https://algs4.cs.princeton.edu

Randomness

- randomness and algorithms
- treasure hunt problem
- duplicates finding
- Karger's algorithm
- more applications

https://algs4.cs.princeton.edu

Randomness

- *randomness and algorithms*
- *treasure hunt problem*
- *duplicates finding*
- *Karger's algorithm*
- *more applications*

Randomness: quiz 1

Which of these outcomes is most likely to occur in a sequence of 6 coin flips?

A. 

B. 

C. 

D. All of the above.

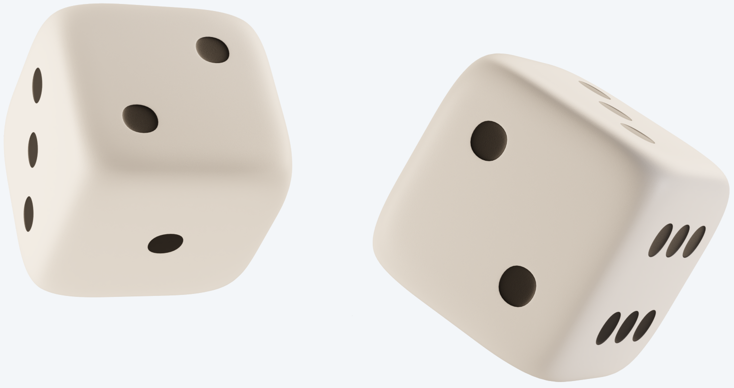
E. Both B and C.

The uniform distribution

Coin flip.

$$\mathbb{P}[C \text{ lands heads}] = \mathbb{P}[C \text{ lands tails}] = \frac{1}{2}.$$

Roll of a die.



$$\mathbb{P}[D \text{ rolls } 1] = \mathbb{P}[D \text{ rolls } 2] = \cdots = \mathbb{P}[D \text{ rolls } 6] = \frac{1}{6}.$$

Terminology and notation.

"C lands heads" and "D is even" are **events** with probabilities $\mathbb{P}[C \text{ lands heads}]$, $\mathbb{P}[D \text{ rolls even}]$.

Distribution: all outcome-probability pairs.

outcome	probability
heads	1/2
tails	1/2

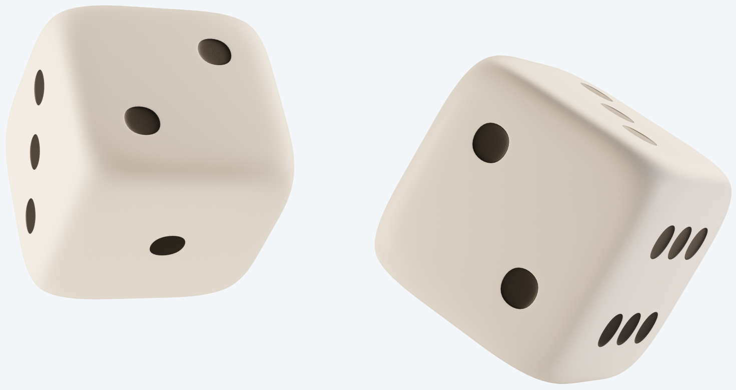
distribution of unbiased coin

The uniform distribution

Coin flip.

$\mathbb{P}[C \text{ lands heads}] = \mathbb{P}[C \text{ lands tails}] = \frac{1}{2}$. ← *uniform over 2 outcomes*

Roll of a die.



$\mathbb{P}[D \text{ rolls } 1] = \mathbb{P}[D \text{ rolls } 2] = \cdots = \mathbb{P}[D \text{ rolls } 6] = \frac{1}{6}$. ← *uniform over 6 outcomes*

Terminology and notation.

"C lands heads" and "D is even" are **events** with probabilities $\mathbb{P}[C \text{ lands heads}]$, $\mathbb{P}[D \text{ rolls even}]$.

Distribution: all outcome-probability pairs.

[uniform distribution: all probabilities equal]

outcome	probability
1	1/6
2	1/6
3	1/6
4	1/6
5	1/6
6	1/6

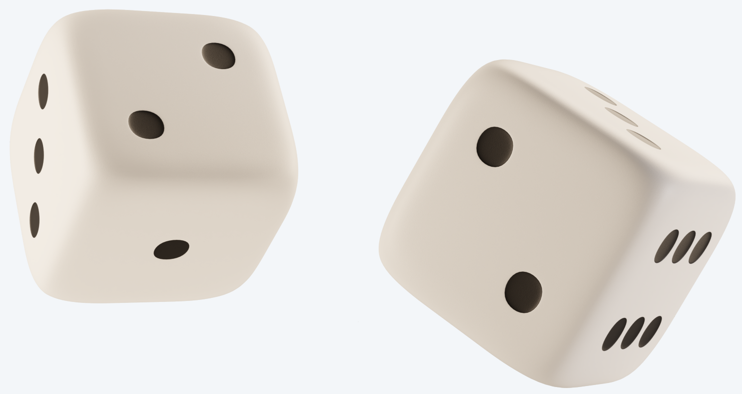
distribution of 6-sided die

The uniform distribution

Coin flip.

$\mathbb{P}[C \text{ lands heads}] = \mathbb{P}[C \text{ lands tails}] = \dfrac{1}{2}.$ ← *uniform over 2 outcomes*

Roll of a die.



$\mathbb{P}[D \text{ rolls } 1] = \mathbb{P}[D \text{ rolls } 2] = \cdots = \mathbb{P}[D \text{ rolls } 6] = \dfrac{1}{6}.$ ← *uniform over 6 outcomes*

Independent coin flips.

$\mathbb{P}[C_1 \text{ heads}, C_2 \text{ tails}, \ldots\ C_k \text{ heads}] = \dfrac{1}{2} \times \dfrac{1}{2} \cdots \times \dfrac{1}{2} = \dfrac{1}{2^k}.$ ← *uniform over 2^k outcomes*

Terminology and notation.

"C lands heads" and "D is even" are **events** with probabilities $\mathbb{P}[C \text{ lands heads}]$, $\mathbb{P}[D \text{ rolls even}]$.

Distribution: all outcome-probability pairs.

[uniform distribution: all probabilities equal]

Randomness: quiz 2

Flip a coin 6 times and count how often it lands heads. Which count is most likely?

A. 2

B. 3

C. 4

D. All of the above.

E. None of the above.

Deterministic and Randomized Algorithms

Def. A *deterministic algorithm* is an algorithm that doesn't use randomness, i.e., given a certain input, its behavior (output, running time, memory, …) is always the same

most of algorithms you've seen so far are deterministic

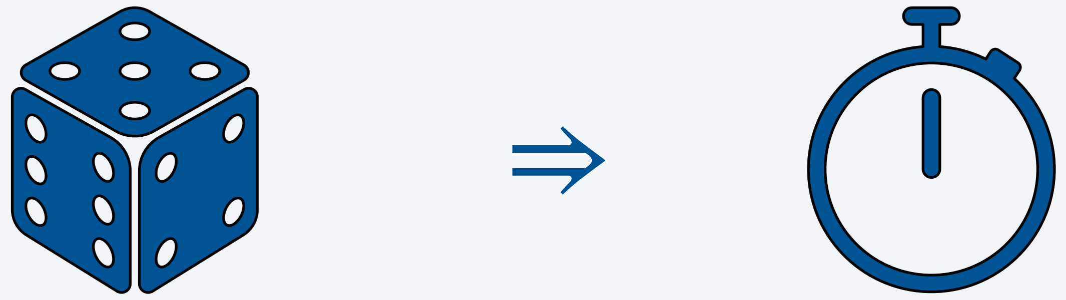
(also known as a *probabilistic algorithm*)

Def. A *randomized algorithm* is an algorithm that uses randomness as part of its logic

you've seen some randomized algorithms already! E.g., Quicksort with shuffling

randomized

Goal for today: Use probability to help us design algorithms that are better on average (i.e., most of the time)



Two "Flavors" of Algorithms Using Randomness

Monte Carlo algorithm.

- Running time is deterministic.
  [doesn't depend on coin flips]
- Not guaranteed to be correct.

Las Vegas algorithm.

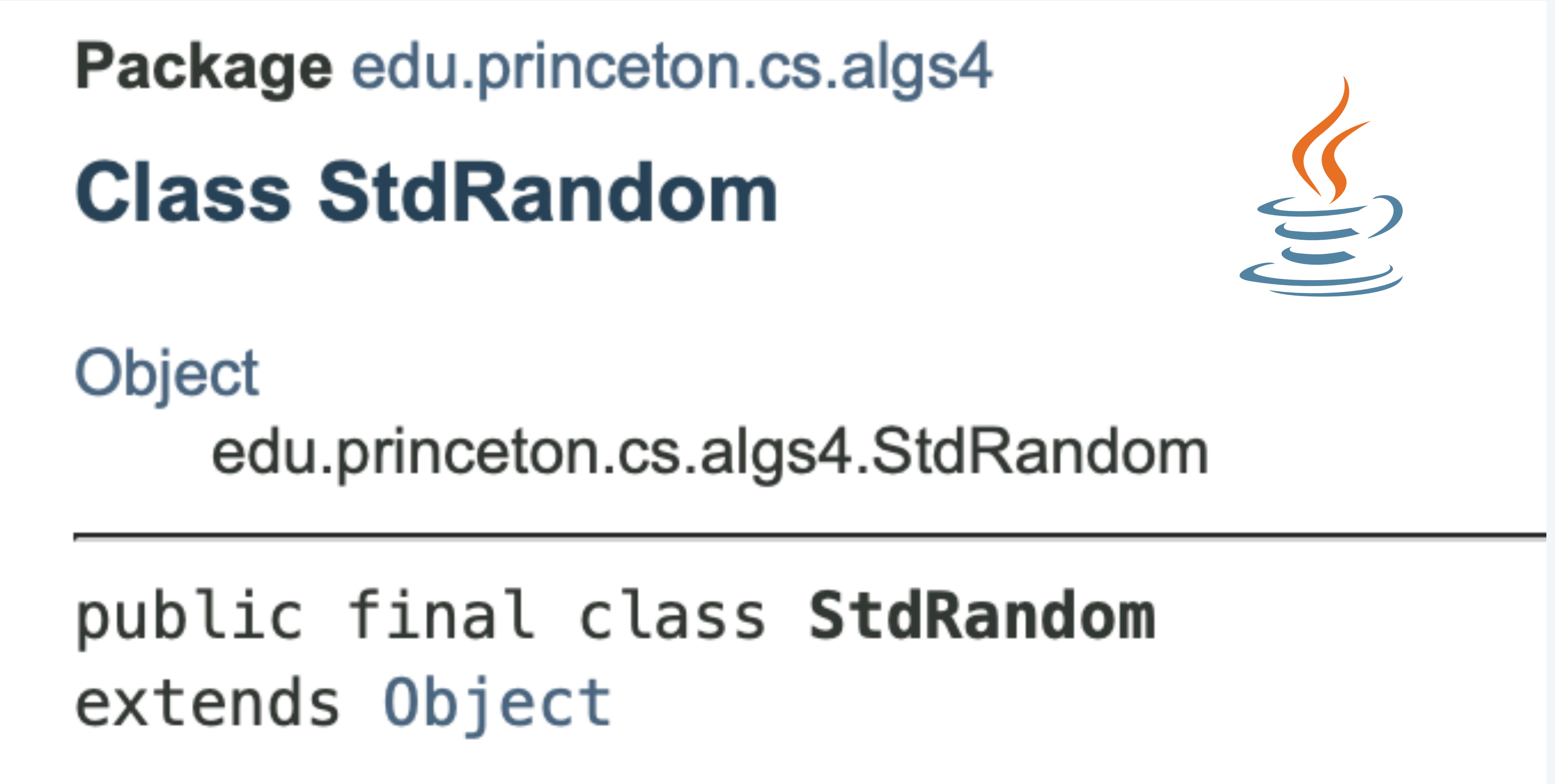
- Guaranteed to be correct.
- Running time depends on outcomes of random coin flips.

Ex. Quicksort, quickselect.

How do we use randomness?

Question. How do we toss a coin in a program?

Easy, just use `StdRandom.uniformInt(2)`


Package edu.princeton.cs.algs4

Class StdRandom

Object
    edu.princeton.cs.algs4.StdRandom

```
public final class StdRandom
extends Object
```

Question. How is `StdRandom.uniformInt(n)` implemented?

That's pretty tricky. Randomness is rare so we usually pseudorandomness: using a small amount of randomness that gets "boosted" into a large amount of something that looks random

E.g., this is like simulating tossing n coins by tossing a small number of coins (say $\sim \log n$)

https://algs4.cs.princeton.edu

Randomness

- randomness and algorithms
- **treasure hunt problem**
- duplicates finding
- Karger's algorithm
- more applications

Treasure Hunt Problem

Input. An array a of length n containing 50% treasures and 50% duds (i.e., empty)

Output. Any index containing a treasure

Goal. Minimize array accesses

Treasure Hunt Problem - Deterministic Algorithms

Deterministic algorithms.

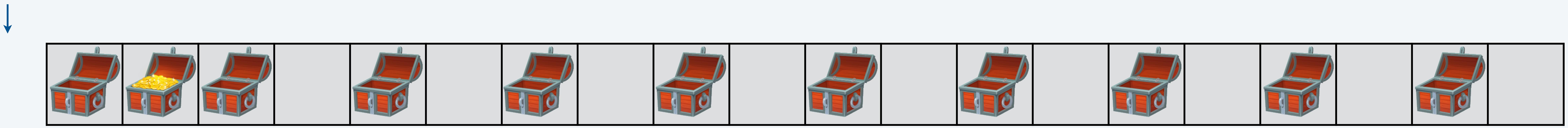
- scan the array left-to-right; return once treasure found. ⟵ $\frac{n}{2} + 1$ *accesses in worst case*

Treasure Hunt Problem - Deterministic Algorithms

Deterministic algorithms.

- scan the array left-to-right; return once treasure found. ⟵ $\frac{n}{2} + 1$ *accesses in worst case*
- scan the array right-to-left; return once treasure found. ⟵ $\frac{n}{2} + 1$ *accesses in worst case*

Treasure Hunt Problem - Deterministic Algorithms



Deterministic algorithms.

- scan the array left-to-right; return once treasure found. ← $\frac{n}{2} + 1$ *accesses in worst case*
- scan the array right-to-left; return once treasure found. ← $\frac{n}{2} + 1$ *accesses in worst case*
- look at even entries, then odd; return once treasure found. ← $\frac{n}{2} + 1$ *accesses in worst case*

Treasure Hunt Problem - Deterministic Algorithms

Proposition. For every deterministic algorithm, there is a 50%-treasure array where it makes $\frac{n}{2} + 1$ accesses.

Pf.

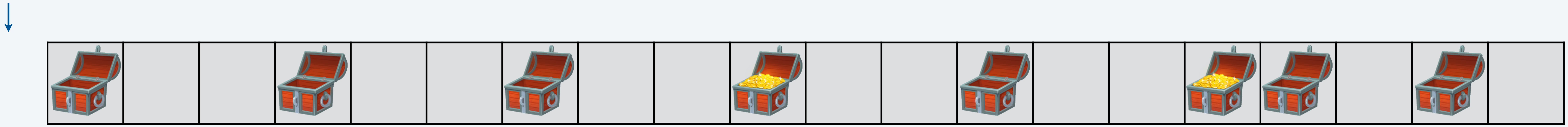
A deterministic algorithm always accesses the array in the same order

Consider the sequence of the first $n/2$ accesses it makes

Create an array with duds on those positions and treasures elsewhere, it requires $\frac{n}{2} + 1$ accesses

Treasure Hunt Problem - A Monte Carlo Algorithm

What can we do with randomness?



Randomized algorithm (Monte Carlo):

- look at `a[StdRandom.uniformInt(n)]`, return treasure (if found). ← *1 flip lands tails*
- look at two uniformly random entries, return 1st treasure found (if any). ← *2 flips land tails*
- look at three uniformly random entries, return 1st treasure found (if any). ← *3 flips land tails*
- look at k uniformly random entries, return 1st treasure found (if any).

Fails with probability $\frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} \ldots \times \frac{1}{2} = \frac{1}{2^k}$ =

Treasure Hunt Problem: A Monte Carlo Algorithm

```
int treasureHuntMonteCarlo(int[] a, int k) {
    for (int i = 0; i < k; i++) {
        if (a[StdRandom.uniformInt(a.length)] == 1)
            return i;
    }
    return -1; // Fail
}
```

Properties.

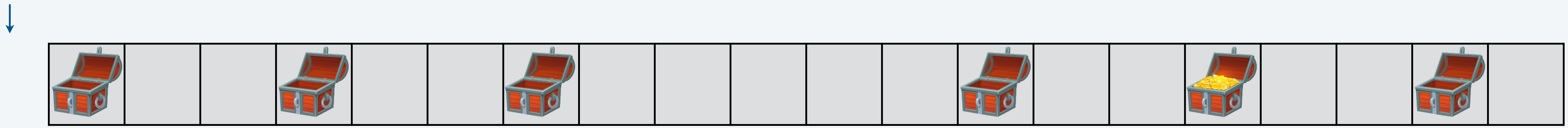
- Number of accesses = $O(k)$
- Failure probability = $\mathbb{P}[k$ coin flips land tails$] = \dfrac{1}{2^k}$

If we want a $99\,\%$ probability of success then:

- Pick $k = 7$, then number of accesses is $O(1)$
- Failure probability $\leq 1\,\%$

Treasure Hunt Problem: A Las Vegas Algorithm

What if we always want to be correct?



Randomized algorithm (Las Vegas):

- repeatedly look at uniformly random entry; return *only when* treasure found.

Returns in 1st try with probability $1/2$.

Returns in 2nd try with probability $1/4$.

⋮

Returns in kth try with probability $1/2^k$.

Randomness: quiz 3

At most how many array accesses made by Las Vegas treasure hunt?

(Recall: we can look at the same entry twice.)

A. 1

B. 2

C. $n/2$

D. n

E. None of the above.

Las Vegas Algorithms - Expected Value

Definition. The *expected number of accesses* of an algorithm A on a given input I is the average number of accesses, weighted by $\mathbb{P}[A(I)$ makes k accesses$]$ for all possible k. It's given by the following formula:

$$E(A, I) = 1 \times \mathbb{P}[A(I) \text{ makes } 1 \text{ access}] + 2 \times \mathbb{P}[A(I) \text{ makes } 2 \text{ accesses}] + 3 \times \mathbb{P}[A(I) \text{ makes } 3 \text{ accesses}] + \cdots$$

Definition. The *worst-case expected number of accesses* of an algorithm A is the maximum of the expected number of accesses over all possible inputs. It's given by the following formula:

$$E(A) = \max_I E(A, I)$$

Note. The above definition is a "worst-case" definition, the probability is over the randomness in the algorithm, not the randomness of the input

We can extend the above definitions to any other cost model (running time, compares, memory). For example, *expected running time* is given by:

$$T(A, I) = 1 \times \mathbb{P}[A(I) \text{ takes } 1 \text{ units of time}] + 2 \times \mathbb{P}[A(I) \text{ takes } 2 \text{ units of time}] + 3 \times \mathbb{P}[A(I) \text{ takes } 3 \text{ units of time}] + \cdots$$

Example. We previously saw that the worst-case expected number of compares Quicksort does is $\sim 2n \ln n$

Treasure Hunt Problem: A Las Vegas Algorithm

```
int treasureHuntLasVegas(int[] a, int k) {
    while (true) {
        if (a[StdRandom.uniformInt(a.length)] == 1)
            return i;
    }
}
```

Worst-case expected number of accesses.

$$1 \times \mathbb{P}[A(I) \text{ makes } 1 \text{ access}] + 2 \times \mathbb{P}[A(I) \text{ makes } 2 \text{ accesses}] + \cdots = 1 \times \frac{1}{2} + 2 \times \frac{1}{4} + \ldots + k\frac{1}{2^k} + \ldots = 2$$

Which is $O(1)$!

variant of geometric sum:

$$\lim_{n \to \infty} \frac{1}{2} + \frac{2}{4} + \ldots + \frac{i}{2^n} = 2$$

Treasure Hunt Summary

	doesn't need randomness	worst-case accesses	expected accesses	can't fail?
deterministic	✔	$\frac{n}{2}+1$	$\frac{n}{2}+1$	✔
Monte Carlo		$O(1)$	$O(1)$	
Las Vegas		∞	$O(1)$	✔

Randomness: quiz 4

Suppose 1% of the array contains treasure and 99% contain duds. Then `a[StdRandom.uniformInt(n)]` **finds a treasure with probability**

A. 1%

B. 10%

C. 50%

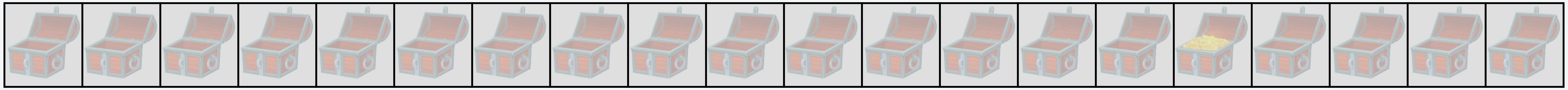
D. 99%

E. None of the above.

Rare treasures and biased coins

Input. An array of length n containing **1%** treasures and **99%** duds (i.e., empty)

Output. Any index containing a treasure



Randomized algorithm (Monte Carlo):

- look at k uniformly random entries, return treasure (if found).

Failure probability = $\mathbb{P}$[k biased coin flips land tails]

= $(0.99)^k$.

outcome	probability
heads	1/100
tails	99/100

distribution of 99%-1% biased coin

Example. If we want $0.99^k < 1\,\%$, setting $k = 459$ suffices!

Error Reduction

Note. We can generalize the previous method to any algorithm. Suppose we have a randomized algorithm A

Error reduction.

If $\mathbb{P}[A \text{ fails}] = p$ and want failure $\leq q$, repeat $k \geq \log_p q$ times.

Then, $\mathbb{P}[A \text{ fails } k \text{ times}] = p^k \leq q$.

independence

https://algs4.cs.princeton.edu

Randomness

- randomness and algorithms
- treasure hunt problem
- **duplicates finding**
- Karger's algorithm
- more applications

Duplicates Finding Problem

Input. An array a of length n containing $n/2$ pairs of integers, one per element between 1 and $n/2$

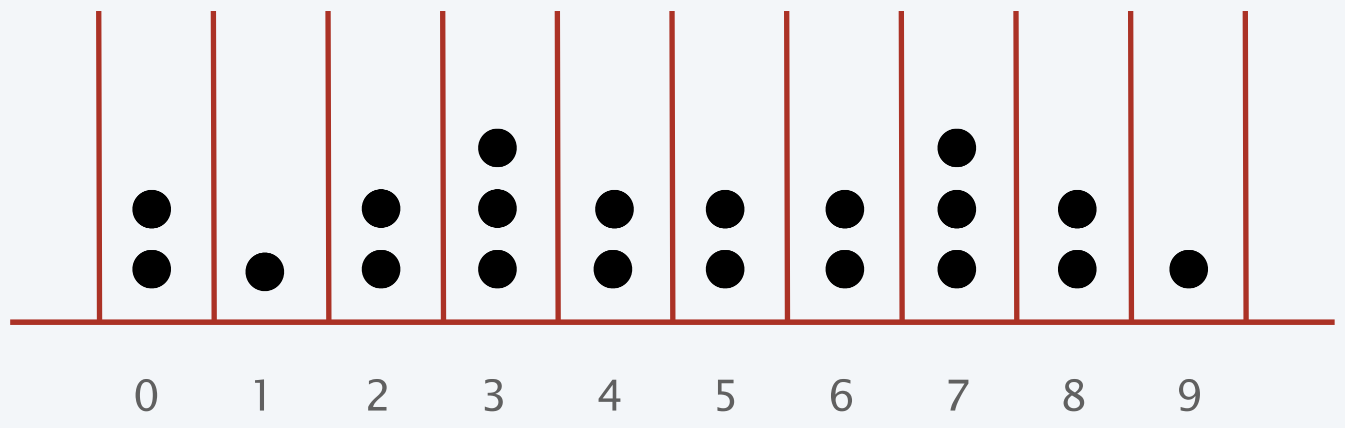
Output. Any two indices that have the same integer

$n = 12$

1	4	5	3	2	5	2	3	4	6	1	6

Goal. Minimize array accesses

Motivation. Finding collisions in hash tables



Duplicates Finding Problem - Deterministic Algorithms

1	4	5	3	2	5	2	3	4	6	1	6

Deterministic algorithms.

- scan the array left-to-right; keep a counter array; return once a pair is found.
- scan the array right-to-left; keep a counter array; return once a pair is found.

Proposition. For every deterministic algorithm, there is an array where it makes $\frac{n}{2} + 1$ accesses.

Pf.

Same as in the case of the treasure hunt problem

Duplicates Finding Problem - A Monte Carlo Algorithm

1	4	5	3	2	5	2	3	4	6	1	6

Randomized algorithm (Monte Carlo):

Repeat k times:

- pick two distinct uniformly random entries, `i1` and `i2`
- return them if `a[i1] == a[i2]`

Failure probability (of one single iteration). It's the probability that we exactly find the pair, which is $\frac{1}{n-1}$

If we want a $99\,\%$ probability of success then we need $k = O(n)$, so this is no better than the deterministic one!

Duplicates Finding Problem - A Monte Carlo Algorithm - Take 2

1	4	5	3	2	5	2	3	4	6	1	6

Randomized algorithm 2 (Monte Carlo):

- pick k uniformly random entries (not necessarily distinct)
- check if there is a pair among them; if so return it

Theorem (birthday paradox). Suppose we draw a integers uniformly from 1 to b (and $a < b$). Then the probability we get the same number twice is $1 - \exp\left(-\frac{a(a-1)}{2b}\right) = 1 - \exp\left(-O\left(\frac{a^2}{b}\right)\right)$

when $b = 365$, this is the probability two people in a room of a people share a birthday

Failure probability of the algorithm. Since there are $\frac{n}{2}$ distinct values appearing the same number of times,

picking a random array index is the same as picking a uniformly random integer from 1 to $\frac{n}{2}$.

So put $a = k$ and $b = n/2$ in the birthday paradox theorem and we get $\exp\left(-O\left(\frac{k^2}{n}\right)\right)$

If we want a $99\,\%$ probability of success then pick $k = 4\sqrt{n}$ and the above becomes smaller than $1\,\%$

Duplicates Finding Problem - A Monte Carlo Algorithm - Take 2

1	4	5	3	2	5	2	3	4	6	1	6

```java
Pair duplicatesFindingMonteCarlo(int[] a) {
    int n = a.length;
    LinearProbingHashST<Integer, Integer> seen = new LinearProbingHashST<Integer, Integer>();
    for (int i = 0; i*i < 16 * n; i++) {
        int randomId = StdRandom.uniformInt(n);
        if (seen.contains(a[randomId])
            return new Pair(randomId, seen.get(a[randomId]));
        seen.put(a[randomId], randomId);
    }
    return null; // Fail
}
```

$O\left(\sqrt{n}\right)$ accesses versus $O(n)$ for a deterministic algorithm

https://algs4.cs.princeton.edu

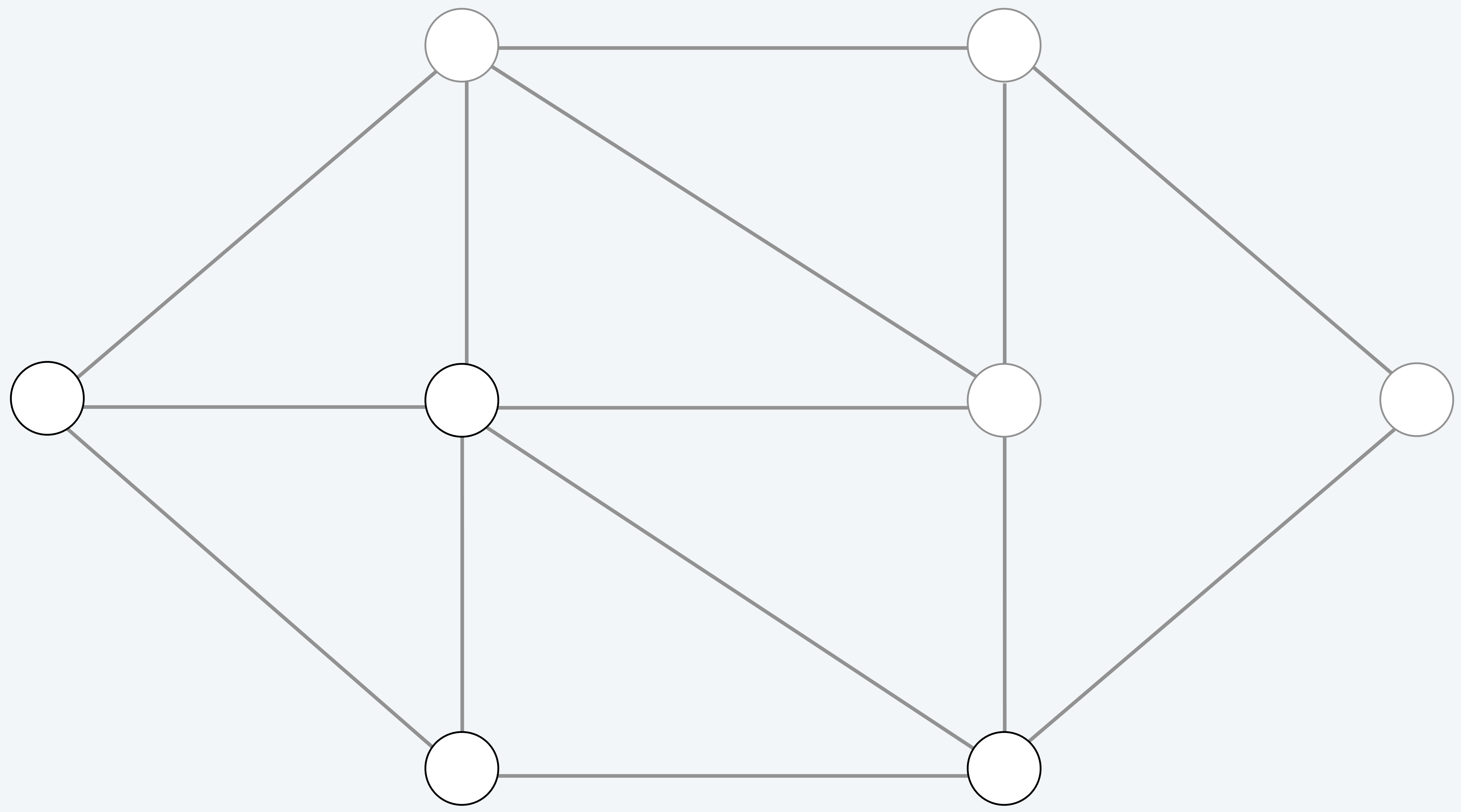
Randomness

- *randomness and algorithms*
- *treasure hunt problem*
- *duplicates finding*
- *Karger's algorithm*
- *more applications*

Global minimum cut problem

Goal. Find cut in undirected graph with fewest edges (for any source and sink).

Equivalent. Smallest min st-cut among all pairs (s, t) with antiparallel edges of capacity 1.

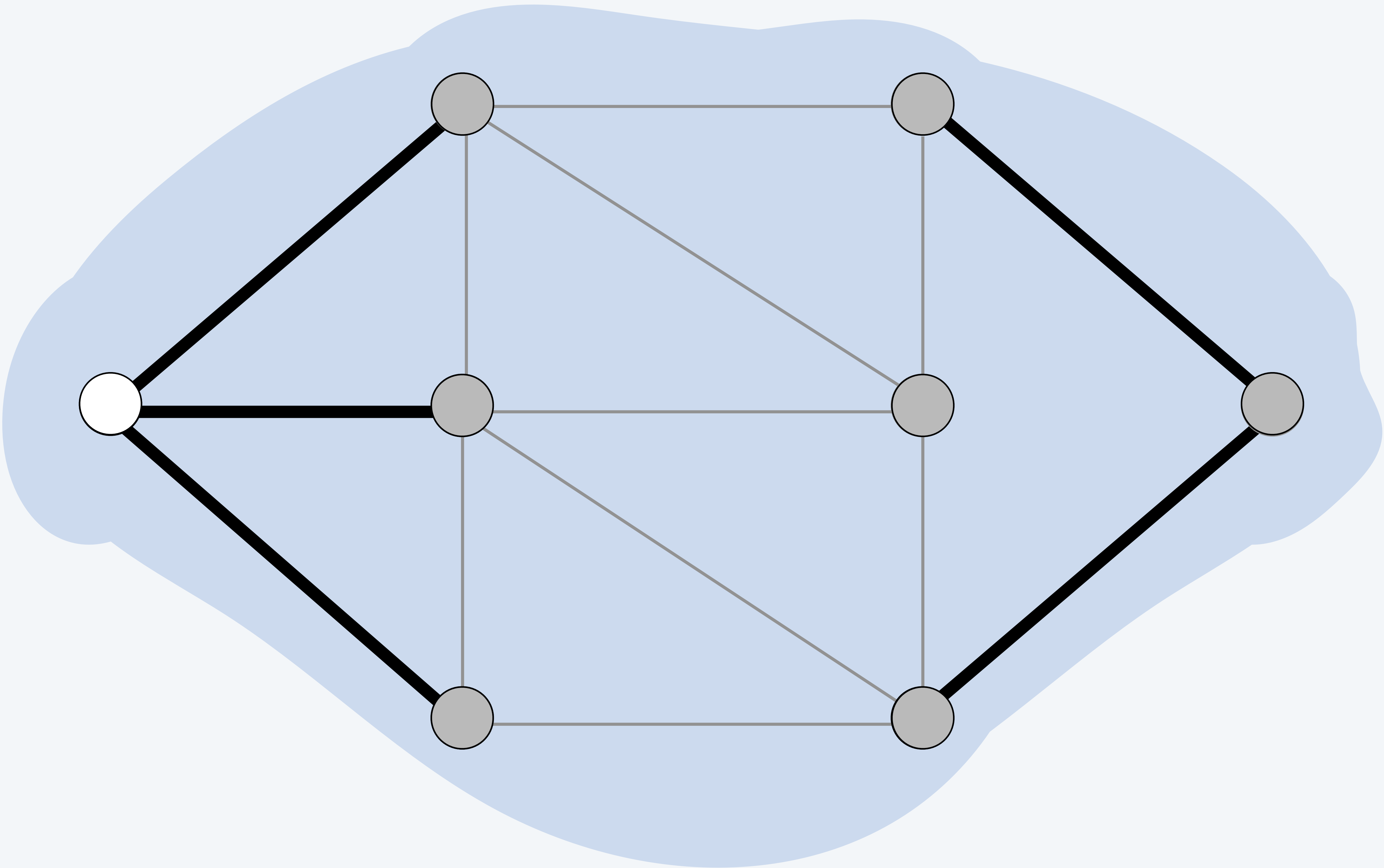


Global minimum cut problem - Deterministic Algorithms

Goal. Find cut in undirected graph with fewest edges (for any source and sink).

Deterministic algorithms.

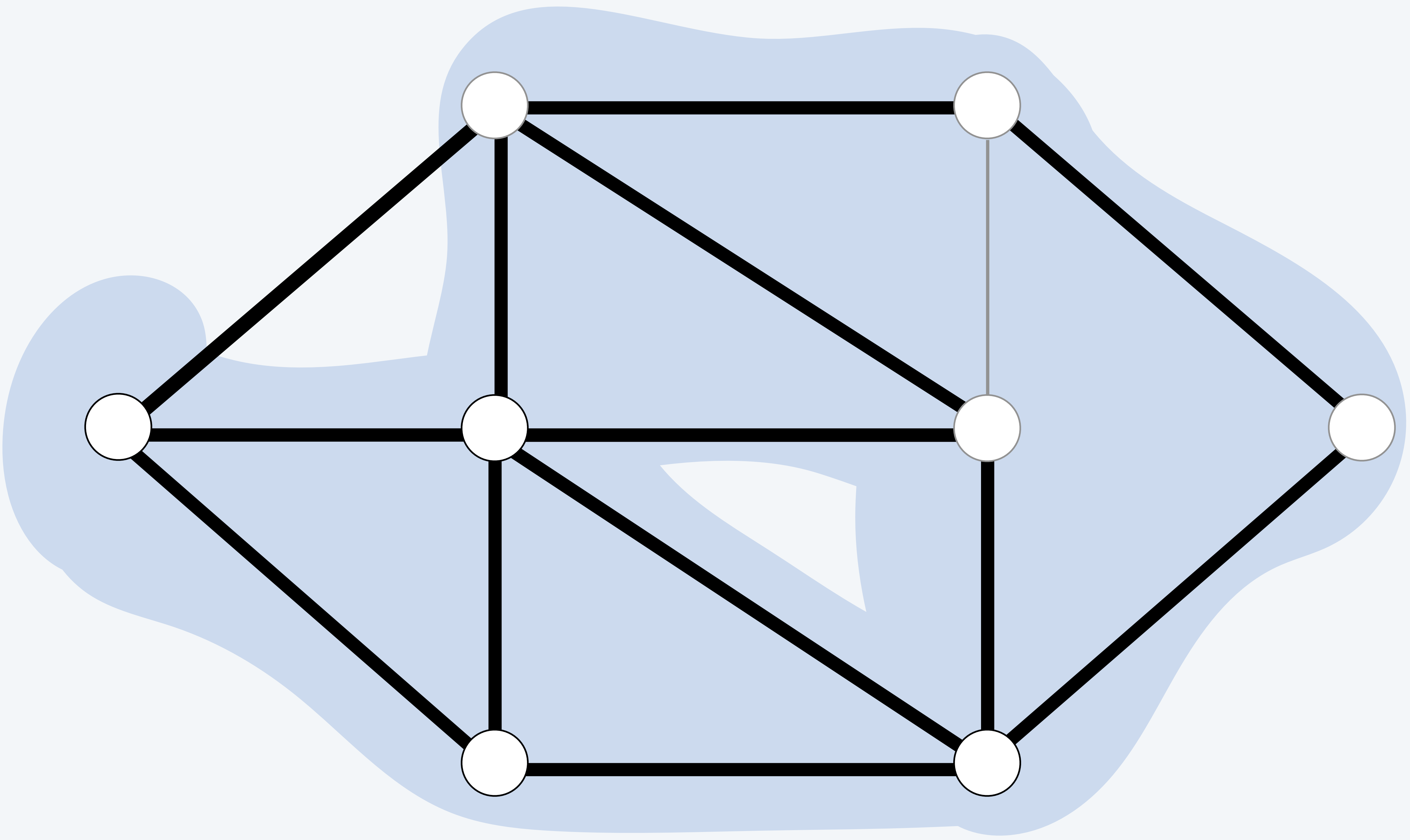
- Brute-force: iterate over all cuts, return smallest. [$2^{V-1} - 1$ cuts $\Longrightarrow$ exponential time!]
- Ford-Fulkerson-based: pick any s as source, try every t as target. [$V - 1$ runs of FF $\Longrightarrow$ $\Theta(VE^2)$ runtime.]



Global minimum cut problem - Randomized Attempt 1

Goal. Find cut in undirected graph with fewest edges (for any source and sink).

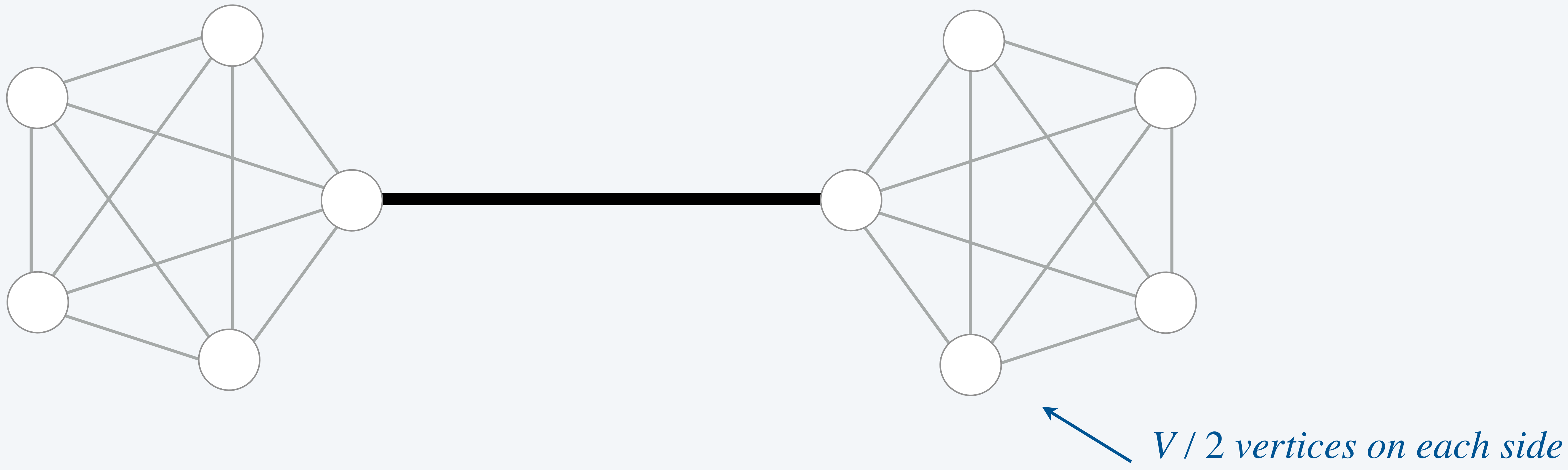
Idea. Pick a uniformly random cut, by tossing a coin per vertex and keeping the ones that landed heads



Global minimum cut problem - Randomized Attempt 1

How good is it? There may be 1 mincut but $O(2^V)$ total cuts — takes a *lot* of luck to find it

Example.



The algorithm only succeeds if all of the vertices on one side are picked and none on the other side, which happens with probability $2 \times \frac{1}{2^V} = \frac{1}{2^{V-1}}$

Failure probability. Is $1 - \frac{1}{O(2^V)}$ for this graph, which is pretty low

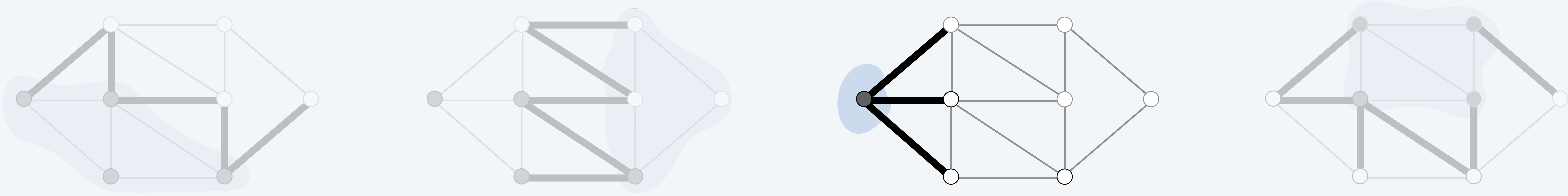
We can try running algorithm many times and return best cut.

If we want $99\,\%$ success probability we need to repeat $O(2^V)$ times ☹️

Global minimum cut problem - Randomized Attempt 2 - Karger's algorithm

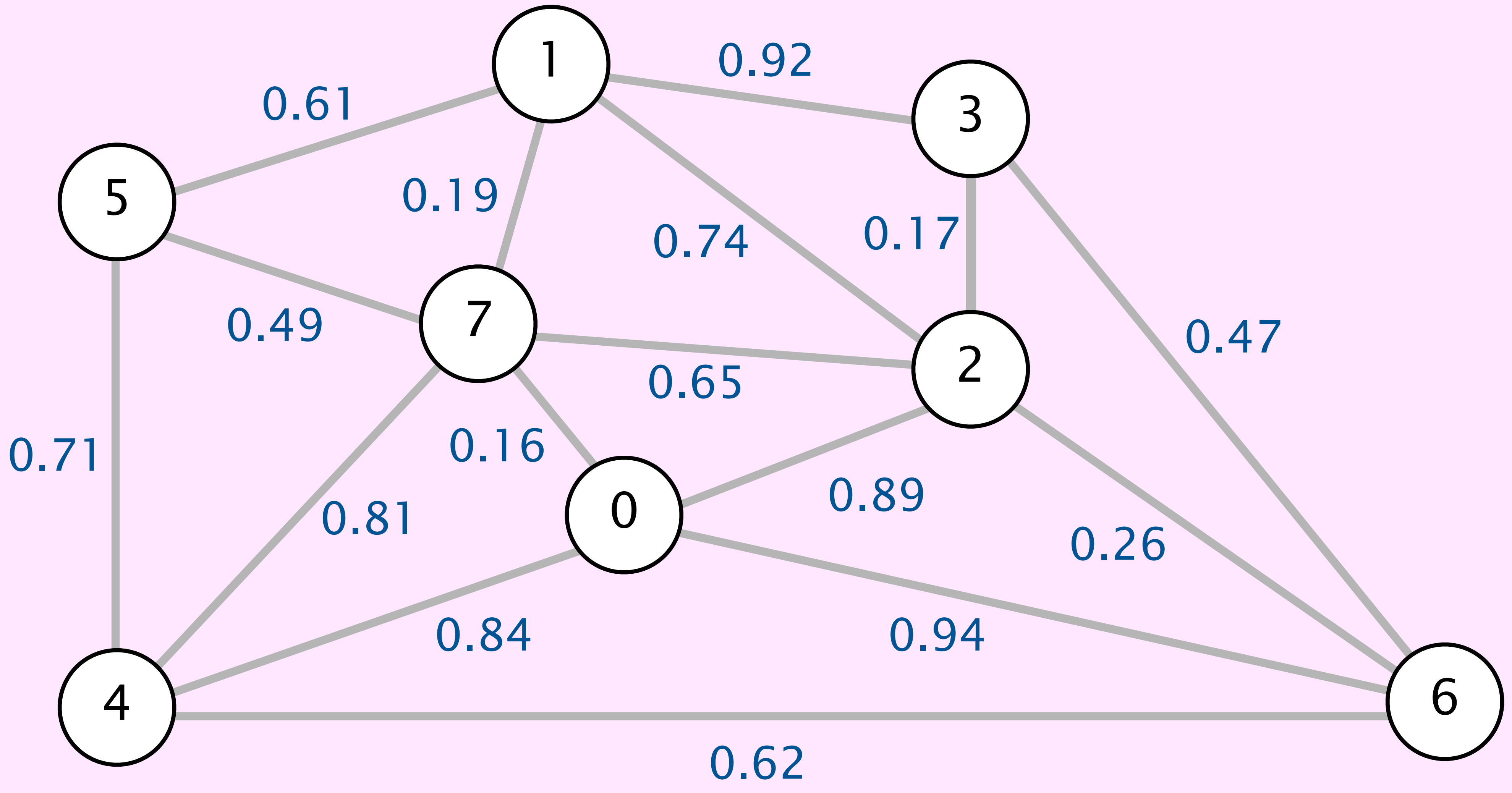
Algorithm.

- Assign a random weight (uniform between 0 and 1) to each edge e.
- Run Kruskal's MST algorithm until 2 connected components left.
- Return cut defined by connected components.



Karger's algorithm demo

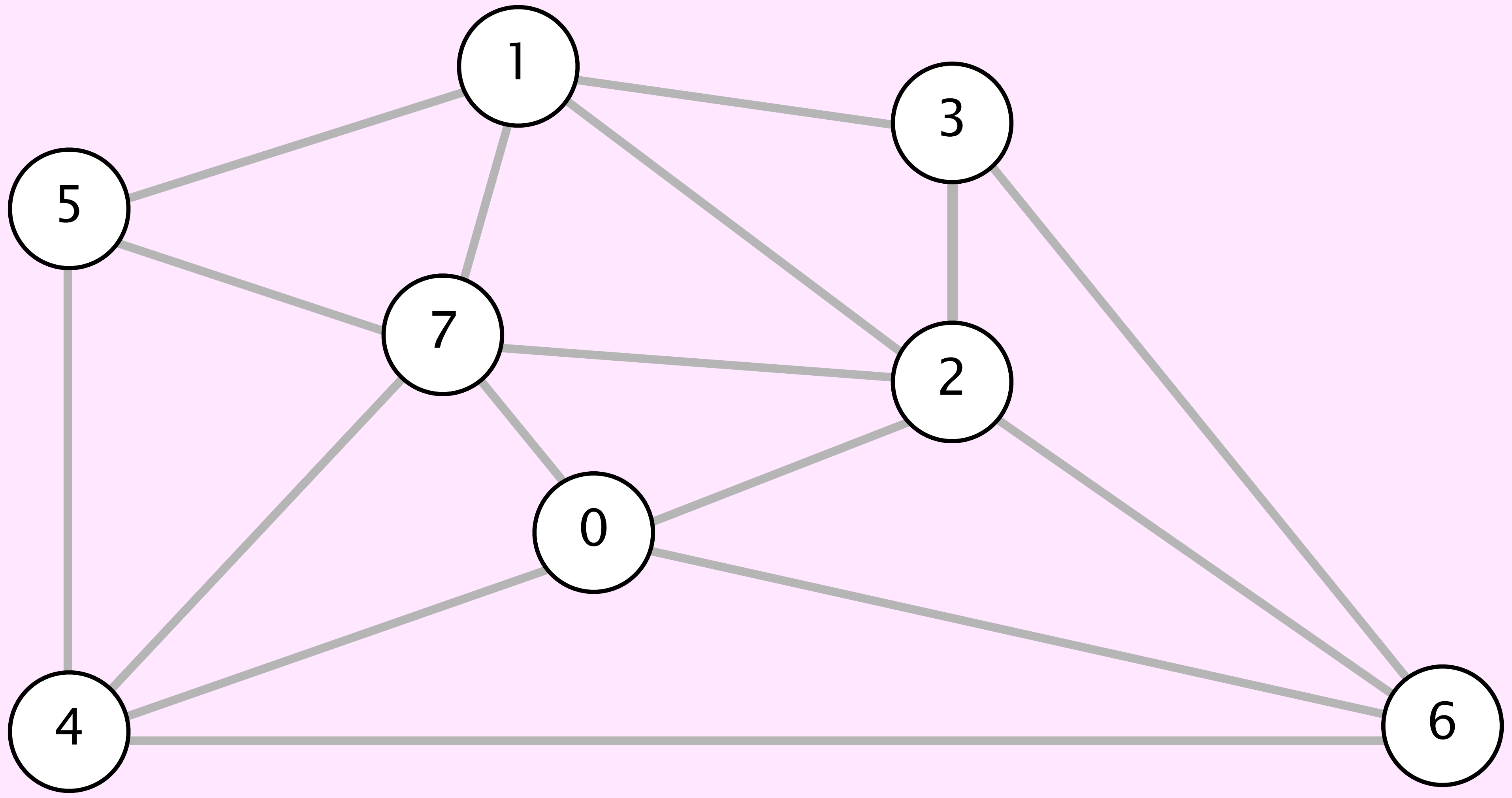
- **Assign random edge weights.**
- Run Kruskal's algorithm until 2 connected components left.



0-2	0.89
0-4	0.84
0-7	0.16
0-6	0.94
1-2	0.74
1-3	0.92
1-5	0.61
1-7	0.19
2-3	0.17
2-6	0.26
2-7	0.65
3-6	0.47
4-6	0.62
4-5	0.71
4-7	0.81
5-7	0.49

Karger's algorithm demo

- Assign random edge weights.
- **Run Kruskal's algorithm until 2 connected components left.**



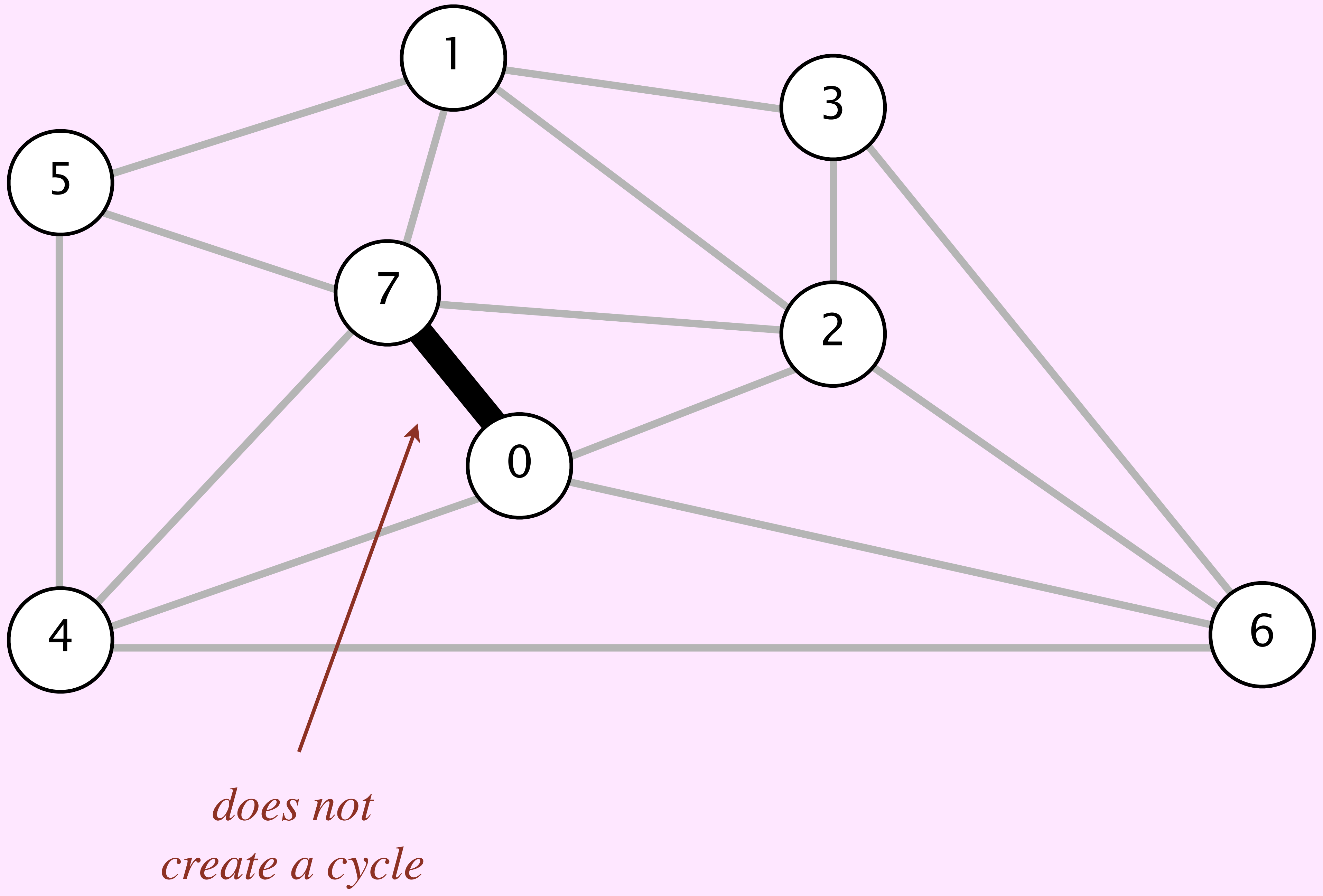
graph edges sorted by weight

0-7	0.16
2-3	0.17
1-7	0.19
2-6	0.26
3-6	0.47
5-7	0.49
1-5	0.61
4-6	0.62
2-7	0.65
4-5	0.71
1-2	0.74
4-7	0.81
0-4	0.84
0-2	0.89
1-3	0.92
0-6	0.94

Karger's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to T unless doing so would create a cycle.
- Stop if T contains $V - 2$ edges.



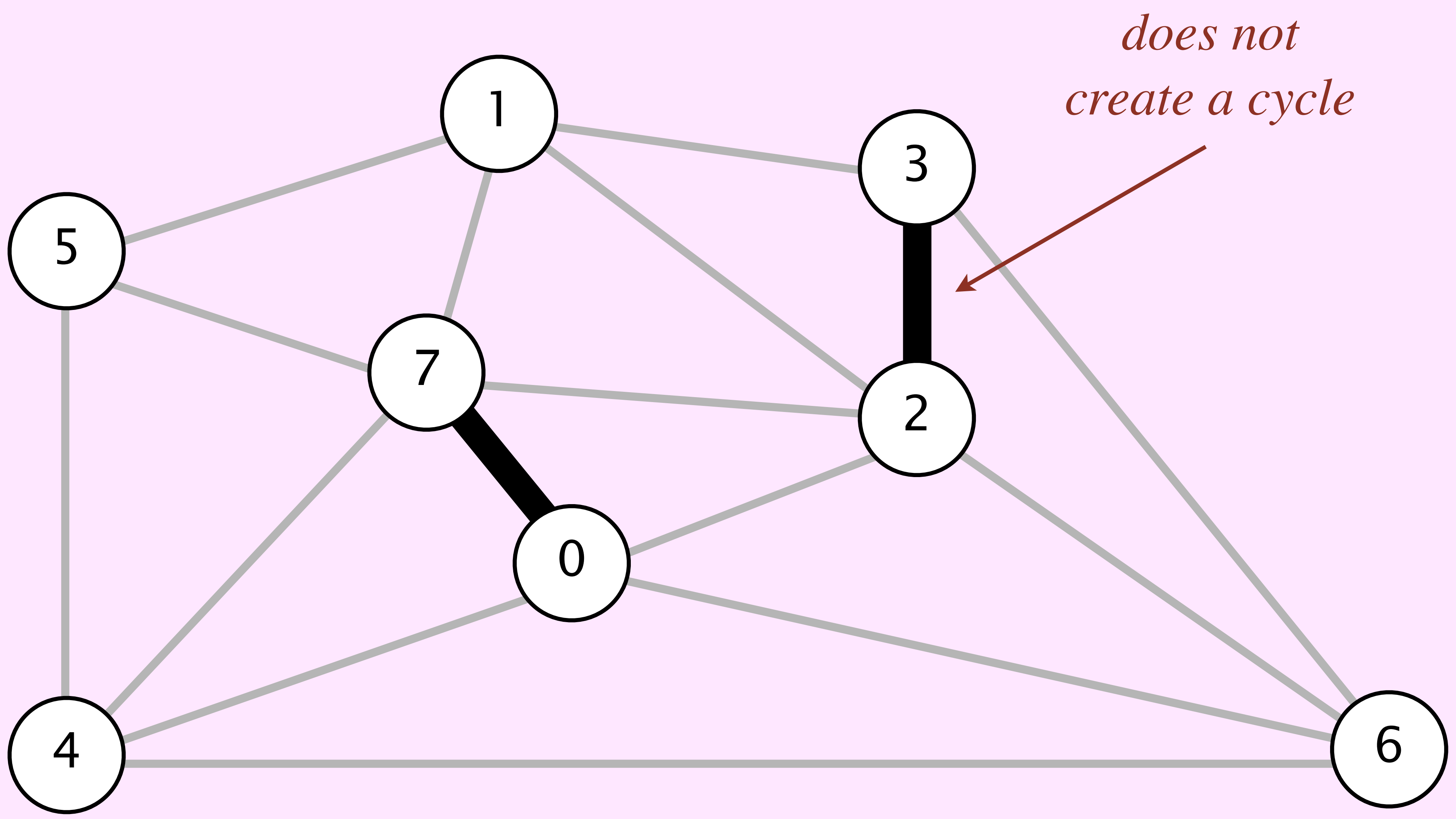
in MST →

0-7	0.16
2-3	0.17
1-7	0.19
2-6	0.26
3-6	0.47
5-7	0.49
1-5	0.61
4-6	0.62
2-7	0.65
4-5	0.71
1-2	0.74
4-7	0.81
0-4	0.84
0-2	0.89
1-3	0.92
0-6	0.94

Karger’s algorithm demo

Consider edges in ascending order of weight.

- Add next edge to T unless doing so would create a cycle.
- Stop if T contains $V-2$ edges.



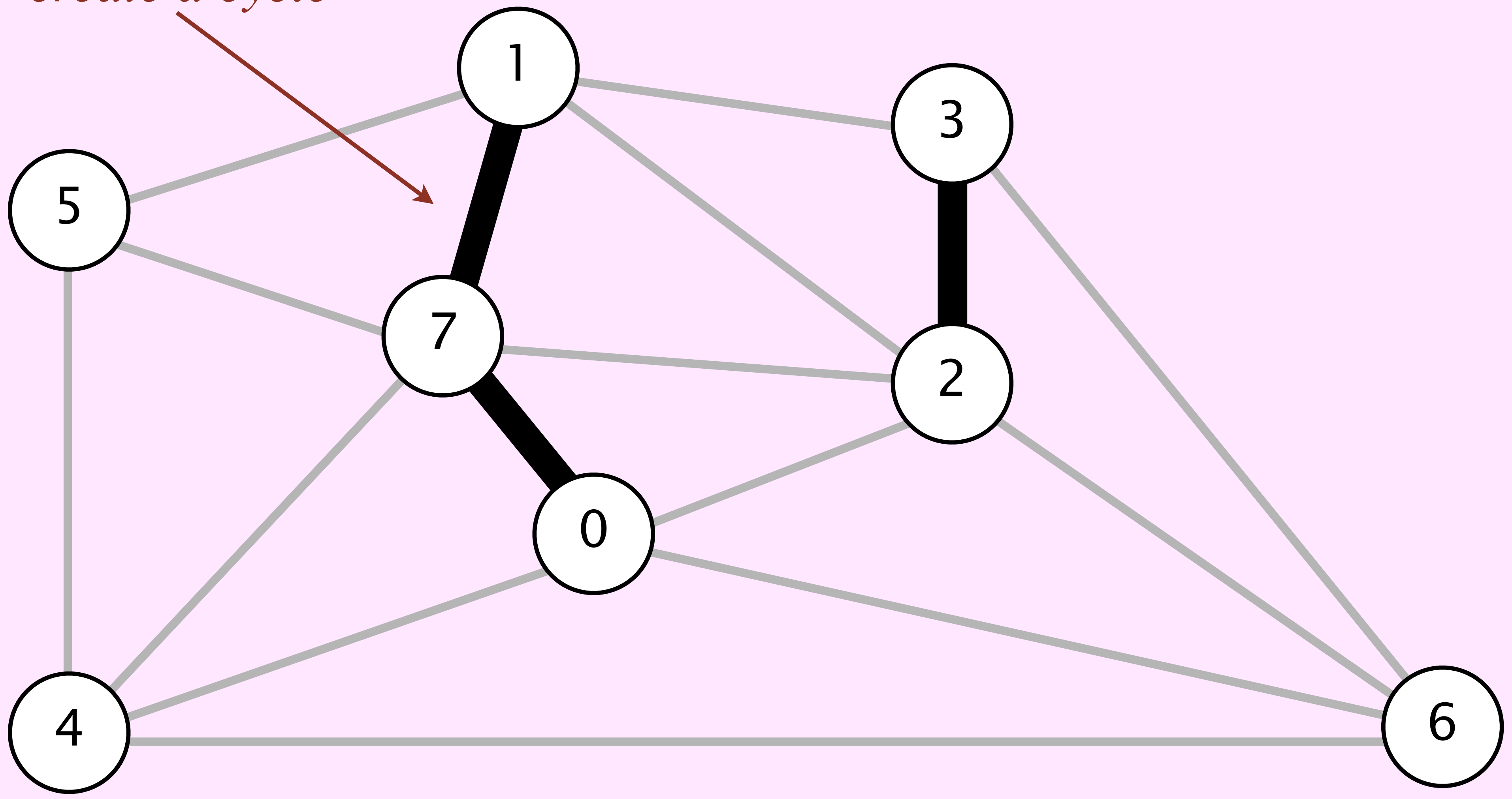
	0-7	0.16
in MST →	2-3	0.17
	1-7	0.19
	2-6	0.26
	3-6	0.47
	5-7	0.49
	1-5	0.61
	4-6	0.62
	2-7	0.65
	4-5	0.71
	1-2	0.74
	4-7	0.81
	0-4	0.84
	0-2	0.89
	1-3	0.92
	0-6	0.94

Karger's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to T unless doing so would create a cycle.
- Stop if T contains $V-2$ edges.

does not create a cycle



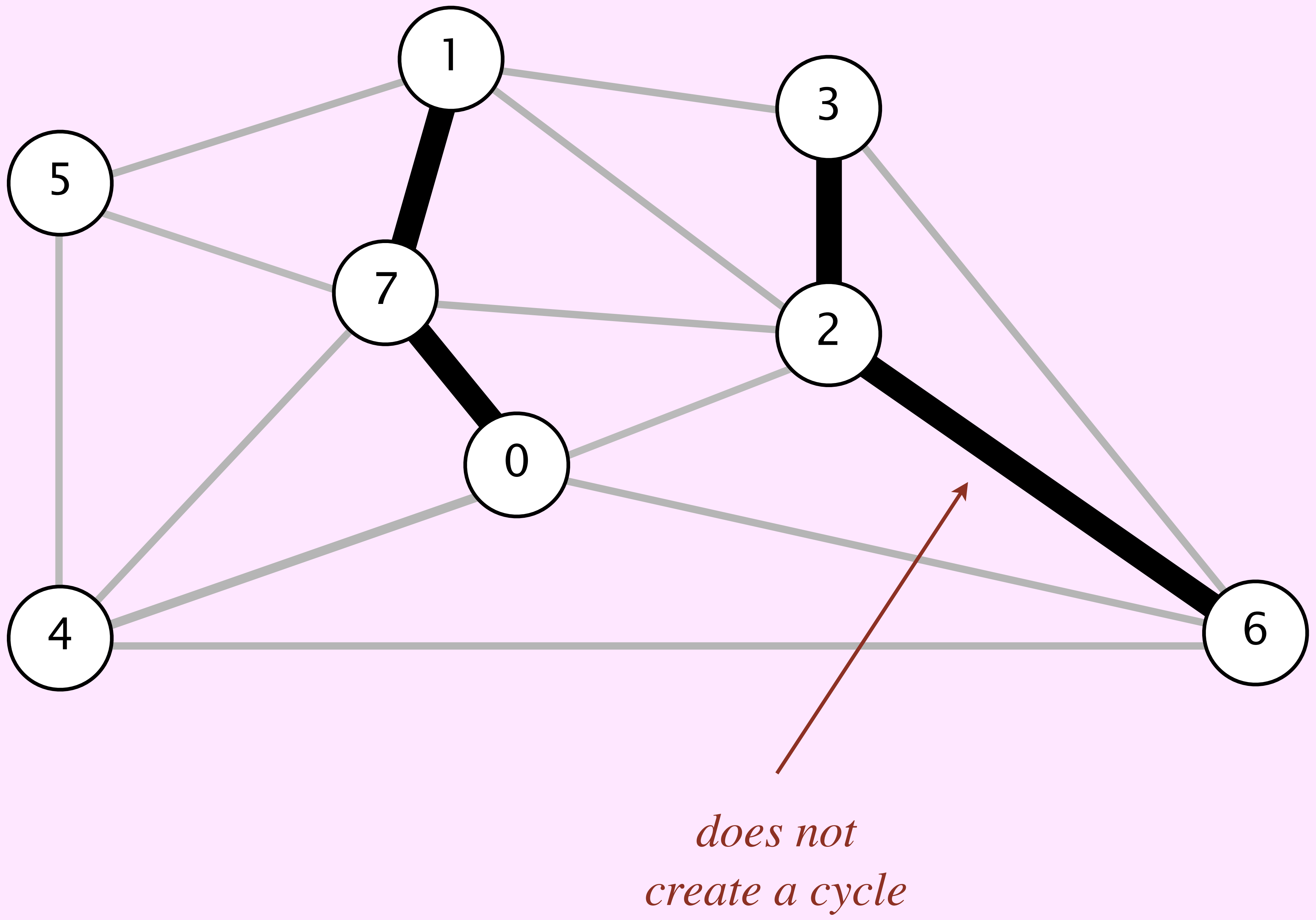
in MST →

0-7	0.16
2-3	0.17
1-7	0.19
2-6	0.26
3-6	0.47
5-7	0.49
1-5	0.61
4-6	0.62
2-7	0.65
4-5	0.71
1-2	0.74
4-7	0.81
0-4	0.84
0-2	0.89
1-3	0.92
0-6	0.94

Karger's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to T unless doing so would create a cycle.
- Stop if T contains $V - 2$ edges.

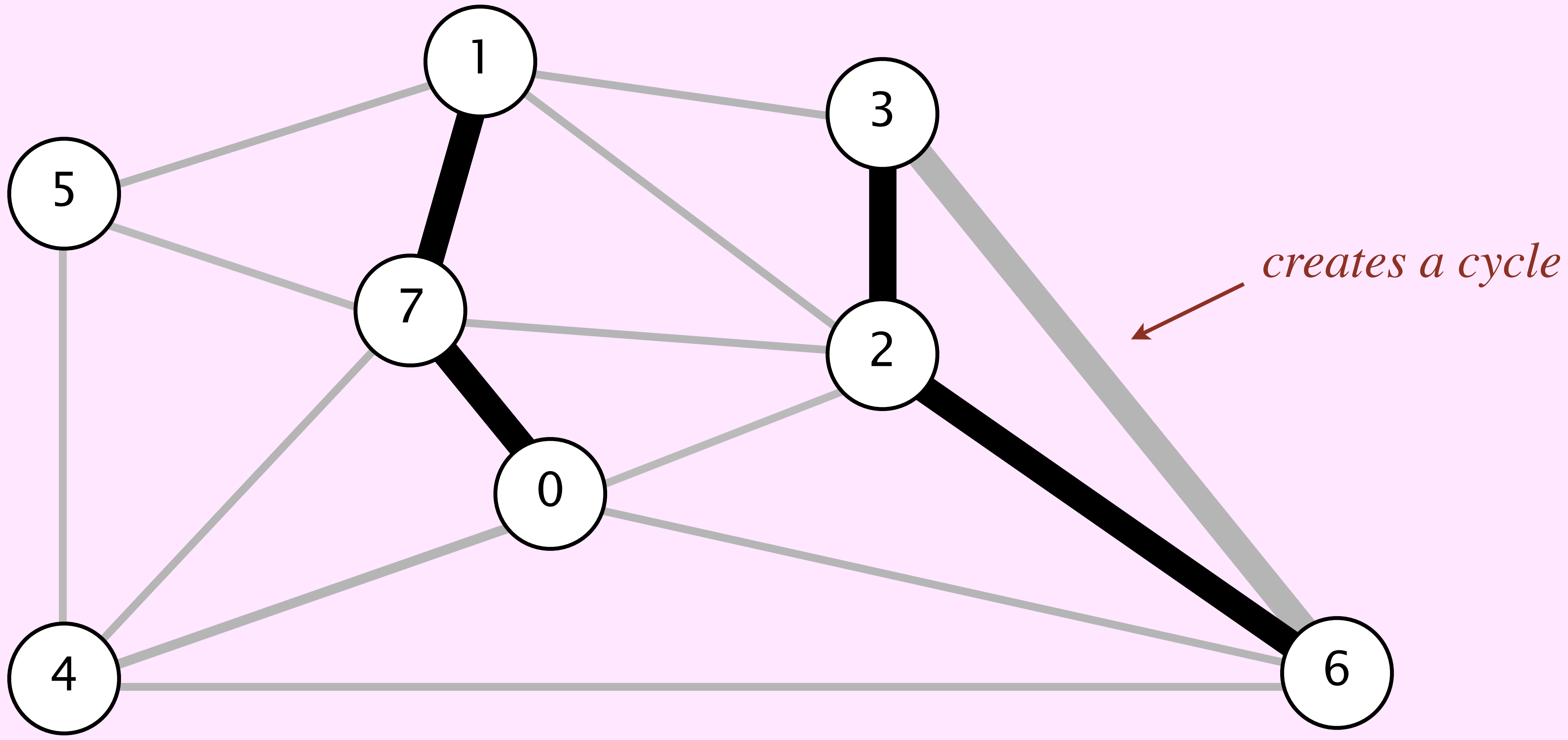


	0-7	0.16
	2-3	0.17
	1-7	0.19
in MST →	2-6	0.26
	3-6	0.47
	5-7	0.49
	1-5	0.61
	4-6	0.62
	2-7	0.65
	4-5	0.71
	1-2	0.74
	4-7	0.81
	0-4	0.84
	0-2	0.89
	1-3	0.92
	0-6	0.94

Karger's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to T unless doing so would create a cycle.
- Stop if T contains $V-2$ edges.



not in MST →

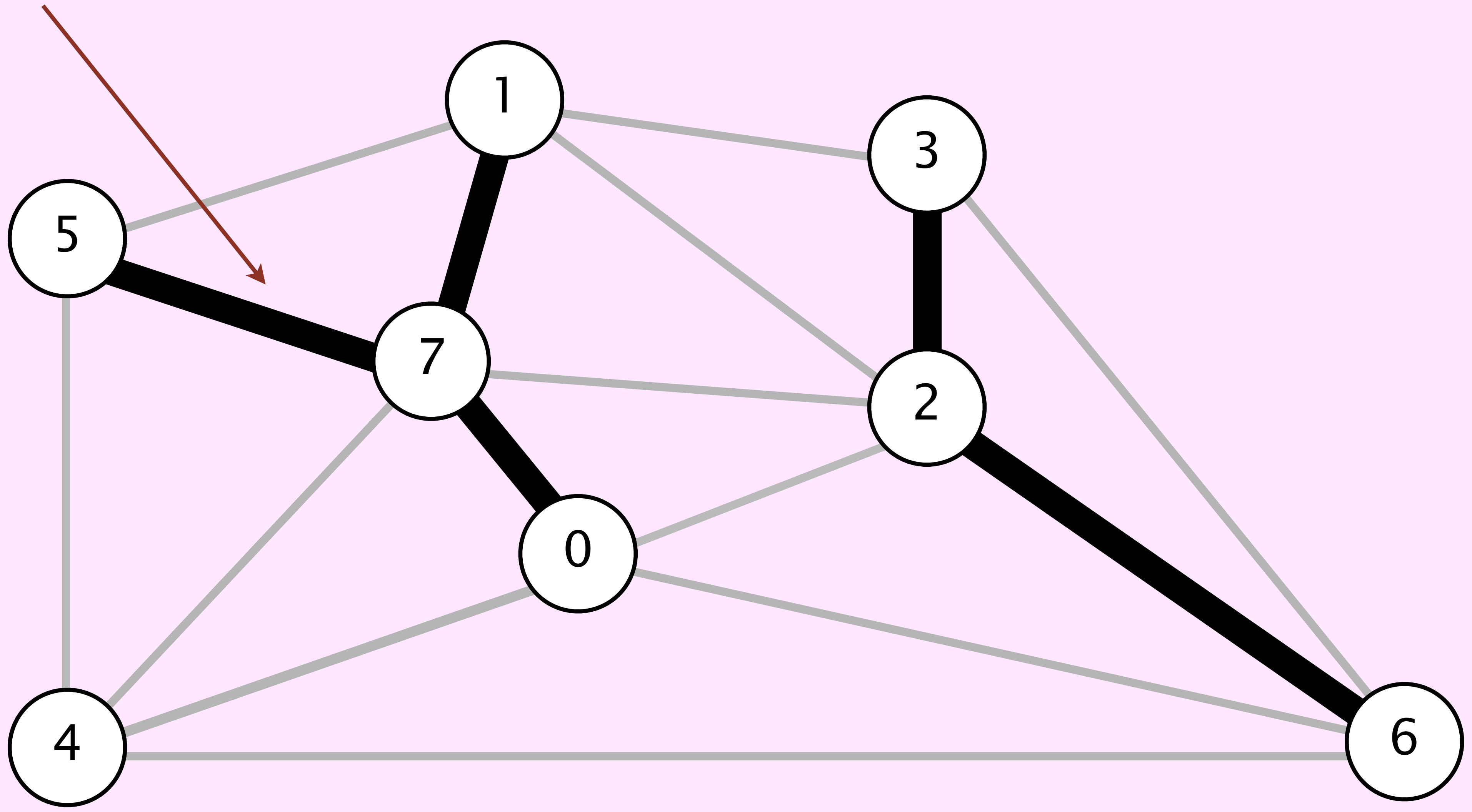
0-7	0.16
2-3	0.17
1-7	0.19
2-6	0.26
3-6	0.47
5-7	0.49
1-5	0.61
4-6	0.62
2-7	0.65
4-5	0.71
1-2	0.74
4-7	0.81
0-4	0.84
0-2	0.89
1-3	0.92
0-6	0.94

Karger's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to T unless doing so would create a cycle.
- Stop if T contains $V-2$ edges.

does not create a cycle

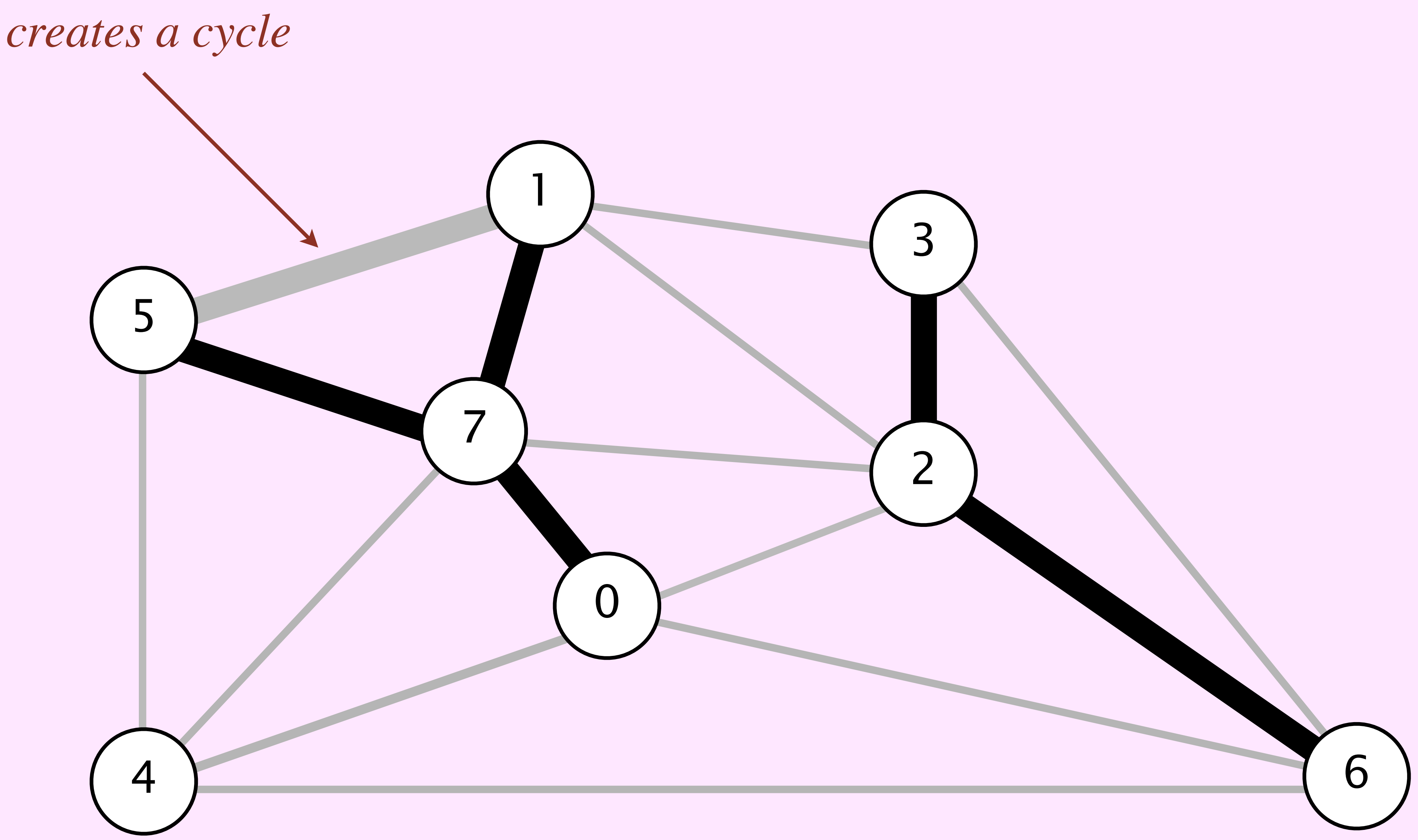


	0-7	0.16
	2-3	0.17
	1-7	0.19
	2-6	0.26
	3-6	0.47
in MST →	5-7	0.49
	1-5	0.61
	4-6	0.62
	2-7	0.65
	4-5	0.71
	1-2	0.74
	4-7	0.81
	0-4	0.84
	0-2	0.89
	1-3	0.92
	0-6	0.94

Karger's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to T unless doing so would create a cycle.
- Stop if T contains $V - 2$ edges.



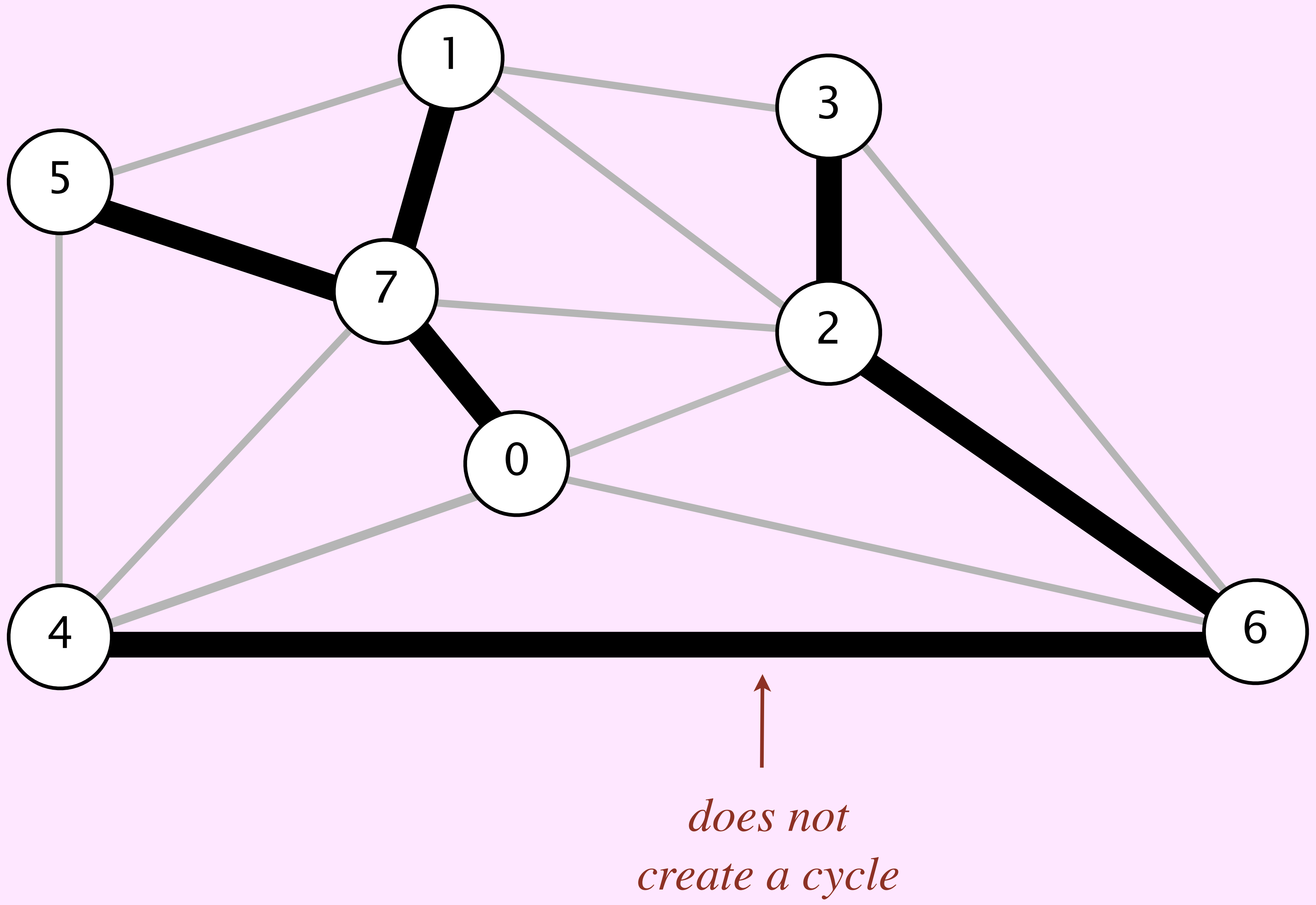
0-7	0.16
2-3	0.17
1-7	0.19
2-6	0.26
3-6	0.47
5-7	0.49
1-5	0.61
4-6	0.62
2-7	0.65
4-5	0.71
1-2	0.74
4-7	0.81
0-4	0.84
0-2	0.89
1-3	0.92
0-6	0.94

not in MST → 1-5 0.61

Karger's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to T unless doing so would create a cycle.
- Stop if T contains $V-2$ edges.



in MST → 4-6 0.62

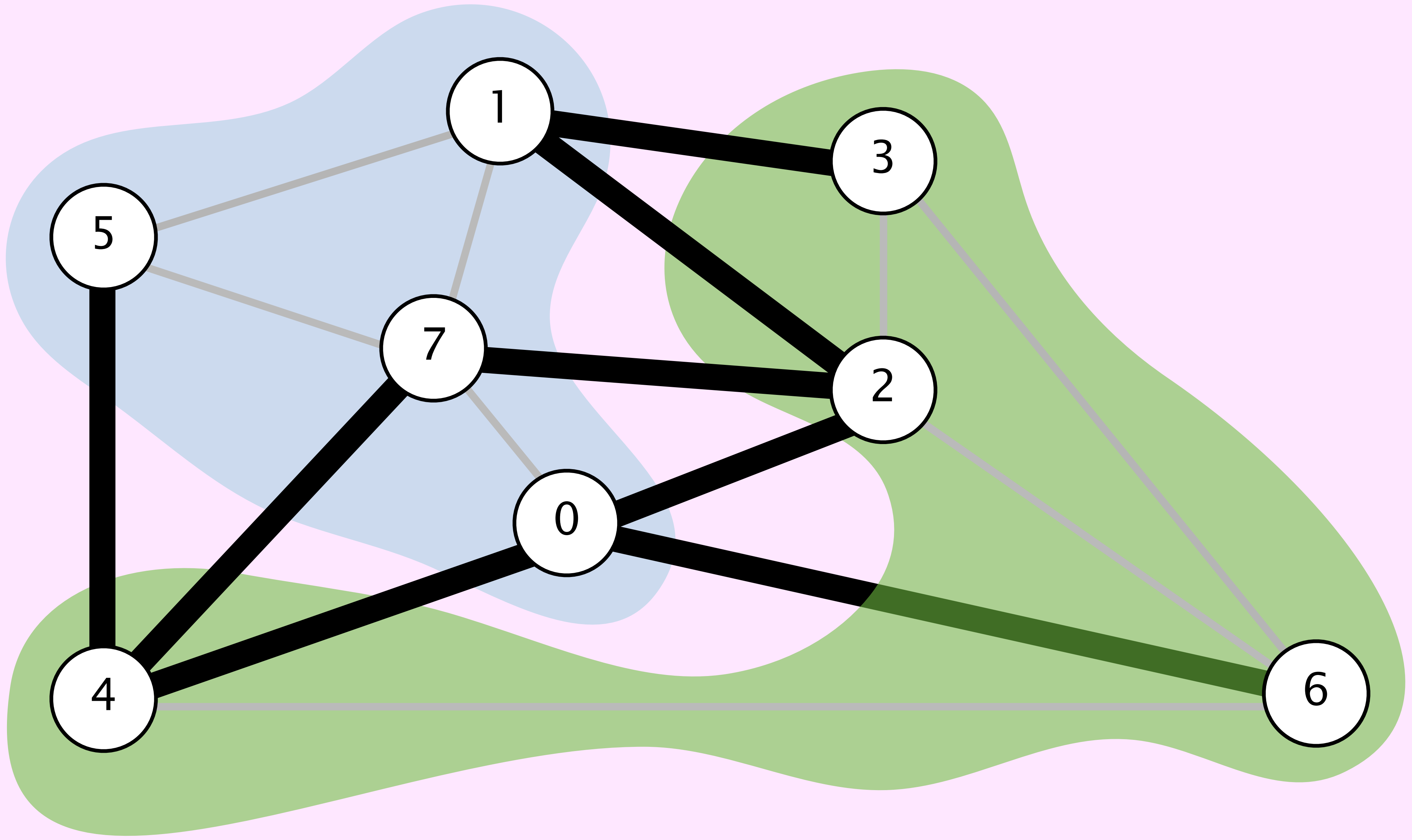
0-7	0.16
2-3	0.17
1-7	0.19
2-6	0.26
3-6	0.47
5-7	0.49
1-5	0.61
4-6	0.62
2-7	0.65
4-5	0.71
1-2	0.74
4-7	0.81
0-4	0.84
0-2	0.89
1-3	0.92
0-6	0.94

Karger's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to T unless doing so would create a cycle.
- Stop if T contains $V-2$ edges.

0-7	0.16
2-3	0.17
1-7	0.19
2-6	0.26
3-6	0.47
5-7	0.49
1-5	0.61
4-6	0.62
2-7	0.65
4-5	0.71
1-2	0.74
4-7	0.81
0-4	0.84
0-2	0.89
1-3	0.92
0-6	0.94



a cut: $\{0,1,5,7\}$ **and** $\{2,3,4,6\}$

Global minimum cut problem - Randomized Attempt 2 - Karger's algorithm

How good is it on the previous hard case?

V / 2 vertices on each side

The algorithm only succeeds if the middle edge is **not** picked by the Kruskal's algorithm step

If the middle edge has the largest weight, then it won't be picked → happens with probability $\dfrac{1}{E} \sim \dfrac{4}{V^2}$

Failure probability. Is $1 - \dfrac{1}{O(V^2)}$ for this graph, which is much better!

If we want $99\,\%$ success probability we need to repeat $O(V^2)$ times 🙂

(optional) use error reduction and exponential inequality

$$\left(1 - \frac{1}{x}\right)^{kx} \leq e^{-k}$$

$$\Downarrow$$

$$\left(1 - \frac{1}{V^2}\right)^{5V^2} \leq e^{-5} \approx 0.67\,\%$$

Global minimum cut problem - Randomized Attempt 2 - Karger's algorithm

What about for a general graph?

(optional) we have to observe that probability the ith edge added by Kruskal is:

$$\left(1 - \frac{2}{V - i + 1}\right)$$

and then we multiply these for $1 \leq i \leq V - 2$

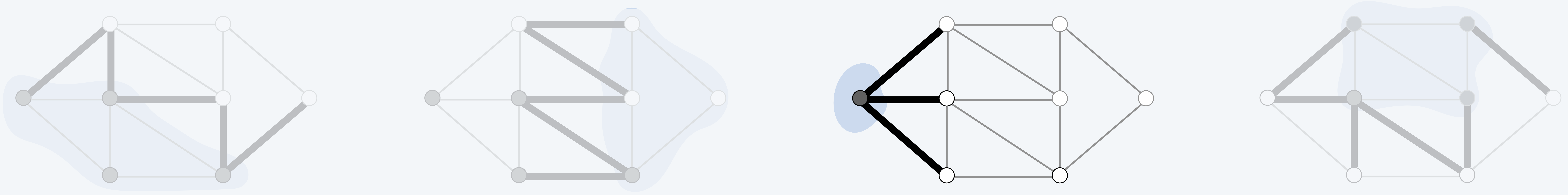
Failure probability. Surprisingly, it's still $1 - \frac{1}{O(V^2)}$!

So we can repeat $O(V^2)$ times the $\Theta(E \log E)$ Kruskal iteration and get a $\Theta(V^2 E \log E)$ time algorithm

Remark 1. Finds global mincut in $\Theta(V^2 E \log E)$ time — better than the Ford-Fulkerson based algorithm!

Remark 2. With clever idea, improved to $\Theta(E \log^3 V)$ time (still randomized)

Remark 3. With (really really) clever idea, improved to $\Theta(E \log^3 V)$ time deterministic



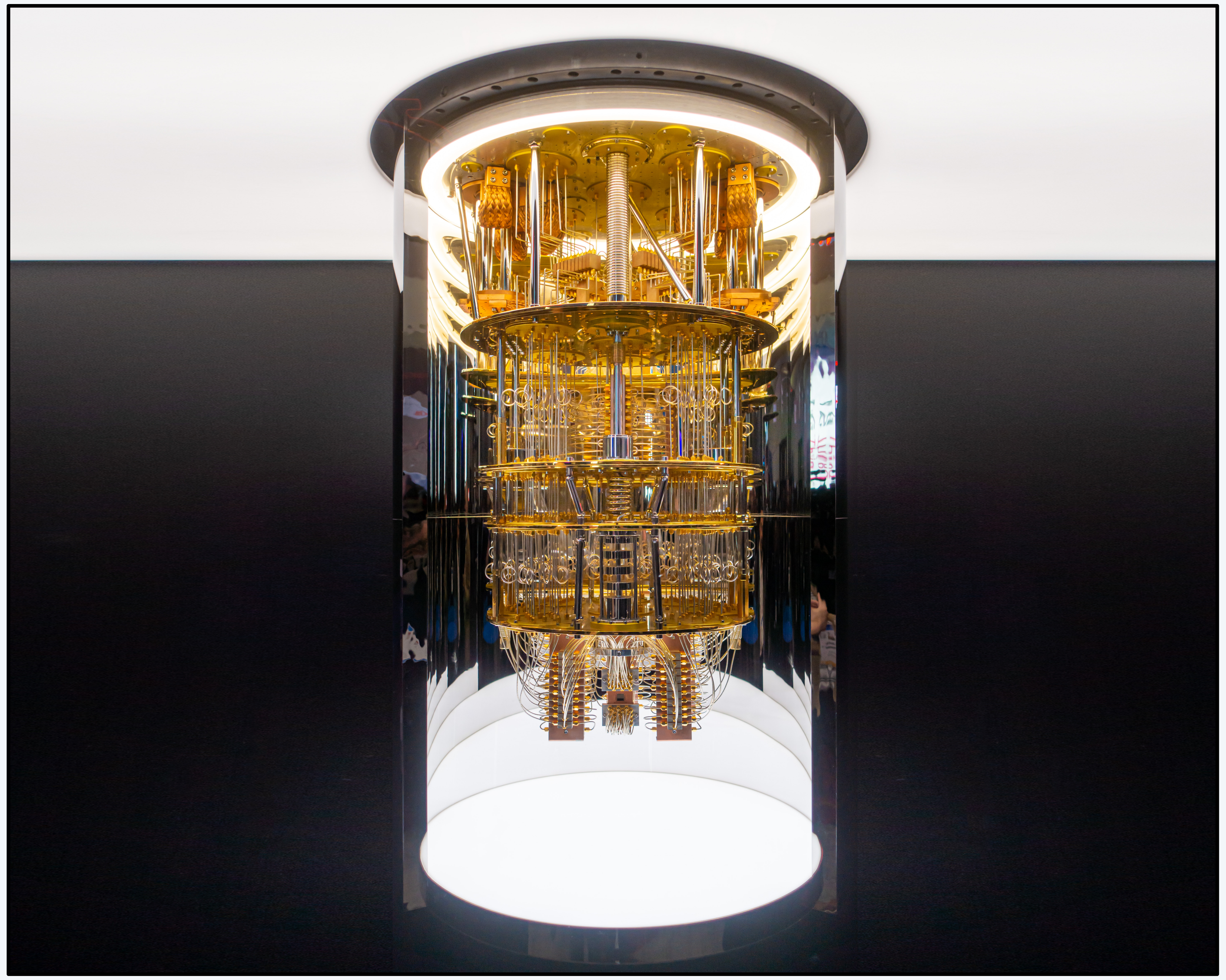
https://algs4.cs.princeton.edu

Randomness

- ‣ *randomness and algorithms*
- ‣ *treasure hunt problem*
- ‣ *duplicates finding*
- ‣ *Karger's algorithm*
- ‣ *more applications*

Beyond this course

- Approximation algorithms [intractability: stay tuned!]
- Machine learning [randomized MW]
- Optimization [stochastic gradient descent]
- Cryptography [average-case hardness]
- Complexity theory [derandomization]
- Quantum computation [Shor's factoring algorithm]
- Networking [load balancing]
- Graphics [procedural generation]
- Mathematics [probabilistic method]
- Health sciences [randomized control trials]

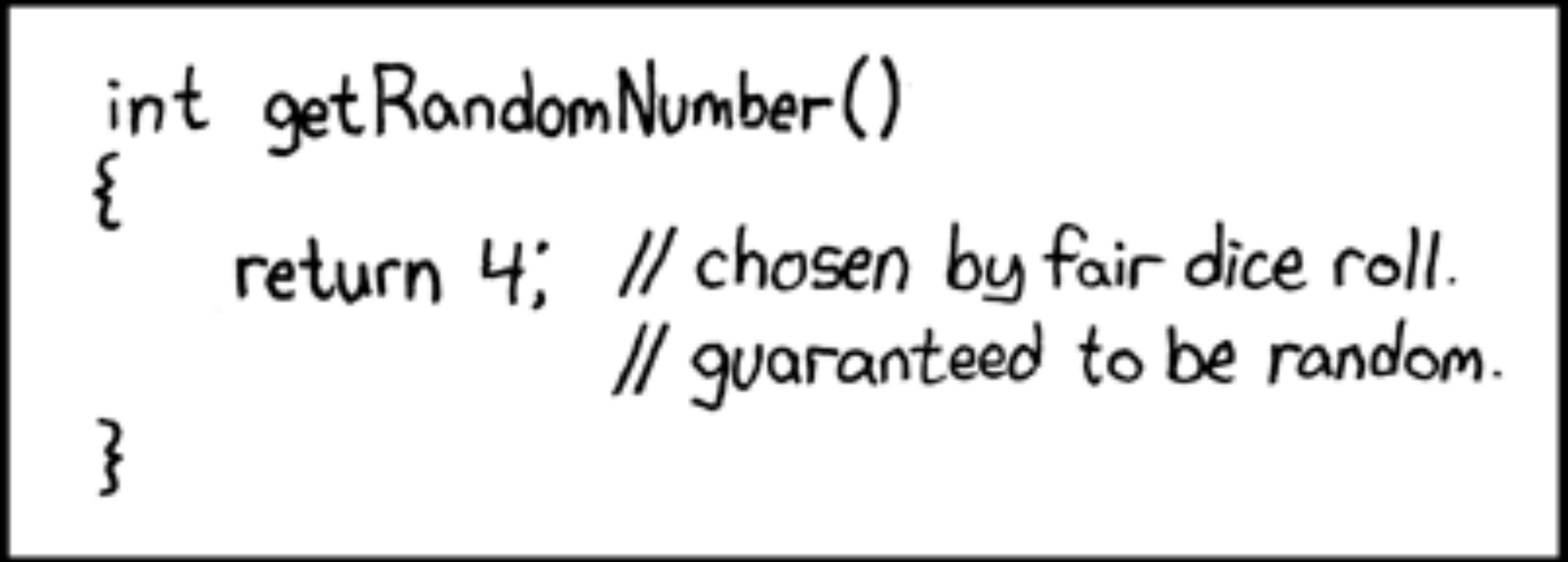


IBM Quantum System One

ORF 309. Probability and Stochastic Systems

COS 330. Great Ideas in Theoretical Computer Science

COS 433. Cryptography



https://xkcd.com/221/

Credits

image	source	license
Quarter	Adobe Stock	Education License
6-sided dice	Adobe Stock	Education License
20-sided die	Adobe Stock	Education License
Lava lamps	Fast Company	
Coin Toss	clipground.com	CC BY 4.0
IDQ Quantum Key Factory	idquantique.com	
SG100	protego.bytehost16.com	
Las Vegas	Adobe Stock	Education License
Monte Carlo	Adobe Stock	Education License
Treasure chests	Adobe Stock	Education License
Random number generator	XKCD	CC BY-NC 2.5

