Theory of Computation Lecture Notes

Prof. Yuh-Dauh Lyuu

Dept. Computer Science & Information Engineering

and

Department of Finance

National Taiwan University

Class Information

- Papadimitriou. *Computational Complexity.* 2nd printing. Addison-Wesley. 1995.
  - The best book on the market for graduate students.
  - We more or less follow the topics of the book.
  - More "advanced" materials may be added.
- Check

  `www.csie.ntu.edu.tw/~lyuu/complexity/2003`

  for last year's lecture notes.
- You may want to review discrete mathematics.

Class Information (concluded)

- More information and future lecture notes (in PDF format) can be found at

  `www.csie.ntu.edu.tw/~lyuu/complexity.html`

- Please ask many questions in class.
  - The best way for me to remember you in a large class.[a]
- Teaching assistants will be announced later.

[a] "[A] science concentrator [...] said that in his eighth semester of [Harvard] college, there was not a single science professor who could identify him by name." (*New York Times*, September 3, 2003.)

Grading

- No roll calls.
- No homeworks.
  - Try some of the exercises at the end of each chapter.
- Two to three examinations.
- You must show up for the examinations, in person.
- If you cannot make it to an examination, please email me beforehand (unless there is a legitimate reason).
- Missing the final examination will earn a “fail” grade.

A Brief History (Biased towards Complexity)

1930–1931: Gödel's (1906–1978) completeness and incompleteness theorems and recursive functions.

1935–1936: Kleene (1909–1994), Turing (1912–1954), Church (1903–1995), Post (1897–1954) on computability.

1936: Turing defined Turing machines and oracle Turing machines.

1938: Shannon (1916–2001) used boolean algebra for the design and analysis of switching circuits. Circuit complexity was also born. Shannon's master's thesis was "possibly the most important, and also the most famous, master's thesis of the century."

A Brief History (continued)

1947: Dantzig invented linear programming simplex algorithm.

1947: Paul Erdős (1913–1996) popularized the probabilistic method. (Also Shannon (1948).)

1949: Shannon established information theory.

1949: Shannon's study of cryptography was published.

1956: Ford and Fulkerson's network flows.

1959: Rabin and Scott's notion of nondeterminism.

A Brief History (continued)

1964–1966: Solomonoff, Kolmogorov, and Chaitin formalized Kolmogorov complexity (program size and randomness).

1965: Hartmanis and Stearns started complexity theory and hierarchy theorems (see also Rabin (1960)).

1965: Edmonds identified NP and P (actual names were coined by Karp in 1972).

1971: Cook invented the idea of NP-completeness.

1972: Karp established the importance of NP-completeness.

1972–1973: Karp, Meyer, and Stockmeyer defined the polynomial hierarchy.

A Brief History (continued)

1973: Karp studied PSPACE-completeness.

1973: Meyer and Stockmeyer studied exponential time and space.

1973: Baker, Gill, and Solovay studied "NP=P" relative to oracles.

1975: Ladner studied P-completeness.

1976–1977: Rabin, Solovay, Strassen, and Miller proposed probabilistic algorithms (for primality testing).

1976–1978: Diffie, Hellman, and Merkle invented public-key cryptography.

A Brief History (continued)

1977: Gill formalized randomized complexity classes.

1978: Rivest, Shamir, and Adleman invented RSA.

1978: Fortune and Wyllie defined the PRAM model.

1979: Garey and Johnson published their book on computational complexity.

1979: Valiant defined #P.

1979: Pippenger defined NC.

1979: Khachiyan proved that linear programming is in polynomial time.

1979: Yao founded communication complexity.

A Brief History (continued)

1980: Lamport, Shostak, and Pease defined the Byzantine agreements problem in distributed computing.

1981: Shamir proposed cryptographically strong pseudorandom numbers.

1982: Goldwasser and Micali proposed probabilistic encryption.

1982: Yao founded secure multiparty computation.

1982: Goldschlager, Shaw, and Staples proved that the maximum flow problem is P-complete.

1982–1984: Yao, Blum, and Micali founded pseudorandom number generation on complexity theory.

A Brief History (continued)

1983: Ajtai, Komlós, and Szemerédi constructed an $O(\log n)$-depth, $O(n \log n)$-size sorting network.

1984: Valiant founded computational learning theory.

1984–1985: Furst, Saxe, Sipser, and Yao proved exponential bounds for parity circuits of constant depth.

1985: Razborov proved exponential lower bounds for monotone circuits.

1985: Goldwasser, Micali, and Rackoff invented zero-knowledge proofs.

1985: Sleator and Tarjan invented on-line algorithms.

A Brief History (continued)

1986: Goldreich, Micali, and Wigderson proved that every problem in NP has a zero-knowledge proof under certain complexity assumptions.

1987: Adleman and Huang proved that primality testing can be solved in randomized polynomial time.

1987–1988: Szelepscényi and Immerman proved that NL equals coNL.

1989: Blum and Kannan proposed program checking.

1990: Shamir proved IP=PSPACE.

1990: Du and Hwang settled the Gilbert-Pollak conjecture on Steiner tree problems.

A Brief History (concluded)

1992: Arora, Lund, Motwani, Sudan, and Szegedy proved the PCP theorem.

1993: Bernstein, Vazirani, and Yao established quantum complexity theory.

1994: Shor presented a quantum polynomial-time algorithm for factoring.

1996: Ajtai on the shortest lattice vector problem.

2002: Agrawal, Kayal, and Saxena discovered a polynomial-time algorithm for primality testing.

Problems and Algorithms

I have never done anything "useful."
— Godfrey Harold Hardy (1877–1947),
A Mathematician's Apology (1940)

What This Course Is All About

Computability: What can be computed?

- What is computation anyway?
- There are *well-defined* problems that cannot be computed.
- In fact, "most" problems cannot be computed.

What This Course Is All About (concluded)

Complexity: What is a computable problem's inherent complexity?

- Some computable problems require at least exponential time and/or space; they are **intractable**.
- Some practical problems require superpolynomial resources unless certain conjectures are disproved.
- Other resource limits besides time and space?
  - Program size, circuit size (growth), number of random bits, etc.

Tractability and intractability

- Polynomial in terms of the input size n defines tractability.
  - n, $n \log n$, n^2, n^{90}.
  - Time, space, circuit size, number of random bits, etc.
- It results in a fruitful and practical theory of complexity.
- Few practical, tractable problems require a large degree.
- Exponential-time or superpolynomial-time algorithms are usually impractical.
  - $n^{\log n}$, $2^{\sqrt{n}}$, 2^n, $n! \sim \sqrt{2\pi n}\,(n/e)^n$.

Growth of Factorials

n	$n!$	n	$n!$
1	1	9	362,880
2	2	10	3,628,800
3	6	11	39,916,800
4	24	12	479,001,600
5	120	13	6,227,020,800
6	720	14	87,178,291,200
7	5040	15	1,307,674,368,000
8	40320	16	20,922,789,888,000

Most Important Results: a Sampler

- An operational definition of computability.
- Decision problems in logic are undecidable.
- Decisions problems on program behavior are usually undecidable.
- Complexity classes and the existence of intractable problems.
- Complete problems for a complexity class.
- Randomization and cryptographic applications.
- Approximability.

Turing Machines

What Is Computation?

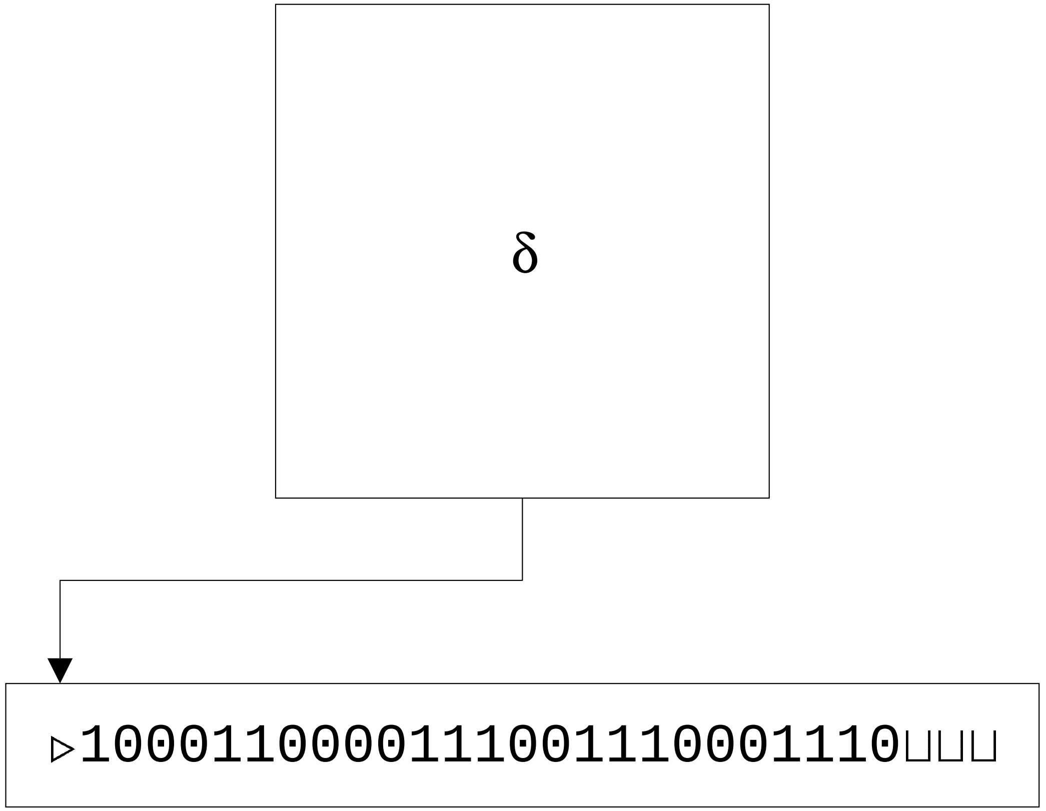
- That can be coded in an **algorithm**.
- An algorithm is a detailed step-by-step method for solving a problem.
  - The Euclidean algorithm for the greatest common divisor is an algorithm.
  - "Let s be the least upper bound of compact set A" is not an algorithm.
  - "Let s be a smallest element of a finite-sized array" can be solved by an algorithm.

Turing Machines[a]

- A Turing machine (TM) is a quadruple $M = (K, \Sigma, \delta, s)$.
- K is a finite set of **states**.
- $s \in K$ is the **initial state**.
- Σ is a finite set of **symbols** (disjoint from K).
  - Σ includes $\bigsqcup$ (blank) and $\triangleright$ (first symbol).
- $\delta : K \times \Sigma \to (K \cup \{h, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$ is a **transition function**.
  - $\leftarrow$ (left), $\rightarrow$ (right), and $-$ (stay) signify cursor movements.

[a]Turing (1936).

A TM Schema



"Physical" Interpretations

- The tape: computer memory and registers.
- δ: program.
- K: instruction numbers.
- s: "`main()`" in C.
- Σ: **alphabet** much like the ASCII code.

More about δ

- The program has the **halting state** (h), the **accepting state** ("yes"), and the **rejecting state** ("no").
- Given current state $q \in K$ and current symbol $\sigma \in \Sigma$,

$$\delta(q, \sigma) = (p, \rho, D).$$

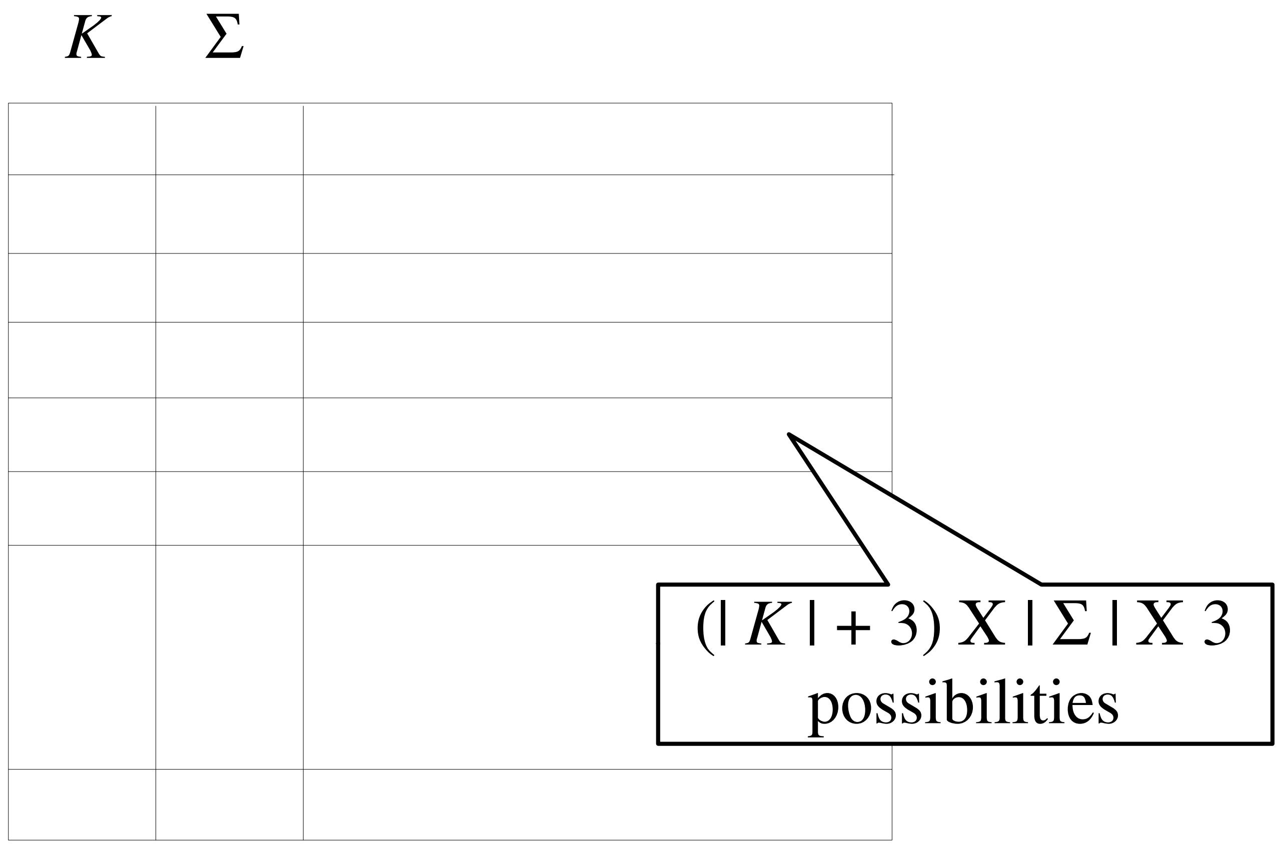
  - It specifies the next state p, the symbol ρ to be written over σ, and the direction D the cursor will move *afterwards*.
- We require $\delta(q, \triangleright) = (p, \triangleright, \rightarrow)$ so that the cursor never falls off the left end of the string.

The Operations of TMs

- Initially the state is s.
- The string on the tape is initialized to a $\rhd$, followed by a *finite-length* string $x \in (\Sigma - \{\sqcup\})^*$.
- x is the **input** of the TM.
  - The input must not contain $\sqcup$s (why?)!
- The cursor is pointing to the first symbol, always a $\rhd$.
- The TM takes each step according to δ.
- The cursor may overwrite $\sqcup$ to make the string longer during the computation.

Program Count

- A program has a *finite* size.
- Recall that
  $\delta : K \times \Sigma \to (K \cup \{h, \text{"yes"}, \text{"no"}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$.
- So $|K| \times |\Sigma|$ "lines" suffice to specify a program, one line per pair from $K \times \Sigma$.
- Given K and Σ, there are
  $$((|K| + 3) \times |\Sigma| \times 3)^{|K| \times |\Sigma|}$$
  possible δ's (see next page).
  - This is a constant—albeit large.
- Different δ's may define the same behavior.


K
Σ
$(|K| + 3) \times |\Sigma| \times 3$ possibilities

The Halting of a TM

- A TM M may **halt** in three cases.

  "yes": M **accepts** its input x, and $M(x) =$ "yes".

  "no": M **rejects** its input x, and $M(x) =$ "no".

  h**:** $M(x) = y$, where the string consists of a $\rhd$, followed by a finite string y, whose last symbol is not $\sqcup$, followed by a string of $\sqcup$s.

  - y is the **output** of the computation.
  - y may be empty denoted by ϵ.

- If M never halts on x, then write $M(x) = \nearrow$.

Why TMs?

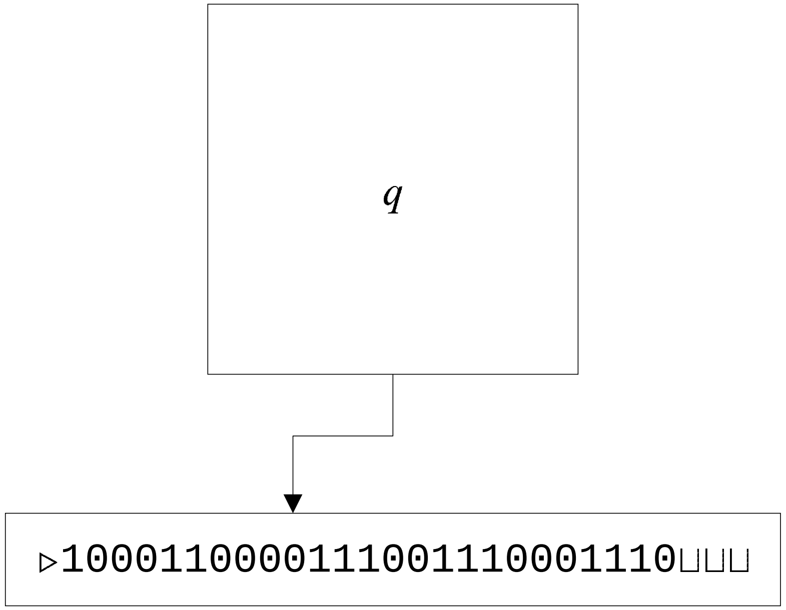
- Because of the simplicity of the TM, the model has the advantage when it comes to complexity issues.
- One can develop a complexity theory based on C++ or Java, say.
- But the added complexity does not yield additional fundamental insights.
- We will describe TMs in pseudocode.

The Concept of Configuration

- A **configuration** is a complete description of the current state of the computation.
- The specification of a configuration is sufficient for the computation to continue as if it had not been stopped.
  - What does your PC save before it sleeps?
  - Enough for it to resume work later.
- Similar to the concept of **Markov process** in stochastic processes or dynamic systems.

Configurations (concluded)

- A configuration is a triple (q, w, u):
  - $q \in K$.
  - $w \in \Sigma^*$ is the string to the left of the cursor (inclusive).
  - $u \in \Sigma^*$ is the string to the right of the cursor.
- Note that (w, u) describes both the string and the cursor position.


q
$\triangleright$1000110000111001110001110$\sqcup\sqcup\sqcup$

- $w = \triangleright$1000110000.
- $u =$ 111001110001110.

Yielding

- Fix a TM M.
- Configuration (q, w, u) **yields** configuration (q', w', u') in one step,
$$(q, w, u) \xrightarrow{M} (q', w', u'),$$
if a step of M from configuration (q, w, u) results in configuration (q', w', u').
- $(q, w, u) \xrightarrow{M^k} (q', w', u')$: Configuration (q, w, u) yields configuration (q', w', u') in $k \in \mathbb{N}$ steps.
- $(q, w, u) \xrightarrow{M^*} (q', w', u')$: Configuration (q, w, u) yields configuration (q', w', u').

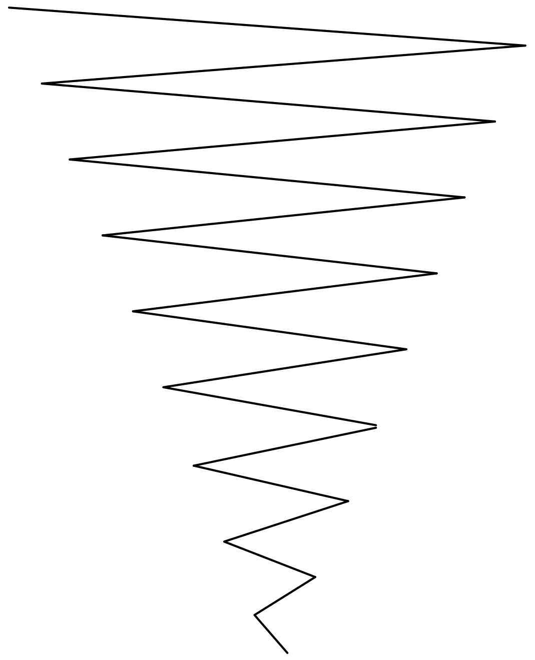
Example: How to Insert a Symbol

- We want to compute $f(x) = ax$.
  - The TM moves the last symbol of x to the right by one position.
  - It then moves the next to last symbol to the right, and so on.
  - The TM finally writes a in the first position.
- The total number of steps is $O(n)$, where n is the length of x.

Palindromes

- A string is a **palindrome** if it reads the same forwards and backwards (e.g., `001100`).
- A TM program can be written to recognize palindromes:
  - It matches the first character with the last character.
  - It matches the second character with the next to last character, etc. (see next page).
  - "yes" for palindromes and "no" for nonpalindromes.
- This program takes $O(n^2)$ steps.
- Can we do better?

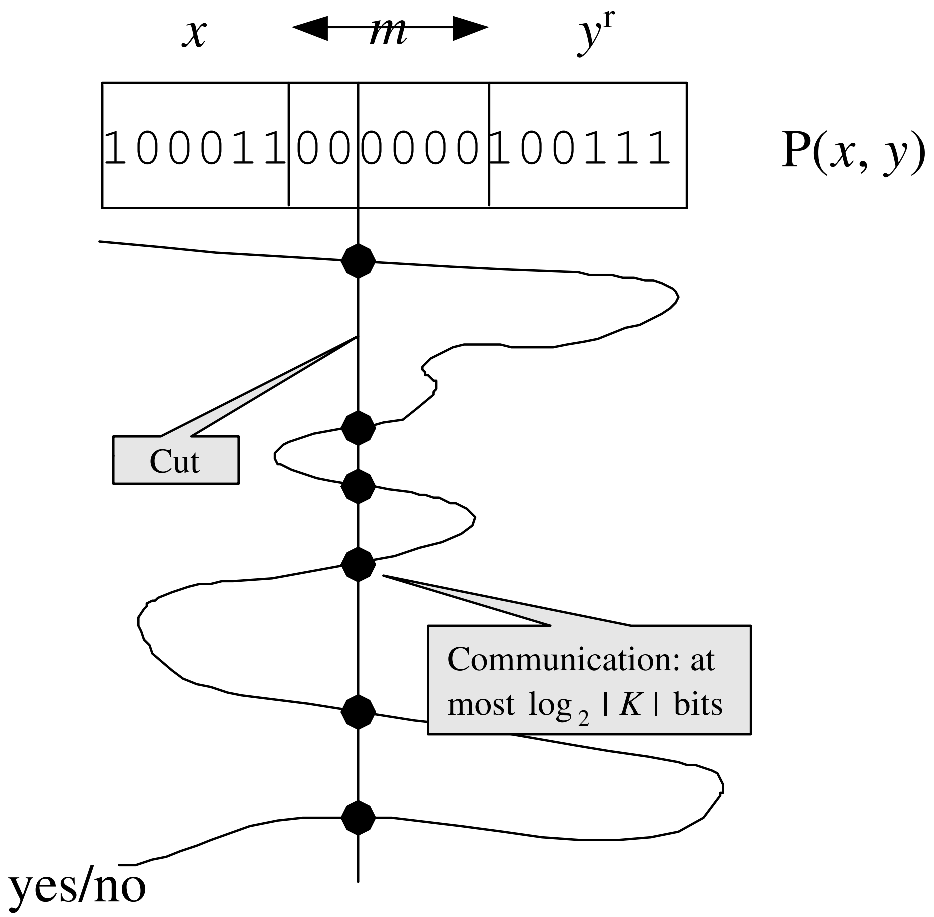
$$\boxed{100011000000100111}$$



A Matching Lower Bound for PALINDROME

Theorem 1 (Hennie (1965)) PALINDROME *on single-string TMs takes* $\Omega(n^2)$ *steps in the worst case.*

The Proof: Setup

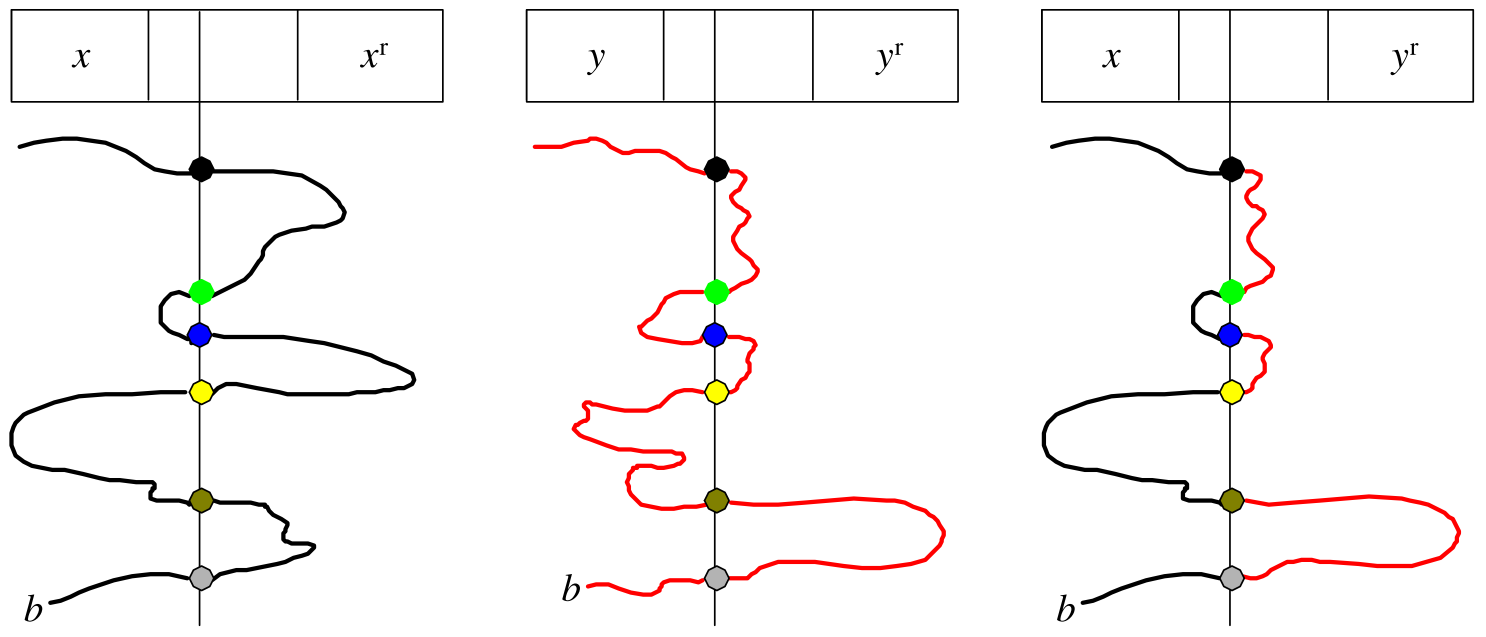


The Proof: Communications

- Our input is more restricted; hence any lower bound holds for the original problem.
- Each communication between the two halves across the cut is a state from K, hence of size $O(1)$.
- $\mathrm{C}(x, x)$: the sequence of communications for palindrome problem $\mathrm{P}(x, x)$ *across* the cut.
  - It is a sequence of states from K.

The Proof: Communications (concluded)

- $\mathrm{C}(x, x) \neq \mathrm{C}(y, y)$ when $x \neq y$.
  - Suppose otherwise, $C(x, x) = C(y, y)$.
  - Then $C(y, y) = C(x, y)$ by the cut-and-paste argument (see next page).
  - Hence $\mathrm{P}(x, y)$ has the same answer as $\mathrm{P}(y, y)$!
- So $\mathrm{C}(x, x)$ is distinct for each x.


x
x^r
b
y
y^r
b
x
y^r
b

The Proof: Amount of Communications

- Assume $|\,x\,| = |\,y\,| = m = n/3$.
- $|\,\mathrm{C}(x,x)\,|$ is the number of times the cut is crossed.
- We first seek a lower bound on the total number of communications:

$$\sum_{x \in \{0,1\}^m} |\,\mathrm{C}(x,x)\,|.$$

- Define

$$\kappa \equiv (m+1)\log_{|\,K\,|} 2 - \log_{|\,K\,|} m - 1 + \log_{|\,K\,|}(|\,K\,| - 1).$$

The Proof: Amount of Communications (continued)

- There are $\leq |\, K \,|^i$ distinct $\mathrm{C}(x,x)$s with $|\, \mathrm{C}(x,x) \,| = i$.
- Hence there are at most
$$\sum_{i=0}^{\kappa} |\, K \,|^i = \frac{|\, K \,|^{\kappa+1} - 1}{|\, K \,| - 1} \leq \frac{|\, K \,|^{\kappa+1}}{|\, K \,| - 1} = \frac{2^{m+1}}{m}$$
distinct $\mathrm{C}(x,x)$s with $|\, \mathrm{C}(x,x) \,| \leq \kappa$.
- The rest must have $|\, \mathrm{C}(x,x) \,| > \kappa$.
- Because $\mathrm{C}(x,x)$ is distinct for each x (p. 42), there are at least $2^m - \frac{2^{m+1}}{m}$ of them with $|\, \mathrm{C}(x,x) \,| > \kappa$.

The Proof: Amount of Communications (concluded)

- Thus

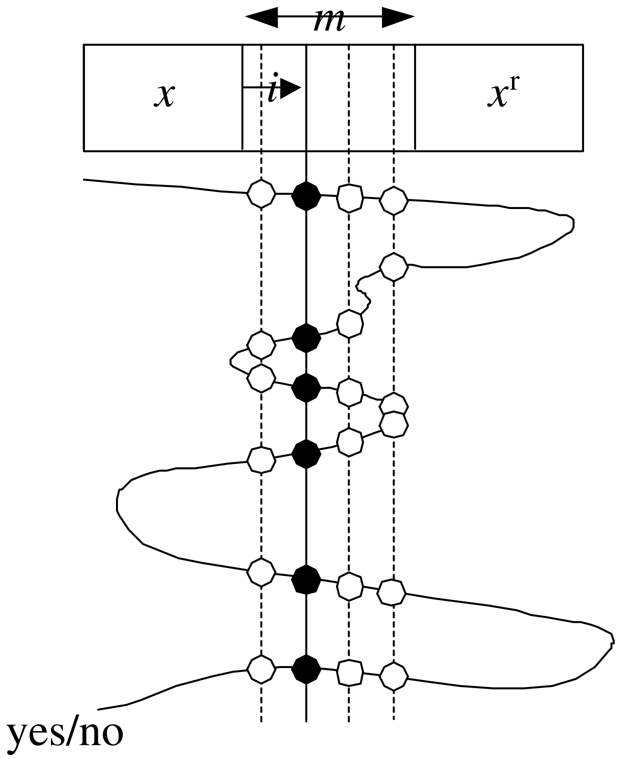
$$\begin{aligned}
\sum_{x \in \{0,1\}^m} |\, \mathrm{C}(x,x) \,| &\geq \sum_{x \in \{0,1\}^m, |\, \mathrm{C}(x,x) \,| > \kappa} |\, \mathrm{C}(x,x) \,| \\
&> \left(2^m - \frac{2^{m+1}}{m} \right) \kappa \\
&= \kappa 2^m \frac{m-2}{m}.
\end{aligned}$$

- As $\kappa = \Theta(m)$, the total number of communications is

$$\sum_{x \in \{0,1\}^m} |\, \mathrm{C}(x,x) \,| = \Omega(m 2^m). \tag{1}$$

The Proof (continued)

We now lower-bound the worst-case number of communication points in the middle section.



The Proof (continued)

- $\mathrm{C}_i(x, x)$ denotes the sequence of communications for $\mathrm{P}(x, x)$ given the cut at position i.
- Then $\sum_{i=1}^{m} |\, \mathrm{C}_i(x, x) \,|$ is the number of steps spent in the middle section for $P(x, x)$.
- Let $T(n) = \max_{x \in \{0,1\}^m} \sum_{i=1}^{m} |\, \mathrm{C}_i(x, x) \,|$.
  - $T(n)$ is the worst-case running time spent in the middle section when dealing with any $P(x, x)$ with $|\, x \,| = m$.
- Note that $T(n) \geq \sum_{i=1}^{m} |\, \mathrm{C}_i(x, x) \,|$ for any $x \in \{0, 1\}^m$.

The Proof (continued)

- Now,

$$
\begin{aligned}
& 2^m T(n) \\
= \;& \sum_{x \in \{0,1\}^m} T(n) \\
\geq \;& \sum_{x \in \{0,1\}^m} \sum_{i=1}^{m} |\, \mathrm{C}_i(x,x) \,| \\
= \;& \sum_{i=1}^{m} \sum_{x \in \{0,1\}^m} |\, \mathrm{C}_i(x,x) \,|.
\end{aligned}
$$

The Proof (concluded)

- By the pigeonhole principle,[a] there exists an $0 \leq i^* \leq m$,

$$\sum_{x \in \{0,1\}^m} |\, \mathrm{C}_{i^*}(x, x) \,| \leq \frac{2^m T(n)}{m}.$$

- Eq. (1) on p. 46 says that

$$\sum_{x \in \{0,1\}^m} |\, \mathrm{C}_{i^*}(x, x) \,| = \Omega(m 2^m).$$

- Hence

$$T(n) = \Omega(m^2) = \Omega(n^2).$$

[a]Dirichlet (1805–1859).