

# Advanced Data Structures and Algorithms

Min Cut and Karger's Algorithm

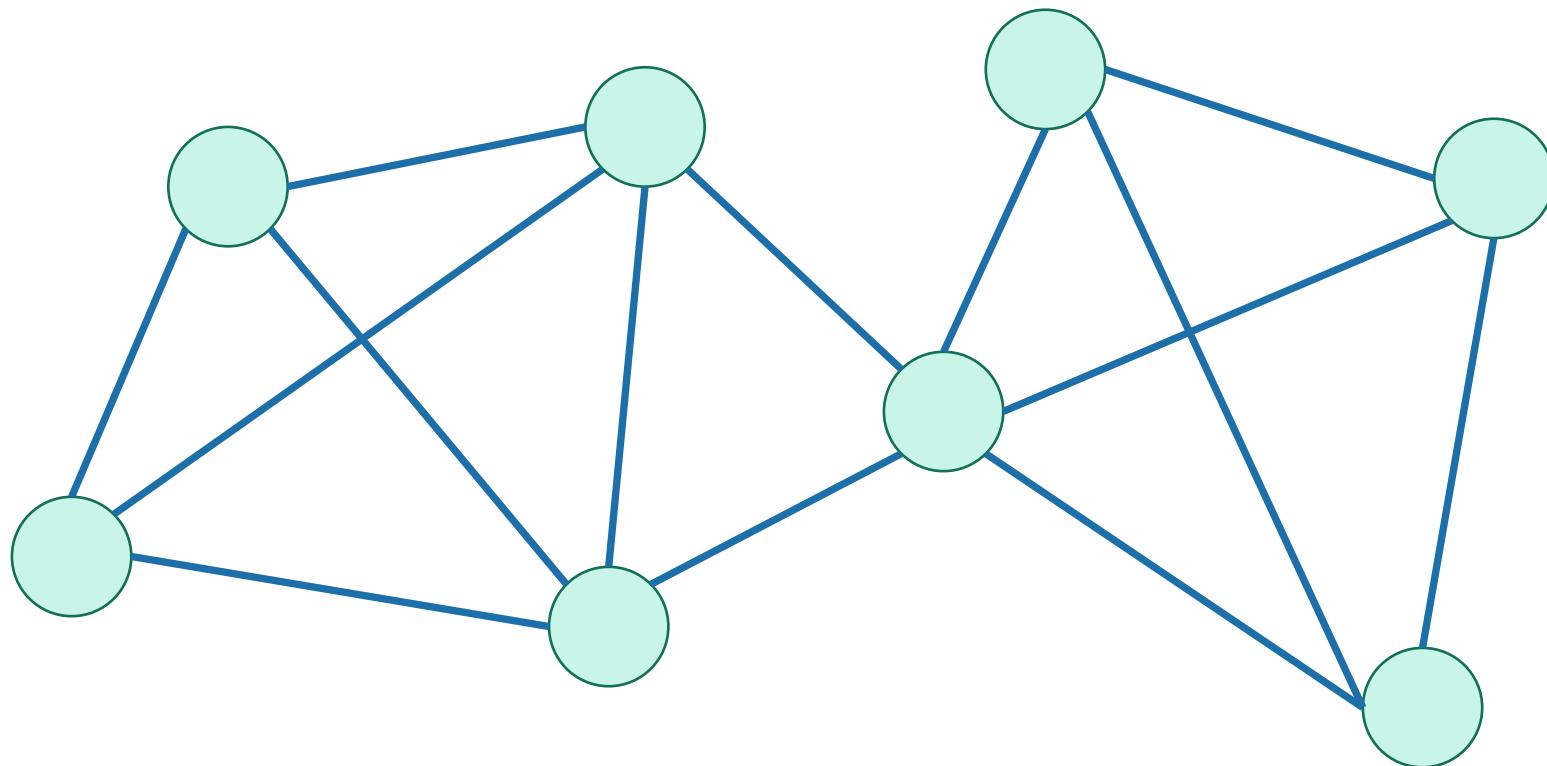
# This Module

- Minimum Cuts!
  - Karger's algorithm
  - Karger-Stein algorithm
- Back to **randomized algorithms!**

\*For today, all graphs  
are **undirected** and  
**unweighted**.

# Recall: cuts in graphs

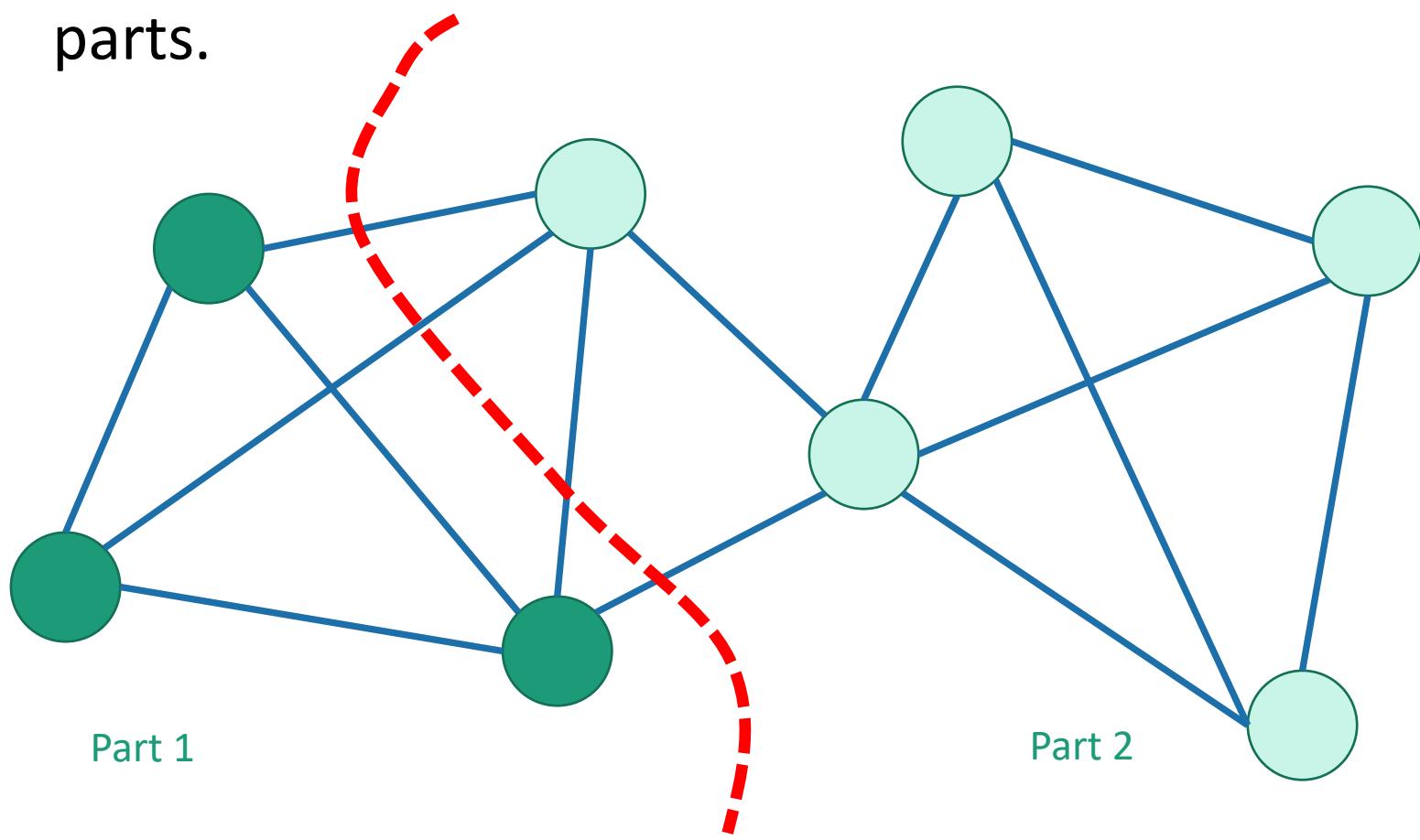
- A cut is a partition of the vertices into two **nonempty** parts.



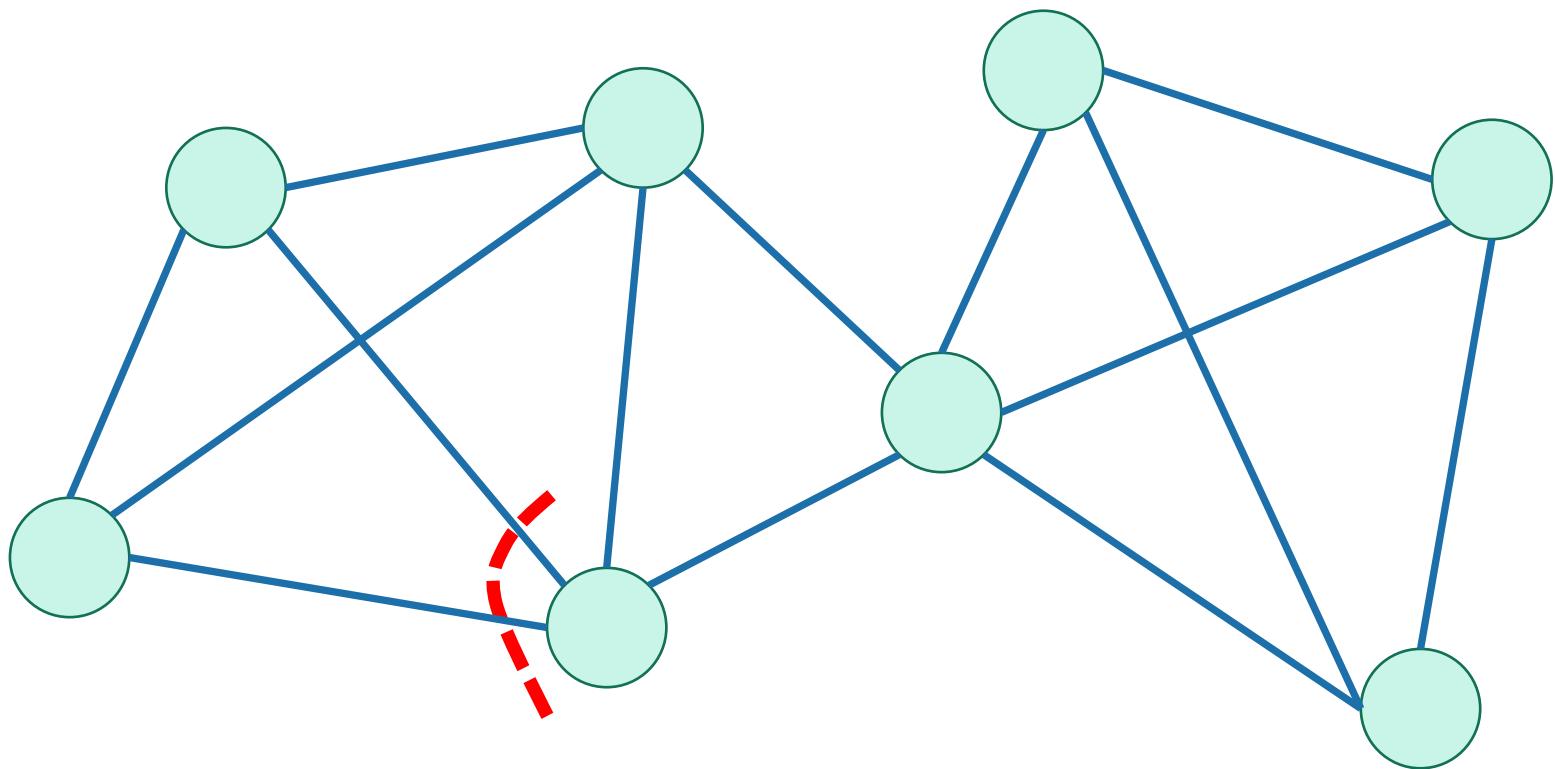
\*For today, all graphs are **undirected** and **unweighted**.

# Recall: cuts in graphs

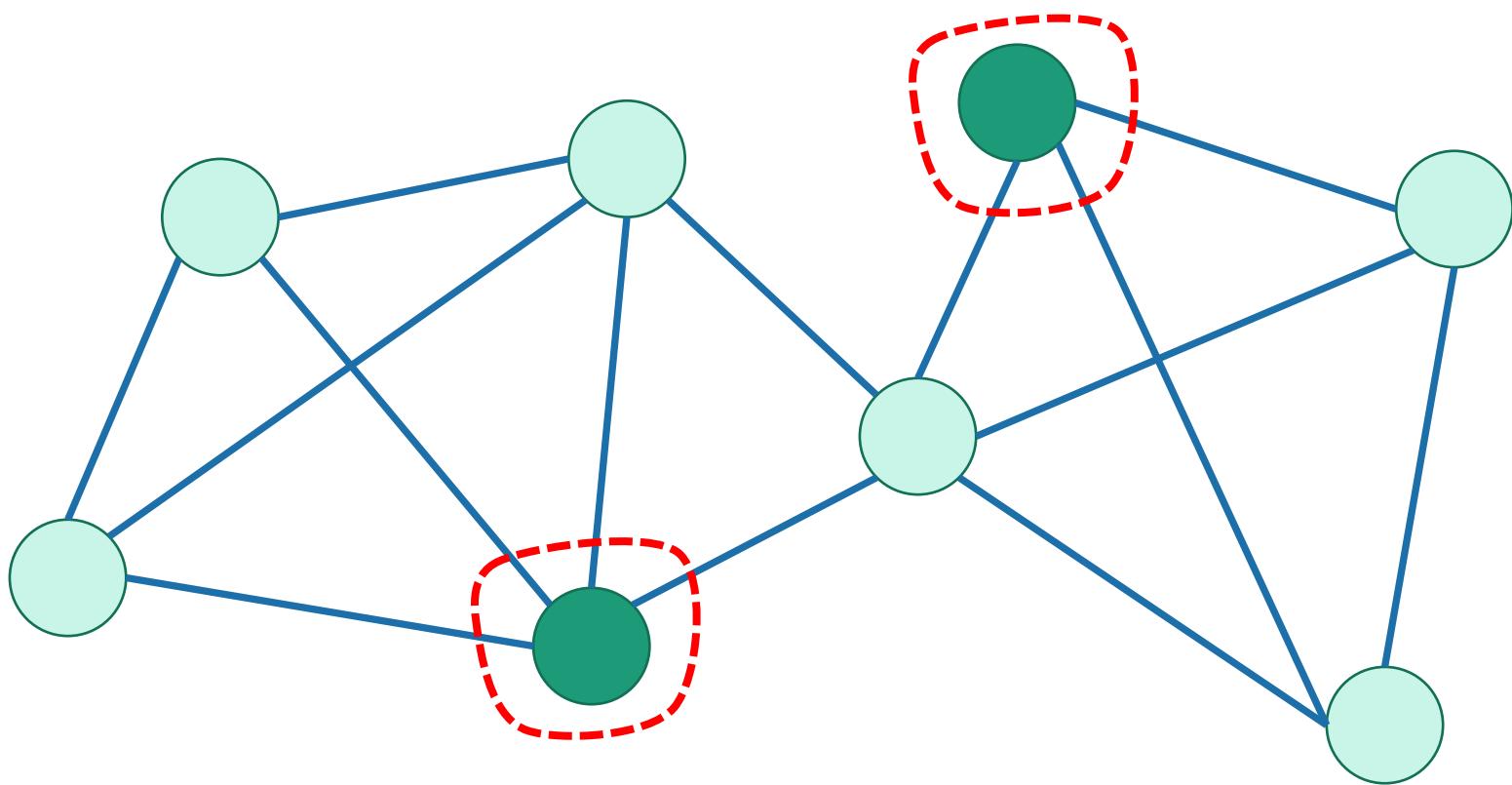
- A cut is a partition of the vertices into two **nonempty** parts.



# This is not a cut



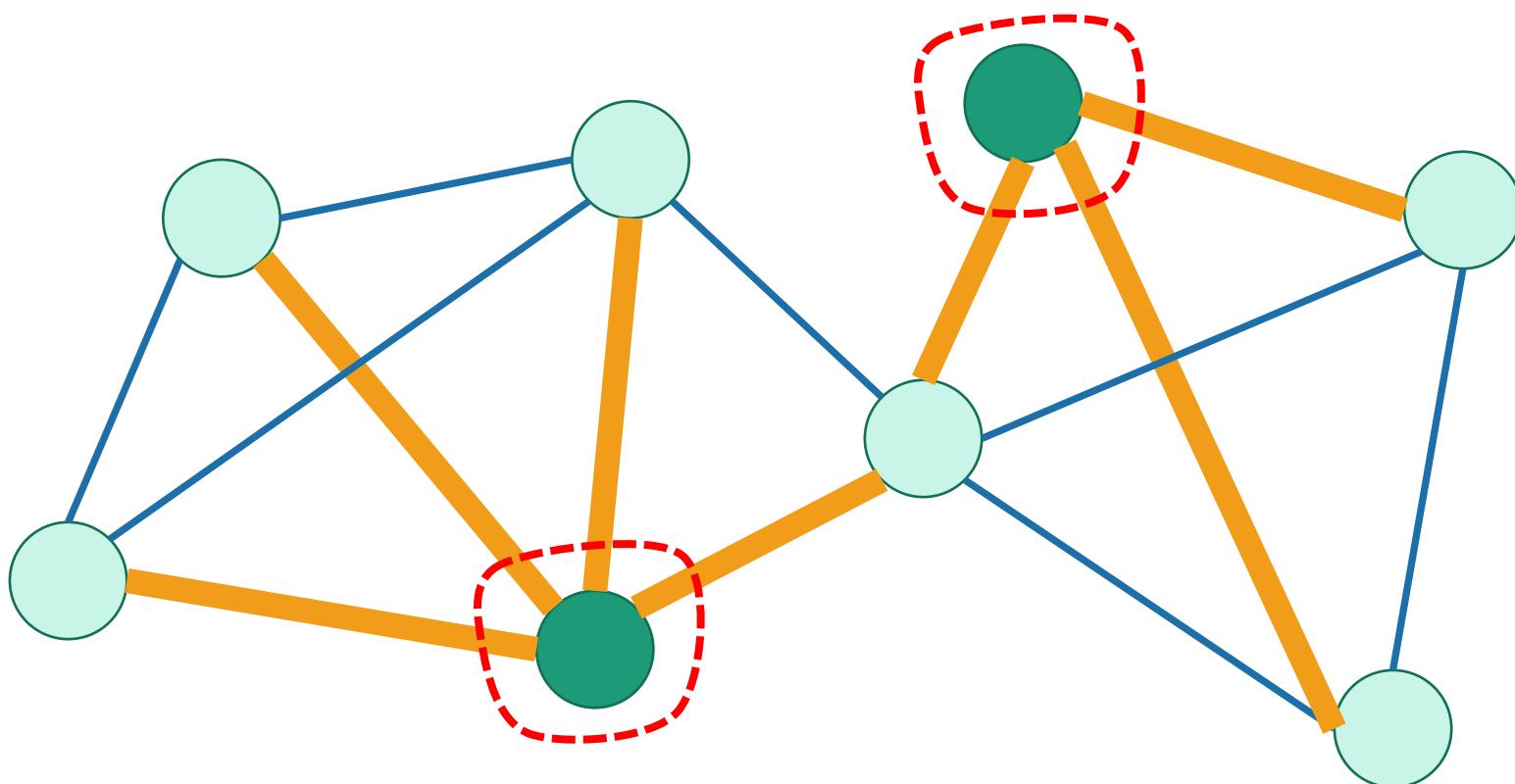
This is a cut



# This is a cut

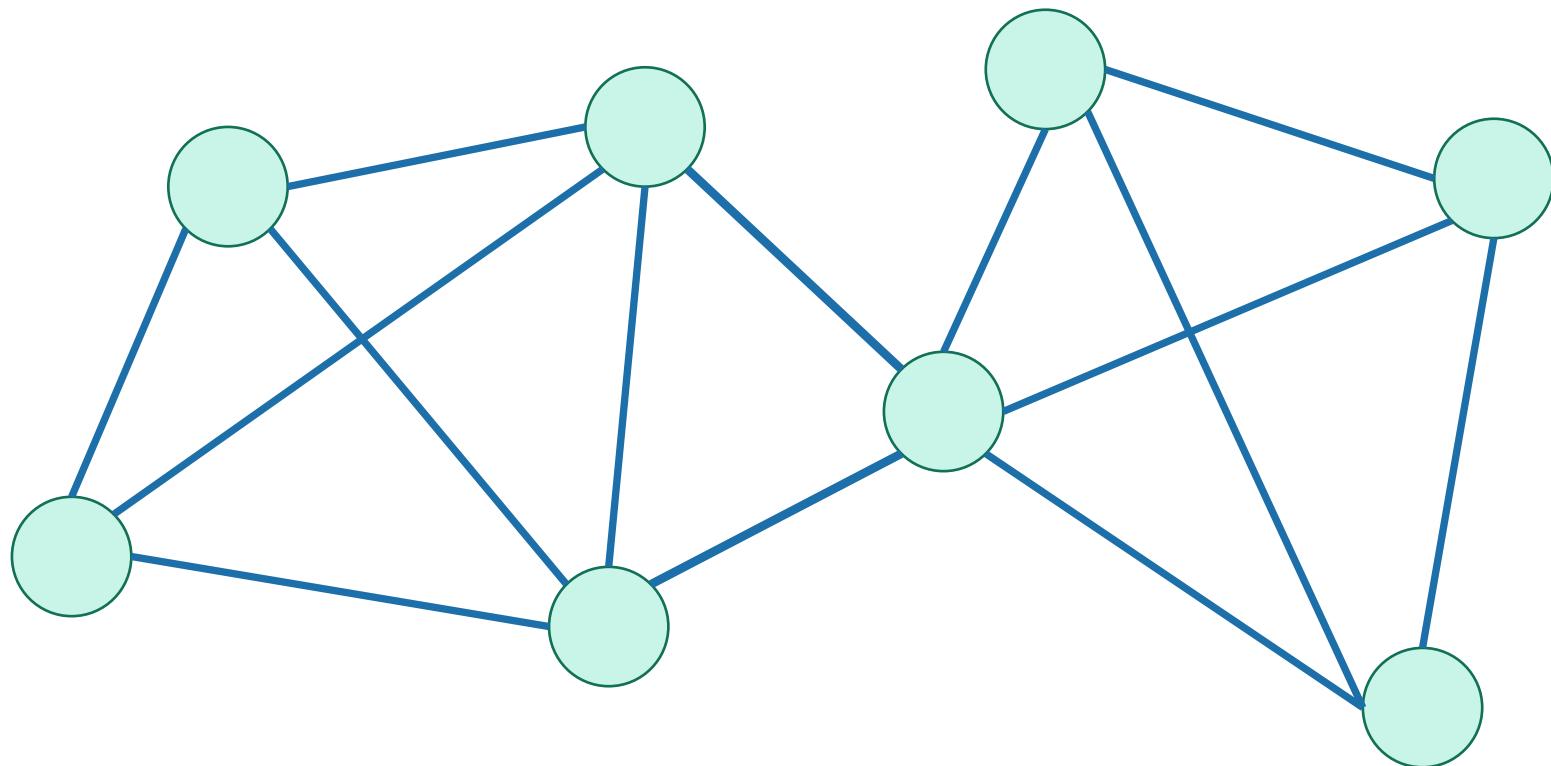
These edges **cross the cut**.

- They go from one part to the other.



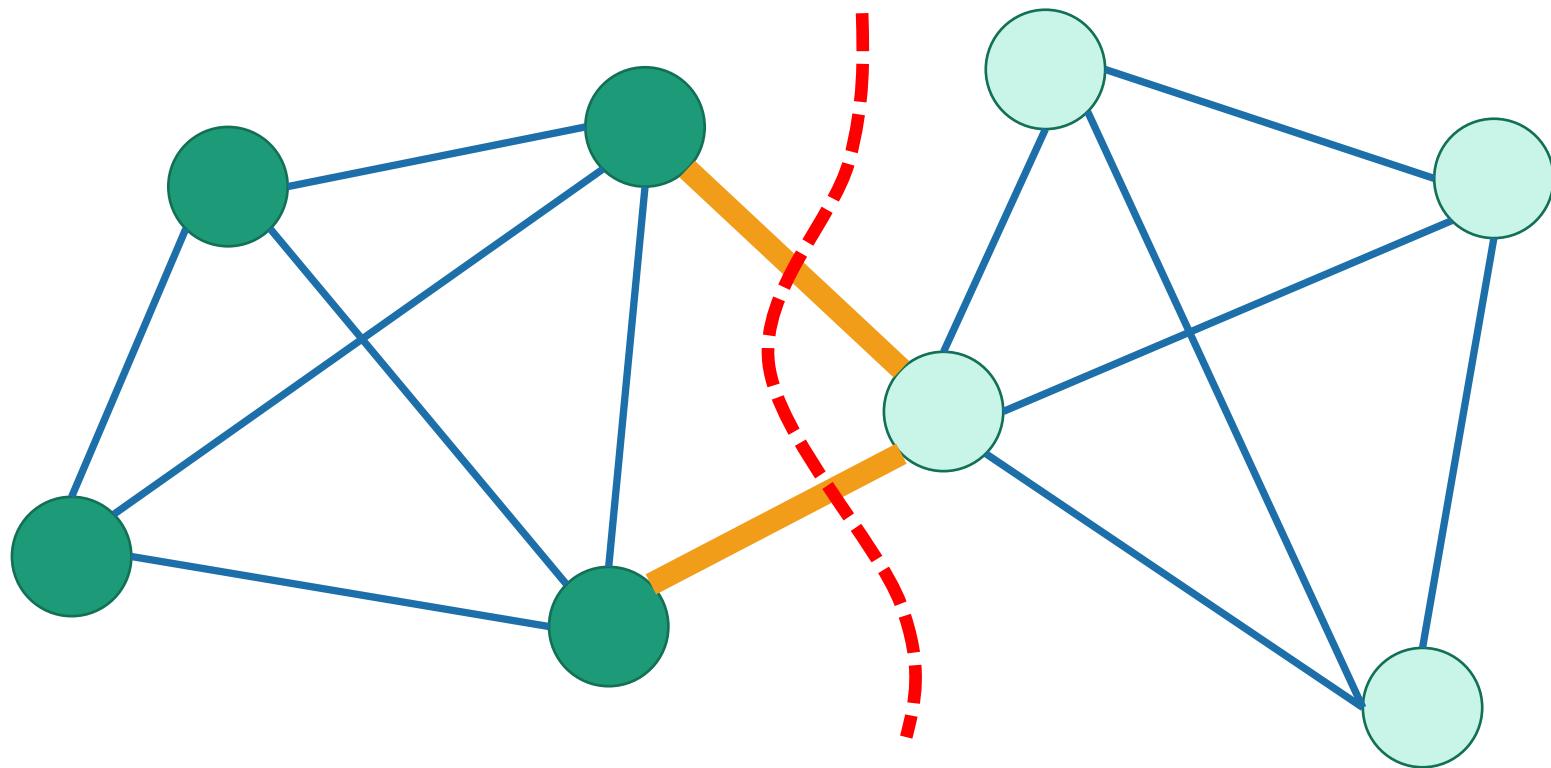
# A (global) minimum cut

is a cut that has the fewest edges possible crossing it.



# A (global) minimum cut

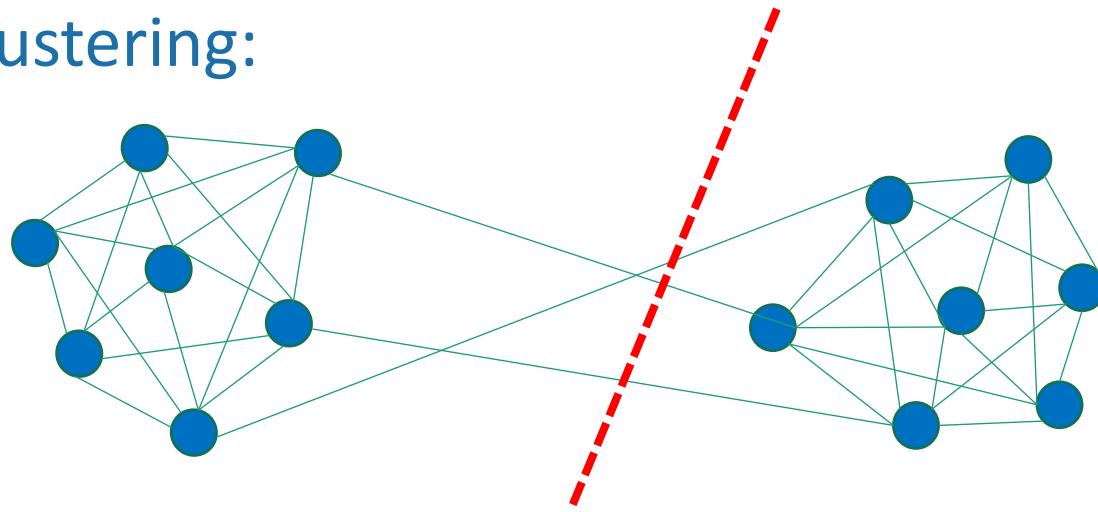
is a cut that has the fewest edges possible crossing it.



# Why might we care about global minimum cuts?

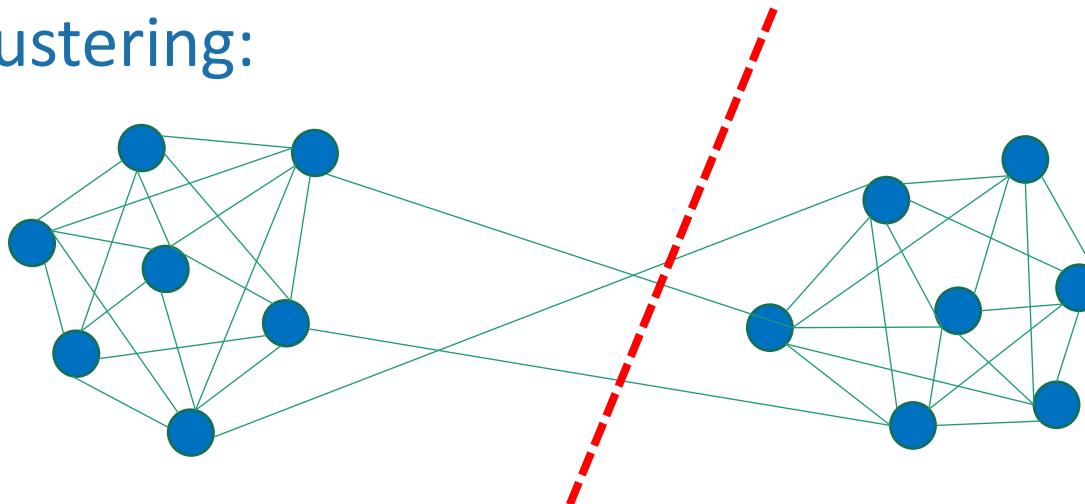
# Why might we care about global minimum cuts?

- Clustering:

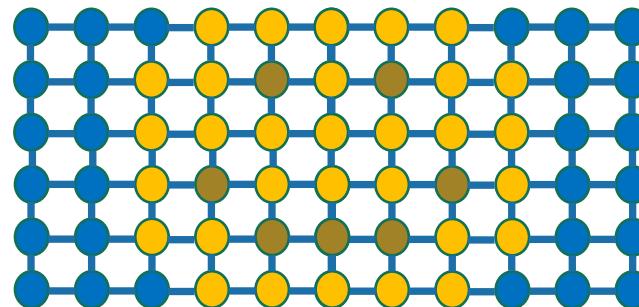


# Why might we care about global minimum cuts?

- Clustering:

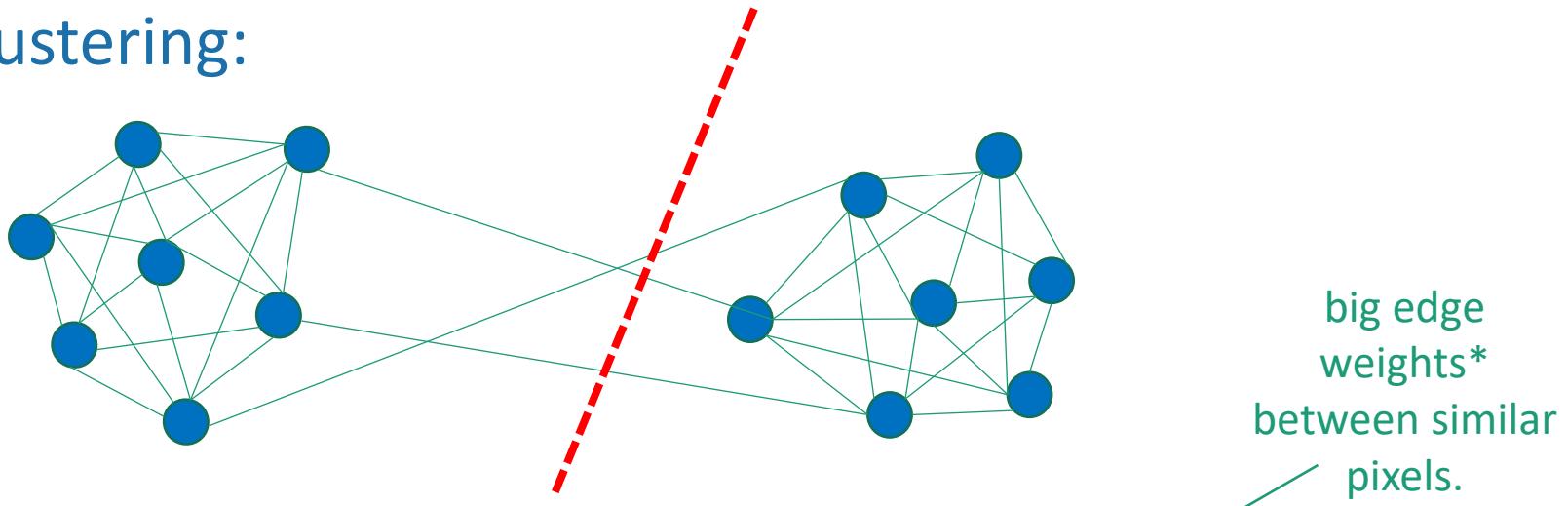


- Image Segmentation

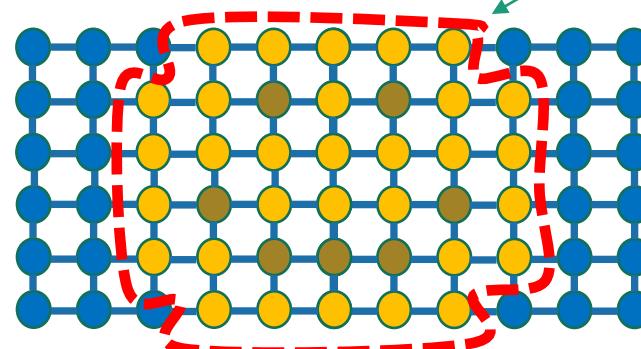


# Why might we care about global minimum cuts?

- Clustering:



- Image Segmentation



\*For the rest of today edges aren't weighted; but the algorithm can be adapted to deal with edge weights.

# Karger's algorithm

- Finds **global minimum cuts** in undirected graphs
- Randomized algorithm

# Karger's algorithm

- Finds **global minimum cuts** in undirected graphs
- Randomized algorithm
- Karger's algorithm **might be wrong.**
  - Compare to QuickSort, which just might be slow.

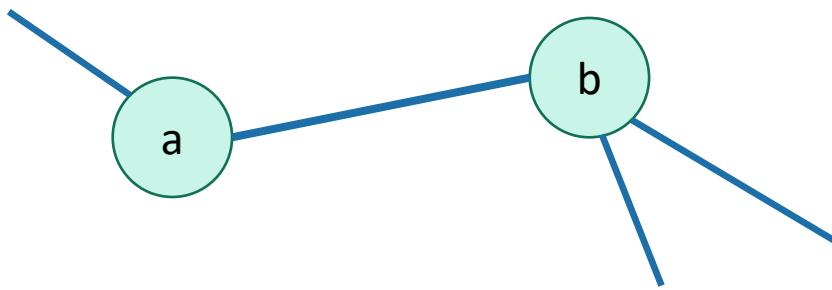
# Karger's algorithm

- Finds **global minimum cuts** in undirected graphs
- Randomized algorithm
- Karger's algorithm **might be wrong.**
  - Compare to QuickSort, which just might be slow.
- Why would we want an algorithm that might be wrong?

# Karger's algorithm

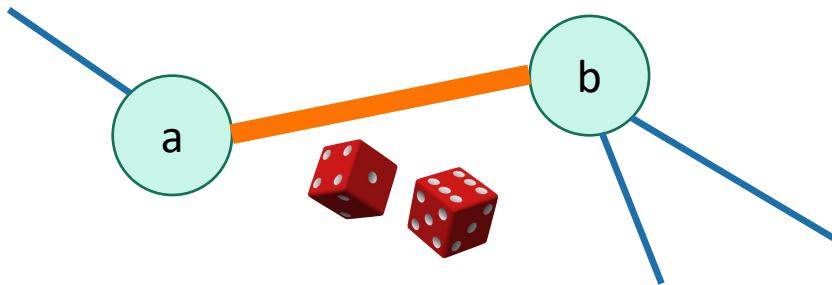
- Finds **global minimum cuts** in undirected graphs
- Randomized algorithm
- Karger's algorithm **might be wrong.**
  - Compare to QuickSort, which just might be slow.
- Why would we want an algorithm that might be wrong?
  - With high probability it won't be wrong.
  - Maybe the stakes are low and the cost of a deterministic algorithm is high.

# Karger's algorithm



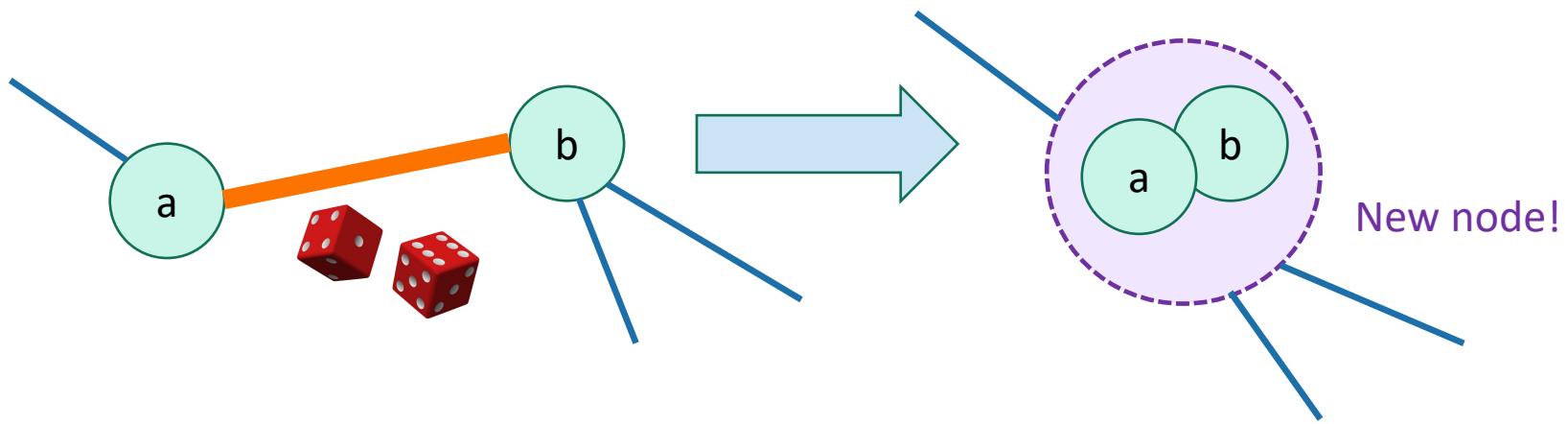
# Karger's algorithm

- Pick a random edge.



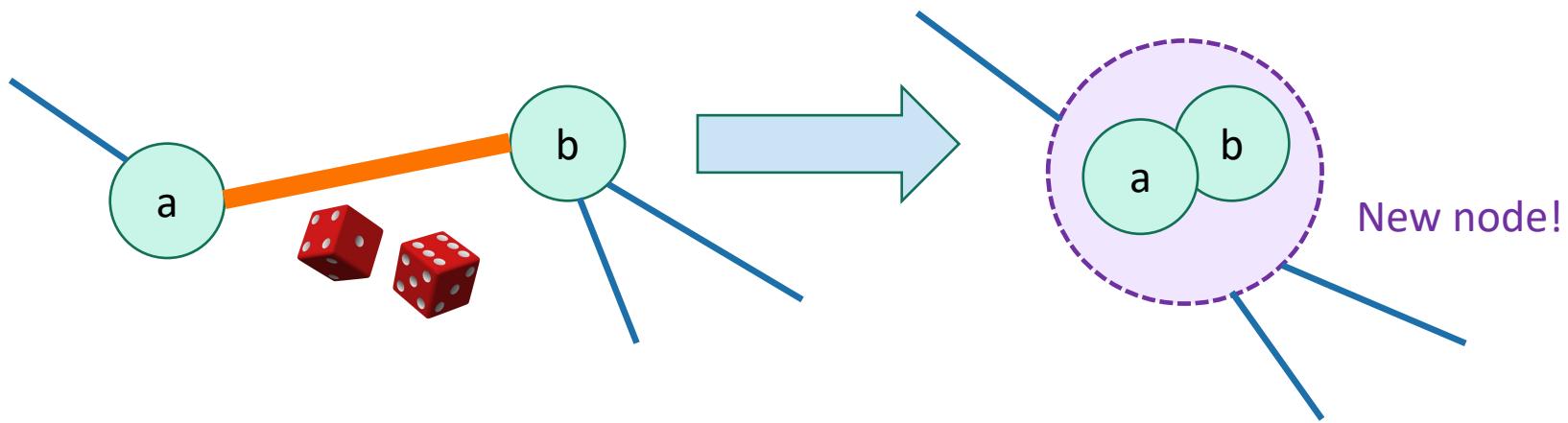
# Karger's algorithm

- Pick a random edge.
- **Contract** it.



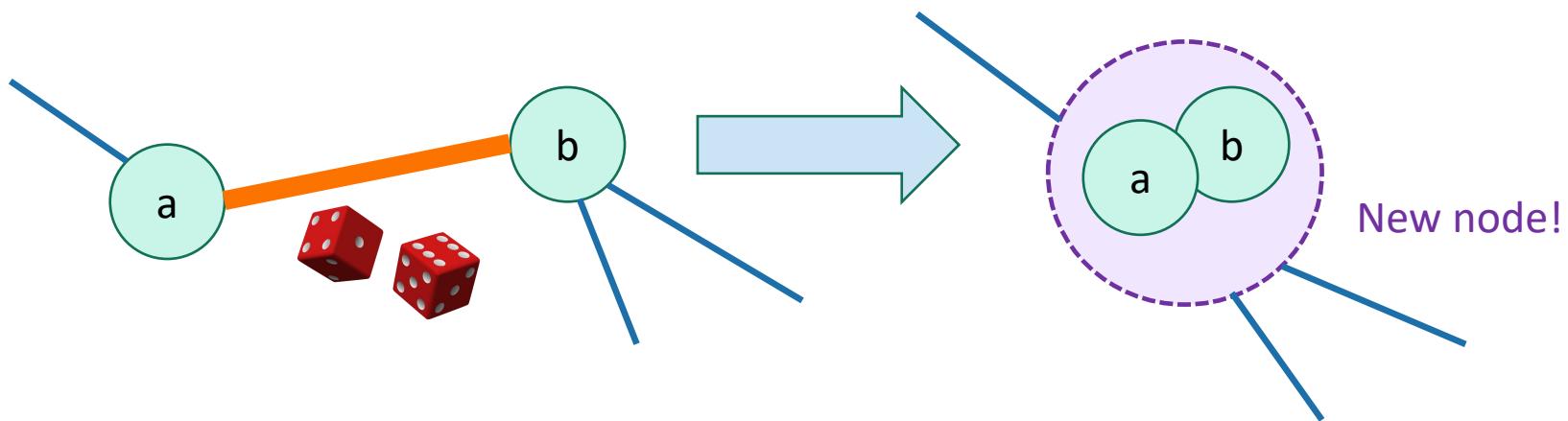
# Karger's algorithm

- Pick a random edge.
- **Contract** it.
- Repeat until you only have two vertices left.



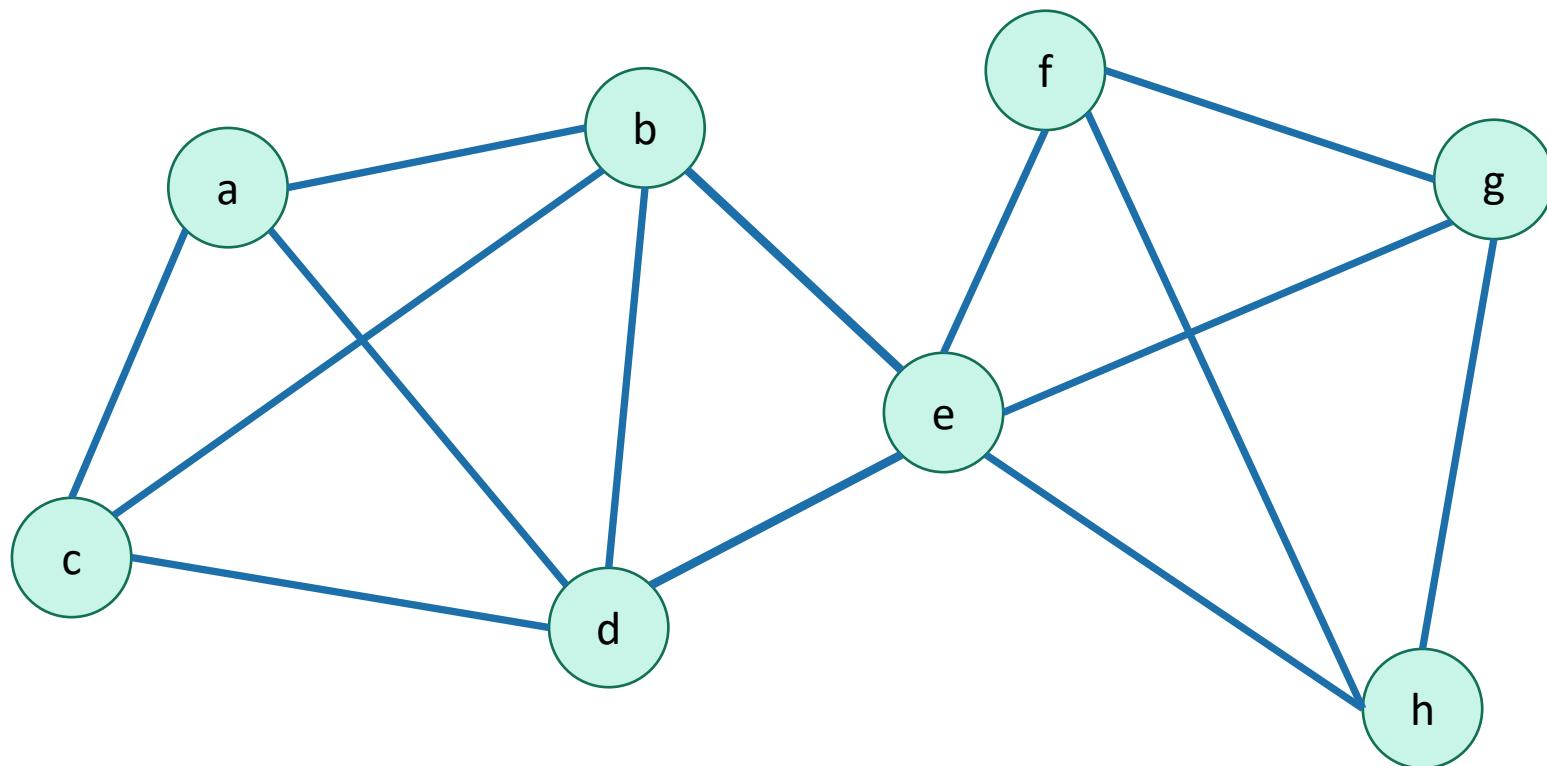
# Karger's algorithm

- Pick a random edge.
- **Contract** it.
- Repeat until you only have two vertices left.

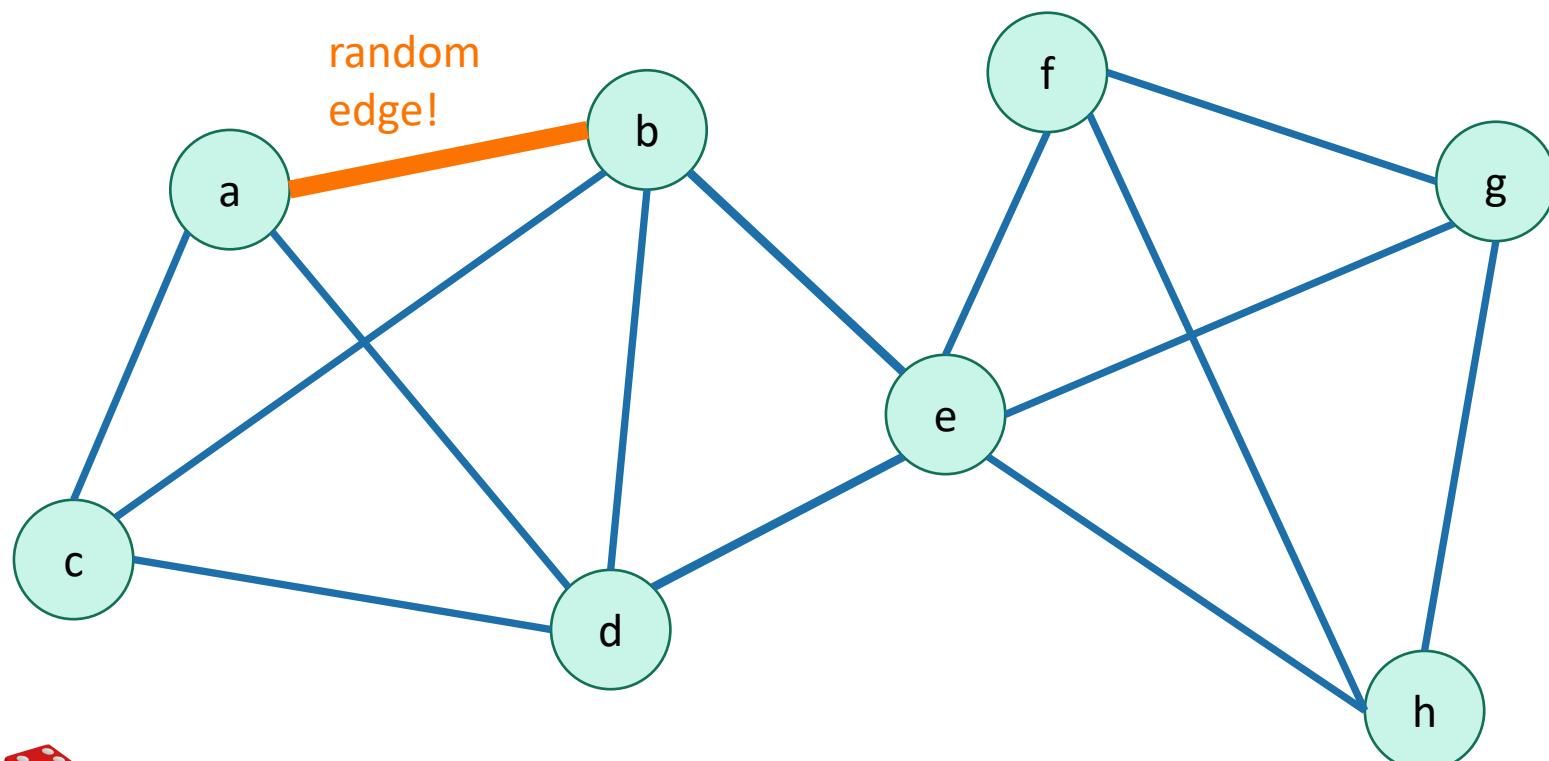


**Why is this a good idea? We'll see shortly.**

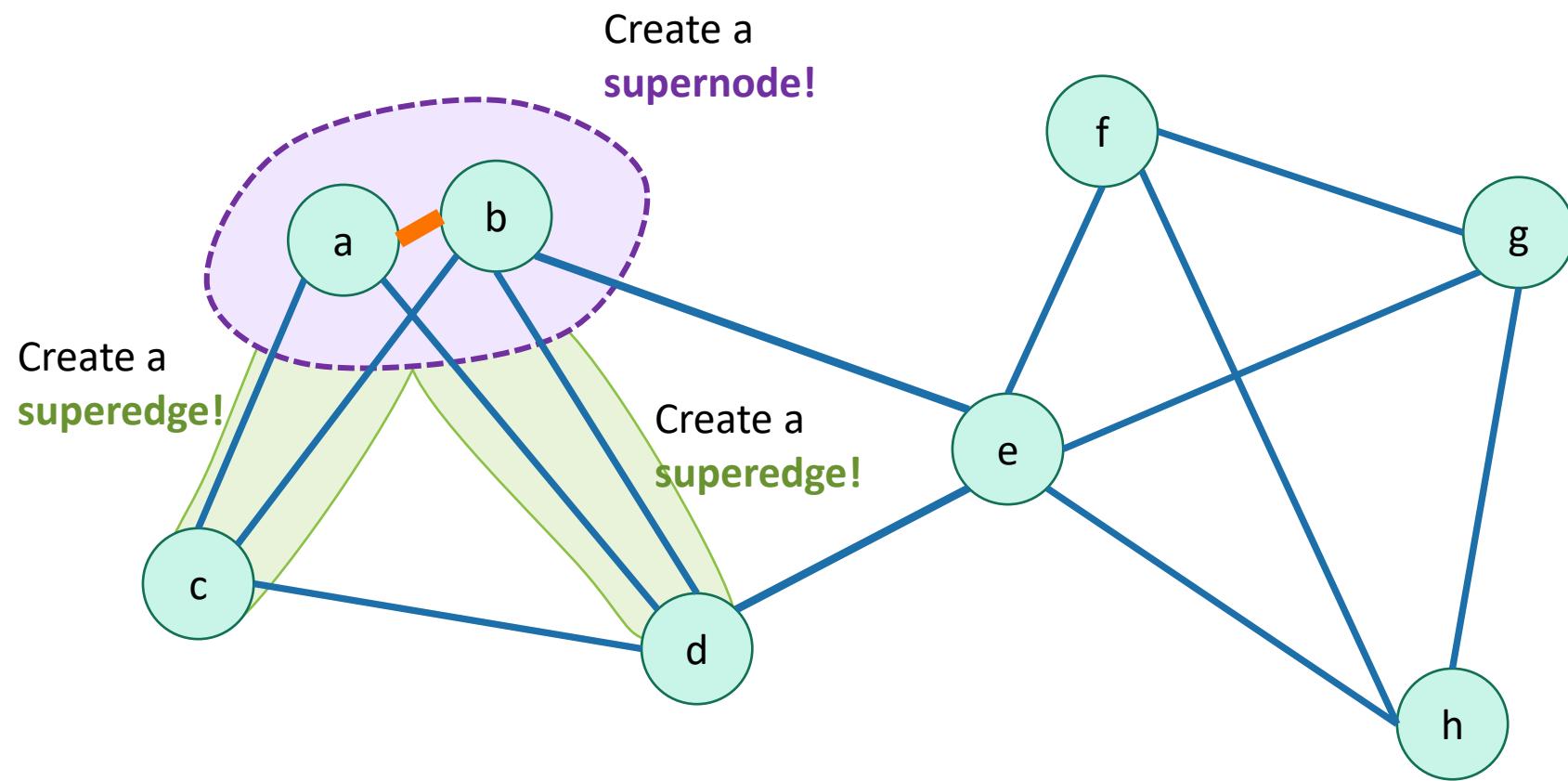
# Karger's algorithm



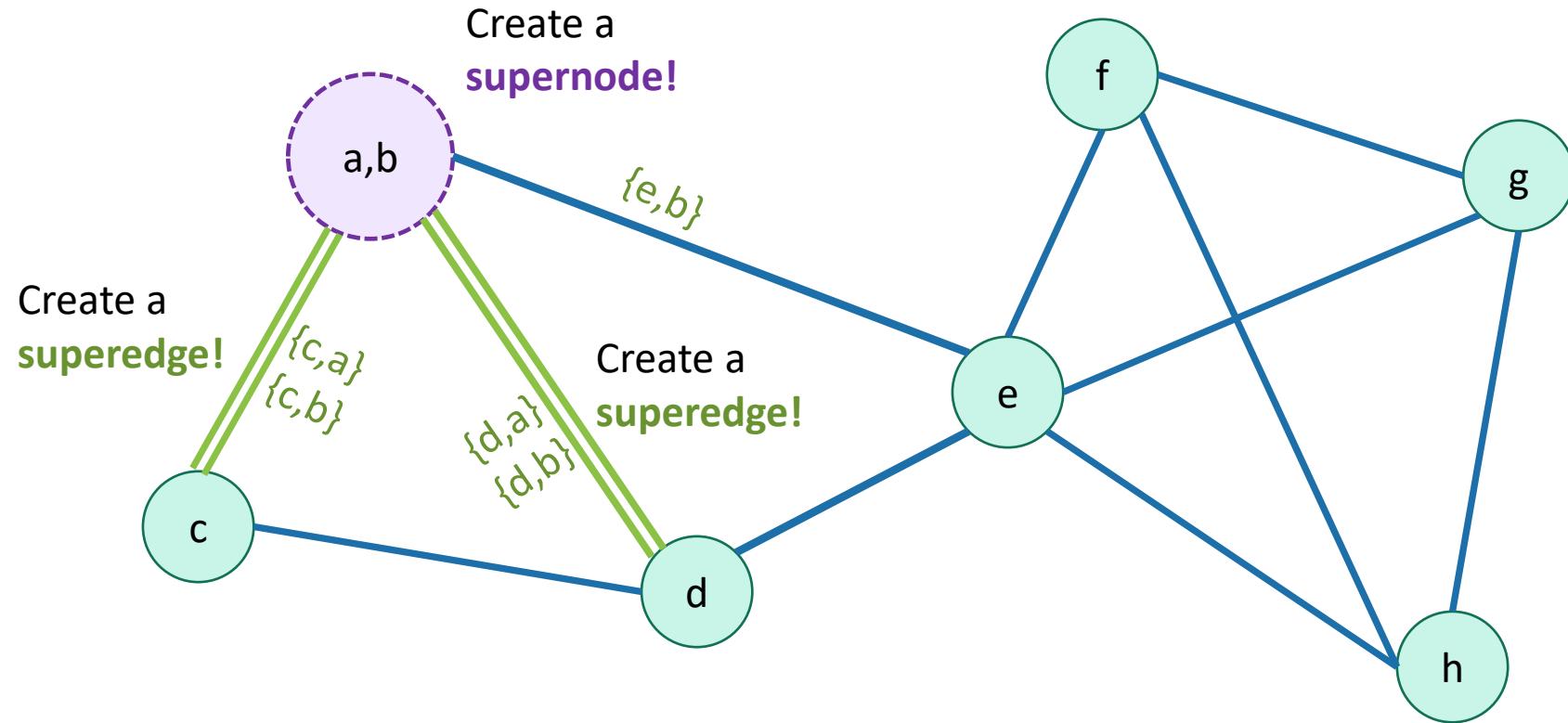
# Karger's algorithm



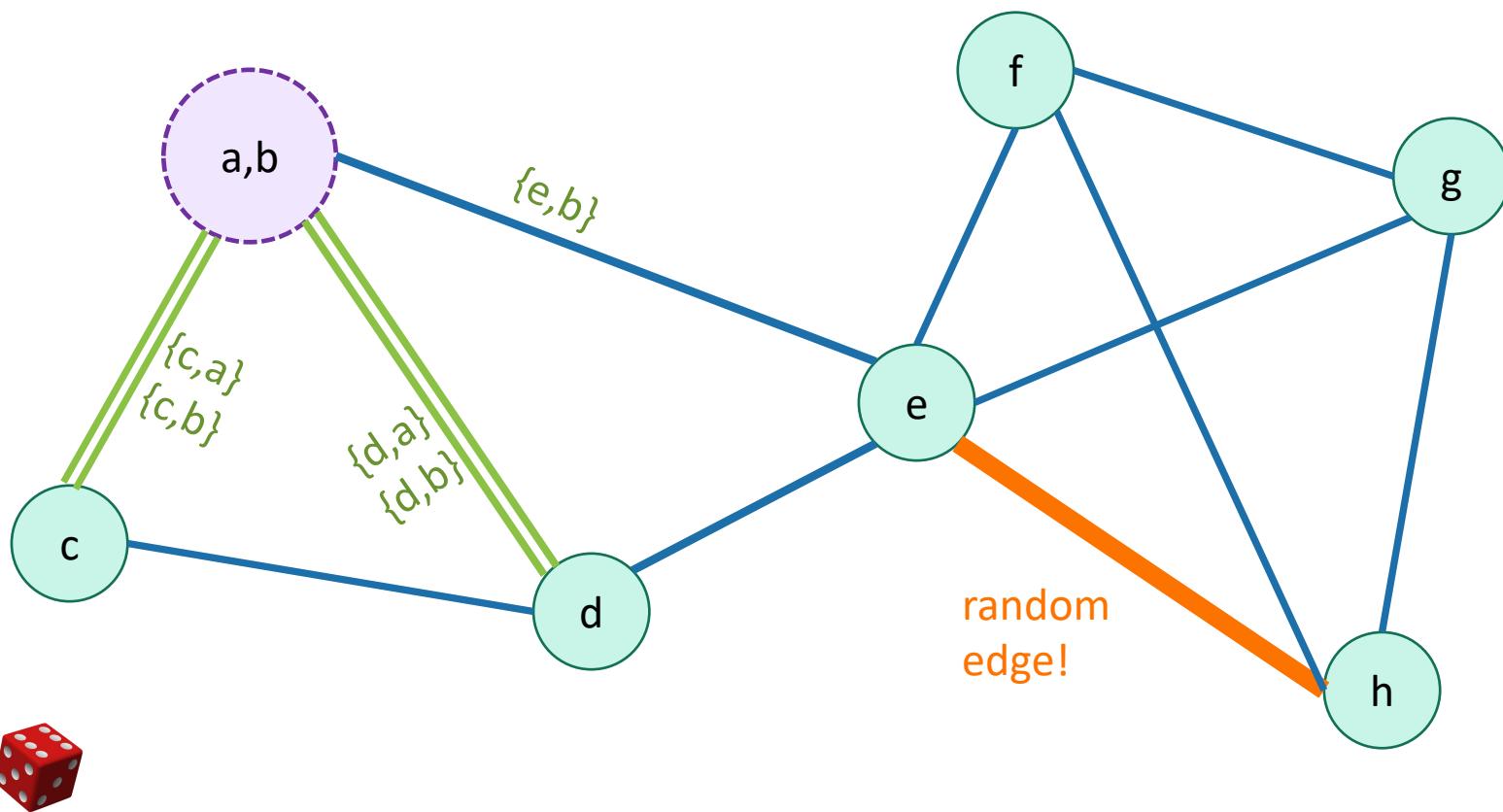
# Karger's algorithm



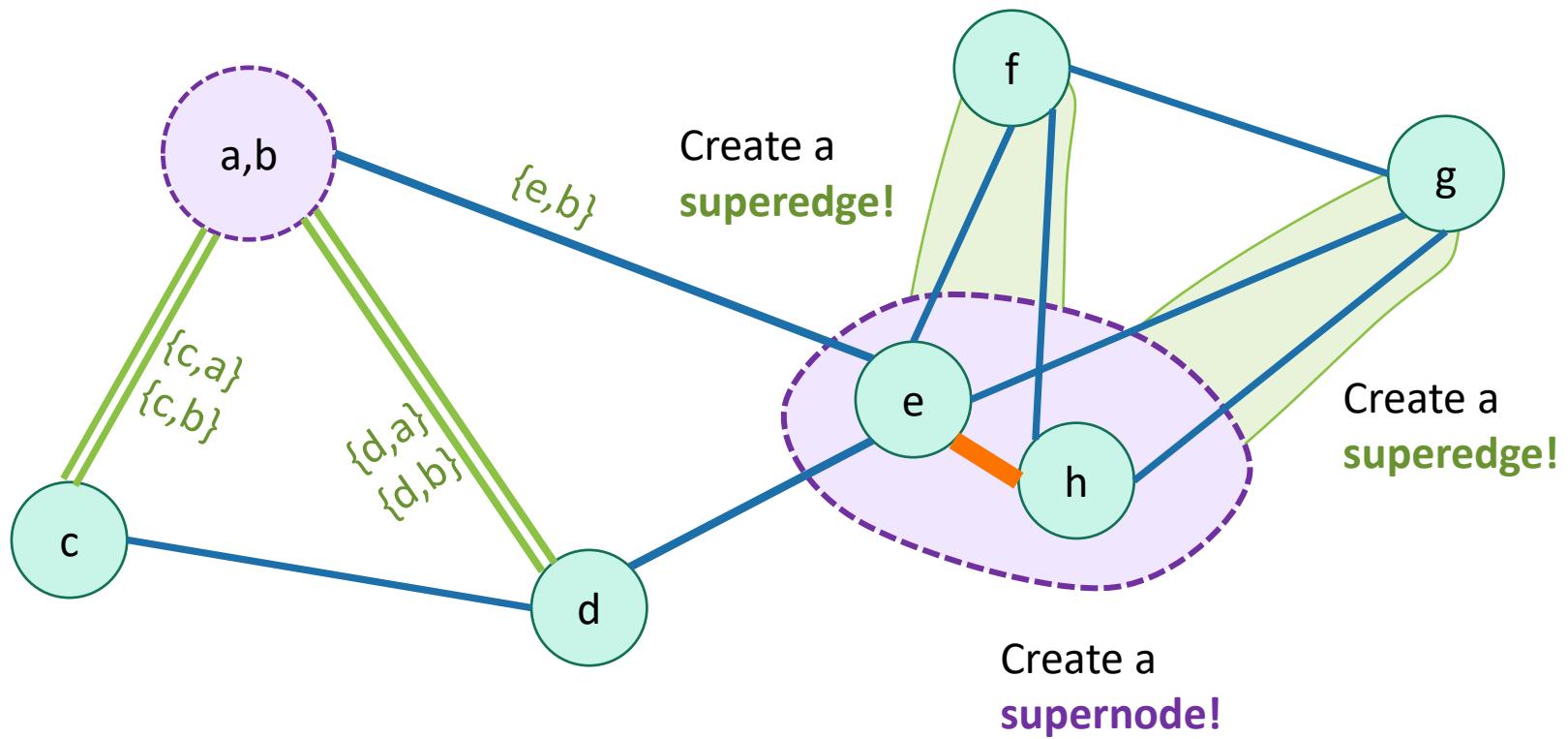
# Karger's algorithm



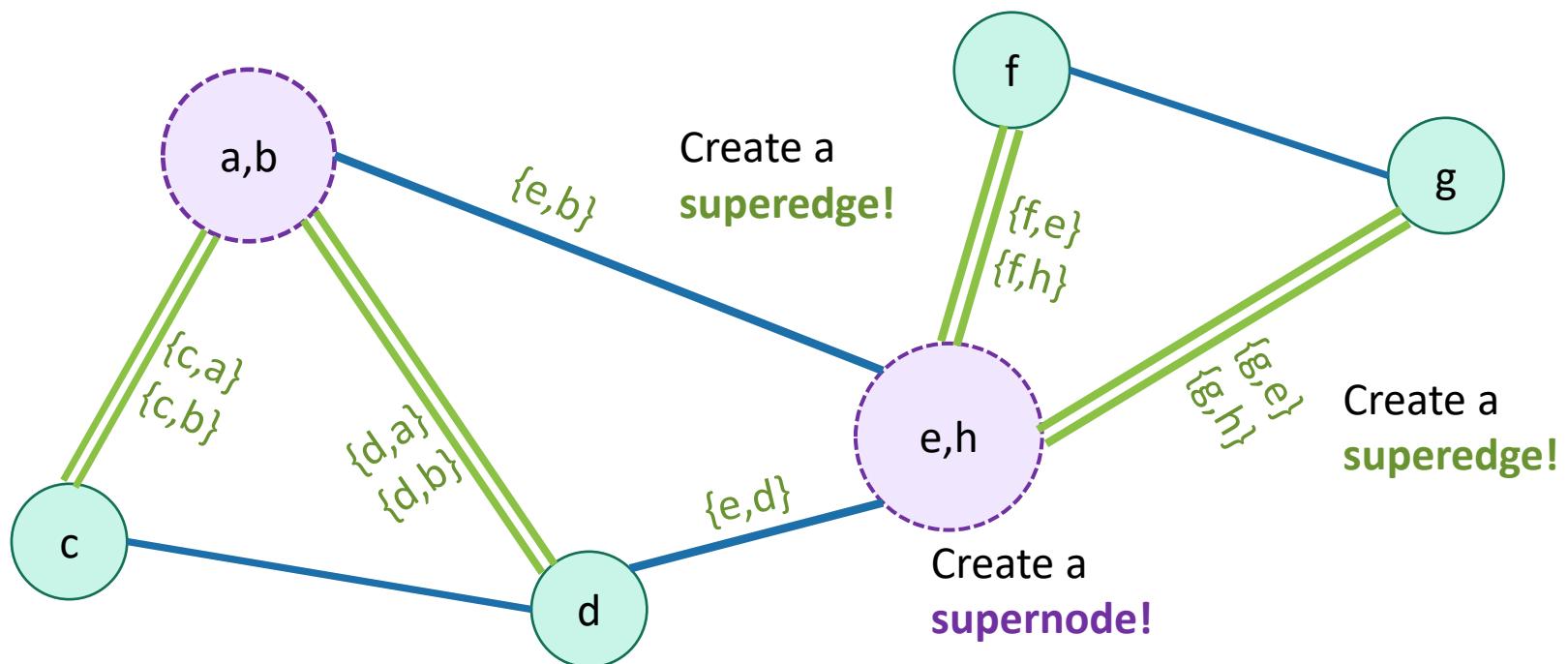
# Karger's algorithm



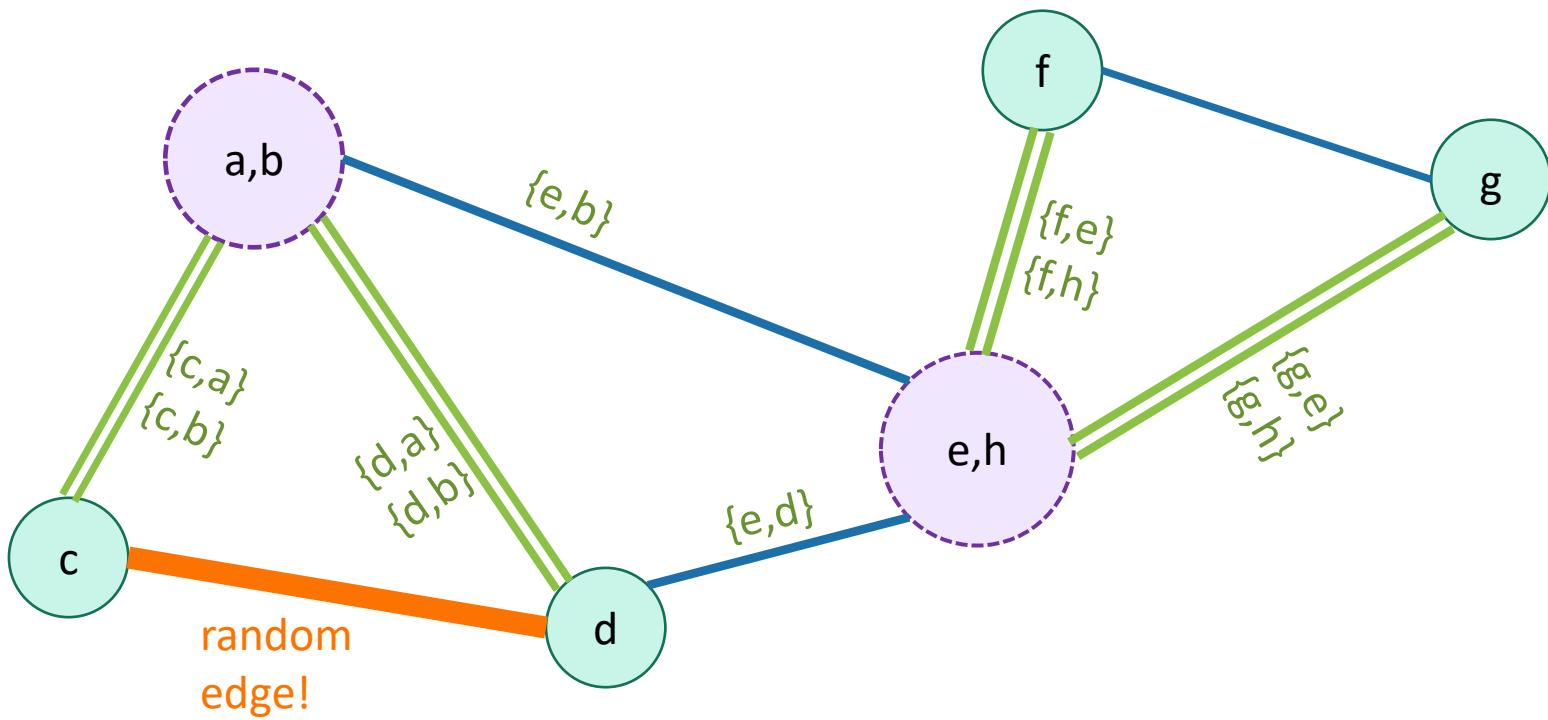
# Karger's algorithm



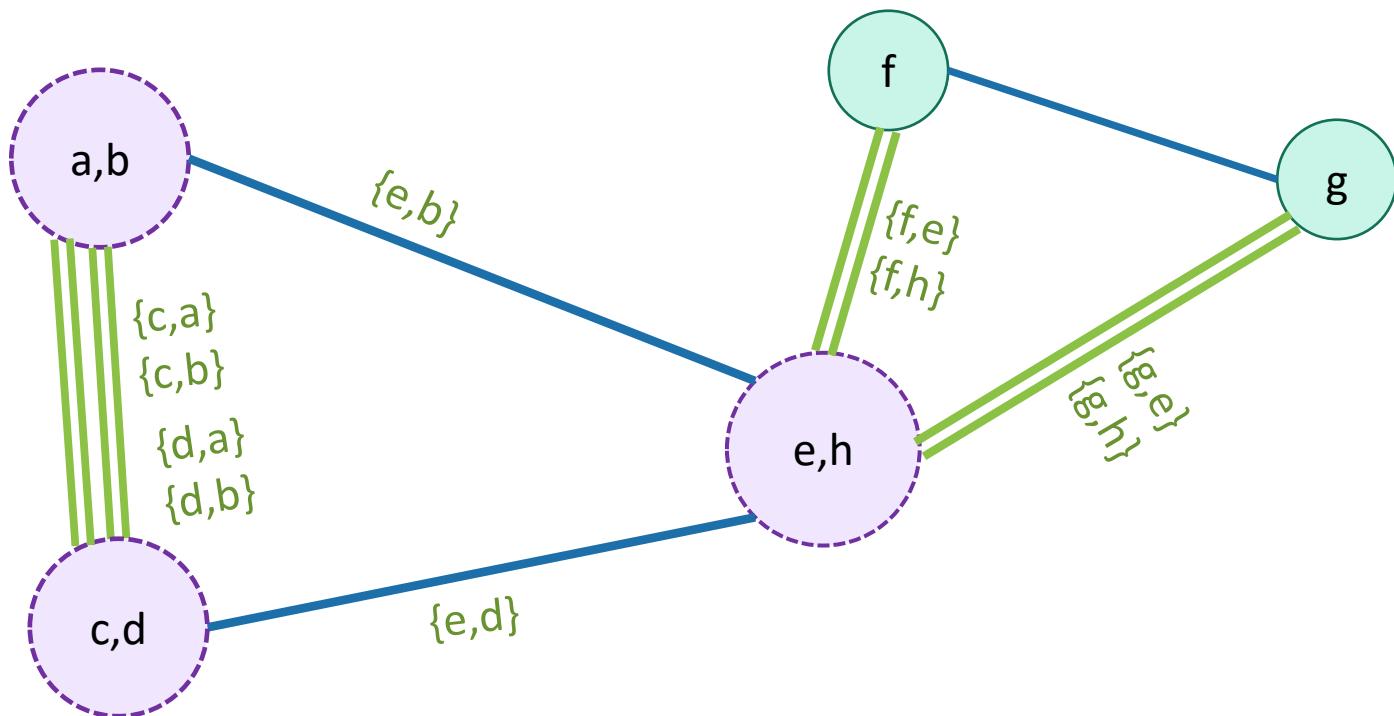
# Karger's algorithm



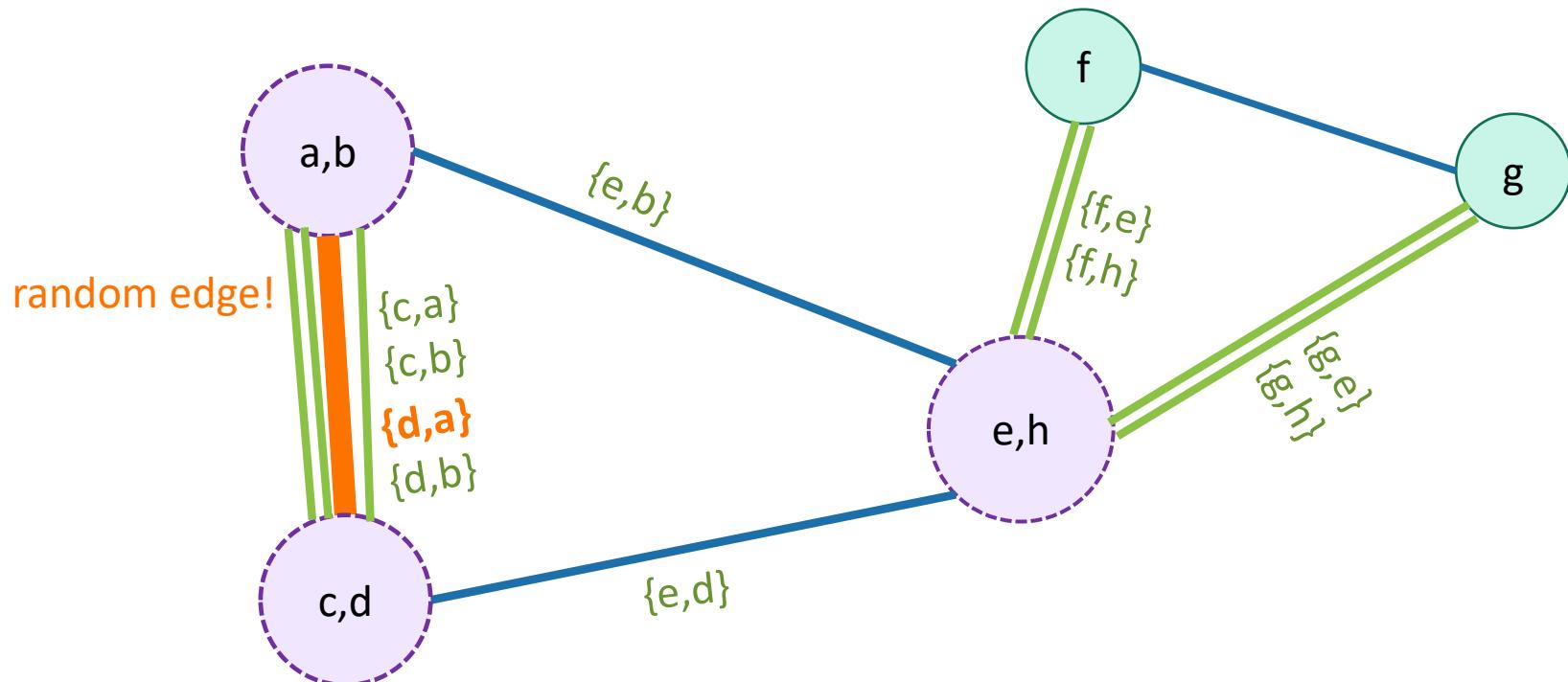
# Karger's algorithm



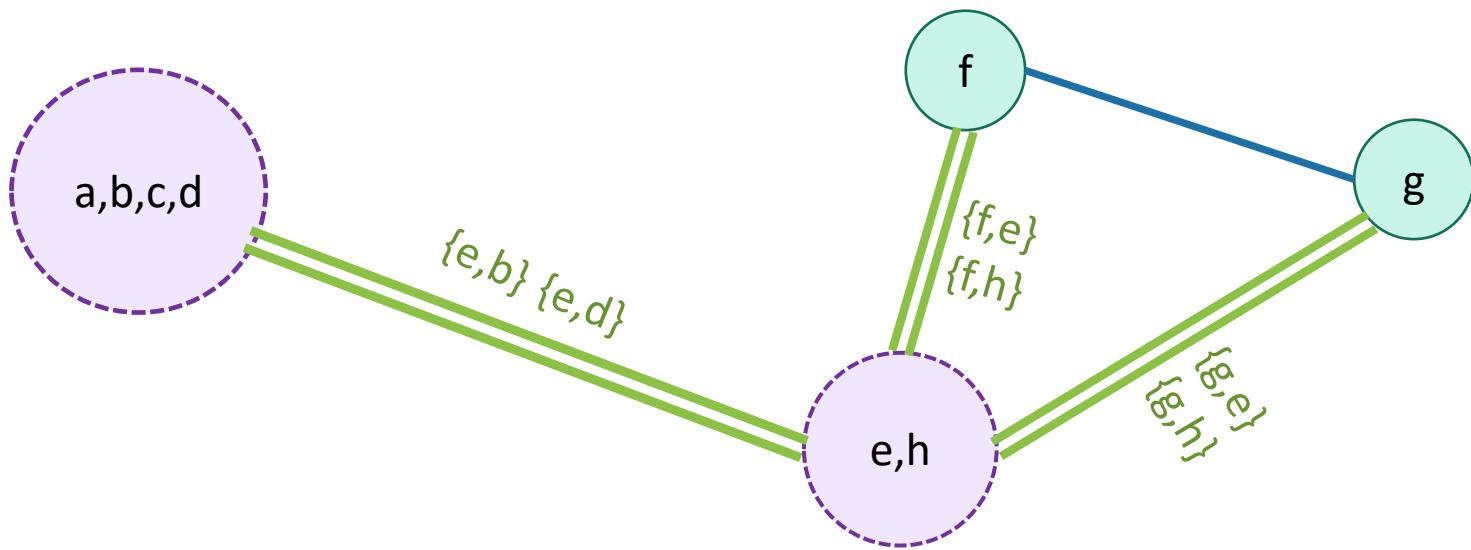
# Karger's algorithm



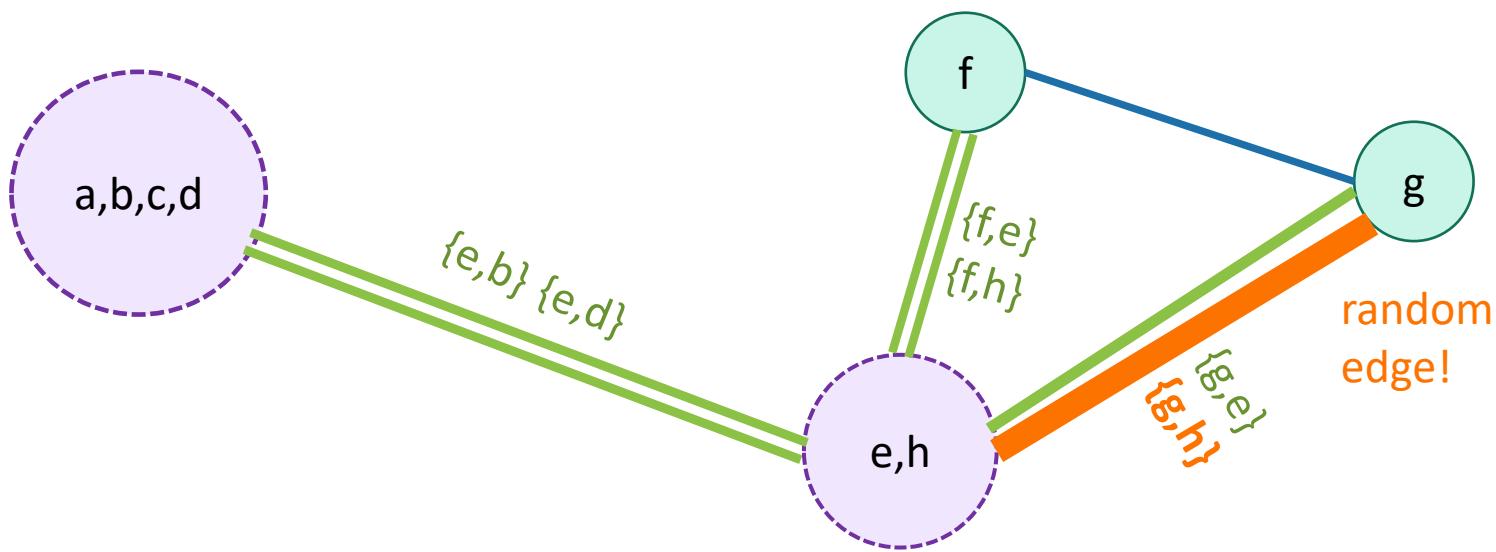
# Karger's algorithm



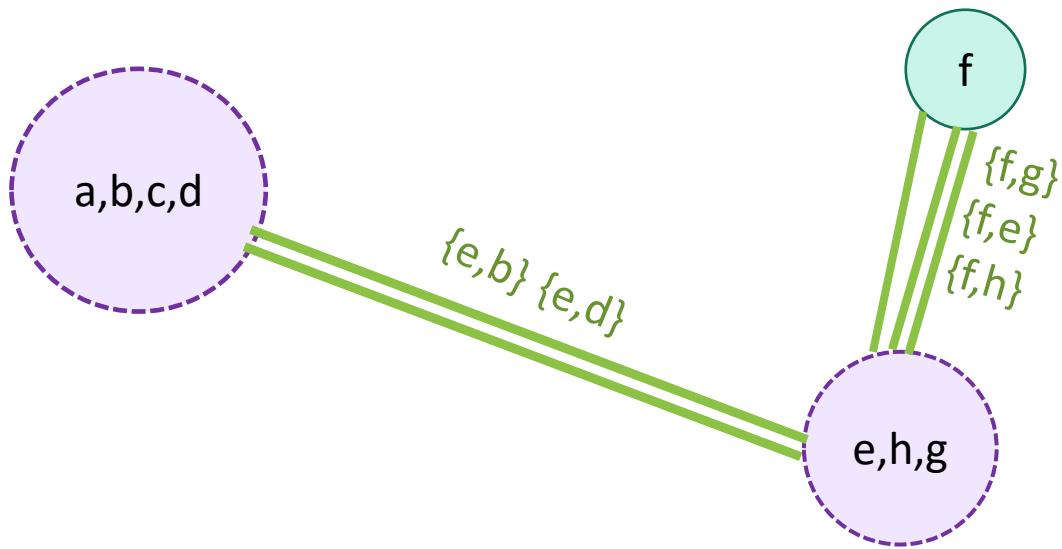
# Karger's algorithm



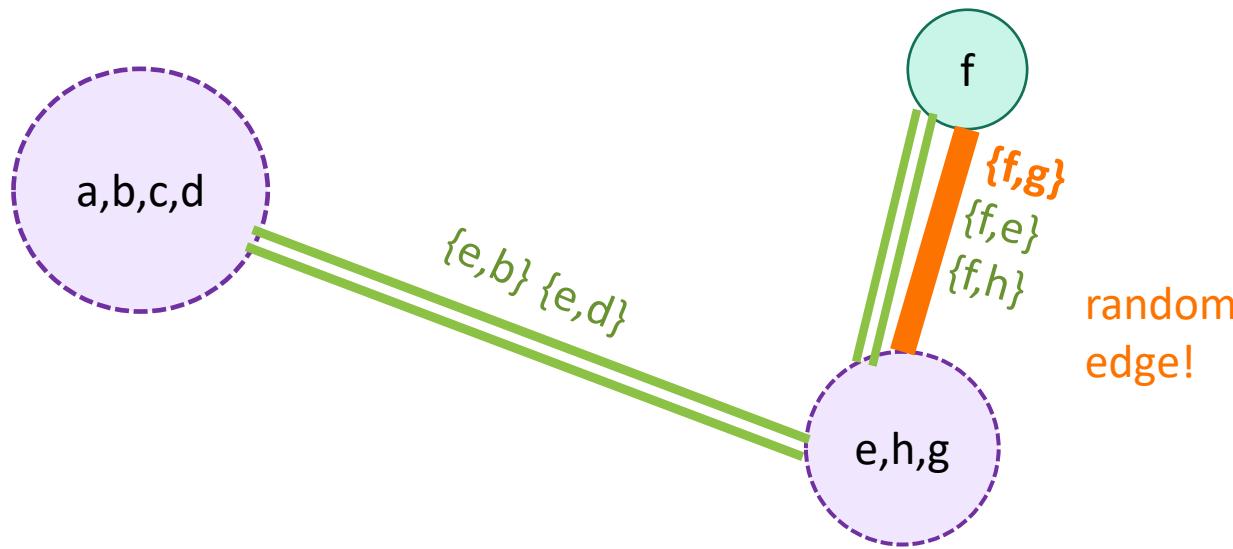
# Karger's algorithm



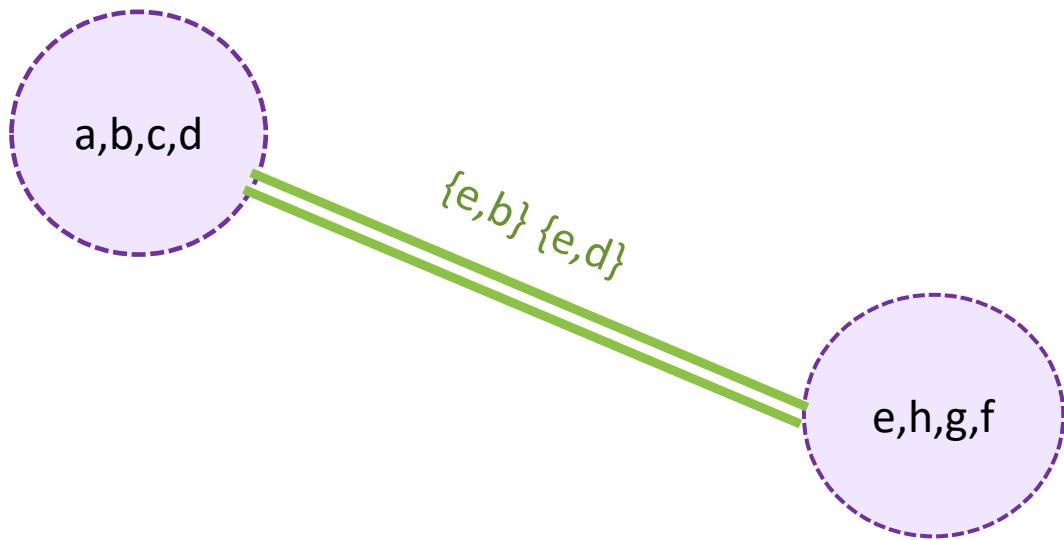
# Karger's algorithm



# Karger's algorithm



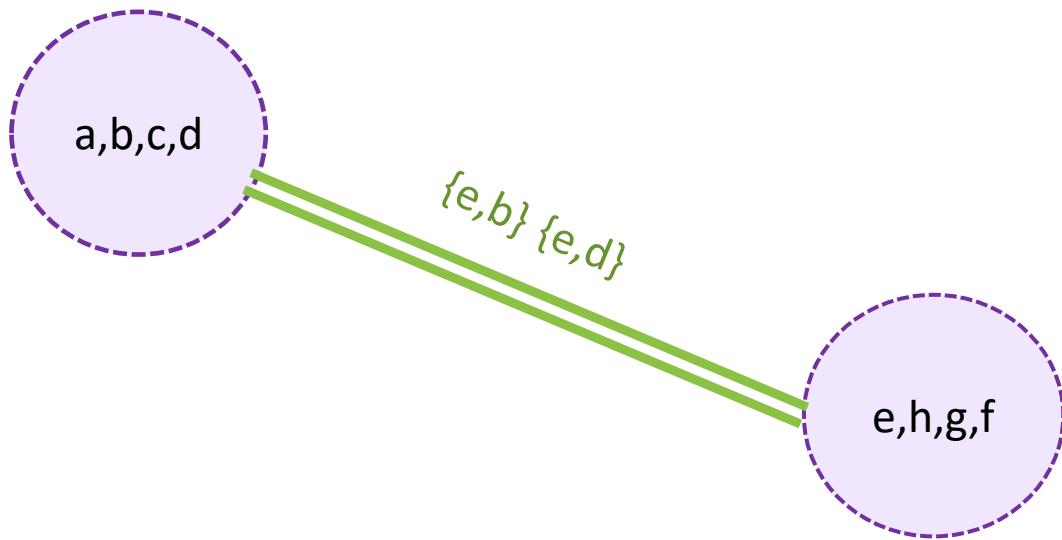
# Karger's algorithm



# Karger's algorithm

**Now stop!**

- There are only two nodes left.



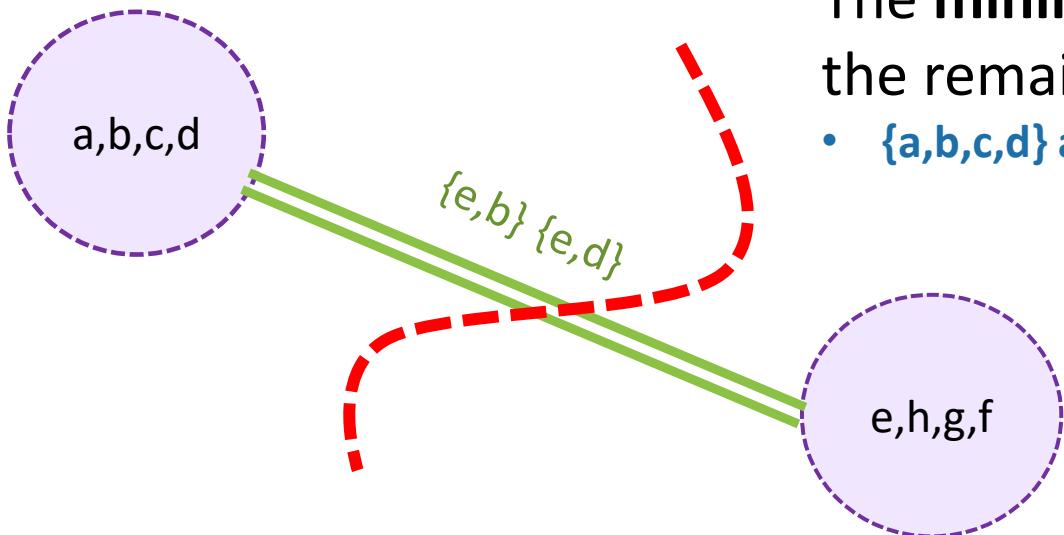
# Karger's algorithm

**Now stop!**

- There are only two nodes left.

The **minimum cut** is given by the remaining super-nodes:

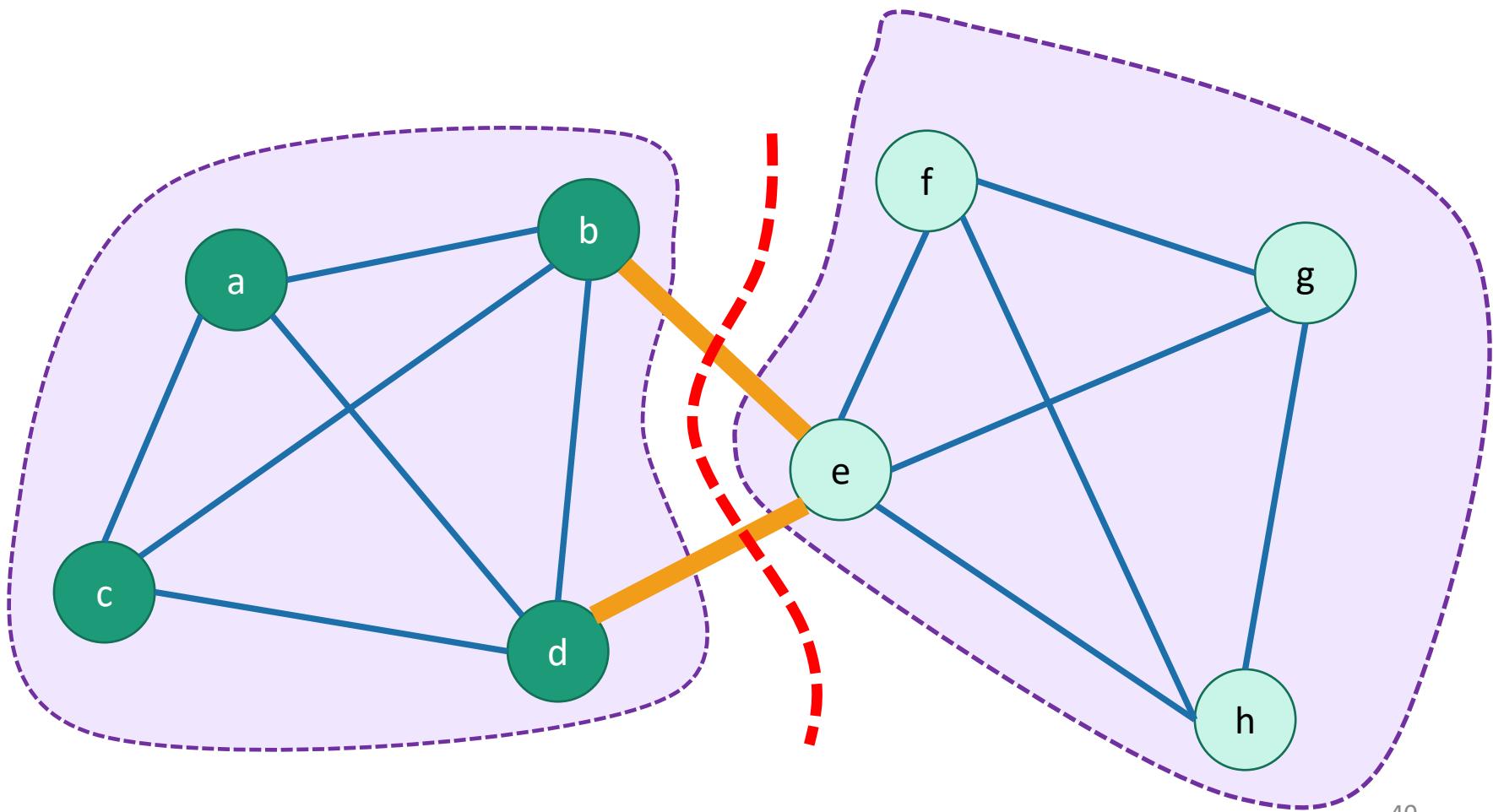
- {a,b,c,d} and {e,h,f,g}



# Karger's algorithm

The **minimum cut** is given by the remaining super-nodes:

- {a,b,c,d} and {e,h,f,g}



# Karger's algorithm

- Does it work?

- Is it fast?



# Pseudocode

Let  $\bar{u}$  denote the SuperNode in  $\Gamma$  containing  $u$   
Say  $E_{\bar{u}, \bar{v}}$  is the SuperEdge between  $\bar{u}, \bar{v}$ .

- **Karger( G=(V,E) ):**

- $\Gamma = \{ \text{SuperNode}(v) : v \text{ in } V \}$  // one supernode for each vertex
- $E_{\bar{u}, \bar{v}} = \{(u, v)\} \text{ for } (u, v) \text{ in } E$  // one superedge for each edge
- $E_{\bar{u}, \bar{v}} = \{\}$  for  $(u, v)$  not in  $E$ .
- $F = \text{copy of } E$  // we'll choose randomly from  $F$

# Pseudocode

Let  $\bar{u}$  denote the SuperNode in  $\Gamma$  containing  $u$   
Say  $E_{\bar{u}, \bar{v}}$  is the SuperEdge between  $\bar{u}, \bar{v}$ .

- **Karger( G=(V,E) ):**

- $\Gamma = \{ \text{SuperNode}(v) : v \text{ in } V \}$  // one supernode for each vertex
- $E_{\bar{u}, \bar{v}} = \{(u, v)\} \text{ for } (u, v) \text{ in } E$  // one superedge for each edge
- $E_{\bar{u}, \bar{v}} = \{\} \text{ for } (u, v) \text{ not in } E.$
- $F = \text{copy of } E$  // we'll choose randomly from F
- **while**  $|\Gamma| > 2$ :
  - $(u, v) \leftarrow \text{uniformly random edge in } F$

# Pseudocode

Let  $\bar{u}$  denote the SuperNode in  $\Gamma$  containing  $u$   
Say  $E_{\bar{u}, \bar{v}}$  is the SuperEdge between  $\bar{u}, \bar{v}$ .

- **Karger( G=(V,E) ):**

- $\Gamma = \{ \text{SuperNode}(v) : v \text{ in } V \}$  // one supernode for each vertex
- $E_{\bar{u}, \bar{v}} = \{(u,v)\} \text{ for } (u,v) \text{ in } E$  // one superedge for each edge
- $E_{\bar{u}, \bar{v}} = \{\} \text{ for } (u,v) \text{ not in } E.$
- $F = \text{copy of } E$  // we'll choose randomly from F
- **while**  $|\Gamma| > 2$ :
  - $(u,v) \leftarrow \text{uniformly random edge in } F$
  - **merge( u, v )**
    - // merge the SuperNode containing  $u$  with the SuperNode containing  $v$ .

# Pseudocode

Let  $\bar{u}$  denote the SuperNode in  $\Gamma$  containing  $u$   
Say  $E_{\bar{u}, \bar{v}}$  is the SuperEdge between  $\bar{u}, \bar{v}$ .

- **Karger( G=(V,E) ):**

- $\Gamma = \{ \text{SuperNode}(v) : v \text{ in } V \}$  // one supernode for each vertex
- $E_{\bar{u}, \bar{v}} = \{(u,v)\} \text{ for } (u,v) \text{ in } E$  // one superedge for each edge
- $E_{\bar{u}, \bar{v}} = \{\} \text{ for } (u,v) \text{ not in } E.$
- $F = \text{copy of } E$  // we'll choose randomly from F
- **while**  $|\Gamma| > 2$ :
  - $(u,v) \leftarrow \text{uniformly random edge in } F$
  - **merge( u, v )**  
// merge the SuperNode containing  $u$  with the SuperNode containing  $v$ .

- **merge( u, v ):**

- $\bar{x} = \text{SuperNode}( \bar{u} \cup \bar{v} )$  // merge also knows about  $\Gamma$  and the  $E_{\bar{u}, \bar{v}}$ 's
- **for each**  $w$  in  $\Gamma - \{\bar{u}, \bar{v}\}$ :
  - $E_{\bar{x}, \bar{w}} = E_{\bar{u}, \bar{w}} \cup E_{\bar{v}, \bar{w}}$  // create a new supernode
- Remove  $\bar{u}$  and  $\bar{v}$  from  $\Gamma$  and add  $\bar{x}$ .

# Pseudocode

Let  $\bar{u}$  denote the SuperNode in  $\Gamma$  containing  $u$   
Say  $E_{\bar{u}, \bar{v}}$  is the SuperEdge between  $\bar{u}, \bar{v}$ .

- **Karger( G=(V,E) ):**

- $\Gamma = \{ \text{SuperNode}(v) : v \in V \}$  // one supernode for each vertex
- $E_{\bar{u}, \bar{v}} = \{(u, v)\} \text{ for } (u, v) \in E$  // one superedge for each edge
- $E_{\bar{u}, \bar{v}} = \{\} \text{ for } (u, v) \notin E.$
- $F = \text{copy of } E$  // we'll choose randomly from F
- **while**  $|\Gamma| > 2$ :
  - $(u, v) \leftarrow \text{uniformly random edge in } F$
  - **merge( u, v )**
    - // merge the SuperNode containing  $u$  with the SuperNode containing  $v$ .
  - $F \leftarrow F - E_{\bar{u}, \bar{v}}$ 
    - // remove all the edges in the SuperEdge between those SuperNodes.

- **merge( u, v ):**

- $\bar{x} = \text{SuperNode}( \bar{u} \cup \bar{v} )$  // merge also knows about  $\Gamma$  and the  $E_{\bar{u}, \bar{v}}$ 's
- **for each**  $w$  in  $\Gamma - \{\bar{u}, \bar{v}\}$ :
  - $E_{\bar{x}, \bar{w}} = E_{\bar{u}, \bar{w}} \cup E_{\bar{v}, \bar{w}}$  // create a new supernode
  - Remove  $\bar{u}$  and  $\bar{v}$  from  $\Gamma$  and add  $\bar{x}$ .

# Pseudocode

Let  $\bar{u}$  denote the SuperNode in  $\Gamma$  containing  $u$   
Say  $E_{\bar{u}, \bar{v}}$  is the SuperEdge between  $\bar{u}, \bar{v}$ .

- **Karger( G=(V,E) ):**

- $\Gamma = \{ \text{SuperNode}(v) : v \in V \}$  // one supernode for each vertex
- $E_{\bar{u}, \bar{v}} = \{(u, v)\} \text{ for } (u, v) \in E$  // one superedge for each edge
- $E_{\bar{u}, \bar{v}} = \{\} \text{ for } (u, v) \notin E.$
- $F = \text{copy of } E$  // we'll choose randomly from F
- **while**  $|\Gamma| > 2$ :
  - $(u, v) \leftarrow \text{uniformly random edge in } F$
  - **merge( u, v )**
    - // merge the SuperNode containing  $u$  with the SuperNode containing  $v$ .
  - $F \leftarrow F - E_{\bar{u}, \bar{v}}$ 
    - // remove all the edges in the SuperEdge between those SuperNodes.
- **return the cut given by the remaining two superNodes.**

- **merge( u, v ):**

- $\bar{x} = \text{SuperNode}( \bar{u} \cup \bar{v} )$  // merge also knows about  $\Gamma$  and the  $E_{\bar{u}, \bar{v}}$ 's
- **for each w in  $\Gamma - \{\bar{u}, \bar{v}\}$ :**
  - $E_{\bar{x}, \bar{w}} = E_{\bar{u}, \bar{w}} \cup E_{\bar{v}, \bar{w}}$  // create a new supernode
- Remove  $\bar{u}$  and  $\bar{v}$  from  $\Gamma$  and add  $\bar{x}$ .

# Pseudocode

Let  $\bar{u}$  denote the SuperNode in  $\Gamma$  containing  $u$   
Say  $E_{\bar{u}, \bar{v}}$  is the SuperEdge between  $\bar{u}, \bar{v}$ .

- **Karger( G=(V,E) ):**

- $\Gamma = \{ \text{SuperNode}(v) : v \in V \}$

// one supernode for each vertex

- $E_{\bar{u}, \bar{v}} = \{(u, v)\}$  for  $(u, v)$  in  $E$

// one superedge for each edge

- $E_{\bar{u}, \bar{v}} = \{\}$  for  $(u, v)$  not in  $E$ .

- $F = \text{copy of } E$

// we'll choose randomly from  $F$

- **while**  $|\Gamma| > 2$ :

- $(u, v) \leftarrow \text{uniformly random edge in } F$

- **merge( u, v )**

// merge the SuperNode containing  $u$  with the SuperNode containing  $v$ .

- $F \leftarrow F - E_{\bar{u}, \bar{v}}$

// remove all the edges in the SuperEdge between those SuperNodes.

- **return the cut given by the remaining two superNodes.**

- **merge( u, v ):**

- $\bar{x} = \text{SuperNode}( \bar{u} \cup \bar{v} )$

// merge also knows about  $\Gamma$  and the  $E_{\bar{u}, \bar{v}}$ 's

- for each  $w$  in  $\Gamma - \{\bar{u}, \bar{v}\}$ :

// create a new supernode

- $E_{\bar{x}, \bar{w}} = E_{\bar{u}, \bar{w}} \cup E_{\bar{v}, \bar{w}}$

- Remove  $\bar{u}$  and  $\bar{v}$  from  $\Gamma$  and add  $\bar{x}$ .

The **while** loop runs  $n-2$  times

**merge** takes time  $O(n)$  naively

# Pseudocode

Let  $\bar{u}$  denote the SuperNode in  $\Gamma$  containing  $u$   
Say  $E_{\bar{u}, \bar{v}}$  is the SuperEdge between  $\bar{u}, \bar{v}$ .

- **Karger( G=(V,E) ):**

- $\Gamma = \{ \text{SuperNode}(v) : v \in V \}$

// one supernode for each vertex

- $E_{\bar{u}, \bar{v}} = \{(u, v)\}$  for  $(u, v)$  in  $E$

// one superedge for each edge

- $E_{\bar{u}, \bar{v}} = \{\}$  for  $(u, v)$  not in  $E$ .

- $F = \text{copy of } E$

// we'll choose randomly from  $F$

- **while**  $|\Gamma| > 2$ :

- $(u, v) \leftarrow \text{uniformly random edge in } F$

- **merge( u, v )**

// merge the SuperNode containing  $u$  with the SuperNode containing  $v$ .

- $F \leftarrow F - E_{\bar{u}, \bar{v}}$

// remove all the edges in the SuperEdge between those SuperNodes.

- **return the cut given by the remaining two superNodes.**

- **merge( u, v ):**

- $\bar{x} = \text{SuperNode}( \bar{u} \cup \bar{v} )$

// merge also knows about  $\Gamma$  and the  $E_{\bar{u}, \bar{v}}$ 's

- for each  $w$  in  $\Gamma - \{\bar{u}, \bar{v}\}$ :

// create a new supernode

- $E_{\bar{x}, \bar{w}} = E_{\bar{u}, \bar{w}} \cup E_{\bar{v}, \bar{w}}$

**total runtime  $O(n^2)$**

- Remove  $\bar{u}$  and  $\bar{v}$  from  $\Gamma$  and add  $\bar{x}$ .

We can do a bit better with fancy data structures, but let's go with this for now.

# How do we implement this?

- Maintain a secondary “superGraph” which keeps track of superNodes and superEdges
- Running time?
  - We contract  $n-2$  edges
    - Each time we contract an edge we get rid of a vertex, and we get rid of  $n - 2$  vertices total.
  - Naively each contraction takes time  $O(n)$ 
    - Maybe there are  $\Omega(n)$  nodes in the superNodes that we are merging. (We can do better with fancy data structures).
  - So total running time  $O(n^2)$ .
    - We can do  $O(m \cdot \alpha(n))$  with a union-find data structure, but  $O(n^2)$  is good enough for today.

# Karger's algorithm

- Does it work?

- Is it fast?

- $O(n^2)$

# Karger's algorithm

- Does it work?



- Is it fast?

- $O(n^2)$

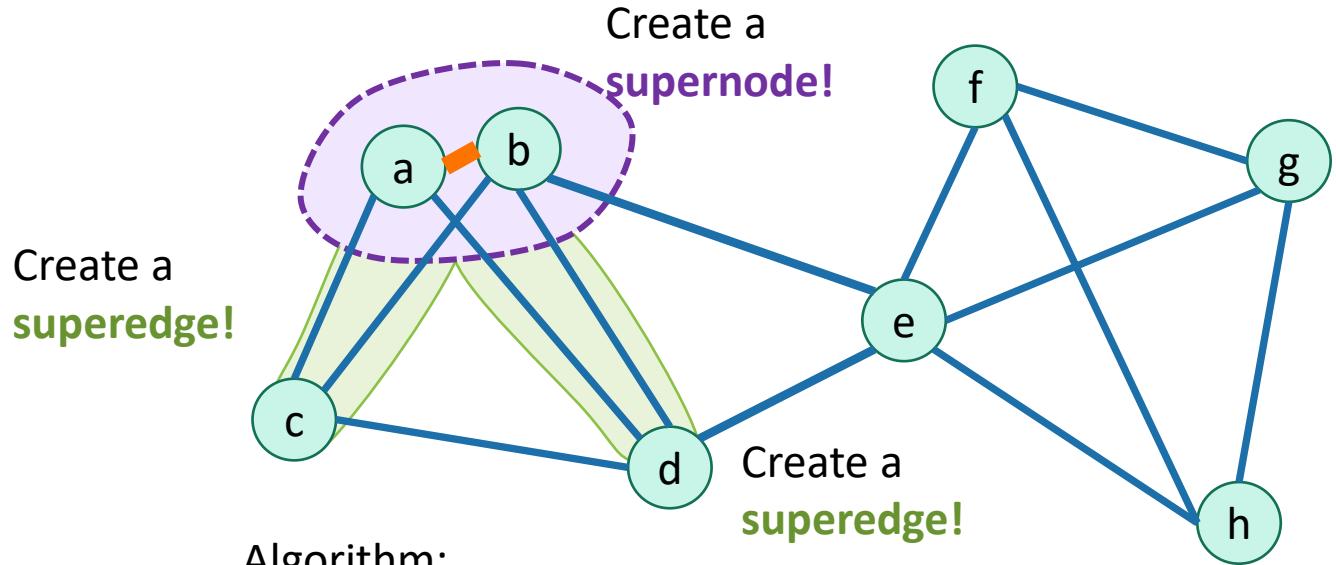
# Karger's algorithm

- Does it work?



- Is it fast?

- $O(n^2)$

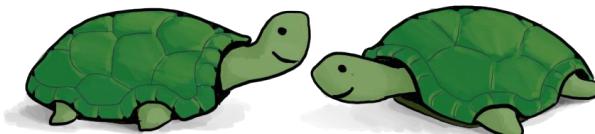


Algorithm:

- Randomly contract edges until there are only two supernodes left.

# Karger's algorithm

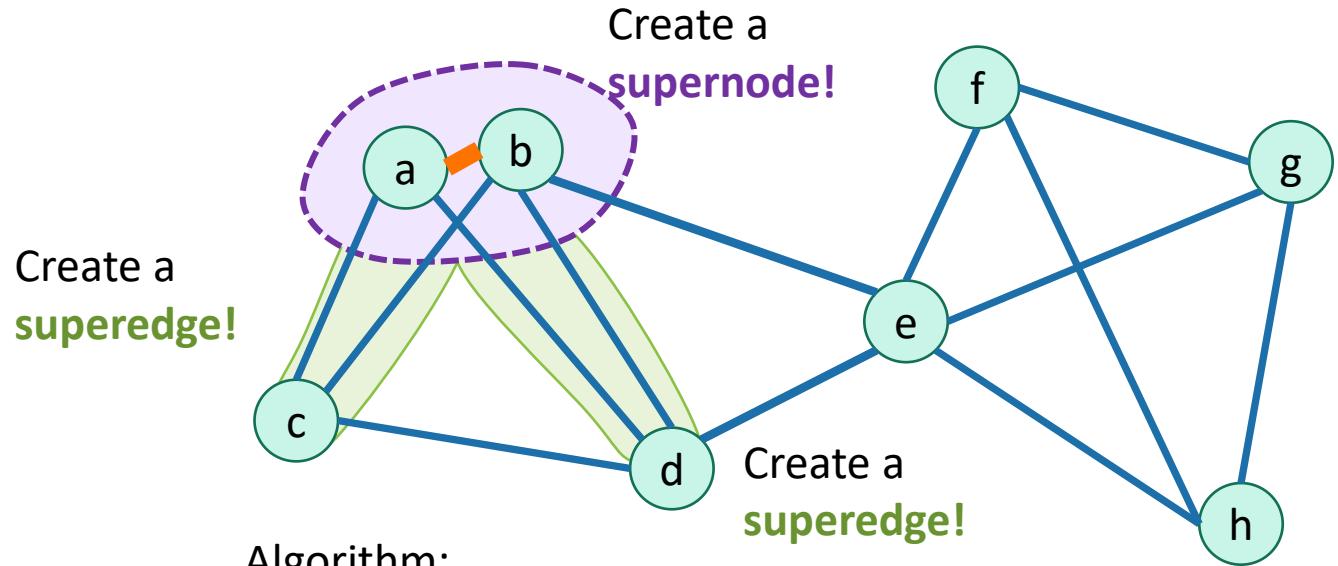
- Does it work?



Think-pair-share!

- Is it fast?

- $O(n^2)$



Algorithm:

- Randomly contract edges until there are only two supernodes left.

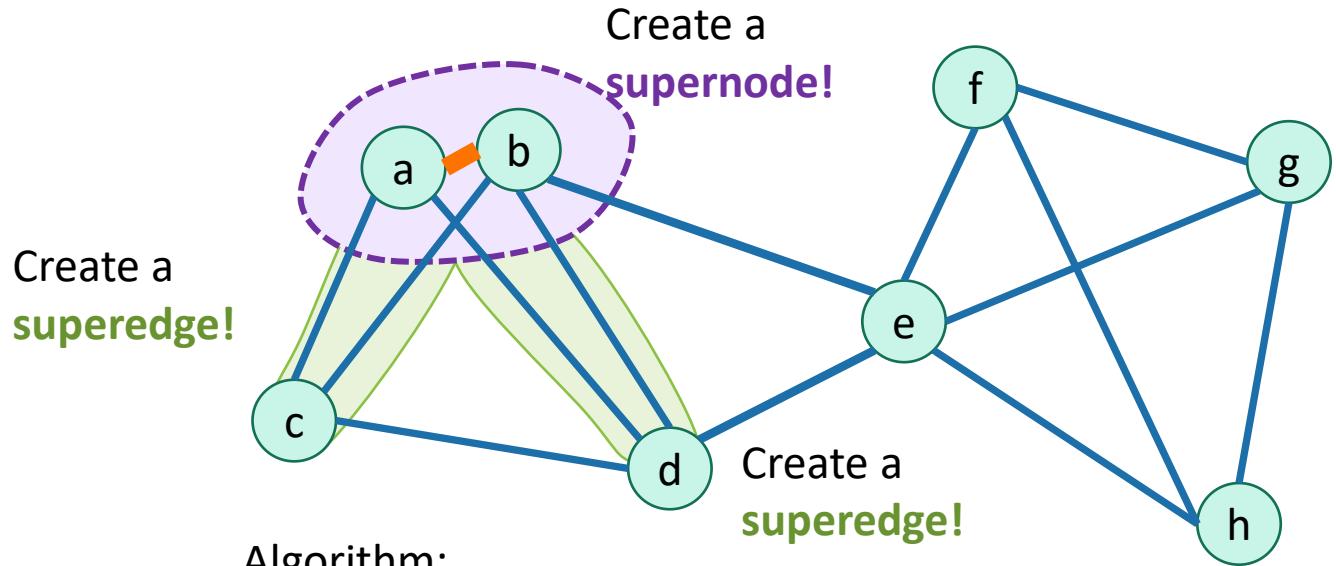
# Karger's algorithm

- Does it work?

No?

- Is it fast?

- $O(n^2)$



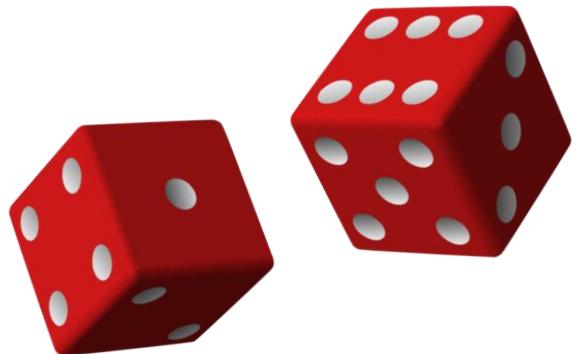
Algorithm:

- Randomly contract edges until there are only two supernodes left.

# Why did that work?

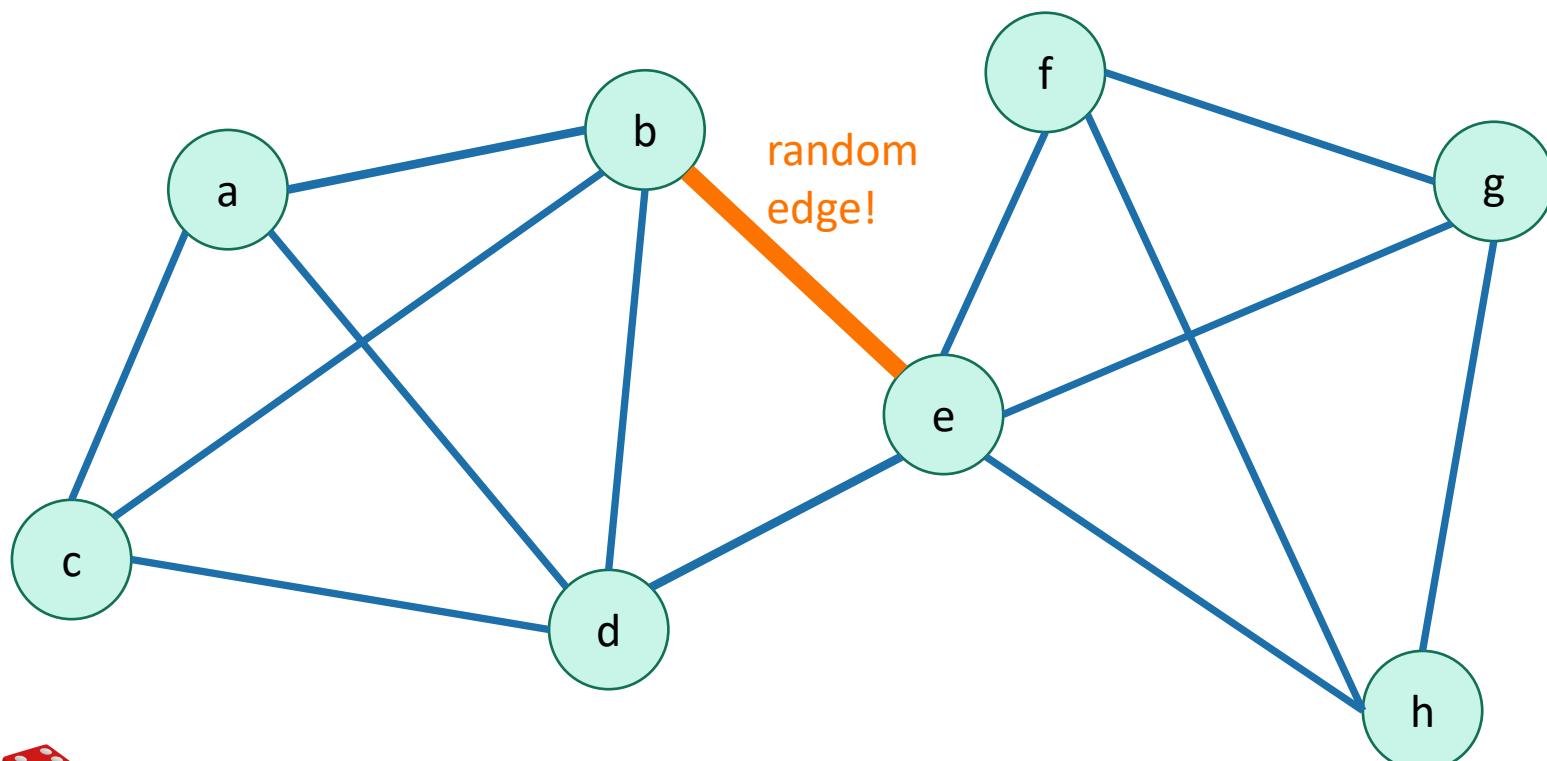
# Why did that work?

- We got really lucky!
- This could have gone wrong in so many ways.



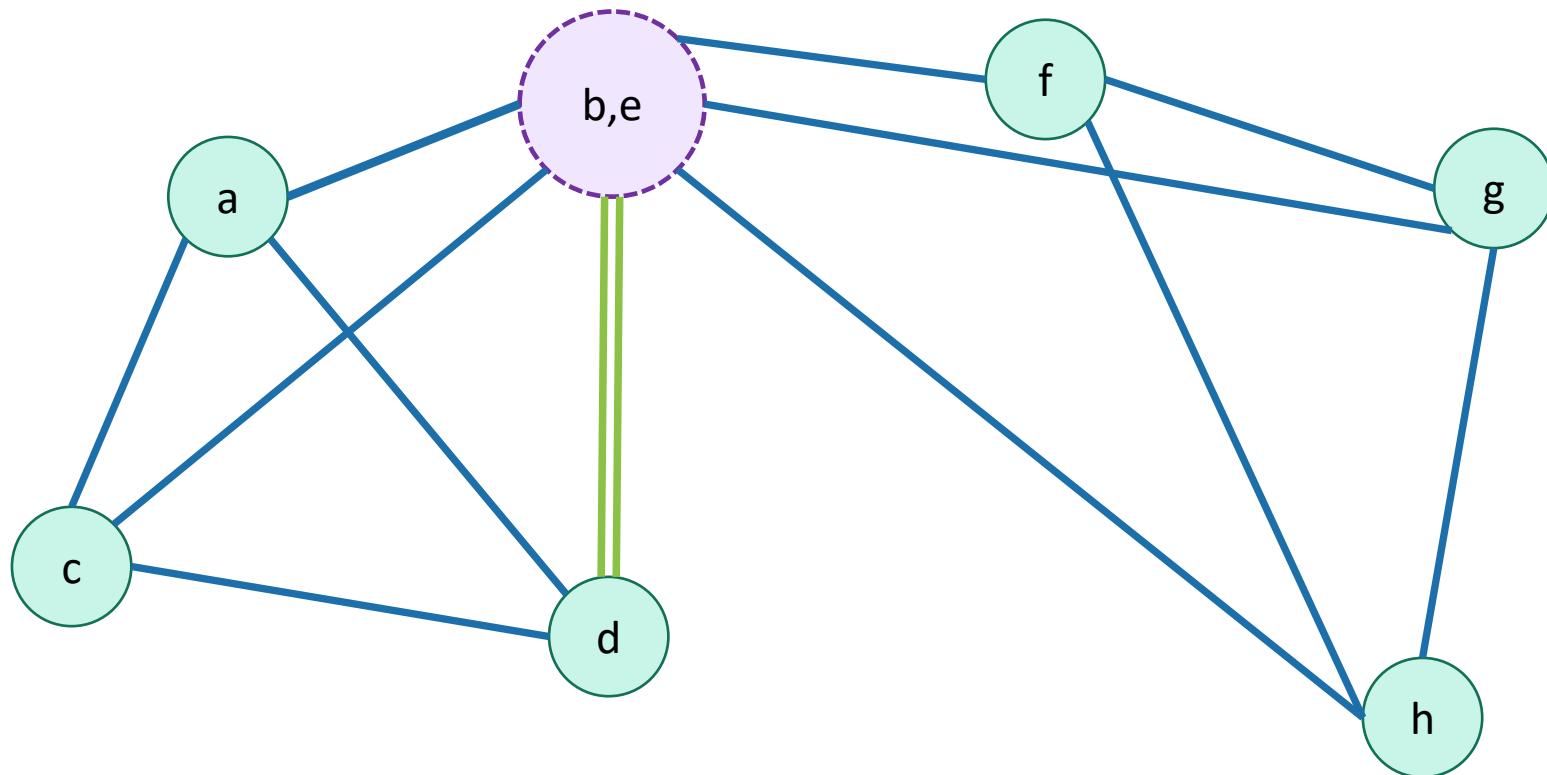
# Karger's algorithm

Say we had chosen this edge



# Karger's algorithm

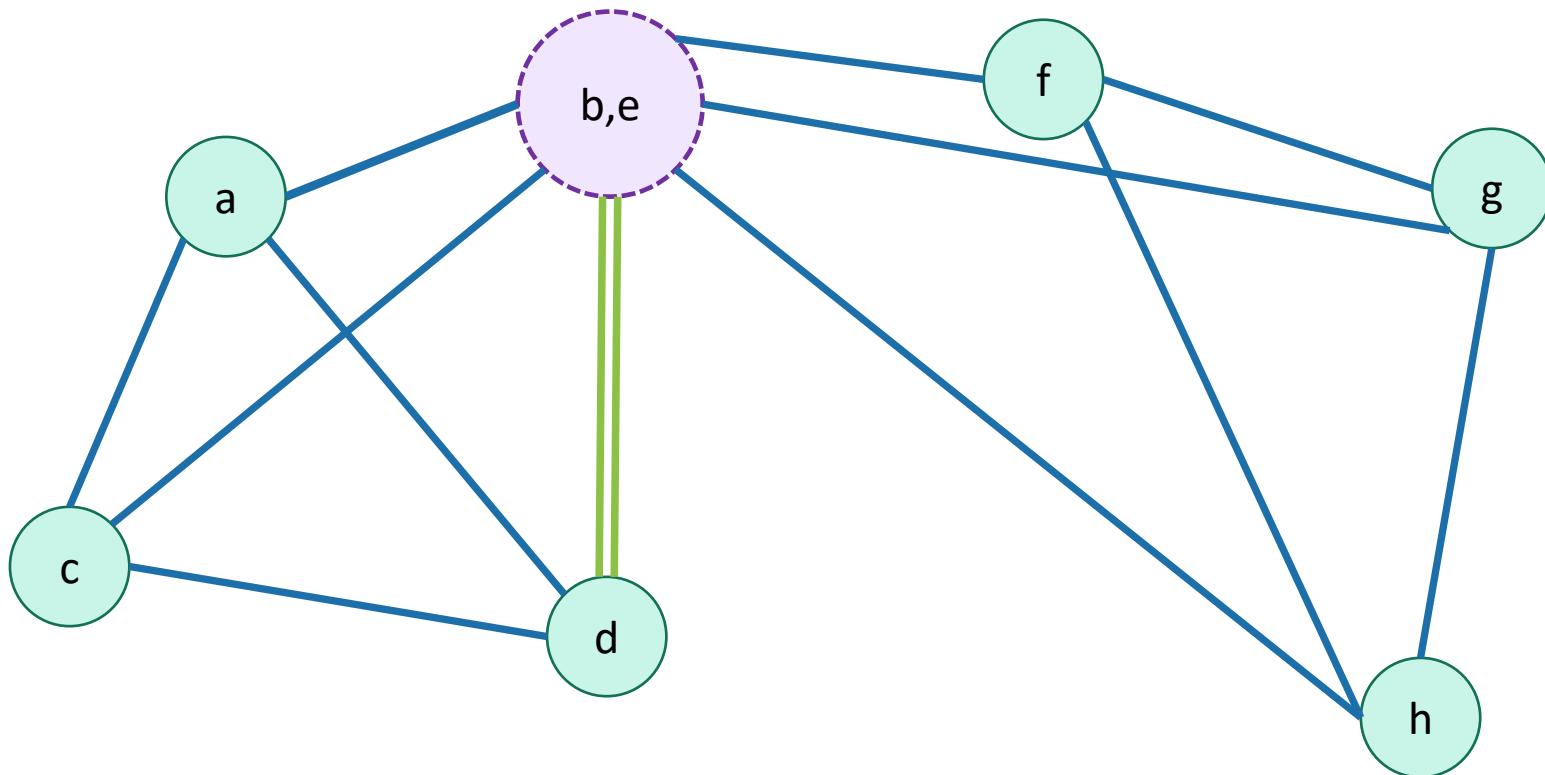
Say we had chosen this edge



# Karger's algorithm

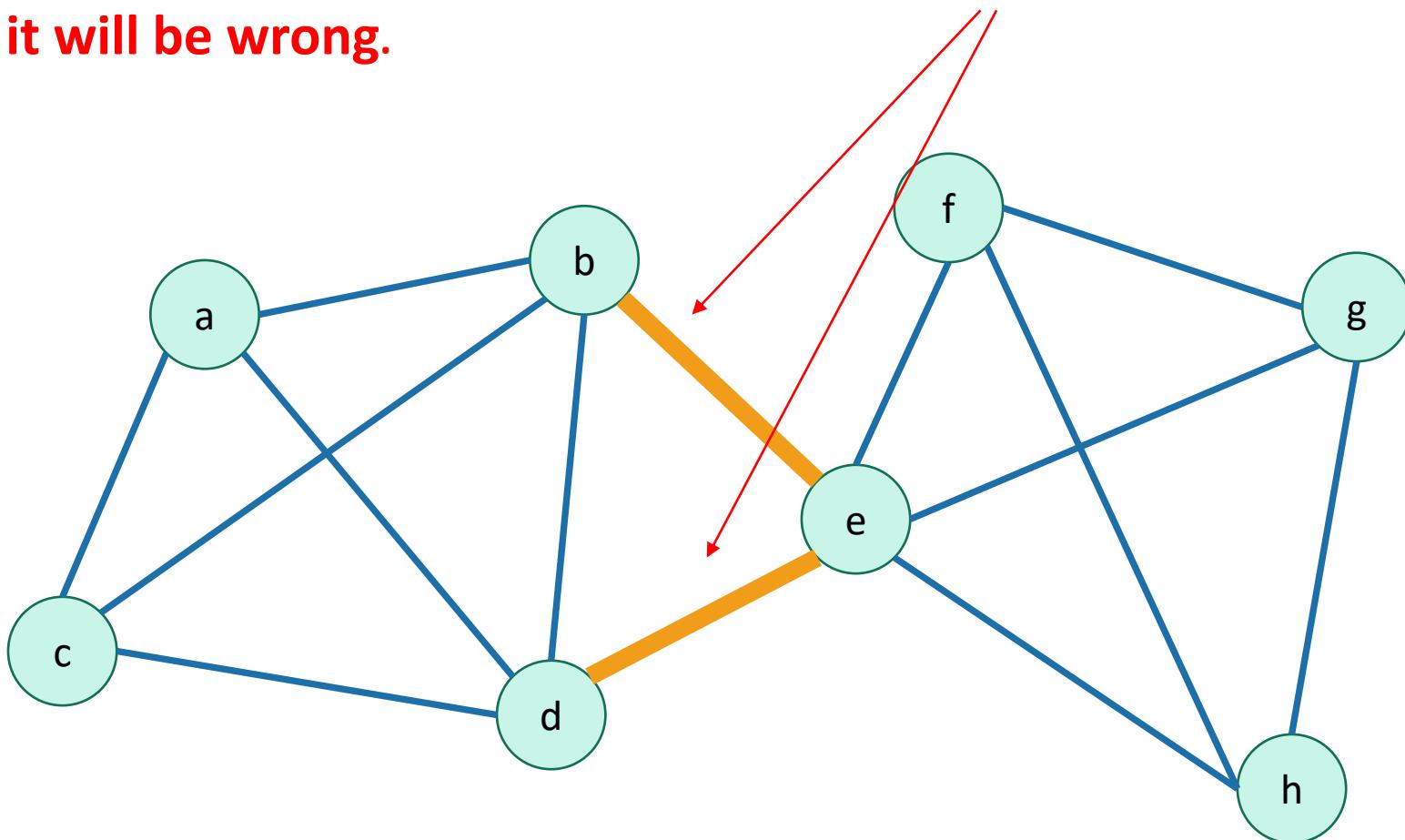
Say we had chosen this edge

Now there is **no way** we could return a cut  
that separates b and e.

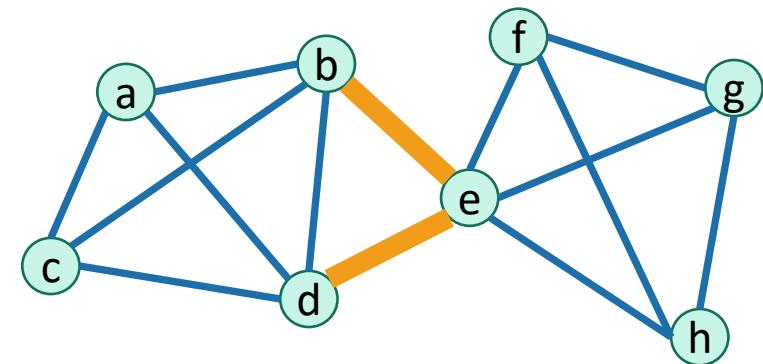


# Even worse

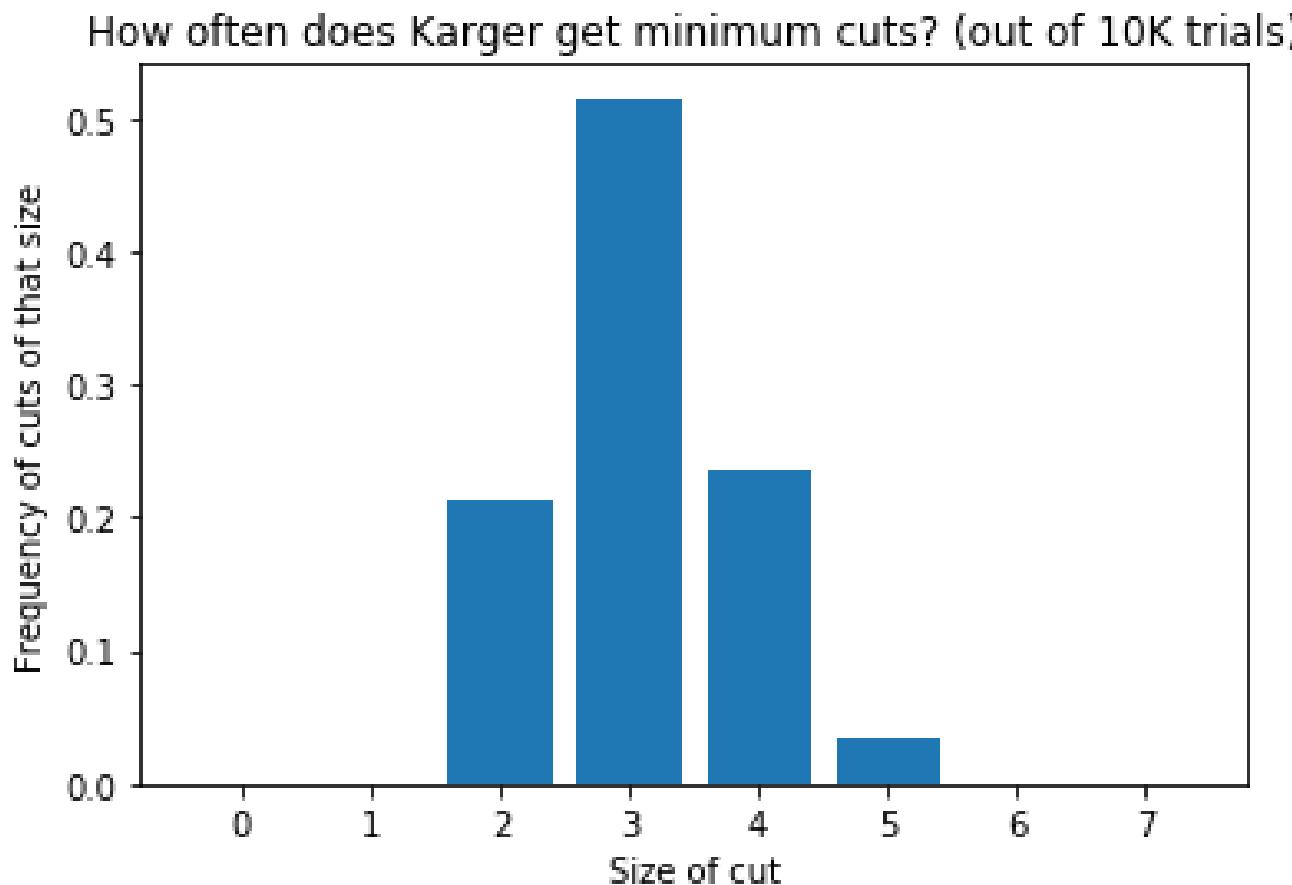
If the algorithm **EVER** chooses either of **these edges**,  
**it will be wrong.**



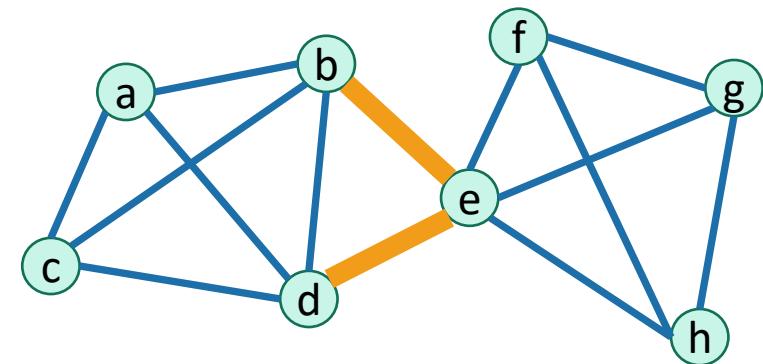
# How likely is that?



- For this particular graph, if we do 10,000 times:

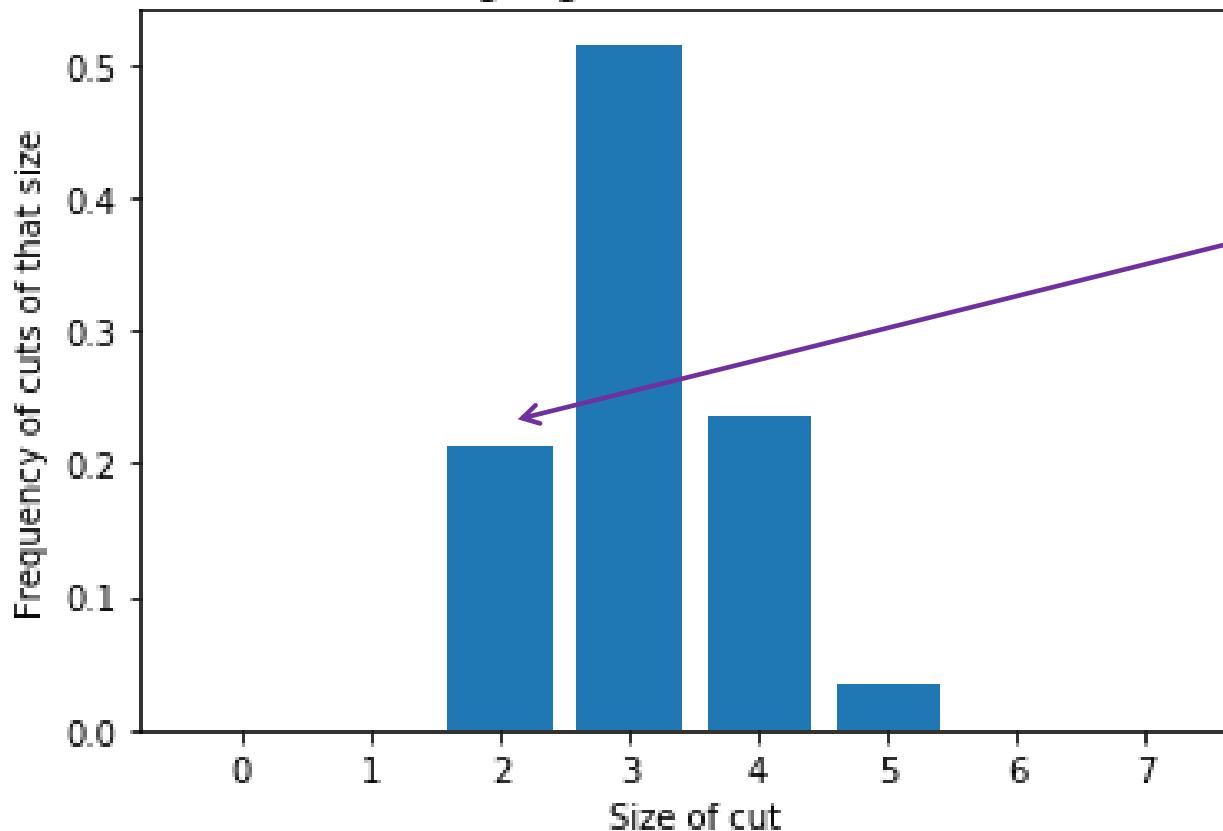


# How likely is that?



- For this particular graph, if we do 10,000 times:

How often does Karger get minimum cuts? (out of 10K trials)

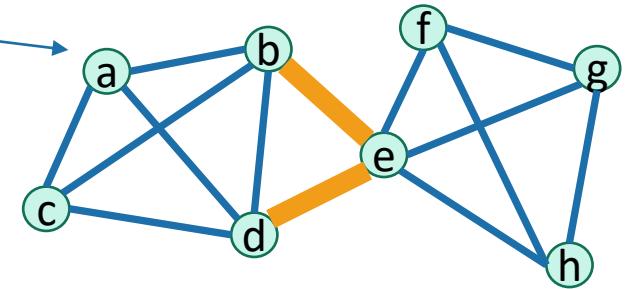


The algorithm  
is only correct  
about 20% of  
the time!

That doesn't sound good

# That doesn't sound good

- To see why it's good after all, we'll do a case study of this graph.

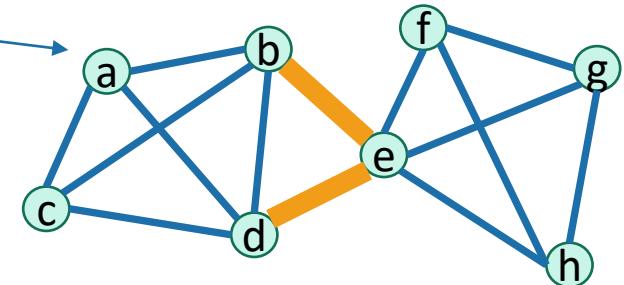


# That doesn't sound good

- To see why it's good after all, we'll do a case study of this graph.

The plan:

- See that 20% chance of correctness is actually nontrivial.
- Use repetition to boost an algorithm that's correct 20% of the time to an algorithm that's correct 99% of the time.

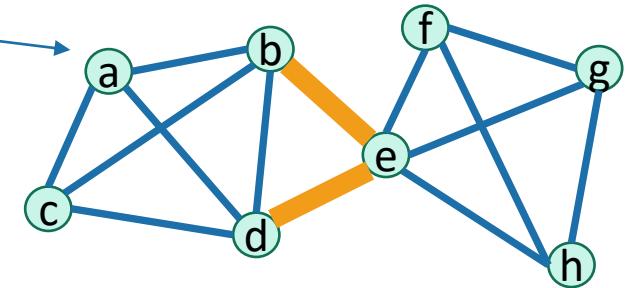


# That doesn't sound good

- To see why it's good after all, we'll do a case study of this graph.

The plan:

- See that 20% chance of correctness is actually nontrivial.
- Use repetition to boost an algorithm that's correct 20% of the time to an algorithm that's correct 99% of the time.



- To see the first point, let's compare Karger's algorithm to the algorithm:

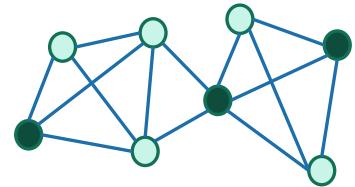
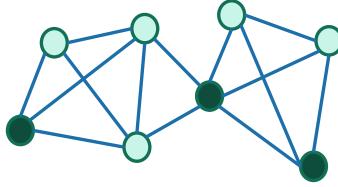
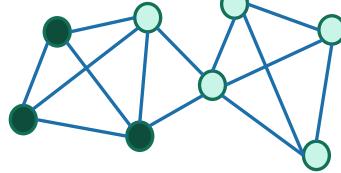
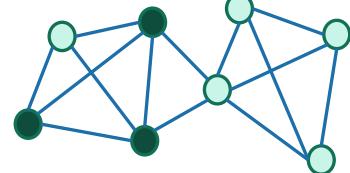
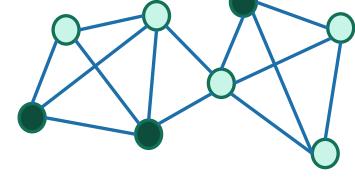
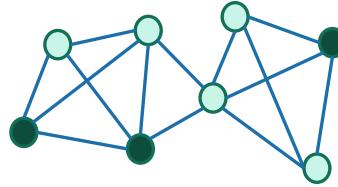
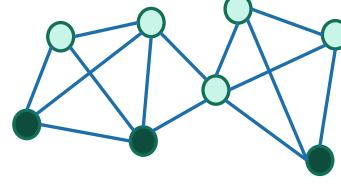
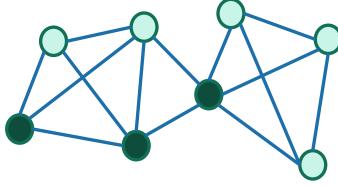
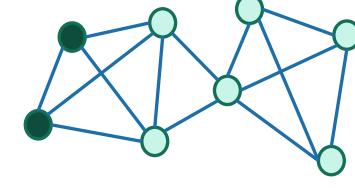
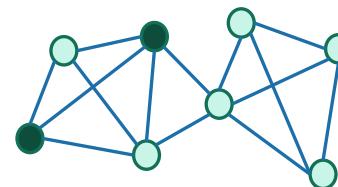
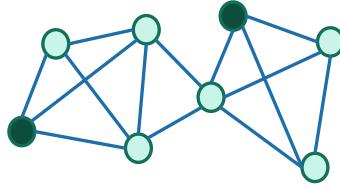
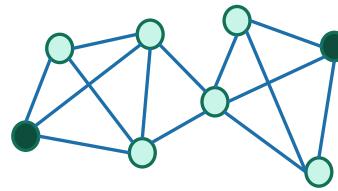
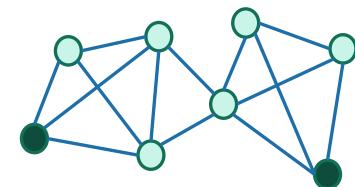
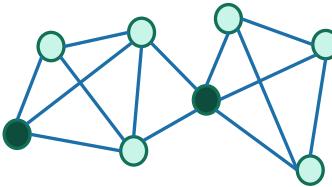
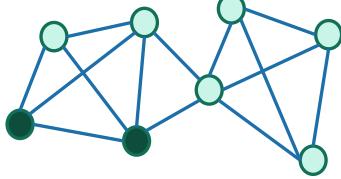
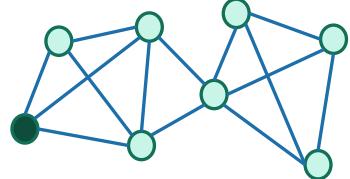
*Choose a completely random cut and hope that it's a minimum cut.*

# Uniformly random cut

- Pick a random way to split the vertices into two parts:

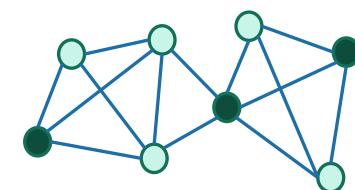
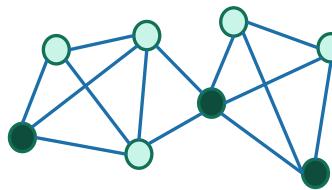
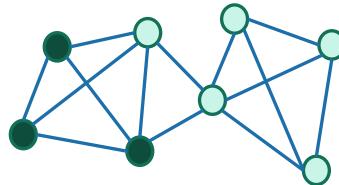
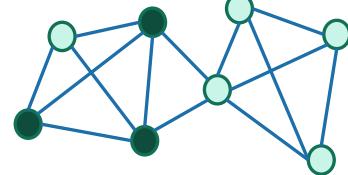
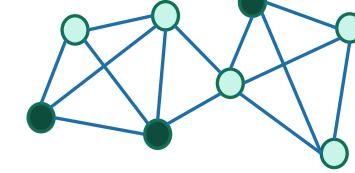
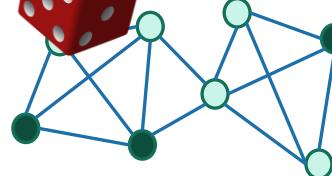
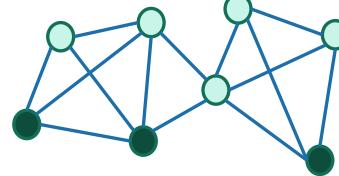
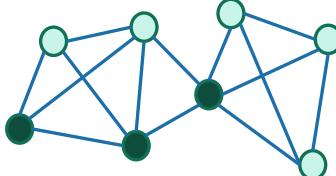
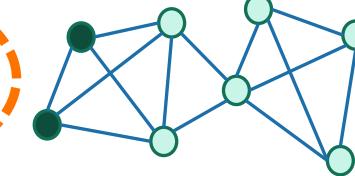
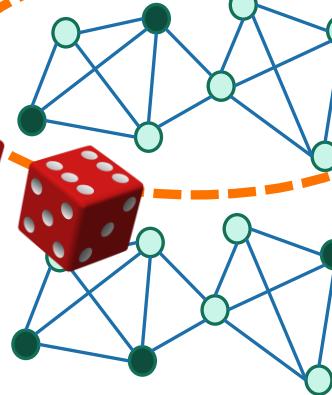
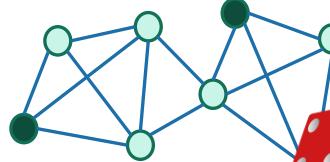
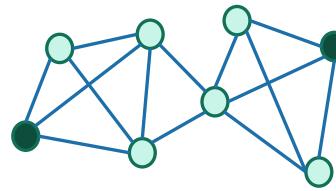
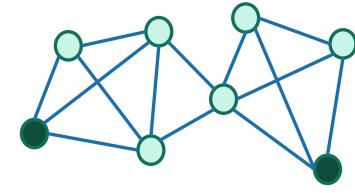
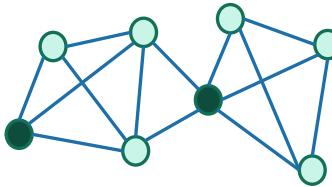
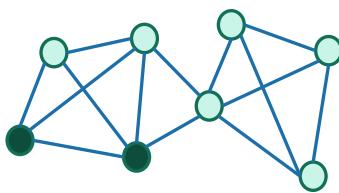
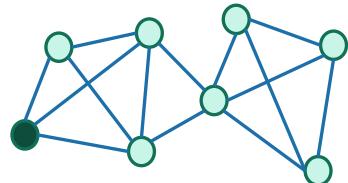
# Uniformly random cut

- Pick a random way to split the vertices into two parts:



# Uniformly random cut

- Pick a random way to split the vertices into two parts:



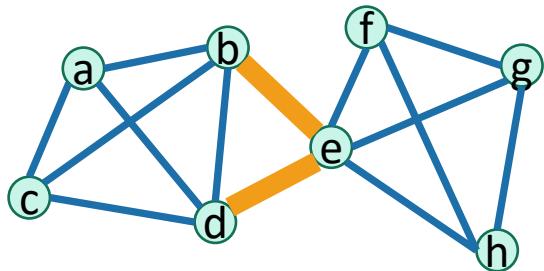
# Uniformly random cut

- Pick a random way to split the vertices into two parts:
- The probability of choosing the minimum cut is\*...

$$\frac{\text{number of min cuts in that graph}}{\text{number of ways to split 8 vertices in 2 parts}} = \frac{2}{2^8 - 2} \approx 0.008$$

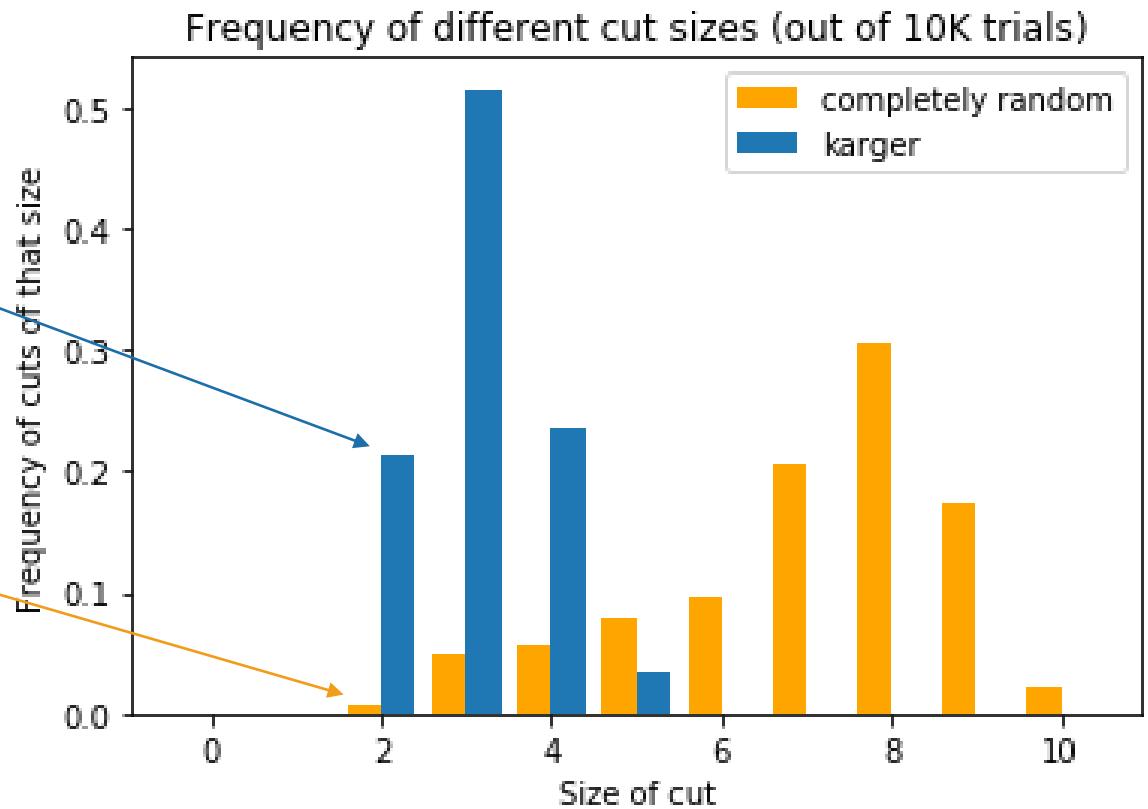
- Aka, we get a minimum cut **0.8% of the time.**

# Karger is better than completely random!



Karger's alg. is correct about 20% of the time

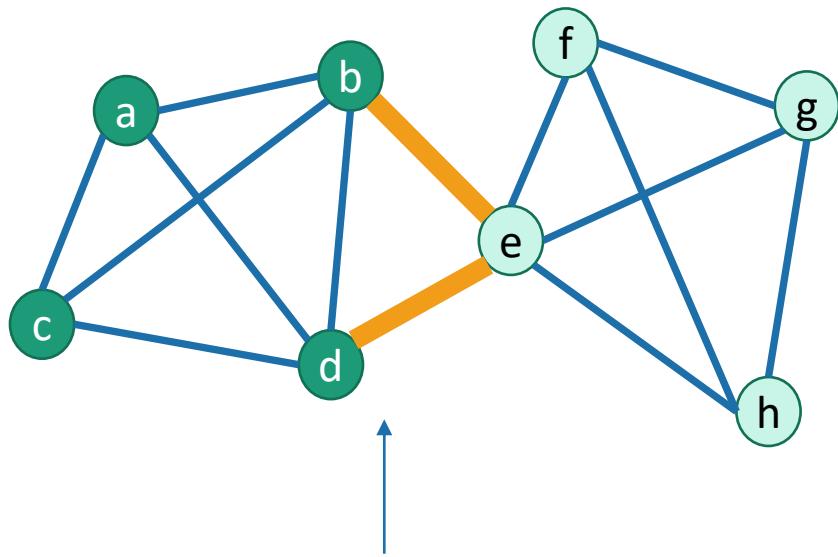
Completely random is correct about 0.8% of the time



What's going on?

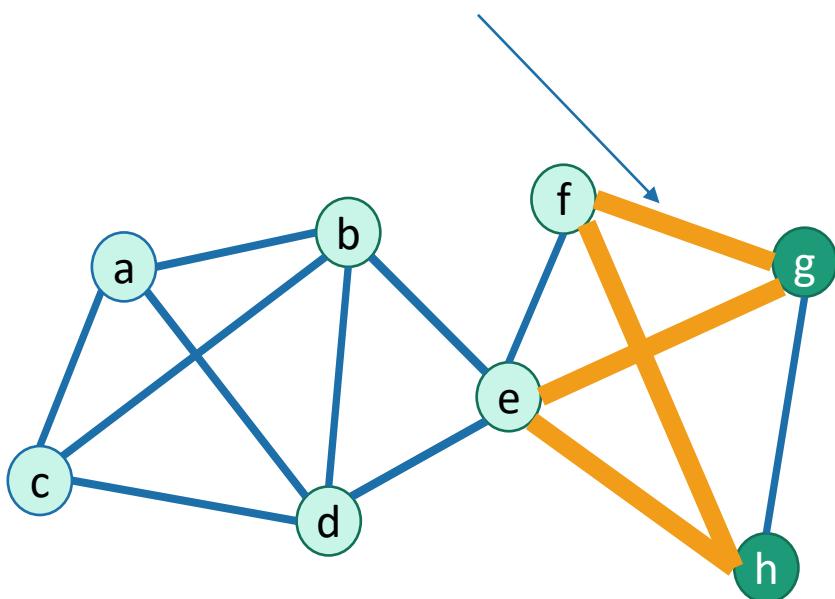
# What's going on?

- Which is more likely?



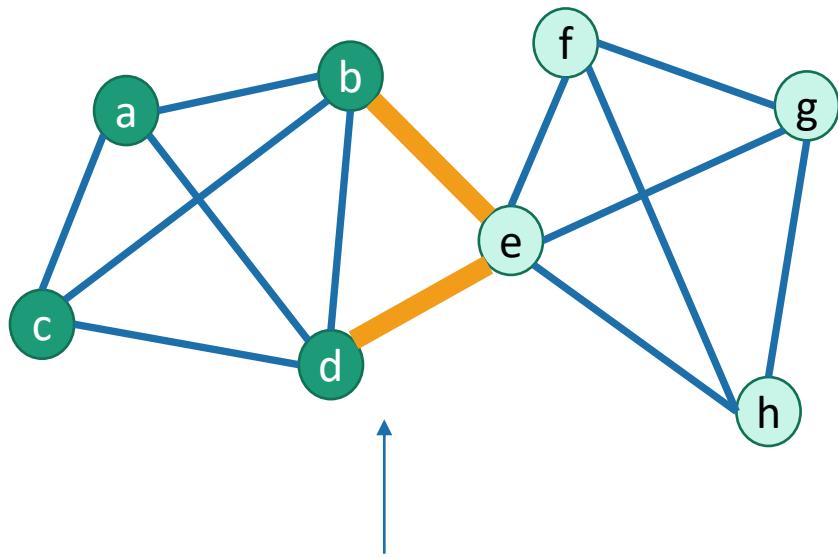
A: The algorithm never chooses either of the edges in **the minimum cut**.

B: The algorithm never chooses any of the edges in **this big cut**.



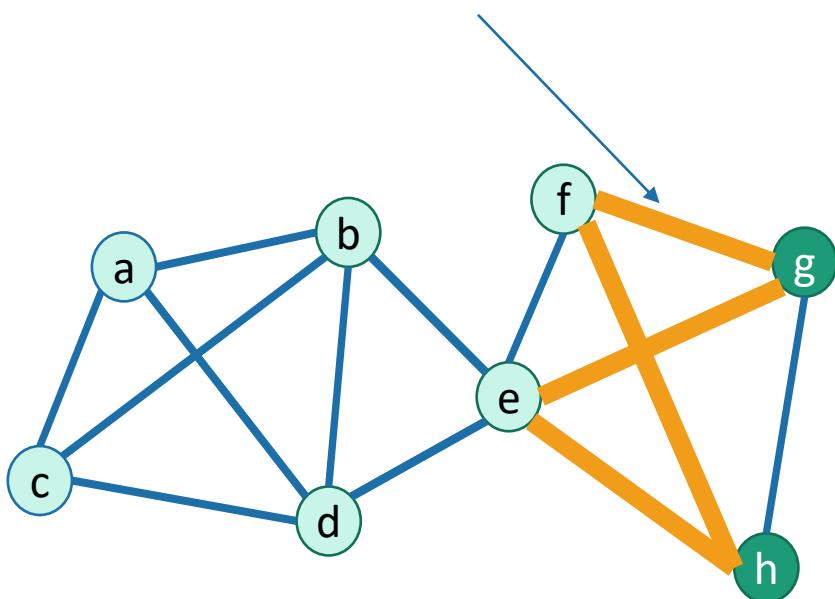
# What's going on?

- Which is more likely?



A: The algorithm never chooses either of the edges in **the minimum cut**.

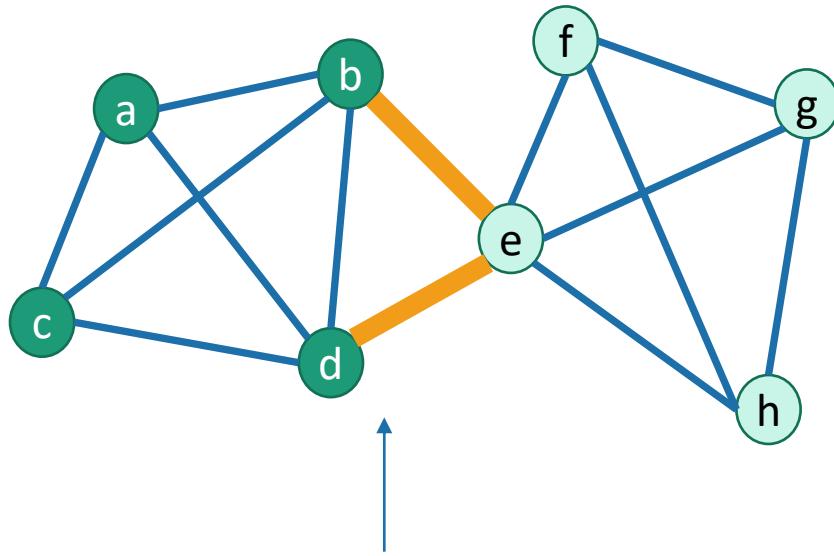
B: The algorithm never chooses any of the edges in **this big cut**.



- Neither A nor B are very likely, **but A is more likely than B.**

# What's going on?

- Which is more likely?

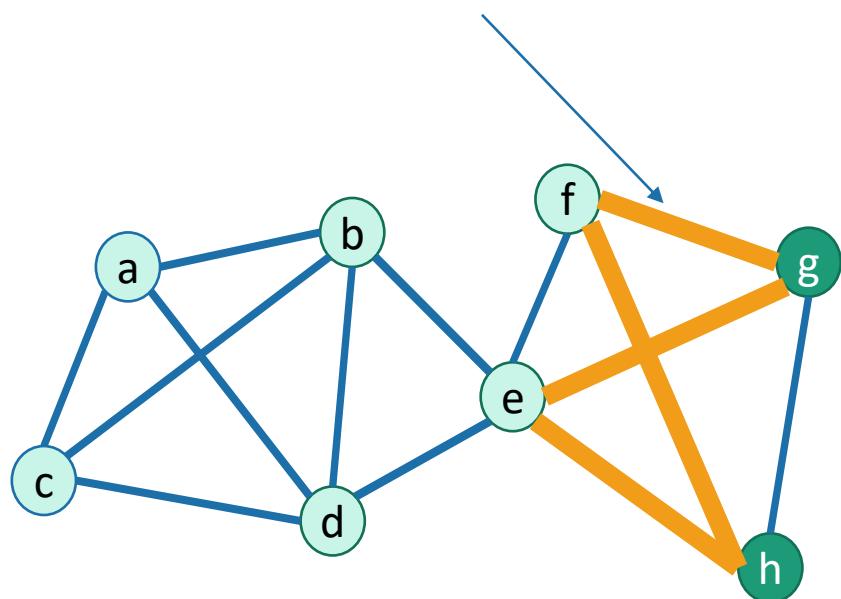


A: The algorithm never chooses either of the edges in **the minimum cut**.

Thing 1: It's unlikely that Karger will hit the min cut since it's so small!

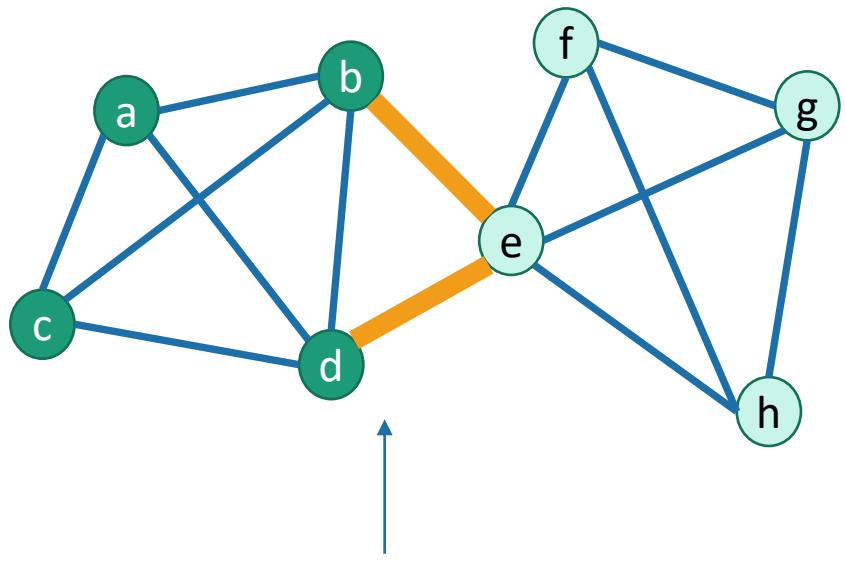


B: The algorithm never chooses any of the edges in **this big cut**.



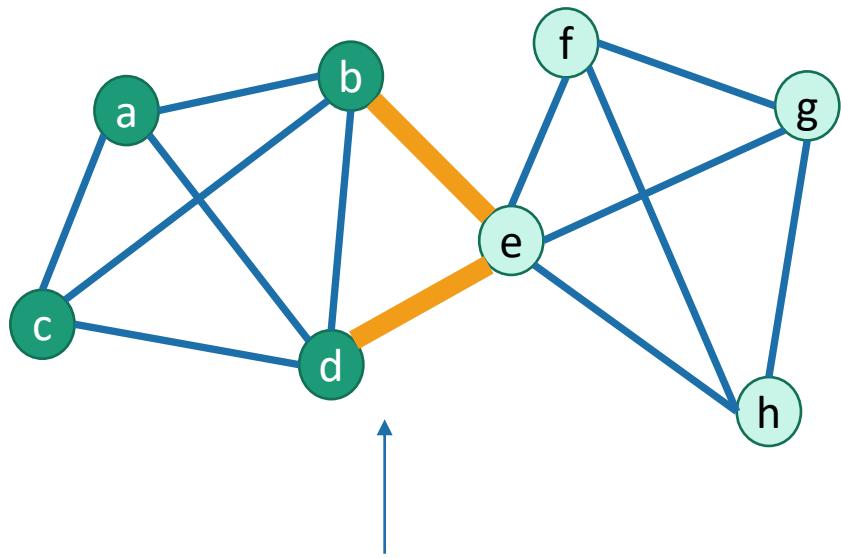
- Neither A nor B are very likely, **but A is more likely than B**.

# What's going on?



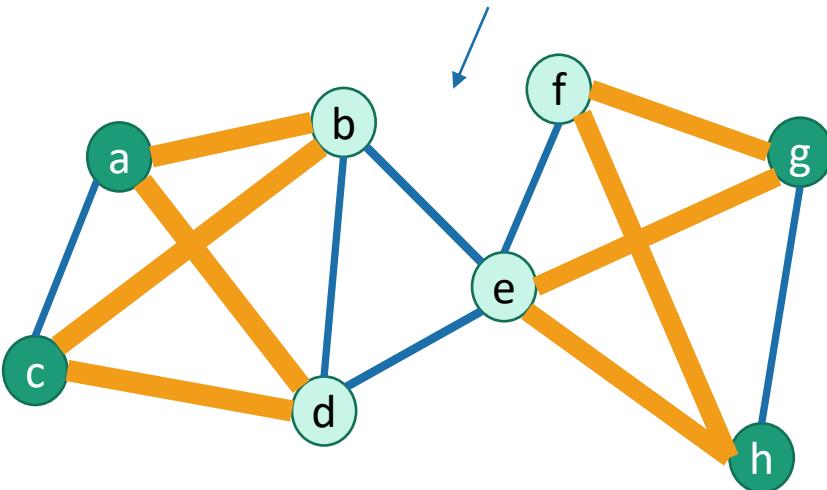
A: This cut can be returned  
by Karger's algorithm.

# What's going on?

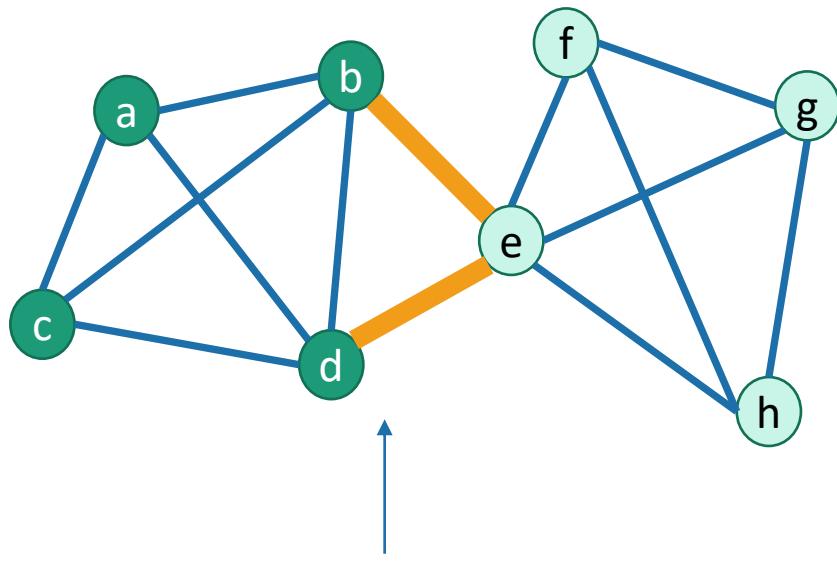


A: This cut can be returned  
by Karger's algorithm.

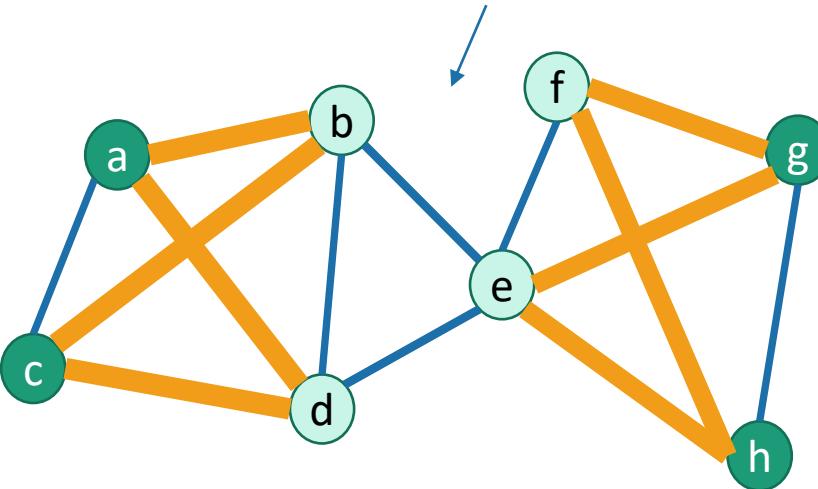
B: This cut can't be returned by  
Karger's algorithm!  
(Because how would a and g end up  
in the same super-node?)



# What's going on?

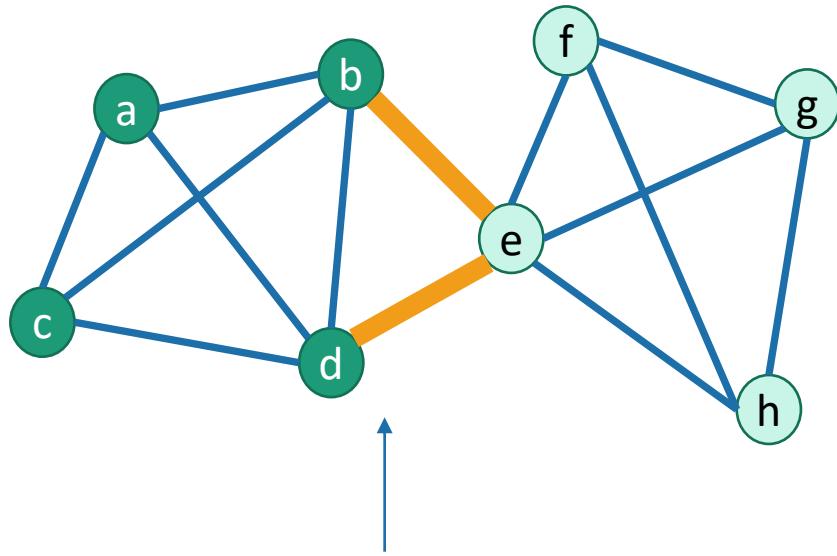


B: This cut can't be returned by Karger's algorithm!  
(Because how would a and g end up in the same super-node?)



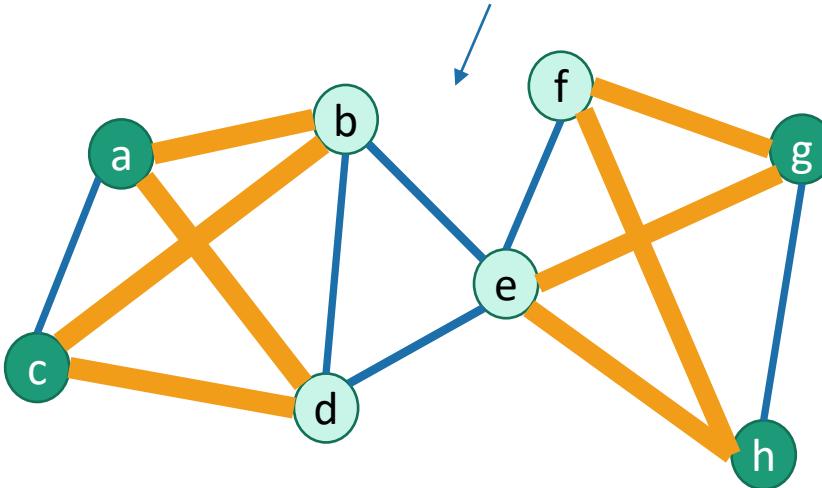
# What's going on?

Thing 2: By only contracting edges we are ignoring certain really-not-minimal cuts.



A: This cut can be returned by Karger's algorithm.

B: This cut can't be returned by Karger's algorithm!  
(Because how would a and g end up in the same super-node?)



This cut actually separates the graph into three pieces, so it's not minimal – either half of it is a smaller cut.

# Why does that help?

- Okay, so it's better than completely random...
- We're still wrong about 80% of the time.

The plan:

- 
- See that 20% chance of correctness is actually nontrivial.
  - Use repetition to boost an algorithm that's correct 20% of the time to an algorithm that's correct 99% of the time.

# Why does that help?

- Okay, so it's better than completely random...
- We're still wrong about 80% of the time.
- The main idea: **repeat!**
  - If I'm wrong 20% of the time, then if I repeat it a few times I'll eventually get it right.

The plan:

- 
- See that 20% chance of correctness is actually nontrivial.
  - Use repetition to boost an algorithm that's correct 20% of the time to an algorithm that's correct 99% of the time.

# Thought experiment

- Suppose you have a magic button that produces one of 5 numbers,  $\{a,b,c,d,e\}$ , uniformly at random when you push it.
- You don't know what  $\{a,b,c,d,e\}$  are.
- Q: What is the minimum of  $a,b,c,d,e$ ?



# Thought experiment

- Suppose you have a magic button that produces one of 5 numbers,  $\{a,b,c,d,e\}$ , uniformly at random when you push it.
- You don't know what  $\{a,b,c,d,e\}$  are.
- Q: What is the minimum of  $a,b,c,d,e$ ?



# Thought experiment

- Suppose you have a magic button that produces one of 5 numbers,  $\{a,b,c,d,e\}$ , uniformly at random when you push it.
- You don't know what  $\{a,b,c,d,e\}$  are.
- Q: What is the minimum of  $a,b,c,d,e$ ?



# Thought experiment

- Suppose you have a magic button that produces one of 5 numbers,  $\{a,b,c,d,e\}$ , uniformly at random when you push it.
- You don't know what  $\{a,b,c,d,e\}$  are.
- Q: What is the minimum of  $a,b,c,d,e$ ?



# Thought experiment

- Suppose you have a magic button that produces one of 5 numbers,  $\{a,b,c,d,e\}$ , uniformly at random when you push it.
- You don't know what  $\{a,b,c,d,e\}$  are.
- Q: What is the minimum of  $a,b,c,d,e$ ?



# Thought experiment

- Suppose you have a magic button that produces one of 5 numbers,  $\{a,b,c,d,e\}$ , uniformly at random when you push it.
- You don't know what  $\{a,b,c,d,e\}$  are.
- Q: What is the minimum of  $a,b,c,d,e$ ?



# Thought experiment

- Suppose you have a magic button that produces one of 5 numbers,  $\{a,b,c,d,e\}$ , uniformly at random when you push it.
- You don't know what  $\{a,b,c,d,e\}$  are.
- Q: What is the minimum of  $a,b,c,d,e$ ?



4      3      3      2      5      2

# Thought experiment

- Suppose you have a magic button that produces one of 5 numbers,  $\{a,b,c,d,e\}$ , uniformly at random when you push it.
- You don't know what  $\{a,b,c,d,e\}$  are.
- Q: What is the minimum of  $a,b,c,d,e$ ?



# Thought experiment

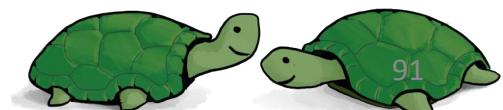
- Suppose you have a magic button that produces one of 5 numbers,  $\{a,b,c,d,e\}$ , uniformly at random when you push it.
- You don't know what  $\{a,b,c,d,e\}$  are.
- Q: What is the minimum of  $a,b,c,d,e$ ?



4            3            3            2            5            5  
3            2

How many times do you have to push the button, in expectation, before you see the minimum value?

What is the probability that you have to push it more than 5 times? 10 times?



We push the button  
once and don't ever get the min

$$\Pr[(1 - 0.2) = 0.8]$$

We push the button  
once and don't ever get the min

$$\Pr[(1 - 0.2) = 0.8]$$

Number of times  
we push the button until we get the minimum value

$$E[1/(0.20)] = 5$$

We push the button  
once and don't ever get the min

$$\Pr[(1 - 0.2) = 0.8]$$

Number of times  
we push the button until we get the minimum value

$$E[1/(0.20) = 5]$$

We push the button t times and don't ever get the min

$$\Pr[(1 - 0.2)^t]$$

We push the button  
once and don't ever  
get the min

$$\Pr[\text{we push the button once and don't ever get the min}] = (1 - 0.2) = 0.8$$

Number of times  
we push the button until we get the minimum value

$$E[\text{Number of times we push the button until we get the minimum value}] = 1/(0.20) = 5$$

We push the button t times and don't ever get the min

$$\Pr[\text{We push the button } t \text{ times and don't ever get the min}] = (1 - 0.2)^t$$

We push the button 5 times and don't ever get the min

$$\Pr[\text{We push the button 5 times and don't ever get the min}] = (1 - 0.2)^5 \approx 0.33$$

We push the button  
once and don't ever  
get the min

$$\Pr[\text{we push the button once and don't ever get the min}] = (1 - 0.2) = 0.8$$

Number of times  
we push the button until we get the minimum value

$$E[\text{Number of times we push the button until we get the minimum value}] = 1/(0.20) = 5$$

We push the button t times and don't ever get the min

$$\Pr[\text{We push the button } t \text{ times and don't ever get the min}] = (1 - 0.2)^t$$

We push the button 5 times and don't ever get the min

$$\Pr[\text{We push the button 5 times and don't ever get the min}] = (1 - 0.2)^5 \approx 0.33$$

We push the button 10 times and don't ever get the min

$$\Pr[\text{We push the button 10 times and don't ever get the min}] = (1 - 0.2)^{10} \approx 0.1$$

In this context

(Lets Run Karger's!, 5 times)

# In this context

(Lets Run Karger's!, 5 times)



- Run Karger's! The cut size is 4!

# In this context

(Lets Run Karger's!, 5 times)



- Run Karger's! The cut size is 4!
- Run Karger's! The cut size is 3!

# In this context

(Lets Run Karger's!, 5 times)



- Run Karger's! The cut size is 4!
- Run Karger's! The cut size is 3!
- Run Karger's! The cut size is 3!



# In this context

(Lets Run Karger's!, 5 times)



- Run Karger's! The cut size is 4!



- Run Karger's! The cut size is 3!



- Run Karger's! The cut size is 3!



- Run Karger's! The cut size is 2!

# In this context

(Lets Run Karger's!, 5 times)



- Run Karger's! The cut size is 4!



- Run Karger's! The cut size is 3!



- Run Karger's! The cut size is 3!



- Run Karger's! The cut size is 2!



- Run Karger's! The cut size is 5!

# In this context

(Lets Run Karger's!, 5 times)



- Run Karger's! The cut size is 4!



- Run Karger's! The cut size is 3!



- Run Karger's! The cut size is 3!



- Run Karger's! The cut size is 2!



Correct!



- Run Karger's! The cut size is 5!

# In this context

(Lets Run Karger's!, 5 times)



- Run Karger's! The cut size is 4!



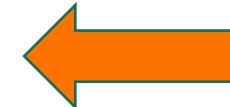
- Run Karger's! The cut size is 3!



- Run Karger's! The cut size is 3!



- Run Karger's! The cut size is 2!



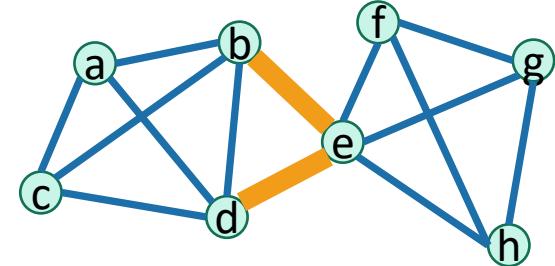
Correct!



- Run Karger's! The cut size is 5!

If the success probability is about 20%, then if you run Karger's algorithm 5 times and take the best answer you get, that will likely be correct! (with probability about 0.66)

# For this particular graph

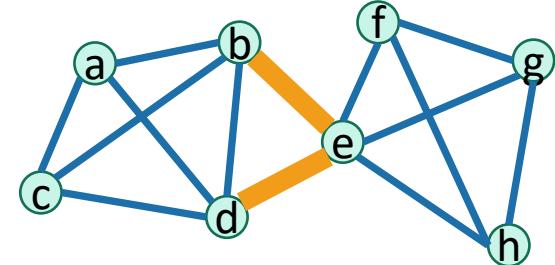


- Repeat Karger's algorithm about 5 times, and we will get a min cut with decent probability.
  - In contrast, we'd have to choose a random cut about  $1/0.008 = 125$  times!

The plan:

- See that 20% chance of correctness is actually nontrivial.
- Use repetition to boost an algorithm that's correct 20% of the time to an algorithm that's correct most of the time.

# For this particular graph



- Repeat Karger's algorithm about 5 times, and we will get a min cut with decent probability.
  - In contrast, we'd have to choose a random cut about  $1/0.008 = 125$  times!

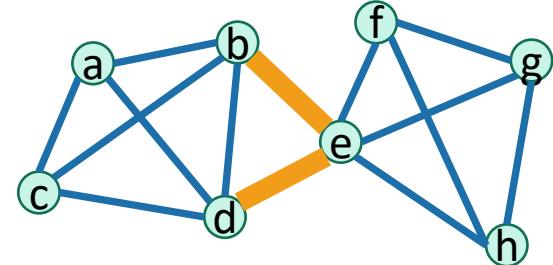
Hang on! This “20%” figure just came from running experiments on this particular graph. What about general graphs? Can we prove something?



The plan:

- See that 20% chance of correctness is actually nontrivial.
- Use repetition to boost an algorithm that's correct 20% of the time to an algorithm that's correct most of the time.

# For this particular graph



- Repeat Karger's algorithm about 5 times, and we will get a min cut with decent probability.
  - In contrast, we'd have to choose a random cut about  $1/0.008 = 125$  times!

Hang on! This “20%” figure just came from running experiments on this particular graph. What about general graphs? Can we prove something?



Also, we should be a bit more precise about this “about 5 times” statement.

The plan:

- See that 20% chance of correctness is actually nontrivial.
- Use repetition to boost an algorithm that's correct 20% of the time to an algorithm that's correct most of the time.

# Questions



To generalize this approach to all graphs

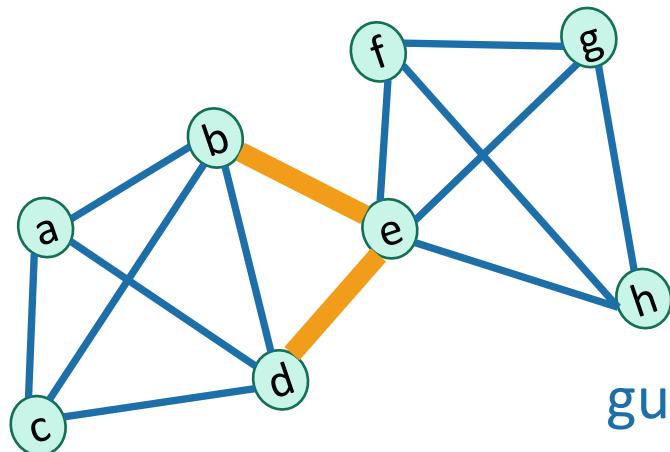
1. What is the probability that Karger's algorithm returns a minimum cut?
2. How many times should we run Karger's algorithm to “probably” succeed?
  - Say, with probability 0.99?
  - Or more generally, probability  $1 - \delta$  ?

# Answer to Question 1

**Claim:**

The probability that Karger's algorithm returns a minimum cut is

at least  $\frac{1}{\binom{n}{2}}$



In this case,  $\frac{1}{\binom{8}{2}} = 0.036$ , so we are guaranteed to win at least 3.6% of the time.

# Questions



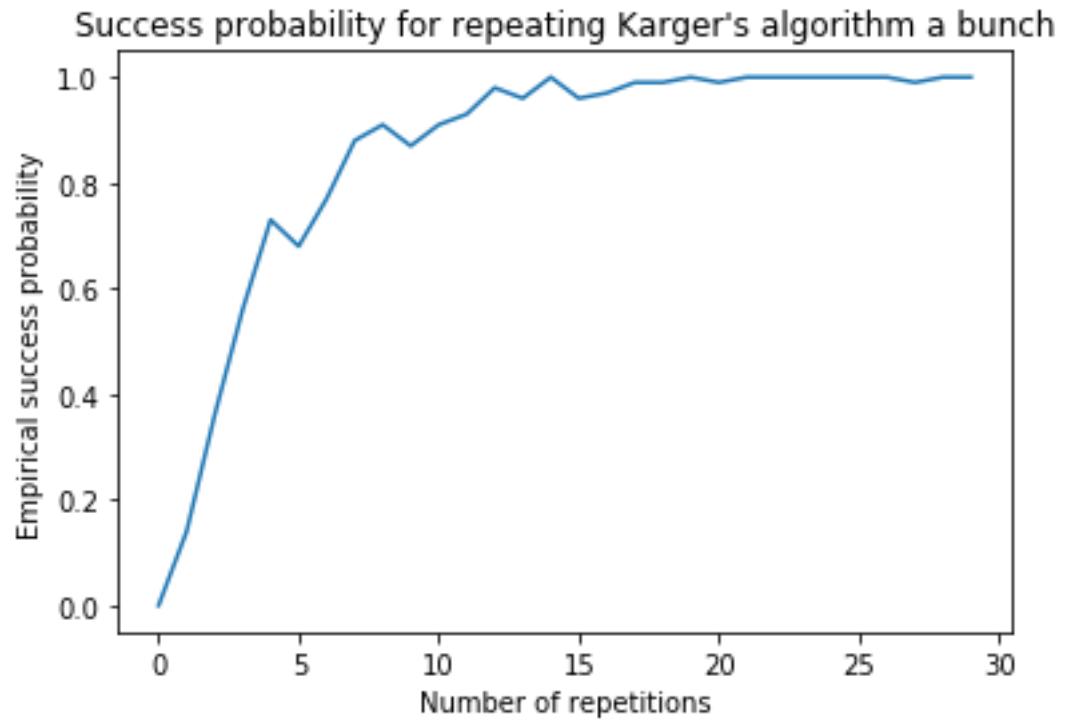
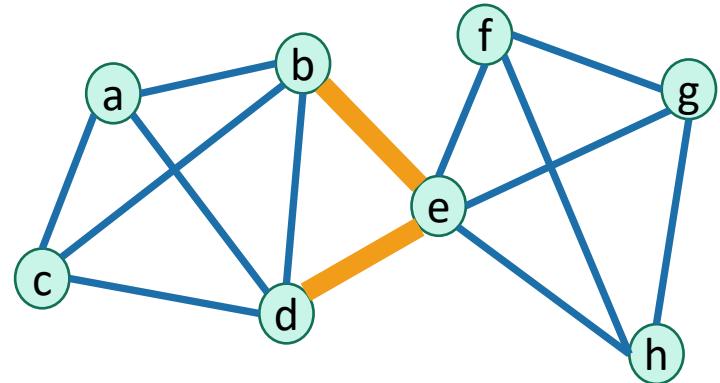
1. What is the probability that Karger's algorithm returns a minimum cut?

According to the claim, at least  $\frac{1}{\binom{n}{2}}$

2. How many times should we run Karger's algorithm to "probably" succeed?
  - Say, with probability 0.99?
  - Or more generally, probability  $1 - \delta$  ?

# Before we prove the Claim

2. How many times should we run Karger's algorithm to succeed with probability  $1 - \delta$  ?



# A computation

- Suppose :
  - the probability of successfully returning a minimum cut is  $p \in [0, 1]$ ,
  - we want failure probability at most  $\delta \in (0,1)$ .

# A computation

- Suppose :
  - the probability of successfully returning a minimum cut is  $p \in [0, 1]$ ,
  - we want failure probability at most  $\delta \in (0, 1)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}] = (1 - p)^T$

# A computation

- Suppose :
  - the probability of successfully returning a minimum cut is  $p \in [0, 1]$ ,
  - we want failure probability at most  $\delta \in (0, 1)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}] = (1 - p)^T$
- So  $p = 1/\binom{n}{2}$  by the Claim. Let's choose  $T = \binom{n}{2} \ln(1/\delta)$ .

# A computation

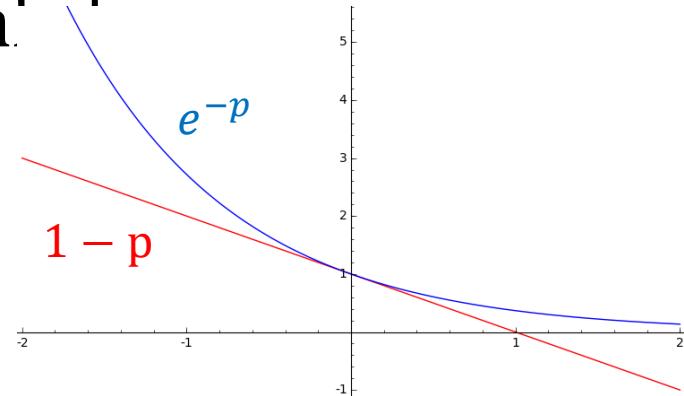
- Suppose :
  - the probability of successfully returning a minimum cut is  $p \in [0, 1]$ ,
  - we want failure probability at most  $\delta \in (0, 1)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}] = (1 - p)^T$
- So  $p = 1/\binom{n}{2}$  by the Claim. Let's choose  $T = \binom{n}{2} \ln(1/\delta)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}] = (1 - p)^T$

# A computation

- Suppose :
  - the probability of successfully returning a minimum cut is  $p \in [0, 1]$ ,
  - we want failure probability at most  $\delta \in (0, 1)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}] = (1 - p)^T$
- So  $p = 1/\binom{n}{2}$  by the Claim. Let's choose  $T = \binom{n}{2} \ln(1/\delta)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}]$   
 $= (1 - p)^T$   
 $\leq (e^{-p})^T$

# A computation

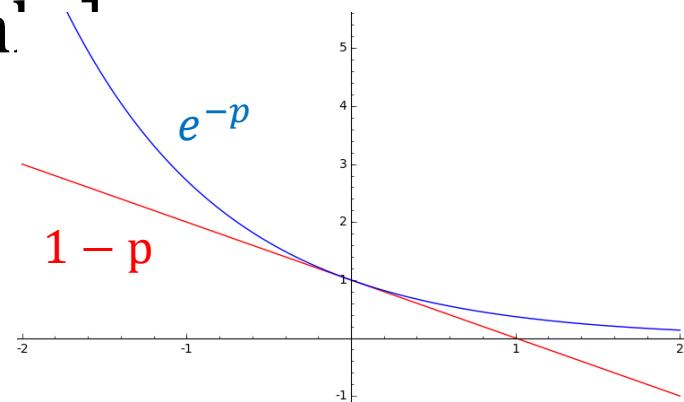
- Suppose :
  - the probability of successfully returning a minimum cut is  $p \in [0, 1]$ ,
  - we want failure probability at most  $\delta \in (0,1)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}] = (1 - p)^T$
- So  $p = 1/\binom{n}{2}$  by the Claim. Let's choose  $T = \binom{n}{2} \ln(1/\delta)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}]$   
 $= (1 - p)^T$   
 $\leq (e^{-p})^T$



$$1 - p \leq e^{-p}$$

# A computation

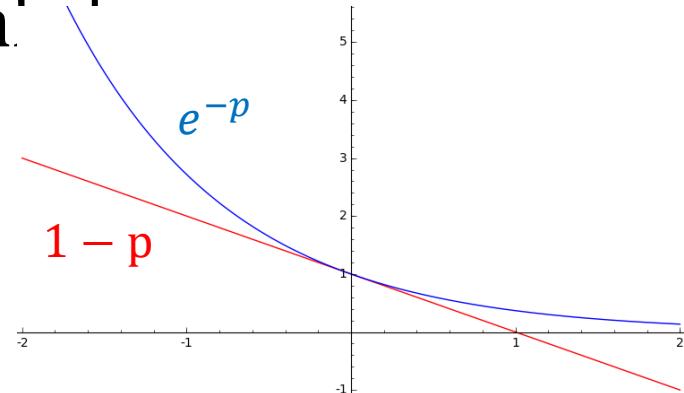
- Suppose :
  - the probability of successfully returning a minimum cut is  $p \in [0, 1]$ ,
  - we want failure probability at most  $\delta \in (0,1)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}] = (1 - p)^T$
- So  $p = 1/\binom{n}{2}$  by the Claim. Let's choose  $T = \binom{n}{2} \ln(1/\delta)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}]$   
 $= (1 - p)^T$   
 $\leq (e^{-p})^T$   
 $= e^{-pT}$



$$1 - p \leq e^{-p}$$

# A computation

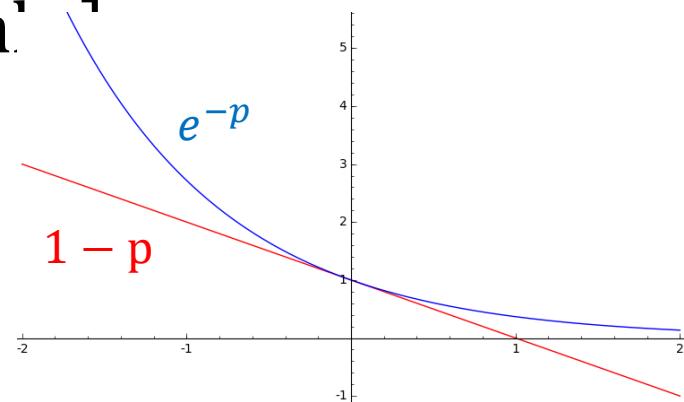
- Suppose :
  - the probability of successfully returning a minimum cut is  $p \in [0, 1]$ ,
  - we want failure probability at most  $\delta \in (0,1)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}] = (1 - p)^T$
- So  $p = 1/\binom{n}{2}$  by the Claim. Let's choose  $T = \binom{n}{2} \ln(1/\delta)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}]$   
 $= (1 - p)^T$   
 $\leq (e^{-p})^T$   
 $= e^{-pT}$   
 $= e^{-\ln(\frac{1}{\delta})}$



$$1 - p \leq e^{-p}$$

# A computation

- Suppose :
  - the probability of successfully returning a minimum cut is  $p \in [0, 1]$ ,
  - we want failure probability at most  $\delta \in (0,1)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}] = (1 - p)^T$
- So  $p = 1/\binom{n}{2}$  by the Claim. Let's choose  $T = \binom{n}{2} \ln(1/\delta)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}]$   
 $= (1 - p)^T$   
 $\leq (e^{-p})^T$   
 $= e^{-pT}$   
 $= e^{-\ln(\frac{1}{\delta})}$   
 $= \delta$



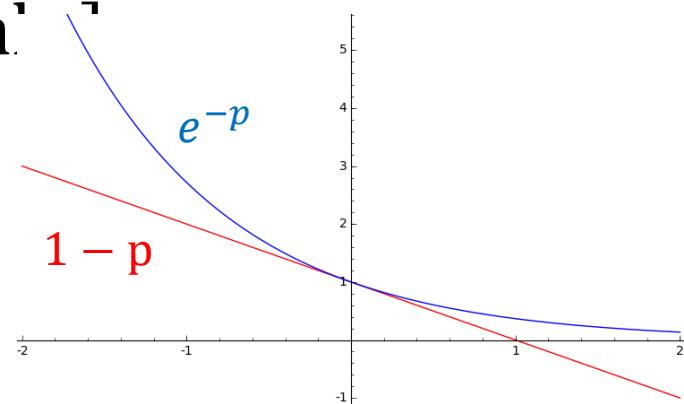
$$1 - p \leq e^{-p}$$

# A computation

Punchline:

If we repeat  $T = \binom{n}{2} \ln(1/\delta)$  times,  
we win with probability at least  $1 - \delta$ .

- Suppose :
  - the probability of successfully returning a minimum cut is  $p \in [0, 1]$ ,
  - we want failure probability at most  $\delta \in (0,1)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}] = (1 - p)^T$
- So  $p = 1/\binom{n}{2}$  by the Claim. Let's choose  $T = \binom{n}{2} \ln(1/\delta)$ .
- $\Pr[\text{don't return a min cut in } T \text{ trials}]$   
 $= (1 - p)^T$   
 $\leq (e^{-p})^T$   
 $= e^{-pT}$   
 $= e^{-\ln(\frac{1}{\delta})}$   
 $= \delta$



$$1 - p \leq e^{-p}$$

# Answers



1. What is the probability that Karger's algorithm returns a minimum cut?

According to the claim, at least  $\frac{1}{\binom{n}{2}}$

2. How many times should we run Karger's algorithm to "probably" succeed?
  - Say, with probability 0.99?
  - Or more generally, probability  $1 - \delta$  ?

$\binom{n}{2} \ln \left( \frac{1}{\delta} \right)$  times.

# Theorem

Assuming the claim about  $1/\binom{n}{2} \dots$

- Suppose  $G$  has  $n$  vertices.
  - Consider the following algorithm:
    - `bestCut = None`
    - **for**  $t = 1, \dots, \binom{n}{2} \ln\left(\frac{1}{\delta}\right)$  :
      - `candidateCut ← Karger(G)`
      - **if** `candidateCut` is smaller than `bestCut`:
        - `bestCut ← candidateCut`
  - **return** `bestCut`
- Then  $\Pr[\text{this doesn't return a min cut}] \leq \delta$ .

# What's the running time?

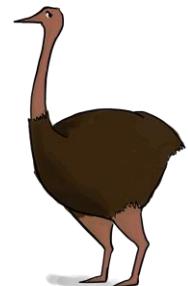
# What's the running time?

- $\binom{n}{2} \ln\left(\frac{1}{\delta}\right)$  repetitions, and  $O(n^2)$  per repetition.
- So,  $O\left(n^2 \cdot \binom{n}{2} \ln\left(\frac{1}{\delta}\right)\right) = O(n^4)$  Treating  $\delta$  as constant.

# What's the running time?

- $\binom{n}{2} \ln\left(\frac{1}{\delta}\right)$  repetitions, and  $O(n^2)$  per repetition.
- So,  $O\left(n^2 \cdot \binom{n}{2} \ln\left(\frac{1}{\delta}\right)\right) = O(n^4)$  Treating  $\delta$  as constant.

Again we can do better with a union-find data structure.



# Theorem

Assuming the claim about  $1/\binom{n}{2} \dots$

Suppose  $G$  has  $n$  vertices. Then [repeating Karger's algorithm] finds a min cut in  $G$  with probability at least 0.99 in time  $O(n^4)$ .

Now let's prove the claim...

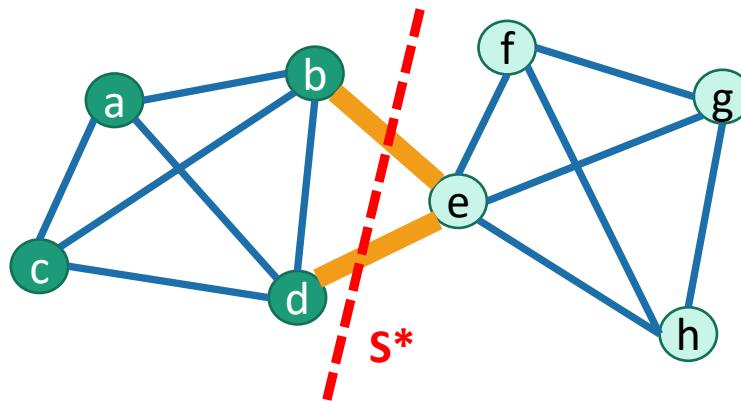
# Claim

The probability that Karger's algorithm returns a minimum cut is

at least  $\frac{1}{{n \choose 2}}$

# Now let's prove that claim

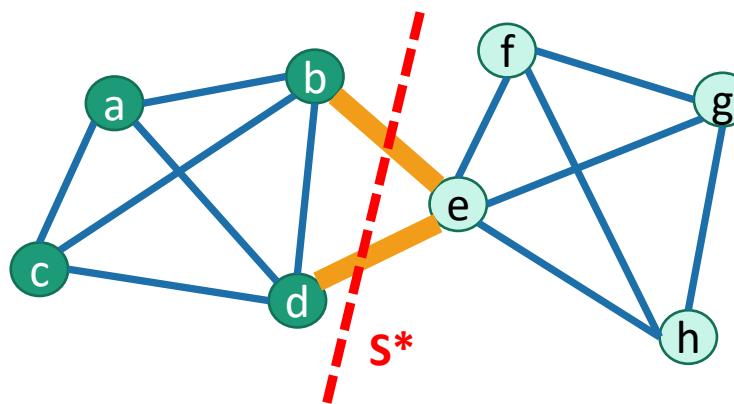
Say that  $S^*$  is a minimum cut.



# Now let's prove that claim

Say that  $S^*$  is a minimum cut.

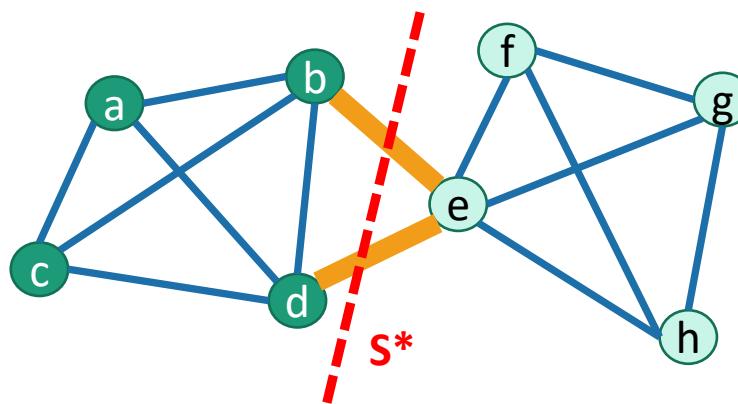
- Suppose the edges that we choose are  $e_1, e_2, \dots, e_{n-2}$
- **PR[ return  $S^*$  ] = PR[ none of the  $e_i$  cross  $S^*$  ]**



# Now let's prove that claim

Say that  $S^*$  is a minimum cut.

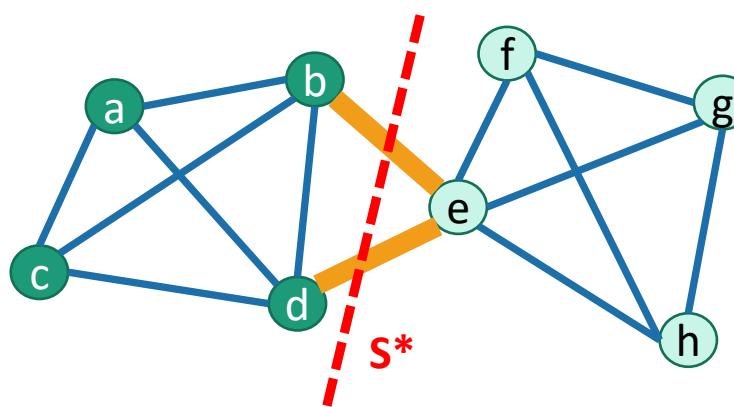
- Suppose the edges that we choose are  $e_1, e_2, \dots, e_{n-2}$
- $\text{PR}[\text{return } S^*] = \text{PR}[\text{none of the } e_i \text{ cross } S^*]$   
 $= \text{PR}[e_1 \text{ doesn't cross } S^*]$



# Now let's prove that claim

Say that  $S^*$  is a minimum cut.

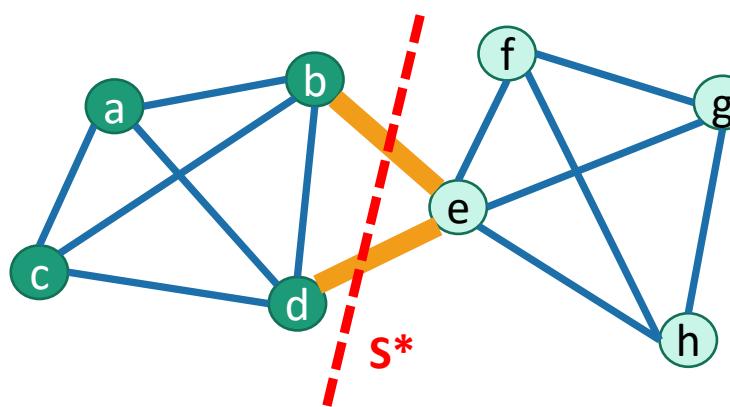
- Suppose the edges that we choose are  $e_1, e_2, \dots, e_{n-2}$
- $\text{PR}[\text{return } S^*] = \text{PR}[\text{none of the } e_i \text{ cross } S^*]$   
 $= \text{PR}[e_1 \text{ doesn't cross } S^*]$   
 $\times \text{PR}[e_2 \text{ doesn't cross } S^* \mid e_1 \text{ doesn't cross } S^*]$



# Now let's prove that claim

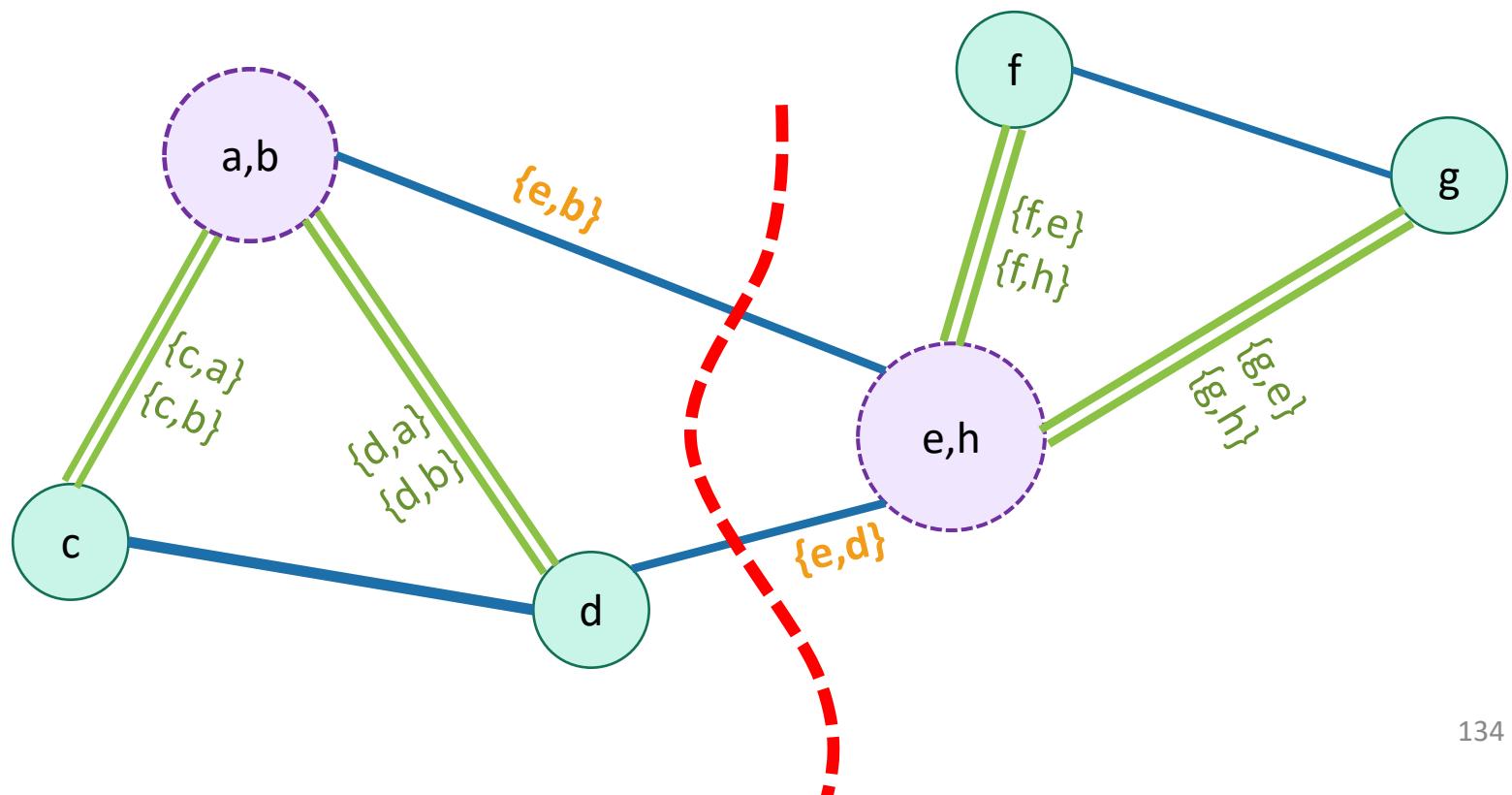
Say that  $S^*$  is a minimum cut.

- Suppose the edges that we choose are  $e_1, e_2, \dots, e_{n-2}$
- $\text{PR}[\text{return } S^*] = \text{PR}[\text{none of the } e_i \text{ cross } S^*]$   
 $= \text{PR}[e_1 \text{ doesn't cross } S^*]$   
 $\times \text{PR}[e_2 \text{ doesn't cross } S^* \mid e_1 \text{ doesn't cross } S^*]$   
 $\dots$   
 $\times \text{PR}[e_{n-2} \text{ doesn't cross } S^* \mid e_1, \dots, e_{n-3} \text{ don't cross } S^*]$



Focus in on:

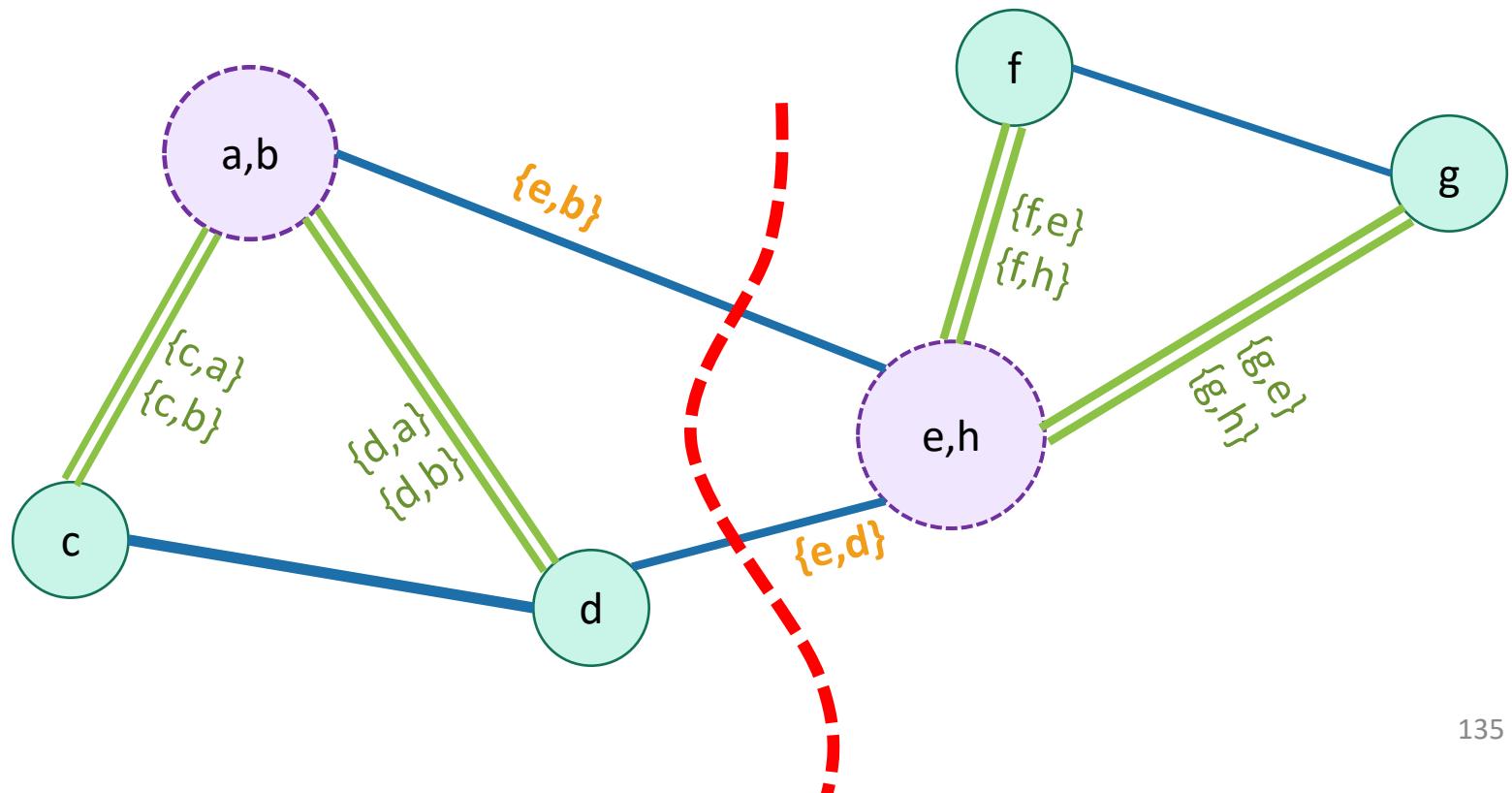
$\text{PR}[ e_j \text{ doesn't cross } S^* \mid e_1, \dots, e_{j-1} \text{ don't cross } S^* ]$



Focus in on:

$\text{PR}[ e_j \text{ doesn't cross } S^* \mid e_1, \dots, e_{j-1} \text{ don't cross } S^* ]$

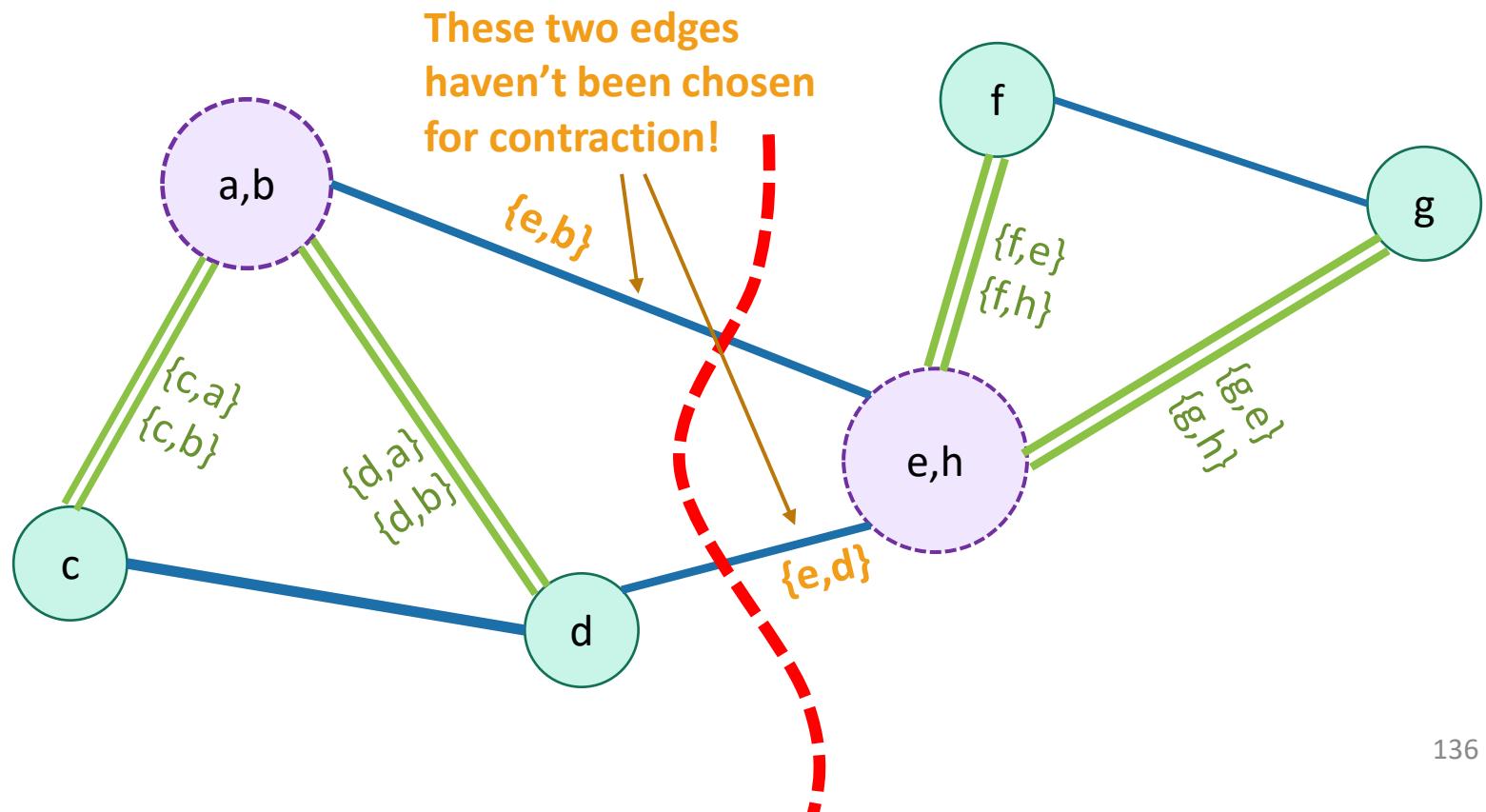
- Suppose: After  $j-1$  iterations, we haven't messed up yet!



Focus in on:

$\text{PR}[ e_j \text{ doesn't cross } S^* \mid e_1, \dots, e_{j-1} \text{ don't cross } S^* ]$

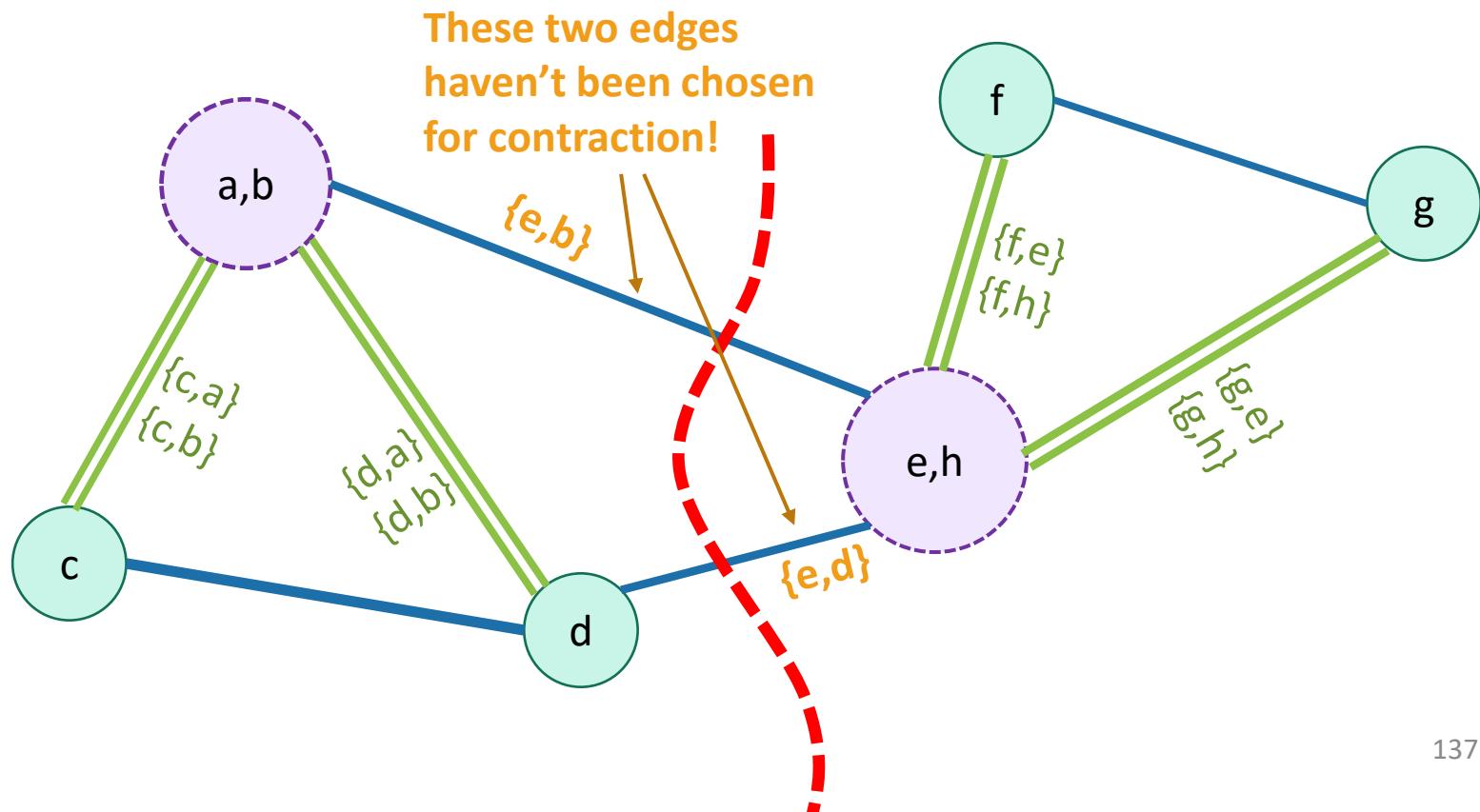
- Suppose: After  $j-1$  iterations, we haven't messed up yet!



Focus in on:

$\text{PR}[ e_j \text{ doesn't cross } S^* \mid e_1, \dots, e_{j-1} \text{ don't cross } S^* ]$

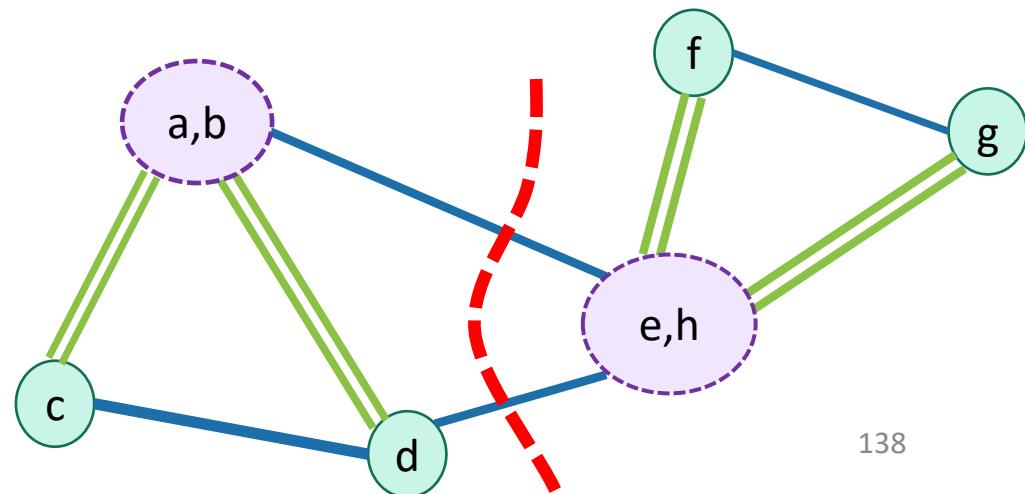
- Suppose: After  $j-1$  iterations, we haven't messed up yet!
- What's the probability of messing up now?



Focus in on:

$$\text{PR}[ e_j \text{ doesn't cross } S^* \mid e_1, \dots, e_{j-1} \text{ don't cross } S^* ]$$

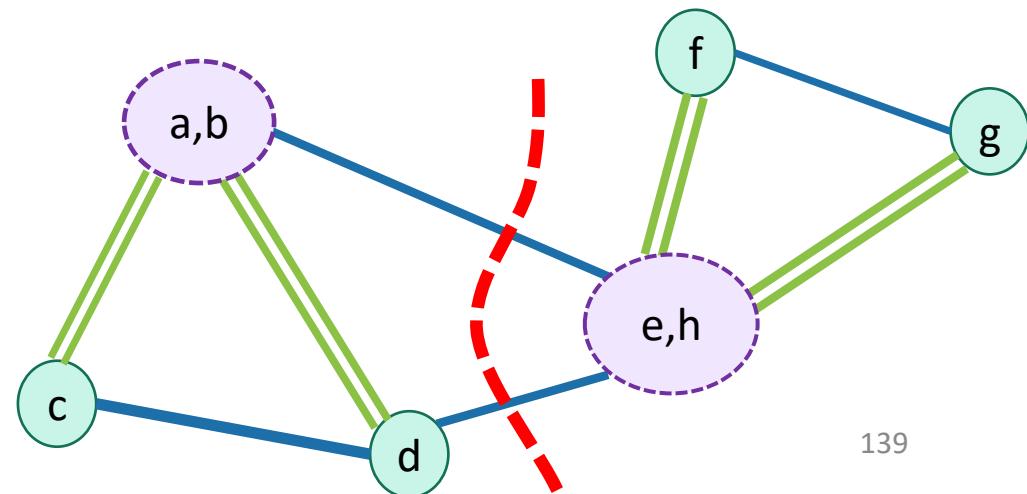
- Suppose: After  $j-1$  iterations, we haven't messed up yet!
- What's the probability of messing up now?



Focus in on:

$$\text{PR}[ e_j \text{ doesn't cross } S^* \mid e_1, \dots, e_{j-1} \text{ don't cross } S^* ]$$

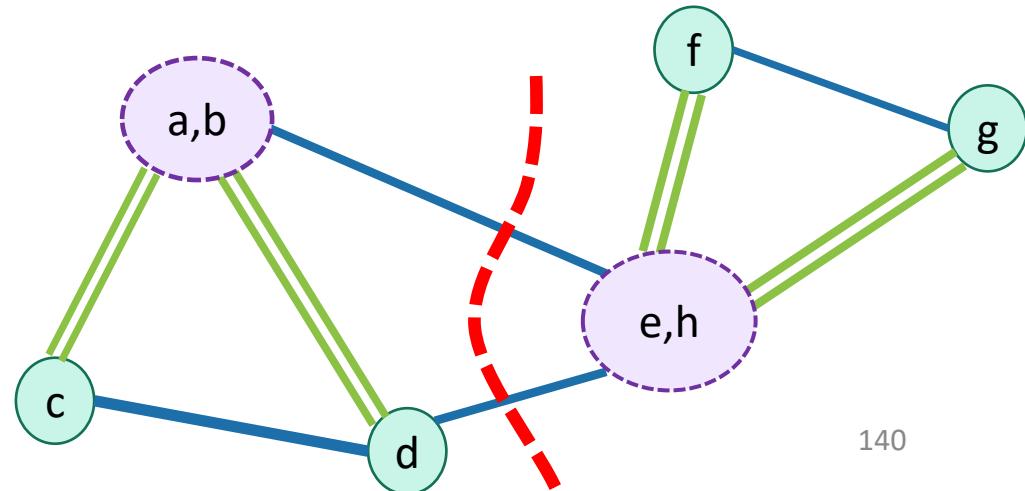
- Suppose: After  $j-1$  iterations, we haven't messed up yet!
- What's the probability of messing up now?
- Say there are **k edges that cross  $S^*$**
- Every supernode has at least  $k$  (original) edges coming out.
  - Otherwise we'd have a smaller cut.



Focus in on:

$$\text{PR}[ e_j \text{ doesn't cross } S^* \mid e_1, \dots, e_{j-1} \text{ don't cross } S^* ]$$

- Suppose: After  $j-1$  iterations, we haven't messed up yet!
- What's the probability of messing up now?
- Say there are **k edges that cross  $S^*$**
- Every supernode has at least  $k$  (original) edges coming out.
  - Otherwise we'd have a smaller cut.
- Thus, there are at least  **$(n-j+1)k/2$  edges total**.
  - because there are  $n - j + 1$  supernodes left, each with  $k$  edges.



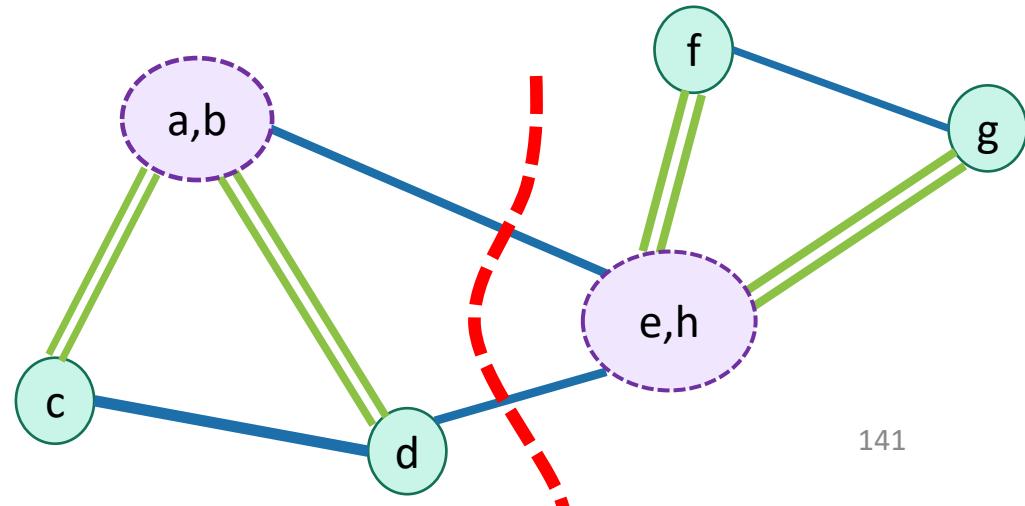
Focus in on:

$$\text{PR}[ e_j \text{ doesn't cross } S^* \mid e_1, \dots, e_{j-1} \text{ don't cross } S^* ]$$

- Suppose: After  $j-1$  iterations, we haven't messed up yet!
- What's the probability of messing up now?
- Say there are **k edges that cross  $S^*$**
- Every supernode has at least  $k$  (original) edges coming out.
  - Otherwise we'd have a smaller cut.
- Thus, there are at least  **$(n-j+1)k/2$  edges total**.
  - because there are  $n - j + 1$  supernodes left, each with  $k$  edges.

So the probability that we choose one of the  $k$  edges crossing  $S^*$  at step  $j$  is at most:

$$\frac{k}{\binom{(n-j+1)k}{2}} = \frac{2}{n-j+1}$$

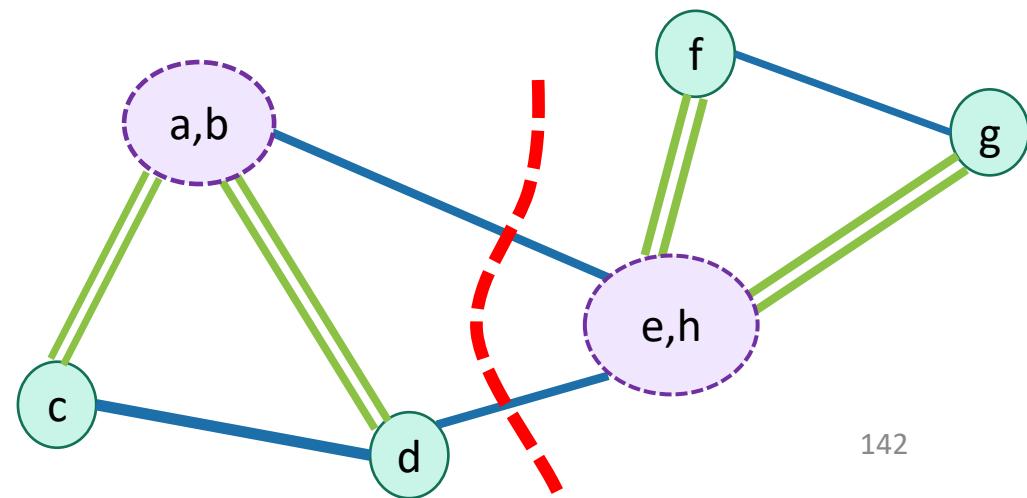


Focus in on:

$$\text{PR}[ e_j \text{ doesn't cross } S^* \mid e_1, \dots, e_{j-1} \text{ don't cross } S^* ]$$

- So the probability that we choose one of the  $k$  edges crossing  $S^*$  at step  $j$  is at most:

$$\frac{k}{\binom{(n-j+1)k}{2}} = \frac{2}{n-j+1}$$



Focus in on:

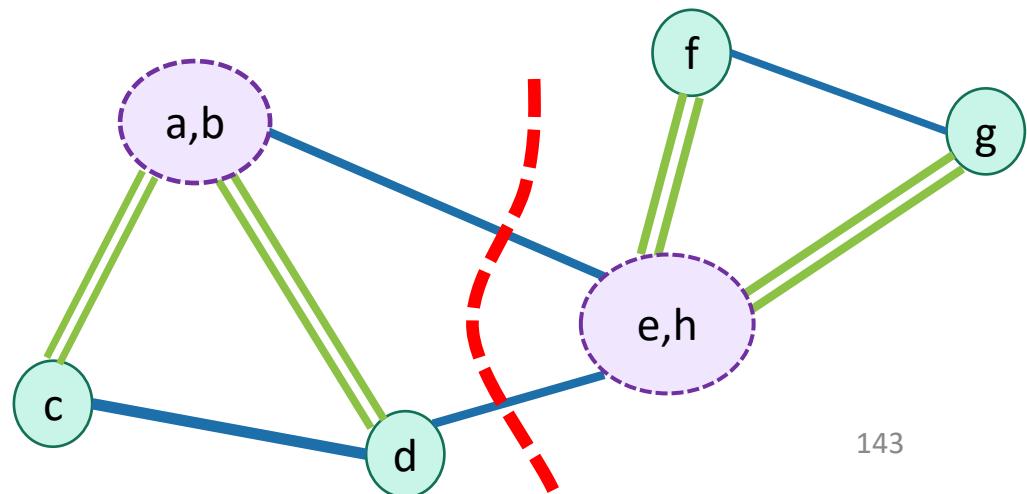
$$\text{PR}[ e_j \text{ doesn't cross } S^* \mid e_1, \dots, e_{j-1} \text{ don't cross } S^* ]$$

- So the probability that we choose one of the  $k$  edges crossing  $S^*$  at step  $j$  is at most:

$$\frac{k}{\binom{(n-j+1)k}{2}} = \frac{2}{n-j+1}$$

- The probability we **don't** choose one of the  $k$  edges is at least:

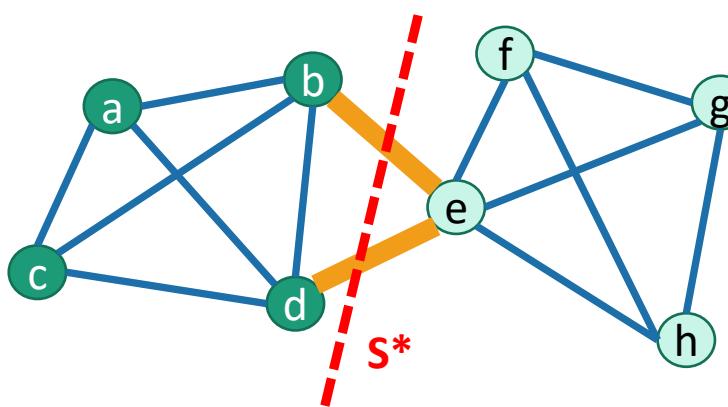
$$1 - \frac{2}{n-j+1} = \frac{n-j-1}{n-j+1}$$



# Now let's prove that claim

Say that  $S^*$  is a minimum cut.

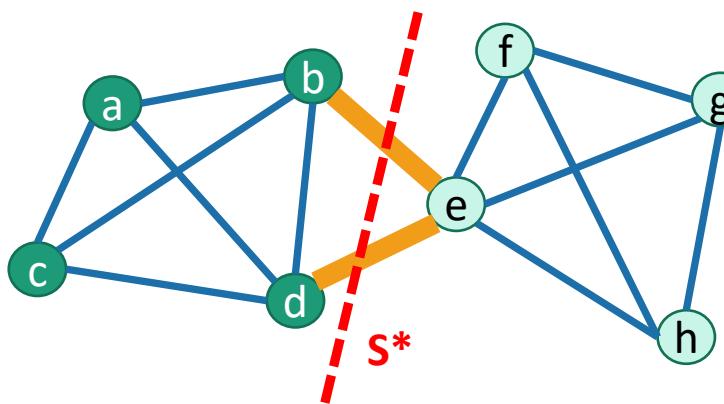
- Suppose the edges that we choose are  $e_1, e_2, \dots, e_{n-2}$
- $\text{PR}[\text{return } S^*] = \text{PR}[\text{none of the } e_i \text{ cross } S^*]$   
 $= \text{PR}[e_1 \text{ doesn't cross } S^*]$   
 $\times \text{PR}[e_2 \text{ doesn't cross } S^* \mid e_1 \text{ doesn't cross } S^*]$   
 $\dots$   
 $\times \text{PR}[e_{n-2} \text{ doesn't cross } S^* \mid e_1, \dots, e_{n-3} \text{ don't cross } S^*]$



# Now let's prove that claim

Say that  $S^*$  is a minimum cut.

- Suppose the edges that we choose are  $e_1, e_2, \dots, e_{n-2}$
- $\text{PR}[\text{return } S^*] = \text{PR}[\text{none of the } e_i \text{ cross } S^*]$   
 $= \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \left(\frac{n-4}{n-2}\right) \left(\frac{n-5}{n-3}\right) \left(\frac{n-6}{n-4}\right) \dots \left(\frac{4}{6}\right) \left(\frac{3}{5}\right) \left(\frac{2}{4}\right) \left(\frac{1}{3}\right)$

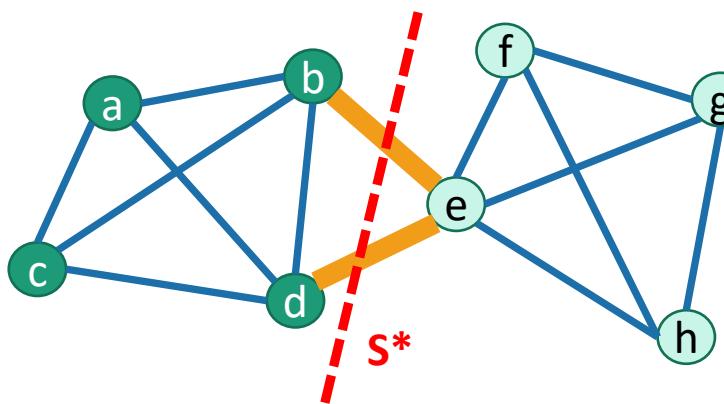


# Now let's prove that claim

Say that  $S^*$  is a minimum cut.

- Suppose the edges that we choose are  $e_1, e_2, \dots, e_{n-2}$
- $\text{PR}[\text{return } S^*] = \text{PR}[\text{none of the } e_i \text{ cross } S^*]$

$$= \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \left(\frac{n-4}{n-2}\right) \left(\frac{n-5}{n-3}\right) \left(\frac{n-6}{n-4}\right) \cdots \left(\frac{4}{6}\right) \left(\frac{3}{5}\right) \left(\frac{2}{4}\right) \left(\frac{1}{3}\right)$$

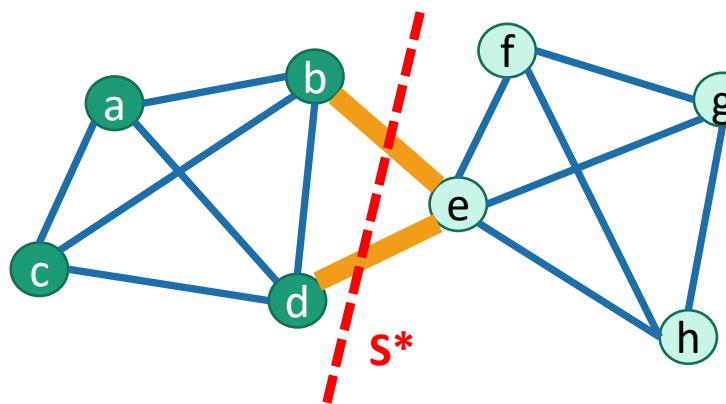


# Now let's prove that claim

Say that  $S^*$  is a minimum cut.

- Suppose the edges that we choose are  $e_1, e_2, \dots, e_{n-2}$
- $\text{PR}[\text{return } S^*] = \text{PR}[\text{none of the } e_i \text{ cross } S^*]$

$$= \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \left(\frac{n-4}{n-2}\right) \left(\frac{n-5}{n-3}\right) \left(\frac{n-6}{n-4}\right) \cdots \left(\frac{4}{6}\right) \left(\frac{3}{5}\right) \left(\frac{2}{4}\right) \left(\frac{1}{3}\right)$$

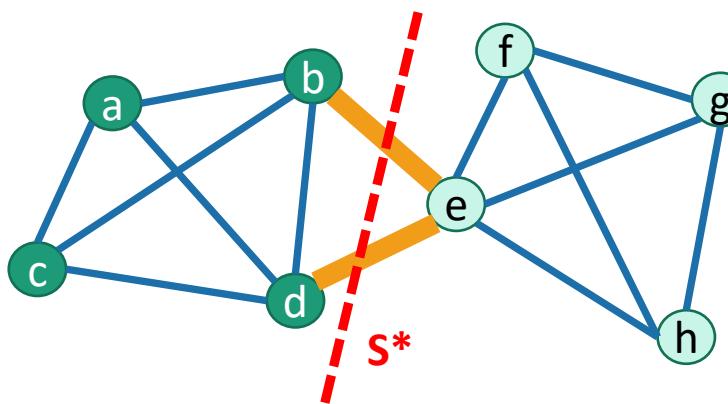


# Now let's prove that claim

Say that  $S^*$  is a minimum cut.

- Suppose the edges that we choose are  $e_1, e_2, \dots, e_{n-2}$
- $\text{PR}[\text{return } S^*] = \text{PR}[\text{none of the } e_i \text{ cross } S^*]$

$$= \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \left(\frac{n-4}{n-2}\right) \left(\frac{n-5}{n-3}\right) \left(\frac{n-6}{n-4}\right) \cdots \left(\frac{4}{6}\right) \left(\frac{3}{5}\right) \left(\frac{2}{4}\right) \left(\frac{1}{3}\right)$$

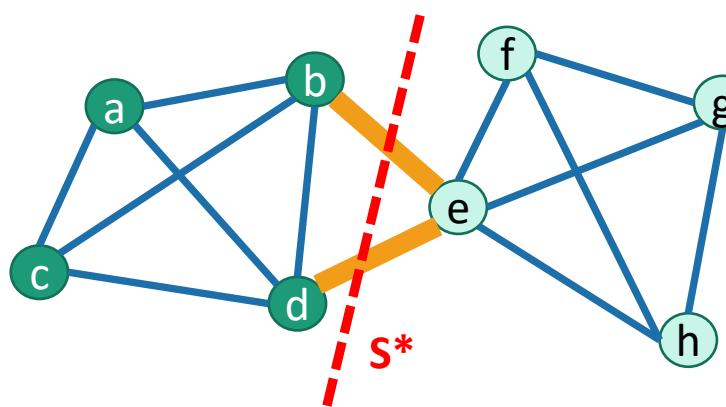


# Now let's prove that claim

Say that  $S^*$  is a minimum cut.

- Suppose the edges that we choose are  $e_1, e_2, \dots, e_{n-2}$
- $\text{PR}[\text{return } S^*] = \text{PR}[\text{none of the } e_i \text{ cross } S^*]$

$$= \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \left(\frac{n-4}{n-2}\right) \left(\frac{n-5}{n-3}\right) \left(\frac{n-6}{n-4}\right) \cdots \left(\frac{4}{6}\right) \left(\frac{3}{5}\right) \left(\frac{2}{4}\right) \left(\frac{1}{3}\right)$$

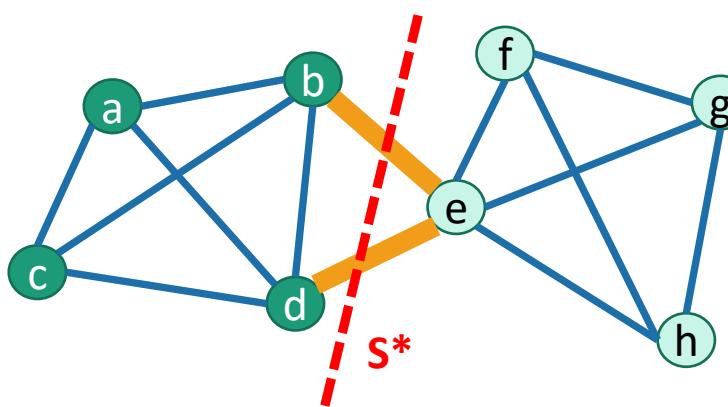


# Now let's prove that claim

Say that  $S^*$  is a minimum cut.

- Suppose the edges that we choose are  $e_1, e_2, \dots, e_{n-2}$
- $\text{PR}[\text{return } S^*] = \text{PR}[\text{none of the } e_i \text{ cross } S^*]$

$$\begin{aligned}&= \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \left(\frac{n-4}{n-2}\right) \left(\frac{n-5}{n-3}\right) \left(\frac{n-6}{n-4}\right) \cdots \left(\frac{4}{6}\right) \left(\frac{3}{5}\right) \left(\frac{2}{4}\right) \left(\frac{1}{3}\right) \\&= \left(\frac{2}{n(n-1)}\right)\end{aligned}$$



# Now let's prove that claim

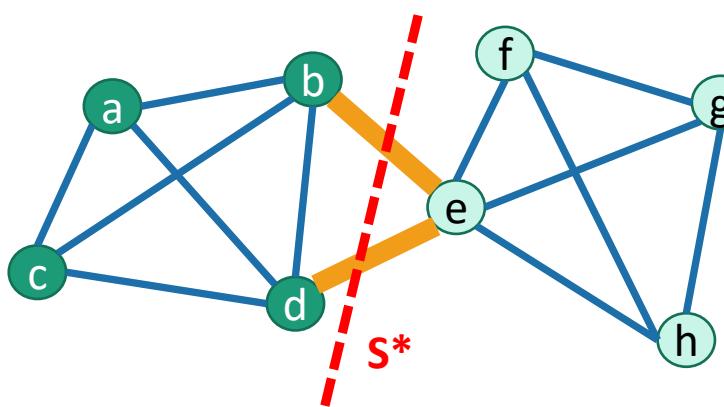
Say that  $S^*$  is a minimum cut.

- Suppose the edges that we choose are  $e_1, e_2, \dots, e_{n-2}$
- $\text{PR}[\text{return } S^*] = \text{PR}[\text{none of the } e_i \text{ cross } S^*]$

$$= \left( \frac{n-2}{n} \right) \left( \frac{n-3}{n-1} \right) \left( \frac{n-4}{n-2} \right) \left( \frac{n-5}{n-3} \right) \left( \frac{n-6}{n-4} \right) \cdots \left( \frac{4}{6} \right) \left( \frac{3}{5} \right) \left( \frac{2}{4} \right) \left( \frac{1}{3} \right)$$

$$= \left( \frac{2}{n(n-1)} \right)$$

$$= \frac{1}{\binom{n}{2}}$$



# Now let's prove that claim

Say that  $S^*$  is a minimum cut.

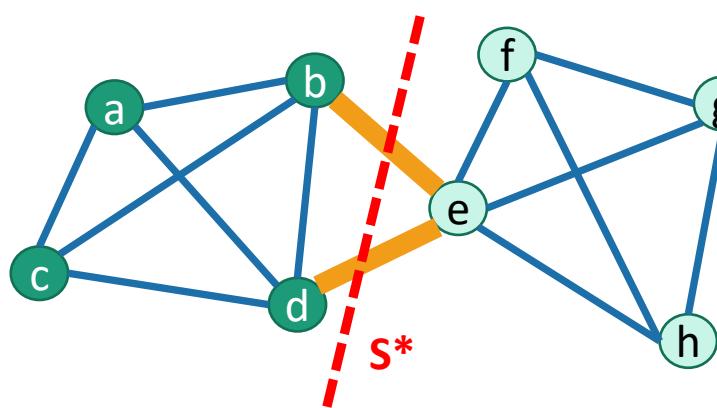
- Suppose the edges that we choose are  $e_1, e_2, \dots, e_{n-2}$
- $\text{PR}[\text{return } S^*] = \text{PR}[\text{none of the } e_i \text{ cross } S^*]$

$$= \left( \frac{n-2}{n} \right) \left( \frac{n-3}{n-1} \right) \left( \frac{n-4}{n-2} \right) \left( \frac{n-5}{n-3} \right) \left( \frac{n-6}{n-4} \right) \cdots \left( \frac{4}{6} \right) \left( \frac{3}{5} \right) \left( \frac{2}{4} \right) \left( \frac{1}{3} \right)$$

$$= \left( \frac{2}{n(n-1)} \right)$$

$$= \frac{1}{\binom{n}{2}}$$

**CLAIM  
PROVED**



# Theorem

Assuming the claim about  $1/\binom{n}{2} \dots$

Suppose  $G$  has  $n$  vertices. Then [repeating Karger's algorithm] finds a min cut in  $G$  with probability at least 0.99 in time  $O(n^4)$ .

**That proves this  
Theorem!**

# What have we learned?

- If we randomly contract edges:
  - It's unlikely that we'll end up with a min cut.
  - But it's not **TOO** unlikely
  - By repeating, we likely will find a min cut.
- Repeating this process:
  - Finds a **global min cut in time  $O(n^4)$ , with probability 0.99.**
  - We can run a bit faster if we use a **union-find** data structure.

Here I chose  $\delta = 0.01$   
just for concreteness.



# More generally

- If we have a Monte-Carlo algorithm<sup>1</sup> with a small success probability,
- and we can check how good a solution is,
- Then we can **boost** the success probability by repeating it a bunch and taking the best solution.



<sup>1</sup>In computing, a Monte Carlo algorithm is a randomized algorithm whose output may be incorrect with a certain (typically small) probability.

# Can we do better?

- Repeating  $O(n^2)$  times is pretty expensive.
  - $O(n^4)$  total runtime to get success probability 0.99.

# Can we do better?

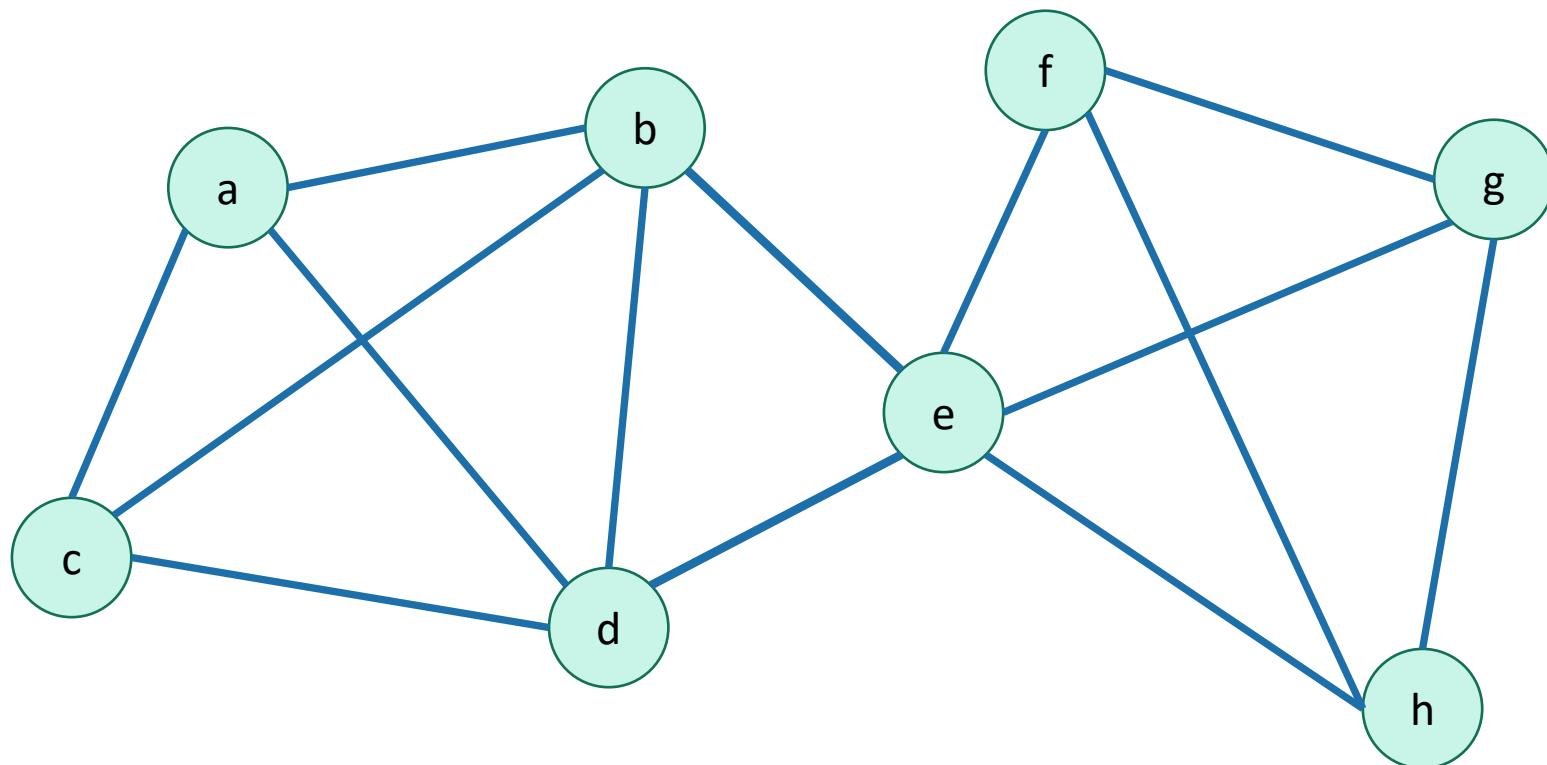
- Repeating  $O(n^2)$  times is pretty expensive.
  - $O(n^4)$  total runtime to get success probability 0.99.
- The **Karger-Stein Algorithm** will do better!
  - The trick is that we'll do the repetitions in a clever way.
  - $O(n^2 \log^2(n))$  runtime for the same success probability.
  - **Warning!** This is a tricky algorithm! We'll sketch the approach here: the important part is the high-level idea, not the details of the computations.

# Can we do better?

- Repeating  $O(n^2)$  times is pretty expensive.
  - $O(n^4)$  total runtime to get success probability 0.99.
- The **Karger-Stein Algorithm** will do better!
  - The trick is that we'll do the repetitions in a clever way.
  - $O(n^2 \log^2(n))$  runtime for the same success probability.
  - **Warning!** This is a tricky algorithm! We'll sketch the approach here: the important part is the high-level idea, not the details of the computations.

To see how we might save on repetitions,  
let's run through Karger's algorithm again.

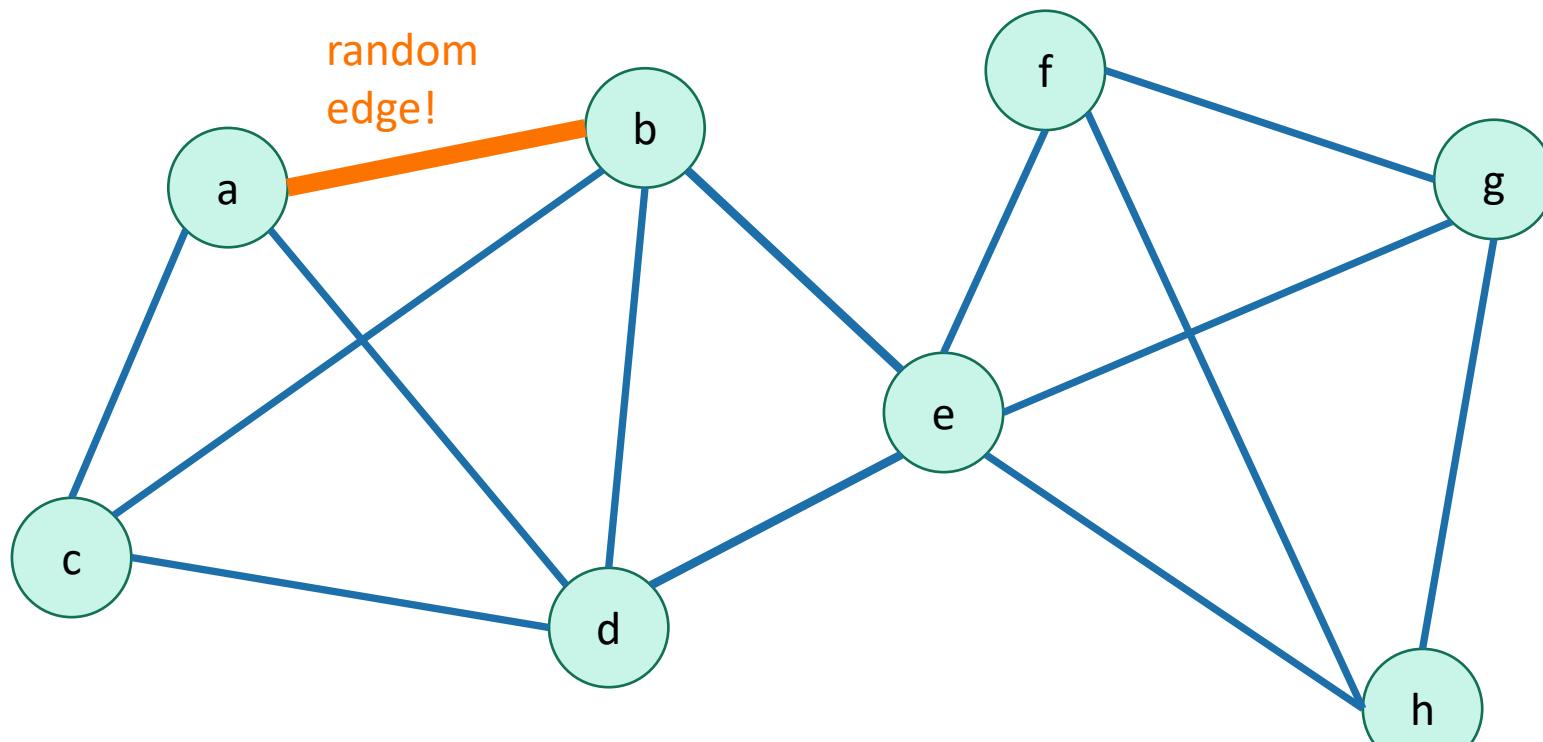
# Karger's algorithm



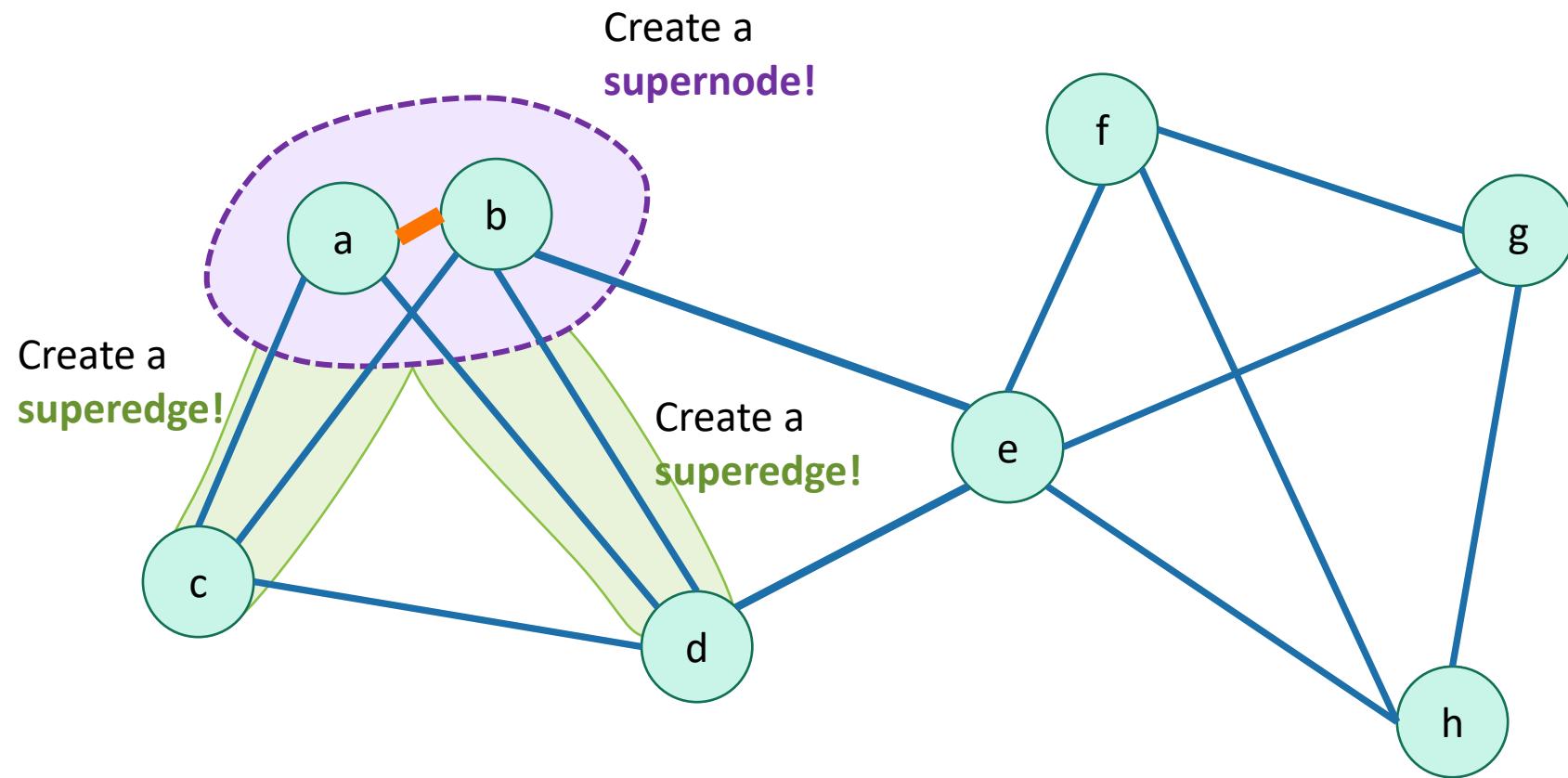
# Karger's algorithm

Probability that we didn't mess up:  
**12/14**

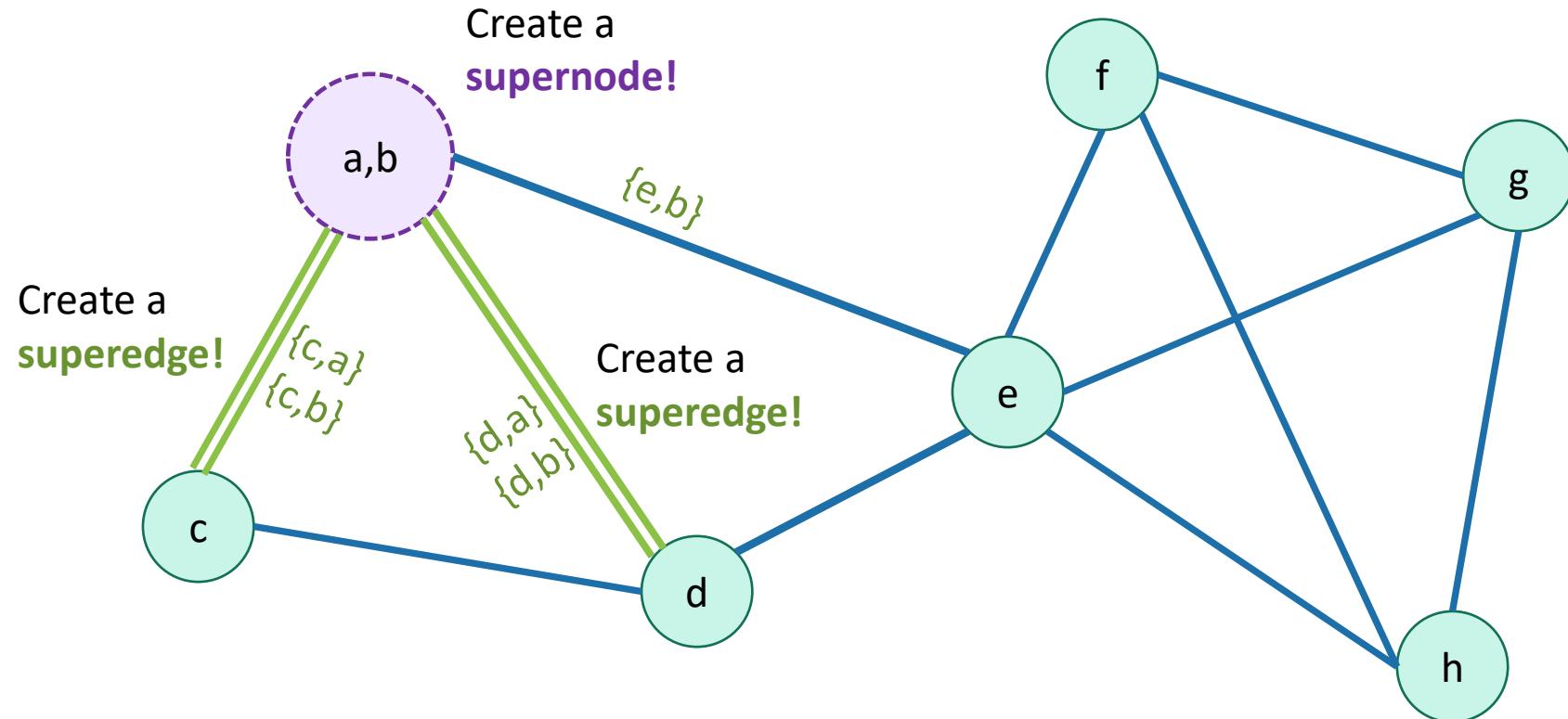
There are 14 edges, 12 of  
which are good to contract.



# Karger's algorithm



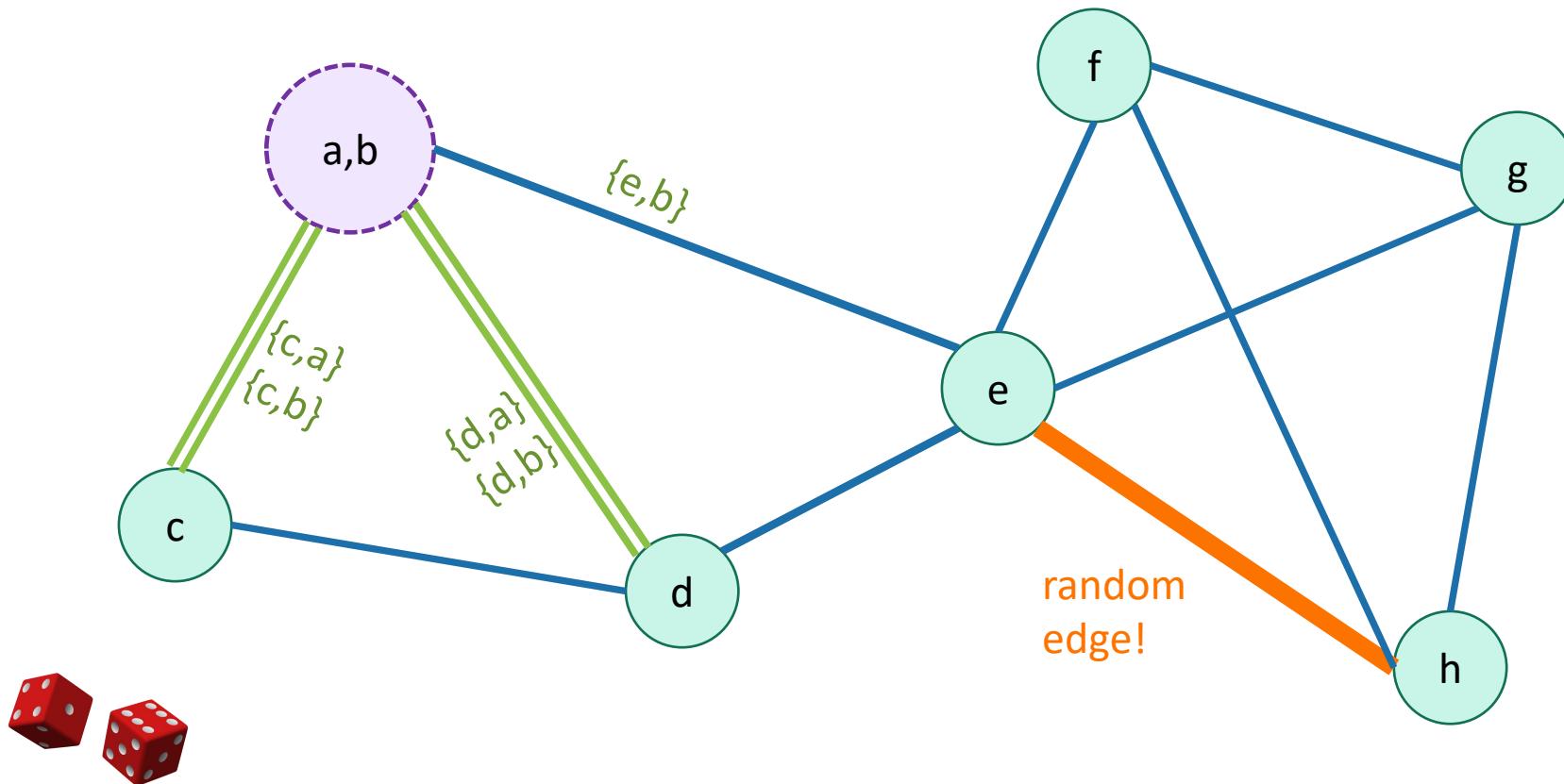
# Karger's algorithm



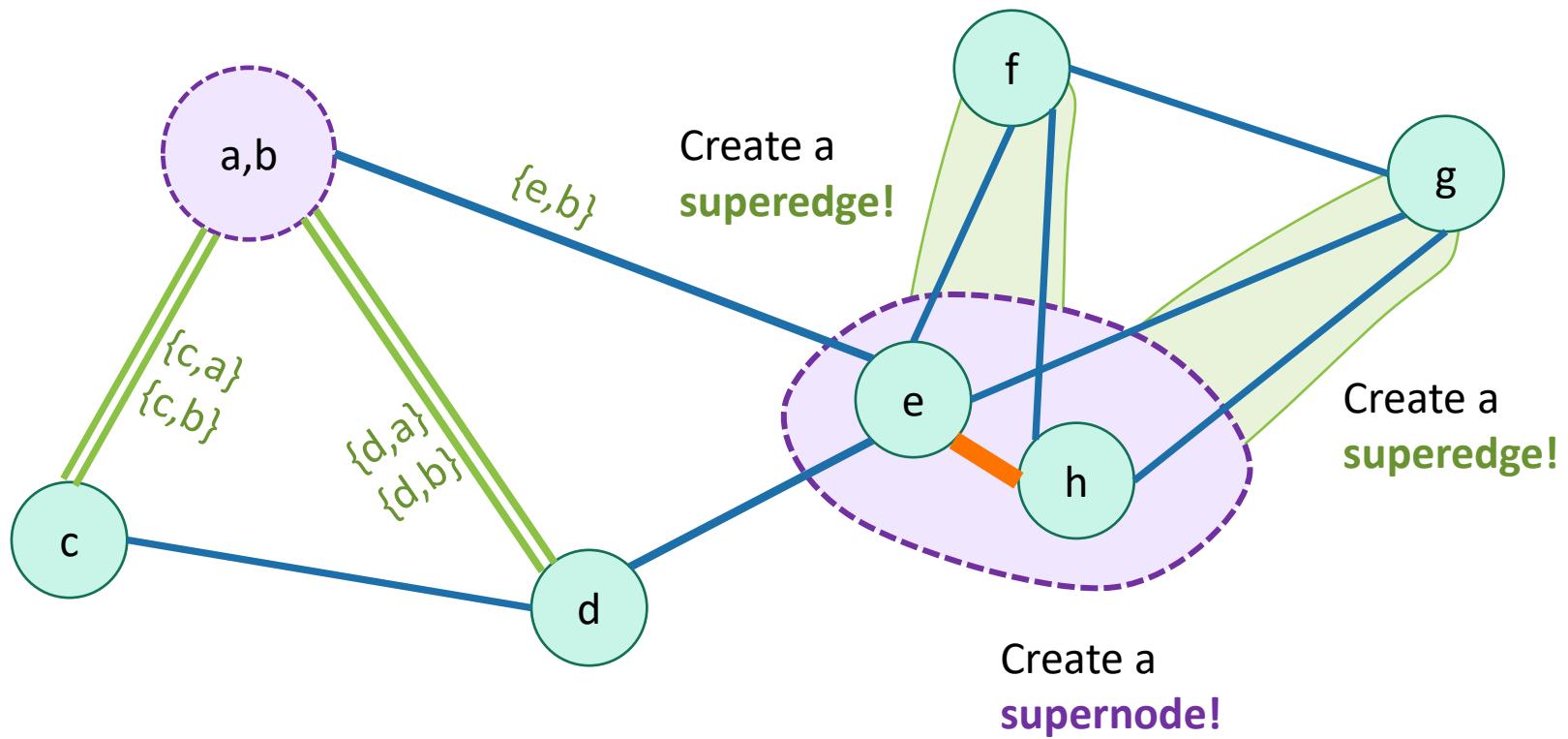
# Karger's algorithm

Probability that we didn't mess up:  
**11/13**

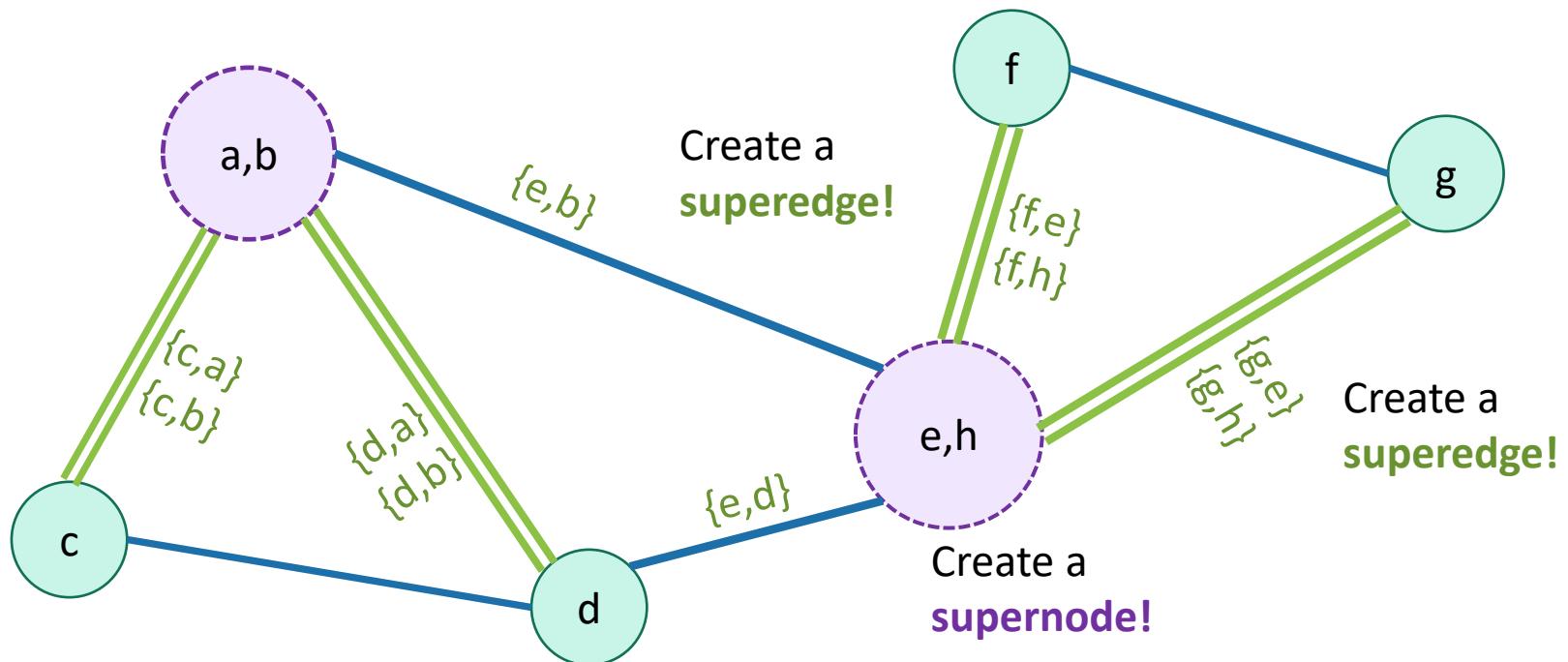
Now there are only 13 edges,  
since the edge between a and b  
disappeared.



# Karger's algorithm



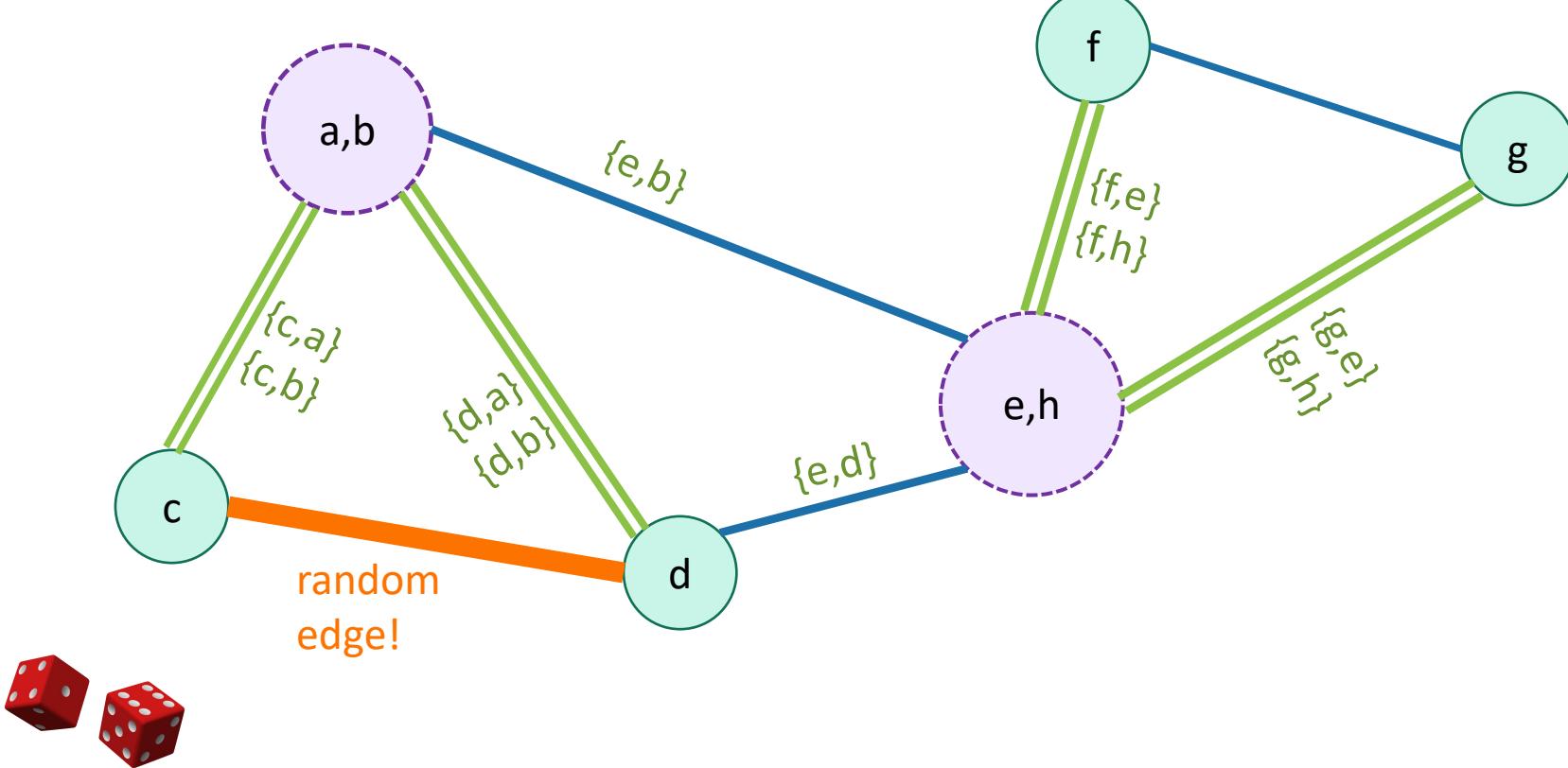
# Karger's algorithm



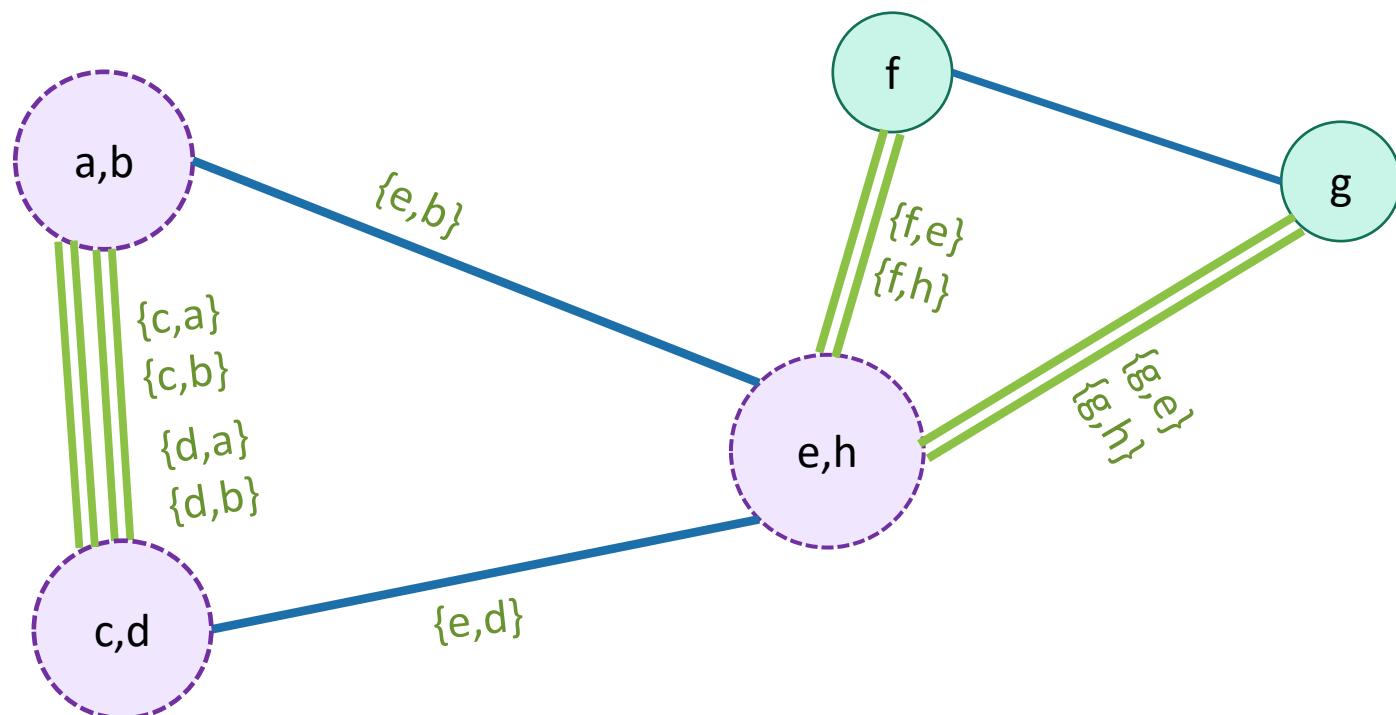
# Karger's algorithm

Probability that we didn't mess up:  
**10/12**

Now there are only 12 edges,  
since the edge between e and h  
disappeared.

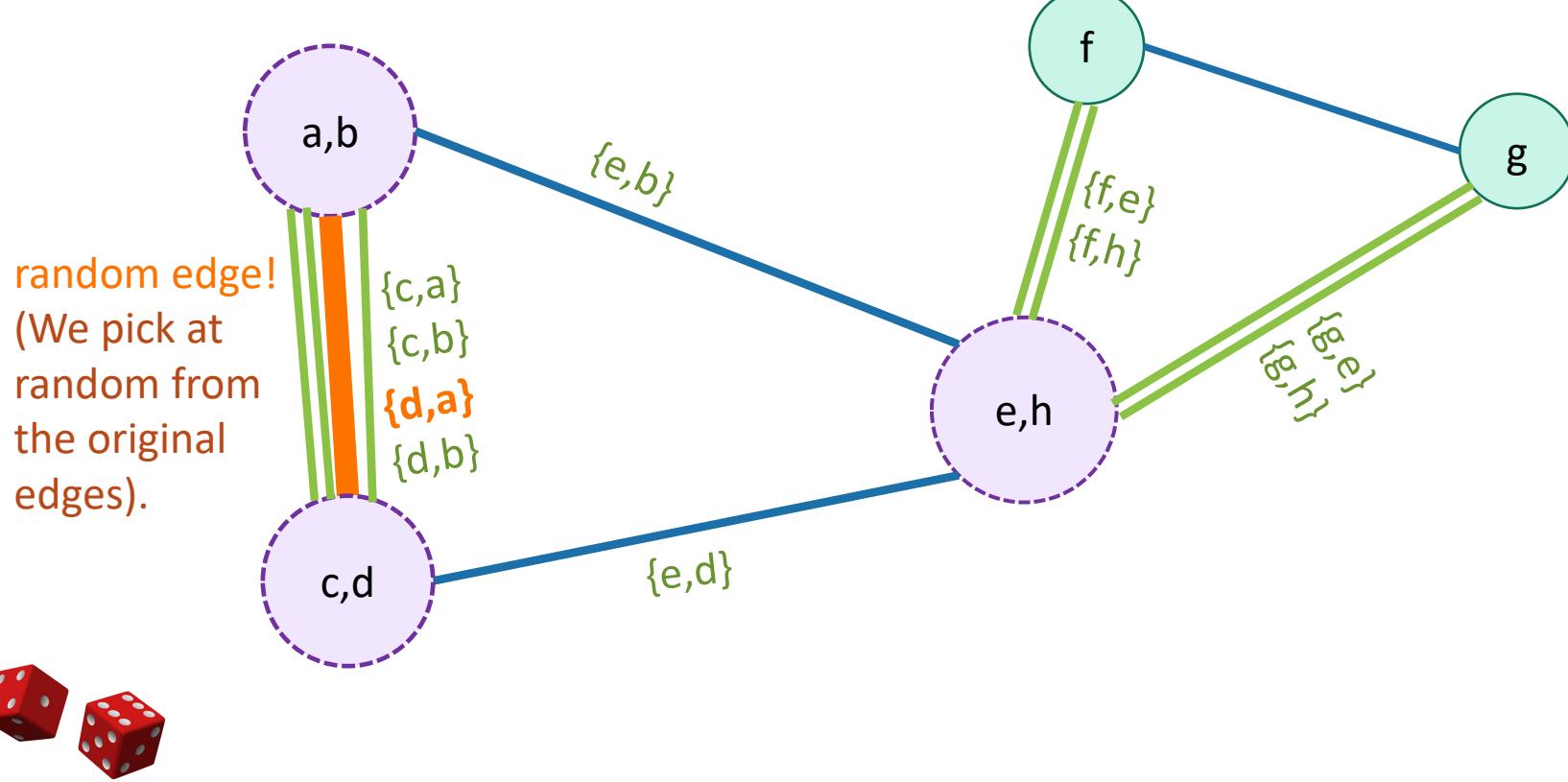


# Karger's algorithm

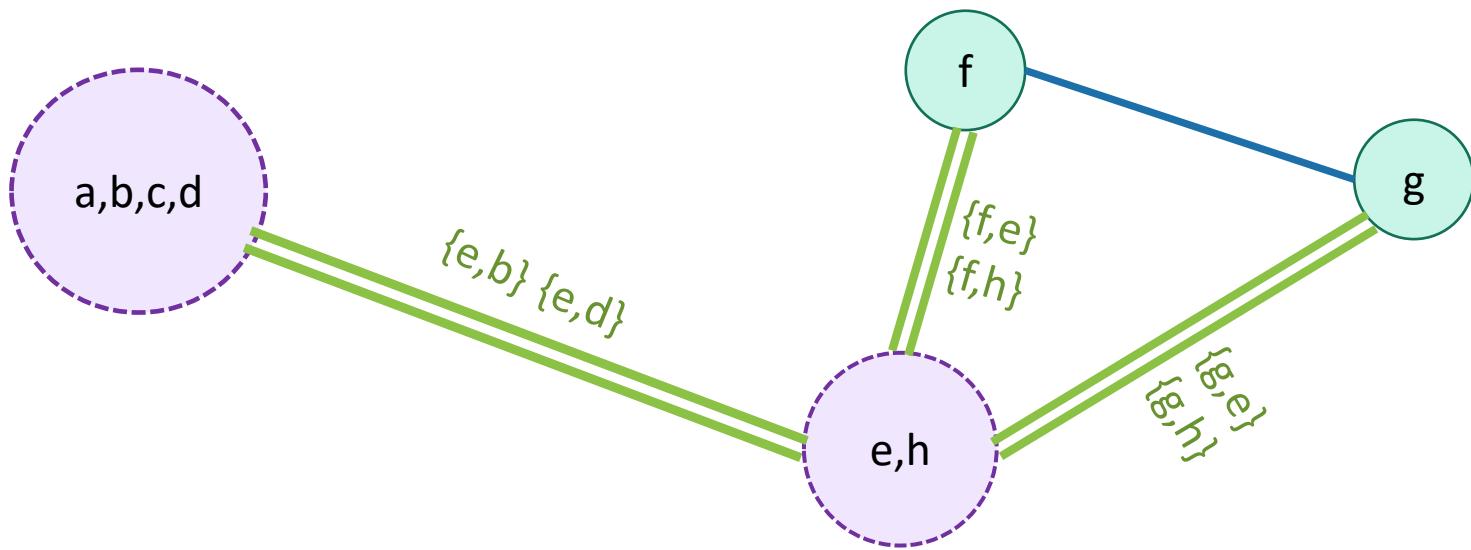


# Karger's algorithm

Probability that we didn't mess up:  
**9/11**

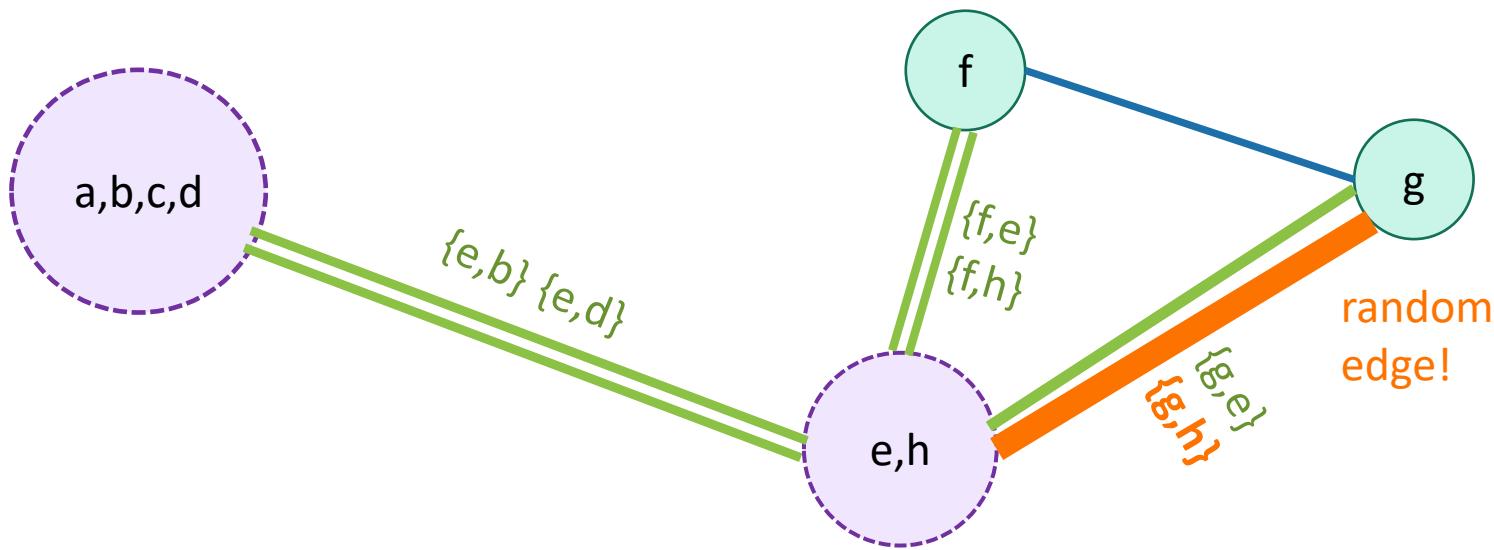


# Karger's algorithm

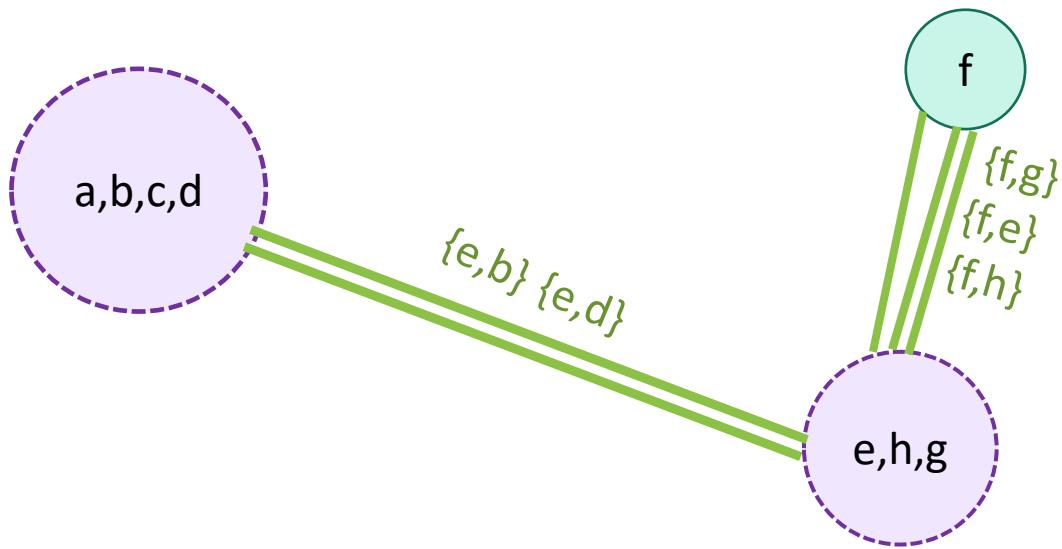


# Karger's algorithm

Probability that we didn't mess up:  
5/7

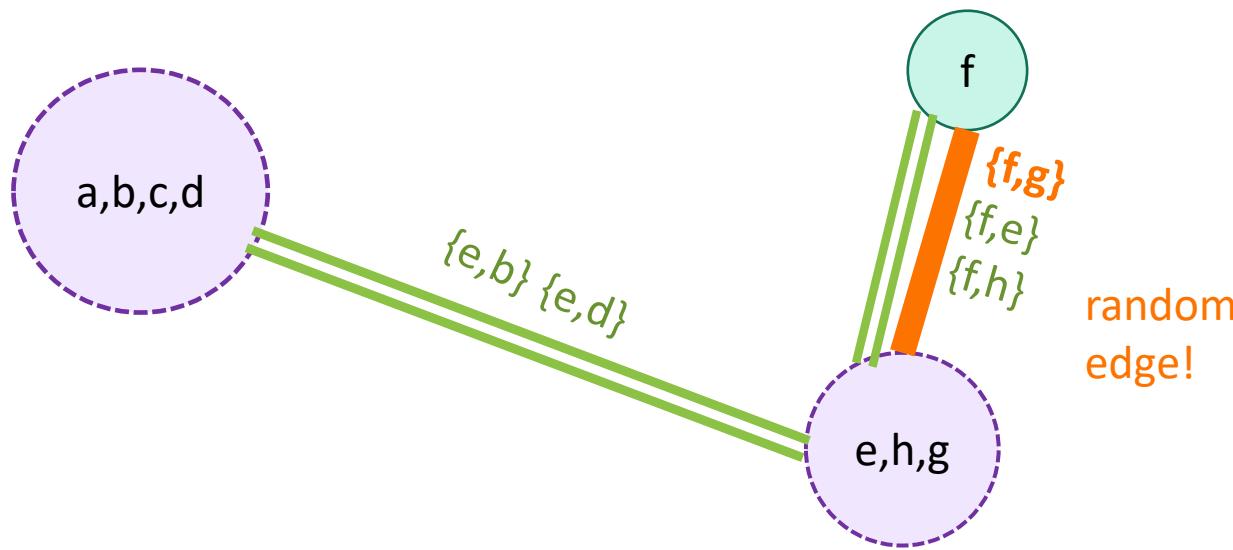


# Karger's algorithm

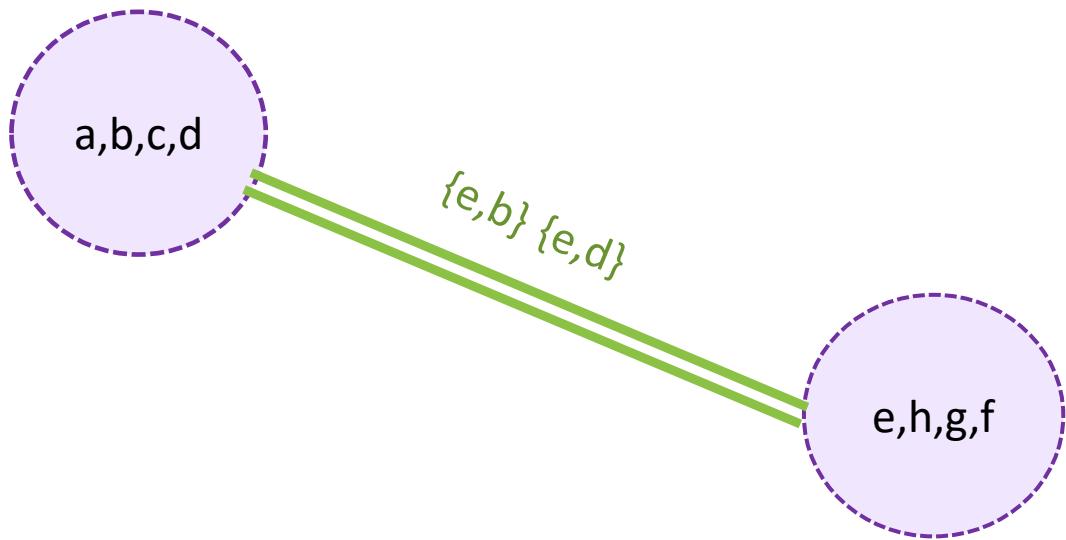


# Karger's algorithm

Probability that we didn't mess up:  
3/5



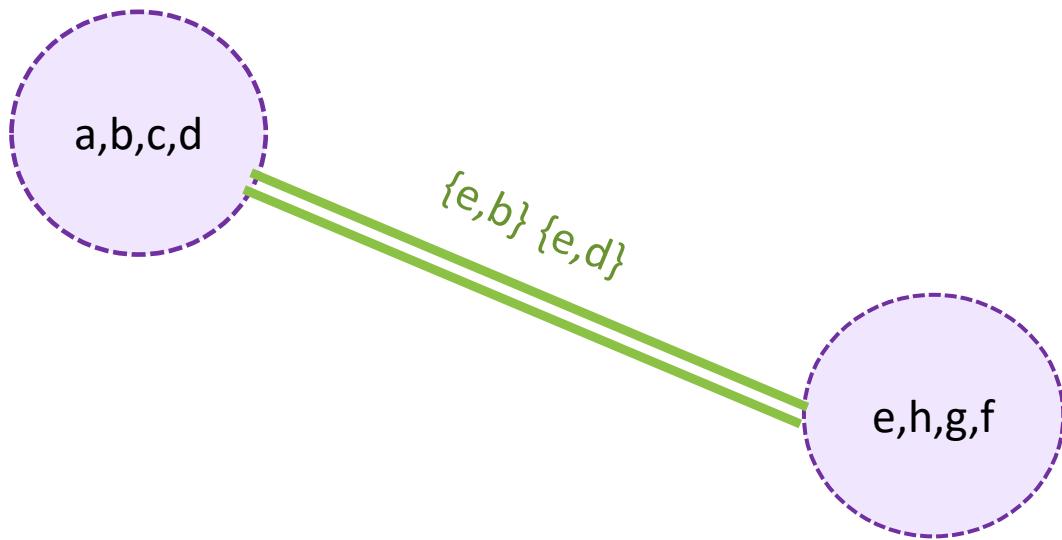
# Karger's algorithm



# Karger's algorithm

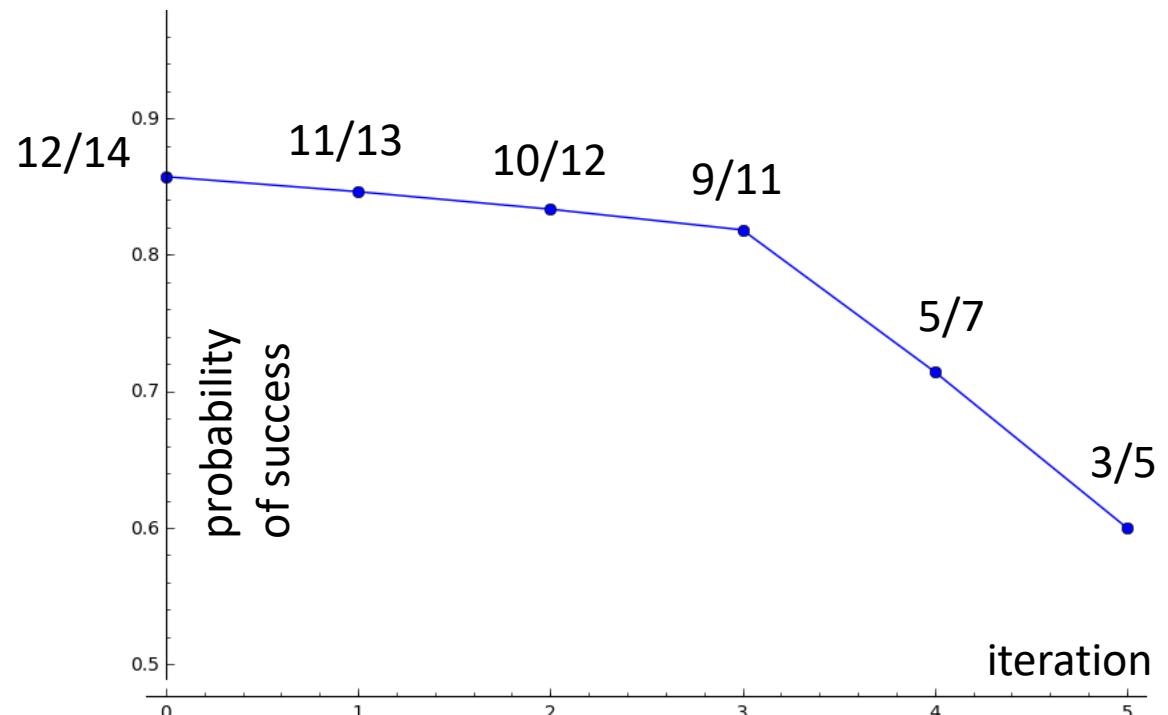
**Now stop!**

- There are only two nodes left.



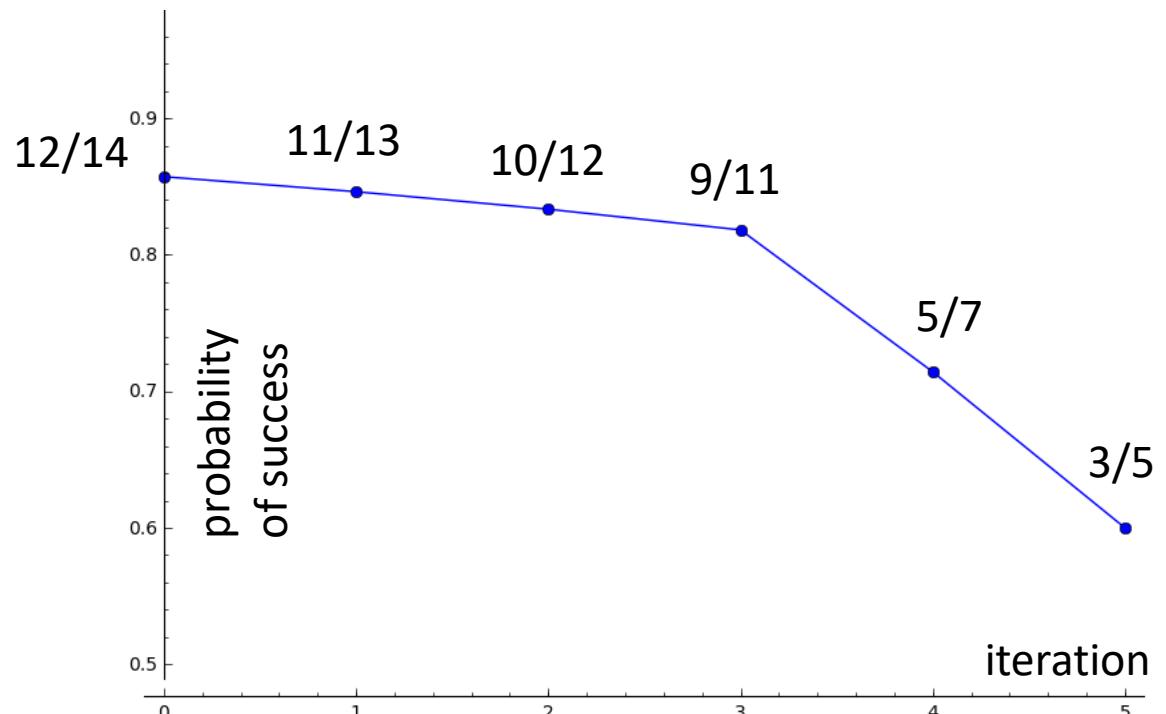
# Probability of not messing up

- At the beginning, it's pretty likely we'll be fine.
- The probability that we mess up gets worse and worse over time.



# Probability of not messing up

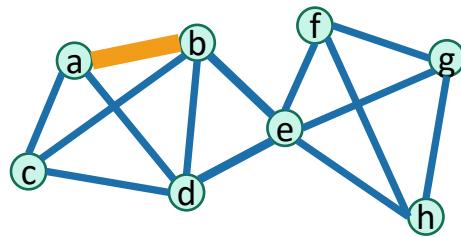
- At the beginning, it's pretty likely we'll be fine.
- The probability that we mess up gets worse and worse over time.



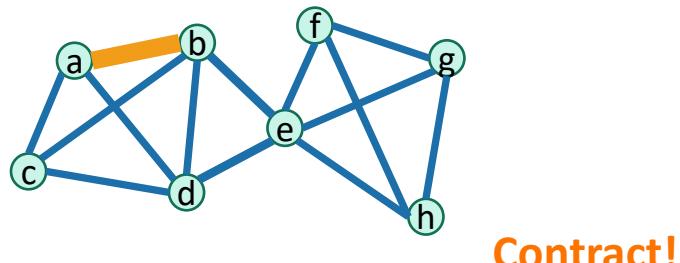
## Moral:

Repeating the stuff from  
the beginning of the  
algorithm is **wasteful!**

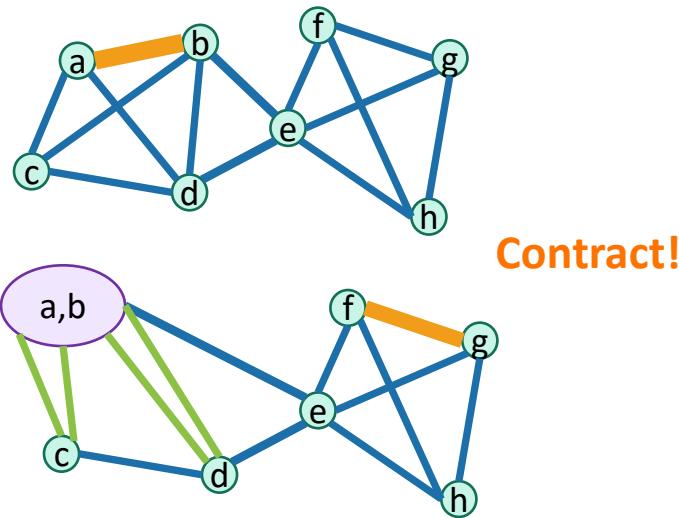
Instead...



Instead...

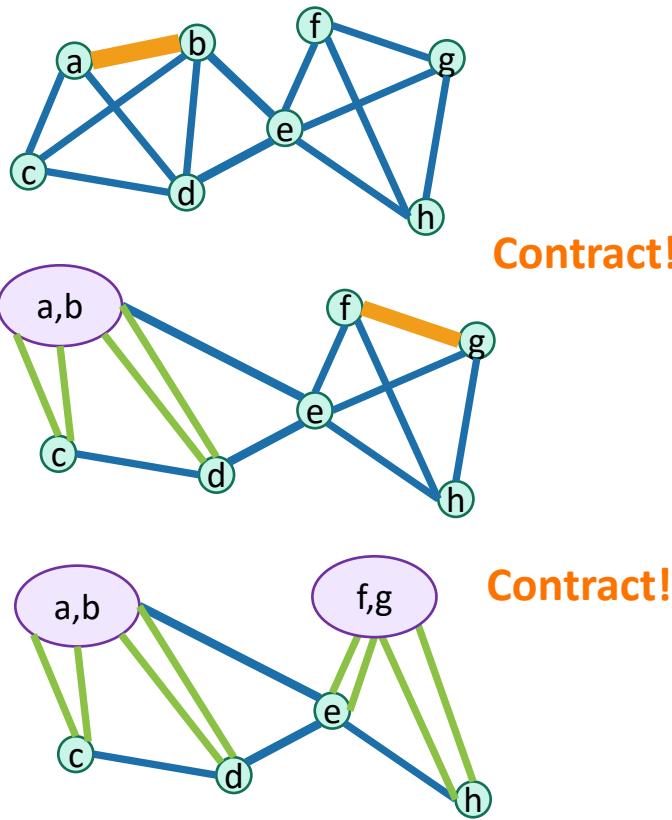


Instead...

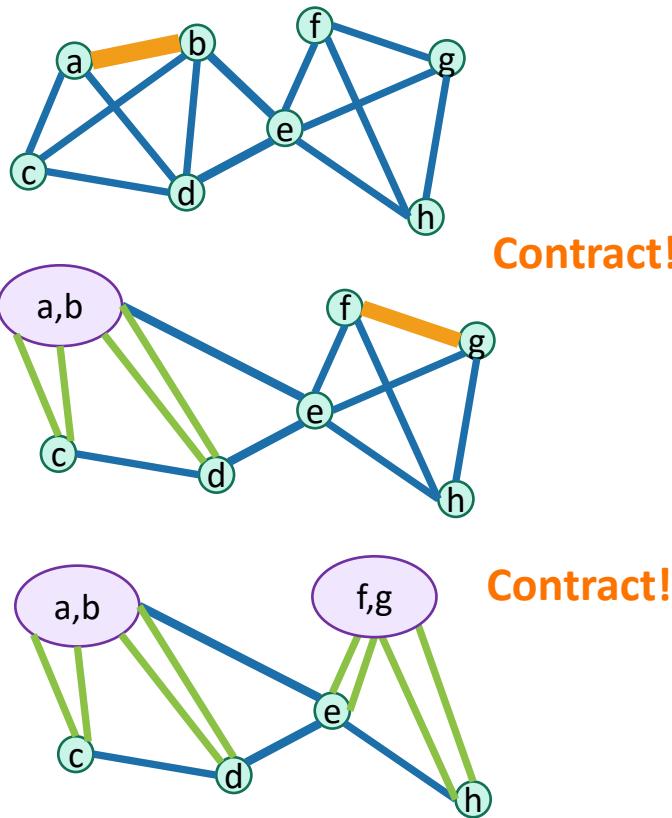


Contract!

Instead...



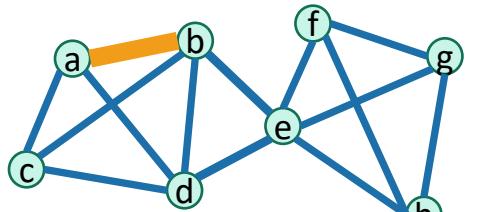
Instead...



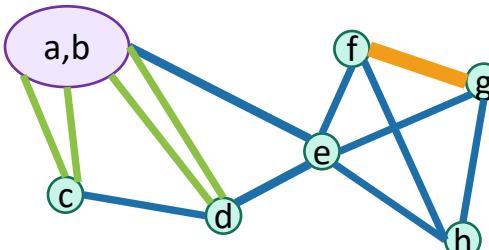
**FORK!**



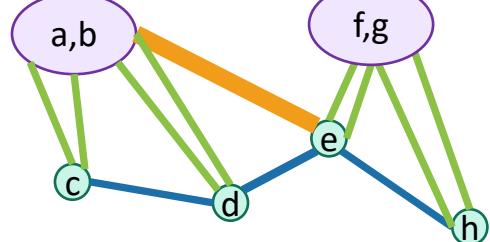
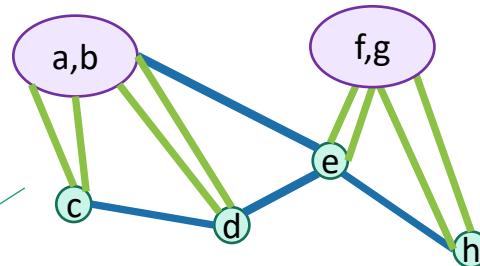
Instead...



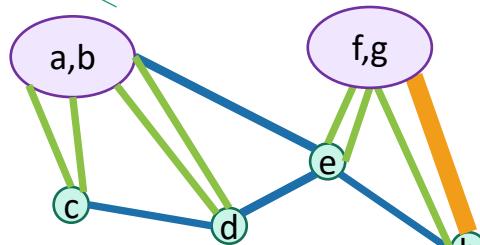
Contract!



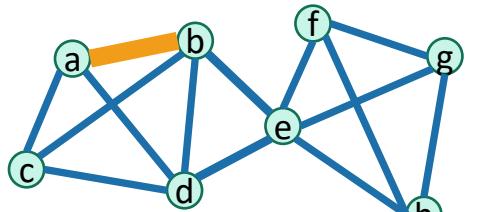
Contract!



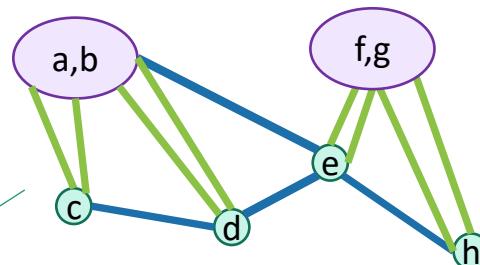
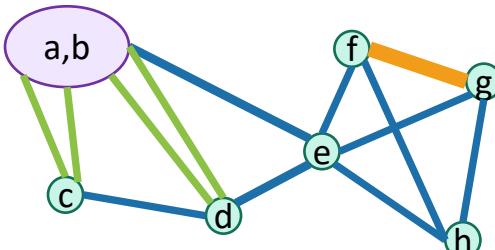
FORK!



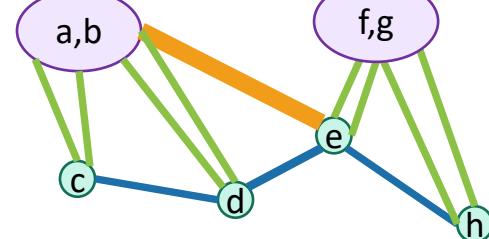
Instead...



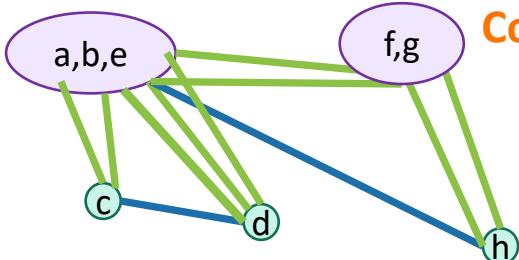
Contract!



Contract!



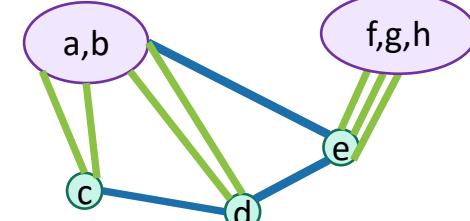
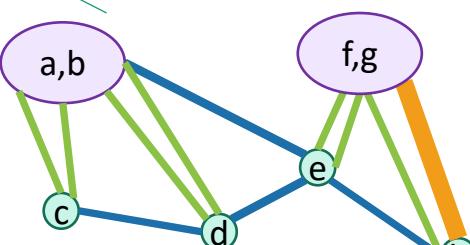
Contract!



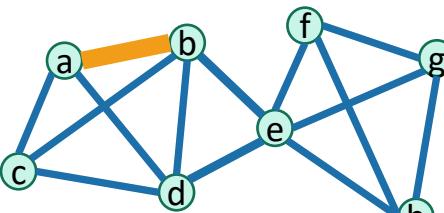
FORK!



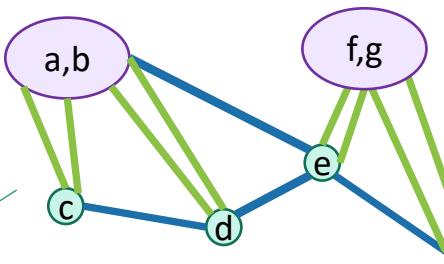
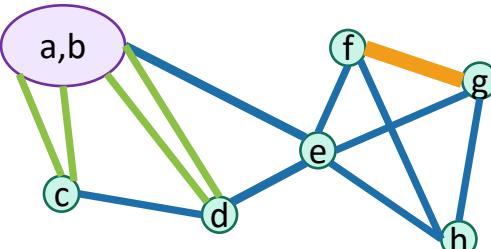
Contract!



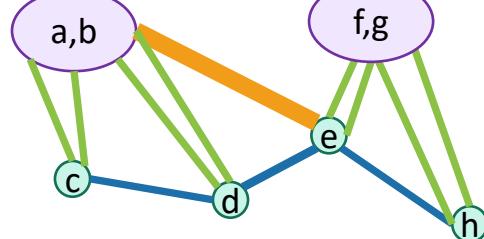
Instead...



Contract!



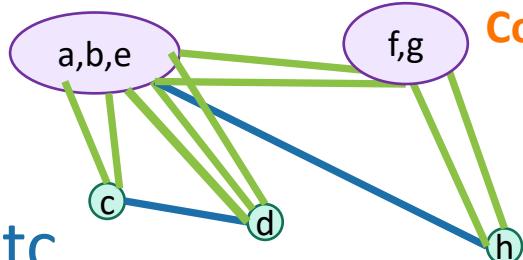
Contract!



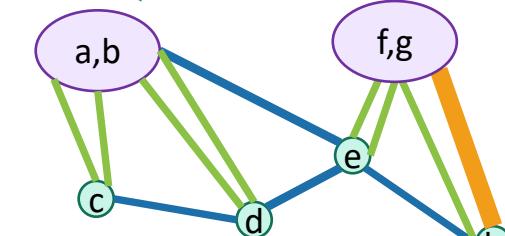
FORK!



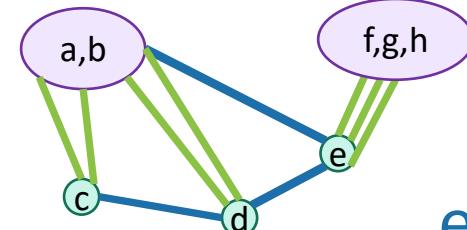
Contract!



etc

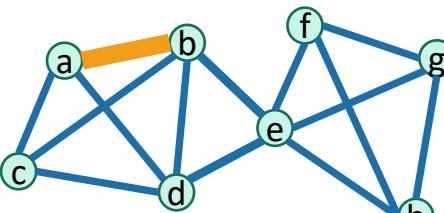


Contract!

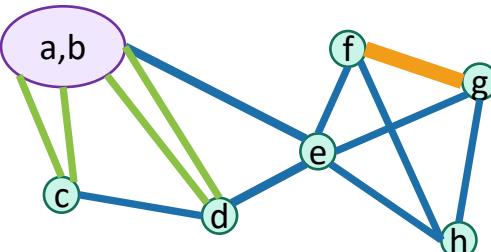


etc

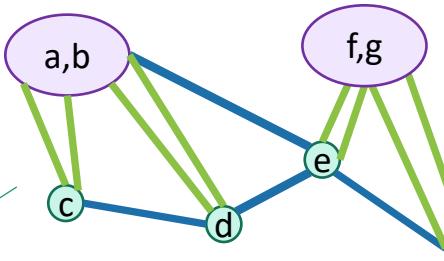
# Instead...



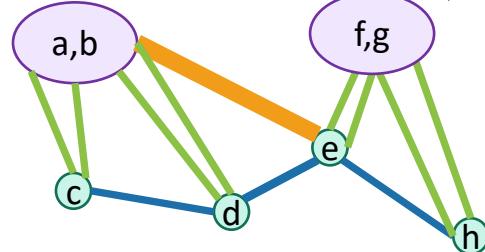
Contract!



This branch  
made a bad  
choice.



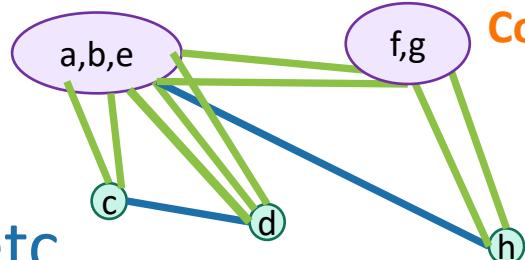
Contract!



FORK!

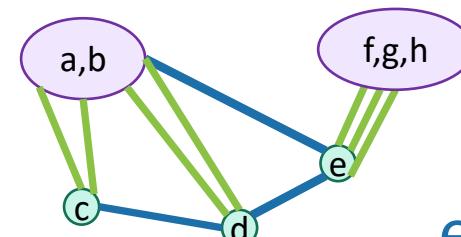


Contract!



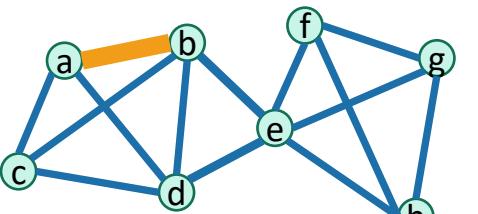
etc

Contract!

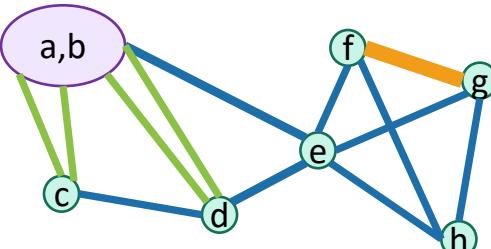


etc

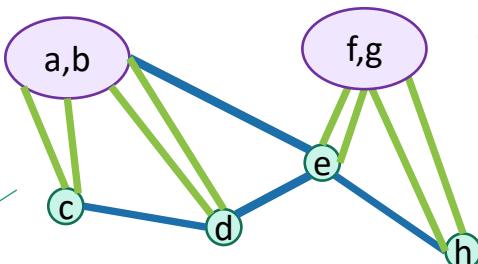
Instead...



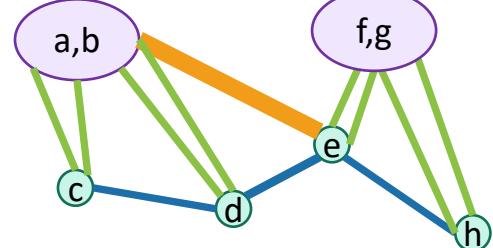
Contract!



This branch  
made a bad  
choice.



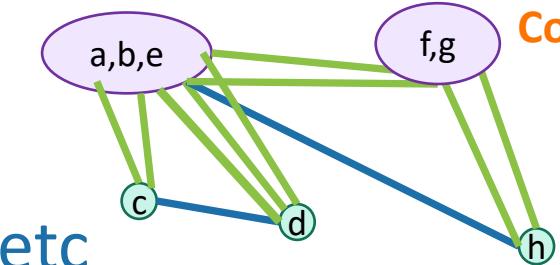
Contract!



FORK!

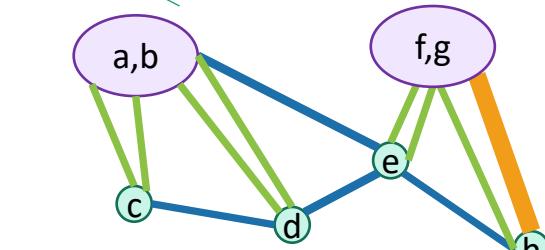


Contract!

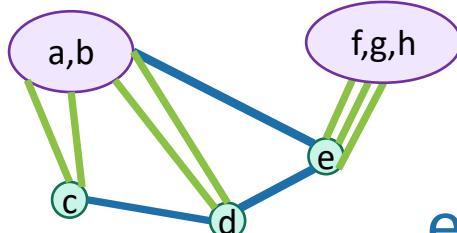


etc

But it's okay  
since this  
branch made a  
good choice.



Contract!



etc

# In words

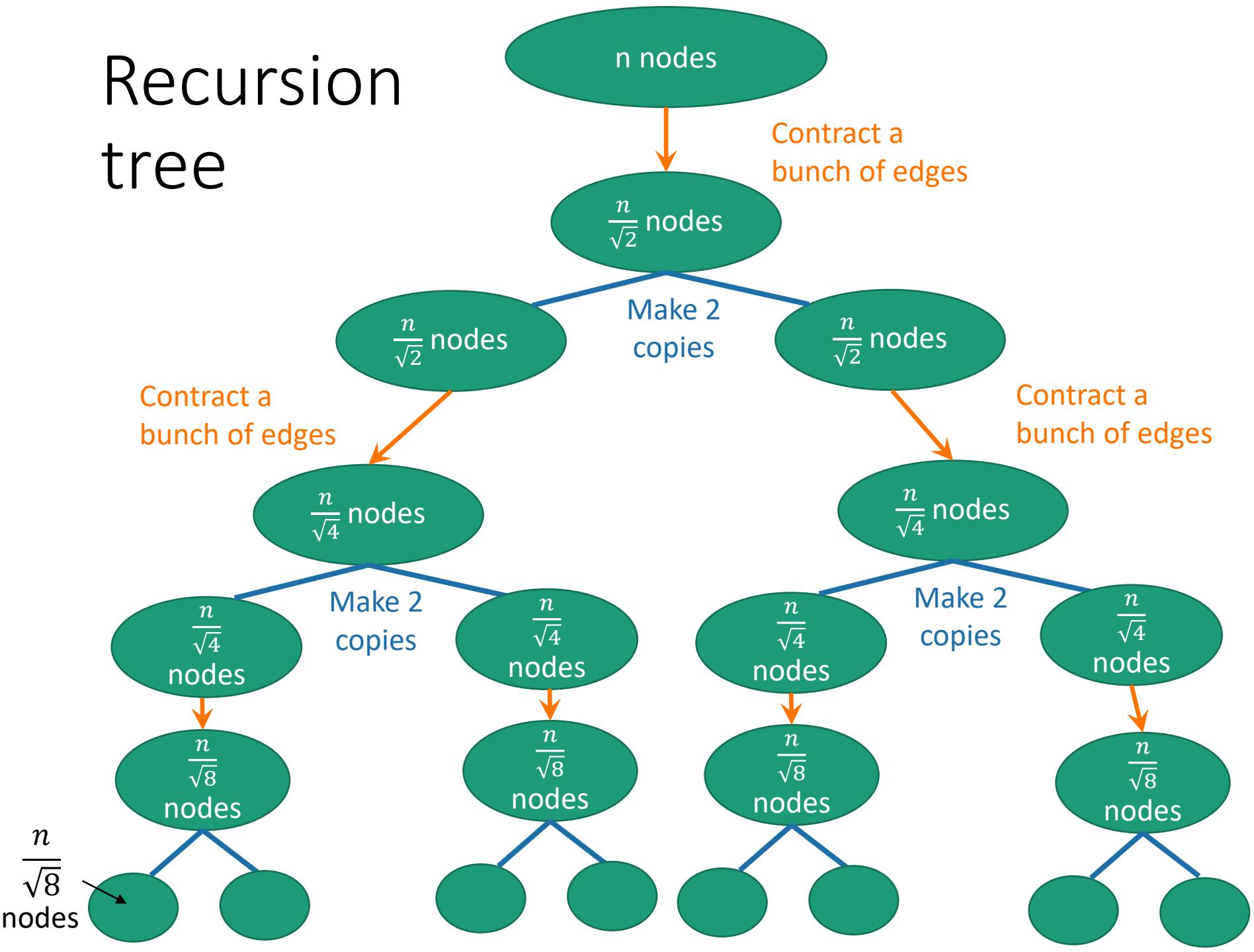
- Run Karger's algorithm on  $G$  for a bit.
  - Until there are  $\frac{n}{\sqrt{2}}$  supernodes left.
- Then split into two independent copies,  $G_1$  and  $G_2$
- Run Karger's algorithm on each of those for a bit.
  - Until there are  $\frac{\left(\frac{n}{\sqrt{2}}\right)}{\sqrt{2}} = \frac{n}{2}$  supernodes left in each.
- Then split each of those into two independent copies...

Why  $\frac{n}{\sqrt{2}}$ ? We'll see later.

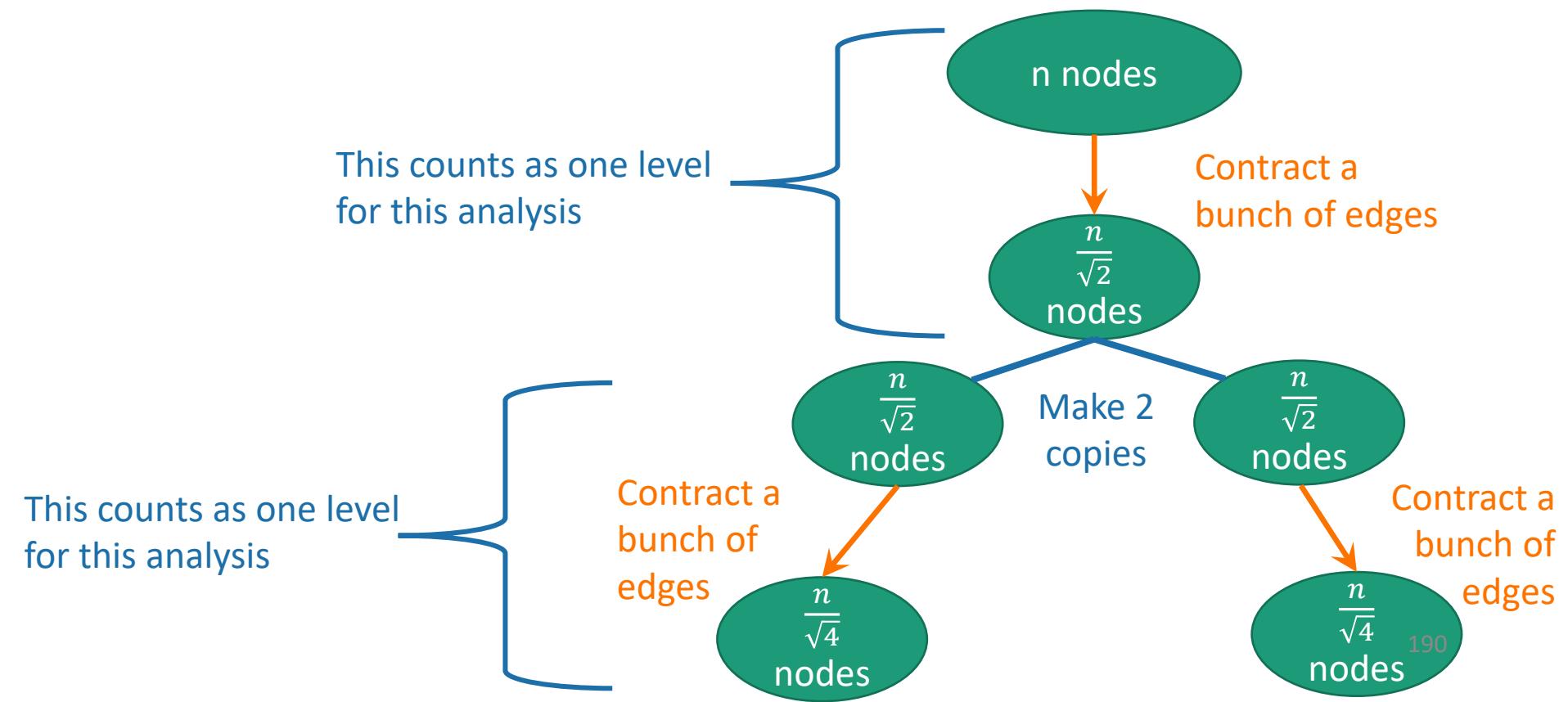
# In pseudocode

- **KargerStein( $G = (V, E)$ ):**
  - $n \leftarrow |V|$
  - if  $n < 4$ :
    - find a min-cut by brute force \\ time  $O(1)$
  - Run Karger's algorithm on  $G$  with independent repetitions until  $\left\lfloor \frac{n}{\sqrt{2}} \right\rfloor$  nodes remain.
  - $G_1, G_2 \leftarrow$  copies of what's left of  $G$
  - $S_1 = \text{KargerStein}(G_1)$
  - $S_2 = \text{KargerStein}(G_2)$
  - **return** whichever of  $S_1, S_2$  is the smaller cut.

# Recursion tree

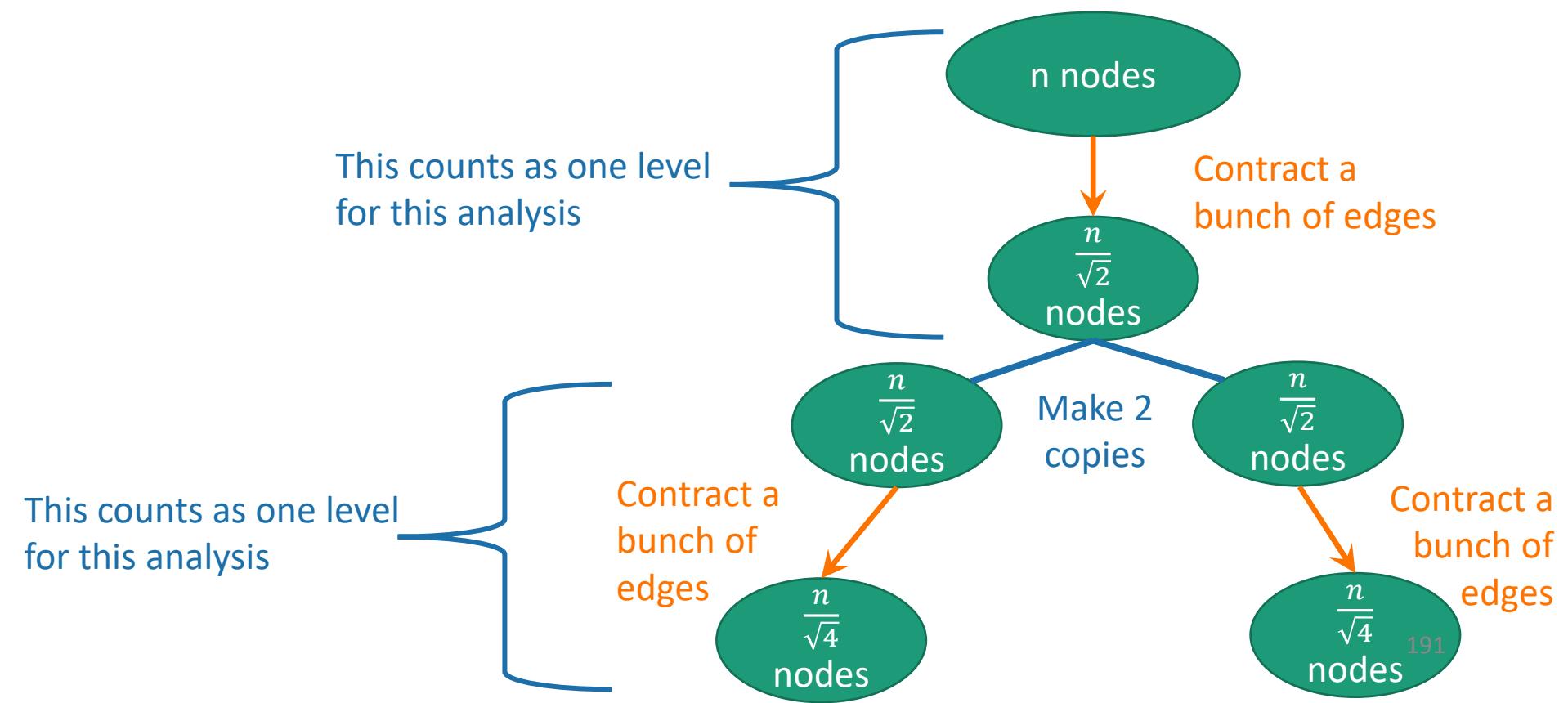


# Recursion tree



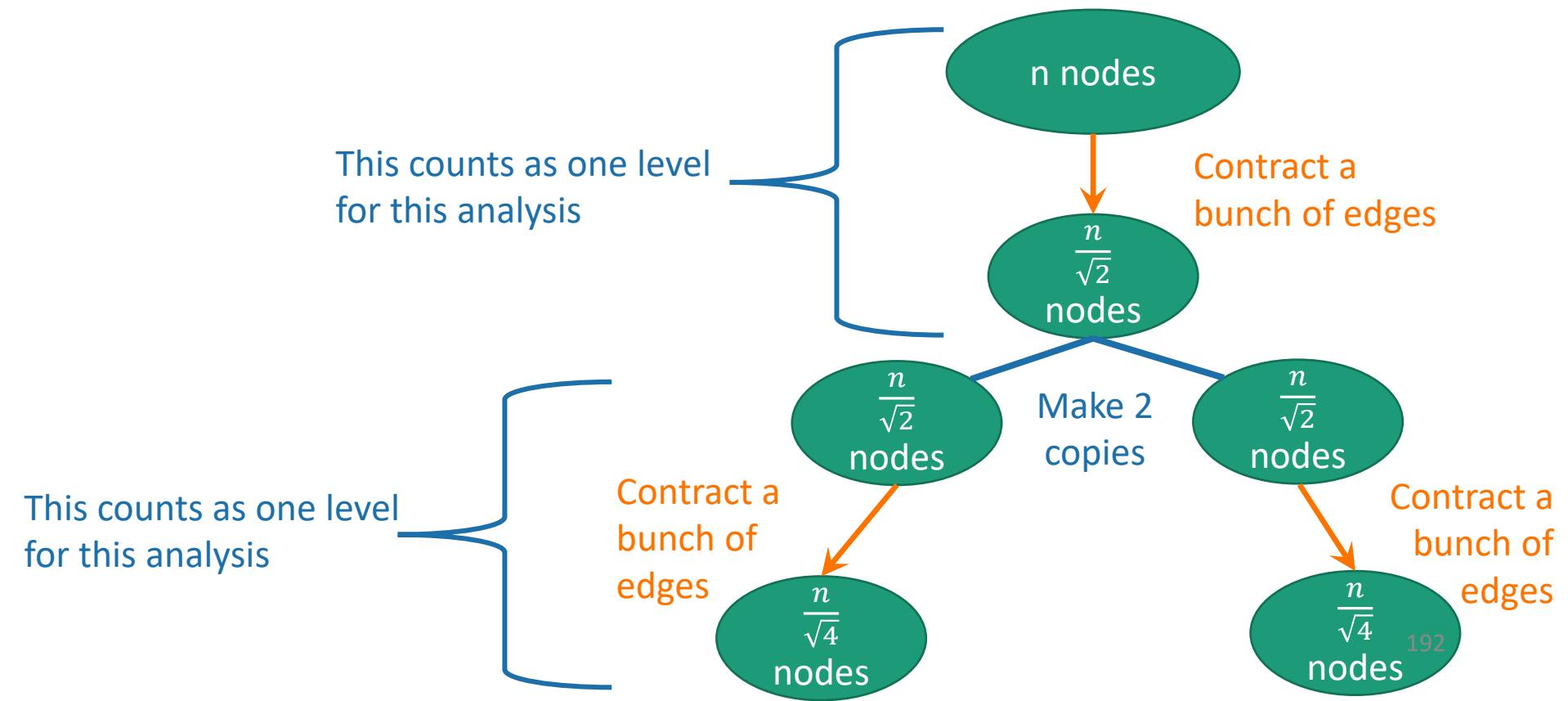
# Recursion tree

- depth is  $\log_{\sqrt{2}}(n) = \frac{\log(n)}{\log(\sqrt{2})} = 2\log(n)$



# Recursion tree

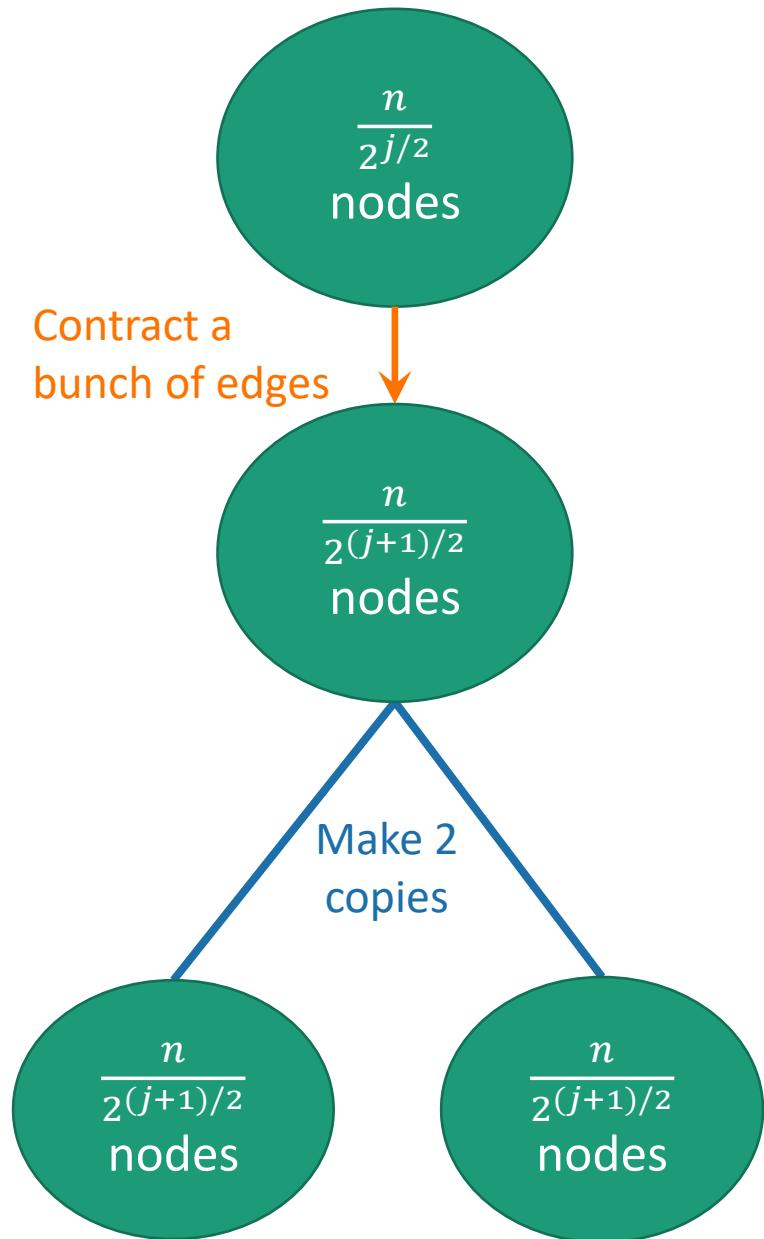
- depth is  $\log_{\sqrt{2}}(n) = \frac{\log(n)}{\log(\sqrt{2})} = 2\log(n)$
- number of leaves is  $2^{2\log(n)} = n^2$



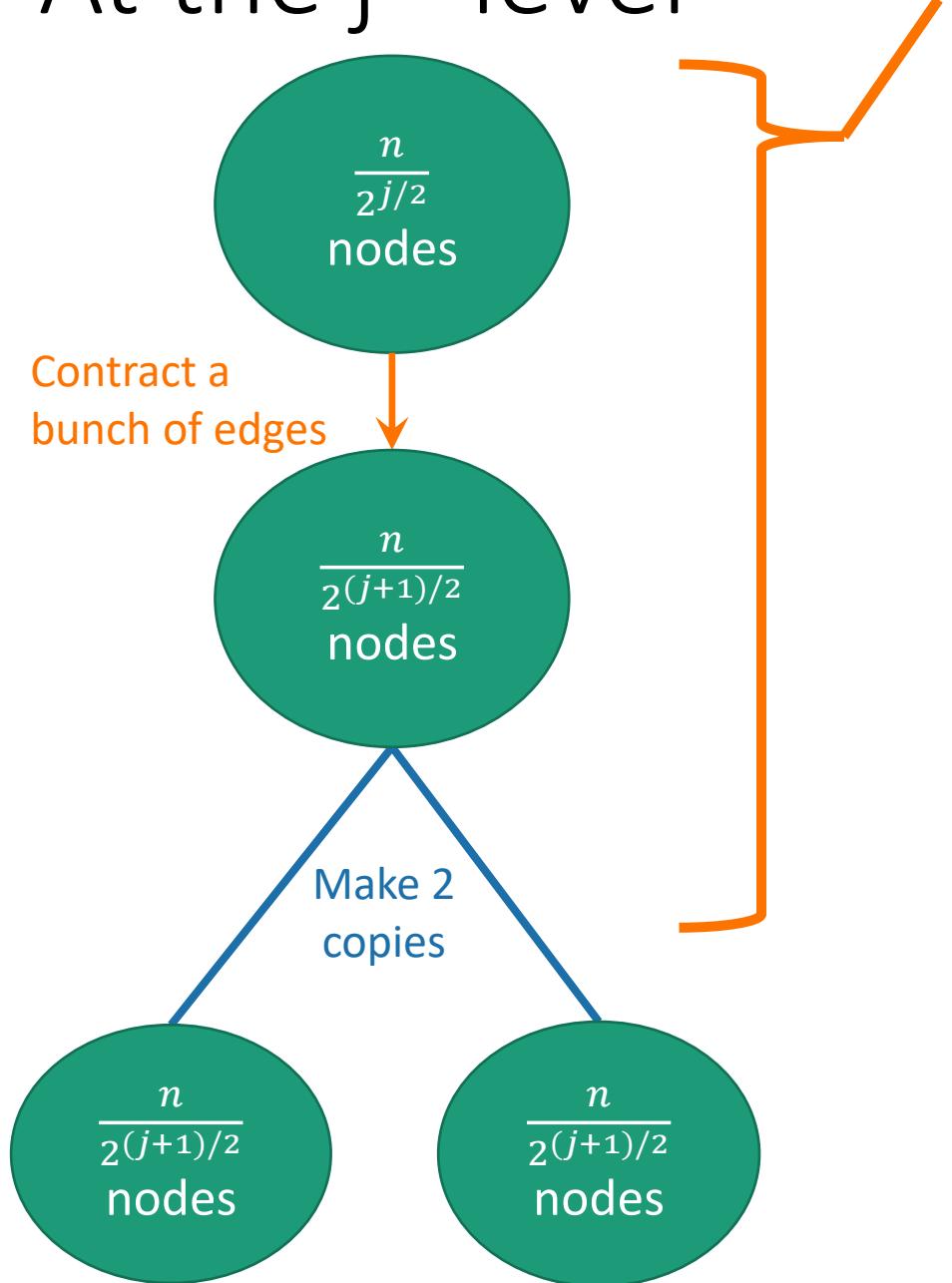
# Two questions

- Does this work?
- Is it fast? 

# At the $j^{\text{th}}$ level

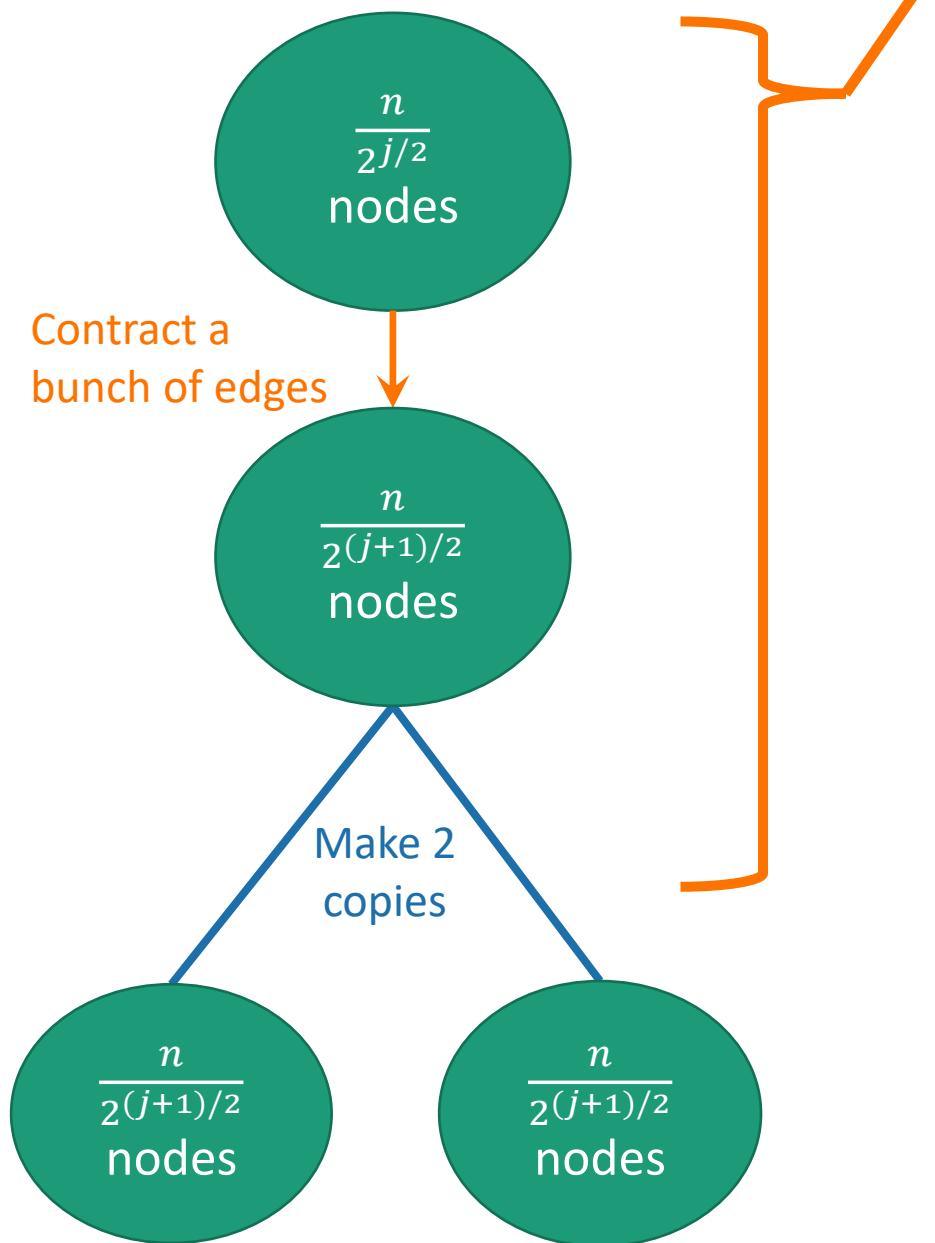


# At the $j^{\text{th}}$ level



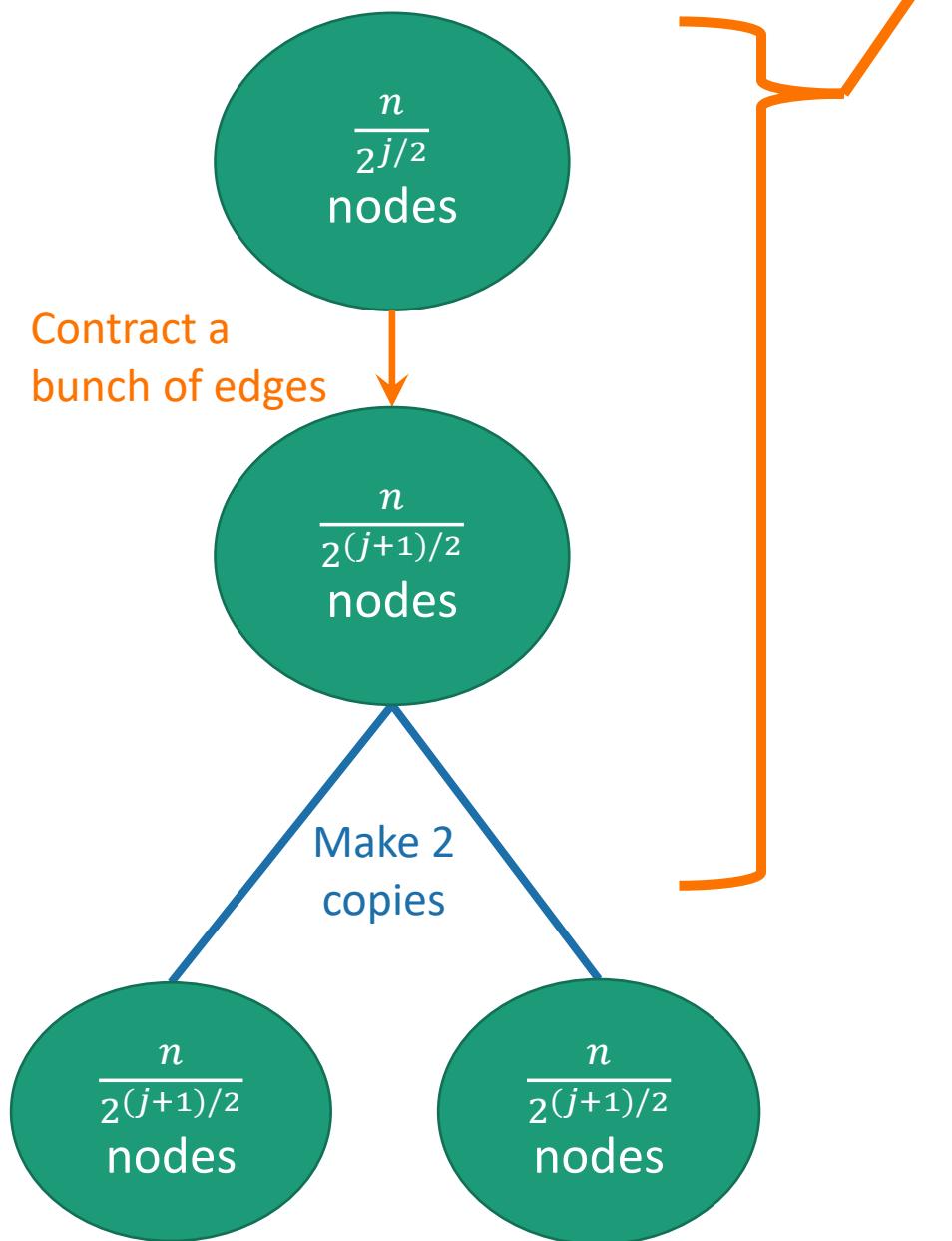
- The amount of work per level is the amount of work needed to reduce the number of nodes by a factor of  $\sqrt{2}$ .

# At the $j^{\text{th}}$ level



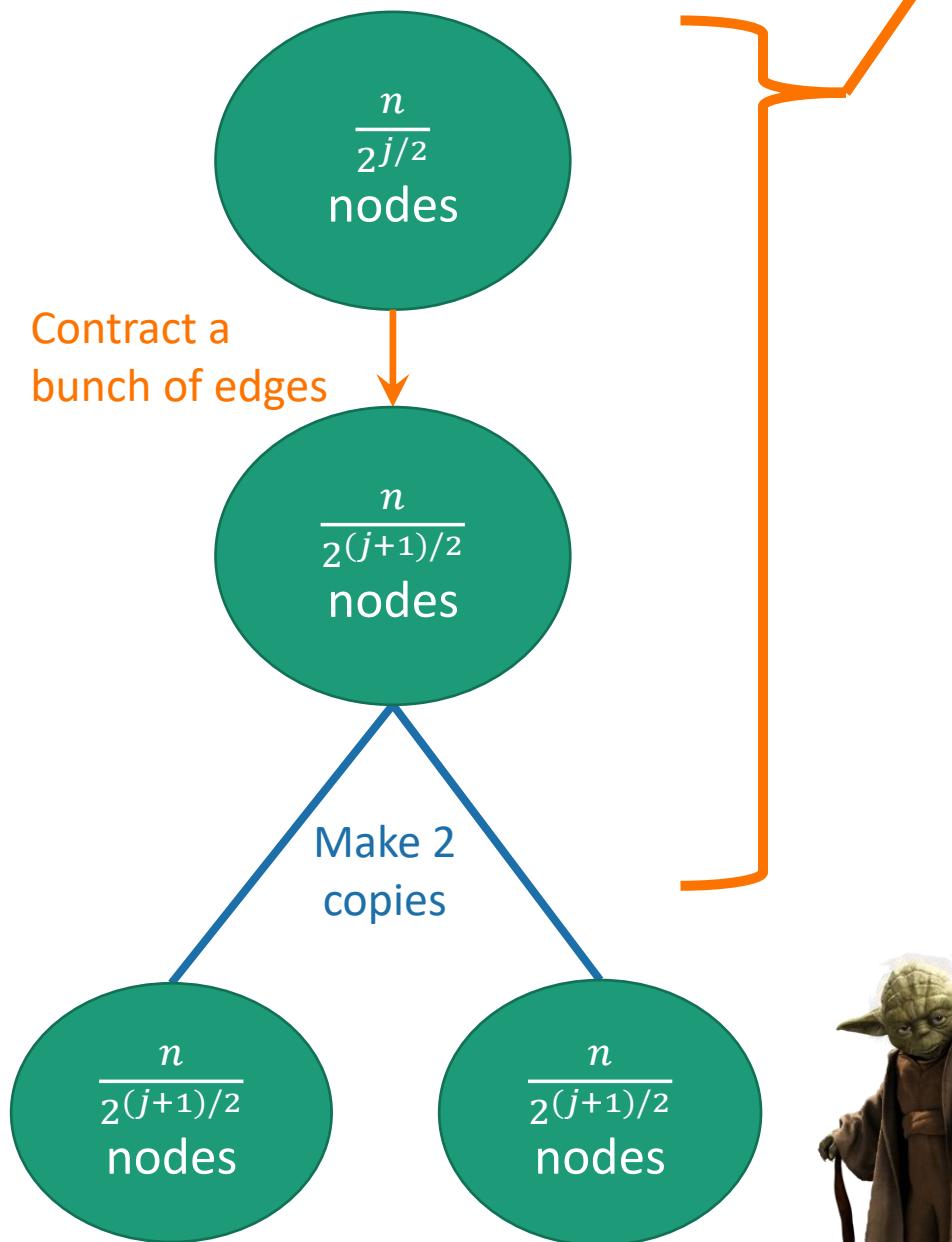
- The amount of work per level is the amount of work needed to reduce the number of nodes by a factor of  $\sqrt{2}$ .
- That's at most  $O(n^2)$ .
  - since that's the time it takes to run Karger's algorithm once, cutting down the number of supernodes to **two**.

# At the $j^{\text{th}}$ level



- The amount of work per level is the amount of work needed to reduce the number of nodes by a factor of  $\sqrt{2}$ .
- That's at most  $O(n^2)$ .
  - since that's the time it takes to run Karger's algorithm once, cutting down the number of supernodes to **two**.
- Our recurrence relation is...
$$T(n) = 2T(n/\sqrt{2}) + O(n^2)$$

# At the $j^{\text{th}}$ level



- The amount of work per level is the amount of work needed to reduce the number of nodes by a factor of  $\sqrt{2}$ .
- That's at most  $O(n^2)$ .
  - since that's the time it takes to run Karger's algorithm once, cutting down the number of supernodes to **two**.
- Our recurrence relation is...
$$T(n) = 2T(n/\sqrt{2}) + O(n^2)$$
- The Master Theorem says...
$$T(n) = O(n^2 \log(n))$$



Jedi Master Yoda

# Two questions

- Does this work?
- Is it fast?
  - Yes,  $O(n^2 \log(n))$ .



# Why $n/\sqrt{2}$ ?

Suppose we contract  $n - t$  edges, until there are  $t$  supernodes remaining.

- Suppose the first  $n-t$  edges that we choose are

$$e_1, e_2, \dots, e_{n-t}$$

# Why $n/\sqrt{2}$ ?

Suppose we contract  $n - t$  edges, until there are  $t$  supernodes remaining.

- Suppose the first  $n-t$  edges that we choose are

$$e_1, e_2, \dots, e_{n-t}$$

- $\text{PR}[\text{ none of } e_1, e_2, \dots, e_{n-t} \text{ cross } S^*]$

$$= \text{PR}[e_1 \text{ doesn't cross } S^*]$$

$$\times \text{PR}[e_2 \text{ doesn't cross } S^* \mid e_1 \text{ doesn't cross } S^*]$$

...

$$\times \text{PR}[e_{n-t} \text{ doesn't cross } S^* \mid e_1, \dots, e_{n-t-1} \text{ don't cross } S^*]$$

# Why $n/\sqrt{2}$ ?

Suppose we contract  $n - t$  edges, until there are  $t$  supernodes remaining.

- Suppose the first  $n-t$  edges that we choose are

$$e_1, e_2, \dots, e_{n-t}$$

- $\text{PR}[\text{ none of } e_1, e_2, \dots, e_{n-t} \text{ cross } S^*]$

$$= \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \left(\frac{n-4}{n-2}\right) \left(\frac{n-5}{n-3}\right) \left(\frac{n-6}{n-4}\right) \dots \left(\frac{t+1}{t+3}\right) \left(\frac{t}{t+2}\right) \left(\frac{t-1}{t+1}\right)$$

# Why $n/\sqrt{2}$ ?

Suppose we contract  $n - t$  edges, until there are  $t$  supernodes remaining.

- Suppose the first  $n-t$  edges that we choose are

$$e_1, e_2, \dots, e_{n-t}$$

- $\text{PR}[\text{ none of } e_1, e_2, \dots, e_{n-t} \text{ cross } S^*]$

$$\begin{aligned}&= \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \left(\frac{n-4}{n-2}\right) \left(\frac{n-5}{n-3}\right) \left(\frac{n-6}{n-4}\right) \dots \left(\frac{t+1}{t+3}\right) \left(\frac{t}{t+2}\right) \left(\frac{t-1}{t+1}\right) \\&= \frac{t \cdot (t-1)}{n \cdot (n-1)}\end{aligned}$$

# Why $n/\sqrt{2}$ ?

Suppose we contract  $n - t$  edges, until there are  $t$  supernodes remaining.

- Suppose the first  $n-t$  edges that we choose are

$$e_1, e_2, \dots, e_{n-t}$$

- $\text{PR}[\text{ none of } e_1, e_2, \dots, e_{n-t} \text{ cross } S^*]$

$$\begin{aligned}&= \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \left(\frac{n-4}{n-2}\right) \left(\frac{n-5}{n-3}\right) \left(\frac{n-6}{n-4}\right) \dots \left(\frac{t+1}{t+3}\right) \left(\frac{t}{t+2}\right) \left(\frac{t-1}{t+1}\right) \\&= \frac{t \cdot (t-1)}{n \cdot (n-1)} \quad \text{Choose } t = n/\sqrt{2}\end{aligned}$$

# Why $n/\sqrt{2}$ ?

Suppose we contract  $n - t$  edges, until there are  $t$  supernodes remaining.

- Suppose the first  $n-t$  edges that we choose are

$$e_1, e_2, \dots, e_{n-t}$$

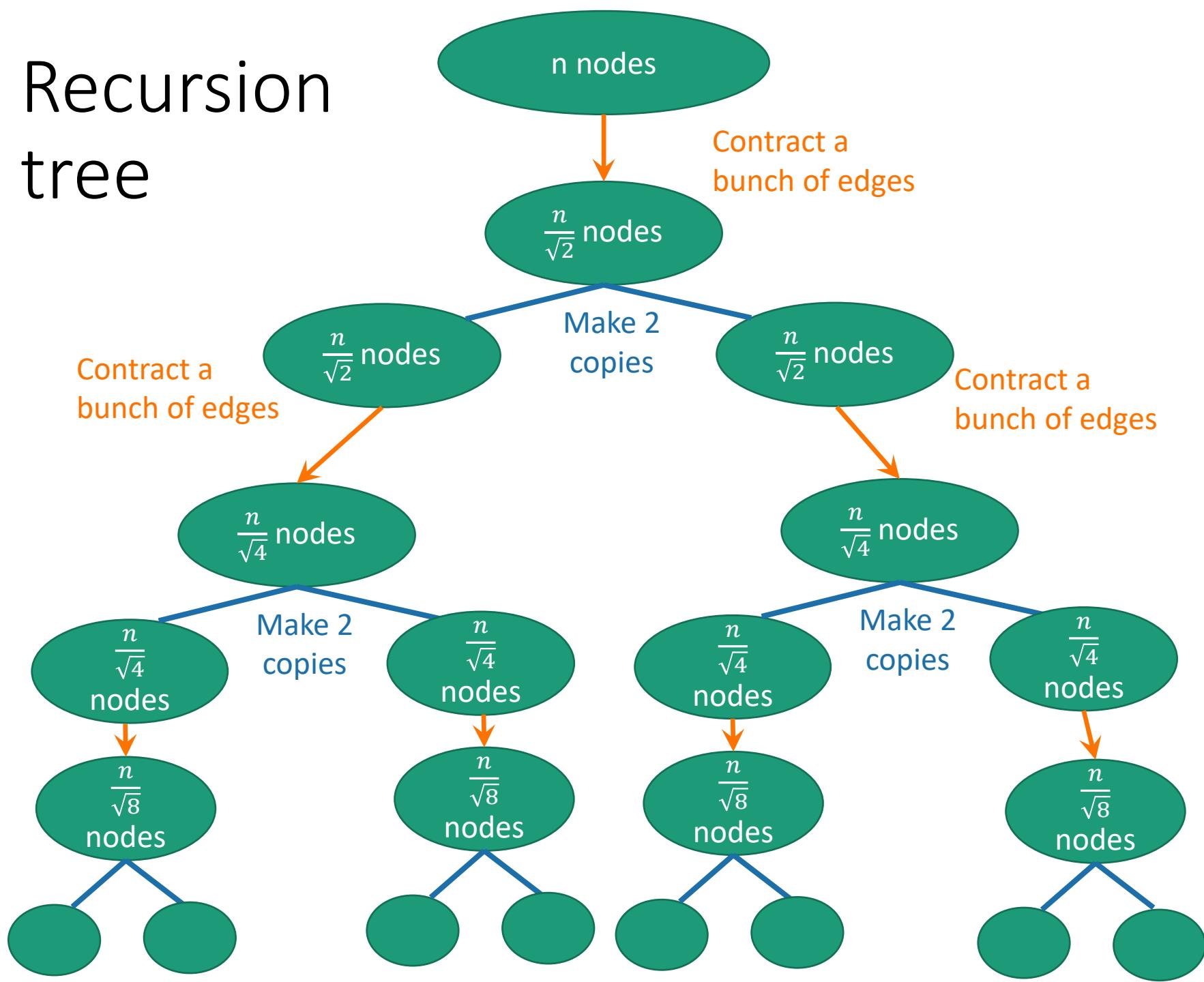
- $\text{PR}[\text{ none of } e_1, e_2, \dots, e_{n-t} \text{ cross } S^*]$

$$= \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \left(\frac{n-4}{n-2}\right) \left(\frac{n-5}{n-3}\right) \left(\frac{n-6}{n-4}\right) \dots \left(\frac{t+1}{t+3}\right) \left(\frac{t}{t+2}\right) \left(\frac{t-1}{t+1}\right)$$

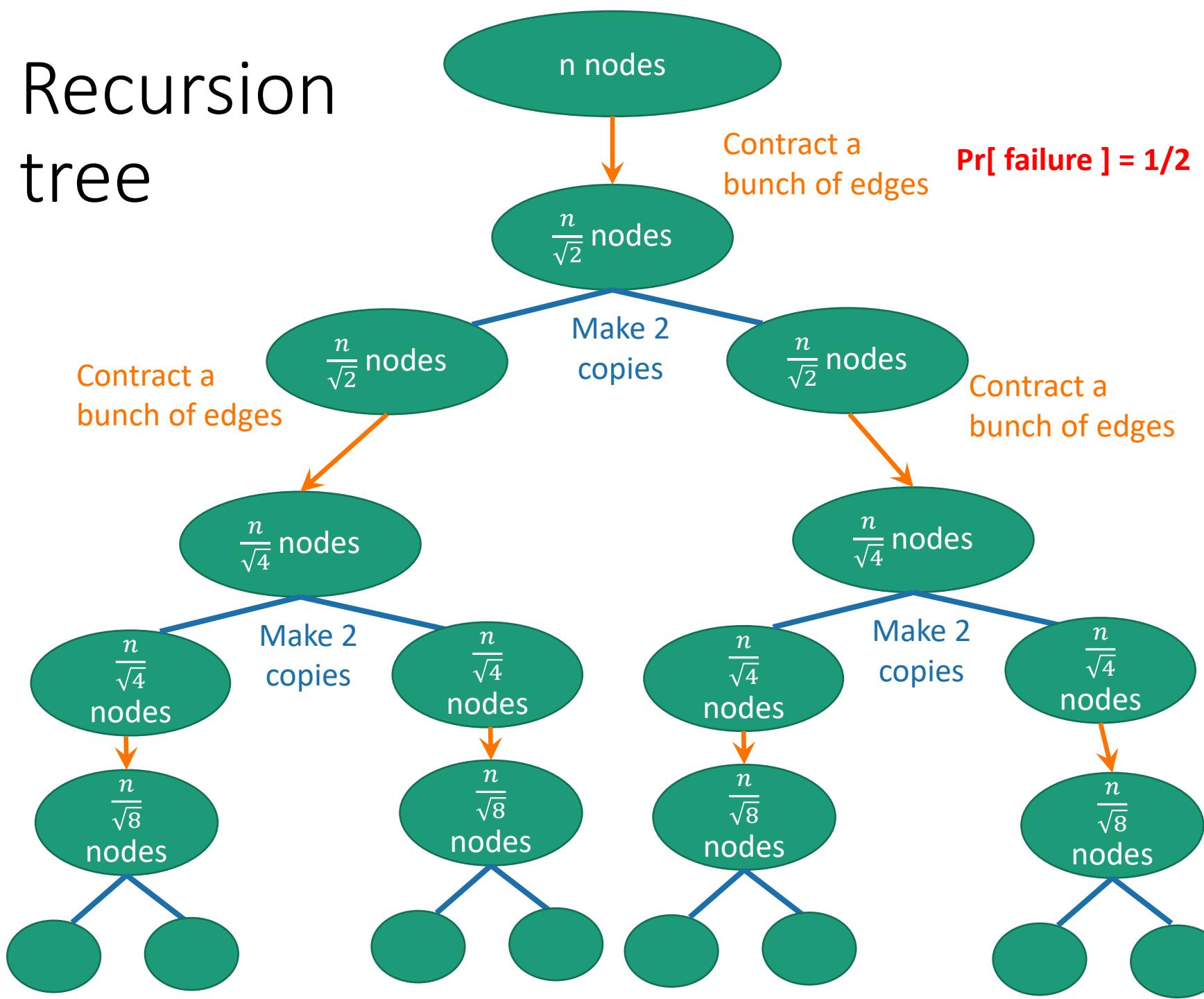
$$= \frac{t \cdot (t-1)}{n \cdot (n-1)} \quad \text{Choose } t = n/\sqrt{2}$$

$$= \frac{\frac{n}{\sqrt{2}} \cdot \left(\frac{n}{\sqrt{2}} - 1\right)}{n \cdot (n-1)} \approx \frac{1}{2} \quad \text{when } n \text{ is large}$$

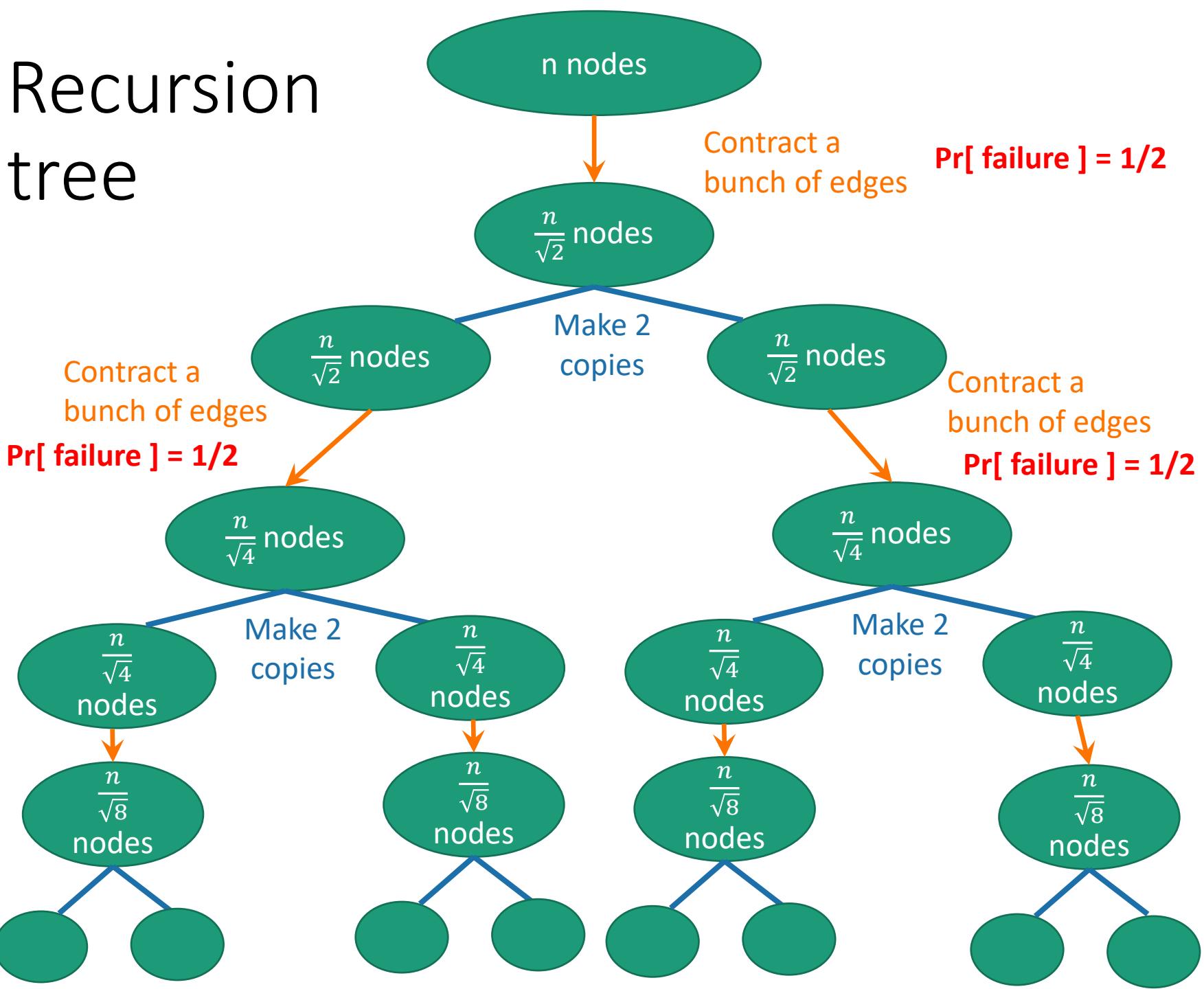
# Recursion tree



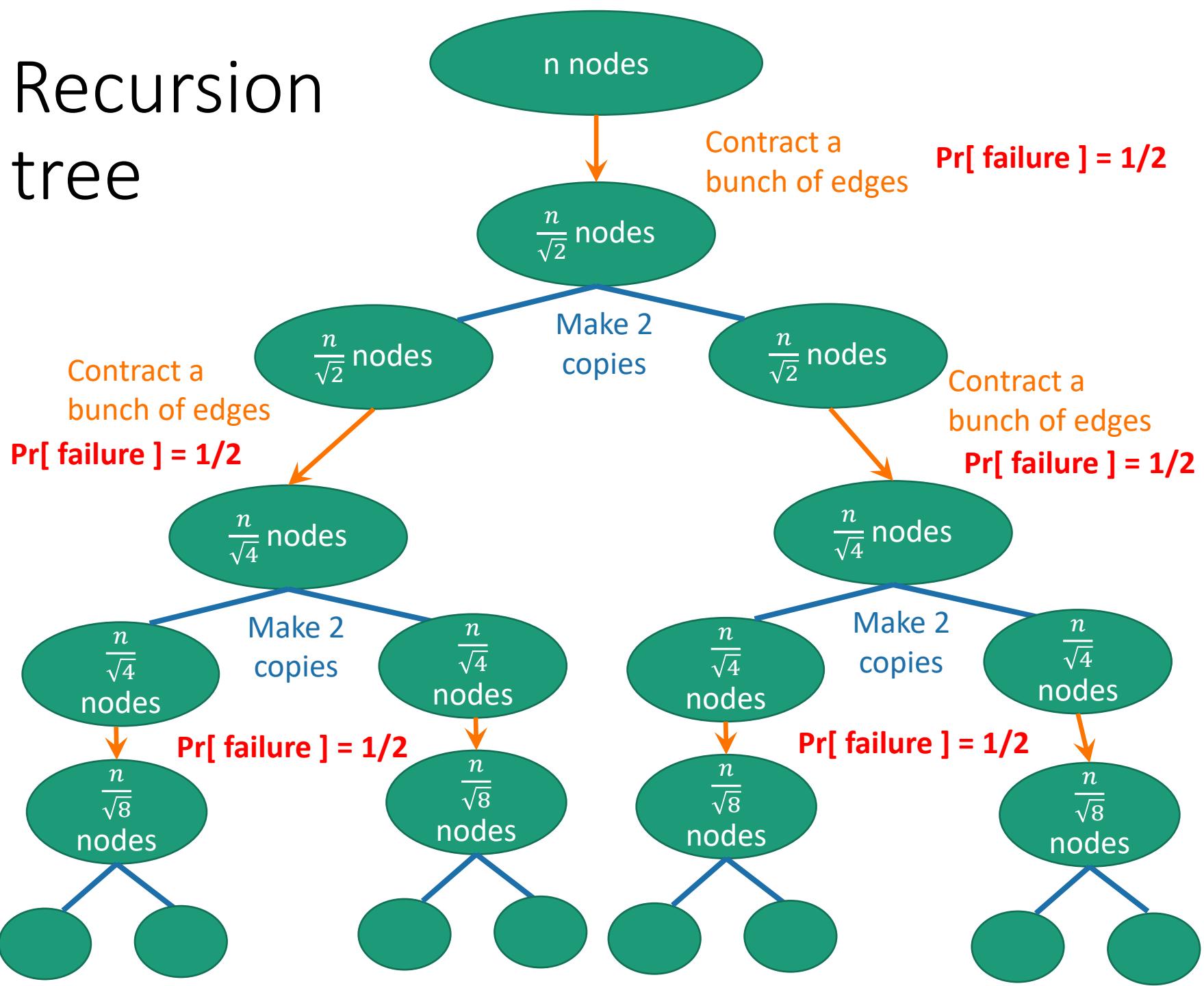
# Recursion tree



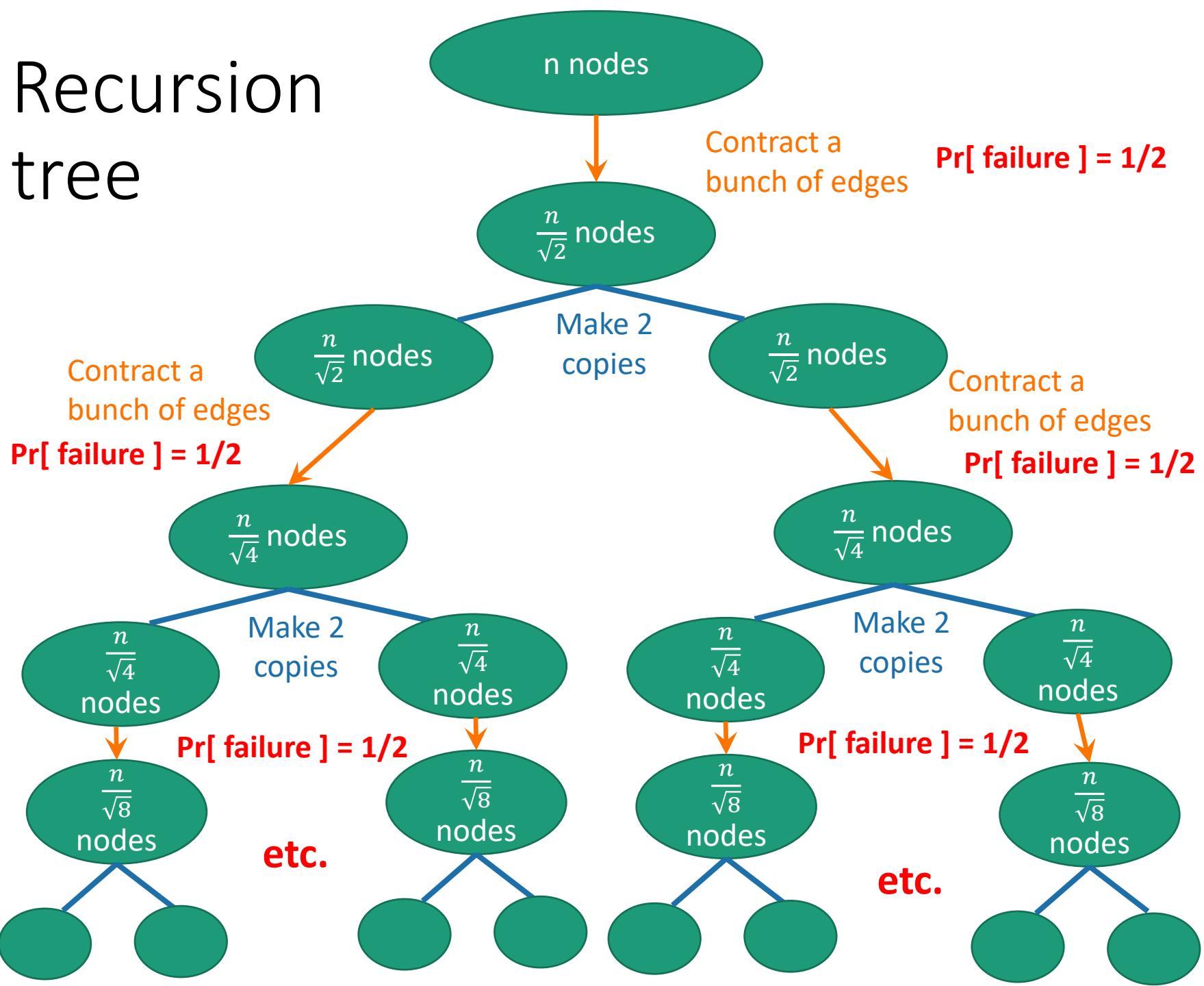
# Recursion tree



# Recursion tree

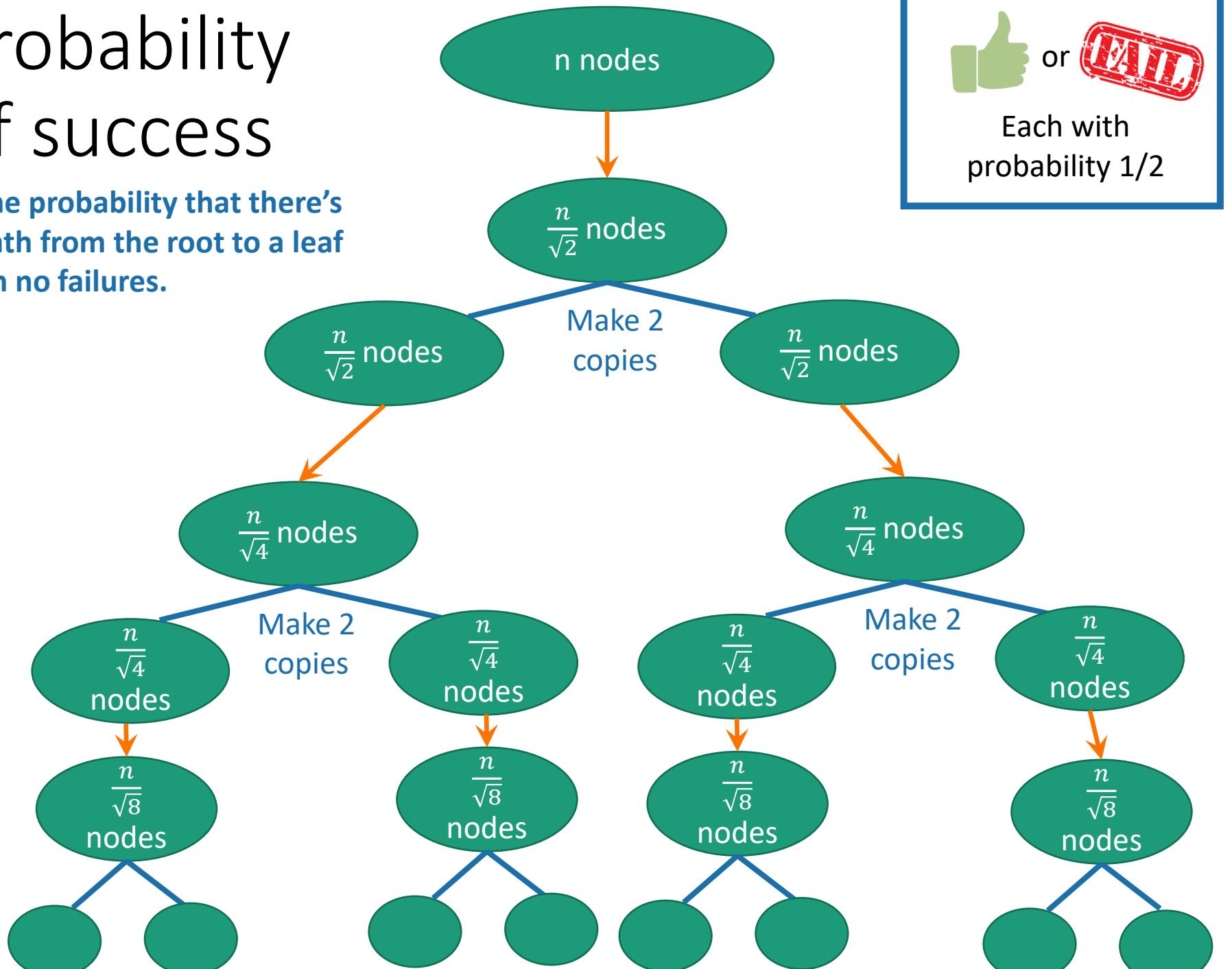


# Recursion tree



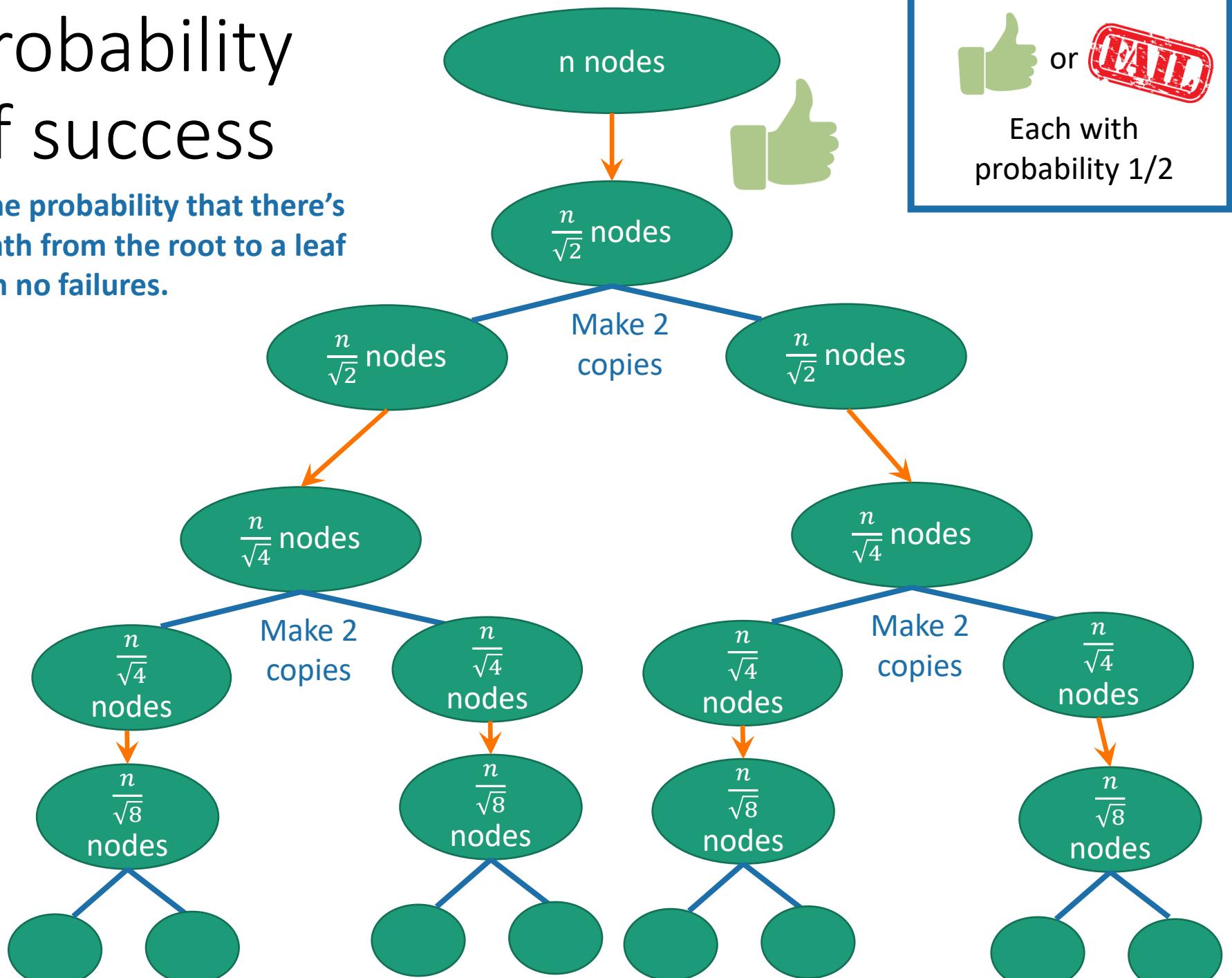
# Probability of success

Is the probability that there's a path from the root to a leaf with no failures.



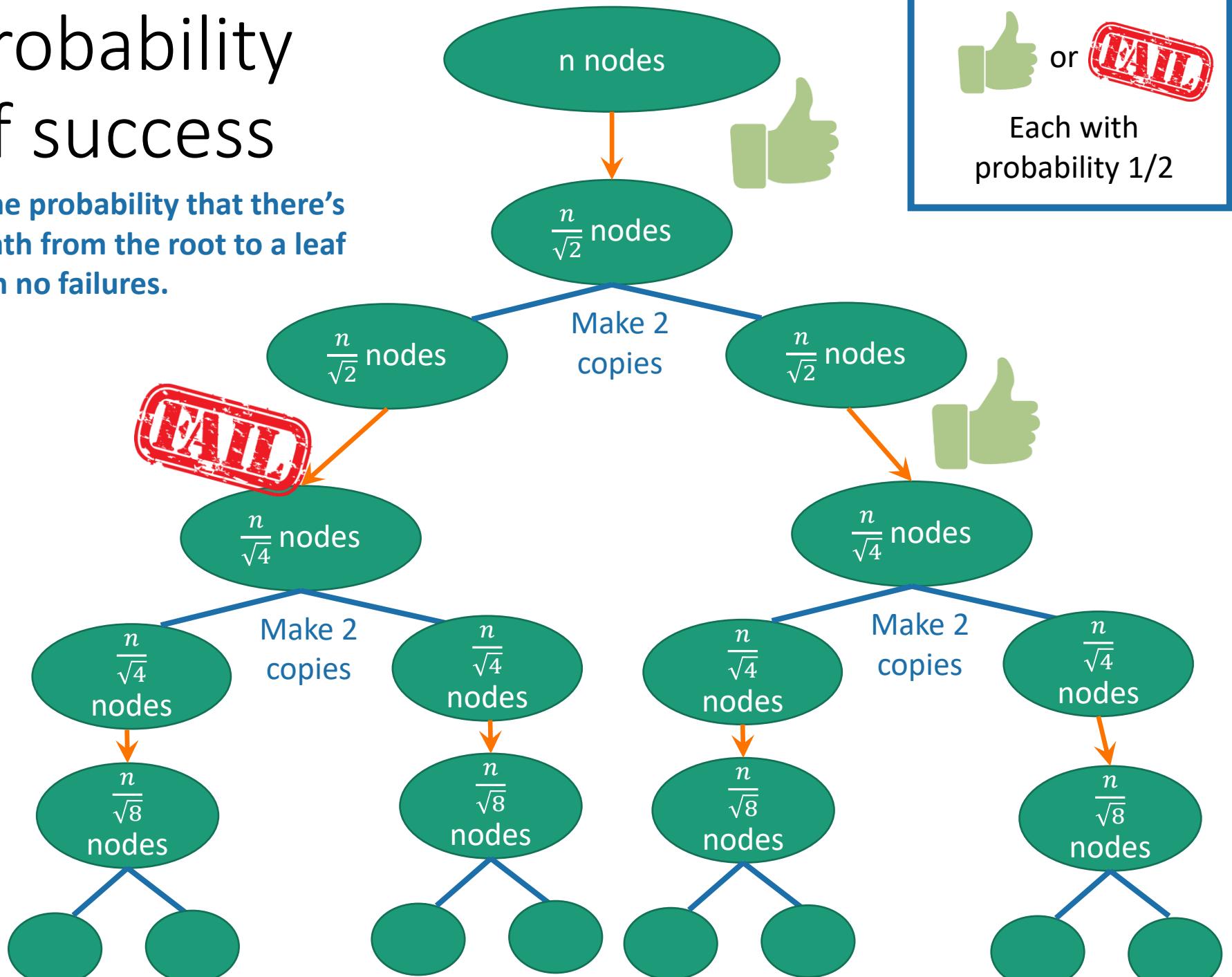
# Probability of success

Is the probability that there's a path from the root to a leaf with no failures.



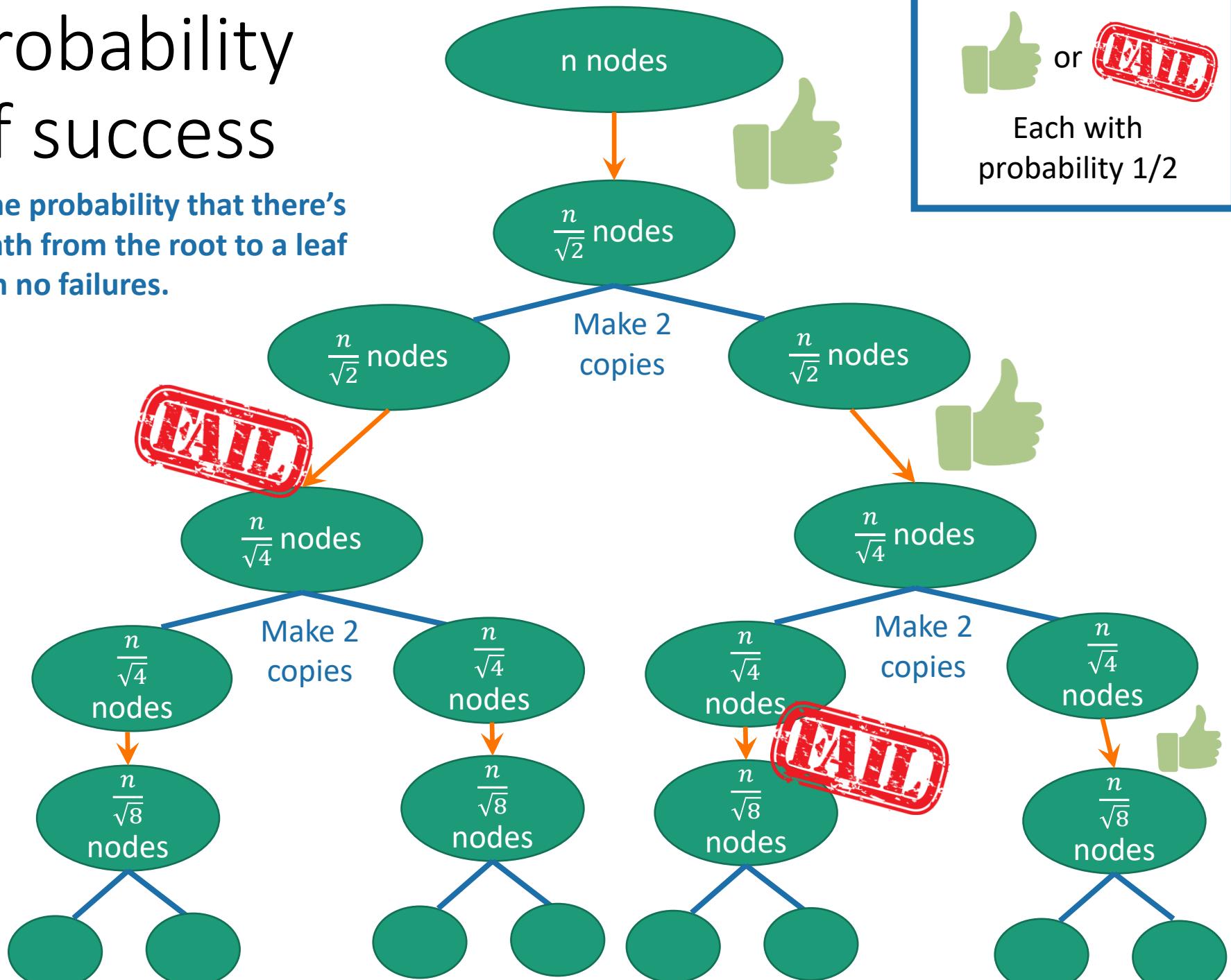
# Probability of success

Is the probability that there's a path from the root to a leaf with no failures.



# Probability of success

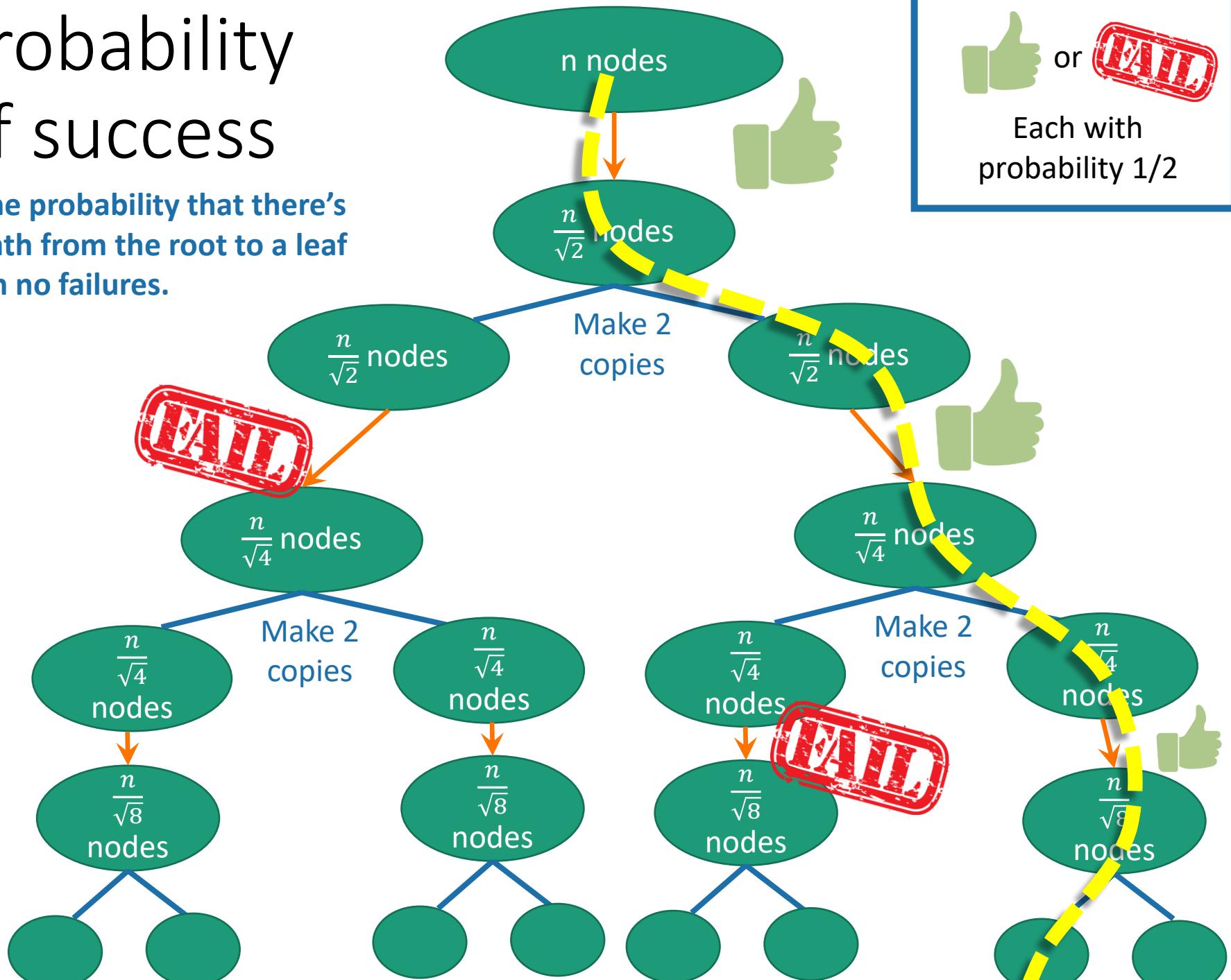
Is the probability that there's a path from the root to a leaf with no failures.



or  
Each with probability 1/2

# Probability of success

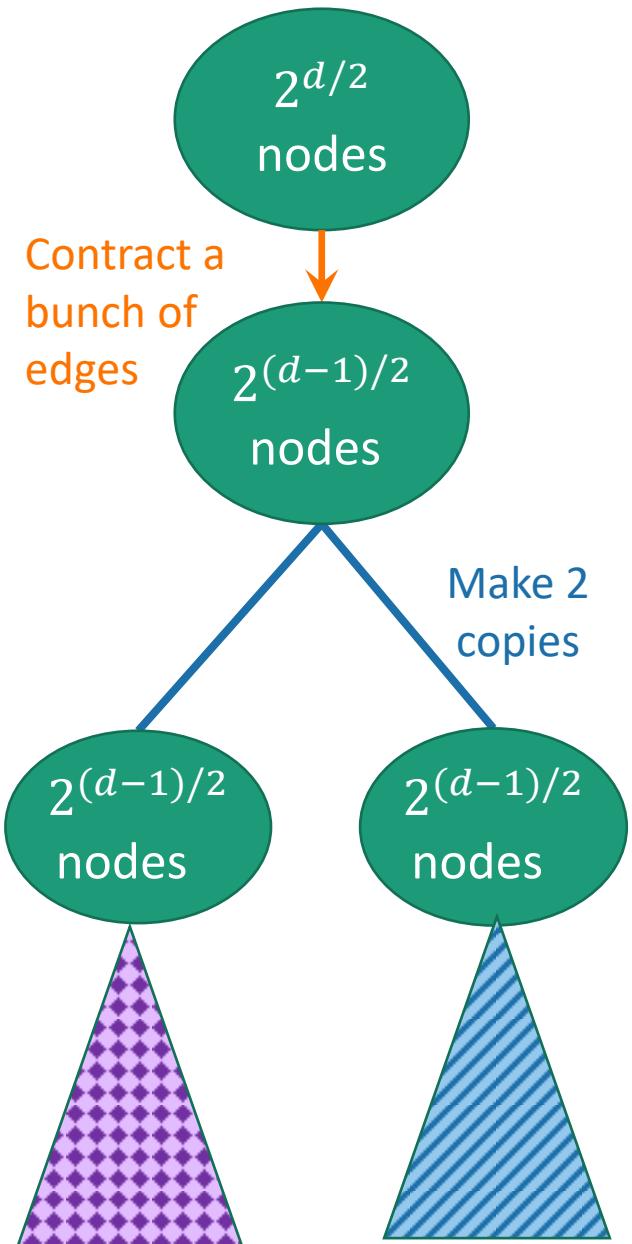
Is the probability that there's a path from the root to a leaf with no failures.



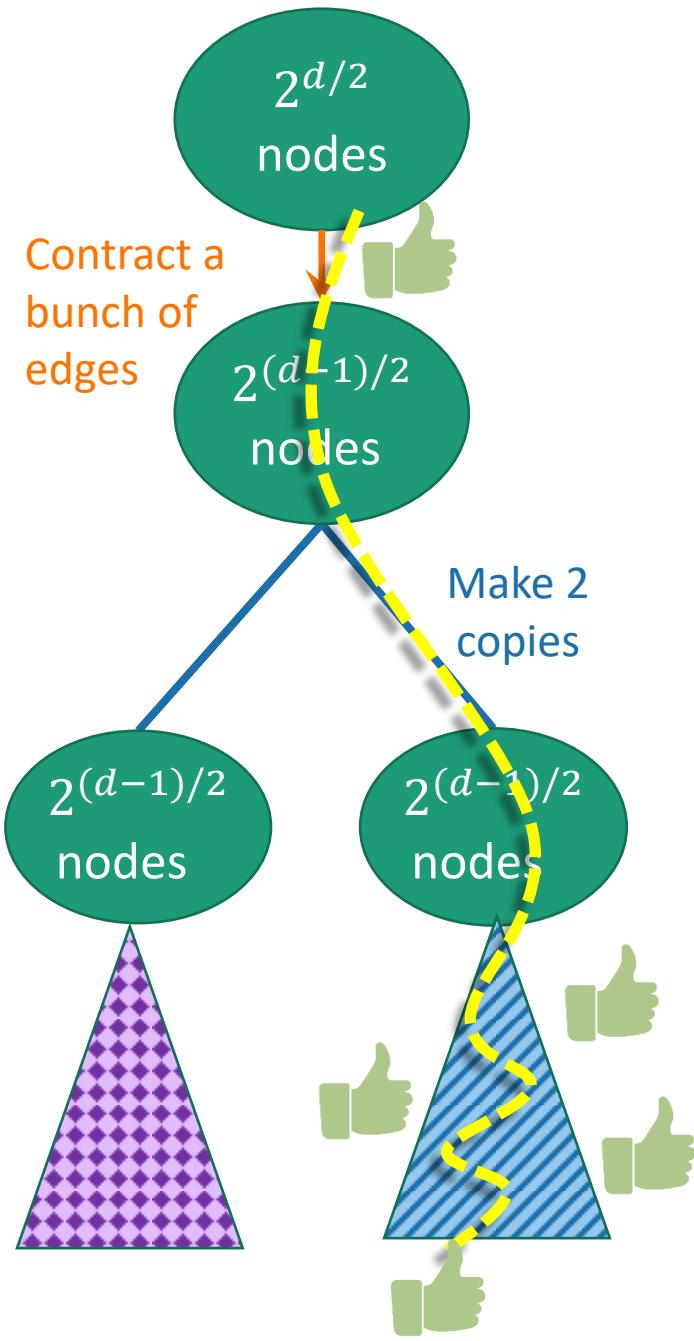
# The problem we need to analyze

- Let  $T$  be binary tree of depth  $2\log(n)$
- Each node of  $T$  succeeds or fails independently with probability  $1/2$
- What is the probability that there's a path from the root to any leaf that's entirely successful?

# Analysis

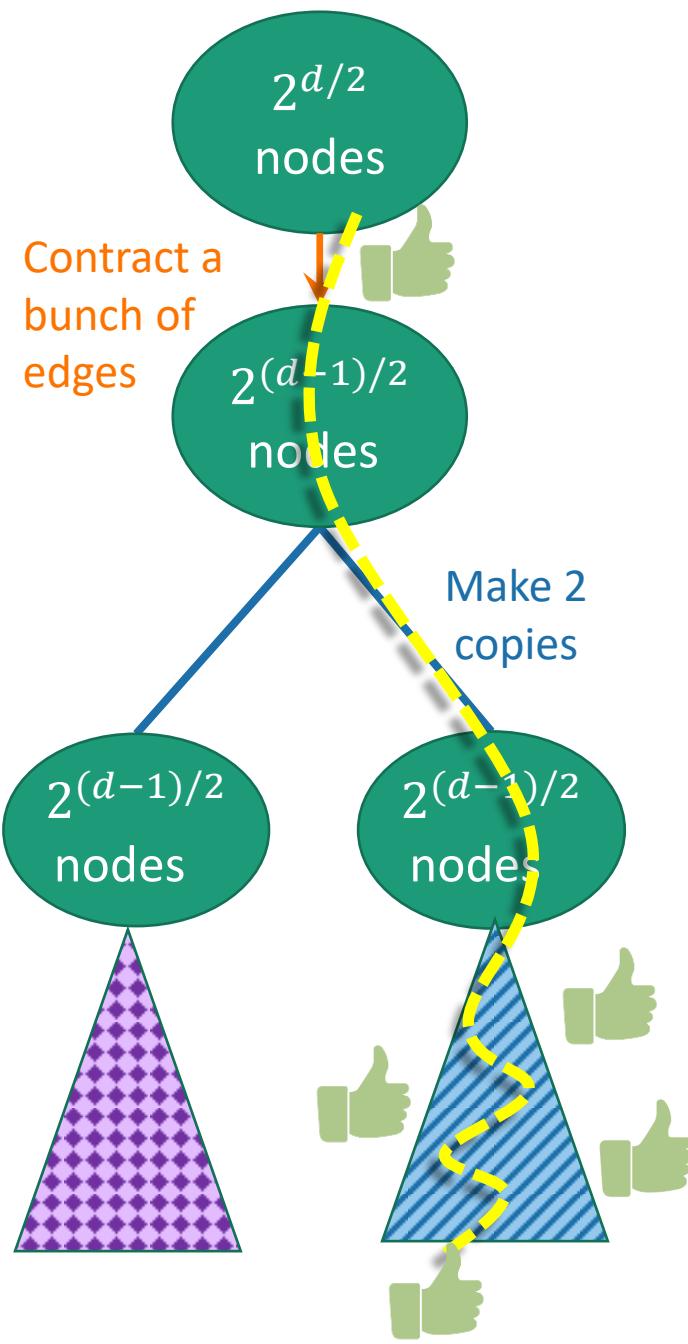


# Analysis



# Analysis

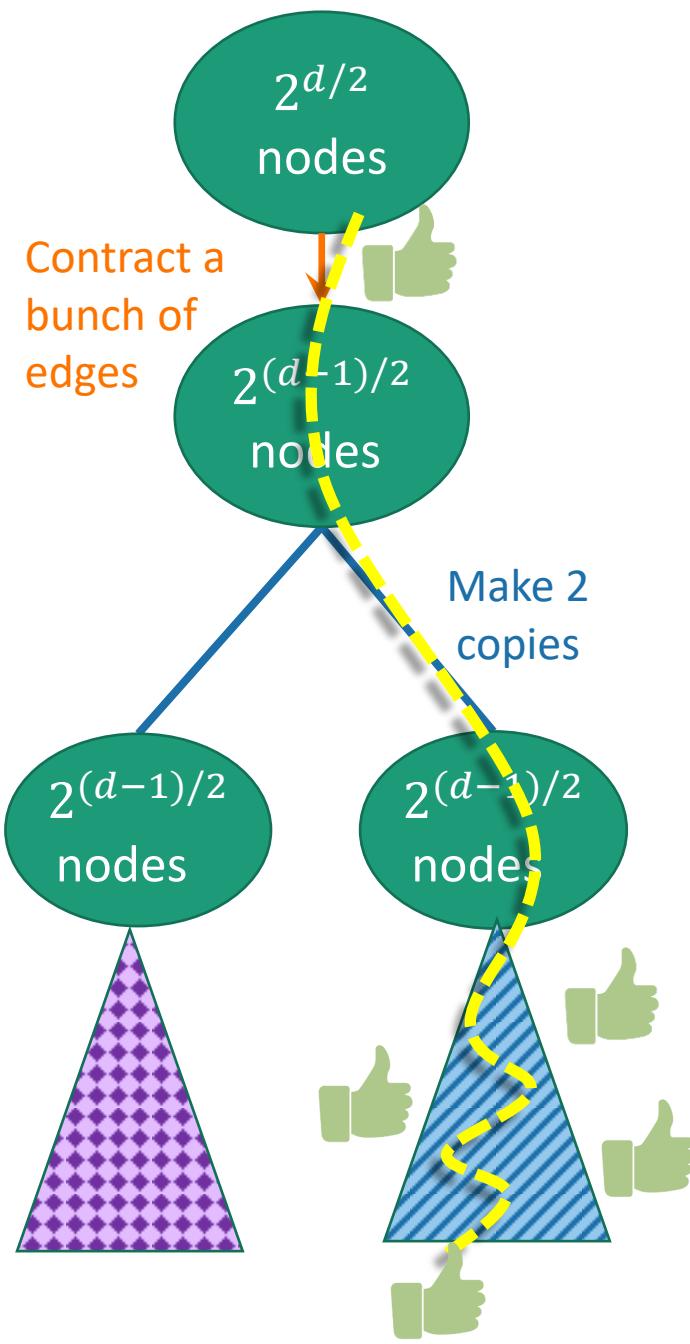
- Say the tree has height  $d$ .
- Let  $p_d$  be the probability that there's a path from the root to a leaf that **doesn't fail**.



# Analysis

- Say the tree has height  $d$ .
- Let  $p_d$  be the probability that there's a path from the root to a leaf that **doesn't fail**.

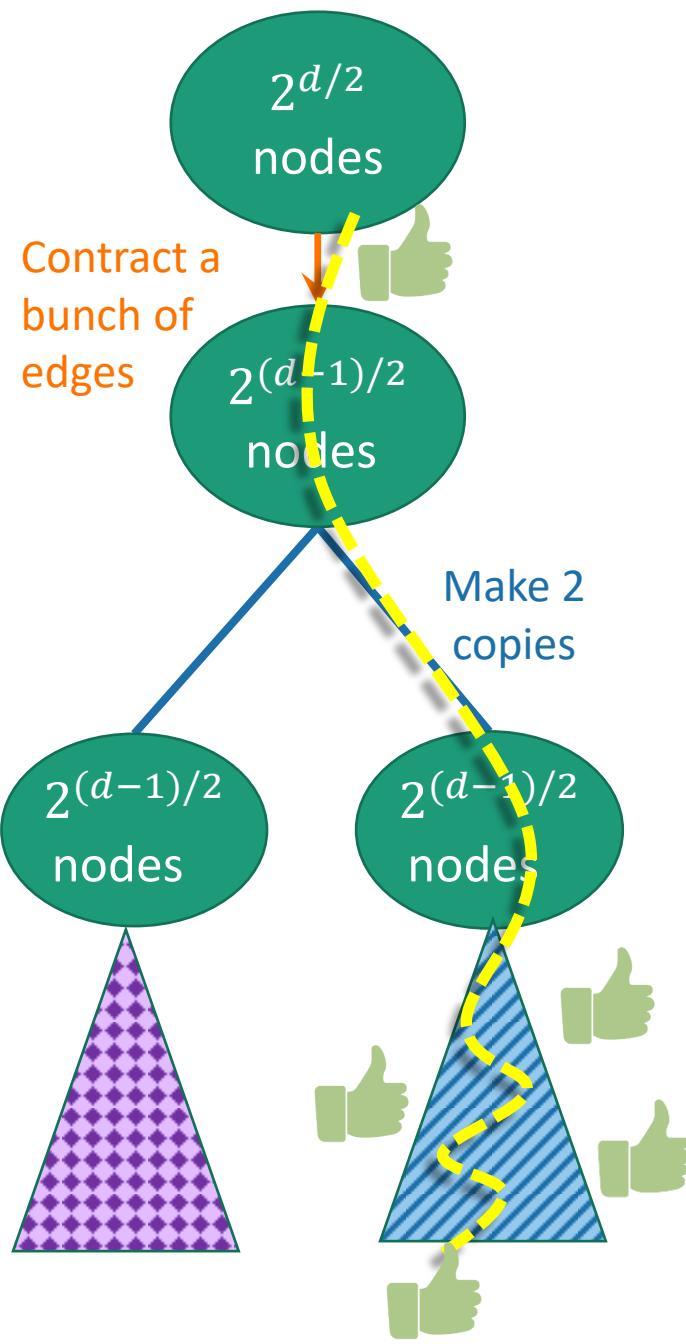
$$p_d = \frac{1}{2} \cdot \Pr \left[ \text{at least one subtree has a successful path} \right]$$



# Analysis

- Say the tree has height  $d$ .
- Let  $p_d$  be the probability that there's a path from the root to a leaf that **doesn't fail**.

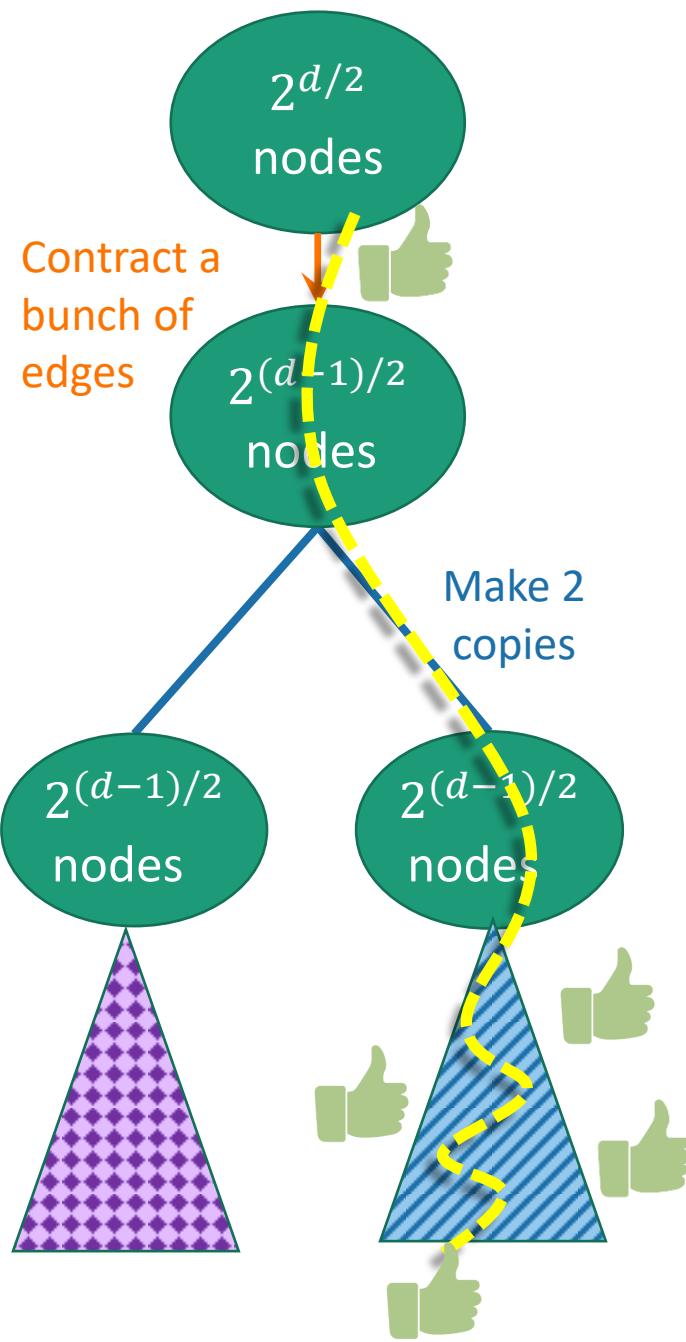
$$p_d = \frac{1}{2} \cdot \Pr \left[ \text{at least one subtree has a successful path} \right]$$
$$= \frac{1}{2} \cdot \left( \Pr \left[ \begin{array}{c} \text{wins} \\ \text{purple triangle} \end{array} \right] + \Pr \left[ \begin{array}{c} \text{wins} \\ \text{blue triangle} \end{array} \right] - \Pr \left[ \begin{array}{c} \text{both win} \\ \text{purple and blue triangles} \end{array} \right] \right)$$



# Analysis

- Say the tree has height  $d$ .
- Let  $p_d$  be the probability that there's a path from the root to a leaf that **doesn't fail**.

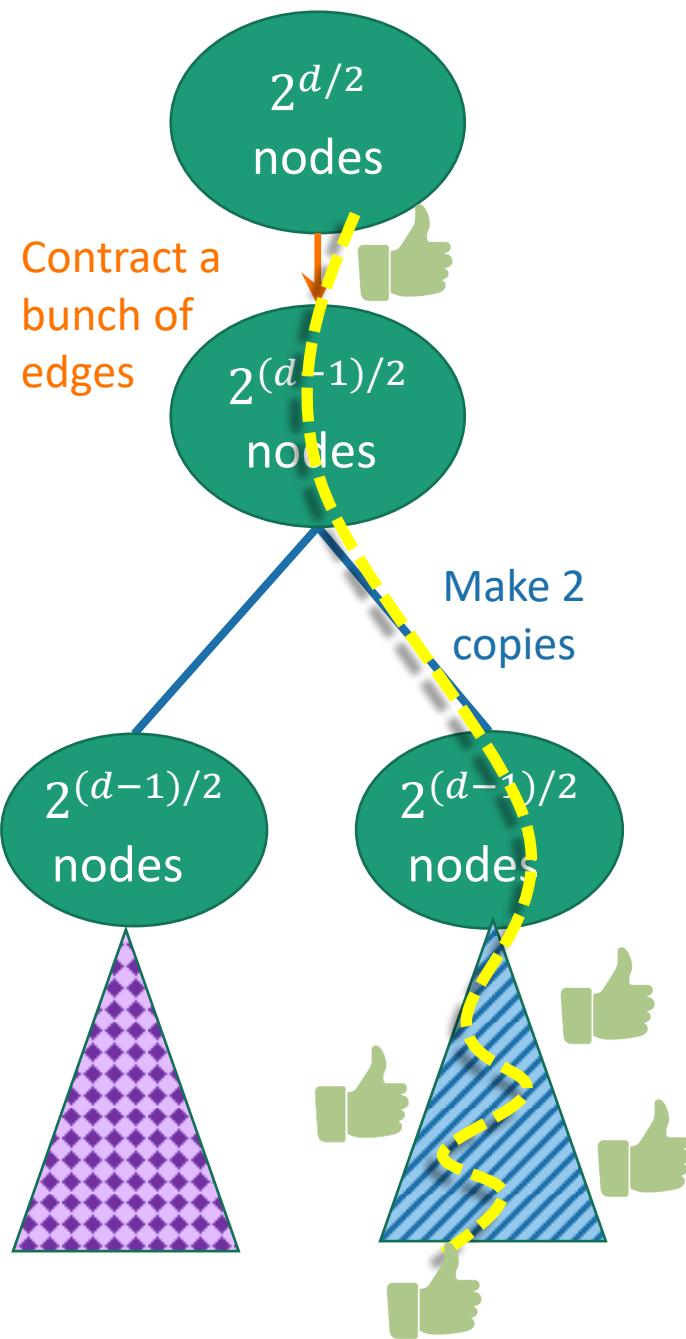
$$\begin{aligned} p_d &= \frac{1}{2} \cdot \Pr \left[ \text{at least one subtree has a successful path} \right] \\ &= \frac{1}{2} \cdot \left( \Pr \left[ \begin{array}{c} \text{wins} \\ \text{purple triangle} \end{array} \right] + \Pr \left[ \begin{array}{c} \text{wins} \\ \text{blue triangle} \end{array} \right] - \Pr \left[ \begin{array}{c} \text{both win} \\ \text{purple and blue triangles} \end{array} \right] \right) \\ &= \frac{1}{2} \cdot (p_{d-1} + p_{d-1} - p_{d-1}^2) \end{aligned}$$



# Analysis

- Say the tree has height  $d$ .
- Let  $p_d$  be the probability that there's a path from the root to a leaf that **doesn't fail**.

$$\begin{aligned}
 p_d &= \frac{1}{2} \cdot \Pr \left[ \text{at least one subtree has a successful path} \right] \\
 &= \frac{1}{2} \cdot \left( \Pr \left[ \begin{array}{c} \text{wins} \\ \text{purple triangle} \end{array} \right] + \Pr \left[ \begin{array}{c} \text{wins} \\ \text{blue triangle} \end{array} \right] - \Pr \left[ \begin{array}{c} \text{both win} \\ \text{purple and blue triangles} \end{array} \right] \right) \\
 &= \frac{1}{2} \cdot (p_{d-1} + p_{d-1} - p_{d-1}^2) \\
 &= p_{d-1} - \frac{1}{2} \cdot p_{d-1}^2
 \end{aligned}$$



# It's a recurrence relation!

- $p_d = p_{d-1} - \frac{1}{2} \cdot p_{d-1}^2$
- $p_0 = 1$
- We are really good at those.

# It's a recurrence relation!

- $p_d = p_{d-1} - \frac{1}{2} \cdot p_{d-1}^2$
- $p_0 = 1$
- We are really good at those.
- In this case, the answer is:
  - **Claim:** for all  $d$ ,  $p_d \geq \frac{1}{d+1}$

# Recurrence relation

- $p_d = p_{d-1} - \frac{1}{2} \cdot p_{d-1}^2$
- $p_0 = 1$

- **Claim:** for all  $d$ ,  $p_d \geq \frac{1}{d+1}$
- **Proof:** induction on  $d$ .
  - **Base case:**  $1 \geq 1$ . YEP.
  - **Inductive step:** say  $d > 0$ .
    - Suppose that  $p_{d-1} \geq \frac{1}{d}$ .
    - $p_d = p_{d-1} - \frac{1}{2} \cdot p_{d-1}^2$
    - $\geq \frac{1}{d} - \frac{1}{2} \cdot \frac{1}{d^2}$
    - $\geq \frac{1}{d} - \frac{1}{d(d+1)}$
    - $= \frac{1}{d+1}$

# What does that mean for Karger-Stein?

**Claim:** for all  $d$ ,  $p_d \geq \frac{1}{d+1}$

# What does that mean for Karger-Stein?

**Claim:** for all  $d$ ,  $p_d \geq \frac{1}{d+1}$

- For  $d = 2\log(n)$ 
  - that is,  $d = \text{the height of the tree}:$

$$p_{2\log(n)} \geq \frac{1}{2\log(n) + 1}$$

# What does that mean for Karger-Stein?

**Claim:** for all  $d$ ,  $p_d \geq \frac{1}{d+1}$

- For  $d = 2\log(n)$ 
  - that is,  $d = \text{the height of the tree}:$

$$p_{2\log(n)} \geq \frac{1}{2\log(n) + 1}$$

- aka,

$$\Pr[\text{Karger-Stein is successful}] = \Omega\left(\frac{1}{\log(n)}\right)$$

# Altogether now



- We can do the same trick as before to amplify the success probability.
  - Run Karger-Stein  $O\left(\log(n) \cdot \log\left(\frac{1}{\delta}\right)\right)$  times to achieve success probability  $1 - \delta$ .

# Altogether now



- We can do the same trick as before to amplify the success probability.
  - Run Karger-Stein  $O\left(\log(n) \cdot \log\left(\frac{1}{\delta}\right)\right)$  times to achieve success probability  $1 - \delta$ .
- Each iteration takes time  $O(n^2 \log(n))$ 
  - That's what we proved before.

# Altogether now



- We can do the same trick as before to amplify the success probability.
  - Run Karger-Stein  $O\left(\log(n) \cdot \log\left(\frac{1}{\delta}\right)\right)$  times to achieve success probability  $1 - \delta$ .
- Each iteration takes time  $O(n^2 \log(n))$ 
  - That's what we proved before.
- Choosing  $\delta = 0.01$  as before, the total runtime is  $O(n^2 \log(n) \cdot \log(n)) = O(n^2 \log^2(n))$

Much better than  $O(n^4)$ !

# What have we learned?

- Just repeating Karger's algorithm isn't the best use of repetition.
  - We're probably going to be correct near the beginning.
- Instead, Karger-Stein repeats when it counts.
  - If we wait until there are  $\frac{n}{\sqrt{2}}$  nodes left, the probability that we fail is close to  $\frac{1}{2}$ .
- This lets us (probably) find a global minimum cut in an undirected graph in time  **$O(n^2 \log^2(n))$** .
  - Notice that we can't do better than  $n^2$  in a dense graph (we need to look at all the edges), so this is pretty good.

# Recap

- Some algorithms:
  - Karger's algorithm for global min-cut
  - Improvement: Karger-Stein
- Some concepts:
  - Monte Carlo algorithms:
    - Might be wrong, are always fast.
    - We can boost their success probability with repetition.
    - Sometimes we can do this repetition very cleverly.

# Acknowledgement

- Stanford University