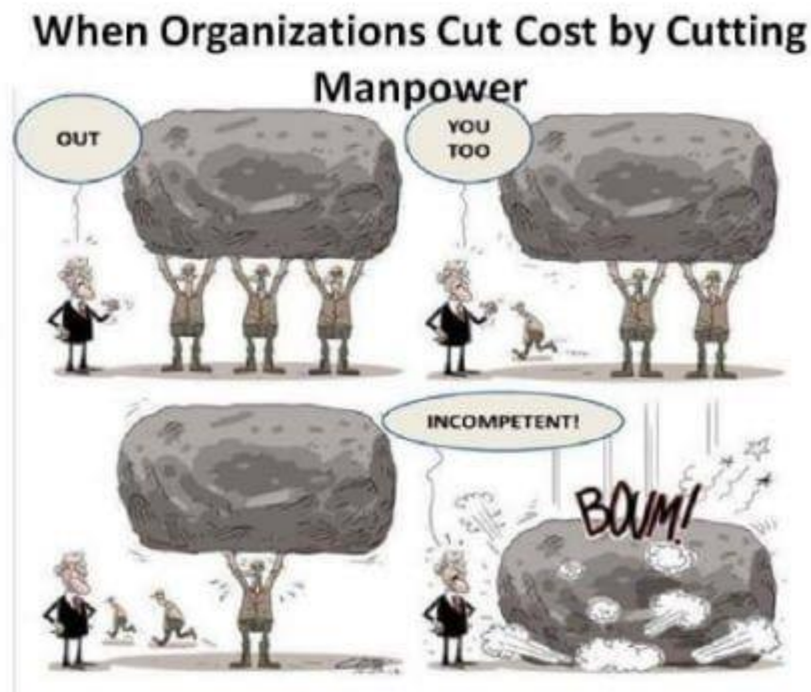
CSE100: Design and Analysis of Algorithms
Lecture 26 – Min Cut and Karger's Algorithm (wrap up)

May 3rd 2022

Min Cut, Karger and Karger-Stein's Algorithms,

Today

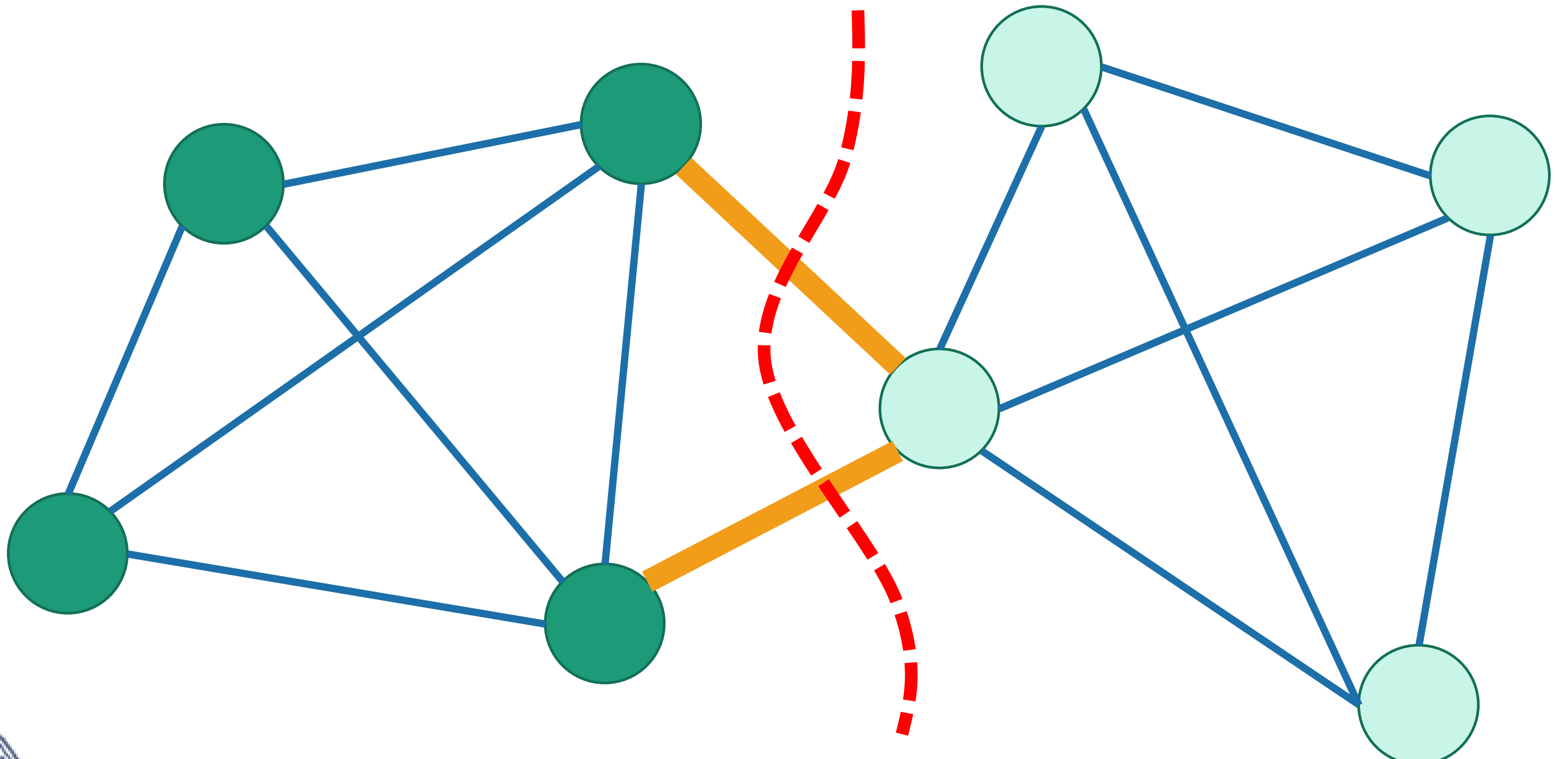
- Minimum Cuts!
  - Karger's algorithm
  - Karger-Stein algorithm



  - Back to **randomized algorithms!**
    - but in a different way than we've seen so far

A (global) minimum cut (review)

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



Karger's Algorithm (review)

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

- **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

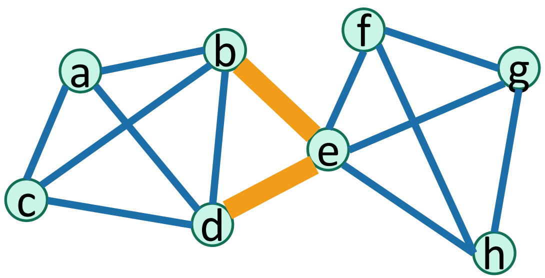
merge takes time O(n) naively

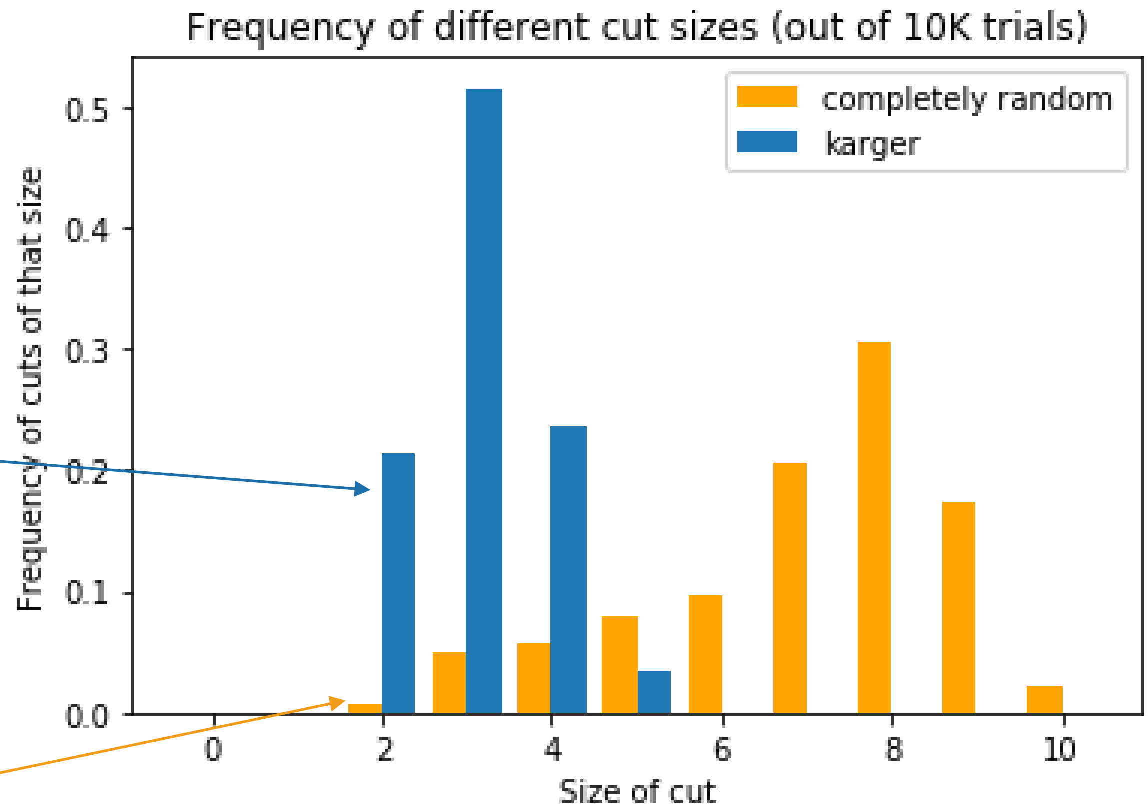
- **merge**(u, v): // merge also knows about Γ and the $E_{\overline{\boldsymbol{u}},\overline{\boldsymbol{v}}}$'s
  - $\overline{\boldsymbol{x}}$ = SuperNode($\overline{\boldsymbol{u}} \cup \overline{\boldsymbol{v}}$) // create a new supernode
  - for each $\overline{\boldsymbol{w}}$ in $\Gamma \setminus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

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.

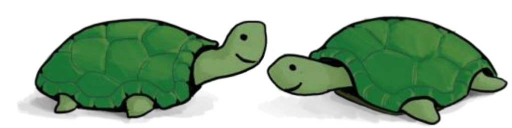
Thought experiment (review)

- 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?

6 3 3 2 5 2 5

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?



Let’s calculate the probabilities (review)

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:

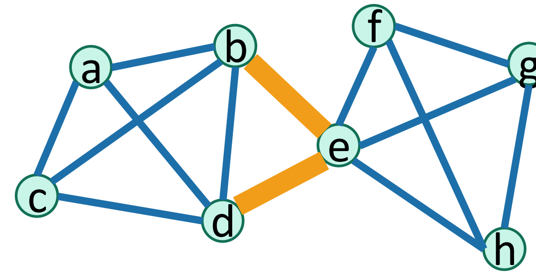
- Pr[We push the button t times and don’t ever get the min] = $(1 - 0.2)^t$
- Pr[We push the button 5 times and don’t ever get the min] = $(1 - 0.2)^5 \approx 0.33$
- Pr[We push the button 10 times and don’t ever get the min] = $(1 - 0.2)^{10} \approx 0.1$

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! (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!

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.

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 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?

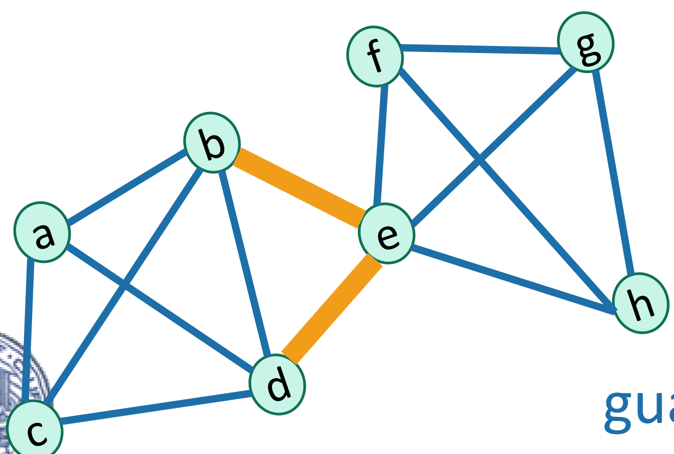
2. How many times should we run Karger's algorithm to "probably" succeed?
   - Say, with probability 0.99?
   - Or more generally, probability $1 - \delta$?

Answer to Question 1

Claim:

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

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



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

Questions

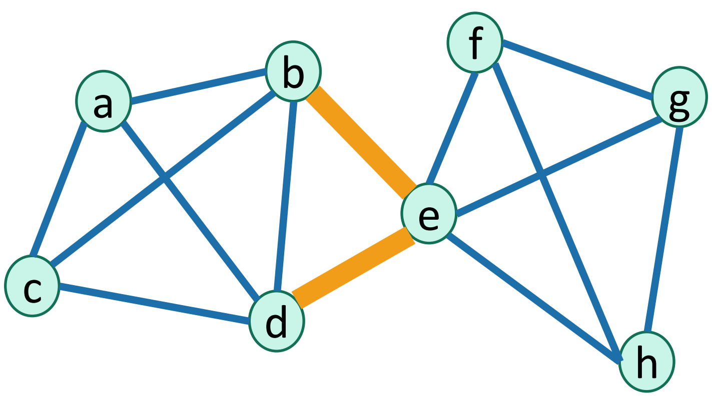
1. What is the probability that Karger's algorithm returns a minimum cut?

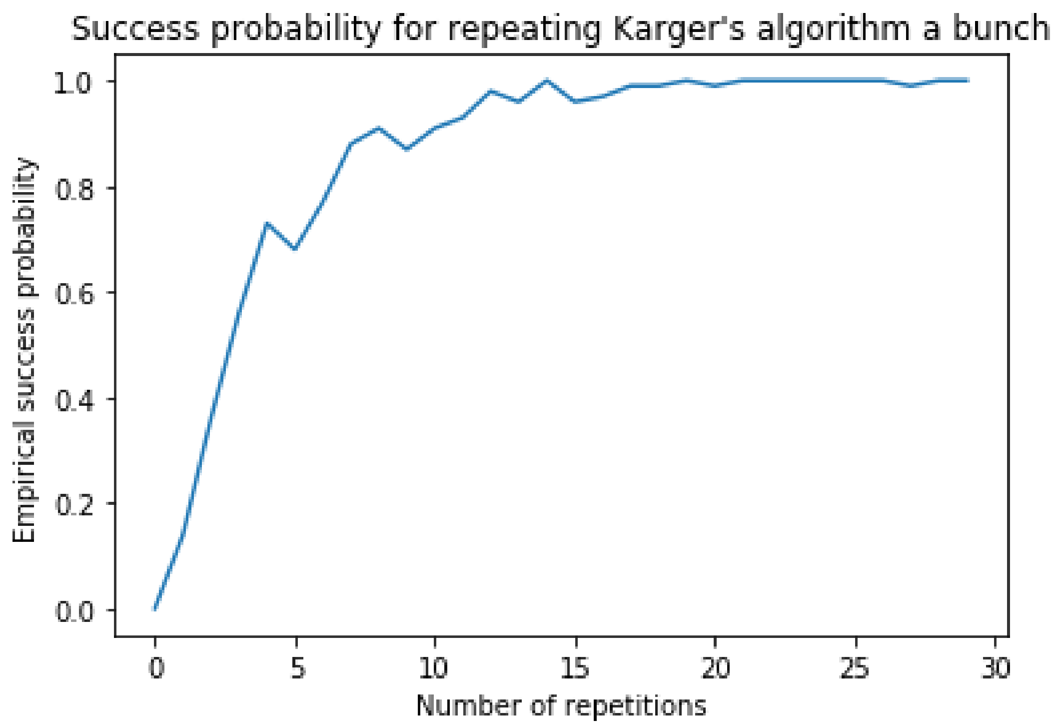
   According to the claim, at least $\frac{1}{\binom{n}{2}}$

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 $1 - \delta$?





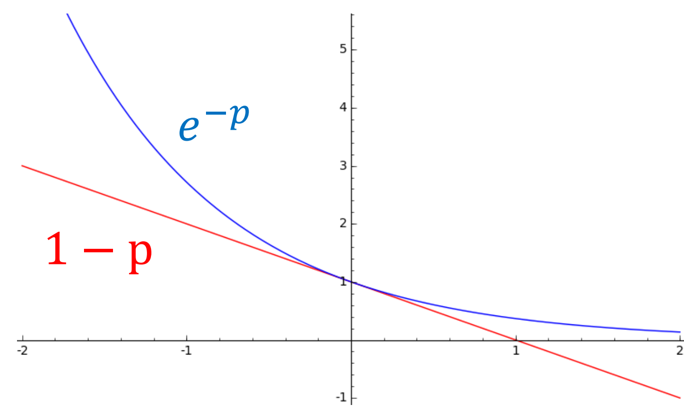
A computation

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

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



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

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 $1/\binom{n}{2}$ …

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

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

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.

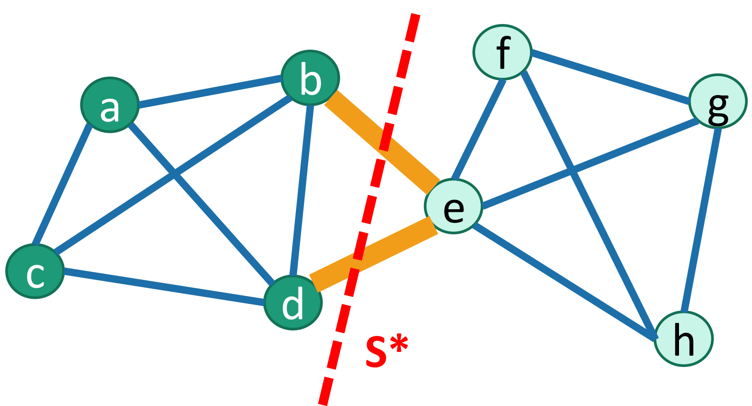
- Suppose the edges that we choose are $e_1, e_2, \ldots, e_{n-2}$
- **PR**[return S*] = **PR**[none of the e_i cross S*]

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

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

  …

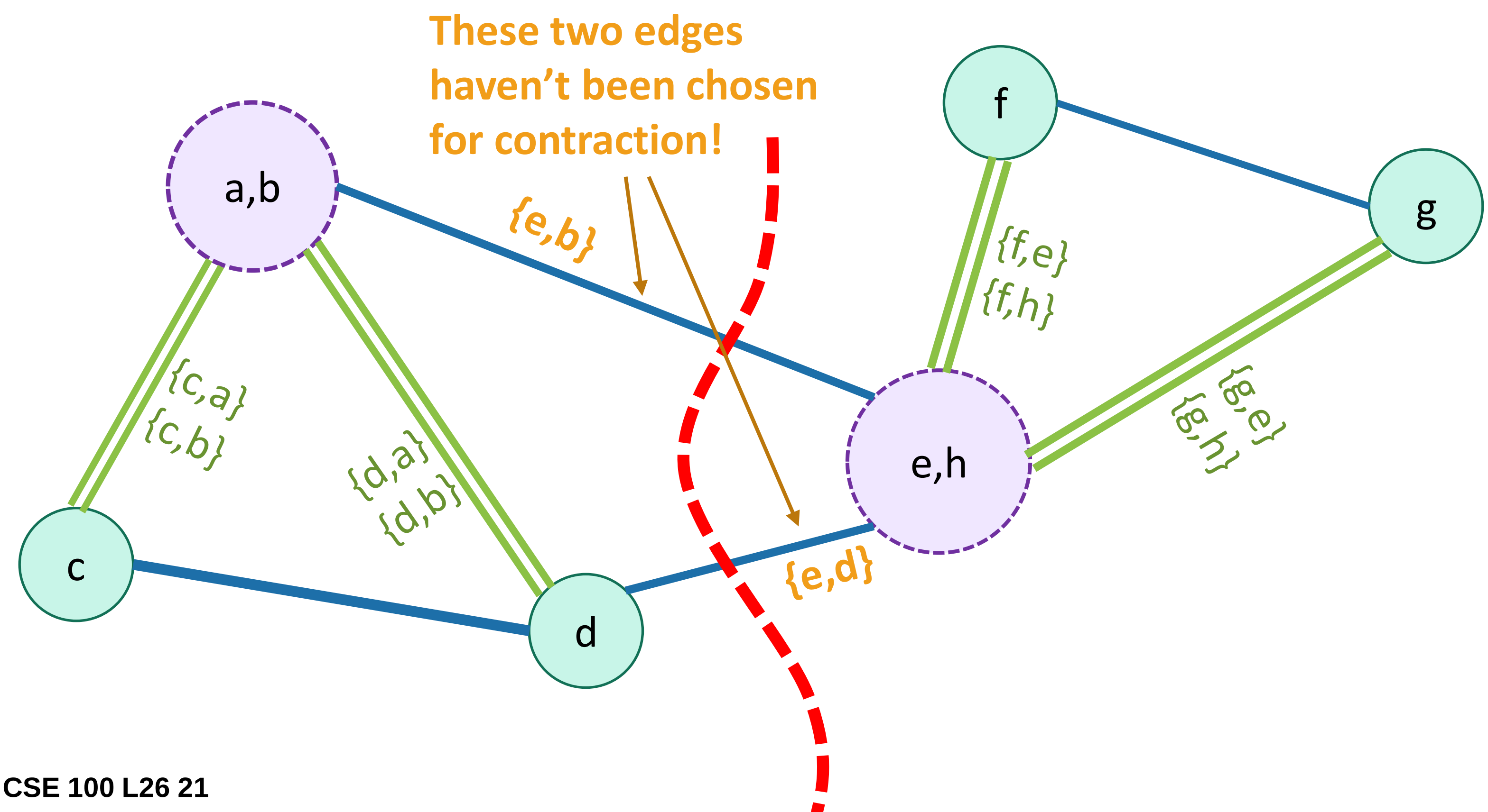
  × **PR**[e_{n-2} doesn't cross S* | $e_1,\ldots,e_{n-3}$ don't cross S*]



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:

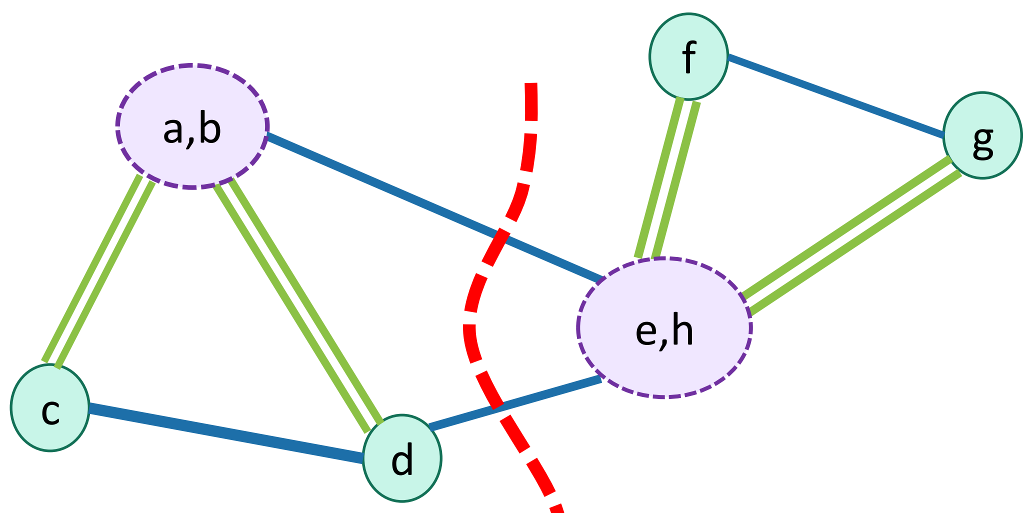
PR[e_j doesn't cross S* | $e_1,\ldots,e_{j-1}$ don't cross S*]

- 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 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.
  - b/c 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:

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



Focus in on:

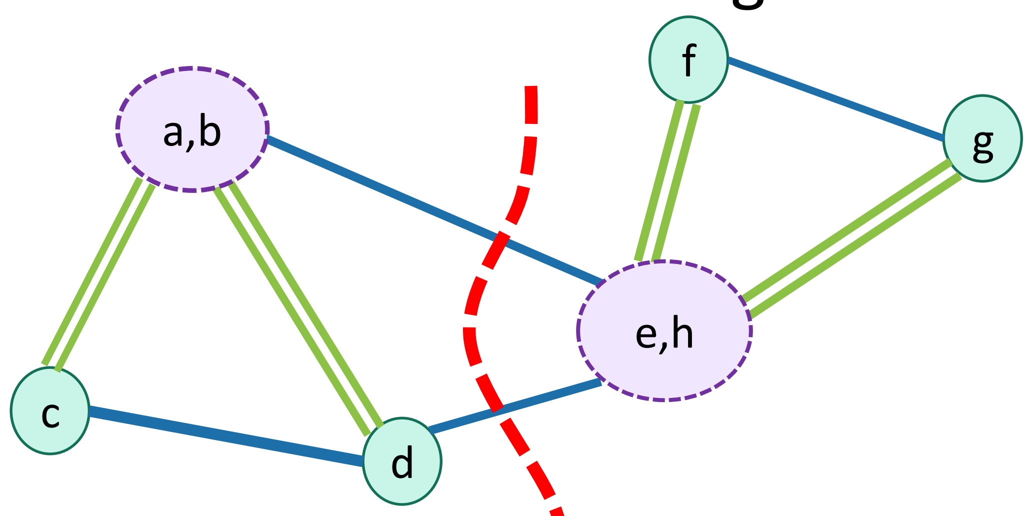
PR[e_j doesn't cross S* | $e_1,\ldots,e_{j-1}$ don't cross S*]

- 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.

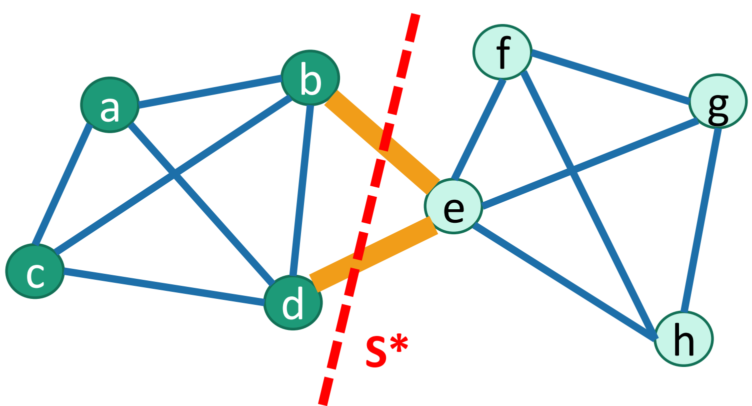
- Suppose the edges that we choose are $e_1, e_2, \ldots, e_{n-2}$
- **PR**[return S*] = **PR**[none of the e_i cross S*]

  = **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*]

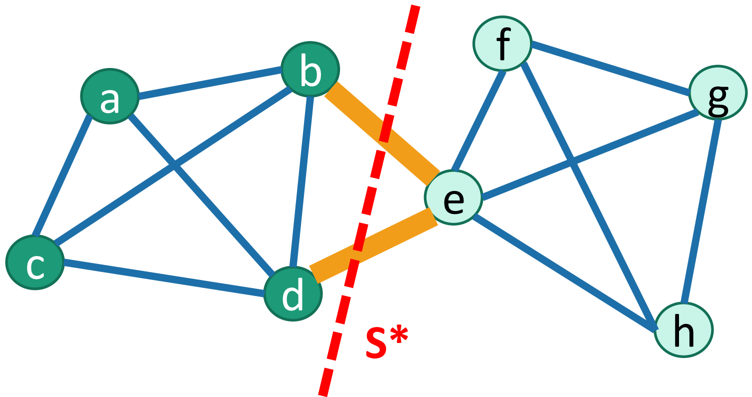


Now let's prove that claim

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.

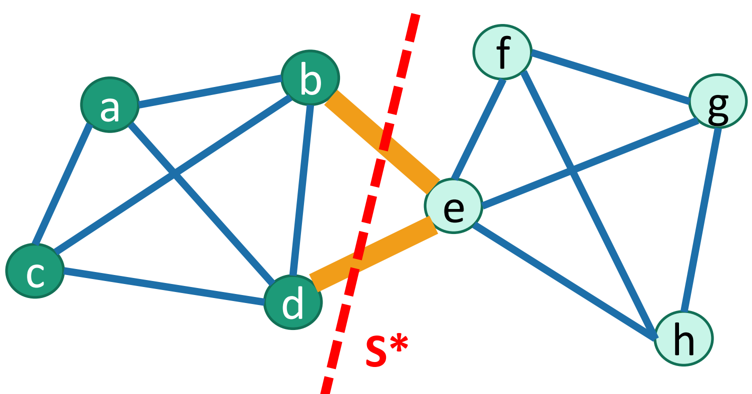
- Suppose the edges that we choose are $e_1, e_2, \ldots, e_{n-2}$
- **PR**[return S*] = **PR**[none of the e_i cross S*]

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

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

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

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!

Theorem

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

That proves this Theorem!

- 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) // independent randomness
    - **if** candidateCut is smaller than bestCut:
      - bestCut ← candidateCut
  - **return** bestCut
- Then $\Pr[\text{ this doesn't return a min cut }] \leq \delta.$

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 δ = 1/n, which makes the running time $O(n^4 \log(n))$. It depends on how sure you want to be!

More generally

- If we have a Monte-Carlo algorithm 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.

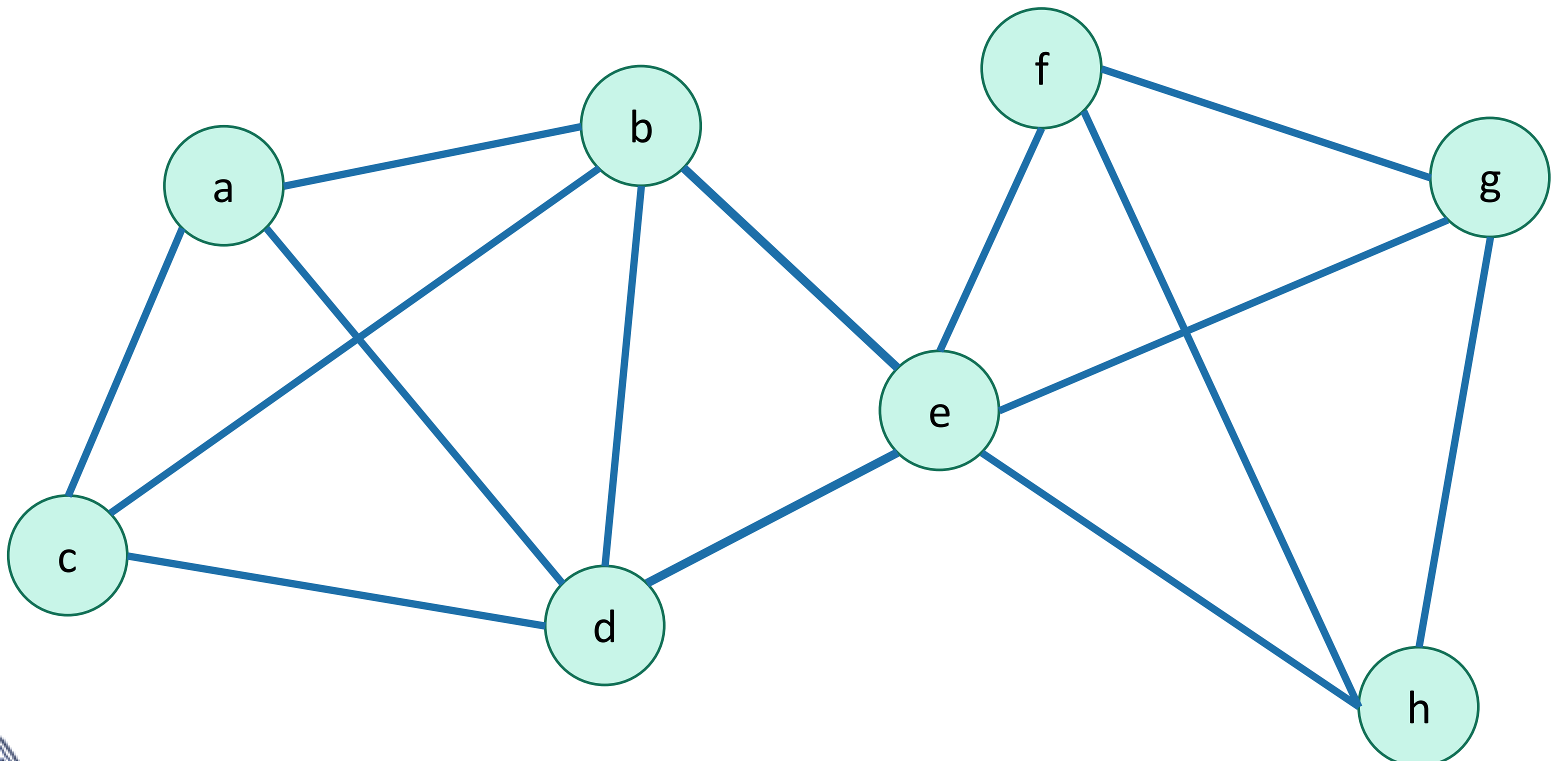
Can we do better?

- 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

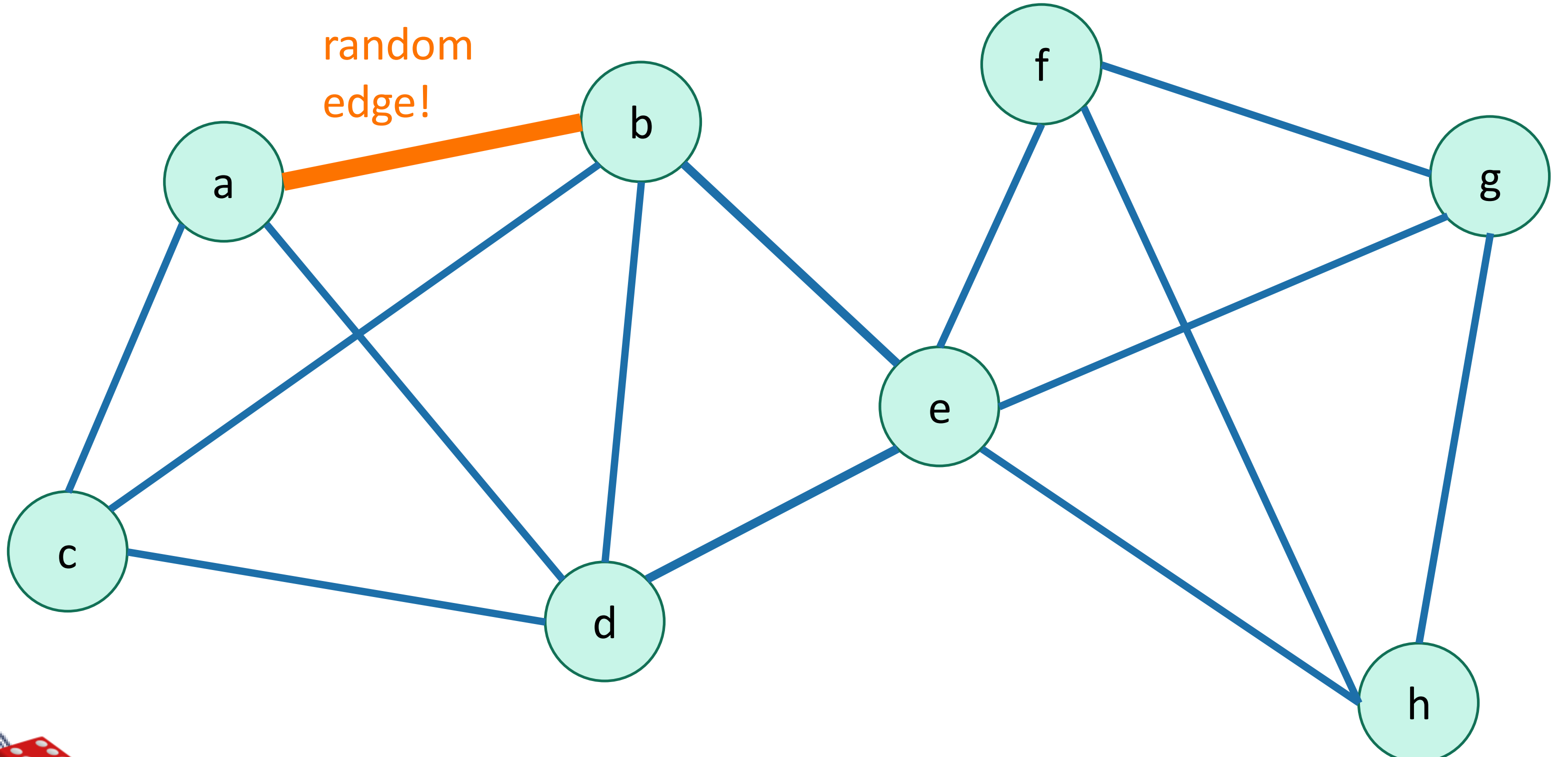


Karger's algorithm

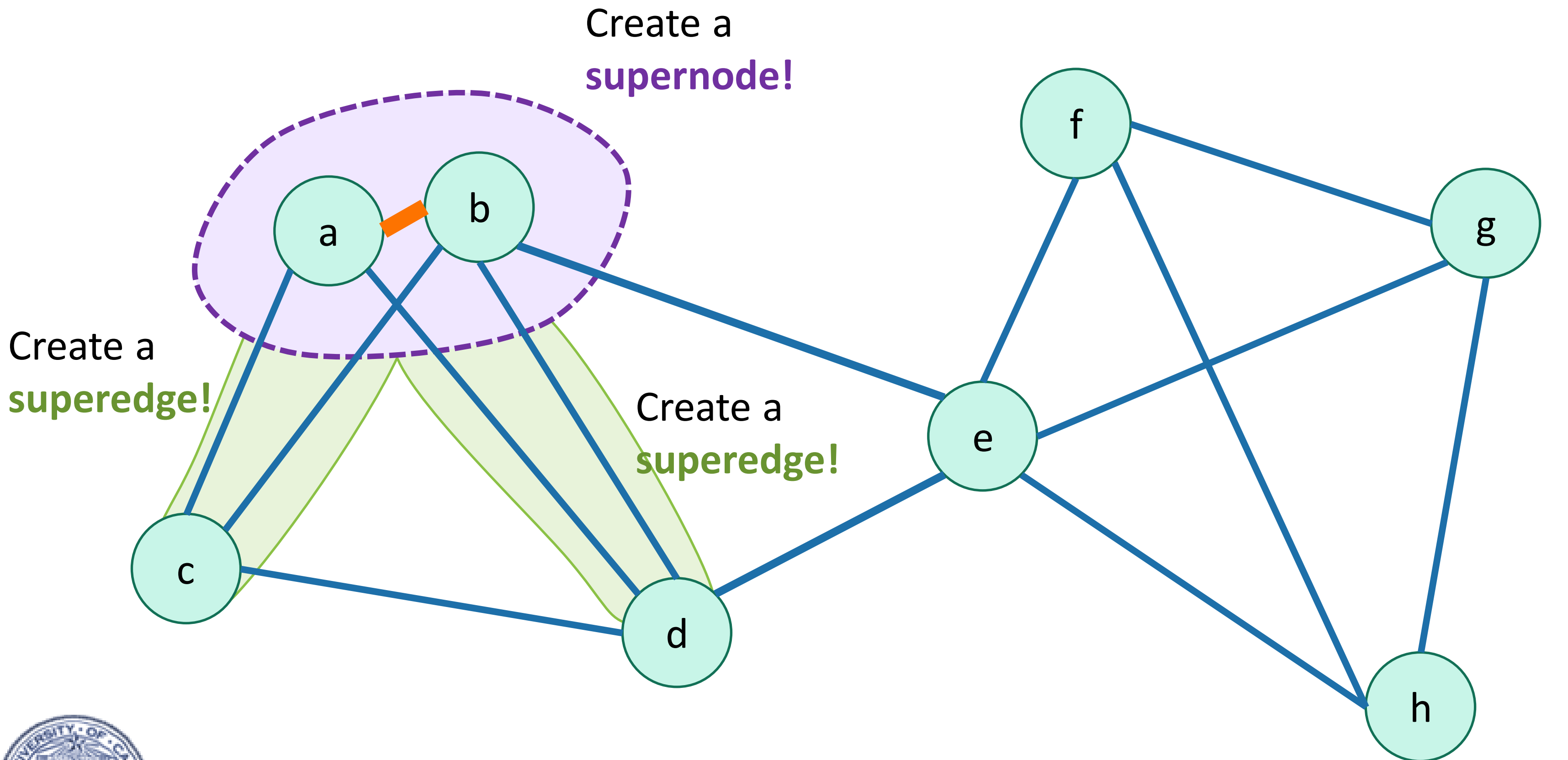
Probability that we didn't mess up:

12/14

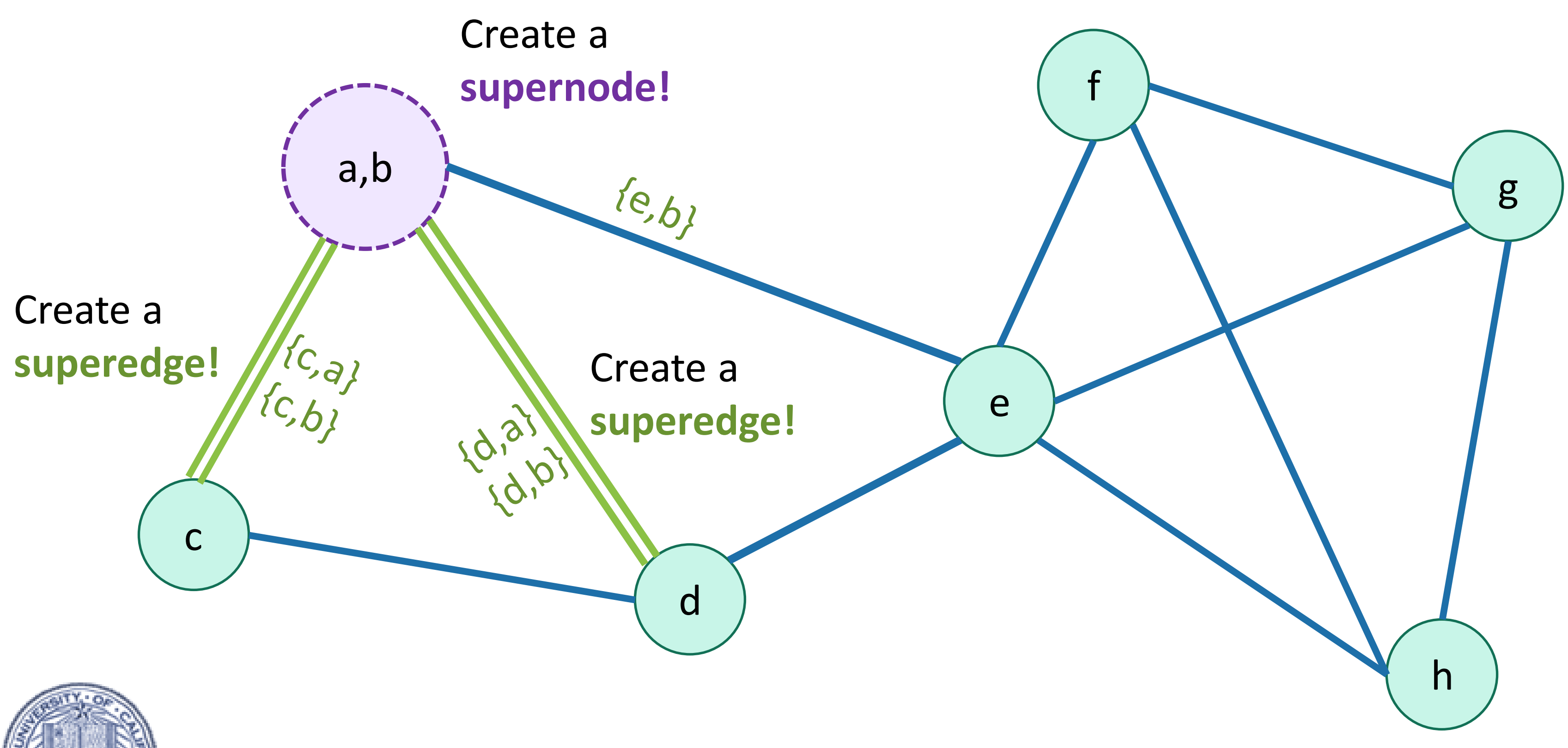
There are 14 edges, 12 of which are good to contract.



Karger's algorithm



Karger's algorithm

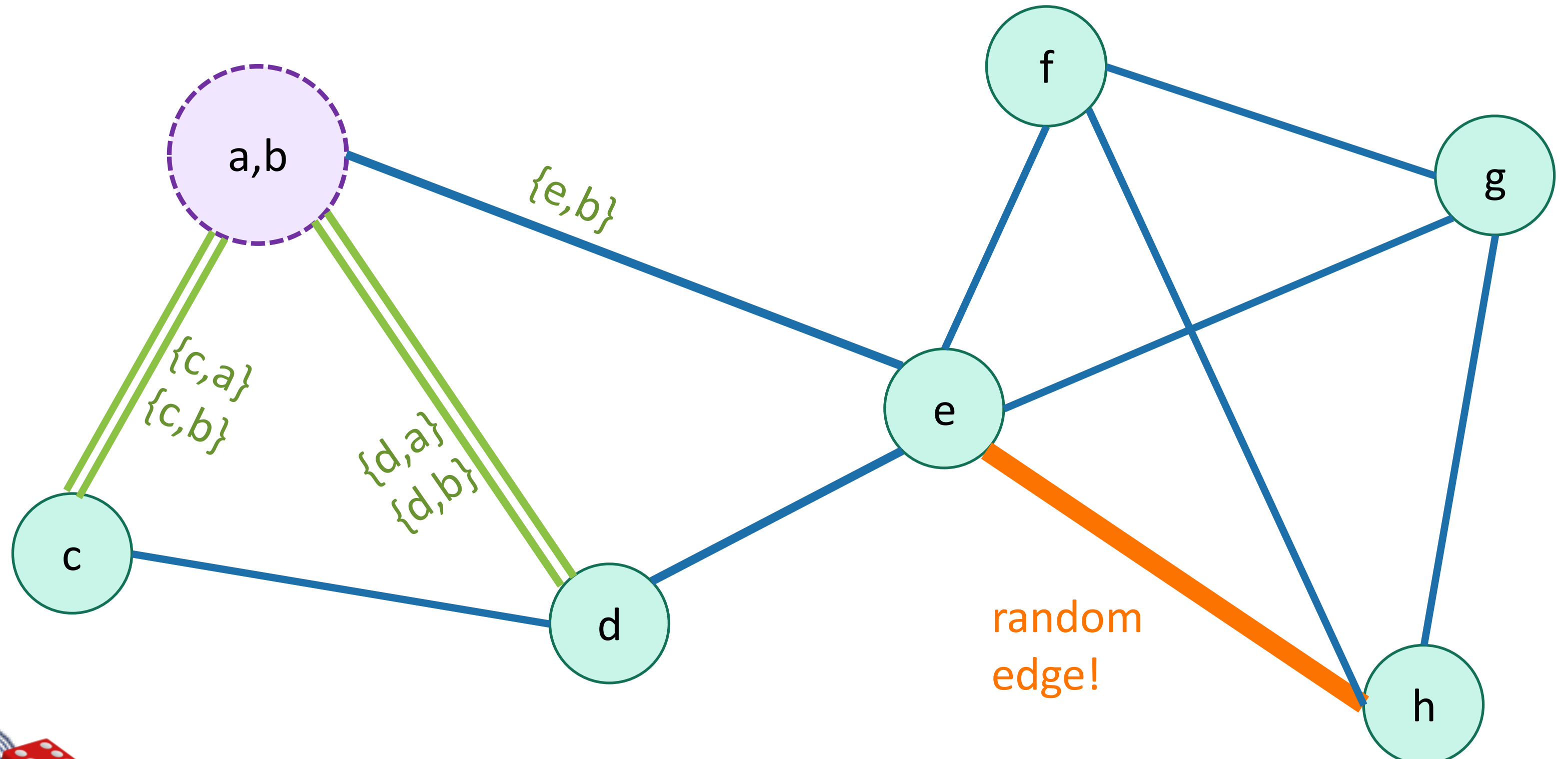


Karger's algorithm

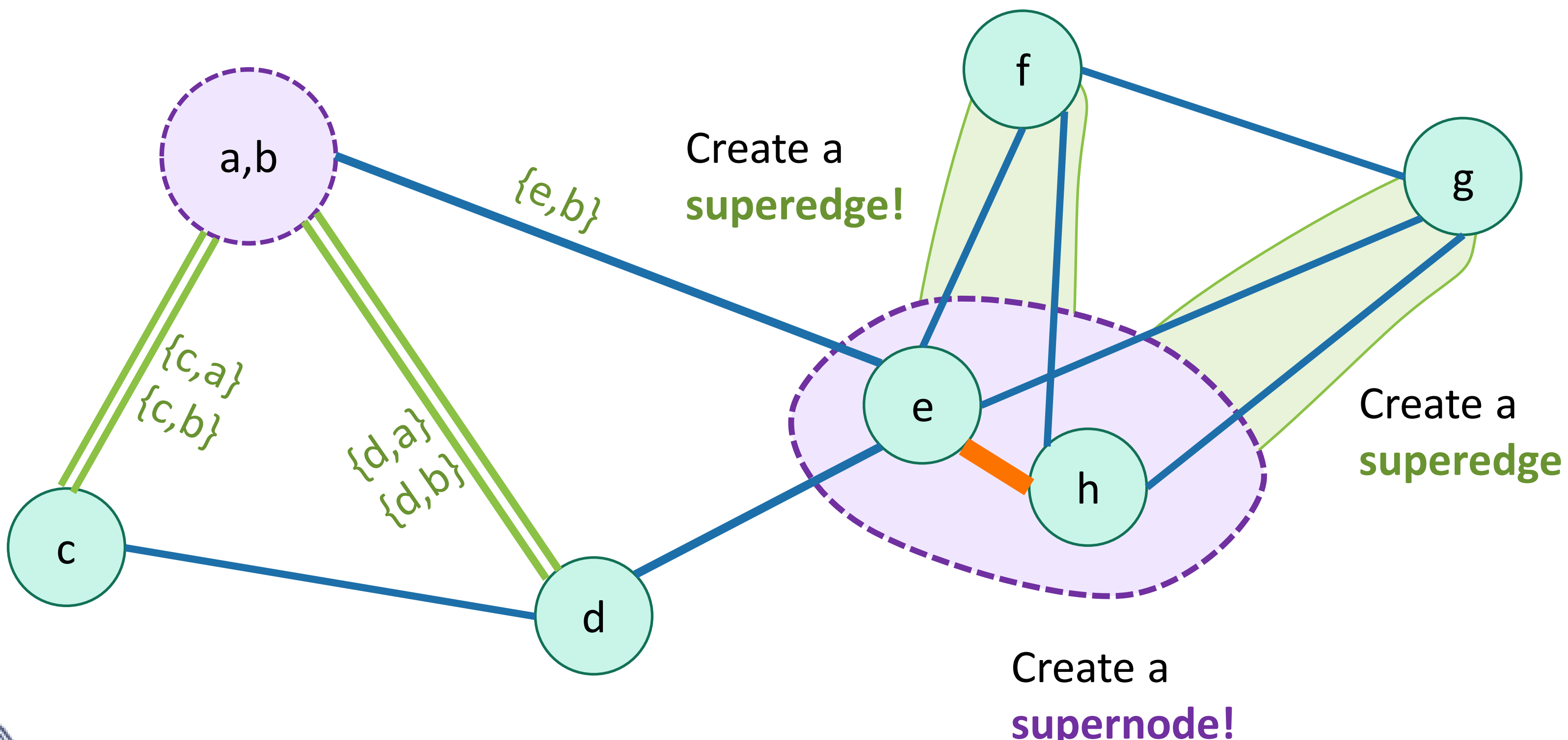
Probability that we didn't mess up:

11/13

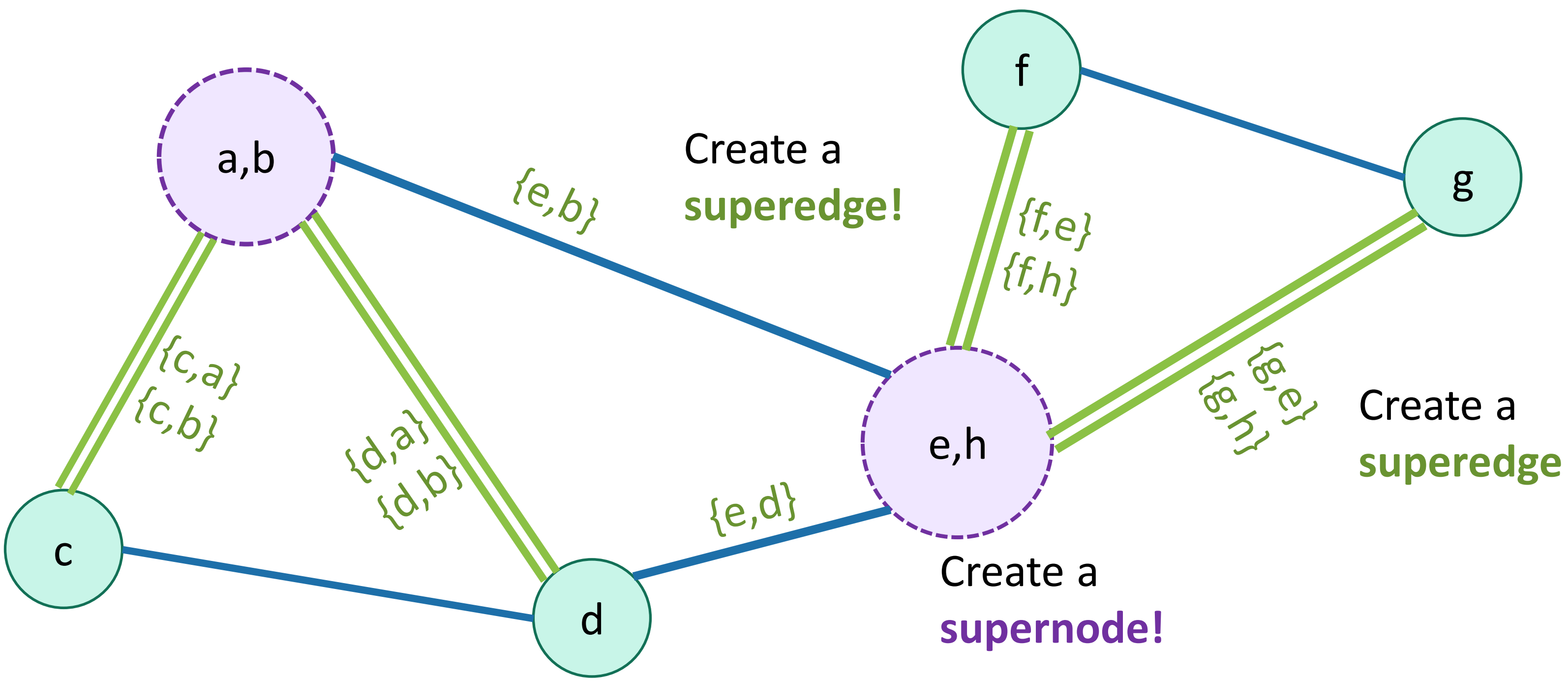
Now there are only 13 edges, since the edge between a and b disappeared.



Karger's algorithm



Karger's algorithm

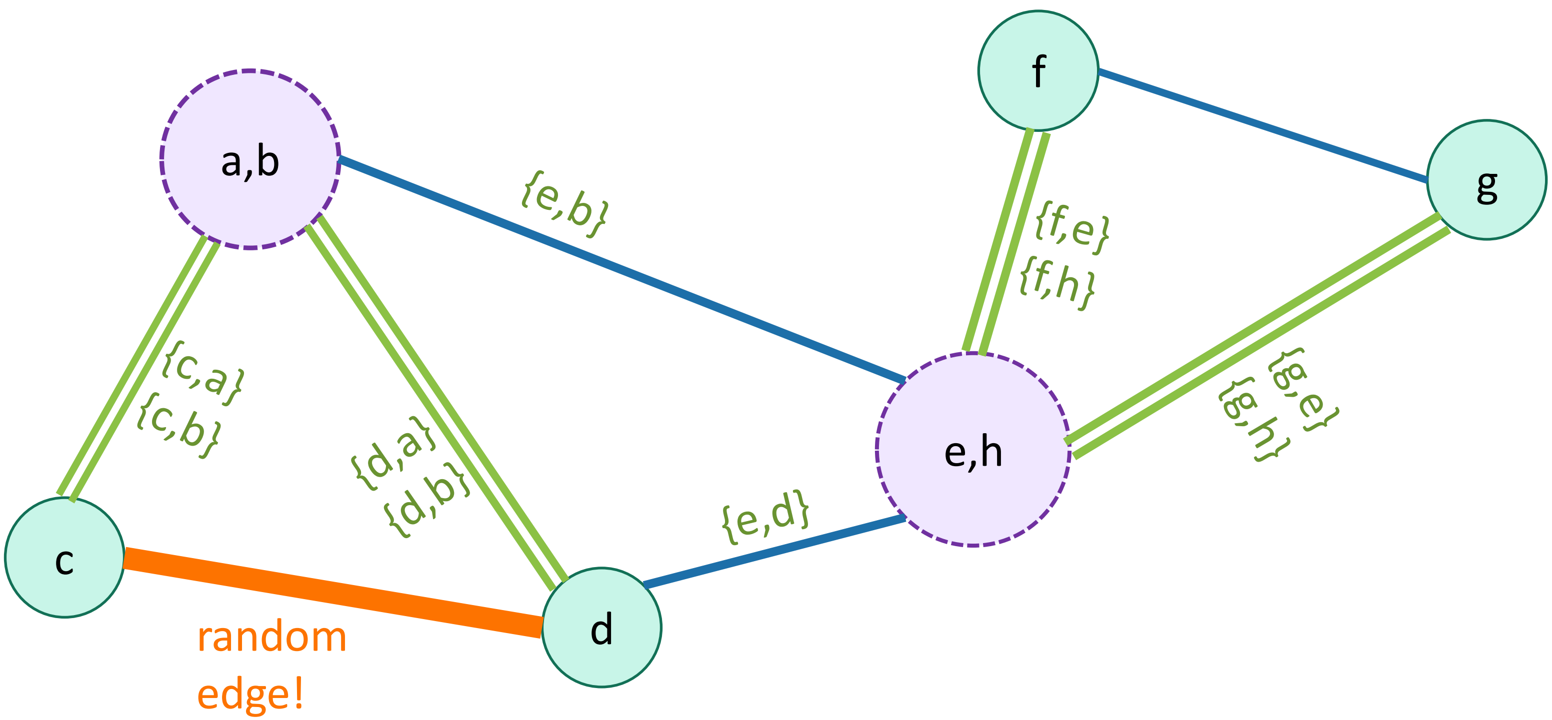


Karger's algorithm

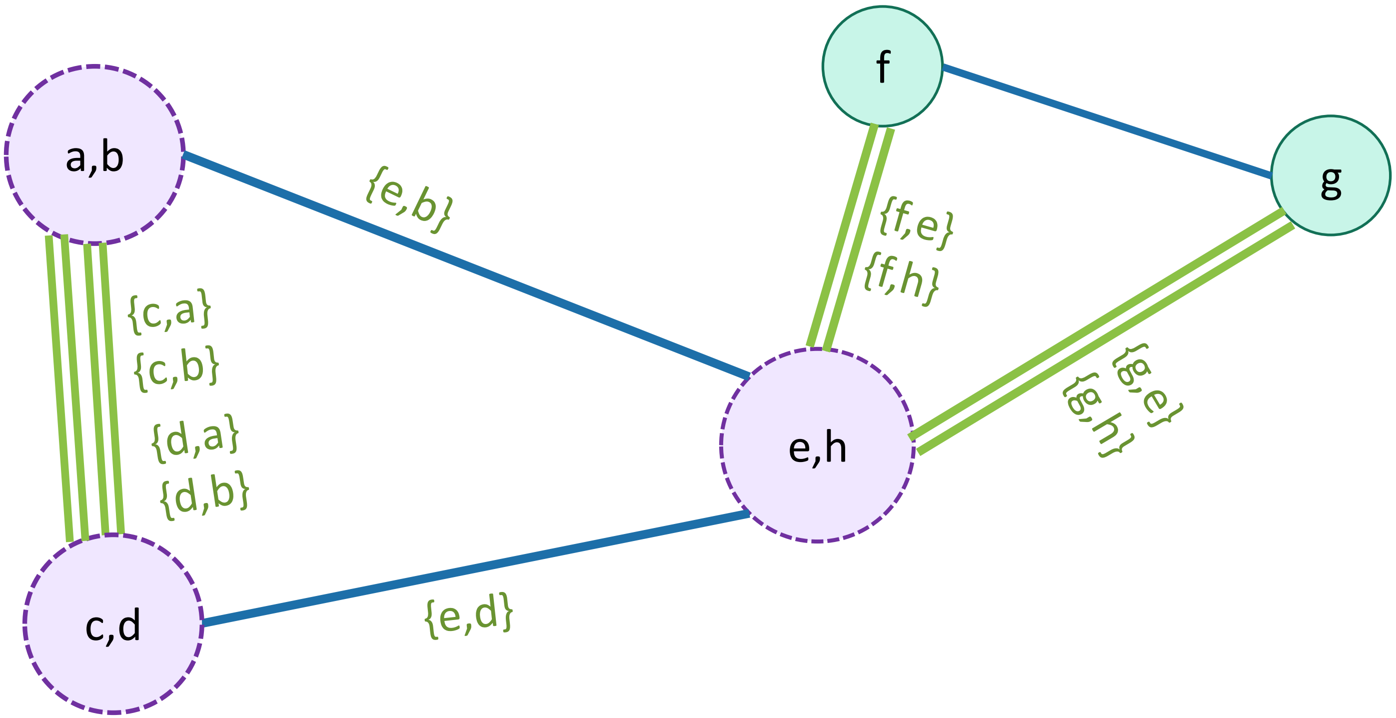
Probability that we didn't mess up:

10/12

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



Karger's algorithm

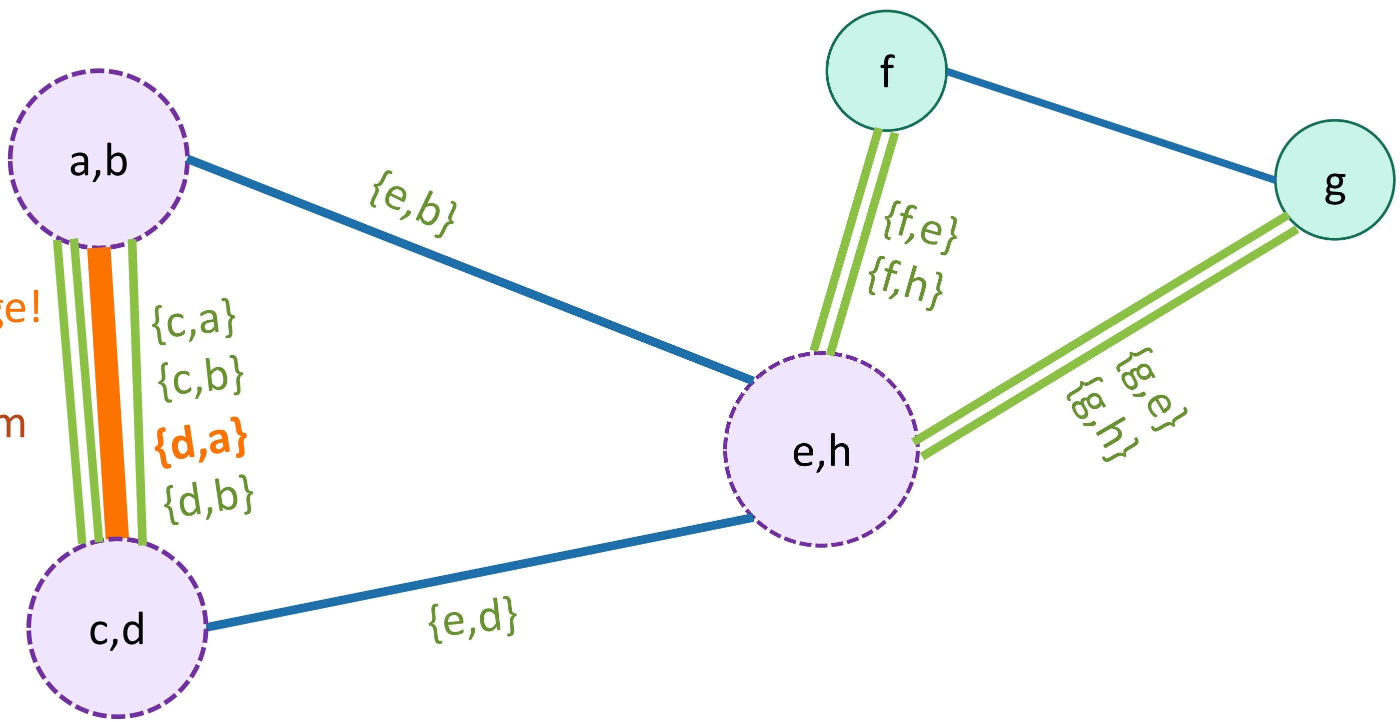


Karger's algorithm

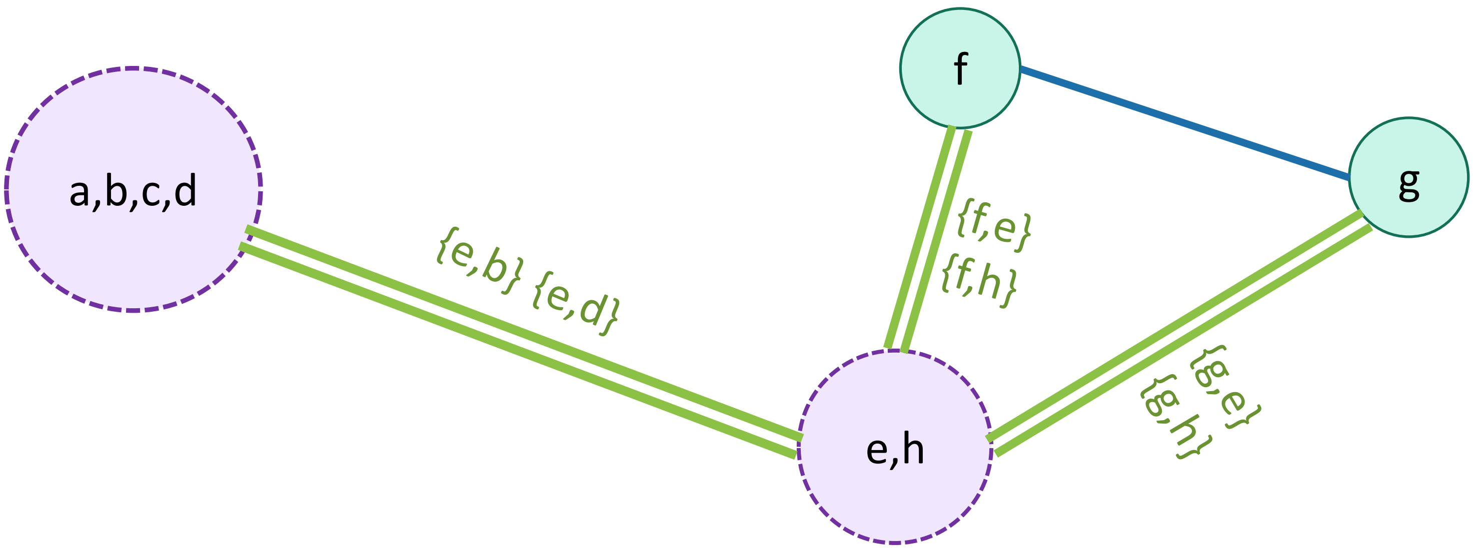
Probability that we didn't mess up:

9/11

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



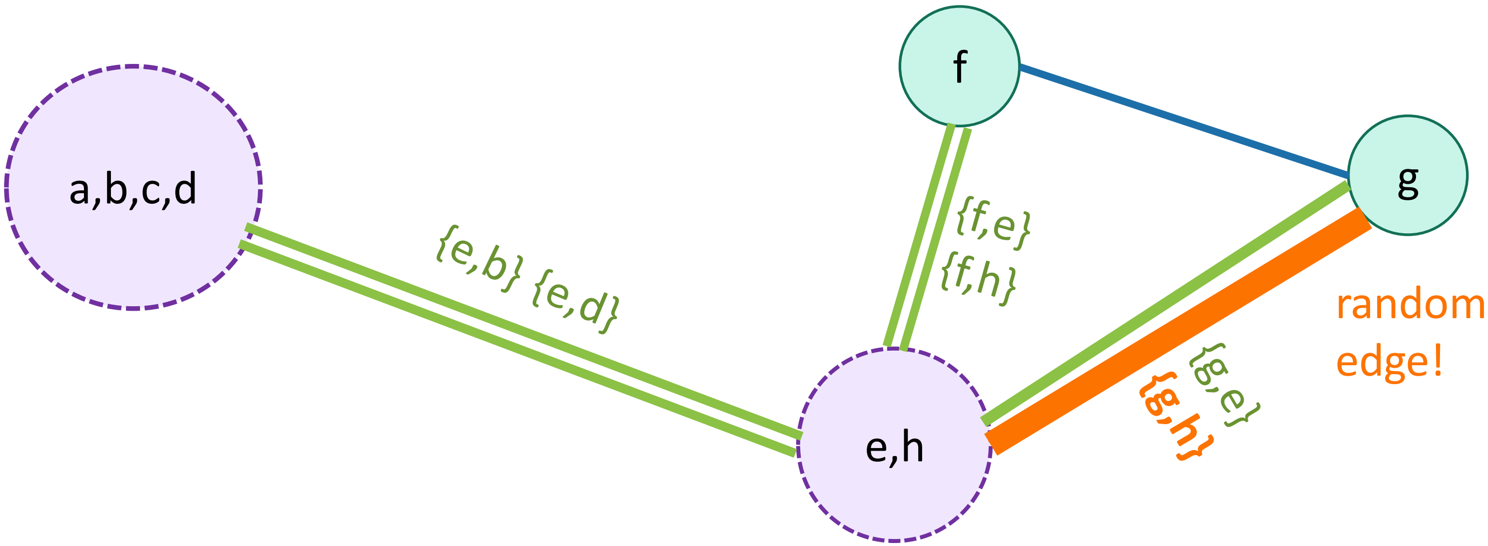
Karger's algorithm



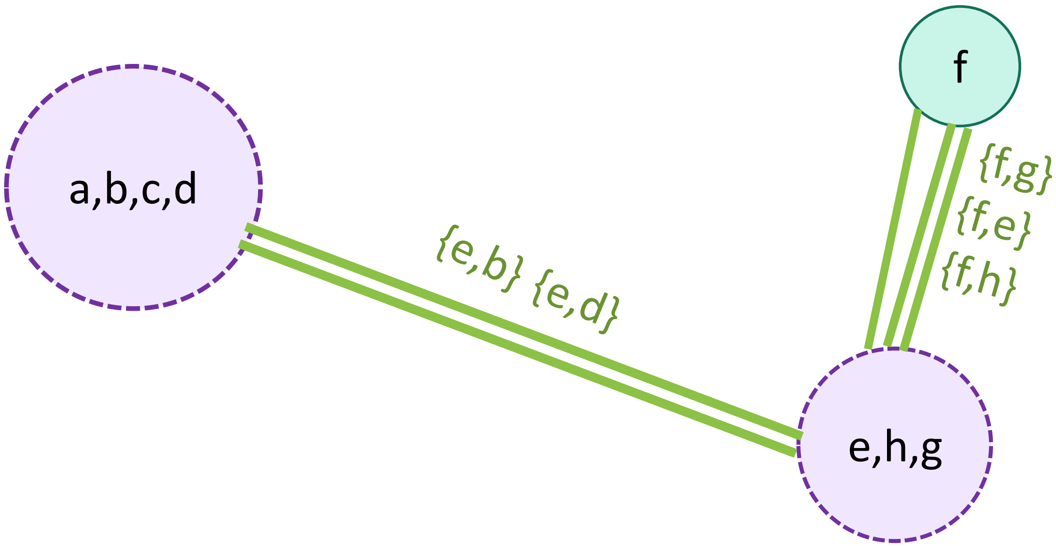
Karger's algorithm

Probability that we didn't mess up:

5/7



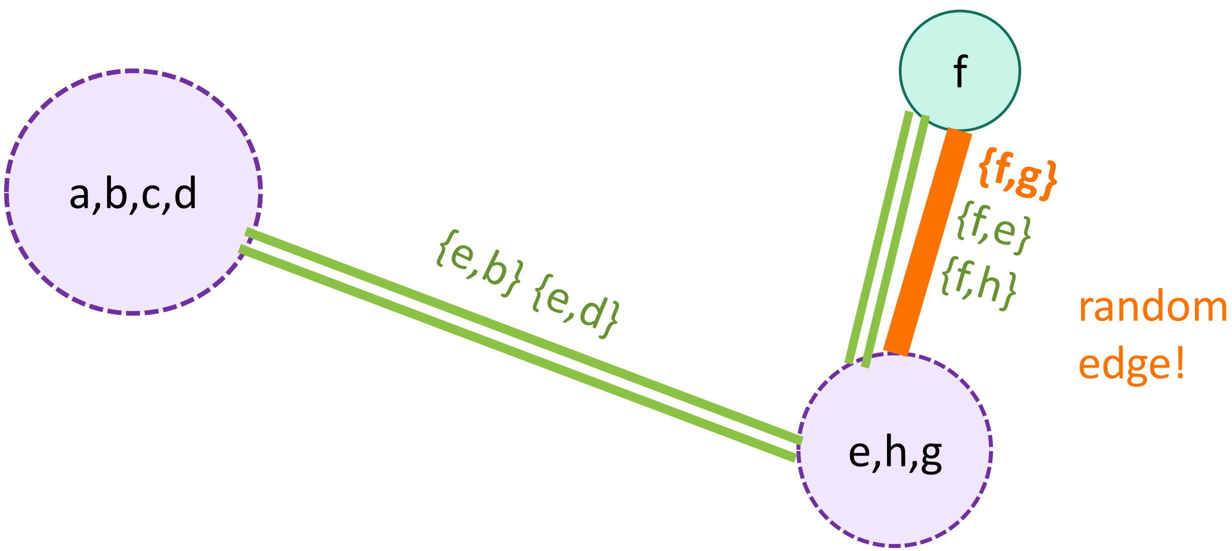
Karger's algorithm



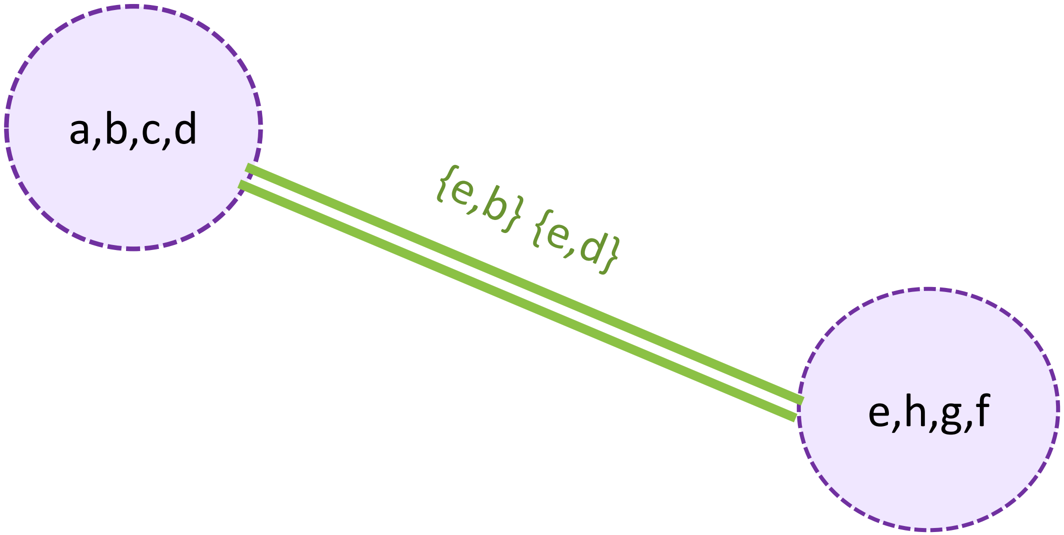
Karger's algorithm

Probability that we didn't mess up:

3/5



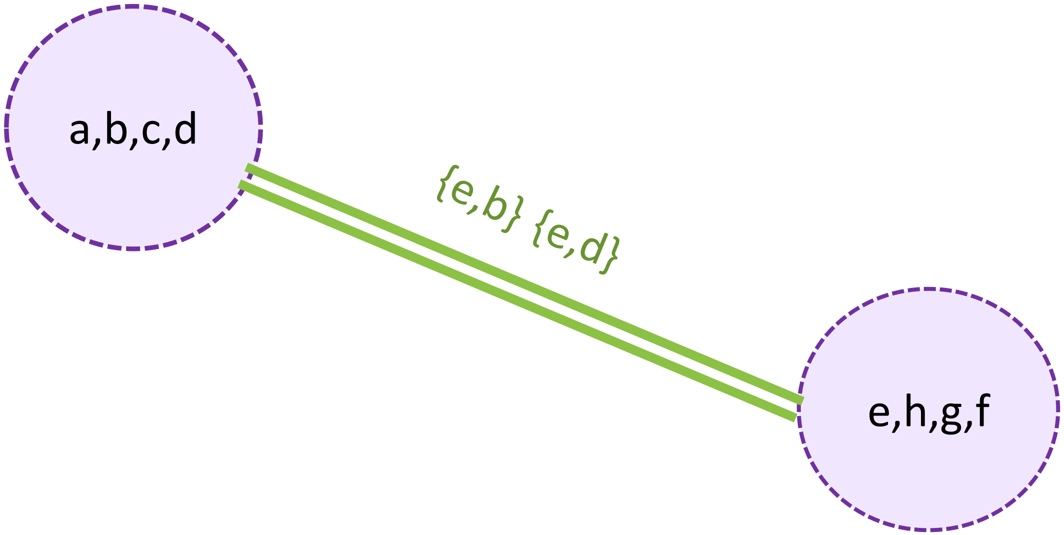
Karger's algorithm



Karger's algorithm

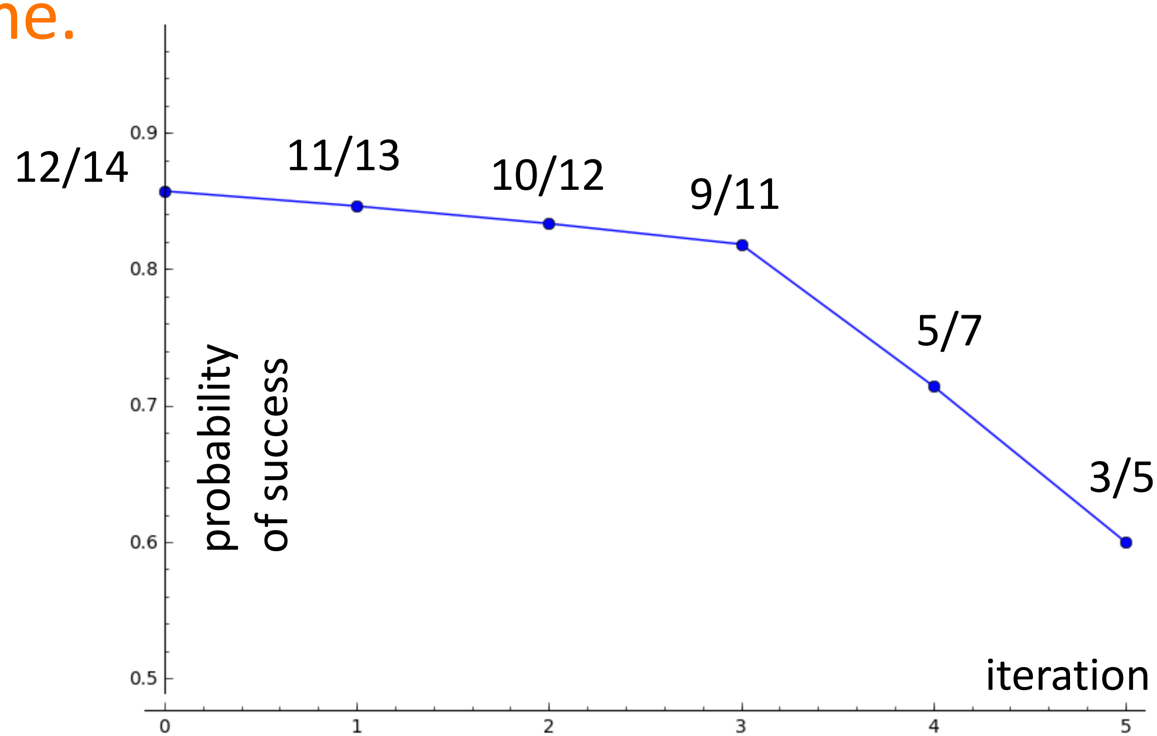
Now stop!

- There are only two nodes left.



Probability of not messing up

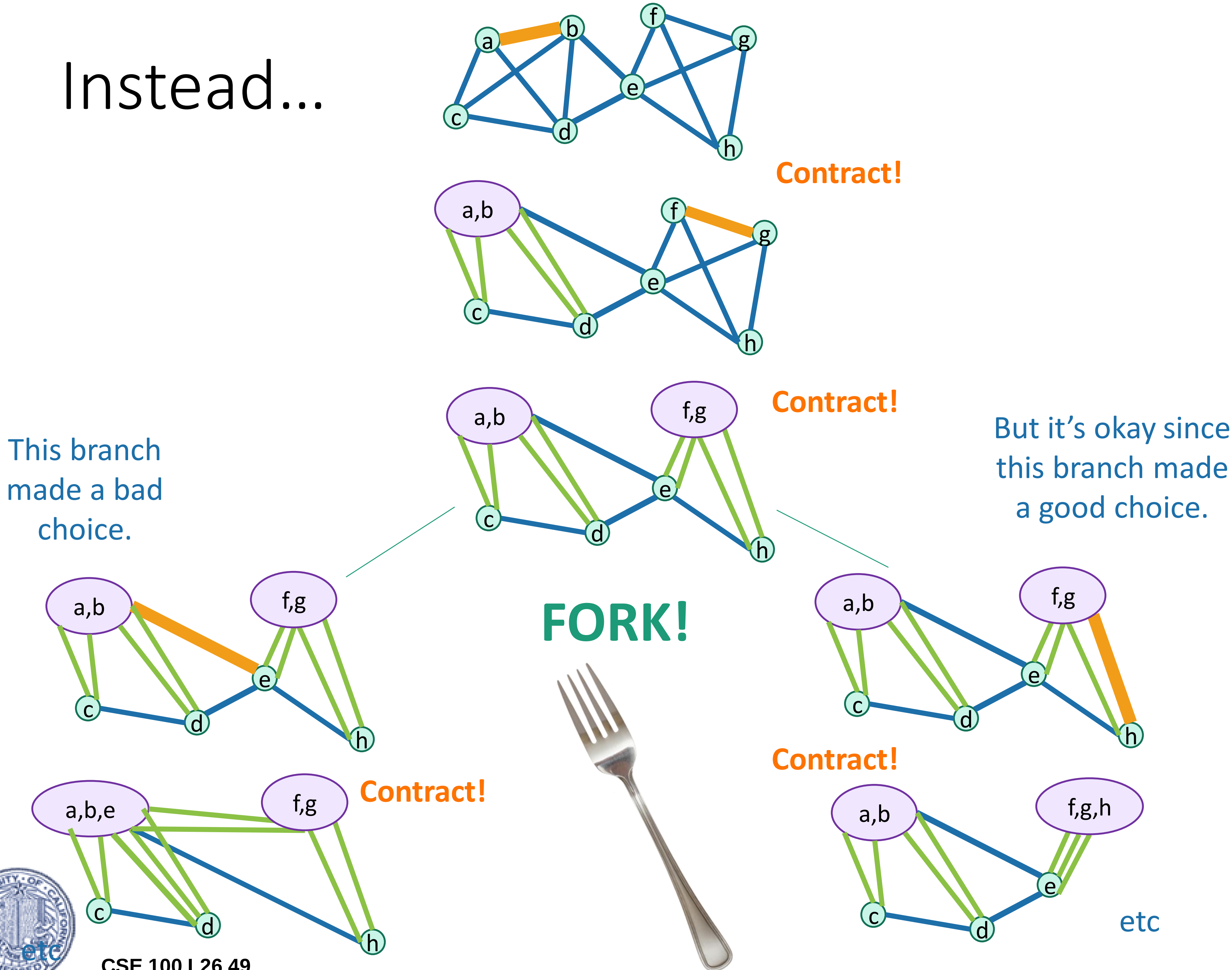
- At the beginning, it's pretty likely we'll be fine.
- The probability that we mess up gets worse and worse over time.



Moral:

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

Instead…



a
b
c
d
e
f
g
h
Contract!
a,b
f,g
Contract!
This branch made a bad choice.
But it's okay since this branch made a good choice.
FORK!
a,b,e
f,g,h
Contract!
Contract!
etc
etc

In words

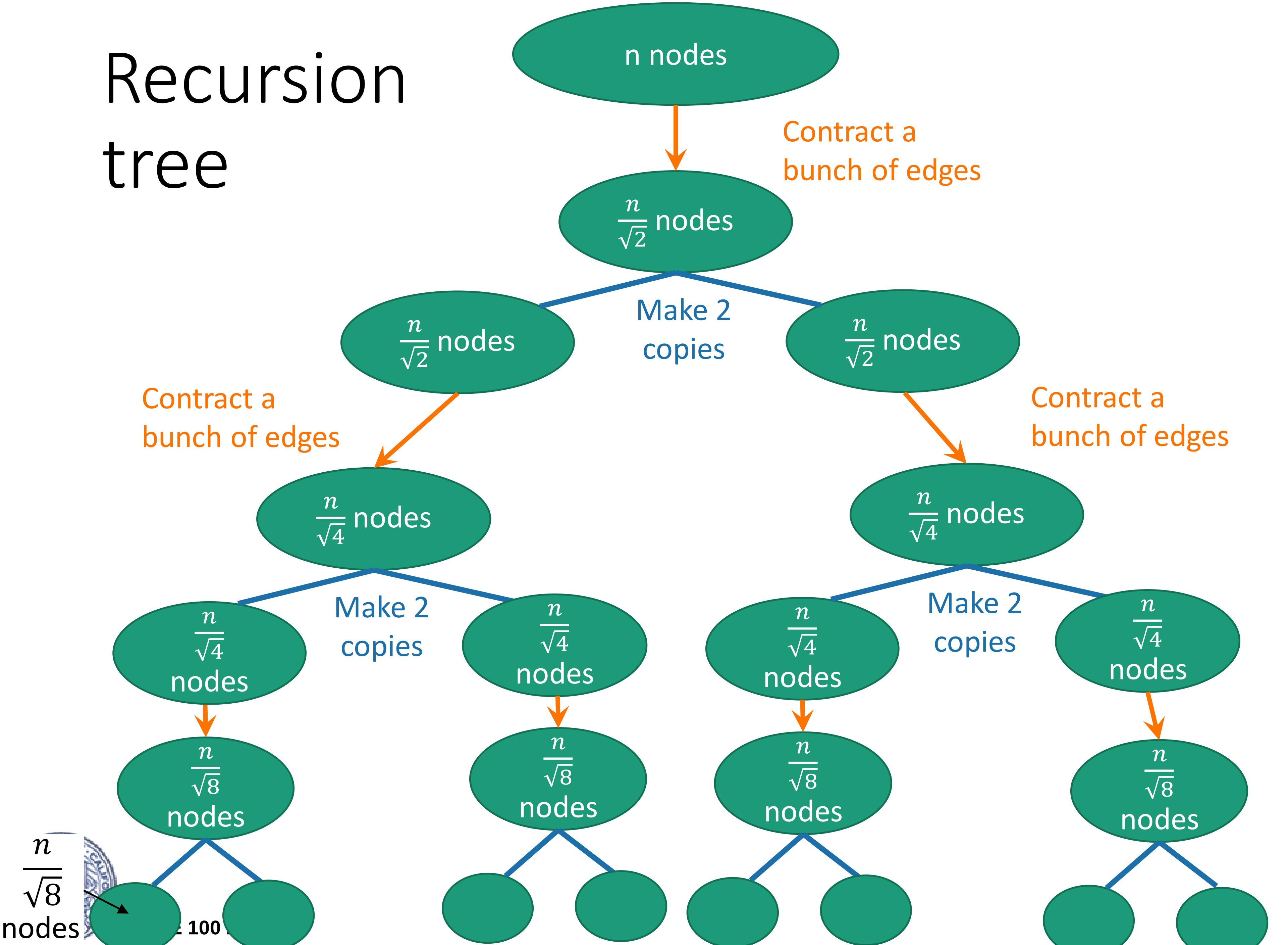
Why $\frac{n}{\sqrt{2}}$? We'll see later.

- 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…

In pseudocode

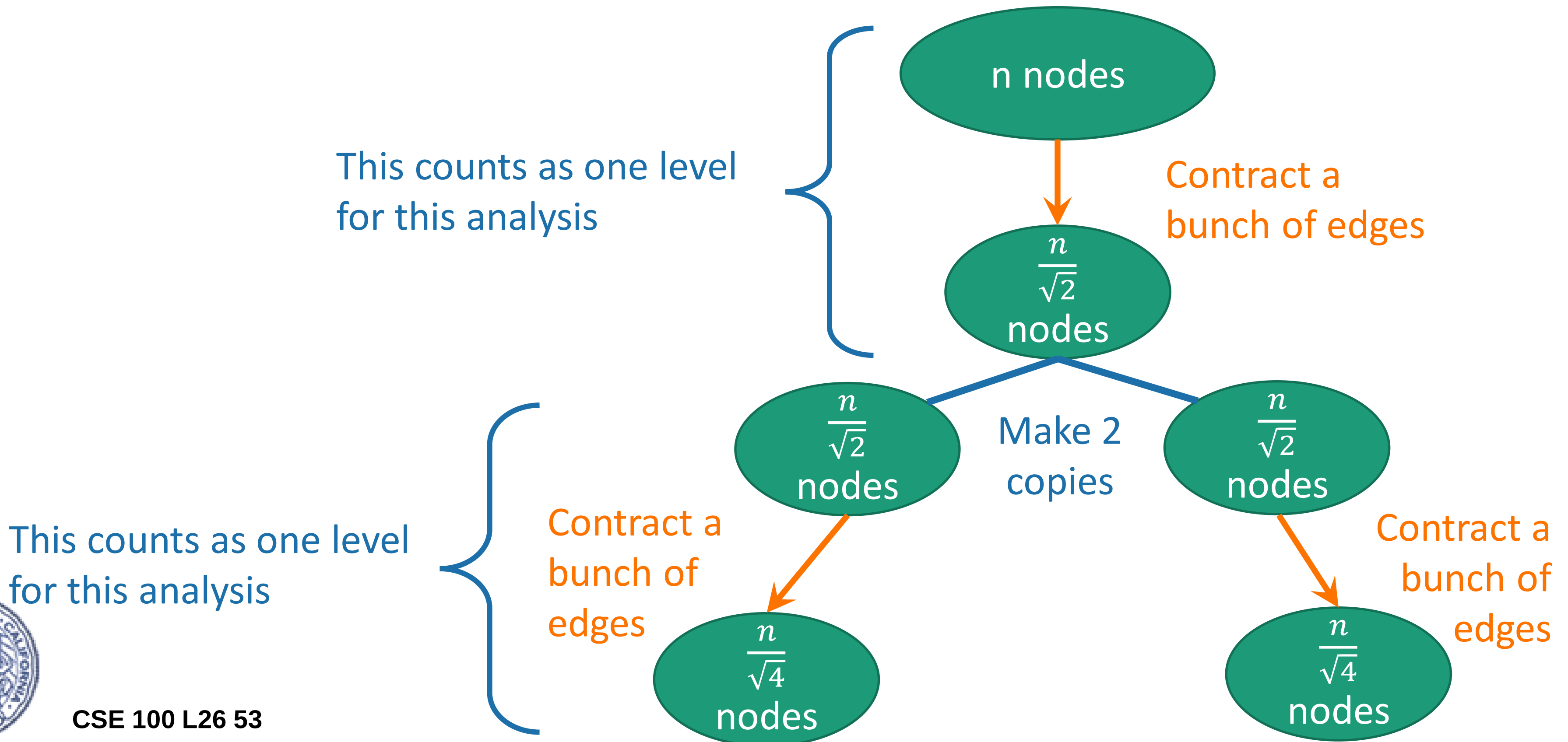
- **KargerStein**(G = (V,E)):
  - n ← |V|
  - if n < 4:
    - find a min-cut by brute force \\ time O(1)
  - Run Karger's algorithm on G with independent repetitions until $\left\lfloor \frac{n}{\sqrt{2}} \right\rfloor$ nodes remain.
  - G_1, G_2 ← copies of what's left of G
  - S_1 = **KargerStein**(G_1)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

Recursion tree



Recursion tree

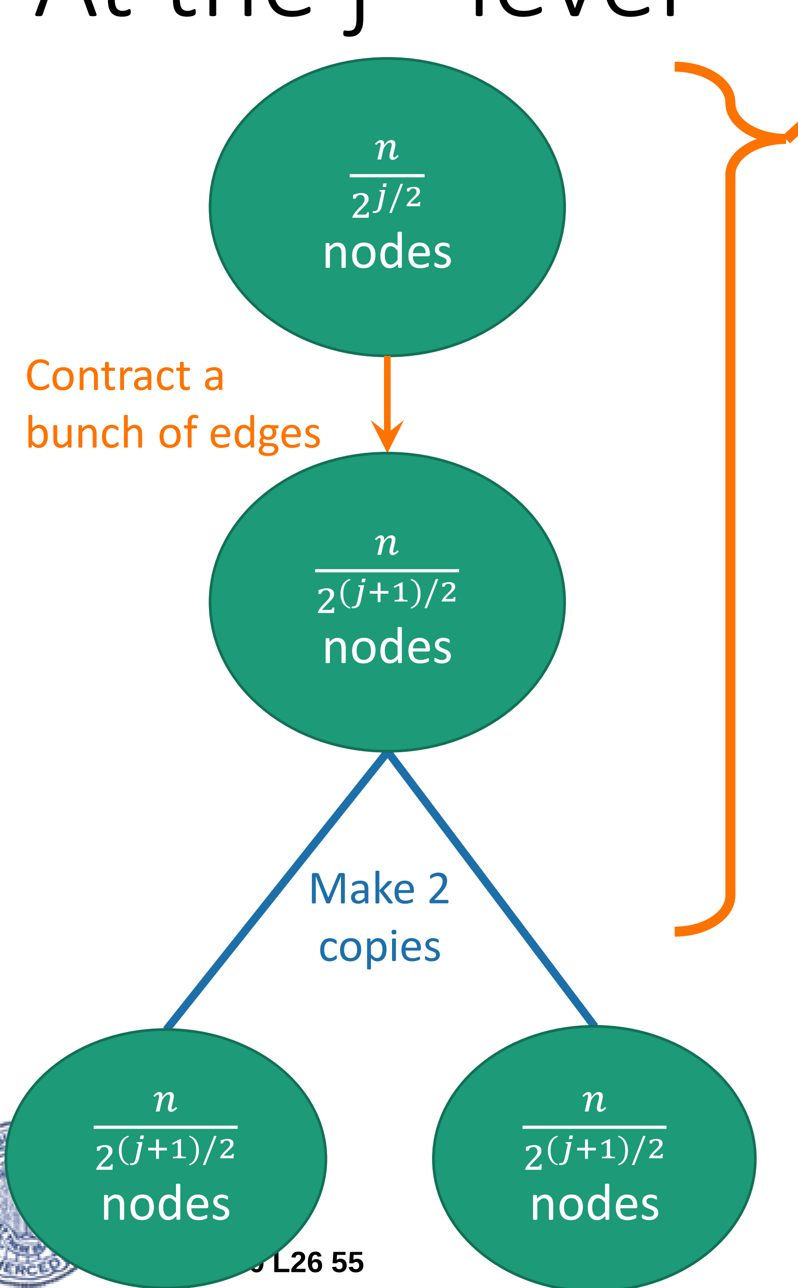
- depth is $\log_{\sqrt{2}}(n) = \frac{\log(n)}{\log(\sqrt{2})} = 2\log(n)$
- number of leaves is $2^{2\log(n)} = n^2$



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, $T(n) = O(n^2 \log(n))$

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 $e_1, e_2, \ldots, e_{n-t}$ 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-t} doesn't cross S* | $e_1, \ldots, e_{n-t-1}$ 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

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

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

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

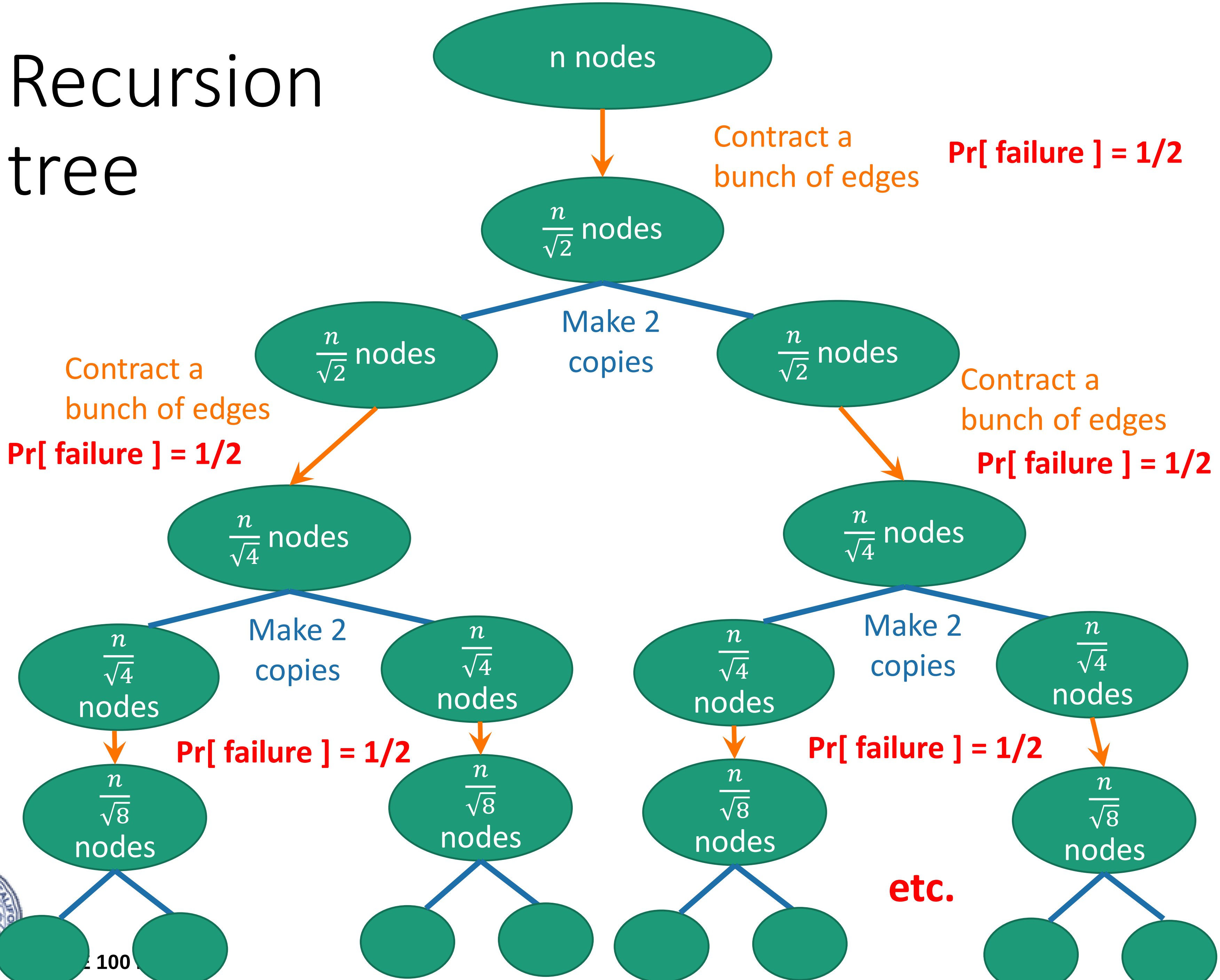
$$= \frac{t\cdot(t-1)}{n\cdot(n-1)}$$

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

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

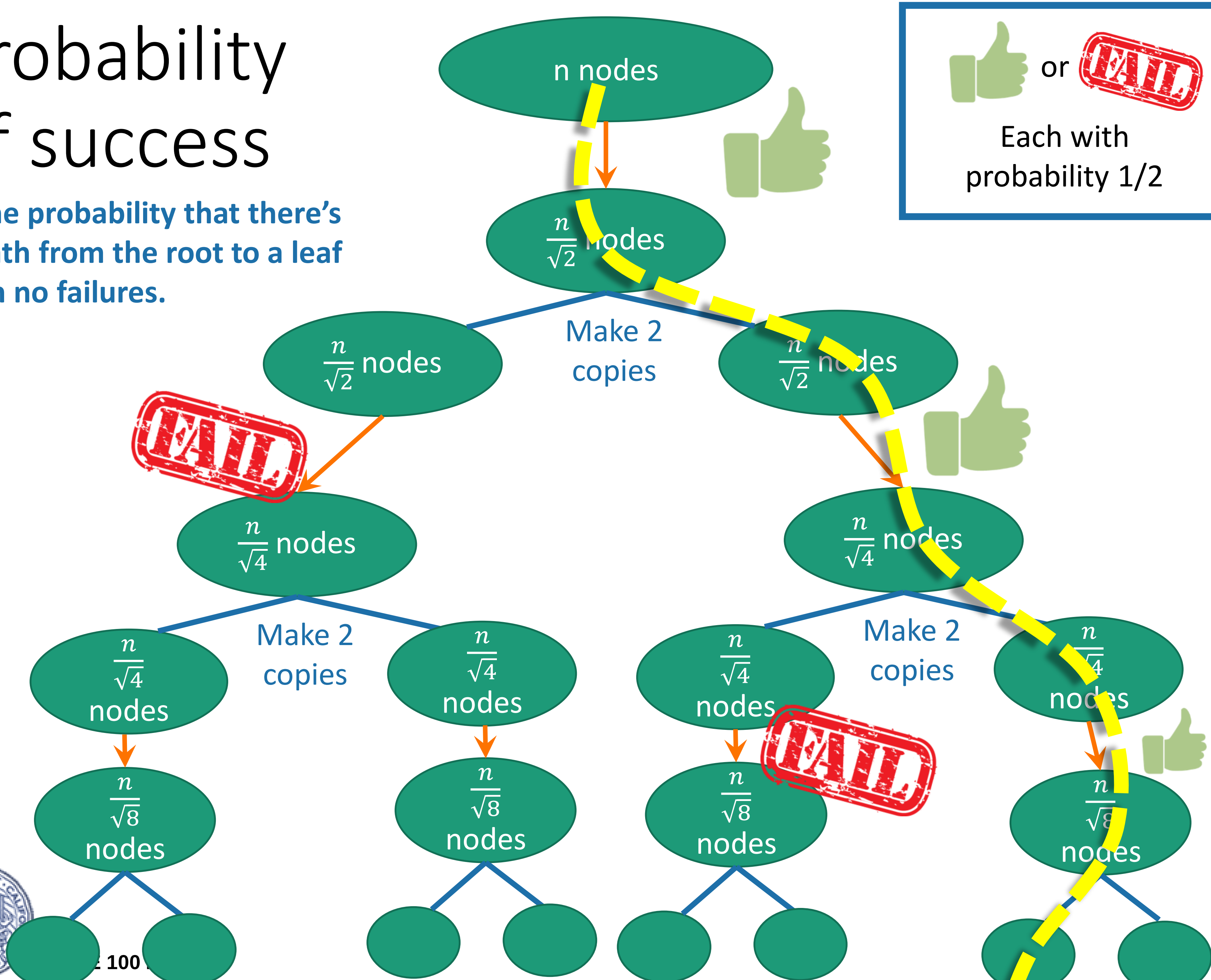
when n is large

Recursion tree



Probability of success

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



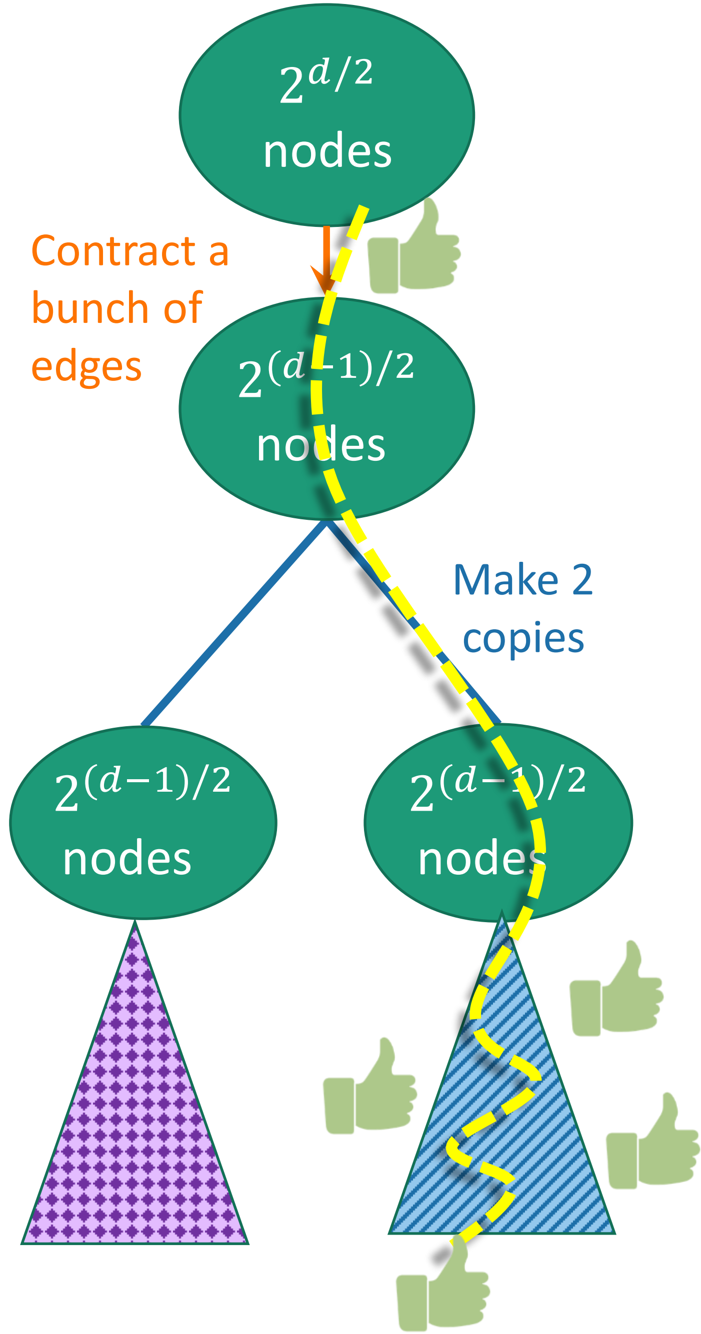
The problem we need to analyze

- 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**.

- $p_d = \frac{1}{2} \cdot \Pr\left[\begin{array}{l} \text{at least one subtree} \\ \text{has a successful path} \end{array} \right]$
- $= \frac{1}{2} \cdot \left(\Pr\left[\quad \text{wins} \right] + \Pr\left[\quad \text{wins} \right] - \Pr\left[\quad \text{both win} \right] \right)$
- $= \frac{1}{2} \cdot \left(p_{d-1} + p_{d-1} - p_{d-1}^2 \right)$
- $= p_{d-1} - \frac{1}{2} \cdot p_{d-1}^2$



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 next 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 $O\left(n^2 \log(n)\right)$
  - That's what we proved before.
- Choosing $\delta = 0.01$ as before, the total runtime is

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

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 (probably) find a global minimum cut in an undirected graph in time $\mathbf{O(n^2 \log^2(n))}$.
  - 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!