CS2040S
Data Structures and Algorithms

All about minimum spanning trees…

Riddle of the Week: The Travelling SalesPeople

Three travelers show up at a hotel where a room costs $300. They each pay $100 and go to their room.

The manager realizes there is a special sale and the room only costs $250. He gives his assistant $50 to return to the travelers. The assistant only has tens for change, and so gives each traveler $10 in change, keeping $20 for himself.

So each traveler paid $90, and the assistant kept $20, leading to a total of 3*90+20 = 290 dollars. What happened to the remaining 10 dollars?

Tech Interview Preparation for Summer (TIPS)

Overview

SoC will be conducting a **10-week** programme this summer to ensure that our freshmen are ***well-prepared*** *to* handle technical (coding) interviews when they apply for **technical internships**.

Team

TIPS is taught and run by senior students with a wide range of internship experiences from companies like Meta, Tiktok, GIC, etc.

Programme Outline

- Resume preparation & reviews
- Technical interview skills & techniques
- Weekly technical interview preparation questions
- Weekly mock interviews
- Career & internship sharings from senior students & alumni from different companies
- Career discussion panels with recruiters & tech chiefs from tech companies with a local presence.

Tech Interview Preparation for Summer (TIPS)

Weekly Commitment

- Attend weekly sharing session on **Monday evening 8pm - 9.30pm** (19th May - 21st July)
- Complete at least **6 LeetCode questions**
- Complete one **mock interview** with a classmate from Week 3 onwards

* in order to make sure everyone in the program is responsible and committed to the peer activity (e.g. mock interview), failure in completing the weekly commitment may result students being removed from the programme

FAQs

Q: Can I join this if I have Orbital / CVWO / Internship / anything else?

A: Yes! As long as you are able to complete the weekly commitment.

For any enquiries, please email

soc-tips@googlegroups.com.

500+ of your SoC seniors have benefitted from TIPS last summer.
What are you waiting for?

Sign up now!

https://coursemology.org/courses/2918/surveys/2547

CS2040S
Data Structures and Algorithms

All about minimum spanning trees…

Last Week:

UFDS

- Various Heuristics

Minimum Spanning Trees

- Kruskal’s Algorithm

Roadmap

Minimum Spanning Trees

- Background
- Prim's Algorithm
- Kruskal's Algorithm

Variations:

- Constant weight edges
- Bounded integer edge weights
- Directed graphs
- Maximum Spanning Tree

Roadmap

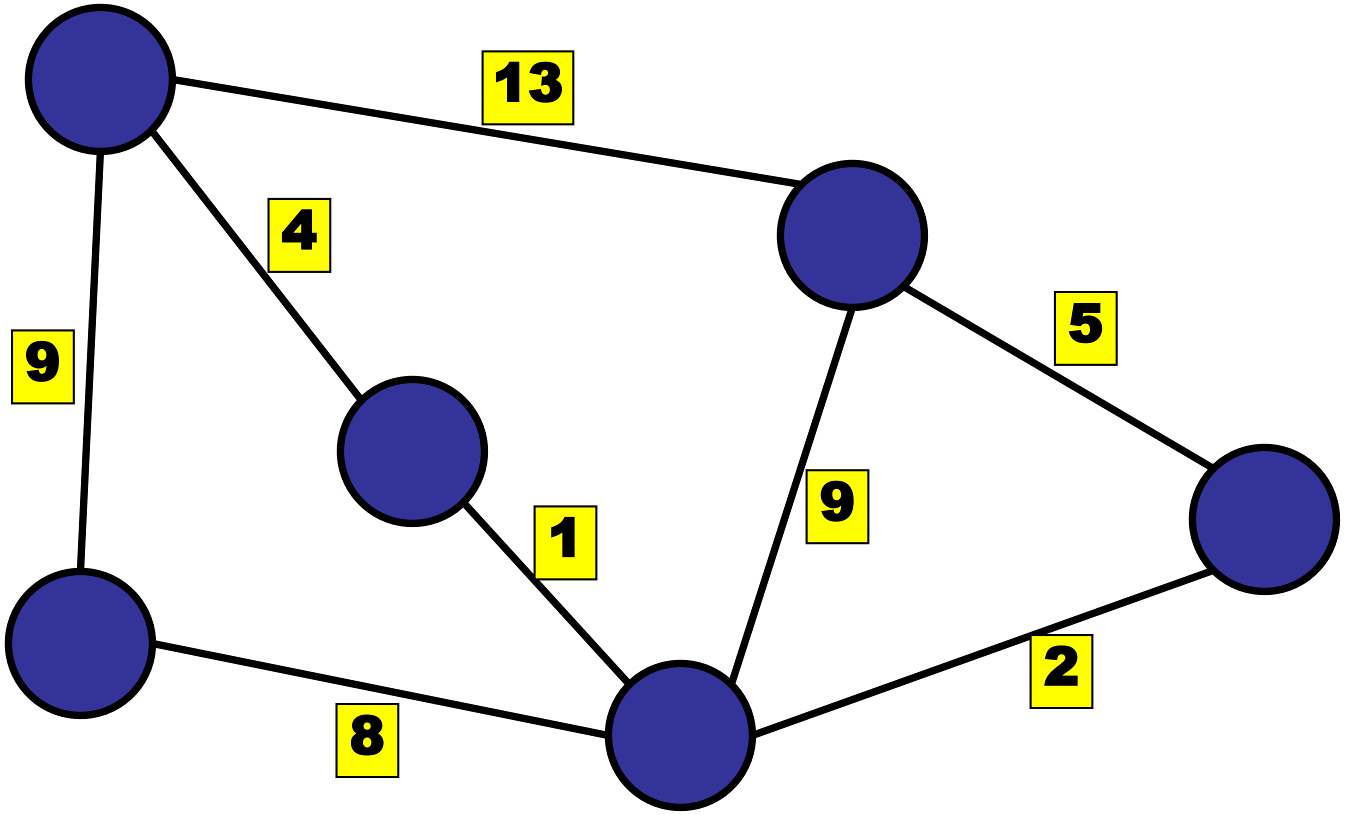
Minimum Spanning Trees

- **The MST Problem**
- Basic Properties of an MST
- Generic MST Algorithm
- Prim’s Algorithm
- Kruskal’s Algorithm
- Variations

Spanning Tree

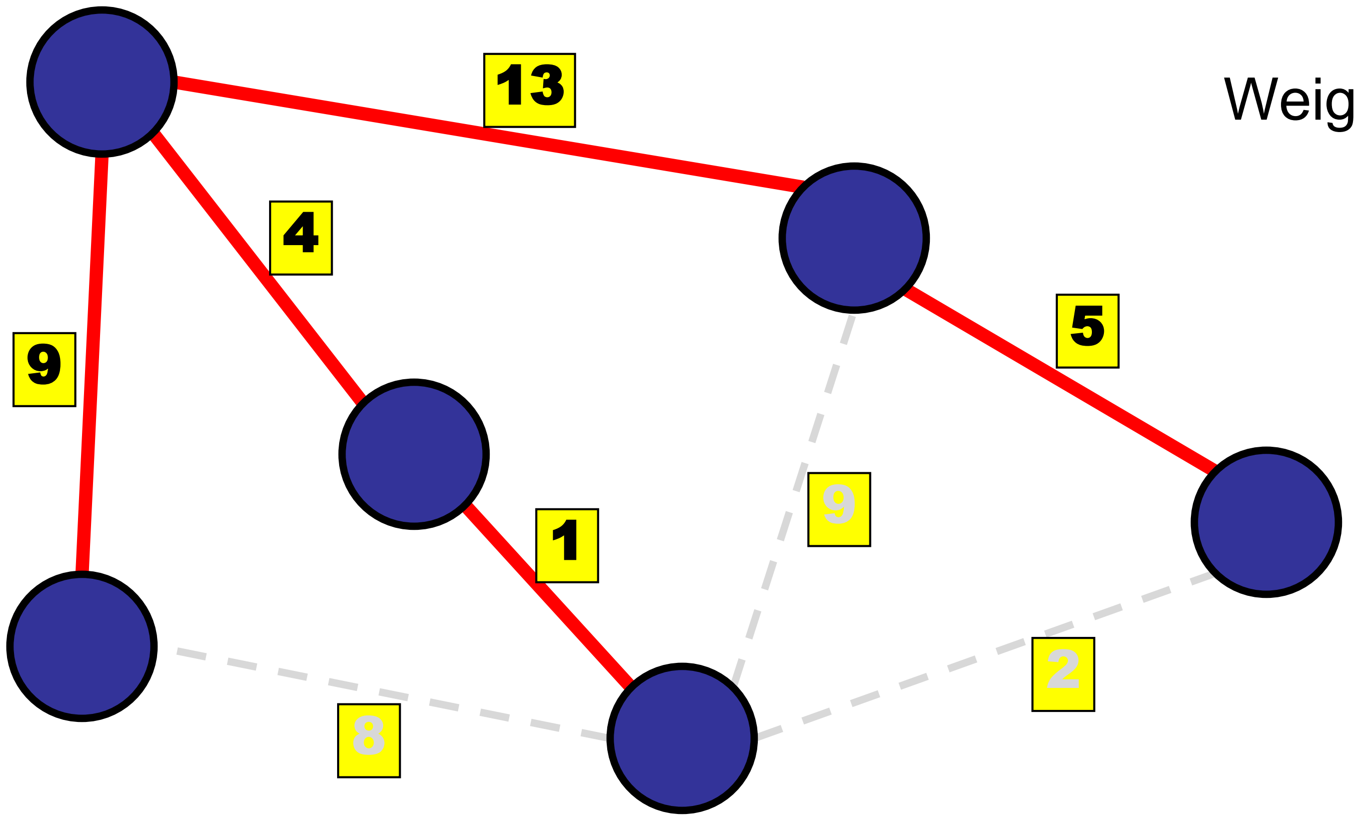
Weighted, undirected graph:

To think about:
Why is this more complicated with directed graphs?



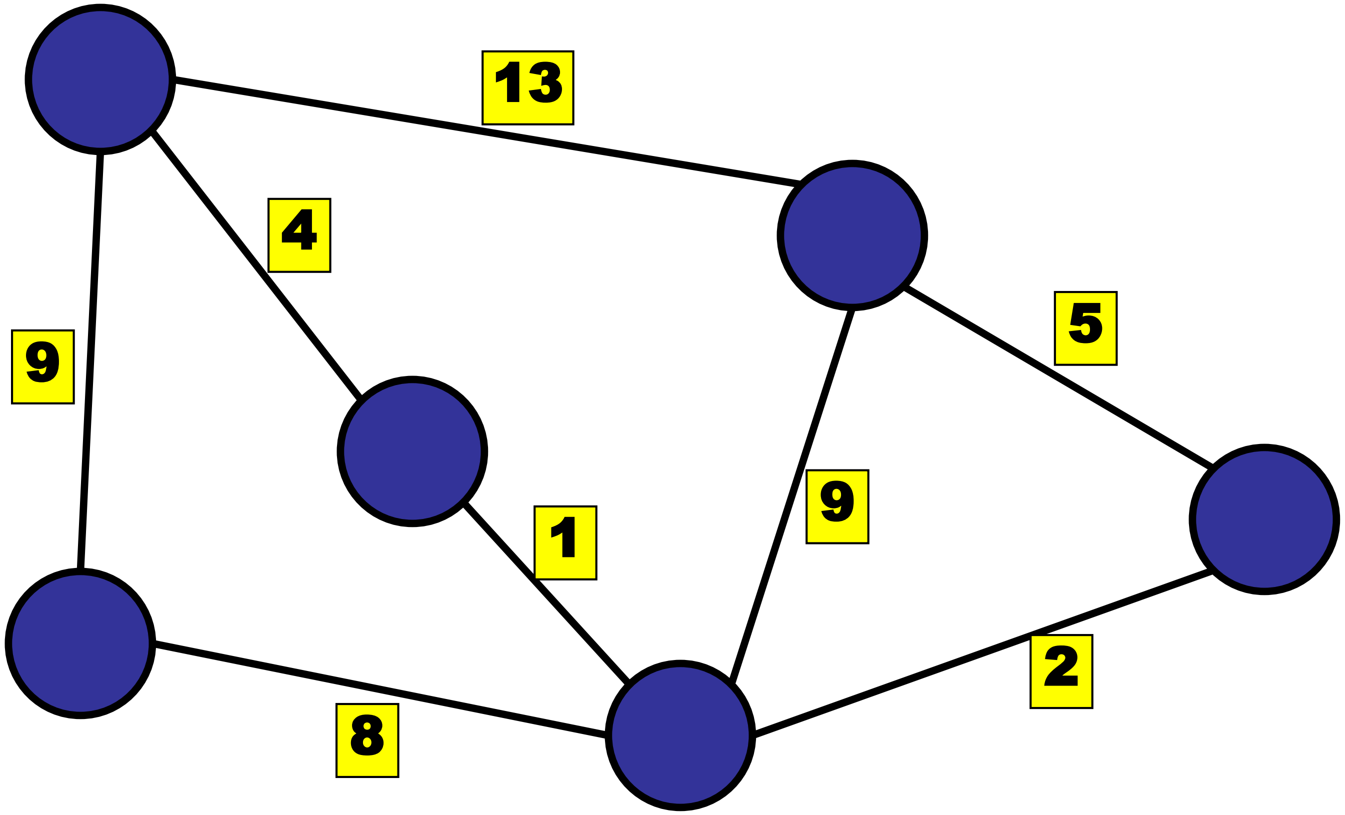
Spanning Tree

Definition: a spanning tree is an acyclic subset of the edges that connects all nodes



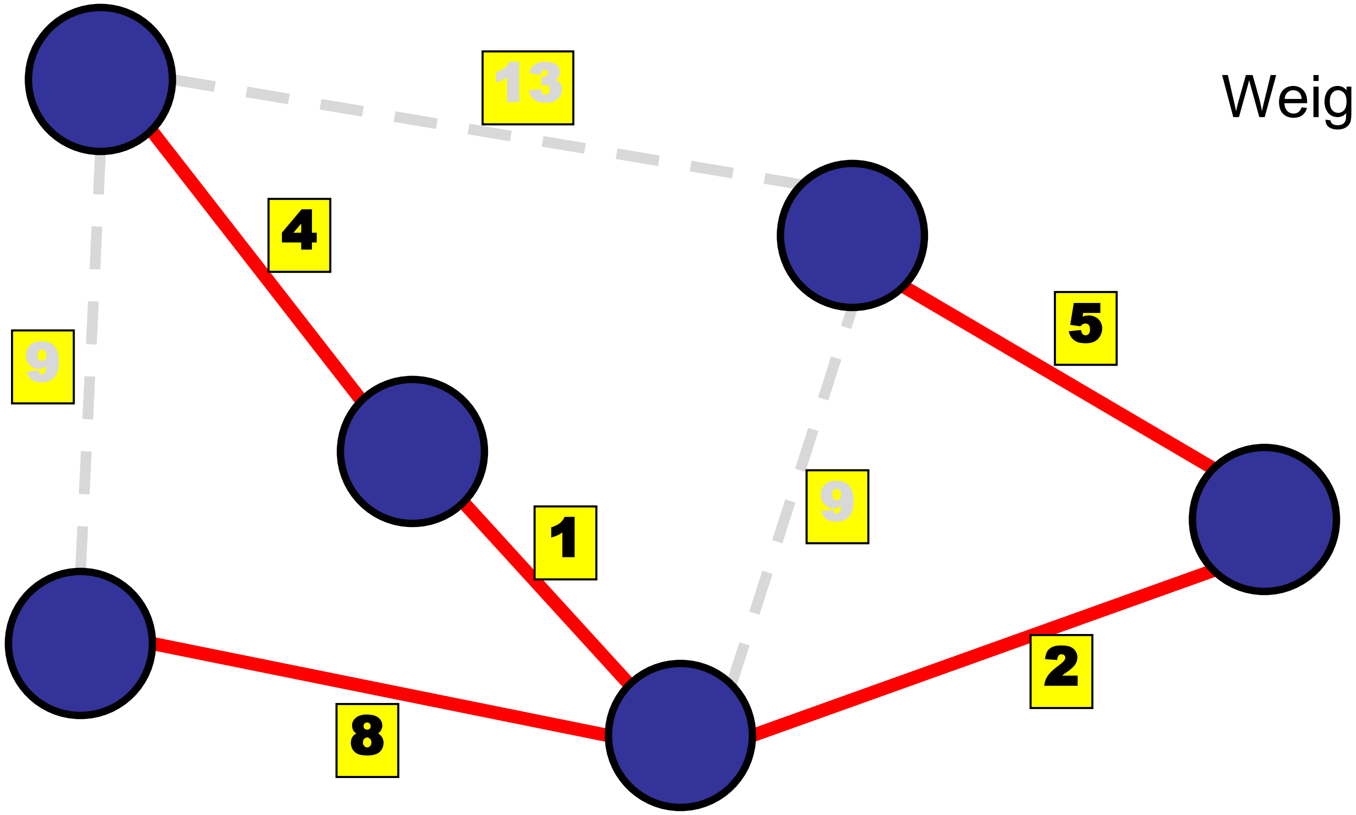
Minimum Spanning Tree

Definition: a spanning tree with minimum weight



Minimum Spanning Tree

Definition: a spanning tree with minimum weight

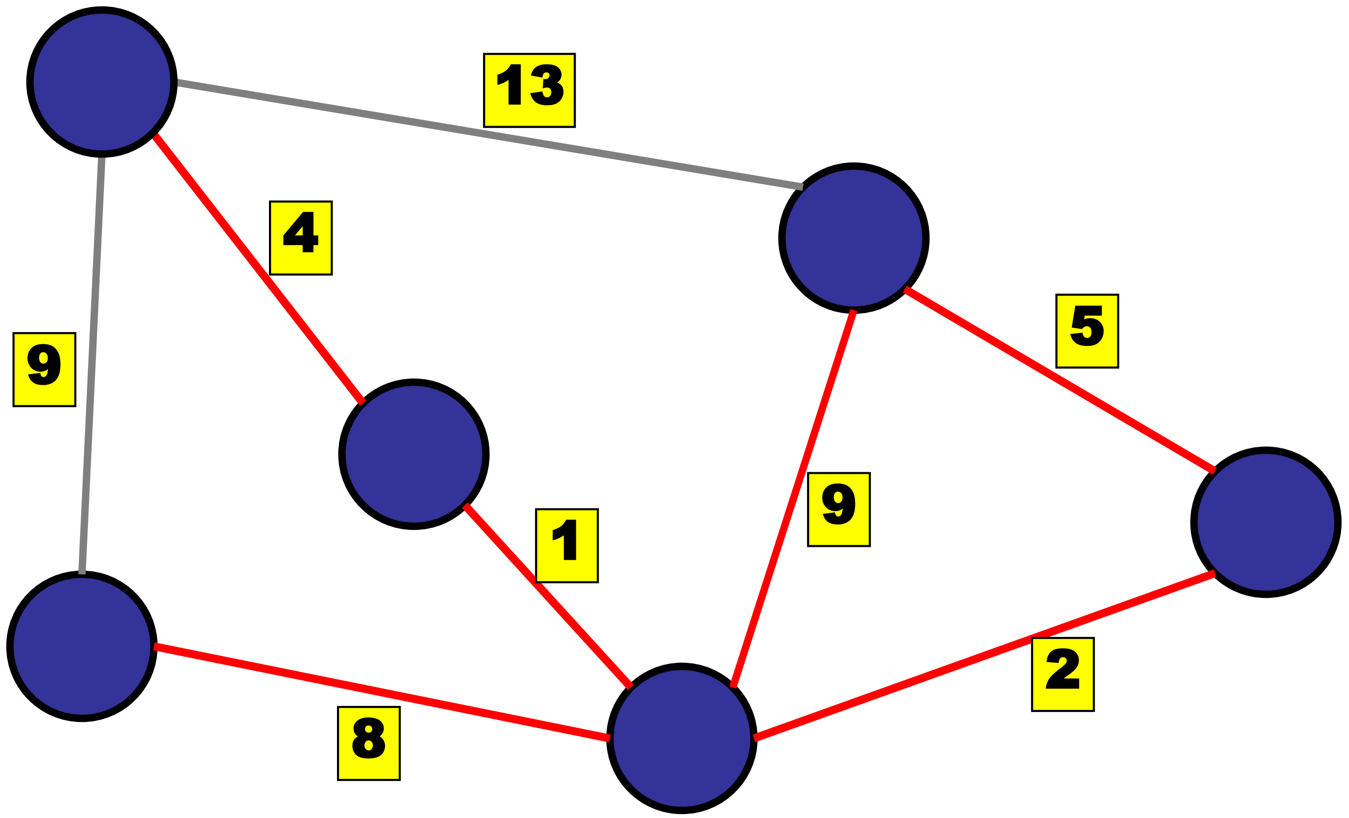


Weight: 20

Minimum Spanning Tree

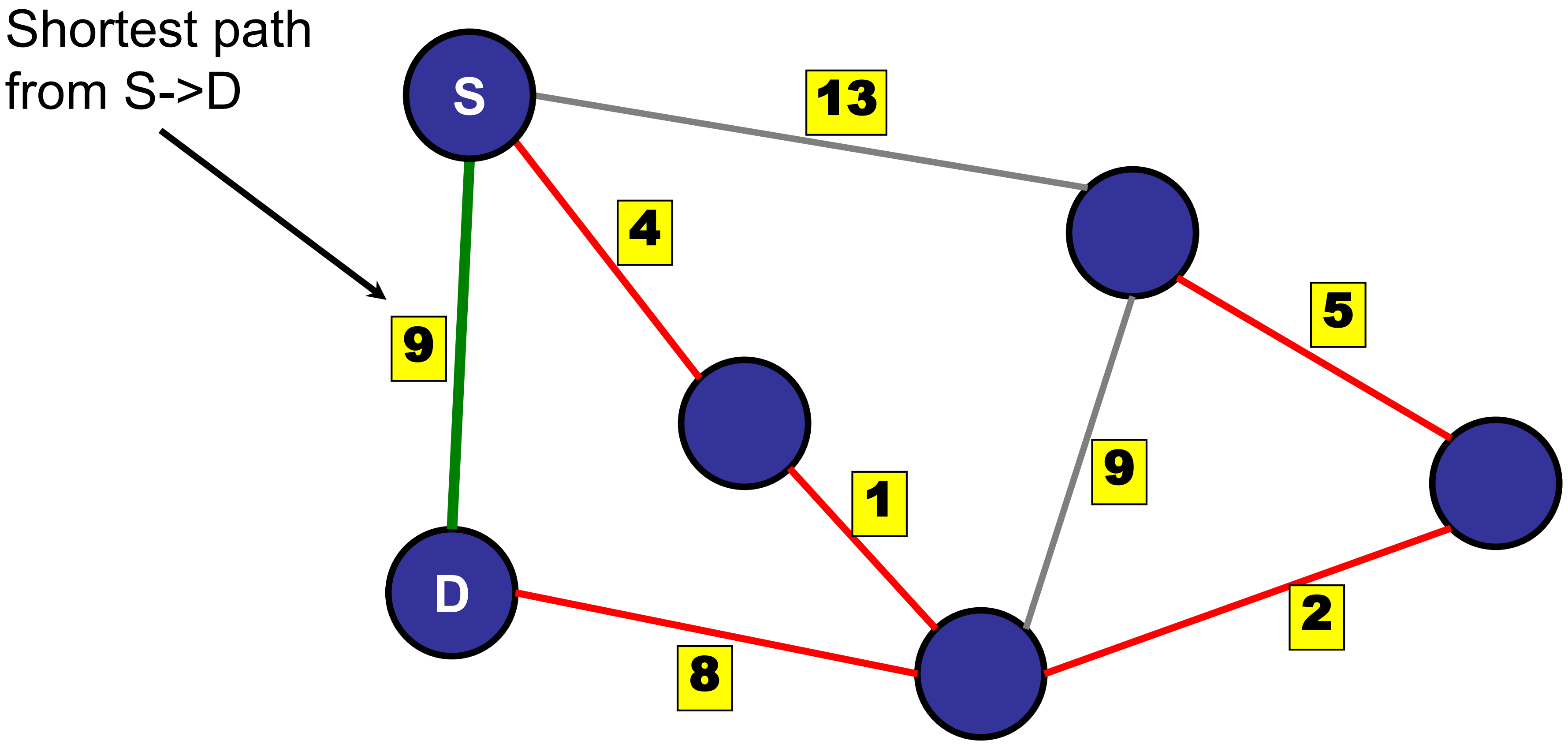
Note: no cycles

Why? If there were cycles, we could remove one edge and reduce the weight!



Minimum Spanning Tree

Recall: Not the same a shortest paths:



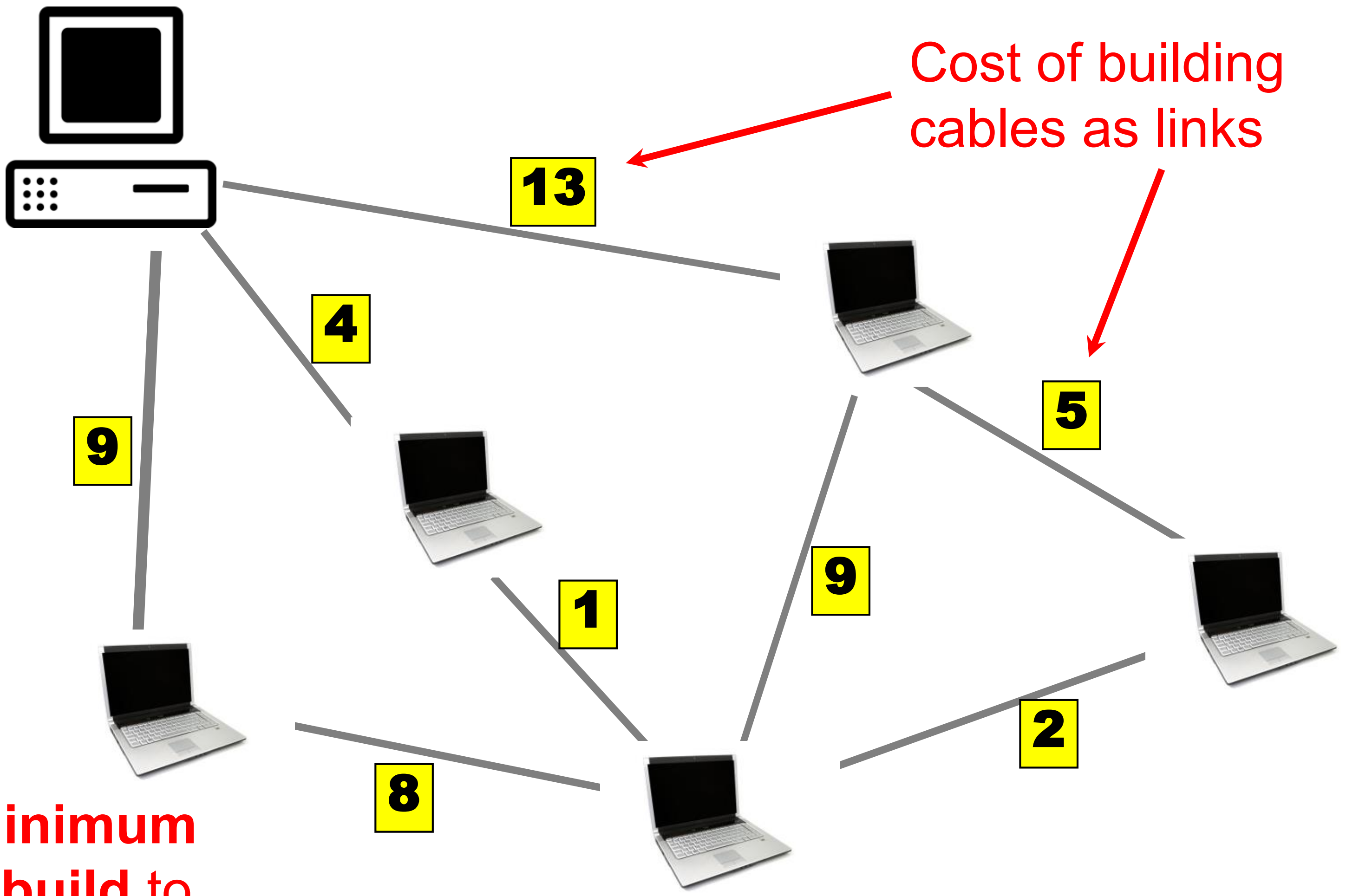
Applications of MST

Network design problems:

- Telephone networks
- Electrical networks
- Computer networks
- Ethernet autoconfig
- Road networks
- Bottleneck paths

Data distribution

Connecting people over the Internet:



Cost of building cables as links

What is the **minimum cost links to build** to connect everyone**?**

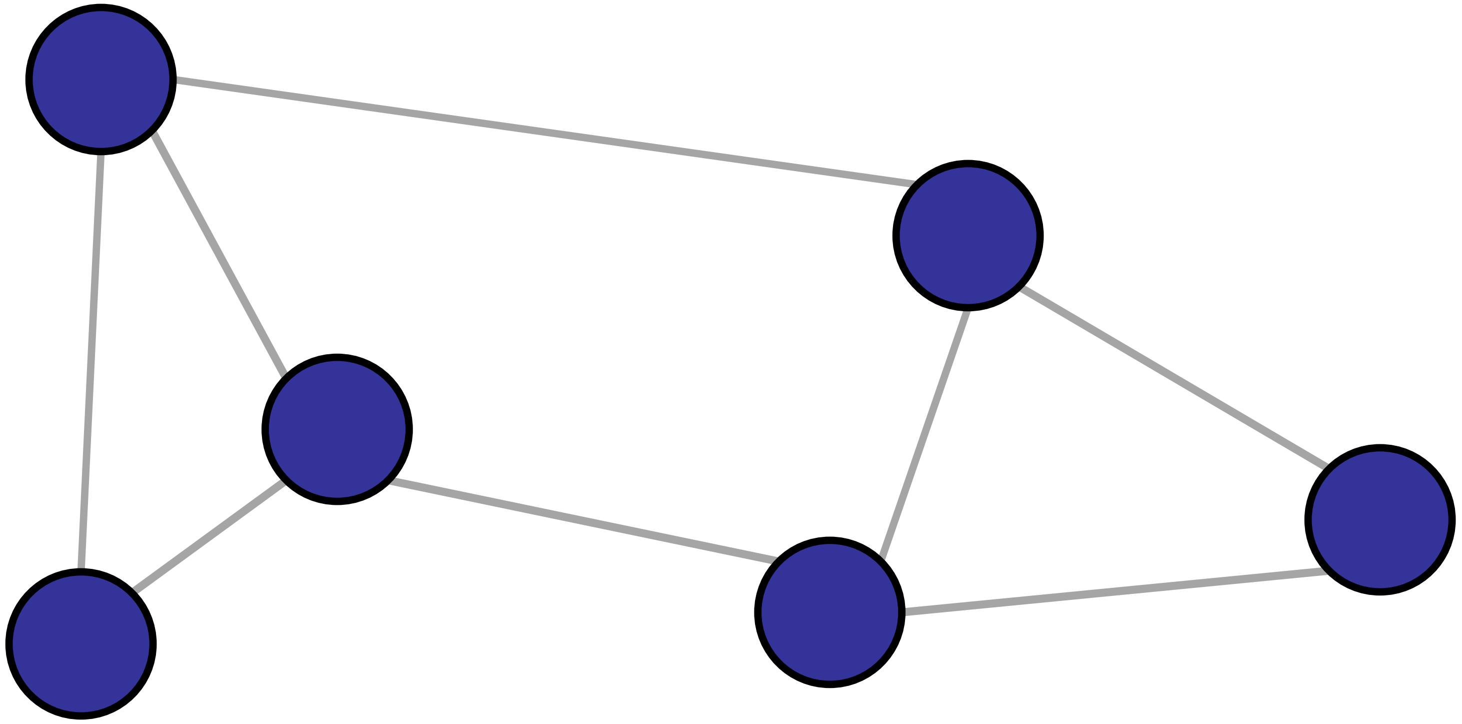
Applications of MST

Less obvious applications:

- Error correcting codes
- Face verification
- Cluster analysis for data science
  - https://en.wikipedia.org/wiki/Hierarchical_clustering
- Image registration
- Approximating solutions for hard-to-solve problems (PSet 8)

Assumption

Assume graph is connected.
(Otherwise we cannot create a spanning tree in the first place)



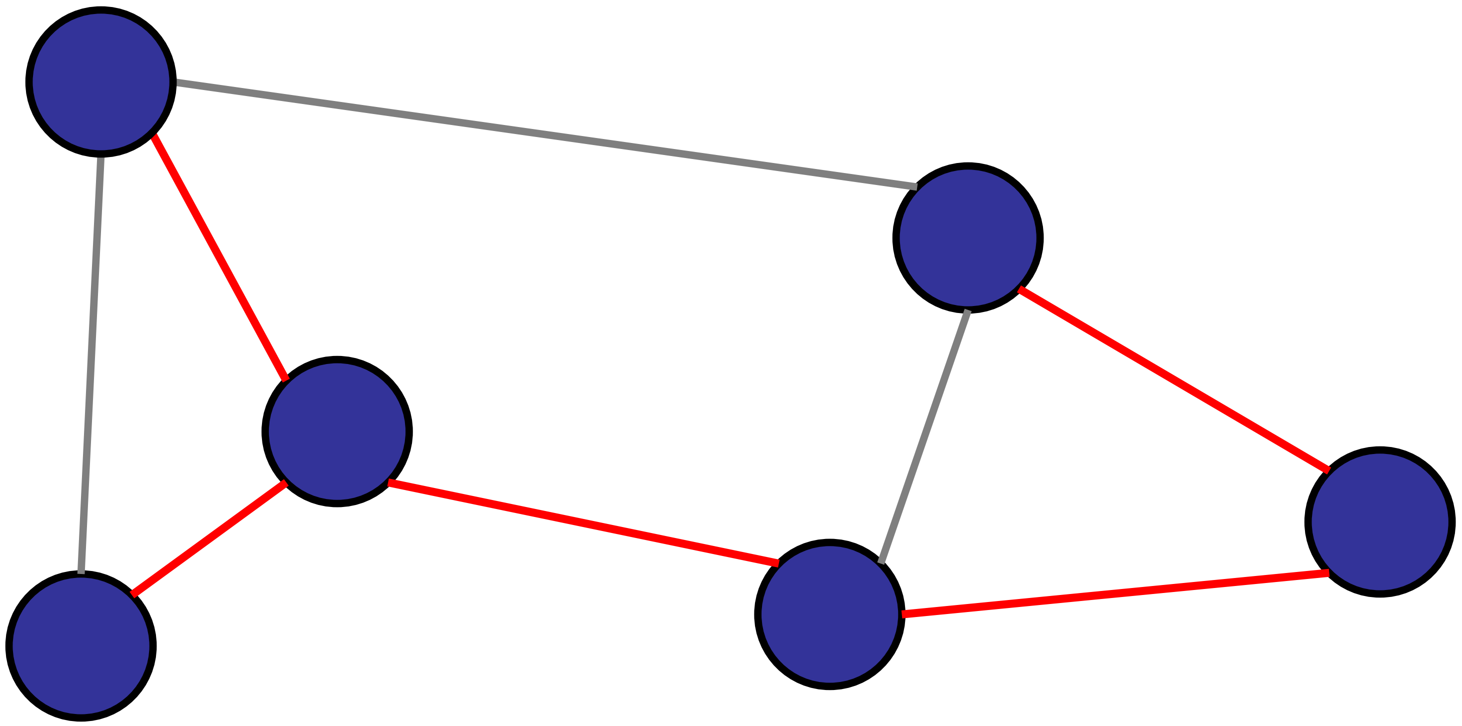
Roadmap

Minimum Spanning Trees

- The MST Problem
- **Basic Properties of an MST**
- Generic MST Algorithm
- Prim’s Algorithm
- Kruskal’s Algorithm
- Boruvka’s Algorithm
- Variations

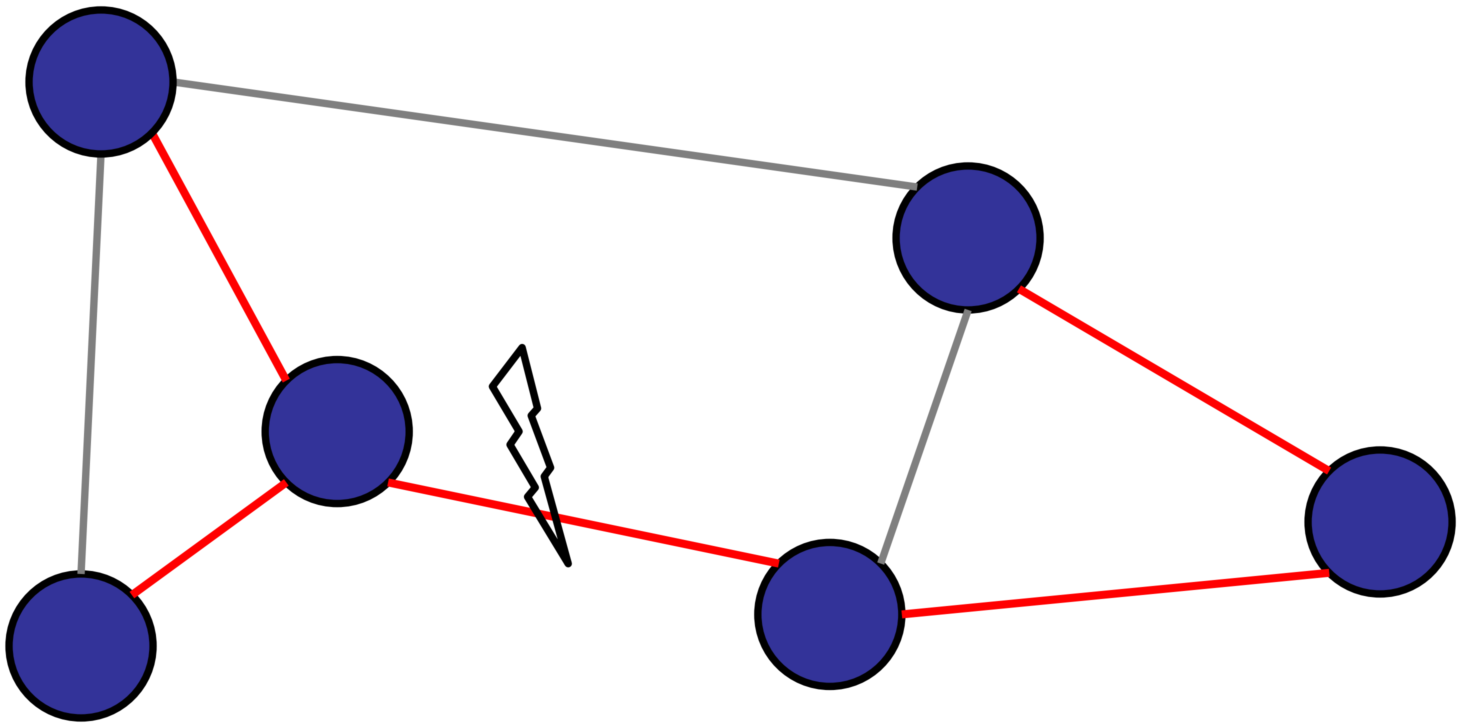
Properties of MST

Property 1: No cycles



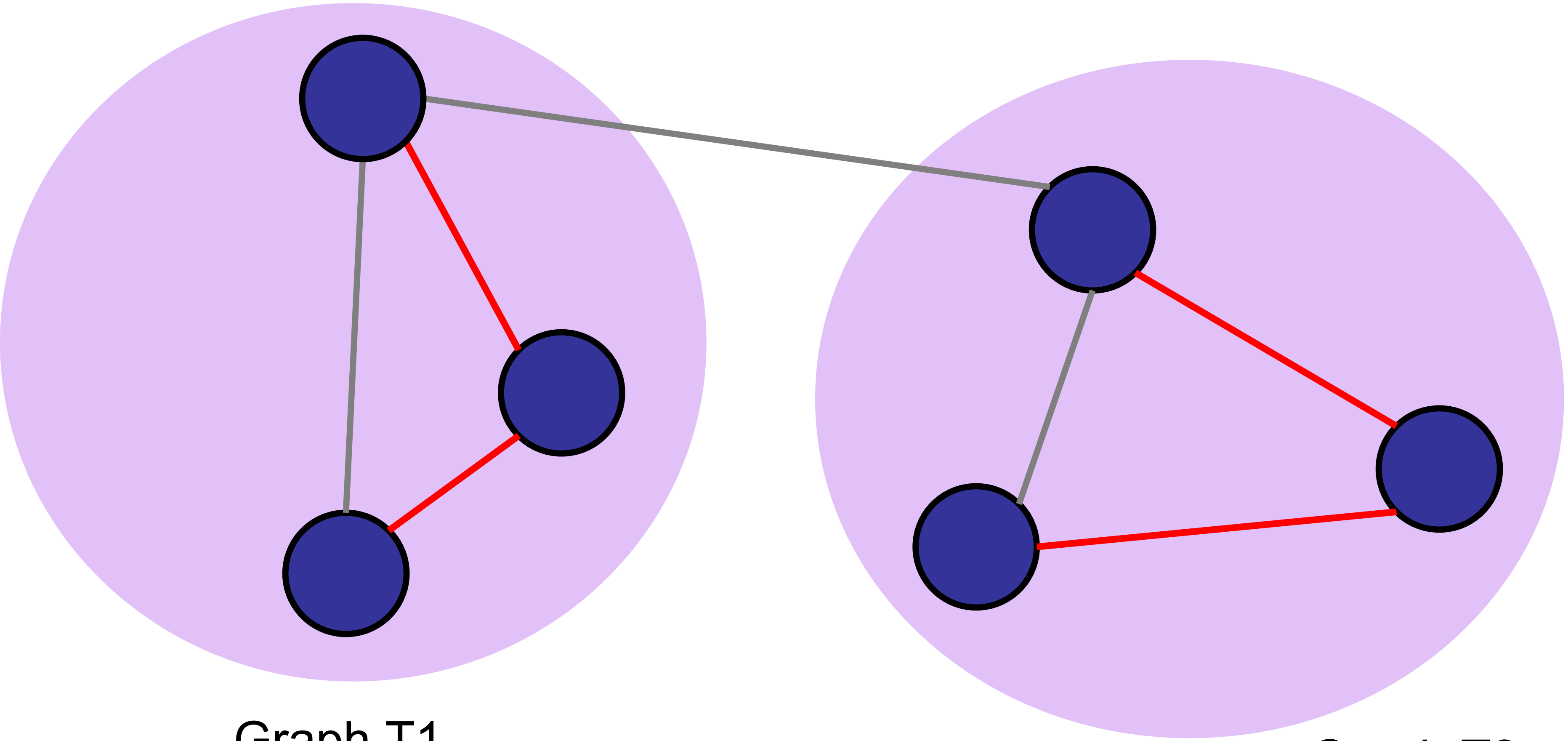
Properties of MST

What happens if you cut an MST?



Properties of MST

What happens if you cut an MST?

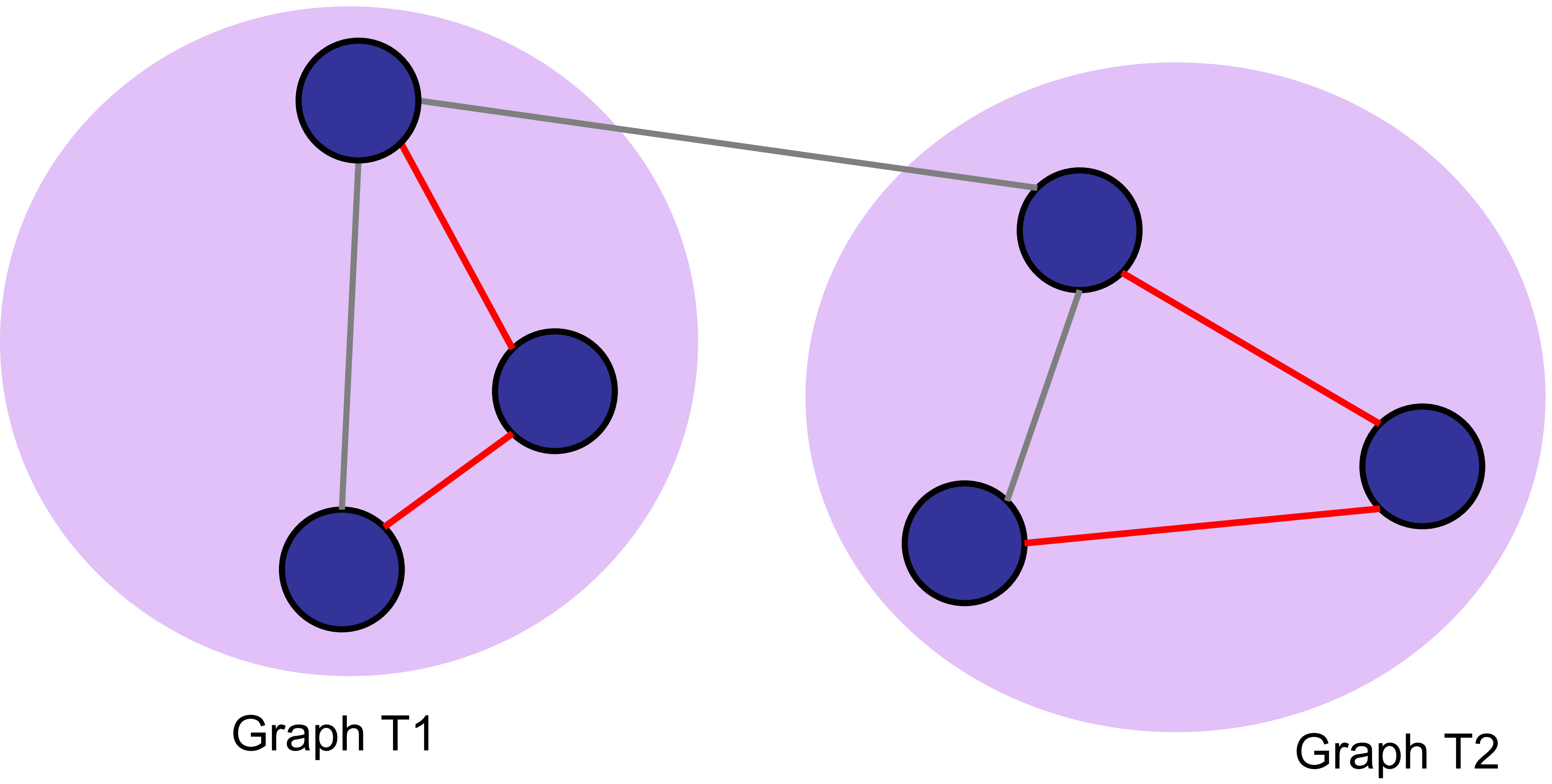


Graph T1

Graph T2

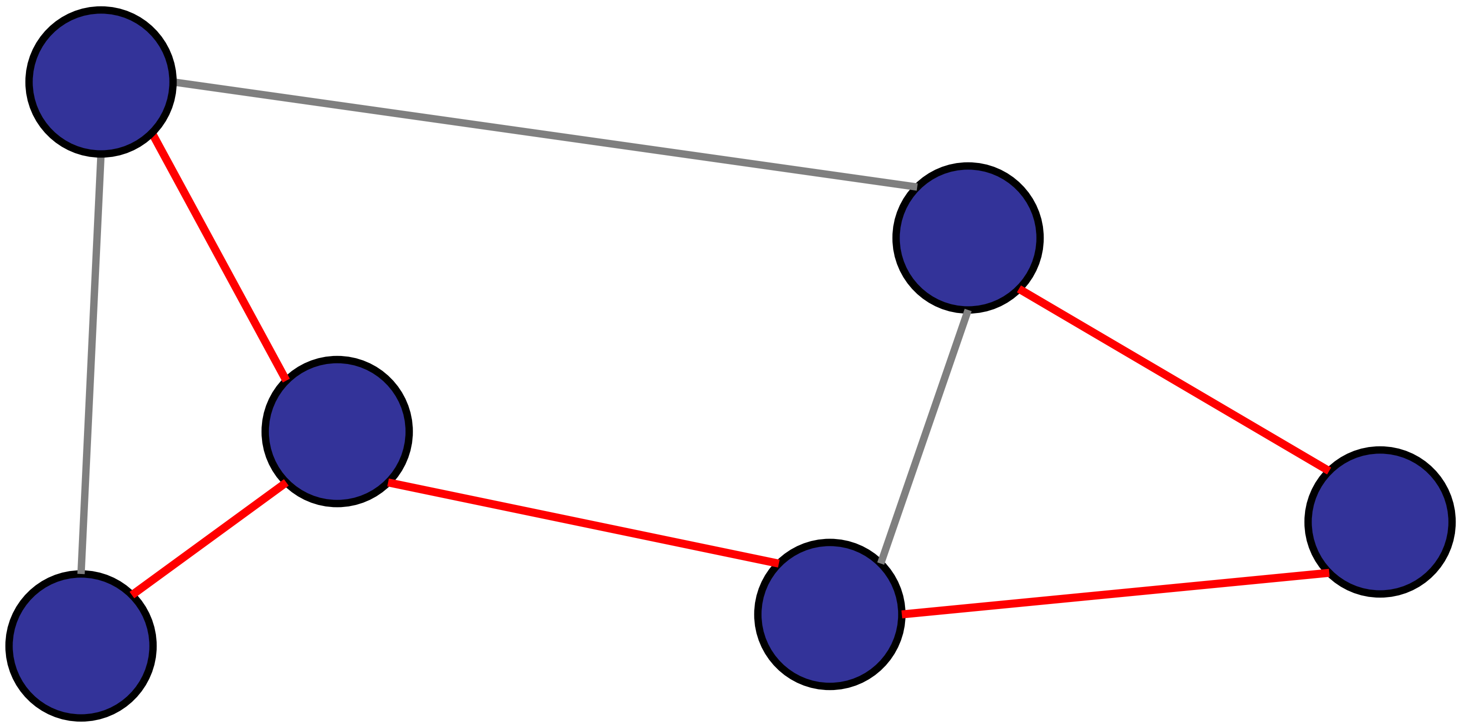
Properties of MST

Theorem: T1 is an MST and T2 is an MST.



Properties of MST

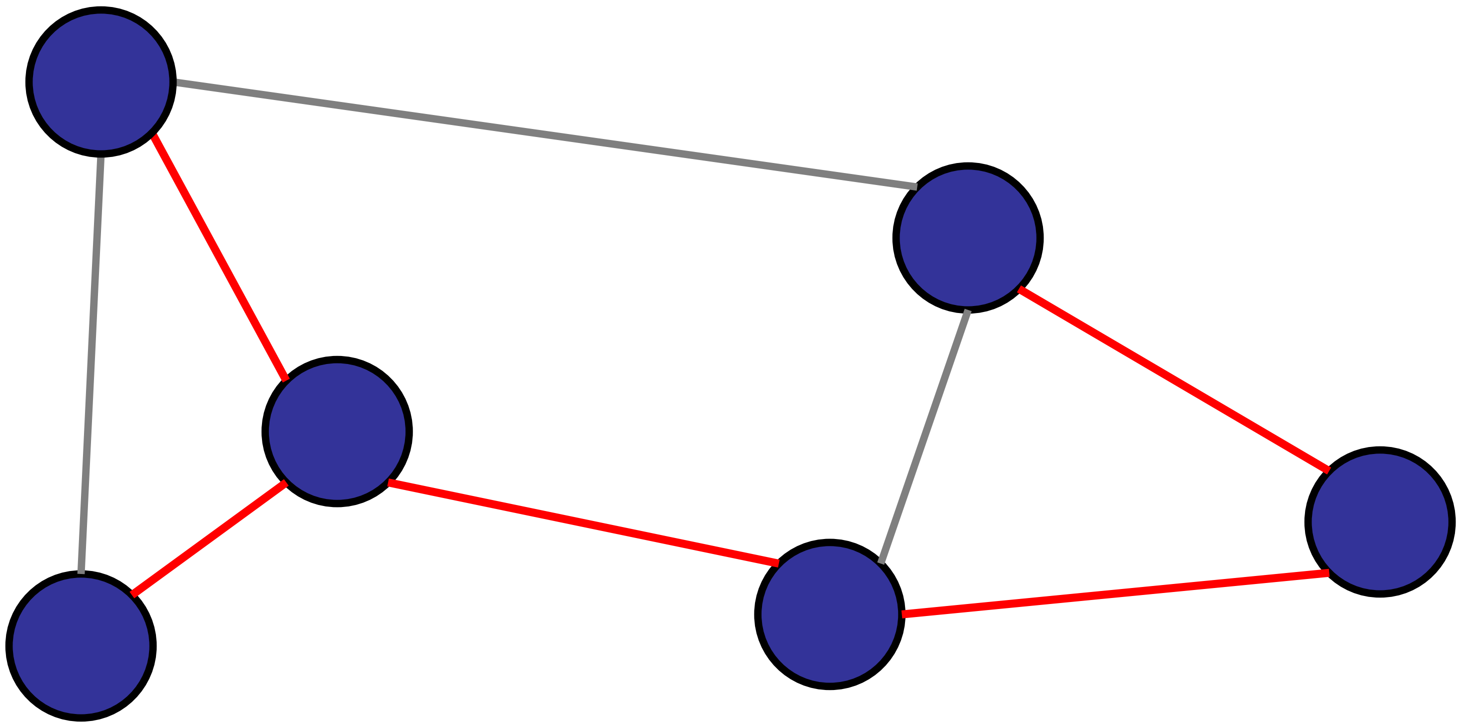
Property 2: If you cut an MST, the two pieces are both MSTs.



Overlapping sub-problems! Dynamic programming? Yes, but better…

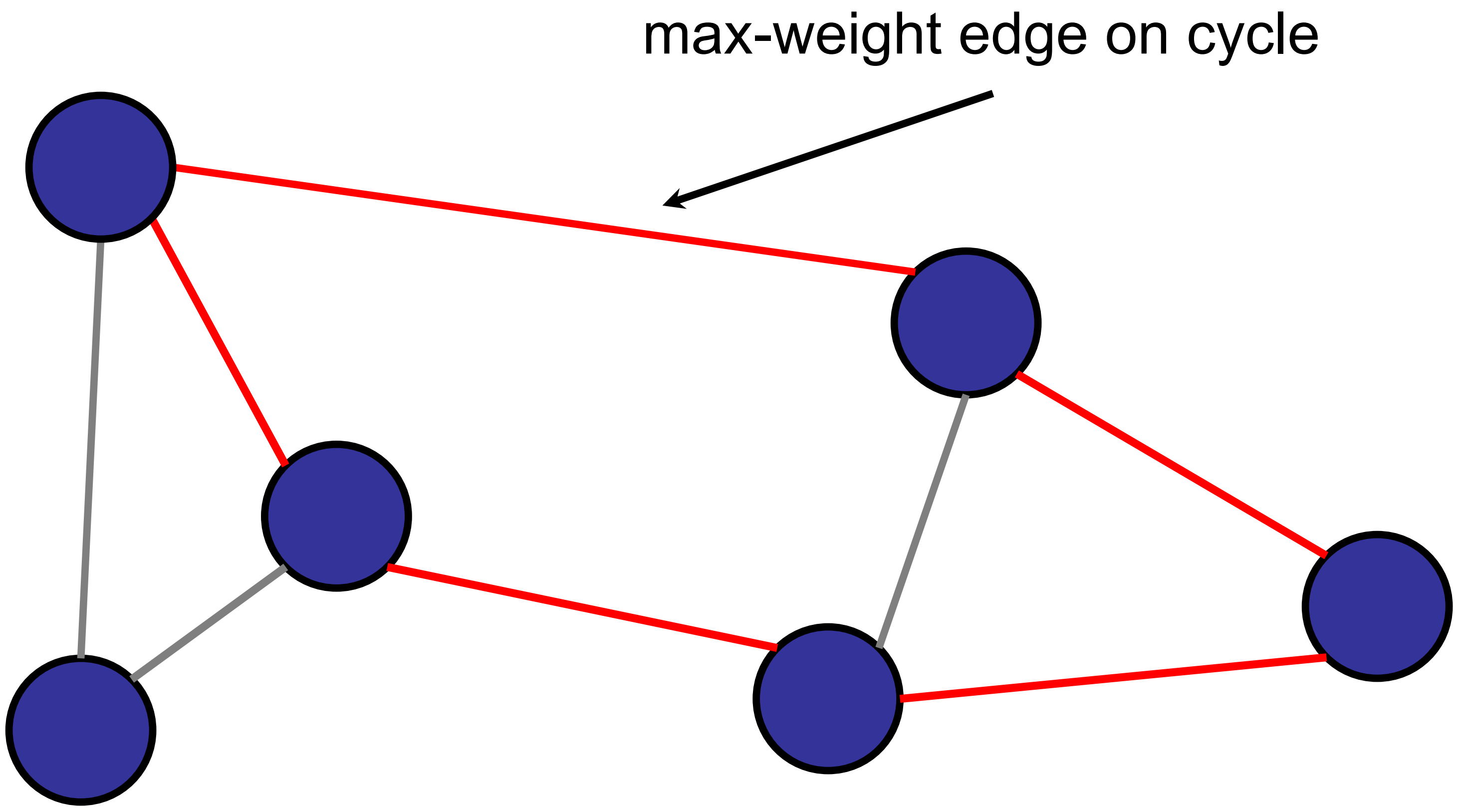
Properties of MST

Property 2: If you cut an MST, the two pieces are both MSTs.



Properties of MST

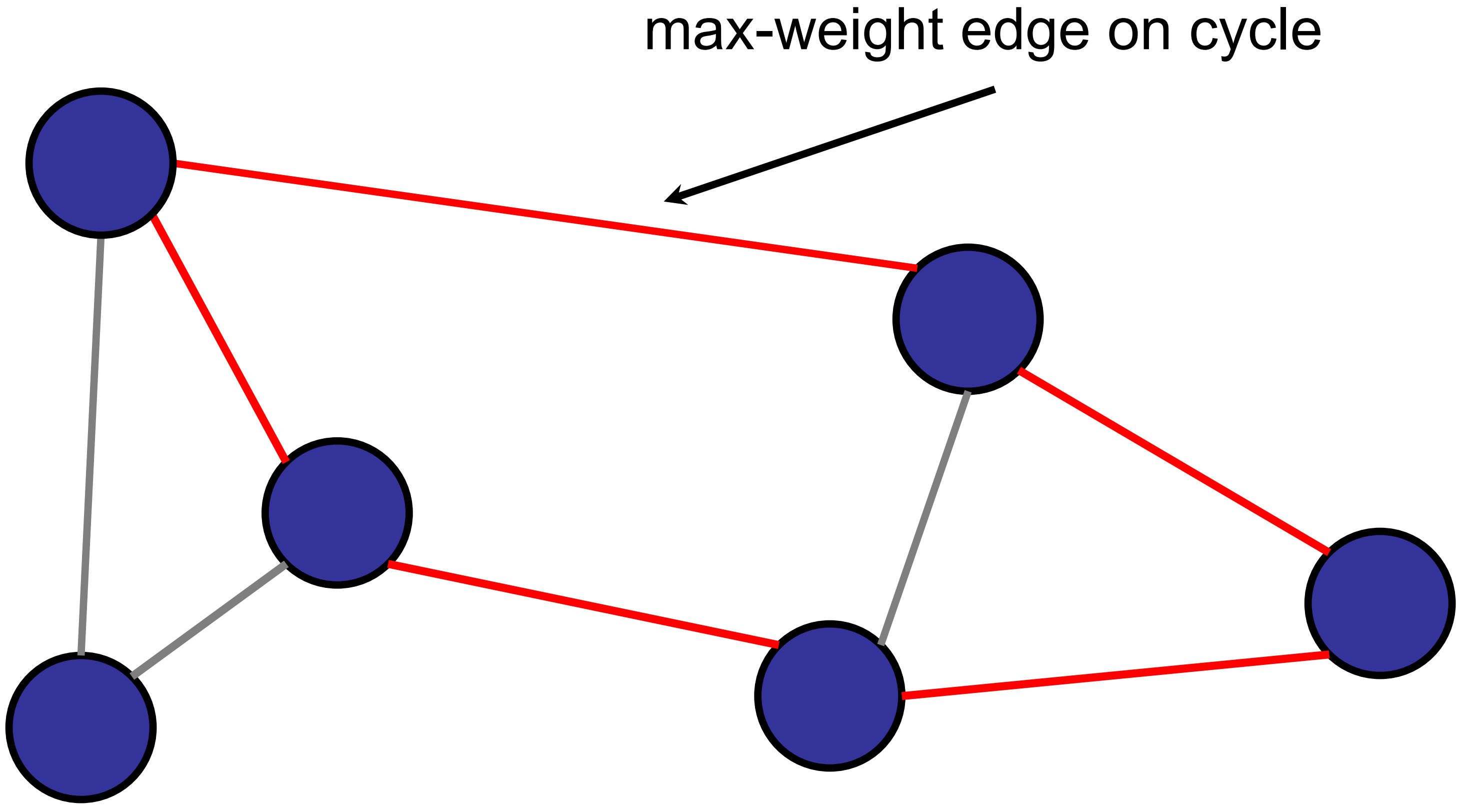
Property 3: Cycle property



Properties of MST

Property 3: Cycle property

For every cycle, the maximum weight edge is ***not*** in the MST. (assuming distinct weights)

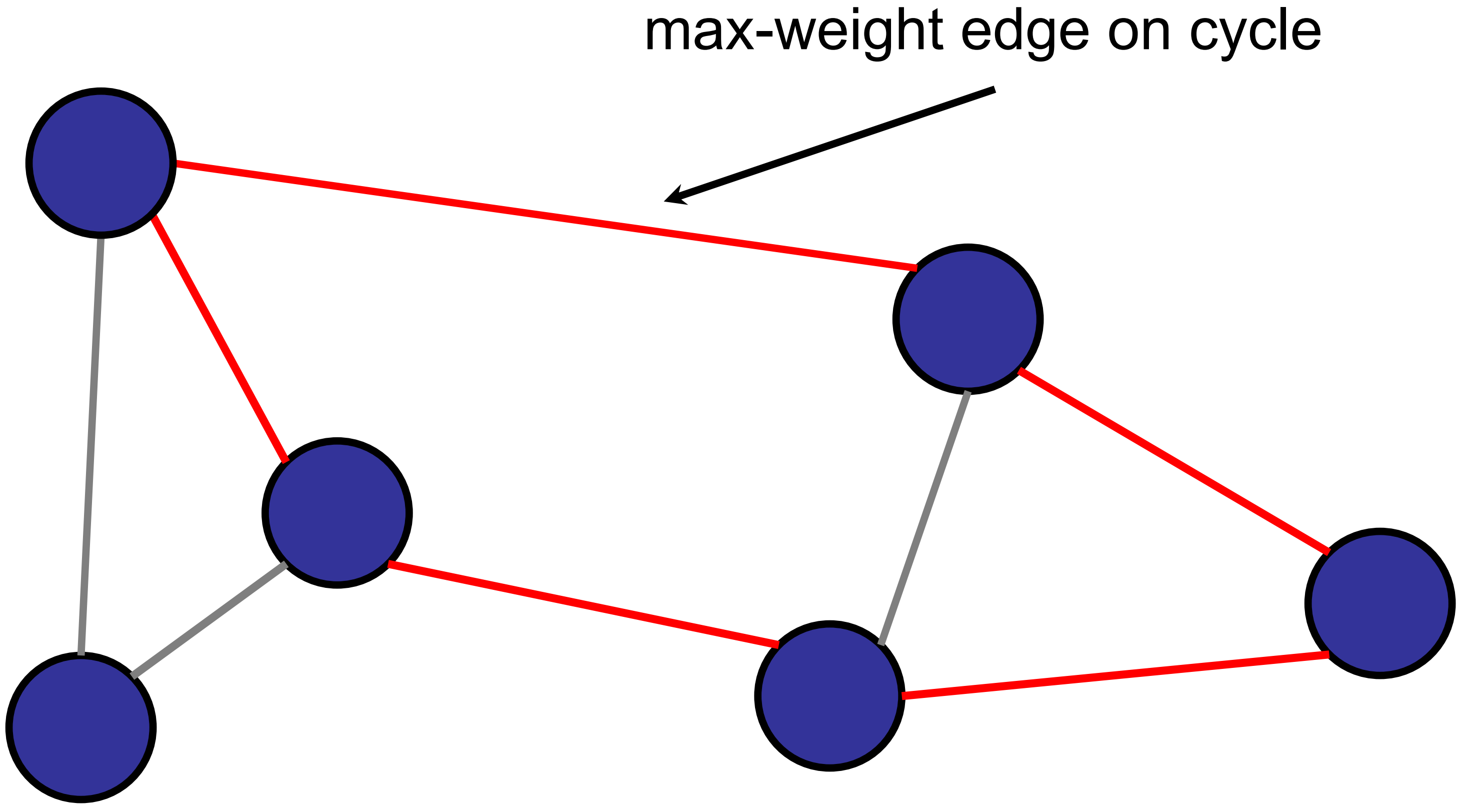


Properties of MST

To think about:
What happens if edge weights are not distinct?

Property 3: Cycle property

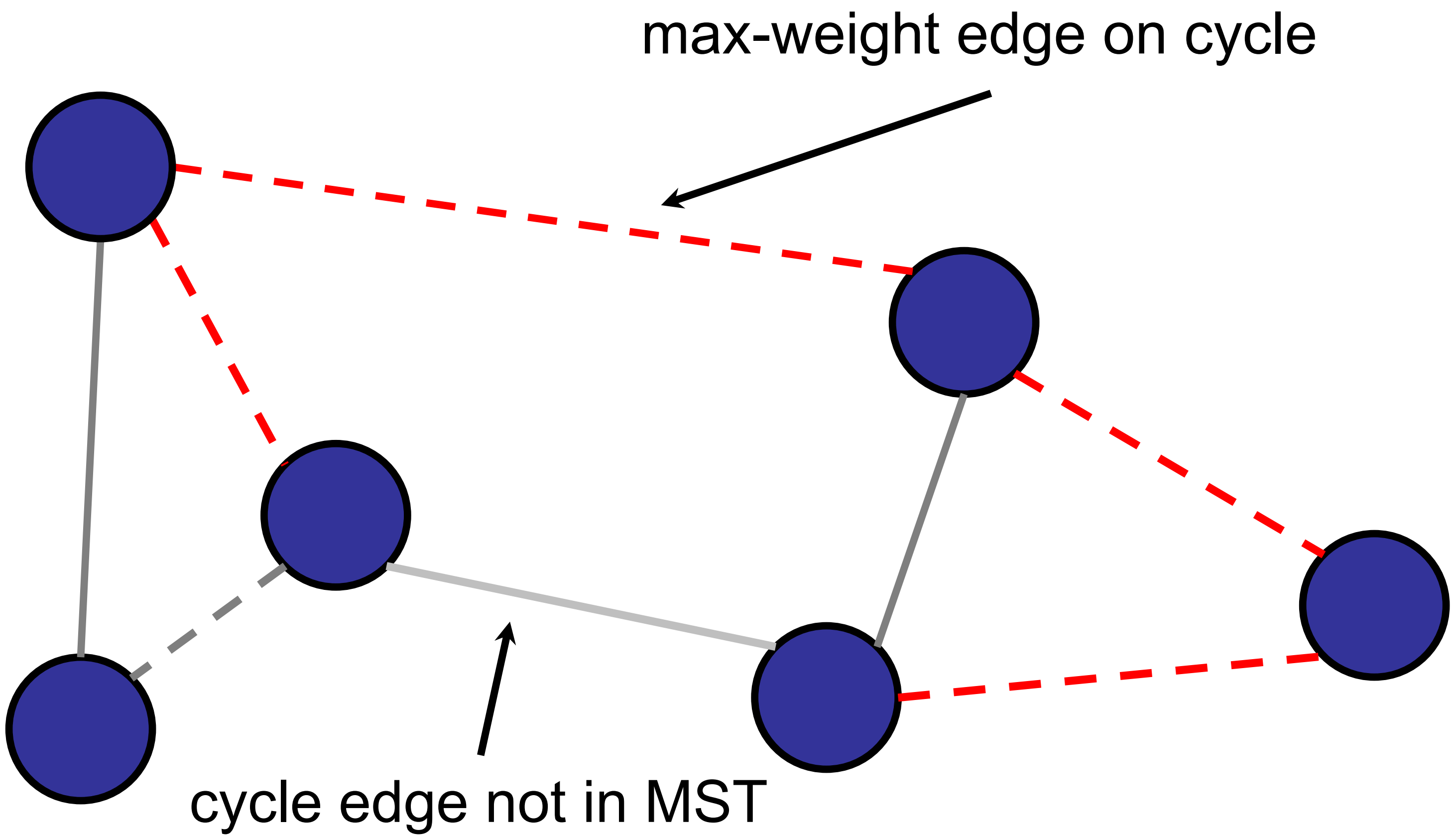
For every cycle, the maximum weight edge is ***not*** in the MST. (assuming distinct weights)



Properties of MST

Proof: Exchange argument

Assume heavy edge is in the MST.

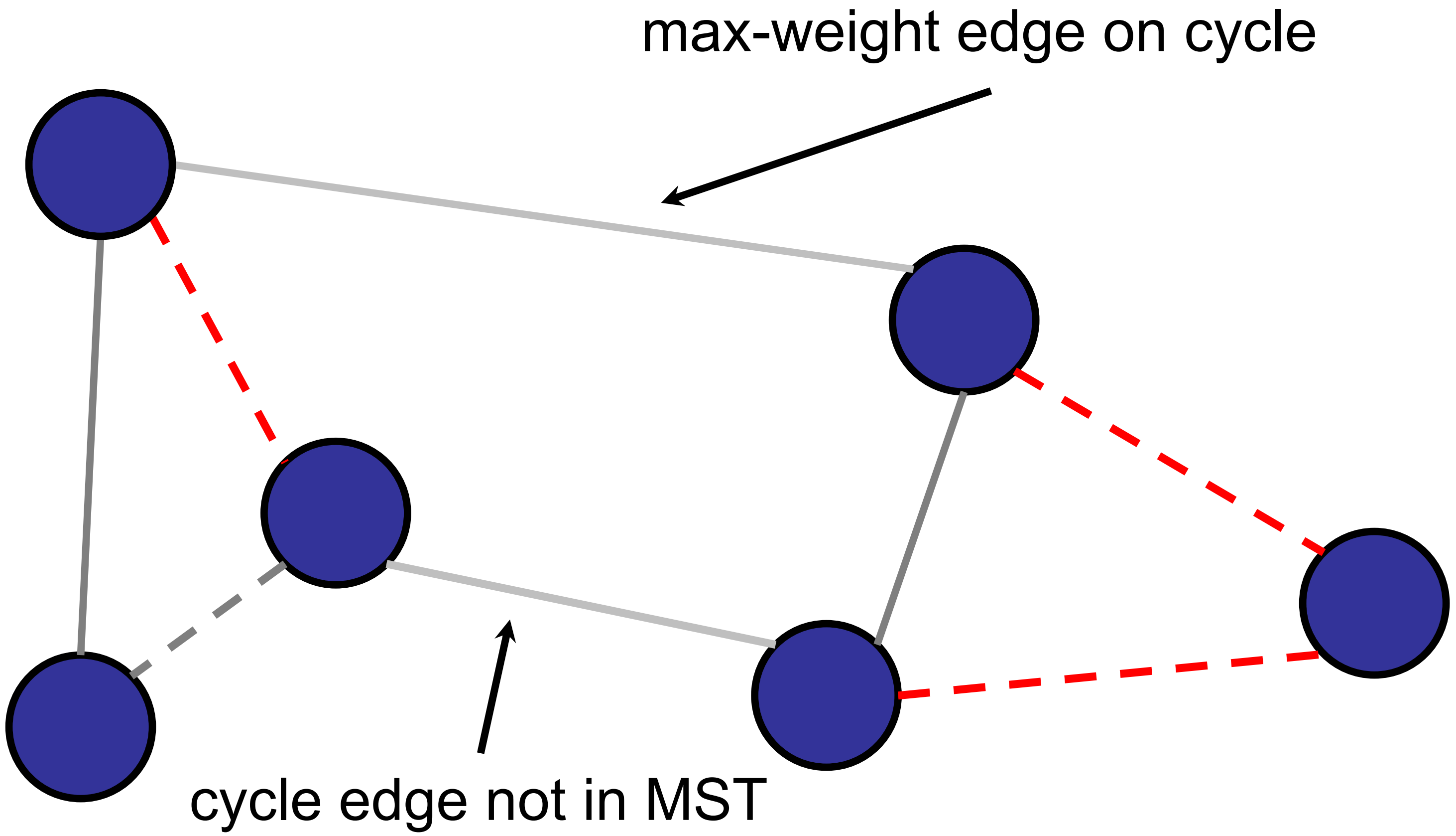


Properties of MST

Proof: Exchange argument

Assume heavy edge is in the MST.

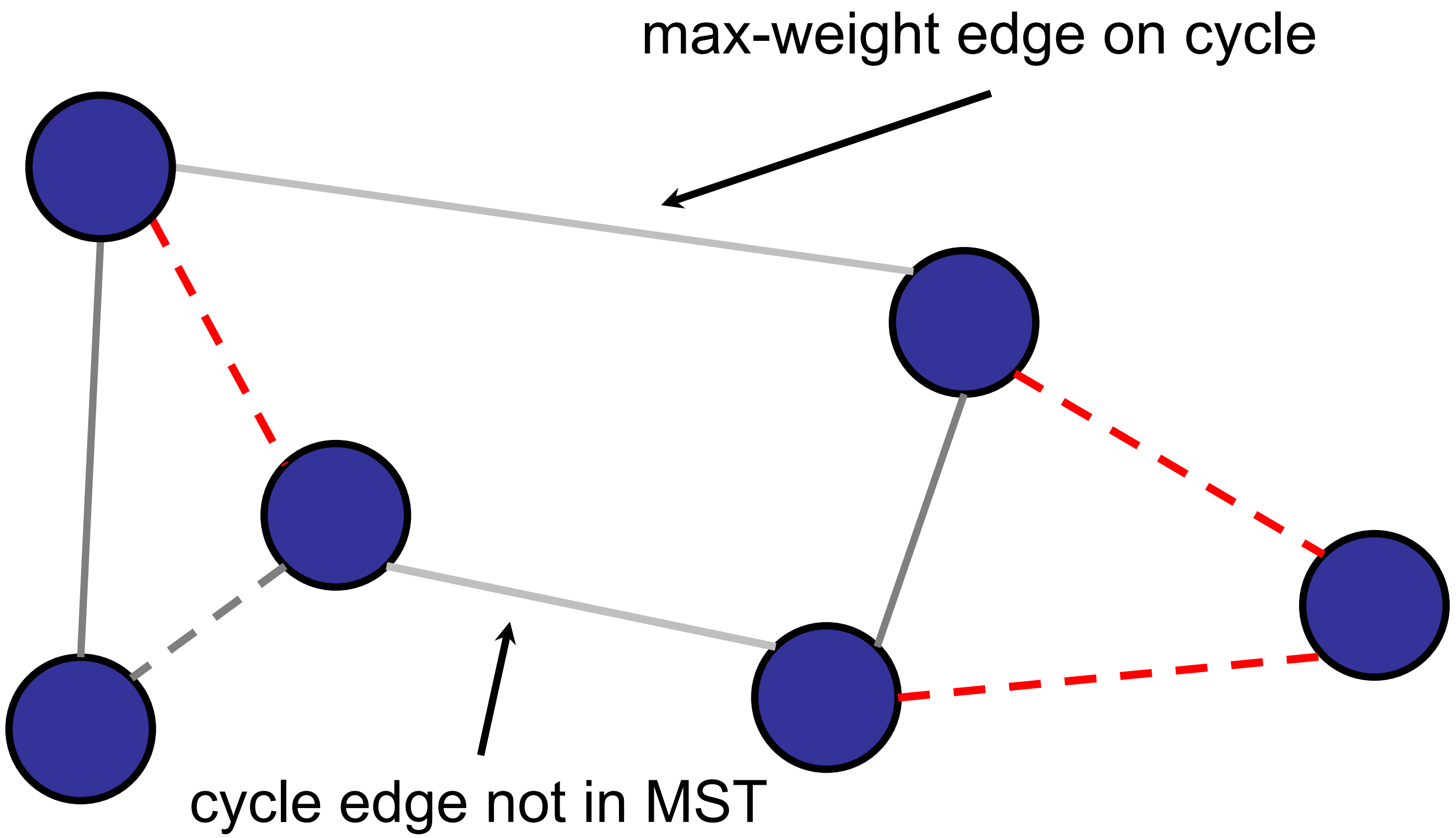
Remove max-weight edge; cuts graph.



Properties of MST

Proof: Exchange argument

There exists another cycle edge that crosses the cut.

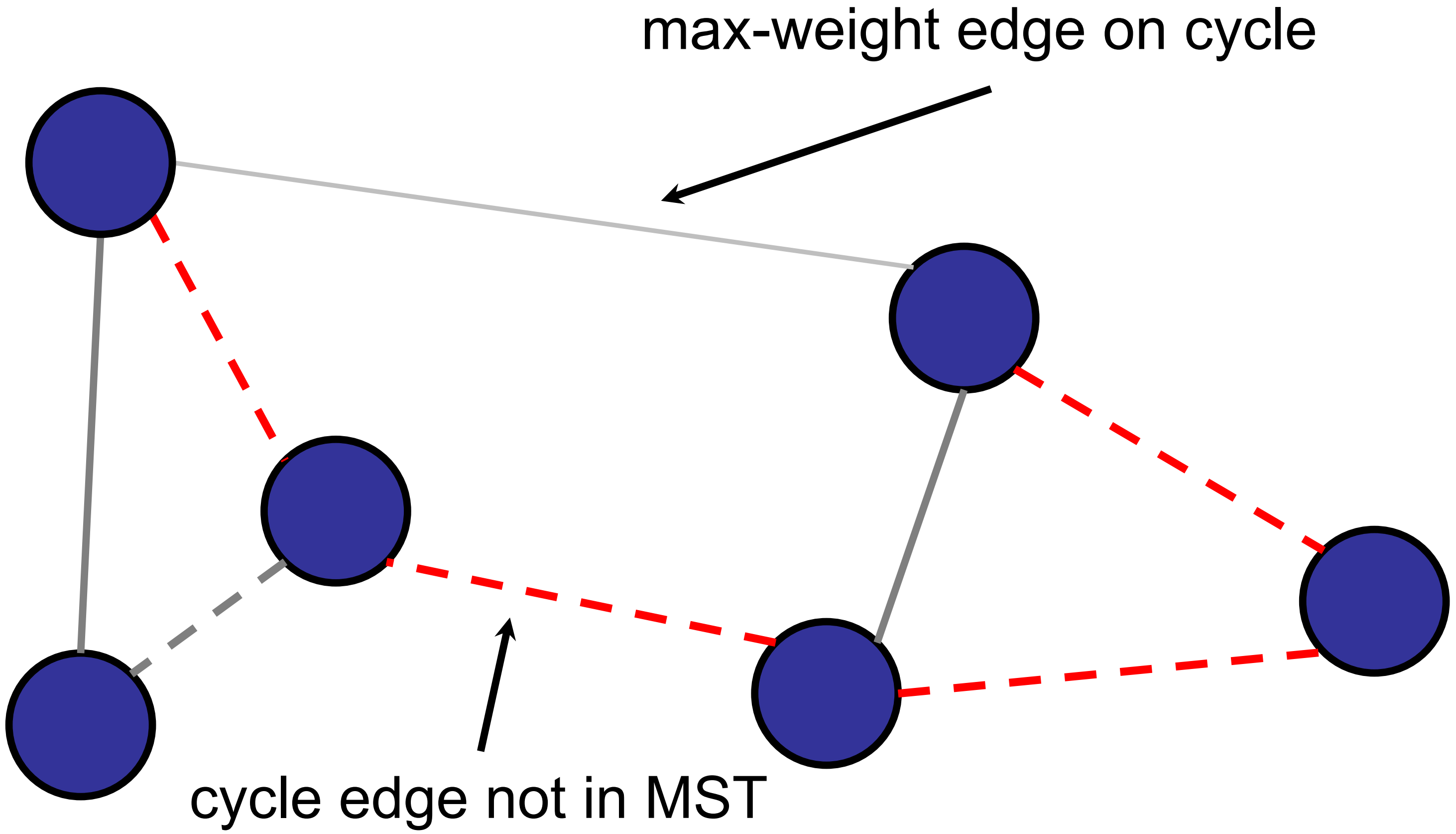


Properties of MST

Proof: Exchange argument

Replace heavy edge with lighter edge.

Still a spanning tree: Property 2.

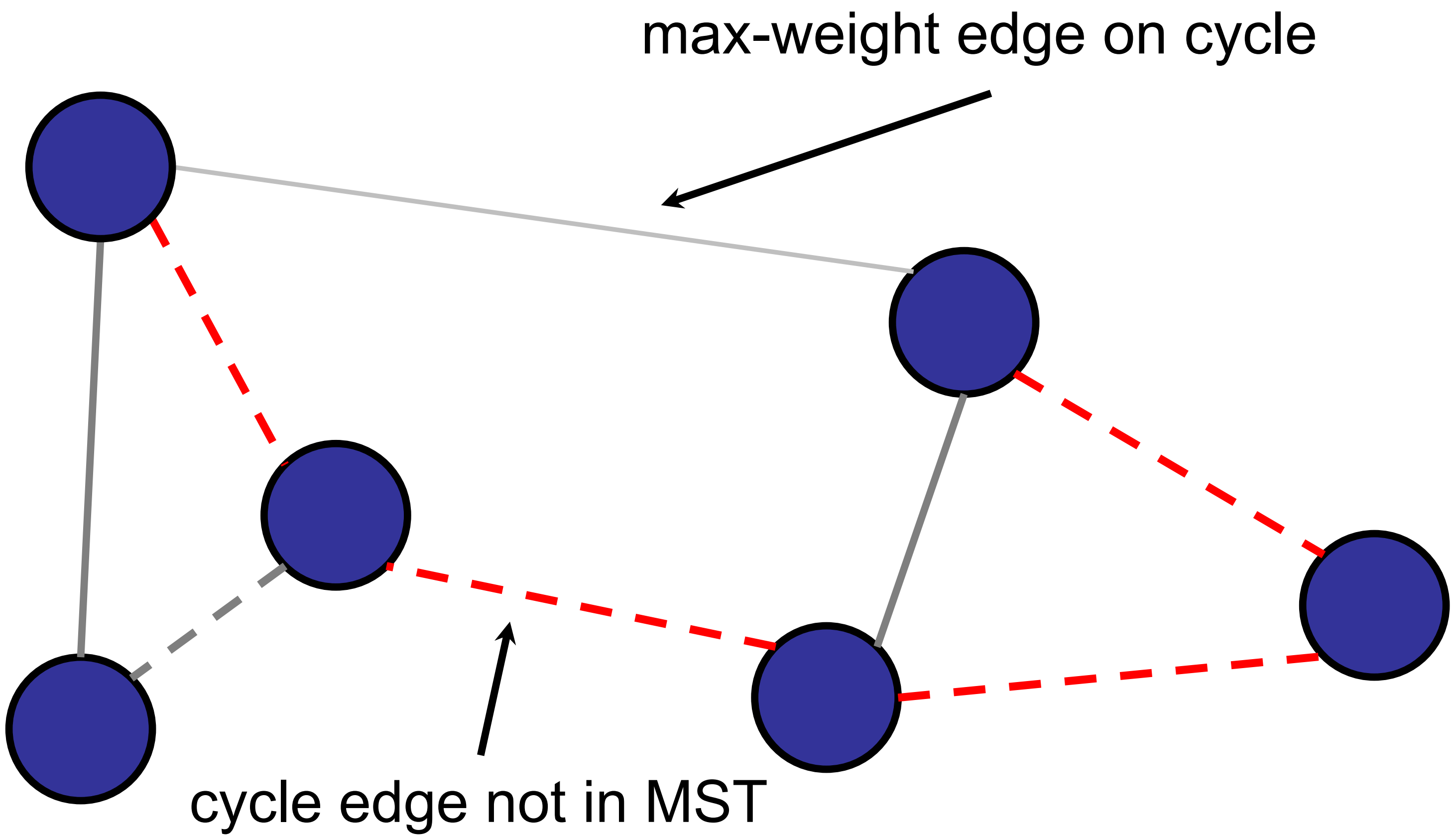


Properties of MST

Proof: Exchange argument

Replace heavy edge with lighter edge.

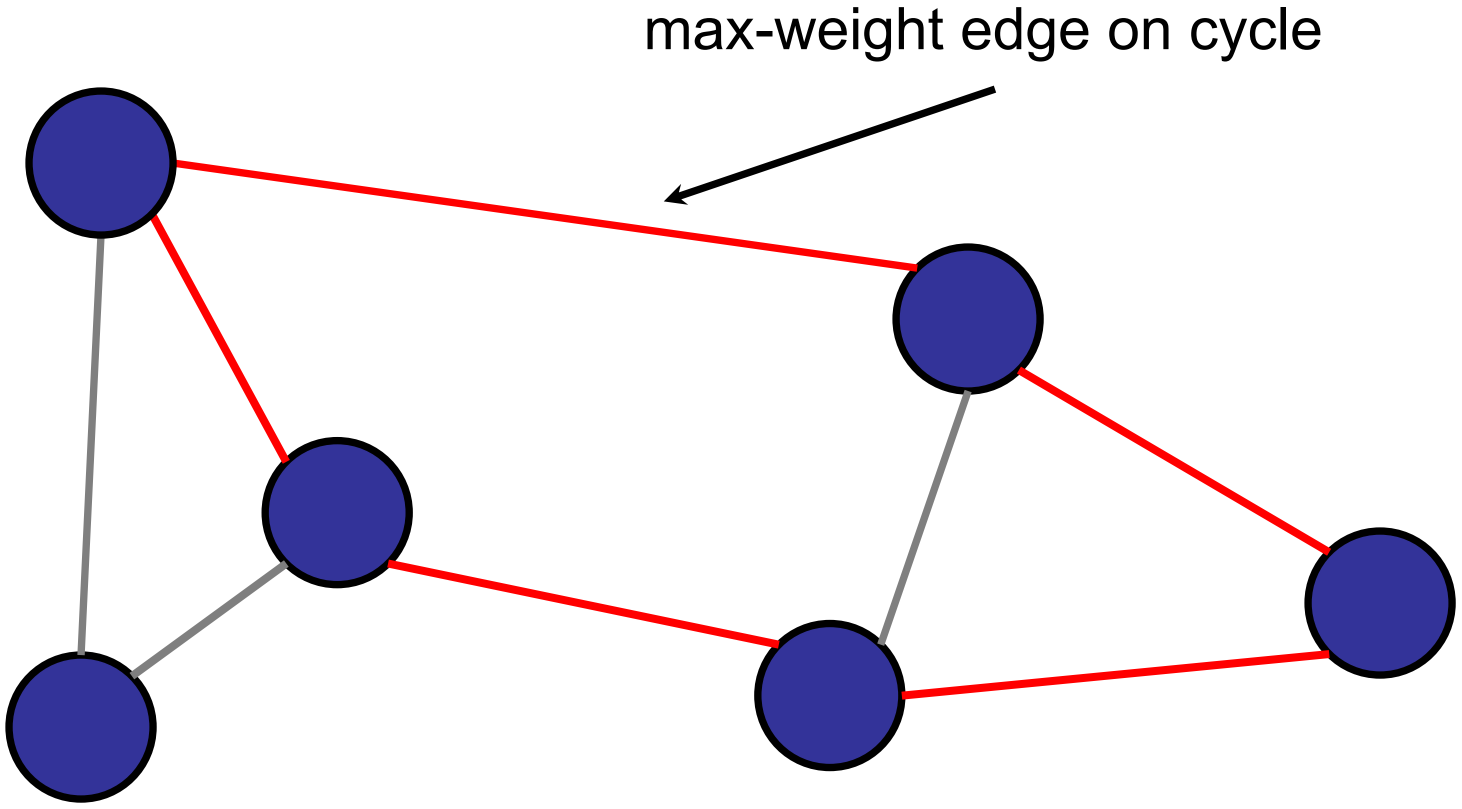
Less weight! Contradiction…



Properties of MST

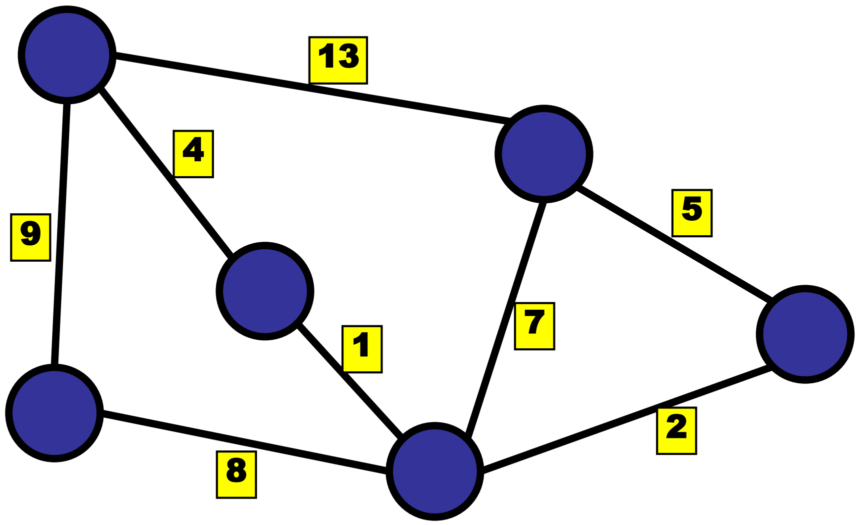
Property 3: Cycle property

For every cycle, the maximum weight edge is ***not*** in the MST.



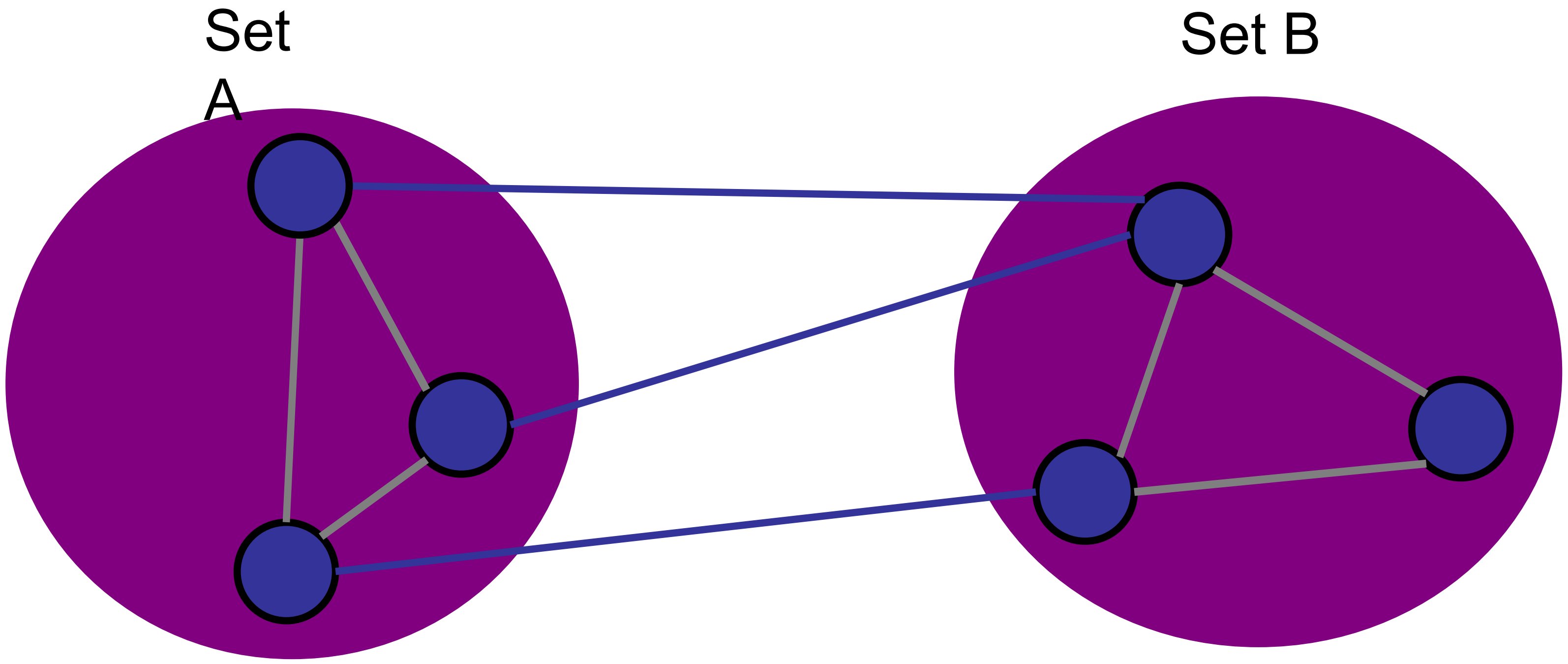
True or False:
<u>For every cycle</u>, the minimum weight edge is always in the MST.

1. True
2. [✓ False]
3. I don't know.



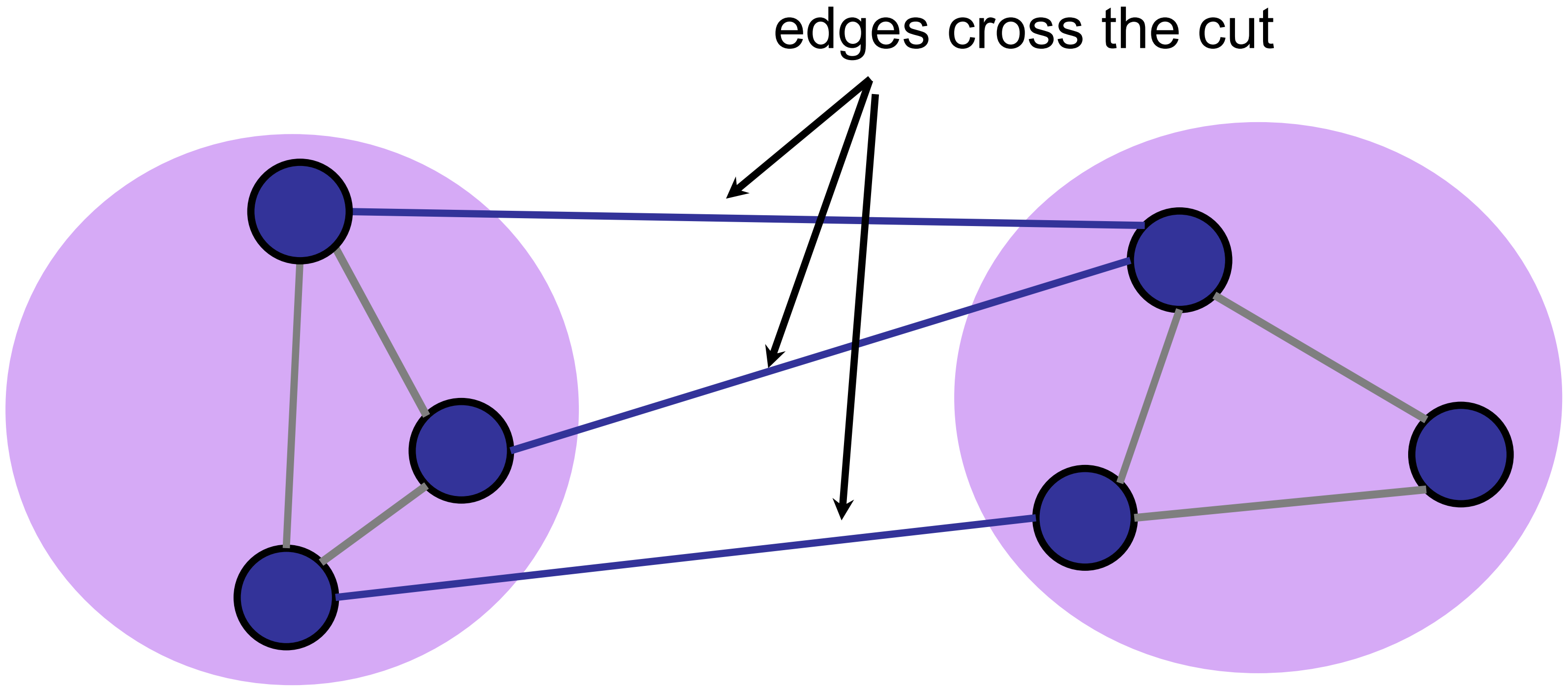
Properties of MST

Definition: A *cut* of a graph G=(V,E) is a partition of the vertices V into two disjoint subsets.



Properties of MST

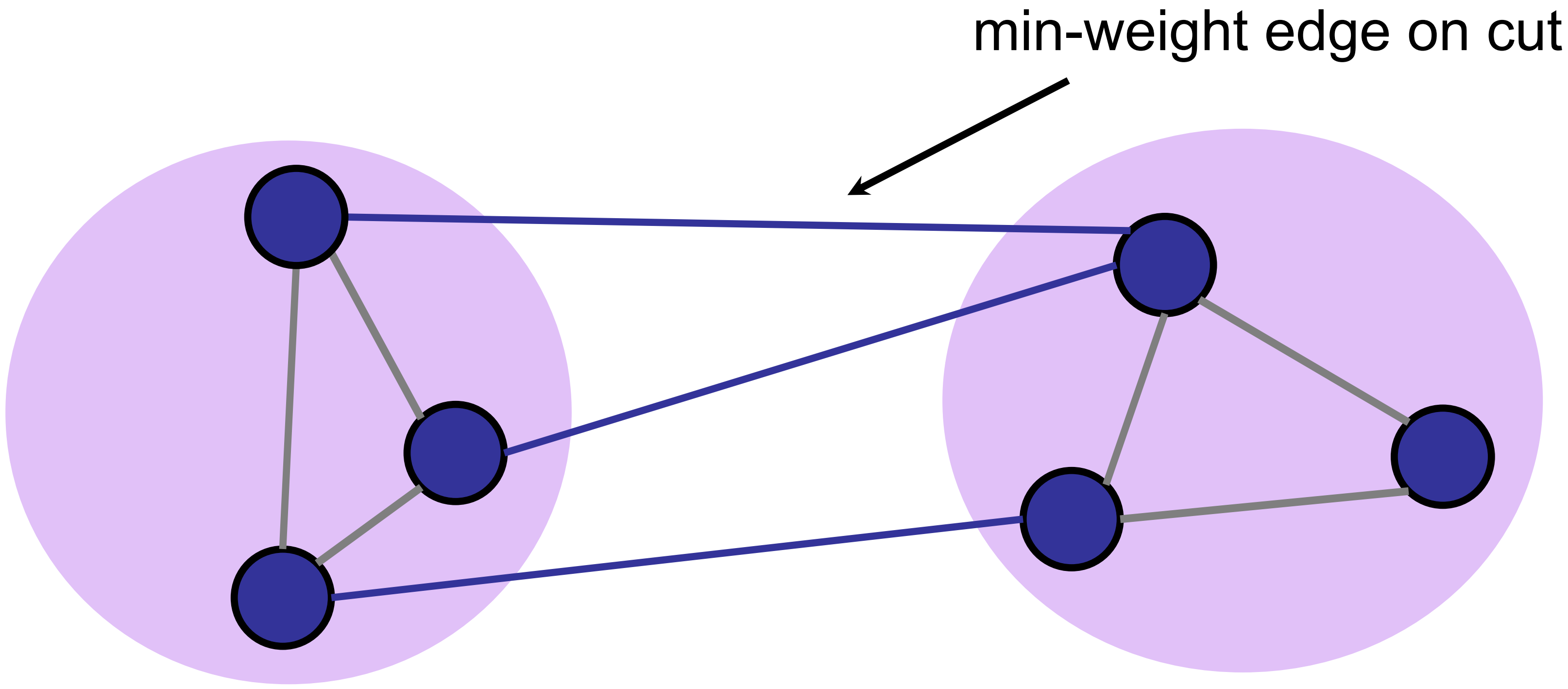
Definition: An edge *crosses a cut* if it has one vertex in each of the two sets.



Properties of MST

Property 4: Cut property

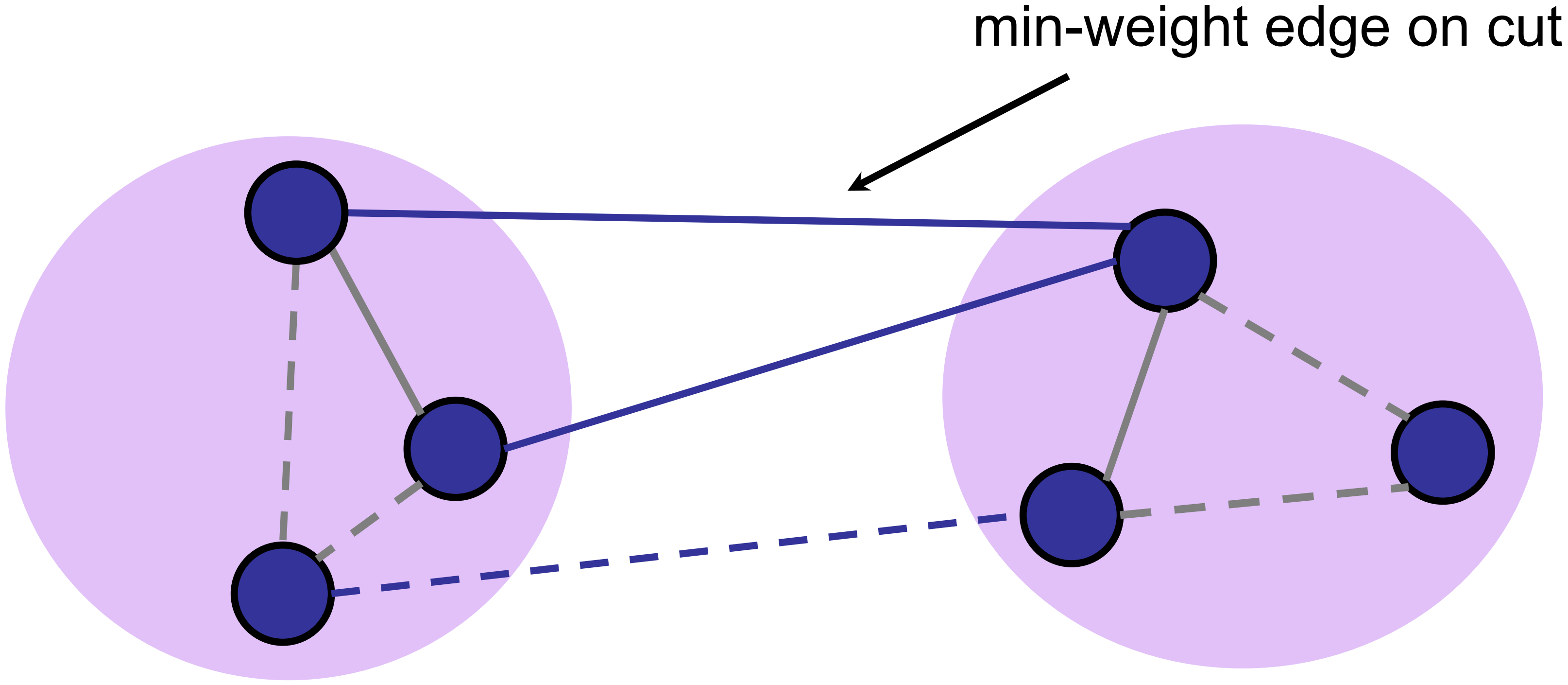
For every partition of the nodes, the minimum weight edge across the cut *is* in the MST.



Properties of MST

Proof: Exchange argument

Assume not.

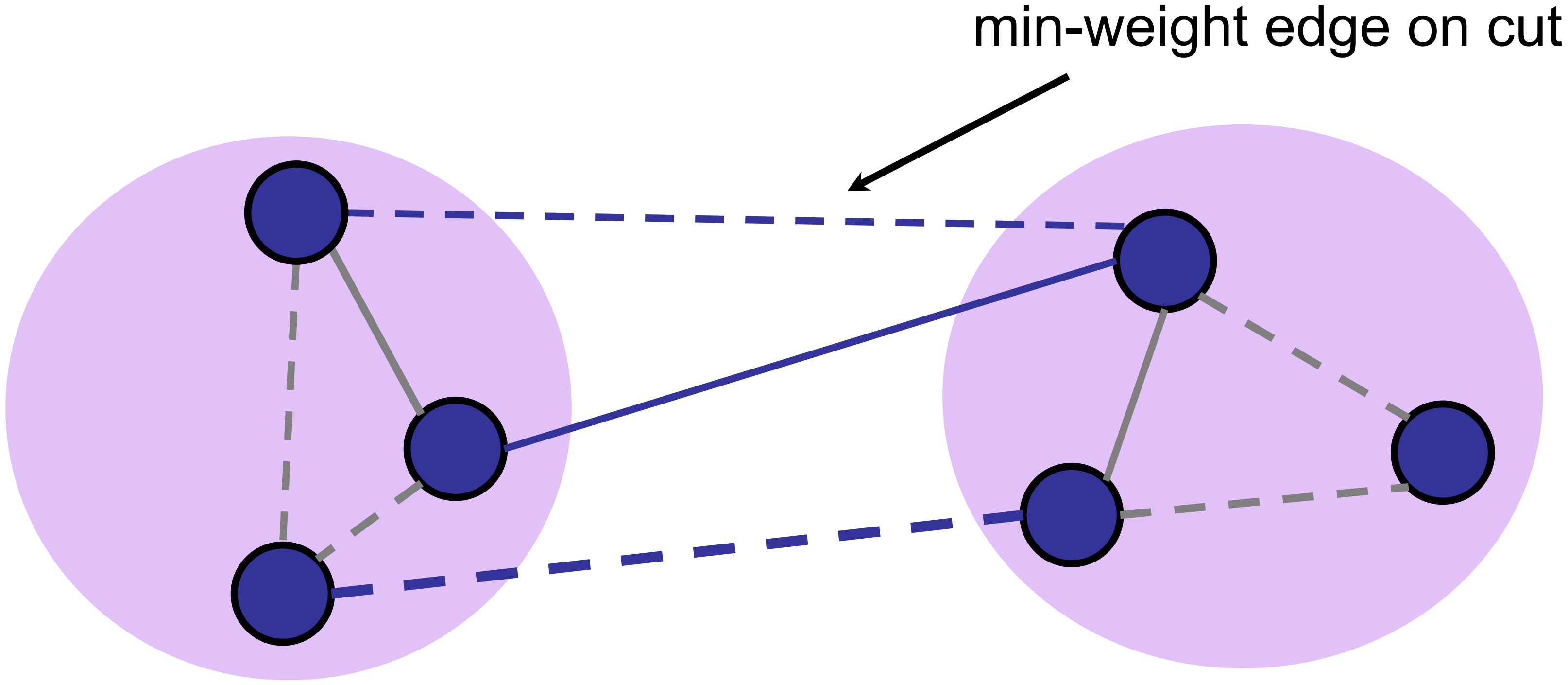


Properties of MST

Proof: Exchange argument

Assume not.

Add min-weight edge on cut.



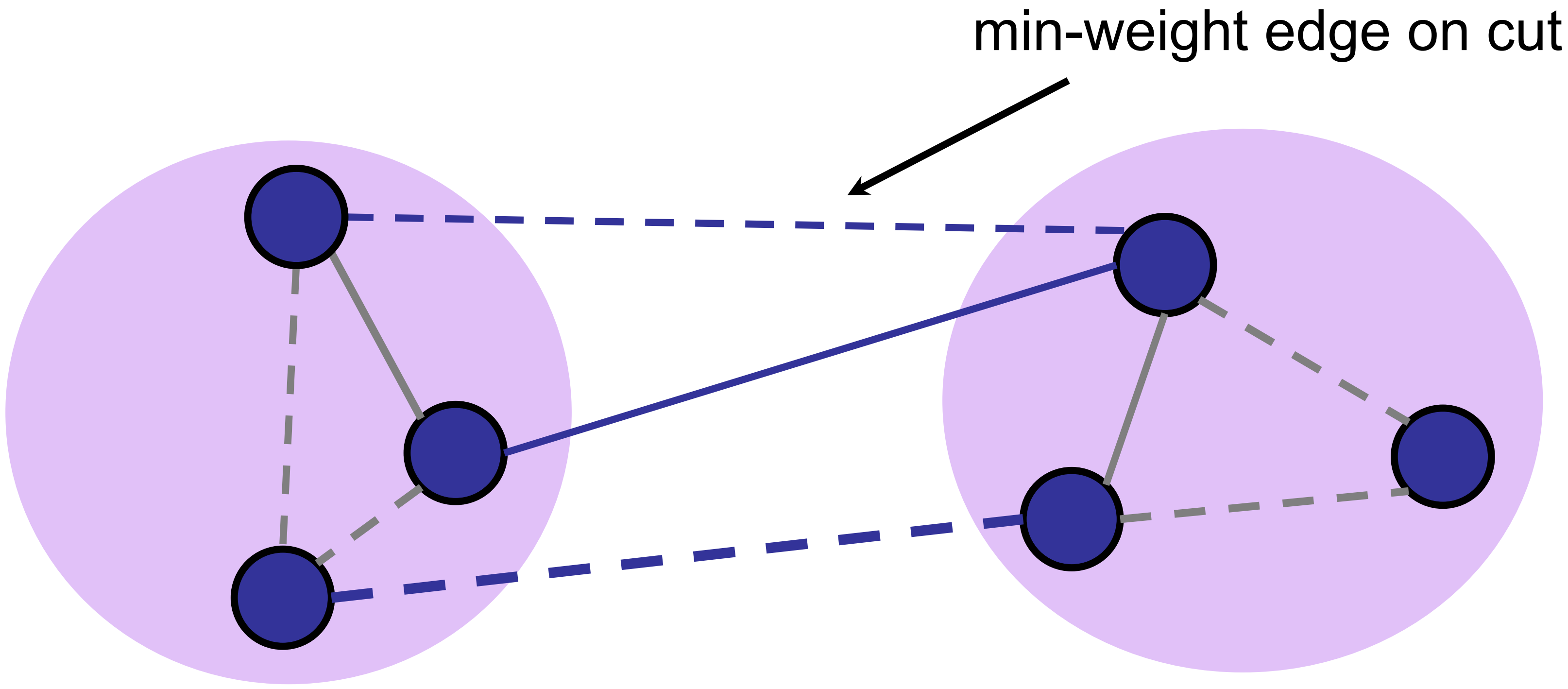
Properties of MST

Proof: Exchange argument

Assume not.

Add min-weight edge on cut.

Oops, creates a cycle!



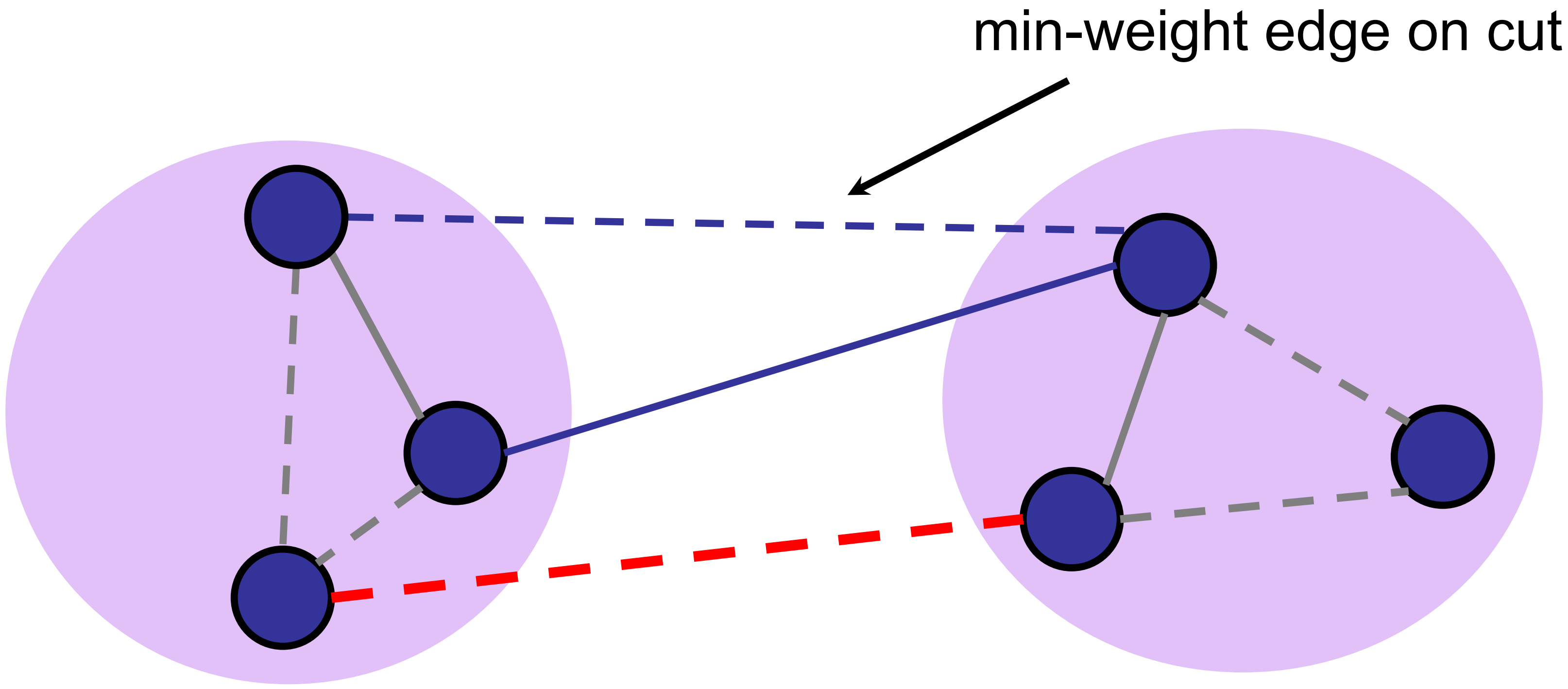
Properties of MST

Proof: Exchange argument

Assume not.

Add min-weight edge on cut.

Remove heaviest edge on cycle.



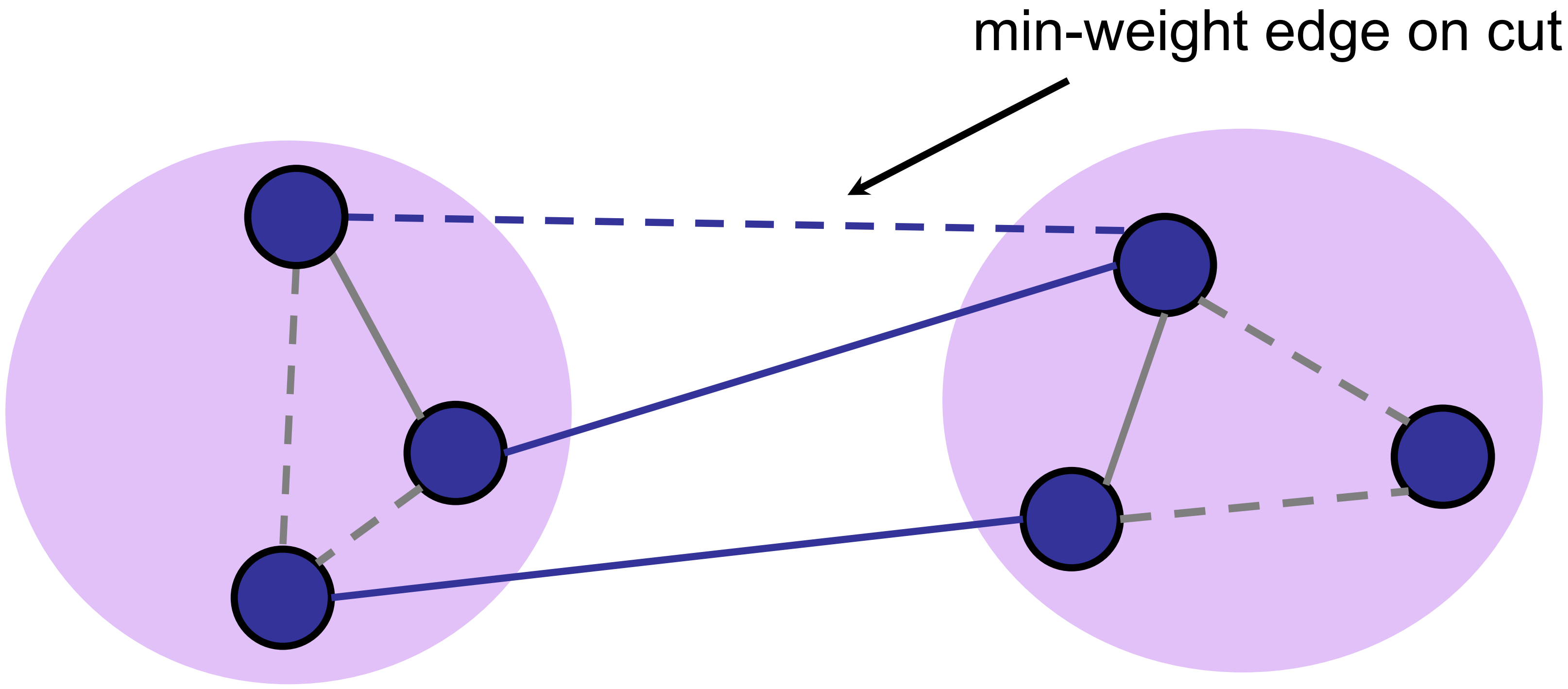
Properties of MST

Proof: Exchange argument

Assume not.

Add min-weight edge on cut.

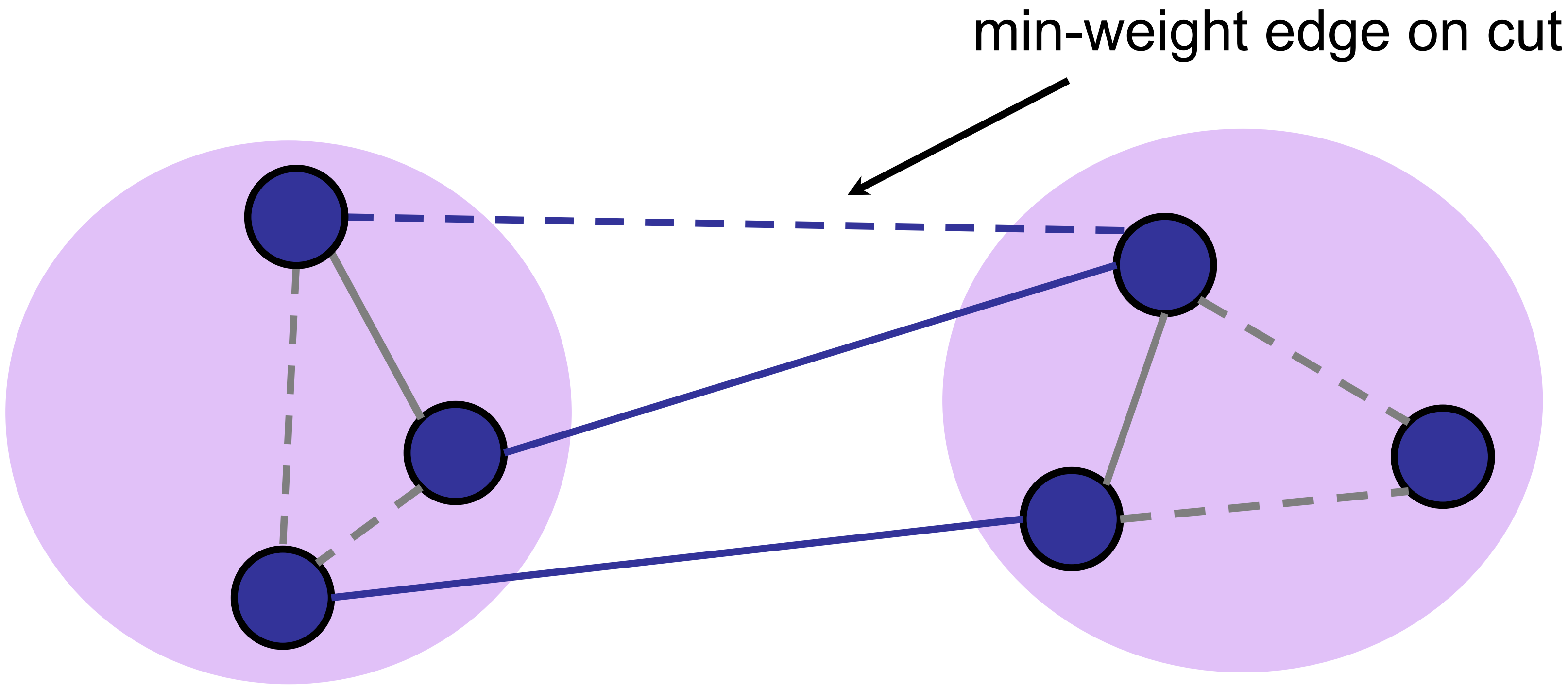
Remove heaviest edge on cycle.



Properties of MST

Proof: Exchange argument

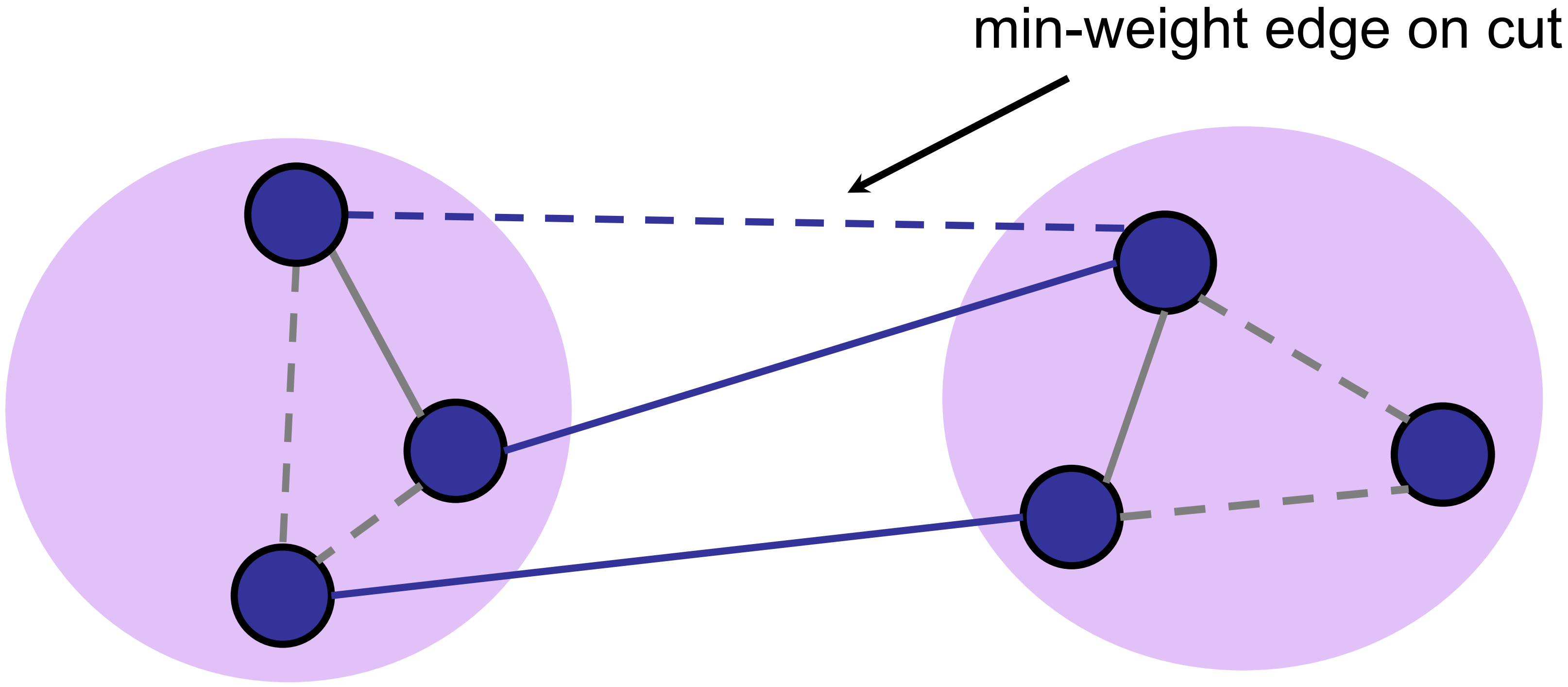
Result: a new spanning tree.



Properties of MST

Proof: Exchange argument

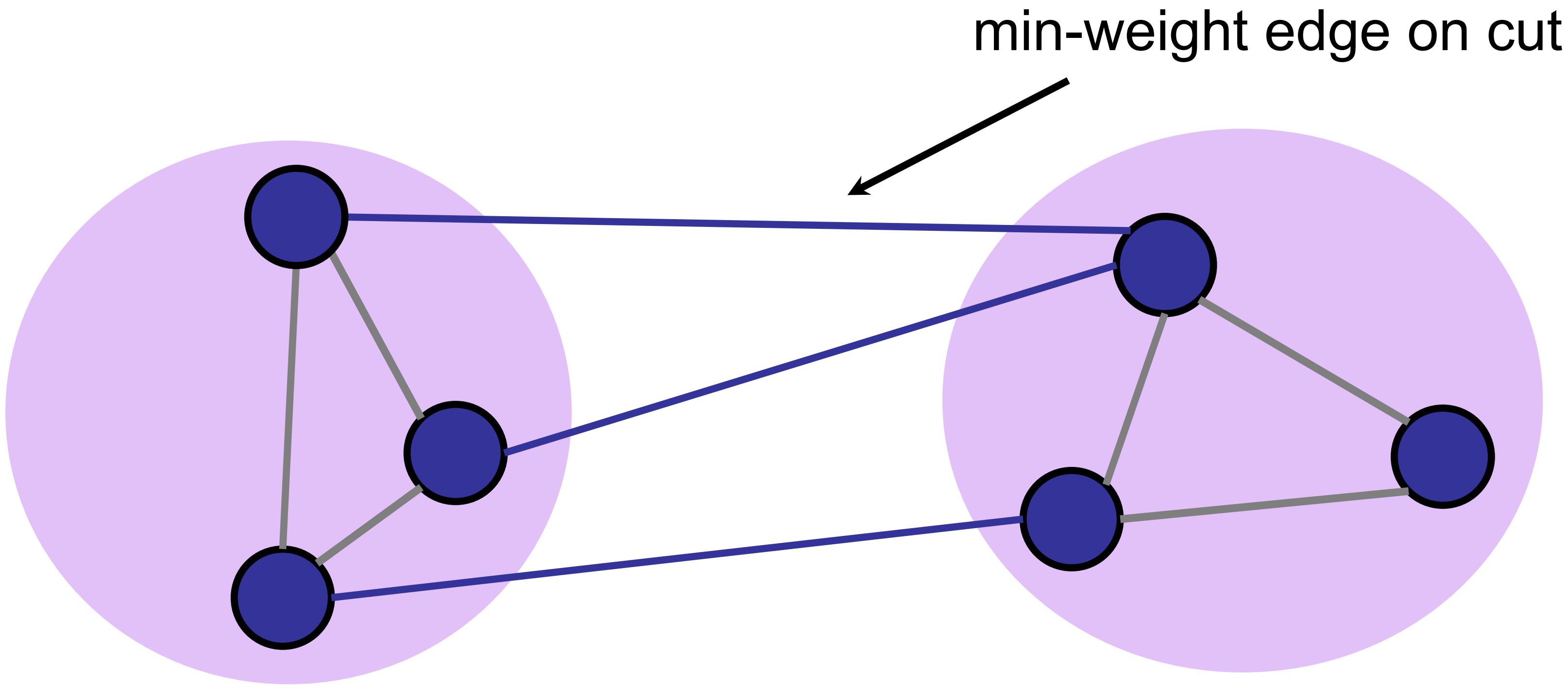
Less weight: replaced heavier edge with lighter edge.



Properties of MST

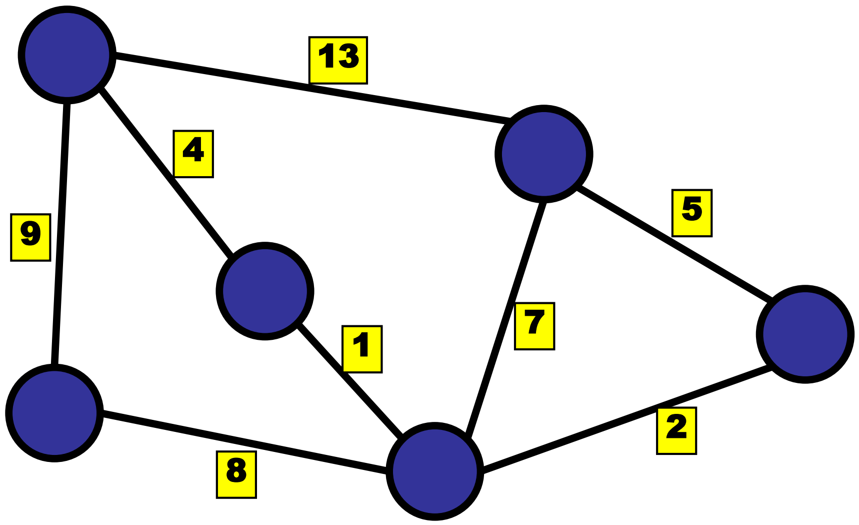
Property 4: Cut property

For every partition of the nodes, the minimum weight edge across the cut *is* in the MST.



True or False:
For every vertex, the minimum outgoing edge is always part of the MST.

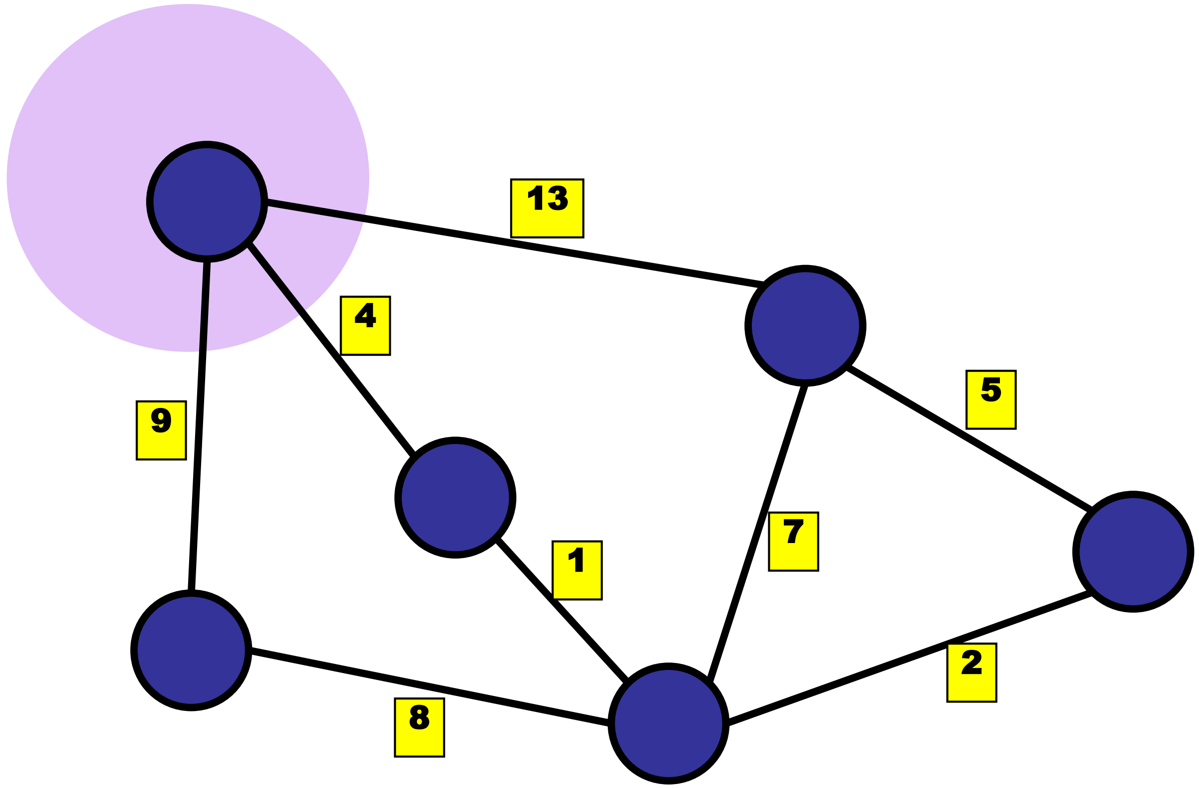
[✓ 1. True]
2. False
3. I don’t know.



Properties of MST

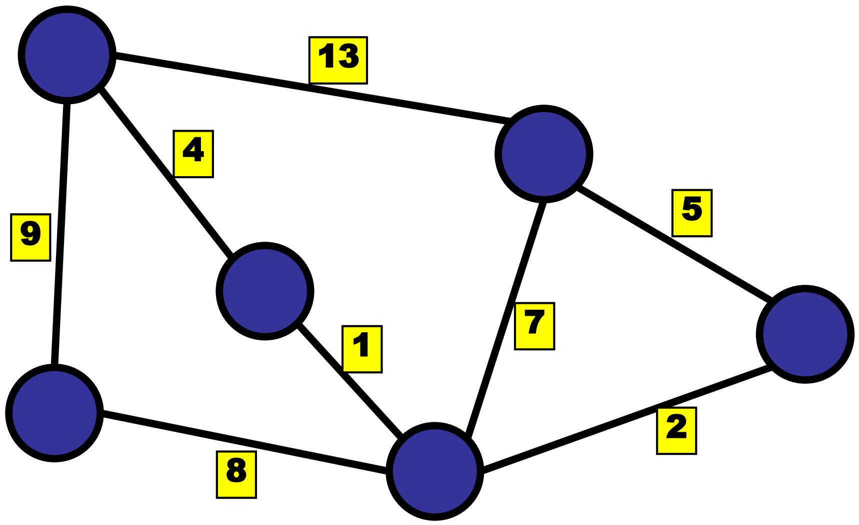
Property 4: Cut property

For every partition of the nodes, the minimum weight edge across the cut *is* in the MST.



True or False:
For every vertex, the maximum outgoing edge is never part of the MST.

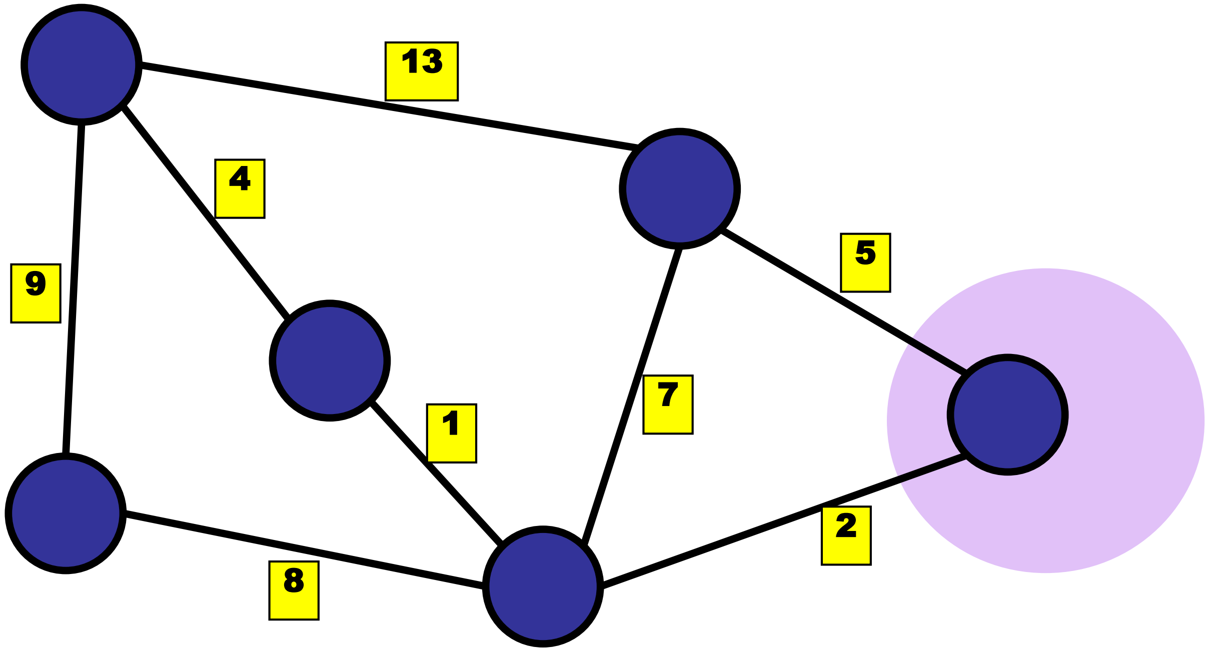
1. True
2. [✓ False]
3. I don't know.



Properties of MST

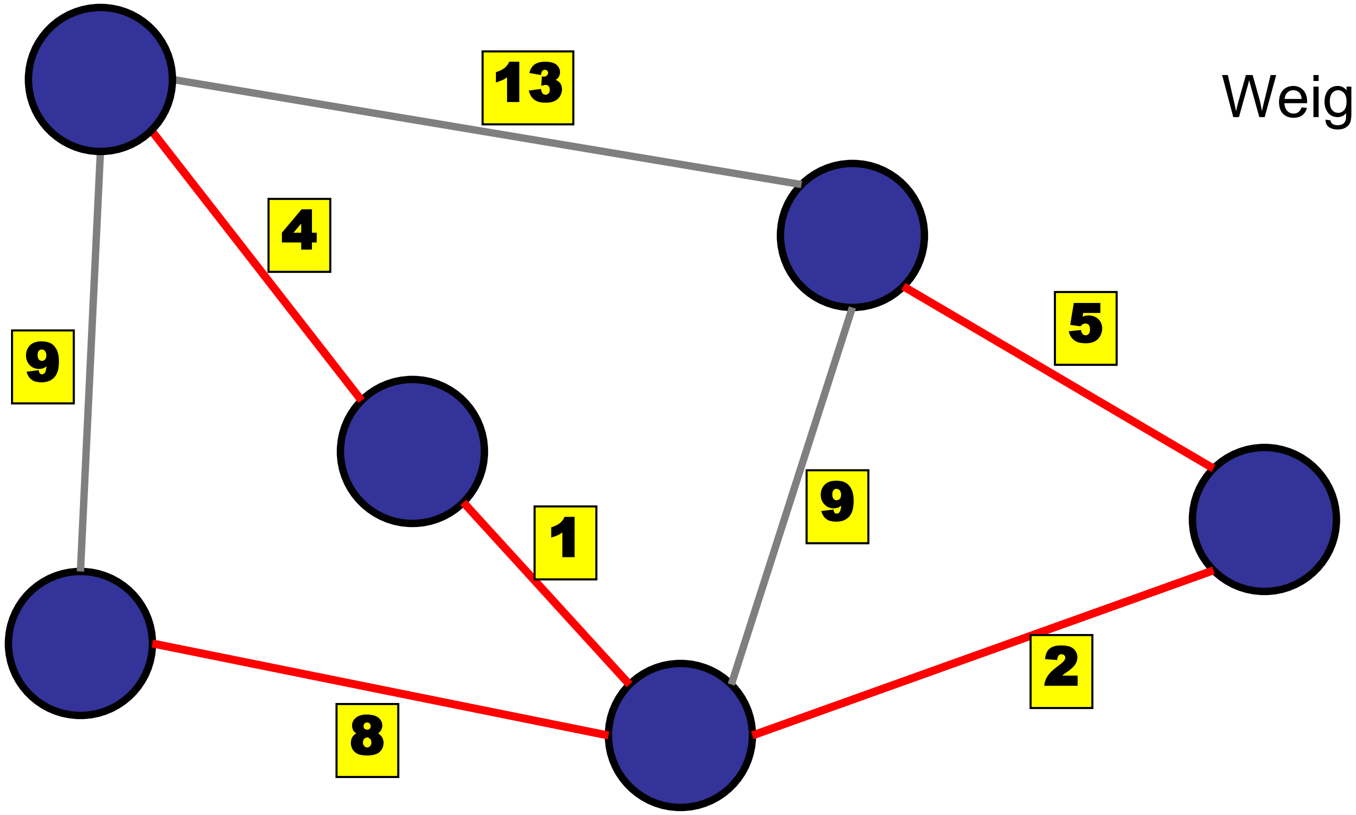
Property 4: Cut property

For every partition of the nodes, the minimum weight edge across the cut *is* in the MST.



Minimum Spanning Tree

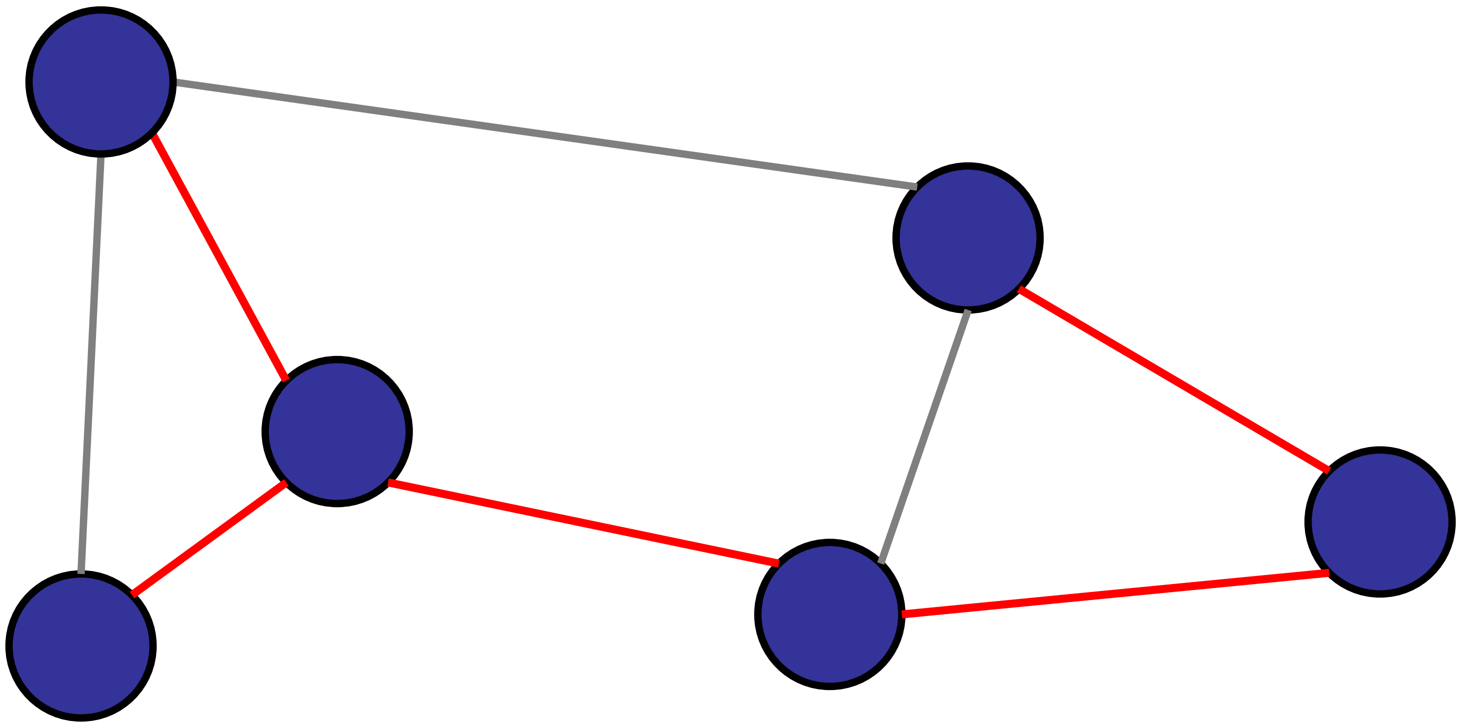
Definition: a spanning tree with minimum weight



Weight: 20

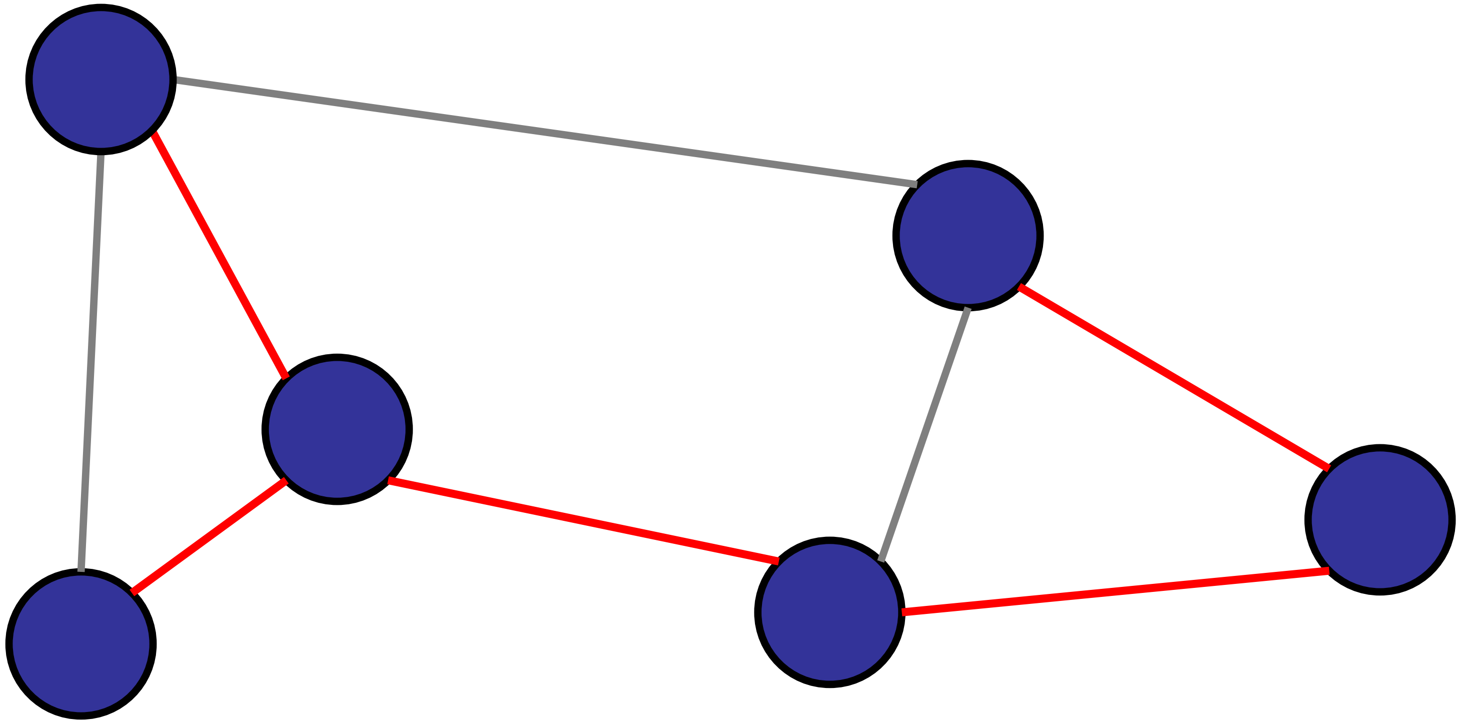
Properties of MST

Property 1: No cycles



Properties of MST

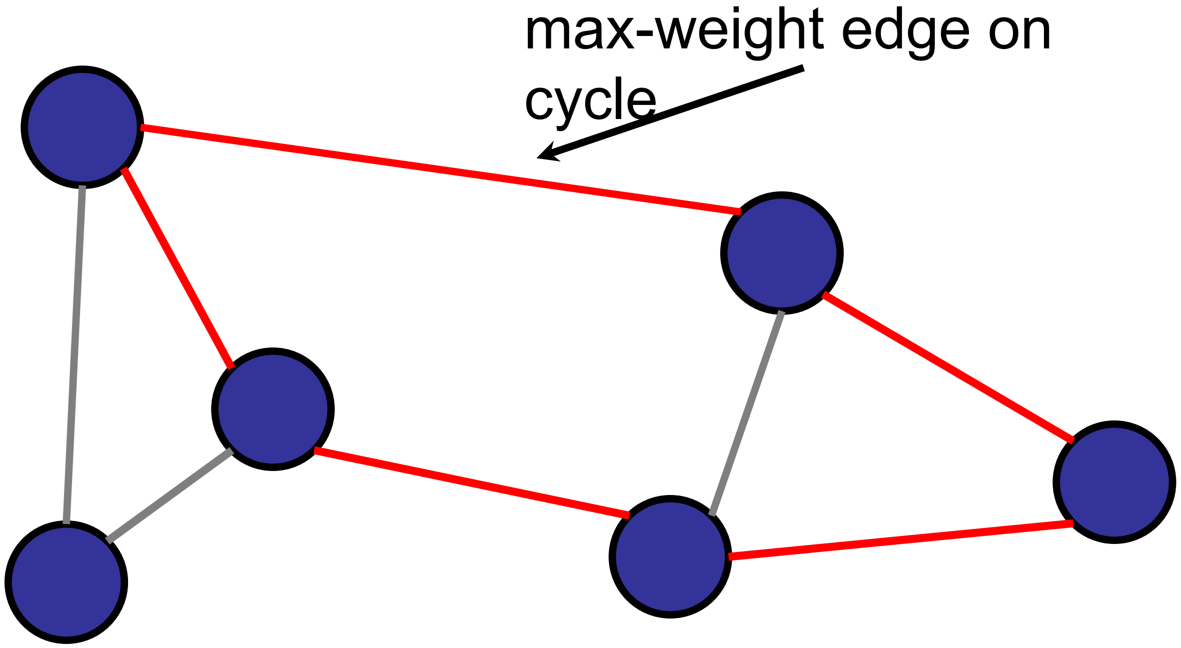
Property 2: **If you cut an MST**, the two pieces are both MSTs.



Properties of MST

Property 3: Cycle property

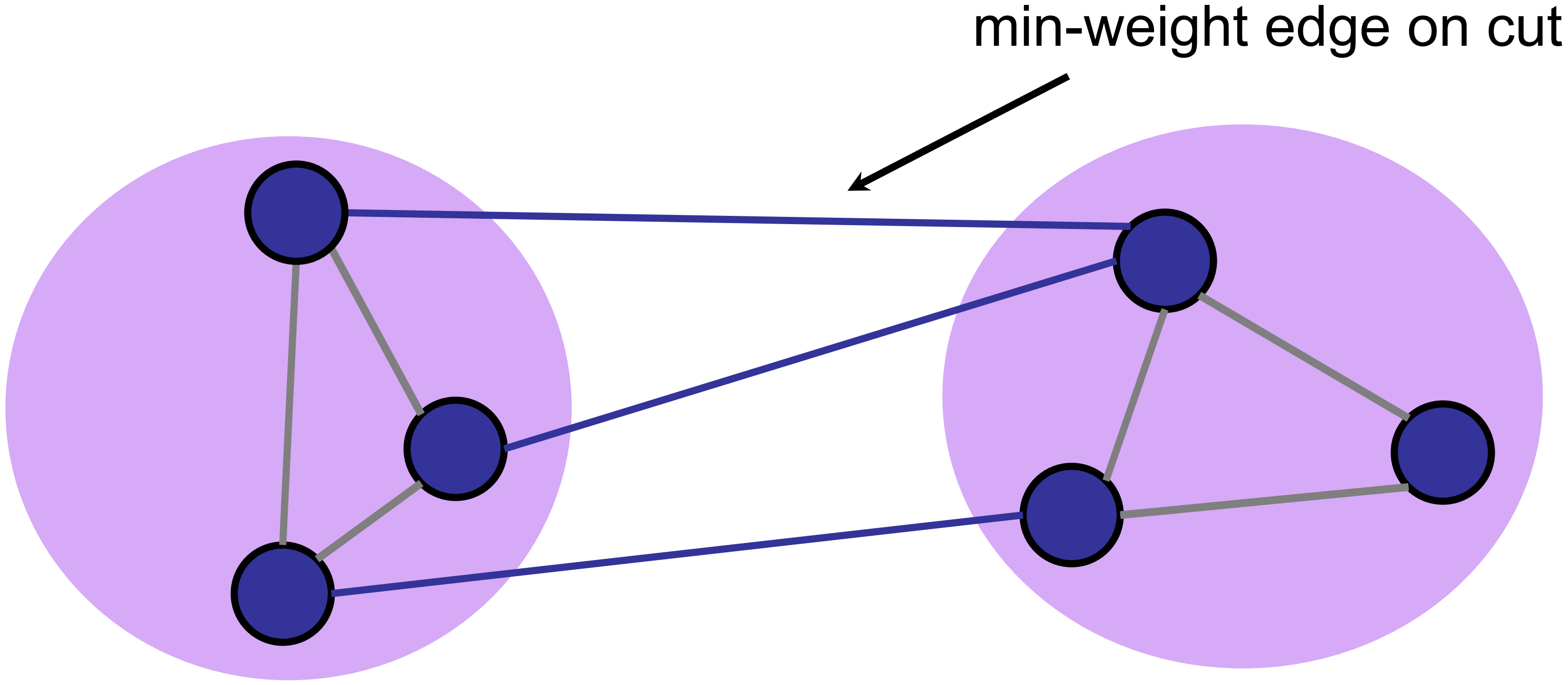
For every cycle, the maximum weight edge is *not* in the MST.



Properties of MST

Property 4: Cut property

For every cut D, the minimum weight edge that crosses the cut *is* in the MST.



Property of MST

- No cycles
- If you cut an MST, the two pieces are both MSTs.
- Cycle property
  - For every cycle, the maximum weight edge is not in the MST.
- Cut property
  - For every cut D, the minimum weight edge that crosses the cut is in the MST.

Roadmap

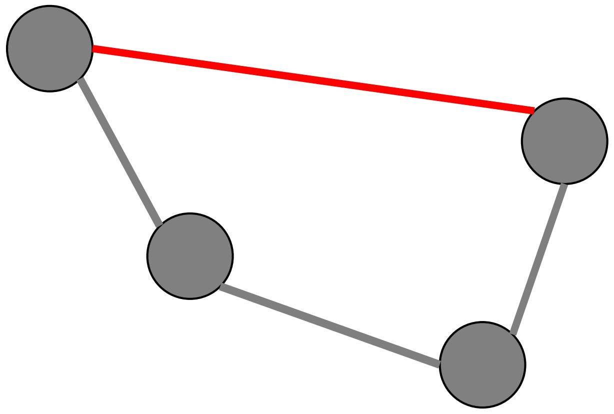
Minimum Spanning Trees

- The MST Problem
- Basic Properties of an MST
- **Generic MST Algorithm**
- Prim’s Algorithm
- Kruskal’s Algorithm
- Boruvka’s Algorithm
- Variations

Generic MST Algorithm

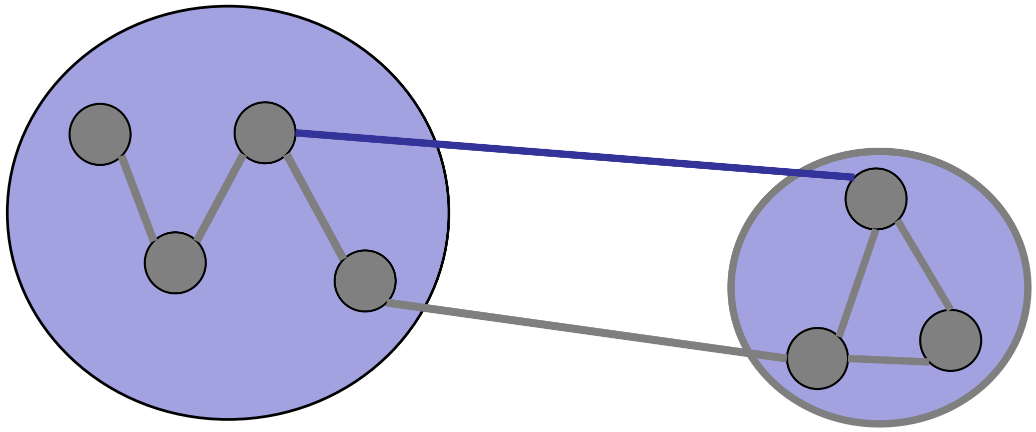
Red rule:

If C is a cycle with no red arcs, then color the max-weight edge in C red.



Blue rule:

If D is a cut with no blue arcs, then color the min-weight edge in D blue.



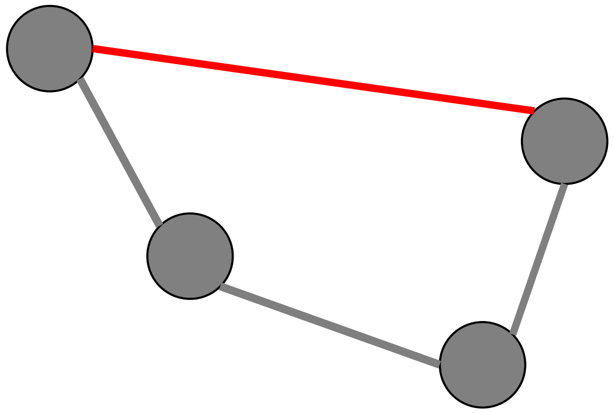
Generic MST Algorithm

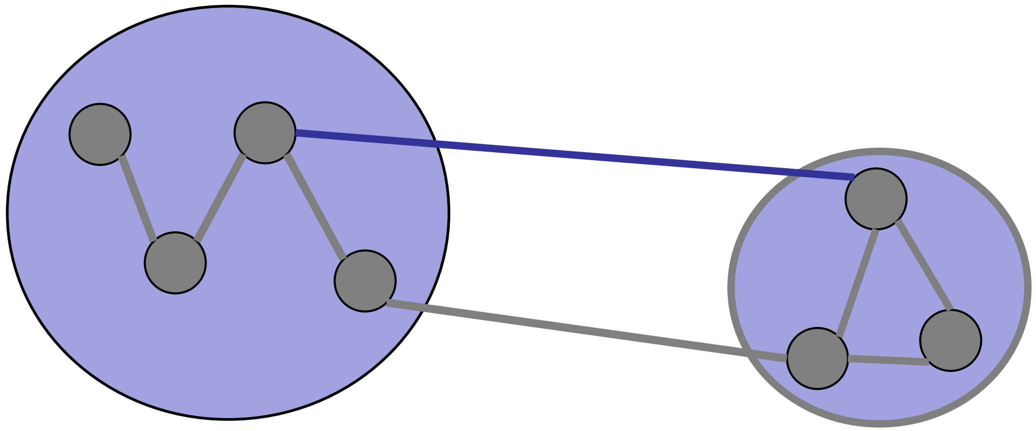
Greedy Algorithm:

Repeat:

Apply red rule or blue rule to an arbitrary edge.

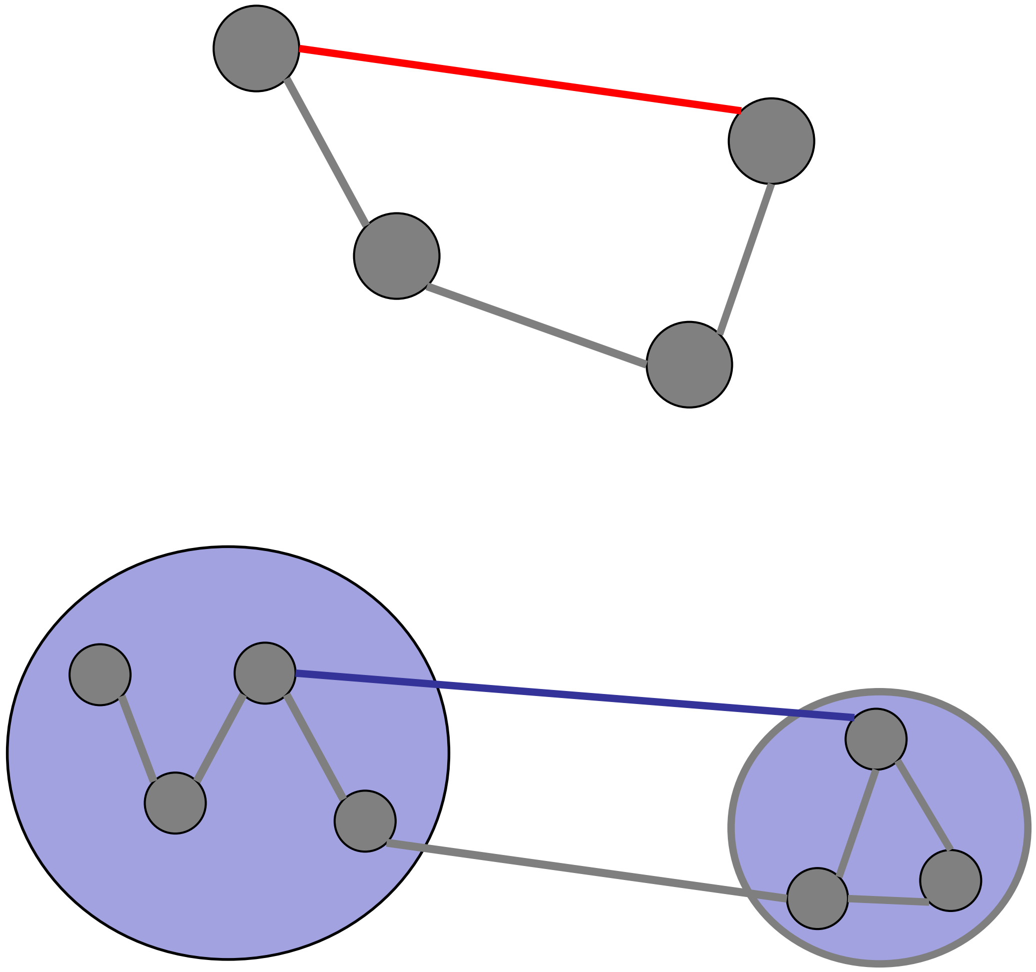
until all edges are either **blue** or **red**.





Generic MST Algorithm

Claim: On termination, the blue edges are an MST.

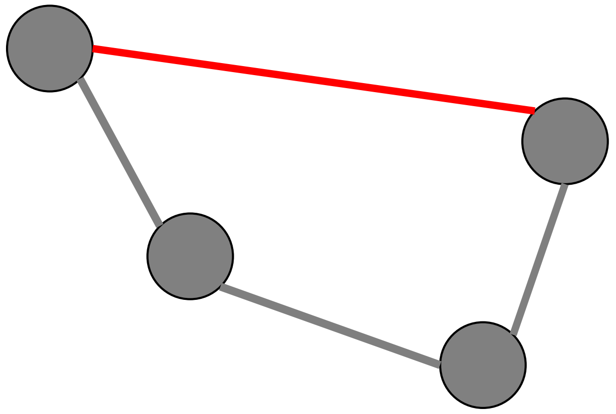


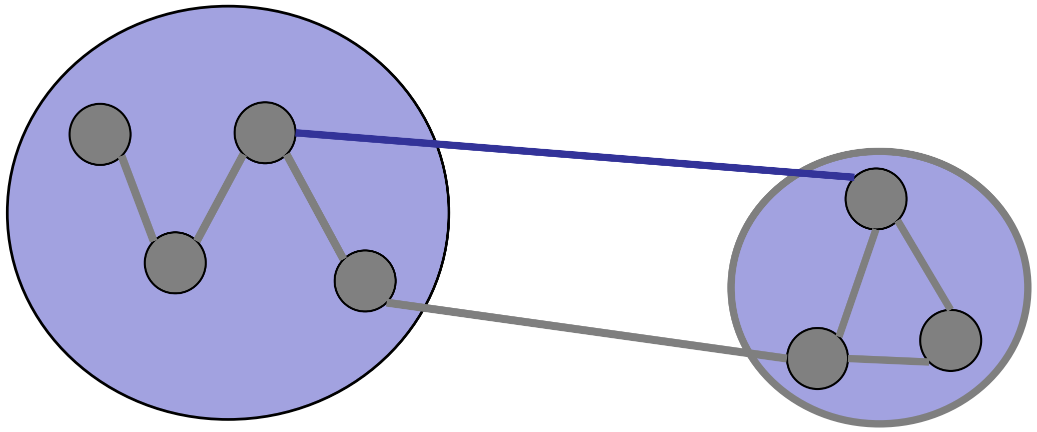
Generic MST Algorithm

Claim: On termination, the blue edges are an MST.

On termination:

1. Every cycle has a red edge.
   No blue cycles ☐ forest.



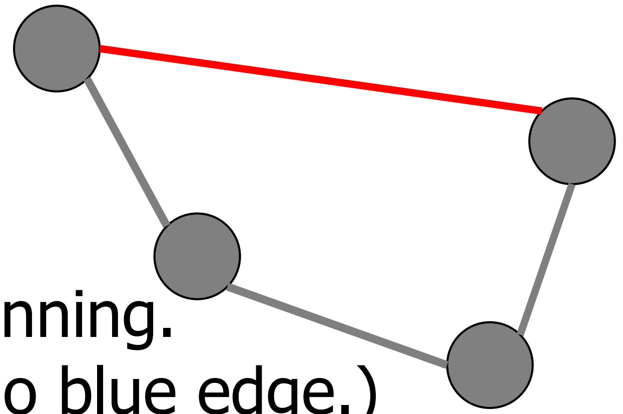


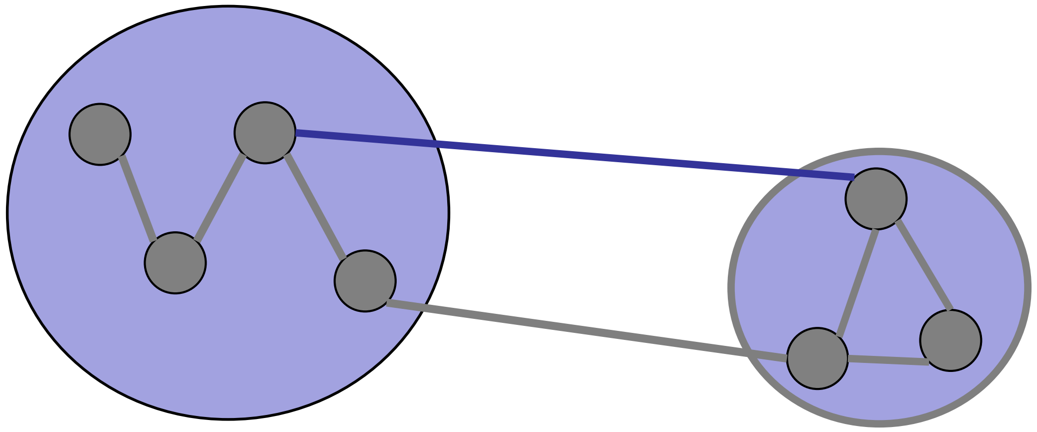
Generic MST Algorithm

Claim: On termination, the blue edges are an MST.

On termination:

1. Every cycle has a red edge.
   No blue cycles ☐ forest.
2. Blue edges form a tree ☐ spanning.
   (Otherwise, there is a cut with no blue edge.)



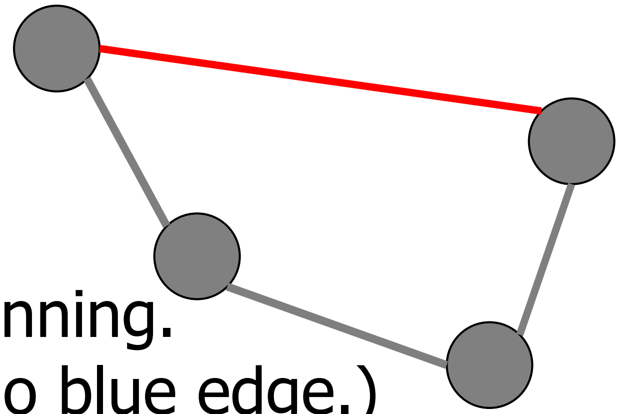


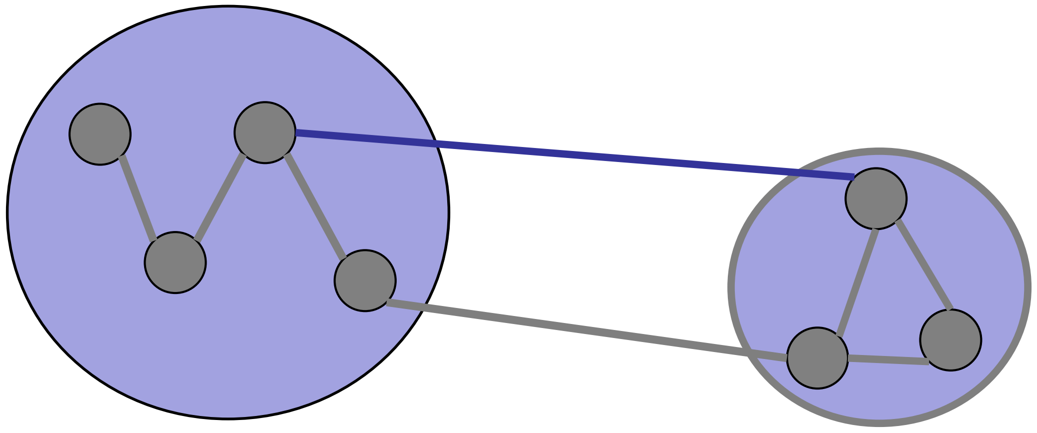
Generic MST Algorithm

Claim: On termination, the blue edges are an MST.

On termination:

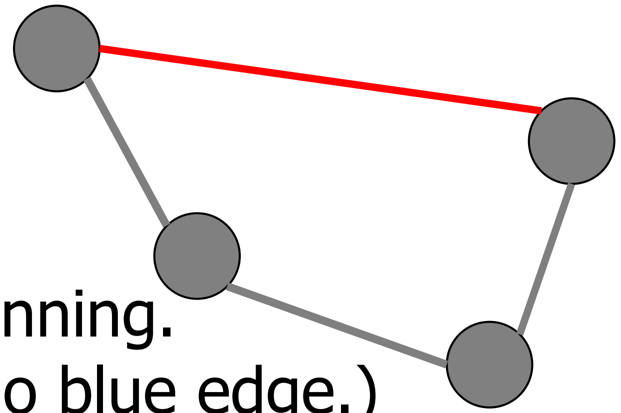
1. Every cycle has a red edge.
   No blue cycles □ forest.
2. Blue edges form a tree □ spanning.
   (Otherwise, there is a cut with no blue edge.)
3. Every edge is colored.





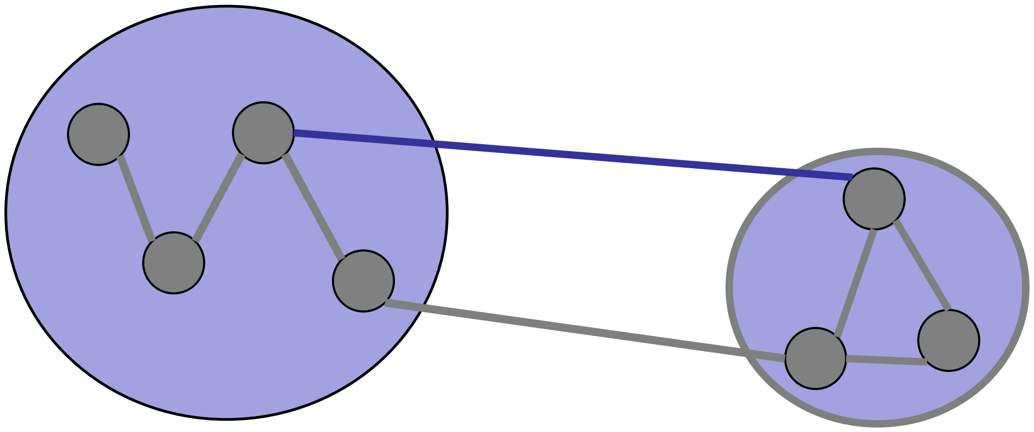
Generic MST Algorithm

Claim: On termination, the blue edges are an MST.



On termination:

1. Every cycle has a red edge.
   No blue cycles ☐ forest.
2. Blue edges form a tree ☐ spanning.
   (Otherwise, there is a cut with no blue edge.)
3. Every edge is colored.
4. Every blue edge is in the MST (Property 4).



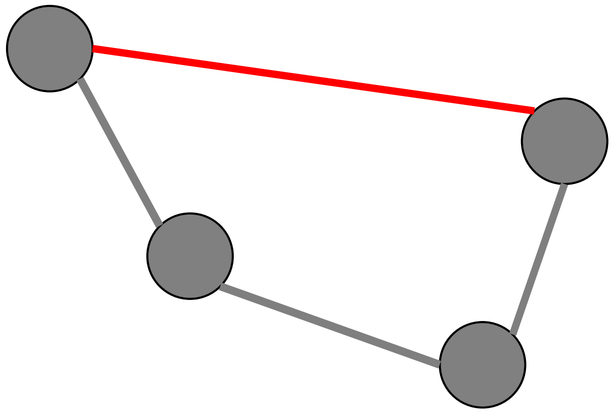
Generic MST Algorithm

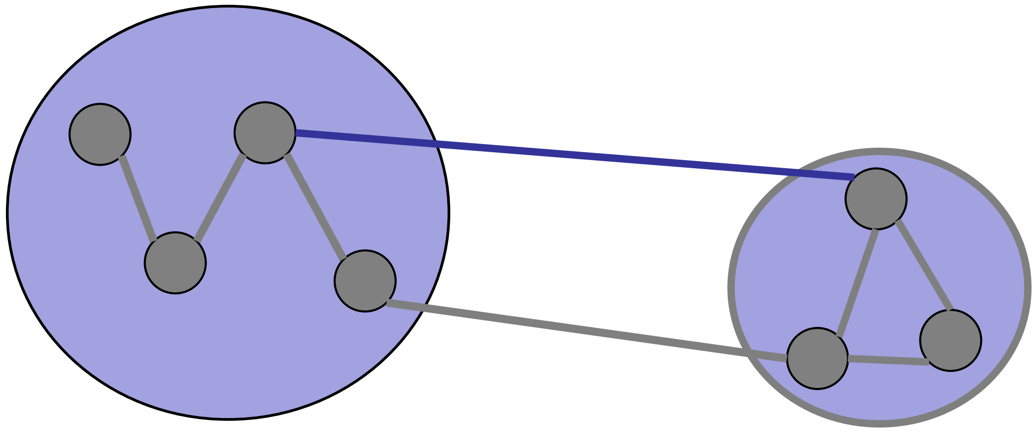
Greedy Algorithm:

Repeat:

> **Apply red rule or blue rule to an arbitrary edge.**

until all edges are either **blue** or **red**.





Candidate MST Algorithm

Divide-and-Conquer:

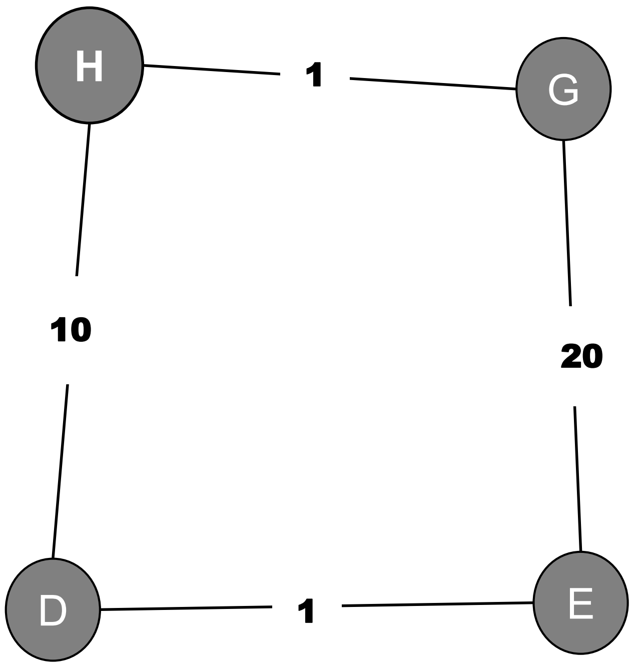
1. If the number of vertices is 1, then return.
2. Divide the nodes into two sets.
3. Recursively calculate the MST of each set.
4. Find the lightest edge the connects the two sets and add it to the MST.
5. Return.

The problem with this algorithm is?

1. Nothing. It efficiently implements the red-blue strategy.
2. It is too expensive to implement because finding the lightest edge is hard.
3. It is too expensive to implement because partitioning the nodes is expensive.
4. [✓ It returns the wrong answer.]

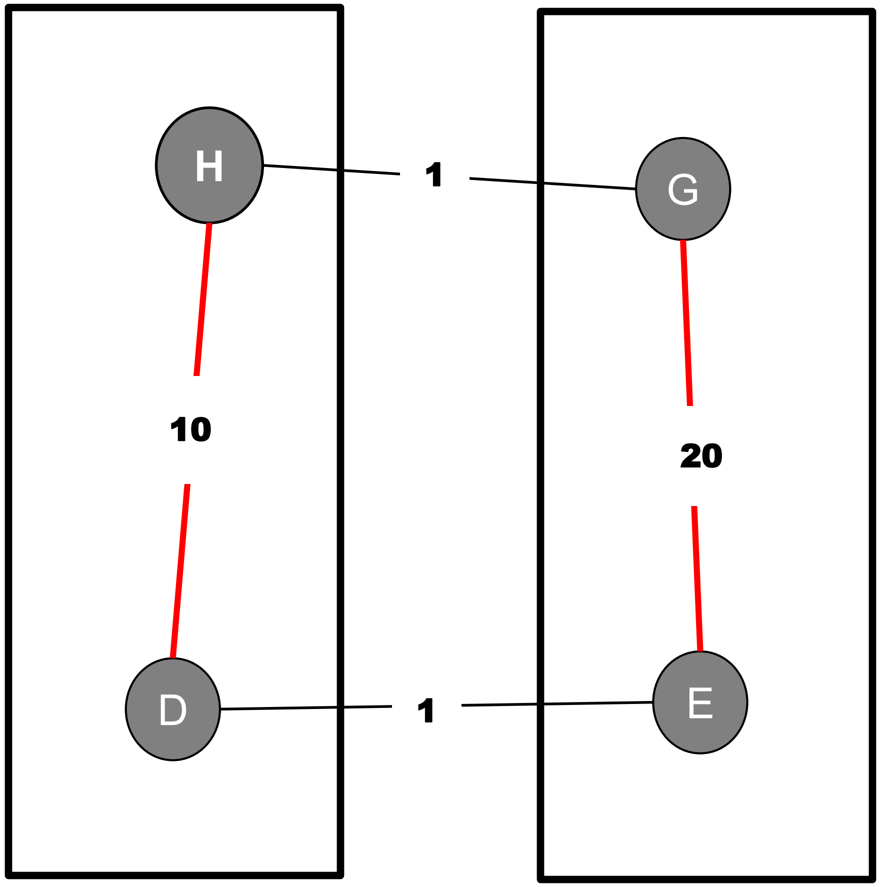
Candidate MST Algorithm

Example:



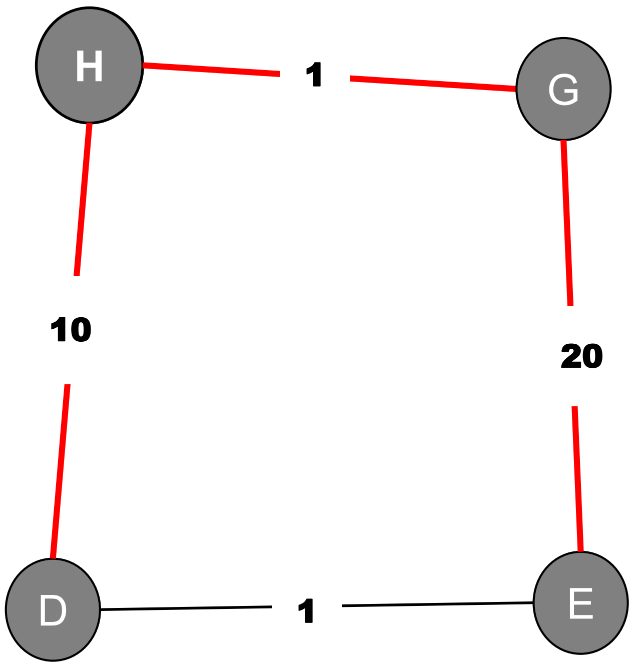
Candidate MST Algorithm

Example: Divide-and-Conquer



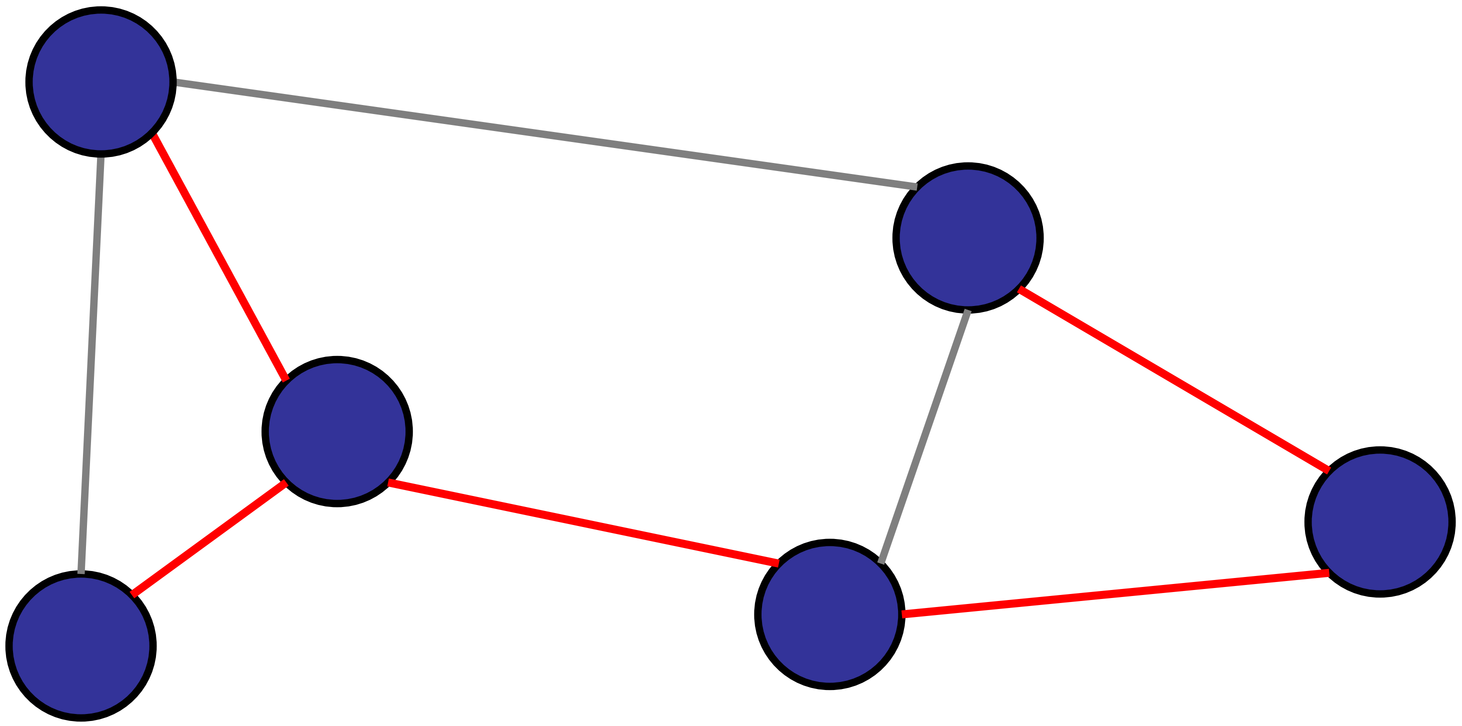
Candidate MST Algorithm

Example: Divide-and-Conquer



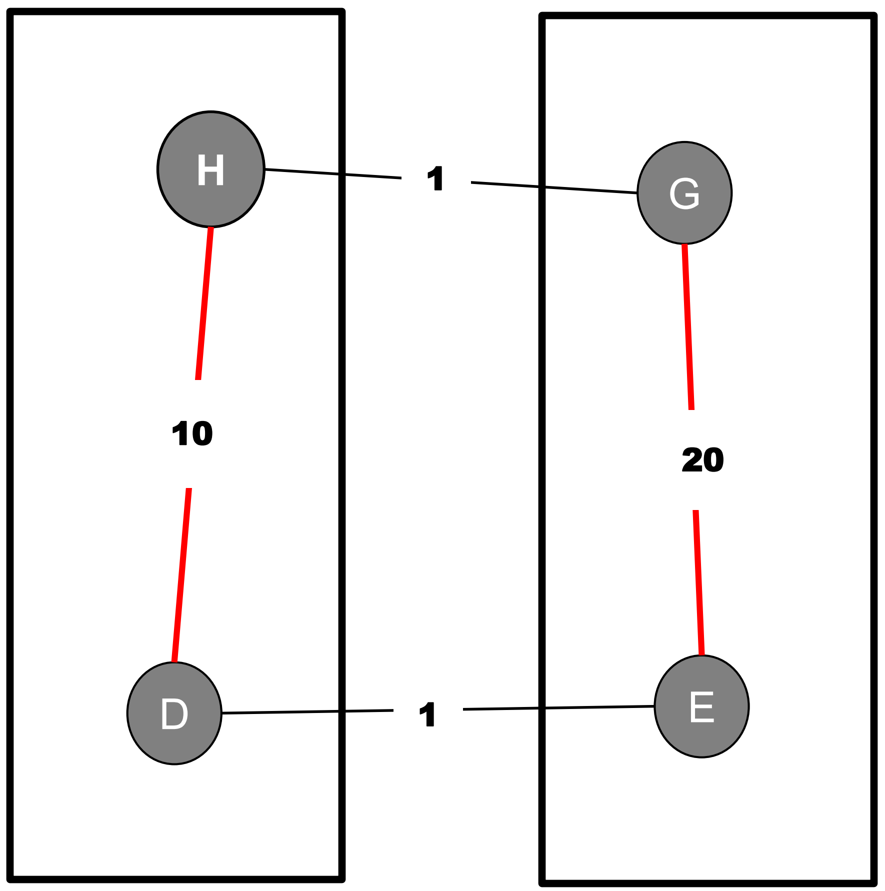
Properties of MST

Property 2: **If you cut an MST**, the two pieces are both MSTs.



Candidate MST Algorithm

Example: Divide-and-Conquer



When we did this cut, we were not cutting our MST.

BAD MST Algorithm

Divide-and-Conquer:

1. If the number of vertices is 1, then return.
2. Divide the nodes into two sets.
3. Recursively calculate the MST of each set.
4. Find the lightest edge the connects the two sets and add it to the MST.
5. Return.

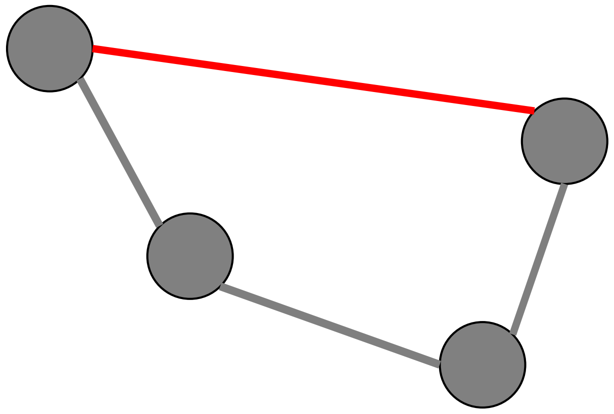
Generic MST Algorithm

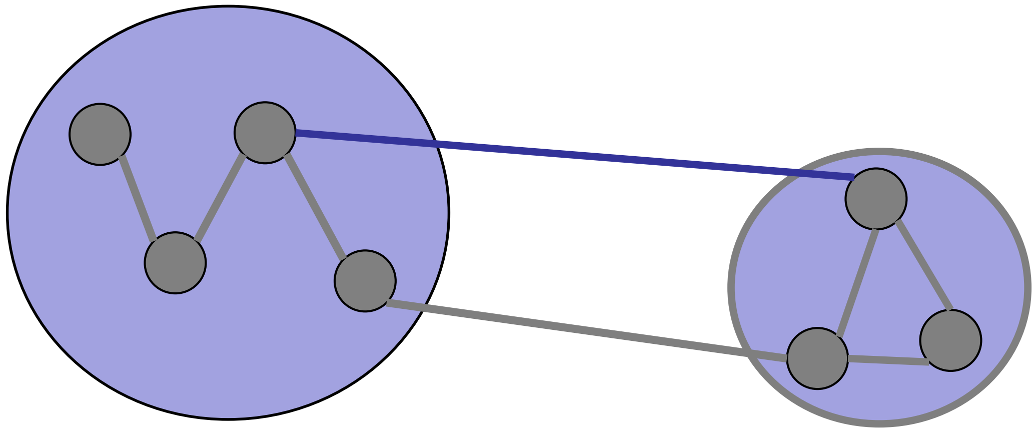
Greedy Algorithm:

Repeat:

Apply red rule or blue rule to an arbitrary edge.

until no more edges can be colored.





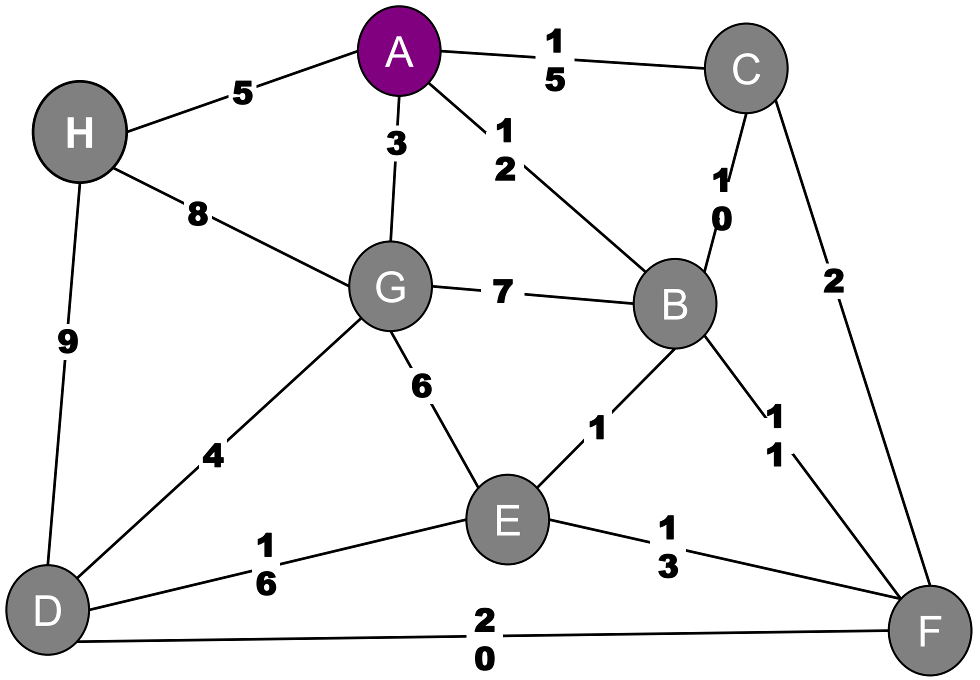
Roadmap

Minimum Spanning Trees

- The MST Problem
- Basic Properties of an MST
- Generic MST Algorithm
- **Prim’s Algorithm**
- Kruskal’s Algorithm
- Boruvka’s Algorithm
- Variations

Prim’s Algorithm

Prim’s Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

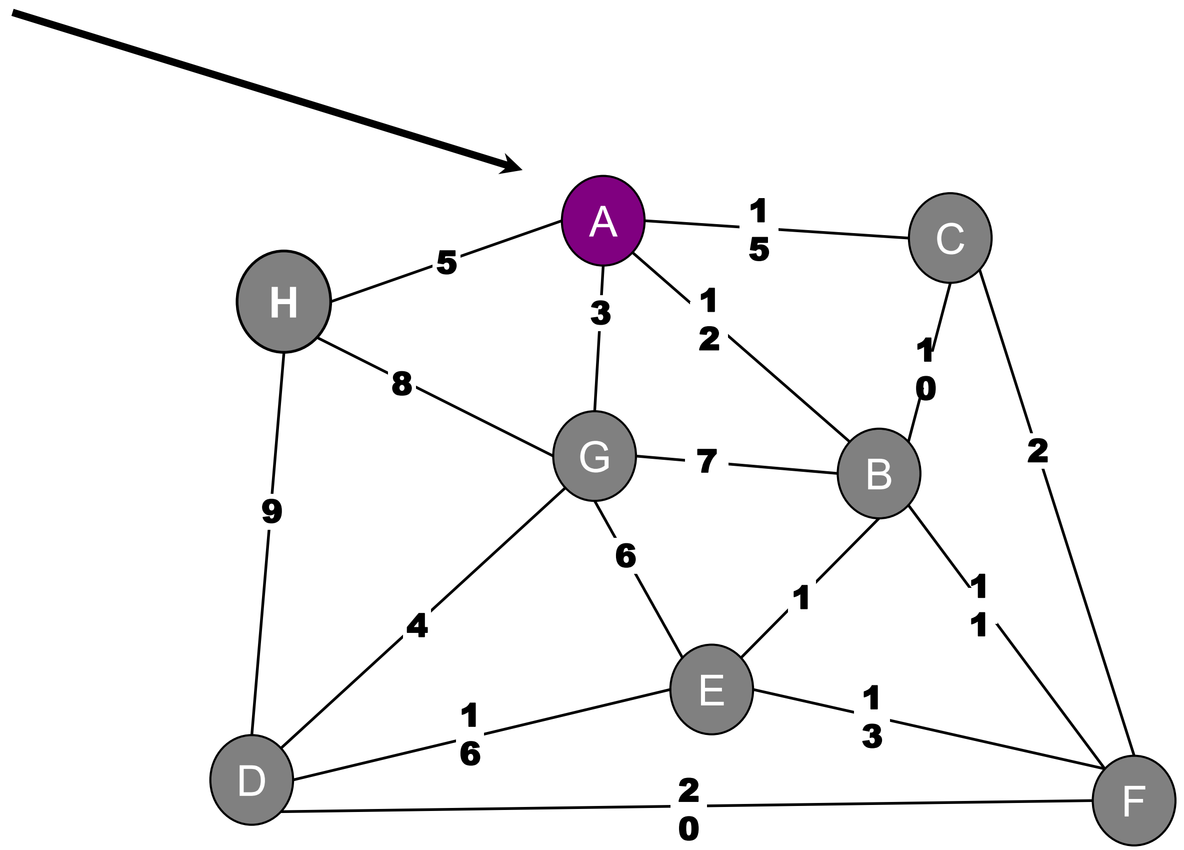


Prim's Algorithm

Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}

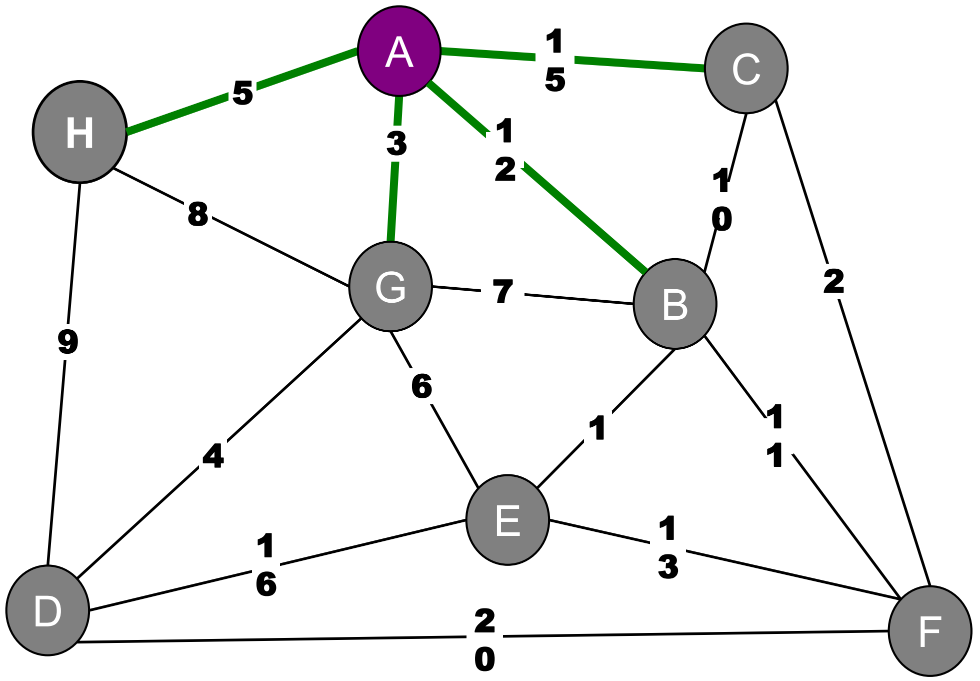


Prim's Algorithm

Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Identify cut: {S, V–S}

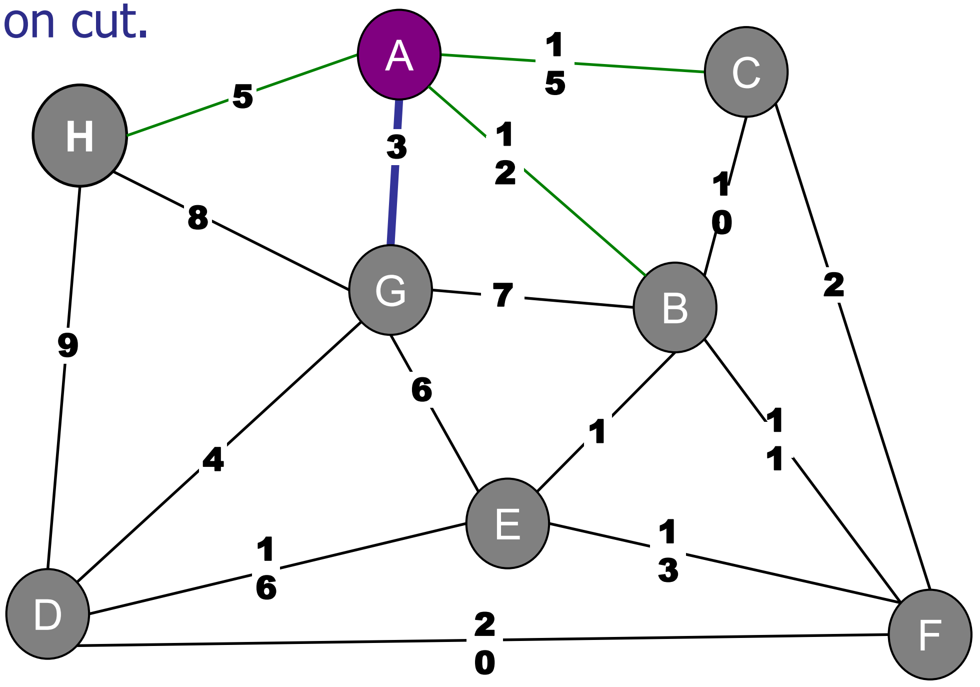


Prim’s Algorithm

Prim’s Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Identify cut: {S, V–S}
- Find minimum weight edge on cut.

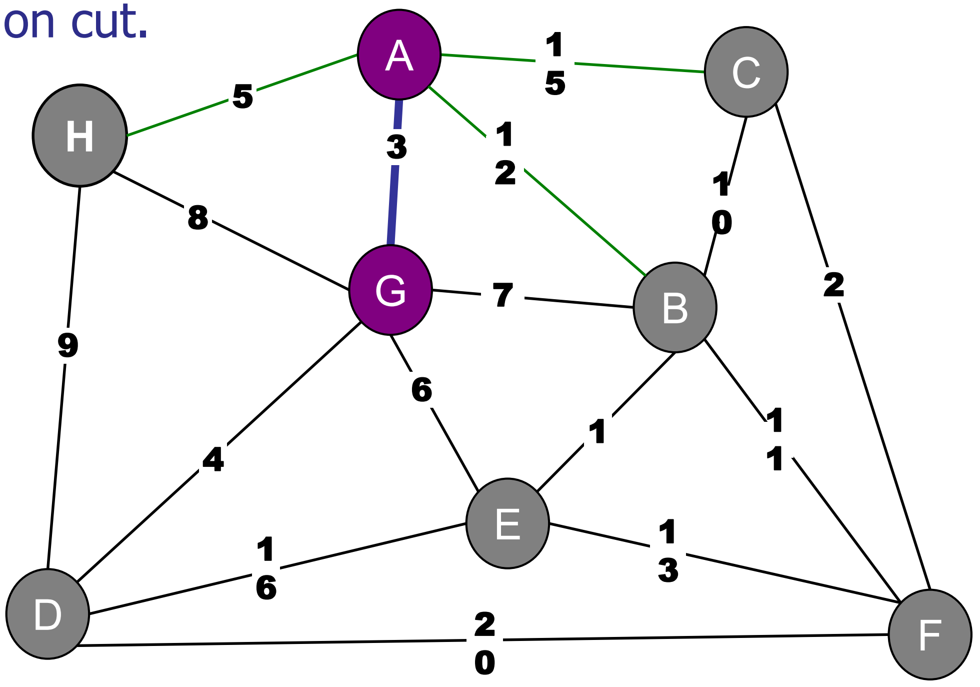


Prim's Algorithm

Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Identify cut: {S, V–S}
- Find minimum weight edge on cut.
- Add new node to S.

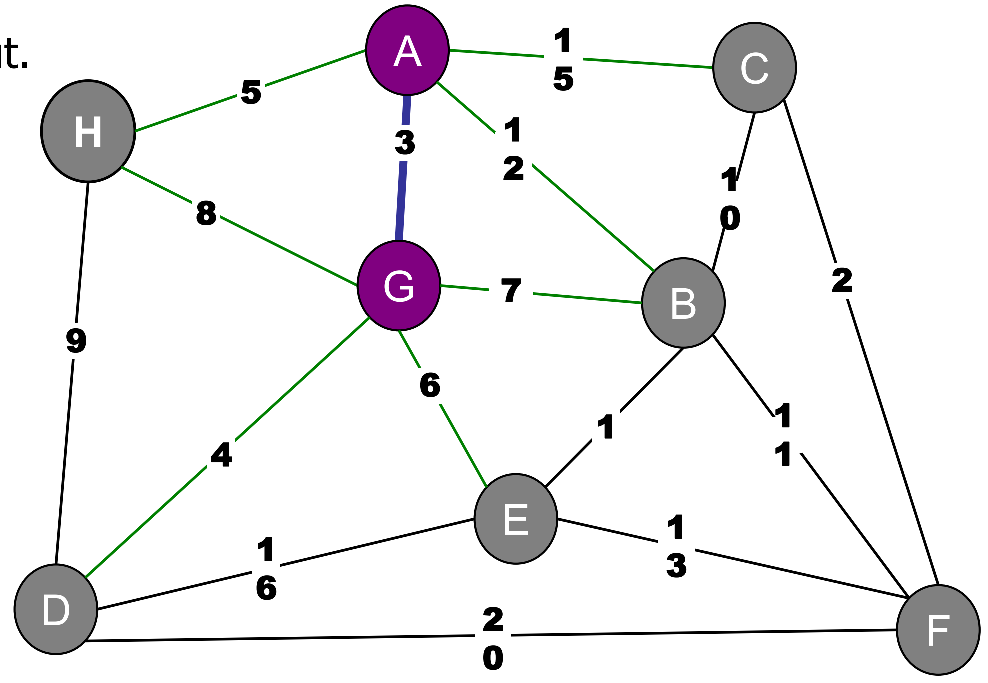


Prim's Algorithm

Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Repeat:
  - Identify cut: {S, V–S}
  - Find minimum weight edge on cut.
  - Add new node to S.

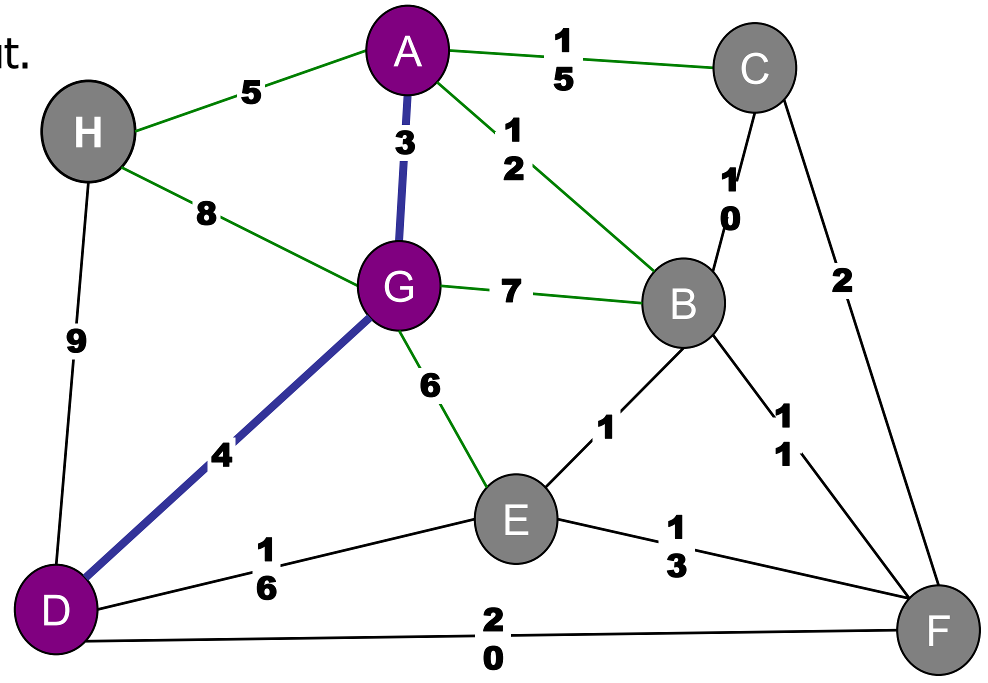


Prim's Algorithm

Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Repeat:
    - Identify cut: {S, V–S}
    - Find minimum weight edge on cut.
    - Add new node to S.

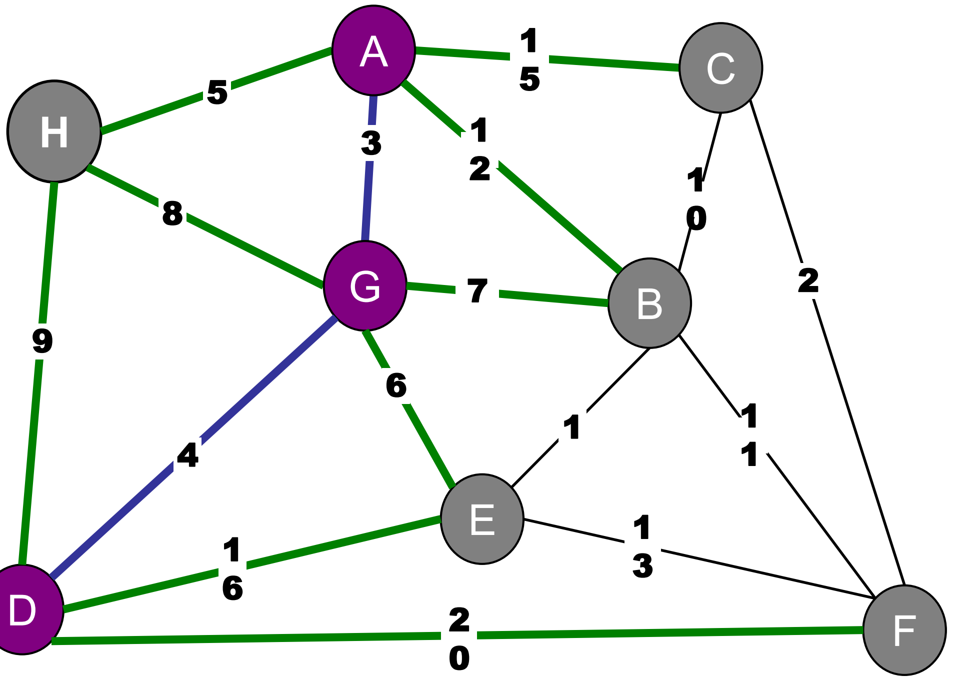


Prim's Algorithm

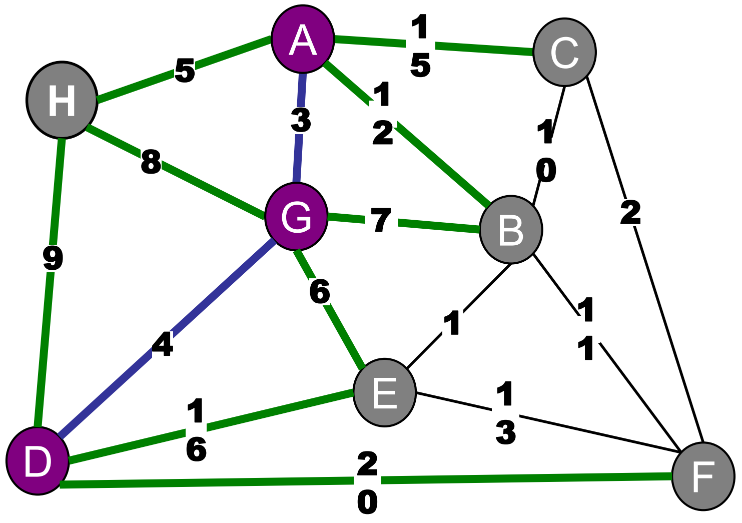
Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

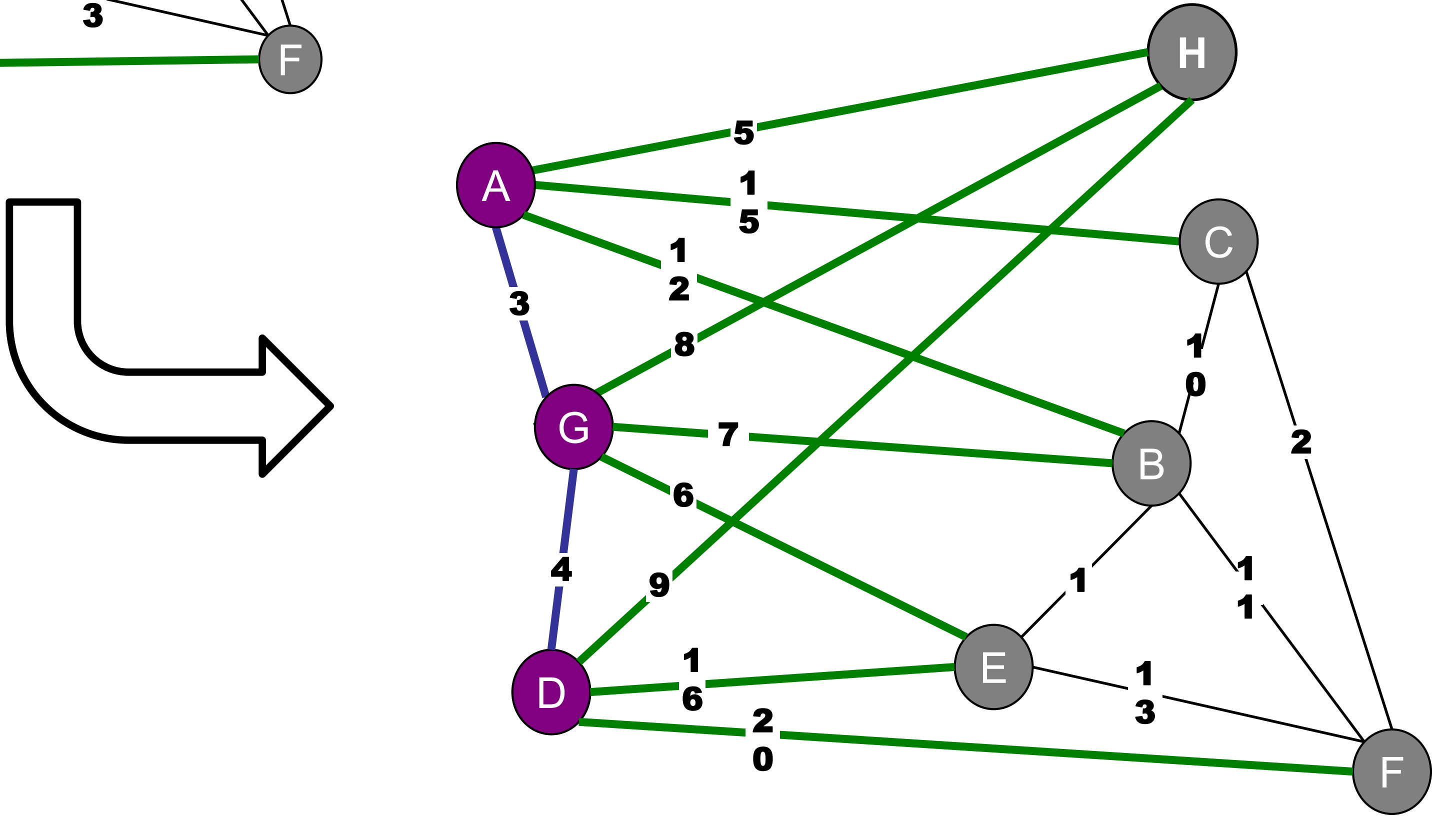
Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Repeat:
  - Identify cut: {S, V–S}
  - Find minimum weight edge on cut.
  - Add new node to S.



Prim's Algorithm





How do we find the lightest edge on a cut?

1. ✓ Priority Queue
2. Union-Find
3. Max-flow / Min-cut
4. BFS
5. DFS

Prim's Algorithm

1. Set min-pq to contain only source node

2. while min-pq is not empty
    a. Take next node out of min-pq

    b. If node has not been added before, include the edge used into the MST. Otherwise, skip!

    c. Go through all neighbours **n** of node

    d. If the edge has weight **w**, insert **n** into min-pq with priority **w**

Prim's Algorithm

1. Set min-pq to contain only source node

2. while min-pq is not empty
    a. Take next node out of min-pq

    b. If node has not been added before, include the edge used into the MST. Otherwise, skip!

    c. Go through all neighbours **n** of node

    d. If the edge has weight **w**, insert **n** into min-pq with priority **w**

What data structure should we use to know if a node has been visited before?

✓ 1. Hashmap
✓ 2. Array
3. Adjacency List
4. Priority Queue
5. Balanced BST

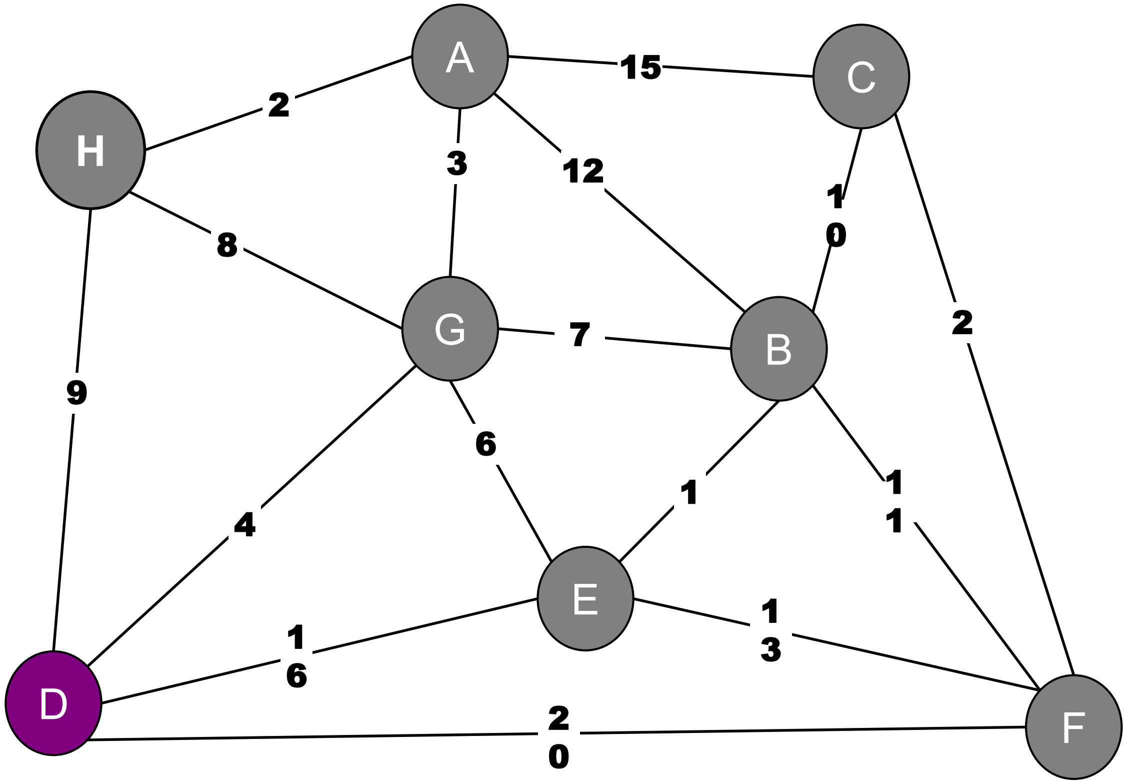
Prim’s Algorithm: Variant

Note: There are a few variants in terms of implementation. For example

Instead of inserting multiple copies of the node, start the pq with all nodes with priority infinity, and source node with priority 0.

When we process a node’s neighbours, decrease key the neighbour if the new edge weight is smaller than its current priority.

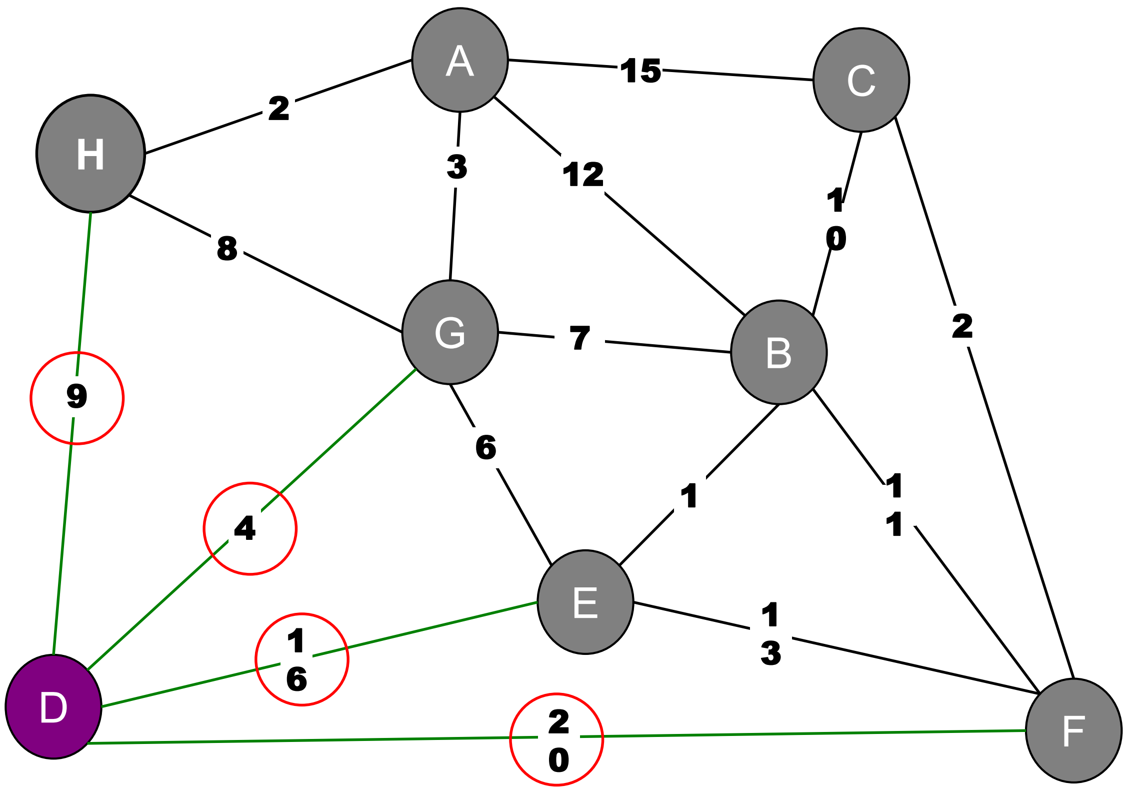
Prim's Example



Vertex	Weight
D	0

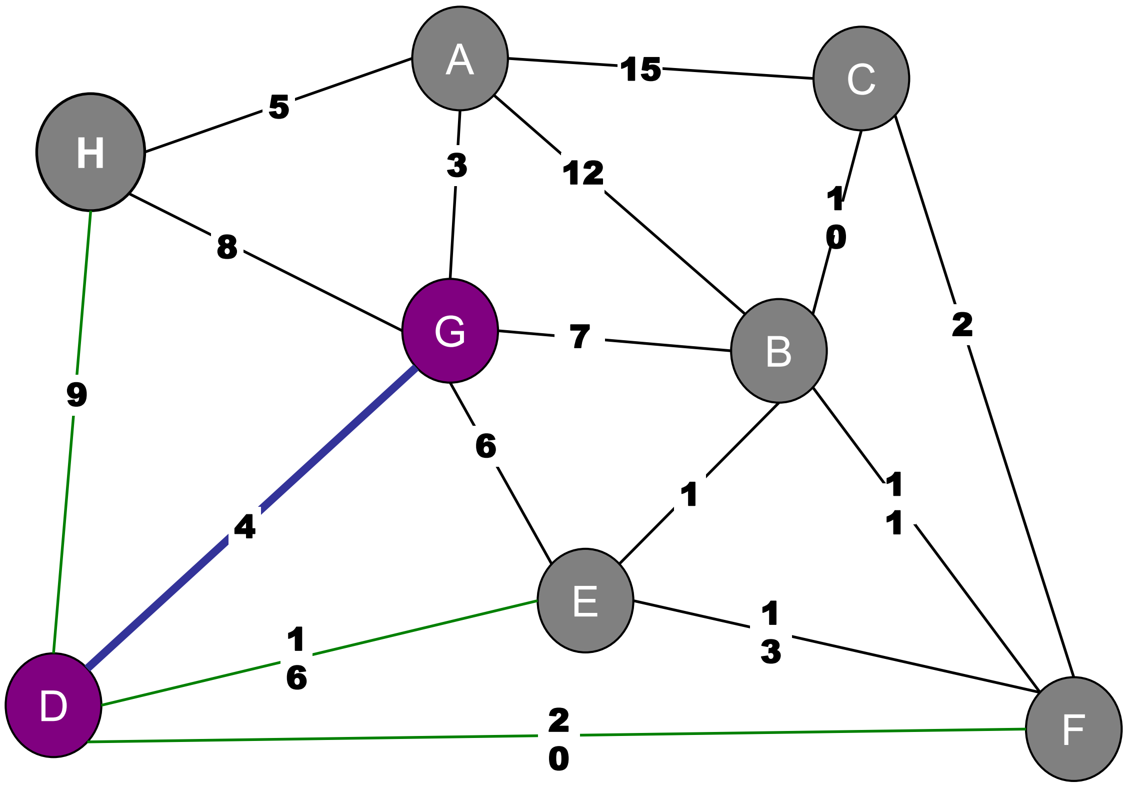
Not drawing infinite weight nodes in PQ.

Prim’s Example



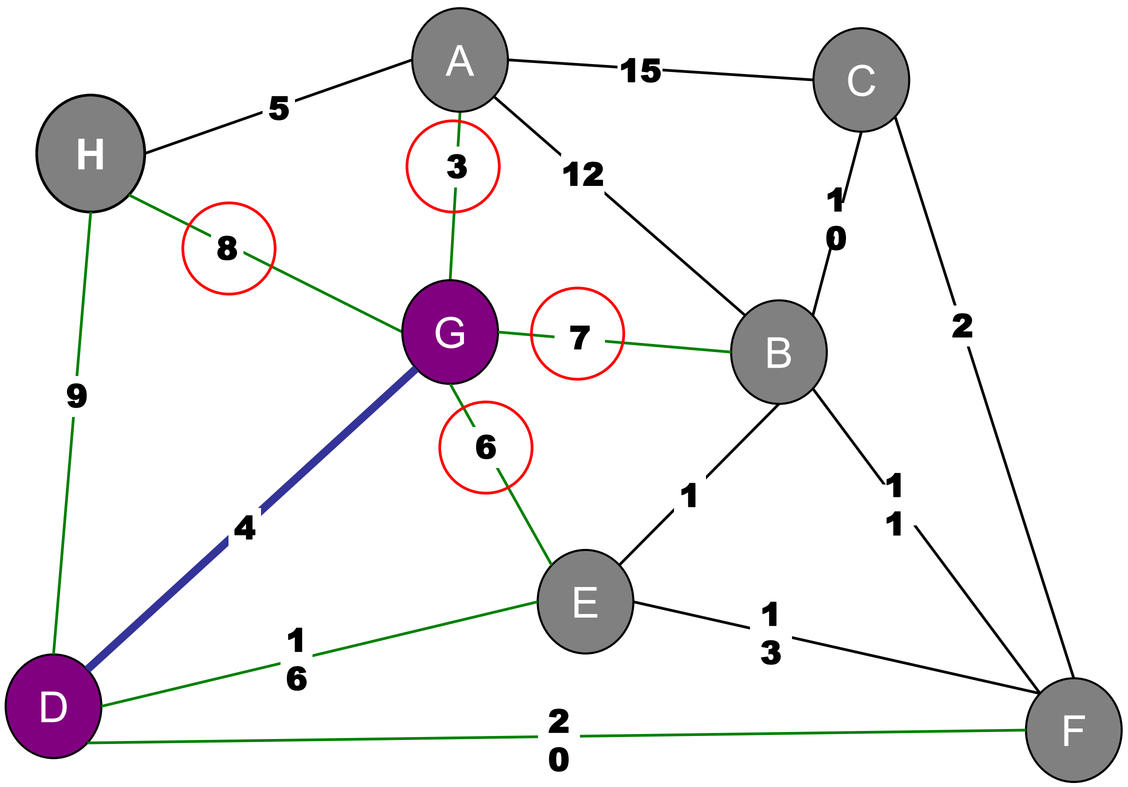
Vertex	Weight
G	4
H	9
E	16
F	20

Prim’s Example



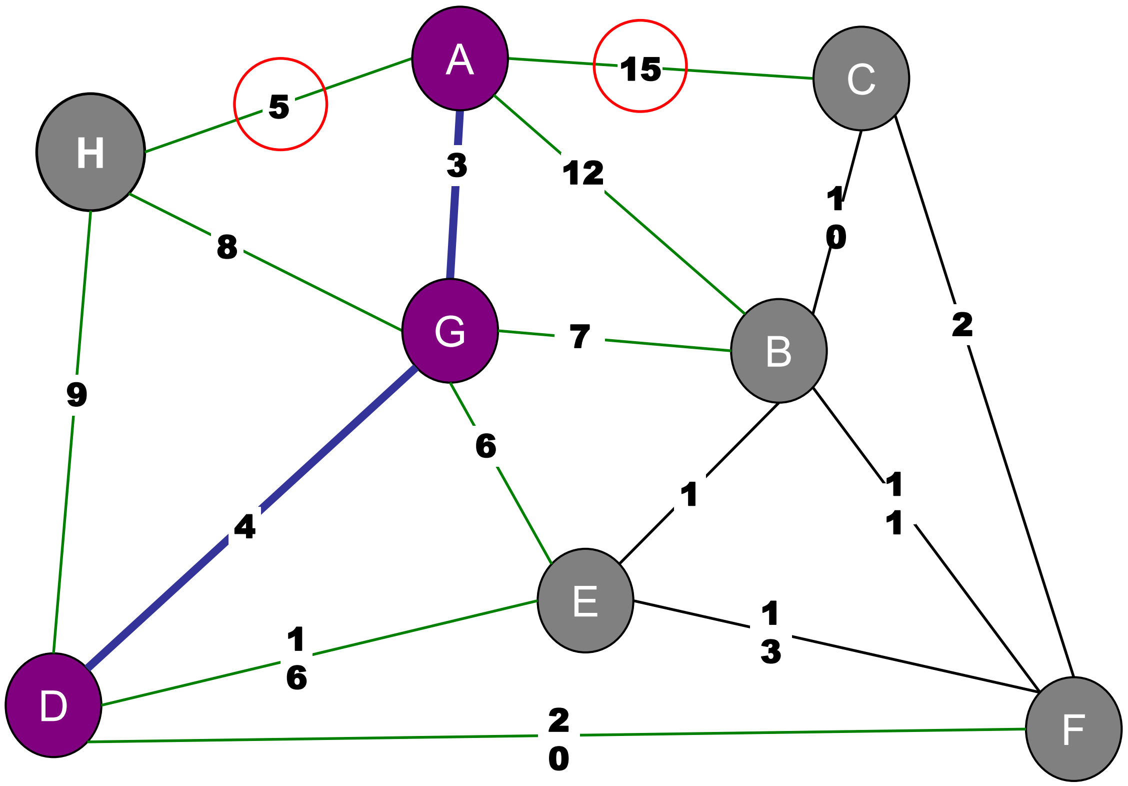
Vertex	Weight
H	9
E	16
F	20

Prim’s Example



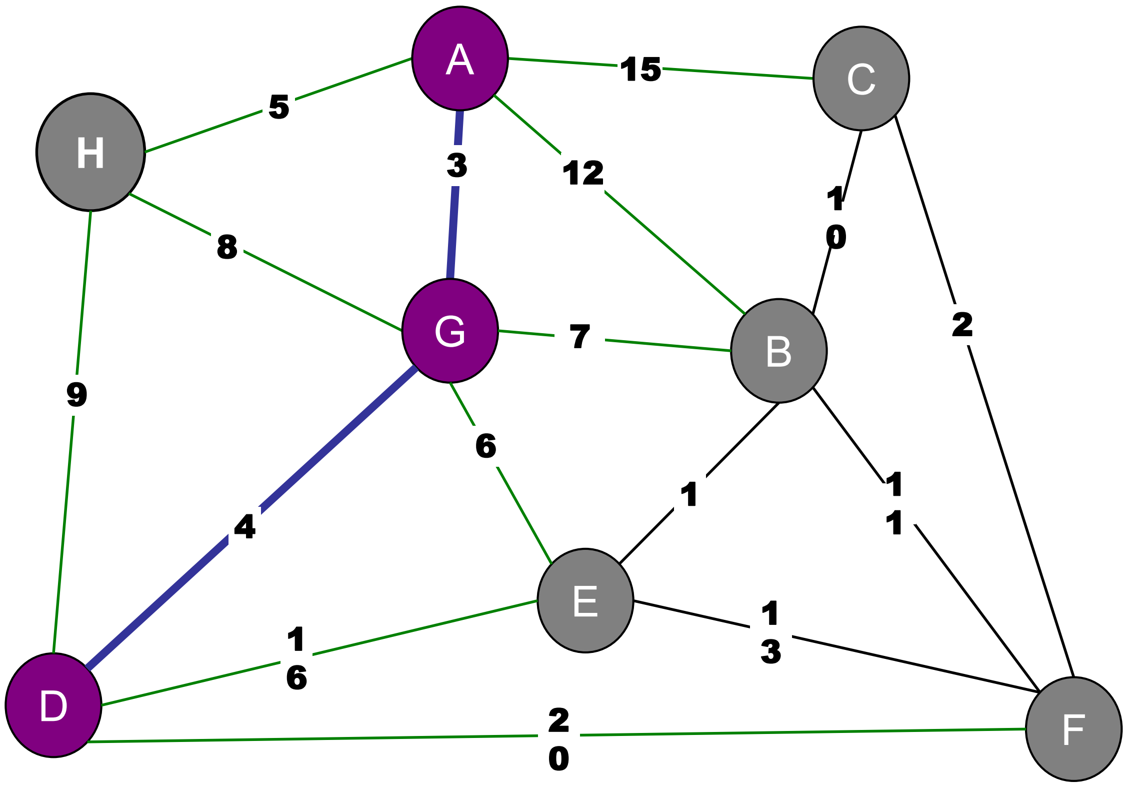
Vertex	Weight
A	3
E	16->6
B	7
H	9->8
F	20

Prim’s Example



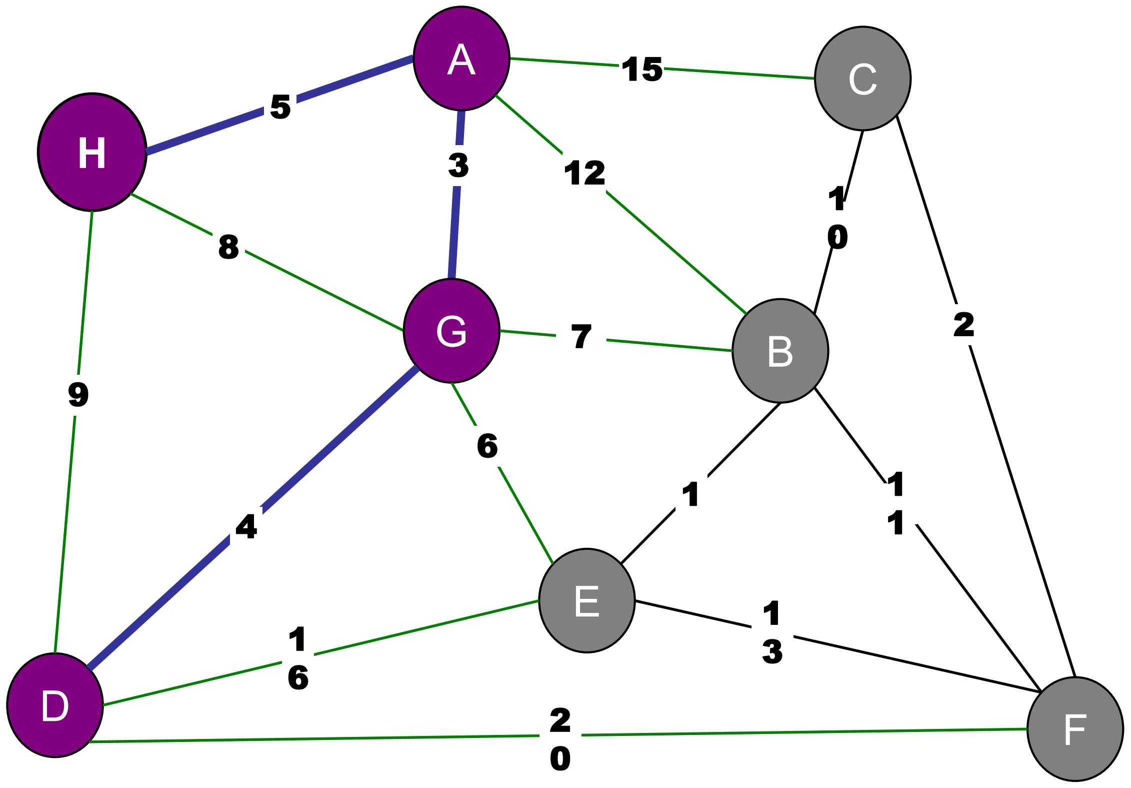
Vertex	Weight
H	**8->5**
E	6
B	7
C	**15**
F	20

Prim’s Example



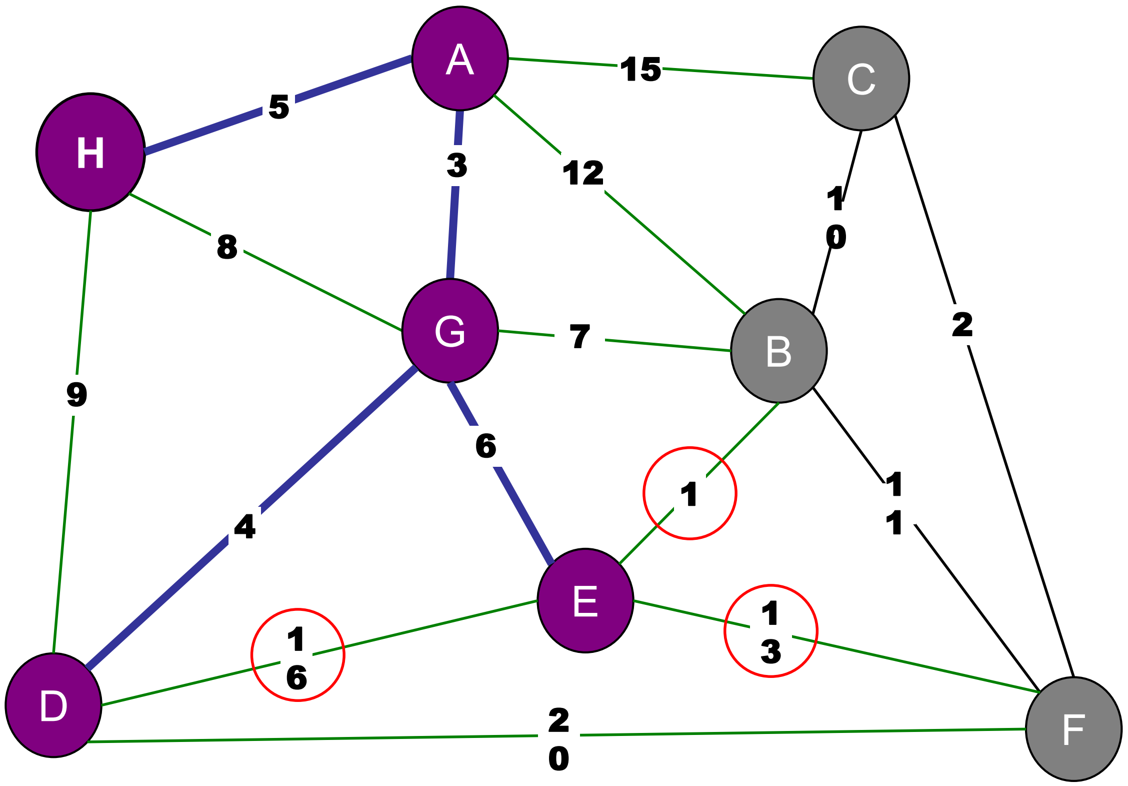
Vertex	Weight
H	5
E	6
B	7
C	15
F	20

Prim’s Example



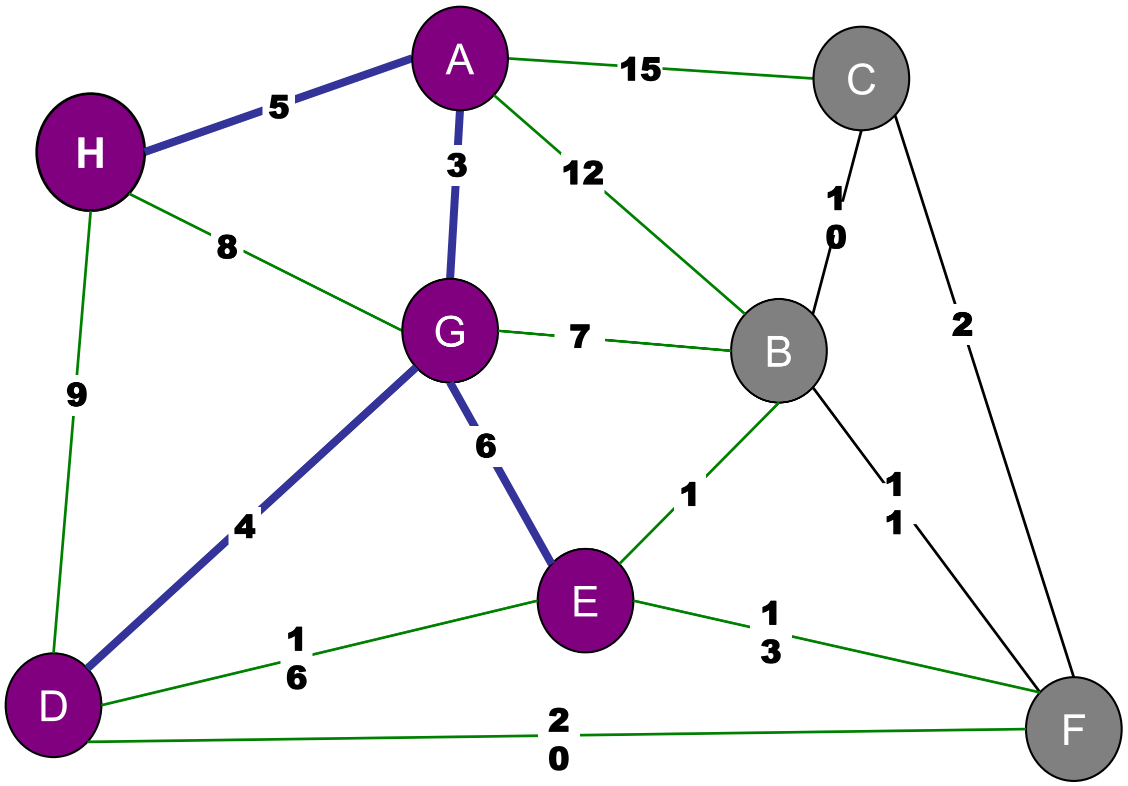
Vertex	Weight
E	6
B	7
C	15
F	20

Prim’s Example



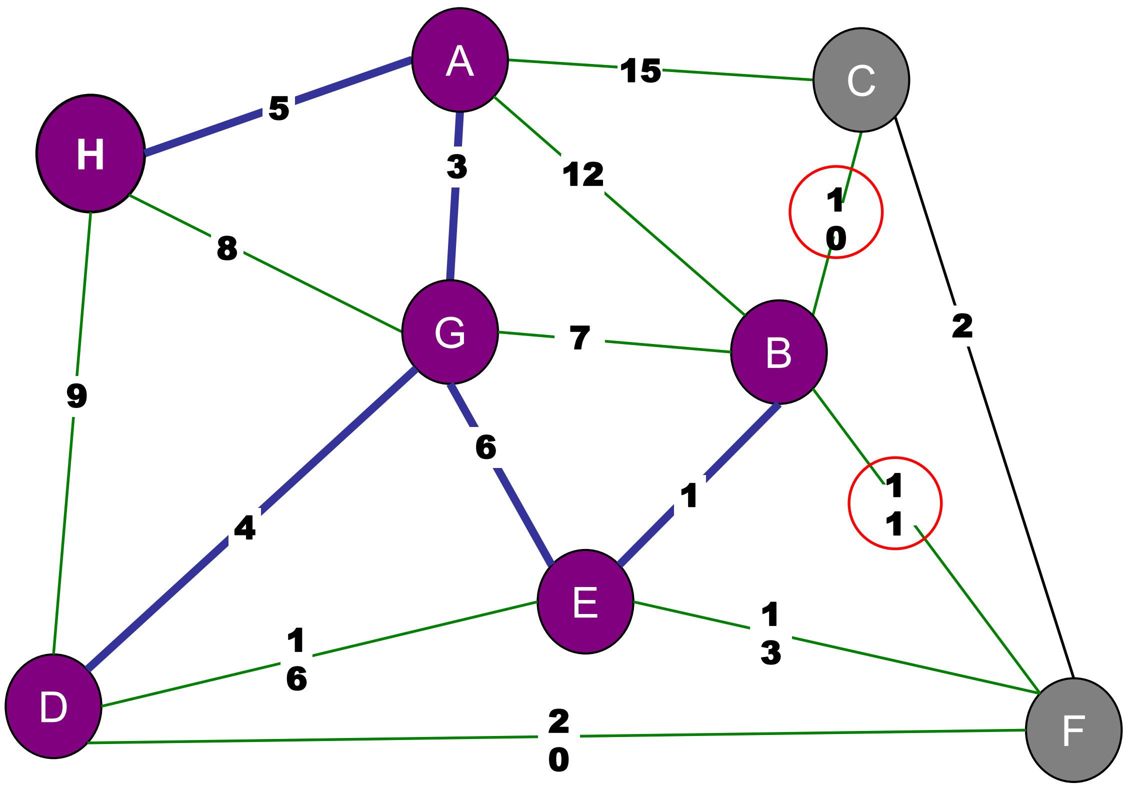
Vertex	Weight
B	**7->1**
C	15
F	**20->13**

Prim’s Example



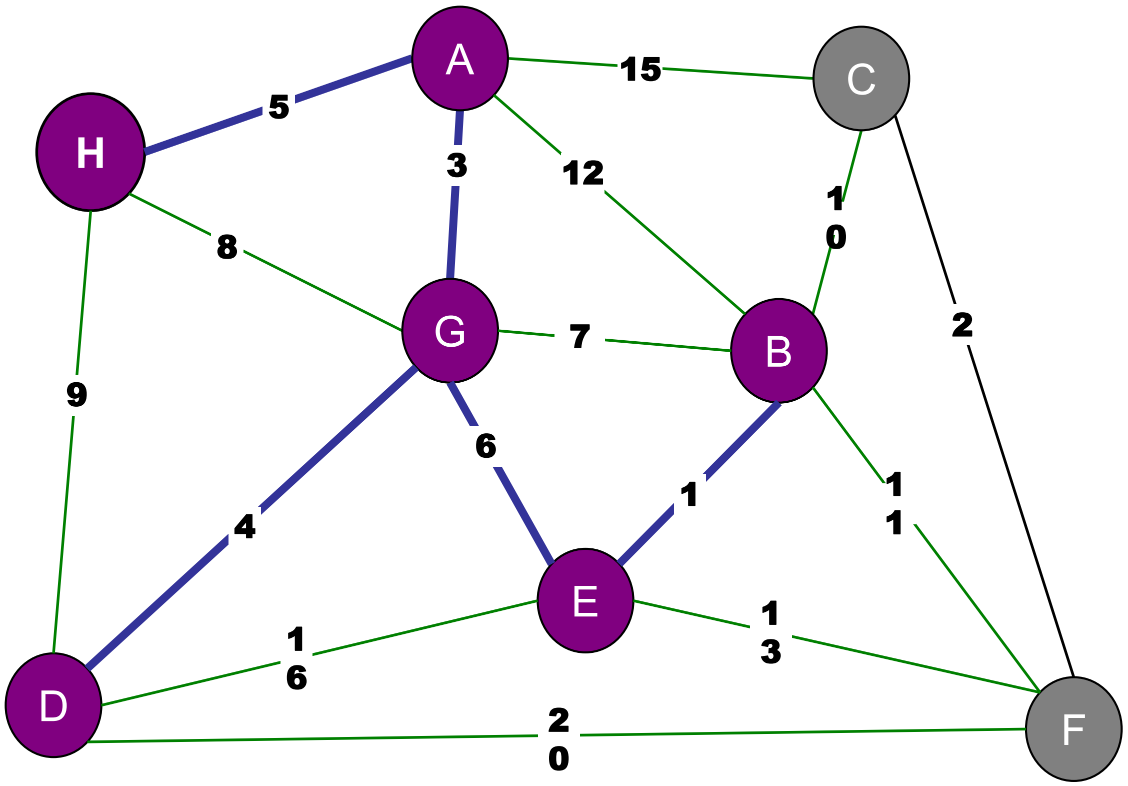
Vertex	Weight
B	1
C	15
F	13

Prim’s Example



Vertex	Weight
C	15->10
F	13->11

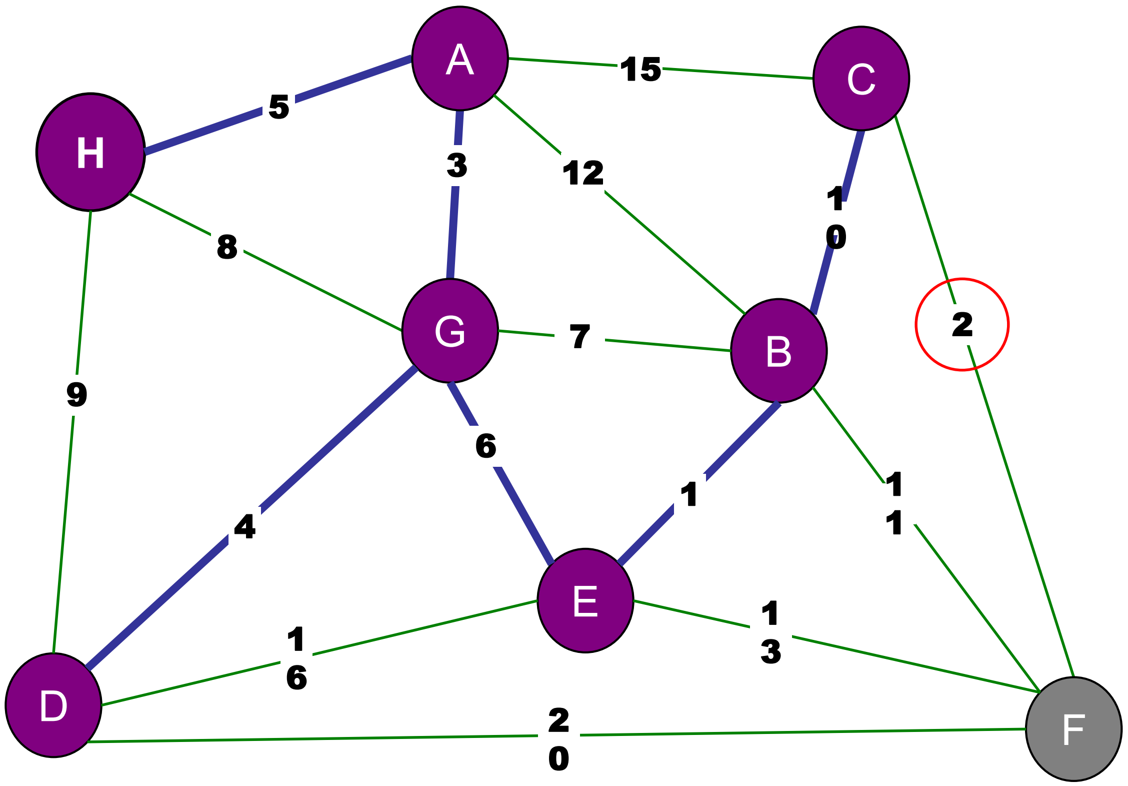
Prim's Example



Vertex	Weight
C	10
F	11

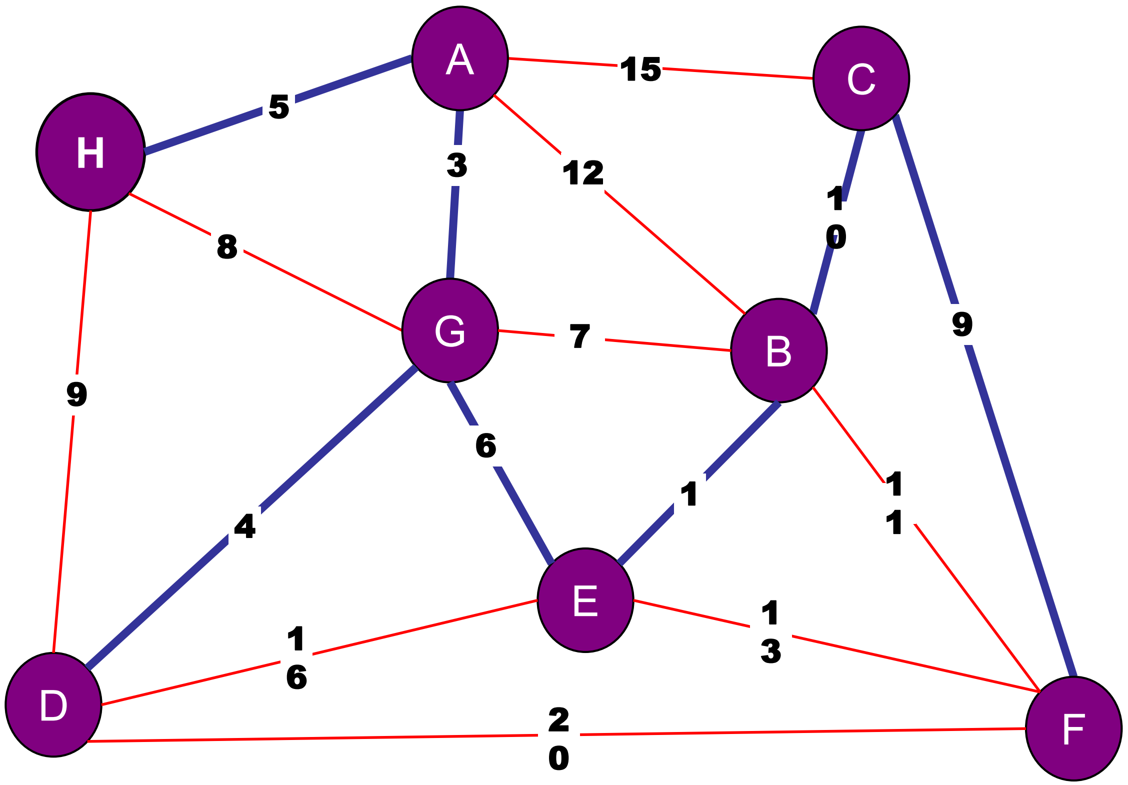
Prim's Example

Vertex	Weight
F	11->2



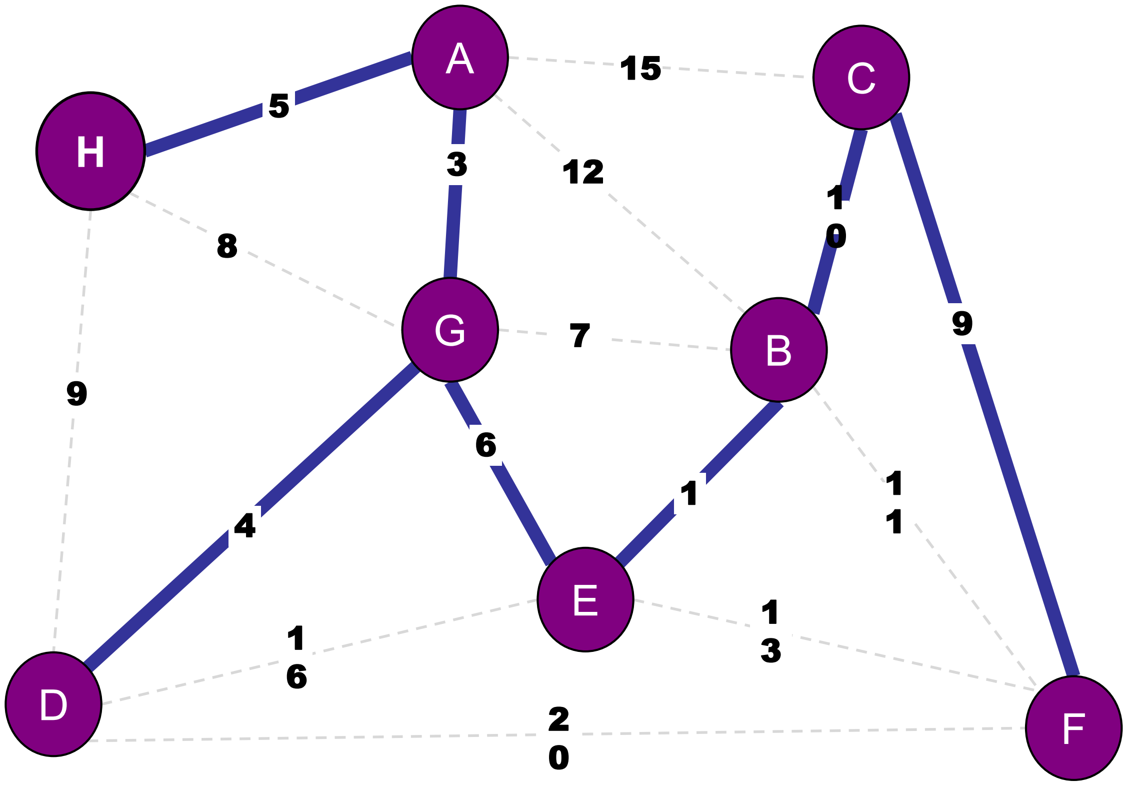
Prim's Example

Vertex	Weight



Prim's Example

Vertex	Weight



Prim’s Algorithm

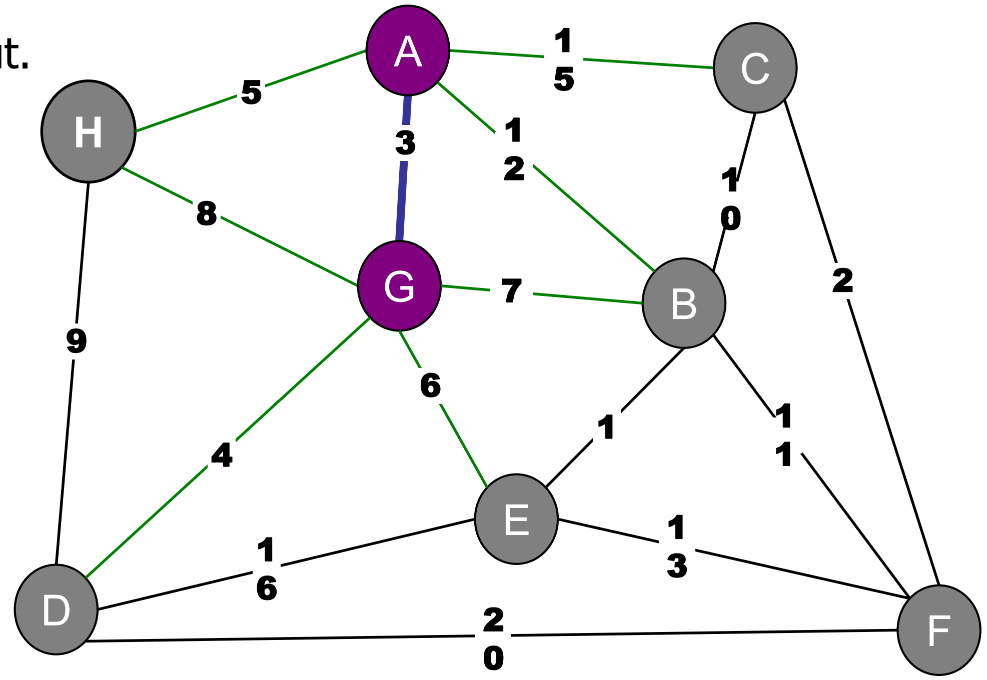
Prim’s Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Repeat:
  - Identify cut: {S, V–S}
  - Find minimum weight edge on cut.
  - Add new node to S.

Proof:

- Each added edge is the lightest on some cut.
- Hence each edge is in the MST.



What is the running time of Prim’s Algorithm, using an AVL tree for PQ?

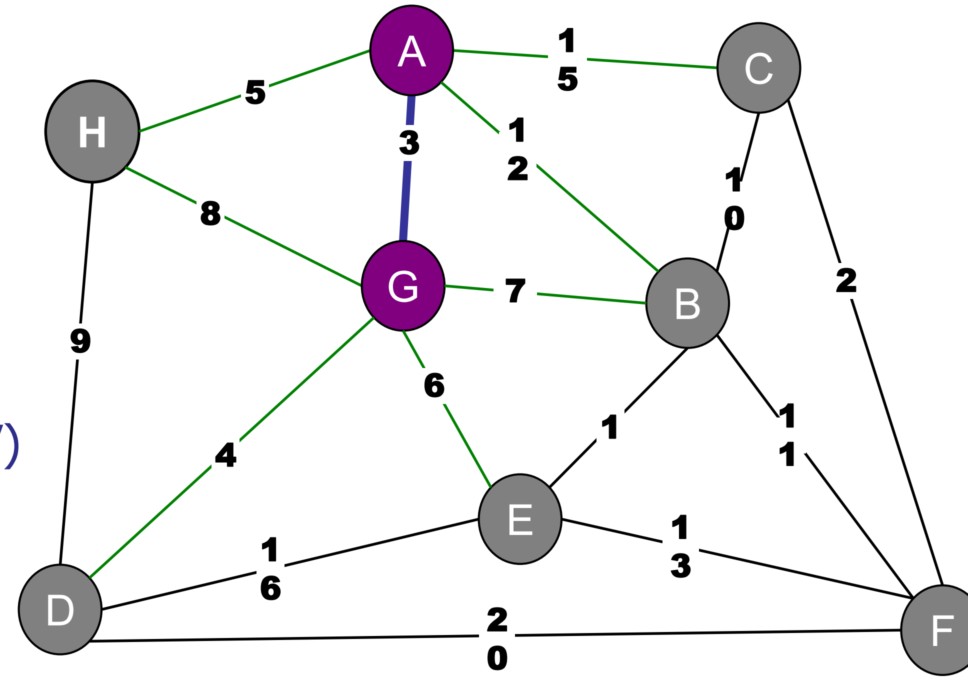
1. O(V)
2. O(E)
3. [✓ O(E log V)]
4. O(V log E)
5. O(EV)

Prim's Algorithm

Prim's Algorithm.(Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: S = {A}
- Repeat:
  - Identify cut: {S, V–S}
  - Find minimum weight edge on cut.
  - Add new node to S.



Analysis:

- Each vertex added/removed once from the priority queue: O(**V log V**)
- Each edge => one decreaseKey: O(**E log V**).

Two Algorithms

Prim's Algorithm.

Basic idea:

- Maintain a set of visited nodes.
- Greedily grow the set by adding node connected via the lightest edge.
- Use Priority Queue to order nodes by **edge weight**.

Dijkstra's Algorithm.

Basic idea:

- Maintain a set of visited nodes.
- Greedily grow the set by adding neighboring node that is closest to the source.
- Use Priority Queue to order nodes by **distance from source**.

Roadmap

Minimum Spanning Trees

- The MST Problem
- Basic Properties of an MST
- Generic MST Algorithm
- Prim’s Algorithm
- **Kruskal’s Algorithm**
- Boruvka’s Algorithm
- Variations

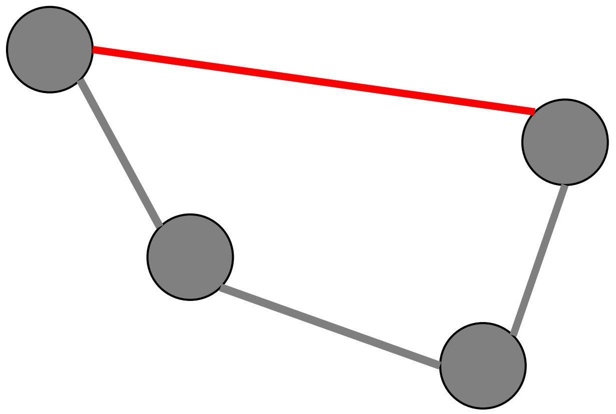
Generic MST Algorithm

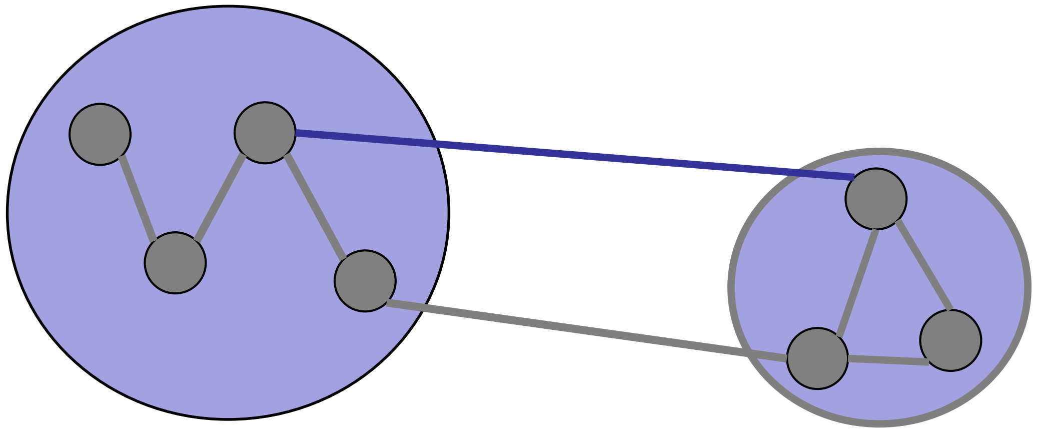
Greedy Algorithm:

Repeat:

Apply red rule or blue rule to an arbitrary edge.

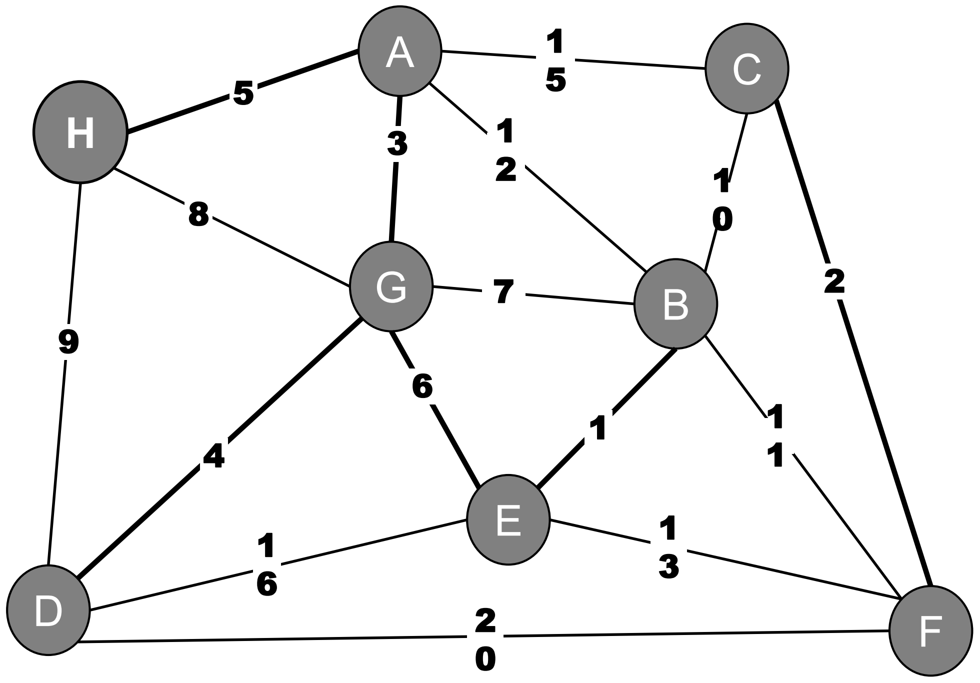
until no more edges can be colored.





Kruskal’s Algorithm

Kruskal’s Algorithm. (Kruskal 1956)

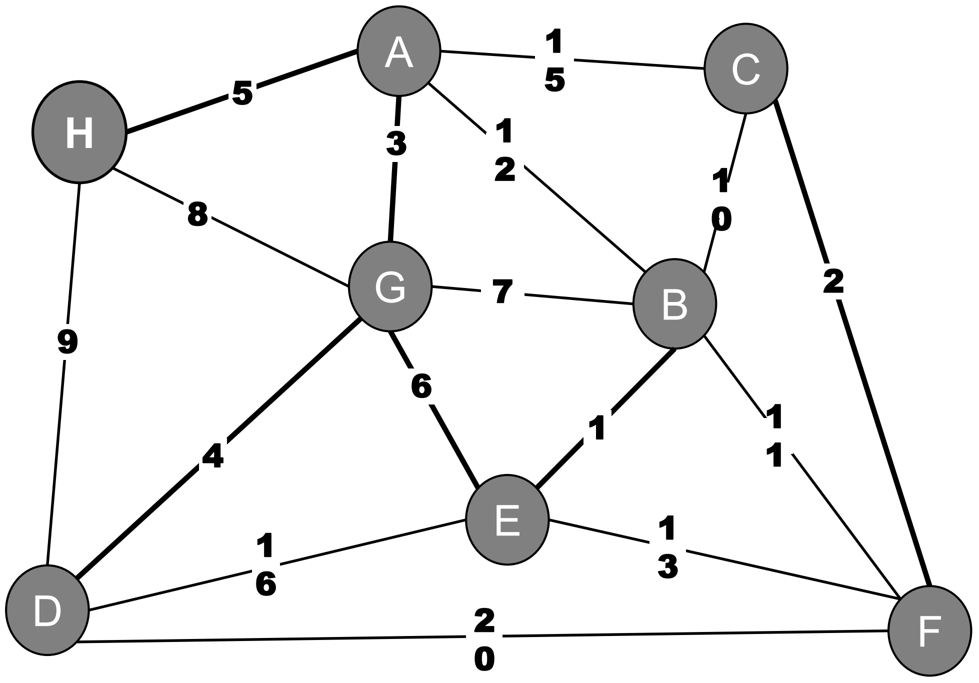


Kruskal’s Algorithm

Kruskal’s Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight from smallest to biggest.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.



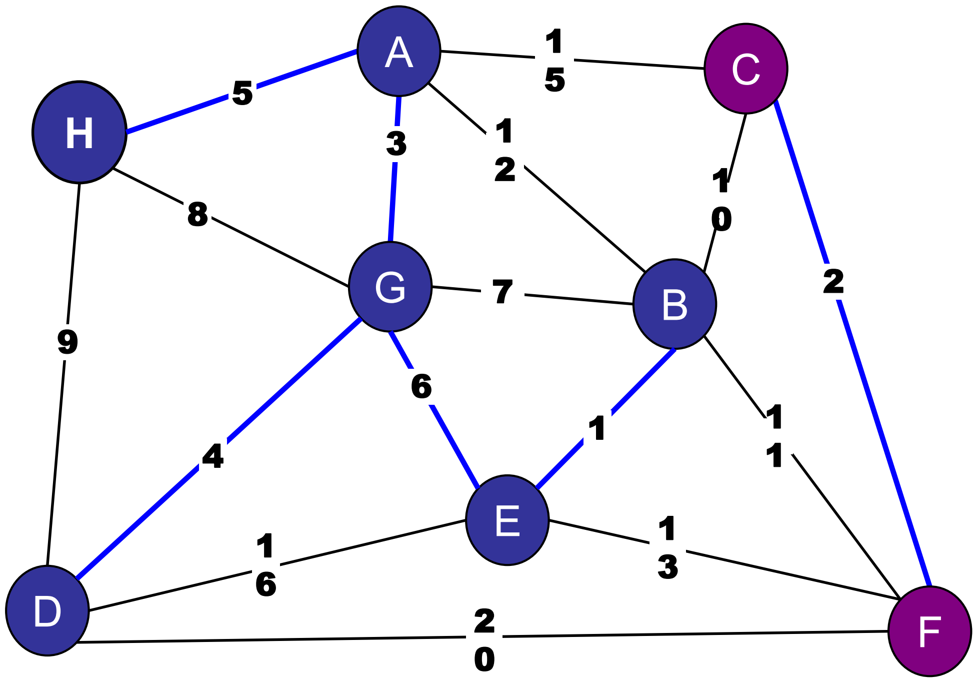
Kruskal's Algorithm

Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight from smallest to biggest.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.

Must be the heaviest edge on the cycle!

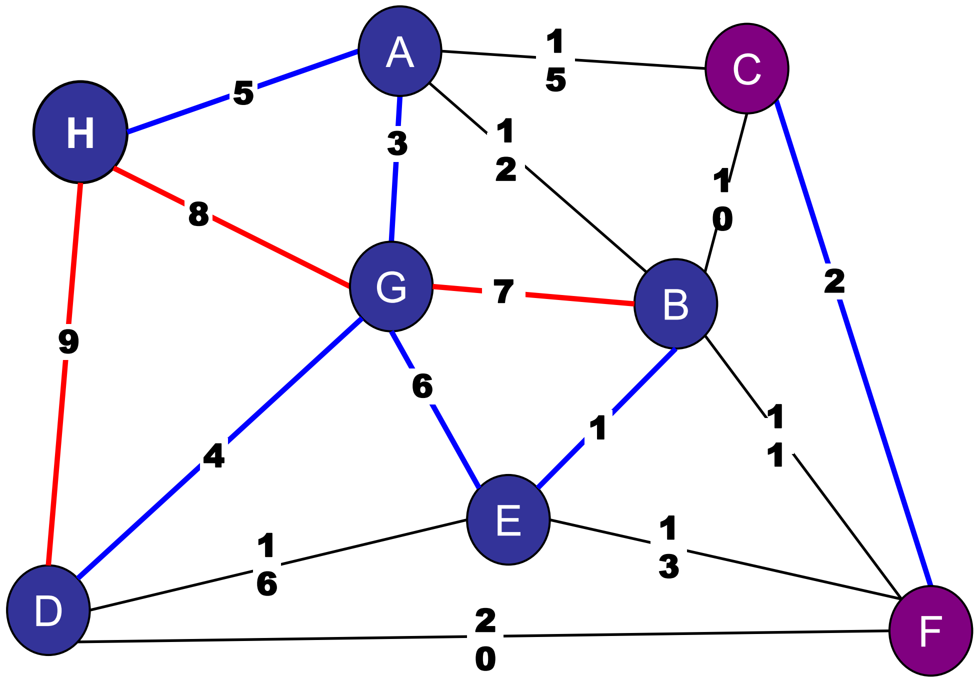


Kruskal’s Algorithm

Kruskal’s Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight from smallest to biggest.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.



Kruskal’s Algorithm

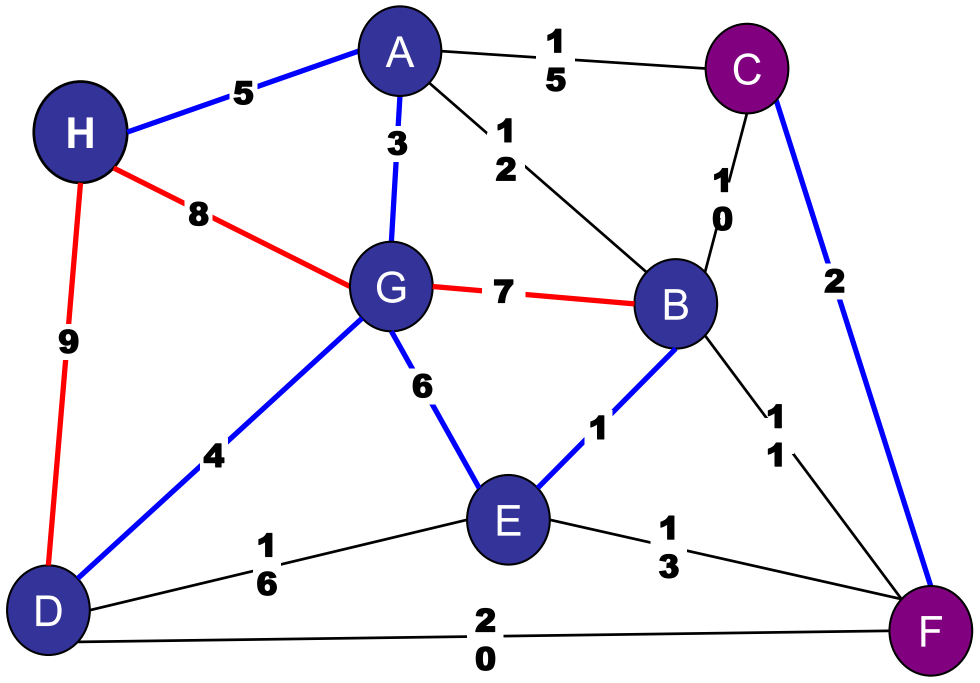
Kruskal’s Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight from smallest to biggest.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.

Data structure:

- Union-Find
- Connect two nodes if they are in the same blue tree.



Kruskal's Algorithm

1. Initialise UFDS so that every node is its own disjoint set.
2. Sort the edges by their weights in increasing order.
3. For every edge **e = (u, v)**:
    a. If nodes **u** and **v** are in the same set, skip!

    b. Include edge **e** in our set of edges to output.

    c. Union the sets containing **u** and **v**.

Kruskal’s Algorithm

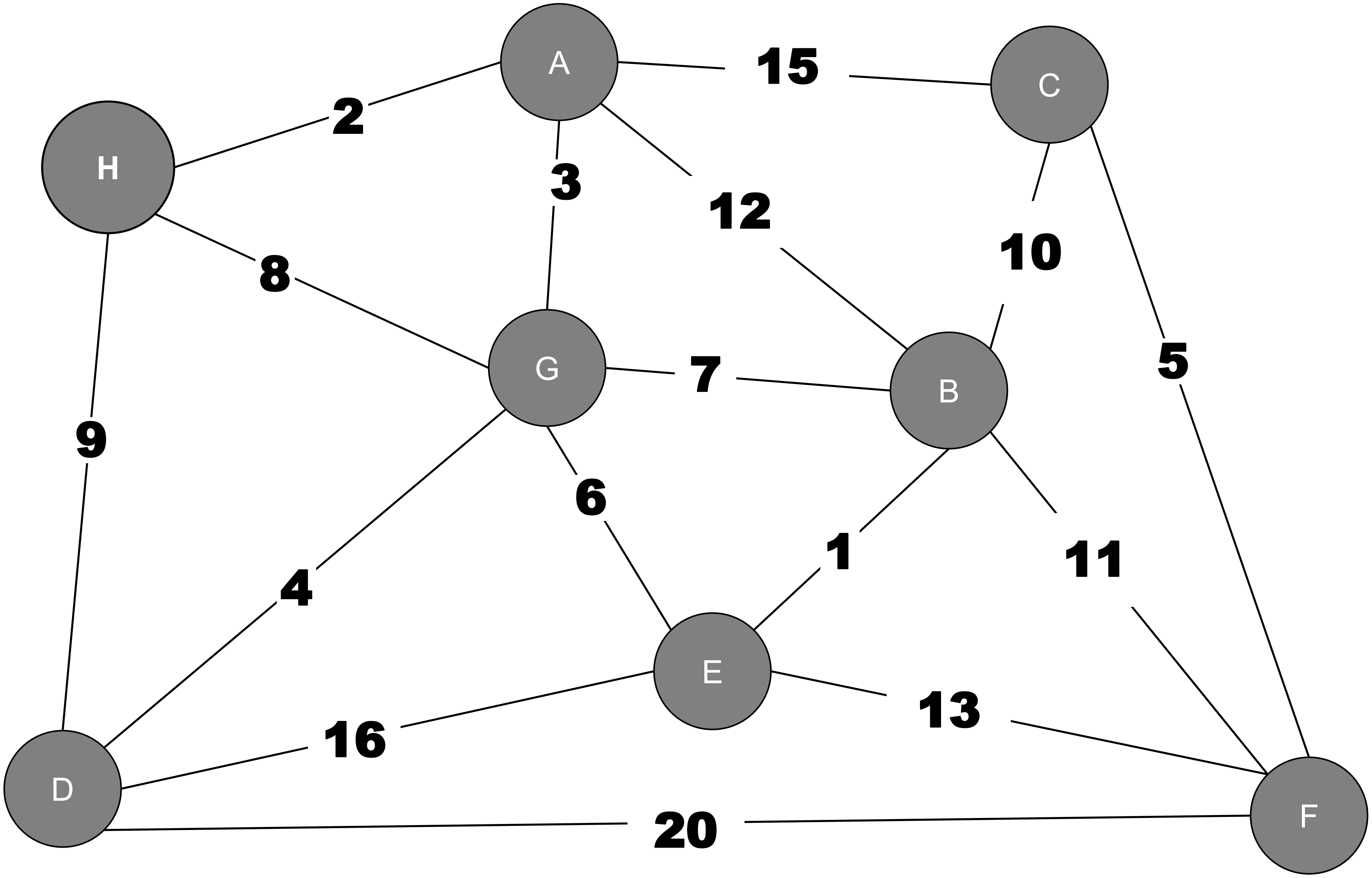
1. Initialise UFDS so that every node is its own disjoint set.
2. Sort the edges by their weights in increasing order.
3. For every edge **e = (u, v)**:
   a. If nodes **u** and **v** are in the same set, skip!

   b. Include edge **e** in our set of edges to output.

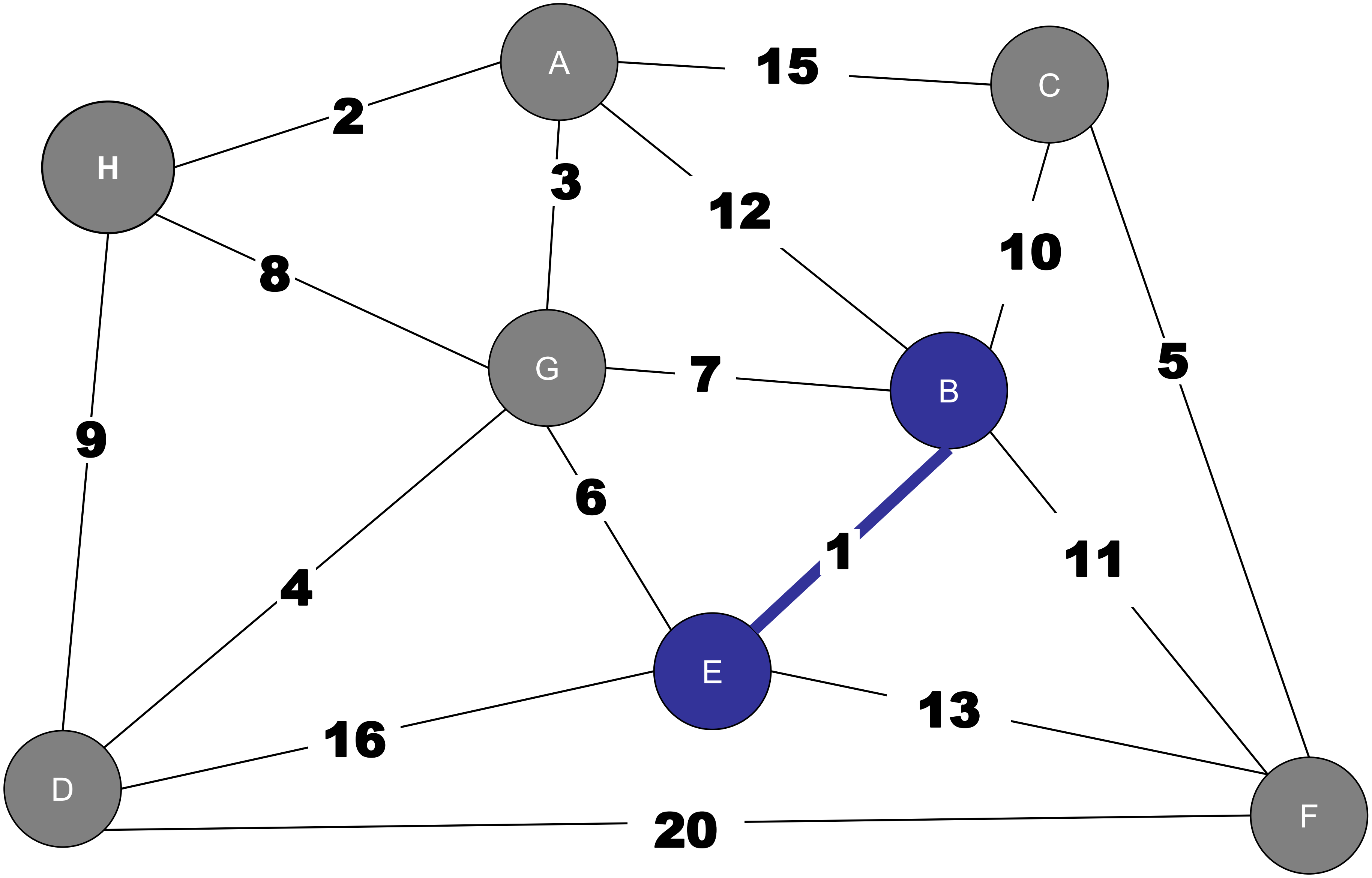
   c. Union the sets containing **u** and **v**.

   d. If we have added |V| - 1 edges, terminate!

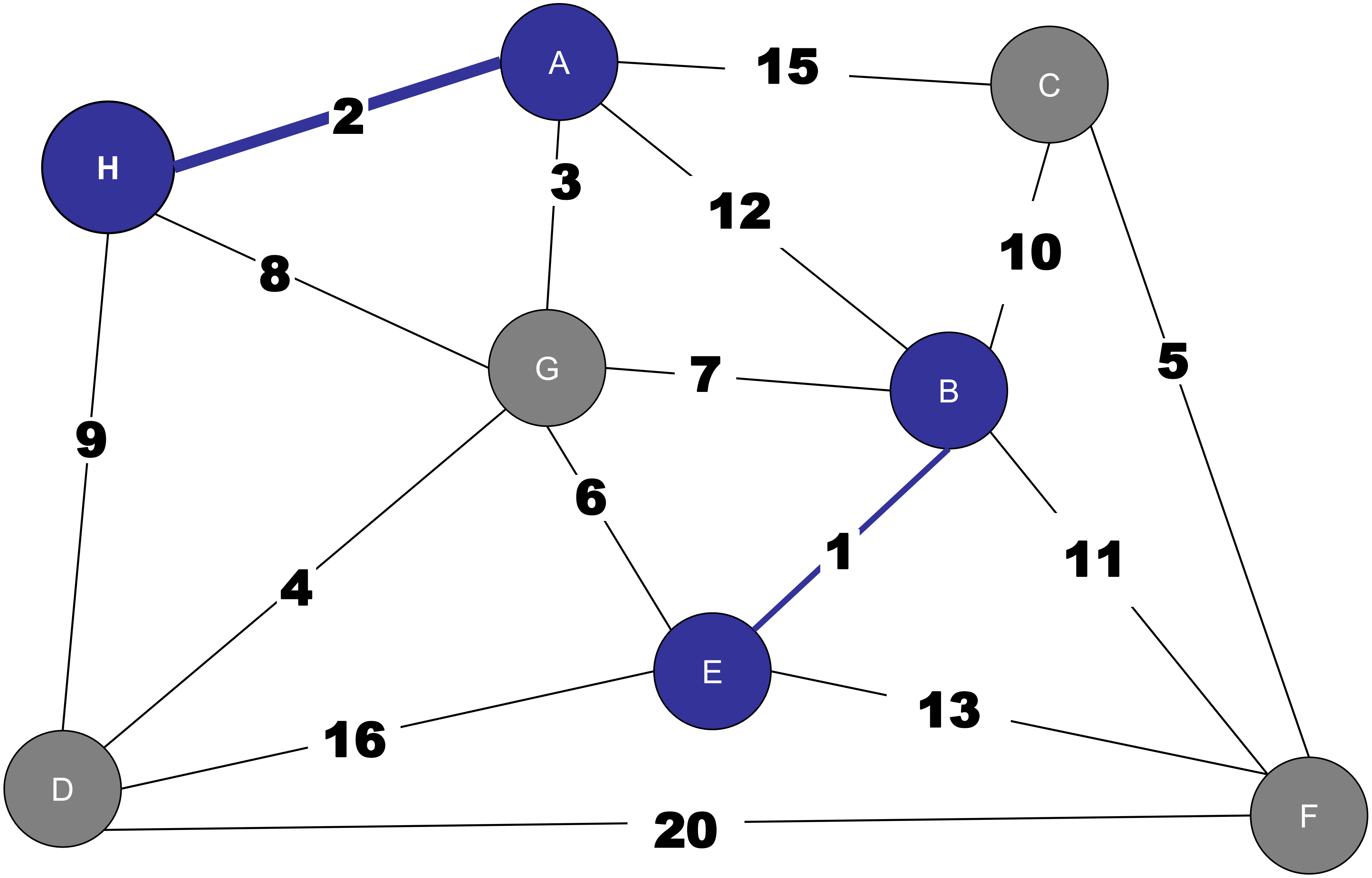
Kruskal's Example



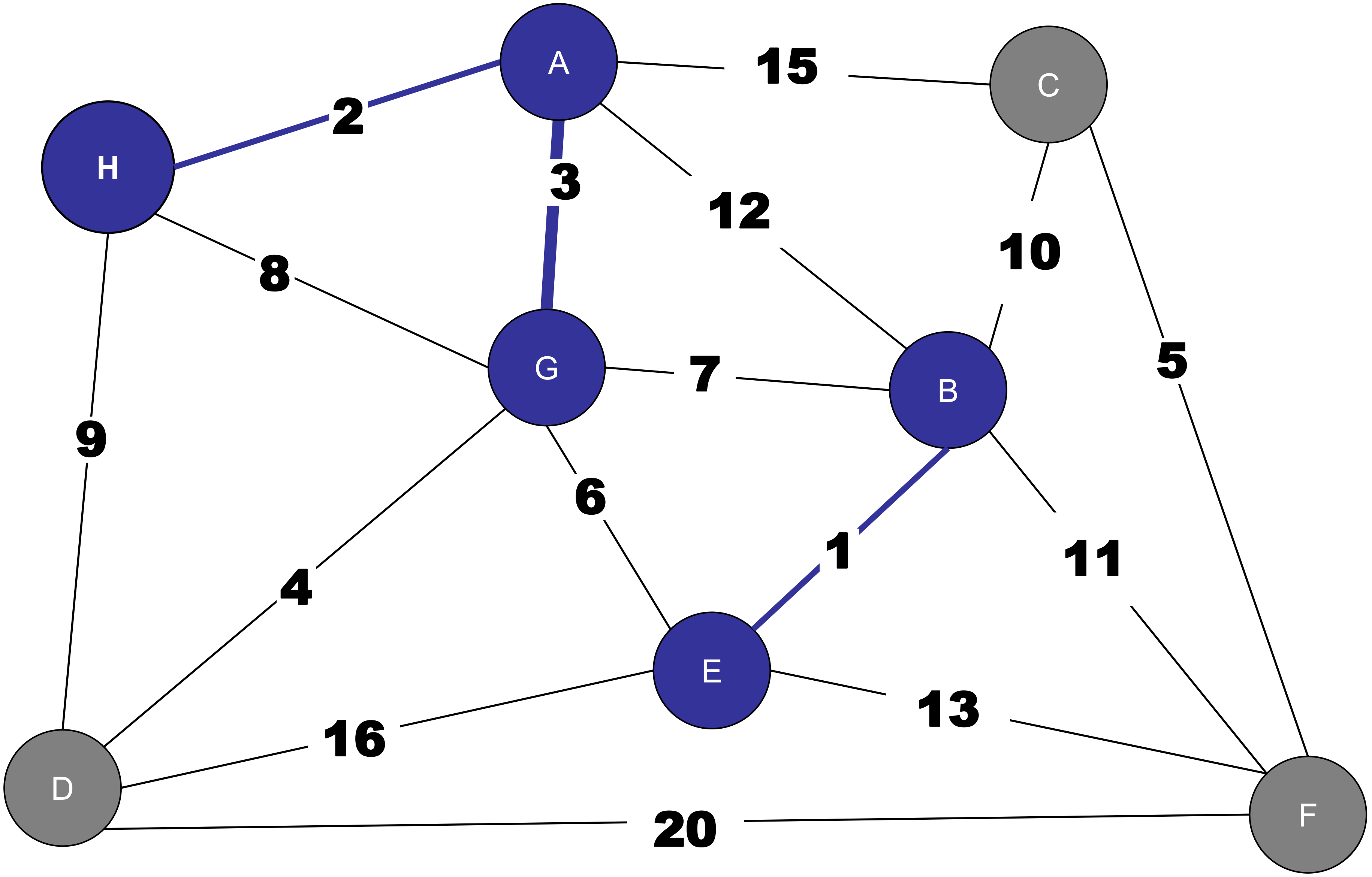
Kruskal’s Example



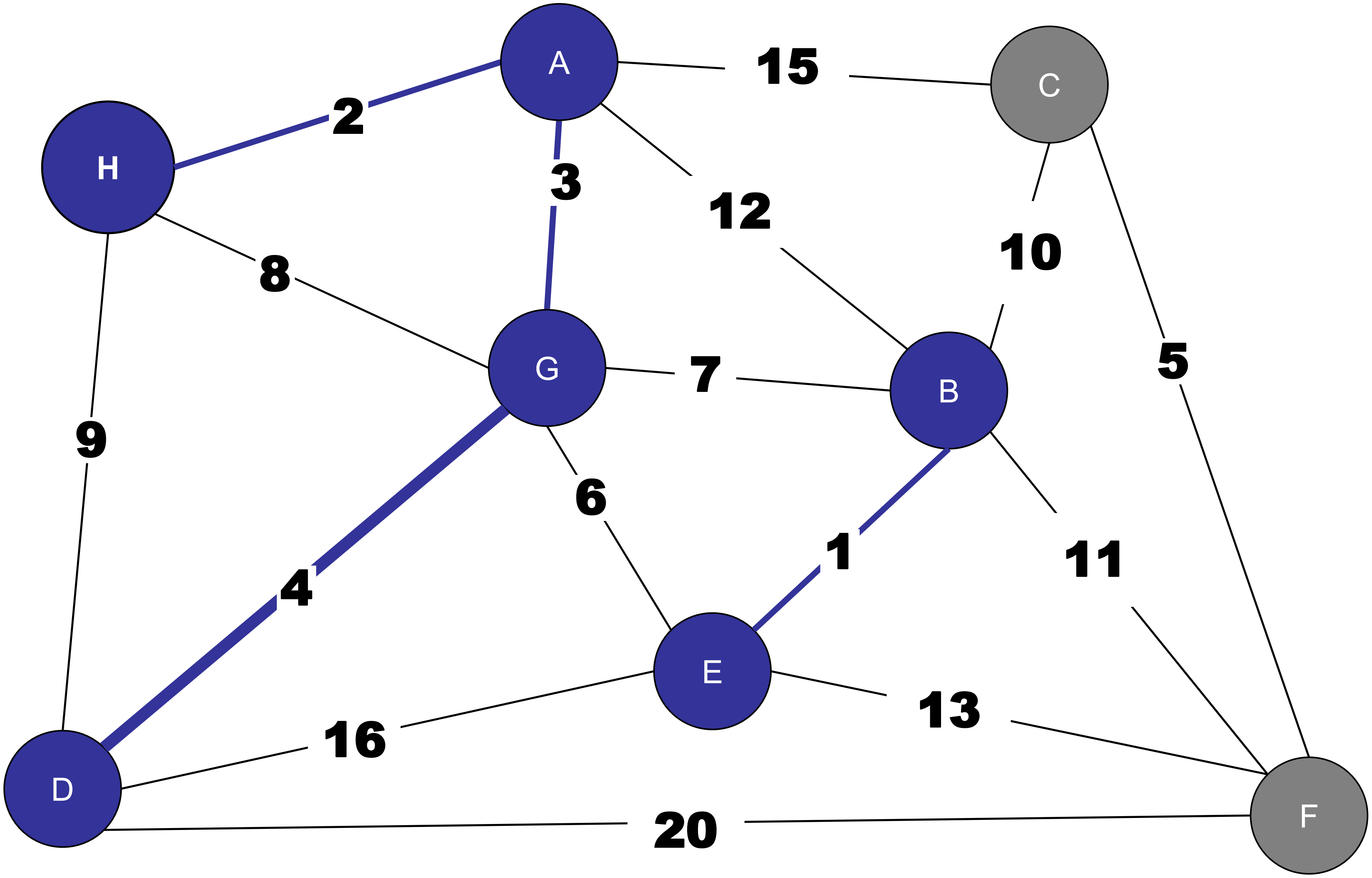
Kruskal's Example



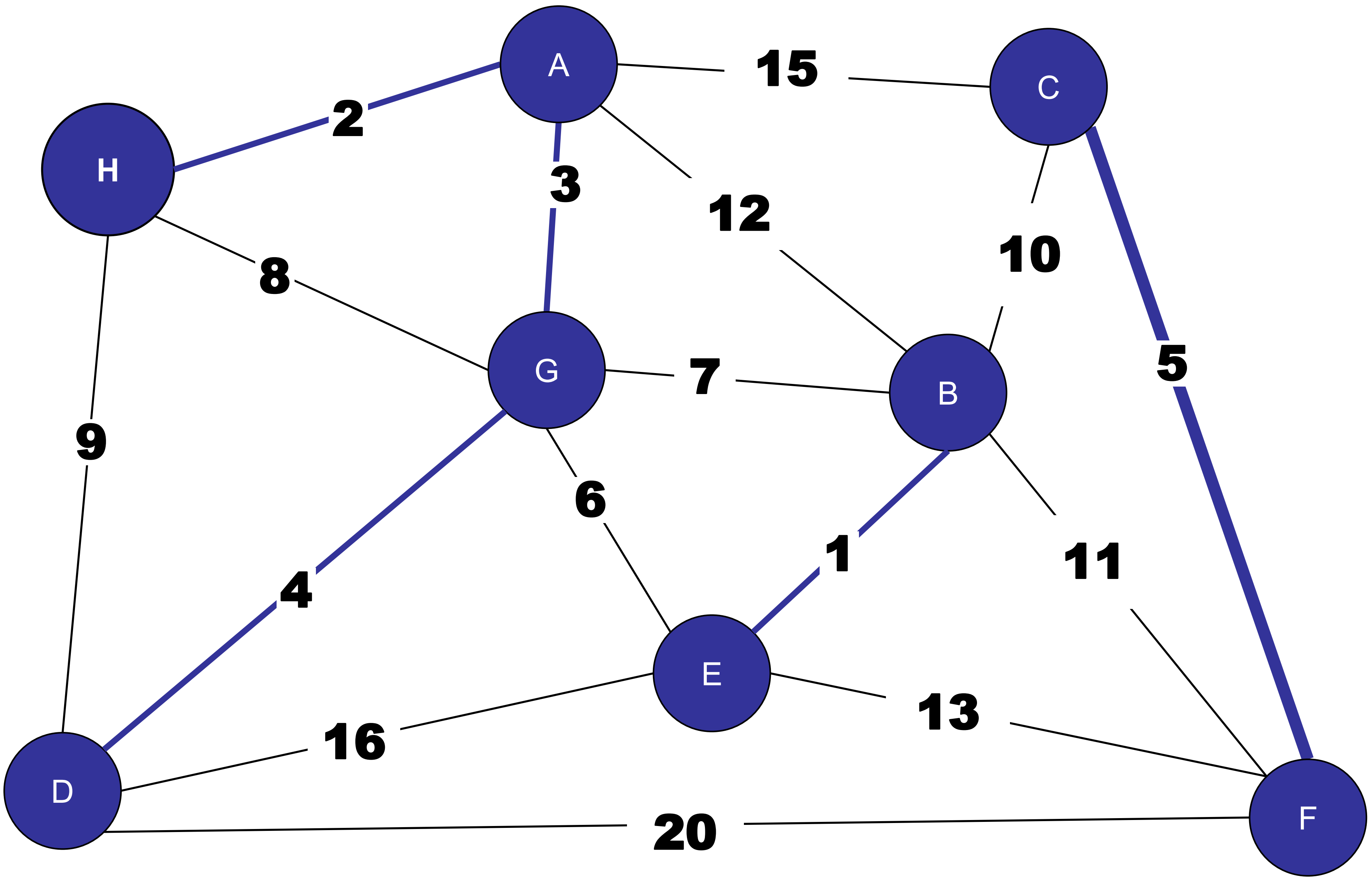
Kruskal's Example



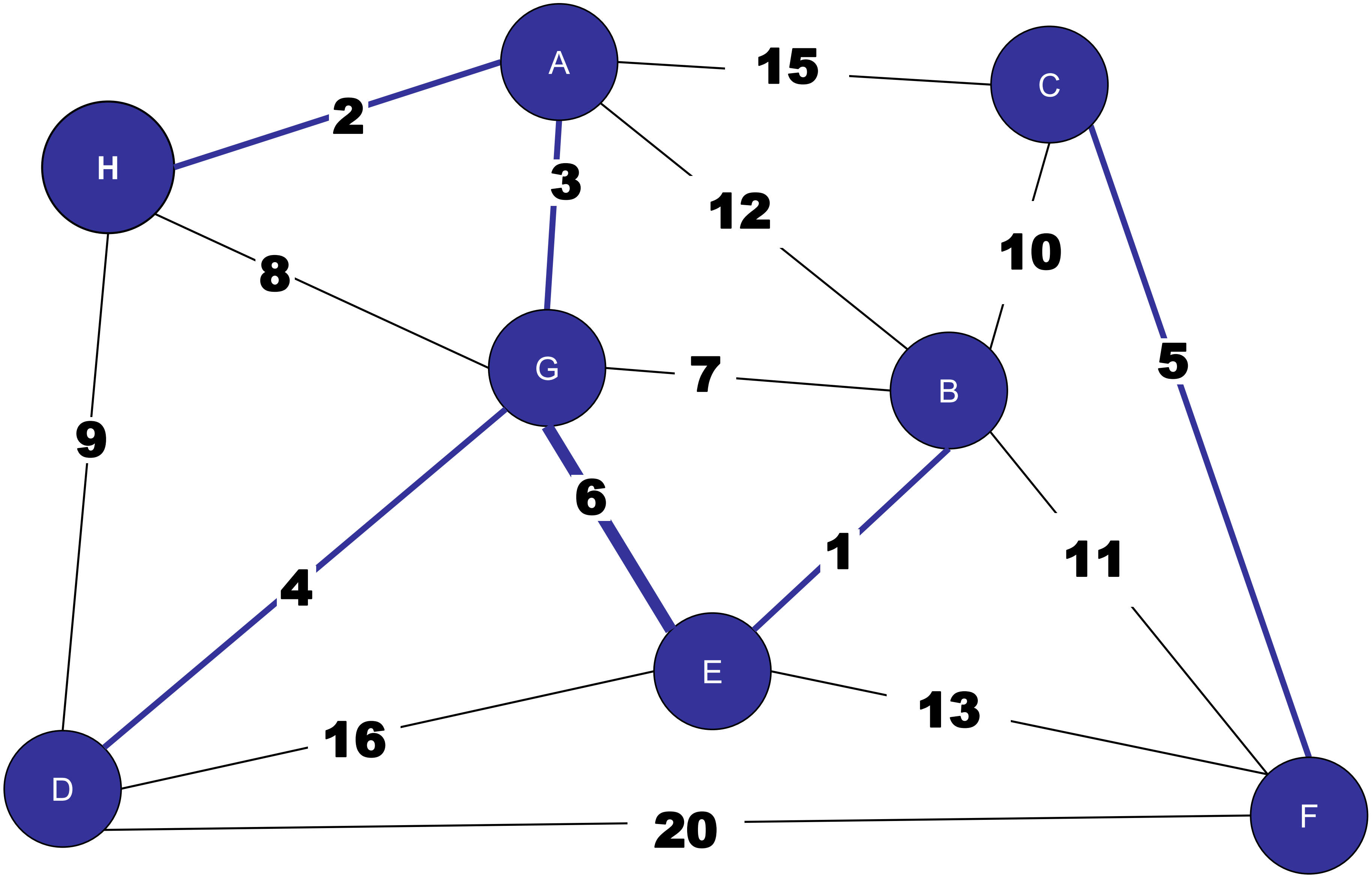
Kruskal's Example



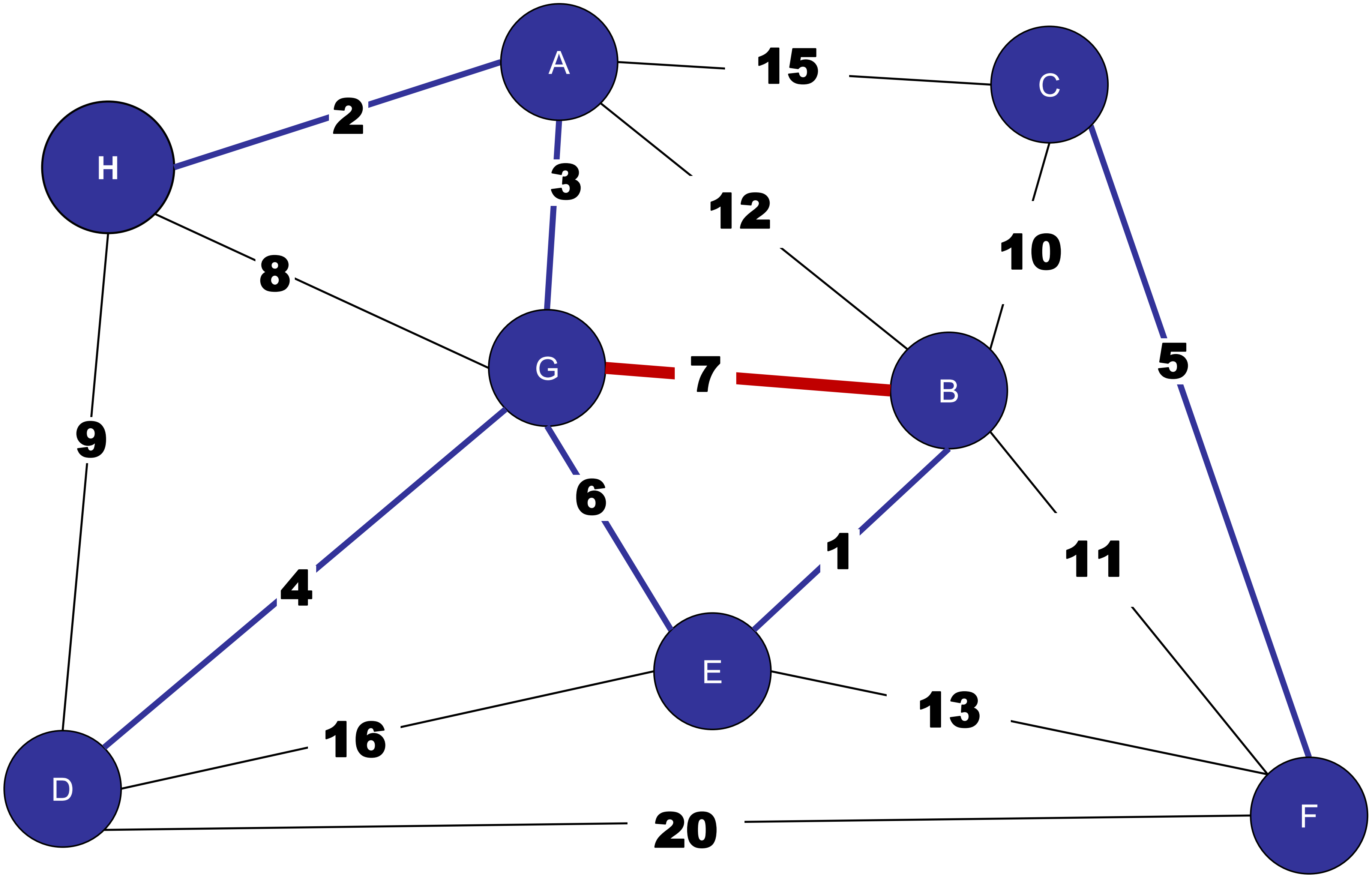
Kruskal's Example



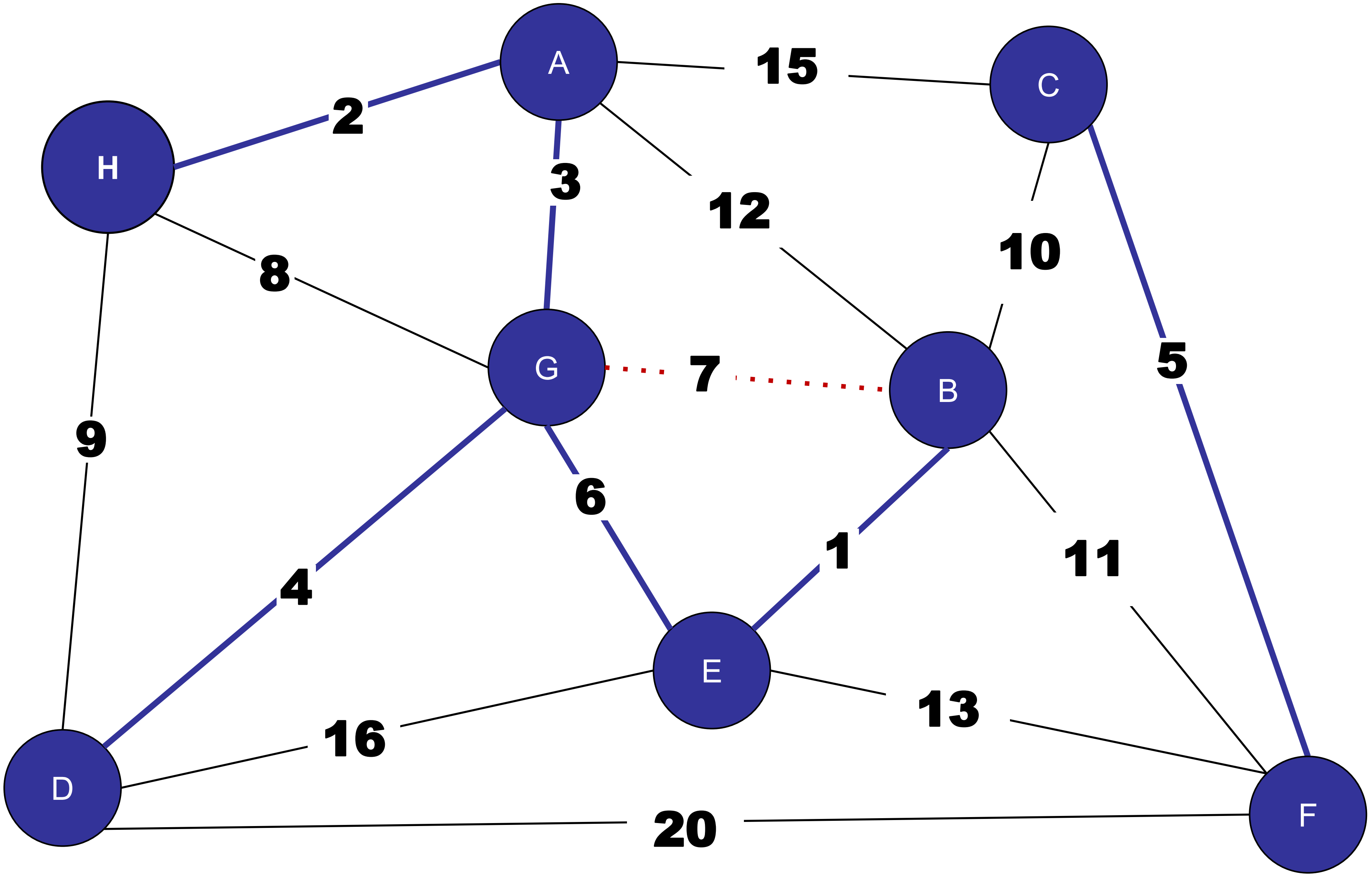
Kruskal's Example



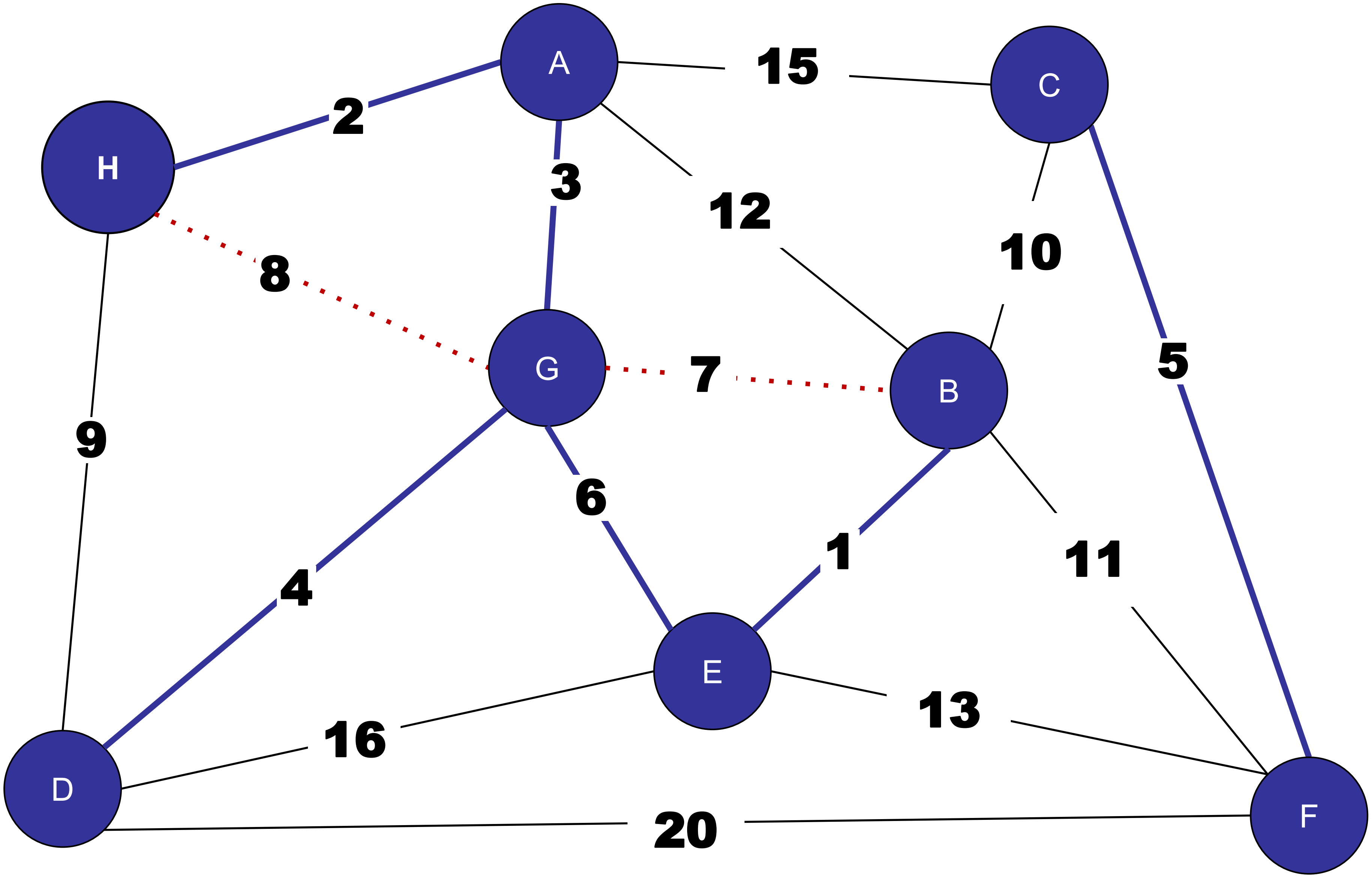
Kruskal's Example



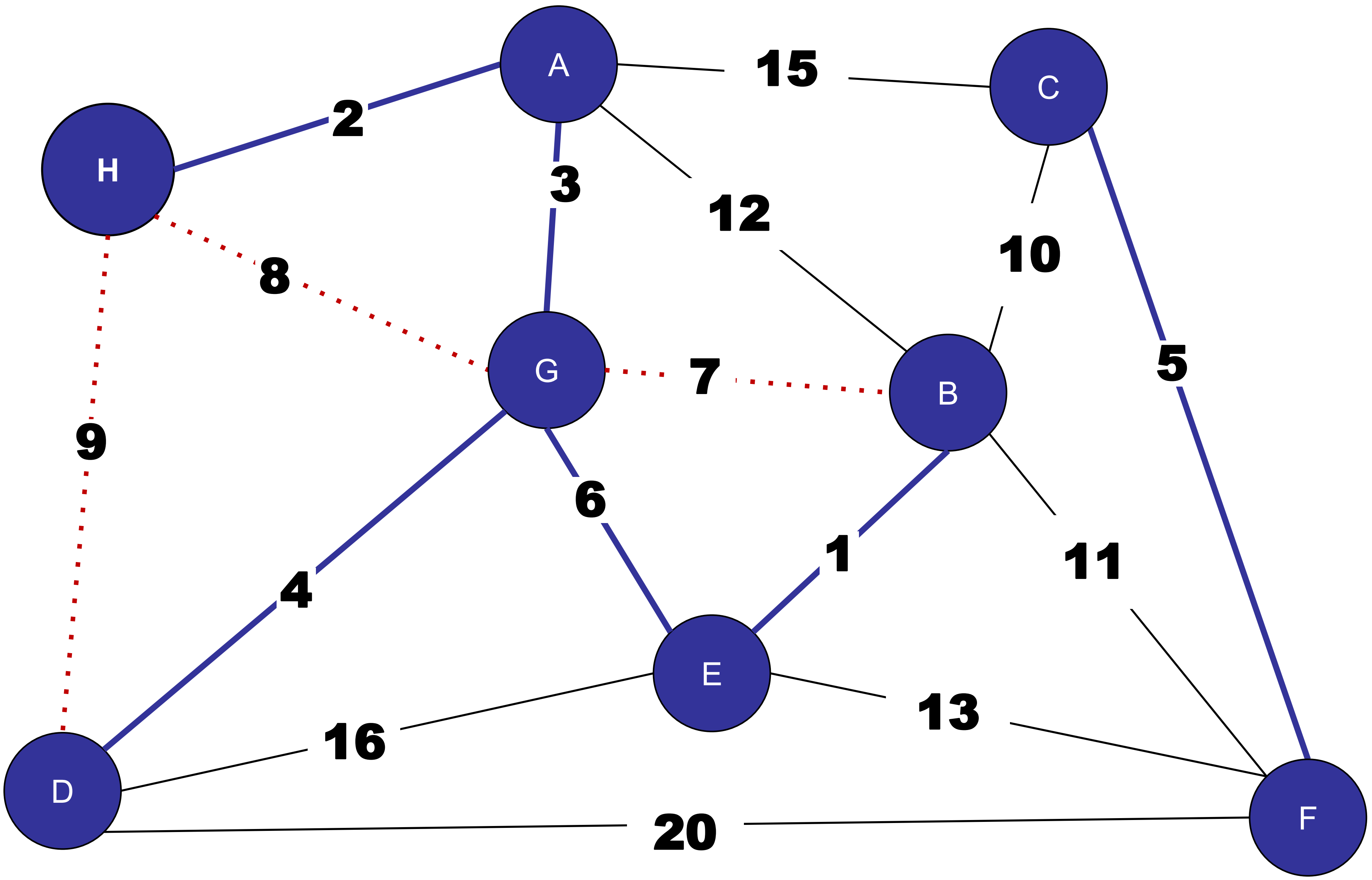
Kruskal’s Example



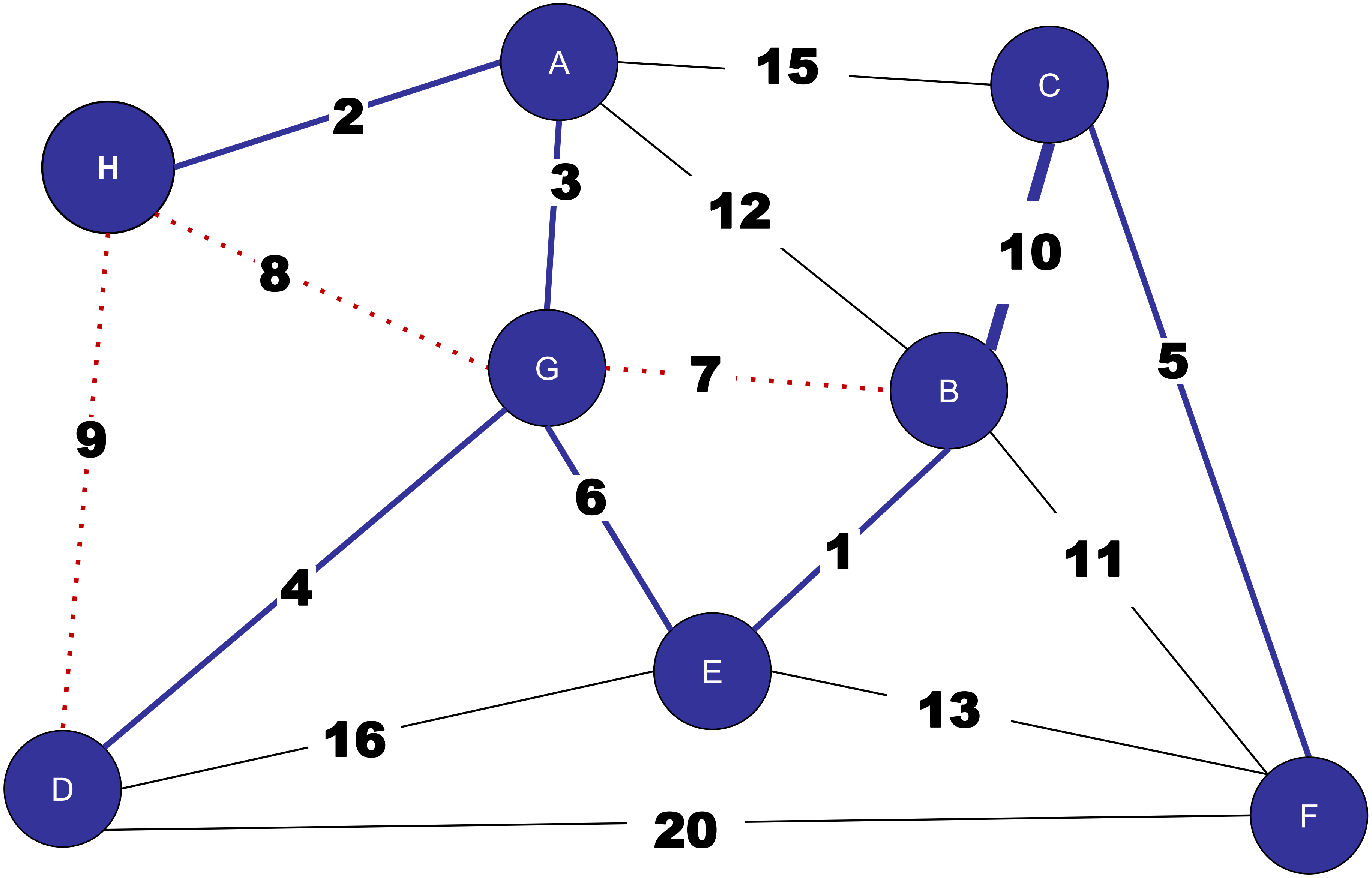
Kruskal's Example



Kruskal's Example

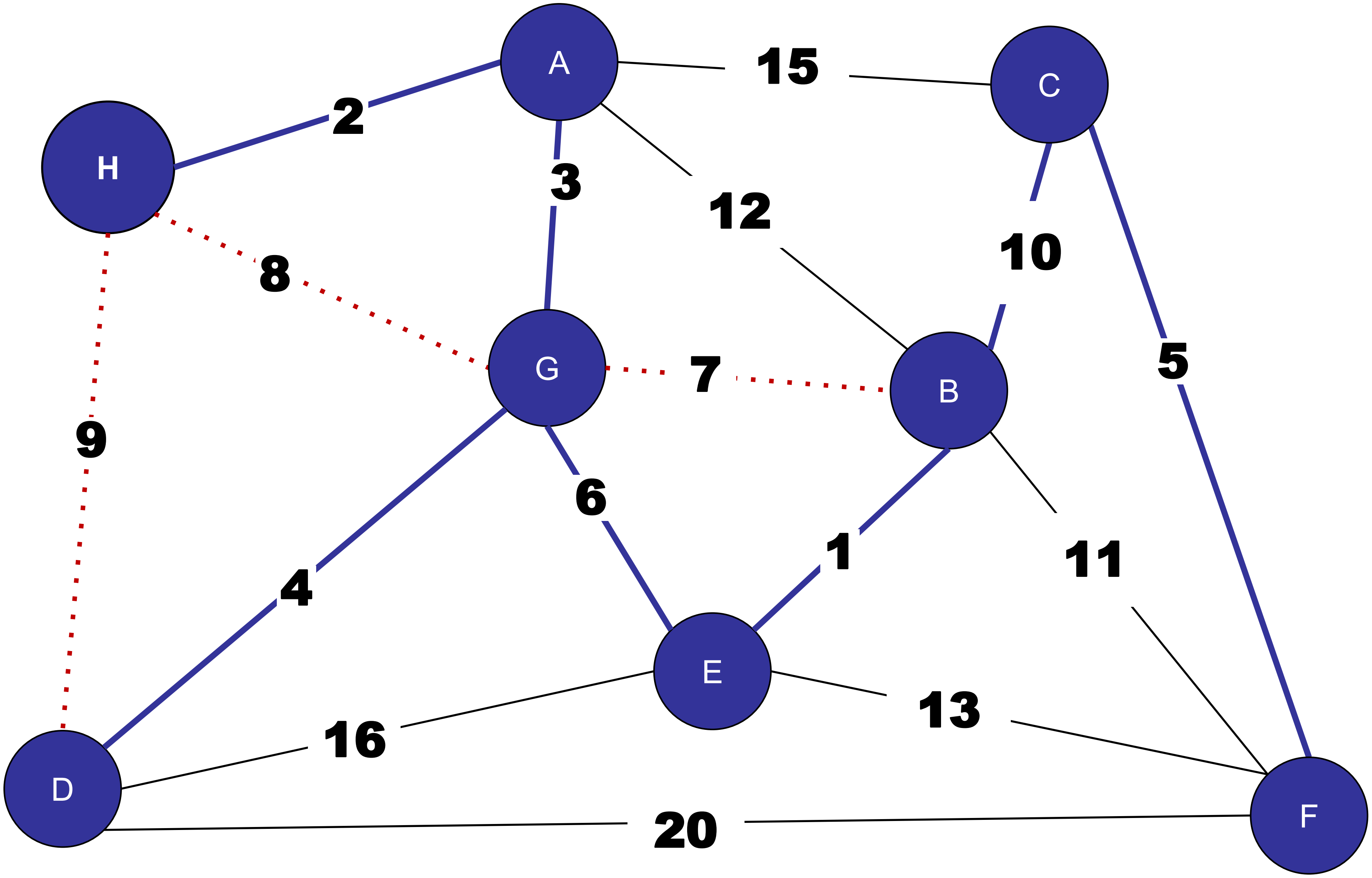


Kruskal’s Example



Kruskal's Example

Adding any more edges will just cause cycles.



Kruskal's Algorithm

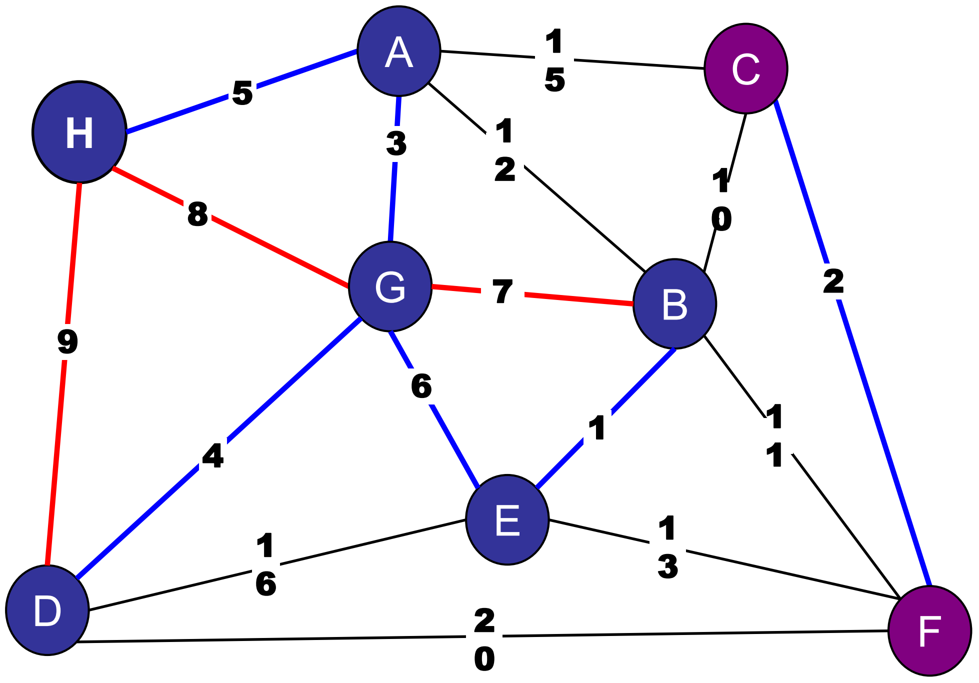
Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
    - If both endpoints are in the **same** blue tree, then color the edge red.
    - Otherwise, color the edge blue.

Proof:

- Each added edge crosses a cut.
- Each edge is the lightest edge across the cut: all other lighter edges across the cut have already been considered.



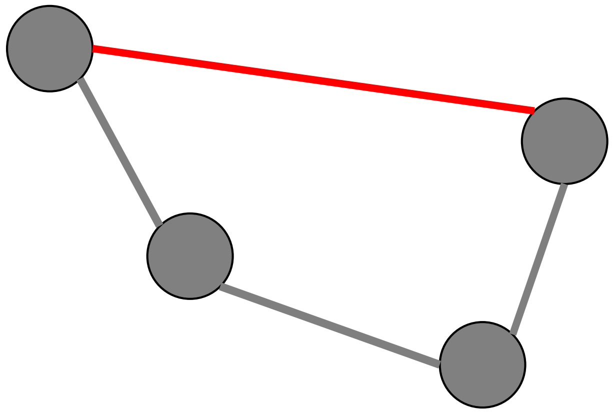
Generic MST Algorithm

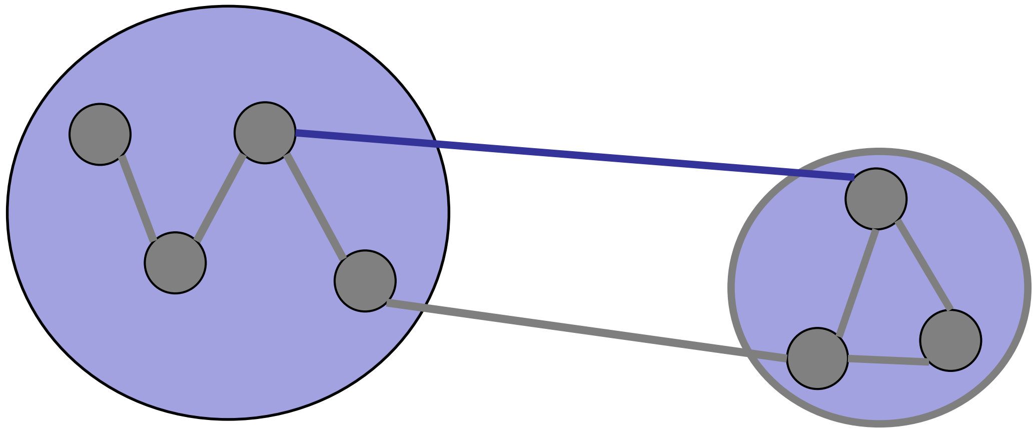
Greedy Algorithm:

Repeat:

Apply red rule or blue rule to an arbitrary edge.

until no more edges can be colored.





Recall: What is the running time of Kruskal's Algorithm on a connected graph?

1. O(V)
2. O(E)
3. O(E α)
4. O(V α)
5. [✓ O(E log V)]
6. O(V log E)

Kruskal's Algorithm

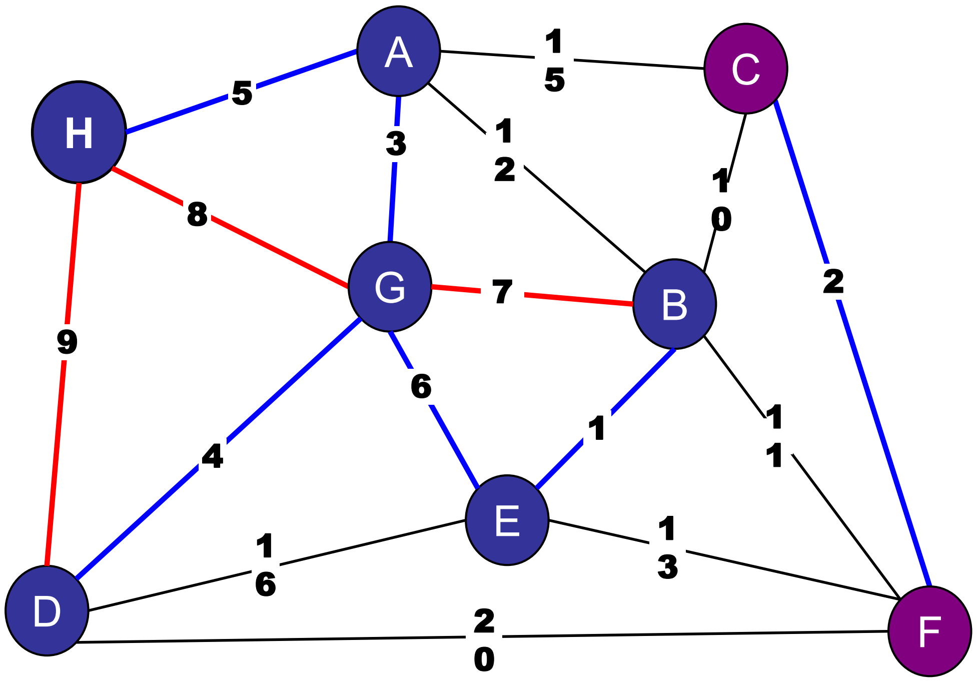
Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
    - If both endpoints are in the **same** blue tree, then color the edge red.
    - Otherwise, color the edge blue.

Performance:

- Sorting: O(E log E) = O(E log V)
- For E edges:
    - Find: O(α(n)) or O(log V)
    - Union: O(α(n)) or O(log V)



Roadmap

Minimum Spanning Trees

- Prim’s Algorithm
- Kruskal’s Algorithm
- (Boruvka’s Algorithm)

Variations

- Constant weight edges
- Bounded integer edge weights
- Directed graphs
- Maximum Spanning Tree

MST Variants

What if all the edges have the same weight?

How fast can you find an MST?

1. O(V)
2. [✓ O(E)]
3. O(E log V)
4. O(V log E)
5. O(VE)

MST Variants

What if all the edges have the same weight?

Depth-First-Search or Breadth-First-Search

The traversal tree is cycle-free!

If all edge-weights are 2, what is the **cost** of a MST?

1. V-1
2. V
3. [✓ 2(V-1)]
4. 2V
5. E-V
6. E

MST Variants

What if all the edges have the same weight?

- Depth-First-Search or Breadth-First-Search
- An MST contains exactly (V-1) edges.
- Every spanning tree contains (V-1) edges!
- Thus, any spanning tree you find with DFS/BFS is a minimum spanning tree.

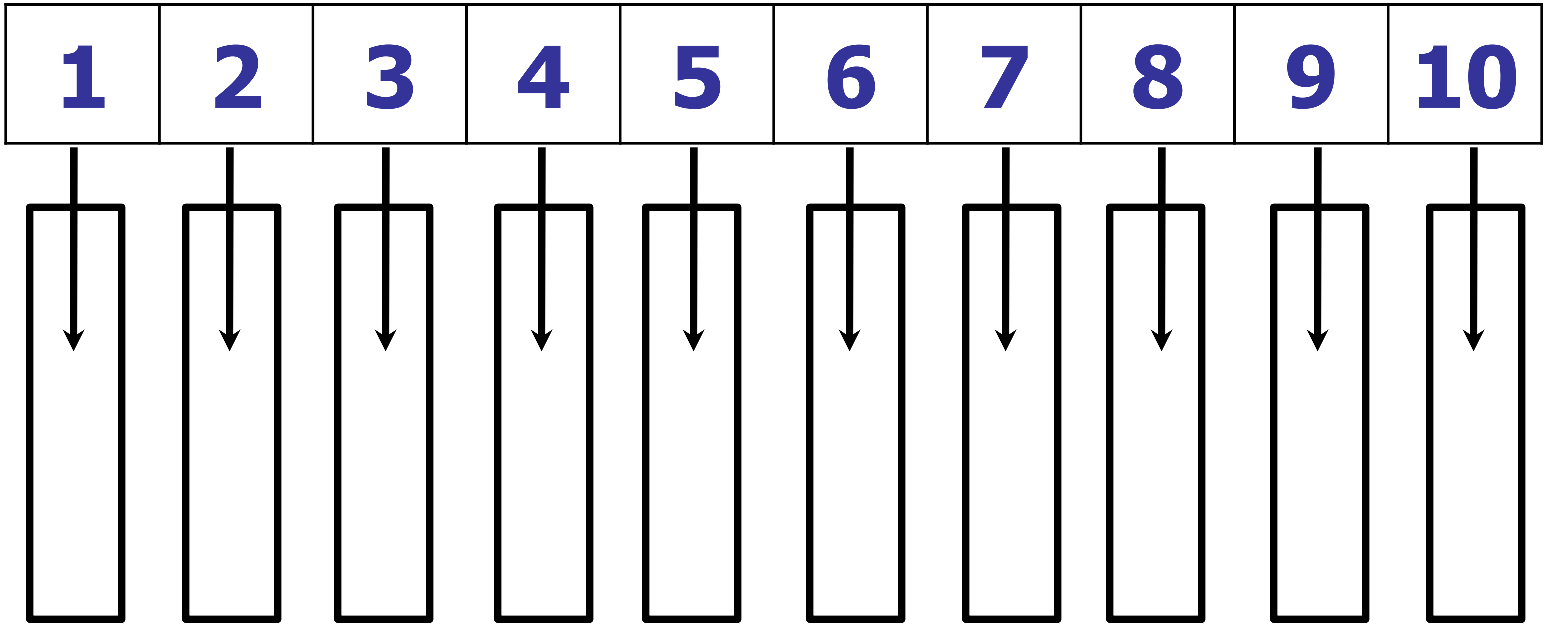
Kruskal’s Variants

What if all the edges have weights from {1..10}?

Kruskal’s Variants

What if all the edges have weights from {1..10}?

Idea: Use an array of size 10 to sort



slot A[j] holds a linked list of edges of weight j

Kruskal's Variants

What if all the edges have weights from {1..10}?

Idea: Use an array of size 10

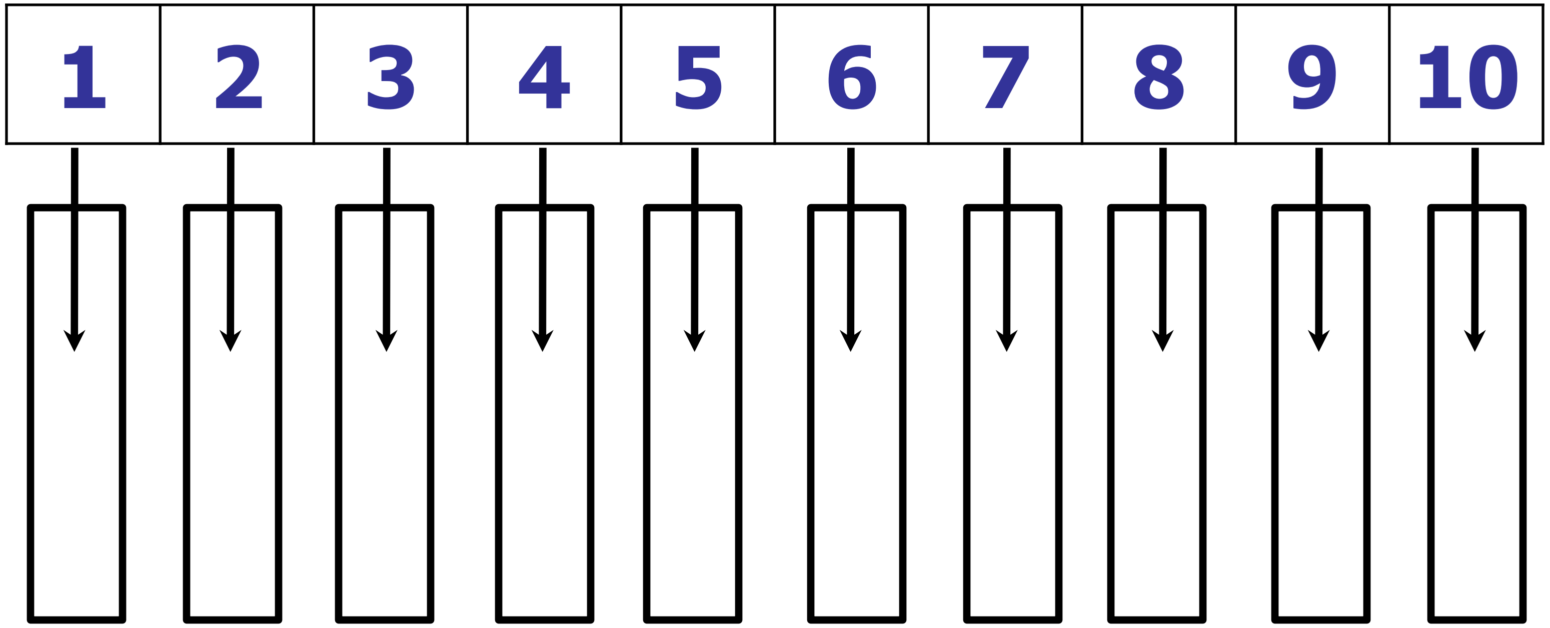
- Putting edges in array of linked lists: O(E)
- Iterating over all edges in ascending order: O(E)
- For each edge:
  - Checking whether to add an edge: O(α)
  - Union two components: O(α)

Total: O(αE)

Prim’s Variants

What if all the edges have weights from {1..10}?

Idea: Use an array of size 10 as a “Priority Queue”

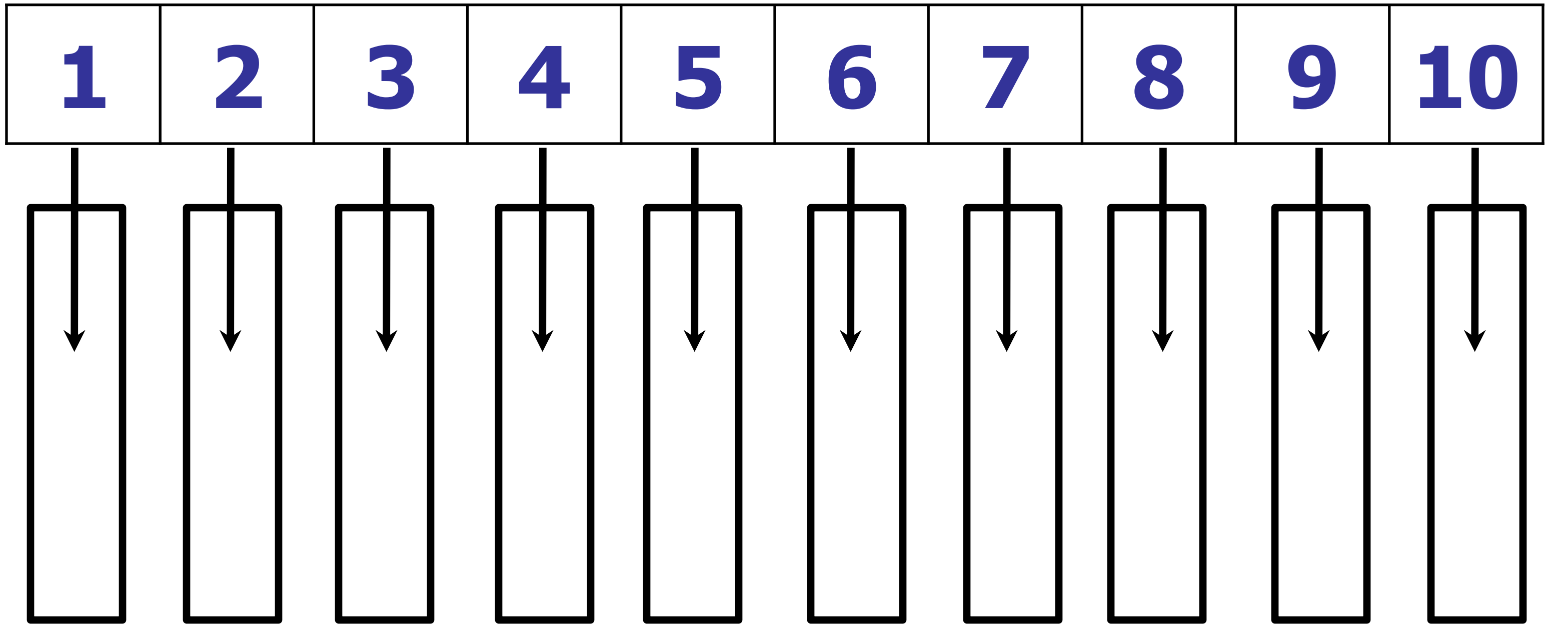


slot A[j] holds a linked list of **nodes** of weight j

Prim’s Variants

What if all the edges have weights from {1..10}?

Idea: Use an array of size 10 as a “Priority Queue”



decreaseKey: move node to new linked list

What is the running time of (modified) Prim’s if all the edge weights are in {1..10}?

1. O(V)
2. [✓ O(E)]
3. O(E log V)
4. O(V log E)
5. O(EV)

Prim’s Variants

What if all the edges have weights from {1..10}?

Implement Priority Queue:

- Use an array of size 10 to implement
- Insert: put node in correct list
- Remove: lookup node (e.g., in hash table) and remove from linked list.
- ExtractMin: Remove from the minimum bucket.
- DecreaseKey: lookup node (e.g., in hash table) and move to correct liked list.

Prim's Variants

What if all the edges have weights from {1..10}?

Idea: Use an array of size 10

- Inserting/Removing nodes from PQ: O(V)
- decreaseKey: O(E)

Total: O(V + E) = O(E)

Prim’s Variants

What if all the edges have weights from {1..10}?

Implement Priority Queue….

To think about: Why does this same idea fail for Dijkstra’s Algorithm?

Roadmap

Minimum Spanning Trees

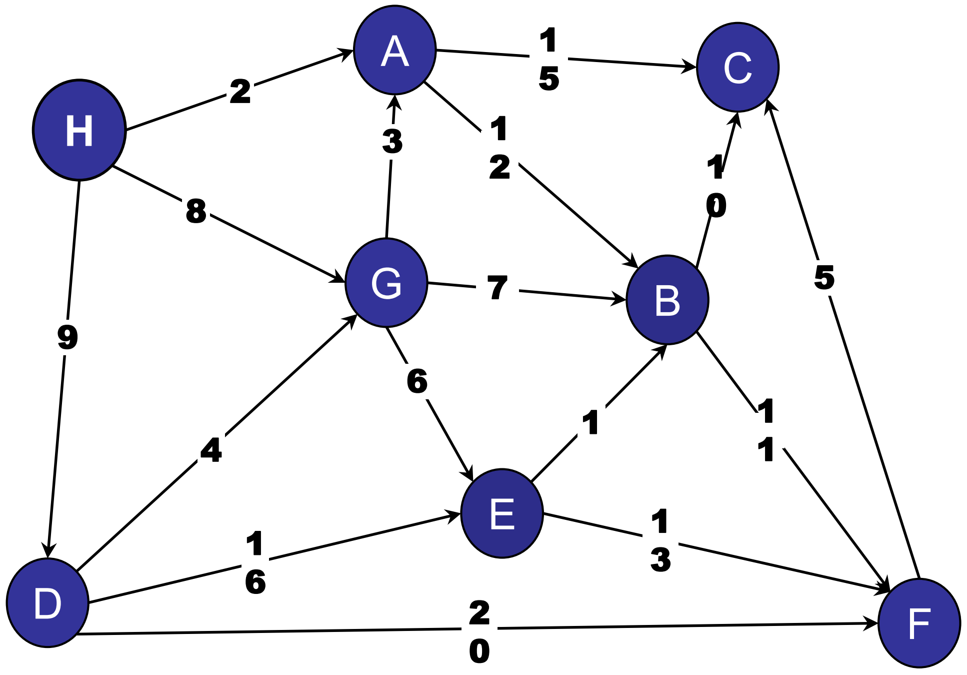
- Prim’s Algorithm
- Kruskal’s Algorithm
- (Boruvka’s Algorithm)

Variations:

- Constant weight edges
- Bounded integer edge weights
- Directed graphs
- Maximum Spanning Tree

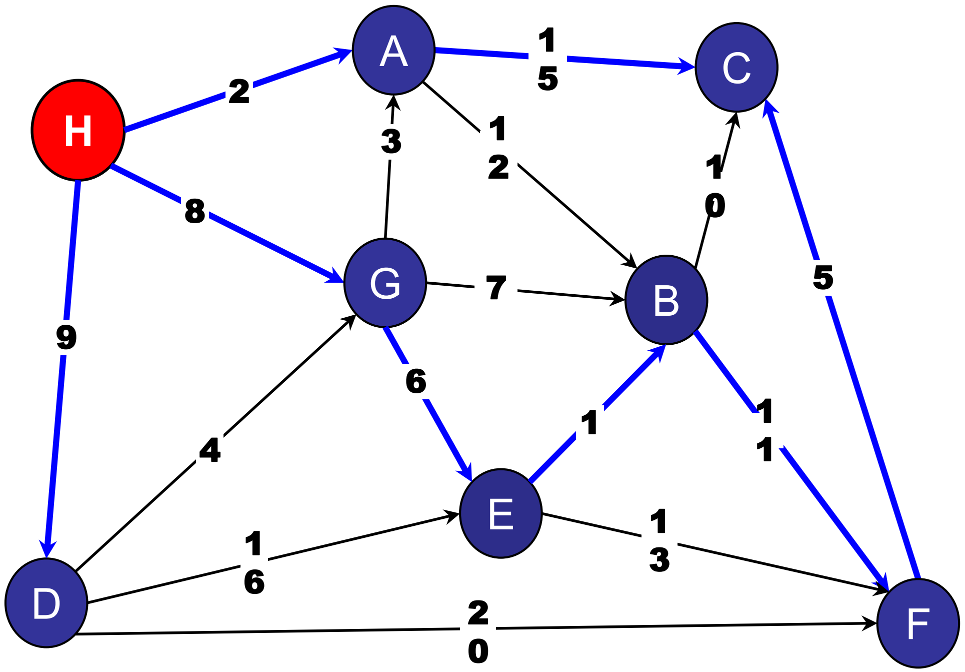
Directed Minimum Spanning Tree

What if the edges are directed?



Directed Minimum Spanning Tree

A rooted spanning tree:



Every node is reachable on a path from the root.

No cycles.

Directed Minimum Spanning Tree

Harder problem:

- Cut property does not hold.
- Cycle property does not hold.
- Generic MST algorithm does not work.

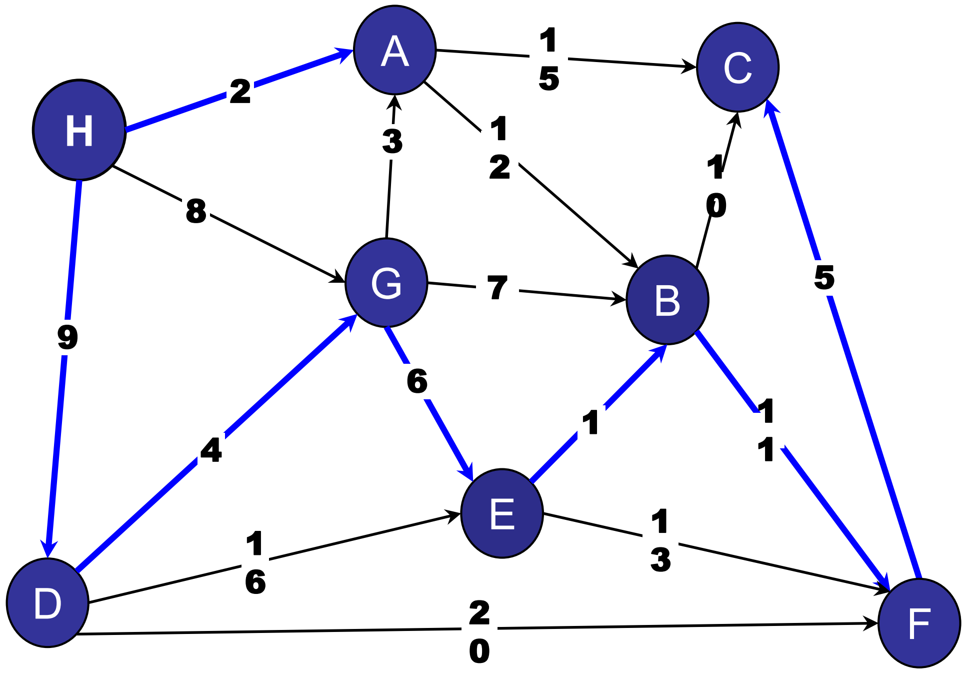
Prim’s, Kruskal’s, Boruvka’s do not work.

See CS3230 for more details…

Exercise: Draw a directed graph where the cut property or cycle property is violated.

Directed Minimum Spanning Tree

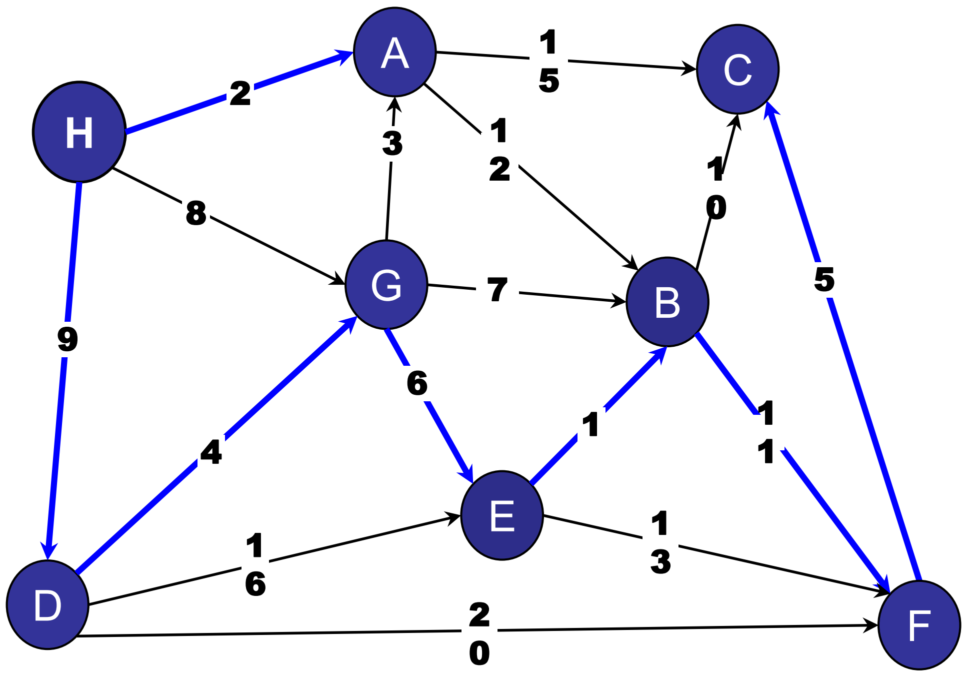
Special case: a directed acyclic graph with one “root”:



Directed Minimum Spanning Tree

For a directed acyclic graph with one “root”:

For every node except the root: add minimum weight incoming edge.



Directed Minimum Spanning Tree

For a directed acyclic graph with one "root":

For every node except the root: add minimum weight incoming edge.

Observations:

- No cycles (since acyclic graph).
- Each edge is chosen only once.

Tree:
- V nodes
- V – 1 edges
- No cycles

Directed Minimum Spanning Tree

For a directed acyclic graph with one "root":

For every node except the root: add minimum weight incoming edge.

Observations:

- No cycles (since acyclic graph).
- Each edge is chosen only once.

Tree:
    V nodes
    V – 1 edges
    No cycles

- Every node has to have at least one incoming edge in the MST, so this is the minimum spanning tree.

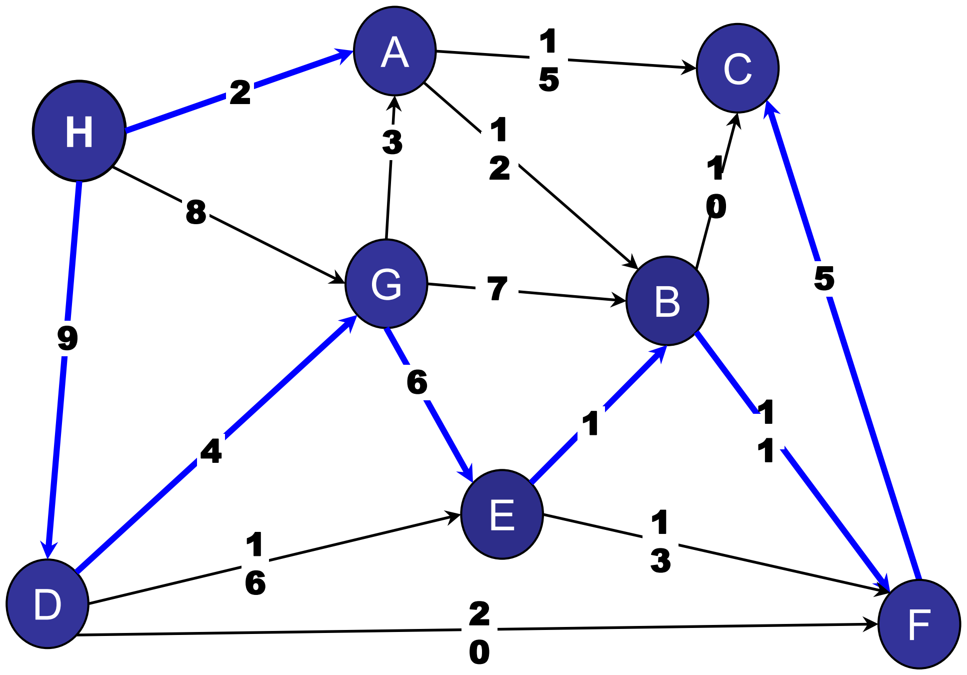
Directed Minimum Spanning Tree

For a directed acyclic graph with one "root":

For every node except the root: add minimum weight incoming edge.

Conclusion: Minimum Spanning Tree

O(E) time



Maximum Spanning Tree

A MaxST is a spanning tree of maximum weight.

How do you find a MaxST?

Maximum Spanning Tree

Reweighting a spanning tree:

- What happens if you add a constant k to the weight of every edge?

Kruskal's Algorithm

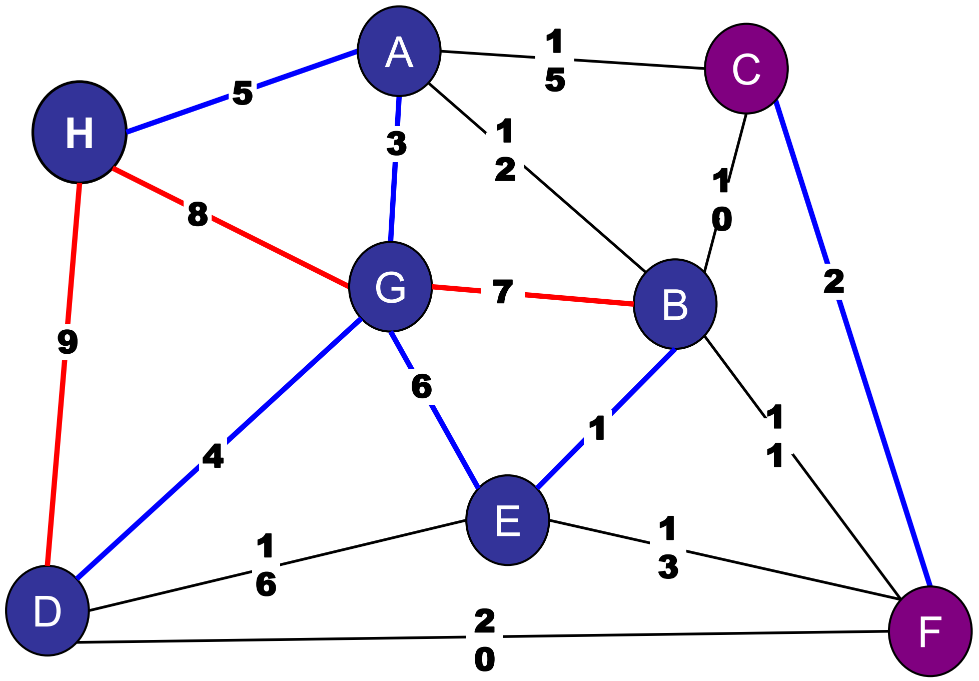
Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.

What matters?

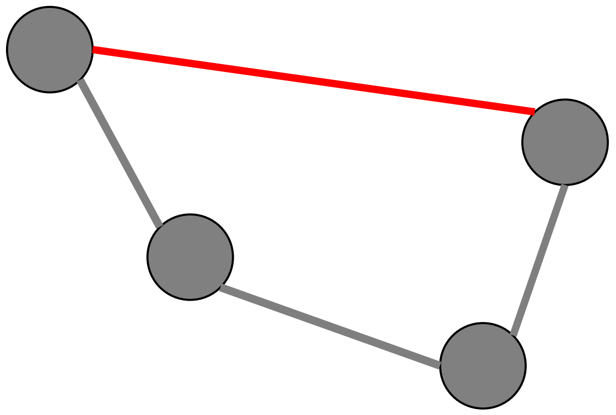
- Relative edge weights.
- Absolute edge weights have no impact.



Generic MST Algorithm

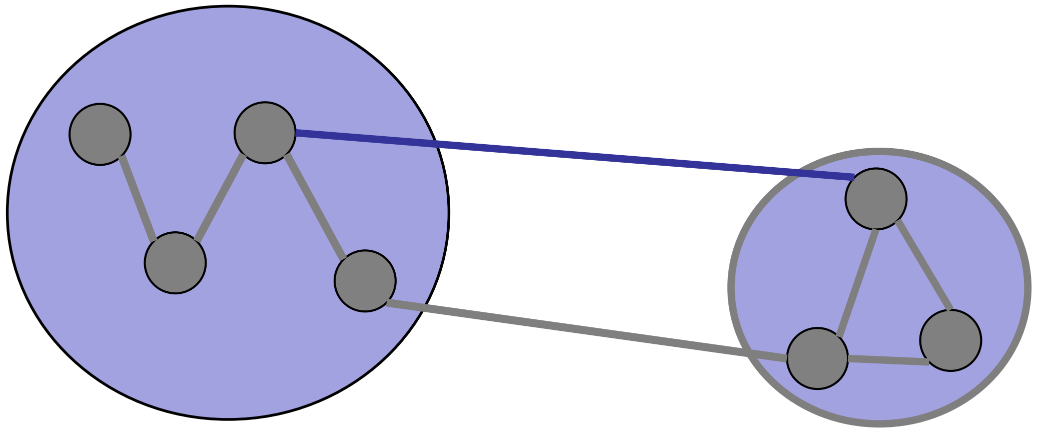
Red rule:

If C is a cycle with no red arcs, then color the max-weight edge in C red.



Blue rule:

If D is a cut with no blue arcs, then color the min-weight edge in D blue.



Maximum Spanning Tree

Reweighting a spanning tree:

- What happens if you add a constant k to the weight of every edge?

No change!

We can add or subtract weights without affecting the MST.

(Very *different* from shortest paths…)

Maximum Spanning Tree

MST with negative weights?

Maximum Spanning Tree

MST with negative weights?

No problem!

1. Reweight MST by adding a big enough value to each edge so that it is positive.

1. Actually, no need to reweight. Only relative edge weights matter, so negative weights have no bad impact.

Maximum Spanning Tree

A MaxST is a spanning tree of maximum weight.

How do you find a MaxST?

Easy!

1. Multiply each edge weight by -1.
2. Run MST algorithm.
3. MST that is “most negative” is the maximum.

Maximum Spanning Tree

A MaxST is a spanning tree of maximum weight.

How do you find a MaxST?

Or… run Kruskal’s in reverse.

Roadmap

Minimum Spanning Trees

- The MST Problem
- Basic Properties of an MST
- Generic MST Algorithm
- Prim's Algorithm
- Kruskal's Algorithm
- Variations
- **Boruvka's Algorithm**

MST Algorithms

Classic:

- Prim’s Algorithm
- Kruskal’s Algorithm

Modern requirements:

- Parallelizable
- Faster in “good” graphs (e.g., planar graphs)
- Flexible

Boruvka's Algorithm

Origin: 1926

- Otakar Boruvka
- Improve the electrical network of Moravia

Based on generic algorithm:

- Repeat: add all "obvious" blue edges.
- Very simple, very flexible.

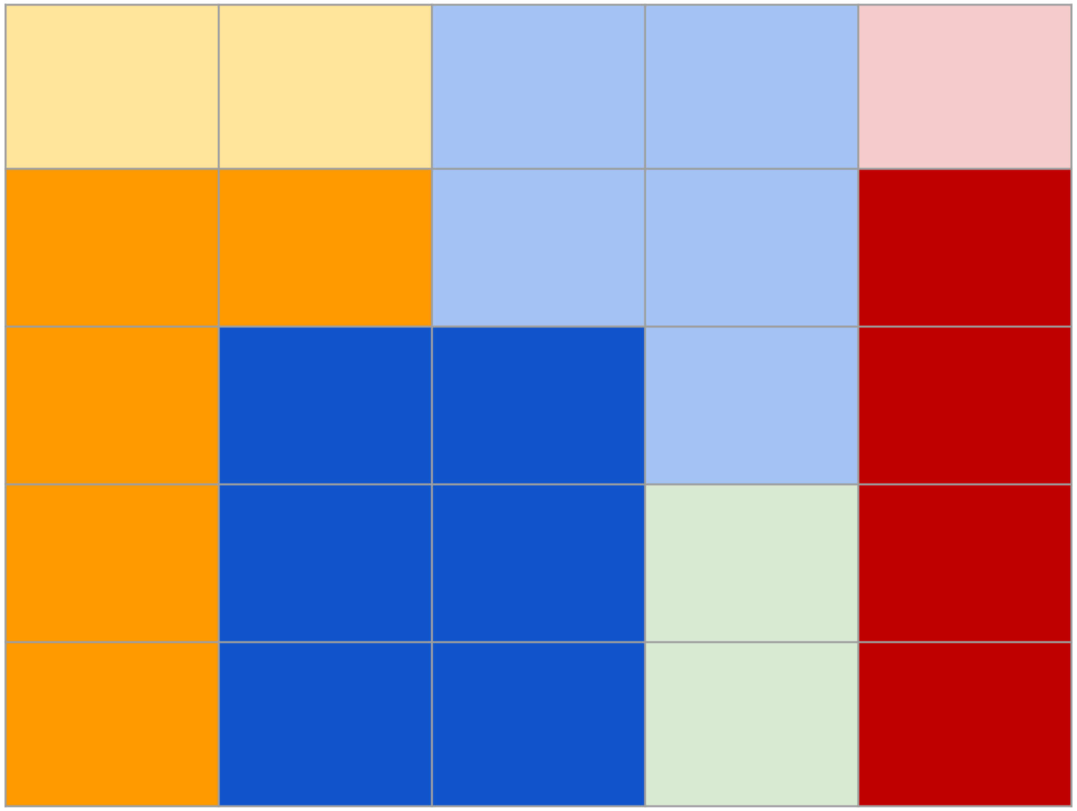
Why MST? Real Life Applications

MST as an idea is really useful for “clustering” pixels in pictures to segment portions of an image.

It’s not the only way, but it can be easily parallelised!

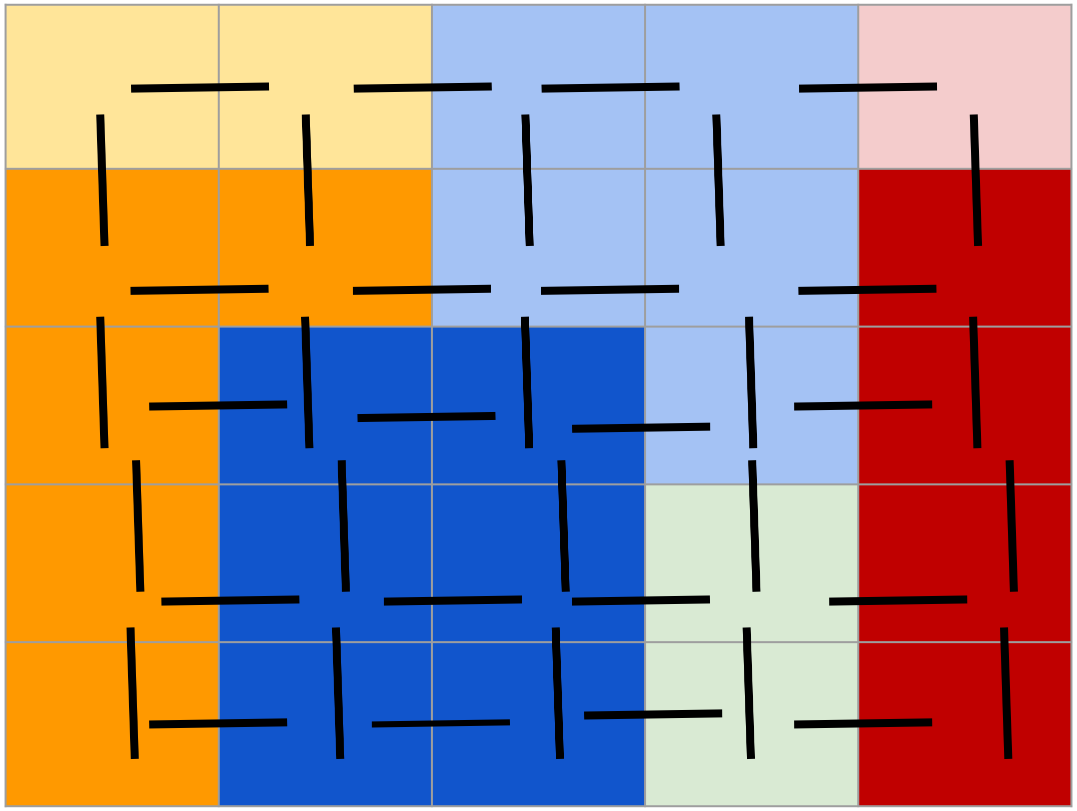
Why MST? Real Life Applications

Each pixel is a node



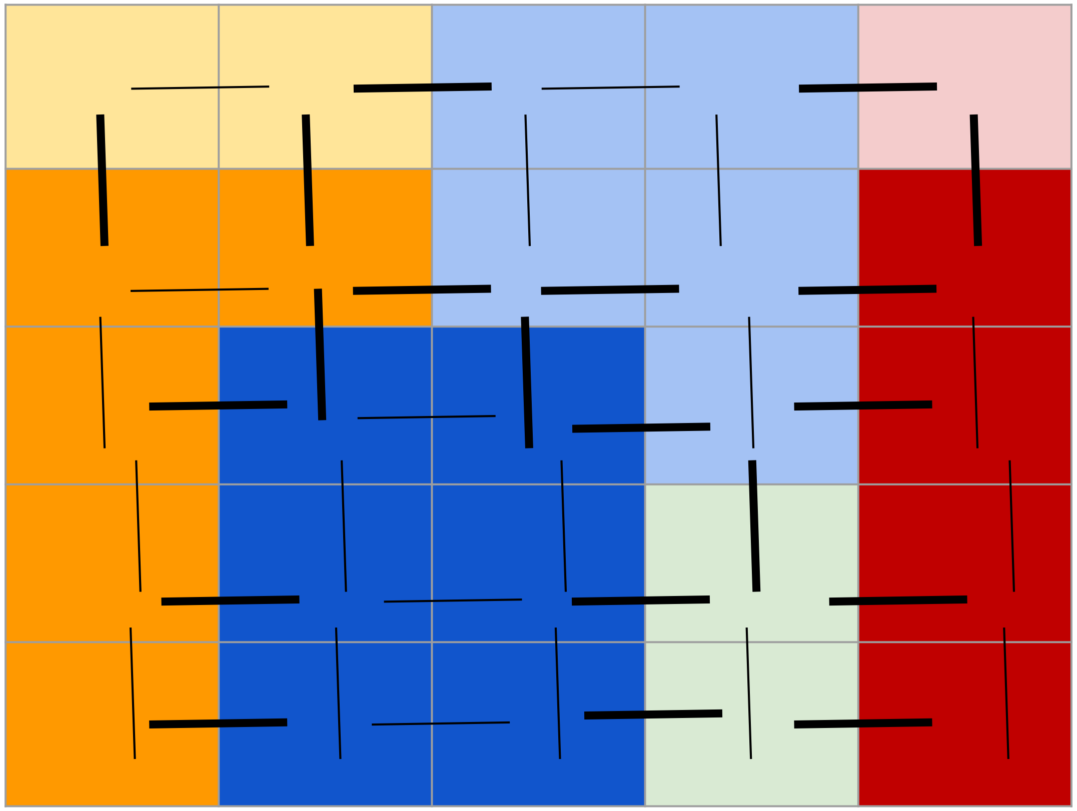
Why MST? Real Life Applications

Each pixel is a node, and there is a weighted edge to its neighbours. (Weight is based on various factors like RGB value, and hue)



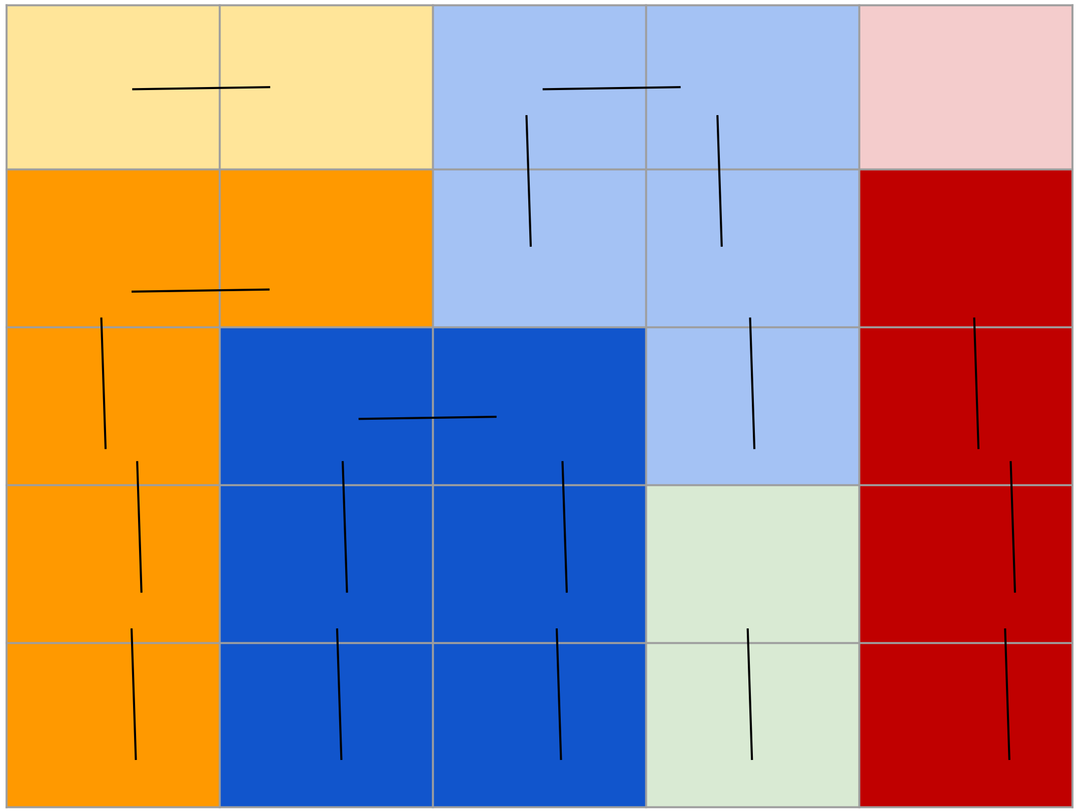
Why MST? Real Life Applications

Higher dissimilarity implies higher weight



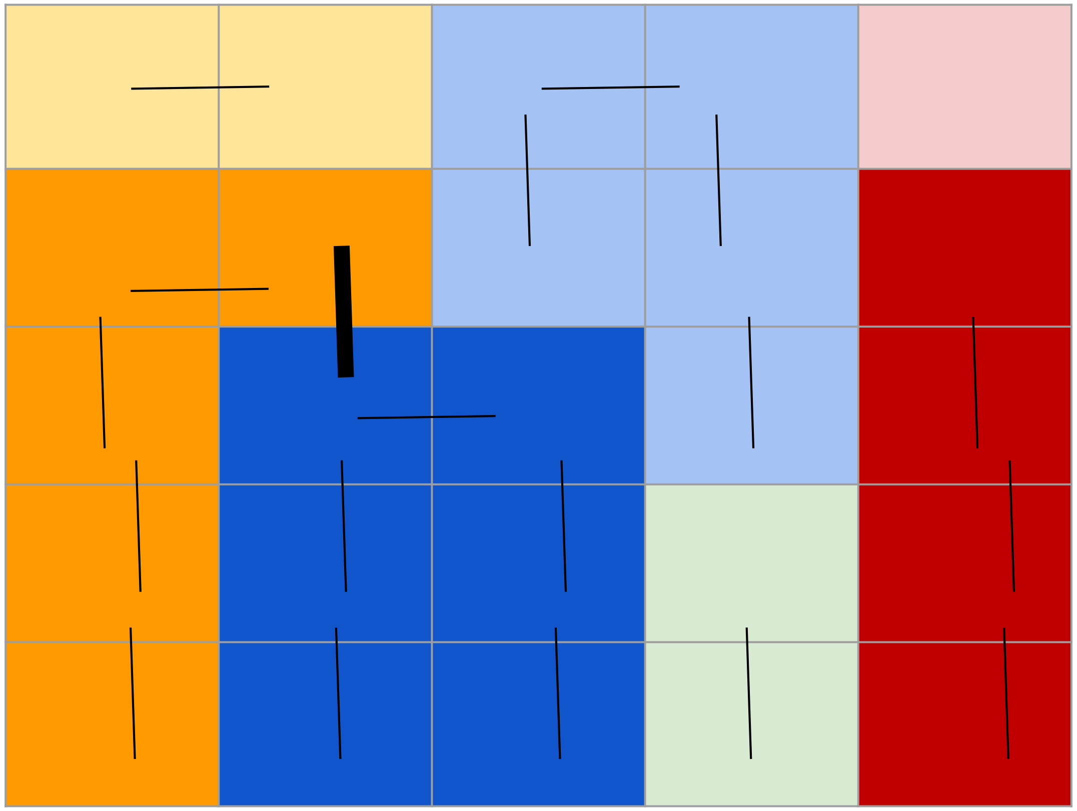
Why MST? Real Life Applications

Intuition: Similar pixels all have lower weight, likely to be grouped first



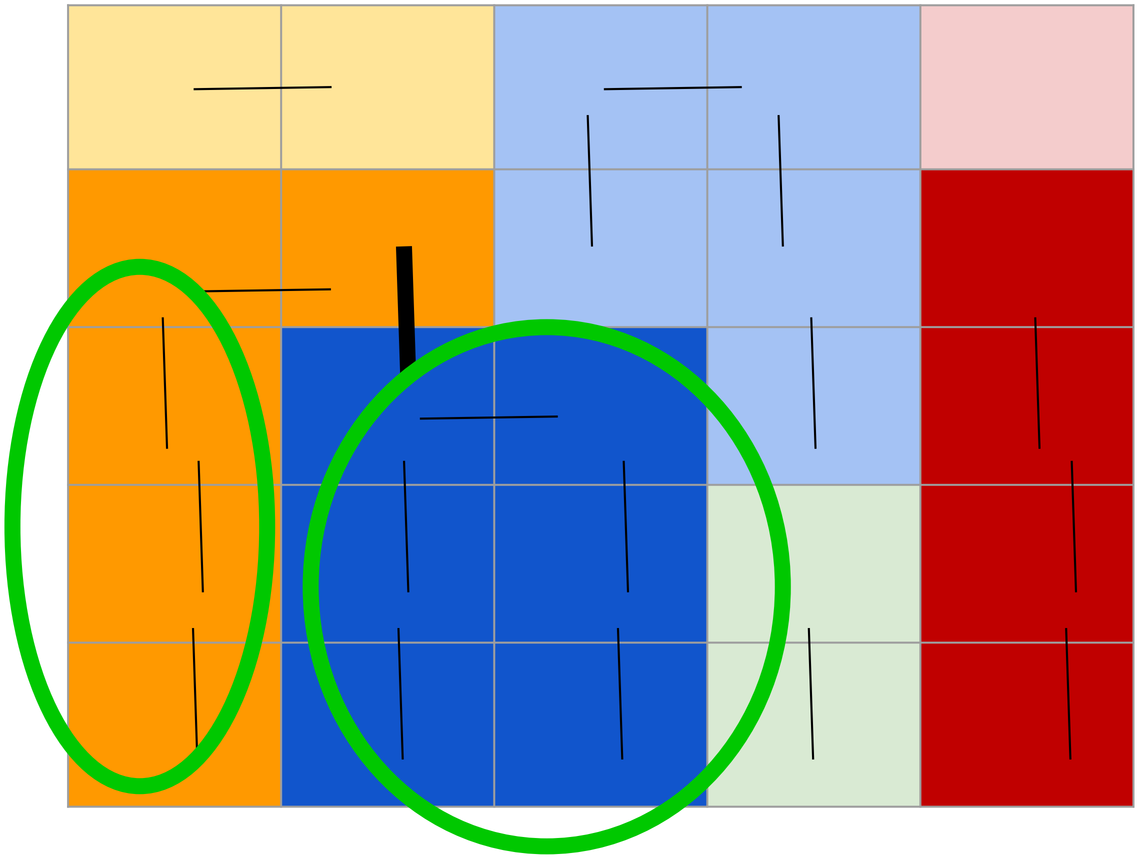
Why MST? Real Life Applications

Intuition: By the time we come around to looking at potentially border crossing edges,



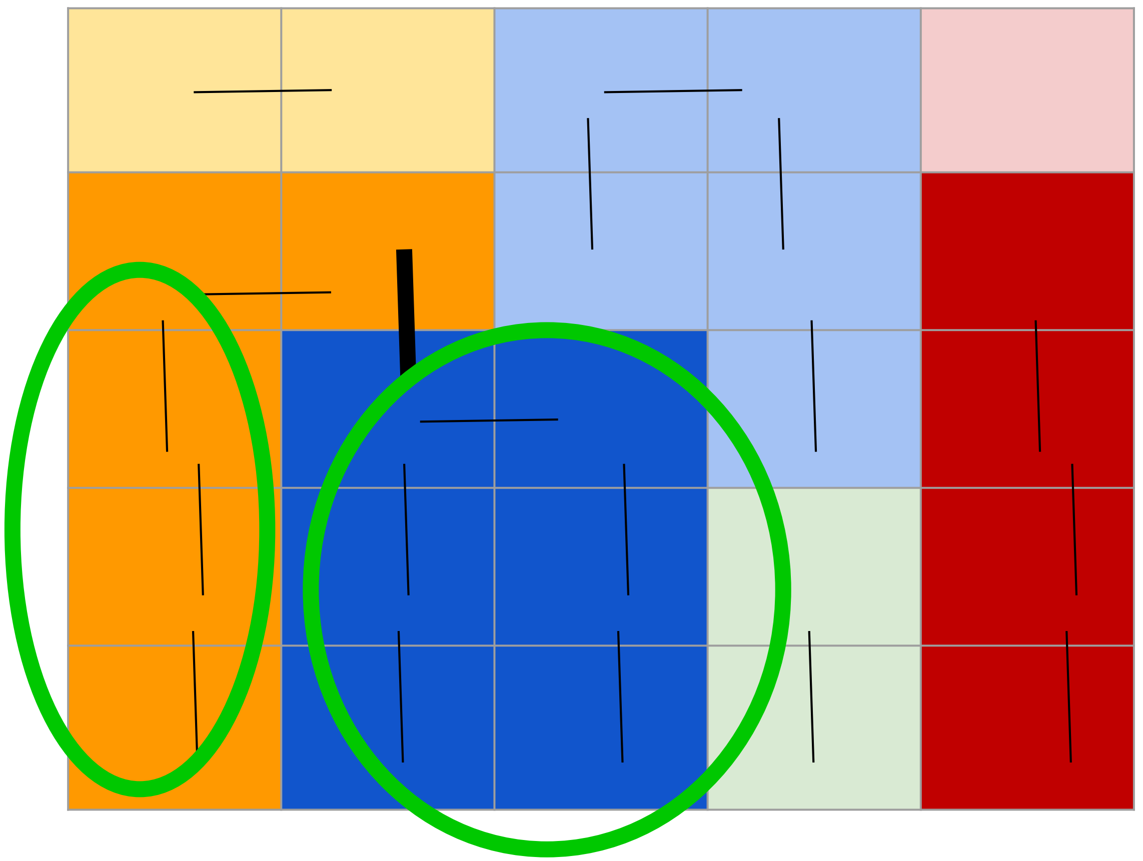
Why MST? Real Life Applications

Intuition: By the time we come around to looking at potentially border crossing edges, we can check how it compares against our existing components



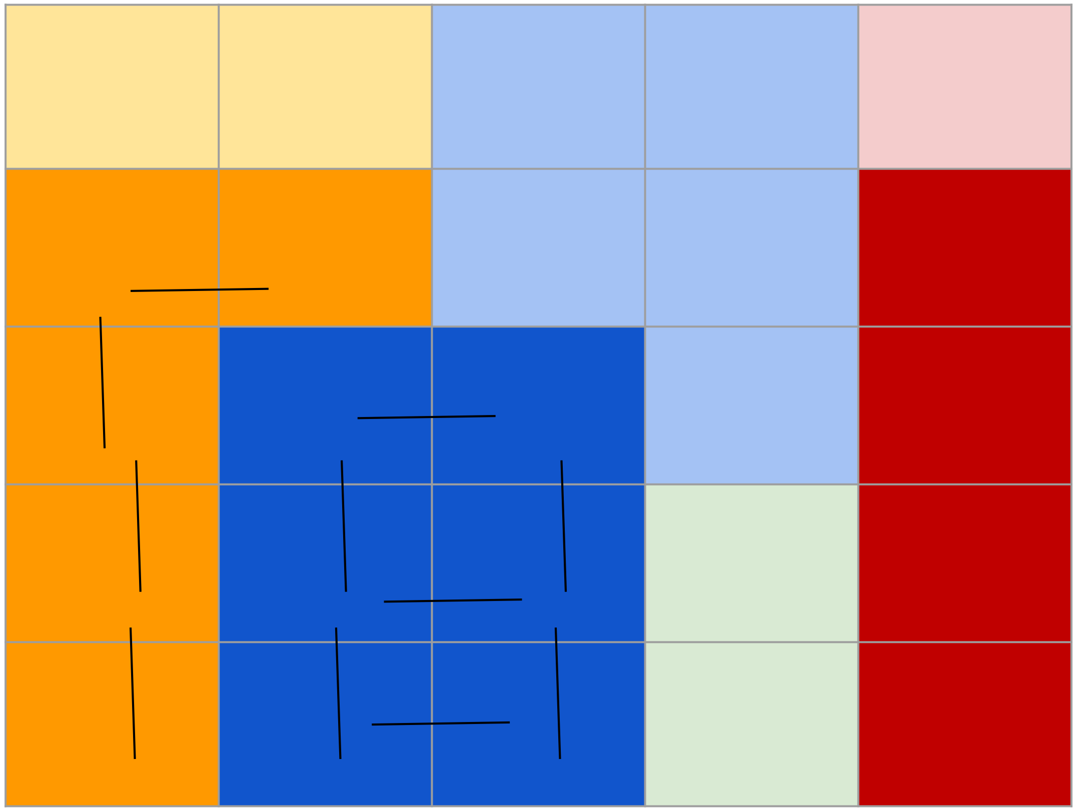
Why MST? Real Life Applications

Intuition: Too large? Keep the two components segmented from each other!



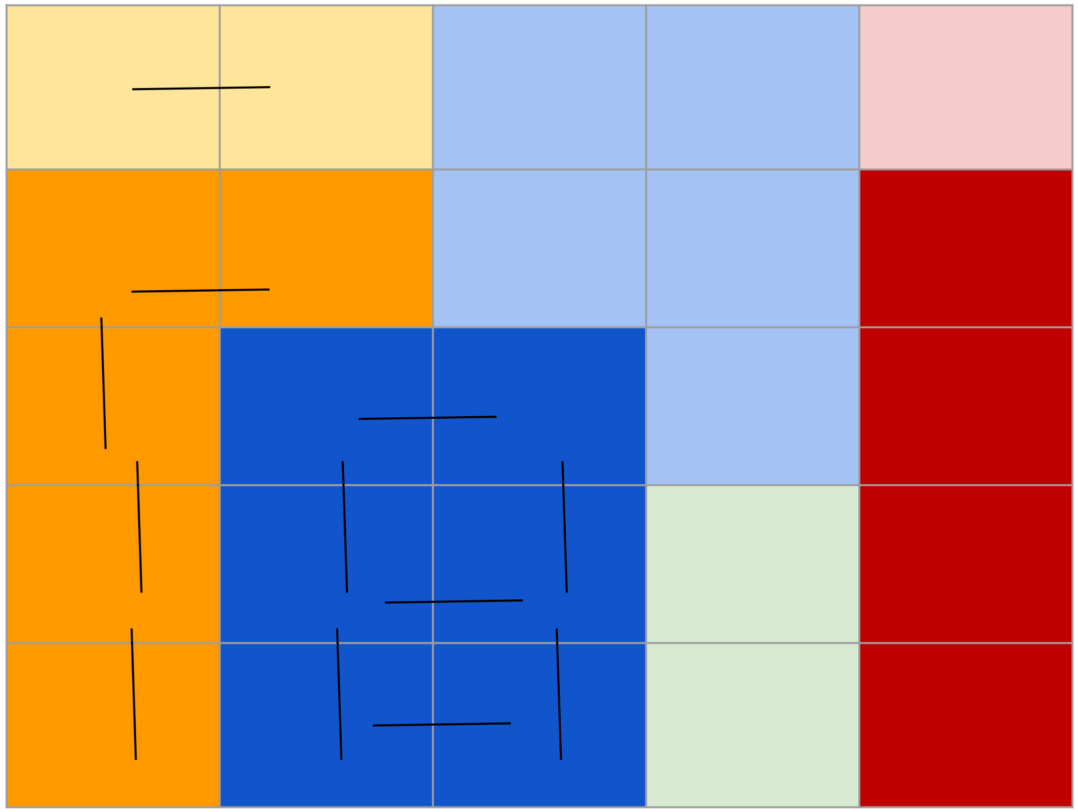
Why MST? Real Life Applications

Intuition: I.e. We don't merge the two regions.



Why MST? Real Life Applications

Intuition: If the if the difference is small, we merge the regions instead.

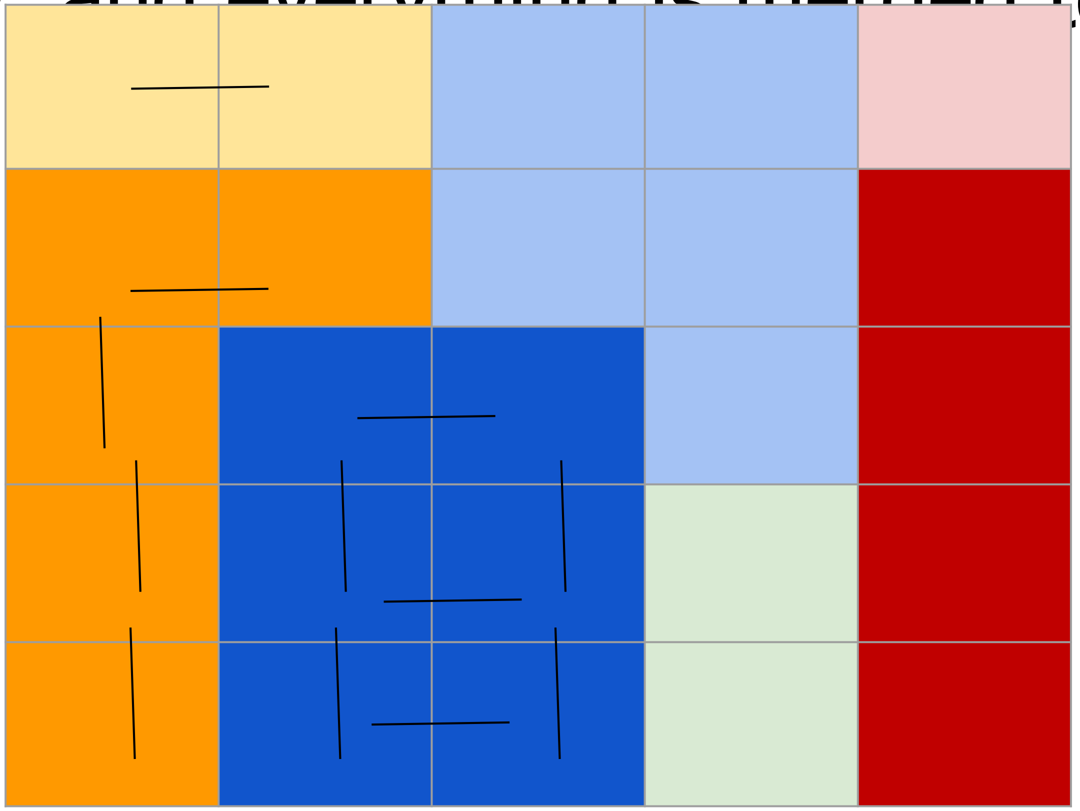


Why MST? Real Life Applications

Setting different thresholds is done by the user.

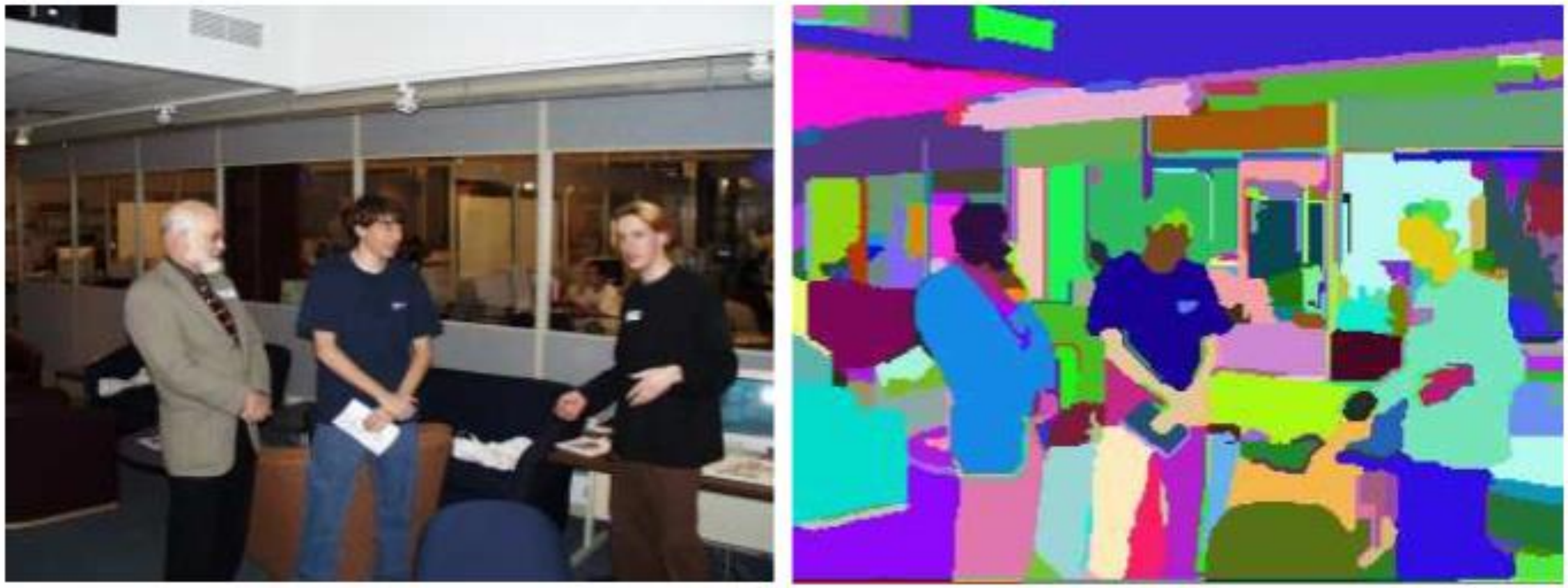
Too high of a threshold and nothing is merged.

Too low, and everything is merged together.



Felzenszwalb and Huttenlocher: 2004

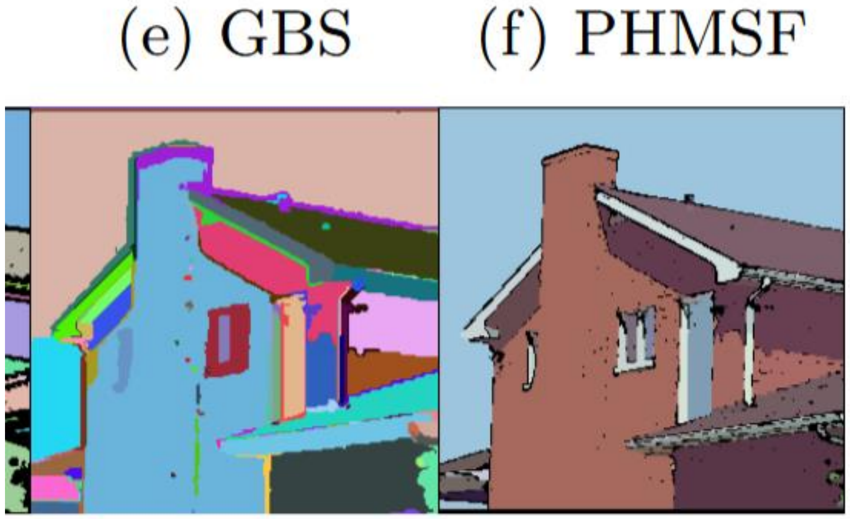
Efficient Graph-Based Image Segmentation (IJCV 2004)

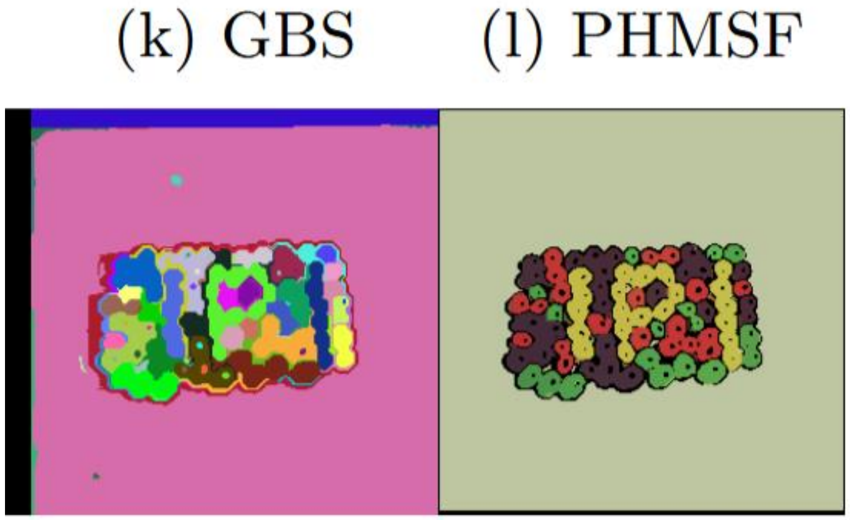


Wassenber, Middelmann, and Sanders: 2009

An Efficient Parallel Algorithm for Graph-Based Image Segmentation

(CAIP 2009)





Minimum Spanning Tree Summary

Classic greedy algorithms: O(E log V)

- Prim's (Priority Queue)
- Kruskal's (Union-Find)
- Boruvka's

Best known: O(m α(m, n))

- Deterministic: Chazelle (2000) O(E α(E, V))
- Randomized: Karger, Klein, Tarjan (1995) O(E)

Holy grail and major open problem: O(E) without randomization

Roadmap

So far:

Minimum Spanning Trees

– Prim’s Algorithm
– Kruskal’s Algorithm

Roadmap

Wednesday:

- Dynamic Programming

CS2040S

Data Structures and Algorithms

All about minimum spanning trees…