The critical random graph

L. Addario-Berry

Random Structures and Dynamics

Oxford

April 11-14, 2011

Parts are joint with Nicolas Broutin, Luc Devroye, Christina Goldschmidt, Svante Janson, Grégory Miermont

The critical random graph

L. Addario-Berry

Random Structures and Dynamics

Oxford

April 11-14, 2011

Parts are joint with Nicolas Broutin, Luc Devroye, Christina Goldschmidt, Svante Janson, Grégory Miermont

The critical random graph

L. Addario-Berry

Random Structures and Dynamics

Oxford

April 11-14, 2011

Parts are joint with Nicolas Broutin, Luc Devroye, Christina Goldschmidt, Svante Janson, Grégory Miermont

The critical random graph

L. Addario-Berry

Random Structures and Dynamics

Oxford

April 11-14, 2011

Parts are joint with Nicolas Broutin, Luc Devroye, Christina Goldschmidt, Svante Janson, Grégory Miermont

The critical random graph

L. Addario-Berry

Random Structures and Dynamics

Oxford

April 11-14, 2011

Parts are joint with Nicolas Broutin, Luc Devroye, Christina Goldschmidt, Svante Janson, Grégory Miermont

The critical random graph

L. Addario-Berry

Random Structures and Dynamics

Oxford

April 11-14, 2011

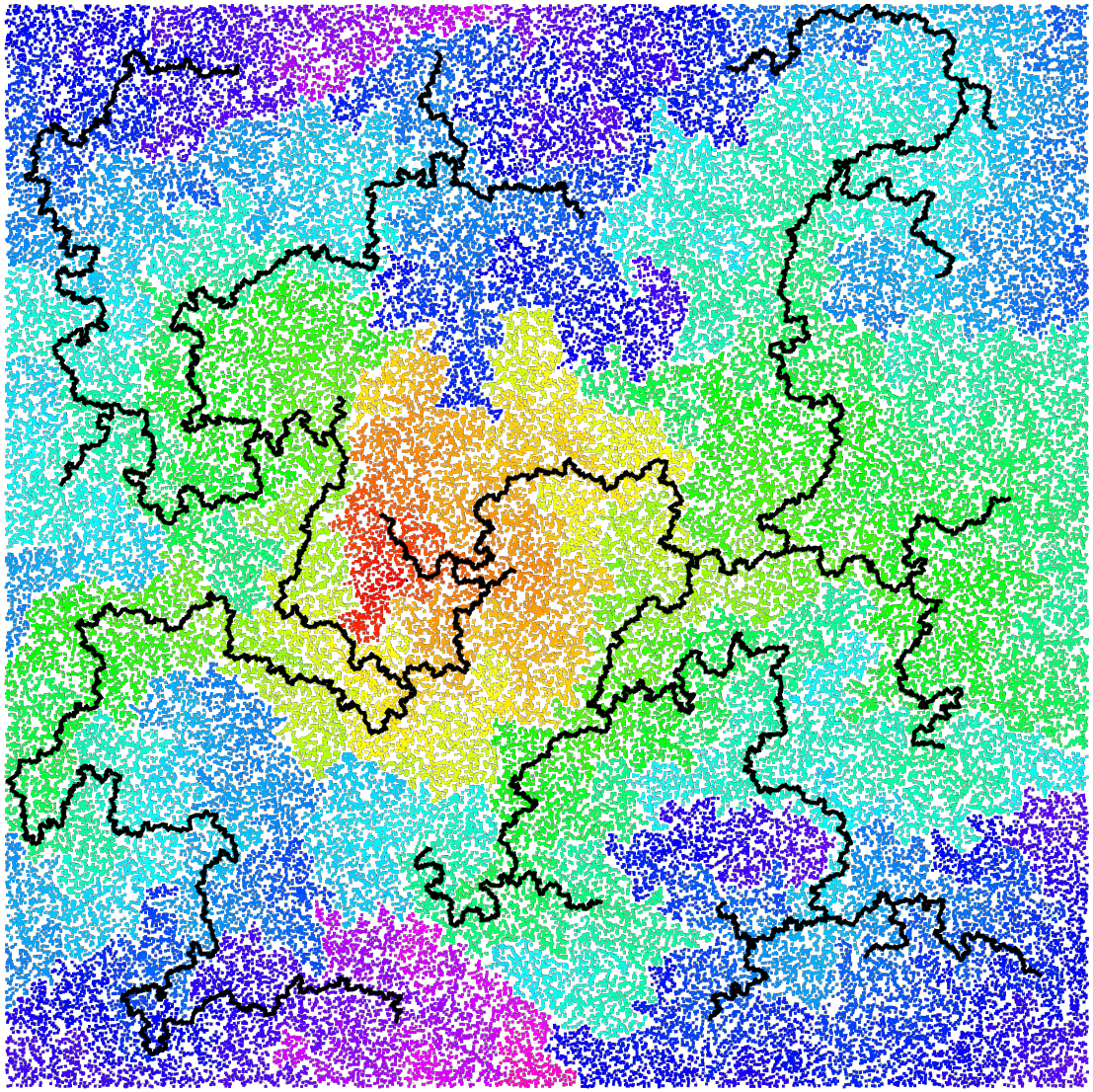
Parts are joint with Nicolas Broutin, Luc Devroye, Christina Goldschmidt, Svante Janson, Grégory Miermont

Motivation

Understand the following picture.

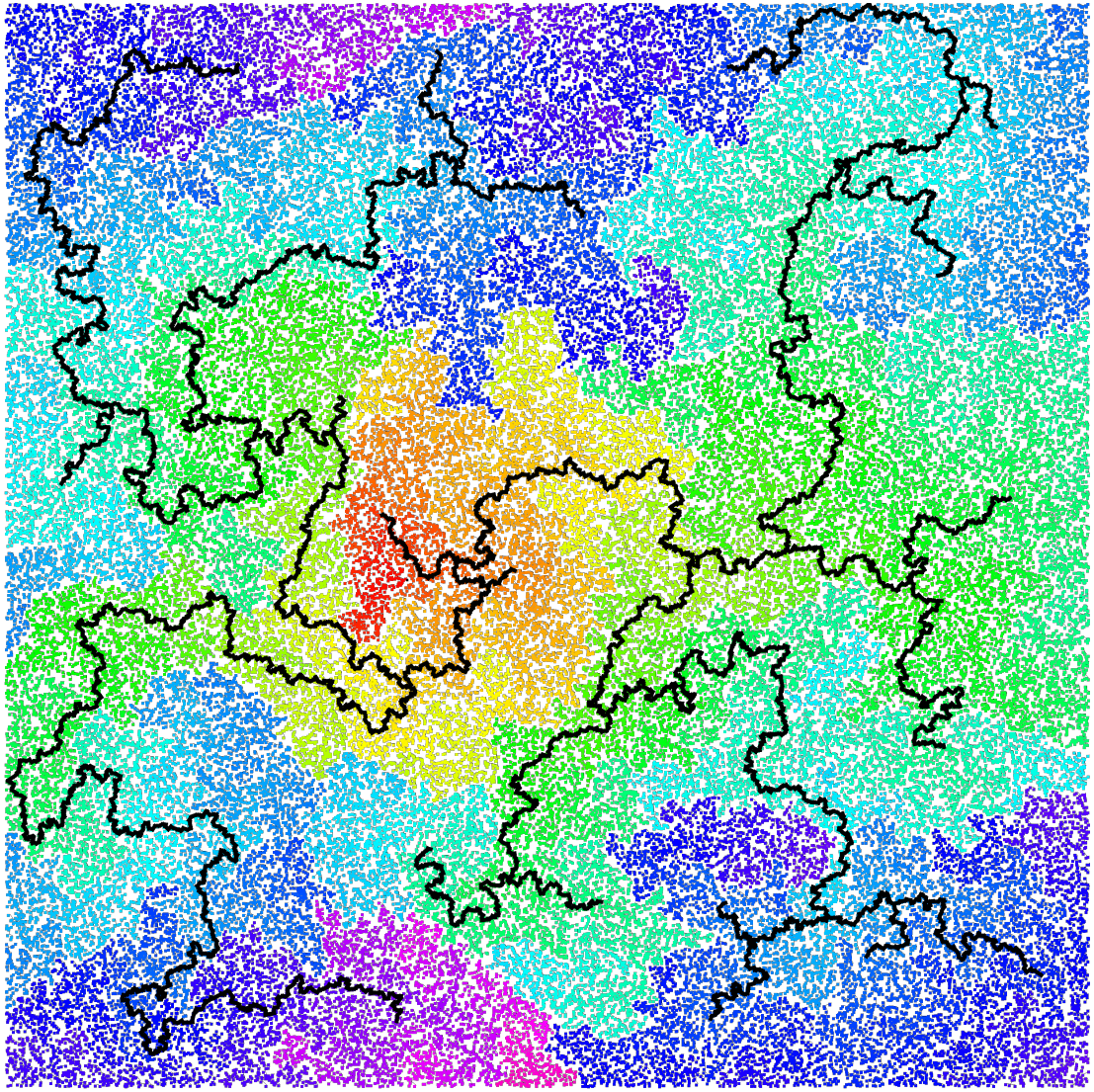
Motivation

Understand the following picture.



Motivation

Understand the following picture. (The (Euclidean) Minimum Spanning Tree of 100,000 uniformly random points.)



Motivation: minimum-weight spanning trees

Caveat: this is what motivates (originally motivated) *me*.

Motivation: minimum-weight spanning trees

Caveat: this is what motivates (originally motivated) *me*.

- My background: combinatorics and theoretical computer science.

Motivation: minimum-weight spanning trees

Caveat: this is what motivates (originally motivated) *me*.

- My background: combinatorics and theoretical computer science. Analysis of algorithms.

Motivation: minimum-weight spanning trees

Caveat: this is what motivates (originally motivated) *me*.

- My background: combinatorics and theoretical computer science. Analysis of algorithms.
- *Average-case analysis*: understand performance of an algorithm averaged over all possible inputs.

Motivation: minimum-weight spanning trees

Caveat: this is what motivates (originally motivated) *me*.

- My background: combinatorics and theoretical computer science. Analysis of algorithms.
- *Average-case analysis*: understand performance of an algorithm averaged over all possible inputs.
- Since 2004, I've been interested in average-case analysis of *minimum weight spanning tree algorithms*.

Motivation: minimum-weight spanning trees

Caveat: this is what motivates (originally motivated) *me*.

- My background: combinatorics and theoretical computer science. Analysis of algorithms.
- *Average-case analysis*: understand performance of an algorithm averaged over all possible inputs.
- Since 2004, I've been interested in average-case analysis of *minimum weight spanning tree algorithms*.
- Minimum spanning tree is *algorithmically simple* but *probabilistically challenging*.

Minimum-weight spanning trees

- Given points $x_1, \ldots, x_n \in \mathbb{R}^d$.

Minimum-weight spanning trees

- Given points $x_1, \ldots, x_n \in \mathbb{R}^d$.
- Minimum-weight spanning tree:
  a set of $n-1$ edges $e_1, \ldots, e_{n-1}$
  (say $e_i = x_{i,1}x_{i,2}$) that connects
  all the points

Minimum-weight spanning trees

- Given points $x_1, \ldots, x_n \in \mathbb{R}^d$.
- Minimum-weight spanning tree:
  a set of $n-1$ edges $e_1, \ldots, e_{n-1}$
  (say $e_i = x_{i,1}x_{i,2}$) that connects
  all the points, and minimizes

$$\sum_{i=1}^{n-1} \|e_i\|_2$$

Minimum-weight spanning trees

- Given points $x_1, \ldots, x_n \in \mathbb{R}^d$.
- Minimum-weight spanning tree: a set of $n-1$ edges $e_1, \ldots, e_{n-1}$ (say $e_i = x_{i,1}x_{i,2}$) that connects all the points, and minimizes

$$\sum_{i=1}^{n-1} \|e_i\|_2 = \sum_{i=1}^{n-1} |x_{i,2} - x_{i,1}|.$$

Minimum-weight spanning trees

- Given points $x_1, \ldots, x_n \in \mathbb{R}^d$.
- Minimum-weight spanning tree: a set of $n-1$ edges $e_1, \ldots, e_{n-1}$ (say $e_i = x_{i,1}x_{i,2}$) that connects all the points, and minimizes

$$\sum_{i=1}^{n-1} \|e_i\|_2 = \sum_{i=1}^{n-1} |x_{i,2} - x_{i,1}|.$$

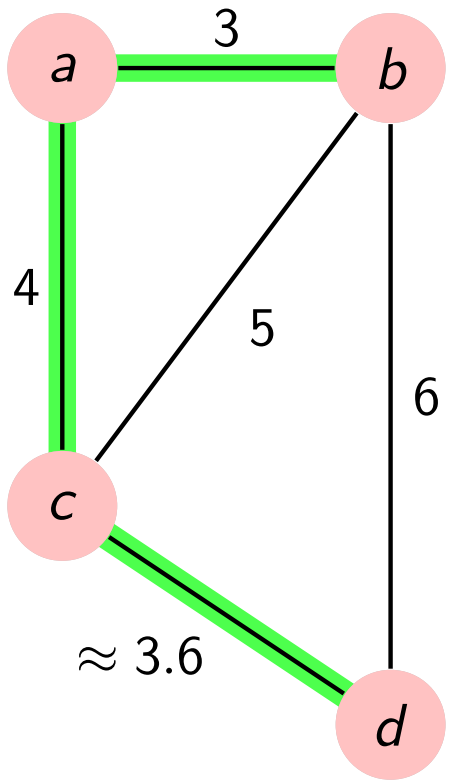
- Definition makes sense for any finite metric space (and for many infinite ones).

Minimum-weight spanning trees

- Given points $x_1, \ldots, x_n \in \mathbb{R}^d$.
- Minimum-weight spanning tree: a set of $n-1$ edges $e_1, \ldots, e_{n-1}$ (say $e_i = x_{i,1}x_{i,2}$) that connects all the points, and minimizes

$$\sum_{i=1}^{n-1} \|e_i\|_2 = \sum_{i=1}^{n-1} |x_{i,2} - x_{i,1}|.$$

- Definition makes sense for any finite metric space (and for many infinite ones).

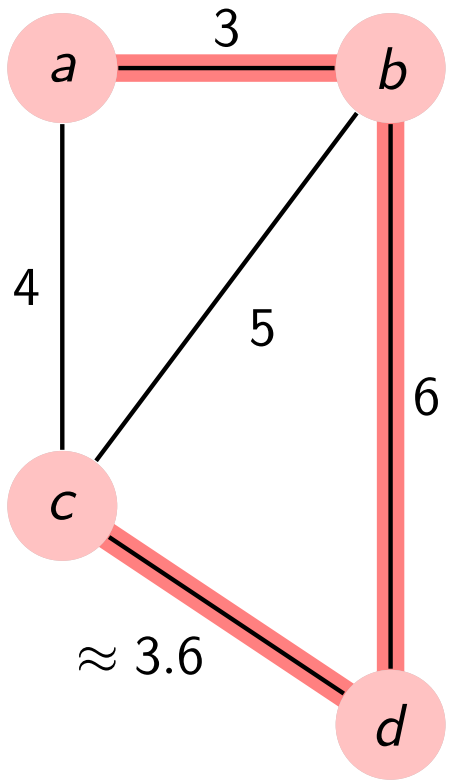


Minimum-weight spanning trees

- Given points $x_1, \ldots, x_n \in \mathbb{R}^d$.
- Minimum-weight spanning tree: a set of $n-1$ edges $e_1, \ldots, e_{n-1}$ (say $e_i = x_{i,1}x_{i,2}$) that connects all the points, and minimizes

$$\sum_{i=1}^{n-1} \|e_i\|_2 = \sum_{i=1}^{n-1} |x_{i,2} - x_{i,1}|.$$

- Definition makes sense for any finite metric space (and for many infinite ones).

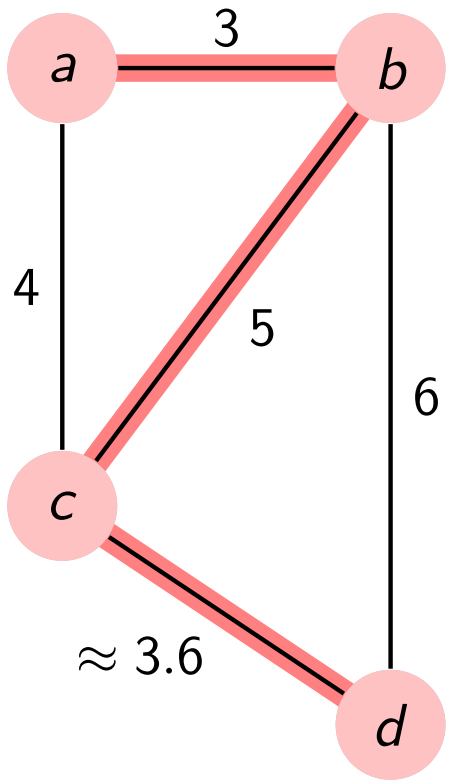


Minimum-weight spanning trees

- Given points $x_1, \ldots, x_n \in \mathbb{R}^d$.
- Minimum-weight spanning tree: a set of $n-1$ edges $e_1, \ldots, e_{n-1}$ (say $e_i = x_{i,1}x_{i,2}$) that connects all the points, and minimizes

$$\sum_{i=1}^{n-1} \|e_i\|_2 = \sum_{i=1}^{n-1} |x_{i,2} - x_{i,1}|.$$

- Definition makes sense for any finite metric space (and for many infinite ones).



Intended learning outcomes

Goal for today:

Intended learning outcomes

Goal for today: Explain the connection between minimum spanning trees and critical percolation.

Minimum-weight spanning trees

A small aside!

- Can always represent a finite metric space (V, d) as a graph $G = (V, E)$ with edge weights $\{w_e : e \in E\}$.

Minimum-weight spanning trees

A small aside!

- Can always represent a finite metric space (V, d) as a graph $G = (V, E)$ with edge weights $\{w_e : e \in E\}$.
  (For edge $e = \{u, v\}$ take $w_e = d(u, v)$.)

Minimum-weight spanning trees

A small aside!

- Can always represent a finite metric space (V, d) as a graph $G = (V, E)$ with edge weights $\{w_e : e \in E\}$.
  (For edge $e = \{u, v\}$ take $w_e = d(u, v)$.)
- Conversely, given connected $G = (V, E)$ and non-negative edge weights $\{w_e : e \in E\}$,

Minimum-weight spanning trees

A small aside!

- Can always represent a finite metric space (V, d) as a graph $G = (V, E)$ with edge weights $\{w_e : e \in E\}$.
  (For edge $e = \{u, v\}$ take $w_e = d(u, v)$.)
- Conversely, given connected $G = (V, E)$ and non-negative edge weights $\{w_e : e \in E\}$, obtain a metric space by taking the *metric completion*.

Minimum-weight spanning trees

A small aside!

- Can always represent a finite metric space (V, d) as a graph $G = (V, E)$ with edge weights $\{w_e : e \in E\}$.
  (For edge $e = \{u, v\}$ take $w_e = d(u, v)$.)
- Conversely, given connected $G = (V, E)$ and non-negative edge weights $\{w_e : e \in E\}$, obtain a metric space by taking the *metric completion*. This just means setting $d(u, v) = \min \sum_{e \in P} w_e$.

Minimum-weight spanning trees

A small aside!

- Can always represent a finite metric space (V, d) as a graph $G = (V, E)$ with edge weights $\{w_e : e \in E\}$.
  (For edge $e = \{u, v\}$ take $w_e = d(u, v)$.)
- Conversely, given connected $G = (V, E)$ and non-negative edge weights $\{w_e : e \in E\}$, obtain a metric space by taking the *metric completion*. This just means setting $d(u, v) = \min \sum_{e \in P} w_e$.
  Here minimum is over all paths (sequences of edges) leading from u to v.

Minimum-weight spanning trees

A small aside!

- Can always represent a finite metric space (V, d) as a graph $G = (V, E)$ with edge weights $\{w_e : e \in E\}$.
  (For edge $e = \{u, v\}$ take $w_e = d(u, v)$.)
- Conversely, given connected $G = (V, E)$ and non-negative edge weights $\{w_e : e \in E\}$, obtain a metric space by taking the *metric completion*. This just means setting $d(u, v) = \min \sum_{e \in P} w_e$.
  Here minimum is over all paths (sequences of edges) leading from u to v.
- From now on, we usually think about weighted graphs.

Minimum-weight spanning trees

A small aside!

- Can always represent a finite metric space (V, d) as a graph $G = (V, E)$ with edge weights $\{w_e : e \in E\}$.
  (For edge $e = \{u, v\}$ take $w_e = d(u, v)$.)
- Conversely, given connected $G = (V, E)$ and non-negative edge weights $\{w_e : e \in E\}$, obtain a metric space by taking the *metric completion*. This just means setting $d(u, v) = \min \sum_{e \in P} w_e$.
  Here minimum is over all paths (sequences of edges) leading from u to v.
- From now on, we usually think about weighted graphs.
- Also: always assume all edge weights distinct!

Minimum-weight spanning trees

Algorithmic considerations

How does one find the MST of a given (finite) graph $G = (V, E)$ with edge weights $\{w_e\}$?

Minimum-weight spanning trees

Algorithmic considerations

How does one find the MST of a given (finite) graph $G = (V, E)$ with edge weights $\{w_e\}$?

There are three natural algorithms.

Minimum-weight spanning trees

Algorithmic considerations

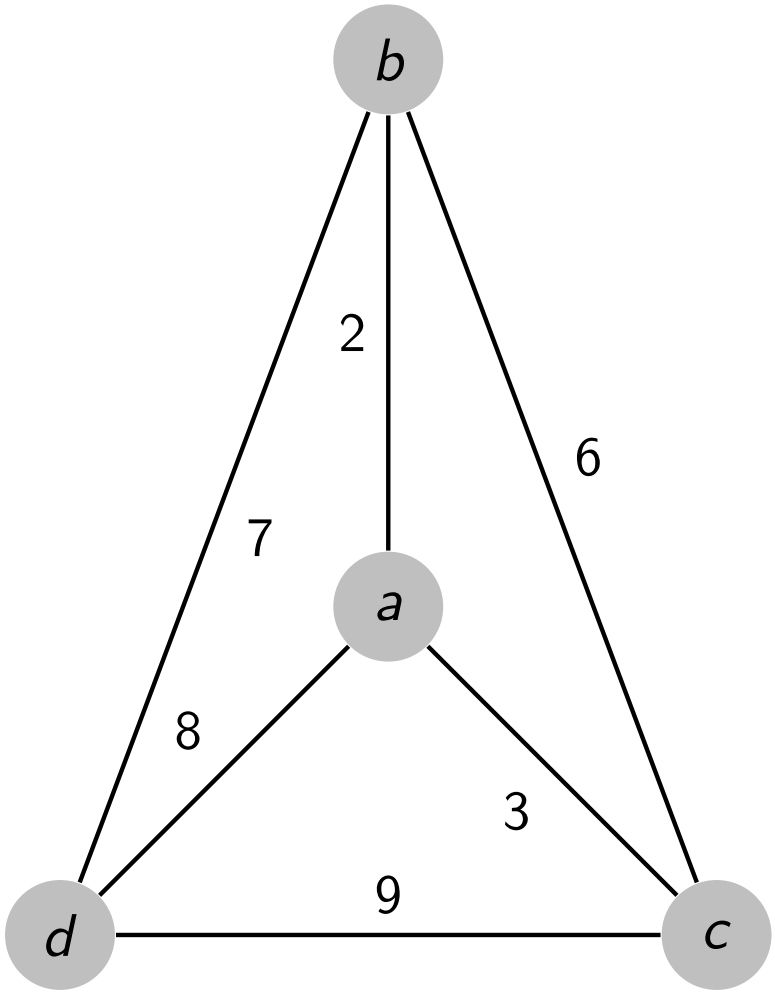
How does one find the MST of a given (finite) graph $G = (V, E)$ with edge weights $\{w_e\}$?

There are three natural algorithms.

Each turns out to expose a different probabilistic aspect of MSTs.

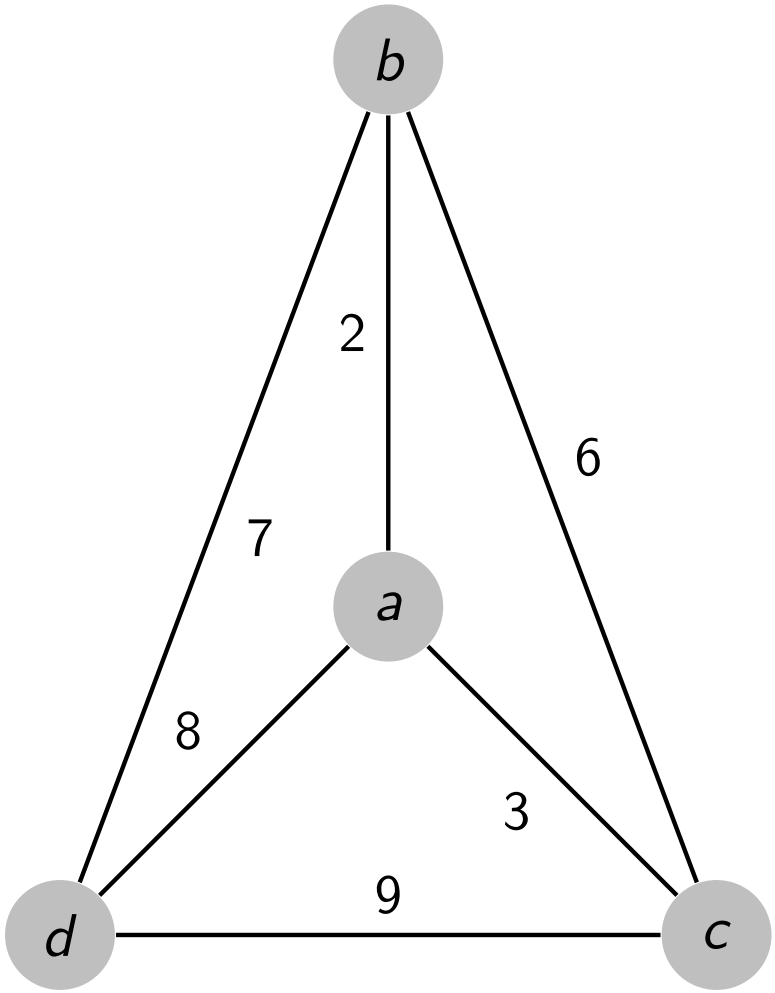
Kruskal's algorithm/Borůvka's algorithm/Percolation process

- Order edges in increasing order of weight as $e_1, \ldots, e_m$.



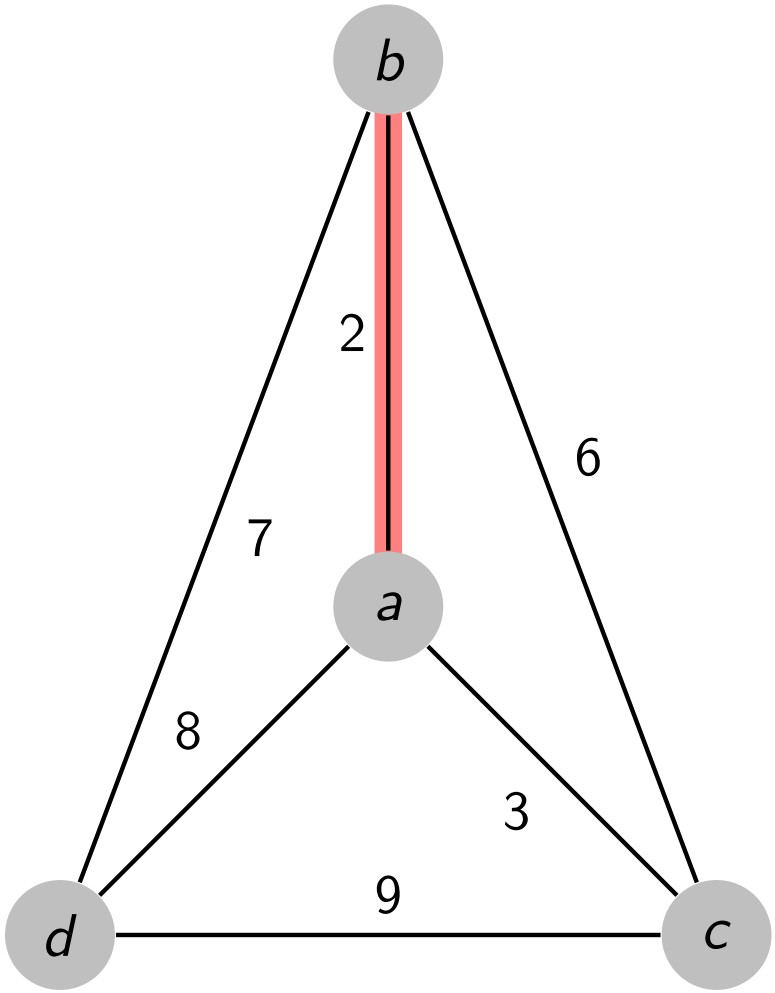
Kruskal's algorithm/Borůvka's algorithm/Percolation process

- Order edges in increasing order of weight as $e_1, \ldots, e_m$.
- For each i, add edge e_i unless doing so would create a cycle.



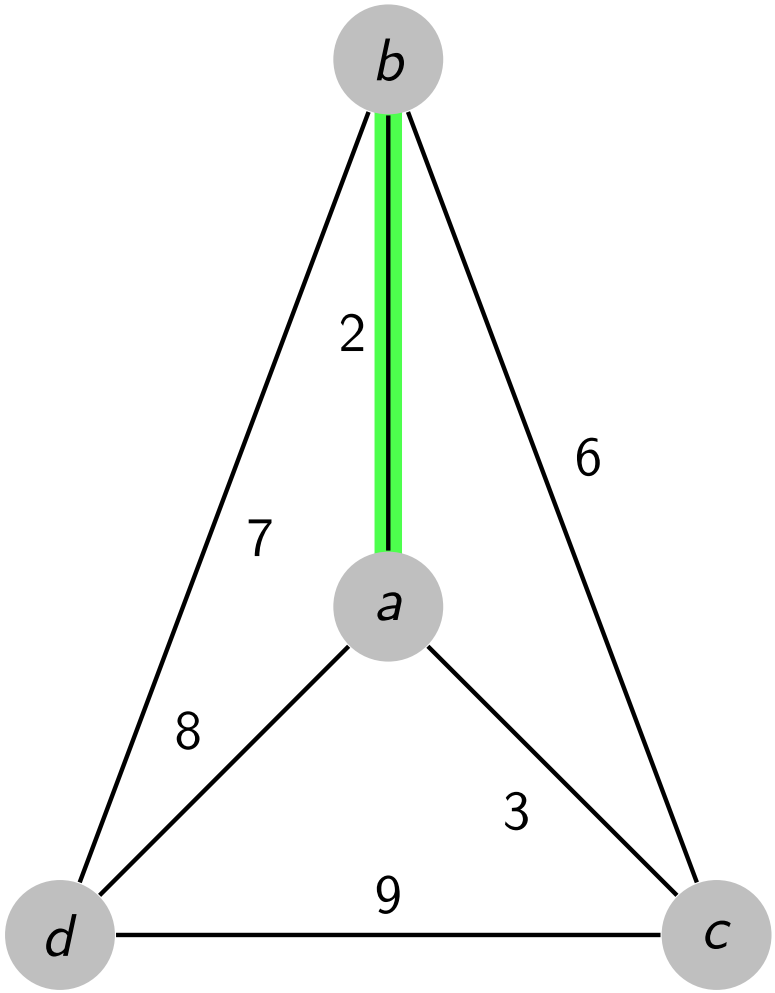
Kruskal's algorithm/Borůvka's algorithm/Percolation process

- Order edges in increasing order of weight as $e_1, \ldots, e_m$.
- For each i, add edge e_i unless doing so would create a cycle.



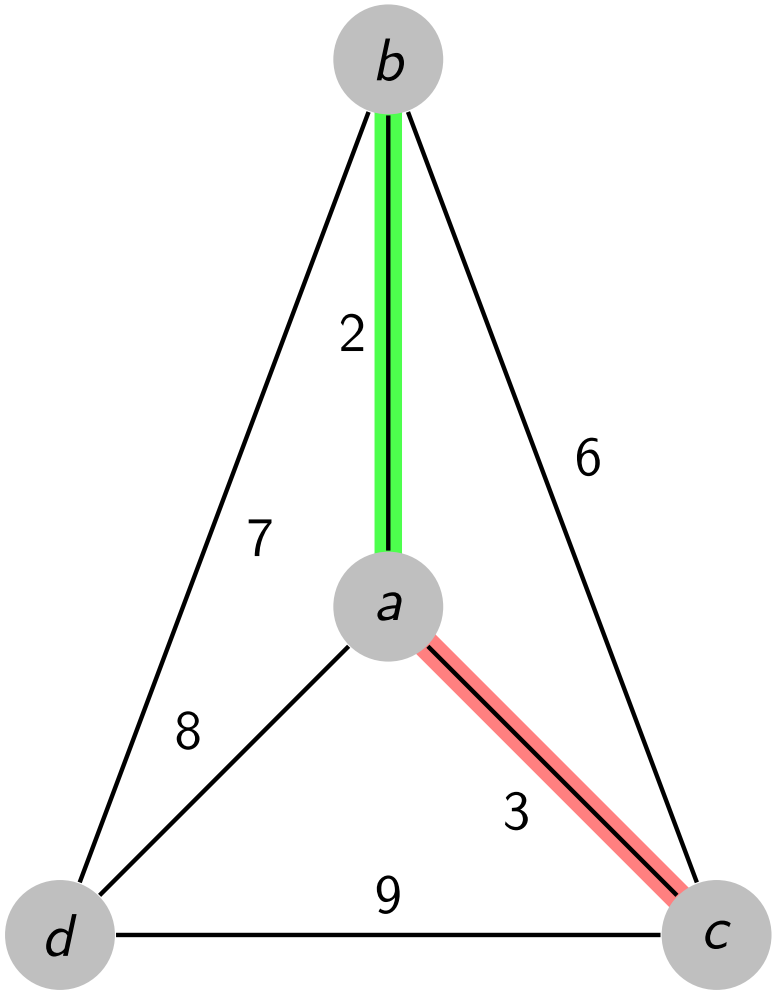
Kruskal's algorithm/Borůvka's algorithm/Percolation process

- Order edges in increasing order of weight as $e_1, \ldots, e_m$.
- For each i, add edge e_i unless doing so would create a cycle.



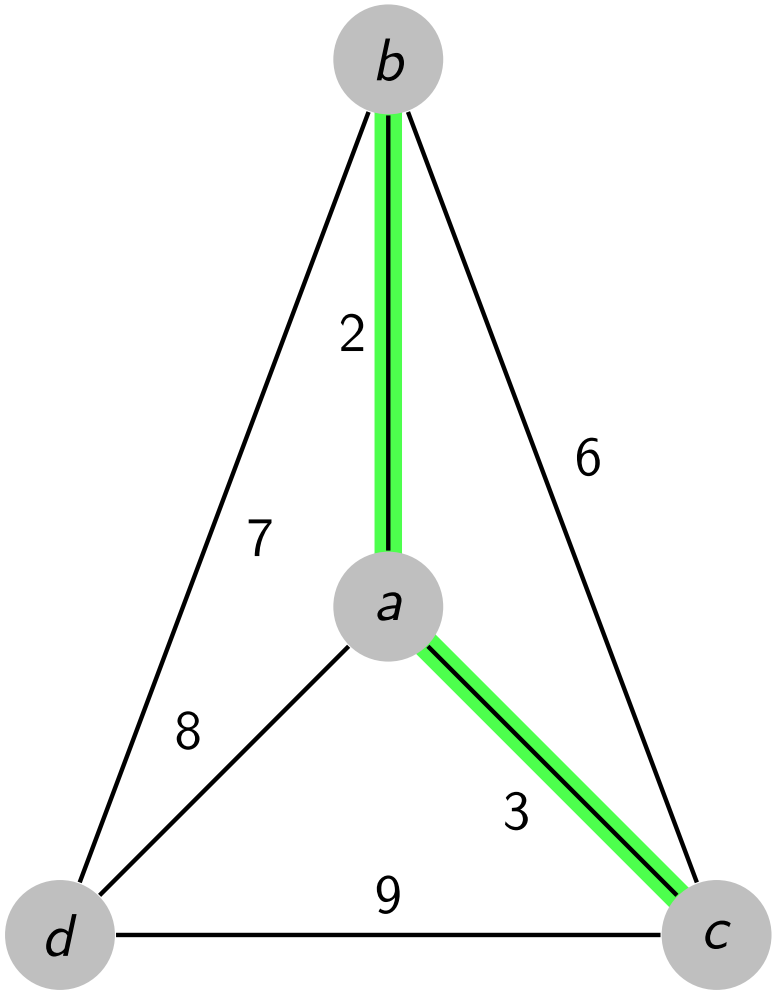
Kruskal's algorithm/Borůvka's algorithm/Percolation process

- Order edges in increasing order of weight as $e_1, \ldots, e_m$.
- For each i, add edge e_i unless doing so would create a cycle.



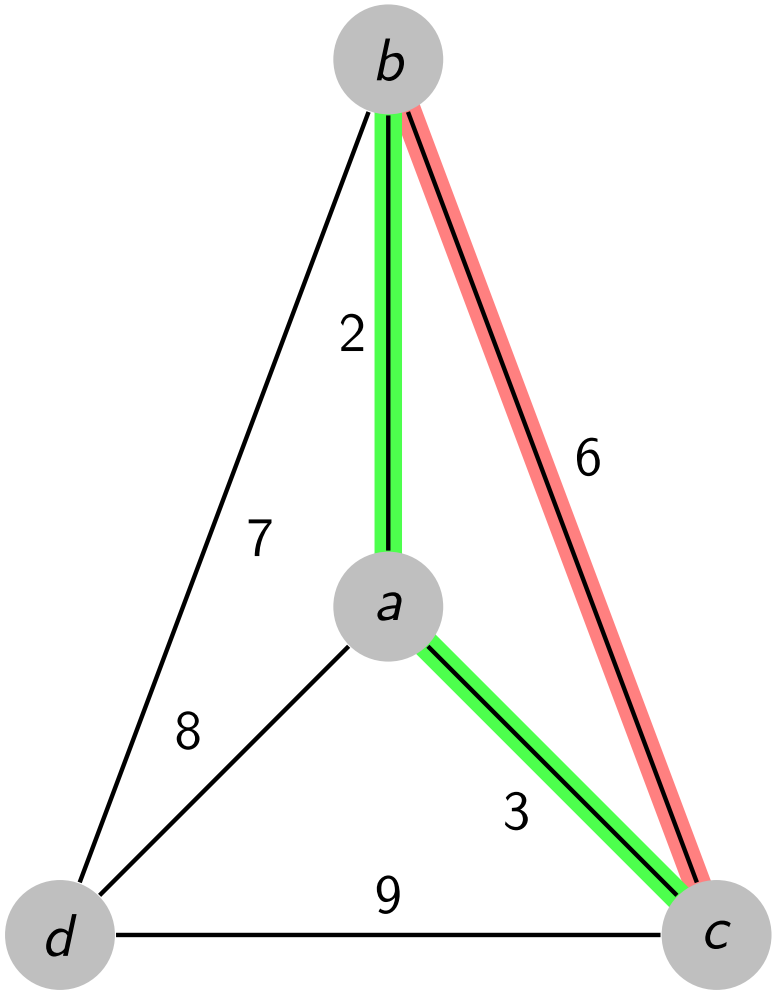
Kruskal's algorithm/Borůvka's algorithm/Percolation process

- Order edges in increasing order of weight as $e_1, \ldots, e_m$.
- For each i, add edge e_i unless doing so would create a cycle.



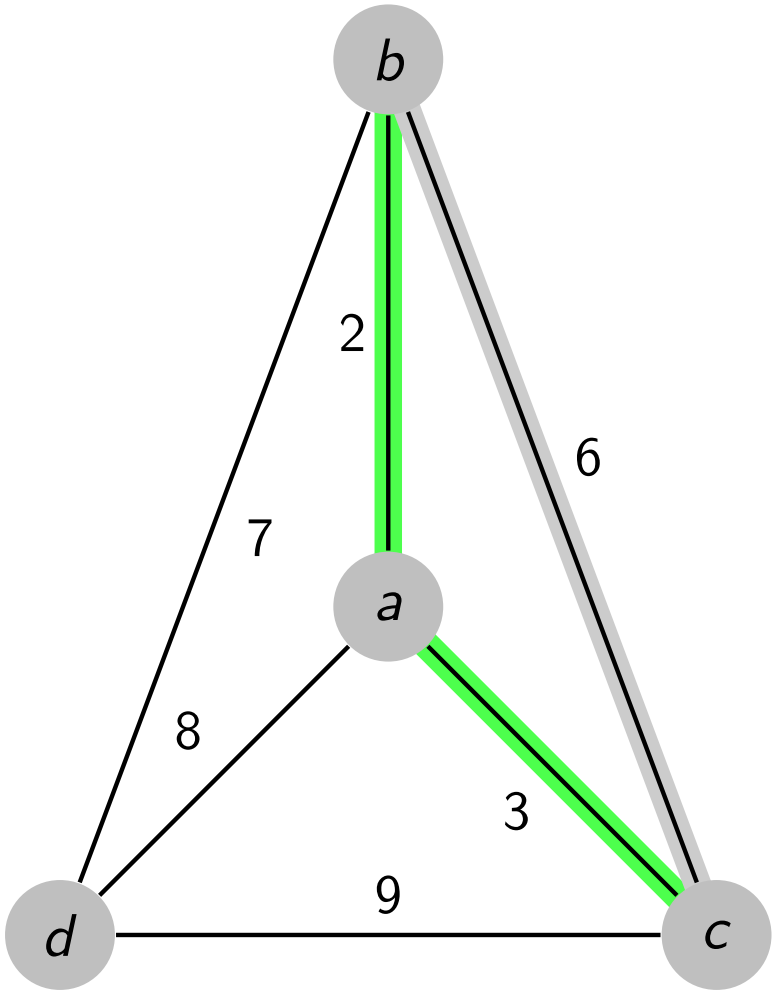
Kruskal's algorithm/Borůvka's algorithm/Percolation process

- Order edges in increasing order of weight as $e_1, \ldots, e_m$.
- For each i, add edge e_i unless doing so would create a cycle.



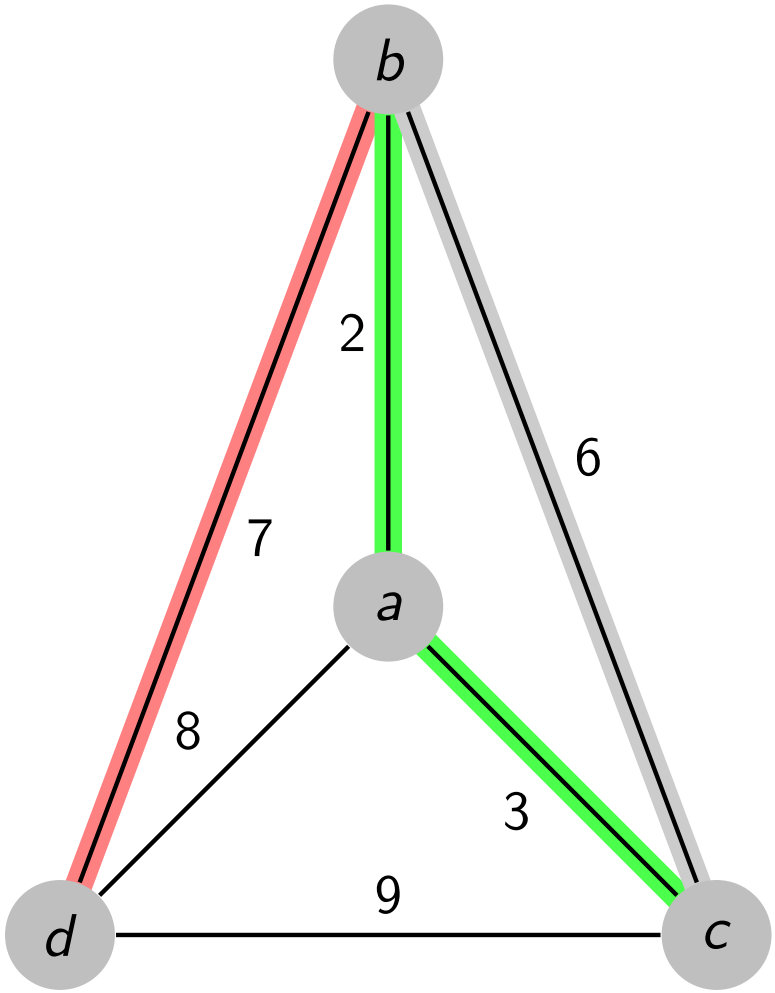
Kruskal's algorithm/Borůvka's algorithm/Percolation process

- Order edges in increasing order of weight as $e_1, \ldots, e_m$.
- For each i, add edge e_i unless doing so would create a cycle.



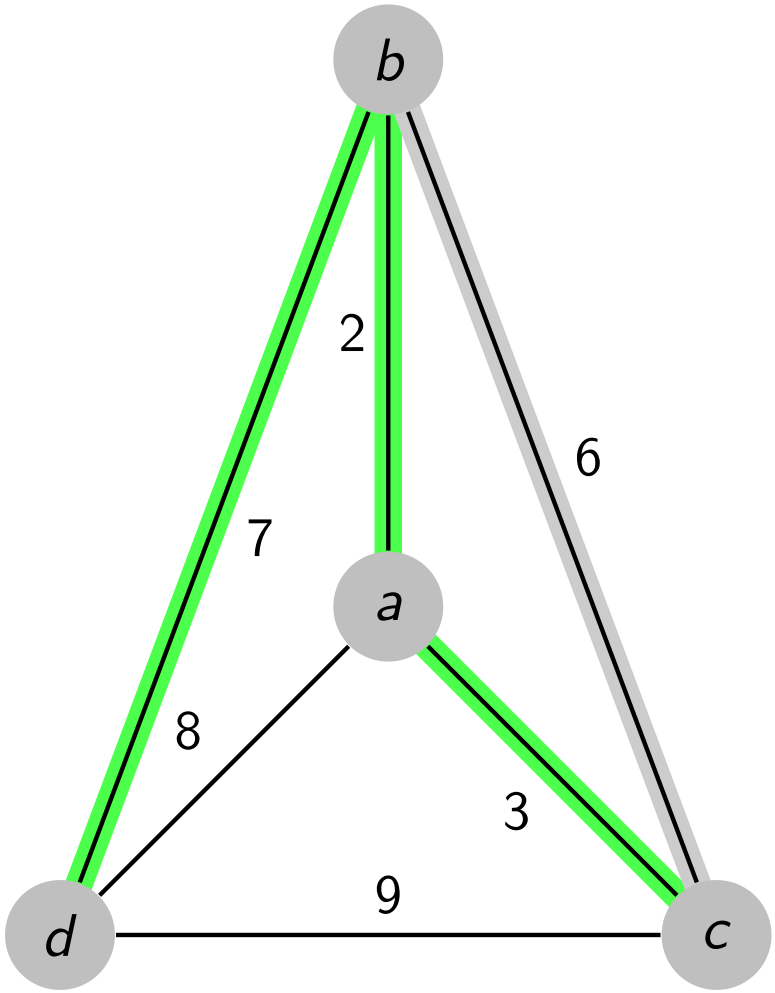
Kruskal's algorithm/Borůvka's algorithm/Percolation process

- Order edges in increasing order of weight as $e_1, \ldots, e_m$.
- For each i, add edge e_i unless doing so would create a cycle.



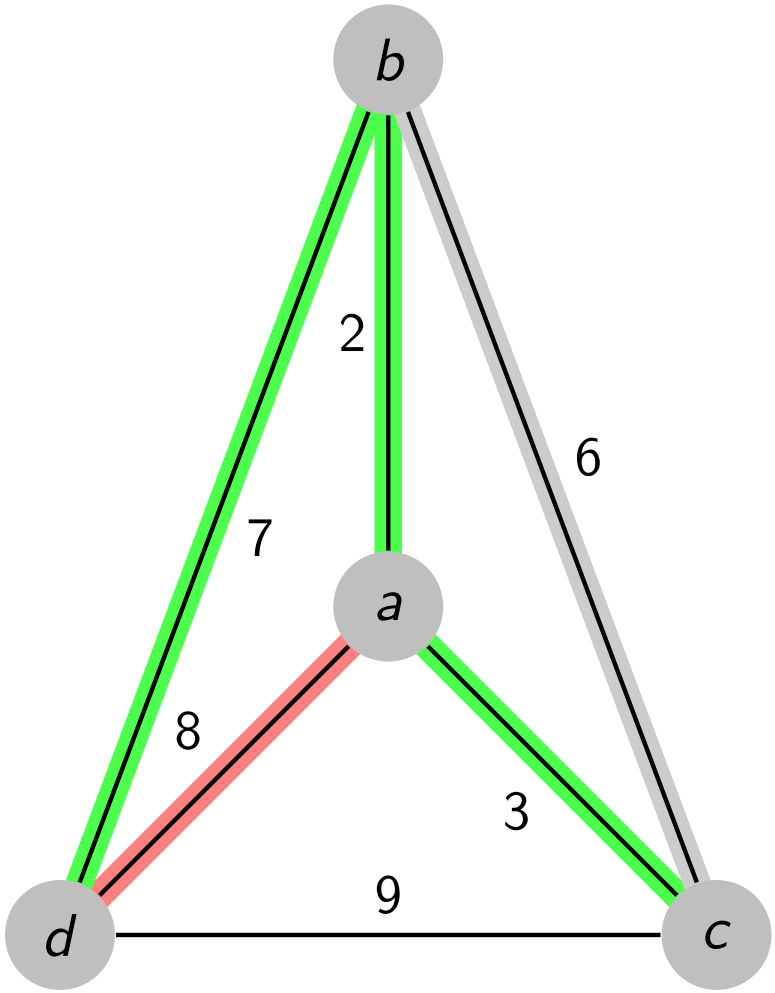
Kruskal's algorithm/Borůvka's algorithm/Percolation process

- Order edges in increasing order of weight as $e_1, \ldots, e_m$.
- For each i, add edge e_i unless doing so would create a cycle.



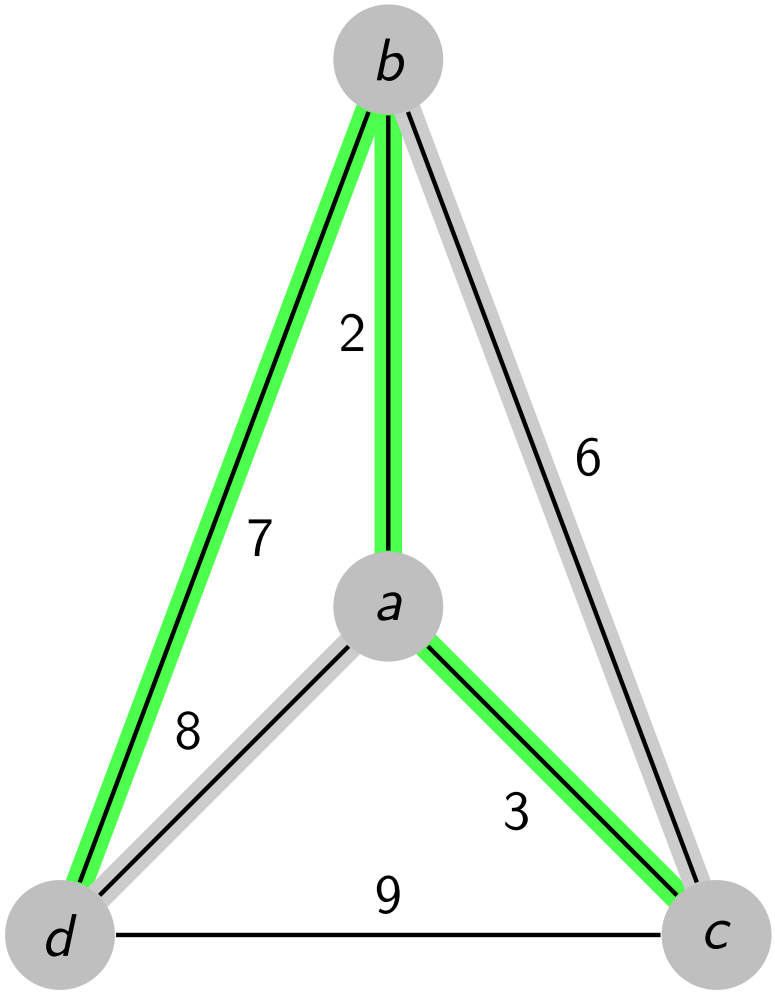
Kruskal's algorithm/Borůvka's algorithm/Percolation process

- Order edges in increasing order of weight as $e_1, \ldots, e_m$.
- For each i, add edge e_i unless doing so would create a cycle.



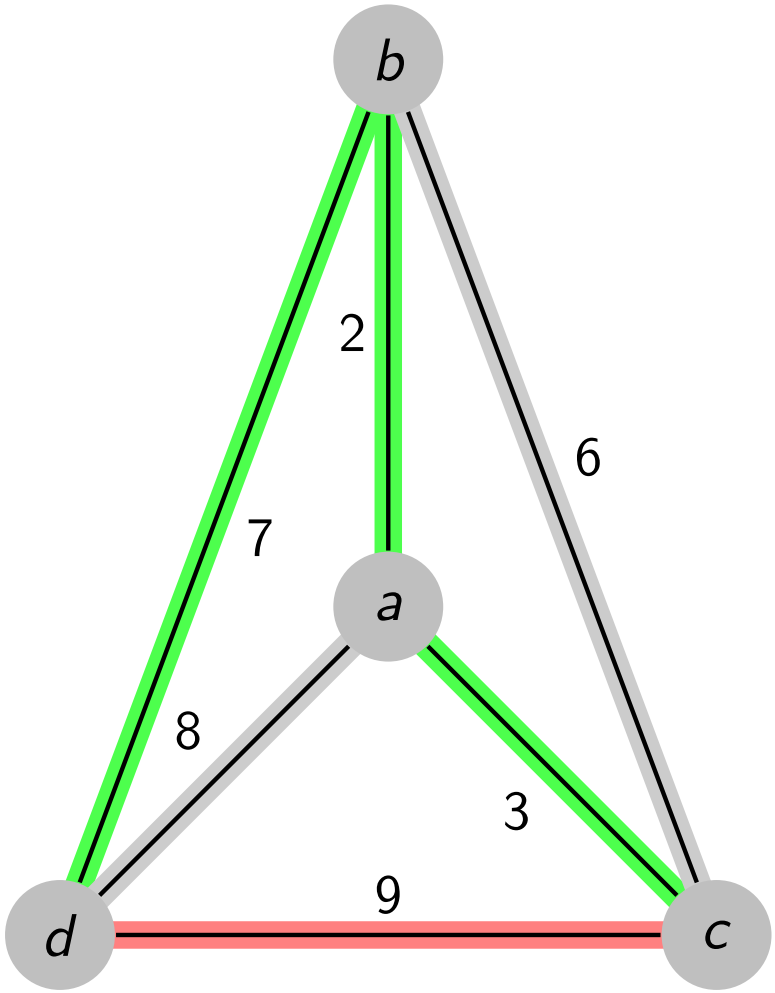
Kruskal's algorithm/Borůvka's algorithm/Percolation process

- Order edges in increasing order of weight as $e_1, \ldots, e_m$.
- For each i, add edge e_i unless doing so would create a cycle.



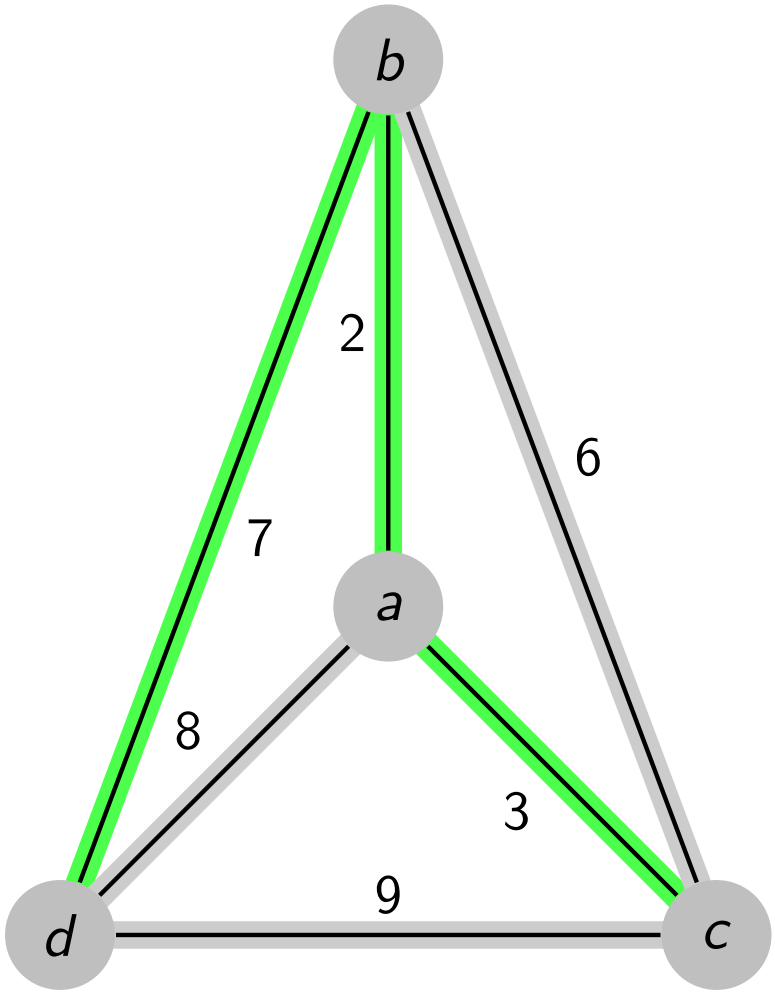
Kruskal's algorithm/Borůvka's algorithm/Percolation process

- Order edges in increasing order of weight as $e_1, \ldots, e_m$.
- For each i, add edge e_i unless doing so would create a cycle.



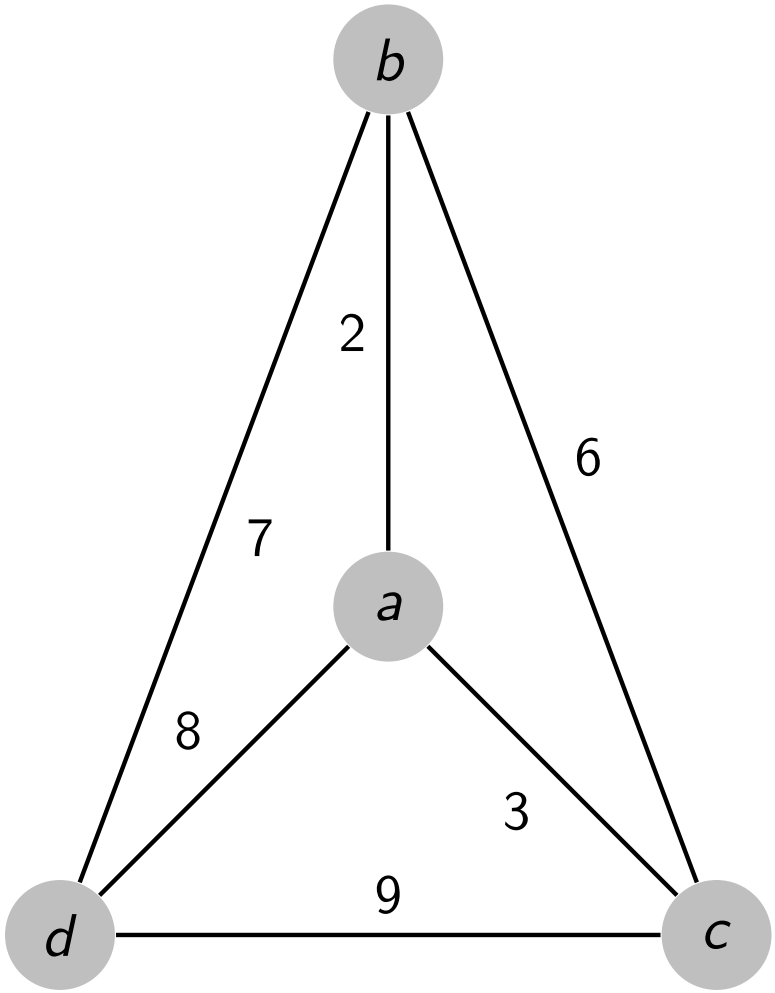
Kruskal's algorithm/Borůvka's algorithm/Percolation process

- Order edges in increasing order of weight as $e_1, \ldots, e_m$.
- For each i, add edge e_i unless doing so would create a cycle.



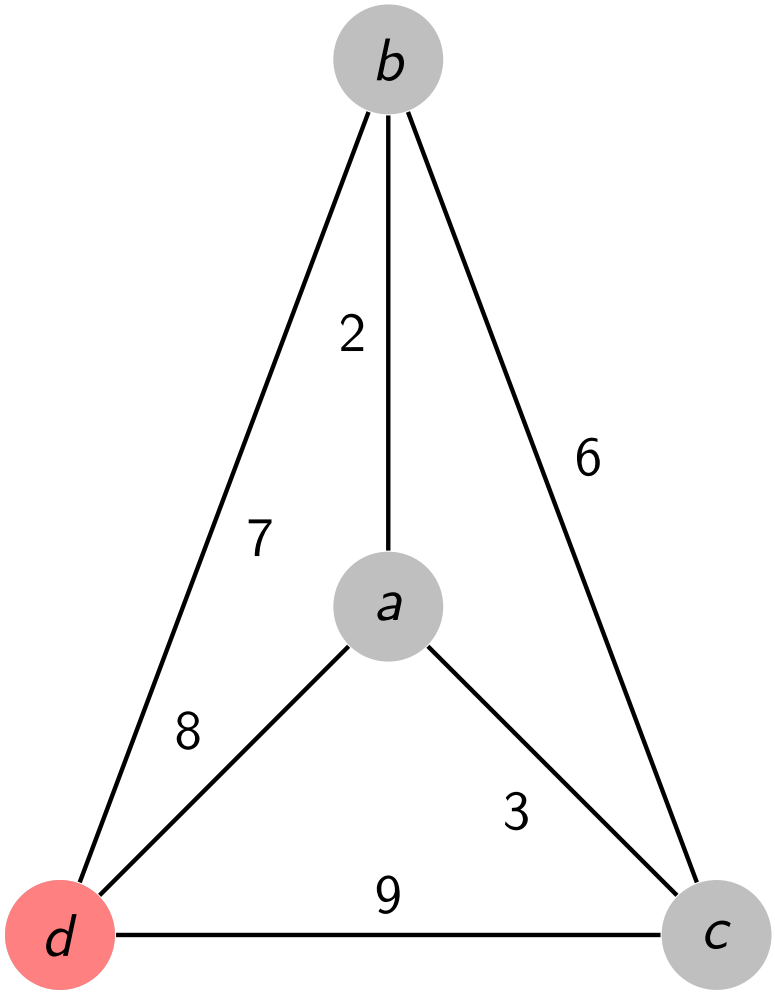
Prim's algorithm/Jarník's algorithm/Invasion percolation process

- Fix a *root vertex*.



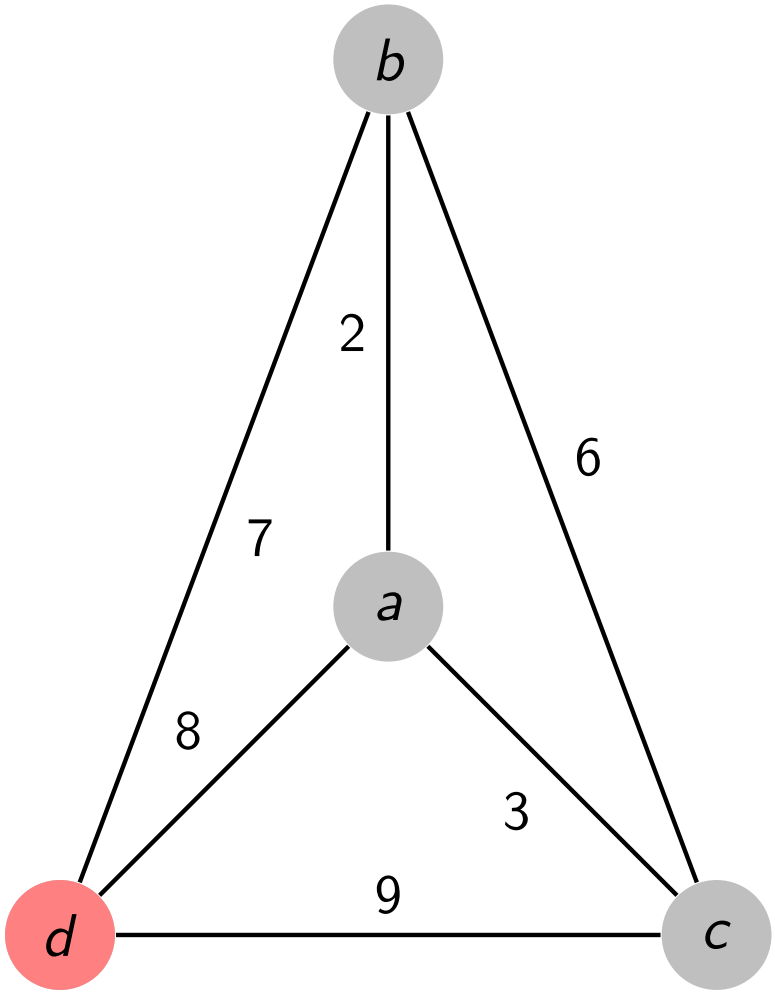
Prim's algorithm/Jarník's algorithm/Invasion percolation process

- Fix a *root vertex*.



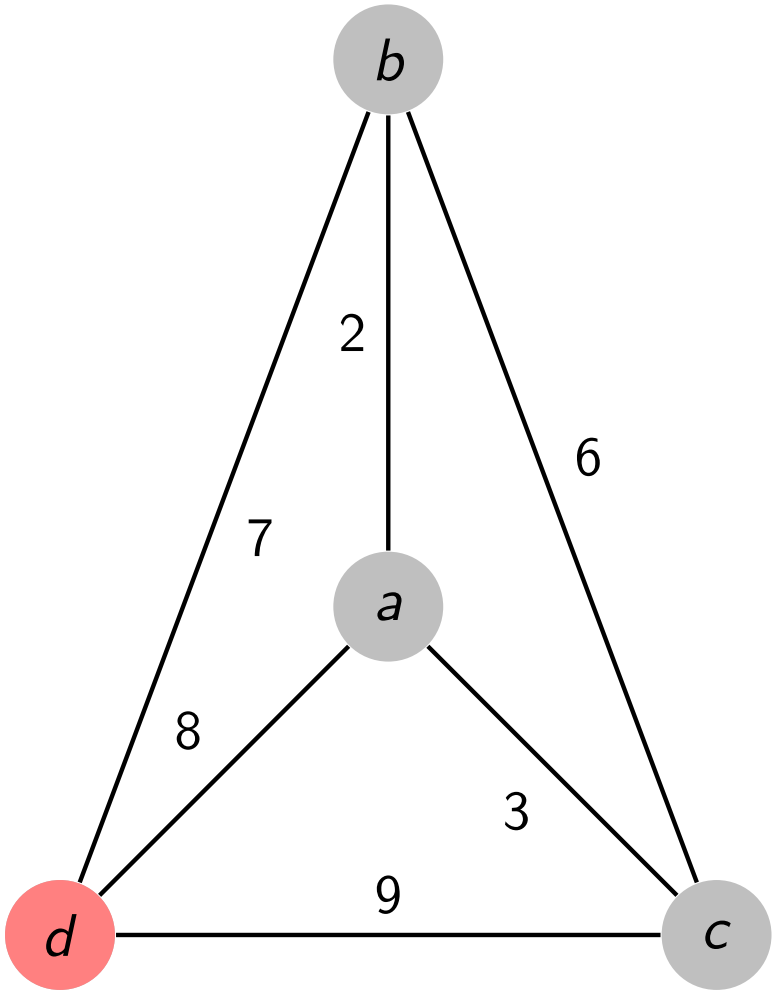
Prim's algorithm/Jarník's algorithm/Invasion percolation process

- Fix a *root vertex*.
- Add the lightest edge leaving the root.



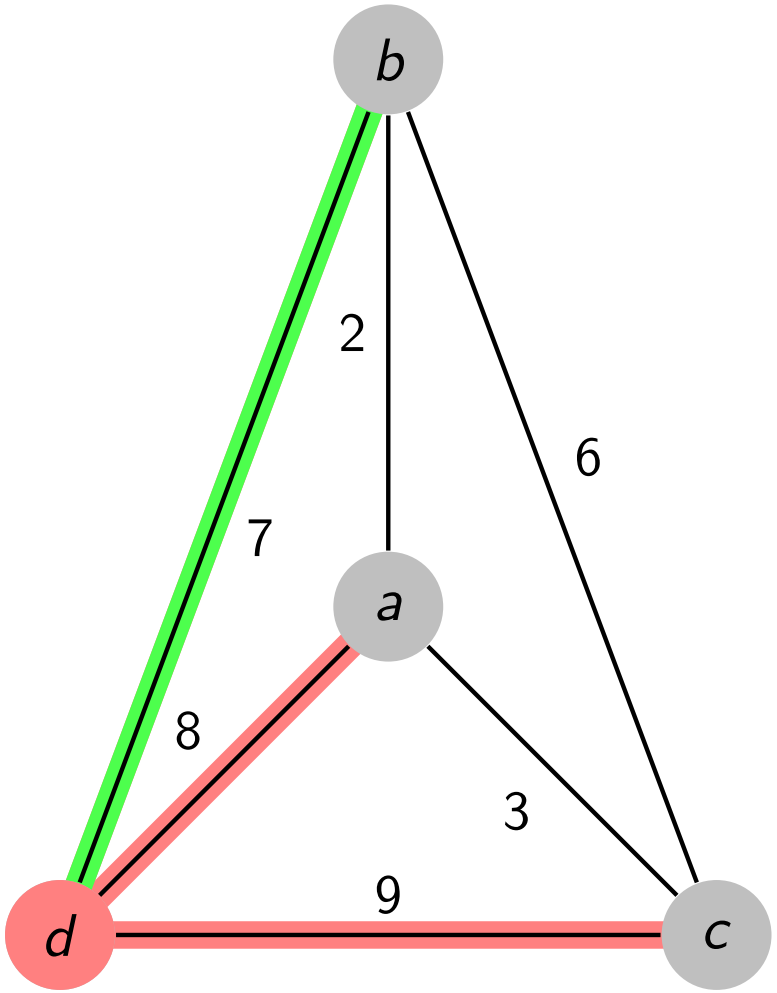
Prim's algorithm/Jarník's algorithm/Invasion percolation process

- Fix a *root vertex*.
- Add the lightest edge leaving the root.
- At each step, add the lightest edge *leaving* the component containing the root.



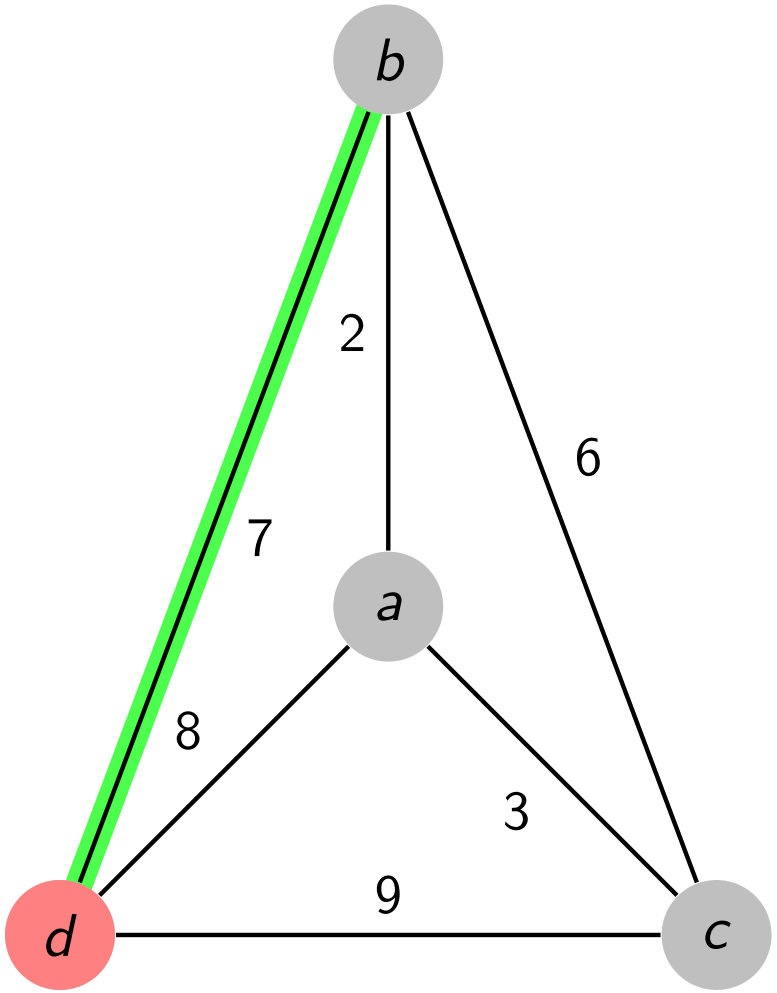
Prim's algorithm/Jarník's algorithm/Invasion percolation process

- Fix a *root vertex*.
- Add the lightest edge leaving the root.
- At each step, add the lightest edge *leaving* the component containing the root.



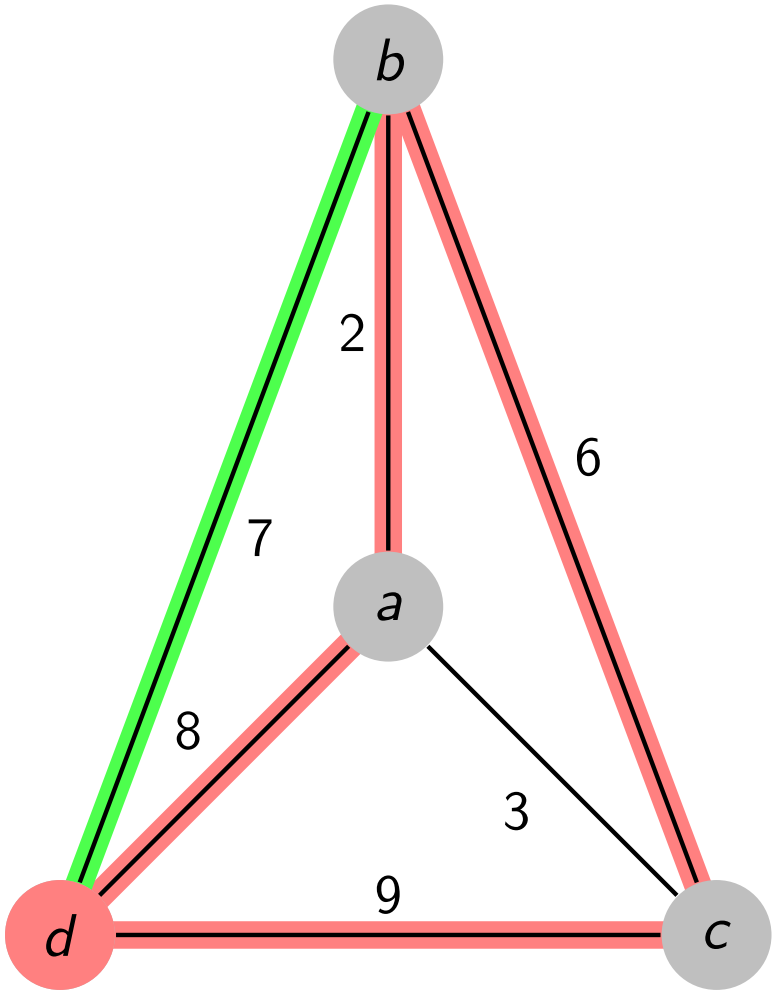
Prim's algorithm/Jarník's algorithm/Invasion percolation process

- Fix a *root vertex*.
- Add the lightest edge leaving the root.
- At each step, add the lightest edge *leaving* the component containing the root.



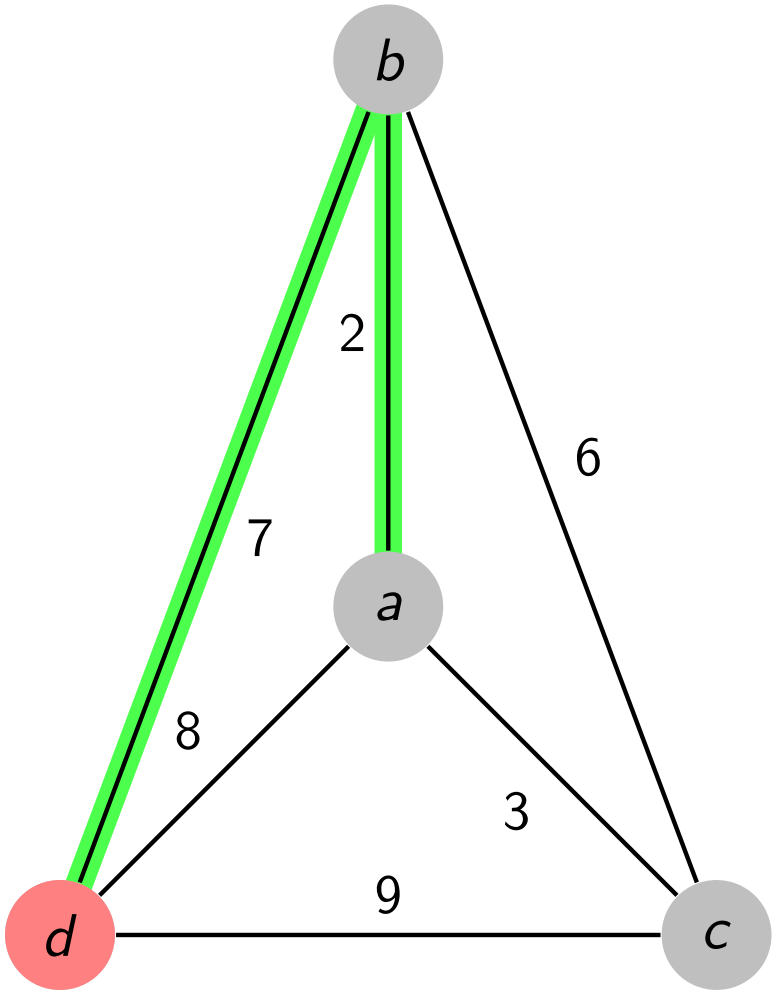
Prim's algorithm/Jarník's algorithm/Invasion percolation process

- Fix a *root vertex*.
- Add the lightest edge leaving the root.
- At each step, add the lightest edge *leaving* the component containing the root.



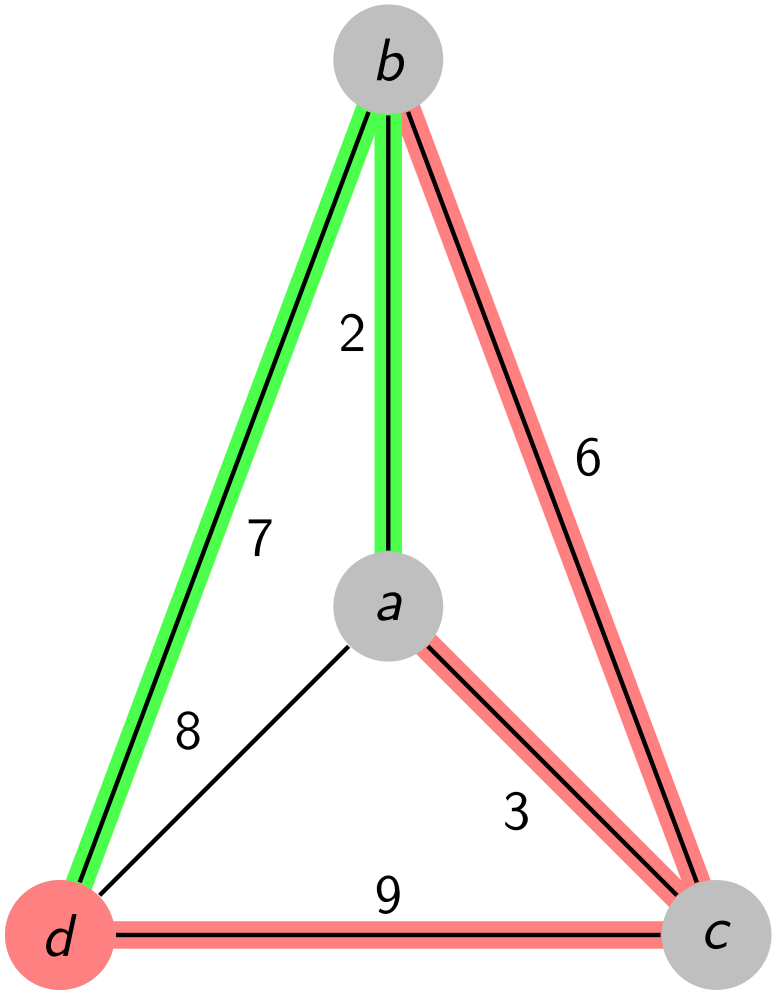
Prim's algorithm/Jarník's algorithm/Invasion percolation process

- Fix a *root vertex*.
- Add the lightest edge leaving the root.
- At each step, add the lightest edge *leaving* the component containing the root.



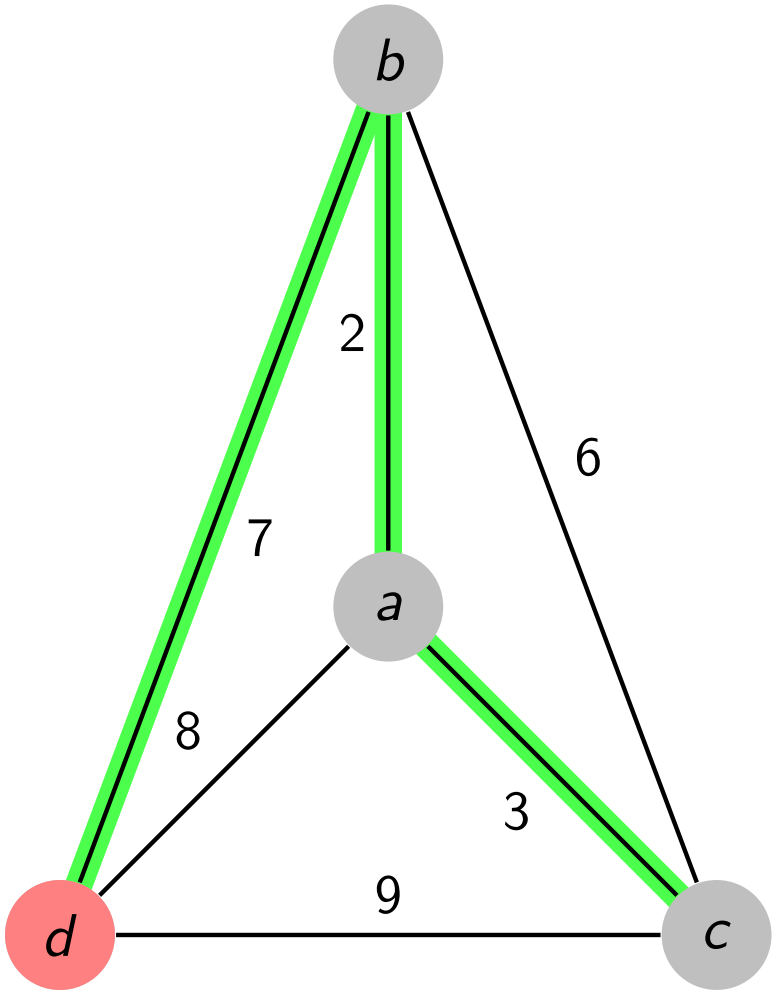
Prim's algorithm/Jarník's algorithm/Invasion percolation process

- Fix a *root vertex*.
- Add the lightest edge leaving the root.
- At each step, add the lightest edge *leaving* the component containing the root.



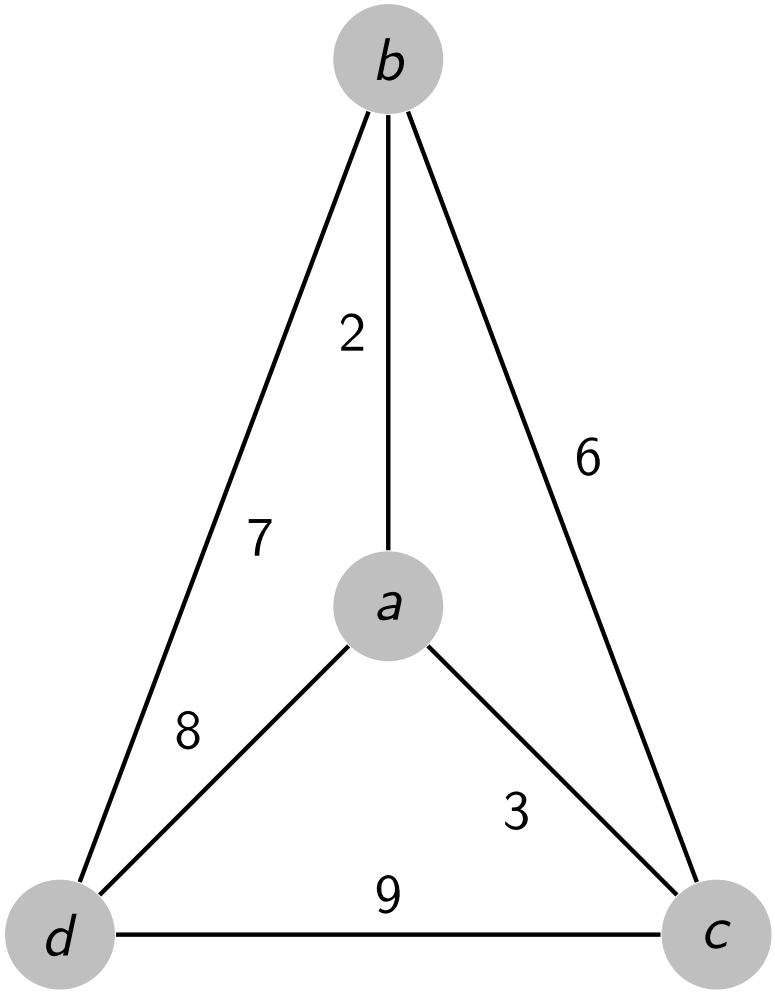
Prim's algorithm/Jarník's algorithm/Invasion percolation process

- Fix a *root vertex*.
- Add the lightest edge leaving the root.
- At each step, add the lightest edge *leaving* the component containing the root.



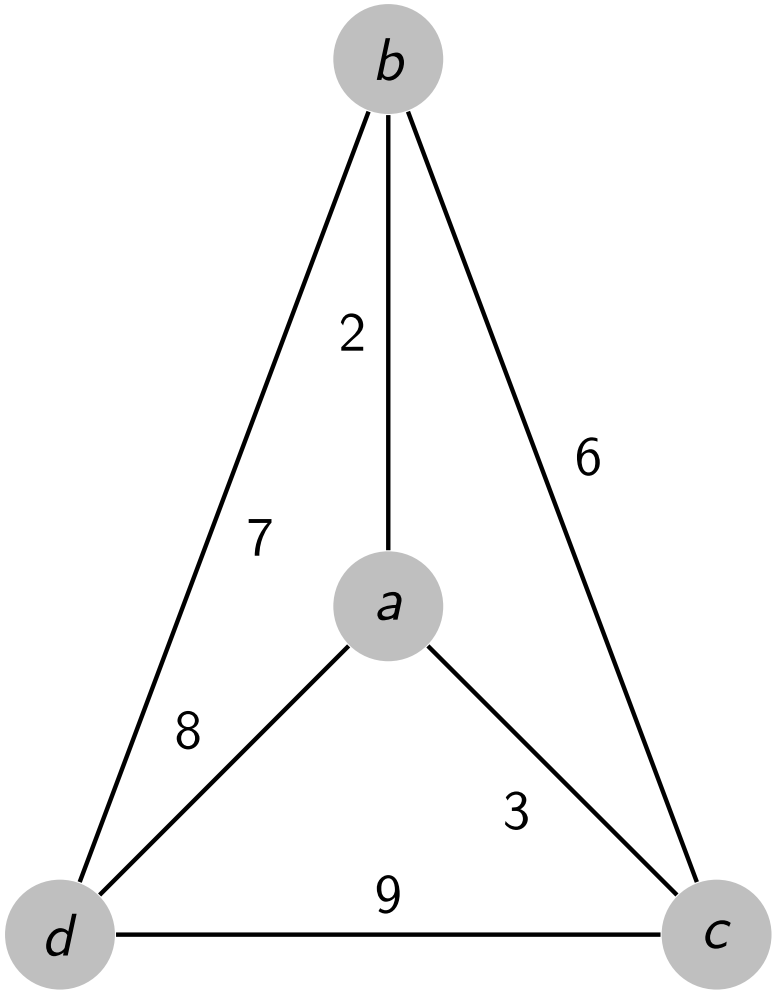
Reverse Kruskal/Cycle breaking

- Start with all edges present.



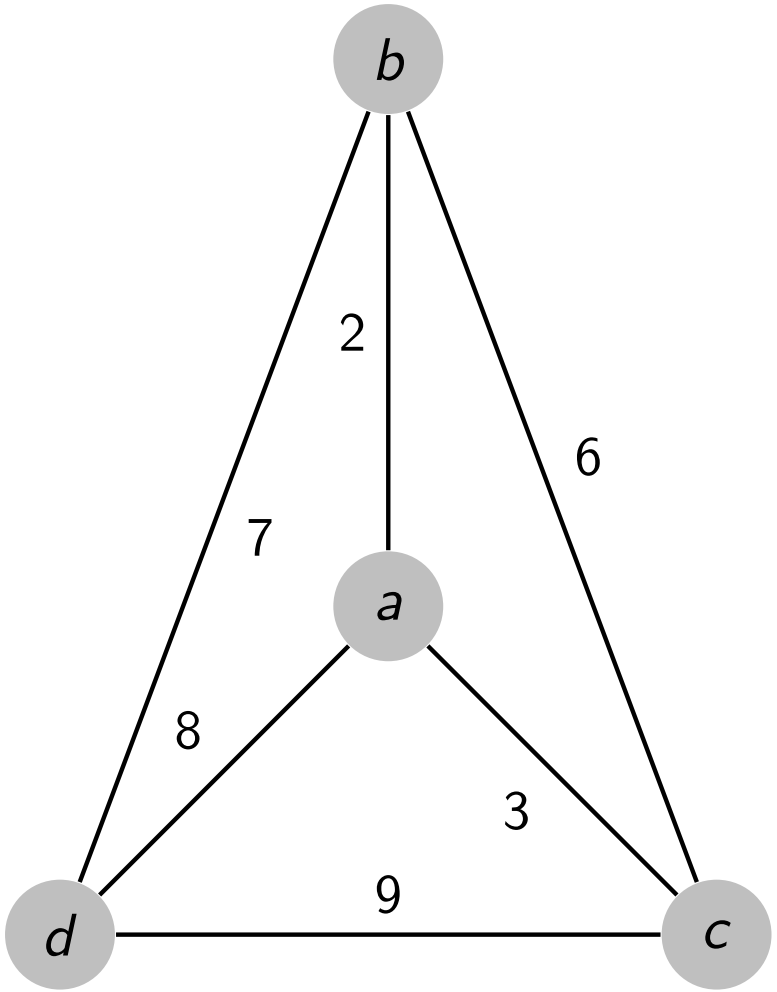
Reverse Kruskal/Cycle breaking

- Start with all edges present.
- Order edges in *decreasing* order of weight as $e_1, \ldots, e_m$.



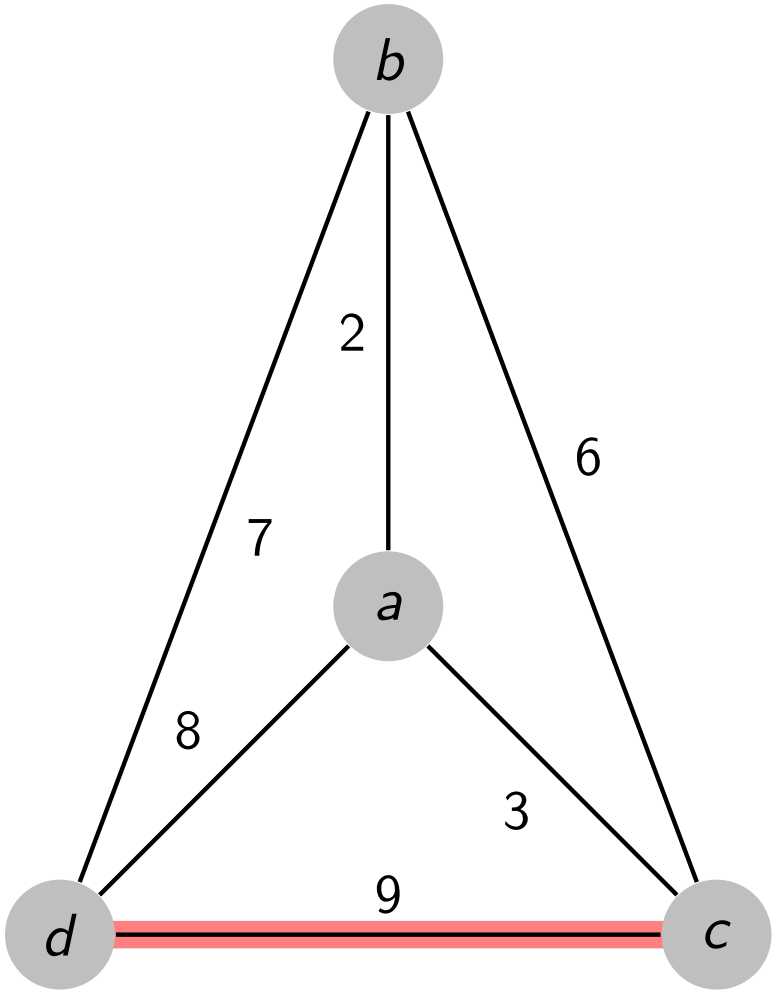
Reverse Kruskal/Cycle breaking

- Start with all edges present.
- Order edges in *decreasing* order of weight as $e_1, \ldots, e_m$.
- For each i, remove edge e_i unless doing so would break a component in two.



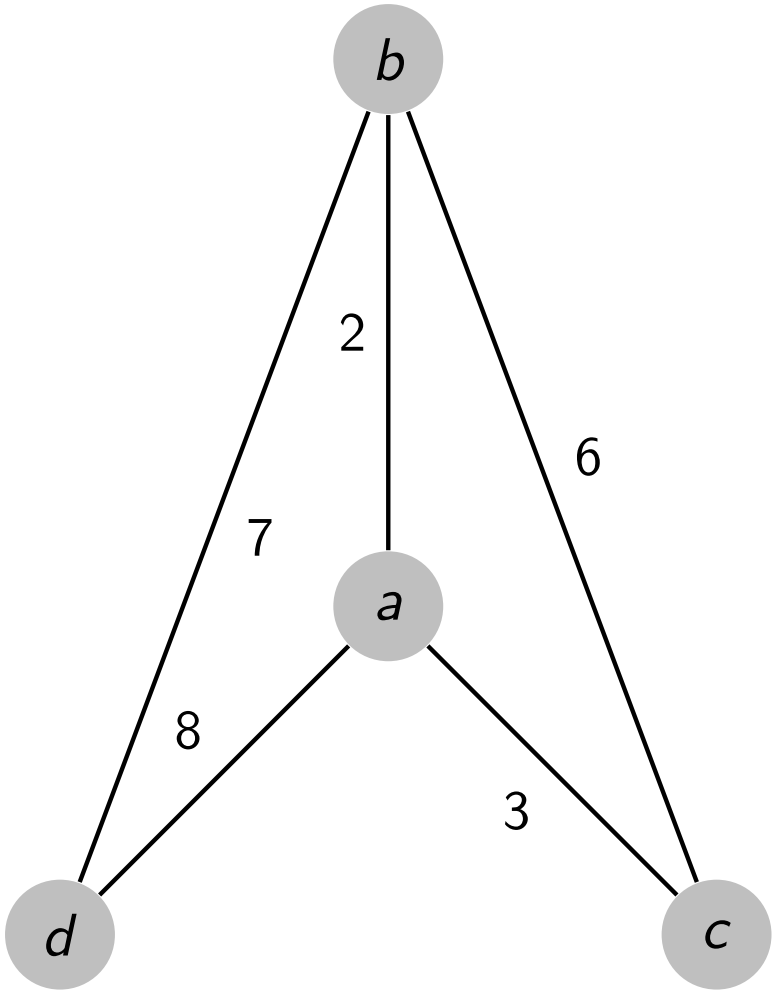
Reverse Kruskal/Cycle breaking

- Start with all edges present.
- Order edges in *decreasing* order of weight as $e_1, \ldots, e_m$.
- For each i, remove edge e_i unless doing so would break a component in two.



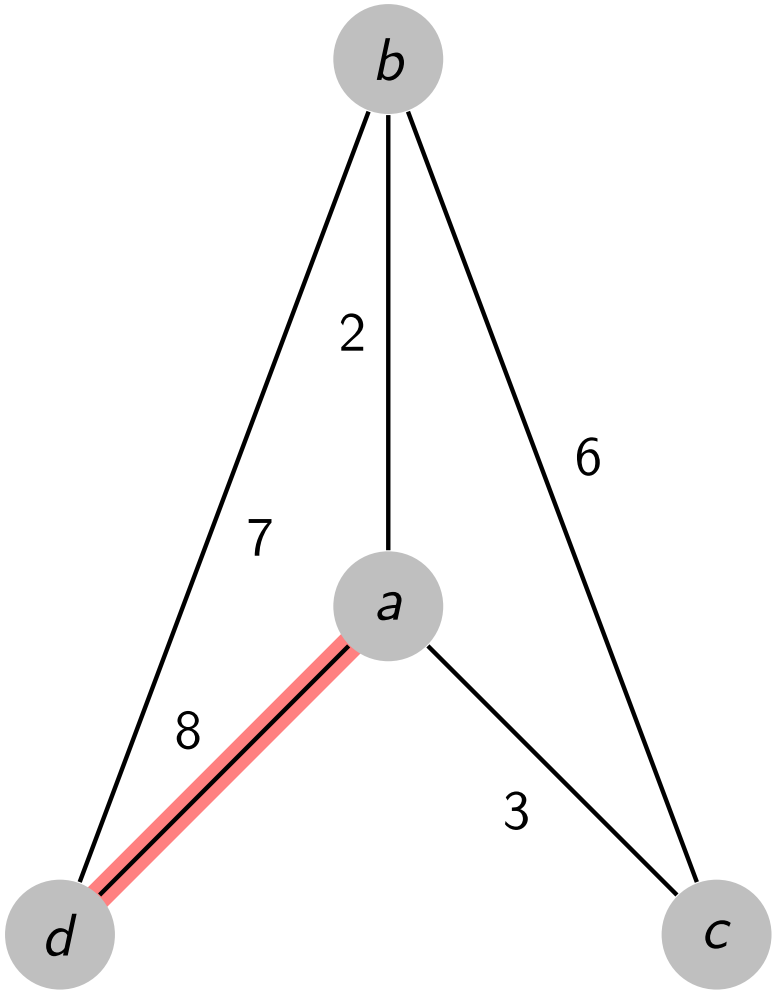
Reverse Kruskal/Cycle breaking

- Start with all edges present.
- Order edges in *decreasing* order of weight as $e_1, \ldots, e_m$.
- For each i, remove edge e_i unless doing so would break a component in two.



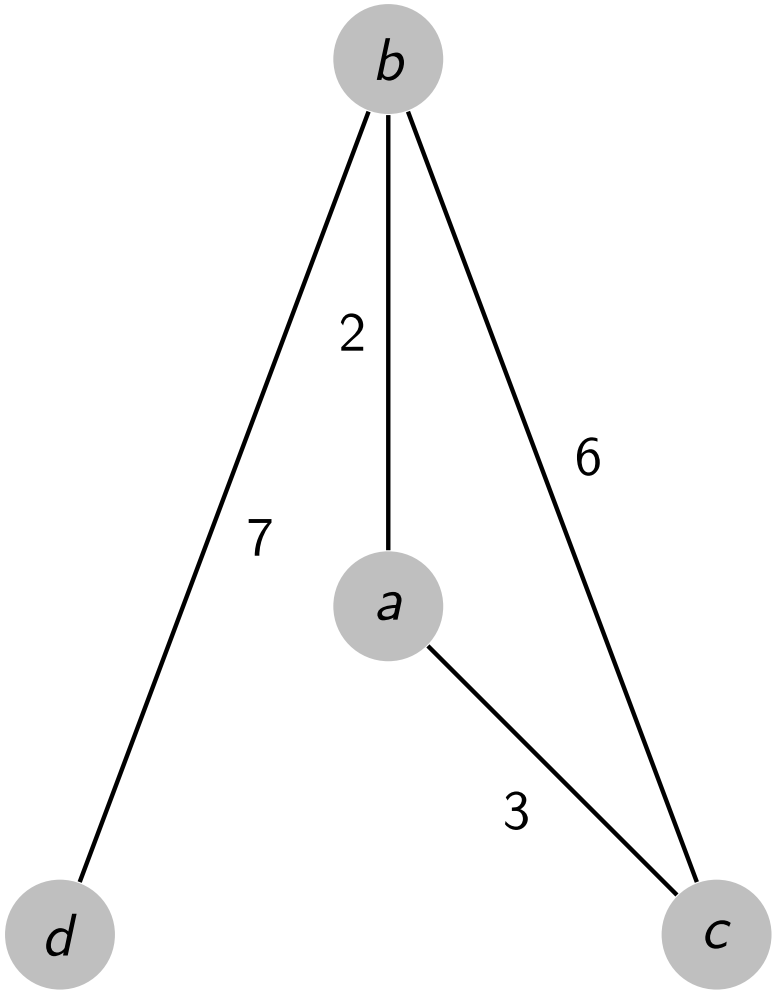
Reverse Kruskal/Cycle breaking

- Start with all edges present.
- Order edges in *decreasing* order of weight as $e_1, \ldots, e_m$.
- For each i, remove edge e_i unless doing so would break a component in two.



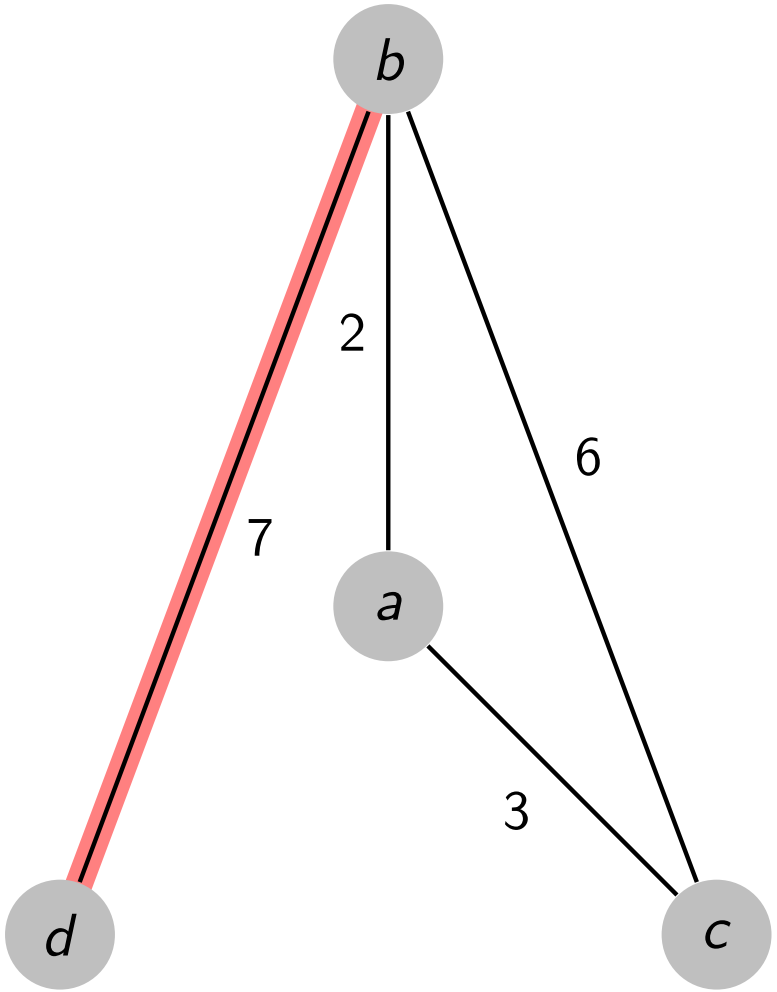
Reverse Kruskal/Cycle breaking

- Start with all edges present.
- Order edges in *decreasing* order of weight as $e_1, \ldots, e_m$.
- For each i, remove edge e_i unless doing so would break a component in two.



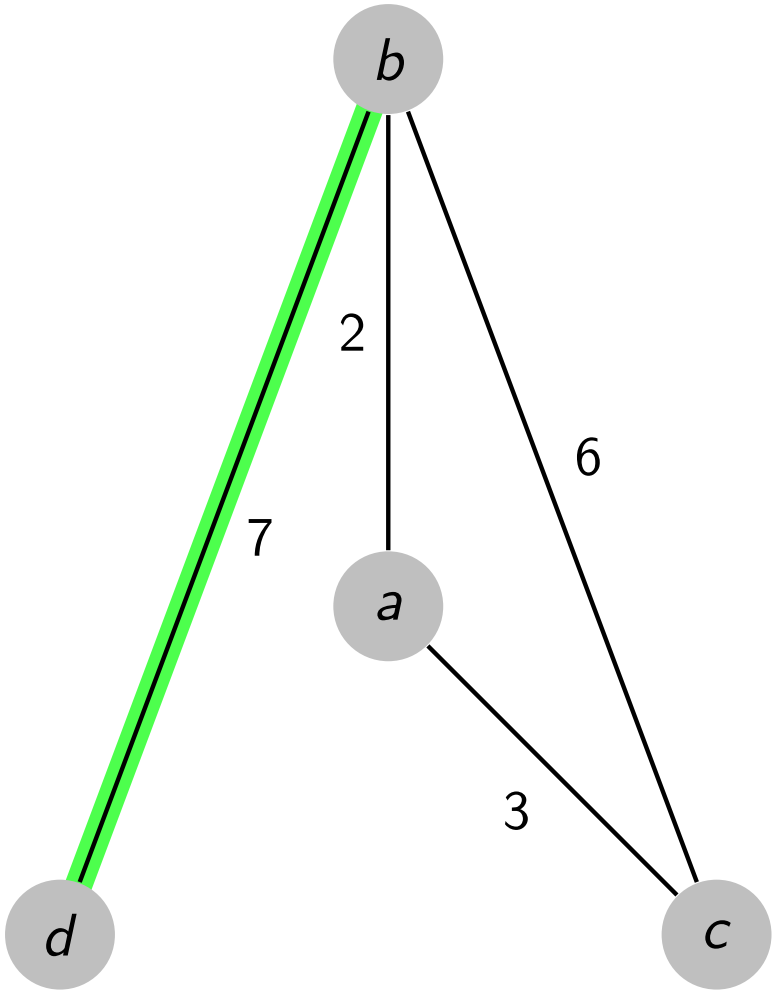
Reverse Kruskal/Cycle breaking

- Start with all edges present.
- Order edges in *decreasing* order of weight as $e_1, \ldots, e_m$.
- For each i, remove edge e_i unless doing so would break a component in two.



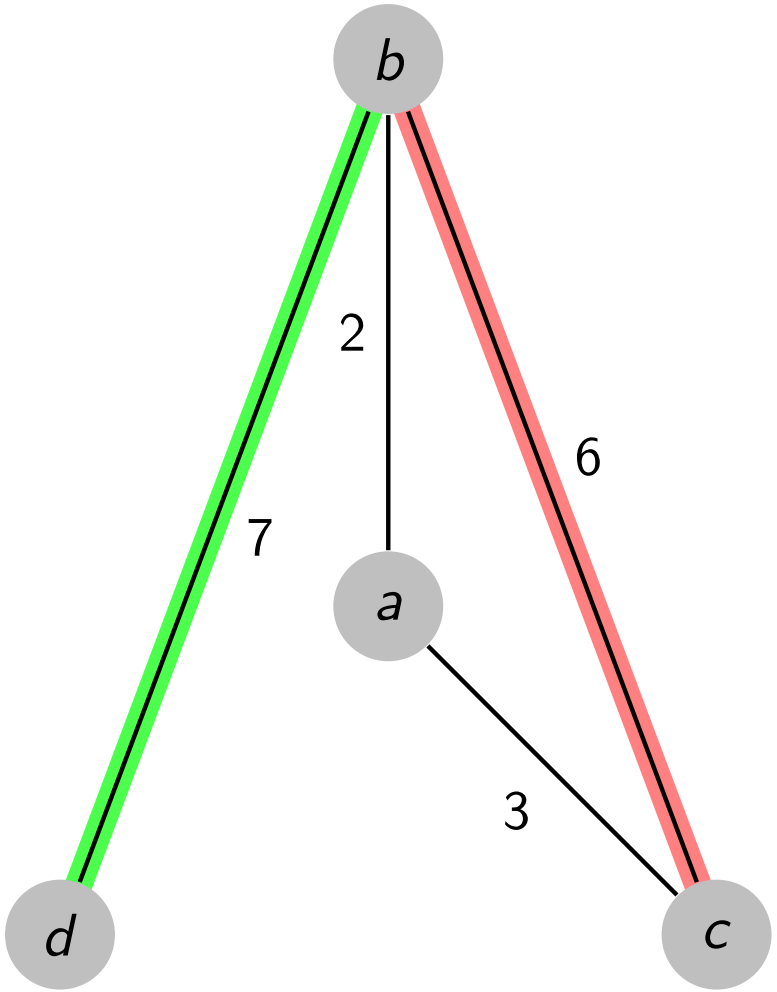
Reverse Kruskal/Cycle breaking

- Start with all edges present.
- Order edges in *decreasing* order of weight as $e_1, \ldots, e_m$.
- For each i, remove edge e_i unless doing so would break a component in two.



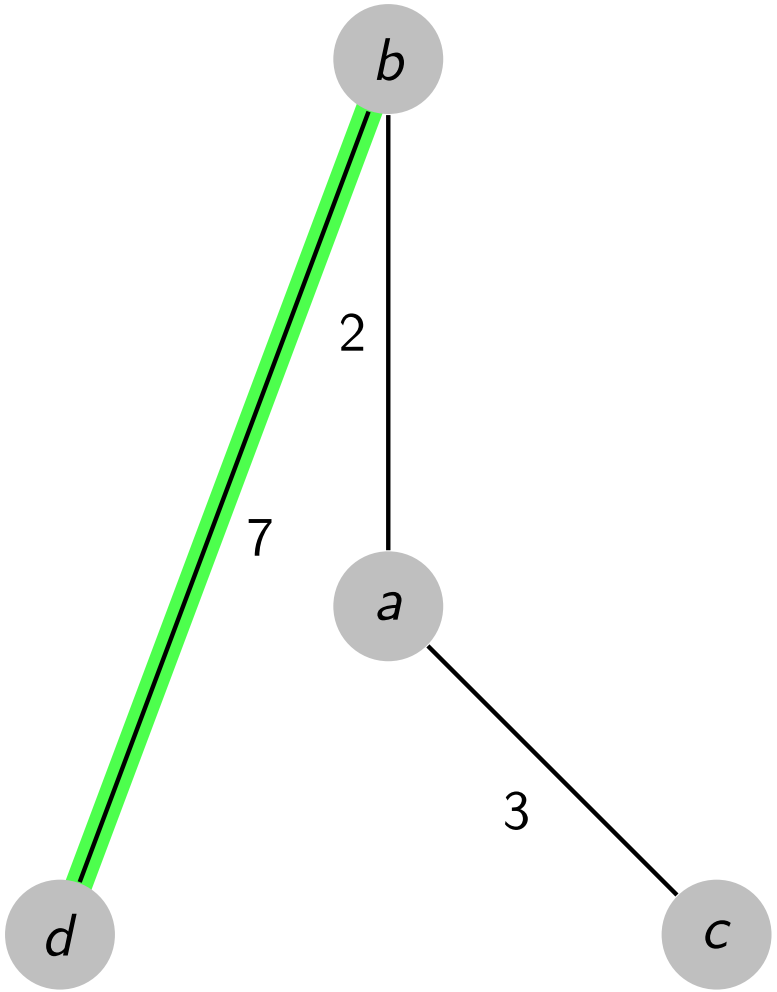
Reverse Kruskal/Cycle breaking

- Start with all edges present.
- Order edges in *decreasing* order of weight as $e_1, \ldots, e_m$.
- For each i, remove edge e_i unless doing so would break a component in two.



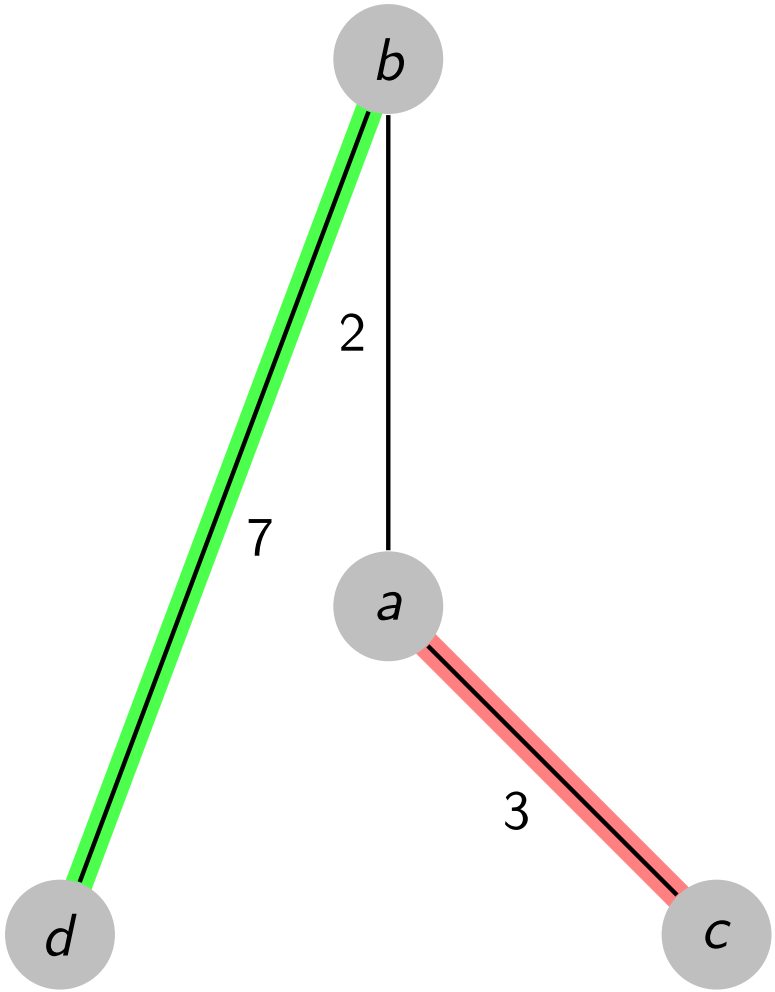
Reverse Kruskal/Cycle breaking

- Start with all edges present.
- Order edges in *decreasing* order of weight as $e_1, \ldots, e_m$.
- For each i, remove edge e_i unless doing so would break a component in two.



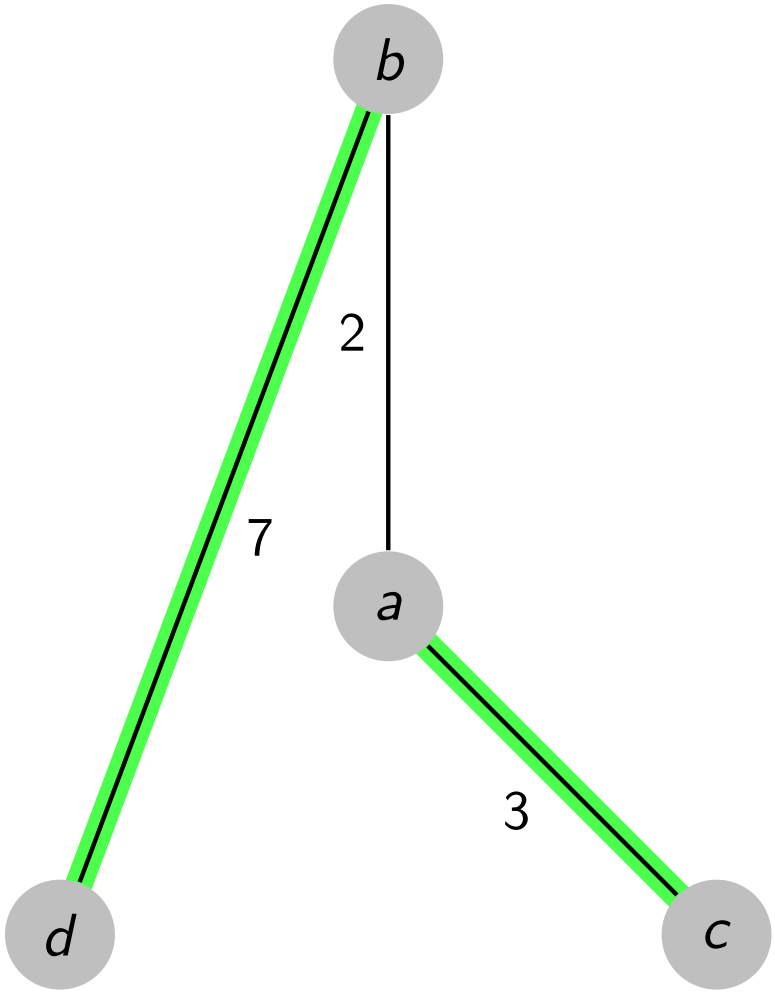
Reverse Kruskal/Cycle breaking

- Start with all edges present.
- Order edges in *decreasing* order of weight as $e_1, \ldots, e_m$.
- For each i, remove edge e_i unless doing so would break a component in two.



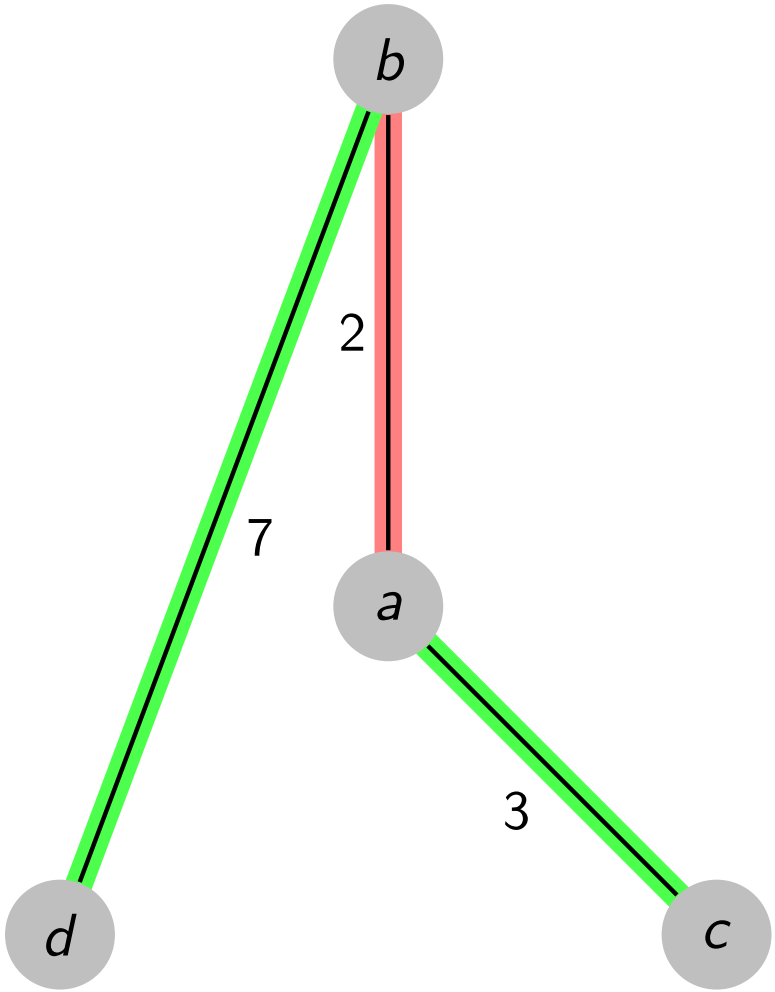
Reverse Kruskal/Cycle breaking

- Start with all edges present.
- Order edges in *decreasing* order of weight as $e_1, \ldots, e_m$.
- For each i, remove edge e_i unless doing so would break a component in two.



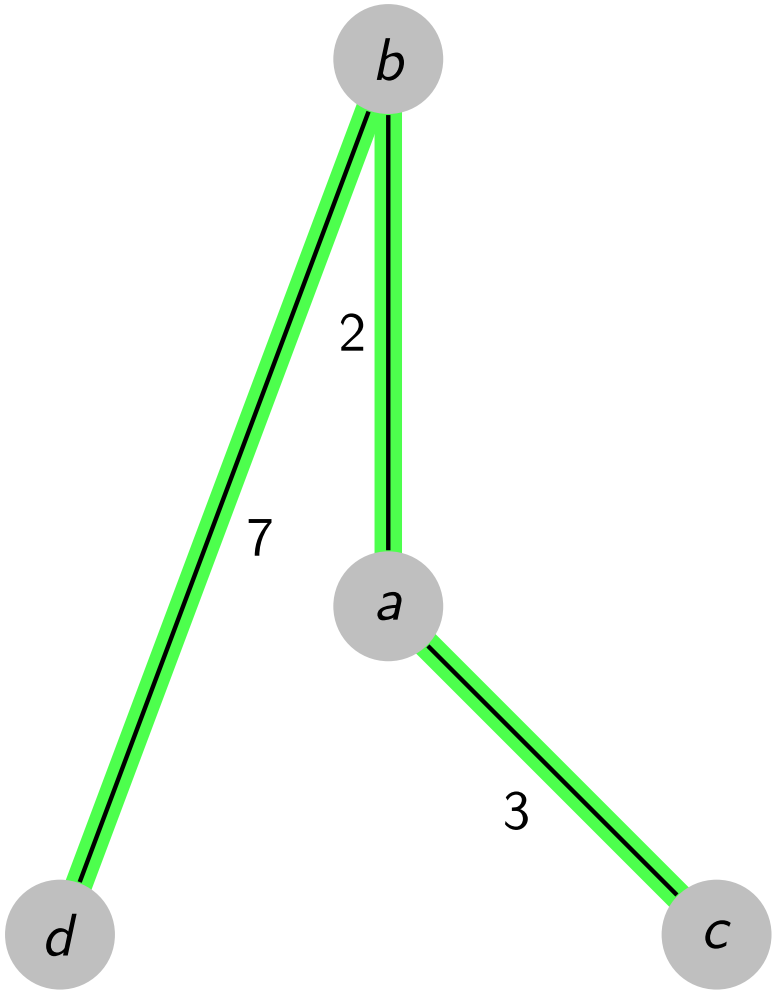
Reverse Kruskal/Cycle breaking

- Start with all edges present.
- Order edges in *decreasing* order of weight as $e_1, \ldots, e_m$.
- For each i, remove edge e_i unless doing so would break a component in two.



Reverse Kruskal/Cycle breaking

- Start with all edges present.
- Order edges in *decreasing* order of weight as $e_1, \ldots, e_m$.
- For each i, remove edge e_i unless doing so would break a component in two.

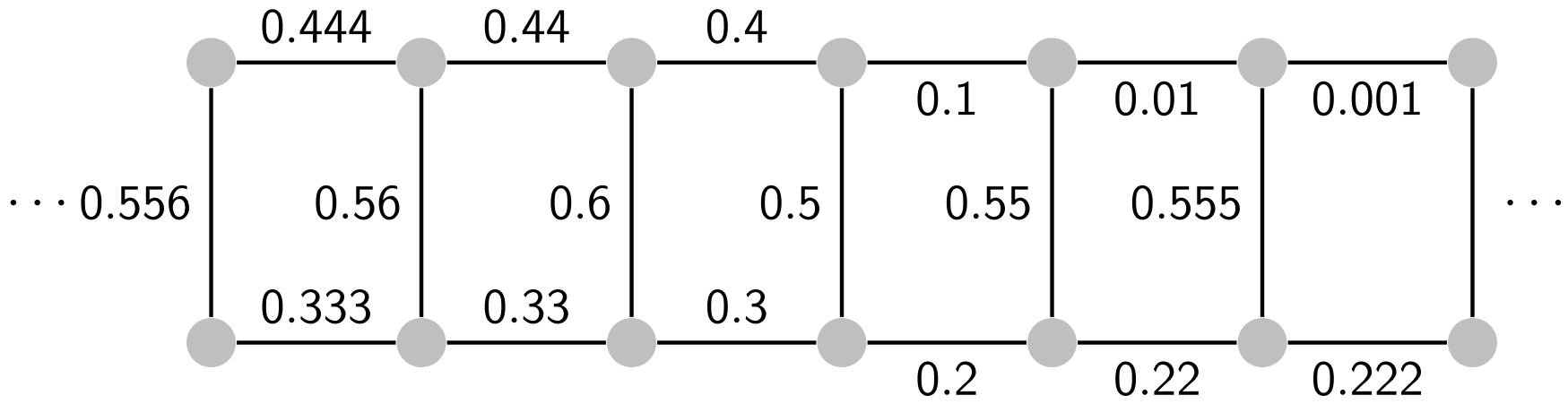


MSTs of infinite graphs.

Each algorithm runs into problems when applied to infinite graphs.

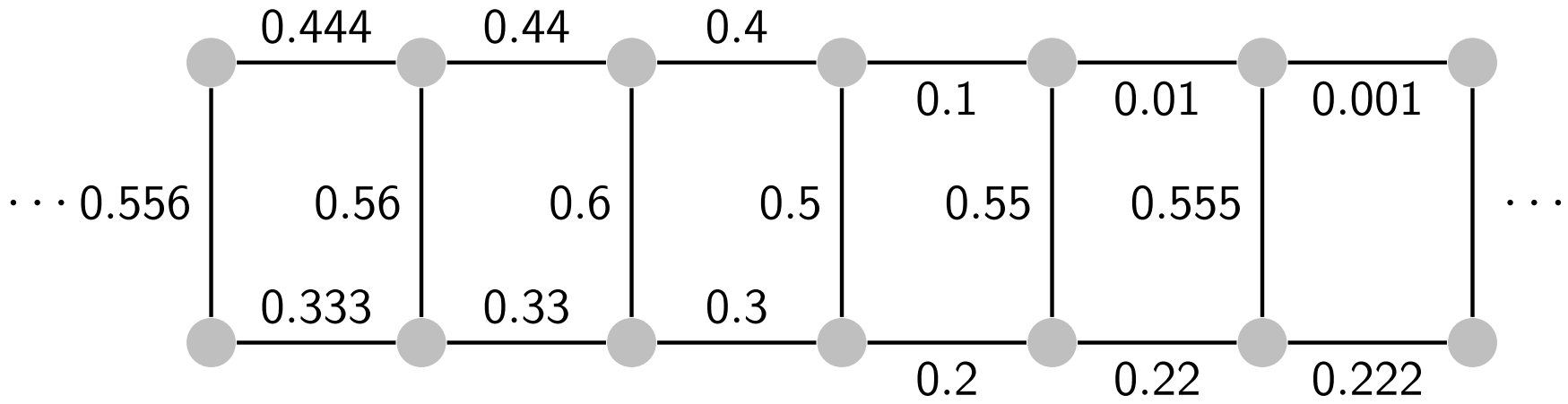
MSTs of infinite graphs.

Each algorithm runs into problems when applied to infinite graphs.



MSTs of infinite graphs.

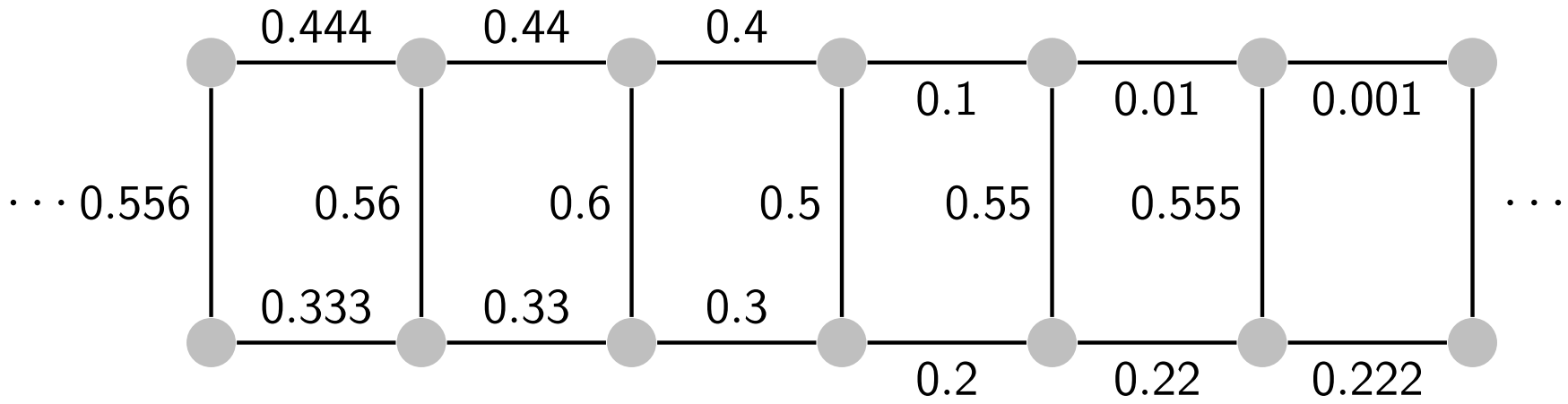
Each algorithm runs into problems when applied to infinite graphs.



Kruskal's algorithm:

MSTs of infinite graphs.

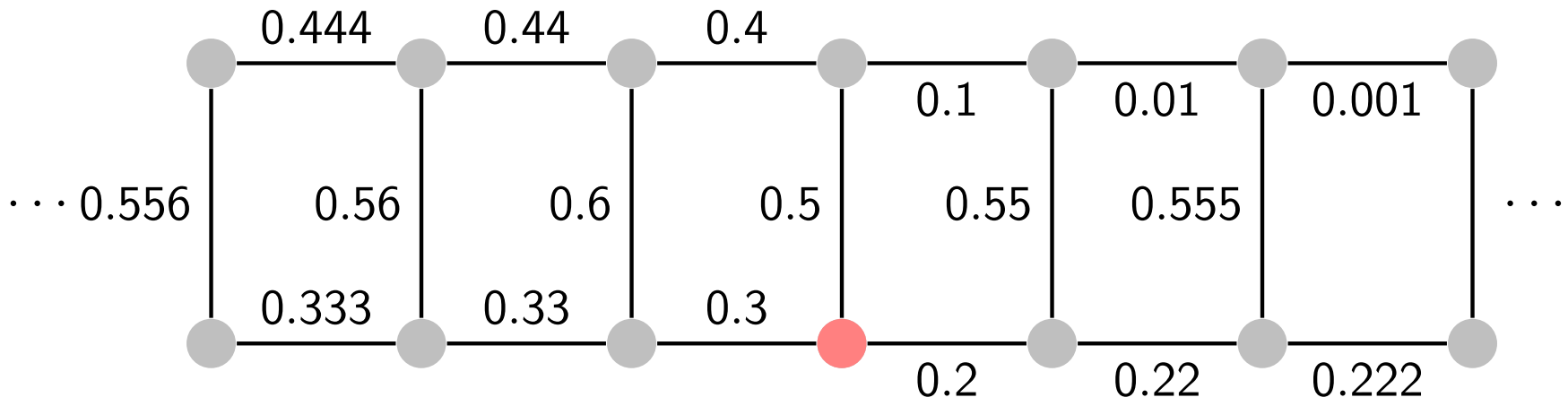
Each algorithm runs into problems when applied to infinite graphs.



Kruskal's algorithm: There is no smallest edge!

MSTs of infinite graphs.

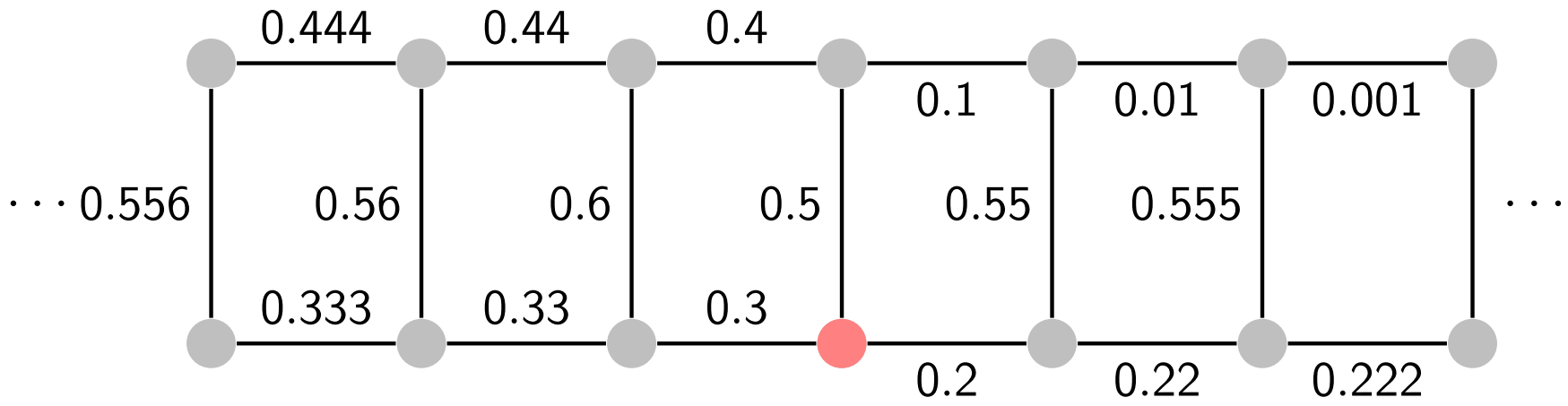
Each algorithm runs into problems when applied to infinite graphs.



Prim's algorithm:

MSTs of infinite graphs.

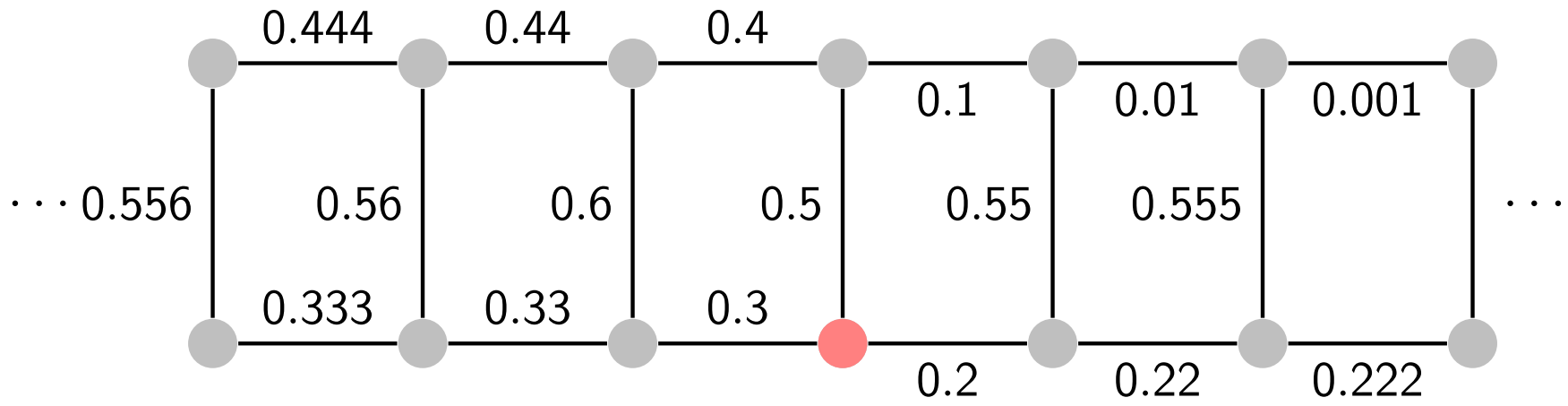
Each algorithm runs into problems when applied to infinite graphs.



Prim's algorithm: Well defined on locally finite graphs.

MSTs of infinite graphs.

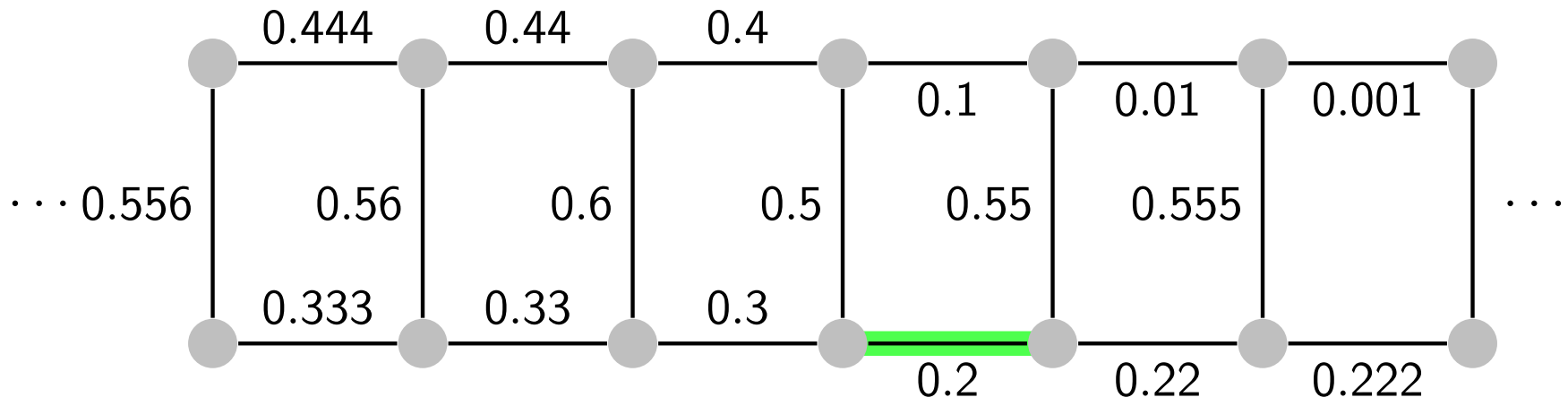
Each algorithm runs into problems when applied to infinite graphs.



Prim's algorithm: Well defined on locally finite graphs. But may not build a spanning tree (certain vertices never reached).

MSTs of infinite graphs.

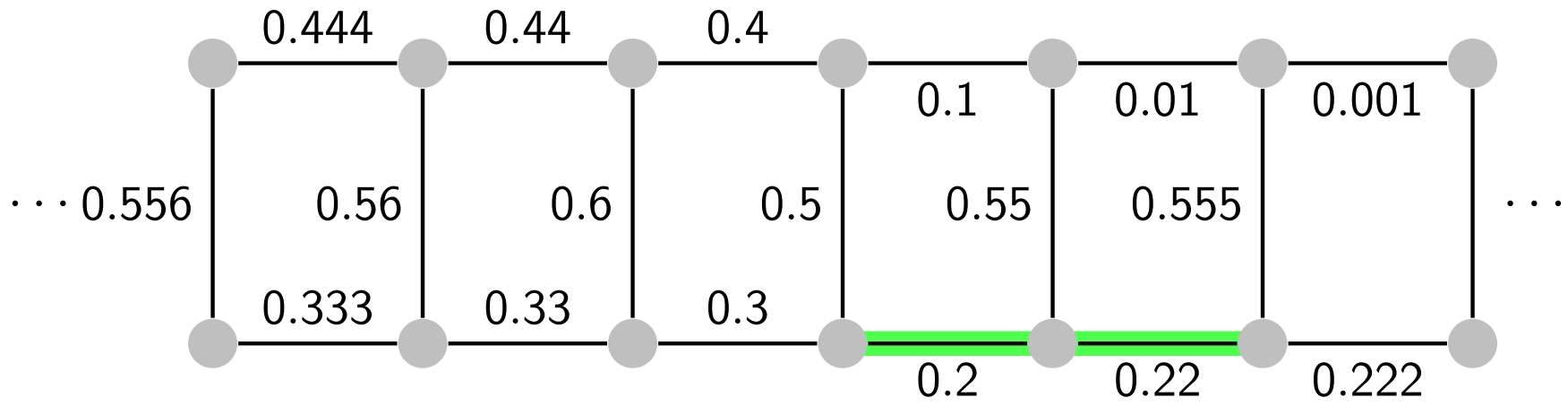
Each algorithm runs into problems when applied to infinite graphs.



Prim's algorithm: Well defined on locally finite graphs. But may not build a spanning tree (certain vertices never reached).

MSTs of infinite graphs.

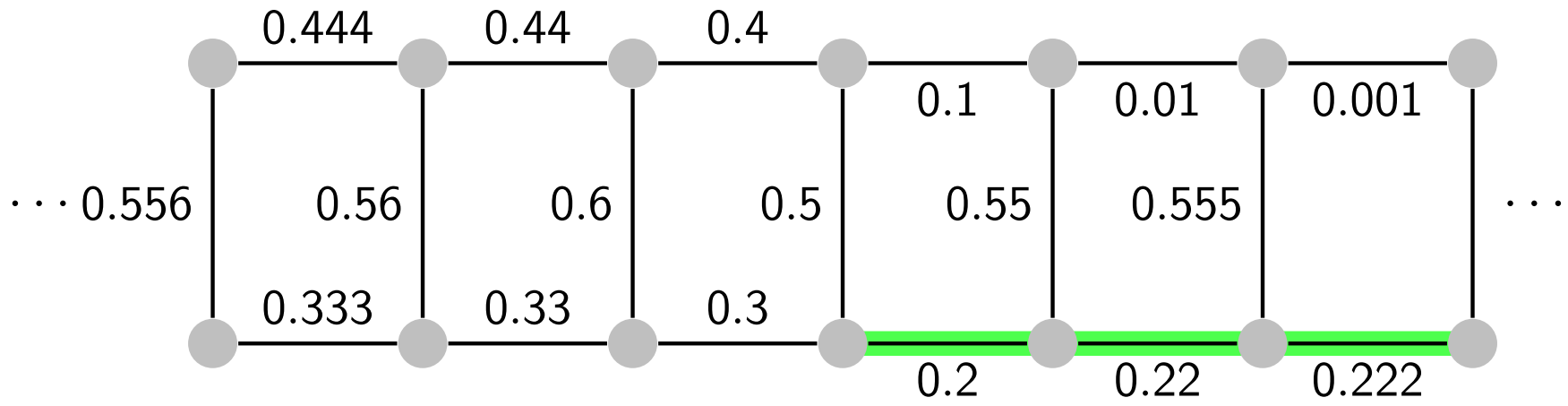
Each algorithm runs into problems when applied to infinite graphs.



Prim's algorithm: Well defined on locally finite graphs. But may not build a spanning tree (certain vertices never reached).

MSTs of infinite graphs.

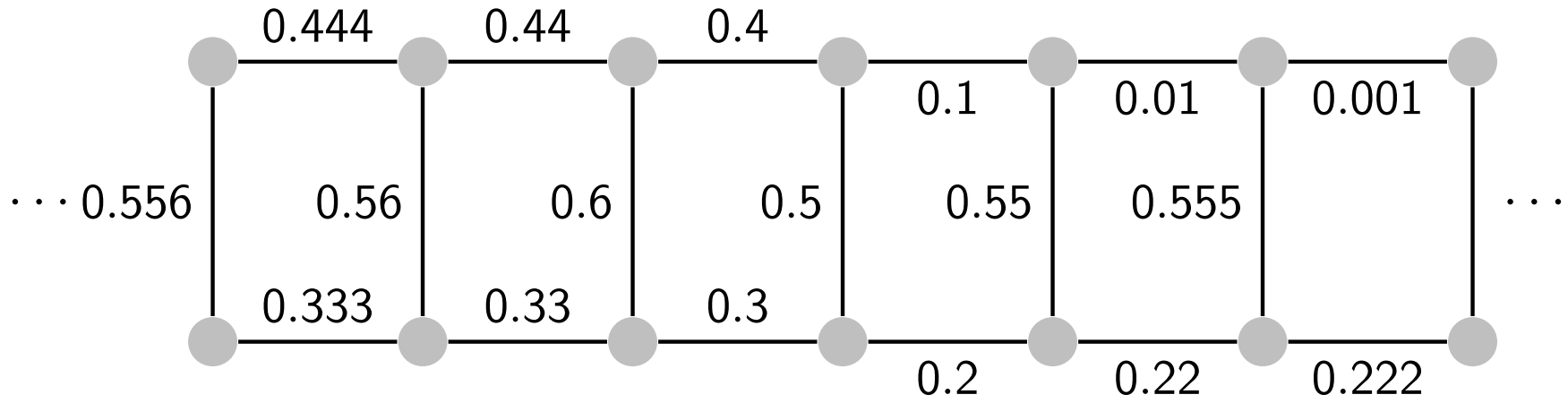
Each algorithm runs into problems when applied to infinite graphs.



Prim's algorithm: Well defined on locally finite graphs. But may not build a spanning tree (certain vertices never reached).

MSTs of infinite graphs.

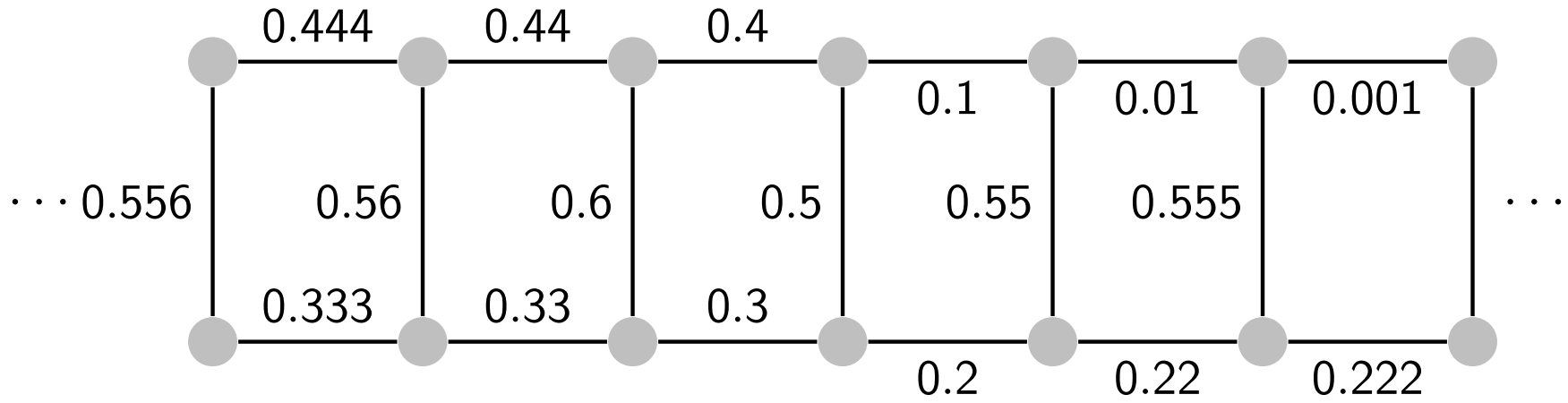
Each algorithm runs into problems when applied to infinite graphs.



Cycle breaking:

MSTs of infinite graphs.

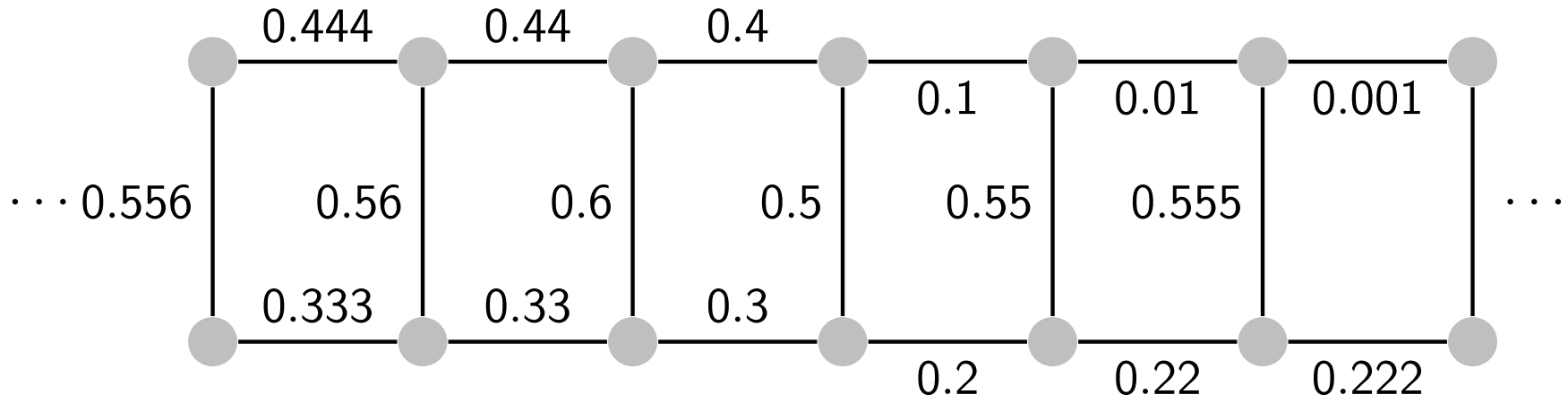
Each algorithm runs into problems when applied to infinite graphs.



Cycle breaking: May not be well-defined.

MSTs of infinite graphs.

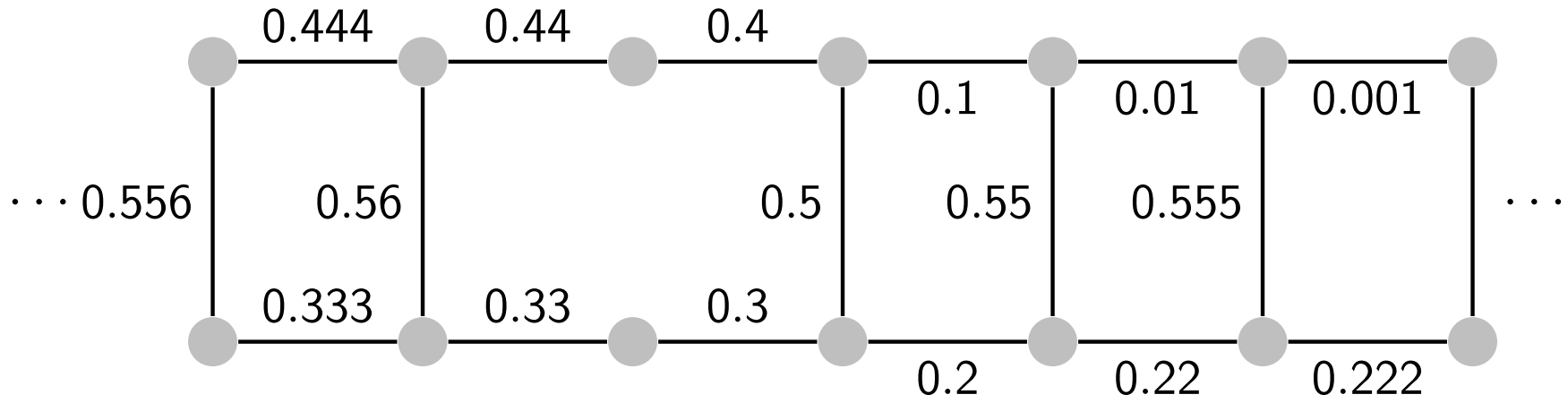
Each algorithm runs into problems when applied to infinite graphs.



Cycle breaking: May not be well-defined. Even when well-defined, may not break all cycles.

MSTs of infinite graphs.

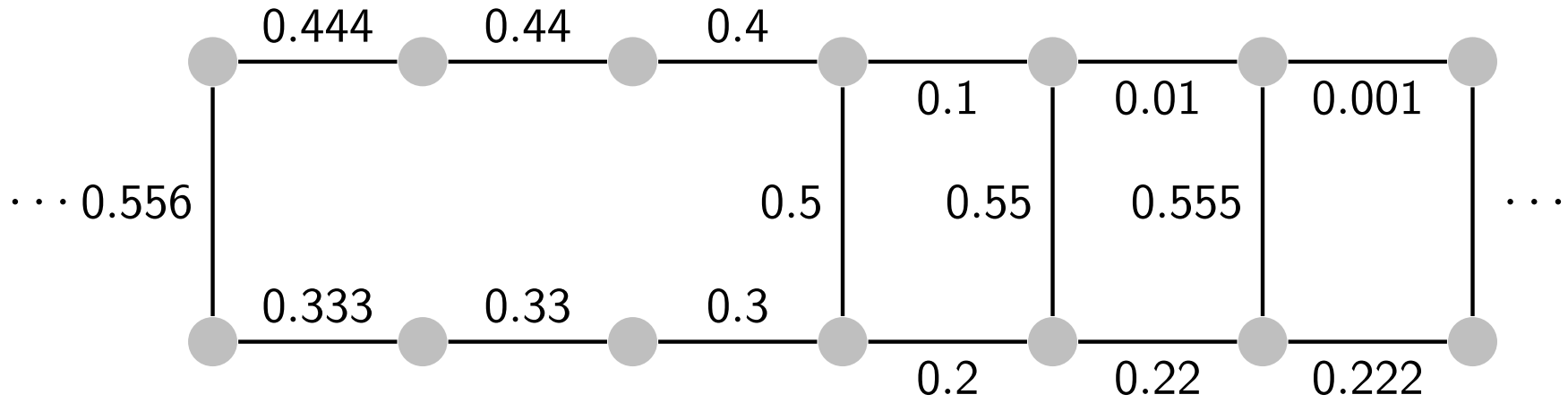
Each algorithm runs into problems when applied to infinite graphs.



Cycle breaking: May not be well-defined. Even when well-defined, may not break all cycles.

MSTs of infinite graphs.

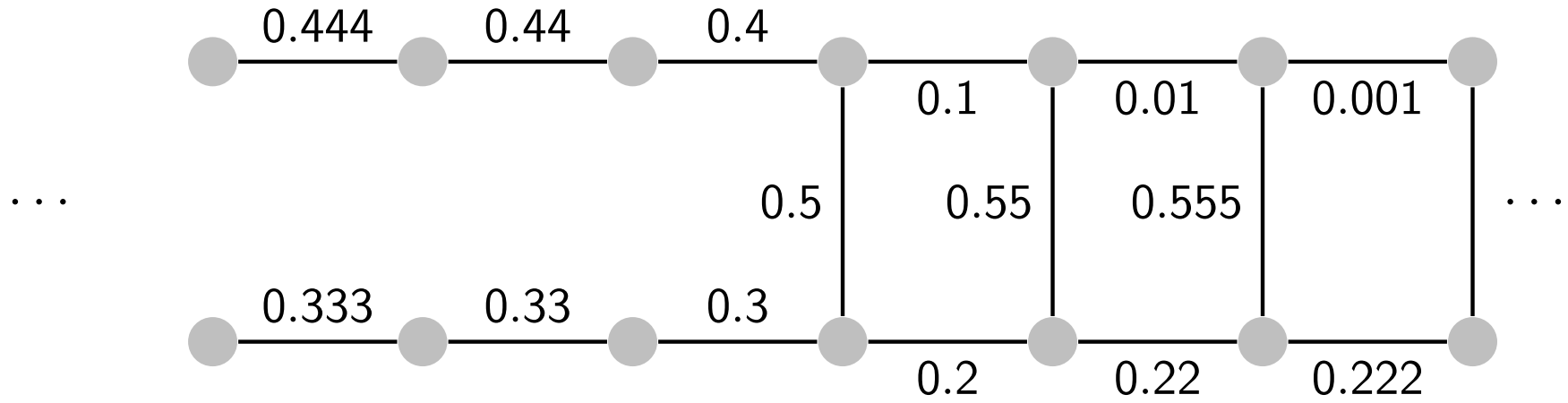
Each algorithm runs into problems when applied to infinite graphs.



Cycle breaking: May not be well-defined. Even when well-defined, may not break all cycles.

MSTs of infinite graphs.

Each algorithm runs into problems when applied to infinite graphs.



Cycle breaking: May not be well-defined. Even when well-defined, may not break all cycles.

MSTs of infinite graphs.

The right definition

In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation,

MSTs of infinite graphs.

The right definition

In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.

MSTs of infinite graphs.

The right definition

In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.

MSTs of infinite graphs.

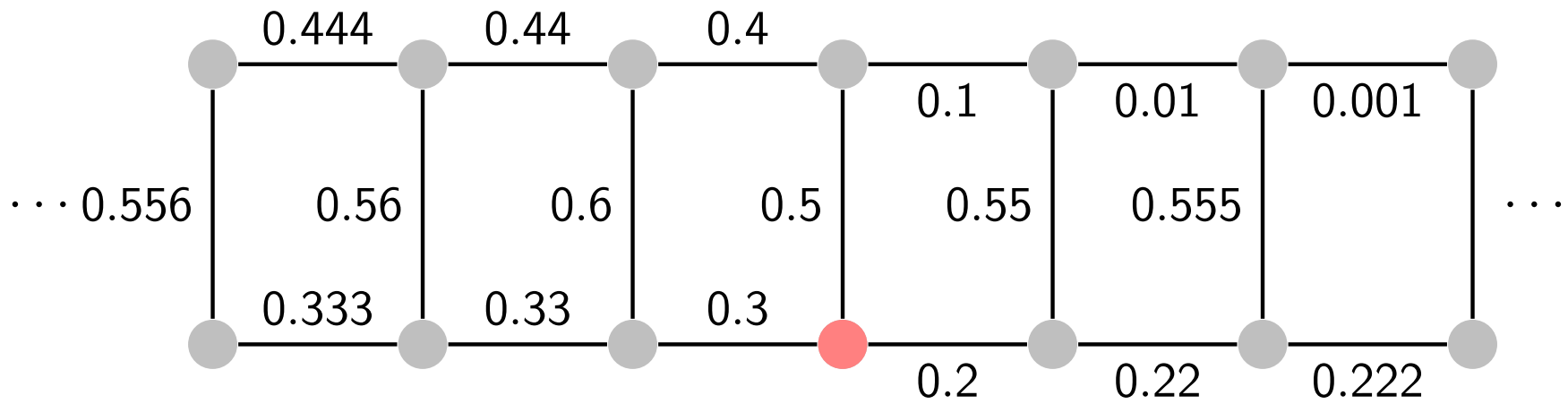
The right definition

In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.

MSTs of infinite graphs.

The right definition

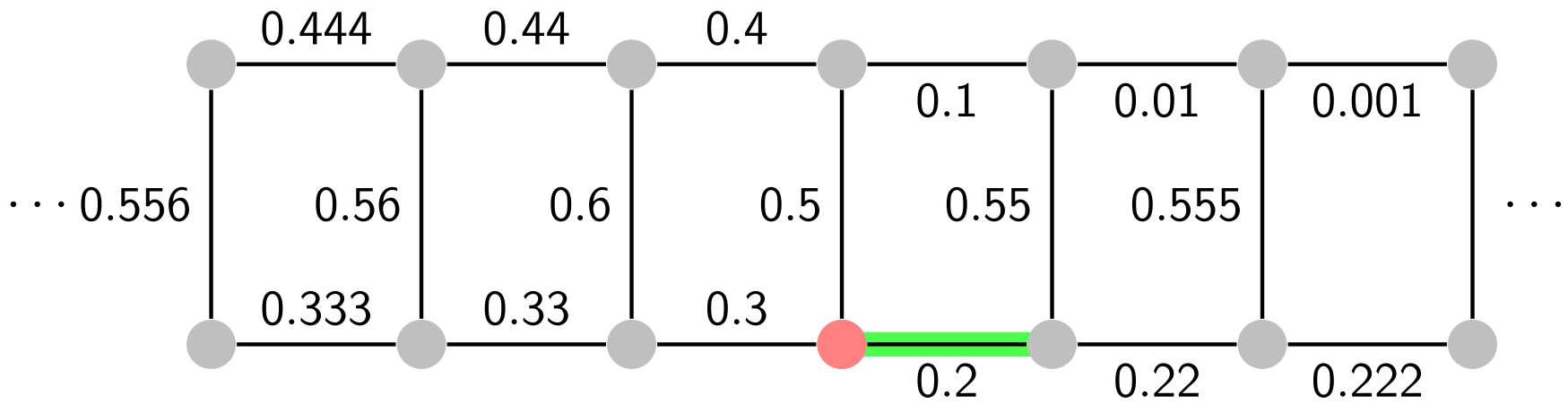
In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.



MSTs of infinite graphs.

The right definition

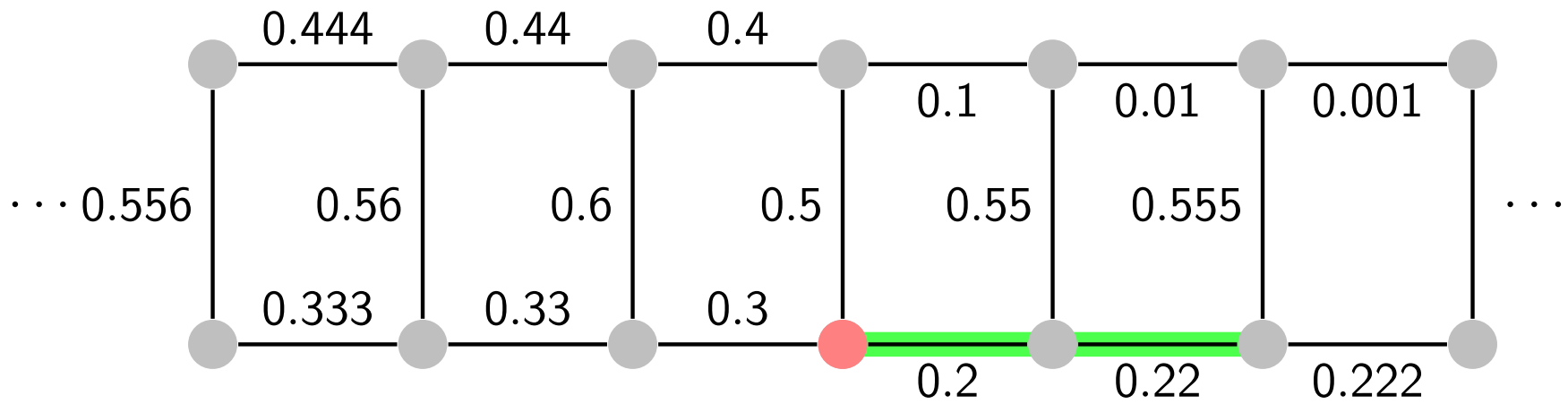
In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.



MSTs of infinite graphs.

The right definition

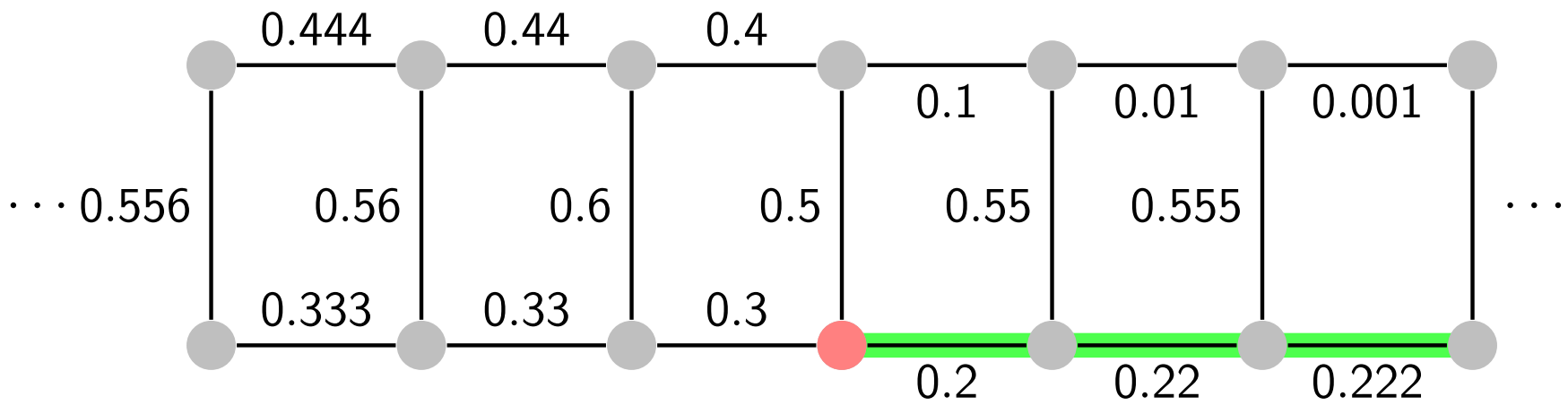
In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.



MSTs of infinite graphs.

The right definition

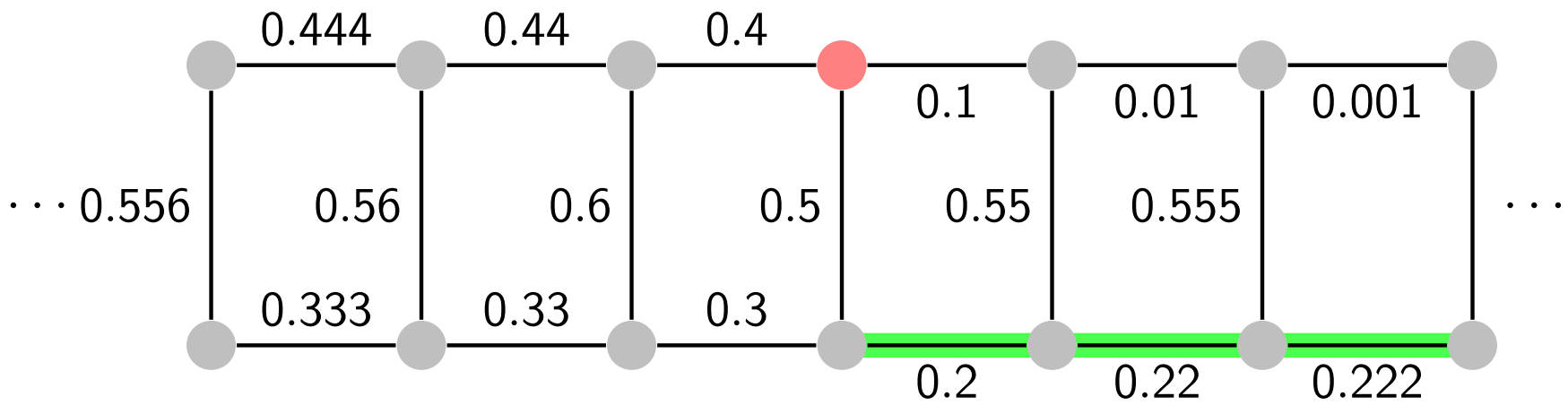
In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.



MSTs of infinite graphs.

The right definition

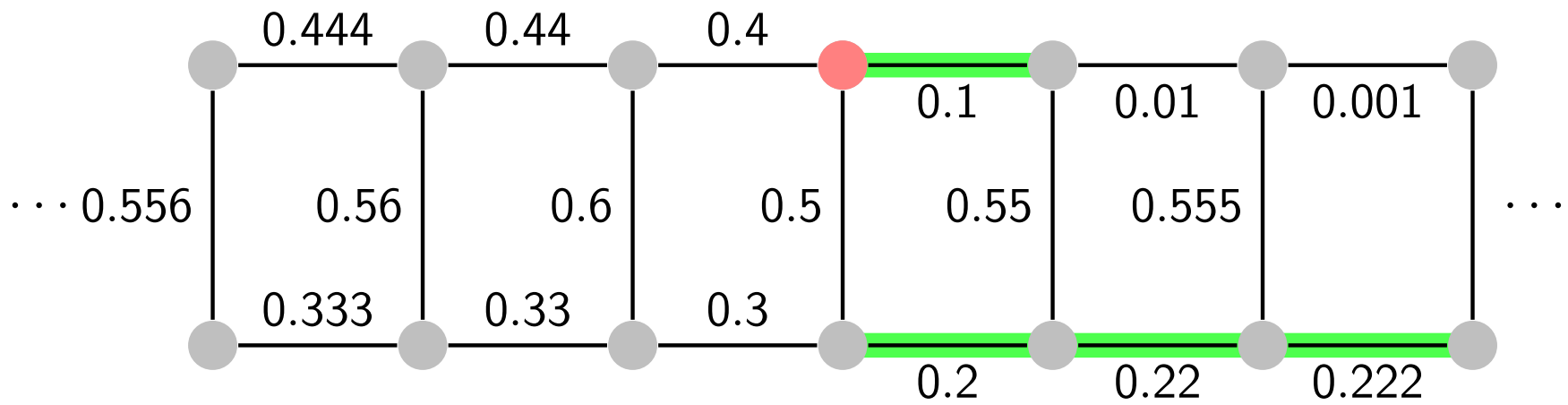
In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.



MSTs of infinite graphs.

The right definition

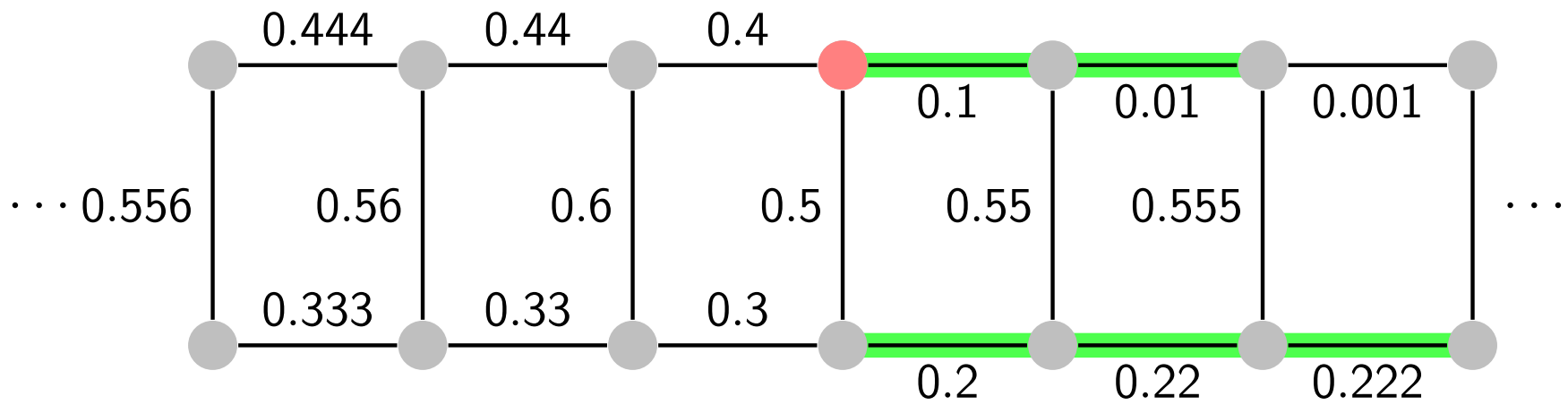
In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.



MSTs of infinite graphs.

The right definition

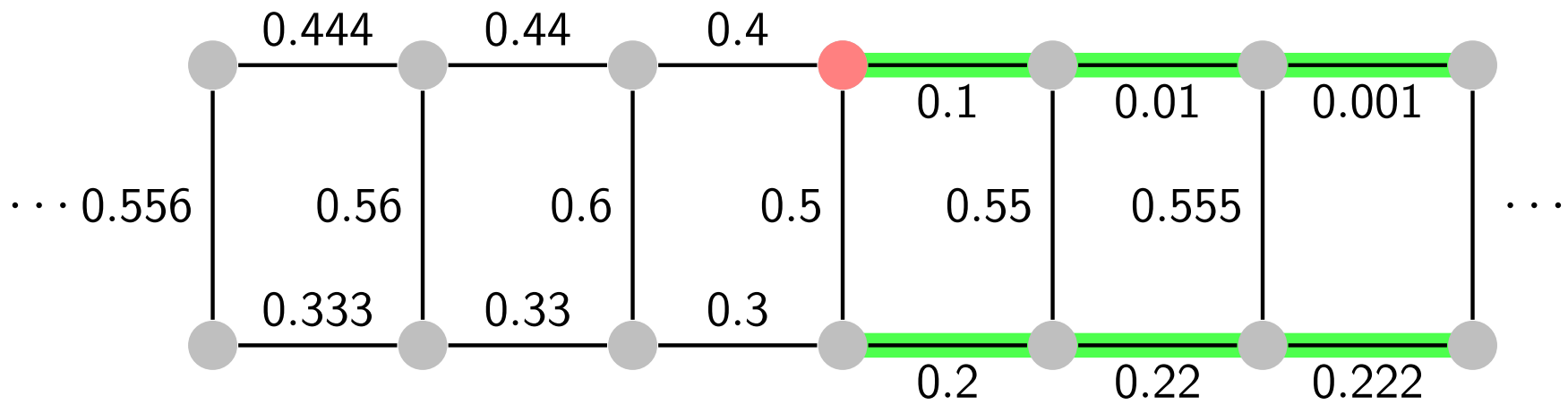
In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.



MSTs of infinite graphs.

The right definition

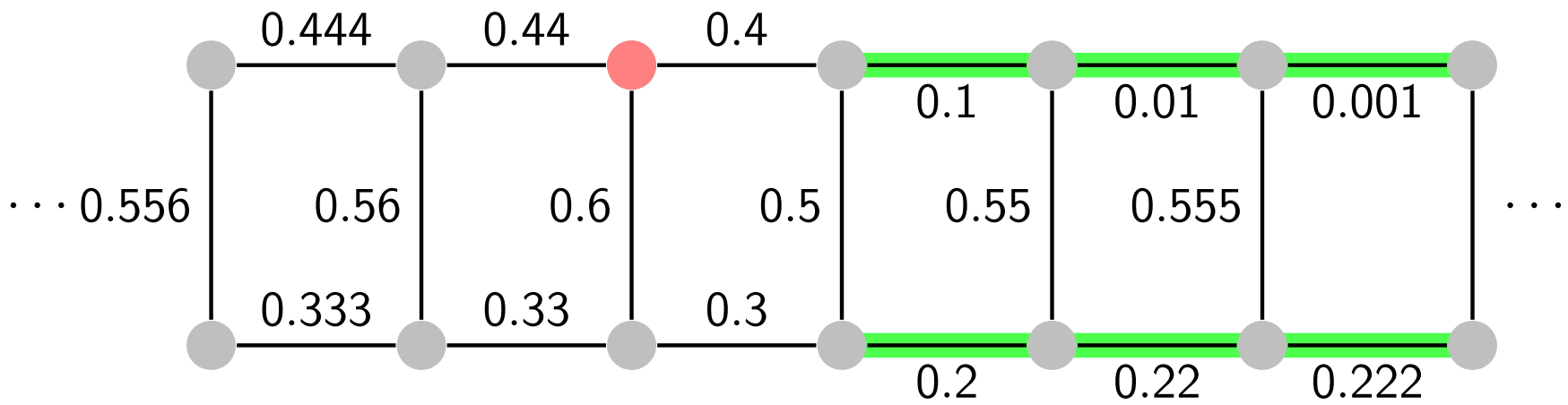
In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.



MSTs of infinite graphs.

The right definition

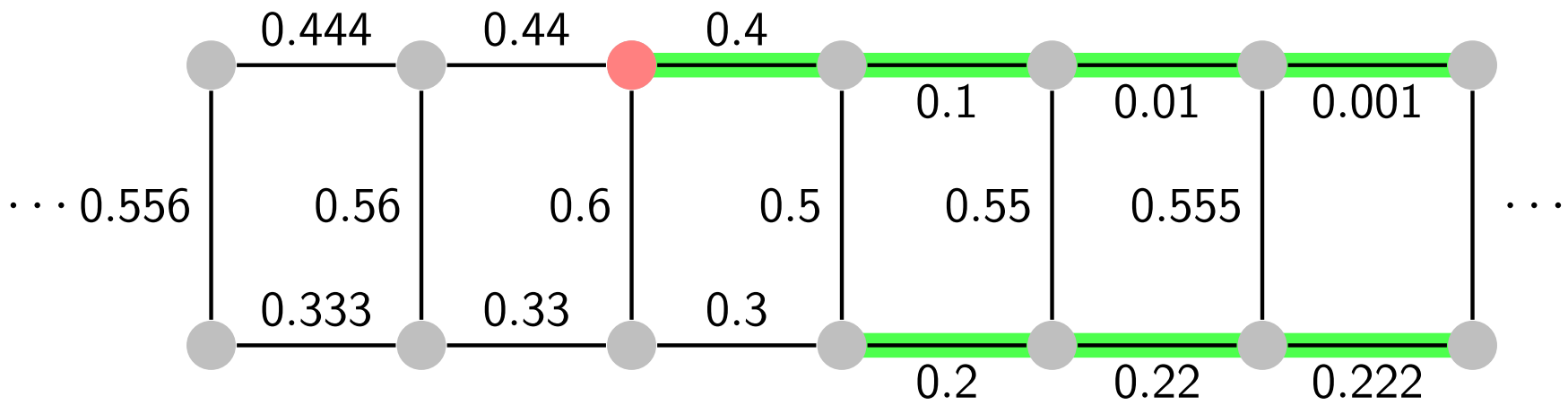
In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.



MSTs of infinite graphs.

The right definition

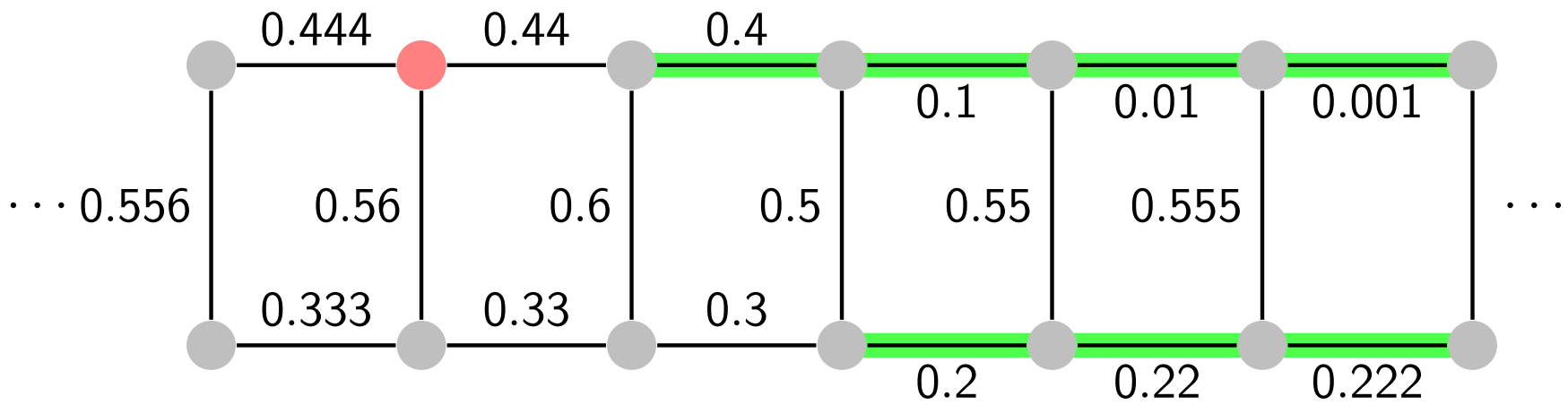
In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.



MSTs of infinite graphs.

The right definition

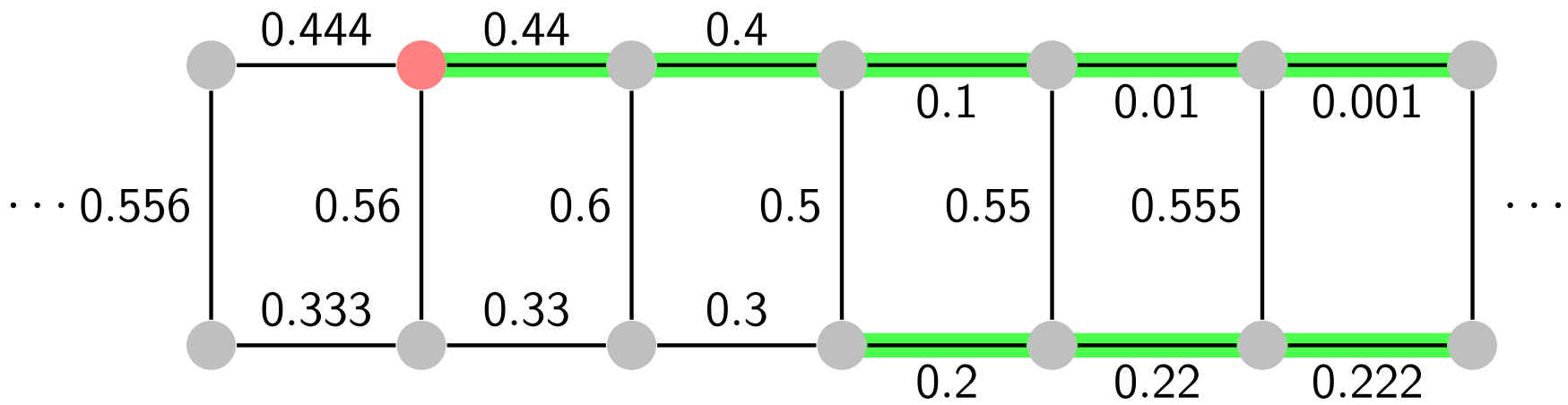
In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.



MSTs of infinite graphs.

The right definition

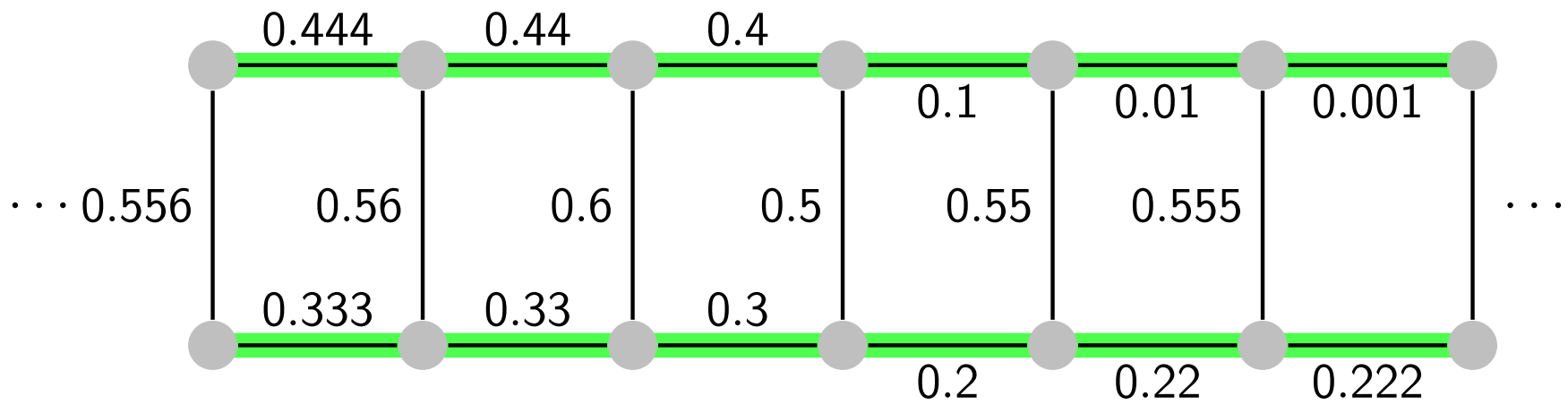
In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.



MSTs of infinite graphs.

The right definition

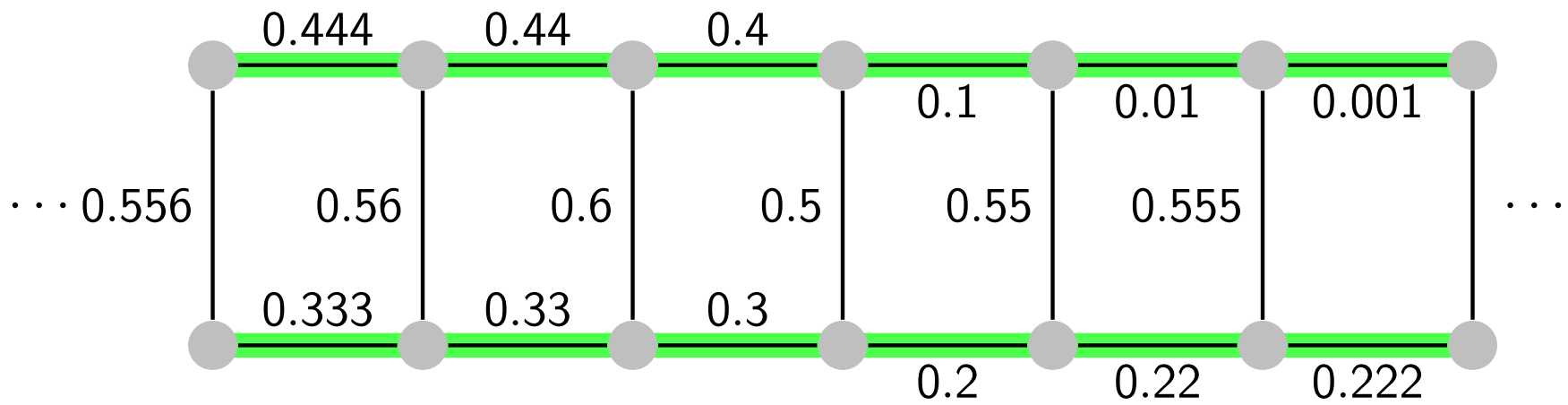
In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.



MSTs of infinite graphs.

The right definition

In percolation-type settings (where graphs are usually infinite and locally finite) the solution is generally to use Prim's algorithm/invasion percolation, started once from each node of the graph.
Then take the union of all the trees that are built.



The result is called the Minimum Spanning Forest.

Due diligence

Lemma (Aldous & Steele, 2004)

For nice graphs, the union of all invasion percolation trees is a forest,*

Due diligence

Lemma (Aldous & Steele, 2004)

For nice graphs, the union of all invasion percolation trees is a forest, each component of which is an infinite tree.*

Due diligence

Lemma (Aldous & Steele, 2004)

For nice graphs, the union of all invasion percolation trees is a forest, each component of which is an infinite tree.*

Due diligence

Lemma (Aldous & Steele, 2004)

For nice graphs, the union of all invasion percolation trees is a forest, each component of which is an infinite tree.*

nice: infinite, locally finite, all edge weights distinct.

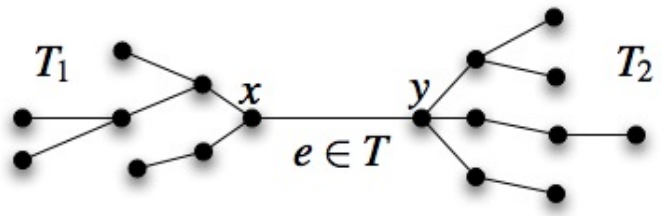
Another way to look at the MSF

For finite G, an edge $e = xy$ is in the MST $\Leftrightarrow$ any path P from x to y contains an edge of weight $\geq w_e$.

Another way to look at the MSF

For finite G, an edge $e = xy$ is in the MST $\Leftrightarrow$ any path P from x to y contains an edge of weight $\geq w_e$.

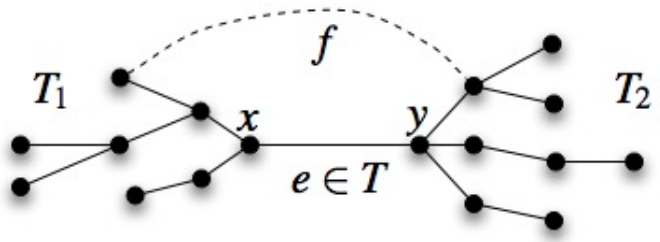
$\forall f$ from T_1 to T_2, $w_f > w_e$.



Another way to look at the MSF

For finite G, an edge $e = xy$ is in the MST $\Leftrightarrow$ any path P from x to y contains an edge of weight $\geq w_e$.

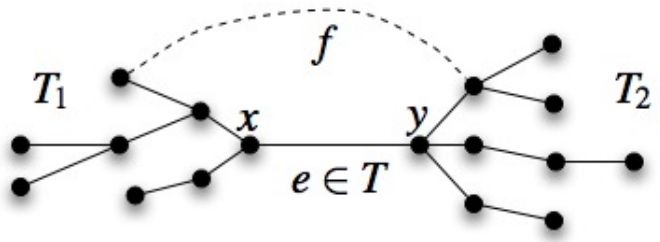
$\forall f$ from T_1 to T_2, $w_f > w_e$.



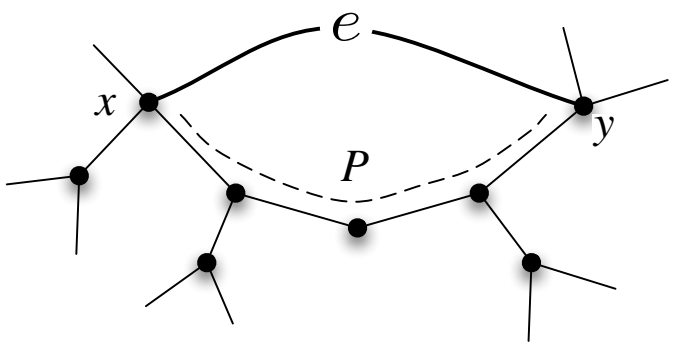
Another way to look at the MSF

For finite G, an edge $e = xy$ is in the MST $\Leftrightarrow$ any path P from x to y contains an edge of weight $\geq w_e$.

$\forall f$ from T_1 to T_2, $w_f > w_e$.



Every edge f of P satisfies $w_f < w_e$



Another way to look at the MSF

Given an infinite weighted graph $G = (V, E)$, with edge weights $\{w_e\}_{e \in E}$.

Another way to look at the MSF

Given an infinite weighted graph $G = (V, E)$, with edge weights $\{w_e\}_{e \in E}$.
For $r \in \mathbb{R}$, write G_r for the subgraph of G containing only edges e with $w_e < r$.

Another way to look at the MSF

Given an infinite weighted graph $G = (V, E)$, with edge weights $\{w_e\}_{e \in E}$.
For $r \in \mathbb{R}$, write G_r for the subgraph of G containing only edges e with $w_e < r$.
Aldous & Steele also prove:

Lemma

For nice graphs G, edge $e = uv$ with weight $w_e = r$ is in the MSF if and only if u and v are in distinct components of G_r, and one of these components is finite.

Another way to look at the MSF

Given an infinite weighted graph $G = (V, E)$, with edge weights $\{w_e\}_{e \in E}$.
For $r \in \mathbb{R}$, write G_r for the subgraph of G containing only edges e with $w_e < r$.
Aldous & Steele also prove:

Lemma

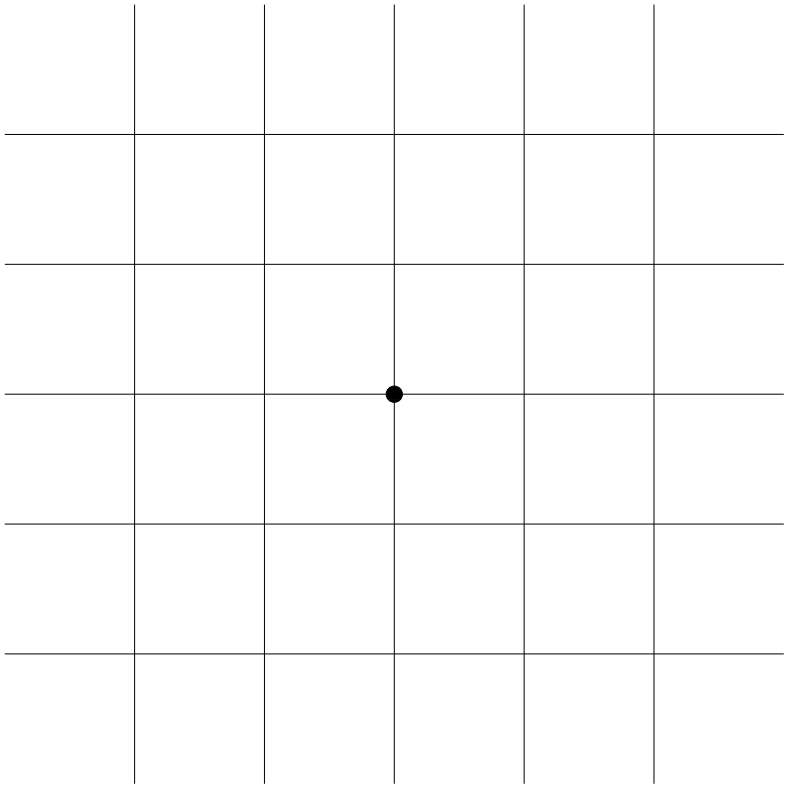
For nice graphs G, edge $e = uv$ with weight $w_e = r$ is in the MSF if and only if u and v are in distinct components of G_r, and one of these components is finite.

Note: in the percolation process, there are never two distinct infinite components!

Next: a very quick refresher on percolation.

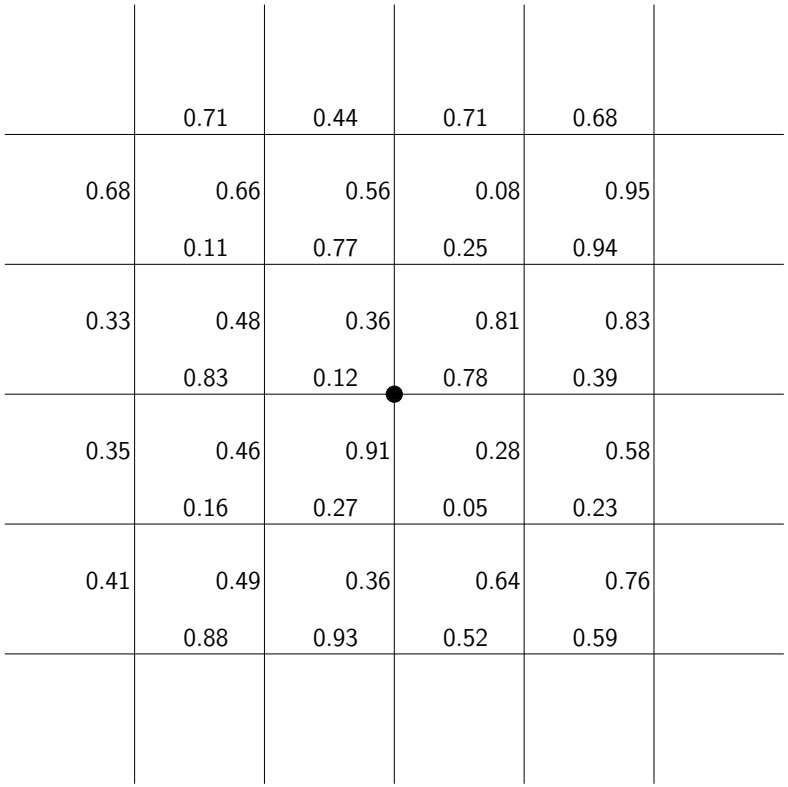
Bernoulli bond percolation

- Example: Nearest neighbour graph on $\mathbb{Z}^2$.



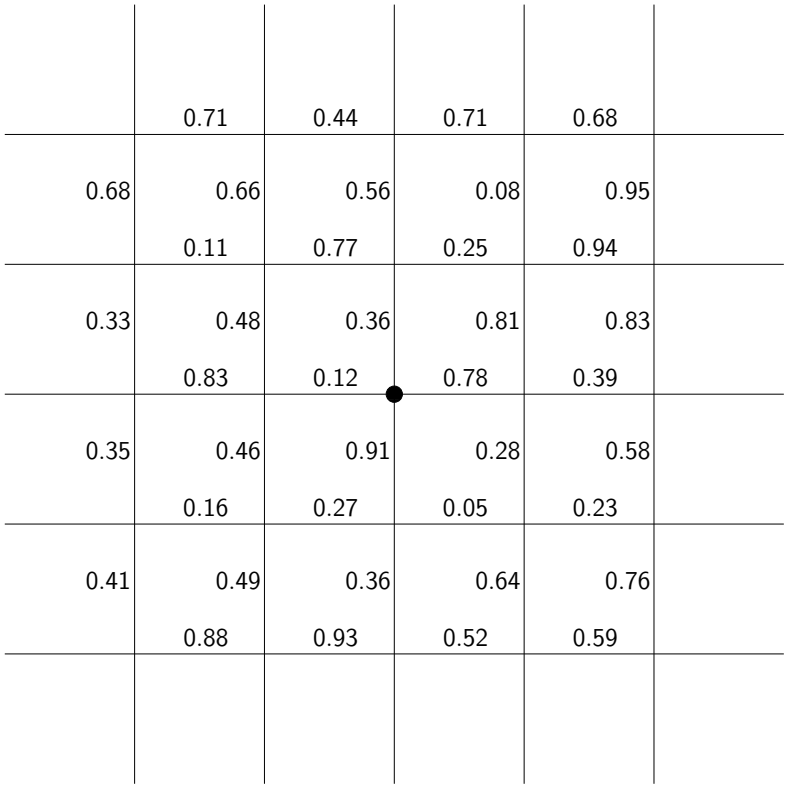
Bernoulli bond percolation

- Example: Nearest neighbour graph on $\mathbb{Z}^2$.
- Edges augmented with i.i.d. uniform $[0, 1]$ r.v.s.



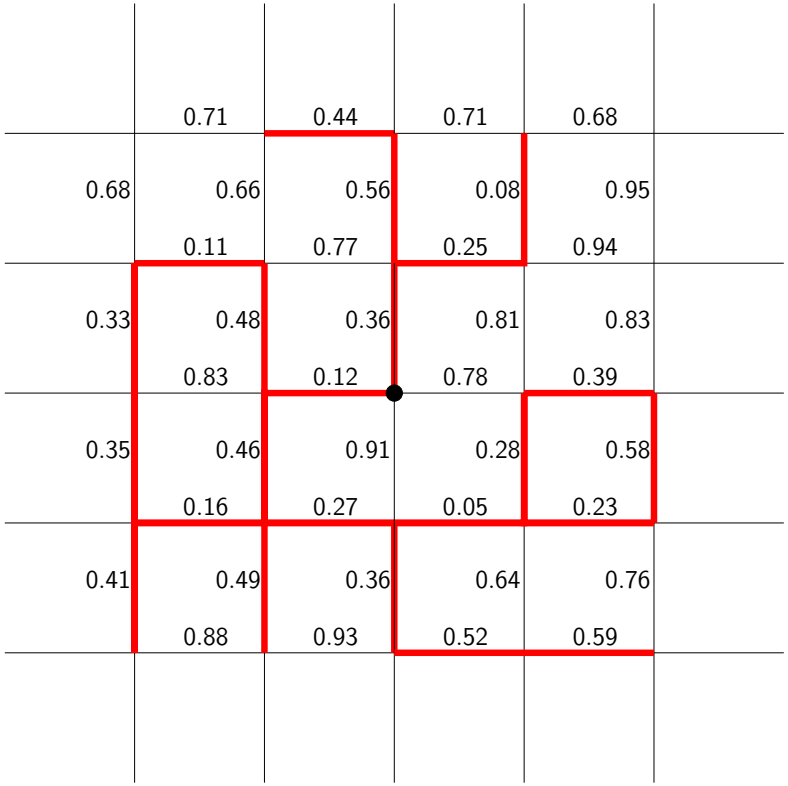
Bernoulli bond percolation

- Example: Nearest neighbour graph on $\mathbb{Z}^2$.
- Edges augmented with i.i.d. uniform $[0, 1]$ r.v.s.
- Fix p, $0 < p < 1$, and keep only edges of weight at most p. (Call the rezult $\mathbb{Z}^2_p$.)



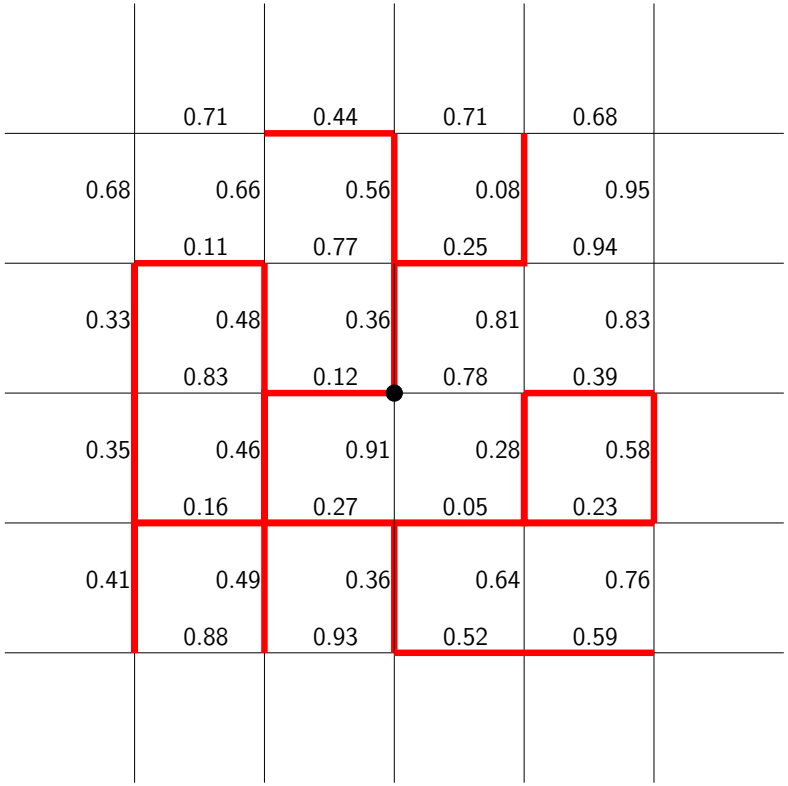
Bernoulli bond percolation

- Example: Nearest neighbour graph on $\mathbb{Z}^2$.
- Edges augmented with i.i.d. uniform $[0, 1]$ r.v.s.
- Fix p, $0 < p < 1$, and keep only edges of weight at most p. (Call the rezult $\mathbb{Z}^2_p$.)



Bernoulli bond percolation

- Example: Nearest neighbour graph on $\mathbb{Z}^2$.
- Edges augmented with i.i.d. uniform $[0, 1]$ r.v.s.
- Fix p, $0 < p < 1$, and keep only edges of weight at most p. (Call the rezult $\mathbb{Z}^2_p$.)
- Here $p = 0.6$.



Bernoulli bond percolation on $\mathbb{Z}^2$

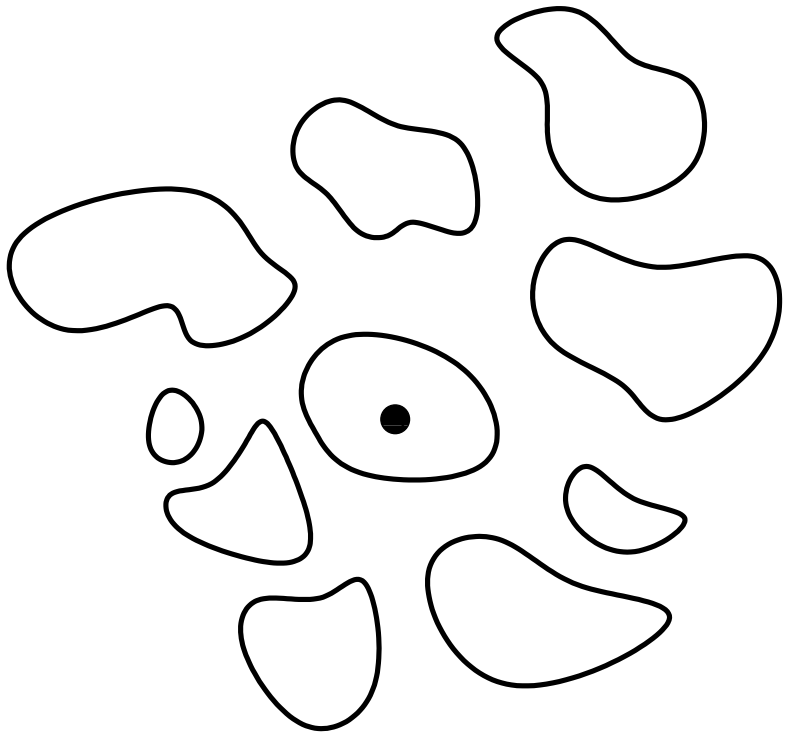
- A phase transition occurs at $p = 1/2$.

Bernoulli bond percolation on $\mathbb{Z}^2$

- A phase transition occurs at $p = 1/2$.
- Write $\mathcal{C}_p$ for the component of $\mathbb{Z}_p^2$ containing the origin.

Bernoulli bond percolation on $\mathbb{Z}^2$

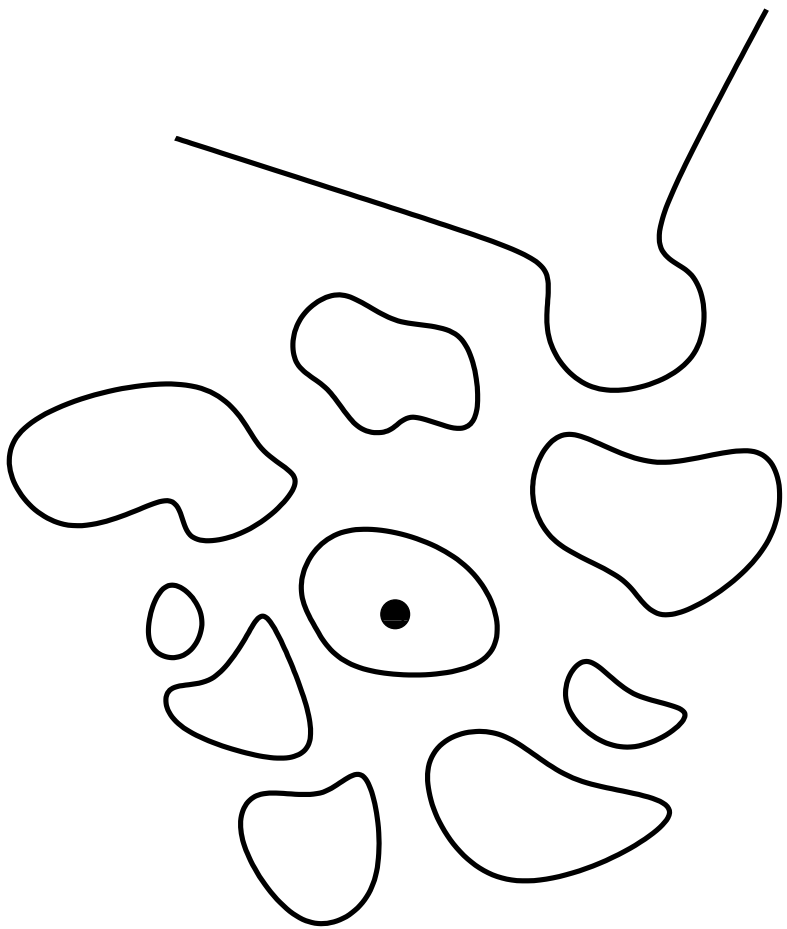
- A phase transition occurs at $p = 1/2$.
- Write $\mathcal{C}_p$ for the component of $\mathbb{Z}_p^2$ containing the origin.
- $p < 1/2$: all components are finite, and $|\mathcal{C}_p|$ has exponential tails. (Subcritical case)



$p < \frac{1}{2}$

Bernoulli bond percolation on $\mathbb{Z}^2$

- A phase transition occurs at $p = 1/2$.
- Write $\mathcal{C}_p$ for the component of $\mathbb{Z}_p^2$ containing the origin.
- $p < 1/2$: all components are finite, and $|\mathcal{C}_p|$ has exponential tails. (Subcritical case)
- $p > 1/2$: with probability 1 there is a unique infinite component. (Supercritical case)

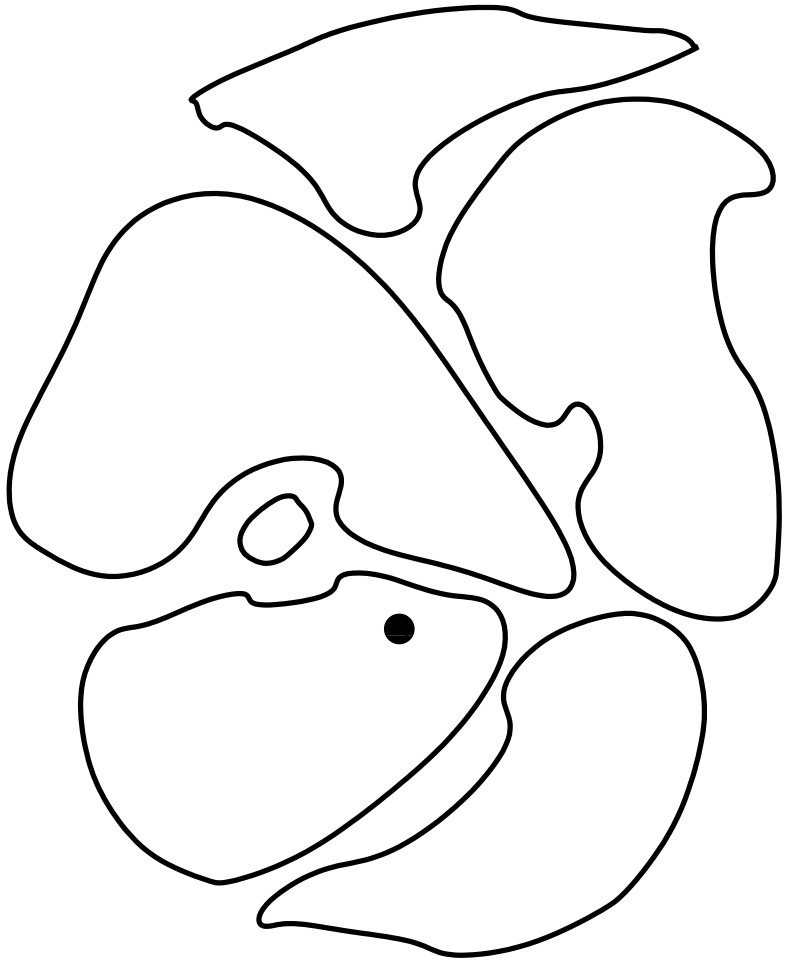


$p > \frac{1}{2}$

Bernoulli bond percolation on $\mathbb{Z}^2$

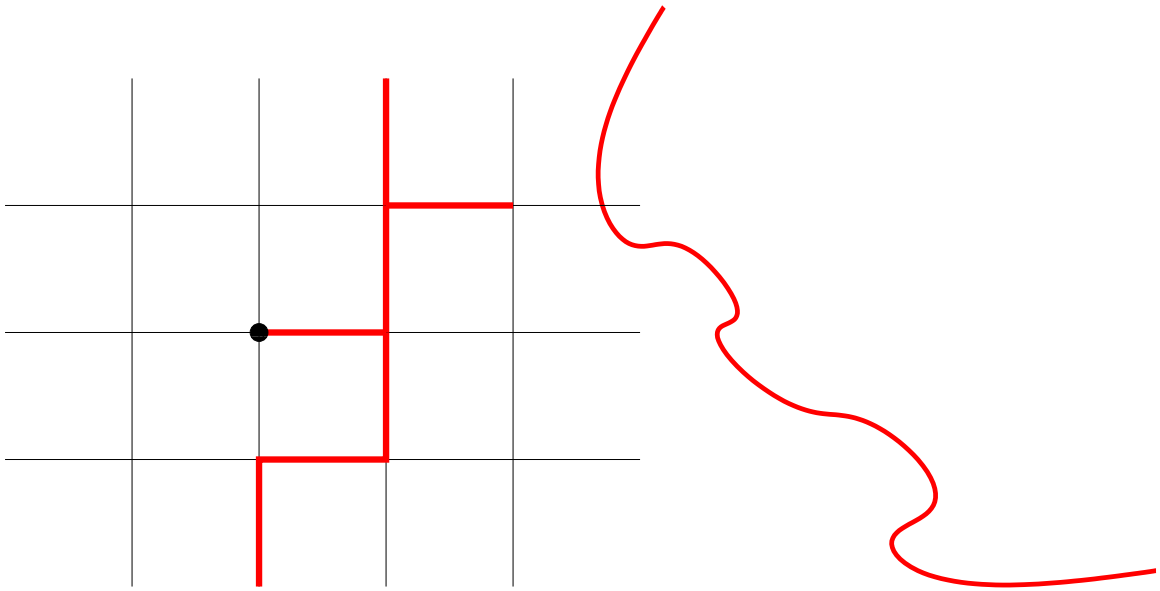
- A phase transition occurs at $p = 1/2$.
- Write $\mathcal{C}_p$ for the component of $\mathbb{Z}_p^2$ containing the origin.
- $p < 1/2$: all components are finite, and $|\mathcal{C}_p|$ has exponential tails. (Subcritical case)
- $p > 1/2$: with probability 1 there is a unique infinite component. (Supercritical case)
- $p = 1/2$: there is no infinite component* but $|\mathcal{C}_p|$ only has polynomial tails.

  (Critical case)



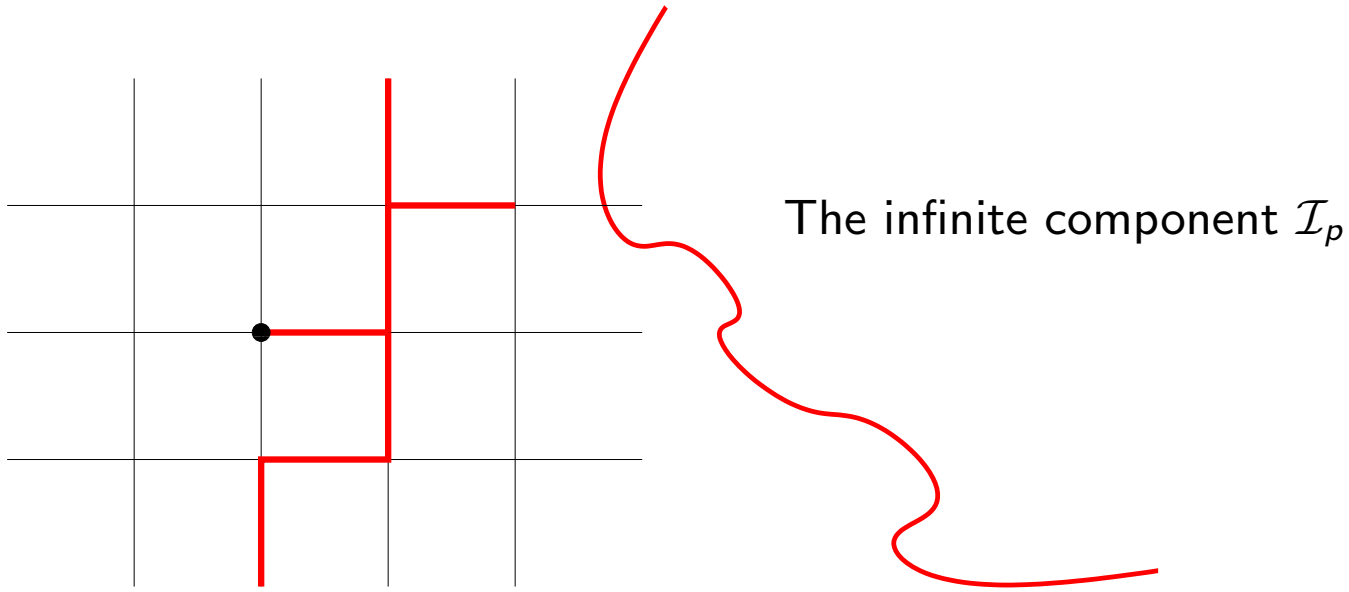
$p = \frac{1}{2}$

Self-organized criticality



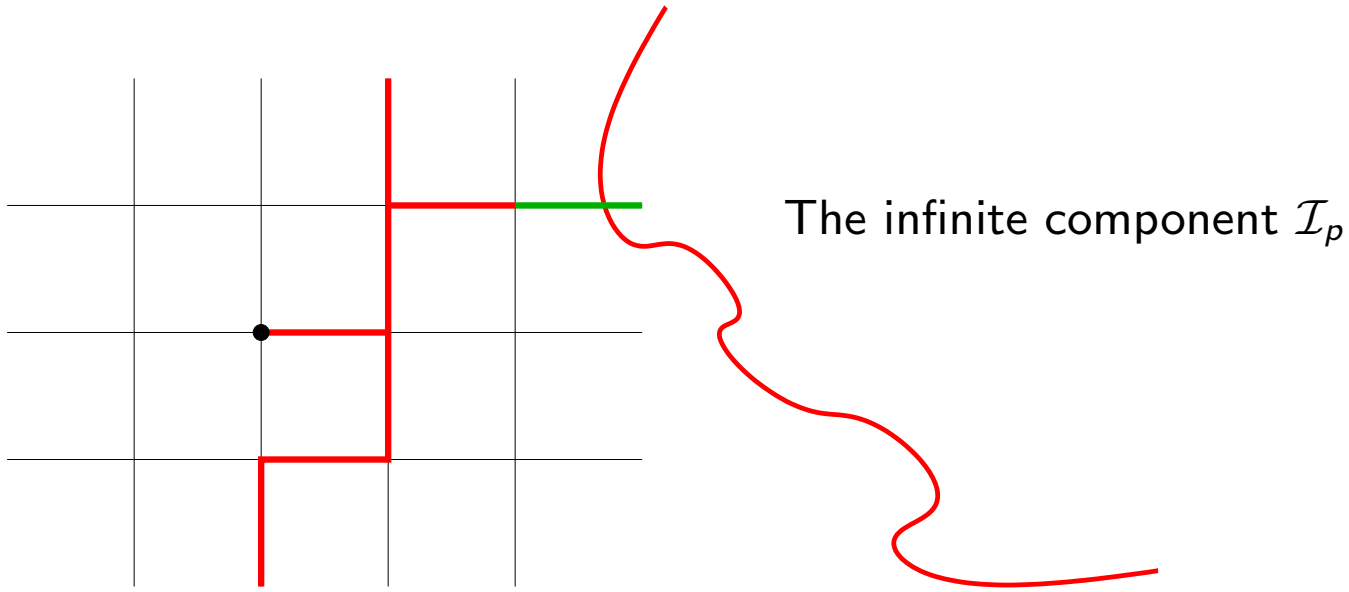
- Fix $p > 1/2$: then $\mathbb{Z}^2_p$, the subgraph of edges of weight at most p, has a unique infinite connected component $\mathcal{I}_p$.

Self-organized criticality



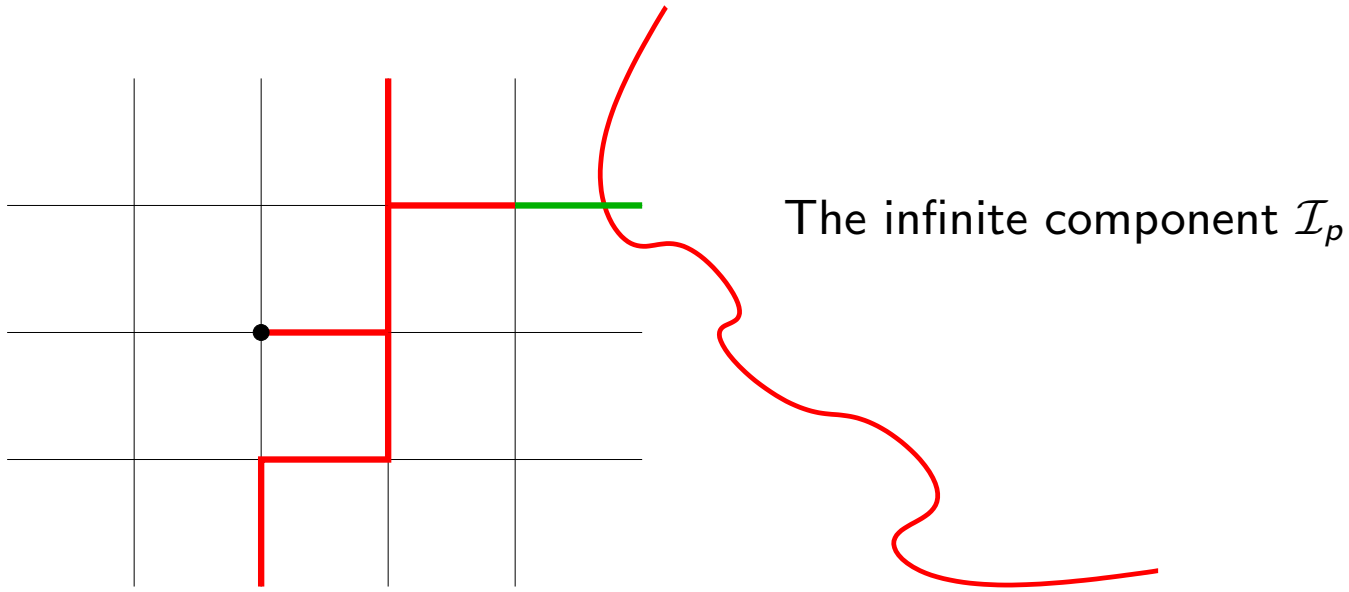
- Fix $p > 1/2$: then $\mathbb{Z}^2_p$, the subgraph of edges of weight at most p, has a unique infinite connected component $\mathcal{I}_p$.

Self-organized criticality



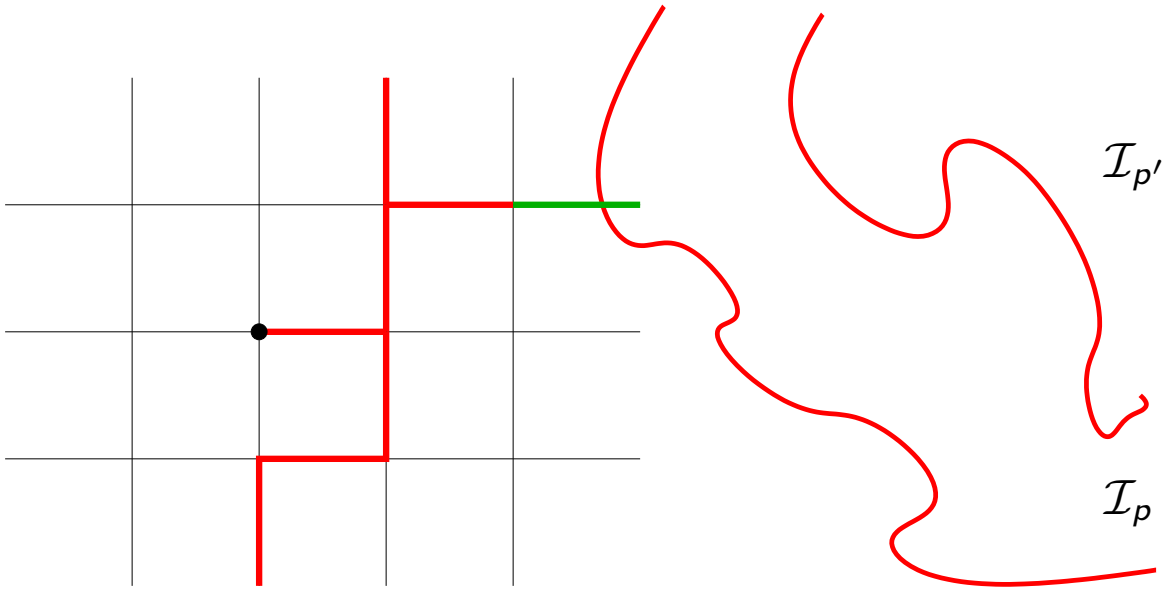
- Fix $p > 1/2$: then $\mathbb{Z}^2_p$, the subgraph of edges of weight at most p, has a unique infinite connected component $\mathcal{I}_p$.
- At some point, invasion percolation adds an edge into $\mathcal{I}_p$.

Self-organized criticality



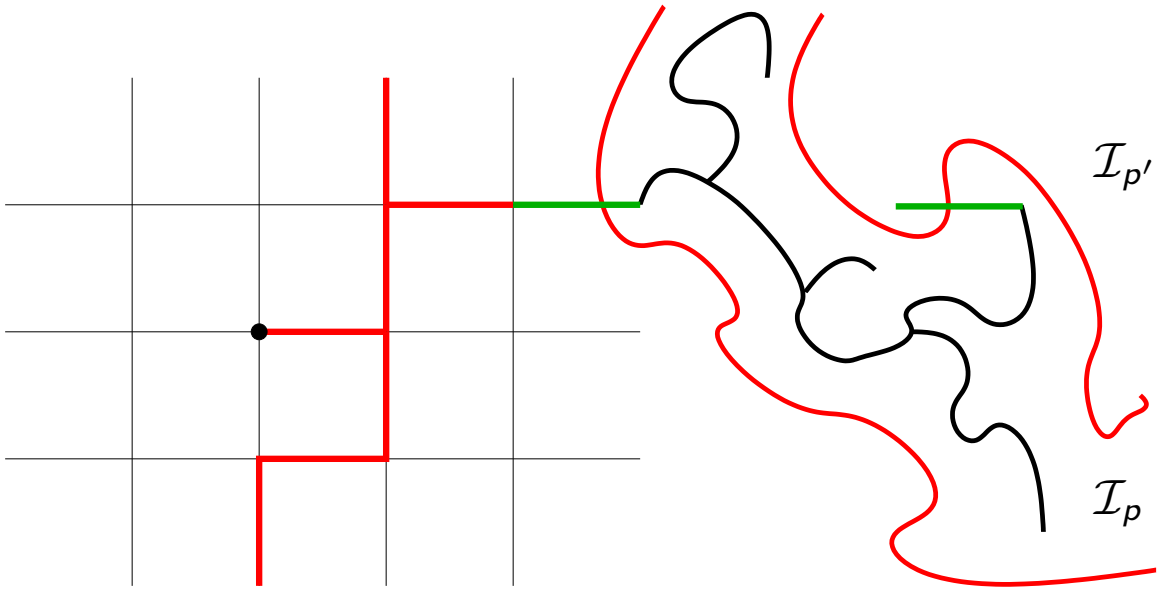
- Fix $p > 1/2$: then $\mathbb{Z}^2_p$, the subgraph of edges of weight at most p, has a unique infinite connected component $\mathcal{I}_p$.
- At some point, invasion percolation adds an edge into $\mathcal{I}_p$.
- After this point, no edge outside of $\mathcal{I}_p$ will ever be explored.

Self-organized criticality



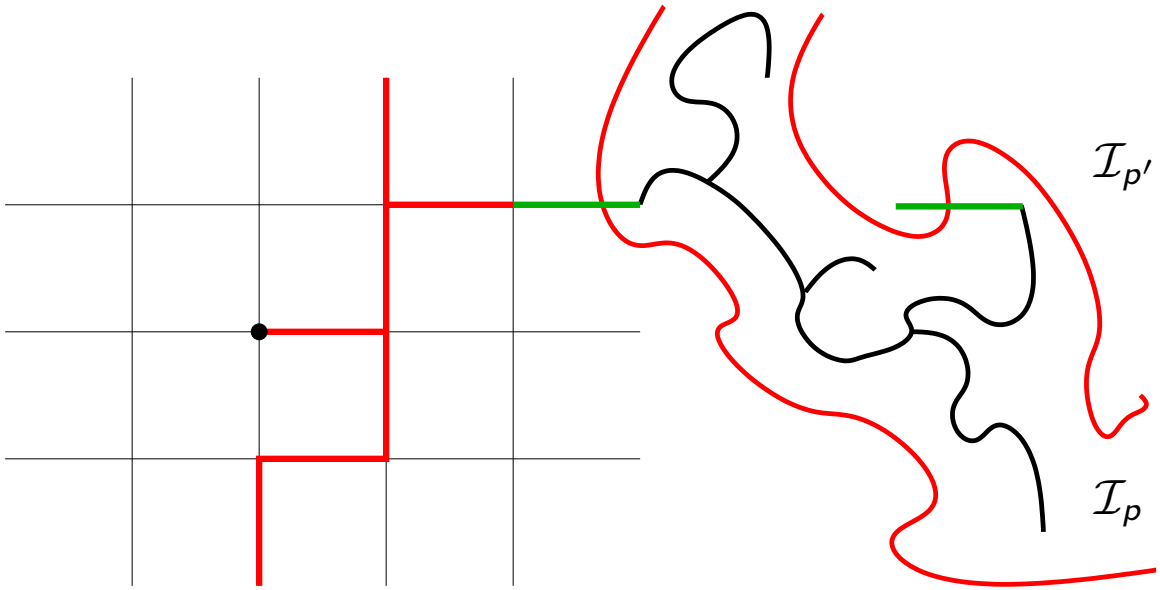
- Now fix p' with $1/2 < p' < p$.

Self-organized criticality



- Now fix p' with $1/2 < p' < p$.
- At some point, invasion percolation adds an edge into $\mathcal{I}_{p'}$.

Self-organized criticality



- Now fix p' with $1/2 < p' < p$.
- At some point, invasion percolation adds an edge into $\mathcal{I}_{p'}$.
- Invasion percolation constructs a tree with a unique infinite path that "zooms in on" the near-critical infinite Bernoulli percolation cluster. ("Self-organized criticality")

A beautiful theorem

Theorem (Chuck Newman, 1995)

The invasion percolation tree on $\mathbb{Z}^d$ a.s. has zero density if and only if

$$\mathbf{P}\{\textit{Critical percolation has an infinite cluster}\} = 0.$$

A beautiful theorem

Theorem (Chuck Newman, 1995)

The invasion percolation tree on $\mathbb{Z}^d$ a.s. has zero density if and only if

$$\mathbf{P}\{\textit{Critical percolation has an infinite cluster}\} = 0.$$

Gives one possible approach to proving that there is no percolation at criticality.

Critical dimensions

A heuristic: for many statistical physical models, there is believed to be an *upper critical dimension*, above which "mean-field" behaviour applies.

Critical dimensions

A heuristic: for many statistical physical models, there is believed to be an *upper critical dimension*, above which "mean-field" behaviour applies.
Example: self-avoiding walk.

Critical dimensions

A heuristic: for many statistical physical models, there is believed to be an *upper critical dimension*, above which "mean-field" behaviour applies.

Example: self-avoiding walk.

For $d > 4$, self-avoiding walk behaves like Brownian motion (Hara & Slade, 1991).

Critical dimensions

A heuristic: for many statistical physical models, there is believed to be an *upper critical dimension*, above which "mean-field" behaviour applies.

Example: self-avoiding walk.

For $d > 4$, self-avoiding walk behaves like Brownian motion (Hara & Slade, 1991).

Reason:

Critical dimensions

A heuristic: for many statistical physical models, there is believed to be an *upper critical dimension*, above which "mean-field" behaviour applies.
Example: self-avoiding walk.
For $d > 4$, self-avoiding walk behaves like Brownian motion (Hara & Slade, 1991).
Reason:

- Brownian motions are 2-dimensional (Hausdorff dimension 2).

Critical dimensions

A heuristic: for many statistical physical models, there is believed to be an *upper critical dimension*, above which "mean-field" behaviour applies.
Example: self-avoiding walk.
For $d > 4$, self-avoiding walk behaves like Brownian motion (Hara & Slade, 1991).
Reason:

- Brownian motions are 2-dimensional (Hausdorff dimension 2).

Critical dimensions

A heuristic: for many statistical physical models, there is believed to be an *upper critical dimension*, above which "mean-field" behaviour applies.
Example: self-avoiding walk.
For $d > 4$, self-avoiding walk behaves like Brownian motion (Hara & Slade, 1991).
Reason:

- Brownian motions are 2-dimensional (Hausdorff dimension 2).
- When $d > 4$, two independent 2-dimensional objects "generically" do not meet.

Critical dimensions

A heuristic: for many statistical physical models, there is believed to be an *upper critical dimension*, above which "mean-field" behaviour applies.
Example: self-avoiding walk.
For $d > 4$, self-avoiding walk behaves like Brownian motion (Hara & Slade, 1991).
Reason:

- Brownian motions are 2-dimensional (Hausdorff dimension 2).
- When $d > 4$, two independent 2-dimensional objects "generically" do not meet.

Critical dimensions

A heuristic: for many statistical physical models, there is believed to be an *upper critical dimension*, above which "mean-field" behaviour applies.
Example: self-avoiding walk.
For $d > 4$, self-avoiding walk behaves like Brownian motion (Hara & Slade, 1991).
Reason:

- Brownian motions are 2-dimensional (Hausdorff dimension 2).
- When $d > 4$, two independent 2-dimensional objects "generically" do not meet.
- Thus, for $d > 4$ the effects of "self-interaction" between first and second halves of an n-step self-avoiding walk are negligible.

Mean-field behaviour for percolation

- In high dimensions, critical percolation on $\mathbb{Z}^d$ (or on a d-dimensional torus) should take its mean-field behaviour.

Mean-field behaviour for percolation

- In high dimensions, critical percolation on $\mathbb{Z}^d$ (or on a d-dimensional torus) should take its mean-field behaviour.

Mean-field behaviour for percolation

- In high dimensions, critical percolation on $\mathbb{Z}^d$ (or on a d-dimensional torus) should take its mean-field behaviour.
- One candidate for what this means: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by critical percolation on the $2d$-ary tree (Bethe lattice).

Mean-field behaviour for percolation

- In high dimensions, critical percolation on $\mathbb{Z}^d$ (or on a d-dimensional torus) should take its mean-field behaviour.
- One candidate for what this means: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by critical percolation on the $2d$-ary tree (Bethe lattice).

Mean-field behaviour for percolation

- In high dimensions, critical percolation on $\mathbb{Z}^d$ (or on a d-dimensional torus) should take its mean-field behaviour.
- One candidate for what this means: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by critical percolation on the $2d$-ary tree (Bethe lattice).
- This is just a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.

Mean-field behaviour for percolation

- In high dimensions, critical percolation on $\mathbb{Z}^d$ (or on a d-dimensional torus) should take its mean-field behaviour.
- One candidate for what this means: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by critical percolation on the $2d$-ary tree (Bethe lattice).
- This is just a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.

Mean-field behaviour for percolation

- In high dimensions, critical percolation on $\mathbb{Z}^d$ (or on a d-dimensional torus) should take its mean-field behaviour.
- One candidate for what this means: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by critical percolation on the $2d$-ary tree (Bethe lattice).
- This is just a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.
- A critical Galton-Watson tree doesn't encode *spatial* information about its vertices.

Mean-field behaviour for percolation

- In high dimensions, critical percolation on $\mathbb{Z}^d$ (or on a d-dimensional torus) should take its mean-field behaviour.
- One candidate for what this means: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by critical percolation on the $2d$-ary tree (Bethe lattice).
- This is just a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.
- A critical Galton-Watson tree doesn't encode *spatial* information about its vertices.

Mean-field behaviour for percolation

- In high dimensions, critical percolation on $\mathbb{Z}^d$ (or on a d-dimensional torus) should take its mean-field behaviour.
- One candidate for what this means: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by critical percolation on the $2d$-ary tree (Bethe lattice).
- This is just a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.
- A critical Galton-Watson tree doesn't encode *spatial* information about its vertices. For this, should instead use a critical branching random walk or branching Brownian motion.

Mean-field behaviour for percolation

- In high dimensions, critical percolation on $\mathbb{Z}^d$ (or on a d-dimensional torus) should take its mean-field behaviour.
- One candidate for what this means: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by critical percolation on the $2d$-ary tree (Bethe lattice).
- This is just a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.
- A critical Galton-Watson tree doesn't encode *spatial* information about its vertices. For this, should instead use a critical branching random walk or branching Brownian motion.
- Branching Brownian motion is a 4-dimensional object.

Mean-field behaviour for percolation

- In high dimensions, critical percolation on $\mathbb{Z}^d$ (or on a d-dimensional torus) should take its mean-field behaviour.
- One candidate for what this means: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by critical percolation on the $2d$-ary tree (Bethe lattice).
- This is just a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.
- A critical Galton-Watson tree doesn't encode *spatial* information about its vertices. For this, should instead use a critical branching random walk or branching Brownian motion.
- Branching Brownian motion is a 4-dimensional object.

Mean-field behaviour for percolation

- In high dimensions, critical percolation on $\mathbb{Z}^d$ (or on a d-dimensional torus) should take its mean-field behaviour.
- One candidate for what this means: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by critical percolation on the $2d$-ary tree (Bethe lattice).
- This is just a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.
- A critical Galton-Watson tree doesn't encode *spatial* information about its vertices. For this, should instead use a critical branching random walk or branching Brownian motion.
- Branching Brownian motion is a 4-dimensional object.
- Theorem (Angel et. al. 2006): Invasion percolation on the $2d$-ary tree is 4-dimensional.

Predictions

Recall: the *minimum spanning forest* of the percolation process is the union of all invasion percolation clusters.

Predictions

Recall: the *minimum spanning forest* of the percolation process is the union of all invasion percolation clusters.
Here is a heuristic argument of Newman & Stein (1994):

- For $d > 6$, the invasion percolation tree should take its mean-field behaviour and so be a four-dimensional object.

Predictions

Recall: the *minimum spanning forest* of the percolation process is the union of all invasion percolation clusters.
Here is a heuristic argument of Newman & Stein (1994):

- For $d > 6$, the invasion percolation tree should take its mean-field behaviour and so be a four-dimensional object.

Predictions

Recall: the *minimum spanning forest* of the percolation process is the union of all invasion percolation clusters.
Here is a heuristic argument of Newman & Stein (1994):

- For $d > 6$, the invasion percolation tree should take its mean-field behaviour and so be a four-dimensional object.
- Generically, four-dimensional objects do not intersect in dimensions $d > 8$.

Predictions

Recall: the *minimum spanning forest* of the percolation process is the union of all invasion percolation clusters.
Here is a heuristic argument of Newman & Stein (1994):

- For $d > 6$, the invasion percolation tree should take its mean-field behaviour and so be a four-dimensional object.
- Generically, four-dimensional objects do not intersect in dimensions $d > 8$.

Predictions

Recall: the *minimum spanning forest* of the percolation process is the union of all invasion percolation clusters.
Here is a heuristic argument of Newman & Stein (1994):

- For $d > 6$, the invasion percolation tree should take its mean-field behaviour and so be a four-dimensional object.
- Generically, four-dimensional objects do not intersect in dimensions $d > 8$.
- Thus, the minimum spanning forest (MSF) should be a tree in dimensions $d < 8$, but have infinitely many components in dimensions $d > 8$.

Predictions

Recall: the *minimum spanning forest* of the percolation process is the union of all invasion percolation clusters.
Here is a heuristic argument of Newman & Stein (1994):

- For $d > 6$, the invasion percolation tree should take its mean-field behaviour and so be a four-dimensional object.
- Generically, four-dimensional objects do not intersect in dimensions $d > 8$.
- Thus, the minimum spanning forest (MSF) should be a tree in dimensions $d < 8$, but have infinitely many components in dimensions $d > 8$.

Predictions

Recall: the *minimum spanning forest* of the percolation process is the union of all invasion percolation clusters.
Here is a heuristic argument of Newman & Stein (1994):

- For $d > 6$, the invasion percolation tree should take its mean-field behaviour and so be a four-dimensional object.
- Generically, four-dimensional objects do not intersect in dimensions $d > 8$.
- Thus, the minimum spanning forest (MSF) should be a tree in dimensions $d < 8$, but have infinitely many components in dimensions $d > 8$.
- No one can prove anything at all about this, except when $d = 2$.

Predictions

Recall: the *minimum spanning forest* of the percolation process is the union of all invasion percolation clusters.

- For $d = 2$, Alexander & Molchanov (1994) used duality to show that the MSF is indeed a tree on the square, triangular, and hexagonal lattices.

Predictions

Recall: the *minimum spanning forest* of the percolation process is the union of all invasion percolation clusters.

- For $d = 2$, Alexander & Molchanov (1994) used duality to show that the MSF is indeed a tree on the square, triangular, and hexagonal lattices.
- Bollobas & Riordan (2006): The critical probability for Voronoi percolation in the plane is $1/2$.

Predictions

Recall: the *minimum spanning forest* of the percolation process is the union of all invasion percolation clusters.

- For $d = 2$, Alexander & Molchanov (1994) used duality to show that the MSF is indeed a tree on the square, triangular, and hexagonal lattices.
- Bollobas & Riordan (2006): The critical probability for Voronoi percolation in the plane is $1/2$.

Predictions

Recall: the *minimum spanning forest* of the percolation process is the union of all invasion percolation clusters.

- For $d = 2$, Alexander & Molchanov (1994) used duality to show that the MSF is indeed a tree on the square, triangular, and hexagonal lattices.
- Bollobas & Riordan (2006): The critical probability for Voronoi percolation in the plane is $1/2$.
- Zvavitch (1996): Voronoi percolation at $p = 1/2$ doesn't percolate.

Predictions

Recall: the *minimum spanning forest* of the percolation process is the union of all invasion percolation clusters.

- For $d = 2$, Alexander & Molchanov (1994) used duality to show that the MSF is indeed a tree on the square, triangular, and hexagonal lattices.
- Bollobas & Riordan (2006): The critical probability for Voronoi percolation in the plane is $1/2$.
- Zvavitch (1996): Voronoi percolation at $p = 1/2$ doesn't percolate.

Predictions

Recall: the *minimum spanning forest* of the percolation process is the union of all invasion percolation clusters.

- For $d = 2$, Alexander & Molchanov (1994) used duality to show that the MSF is indeed a tree on the square, triangular, and hexagonal lattices.
- Bollobas & Riordan (2006): The critical probability for Voronoi percolation in the plane is $1/2$.
- Zvavitch (1996): Voronoi percolation at $p = 1/2$ doesn't percolate.
- These facts allow Alexander & Molchanov's proof to be extended to two-dimensional Euclidean MSF.

Predictions

Recall: the *minimum spanning forest* of the percolation process is the union of all invasion percolation clusters.

- For $d = 2$, Alexander & Molchanov (1994) used duality to show that the MSF is indeed a tree on the square, triangular, and hexagonal lattices.
- Bollobas & Riordan (2006): The critical probability for Voronoi percolation in the plane is $1/2$.
- Zvavitch (1996): Voronoi percolation at $p = 1/2$ doesn't percolate.
- These facts allow Alexander & Molchanov's proof to be extended to two-dimensional Euclidean MSF.
- (In fact, in two dimensions the Euclidean MSF was already known to be a tree (Alexander 1995).)

Another connection with critical percolation

Prediction: the minimum spanning forest (MSF) should be a tree in dimensions $d < 8$, but have infinitely many components in dimensions $d > 8$.

Another connection with critical percolation

Prediction: the minimum spanning forest (MSF) should be a tree in dimensions $d < 8$, but have infinitely many components in dimensions $d > 8$.

Fact

The MSF is a tree whenever $\theta(p_c) > 0$.

Another connection with critical percolation

Prediction: the minimum spanning forest (MSF) should be a tree in dimensions $d < 8$, but have infinitely many components in dimensions $d > 8$.

Fact

The MSF is a tree whenever $\theta(p_c) > 0$.

Another connection with critical percolation

Prediction: the minimum spanning forest (MSF) should be a tree in dimensions $d < 8$, but have infinitely many components in dimensions $d > 8$.

Fact

The MSF is a tree whenever $\theta(p_c) > 0$.

This gives another possible approach to (re-)proving that $\theta(p_c) = 0$ for $d > 8$.

Another heuristic

Heuristic: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.

Another heuristic

Heuristic: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.

- But trees are not very useful for understanding the *cycle structure* of critical percolation.

Another heuristic

Heuristic: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.

- But trees are not very useful for understanding the *cycle structure* of critical percolation.

Another heuristic

Heuristic: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.

- But trees are not very useful for understanding the *cycle structure* of critical percolation.
- Understanding the geometry of the cycles is key to understanding the structure of the MST.

Another heuristic

Heuristic: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.

- But trees are not very useful for understanding the *cycle structure* of critical percolation.
- Understanding the geometry of the cycles is key to understanding the structure of the MST.

Another heuristic

Heuristic: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.

- But trees are not very useful for understanding the *cycle structure* of critical percolation.
- Understanding the geometry of the cycles is key to understanding the structure of the MST.
- Another mean-field candidate: the critical random graph $G_{n,1/n}$.

Another heuristic

Heuristic: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.

- But trees are not very useful for understanding the *cycle structure* of critical percolation.
- Understanding the geometry of the cycles is key to understanding the structure of the MST.
- Another mean-field candidate: the critical random graph $G_{n,1/n}$.

Another heuristic

Heuristic: structure of critical percolation on $\mathbb{Z}^d$ well-modelled by a critical Galton-Watson tree with branching distribution $\mathrm{Bin}(2d, 1/2d)$.

- But trees are not very useful for understanding the *cycle structure* of critical percolation.
- Understanding the geometry of the cycles is key to understanding the structure of the MST.
- Another mean-field candidate: the critical random graph $G_{n,1/n}$.
- This is locally well-modelled by a critical Binomial Galton-Watson tree, but has sparse additional long-range connections (cycles).

Next lecture

Explain the global limiting structure of the critical random graph $G_{n,1/n}$.

The last slide, I promise

I like to end with an open problem.

The last slide, I promise

I like to end with an open problem.
The MST of a $\sqrt{n}$ by $\sqrt{n}$ grid (with independent uniform edge weights) has diameter at least $\sqrt{n}$ (have to get from bottom to top).

The last slide, I promise

I like to end with an open problem.
The MST of a $\sqrt{n}$ by $\sqrt{n}$ grid (with independent uniform edge weights) has diameter at least $\sqrt{n}$ (have to get from bottom to top).
It also has diameter $O(n)$ (there are only n nodes).

The last slide, I promise

I like to end with an open problem.
The MST of a $\sqrt{n}$ by $\sqrt{n}$ grid (with independent uniform edge weights) has diameter at least $\sqrt{n}$ (have to get from bottom to top).
It also has diameter $O(n)$ (there are only n nodes).

Open Problem

Improve either of these bounds.

The last slide, I promise

I like to end with an open problem.
The MST of a $\sqrt{n}$ by $\sqrt{n}$ grid (with independent uniform edge weights) has diameter at least $\sqrt{n}$ (have to get from bottom to top).
It also has diameter $O(n)$ (there are only n nodes).

Open Problem

Improve either of these bounds.

If you want to work on this with me, find me at tea!