Randomized Algorithms

Ioannis Caragiannis (this time) and Kasper Green Larsen

Practical issues

- When: Tuesdays, 8-11am
- Where: 5510-104 Lille Auditorium
- 3 projects in groups of three
- Oral exam

Grading rules:

- To pass the course, you need to pass the projects and the oral exam
- The projects can affect your grade in the oral exam by one point (up or down)

This lecture

- Quick recap: randomization, useful inequalities
- Computing a (global) cut of minimum size in a graph
- A randomized algorithm
- Analysis
- Implications and improvements

Randomization

The elephant in the room

- Randomized algorithms use **random coins**, **dice**, **card shuffling**, etc



- For example, the code of a randomized algorithm implementation will typically have a line like this:
- `if (coin_toss() == HEADS) {…}`

Usual assumptions

Basic operation:

- Access to **fair coins** (Pr[HEADS]=Pr[TAILS]=1/2)

More complicated operations:

- **Random selection** among a finite set of items
- Access to a **random permutation** of elements
- Selection of a **random point** in the interval [0,1]
- Selection from a finite or infinite set according to a **non-uniform probability distribution**

Note: there are important **implementation issues** that we most of the time ignore

Main characteristics

Deterministic algorithms: performs the very **same steps** in **any execution** on the **same input**

Randomized algorithms do not!

- They may produce **different outputs** in different executions
- Their **running time** may not always be the same
- In other words, their output, their running time, the amount of space they use are **random variables**

With the **analysis of randomized algorithms**, our aim is to understand these random variables

Randomized algorithms: Why do we want them?

- Sometimes randomization is absolutely **necessary**
- They are usually **simple**
- Work **well on average** or **with high probability**
- Sometimes, the give insights to the design of better deterministc algorithms (**derandomization**)

Analysis tools

Useful (in)equalities

- Linearity of expectation
- Markov’s inequality
- Chernoff bounds (sharp concentration bounds)
- Union bound

Today: use of a very simple fact

- An algorithm is successful on an input with probability at least t
- After (at least) $\frac{\ln n}{t}$ executions, the algorithm will be successful at least once with probability at least $1 - 1/n$
- $\Pr[\text{success}] = 1 - (1-t)^{\frac{\ln n}{t}} \geq 1 - (e^{-t})^{\frac{\ln n}{t}} = 1 - 1/n$

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

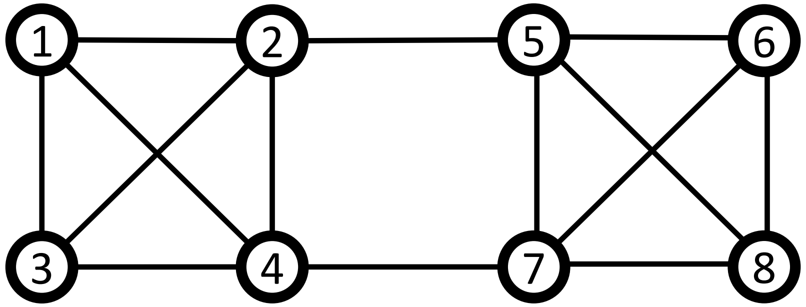
The minimum s-t cut problem in graphs

Cut problem in graphs

- **Minimum (global) cut**: Given a graph, compute a set of edges of minimum cardinality whose removal disconnects the graph
- **Minimum s-t cut**: Given a graph and two nodes s and t, compute a set of edges whose removal disconnect node s from node t

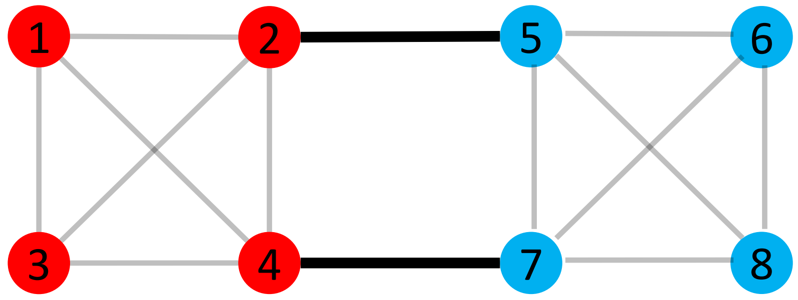
Cut problem in graphs

- **Minimum (global) cut**: Given a graph, compute a set of edges of minimum cardinality whose removal disconnects the graph
- **Minimum s-t cut**: Given a graph and two nodes s and t, compute a set of edges whose removal disconnect node s from node t



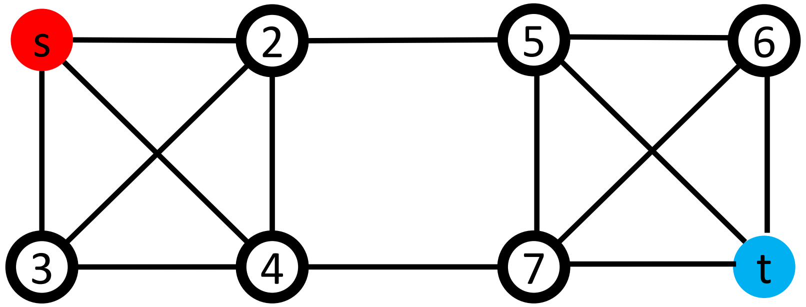
Cut problem in graphs

- **Minimum (global) cut**: Given a graph, compute a set of edges of minimum cardinality whose removal disconnects the graph
- **Minimum s-t cut**: Given a graph and two nodes s and t, compute a set of edges whose removal disconnect node s from node t



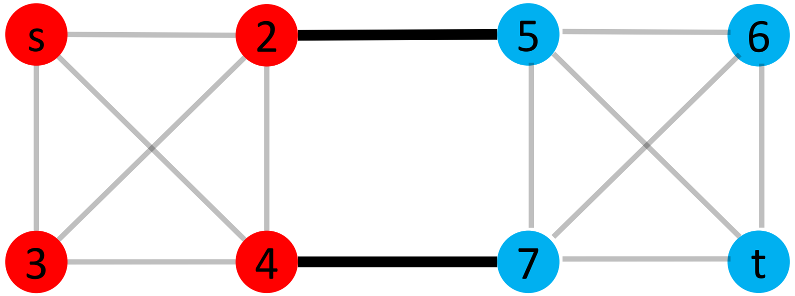
Cut problem in graphs

- **Minimum (global) cut**: Given a graph, compute a set of edges of minimum cardinality whose removal disconnects the graph
- **Minimum s-t cut**: Given a graph and two nodes s and t, compute a set of edges whose removal disconnect node s from node t



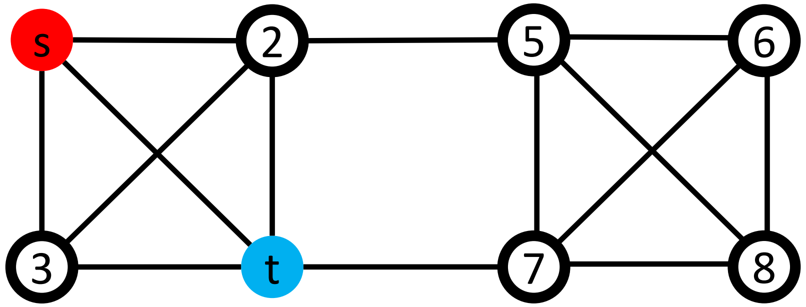
Cut problem in graphs

- **Minimum (global) cut**: Given a graph, compute a set of edges of minimum cardinality whose removal disconnects the graph
- **Minimum s-t cut**: Given a graph and two nodes s and t, compute a set of edges whose removal disconnect node s from node t



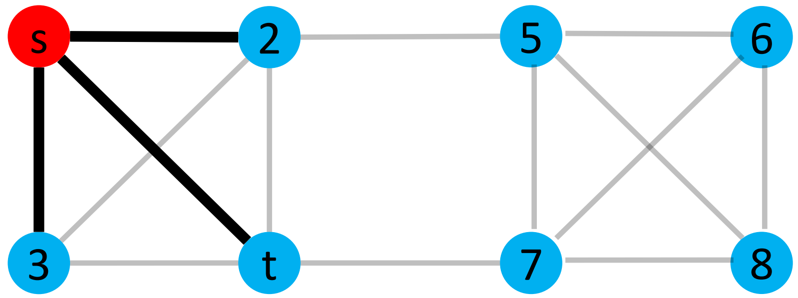
Cut problem in graphs

- **Minimum (global) cut**: Given a graph, compute a set of edges of minimum cardinality whose removal disconnects the graph
- **Minimum s-t cut**: Given a graph and two nodes s and t, compute a set of edges whose removal disconnect node s from node t



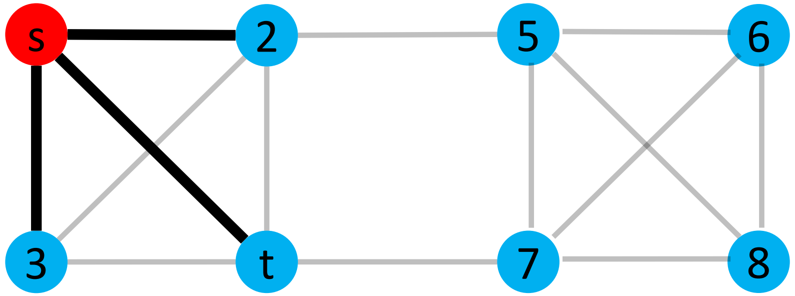
Cut problem in graphs

- **Minimum (global) cut**: Given a graph, compute a set of edges of minimum cardinality whose removal disconnects the graph
- **Minimum s-t cut**: Given a graph and two nodes s and t, compute a set of edges whose removal disconnect node s from node t



Cut problem in graphs

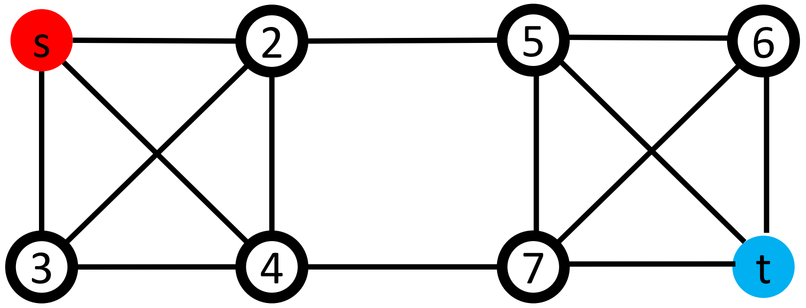
- **Minimum (global) cut**: Given a graph, compute a set of edges of minimum cardinality whose removal disconnects the graph
- **Minimum s-t cut**: Given a graph and two nodes s and t, compute a set of edges whose removal disconnect node s from node t



- The input graph is undirected, unweighted, and may contain multiple parallel edges

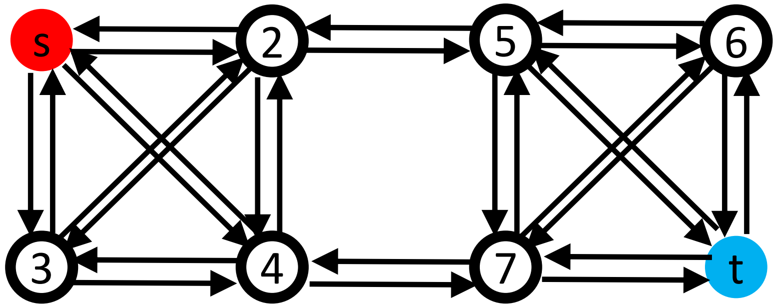
Algorithms

- Size of **minimum s-t cut** = value of **maximum s-t flow**
- Algorithms for computing maximum flows from node s to node t
- **Ford-Fulkerson**: compute augmenting paths in a **residual network**
- Several implementations: e.g., **Edmonds-Karp** algorithm has runnning time $O(n|E|^2)$, there are several **improvements**



Algorithms

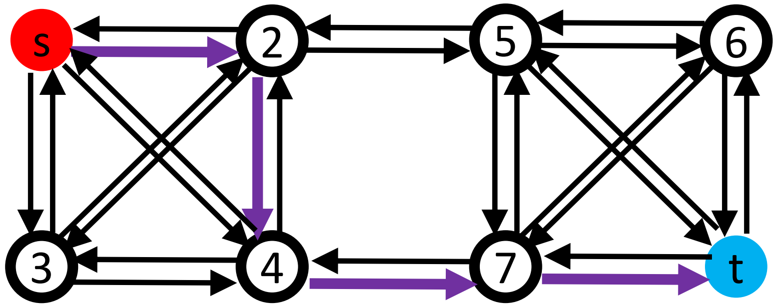
- Size of **minimum s-t cut** = value of **maximum s-t flow**
- Algorithms for computing maximum flows from node s to node t
- **Ford-Fulkerson**: compute augmenting paths in a **residual network**
- Several implementations: e.g., **Edmonds-Karp** algorithm has runnning time $O(n|E|^2)$, there are several **improvements**



residual network

Algorithms

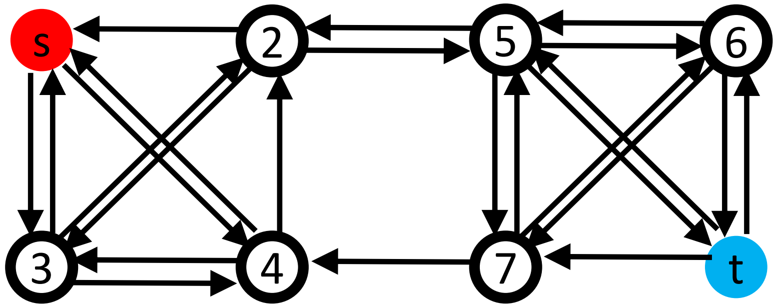
- Size of **minimum s-t cut** = value of **maximum s-t flow**
- Algorithms for computing maximum flows from node s to node t
- **Ford-Fulkerson**: compute augmenting paths in a **residual network**
- Several implementations: e.g., **Edmonds-Karp** algorithm has runnning time $O(n|E|^2)$, there are several **improvements**



compute augmenting paths

Algorithms

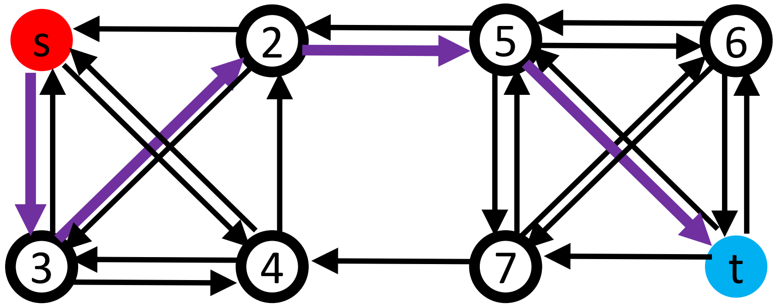
- Size of **minimum s-t cut** = value of **maximum s-t flow**
- Algorithms for computing maximum flows from node s to node t
- **Ford-Fulkerson**: compute augmenting paths in a **residual network**
- Several implementations: e.g., **Edmonds-Karp** algorithm has runnning time $O(n|E|^2)$, there are several **improvements**



remove path from network

Algorithms

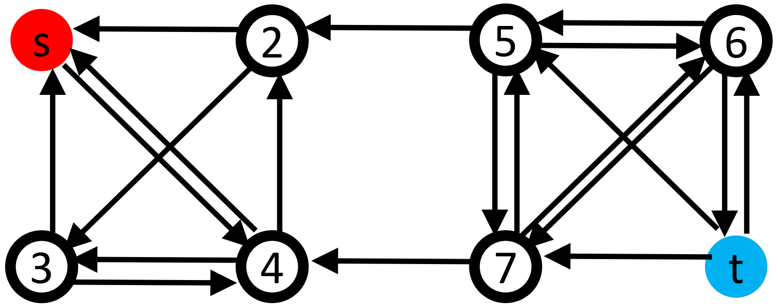
- Size of **minimum s-t cut** = value of **maximum s-t flow**
- Algorithms for computing maximum flows from node s to node t
- **Ford-Fulkerson**: compute augmenting paths in a **residual network**
- Several implementations: e.g., **Edmonds-Karp** algorithm has runnning time $O(n|E|^2)$, there are several **improvements**



and repeat …

Algorithms

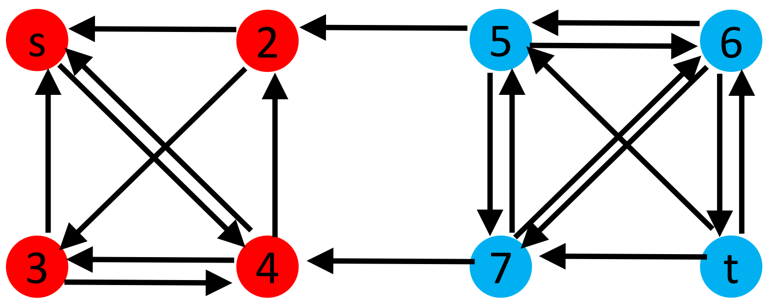
- Size of **minimum s-t cut** = value of **maximum s-t flow**
- Algorithms for computing maximum flows from node s to node t
- **Ford-Fulkerson**: compute augmenting paths in a **residual network**
- Several implementations: e.g., **Edmonds-Karp** algorithm has runnning time $O(n|E|^2)$, there are several **improvements**



until it is not possible anymore

Algorithms

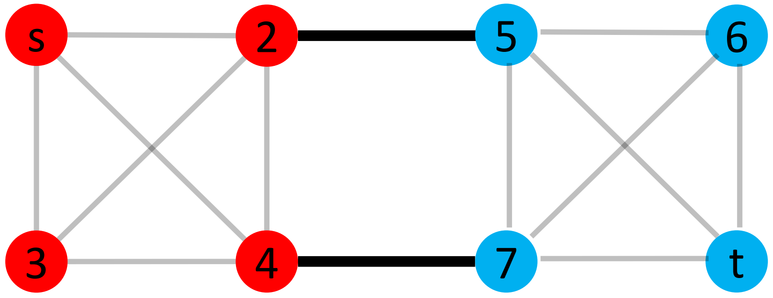
- Size of **minimum s-t cut** = value of **maximum s-t flow**
- Algorithms for computing maximum flows from node s to node t
- **Ford-Fulkerson**: compute augmenting paths in a **residual network**
- Several implementations: e.g., **Edmonds-Karp** algorithm has runnning time $O(n|E|^2)$, there are several **improvements**



then, BFS computation identifies the two sides of the cut

Algorithms

- Size of **minimum s-t cut** = value of **maximum s-t flow**
- Algorithms for computing maximum flows from node s to node t
- **Ford-Fulkerson**: compute augmenting paths in a **residual network**
- Several implementations: e.g., **Edmonds-Karp** algorithm has runnning time $O(n|E|^2)$, there are several **improvements**



minimum s-t cut

Computing a (global) cut of minimum size

Computing a (global) cut of minimum size

- Fix a node s
- The global cut should disconnect node s from any other node t
- For every selection of t, run the **Edmonds-Karp algorithm**
- Return the **minimum s-t cut in all these executions**

Computing a (global) cut of minimum size

- Fix a node s
- The global cut should disconnect node s from any other node t
- For every selection of t, run the **Edmonds-Karp algorithm**
- Return the **minimum s-t cut in all these executions**

running time: $O(n|E|^2)$

$n - 1$ executions

Computing a (global) cut of minimum size

- Fix a node s
- The global cut should disconnect node s from any other node t
- For every selection of t, run the **Edmonds-Karp algorithm**
- Return the **minimum s-t cut in all these executions**

- Overall running time: $O(n^2|E|^2)$

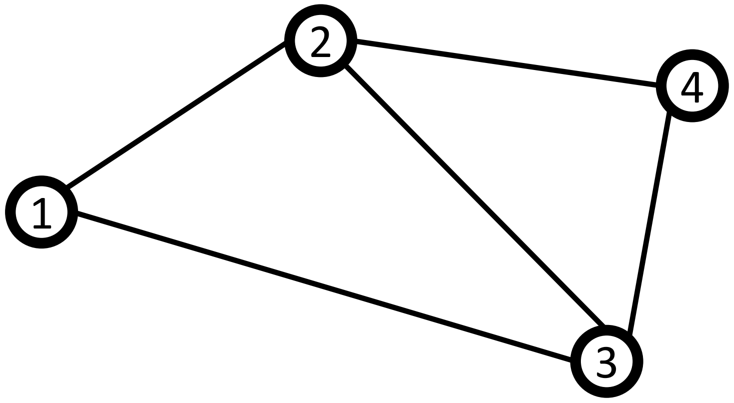
running time: $O(n|E|^2)$

$n-1$ executions

A randomized algorithm

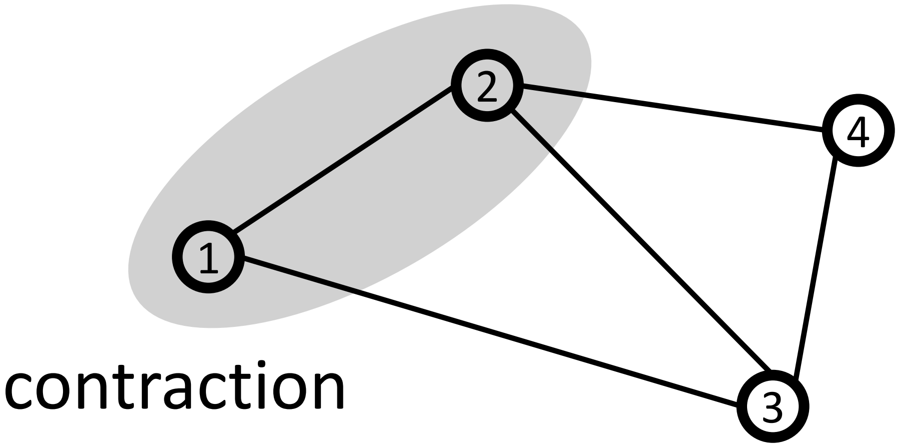
Contractions

- Input: a (multi)graph $G = (V, E)$
- A **contraction** of nodes u and v results in a **new graph** G'
- The nodes u and v are **merged** into a new node uv in G'
- The edges between u and v **disappear**
- Edges (u, w) and (v, w) in G become edges (uv, w) in G'



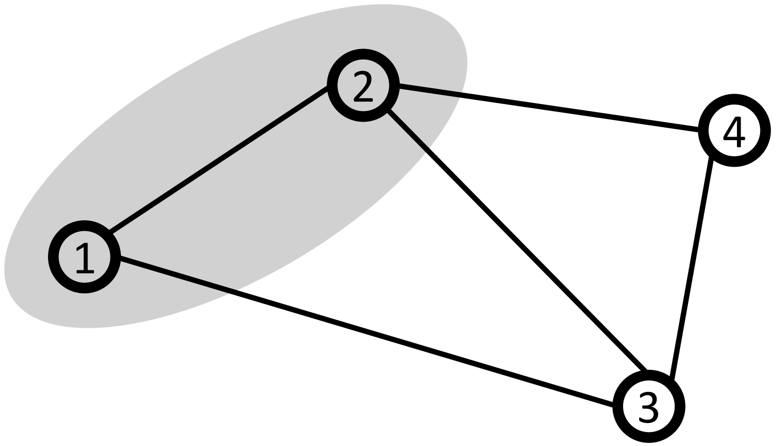
Contractions

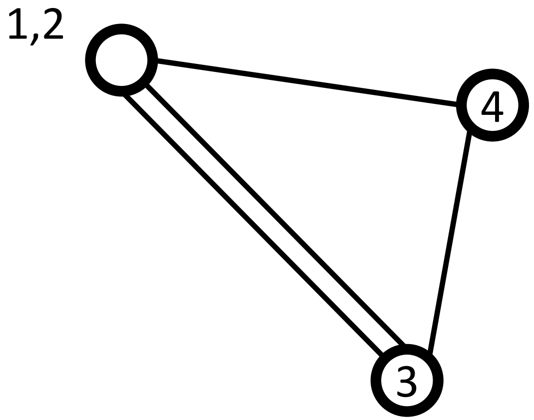
- Input: a (multi)graph $G = (V, E)$
- A **contraction** of nodes u and v results in a **new graph** G'
- The nodes u and v are **merged** into a new node uv in G'
- The edges between u and v **disappears**
- Edges (u, w) and (v, w) in G become edges $(\boldsymbol{uv}, \boldsymbol{w})$ in G'



Contractions

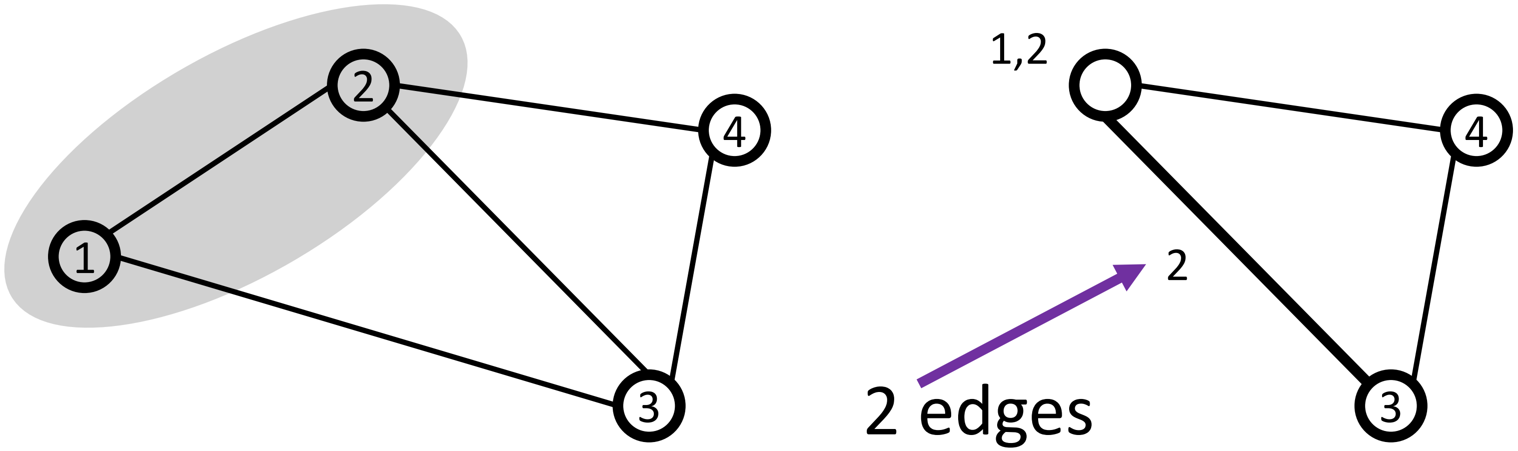
- Input: a (multi)graph $G = (V, E)$
- A **contraction** of nodes u and v results in a **new graph** G'
- The nodes u and v are **merged** into a new node uv in G'
- The edges between u and v **disappears**
- Edges (u, w) and (v, w) in G become edges $(\boldsymbol{uv}, \boldsymbol{w})$ in G'





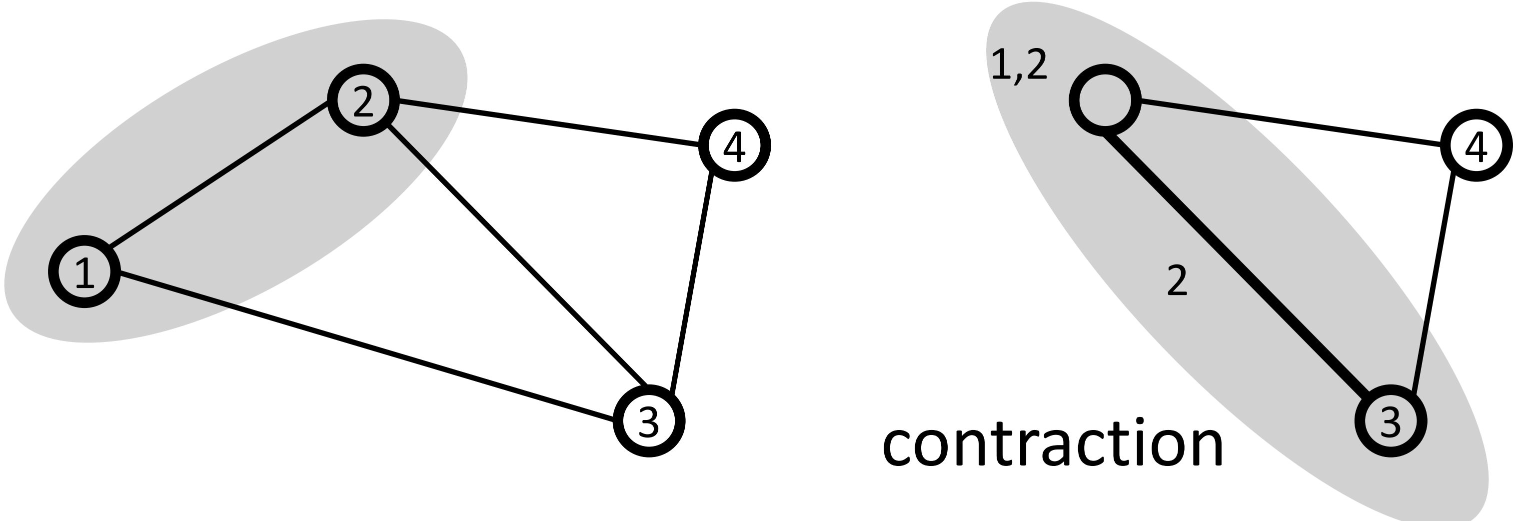
Contractions

- Input: a (multi)graph $G = (V, E)$
- A **contraction** of nodes u and v results in a **new graph** G'
- The nodes u and v are **merged** into a new node uv in G'
- The edges between u and v **disappears**
- Edges (u, w) and (v, w) in G become edges $(\boldsymbol{uv}, \boldsymbol{w})$ in G'



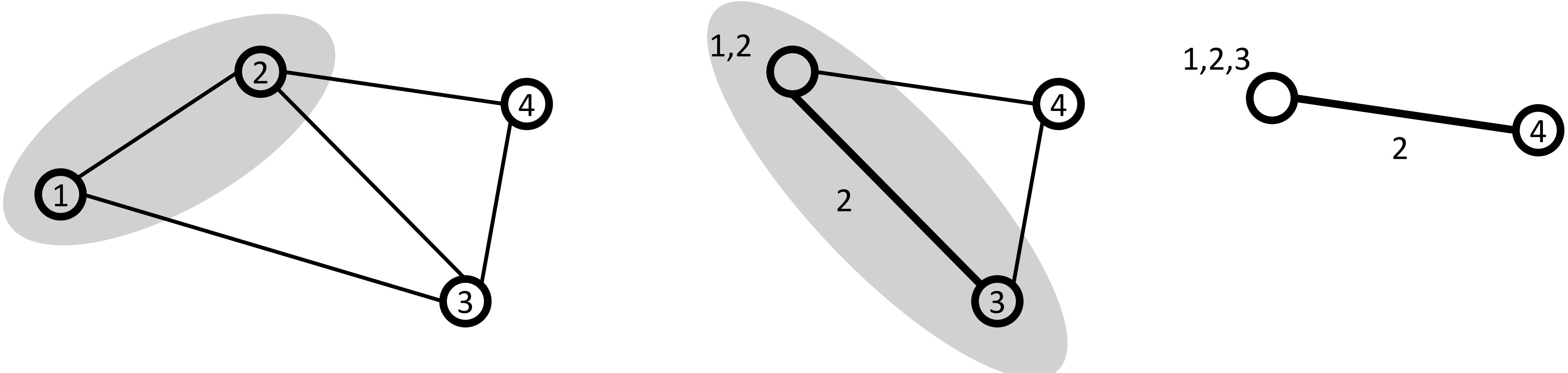
Contractions

- Input: a (multi)graph $G = (V, E)$
- A **contraction** of nodes u and v results in a **new graph** G'
- The nodes u and v are **merged** into a new node uv in G'
- The edges between u and v **disappears**
- Edges (u, w) and (v, w) in G become edges $(\boldsymbol{uv}, \boldsymbol{w})$ in G'



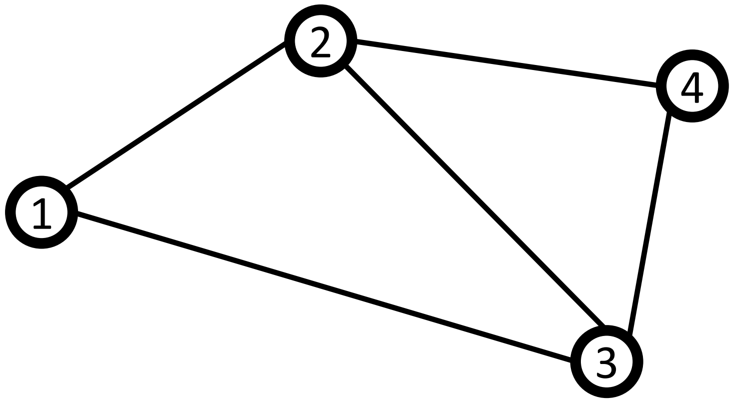
Contractions

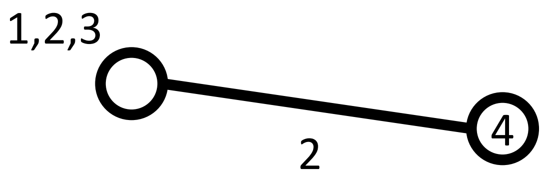
- Input: a (multi)graph $G = (V, E)$
- A **contraction** of nodes u and v results in a **new graph** G'
- The nodes u and v are **merged** into a new node uv in G'
- The edges between u and v **disappears**
- Edges (u, w) and (v, w) in G become edges $(\boldsymbol{uv}, \boldsymbol{w})$ in G'



Contractions

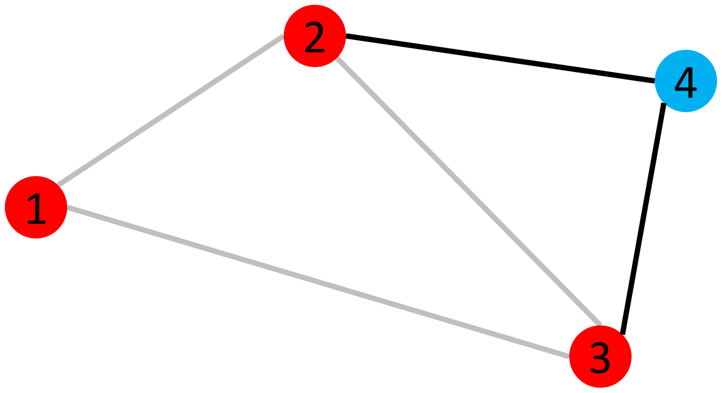
- What kind of **information** do we see in the contracted graph?

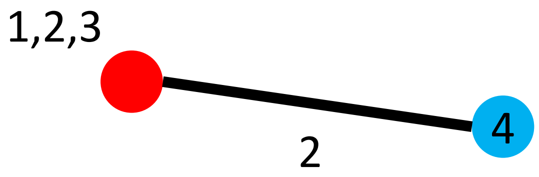




Contractions

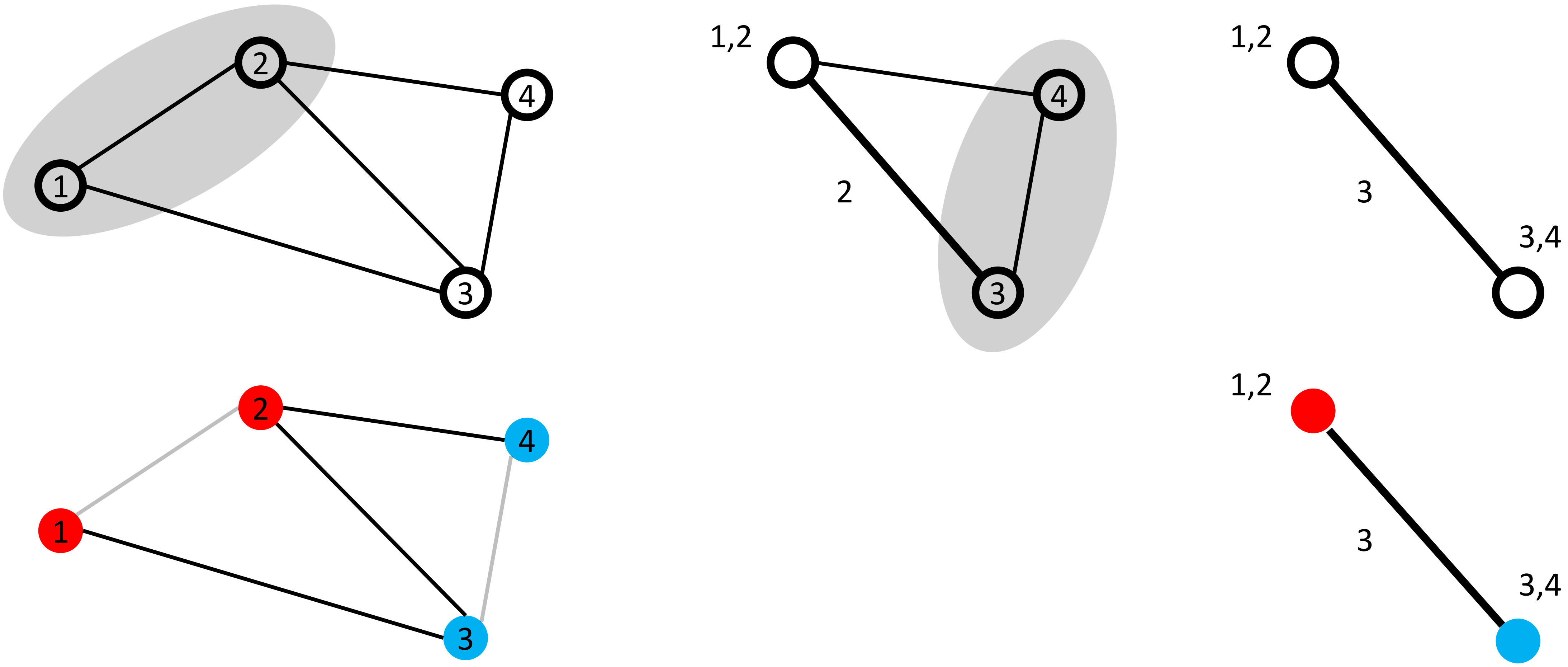
- What kind of **information** do we see in the contracted graph?





- Number of edges between the sets of nodes {1,2,3} and {4}

Another sequence of contractions



The contraction algorithm

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

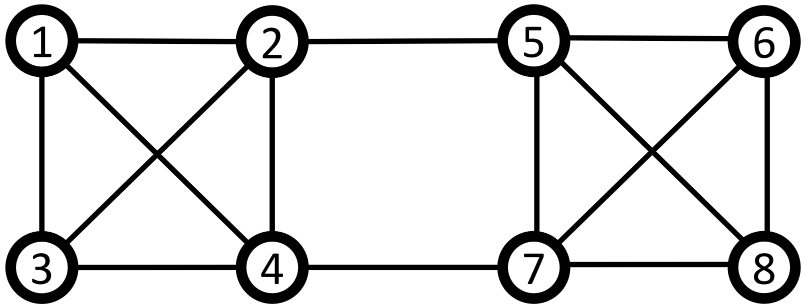
Return the set of edges between the two nodes

The contraction algorithm: an example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

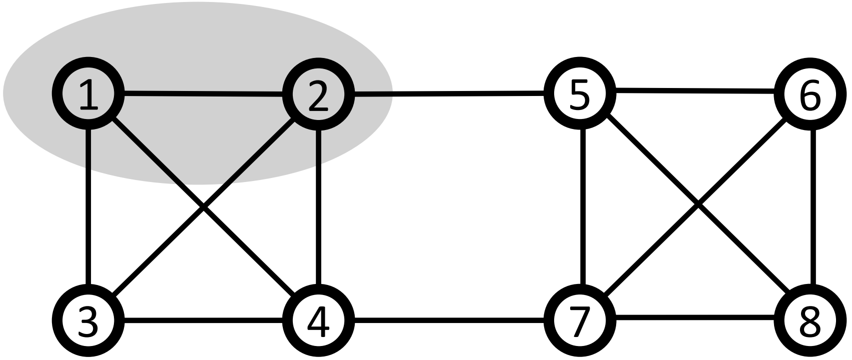
Return the set of edges between the two nodes



The contraction algorithm: an example

Repeat until just two nodes remain
- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes

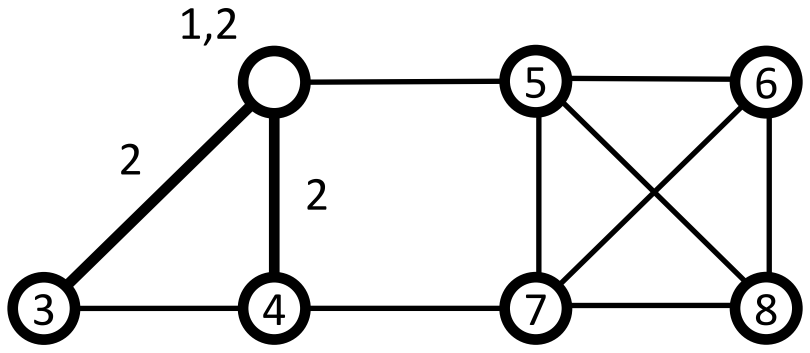


The contraction algorithm: an example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes

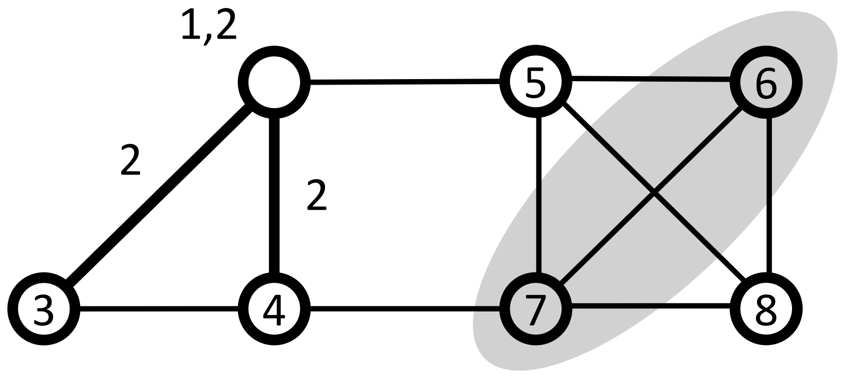


The contraction algorithm: an example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes

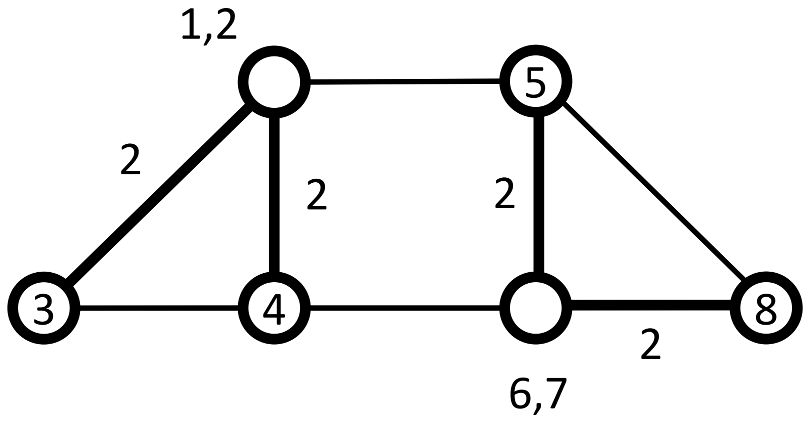


The contraction algorithm: an example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes

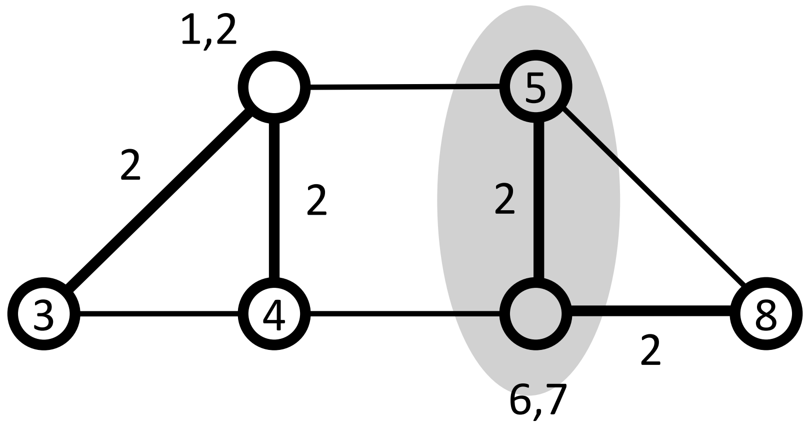


The contraction algorithm: an example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes

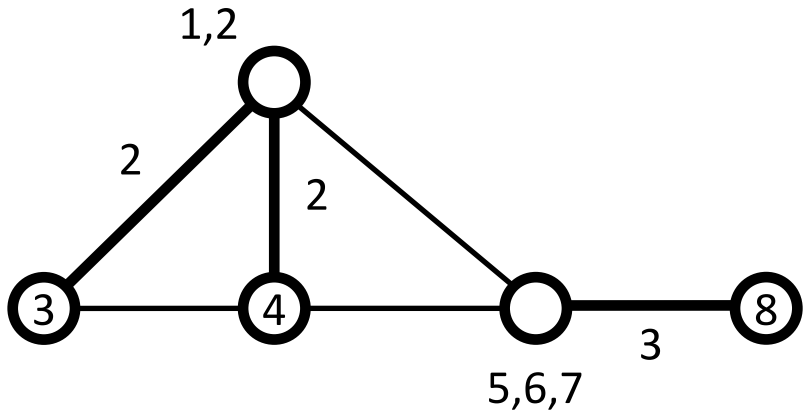


The contraction algorithm: an example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes

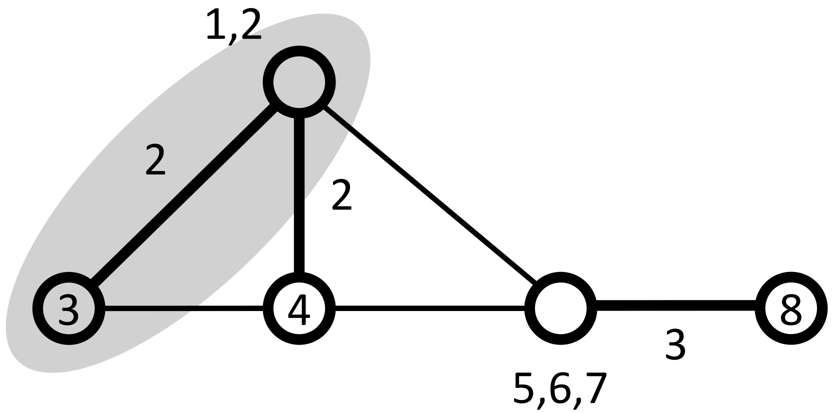


The contraction algorithm: an example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

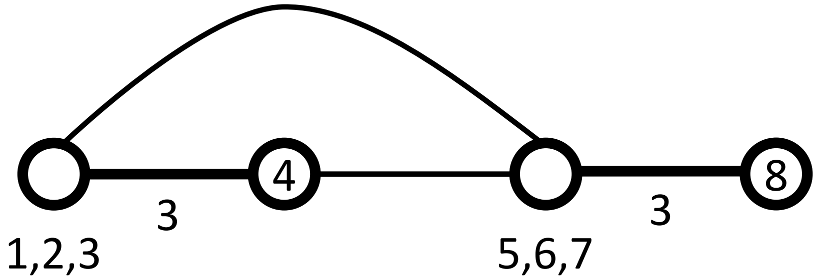
Return the set of edges between the two nodes



The contraction algorithm: an example

Repeat until just two nodes remain
- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes

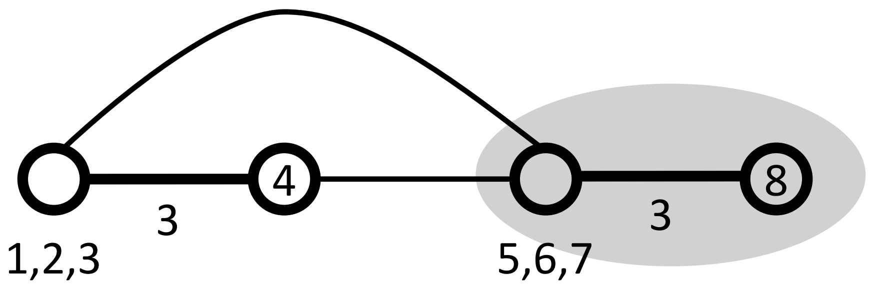


The contraction algorithm: an example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes

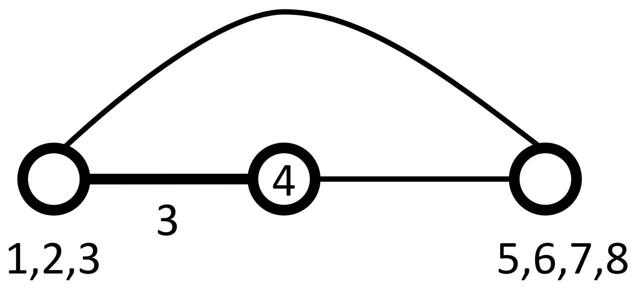


The contraction algorithm: an example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes

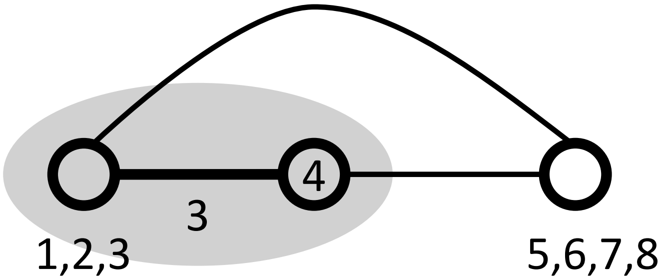


The contraction algorithm: an example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes

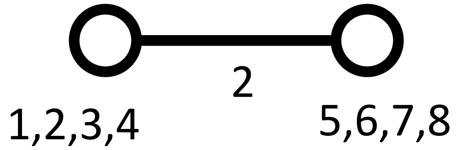


The contraction algorithm: an example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes



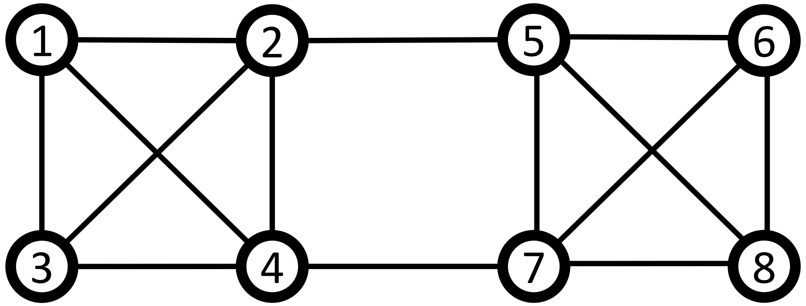
the minimum cut was found ☺

The contraction algorithm: another example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes

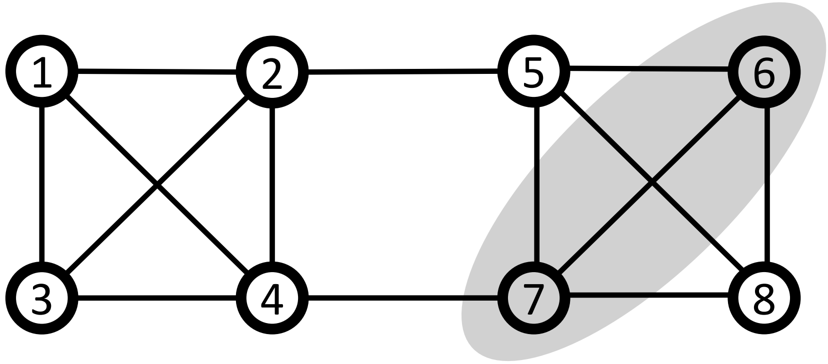


The contraction algorithm: another example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes

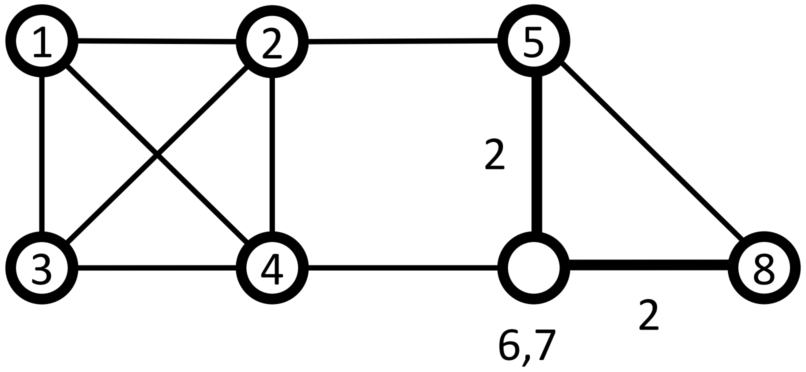


The contraction algorithm: another example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes

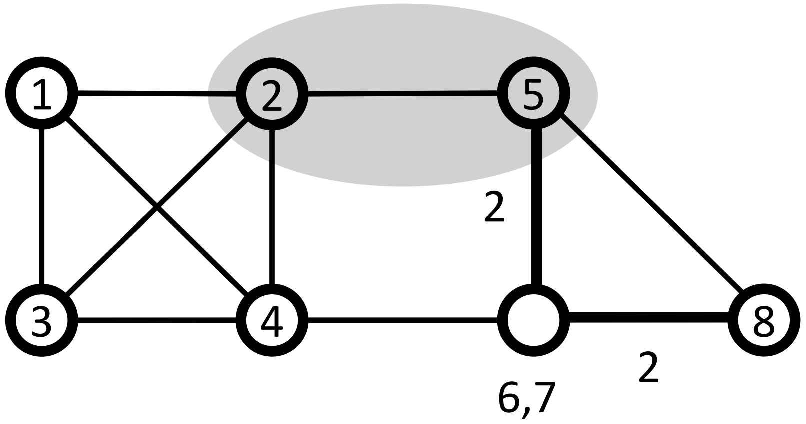


The contraction algorithm: another example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes

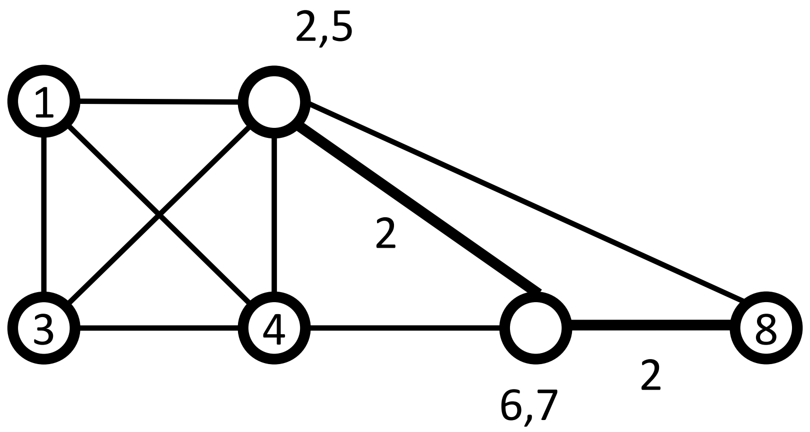


The contraction algorithm: another example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes

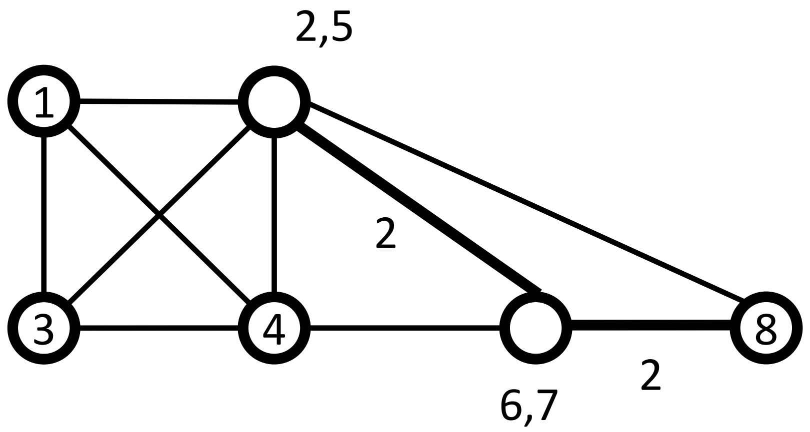


The contraction algorithm: another example

Repeat until just two nodes remain

- Pick an edge uniformly at random and contract its endpoints

Return the set of edges between the two nodes



no chance to find the minimum cut ☹

Analysis of the contraction algorithm

The algorithm has running time $O(n^2)$

- Time $O(\log n)$ to select uniformly at random per step (not very important)
- Linear time to update the node/edge information after a contraction
- $n - 2$ contractions in total

Analysis of the contractio

Much better than the
Edmonds-Karp algorithm

The algorithm has running time $O(n^2)$

- Time $O(\log n)$ to select uniformly at random per step (not very important)
- Linear time to update the node/edge information after a contraction
- $n - 2$ contractions in total

Analysis of the contraction algorithm (contd.)

What is the prob. that the algorithm is successful?

Analysis of the contraction algorithm (contd.)

What is the prob. that the algorithm is successful?

Fix a cut C of minimum size

All edges in C will have survived at the end if none of them is selected for the $n-2$ contractions

Analysis of the contraction algorithm (contd.)

What is the prob. that the algorithm is successful?

Fix a cut C of minimum size

All edges in C will have survived at the end if none of them is selected for the $n-2$ contractions

Event $E_i =$ some edge of C is selected for the i-th contraction

Event $S_i =$ all edges of C have survived after the i-th contraction

$$\Pr[S_{n-2}] = \Pr[\overline{E_1}] \cdot \Pr[\overline{E_2} | \overline{E_1}] \cdot \Pr[\overline{E_3} | \overline{E_1} \cap \overline{E_2}] \cdots \Pr[\overline{E_{n-2}} | \overline{E_1} \cap \overline{E_2} \ldots \cap \overline{E_{n-3}}]$$

Analysis of the contraction algorithm (contd.)

Bounding the probability $\Pr[\overline{E_i} | \overline{E_1} \cap \overline{E_2} \ldots \cap \overline{E_{i-1}}]$

At step i:

- The edges of C still appear in the graph
- The graph has $n - i + 1$ nodes

No node has degree less than $|C|$; otherwise, C would not be of minimum size

Hence, the number of edges is at least $(n - i + 1)|C|/2$; recall the property $\sum_{v \in V} \deg v = 2|E|$

Then, the probability $\Pr[E_i | \overline{E_1} \cap \overline{E_2} \ldots \cap \overline{E_{i-1}}]$ is at most $\frac{|C|}{\#\text{edges}} \leq \frac{2}{n-i+1}$

Analysis of the contraction algorithm (contd.)

Recap:

The probability of success of the contraction algorithm is

$\Pr[S_{n-2}] = \Pr[\overline{E_1}] \cdot \Pr[\overline{E_2}|\overline{E_1}] \cdot \Pr[\overline{E_3}|\overline{E_1} \cap \overline{E_2}] \cdots \Pr[\overline{E_{n-2}}|\overline{E_1} \cap \overline{E_2} \ldots \cap \overline{E_{n-3}}]$

where $\Pr[\overline{E_i}|\overline{E_1} \cap \overline{E_2} \ldots \cap \overline{E_{i-1}}] \geq 1 - \frac{2}{n-i+1}$

Analysis of the contraction algorithm (contd.)

Recap:

The probability of success of the contraction algorithm is

$\Pr[S_{n-2}] = \Pr[\overline{E_1}] \cdot \Pr[\overline{E_2}|\overline{E_1}] \cdot \Pr[\overline{E_3}|\overline{E_1} \cap \overline{E_2}] \cdots \Pr[\overline{E_{n-2}}|\overline{E_1} \cap \overline{E_2} \ldots \cap \overline{E_{n-3}}$

where $\Pr[\overline{E_i}|\overline{E_1} \cap \overline{E_2} \ldots \cap \overline{E_{i-1}}] \geq 1 - \frac{2}{n-i+1}$

Hence,

$\Pr[S_{n-2}] \geq \left(1 - \frac{2}{n}\right) \cdot \left(1 - \frac{2}{n-1}\right) \cdot \left(1 - \frac{2}{n-2}\right) \cdots \left(1 - \frac{2}{5}\right) \cdot \left(1 - \frac{2}{4}\right) \cdot \left(1 - \frac{2}{3}\right)$

$= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \ldots \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3}$

Analysis of the contraction algorithm (contd.)

Recap:

The probability of success of the contraction algorithm is

$$\Pr[S_{n-2}] = \Pr[\overline{E_1}] \cdot \Pr[\overline{E_2}|\overline{E_1}] \cdot \Pr[\overline{E_3}|\overline{E_1} \cap \overline{E_2}] \cdots \Pr[\overline{E_{n-2}}|\overline{E_1} \cap \overline{E_2} \ldots \cap \overline{E_{n-3}}]$$

where $\Pr[\overline{E_i}|\overline{E_1} \cap \overline{E_2} \ldots \cap \overline{E_{i-1}}] \geq 1 - \frac{2}{n-i+1}$

Hence,

$$\Pr[S_{n-2}] \geq \left(1 - \frac{2}{n}\right) \cdot \left(1 - \frac{2}{n-1}\right) \cdot \left(1 - \frac{2}{n-2}\right) \cdots \left(1 - \frac{2}{5}\right) \cdot \left(1 - \frac{2}{4}\right) \cdot \left(1 - \frac{2}{3}\right)$$

$$= \frac{\cancel{n-2}}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{\cancel{n-2}} \cdots \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3}$$

Analysis of the contraction algorithm (contd.)

Recap:

The probability of success of the contraction algorithm is

$\Pr[S_{n-2}] = \Pr[\overline{E_1}] \cdot \Pr[\overline{E_2} | \overline{E_1}] \cdot \Pr[\overline{E_3} | \overline{E_1} \cap \overline{E_2}] \cdots \Pr[\overline{E_{n-2}} | \overline{E_1} \cap \overline{E_2} \ldots \cap \overline{E_{n-3}}$

where $\Pr[\overline{E_i} | \overline{E_1} \cap \overline{E_2} \ldots \cap \overline{E_{i-1}}] \geq 1 - \frac{2}{n-i+1}$

Hence,

$\Pr[S_{n-2}] \geq \left(1 - \frac{2}{n}\right) \cdot \left(1 - \frac{2}{n-1}\right) \cdot \left(1 - \frac{2}{n-2}\right) \cdots \left(1 - \frac{2}{5}\right) \cdot \left(1 - \frac{2}{4}\right) \cdot \left(1 - \frac{2}{3}\right)$

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

Analysis of the contraction algorithm (contd.)

- The algorithm has running time $O(n^2)$ and is successful with prob. at least $2/n(n-1)$
- Run the algorithm $O(n^2 \log n)$ times and return the best cut
- Then, the algorithm computes the minimum cut with prob. at least $1 - 1/n$ in time $O(n^4 \log n)$

Analysis of the contraction algorithm (contd.)

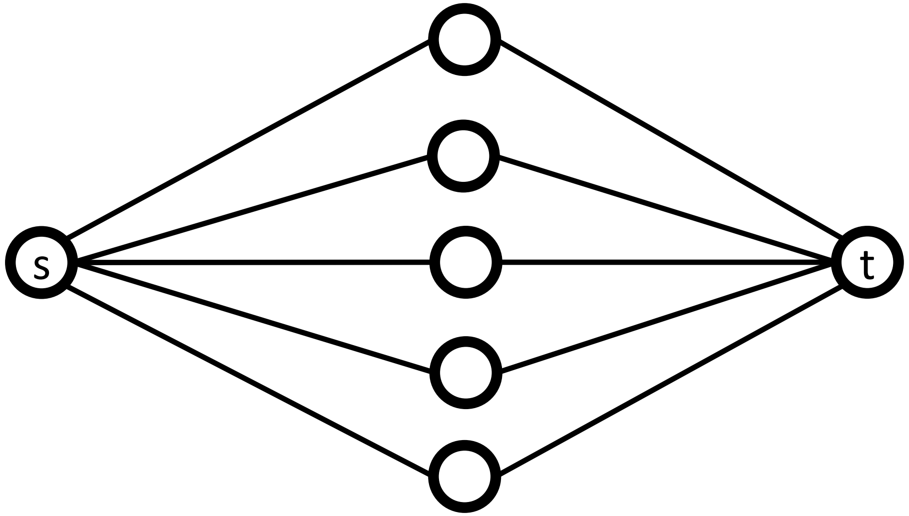
- The algorithm has running time … h, at least $2/n(n-1)$
- Run the algorithm $O(\,$…
- Then, the algorithm … at least $1 - 1/n$ in time $O(n^4 \log n)$

Still better than the Edmonds-Karp algorithm for not very sparse graphs

Counting the number of minimum cuts in graphs

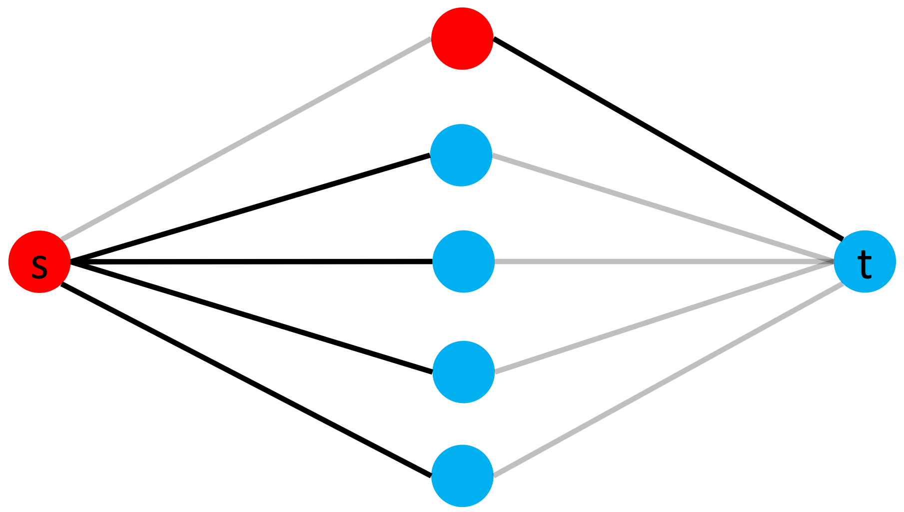
How many cuts are there in a graph?

- Too many minimum s-t cuts



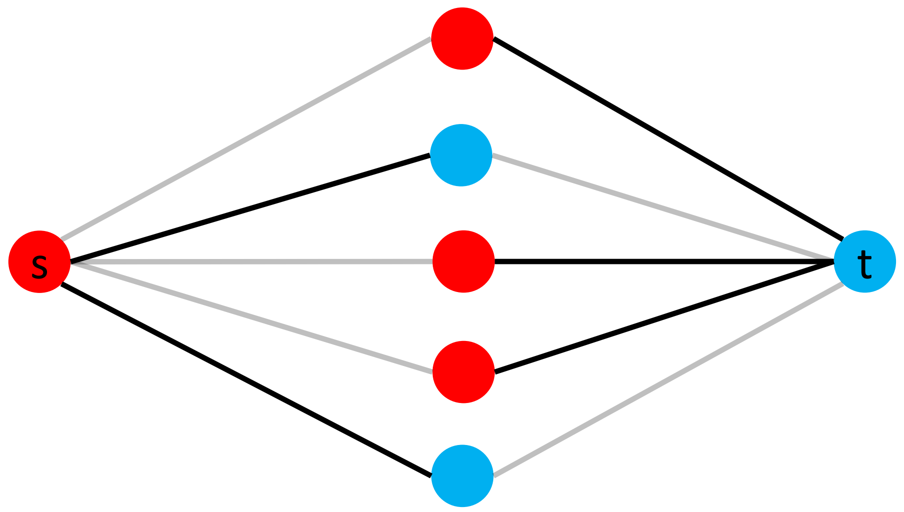
How many cuts are there in a graph?

- Too many minimum s-t cuts



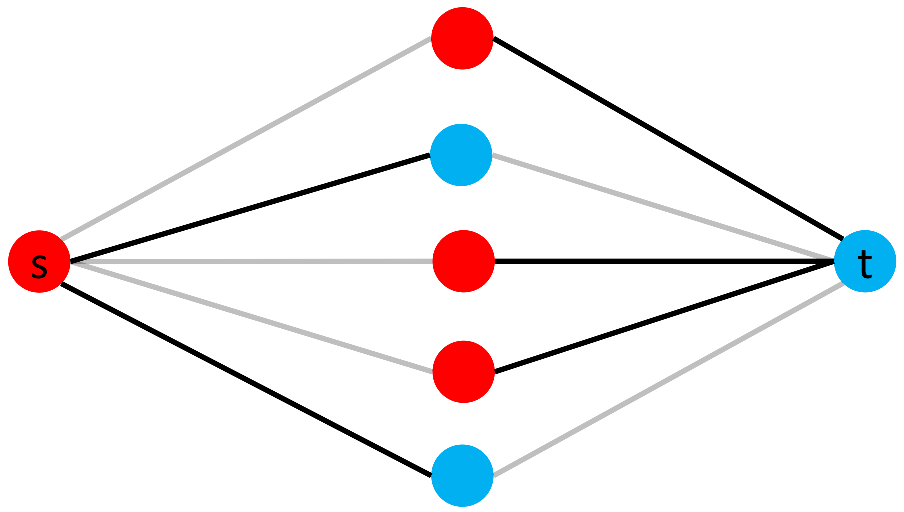
How many cuts are there in a graph?

- Too many minimum s-t cuts



How many cuts are there in a graph?

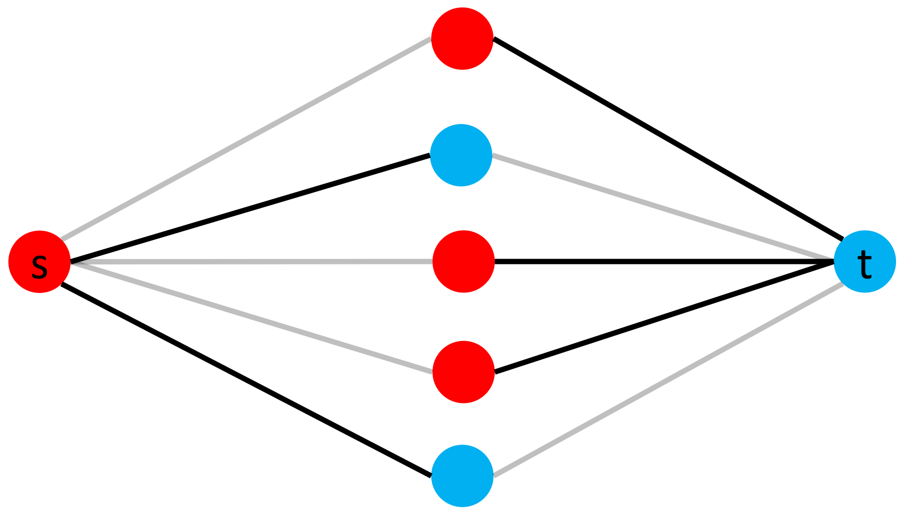
- Too many minimum s-t cuts



- 2^{n-2} in the above example

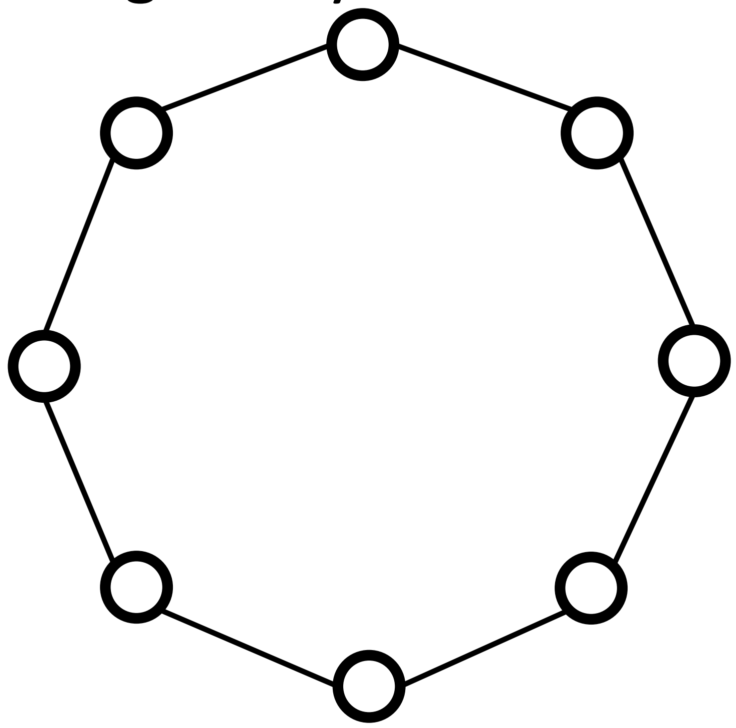
How many cuts are there in a graph?

- Too many minimum s-t cuts



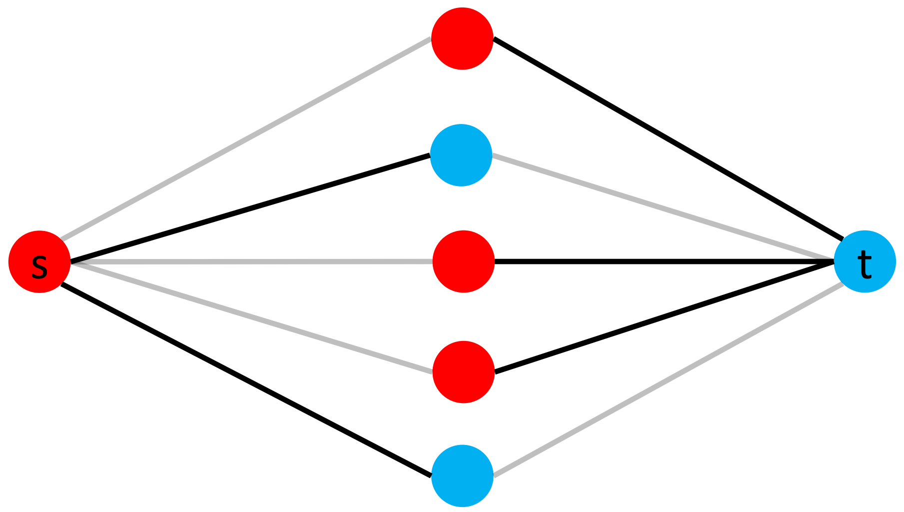
- 2^{n-2} in the above example

- Too few globally minimum cuts



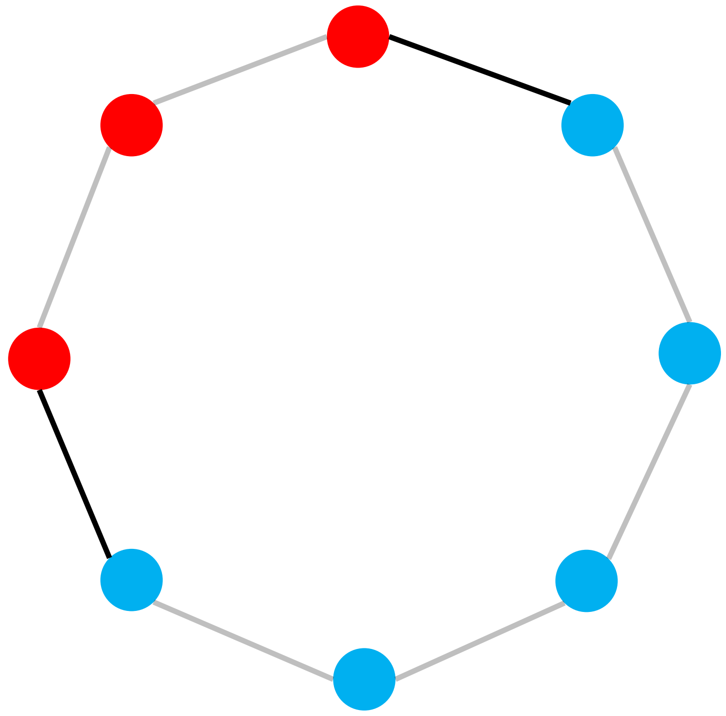
How many cuts are there in a graph?

- Too many minimum s-t cuts



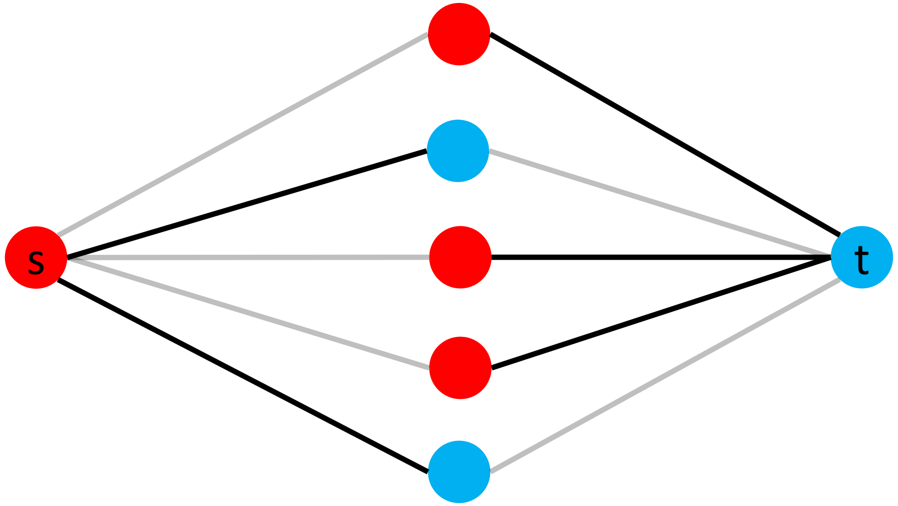
- 2^{n-2} in the above example

- Too few globally minimum cuts



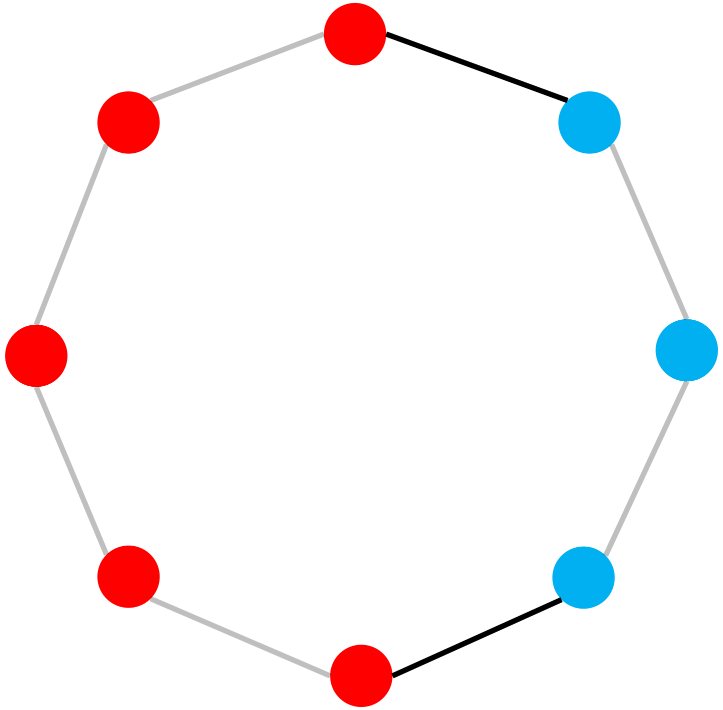
How many cuts are there in a graph?

- Too many minimum s-t cuts



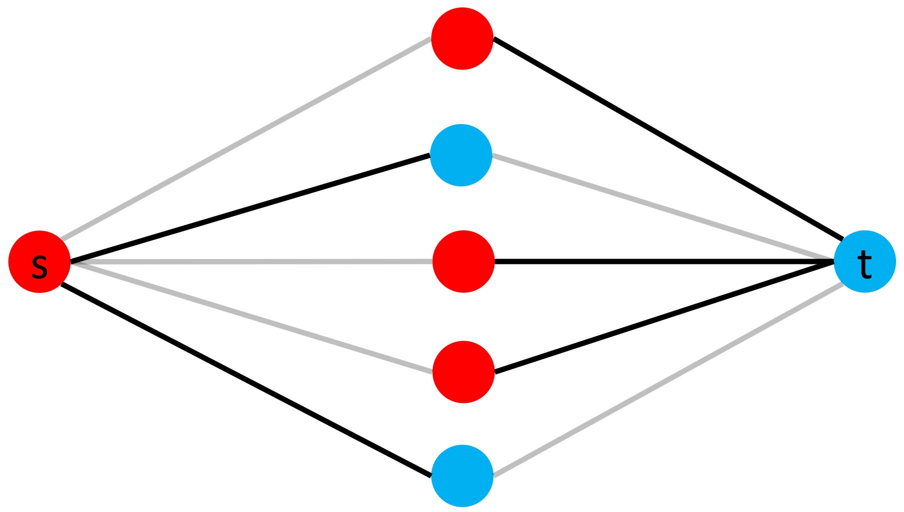
- 2^{n-2} in the above example

- Too few globally minimum cuts



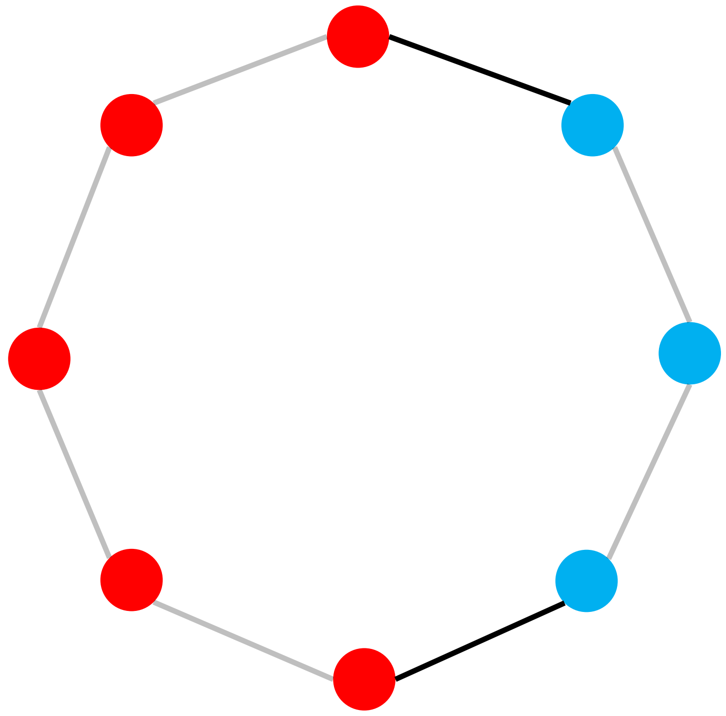
How many cuts are there in a graph?

- Too many minimum s-t cuts



- 2^{n-2} in the above example

- Too few globally minimum cuts



- $\frac{n(n-1)}{2}$ here. How worse can it be?

A graph-theoretic implication

Theorem: Any graph has at most $n(n-1)/2$ distinct global cuts of minimum size

A graph-theoretic implication

<u>Theorem</u>: Any graph has at most $n(n-1)/2$ distinct global cuts of minimum size

- Can we use our result for the contraction algorithm?

A graph-theoretic implication

Theorem: Any graph has at most $n(n-1)/2$ distinct global cuts of minimum size

- Can we use our result for the contraction algorithm?
- Recall that **we fixed a particular minimum cut** and proved that the prob. that it is returned by the contraction algorithm (in one execution) is at least $2/n(n-1)$
- Denote by C_i the event that the i-th minimum cut is returned in an execution by the contraction algorithm
- Hence, $\mathbf{Pr[C_i] \geq 2/n(n-1)}$
- **How many i's** can we have?

A graph-theoretic implication

Theorem: Any graph has at most $n(n-1)/2$ distinct global cuts of minimum size

A trivial argument:

- We have a **random process** that **selects a card** from a subset of the deck
- The selection can be arbitrary but we know that any spade is selected with probability at least t
- Then, the number of **spades** cannot exceed $1/t$

Proof idea: apply the argument with the **contraction algorithm** as the random process, the **cards as outcomes** of the contraction algorithm, and the **minimum cuts** as the spades

Improving the contraction algorithm

A crucial observation

- The probability that the minimum cut C has survived the first contractions is very high

$$\Pr[S_{n-2}] \geq \underbrace{\left(1-\frac{2}{n}\right)\cdot\left(1-\frac{2}{n-1}\right)\cdot\left(1-\frac{2}{n-2}\right)}_{\text{large factors}}\cdots\underbrace{\left(1-\frac{2}{4}\right)\cdot\left(1-\frac{2}{3}\right)}_{\text{small factors}}$$

A crucial observation

- The probability that the minimum cut C has survived the first contractions is very high

$$\Pr[S_{n-2}] \geq \underbrace{\left(1-\frac{2}{n}\right)\cdot\left(1-\frac{2}{n-1}\right)\cdot\left(1-\frac{2}{n-2}\right)}_{\text{large factors}}\cdots\underbrace{\left(1-\frac{2}{4}\right)\cdot\left(1-\frac{2}{3}\right)}_{\text{small factors}}$$

- $\Pr[S_{n-l}] \geq \left(1-\frac{2}{n}\right)\cdot\left(1-\frac{2}{n-1}\right)\cdot\left(1-\frac{2}{n-2}\right)\cdot \quad \cdot\left(1-\frac{2}{l+2}\right)\cdot\left(1-\frac{2}{l+1}\right)$

$$= \frac{n-2}{n}\cdot\frac{n-3}{n-1}\cdot\frac{n-4}{n-2}\cdot \quad \cdot\frac{l}{l+2}\cdot\frac{l-1}{l+1}$$

i.e., $\Pr[S_{n-l}]$ is constant if $l = \Omega(n)$

A crucial observation

- The probability that the minimum cut C has survived the first contractions is very high

$$\Pr[S_{n-2}] \geq \underbrace{\left(1-\frac{2}{n}\right)\cdot\left(1-\frac{2}{n-1}\right)\cdot\left(1-\frac{2}{n-2}\right)}_{\text{large factors}}\cdots\underbrace{\left(1-\frac{2}{4}\right)\cdot\left(1-\frac{2}{3}\right)}_{\text{small factors}}$$

- $\Pr[S_{n-l}] \geq \left(1-\frac{2}{n}\right)\cdot\left(1-\frac{2}{n-1}\right)\cdot\left(1-\frac{2}{n-2}\right)\cdot\quad\cdot\left(1-\frac{2}{l+2}\right)\cdot\left(1-\frac{2}{l+1}\right)$

$$= \frac{\cancel{n-2}}{n}\cdot\frac{\cancel{n-3}}{n-1}\cdot\frac{\cancel{n-4}}{\cancel{n-2}}\cdot\quad\cdot\frac{l}{\cancel{l+2}}\cdot\frac{l-1}{\cancel{l+1}} = \frac{l(l-1)}{n(n-1)}$$

i.e., $\Pr[S_{n-l}]$ is constant if $l = \Omega(n)$

A crucial observation

- $\Pr[S_{n-l}]$ is constant if $l = \Omega(n)$
- E.g., $\boxed{\text{if } l \approx \frac{n}{\sqrt{2}} \text{ then } \Pr[S_{n-l}] \geq 1/2}$
- So, much fewer (e.g., $O(\log n)$) executions of the first $n - l$ contractions suffice to make almost certain that C has survived in some of them
- Instead, these steps are executed in all the $O(n^2 \log n)$ executions of the contraction algorithm

- Question: how can we **modify the algorithm** to avoid the execution of these steps?

A faster algorithm

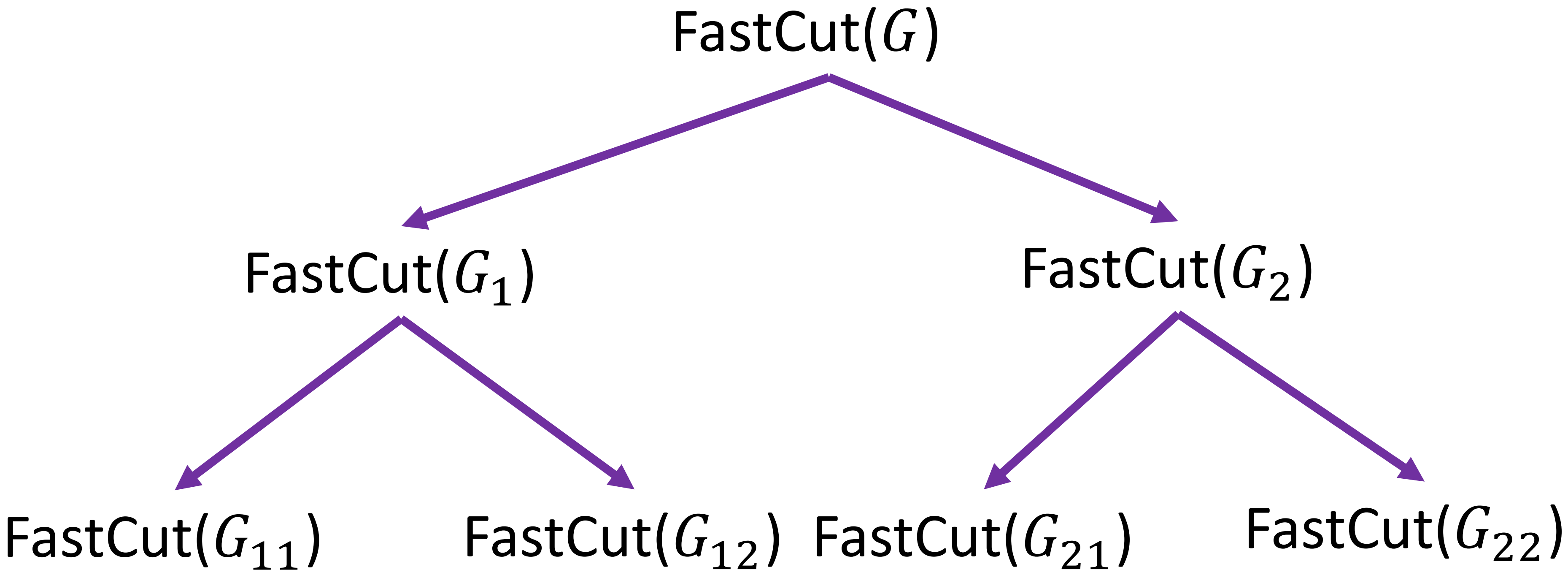
FastCut (multigraph G)

- If $n \leq 6$, compute the minimum cut via **exhaustive search** and return
- $G_1 :=$ a graph obtained after $n - n/\sqrt{2}$ (random) contractions
- $G_2 :=$ a graph obtained after $n - n/\sqrt{2}$ (random) contractions

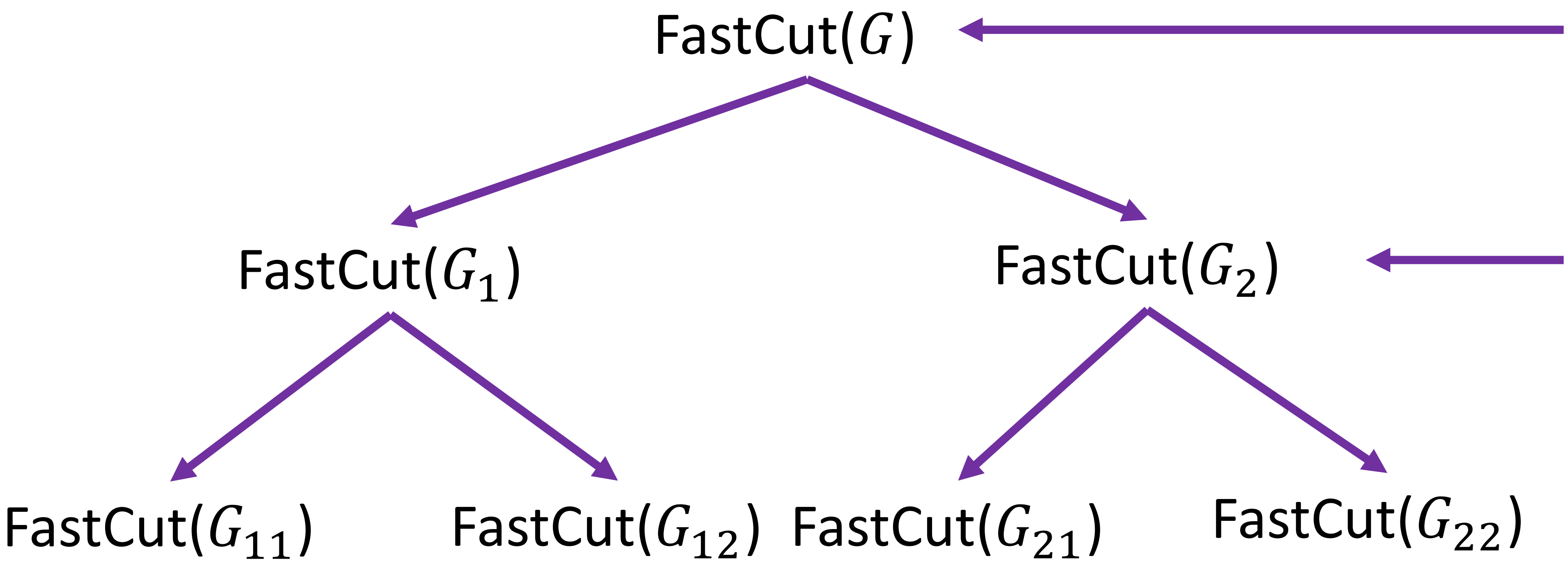
$$\left.\begin{array}{l} \bullet\ X_1 := \text{FastCut}\,(G_1) \\ \bullet\ X_2 := \text{FastCut}\,(G_2) \end{array}\right\} \textbf{recursive calls}$$

- Return the minimum cut between X_1 and X_2

Analysis of FastCut



Analysis of FastCut

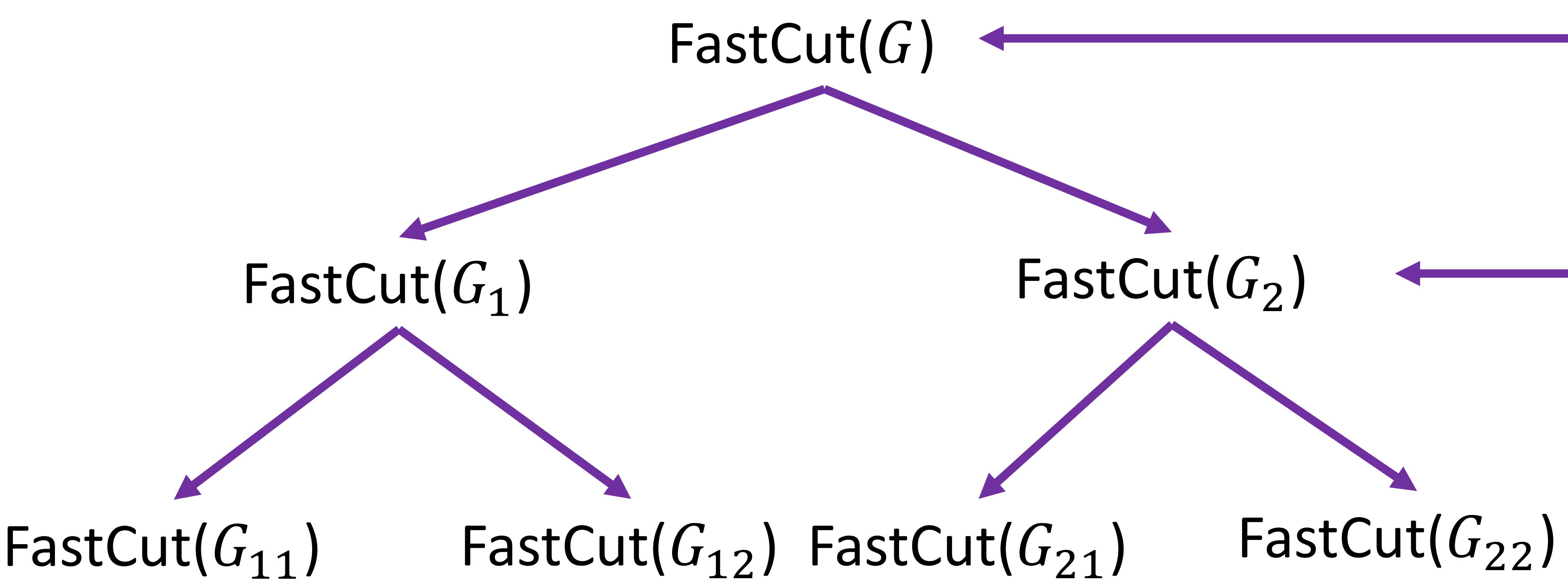


$O(n^2)$ steps for the contractions

2 problems of size $n/\sqrt{2}$ each

- $T(n) = 2T\left(\frac{n}{\sqrt{2}}\right) + O(n^2)$
- I.e., $T(n) = O(n^2 \log n)$

Analysis of FastCut



$O(n^2)$ steps for the contractions

2 problems of size $n/\sqrt{2}$ each

- $T(n) = 2T\left(\frac{n}{\sqrt{2}}\right) + O(n^2)$
- I.e., $T(n) = O(n^2 \log n)$

So, the running time is **higher** than that of the contraction algorithm

How do we gain?

Analysis of FastCut (contd.)

- <u>Theorem</u>: FastCut finds a minimum cut with prob. at least $c/\log n$
- $P(n) =$ prob. that a call of FastCut with an n-node input is successful

$$P(n) \geq 1 - \left(1 - \frac{1}{2} \cdot P\left(\frac{n}{\sqrt{2}}\right)\right)^2$$

Analysis of FastCut (contd.)

- Theorem: FastCut finds a minimum cut with prob. at least $c/\log n$
- $P(n) =$ prob. that a call of FastCut with an n-node input is successful

$$P(n) \geq 1 - \overbrace{\left(1 - \tfrac{1}{2} \cdot \underbrace{P\left(\tfrac{n}{\sqrt{2}}\right)}_{\text{one successful recursive call}}\right)^2}^{\text{not successful}}$$

two independent contractions

Analysis of FastCut (contd.)

- Theorem: FastCut finds a minimum cut with prob. at least $c/\log n$
- $P(n) =$ prob. that a call of FastCut with an n-node input is successful

$$P(n) \geq 1 - \overbrace{\left(1 - \frac{1}{2} \cdot \underbrace{P\left(\frac{n}{\sqrt{2}}\right)}_{\text{one successful recursive call}}\right)}^{\text{not successful}}{}^{2}$$

two independent contractions

- Yields $P(n) = O(1/\log n)$ (why?)

Analysis of FastCut (contd.)

- <u>Theorem</u>: FastCut finds a minimum cut with prob. at least $c/\log n$
- $P(n)$ = prob. that a call of FastCut with an n-node input is successful

$$P(n) \geq 1 - \overbrace{\left(1 - \underset{\text{two independent contractions}}{\frac{1}{2}} \cdot \underbrace{P\left(\frac{n}{\sqrt{2}}\right)}_{\text{one successful recursive call}}\right)^2}^{\text{not successful}}$$

- Yields $P(n) = O(1/\log n)$ (why?)
- By repeating FastCut $O(\log^2 n)$ times, it will find the minimum cut with prob. at least $1 - 1/n$ in one of them
- Overall running time $\boxed{O(n^2 \log^3 n)}$

The solution of $P(n) \geq 1 - \left(1 - \frac{1}{2} \cdot P\left(\frac{n}{\sqrt{2}}\right)\right)^2$ satisfies $P(n) = O(1/\log n)$

- Set $Q(h) = P\left(2^{h/2}\right)$
- The inequality becomes $Q(h) \geq Q(h-1) - \frac{Q(h-1)^2}{4}$
- It suffices to show inductively that $Q(h) \geq \frac{1}{h+1}$
- Observe that the RHS of the inequality is increasing in $Q(h-1)$ (why?)
- Since, by the induction hypothesis, $Q(h-1) \geq 1/h$, the inequality implies that $Q(h) \geq 1/h - \frac{(1/h)^2}{4} \geq \frac{1}{h} - \frac{1}{h(h+1)} = \frac{1}{h+1}$ as desired
- The inequality (and the base case) follows since $n \geq 6$ and, hence, $h \geq 3$

References

- the contraction algorithm is due to Karger (SODA 1993)
- FastCut is due to Karger and Stein (JACM 1996)

Last slide

- Quick recap: randomization, useful inequalities
- Computing a (global) cut of minimum size in a graph
- A randomized algorithm
- Analysis
- Implications and improvements