CS 2009
Design and Analysis of Algorithms

Waheed Ahmed
Email : waheedahmed@nu.edu.pk

LAST TIME

- Greedy algorithms! Three examples:
  - Activity selection (greedy choice: pick activity with earliest finish time)
  - Coin Change (greedy choice: take the largest possible bill or coin that does not overshoot)
  - Fractional Knapsack (greedy choice: select item with highest value/weight value until bag is full)

THE GREEDY PARADIGM

Commit to choices one-at-a-time,

never look back,
and hope for the best.

Greedy doesn't always work.

WHAT WE'LL COVER TODAY

- Applications of the greedy algorithm design paradigm to **Minimum Spanning Trees**
  - Prim's algorithm
  - Kruskal's algorithm

MINIMUM SPANNING TREES

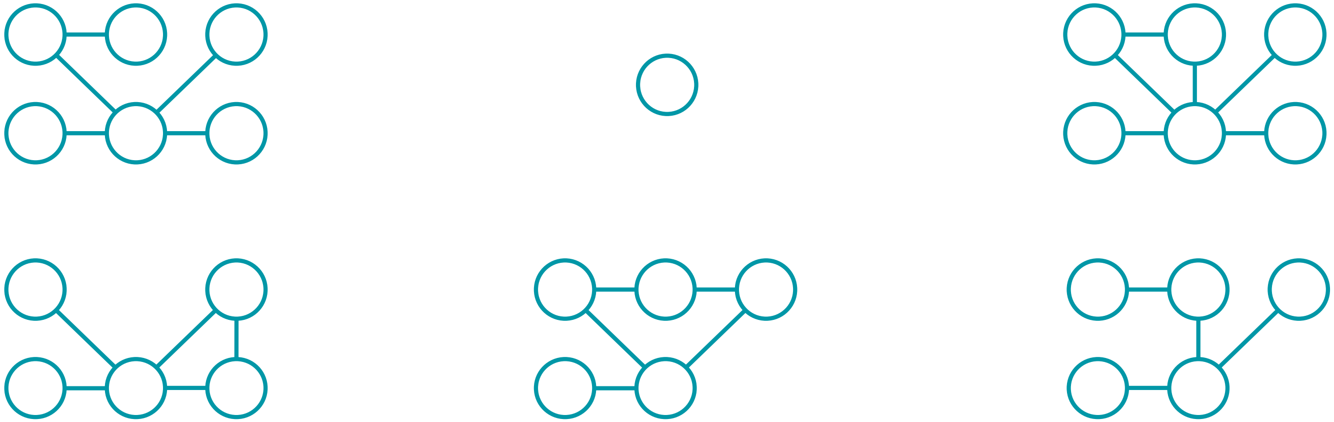
What are minimum spanning trees (MSTs)?

TREES IN GRAPHS

Let's go over some terminology that we'll be using today.

A tree is an undirected, *acyclic*, connected graph.

Which of these graphs are trees?

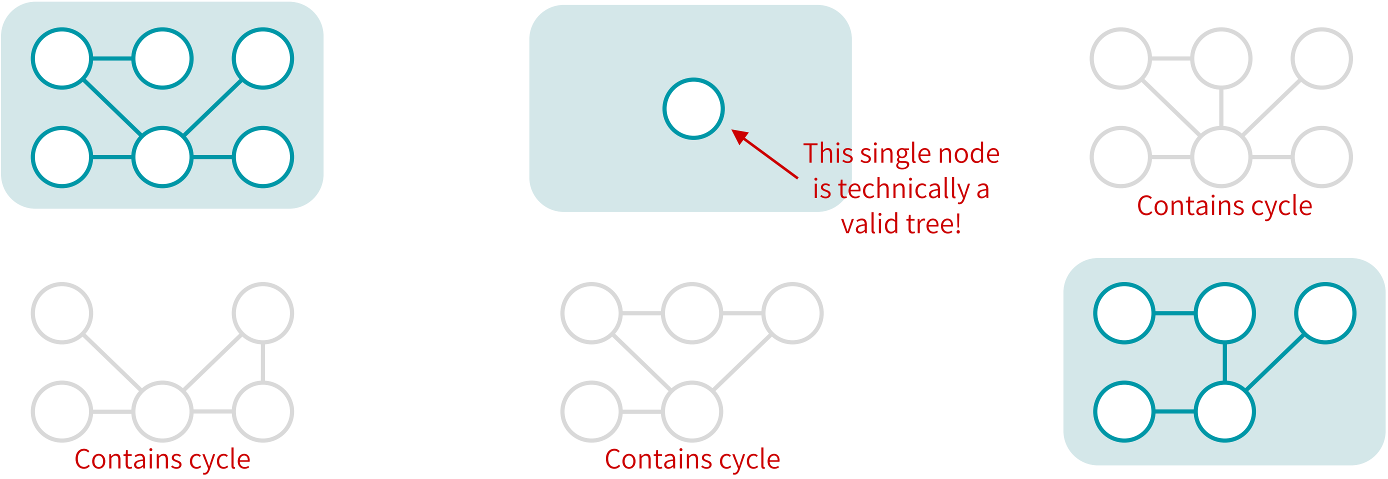


TREES IN GRAPHS

Let’s go over some terminology that we’ll be using today.

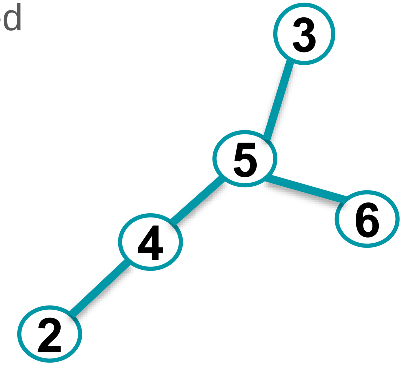
A tree is an undirected, *acyclic*, connected graph.

Which of these graphs are trees?



TREES IN UNIDIRECTED GRAPHS?

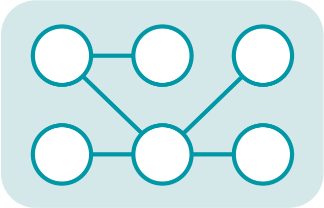
- However, in undirected graphs, there is another definition of trees
- Tree
    - A undirected graph (V, E), where E is the set of undirected edges
    - All vertices are connected
    - $|E|=|V|-1$



SPANNING TREES

A spanning tree is a tree that connects all of the vertices

Which of these graphs are spanning trees?



Doesn’t connect all vertices

Not a tree

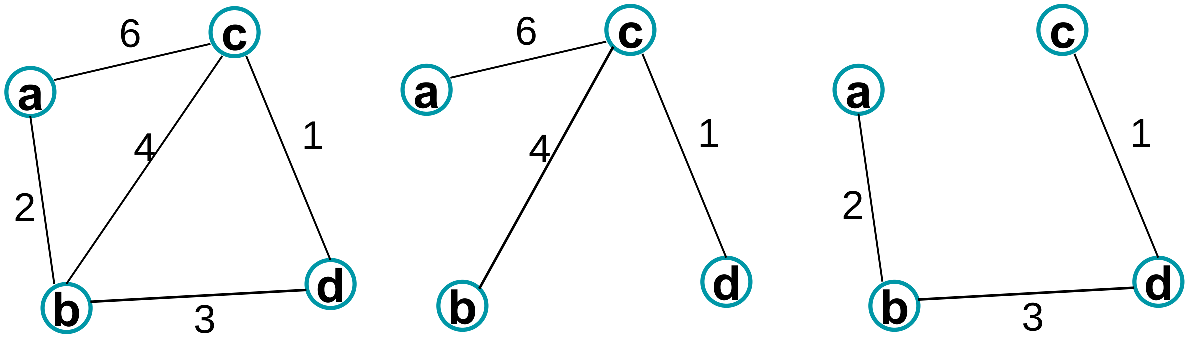
Not a tree

Not a tree

Doesn’t connect all vertices

Examples of MST

Example:



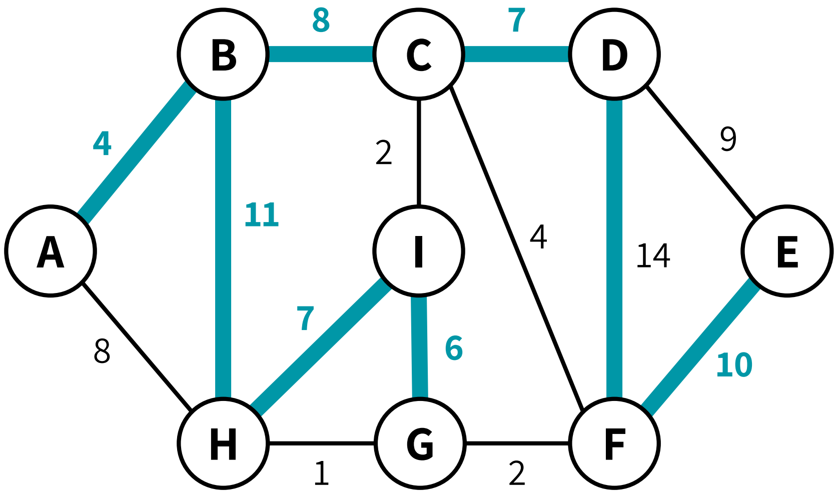
MINIMUM SPANNING TREES (MSTs)

For the remainder of today, we're going to work with **undirected, weighted, connected graphs.**

The **cost of a spanning tree** is the **sum of the weights on the edges**.

An **MST** of a graph is a spanning tree of the graph with minimum cost.

Note: A graph may have multiple spanning trees.
It may also have multiple MSTs
(if 2 different spanning trees have the same exact cost)



This spanning tree has a cost of **67**.

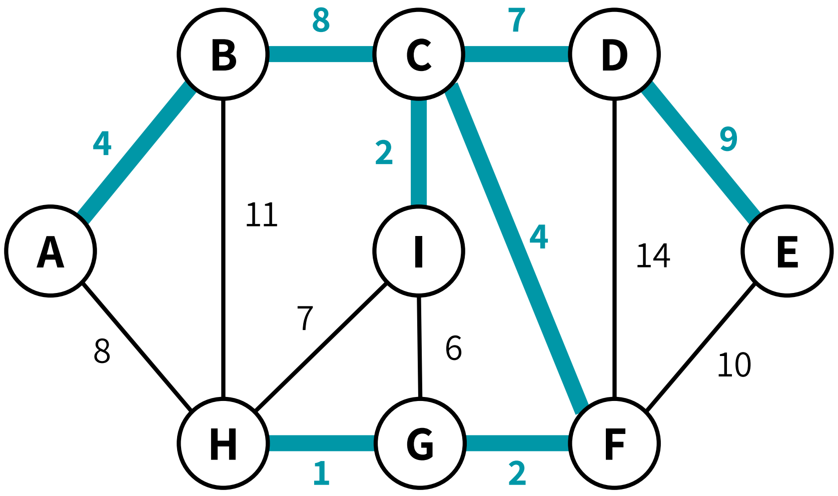
MINIMUM SPANNING TREES (MSTs)

For the remainder of today, we're going to work with **undirected, weighted, connected graphs.**

The **cost of a spanning tree** is the **sum of the weights on the edges**.

An **MST** of a graph is a spanning tree of the graph with minimum cost.

Note: A graph may have multiple spanning trees. It may also have multiple MSTs (if 2 different spanning trees have the same exact cost)

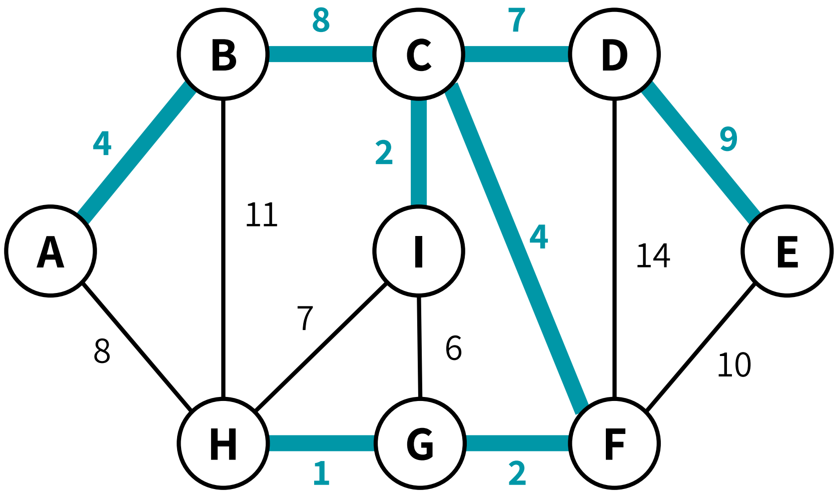


This spanning tree has a cost of **37**.

This is an MST of this graph, since there is no other spanning tree with smaller cost.

MINIMUM SPANNING TREES (MSTs)

The task for today:
Given an undirected, weighted, and connected graph G,
find the minimum spanning tree (as a subset of the G's edges)



We would return this MST.
Sometimes, there may be more than one MST as well, so return any MST of G.

APPLICATIONS OF MSTs

Network design

Find the most cost-effective way to connect cities with roads/water/electricity/phone

Image processing

Image segmentation, which finds connected regions in the image with minimal differences

Cluster analysis

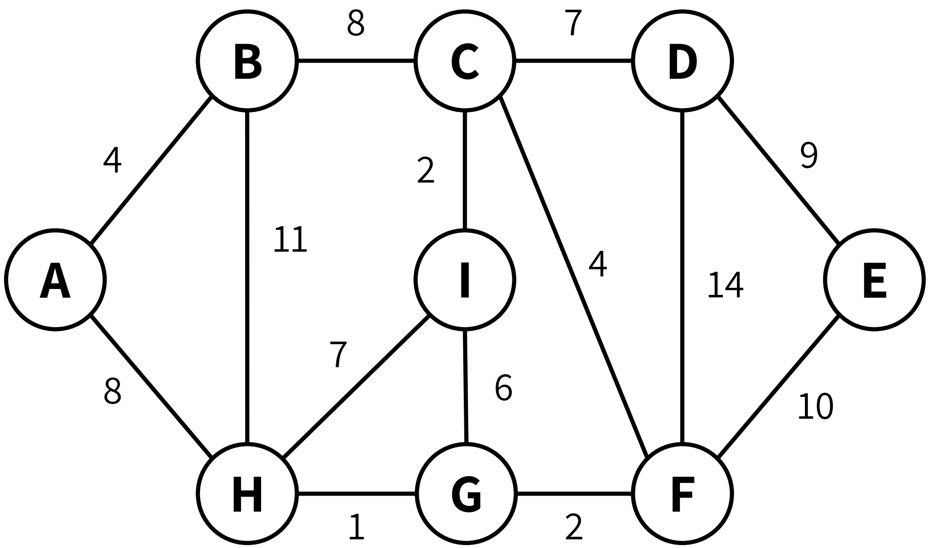
Find clusters in a dataset (one of the algorithms we'll see today can be modified slightly to basically do this)

Useful primitive

Finding an MST is often useful as a subroutine or approximation for more advanced graph algorithms

MINIMUM SPANNING TREES (MSTs)

Brainstorm some greedy algorithms to find an MST!



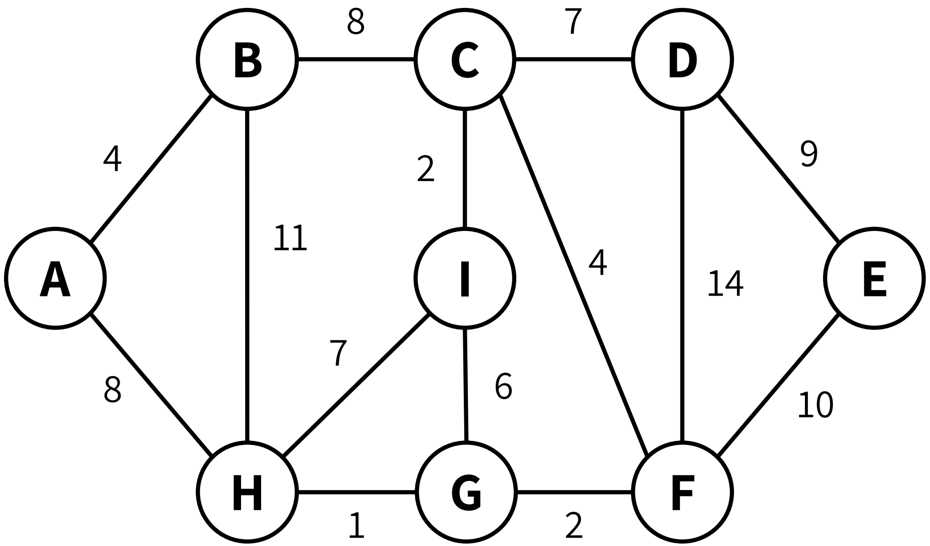
PRIM'S ALGORITHM

Greedily add the closest vertex!

PRIM'S ALGORITHM: THE IDEA

Greedy choice:

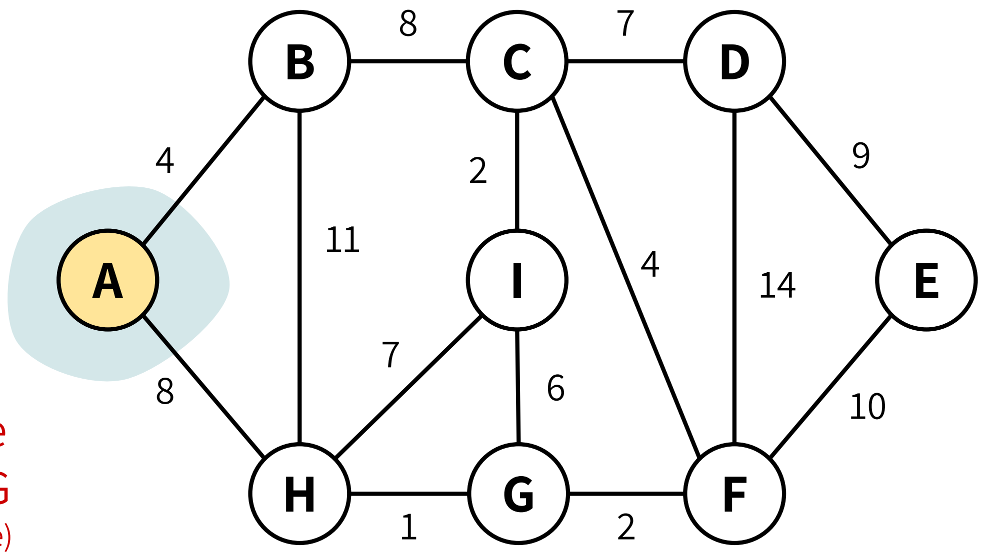
Grow a single tree, & greedily add the shortest edge that could grow our tree



PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree

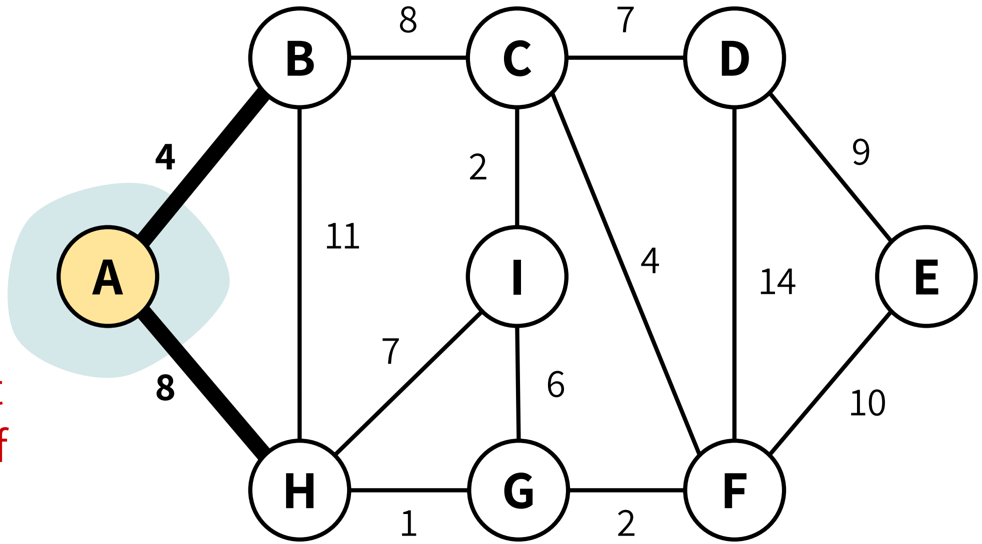


First, we can initialize our tree to contain a single arbitrary node in G
(doesn't matter which node)

PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree



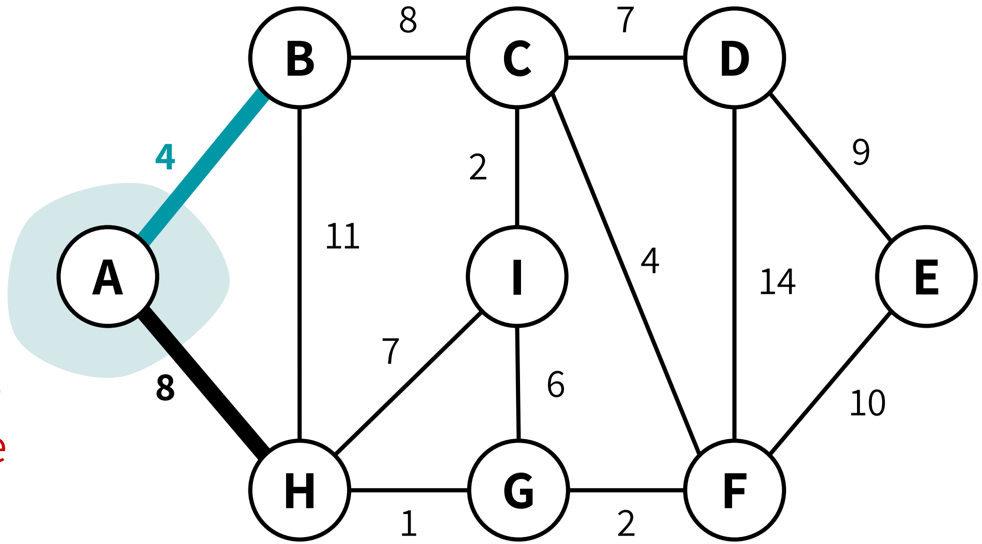
Consider the edges coming out of the "frontier" of our growing tree.

PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree

Claim the edge coming out of the "frontier" with the smallest weight

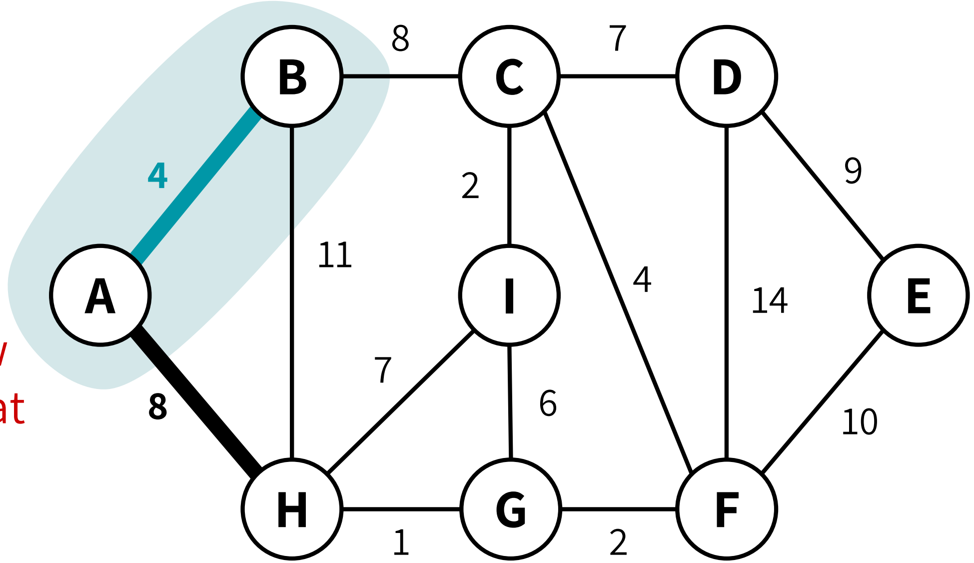


PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree

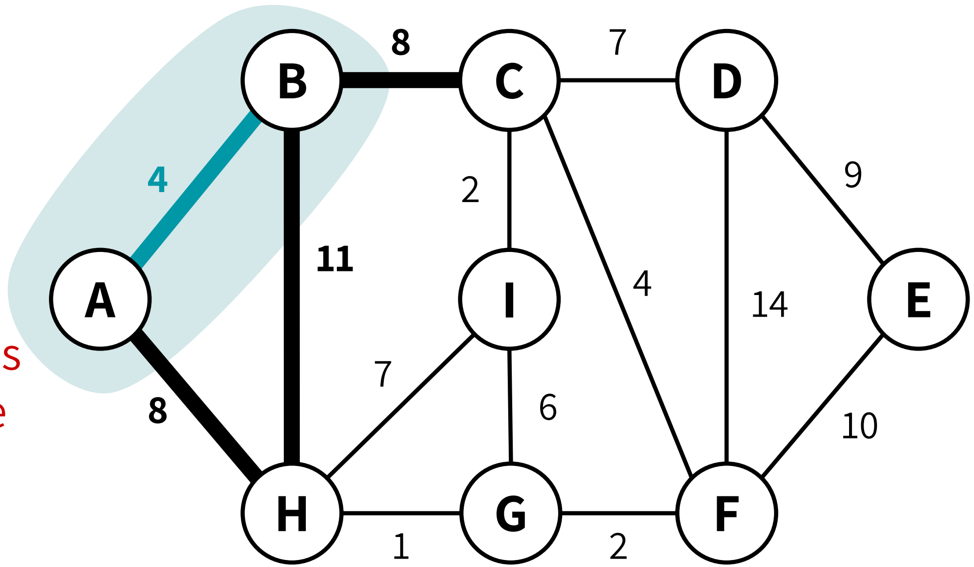
Our tree just grew by one! Now repeat until we reach all the nodes.



PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree

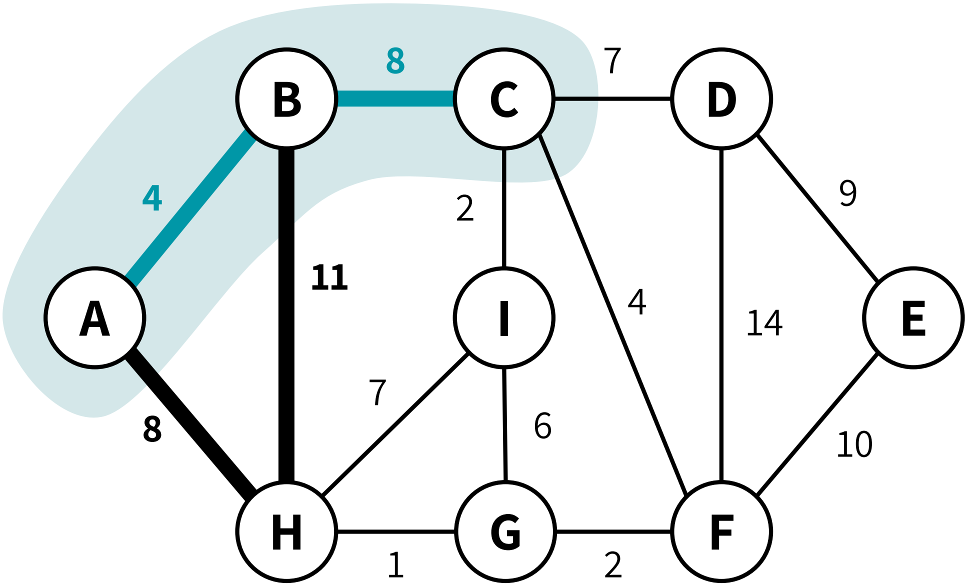


Consider the edges coming out of the "frontier" of our growing tree.

PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree



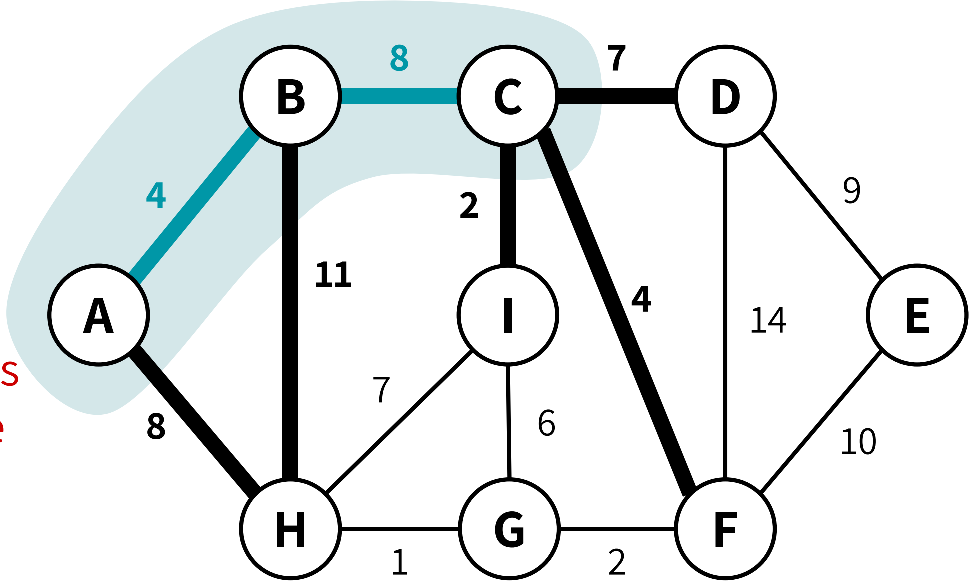
Claim the edge coming out of the "frontier" with the smallest weight

(if there's a tie, choose any)

PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree

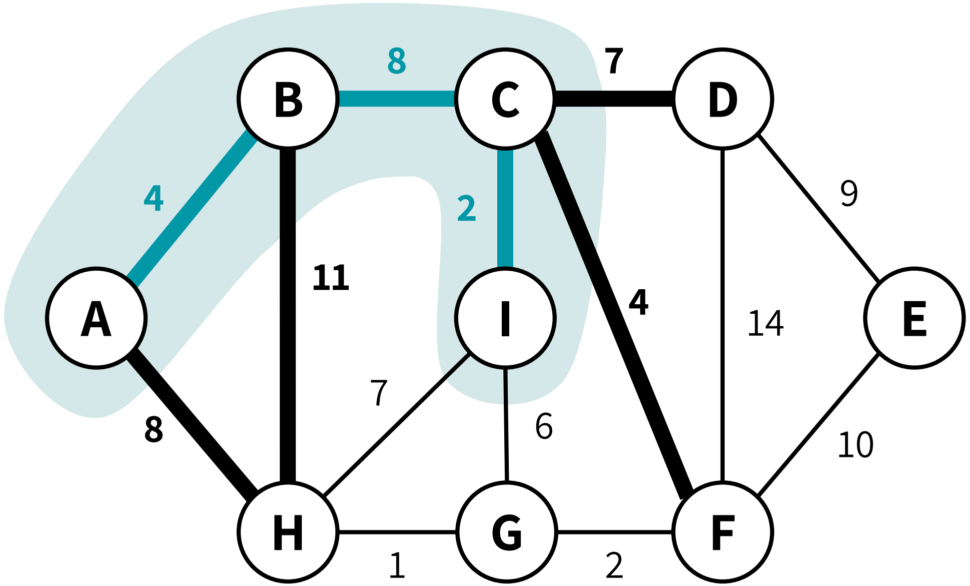


Consider the edges coming out of the "frontier" of our growing tree.

PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree

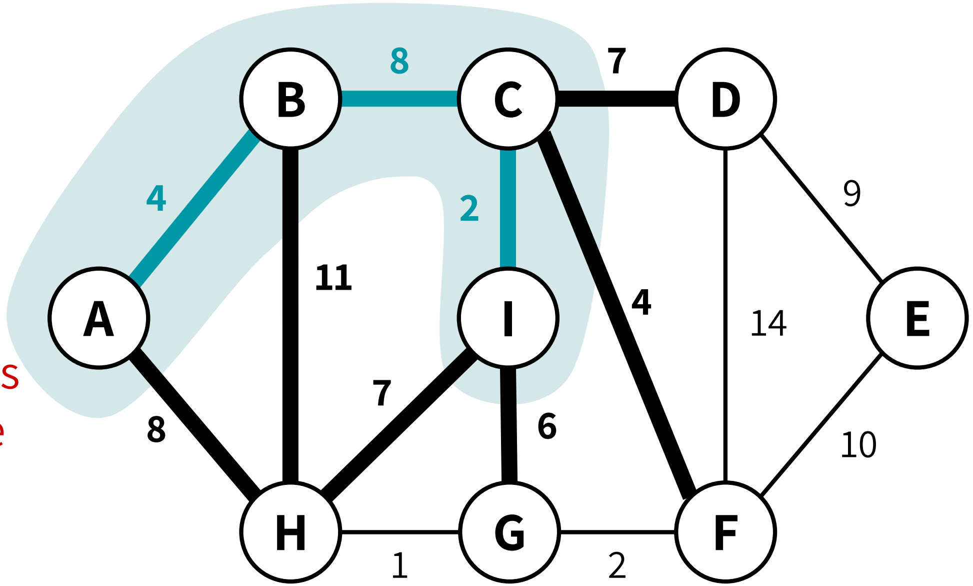


Claim the edge coming out of the "frontier" with the smallest weight

PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree



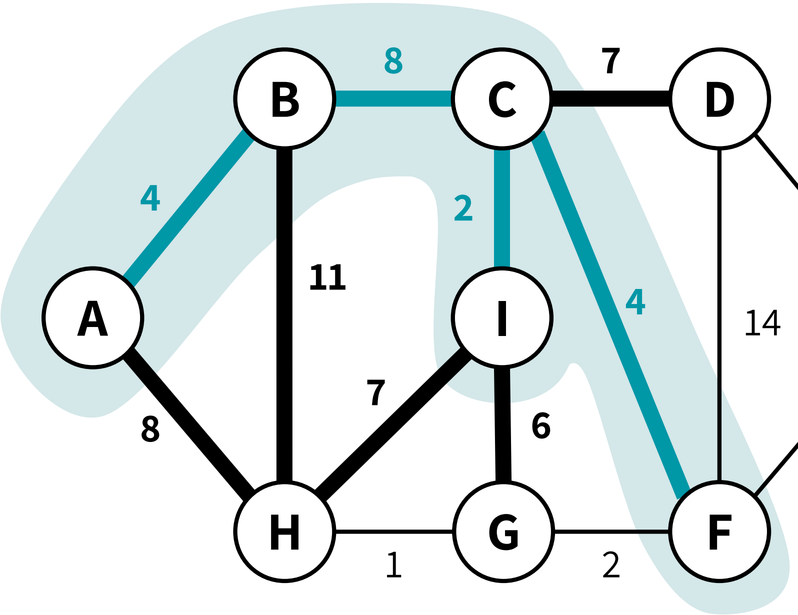
Consider the edges coming out of the "frontier" of our growing tree.

PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree

Claim the edge coming out of the "frontier" with the smallest weight

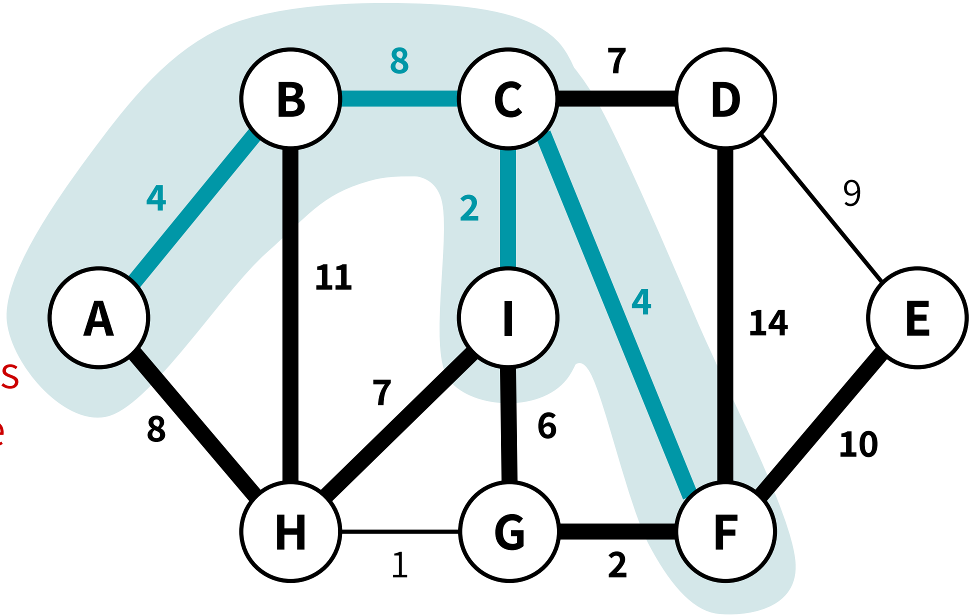


PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree

Consider the edges coming out of the "frontier" of our growing tree.

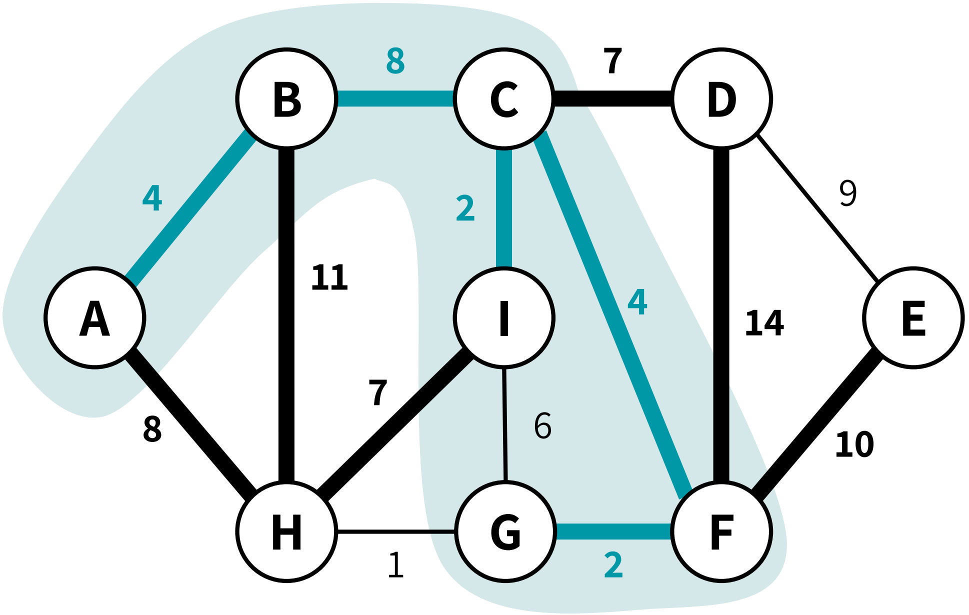


PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree

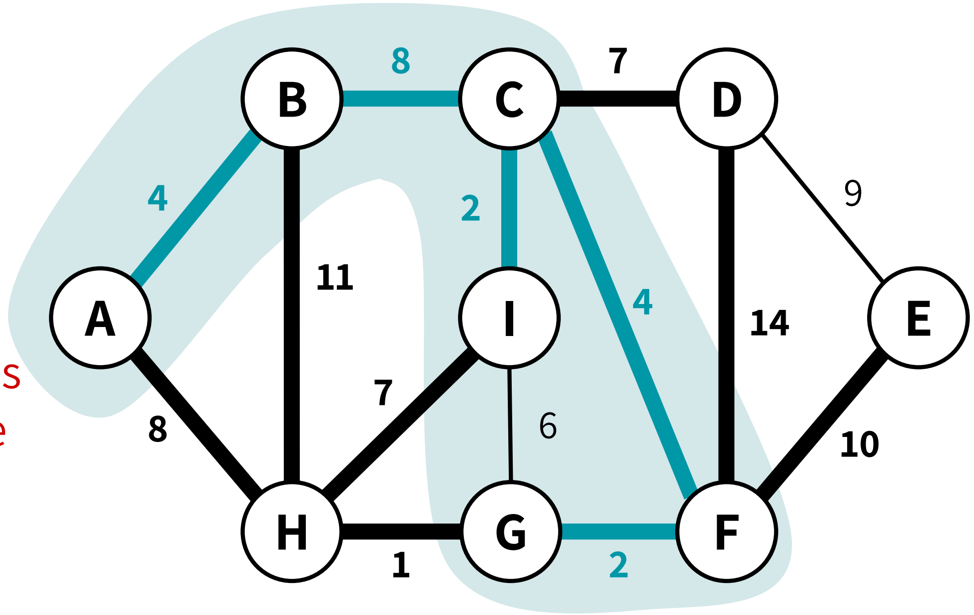
Claim the edge coming out of the "frontier" with the smallest weight



PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree



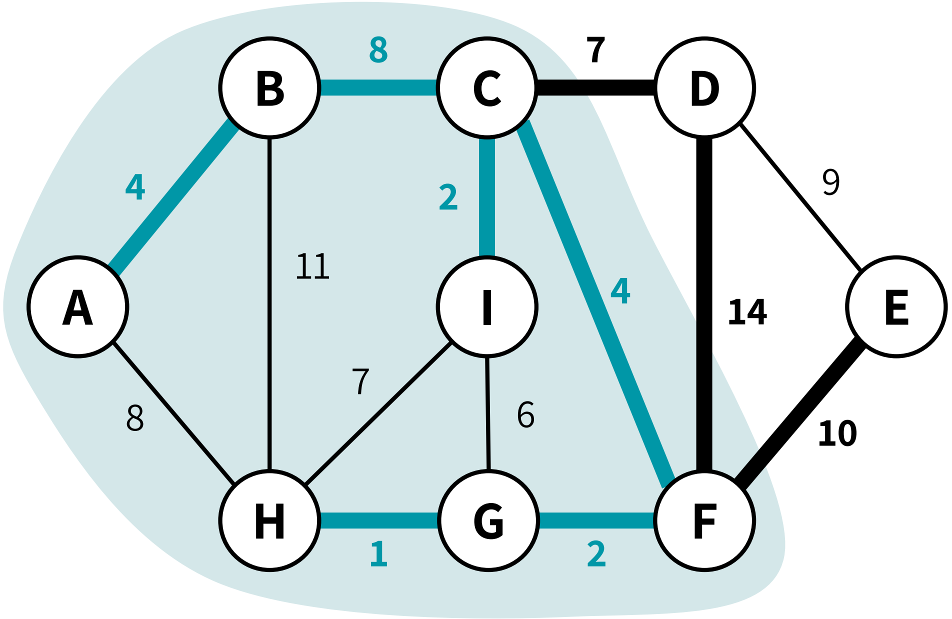
Consider the edges coming out of the "frontier" of our growing tree.

PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree

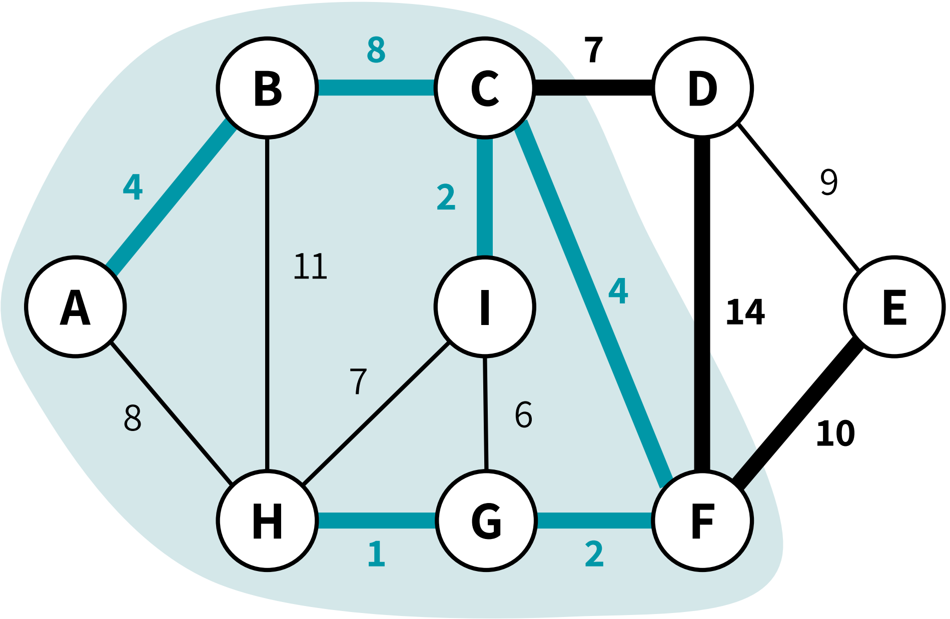
Claim the edge coming out of the "frontier" with the smallest weight



PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree



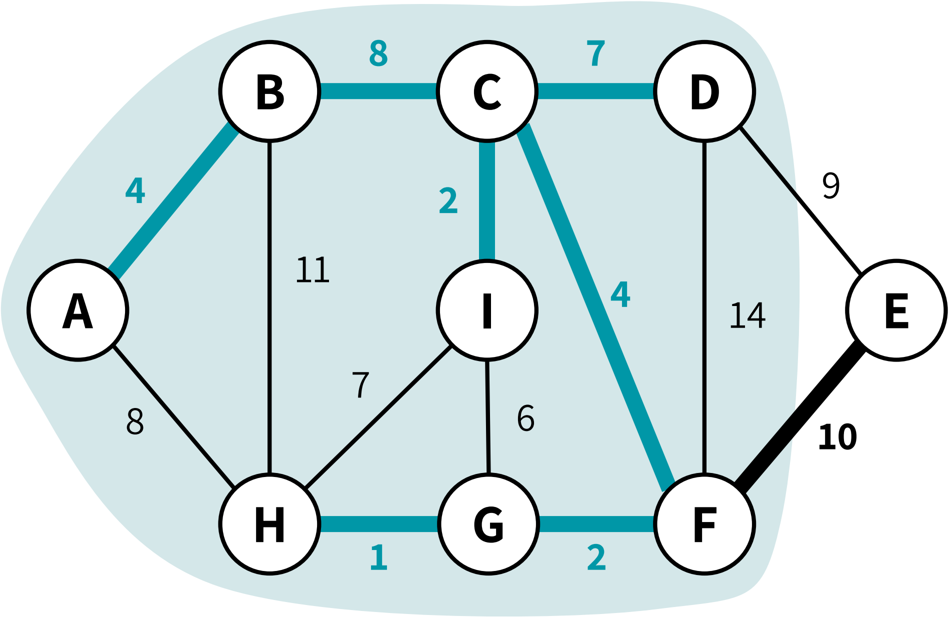
Consider the edges coming out of the "frontier" of our growing tree.

PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree

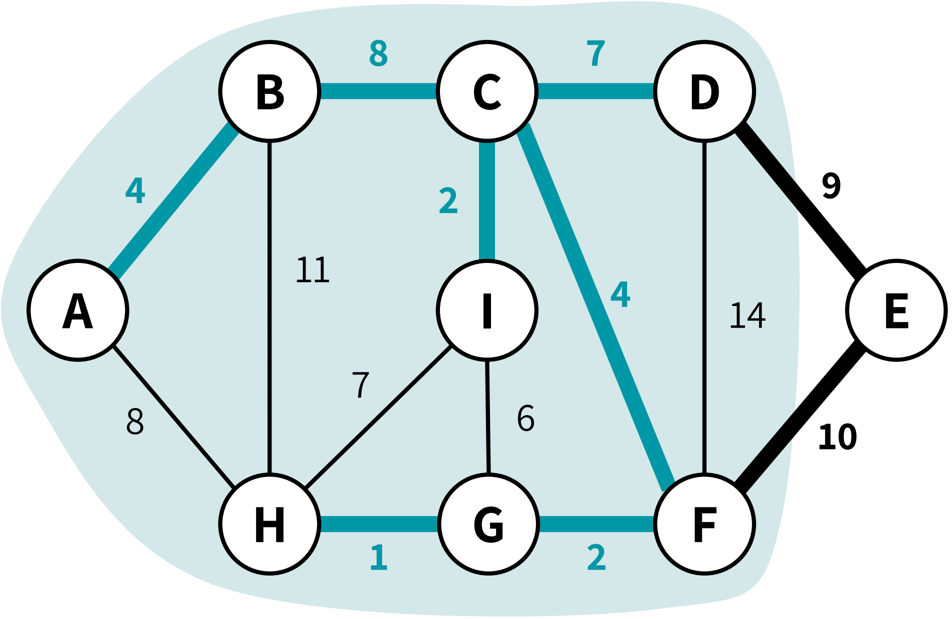
Claim the edge coming out of the "frontier" with the smallest weight



PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree



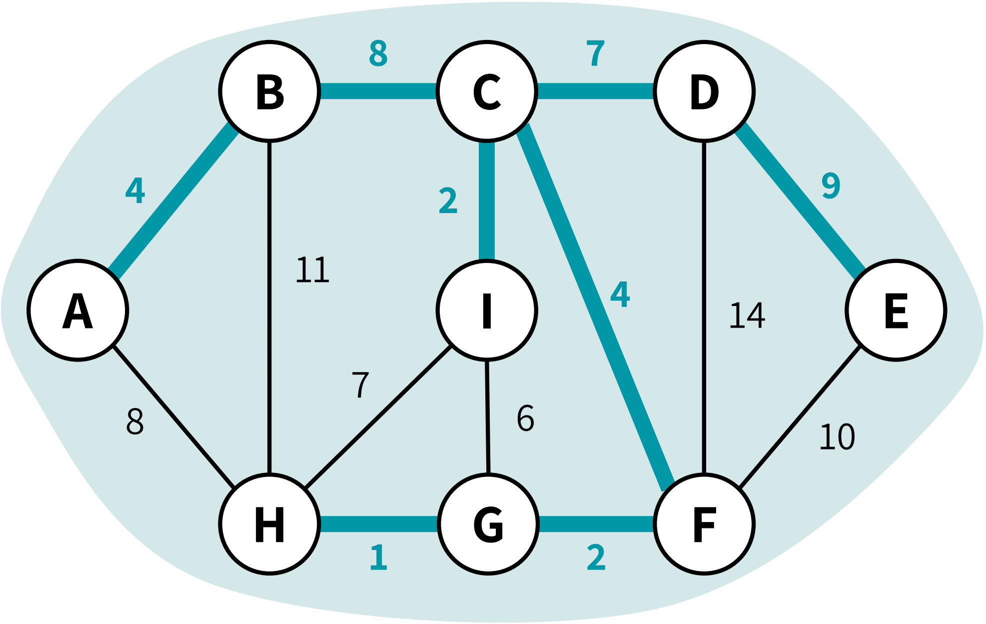
Consider the edges coming out of the "frontier" of our growing tree.

PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree

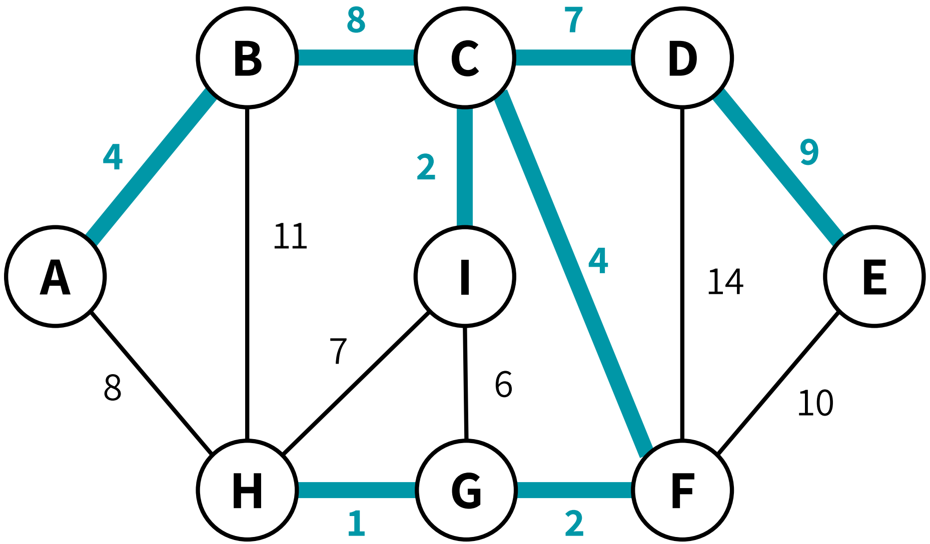
Claim the edge coming out of the "frontier" with the smallest weight



PRIM'S ALGORITHM: THE IDEA

Greedy choice:

Grow a single tree, & greedily add the shortest edge that could grow our tree



And we're done!

This is our MST.

(with weight 37)

CLRS textbook version PSEUDOCODE For PRIM'S ALGORITHM

```
MST-PRIM(G, w, r)
1   for each u ∈ G.V
2       u.key = ∞
3       u.π = NIL
4   r.key = 0
5   Q = G.V
6   while Q ≠ ∅
7       u = EXTRACT-MIN(Q)
8       for each v ∈ G.Adj[u]
9           if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

v.key is the minimum weight of any edge connecting v to a vertex in the tree.

v.key = ∞ if there is no such edge.

The attribute **v.π** names the parent of v in the tree.

Runtime (Build Min heap line 1-5): O(V)

(while loop excute |V| and EXTRACT-MIN log V): O(V log V)

For loop line 8-11: O (E)

Total Prim Algo Runtime = O (V log V + E log V) = O (E log V) ???

PSEUDOCODE For PRIM'S ALGORITHM

```
MST-PRIM(G, w, r)
1   for each vertex u ∈ G.V
2       u.key = ∞
3       u.π = NIL
4   r.key = 0
5   Q = ∅
6   for each vertex u ∈ G.V
7       INSERT(Q, u)
8   while Q ≠ ∅
9       u = EXTRACT-MIN(Q)          // add u to the tree
10      for each vertex v in G.Adj[u]   // update keys of u's non-tree neighbors
11          if v ∈ Q and w(u, v) < v.key
12              v.π = u
13              v.key = w(u, v)
14              DECREASE-KEY(Q, v, w(u, v))
```

v.key is the minimum weight of any edge connecting v to a vertex in the tree.

v.key = ∞ if there is no such edge.

The attribute **v.π** names the parent of v in the tree.

Prim's algorithm operates much like Dijkstra's algorithm.

Runtime (Build Min heap line 1-7): O(V)

(while loop excute |V| and EXTRACT-MIN log V): O(V log V)

For loop line 10-12: O (E)

Total Prim Algo Runtime = O (V log V + E log V) = O (E log V) ???

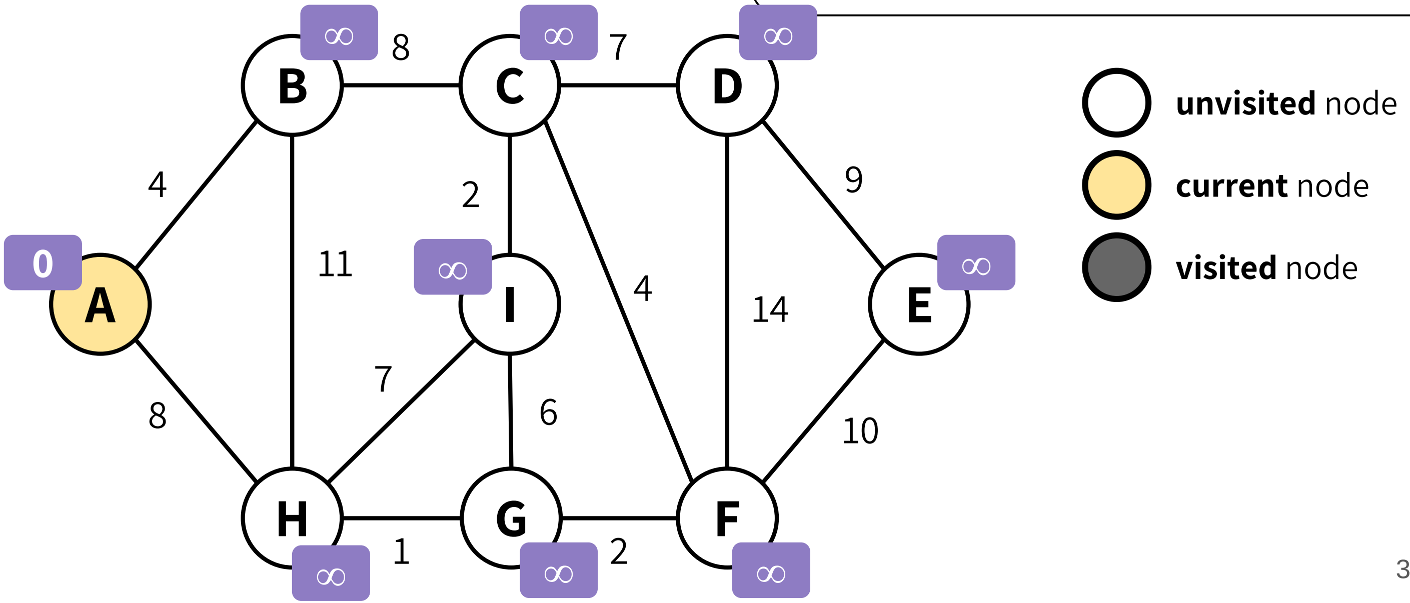
HOW DO WE IMPLEMENT THIS?

Each vertex that's not yet reached by the growing tree keeps track of:

1) the **distance** from itself to the growing spanning tree using *one edge*
2) **how to get to there** (the closest neighbor that's reached by the tree already)

PRIM(G = (V,E), s):

for all v besides s: **d[v] =** ∞ and **k[v] = NULL**



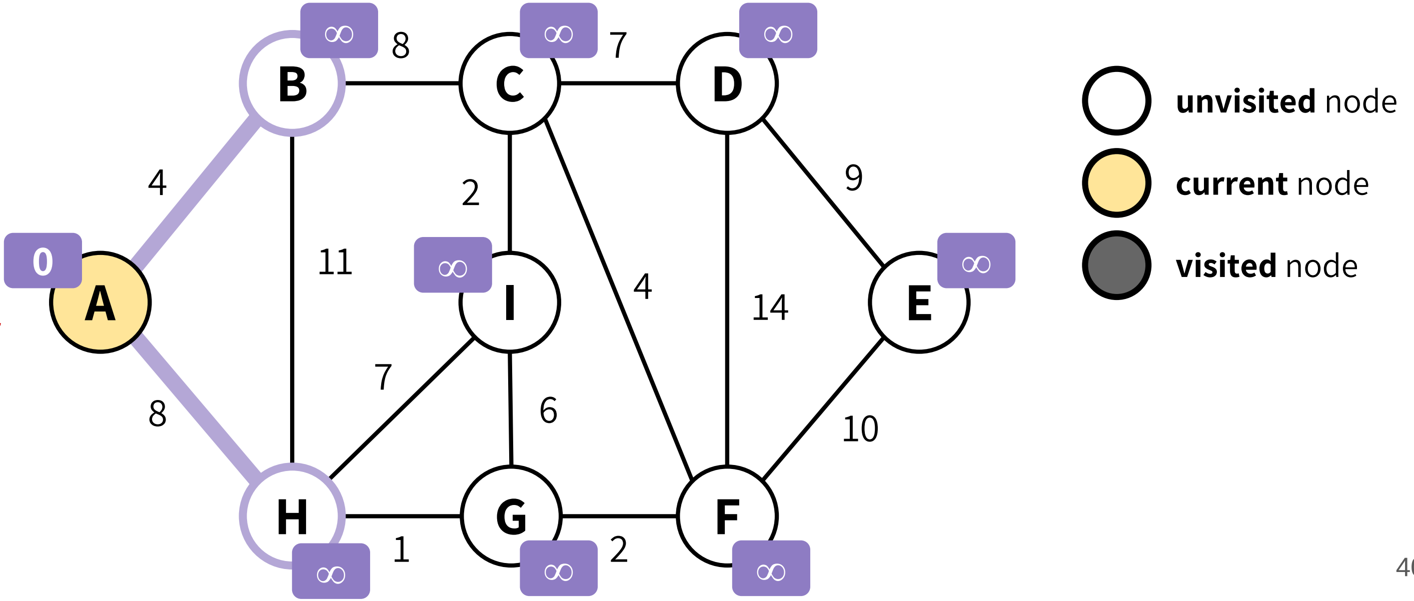
A is part of the growing tree first

HOW DO WE IMPLEMENT THIS?

Each vertex that's not yet reached by the growing tree keeps track of:

1) the **distance** from itself to the growing spanning tree using *one edge*
2) **how to get to there** (the closest neighbor that's reached by the tree already)

Now that A got added, see if any of its neighbors are closer to the tree because of it!

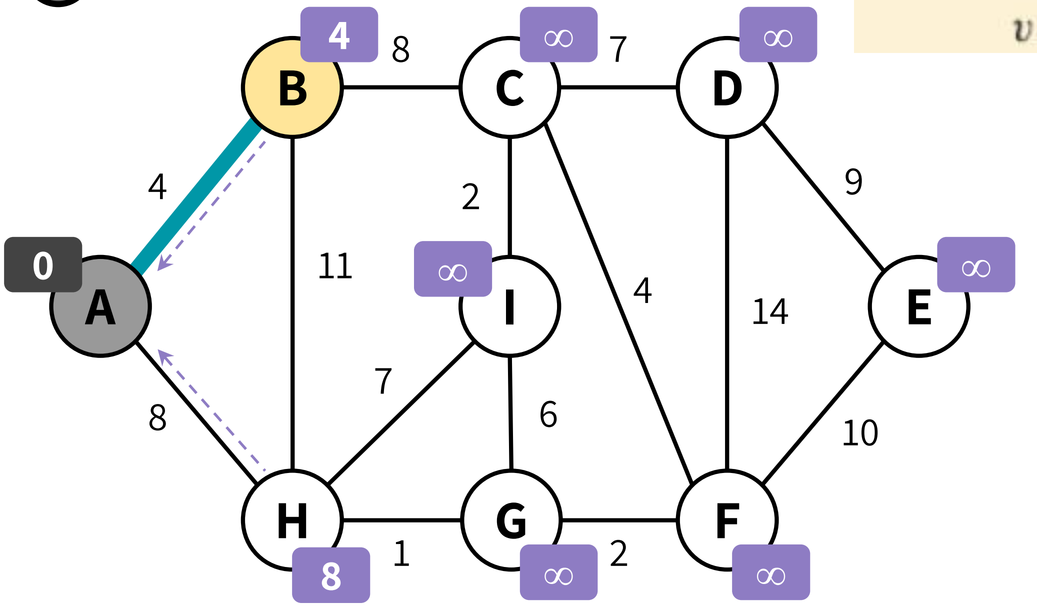


HOW DO WE IMPLEMENT THIS?

unvisited node

current node

visited node




```
u = EXTRACT-MIN(Q)              // add u to the tree
for each vertex v in G.Adj[u]   // update keys of u's
    if v ∈ Q and w(u, v) < v.key
        v.π = u
        v.key = w(u, v)
```

B is the closest node to the growing tree.

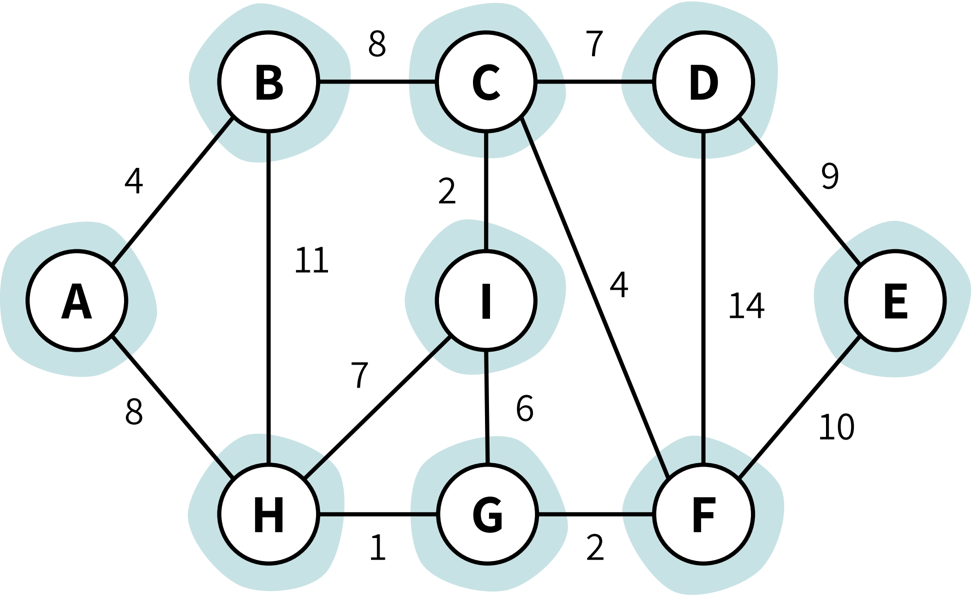
KRUSKAL’S ALGORITHM

Greedily add the cheapest edge!

KRUSKAL'S ALGORITHM: THE IDEA

Greedy choice:

Maintain a forest of trees, & greedily add the cheapest edge to combine trees



Every node on its own starts as an individual tree in this forest

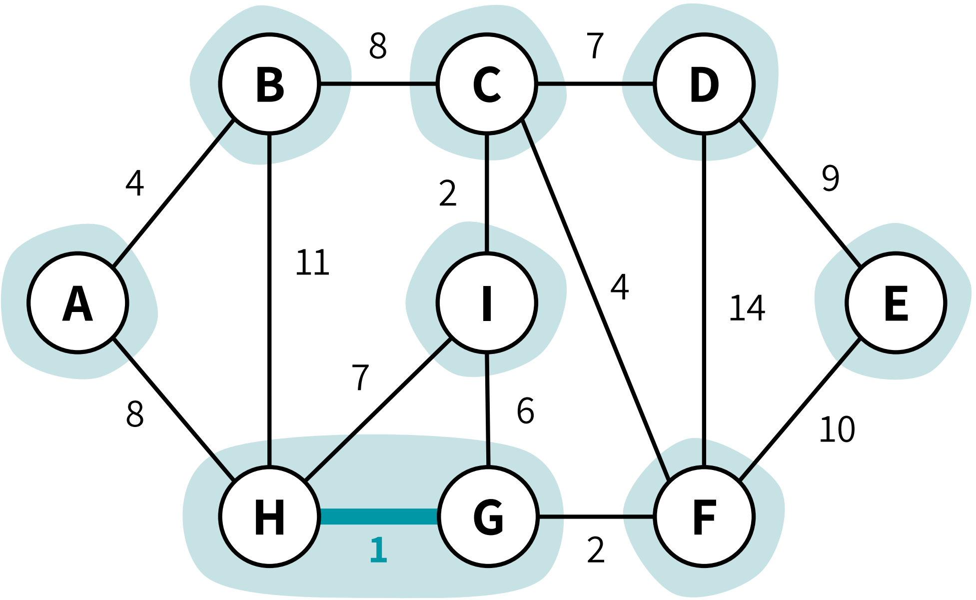
KRUSKAL'S ALGORITHM: THE IDEA

Greedy choice:

Maintain a forest of trees, & greedily add the cheapest edge to combine trees

Choose the cheapest edge that would combine two trees

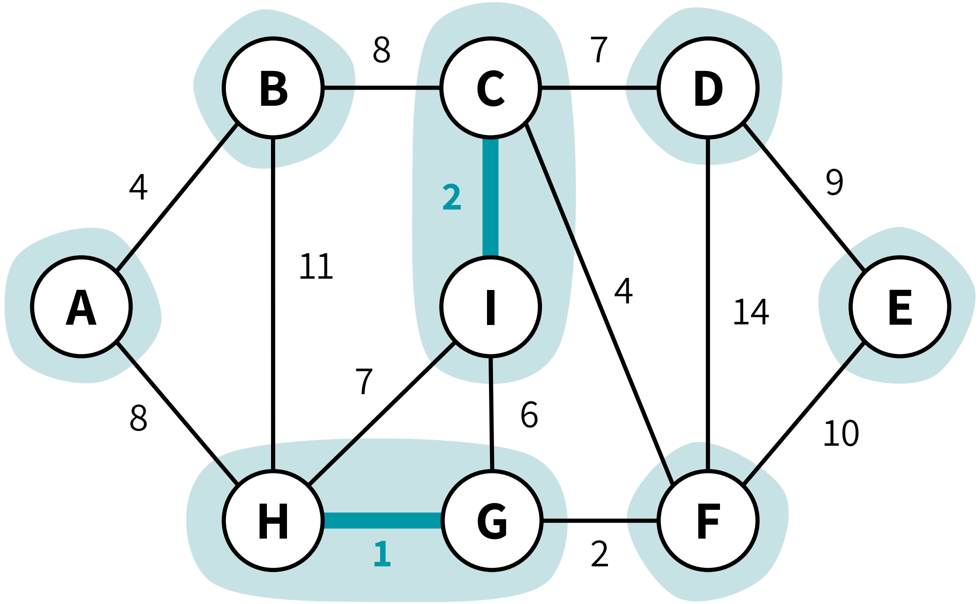
(i.e. that won't cause a cycle)



KRUSKAL'S ALGORITHM: THE IDEA

Greedy choice:

Maintain a forest of trees, & greedily add the cheapest edge to combine trees



Choose the cheapest edge that would combine two trees

(i.e. that won't cause a cycle)

If there's a tie, choose one of the edges

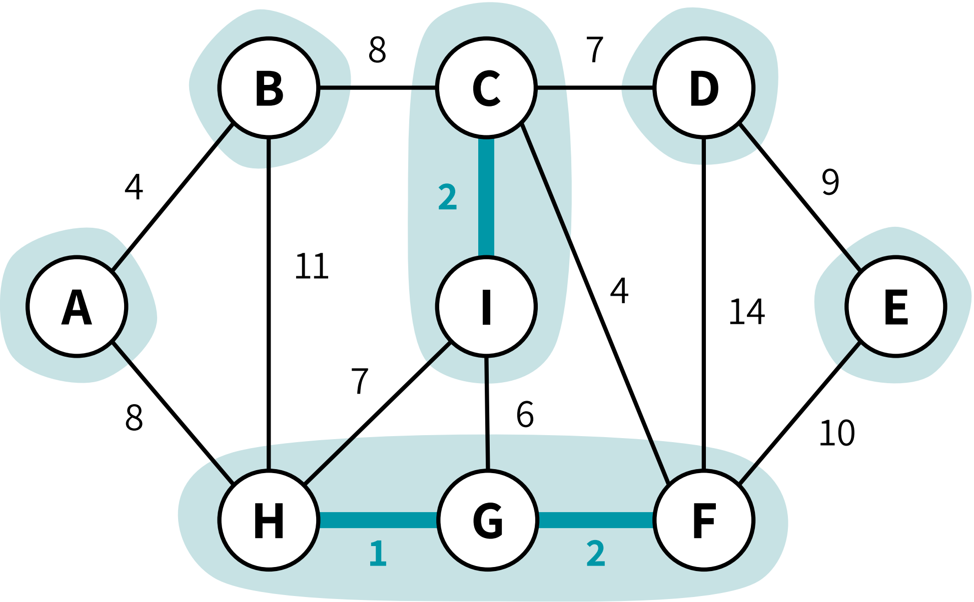
KRUSKAL'S ALGORITHM: THE IDEA

Greedy choice:

Maintain a forest of trees, & greedily add the cheapest edge to combine trees

Choose the cheapest edge that would combine two trees

(i.e. that won't cause a cycle)



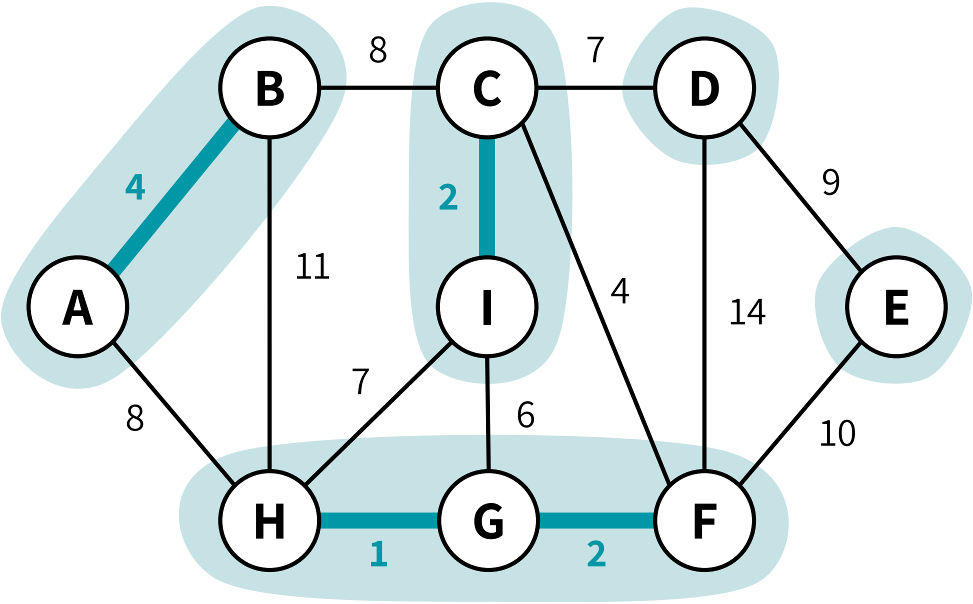
KRUSKAL'S ALGORITHM: THE IDEA

Greedy choice:

Maintain a forest of trees, & greedily add the cheapest edge to combine trees

Choose the cheapest edge that would combine two trees

(i.e. that won't cause a cycle)



If there's a tie, choose one of the edges

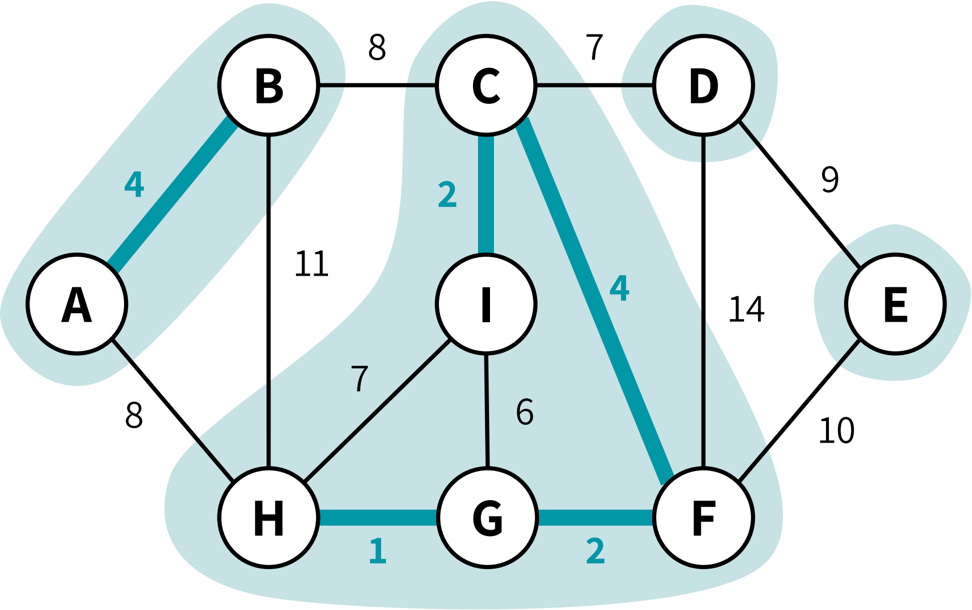
KRUSKAL'S ALGORITHM: THE IDEA

Greedy choice:

Maintain a forest of trees, & greedily add the cheapest edge to combine trees

Choose the cheapest edge that would combine two trees

(i.e. that won't cause a cycle)



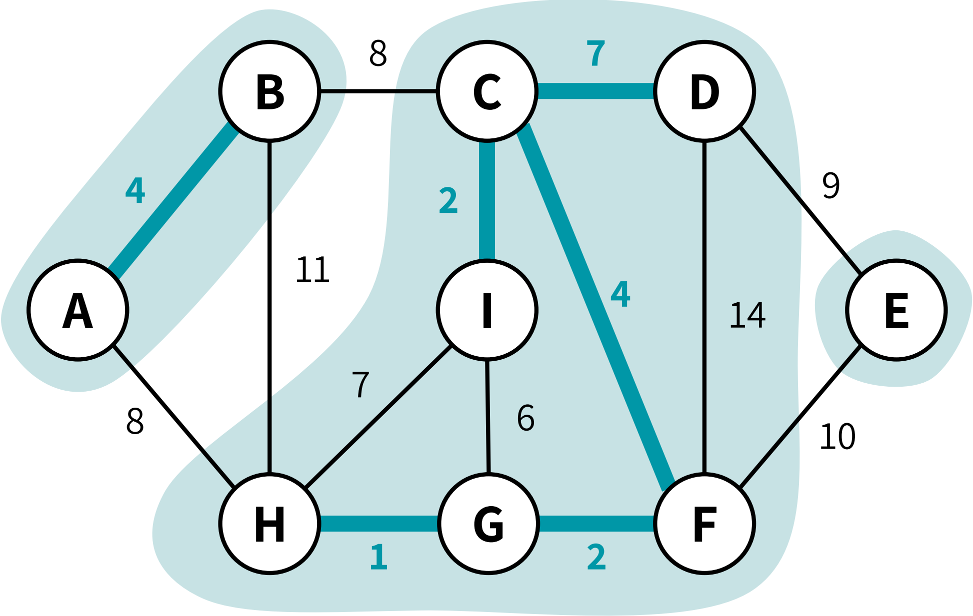
KRUSKAL'S ALGORITHM: THE IDEA

Greedy choice:

Maintain a forest of trees, & greedily add the cheapest edge to combine trees

Choose the cheapest edge that would combine two trees

(i.e. that won't cause a cycle)



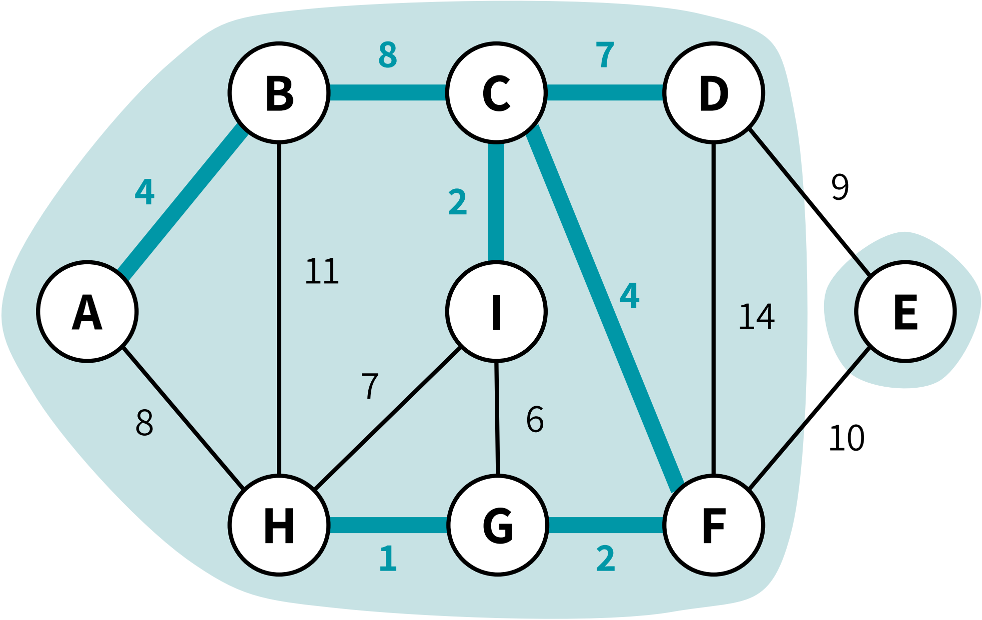
KRUSKAL'S ALGORITHM: THE IDEA

Greedy choice:

Maintain a forest of trees, & greedily add the cheapest edge to combine trees

Choose the cheapest edge that would combine two trees

(i.e. that won't cause a cycle)



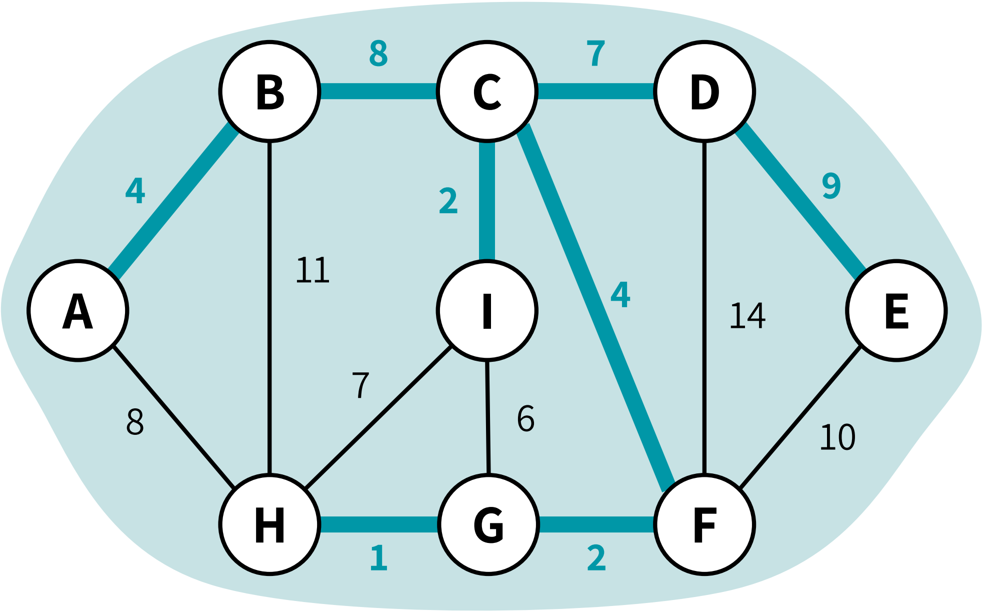
KRUSKAL'S ALGORITHM: THE IDEA

Greedy choice:

Maintain a forest of trees, & greedily add the cheapest edge to combine trees

Choose the cheapest edge that would combine two trees

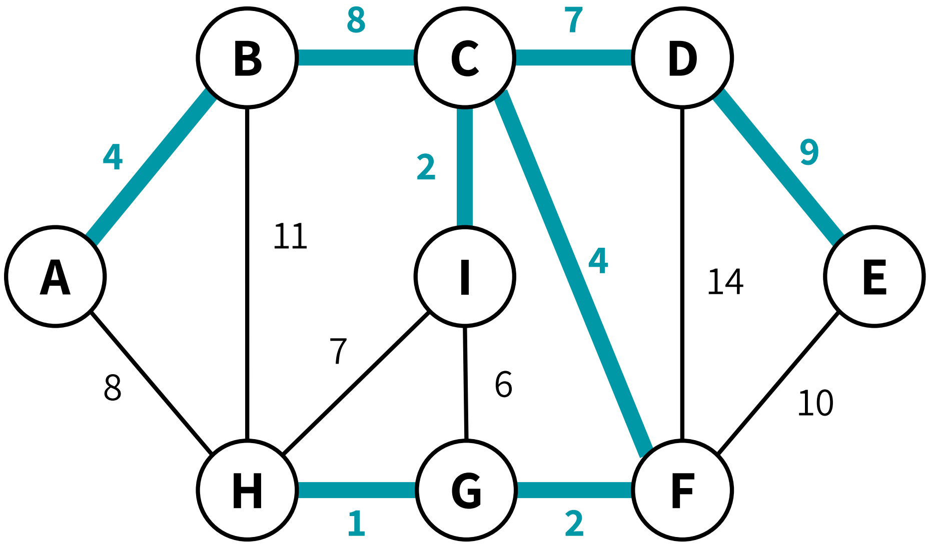
(i.e. that won't cause a cycle)



KRUSKAL'S ALGORITHM: THE IDEA

Greedy choice:

Maintain a forest of trees, & greedily add the cheapest edge to combine trees



We're done!
This is the MST.

KRUSKAL'S ALGORITHM: PSEUDOCODE

```
KRUSKAL-NOT-VERY-DETAILED(G = (V,E)):
   E-SORTED = E sorted by weight in non-decreasing order
   MST = {}
   for v in V:
      put v in its own tree
   for (u,v) in E-SORTED:
      if u's tree and v's tree are not the same:
         MST.add((u,v))
         merge u's tree with v's tree
   return MST
```

To implement these lines, we'll use a ***Union-Find data structure,***
which supports 3 operations: **MAKE-SET(x)**, **FIND(x)**, and **UNION(x,y)**

CLRS textbook version PSEUDOCODE For KRUSKAL'S ALGORITHM

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

since $E \leq V^2$, we have $\log E = O(\log V)$
$O(E \log E) = O(E \log V)$,

Runtime (Time to sort line 4): O(E log E) (merge sort)

(Make Set |V|, for loop 5-8 : O (E)
Total Algo Runtime = O (E log E)