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.
  - We more or less follow the topics of the book.
  - Extra materials may be added.
- You may want to review discrete mathematics.

Class Information (concluded)

- More information and lecture notes can be found at

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

  - Homeworks, exams, solutions and teaching assistants will be announced there.
- Please ask many questions in class.
  - This is the best way for me to remember you in a large class.[a]

[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

- Homeworks.
  - Do not copy others' homeworks.
  - Do not give your homeworks for others to copy.
- Two to three exams.
- You must show up for the exams in person.
- If you cannot make it to an exam for a legitimate reason, please email me or a TA beforehand to the extent possible.
- Missing the final exam will automatically earn a "fail" grade.

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

Computation: What is computation?

Computability: What can be computed?

- There are *well-defined* problems that cannot be computed.
- In fact, most problems cannot be computed.

What This Course Is All About (continued)

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

- Some computable problems require at least exponential time and/or space.
  - They are said to be **intractable**.
- Some practical problems require superpolynomial[a] resources unless certain conjectures are disproved.
- Resources besides time and space: Circuit size, circuit layout area, program size, number of random bits, etc.

---

[a]The prefix "super" means "above, beyond."

What This Course Is All About (concluded)

Applications: Intractability results can be very useful.

- Cryptography and security.
- Approximations.
- Conjectures about nature.

Tractability and Intractability

- Tractability means polynomial in terms of the input size n.
  - n, $n \log n$, n^2, n^{90}.
- It results in a fruitful and practical theory of complexity.
- Few practical, tractable problems require a large degree.
- Superpolynomial-time algorithms are seldom practical.
  - $n^{\log n}$, $2^{\sqrt{n}}$,[a] 2^n, $n! \sim \sqrt{2\pi n}\,(n/e)^n$.

---

[a] Size of depth-3 circuits to compute the majority function (Wolfovitz (2006)) and certain stochastic models used in finance (Dai (`R86526008`, `D8852600`) and Lyuu (2007), Lyuu and Wang (`F95922018`) (2011), and Chiu (`R98723059`) (2012)).

Exponential Growth of *E. Coli*[a]

- Under ideal conditions, *E. Coli* bacteria divide every 20 minutes.
- In two days, a single *E. Coli* bacterium would become 2^{144} bacteria.
- They would weigh 2,664 times the Earth!

[a] Nick Lane, *Power, Sex, Suicide: Mitochondria and the Meaning of Life* (2005).

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

Moore's Law[a] to the Rescue?[b]

- Moore's law says the computing power doubles every 1.5 years.
- So the computing power grows like
$$4^{y/3},$$
where y is the number of years from now.
- Assume Moore's law holds forever.
- Can you let the law take care of exponential complexity?

[a] Moore (1965).

[b] Contributed by Ms. Amy Liu (J94922016) on May 15, 2006. Thanks also to a lively discussion on September 14, 2010.

Moore's Law to the Rescue (continued)?

- Suppose a problem takes a^n seconds of CPU time to solve now, where n is the input length.
- The same problem will take

$$\frac{a^n}{4^{y/3}}$$

seconds to solve y years from now.

- In particular, the hardware $3n \log_4 a$ years from now takes 1 second to solve it.
- The overall complexity becomes linear in n!

Moore's Law to the Rescue (concluded)?

- Potential objections:
  - Moore's law may not hold forever.
  - The total number of operations is the same; so the *algorithm* remains exponential in complexity.[a]
- What is a "good" theory on computational complexity?

---

[a]Contributed by Mr. Hung-Jr Shiu (`D00921020`) on September 14, 2011.

Turing Machines

Tarski has stressed in his lecture
(and I think justly)
the great importance of
the concept of general recursiveness
(or Turing's computability).
— Kurt Gödel (1946)

What Is Computation?

- That can be coded in an **algorithm**.[a]
- 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.

[a] Muhammad ibn Mūsā Al-Khwārizmī (780–850).

Turing Machines[a]

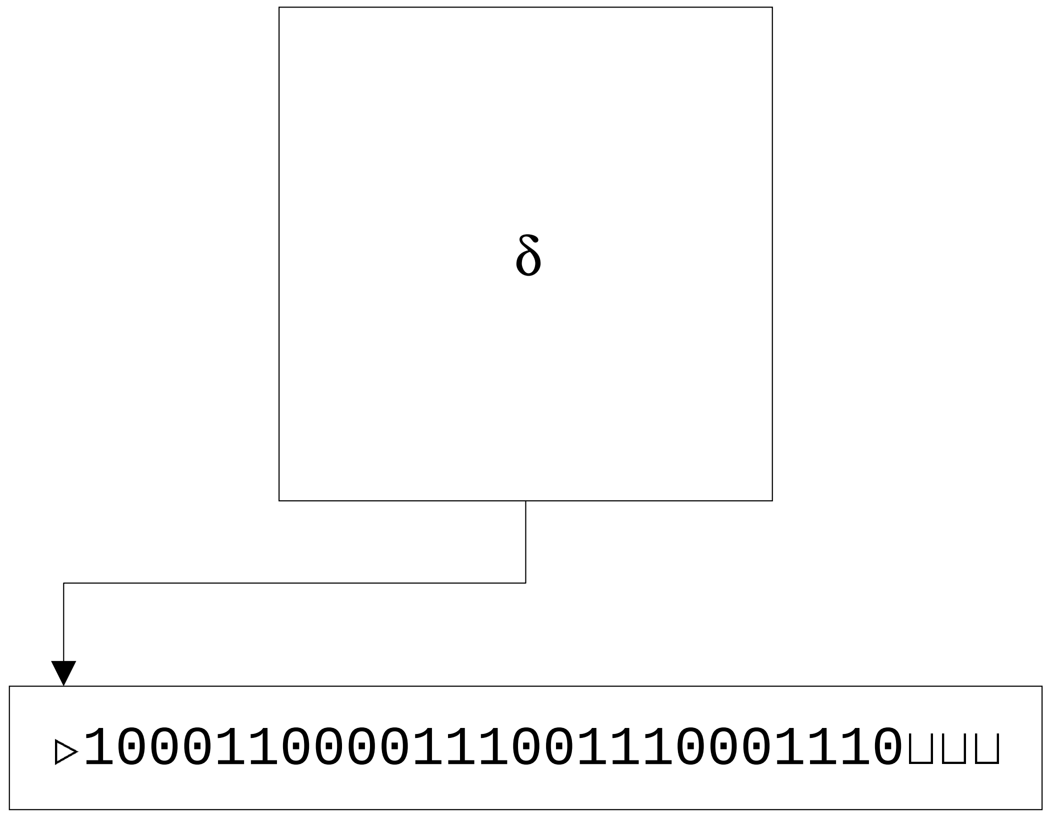
- A Turing machine (TM) is a quadruple $M = (K, \Sigma, \delta, s)$.
- K is a finite set of **states**.[b]
- $s \in K$ is the **initial state**.
- Σ is a finite set of **symbols** (disjoint from K).
  - Σ includes $\bigsqcup$ (blank) and $\triangleright$ (first symbol).
- $\delta : K \times \Sigma \to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

---

[a] Turing (1936).

[b] Turing (1936), "If we admitted an infinity of states of mind, some of them will be 'arbitrarily close' and will be confused."

A TM Schema



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;
    * The direction D the cursor will move *afterwards*.
- Assume $\delta(q, \triangleright) = (p, \triangleright, \rightarrow)$.
  - So the cursor never falls off the left end of the string.

More about δ (concluded)

- Think of the program as lines of codes:

$$\begin{aligned}
\delta(q_1, \sigma_1) &= (p_1, \rho_1, D_1), \\
\delta(q_2, \sigma_2) &= (p_2, \rho_2, D_2), \\
&\vdots \\
\delta(q_n, \sigma_n) &= (p_n, \rho_n, D_n).
\end{aligned}$$

- Assume the state is q and the symbol under the cursor σ.
- The line of code that matches (q, σ) is executed.[a]
- Then the process is repeated.

[a]So there should be one and only one instruction for every possible pair (q, σ). Contributed by Mr. Ya-Hsun Chang (`B96902025`, `R00922044`) on September 13, 2011.

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.

"Physical" Interpretations

- The tape: computer memory and registers.
  - Except that the tape can be lengthened on demand.
- δ: program.
  - A program has a *finite* size.
- K: instruction numbers.
- s: "`main()`" in the C programming language.
- Σ: **alphabet**, much like the ASCII code.

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**:** $M(x) = y$ means the string (tape) consists of a $\triangleright$, 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 $M(x) = \nearrow$.

The First TM Program[a]

- Assume $M = (K, \Sigma, \delta, s)$, where $K = \{s, h\}$, $\Sigma = \{\, 0, 1, \sqcup, \triangleright \,\}$, and

$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$
s	1	$(s, 0, \rightarrow)$
s	0	$(s, 1, \rightarrow)$
s	$\sqcup$	$(h, \sqcup, -)$

- This TM converts all 1's in the input string to 0's and vice versa.

[a] Contributed by Mr. Zheyuan (Jeffrey) Gao (`R01922142`) on September 21, 2013.

The Second TM Program[a]

- Assume $M = (K, \Sigma, \delta, s)$, where $K = \{s, s_1, h\}$, $\Sigma = \{\,0, 1, \sqcup, \triangleright\,\}$, and

[a] Contributed by Mr. Zheyuan (Jeffrey) Gao (`R01922142`) on September 21, 2013.

$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	$\rhd$	$(s, \rhd, \rightarrow)$
s	1	$(s_1, 1, \rightarrow)$
s	0	$(s, 0, \rightarrow)$
s_1	0	$(s, 0, \rightarrow)$
s_1	1	$(h, 1, -)$
s	$\sqcup$	$(h, \sqcup, -)$
s_1	$\sqcup$	$(h, \sqcup, -)$

The Second TM Program (concluded)

- This TM scans to the right until it finds two consecutive 1's and then halts.
- Otherwise, it halts at the end of the input string.

The Third TM Program

- Assume $M = (K, \Sigma, \delta, s)$, where $K = \{s, s_1, \text{“yes”}, \text{“no”}\}$, $\Sigma = \{\,0, 1, \sqcup, \triangleright\,\}$, and

$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$
s	1	$(s_1, 1, \rightarrow)$
s	0	$(s, 0, \rightarrow)$
s_1	0	$(s, 0, \rightarrow)$
s_1	1	$(\text{“yes”}, 1, -)$
s	$\sqcup$	$(\text{“no”}, \sqcup, -)$
s_1	$\sqcup$	$(\text{“no”}, \sqcup, -)$

The Third TM Program (concluded)

- This TM accepts the input if there are two consecutive 1's.
- Otherwise, it rejects the input string.

Why Turing Machines?

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

---

[a]But students are strongly encouraged to read and understand the TM codes in the textbook to gain insight on this programming language.

Remarks

- A computation model should be "physically" realizable.
  - E.g., our brain, at least as powerful as a Turing machine, is physical.
- Although a TM requires a tape of potentially infinite length, which is not realizable, it is not a major *conceptual* issue.[a]
  - Imagine you ("the program") are living next to a paper mill while carrying out a TM code using pencil ("the cursor") and paper ("the tape").
  - The mill will produce extra paper if needed.

[a]Thanks to a lively discussion on September 20, 2006.

Remarks (concluded)

- Even our computer is only an approximation of a TM for the same reason.
  - But it is easy to imagine our computer with more and more address space, memory space, and disk space.

The Concept of Configuration

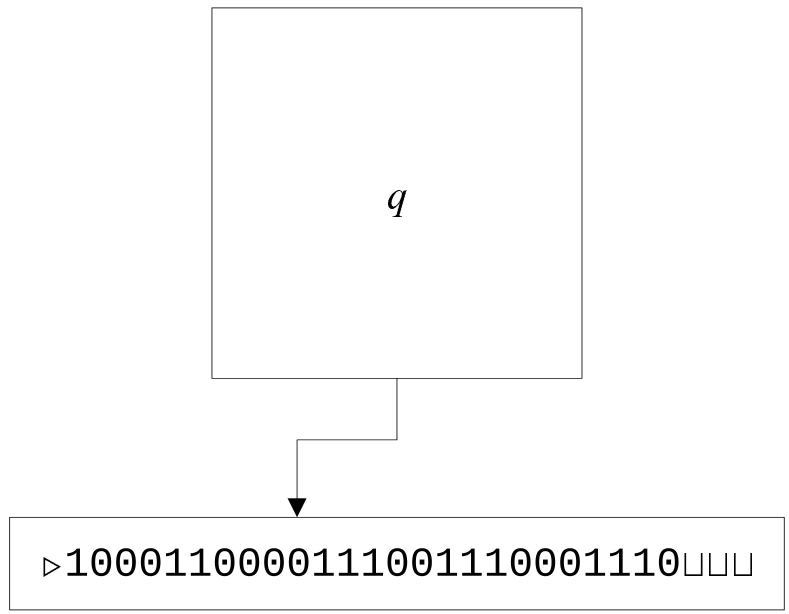
- A **configuration**[a] is a complete description of the current state of the computation.
- The specification of a configuration is sufficient for the computation to continue as if it had not been stopped.
  - What does your PC save before it sleeps?
  - Enough for it to resume work later.
- Similar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

---

[a] This term was due to Turing (1936).

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.



- $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') after $k \in \mathbb{N}$ steps.
- $(q, w, u) \xrightarrow{M^*} (q', w', u')$: Configuration (q, w, u) yields configuration (q', w', u').

Alan Turing (1912–1954)

Richard Dawkins (2006), "Turing arguably made a greater contribution to defeating the Nazis than Eisenhower or Churchill."

A TM Program To Insert a Symbol

- We want to compute $f(x) = ax$.
  - The TM moves its cursor to the last symbol.
  - It moves the last symbol of x to the right by one position.
  - It 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.

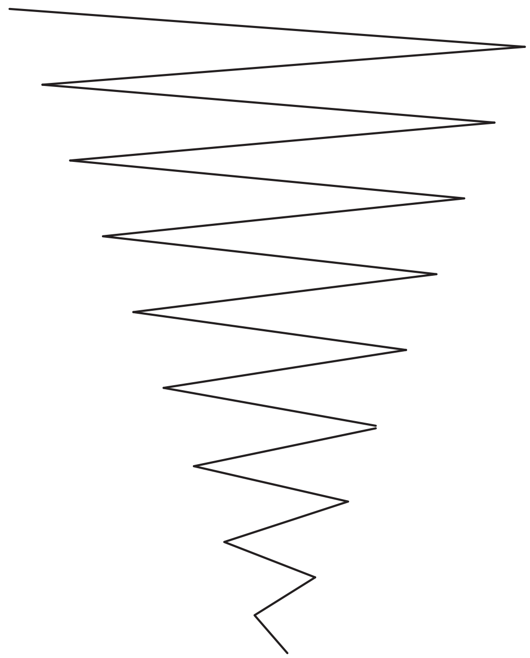
Palindromes

- 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.[a]
  - 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?

---

[a] Bryson (2001), "Possibly the most demanding form of wordplay in English[.]"

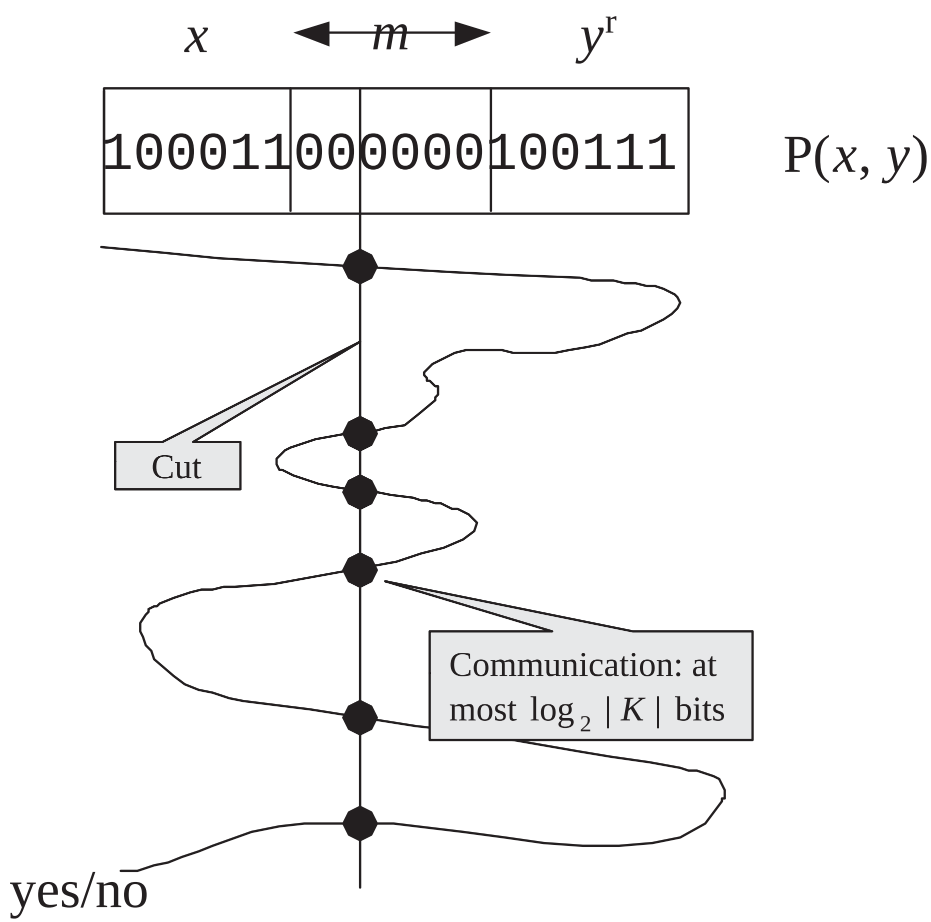
$\boxed{100011000000100111}$



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

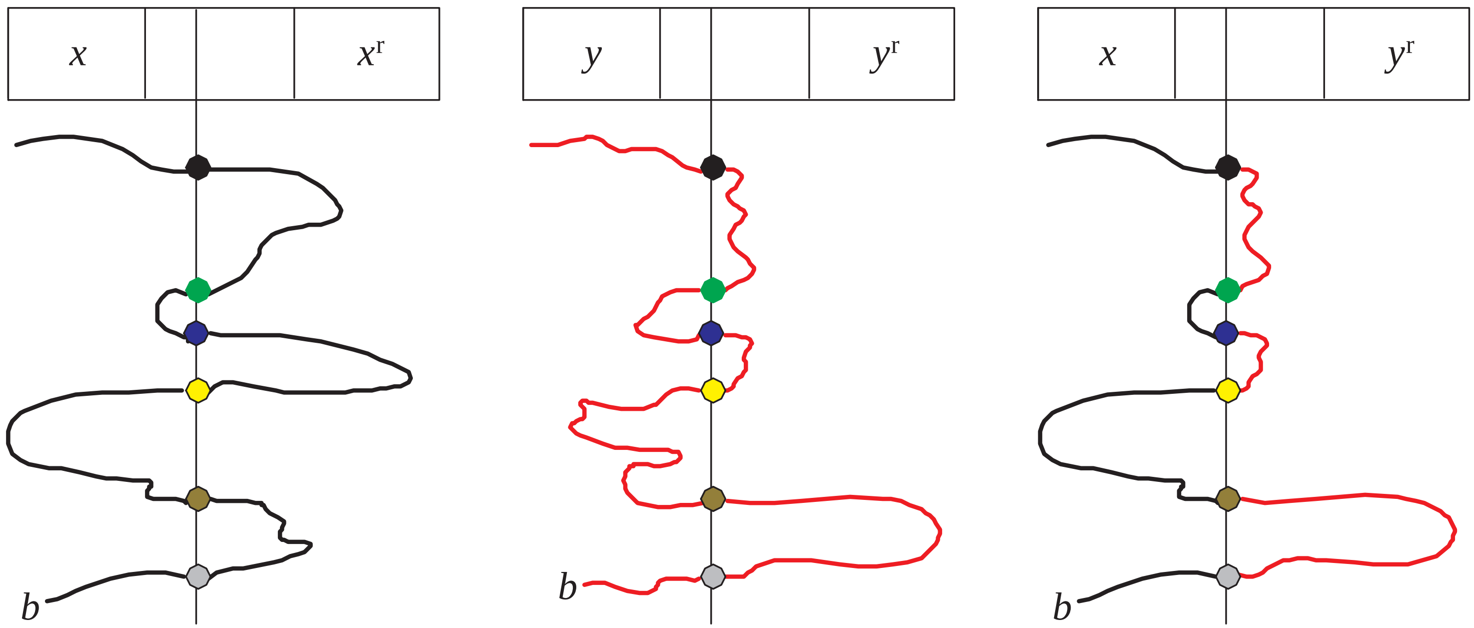


The Proof: Communications

- $\mathrm{P}(x, y) =$ "yes" if and only if $x = y$.
- 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 $O(1)$.
- $\mathrm{C}(x, y)$: the sequence of communications for palindrome problem $\mathrm{P}(x, y)$ *across* the cut.
  - This crossing sequence is a sequence of states from K.

The Proof: Communications (concluded)

- C$(x, x) \neq$ C(y, y) when $x \neq y$.
  - Suppose otherwise, $C(x, x) = C(y, y)$.
  - Then $C(x, y) = C(y, y)$ by the cut-and-paste argument (see next page).
  - Hence P(x, y) has the same answer as P(y, y)!
- So 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 for n-bit palindromes:

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

- As $\mathrm{C}(x,x)$ is distinct for each x (p. 46), there are 2^m distinct $\mathrm{C}(x,x)$s.
- 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 $|\mathrm{C}(x,x)| > \kappa$.
- So at least $2^m - \frac{2^{m+1}}{m}$ $\mathrm{C}(x,x)$s have $|\mathrm{C}(x,x)| > \kappa$.

The Proof: Amount of Communications (concluded)

- Thus

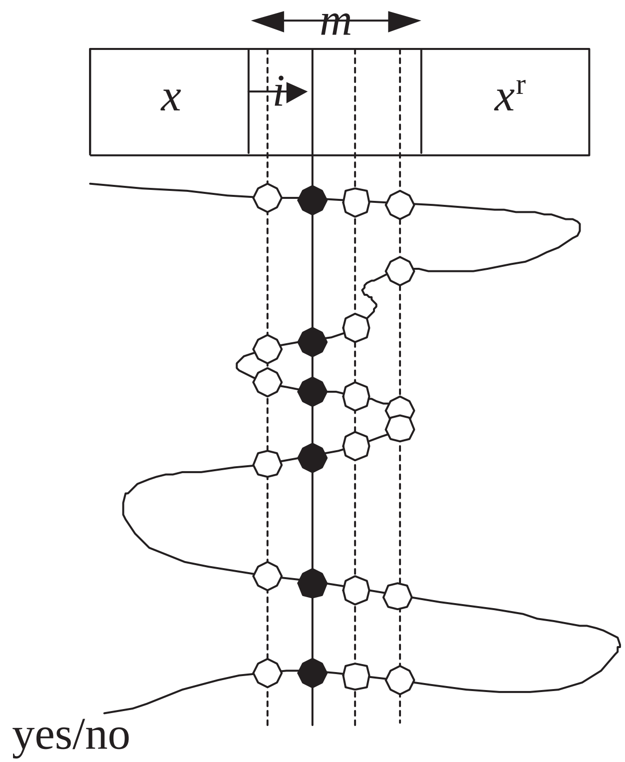
$$\begin{aligned}\sum_{x\in\{0,1\}^m} \mid \mathrm{C}(x,x) \mid &\geq \sum_{x\in\{0,1\}^m, \mid \mathrm{C}(x,x) \mid > \kappa} \mid \mathrm{C}(x,x) \mid \\ &> \left(2^m - \frac{2^{m+1}}{m}\right)\kappa \\ &= \kappa 2^m \frac{m-2}{m}.\end{aligned}$$

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

$$\sum_{x\in\{0,1\}^m} \mid \mathrm{C}(x,x) \mid = \Omega(m2^m). \tag{1}$$

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 $\mathrm{P}(x, x)$.
- 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 $\mathrm{P}(x, x)$ 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 $1 \leq i^* \leq m$,

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

- Eq. (1) on p. 50 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).

Comments on Lower-Bound Proofs

- They are usually difficult.
  - Worthy of a Ph.D. degree.
- An algorithm whose running time matches a lower bound means it is optimal.
  - The simple $O(n^2)$ algorithm for PALINDROME is optimal.
- This happens rarely and is model dependent.
  - Searching, sorting, PALINDROME, matrix-vector multiplication, etc.