asdf Cheat Sheet

by j24 via cheatography.com/195607/cs/41793/

Old

Countable. A set that is either finite or has the same cardinality as the set of positive integers (ℤ!) is called countable. To be countable, there must exist a 1-1 and onto (bijection) between the set and ℕ! (i.e. ℤ!)	Show that the set of positive even integers E is countable. Let f(x) = 2x, E = {1,2,3,4,...}, f(x) = {2,4,6,8,...}. Then f is a bijection from N to E since f is both one-to-one and onto. To show that it is one-to-one, suppose that f(n) = f(m).	All integers between 10 and 10000: Finite. All integers less than 10: Countably infinite. S = {(x,y) | x, y in N}: Countably inf. All real numbers between 0 and 1: Uncountable. All rational numbers between 0 and 1: Countably inf. All integers that are multiples of 8: Countably inf.

Old (cont)

Induction. To prove that P(n) is true for all positive integers n, we complete these steps: Basis Step: Show that P(1) is true. Inductive Step: Show that P(k) → P(k + 1) is true for all positive integers k. To complete the inductive step, assuming the inductive hypothesis that P(k) holds for an arbitrary integer k, show that P(k + 1) must be true.	Template for Proofs by Mathematical Induction 1. Express the statement that is to be proved in the form “for all n ≥ b, P (n)” for a fixed integer b. 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.” 4. State, and clearly identify, the inductive hypothesis, in the form “assume that P (k) is true for an arbitrary fixed integer k ≥ b.” 5. State what needs to be proved under the assumption that the inductive hypothesis is true. That is, write out what P (k + 1) says. 6. Prove the statement P (k + 1) making use the assumption P (k). Be sure that your proof is valid for all integers k with k ≥ b, taking care that the proof works for small values of k, including k = b. 7. Clearly identify the conclusion of the inductive step, such as by saying “this completes the inductive step.” 8. After completing the basis step and the inductive step, state the conclusion, namely that by mathematical induction, P (n) is true for all integers n with n ≥ b.

Old (cont)

Strong Induction: To prove that P(n) is true for all positive integers n, where P(n) is a propositional function, complete two steps: Basis Step: Verify that the proposition P(1) is true. Inductive Step: Show the conditional statement [P(1) ∧ P(2) ∧••• ∧ P(k)] → P(k + 1) holds for all positive integers k. Ordinary/weak induction • Rule 1: P(0) (or any other base case) • Rule 2: P(n) -> P(n+1) Strong induction • Rule 1: P(0) (or any other base case) • Rule 2: P(1),P(2), P(3),....P(n) -> P(n+1) The general	Example: Show that if n is an integer greater than 1, then n can be written as the product of primes. Solution: Let P(n) be the proposition that n can be written as a product of primes. BASIS STEP: P(2) is true since 2 itself is prime. INDUCTIVE STEP: The inductive hypothesis is P(j) is true for all integers j with 2 ≤ j ≤ k. To show that P(k + 1) must be true under this assumption, two cases need to be considered: If k + 1 is prime, then P(k + 1) is true. Otherwise, k + 1 is composite and can be written as the product of two positive integers a and b with 2 ≤ a ≤ b < k + 1. By the inductive	Example: Prove that every amount of postage of 12 cents or more can be formed using just 4-cent and 5-cent stamps. Solution: Let P(n) be the proposition that postage of n cents can be formed using 4-cent and 5-cent stamps. BASIS STEP: P(12), P(13), P(14), and P(15) hold. P(12) uses three 4-cent stamps. P(13) uses two 4-cent stamps and one 5-cent stamp. P(14) uses one 4-cent stamp and two 5-cent stamps. P(15) uses three 5-cent stamps. INDUCTIVE STEP: The inductive hypothesis states that P(j) holds for 12 ≤ j ≤ k, where k ≥ 15. Assuming the inductive hypothesis, it

Turing Machine

A Turing machine T = (S, I, f, s 0) consists of a finite set S of states, an alphabet I that includes the blank symbol B, a partial function f from (S × I)→ (S × I ×{R,L}) a starting state s0 . For some (state, symbol) pairs the partial function f may be undefined, but for a pair for which it is defined, there is a unique (state, symbol, direction) triple associated to this pair. The five-tuples corresponding to the partial function in the definition of a TM are called the transition rules of	1. At the beginning of its operation a TM is assumed to be in the initial state s0 and to be positioned over the leftmost nonblank symbol on the tape. This is the initial position of the machine. 2. At each step, the control unit reads the current tape symbol x. 3. If the control unit is in state s and if the partial function f is defined for the pair (s, x) with f(s, x) = (s′, x′, d), the control unit: enters the state s′, writes the symbol x′ in the current cell, erasing x, and moves right one cell if d = R or moves left one cell if d = L. 4. This step is written as the five-tuple (s, x, s′, x′, d). Turing machines are defined by specifying a set of such five-tuples. If the partial	Let V be a subset of an alphabet I. A TM T = (S, I, f, s0) recognizes a string x in V *if and only if T, starting in the initial position when x is written on the tape, halts in a final state. T is said to recognize a subset A of V* if it is the case that a string x is recognized by T if and only if x belongs to A. Note that to recognize a subset A of V* we can use symbols not in V. This means that the input alphabet I may include symbols not in V. We will see that these extra symbols are used as markers. A TM operating on a tape containing the symbols of a string x in consecutive cells,

Number Theory

Let a = b mod (m) • a is the remainder when b is divided by m Reflexive: $a \equiv a$ mod m Symmetric: If a ≡ ☐ ☐ mod ☐ ☐, then ☐ ☐ ≡ a mod m Transitivity: If a ≡ ☐ ☐ mod ☐ ☐ and b ≡ c mod m, then $a \equiv c$ mod m Additive inverse For any a, there exists a b such that $a + b = 0$ $(mod\ m)$ In this case, the b is called the additive inverse of a and vice versa Multiplicative inverse. For any ☐ ☐ relatively prime to ☐ ☐ where gcd a, m = 1, there exists a b such that ab = 1 (mod ☐	Base Conversions Convert 1011 0111 to decimal, octal, and hexadecimals Decimal • 2' ∗ 1 + 2(∗ 0 + 2) ∗ 1 + 2 ∗ *1 + 2! ∗ 0 + 2# ∗ 1 + (2" ∗ 1) + (2% ∗ 1)* • *183 10* • *183 =* *8*22+7 • 22 = 8*2+6 • 0 = 8*0+2 • 267. From Decimals to Binary, Octal and Hexadecimals Convert 24680 to Binary, Octal and Hexadecimals To convert to binary, divide by 2 repeatedly and record the remainder at each stage. 24680 = 2*12340 + 0 12340 =* *2*6170 + 0 6170 = 2*3085 + 0 3085 =* *2*1542 + 1 1542 = 2*771 + 0*	GCD and LCM Greatest Common Divisor (GCD) – is the largest number that divides both a and b Least Common Multiple (LCM) – Is the smallest positive integer that is divisible by a and b To find LCM Obviously, LCM(a,b) is no more than ab Start by finding the prime factors of a and b Build LCM using the largest power of each prime that is in a or b. Least Common Multiple Find lcm(40,12) • 40 = 2! 5" • 12 = 2# 3" For each prime base, use the largest exponent between the two numbers • 2! 3" 5" = 120 lcm(40,12)=120 Find lcm(5292,810) • 5292 = 2# 3! 7# • 810 = 2" 3 *5"* • *2# 3*5" 7# = 79380 lcm(5292,810)=79380. GCD as a linear combination If

rule. 1. If P(n+1) can be proven from P(n) only, then weak/o-rdinary induction is sufficient 2. If P(n+1) requires other propos-itions prior to P(n) (e.g. P(n-1) or P(n-2)) then strong induction may be appropriate

hypothesis a and b can be written as the product of primes and therefore k + 1 can also be written as the product of those primes. Hence, it has been shown that every integer greater than 1 can be written as the product of primes.

can be shown that P(k + 1) holds. Using the inductive hypothesis, P(k − 3) holds since k − 3 ≥ 12. To form postage of k + 1 cents, add a 4-cent stamp to the postage for k − 3 cents. Hence, P(n) holds for all n ≥ 12.

the machine.

function f is undefined for the pair (s, x) then T will halt. 5. The Turing Machine outputs the revised tape.

does not recognize x if it does not halt or halts in a state that is not final.

☐) In this case, the b is called the multip-licative inverse of a and vice versa. *a* ≡ *b* ☐ ☐*od m* is equivalent to *a* − ☐ ☐ = *kn* for some ☐ ☐ ∈ ℤ If ☐ ☐ ≡ *b* (*mod n*) and c ≡ ☐ ☐ (*mod* ☐ ☐), then ☐ ☐*c* ≡ *bd* (*mod n*) Example. 5 ~ 3 (mod 2) Congruent Class. The congruent class of an integer a, denoted [a] is defined as [a] = { b in Z | a is congruent to b}

*771 = 2*385 + 1 385 = 2*192 + 1 192 = 2*96 + 0 96 = 2*48 + 0 48 = 2*24 + 0 24 = 2*12 + 0 12 = 2*6 + 0 6 = 2*3 + 0 3 = 2*1 + 1 1 = 2*0 + 1 110000-001101-000_2. From Decimals to Binary, Octal and Hexade-cimals Convert 24680 to Binary, Octal and Hexade-cimals 24680 = 8*3085 + 0 3085 = 8*385 + 5 385 = 8*48 + 1 48 = 8*6 + 0 6 = 8*0+6 60150 8 24680 = 16*1542+8 1542 = 16*96+6 96=16*6+0 6=16*0+6 6068_16

a and b are positive integers, the gcd(a, b) can be written as gcd(a, b) = am + bn for some integers m and n. Note. Multiples of GCD are Linear Combinations of a and b E.g. write gcd(312, 125) as a linear combin-ation 312 m + 125 n Solution. gcd(312, 125) = gcd(312, 62) è 312 = 2*125 + 62 ---(1) gcd(312, 62) = GCD(62, 1) è 125 = 2*62 + 1 ---(2) gcd(62,1) = 1 Using (2). 1 = 125 + (-2)*62 = 125 + (-2) (312- 2*125) using (1) = 5*125 + (-2)*312.

FSM FSA NFA

A finite-state machine M =(S, I, O, f, g, s 0) consists of a finite set S of states a finite input alphabet I a finite output alphabet O a transition function f that assigns to each state and input pair a new state an output function g that assigns to each state and input pair an output an initial state s 0 . A state table is used to represent the values of the transition function f and the output function g for all (state, input). FSMs with no output, but with some states designated as accepting states, are specifically designed for recognizing languages. A finite-state automaton M = (S, I, f, s0, F) consists of a finite set S of states, a finite input alphabet I, a transition function f that assigns a next state to every pair of state and input (so that f: S × I → S), an initial or start state s 0, and a subset F of S consisting of final (or accepting) states. FSAs can be represented using either state tables or state diagrams, in which final states are indicated with a double circle. A finite state machine (FSM) with no output is called a finite state automata (FSA). A string x is said to be recognized (or accepted) by the machine M = (S, I, f, s 0, F) if it takes the A nondeterministic finite-state automaton M = (S, I, f, s 0, F) consists of A finite set S of states A finite input alphabet I A transition function f that assigns a set of states to every pair of state and input (so that f: S × I → P(S)) An initial or start state s0 A subset F of S consisting of final (or accepting) states. For every NFA there is an equivalent DFA. That is, if the language L is recognized by a NFA M 0, then L is also recognized by a DFA M 1. We construct the DFA M 1 so that The start symbol of M 1 is {s 0}. The input set of M 1 is the same as the input set of M 0. Each state in M 1 is made from of a set of

FSM FSA NFA (cont)

A vocabulary (or alphabet) V is a finite, nonempty set of elements called symbols. A word (or sentence) over V is a string of finite length of elements of V . The empty string or null string, denoted by λ, is the string containing no symbols. The set of all words over V is denoted by V ∗. A language over V is a subset of V ∗. A phrase-structure grammar G = (V , T , S, P) consists of a vocabulary V , a subset T of V consisting of terminal symbols, a start symbol S from V , and a finite set of productions P . The set V − T is denoted by N. Elements of N are called nonterminal symbols. Every production in P must contain at least one nonterminal on its left side. Let G = (V , T , S, P) be a phrase-structure grammar. The language generated by G (or the language of G), denoted by L(G), is the set of all strings of terminals that are derivable from the starting state S. In other words, L(G) = {w ∈ T ∗ | S ∗ *⇒ w}. EXAMPLE 5 Give a phrase-structure grammar that generates the set {0n1n | n = 0, 1, 2, . . . }. The solution is the grammar G = (V , T , A type 0 grammar has no restrictions on its productions. A type 1 grammar can have productions of the form w1 → w2 , where w1 = lAr and w2 = lwr, where A is a nonterminal symbol, l and r are strings of zero or more terminal or nonterminal symbols, and w is a nonempty string of terminal or nonterminal symbols. It can also have the production S → λ as long as S does not appear on the right-hand side of any other production. A type 2 grammar can have productions only of the form w1 → w2 , where w1 is a single symbol that is not a terminal symbol. A type 3 grammar can have productions only of the form w1 → w2 with w1 = A and either w2 = aB or w2 = a, where A and B are nonterminal symbols and a is a terminal symbol, or with w1 = S and w2 = λ. EXAMPLE 9 It follows from Example 5 that {0n1n | n = 0, 1, 2, . . . } is a context-

Relations

A binary relation R on a A and B is defined as R is a subset of A x B. A relation is a subset of the cartesian product of two sets A and B, which is a set of ordered pairs. A x B = {(a,1), (a,2), (a,3), (b,1), (b,2), (b,3), (c,1), (c,2), (c,3), (d,1), (d,2), (d,3)}. A relation is usually written in set format: R = {(a,2), (b,1), (c,1), (d,3), (c,2)}. We say that a is related to 2 in one of the following notations: (a,2) is e R, or a R 2. 1) A relation R on a set A is called reflexive if (a, a) ∈ R for every element a ∈ A. 2) A relation R on a set A is called symmetric if (b, a) ∈ R whenever (a, b) ∈ R, for all a, b ∈ A. A relation R on a set A such that for all a, b ∈ A, if (a, b) ∈ R and (b, a) ∈ R, then a = b is called antisymmetric. 3) A relation R on a set A is called transitive if whenever (a, b) ∈ R and (b, c) ∈ R, then (a, c) ∈ R, for all a, b, c ∈ A. 4) Let R be a relation from a set A to a set B and S a relation from B to Because this relation contains R, is reflexive, and is contained within every reflexive relation that contains R, it is called the reflexive closure of R. This new relation is symmetric and contains R. Furthermore, any symmetric relation that contains R must contain this new relation, because a symmetric relation that contains R must contain (2, 1) and (1, 3). Consequently, this new relation is called the symmetric closure of R. Let R be a relation on a set A. The connectivity relation R∗ consists of the pairs (a, b) such that there is a path of length at least one from a to b in R. The transitive closure of a relation R equals the connectivity relation R∗. A relation on a set A is called an equivalence relation if it is reflexive, symmetric, and transitive. Two elements a and b that are related

Alternatively, a finite-state machine can be represented by a state diagram, which is a directed graph with labeled edges. Each state is represented by a circle, and arrows labeled with the input and output pair represent the transitions. The state table and state diagram both represent the finite state machine with S = {s 0 ,s 1 ,s 2 ,s 3 }, I = {0, 1}, and O = {0, 1}.

initial state s 0 to a final state, that is, f(s 0, x). The language recognized (or accepted) by M, denoted by L(M), is the set of all strings that are recognized by M. Two finite-state automata are called equivalent if they recognize the same language. The final state of M 3 are s 0 and s 3 . The strings that take s 0 to itself are λ, 0, 00, 000,... . The strings that take s 0 to s 3 are a string of zero or more consecutive 0s, followed by 10, followed by any string. Hence, L(M 3) = {0n ,0n 10x | n = 0, 1, 2, ..., and x is any string}

states in M 0. Construct new states in M 1 by interpreting each unique output in the M 0 transition table as a its singular own state, e.g. s ! , 𝑠" , ☐ ☐# , ∅ Given a state {𝑠$! , 𝑠$ " ,..., 𝑠$ # } in M 1 and an input symbol x, the transitions from this state to the next is the union of transitions f(𝑠$! , x), f(𝑠$ " ,x), ... , f(𝑠$ # ,x) from M 0 for the states that compose the state from ☐ ☐" The final states of M 1 are any sets that contain a final state of M 0.

S, P), where V = {0, 1, S}, T = {0, 1}, S is the starting symbol, and the productions are S → 0S1 S → λ.

free language, because the productions in this grammar are S → 0S1 and S → λ.

a set C. The composite of R and S is the relation consisting of ordered pairs (a, c), where a ∊ A, c ∊ C, and for which there exists an element b ∊ B such that (a, b) ∊ R and (b, c) ∊ S. We denote the composite of R and S by S ∘R. 5) Let R be a relation on the set A. The powers R n , n = 1, 2, 3, . . . , are defined recursively by R1 = R and R n+1 = R n ∘ R. 6) The relation R on a set A is transitive if and only if R n ⊆ R for n = 1, 2, 3, ...

by an equivalence relation are called equivalent. The notation a ~ b is often used to denote that a and b are equivalent elements with respect to a particular equivalence relation. Let R be an equivalence relation on a set A. The set of all elements that are related to an element a of A is called the equivalence class of a. The equivalence class of a with respect to R is denoted by [a]R . When only one relation is under consideration, we can delete the subscript R and write [a] for this equivalence class. Let R be an equivalence relation on a set A. These statements for elements a and b of A are equivalent: (i) aRb (ii) [a] = [b] (iii) [a] ∩ [b] =/= ∅. A relation R on a set S is called a partial ordering or partial order if it is reflexive, antisym- metric, and transitive. A set S together with a partial ordering R is called a partially ordered set, or poset, and is denoted by (S, R). Members of S are called elements of the poset. When every two elements in the set are comparable, the relation is called a total ordering.

Boolean functions

The complement of an element, denoted with a bar, is defined by bar0 = 1 and bar1 = 0. The variable x is called a Boolean variable if it assumes values only from B, that is, if its only possible values are 0 and 1. A function from B n to B is called a Boolean function of degree n. x | y (or x NAND y): the expression that has the value 0 when both x and y have the value 1 and the value 1 otherwise. x ↓ y (or x NOR y): the expression that has the value 0 when either x or y or both have the value 1 and the value 0 other- wise	barbarx = x Law of the double complement x + x = x Idempotent laws x · x = x x + 0 = x Identity laws x · 1 = x x + 1 = 1 Domination laws x · 0 = 0 x + y = y + x Commut-ative laws xy = yx x + (y + z) = (x + y) + z Associative laws x(yz) = (xy)z x + yz = (x + y)(x + z) Distri-butive laws x(y + z) = xy + xz bar(xy) = barx + bary De Morgan’s laws bar(x + y) = barx bary. x + xy = x Absorption laws x(x + y) = x x + barx = 1 Unit property xbarx = 0 Zero property	A literal is a Boolean variable or its complement. A minterm of the Boolean variables x1, x2, . . . , xn is a Boolean product y1y2 · · · yn, where y i = xi or yi = x i . Hence, a minterm is a product of n literals, with one literal for each variable. The sum of minterms that represents the function is called the sum-of-pr-oducts expansion or the disjun-ctive normal form of the Boolean function. Find the sum-of-products expansion for the function F (x, y, z) = (x + y)z. Solution: We will find the sum-of-products expansion of F (x, y, z) in two ways. First, we will use Boolean identities to expand the product and simplify. We find that F (x, y, z) = (x +

y)z = xz + yz Distributive law = x1z + 1yz Identity law = x(y + y)z + (x + x)yz Unit property = xyz + xy z + xyz + xyz Distributive law = xyz + xy z + xy z. Idempotent law. The resulting expansion is called the conjunctive normal form or product-of-sums expansion of the function. These expansions can be found from sum-of-products expansions by taking duals. Because every Boolean function can be represented using these operators we say that the set {·, +,− } is functionally complete