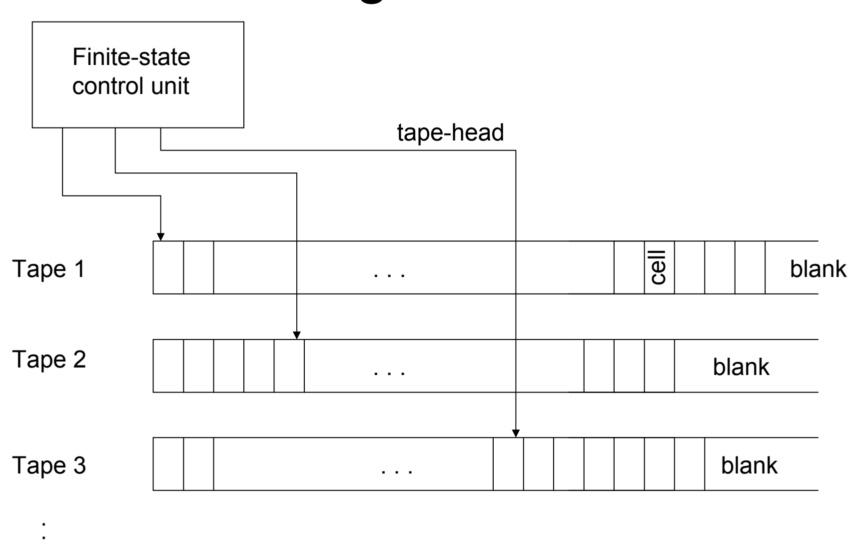
T-79.5502 Advanced Course in Cryptology

Lecture 2, March 21, 2006
Computational Complexity
•Turing Machines
•Deterministic Polynomial Time
•Probabilistic Polynomial Time
•Non-deterministic Polynomial Time
•Non-Polynomial Bounds
•Polynomial-time Indistinguishability

Turing Machine



Turing computation

- A finite number of symbols are placed in the leftmost cells of the tape. The remaining cells to the right are set to blank.
- When in initial state the scanning starts from the leftmost cell.
- The tapeheads read contents of the cells. A step of access by tapehead is called a legal move.
- When a termination condition is reached, the machine is said to recognize the input.
- An input which can reach a termination condition is called an instance in a recognizable language.

Deterministic Polynomial Time – Class $\mathcal{P}$

Definition 4.1: We write $\mathcal{P}$ to denote the class of languages with the following characteristics. A language L is in $\mathcal{P}$ if there exists a Turing machine M and a polynomial $p(n)$ such that M recognizes any instance $I \in L$ in time $T_M(n)$ with $T_M(n) \leq p(n)$, for all $n \geq 0$, where n is an integer representing the size of an instance I. Then we say that L is recognizable in polynomial time.

Languages in $\mathcal{P}$ can always be recognized with a deterministic Turing machine. A deterministic Turing machine outputs a result which is entirely determined by the input to, and the initial state of the machine.

Example: Language DIV3

The finite state control unit of DIV3

Current state	Symbol on the tape	Next move	New state
q_0	0 1 "blank"	right right "yes" and stop	q_0 q_1 -
q_1	0 1 "blank"	right right "no" and stop	q_2 q_0 -
q_2	0 1 "blank"	right right "no" and stop	q_1 q_2 -

Polynomial-Time Computational Problems

- The problems in $\mathcal{P}$ are decisional problems, output is one bit.
- Since Turing machines can also write symbols on the tape, they can handle polynomial-time computational problems.
- E.g., using DIV3 repeatedly a Turing machine can compute base-3 representation of a given non-negative integer x, and hence a Turing machine can compute division with 3 in time $C \cdot |x|$.

Von Neumann Architecture

- Building blocks: counter, memory, CPU
- Micro-instructions: Load, Store, Add, Comp, Jump, JumpZ, Stop
- Any problem solvable using von Neumann computer in polynomial time, is in $\mathcal{P}$.
- Turing machine has a uniform cost measure
- Von Neumann (circuit based) computers have non-uniform cost measure.

Order notations

Definition 4.2. We write $\mathcal{O}(f(n))$ to denote a function $g(n)$ such that there exists a constant $c > 0$ and a natural number N with $|g(n)| \leq c|f(n)|$, for all $n \geq N$.

Bitwise Computation Model: All variables have the values 0 or 1, and the operations used are logical rather than arithmetic: $\oplus$, $\wedge$, $\vee$, $\neg$

Definition 4.3. We write $\mathcal{O}_B(f(n))$ to denote $\mathcal{O}(f(n))$ in the bitwise computation model.

Example. Extended Euclidean Algorithm has time complexity $\mathcal{O}(|a|)$ (Thm 4.1) and $\mathcal{O}_B((\log |a|)^3)$ (can be shown using Fibonacci sequences).

Basic Modular Arithmetic Operations

Operation for $a, b \in [1,n-1]$	Time Complexity
$a \pm b \pmod n$	$\mathcal{O}_B(\log n)$
$a \cdot b \pmod n$	$\mathcal{O}_B((\log n)^2)$
$b^{-1} \pmod n$	$\mathcal{O}_B((\log n)^2)$
$a/b \pmod n$	$\mathcal{O}_B((\log n)^2)$
$a^b \pmod n$	$\mathcal{O}_B((\log n)^3)$

Probabilistic Polynomial Time - $\mathcal{PP}$

Non-deterministic Turing machines make random moves, and errors become possible. Non-deterministic Turing machine with a bounded error is a probabilistic Turing machine.

Definition 4.5. We write $\mathcal{PP}$ to denote the class of languages with the following characteristics. A language L is in $\mathcal{PP}$ if there exists a probabilistic Turing machine PM and a polynomial $p(n)$ such that PM recognizes any instance $I \in L$ with certain error probability, which is a random variable of PM's random move, in time $T_{PM}(n)$ with $T_{PM}(n) \leq p(n)$ for all nonnegative integers n, where n is an integer parameter representing the size of the instance I.

Error Probabilities

Prob[PM recognizes $I \in L \mid I \in L$] ≥ ε

Prob[PM recognizes $I \in L \mid I \notin L$] ≤ δ

where the probabilities are taken over the random tape (random moves) of PM.

The bounds ε and δ are constans such that

½ < ε ≤ 1 and 0 ≤ δ < ½ . That is

Prob[PM recognizes $I \notin L \mid I \in L$] ≤ 1- ε < ½

ε is the completeness probability bound (1- ε is the upper bound for probabilities of false rejection)

δ is the soundness probability bound (that is, the upperbound of probability for false acceptance)

Always fast and always correct - $\mathcal{ZPP}$

ε = 1 and δ = 0

Example: Searching Through Phone Book

Always Fast and Probably Correct- $\mathcal{PP}$(MonteCarlo)

ε = 1 and $\delta > 0$

Example: Solovay-Strassen primality test

Input: p a positive integer

a, $1 < a < p - 1$, check if

$$\left(\frac{a}{p}\right) = a^{p-1} (\text{mod } p)$$

no-biased algorithm: rejection is always correct

Probably Fast and Always Correct $\mathcal{PP}$(Las Vegas)

$\varepsilon < 1$ and $\delta = 0$

May terminate without output, but if there is output it is always correct

Example 1: Finding collisions

Example 2: Quantum Factorization

For any N composite, the proportion of a, for which the least integer r such that $a^r = 1 (\text{mod } N)$ is even, is non-negligible. Then one can find non-trivial square roots of 1, which is sufficient to factor N.

Probably Fast and Probably Correct $\mathcal{BPP}$

$\frac{1}{2} + \alpha \leq \varepsilon < 1$ and $0 < \delta \leq \frac{1}{2} - \beta$,

where $\alpha, \beta \in (0, \frac{1}{2})$

Bounded error probability Probabilistic Polynomial time

"Atlantic City Algorithms"

Example: Quantum Key Distribution

Efficient Algorithms

$$\mathcal{P} \subseteq \mathcal{ZPP} \subseteq \left\{ \begin{matrix} \mathcal{PP}\text{(Monte Carlo)} \\ \mathcal{PP}\text{(Las Vegas)} \end{matrix} \right\} \subseteq \mathcal{BPP} \subseteq \mathcal{PP}$$

Definition4.6: An algorithm is said to be efficient if it is deterministic or randomised with execution time bounded from above by a polynomial in the size of the input.

Non-deterministic Polynomial Time $\mathcal{NP}$

Example: Square-Freeness

Input: N a positive and odd composite integer

Question: Is N square-free? Answer YES if there exists no prime p such that $p^2|N$.

Solution: Use witness $\phi(N)$. If $p^2|N$ then $p \mid \gcd(N, \phi(N))$

An algorithm to recognize languages in $\mathcal{NP}$ has at each step a finite number of possible moves. The algorithm recognizes L if there exists at least one sequence of legal moves (recognition sequence) leading to the terminating condition. Such a sequence may be difficult to find, but its existence is can be verified in polynomial time given a witness.

Complexity hierarchy

$\mathcal{P} \subseteq \mathcal{ZPP}$, $\mathcal{PP}$(Monte Carlo) $\subseteq$ $\mathcal{NP} \subseteq$ $\mathcal{PP}$

Definition 4.10. We say that a language L is polynomially reducible to another language L_0 if there exists a deterministic polynomially-bounded Turing machine M which will convert each instance $I \in L$ into instance $I_0 \in L_0$, such that $I \in L$ if and only if $I_0 \in L_0$.

Definition 4.11. A language $L_0 \in \mathcal{NP}$ is $\mathcal{NP}$-complete if any $L \in \mathcal{N}$P is polynomially reducible to L_0.

Example: Satisfiability

Non-Polynomial Bounds

Definition 4.12. A function $f(\mathrm{n})$: **N** → **R** is said to be unbounded by any polynomial in n (or, non-polynomially bounded quantity) if for any polynomial $p(n)$ there exists a natural number n_0 such that $f(n) > p(n)$, for all $n > n_0$.

Definition 4.13. A function $\varepsilon(\mathrm{n})$: **N** → **R** is said to be a negligible in n if its inverse $1/\varepsilon(\mathrm{n})$ is a non-polynomially bounded quantity.

Polynomial-time Indistinguishability

Definition 4.14. Let S be a set and E and E' be subsets of S. A distinguisher $\mathcal{D}$ is a probabilistic algorithm, which makes use of ℓ elements $a \in S$ where $\ell \leq k$ and which halts in time polynomial in k with output in $\{0,1\}$. $\mathcal{D}$ satisfies $\mathcal{D}(a,E) = 1$ if $a \in E$, and $\mathcal{D}(a,E') = 1$ if $a \in E'$. Then we say that $\mathcal{D}$ distinguishes E, E' with advantage Adv($\mathcal{D}$) > 0, if

$$\mathrm{Adv}(\mathcal{D}) = |\mathrm{Prob}[\mathcal{D}(a,E) = 1] - \mathrm{Prob}[\mathcal{D}(a,E') = 1]| > 0.$$

Definition 4.15. Let sets E and E' and security parameter k be as defined in Definition 4.14. Then E, E' are said to be polynomially indistiguishable if there exists no distinguisher for E, E' for which Adv($\mathcal{D}$)> 0 is non-negligible in k for sufficiently large k.

Example: Prime_Gen(k)

Algorithm 4.7: Random k-bit Probabilistic Prime Generation

INPUT: k: a positive integer (input to be written to have size k)

OUTPUT: a k-bit random prime

Prime_Gen(k)

1. $p \in_U (2^{k-1}, 2^k - 1]$ with p odd;
2. if Prime_Test(p) = NO, return (Prime_Gen(k));
3. Return (p)

Here we make use of Prime_Test (p) which is $\mathcal{PP}$(Monte Carlo) with $\varepsilon = 1$ and $\delta = 2^{-k}$. E.g., Solovay-Strassen repeated k times.

Prime_Gen(k)

Prime_Gen(k) is an $\mathcal{PP}$(Atlantic City) with $\varepsilon > 1/2$ and $\delta \cong 2^{-k}$.

(After k rounds the probability that the algorithm has halted is at least ½ as proportion of primes in k-bit odd numbers is about $1/k$)

"input k to be written to have size k"

Unary representation of a number

Definition 4.7. The unary representation of a positive natural number is

$$1^n = \underbrace{111\ldots1}_{n \text{ times}}$$