

# Foundations of Computing I @ University of Zürich

## Syllabus

- Chapter 2: Introduction to Logic
- Chapter 5: Induction and recursion
- Chapter 7: Functions
- Extra topic: Convolution and neural networks
- Chapter 8: Relations and Modular Arithmetic
- Chapter 10: Graphs and trees
- Extra topic: Google Image Search
- Chapter 11: Algorithm Complexity

Week 1 - Proportional Logic - Compound statements (connectives) - Truth tables  
- Negation - Conjunction - Disjunction - Exclusive or (XOR) - Three input  
statements - De Morgan's law - Tautology and contradiction - Basic logical  
equivalences - Conditional statements

### Week 2

- Digital logic circuits
  - Simple circuit
  - Combinatorial circuit
  - Logic gates
  - Disjunctive Normal Form (DNF)
  - Conjunctive Normal Form (CNF)
  - Multiplexer
  - Binary to decimal
  - Decimal to binary
  - Half-Adder
  - Full-Adder

### Week 3

- Sequences
  - Geometric sequence
  - Arithmetic sequence
- Mathematical induction
  - Example: Sum of first n integers
  - Example: The triomino problem
  - Example: The towers of Hanoi
- Strong mathematical induction
  - Example: Prime factorization theorem
- Recursive functions
  - Example: Factorial function
  - Example: GCD (Greatest Common Divisor)
  - Example: The Euclidian Algorithm

- Algorithmic correctness

#### Week 4

- Set Theory
  - Cantor's definition of a set
  - Ordered tuples
  - Cartesian products
- Functions
  - Notable functions
    - \* Exponential rules
    - \* Logarithmic rules
- Convolution

#### Week 5

- Relations
  - Inverse of a Relation
  - Finite sets and directed graphs
  - Equivalence relations
    - \* Reflexivity
    - \* Symmetry
    - \* Transitivity
  - Example: Equivalence relation
- Congruences
  - Examples
- Modular arithmetic
  - Inverse modulo  $n$
  - Bezout's theorem
  - Example: Caesar cipher
  - RSA cryptography

#### Week 6

- Graphs
  - Terminology of graphs
  - Matrices and directed graphs
  - Matrices and undirected graphs
  - Matrices and connected components
  - Isomorphisms of graphs
- Trails
- Paths
- Circuits
  - Euler circuits
  - Hamiltonian circuits
  - Example: Travelling salesman problem
- Trees
  - Rooted trees
  - Binary trees

- Octrees
  - Spanning trees
    - \* Minimum spanning trees
- Kruskal's Algorithm
- Prim's Algorithm
- Dijkstra's Algorithm

## Proportional Logic

- The field of Natural Language Processing (NLP) is advancing at a high speed, with devices such as Amazon Alexa, Google Home etc. using the underlying technology to turn spoken language into commands. This technology is based on proportional logic, because it tries to immitate the human language -> the same sentence can be said in multiple ways, but always keeping the same meaning.
- A statement is either **true** or **false**, it can not be both.

### Compound statements (connectives)

Name	Term	Symbol
NOT	Negation	$\sim$
AND	Conjunction	$\wedge$
OR	Disjunction	$\vee$

- Variables in proportional logic can only have two values, either true or false. This is why they are called boolean variables.
- Negation has presedence over conjunction and disjunction.

Name	Term
"It is hot"	$p$
"It is sunny"	$q$
"It is not hot but it is sunny"	$\sim p \wedge q$
"It is neither hot nor sunny"	$\sim (p \wedge q)$
"It is not true that it is hot and sunny"	$\neg(p \wedge q)$

## Truth tables

### Negation

$p$	$\sim p$
T	F
F	T

### Conjunction

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

### Disjunction

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

### Exclusive or (XOR)

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

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

**Three input statements** With three statements, it makes sense to evaluate the each of the statements individually and then put the stements together.

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

**De Morgan's law** When the negation sign is applied to a parantheses, the expressions “and” as well as “or” are interchanged.

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

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

**Tautology and contradiction** A tautology (denoted by the symbol t) is a statement that is always true regardless of the truth values of its component statements

$$p \wedge t \equiv \mathbf{t}$$

A contradiction (denote by the symbol c) is a statement that is always false regardless of the truth values of its component statements

$$p \wedge c \equiv \mathbf{c}$$

## Basic logical equivalences

Name	Laws
Commutative	$p \wedge q \equiv q \wedge p$
Associative	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Distributive	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
Identity	$p \wedge \mathbf{t} \equiv p$
Negation	$p \vee \sim p \equiv \mathbf{t}$
Double negative	$\sim(\sim p) \equiv p$
Idempotent	$p \wedge p \equiv p$
Universal bound	$p \vee \mathbf{t} \equiv \mathbf{t}$
De Morgan's	$\sim(p \wedge q) \equiv \sim p \vee \sim q$
Absorption	$p \vee (p \wedge q) \equiv p$
Negations of t and c	$\sim \mathbf{t} \equiv \mathbf{c}$

Name	Laws
Commutative	$p \vee q \equiv q \vee p$
Associative	$(p \vee q) \vee r \equiv p \vee (q \vee r)$
Distributive	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
Identity	$p \vee \mathbf{c} \equiv p$
Negation	$p \wedge \sim p \equiv \mathbf{c}$
Double negative	N. A.
Idempotent	$p \vee p \equiv p$
Universal bound	$p \wedge \mathbf{c} \equiv \mathbf{c}$
De Morgan's	$\sim(p \vee q) \equiv \sim p \wedge \sim q$
Absorption	$p \wedge (p \vee q) \equiv p$
Negations of t and c	$\sim \mathbf{c} \equiv \mathbf{t}$

## Conditional statements

If-then conditional statements, if is the hypothesis and q is the conclusion

$p$	$q$	$p \rightarrow q$	$\sim p \vee q$
T	T	T	T
T	F	F	F
F	T	T	F
F	F	T	T

If-and only if means p implies q and q implies p.

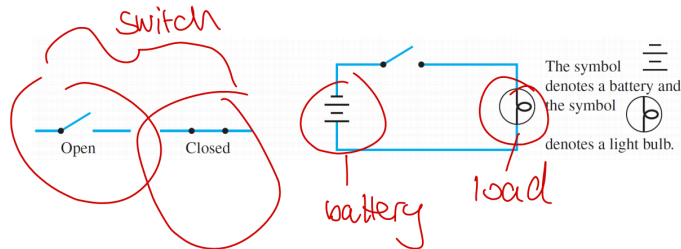
$$(p \rightarrow q) \wedge (q \rightarrow p)$$

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

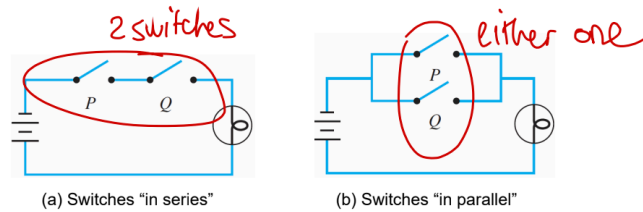


# Digital logic circuits

## Simple circuit



A simple circuit has three elements: switch, battery and load



Switches can either be used "in series" (the circuit will only work when both switches are closed) or "in parallel" (the circuit will work when one of the both switches are closed).

The in series circuit is equivalent to the AND truth table, where closed is equivalent to true and the truth table is only true when both values are true (closed).

The in parallel circuit is equivalent to the OR truth table, where closed is equivalent to true and the truth table is only false when both values are false (open).

## Combinatorial circuit

- combinational: type of digital circuit whose output only depends on the current input.
- sequential: output depends on both the current input and previous outputs. This means a sequential circuit preserves a memory of the input while a combinational circuit does not (out of scope for this course)

3 possible inputs




Input			Output
P	Q	R	S
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1
0	0	0	0

1 possible output

### Rules for combinational circuits

1. Never combine the input wires.
  2. Never feed back the output of a gate into the same gate
- combinational equivalence: two circuits can have the same output but built completely differently
  - circuit minimization: for production of electrical circuit it makes sense minimizing more complicated circuits to simplified ones so that they cost lesser in production.

## Logic gates

Type of Gate	Symbolic Representation	Action																	
NOT		<table><tr><th>Input</th><th>Output</th></tr><tr><th><math>P</math></th><th><math>R</math></th></tr><tr><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td></tr></table>	Input	Output	$P$	$R$	1	0	0	1									
Input	Output																		
$P$	$R$																		
1	0																		
0	1																		
AND		<table><tr><th>Input</th><th>Output</th></tr><tr><th><math>P</math></th><th><math>Q</math></th><th><math>R</math></th></tr><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table>	Input	Output	$P$	$Q$	$R$	1	1	1	1	0	0	0	1	0	0	0	0
Input	Output																		
$P$	$Q$	$R$																	
1	1	1																	
1	0	0																	
0	1	0																	
0	0	0																	
OR		<table><tr><th>Input</th><th>Output</th></tr><tr><th><math>P</math></th><th><math>Q</math></th><th><math>R</math></th></tr><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table>	Input	Output	$P$	$Q$	$R$	1	1	1	1	0	1	0	1	1	0	0	0
Input	Output																		
$P$	$Q$	$R$																	
1	1	1																	
1	0	1																	
0	1	1																	
0	0	0																	

## Disjunctive Normal Form (DNF)

Also called “Or of And’s”

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

- Best when output is 0
- AND together all variables where output is 1
- OR together all the 1 rows
- Simplify

## Conjunctive Normal Form (CNF)

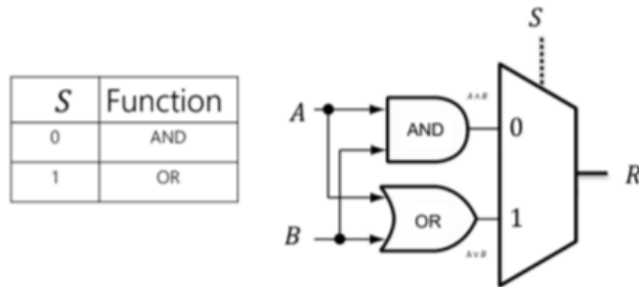
Also called “And of Or’s”

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

- Best when output is mostly 1
- Identify rows with output 0
- OR together all variables in some row (negate when 1)
- AND together the 0 rows

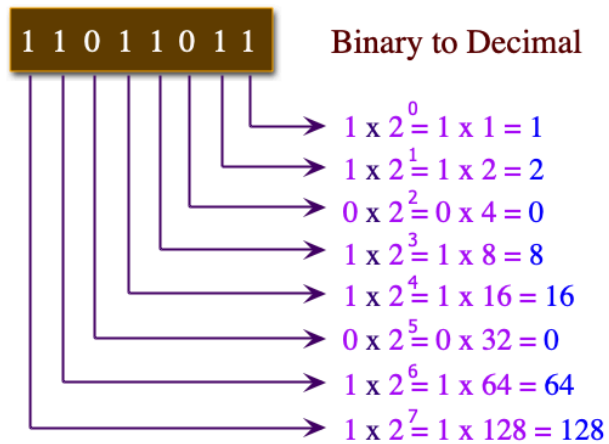
## Multiplexer

This Multiplexer selects the output of the AND operation if the selector  $S=0$ , otherwise it selects the output of the OR operation.



## Binary to decimal

For the binary number, read the numbers from bottom to top.



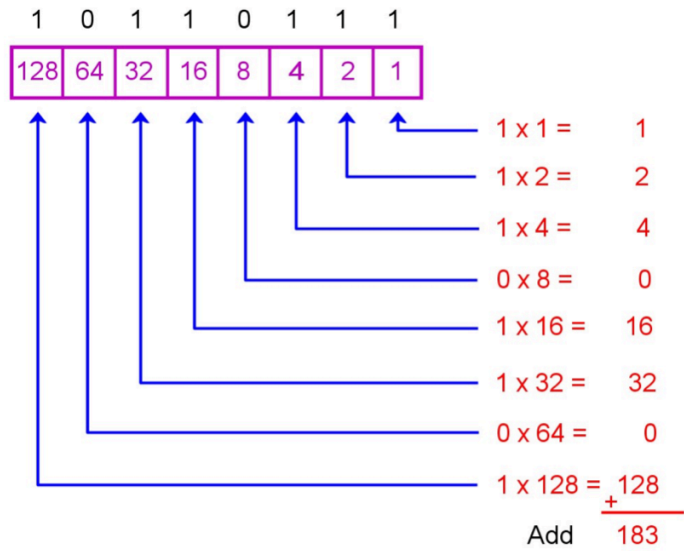
$$1 + 2 + 8 + 16 + 64 + 128 = 219$$

$$(11011011)_2 = (219)_{10}$$

## Decimal to binary

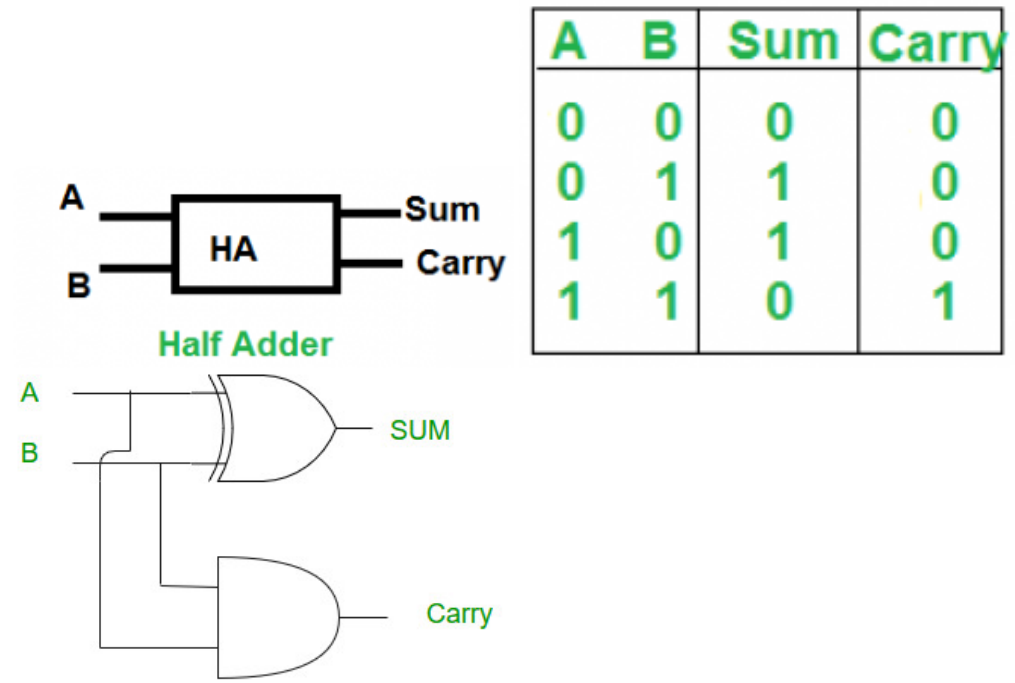
Multiply each digit with it's binary equivalent.

Convert 10110111 to Decimal

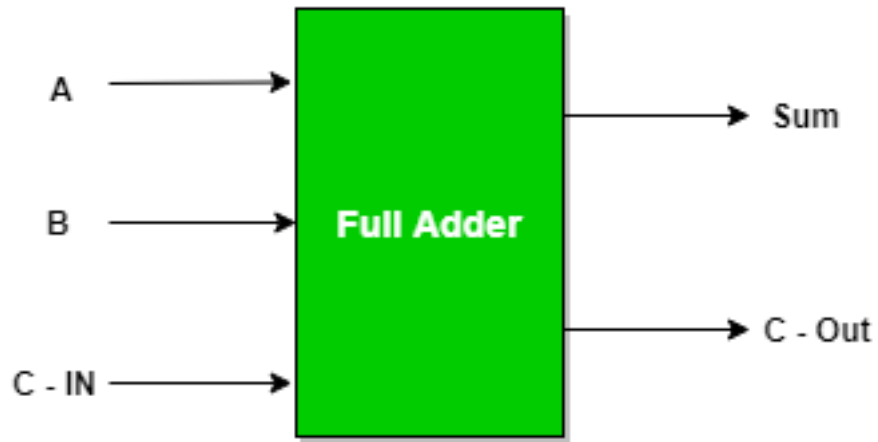


10110111 = 183 decimal

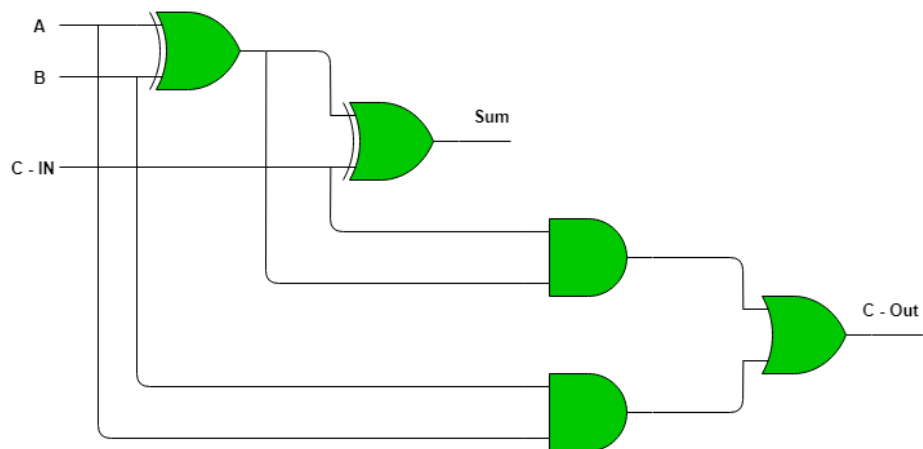
## Half-Adder



## Full-Adder



Inputs			Outputs	
A	B	C - IN	Sum	C - Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



## Sequences

A sequence is a function whose domain is either all the integers between two given integers or all the integers greater than or equal to a given integer. The sequence is typically represented as a set of elements written in a row  $a_m, a_{m+1}, a_{m+2}, \dots, a_n$ , where each individual element  $a_k$  is called a term. The  $k$  is called subscript or index.

### Geometric sequence

A Sequence is only called an geometric sequence if and if only there is a constant  $r$  such that

$$a_k = a_{k-1} * r$$

$$1, r, r^2, r^3, r^4, \dots, r^n = \sum_{i=0}^n r^i = \frac{r^{n+1} - 1}{r - 1}$$

1) base case

$$\begin{aligned} P(0) \quad \sum_{i=0}^1 r^i &= \frac{r^{0+1} - 1}{r - 1} \\ &= \frac{r - 1}{(r - 1)} \\ &= 1 \end{aligned}$$

2) inductive step

" $P(k) \rightarrow P(k+1)$  is true"

$$\begin{aligned} \sum_{i=0}^{k+1} r^i &= \frac{r^{(k+1)+1} - 1}{r - 1} \\ &= \frac{r^{k+2} - 1}{r - 1} \end{aligned}$$

$$\sum_{i=0}^{k+1} r^i = 1 + r + r^2 + \dots + r^k + r^{k+1}$$

$$\begin{aligned} &\frac{r^{k+1} - 1}{r - 1} + r^{k+1} \\ &= \frac{(r^{k+1} - 1) + r^{k+1}(r - 1)}{(r - 1)} \\ &= \frac{\cancel{r^{k+1}} - 1 + r^{k+2} - \cancel{r^{k+1}}}{(r - 1)} \\ &= \frac{r^{k+2} - 1}{r - 1} \end{aligned}$$

65

### Arithmetic sequence

A Sequence is only called an arithmetic sequence if and if only there is a constant  $d$  such that

$$a_k = a_{k-1} + d$$

## Mathematical induction

4 step plan for solving induction problems:

1. Identify base case
2. Set up induction hypothesis
3. Prove hypothesis (if true) with induction step
4. Conclude



### Example: Sum of first n integers

$1 + 2 + \dots + n = \frac{n(n+1)}{2}$  for all integers  $n \geq 1$

Base case:  $P(1) = \frac{1(1+1)}{2} = \frac{2}{2} = 1$

Inductive hypothesis: “ $P(k) = \frac{k(k+1)}{2}$  is true”

Inductive step:  $P(k+1) = \frac{k+1((k+1)+1)}{2} = \frac{(k+1)(k+2)}{2}$

$\frac{(k+1)(k+2)}{2}$  is of the same form as  $\frac{n(n+1)}{2} \rightarrow (k+1) = n$

### Example: The triomino problem

See this link for more

### Example: The towers of Hanoi

See this link for more

## Strong mathematical induction

- Base case: Show that  $P(a) \wedge P(a+1) \wedge \dots \wedge P(b) \wedge \dots \wedge P(k)$  is true
- Inductive step
  - Hypothesis: Assume base case is true
  - Show that  $P(k+1)$  is true
- Conclusion if base case and inductive hypothesis is true for all  $n \geq a$

### Example: Prime factorization theorem

See this link for more

## Recursive functions

A recursive function is recursive when its definition refers to itself. It is made of a base case and one or multiple recursive calls. The recursive functions are easier to implement for complex problems. It's more elegant and easier to understand, short and concise. But it is more expensive in regards of time and memory. Unnecessary calls do double check on for / if / while loops.

### Example: Factorial function

```
int factorial (int n)
{
    if (n=0)
        return 1;
    else
```

```

        return n * factorial(n-1)
    }

```

The function refers to itself, calling itself as long as  $n$  is not equal to 0.

### Example: GCD (Greatest Common Divisor)

The GCD is one of the most used algorithms in Computer Science, mostly in the field of cryptography.

$$\text{gcd}(72, 63) \rightarrow 72 = 3^2 * 2^3, 63 = 3^2 * 7$$

Both elements have a  $3^2$ , which means  $\text{gcd}(72, 63) = 3^2 = 9$

### Example: The Euclidian Algorithm

```

int gcd (int A, int B)
{
    if (B=0)
        return A;
    else return gcd(B, A mod B)
}

```

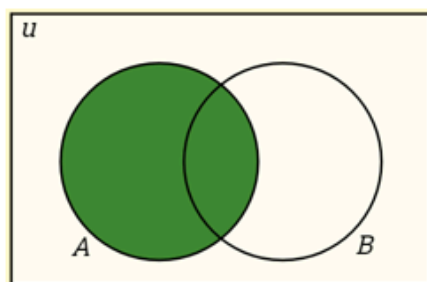
### Algorithmic correctness

- Base property:  $P(0)$  is true
- Inductive property: *Guard* and loop invariant  $P(k)$
- Eventual Falsity of the Guard: after finite number of repetitions,  $G$  becomes false.
- Correctness of post-condition.: if  $N$  is at least to, which makes  $G$  becomes false

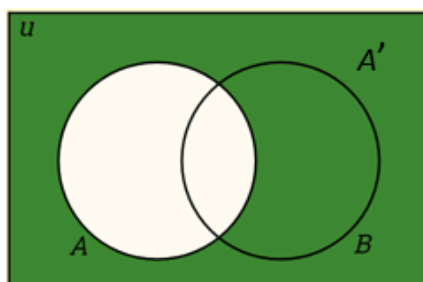
## Set Theory

### Cantor's definition of a set

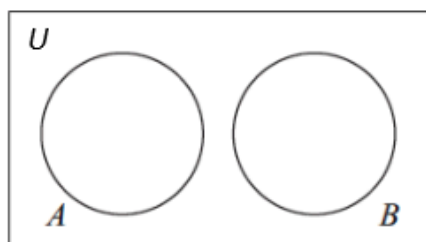
A set is a collection of definite and separate objects. These objects are called elements of the set.



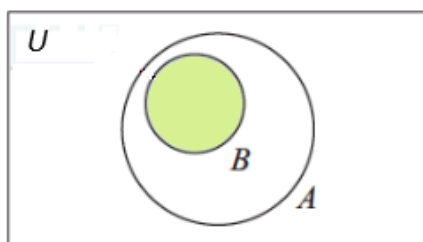
Set A



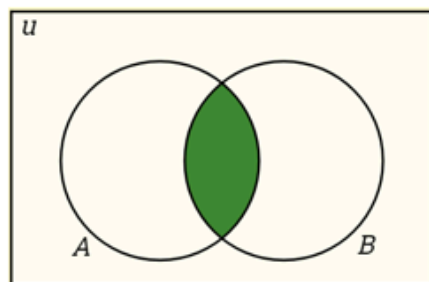
$A'$  the complement of A



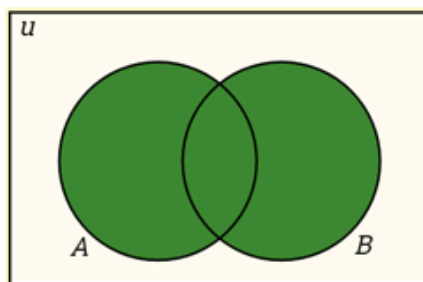
A and B are disjoint sets



$B$  is proper subset of A  
 $B \subset A$



Both A and B  
A intersect B  
 $A \cap B$



Either A or B  
A union B  
 $A \cup B$

An empty set is denoted by  $\{ \}$ , it is a unique set with no elements.

### Ordered tuples

An ordered  $n$ -tuple is defined by  $(x_1, x_2, \dots, x_n)$

An ordered pair is defined by  $(x_1, x_2)$

An ordered triple is defined by  $(x_1, x_2, x_3)$

Rule:  $(a, b) = (c, d)$  iff (in and only if)  $(a = c), (b = d)$

### Cartesian products

The Cartesian product  $A_1 \times A_2 \times \dots \times A_n$  of ordered n-tuples  $(a_1, a_2, \dots, a_n) | (a_1 \in A_1) \wedge (a_2 \in A_2) \wedge \dots \wedge (a_n \in A_n)$

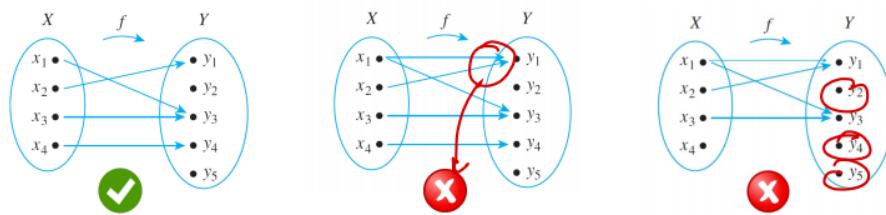
### Functions

A function is built from two sets,  $X$  and  $Y$ . What the function does is build a relation between these two sets following these two rules:

- Every element of  $X$  is related to some elements in  $Y$
- No element of  $X$  is related to more than one element in  $Y$

Following cases are also allowed:

- There can be elements of the co-domain  $Y$  that are not function of any input element.
- Different inputs can mapped to the same element of  $Y$ .



### Notable functions

- The identity function:  $I : X \rightarrow X$ , where each element is mapped to itself.
- The exponential function
- The log-function
- The Hamming-Distance Function
- The image function

### 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:

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

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

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

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

Exponential rules

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

Logarithmic rules

### Convolution

- Check slides for explanation
- See this link for a demo

## Relations

Relations are more general than functions. The key difference is that the same element in the Domain may be related to multiple elements in the Co-domain

### Inverse of a Relation

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

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

What this means is visualized in the following diagram:



In this case, the second diagram is an inverse of the first.

### Finite sets and directed graphs

A directed graph displays the relations inside a finite set. What we see is that from  $A = \{2, 3, 4, 6, 7, 9\}$  - we created the three sets  $\{(4, 7)\}, \{(2)\}, \{(3, 6, 9)\}$

Example:

let  $A = \{2, 3, 4, 6, 7, 9\}$  and a relation  $R$  on the set  $A$  be defined by the following directed graph:



### Equivalence relations

An equivalence relation is only one if the three following conditions are fulfilled:

#### Reflexivity

$R$  is reflexive if, and only if, for all  $x \in A$ ,  $xRx$ .

What this means, is that  $x$  is related to itself. In a directed graph, this would be an arrow from a number showing back to the same number.

## Symmetry

$R$  is symmetric if, and only if, for all  $x, y \in A$ , if  $xRy$  then  $yRx$ .

What this means, is that two elements are related to each other symmetrically. In a directed graph, these two numbers are connected by two arrows pointing at each other.

## Transitivity

$R$  is transitive if, and only if, for all  $x, y, z \in A$ , if  $xRy$  and  $yRz$  then  $xRz$ .

What this means, when we have three elements  $x, y, z$ ,  $x$  is related to  $y$  and  $y$  is related to  $z$  which means that  $x$  is also related to  $z$ . In directed graph, we would have 3 elements in a triangular form all pointing at each other.

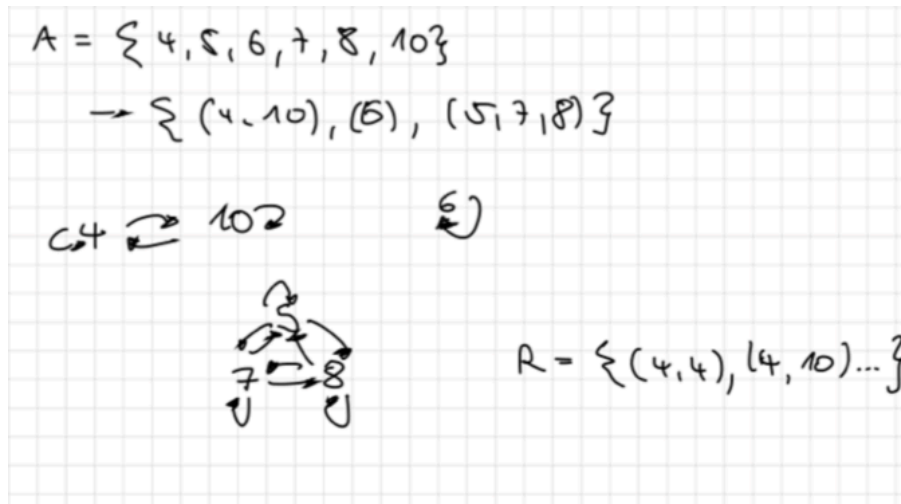
## Example: Equivalence relation

From  $A = 4, 5, 6, 7, 8, 10$  we arrived to the relation

$$R = (4, 4), (4, 10), (10, 4), (10, 10), (6, 6), (5, 5), (7, 7), (8, 8), (5, 8), (8, 7), (7, 5)$$

Here we can also see the three conditions clearly.

- Reflexivity:  $(4, 4), (10, 10), (6, 6), (5, 5), (7, 7), (8, 8)$
- Symmetry:  $(4, 10), (10, 4)$
- Transitivity:  $(5, 8), (8, 7), (7, 5)$



## Congruences

Let  $m$  and  $n$  be integers and let  $d$  be a positive integer. We say that  $m$  is congruent to  $n$  modulo  $d$  and write

$$m \equiv n \pmod{d}$$

if and only if,  $d \mid (m - n)$

## Examples

- $12 \equiv 7 \pmod{5} \rightarrow 5 \mid (12 - 7) = 5 \rightarrow 5 \mid 5 \rightarrow \text{true}$
- $6 \equiv -8 \pmod{4} \rightarrow 4 \mid (6 - (-8)) = 14 \rightarrow 4 \nmid 14 \rightarrow \text{false}$
- $3 \equiv 3 \pmod{7} \rightarrow 7 \mid (3 - 3) = 0 \rightarrow 7 \mid 0 \rightarrow \text{true}$

## Modular arithmetic

### 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.  $a \bmod n = b \bmod n$

## Inverse modulo $n$

The modular inverse of an integer  $a$  is an integer  $x$  such that:  $ax \equiv 1 \pmod{n}$

Example: Inverse of 3 modulo 7

- $3 \cdot 0 \equiv 0 \pmod{7}$
- $3 \cdot 1 \equiv 3 \pmod{7}$
- $3 \cdot 2 \equiv 6 \pmod{7}$
- $3 \cdot 3 \equiv 2 \pmod{7}$
- $3 \cdot 4 \equiv 5 \pmod{7}$
- **$3 \cdot 5 \equiv 1 \pmod{7}$**
- $3 \cdot 6 \equiv 4 \pmod{7}$

## Bezout's theorem and Euclidian Algorithm

$$\gcd(a, b) = sa + tb$$



$$\begin{aligned}
\gcd(35, 27) &= \gcd(27, 35 \bmod 27) = \gcd(27, 8) & \rightarrow 35 &= 1 \cdot 27 + 8 & \rightarrow 8 &= 35 - 27 \\
&= \gcd(8, 27 \bmod 8) = \gcd(8, 3) & \rightarrow 27 &= 3 \cdot 8 + 3 & \rightarrow 3 &= 27 - 3 \cdot 8 \\
&= \gcd(3, 8 \bmod 3) = \gcd(3, 2) & \rightarrow 8 &= 2 \cdot 3 + 2 & \rightarrow 2 &= 8 - 2 \cdot 3 \\
&= \gcd(2, 3 \bmod 2) = \gcd(2, 1) = 1 & \rightarrow 3 &= 1 \cdot 2 + 1 & \rightarrow 1 &= 3 - 2
\end{aligned}$$

$$\begin{aligned}
\rightarrow 1 &= 3 - 2 = 3 - (8 - 2 \cdot 3) = \\
&= 3 \cdot 3 - 8 = 3 \cdot (27 - 3 \cdot 8) - 8 = 3 \cdot 27 - 10 \cdot 8 \\
&= 3 \cdot 27 - 10 \cdot (35 - 27) = 13 \cdot 27 - 10 \cdot 35 \quad \boxed{\rightarrow 1 = 13 \cdot 27 - 10 \cdot 35}
\end{aligned}$$

### Example: Ceasar cipher

An encryption system which uses the 26 letters of the alphabet but just by pushing them some places forward.

Example:  $A = D$  because our steps we are using is 3, which would mean  $B = E$ ,  $C = F$  etc.

Formula for encryption:  $C = (M+3) \bmod 26$

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

Very easy to hack once the factor is known.

### RSA cryptography

In RSA, the plaintext  $M$  is converted into ciphertext  $C$  according to the following formula:

$$C = M^e \bmod pq$$

$pq$  and  $e$  are the public keys and anyone can use them to encrypt their messages!

The plaintext  $M$  for a ciphertext  $C$  is then recovered as follows:

$M = C^d \bmod pq$ . Where  $d$  is the private key; it is secret and only the recipient knows it.

### When does the RSA Cipher work?

For the RSA to work, the following expression must all for all positive integers  $M < pq$ :  $M = (M^e)^d \bmod pq$

This holds if:

- $p$  and  $q$  are prime

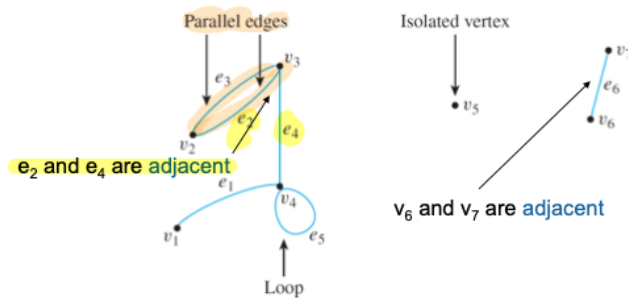
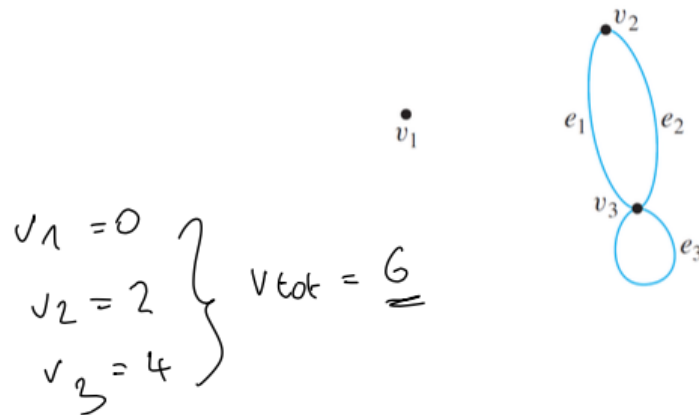
- $e$  and the product  $(p-1)(q-1)$  are relatively prime (e.g., their greatest common divisor is 1)
- $ed \equiv 1 \pmod{(p-1)(q-1)}$ .
  - (i.e.,  $d$  is the inverse of  $e$  modulo  $(p-1)(q-1)$ ).

# Graphs

Graphs are a powerful problem-solving tool because they enable us to represent a complex situation with a single image that can be analyzed both visually and with the aid of a computer

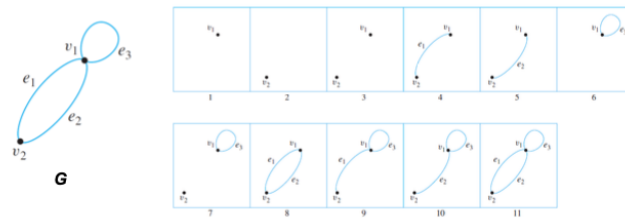
## Terminology of graphs

- Graph: The whole drawing
- Parallel edges: 2 edges connecting to each other
- Isolated vertex: No edges connected
- Adjacency: 2 connected edges
- Degree: Number of edges that start or end at a vertex



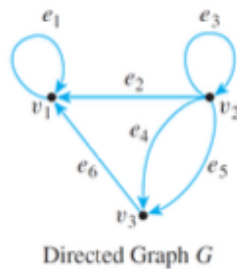
## Sub-graphs

The left graph has 11 sub-graphs



## Matrices and directed graphs

Show connections from vertices to other vertices in form of a matrix



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

Adjacency Matrix

## Matrices and undirected graphs

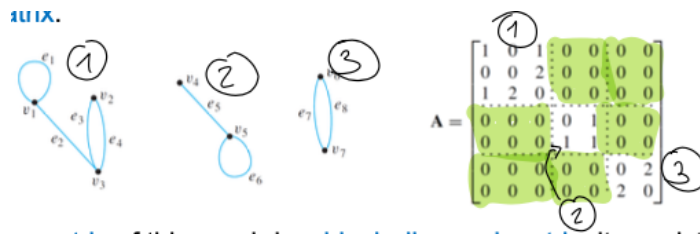
Note that an adjacency matrix of an undirected graph is symmetric.

• **Definition**

An  $n \times n$  square matrix  $A = (a_{ij})$  is called **symmetric** if, and only if, for all  $i, j = 1, 2, \dots, n$ ,

$$a_{ij} = a_{ji}.$$

## Matrices and connected components

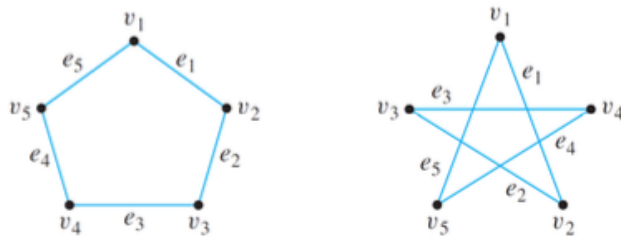


The adjacency matrix of this graph is a block-diagonal matrix: it consists of square matrix blocks for each connected component, along the diagonal, and

blocks of 0's everywhere else. The reason is that vertices in each connected component share no edges with vertices in other components.

## Isomorphisms of graphs

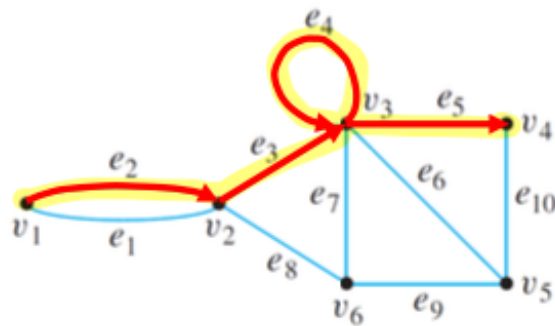
Two graphs that are the same except for the labeling of their vertices and edges are called isomorphic



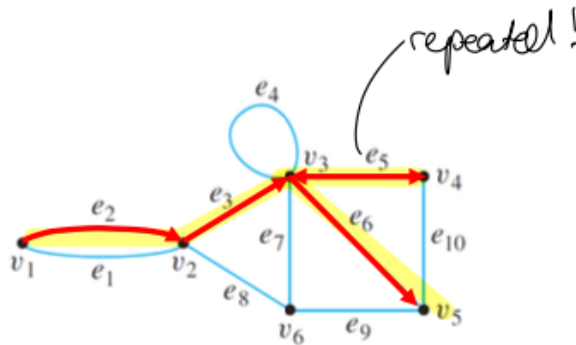
The first two graphs are identical although their topology is different: their vertex and edge sets are identical and their edge-endpoint functions are the same.

## Trails

Trails are routes that do not have a repeated edge.



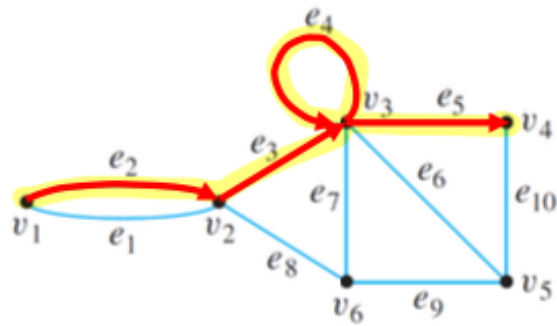
Not a trail:



## Paths

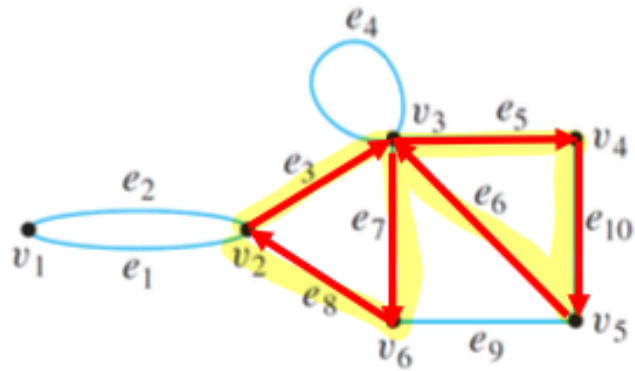
Paths are trails that do not have repeated vertices

Without the loop on  $v_3$  this would be a path



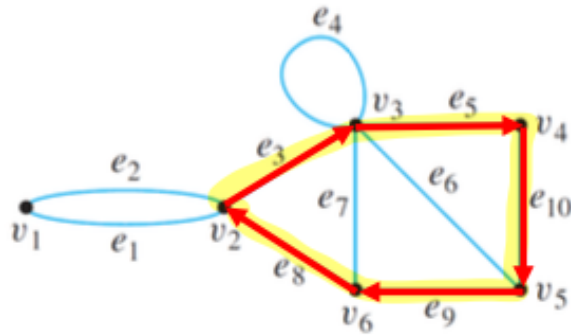
## Circuits

Circuits are trails that start and end at the same vertex



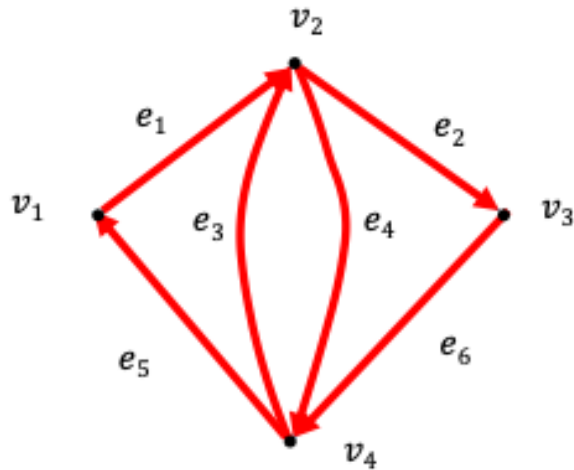
### Simple circuits

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



### Euler circuits

An Euler circuit passes through all the vertices of  $G$  at least once and passes through all the edges only once.



A graph has an Euler circuit if, and only if, the graph is connected and every vertex of the graph has even degree (because the total number of arrivals and departures from each vertex must be a multiple of 2).

### Hamiltonian circuits

A Hamiltonian circuit for a graph  $G$  is a circuit that passes through all the vertices of  $G$  only once (except for the start and end vertices). Since it is a circuit, it has no repeated edges.

A Hamiltonian circuit is a simple circuit that passes through all vertices of  $G$  (remember that, according to the definition, a simple circuit does not need to pass through all vertices of  $G$ ).

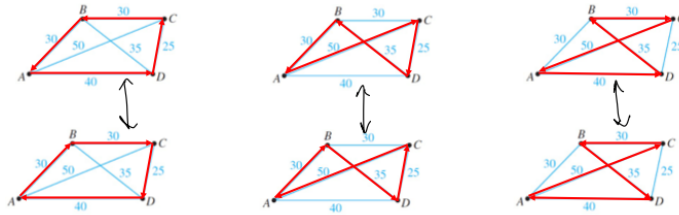
### Example: Travelling salesman problem

Suppose we had a complete graph with 4 cities, like the example before.

1. From A there are 3 cities we can visit first (B, C, D).
2. From each of those, there are 2 possible cities to visit.
3. And, from each of those, there is then only 1 choice before returning home.

Total 6 routes, but 3 of them are duplicates. So 3 unique routes. Formula  $\frac{(n-1)!}{2}$  for  $n$  cities.

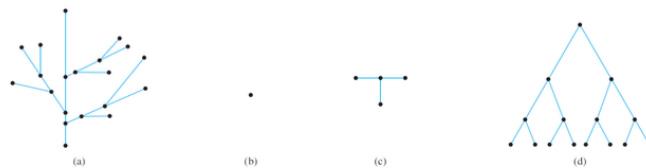




At present, there is no efficient and optimal algorithm to solve the Travelling Salesman Problem (TSP).

## Trees

A tree is a connected graph that does not contain any circuits. A tree with  $n$  vertices has  $n - 1$  edges.



## Rooted trees

- Rooted trees (Example d in screenshot above)
- Unique path from root to leaf
- Leveled
- Height = maximal level

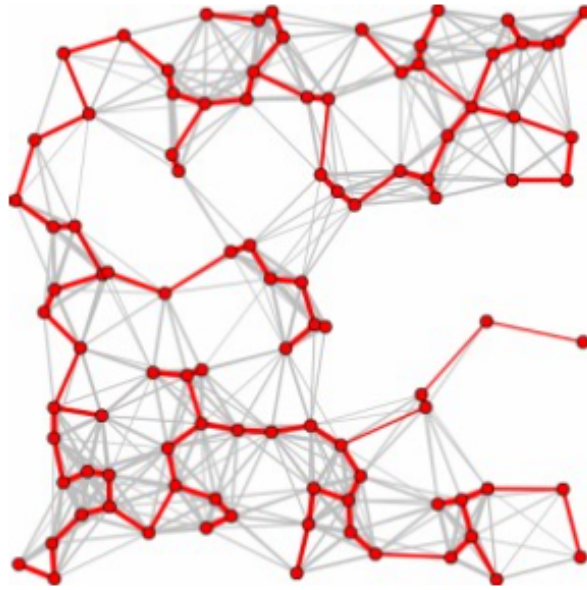
## Binary trees

- Each vertex has at most two children
- When each internal vertex has exactly two children, it is called full binary tree

## Octrees

- Each vertex has exactly 8 children
- Used in 3D graphics

## Spanning trees



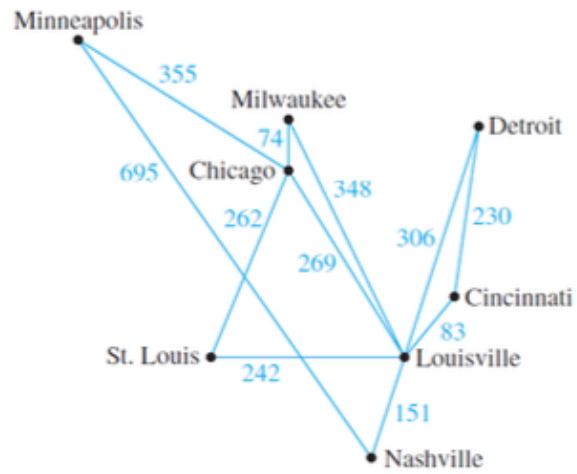
- A spanning tree for a graph is a tree that contains every vertex of the graph.
- Every connected graph has a spanning tree.
- Any two spanning trees for a graph have the same number of edges.

### Minimum spanning trees

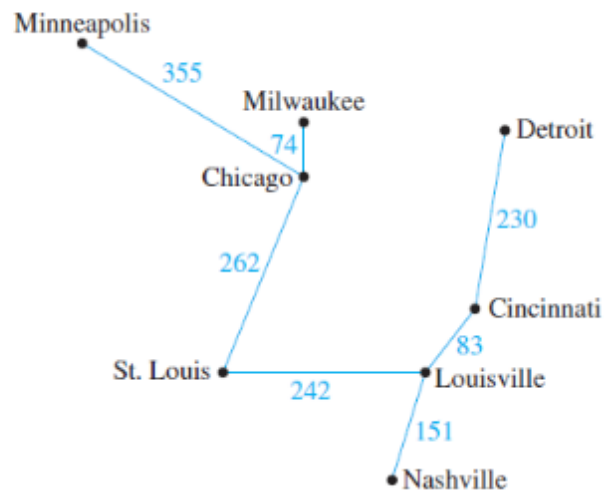
A graph whose edges are labeled with numbers (weights) is called a weighted graph.

A minimum spanning tree, is a spanning tree for which the sum of the weights of all the edges is as small as possible.

Problem:



Solution:



## Kruskal's Algorithm

1. Mark the edge with minimum weight and mark its connected vertices as visited.
2. Add the next unmarked edge with minimum weight only if its addition does not generate a circuit. Mark its connected vertices as visited.
3. Repeat from 2 until all vertices have been visited.

## Prim's Algorithm

1. Pick an arbitrary vertex.
2. Mark the edge connected to it that has minimum weight and mark its connected vertex as visited.
3. Check all vertices visited so far which still “see” unvisited vertices and mark the one adjacent edge with minimum weight and mark its connected vertex as visited.
4. Repeat from 3 until all vertices have been visited.

## Dijkstra's Algorithm

Main idea:

1. Define the start vertex S (this is usually given)
2. Assign to all its adjacent vertices the cost to reach them from S and to all non-adjacent vertices an infinite cost.
3. Check all vertices visited so far which still “see” unvisited vertices, then visit the vertex with minimum cost and update the cost of all its adjacent vertices with the total cost to reach them from S passing through the current vertex.
4. Repeat from 3 until all vertices have been visited