CSI 2101 Summary

Winter 2013

Propositional Logic

- **PROPOSITION**: a statement with some definite meaning, and has a truth value. You may not know the actual truth value, and the truth value may depend on the situation or the context.
  - Proving propositions are equivalent or consistent: show they yield the same truth values, for example using a truth table.
- **TAUTOLOGY**: a compound proposition that is true regardless of the truth values of the atomic propositions
- **CONTRADICTION**: a compound proposition that is false regardless of the truth values of the atomic propositions
- **CONTINGENCY**: can be true or false, depending on the truth values of the atomic propositions.

Predicate Logic

- **PREDICATE**: modeled as a function $P(\cdot)$ from objects to propositions. If you have $P(x)$, x is the variable, and P is the property. Alternate definition: a function mapping an object x to proposition $P(x)$.
  - "The dog is sleeping." Same thing as saying x is sleeping, where x is the dog. P is the property of sleeping. So, we have $P(x)$.
  - Convention: lowercase → objects, uppercase → predicates.
- **PREDICATES IN COMPUTING**
  - **PRECONDITION**: what holds *before* the code segment. Can be denoted $Q(x, y)$.
  - **POSTCONDITION**: what holds *after* the code segment. Can be denoted $R(x, y)$.
    - To verify a postcondition holds, we must start from the assumption that the precondition holds and go over every stop of the code and examine what it does to see whether the postcondition is really true.
- **QUANTIFIERS**: provide a notation that allows us to count how many objects in the universe of discourse satisfy a given predicate.
  - **UNIVERSAL QUANTIFIER**: ∀ means "for all." $\forall x P(x) \equiv P(x_1) \wedge P(x_2) \wedge P(x_3) \wedge \ldots$
    - True if all x are true.
    - False if one x is false
  - **EXISTENTIAL QUANTIFIER**: ∃ means "there exists." $\exists x P(x) \equiv P(x_1) \vee P(x_2) \vee P(x_3) \vee \ldots$
    - True if there is one x that is true
    - False if all x are false
    - Can define new quantifiers, like $\exists! \, x P(x)$ means there exists exactly one x in the universe of discourse.
  - **FREE VARIABLE**: a variable that is not defined. Example: in $P(x)$, x is a free variable.
  - **BOUND VARIABLE**: when a variable is bounded by ∀ or ∃. Example: in $\forall x P(x)$, x is a bound variable.
    - Predicate becomes a proposition when all variables are bound.

Proofs

- **PROOF**: correct/well reasoned/logically valid, and complete/clear/detailed argument that rigorously and undeniably establishes the truth of a mathematical statement.
  - **THEOREM**: a statement that has been proven true.
  - **AXIOM, POSTULATE, HYPOTHESIS, PREMISE**: assumptions, often unproven defining the structures about which we are reasoning.

  - **RULES OF INFERENCE**: patterns of logically valid deductions from hypothesis to conclusions.
  - **LEMMA**: a minor theorem used as a stepping-stone to proving a major theorem.
  - **COROLLARY**: a minor theorem proved as an easy consequence of a major theorem.
  - **CONJECTURE**: a statement whose truth value has not been proven. May be widely believed to be true, regardless.
  - **THEORY**: a set of all theorems that can be proven from a given set of axioms.
  - **FORMAL PROOF**: sequence of statements ending in a conclusion. Statements preceding the conclusion are called premises. Each statement is either an axiom, or is derived from previous premises using a rule of inference.
  - **INFORMAL PROOFS**: Formal proofs are a pain and really tedious. We don't need that much detail. Obvious and easy steps are skipped or grouped together. Some axioms may be skipped (implicitly assumed). Must still be formal and precise to a certain degree.
- **PROVING EXISTENTIAL STATEMENTS**: if in the form $\exists x P(x)$, just find one x that satisfies $P(x)$.
- **DISPROVING THE NEGATION OF AN EXISTENTIAL STATEMENT**: if in the form $\neg\exists x P(x)$, just find one counterexample x that satisfies $P(x)$.
- **DISPROVING AN EXISTENTIAL STATEMENT**: to disprove, you need to prove the negation. In other words, if you're trying to disprove $\exists x P(x)$, then prove $\forall x\big(\neg P(x)\big)$.
- **PROVING A UNIVERSAL THEOREM**: if the theorem is in the form $p = \forall x\big(P(x) \to Q(x)\big)$, you can prove it by:
  - **EXHAUSTION**: works of the domain is finite or the number of x for which $P(x)$ holds is finite. Just prove $P(x)$ for every x in your domain.
  - **DIRECT PROOF**: assume $P(x)$ and derive $Q(x)$.
  - **CONTRADICTION**: show that $\neg p \to F$. So assume the negation of the statement to be proven, and show it leads to some sort of nonsense. The fact that you reached nonsense means an assumption was wrong, this assumption being that you assumed $\neg p$.
  - **CONTRAPOSITION**: prove the contrapositive directly. The contrapositive is defined as $\forall x\big(\neg Q(x) \to \neg P(x)\big)$.
    - *Advantage*: don't have to make potentially error-prone negation of the statement, and you know what you want to prove.
    - *Disadvantage*: only usable for statements that are universal and conditional.
- **CHECKLIST** for writing good proofs:
  - Be clean and complete
  - State the theorem to be proven
  - Clearly mark the beginning of the proof
  - Make the proof self-contained (introduce and identify all the variables)
  - Write in full sentences
  - Give reason for each assertion (by hypothesis, by definition, by substitution…)
  - Use connecting words (the, thus, hence, therefore, observe, note, let…)
- **STRATEGY**: Say you're given if $P(x)$, then $Q(x)$. Imagine elements that satisfy $P(x)$. Ask yourself, "Must they satisfy $P(x)$?" If the answer is:
  - "Yes." Use the reasons why you feel this way as a basis for a direct proof.
  - "Not really." Think why, you may come up with a counterexample. If you can't find a counterexample, think of why.
    - Maybe from assuming $P(x) \wedge \neg Q(x)$, you can derive a contradiction.
    - Maybe from assuming $\neg Q(x)$, you can derive $\neg P(x)$.

- **NORMAL INDUCTION**: In order to prove statement $P(n)$ by induction, we must first prove the base case, i.e. prove that $P(1)$ is true. Then, we assume $P(k)$ is true (induction hypothesis), and prove that $P(k) \rightarrow P(k+1)$, called the inductive step.
  If the base case and inductive step can be proved, then we say $P(n)$ is proved by mathematical induction.
  - Is a rule of inference that says:
    $P(1)$
    $\underline{\forall\big(P(k) \rightarrow P(k+1)\big)}$
    $\therefore \forall n\, P(n)$
- **WELL-ORDERED**: A set S is "well-ordered" if every non-empty subset of S has a least element.
- **STRONG INDUCTION**: In order to prove statement $P(n)$ by strong induction, we must first prove the base case, i.e. prove that $P(1)$ is true. Then, we assume $P(1) \wedge \ldots \wedge P(k)$ is true (induction hypothesis), and prove that $P(1) \wedge \ldots \wedge P(k) \rightarrow P(k+1)$, called the inductive step.
  If the base case and inductive step can be proved, then we say $P(n)$ is proved by strong induction.
  - Is a rule of inference that says:
    $P(1)$
    $\underline{\forall(P(1) \wedge \ldots \wedge P(k) \rightarrow P(k+1))}$
    $\therefore \forall n\, P(n)$
- **STRUCTURAL INDUCTION**: In order to prove statement $P(x)$ by structural induction, we must first prove the base case, i.e. prove that $P(x)$ is true for each element x in the base definition of the set S. For the inductive step, for every way to construct $y_1, \ldots, y_k$, show that $P(y_1) \wedge \ldots \wedge P(y_k) \rightarrow P(x)$.

Complexity

- **COMPLEXITY**: the number of steps that it takes to transform the input data into the desired output. It is a function of the size of the input (or size of the instance). In general, we will deal with the worst case complexity.
- The main question is how the complexity behaves asymptotically, i.e. when the problem sizes tend to infinity.
- We're not interested in the exact time it takes to run, we usually just want to compare two algorithms. We will choose the right algorithm based on which has the better growth rate.
- **BIG-OH**: We say $f(n)$ is "big oh" of $g(n)$ if there exists two constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for $n \geq n_0$. To prove a function $f(n)$ is $O\big(g(n)\big)$, find these two constants c and n_0
- $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n) < \mathrm{n!}$. Only the largest term matters. It means that an algorithm has the complexity of **AT MOST** $g(n)$**.**
- **BIG-OMEGA:** We say $f(n)$ is "omega" of $g(n)$, or $f(n) \in \Omega(g(n))$ if $f(n) \geq cg(n)$ for all $n \geq n_0$. It means an algorithm has the complexity of **AT LEAST** $g(n)$.
- **BIG-THETA**: We say $f(n)$ is "theta" of $g(n)$, or $f(n) \in \theta(g(n))$ if it is $O\big(g(n)\big) \& \Omega\big(g(n)\big)$. It means the complexity **IS EXACTLY** $g(n)$.
- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & f(n) \in O\big(g(n)\big) \\ c > 0 & f(n) \in \theta\big(g(n)\big) \\ \infty & f(n) \in \Omega\big(g(n)\big) \end{cases}$

Recursion

- **RECURSION**: the practice of defining an object in terms of itself, or a part of itself. An inductive proof establishes the truth of $P(k+1)$ recursively in terms of $P(k)$.
  - Can define a recursive function $f(n)$, and prove the explicit value of $f(n)$ using induction.
  - Must be careful that it's well-defined
    - It is defined for each element in the domain
    - It is defined unambiguously
  - An infinite set S may be defined recursively by giving:
    - A small finite set of base elements of S
    - A rule for constructing new elements of S from previously established elements
    - Implicitly, S has no other elements but these
- **FIBONACCI NUMBERS**: They are defined by the function $f(1) = 1,\ f(2) = 1,\ f(n) = f(n-1) + f(n-2)\ \ \forall n \geq 3$.
  - **THEOREM**: $\forall n \geq 3,\ f(n) > \alpha^{n-2}$ where $\alpha = (1+\sqrt{5})/2$. Proof is done by induction.
- Recursive definitions can be used to describe algorithms. Typical problem solving approach: solve the problem for smaller/simpler sub-problems and obtain the result from that.

Program Correctness

- **CORRECT**: a program is correct if it produces the correct output for every input.
- **PARTIALLY CORRECT**: a program is correct if it produces the correct output for every input for which the program eventually halts.
- **PRECONDITION**: the initial assertion p is the condition that the program's input (its initial state) is guaranteed (by its user) to satisfy.
- **POST-CONDITION**: the final assertion q is the condition that the output produced by the program (its final state) is required to satisfy.
- **HOARE TRIPLE NOTATION**: the notation $p\{S\}q$ means that for all inputs I such that $p(I)$ is true, if program S (given input I) halts and produces output $O = S(I)$, then $q(O)$ is true. That is, S is partially correct with respect to specification p, q. Deduction rules for Hoare Triple statements:
  - **COMPOSITION RULE**: if program S_1 given condition p produces condition q, and S_2 given condition q produces r, then the program S_1 followed by S_2 if given p yields r.
    $p\{S_1\}q$
    $q\{S_2\}r$
    $\therefore\ p\{S_1,\ S_2\}r$
  - **IF STATEMENTS**: Let C be the condition. Then
    $(p \wedge C)\{S\}q$
    $(p \wedge \neg C) \rightarrow q$
    $\therefore\ p\{\text{if } C \text{ then } S\}q$
  - **IF-THEN-ELSE RULE**: Let C be the condition. Then
    $(p \wedge C)\{S_1\}q$
    $(p \wedge \neg C)\{S_2\}q$
    $\therefore\ p\{\text{if } C \text{ then } S_1 \text{ else } S_2\}q$
  - **LOOP INVARIANTS**: Let C be the condition. For a while loop $\text{while } C,\ S$, we say that p is a loop invariant of this loop if $(p \wedge C)\{S\}p$. So if p is true before executing the body, then p remains true after.
    $(p \wedge C)\{S\}q$
    $\therefore\ p\{while\ C,\ S\}(\neg C \wedge p)$

- **PROVING THAT A LOOP HALTS**: For each iteration i, we associate an integer k_i such that $\{k_0, k_1, \dots, \}$ is a decreasing sequence. Because of the well-ordering principle of any subset of integers, this sequence of integers $\{k_i\}$ is finite.

Number Theory

- **DIVIDES**: a divides b, denoted $a|b$ for $a, b \in \boldsymbol{Z}$ if there is an integer c such that $c \cdot a = b$.
  - **FACTOR/DIVISOR**: if $a|b$, then a is a factor/divisor of b.
  - **MULTIPLE**: if $a|b$, then b is a multiple of a.
  - **FACTS**:
    - $a|0$
    - If $a|b$ and $a|c$, then $a|(b + c)$
    - If $a|b$, then $a|(bc)\ \ \forall c \in \boldsymbol{Z}$
    - If $a|b$ and $b|c$, then $a|c$
    - If $a, b, c \in \boldsymbol{Z}$ such that $a|b$ and $a|c$, then $a|(mb + nc)$ when $m, n \in Z$
- **DIVISION ALGORITHM**: Let $a \in \boldsymbol{Z}$ and $d \in \boldsymbol{Z}^+$. Then there are unique integers q and r, with $0 \leq r < d$, such that $a = dq + r$. r = remainder, d = dividend, q = quotient.
- **CONGRUENCE**: If $a, b \in \boldsymbol{Z}$ and $m \in \boldsymbol{Z}^+$, then a is congruent to b modulo m (denoted $a \equiv b \bmod m$ or $a \bmod m \equiv b \bmod m$) if $m|(a - b)$.
  - **FACTS**:
    - $a \equiv b \bmod m$ if and only if $\exists k \in Z$ such that $a = b + km$
    - If $a \equiv b \bmod m$ and $c \equiv d \bmod m$, then $a + c \equiv b + d \bmod m$ and $ac \equiv bd \bmod m$
    - We can add or multiply (not divide) the two sides of the congruence by the same integer.
  - **ADDITION MOD m**: the operation $+_m$ is defined by $a +_m b = (a + b) \bmod m$
    - **ADDITIVE INVERSE**: if $a \neq 0 \in \boldsymbol{Z_m}$, $then\ m - a$ is the additive inverse of a modulo m. Note that 0 is its own additive inverse.
    - **ADDITIVE IDENTITY 0**: if $a \in Z_m$, then $a +_m 0 = a$.
  - **MULTIPLICATION MOD m**: the operation $\cdot_m$ is defined by $a \cdot_m b = (a \cdot b) \bmod m$
    - **MULTIPLICATIVE INVERSE**: if $ad = 1 \bmod m$, then d is called the multiplicative inverse. Note that this inverse does not always exist.
    - **MULTIPLICATIVE IDENTITY 1**: if $a \in Z_m$, then $a \cdot_m 1 = a$.
- **PRIME**: the number $p > 2$ is prime if there are exactly two positive factors: 1 and itself.
  - **FUNDAMENTAL THEOREM OF ARITHMETIC**: Every positive integer $n \geq 2$ can be uniquely written as a prime or as the product of two or more primes, where the primes are written in order of non-decreasing size. The non-primes are called composite integers.
  - **THEOREM**: If n is a composite integer, then n has a prime divisor less than or equal to $\sqrt{n}$. Can be used to show that a number is prime.
  - Primes are the building blocks of the natural numbers.
- **MERSENNE NUMBER**: a number of the form $M = 2^n - 1, n \in \boldsymbol{N}$.
  - **MERSENNE PRIME**: a prime number of the form $P = 2^p - 1$, where p is a prime number.
  - **THEOREM**: If M is a Mersenne prime, then $r = \frac{M(M+1)}{2}$ is a perfect number (i.e. its divisors equal itself).

- **GREATEST COMMON DIVISOR**: denoted $\gcd(a, b)$ where a, b are integers, it is the largest integer d such that $d|a$ and $d|b$.
  - **RELATIVELY PRIME**: two numbers a and b are relatively prime if $\gcd(a, b) = 1$.
- **LEAST COMMON MULTIPLE**: denoted $\text{lcm}(a, b)$ where a, b are integers, it is the smallest positive integer that is divisible both by a and b.
- **THEOREM**: If a and b are integers, then $ab = \gcd(a, b) \cdot \text{lcm}(a, b)$.
- **EUCLID**: $\forall a, b,\ a > b > 1, \gcd(a, b) = \gcd((a \bmod b),\ b)$
- **THEOREM**: Let a, b, q, r be integers. If we have $a = bq + r$, then $\gcd(a, b) = \gcd(b, r)$.
- **THEOREM**: $\forall a, b \in \boldsymbol{Z}^+, \exists s, t$ such that $\gcd(a, b) = sa + tb$
- **LEMMA**: $\forall a, b, c \in \boldsymbol{Z}^+$, if $\gcd(a, b) = 1$ and $a|bc$, then $a|c$.
- **LEMMA**: If p is prime and $p|a_1a_2 \dots a_n,\ a_i \in \boldsymbol{Z}$, then there exists an i such that $p|a_i$.
- **THEOREM**: If $ac = bc \bmod m$, and $\gcd(c, m) = 1$, then $a \equiv b \bmod m$.
- **LINEAR CONGRUENCE**: a congruence of the form $ax = b \bmod m$. Solving the congruence is finding the x's that satisfy it. You can solve it by finding a' such that $x = a'b$, where a' is the multiplicative inverse of a. The inverse is unique.
  - **THEOREM**: If n is prime, then $2^{n-1} \equiv 1 \bmod n$. Note that the converse is not true.
- **FERMAT'S LITTLE THEOREM**: If p is prime and a is an integer not divisible by p, then $a^{p-1} \equiv 1 \bmod p$. Furthermore, for every integer a, $a^p = a \bmod p$.
- **CARMICHAEL NUMBER**: a composite number n such that $a^{n-1} \equiv 1 \bmod n$ for all a relatively prime to n. These numbers are important because they fool the Fermat primarily test; the are "Fermat liars." They have at least three prime factors.

Hashing Functions

- Two functions:
  1. Indexing hash tables (data structures which support $O(1)$ time access)
  2. Creating short, unique IDs for long documents (used in digital signatures)
- **HASH FUNCTION**: a hash function $\text{h}: A \to B$ is a map from a set A to the smaller set B. An effective hash function should have the following properties:
  - It should be onto (cover its codomain B)
  - Efficient to calculate
  - Elements of B should be generated with roughly uniform probability, i.e. $\forall b_1, b_2 \in B,\ |\text{h}^{-1}(b_1)| \approx |\text{h}^{-1}(b_2)|$.
  - The map should appear random, so similar elements are not all mapped to the same or similar elements of B.
  - A cryptographically secure hash function: given a $b \in B$, the problem of finding an $a \in A$ such that $\text{h}(a) = b$ should have an average-case time complexity of $\Omega(|B|^c)$ for some $c > 0$. This way, you can't fake a document with the same ID.
- **MODULAR HASH FUNCTIONS**: Let A, B be bounded subsets of $\boldsymbol{N}$ with upper bounds a_{lim} and b_{lim} respectively. Then an acceptable hash function from A to B is $\text{h}(a) = a \bmod b_{lim}$
  - Pros
    - h is onto
    - When $a_{lim} \gg b_{lim}$, then each $b \in B$ has a preimage of about the same size. $|\text{h}^{-1}(b)| = \left\lfloor \frac{a_{lim}}{b_{lim}} \right\rfloor$ or $\left\lceil \frac{a_{lim}}{b_{lim}} \right\rceil$

  - Cons
    - Not very random
    - Not cryptographically secure
- **DIGITAL SIGNATURES**: Given a document a and a cryptographically secure function h:
  - Signing procedure
    - Quickly compute the hash of the document a, which is $b = \mathrm{h}(a)$.
    - Compute a function $c = f(b)$, which is only known to the signer (this step is slow, that's why we have to hash the document first)
    - Send the signature c
  - Signature verification process
    - If we have a signature c, we find a's hash $b = \mathrm{h}(a)$
    - Compute $b' = f^{-1}(c)$. This is possible if f^{-1} is public, but keep in mind that f isn't.
    - Compare b to b'. If $b = b'$, then the signature is valid.
- **PSEUDORANDOM NUMBERS**: called pseudorandom because computers cannot create truly random numbers.
  - **LINEAR CONGRUENTIAL METHOD**:
    - Choose 4 integers.
      - **SEED**: Starting value x_0
      - **MODULUS**: maximum possible value m
      - **MULTIPLIER**: a number a such that $2 \leq a < m$
      - **INCREMENT**: a number c such that $0 < c < m$
    - To generate a sequence of pseudorandom numbers $\{X_n | 0 \leq x < m\}$, we apply the formula $x_{n+1} = (ax_n + c) \bmod m,\ n \geq 0$.
    - The sequence eventually repeats itself, but it selects all the possible numbers before doing so. Most of the algorithms today use $m = 2^{32} - 1$, so you need to go through 4 billion numbers before the sequence starts repeating itself.

Cryptography

- **CAESAR CIPHER**: the function defined by $f(p) = (p + 3) \bmod 26$, where $p \in \{0,1, \ldots,25\}$, $A = 0,\ B = 1,\ \ldots,\ Z = 25$ and $f^{-1}(p) = (p - 3) \bmod 26$. You're subbing one letter with another. It's called a substitution cipher.
- **NORMAL CRYPTOGRAPHY**: Communicating parties both need to know a secret key k. The sender encodes the message m using the key k and gets the ciphertext $c = f(m, k)$. The receiver decodes c using the key k and recovers the original message $m = g(c, k)$. The problem is how to securely distribute the key k?
- **RSA**: brings the idea of public key cryptography. The receiver lets everyone know its public key k, and everyone can send an encoded message $c = f(m, k)$ to the receiver, where f is a known encoding function. But, only the receiver knows the secret key k' that can decode the ciphertext using $m = g(c, k')$, where g is also known.
  - Let p, q be very large primes.
  - **PUBLIC KEY**: it's the pair (n, e) where $n = pq$ and e is relatively prime to $(p - 1)(q - 1)$.
    - Anyone can encode, only receiver can decode.
    - Also could be made so anyone can decode, and know it came from the only person who knows how to encode.

- **ENCODING FUNCTION**: it's $f(m,k) = m^e \bmod n$. Note that every message m can be split into $m_1, m_2, \ldots$ messages, which you can individually encode .
- **PRIVATE KEY**: it's the number d, which is the multiplicative inverse of $e \bmod (p-1)(q-1)$
- **DECODING FUNCTION**: it's $g(c,d) = c^d \bmod n$
- **BASIC IDEA**: knowing n, it's hard to find p and q, and thus it's hard to find d.
- Summary:
  - To encrypt a message $M < n$, compute $C = M^e \bmod n$
  - To decrypt a message C, compute $M = C^d \bmod n$.

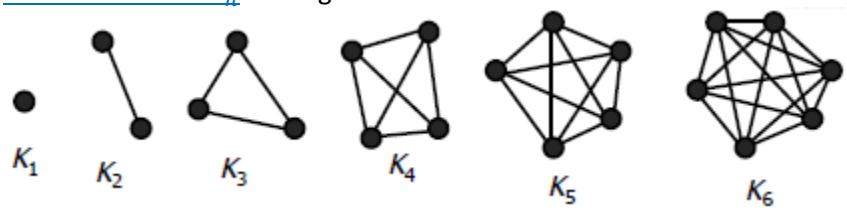
- **PRIVATE-KEY CRYPTOSYSTEM**: the same secret "key" string is used to both encode and decode messages. Problem with this is security.
- **PUBLIC-KEY CRYPTOSYSTEM**: two complementary keys. One decrypts, and one encrypts. One can be made public, and one private. RSA is the most famous public-key cryptosystem.

Recurrence Relations

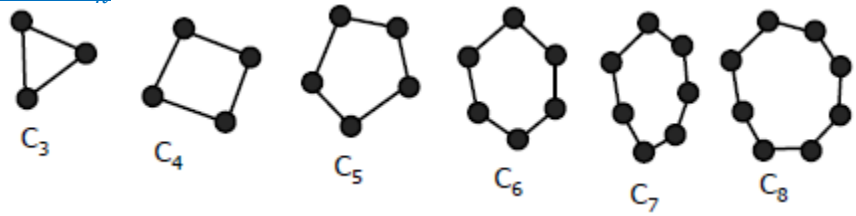
- **LINEAR HOMOGENEOUS RECURRENCE RELATION OF DEGREE k WITH CONSTANT COEFFICIENTS**:
  $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$
  - Linear means no a_i is multiplied with an a_j
  - Homogeneous means there are no additional terms to the a_i
  - Constant coefficients means that the c_i are not functions of n.
  - Degree k means there are k constants
- **SOLVING**:
  Find the characteristic equation. For $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$, the characteristic equation is $r^k = c_1 r^{k-1} + c_2 r^{k-2} + \cdots + c_k$.
  - Find the roots of the characteristic equation.
    - If $r = r_1 = r_2 = \cdots = r_k$, then $a_n = \alpha_0 r^n + \alpha_1 n r^n + a_2 n^2 r^n + \cdots + \alpha_k n^k r^n$
    - If $r_1 \neq \cdots \neq r_k$, then $a_n = \alpha_0 (r_1)^n + \cdots + \alpha_k (r_k)^n$
  - To solve for α_i's, you must plug in the initial conditions given with the problems.
- **LINEAR NON-HOMOGENEOUS RECURRENCE RELATION OF DEGREE k WITH CONSTANT COEFFICIENTS**:
  $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + F(n)$
  - Same as homogenous ones with an addition of a function of n, $F(n)$.
- **SOLVING**:
  - Solve the homogeneous recurrence relation in the same fashion as before. The solution to this is denoted $a_n^{(\mathrm{h})}$.
  - Solve $F(n)$, giving you $a_n^{(p)}$. We can do this if $F(n)$ is in the form $\big(p(n)\big)s^n$, where $p(n)$ is a polynomial of degree t.
    - Find s by putting $F(n)$ into that form. The, find the roots $r_1, \ldots, r_t$ of the polynomial $p(n)$ inside the brackets.
      - If $s \neq r_i$, the particular solution is of the form $(p_t n^t + \cdots + p_1 n + p_0)s^n$
      - If $s = r_{i_1} = \cdots = r_{i_m}$, then we have a root with multiplicity m, and the particular solution is of the form $n^m(p_t n^t + \cdots + p_1 n + p_0)s^n$.
    - Solve for $p_0, p_1, \ldots, p_t$, and put them into the applicable form to get $a_n^{(p)}$
    - Final solution is $a_n = a_n^{(\mathrm{h})} + a_n^{(p)}$

Graphs

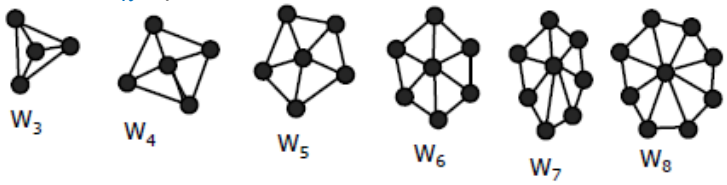
- **SIMPLE GRAPHS**: Correspond to symmetric and reflexive binary relations R. A simple graph $G = (V, E)$ has:
  - **VERTICES**: A set V of vertices or nodes (V corresponds to the universe of the relation R),
  - **EDGES**: A set E of edges / arcs / links: unordered pairs of [distinct] elements $u, v \in V$, such that $u\mathrm{R}v$.
  - If we have the edge $e = \{u, v\}$ then we have the following definitions:
    - **INCIDENT**: e is incident with u and v.
    - **ADJACENT**: u is adjacent to v.
    - **CONNECTS**: e connects u and v.
    - **ENDPOINT**: u and v are endpoints of e.
- **DIRECTED GRAPH**: A directed graph (V, E) consists of a set of vertices V and a binary relation (need not be symmetric) E on V.
- **DEGREE**: the degree of a vertex $v \in V$ is denoted $\deg(v)$ and it represents the number of incident edges that v has. Loops are counted twice.
  - **ISOLATED**: a vertex with degree 0.
  - **PENDANT**: a vertex with degree 1.
  - **HANDSHAKING THEOREM**:
    - If $G = (V, E)$ is an undirected graph, then $\sum_{v \in V} \deg(v) = 2|E|$.
      - **COROLLARY**: Any undirected graph has an even number of vertices of odd degree.
    - If $G' = (V', E')$ is a directed graph, then $\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = \frac{1}{2}\sum_{v \in V} \deg(v) = |E|$.
      - **INDEGREE**: The number of edges going into v, denoted $\deg^-(v)$.
      - **OUTDEGREE**: The number of edges coming from v, denoted $\deg^+(v)$.
      - $\sum_{v \in V} \deg(v) = \sum_{v \in V} \deg^-(v) + \sum_{v \in V} \deg^+(v)$
- **COMPLETE GRAPH K_n**: all edges are connected.



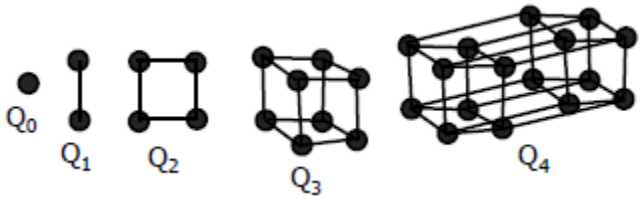
- **CYCLES C_n**: all consecutive vertices are connected.



- **WHEELS W_n**: cycles with an extra vertex in the middle, which is connected to all the other vertices.



- **HYPERCUBE Q_n**: constructed by doubling the last hypercube and connecting all the edges.
  - Q_n has 2^n vertices and $n2^{n-1}$ edges.



- **BIPARTITE**: A graph $G = (V, E)$ is bipartite if $V = V_1 \cup V_2$ where $V_1 \cap V_2 = \emptyset$, and for all $e \in E$, there exists a $v_1 \in V_1$ and a $v_2 \in V_2$ such that $e = \{v_1, v_2\}$.
  - **COMPLETE BIPARTITE GRAPH $K_{m,n}$**: it's a bipartite graph where $|V_1| = m$, $|V_2| = n$, and $E = \{\{v_1, v_2\} | v_1 \in V_1 \text{ and } v_2 \in V_2\}$.
- **SUBGRAPH**: a graph $H = (V', E')$ is a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.
- **ISOMORPHISM**: two graphs are isomorphic if and only if they are identical except for their node names. If H is isomorphic to G, then H is an isomorphism of G. An alternate, more formal definition is: $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if and only if there exists a bijective function $f: V_1 \to V_2$ such that for all $a, b \in V_1$, a and b are adjacent in G_1 if and only if $f(a)$ and $f(b)$ are adjacent in V_2. The function f is called the renaming function.
  - Easy tools to check if graphs are not isomorphic:
    - If $|V_1| \neq |V_2|$, then the graphs are not isomorphic.
    - If $|E_1| \neq |E_2|$, then the graphs are not isomorphic.
    - If the degree sequence in both graphs are not the same, the graphs are not isomorphic.
    - For every proper subgraph g in one graph, there is a proper subgraph of the other graph that is isomorphic to g. If this does not hold, the graphs are not isomorphic.
    - You should be able to draw one graph with the vertices of the other.
    - If one graph is connected and the other isn't, they are not isomorphic.
    - If a circuit exists in one graph but not the other, they are not isomorphic.
- **PATH**: in an undirected graph, a path of length n from u to v is a sequence of adjacent edges going from vertex u to vertex v.
- **CIRCUIT**: a path where $u = v$.
- **TRAVERSES**: a path travels the vertices along it.
- **SIMPLE PATH**: a path that contains no edge more than once.
- **CONNECTED**: applies to undirected graphs. A graph is connected if and only if there is a path between every pair of distinct vertices in the graph.
- **CONNECTED COMPONENT**: a connected subgraph of the graph.
- **CUT VERTEX/EDGE**: separates one connected component into two if removed.
- **STRONGLY CONNECTED**: applies to directed graphs. A graph is strongly connected if there is a directed path from a to b for any two vertices a, b.
- **WEAKLY CONNECTED**: applies to directed graphs. A graph is weakly connected if the underlying undirected graph is connected. If a graph is strongly connected, it is also weakly connected.
- **COUNTING DIFFERENT PATHS**: Let r be the length of the path and A be the adjacency matrix of the graph. The number of different paths of length r from a vertex i to a vertex j is the entry (i, j) of A^r.
- **EULER CIRCUIT/TOUR/CYCLE**: a simple circuit containing every edge of the graph.
  - **THEOREM**: a connected multigraph has an Euler circuit if and only if each vertex has even degree.

  - **THEOREM**: In a Euler circuit in a directed graph, the indegrees must match the outdegrees.
  - **ALGORITHM**
    - Begin with any arbitrary node
    - Construct a simple path from this node until you get back to the start
    - Repeat for each remaining subgraph, splicing results back into the original cycle.
- **EULER PATH**: a simple path containing every edge of the path.
  - **THEOREM:** a connected multigraph has an Euler path if and only if it has exactly two vertices of odd degree.
- **HAMILTONIAN CIRCUIT**: a circuit that traverses each vertex in the graph exactly once.
  - **DIRAC'S THEOREM**: If G is connected, simple, and has $n \geq 3$ vertices, and for all v we have that $\deg(v) \geq \frac{n}{2}$, then G has a Hamiltonian circuit.
  - **ORE'S COROLLARY**: If G is connected, simple, and has $n \geq 3$ vertices, and $\deg(u) + \deg(v) \geq n$ for every pair u, v of non-adjacent vertices in G, then G has a Hamiltonian circuit.
- **HAMILTONIAN PATH**: a path that traverses each vertex in the graph exactly once.
  - **TRAVELING SALESMAN PROBLEM**: in a weighted graph, we can find the shortest tour visiting every vertex by finding the shortest Hamiltonian path in complete graphs
- **GRAY CODE**: a sequence of codewords such that each binary string is used, but adjacent codewords are close to each other (i.e. vary by only 1 bit). All binary strings are of length n, which represents the number of vertices in a hypercube Q_n The edges of the hypercube represent the vertices that only differ by 1 bit.
  - Our problem is to find a Hamiltonian circuit in hypercubes.
  - One solution: Take two of the same gray code sequences. To one, add a "0" to the front. With the other, reverse the sequence, and add a "1" to the front. Add the two sequences together. Now, you have another gray code sequence. It's defined recursively, just like hypercubes are.
- **PLANAR GRAPHS**: graphs that can be drawn in the plane without edges having to cross.
  - A graph is planar if it is isomorphic to a graph which is planar.
  - To prove that a graph is planar, we must redraw the edges in a way that no edges will cross.
  - The smallest graphs that are not planar: K_5, $K_{3,3}$
  - **MINOR**: H is minor of G if H can be obtained from G by a series of 0 or more deletions of vertices or edges, and contraction of edges.
  - A graphs is planar if and only if it does not contain K_5 and $K_{3,3}$ as a minor.
- **EULER'S THEOREM**: Let G be a connected plane graph with v vertices, e edges, and f faces. Then $v + f - e = 2$.
  - **COROLLARY**: If G is a connected planar graph with no parallel edges and no self-loops with $v \geq 3$, then $e \leq 3v - 6$.
    - Every face has at least three edges. Each edge is on two faces, or twice on the same face.
  - **COROLLARY**: Every planar graph with no parallel edges and no self-loops has a vertex of degree at most 5, because K_5 is not a planar graph.
  - **COROLLARY**: If G is a planar simple graph with v>=3 and no cycles of length 3, then $e \leq 2v - 4$, because $K_{3,3}$ is not planar.
  - **COROLLARY**: Every simple, planar graph has a vertex of degree less than 6.
- **VERTEX COLORING**: assigning a color to each vertex so that adjacent vertices are of different colors. In other words, finding a function $c: V \to N$ such that $(u, v) \in E \to c(u) \neq c(v)$.

- **CHROMATIC NUMBER** $\chi(G)$: the least amount of colors needed to color the graph.
  - The graph G is bipartite if and only if $\chi(G) = 2$.
  - **FOUR-COLOR THEOREM**: the chromatic number of every simple planar graph is at most four.
    - For general graphs, only exponential algorithms are known, and finding an approximation is difficult.

Logical Equivalencies

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

Rules of Inference

TABLE 1 Rules of Inference.

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

Quantifiers

TABLE 1 Quantifiers.

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

TABLE 2 Rules of Inference for Quantified Statements.	
Rule of Inference	***Name***
$\forall x\, P(x)$ $\therefore\ P(c)$	Universal instantiation
$P(c)$ for an arbitrary c $\therefore\ \forall x\, P(x)$	Universal generalization
$\exists x\, P(x)$ $\therefore\ P(c)$ for some element c	Existential instantiation
$P(c)$ for some element c $\therefore\ \exists x\, P(x)$	Existential generalization

Conditional Statements

TABLE 7 Logical Equivalences Involving Conditional Statements.

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

Biconditional Statements

TABLE 8 Logical Equivalences Involving Biconditional Statements.

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