

204753

Theory of computation

Lecture 1: Introduction

Course Information

- ผู้สอน
 - ครั้งแรก: อ.ดร. จักริน ชวชาติ
 - ครั้งหลัง: รศ.ดร. จีรยุทธ ไชยจารุณนิช
- คะแนนครั้งแรก
 - เข้าห้องและการบ้าน 10%
 - สอบ 40%
- หนังสือ
 - Sipser. Introduction to the Theory of Computation, 2nd Edition.
- Website
 - http://www.cs.science.cmu.ac.th/person/jakarin/doku.php?id=204753_2015

Outline ครึ่งแรก

- ทบทวนคณิตศาสตร์
- Finite Automata
- Nondeterminism
- Regular Expression
- Nonregular Language
- Push-down Automata

AUTOMATA, COMPUTABILITY, COMPLEXITY

โดยทั่วไปแล้ว theory of computation นั้นก็จะพูดถึงเรื่อง
สาขาวิชาหลักๆ 3 สาขานั้นคือ

automata

computability

complexity

ซึ่งทั้ง 3 สาขานี้สัมพันธ์กันด้วยคำถามที่ว่า

**What are the fundamental capabilities and limitations of
computers?**

Complexity Theory

ในแต่ละสาขานั้น ก็จะตีความคำถามที่ว่านี้ในมุมมองที่แตกต่างกัน

Complexity theory:

ปัญหาทางคอมพิวเตอร์นั้นอยู่ในหลาย ๆ รูปแบบ บางปัญหาง่าย บางปัญหายาก ตัวอย่างเช่น ปัญหาการเรียงข้อมูล นั่นคือเราต้องการจัดลำดับของข้อมูลจากน้อยไปมาก ปัญหานี้ถือว่าเป็นปัญหาง่าย แม้แต่เครื่องคอมพิวเตอร์ขนาดเล็กก็สามารถเรียงลำดับข้อมูลเป็นล้านตัวได้อย่างรวดเร็ว

เมื่อเปรียบเทียบกับปัญหาการจัดตารางเวลาเรียน (Scheduling) ซึ่งคือเราต้องการหาตารางเวลาของวิชาเรียนที่สอดคล้องกับเงื่อนไขบางอย่าง เช่น ห้ามมีสองรายวิชาที่ใช้ห้องเรียนเดียวกัน เวลาเดียวกัน ปัญหาการจัดตารางเรียนดูยากกว่าปัญหาการเรียงข้อมูล

Complexity Theory

**What makes some problems computationally hard
and others easy?**

นี่เป็นคำถามหลักของ complexity theory ซึ่งคำถามนี้เราไม่รู้คำตอบมาเป็นเวลานาน จนไม่กี่สิบปีมานี้ ถือว่าเป็นหนึ่งในความสำเร็จของสาขานี้ คือ นักวิจัยได้ค้นพบแนวทางในการแบ่งกลุ่มปัญหาตามความยากในการคำนวณ มันจะคล้ายๆ กับตารางธาตุทางเคมี

Complexity Theory

เรามีหลายทางเลือกเมื่อเจอกับปัญหาที่มองแล้วคำนวณยาก

อย่างแรก ทำความเข้าใจจุดที่ยากของปัญหาคือจุดไหน ซึ่งเราอาจจะปรับเปลี่ยนมันเพื่อให้ปัญหานั้นแก้ได้ง่ายขึ้น

อย่างที่สอง เราอาจจะยอมรับคำตอบที่ไม่ใช่คำตอบที่ดีที่สุดของปัญหา ในกรณีนี้การหาคำตอบที่เป็นการประมาณคำตอบที่ดีที่สุดจะง่ายกว่า

อย่างที่สาม บางปัญหายากในกรณีที่แย่ที่สุดเพียงกรณีเดียว แต่ส่วนใหญ่แก้ได้ง่าย ซึ่งขึ้นกับการนำไปใช้เราอาจจะพอใจกับกระบวนการแก้ปัญหาที่ทำงานช้าในบางครั้งแต่ส่วนใหญ่ทำงานเร็ว

Complexity Theory

หนึ่งในสาขาประยุกต์ที่มีผลโดยตรงกับ Complexity theory คือ Cryptography (วิทยาการเข้ารหัสลับ)

โดยทั่วไปแล้วปัญหาทางการคำนวณที่ง่ายจะเป็นที่ต้องการมากกว่าปัญหายาก แต่ใน Cryptography นั้นต้องการปัญหายาก เพราะว่ารหัสลับต่างๆ ควรจะยากในการแกะรหัสถ้าไม่มี secret key หรือ password

Complexity theory นั้นเป็นแนวทางให้กับนักเข้ารหัสในการสร้างรหัสใหม่

Computability Theory

ช่วงกลางของศตวรรษที่ 20 Kurt Gödel, Alan Turing, Alonzo Church ได้ค้นพบปัญหาพื้นฐานที่ไม่สามารถแก้ได้ด้วยคอมพิวเตอร์ ตัวอย่างหนึ่งของปัญหานี้คือปัญหาของการตัดสินใจว่า ประโยคทางคณิตศาสตร์เป็นจริงหรือเท็จ ซึ่งดูเหมือนว่าน่าจะใช้คอมพิวเตอร์ตอบได้ แต่ computer ตอบไม่ได้

ท่ามกลางการค้นพบนี้ทำให้เกิดแนวคิดในการพัฒนาโมเดลทางทฤษฎีของคอมพิวเตอร์ซึ่งนำไปสู่การสร้างคอมพิวเตอร์จริงๆ

ใน computability theory นั้นจะเป็นการแบ่งแยกปัญหาว่าปัญหาใดแก้ได้ ปัญหาใดแก้ไม่ได้

Automata Theory

Automata theory เกี่ยวข้องกับนิยามและคุณสมบัติของโมเดลทางคณิตศาสตร์
สำหรับการคำนวณ โมเดลเหล่านี้มีบทบาทในหลายๆ สาขาของ
Computer Science

ตัวอย่าง โมเดลที่เราจะได้เรียน Finite automata ถูกใช้ใน Text
processing, compilers, hardware design

อีกโมเดลคือ context free grammar ถูกใช้ใน programming languages
และ AI

เราจะเริ่มเรียน Automata Theory ก่อน

AUTOMATA, COMPUTABILITY, COMPLEXITY

สรุป

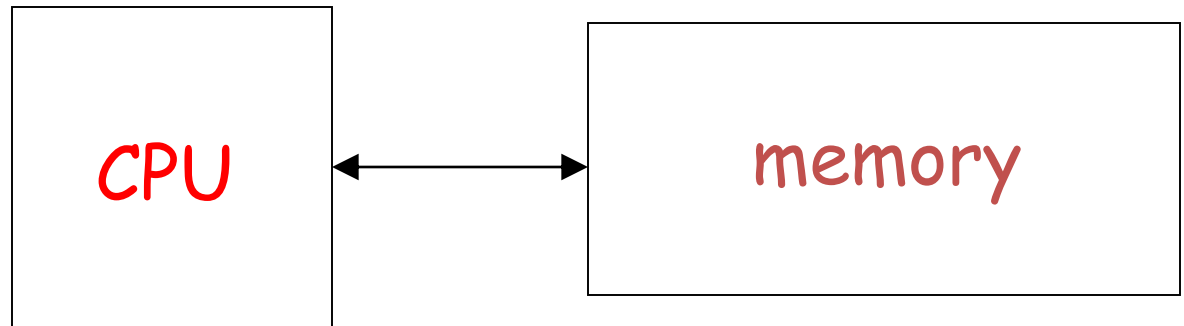
Complexity theory: Classify problems as easy ones and hard ones.

Computability theory: Classify problems as solvable ones and unsolvable ones.

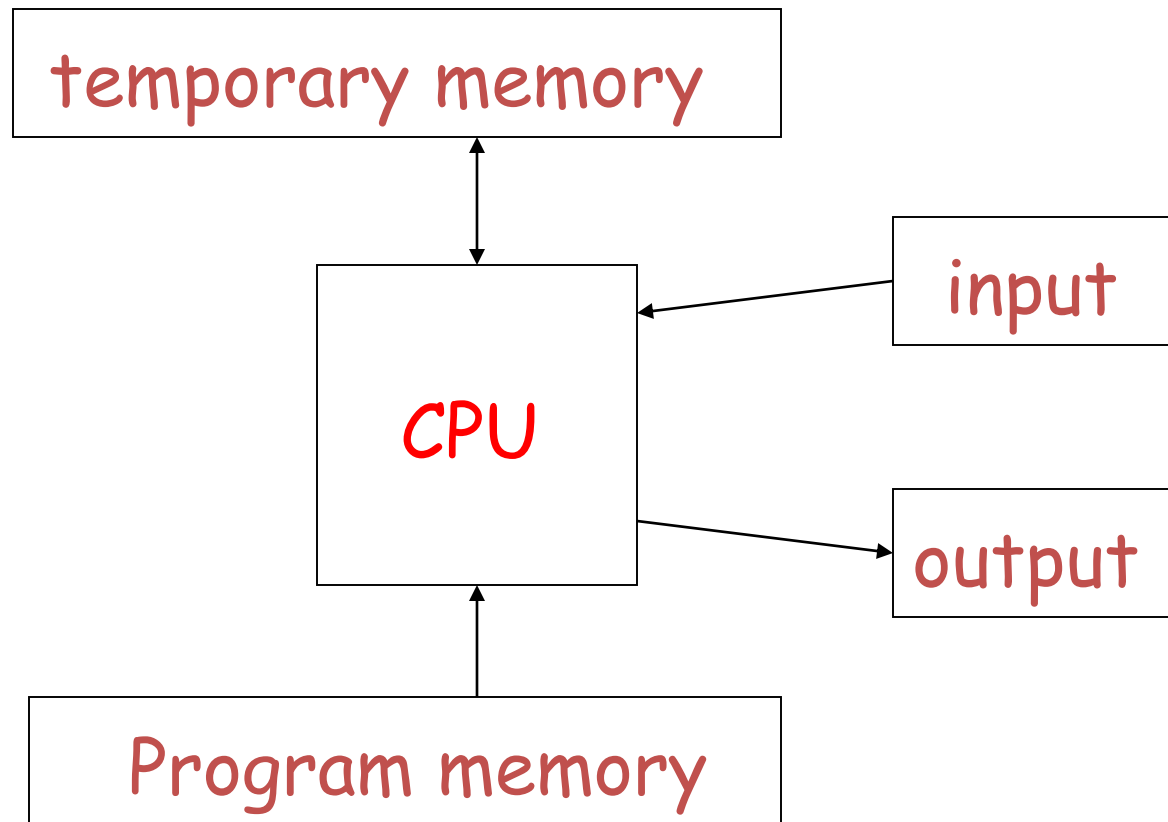
Automata theory: Deal with the definitions and properties of mathematical models of computation. Finite automaton, context-free grammar

Outline of the course contents

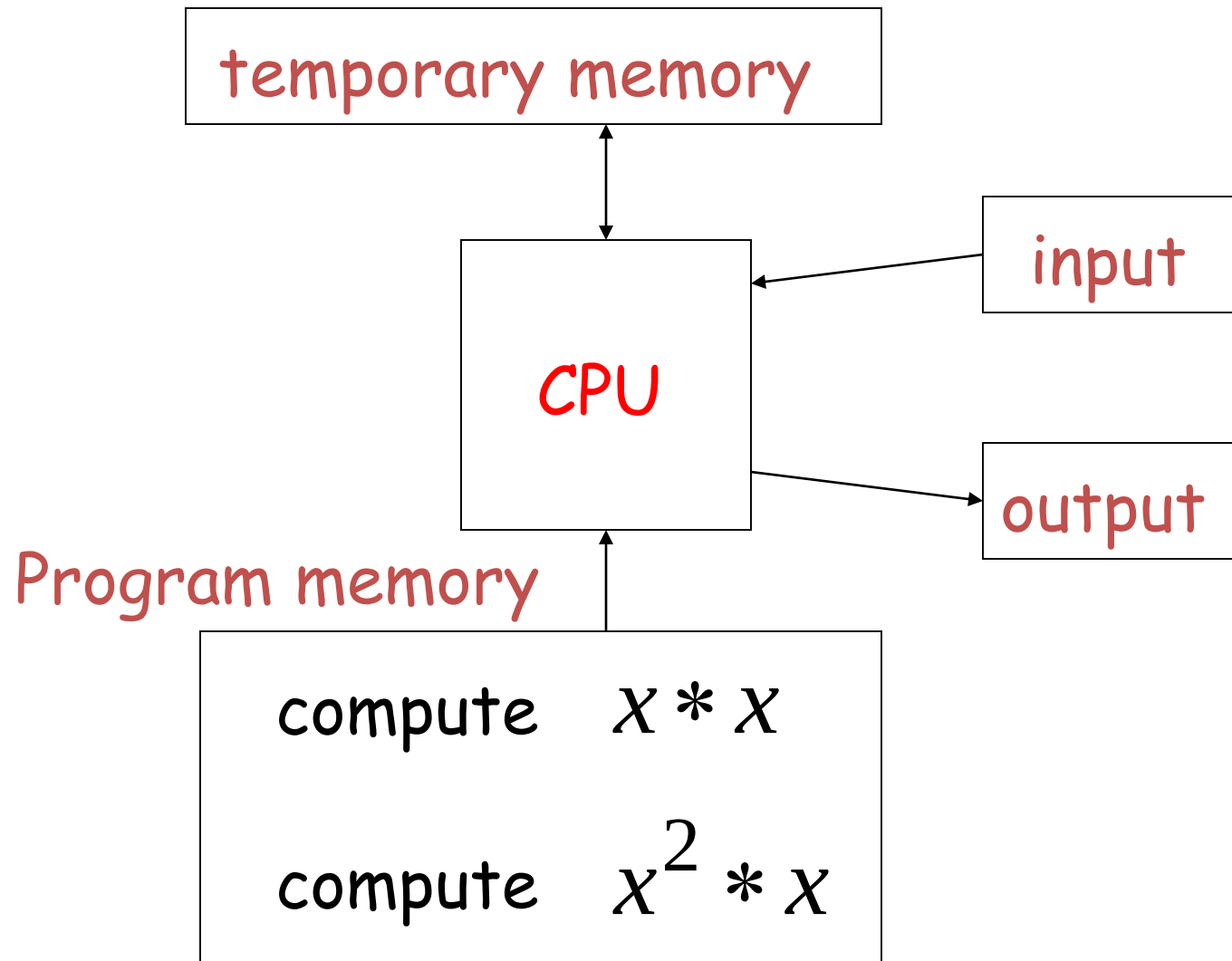
Computation



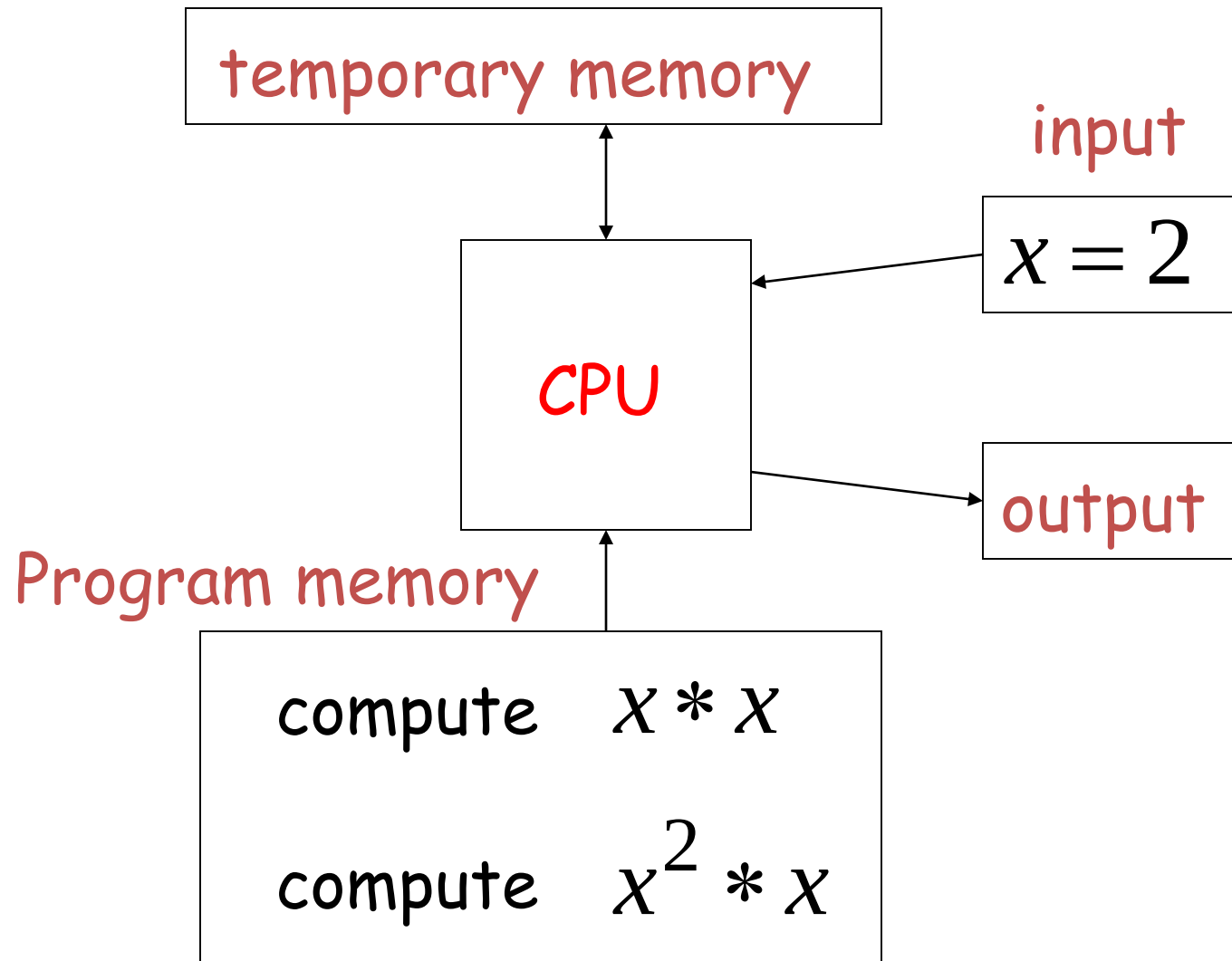
จาก www.cs.rpi.edu/~moorthy/Courses/modcomp/slides/Introduction.ppt



Example: $f(x) = x^3$



$$f(x) = x^3$$



$$f(x) = x^3$$

temporary memory

$$z = 2 * 2 = 4$$
$$f(x) = z * 2 = 8$$

input

$$x = 2$$

CPU

output

Program memory

compute $x * x$

compute $x^2 * x$

$$f(x) = x^3$$

temporary memory

$$z = 2 * 2 = 4$$
$$f(x) = z * 2 = 8$$

input

$$x = 2$$

CPU

$$f(x) = 8$$

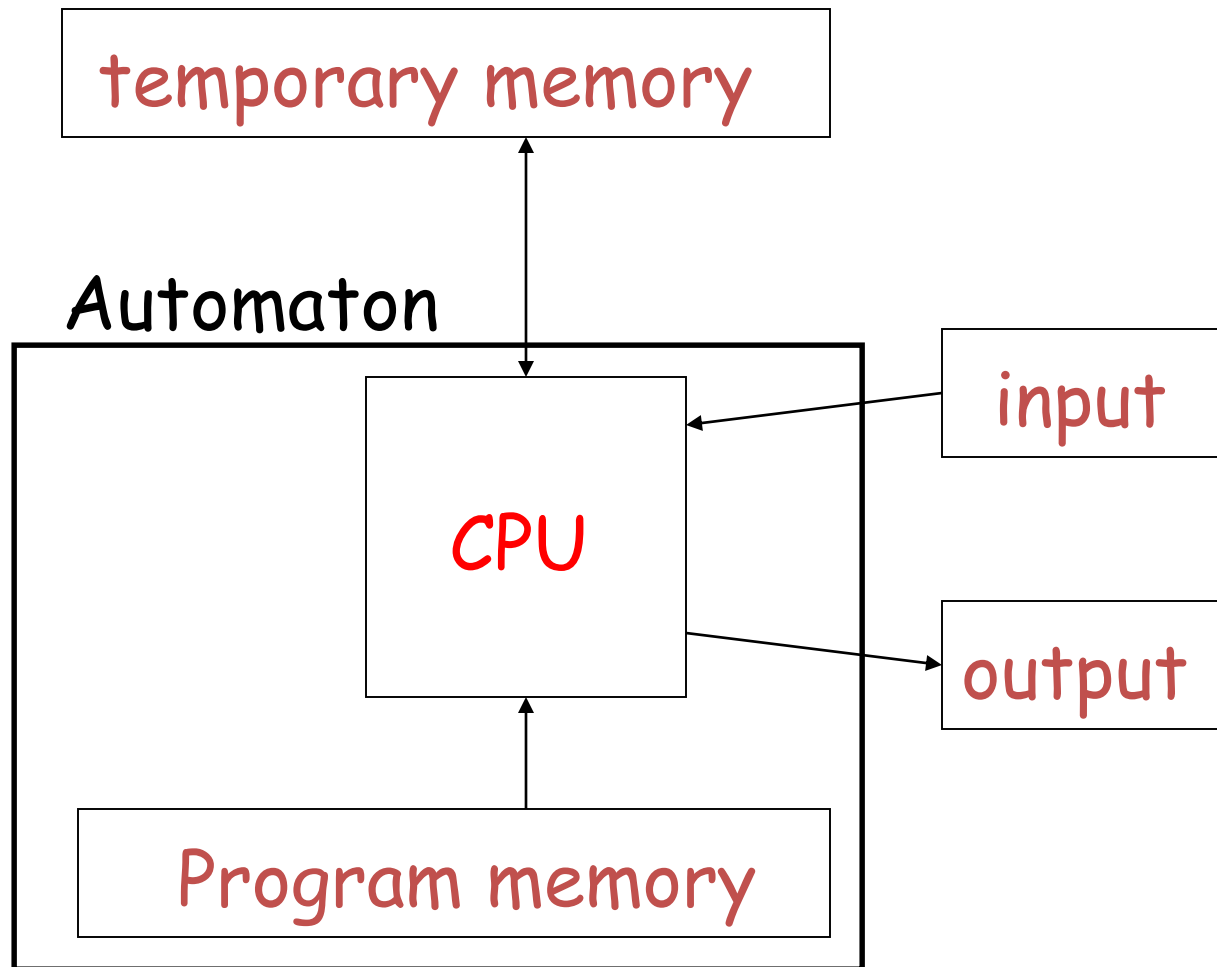
output

Program memory

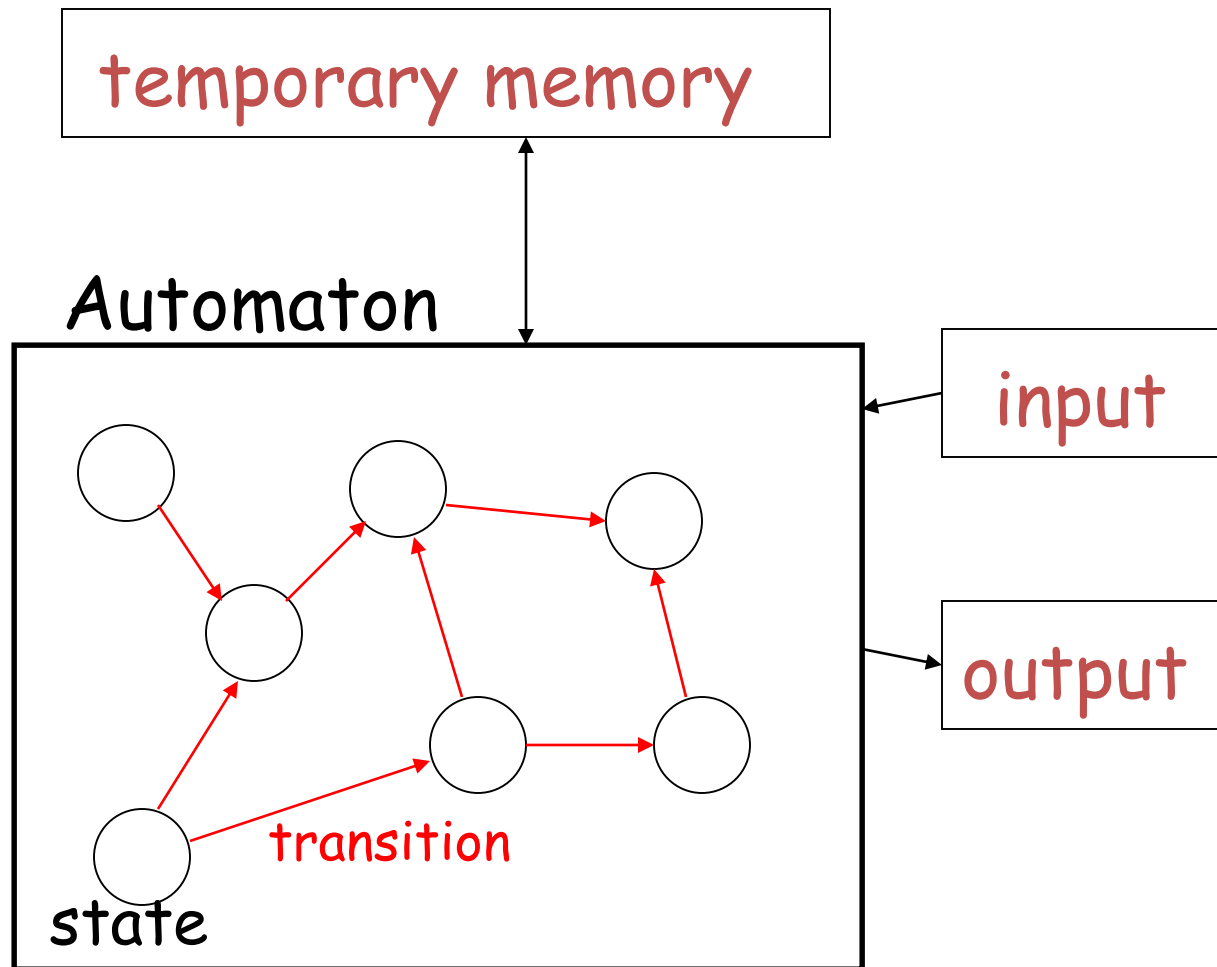
compute $x * x$

compute $x^2 * x$

Automaton



Automaton

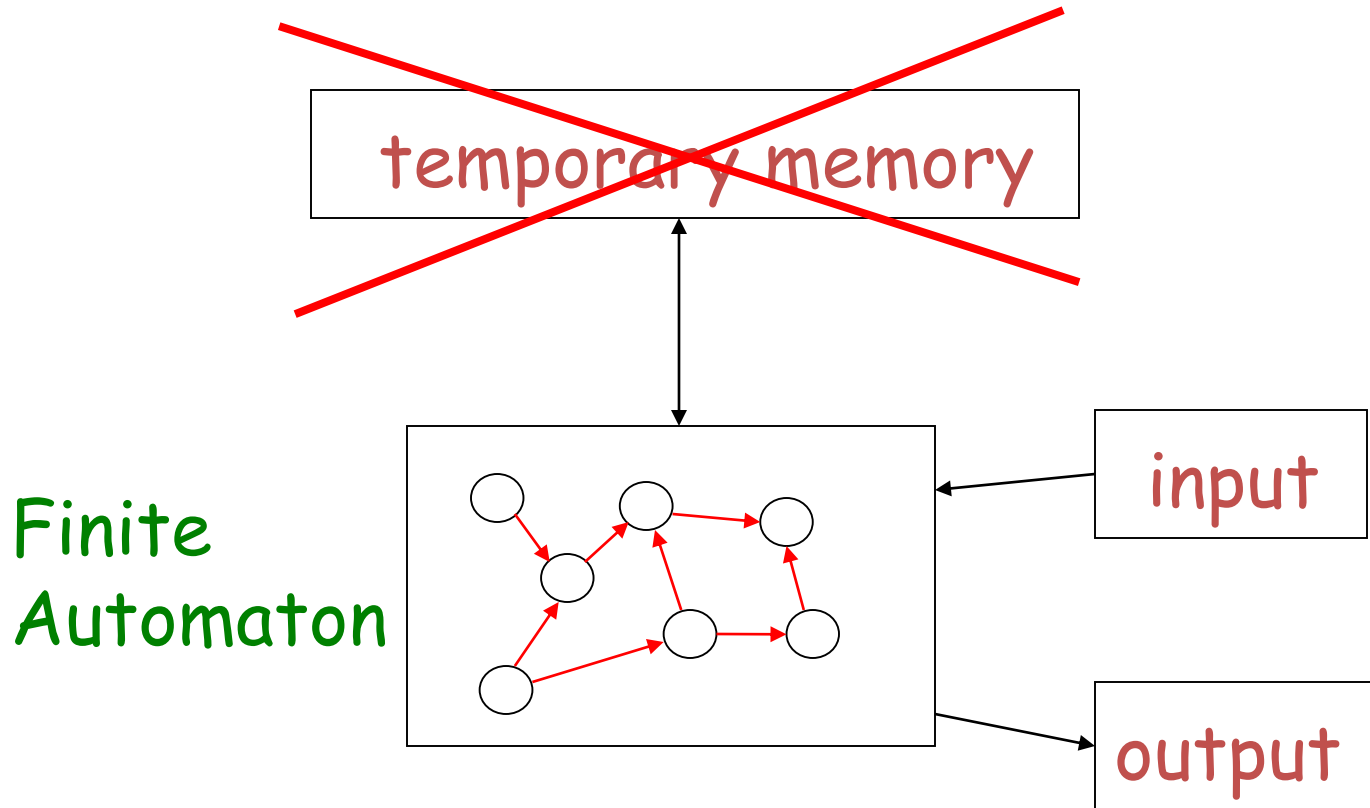


Different Kinds of Automata

Automata are distinguished by the temporary memory

- **Finite Automata:** no temporary memory
- **Pushdown Automata:** stack
- **Turing Machines:** random access memory

Finite Automaton



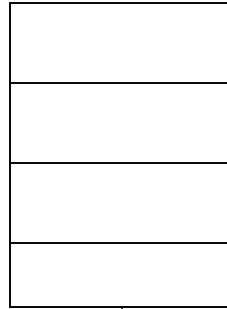
Example: Elevators, Vending Machines
(small computing power)

Pushdown Automaton

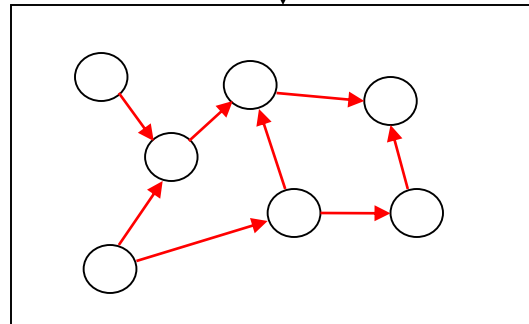
Temp.
memory

Stack

Push, Pop



Pushdown
Automaton



input

output

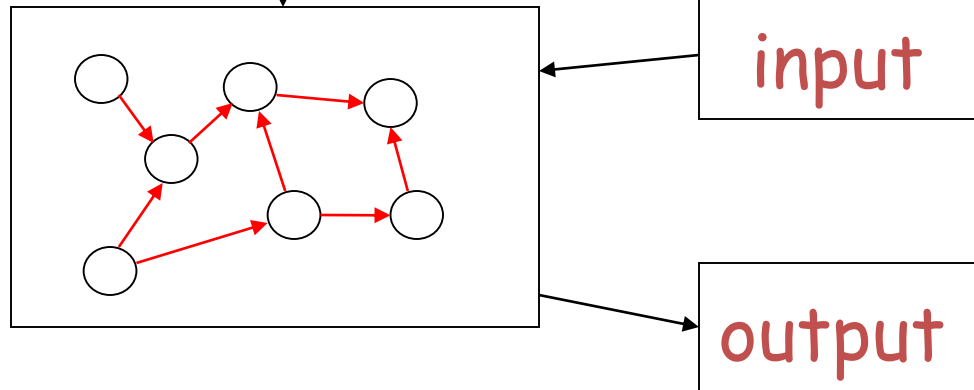
Example: Compilers for Programming Languages
(medium computing power)

Turing Machine

Temp.
memory

Random Access Memory

Turing
Machine



Examples: Any Algorithm

(highest computing power)

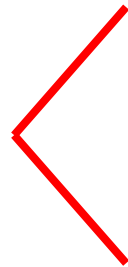
Power of Automata

Simple
problems

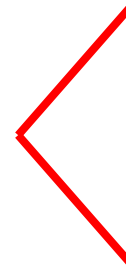
More complex
problems

Hardest
problems

Finite
Automata



Pushdown
Automata



Turing
Machine

Less power



More power

Solve more

computational problems

Turing Machine is the most powerful computational model known

Question: Are there computational problems that a Turing Machine cannot solve?

Answer: Yes (unsolvable problems)

Time Complexity of Computational Problems:

NP-complete problems

Believed to take exponential
time to be solved

P problems

Solved in polynomial time

Mathematical Preliminaries

- Sets
- Functions
- Relations
- Graphs
- Proof Techniques

เซต (Sets)

กลุ่มของสิ่งต่างๆ จะถูกเรียกว่า **เซต** (Set) เช่นเซตของเดือนทั้งสิบสอง, เซตของวันทั้งเจ็ด ฯลฯ

เซตสามารถเก็บวัตถุชนิดอะไรก็ได้รวมทั้งตัวเลข, สัญลักษณ์, หรือแม้กระทั่งเซตอื่น

วัตถุที่อยู่ภายในเซต เรียกว่า **สมาชิก** (Element หรือ member) โดยทั่วไปนิยมตั้งชื่อเซตด้วยตัวอักษรใหญ่และเขียนสัญลักษณ์แทนเซตด้วยปีกกา { } ตัวอย่างเช่น

$$A = \{ 1, 2, 3, \dots \}$$

$$B = \{ \textit{football}, \textit{basketball}, \textit{running}, \textit{swimming} \}$$

การเขียนเซต

ในการแจกแจงสมาชิกของเซตนั้น เราจะใช้เครื่องหมายจุลภาค (,) คั่นระหว่างสมาชิกแต่ละตัว หากมีจำนวนมากจะใช้ ... ในการละสมาชิกบางตัวไว้ในฐานที่เข้าใจ

- การเขียนเซตแบบแจกแจงสมาชิก เช่น $A = \{28, 29, 30, 31\}$
- การเขียนเซตแบบบอกเงื่อนไข เป็นการเขียนในรูป {สมาชิก | เงื่อนไขในการเป็นสมาชิก} อ่านว่า "เซตของ (สมาชิก) โดยที่ (เงื่อนไข)" เช่น $S = \{j \mid j > 0, \text{ and } j = 2k \text{ for some } k > 0\}$, $T = \{x \mid x \text{ เป็นจำนวนเฉพาะที่มากกว่า } 10\}$

เราจะเขียน $x \in A$ แทน "x เป็นสมาชิกของเซต A"

เราจะเขียน $x \notin A$ แทน "x ไม่ เป็นสมาชิกของเซต A"

เซตจำกัดและเซตอนันต์

เซตจำกัด (Finite set) คือเซตที่มีจำนวนสมาชิกเท่ากับจำนวนเต็มบวกหรือศูนย์ (บอกจำนวนสมาชิกได้) เช่น $A = \{4, -10, 55, 6\}$
จำนวนสมาชิกของ A เขียนแทนด้วย $n(A) = 4$

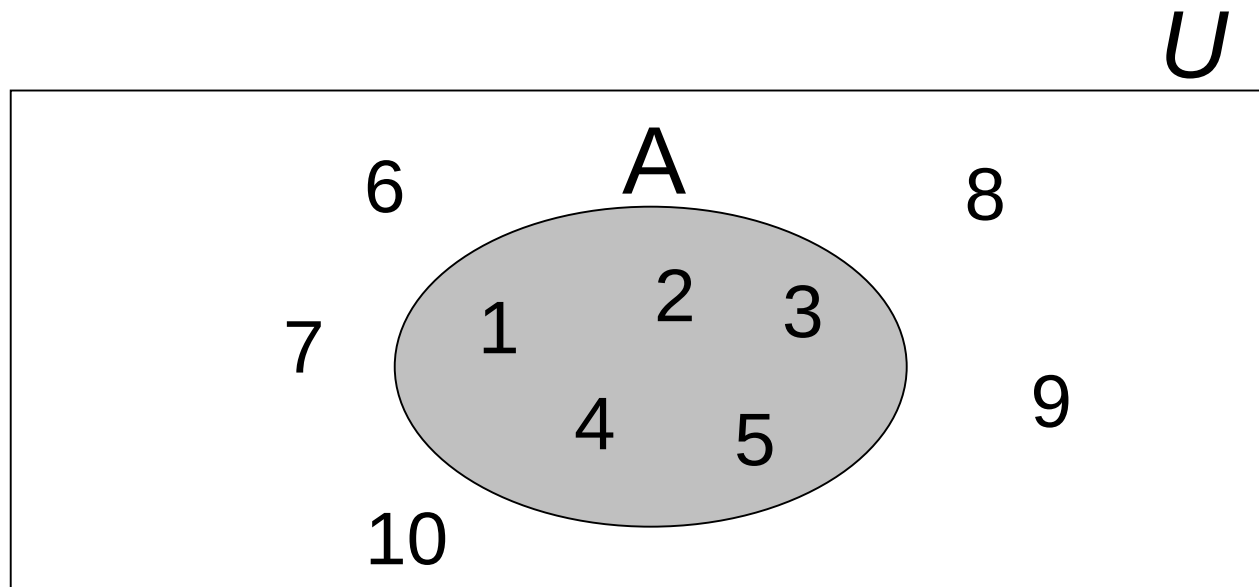
เซตอนันต์ (Infinite set) คือเซตที่ไม่ใช่เซตจำกัด เช่น
 $B = \{2, 4, 6, 8, 10, \dots\}$

คำถาม $C = \{1, 2, 3, 4, \dots, 10\}$ เป็นเซตจำกัดหรือไม่

เซตว่างและเอกภาพสัมพัทธ์

เซตว่าง คือเซตที่ไม่มีสมาชิก เขียนแทนด้วย ϕ หรือ $\{\}$

เอกภาพสัมพัทธ์ คือเซตที่กำหนดขึ้นโดยมีข้อตกลงว่าจะไม่กล่าวถึงสิ่งใดนอกเหนือไปจากสมาชิกของเซตที่กำหนดขึ้น (ขอบเขตที่เราสนใจ) เขียนแทนด้วย U



$$A = \{ 1, 2, 3, 4, 5 \}$$

$$U = \{ 1, \dots, 10 \}$$

ข้อสังเกต

การเขียนสมาชิกภายในเซต จะไม่มีลำดับก่อนหลัง สนใจแค่ว่า "อยู่" หรือ "ไม่อยู่" ในเซตเท่านั้น ดังนั้น สมาชิกที่ปรากฏซ้ำจะนับเป็นตัวเดียวกัน (ถ้าสนใจตัวซ้ำจะเรียก multiset)

หมายเหตุ ภายในเซตอาจจะมีคู่อันดับ หรือมีเซตอยู่อีกชั้น หรือมีสมาชิกเป็นอย่างไรก็ได้ โดยหลักการนับจำนวนสมาชิกจะให้ 1 คู่อันดับ หรือ 1 เซตเป็นสมาชิก 1 ตัวเช่น

$E = \{5, (6,7), \{8,9,10\}, 11, (12,13)\}$ จะได้ว่า $n(E) = 5$

ความสัมพันธ์ระหว่างเซต

สับเซต

A เป็นสับเซตของ B ถ้าสมาชิกทุกตัวของ A เป็นสมาชิกของ B และเขียนแทนด้วย $A \subseteq B$

เราบอกว่า A เป็น proper subset ของ B เขียนแทนด้วย $A \subset B$ ถ้า A เป็น subset ของ B แต่ไม่เท่ากับ B

การเท่ากันของเซต

$A = B$ ก็ต่อเมื่อ

- 1.สมาชิกทุกตัวของ A เป็นสมาชิกของ B และ
- 2.สมาชิกทุกตัวของ B เป็นสมาชิกของ A

นั่นคือ $A = B$ ก็ต่อเมื่อ $A \subset B$ และ $B \subset A$

ข้อสังเกต

สำหรับเซต A ใดๆ

- $\phi \subset A$
- $A \subset A$
- $A \subset U$
- เรียกสับเซตของ A ที่ไม่เท่ากับ A ว่า สับเซตแท้ของ A
- ถ้า A เป็นเซตจำกัด และ $n(A) = m$ แล้ว A จะมีสับเซตแตกต่างกันทั้งหมด 2^m สับเซต

เพาเวอร์เซต

เพาเวอร์เซต คือ เซตที่บรรจุด้วยสับเซตทั้งหมดที่เป็นไปได้ เพาเวอร์เซตของ A เขียนแทนด้วยสัญลักษณ์ $P(A)$ นิยามโดย $P(A) = \{B \mid B \subset A\}$

ตัวอย่าง กำหนดให้ $A = \{1, 2, 3\}$ จงหา $P(A)$

จะได้ว่า $P(A) = \{\phi, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$

ข้อสังเกต สำหรับเซต A ใดๆ

- $P(A)$ คือเซตที่ประกอบไปด้วยสับเซตทั้งหมดของ A
- ถ้า A เป็นเซตจำกัดแล้ว $n(P(A)) = 2^{n(A)}$
- $\phi \in P(A)$ และ $A \in P(A)$
- ถ้า $A \subset B$ แล้ว $P(A) \subset P(B)$
- $P(A) \cup P(B) \subset P(A \cup B)$
- $P(A) \cap P(B) = P(A \cap B)$

การดำเนินการของเซต

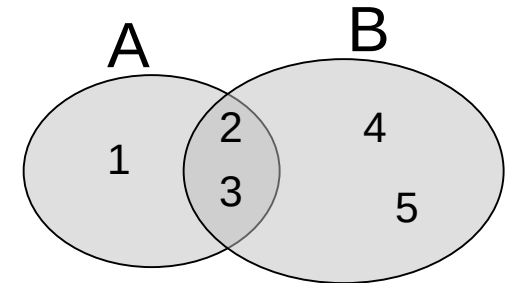
$$U = \{1, 2, 3, 4, 5, 6, 7\}$$

$$A = \{1, 2, 3\}$$

$$B = \{2, 3, 4, 5\}$$

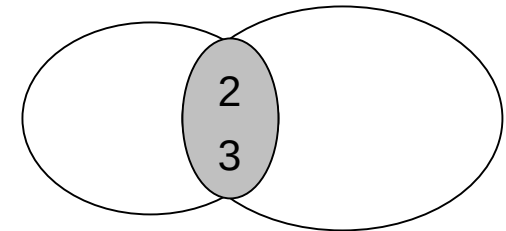
- Union:

- $A \cup B = \{x | x \in A \text{ or } x \in B\}$



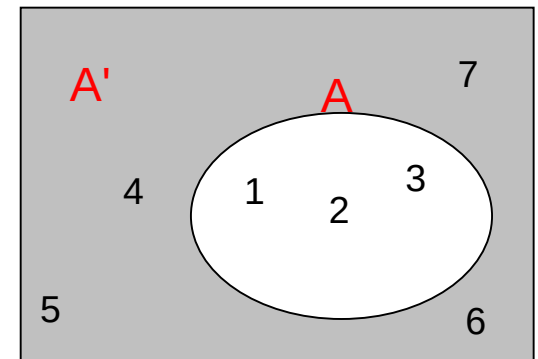
- Intersection:

- $A \cap B = \{x | x \in A \text{ and } x \in B\}$



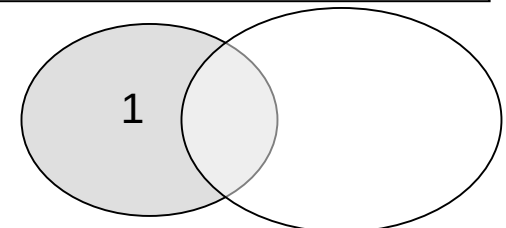
- Complement:

- $A' = \{x | x \notin A\}$



- Difference:

- $A - B = \{x | x \in A \text{ and } x \notin B\}$



สำหรับเซต A, B และ C ใดๆ

$A \cup A = A$	$A \cap A = A$
$A \cup B = B \cup A$	$A \cap B = B \cap A$
$(A \cup B) \cup C = A \cup (B \cup C)$	$(A \cap B) \cap C = A \cap (B \cap C)$
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
$(A \cup B)' = A' \cap B'$	$(A \cap B)' = A' \cup B'$
$(A')' = A$	
$\emptyset' = U$	$U' = \emptyset$
$A \cup \emptyset = A$ $A - \emptyset = A$	$A \cap \emptyset = \emptyset$ $\emptyset - A = \emptyset$
$U \cup A = U$ $U - A = A'$	$U \cap A = A$ $A - U = \emptyset$
$A \cup A' = U$	$A \cap A' = \emptyset$
$A - B = A \cap B'$	
ถ้า $A \subset B$ แล้ว $B' \subset A'$	

แบบฝึกหัด

ในห้องเรียนห้องหนึ่งมีนักเรียน 73 คน ว่ายนํ้าเป็น 45 คน ขับรถเป็น 26 คน ทำไม่เป็นทั้งสองอย่าง 18 คน ถามว่า

1. ว่ายนํ้าเป็นและขับรถเป็น ก็คน
2. ว่ายนํ้าเป็นหรือขับรถเป็น ก็คน

Sequences and Tuples

ลำดับ(Sequence) ของวัตถุคือรายการของวัตถุที่มีลำดับก่อนหลัง เราจะเขียนภายในวงเล็บ ตัวอย่างเช่นลำดับ 7,21,57 เขียนเป็น

(7,21,57)

ในเซตลำดับก่อนหลังไม่สำคัญแต่ในลำดับนั้นสำคัญ ดังนั้น (7,21,57) ไม่เท่ากับ (57,7,21) ตัวที่ซ้ำกันก็มีความสำคัญเช่นกัน

ลำดับอาจจะมีจำนวนจำกัดหรือไม่ก็ได้ ลำดับจำกัดมักจะเรียกว่า **tuple** โดยลำดับที่มี k ตัวจะเรียกว่า **k-tuple** เช่น (7,21,57) คือ 3-tuple ส่วน 2-tuple เรียกว่า **คู่อันดับ**(pair)

ความสัมพันธ์และฟังก์ชัน

คู่อันดับ

ประกอบด้วยสมาชิกสองตัวในรูป (a,b) ซึ่งไม่สามารถเปลี่ยนลำดับสมาชิกตัวหน้ากับตัวหลังได้ และ $(a,b) = (c,d)$ ก็ต่อเมื่อ $a = c$ และ $b = d$ เท่านั้น

ผลคูณคาร์ทีเซียน

กำหนดให้ A และ B เป็นเซตใดๆ ผลคูณคาร์ทีเซียน (Cartesian product) ของ A และ B บางครั้งเรียก Cross product เขียนแทนด้วย $A \times B$ คือเซตของคู่อันดับที่สมาชิกตัวแรกมาจากเซต A และสมาชิกตัวที่สองมาจากเซต B ครบทุกคู่

$$A \times B = \{(a, b) | a \in A \text{ and } b \in B\}$$

ข้อสังเกต โดยทั่วไป $A \times B \neq B \times A$ แต่ $n(A \times B) = n(B \times A) = n(A)n(B)$

Cartesian Product

$$A = \{ 2, 4 \}$$

$$B = \{ 2, 3, 5 \}$$

$$A \times B = \{ (2, 2), (2, 3), (2, 5), (4, 2), (4, 3), (4, 5) \}$$

$$|A \times B| = |A| |B|$$

Generalizes to more than two sets

$$A \times B \times \dots \times Z$$

ถ้า $A = \{1,2\}$ และ $B = \{x,y,z\}$

$$A \times B = \{(1,x),(1,y),(1,z),(2,x),(2,y),(2,z)\}$$

เราสามารถเขียน Cartesian product ของ k เซตด้วย $A_1 \times A_2 \times \dots \times A_k$ โดยมันจะเป็นเซตที่ประกอบไปด้วย ทุก k -tuple $(a_1, a_2, \dots, a_k)$ ที่ $a_i \in A_i$

ถ้า $A \times B \times A$ จะเป็นอย่างไร

ฟังก์ชัน(Function)

ฟังก์ชัน เป็นวัตถุที่สร้างมาจากความสัมพันธ์ของ input-output

นั่นคือฟังก์ชันจะนำเอา input เข้าไปแล้วสร้าง output ออกมา

ในทุกๆ ฟังก์ชันจะมีคุณสมบัติอย่างหนึ่งคือ input ตัวเดิมจะสร้าง output ตัวเดิมเสมอ

ถ้า f เป็นฟังก์ชันที่ output เป็น b เมื่อ input เป็น a เขียนได้เป็น

$$f(a) = b$$

ฟังก์ชันบางครั้งจะเรียกว่า **mapping** และถ้า $f(a) = b$ เราจะเรียกว่า f map จาก a ไป b

ตัวอย่าง ฟังก์ชัน absolute abs ที่รับตัวเลข x เป็น input และคืนค่า x ถ้า x เป็นค่าบวกและคืนค่า $-x$ เมื่อ x เป็นค่าลบ ดังนั้น

$$\text{abs}(2) = \text{abs}(-2) = 2$$

การบวกเป็นอีกตัวอย่างของฟังก์ชัน โดยรับข้อมูลเป็นคู่ของจำนวนและ output เป็นผลรวมของจำนวน

Domain and Range

เซตของ input ที่เป็นไปได้ของฟังก์ชันจะเรียกว่า **Domain**

Output ของฟังก์ชันที่มาจาก set ด้านบนจะถูกเรียกว่า **Range**

สัญลักษณ์ที่บอกว่า f เป็นฟังก์ชันจาก Domain D และ Range R คือ

$$F : D \rightarrow R$$

ในกรณีฟังก์ชัน abs ถ้าเราทำงานบน integer Domain และ range จะเป็น Z (เซตของ integer) เราจะเขียนได้ว่า $\text{abs} : Z \rightarrow Z$

ในกรณีการบวก Domain จะเป็น $Z \times Z$ ส่วน Range เป็น Z ดังนั้นจะเขียนได้เป็น $\text{add} : Z \times Z \rightarrow Z$

ฟังก์ชันที่ใช้ทุกสมาชิกใน Range จะเรียกว่า **onto**

ในการอธิบายฟังก์ชันนั้นมีหลายวิธี ทางหนึ่งที่ยิยมใช้คืออธิบายด้วย
กระบวนการคำนวณ output จากการกำหนด input อีกทางหนึ่งคือใช้
ตารางแสดงทุก input ที่เป็นไปได้ว่าให้ output อะไร

พิจารณาฟังก์ชัน $f: \{0,1,2,3,4\} \rightarrow \{0,1,2,3,4\}$

n	f(n)
0	1
1	2
2	3
3	4
4	0

ฟังก์ชันนี้เพิ่มค่า 1 ให้กับ input ผลลัพธ์ที่ได้ modulo 5

ในบางครั้งตาราง 2 มิติจะถูกใช้เมื่อ Domain ของฟังก์ชันเป็น Cartesian product ของ 2 เซต ต่อไปเป็นตัวอย่างฟังก์ชัน $g:Z_4 \times Z_4 \rightarrow Z_4$ เมื่อ $Z_i = \{0,1,2,\dots,i-1\}$ ซึ่ง g คือฟังก์ชันบวกแล้ว modulo 4

g	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

เมื่อโดเมนของฟังก์ชัน f เป็น $A_1 \times A_2 \times \dots \times A_k$ สำหรับบางเซต $A_1, A_2, \dots, A_k$ input ของฟังก์ชัน f จะเป็น k -tuple $(a_1, a_2, \dots, a_k)$ แล้วเราจะเรียก a_i ว่าเป็น argument ของ f

ฟังก์ชันที่มี k argument เราจะเรียกว่า k -ary function เช่นถ้า $k=2$ เราเรียกว่า **Binary function**

เราคุ้นเคยกับ Binary function ที่เขียนในรูป infix notation (เขียนด้วยสัญลักษณ์ของฟังก์ชันจะอยู่ระหว่าง 2 argument ส่วน prefix notation สัญลักษณ์จะมาก่อน

ตัวอย่างเช่นฟังก์ชัน add จะถูกเขียนในรูป infix notation ด้วยสัญลักษณ์ + ระหว่าง 2 argument เป็น $a+b$ แทนที่จะเขียน $\text{add}(a,b)$

Predicate หรือ **property** เป็นฟังก์ชันที่ Range มีค่าเป็น $\{\text{TRUE}, \text{FALSE}\}$ เช่นให้ even เป็น property ที่เป็น TRUE ถ้า input ที่เข้ามาเป็นเลขคู่และ FALSE ถ้า input เป็นเลขคี่ ดังนั้น $\text{even}(4)=\text{TRUE}$ ส่วน $\text{even}(5)=\text{FALSE}$

Property ที่โดเมนเป็นเซตของ k -tuples

$Ax_1x_2\dots x_k$ จะเรียกว่า k -ary relation หรือ k -ary relation on A

ตัวอย่างเช่น 2-ary relation เรียกว่า binary relation

ถ้า R เป็น binary relation ประโยค aRb หมายความว่า $aRb = \text{TRUE}$

เช่นกันถ้า k -ary relation ประโยค $R(a_1, a_2, \dots, a_k)$ หมายความว่า

$R(a_1, a_2, \dots, a_k) = \text{TRUE}$

ตัวอย่าง เกมเป่ายิ้งฉุบ จะมีผู้เล่นสองคนเลือกของจากเซต {SCISSORS, PAPER, STONE} ถ้าออกเหมือนกันถือว่าให้เริ่มใหม่ แต่ถ้าออกต่างกันจะมีคนหนึ่งชนะ ตามความสัมพันธ์ beats

beats	SCISSORS	PAPER	STONE
SCISSORS			
PAPER			
STONE			

Binary relation ชนิดพิเศษ เรียกว่า **equivalence relation** ซึ่งเป็นการพิจารณาว่า 2 object ว่าเท่ากันหรือไม่

Binary relation R เป็น equivalence relation ถ้า R เป็น equivalence relation ถ้า R สอดคล้องกับ 3 เงื่อนไข

- **reflexive** ถ้าทุก ๆ x , xRx
- **symmetry** ถ้าทุก ๆ x และ y , xRy หมายความว่า yRx
- **transitive** ถ้าทุก ๆ x , y และ z , xRy และ yRz หมายความว่า xRz

ตัวอย่างความสัมพันธ์แบบ equivalence

$R = '='$

Reflexive: ตรวจสอบว่า สำหรับทุก x , xRx

$$x = x$$

Symmetric: ตรวจสอบว่าสำหรับทุก ๆ x และ y , xRy หมายถึง yRx

$$x = y \quad \longrightarrow \quad y = x$$

Transitive: ตรวจสอบว่าสำหรับทุก ๆ x , y และ z , xRy และ yRz หมายถึง xRz

$$x = y \text{ and } y = z \quad \longrightarrow \quad x = z$$

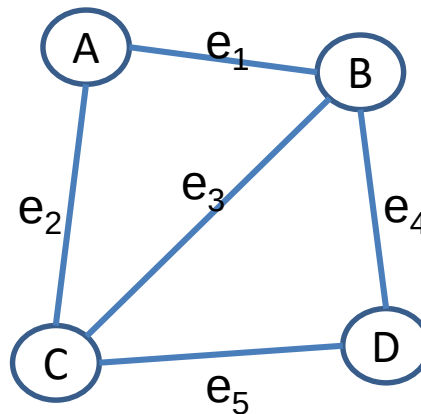
กำหนดให้ relation บนจำนวนธรรมชาติเขียนแทนด้วย $\equiv_7$
สำหรับ $i, j \in \mathbb{N}$ เราจะบอกว่า $i \equiv_7 j$ ถ้า $i-j$ เป็นจำนวนเท่าของ 7
 $\equiv_7$ เป็น equivalence relation หรือไม่

Graphs

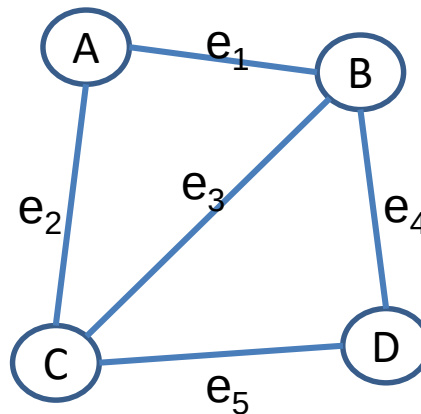
Undirected graph หรือเรียกสั้นๆ ว่า กราฟคือแบบจำลองทาง

คณิตศาสตร์ที่ประกอบไปด้วย 2 เซตคือ เซตของจุดยอด (Vertex, Node) $V(G)$ และ เซตของเส้นเชื่อม (Edge) $E(G)$

ตัวอย่างกราฟ $G = (V, E)$ ที่ $V(G) = \{A, B, C, D\}$ และ $E(G) = \{AB, AC, BC, BD, CD\}$ หรือ อาจจะเขียนเป็น $E(G) = \{e_1, e_2, e_3, e_4, e_5\}$



- จำนวนของเส้นเชื่อมที่ติดกับโหนดใดๆ จะเรียกว่าดีกรีของโหนด
- ในที่นี้เราจะสนใจกรณีที่ระหว่างโหนดใดๆ จะมีได้เพียง 1 เส้น
- ตัวอย่างด้านล่าง โหนด A มีดีกรี 2, โหนด C มีดีกรี 3



กราฟโดยทั่วไปแล้วจะถูกใช้เพื่อแสดงข้อมูล

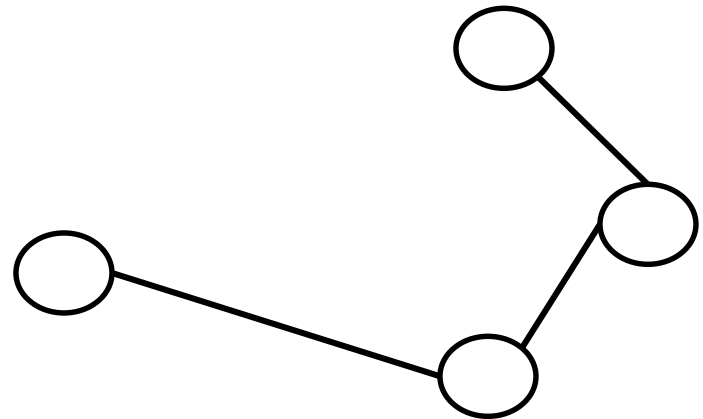
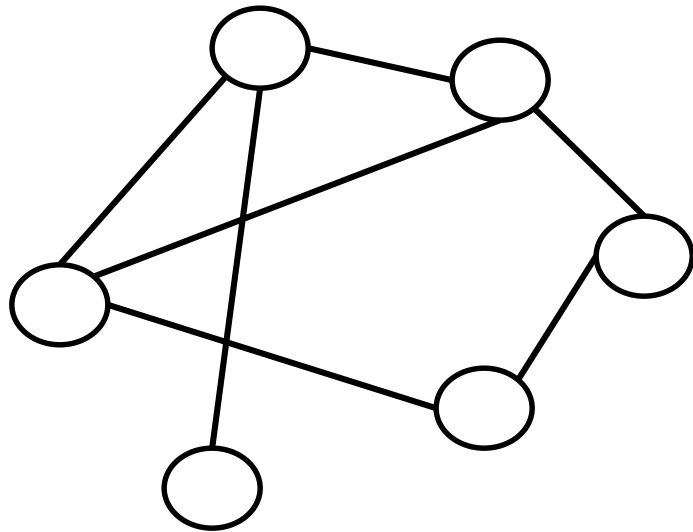
โหนดอาจจะหมายถึงเมืองและเส้นเชื่อมอาจหมายถึงถนนที่เชื่อมระหว่างเมือง

โหนดอาจจะหมายถึงส่วนประกอบทางอิเล็กทรอนิกส์และเส้นเชื่อมอาจหมายถึงสายไฟที่เชื่อมอุปกรณ์

บางครั้งเพื่อความสะดวกเราจะกำหนดชื่อให้กับโหนดหรือเส้นเชื่อมของกราฟ ซึ่งจะเรียกว่า **Labeled graph**

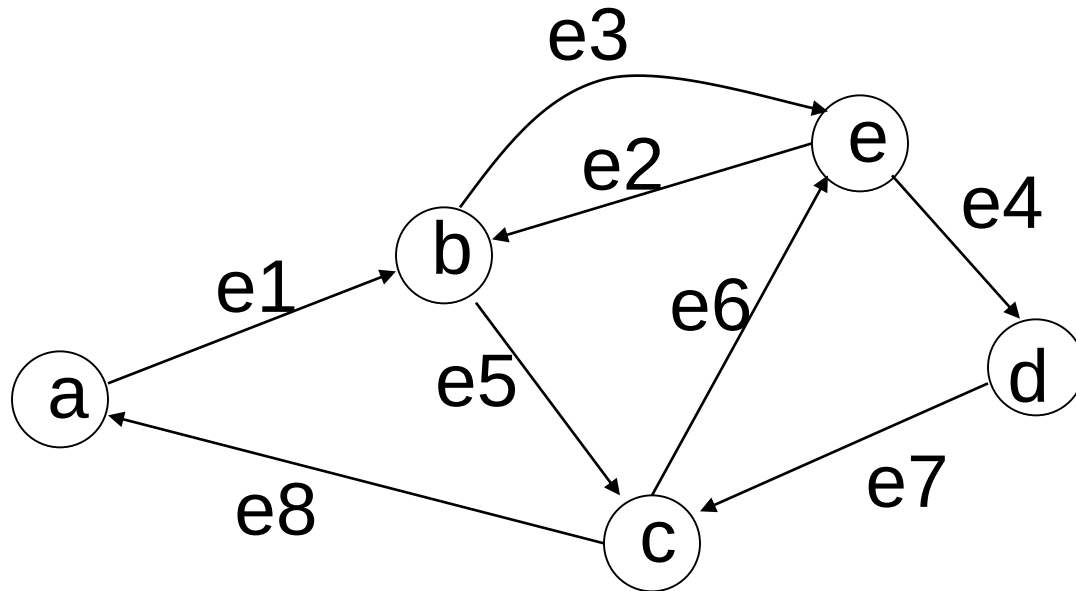
Subgraph

เราจะเรียกกราฟ G ว่าเป็น **subgraph** ของกราฟ H ถ้าโหนดในกราฟ G เป็น subset ของโหนดใน H และเส้นเชื่อมในกราฟ G เป็นเส้นเชื่อมของ H ที่สอดคล้องกับโหนด



Path

Path ในกราฟคือลำดับของโหนดที่ถูกเชื่อมด้วยเส้นเชื่อม



ตัวอย่างวิถีเช่น a,b,c,e,b,e,d

Simple path คือ **path** ที่ไม่มีโหนดซ้ำ

เราจะเรียกกราฟว่าเชื่อมกัน **connected** ถ้าทุกสองโหนดใด ๆ มี **path** ระหว่างมัน

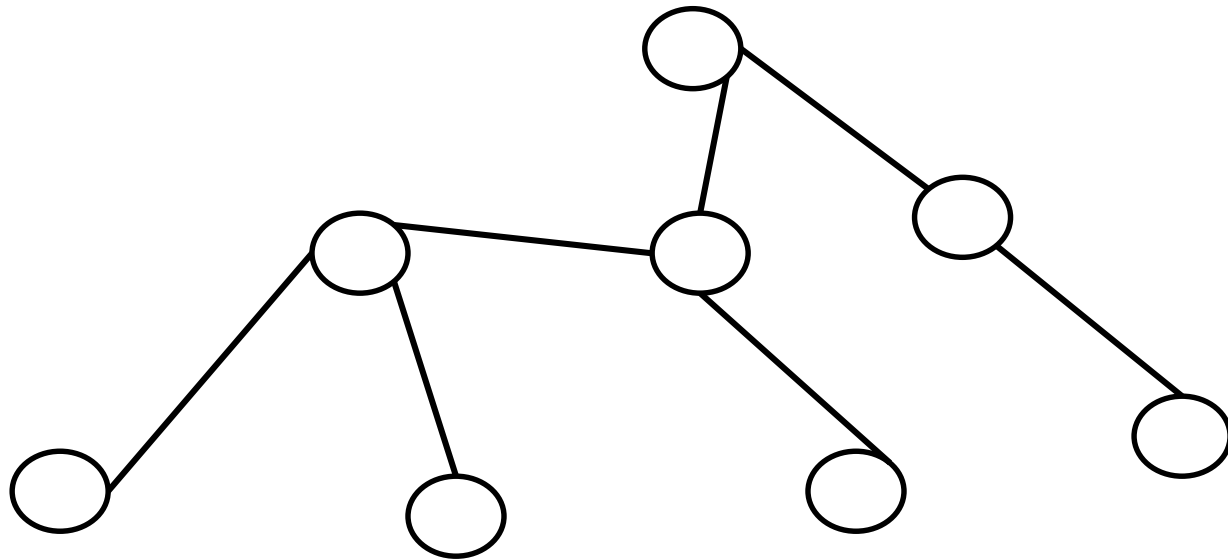
path จะเป็น **cycle** ถ้ามันเริ่มและจบที่จุดเดียวกัน

Simple cycle คือ **cycle** ที่ประกอบด้วยอย่างน้อย 3 โหนดและซ้ำกันแค่ โหนดแรกและโหนดสุดท้าย

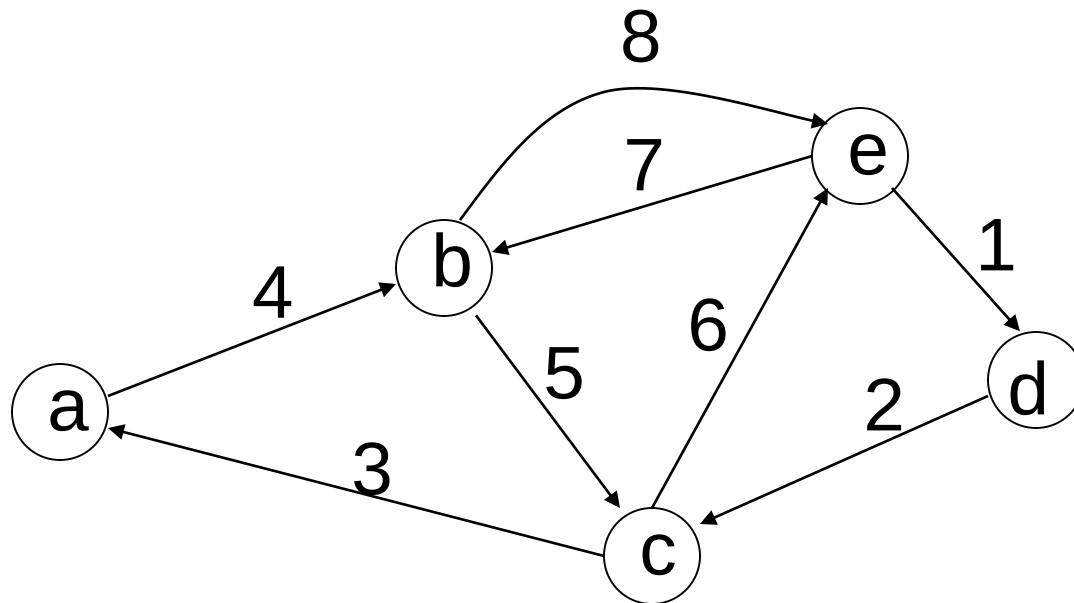
กราฟจะเป็น **tree** ถ้ามัน connected และไม่มี simple cycle

tree อาจจะมีโหนดพิเศษที่เรียกว่า **root**

โหนดที่มีดีกรีเป็น 1 ใน tree ที่ไม่ใช่ root จะเรียกว่า **leaves** ของ tree



ถ้ากราฟมีลูกศรแทนเส้นเชื่อม เราจะเรียกว่า **directed graph**



จำนวนของลูกศรที่ชี้ออกจากโหนดเรียกว่า **outdegree**

จำนวนลูกศรที่ชี้มาที่โหนดเรียกว่า **indegree**

ใน directed graph เราแสดงเส้นเชื่อมจาก i ไป j ด้วยคู่อันดับ (i,j)

นิยามอย่างเป็นทางการของ directed graph $G=(V,E)$ เมื่อ V เป็นเซตของ โหนดและ E เป็นเซตของเส้นเชื่อม

Path ที่ลูกศรใน path ชี้ไปในทิศทางเดียวกันต่อกันเป็นขั้นๆ เราเรียกว่า **directed path**

Directed graph จะเป็น **strongly connected** ถ้า directed path เชื่อมทุก คู่โหนดใดๆ

String and Language

ข้อความของอักขระ (Strings of characters) เป็นพื้นฐานของ computer science โดย อักขระที่ปรากฏบน string นั้นสามารถถูกนิยามได้อย่างหลากหลายขึ้นกับ application

ในที่นี้เราจะนิยาม **เซตอักขระ** (alphabet) ว่าเป็น nonempty finite set โดยที่สมาชิกของเซตอักขระคือ สัญลักษณ์ของอักขระ
เราจะใช้สัญลักษณ์ Σ ในการตั้งชื่อ

ตัวอย่างของเซตอักขระ

$$\begin{aligned}\Sigma_1 &= \{0,1\} \\ \Sigma_2 &= \{a, b, c, d, e, f, g, \dots, x, y, z\}\end{aligned}$$

String over an alphabet

คือลำดับจำกัดของสัญลักษณ์จากเซตอักขระ

ตัวอย่างเช่น

ถ้า $\Sigma_1 = \{0,1\}$ แล้ว 01001 เป็น string จากเซตอักขระ Σ_1

ถ้า w เป็น string จากเซตอักขระ Σ ความยาวของ w เขียนด้วย $|w|$ คือจำนวนของสัญลักษณ์ที่ปรากฏใน w

string ความยาว 0 จะเรียกว่า empty string เขียนแทนด้วย ϵ

ถ้า w มีความยาว n เราสามารถเขียนได้ว่า $w = w_1 w_2 \dots w_n$ เมื่อแต่ละ $w_i \in \Sigma$

Reverse ของ w เขียนแทนด้วย w^R คือ string ที่ได้มาจากการเขียน w ในลำดับตรงข้าม (นั่นคือ $w_n w_{n-1} \dots w_1$)

String z เป็น **substring** ของ w ถ้า z ปรากฏอย่างต่อเนื่องภายใน w

ตัวอย่างเช่น cad เป็น substring ของ $adfacadafgag$

ถ้าเรามี string x ความยาว m และ string y ความยาว n การเชื่อมต่อกัน **concatenations** ของ x และ y เขียนได้เป็น xy คือ string ที่ได้มาจากการต่อ string y ท้าย x

หากเป็นการต่อ string ด้วยตัวเองหลายครั้ง จะใช้ตัวยกบอกจำนวนครั้ง
ตัวอย่างเช่น

$$\underbrace{xxx \dots x}_k = x^k$$

Lexicographic ordering ของ strings คือการเรียงลำดับตามตัวอักษร คล้ายกับการเรียงในพจนานุกรม ยกเว้น string ที่สั้นกว่าจะอยู่ก่อน string ลำดับยาวกว่า ดังนั้น lexicographic ordering ของทุก string บน $\{0,1\}$ คือ $\{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$

ภาษา(Language) คือ set ของ strings

PROOF TECHNIQUES

- Proof by induction
- Proof by contradiction

Induction

We have statements $P_1, P_2, P_3, \dots$

If we know

- for some b that $P_1, P_2, \dots, P_b$ are true
- for any $k \geq b$ that

$$P_1, P_2, \dots, P_k \text{ imply } P_{k+1}$$

Then

Every P_i is true

Proof by Induction

- Inductive basis

Find $P_1, P_2, \dots, P_b$ which are true

- Inductive hypothesis

Let's assume $P_1, P_2, \dots, P_k$ are true,
for any $k \geq b$

- Inductive step

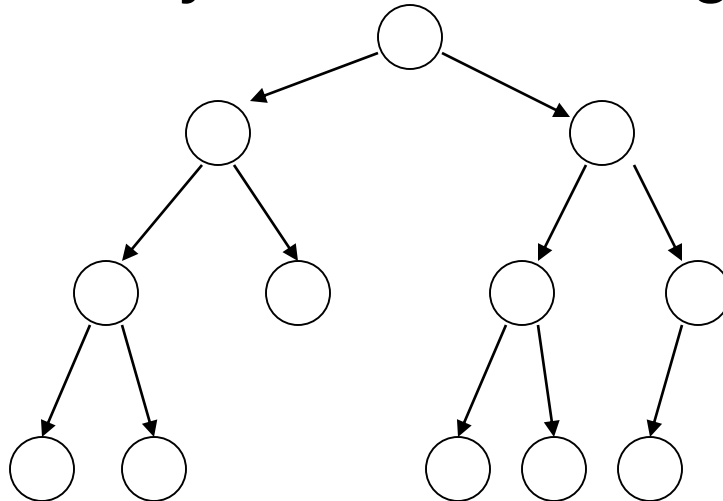
Show that P_{k+1} is true

Example

Theorem: A binary tree of height n
has at most 2^n leaves.

Proof by induction:

let $L(i)$ be the maximum number of
leaves of any subtree at height i



We want to show: $L(i) \leq 2^i$

- Inductive basis

$$L(0) = 1 \quad (\text{the root node}) \quad \bigcirc$$

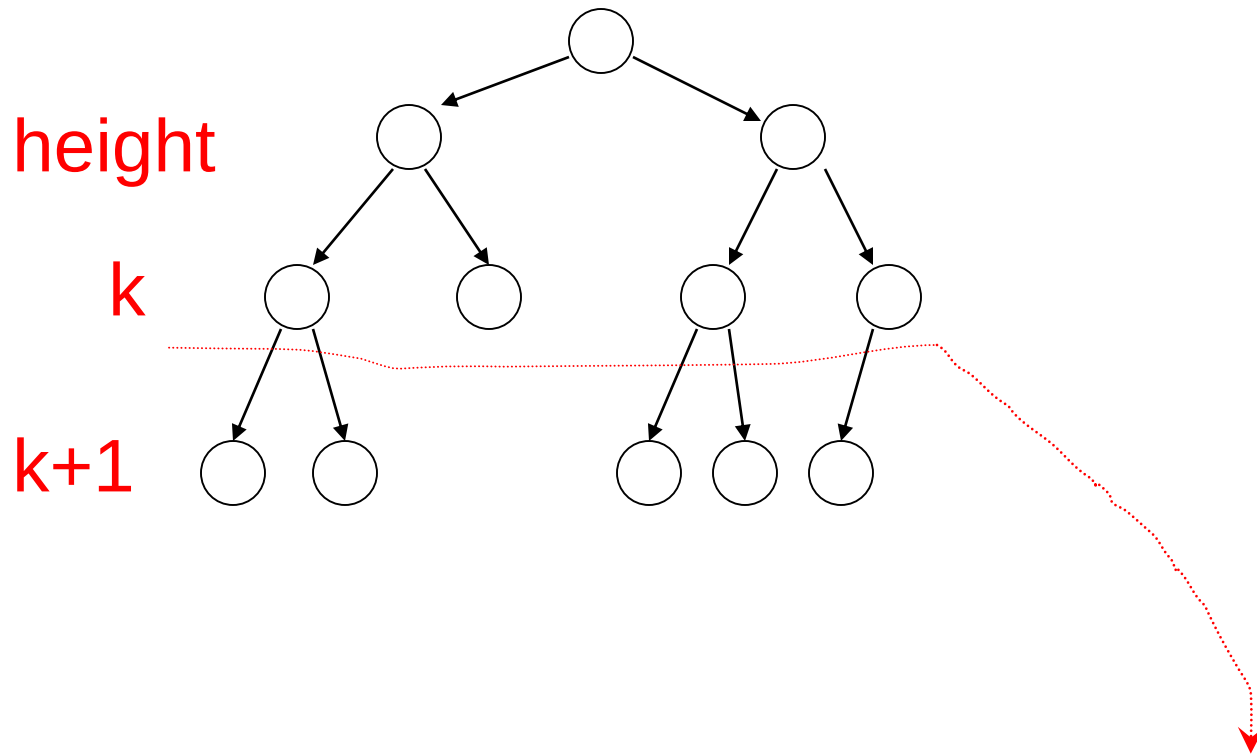
- Inductive hypothesis

Let's assume $L(i) \leq 2^i$ for all $i = 0, 1, \dots, k$

- Induction step

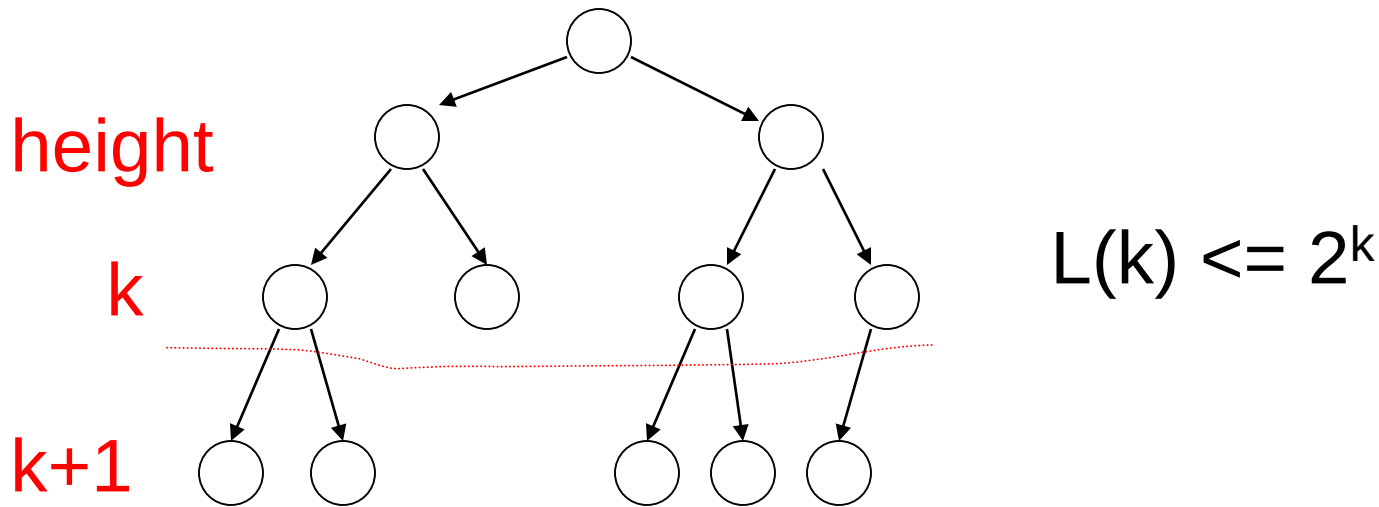
we need to show that $L(k + 1) \leq 2^{k+1}$

Induction Step



From Inductive hypothesis: $L(k) \leq 2^k$

Induction Step



$$L(k+1) \leq 2 * L(k) \leq 2 * 2^k = 2^{k+1}$$

(we add at most two nodes for every leaf of level k)

Remark

Recursion is another thing

Example of recursive function:

$$f(n) = f(n-1) + f(n-2)$$

$$f(0) = 1, \quad f(1) = 1$$

Proof by Contradiction

We want to prove that a statement P is true

- we assume that P is false
- then we arrive at an incorrect conclusion
- therefore, statement P must be true

Example

Theorem: $\sqrt{2}$ is not rational

Proof:

Assume by contradiction that it is rational

$$\sqrt{2} = n/m$$

n and m have no common factors

We will show that this is impossible

$$\sqrt{2} = n/m \quad \longrightarrow \quad 2 m^2 = n^2$$

Therefore, n^2 is even $\longrightarrow$ n is even
 $n = 2 k$

$$2 m^2 = 4k^2 \quad \longrightarrow \quad m^2 = 2k^2 \quad \longrightarrow \quad \begin{array}{l} m \text{ is even} \\ m = 2 p \end{array}$$

Thus, m and n have common factor 2

Contradiction!