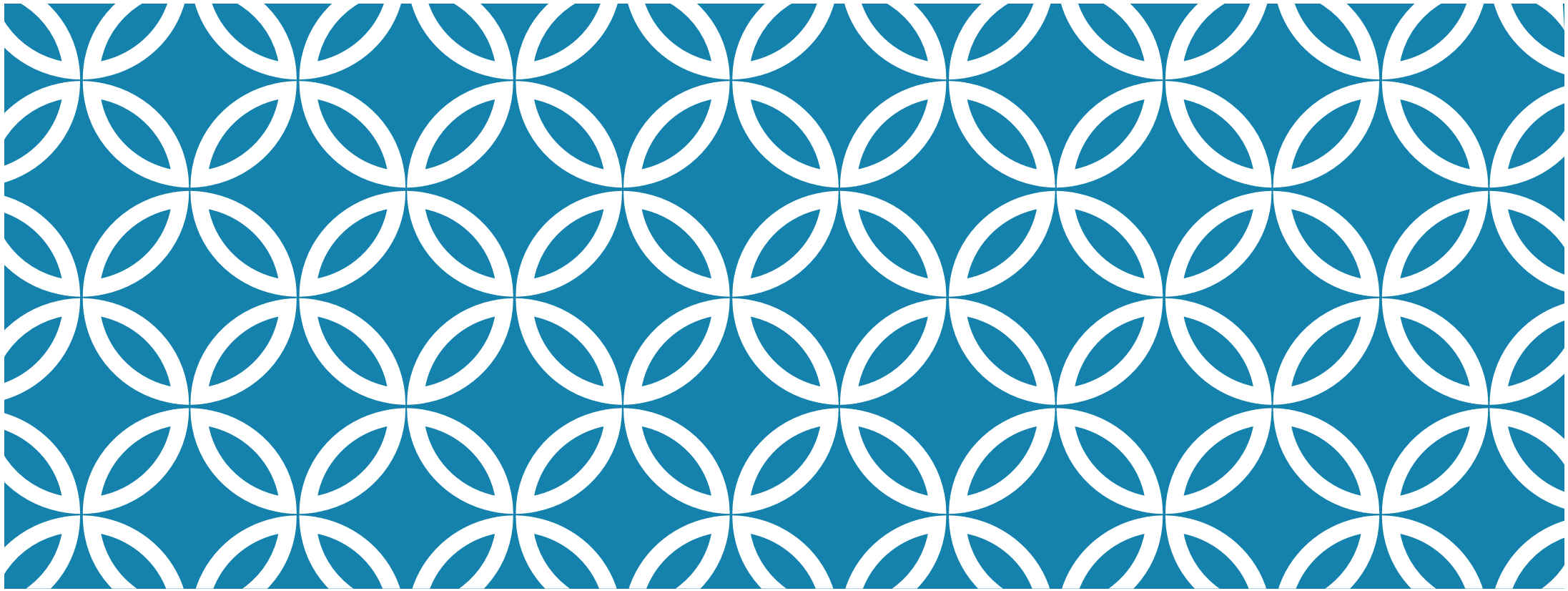


LECTURE 16: MINIMUM SPANNING TREES (MST)

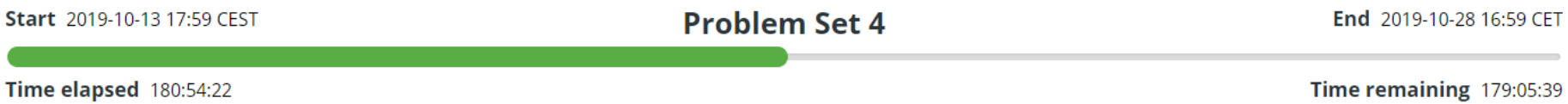
Harold Soh
harold@comp.nus.edu.sg

ADMINISTRATIVE ISSUES

Quiz 3 next week during Wed lecture

No lecture tomorrow (23rd Oct 2019)

PS4 is due end of the week.



MULTIPLICATION METHOD

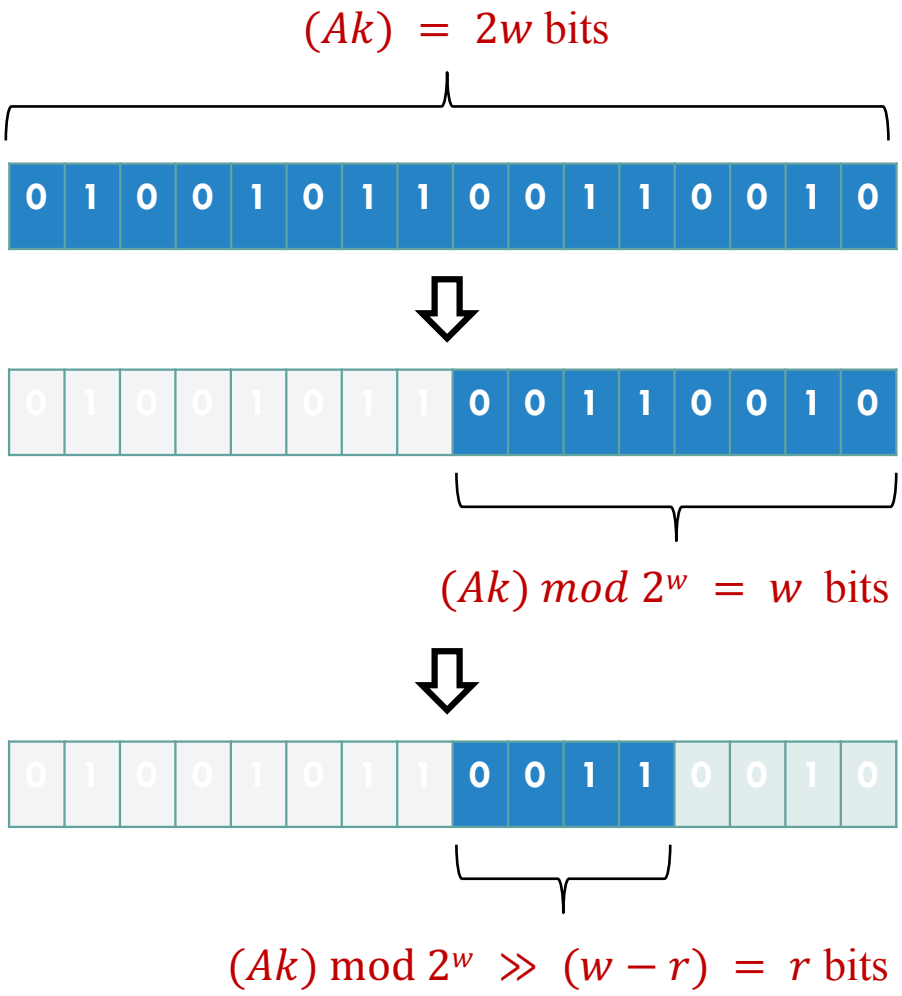
Fix

- table size: $m = 2^r$
- word size: w (size of a key in bits)
- constant: $2^{w-1} < A < 2^w$

Then:

$$h(k) = (Ak) \bmod 2^w \gg (w - r)$$

Why take top bits of lower half?



BOTTOM BITS ARE LESS INFORMATIVE

Rule of thumb: want to use all information in the key.

When you multiply $A * k$, the bottom bits contain less information about the key.

Multiplication in Binary:

	1 0 1 1	this is 11_{10}
×	1 0 1 0	this is 10_{10}
	0 0 0 0	1011×0
	1 0 1 1	1011×1, shifted one position to the left
	0 0 0 0	1011×0, shifted two positions to the left
+	1 0 1 1	1011×1, shifted three positions to the left
	1 1 0 1 1 1 0	this is 110_{10}

```
  1 0 1          0 1 1 0 0 1 0 1
× 8 9          × 0 1 0 1 1 0 0 1
-------        -----------------
  9 0 9          0 1 1 0 0 1 0 1
8 0 8        0 1 1 0 0 1 0 1
-------    0 1 1 0 0 1 0 1
8 9 8 9    -----------------------
           0 0 1 0 0 0 1 1 0 0 0 1 1 1 0 1
```

https://www.sciencedirect.com/topics/engineering/binary-multiplication

QUESTIONS?

Poll Everywhere

https://bit.ly/2LvG9bq

LEARNING OUTCOMES

By the end of this session, students should be able to:

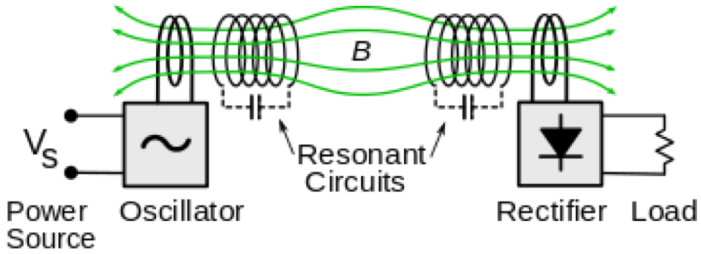
- Explain the **Minimum Spanning Tree (MST)** and its **properties.**
- Describe **Prim's algorithm** and its **running time.**

PROBLEM: CONNECT ME UP!

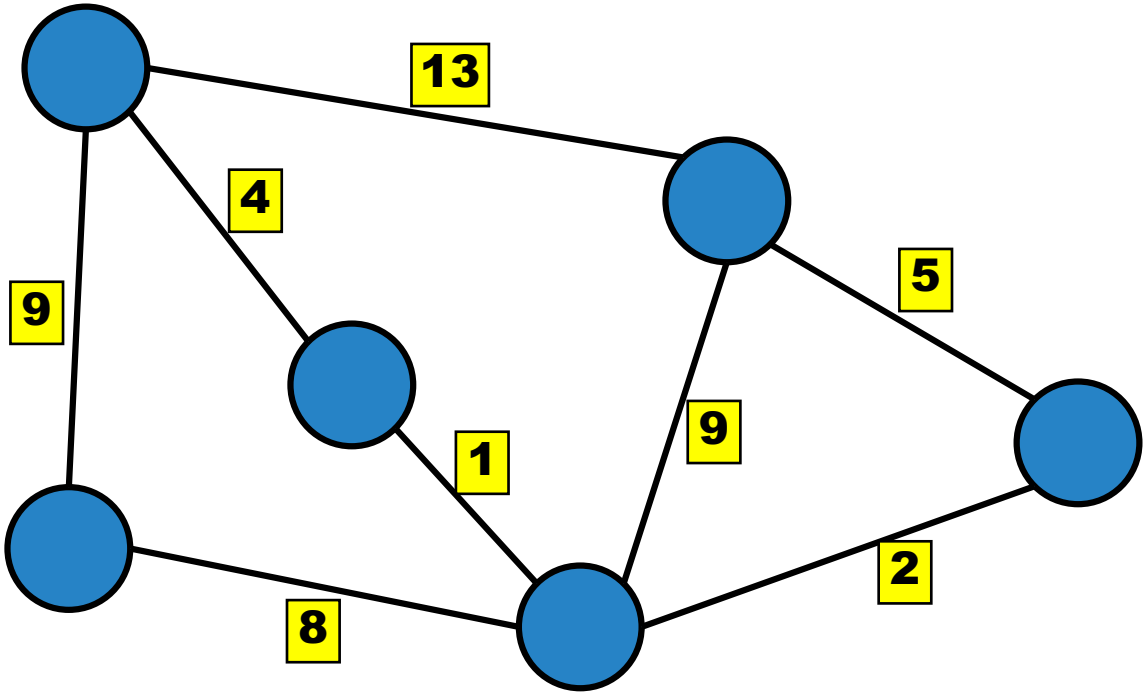
Singaporia has recently rolled out wireless power towers.

Connecting up the transmission towers is not easy (and expensive).

Naruto and you are hired to find a way to connect up the towers to minimize the cost.



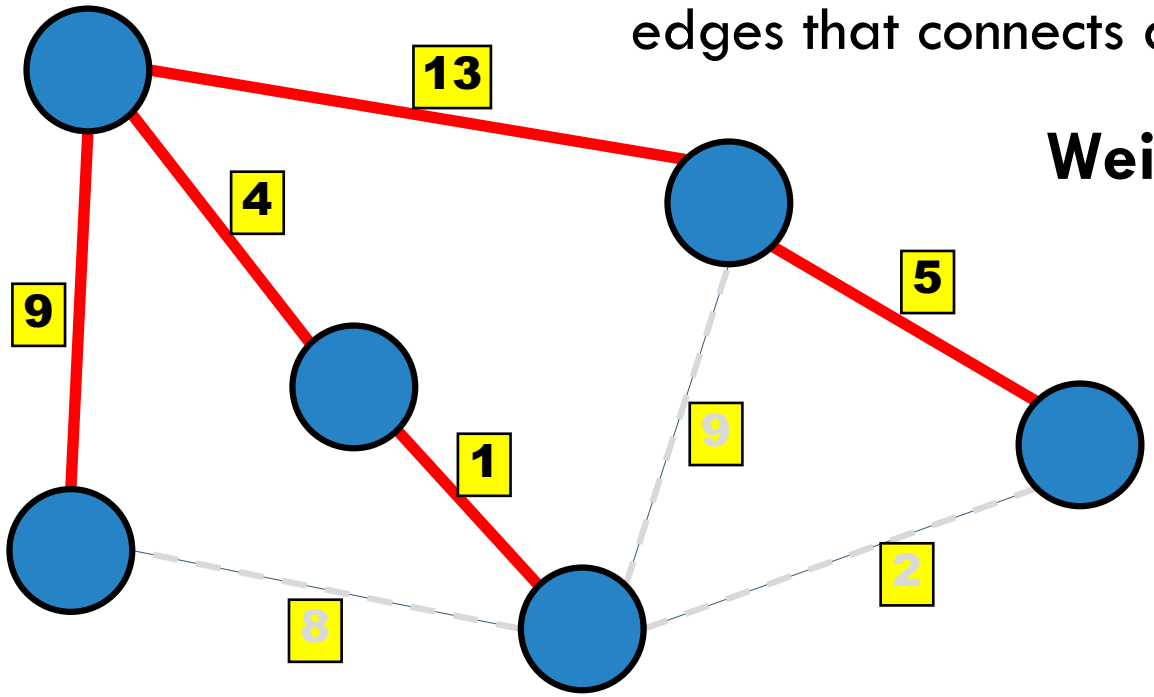
WE HAVE THE FOLLOWING GRAPH



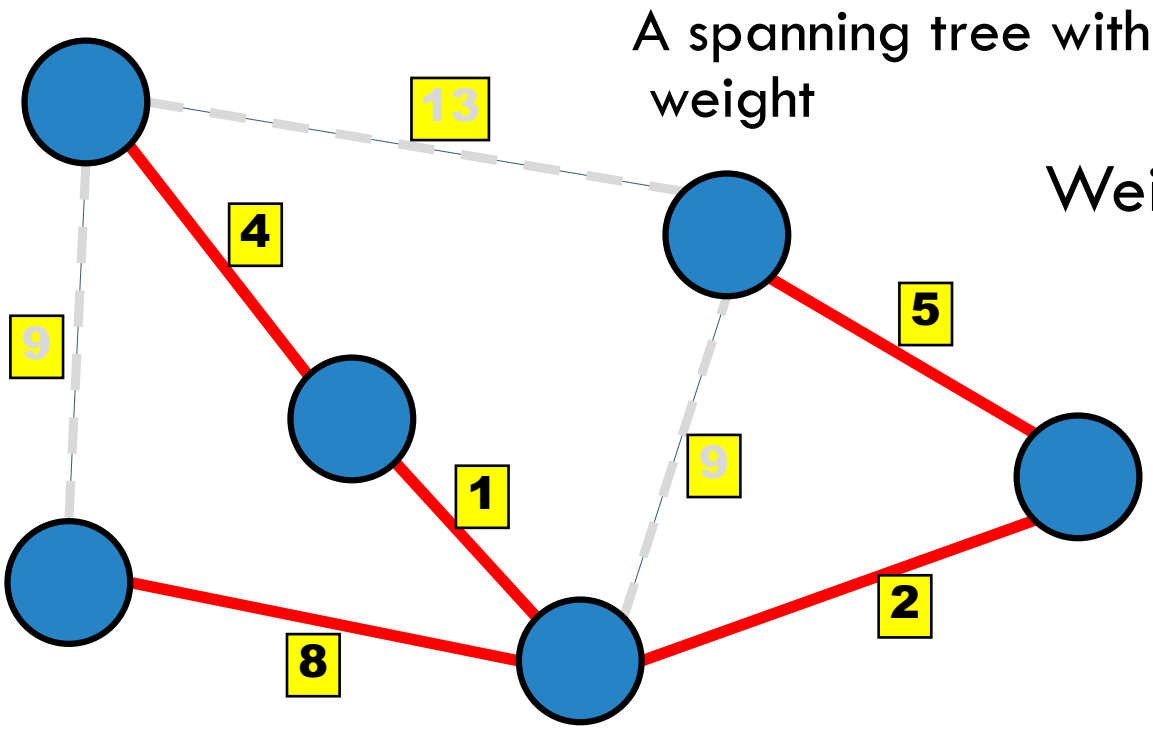
SPANNING TREE: DEFINITION

A **spanning tree** is an acyclic subset of the edges that connects all nodes

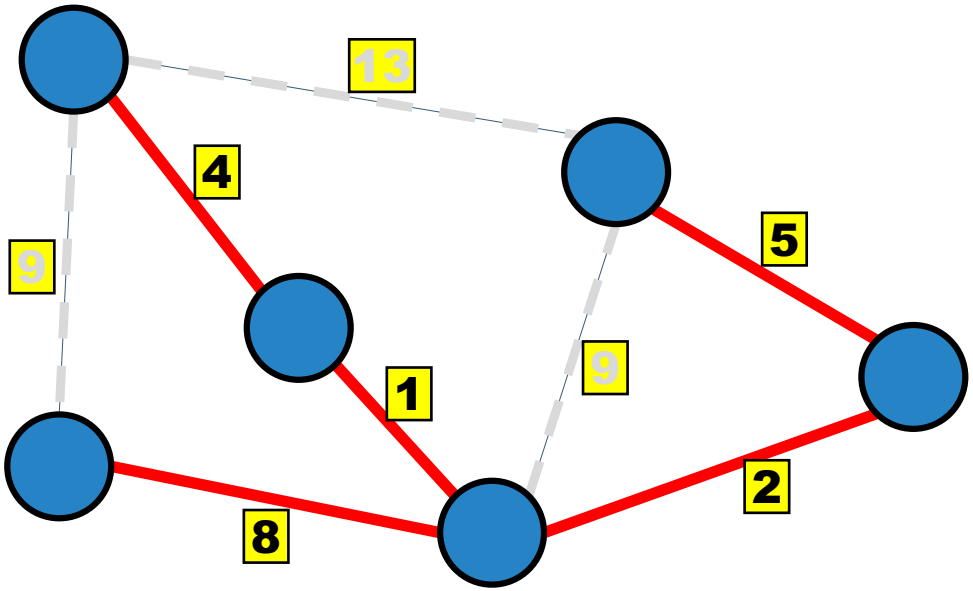
Weight: 32



MINIMUM SPANNING TREE: DEFINITION



MINIMUM SPANNING TREE

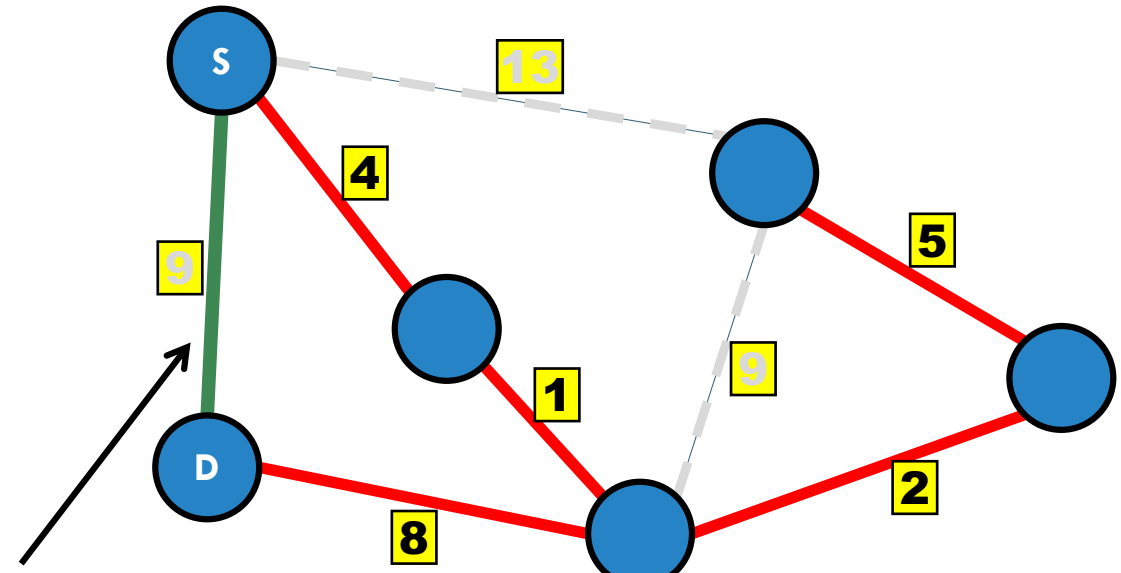


Can we use the MST to find the shortest paths?

A. Yes
B. Only on connected graphs
C. Only on dense graphs
D. Nope.
E. 

MINIMUM SPANNING TREE

MST is not the same as shortest paths



Shortest path from S to D

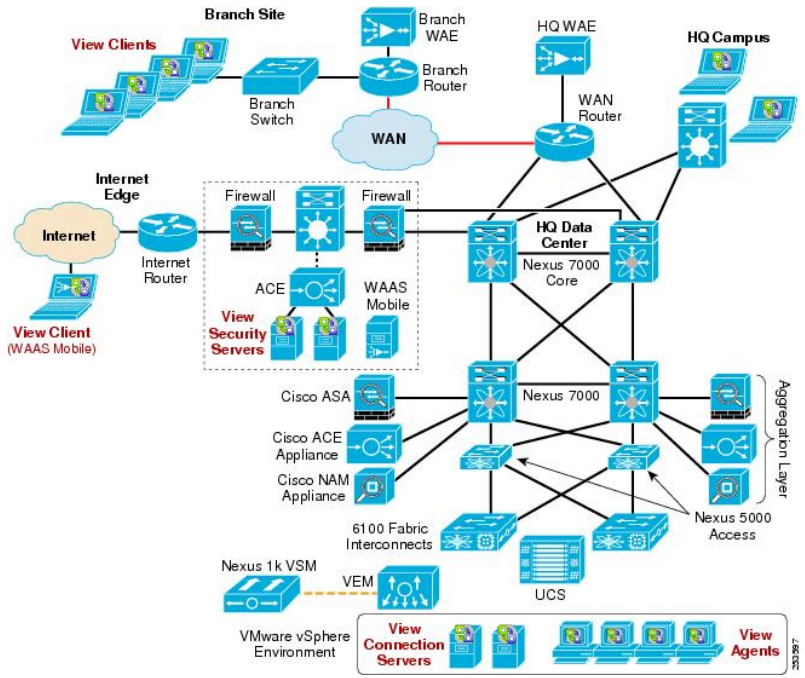
Can we use the MST to find the shortest paths?

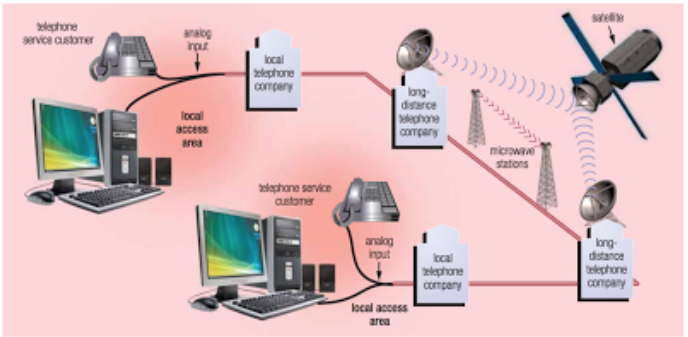
A. Yes
B. Only on connected graphs
C. Only on dense graphs
D. Nope.
E.

MST APPLICATIONS

Network design

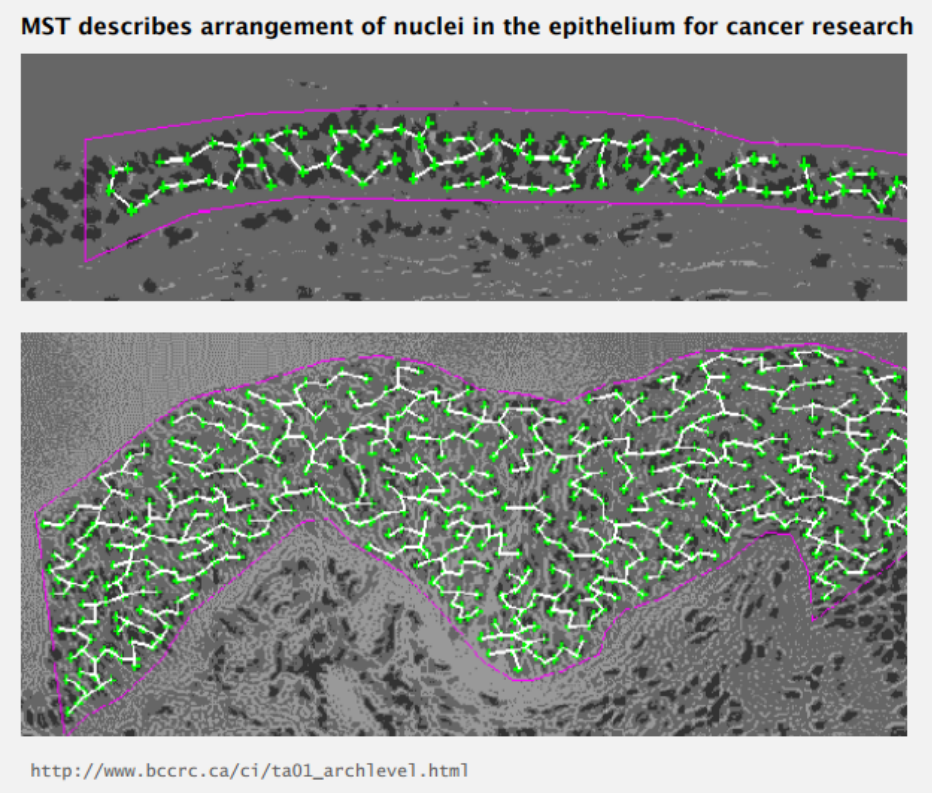
- Telephone networks
- Electrical networks
- Computer networks
- Ethernet autoconfig
- Road networks
- Bottleneck paths
- etc…





OTHER APPLICATIONS OF MSTs

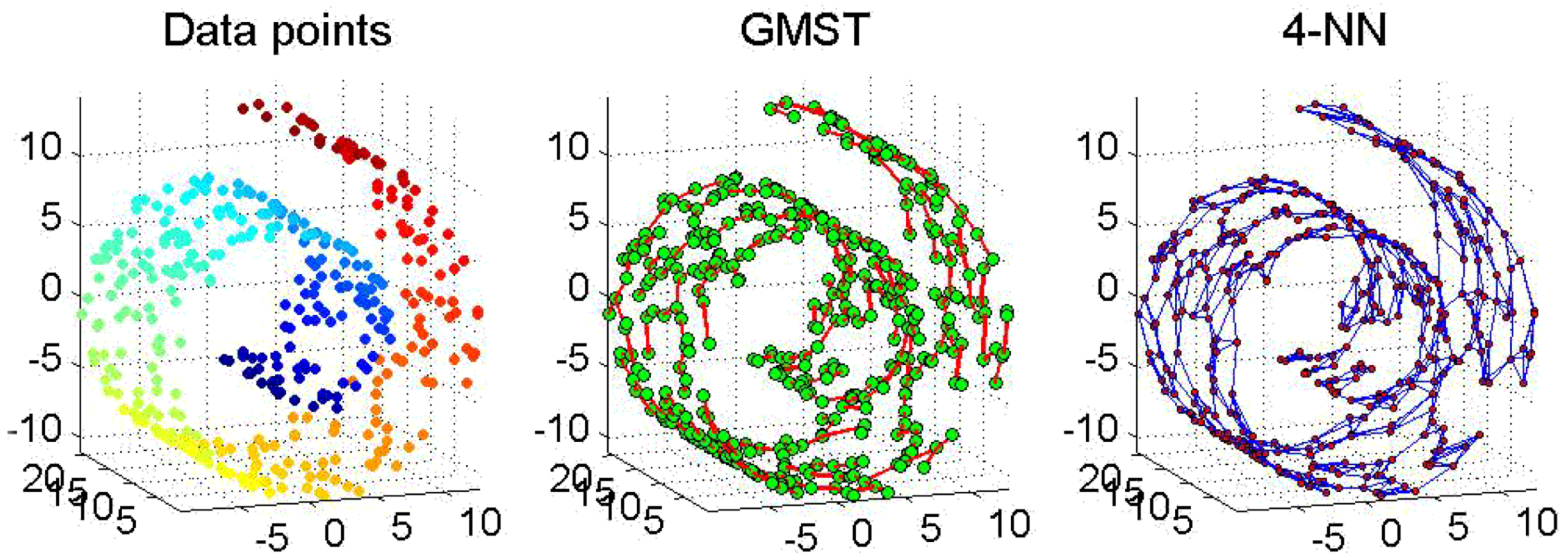
- Medical image processing
- Error correcting codes
- Face verification
- Cluster analysis
- …



[Examples from http://algs4.cs.princeton.edu/lectures/43MinimumSpanningTrees.pdf]

DISCOVERING STRUCTURES FOR HIGH DIMENSIONAL DATA

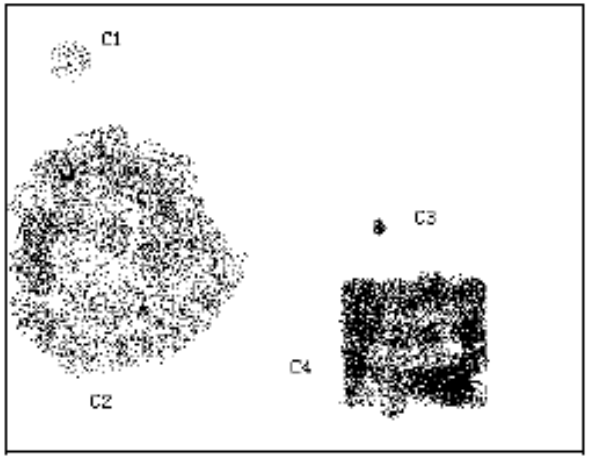
In machine learning, pattern recognition, data mining, etc

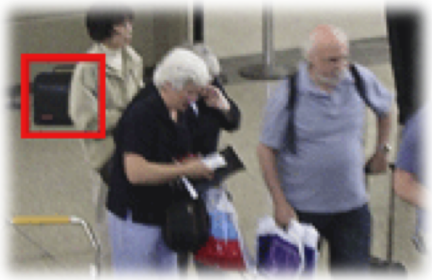


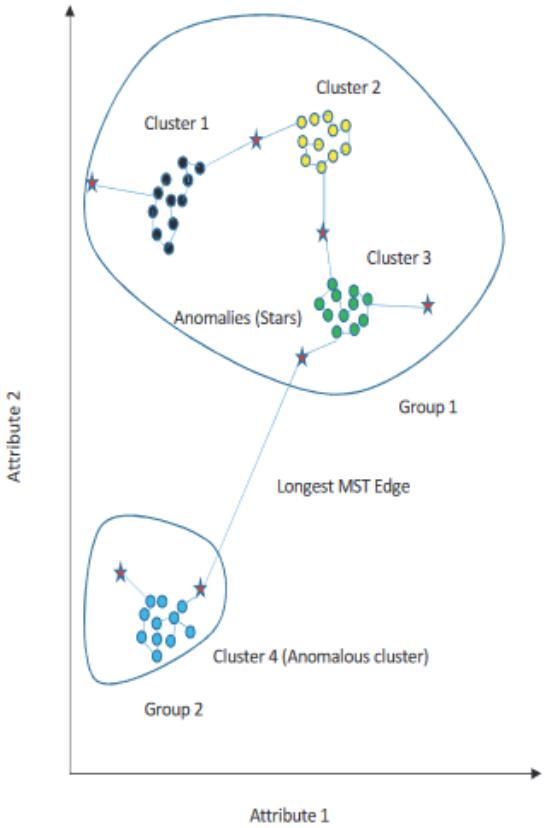
ANOMALY DETECTION

Anomaly is a pattern in the data that does not conform to the expected behavior.

E.g. Cyber intrusions, credit card fraud, air traffic safety

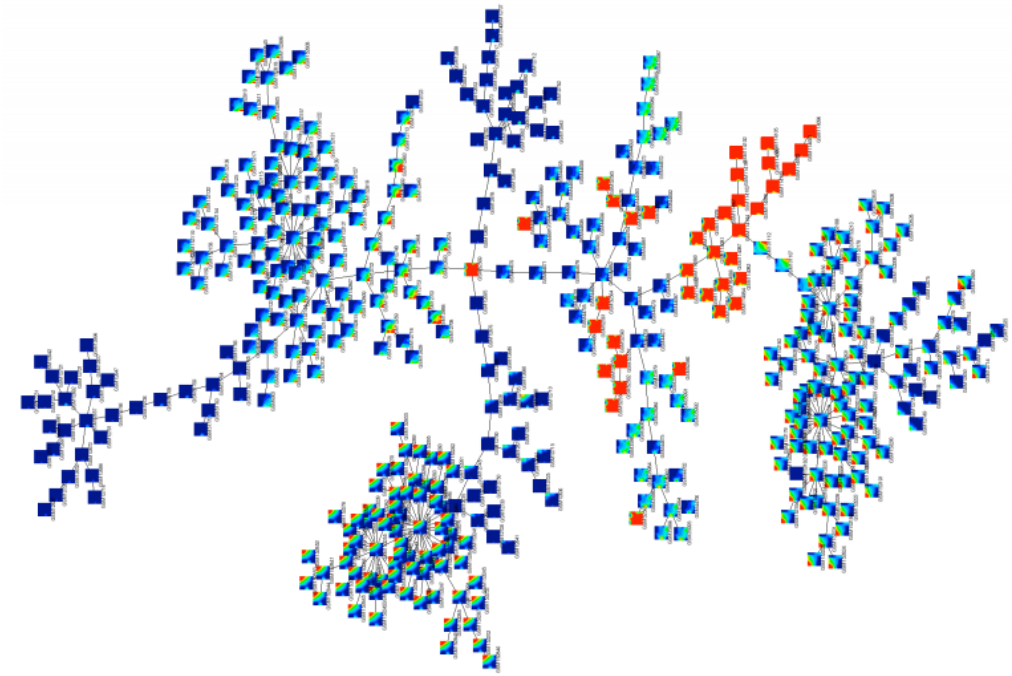






TODAY: MINIMUM SPANNING TREES

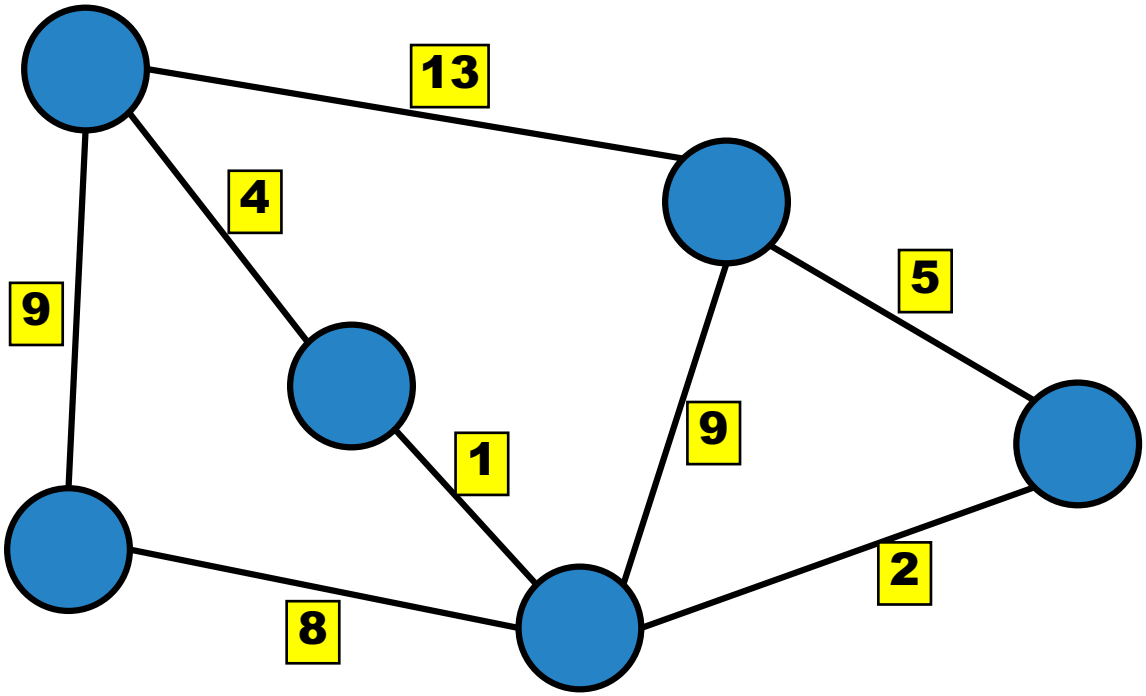
- What is a MST?
- **Basic Properties of an MST**
- Prim's Algorithm
- Kruskal's Algorithm
- Variations



SIMPLIFYING ASSUMPTIONS

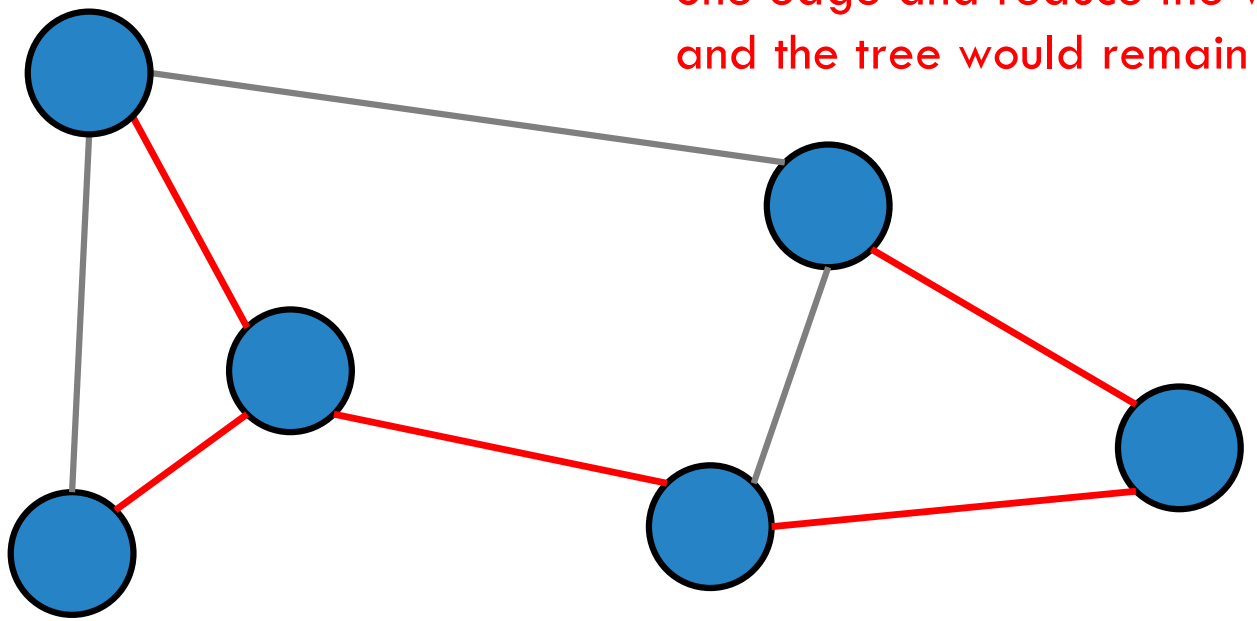
Edge weights are **distinct**.

Graph is **connected** and **undirected**.



MST PROPERTIES

Property 1: No Cycles

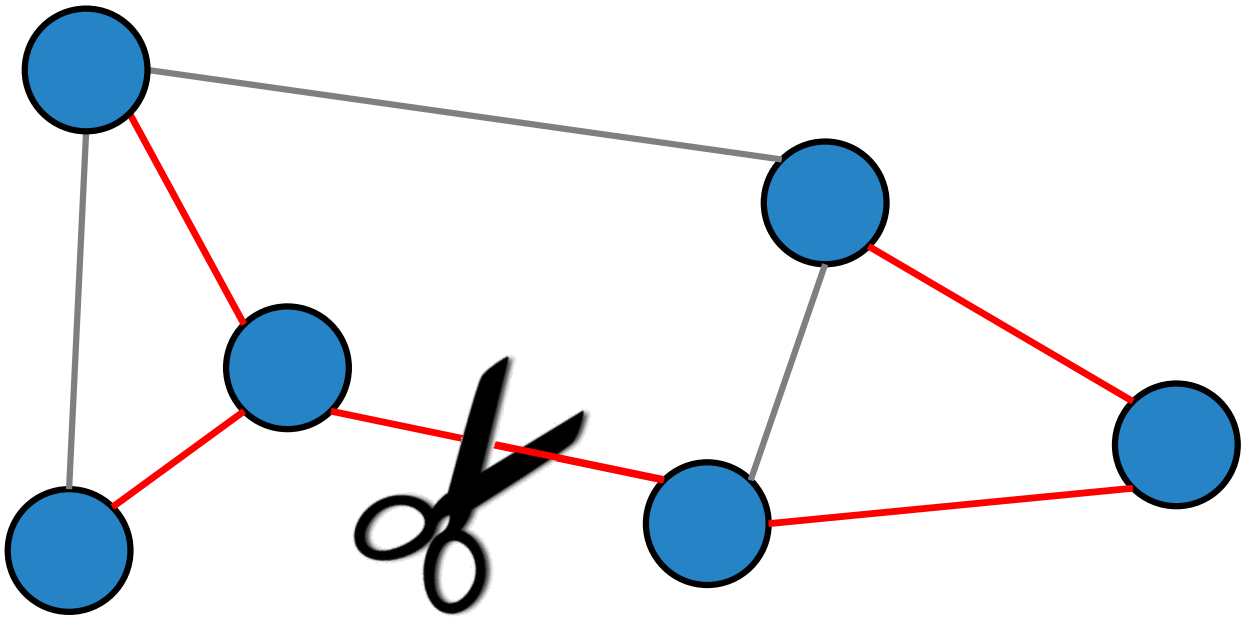


Why?

If there was a cycle, we could remove one edge and reduce the weight, and the tree would remain connected.

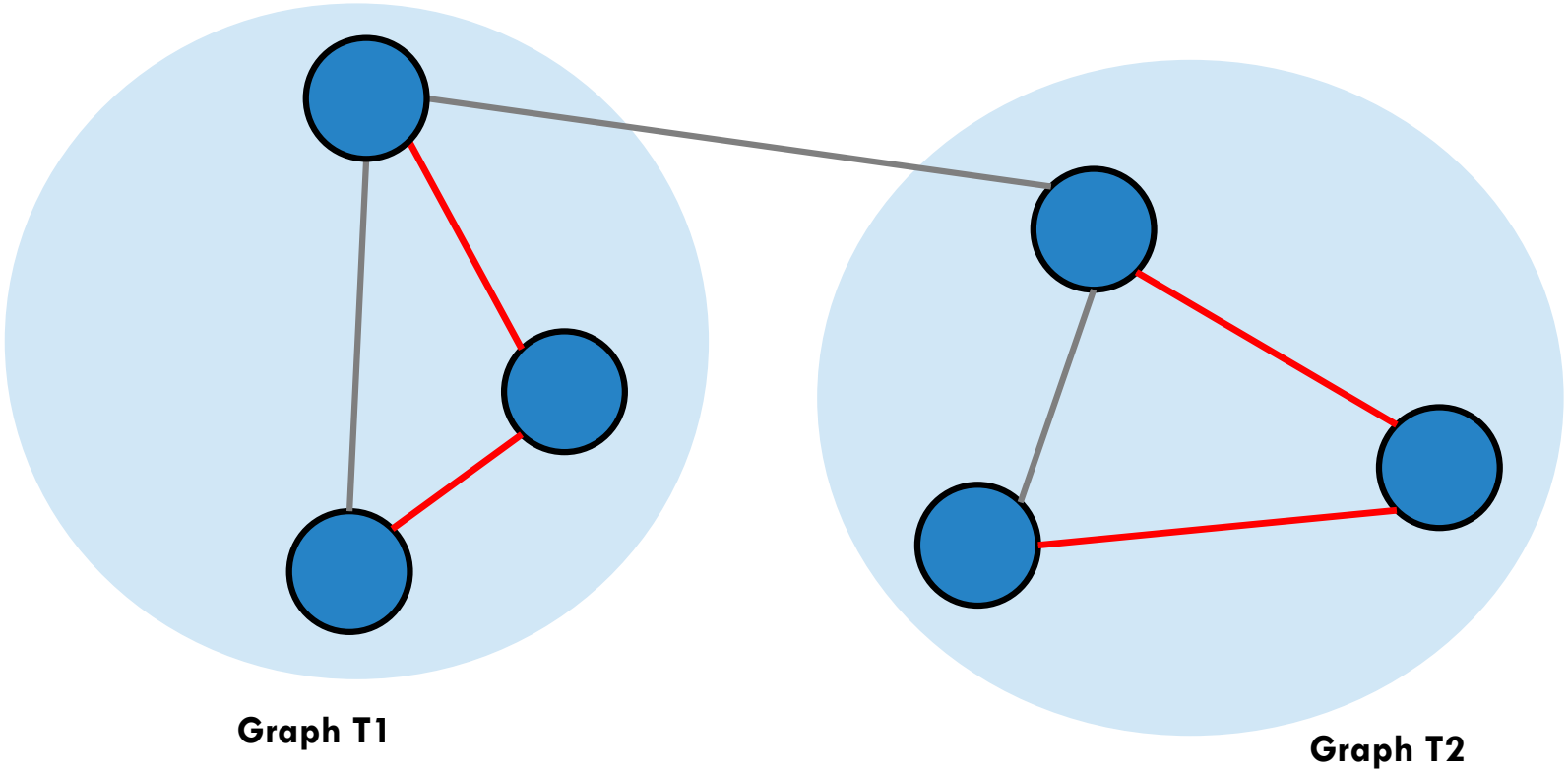
MST PROPERTIES

Question: What happens if you cut an MST into T1 and T2??



MST PROPERTIES

Question: What happens if you cut an MST into T1 and T2??

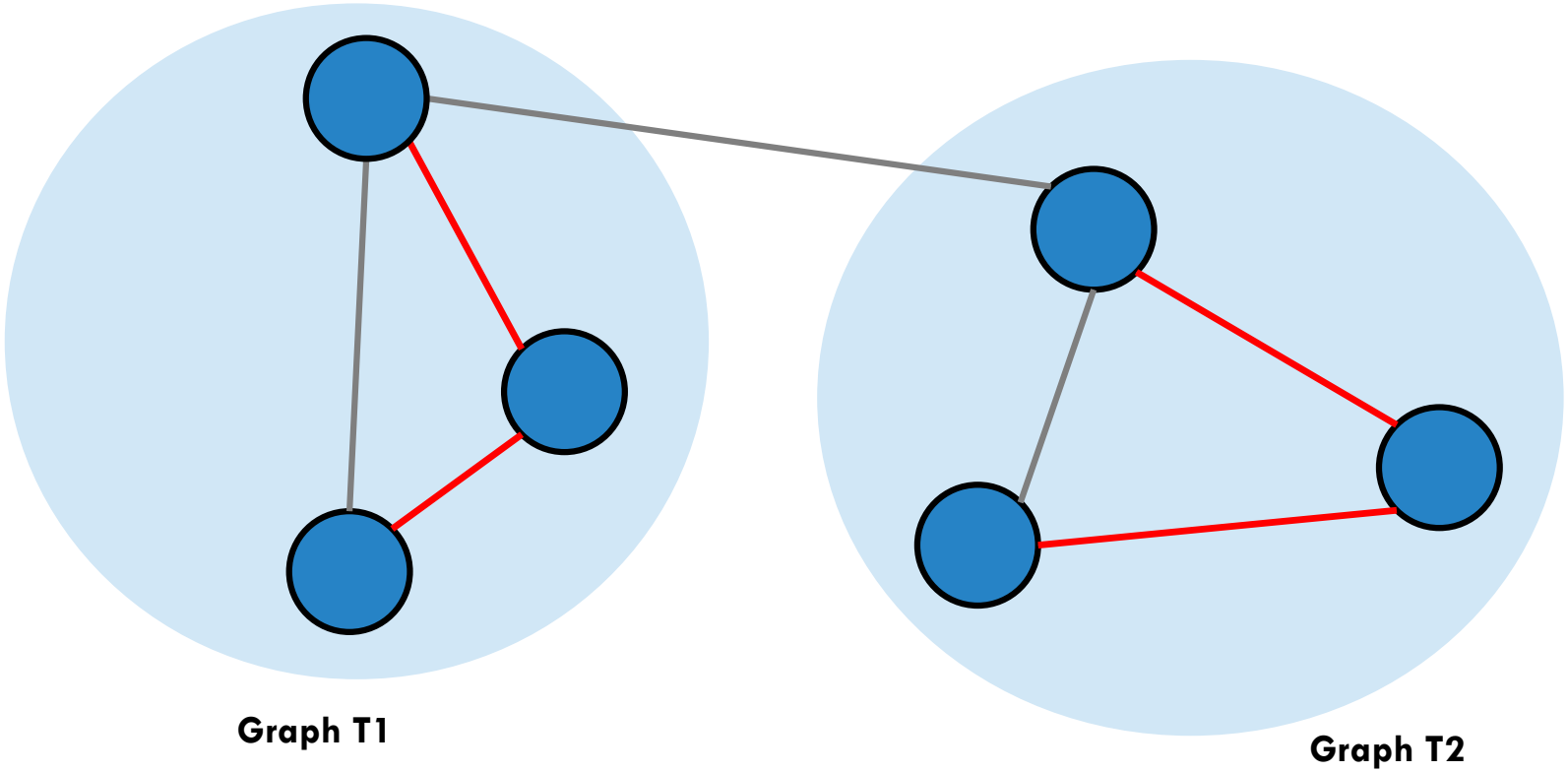


Graph T1

Graph T2

MST PROPERTIES

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

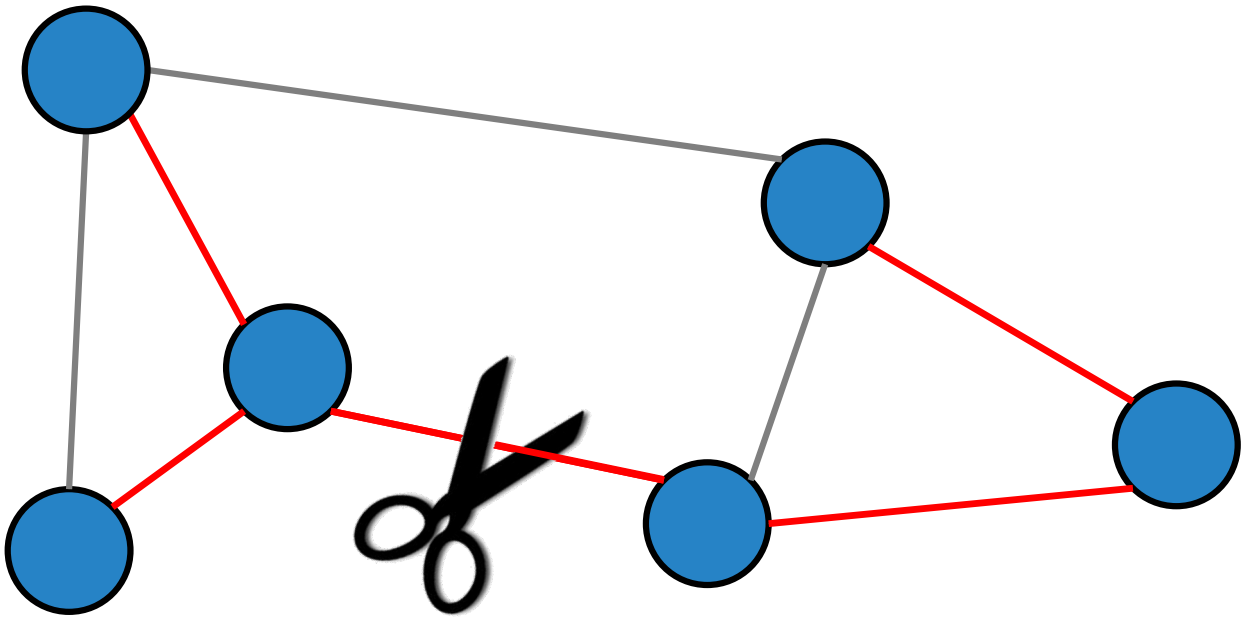


Graph T1

Graph T2

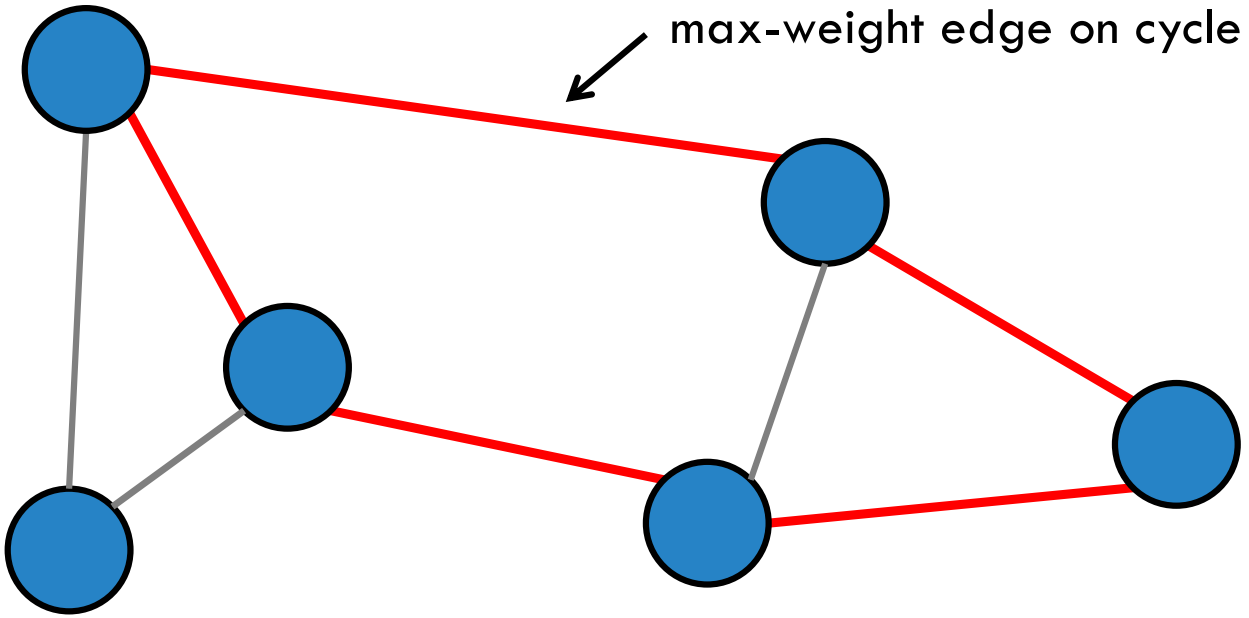
MST PROPERTIES

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



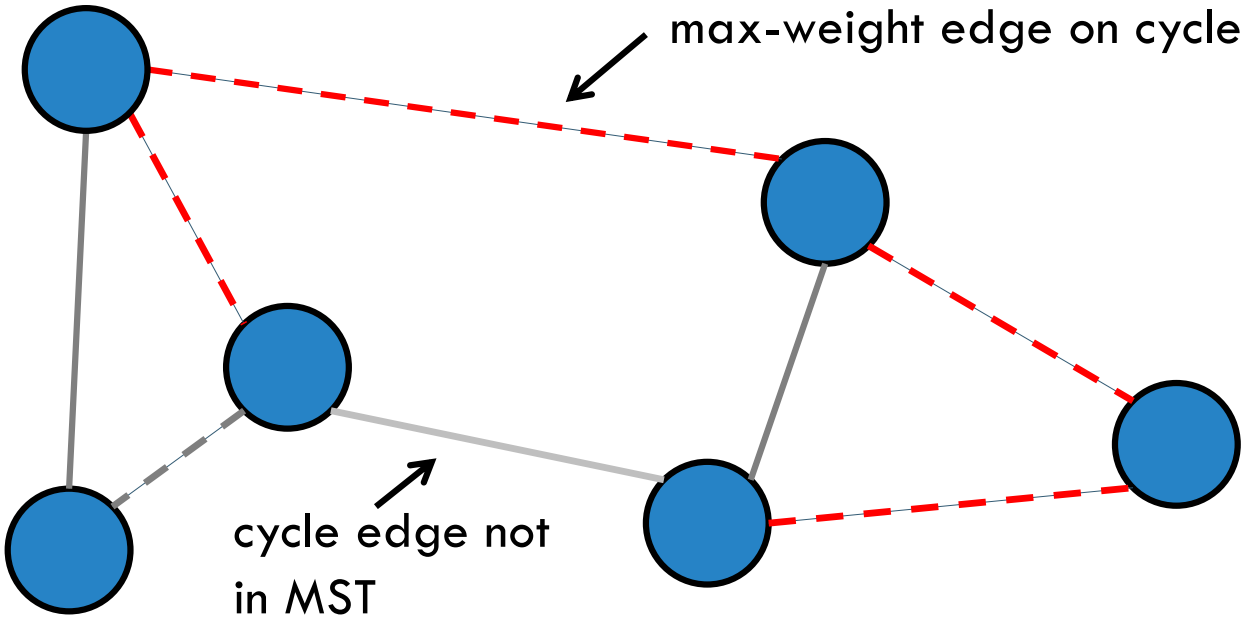
MST PROPERTIES

Property 3: Cycle Property; for every cycle, the maximum edge is **NOT** in the MST



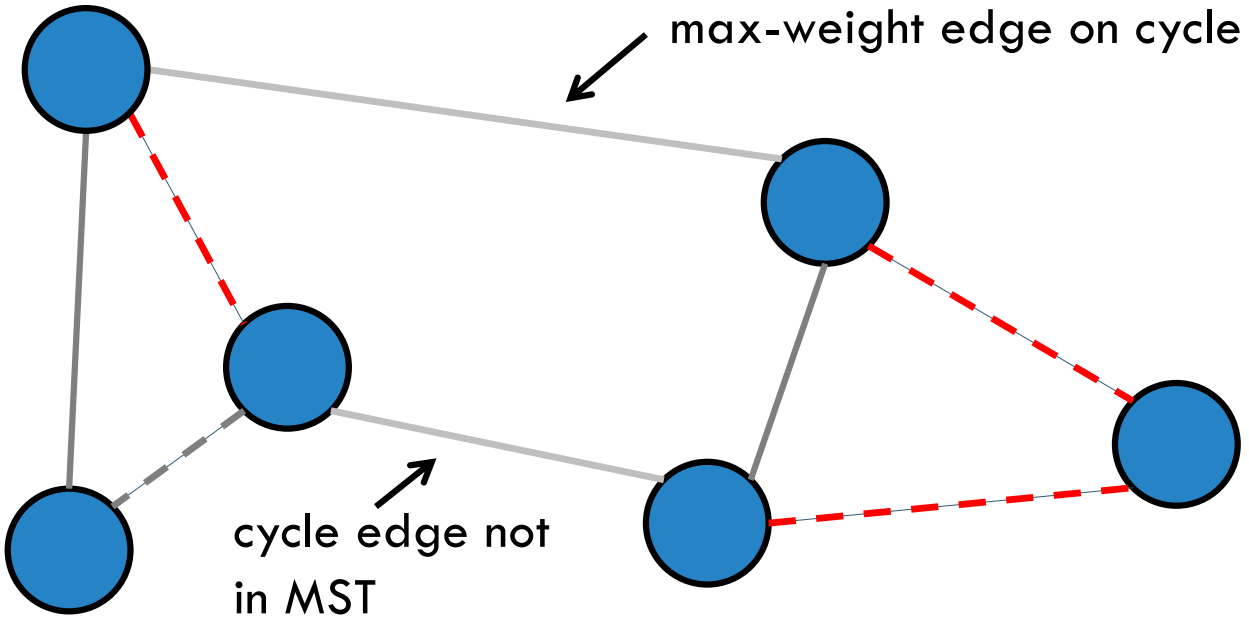
CYCLE PROPERTY: PROOF SKETCH
BY **CONTRADICTION**

Assume heavy edge is in the MST.



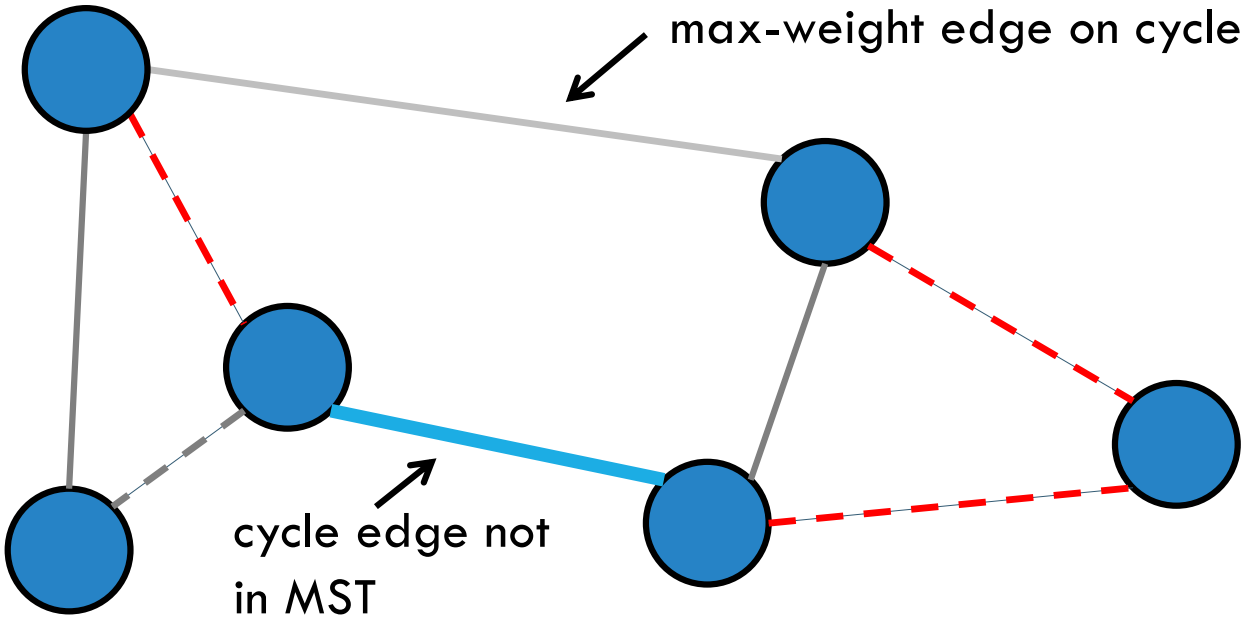
CYCLE PROPERTY: PROOF SKETCH
BY **CONTRADICTION**

Removing the max-weight edge
"cuts" the graph



CYCLE PROPERTY: PROOF SKETCH
BY **CONTRADICTION**

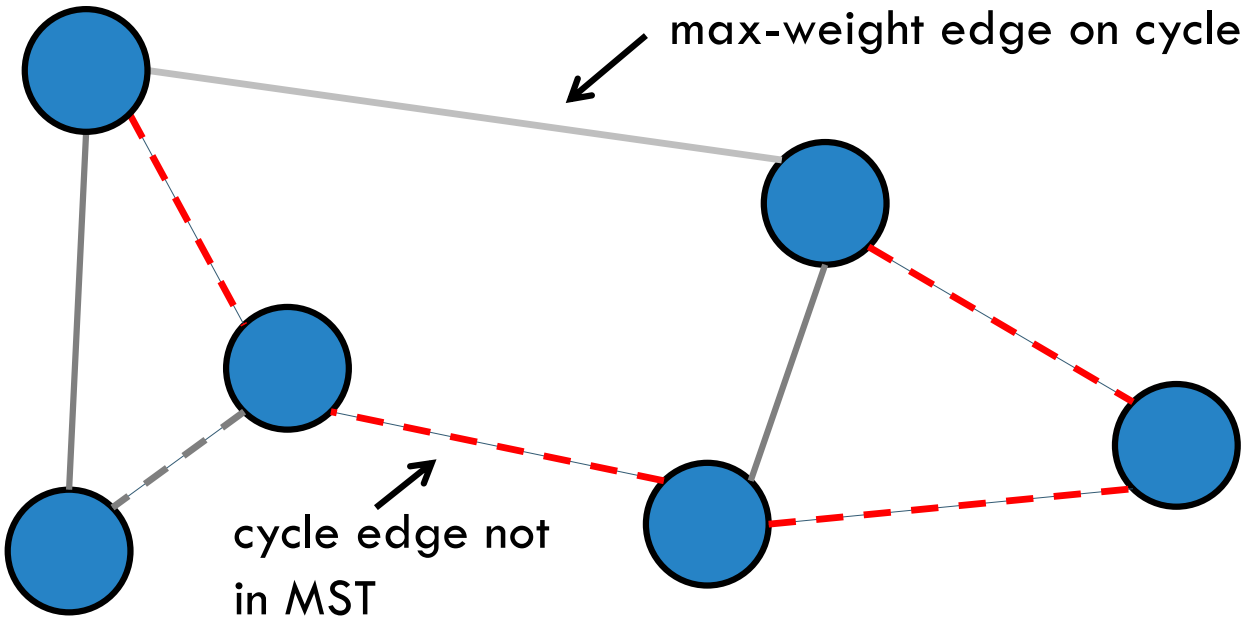
There exists another edge that
crosses this “cut” (because of cycle)



CYCLE PROPERTY: PROOF SKETCH
BY **CONTRADICTION**

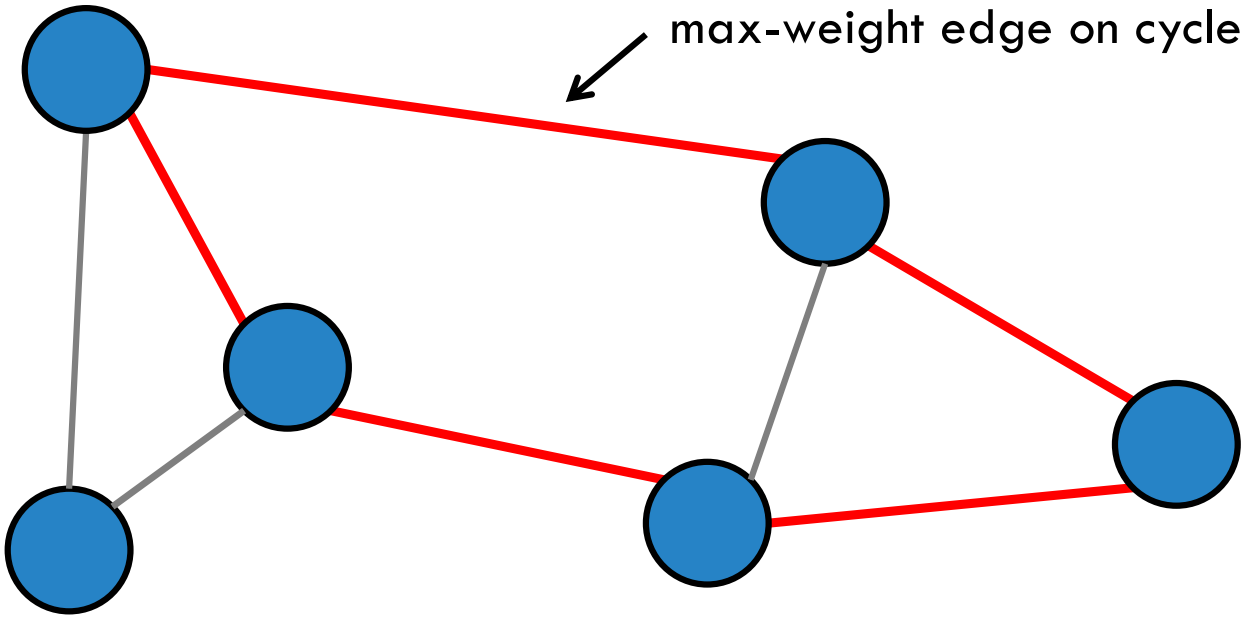
Add this edge to the MST. But now the MST has **less weight.**

The initial MST cannot be a MST! Contradiction! ■

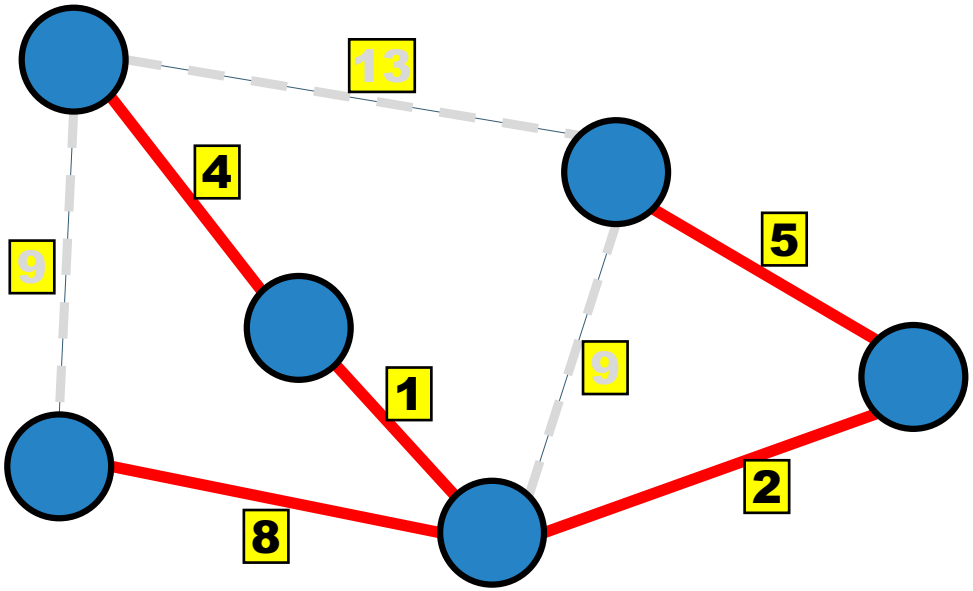


MST PROPERTIES

Property 3: Cycle Property; for every cycle, the maximum edge is **NOT** in the MST



HOW ABOUT THE MINIMUM WEIGHT EDGE IN A CYCLE?

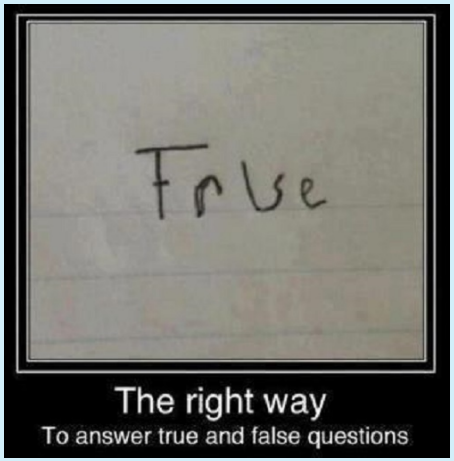


Poll Everywhere

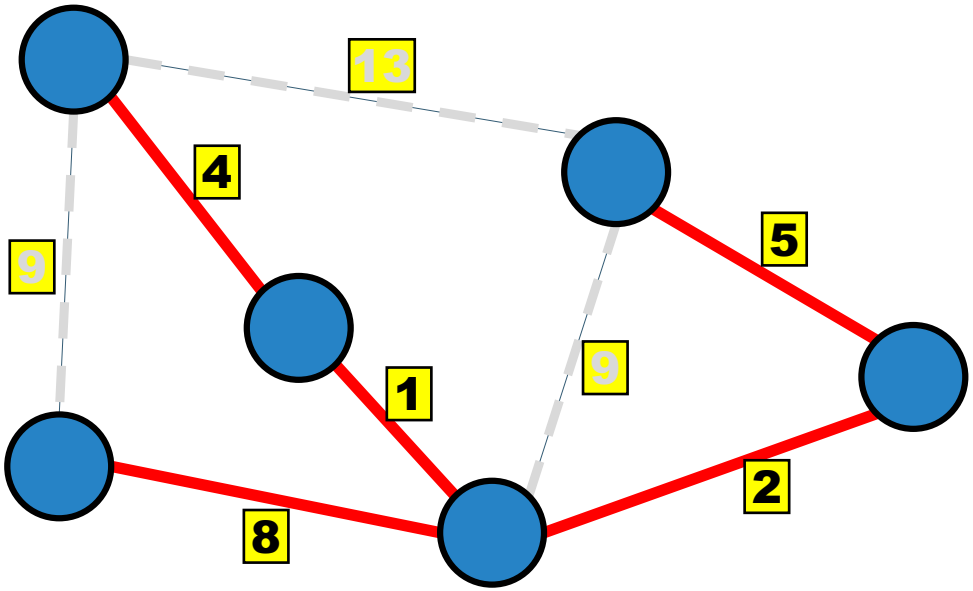
https://bit.ly/2LvG9bq

For every cycle, the minimum weight edge is always in the MST

A. True
B. False
C.



HOW ABOUT THE MINIMUM WEIGHT EDGE IN A CYCLE?



Poll Everywhere

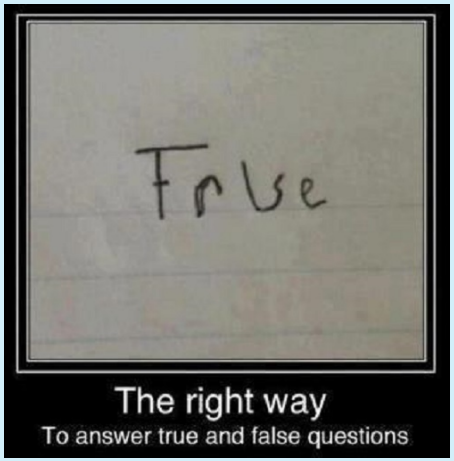
https://bit.ly/2LvG9bq

For every cycle, the minimum weight edge is always in the MST

A. True

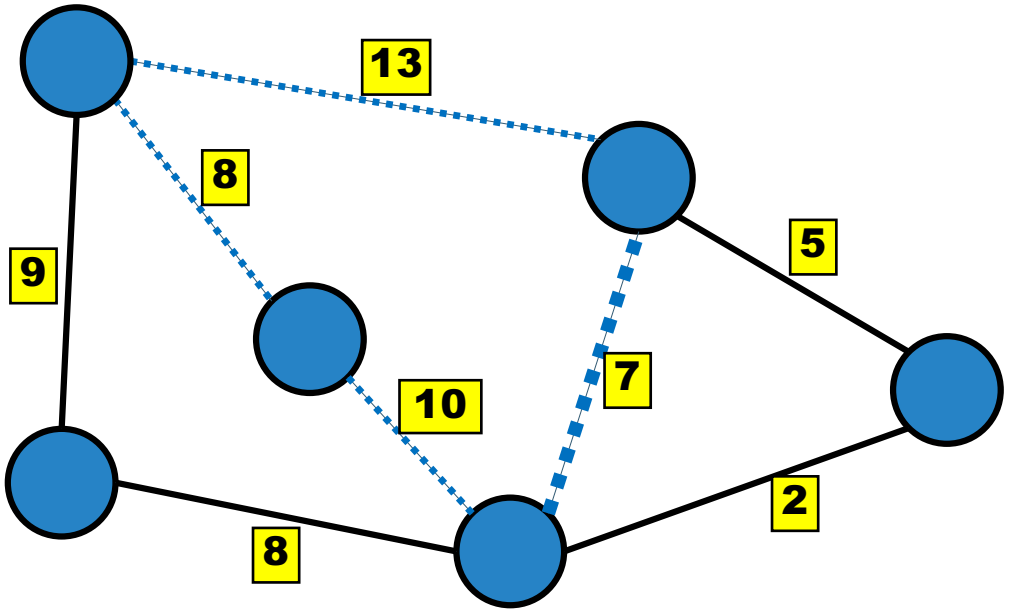
B. False

C.



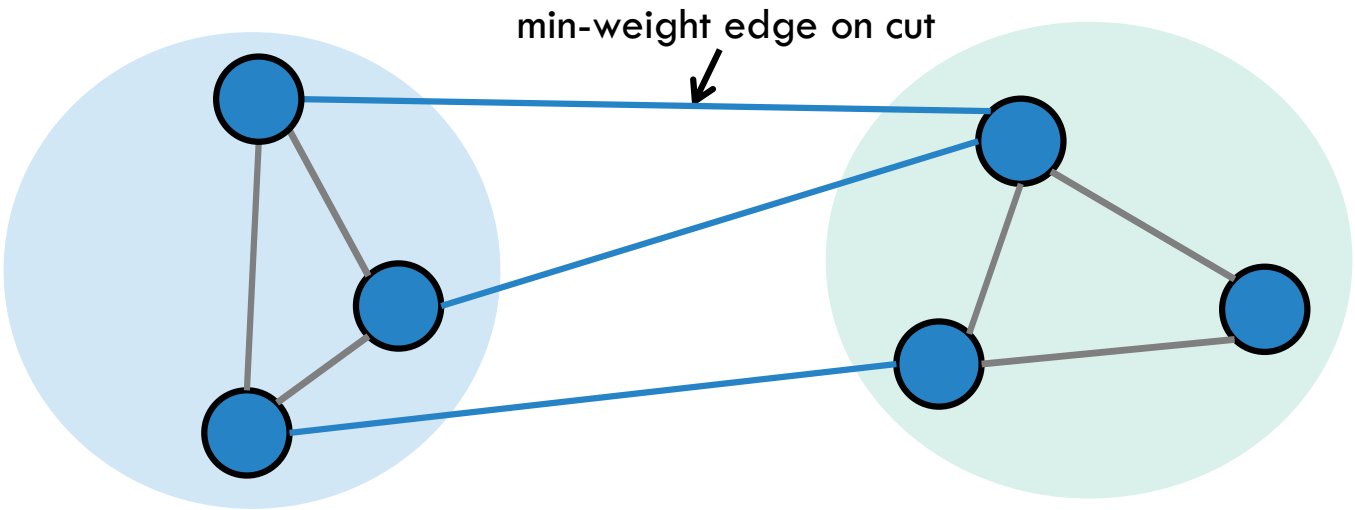
MST PROPERTIES

Watch out! For every cycle, the minimum weight edge **<u>may or may not</u>** be in the MST



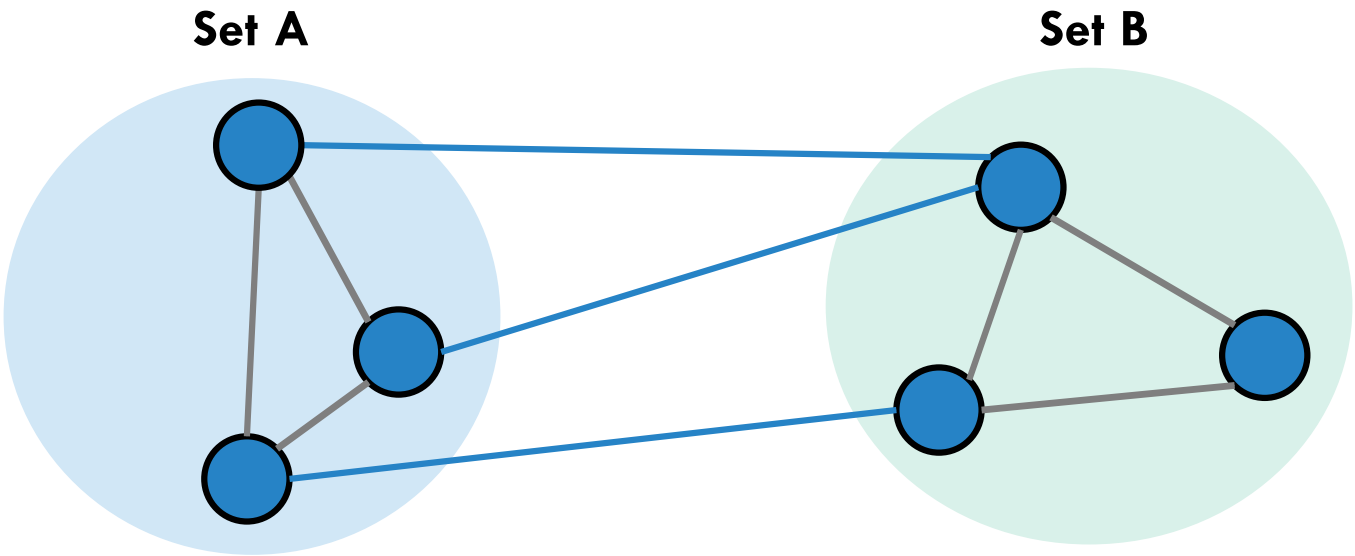
MST PROPERTIES

Property 4: Cut Property; for every partition of nodes, the **minimum weight edge across the cut** is in the MST.



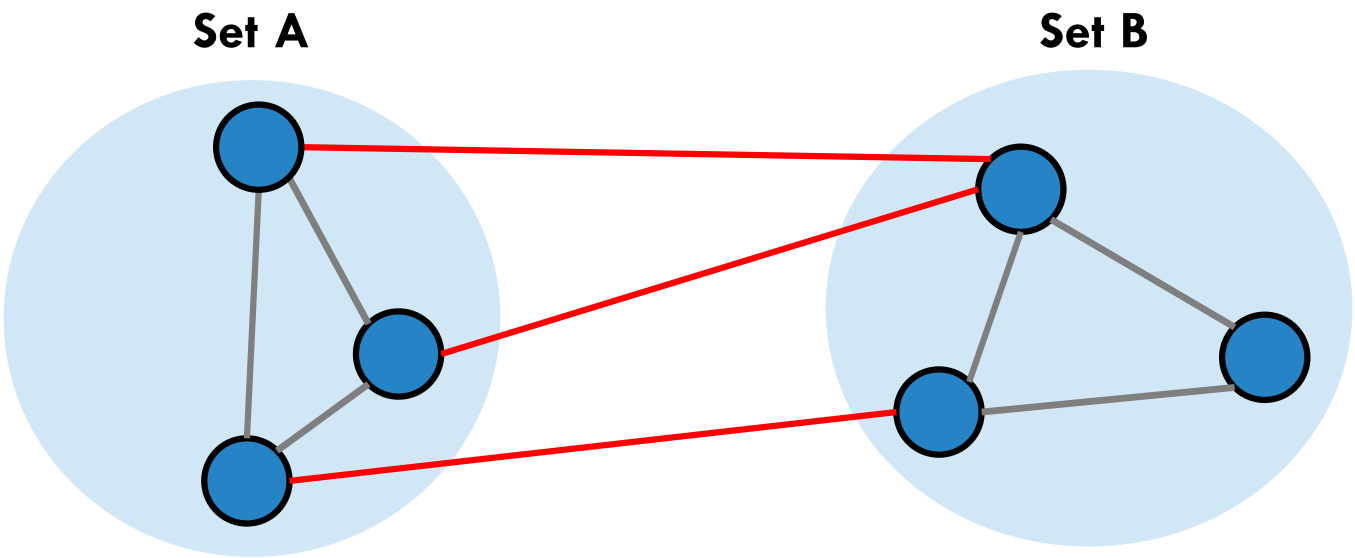
WHAT IS A CUT?

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



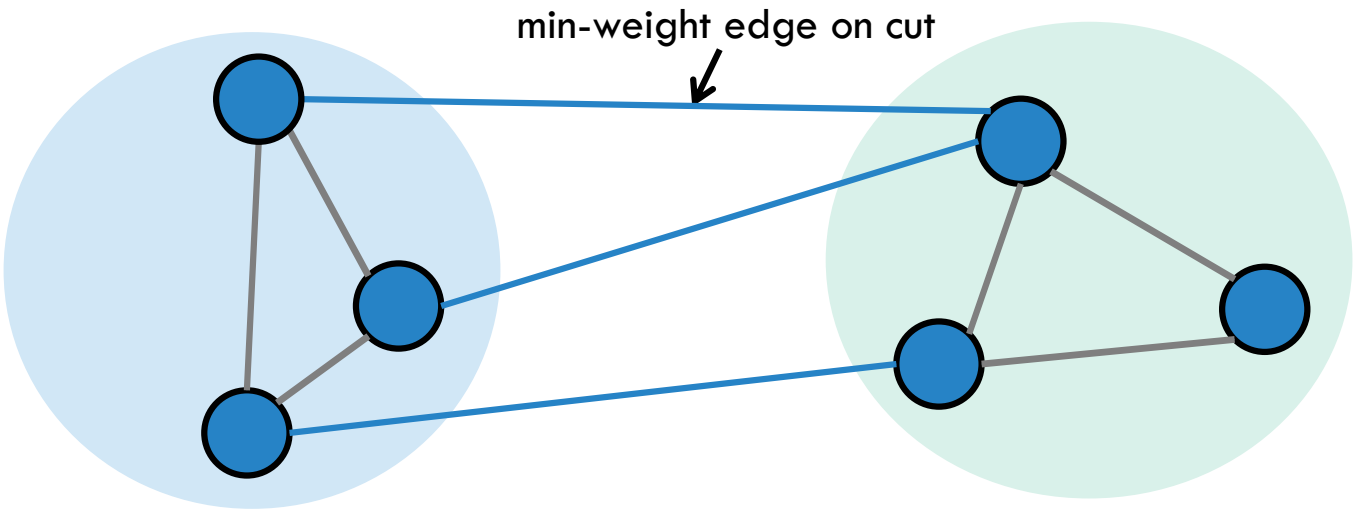
WHAT IS A CUT?

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



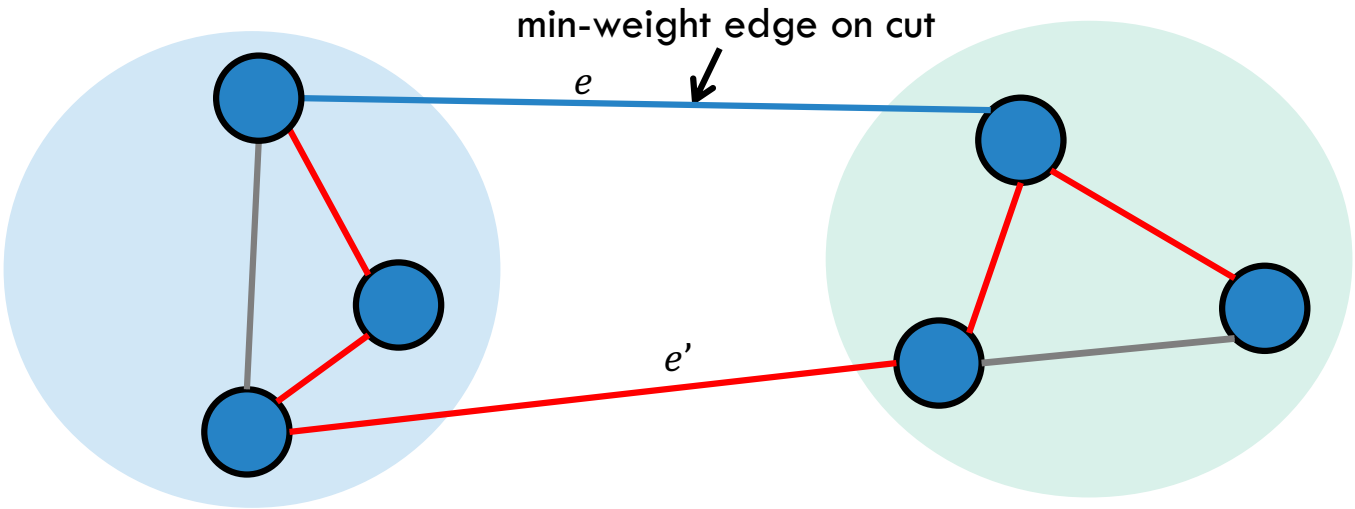
MST PROPERTIES

Property 4: Cut Property; for every partition of nodes, the **minimum weight edge across the cut** is in the MST.



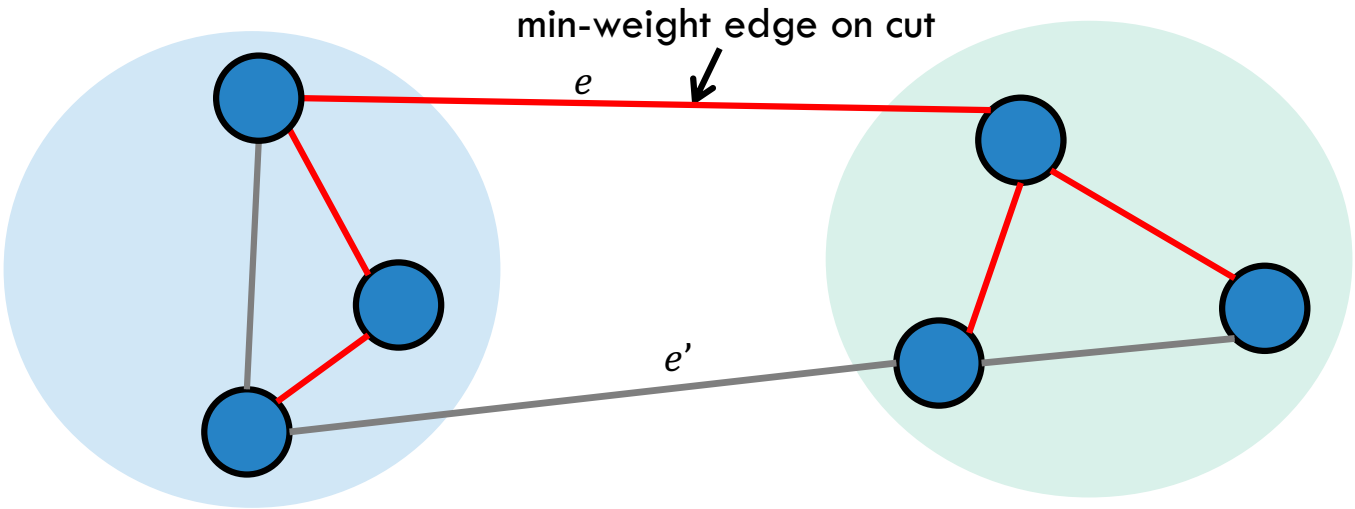
PROOF BY CONTRADICTION

Assume that the min-weight edge e is not on the MST.
Some other edge e' must connect the two partitions.
Remove e' and add e.
You have a new MST with lower weight.
Contradiction! ■



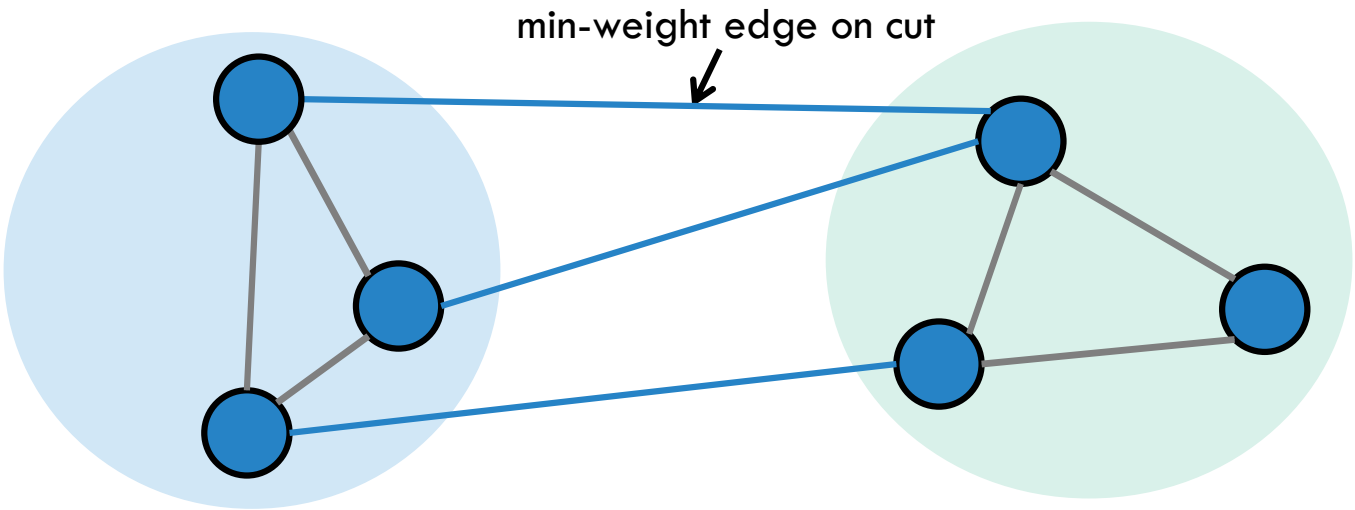
PROOF BY CONTRADICTION

Assume that the min-weight edge e is not on the MST.
Some other edge e' must connect the two partitions.
Remove e' and add e.
You have a new MST with lower weight.
Contradiction! ■

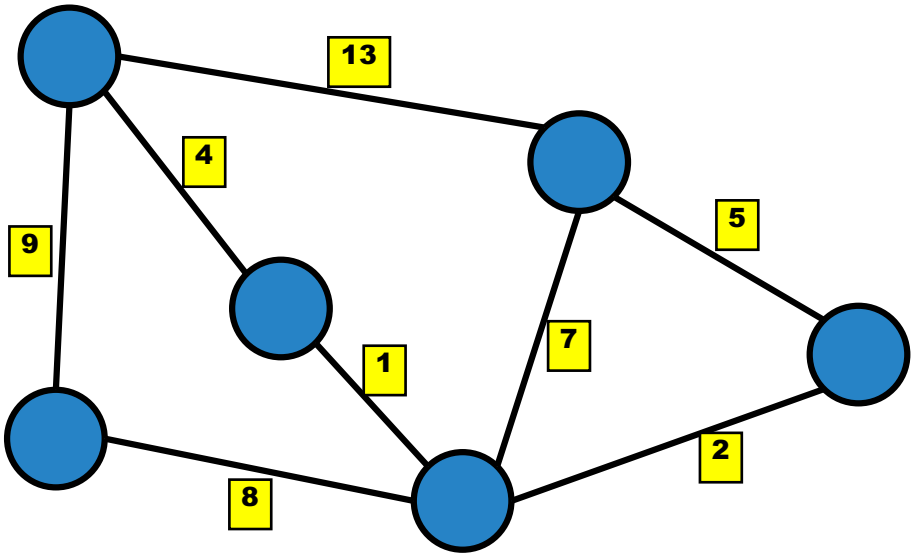


MST PROPERTIES

Property 4: Cut Property; for every partition of nodes, the **minimum weight edge across the cut** is in the MST.



TRUE OR FALSE?



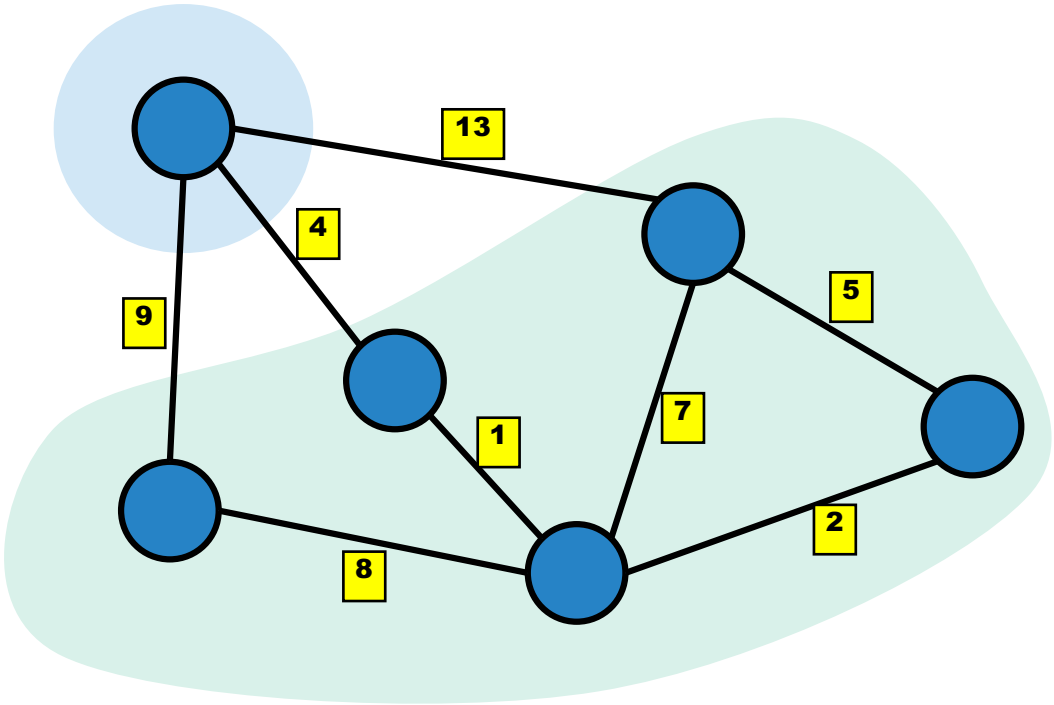
Poll Everywhere

https://bit.ly/2LvG9bq

For every vertex, the minimum incident/outgoing edge is always part of the MST.

A. Yup! Must be true
B. Wrong! False
C. Unable to determine.

TRUE OR FALSE?



Poll Everywhere

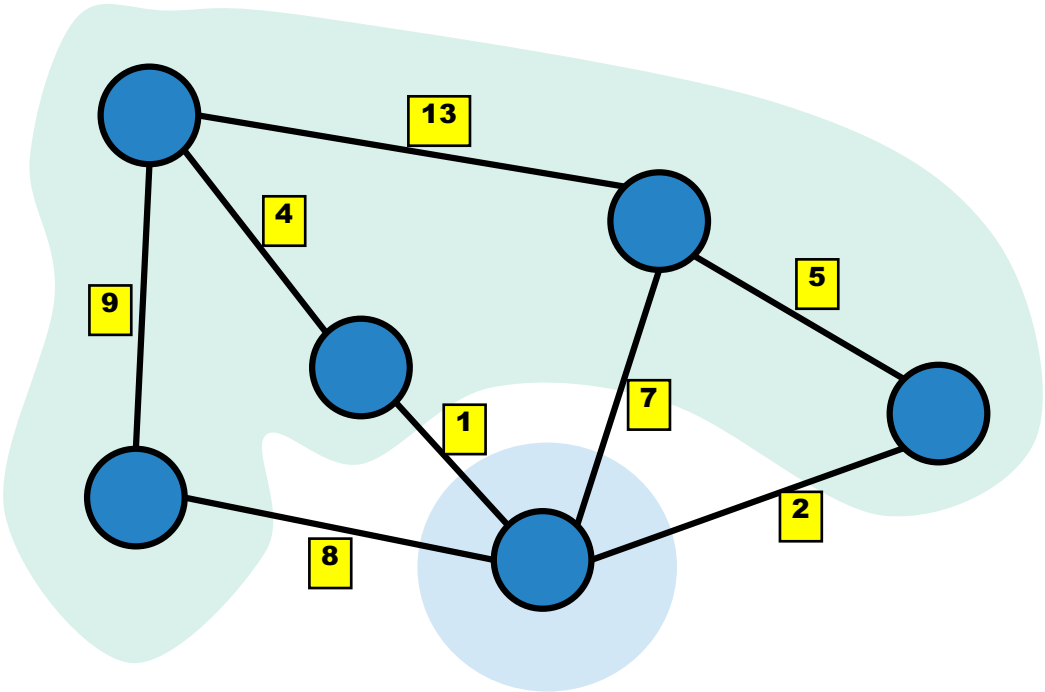
https://bit.ly/2LvG9bq

For every vertex, the minimum incident/outgoing edge is always part of the MST.

A. **Yup! Must be true**
B. Wrong! False
C. Unable to determine.

Property 4: Cut Property; for **every partition** of nodes, the minimum weight edge across the cut is in the MST.

TRUE OR FALSE?



Poll Everywhere

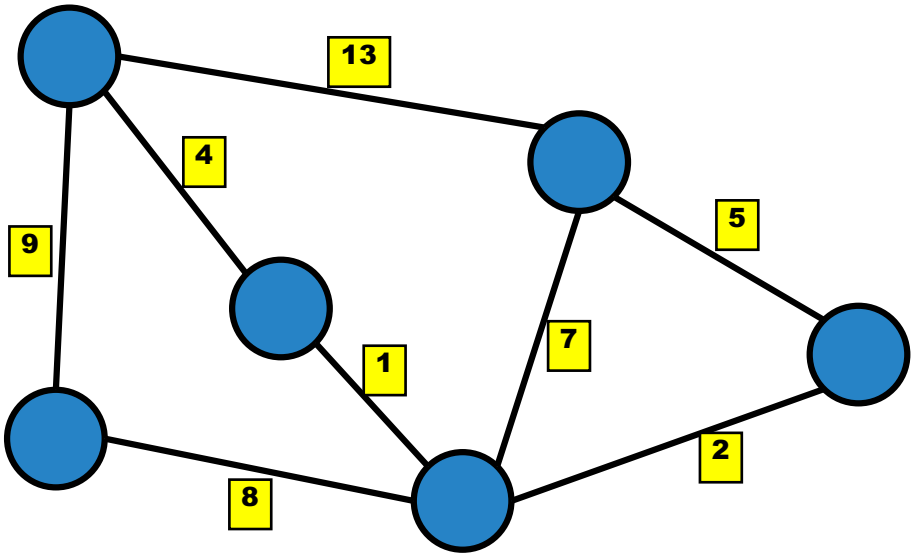
https://bit.ly/2LvG9bq

For every vertex, the minimum incident/outgoing edge is always part of the MST.

A. **Yup! Must be true**
B. Wrong! False
C. Unable to determine.

Property 4: Cut Property; for **every partition** of nodes, the minimum weight edge across the cut is in the MST.

TRUE OR FALSE?



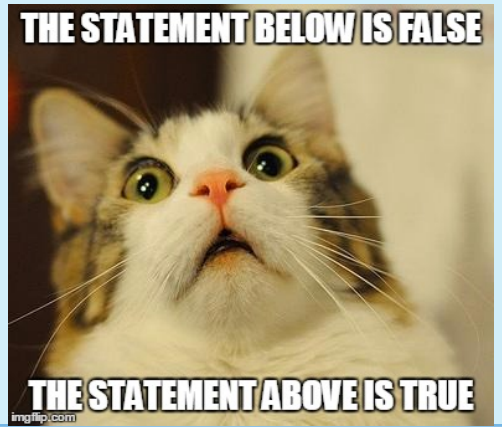
Poll Everywhere

https://bit.ly/2LvG9bq

For every vertex, the maximum incident/outgoing edge is **never** part of the MST.

A. True! You're not tricking me again!

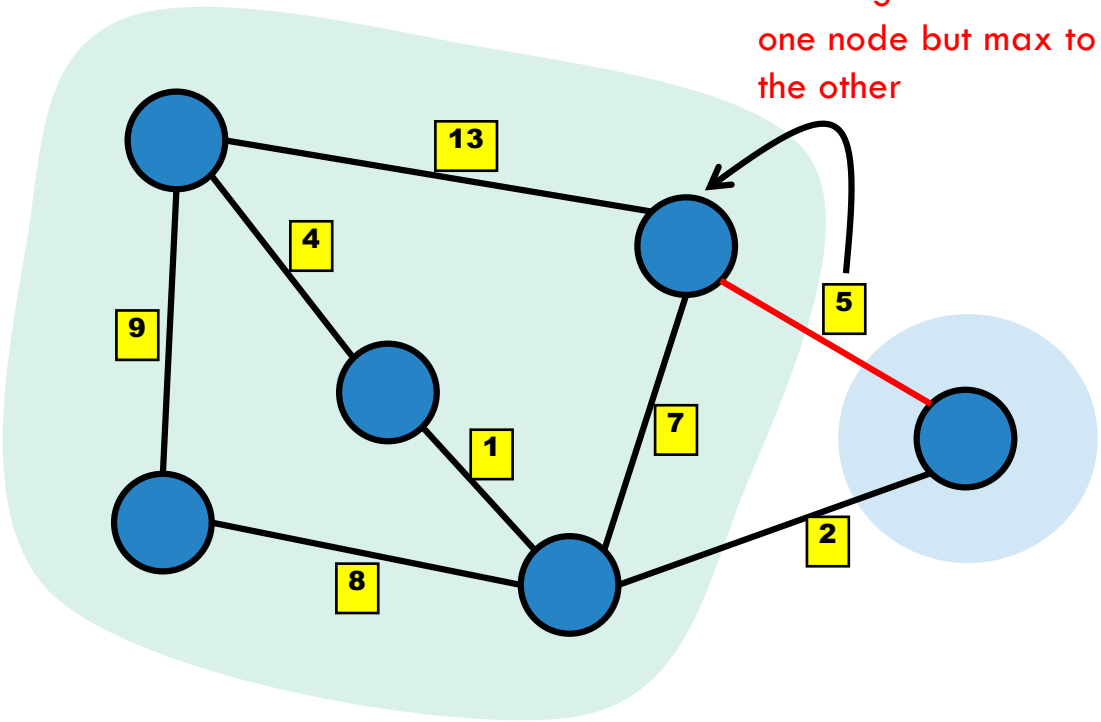
B. False.

C. 

TRUE OR FALSE?

Poll Everywhere

https://bit.ly/2LvG9bq

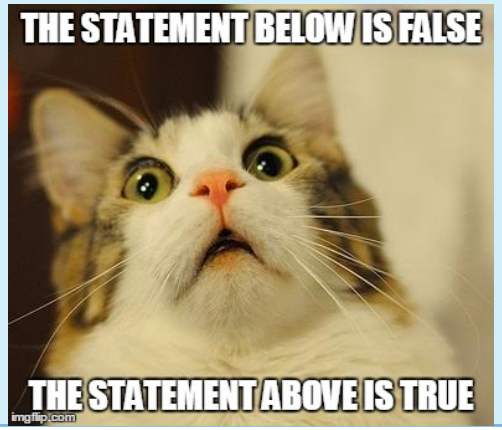


For every vertex, the maximum incident/outgoing edge is **never** part of the MST.

A. True! You're not tricking me again!

B. False.

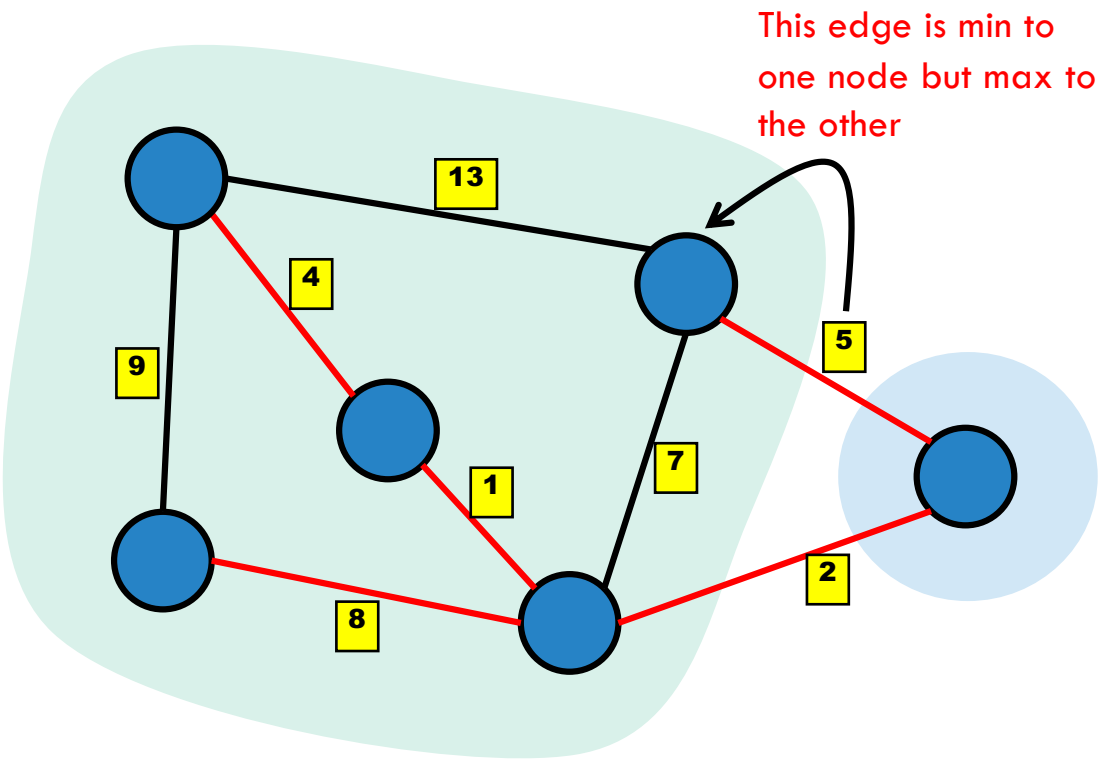
C.



TRUE OR FALSE?

Poll Everywhere

https://bit.ly/2LvG9bq

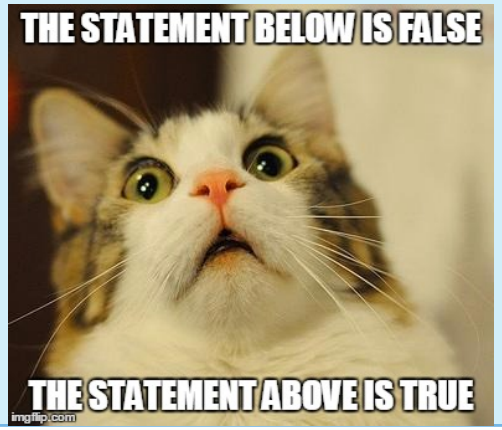


For every vertex, the maximum incident/outgoing edge is **never** part of the MST.

A. True! You're not tricking me again!

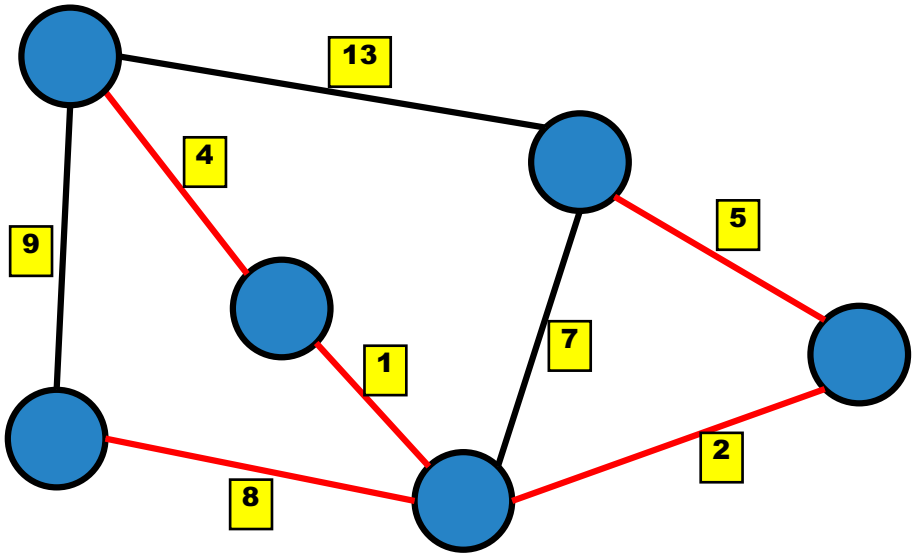
B. False.

C.



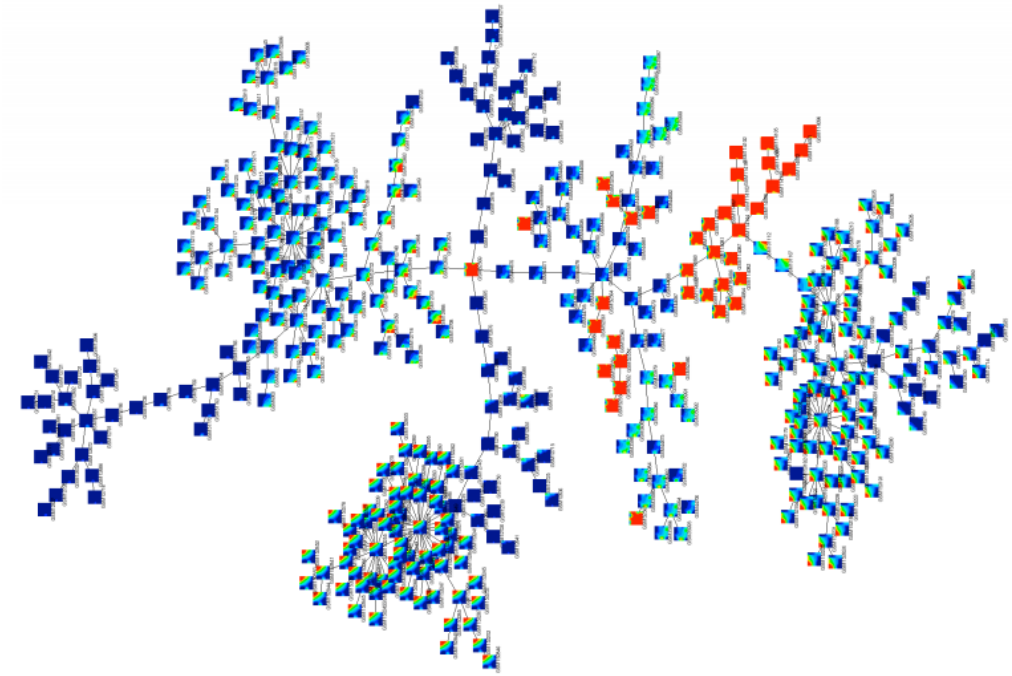
MST PROPERTIES: SUMMARY

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



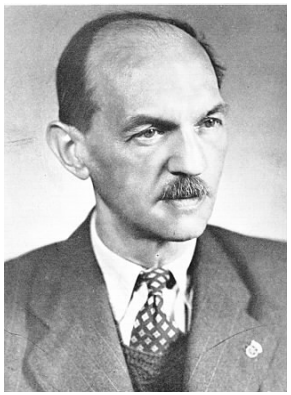
TODAY: MINIMUM SPANNING TREES

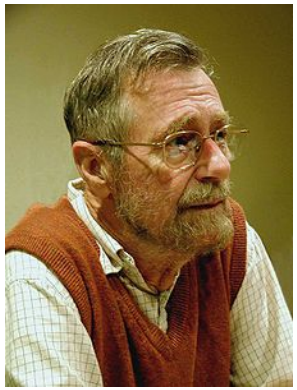
- What is a MST?
- Basic Properties of an MST
- **Prim's Algorithm**
- Kruskal's Algorithm
- Variations

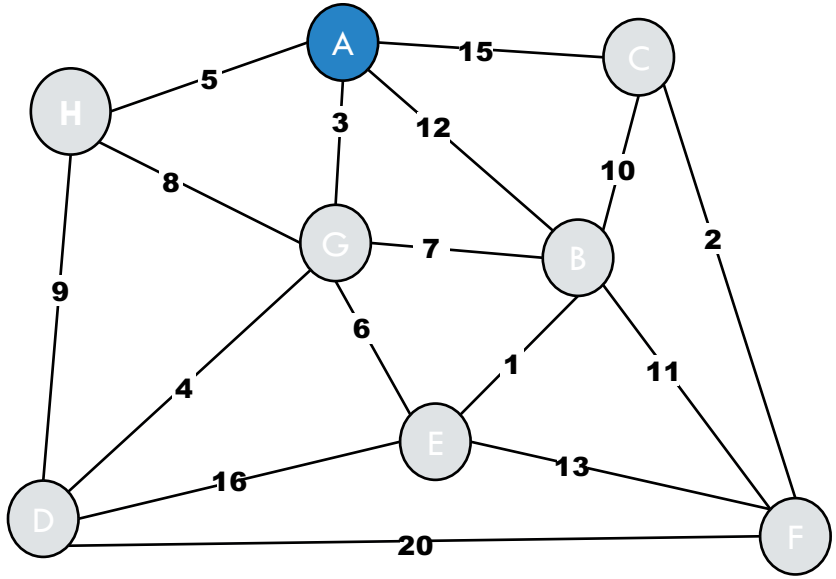


PRIM'S ALGORITHM

(Jarnik 1930, Prim 1957, Dijkstra 1959)







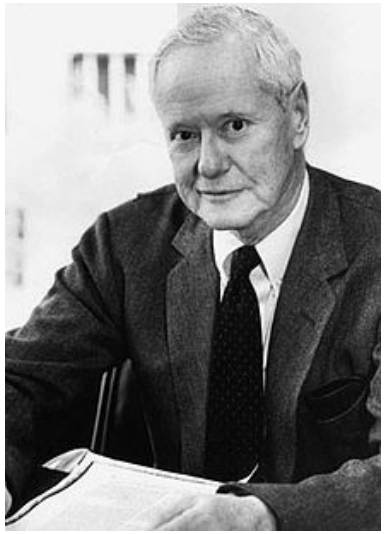
STIGLER'S LAW OF EPONYMY

No scientific discovery is named after its original discoverer.

proposed by University of Chicago statistics professor Stephen Stigler.

Stigler named Columbia University sociology professor **Robert K. Merton** as the original discoverer of “Stigler's law”

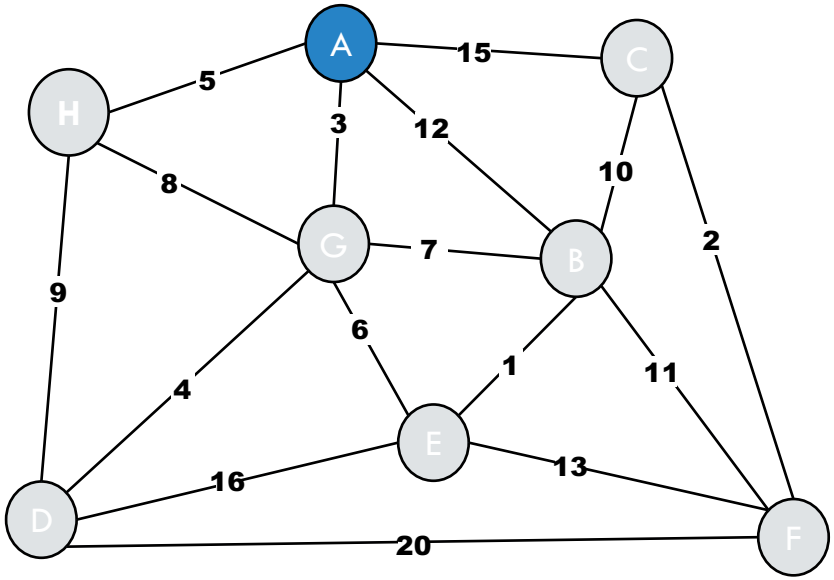
Stigler’s law follows Stigler’s law.



PRIM'S ALGORITHM

Basic idea:

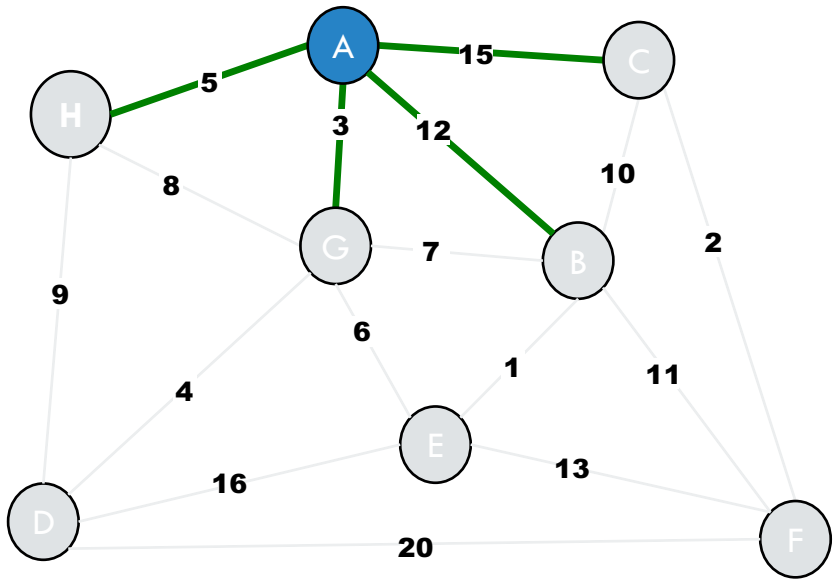
- S : set of nodes connected by blue edges.
- Initially: $S = \{A\}$



PRIM'S ALGORITHM

Basic idea:

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

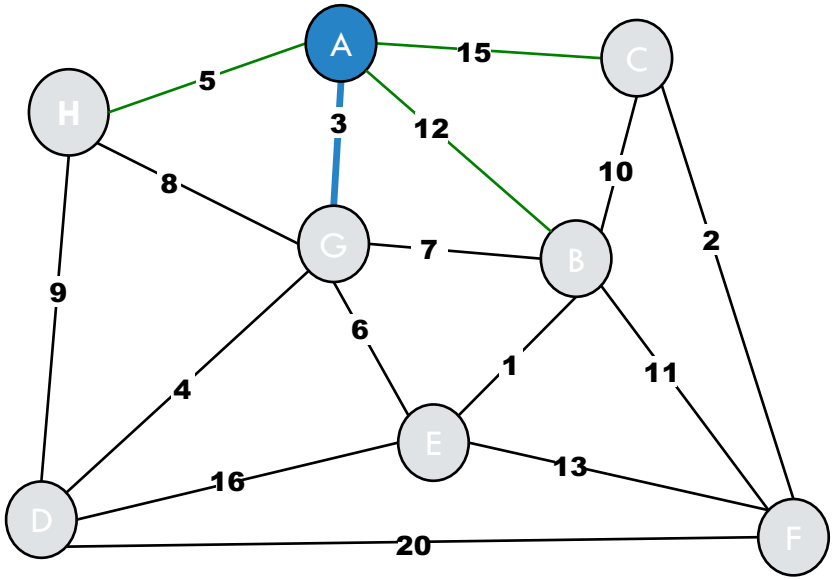


PRIM'S ALGORITHM

Property 4: Cut Property; for **every partition** of nodes, the minimum weight edge across the cut is in the MST.

Basic idea:

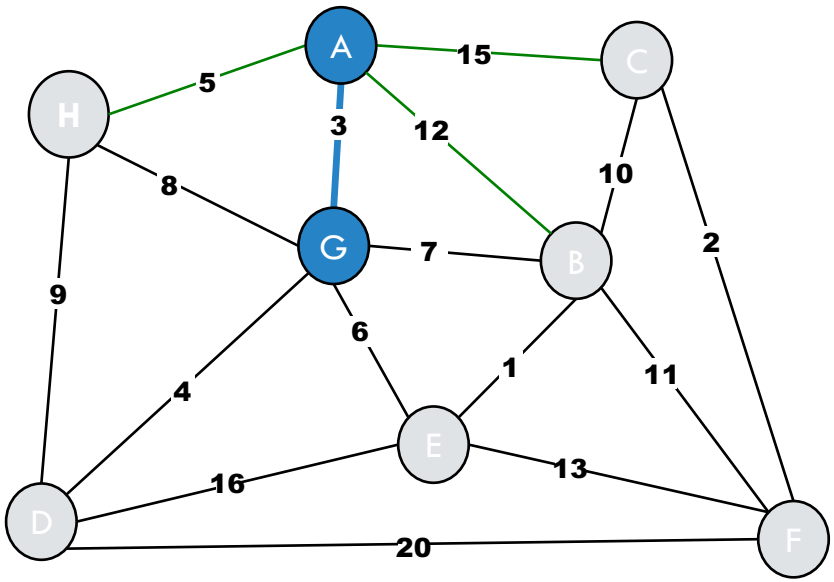
- S : set of nodes connected by blue edges.
- Initially: $S = \{A\}$
- Identify cut: $\{S, V - S\}$
- Find minimum weight edge on cut.



PRIM'S ALGORITHM

Basic idea:

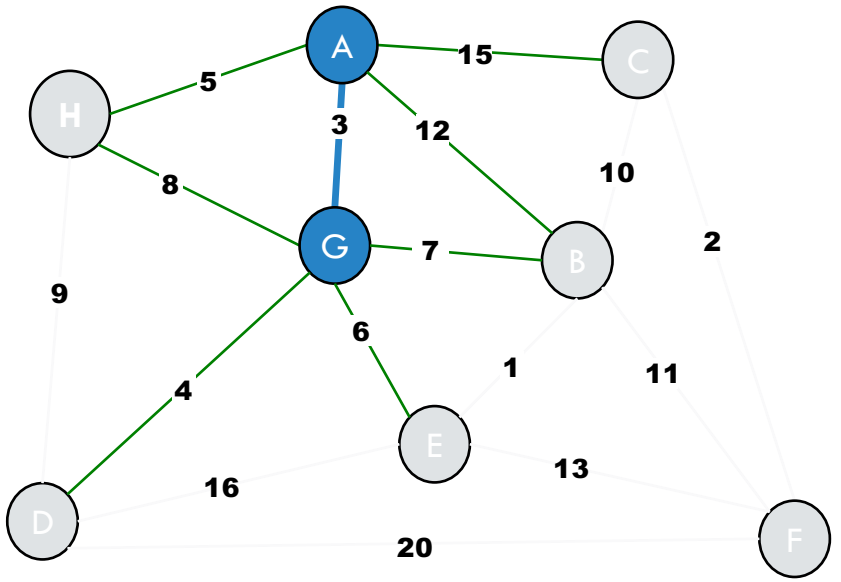
- S : set of nodes connected by blue edges.
- Initially: $S = \{A\}$
- Identify cut: $\{S, V - S\}$
- Find minimum weight edge on cut.
- Add new node to S.



PRIM'S ALGORITHM

Basic idea:

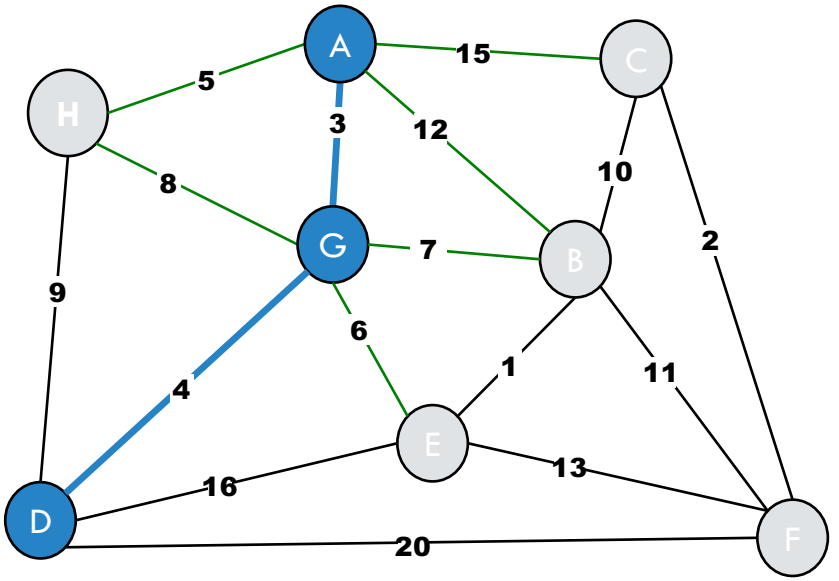
- S : set of nodes connected by blue edges.
- Initially: $S = \{A\}$
- Repeat:
  - Identify cut: $\{S, V - S\}$
  - Find minimum weight edge on cut.
  - Add new node to S.



PRIM'S ALGORITHM

Basic idea:

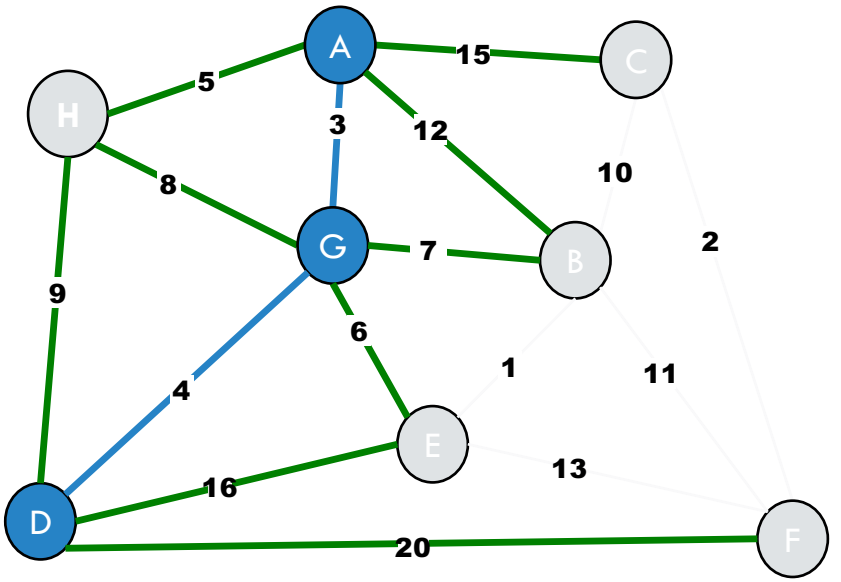
- S : set of nodes connected by blue edges.
- Initially: $S = \{A\}$
- Repeat:
  - Identify cut: $\{S, V - S\}$
  - Find minimum weight edge on cut.
  - Add new node to S.

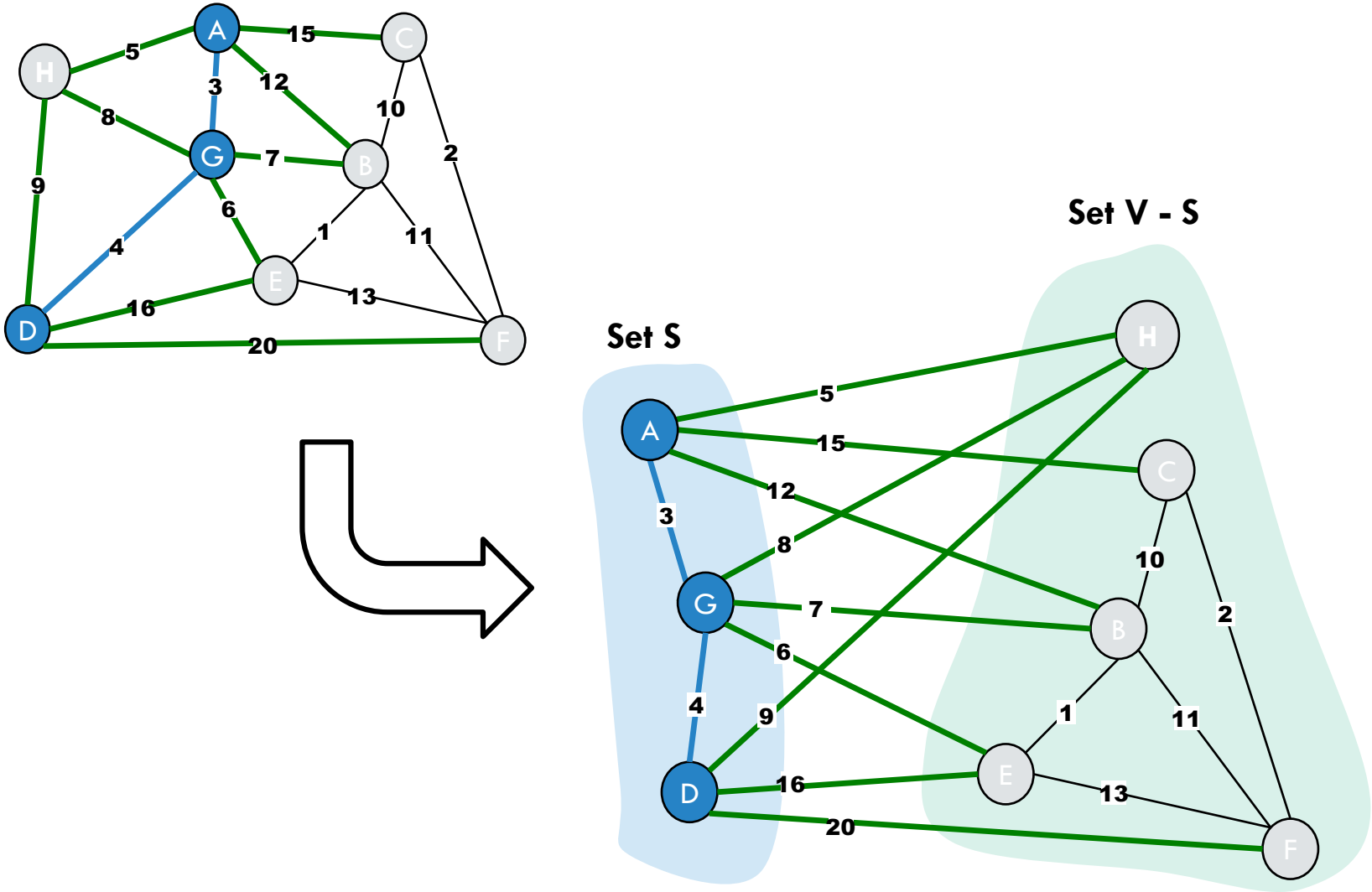


PRIM'S ALGORITHM

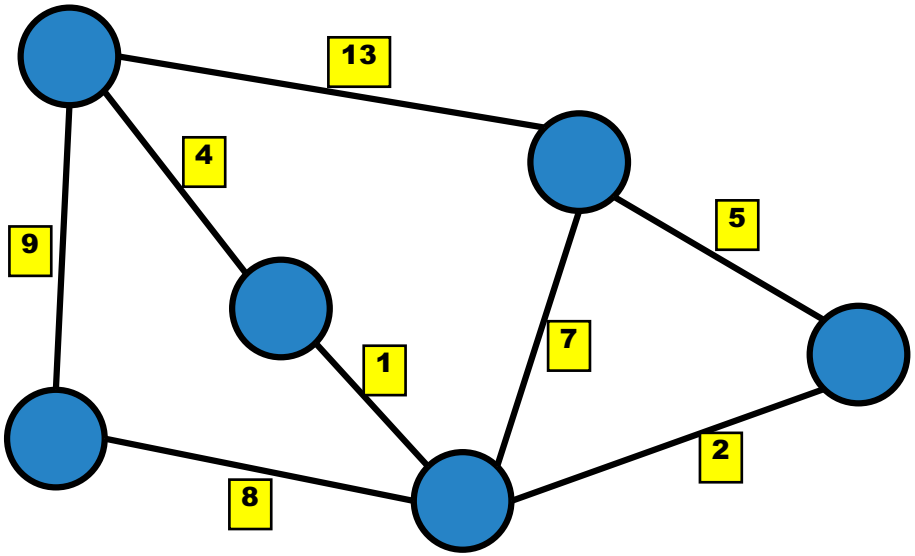
Basic idea:

- 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.




A
C
H
G
B
E
D
F
5
15
3
12
10
8
7
2
9
6
1
11
4
16
13
20
Set V - S
Set S

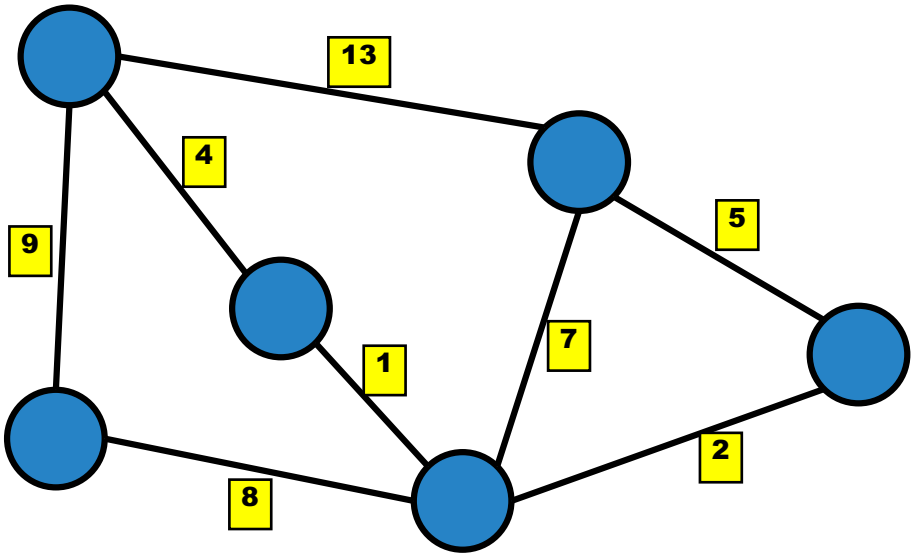
WHICH DATA STRUCTURE?



How do we find the "lightest" edge (with the minimum weight) on a cut?

A. Priority Queue
B. Array
C. BFS / DFS
D. Dijkstra's Algorithm
E. I'm not sure what this question is asking. Why are A & B data structures, and C and D algorithms?

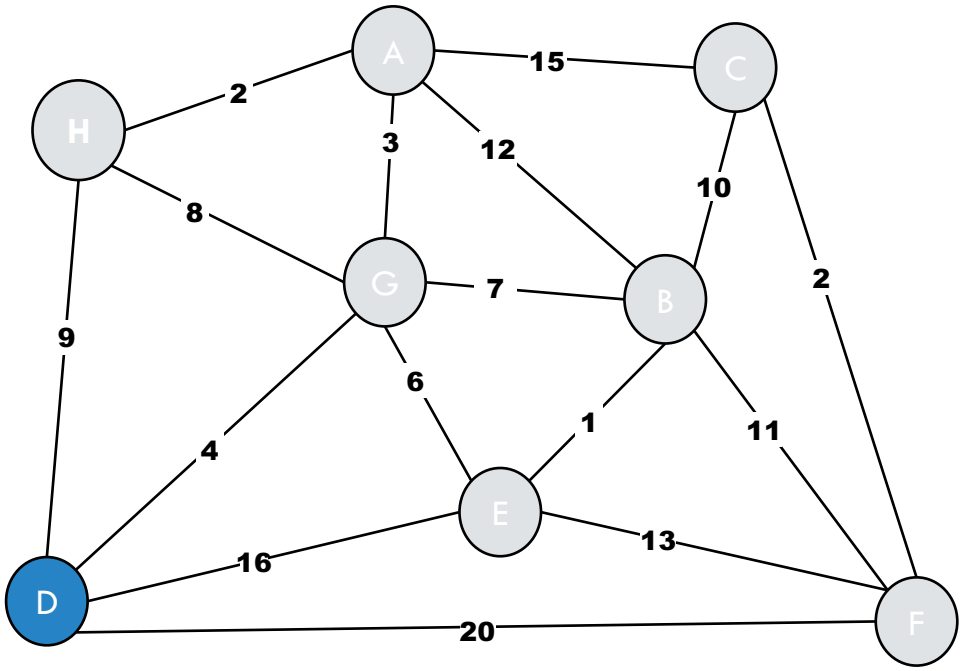
WHICH DATA STRUCTURE?



How do we find the "lightest" edge (with the minimum weight) on a cut?

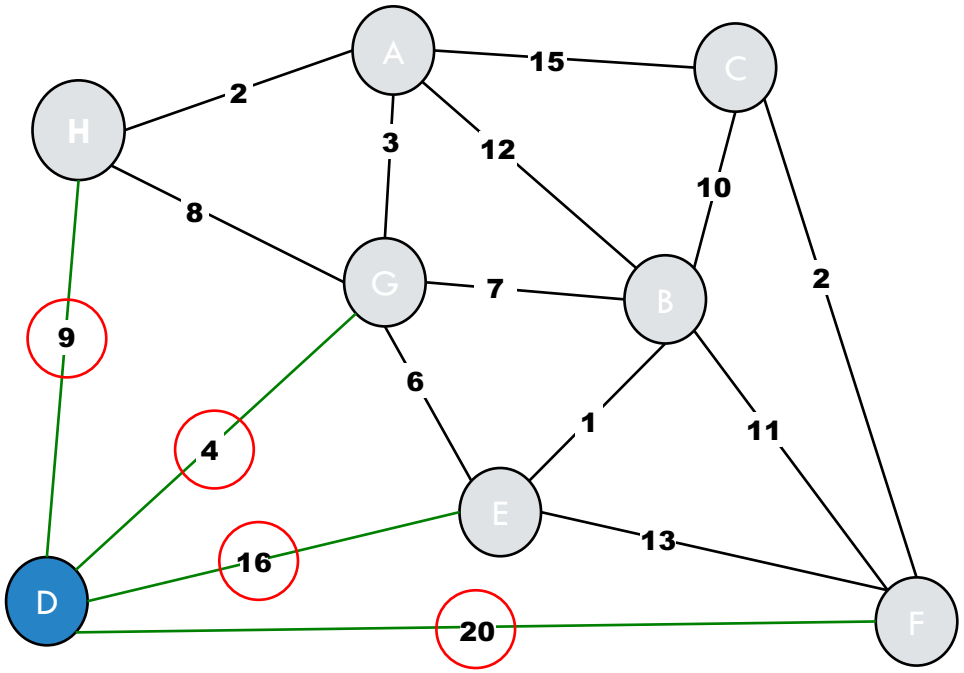
A. **Priority Queue**
B. Array
C. BFS / DFS
D. Dijkstra's Algorithm
E. I'm not sure what this question is asking. Why are A & B data structures, and C and D algorithms?

PRIM'S ALGORITHM: EXAMPLE



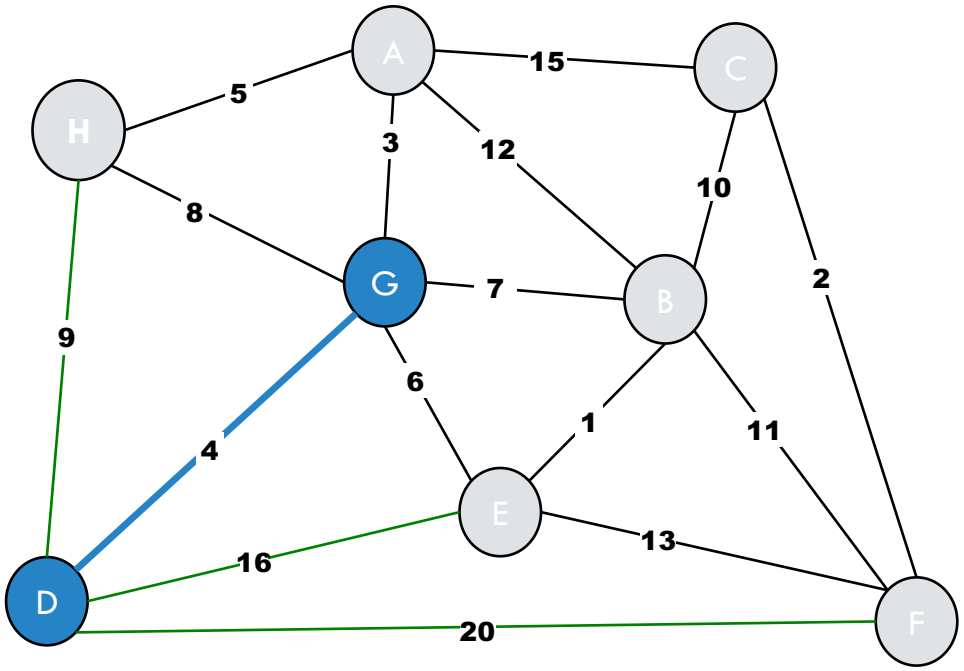
Vertex	Weight
D	0

PRIM'S ALGORITHM: EXAMPLE



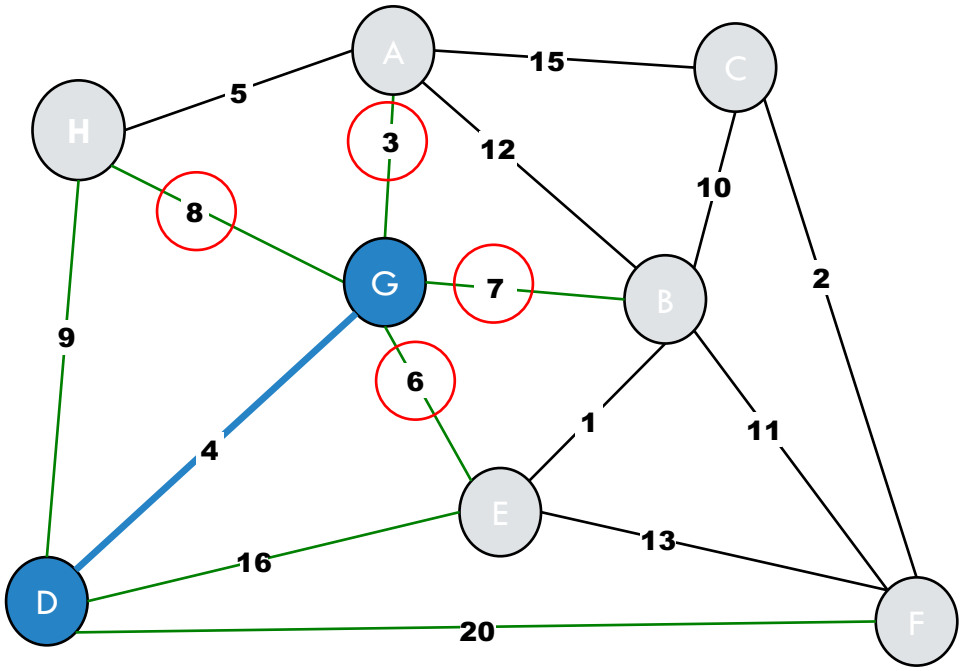
Vertex	Weight
G	4
H	9
E	16
F	20

PRIM'S ALGORITHM: EXAMPLE



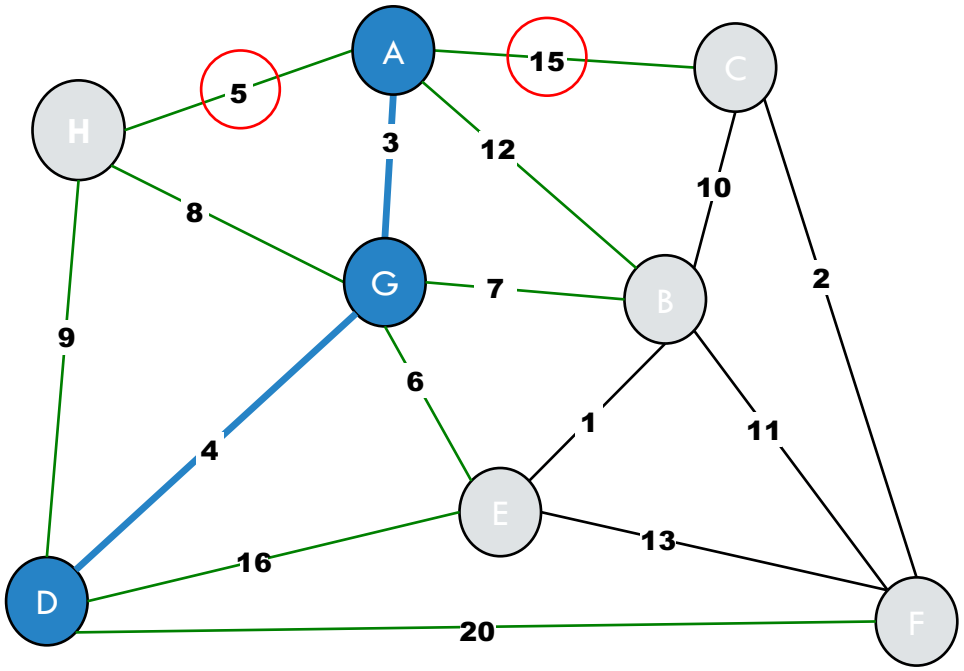
Vertex	Weight
H	9
E	16
F	20

PRIM'S ALGORITHM: EXAMPLE



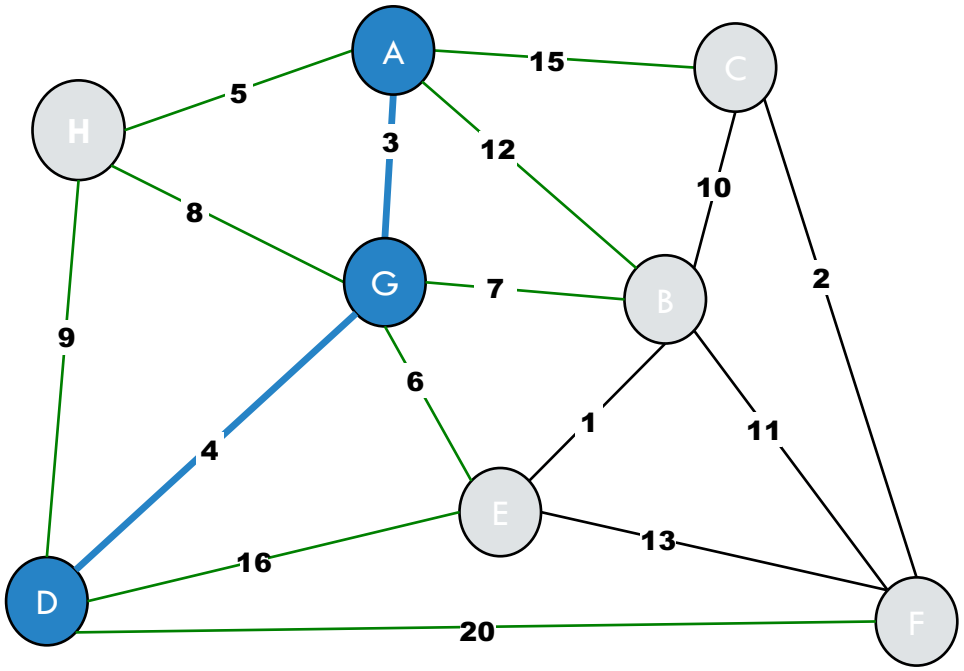
Vertex	Weight
A	3
E	16→6
B	7
H	9→8
F	20

PRIM'S ALGORITHM: EXAMPLE



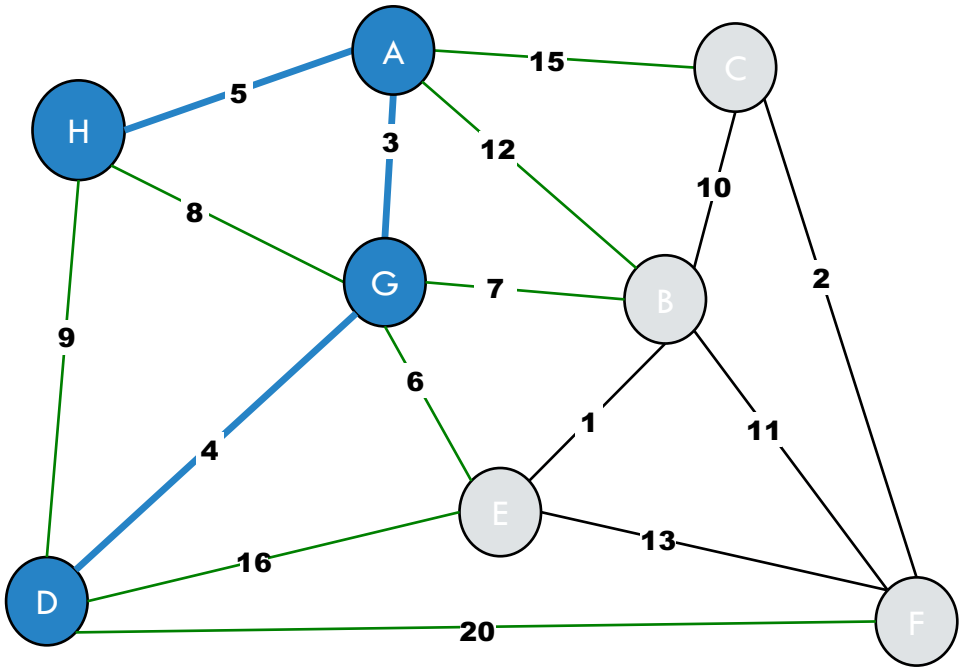
Vertex	Weight
H	**8→5**
E	6
B	7
C	**15**
F	20

PRIM'S ALGORITHM: EXAMPLE



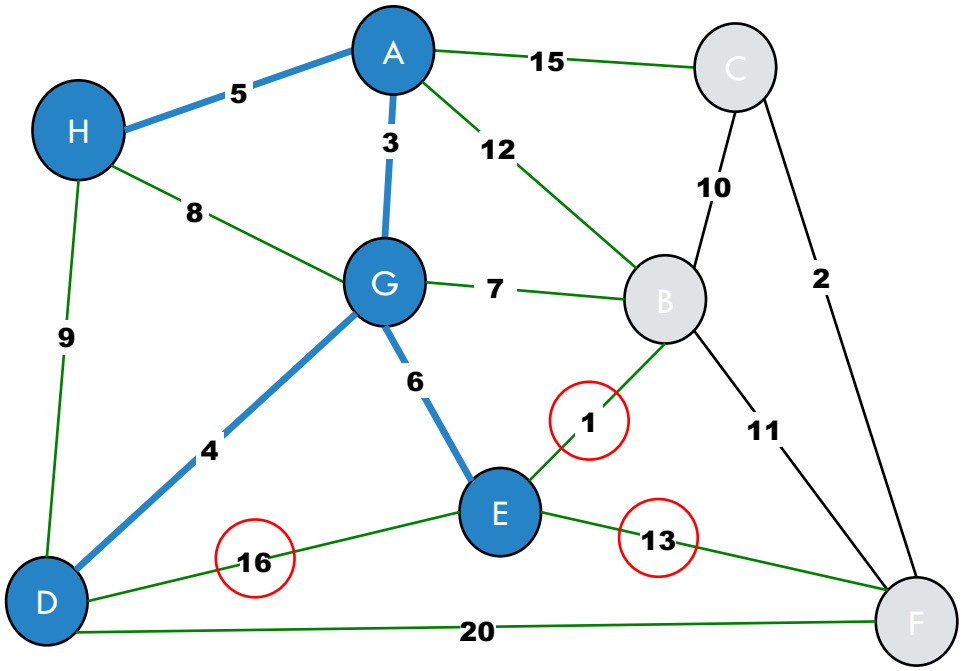
Vertex	Weight
H	5
E	6
B	7
C	15
F	20

PRIM'S ALGORITHM: EXAMPLE



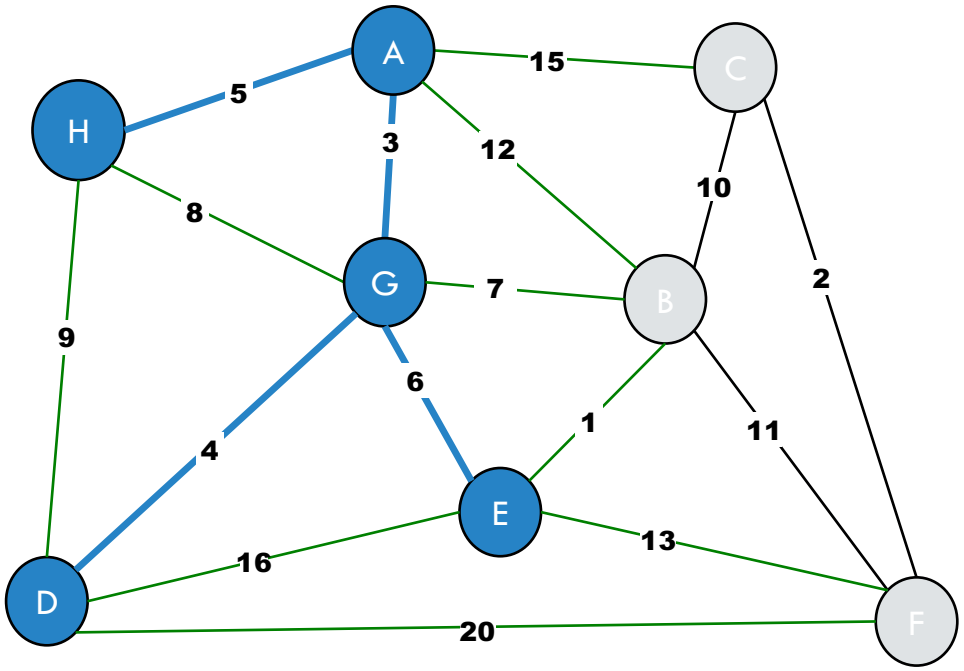
Vertex	Weight
E	6
B	7
C	15
F	20

PRIM'S ALGORITHM: EXAMPLE



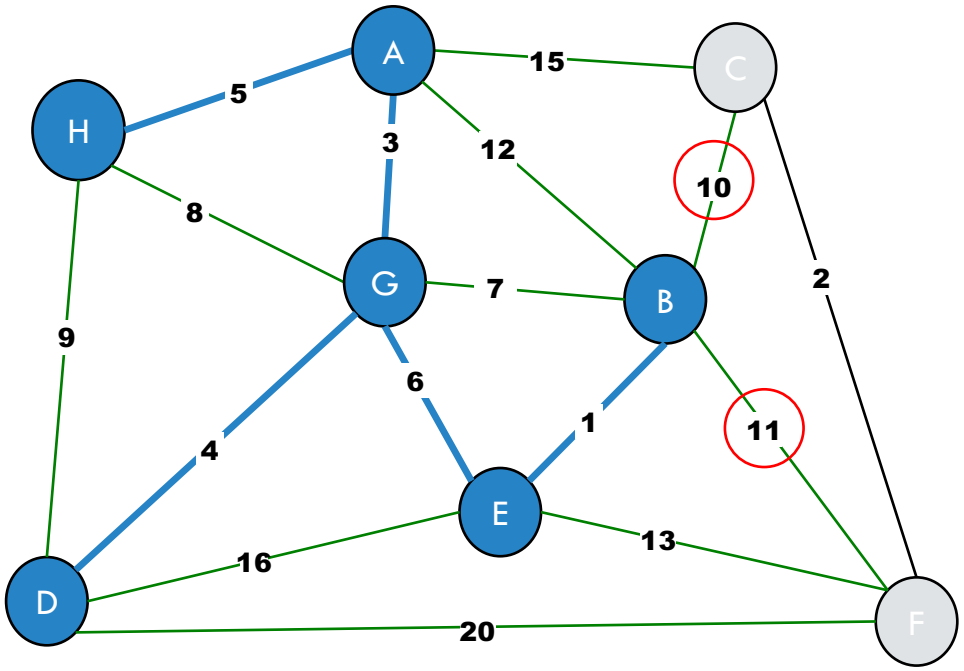
Vertex	Weight
B	**7→1**
C	15
F	**20→13**

PRIM'S ALGORITHM: EXAMPLE



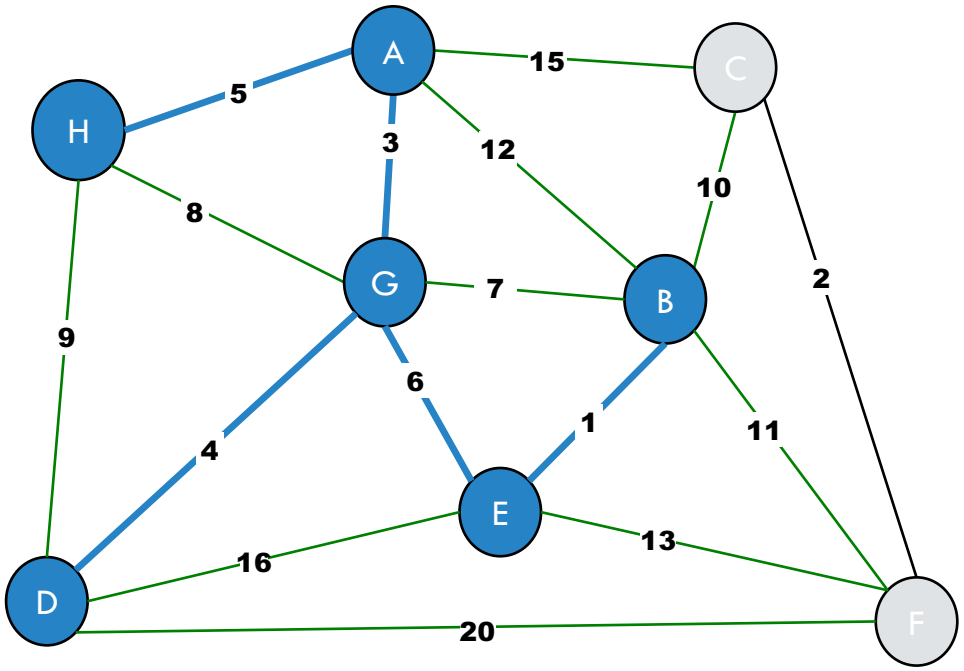
Vertex	Weight
B	1
C	15
F	13

PRIM'S ALGORITHM: EXAMPLE



Vertex	Weight
C	15→10
F	13→11

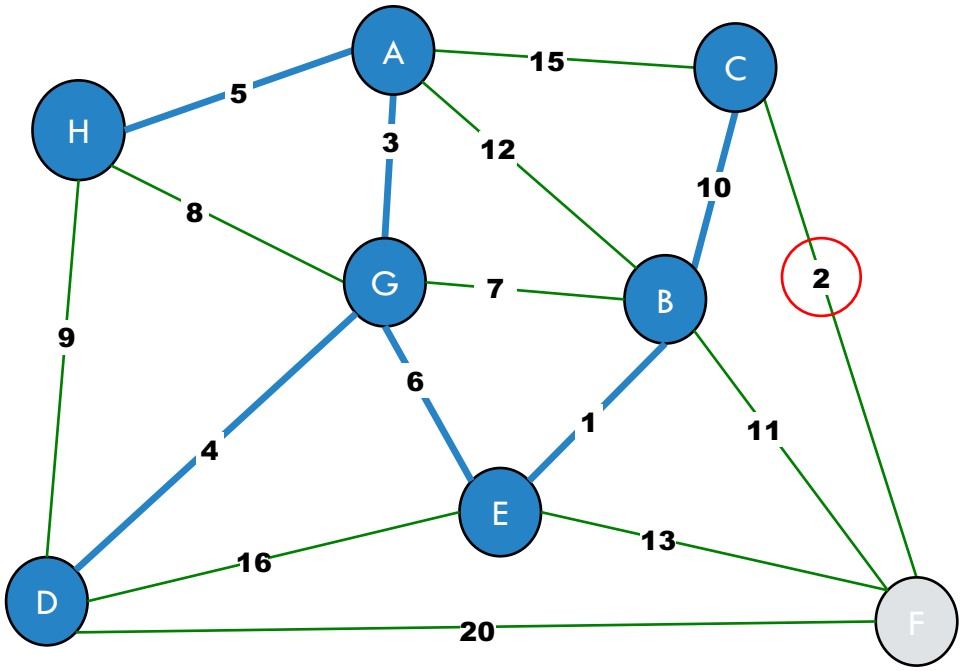
PRIM'S ALGORITHM: EXAMPLE



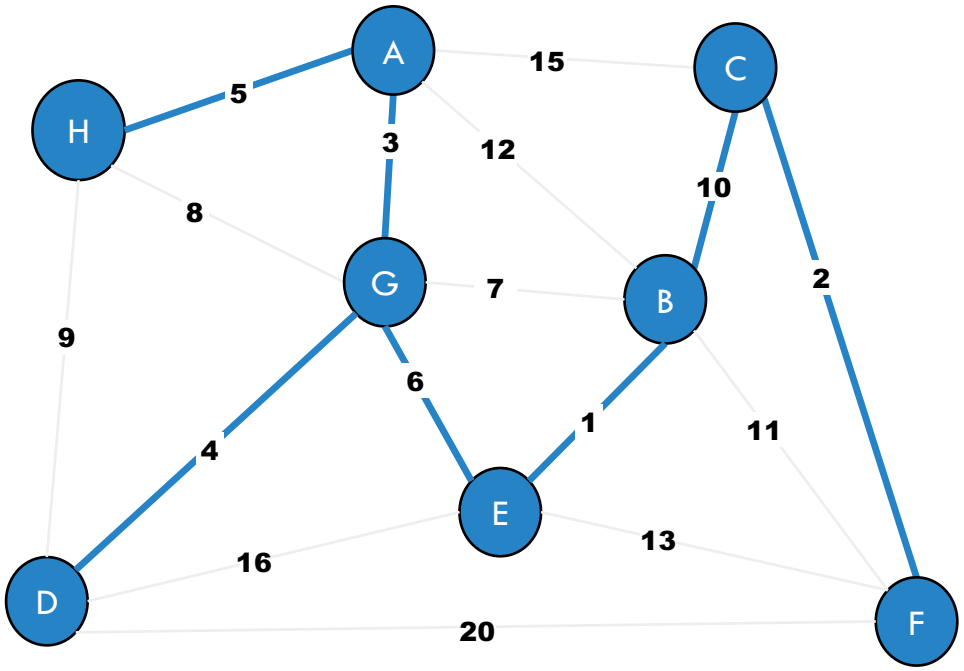
Vertex	Weight
C	10
F	11

PRIM'S ALGORITHM: EXAMPLE

Vertex	Weight
F	11→2



PRIM'S ALGORITHM: EXAMPLE



Vertex	Weight

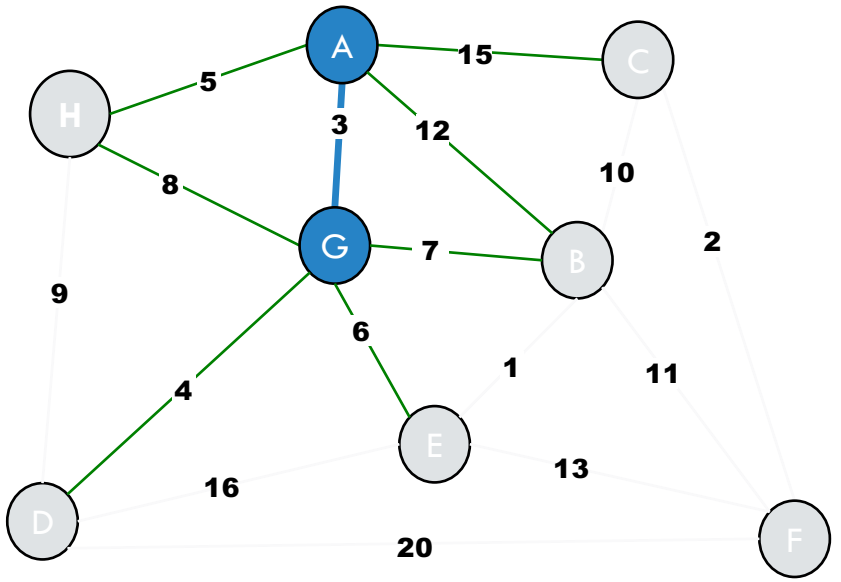
PRIM'S ALGORITHM

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 Sketch:

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



PRIM'S ALGORITHM

Poll Everywhere

https://bit.ly/2LvG9bq

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.

What is the running time of Prim's algorithm

A. $O(V)$
B. $O(E)$
C. $O(E \log V)$
D. $O(V \log E)$
E. $O(EV)$
F. $O(V \log V)$

PRIM'S ALGORITHM

Poll Everywhere

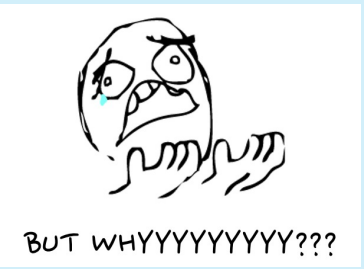
https://bit.ly/2LvG9bq

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.

What is the running time of Prim's algorithm

A. $O(V)$
B. $O(E)$
C. $O(E \log V)$
D. $O(V \log E)$
E. $O(EV)$
F. $O(V \log V)$



PRIM'S ALGORITHM

Poll Everywhere

https://bit.ly/2LvG9bq

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:

$$O(E \log V)$$

A TALE OF 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 neighbor node that is **closest to the source**

Use Priority Queue to order nodes by **distance**

Both are "Greedy Algorithms" that are efficient and optimal

PROBLEM: CONNECT ME UP!

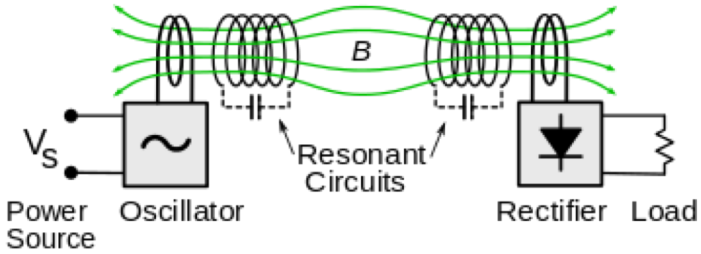
The island of Newworldia has recently rolled out wireless power towers.

Connecting up the transmission towers is not easy (and expensive).

Naruto and you are hired to find a way to connect up the towers to minimize the cost.

Model the problem as a graph.

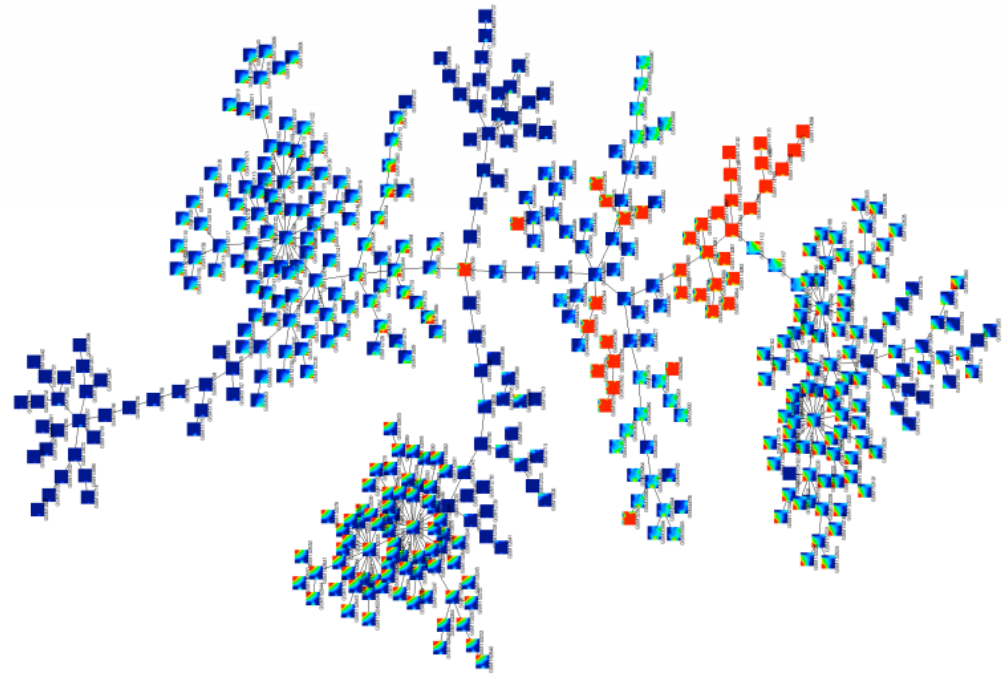
Run Prim's algorithm.



TODAY: MINIMUM SPANNING TREES

- What is a MST?
- Basic Properties of an MST
- Generic MST Algorithm
- Prim's Algorithm
- **Kruskal's Algorithm**
- **Variations**

Next Week!



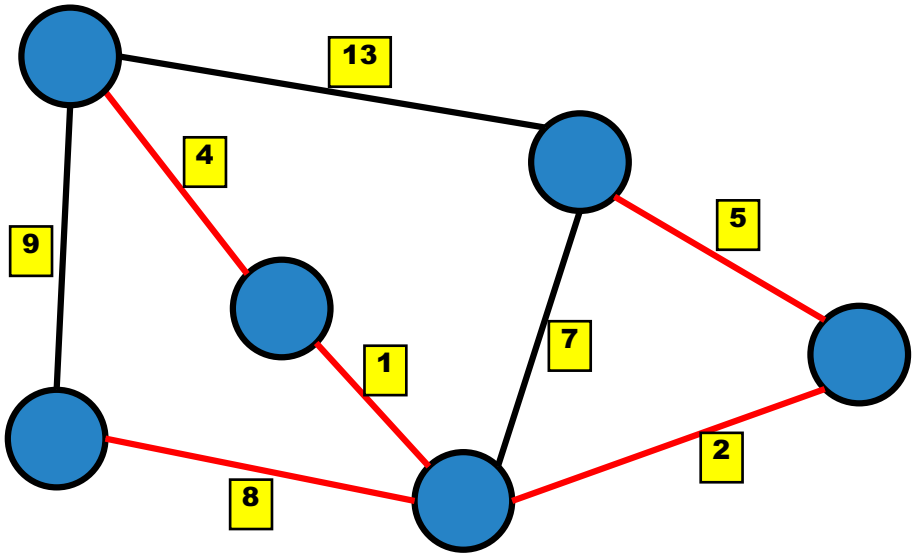
LEARNING OUTCOMES

By the end of this session, students should be able to:

- Explain the **Minimum Spanning Tree (MST)** and its **properties.**
- Describe **Prim's algorithm** and its **running time.**

MST PROPERTIES: SUMMARY

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



A TALE OF 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 neighbor node that is **closest to the source**

Use Priority Queue to order nodes by **distance**

QUESTIONS?

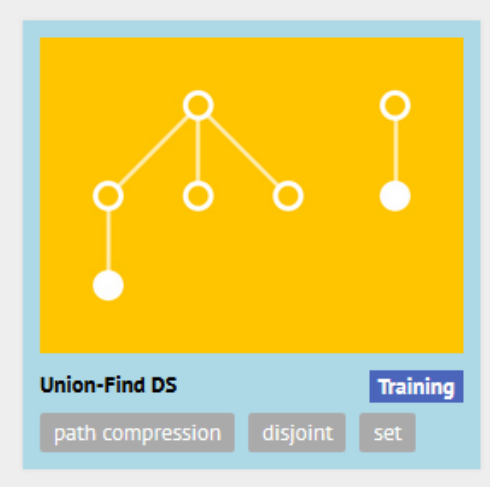
Poll Everywhere

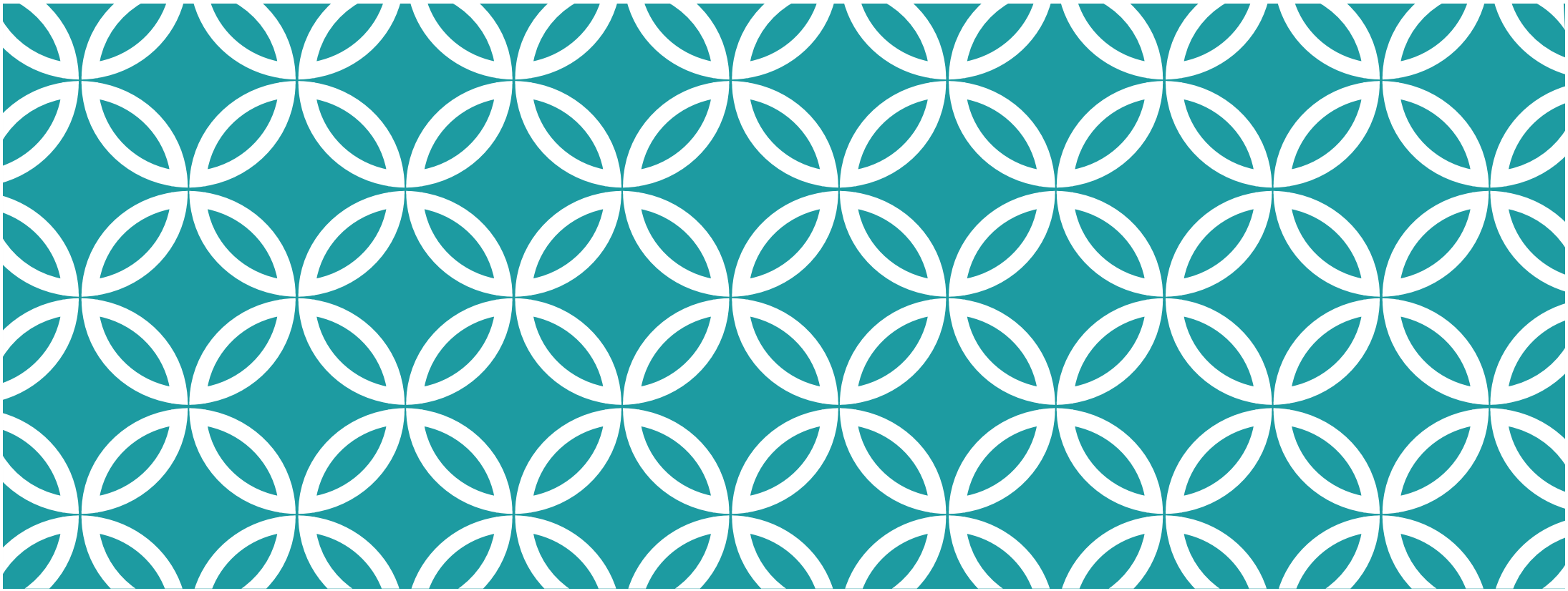
https://bit.ly/2LvG9bq

BEFORE NEXT WEEK

Revise Kruskal's algorithm on Visualgo!

And The Union Find Data Structure!

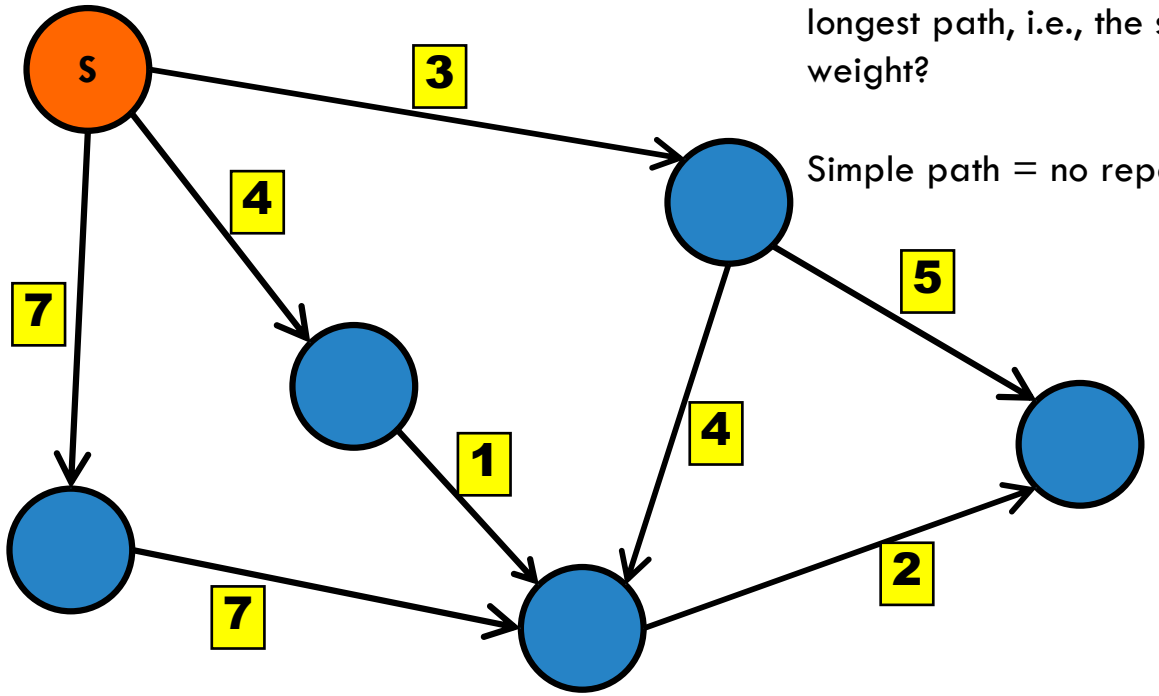




LONGEST PATH IN A DAG

Harold Soh
harold@comp.nus.edu.sg

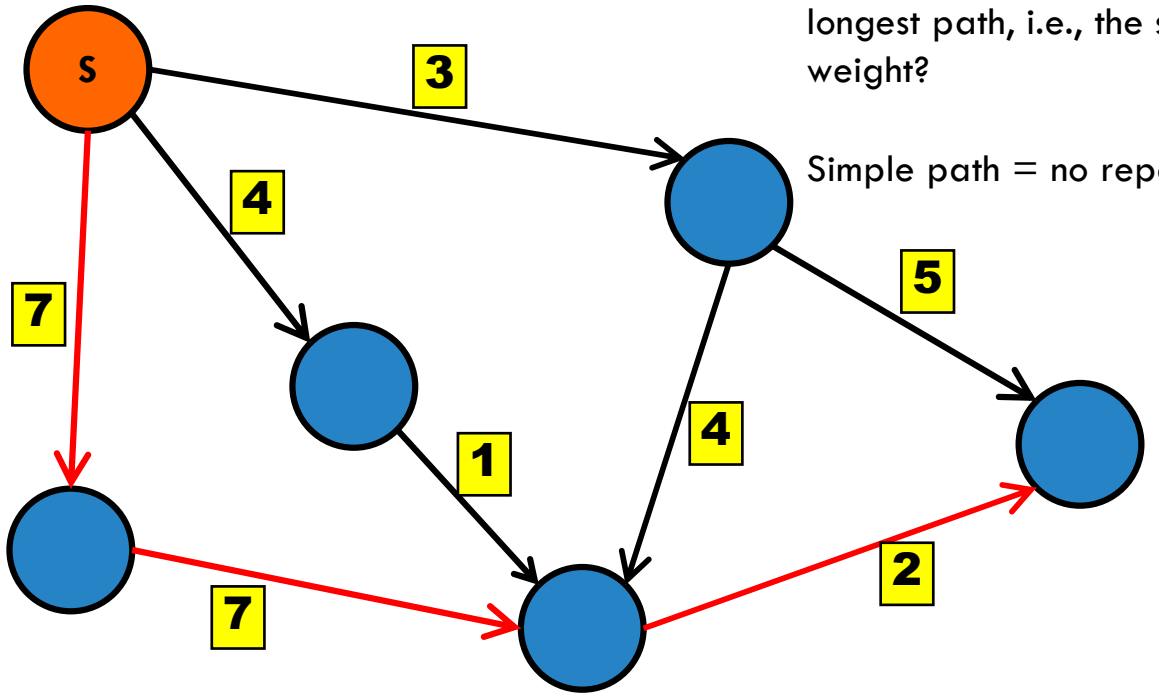
HOW CAN I FIND THE LONGEST PATH IN A DAG?



Given a DAG $G = (V, E)$ how can I find the longest path, i.e., the simple path with maximum weight?

Simple path = no repeated edges

LONGEST PATH IN A DAG?

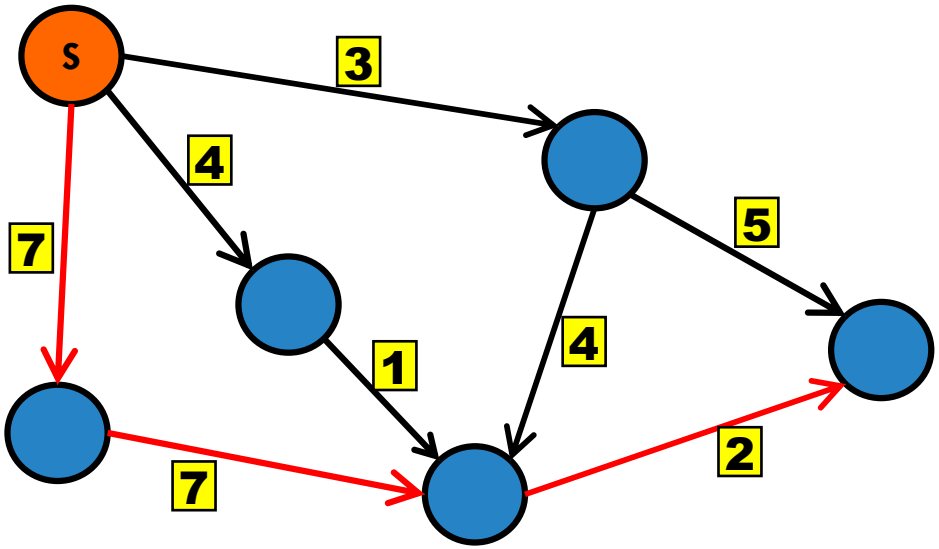


Given a DAG $G = (V, E)$ how can I find the longest path, i.e., the simple path with maximum weight?

Simple path = no repeated edges

LONGEST PATH IN A DAG?

Poll Everywhere
https://bit.ly/2LvG9bq



Given a DAG $G=(V,E)$ finding the longest simple path requires:

A. $O(E)$
B. $O(V+E)$
C. $O(E \log V)$
D. $O(V \log E)$
E. $O(VE)$

SOLUTION #1: MODIFY RELAXATION

Shortest path relaxation:

```
relax(int u, int v){
      if (dist[v] > dist[u] + weight(u,v))
         dist[v] = dist[u] + weight(u,v);
}
```

Longest path relaxation?

Initialize all $d[v] = -\infty$

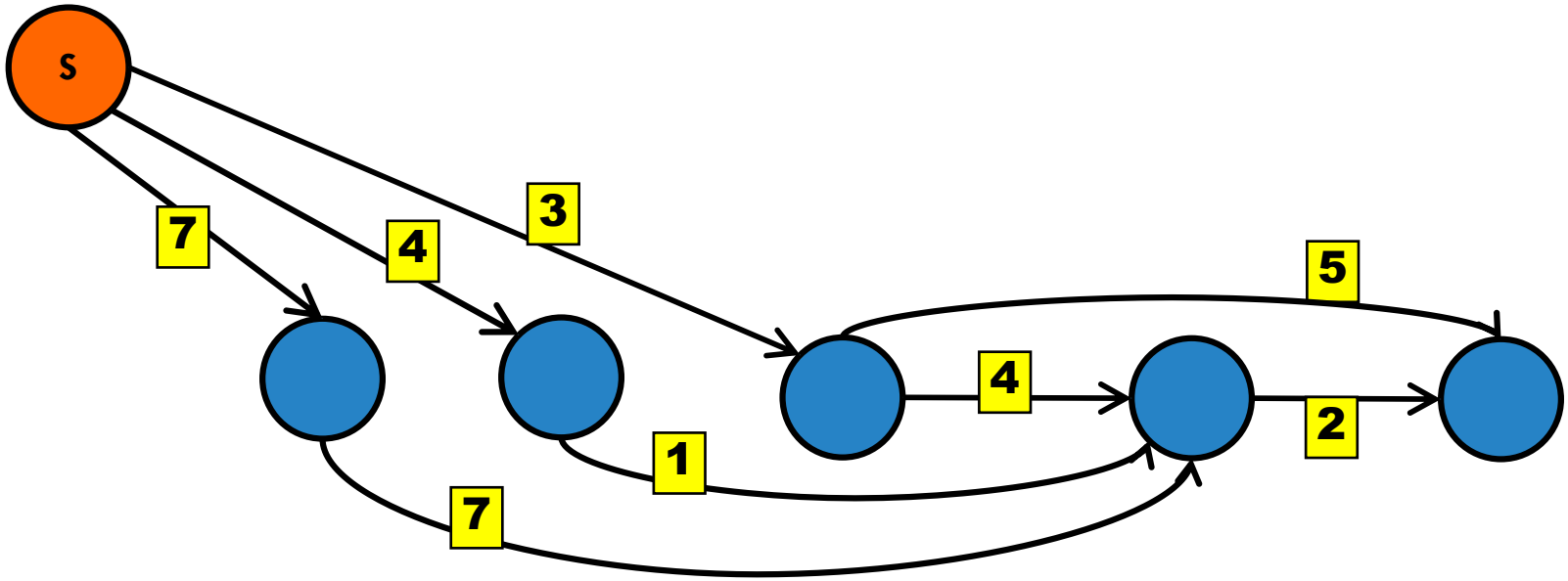
```
longrelax(int u, int v){
      if (dist[v] < dist[u] + weight(u,v))
         dist[v] = dist[u] + weight(u,v);
}
```

SOLUTION #1: MODIFY RELAXATION

Perform longest path relaxations in Topological Sort order.

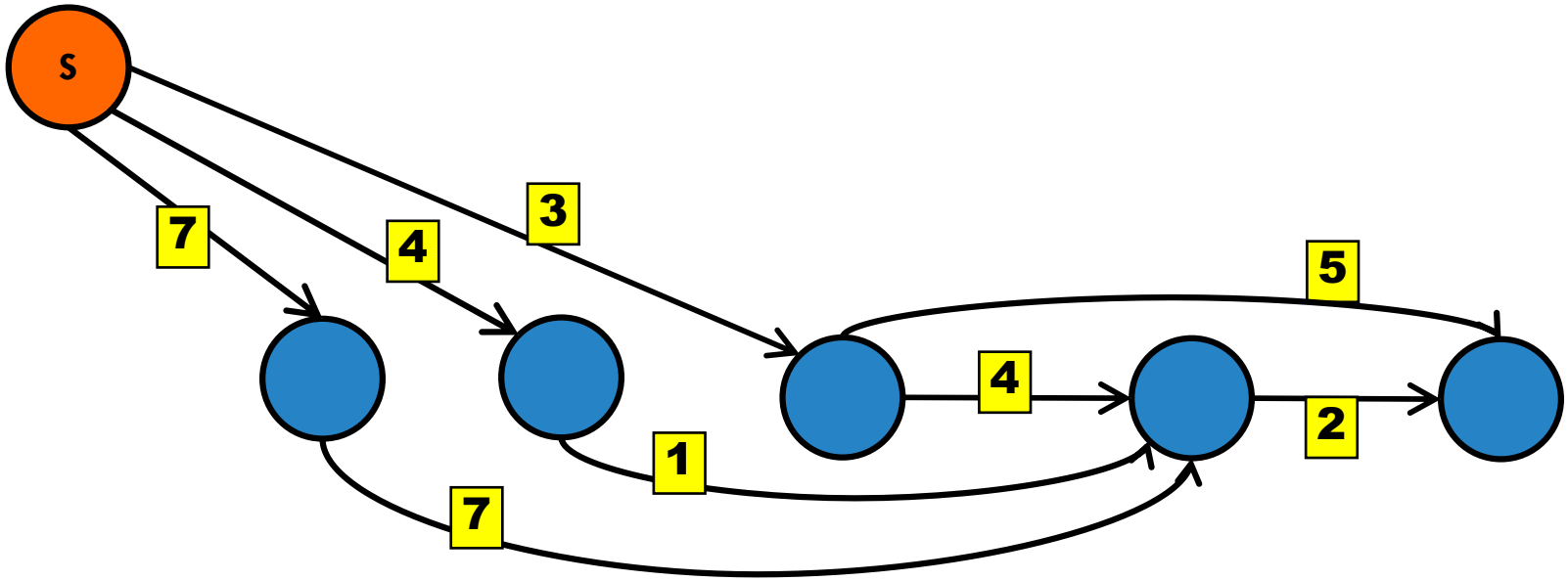
DIRECTED ACYCLIC GRAPH (DAG)

1. Topological sort
2. LongRelax in order.



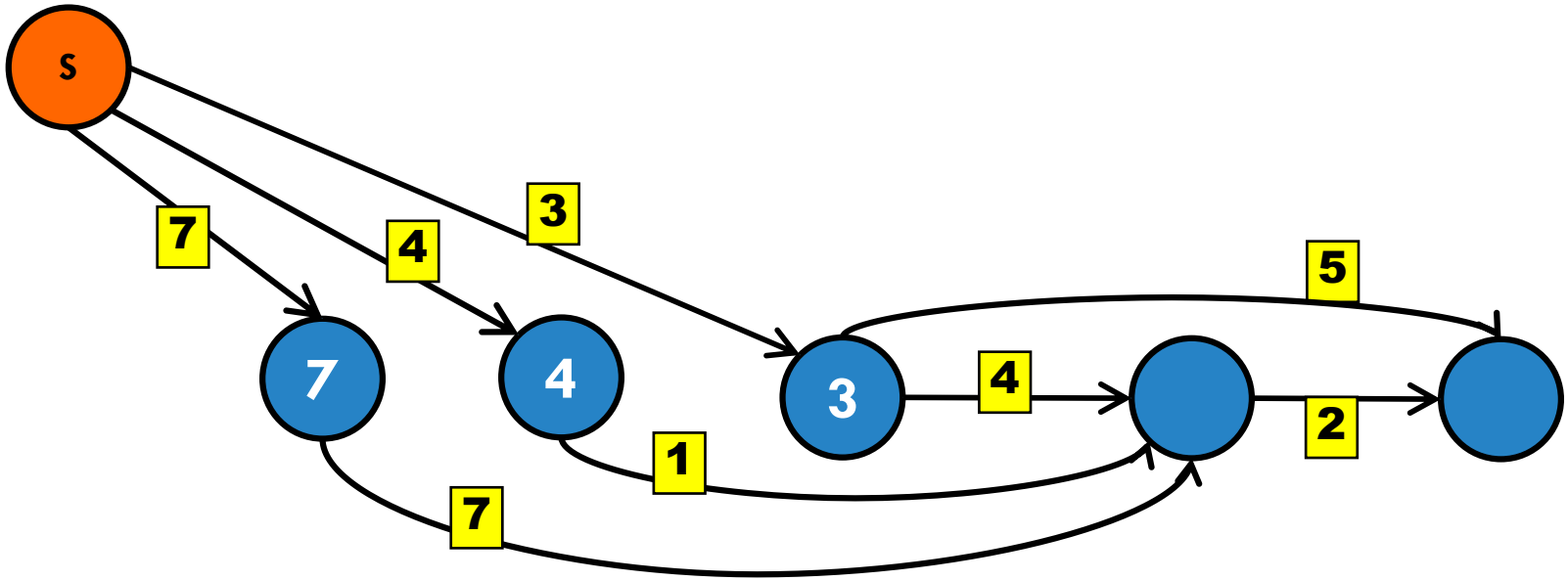
DIRECTED ACYCLIC GRAPH (DAG)

1. Topological sort
2. LongRelax in order.



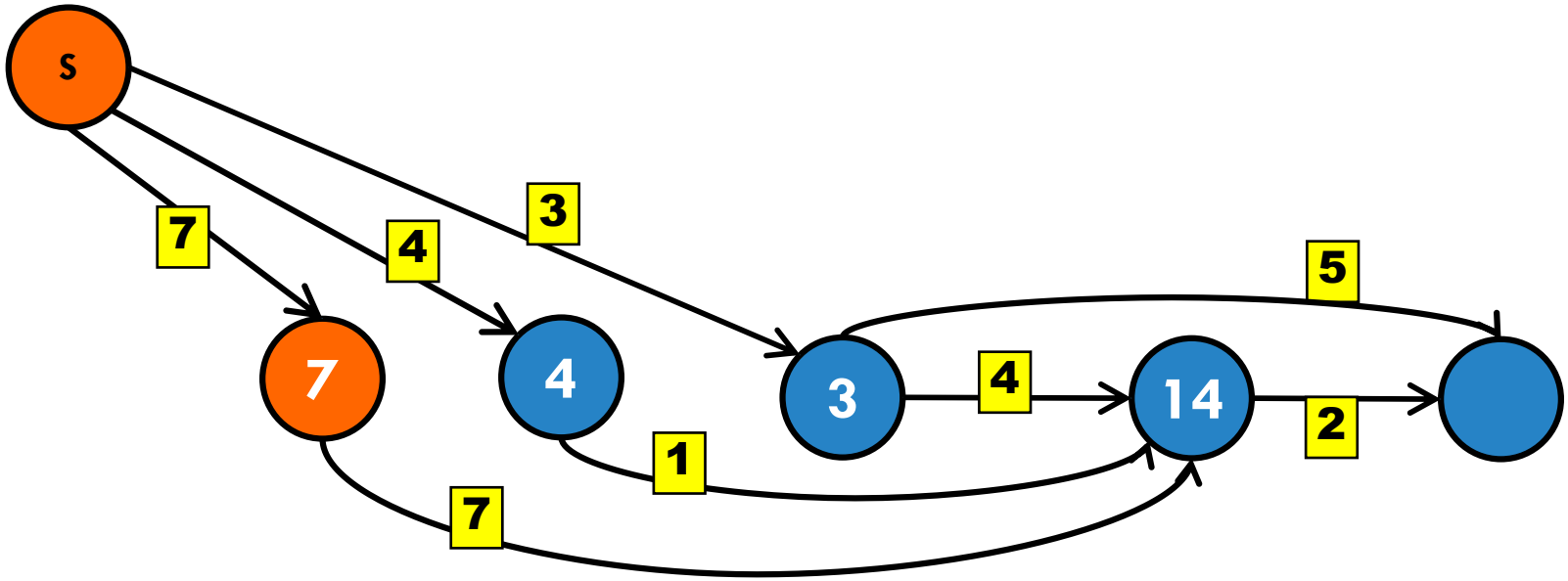
DIRECTED ACYCLIC GRAPH (DAG)

1. Topological sort
2. LongRelax in order.



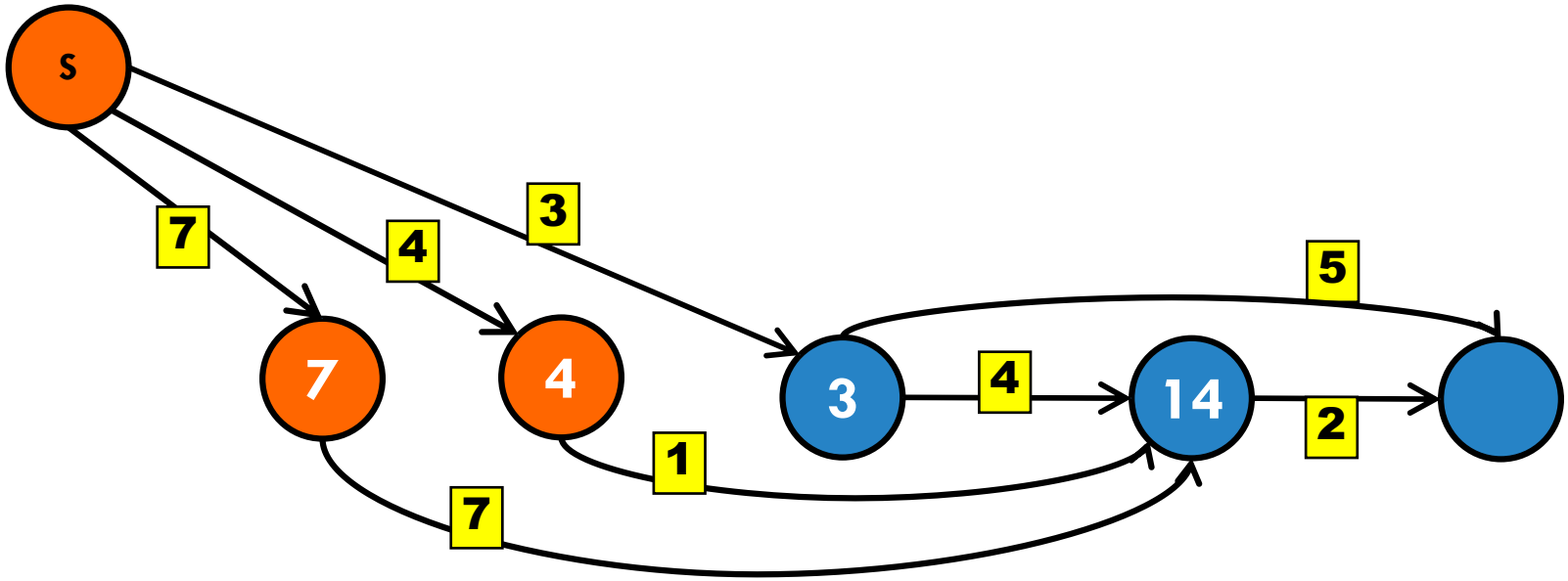
DIRECTED ACYCLIC GRAPH (DAG)

1. Topological sort
2. LongRelax in order.



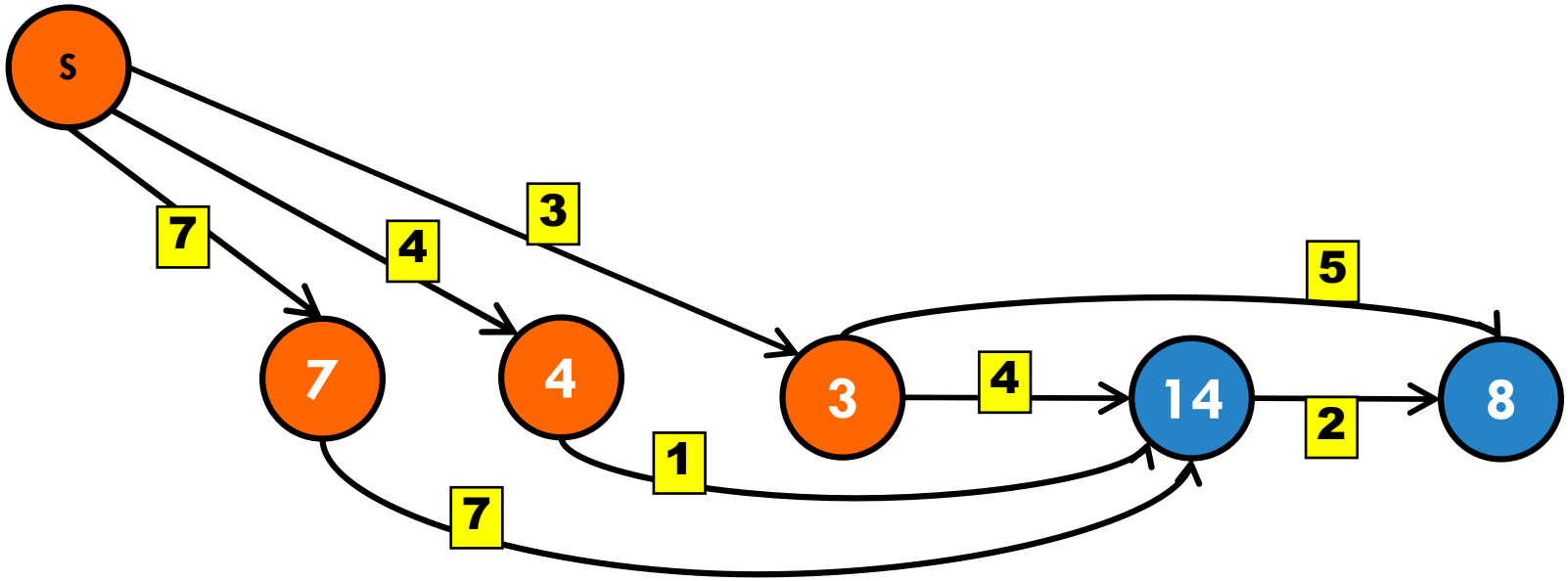
DIRECTED ACYCLIC GRAPH (DAG)

1. Topological sort
2. LongRelax in order.



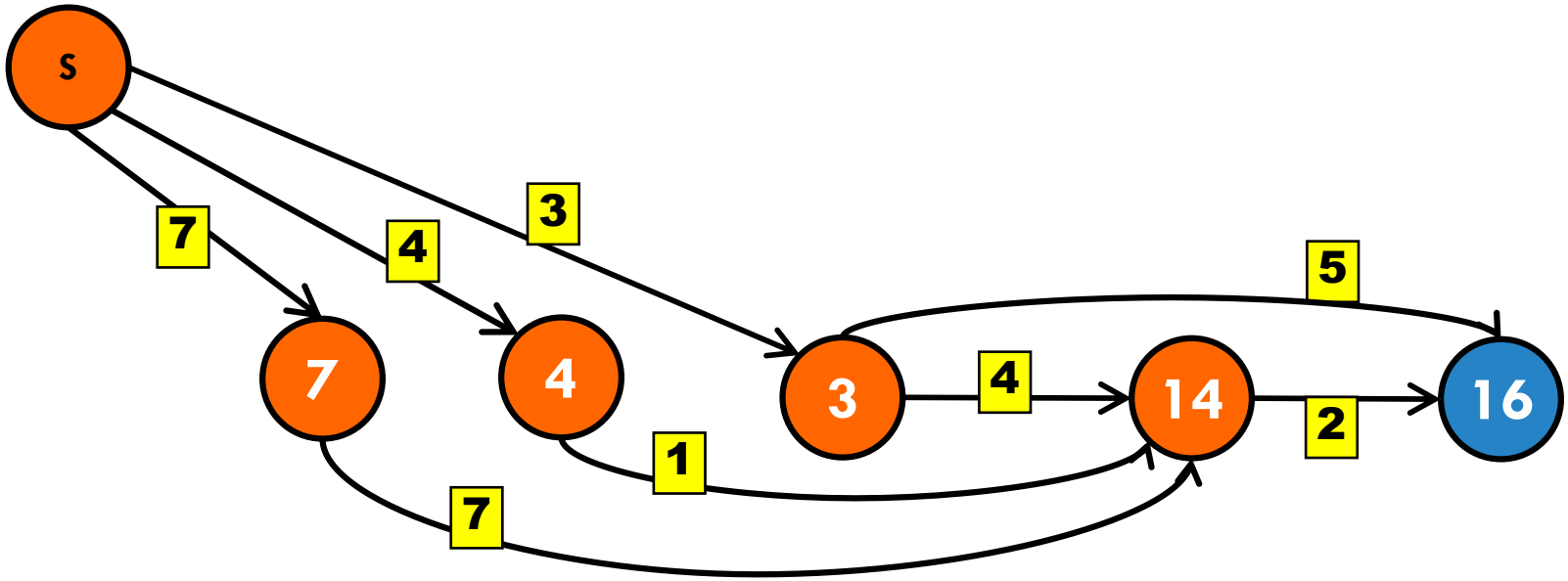
DIRECTED ACYCLIC GRAPH (DAG)

1. Topological sort
2. LongRelax in order.



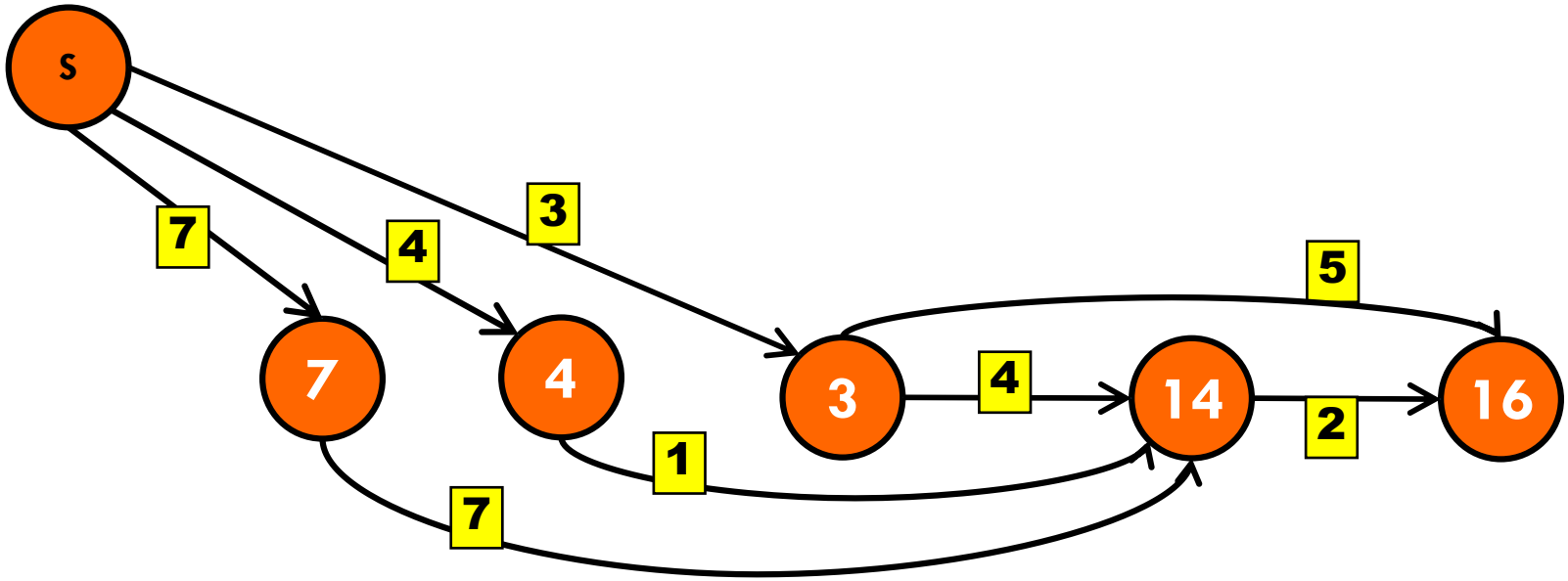
DIRECTED ACYCLIC GRAPH (DAG)

1. Topological sort
2. LongRelax in order.



DIRECTED ACYCLIC GRAPH (DAG)

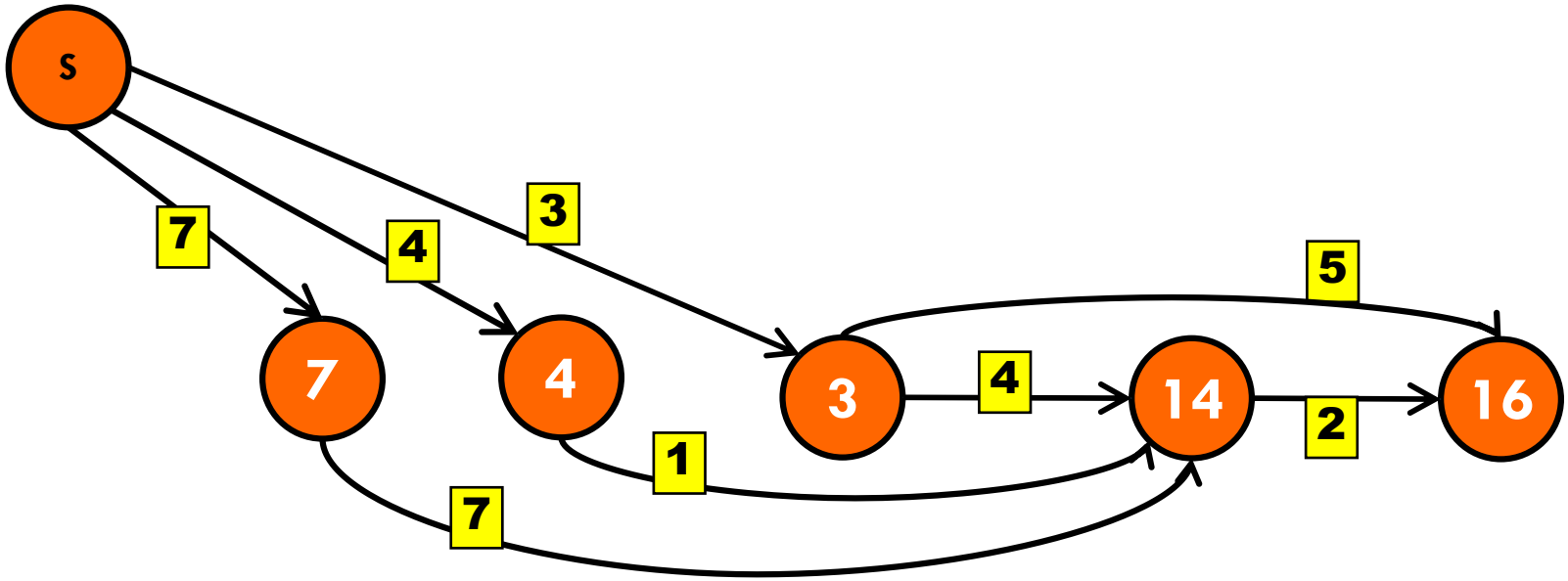
1. Topological sort
2. LongRelax in order.



DIRECTED ACYCLIC GRAPH (DAG)

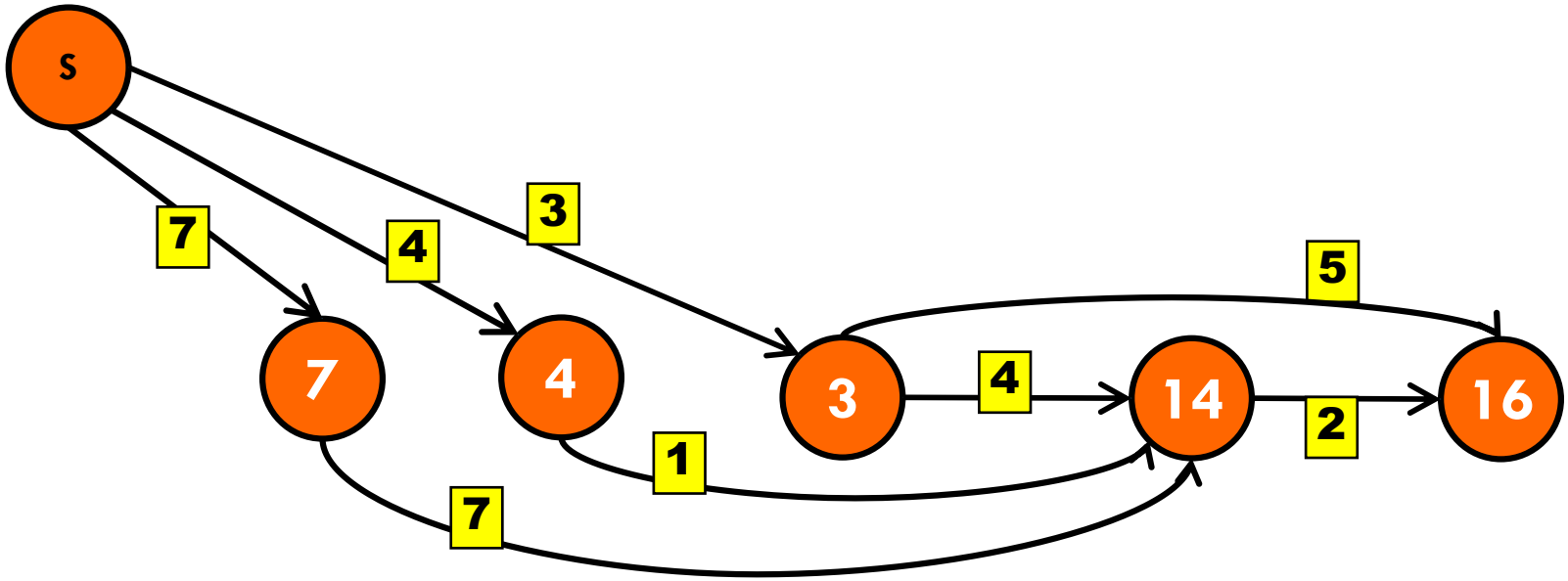
Why does this work?

Because to get from s to any other node v, you need to go through its "predecessors" (in the toposort order).



HOW **LONG** DOES THIS TAKE?

1. Topological sort
2. LongRelax in order.

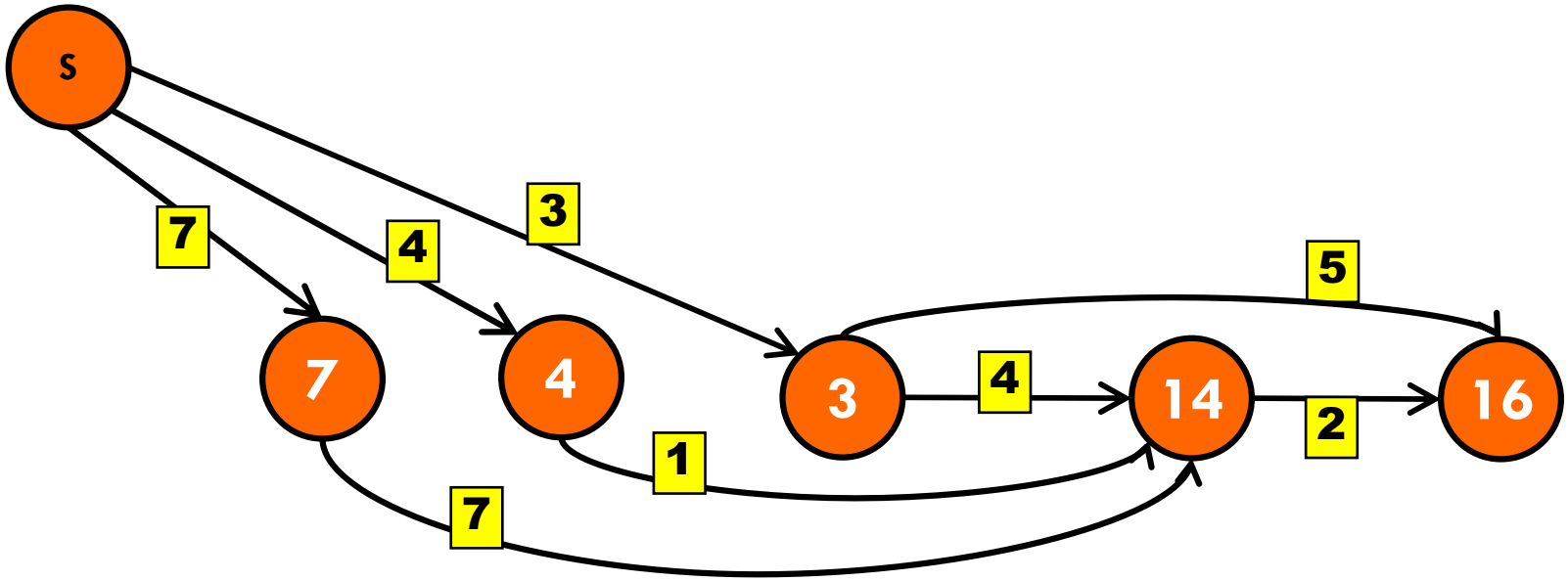


SPECIAL CASES

Condition	Algorithm	Time Complexity
No Negative Weight Cycles	Bellman-Ford Algorithm	$O(VE)$
On Unweighted Graph (or equal weights)	BFS	$O(V+E)$
No Negative Weights	Dijkstra's Algorithm	$O((V+E)log\,V)$
On Tree	BFS / DFS	$O(V)$
On DAG	Topological Sort	$O(V+E)$

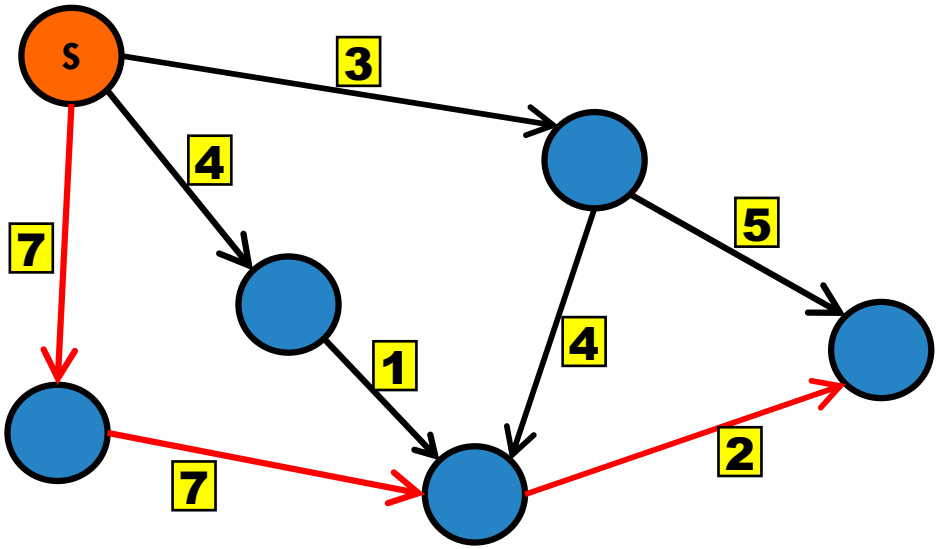
TOTAL TIME: $O(V + E)$

1. Topological sort
2. LongRelax in order.



LONGEST PATH IN A DAG?

Poll Everywhere
https://bit.ly/2LvG9bq

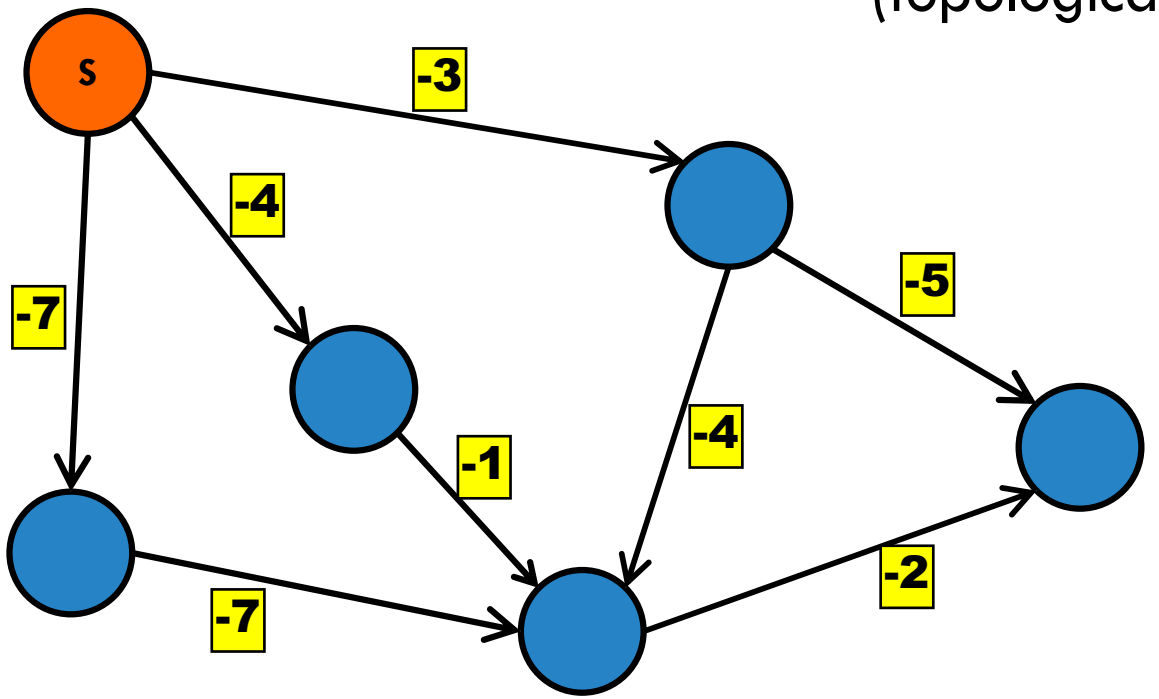


Given a DAG $G=(V,E)$ finding the longest simple path requires:

A. $O(E)$
B. $O(V+E)$
C. $O(E \log V)$
D. $O(V \log E)$
E. $O(VE)$

SOLUTION #2:

Negate the edges! Run shortest path algorithm (topological sort order)



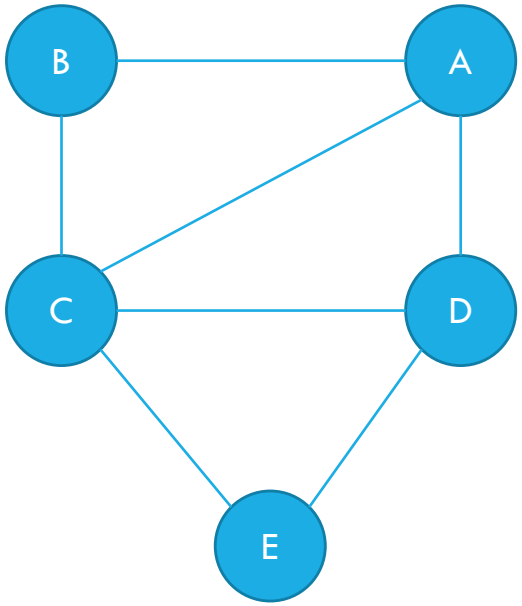
DO THESE SOLUTIONS WORK FOR GRAPHS THAT ARE **NOT** DAGS

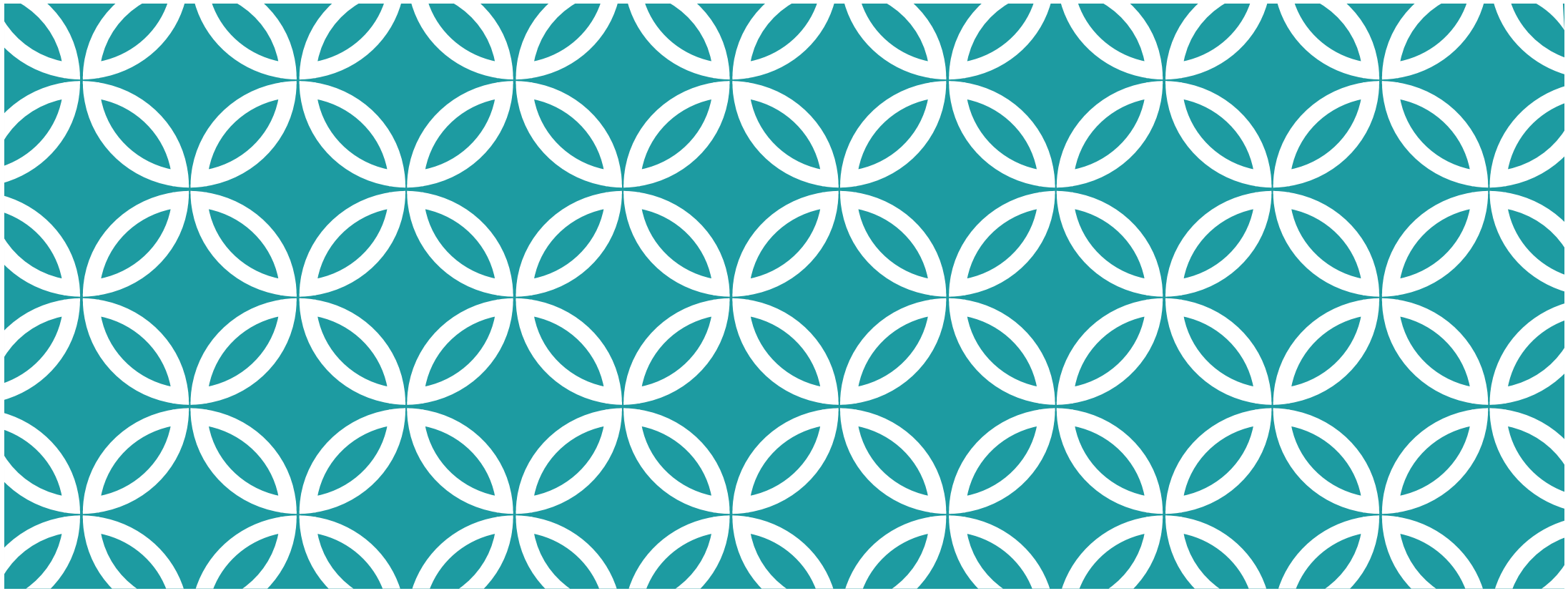
No.

- Negating weights creates negative cycles.
- Finding the longest (or shortest) *simple* path in general graphs NP-hard.

Why?

- We lose the optimal subpaths property when it comes to longest paths.
- What is the longest path from A → E?
- A,B,C,D,E
- The longest path from B → E?
- B,C,A,D,E





SCHEDULING

Harold Soh
harold@comp.nus.edu.sg

SCHEDULING PROBLEM

Given a set of tasks {A, B, C, D, E, F} and a set of constraints:

- A must be completed at least 10 mins before C
- D must be completed at most 20 minutes after E
- B must be done after F

Devise an efficient algorithm to tell if there is a feasible schedule.

Output:

- Feasible?
- Schedule?

FIRST, LET'S MAKE IT MORE SPECIFIC

Inputs: Set of jobs and constraints

```
job id, time needed, {(completed before, time)}
0, 1 min, {(1, 10 mins)}
1, 5 min, {(2, 2 mins), (3, 0 mins)}
...
```

"0 takes 1 min. Must be completed 10 mins before 1"

"1 takes 5 mins. Must be completed 2 mins before 2 and 0 mins before 3".

Outputs:

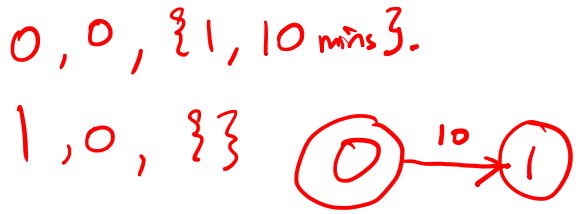
A schedule of **earliest** start times if feasible: (0, 0), (1, 11), (3, 15 mins), (2,17) …

null otherwise

LET'S CREATE A PROBLEM INSTANCE.

job id, time needed, {(completed before, time)}	
0, 1 min, {(1, 10 mins)}	"0 takes 1 min. Must be completed 10 mins before 1"
1, 5 min, {(2, 2 mins), (3, 0 mins)}	"1 takes 5 mins. Must be completed 2 mins before 2 and 0 mins before 3".
2, 7 min, {}	2 takes 7 minutes. No constraints.
3, 1 min	3 takes 1 minute. No constraints.

LET'S CREATE A PROBLEM INSTANCE.



job id, time needed, {(completed before, time)}	
0, 1 min, {(1, 10 mins), (2, 0 mins)}	"0 takes 1 min. Must be completed 10 mins before 1 and before 2"
1, 5 min, {(2, 2 mins), (3, 0 mins)}	"1 takes 5 mins. Must be completed 2 mins before 2 and 0 mins before 3".
2, 7 min, {}	2 takes 7 minutes. No constraints.
3, 1 min	3 takes 1 minute. No constraints.

This is a set of objects and relationships between objects.

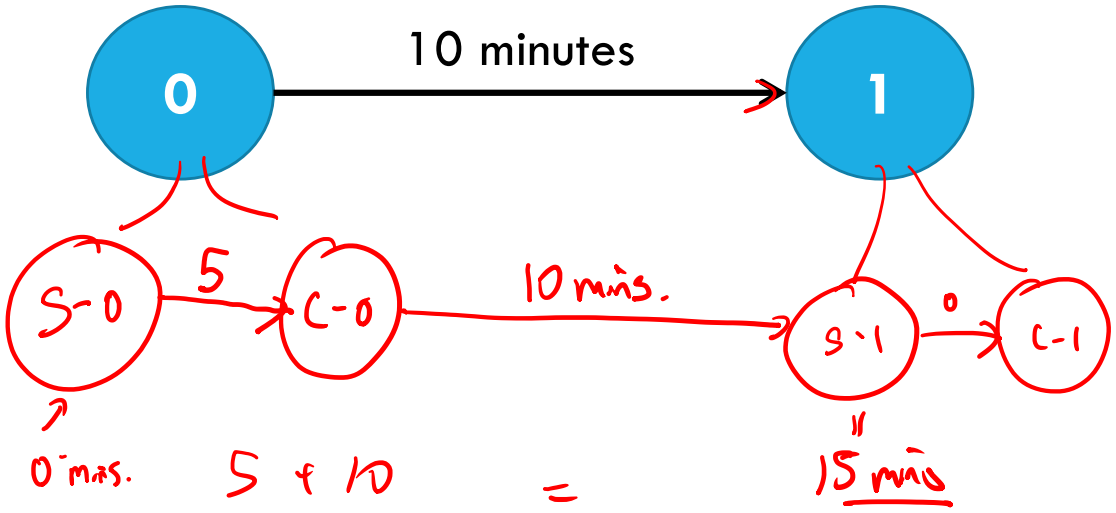
What data structure allows me to model/represent relationships between objects?

Graphs!

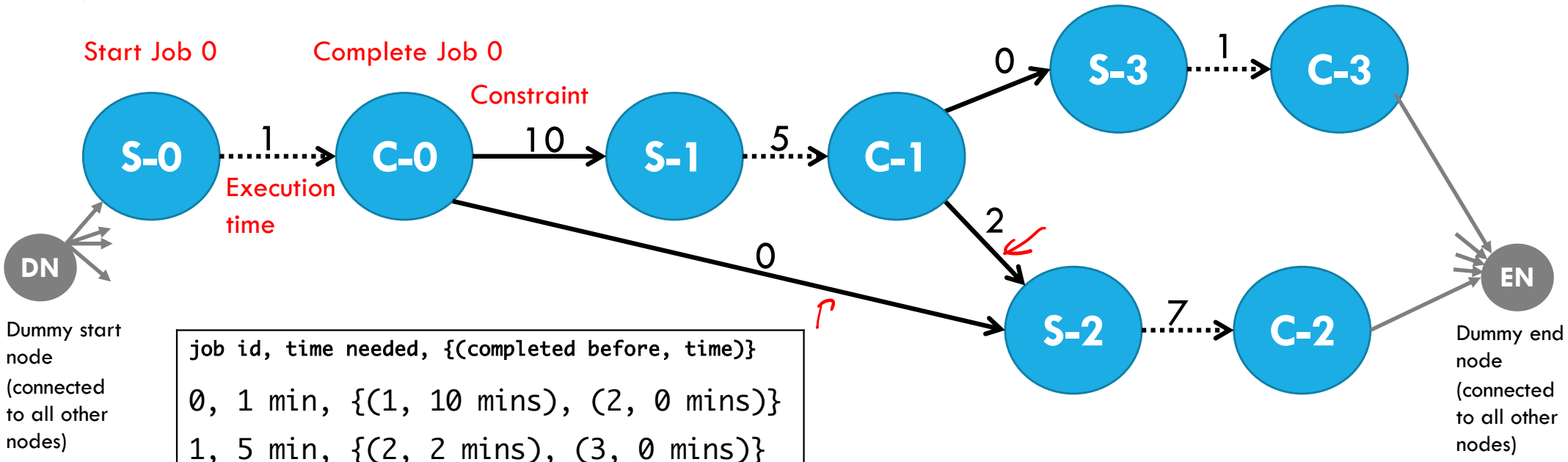
0, 5, {1, 10}.

SCHEDULING PROBLEM

0 must be executed 10 minutes **before 1**



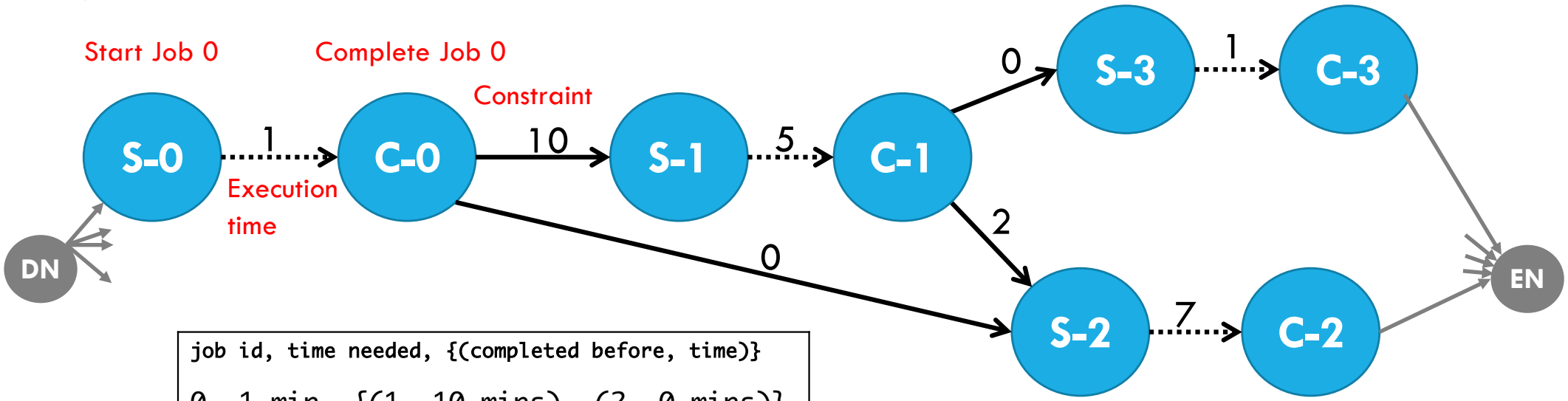
SCHEDULING PROBLEM




```
job id, time needed, {(completed before, time)}
0, 1 min, {(1, 10 mins), (2, 0 mins)}
1, 5 min, {(2, 2 mins), (3, 0 mins)}
2, 7 min, {}
3, 1 min
```

SCHEDULING PROBLEM

We want the EARLIEST start time for each job that satisfies the constraints




```
job id, time needed, {(completed before, time)}
0, 1 min, {(1, 10 mins), (2, 0 mins)}
1, 5 min, {(2, 2 mins), (3, 0 mins)}
2, 7 min, {}
3, 1 min
```

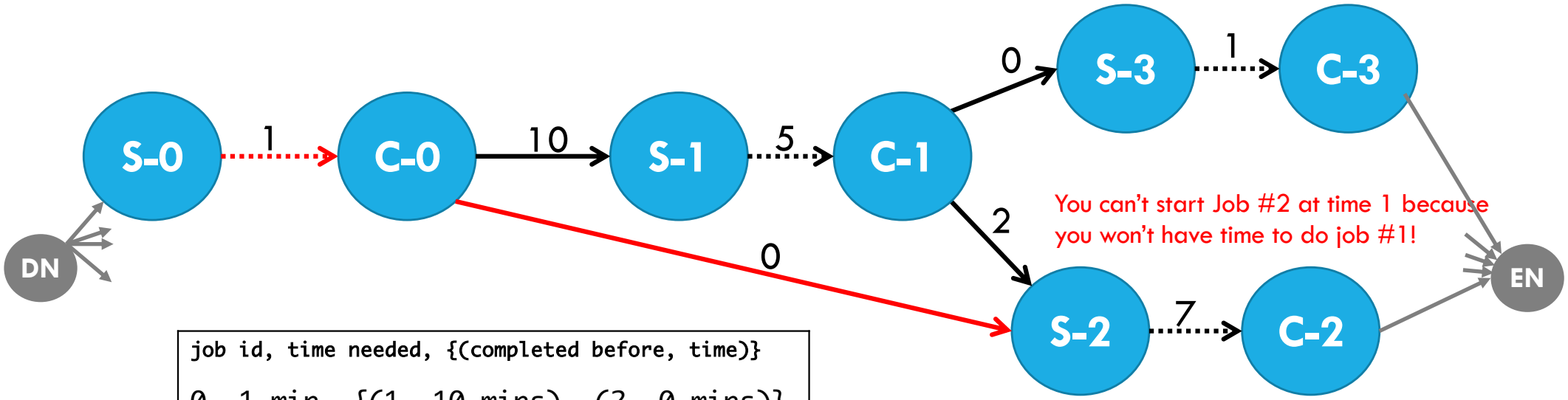
Now what?

A path problem.

Shortest Paths?

SCHEDULING PROBLEM

Shortest Paths won't give the earliest start time because it doesn't satisfy the required constraints.



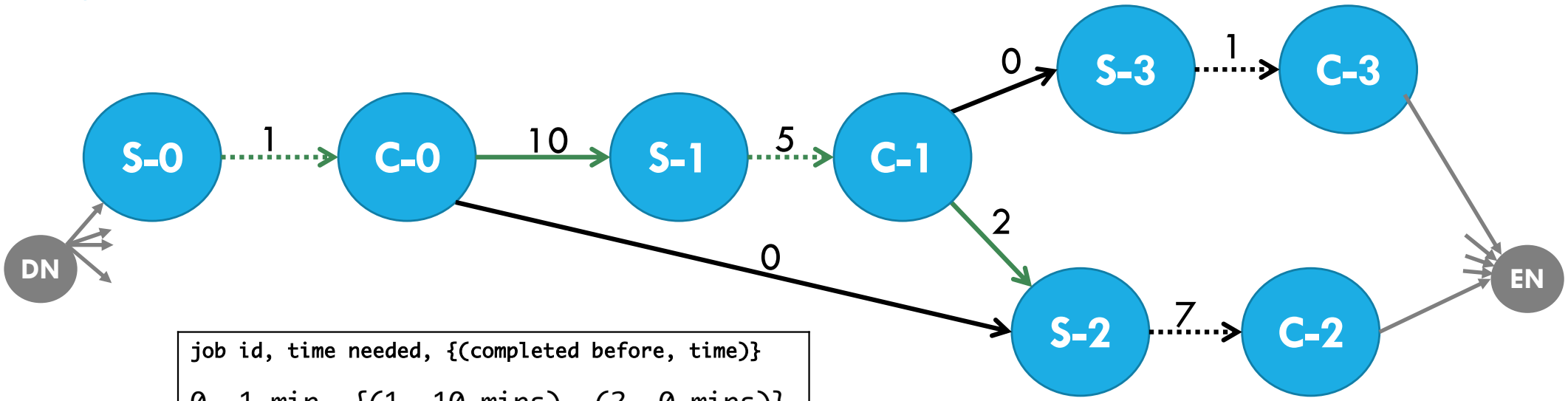

```
job id, time needed, {(completed before, time)}
0, 1 min, {(1, 10 mins), (2, 0 mins)}
1, 5 min, {(2, 2 mins), (3, 0 mins)}
2, 7 min, {}
3, 1 min
```

Now what?

A path problem.

Shortest Paths? No!

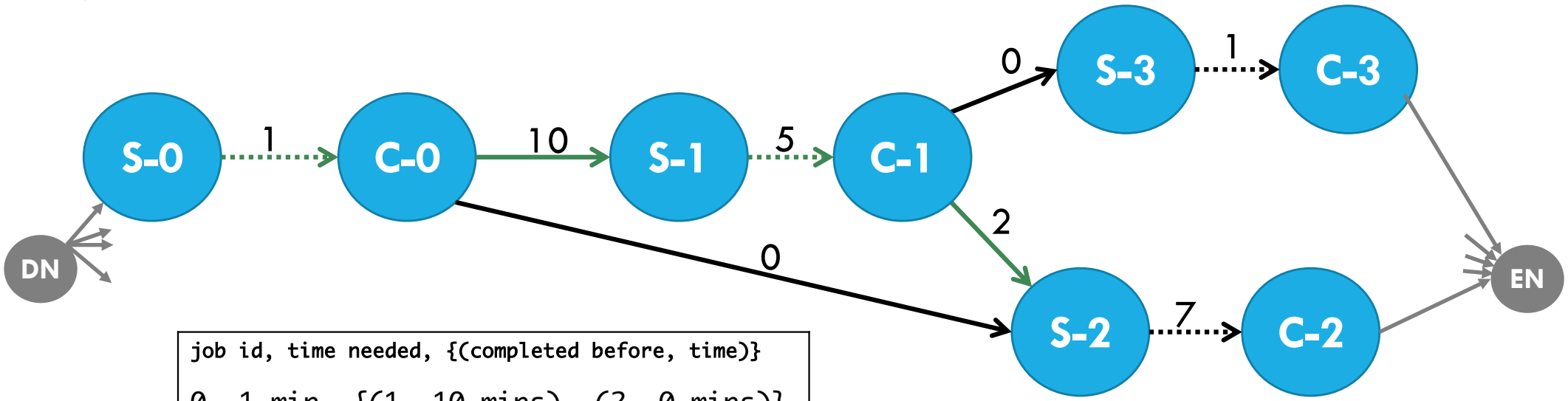
SCHEDULING PROBLEM




```
job id, time needed, {(completed before, time)}
0, 1 min, {(1, 10 mins), (2, 0 mins)}
1, 5 min, {(2, 2 mins), (3, 0 mins)}
2, 7 min, {}
3, 1 min
```

Want the longest path to determine the start time. How to find longest path?

SCHEDULING PROBLEM



job id, time needed, {(completed before, time)}

0, 1 min, {(1, 10 mins), (2, 0 mins)}

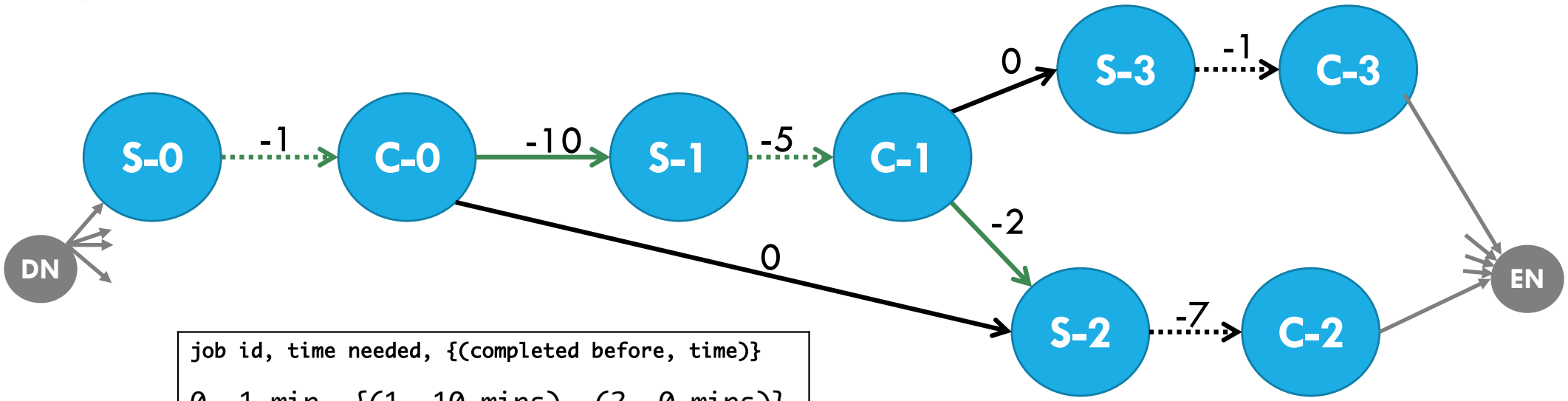
1, 5 min, {(2, 2 mins), (3, 0 mins)}

2, 7 min, {}

3, 1 min

Idea: Negate all the edges & run shortest paths!

SCHEDULING PROBLEM



job id, time needed, {(completed before, time)}

0, 1 min, {(1, 10 mins), (2, 0 mins)}

1, 5 min, {(2, 2 mins), (3, 0 mins)}

2, 7 min, {}

3, 1 min

Idea: Negate all the edges & run shortest paths!

Negative Edges: Bellman-Ford!

SCHEDULING PROBLEM

Given a set of tasks {A, B, C, D, E, F} and a set of constraints:

- A must be completed at least 10 mins before C
- D must be completed at most 20 minutes after E
- B must be done after F

Devise an efficient algorithm to tell if there is a feasible schedule.

- Shortest path (with negative edges) guarantees constraints are met.
- If takes longer than $|V| - 1$, negative cycles exist.
- If you are guaranteed the constraints form a DAG, you can use topological sort order.

QUESTIONS?

Poll Everywhere

https://bit.ly/2LvG9bq