15-251
Great Theoretical Ideas in Computer Science

Lecture 19:
Randomized Algorithms

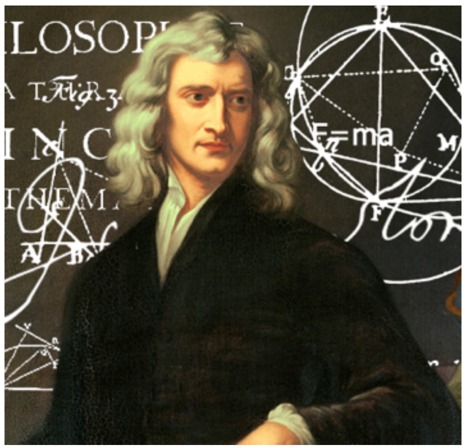
November 1st, 2016

Where we are

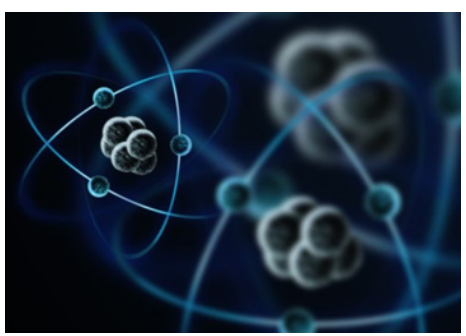
Oct 24	Oct 25 ***Probability 1***	Oct 26 hw7 w.s.	Oct 27 ***Probability 2***	Oct 28
Oct 31	Nov 1 ***Randomized Algs.***	Nov 2 hw8 w.s.	Nov 3 ***Markov Chains***	Nov 4
Nov 7	Nov 8 ***Modular Arithmetic***	Nov 9 hw9 w.s.	Nov 10 ***Cryptography***	Nov 11
Nov 14	Nov 15 ***Group Theory***	Nov 16 Midterm 2	Nov 17 ***Fields and Polys***	Nov 18
Nov 21	Nov 22 ***Communication Comp.***	Nov 23 THANKSGIVING	Nov 24 THANKSGIVING	Nov 25 THANKSGIVING
Nov 28	Nov 29 ***Err. Correcting Codes***	Nov 30 hw10 w.s.	Dec 1 ***Generating Functions***	Dec 2
Dec 5	Dec 6 ***Interactive Proofs***	Dec 7 hw11 w.s.	Dec 8 ***Epilogue***	Dec 9

Randomness and the universe

Does the universe have true randomness?



Newtonian physics suggests that the universe evolves deterministically.

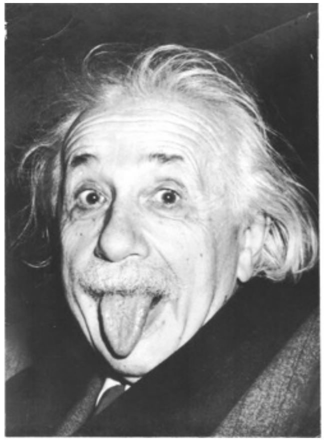


Quantum physics says otherwise.

Randomness and the universe

Does the universe have <u>true randomness</u>?

Opinion 1:



God does not play dice with the world.

- *Albert Einstein*

Opinion 2:

Einstein, don't tell God what to do.

- *Niels Bohr*

Randomness and the universe

Does the universe have <u>true randomness</u>?

Even if it doesn't, we can still model our uncertainty using probability.

Randomness is an essential tool in modeling and analyzing nature.

It also plays a key role in computer science.

Randomness in computer science

- Randomized algorithms

    Does randomness speed up computation?

  Statistics via sampling

    e.g. *election polls*

  Nash equilibrium in Game Theory

    Nash equilibrium always exists if players can have probabilistic strategies.

- Cryptography

    A secret is only as good as the entropy/uncertainty in it.

Randomness in computer science

- Randomized models for deterministic objects

  e.g. *the www graph*

  Quantum computing

  Randomness is inherent in quantum mechanics.

  Machine learning theory

  Data is generated by some probability distribution.

- Coding Theory

  Encode data to be able to deal with random noise.

…

Topic of the Day:

Randomized Algorithms

Randomness and algorithms

How can randomness be used in computation?

Given some algorithm that solves a problem:

(i) the input can be chosen randomly
(average-case analysis).

(ii) the algorithm can make random choices
(randomized algorithm).

Which one will we focus on?

Randomness and algorithms

What is a <u>randomized algorithm</u>?

A *randomized algorithm* is an algorithm that is allowed to flip a coin (i.e., has access to random bits).

In 15-251:

A randomized algorithm is an algorithm that is allowed to call:

- RandInt(n)
- Bernoulli(p)

(we'll assume these take $O(1)$ time)

Deterministic vs Randomized

Deterministic

```
def f(x):
    y = 1
    if(y == 0):
        while(x > 0):
            x = x - 1
    return x+y
```

Randomized

```
def f(x):
    y = Bernoulli(0.5)
    if(y == 0):
        while(x > 0):
            x = x - 1
    return x+y
```

For any <u>fixed</u> input (e.g. x = 3):

Deterministic:

- the output is invariant
- the running time is invariant

Randomized:

- the output can vary
- the running time can vary

Deterministic vs Randomized

A deterministic algorithm A computes $f : \Sigma^* \to \Sigma^*$
in time $T(n)$ means:

- correctness: $\forall x \in \Sigma^*, \quad A(x) = f(x)$.
- running time: $\forall x \in \Sigma^*, \quad \#\text{ steps } A(x) \text{ takes is } \leq T(|x|)$.

Note: we require worst-case guarantees for
correctness and run-time.

Deterministic vs Randomized

A randomized algorithm A computes $f : \Sigma^* \to \Sigma^*$
in time $T(n)$ means:

- correctness: $\forall x \in \Sigma^*,$?
- running time: $\forall x \in \Sigma^*,$?

Deterministic vs Randomized

Try 1

A randomized algorithm A computes $f : \Sigma^* \to \Sigma^*$
in time $T(n)$ means:

- correctness: $\forall x \in \Sigma^*$, $\boxed{A(x)} = f(x)$.
- running time: $\forall x \in \Sigma^*$, $\boxed{\#\text{ steps } A(x) \text{ takes}}$ is $\leq T(|x|)$.

these are random

Deterministic vs Randomized

Try 2

A randomized algorithm A computes $f : \Sigma^* \to \Sigma^*$ in time $T(n)$ means:

- correctness: $\forall x \in \Sigma^*$, $\mathbf{Pr}[A(x) = f(x)] = 1$.
- running time: $\forall x \in \Sigma^*$,

$$\mathbf{Pr}[\# \text{ steps } A(x) \text{ takes is } \leq T(|x|)] = 1 \,.$$

Is this interesting? No.

A randomized algorithm is allowed to gamble with either correctness or running time.

$$\forall x \in \Sigma^*$$

		Correctness	Run-time
Deterministic		always	always $\leq T(n)$
Randomized	Type 0	always	always $\leq T(n)$
	Type 1	w.h.p.	always $\leq T(n)$
	Type 2	always	w.h.p. $\leq T(n)$
	Type 3	w.h.p.	w.h.p. $\leq T(n)$

Type 0: may as well be deterministic

Type 1: "Monte Carlo algorithm"

Type 2: "Las Vegas algorithm"

Type 3: Can be converted to type 1. (exercise)

Example

Input: An array B with n elements (n even).
Half of the array contains 0s, the other half contains 1s.
Output: An index that contains a 1.

Deterministic

```
for i = 0 to n-1:
    if B[i] = 1:
        return i
```

correct: always

run-time: always $O(n)$

Randomized

Type 1 (Monte Carlo)

```
repeat 300 times:
    i = RandInt(n)
    if B[i] = 1:
        return i
return "Failed"
```

correct: w.h.p.

run-time: always $O(1)$

Type 2 (Las Vegas)

```
repeat:
    i = RandInt(n)
    if B[i] = 1:
        return i
```

correct: always

run-time: w.h.p. $O(1)$

Example

Input: An array B with n elements (n even).
Half of the array contains 0s, the other half contains 1s.
Output: An index that contains a 1.

	Correctness	Run-time
Deterministic	always	always $O(n)$
Monte Carlo	w.h.p.	always $O(1)$
Las Vegas	always	w.h.p. $O(1)$

Formal definition: deterministic algorithm

Let $f : \Sigma^* \to \Sigma^*$ be a computational problem.

We say that deterministic algorithm A computes f in time $T(n)$ if:

$\forall x \in \Sigma^*,$ $A(x) = f(x)$

$\forall x \in \Sigma^*,$ $\#$ steps $A(x)$ takes is $\leq T(|x|)$.

Formal definition: Monte Carlo algorithm

Let $f : \Sigma^* \to \Sigma^*$ be a computational problem.

We say that randomized algorithm A
is a $T(n)$-time Monte Carlo algorithm for f
with ϵ error probability if:

$\forall x \in \Sigma^*,$ $\qquad \mathbf{Pr}[A(x) \neq f(x)] \leq \epsilon$

$\forall x \in \Sigma^*,$ $\qquad \#\text{ steps } A(x) \text{ takes is } \leq T(|x|).$

(no matter what the random choices are)

Formal definition: Las Vegas algorithm

Let $f : \Sigma^* \to \Sigma^*$ be a computational problem.

We say that randomized algorithm A is a $T(n)$-time Las Vegas algorithm for f if:

$\forall x \in \Sigma^*,$ $\quad A(x) = f(x)$

(no matter what the random choices are)

$\forall x \in \Sigma^*,$ $\quad \mathbf{E}[\# \text{ steps } A(x) \text{ takes}] \leq T(|x|)$

(this implies run-time is $O(T(n))$ w.h.p.)

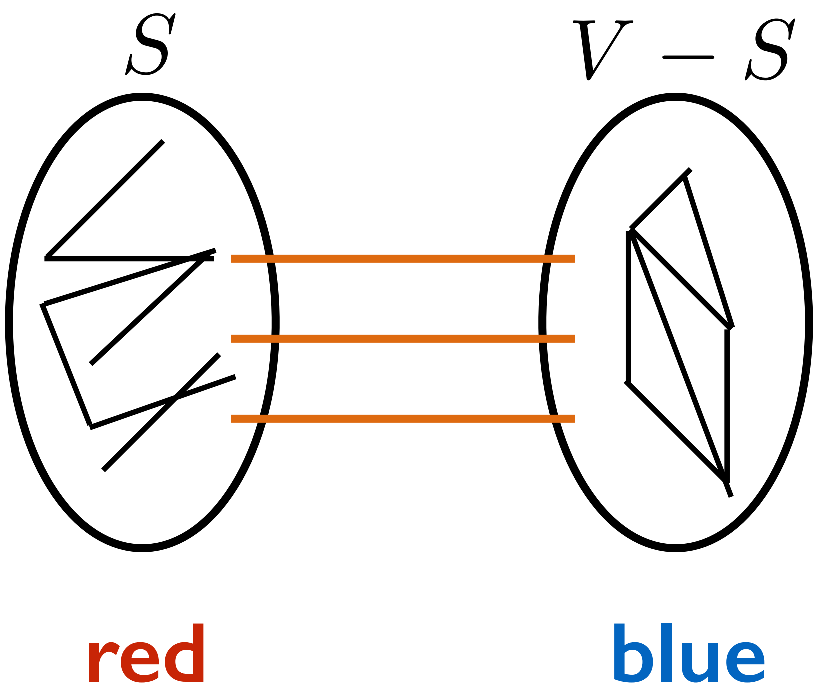
CASE STUDY

Monte Carlo Algorithm for Min Cut

Gambles with correctness.
Doesn't gamble with run-time.

Cut Problems

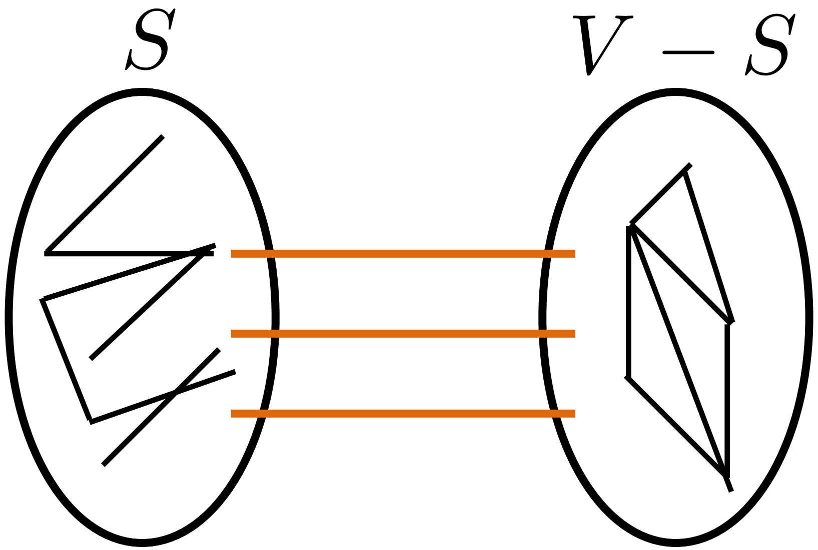
Max Cut Problem (Ryan O'Donnell's favorite problem):
Given a connected graph $G = (V, E)$,
color the vertices **red** and **blue** so that the number of edges with two colors (e = {**u**,**v**}) is maximized.



Cut Problems

<u>Max Cut Problem</u> (Ryan O'Donnell's favorite problem):

Given a connected graph $G = (V, E)$,
find a non-empty subset $S \subset V$ such that
number of edges from S to $V - S$ is maximized.

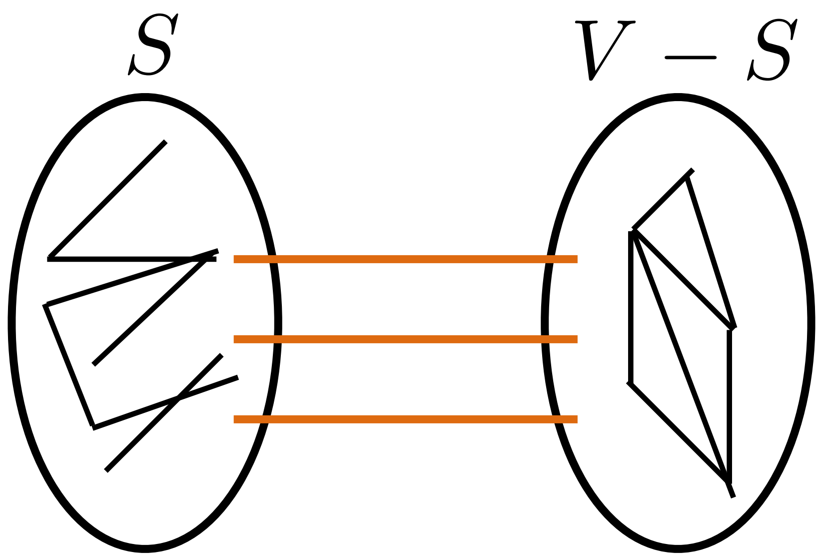


size of the cut = # edges from S to $V - S$.

Cut Problems

<u>Min Cut Problem</u> (my favorite problem):

Given a connected graph $G = (V, E)$,
find a non-empty subset $S \subset V$ such that
number of edges from S to $V - S$ is ***<u>minimized</u>***.



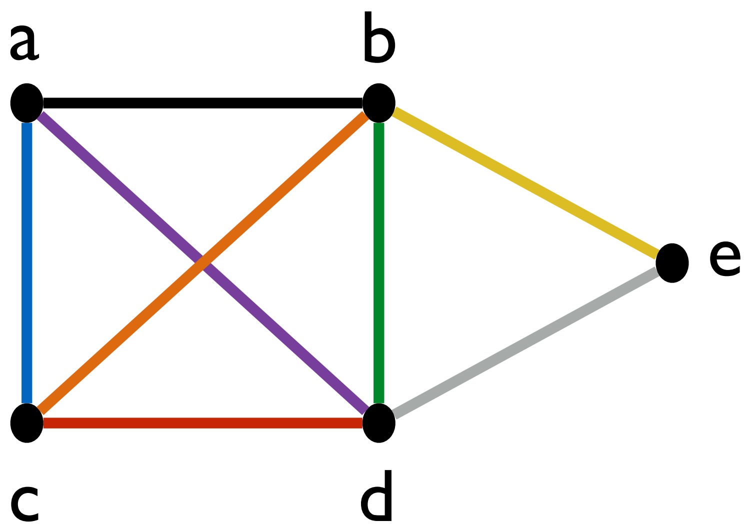
size of the cut = # edges from S to $V - S$.

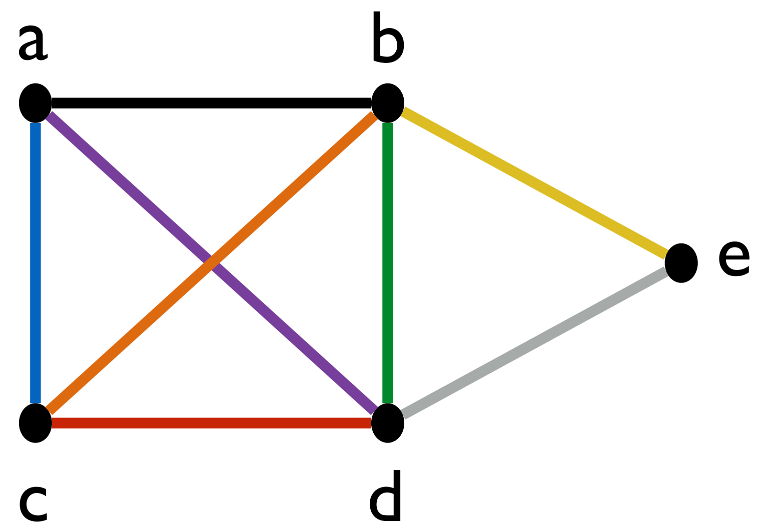
Contraction algorithm for min cut

Let's see a simple randomized algorithm for Min-Cut.

Contraction algorithm for min cut

Example run





Size of min-cut: 2

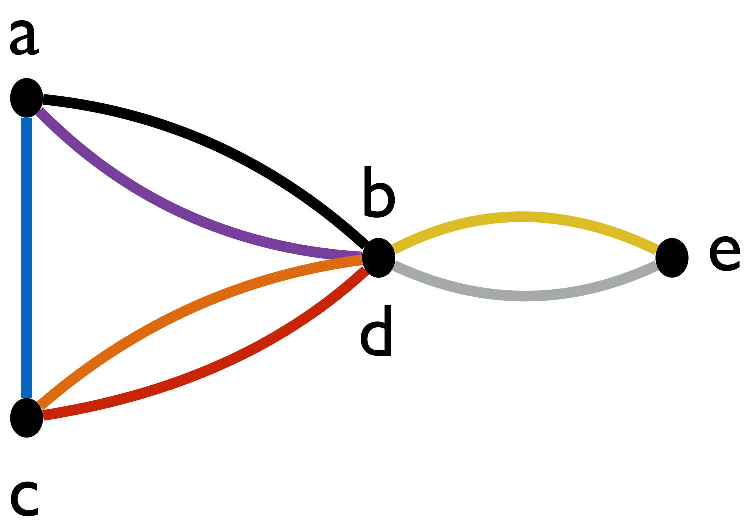
Select an edge randomly:

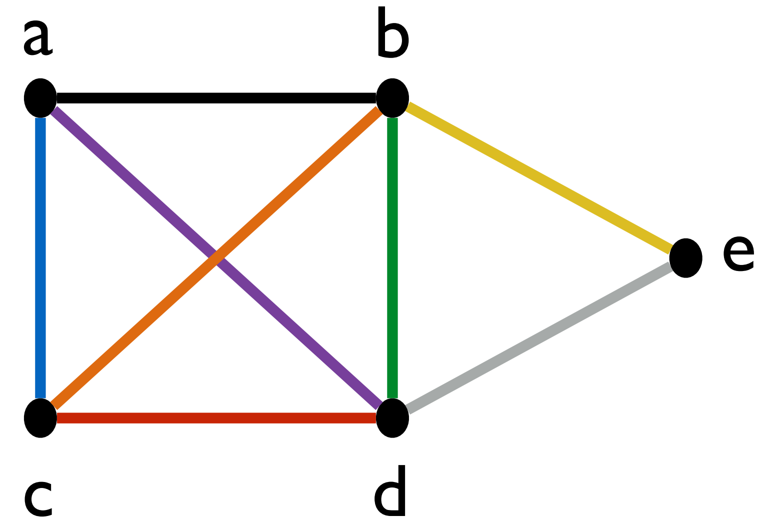
Green edge selected.

Contract that edge.

Contraction algorithm for min cut

Example run





Size of min-cut: 2

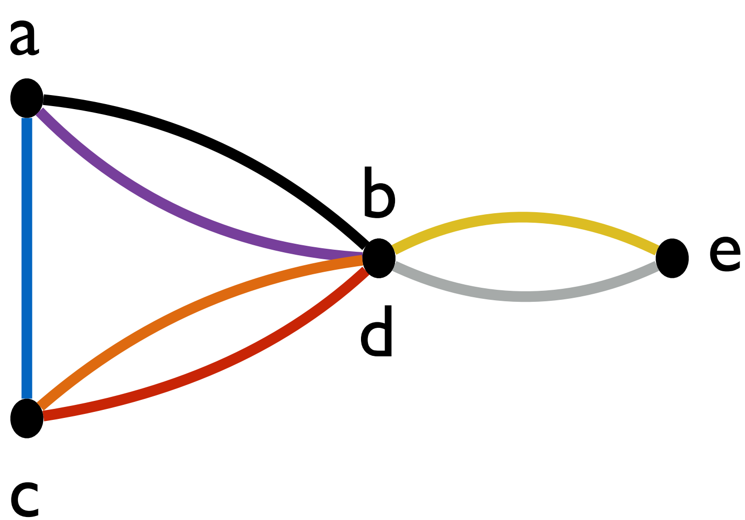
Select an edge randomly:

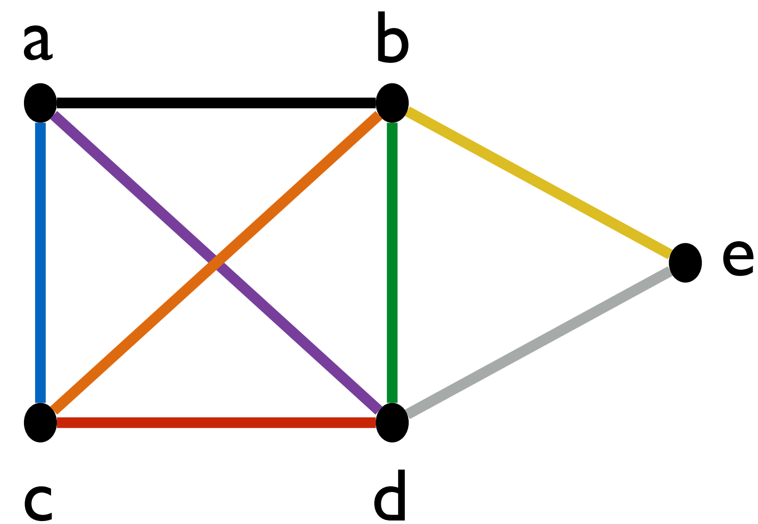
Green edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run





Size of min-cut: 2

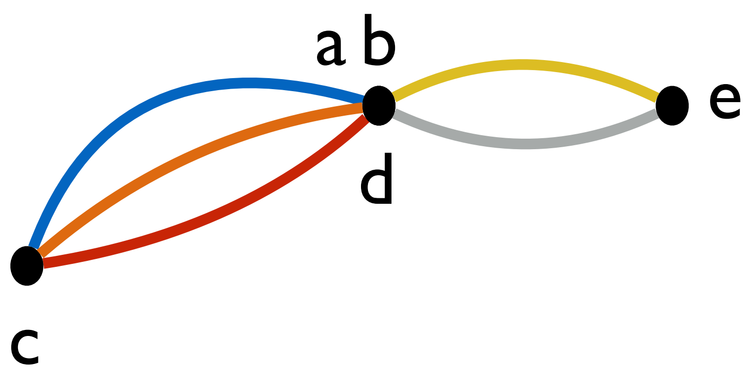
Select an edge randomly:

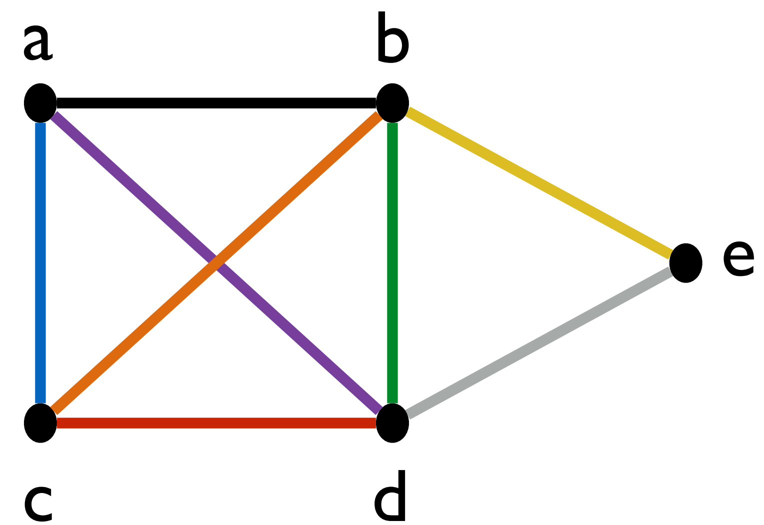
Purple edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run





Select an edge randomly:

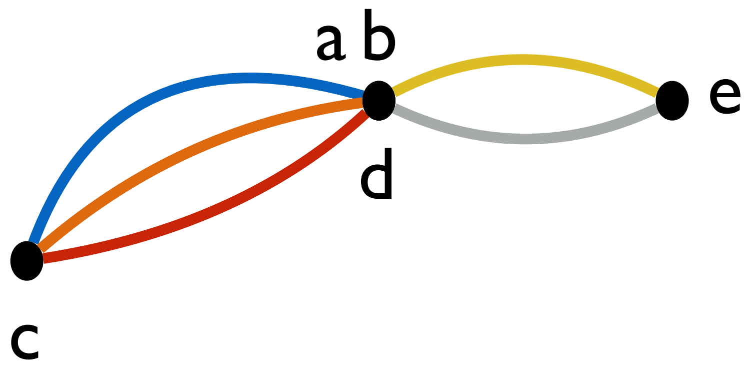
Size of min-cut: 2

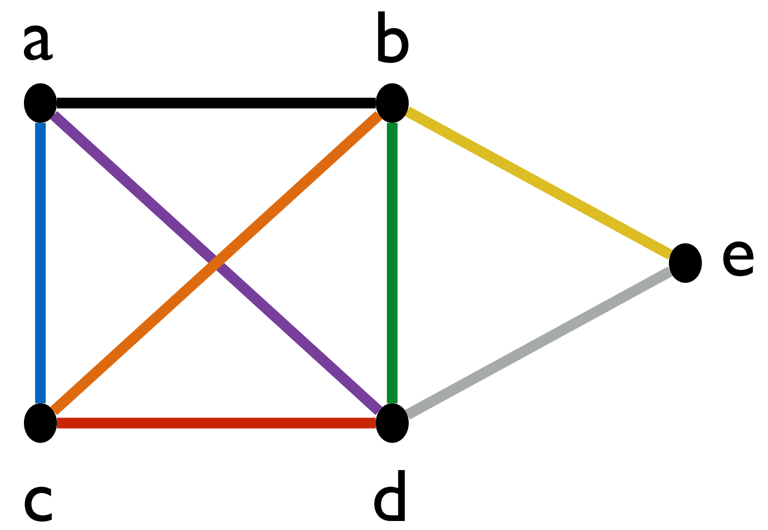
Purple edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run





Size of min-cut: 2

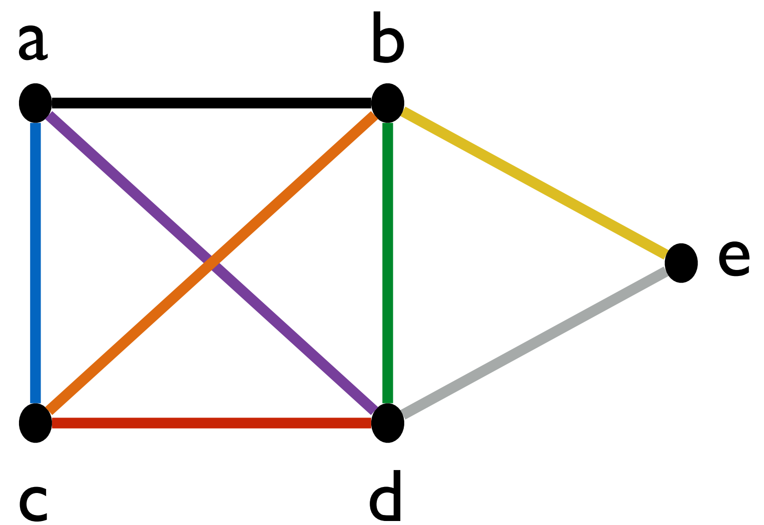
Select an edge randomly:

Blue edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run



Size of min-cut: 2

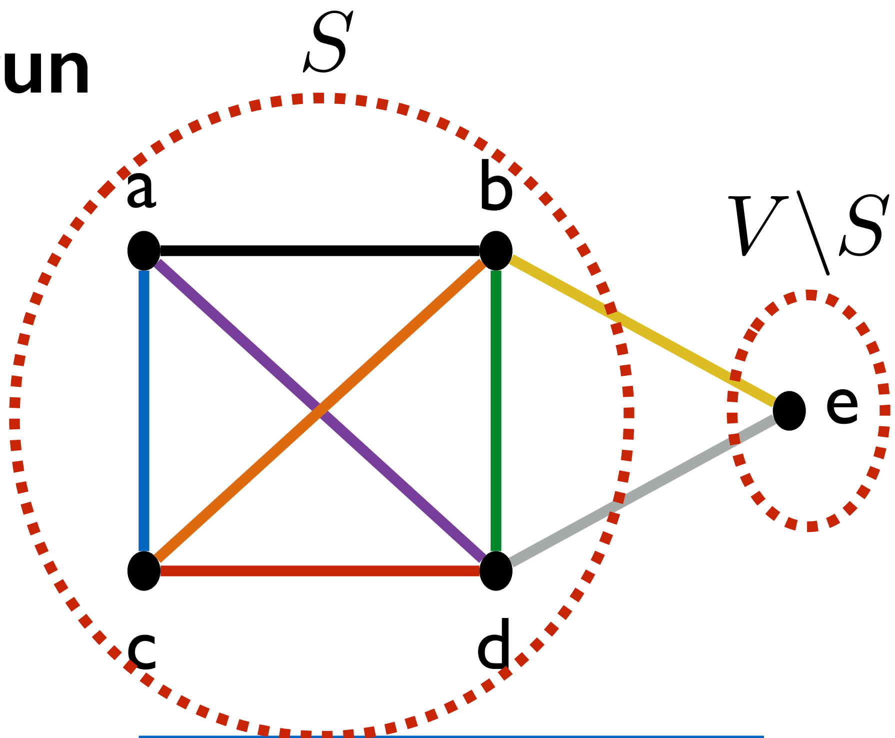
Select an edge randomly:

Blue edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run



Size of min-cut: 2

Select an edge randomly:

Blue edge selected.

Contract that edge. (delete self loops)

When two vertices remain, you have your cut:

S = {a, b, c, d} V\S = {e} *size:* 2

Contraction algorithm for min cut

$$G = G_0 \xrightarrow{\text{contract}} G_1 \xrightarrow{\text{contract}} G_2 \xrightarrow{\text{contract}} \cdots \xrightarrow{\text{contract}} G_{n-2}$$

G_0: n vertices

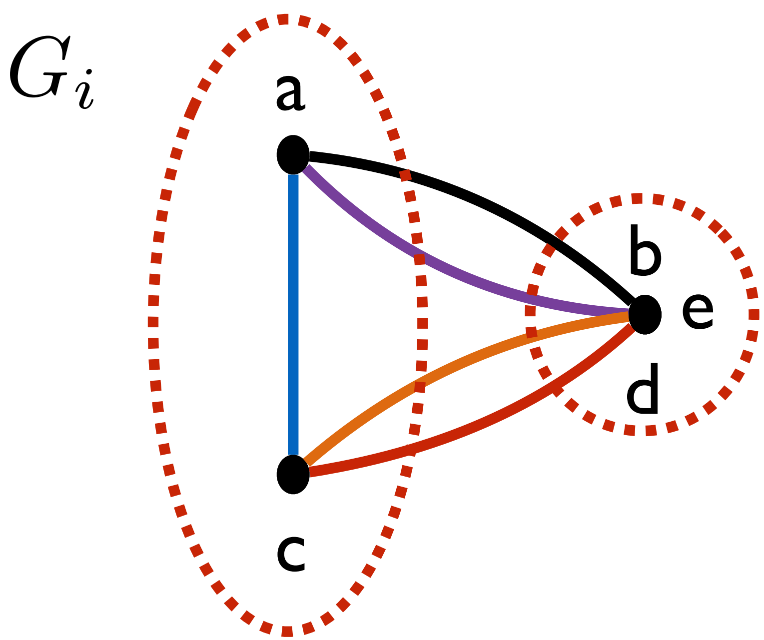
G_{n-2}: 2 vertices

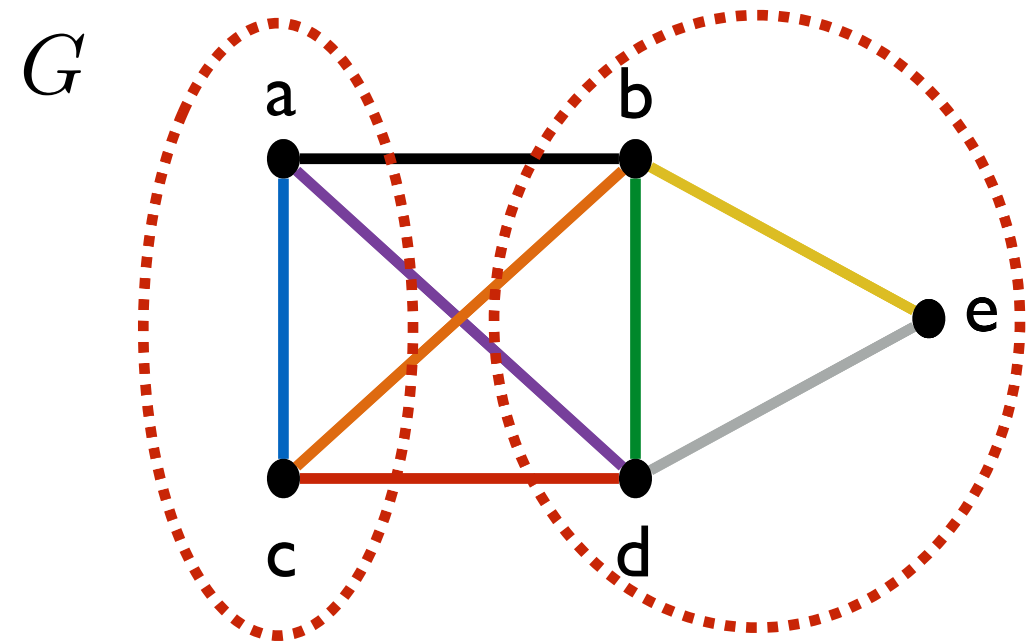
$n-2$ iterations

Contraction algorithm for min cut

Observation:

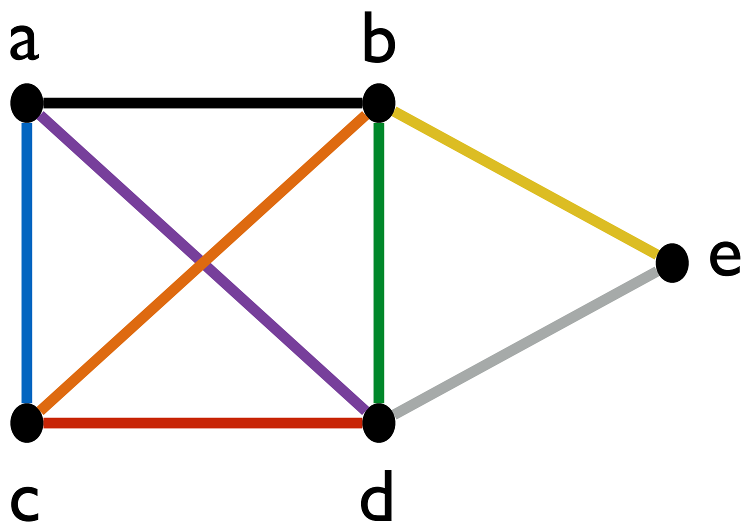
For any i: A cut in G_i of size k corresponds exactly to a cut in G of size k.

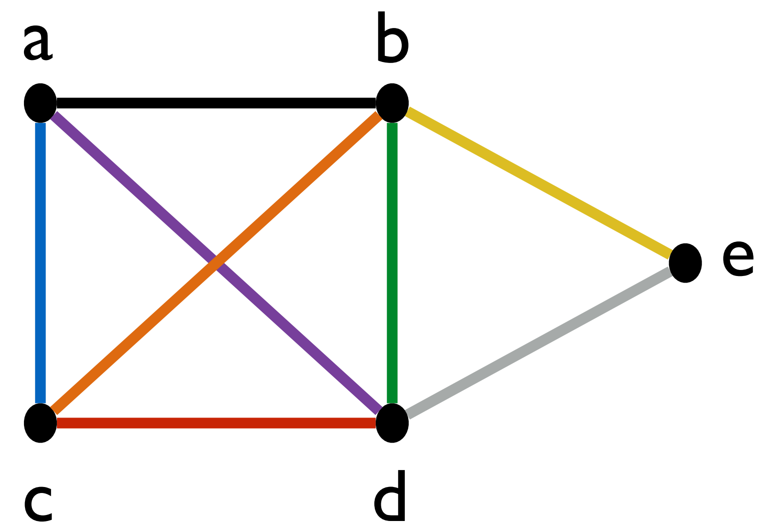




Contraction algorithm for min cut

Example run 2





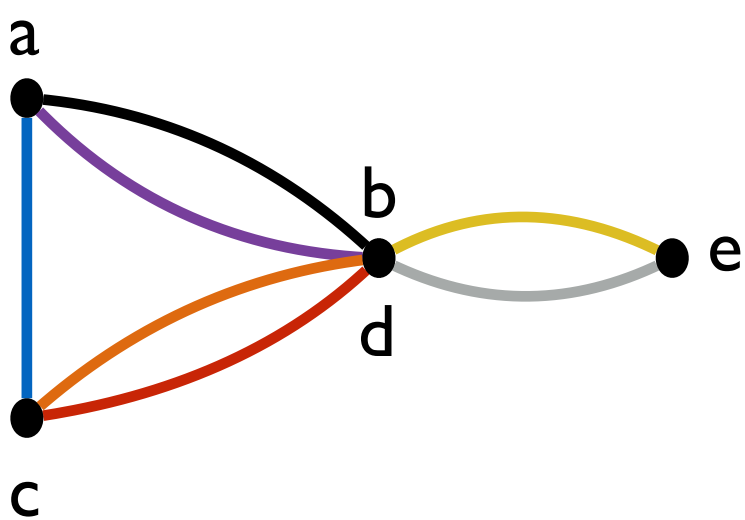
Select an edge randomly:

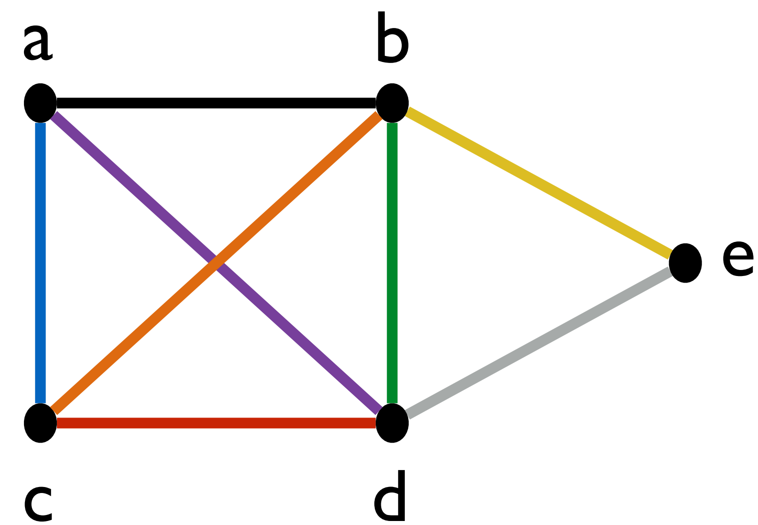
Green edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 2





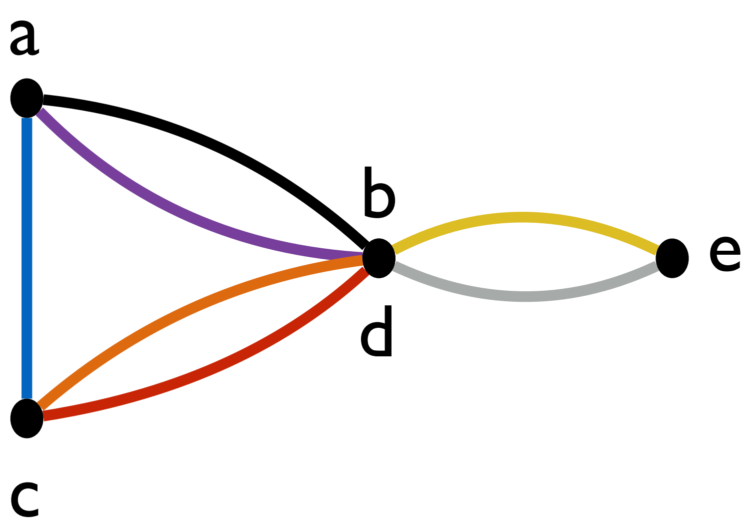
Select an edge randomly:

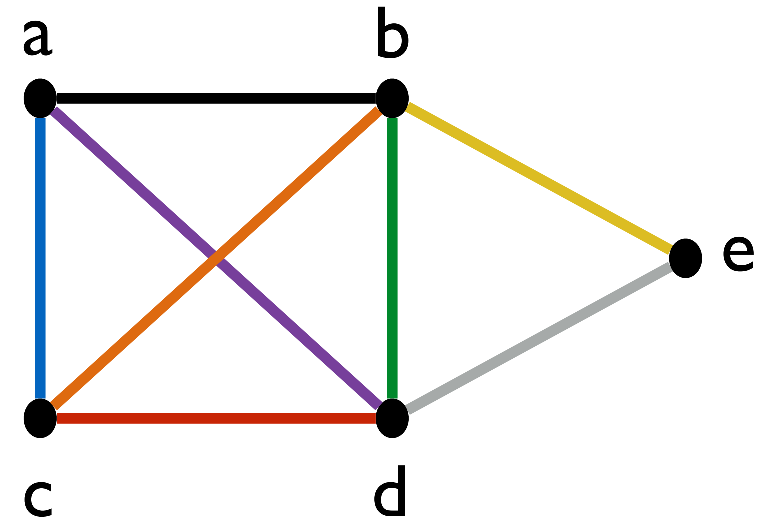
Green edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 2





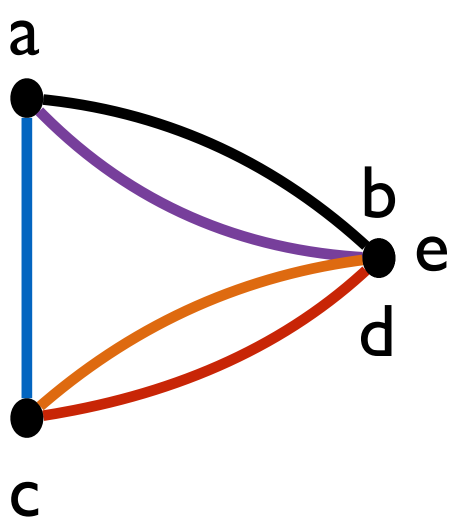
Select an edge randomly:

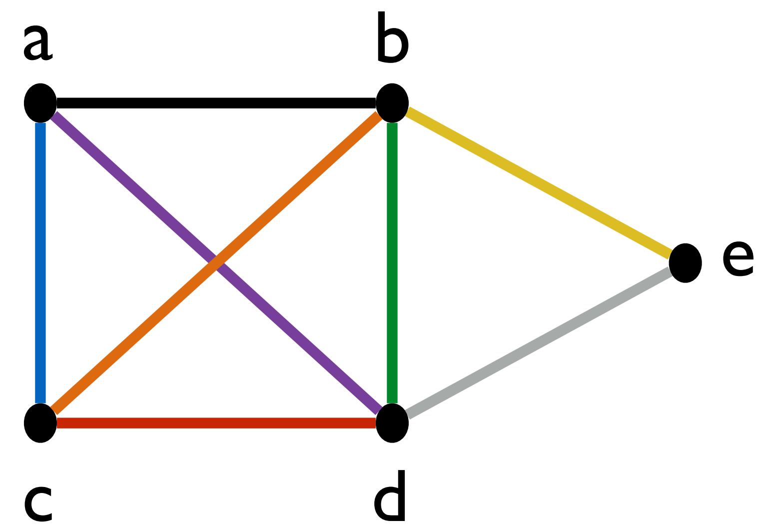
Yellow edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 2





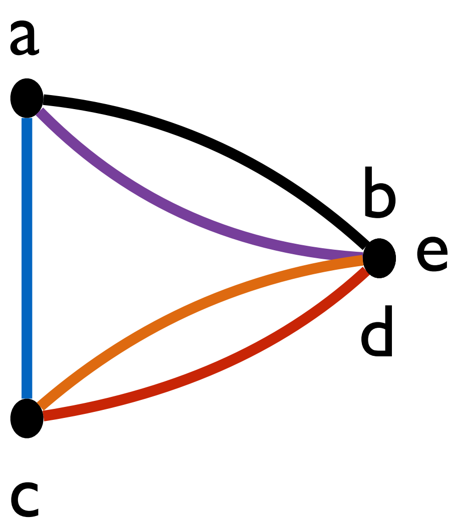
Select an edge randomly:

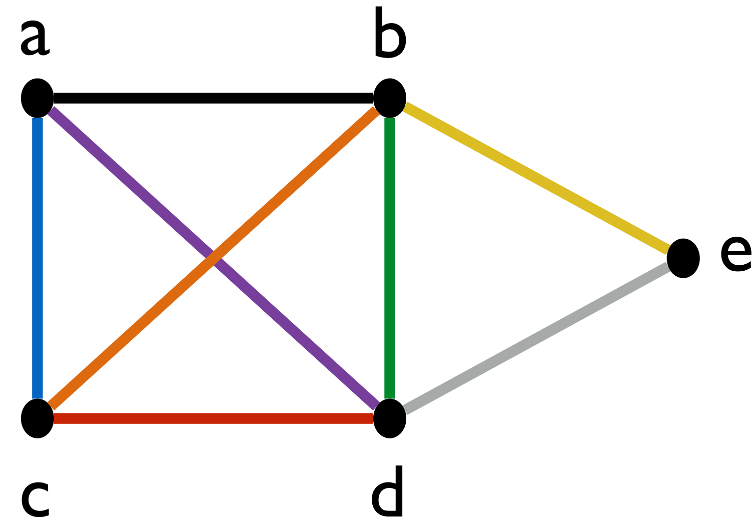
Yellow edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 2





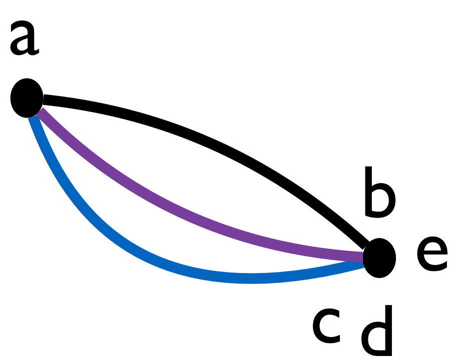
Select an edge randomly:

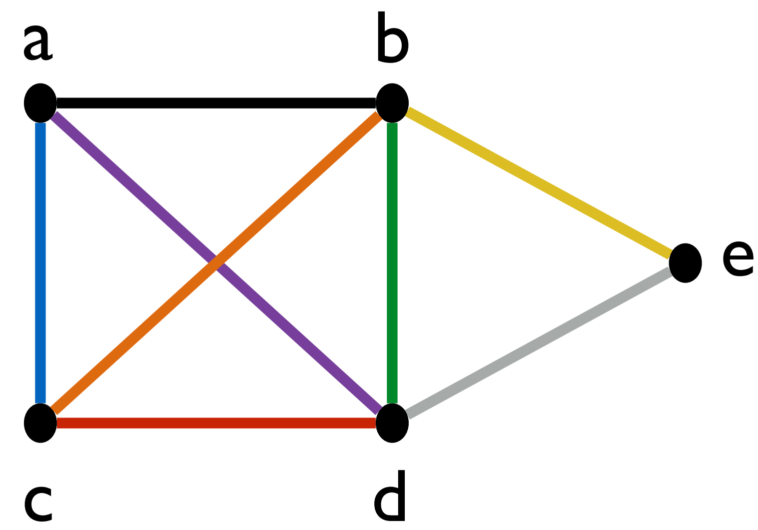
Red edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 2





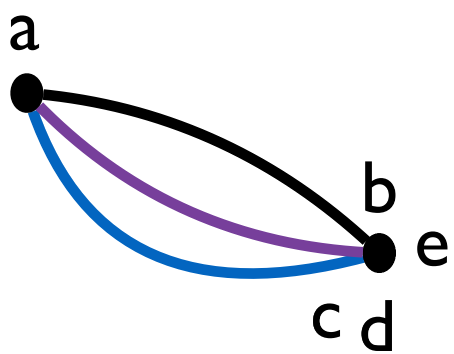
Select an edge randomly:

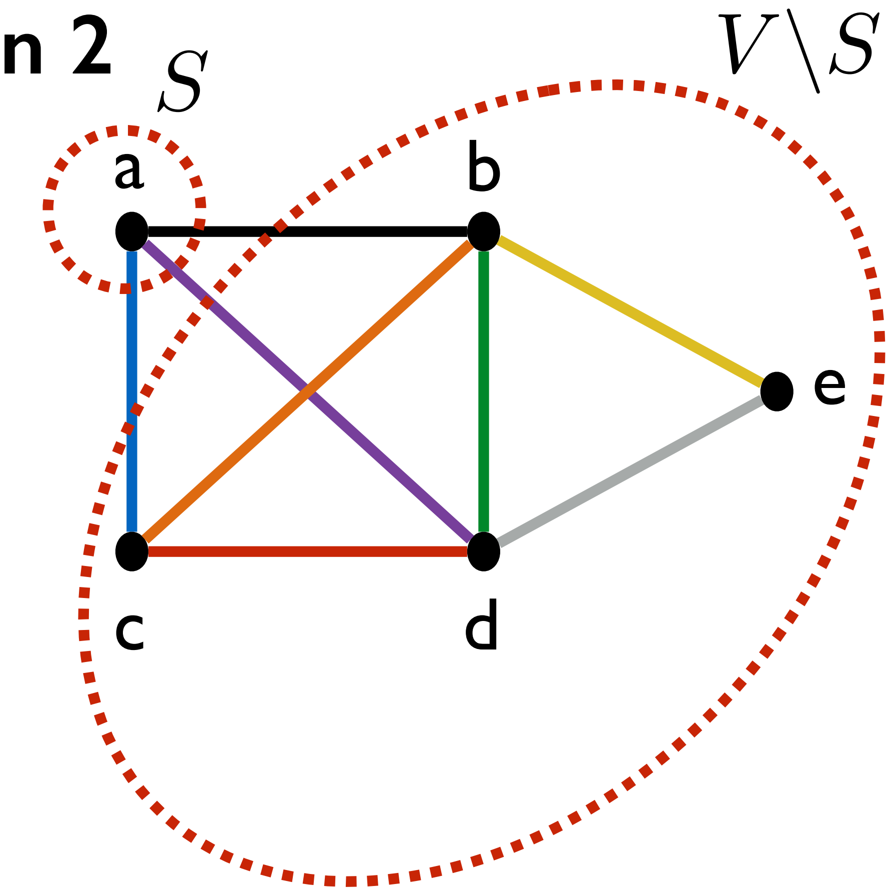
Red edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 2





Select an edge randomly:

Red edge selected.

Contract that edge. (delete self loops)

When two vertices remain, you have your cut:

S = {a} V\S = {b,c,d,e} *size: 3*

Contraction algorithm for min cut

Theorem:

Let $G = (V, E)$ be a graph with *n* vertices.
The probability that the contraction algorithm will output a min-cut is $\geq 1/n^2$.

Should we be impressed?

- The algorithm runs in polynomial time.
- There are exponentially many cuts. ($\approx 2^n$)
- There is a way to boost the probability of success to $1 - \frac{1}{e^n}$ (and still remain in polynomial time)

Pre-proof Poll

Let k be the size of a minimum cut.

Which of the following are true (can select more than one):

For $G = G_0$, $k \le \min_v \deg_G(v)$

For $G = G_0$, $k \ge \min_v \deg_G(v)$

For every G_i, $k \le \min_v \deg_{G_i}(v)$

For every G_i, $k \ge \min_v \deg_{G_i}(v)$

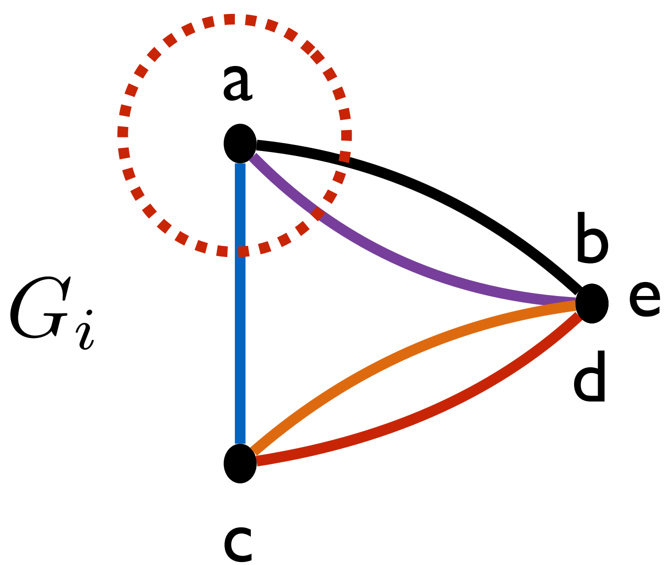
Pre-proof Poll Answer

For every G_i , $k \leq \min_v \deg_{G_i}(v)$

i.e., for every G_i and every $v \in G_i$, $k \leq \deg_{G_i}(v)$

Why?

A single vertex v forms a cut of size $\deg(v)$.



This cut has size $\deg(a) = 3$.

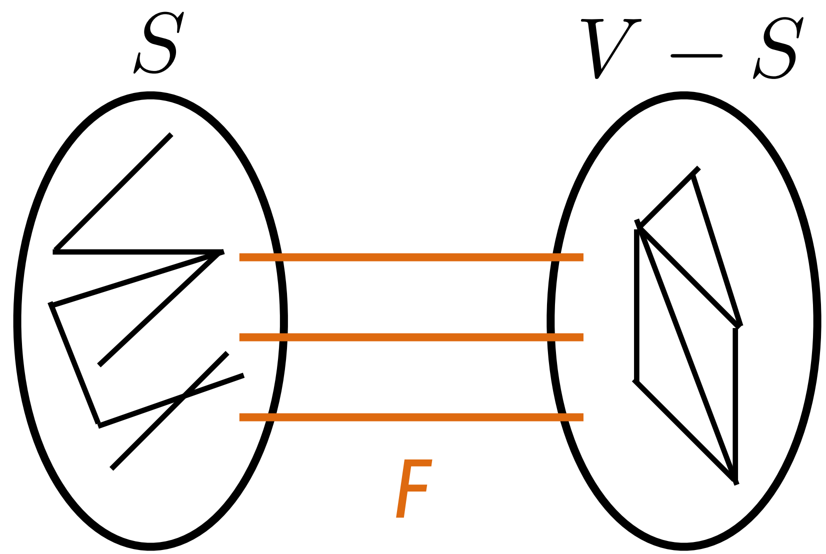
Same cut exists in original graph.

So $k \leq 3$.

Proof of theorem

Fix some minimum cut.

$|F| = k$
$|V| = n$
$|E| = m$



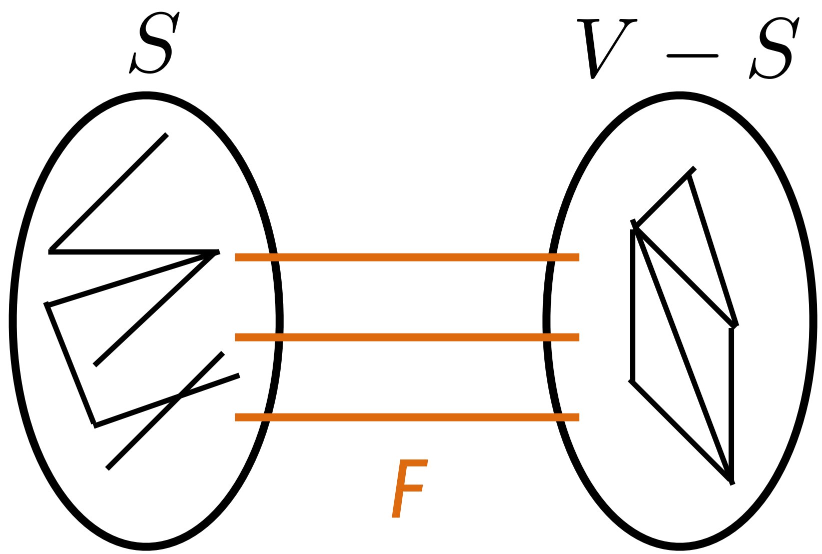
Will show $\Pr[\text{algorithm outputs } F] \geq 1/n^2$

(Note $\Pr[\text{success}] \geq \Pr[\text{algorithm outputs } F]$)

Proof of theorem

Fix some minimum cut.

$|F| = k$
$|V| = n$
$|E| = m$



When does the algorithm output F ?

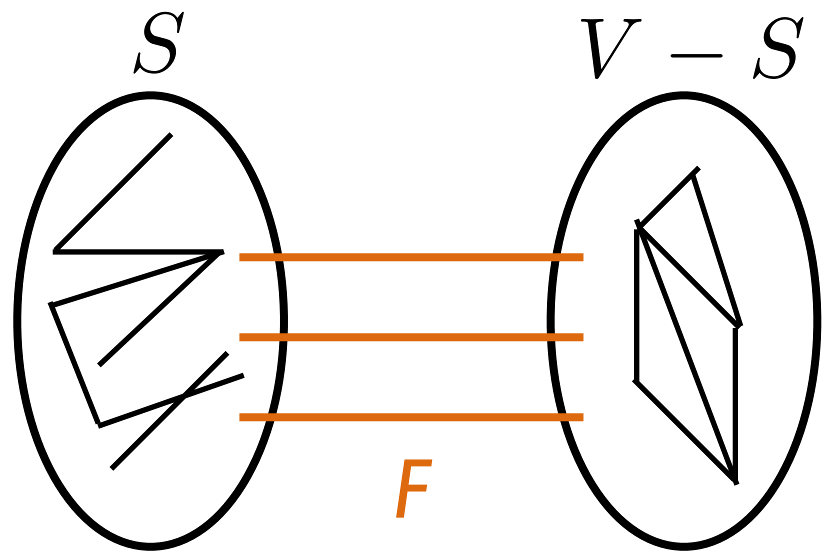
What if it never picks an edge in F to contract?
Then it will output F.

What if the algorithm picks an edge in F to contract?
Then it cannot output F.

Proof of theorem

Fix some minimum cut.

$|F| = k$
$|V| = n$
$|E| = m$



$$\Pr[\text{success}] \geq$$

$$\Pr[\text{algorithm outputs } F] =$$

$$\Pr[\text{algorithm never contracts an edge in } F] =$$

$$\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}]$$

E_i = "an edge in F is contracted in iteration i."

Proof of theorem

E_i = "an edge in *F* is contracted in iteration i."

Goal: $\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}] \geq 1/n^2.$

$$\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}]$$

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

$$\Pr[\overline{E_1}] = 1 - \Pr[E_1] = 1 - \frac{\#\text{ edges in } F}{\text{total } \#\text{ edges}} = 1 - \frac{k}{m}$$

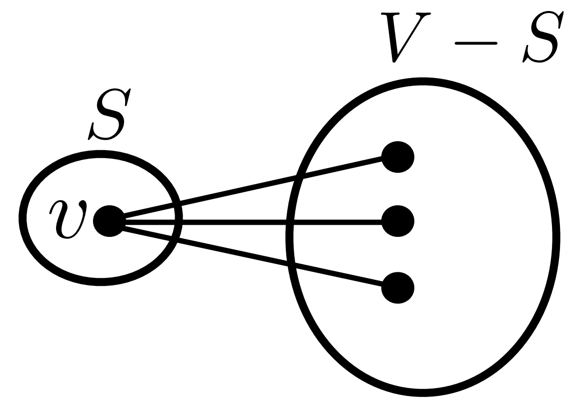
want to write in terms of *k* and *n*

Proof of theorem

E_i = "an edge in *F* is contracted in iteration i."

Goal: $\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}] \geq 1/n^2.$

Observation: $\forall v \in V \;:\; k \leq \deg(v)$



Recall: $\sum_{v \in V} \deg(v) = 2m \implies 2m \geq kn$

$\sum_{v \in V} \deg(v) \geq kn$

$$\implies m \geq \frac{kn}{2}$$

$$\Pr[\overline{E_1}] = 1 - \frac{k}{m} \geq 1 - \frac{k}{kn/2} = \left(1 - \frac{2}{n}\right)$$

Proof of theorem

E_i = "an edge in *F* is contracted in iteration i."

Goal: $\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}] \geq 1/n^2.$

$$\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}]$$
$$\geq \left(1 - \frac{2}{n}\right) \cdot \Pr[\overline{E_2}|\overline{E_1}] \cdot \Pr[\overline{E_3}|\overline{E_1} \cap \overline{E_2}] \cdots$$
$$\Pr[\overline{E_{n-2}}|\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-3}}]$$

$$\Pr[\overline{E_2}|\overline{E_1}] = 1 - \Pr[E_2|\overline{E_1}] = 1 - \frac{k}{\#\text{ remaining edges}}$$

want to write in terms of *k* and *n*

Proof of theorem

E_i = "an edge in *F* is contracted in iteration i."

Goal: $\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}] \geq 1/n^2$.

Let $G' = (V', E')$ be the graph after iteration 1.

Observation: $\forall v \in V' \ : \ k \leq \deg_{G'}(v)$

$$\sum_{v \in V'} \deg_{G'}(v) = 2|E'| \implies 2|E'| \geq k(n-1)$$

$\geq k(n-1)$

$$\implies |E'| \geq \frac{k(n-1)}{2}$$

$$\Pr[\overline{E_2}|\overline{E_1}] = 1 - \frac{k}{|E'|} \geq 1 - \frac{k}{k(n-1)/2} = \left(1 - \frac{2}{n-1}\right)$$

Proof of theorem

E_i = "an edge in *F* is contracted in iteration i."

Goal: $\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}] \geq 1/n^2.$

$$\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}]$$

$$\geq \left(1 - \frac{2}{n}\right) \cdot \left(1 - \frac{2}{n-1}\right) \cdot \Pr[\overline{E_3} | \overline{E_1} \cap \overline{E_2}] \cdots$$

$$\Pr[\overline{E_{n-2}} | \overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-3}}]$$

Proof of theorem

E_i = "an edge in F is contracted in iteration i."

Goal: $\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}] \geq 1/n^2.$

$$\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right)\cdots\left(1 - \frac{2}{n-(n-4)}\right)\left(1 - \frac{2}{n-(n-3)}\right)$$

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

$$= \frac{2}{n(n-1)} \geq \frac{1}{n^2}$$

$\square$

Interlude: how can you generate random bits?

Google | true random bits

Web | Shopping | Videos | Images | News | More ▾ | Search tools

About 2,790,000 results (0.51 seconds)

RANDOM.ORG - True Random Number Service
https://www.random.org/ ▾ Random.org ▾
ORG offers **true random** numbers to anyone on the Internet. The randomness comes ... ORG has generated 1.19 trillion random **bits** for the Internet community.
List Randomizer - Integer Generator - Lottery Quick Pick - Dice Roller

Introduction to Randomness and Random Numbers
https://www.random.org/randomness/ ▾ Random.org ▾
ORG is a **true random** number service that generates randomness via .... which gathers random **bits** from a variety of sources including **HotBits** and RANDOM.

How To Generate Truly Random Bits
openfortress.org/cryptodoc/random/ ▾
A guide to generate cryptographically acceptable **true random bits**.

HotBits: Genuine Random Numbers - Fourmilab
https://www.fourmilab.ch/hotbits/ ▾
Genuine **random** numbers, generated by radioactive decay. ... People working with computers often sloppily talk about their system's "**random** number generator" and the "**random** numbers" it produces. ... HotBits are generated by timing successive pairs of radioactive decays detected by ...
HotBits Hardware Description - HotBits Hardware - How HotBits Works

Hardware random number generator - Wikipedia, the free ...
https://en.wikipedia.org/.../Hardware_random_number_genera... ▾ Wikipedia ▾
In computing, a hardware random number generator (TRNG, **True Random** ... number generators generally produce a limited number of random **bits** per second.

Random number generation - Wikipedia, the free ...
https://en.wikipedia.org/wiki/Random_number_generation ▾ Wikipedia ▾
Jump to "**True**" vs. pseudo-**random** numbers - **True**" vs. pseudo-**random** numbers[edit]
... as filling a hard disk drive with **random bits**, ...

Pseudorandom number generator - Wikipedia, the free ...
https://en.wikipedia.org/wiki/Pseudorandom_number_generator ▾ Wikipedia ▾
The PRNG-generated sequence is not **truly random**, because it is completely ... The period is bounded by the number of the states, usually measured in **bits**.

Quantum Random Bit Generator Service
random.irb.hr/ ▾
of used random numbers. Since **true random** numbers are impossible to generate with

radioactive decay

atmospheric noise

photons measurement

Contraction algorithm for min cut

Theorem:

Let $G = (V, E)$ be a graph with *n* vertices.
The probability that the contraction algorithm will output a min-cut is $\geq 1/n^2$.

Should we be impressed?

- The algorithm runs in polynomial time.
- There are exponentially many cuts. ($\approx 2^n$)
- There is a way to boost the probability of success to

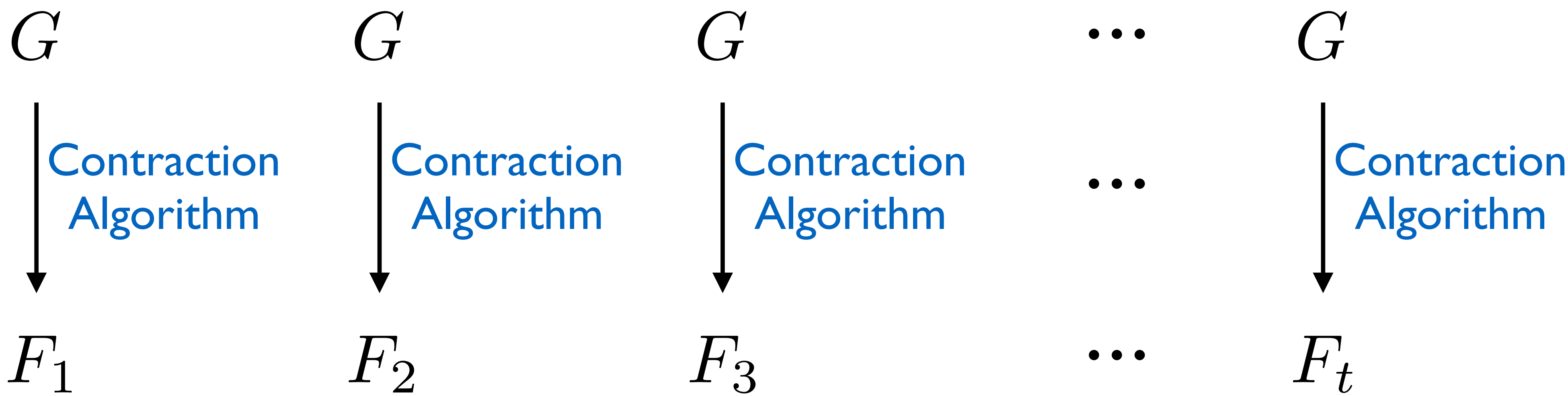
$$1 - \frac{1}{e^n}$$ (and still remain in polynomial time)

Boosting Phase

Boosting phase

Run the algorithm *t* times using fresh random bits.

Output the smallest cut among the ones you find.



Output the minimum among F_i's.

larger $t \implies$ better success probability

What is the relation between t and success probability?

Boosting phase

What is the relation between t and success probability?

Let A_i = "in the i'th repetition, we **don't** find a min cut."

$$\Pr[\text{error}] = \Pr[\text{don't find a min cut}]$$

$$= \Pr[A_1 \cap A_2 \cap \cdots \cap A_t]$$

ind. events

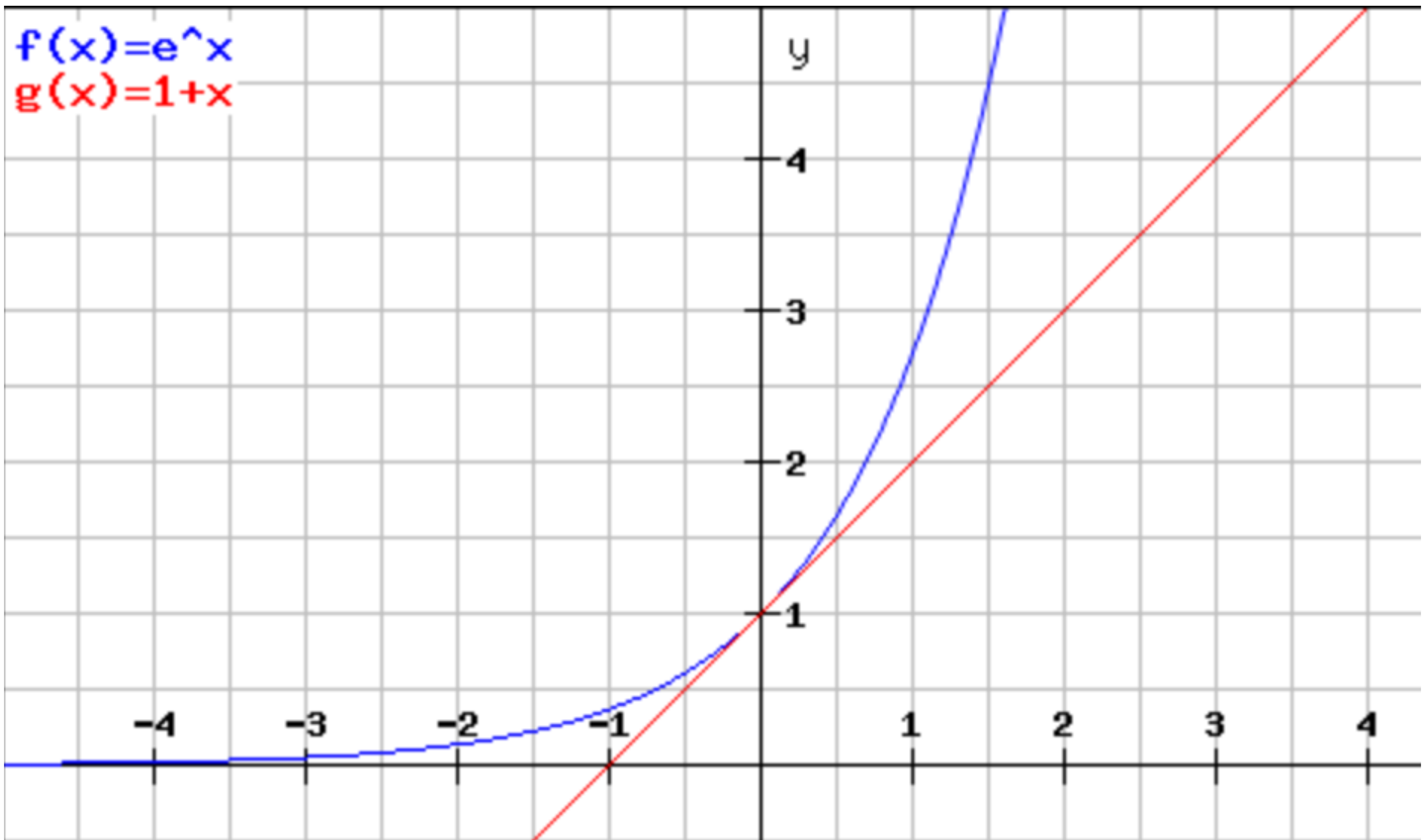
$$= \Pr[A_1]\Pr[A_2]\cdots\Pr[A_t]$$

$$= \Pr[A_1]^t \leq \left(1 - \frac{1}{n^2}\right)^t$$

Boosting phase

$$\Pr[\text{error}] \leq \left(1 - \frac{1}{n^2}\right)^t$$

<u>World's most useful inequality:</u> $\forall x \in \mathbb{R}: \; 1 + x \leq e^x$



Boosting phase

$$\Pr[\text{error}] \leq \left(1 - \frac{1}{n^2}\right)^t$$

World's most useful inequality: $\forall x \in \mathbb{R}: \; 1 + x \leq e^x$

Let $x = -1/n^2$

$$\Pr[\text{error}] \leq (1+x)^t \leq (e^x)^t = e^{xt} = e^{-t/n^2}$$

$$t = n^3 \implies \Pr[\text{error}] \leq e^{-n^3/n^2} = 1/e^n \implies$$

$$\Pr[\text{success}] \geq 1 - \frac{1}{e^n}$$

Conclusion for min cut

We have a polynomial-time algorithm that solves the min cut problem with probability $1 - 1/e^n$.

↓

Theoretically, not equal to 1.
Practically, equal to 1.

Important Note

Boosting is not specific to Min-cut algorithm.

We can boost the success probability of Monte Carlo algorithms via repeated trials.

Final remarks

Randomness adds an interesting dimension to computation.

Randomized algorithms can be faster and more elegant than their deterministic counterparts.

There are some interesting decision problems for which:
- there is a poly-time randomized algorithm,
- we can't find a poly-time deterministic algorithm.

Another (morally) million dollar question:

Is $\mathsf{P} = \mathsf{BPP}$?