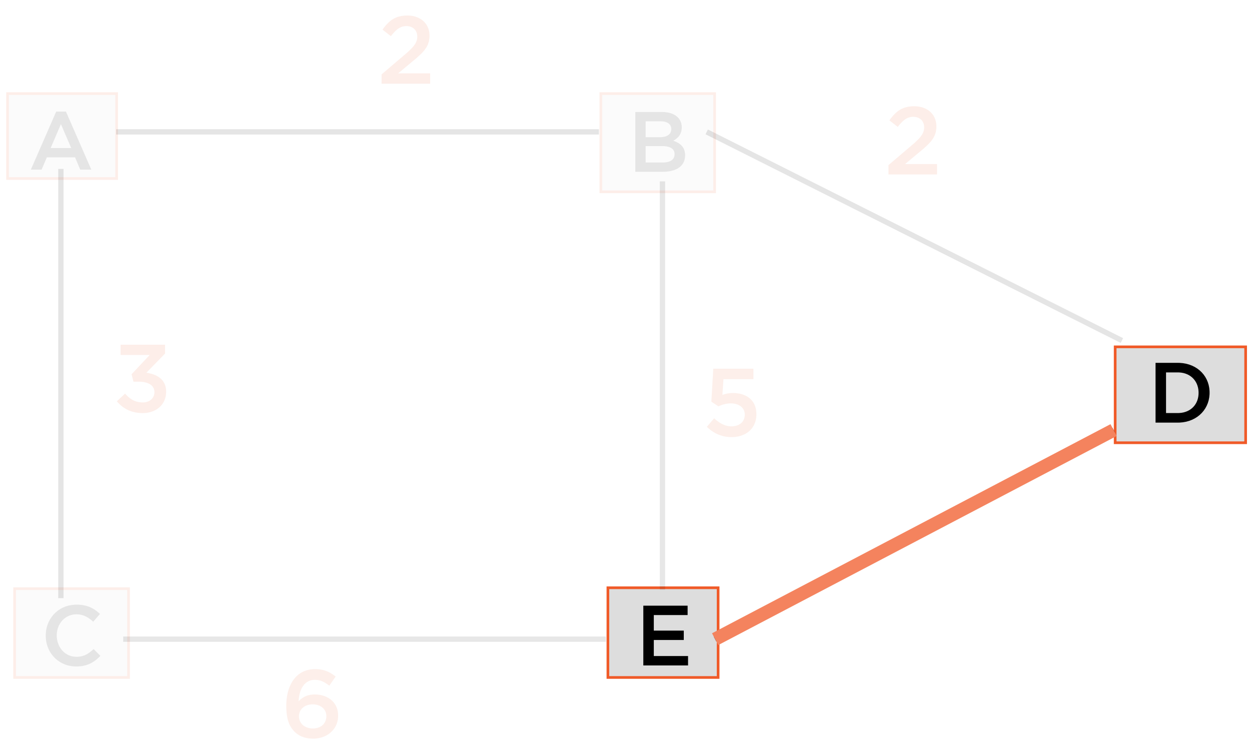
Lowest weighted edge connecting an unvisited node

Prim's Algorithm



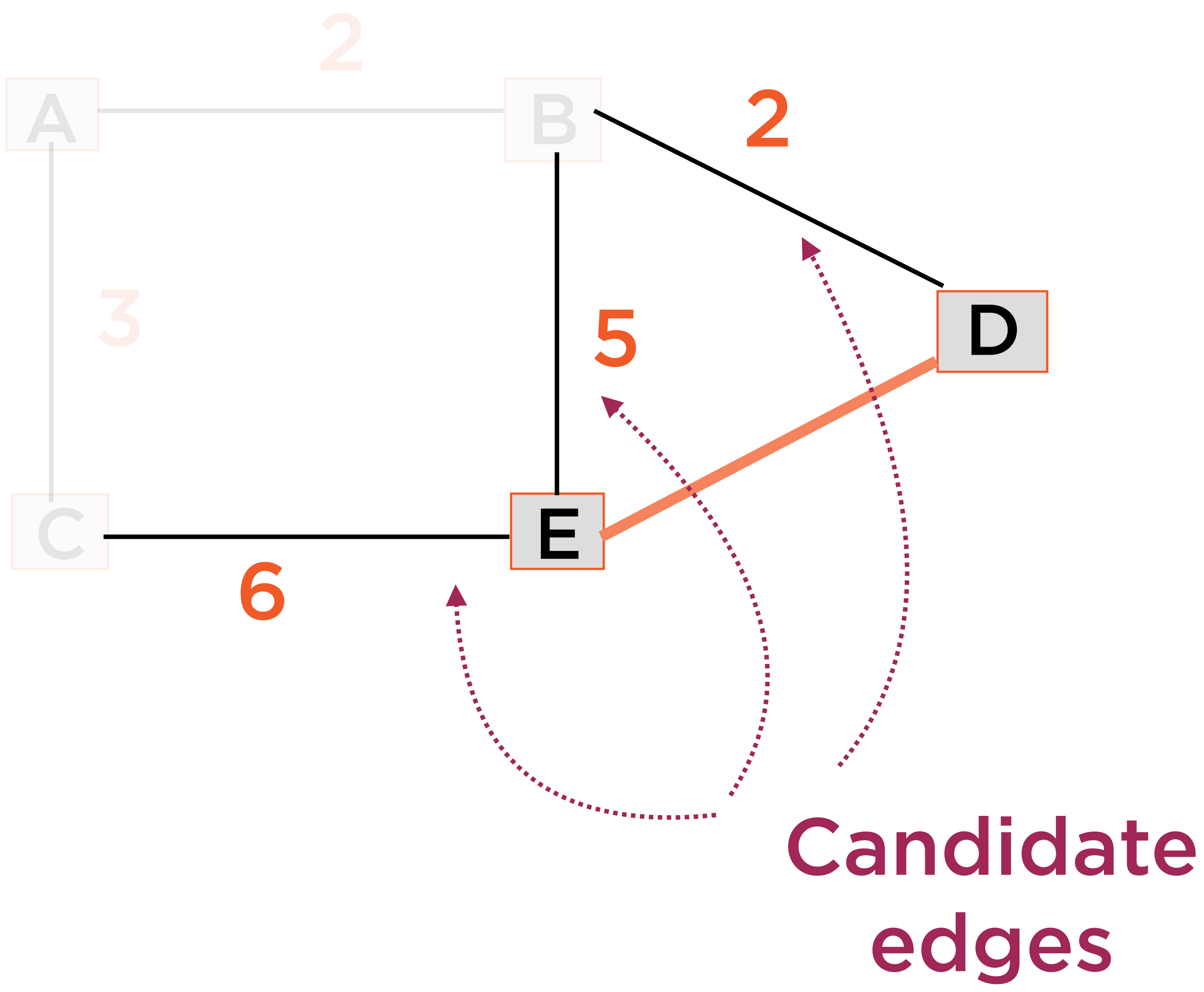
Add that edge to the result

Prim’s Algorithm



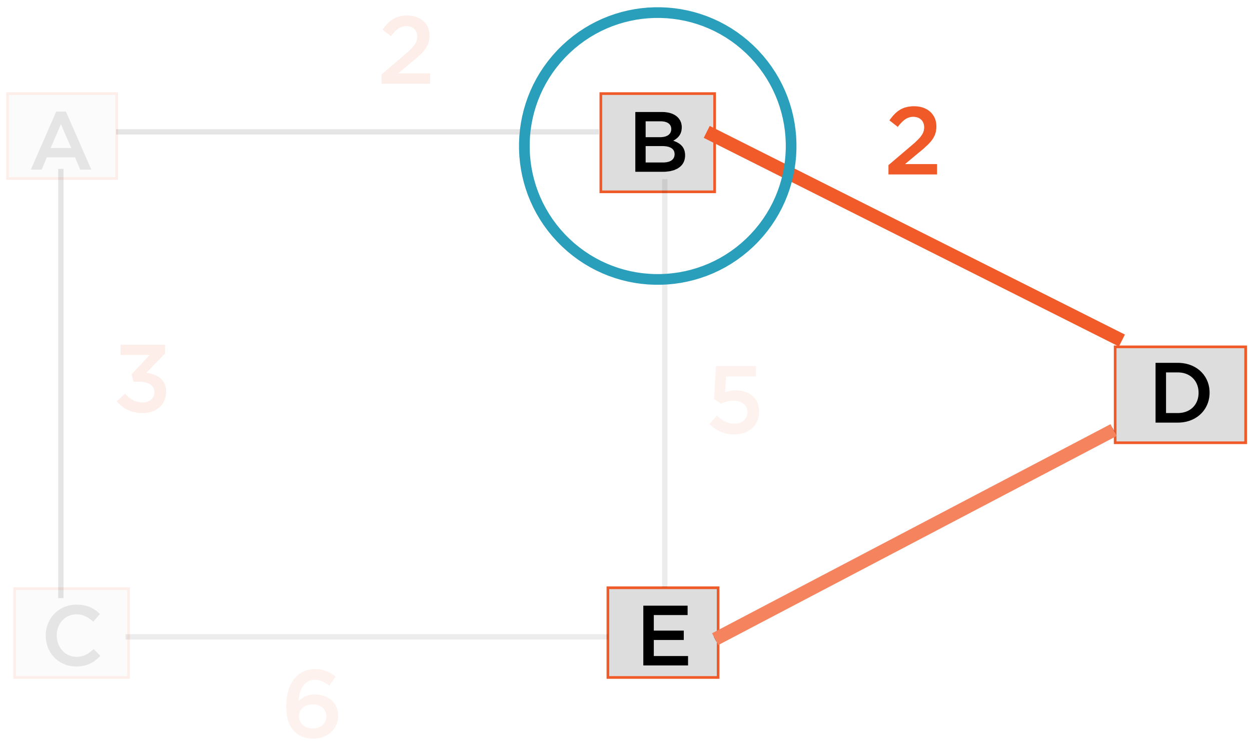
Now find the lowest weight edge out of either node

Prim's Algorithm



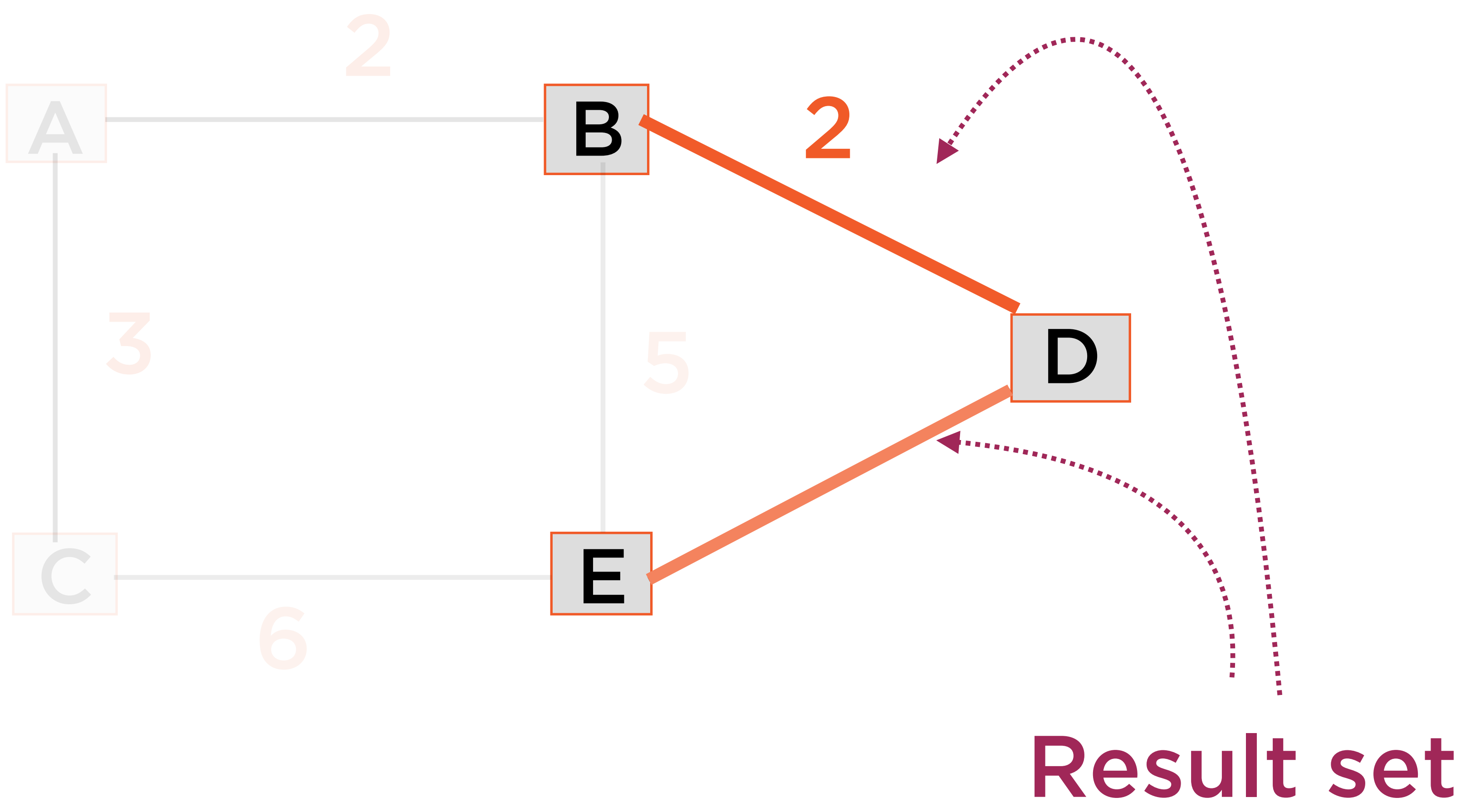
Now find the lowest weight edge out of either node

Prim’s Algorithm



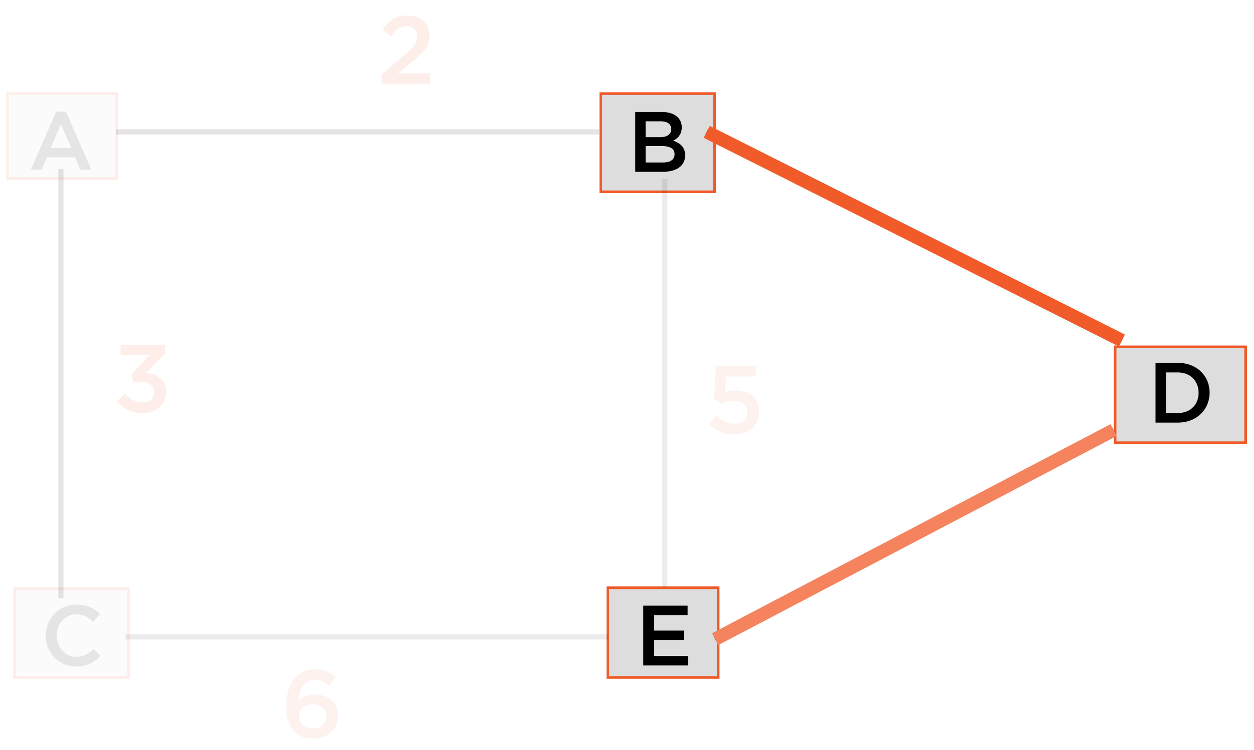
Lowest weighted edge connecting an unvisited node

Prim's Algorithm



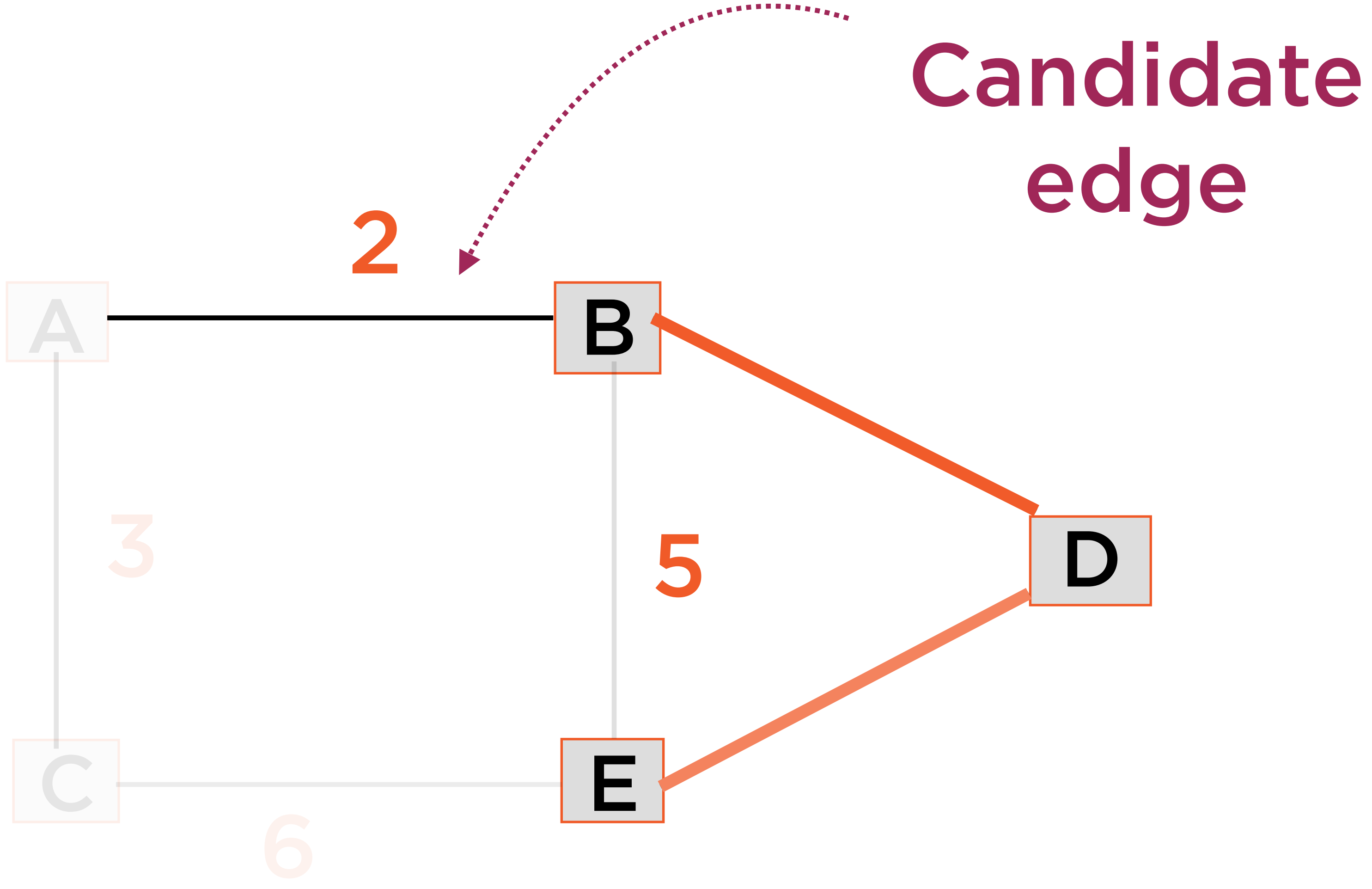
Add that edge to result as well

Prim's Algorithm



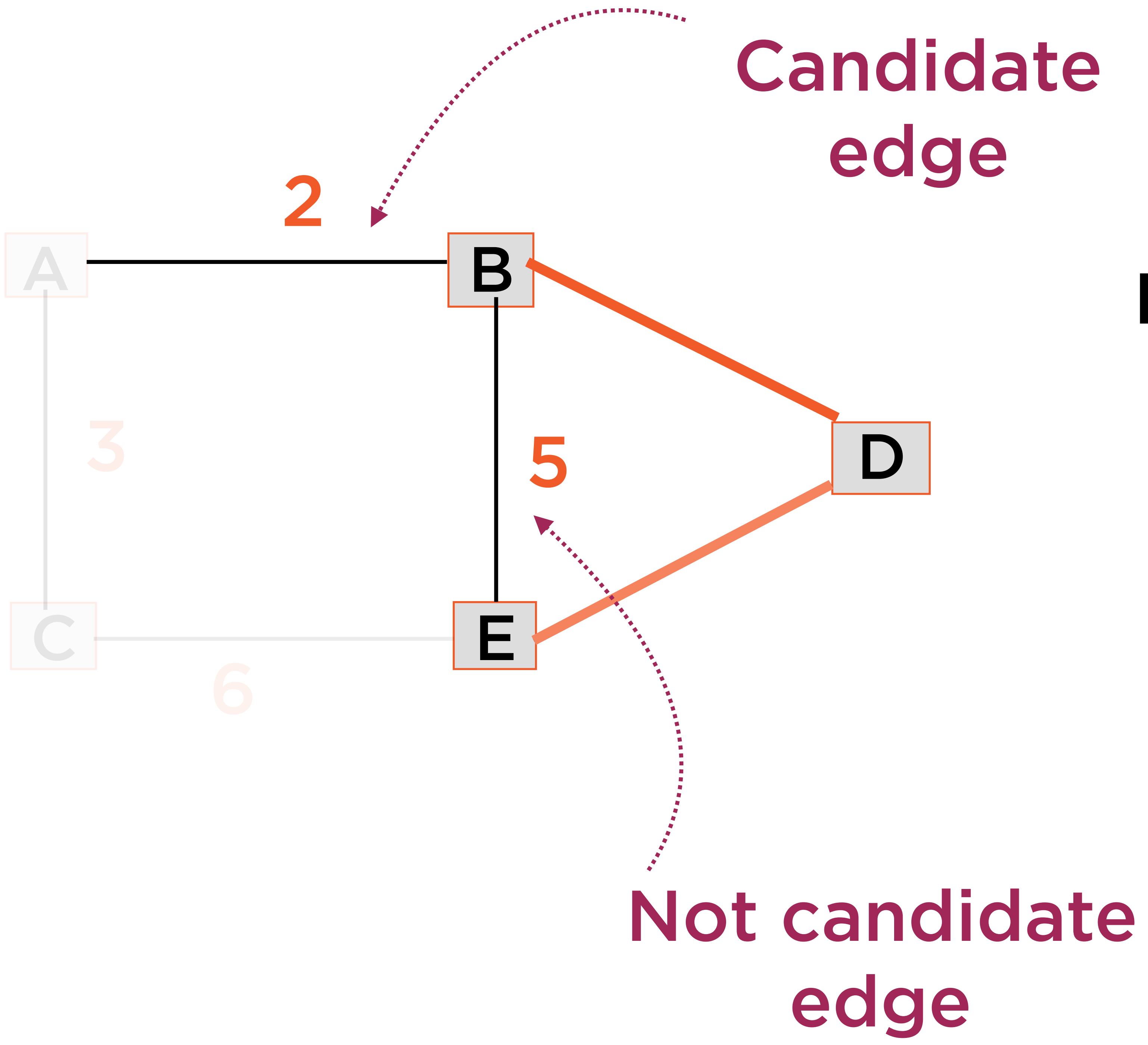
Once again, find lowest weight edge out of result set

Prim's Algorithm



Once again, find lowest weight edge out of result set

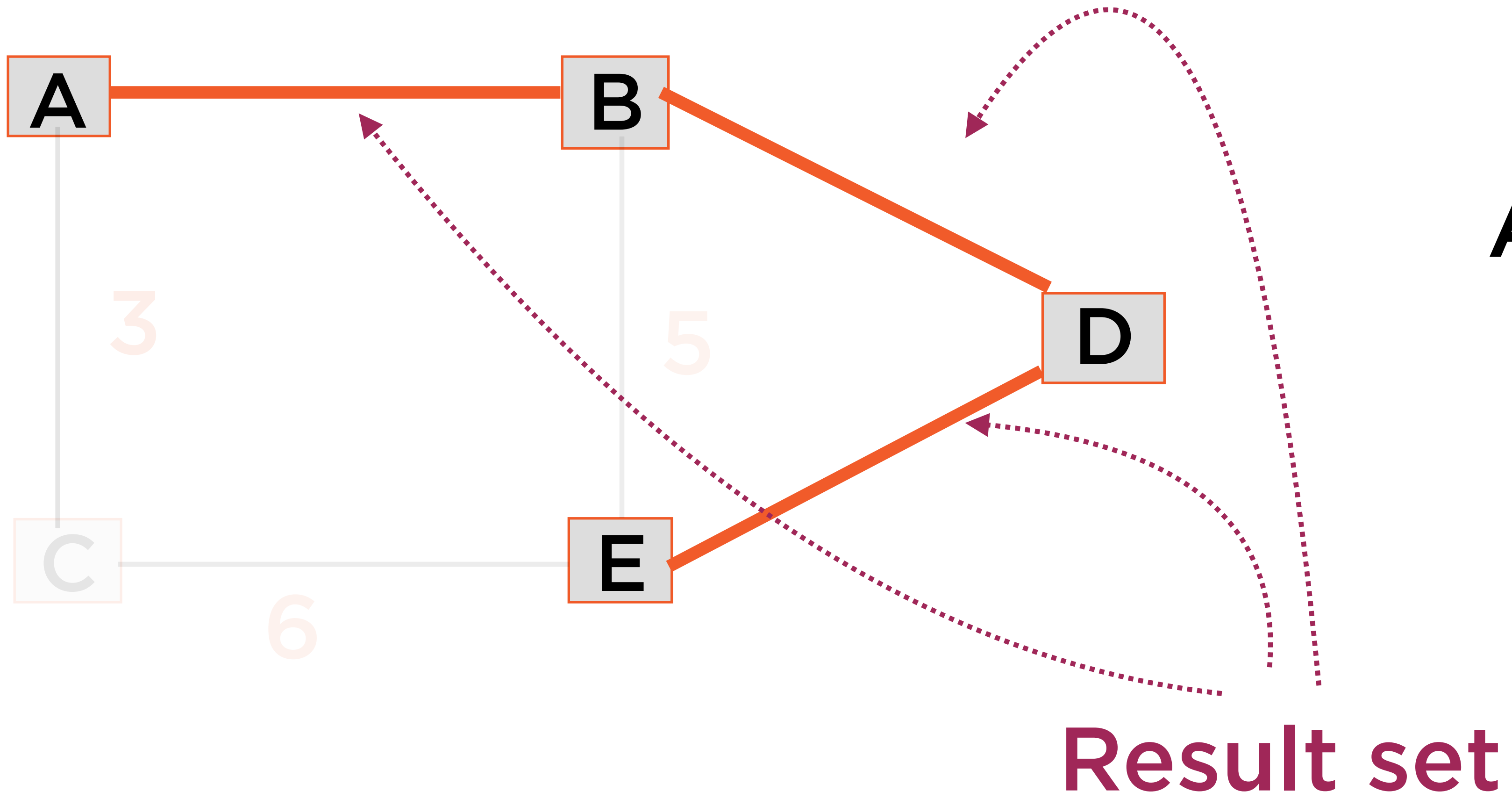
Prim's Algorithm



Note that the edge B - E does not even count

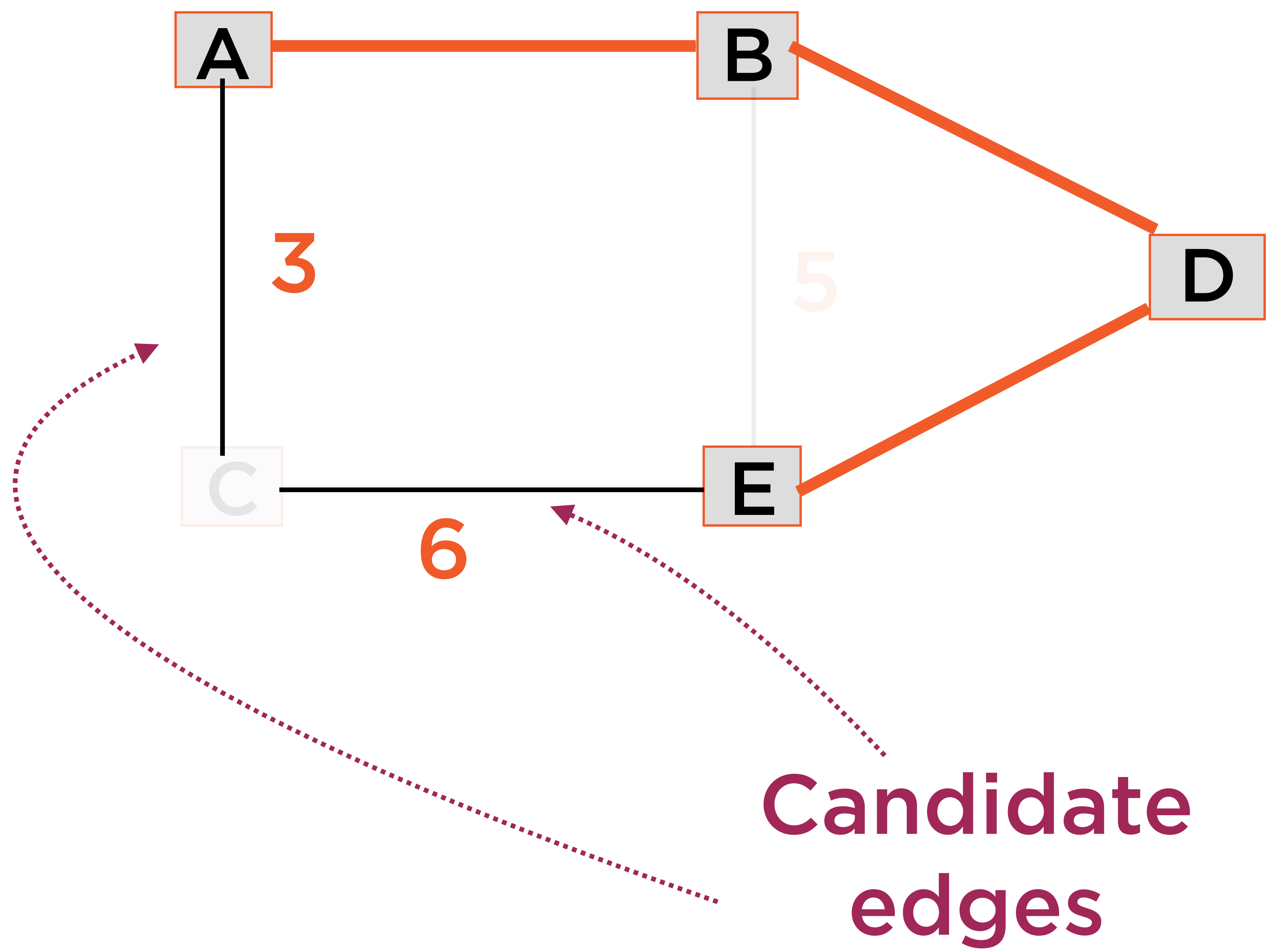
(E has already been visited)

Prim’s Algorithm



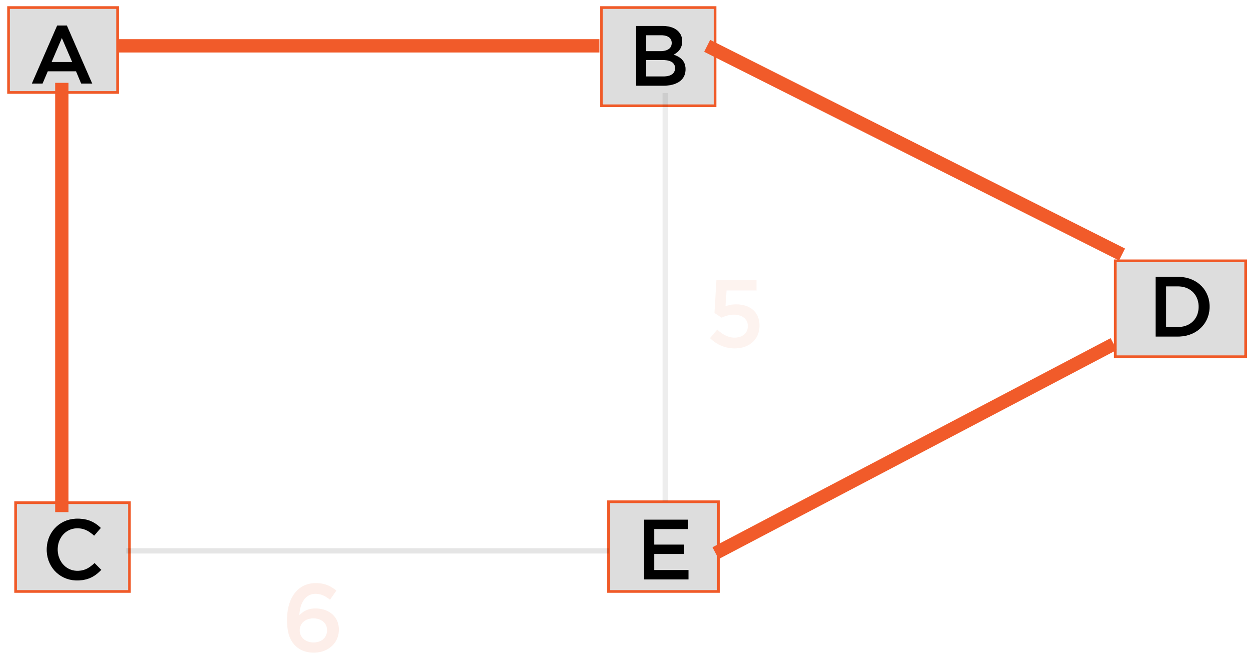
Add that edge to result set

Prim's Algorithm



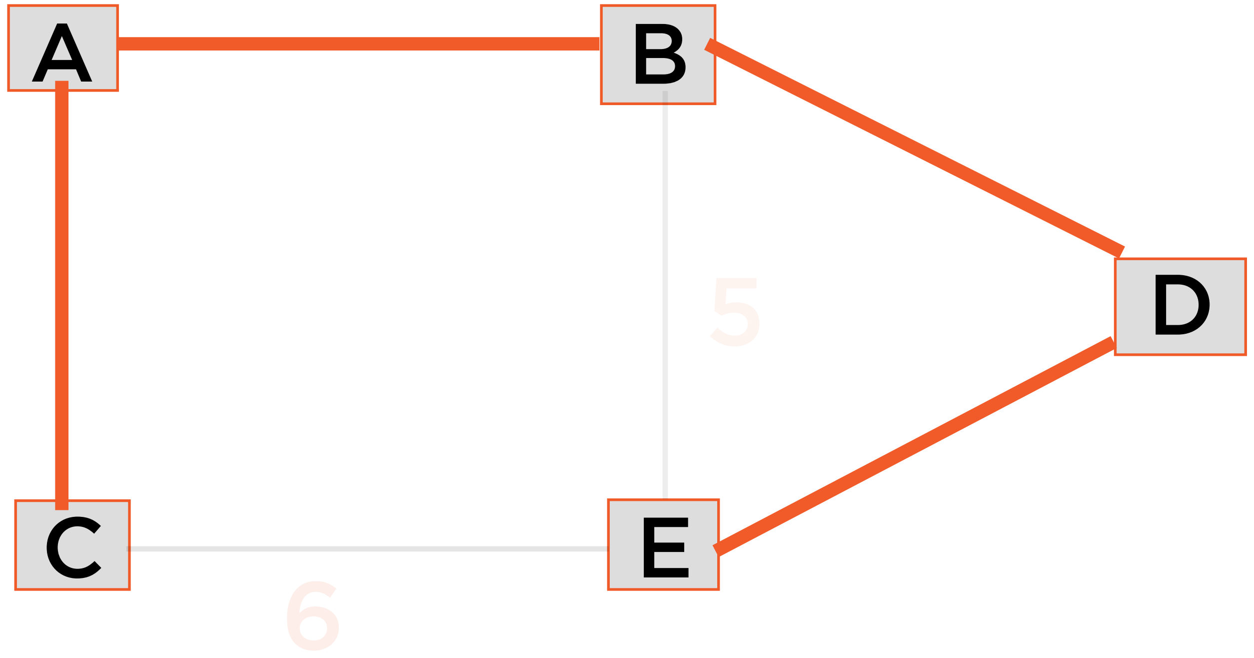
Once again, find lowest weight edge out of result set

Prim’s Algorithm



Add that edge to result set

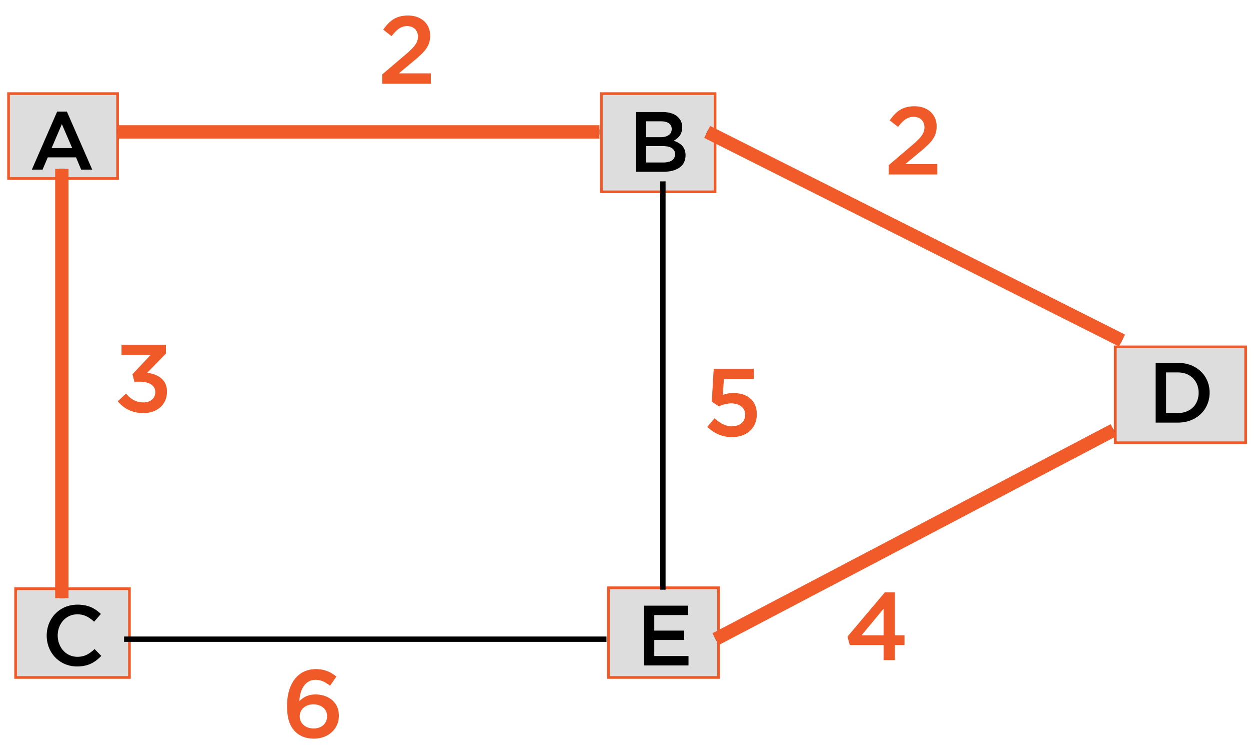
Prim’s Algorithm



All vertices in spanning tree, stop

Minimum spanning tree found

Prim’s Algorithm



Sum weights of edges in result set = 11

Prim’s algorithm finds a local optimum minimal spanning tree - it is a greedy algorithm

Prim's Algorithm

Algorithm considers edges in contiguous order

Benefit: Intermediate result is a tree as well

Drawback: Does not work for disconnected graphs

Prim’s Algorithm

Implementation heavily drawn from Dijkstra’s algorithm

Distance table, but with edge weight as the distance

Requires priority queue to find edge with least cost

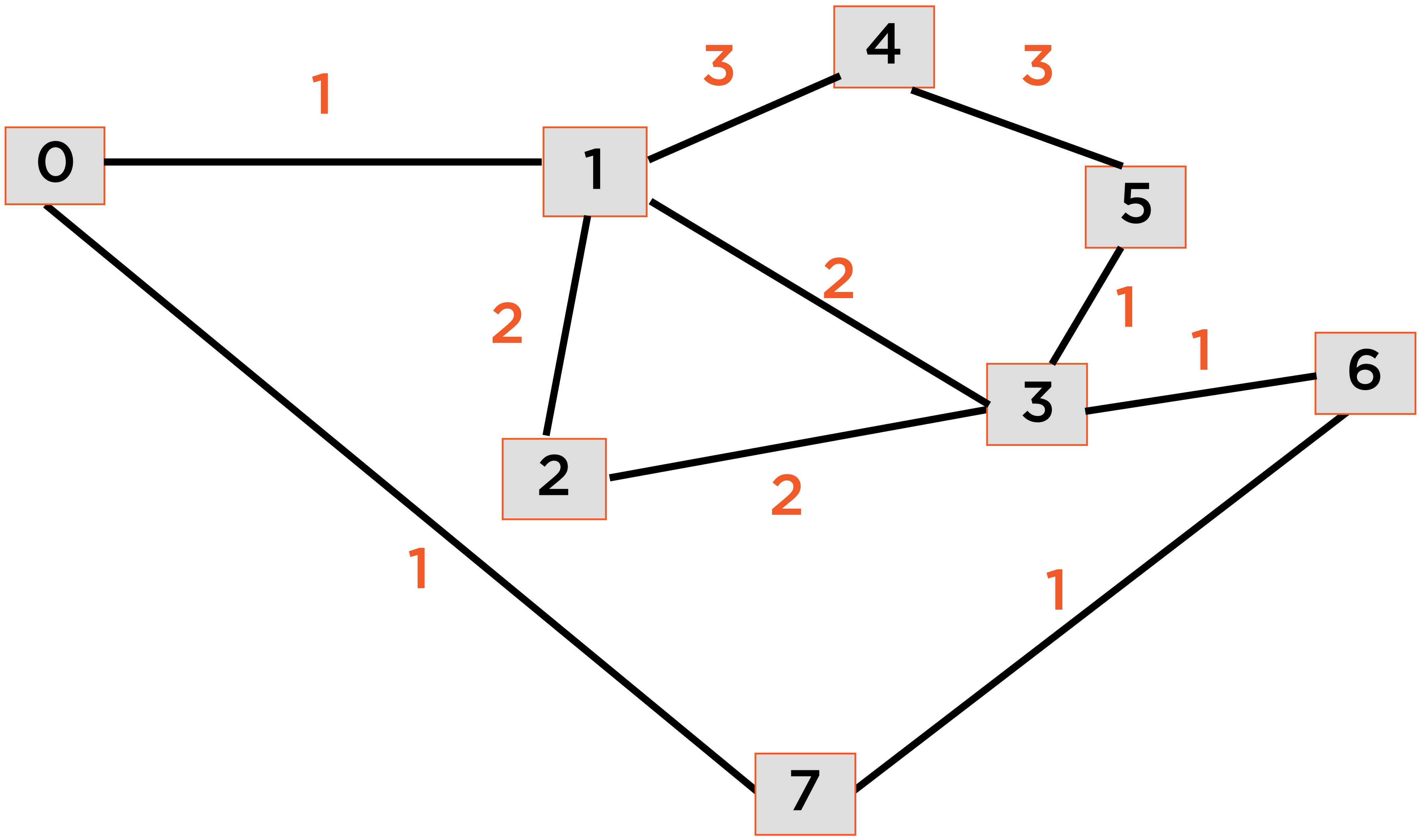
Prim's Algorithm

Queue Data Structure	Running Time
Binary Heap	O(E ln(V))
Array	$O(E + V^2)$

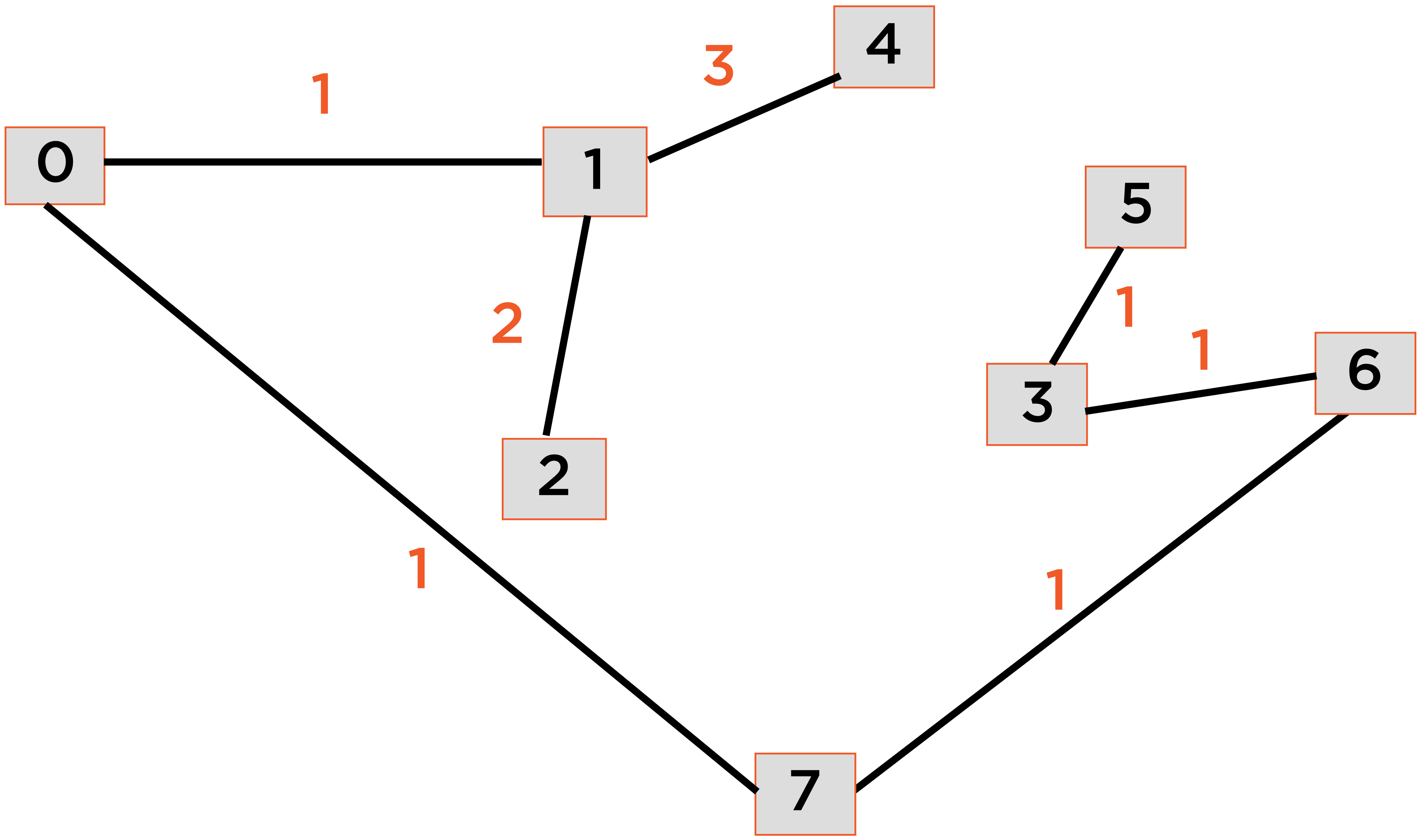
Demo

Implement Prim’s algorithm for a minimal spanning tree

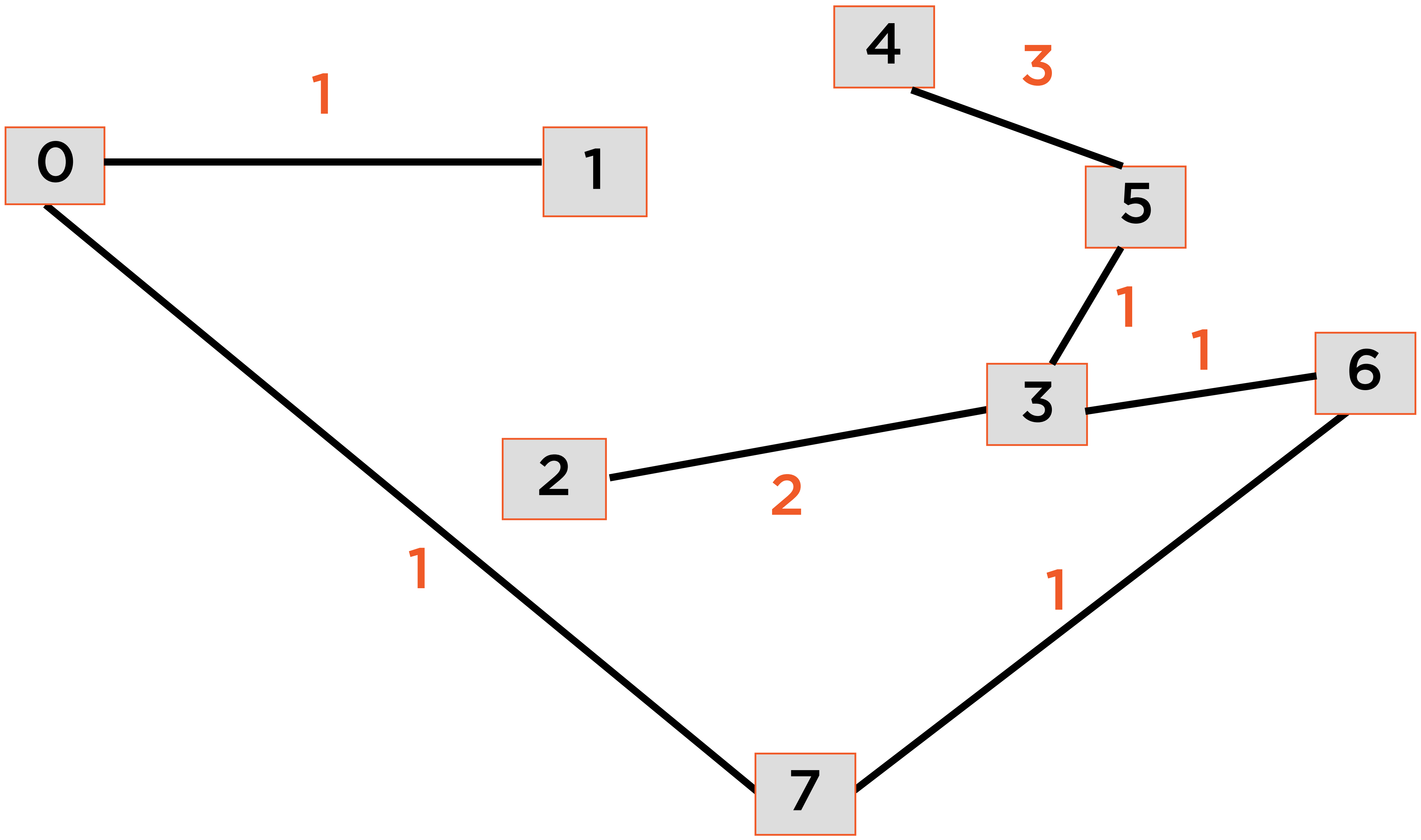
A Sample Undirected Graph



Minimal Spanning Tree Starting at Node 1



Minimal Spanning Tree Starting at Node 3



Kruskal’s Algorithm

Two Minimum Spanning Tree Algorithms

Prim's Algorithm

Works with connected graphs

Kruskal's Algorithm

Works even with disconnected graphs

Two Minimum Spanning Tree Algorithms

Prim's Algorithm

Works with connected graphs

Kruskal's Algorithm

Works even with disconnected graphs

Kruskal’s algorithm is a **greedy** algorithm to find a minimal spanning tree for a **weighted undirected** graph

The graph can be unconnected

Kruskal's Algorithm

Sort edges

Increasing order of weights

Can use priority queue

Initialize empty result

Empty set of edges

At end will hold minimum spanning tree

Find shortest edge

Not currently in result

Dequeue from priority queue

Reject if cycle introduced

Else add to result set

This is a greedy step

Stop

When N-1 edges in result

N = number of vertices in graph

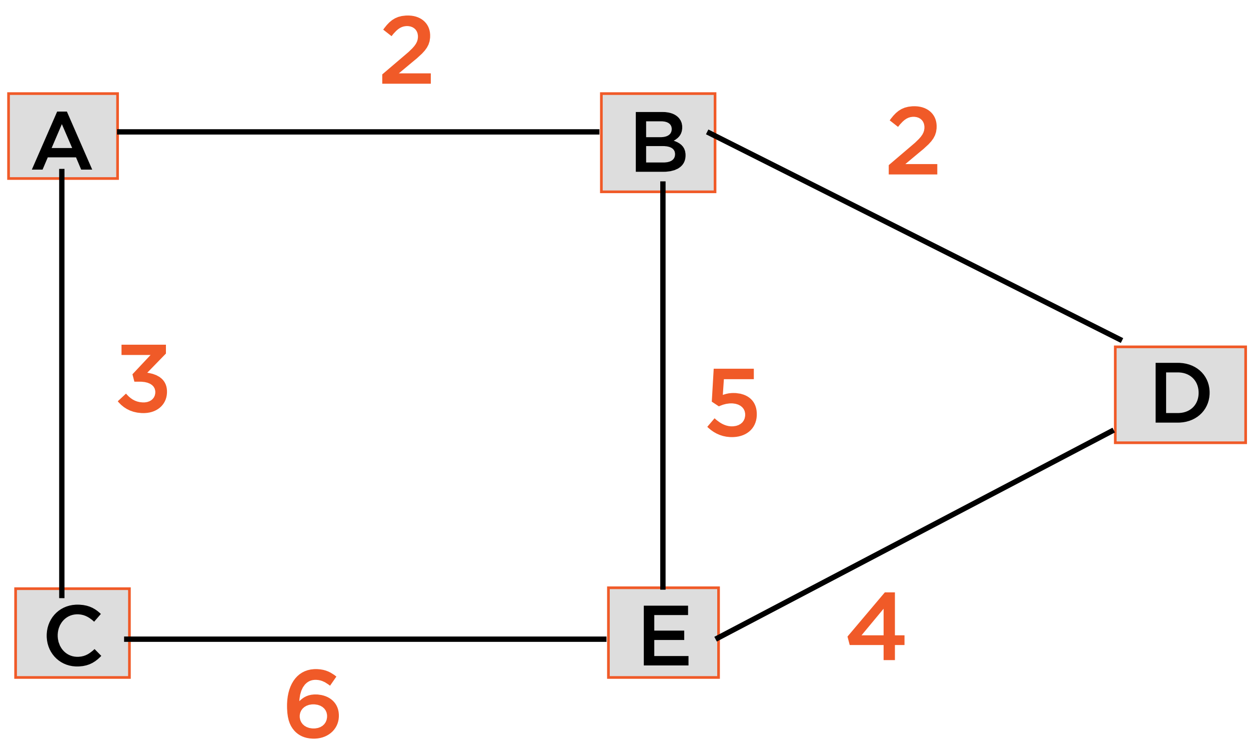
Kruskal's Algorithm

Sort edges

Increasing order of weights

Can use priority queue

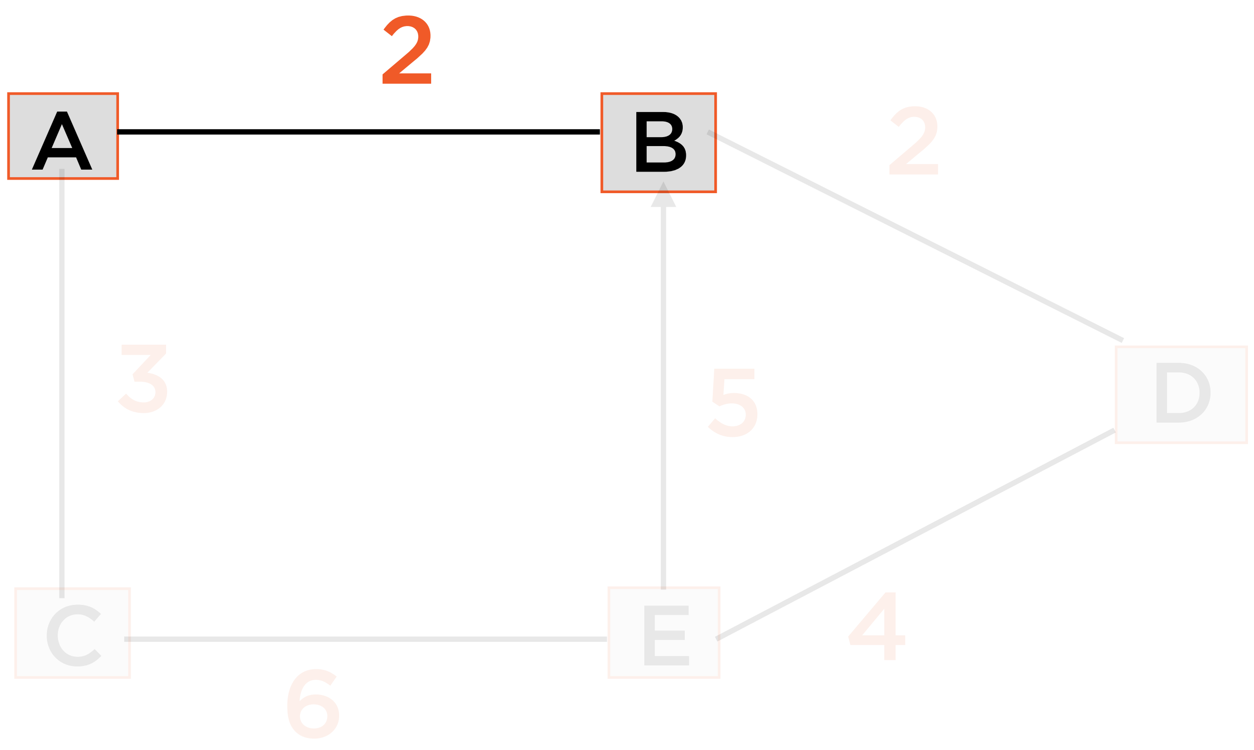
Kruskal's Algorithm



Edge	Weight

Priority Queue

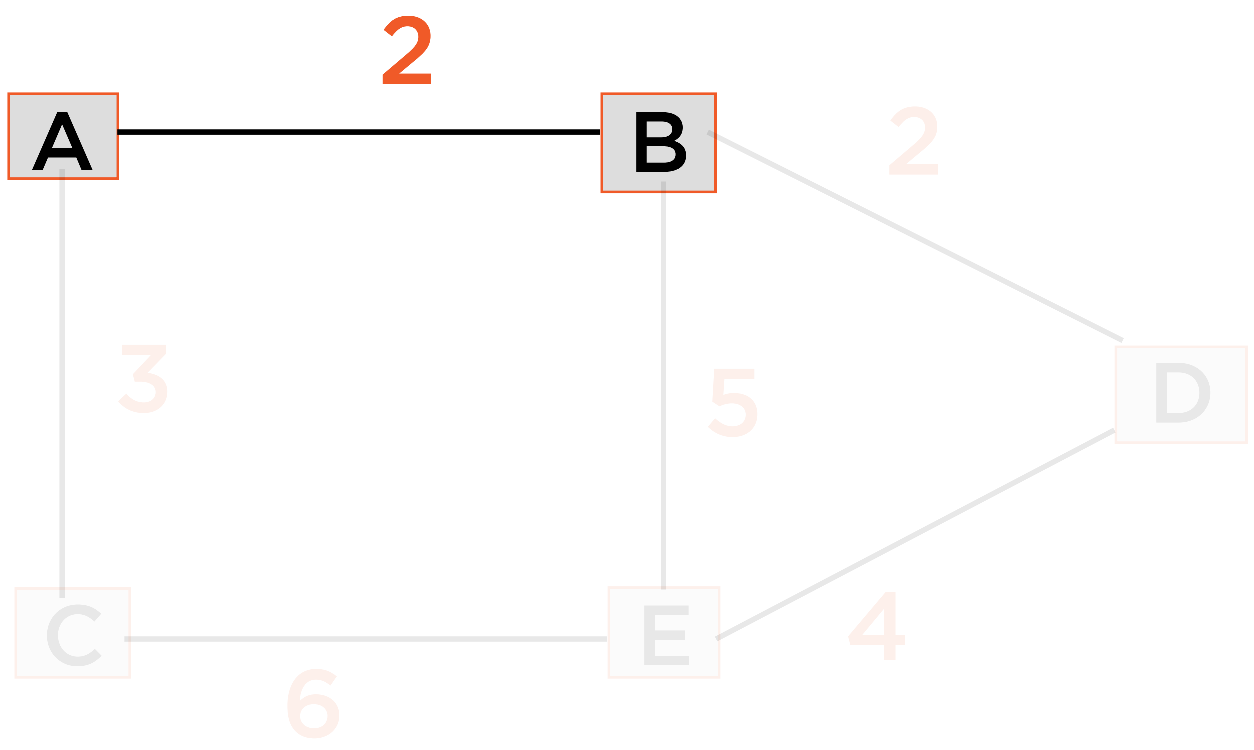
Kruskal's Algorithm



Edge	Weight

Priority Queue

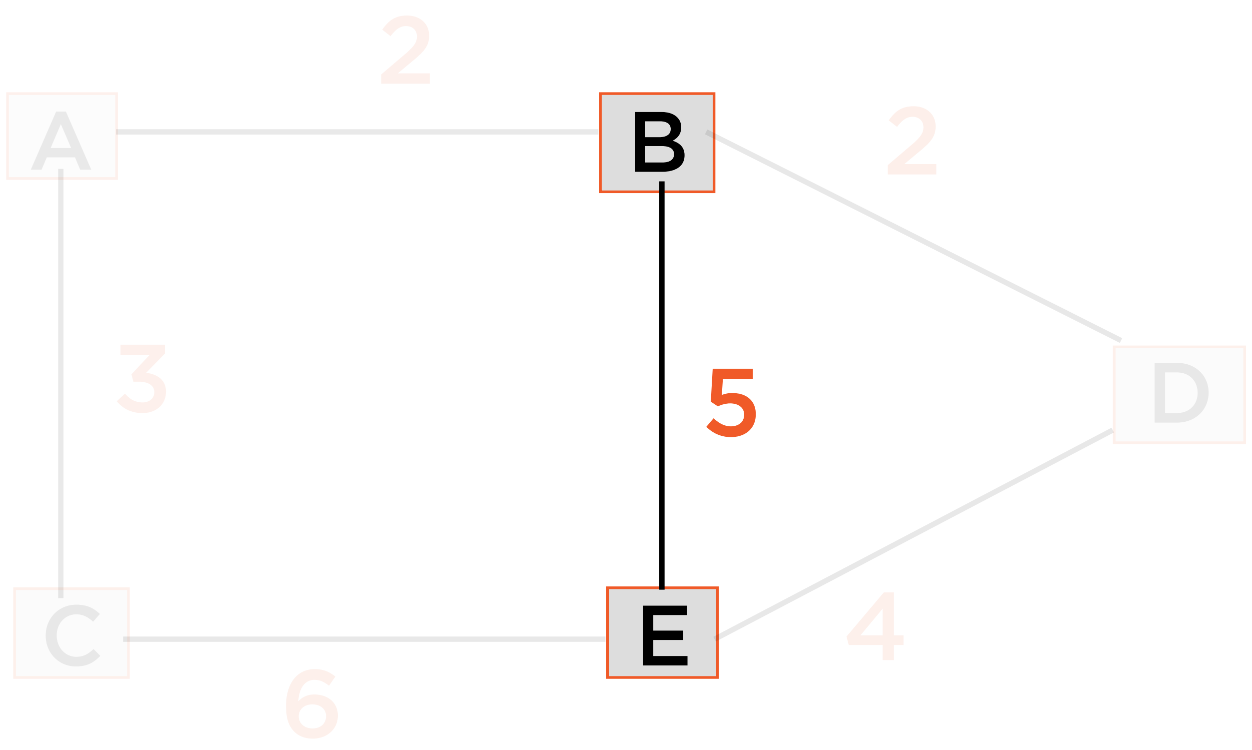
Kruskal's Algorithm



Edge	Weight
A - B	2

Priority Queue

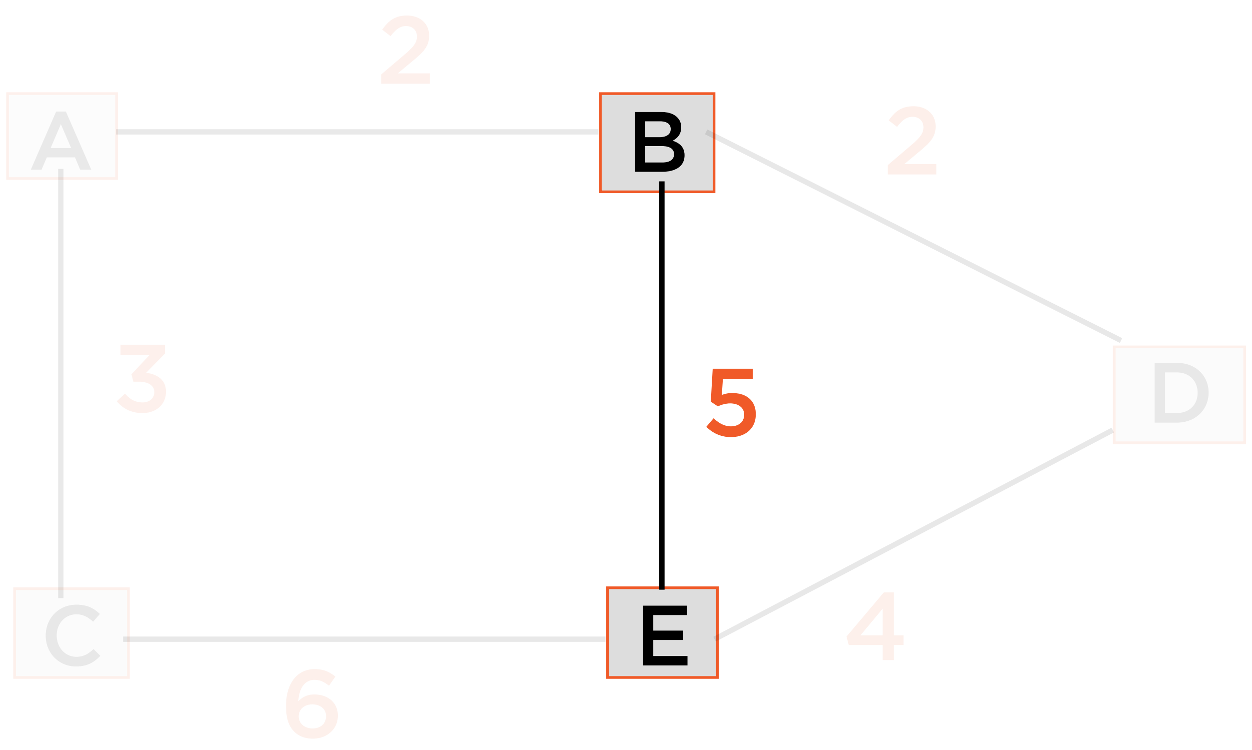
Kruskal's Algorithm



Edge	Weight
A - B	2

Priority Queue

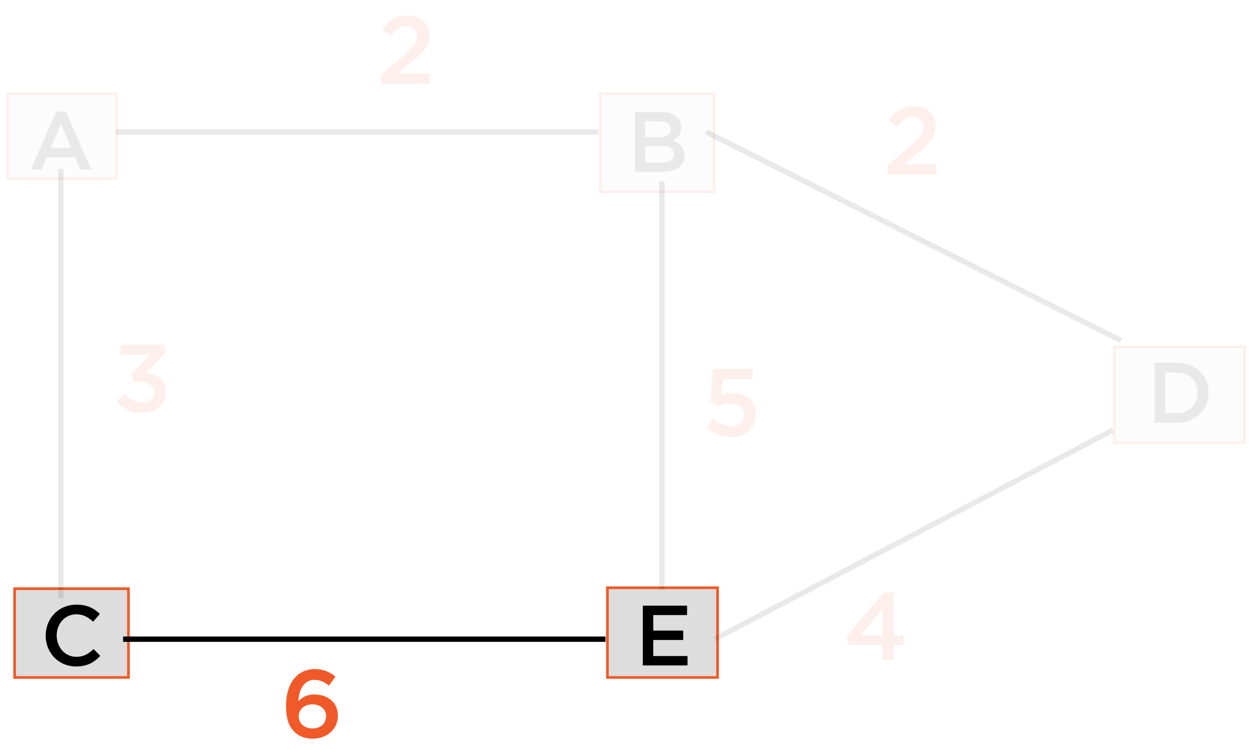
Kruskal's Algorithm



Edge	Weight
A - B	2
B - E	5

Priority Queue

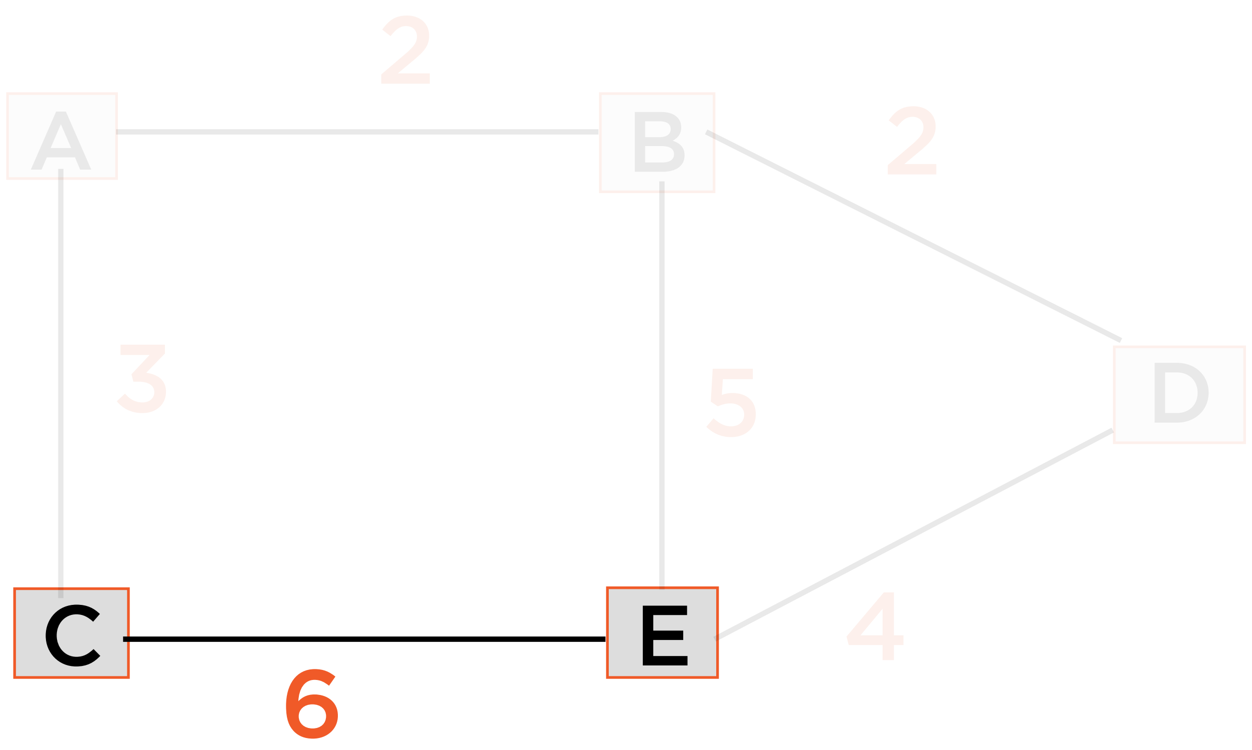
Kruskal's Algorithm



Edge	Weight
A - B	2
B - E	5

Priority Queue

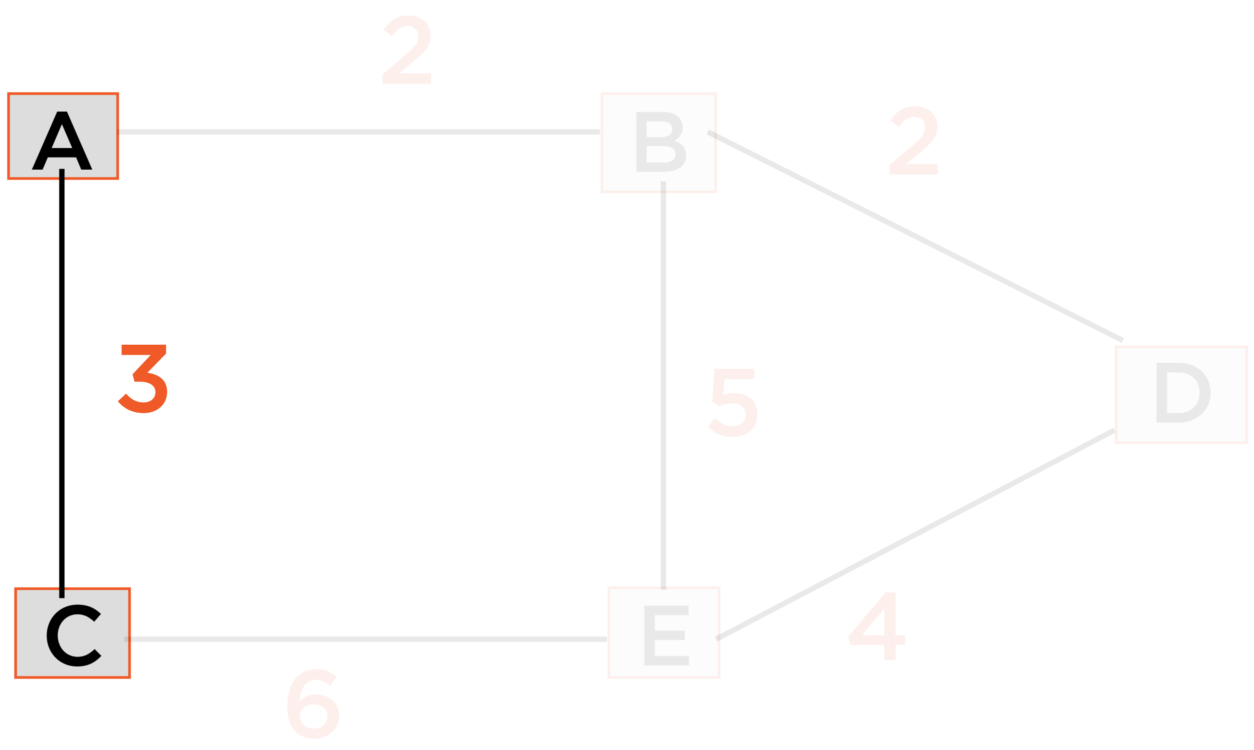
Kruskal’s Algorithm



Edge	Weight
A - B	2
B - E	5
C - E	6

Priority Queue

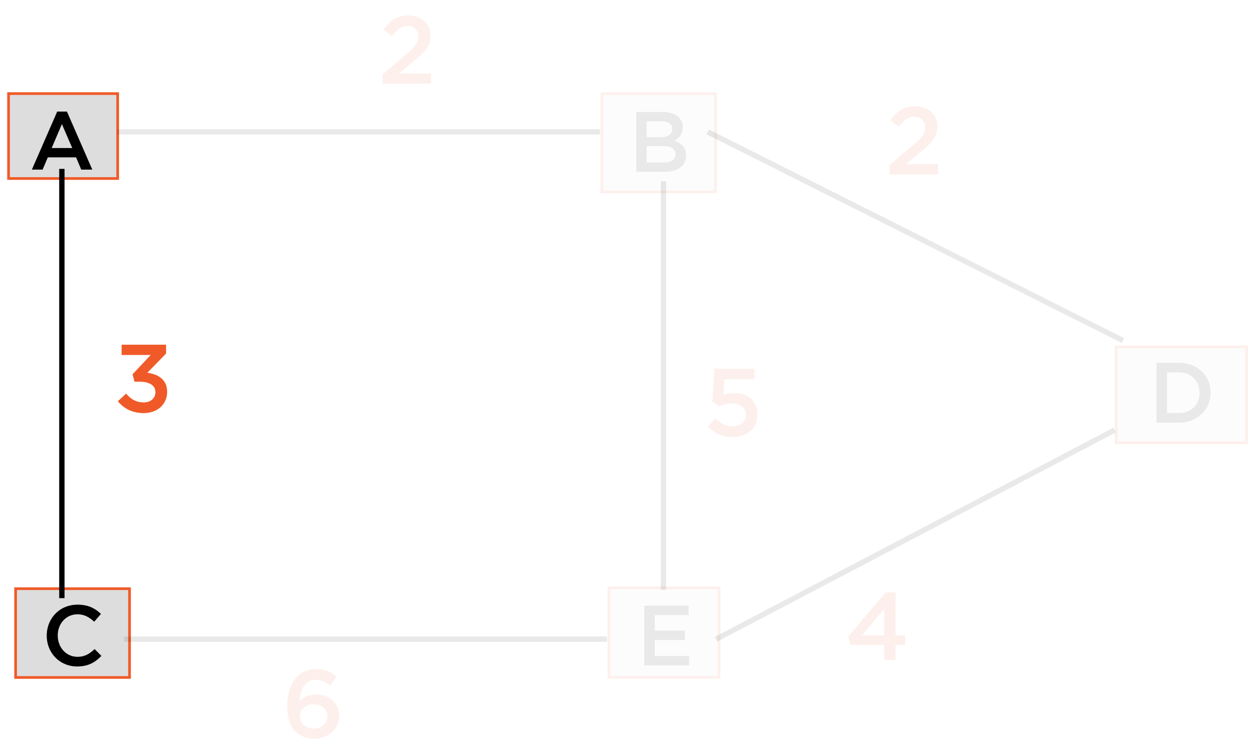
Kruskal's Algorithm



Edge	Weight
A - B	2
B - E	5
C - E	6

Priority Queue

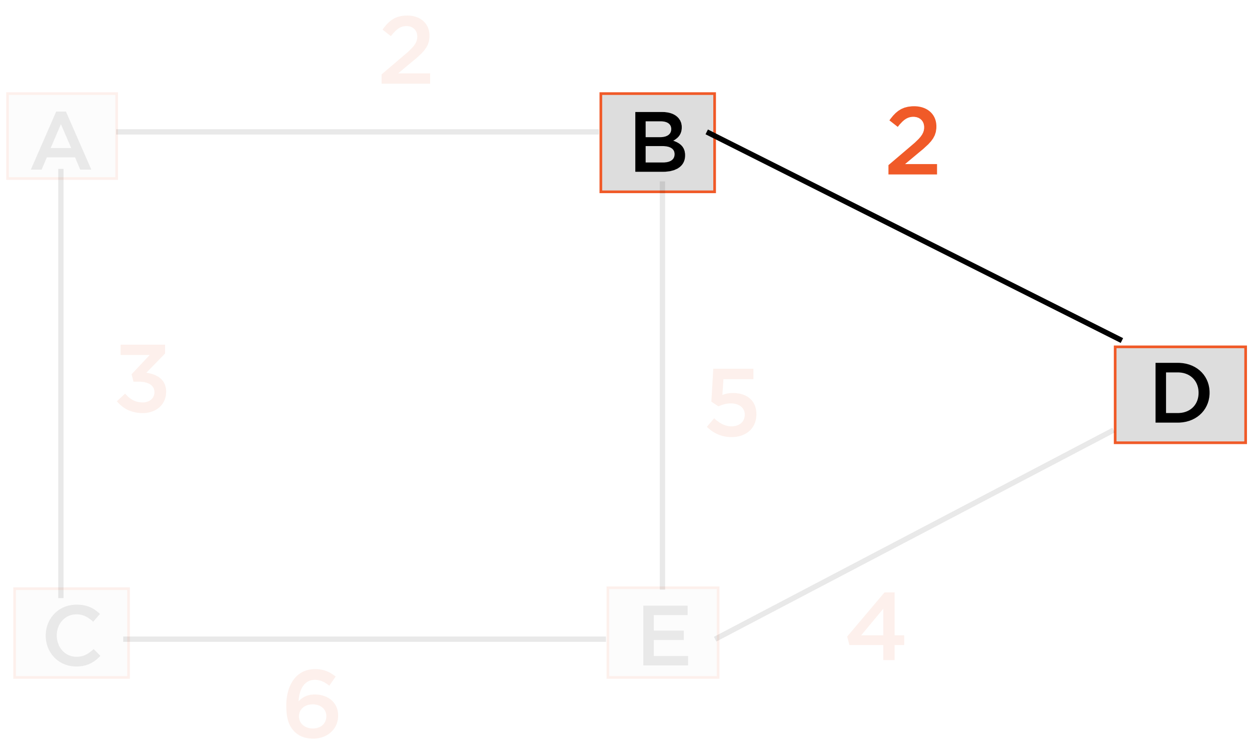
Kruskal's Algorithm



Edge	Weight
A - B	2
A - C	3
B - E	5
C - E	6

Priority Queue

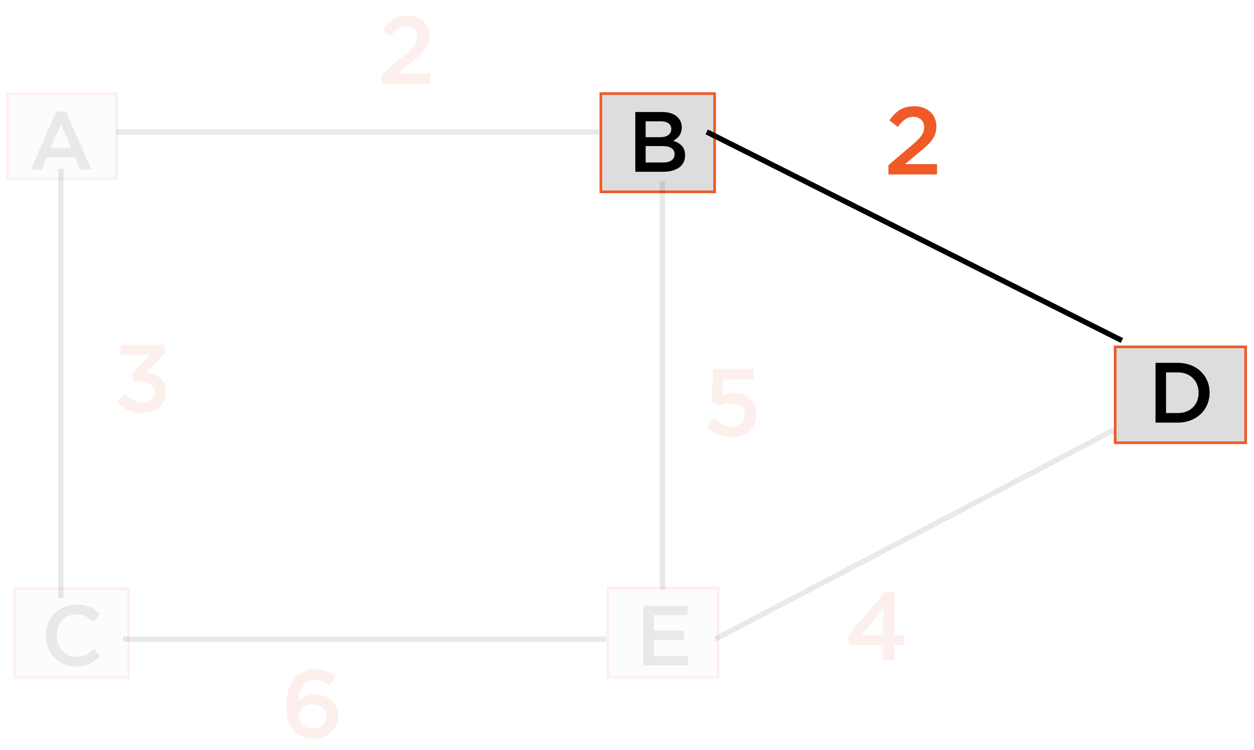
Kruskal's Algorithm



Edge	Weight
A - B	2
A - C	3
B - E	5
C - E	6

Priority Queue

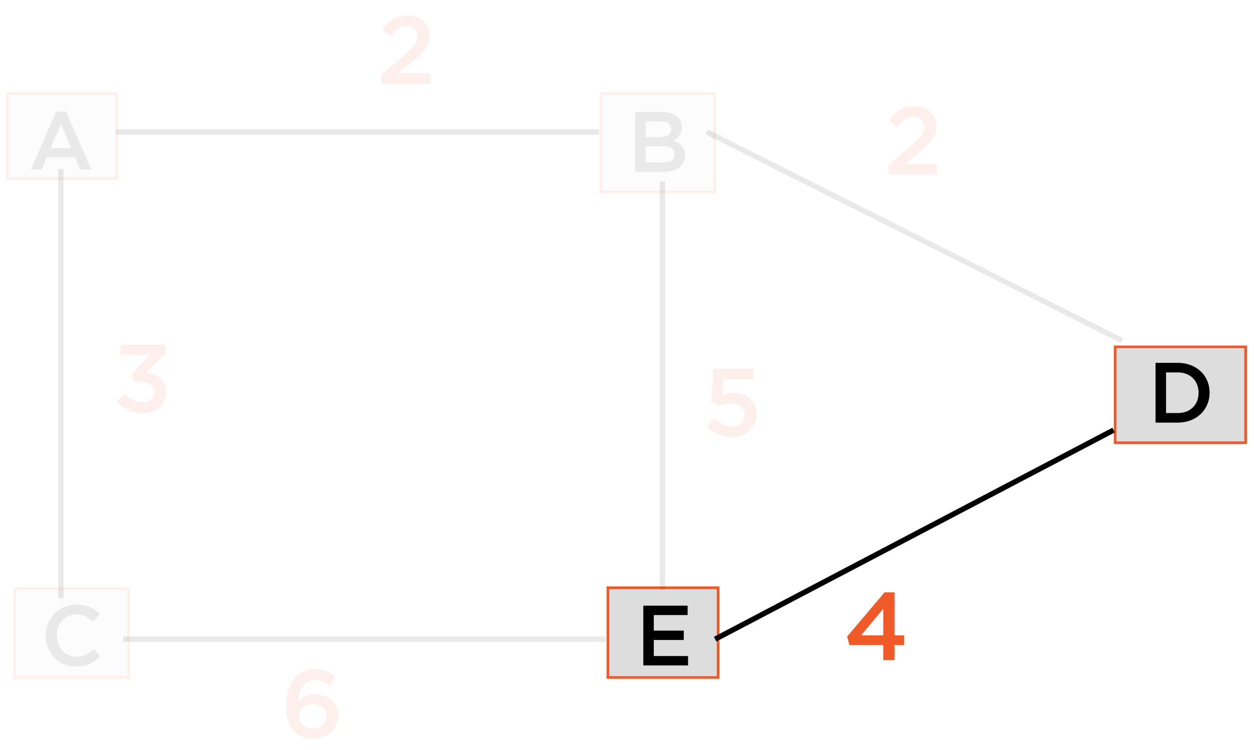
Kruskal's Algorithm



Edge	Weight
A - B	2
B - D	2
A - C	3
B - E	5
C - E	6

Priority Queue

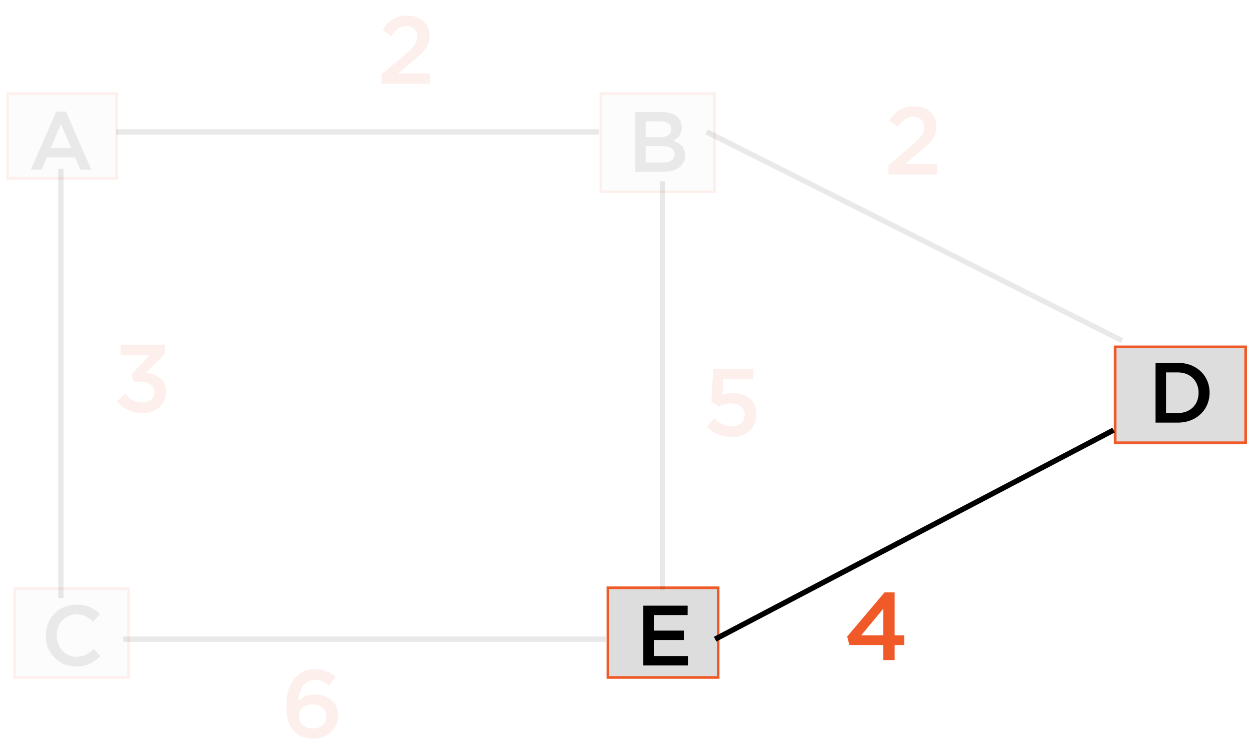
Kruskal's Algorithm



Edge	Weight
A - B	2
B - D	2
A - C	3
B - E	5
C - E	6

Priority Queue

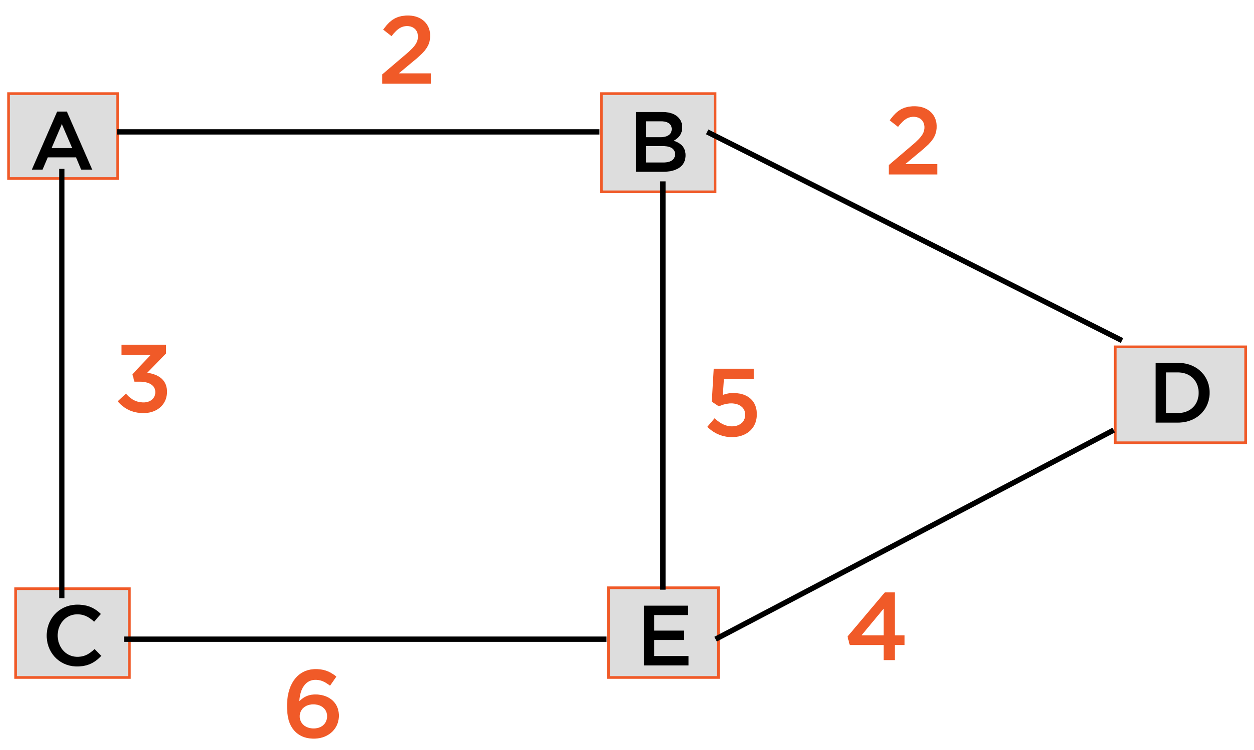
Kruskal's Algorithm



Edge	Weight
A - B	2
B - D	2
A - C	3
E - D	4
B - E	5
C - E	6

Priority Queue

Kruskal's Algorithm



Edge	Weight
A - B	2
B - D	2
A - C	3
E - D	4
B - E	5
C - E	6

Priority Queue

Kruskal's Algorithm

Sort edges

Increasing order of weights

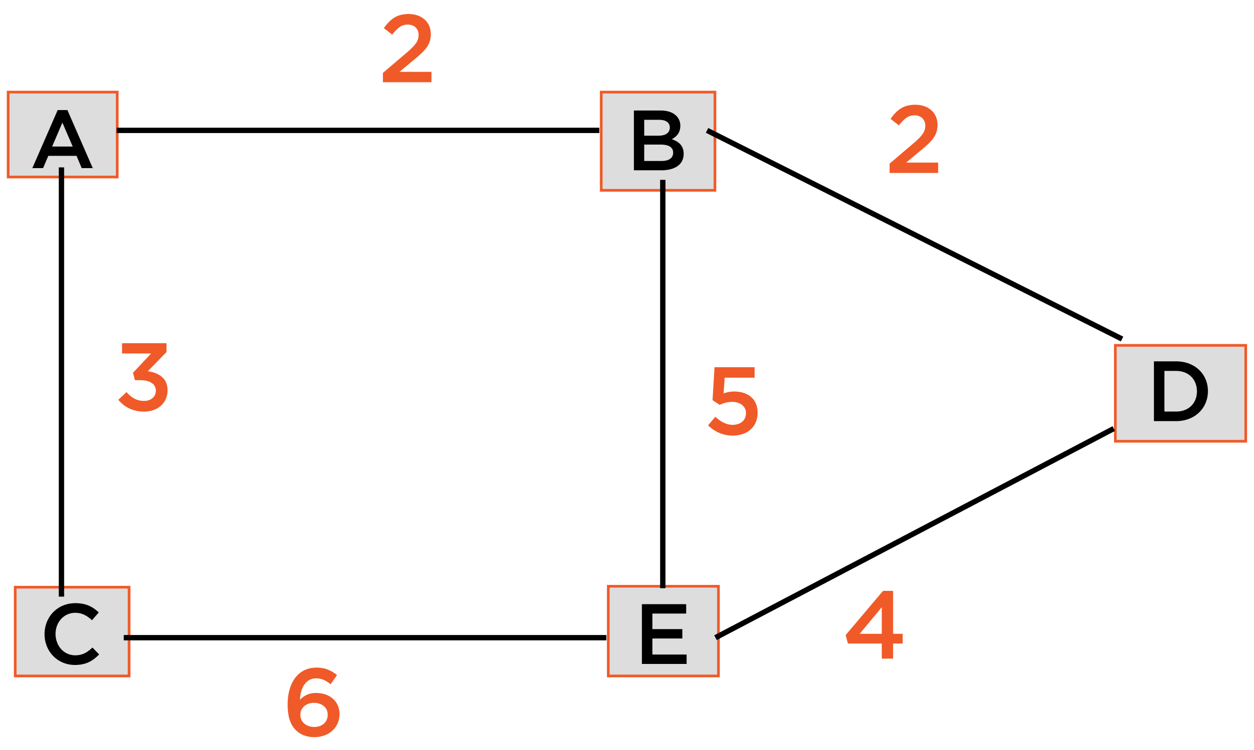
Can use priority queue

Initialize empty result

Empty set of edges

At end will hold minimum spanning tree

Kruskal’s Algorithm



Priority Queue

Edge	Weight
A - B	2
B - D	2
A - C	3
E - D	4
B - E	5
C - E	6

Result

Edge	Weight

Kruskal’s Algorithm

Sort edges

Increasing order of weights

Can use priority queue

Initialize empty result

Empty set of edges

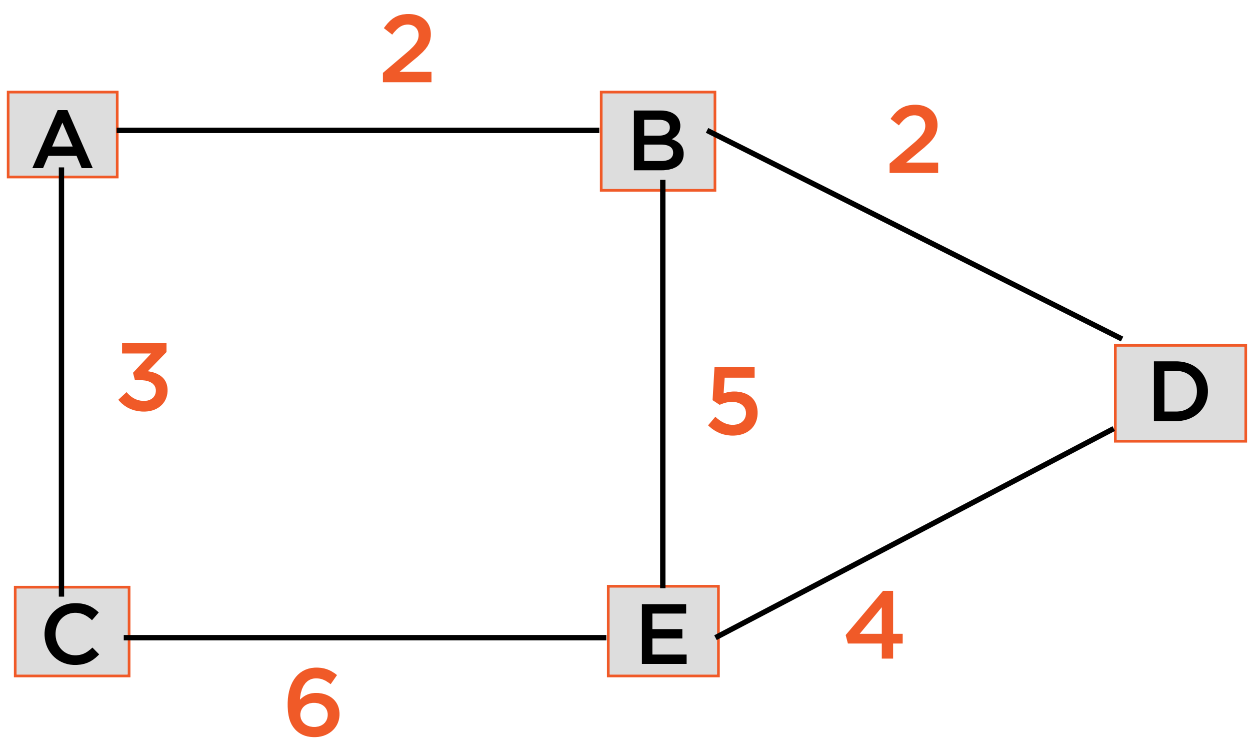
At end will hold minimum spanning tree

Find shortest edge

Not currently in result

Dequeue from priority queue

Kruskal's Algorithm



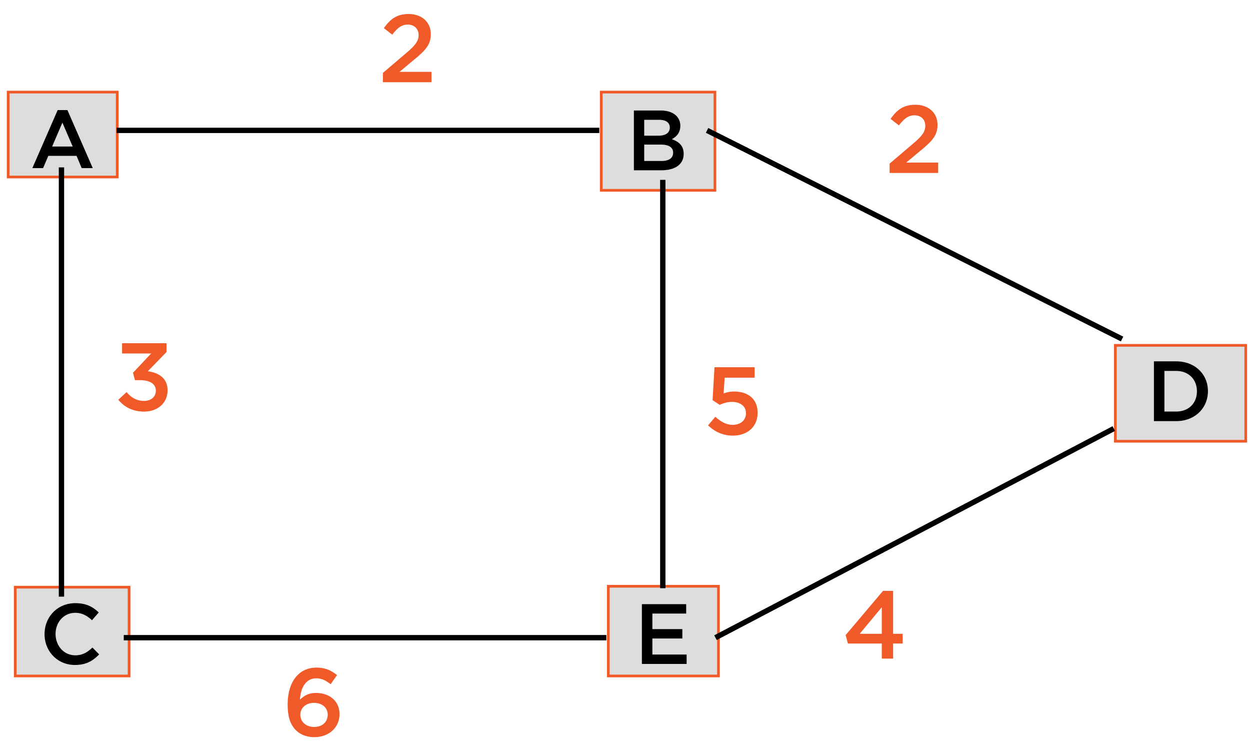
Priority Queue

Edge	Weight
A - B	2
B - D	2
A - C	3
E - D	4
B - E	5
C - E	6

Result

Edge	Weight

Kruskal's Algorithm



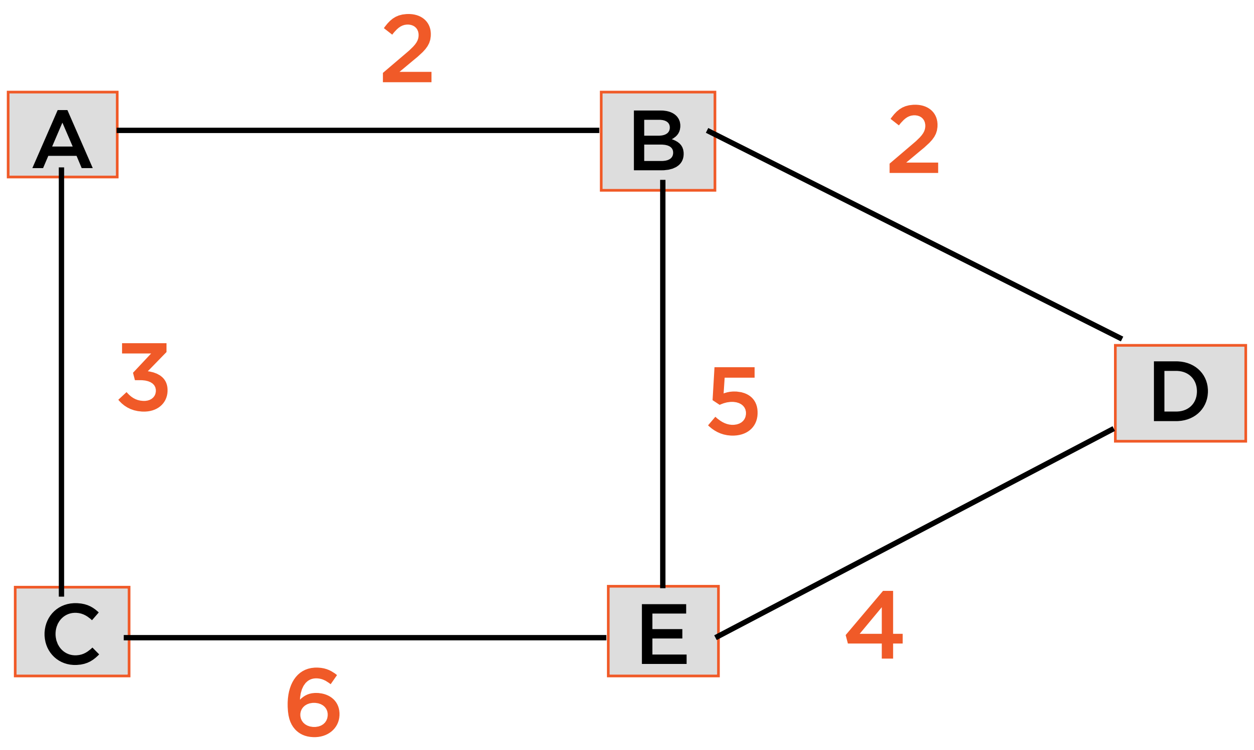
Priority Queue

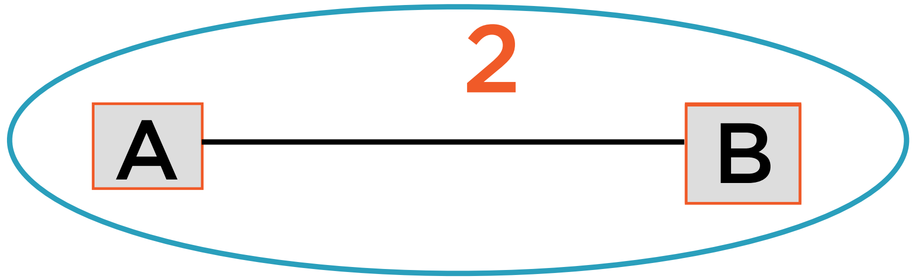
Edge	Weight
A - B	2
B - D	2
A - C	3
E - D	4
B - E	5
C - E	6

Result

Edge	Weight

Kruskal's Algorithm





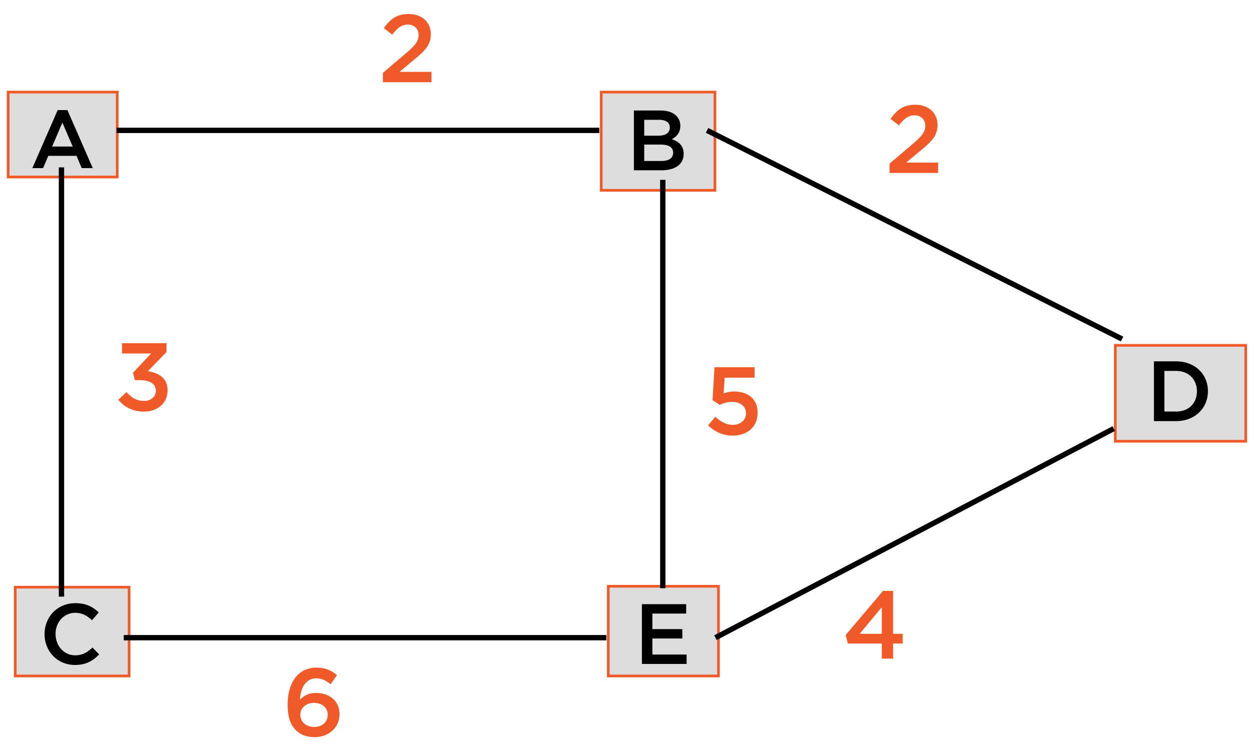
Priority Queue

Edge	Weight
B - D	2
A - C	3
E - D	4
B - E	5
C - E	6

Result

Edge	Weight
A - B	2

Kruskal's Algorithm



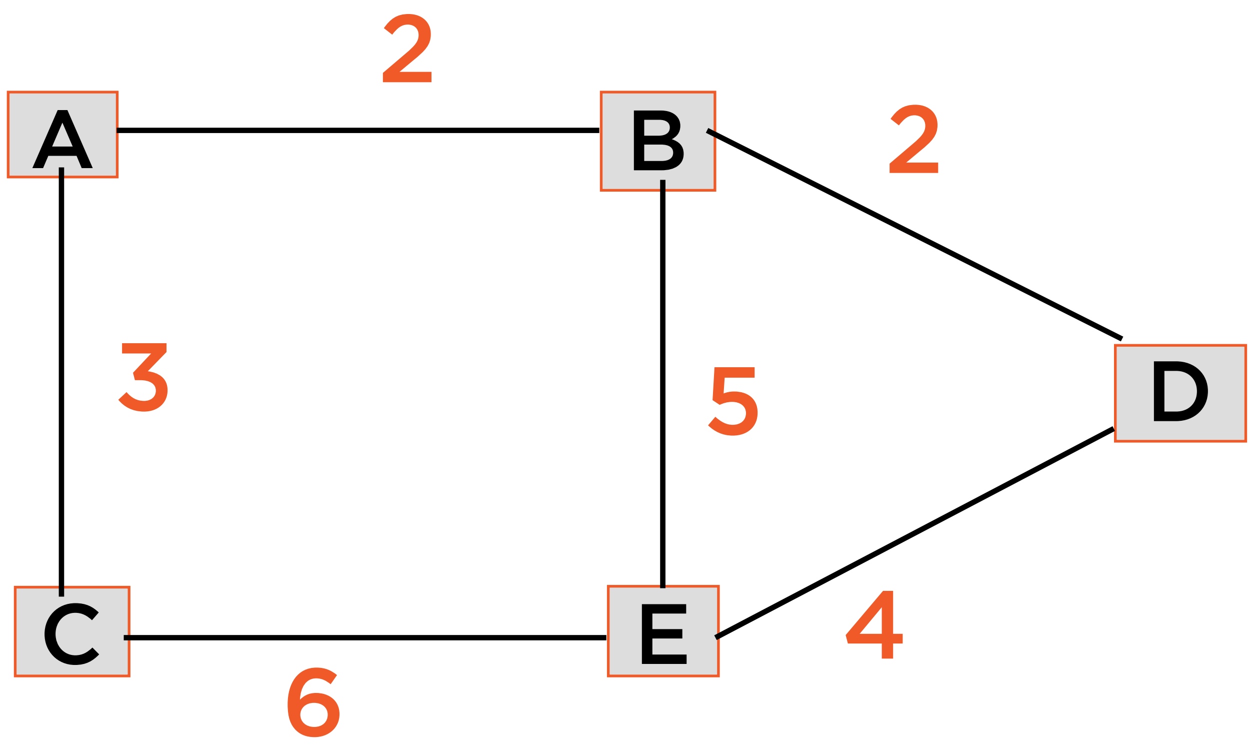
Priority Queue

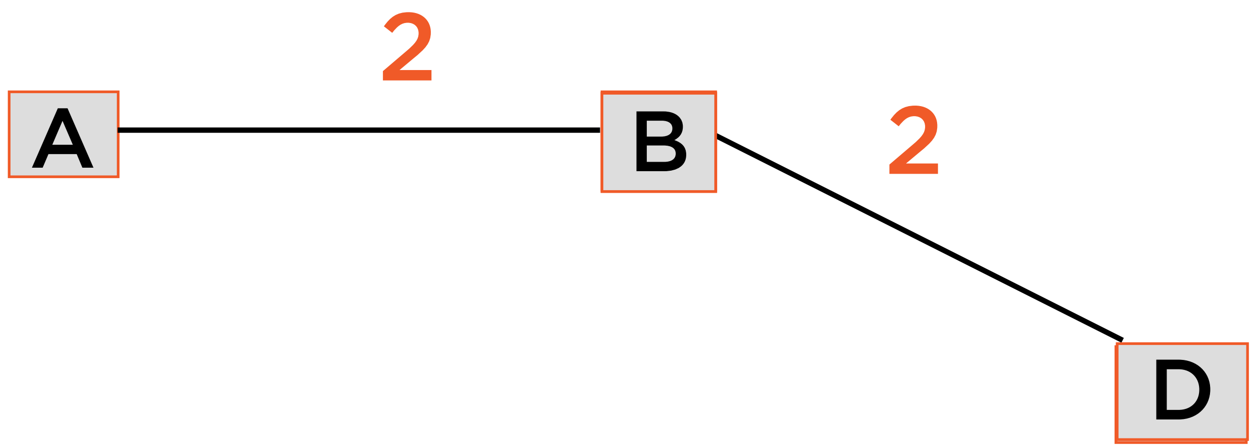
Edge	Weight
B - D	2
A - C	3
E - D	4
B - E	5
C - E	6

Result

Edge	Weight
A - B	2

Kruskal's Algorithm





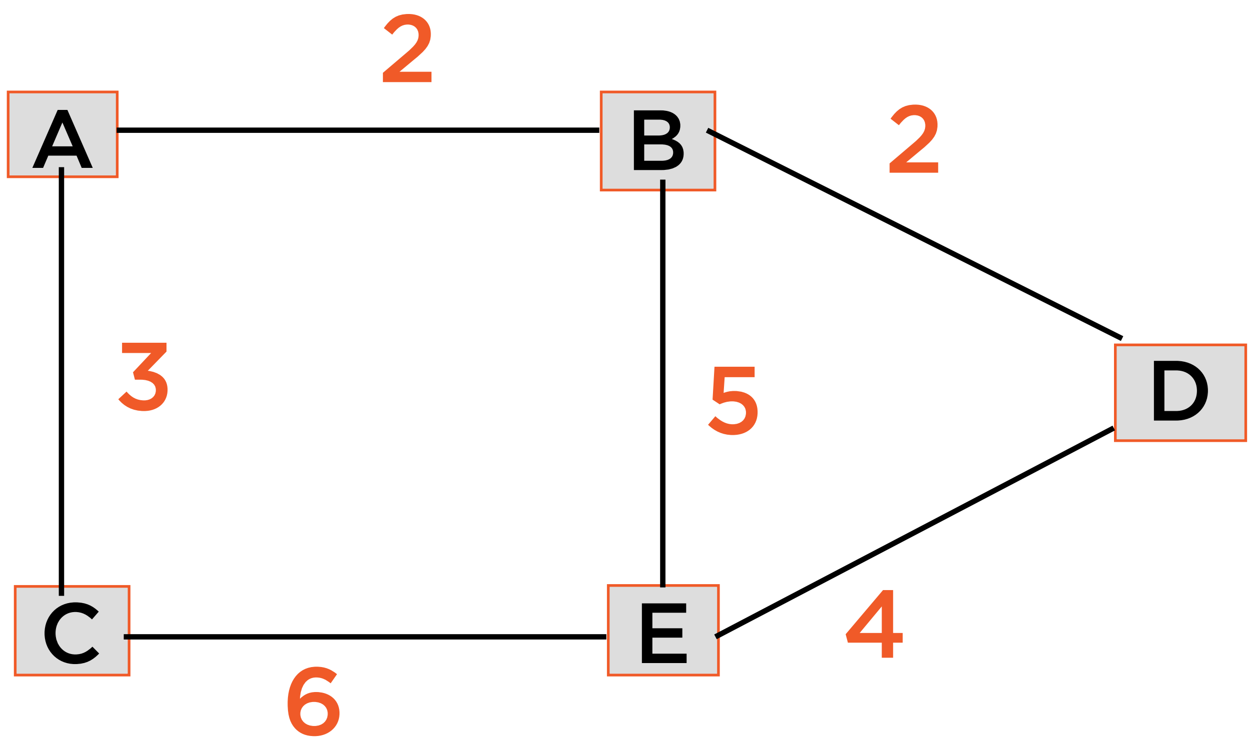
Priority Queue

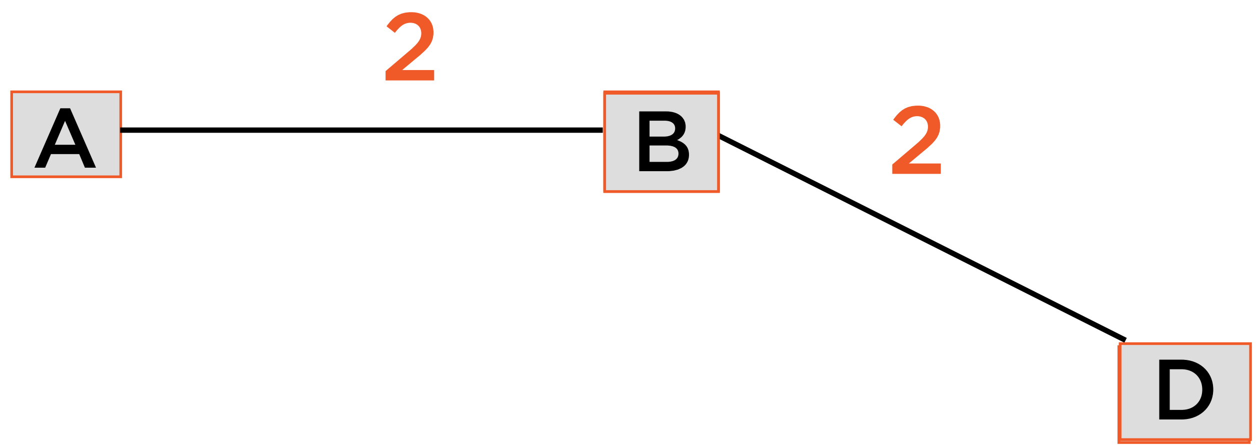
Edge	Weight
A - C	3
E - D	4
B - E	5
C - E	6

Result

Edge	Weight
A - B	2
B - D	2

Kruskal's Algorithm





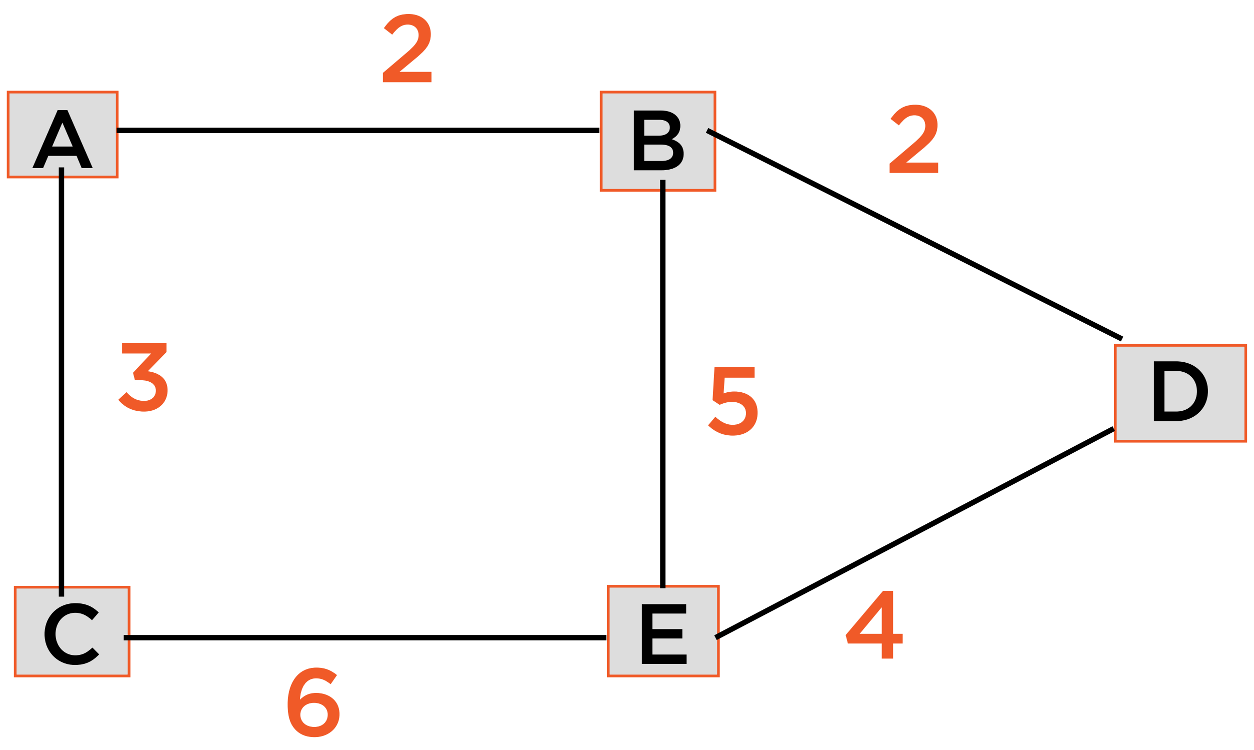
Priority Queue

Edge	Weight
A - C	3
E - D	4
B - E	5
C - E	6

Result

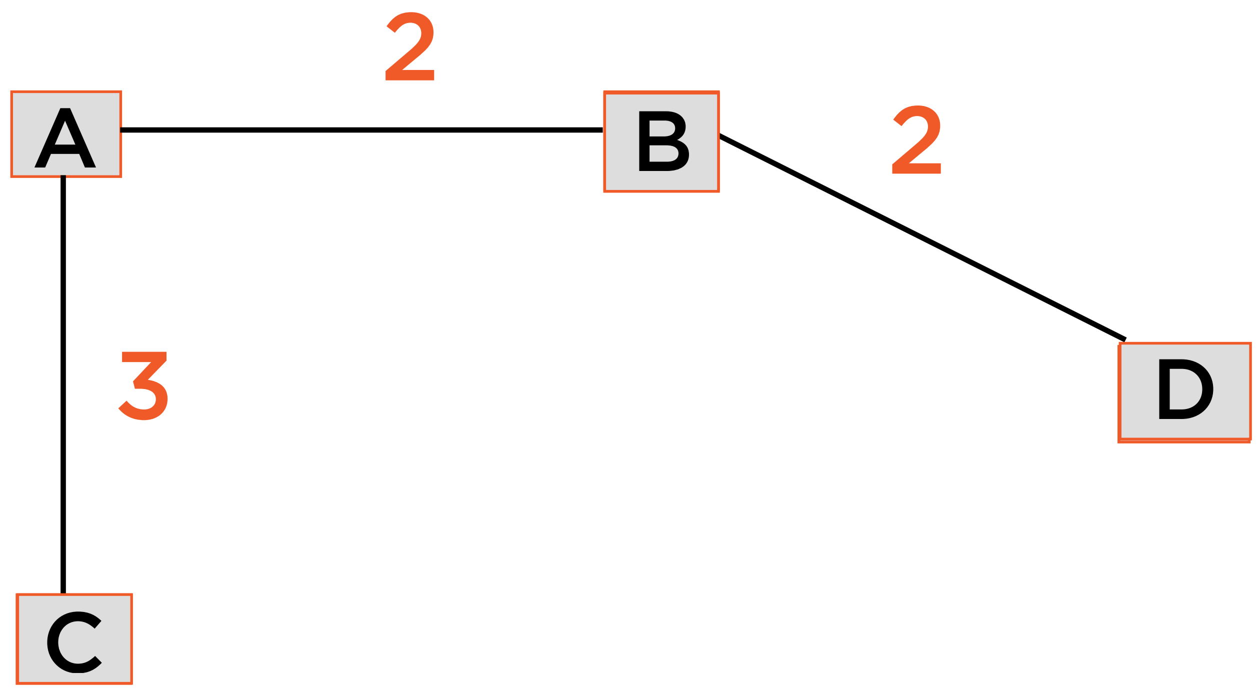
Edge	Weight
A - B	2
B - D	2

Kruskal's Algorithm



Priority Queue

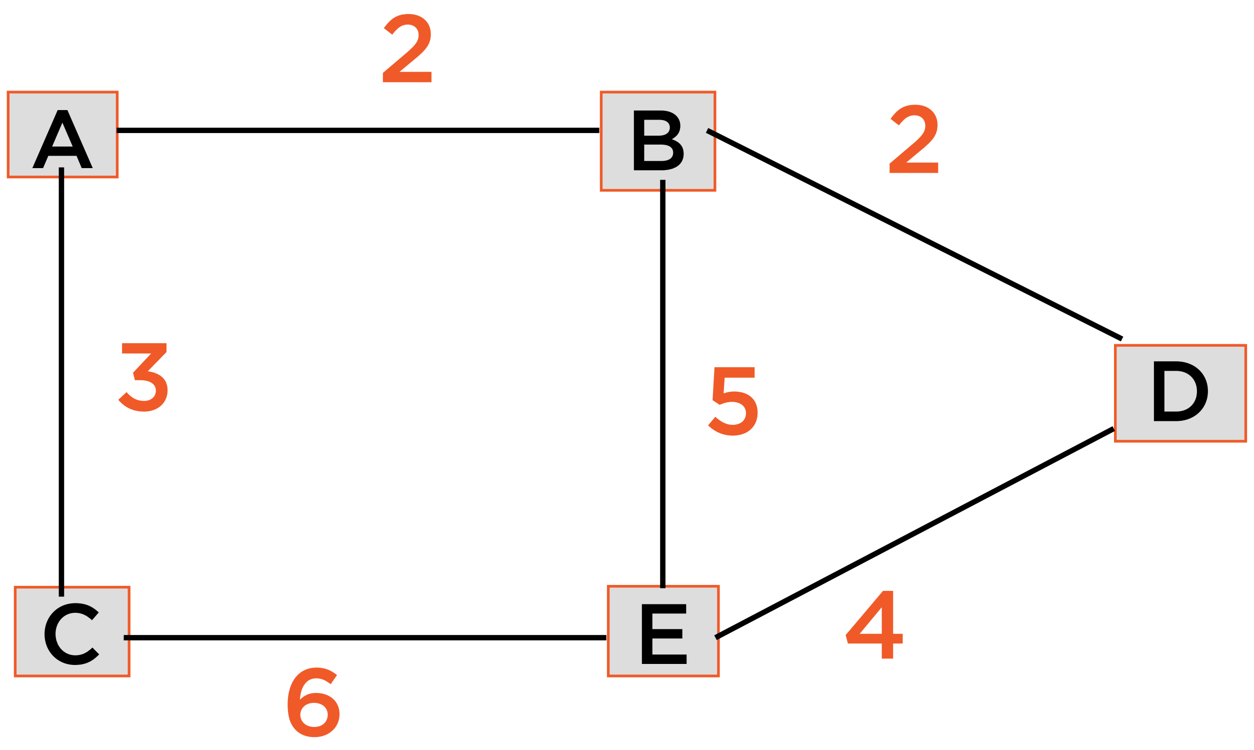
Edge	Weight
E - D	4
B - E	5
C - E	6



Result

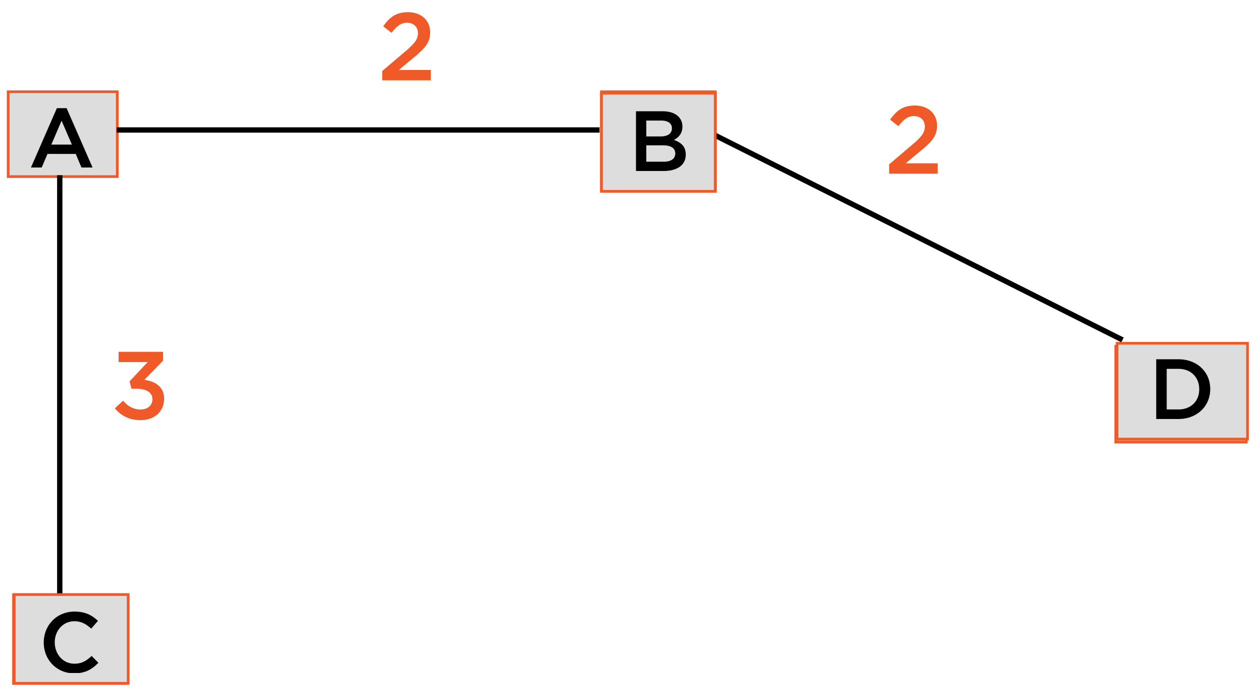
Edge	Weight
A - B	2
B - D	2
A - C	3

Kruskal’s Algorithm



Priority Queue

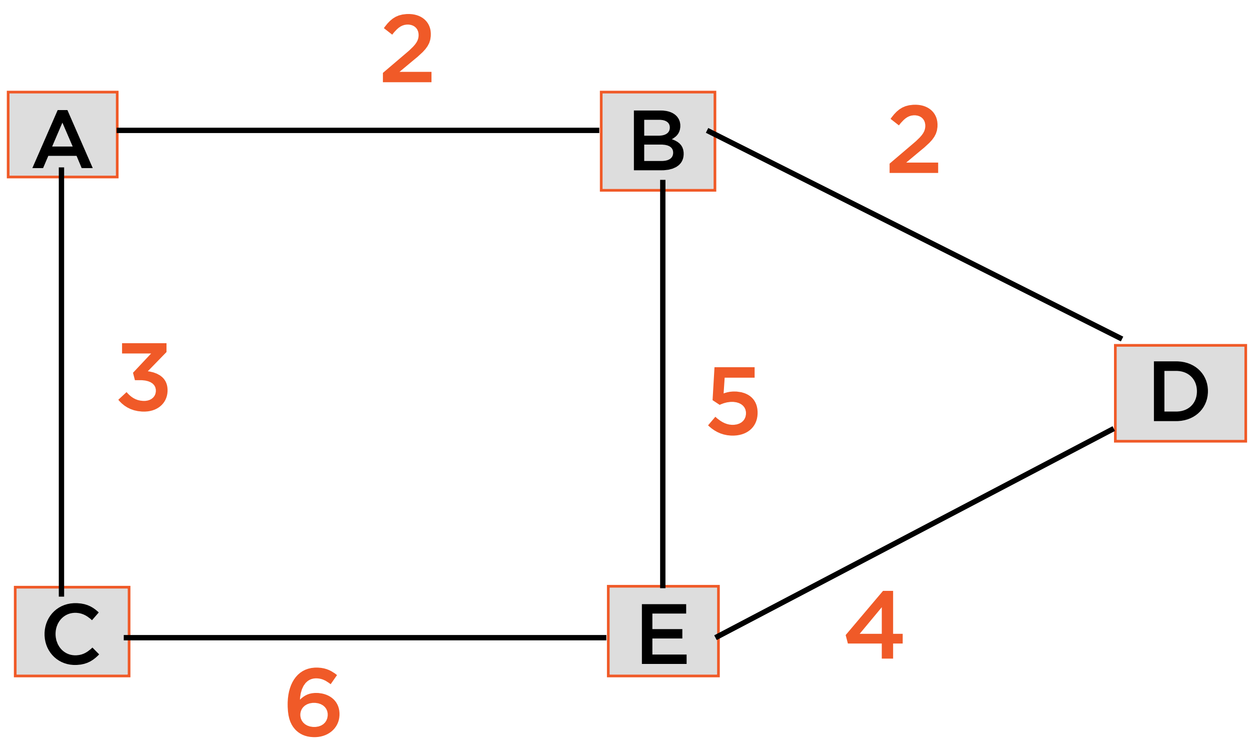
Edge	Weight
E - D	4
B - E	5
C - E	6

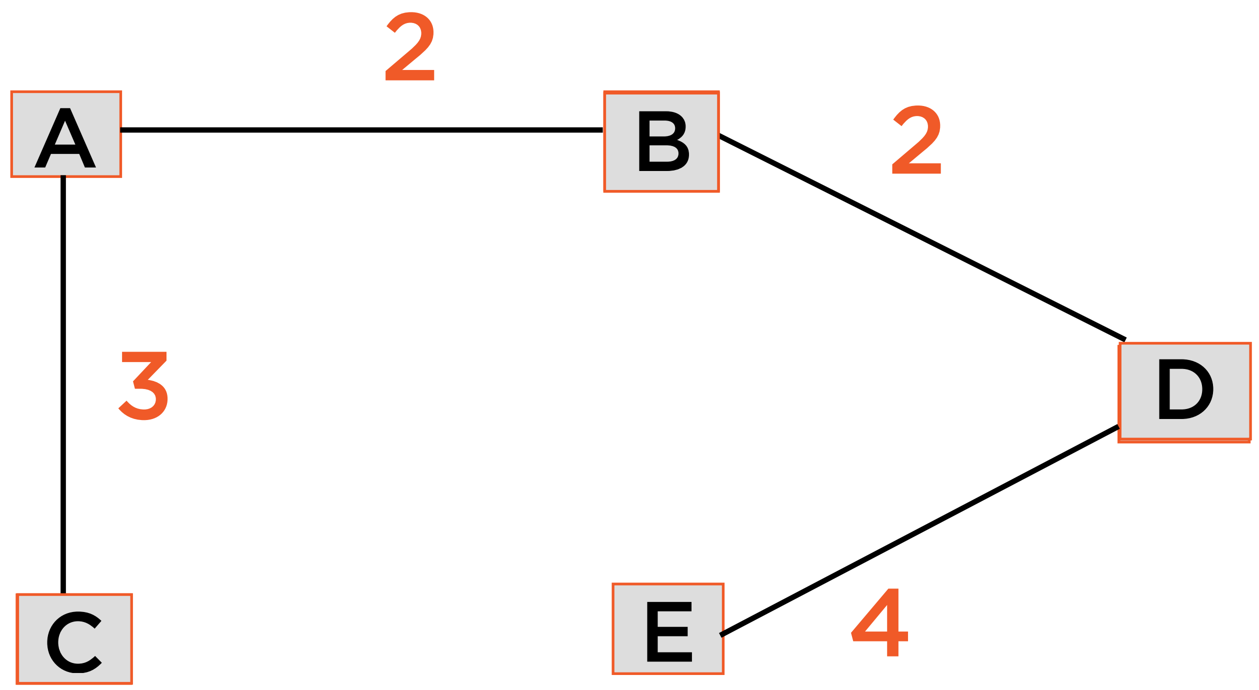


Result

Edge	Weight
A - B	2
B - D	2
A - C	3

Kruskal's Algorithm





Priority Queue

Edge	Weight
B - E	5
C - E	6

Result

Edge	Weight
A - B	2
B - D	2
A - C	3
D - E	4

Graph has 5 nodes, result has 4 edges

Kruskal’s Algorithm

Sort edges

Increasing order of weights

Can use priority queue

Initialize empty result

Empty set of edges

At end will hold minimum spanning tree

Find shortest edge

Not currently in result

Dequeue from priority queue

Reject if cycle introduced

Else add to result set

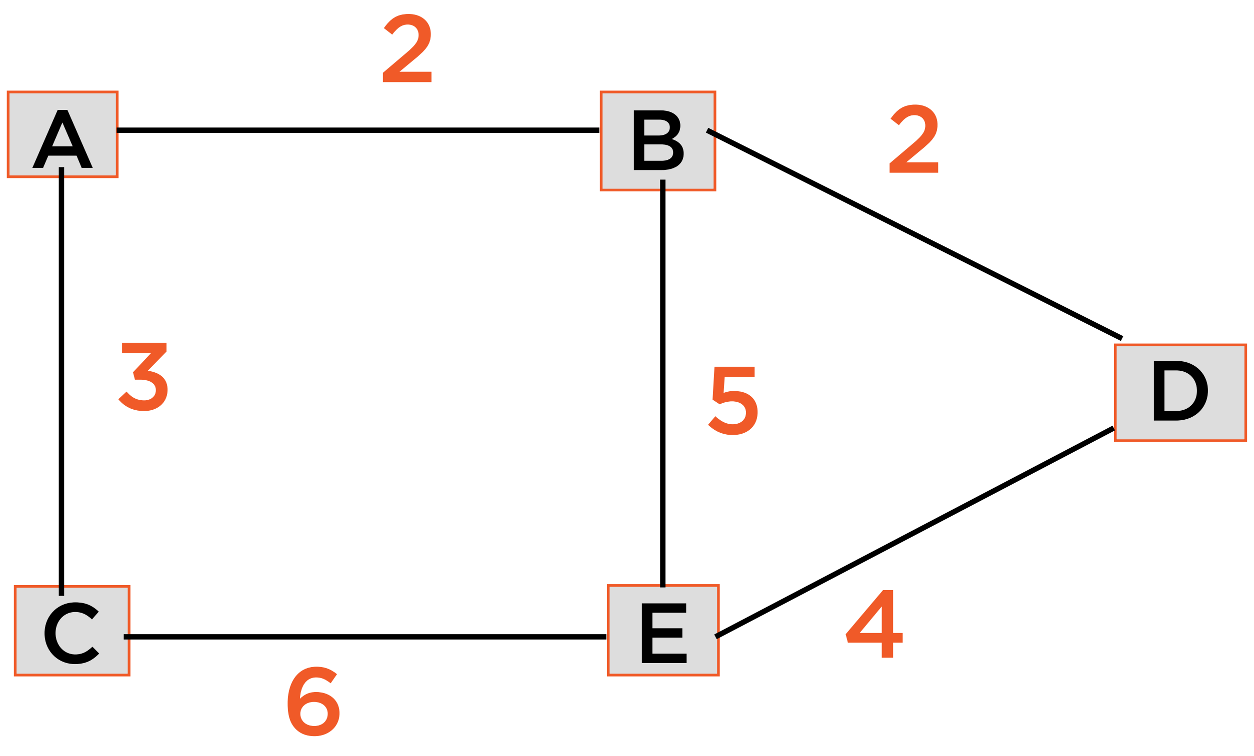
This is a greedy step

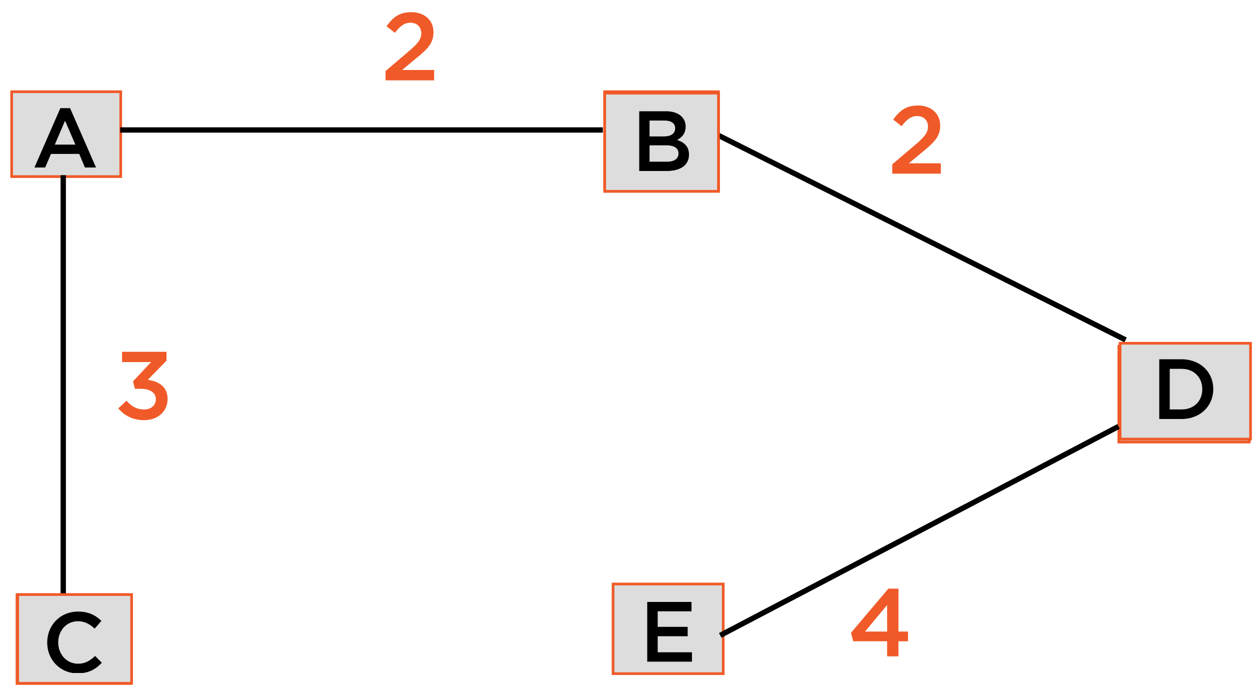
Stop

When N-1 edges in result

N = number of vertices in graph

Kruskal's Algorithm





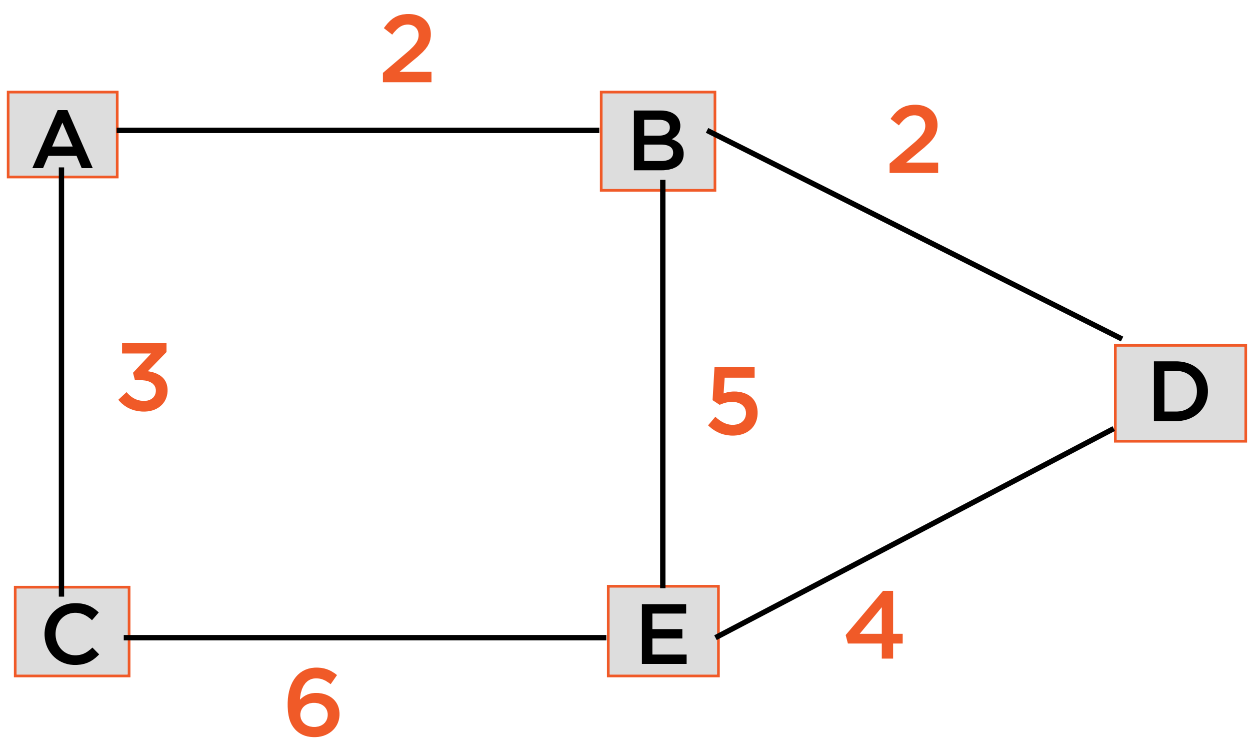
Priority Queue

Edge	Weight
B - E	5
C - E	6

Result

Edge	Weight
A - B	2
B - D	2
A - C	3
D - E	4

Kruskal's Algorithm

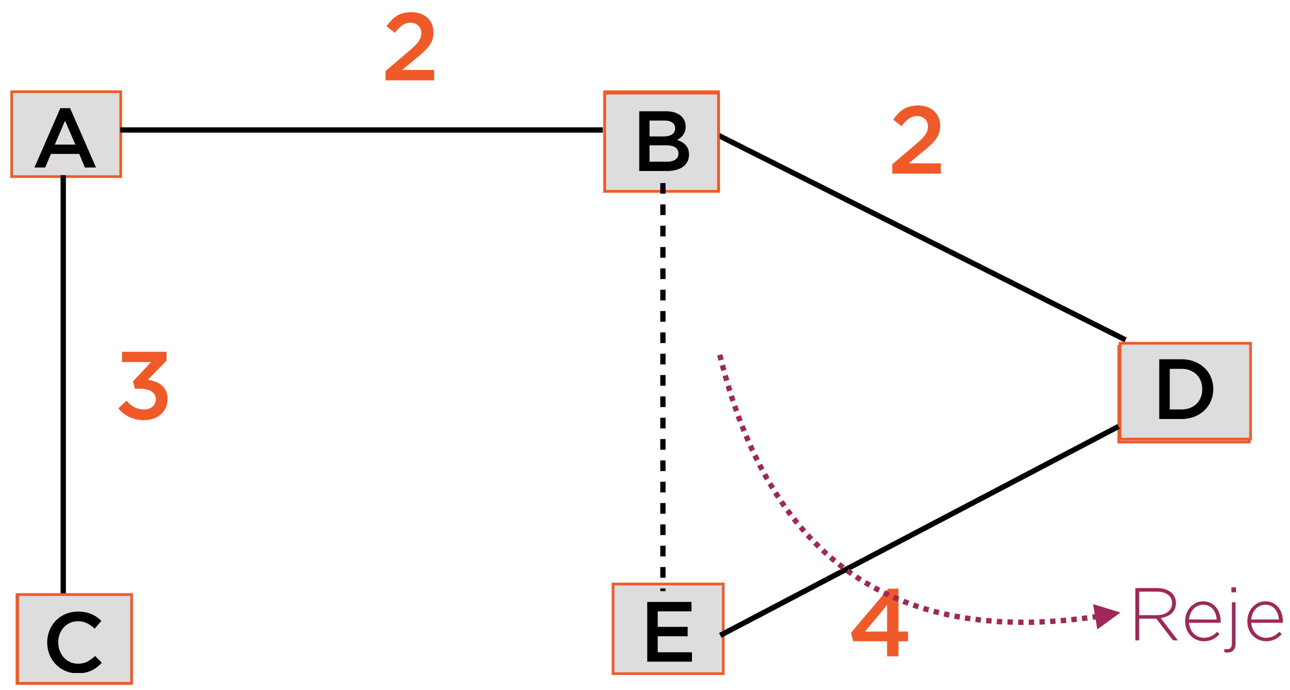


Priority Queue

Edge	Weight
B - E	5
C - E	6

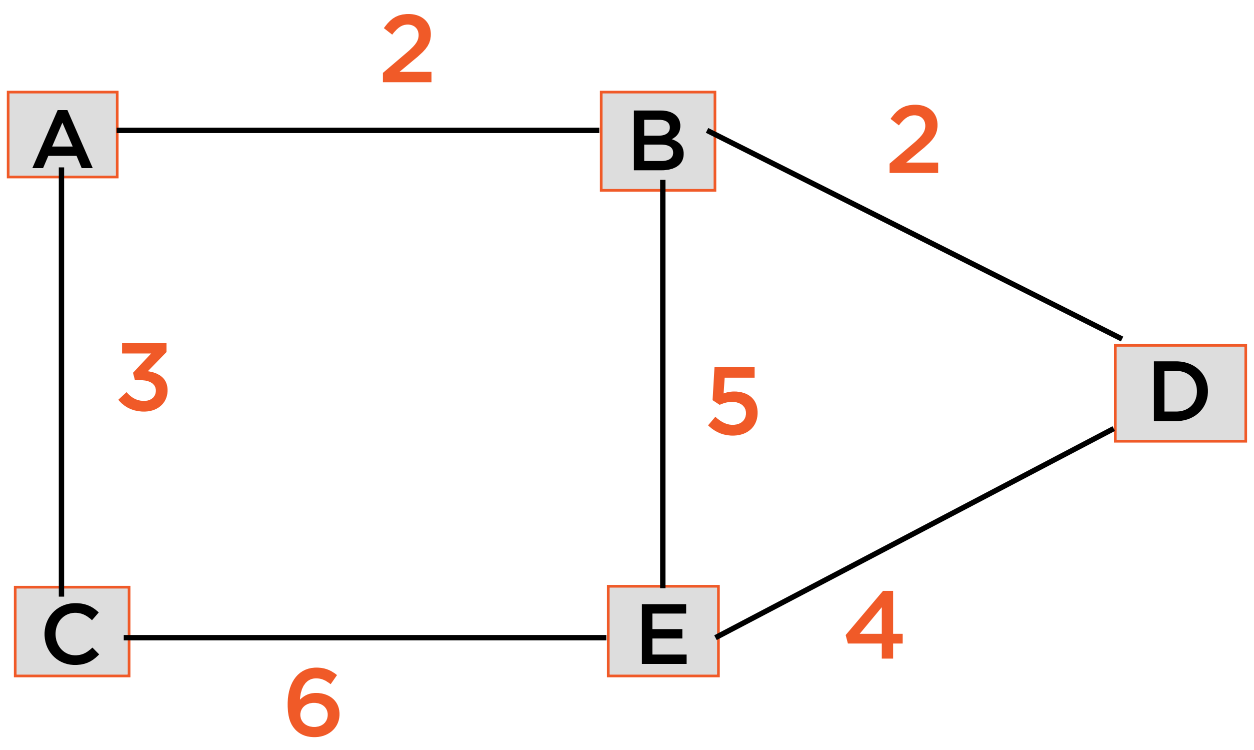
Result

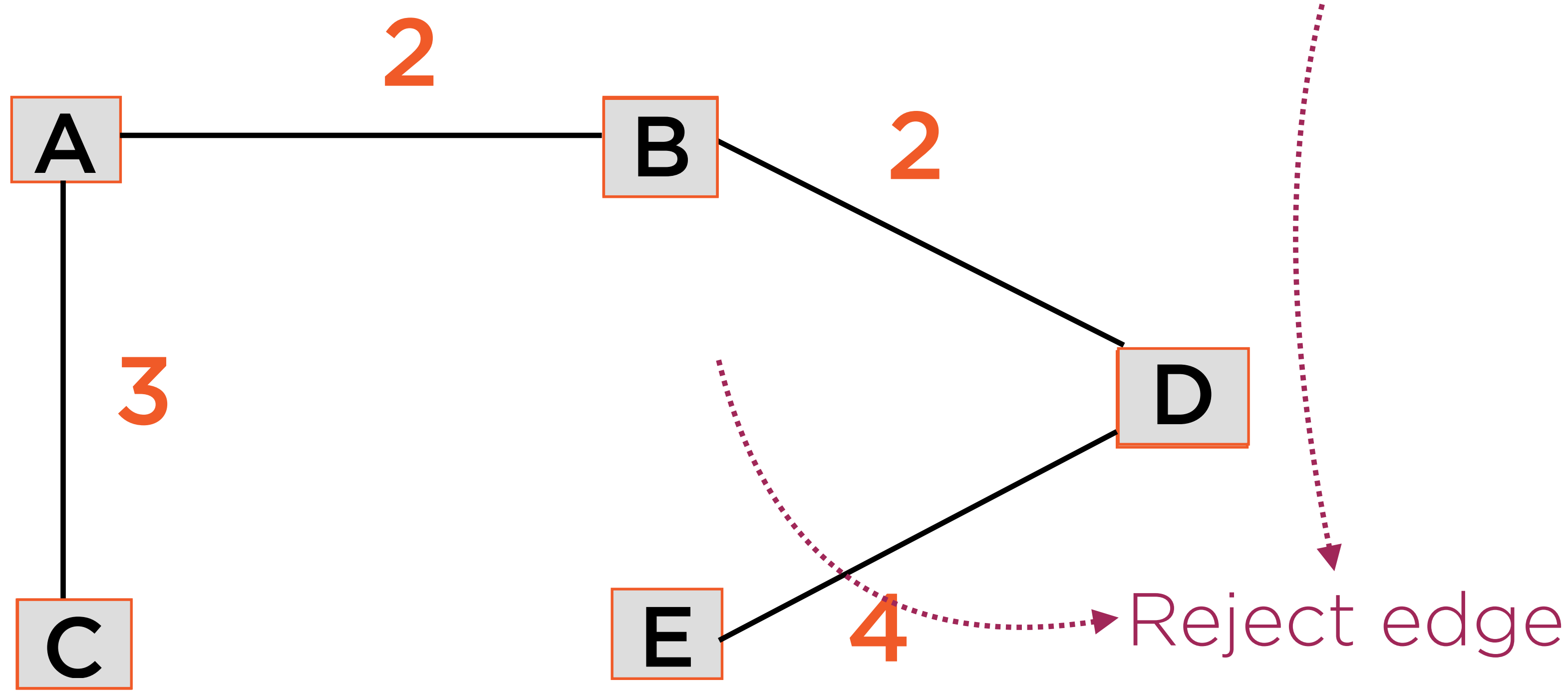
Edge	Weight
A - B	2
B - D	2
A - C	3
D - E	4



Reject edge

Kruskal’s Algorithm





Priority Queue

Edge	Weight
B - E	5
C - E	6

Result

Edge	Weight
A - B	2
B - D	2
A - C	3
D - E	4

Reject any edge in the minimal spanning tree which causes a **cycle**

Kruskal's Algorithm

Sort edges

Increasing order of weights

Can use priority queue

Initialize empty result

Empty set of edges

At end will hold minimum spanning tree

Find shortest edge

Not currently in result

Dequeue from priority queue

Reject if cycle introduced

Else add to result set

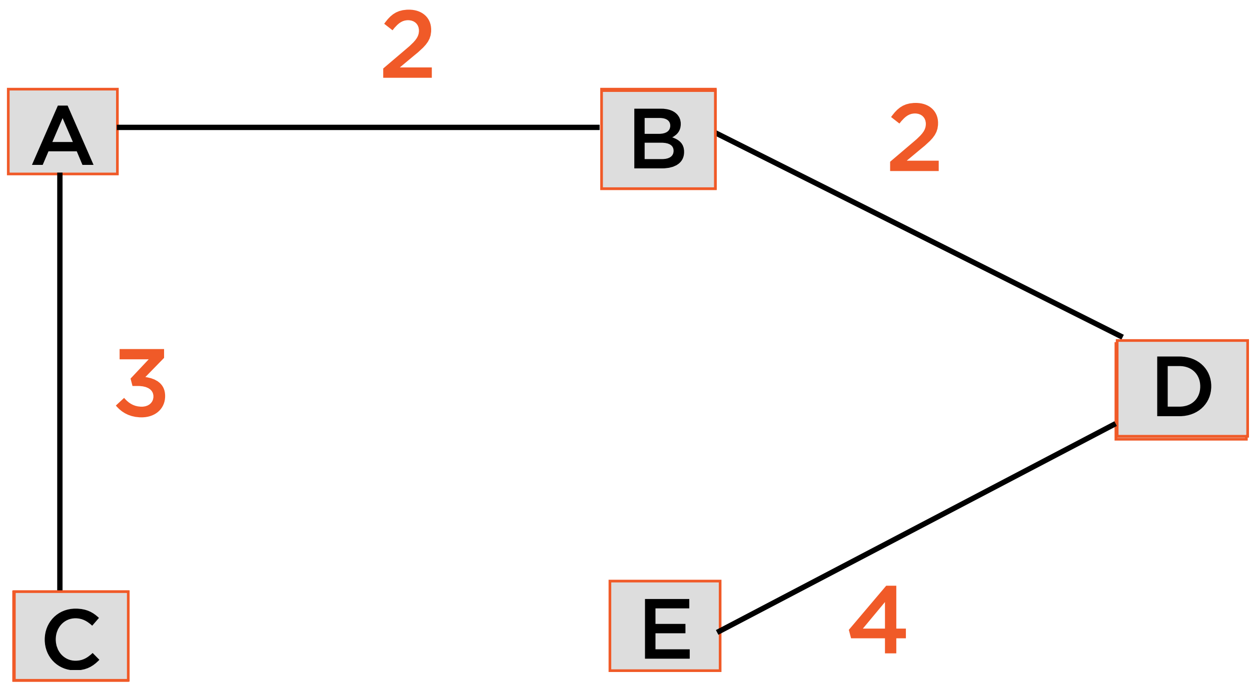
This is a greedy step

Stop

When N-1 edges in result

N = number of vertices in graph

Kruskal’s Algorithm



Result

Edge	Weight
A - B	2
B - D	2
A - C	3
D - E	4

Minimum spanning tree found, weight = 11

Kruskal's Algorithm

Algorithm does not consider edges in contiguous order

Benefit: Works for disconnected graphs too

Drawback: Intermediate result is not necessarily a tree

Kruskal’s Algorithm

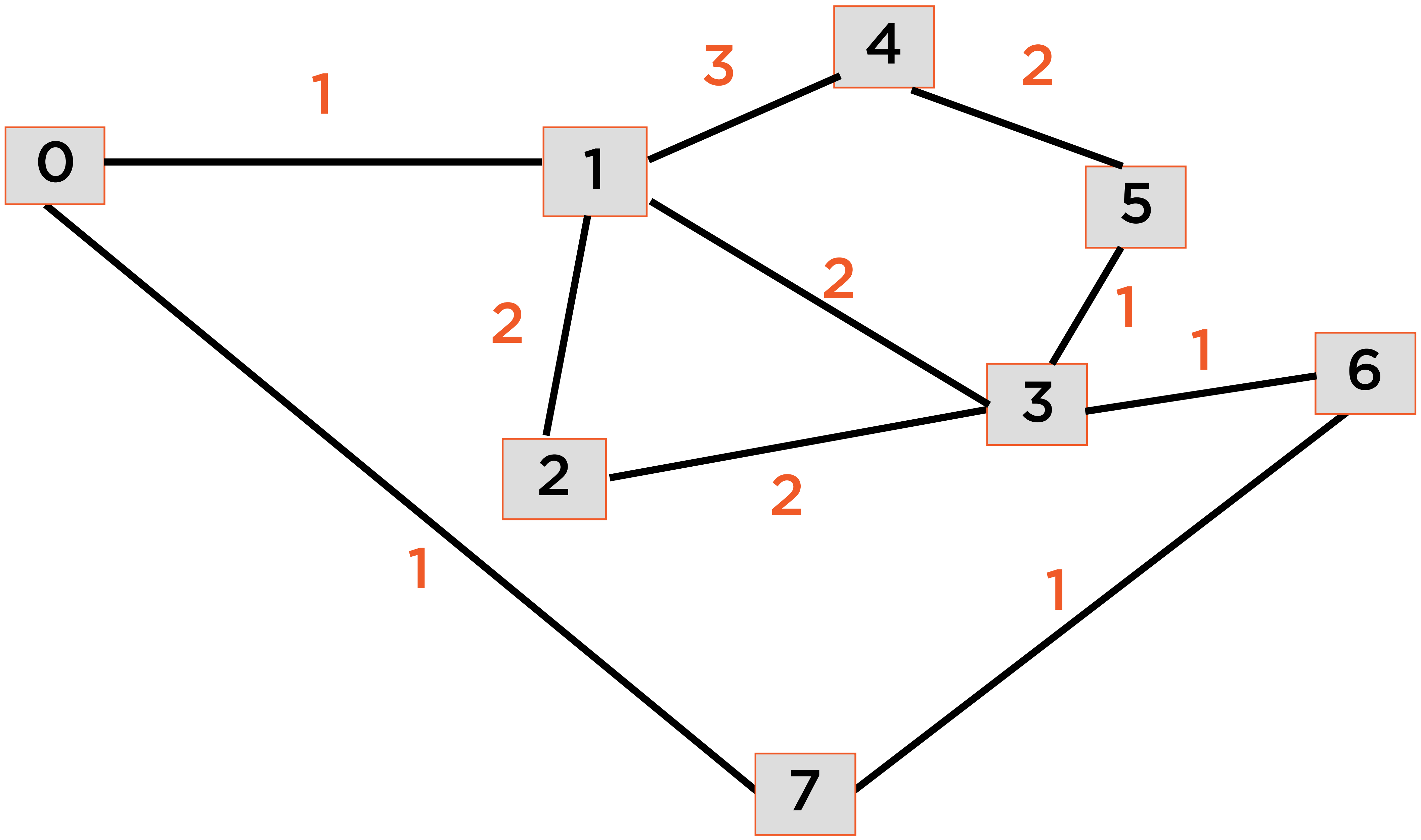
Sorting the edges dominates the running time

O(E ln(E))

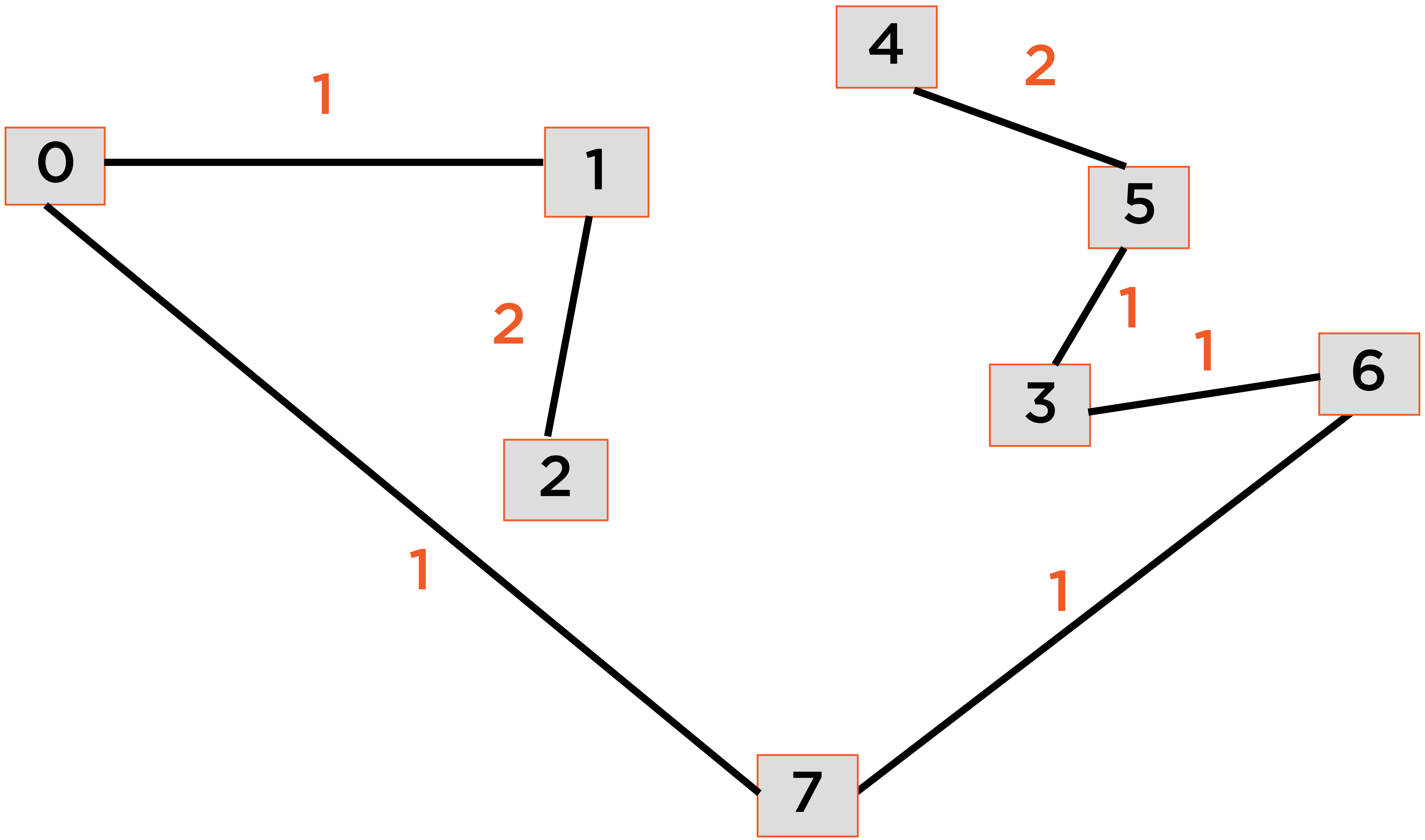
Demo

Implement Kruskal's algorithm for a minimal spanning tree

A Sample Undirected Graph



A Sample Undirected Graph



Summary

Spanning tree algorithms seek to find the shortest way to cover all nodes

Such algorithms are used when start and end nodes do not matter

Prim's algorithm works for connected graphs

Kruskal's algorithm works even for disconnected graphs