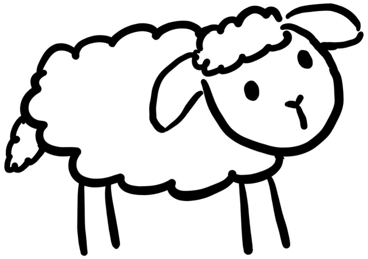
# Greedy Algorithms

# A question to ponder…

Why doesn't the trick of computing $\text{dist}^{[i]}(s,t)$ give a faster-than-BF algorithm for SSSP too?

# Progress on APSP since Floyd-Warshall

| Author | Runtime | Year |
|---|---|---|
| Fredman | $n^3 \log\log^{1/3} n / \log^{1/3} n$ | 1976 |
| Takaoka | $n^3 \log\log^{1/2} n / \log^{1/2} n$ | 1992 |
| Dobosiewicz | $n^3 / \log^{1/2} n$ | 1992 |
| Han | $n^3 \log\log^{5/7} n / \log^{5/7} n$ | 2004 |
| Takaoka | $n^3 \log\log^2 n / \log n$ | 2004 |
| Zwick | $n^3 \log\log^{1/2} n / \log n$ | 2004 |
| Chan | $n^3 / \log n$ | 2005 |
| Han | $n^3 \log\log^{5/4} n / \log^{5/4} n$ | 2006 |
| Chan | $n^3 \log\log^3 n / \log^2 n$ | 2007 |
| Han, Takaoka | $n^3 \log\log n / \log^2 n$ | 2012 |
| Williams | $n^3 / \exp(\sqrt{\log n})$ | 2014 |

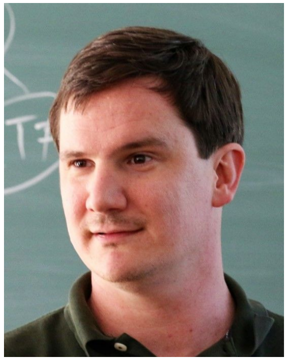

Conclusion: Maybe $O(n^{2.99})$ is impossible?

# Maybe $O(n^{2.99})$ is impossible?

Either ALL of the following have $O(n^{<3})$ time algorithms or NONE of them do: (Virginia Vassilevska Williams, Ryan Williams, 2010)

1. APSP
2. Minimum Weight Triangle
3. Metricity
4. Minimum Cycle
5. Distance Product
6. Second Shortest Path
7. Replacement Paths
8. Negative Triangle Listing

...



# Greedy Algorithms

Pick the best choice NOW.

Prove you end up with an optimal solution.


Proof Technique: Induction + “Exchange” argument

# Warning: Greed is generally bad!

Greed

- Fast
- doesn't work for most problems

Divide and conquer

- Often slower than greed and faster than DP
- works when solutions to disjoint subproblems can be combined into final solution

Dynamic programming

- Generally slower (but still usually efficient)
- applies to many problems

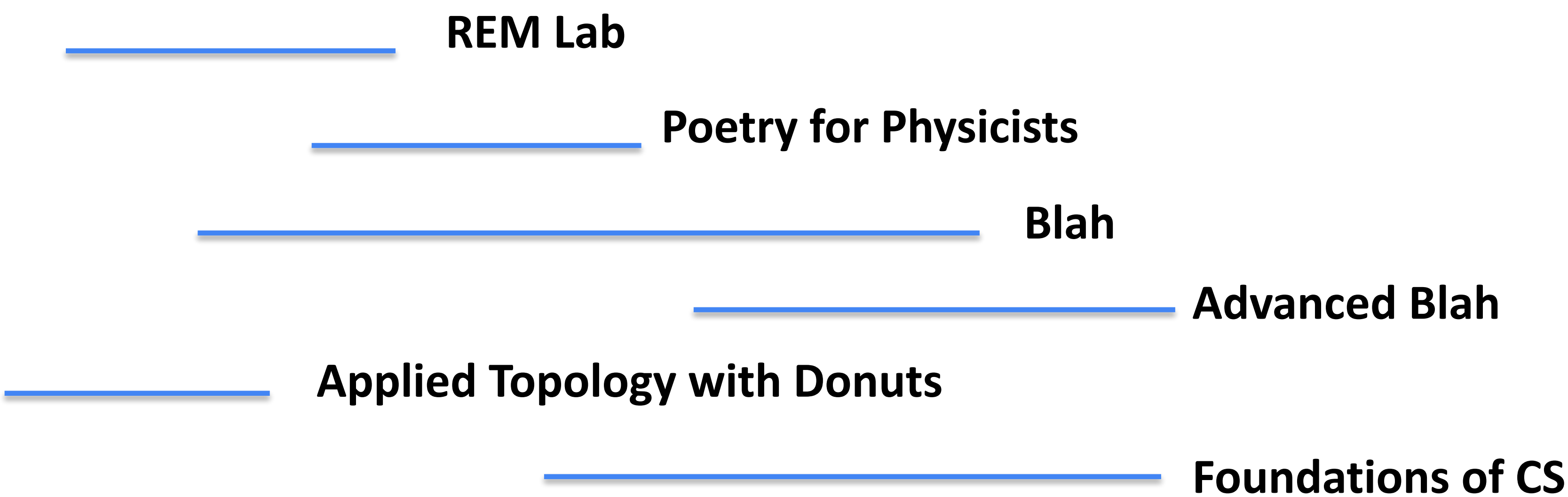
⋮

harder problems where none of these methods apply (coming soon)

But sometimes greed can be good…

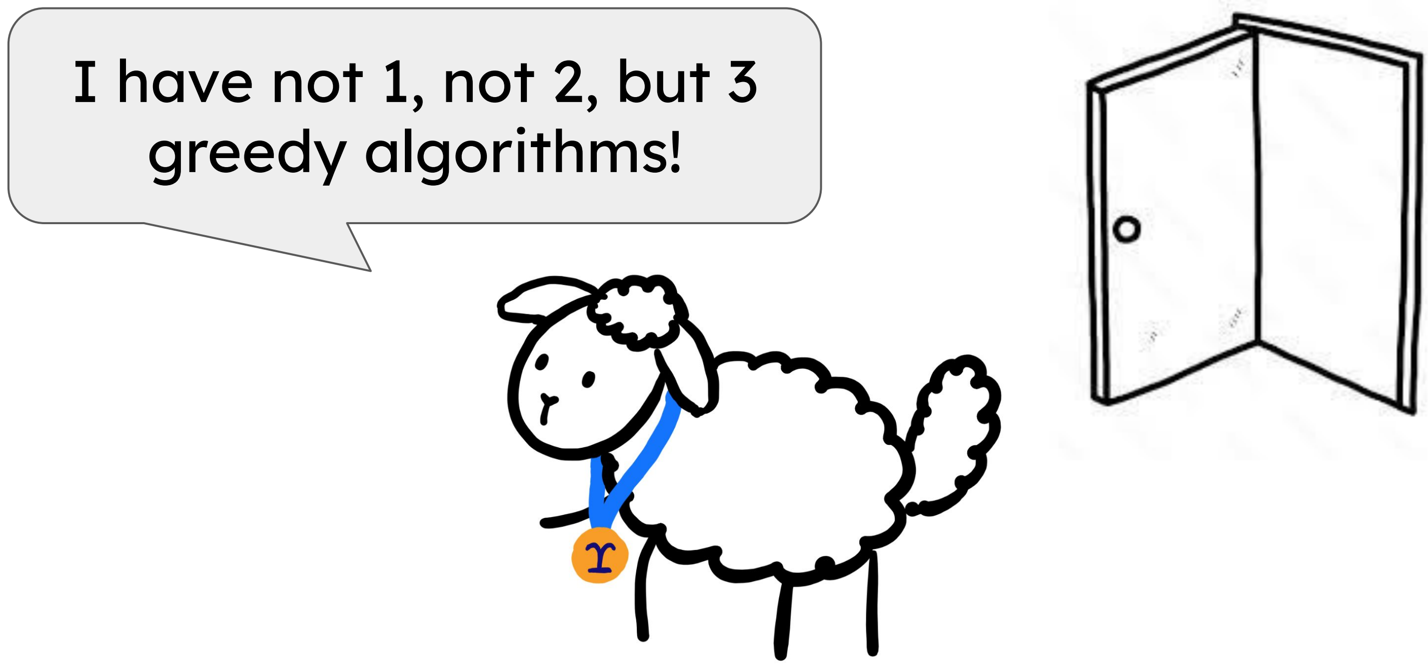
# Unweighted Course Registration

## (aka Task Scheduling)

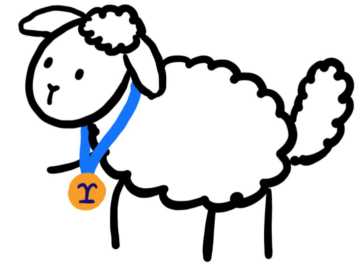



**Goal:** Choose a **largest**[*] possible set of non-intersecting courses (there may be many optimal solutions, we just seek one!)

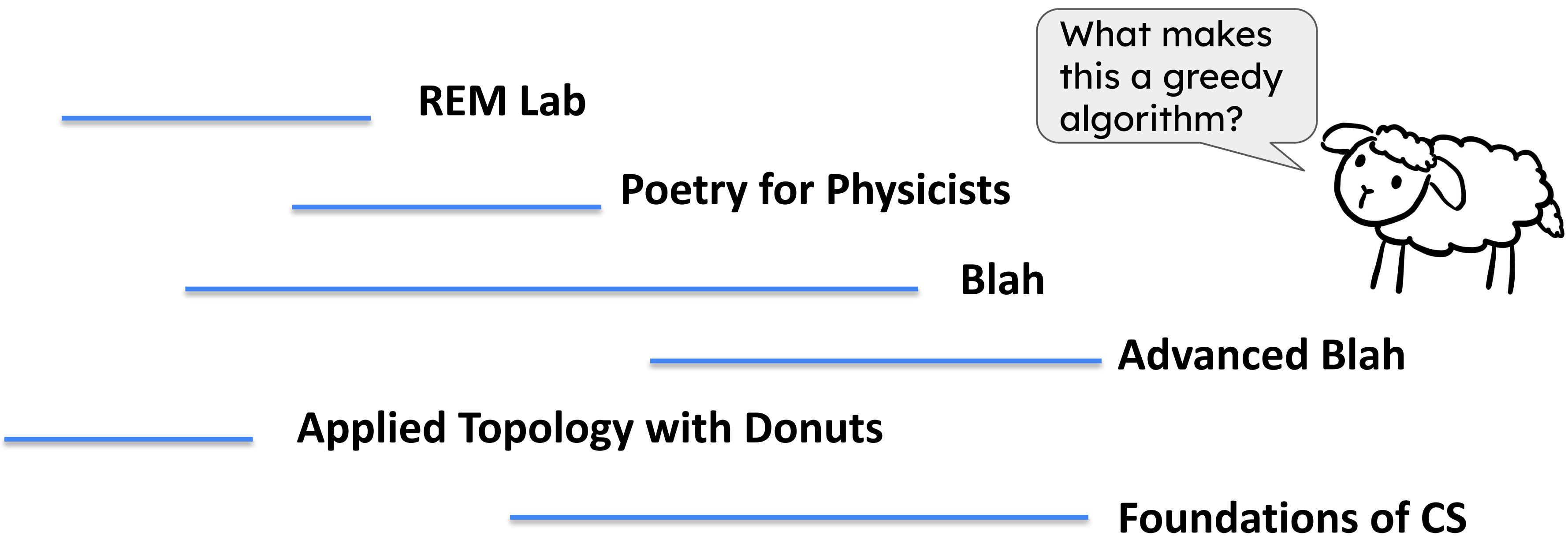
*We do not endorse this course registration strategy


I have not 1, not 2, but 3 greedy algorithms!

# Professor ϒ's Greedy Algorithms

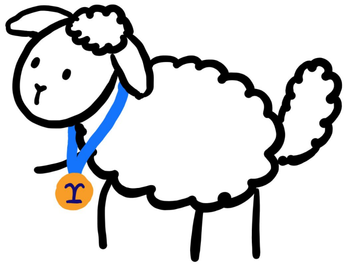


Attempt 1: Choose the **shortest interval** (breaking ties arbitrarily), take it, remove overlaps, recurse on remaining problem.




Counterexample:

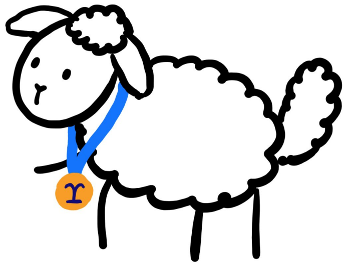
# Professor ϒ's Greedy Algorithms



Attempt 2: Choose the **interval that starts earliest** (breaking ties arbitrarily), take it, remove overlaps, recurse on remaining problem.

Counterexample:

# Professor ϒ's Greedy Algorithms



Attempt 3: Choose the **interval that overlaps with the fewest other intervals** (breaking ties arbitrarily), take it, remove overlaps, recurse on remaining problem.

Counterexample:

# A Correct Greedy Algorithm:
# “Earliest Ending Time (EET)”

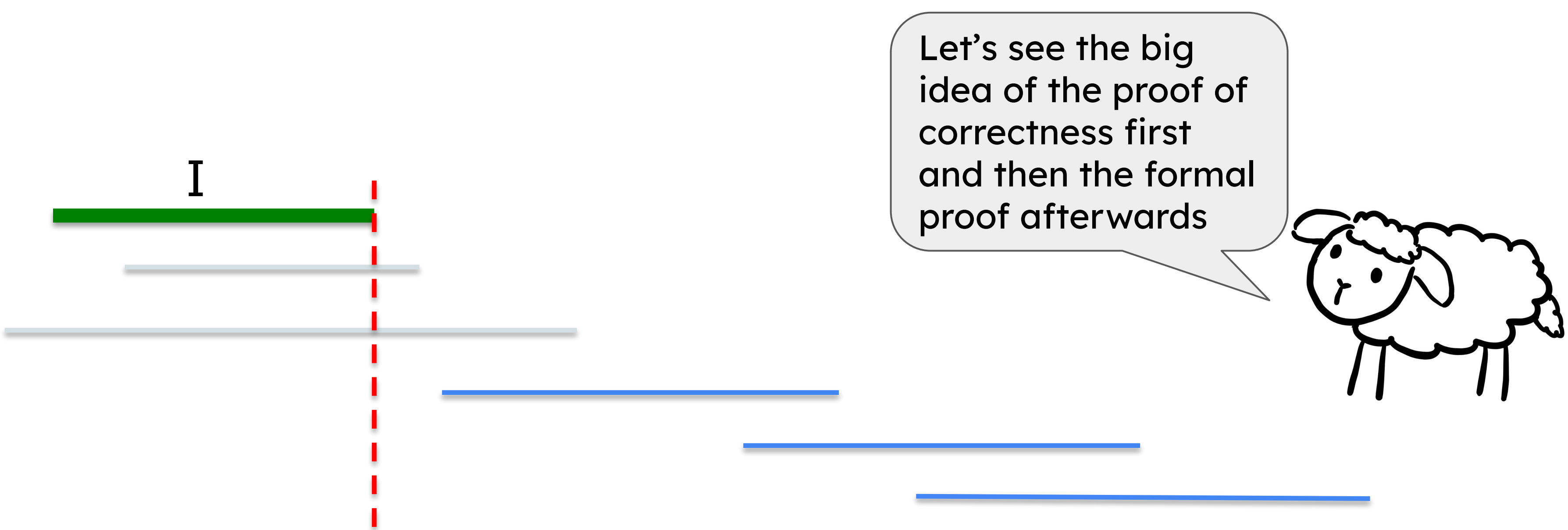
- Sort the intervals by *ending time*
- Greedily take the **interval** $I$ **that ends first** (break ties arbitrarily)
- Remove the intervals that overlap with the one just selected
- Recurse on remaining problem

# A Correct Greedy Algorithm: "Earliest Ending Time (EET)"

- Sort the intervals by *ending time*
- Greedily take the **interval** $\mathrm{I}$ **that ends first** (break ties arbitrarily)
- Remove the intervals that overlap with the one just selected
- Recurse on remaining problem

# A Correct Greedy Algorithm:
# “Earliest Ending Time (EET)”

- Sort the intervals by *ending time*
- Greedily take the **interval** I **that ends first** (break ties arbitrarily)
- Remove the intervals that overlap with the one just selected
- Recurse on remaining problem



# A Correct Greedy Algorithm: “Earliest Ending Time (EET)”

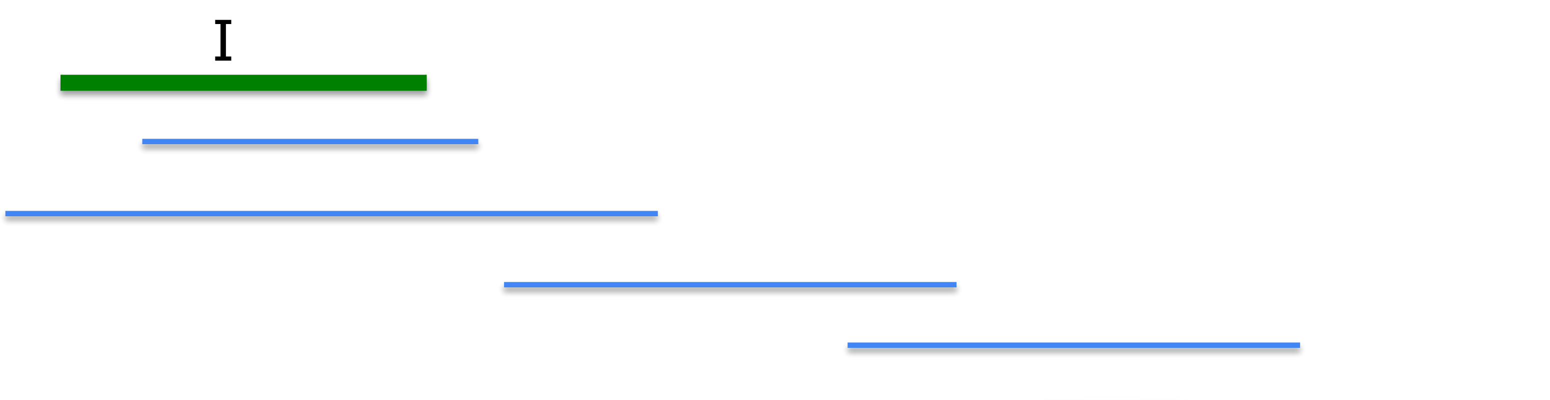
**Key Claim:** The interval $I$ that ends first is a **safe** choice i.e. it is in *some* optimal solution.

**Why?** Consider an optimal solution OPT. Let $I_{OPT}$ be the interval that ends first in OPT.

→ $I_{OPT}$ ends at least as late as $I$.

→ All other intervals in OPT start after $I_{OPT}$ ends, and thus after $I$ ends.

→ Thus, we can take OPT and **exchange** $I_{OPT}$ for $I$ and get a valid solution of the same size!



# Formal Proof of Correctness by Induction

intervals in order of addition (so, increasing end times)

Let $I_1, I_2, I_3, \ldots$ be the output of the EET algorithm

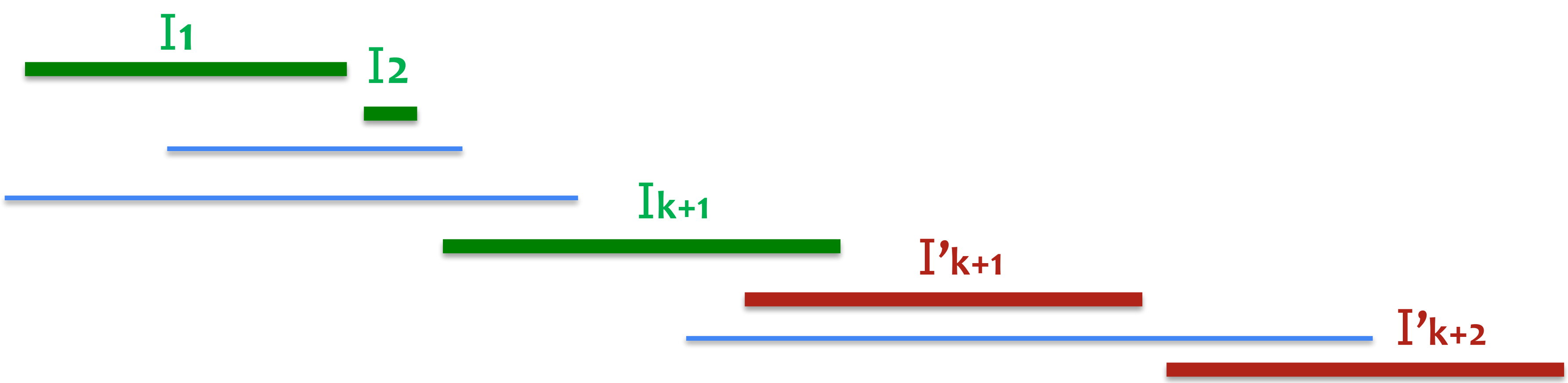
**Goal:** Prove that for all **k**, $I_1, I_2, I_3, \ldots, I_k$ is in <u>some</u> optimal solution.

*Proof by induction on **k**:*

**Base case:** k=0.

**Inductive hypothesis:** Suppose $I_1, I_2, I_3, \ldots, I_k$ is in <u>some</u> optimal solution $I_1, I_2, I_3, \ldots, I_k, I'_{k+1}, I'_{k+2}, \ldots$

**Inductive step:** Goal: show $I_1, I_2, I_3, \ldots, I_{k+1}$ is in <u>some</u> optimal solution.



# Formal Proof of Correctness by Induction

**Inductive step:** Goal: show $I_1, I_2, I_3, \ldots, I_{k+1}$ is in <u>some</u> optimal solution.
By the inductive hypothesis, there <u>exists</u> an optimal solution:

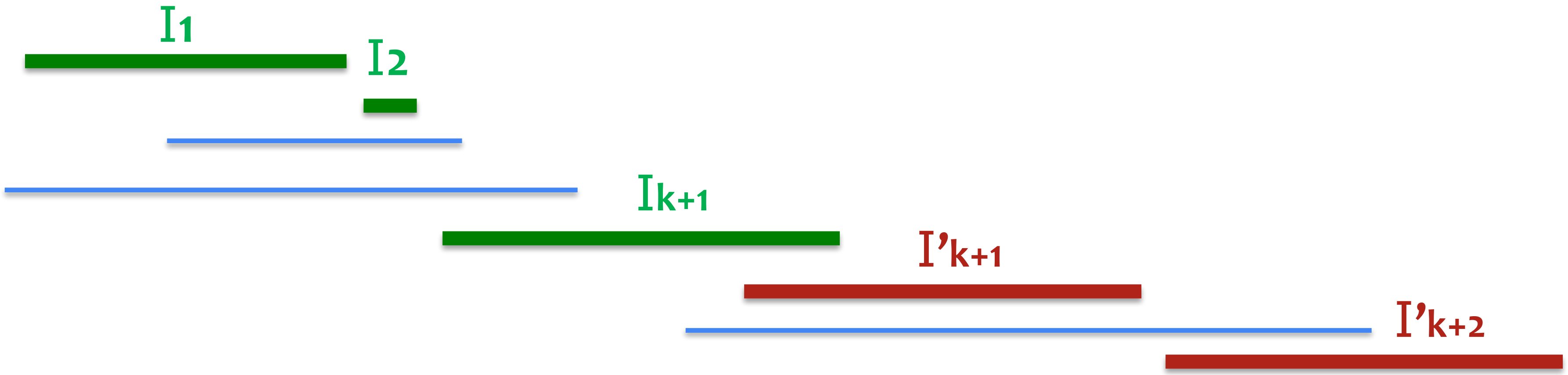
$$\text{OPT} = I_1, I_2, I_3, \ldots, I_k, I'_{k+1}, I'_{k+2}, \ldots$$

Use an **exchange** argument as before!

➔ $I'_{k+1}$ ends at least as late as $I_{k+1}$.

➔ All other intervals $I'_{k+2}, \ldots$ start after $I'_{k+1}$ ends, and thus after $I_{k+1}$ ends.

➔ Thus, we can take OPT and **exchange** $I'_{k+1}$ for $I_{k+1}$ and get a valid solution of the same size!



# General Strategy commonly used for analyzing greedy algorithms:

Proof by induction using an "**exchange**" argument

**The idea:** Show that we can transform any **optimal solution** into the **solution given by our algorithm** by **exchanging** each piece of it out one-by-one without increasing the cost.

**Key part of proof:** Show that my greedy choice is **safe** i.e. it is in some optimal solution.

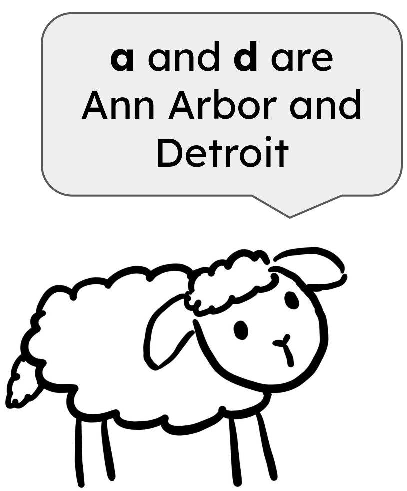
Induction just formalizes the idea that *each successive choice* is **safe**.

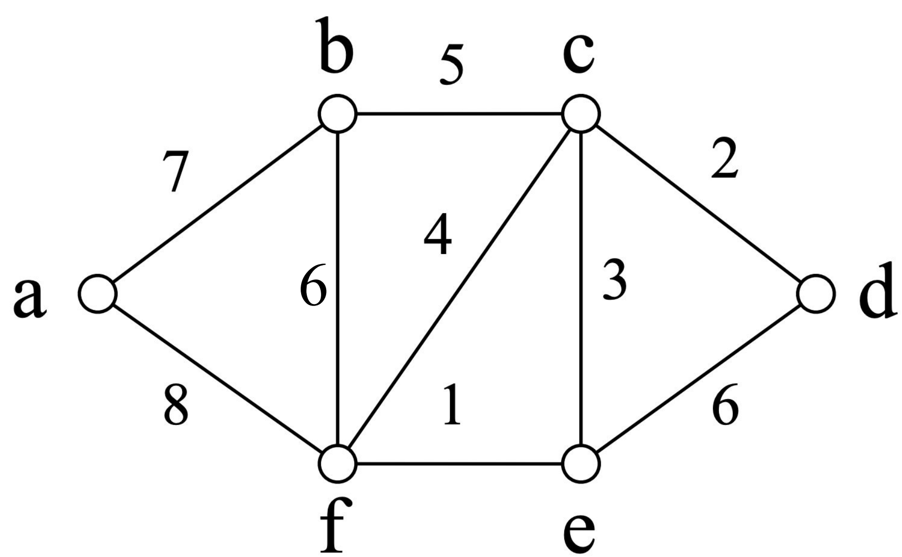
Another problem where greed is good…

# A Highway Problem

**Input:** an undirected graph with positive edge weights
e.g. a set of cities and distances between them

**Output:** *minimum* total length of highway to *connect* all cities
i.e. it should be possible to drive from any city to any other using just the highways



Another problem where greed is good…

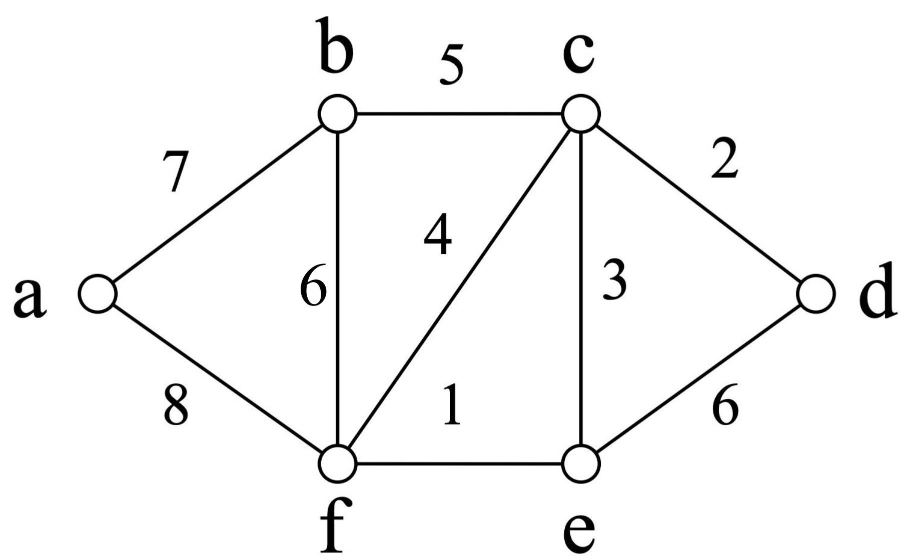
# A Highway Problem

This fact will be useful later too

**Claim:** Solution is a tree.

Why? If a connected graph has a cycle, we can delete an arbitrary edge from the cycle and still have a connected graph.

**Definition** (review): A tree is a connected graph with no cycles.

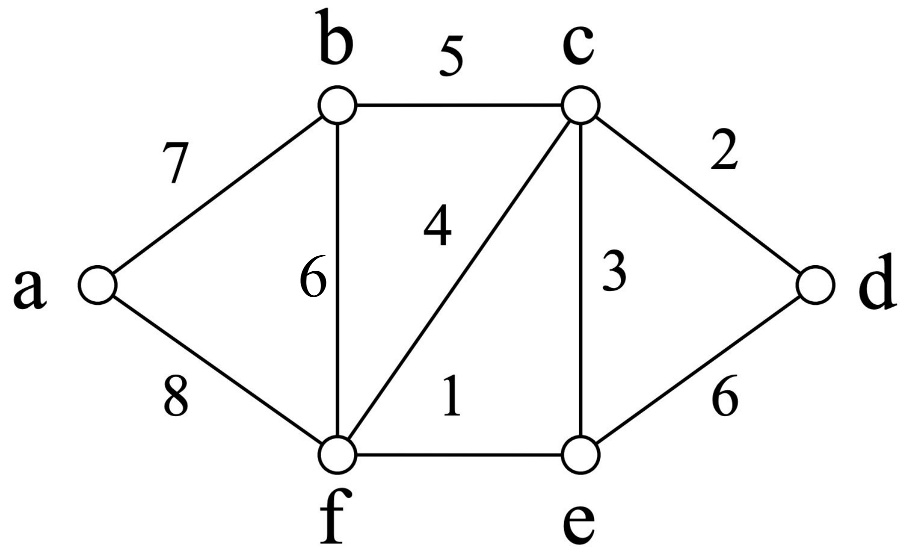


The Highway Problem is commonly known as:

# Minimum Spanning Tree (MST)

Given a graph G, a **spanning tree** is a subgraph of G that is spanning (uses all vertices) and is a tree.

**MST problem:** Given an undirected graph with positive edge weights, find a minimum weight spanning tree.

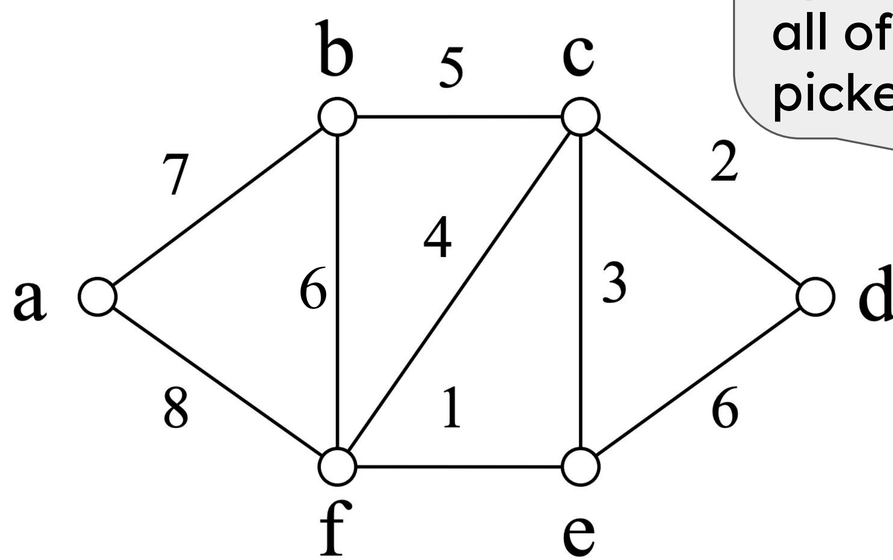


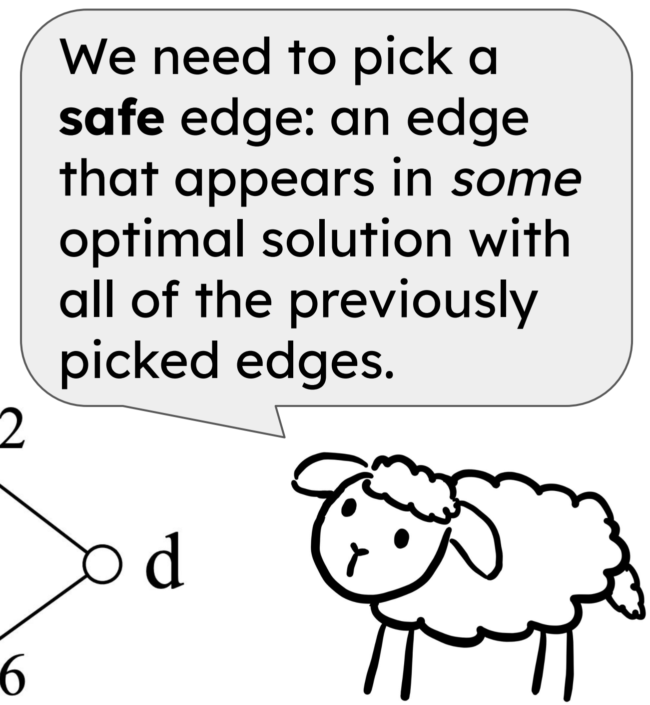
# Minimum Spanning Tree (MST)

**Template for greedy algorithm:**

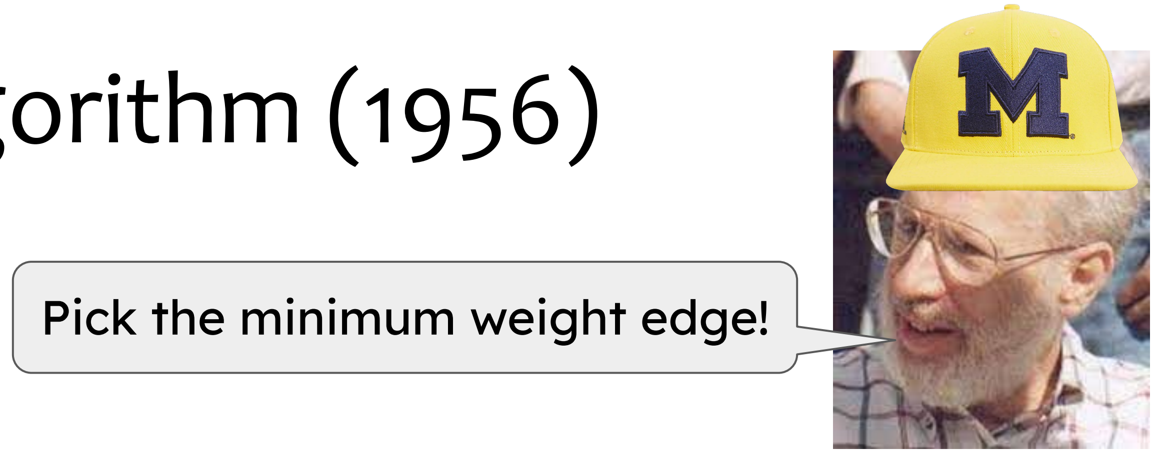
Greedily pick an edge and add it to our MST. Repeat.

But which edge do we pick?



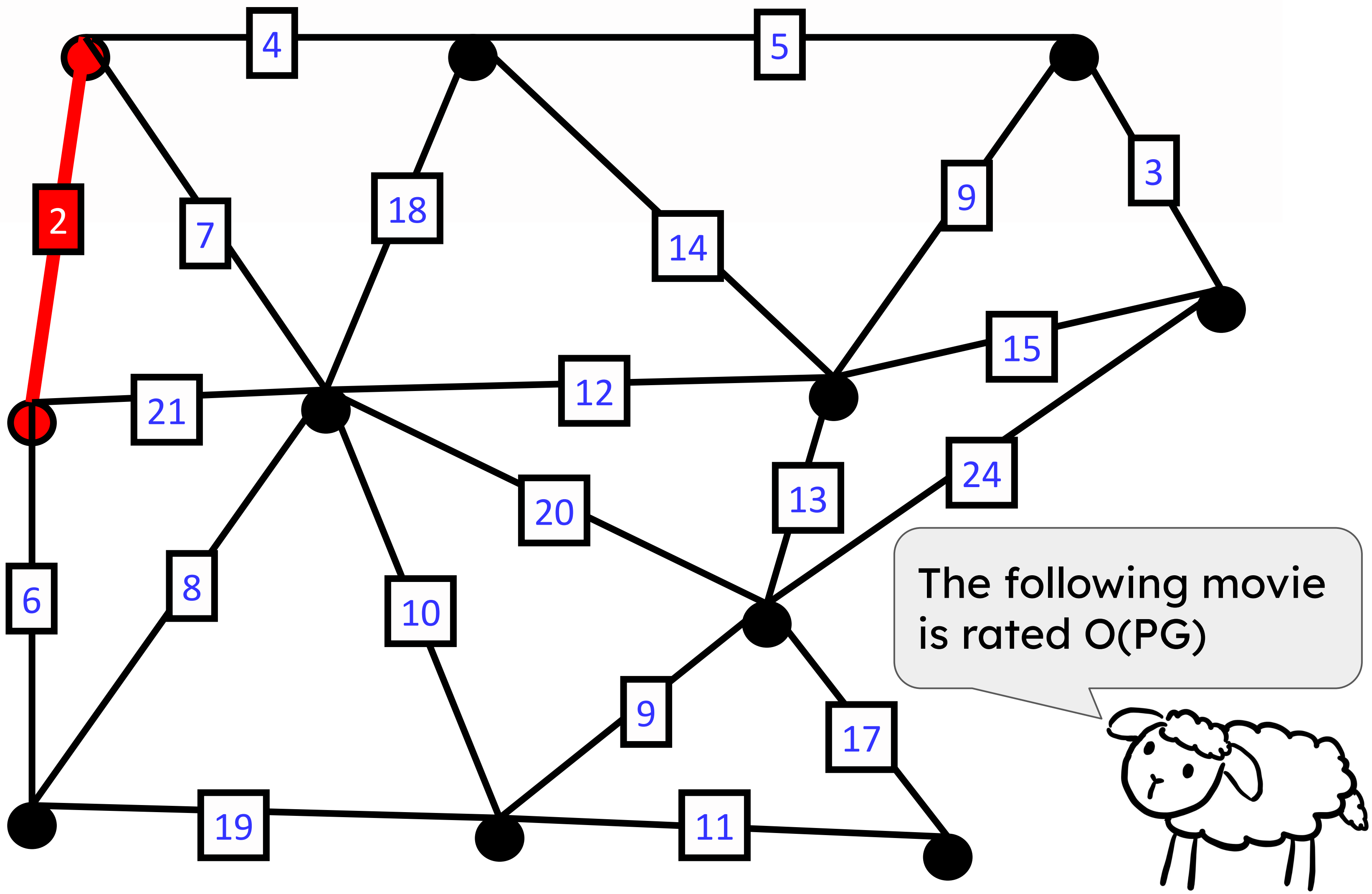



# Kruskal's Algorithm (1956)

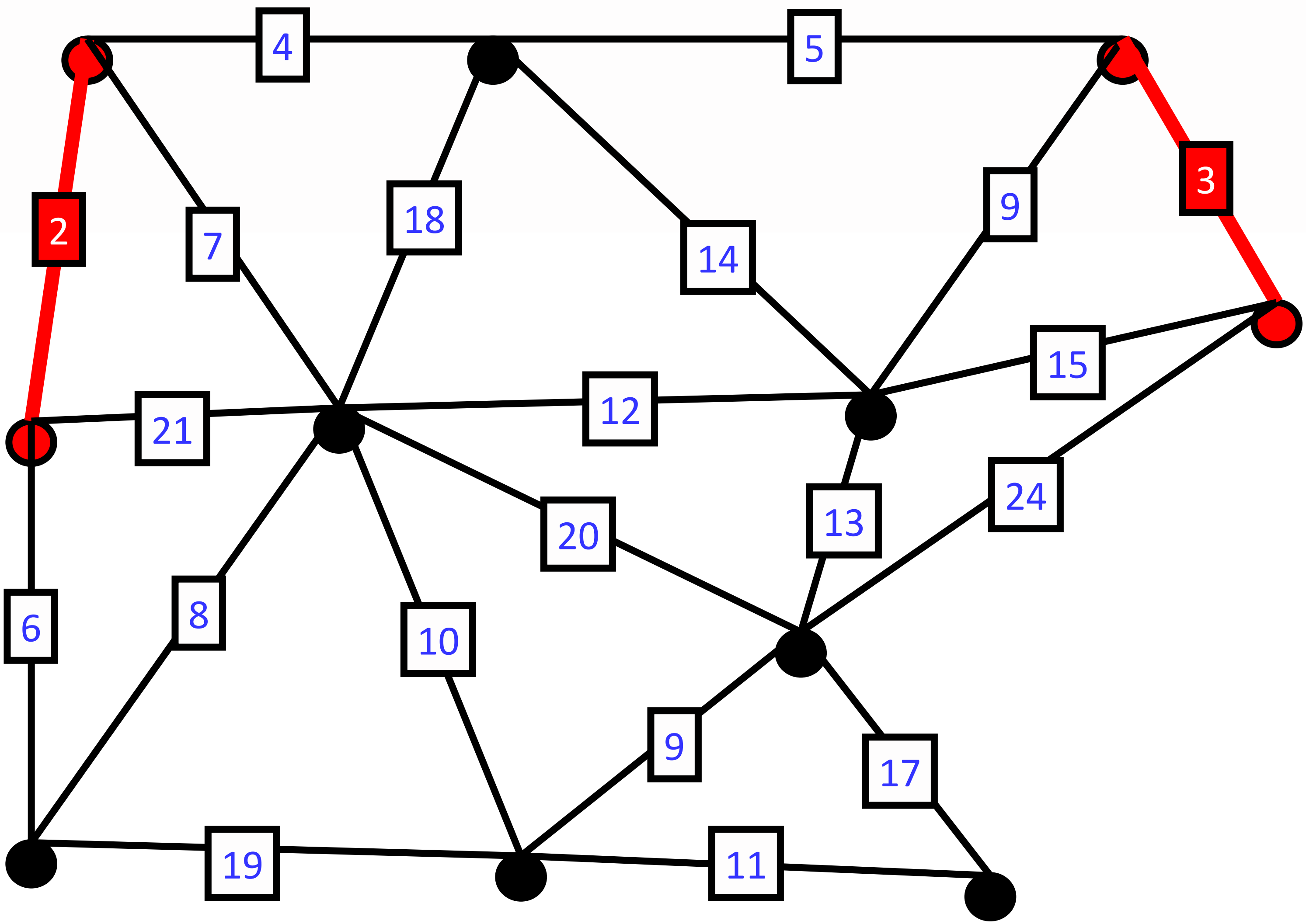



**Kruskal**($G$): // $G$ is a weighted, undirected graph
$T \leftarrow \emptyset$ // invariant: $T$ has no cycles
**for** each edge $e$ in *increasing order of weight*:
  **if** $T + e$ is acyclic: $T \leftarrow T + e$
**return** $T$

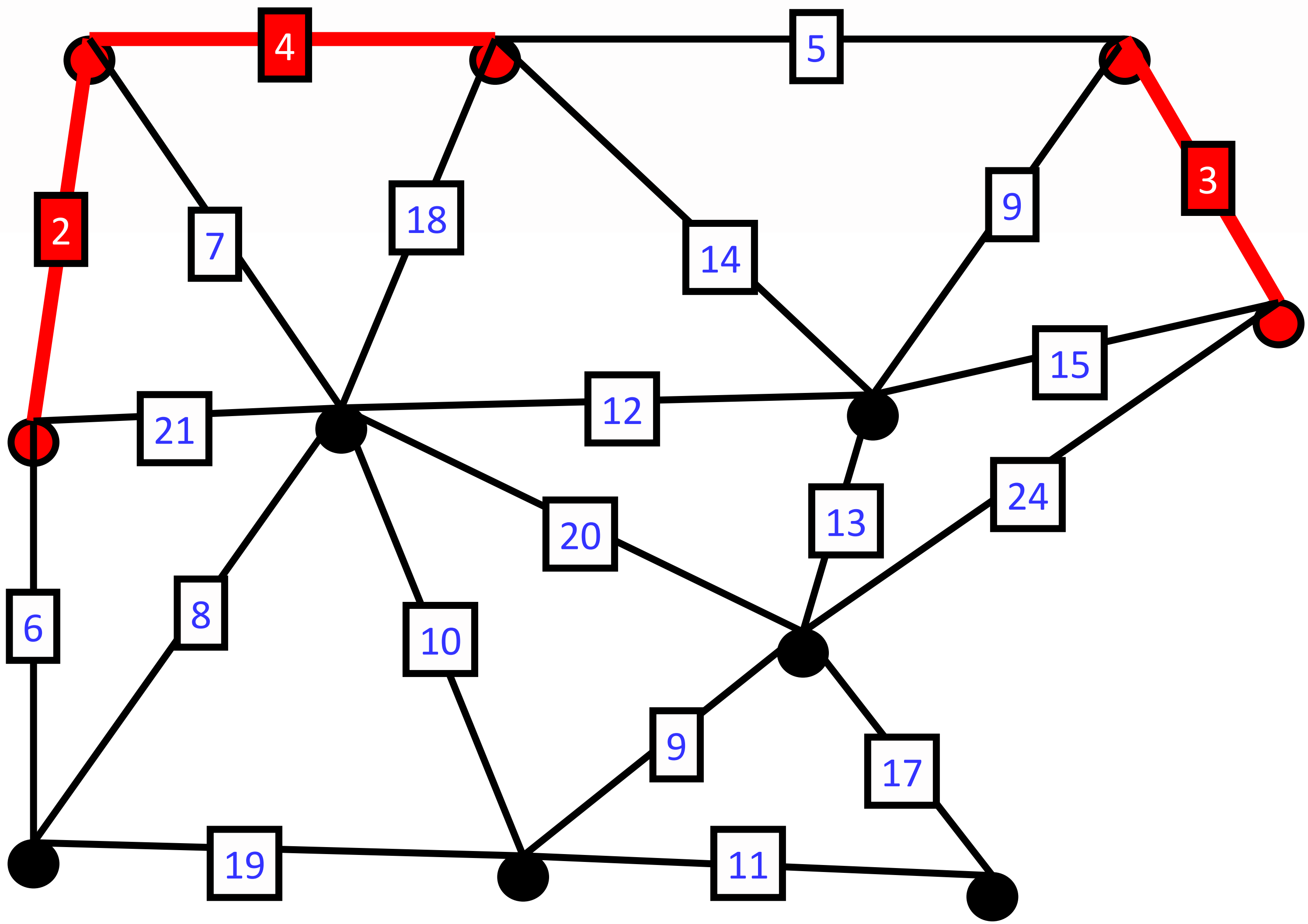
Sorted weights: **2**,3,4,5,6,7,8,9,9,10,11,12,13,14,15,17,18,19,20,21,24



Sorted weights: 2,**3**,4,5,6,7,8,9,9,10,11,12,13,14,15,17,18,19,20,21,24

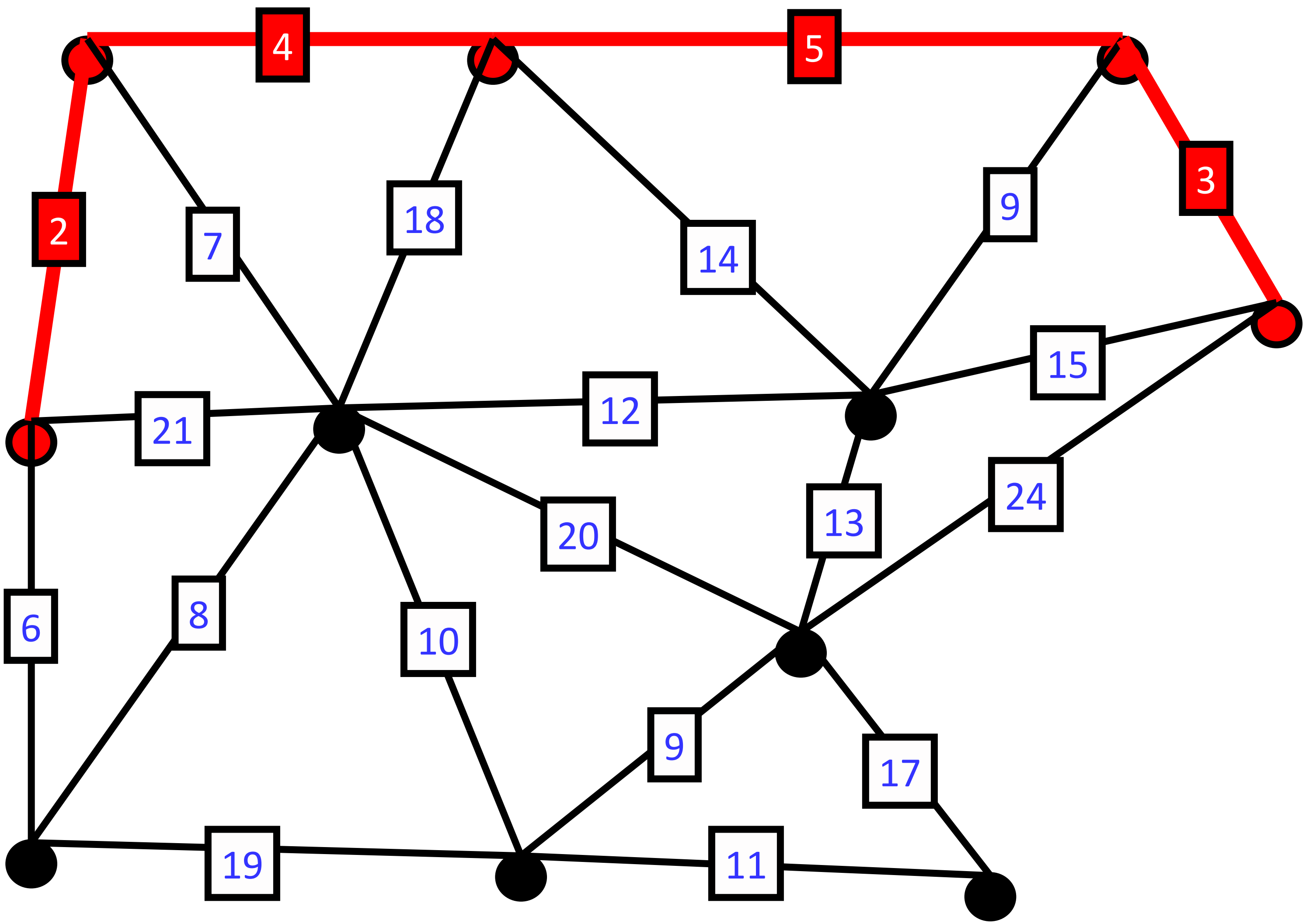


Sorted weights: 2,3,**4**,5,6,7,8,9,9,10,11,12,13,14,15,17,18,19,20,21,24

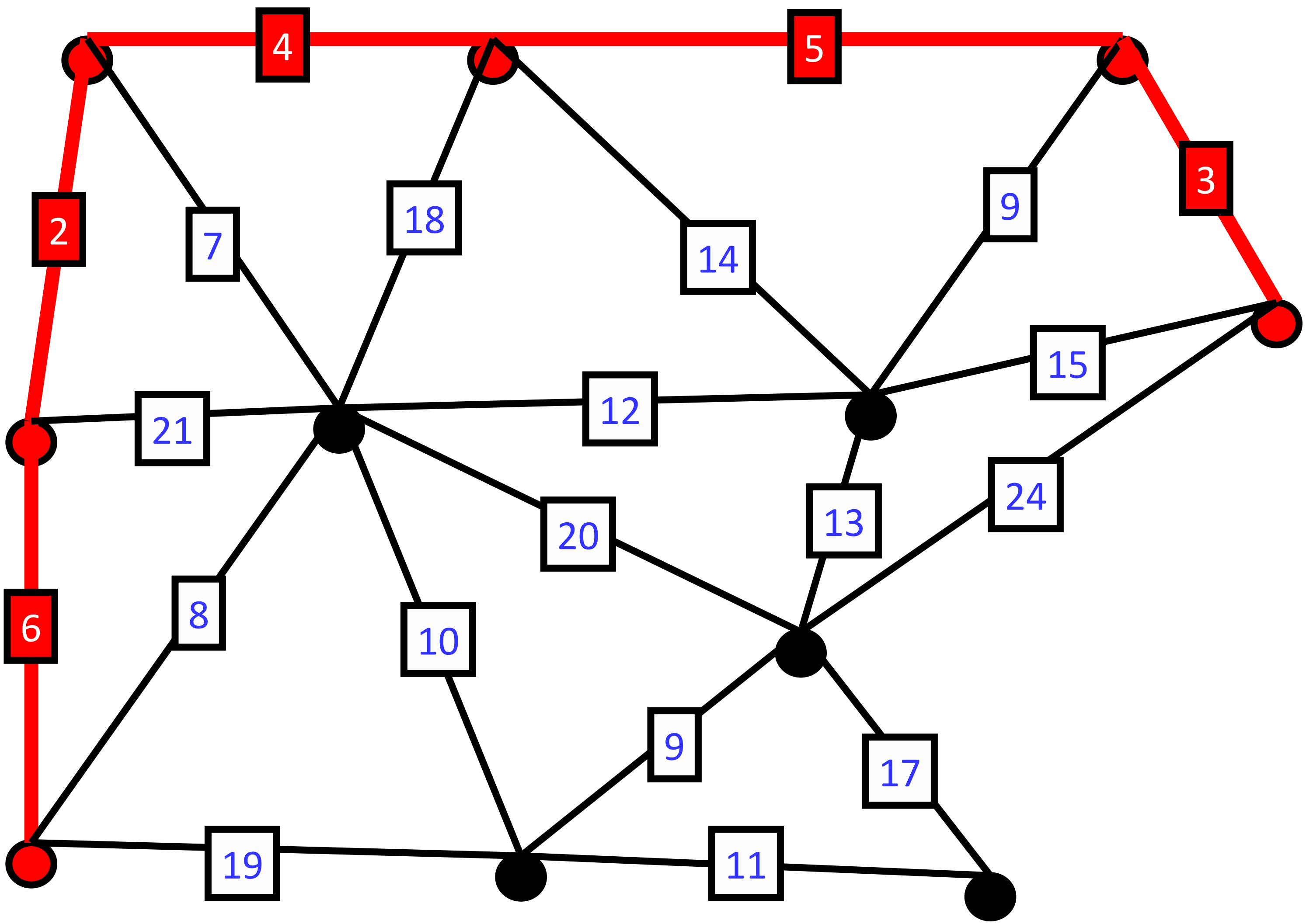


# Kruskal's Algorithm

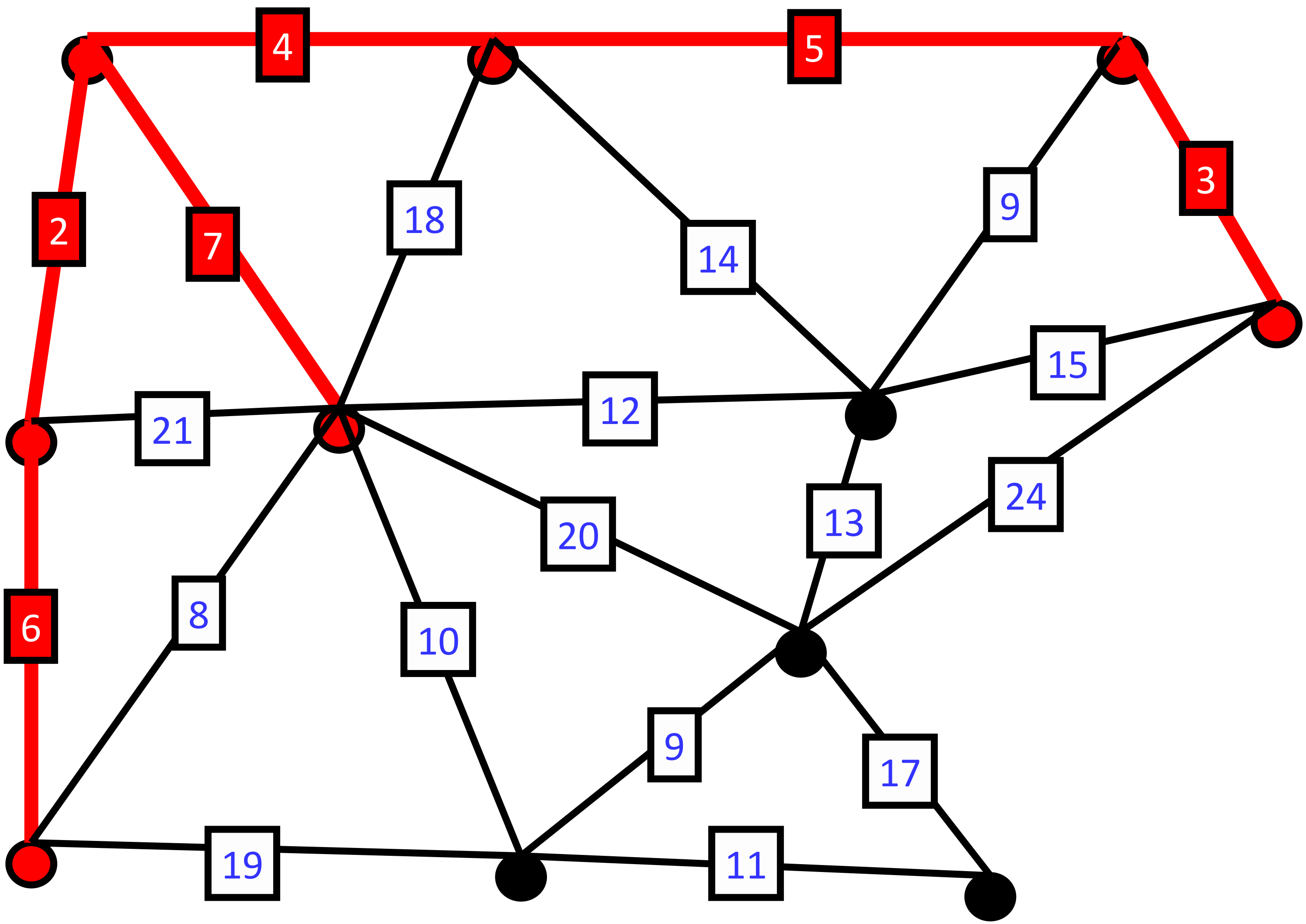
Sorted weights: 2,3,4,**5**,6,7,8,9,9,10,11,12,13,14,15,17,18,19,20,21,24



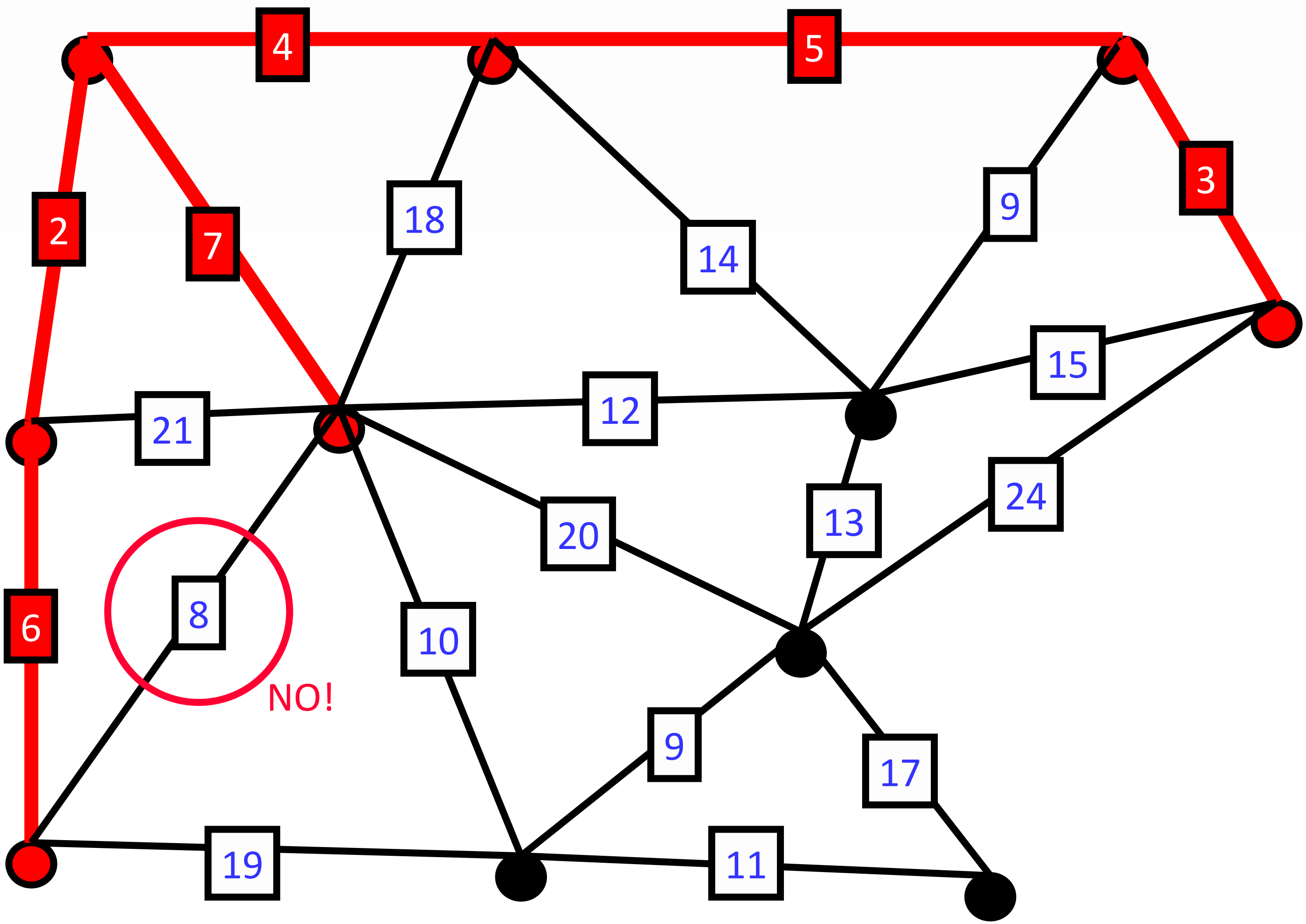
Sorted weights: 2,3,4,5,**6**,7,8,9,9,10,11,12,13,14,15,17,18,19,20,21,24



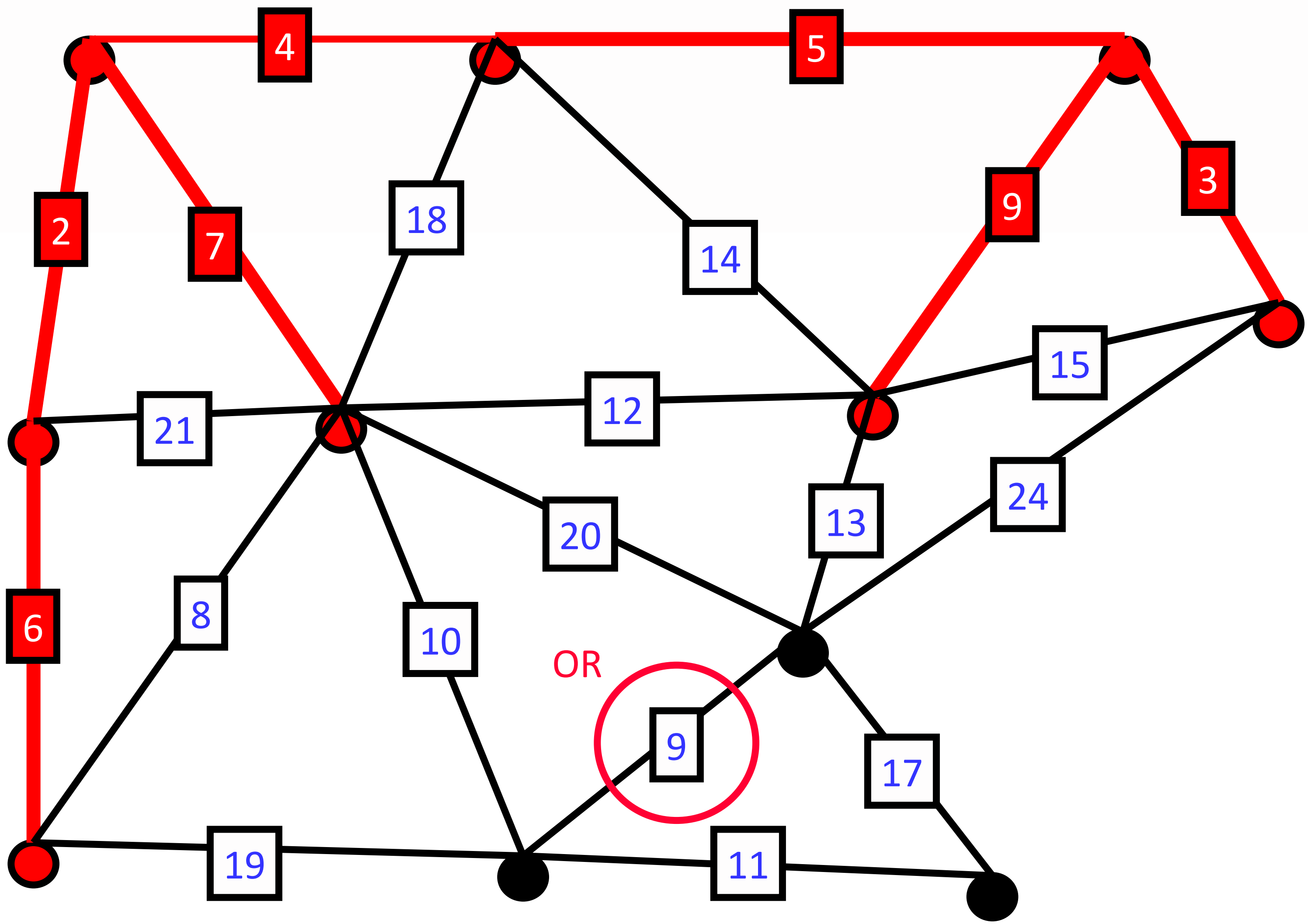
Sorted weights: 2,3,4,5,6,7,8,9,9,10,11,12,13,14,15,17,18,19,20,21,24



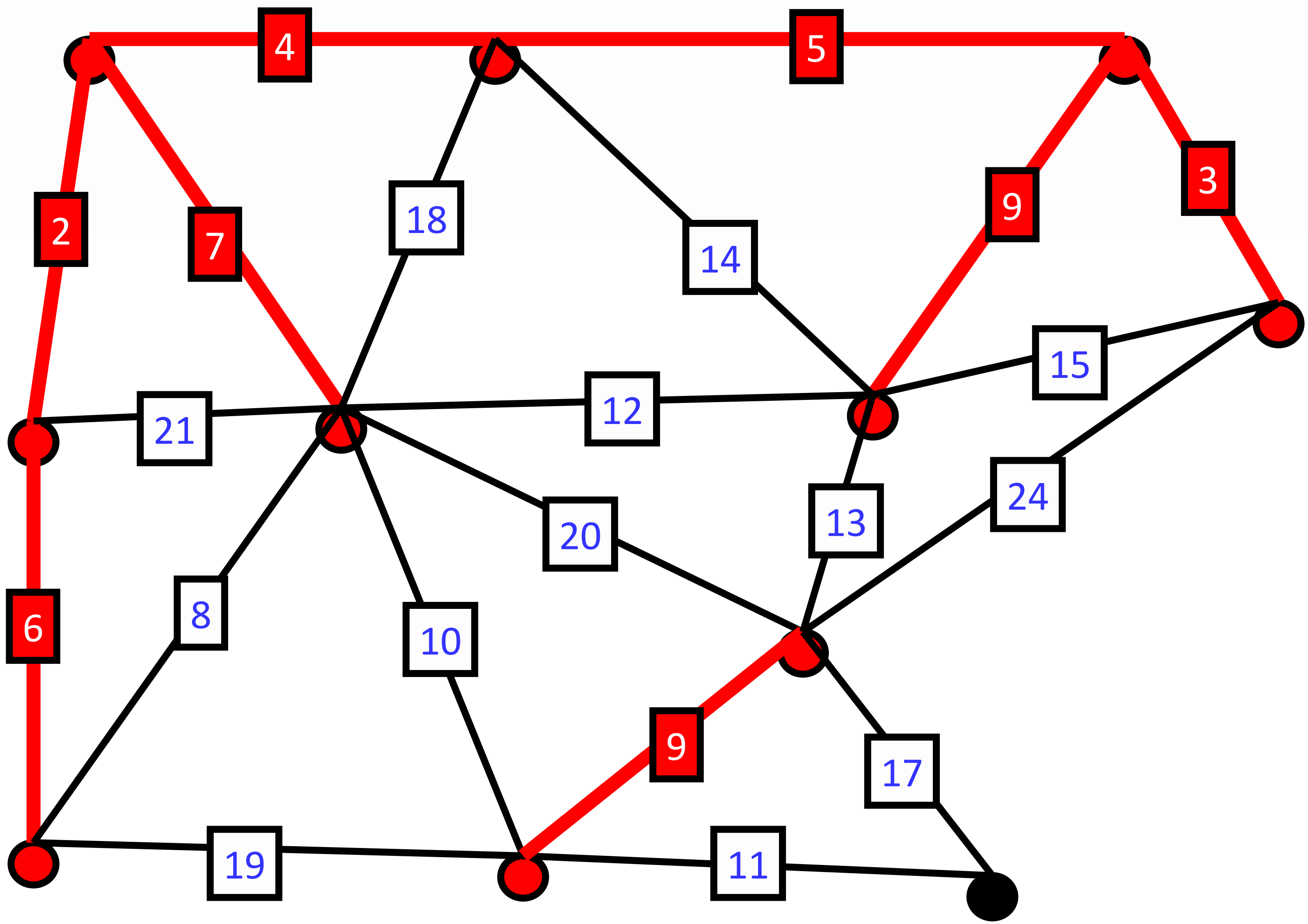
Sorted weights: 2,3,4,5,6,7,**8**,9,9,10,11,12,13,14,15,17,18,19,20,21,24



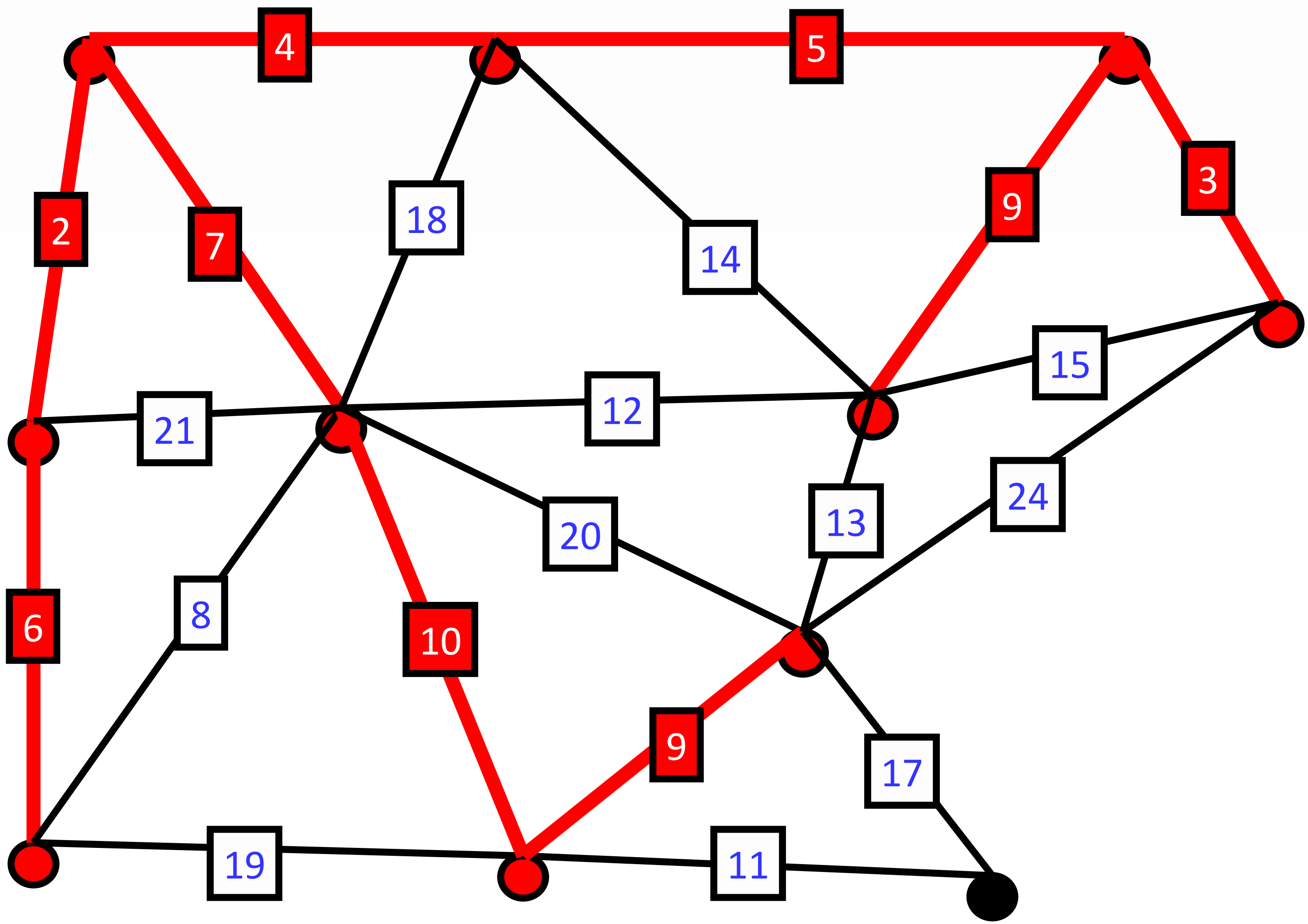
Sorted weights: 2,3,4,5,6,7,8,**9**,9,10,11,12,13,14,15,17,18,19,20,21,24



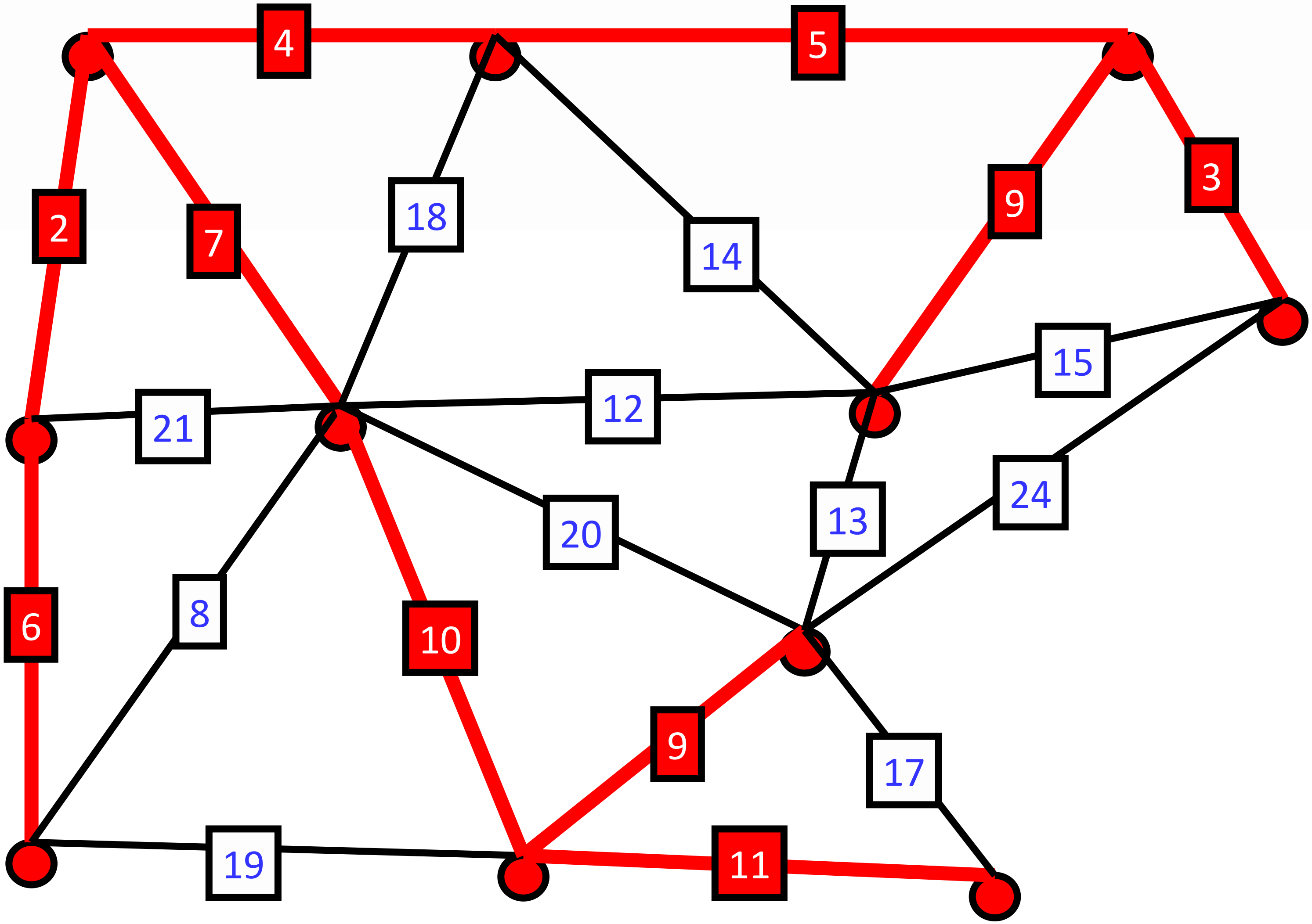
Sorted weights: 2,3,4,5,6,7,8,9,**9**,10,11,12,13,14,15,17,18,19,20,21,24



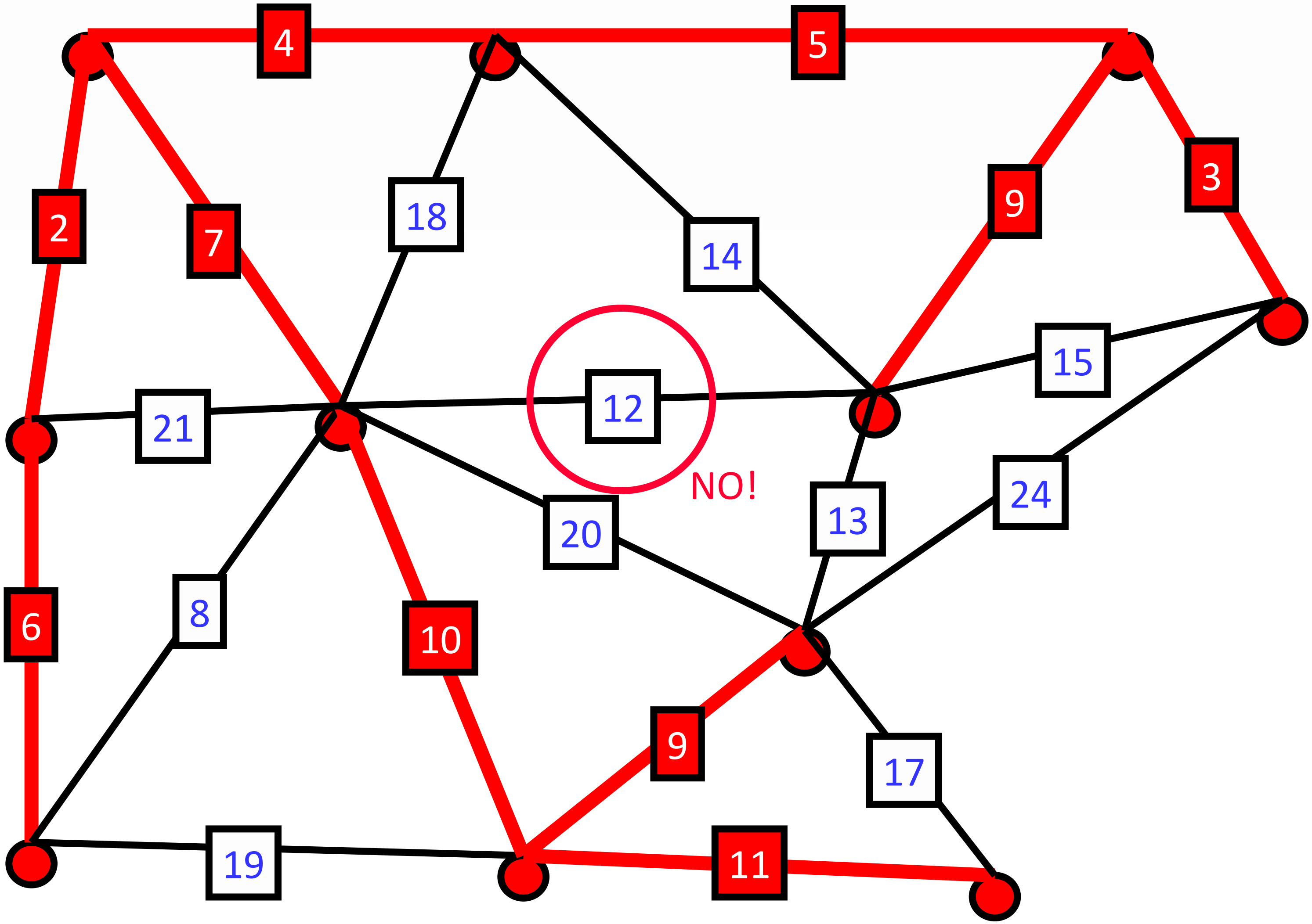
Sorted weights: 2,3,4,5,6,7,8,9,9,**10**,11,12,13,14,15,17,18,19,20,21,24



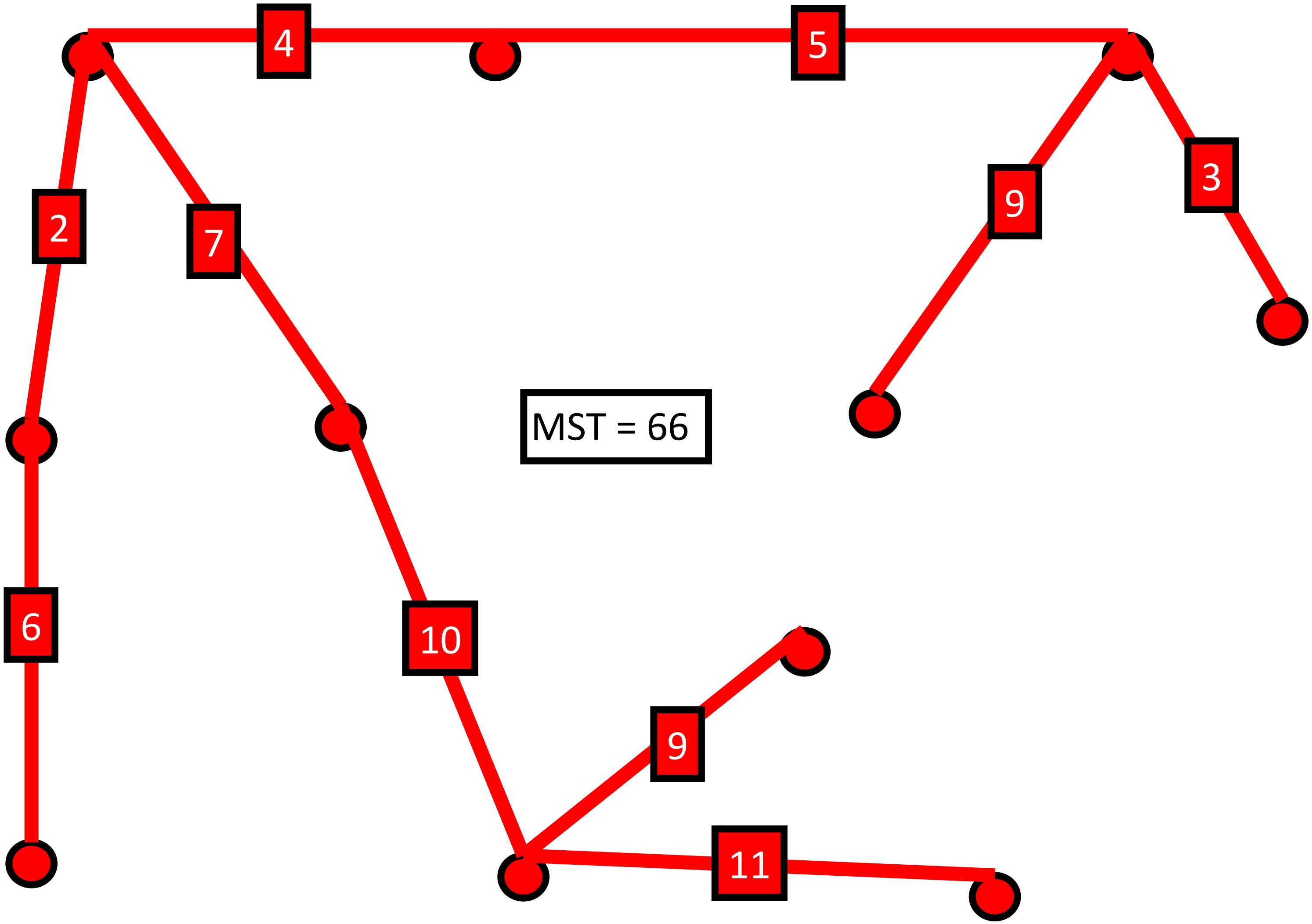
Sorted weights: 2,3,4,5,6,7,8,9,9,10,**11**,12,13,14,15,17,18,19,20,21,24



Sorted weights: 2,3,4,5,6,7,8,9,9,10,11,**12**,13,14,15,17,18,19,20,21,24



# Kruskal's Algorithm



# Kruskal's Algorithm: Correctness

Why does Kruskal's algorithm return a **spanning tree**?

1. Why is it a **tree**?
2. Why is it **spanning** (i.e. connects all vertices)?

Now we need to show it returns a **minimum** spanning tree.

Again, we'll use induction with an **exchange** argument.

# Setting up the Induction

edges in order of addition (so, **non-decreasing** weights)

Let **T = e1, e2, e3, ...** be the output of Kruskal's algorithm

**Goal:** Prove that for all **k**, the edge set **e1, e2, e3, … , ek** is in <u>some</u> MST.

*Proof by induction on **k**:*

**Base case:** k=0.

**Inductive hypothesis:** Suppose the edge set **e1, e2, e3, … , ek** is in <u>some</u> MST **T' = e1, e2, e3, … , ek, f1, f2, …**

**f** edges listed in no particular order

**Inductive step:** Goal: Show the edge set **e1, e2, e3, … , ek+1** is in <u>some</u> MST.

# Inductive step

**Inductive step:** Goal: Show the edge set **$e_1, e_2, e_3, \ldots, e_{k+1}$** is in <u>some</u> MST.

By the inductive hypothesis, there <u>exists</u> an MST:

$$T' = e_1, e_2, e_3, \ldots, e_k, f_1, f_2, \ldots$$

**Case 1: $e_{k+1} \in T'$**

We are done.

**Case 2: $e_{k+1} \notin T'$**

**Claim:** We can perform an edge **exchange**: We can take **$T'$**, add **$e_{k+1}$**, and remove an edge in **$f_1, f_2, \ldots$** to get a spanning tree that still has minimum weight.

**Proof of Claim:**

Which edge $f_i$ do we exchange with $e_{k+1}$?

After the exchange, why is the graph still connected?

After the exchange, why is the graph still a tree?

**Proof of Claim (continued):**

Why did the exchange not increase the weight of the spanning tree?

Want to show: weight($e_{k+1}$) ≤ weight($f_i$).

Other independent inventors of MST algorithms:

Jarnik, Prim, Borůvka

First used for designing electrical networks in 1926!

Today, a number of algorithms for clustering data (useful in machine learning) are variations of Kruskal's algorithm

**Running time** (we won't prove) of Kruskal's algorithm: O(m log n)
⇒ need to analyze a data structure for detecting cycles

**Best-known running time** (Chazelle 2000): $O(m \cdot \alpha(n))$

Inverse Ackermann function: an extreeemly slow growing function: $\alpha(n) \le 4$ even when n is # particles in known universe

**Open problem:** O(m)?

Seth Pettie and Vijaya Ramachandran (2002) gave an asymptotically ***optimal*** algorithm. But nobody knows how fast it is!

You've just finished a crash course on algorithms!

If you found the material interesting, try:

EECS 477:
*Introduction to Algorithms*

---

Next up: **Computability**

- What is a computer?
- What problems can a computer solve?
- What problems can a computer not solve?