Lecture 16

Min Cut and Karger’s Algorithm

Announcements

- HW 7 due Friday
- HW 8 released Friday
  - Psych! **There is no HW8**.

- **FINAL EXAM**:
  - Wednesday December 13
  - 3:30 – 6:30pm

Advertisement!

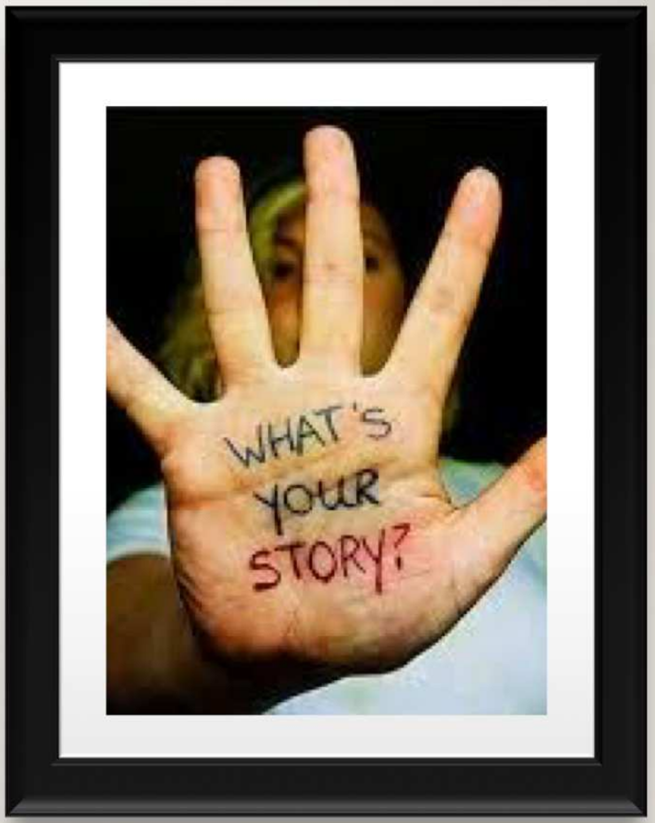
CS83 with Omer Reingold:

Winter quarter
Fri 1:30 PM - 4:20 PM

cs83.stanford.edu

CS 83 - PLAYBACK THEATER FOR RESEARCH

A FEEL GOOD COURSE



Last time

- Minimum Spanning Trees!
    - Prim's Algorithm
    - Kruskal's Algorithm

Today

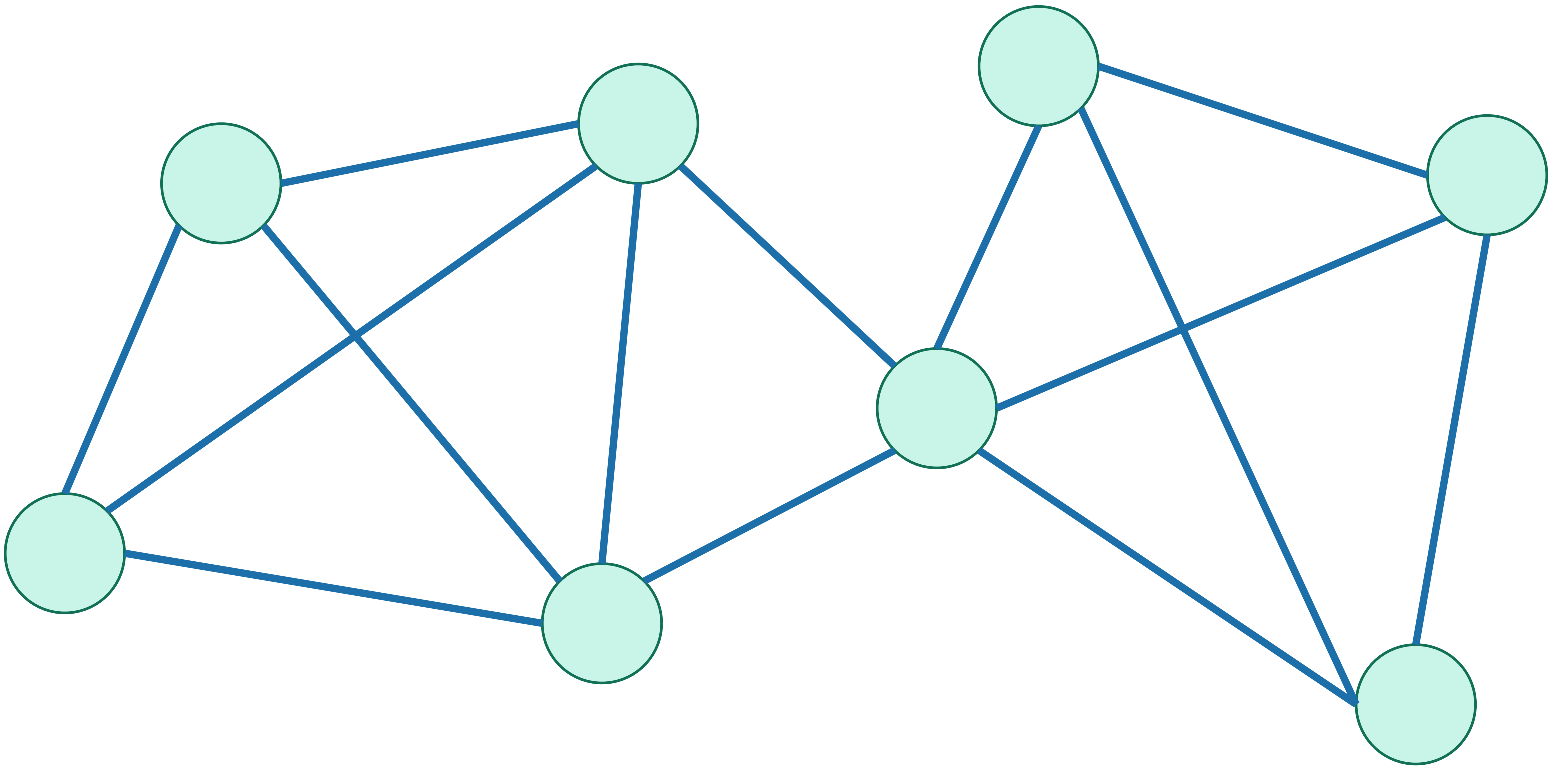
- Minimum Cuts!
  - Karger's algorithm
  - Karger-Stein algorithm

  - Back to **randomized algorithms!**

Recall: cuts in graphs

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

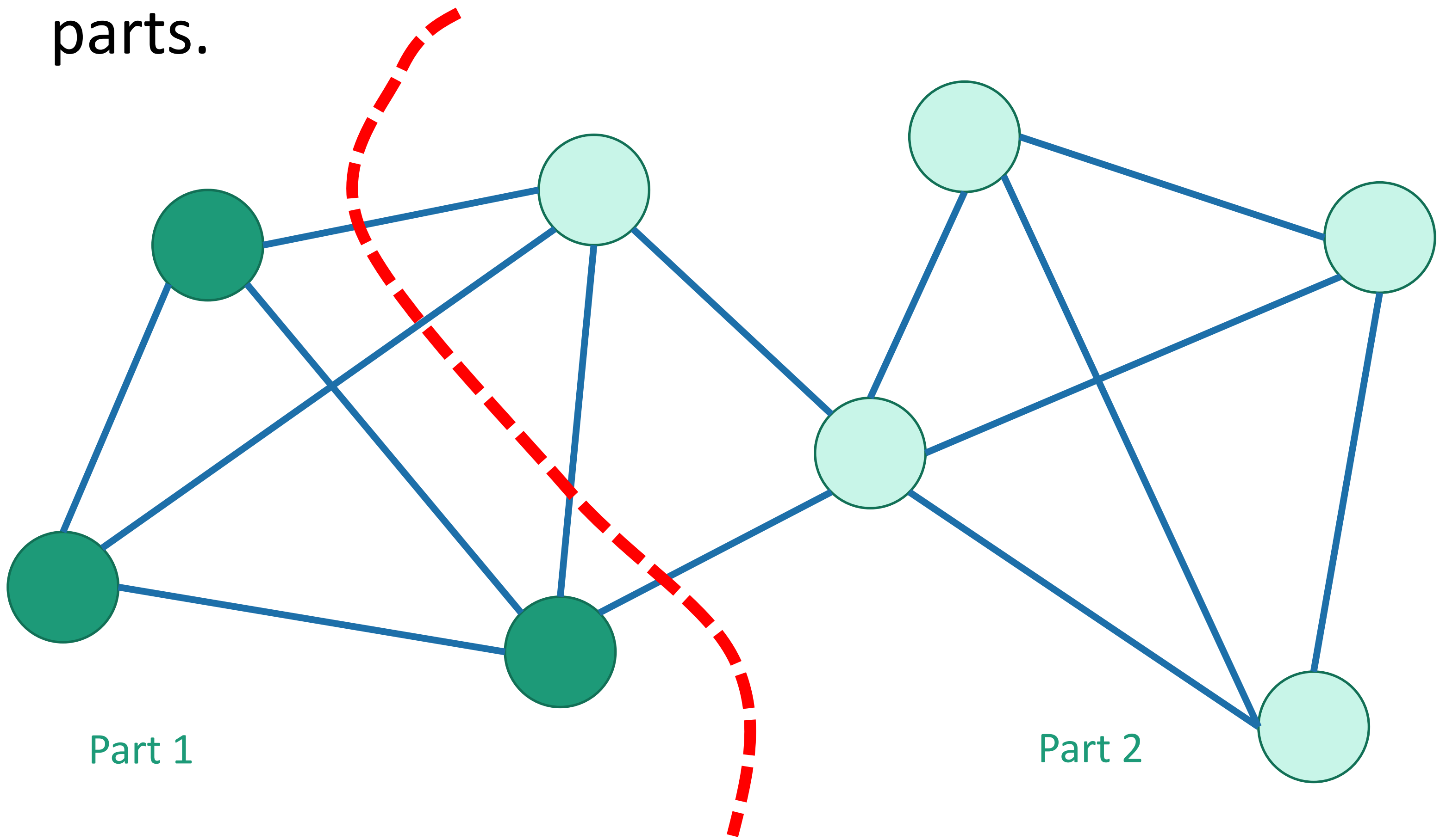
- A cut is a partition of the vertices into two nonempty parts.



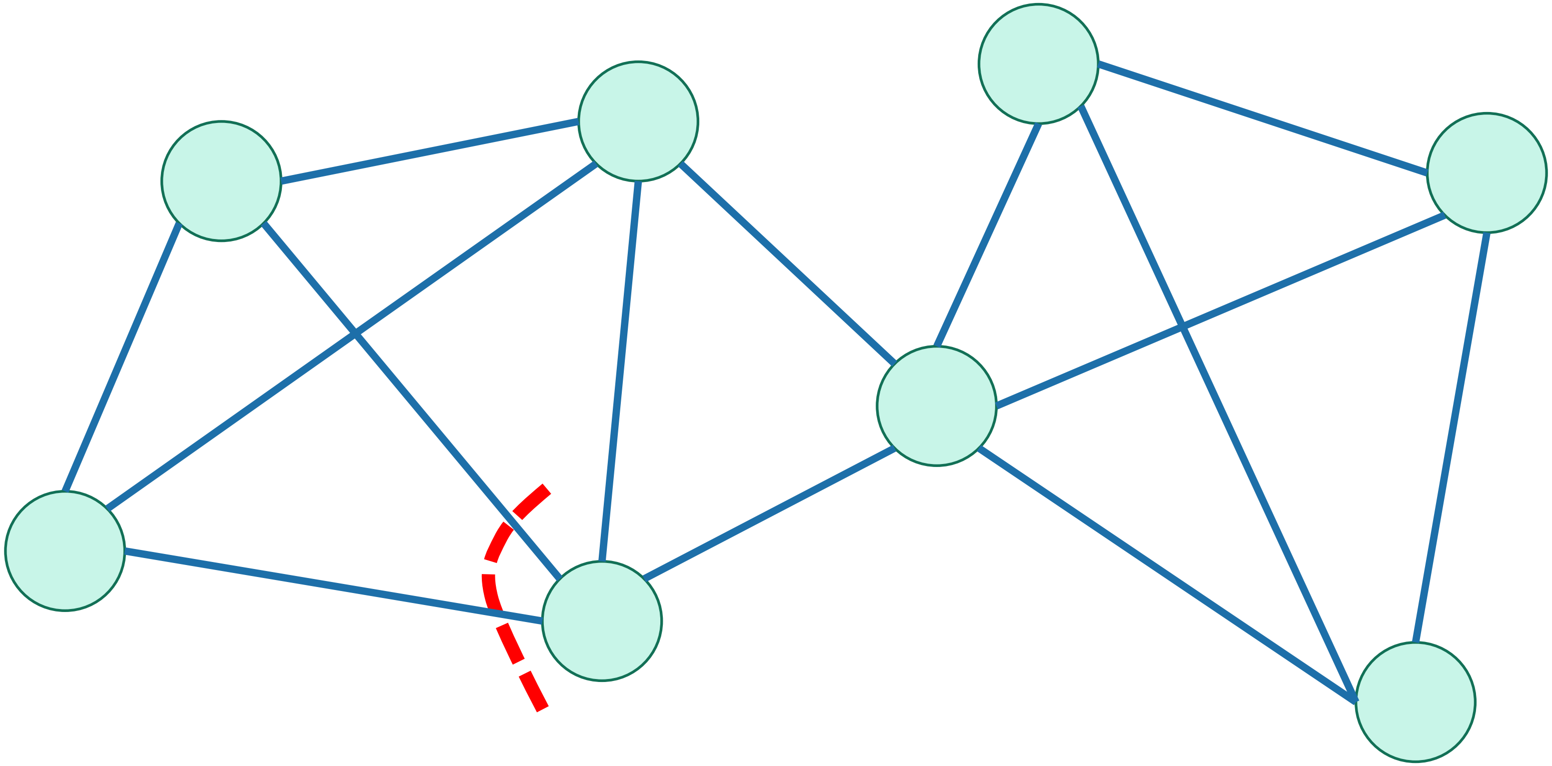
Recall: cuts in graphs

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

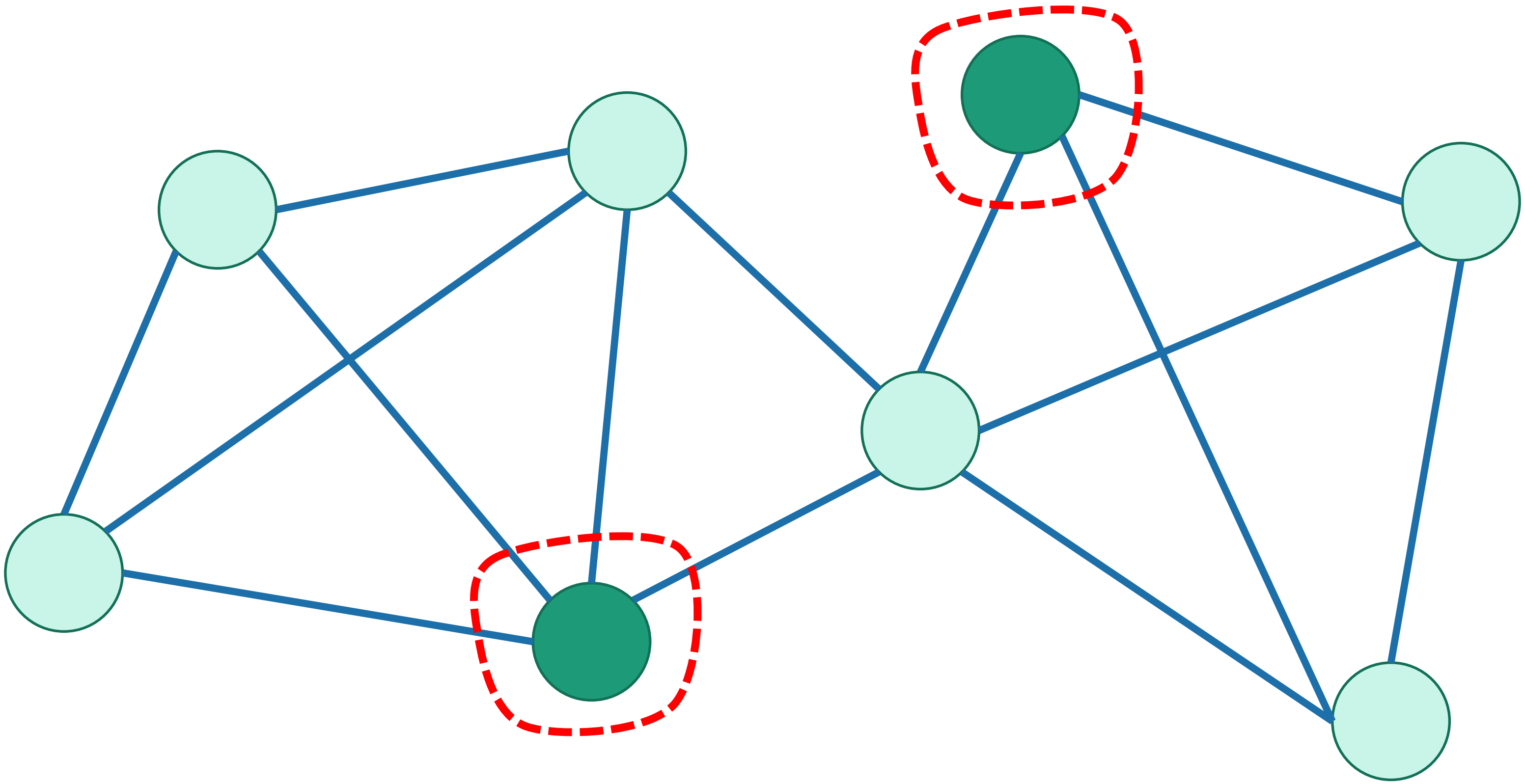
- A cut is a partition of the vertices into two nonempty parts.



This is not a cut



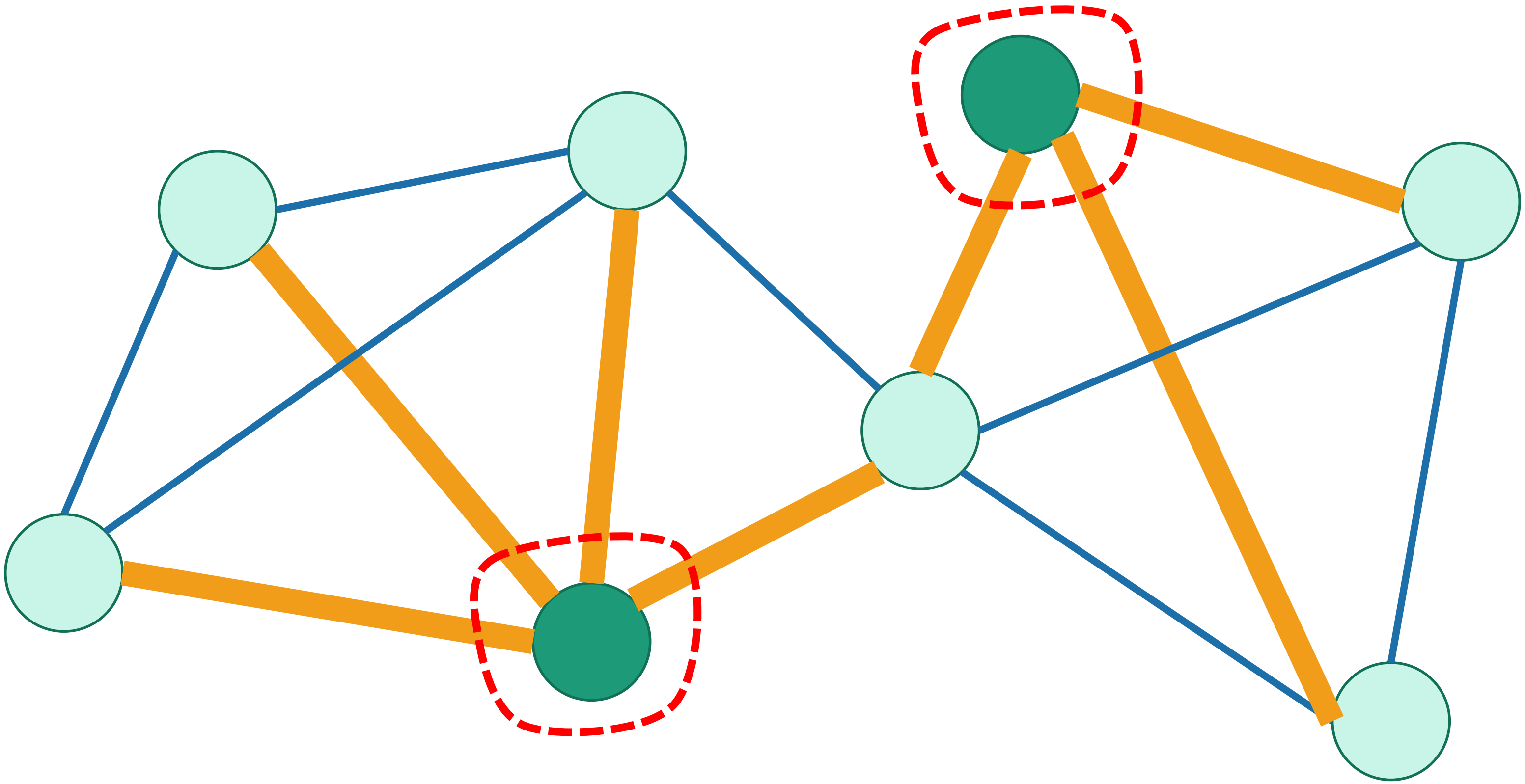
This is a cut



This is a cut

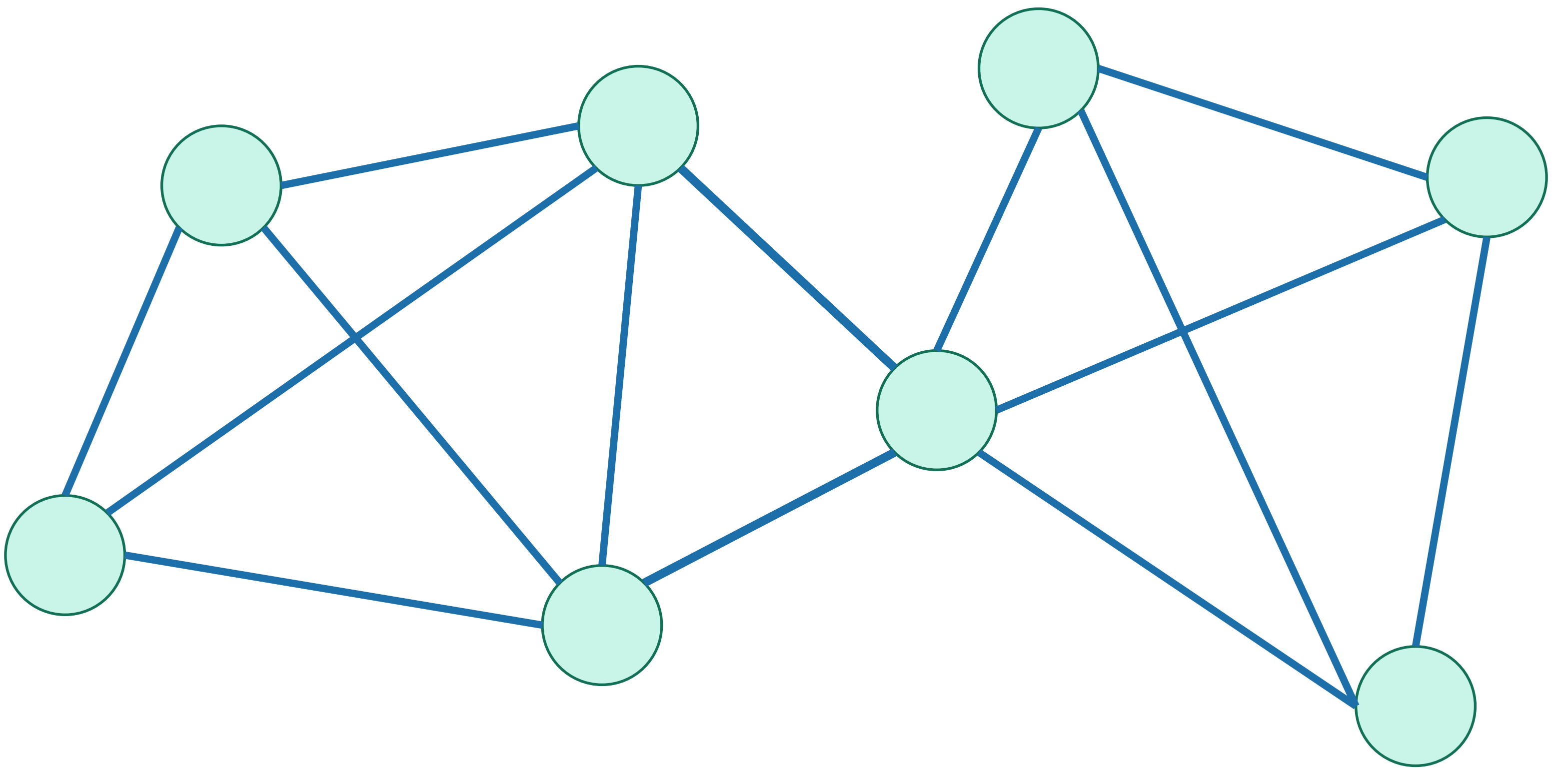
These edges **cross the cut.**

- They go from one part to the other.



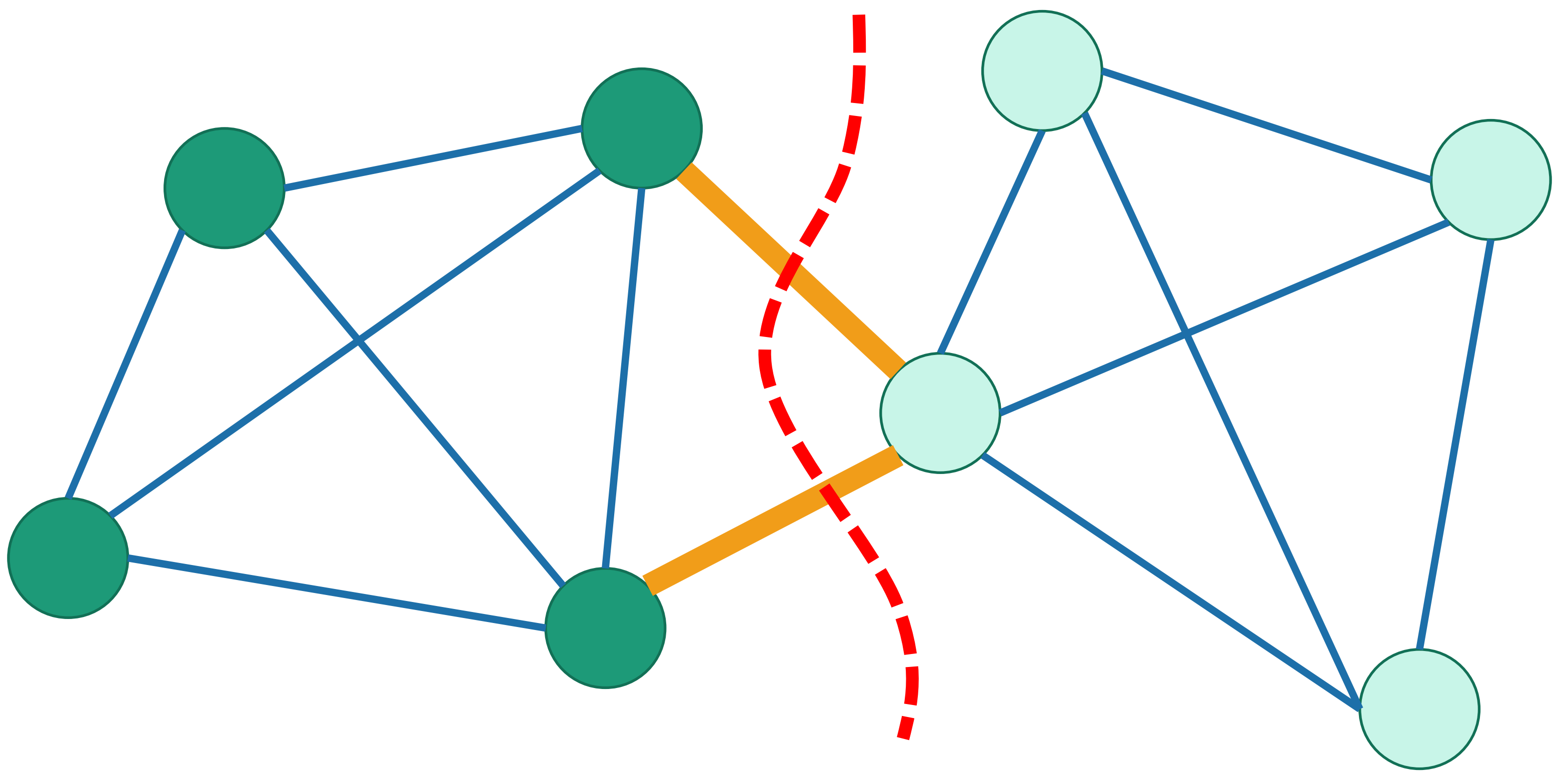
A (global) minimum cut

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



A (global) minimum cut

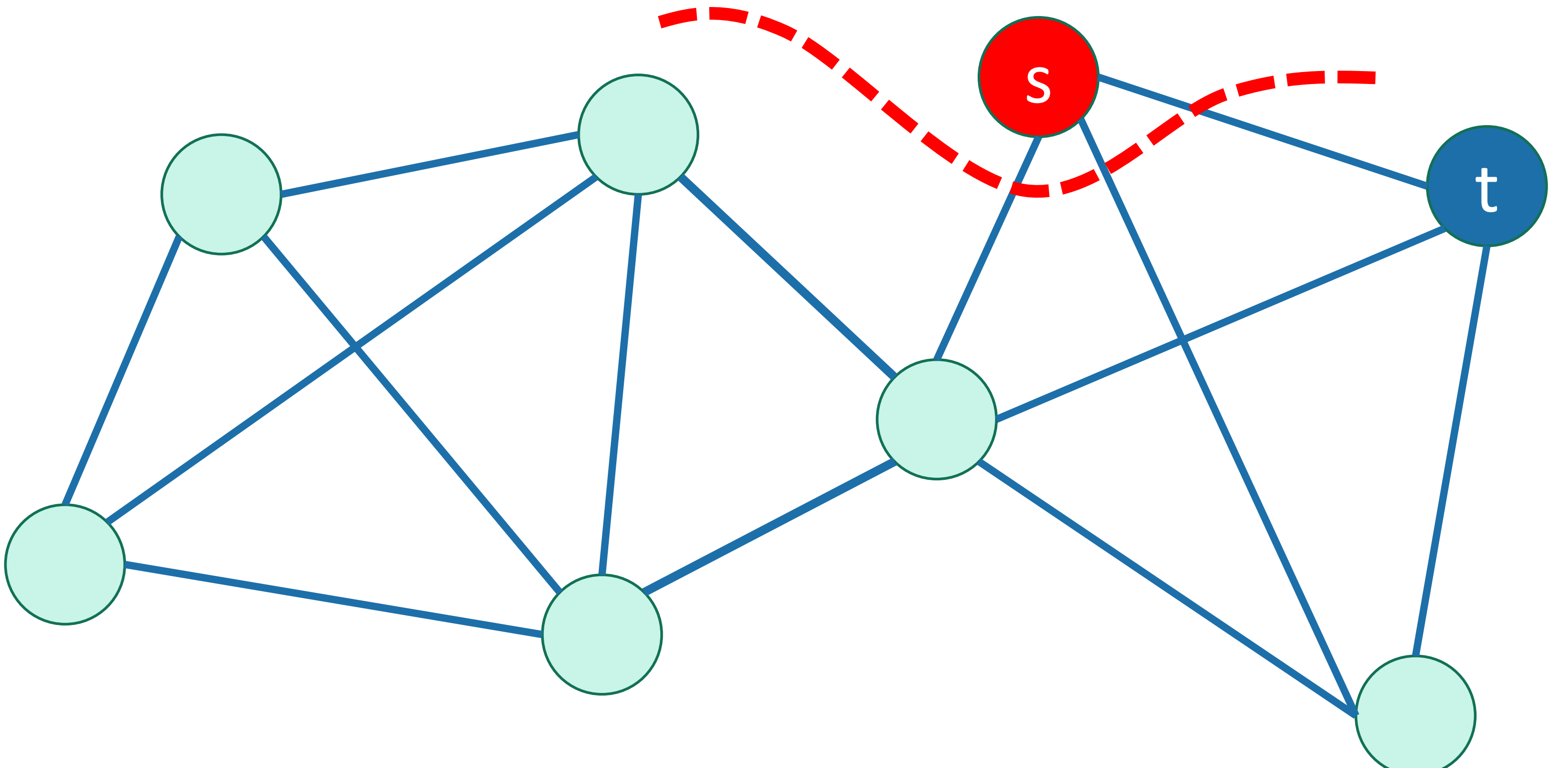
is a cut that has the fewest edges possible crossing it.



Why "global"?

- Next time we'll talk about **min s-t cuts**

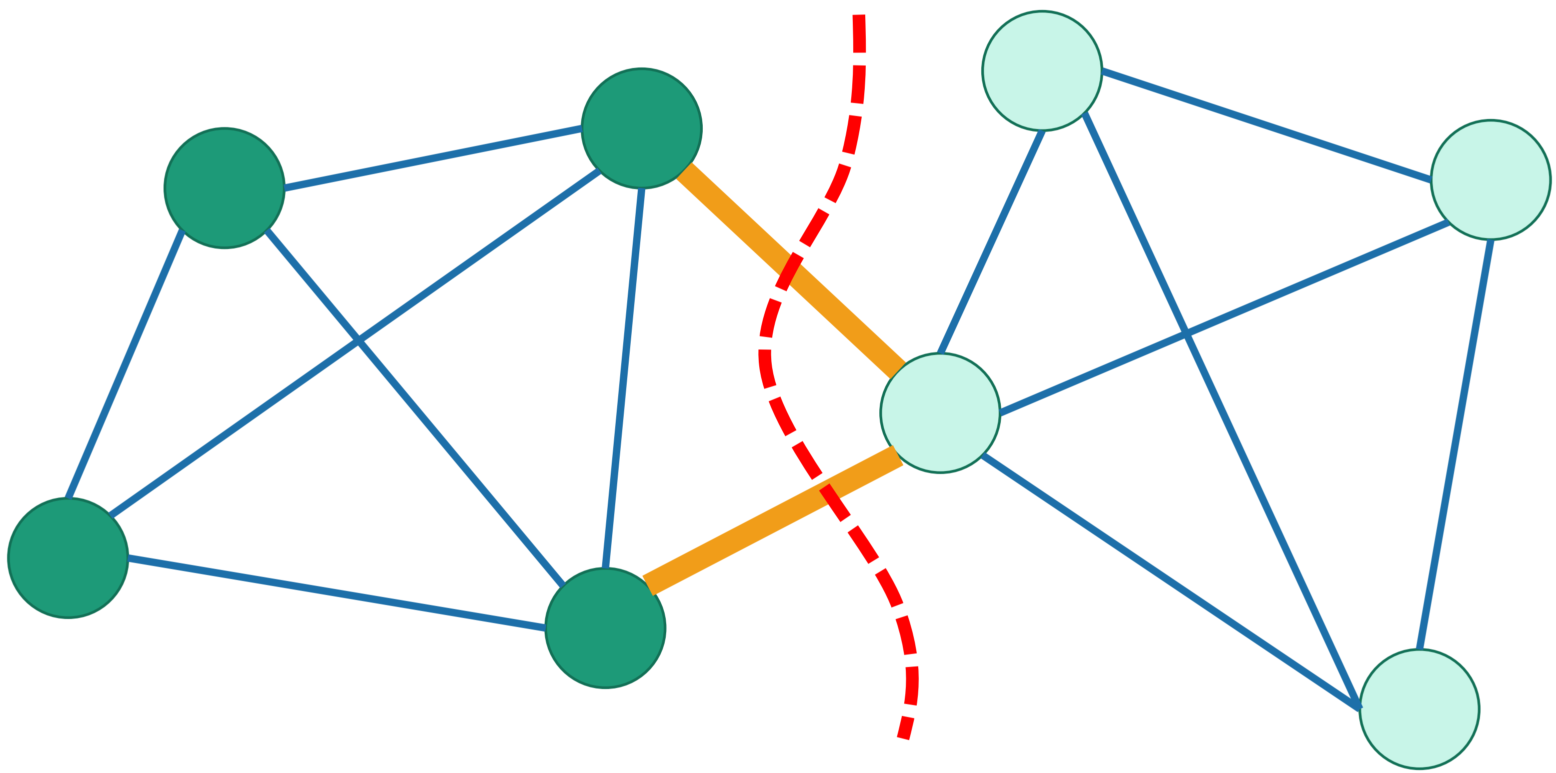
Minimum cut which separates a specified vertex s from t



- Today, there are no special vertices, so the minimum cut is **"global."**

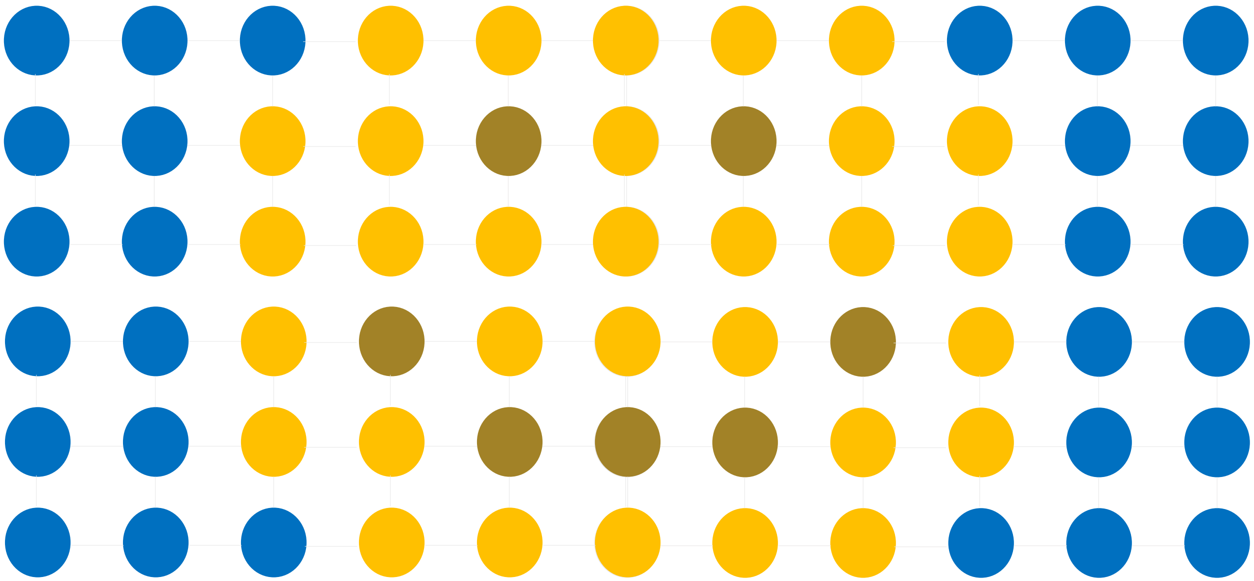
A (global) minimum cut

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



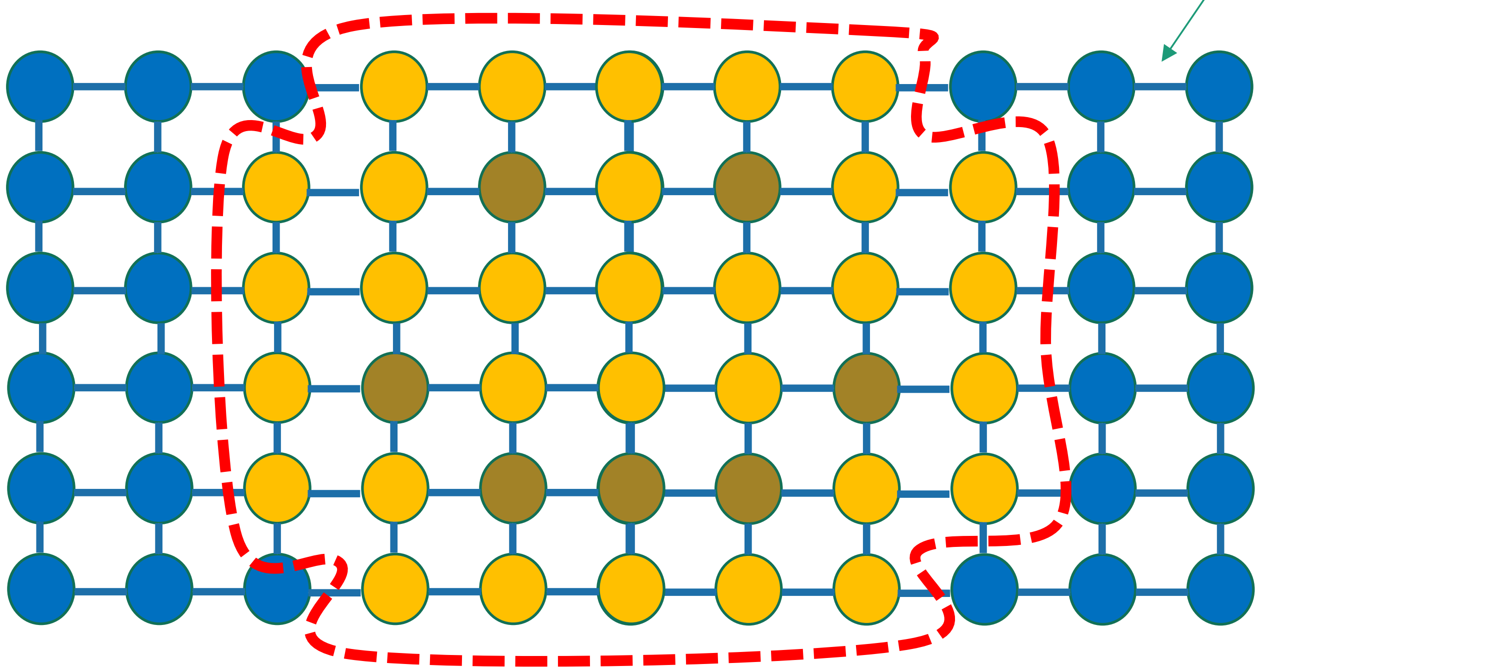
Why might we care about global minimum cuts?

- One example is image segmentation:



Why might we care about global minimum cuts?

- One example is image segmentation:



- We'll see more applications for other sorts of min-cuts next week

*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 **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.

Different sorts of gambling

- QuickSort is a **Las Vegas randomized algorithm**
  - It is always correct.
  - It might be slow.

Yes, this is a technical term.

Formally:

- For all inputs A, QuickSort(A) returns a sorted array.
- For all inputs A, with high probability over the choice of pivots, QuickSort(A) runs quickly.

Different sorts of gambling

- Karger's Algorithm is a **Monte Carlo randomized algorithm**
  - It is always fast.
  - It might be wrong.

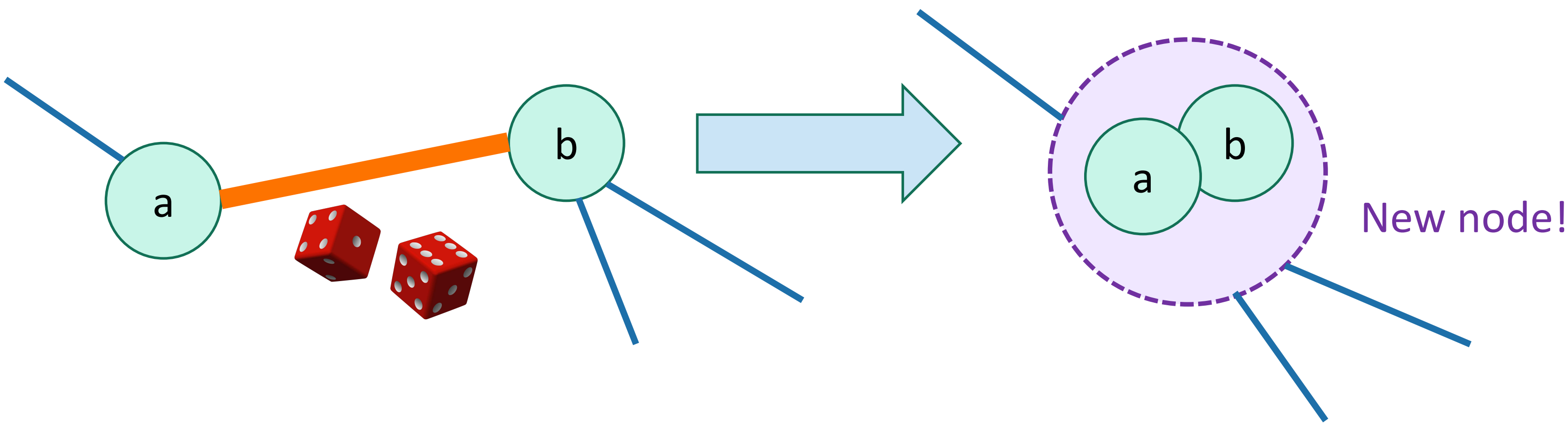
Formally:

- For all inputs G, with probability at least ___ over the randomness in Karger's algorithm, Karger(G) returns a minimum cut.

- For all inputs G, with probability 1 Karger's algorithm runs in time no more than ____.

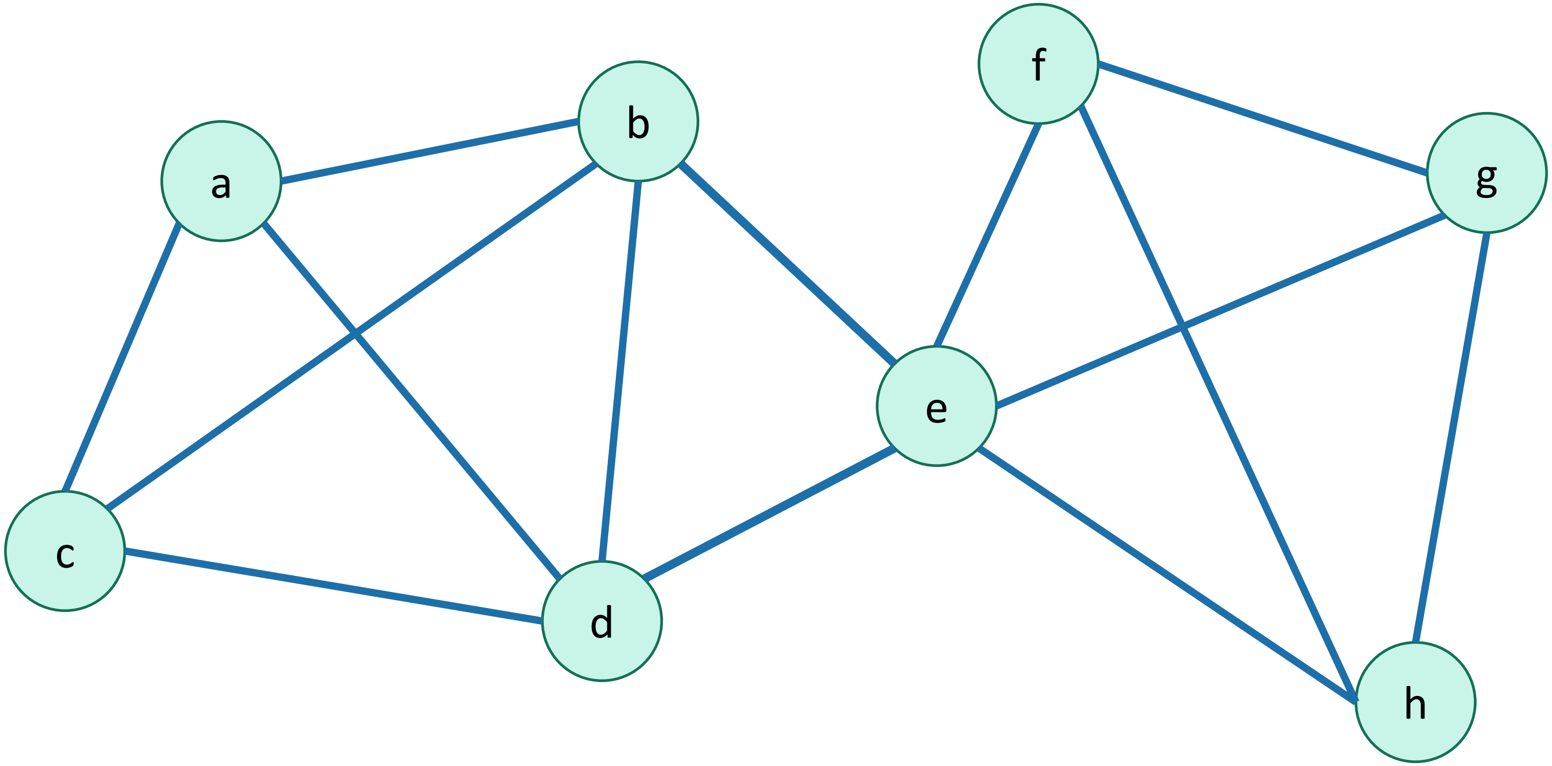
Karger's Algorithm

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

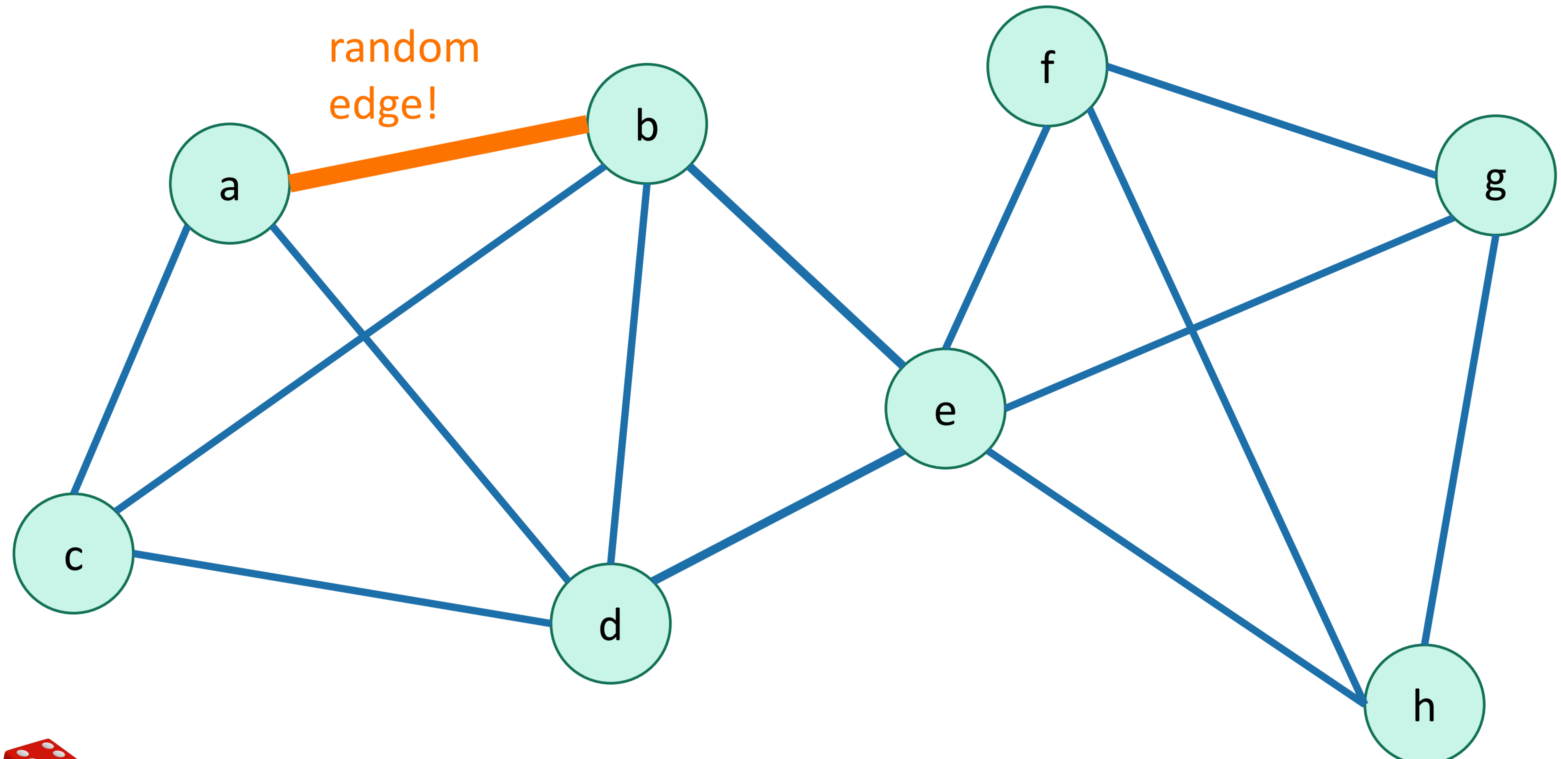


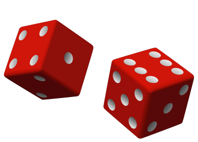
Why is this a good idea? We'll see shortly.

Karger’s algorithm

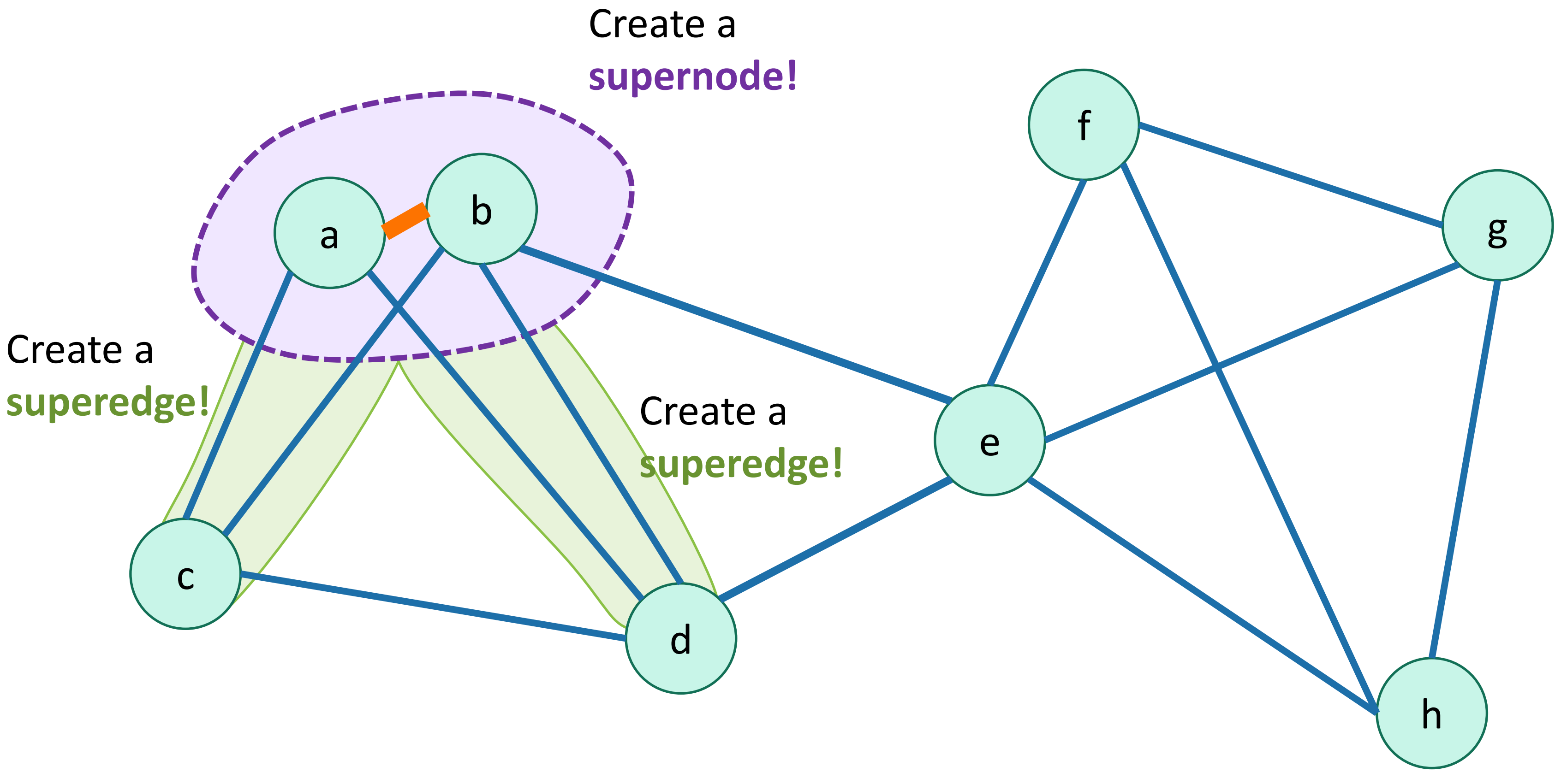


Karger's algorithm

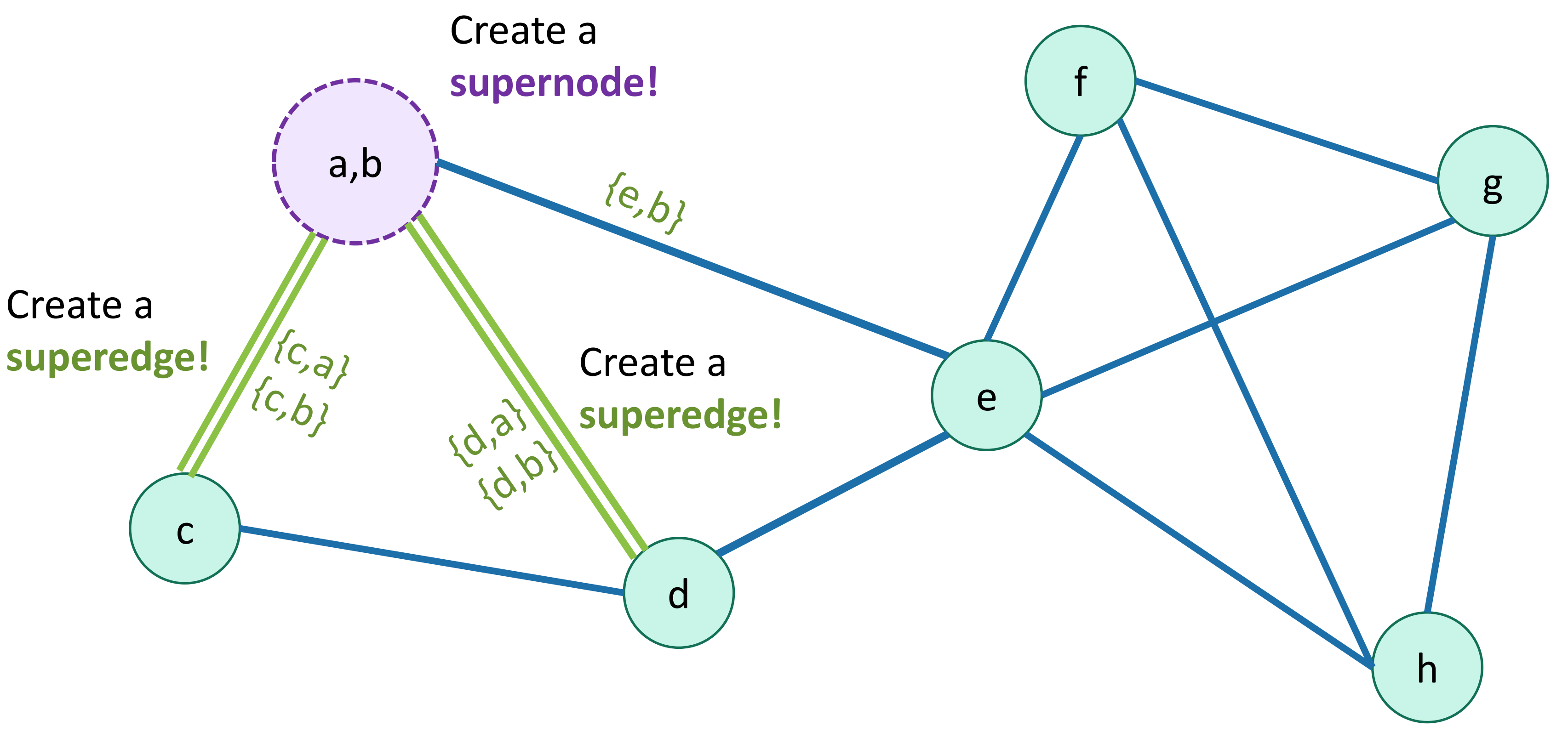




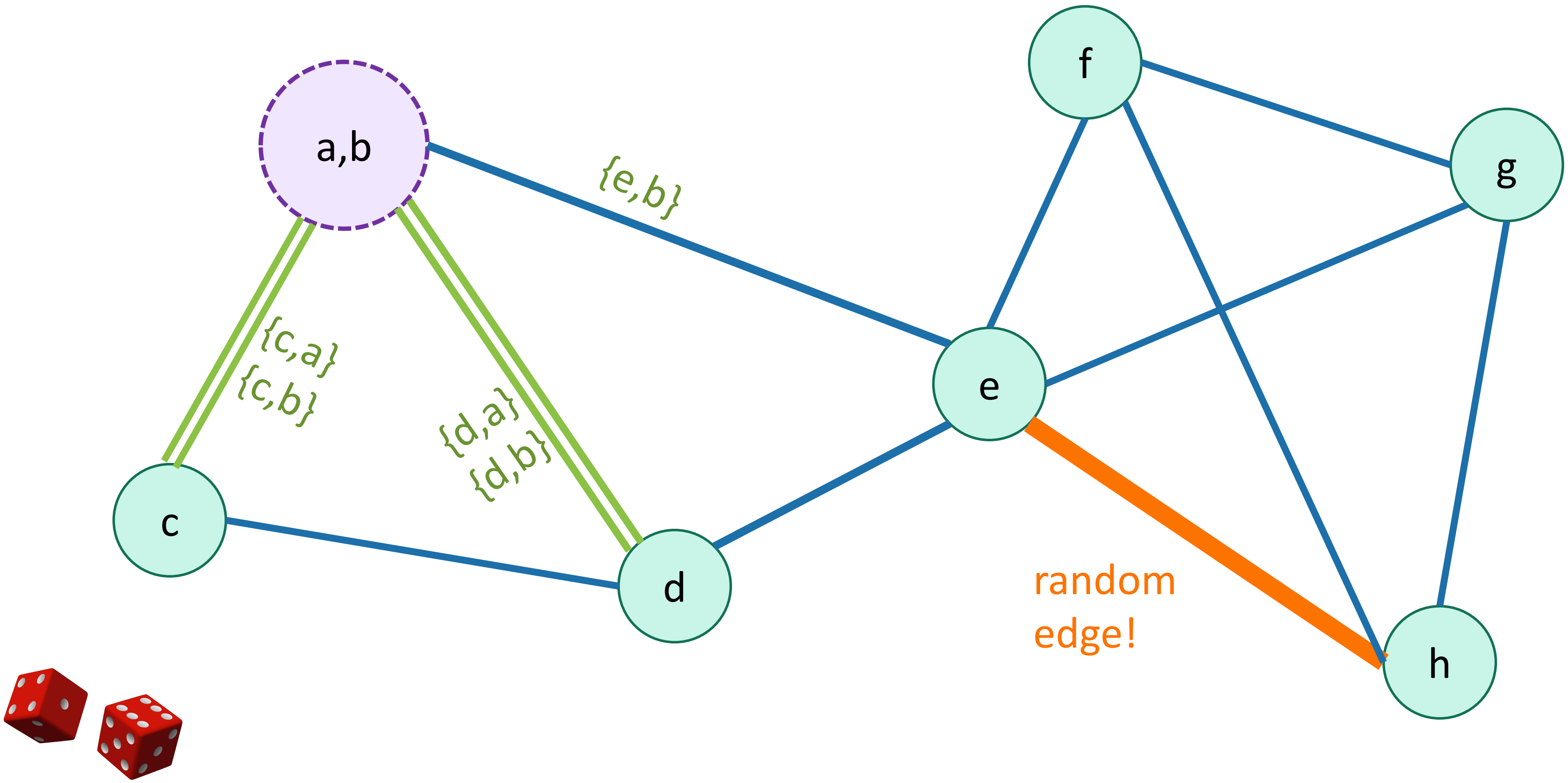
Karger's algorithm



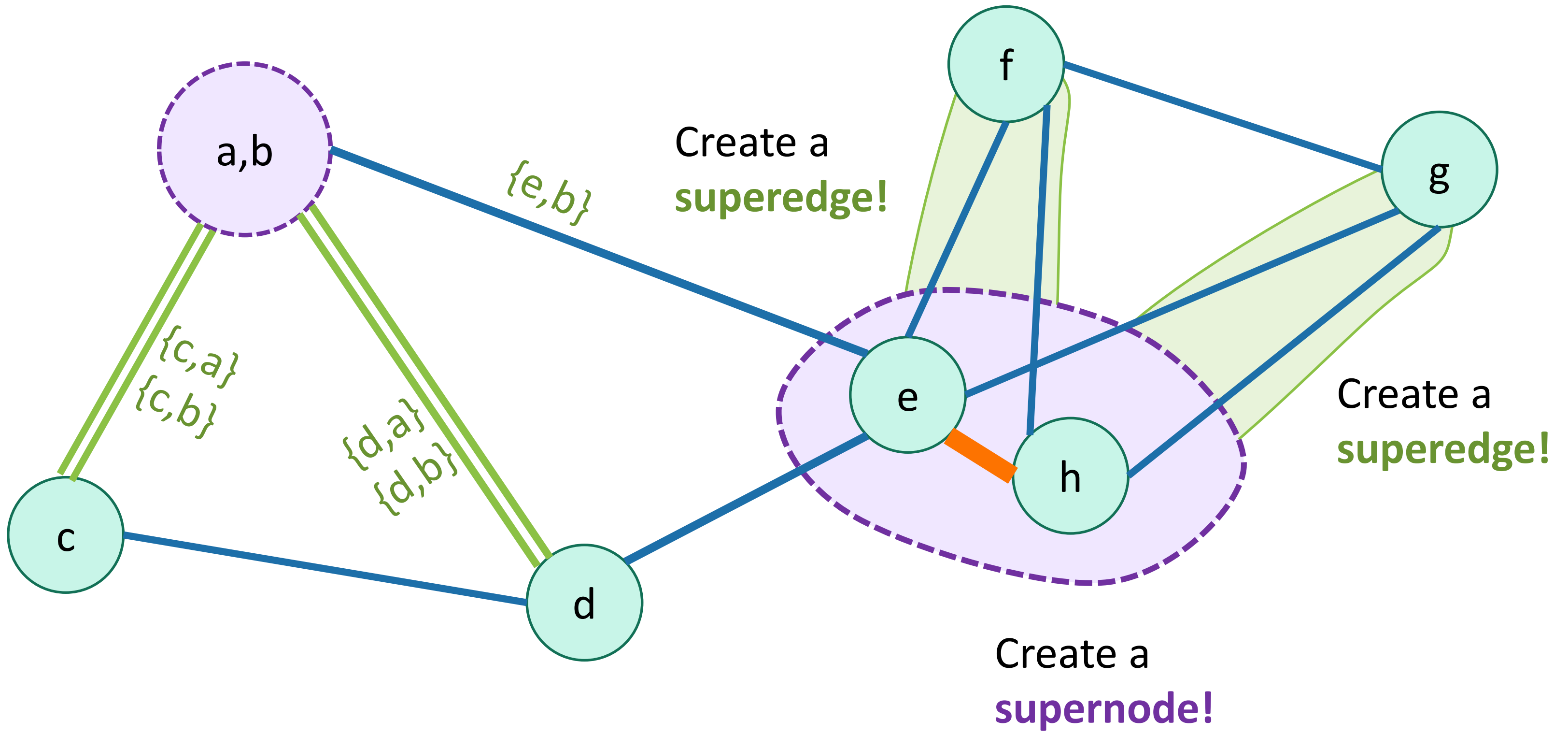
Karger's algorithm



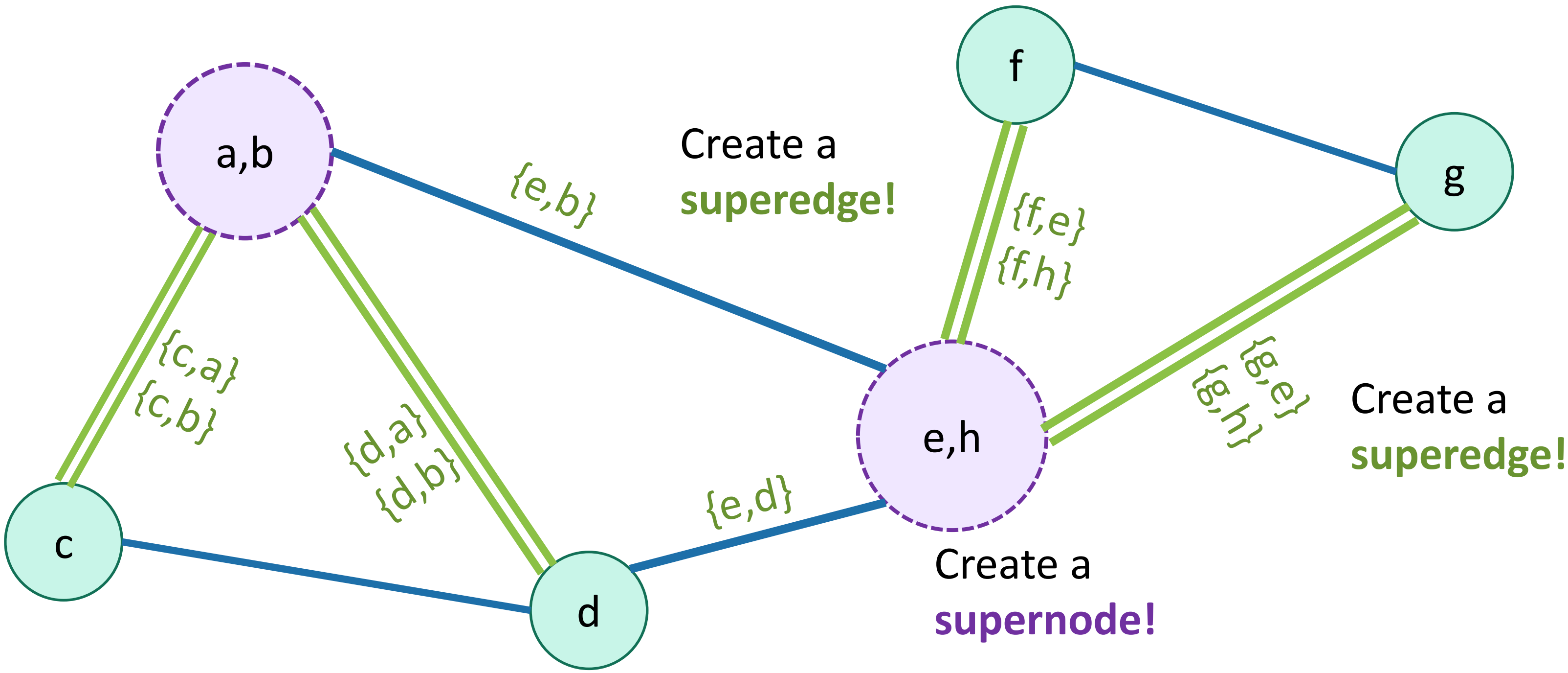
Karger’s algorithm



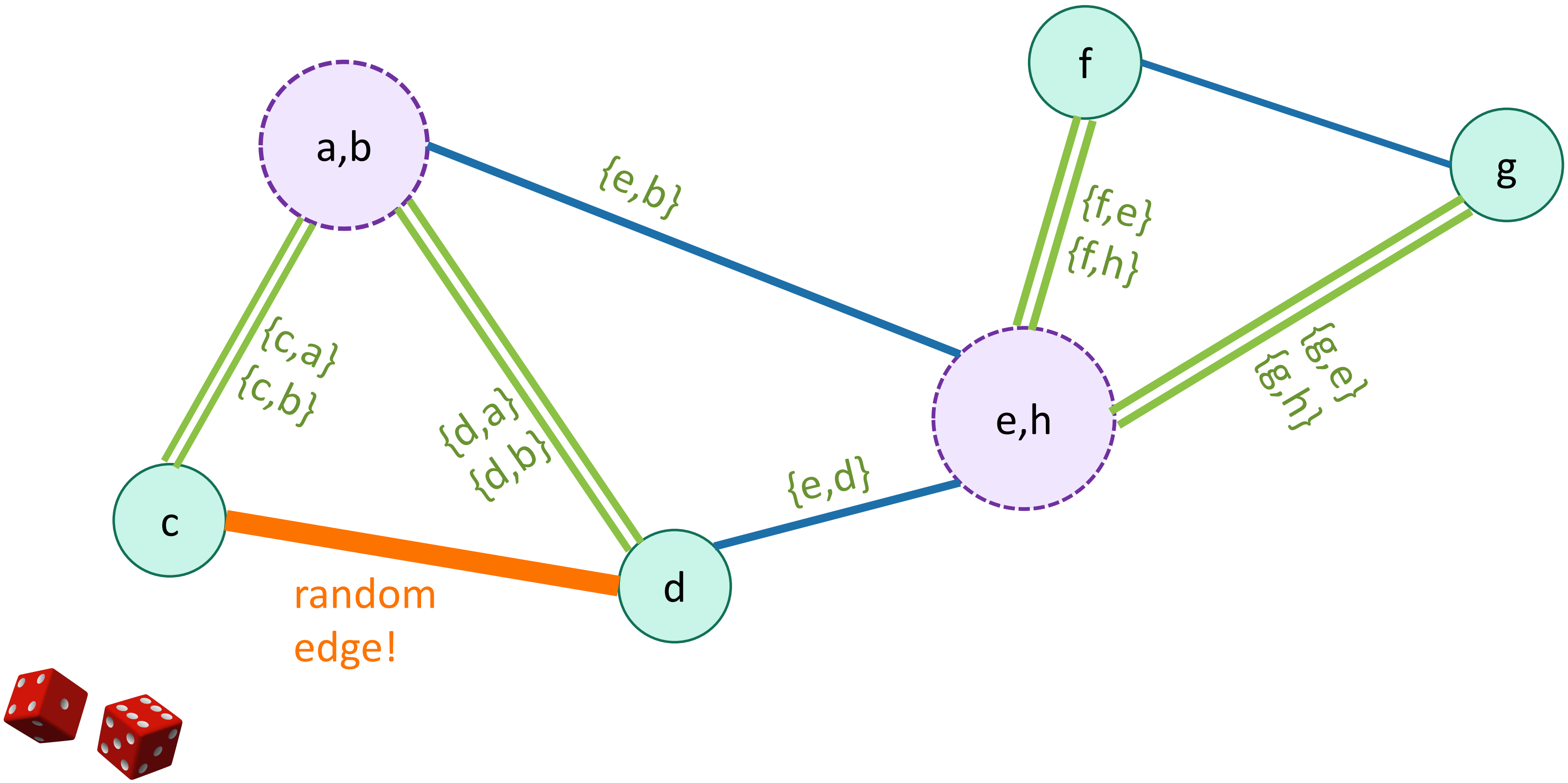
Karger's algorithm



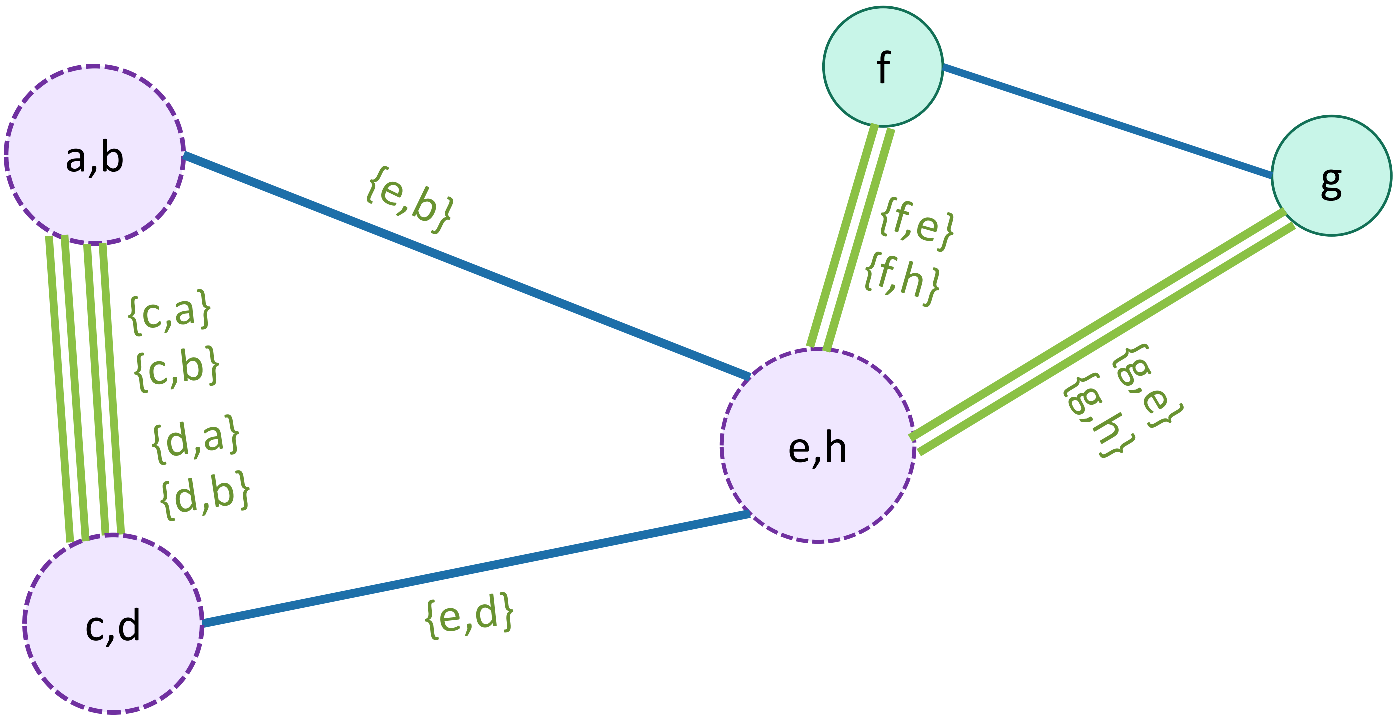
Karger's algorithm



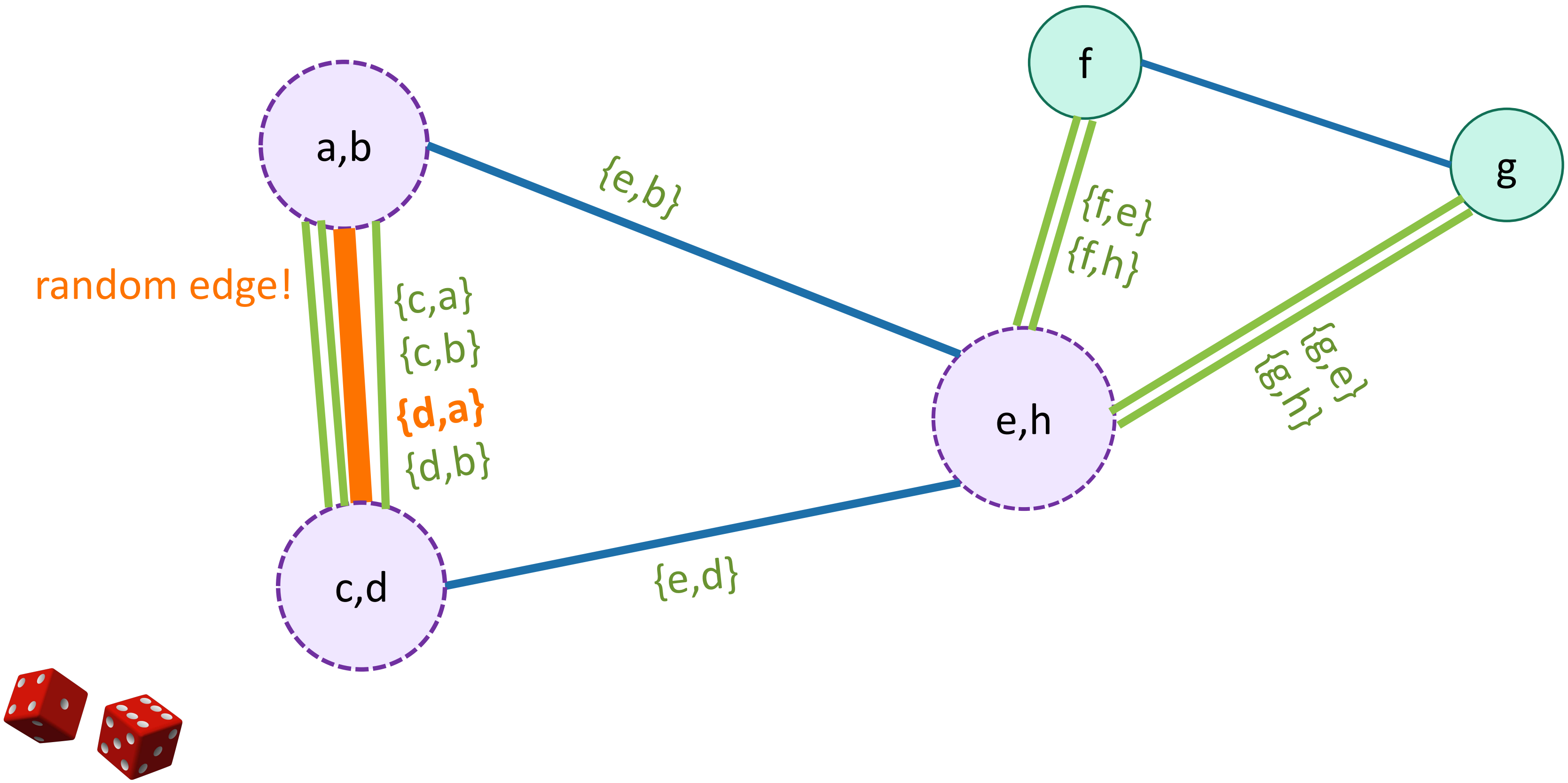
Karger's algorithm



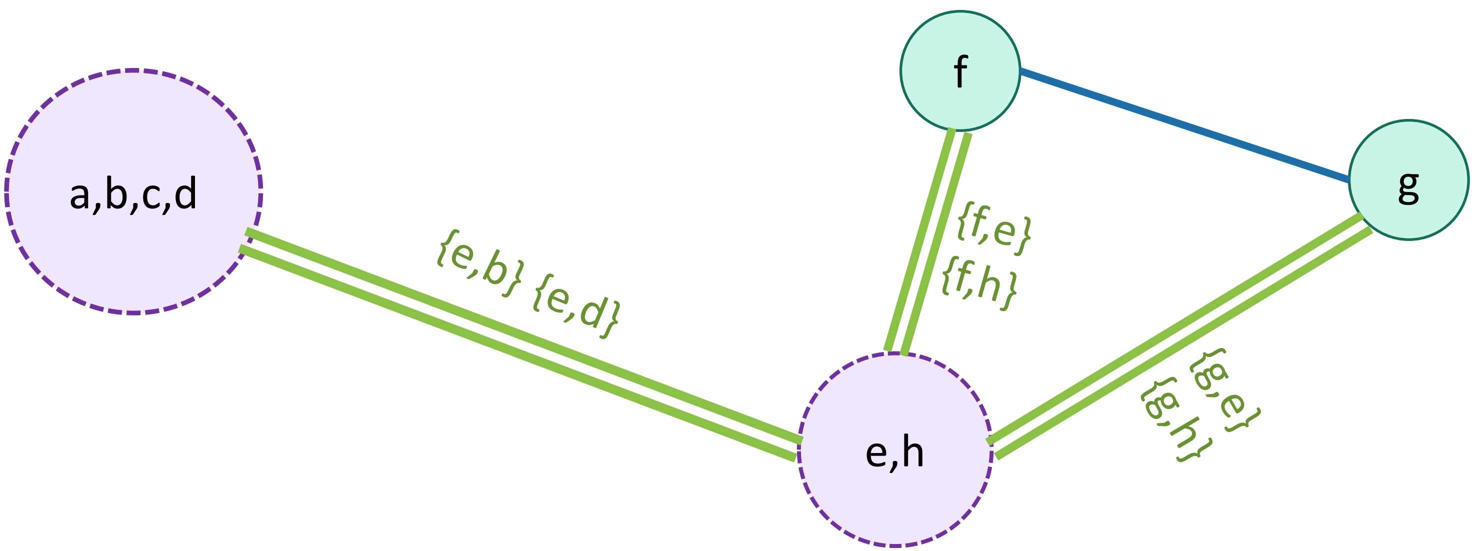
Karger’s algorithm



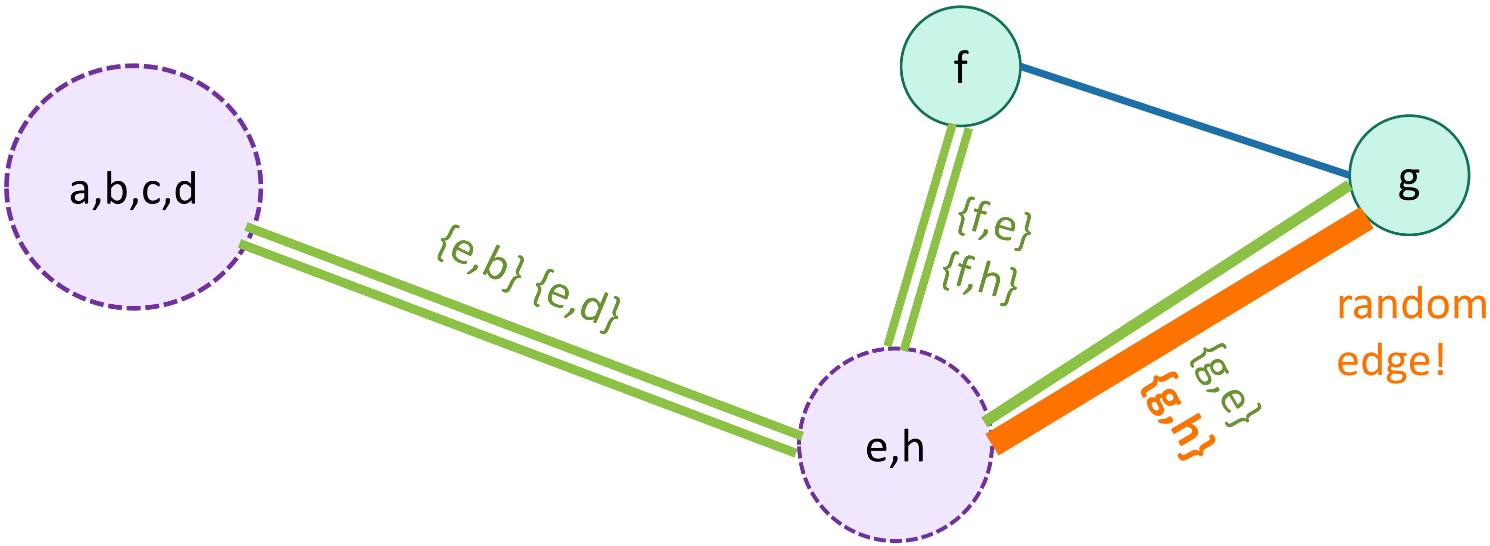
Karger's algorithm

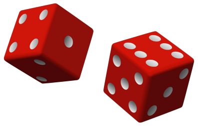


Karger’s algorithm

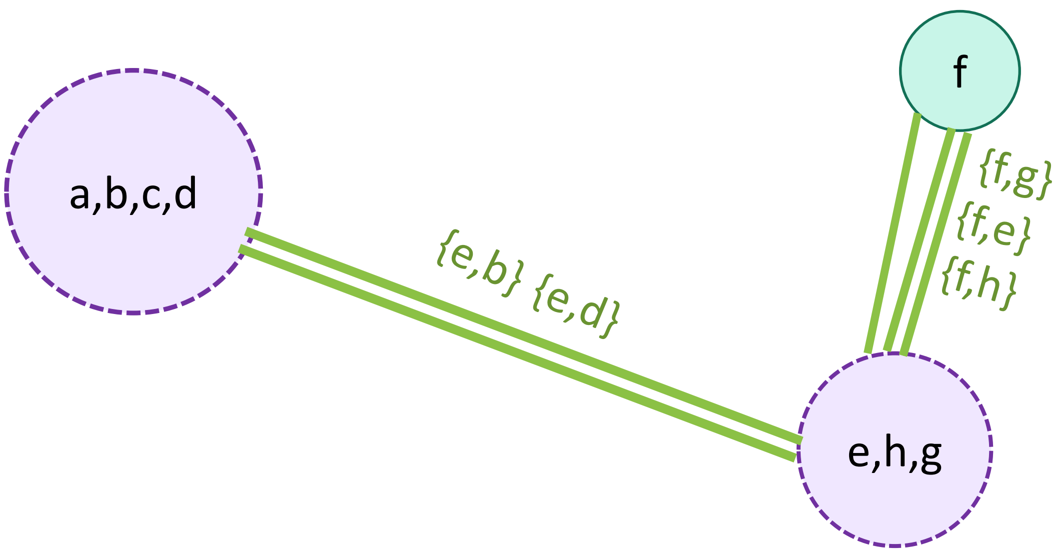


Karger's algorithm

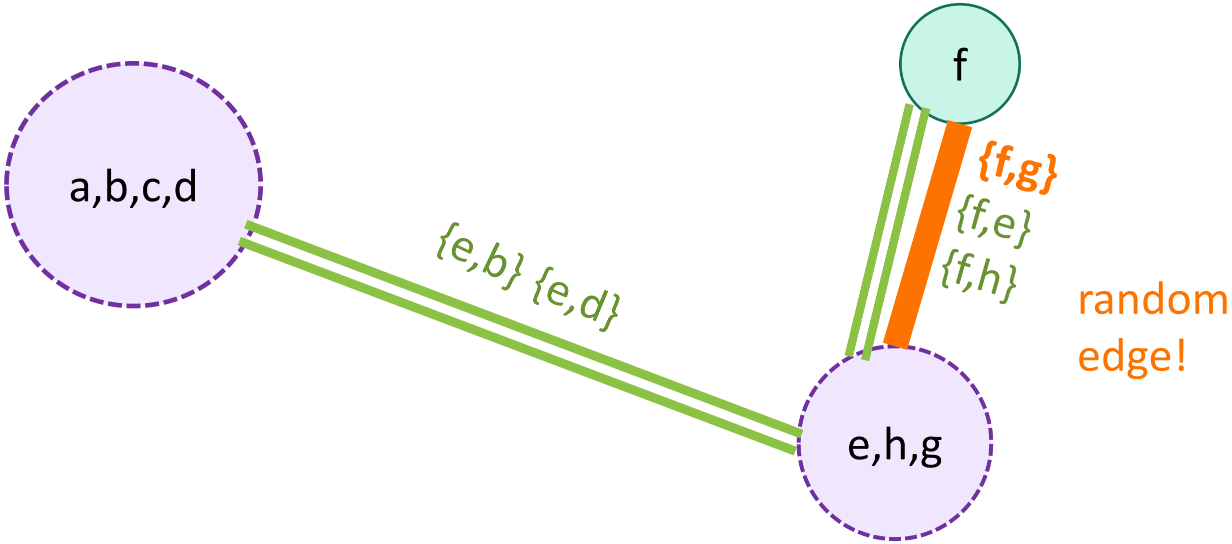


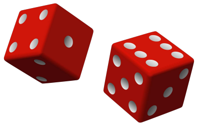


Karger's algorithm

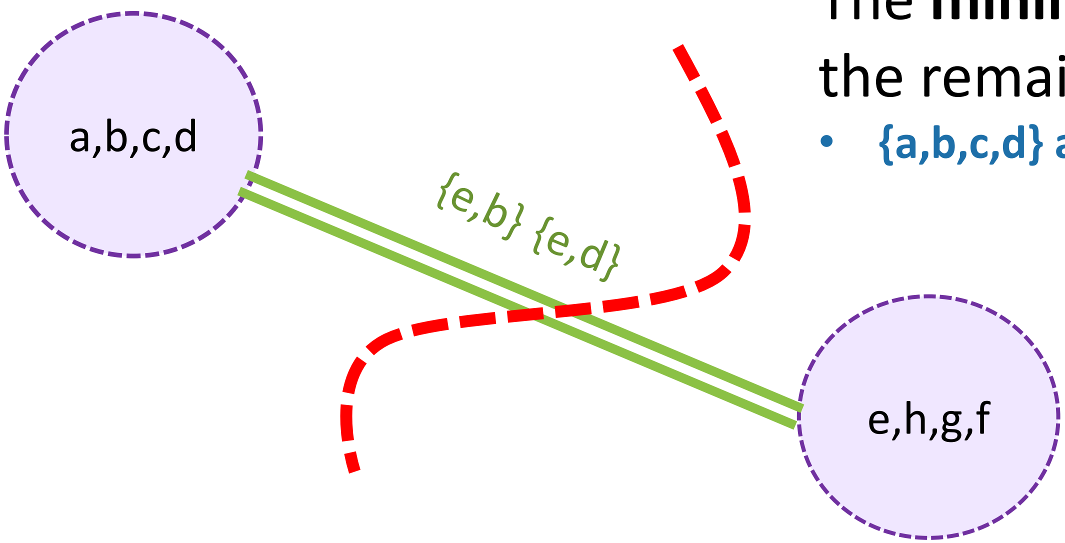


Karger's algorithm





Karger's algorithm



Now stop!

- There are only two nodes left.

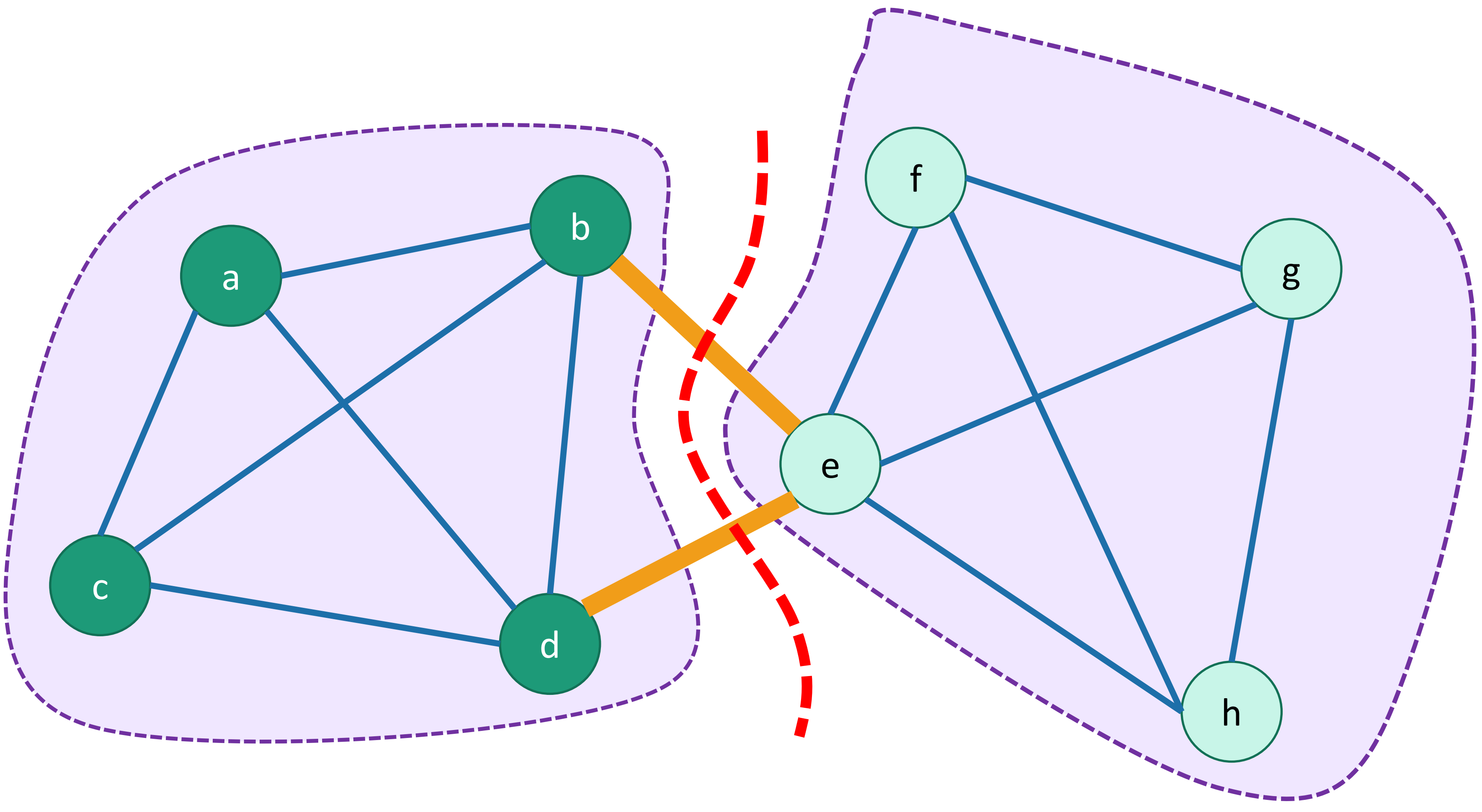
The **minimum cut** is given by the remaining super-nodes:

- **{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?

How do we implement this?

- See Lecture 16 IPython Notebook for one way
  - This maintains a secondary “superGraph” which keeps track of superNodes and superEdges
  - There’s a hidden slide with pseudocode
- Running time?
  - We contract at most n-2 edges
    - Each time we contract an edge we get rid of a vertex, and we get rid of at most n – 2 vertices total.
  - Naively each contraction takes time O(n)
    - Maybe there are about n nodes in the superNodes that we are merging.
  - 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.

Pseudocode

Let $\overline{\boldsymbol{u}}$ denote the SuperNode in Γ containing u
Say $E_{\overline{\boldsymbol{u}},\overline{\boldsymbol{v}}}$ is the SuperEdge between $\overline{\boldsymbol{u}}, \overline{\boldsymbol{v}}$.

This slide skipped in class

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

      // merge the SuperNode containing u with the SuperNode containing v.
    - $F \leftarrow F \setminus E_{\overline{\boldsymbol{u}},\overline{\boldsymbol{v}}}$

      // remove all the edges in the SuperEdge between those SuperNodes.
  - **return** the cut given by the remaining two superNodes.

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

merge takes time O(n) naively

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

total runtime O(n^2)

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

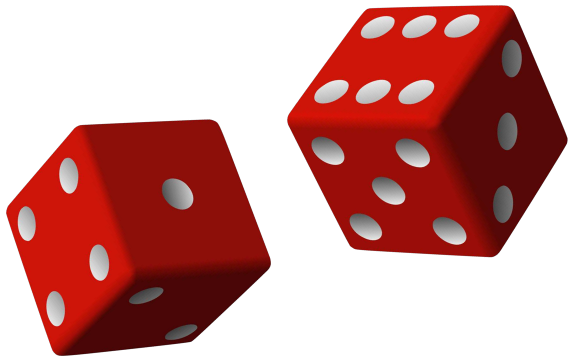
Karger's algorithm

- Does it work?
  - No?

- Is it fast?
  - $O(n^2)$

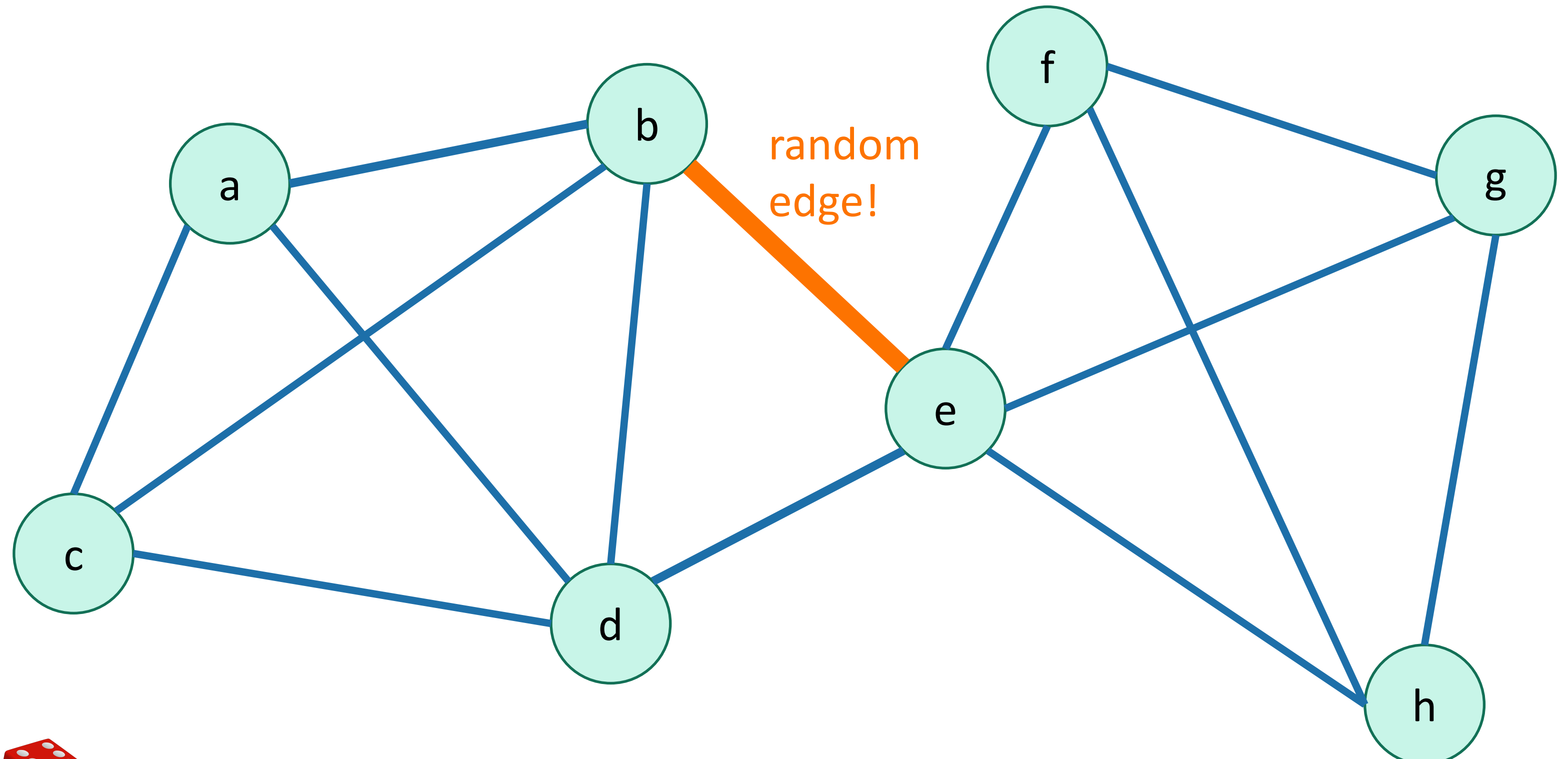
Why did that work?

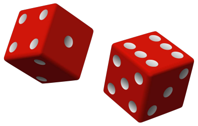
- We got really lucky!
- This could have gone wrong in so many ways.



Karger's algorithm

Say we had chosen this edge

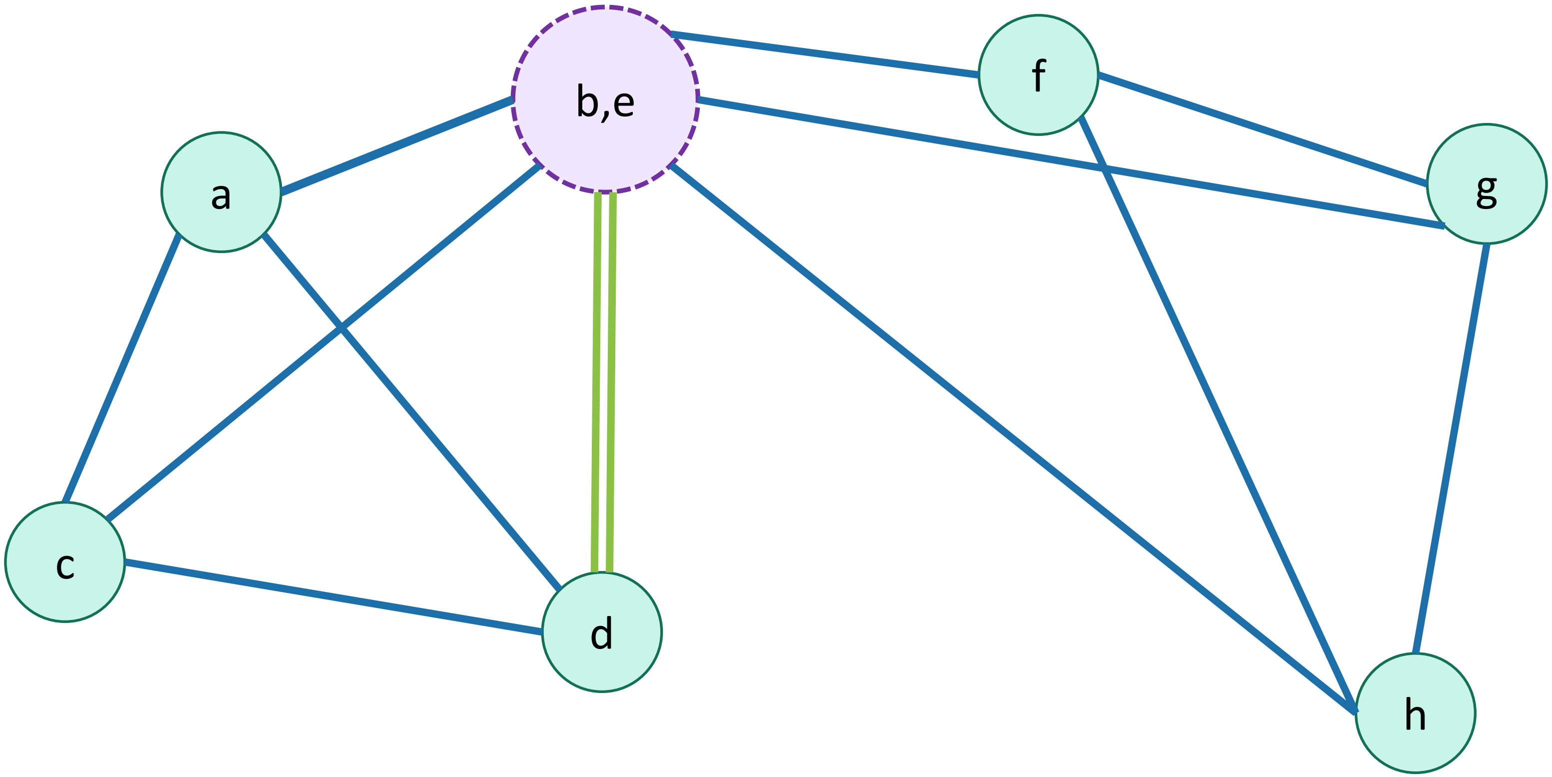




Karger’s algorithm

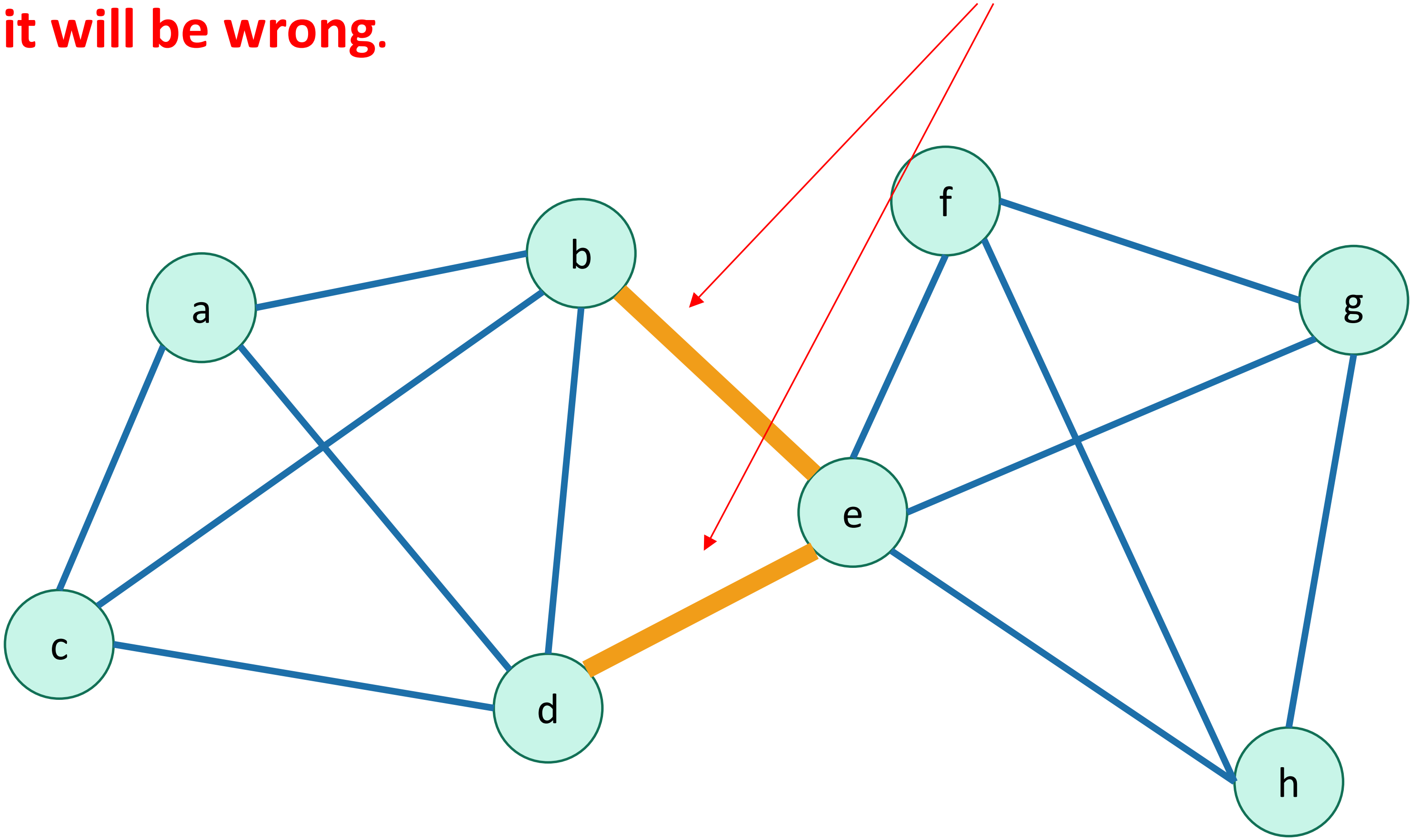
Say we had chosen this edge

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

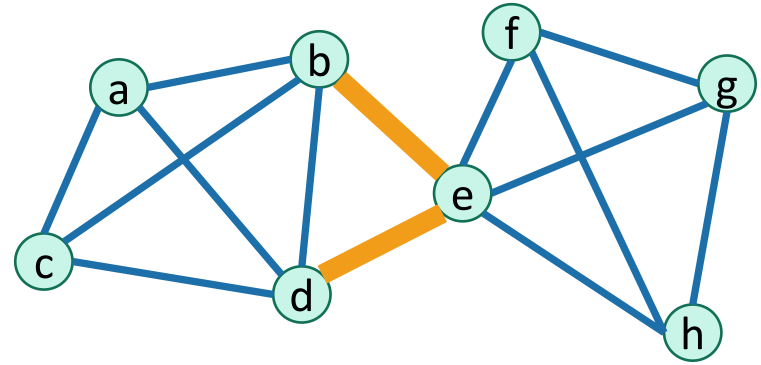


Even worse

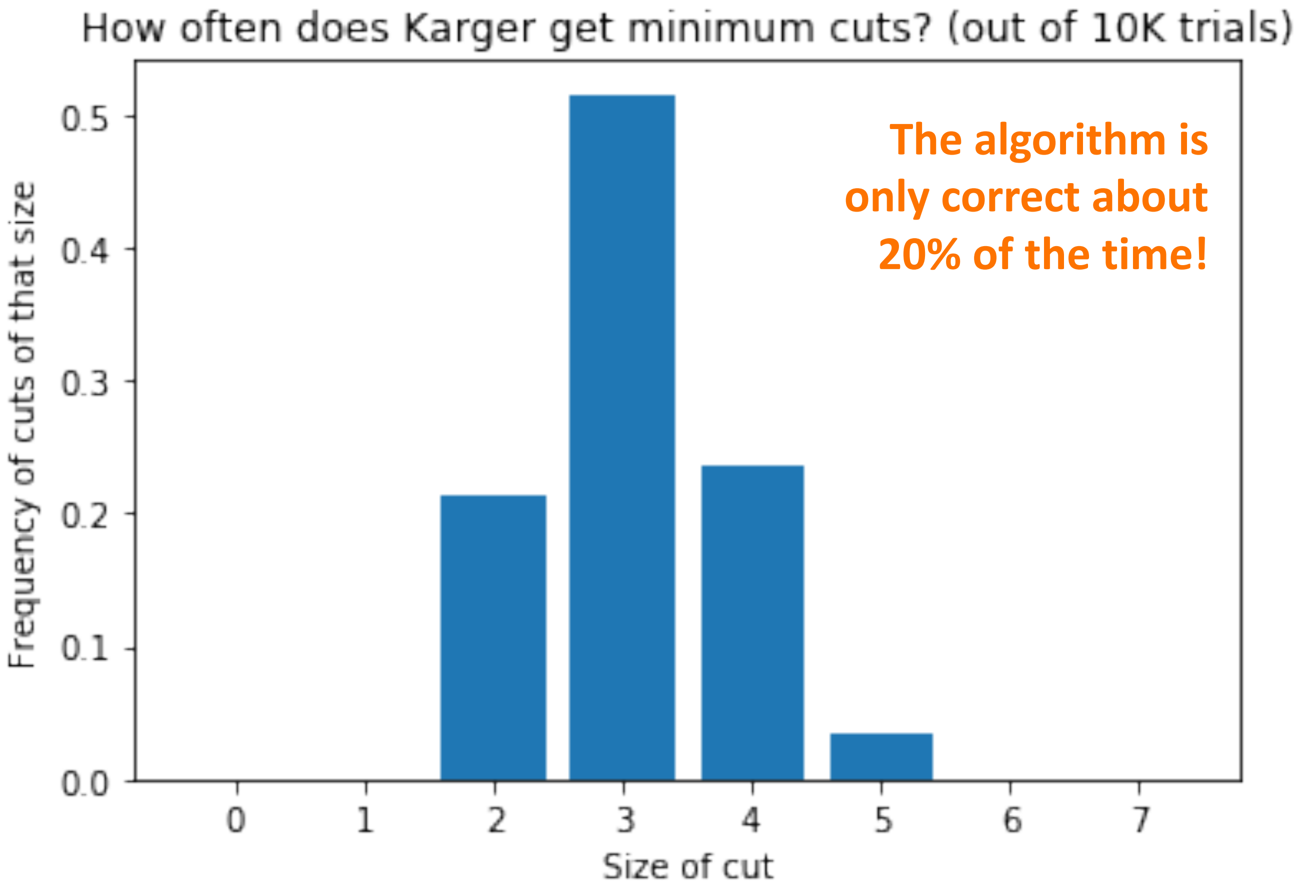
If the algorithm **EVER** chooses either of **these edges**,
it will be wrong.



How likely is that?

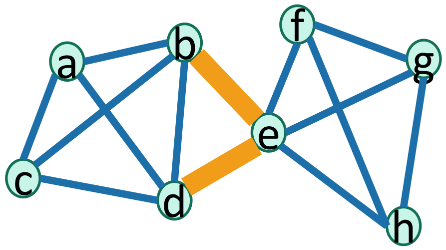


- For this particular graph, I did it 10,000 times:



That doesn't sound good

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



- Let's compare Karger's algorithm to the algorithm:

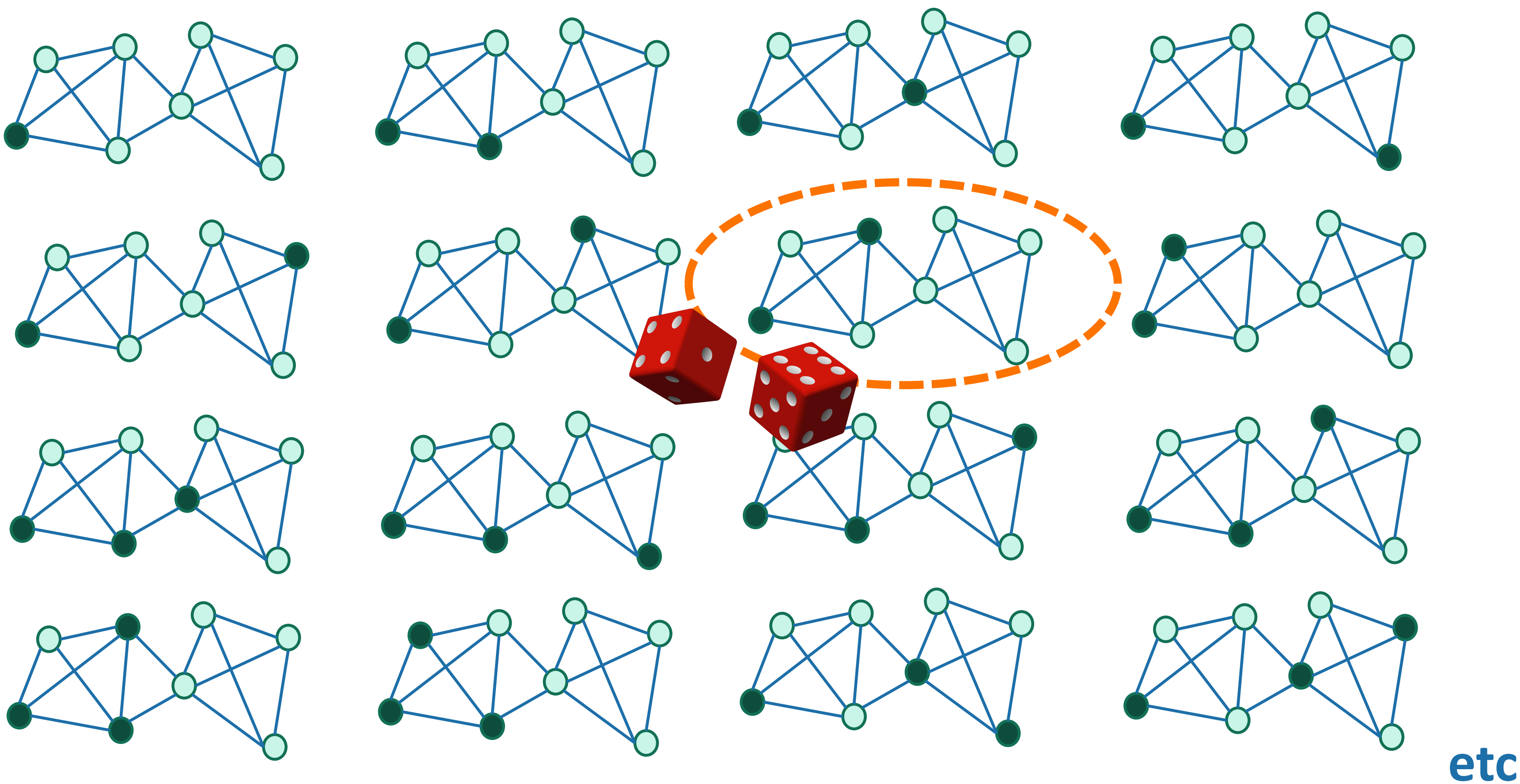
Choose a completely random cut and hope that it's a minimum cut.

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.

Random cuts

- Suppose that we chose cuts uniformly at random.
  - That is, pick a random way to split the vertices into 2 parts.



Random cuts

- Suppose that we chose cuts uniformly at random.
  - That is, pick a random way to split the vertices into 2 parts.

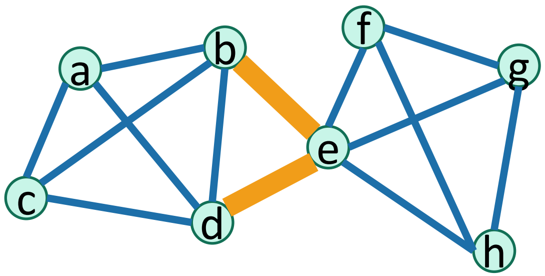
- The probability of choosing the minimum cut is*…

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

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

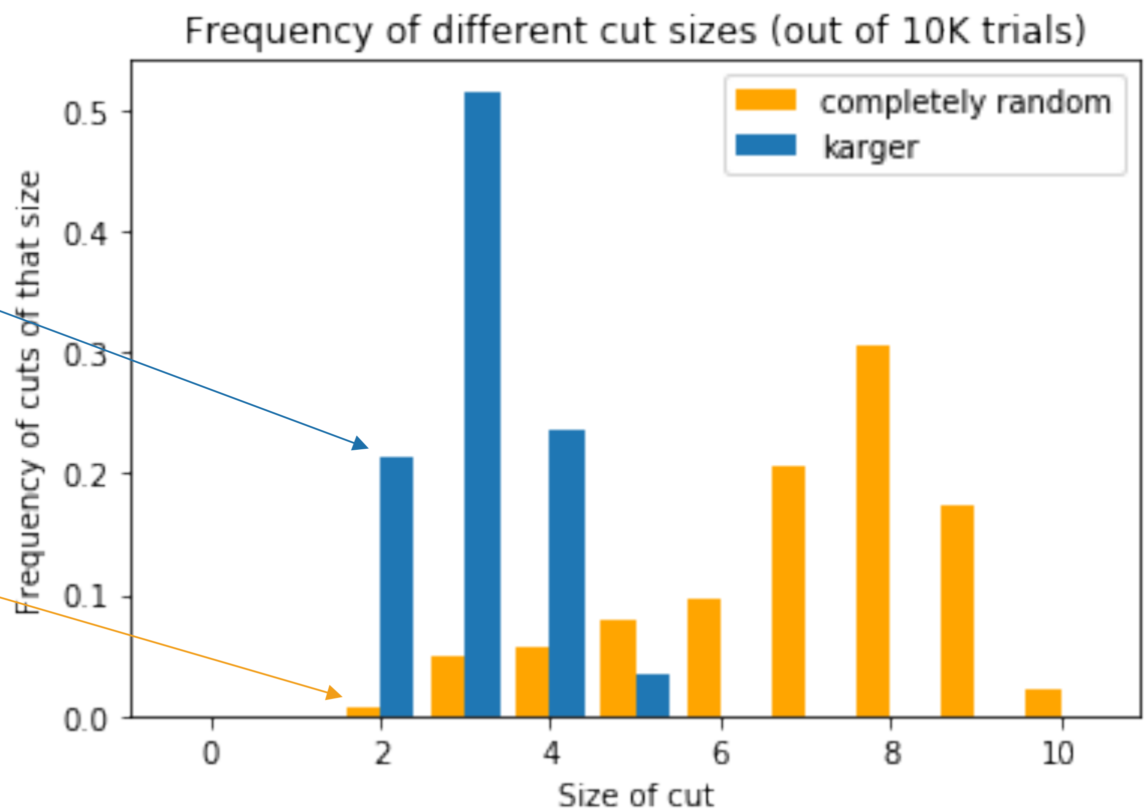
*For this example in particular

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

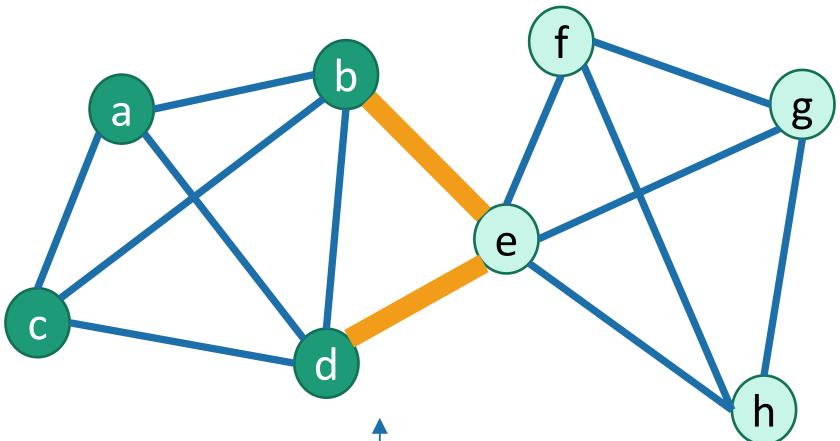


What’s going on?

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

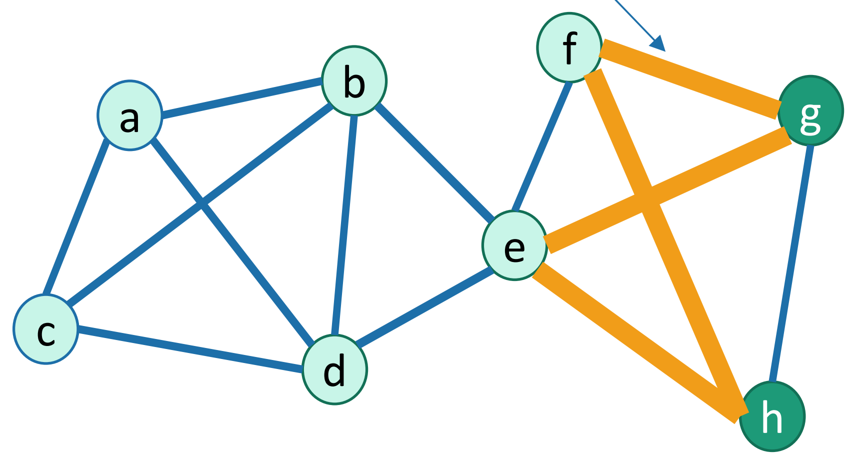
Lucky the lackadaisical lemur

- 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**.

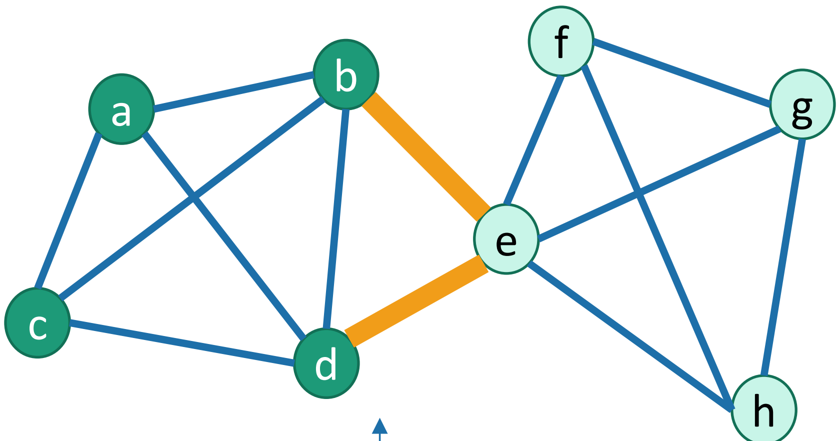


- Neither A nor B are very likely, **but A is more likely than B**.

What’s going on?

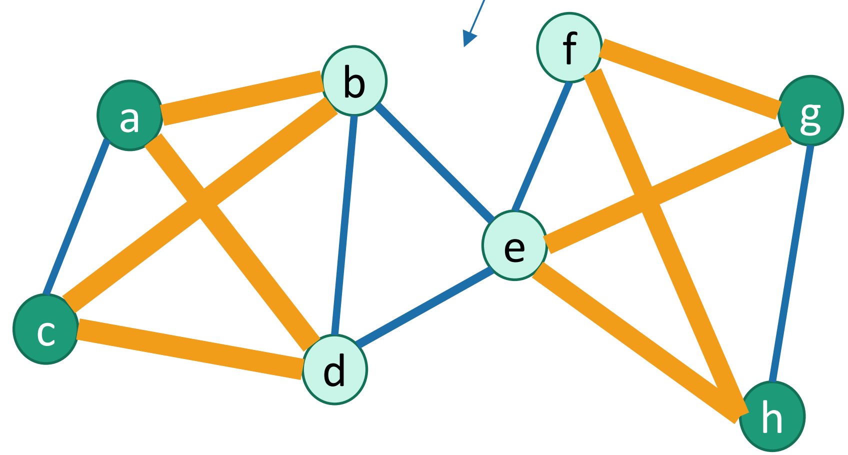
Thing 2: By only contracting edges we are ignoring certain really-not-minimal cuts.

Lucky the lackadaisical lemur



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 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
from pre-lecture exercise

- Suppose you have a magic button that produces one of 5 numbers, {a,b,c,d,e}, uniformly at random when you push it.
- Q: What is the minimum of a,b,c,d,e?

6 3 3 2 5 2 5

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

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

[On board]

[This is approximately what's on the board]

Slide skipped in class

This is the same calculation we've done a bunch of times:

- E[Number of times we push the button until we get the minimum value] = 1/(0.20) = 5

This one we've done less frequently:

- Pr[We push the button t times and don't ever get the min] = $(1 - 0.2)^t$
- Pr[We push the button 5 times and don't ever get the min] = $(1 - 0.2)^5 \approx 0.33$
- Pr[We push the button 10 times and don't ever get the min] = $(1 - 0.2)^{10} \approx 0.1$

In this context

- Run Karger's! The cut size is 6!

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

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

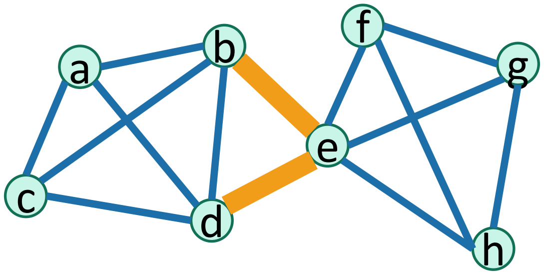
- Run Karger's! The cut size is 2!

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!

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 this?

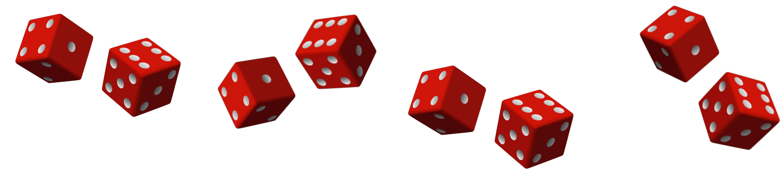
Also, we should be a bit more precise about this "about 5 times" statement.

Plucky the pedantic penguin

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.

Questions



To generalize this approach to all graphs

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

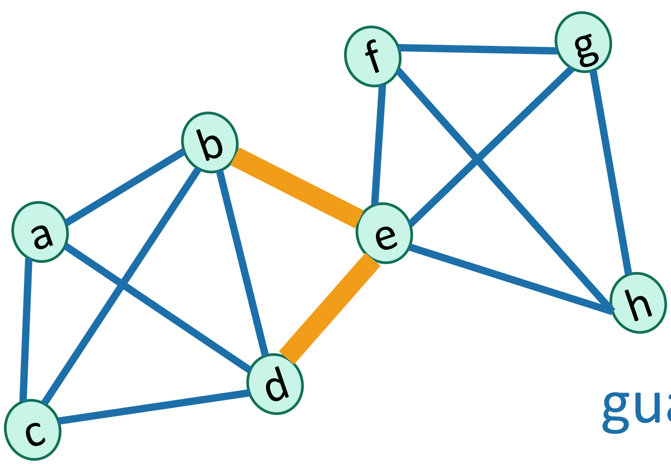
2. How many times should we run Karger's algorithm to "probably" succeed?
   - Say, with probability 0.99?
   - Or more generally, probability $1 - \delta$?

Answer to Question 1

Claim:

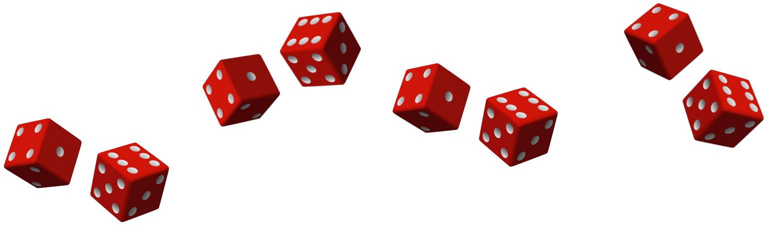
The probability that Karger's algorithm returns a minimum cut is

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



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

Answers



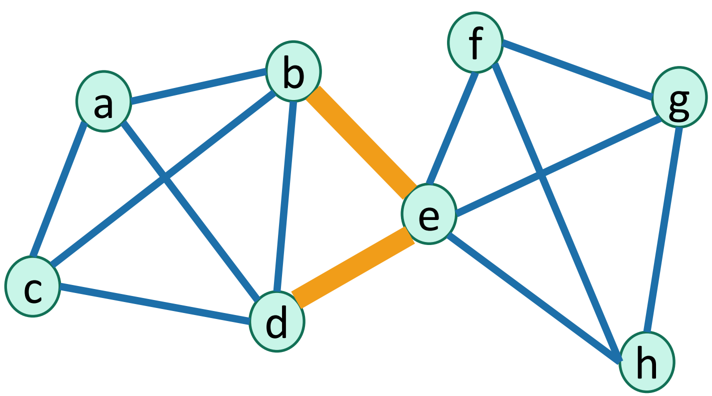
1. What is the probability that Karger's algorithm returns a minimum cut?

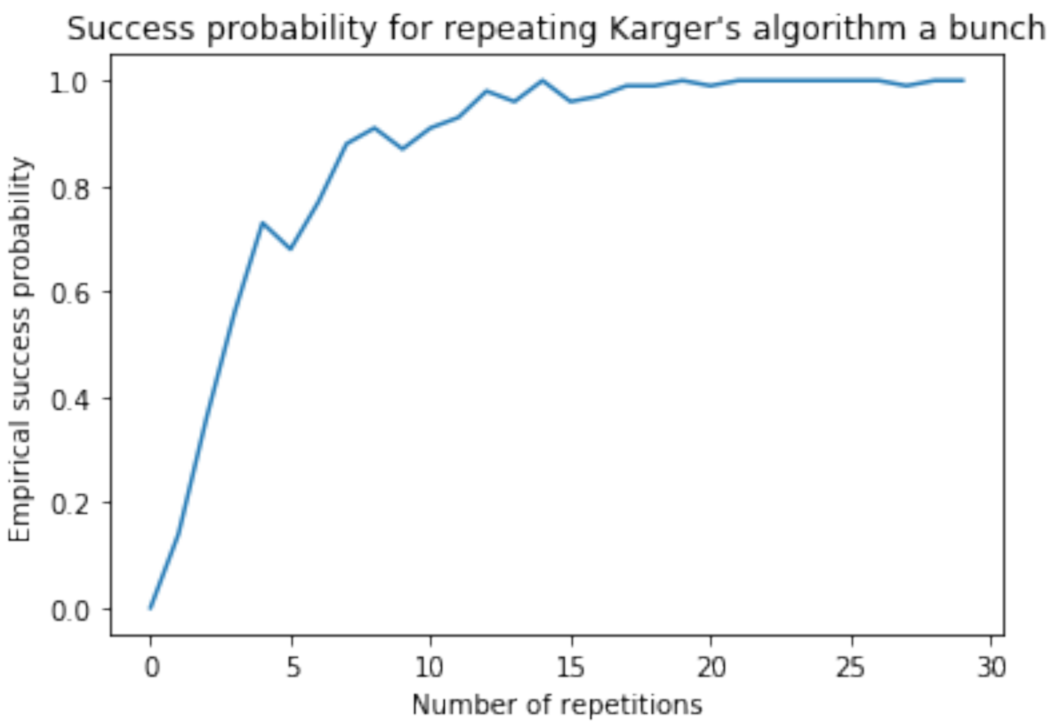
   According to the claim, at most $\frac{1}{\binom{n}{2}}$

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 $1 - \delta$?





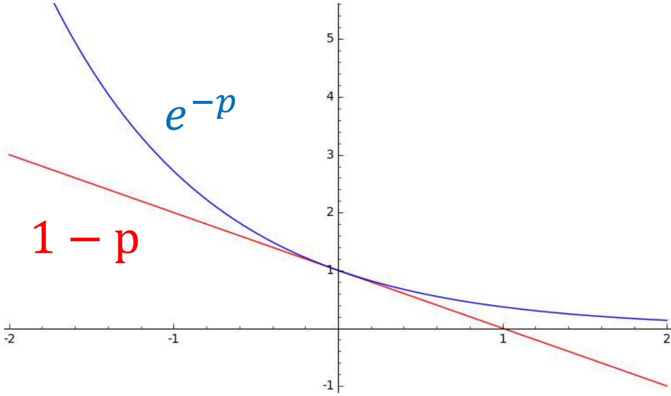
A computation

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

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

Independent

- $\Pr[\text{ don't return a min cut in T trials }] = (1-p)^T$
- So p = $1/\binom{n}{2}$ by the Claim. Let's choose T = $\binom{n}{2}$ ln$(1/\delta)$.
- $\Pr[\text{ don't return a min cut in T trials }]$
  - $= (1\ - p)^T$
  - $\leq (e^{-p})^T$
  - $= e^{-pT}$
  - $= e^{-\ln\left(\frac{1}{\delta}\right)}$
  - $= \delta$



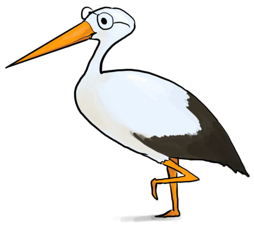
$1-\mathrm{p} \leq e^{-p}$

Theorem

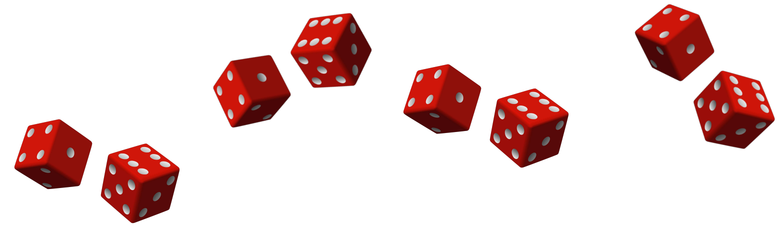
Assuming the claim about $1/\binom{n}{2}$ …

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

How many repetitions would you need if instead of Karger we just chose a uniformly random cut?



Answers



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

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

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

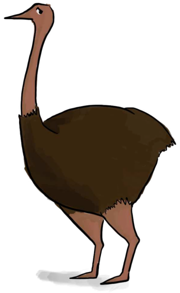
   $\binom{n}{2} \log\left(\frac{1}{\delta}\right)$ times.

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 δ as constant.

Again we can do better with a union-find data structure. Write pseudocode for—or better yet, implement—a fast version of Karger’s algorithm! How fast can you make the asymptotic running time?



Ollie the over-achieving ostrich

Theorem

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

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 $^{1}/_{\binom{n}{2}}$

Now let's prove that claim

Say that S* is a minimum cut.

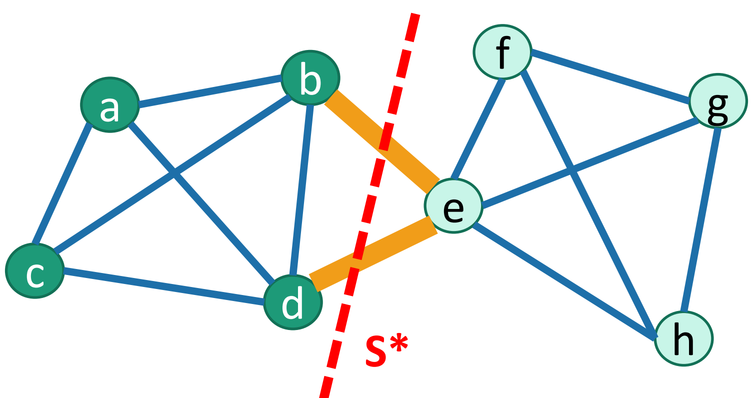
- Suppose the edges that we choose are $e_1, e_2, \ldots, e_{n-2}$
- **PR**[return S*] = **PR**[none of the e_i cross S*]

  = **PR**[e_1 doesn't cross S*]

  × **PR**[e_2 doesn't cross S* | e_1 doesn't cross S*]

  …

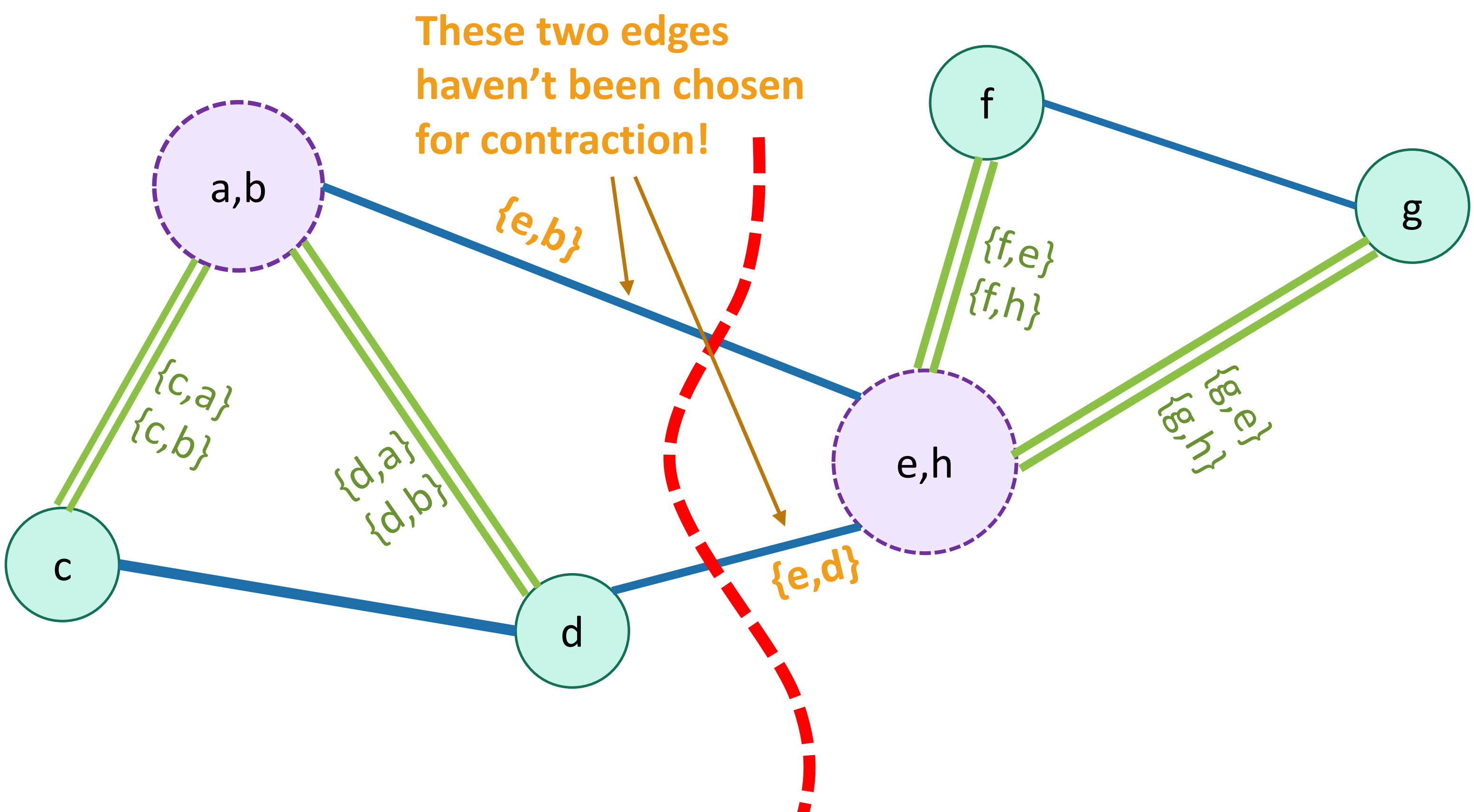
  × **PR**[e_{n-2} doesn't cross S* | $e_1, \ldots, e_{n-3}$ don't cross S*]



Focus in on:

PR[e_j doesn't cross S* | $e_1,\ldots,e_{j-1}$ 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:

PR[e_j doesn't cross S* | $e_1,\ldots,e_{j-1}$ don't cross S*]

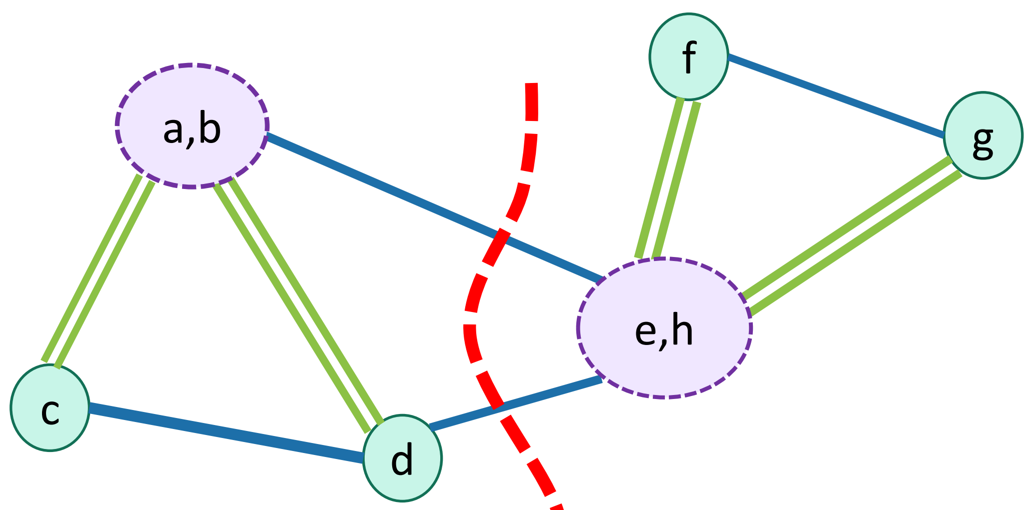
- Suppose: After j-1 iterations, we haven't messed up yet!
- What's the probability of messing up now?

- Say there are k edges that cross S*
- Every remaining node has degree at least k.
  - Otherwise we'd have a smaller cut.
- Thus, there are at least (n-j+1)k/2 edges total.
  - b/c there are n - j + 1 nodes left, each with degree at least k.

Recall: the **degree** of the vertex is the number of edges coming out of it.

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

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



Focus in on:

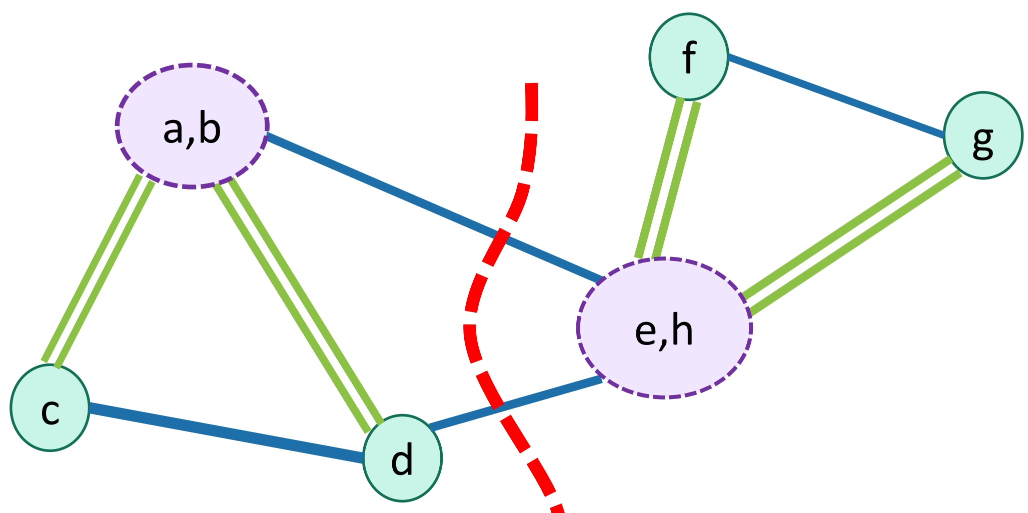
PR[e_j doesn't cross S* | $e_1,\ldots,e_{j-1}$ don't cross S*]

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

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

- 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.

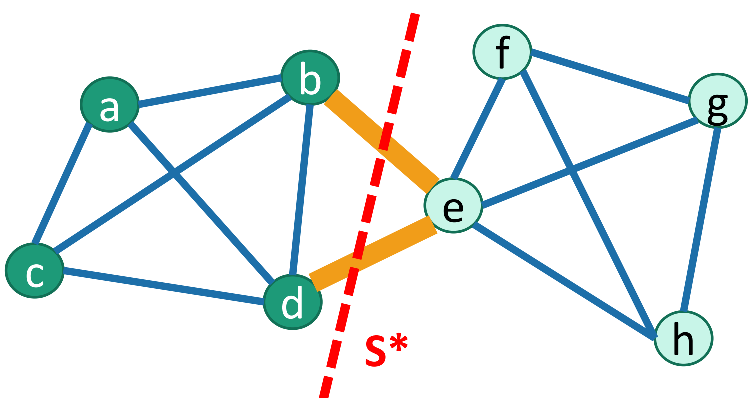
- Suppose the edges that we choose are $e_1, e_2, \ldots, e_{n-2}$
- **PR**[return S*] = **PR**[none of the e_i cross S*]

  = **PR**[e_1 doesn't cross S*]

  × **PR**[e_2 doesn't cross S* | e_1 doesn't cross S*]

  …

  × **PR**[e_{n-2} doesn't cross S* | $e_1,\ldots,e_{n-3}$ don't cross S*]

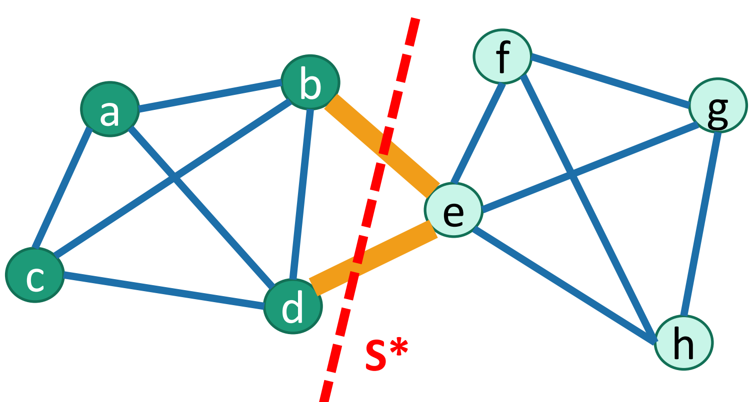


Now let's prove that claim

Say that S* is a minimum cut.

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

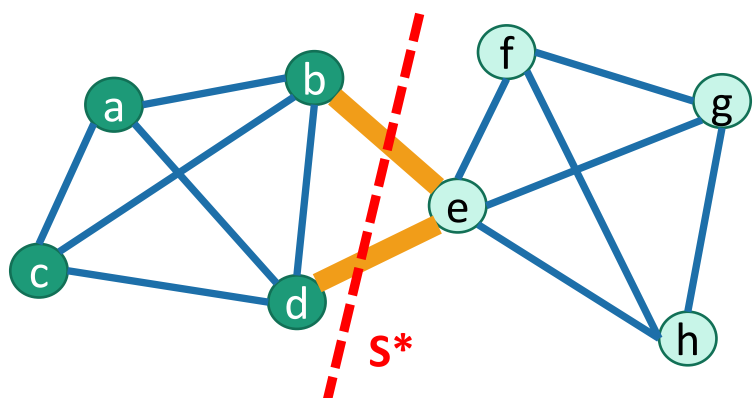
- Suppose the edges that we choose are $e_1, e_2, \ldots, e_{n-2}$
- **PR**[return S*] = **PR**[none of the e_i cross S*]

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

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

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

CLAIM PROVED



Theorem

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

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.

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

- 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.

*Note, in the lecture notes, we take $\delta = \frac{1}{n}$, which makes the running time O(n^4log(n)). It depends on how sure you want to be!

More generally

- Whenever we have a Monte-Carlo algorithm with a small success probability, we can **boost** the success probability by repeating it a bunch and taking the best solution.

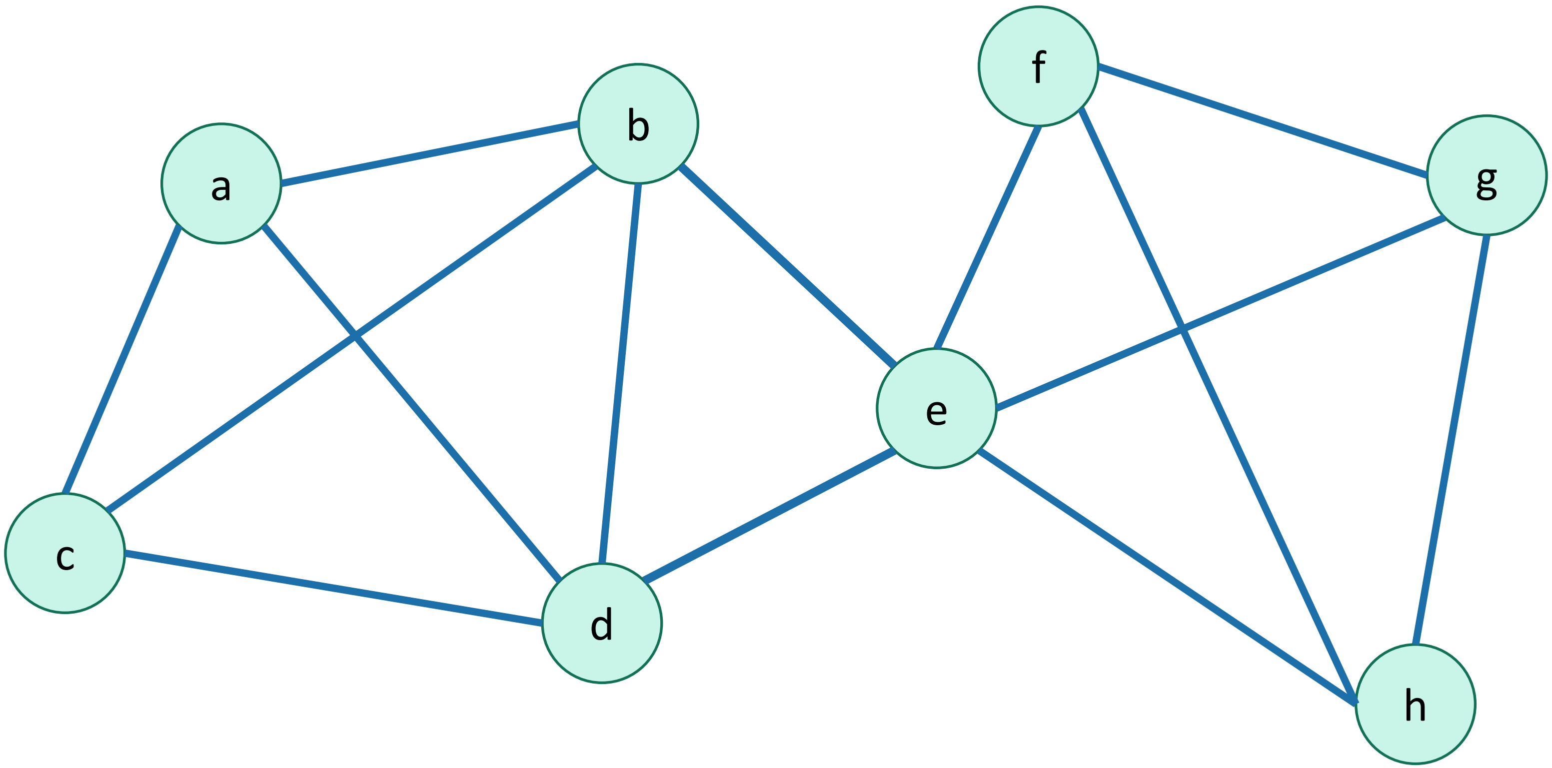
Can we do better?

- 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

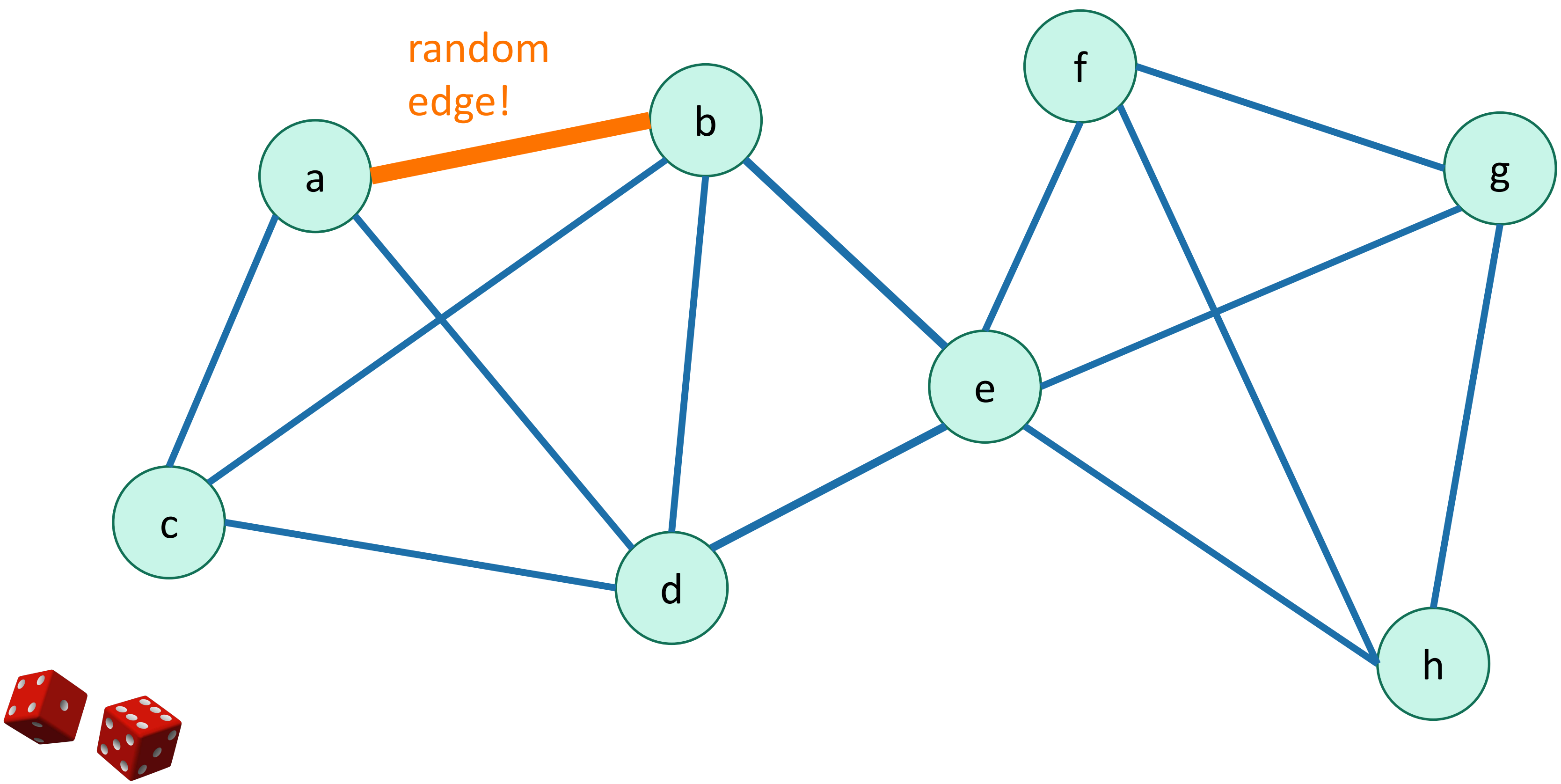


Karger's algorithm

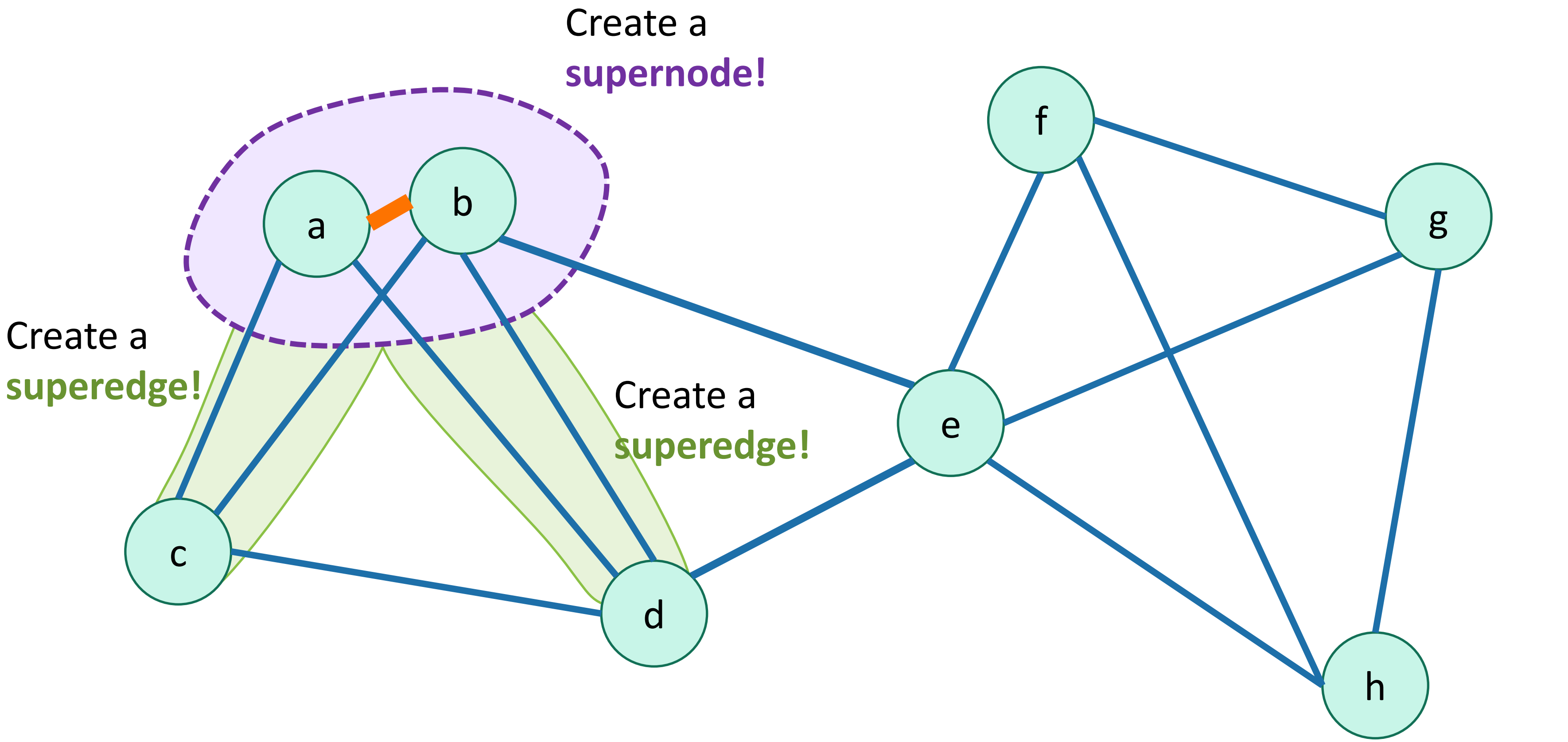
Probability that we didn't mess up:

12/14

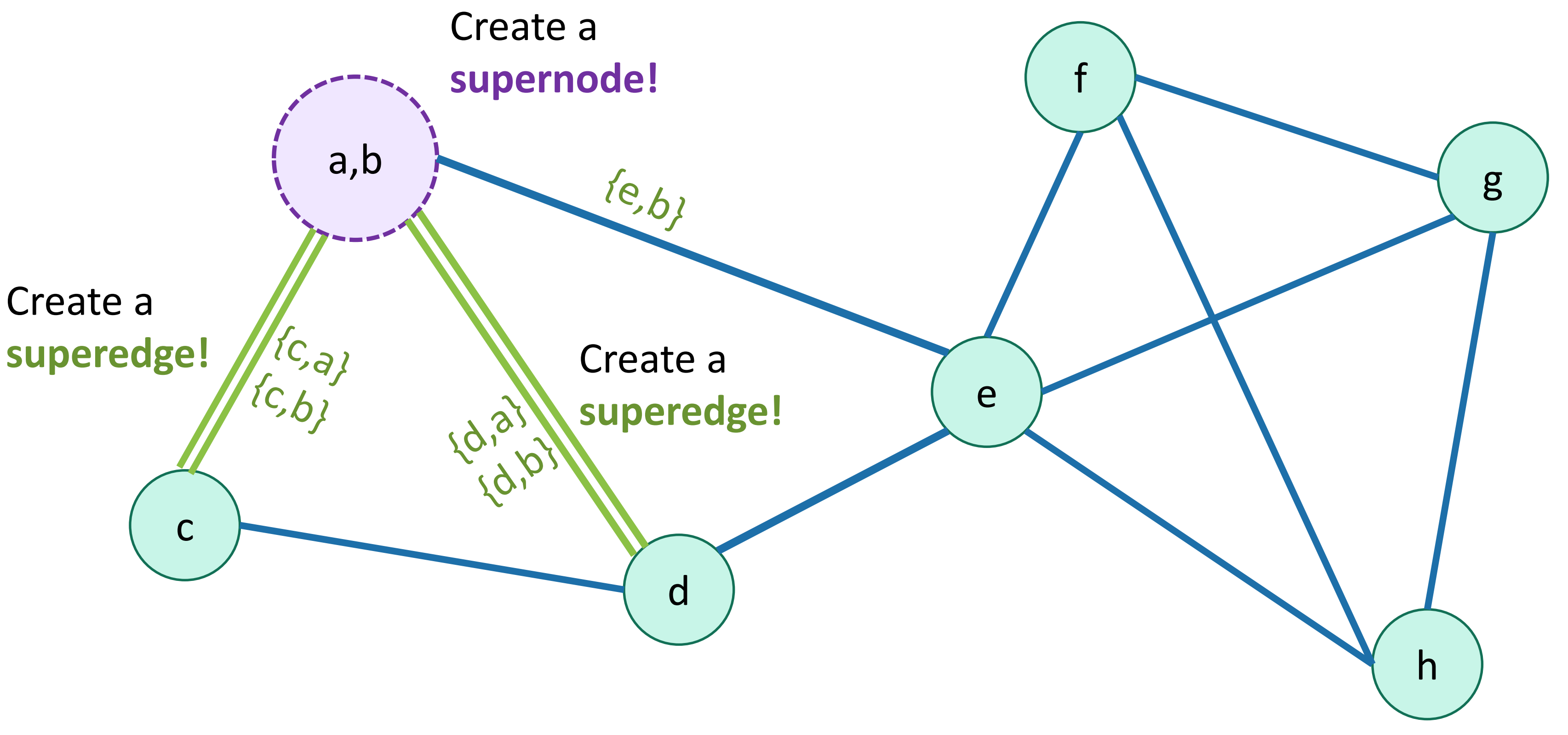
There are 14 edges, 12 of which are good to contract.



Karger's algorithm



Karger's algorithm

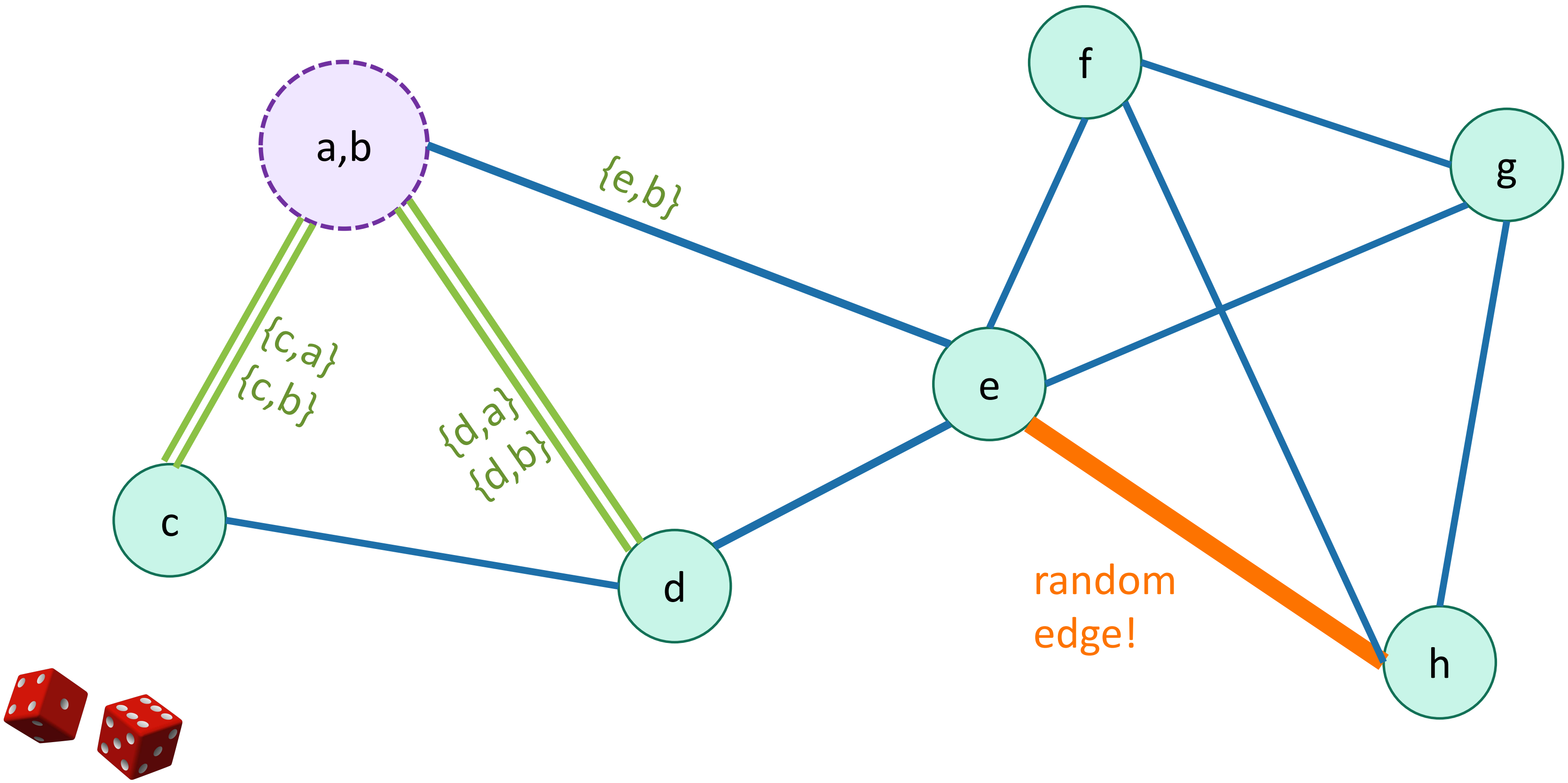


Karger’s algorithm

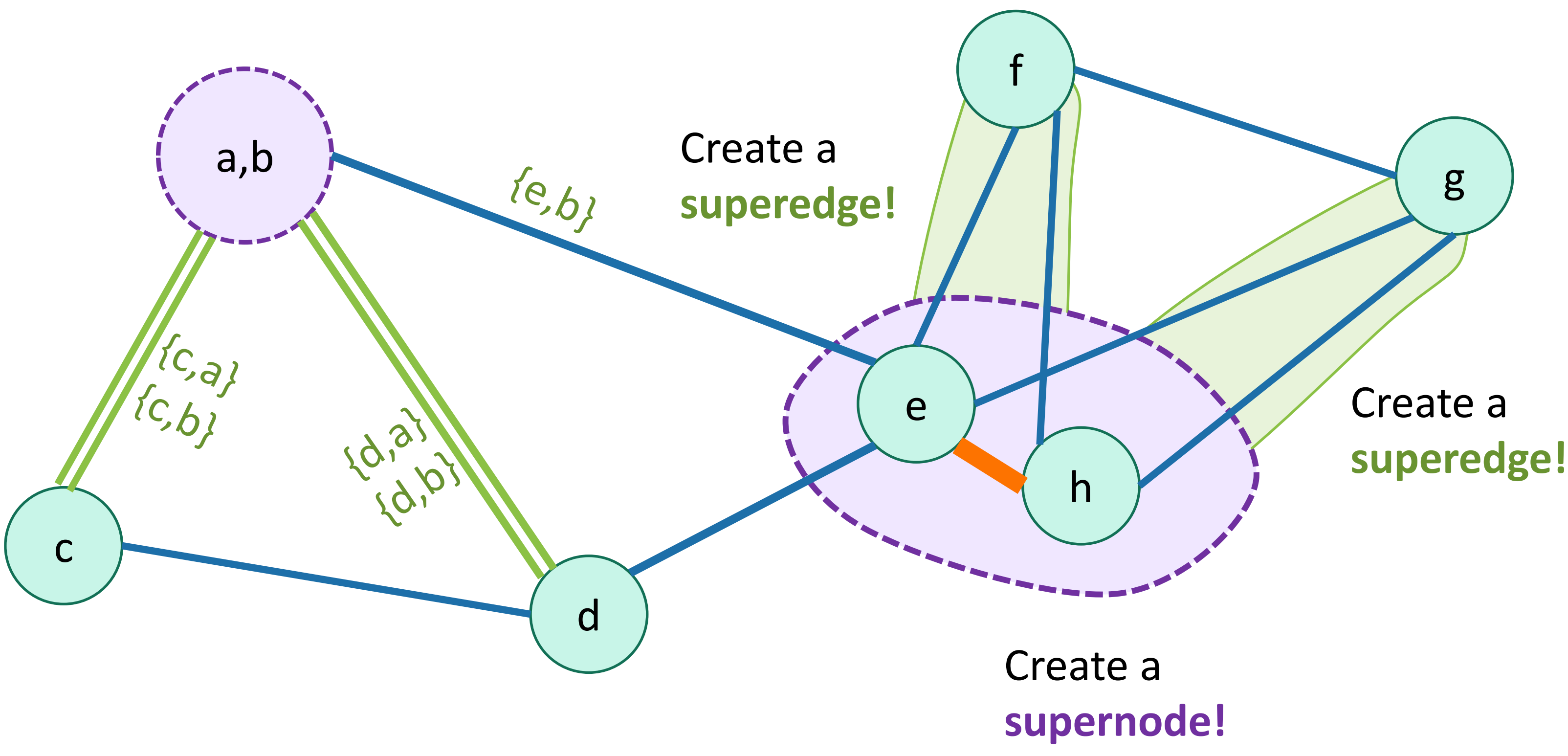
Probability that we didn’t mess up:

11/13

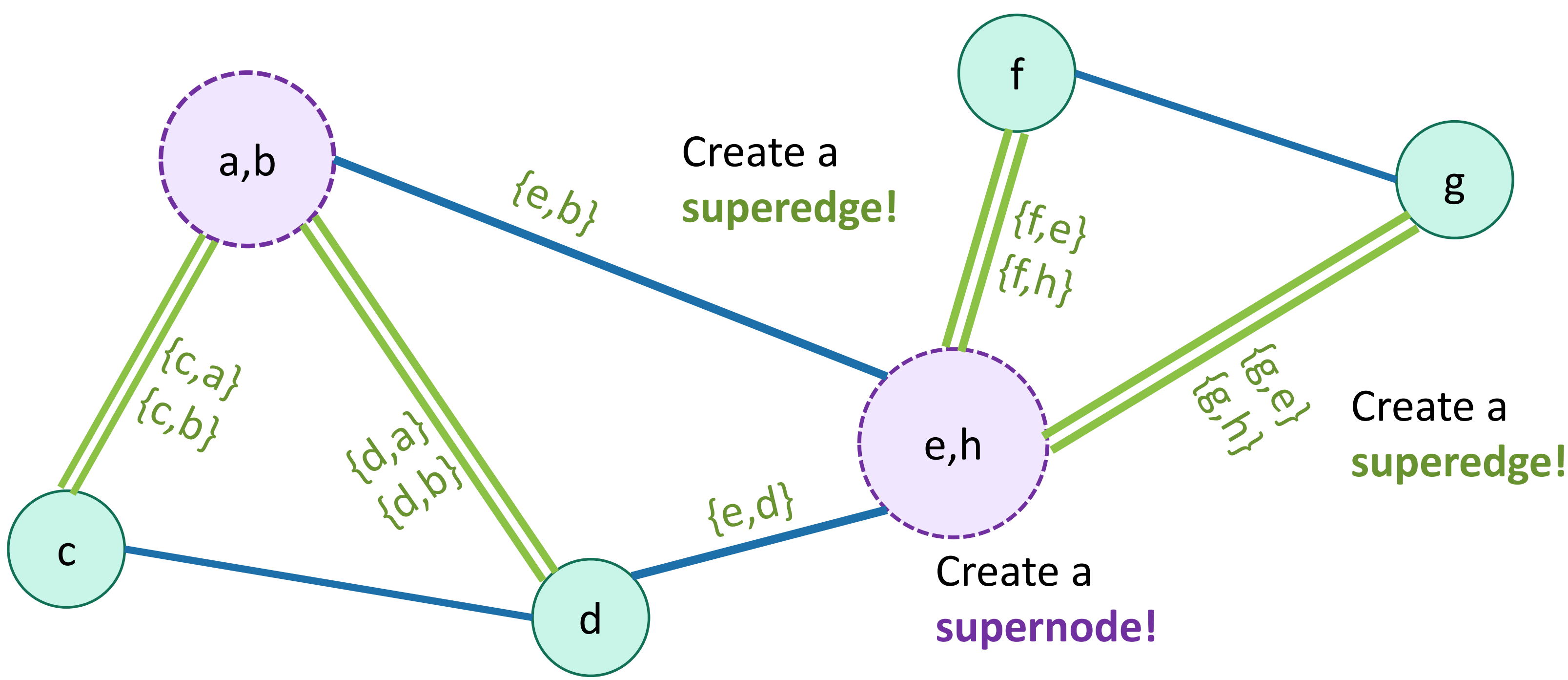
Now there are only 13 edges, since the edge between a and b disappeared.



Karger's algorithm



Karger's algorithm

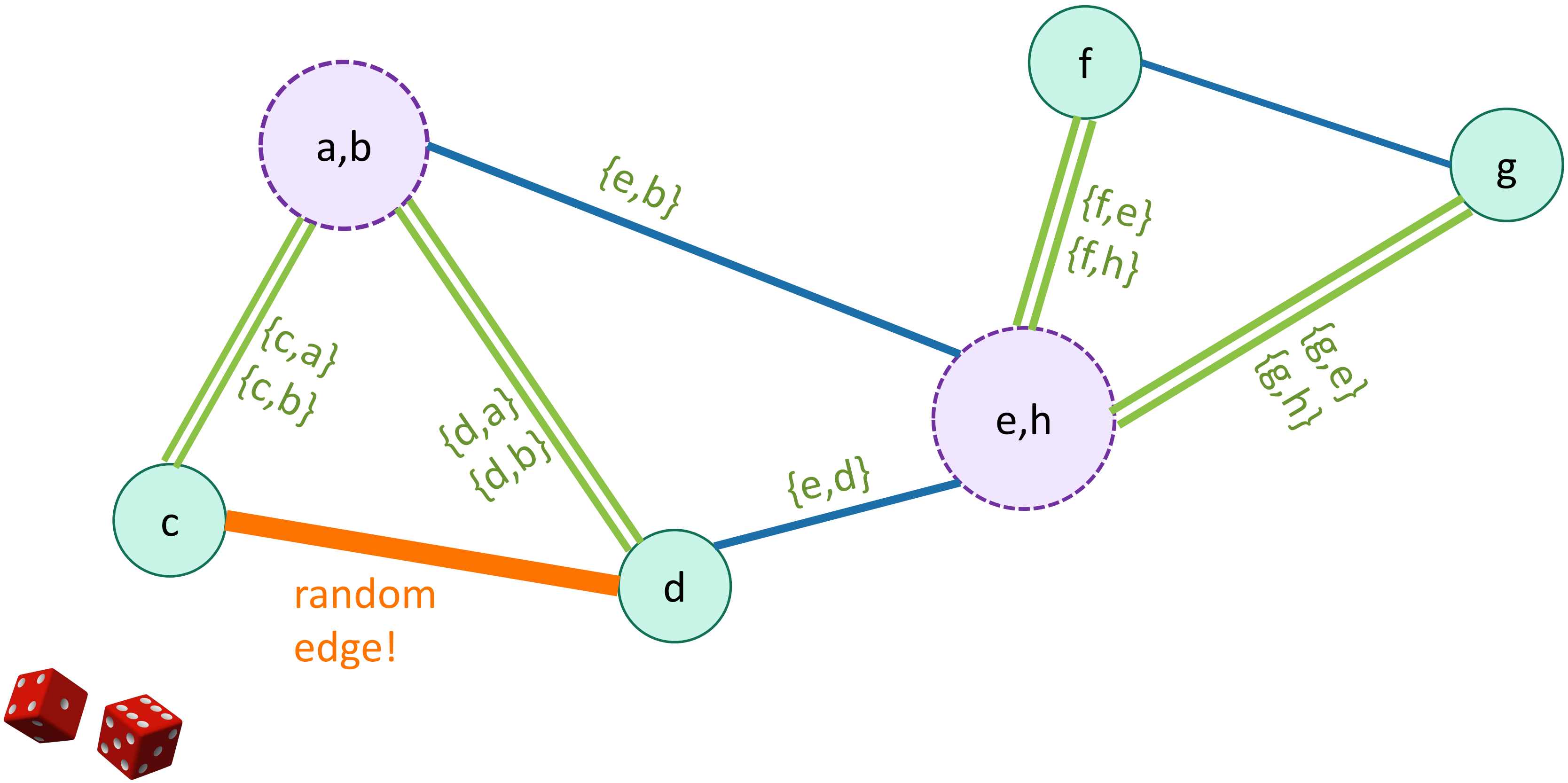


Karger's algorithm

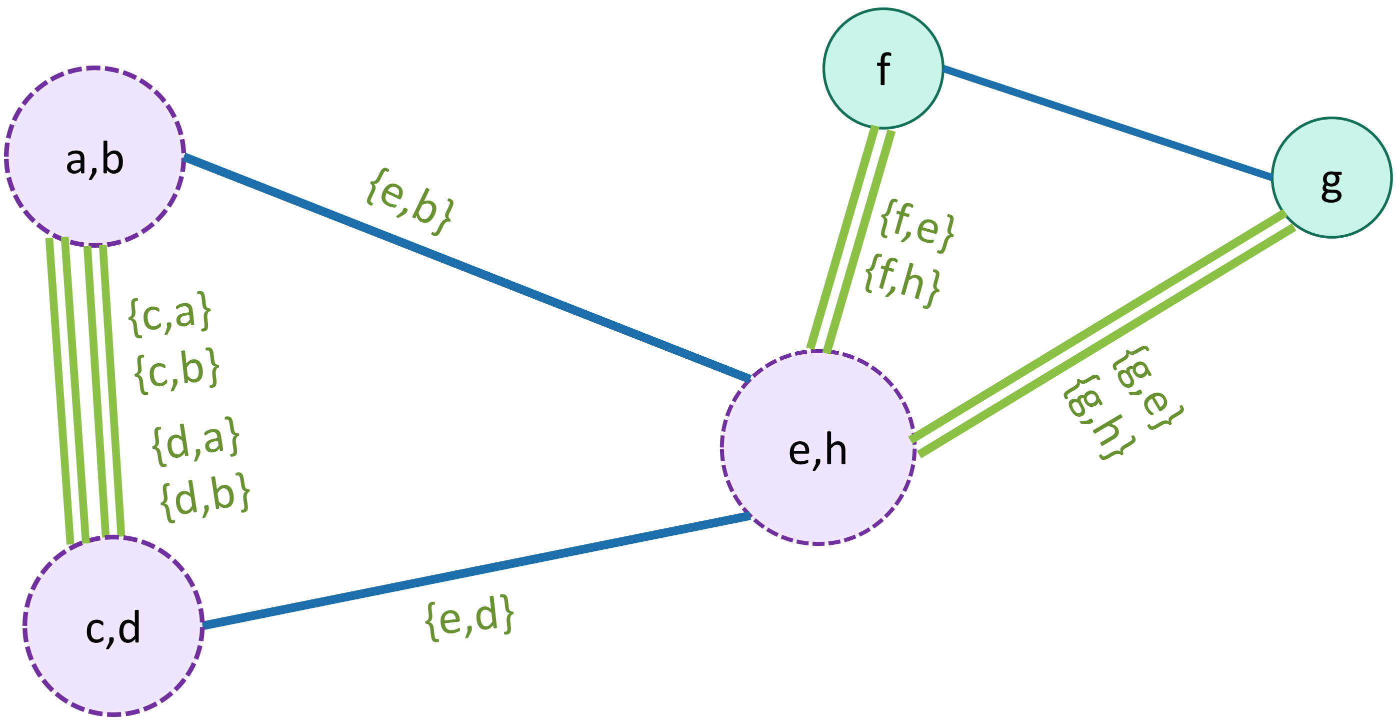
Probability that we didn't mess up:

10/12

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



Karger's algorithm

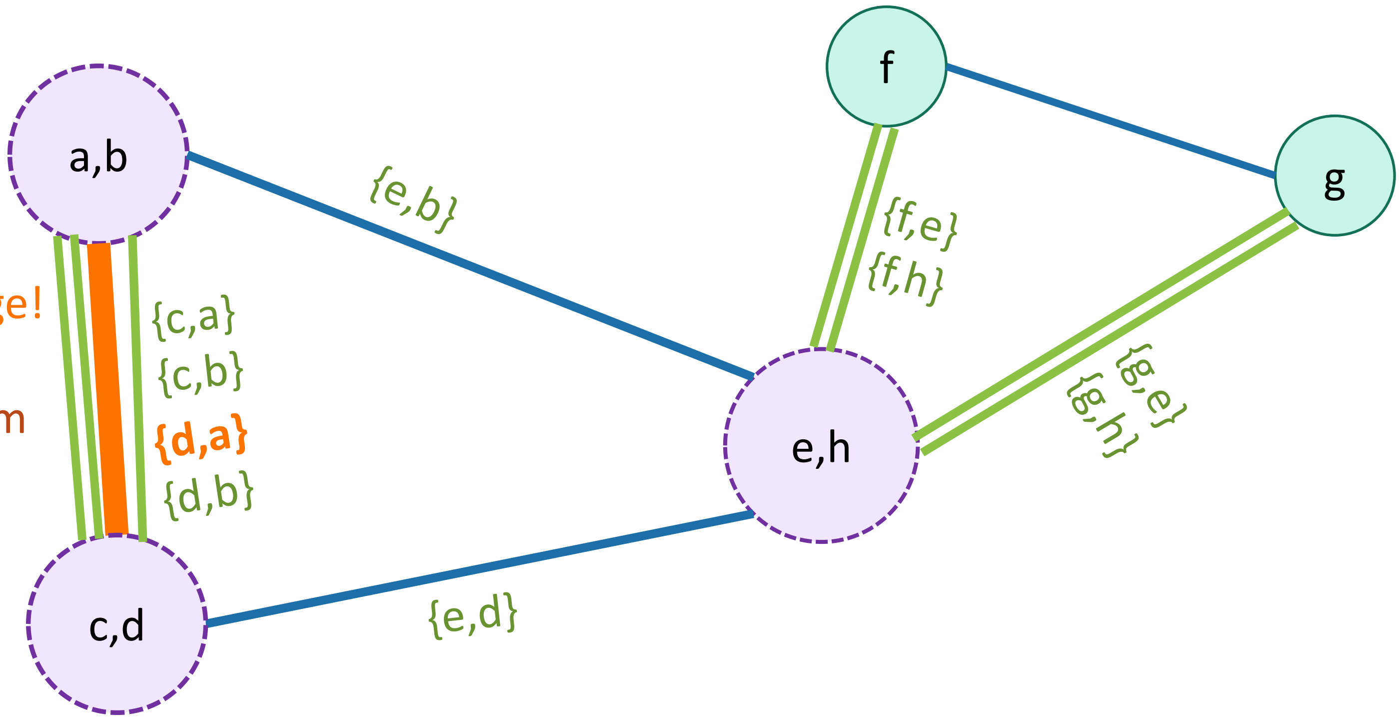


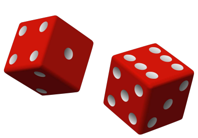
Karger's algorithm

Probability that we didn't mess up:

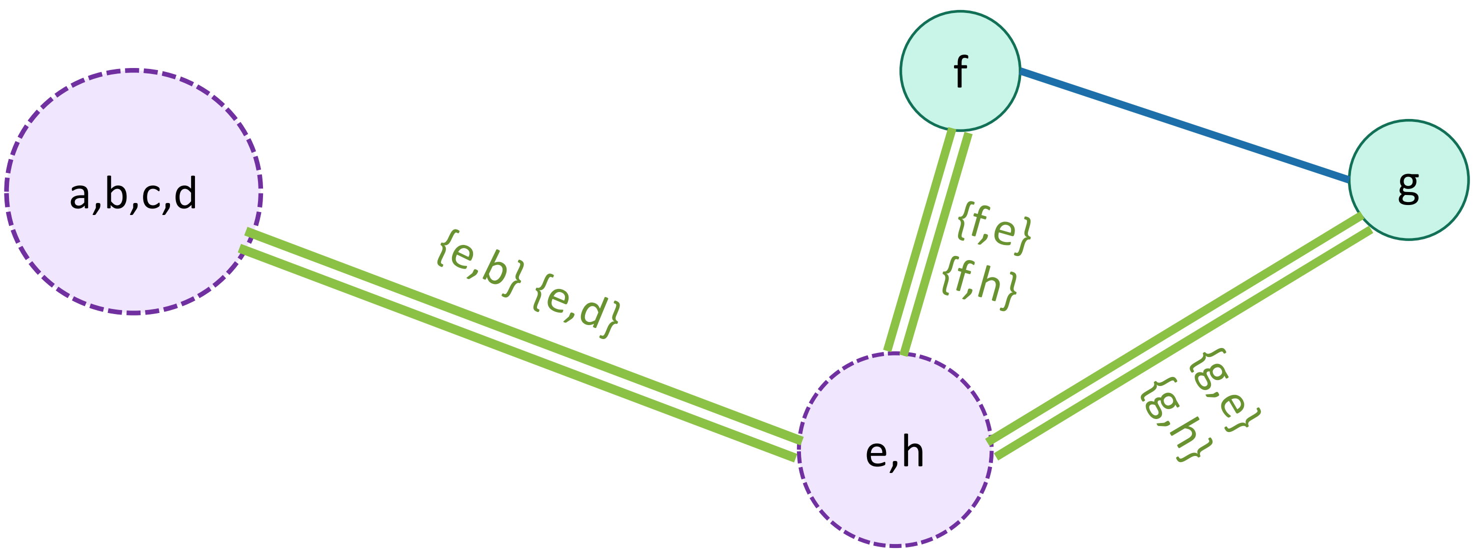
9/11

random edge!
(We pick at random from the original edges).





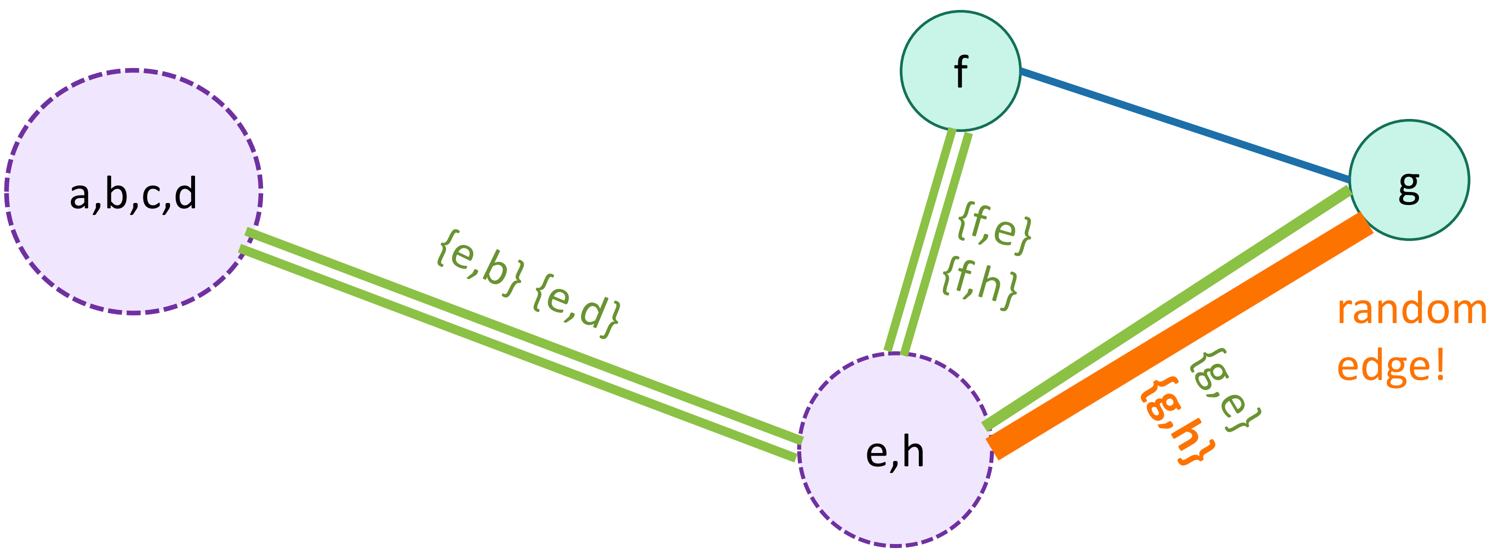
Karger's algorithm



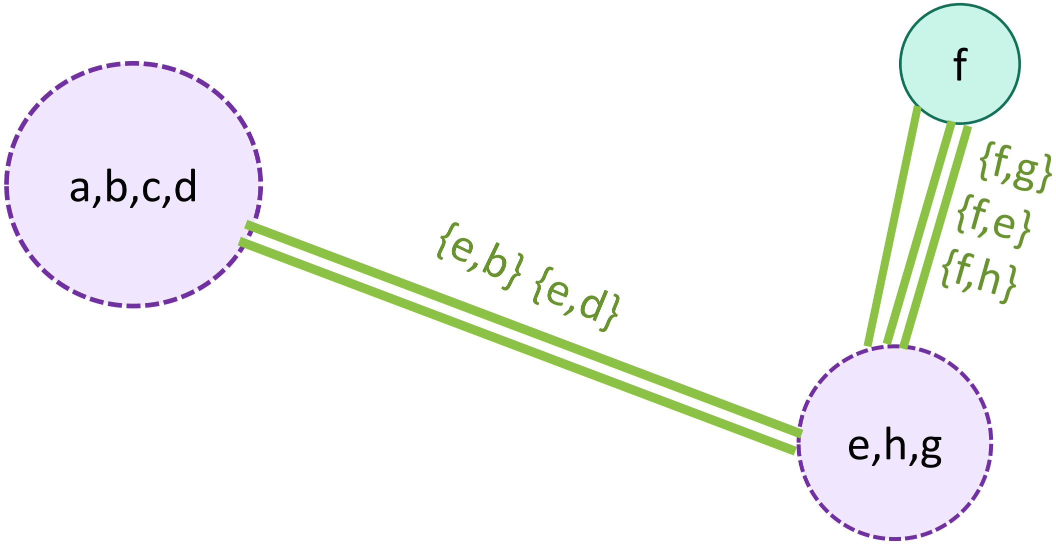
Karger's algorithm

Probability that we didn't mess up:

5/7



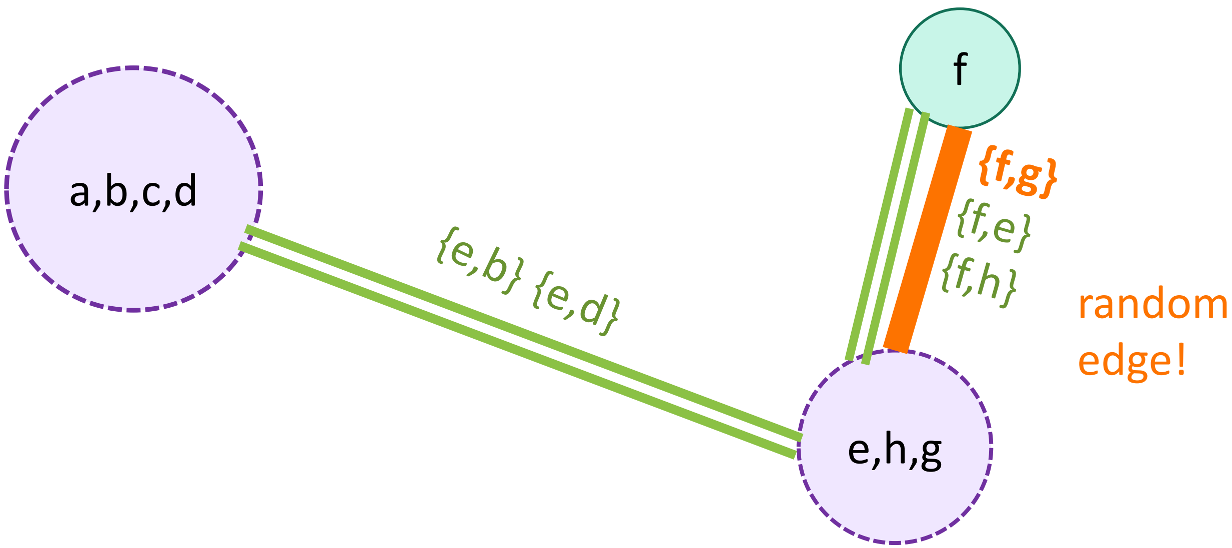
Karger's algorithm

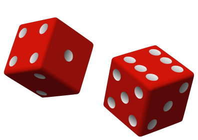


Karger's algorithm

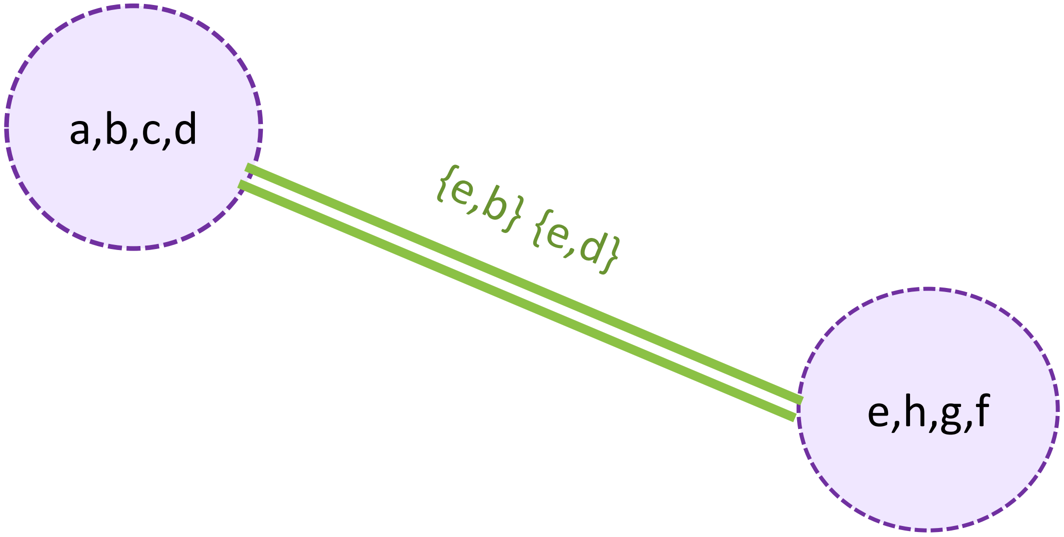
Probability that we didn't mess up:

3/5





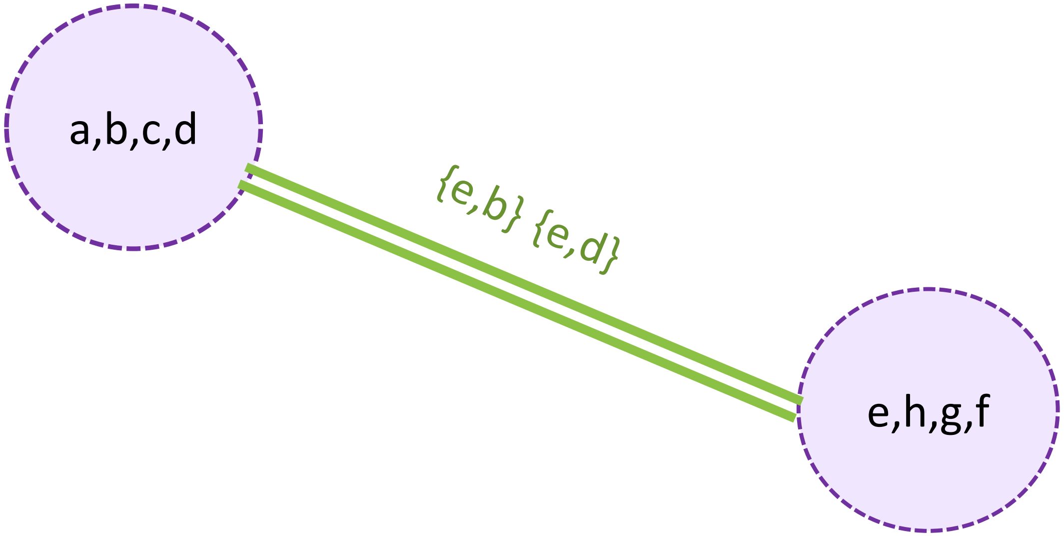
Karger's algorithm



Karger's algorithm

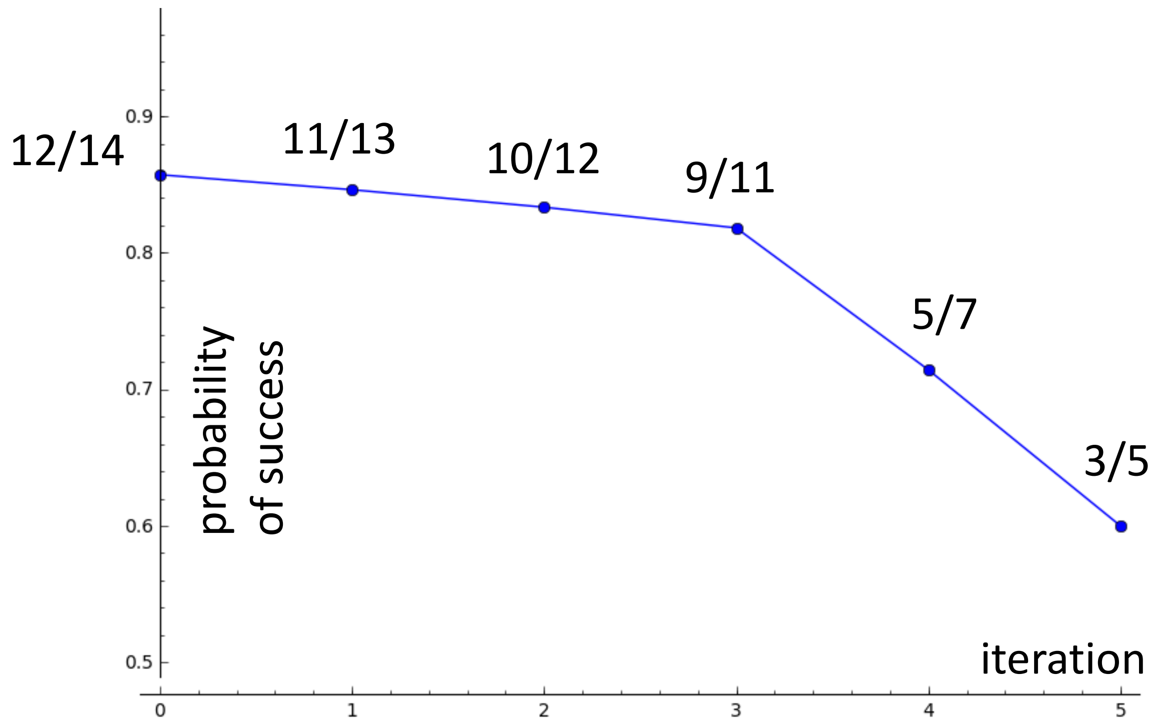
Now stop!

- There are only two nodes left.



Probability of not messing up

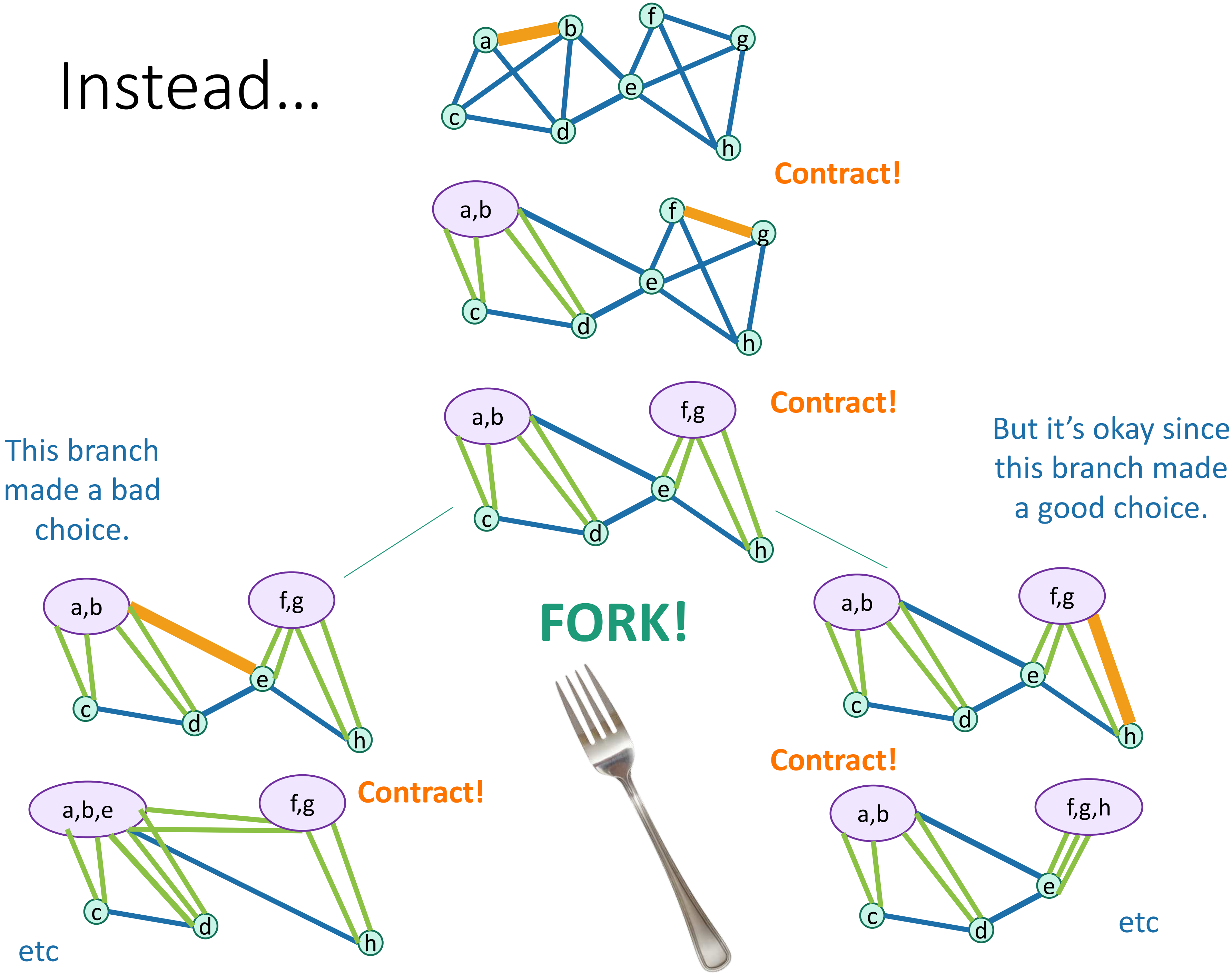
- At the beginning, it's pretty likely we'll be fine.
- The probability that we mess up gets worse and worse over time.



Moral:

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

Instead…


Contract!
Contract!
This branch made a bad choice.
But it's okay since this branch made a good choice.
FORK!
Contract!
Contract!
etc
etc

In words

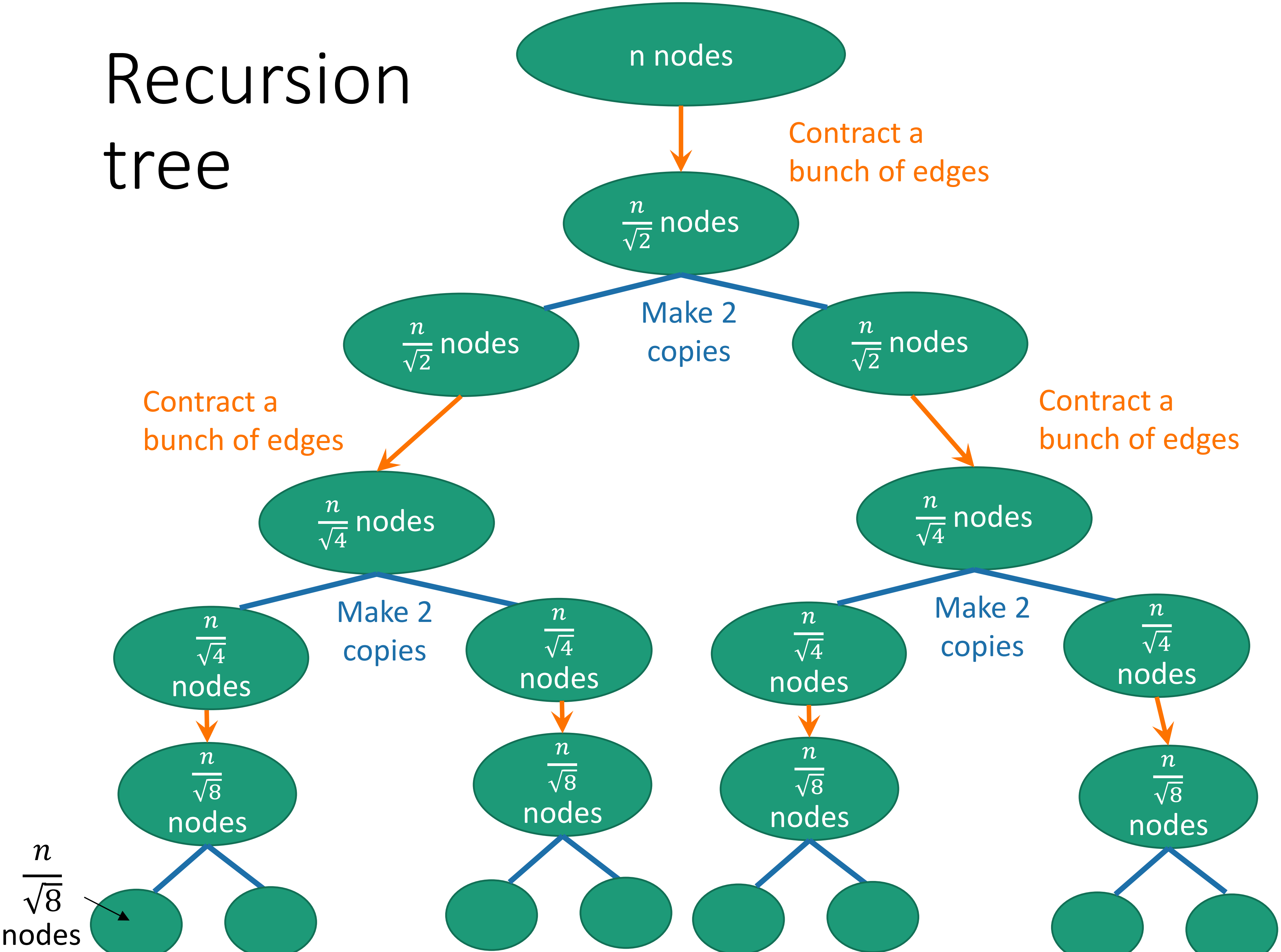
- Run Karger's algorithm on G for a bit.
  - Until there are $\frac{\mathrm{n}}{\sqrt{2}}$ supernodes left.
- Then split into two independent copies, G_1 and G_2
- Run Karger's algorithm on each of those for a bit.
  - Until there are $\frac{\left(\frac{\mathrm{n}}{\sqrt{2}}\right)}{\sqrt{2}} = \frac{n}{2}$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

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

In pseudocode

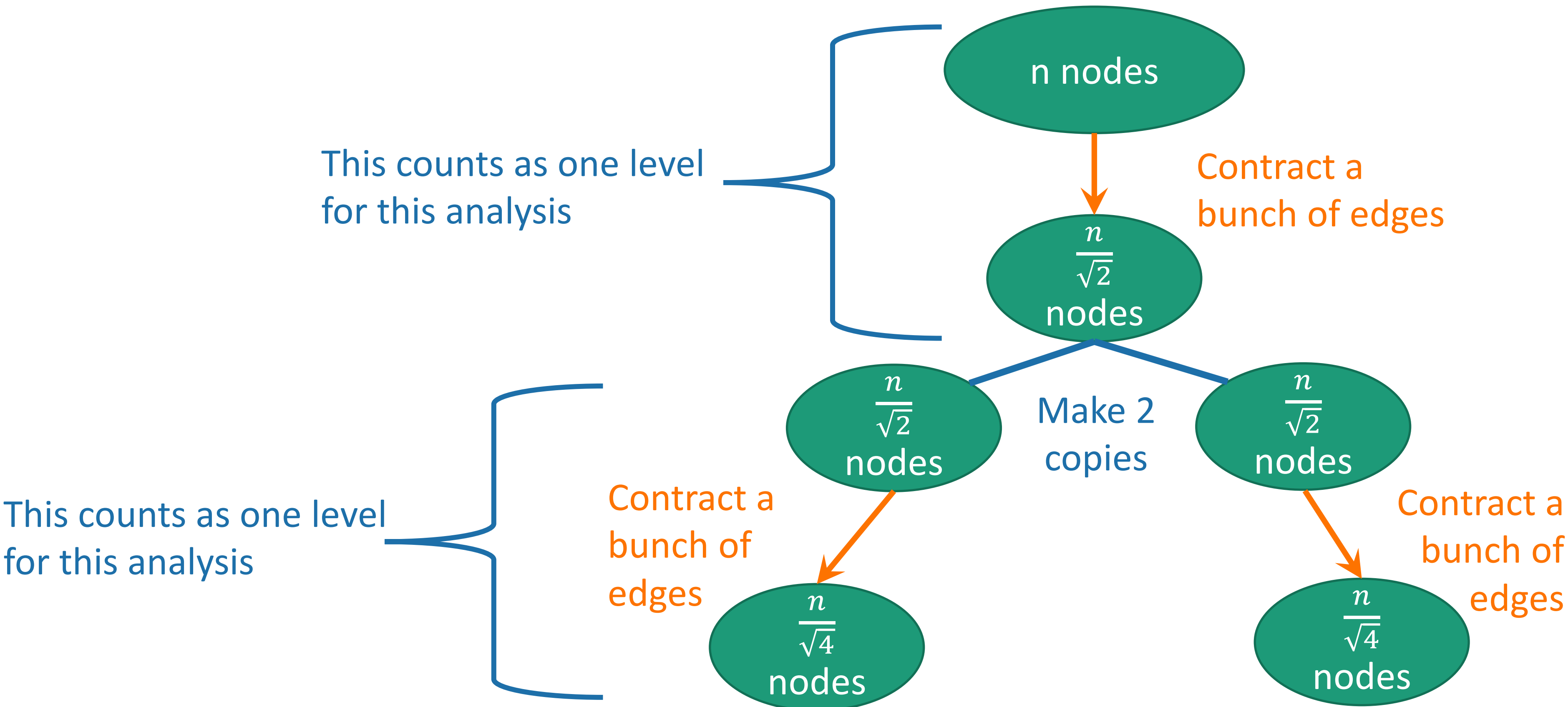
- **KargerStein**(G = (V,E)):
  - n ← |V|
  - if n < 4:
    - find a min-cut by brute force \\\\ time O(1)
  - Run Karger's algorithm on G with independent repetitions until $\left\lfloor \frac{n}{\sqrt{2}} \right\rfloor$ nodes remain.
  - G_1, G_2 ← copies of what's left of G
  - S_1 = **KargerStein**(G_1)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

Recursion tree



Recursion tree

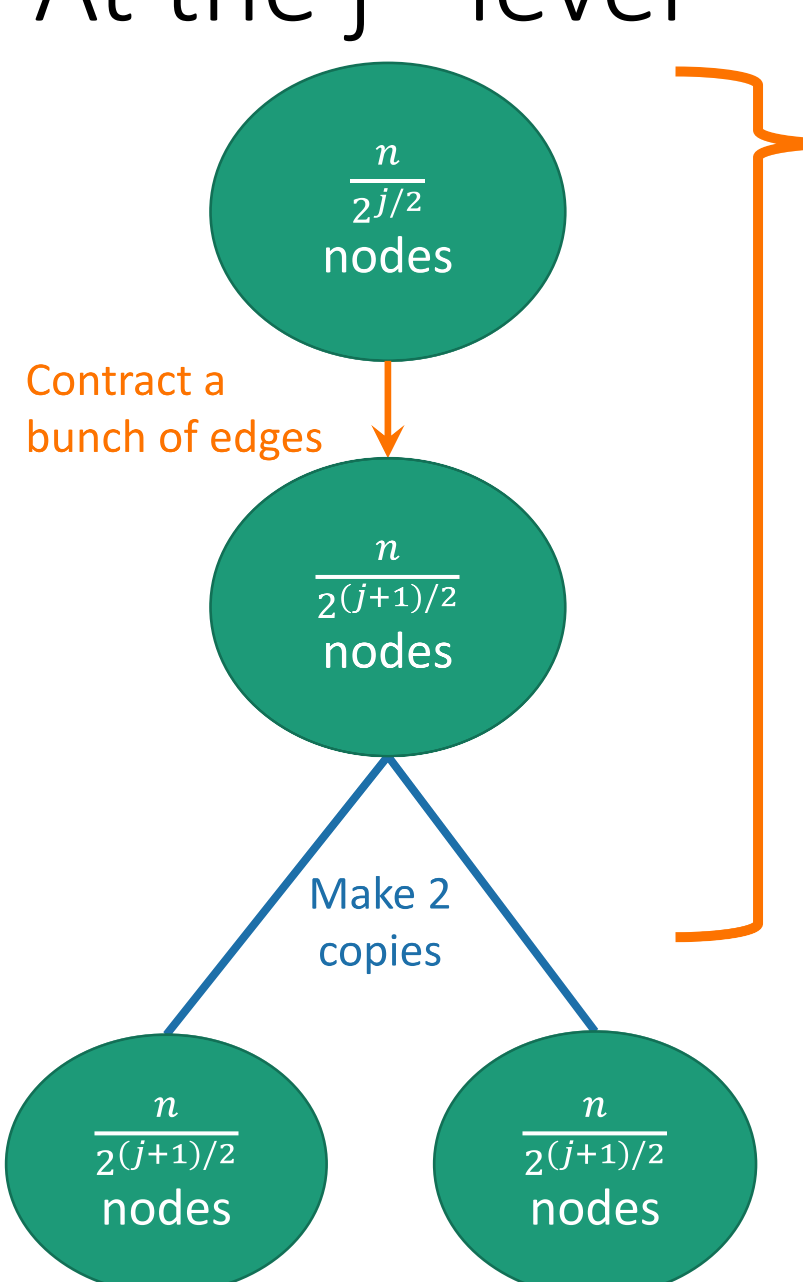
- depth is $\log_{\sqrt{2}}(n) = \frac{\log(n)}{\log(\sqrt{2})} = 2\log(n)$
- number of leaves is $2^{2\log(n)} = n^2$



Two questions

- Does this work?
- Is it fast?

At the j^{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

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

- **PR**[none of the e_i cross S* (up to the n-t'th)]

  = **PR**[e_1 doesn't cross S*]

  $\times$ **PR**[e_2 doesn't cross S* | e_1 doesn't cross S*]

  …

  $\times$ **PR**[e_{n-t} doesn't cross S* | $e_1,\ldots,e_{n-t-1}$ don't cross S*]

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

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

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

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

- **PR**[none of the e_i cross S* (up to the n-t'th)]

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

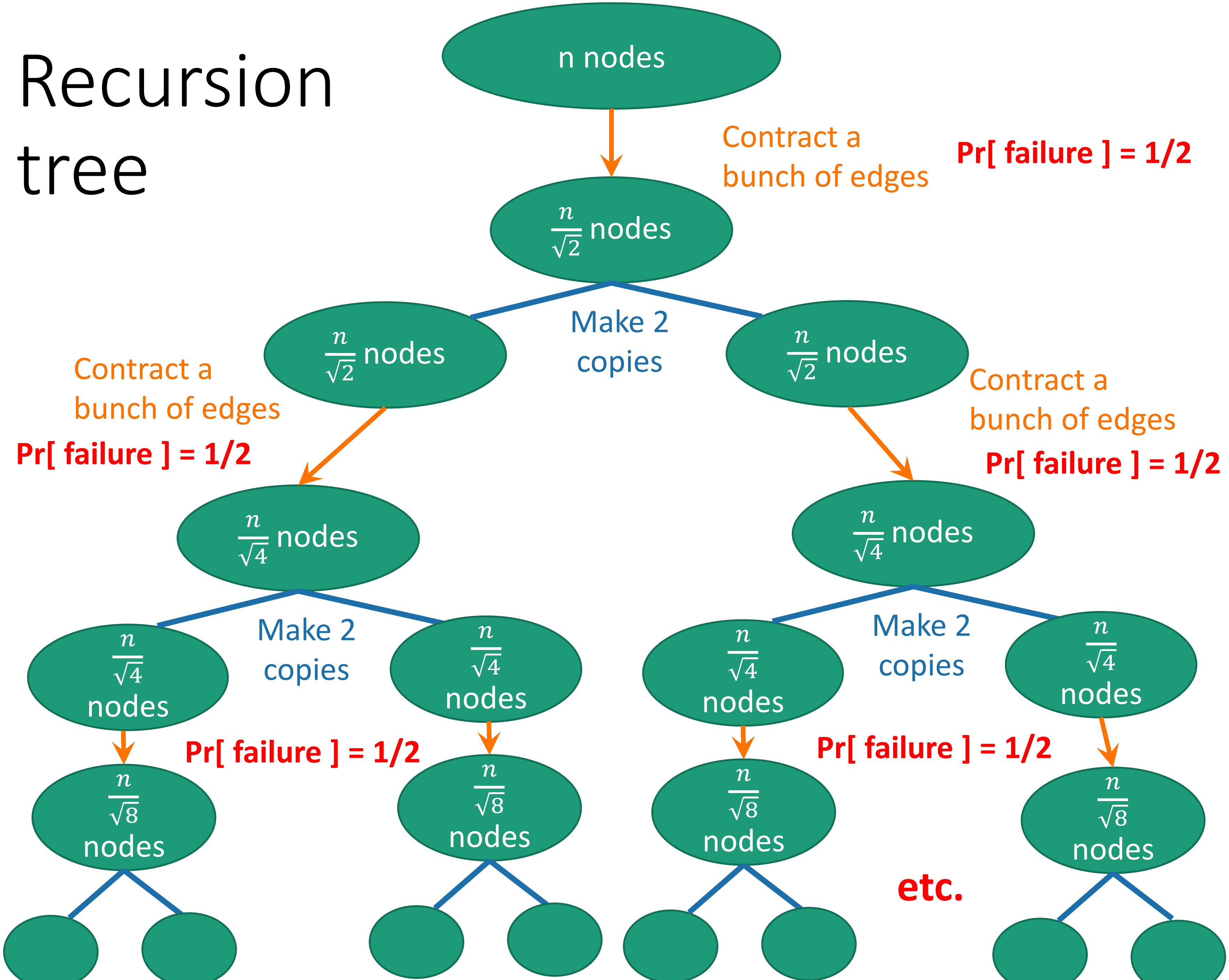
$$= \frac{t\cdot(t-1)}{n\cdot(n-1)}$$

Choose $\boldsymbol{t = n/\sqrt{2}}$

$$= \frac{\frac{n}{\sqrt{2}}\cdot\left(\frac{n}{\sqrt{2}}-1\right)}{n\cdot(n-1)} \approx \frac{\mathbf{1}}{\mathbf{2}}$$

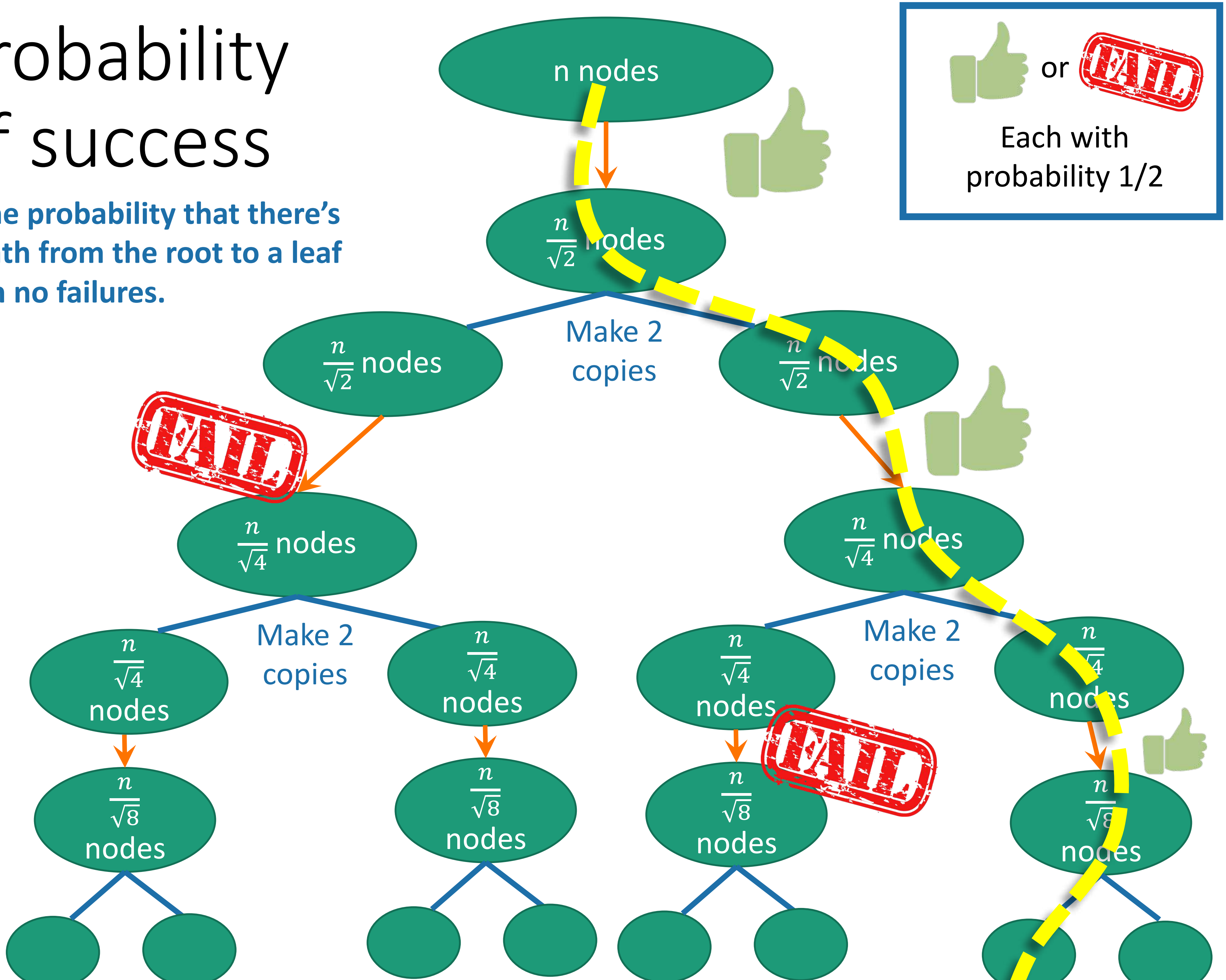
when n is large

Recursion tree



Probability of success

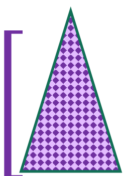
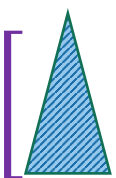
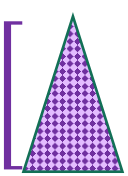
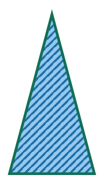
Is the probability that there's a path from the root to a leaf with no failures.

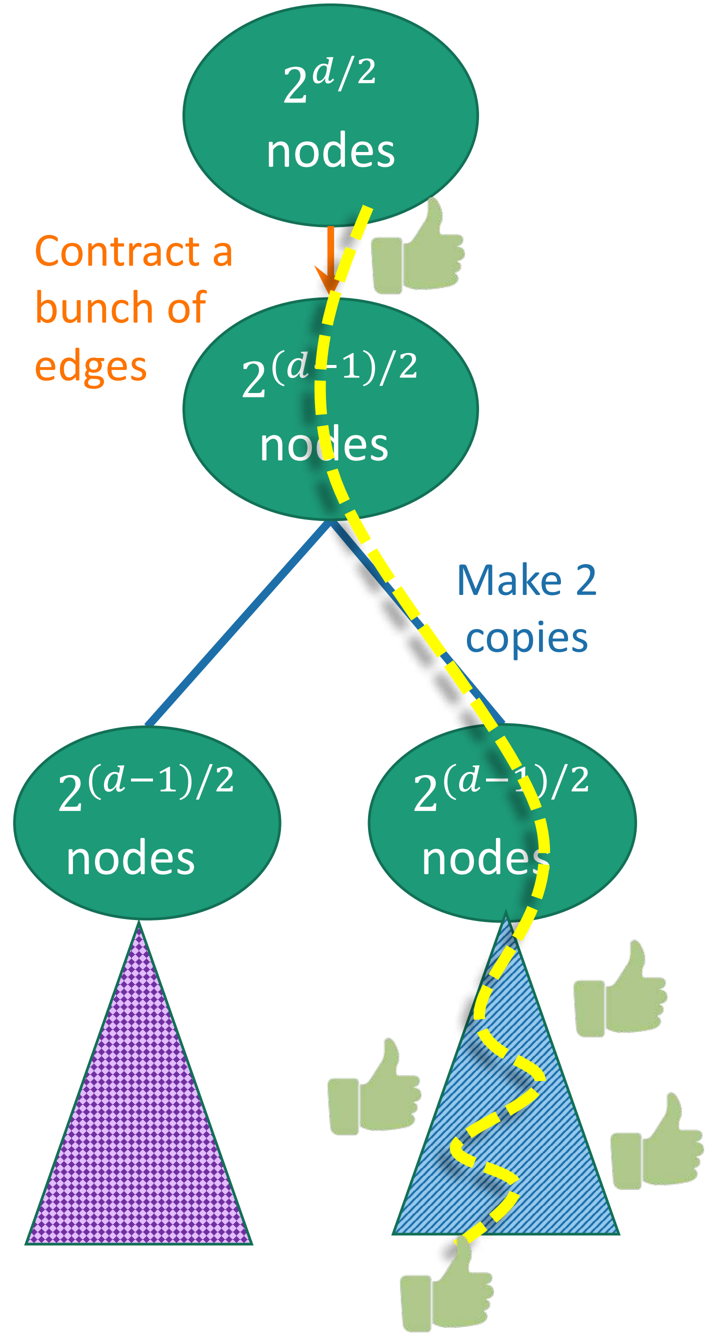


The problem we need to analyze

- Let T be binary tree of depth 2log(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

- Say the tree has height d.
- Let $\boldsymbol{p_d}$ be the probability that there's a path from the root to a leaf that **doesn't fail**.
- $p_d = \frac{1}{2} \cdot \Pr\left[\begin{array}{l} \text{at least one subtree} \\ \text{has a successful path} \end{array} \right]$
- $= \frac{1}{2} \cdot \Big($ $\Pr\big[$  wins $\big] + \Pr\big[$  wins $\big]$ $- \Pr\big[$   both win $\big]$ $\Big)$
- $= \frac{1}{2} \cdot \left(p_{d-1} + p_{d-1} - p_{d-1}^2 \right)$
- $= p_{d-1} - \frac{1}{2} \cdot p_{d-1}^2$

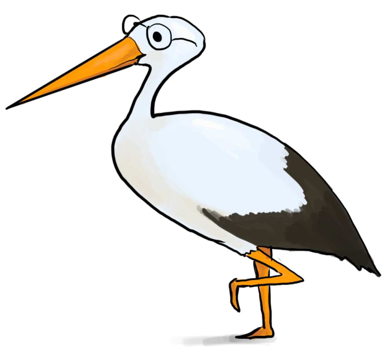


It’s a recurrence relation!

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

- We are real good at those.
- In this case, the answer is:
  - **Claim**: for all d, $p_d \geq \frac{1}{d+1}$

Prove this! (Or see hidden slide for a proof).



Siggi the Studious Stork

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 $p_{d-1} \geq \frac{1}{d}$.
    - $p_d = p_{d-1} - \frac{1}{2} \cdot p_{d-1}^2$
    - $\geq \frac{1}{d} - \frac{1}{2} \cdot \frac{1}{d^2}$
    - $\geq \frac{1}{d} - \frac{1}{d(d+1)}$
    - $= \frac{1}{d+1}$

This slide skipped in class

What does that mean for Karger-Stein?

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

- For d = 2log(n)
  - that is, d = the height of the tree:

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

- aka,

Pr[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 $1 - \delta$.
- 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

$$O\left(n^2 \log(n) \cdot \log(n)\right) = O(n^2 \log(n)^2)$$

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

What have we learned?

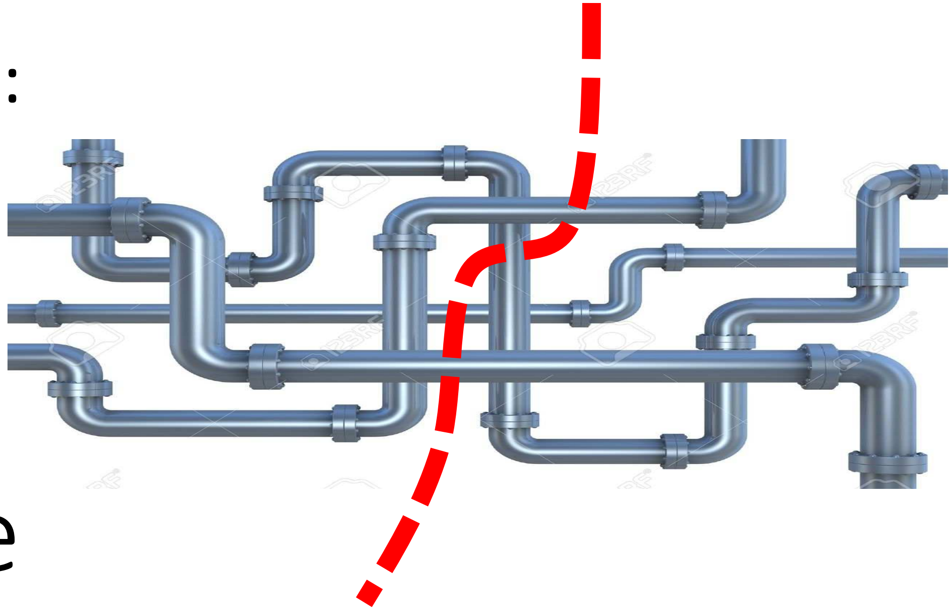
- Just repeating Karger's algorithm isn't the best use of repetition.
  - We're probably going to be correct near the beginning.
- Instead, Karger-Stein repeats when it counts.
  - If we wait until there are $\frac{n}{\sqrt{2}}$ nodes left, the probability that we fail is close to ½.
- This lets us find 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.

Next time

- Another sort of min-cut:
  - s-t min-cut
  - also max-flow!



Before next time

- Pre-lecture exercise: examples of cuts and flows.