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:

- n processes, 1 resource
  (e.g., communication channel, shared database, …)
- There are time slots $1,2,3, \ldots$
- 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: $\overline{\mathcal{A}_{x,t}}$

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

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

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

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

Fixing p

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

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

- **Asymptotics:**

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

- **Success probability:**

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

Time Until First Success

Random Variable $\boldsymbol{T_x}$:

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

- **Distribution:**

- 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:

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

Time Until First Success

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

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

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

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

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

Time Until First Success

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

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

- 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.

Time Until All Processes Succeed

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

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

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:

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

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 $t = \lceil en \cdot (c+1) \ln n \rceil$

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:

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**

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

A Better Algorithm?

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

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 $x \equiv \pm 1 \pmod{p}$

- 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

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:

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

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

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

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

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?

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 probably prime;**
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 probably prime;**
9. **return composite;**

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 $x + y \bmod n$:
- Cost of multiplying two numbers $x \cdot y \bmod n$:
  - It's like multiplying degree $O(\log n)$ polynomials
    → use FFT to compute $z = x \cdot y$

Running Time

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. $y \coloneqq 1;$
  2. **for** $i \coloneqq \lfloor \log d \rfloor$ **to** 0 **do**
  3. $\quad y \coloneqq y^2 \bmod n;$
  4. $\quad$ **if** $d_i = 1$ **then** $y \coloneqq y \cdot x \bmod n;$
  5. **return** $y;$
- **Example:** $d = 22 = 10110_2$

Running Time

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

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 probably prime;**
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 probably prime;**
9. **return composite;**

Deterministic Primality Test

- 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

- 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