Exam preparation cheetzheet:
Stuff from

MNF130V2020

May 23, 2020

1 Chapter 1

Propositional logic:. Logical $\wedge, \vee, \oplus$ are trivial.
Conditional statements (implication): $p \to q$, if p then $q \equiv p$ only if $q \equiv p$ is a sufficient condition for q.

In other words, q is a necessary condition for p. $p \to q$ is false then p is true and q is false and otherwise true. $\neg(p \to q) \equiv p \wedge (\neg q)$, $p \to q$ is equivalent to its contrapositive $\neg q \to \neg p$, but **not** to its **converse** $q \to p$ **or** its inverse $\neg p \to \neg q$.

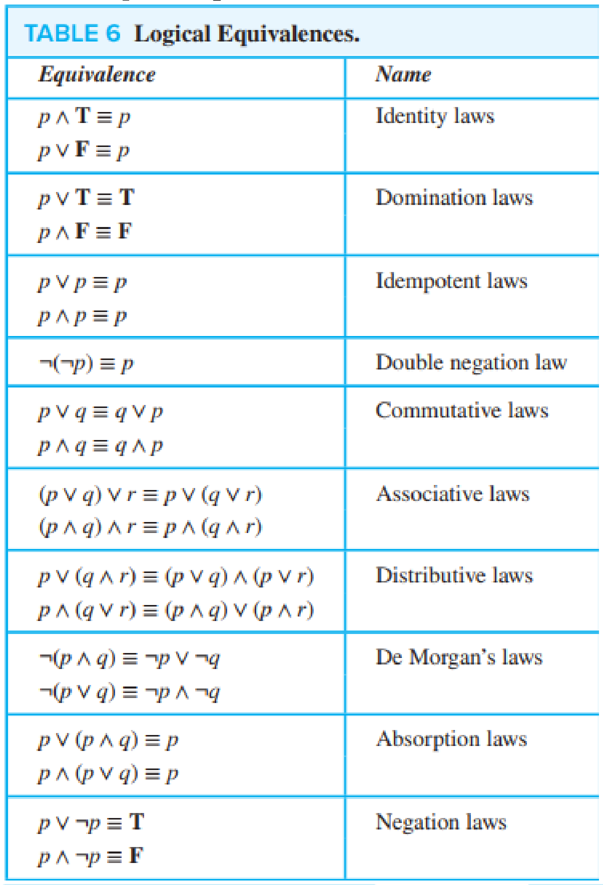


TABLE 6 **Logical Equivalences.**

Equivalence	*Name*
$p \wedge \mathbf{T} \equiv p$ $p \vee \mathbf{F} \equiv p$	Identity laws
$p \vee \mathbf{T} \equiv \mathbf{T}$ $p \wedge \mathbf{F} \equiv \mathbf{F}$	Domination laws
$p \vee p \equiv p$ $p \wedge p \equiv p$	Idempotent laws
$\neg(\neg p) \equiv p$	Double negation law
$p \vee q \equiv q \vee p$ $p \wedge q \equiv q \wedge p$	Commutative laws
$(p \vee q) \vee r \equiv p \vee (q \vee r)$ $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	Associative laws
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	Distributive laws
$\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\neg(p \vee q) \equiv \neg p \wedge \neg q$	De Morgan's laws
$p \vee (p \wedge q) \equiv p$ $p \wedge (p \vee q) \equiv p$	Absorption laws
$p \vee \neg p \equiv \mathbf{T}$ $p \wedge \neg p \equiv \mathbf{F}$	Negation laws

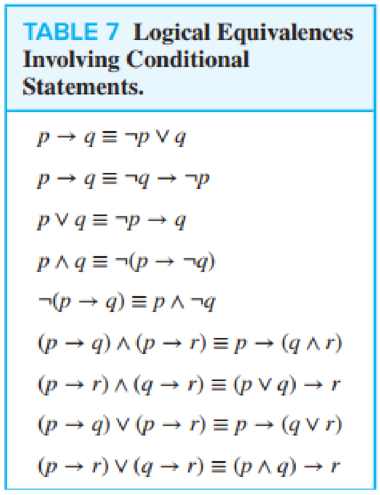


TABLE 7 **Logical Equivalences Involving Conditional Statements.**

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

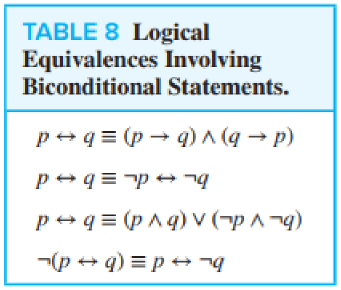


TABLE 8 **Logical Equivalences Involving Biconditional Statements.**

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

Biconditional statements: $p \leftrightarrow q$ or expanded to $(p \to q) \wedge (q \to p)$.
De Morgan: $\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$; $\neg(p \wedge q) \equiv (\neg p) \vee (\neg q)$ Propositional logic can be represented by gates, creating combinational circuits which can represent **any** logical expression.
Quantifiers:

$\forall x(P(x) \to Q(x)) \equiv$ *for all x, if P(x) then Q(x)*
$\exists x(P(x) \wedge Q(x)) \equiv$ *there exists an x such that P(x) and Q(x)*

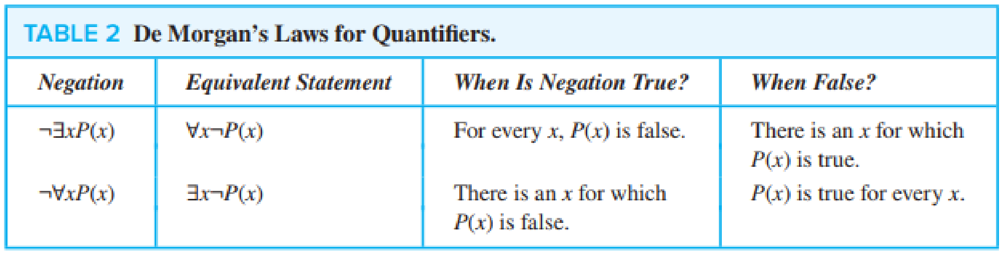


TABLE 2 De Morgan's Laws for Quantifiers.

Negation	*Equivalent Statement*	*When Is Negation True?*	*When False?*
$\neg\exists x P(x)$	$\forall x \neg P(x)$	For every x, $P(x)$ is false.	There is an x for which $P(x)$ is true.
$\neg\forall x P(x)$	$\exists x \neg P(x)$	There is an x for which $P(x)$ is false.	$P(x)$ is true for every x.

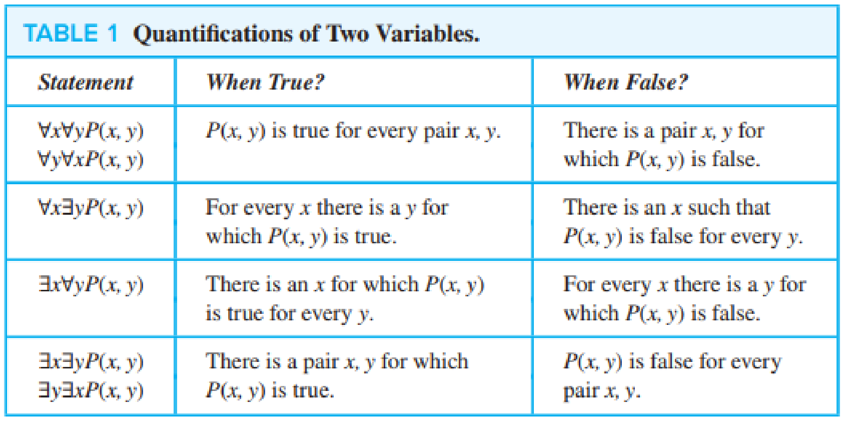


TABLE 1 Quantifications of Two Variables.

Statement	*When True?*	*When False?*
$\forall x \forall y P(x, y)$ $\forall y \forall x P(x, y)$	$P(x, y)$ is true for every pair x, y.	There is a pair x, y for which $P(x, y)$ is false.
$\forall x \exists y P(x, y)$	For every x there is a y for which $P(x, y)$ is true.	There is an x such that $P(x, y)$ is false for every y.
$\exists x \forall y P(x, y)$	There is an x for which $P(x, y)$ is true for every y.	For every x there is a y for which $P(x, y)$ is false.
$\exists x \exists y P(x, y)$ $\exists y \exists x P(x, y)$	There is a pair x, y for which $P(x, y)$ is true.	$P(x, y)$ is false for every pair x, y.

P(x),Q(x) are propositional functions and there is always a **domain** or **universe of discourse**, either implicit or explicitly stated,over which the variable ranges.
Negations of quantified propositions: $\neg\forall x P(x) \equiv \exists x \neg P(x); \neg\exists P(x) \equiv \forall x \neg P(x)$.

Theorem: A proposition that can be proved; **lemma:** a simple theorem, commonly used as part of a greater picture to prove other theorems; **proof:** A demonstration that a proposition is true, **collorary:** A proposition that can be proved as a consequence of a theorem that has just been proved. A collorary can be seen as "Side effects" of the prooved theorem.
A **valid** argument is an argument using correct rules of inference based on tautologies (something that will always give the **true** conclusion in **any** given scenario. I. E. a tautology is something that is always true for all possible combinations.)
An **invalid** argument can be referred to as a **fallacy**, such as affirming the conclusion, denying the hypothesis, begging the question or circular reasoning. They can lead to false conclusions.
Some rules of inference:

- $[p \wedge (p \rightarrow q)]$ Modus Ponens
- $[\neg q \wedge (p \rightarrow q)]$ Modus Tollens
- $[(p \rightarrow q) \wedge (q \rightarrow r)] \rightarrow (p \rightarrow r)$ Hypothetical syllogism (Transitivity)
- $[(p \vee q) \wedge (\neg p)] \rightarrow q$ Disjunctive syllogism
- $\{P(a) \wedge \forall x[P(x) \rightarrow Q(x)]\} \rightarrow Q(a)$ Universal modus ponens
- $\{\neg Q(a) \wedge \forall x[P(x) \rightarrow Q(x)]\} \rightarrow \neg P(a)$ Universal modus tollens
- $(\forall x P(x)) \rightarrow P(c)$ Universal instantiation
- $(P(c) arbitrary\ c) \rightarrow \forall x P(x)$ Universal generalization
- $(\exists x P(x)) \rightarrow (P(c)\ for\ some\ c)$ Existential instantiation
- $(P(c)\ for\ some\ element\ c) \rightarrow \exists x P(x)$ Existential generalization.

TABLE 1 **Rules of Inference.**

Rule of Inference	***Tautology***	***Name***
p $p \to q$ $\therefore q$	$(p \wedge (p \to q)) \to q$	Modus ponens
$\neg q$ $p \to q$ $\therefore \neg p$	$(\neg q \wedge (p \to q)) \to \neg p$	Modus tollens
$p \to q$ $q \to r$ $\therefore p \to r$	$((p \to q) \wedge (q \to r)) \to (p \to r)$	Hypothetical syllogism
$p \vee q$ $\neg p$ $\therefore q$	$((p \vee q) \wedge \neg p) \to q$	Disjunctive syllogism
p $\therefore p \vee q$	$p \to (p \vee q)$	Addition
$p \wedge q$ $\therefore p$	$(p \wedge q) \to p$	Simplification
p q $\therefore p \wedge q$	$((p) \wedge (q)) \to (p \wedge q)$	Conjunction
$p \vee q$ $\neg p \vee r$ $\therefore q \vee r$	$((p \vee q) \wedge (\neg p \vee r)) \to (q \vee r)$	Resolution

a Proofs

Trivial proof: A proof that $p \rightarrow q$ just shows that q is true witout using the hypothesis p.
Vacuous proof: A proof of $p \rightarrow q$ that just shows that the hypothesis p is false.
Direct proof: A proof of $p \rightarrow q$ that shows that the assumption of the hypothesis p implies the conclusion of q.
Proof by contraposition: A proof of $p \rightarrow q$ that shows that the assumption of the negation of the conclusion q implies the negation of the hypothesis p (in other words, proof of contrapositive).
Proof by contradiction: A proof of p that shows that the assumption of the negation of p leads to a contradiction.
Proof by cases: A proof of $(p_1 \vee p_2 \vee p_3 ... p_n) \rightarrow q$ that shows that each conditional statement $p_i \rightarrow q$ is true.
Statements of the form $p \leftrightarrow q$ require that both $p \rightarrow q$ and $q \rightarrow p$ be proved. It is sometimes necessary to give the two separate proof (usually a direct proof or a proof by contraposition); other times a string of equivalences can be constructed starting with p and ending with $q : p \leftrightarrow p_1 \leftrightarrow p_2 ... \leftrightarrow p_n \leftrightarrow q$.
To give a **constructive proof** of $\exists x P(x)$ is to show how to find an element x that makes $P(x)$ true. **Non-constructive existence proofs** are also possible, often using **proof by contradiction**.
One can **disprove** a universally quantified proposition $\forall x P(x)$ simply by giving a **counter example**, i.e. an object x such that $P(x)$ is **false**. One can, however, not proove it with such an example.
Fermat's last theorem: There are no positive integer solutions of $x^n + y^n = z^n \; if \; n > 2$.

An integer is **even** if it can be written as $2k$ for some integer k; an integer is **odd** if it can be written as $2k+1$ for some integer k. Every number is even or odd but not both. A number is **rational**, if it can be written as p/q with p being an integer and q strictly a non-zero integer.

2 Chapter 2

a Sets

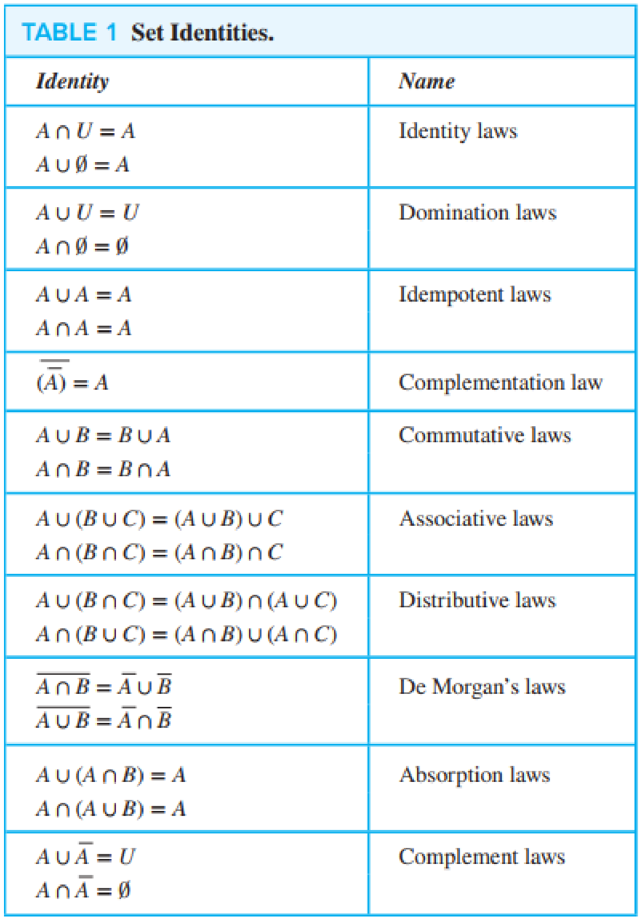


TABLE 1 Set Identities.

Identity	*Name*
$A \cap U = A$ $A \cup \emptyset = A$	Identity laws
$A \cup U = U$ $A \cap \emptyset = \emptyset$	Domination laws
$A \cup A = A$ $A \cap A = A$	Idempotent laws
$\overline{(\overline{A})} = A$	Complementation law
$A \cup B = B \cup A$ $A \cap B = B \cap A$	Commutative laws
$A \cup (B \cup C) = (A \cup B) \cup C$ $A \cap (B \cap C) = (A \cap B) \cap C$	Associative laws
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	Distributive laws
$\overline{A \cap B} = \overline{A} \cup \overline{B}$ $\overline{A \cup B} = \overline{A} \cap \overline{B}$	De Morgan's laws
$A \cup (A \cap B) = A$ $A \cap (A \cup B) = A$	Absorption laws
$A \cup \overline{A} = U$ $A \cap \overline{A} = \emptyset$	Complement laws

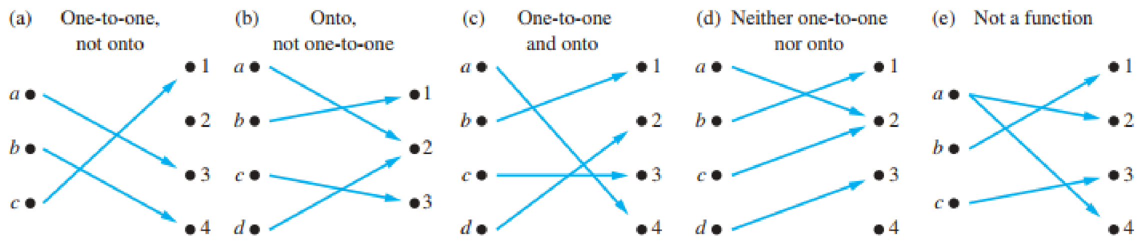
Empty set: A set with no elements, commonly denoted as $\emptyset$. Do not confuse this with the set only containing the empty set. The difference is that the empty itself is empty, whereas the set containing the empty set has a single element.
Subset: $A \subseteq B \equiv \forall x(x \in A \to x \in B)$, whereas a proper subset is $A \subset B \equiv (A \subseteq B) \wedge (A \neq B)$, in other words, B has at least one element different from the set A.
Equality of sets: $A = B \equiv (A \subseteq B \wedge B \subseteq A) \equiv \forall x(x \in A \leftrightarrow x \in B)$.
Power set: $\mathcal{P}(A) = \{B | B \subseteq A\}$, the set of all subsets of A. A set with n elements has 2^n subsets.
Cardinality: $|S|$, the number of elements in S.
Some specific sets in regards to cardinality: $\mathbb{R}$ is the set of real numbers, represented by either finite or infinite decimals;
$\mathbb{N}$ is the set of all natural numbers (eg. $\{0,1,2,3,4,5...\}$), $\mathbb{Z}$ is the set of integers $\{...-2,-1,0,1,2,...\}$ and can also be denoted with only the positive or negative subset. $\mathbb{Q}$ is the set of rational numbers, where $\{p/q | p,q \in \mathbb{Z} \wedge q \neq 0\}$, $\mathbb{Q}^+$ is the set of positive rational numbers and a subset of $\mathbb{Q}$.
Set operations: $A \times B = \{(a,b) | a \in A \wedge b \in B\}$ (**Cartesian Product**); $\overline{A}$ is the set of elements in the universe which are **not** in A (**complement**); $A \cap B = \{x | x \in A \wedge x \in B\}$ (**intersection**); $A \cup B = \{x | x \in A \vee x \in B\}$ (**union**); $A - B = A \cap \overline{B}$ (**difference**); $A \oplus B = (A - B) \cup (B - A)$, (**symmetric difference/xor**)
Inclusion-exclusion (simple case): $|A \cup B| = |A| + |B| - |A \cap B|$
De Morgan's laws for sets: $\overline{A \cap B} = \overline{A} \cup \overline{B}$; $\overline{A \cup B} = \overline{A} \cap \overline{B}$
A **function** f from A (**the domain**) to B (**the co-domain**) is an assignment of a unique element of B to each element of A. Write $f : A \to B$. Write $f(a) = b$ if b is assigned to a. **Range** of f is $\{f(a) | a \in A\}$; f is **onto/surjective** $\equiv$ range $(f) = B$; f is **one-to-one/injective** $\equiv \forall a_1 \forall a_2 [f(a_1) = f(a_2) \to a_1 = a_2]$

EXAMPLE 11 Use set builder notation and logical equivalences to establish the first De Morgan law $\overline{A \cap B} = \overline{A} \cup \overline{B}$.

Solution: We can prove this identity with the following steps.

$\overline{A \cap B} = \{x \mid x \notin A \cap B\}$	by definition of complement
$= \{x \mid \neg(x \in (A \cap B))\}$	by definition of does not belong symbol
$= \{x \mid \neg(x \in A \wedge x \in B)\}$	by definition of intersection
$= \{x \mid \neg(x \in A) \vee \neg(x \in B)\}$	by the first De Morgan law for logical equivalences
$= \{x \mid x \notin A \vee x \notin B\}$	by definition of does not belong symbol
$= \{x \mid x \in \overline{A} \vee x \in \overline{B}\}$	by definition of complement
$= \{x \mid x \in \overline{A} \cup \overline{B}\}$	by definition of union
$= \overline{A} \cup \overline{B}$	by meaning of set builder notation

If f is one-to-one **and** onto, it is **bijective** and the **inverse** function $f^{-1} : B \to A$ is defined by $f^{-1}(y) = x \equiv f(x) = y$.



If $f : B \to C$ and $g : A \to B$, then the **composition** $f \circ g$ is the function from A to C defined by $f \circ g(x) = f(g(x))$.
Rounding functions: $\lfloor x \rfloor$ is the largest integer less than or equal to x **floor function**; $\lceil x \rceil$ is the smallest integer greater than or equal to x **the ceiling function**.
Summation notation:

$$\sum_{n=1}^{n} a_i = a_1 + a_2 + a_3 + \ldots + a_n$$

Sum of first n positive integers:

$$\sum_{j=1}^{n} j = 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$

Sum of squares of first n positive integers:

$$\sum_{j=1}^{n} j^2 = 1^2 + 2^2 + \ldots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

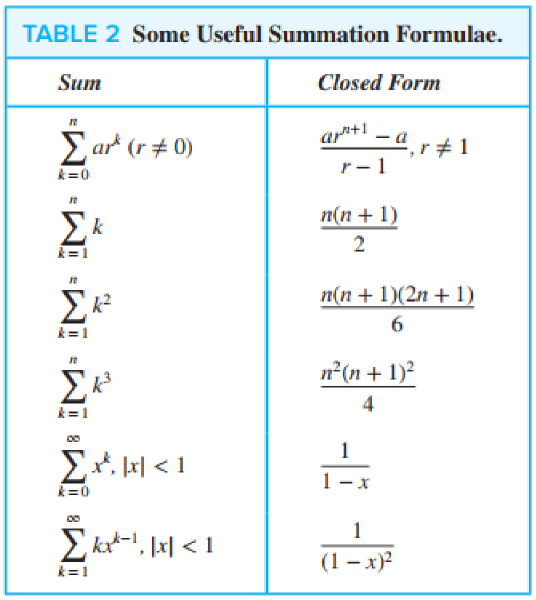


TABLE 2 Some Useful Summation Formulae.

Sum	*Closed Form*
$\sum_{k=0}^{n} ar^k \ (r \neq 0)$	$\frac{ar^{n+1} - a}{r - 1}, r \neq 1$
$\sum_{k=1}^{n} k$	$\frac{n(n+1)}{2}$
$\sum_{k=1}^{n} k^2$	$\frac{n(n+1)(2n+1)}{6}$
$\sum_{k=1}^{n} k^3$	$\frac{n^2(n+1)^2}{4}$
$\sum_{k=0}^{\infty} x^k, \lvert x \rvert < 1$	$\frac{1}{1-x}$
$\sum_{k=1}^{\infty} kx^{k-1}, \lvert x \rvert < 1$	$\frac{1}{(1-x)^2}$

Sum of geometric progression: *(I don't think we did this in the course)*

Two sets are said to have the **same cardinality** if there is a **bijection** between them. We can say that $|A| \leq |B|$ if there is a one-to-one function from A to B.
A set is *countable* if it is finite or there is a **bijection** from the positive integers to the set. **In other words**, if the elements of the set can be listed (e.g. $a_1, a_2, ...$). Sets of the latter type are called $countablyinfinite$ and the **cardinality of these sets are denoted with** $\aleph_0$. The empty set, the integers and the rational numbers **are countable**. The union of a countable number of countable sets is countable.
Schroder-Bernstein theorem: If $|A| \leq |B|$ and $|B| \leq |A|$ then it must be that $|A| = |B|$. This can be explained as if there is a one-to-one function from A to B and a one-to-one function from B to A, then there is a one-to-one and onto function from A to B.
Matrix Multiplication: The $(i, j)^{th}$ entry of **AB** is $\sum_{t=1}^{k} a_{it} b_{tj}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$, where **A** is an $m \times k$ matrix and **B** is a $k \times n$ matrix.
Identity matrix I_n with 1's on the main diagonal and 0's elsewhere is the multiplicative identity.
Cardinality arguments can be used to show that some functions are **uncomputable**.
Matrix addition (+), Boolean meet ($\wedge$) and join ($\vee$) are done entry-wise; Boolean matrix product ($\odot$) is like matrix multiplication using boolean operators.
Transpose: $\mathbf{A}^t$ is the matrix whose $(i, j)^{th}$ entry is a_{ij} (the $(j, i)^{th}$ entry of **A**);
A is **symmetric** if $A^t = A$;

3 Chapter 3

Algorithm are commonly expressed in **pseudo-code** when not directly implemented in a domain specific area.
Keywords for algorithms: {input, output, definiteness, correctness, finiteness, effectiveness, generality}.
Greedy algorithms: Will examine and pick the best choice at a given step. Not always the best.
Brute forcing: Specifically in discrete mathematics, this referres to examining the entire space of solutions and then determine the best one (very inefficient, sometimes necessary). Not explained in this course: **dynamic programming, probabilistic algorithms, divide-and-conquer**.
Halting problem: The unsolvable computing problem whether a program will halt given input. (Alan Turing for reference...)
Big-O: Half of inf102 is just this:
$f(x) = O(g(x))$ means $\exists C \exists k \forall x (x > k \rightarrow |f(x)| \leq C|g(x)|)$. Big-O of a sum is the largest (fastest growing) of the functions in the sum. Big-O of a product is the product of the big-O's of the factors. If f is $O(g)$, the g is $\Omega(f)$ "big-omega". If f is both big-O and big-Omega of g, then f is $\theta(g)$ "big-theta".
Little-O: We say that $f(x)$ is $o(g(x))$ if $\lim_{x\to\infty} f(x)/g(x) = 0$.
Powers grow faster than logs: $(\log n)^c$ is $O(x^d)$ but not the other way around, where c and d are positive numbers.
If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$, then $(f_1+f_2)(x)$ is $O(\max(g_1(x), g_2(x)))$ and $(f_1 f_2)(x)$ is $O(g_1(x)g_2(x))$.
$\log n!$ is $O(n \log n)$.
Time complexity: Binary search = $O(\log n)$ (cut half of possibilites at each step), linear search $O(n)$ (all input is examined exactly once), both have **space complexity (in terms of computer memory) O(1)** without taking the input into account. Bubble sort and insertion sort have $O(n^2)$.
Matrix multiplication has standard algorithm time complexity of $O(m_1 m_2 m_3)$ if the matrices have dimensions $m_1 \times m_2$ and $m_2 \times m_3$.
Efficient algorithms can reduce the complexity of multiplying two $n \times x$ matrices from $O(n^3)$ to $O(n^{\sqrt{7}})$ Important complexity classes include polynomials n^b, exponential (b^n for $b > 1$) and factorial ($n!$).
A problem that can be solved by an algorithm with polynomial worst-case time complexity is called **tractable**; otherwise **intractable**.
P=NP problem: The class **P** is the class of tractable problems. The class **NP** consists of the problems for which it is possible to check solutions (**not FIND solutions**) in polynomial time. This means that $P \subseteq NP$ yet the **P=NP** problem is unsolved because it has not been shown whether **P=NP**.

4 Chapter 4

Divisibility: $a|b$ means $a \neq 0 \wedge \exists c(a \cdot c = b)$ (a is a **divisor** or **factor** of b such that b is a multiple of a).
Base b **representiations:** $(a_{n-1}a_{n-2}...a_2a_1a_0)_b = a_{n-1}b^{n-1} + ... + a_2b^2 + a_1b + a_0$.
To convert from base 10 to base b, continually divide by b and record remainders as $a_0, a_1, a_2, ...$ ($b = 8$ is **octal**, $b = 16$ is **hexadecimal**, using A-F for 10-15). Convert from binary to octal by grouping bits by **threes**, from the right, to hexadecimal by grouping by fours; because $2^3 = 8$ and $2^4 = 16$.
Addition: of two **binary numerals** each of n bits ($(a_{n-1}a_{n-2}...a_2a_1a_0)_2$) requires $O(n)$ bit operations.
Multiplication: requires $O(n^2)$ bit operations if done naively, $O(n^{1.585})$ steps by more sophisticated algorithms.
Division "algorithm":

$$\forall a \forall d > 0 \exists q \exists r(a = dq + r \wedge 0 \leq r < d)$$

where q is the quotient and r is the remainder; we write a **mod** d for the remainder.
Example of the division "algorithm":

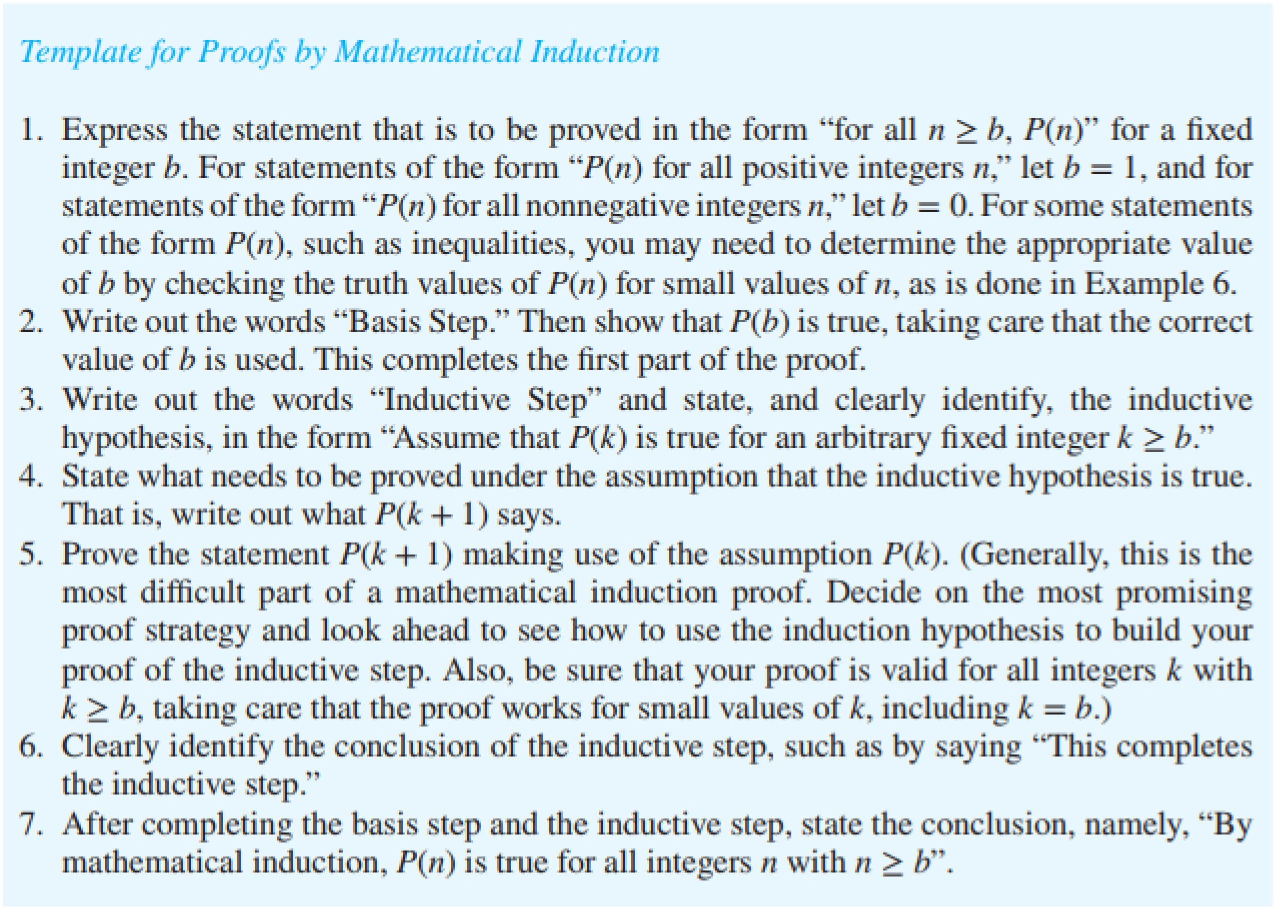
$$-18 = 5 \cdot (-4) + 2 \rightarrow -15 \textbf{ mod } 5 = 2$$

Congruent modulo m**:** $a \equiv b(\ mod\ m) \ \leftrightarrow\ m|a-b \ \leftrightarrow\ a\ mod\ m = b\ mod\ m$
One can do arithmetic in $\mathbb{Z}_m = \{0, 1, ..., m-1\}$ by working modulo m. There are fast algorithms for computing $b^n\ mod\ m$, based on successive squaring.
Integer $n > 1$ is said to be **prime** $\leftrightarrow$ its only factors are 1 and itself; otherwise it is referred to as a **composite**.
There are infinitely many **primes**, but it is not known whether there are infinitely many twin primes (**primes that differ by 2**), or whether every even positive integer greater than 2 is the sum of two primes (**Goldbach's conjecture**) or whether there are infinitely many **Mersenne primes** $\rightarrow$ **primes of the form** $2^p - 1$.
Naive test for primeness and test for prime factorization: To find prime factorization of n, successively divide it by all primes less than $\sqrt{n}$; if none is found, then **n is prime**. If a prime factor p is found, then continue the process to find the prime factorization of the remaining factor, namely n/p; this time the trial divisions can start with p. Continue until a prime factor remains.
Prime number theorem states that there are approximately $n/\ln(n)$ primes less than or equal to n.
Fundamental theorem of arithmetic: Every integer greater than 1 can be written as a product of one or more primes, and the product is unique except for the order of the factors. (A proof based on fact that if a prime divides a product of integers, then it divides at least one of those integers.)
Euclidean algorithm for greatest common divisor: $gcd(x,y) = gcd(y, x \textbf{ mod } y)$ if $y \neq 0$; $gcd(x,0) = x$.
Using extended Euclidean algorithm or working backwards, one can find **Bezout coefficients** and write $gcd(a,b) = sa + tb$.
Two integers are **relatively prime** if their greatest common divisor (gcd) is 1. The integers $a_1, a_2, ..., a_n$ are **pairwise relatively prime** $\leftrightarrow$ $gcd(a_i, a_j) = 1$ whenever $1 \leq i < j \leq n$.
Chinese reminder theorem: If $m_1, m_2, ...m_n$ are pairwise relatively prime, then the system $\forall i(x \equiv a_i(\ mod\ m_i))$ has unique solution modulo $m_1m_2...m_n$. An example of application of this: Handling very large integers on a computer.
Fermat's little theorem: $a^{p-1} \equiv 1(\ mod\ p)$ if p prime and does not divide a. The converse is not true; for example $2^{340} \equiv 1(\ mod\ 341)$ so $341(= 11 \cdot 31)$ is referred to as a **pseudo prime**;
If a and b are positive integers, then there exist integers s and t such that $as + bt = gcd(a,b)$ **linear combination**.
This theorem allows one to compute the **multiplicative inverse** $\overline{a}$ of $a\ modulo\ b$ (i.e $\overline{a}a \equiv 1(\ mod\ b)$) as long as a and b are relatively prime, which enables one to solve **linear congruences** $ax \equiv c(\ mod\ b)$.
A primitive root modulo a prime p is an integer r in $\mathbb{Z}_p$ such that every nonzero element of $\mathbb{Z}_p$ is a power of r.
Discrete logarithms: $\log_r a = e$ modulo p if r^e **mod** $p = a$ and $1 \leq e \leq p-1$
A common **hashing function:** $h(k) = k$ **mod** m, where k is the key.
Check digits for error-correcting codes like UPCs, involve modular arithmetic (?????????)
Pseudorandom numbers: can be generated by the **linear congruential method:** $x_{n+1} = (ax_n + c)$ **mod** m, where

x_0 is arbitrarily chosen **seed**. Then $\{x_n/m\}$ will be rather randomly distributed numbers between 0 and 1.
Shift cipher: $f(p) = (p+k)$ **mod** $26[A \leftrightarrow 0, B \leftrightarrow 1, ...]$. Caeser cipher used $k = 3$.
Affine cipher: Uses $f(p) = (ap+b)$ **mod** 26 with $gcd(a,26) = 1$.
RSA public key encryption system: An integer M representing the plaintext is translated into an integer C representing the ciphertext using the function $C = M^e$ **mod** b, where n is a public numbers that is the product of two large (100-digit or so) primes and e is a public number *relatively* prime to $(p-1)(q-1)$; the primes p and q are kept secret. Decryption is accomplished via $M = C^d$ **mod** n, where d is an inverse of e modulo $(p-1)(q-1)$. It is infeasible to compute d without knowing p and q, which are infeasible to compute from n.
Similar methods can be used for **key exchange protocols**, **digital signatures**, **signing stuff in general**.

5 Chapter 5

The well-ordering property: Every non-empty set of nonnegative integers has a "least element".
The principle of mathematical induction: Let $P(n)$ be a propositional function in which the domain (the universe of discourse) is the set of positive integers. Then if one can show that $P(1)$ is true (through **Base case/Base step**) and that for every positive integer k, the conditional statement $P(k) \to P(k+1)$ is true **(inductive step)**, then one has proved that $\forall n P(n)$. The hypothesis $P(k)$ in a proof of the inductive step is called the **inductive hypothesis**.
More generally, the indcution can start at any integer, and there could potentially be several base cases.



Template for Proofs by Mathematical Induction

1. Express the statement that is to be proved in the form "for all $n \geq b$, $P(n)$" for a fixed integer b. For statements of the form "$P(n)$ for all positive integers n," let $b = 1$, and for statements of the form "$P(n)$ for all nonnegative integers n," let $b = 0$. For some statements of the form $P(n)$, such as inequalities, you may need to determine the appropriate value of b by checking the truth values of $P(n)$ for small values of n, as is done in Example 6.
2. Write out the words "Basis Step." Then show that $P(b)$ is true, taking care that the correct value of b is used. This completes the first part of the proof.
3. Write out the words "Inductive Step" and state, and clearly identify, the inductive hypothesis, in the form "Assume that $P(k)$ is true for an arbitrary fixed integer $k \geq b$."
4. State what needs to be proved under the assumption that the inductive hypothesis is true. That is, write out what $P(k + 1)$ says.
5. Prove the statement $P(k + 1)$ making use of the assumption $P(k)$. (Generally, this is the most difficult part of a mathematical induction proof. Decide on the most promising proof strategy and look ahead to see how to use the induction hypothesis to build your proof of the inductive step. Also, be sure that your proof is valid for all integers k with $k \geq b$, taking care that the proof works for small values of k, including $k = b$.)
6. Clearly identify the conclusion of the inductive step, such as by saying "This completes the inductive step."
7. After completing the basis step and the inductive step, state the conclusion, namely, "By mathematical induction, $P(n)$ is true for all integers n with $n \geq b$".

Strong induction: Let $P(n)$ be a propositional function in which the domain (again, **universe of discourse**) is the set of positive integers. Then if one can show that $P(1)$ is true, and that for every positive integer k the conditional statement $[P(1) \wedge P(2) \wedge ... \wedge P(k)] \to P(k+1)$ is true **(inductive step)**, then one has proved $\forall n P(n)$. The hypothesis $\forall j \leq k P(j)$ in a proof of the inductive step is called the (**(strong) inductive hypothesis**). Again, the induction can start at any integer, and there can be several base cases.
Inductive/Recursive definitions (functions): Is a definition of a function f with the set of nonnegative integers as its domain: Specification of $f(0)$, together with, for each $n > 0$, a rule for finding $f(n)$ from values of $f(k)$ for $k < n$.
Example: $0! = 1$ and $(n+1)! = (n+1) \cdot n!$ **(factorial function)**
Inductive/Recusive definitions (sets): Definition of a set S: A rule specifying one or more particular elements of S, together with a rule for obtaining more elements of S from those already in it. It is understood that S consists precisely of those elements that can be obtained by applying these two rules.
Structural induction: can be used to prove facts about recursively defined objects.
Fibonacci numbers: $f_0, f_1, f_2, ... : f_0 = 0, f_1 = 1, f_n = f_{n-1} + f_{n-2}$ for all $n \geq 2$.
Lame's theorem: The number of divisions used by the Euclidean algorithm to find $gcd(a,b)$ is $O(\log b)$.
An algorithm is **recursive** if it solves a problem by reducing it to an instance of the same problem with smaller input. It is **iterative** if it is based on the repeated use of operations in a loop.

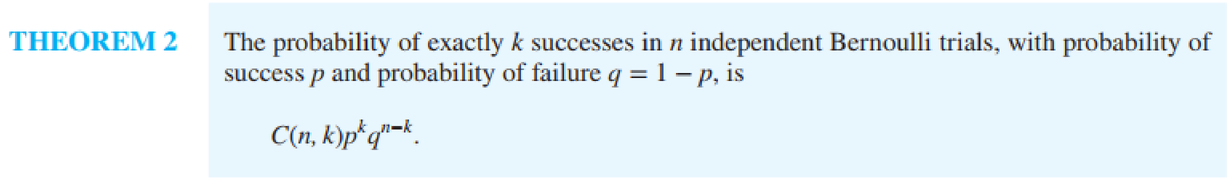
There is an efficient recursive algorithm for computing **modular powers** (b^n **mod** m), based on computing $b^{[\frac{n}{2}]}$ **mod** m.
Merge sort: is an efficient recursive algorithm for sorting a list: break the list into two parts, recursively sort each half, and then merge them together in order. It has $O(n \log n)$ time complexity in **all** cases.
A program segment S is **partially correct** with respect to **initial assertion** p and **final assertion** q, written $p\{S\}q$, if whenever p is true for the input values of S and S terminates, q is true for the output values of S.
A **loop invariant** for **while** *condition* S is an assertion p that remains true each time S is executed in the loop; i.e. $(p \wedge \ condition)\{S\}p$. If p is true before the program segment is executed, then p and $\neg condition$ are true after it terminates (if it terminates at all). In symbols, $p\{$**while** $condition\ S\}(\neg condition\ \wedge p)$.

6 Chapter 6

Sum rule: Given t mutually exclusive tasks, if task i can be done in n_i ways, then the number of ways to do exactly one of the tasks is $n_1 + n_2 + ... + n_t$.
Size of union of disjoint sets: $|A_1 \cup A_2 \cup ... \cup A_n| = |A_1| + |A_2| + ... + |A_n|$. ?? Duplicates ??
Two-set case of inclusion-exclusion: $|A \cup B| = |A| + |B| - |A \cap B|$. **Product rule:** If a task consists of successively performing t tasks, and if task i can be done in n_i ways (after previous tasks have been completed), then the number of ways to do the task is $n_1 \cdot n_2 ... n_t$.
A set with n elements has 2^n subsets (equivalently, there are 2^n bit strings of length n).
Tree diagrams: Can be used to organize counting problems.
Pigeonhole principle: If more than k objects are placed in k boxes, then some box will have more than 1 object.
Generalized version of the pigeonhole principle: If N objects are placed in k boxes, then some box will have at least $[N/k]$ objects.
Ramsey number: $R(m,n)$ is the smallest number of people there must be at a party so that there exist either m mutual friends or n mutual enemies (assuming each pair of people are either friends or enemies). $R(3,3) = 6$.
r-permutation of set with n objects, *ordered* arrangement of r of the objects from the set (no repetitions allowed); there are $P(n,r) = n!/(n-r)!$ such permutations. **r-combination** of set with n objects, *unordered* selection (i.e subset) of r of the objects from the set (no repetitions allowed); there are $C(n,r) = n!/[r!(n-r)!]$ such combinations. Alternative notation is $\binom{n}{r}$, also called the binomial coefficient.
Pascal's identity: *not in curriculum.*
Combinatorial identities: *not in curriculum.*
Number of **r-permutations** of an n-set **with repetitions allowed** is n^r; number of **r-combinations** of an n-set **with repetitions** allowed is $C(n+r-1,r)$. This latter value is the same as the **number of solutions in nonnegative integers** to $x_1 + x_2 + ... + x_n = r$.

7 Chapter 7

If all outcomes are equally likely in a **sample space** S with n outcomes, then the **probability of an event** E is $p(E) = |E|/n$; more generally, if $p(s_i)$ is probability of i^{th} outcome s_i, then $p(E) = \sum_{s_i \in E} p(s_i)$.
Probability distributions: satisfy these conditions: $0 \leq p(s) \leq 1$ for each $s \in S$ and $\sum_{s \in S} P(s) = 1$.
For **complementary event**, $p(\overline{E}) = 1 - p(E)$; for **union** of two events (either one or both happen), $p(E \cup F) = p(E) + p(F) - p(E \cap F)$; for **independent events**, $p(E \cap F) = p(E)p(F)$.
The **conditional probability** of E given F (probability that E will happen after it is known that F happened) is $p(E|F) = p(E \cap F)/p(F)$.
Bernoulli trials: If only two outcomes are **success** and **failure**, with $p(success) = p$ and $p(failure) = q = 1 - p$, then the **binomial distribution** applies, with probability of exactly k success in n trials being $b(k; n, p) = C(n, k)p^k q^{n-k}$.


THEOREM 2 The probability of exactly k successes in n independent Bernoulli trials, with probability of success p and probability of failure $q = 1 - p$, is

$$C(n, k)p^k q^{n-k}.$$

Bayes Theorem: *Dont think this was in the curriculum.*

8 Chapter 9 (Chapter 8 not in curriculum)

(Binary) Relations: A binary relation R from A to B: subset of the **cartesian** set $A \times B$. Write aRb for $(a,b) \in R$; relation **on** A is relation from A to A; graph of a function from A to B is a relation such that $\forall a \in A \exists$ exactly one pair (a,b) in the relation.
Relation R on a set A is **reflexive** if aRa for all $a \in A$.
Irreflexive if aRa for no $a \in A$.
Symmetric if aRb implies bRa for all $a,b \in A$.
Asymmetric if aRb implies that b is not related to a, for all $a,b \in A$.
Antisymmetric if $aRb \wedge bRa$ implies $a = b$ for all $(a,b) \in A$.
Transitive if $aRb \wedge bRc$ implies aRc for all $a,b,c \in A$.
Inverse relation to R is given by $bR^{-1}a \leftrightarrow aRB$.
If R is a relation from A to B and S is a relation from B to C, then the **composite** is the relation $S \circ R$ from A to C in which a is related to $c \leftrightarrow$ there exists a $b \in B$ such that aRb and bSc. $R^n = R \circ R \circ ... \circ R$.
N-ary relations: on domains $A_1, A_2, ..., A_n$ is a subset of $A_1 \times A_2 \times ... \times A_n$; databases using the relational data model are just sets of **n-ary** relations in which each n-tuple is called a **record** made up of **fields** (n is the **degree**).
A relation as matrix representation: A relation R on $A = \{a_1, a_2, ..., a_n\}$ can be represented by an $n \times x$ matrix M_R whose $(i,j)^{th}$ entry is 1 if a_iRa_j and is 0 otherwise. Reflexivity, symmetry, antisymmetry, transitivity can easily be read off the matrix. Boolean products of the matrices give the matrix for the composite ($M_{S \circ R} = M_R \odot M_S$).
A domain is a **primary key** if the corresponding field uniquely determines the record.
Equivalence classes: A relation R of set A is an **equivalence relation** if R is reflexive, symmetric and transitive. Usually, equivalence relations can be recognized by their definitions being of the form "two elements are related $\leftrightarrow$ they have the same [something]".
For each $a \in A$ the set of elements in A related to (i.e **equivalent to**) a is the **equivalence class** of a, denoted $[a]$.
Canonical example: Congruence modulo m with equivalence.
The set of equivalence classes partitions A into pairwise disjoint nonempty sets; conversely, every partition of A induces an equivalence relation by declaring two elements to be related if they are in the same set of the partition.

9 Chapter 10

Simple graph: $G = (V, E)$ consists of a nonempty set of **vertices (singularis: vertex)**, and a set of unordered pairs of distinct vertices called **edges**.
A multigraph: allows more than one **edge** joining the same pair of vertices (called **multiple edges or parallel edges**); E is just a set with an endpoint function f taking each edge e to its two *distinct* endpoints.
A pseudograph: A pseudograph is like a multigraph but enpoints $f(e)$ need not be distinct, allowing **loops**.
A directed graph is also called a **digraph**, it is like a simple graph except that the edges are directed (they have an arrow/ a direction attached to them) (each e is an $ordered pair$, and loops are allowed).
A directed multigraph is just like a multihgraph with parallel edges allowed, except that edges are directed (loops are allowed).
Given a directed graph, we can ignore order and look at the **underlying undirected graph.**
Graphs can be used to model relationships: herein examples such as: **acquaintances, food webs, telephone calls, networks, roads, internet, tournaments, organizations**.
Vertices joined by an edge are **adjacent** and the edge is **incident** to them.
A degree of a vertex (deg(v)) is the number of incident edges, with loops counted double. An **isolated vertex** has degree 0, a **pendant vertex** has degree 1.
A regular graph has all degrees equal.
In a **digraph**, $deg(v^- f)$ is the number of edges leading into v (**in-degree; v is terminal vertex**), and $deg(v^+)$ is number of edges leading out of v (**out-degree; v is initial vertex**).
Bipartite graph: vertex set can be partitioned into two nonempty sets with no edges joining vertices in same set.
The complete graph: K_n has n verttices and an edge joining every pair $(n(n-1)/2))$ edges in all.
Complete bipartite graph $K_{m,n}$ has $m+n$ vertices in parts of sizes m and n and an edge joining every pair of vertices in different parts (mn edges in all). The **cycle** C_n has n vertices and n edges, joined in a circle. **The wheel** W_n is C_n with one more vertex joined to these n vertices. The **cube** Q_n has all n-bit binary strings for vertices, with an edge between every pair of vertices differing in only one bit position.
(V,E) is subgraph of (W, F) if $V \subseteq W$ and $E \subseteq F$.
Union of two simple graphs is formed by taking union of corresponding vertex sets and corresponding edge sets.
Graphs can be represented by an adjancancy matrix: m in position (i, j) denotes m parallel edges from i to j.
Adjacency lists, incidence matrix (1 is position (i, j) says that vertex i is incident to edge j).
Graphs are isomorphic if there is a bijection between vertex sets that preserves all adjacencies and nonadjacencies. To show graphs are **not** isomorphic, find **invariant** on which they differ (e.g. degree sequence, existence of cycles).
A path of length n from u to v is a sequence of n edges leading successively from u to v; is a **circuit** if $n > 0$ and $u = v$, is **simple** if no edge occurs more than once.
A graph is connected if every pair of vertices is joined by a path. A digraph is **strongly connected** if every pair of vertices is joined by a path in each direction, **weakly connected** if underlying undirected graph is connected.
Components are maximal connected subgraphs.
Removal of cut edge (bridge) or **cut vertex (articulation point)** creates more components.

10 Chapter 11 (last one)

A **tree** is a connected undirected graph with no simple circuits; characterized by having unique simple path between every pair of vertices, and by being connected and satisfying $e = v - 1$. A **forest** is an undirected graph with no simple circuits; each component is a tree, and $e = v-$ number of components. Every tree has at least two vetices of degree 1.
A **rooted tree** is a tree with one vertex specified as root; can be viewed as directed graph away from root. If uv is a directed edge, then u is **parent** and v is a **child; ancestor, descendant, sibling** defined genealogically. Vetices without children are **leaves**; others are **internal**. Draw trees with root at the top, so that vetices occur at **levels**, with root at level 0; the **height** is maximum level number. A tree is **balanced** if all leaves occur only at bottom or next-to-bottom level, **complete** if **only** at bottom level. The **subtree rooted at** a is the tree involving a and all its descendants.
An **m-ary tree** is a rooted tree in which every vertex has at most m children (**binary** tree when $m = 2$); a **full** tree is when each child is a **right child** or **left child**, and subtree rooted at right [left] child is called **right [left] subtree**. A **Binary Search Tree (BST)** is a binary tree with a key at each vertex so that at each vertex, all keys in left subtree are less and all keys in right subtree are greater than key at the vertex. $O(\log n)$ algorithm for insertion and search in a **BST**.
Decision trees provide lower bounds on number of questions an algorithm needs to ask to accomplish its task for all inputs (e.g. coin-weighing, searching).
Binary trees can be used to encode **prefix codes**, binary codes for symbols so that no code word is the beginning of another code word. **Huffman codes** are efficient prefix codes for data compression.
Universal address system; root is labeled 0; children of root are labeled 1,2,...; children of vertex labeled x are labeled $x.1, x.2, \ldots$ Addresses are ordered using preorder traversal.
Preorder visits root, then subtrees (recursively) in preorder; **postorder** visits subtrees (recursively) in postorder, then root; **inorder** visists first subtree (recursively) in inorder, then root, then remaining subtrees (recursively) in inorder.
The **expression tree** for a calculation has constants at leaves, operations at internal vertices (evaluated by applying operation to values of its children.)
A spanningtree is a tree containing all vertices of a connected given graph; can be found by **depth-first** search (recursively search the unvisited neighbors) or **breadth-first** search (fan out). Edges not in depth-first search spanning tree **(back edges)** join vertices to ancestors or descendants. Edges not in bgreadth-frist search spanning tree **(cross-edges)** join vertices at same level or level differing by one. Depth-first search can be modified to implement **(backtracking (keeping track of where you went))** for exhaustive consideration of all cases of a problem (coloring a graph for example.)
Minimum spanning trees can be found in weighted graphs using greedy algorithms such as **Prims or Kruskals** algorithms.