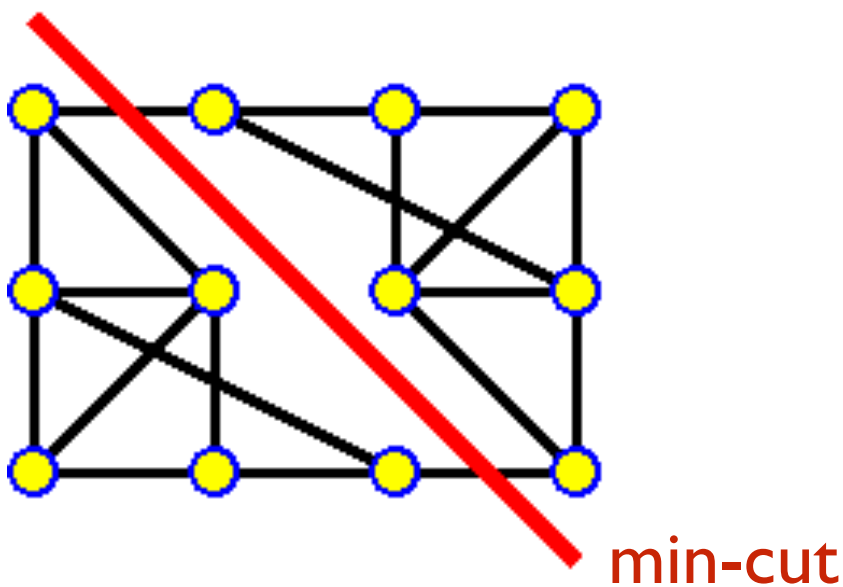
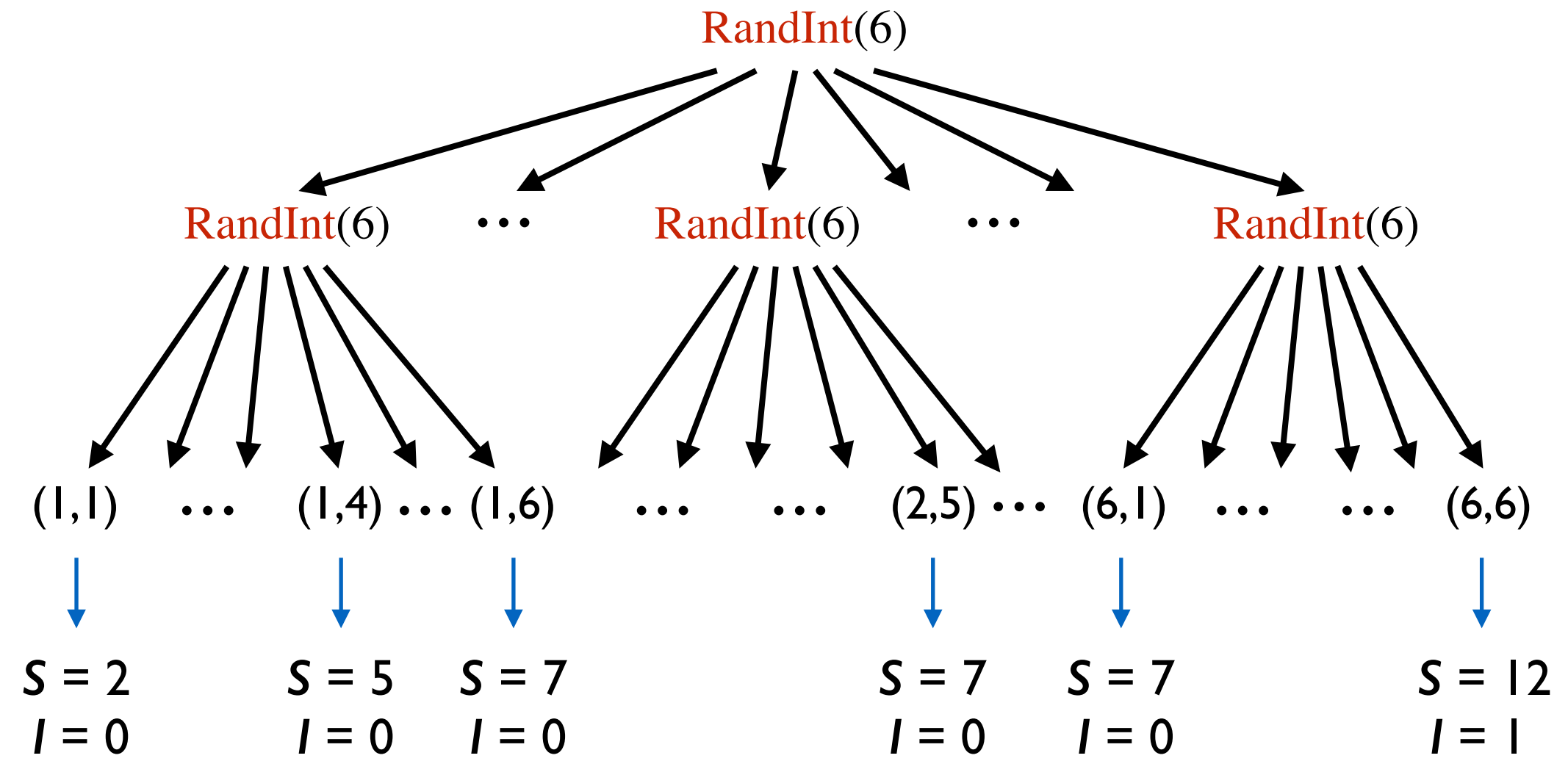
15-251
Great Theoretical Ideas in Computer Science

Lecture 23:
Randomized Algorithms Continued

April 11th, 2017



```
S <— RandInt(6) + RandInt(6)
if S = 12:   I <— 1
else:        I <— 0
```



Formal Definition: Deterministic algorithm

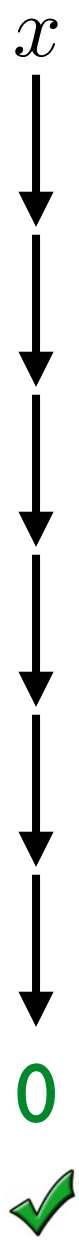
Let $f : \Sigma^* \to \Sigma^*$ be a computational problem.

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

$$\boxed{\forall x \in \Sigma^*,} \qquad A(x) = f(x)$$

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

Picture:



Deterministic:

Each input x induces a deterministic path.

Length of the path is $\leq T(|x|)$.

Formal Definition: Monte Carlo algorithm

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

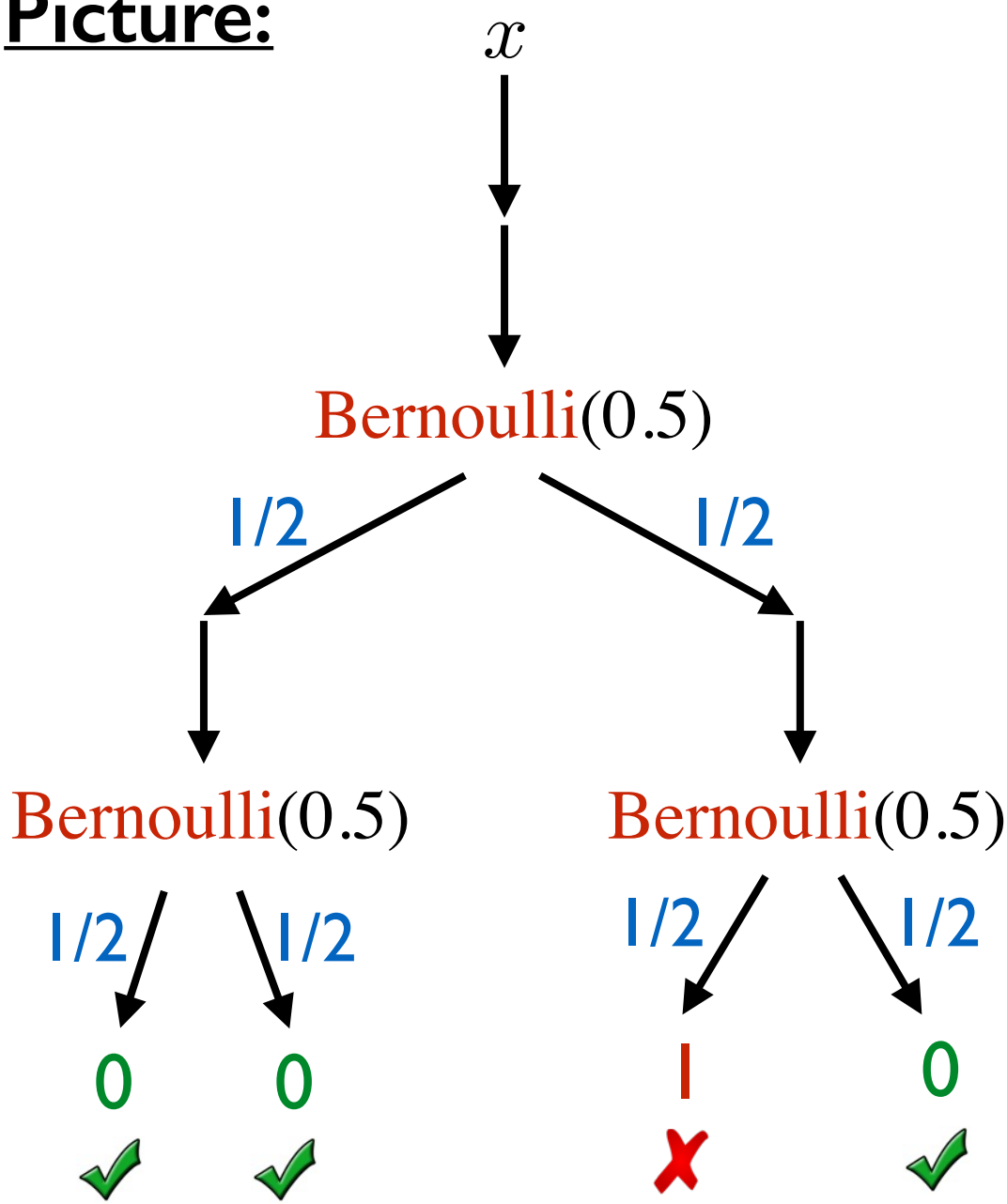
We say that randomized algorithm A
is a $T(n)$-time **Monte Carlo algorithm** for f
with ϵ error probability if:

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

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

(no matter what the random choices are
i.e. no matter what leaf we end up in.)

Picture:



Monte Carlo:

Each input x induces a probability tree.

Probability of incorrect leaves $\leq \epsilon$.

Longest path to a leaf $\leq T(|x|)$.

Formal Definition: Las Vegas algorithm

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

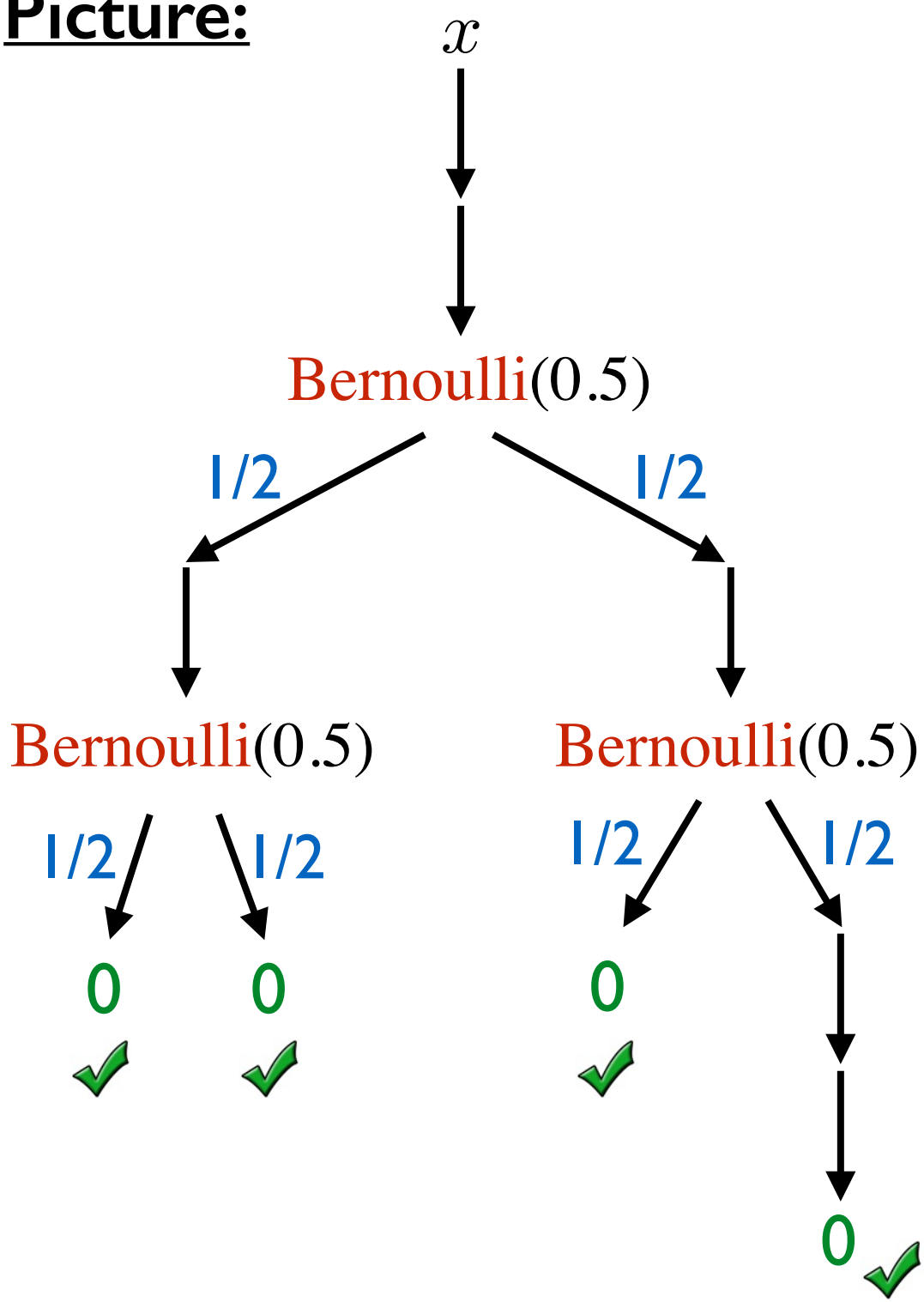
We say that randomized algorithm A
is a $T(n)$-time **Las Vegas algorithm** for f if:

$$\forall x \in \Sigma^*, \qquad \mathbf{Pr}[A(x) = f(x)] = 1$$

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

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

Picture:



Las Vegas:

Each input x induces a probability tree.

No incorrect leaves.

Expected length of a path to a leaf

$$\leq T(|x|).$$

EXAMPLES

3 IMPORTANT PROBLEMS

Integer Factorization

Input: integer N

Ouput: a prime factor of N

isPrime

Input: integer N

Ouput: True if N is prime.

Generating a random n-bit prime

Input: integer n

Ouput: a random n-bit prime

Most crypto systems start like:

- pick two random n-bit primes P and Q.
- let N = PQ. (N is some kind of a "key")
- (more steps…)

We should be able to do efficiently the following:

- check if a given number is prime.
- generate a random prime.

We should **not** be able to do efficiently the following:

- given N, find P and Q. (the system is broken if we can do this!!!)

isPrime

```
def isPrime(N):
    if (n < 2):  return False
    for factor in range(2, N):
        if (N % factor == 0):  return False
    return True
```

Problems:

- exponential running time.

isPrime

```
def isPrime(N):
    if (n < 2):  return False
    maxFactor = round(N**0.5)
    for factor in range(2, maxFactor+1):
        if (N % factor == 0):  return False
    return True
```

Problems:

- exponential running time.
- tries to factor the input.

isPrime

Amazing result from 2002:

There is a poly-time algorithm for isPrime.

Agrawal, Kayal, Saxena

undergraduate students at the time

However, best known implementation is ~ $O(n^6)$ time.

Not feasible when $n = 2048$.

isPrime

So that's **not** what we use in practice.

Everyone uses the Miller-Rabin algorithm (1975).

CMU
Professor

The running time is ~ $O(n^2)$.

It is a Monte Carlo algorithm with tiny error probability (say $1/2^{300}$).

Generating a random prime

```
repeat:
    let N be a random n-bit number
    if isPrime(N): return N
```

Prime Number Theorem (informal):

About 1/n fraction of n-bit numbers are prime.

$\Longrightarrow$ expected run-time of the above algorithm ~ $O(n^3)$.

No poly-time deterministic algorithm is known to generate an n-bit prime!!!

Polymath projects:

Massively collaborative online mathematical projects

Gowers's Weblog

Mathematics related discussions

« A Tricki issue

Background to a Polymath project »

Is massively collaborative mathematics possible?

Of course, one might say, there are certain kinds of problems that lend themselves to huge collaborations. One has only to think of the proof of the classification of finite simple groups, or of a rather different kind of example such as a search for a new largest prime carried out during the downtime of thousands of PCs around the world. But my question is a different one. What about the solving of a problem that does not naturally split up into a vast number of subtasks? Are such problems best tackled by n people for some n that belongs to the set $\{1, 2, 3\}$? (Examples of famous papers with four authors do not count as an interesting answer to this question.)

• • •

Timothy Gowers

The polymath blog

July 27, 2009

Proposal: deterministic way to find primes

Filed under: finding primes, polymath proposals, research — Terence Tao @ 2:24 am

Here is a proposal for a polymath project:

> **Problem.** Find a *deterministic* algorithm which, when given an integer k, is guaranteed to find a prime of at least k digits in length of time polynomial in k. You may assume as many standard conjectures in number theory (e.g. the generalised Riemann hypothesis) as necessary, but avoid powerful conjectures in complexity theory (e.g. P=BPP) if possible.

The point here is that we have no explicit formulae which (even at a conjectural level) can quickly generate large prime numbers. On the other hand, given any specific large number n, we can test it for primality in a deterministic manner in a time polynomial in the number of digits (by the AKS primality test). This leads to a *probabilistic* algorithm to quickly find k-digit primes: simply select k-digit numbers at random, and test each one in turn for primality. From the prime number theorem, one is highly likely to eventually hit on a prime after about O(k) guesses, leading to a polynomial time algorithm. However, there appears to be no obvious way to derandomise this algorithm.

Now, given a sufficiently strong pseudo-random number generator – one which was computationally indistinguishable from a genuinely random number generator – one could derandomise this algorithm (or indeed, any algorithm) by substituting the random number generator with the pseudo-random one. So, given sufficiently strong conjectures in complexity theory (I don't think P=BPP is quite sufficient, but there are stronger hypotheses than this which would work), one could solve the problem.

Cramer conjectured that the largest gap between primes in [N,2N] is of size $O(\log^2 N)$. Assuming this conjecture, then the claim is

• • •

133 Comments

1. This is certainly an interesting problem, from a pure mathematician's perspective anyway. (From a practical point of view, one would be happy with a randomized algorithm, but there's no denying that to find a deterministic algorithm for such a basic problem would be a great achievement if it could be done.)

   My first reaction to the question of whether it is feasible was that it seems to be the kind of problem where what is needed is a clever idea that comes seemingly out of the blue and essentially cracks the problem in one go. Or alternatively, it might need a massive advance in number theory, such as a proof of Cramer's conjecture. But on further reflection, I can see that there might be other avenues to explore, such as a clever use of GRH to show that there is *some* set (not necessarily an interval) that must contain a prime. Even so, it feels to me as though this project might not last all that long before it was time to give up. But you've probably thought about it a lot harder and may see a number of interesting angles that I don't expect, so this thought is not necessarily to be taken all that seriously. And perhaps the magic of Polymath would lead to unexpected lines of thought — in a sense, that is the whole point of doing things collectively.

   This raises another issue. It might be that a Polymath project dwindles after a while, but then someone has an idea that suddenly makes it seem feasible again. In such cases, it might be good to have a list that people could subscribe to, where a moderator for a given project could, at his or her discretion, decide to email everyone on the list to tell them that there was a potentially important comment.

   6 1 Rate This

   Comment by gowers — July 27, 2009 @ 1:47 pm

   Dear Tim,

   Actually I've only thought about this problem since Bremen, where I mentioned it in my talk. As I mentioned in my post, I doubt a polymath would be able to solve it unconditionally, but perhaps some partial result could be made – for instance, one could replace the primes by some other similarly dense set (almost primes is an obvious candidate, as sieve theory techniques become available, though the resolution of sieves seems too coarse still). Another question is whether this derandomisation result would be implied by P=BPP; I mistakenly thought this to be the case back in Bremen, but realised afterwards that P and BPP refer to decision problems rather than to search problems, and so I was not able to make the argument work properly.

   Regarding notification: we already have RSS feeds that kind of do this job already. For instance, if one has a feed

CASE STUDY

Monte Carlo Algorithm for Min Cut

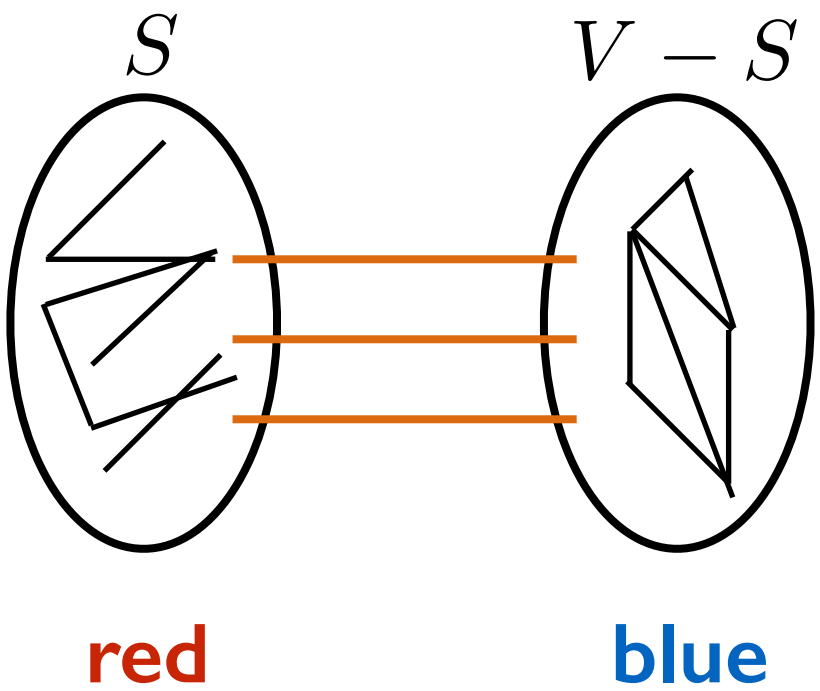
Gambles with **correctness**.

Doesn't gamble with **run-time**.

Cut Problems

Max Cut Problem (Ryan O'Donnell's favorite problem):

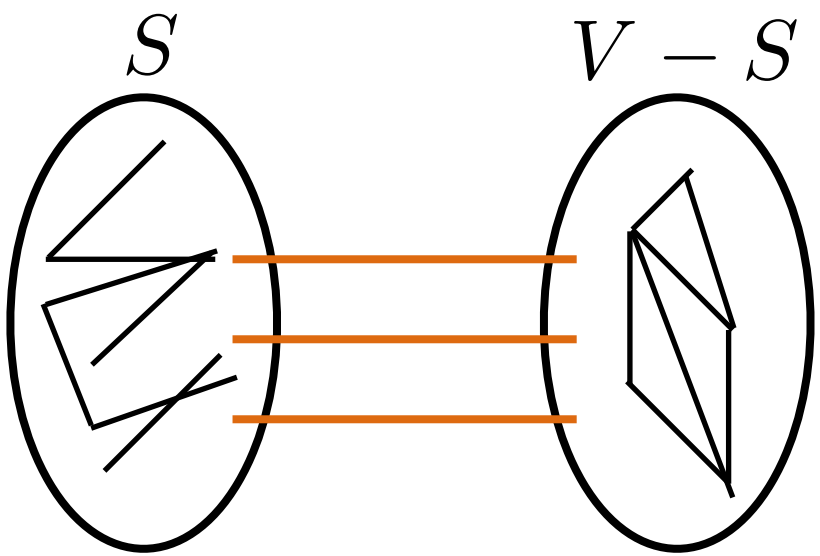
Given a connected graph $G = (V, E)$,
color the vertices **red** and **blue** so that the number of edges with two colors (e = {**u**,**v**}) is maximized.



Cut Problems

Max Cut Problem (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.



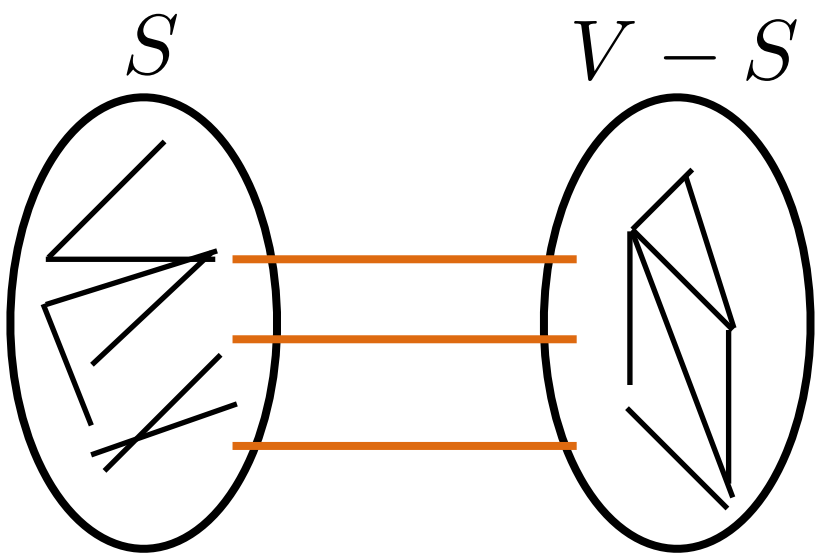
size of the cut = # edges from S to $V - S$.

Max Cut Problem is NP-complete!

Cut Problems

Min Cut Problem (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 ***minimized***.



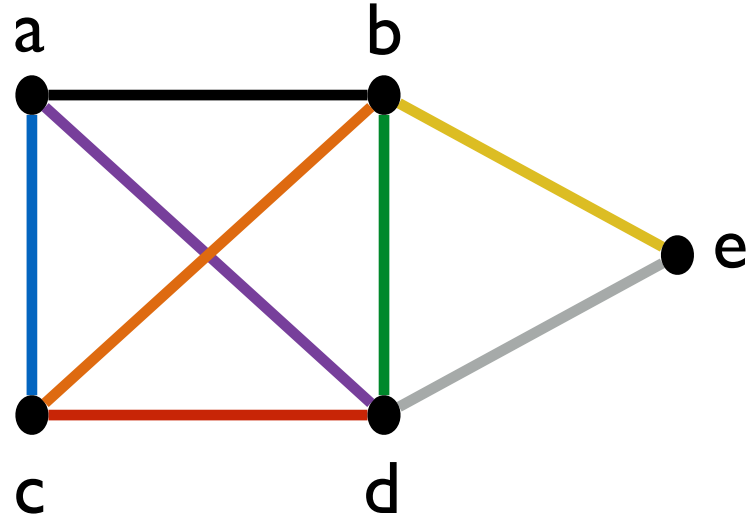
size of the cut = # edges from S to $V - S$.

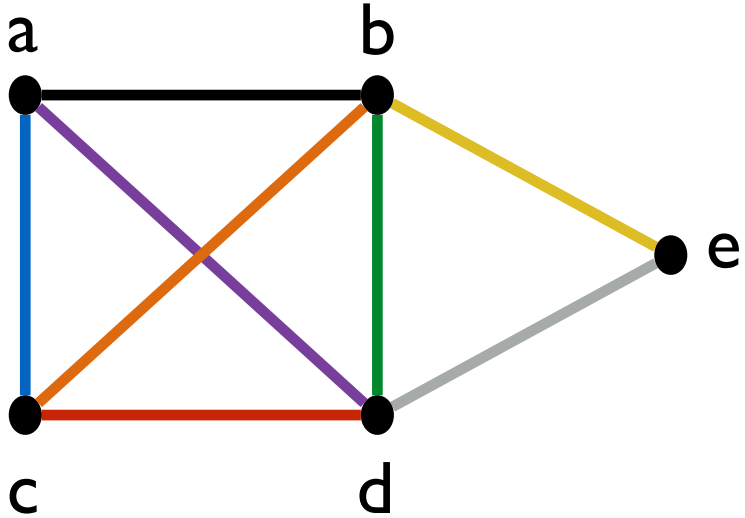
(how many possible "cuts" are there?)

Randomized Algorithm for Min Cut
(contraction algorithm)

Contraction algorithm for min cut

Example run 1





Select an edge randomly:

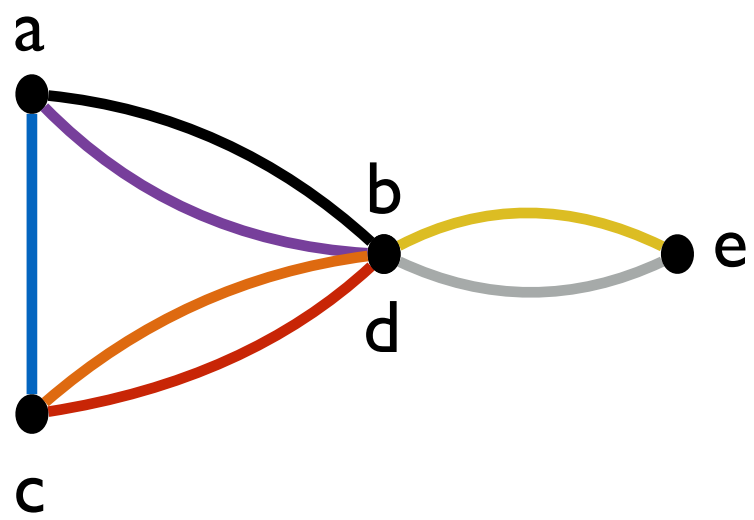
Green edge selected.

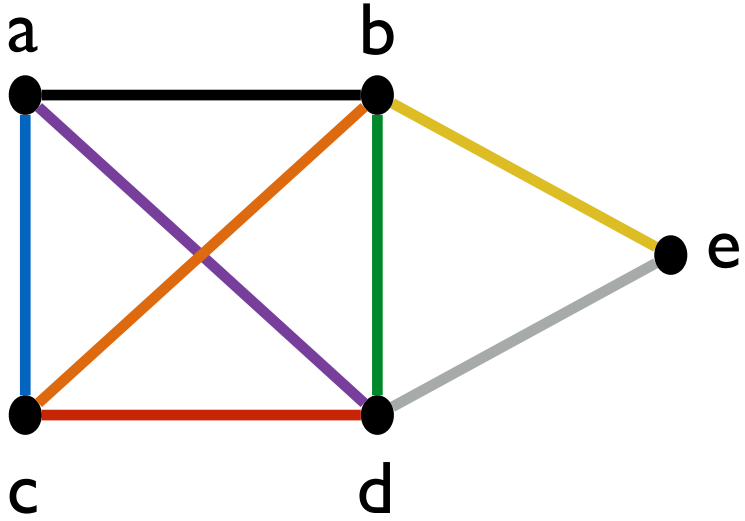
Contract that edge.

Size of min-cut: 2

Contraction algorithm for min cut

Example run 1





Select an edge randomly:

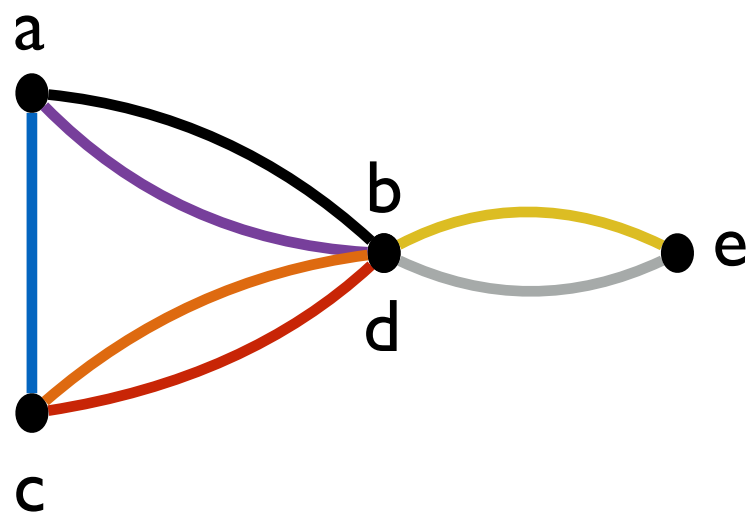
Size of min-cut: 2

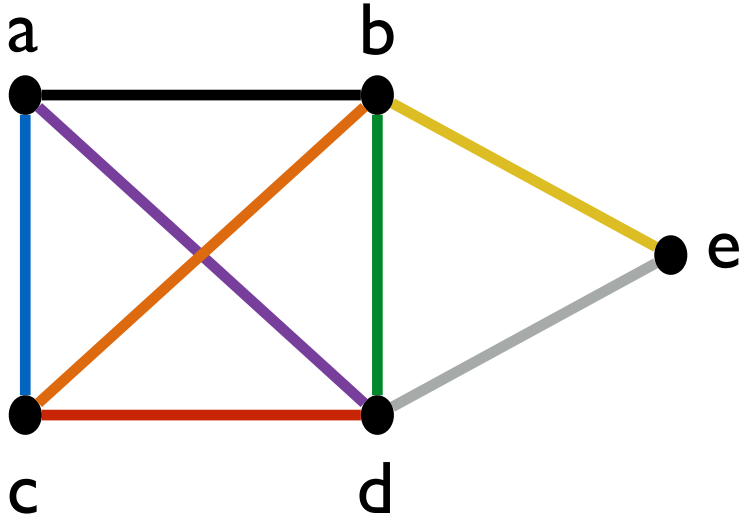
Green edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 1





Select an edge randomly:

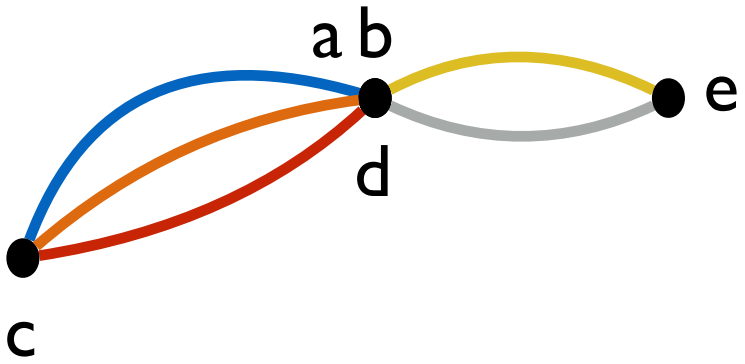
Size of min-cut: 2

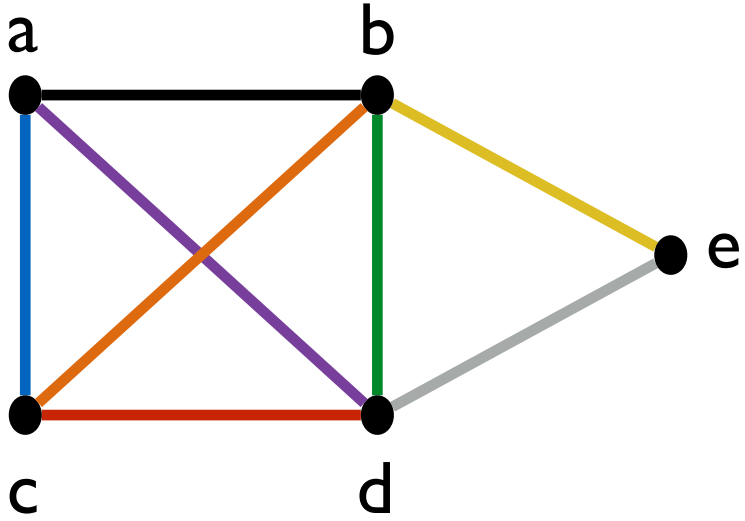
Purple edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 1





Select an edge randomly:

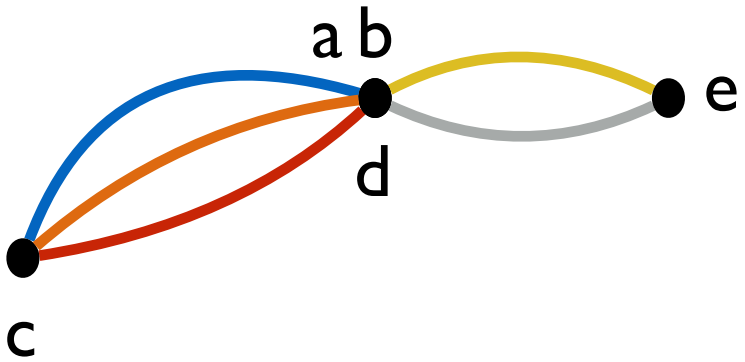
Size of min-cut: 2

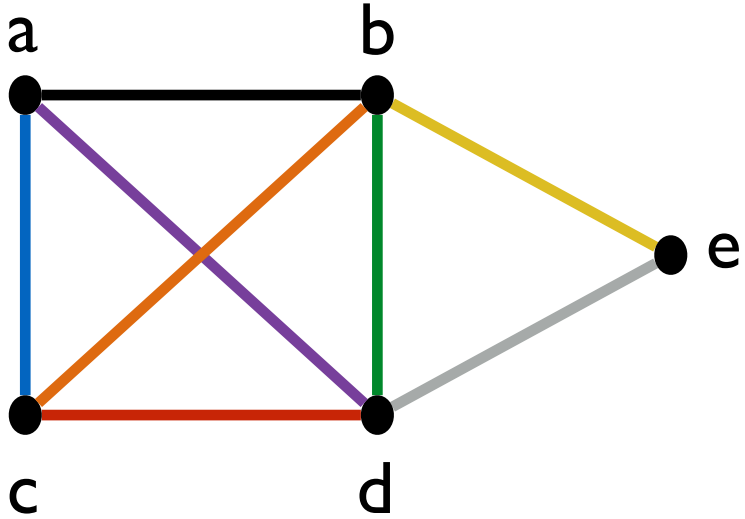
Purple edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 1





Select an edge randomly:

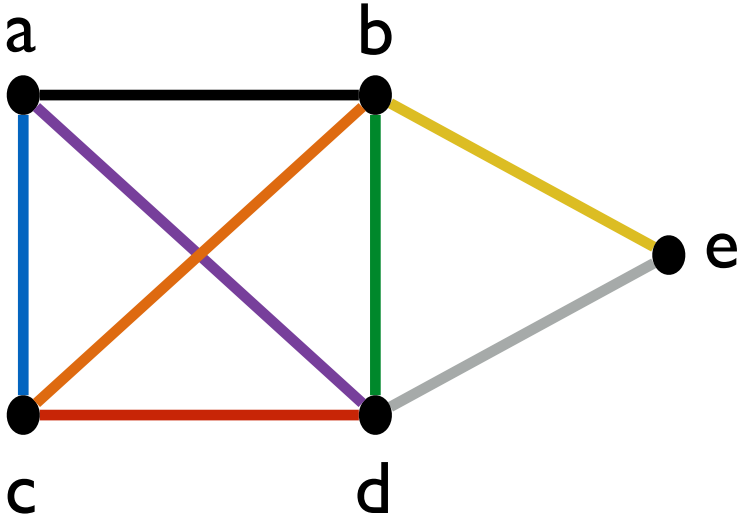
Size of min-cut: 2

Blue edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 1



Select an edge randomly:

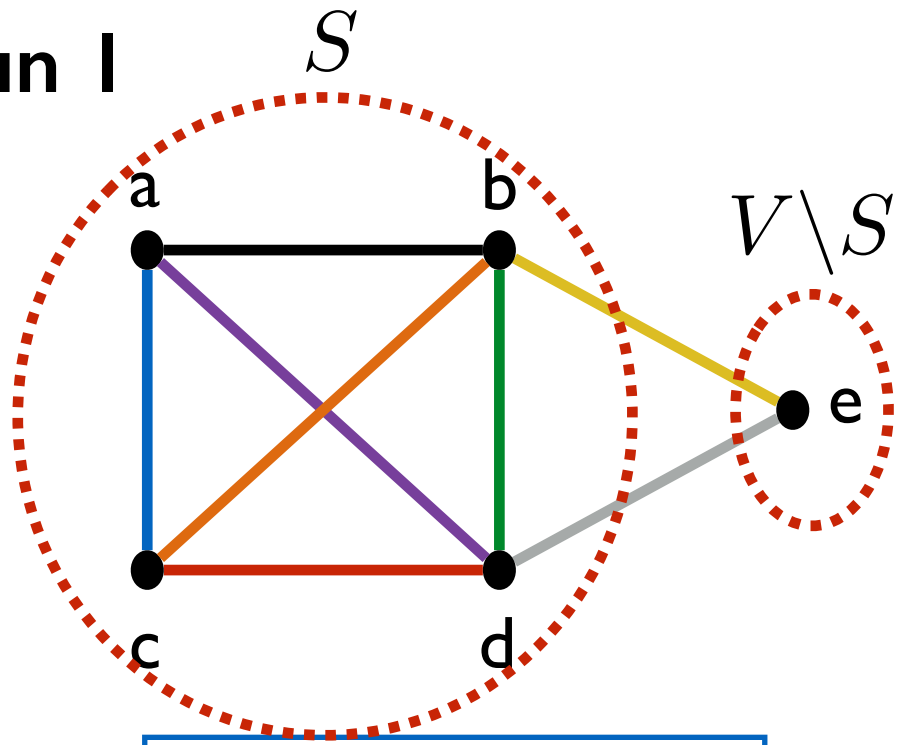
Size of min-cut: 2

Blue edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 1



Size of min-cut: 2

Select an edge randomly:

Blue edge selected.

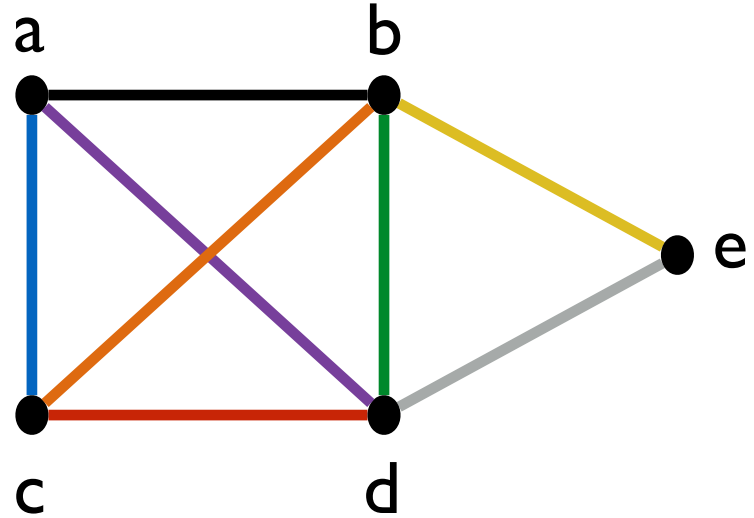
Contract that edge. (delete self loops)

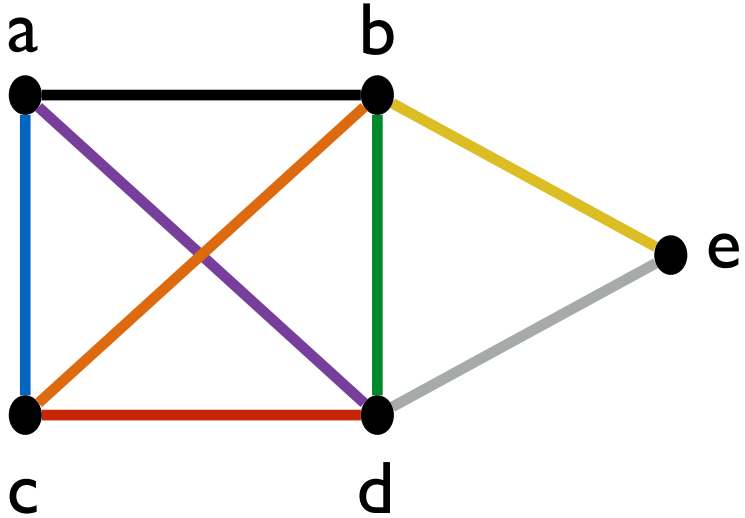
When two vertices remain, you have your cut:

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

Contraction algorithm for min cut

Example run 2





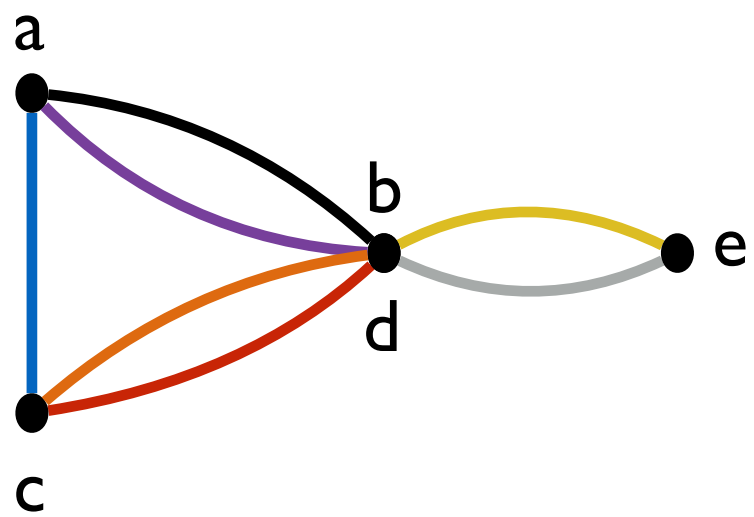
Select an edge randomly:

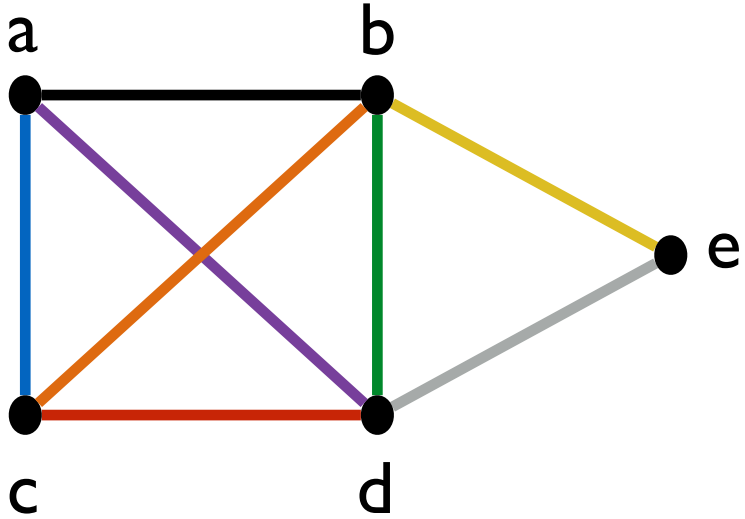
Green edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 2





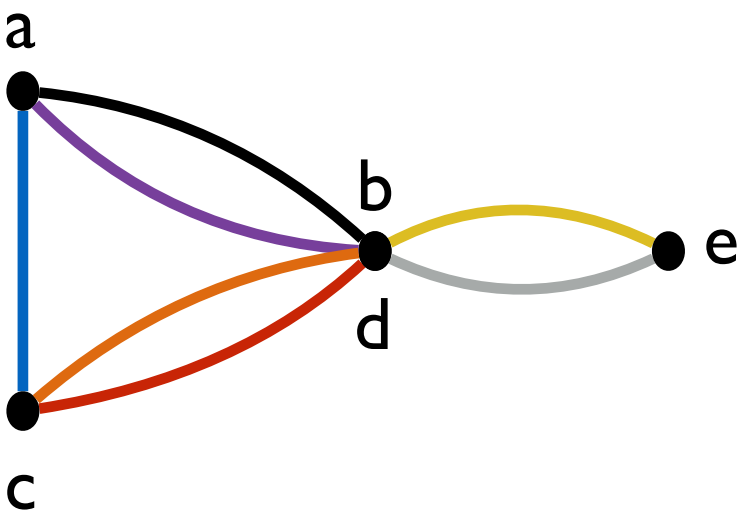
Select an edge randomly:

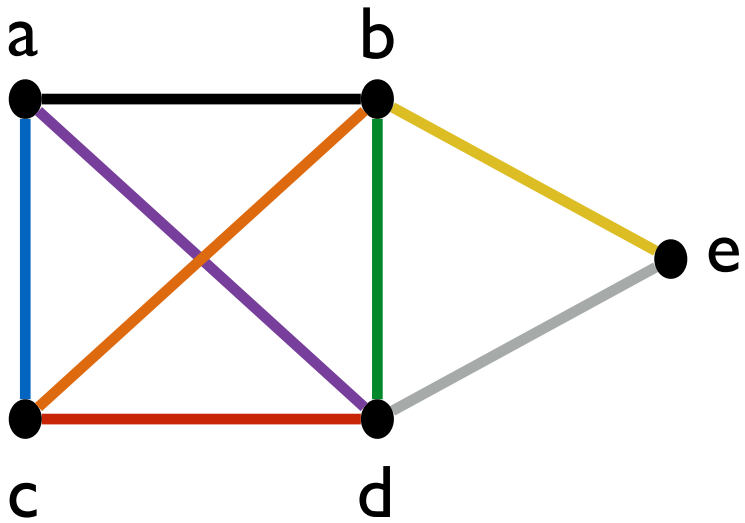
Green edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 2





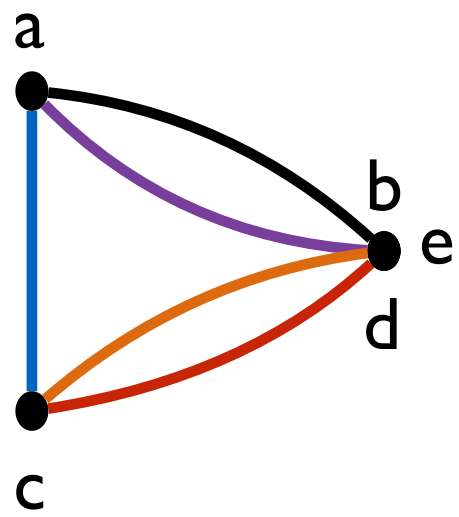
Select an edge randomly:

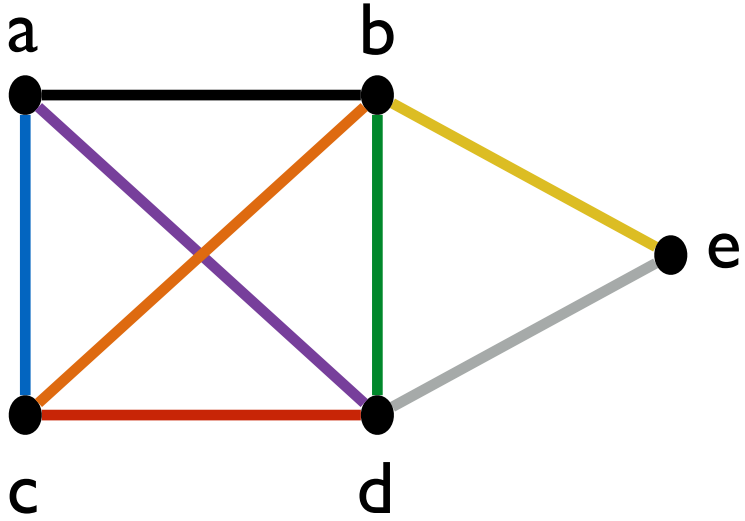
Yellow edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 2





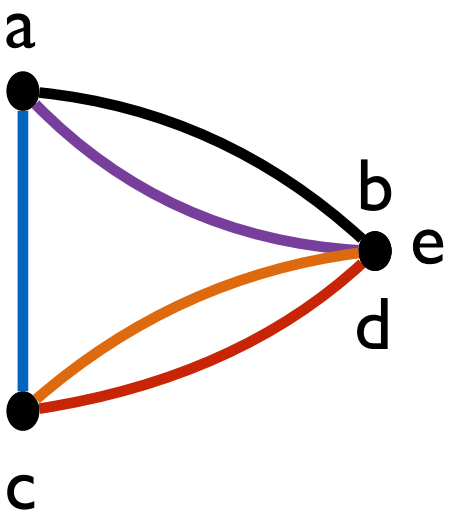
Select an edge randomly:

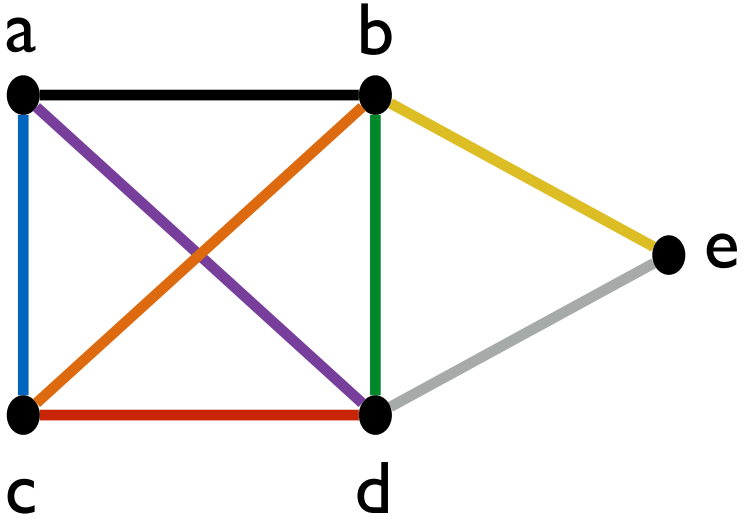
Yellow edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 2





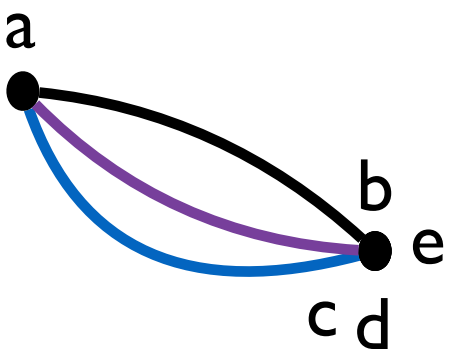
Select an edge randomly:

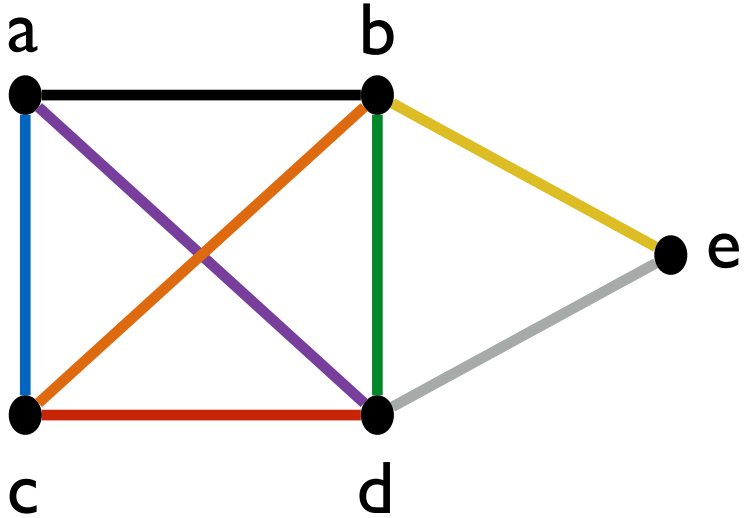
Red edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 2





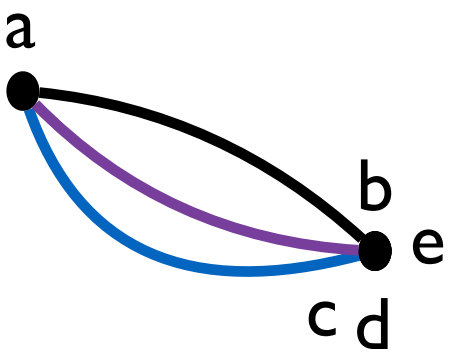
Select an edge randomly:

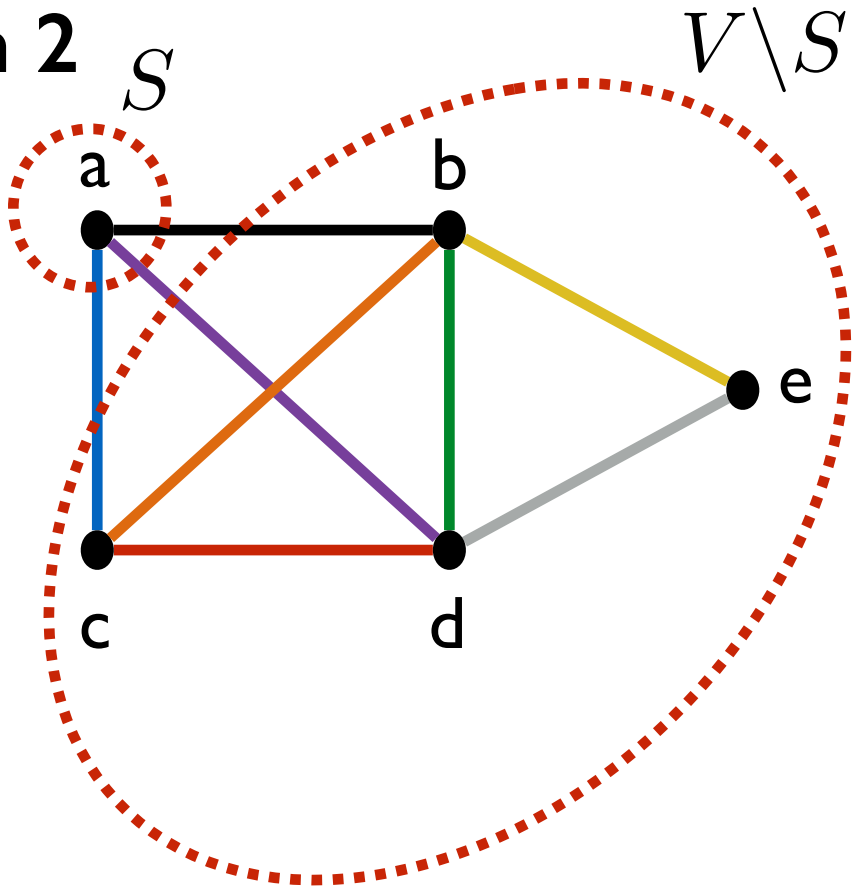
Red edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Example run 2





Select an edge randomly:

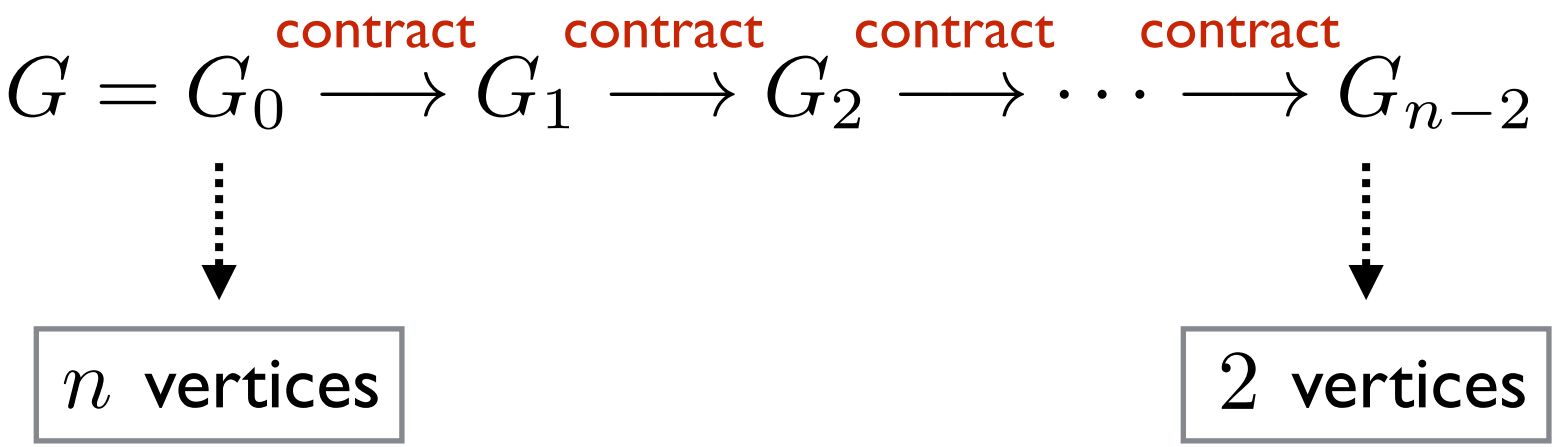
Red edge selected.

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

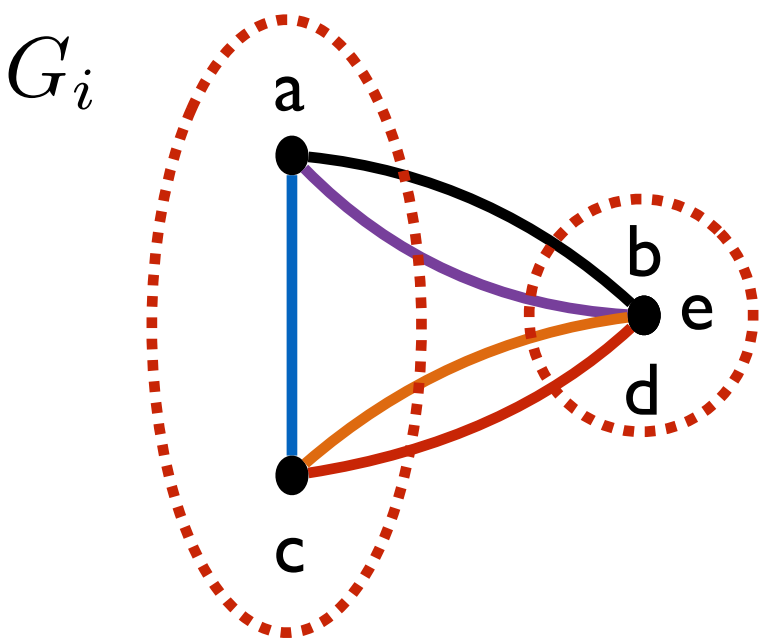


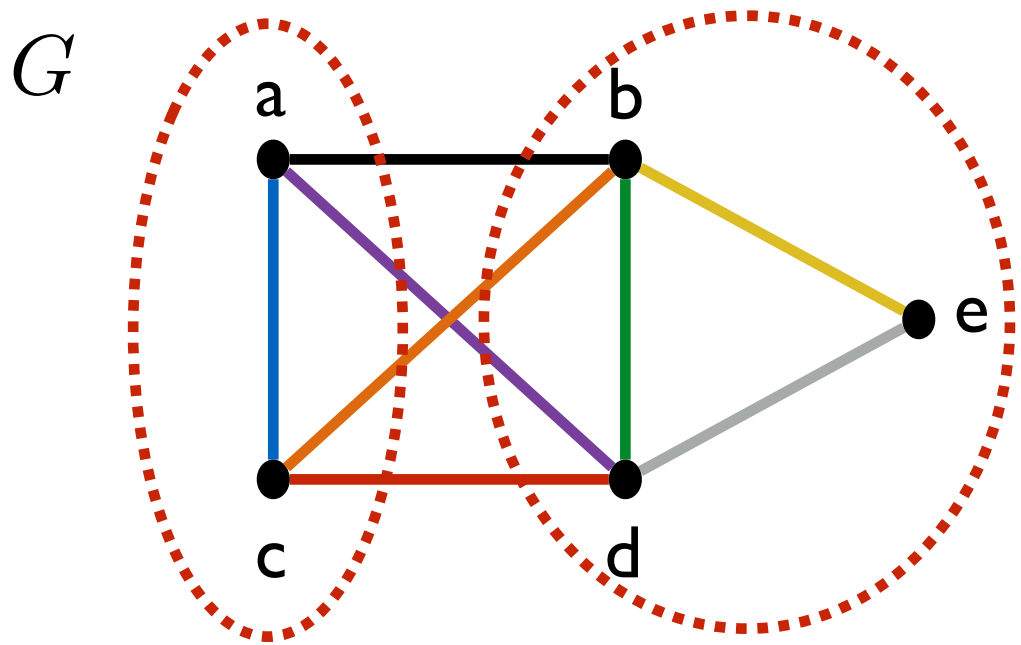
$n - 2$ iterations

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

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)

Proof of Theorem

Pre-proof Poll

Let k be the size of a minimum cut.

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

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

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

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

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

Answer

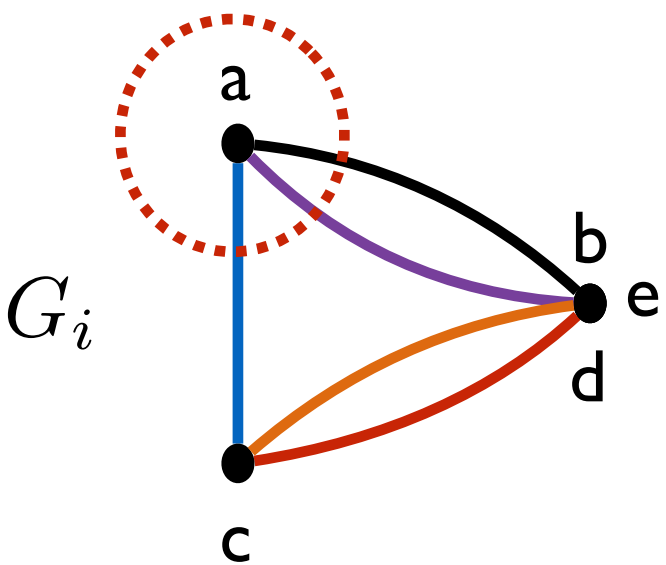
For every G_i , $\quad k \leq \min_v \deg_{G_i}(v)$

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

Why?

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

Example:



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

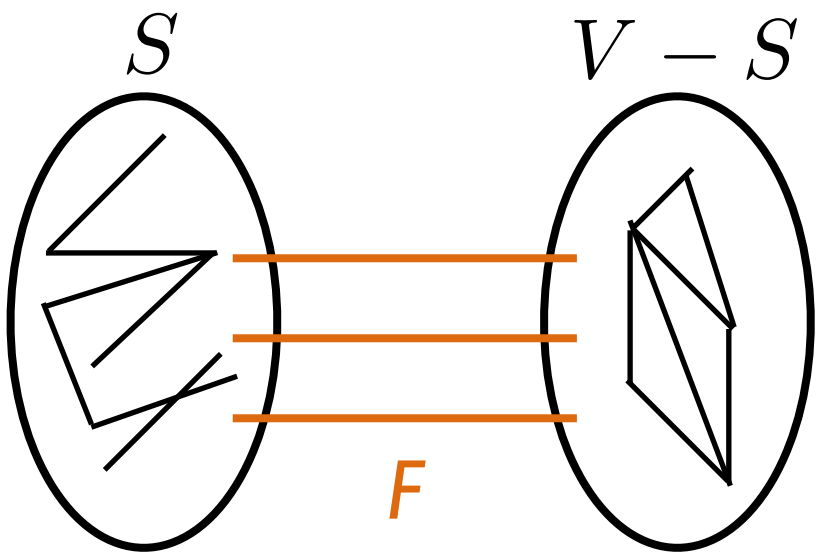
Same cut exists in original graph.

So $k \leq 3$.

Proof of theorem

Fix some minimum cut.

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



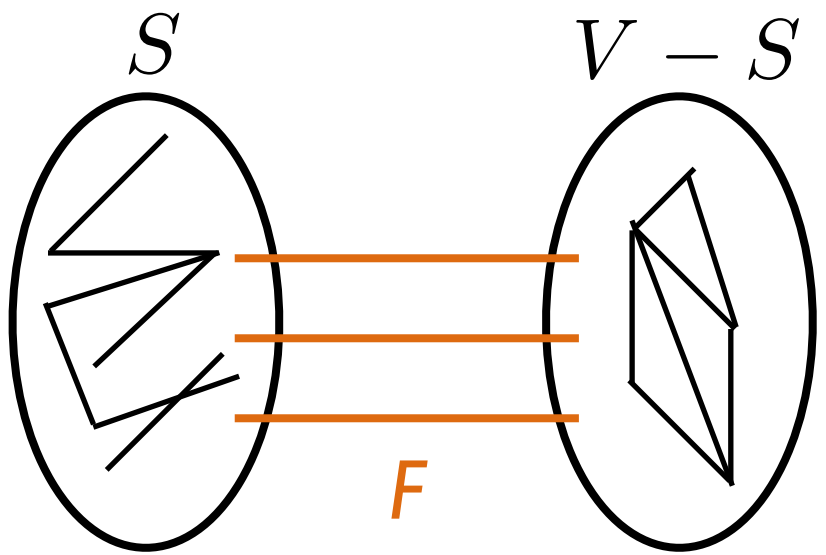
Will show $\Pr[\text{algorithm outputs } F] \geq 1/n^2$

(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.

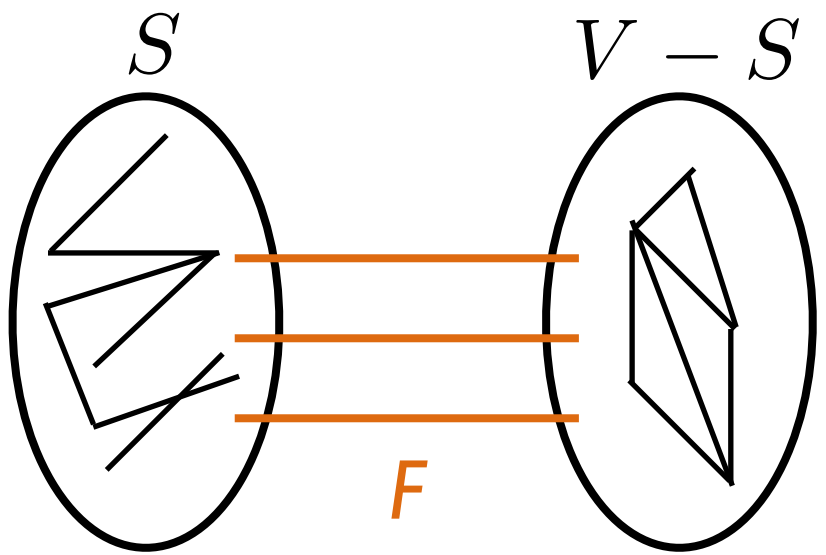
What if the algorithm picks an edge in F to contract?

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

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

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

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

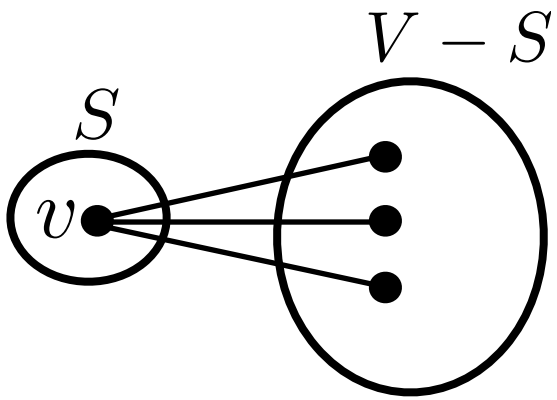
want to write in terms of *k* and *n*

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

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

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

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

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

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

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

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

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

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

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

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

□

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?

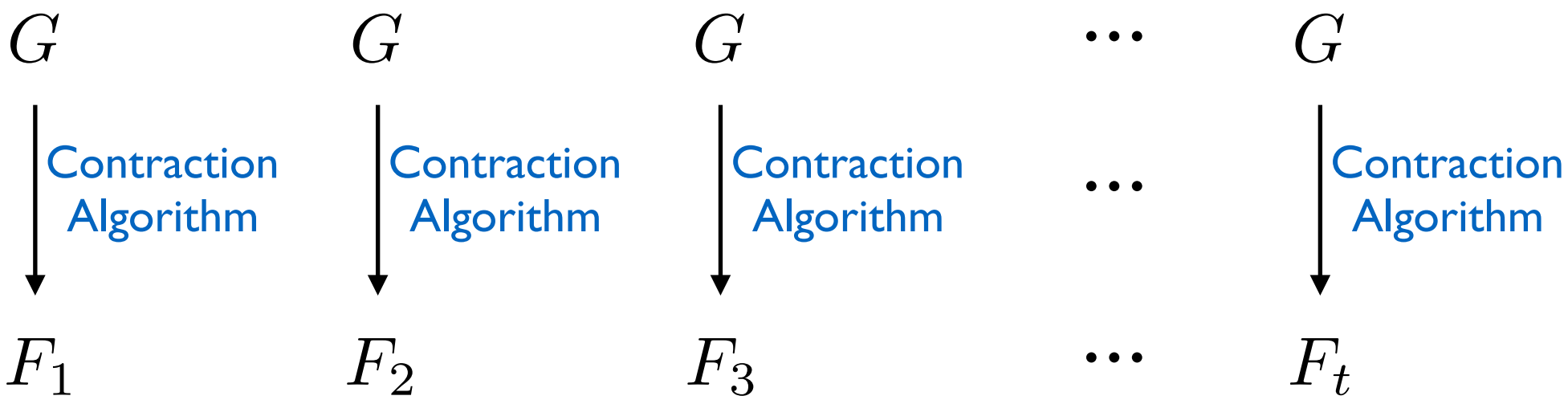
- The algorithm runs in polynomial time.
- There are exponentially many cuts. ($\approx 2^n$)
- There is a way to boost the probability of success to $1 - \frac{1}{e^n}$ (and still remain in polynomial time)

Boosting Phase

(and the world's greatest approximation!)

Boosting phase

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



Output the minimum among F_i's.

larger t $\Longrightarrow$ better success probability

What is the relation between t and success probability?

Boosting phase

What is the relation between t and success probability?

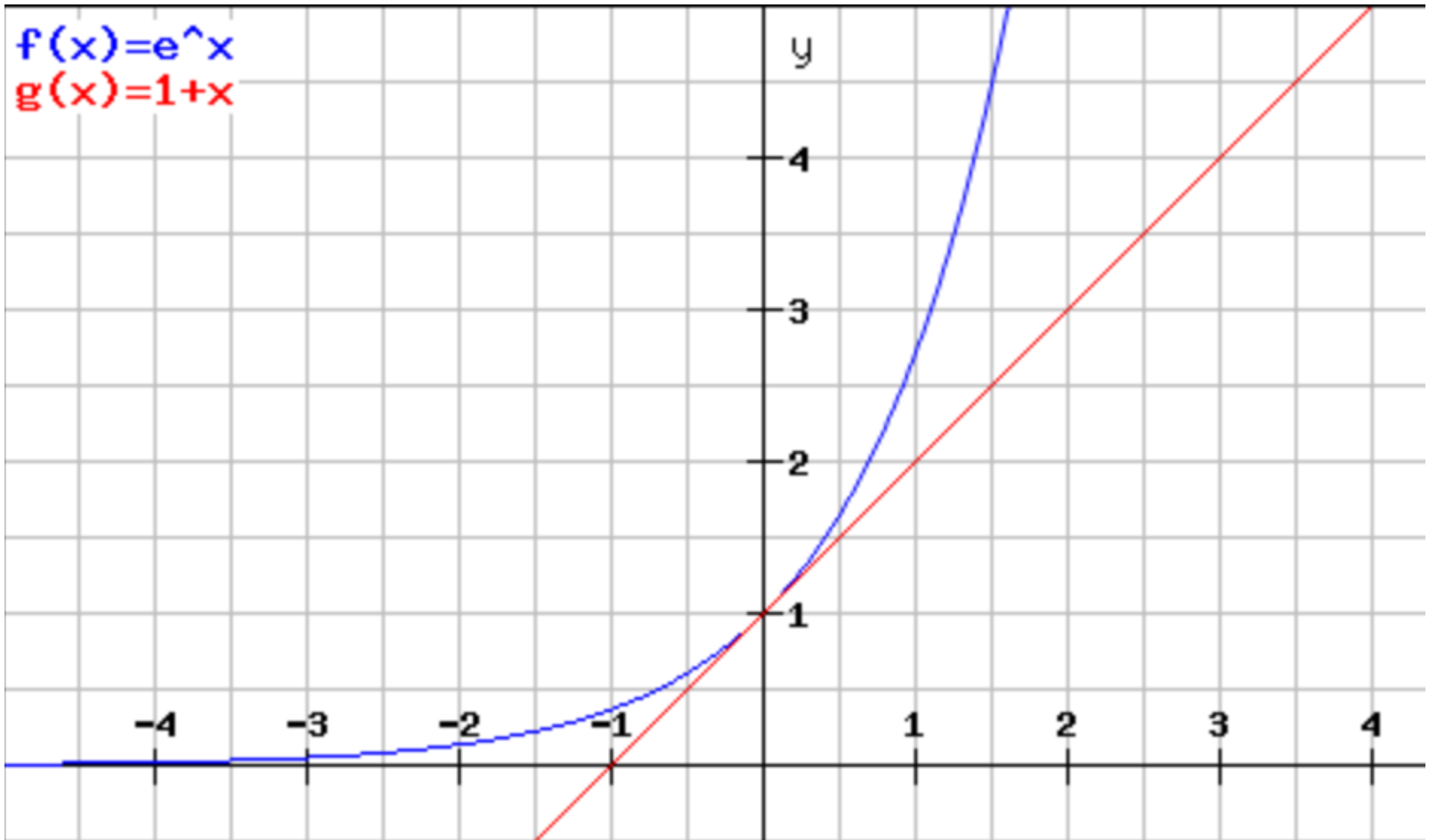
Let A_i = "in the i'th repetition, we **don't** find a min cut."

$$\begin{aligned}
\Pr[\text{error}] &= \Pr[\text{don't find a min cut}] \\
&= \Pr[A_1 \cap A_2 \cap \cdots \cap A_t] \\
&\overset{\text{ind. events}}{=} \Pr[A_1]\Pr[A_2]\cdots\Pr[A_t] \\
&= \Pr[A_1]^t \leq \left(1 - \frac{1}{n^2}\right)^t
\end{aligned}$$

Boosting phase

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

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



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 problems for which:

- there is a poly-time randomized algorithm,
- we can't find a poly-time deterministic algorithm.