Chapter 7
Randomization

Algorithm Theory
WS 2018/19

Fabian Kuhn

Randomization

Randomized Algorithm:

- An algorithm that uses (or can use) random coin flips in order to make decisions

We will see: randomization can be a powerful tool to

- Make algorithms faster
- Make algorithms simpler
- Make the analysis simpler
  - Sometimes it's also the opposite…
- Allow to solve problems (efficiently) that cannot be solved (efficiently) without randomization
  - True in some computational models (e.g., for distributed algorithms)
  - Not clear in the standard sequential model

Contention Resolution

A simple starter example (from distributed computing)

- Allows to introduce important concepts
- … and to repeat some basic probability theory

Setting: (nodes)

- n processes, 1 resource
  (e.g., communication channel, shared database, …)
- There are time slots 1,2,3, …
- In each time slot, only one client can access the resource
- All clients need to regularly access the resource
- If client i tries to access the resource in slot t:
  - Successful iff no other client tries to access the resource in slot t

Algorithm

Algorithm Ideas:

- Accessing the resource deterministically seems hard
  - need to make sure that processes access the resource at different times
  - or at least: often only a single process tries to access the resource
- **Randomized solution:**
  In each time slot, each process tries with probability p.

Analysis:

- How large should p be?
- How long does it take until some process i succeeds?
- How long does it take until all processes succeed?
- What are the probabilistic guarantees?

Analysis

Events:

- $\boldsymbol{\mathcal{A}_{x,t}}$: process x tries to access the resource in time slot t
  - Complementary event: $\overline{\mathcal{A}_{x,t}}$

$$\mathbb{P}(\mathcal{A}_{x,t}) = p, \qquad \mathbb{P}(\overline{\mathcal{A}_{x,t}}) = 1 - p$$

- $\boldsymbol{\mathcal{S}_{x,t}}$: process x is successful in time slot t

$$\mathcal{S}_{x,t} = \mathcal{A}_{x,t} \cap \left(\bigcap_{y \neq x} \overline{\mathcal{A}_{y,t}} \right)$$

$\mathcal{A}_{x,t}$, $\overline{\mathcal{A}_{y,t}}$ are independent

- **Success probability** (for process x):

$$\mathbb{P}(\mathcal{S}_{x,t}) = \mathbb{P}(\mathcal{A}_{x,t}) \cdot \prod_{y \neq x} \mathbb{P}(\overline{\mathcal{A}_{y,t}}) = p \cdot (1-p)^{n-1}$$

choose p s.t. $\mathbb{P}(\mathcal{S}_{x,t})$ is maximized

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

Fixing p

- $\mathbb{P}(\mathcal{S}_{x,t}) = p(1-p)^{n-1}$ is maximized for

$$\boldsymbol{p = \frac{1}{n}} \quad \Longrightarrow \quad \mathbb{P}(\mathcal{S}_{x,t}) = \frac{1}{n}\underbrace{\left(1-\frac{1}{n}\right)^{n-1}}_{\to 1/e}.$$

- **Asymptotics:**

$$\text{For } n \geq 2\text{:} \quad \frac{1}{4} \leq \left(1-\frac{1}{n}\right)^{n} < \frac{1}{e} < \left(1-\frac{1}{n}\right)^{n-1} \leq \frac{1}{2}$$

- **Success probability:**

$$\boldsymbol{\frac{1}{en} < \mathbb{P}(\mathcal{S}_{x,t}) \leq \frac{1}{2n}}$$

Time Until First Success $q := \mathbb{P}(S_{x,t}) = \frac{1}{n}(1-\frac{1}{n})^{n-1}$

Random Variable T_x:

- $T_x = t$ if proc. x is successful in slot t for the first time

- **Distribution:**

$\mathbb{P}(T_x=1) = q$, $\mathbb{P}(T_x=2) = (1-q)\cdot q$, $\mathbb{P}(T_x=t) = (1-q)^{t-1}\cdot q$

- T_x is geometrically distributed with parameter

$$q = \mathbb{P}(\mathcal{S}_{x,t}) = \frac{1}{n}\left(1-\frac{1}{n}\right)^{n-1} > \frac{1}{en}.$$

- **Expected time** until first success:

$$\mathbb{E}[T_x] = \frac{1}{q} < en$$

Time Until First Success

Failure Event $\mathcal{F}_{x,t}$: Process x does not succeed in time slots $1, \dots, t$

$$\mathcal{F}_{x,t} = \bigcap_{r=1}^{t} \overline{S_{x,r}}$$

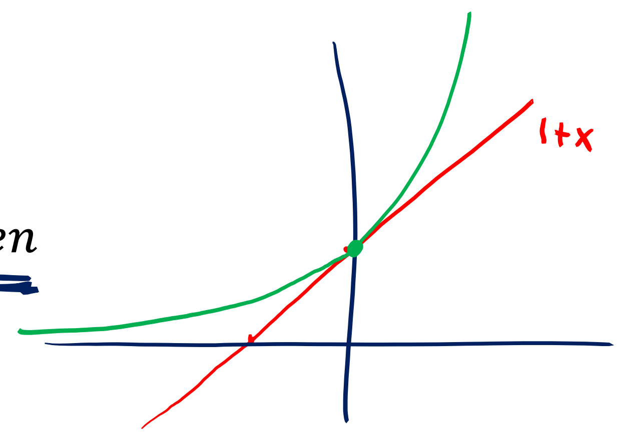
- The events $\mathcal{S}_{x,t}$ are independent for different t:

$$\mathbb{P}(\mathcal{F}_{x,t}) = \mathbb{P}\left(\bigcap_{r=1}^{t} \overline{\mathcal{S}_{x,r}}\right) = \prod_{r=1}^{t} \mathbb{P}\left(\overline{\mathcal{S}_{x,r}}\right) = \left(1 - \mathbb{P}(\mathcal{S}_{x,r})\right)^{t}$$

$$\forall x \in \mathbb{R}: \ 1 + x \leq e^{x}$$

- We know that $\mathbb{P}(\mathcal{S}_{x,r}) > {}^{1}/_{en}$:

$$\mathbb{P}(\mathcal{F}_{x,t}) < \underbrace{\left(1 - \frac{1}{en}\right)}_{< e^{-1/en}}{}^{t} < e^{-t/en}$$



Time Until First Success

No success by time t: $\mathbb{P}(\mathcal{F}_{x,t}) < e^{-t/en}$

$e^{c \ln n} = (e^{\ln n})^c = n^c$

$t = \lceil en \rceil$: $\mathbb{P}(\mathcal{F}_{x,t}) < 1/e$

- Generally if $t = \Theta(n)$: constant success probability

$t \geq en \cdot c \cdot \ln n$: $\mathbb{P}(\mathcal{F}_{x,t}) < 1/e^{c \cdot \ln n} = 1/n^c$

- For success probability $1 - 1/n^c$, we need $t = \Theta(n \log n)$.
- We say that x succeeds **with high probability** in $O(n \log n)$ time.

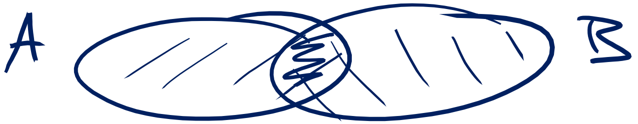
with prob. $\geq 1 - \frac{1}{n^c}$

for any const. $c > 0$

c can only affect the hidden const.

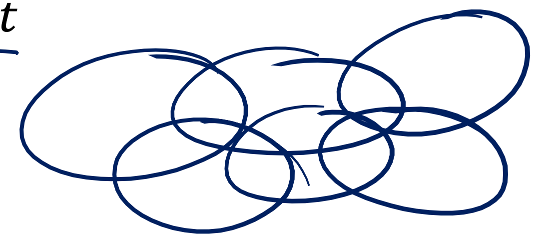
Time Until All Processes Succeed

Event $\mathcal{F}_t$: some process has not succeeded by time t



$$\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B)$$
$$\leq \mathbb{P}(A) + \mathbb{P}(B)$$

$$\mathcal{F}_t = \bigcup_{x=1}^{n} \mathcal{F}_{x,t}$$



Union Bound: For events $\mathcal{E}_1, \dots, \mathcal{E}_k$,

$$\mathbb{P}\left(\bigcup_{x}^{k} \mathcal{E}_x\right) \leq \sum_{x}^{k} \mathbb{P}(\mathcal{E}_x)$$

Probability that not all processes have succeeded by time t:

$$\mathbb{P}(\mathcal{F}_t) = \mathbb{P}\left(\bigcup_{x=1}^{n} \mathcal{F}_{x,t}\right) \leq \sum_{x=1}^{n} \underbrace{\mathbb{P}(\mathcal{F}_{x,t})}_{< e^{-t/en}} < \underbrace{n \cdot e^{-t/en}}_{\text{has to be small}}.$$

Time Until All Processes Succeed

Claim: With high probability, all processes succeed in the first $O(n \log n)$ time slots.

Proof:

- $\mathbb{P}(\mathcal{F}_t) < n \cdot e^{-t/en}$
- Set $t = \lceil en \cdot (c+1) \ln n \rceil$

$$\mathbb{P}(\mathcal{F}_t) < n \cdot e^{-(c+1)\ln n} = n \cdot \frac{1}{n^{c+1}} = \frac{1}{n^c}$$

$$\mathbb{P}(\overline{\mathcal{F}_t}) > 1 - \frac{1}{n^c}$$

Remark: $\Theta(n \log n)$ time slots are necessary for all processes to succeed with reasonable probability

Primality Testing

Problem: Given a natural number $n \geq 2$, is n a prime number?

Simple primality test:

$n = a \cdot b$

1. **if** n is even **then**
2. **return** $(n = 2)$
3. **for** $i := 1$ **to** $\lfloor \sqrt{n}/2 \rfloor$ **do**
4. **if** $2i + 1$ divides n **then**
5. **return false**
6. **return true**

Size of input: $O(\log n)$

- **Running time:** $O(\sqrt{n})$

A Better Algorithm?

- How can we test primality efficiently?
- We need a little bit of basic number theory…

integers $1,\dots,p-1$, multipl. mod p

Square Roots of Unity: In $\mathbb{Z}_p^*$, where p is a prime, the only solutions of the equation $x^2 \equiv 1 \pmod p$ are $x \equiv \pm 1 \pmod p$

$\mathbb{Z}_p^* = \{1,\dots,p-1\}$

$x^2 \equiv 1 \pmod p$

$x^2 - 1 \equiv 0 \pmod p$

$(x+1)(x-1) \equiv 0 \pmod p \iff (x+1)(x-1) = c \cdot p$

p has to be a prime factor of $x+1$ or $x-1$

$x+1 \equiv 0 \pmod p$

$x-1 \equiv 0 \pmod p$

not true if p is not prime

$p = 15$

$x = 4 \qquad x^2 \equiv 1 \pmod{15}$

- If we find an $x \not\equiv \pm 1 \pmod n$ such that $x^2 \equiv 1 \pmod n$, we can conclude that n is not a prime.

Algorithm Idea

$\mathbb{Z}_p^* = \{1, \dots, p-1\}$

Claim: Let $p > 2$ be a prime number such that $p - 1 = 2^s d$ for an integer $s \geq 1$ and some odd integer $d \geq 3$. Then for all $a \in \mathbb{Z}_p^*$,

$$a^d \equiv 1 \pmod{p} \;\; \textbf{or} \;\; a^{2^r d} \equiv -1 \pmod{p} \;\text{ for some } 0 \leq r < s.$$

Proof: recall $x^2 \equiv 1 \pmod{p} \Rightarrow x \equiv \pm 1 \pmod{p}$

- **Fermat's Little Theorem:** Given a prime number p,

$$\forall a \in \mathbb{Z}_p^*: \quad a^{p-1} \equiv 1 \pmod{p}$$

$\underbrace{a^{\frac{p-1}{2}}}_{x}$ $= \begin{cases} +1 \pmod{p} & \text{we're done if } s=1 \\ -1 \pmod{p} \checkmark \end{cases}$

$\frac{p-1}{2} = 2^{s-1} d$

$+1 \pmod{p} \to \frac{p-1}{2} = d \; (s=1) \Rightarrow a^d \equiv 1 \pmod{p} \checkmark$

$\to \frac{p-1}{2}$ even $\Rightarrow a^{\frac{p-1}{4}} = \begin{cases} +1 \begin{cases} \to s=2 \checkmark \\ s>2 \to a^{\frac{p-1}{8}} \end{cases} \\ -1 \checkmark \end{cases}$

Primality Test

We have: If n is an odd prime and $n - 1 = 2^s d$ for an integer $s \geq 1$ and an odd integer $d \geq 3$. Then for all $a \in \{1, \dots, n-1\}$,

$$a^d \equiv 1 \pmod{n} \;\; \textbf{or} \;\; a^{2^r d} \equiv -1 \pmod{n} \;\; \text{for some } 0 \leq r < s.$$

(*)

Idea: If we find an $a \in \{1, \dots, n-1\}$ such that

$$a^d \not\equiv 1 \pmod{n} \;\; \textbf{and} \;\; a^{2^r d} \not\equiv -1 \pmod{n} \;\; \text{for all } 0 \leq r < s,$$

($\neg$*)

we can conclude that n is not a prime.

- For every odd composite $n > 2$, at least $3/4$ of all possible a satisfy the above condition
- How can we find such a *witness* a efficiently?

idea: pick a at random

Miller-Rabin Primality Test

- Given a natural number $n \geq 2$, is n a prime number?

Miller-Rabin Test:

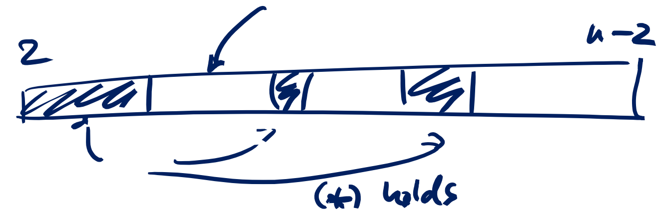
1. **if** n is even **then return** $(n = 2)$
2. compute s, d such that $n - 1 = 2^s d$;
3. choose $a \in \{2, \dots, n-2\}$ uniformly at random;
4. $x \coloneqq a^d \bmod n$;
5. **if** $x = 1$ **or** $x = n - 1$ **then return** $\underbrace{\textbf{probably prime;}}_{\text{true}}$
6. **for** $r \coloneqq 1$ **to** $s - 1$ **do**
7. $\quad x \coloneqq x^2 \bmod n$;
8. $\quad$ **if** $x = n - 1$ **then return** $\underbrace{\textbf{probably prime;}}_{\text{true}}$
9. $\underbrace{\textbf{return composite;}}_{\text{false}}$

(*) holds

Analysis

Theorem:

- If n is prime, the Miller-Rabin test always returns **true**.
- If n is composite, the Miller-Rabin test returns **false** with probability at least $^3/_4$.



Proof:

- If n is prime, the test works for all values of a
- If n is composite, we need to pick a good witness a

Corollary: If the Miller-Rabin test is repeated k times, it fails to detect a composite number n with probability at most 4^{-k}.

Running Time

Cost of Modular Arithmetic:

- Representation of a number $x \in \mathbb{Z}_n$: $O(\log n)$ bits
- Cost of adding two numbers $x + y \bmod n$: $O(\log n)$
- Cost of multiplying two numbers $x \cdot y \bmod n$: naively $O(\log^2 n)$
  - It's like multiplying degree $O(\log n)$ polynomials
    → use FFT to compute $z = x \cdot y$ $O(\log n \cdot \log\log n \cdot \log\log\log n)$

Running Time

$x, d \in \mathbb{Z}_p^*$

Cost of exponentiation $x^d \bmod n$:

- Can be done using $O(\log d)$ multiplications
- Base-2 representation of d: $d = \sum_{i=0}^{\lfloor \log d \rfloor} d_i 2^i$
- **Fast exponentiation:**
  1. $y := 1$;
  2. **for** $i := \lfloor \log d \rfloor$ **to** 0 **do**
  3. $\quad y := y^2 \bmod n$;
  4. $\quad$ **if** $d_i = 1$ **then** $y := y \cdot x \bmod n$;
  5. **return** y;

$(((x^2)^2 \cdot x)^2 \cdot x)^2$

10110

- **Example:** $d = 22 = 10110_2$

$x^{22} = (x^{11})^2 = ((x^5)^2 \cdot x)^2 = (((x^2)^2 \cdot x)^2 \cdot x)^2$

$x^{23} = (x^{11})^2 \cdot x$

Running Time

Theorem: One iteration of the Miller-Rabin test can be implemented with running time $O(\log^2 n \cdot \log\log n \cdot \log\log\log n)$. $\in \tilde{O}(\log^2 n)$

$1 - \frac{1}{n}$

$4^{-k} = \frac{1}{n}$

1. **if** n is even **then return** $(n = 2)$
2. compute s, d such that $n - 1 = 2^s d$;
3. choose $a \in \{2, \dots, n-2\}$ uniformly at random;
4. $x := a^d \bmod n$; $\quad O(\log d) = O(\log n)$ mult.
5. **if** $x = 1$ **or** $x = n - 1$ **then return probably prime;**
6. **for** $r := 1$ **to** $s - 1$ **do** $\quad s = O(\log n)$
7. $\quad x := x^2 \bmod n$;
8. $\quad$ **if** $x = n - 1$ **then return probably prime;**
9. **return composite;**

Deterministic Primality Test

$\tilde{O}(f(n)) = f(n) \cdot (\log(f(n)))^c$

- If a conjecture called the generalized Riemann hypothesis (GRH) is true, the Miller-Rabin test can be turned into a polynomial-time, deterministic algorithm

  → It is then sufficient to try all $a \in \{1, \dots, O(\log^2 n)\}$

- It has long not been proven whether a deterministic, polynomial-time algorithm exists

  hides factors polynomial in $\log\log n$

- In 2002, Agrawal, Kayal, and Saxena gave an $\tilde{O}(\log^{12} n)$-time deterministic algorithm
  - Has been improved to $\tilde{O}(\log^6 n)$

- In practice, the randomized Miller-Rabin test is still the fastest algorithm