ZUSAMMENFASSUNG

Propositional Logic

You can not ride the roller coaster if you are under 1 meter tall unless you are older than 16. (1)

Either you are not under 1 meter tall or you are older than 16, or you can not ride the roller coaster (2)

- Assign atomic propositions to propositional variables

  q: you can ride the roller coaster

  p: you are under 1 meter tall

  s: you are older than 16

(1) IF (p AND (NOT s)) THEN (NOT q)

$(p \wedge \neg s) \rightarrow \neg q$

(2) ((NOT p) OR s) OR (NOT q)

$(\neg p \vee s) \vee \neg q$

- used in mathematical proofs & computer programs
- allows simplifying programs or testing for equivalence

Applications

- Query Optimization
- Search Optimization
- AI / NLP
- Program simplification / verification

Logical Forms & Logical Equivalence

A proposition (statement) is either TRUE or FALSE

- The swiss flag is red and white	(TRUE)	✓
- It snowed last week in Zurich	(FALSE)	✓
- There is life on Mars	(FALSE)	✓
- $2+2=5$	(FALSE)	✓
- Study for this course!		✗
- This statement is false	(not decidable)	✗
- $X+Y>0$	(not decidable, depends on X & Y)	✗

Any complicated logical expression can be transformed into a proposition by using AND, OR, NEGATION (compound statement)

either p or q, or ... $(p \vee q) \vee \dots$

neither p nor q, or ... $(\neg p \wedge \neg q) \vee \dots$

Two compound statements are logically equivalent if and only if they have the same truth table

• Definition

A **statement** (or **proposition**) is a sentence that is true or false but not both.

~~Example~~, "$x + y > 0$" is not a statement because for some values of x and y the sentence is true, whereas for others it is false. For instance, if $x = 1$ and $y = 2$, the sentence is true; if $x = -1$ and $y = 0$, the sentence is false.

• Definition

A **statement form** (or **propositional form**) is an expression made up of statement variables (such as p, q, and r) and logical connectives (such as $\sim$, $\wedge$, and $\vee$) that becomes a statement when actual statements are substituted for the component statement variables. The **truth table** for a given statement form displays the truth values that correspond to all possible combinations of truth values for its component statement variables.

Symbols

$\sim p$: NOT P ("it is not the case of p") -> negation of P ~ is performed first (it has the precedence)

$p \wedge q$: P and q -> conjunction (of p and q)

$p \vee q$: p or q -> disjunction (of p and q)

$p \oplus q$: p or q but not both -> XOR (exclusive OR) true when one of the inputs true but not both.

~~equal in order of operation, and~~ an expression such as $p \wedge q \vee r$ is considered ambiguous. This expression must be written as either $(p \wedge q) \vee r$ or $p \wedge (q \vee r)$ to have meaning.

Words associated with symbols

p but q	means	p and q
neither p nor q	means	$\sim p$ and $\sim q$.

Truth Tables

p	q	$p \wedge q$	$q \wedge p$
T	T	T	T
T	F	F	F
F	T	F	F
F	F	F	F

$p \wedge q$ and $q \wedge p$ always have the same truth values, so they are logically equivalent

Two compound statements are logically equivalent if and only if they have the same truth table

NOT AND = OR
NOT OR = AND

Number of the rows we need: 2^n (where n represents the number of statements)

Basic Logical Equivalences

1.	*Commutative laws:*	$p \wedge q \equiv q \wedge p$	$p \vee q \equiv q \vee p$
2.	*Associative laws:*	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	$(p \vee q) \vee r \equiv p \vee (q \vee r)$
3.	*Distributive laws:*	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
4.	*Identity laws:*	$p \wedge \mathbf{t} \equiv p$	$p \vee \mathbf{c} \equiv p$
5.	*Negation laws:*	$p \vee \sim p \equiv \mathbf{t}$	$p \wedge \sim p \equiv \mathbf{c}$
6.	*Double negative law:*	$\sim(\sim p) \equiv p$	
7.	*Idempotent laws:*	$p \wedge p \equiv p$	$p \vee p \equiv p$
8.	*Universal bound laws:*	$p \vee \mathbf{t} \equiv \mathbf{t}$	$p \wedge \mathbf{c} \equiv \mathbf{c}$
9.	*De Morgan's laws:*	$\sim(p \wedge q) \equiv \sim p \vee \sim q$	$\sim(p \vee q) \equiv \sim p \wedge \sim q$
10.	*Absorption laws:*	$p \vee (p \wedge q) \equiv p$	$p \wedge (p \vee q) \equiv p$
11.	*Negations of* $\mathbf{t}$ *and* $\mathbf{c}$*:*	$\sim \mathbf{t} \equiv \mathbf{c}$	$\sim \mathbf{c} \equiv \mathbf{t}$

p	q	$p \wedge q$	$q \wedge p$
T	T	T	T
T	F	F	F
F	T	F	F
F	F	F	F

$p \wedge q$ and $q \wedge p$ always have the same truth values, so they are logically equivalent

• Definition

Two *statement forms* are called **logically equivalent** if, and only if, they have identical truth values for each possible substitution of statements for their statement variables. The logical equivalence of statement forms P and Q is denoted by writing $P \equiv Q$.

Two *statements* are called **logically equivalent** if, and only if, they have logically equivalent forms when identical component statement variables are used to replace identical component statements.

$(p \wedge \sim q) \vee p \equiv p \vee (p \wedge \sim q)$ by the commutative law for $\vee$

$\equiv p$ by the absorption law (with $\sim q$ in place of q)

Use of absorption law even if the statement is not the same (not q instead of q). What is important is p (q is not so relevant)

De Morgan Laws

De Morgan's Laws

The negation of an *and* statement is logically equivalent to the *or* statement in which each component is negated.

The negation of an *or* statement is logically equivalent to the *and* statement in which each component is negated.

$\sim(p \wedge q) \equiv \sim p \vee \sim q.$

$\sim(p \vee q) \equiv \sim p \wedge \sim q.$

Tautology and Contradiction

• Definition

A **tautology** is a statement form that is always true regardless of the truth values of the individual statements substituted for its statement variables. A statement whose form is a tautology is a **tautological statement.**

A **contradiction** is a statement form that is always false regardless of the truth values of the individual statements substituted for its statement variables. A statement whose form is a contradiction is a **contradictory statement.**

Exclusive and Inclusive OR

Or has two meanings

Examples:

- a waiter may ask: "coffee, tea, or milk" in an exclusive sense (either one but not all)
- A waiter may ask: "cream or sugar?" in an inclusive sense (meaning you can have either one or both)

$(p \vee q) \wedge \sim(p \wedge q)$ XOR

IF-THEN (is a conditional statement)

Definition corrective

If p and q are statement variables, the **conditional** of q by p is "If p then q" or "p implies q" and is denoted $p \rightarrow q$. It is false when p is true and q is false; otherwise it is true. We call p the **hypothesis** (or **antecedent**) of the conditional and q the **conclusion** (or **consequent**).

$p \rightarrow q \equiv \sim p \vee q.$

Truth Table for $p \rightarrow q$

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Caution! Remember that **the negation of an if-then statement does not start with the word *if*.**

The negation of "if p then q" is logically equivalent to "p and not q."

$\sim(p \rightarrow q) \equiv p \wedge \sim q$

Write negations for each of the following statements:

a. If my car is in the repair shop, then I cannot get to class. p, q — ~(p->q) ≡ p ∧ ~q — My car is in the repair shop and I can get to class

b. If Sara lives in Athens, then she lives in Greece. — Sara lives in Athens and she doesn't live in Greece

• Definition

The **contrapositive** of a conditional statement of the form "If p then q" is

If $\sim q$ then $\sim p$.

Symbolically,

The contrapositive of $p \rightarrow q$ is $\sim q \rightarrow \sim p$.

A conditional statement is logically equivalent to its

Write each of the following statements in its equivalent contrapositive form:

a. If Howard can swim across the lake, then Howard can swim to the island.

b. If today is Easter, then tomorrow is Monday.

Solution

a. If Howard cannot swim to the island, then Howard cannot swim across the lake

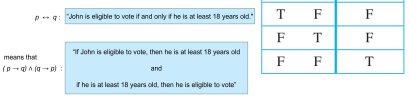
b. If tomorrow is not Monday, then today is not Easter.

IF AND ONLY IF (biconditional

• Definition

Given statement variables p and q, the **biconditional of p and q** is "p if, and only if, q" and is denoted $p \leftrightarrow q$. It is true if both p and q have the same truth values and is false if p and q have opposite truth values. The words *if and only if* are sometimes abbreviated **iff.**

The biconditional has the following truth table:



Truth Table for $p \leftrightarrow q$

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

and is read:

- "***p if and only if q***"
- "***p iff q***"
- "***p* implies *q* and *q* implies *p***"
- "***p* is necessary and sufficient condition for *q***

Btw: Notice that the negation of a biconditional is a XOR

$$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$$

If and *Only If*

Rewrite the following statement as a conjunction of two if-then statements:

This computer program is correct if, and only if, it produces correct answers for all possible sets of input data.

Solution If this program is correct, then it produces the correct answes for all possible sets of input data; and if this program produces the correct answers for all possible sets of input data, then it is correct. ■

Converting a Necessary Condition to If-Then Form

Use the contrapositive to rewrite the following statement in two ways:

George's attaining age 35 is a necessary condition for his being president of the United States.

Solution *Version 1:* If George has not attained the age of 35, then he cannot be president of the United States.

Version 2: If George can be president of the United States, then he has attained the age of 35. ■

"p if, and only if, q." For example, consider the statement "You will get dessert if, and only if, you eat your dinner." Logically, this is equivalent to the conjunction of the following two statements.

Statement 1: If you eat your dinner, then you will get dessert.

Statement 2: You will get dessert only if you eat your dinner.

or

If you do not eat your dinner, then you will not get dessert.

Test Yourself

1. An *if-then* statement is false if, and only if, the hypothesis is True and the conclusion is False
2. The negation of "if p then q" is P and not q
3. The converse of "if p then q" is If q then p
4. The contrapositive of "if p then q" is If not q then not p
5. The inverse of "if p then q" is If not p then not q
6. A conditional statement and its contrapositive are logically equivalent
7. A conditional statement and its converse are not logically equivalent
8. "R is a sufficient condition for S" means "if R then S."
9. "R is a necessary condition for S" means "if S then R."
10. "R only if S" means "if R then S."

Some programming languages use statements of the form "r unless s" to mean that as long as s does not happen, then r will happen. More formally:

Definition: If r and s are statements,

r unless s means if $\sim s$ then r.

• Definition

Suppose a conditional statement of the form "If p then q" is given.

1. The **converse** is "If q then p."
2. The **inverse** is "If $\sim p$ then $\sim q$."

Symbolically,

The converse of $p \rightarrow q$ is $q \rightarrow p$,

and

The inverse of $p \rightarrow q$ is $\sim p \rightarrow \sim q$.

Caution! Many people believe that if a conditional statement is true, then its converse and inverse must also be true. This is not correct!

1. A conditional statement and its converse are *not* logically equivalent.
2. A conditional statement and its inverse are *not* logically equivalent.
3. The converse and the inverse of a conditional statement are logically equivalent to each other.

The inverse of a conditional statement is the contrapositive

Converting *Only If* to *If-Then*

Rewrite the following statement in if-then form in two ways, one of which is the contrapositive of the other.

John will break the world's record for the mile run only if he runs the mile in under four minutes.

Solution *Version 1:* If John does not run the mile in under four minutes, then he will not break the world's record.

Version 2: If John breaks the world's record, then he will have run the mile in under four minutes. ■

Order of Operations for Logical

Order of Operations for Logical Operators

1. $\sim$ — Evaluate negations first.
2. $\wedge, \vee$ — Evaluate $\wedge$ and $\vee$ second. When both are present, parentheses may be needed.
3. $\rightarrow, \leftrightarrow$ — Evaluate $\rightarrow$ and $\leftrightarrow$ third. When both are present, parentheses may be needed.

Necessary and Sufficient Conditions

• Definition

If r and s are statements:

r is a **sufficient condition** for s means "if r then s."

r is a **necessary condition** for s means "if not r then not s."

In other words, to say "r is a sufficient condition for s" means that the occurrence of r is *sufficient* to guarantee the occurrence of s. On the other hand, to say "r is a necessary condition for s" means that if r does not occur, then s cannot occur either:

Consider the statement "If John is eligible to vote, then he is at least 18 years old." The truth of the condition "John is eligible to vote" is *sufficient* to ensure the truth of the condition "John is at least 18 years old." In addition, the condition "John is at least 18 years old" is *necessary* for the condition "John is eligible to vote" to be true. If John were younger than 18, then he would not be eligible to vote. ■

Rewrite the following statement in the form "If A then B":

Pia's birth on U.S soil is a sufficient condition for her to be a U.S. citizen.

Solution If Pia was born on U.S. soil, then she is a U.S. citizen.

AND (Conjunction)

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

$\sim p \wedge q = (\sim p) \wedge q$

$(p \wedge q) \vee r = p \wedge (q \vee r)$

OR (Disjunction)

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

XOR $\oplus$ -> True if they differ (one true and the other false)

p	q	$p \vee q$	$p \wedge q$	$\sim(p \wedge q)$	$(p \vee q) \wedge \sim(p \wedge q)$
T	T	T	T	F	F
T	F	T	F	T	T
F	T	T	F	T	T
F	F	F	F	T	F

p or q, but not both

→ IF-THEN

p	q	p → q
T	T	T
T	F	F
F	T	T
F	T	T

hypothesis, conclusion

≡

¬p ∨ q
T
F
T
T

If 0=1 (F) then 1=2 (F) (T)

If 2+2=4 (T), then earth is a planet (T) (T)

If 2+2=5 (F), then 11 is a prime (T) (T)

If my car is broken (T), Lea will fix it (T)
T → T ≡ T

contradiction

Never the case, that my car is broken (T) AND Lea won't fix it (F)
T → F ≡ F

Either my car is not broken (F) OR Lea will fix it (T)
T → (T) ≡ T
(F) → T ≡ T
(F) → F ≡ T

T → F ≡ F
(✗) ∨ (✗)

!

If my car is broken (T), Lea will fix it (T)	T
If my car is broken (T), Lea won't fix it (F)	F
If my car is not broken (F), Lea will fix it (T)	T
If my car is not broken (F), Lea won't fix it (F)	T

$\leftrightarrow$ Biconditional IF

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

$\equiv$

$(p \rightarrow q) \wedge (q \rightarrow p)$
T
F
F
T

John is eligible to vote IF AND ONLY IF he is older than 18
(p) (q)

If John is eligible to vote, then he is older than 18

AND

If he is older than 18, then he is eligible to vote

$(p \rightarrow q) \wedge (q \rightarrow p)$

Either John is <u>not</u> eligible to vote OR he is older than 18

AND

Either he is <u>not</u> older than 18 OR he is eligible to vote

$(\sim p \vee q) \wedge (\sim q \vee p)$

Logical Equivalences

1) Commutative laws: $p \wedge q \equiv q \wedge p$ switch pos
$p \vee q \equiv q \vee p$

2) Associative laws: $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$ switch brackets
$(p \vee q) \vee r \equiv p \vee (q \vee r)$

---

3) Distributive laws: $(p \wedge q) \vee (p \wedge r) \equiv p \wedge (q \vee r)$!
$(p \vee q) \wedge (p \vee r) \equiv p \vee (q \wedge r)$

---

4) Identity laws: $p \wedge t \equiv p$
$p \vee c \equiv p$

8) Universal bound laws: $p \vee t \equiv t$ OR true = true
$p \wedge c \equiv c$ AND false = false

5) Negation laws: $p \vee \neg p \equiv t$
$p \wedge \neg p \equiv c$

---

6) Double negative law: $\neg(\neg p) \equiv p$

7) Idempotent laws: $p \wedge p \equiv p$
$p \vee p \equiv p$

9) De Morgan's laws: $\sim(p \wedge q) \equiv \sim p \vee \sim q$

$\sim(p \vee q) \equiv \sim p \wedge \sim q$

---

10) Absorption laws: $p \vee (p \wedge q) \equiv p$

$p \wedge (p \vee q) \equiv p$

---

11) Negations of t, c $\sim t \equiv c$

$\sim c \equiv t$

$p \to q \equiv \sim p \vee q$

---

t = tautology = always true

c = contradiction = always false

De Morgan's law: The bus was late or Tom's watch was slow

p: Bus was late

q: Tom's watch was slow $p \vee q$

Negation $\sim(p \vee q) \equiv \sim p \wedge \sim q$

The bus was not late and Tom's watch was not slow

Simplification

$$\sim(\sim p \wedge q) \wedge (p \vee q) \equiv p$$

$\equiv (\sim(\sim p) \vee \sim q) \wedge (p \vee q)$ — DE MORGAN

$\equiv (p \vee \sim q) \wedge (p \vee q)$ — DOUBLE NEGATION

$\equiv p \vee (\sim q \wedge q)$ — DISTRIBUTIVE

$\equiv p \vee c$ — NEGATION LAW

$\equiv p$ — IDENTITY LAW

Prove Equivalence

$(p \wedge q) \vee (p \wedge \sim q)$

$\equiv p \wedge (q \vee \sim q)$

$\equiv p \wedge t$

$\equiv p$

$\sim((\sim p \wedge q) \vee (\sim p \wedge \sim q)) \vee (p \wedge q)$

$\equiv \sim(\sim p \wedge (q \vee \sim q)) \vee (p \wedge q)$

$\equiv \sim(\sim p \wedge t) \vee (p \wedge q)$

$\equiv \sim(\sim p) \vee \sim t \vee (p \wedge q)$

$\equiv p \vee c \vee (p \wedge q)$

$\equiv p \vee p \wedge q$

$\equiv p \wedge q$

NOT EQUIVALENT

Digital Logic Circuits

Arithmetic Logic Unit (ALU)

- a combinational digital circuit
- performs arithmetic and bitwise operations on two binary integers
- building block of CPU & GPU

Arithmetic Operations: ADD, SUBTRACT, INCREMENT, DECREMENT

Bitwise logical operations: Bitwise AND, Bitwise OR, Bitwise XOR,
Bit shift (• 10, 110 → 1100)

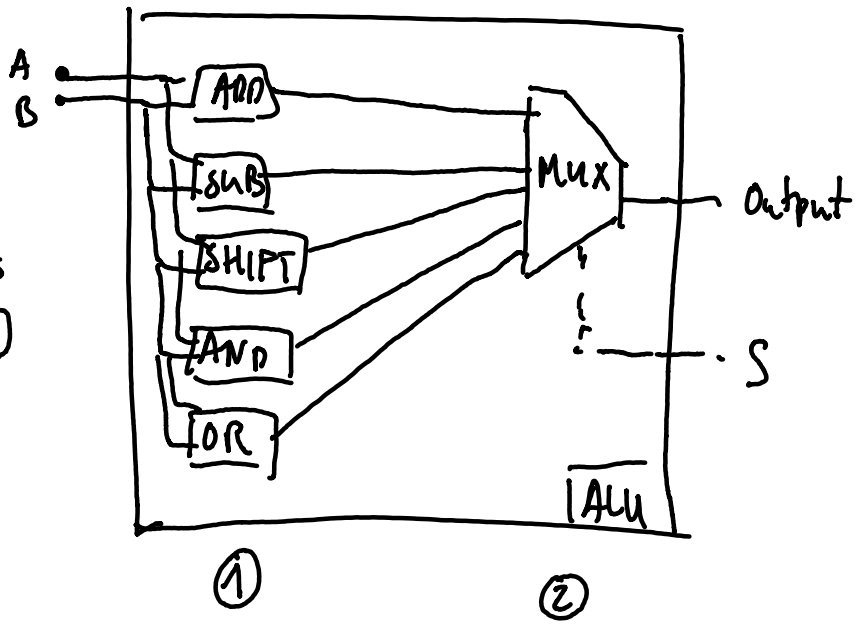
Multiplexer (MUX)

→ special circuit that allows to report the values of either the inputs to the output.

like a switch

When A and B are input to the ALU, all instructions are executed in parallel ①

Only the output of the operation specified as OPCODE is selected ②



light bulb turns off if current flows through it.

Digital Logic Circuits

✓ switch can be opened/closed

In series (AND)
↳ current flows if and only if both switches P and Q are closed.

In parallel (OR) → current flows, if and only if, at least on of the switches P or Q is closed.

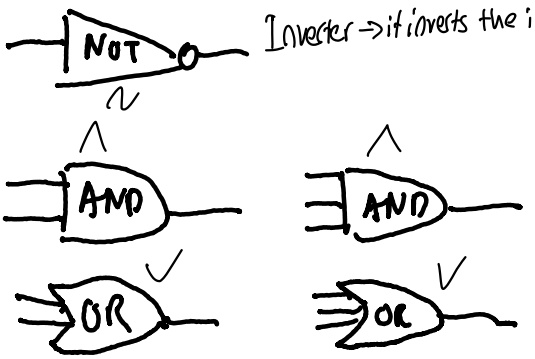
- Electrical switches are implemented with transistors and 1 and 0 physically represents high voltage and low voltage
  ↳ bits (binary digits)

Combinatorial Circuit: Output only depends on current input
↳ its operation is precisely specified by its input/output table
↳ built by connecting logic gates

Sequential Circuit: output depends on current input and previous outputs (memory)

Gates can be combined into a combinational circuit.

Logic Gates



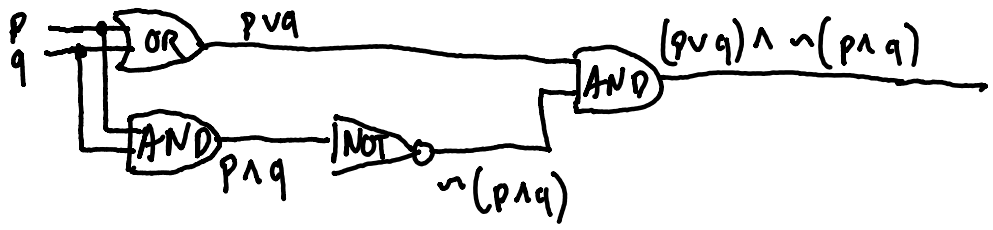
Inverter → it inverts the input value.

Rules of combinatorial Circuit

Never link inputs

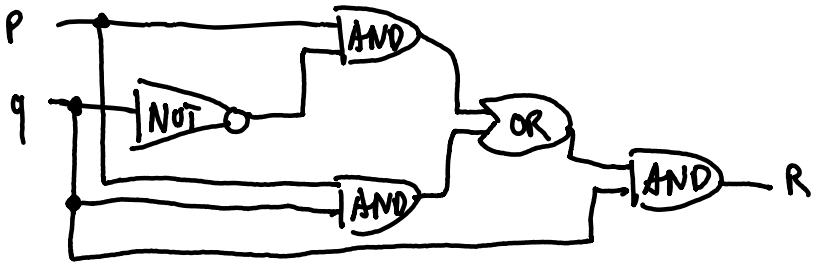
Never feed back

NOT ≡ ∼
AND ≡ ∧
OR ≡ ✓



Always try to simplify the expression before drawing the circuit. → circuit minimization (also helps to reduce the delay between input and output).

Equivalence of Combinatorial Circuits



$$\equiv (p \wedge (\neg q \vee q)) \wedge q$$

$$\equiv (p \wedge t) \wedge q$$

$$\equiv p \wedge q$$

Combinational circuits are equivalent if their input-output tables are identical, which means they implement the same logical expression.

SAME

Input/Output Table for a Recognizer

P	Q	R	$(P \wedge Q) \wedge \neg R$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

Recognizer

digital logic circuit that outputs a 1 for exactly one particular combination of input signals and 0 for all other combinations.

$$\equiv p \wedge q$$

Reverse-Engineering

How to build the circuit from the table?

Recognizer

P	Q	R	S
1	1	1	1
1	1	0	0
1	0	1	1
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

Disjunctive Normal Form "OR of AND's"

↳ to use with few 1 outputs and many 0 outputs.

- Identify the rows whose output is 1
- AND together all variables in the same row (negate the variable when 0)
- OR together the 1 rows

$P \wedge Q \wedge R$

$P \wedge \neg Q \wedge R$

$P \wedge \neg Q \wedge \neg R$

p	q	$G(p,q)$	
1	1	1	→ $p \wedge q$
1	0	0	
0	1	1	→ $\neg p \wedge q$
0	0	0	

$G(p,q) = (p \wedge q) \vee (\neg p \wedge q)$

$$\equiv (P \wedge Q \wedge R) \vee (P \wedge \neg Q \wedge R) \vee (P \wedge \neg Q \wedge \neg R)$$

P	Q	R	S
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	1
0	0	1	1
0	0	0	1

Conjunctive Normal Form "AND of OR's"

$\neg P \vee \neg Q \vee \neg R$ (negate if 1)

$\neg P \vee Q \vee \neg R$ (negate if 1)

$\neg P \vee Q \vee R$ (negate if 1)

$$\equiv (\neg P \vee \neg Q \vee \neg R) \wedge (\neg P \vee Q \vee \neg R) \wedge (\neg P \vee Q \vee R)$$

- Identify the rows whose output is 0.
- OR together all variables in same row (negate the variable when 1)
- AND together the 0 rows

→ to use with few 0 outputs and many 1 outputs.

p	q	$G(p,q)$	
1	1	1	
1	0	0	→ $\neg p \vee q$
0	1	1	
0	0	0	→ $p \vee q$

$G(p,q) \equiv (\neg p \vee q) \wedge (p \vee q)$

Convert binary to decimal

←

$$110101_2 = 1\cdot 2^0 + 1\cdot 2^2 + 1\cdot 2^4 + 1\cdot 2^5$$
$$= 1 + 4 + 16 + 32 = 53_{10}$$

Convert binary to decimal (lecture method)

→

$$\underset{\text{most sig.}}{1}\,1\,0\,1\,0\,\underset{\text{least sig.}}{1}_2 = \Big(\Big(\Big(\big(\underset{\text{most sig.}}{1}\cdot 2 + 1\big)\cdot 2 + 0\Big)\cdot 2 + 1\Big)\cdot 2 + 0\Big)\cdot 2 + \underset{\text{least sig.}}{1}$$
$$= 53$$

Convert decimal to binary (successive division)

$142_{10} = ?_2$

142/2	= 71	R	0
71/2	= 35	R	1
35/2	= 17	R	1
17/2	= 8	R	1
8/2	= 4	R	0
4/2	= 2	R	0
2/2	= 1	R	0
1/2	= 0	R	1

↑

↳ stop

10001110_2

Convert decimal to binary

Addition of Binary Numbers

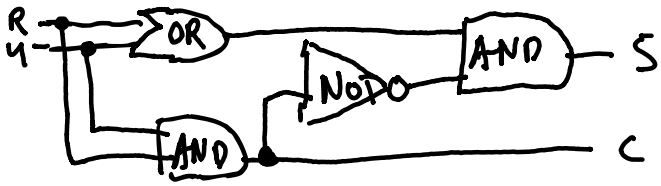
Carry					
	0	1	0	1	1
	0	0	1	1	1
	0	1	1	10	11

↳ outputs the sum of (any) two binary digits.

Half Adder

2 digits (right most digits)

R	V	Carry	Sum
1	1	1	0
1	0	0	1
0	1	0	1
0	0	0	0
		$R \wedge V$	$(R \vee V) \wedge \sim(R \wedge V)$
		AND	XOR

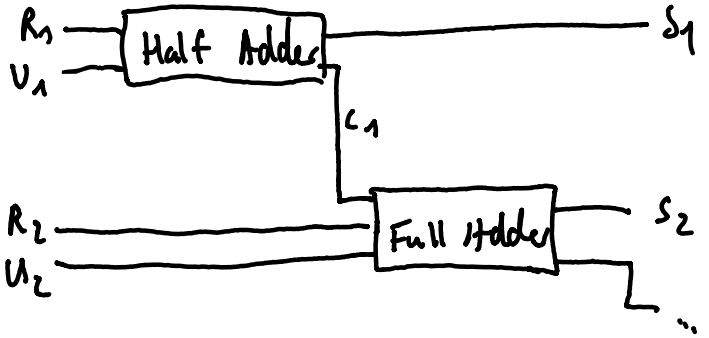


↳ outputs the sum of (any) 3 binary digits.

Full Adder

3 digits

C	R	V	Carry	Sum
1	1	1	1	1
1	1	0	1	0
1	0	1	1	0
1	0	0	0	1
0	1	1	1	0
0	1	0	0	1
0	0	1	0	1
0	0	0	0	0



Add 1 bit integers → half adder

Add n bit integers → half adder + (n-1 full adder)

Algorithms & Recursion

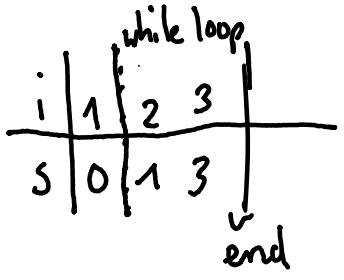
Algorithm

- step-by-step description of a method to perform an action
- sequence of instructions to solve a problem

Iterative

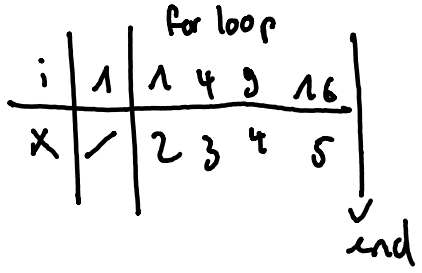
while-loop

```
i := 1
s := 0
while (i ≤ 2)
    s := s+i
    i := i+1
end while
```



for-loop

```
for i := 1 to 4
    x := i²
next i
```



→ use while-loop if we don't know how many times to loop beforehand

Recursion

- Algorithm calls itself
- break down problem into smaller pieces (find psyc in book)
  Binary Search

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2f(n-1) & \text{if } n > 0 \end{cases}$$

leap of faith: assume that it is going to work for the k-th step and prove that this can be used to solve k+1 step

Algorithm Correctness

- A program is correct if it produces the correct output for a set of expected inputs
- Use induction / loop invariant to prove correctness
- An iterative / recursive algorithm is correct if and only if, whenever the input variables belong to the domain of acceptable inputs and the algorithm terminates after a finite number of steps, the correct output has been returned

  → Fullfil Pre-Condition, terminates, fullfil Post-condition

Pre-condition: initial state, determines the domain of acceptable inputs

Post-Condition: final state, determines what the algorithm is supposed to compute

Example gcd(A,B)

A & B are integers with $A > B \geq 0$

$a = gcd(A,B)$

↓

result variable in algo

Loop Invariant

- Property $P(n)$ for a loop defined on a set of integers satisfying the condition:

  For each iteration K of the loop,
  if the $P(K)$ is true before the iteration of K,
  then after the iteration $P(K) \rightarrow P(K+1)$ is true

- based on mathematical induction

Let $P(n)$ be the loop invariant for a loop. If the following four properties are true, the the loop is correct with respect to the Pre-Condition / Post-Condition:

1) Basis Property: $P(0)$ is true, which is before the first iteration
2) Inductive Property: For any integer $K \geq 0$, if the guarol G and the loop invariant $P(K)$ are both true before the iteration, then $P(K+1)$ is true after the iteration
3) Eventual Falsity of the Guard: the guard G becomes false after a finite number of iterations
4) Correctness of the post-condition: if n is the last iteration after which G is false and $P(n)$ is true, then the post-condition is satisfied

→ example 03_Induction from slide 105

Sequences

- set of elements that can be indexed by an integer number (array), $\{a_1, a_2, a_3, a_4, \dots\}$ a_k is called term, k is called subscript or index
- it's convenient to find an <u>explicit formula</u> to express the generic element a_k as a function of k

Recurrences

A recurrence relation for a subsequence is a formula that relates each term to some of its predecessors

$$a_k = f(a_{k-1}, a_{k-2}, \dots a_{k-i})$$

with initial values: $a_0, a_1, a_2 \dots a_{i-1}$

- If a_k only depends on its predecessor (a_{k-1}), it's called 1st order recurrence relation
- If a_k only depends on its two predecessors (a_{k-1}, a_{k-2}), it's called 2nd order recurrence relation (fibonacci)

$$\sum_{i=1}^{n} s_i \rightarrow \begin{cases} s_1 = 1 \\ s_k = s_{k-1} + k \end{cases}$$

$$\prod_{i=1}^{n} s_i \rightarrow \begin{cases} s_1 = 1 \\ s_k = s_{k-1} \cdot k \end{cases}$$

proven by induction

Theorem 5.1.1

If $a_m, a_{m+1}, a_{m+2}, \ldots$ and $b_m, b_{m+1}, b_{m+2}, \ldots$ are sequences of real numbers and c is any real number, then the following equations hold for any integer $n \geq m$:

1. $\displaystyle\sum_{k=m}^{n} a_k + \sum_{k=m}^{n} b_k = \sum_{k=m}^{n} (a_k + b_k)$

2. $\displaystyle c \cdot \sum_{k=m}^{n} a_k = \sum_{k=m}^{n} c \cdot a_k$ generalized distributive law

3. $\displaystyle \left(\prod_{k=m}^{n} a_k\right) \cdot \left(\prod_{k=m}^{n} b_k\right) = \prod_{k=m}^{n} (a_k \cdot b_k).$

Method of Proof by Mathematical Induction

Consider a statement of the form, "For all integers $n \geq a$, a property $P(n)$ is true." To prove such a statement, perform the following two steps:

Step 1 (basis step): Show that $P(a)$ is true.

Step 2 (inductive step): Show that for all integers $k \geq a$, if $P(k)$ is true then $P(k+1)$ is true. To perform this step,

> **suppose** that $P(k)$ is true, where k is any particular but arbitrarily chosen integer with $k \geq a$.
> *[This supposition is called the* **inductive hypothesis.***]*
>
> Then
>
> **show** that $P(k+1)$ is true.

Principle of Strong Mathematical Induction

Let $P(n)$ be a property that is defined for integers n, and let a and b be fixed integers with $a \leq b$. Suppose the following two statements are true:

1. $P(a), P(a+1), \ldots,$ and $P(b)$ are all true. **(basis step)** We prove more base cases
2. For any integer $k \geq b$, if $P(i)$ is true for all integers i from a through k, then $P(k+1)$ is true. **(inductive step)** and not just for an arbitrary $k \geq b$

Then the statement

for all integers $n \geq a$, $P(n)$

is true. (The supposition that $P(i)$ is true for all integers i from a through k is called the **inductive hypothesis.** Another way to state the inductive hypothesis is to say that $P(a), P(a+1), \ldots, P(k)$ are all true.)

Prime Factorization Theorem

"Every integer $n > 1$ is either a prime number or can be written as the product of prime numbers."

Geometric Series

$$\sum_{i=0}^{n} r^i = \frac{r^{n+1} - 1}{r - 1}.$$

Laws of Exponents

If b and c are any positive real numbers and u and v are any real numbers, the following laws of exponents hold true:

$$b^u b^v = b^{u+v} \quad 7.2.1$$

$$(b^u)^v = b^{uv} \quad 7.2.2$$

$$\frac{b^u}{b^v} = b^{u-v} \quad 7.2.3$$

$$(bc)^u = b^u c^u \quad 7.2.4$$

Theorem 7.2.1 Properties of Logarithms

For any positive real numbers b, c and x with $b \neq 1$ and $c \neq 1$:

a. $\log_b(xy) = \log_b x + \log_b y$

b. $\log_b\left(\frac{x}{y}\right) = \log_b x - \log_b y$

c. $\log_b(x^a) = a \log_b x$

d. $\log_c x = \frac{\log_b x}{\log_b c}$

SETS

$A \subseteq B \Leftrightarrow \forall x$, if $x \in A$ then $x \in B$.

$A \not\subseteq B \Leftrightarrow \exists x$ such that $x \in A$ and $x \notin B$.

• Definition

Given sets A and B, A **equals** B, written $A = B$, if, and only if, every element of A is in B and every element of B is in A.

Symbolically:

$$A = B \Leftrightarrow A \subseteq B \text{ and } B \subseteq A.$$

• Definition

Let A and B be subsets of a universal set U.

1. The **union** of A and B, denoted $A \cup B$, is the set of all elements that are in at least one of A or B.
2. The **intersection** of A and B, denoted $A \cap B$, is the set of all elements that are common to both A and B.
3. The **difference** of B minus A (or **relative complement** of A in B), denoted $B - A$, is the set of all elements that are in B and not A.
4. The **complement** of A, denoted A^c, is the set of all elements in U that are not in A.

Symbolically:

$$A \cup B = \{x \in U \mid x \in A \text{ or } x \in B\},$$
$$A \cap B = \{x \in U \mid x \in A \text{ and } x \in B\},$$
$$B - A = \{x \in U \mid x \in B \text{ and } x \notin A\},$$
$$A^c = \{x \in U \mid x \notin A\}.$$

There is a convenient notation for subsets of real numbers that are intervals.

• Notation

Given real numbers a and b with $a \leq b$:

$$(a, b) = \{x \in \mathbf{R} \mid a < x < b\} \qquad [a, b] = \{x \in \mathbf{R} \mid a \leq x \leq b\}$$
$$(a, b] = \{x \in \mathbf{R} \mid a < x \leq b\} \qquad [a, b) = \{x \in \mathbf{R} \mid a \leq x < b\}.$$

The symbols ∞ and $-\infty$ are used to indicate intervals that are unbounded either on the right or on the left:

$$(a, \infty) = \{x \in \mathbf{R} \mid x > a\} \qquad [a, \infty) = \{x \in \mathbf{R} \mid x \geq a\}$$
$$(-\infty, b) = \{x \in \mathbf{R} \mid x < b\} \qquad [-\infty, b) = \{x \in \mathbf{R} \mid x \leq b\}.$$

• Definition

Two sets are called **disjoint** if, and only if, they have no elements in common.

Symbolically:

$$A \text{ and } B \text{ are disjoint} \Leftrightarrow A \cap B = \emptyset.$$

• Definition

Sets $A_1, A_2, A_3 \ldots$ are **mutually disjoint** (or **pairwise disjoint** or **nonoverlapping**) if, and only if, no two sets A_i and A_j with distinct subscripts have any elements in common. More precisely, for all $i, j = 1, 2, 3, \ldots$

$$A_i \cap A_j = \emptyset \quad \text{whenever } i \neq j.$$

a. Let $A_1 = \{3, 5\}$, $A_2 = \{1, 4, 6\}$, and $A_3 = \{2\}$. Are A_1, A_2, and A_3 mutually disjoint?

a. Yes. A_1 and A_2 have no elements in common, A_1 and A_3 have no elements in common, and A_2 and A_3 have no elements in common.

• Definition

Given a set A, the **power set** of A, denoted $\mathscr{P}(A)$, is the set of all subsets of A.

Theorem 6.2.2 Set Identities

Let all sets referred to below be subsets of a universal set U.

1. *Commutative Laws:* For all sets A and B,
$$\text{(a) } A \cup B = B \cup A \quad \text{and} \quad \text{(b) } A \cap B = B \cap A.$$
2. *Associative Laws:* For all sets A, B, and C,
$$\text{(a) } (A \cup B) \cup C = A \cup (B \cup C) \quad \text{and}$$
$$\text{(b) } (A \cap B) \cap C = A \cap (B \cap C).$$
3. *Distributive Laws:* For all sets, A, B, and C,
$$\text{(a) } A \cup (B \cap C) = (A \cup B) \cap (A \cup C) \quad \text{and}$$
$$\text{(b) } A \cap (B \cup C) = (A \cap B) \cup (A \cap C).$$
4. *Identity Laws:* For all sets A,
$$\text{(a) } A \cup \emptyset = A \quad \text{and} \quad \text{(b) } A \cap U = A.$$
5. *Complement Laws:*
$$\text{(a) } A \cup A^c = U \quad \text{and} \quad \text{(b) } A \cap A^c = \emptyset.$$
6. *Double Complement Law:* For all sets A,
$$(A^c)^c = A.$$
7. *Idempotent Laws:* For all sets A,
$$\text{(a) } A \cup A = A \quad \text{and} \quad \text{(b) } A \cap A = A.$$
8. *Universal Bound Laws:* For all sets A,
$$\text{(a) } A \cup U = U \quad \text{and} \quad \text{(b) } A \cap \emptyset = \emptyset.$$
9. *De Morgan's Laws:* For all sets A and B,
$$\text{(a) } (A \cup B)^c = A^c \cap B^c \quad \text{and} \quad \text{(b) } (A \cap B)^c = A^c \cup B^c.$$
10. *Absorption Laws:* For all sets A and B,
$$\text{(a) } A \cup (A \cap B) = A \quad \text{and} \quad \text{(b) } A \cap (A \cup B) = A.$$
11. *Complements of U and $\emptyset$:*
$$\text{(a) } U^c = \emptyset \quad \text{and} \quad \text{(b) } \emptyset^c = U.$$
12. *Set Difference Law:* For all sets A and B,
$$A - B = A \cap B^c.$$

Functions

• Definition

An **(n-place) Boolean function** f is a function whose domain is the set of all ordered n-tuples of 0's and 1's and whose co-domain is the set $\{0, 1\}$. More formally, the domain of a Boolean function can be described as the Cartesian product of n copies of the set $\{0, 1\}$, which is denoted $\{0, 1\}^n$. Thus $f: \{0, 1\}^n \to \{0, 1\}$.

Convolution

- Result of convolution between two functions is either a trapezoid (if rectangles/boxes are not identical) or a triangle (if identical rectangles)
- Two functions reach the peak of convolution when they perfectly overlap
- The convolution reaches 0 when the functions don't overlap anymore
- A function that is constant in a given interval and 0 elsewhere is called box function (or rectangle)
- $(f*f)(n)$: result is a triangle function

Arithmetic Sequences

- adding constant to its predecessor : $a_n = a_{n-1} + d$

$= a_0 + d \cdot n$

Geometric Sequences

- multiplying its predecessor by a constant : $a_n = a_{n-1} \cdot r$

$= a_0 \cdot r^n$

Sum of geometric sequence / Proof

$$\sum_{i=0}^{n} r^i = \frac{r^{n+1} - 1}{r - 1}$$ if $r=2$, $2^0, 2^1, 2^2, 2^3 \dots 2^n$

$$P(n) = \sum_{i=0}^{n} r^i = \frac{r^{n+1} - 1}{r - 1}$$ where $r \neq 1$ and $n \geq 0$

1) Base Case : $P(0) = \sum_{i=0}^{0} r^i = \frac{r^1 - 1}{r - 1}$

$= r^0 = \frac{r-1}{r-1} = 1$ TRUE

2) Inductive Hypothesis : $P(k) \rightarrow P(k+1)$ Inductive Hypothesis

ASSUME $P(k) = \sum_{i=0}^{k} r^i = \frac{r^{k+1} - 1}{r - 1}$ $n=k$

PROVE $P(k+1) = \sum_{i=0}^{k+1} r^i = \sum_{i=0}^{k} r^i + r^{k+1}$ $n=k+1$

$$= \underbrace{\sum_{i=0}^{k} r^i}_{\text{↑}} + \underbrace{r^{k+1}}_{\text{↑}}$$

$$= \frac{r^{k+1}-1}{r-1} + \frac{r^{k+1}(r-1)}{r-1}$$

$$= \frac{r^{k+1}-1}{r-1} + \frac{r^k r^1 (r-1)}{r-1} = \frac{r^{k+1}-1}{r-1} + \frac{r^k r^2 - r^k r^1}{r-1}$$

$$= \frac{r^{k+1}-1}{r-1} + \frac{r^{k+2}-r^{k+1}}{r-1} = \frac{r^{k+1}-1+r^{k+2}-r^{k+1}}{r-1}$$

$$= \frac{r^{k+2}-1}{r-1} \quad \square$$

Application: - onset of a pandemic
- memory size of a tree data structure

Recursive Function

- one or more base case
- one or more recursive calls

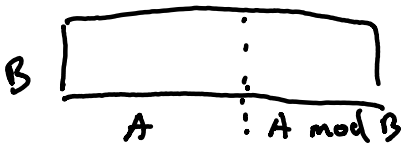
$$n! = \begin{cases} 1 & \text{if } n=0 \\ n \cdot (n-1)! & \text{if } n>0 \end{cases}$$

Euclidean Algorithm

$$gcd(A,B) = \begin{cases} A & \text{if } B = 0 \\ gcd(B, A \bmod B) & \text{if } B > 0 \end{cases} \quad | \quad A > B \geq 0$$

Finding the gcd(A,B) means finding the largest square that can tile a AxB rectangle.

Double the square size until it's not fully tileable



→ The largest square that can tile AxB must also tile Bx (A mod B), so gcd(AB) = gcd (B, A mod B)

Recursion vs. Iteration

- Any recursive algorithm can be converted into an iterative one and vice versa (iterative to recursive is more difficult)

Recursion

⊕ Easier to implement for complex problems

⊕ More elegant and easier to interpret

⊕ More concise (fewer lines)

⊖ More expensive in processor time and memory space

Induction

- The principle of mathematical induction can only be used to prove statements whose domain is a subset of the integer numbers (because of K+1)

1) Base Case: Show that Base Case is true by giving the example of the first element

   P(K) is true for K = first element (b)
   → P(b) is true

2) Inductive Step:

   2.1) Inductive hypothesis: Assume P(K) true for any $K \geq b$

   2.2) Show whether P(K) → P(K+1) is true: Put in K+1 for K

3) Conclusion: if P(b) is true and P(k) → P(k+1) is true, conclude that P(n) is true for all $n \geq b$

Property to prove

$$P(n): \text{“} \sum_{i=1}^{n} i = 1+2+3+4 \dots +n = \frac{n(n+1)}{2} \text{”}$$

Proof by induction

1) Base case:

$b=1$, $P(1)$ $\sum_{i=1}^{1} i = \frac{1(1+1)}{2} = \frac{2}{2} = 1$ TRUE

2) Inductive step

Assume $P(k)$ is TRUE, then $P(k+1)$ is TRUE

→ Assume $P(k) \rightarrow P(k+1)$ **Inductive Hypothesis**

$$\sum_{i=1}^{k} i = 1+2+3+4 \dots k = \frac{k(k+1)}{2} \qquad n=k$$

$$\sum_{i=1}^{k+1} i = \underbrace{1+2+3 \dots +k}_{\sum_{i=1}^{k} i = \frac{k(k+1)}{2}} + (k+1) = \frac{(k+1)(k+2)}{2} \qquad n=k+1$$

PATTERN

$$\sum_{i=1}^{k+1} i = \underbrace{\frac{k(k+1)}{2} + (k+1)}_{*} = \frac{(k+1)(k+2)}{2}$$

$\boxed{n = k+1}$

* $n=k$, we assume it's true

$$= \frac{k(k+1)}{2} + \frac{2(k+1)}{2}$$

$$= \frac{(k+1)(k+2)}{2} \qquad = \frac{(k+1)(k+2)}{2}$$

TRUE

3) Conclusion

So, we can conclude $P(n)$ is true $\forall n \geq b$ because $P(b)$ is true and $P(k) \rightarrow P(k+1)$ is true

Property to prove

$$P(n): \text{“} \sum_{i=1}^{n} 2i-1 = 1+3+5+7 \ldots +(2n-1) = n^2$$

Proof by induction

1) Base case:

$b=1$, $P(1)$ $\underbrace{\sum_{i=1}^{1} 2i-1 = 1^2}_{\text{TRUE}}$ $\Rightarrow P(1)$ is TRUE

2) Inductive step

Assume $P(k)$ is TRUE, then $P(k+1)$ is TRUE

$\rightarrow$ Assume $P(k) \rightarrow P(k+1)$ **Inductive Hypothesis**

$$\sum_{i=1}^{k} 2i-1 = 1+3+5 \ldots +(2k-1) = k^2 \qquad n=k$$

$$\sum_{i=1}^{k+1} 2i-1 = 1+3+5 \ldots +\underbrace{(2(k)-1)}_{+(2k-1)} + (2(k+1)-1) = (k+1)^2 \qquad n=k+1$$

$$\sum_{i=1}^{k} 2i-1 = k^2$$

PATTERN

$$\sum_{i=1}^{k+1} 2i-1 = k^2 + \left(2(k+1)-1\right) = (k+1)^2$$

$n = k+1$

$$k^2 + (2k+1) = k^2 + 2k + 1$$

TRUE

3) Conclusion

So, we can conclude $P(n)$ is true $\forall n \geq b$ because $P(b)$ is true and $P(k) \rightarrow P(k+1)$ is true

Property to prove

$P(n)$: " $3 \mid n^3 - n \quad \forall\, n \geq 0$

→ 3 divides $n^3 - n$ for every $n \geq 0$

→ $n^3 - n$ is a multiple of 3 $(0, 3, 6, 9, 12 \ldots)$

Proof by induction

1) Base case:

$b = 0$, $P(0)$ $\quad 3 \mid 0^3 - 0 = \underbrace{3 \mid 0}_{\text{TRUE}}$ $\quad \Rightarrow P(0)$ is TRUE

2) Inductive step

Assume $P(k)$ is TRUE, then $P(k+1)$ is TRUE

→ Assume $P(k) \rightarrow P(k+1)$ **Inductive Hypothesis**

$3 \mid k^3 - k$ is TRUE (ASSUME) $\quad n = k$

$3 \mid (k+1)^3 - (k+1)$ is TRUE ⟵ (PROVE) $\quad n = k+1$

$= (k+1)^3 - (k+1)$

$= (k+1)(k+1)(k+1) - (k+1)$

$= (k^2+2k+1)(k+1) - (k+1)$

$= (k^3+2k^2+k+k^2+2k+1) - (k+1)$

$= (k^3+3k^2+3k+1) - (k+1)$

$= k^3+3k^2+3k+1-k-1$

$= k^3+3k^2+3k-k$

$= \underbrace{(k^3-k)} + \underbrace{3k^2+3k}$

$3 \mid k^3-k$ is TRUE (hypothesis)

$3(k^2+k)$ → k is an integer; 3 * an integer is a multiple of 3 by definition

$=$ multiple of 3 $+$ multiple of 3 $=$ multiple of 3 $\Rightarrow P(k+1)$ is TRUE

3) Conclusion

So, we can conclude $P(n)$ is true $\forall n \geq 0$ because $P(b)$ is true and $P(k) \rightarrow P(k+1)$ is true

Property to prove

$P(n)$: " $2n+1 < 2^n$ $\quad \forall\ n \geq 3$ "

Proof by induction

1) Base case:

$b=3$, $P(3)$ $\quad 2\cdot 3+1 < 2^3 = \underbrace{7<8}_{\text{TRUE}}$ $\quad \Rightarrow P(3)$ is TRUE

2) Inductive step

Assume $P(k)$ is TRUE, then $P(k+1)$ is TRUE

→ Assume $P(k) \rightarrow P(k+1)$ [Inductive Hypothesis]

$2k+1 < 2^k$ is TRUE (ASSUME) $\quad [n=k]$

$2(k+1)+1 < 2^{(k+1)}$ is TRUE ⇐ (PROVE) $\quad [n=k+1]$

$2(k+1)+1$

$2k+2+1$

$2k+1+2$

$< 2^k + 2$

$2 < 2^k$ $\quad (\forall\ k \geq 3)$

$< 2^k + 2^k$

$1\cdot 2^k + 1\cdot 2^k$

$(1+1)2^k$

$2\cdot 2^k$

$2^1 \cdot 2^k = 2^k \cdot 2^1 = 2^{k+1}$

$< 2^{k+1}$

⇒ proven

$2(k+1)+1 < 2^{k+1}$

3) Conclusion

So, we can conclude $P(n)$ is true $\forall n \geq 3$ because $P(b)$ is true and $P(k) \rightarrow P(k+1)$ is true

More Proofs

- Trominoes
- Tower of Hanoi

Strong mathematical Induction — 2nd order recurrence relation

- makes stronger hypothesis, namely
  - Base case may contain proofs for several initial values for computing the base case ①
  - Inductive step assumes truth of P(k) not only for an arbitrary $k \geq b$ (larger integer than base case) but for all values of k ②

P(n) = " Every integer $n > 1$ is either a prime number or can be written as the product of two prime numbers "

1) Base case P(2) = 2 is prime is TRUE
P(3) = 3 is prime is TRUE

① we have 2 init values

2) Inductive hypothesis: Case 1: k+1 is prime, then P(k+1) is true
Case 2: k+1 is not prime, then $a \cdot b = k+1$ with a & b smaller than k+1, then k+1 can not be prime because k+1 is then divisible by at least a & b

Function in Computer Programming

- piece of code that generates some output in correlation to a given input

⊕ enhances readability of computer program

⊕ easy to understand and makes it reusable

⊕ reduce the complexity, give it modular structure

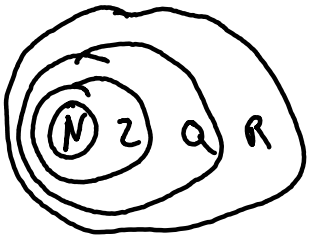
⊕ individually testable

Set theory

- set is a <u>distinct</u> collection of objects

- $x \in S$, x is in S / S contains x

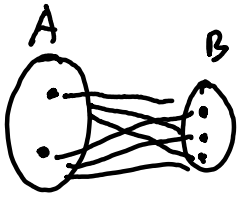
- $S = \{A, B, C, 1, 2, 3, \{-2, -10\}\}$ OR $S = \{x \in R \mid -1 < x \leq 0\}$



$\emptyset = \{\}$ = empty set

- $(a,b) = (c,d)$ iff $(a=c)$ and $(b=d)$

- Cartesian Product (ordered pairs) $\{(x,y) \mid (x \in A) \wedge (y \in B)\}$



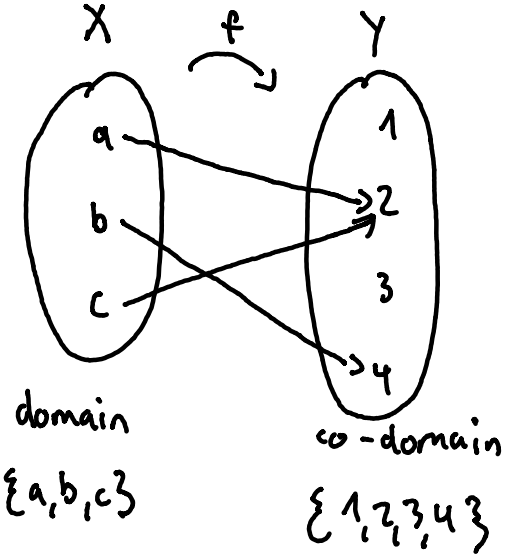
$(A_1, B_1), (A_1, B_2), (A_1, B_3),$
$(A_2, B_1), (A_2, B_2), (A_2, B_3)$

Functions

$f: X \to Y$ $\quad f(x) = y$

X domain
Y co-domain

- Every element of X is related to one element in Y
  - There can be elements in Y that are not function of any input element of X (lonely elements)
  - Different inputs of X can map to the same element in Y



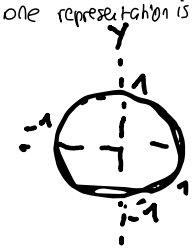
- $f(a) = 2$ / $f(b) = 4$ / $f(c) = 2$
- range of f? $\{2,4\}$ (reachable in Y)
- Inverse of $2 = \{a, c\}$
- Inverse of $1 = \emptyset$
- Set of ordered pairs: $\{(a,2),(b,4),(c,2)\}$

$f\left(\frac{m}{n}\right) = m$ NO function

$f\left(\frac{3}{1}\right) = 3 = f\left(\frac{1}{2}\right) = 1$ $(\mathbb{Q} \to \mathbb{Z})$

more than one representation is possible

$f(x) = y \mid x^2 + y^2 = 1$ NO function



$(\mathbb{R} \to \mathbb{R})$

$f(0) = 1 = 0^2 + 1^2 = 1$ ✓

$f(0) = -1 = 0^2 + (-1)^2 = 1$ ✓ 2 Y-Values

Equality

Let $f: X \to Y$ and $g: X \to Y$ be functions, then $f = g$ iff

$f(x) = g(x) \quad \forall x \in X$ (Every X value has to produce the exact same output)

Notable Functions

- The identity function maps each element $x \in X$ to itself $\forall x \in X$

- The exponential function for positive real numbers $b \neq 1$

$$\exp_b(x) = b^x$$

$$b^u b^v = b^{u+v} \qquad \frac{b^u}{b^v} = b^{u-v}$$

$$(b^u)^v = b^{uv} \qquad (bc)^u = b^u c^u$$

- The logarithmic function for positive real numbers $b \neq 1$

$$\log_b(x) = y \iff b^y = x$$

$$\log_b(xy) = \log_b x + \log_b y \qquad \log_b(x^a) = a \log_b x$$

$$\log_b\left(\frac{x}{y}\right) = \log_b x - \log_b y \qquad \log_c x = \frac{\log_b x}{\log_b c}$$

- The boolean function

example: AND function $\underbrace{\{0,1\}^2}_{} \to \{0,1\}$

$$\{(0,0),(0,1),(1,0),(1,1)\}$$

→ will map the set of pairs to a set of bits

- The Hamming Distance function is a metric for comparing two binary strings of equal length. It returns the number of bit positions in which the two strings are different

a = 11011100
b = 11110110
H(a,b) = 3

→ bitwise XOR (true if different) and then summing up

- The Image Function. An 8-bit grayscale image can be represented as function: $I: [0, W-1] \times [0, H-1] \to [0, 255]$ (width, height)

→ every pixel holds a value between 0 to $2^8 - 1$

0 represents black (low means dark)
255 represents white (high means bright)

For colors: 3 dimensions Red, Green, Blue
→ Every pixel holds 3 values

$I: [0, W-1] \times [0, H-1] \to [0, 255]^3$ (width, height)

Convolution

- Image Filtering is a function that takes an image as input and outputs a new filtered image

  low-pass filter: smooth / blur the image

  high-pass filter: highlight contours of image (edge detection)

1D Convolution

- Convolution replaces each pixel value with a weighted sum of all the neighborhood values

values	...	2	5	1	7	9	...	(considers two left & two right)
weights 1		1	1	1	1	1		(equal weights)
weights 2		1	2	4	2	1		(center has more weight)

↑

$f(n), g(n), y(n) : \mathbb{Z} \to \mathbb{R} \qquad \forall n \in \mathbb{Z}$

$$y(n) = (f * g)(n) = \sum_{k=-\infty}^{+\infty} f(k)\, g(n-k)$$

(* = convolution) — two functions create a third one

f is the input function

g is the Kernel or filter function

y is the convolution function

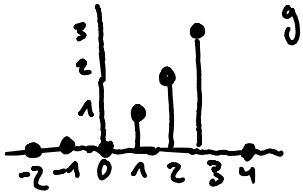
Length of $y(n) = f(n) * g(n) = [\text{len}(f(n)) + \text{len}(g(n))] - 1$ (n.elements)

$$y(n) = (f * g)(n) = \sum_{k=-\infty}^{+\infty} f(k)\, g(n-k)$$

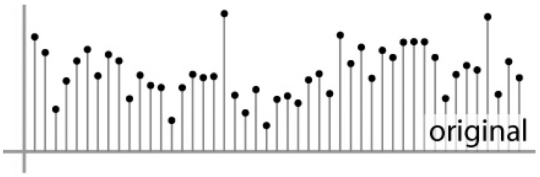
$f(k)\,g(k)$:

$f(k)$

$g(k)$



point-wise multiplication



$f(0)\,g(0) = 1 * 0 = 0$

$f(1)\,g(1) = 1 * 1 = 1$

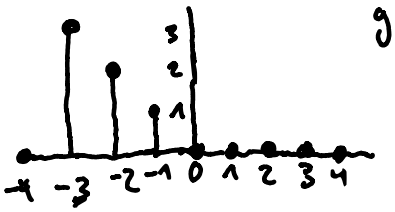
$f(2)\,g(2) = 1 * 2 = 2$

$f(3)\,g(3) = 0 * 3 = 0$

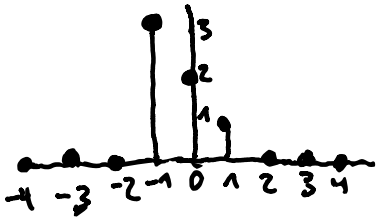
$\approx$

$g(n-k)$: $-k$ means flip at 0 axis

$g(-k)$



$+n$ means shift



$g(2-k)$

$n=2$

$$y(n) = (f * g)(n) = \sum_{k=-\infty}^{+\infty} f(k)\, g(n-k)$$

commutative
$g \circ f \equiv f \circ g$

$y(n) = 0$ when $n \leq 0$ mmm $(x \cdot 0 = 0)$

$y(1) = f(0)\, g(1-0) = 1 \cdot 1 = 1$

$y(2) = f(0)\, g(2-0) + f(1)\, g(2-1)$

$= \underbrace{1 \cdot 2}_{f(0) \cdot \text{flip} + 2} + \underbrace{1 \cdot 1}_{f(1) \cdot \text{flip} + 2} = 3$

$y(3) = f(0)\, g(3-0) + f(1)\, g(3-1) + f(2)\, g(3-2)$

$= \underbrace{1 \cdot 3}_{f(0) \cdot \text{flip} + 3} + \underbrace{1 \cdot 2}_{f(1) \cdot \text{flip} + 3} + \underbrace{1 \cdot 1}_{f(2) \cdot \text{flip} + 3} = 6$

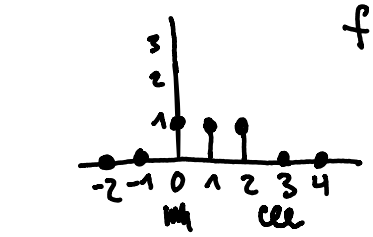
$y(4) = f(1)\, g(4-1) + f(2)\, g(4-2)$

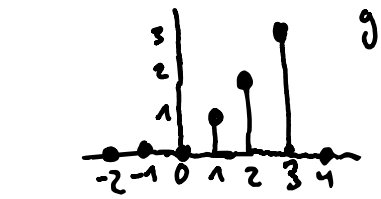
$= 1 \cdot 3 + 1 \cdot 2 = 5$

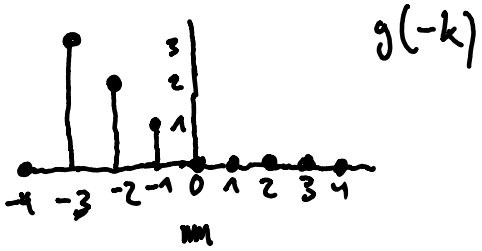
$y(5) = f(2)\, g(5-2)$

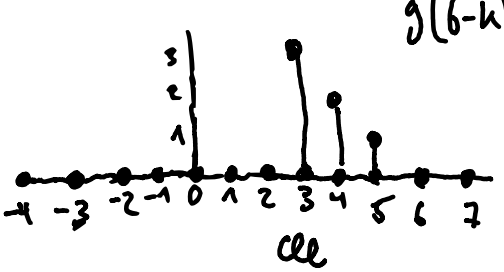
$= 1 \cdot 3 = 3$

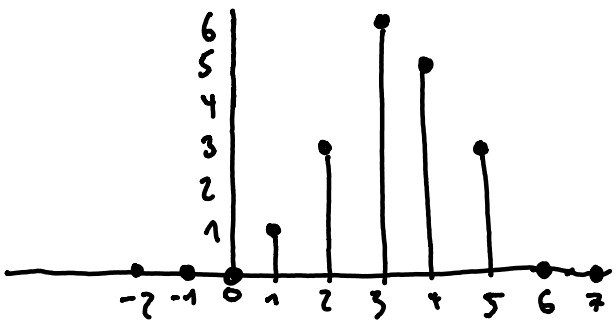
$y(n) = 0$ when $n \geq 6$ cee $(x \cdot 0 = 0)$





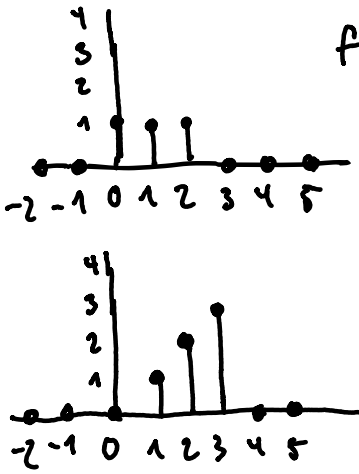






$y(n)$

The Convolution Table → alternative and quicker way to manually compute the convolution between two vectors

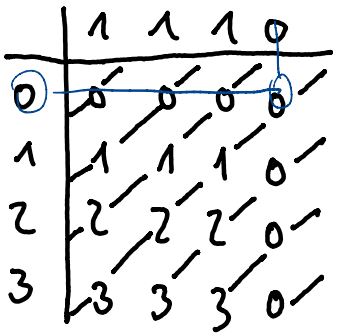


$\rightarrow \bar{f} = [1,1,1,0]$

$\bar{g} = [0,1,2,3]$

(multiply)

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

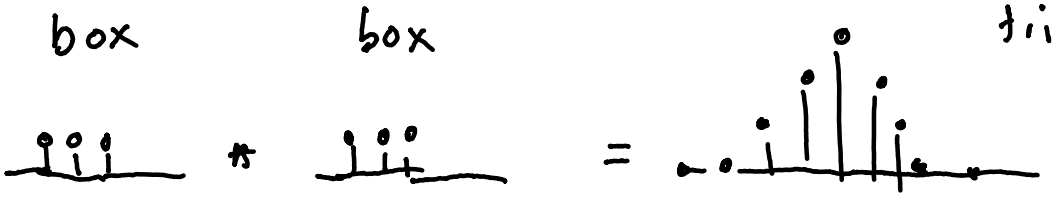


sum diagonals $[0,1,3,6,5,3,0]$

$\bar{y} = \bar{f} * \bar{g} = [0,1,3,6,5,3,0]$

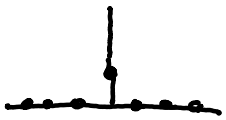
→ read $n \geq x$ & $n \leq x$ from function

box * box = triangle



Convolution with a Dirac function

$$\delta(n) = \begin{cases} 1 & n=0 \\ 0 & \text{elsewhere} \end{cases}$$



The result of the convolution of any function $f(n)$ with the Dirac function is the function itself

$$(f * \delta)(n) = f(n)$$

because δ flipped results in δ itself, then overlapping means $1 \cdot f(n) = f(n)$

Properties of Convolution

$f * g = g * f$ Commutative, only possible through the flipping

$((f * g) * h = f * (g * h)$

$(\lambda f) * g = f * (\lambda g) = \lambda(f * g)$ (λ: scalar)

$f * (g+h) = (f * g) + (f * h)$

$f * \delta = f$

$$(\ell * g)(n) = (g * \ell)(n) \Rightarrow \sum_{k=-\infty}^{+\infty} \ell(k)\, g(n-k) = \sum_{k=-\infty}^{\infty} g(k)\, \ell(n-k)$$

2D Convolution

1D Convolution

$f(n), g(n), y(n) : \mathbb{Z} \to \mathbb{R} \qquad \forall n \in \mathbb{Z}$

$$y(n) = (f * g)(n) = \sum_{k=-\infty}^{+\infty} f(k)\, g(n-k)$$

2D Convolution

$F(i,j), G(i,j), Y(i,j) : \mathbb{Z}^2 \to \mathbb{R} \qquad \forall (i,j) \in \mathbb{Z}^2$

$$Y(i,j) = (F * G)(i,j) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} F(u,v)\, G(i-u, j-v)$$

$F(i,j)$ referred as input image function

$G(i,j)$ referred as Kernel or filter function

- We assume the image function to have value 0 for every pixel outside of the image boundaries

- Flip = flip to left and flip upwards

Boundary Issues

- Pad the image with 0s outside the boundaries and start the filter center at top left

filter image padding with 0

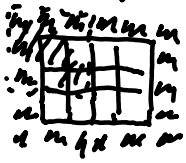


Image smoothing = low-pass = constant value filter

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Edge detection = high pass = weight values depending on shape to detect

0	1	0
1	1	1
0	1	0

→ detects circle shapes

Convolutional Neural Network (CNN)

- consists of a series of convolutional layers that convolve the original image with different filters, each one dedicated to detect different features (lines, circles, faces, eyes...)
- These filters are trained by backpropagation
- The output can be classified to a certain probability to a certain class

Lower Layer Filters: simple features (lines, curves)

Higher Layer Filters: complex features (eyes, faces, textures)

Relations

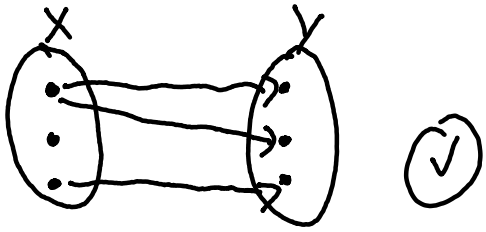
- more general than functions

Relations

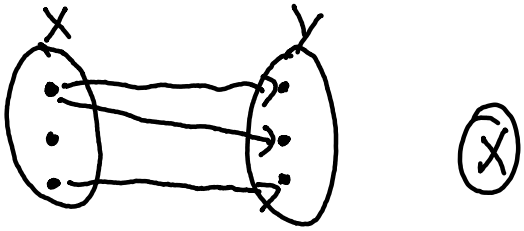
- Same element of the domain may be related to multiple elements in the co-domain

Functions

- One element in the domain maps to exactly one element in the co-domain



✓



✗

$x R y$ = x is related to y

Definition

A relation R from a set A (domain) to a set B (co-domain) is defined as a subset of the Cartesian product $A \times B$

$$R = \{ (x,y) \in A \times B \mid x R y \}$$

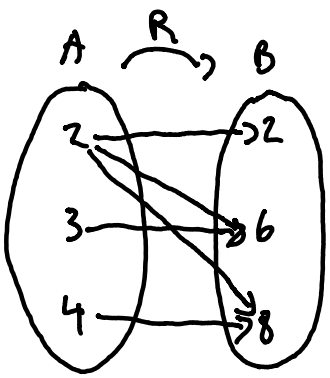
Inverse

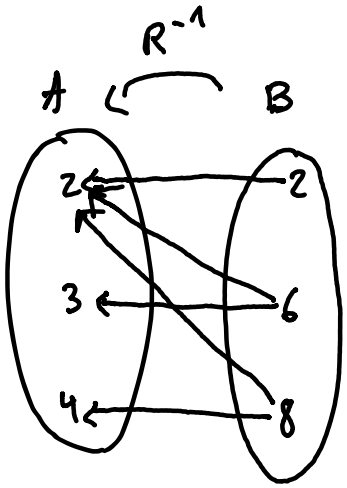
$$R^{-1} = \{ (y,x) \in B \times A \mid (x,y) \in R \}$$

For all $x \in A$ and $y \in B$, $(y,x) \in R^{-1} \iff (x,y) \in R$

$$x R y \iff x \mid y$$

$x|y$: x divides y, y/x is an integer





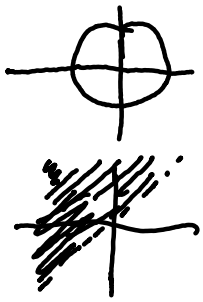
R^{-1} : y is a multiple of x

Relations on the same set

R is subset of $R \times R$

$R_1 = \{ (x,y) \in R^2 \mid x^2 + y = 1 \}$

$R_2 = \{ (x,y) \in R^2 \mid x < y \}$



Relations on a Finite set

- Representation via a directed graph

$A = \{3,4,5,6,7,8\}$ $R = \{ (x,y) \in A \mid x R y \iff 2 \mid (x-y) \}$

→ x and y are related iff their difference is a multiple of 2

→ Congruence modulo 2 Relation

$= \{ (3,3), (3,5), (3,7), \vdots (4,4), (4,6), (4,8), \vdots (5,5), (5,3), (5,7), \vdots (6,6) \ldots \}$

Reflexivity, Symmetry, Transitivity

R is reflexive, iff $\forall x \in A, xRx$ and it is not reflexive when there is at least an element x in A s.t. $x \not R x$

→ every element loops to itself

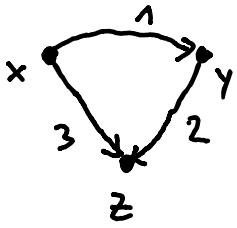
R is symmetric, iff $\forall x, y \in A$, if xRy then yRx (*)

(x,y) & (x,y) needs to be in R

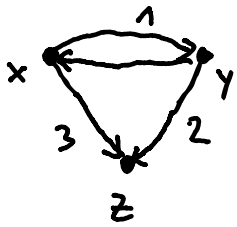
x,y,z don't have to be distinct (x could 2 and y,z = 5)

R is transitive, iff $\forall x, y, z \in A$, if xRy & yRz then xRz (*)

(1) (2) (3)



then



2. R is **not symmetric** ⇔ there are elements x and y in A such that $x\,R\,y$ but $y\,R\,x$

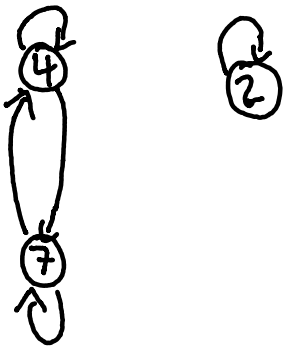
3. R is **not transitive** ⇔ there are elements x, y and z in A such that $x\,R\,y$ and $y\,R\,z$ but $x\,R\,z$

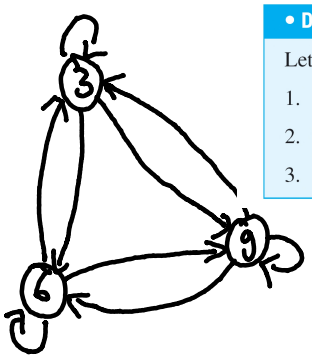
(*) elements are not required to be distinct

reflexive & symmetric & transitive = equivalence relation

Congruence Modulo 3 Relation

$A = \{2,3,4,6,7,9\}$ $\quad R = \{(x,y) \in A \mid xRy \Leftrightarrow 3 \mid (x-y)\}$





Definition

Let R be a relation on a set A.

1. R is **reflexive** if, and only if, for all $x \in A$, $x\,R\,x$. each element relates to itself
2. R is **symmetric** if, and only if, for all $x, y \in A$, *if* $x\,R\,y$ then $y\,R\,x$.
3. R is **transitive** if, and only if, for all $x, y, z \in A$, *if* $x\,R\,y$ and $y\,R\,z$ then $x\,R\,z$.

reflexive ? YES (loops everywhere)

symmetric ? YES everywhere

transitive ? YES there is no arrow from A to C

Transitive Closure of a Relation

- To make a relation transitive we need to add ordered pairs
- Obtained by adding least number of ordered pairs to ensure transitivity

$A = \{0, 1, 2, 3\}$ $R = \{(0,1), (1,2), (2,3)\}$

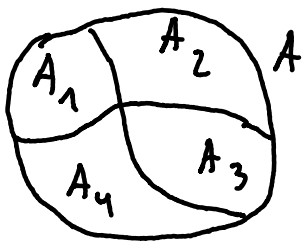
R^+ contains $\{(0,1), (0,2), (0,3), (1,2), (1,3), (2,3)\}$

Equivalence Relations

- reflexive, symmetric & transitive. Any congruence modulo i relation is an equivalence relation

Relation induced by Partition

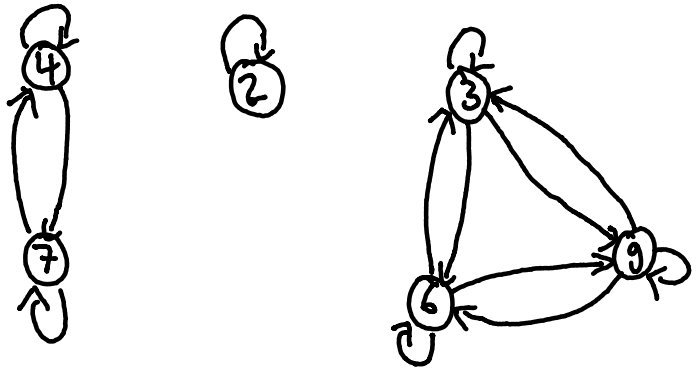
A partition of A is a collection of mutually disjoint subsets A_i whose union is A



The relation induced by partition is a relation where all elements within each subset A_i are related to one another and themselves

→ Equivalence Relation

Example: Congruence modulo 3 Relation



Transitivity: 7R4 & 4R7 ⇒ 7R7 true

Equivalence Classes

Given:

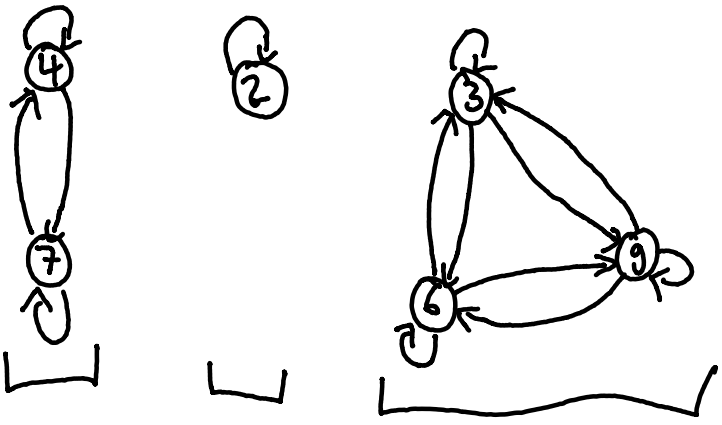
- equivalence relation on set A
- any particular element a in A

Then:

- subset [a] of all elements related to a under R is called equivalence class of a
- Any element of an equivalent class is called class Representative

$$[a] = \{ x \in A \mid xRa \}$$

Example: Congruence modulo 3 Relation



(Distinct) equivalence classes

→ Distinct equivalence classes of A form a partition of A

→ Any element of an equivalence class is called class representative

Theorem 8.4.1 Modular Equivalences

Let a, b, and n be any integers and suppose $n > 1$. The following statements are all equivalent:

1. $n \mid (a - b)$
2. $a \equiv b \pmod{n}$
3. $a = b + kn$ for some integer k
4. a and b have the same (nonnegative) remainder when divided by n
5. $a \bmod n = b \bmod n$

Properties of Congruence Modulo n

1. $(a + b) \bmod n = [(a \bmod n) + (b \bmod n)] \bmod n$
2. $(a - b) \bmod n = [(a \bmod n) - (b \bmod n)] \bmod n$
3. $(a \cdot b) \bmod n = [(a \bmod n) \cdot (b \bmod n)] \bmod n$
4. $(a^m) \bmod n = (a \bmod n)^m \bmod n$

So, the fundamental fact about "Congruence Module n" is that if you first perform an addiction, subtraction, multiplication, of integers and then **reduce the result modulo n**, you obtain the same result as when the operation is performed on the reduced version of the integers!

a mod b = a if a < b (e.g. 17 mod 55 = 17)

Cryptography uses large numbers (hundreds/thousands of bits).
When modular arithmetic is performed with large numbers, computations are facilitated by using two properties of exponents:

$x^{2a} = (x^a)^2$ for all real numbers x and a with $x \geq 0$.

$x^{a+b} = x^a x^b$ for all real numbers x, a, and b with $x \geq 0$.

Example:

$$x^4 \bmod n = (x^2)^2 \bmod n = (x^2 \bmod n)^2 \bmod n$$

The **inverse** of an integer a **modulo** n **exists if and only if** a and n are **relatively prime**.

Modular Arithmetic

Congruence Modulo n

Let m and n be integers and d be a positive integer.

Then, m is congruent to n modulo d

$$m \equiv n \pmod{d} \iff d \mid (m-n)$$

All equivalences

$m \equiv n \pmod{d}$	$12 \equiv 7 \pmod{5}$
$m \pmod{d} = n \pmod{d}$	$12 \bmod 5 = 7 \bmod 5$
$d \mid (m-n)$	$5 \mid (12-7) = (12-7)/5 = \text{int}$
$m = n + kd$ for some integer k	$12 = 7 + 1 \cdot 5 \quad k = 1$
m and n have the same remainder when divided by d	$12/5 = 2 \quad R\,2$ $7/5 = 1 \quad R\,2$ ✓

Properties

$$(a+b) \bmod n = [(a \bmod n) + (b \bmod n)] \bmod n$$

$$(a-b) \bmod n = [(a \bmod n) - (b \bmod n)] \bmod n$$

$$(a \cdot b) \bmod n = [(a \bmod n) \cdot (b \bmod n)] \bmod n$$

$$(a^b) \bmod n = (a \bmod n)^b \bmod n$$

Modular Arithmetic

$x^{2a} = (x^a)^2$ for all real numbers x and a with $x \geq 0$

$x^{a+b} = x^a x^b$ for all real numbers x, a, and b with $x \geq 0$

$\rightarrow x^4 \bmod n = (x^2)^2 \bmod n = (x^2 \bmod n)^2 \bmod n$

Examples

$$
\begin{aligned}
144^4 \bmod 713 &= (144^2)^2 \bmod 713 \\
&= (144^2 \bmod 713)^2 \bmod 713 \\
&= (20736 \bmod 713)^2 \bmod 713 \\
&= 59^2 \bmod 713 \\
&= 3481 \bmod 713 \\
&= 629
\end{aligned}
$$

$$
\begin{aligned}
8^3 \bmod 55 &= [(8^2 \bmod 55) \cdot (8 \bmod 55)] \bmod 55 \\
&= [(64 \bmod 55) \cdot 8 \quad] \bmod 55 \\
&= [9 \cdot 8] \bmod 55 \\
&= 72 \bmod 55 \\
&= 17
\end{aligned}
$$

$(55+26) \bmod 4 = 1$ (formula)

$(55 \bmod 4) + (26 \bmod 4) \overset{\bmod 4}{=} (5) \bmod 4 = 1$

$3 \qquad 2$

$(3+2) \bmod 4$ → When we see that each term is smaller than the modulo, we stop. If we still have larger numbers, then we apply the same property again

calculated with the method of Successive Divisions

$17^{27} \bmod 55$

$\vdots$

$\dots\rightarrow 27 = 16 + 8 + 2 + 1$, thus, $17^{27} \bmod 55 = \left(17^{16} \cdot 17^{8} \cdot 17^{2} \cdot 17^{1}\right) \bmod 55$

$= 2^4 + 2^3 + 2^1 + 2^0$

↳ Every number can be written as a sum of the powers of 2!

↑ We calculate from right to left

$\Rightarrow 17^2 \bmod 55 = 14$

$17^4 \bmod 55 = (17^2)^2 \bmod 55 = (\underline{17^2 \bmod 55})^2 \bmod 55$

$= (14)^2 \bmod 55 = 31$

$\Rightarrow 17^8 \bmod 55 = (17^4)^2 \bmod 55 = (\underline{17^4 \bmod 55})^2 \bmod 55$

$= (31)^2 \bmod 55 = 26$

$\Rightarrow 17^{16} \bmod 55 = (17^8)^2 \bmod 55 = (\underline{17^4 \bmod 55})^2 \bmod 55$

$= (26)^2 \bmod 55 = 16$

$17^{27} \bmod 55 = (16 \cdot 26 \cdot 14 \cdot 17) \bmod 55 = 8$

Inverse Modulo n

Basic Arithmetic: Inverse of $a \neq 0$ is $x = 1/a$

because $a \cdot x = 1$

$x = 1/5$ is the inverse of $a = 5$ | $5 \cdot 1/5 = 1$

Modular Arithmetic: Inverse of integer a is integer x

$a \cdot x \equiv 1 \pmod n \Leftrightarrow (a \cdot x) \bmod n = 1 \bmod n$

$(a \cdot x) \bmod n = 1$

The inverse of an integer a modulo n exists iff a and n are relatively prime

$\gcd(3,7) = 1$ → exists (relatively prime)

$\gcd(2,6) = 2$ → does not exist

useful for cryptography

Find Inverse Modulo : Bezout's theorem + Euclidean Algorithm

Bezout's theorem

If a and b are positive integers, then there exist integer s and t such that $\gcd(a,b) = s \cdot a + t \cdot b$

Euclidean Algorithm

Let A and B be two integers $A > B \geq 0$. Euclid's algorithm is defined by the following recursive function

$$\gcd(A,B) = \begin{cases} A & \text{if } B = 0 \\ \gcd(B, A \bmod B) & \text{if } B > 0 \end{cases} \leftarrow \text{Base Case}$$

A B

$\gcd(35,27) = S \cdot A + T \cdot B$

$= \gcd(27, 35 \bmod 27) = \gcd(27,8)$

$= \gcd(8, 27 \bmod 8) = \gcd(8,3)$

$= \gcd(3, 8 \bmod 3) = \gcd(3,2)$

$= \gcd(2, 3 \bmod 2) = \gcd(2,1)$

$= 1$ (relatively prime)

Standard

$35 = 1 \cdot 27 + 8 \rightarrow 8 = 35 - 1 \cdot 27$ ③

$27 = 3 \cdot 8 + 3 \rightarrow 3 = 27 - 3 \cdot 8$ ②

$8 = 2 \cdot 3 + 2 \rightarrow 2 = 8 - 2 \cdot 3$ ①

$3 = 1 \cdot 2 + 1 \rightarrow 1 = 3 - 1 \cdot 2$ ⓪

Extended

$1 = 3 - 1 \cdot 2$ ⓪

$= 3 - 1(8 - 2 \cdot 3) = 3 - 8 + 2 \cdot 3 = 3 \cdot 3 - 8$ ①

$= 3 \cdot (27 - 3 \cdot 8) - 8 = 3 \cdot 27 - 9 \cdot 8 - 8 = 3 \cdot 27 - 10 \cdot 8$ ②

$= 3 \cdot 27 - 10(35 - 1 \cdot 27) = 3 \cdot 27 - 10 \cdot 35 + 10 \cdot 27$ ③

$= -10 \cdot 35 + 13 \cdot 27$

$S \cdot A \quad + \cdot B$

Inverse of 27 modulo 35

$a \cdot x \equiv 1 \pmod{n}$

$27 \cdot x \equiv 1 \pmod{35}$

$13 \cdot 27 - 10 \cdot 35 = 1$

$13 \cdot 27 = 1 + 10 \cdot 35$

$(13 \cdot 27) \bmod 35 = (1 + 10 \cdot 35) \bmod 35 = 1$

$27 \cdot 13 \equiv 1 \pmod{35}$

$x = 13$

Cryptography

- involves encryption (plaintext to ciphertext) and decryption (ciphertext to plaintext)

Caesar cipher

- substituting each letter of the alphabet by the one three places farther along

$C = (M+3) \bmod 26$ (26 letters) encryption

$M = (C-3) \bmod 26$ decryption

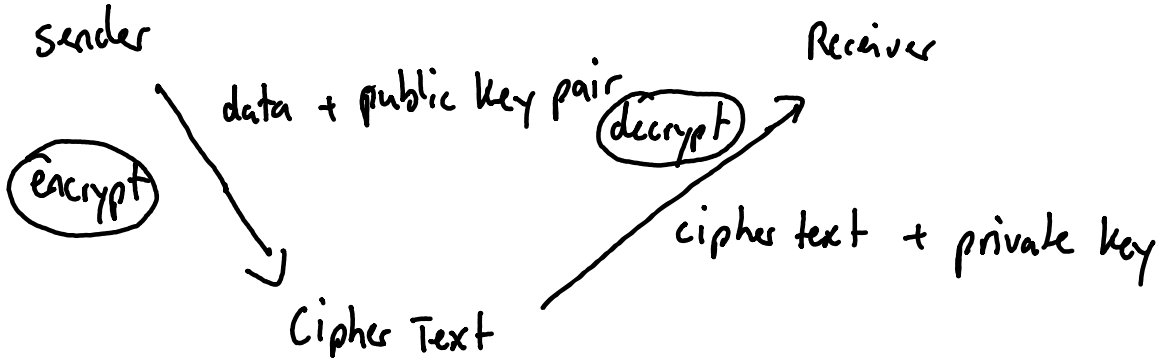
- can be broken by
  - brute-forcing all 26 shifts (we only have to test 26 shift till we find a readable text)
  - guess the shift by frequency analysis of the letters

RSA Cryptography

- RSA was one of the first public-key cryptosystems
- is asymmetric (2 keys needed, public key to encrypt, private key to decrypt)

contrast to symmetric: no secure channel needed
no need to keep public key secret

- SSL based on RSA



Encryption: $C = M^e \bmod S$

C: ciphertext
M: plaintext
e: public key 1
S: public key 2
) pair

Decryption $M = C^d \bmod S$

d: private key

$$M = (M^e \bmod S)^d \bmod S = (M^e)^d \bmod S$$

Requirements

- $M < S$

① $S = pq$ where p and q are prime numbers

② $e \cdot (p-1)(q-1)$ is relatively prime, i.e. $\gcd(e, (p-1)(q-1)) = 1$

③ $ed \equiv 1 \pmod{(p-1)(q-1)}$, i.e. d is the inverse of e modulo $(p-1)(q-1)$

Example

$p = 5$ (prime) $\quad \boxed{pq = 5}$ ① $\quad\quad 5 = 55$

$q = 11$ (prime) $\quad 5 \cdot 11 = 55$

② choose e, which is relatively prime to $(p-1)(q-1)$

$(p-1)(q-1) = (4)(10) = 40$, we take $\underline{e = 3}$

$\gcd(40, 3) = 1$ ✓

$\boxed{\rightarrow 5 = 55 \text{ and } e = 3 \text{ are the public key}}$

③ Then compute d, such that $ed \equiv 1 \pmod{(p-1)(q-1)}$

which is $ed \equiv 1 \pmod{40}$

$3 \cdot d \equiv 1 \pmod{40}$ $\quad \boxed{d = 27 \quad \text{private key}}$

$3 \cdot 27 \equiv 1 \pmod{40}$

Security

hundreds of thousand of bits

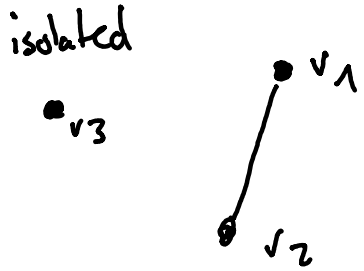
s and e are chosen to be very large integers. Thus, an attacker need to factor s into the product of pq, but this may take very long time!

However, if you know p and q, d can easily determined knowing that $ed \equiv 1 \pmod{(p-1)(q-1)}$

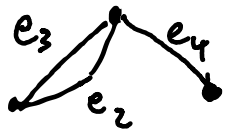
→ the security of RSA is based on the hardness of factoring: It's easy to generate two lary prime numbers p and q but it is hard to factor their product s into pq

Graphs

- graph consists of a set of vertices and a set of edges connecting pairs of vertices



v_1 & v_2 are adjacent



e_3 & e_2 are parallel edges

e_3/e_2 & e_4 are adjacent

- isolated: a vertex does not necessarily need to be an endpoint of an edge
- directed graphs: $\longrightarrow$

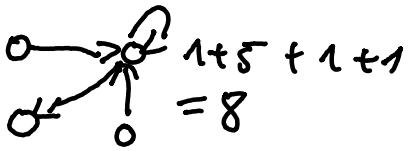
  edges have direction $e_1(v_1, v_2)$

  directed edges are called arrows and are defined as ordered pairs

- Complete Graphs are graphs where all pair combinations of vertices are connected by edges

- Degree of a vertex v is the number of edges that end or start at v

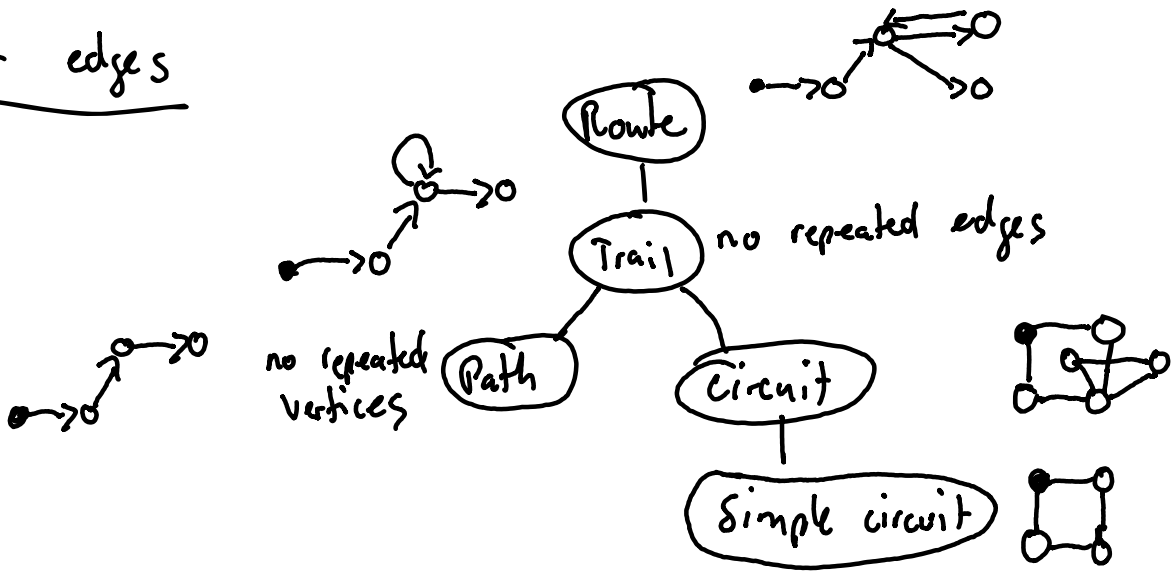
- Total degree is the sum of all degrees of all vertices



Handshake Theorem states that since each edge has two end segments ●—● and each segment ● or —● is counted once, then the total degree of the graph is always twice the number of edges

Trails, Paths, Circuits

Route/Walk/Travel in a graph is accomplished by **moving** from one vertex to another **along a sequence of adjacent edges**



Trail: route without repeated edge

Path: trail without repeated vertices

Circuit: trail that starts and ends at the same vertex (Euler circuit)

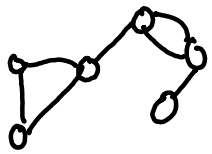
Simple circuit: circuit that only pass through each vertex of the circuit only once, except for the end and start vertices

(Hamiltonian circuit)

Connectedness

- A graph is connected if it is possible to find a route from any vertex to any other vertex along a sequence of adjacent edges of the graph

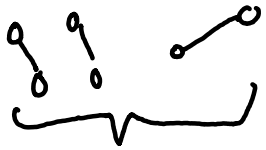
- Not connected iff there are two vertices that are not connected by any route



connected

not connected

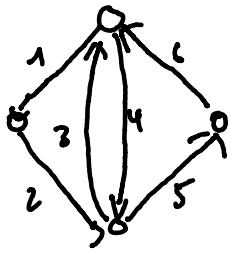
not connected



connected components

Euler Circuit (start & end are the same)

- passes through all vertices at least once
- passes through all edges only once

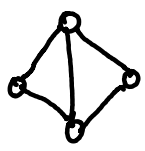


Graph has Euler Circuit, iff:

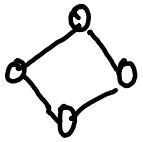
① graph is connected

② every vertex has even degree

NO



YES



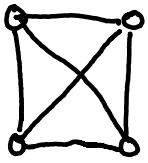
Hamiltonian Circuit (start & end the same)

- passes through <u>all</u> vertices <u>only once</u>

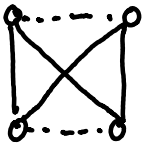
→ since it is a circuit, it has no repeated edges

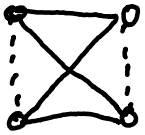
(- passes through <u>an arbitrary edge only once</u>)

→ simple circuit



H_1 H_2 H_3





difficult to find / compute (brute force)

→ used for Travelling Salesmen Problem

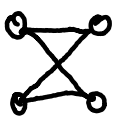
min distance to drive through n cities only once

$\frac{(n-1)!}{2}$ unique Hamiltonian circuits

→ used for Taxi Drivers, Delivery Systems

If a graph G contains a Hamiltonian circuit H, then:

1) G must be connected
2) H must contain all vertices
3) <u>Number of edges</u> of H must <u>equal</u> the <u>number of its vertices</u>
4) <u>Every vertex</u> of H must have <u>degree 2</u>

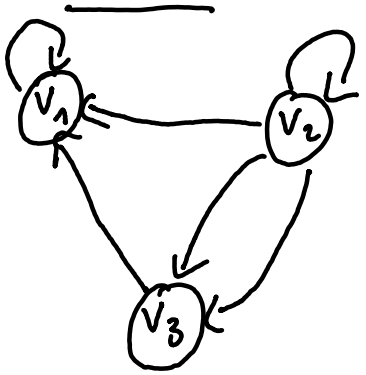


4 edges
4 vertices

All of them have to be true for a Hamiltonian Circuit

Matrix Representation of Graphs

Directed



Adjacency Matrix (number of edges)

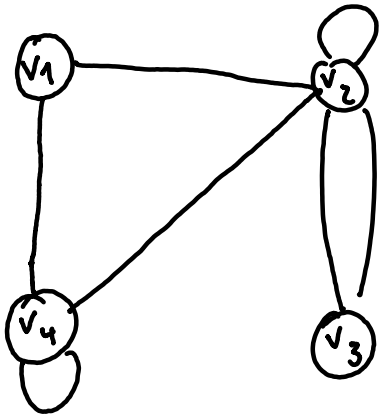
$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 2 \\ 1 & 0 & 0 \end{bmatrix} \end{array}$$

(to) j

(from) i

might be symmetric

Undirected

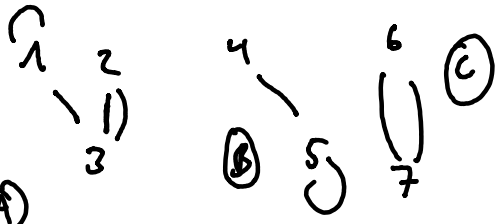


$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 2 & 1 \\ 0 & 2 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

always Symmetric

Application: Computing the number of distinct routes of length n

length 1 → use 1 edge, since it's a route, the same edge can be used several times



several connected components

	1	2	3	4	5	6	7
1				0	0	0	
2	A			0	0	0	
3				0	0	0	
4	0	0	0			0	0
5	0	0	0	B		0	0
6	0	0	0	0	0	C	
7	0	0	0	0	0		

Isomorphism

Identical Graphs: same vertex & edge sets
same edge-endpoint functions
(labels are important!)

Two graphs that are the same except for the labeling of their vertices and edges are called isomorphic (same form)
(labels are not important!)

Application: - Recognizing web users based on behavior
- Mathematical chemistry
- Analysis of business relations

Algorithm: Brute-Force, no simpler method $O(n! \cdot m!)$
$n!$: vertices, $m!$: edges

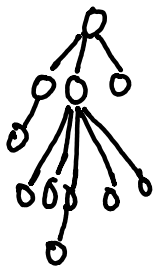
Trees

- connected graph that does not contain any circuits

n vertices

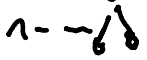
n-1 edges (every vertex has one parent except for root)

o ✓



Rooted Tree

- one vertex is the root
- unique path from root to any other vertex
- height starts at level 0
- leaves at the end



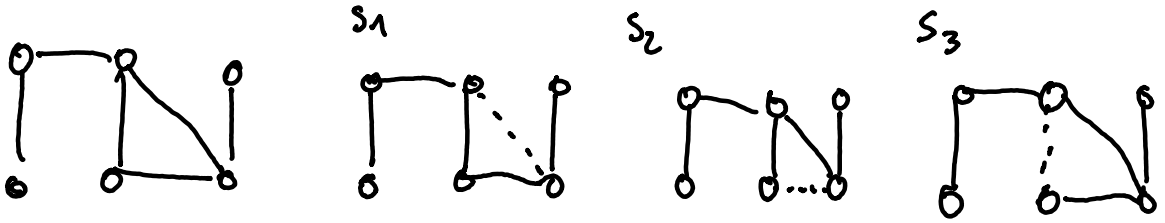
Binary Tree

Rooted Tree

- every vertex has at most two children

Spanning Tree

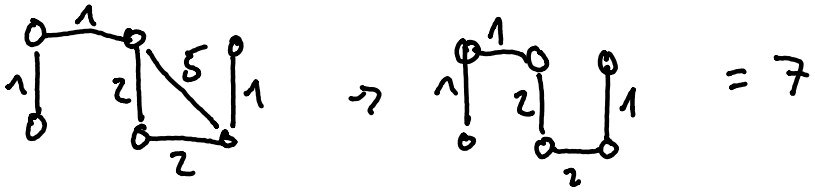
- Tree that contains every vertex of graph
- Every connected graph has a spanning tree
- Any two spanning trees for a graph have the same number of edges



Minimum Spanning Tree (MST)

applicable to weighted graphs

→ MST is a spanning tree for which the sum of weights of all edges is as small as possible

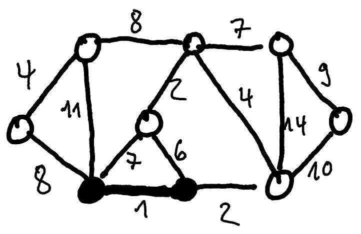


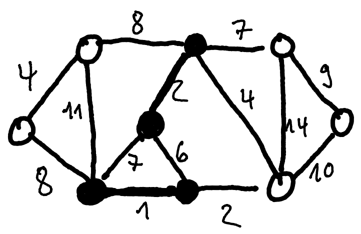
MST: Kruskal's Algorithm

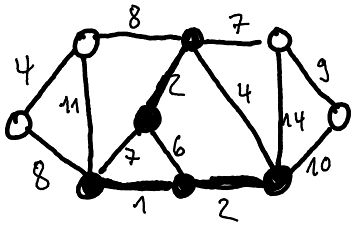
1) Mark edge with minimum weight and mark its connected vertices as visited

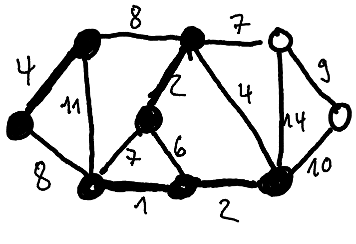
2) Add the <u>next unmarked</u> edge with <u>minimum weight</u> only if its <u>addition does not generate a circuit</u>. Mark connected vertices as marked

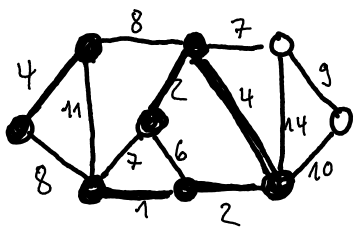
3) Repeat 2 until all vertices have been visited

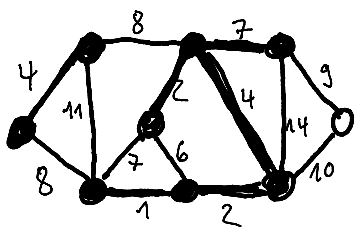


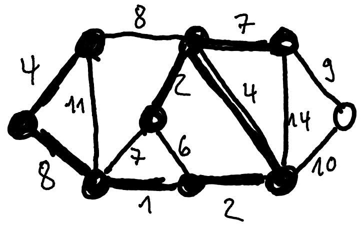


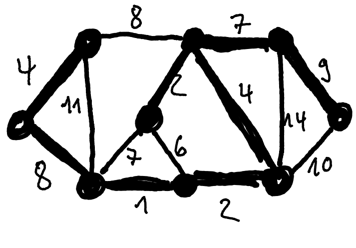






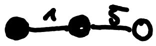






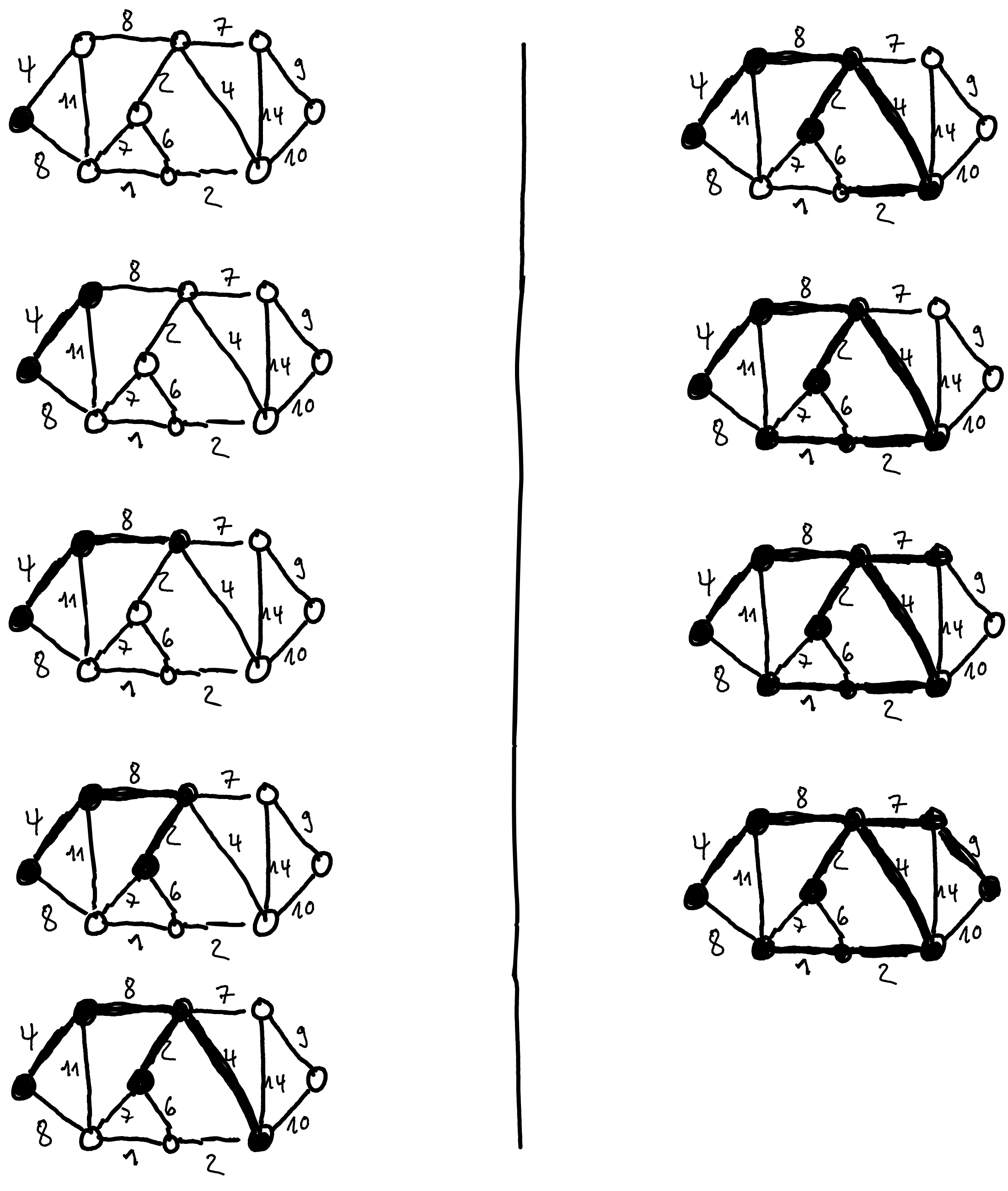
MST: Prim's Algorithm

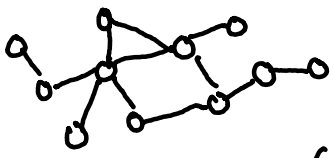
1) pick arbitrary vertex 

2) Mark the edge connected to it with the minimum weight and mark its connected vertex as visited 

3) Check all vertices visited so far which still see unvisited vertices and mark the adjacent edge with minimum weight and mark its connected vertex as visited (no cycle !)

4) Repeat from 3 until all vertices have been visited

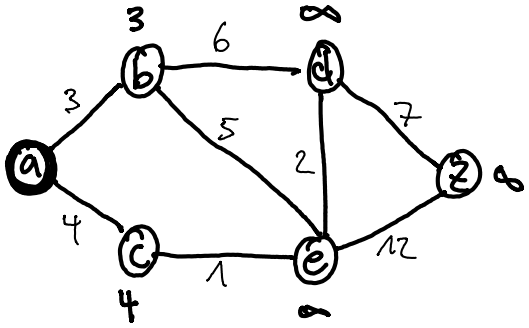


Kruskal's	Prim's
starts from edge with min weight	starts from arbitrary vertex
next: overall min weight edge which not results in cycle	next: connected min weight path to still unvisited vertex
faster for sparse graphs  dünn (less edges)	faster for dense graphs  dicht (more edges)

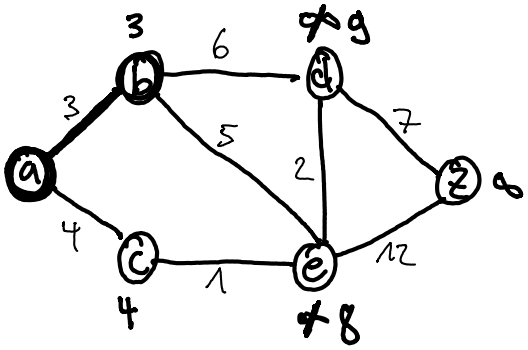
Dijkstra's Shortest Path Algorithm (From S to any other vertex)

- is a spanning tree but may not be the minimum spanning tree of the graph

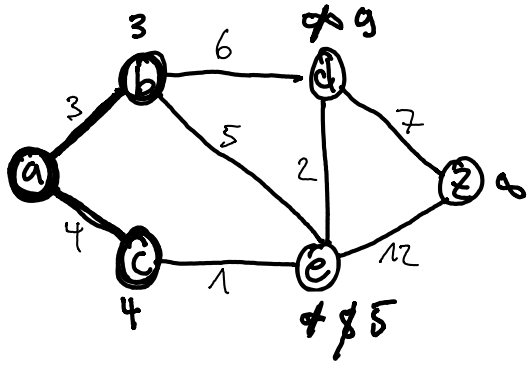
1) Define start vertex S
2) Assign to all its adjacent vertices the cost to reach them from S and to all non-adjacent an infinity cost
3) Check all vertices visited so far which still see unvisited vertices, then visit the vertex with min cumulated cost and update the cost of newly reachable vertices which can be reached in less cost
4) Repeat 3 until all vertices have been visited



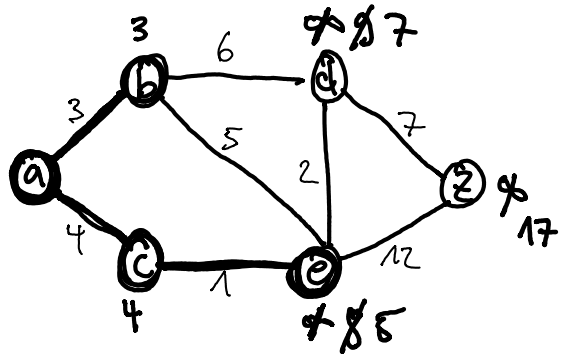
1



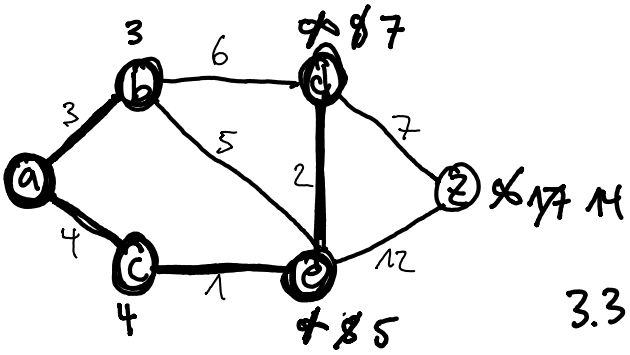
2



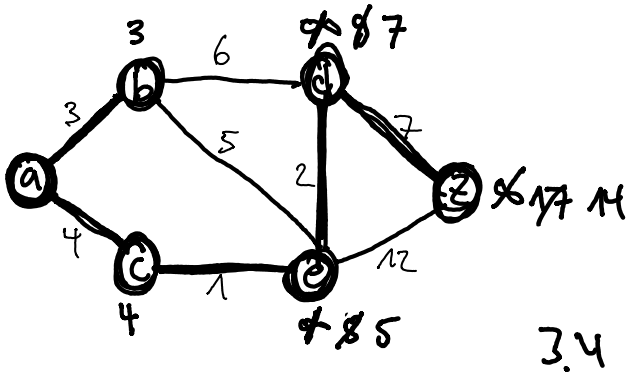
3.1



3.2



3.3



3.4

Application: - Travel Time (SBB / Google Maps) min time

- Fuel consumption min fuel

- CO2 emissions min CO2