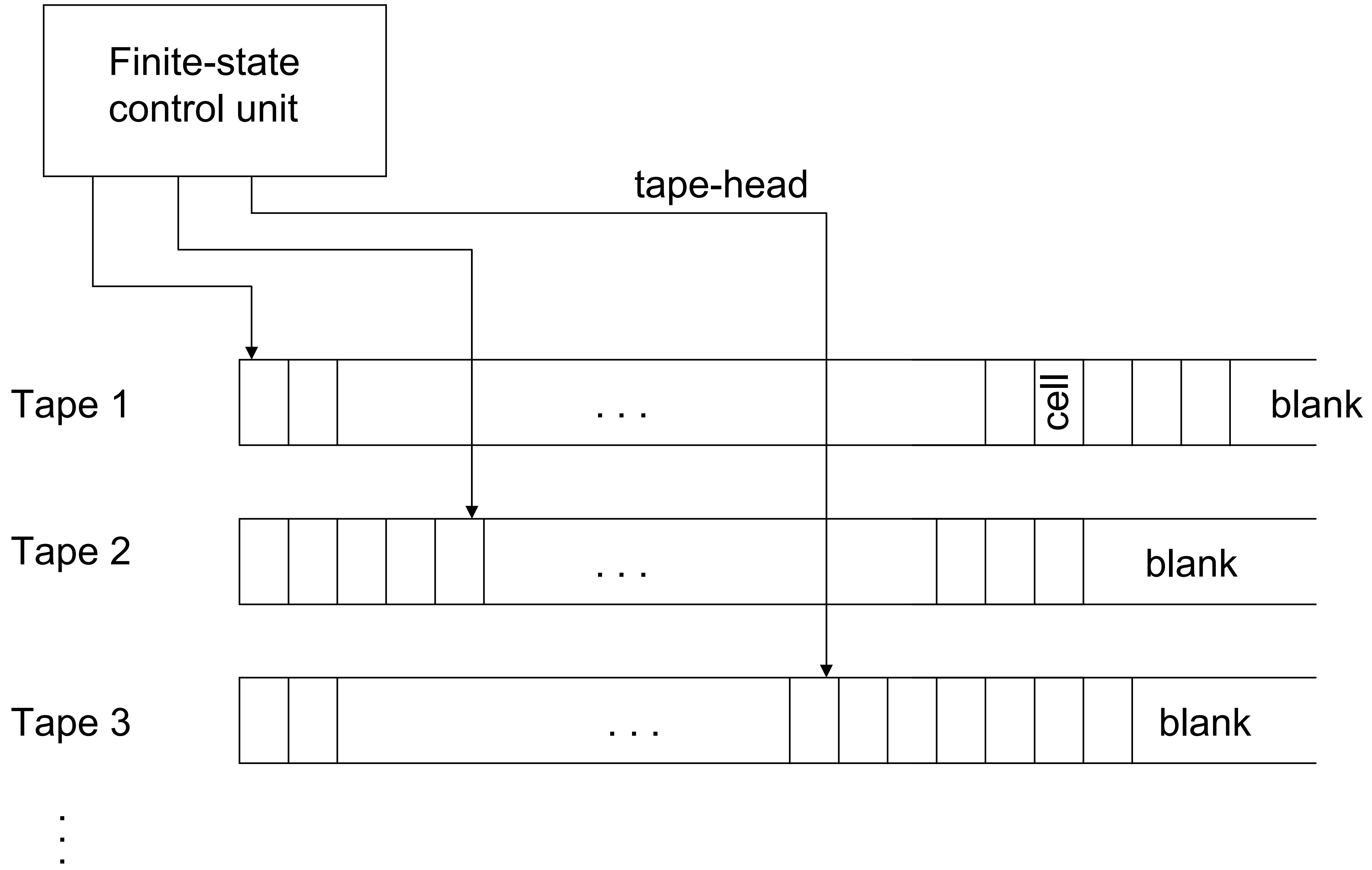
T-79.5502 Advanced Course in Cryptology

Lecture 2, November 6, 2007

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 set to the initial state, the scanning starts from the leftmost cell.
- The tape heads read contents of the cells.
- When a termination condition is reached, the machine is said to recognise the input.
- An input which can reach a termination condition is called an instance in the recognisable 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 recognises 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 recognisable in polynomial time.

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

Example: Language DIV3

The finite state control unit of DIV3

Intial state q_0

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

Polynomial-Time Computational Problems

- The problems in $\mathcal{P}$ are decisional problems, output is one bit: recognised or not.
- Turing machines can also write symbols on the tape. Then they can handle also 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 by 3 in time $C \cdot |x|$, where |x| denotes the number of bits in the binary representation of the integer 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 $O(f(n))$ 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 take values 0 or 1, and the operations used are logical rather than arithmetic: $\oplus$, $\wedge$, $\vee$, $\neg$

Definition 4.3. We write $O_B(f(n))$ to denote $O(f(n))$ in the bitwise computation model.

Example. Given two integers a and b asume that the absolute value $|a|$ of a is larger than the absolute value $|b|$ of b. The Extended Euclidean Algorithm has time complexity at most $O(|a|)$ (Thm 4.1). Further, it can be shown using Fibonacci sequence that it has bit complexity $O_B((\log |a|)^3)$.

Basic Modular Arithmetic Operations

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

Probabilistic Polynomial Time - $\mathcal{PP}$

Non-deterministic Turing machines make random moves. That is, one of the tapes of a non-deterministic Turing machine is a random tape which contains uniformly distributed random symbols.

Non-deterministic Turing machine make errors. Non-deterministic Turing machine with a bounded error is called 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 said to be in $\mathcal{PP}$ if there exists a probabilistic Turing machine PM and a polynomial $p(n)$ such that PM recognises 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

$\text{Prob}[PM$ recognises $I \in L \mid I \in L] \geq \varepsilon$

$\text{Prob}[PM$ recognises $I \in L \mid I \notin L] \leq \delta$

where the probability space is the random tape of PM.

The bounds ε and δ are constans such that

$½ < \varepsilon \leq 1$ and $0 \leq \delta < ½$. That is

$\text{Prob}[PM$ recognises $I \notin L \mid I \in L] \leq 1- \varepsilon < ½$

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 - $\mathcal{ZPP}$

$\mathcal{ZPP}$ = Zero-sided-error Probabilistic Polynomial time

ε = 1 and δ = 0

Example: Searching Through Phone Book

Input: Book (an alphabetic list of names) with N pages, Person's name

Output: The person's phone number

Algorithm: Open the book at a random page. If the name occurs on that page, output the phone number. Otherwise, select the part of the book that contains the name, take it as the book, and repeat the algorithm from the beginning.

Time complexity $O(\log N)$, where N is the number of pages. Hence this randomized algorithm is faster than the deterministic algorithm for searching the 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

Output: Yes if p is prime, otherwise No.

Algorithm: Select random integer a, $1 < a < p - 1$, check if

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

This equality is known as the Euler's criterion, which holds for primes p and may hold for composites. Therefore rejection is always correct, that is, the algorithm is no-biased. Acceptance is false with probability less than ½.

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: Primality proof (see next slide)

Example 2: Quantum Factorization

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

$\mathcal{ZPP} = \mathcal{PP}$(Monte Carlo)$\cap\mathcal{PP}$(Las Vegas)

Primality Proof- A $\mathcal{PP}$(Las Vegas) Algorithm

Input: p a positive integer and all prime factors $q_1, q_2, q_3, \ldots, q_k$ of p-1

Output: Yes if p is prime, otherwise No.

Algorithm: Select random integer a, $1 < a \le p - 1$.

1. For all $i = 1,\ldots,k$, check if the following holds:

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

If it holds for some i, output No-decision and terminate.

2. Check if

$$a^{p-1} \neq 1(\text{mod } p)$$

    If it holds, output No and terminate.

3. Output Yes and teminate.

Yes and No are always correct. No-decision results if a is not a primitive element modulo p.

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

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

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

$\mathcal{BPP}$ = Bounded error probability Probabilistic Polynomial time or

“Atlantic City Algorithms”

Example 1: Quantum Key Distribution

Example 2: Prime_Gen(k)

Quantum Key Distribution

Goal: Agree on a fixed number (k - l) of secret bits

Quantum Channel: Alice sends to Bob m photon states $\in$ {—, | , /, \ }

Open Channel: They choose k = m/10 "sifted bits" from the locations where Alice's polarizers agree with Bob's observers. They further compare random l (< k) "testing bits" in the k sifted bits to detect eavesdropping. If eavesdropper not detected the accept the remaining k - l shared secret bits.

QKD Error Probabilities

- Completeness error: less than $m/10$ of Bob's observers agree with Alice's polarizers

  probability $\cong 3/m$

This follows from the fact that the number of Bob's observers that agree with Alice's polarizers is binomially distributed in $[0,1,..,m]$ with expected value $m/2$.

- Soundness error: Alice and Bob do not detect an eavesdropper

  probability $= (3/4)^l$

Eavesdropper's attack strategy is to forward what it gets by observing the quantum channel. She receives correctly a photon state with probability ½. If she receives correctly, then she succeeds, and if she receives wrong then she succeeds with probability ½. The total success probability for a photon state is $1/2 + 1/2 \cdot 1/2 = 3/4$.

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 $1/k$.

The time complexity of Prime_Gen(k) is bounded by $O(k^5)$. This is polynomial in the size of the input, if the input k is written to have size k. This can be done using the unary representation.

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

$$1^k = \underbrace{111 \ldots 1}_{k \text{ times}}$$

Efficient Algorithms

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

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 recognise languages in $\mathcal{NP}$ has at each step a finite number of possible moves. The algorithm recognises 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

$\mathcal{P} \subseteq \mathcal{ZPP} \subseteq \mathcal{PP}$(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 $L \in \mathcal{N}\mathrm{P}$ is polynomially reducible to L_0.

Example: SAT is in $\mathcal{NP}$, Knapsack is in $\mathcal{NP}$

Non-Polynomial Bounds

Definition 4.12. A function $f\,(\mathrm{n})$: $\mathbf{N} \rightarrow \mathbf{R}$ is said to be unbounded by any polynomial in n (or, a 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 $\varepsilon(n)$: $\mathbf{N} \rightarrow \mathbf{R}$ is said to be a negligible in n if its inverse $1/\varepsilon(n)$ 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. Denote by $k = \log_2 |S|$. A distinguisher $\mathcal{D}$ is a probabilistic algorithm, which makes use of ℓ random elements $a \in S$, where ℓ is bounded by a polynomial in 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 $a \in E'$. We set

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

where the probabilities are taken over the distribution of a. If $\mathrm{Adv}(\mathcal{D}) > 0$, we say that $\mathcal{D}$ is a distinguisher for the sets E and E' with advantage $\mathrm{Adv}(\mathcal{D})$.

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 $\mathrm{Adv}(\mathcal{D})$ is non-negligible in k, for sufficiently large k.