INF 561: Using randomness in algorithms **Winter 2013**

Lecture 1 — 9th January 2013

Lecturer: Frédéric Magniez *Scribe: Lauriane Aufrant and Matthieu Vegreville*

The goal of this course is to present a formal definition of randomized algorithms and some easy applications.

1.1 Formal basis

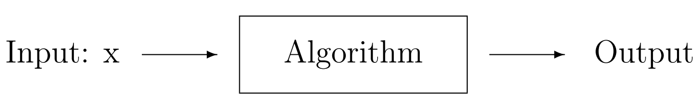
1.1.1 Typology of problems

Given an input x, the purpose of a problem is to search for an appropriate output:

- *decision problem:* $ACCEPT$ or $REJECT$
- *functional problem:* $F(x)$
- *relational problem:* y such that $x\mathcal{R}y$

1.1.2 Deterministic and randomized algorithms

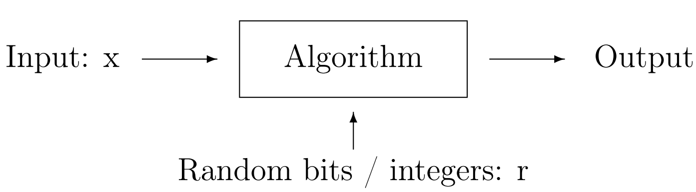
Deterministic algorithm



Goal:

- correctly solve the problem on all inputs
- efficiently: linear or polynomial time on input size

Randomized algorithm



A randomized algorithm, compared to a deterministic algorithm, has an additional input: the random variable r. We suppose that we have access to a source of uniform random bits or integers (which is basically equivalent).
Remarks:

- Behaviour depends on both x and r.
- Once r is fixed, the algorithm is deterministic.

Goal: find a randomized algo such that on all inputs:

- *Monte Carlo algorithms:* output is correct for most of random choices r, complexity is small for all random choices r
- *Las Vegas algorithms:* output is correct for all random choices r, complexity is small in average over random choices r

NB: We do not know yet how to generate random numbers with computers, we have only access to pseudo-random generators.

1.1.3 Typology of randomized algorithms

Definition 1.1. *In this course we will study 3 types of randomized algorithms, presented here for a functional problem.*

1. *Algorithm A computes f without error and with average complexity T if for all inputs x:*
   - *for all random choices r, $A(x,r) = f(x)$*
   - $\mathbb{E}_r[\text{complexity}(A(x,r))] \leq T$

   NB: *T is generally a function of size of x*

   Example: *Quicksort with random pivot.*

2. *Algorithm A computes f without error, with probability $\delta < 1$ to abort and with complexity T if for all inputs x:*
   - *for all random choices r such that $A(x,r)$ does not abort, $A(x,r) = f(x)$*
   - *for all random choices r, $\text{complexity}(A(x,r)) \leq T$*
   - $\mathbb{P}_r[A(x,r) \text{ aborts}] \leq \delta$

   NB: *δ is often $\frac{1}{2}$. The algorithm is generally run a few times with new random bits each time, until it terminates at least once: $\delta_k = \frac{1}{2^k}$*

   Example: *Quicksort with random pivot and finite time of execution.*

3. *Algorithm A computes f with bounded error $\epsilon < \frac{1}{2}$ and complexity T if for all inputs x:*

- $\mathbb{P}_r[A(x,r) \neq f(x)] \leq \epsilon$
- *for all random choices* r, *complexity*$(A(x,r)) \leq T$

Theorem 1.2. *Types* 1 *and* 2 *are equivalent.*

Proof: The basic idea for converting an algorithm to type 2 is to stop the algorithm when T becomes too large.

- Let A be an algorithm of type 1. Let $c > 1$ be some constant.

  Define $B(x,r)$: Run $A(x,r)$ and stop it after running time T. If A has terminated, output the output of A. Otherwise, abort.

  If $B(x,r)$ does not abort, then its output is correct. Running time of $B \leq cT$. Let $\tau(r)$ be the running time of $A(x,r)$. $\mathbb{P}_r(B(x,r) aborts) = \mathbb{P}_r(\tau(r) \geq cT) \leq \frac{1}{c}$ because of Markov property, since $\tau(r) \geq 0$ and $\mathbb{E}_r[\tau(r)] \leq T$. We constructed an equivalent algorithm of type 2.

- Let A be an algorithm of type 2.

  Define $B(x)$: Run $A(x,r)$ with fresh random bits r until A does not abort. Output the output of A.

  B is always correct. A run of A aborts with probability $\delta < 1$.

  $$\mathbb{P}(A(x,r)\ aborts\ k\ times) \leq \delta^k$$

  $$\mathbb{E}[running\ time\ of\ B] \leq \sum_{k=0}^{\infty}(\delta^k T) = \frac{T}{1-\delta} = 2T\ for\ \delta = \frac{1}{2}$$

  We constructed an equivalent algorithm of type 1.

□

Definition 1.3. *A randomized algorithm applied to a decision problem can have several types of error:*

1. *Algorithm* A *has a* ***one-sided error*** ϵ *if*

   - *if the appropriate output for* x *is* $ACCEPT$, *then* $\mathbb{P}_r[A(x,r)\ accepts] = 1$
   - *if the appropriate output for* x *is* $REJECT$, *then* $\mathbb{P}_r[A(x,r)\ accepts] \leq \epsilon$

   NB: *In this case the algorithm is run a few times and* x *is accepted if it has been accepted by every execution. For* $\epsilon = \frac{1}{2}$, $\epsilon_k = \frac{1}{2^k}$

2. *Algorithm* A *has a* ***two-sided error*** ϵ *if*

   - *if the appropriate output for* x *is* $ACCEPT$, *then* $\mathbb{P}_r[A(x,r)\ accepts] \geq 1 - \epsilon$
   - *if the appropriate output for* x *is* $REJECT$, *then* $\mathbb{P}_r[A(x,r)\ accepts] \leq \epsilon$

   NB: *In this case the algorithm is run a few times and* x *is accepted if it has been accepted by most executions. Generally,* $\epsilon = \frac{1}{3}$.

1.1.4 Complexity classes

Our interest lies in two complexity classes:

1. **ZPP complexity class**, with *zero-error* algorithms:
   algorithms of type 1 or 2 with T polynomial on input size and $\delta = \frac{1}{2}$ for type 2
2. **BPP complexity class**, with *bounded-error* algorithms:
   algorithms of type 3 with T polynomial on input size and $\epsilon = \frac{1}{3}$

1.2 Applications

1.2.1 Matrix multiplication

Decision problem:

- input: A, B and C, $n \times n$ matrices over an arbitrary ring
- output: decide if $A \times B = C$

Freivald's test:

- Choose $r \in \{0,1\}^n$
- Evaluate $u = Cr$, $v = Br$ and $w = Av$
- Return $ACCEPT$ if $u = w$, else $REJECT$

Theorem 1.4. *Freivald's algorithm has a one-sided error:*

- *If* $AB = C$, $\mathbb{P}\left[algorithm\ accepts\right] = 1$
- *If* $AB \neq C$, $\mathbb{P}\left[algorithm\ accepts\right] \leq \frac{1}{2}$

Since its running time is at most $3n^2$*, it belongs to BPP complexity class.*

Proof: Assume there are two indices i and j such that $(AB)_{ij} \neq C_{ij}$. Let $D = C - AB$. Then $D_{ij} \neq 0$, $D \neq 0$. We want to prove $\underset{r \in \{0,1\}^n}{\mathbb{P}}\left[Dr = 0\right] \leq \frac{1}{2}$.

$$(Dr)_i = \sum_k D_{ik} r_k = D_{ij} r_j + f((r_k)_{k \neq j})$$

$$\mathbb{P}\left[Dr = 0\right] \leq \mathbb{P}\left[(Dr)_i = 0\right]$$

Fix $r_1, \ldots, r_n$ excepts r_j. Then $v = f((r_k)_{k \neq j})$.

- If $v = -D_{ij}$: if $r_j = 0$ then $(Dr)_i \neq 0$, if $r_j = 1$ then $(Dr)_i = D_{ij} - D_{ij} = 0$. Conditional probability of $(Dr)_i = 0$ is $\frac{1}{2}$.

- If $v = 0$: if $r_j = 0$ then $(Dr)_i = 0$, if $r_j = 1$ then $(Dr)_i = D_{ij} \neq 0$. Conditional probability of $(Dr)_i = 0$ is $\frac{1}{2}$.
- Otherwise: for $r_j = 0, 1$ $(Dr)_i \neq 0$.

$$\mathbb{P}\left[(Dr)_i = 0\right] \leq \frac{1}{2}$$

□

1.2.2 Finding prime numbers

Primality testing

Decision problem:

- input: an integer $N \geq 2$
- output: decide if N is prime

The sieve of Eratosthenes gives a result in $\sqrt{N}$ steps which is too long.

Theorem 1.5. *Fermat's little theorem: p prime number $\Rightarrow \forall a \in [1, p-1], a^{p-1} = 1\,[p]$*

Two random primality tests are based on the above theorem:

- Miller-Rabin test: $O((\log N)^2)$ running time
- Solovay-Strassen test: $O((\log N)^2)$ running time

Tentative algorithm

Lemma 1.6. *Assume there is $1 \leq a < N$ such that $a \wedge N = 1$ and $a^{N-1} \neq 1\,[N]$. Then* $\underset{1 \leq a < N}{\mathbb{P}}[a^{N-1} = 1\,[N]\,|a \wedge N = 1] \leq \frac{1}{2}$

Primality test algorithm:

- Input: $N \geq 2$
- Select a random $a \in [1, N-1]$
- If $a \wedge N \neq 1$ then reject (in this case N is not prime, because $(a \wedge N)|N$)
- Compute a^{N-1} with rapid exponentiation: $a^{2r} = (a^r)^2$, $a^{2r+1} = a(a^r)^2$
- Accept if $a^{N-1} = 1\,[N]$, otherwise reject

Remarks:

- Running time is $O(\log N)$.
- If N is prime then the algorithm accepts N with probability 1.

Corollary 1.7. *Assume there is $1 \leq a < N$ such that $a \wedge N = 1$ and $a^{N-1} \neq 1\,[N]$. Then* $\underset{a}{\mathbb{P}}(\textit{algorithm accepts } N) \leq \frac{1}{2}$

Proof: Take N non prime such that there is $1 \leq a < N$ such that $a \wedge N = 1$ and $a^{N-1} \neq 1\,[N]$

$$\begin{aligned}
\underset{a}{\mathbb{P}}(\textit{algorithm accepts } N) &= \underset{a}{\mathbb{P}}(a \wedge N = 1 \textit{ and } a^{N-1} = 1\,[N]) \\
&= \underbrace{\underset{a}{\mathbb{P}}(a^{N-1} = 1\,[N]\,|a \wedge N = 1)}_{\leq \frac{1}{2}} \times \underbrace{\underset{a}{\mathbb{P}}(a \wedge N = 1)}_{\leq 1} \\
&\leq \frac{1}{2}
\end{aligned}$$

$\square$

Definition 1.8. *An integer N is a Carmichael number if there is $1 \leq a < N$ such that $a \wedge N = 1$ and $a^{N-1} \neq 1\,[N]$ and N is not prime.*

The smallest Carmichael number is $561 = 3 \times 11 \times 17$

Miller-Rabin test

Lemma 1.9. *If p is prime then the only solution of $x^2 = 1\,[p]$ are $\pm 1 \mod p$.*

Algorithm:

- Input: $N \geq 2$
- If $N = 2$, ACCEPT. Otherwise if $2|N$, REJECT.
- Take $a \in [2, N-1]$ uniformly at random.
- If $a \wedge N \neq 1$, REJECT
- Let $N - 1 = 2^t u$ ($t \geq 1$ since N is odd). Compute $b = a^u$. Let $i \leq t$ be the smallest integer such that $b^{2^i} = 1$.
- If i does not exist, REJECT (since $b^{2^t} \neq 1\,[N]$, Fermat's test fails)
- If $i = 0$ or $b^{2^{i-1}} = -1$, ACCEPT
- Otherwise, REJECT

Remark: Running time is $O(\log N)$.

Prime finding algorithm

Relational problem:

- input: integer N
- output: prime $p \in [N, 2N]$

Theorem 1.10. *Let $\pi(x)$ be the number of prime numbers lower than x. Then $\pi(x) \sim \frac{x}{\ln x}$ while $x \to +\infty$*

Algorithm:

- Take a random $p \in [N, 2N]$
- Check if p is prime
- If p is prime, output p
- If not, start again

Analysis: The number of primes between N and $2N$ is $\pi(2N) - \pi(N-1) \sim \frac{2N}{\ln 2N} - \frac{N}{\ln N} \sim \frac{N}{\ln N}$. Therefore p is prime with probability $\sim \frac{1}{\ln N}$.

Lemma 1.11. *Let $X_1, X_2, \ldots, X_n$ be a sequence of random variables in 0,1 such that $\forall i, \mathbb{P}(X_i = 1) \geq p$ and T be the first i such that $X_i = 1$. Then $\mathbb{E}\left[T\right] \leq \frac{1}{p}$*

The expected number of iterations before finding a prime is $\sim \ln N$. The expected time complexity of the algorithm is $O(\log N) \times (\textit{primality test complexity})$.

1.2.3 Polynomial identity testing

Problem

- input: two polynomials $P(X_1, X_2, \ldots, X_n), Q(X_1, X_2, \ldots, X_n)$ of degree $\leq d$
- output: decide if $P = Q$

Representation of P and Q: P and Q are represented as a black box, such that they can be evaluated efficiently, given $a_1, a_2, \ldots, a_n$ The complexity of an algorithm is the number of evaluations of P and Q.
Remark: Checking if $P = Q$ can be done by expanding them but it will cost an exponential time in their representation size.

Lemma 1.12. *Schwartz-Zippel: Let F be a field and $S \subset F$. Let $P(X_1, \ldots, X_n)$ be a non-zero polynomial of degree $\leq d$. Then* $\underset{a_1,\ldots,a_n \in S}{\mathbb{P}}(P(a_1, \ldots, a_n) = 0) \leq \frac{d}{|S|}$

Proof: By induction on n. $n = 1$ is easy since P has at most d roots. □

Algorithm 1

- $S = \{1, 2, 3, \dots, 2d\}$
- Select $a_1, \dots, a_n \in S$ at random
- Accept if $P(a_1, \dots, a_n) = Q(a_1, \dots, a_n)$
- Reject otherwise

Analysis:

- Complexity: 2 evaluations.
- If $P = Q$ then the algorithm accepts with probability 1
- If $P \neq Q$ then $\mathbb{P}(algorithm\, accepts) \leq \frac{d}{|S|} \leq \frac{1}{2}$ with Schwartz-Zippel's lemma

Algorithm 2

Assume the greatest coefficient of P and Q is lower than M.
Issue: Find p such that $P = Q \Leftrightarrow P = Q \mod p$
Then take $p \geq 2M$. In order to adapt the previous algorithm, we also need $p \geq 2d$.
First step: Find a prime between N and $2N$ where N is the maximum of $2d$ and $2M$.
Then it is same algorithm than the first one but we accept if $P(a_1, \dots, a_n) = Q(a_1, \dots, a_n)$ mod p

1.2.4 Fingerprints

Problem

- There are 2 players A and B.
- A's input: u, sequence of n bits
- B's input: v, sequence of n bits
- output: decide if $u = v$
- complexity = number of bits exchanged between A and B

A naive solution would be: A sends u to B. But it costs n bits.

Hashing

We define two polynomials P_u and P_v:

$$\begin{aligned} u &= u_0, u_1, \ldots, u_{n-1} \\ v &= v_0, v_1, \ldots, v_{n-1} \\ P_u &= u_0 + u_1 X + \cdots + u_{n-1} X^{n-1} \\ P_v &= v_0 + v_1 X + \cdots + v_{n-1} X^{n-1} \end{aligned}$$

Remark: $u = v \Leftrightarrow P_u = P_v$
Random hash value:

- Take a prime p between n^2 and $2n^2$.
- Select a random a between 0 and $p - 1$.
- $P_u(a) \mod p$ is the fingerprint of u in $a \mod p$.

Protocol

Algorithm:

1. A selects p and a as described above
2. A sends $P_u(a) \mod p$ to B
3. B checks if $P_u(a) = P_v(a)\,[p]$. If yes, B accepts. Else, B rejects.

Remarks:

- Number of exchanged bits: $6 \log n$
- If $u = v$, B accepts with probability 1. Otherwise, B accepts with probability $\leq \frac{1}{n}$.